Computational Mathematics
Lecture 0: Introduction to Algorithms

Patrick E. Farrell

University of Oxford

For some mathematical problems, we can just write down the solution:

$$\text{For } a_0, a_1 \in \mathbb{R}, a_1 \neq 0, \text{ find } x \in \mathbb{R} \text{ such that } a_1 x + a_0 = 0.$$

For some mathematical problems, we can just write down the solution:

$$\text{For } a_0, a_1 \in \mathbb{R}, a_1 \neq 0, \text{ find } x \in \mathbb{R} \text{ such that } a_1 x + a_0 = 0.$$

This has solution $x = -a_0/a_1$.

For some mathematical problems, we can just write down the solution:

$$\text{For } a_0, a_1 \in \mathbb{R}, a_1 \neq 0, \text{ find } x \in \mathbb{R} \text{ such that } a_1 x + a_0 = 0.$$

This has solution $x = -a_0/a_1$.

For most problems, we can't just write down the solution:

$$\text{For } a_0, \ldots, a_5 \in \mathbb{R}, \text{ find } x \in \mathbb{C} \text{ such that } a_5 x^5 + a_4 x^4 + \cdots + a_0 = 0.$$

For some mathematical problems, we can just write down the solution:

$$\text{For } a_0, a_1 \in \mathbb{R}, a_1 \neq 0, \text{ find } x \in \mathbb{R} \text{ such that } a_1 x + a_0 = 0.$$

This has solution $x = -a_0/a_1$.

For most problems, we can't just write down the solution:

$$\text{For } a_0, \ldots, a_5 \in \mathbb{R}, \text{ find } x \in \mathbb{C} \text{ such that } a_5 x^5 + a_4 x^4 + \cdots + a_0 = 0.$$

Theorem (Abel, 1824)

There are polynomials of degree 5 and higher that cannot be solved by radicals (addition, subtraction, multiplication, division, and nth root extraction).

Niels Henrik Abel, 1802–1829

So what do we do in this situation? We still care about the roots of polynomials!

So what do we do in this situation? We still care about the roots of polynomials!

Response A: prove things *about* the solutions.

We could prove that if x is a root of a polynomial with real coefficients, so is $\bar{x}$. Or we could study Vieta's formulae, that (for example) the product of the roots of an n-th degree polynomial is $(-1)^n a_0/a_n$.

So what do we do in this situation? We still care about the roots of polynomials!

Response A: prove things *about* the solutions.

We could prove that if x is a root of a polynomial with real coefficients, so is $\bar{x}$. Or we could study Vieta's formulae, that (for example) the product of the roots of an n-th degree polynomial is $(-1)^n a_0 / a_n$.

Response B: devise *algorithms* for computing the solutions.

Develop a computational procedure that approximates to arbitrary accuracy the roots of our polynomial: *compute* a sequence that converges to the roots.

The central topic of computational mathematics is algorithms.

Definition (Algorithm, informal)

An algorithm is a finite set of instructions for solving a mathematical problem. It associates to each input a sequence of elementary computational steps to calculate some desired output.

The formalisation of this definition is studied in computer science, e.g. with *Turing machines*.

Muḥammad ibn Mūsā al-Khwārizmī, c. 780–850

Algorithms are of core interest to both pure and applied mathematics.

Algorithms are of core interest to both pure and applied mathematics.

In pure mathematics, we use algorithms to (among other things) prove the existence of some object. We will see examples in the course.

Algorithms are of core interest to both pure and applied mathematics.

In pure mathematics, we use algorithms to (among other things) prove the existence of some object. We will see examples in the course.

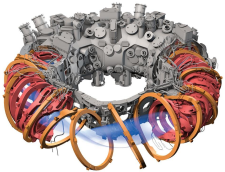
You will see another example in Part A Differential Equations: you will prove that under certain conditions a unique solution exists to the problem

$$\text{find } y(t) \text{ such that } \frac{\mathrm{d}y}{\mathrm{d}t} = f(y,t), \quad y(0) = y_0,$$

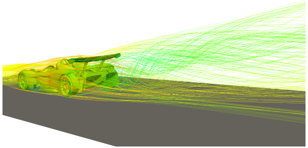
by constructing a sequence of approximations y_n that converges $y_n \to y$.

In applied mathematics, algorithms are used to solve problems arising in science and engineering.

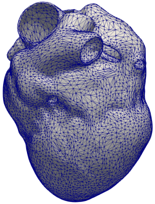
climate



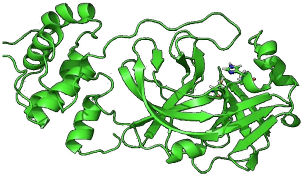
energy



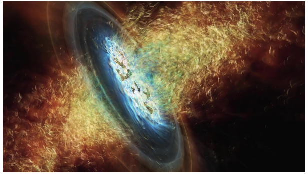
aerodynamics



physiology



covid



galaxies

Questions we ask:

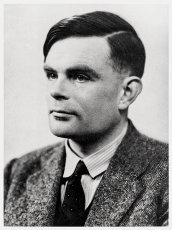
Does our algorithm terminate?

Questions we ask:

Does our algorithm terminate?

Theorem (Halting problem, 1936)

No algorithm exists that always correctly decides if another algorithm terminates on a given input.



Alan Turing, 1912–1954

Questions we ask:

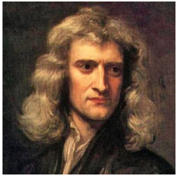
Does our algorithm give the correct answer, and if so, when?

Questions we ask:

Does our algorithm give the correct answer, and if so, when?

In later lectures we will see Newton's method for finding a solution x of a general rootfinding problem $f(x) = 0$.

This converges if we start the iteration close to x, but diverges if we start far away.



Isaac Newton, 1643–1727

Questions we ask:

How fast does the algorithm converge to the right answer?

Questions we ask:

How fast does the algorithm converge to the right answer?

Consider two formulae for π:

$$\pi = 4\sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1}, \quad \pi^{-1} = \frac{2\sqrt{2}}{99^2}\sum_{k=0}^{\infty} \frac{(4k)!}{k!^4}\frac{26390k+1103}{396^{4k}}.$$

If we approximate the series by its partial sums, how many terms do we require for accuracy to ten digits?

Gottfried Leibniz, 1646–1716

Srinivasa Ramanujan, 1887–1920

Questions we ask:

How fast does the algorithm converge to the right answer?

Consider two formulae for π:

$$\pi = 4\sum_{k=0}^{\infty}\frac{(-1)^k}{2k+1}, \quad \pi^{-1} = \frac{2\sqrt{2}}{99^2}\sum_{k=0}^{\infty}\frac{(4k)!}{k!^4}\frac{26390k+1103}{396^{4k}}.$$

If we approximate the series by its partial sums, how many terms do we require for accuracy to ten digits?

About 5 billion, vs 2!

Gottfried Leibniz, 1646–1716

Srinivasa Ramanujan, 1887–1920

Questions we ask:

How many operations does the algorithm take?

Questions we ask:

How many operations does the algorithm take?

There are many algorithms for sorting a list of n numbers.

The number of comparisons required by a naïve algorithm called *bubble* sort scales like n^2, while the *merge* sort of von Neumann in 1945 scales like $n \log n$. This is much, much faster for large n.

John von Neumann, 1903–1957

Questions we ask:

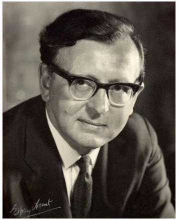
Are the problem and algorithm stable to small perturbations in data?

Questions we ask:

Are the problem and algorithm stable to small perturbations in data?

Consider $p(x) = (x-1)(x-2)\cdots(x-20)$. Expanding in monomials, we have

$$p(x) = x^{20} - 210x^{19} + 20615x^{18} + \cdots + 20!.$$



James H. Wilkinson, 1919–1986

Questions we ask:

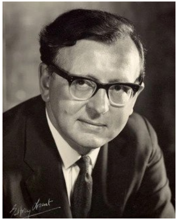
Are the problem and algorithm stable to small perturbations in data?

Consider $p(x) = (x-1)(x-2)\cdots(x-20)$. Expanding in monomials, we have

$$p(x) = x^{20} - 210x^{19} + 20615x^{18} + \cdots + 20!.$$

If we perturb -210 to -210.0000001192, then the roots become (to 5 digits)

1.00000 2.00000 3.00000 4.00000 5.00000



James H. Wilkinson, 1919–1986

Questions we ask:

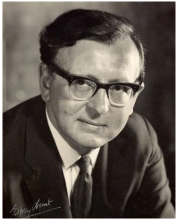
Are the problem and algorithm stable to small perturbations in data?

Consider $p(x) = (x-1)(x-2)\cdots(x-20)$. Expanding in monomials, we have

$$p(x) = x^{20} - 210x^{19} + 20615x^{18} + \cdots + 20!.$$

If we perturb -210 to -210.0000001192, then the roots become (to 5 digits)

1.00000	2.00000	3.00000	4.00000	5.00000
6.00001	6.99970	8.00727	8.91725	20.84691



James H. Wilkinson, 1919–1986

Questions we ask:

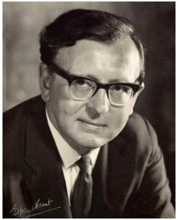
Are the problem and algorithm stable to small perturbations in data?

Consider $p(x) = (x-1)(x-2)\cdots(x-20)$. Expanding in monomials, we have

$$p(x) = x^{20} - 210x^{19} + 20615x^{18} + \cdots + 20!.$$

If we perturb -210 to -210.0000001192, then the roots become (to 5 digits)

1.00000	2.00000	3.00000	4.00000	5.00000
6.00001	6.99970	8.00727	8.91725	20.84691
$10.09527 \pm 0.64350i$	$11.79363 \pm 1.65233i$	$13.99236 \pm 2.51883i$	$16.73074 \pm 2.81262i$	$19.50244 \pm 1.94033i$



James H. Wilkinson, 1919–1986

Computational Mathematics
Week 1: Euclid's algorithm

Patrick E. Farrell

University of Oxford

We start with the natural numbers

$$\mathbb{N} = \{0, 1, 2, 3, \dots\},$$

and consider dividing one natural number t by another $b \neq 0$:

$$t = qb + r, \quad 0 \leq r < b.$$

We start with the natural numbers

$$\mathbb{N} = \{0, 1, 2, 3, \dots\},$$

and consider dividing one natural number t by another $b \neq 0$:

$$t = qb + r, \quad 0 \leq r < b.$$

Either b divides t evenly $(r = 0)$, or we are allowed to divide b by r. If $r \neq 0$ we can *iterate*: we can see what the division of b by r is.

We start with the natural numbers

$$\mathbb{N} = \{0, 1, 2, 3, \dots\},$$

and consider dividing one natural number t by another $b \neq 0$:

$$t = qb + r, \quad 0 \leq r < b.$$

Either b divides t evenly ($r = 0$), or we are allowed to divide b by r. If $r \neq 0$ we can *iterate*: we can see what the division of b by r is.

Take $t = 80, b = 7$:

$$80 = 11 \times 7 + 3 \quad (q = 11, r = 3)$$

We start with the natural numbers

$$\mathbb{N} = \{0, 1, 2, 3, \dots\},$$

and consider dividing one natural number t by another $b \neq 0$:

$$t = qb + r, \quad 0 \leq r < b.$$

Either b divides t evenly ($r = 0$), or we are allowed to divide b by r. If $r \neq 0$ we can *iterate*: we can see what the division of b by r is.

Take $t = 80, b = 7$:

$$\begin{aligned} 80 &= 11 \times 7 + 3 \quad (q = 11, r = 3) \\ 7 &= 2 \times 3 + 1 \quad (q = 2, r = 1) \end{aligned}$$

We start with the natural numbers

$$\mathbb{N} = \{0, 1, 2, 3, \dots\},$$

and consider dividing one natural number t by another $b \neq 0$:

$$t = qb + r, \quad 0 \leq r < b.$$

Either b divides t evenly ($r = 0$), or we are allowed to divide b by r. If $r \neq 0$ we can *iterate*: we can see what the division of b by r is.

Take $t = 80, b = 7$:

$$\begin{aligned} 80 &= 11 \times 7 + 3 \quad (q = 11, r = 3) \\ 7 &= 2 \times 3 + 1 \quad (q = 2, r = 1) \end{aligned}$$

We start with the natural numbers

$$\mathbb{N} = \{0, 1, 2, 3, \dots\},$$

and consider dividing one natural number t by another $b \neq 0$:

$$t = qb + r, \quad 0 \leq r < b.$$

Either b divides t evenly ($r = 0$), or we are allowed to divide b by r. If $r \neq 0$ we can *iterate*: we can see what the division of b by r is.

Take $t = 80, b = 7$:

$$\begin{aligned} 80 &= 11 \times 7 + 3 \quad (q = 11, r = 3) \\ 7 &= 2 \times 3 + 1 \quad (q = 2, r = 1) \\ 3 &= 3 \times 1 + 0 \quad (q = 3, r = 0). \end{aligned}$$

We start with the natural numbers

$$\mathbb{N} = \{0, 1, 2, 3, \dots\},$$

and consider dividing one natural number t by another $b \neq 0$:

$$t = qb + r, \quad 0 \leq r < b.$$

Either b divides t evenly ($r = 0$), or we are allowed to divide b by r. If $r \neq 0$ we can *iterate*: we can see what the division of b by r is.

Take $t = 80, b = 7$:

$$\begin{aligned} 80 &= 11 \times 7 + 3 \quad (q = 11, r = 3) \\ 7 &= 2 \times 3 + 1 \quad (q = 2, r = 1) \\ 3 &= 3 \times 1 + 0 \quad (q = 3, r = 0). \end{aligned}$$

The game ends when $r = 0$.

We start with the natural numbers

$$\mathbb{N} = \{0, 1, 2, 3, \dots\},$$

and consider dividing one natural number t by another $b \neq 0$:

$$t = qb + r, \quad 0 \leq r < b.$$

Either b divides t evenly ($r = 0$), or we are allowed to divide b by r. If $r \neq 0$ we can *iterate*: we can see what the division of b by r is.

Take $t = 80, b = 7$:

$$\begin{aligned} 80 &= 11 \times 7 + 3 \quad (q = 11, r = 3) \\ 7 &= 2 \times 3 + 1 \quad (q = 2, r = 1) \\ 3 &= 3 \times 1 + 0 \quad (q = 3, r = 0). \end{aligned}$$

The game ends when $r = 0$. We're interested in the last remainder before hitting 0. This is the greatest common divisor of the two inputs!

Here is the *algorithm*. It computes the *greatest common divisor* (also called *highest common factor*) of two numbers.

---

function gcd(t, b)
    $r \leftarrow t \bmod b$
    while $r \neq 0$ **do**
        $t \leftarrow b$
        $b \leftarrow r$
        $r \leftarrow t \bmod b$
    end while
    return b
end function

Here is the *algorithm*. It computes the *greatest common divisor* (also called *highest common factor*) of two numbers.

---

function gcd(t, b)
    $r \leftarrow t \bmod b$
    while $r \neq 0$ **do**
        $t \leftarrow b$
        $b \leftarrow r$
        $r \leftarrow t \bmod b$
    end while
    return b
end function

This completely and unambiguously lists the steps for a computer to take.

Here is the *algorithm*. It computes the *greatest common divisor* (also called *highest common factor*) of two numbers.

---

function gcd(t, b)
  $r \leftarrow t \bmod b$
  while $r \neq 0$ **do**
    $t \leftarrow b$
    $b \leftarrow r$
    $r \leftarrow t \bmod b$
  end while
  return b
end function

This completely and unambiguously lists the steps for a computer to take.

Note that this algorithm calls another one (the division algorithm).

Theorem (Elements, book VII, c. 300 BCE)

Given any $t, b \in \mathbb{N}$, $0 < b < t$, Euclid's algorithm computes the greatest common divisor of t and b.

Euclid of Alexandria, c. 300 BCE

Theorem (Elements, book VII, c. 300 BCE)

Given any $t, b \in \mathbb{N}$, $0 < b < t$, Euclid's algorithm computes the greatest common divisor of t and b.

For convenience, let's label each intermediate value:

$$
\begin{aligned}
t &= q_0 b + r_0 \\
b &= q_1 r_0 + r_1 \\
r_0 &= q_2 r_1 + r_2 \\
&\vdots \\
r_j &= q_{j+2} r_{j+1} + r_{j+2} \\
&\vdots
\end{aligned}
$$

Euclid of Alexandria, c. 300 BCE

Theorem (Elements, book VII, c. 300 BCE)

Given any $t, b \in \mathbb{N}$, $0 < b < t$, Euclid's algorithm computes the greatest common divisor of t and b.

For convenience, let's label each intermediate value:

$$\begin{aligned} t &= q_0 b + r_0 \\ b &= q_1 r_0 + r_1 \\ r_0 &= q_2 r_1 + r_2 \\ &\vdots \\ r_j &= q_{j+2} r_{j+1} + r_{j+2} \\ &\vdots \end{aligned}$$

Also for convenience, denote

$$r_{-2} \coloneqq t, \quad r_{-1} \coloneqq b.$$

Euclid of Alexandria, c. 300 BCE

Claim: the algorithm terminates.

Since division yields $r < b$, the sequence of remainders $(r_{-2}, r_{-1}, r_0, \dots)$ is a strictly decreasing sequence of natural numbers. The sequence must therefore eventually reach zero. The algorithm therefore always terminates.

Claim: the algorithm terminates.

Since division yields $r < b$, the sequence of remainders $(r_{-2}, r_{-1}, r_0, \dots)$ is a strictly decreasing sequence of natural numbers. The sequence must therefore eventually reach zero. The algorithm therefore always terminates.

Let i be the index such that $r_i = 0$.

Claim: r_{i-1} divides r_j, $j < i - 1$ (common divisor).

Since $r_i = 0$, r_{i-1} divides r_{i-2}, i.e.

$$r_{i-2} = q_i r_{i-1}.$$

Claim: r_{i-1} divides r_j, $j < i - 1$ (common divisor).

Since $r_i = 0$, r_{i-1} divides r_{i-2}, i.e.

$$r_{i-2} = q_i r_{i-1}.$$

Plugging this into the previous iteration tells us that r_{i-1} also divides r_{i-3}:

$$\begin{aligned} r_{i-3} &= q_{i-1} r_{i-2} + r_{i-1} \\ &= (\cdots) \times r_{i-1} \end{aligned}$$

Claim: r_{i-1} divides r_j, $j < i - 1$ (common divisor).

Since $r_i = 0$, r_{i-1} divides r_{i-2}, i.e.

$$r_{i-2} = q_i r_{i-1}.$$

Plugging this into the previous iteration tells us that r_{i-1} also divides r_{i-3}:

$$\begin{aligned} r_{i-3} &= q_{i-1} r_{i-2} + r_{i-1} \\ &= (\cdots) \times r_{i-1} \end{aligned}$$

Proceeding by induction shows that r_{i-1} divides all remainders in the sequence. In particular, r_{i-1} is a common divisor of the original t and b.

Claim: r_{i-1} is the greatest common divisor.

Assume $d \in \mathbb{N}$ also divides t and b, so there exist $\alpha, \beta \in \mathbb{N}$ such that

$$t = \alpha d, \quad b = \beta d.$$

Claim: r_{i-1} is the greatest common divisor.

Assume $d \in \mathbb{N}$ also divides t and b, so there exist $\alpha, \beta \in \mathbb{N}$ such that

$$t = \alpha d, \quad b = \beta d.$$

Since $t = q_0 b + r_0$, we get $r_0 = (\alpha - q_0 \beta) d$, so d divides r_0.

Claim: r_{i-1} is the greatest common divisor.

Assume $d \in \mathbb{N}$ also divides t and b, so there exist $\alpha, \beta \in \mathbb{N}$ such that

$$t = \alpha d, \quad b = \beta d.$$

Since $t = q_0 b + r_0$, we get $r_0 = (\alpha - q_0 \beta) d$, so d divides r_0.

The next equation is $b = q_1 r_0 + r_1$, but since d divides both b and r_0, it must also divide r_1. Proceeding by induction, d must divide *all* remainders, including r_{i-1}.

Claim: r_{i-1} is the greatest common divisor.

Assume $d \in \mathbb{N}$ also divides t and b, so there exist $\alpha, \beta \in \mathbb{N}$ such that

$$t = \alpha d, \quad b = \beta d.$$

Since $t = q_0 b + r_0$, we get $r_0 = (\alpha - q_0 \beta) d$, so d divides r_0.

The next equation is $b = q_1 r_0 + r_1$, but since d divides both b and r_0, it must also divide r_1. Proceeding by induction, d must divide *all* remainders, including r_{i-1}.

Thus $d \leq r_{i-1}$, and r_{i-1} is the greatest common divisor of t and b.

When we know an algorithm works, our next question is usually: how fast is it? How many operations does it take, as a function of the inputs? This is referred to as its *complexity*.

When we know an algorithm works, our next question is usually: how fast is it? How many operations does it take, as a function of the inputs? This is referred to as its *complexity*.

In this context, we ask: can we bound the number of divisions required in computing $\gcd(t, b)$ in terms of t and b, $t > b > 0$?

When we know an algorithm works, our next question is usually: how fast is it? How many operations does it take, as a function of the inputs? This is referred to as its *complexity*.

In this context, we ask: can we bound the number of divisions required in computing $\gcd(t, b)$ in terms of t and b, $t > b > 0$?

Since the remainder decreases at each iteration, we know at least that we will do at most b iterations, i.e. the cost grows linearly in the size of the inputs.

When we know an algorithm works, our next question is usually: how fast is it? How many operations does it take, as a function of the inputs? This is referred to as its *complexity*.

In this context, we ask: can we bound the number of divisions required in computing $\gcd(t, b)$ in terms of t and b, $t > b > 0$?

Since the remainder decreases at each iteration, we know at least that we will do at most b iterations, i.e. the cost grows linearly in the size of the inputs.

But it is possible to prove a tighter bound!

Theorem

Let $t > b > 0$. The smallest values of t and b for which Euclid's algorithm requires N iterations are the Fibonacci numbers $t = F_{N+2}$ and $b = F_{N+1}$.

Theorem (Complexity of Euclid's algorithm, 1844)

The number of steps taken in Euclid's algorithm can never be more than five times the number of decimal digits of b.

Gabriel Lamé, 1795–1870

Theorem

Let $t > b > 0$. The smallest values of t and b for which Euclid's algorithm requires N iterations are the Fibonacci numbers $t = F_{N+2}$ and $b = F_{N+1}$.

Theorem (Complexity of Euclid's algorithm, 1844)

The number of steps taken in Euclid's algorithm can never be more than five times the number of decimal digits of b.

Gabriel Lamé, 1795–1870

This result shows that the cost grows *logarithmically* in the size of the input b.


PELOUZE
CARNOT
LAME

Section 2

Diophantine equations

A *Diophantine* equation is an algebraic equation for which solutions are sought in the integers $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$. They are named after Diophantus of Alexandria (c. 200–290).

A *Diophantine* equation is an algebraic equation for which solutions are sought in the integers $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$. They are named after Diophantus of Alexandria (c. 200–290).

Some Diophantine equations have no solutions, like $4x + 6y = 3$.

A *Diophantine* equation is an algebraic equation for which solutions are sought in the integers $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$. They are named after Diophantus of Alexandria (c. 200–290).

Some Diophantine equations have no solutions, like $4x + 6y = 3$.

Some do, however. For example, $48x - 35y = 1$ has a solution $x = -8, y = -11$.

A *Diophantine* equation is an algebraic equation for which solutions are sought in the integers $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$. They are named after Diophantus of Alexandria (c. 200–290).

Some Diophantine equations have no solutions, like $4x + 6y = 3$.

Some do, however. For example, $48x - 35y = 1$ has a solution $x = -8, y = -11$.

Diophantus' work was collected in his magnum opus, *Arithmetica*. In 1637, Pierre de Fermat wrote in the margin of his copy of *Arithmetica*,

> *It is impossible …for any number which is a power greater than the second to be written as the sum of two like powers. I have a truly marvelous demonstration of this proposition which this margin is too narrow to contain.*

Pierre de Fermat, 1607–1665

A linear Diophantine equation (LDE) in two variables is of the form: given $a, b, c \in \mathbb{Z}$, find $x, y \in \mathbb{Z}$ such that

$$ax + by = c.$$

A linear Diophantine equation (LDE) in two variables is of the form: given $a, b, c \in \mathbb{Z}$, find $x, y \in \mathbb{Z}$ such that

$$ax + by = c.$$

LDEs with $\gcd(a, b) = 1 = c$ are of particular interest. If we can solve

$$ax + by = 1$$

then we have solved the problem: find $x \in \mathbb{Z}$ such that

$$ax \equiv 1 \pmod{b},$$

the problem of finding modular multiplicative inverses.

A linear Diophantine equation (LDE) in two variables is of the form: given $a, b, c \in \mathbb{Z}$, find $x, y \in \mathbb{Z}$ such that

$$ax + by = c.$$

LDEs with $\gcd(a, b) = 1 = c$ are of particular interest. If we can solve

$$ax + by = 1$$

then we have solved the problem: find $x \in \mathbb{Z}$ such that

$$ax \equiv 1 \pmod{b},$$

the problem of finding modular multiplicative inverses.

In particular, this is a crucial step in RSA key generation: the private key d satisfies

$$de \equiv 1 \pmod{\lambda(n)},$$

where n, e are the public key, and $\lambda(n)$ is easy to compute if you know the prime factorisation of n and difficult otherwise.

Lemma (Bézout's Lemma)

If $\gcd(a, b) = d$, *then the LDE* $ax + by = d$ *always has an integer solution.*

Étienne Bézout, 1730–1783

Lemma (Bézout's Lemma)

If $\gcd(a, b) = d$*, then the LDE* $ax + by = d$ *always has an integer solution.*

The statement for integers was already known before Bézout, appearing in the work of Claude Gaspard Bachet de Méziriac in 1624. Bézout's contribution was actually to extend it to polynomials, but his name has stuck to the general principle.

Étienne Bézout, 1730–1783

Claude Gaspar Bachet de Méziriac, 1581–1638

Lemma (Bézout's Lemma)

If $\gcd(a,b) = d$*, then the LDE* $ax + by = d$ *always has an integer solution.*

Étienne Bézout, 1730–1783

The statement for integers was already known before Bézout, appearing in the work of Claude Gaspard Bachet de Méziriac in 1624. Bézout's contribution was actually to extend it to polynomials, but his name has stuck to the general principle.

Many other results in number theory follow from Bézout's Lemma, such as Euclid's Lemma and Sunzi's Remainder Theorem.

Claude Gaspar Bachet de Méziriac, 1581–1638

Before we prove Bézout's Lemma, let's do an example, with $a = 48$ and $b = -35$.

$$48 = 1 \times 35 + 13 \qquad 13 = 48 - 1 \times 35$$

Before we prove Bézout's Lemma, let's do an example, with $a = 48$ and $b = -35$.

$$\begin{aligned} 48 &= 1 \times 35 + 13 & \qquad 13 &= 48 - 1 \times 35 \\ 35 &= 2 \times 13 + 9 & \qquad 9 &= 35 - 2 \times 13 \end{aligned}$$

Before we prove Bézout's Lemma, let's do an example, with $a = 48$ and $b = -35$.

$$\begin{aligned} 48 &= 1 \times 35 + 13 & 13 &= 48 - 1 \times 35 \\ 35 &= 2 \times 13 + 9 & 9 &= 35 - 2 \times 13 \\ 13 &= 1 \times 9 + 4 & 4 &= 13 - 1 \times 9 \end{aligned}$$

Before we prove Bézout's Lemma, let's do an example, with $a = 48$ and $b = -35$.

$$\begin{aligned} 48 &= 1 \times 35 + 13 & \qquad 13 &= 48 - 1 \times 35 \\ 35 &= 2 \times 13 + 9 & 9 &= 35 - 2 \times 13 \\ 13 &= 1 \times 9 + 4 & 4 &= 13 - 1 \times 9 \\ 9 &= 2 \times 4 + 1 & 1 &= 9 - 2 \times 4 \end{aligned}$$

Before we prove Bézout's Lemma, let's do an example, with $a = 48$ and $b = -35$.

$$\begin{aligned} 48 &= 1 \times 35 + 13 & 13 &= 48 - 1 \times 35 \\ 35 &= 2 \times 13 + 9 & 9 &= 35 - 2 \times 13 \\ 13 &= 1 \times 9 + 4 & 4 &= 13 - 1 \times 9 \\ 9 &= 2 \times 4 + 1 & 1 &= 9 - 2 \times 4 \end{aligned}$$

Climbing up the tower on the right-hand side,

$$1 = 9 + (-2) \times 4$$

Before we prove Bézout's Lemma, let's do an example, with $a = 48$ and $b = -35$.

$$\begin{aligned} 48 &= 1 \times 35 + 13 & \qquad 13 &= 48 - 1 \times 35 \\ 35 &= 2 \times 13 + 9 & 9 &= 35 - 2 \times 13 \\ 13 &= 1 \times 9 + 4 & 4 &= 13 - 1 \times 9 \\ 9 &= 2 \times 4 + 1 & 1 &= 9 - 2 \times 4 \end{aligned}$$

Climbing up the tower on the right-hand side,

$$\begin{aligned} 1 &= 9 + (-2) \times 4 \\ &= 9 - 2 \times (13 - 1 \times 9) \end{aligned}$$

Before we prove Bézout's Lemma, let's do an example, with $a = 48$ and $b = -35$.

$$\begin{aligned} 48 &= 1 \times 35 + 13 \qquad & 13 &= 48 - 1 \times 35 \\ 35 &= 2 \times 13 + 9 & 9 &= 35 - 2 \times 13 \\ 13 &= 1 \times 9 + 4 & 4 &= 13 - 1 \times 9 \\ 9 &= 2 \times 4 + 1 & 1 &= 9 - 2 \times 4 \end{aligned}$$

Climbing up the tower on the right-hand side,

$$\begin{aligned} 1 &= 9 + (-2) \times 4 \\ &= 9 - 2 \times (13 - 1 \times 9) \\ &= (-2) \times 13 + 3 \times 9 \end{aligned}$$

Before we prove Bézout's Lemma, let's do an example, with $a = 48$ and $b = -35$.

$$\begin{aligned} 48 &= 1 \times 35 + 13 & \qquad 13 &= 48 - 1 \times 35 \\ 35 &= 2 \times 13 + 9 & 9 &= 35 - 2 \times 13 \\ 13 &= 1 \times 9 + 4 & 4 &= 13 - 1 \times 9 \\ 9 &= 2 \times 4 + 1 & 1 &= 9 - 2 \times 4 \end{aligned}$$

Climbing up the tower on the right-hand side,

$$\begin{aligned} 1 &= 9 + (-2) \times 4 \\ &= 9 - 2 \times (13 - 1 \times 9) \\ &= (-2) \times 13 + 3 \times 9 \\ &= (-2) \times 13 + 3 \times (35 - 2 \times 13) \end{aligned}$$

Before we prove Bézout's Lemma, let's do an example, with $a = 48$ and $b = -35$.

$$\begin{aligned} 48 &= 1 \times 35 + 13 & \qquad 13 &= 48 - 1 \times 35 \\ 35 &= 2 \times 13 + 9 & 9 &= 35 - 2 \times 13 \\ 13 &= 1 \times 9 + 4 & 4 &= 13 - 1 \times 9 \\ 9 &= 2 \times 4 + 1 & 1 &= 9 - 2 \times 4 \end{aligned}$$

Climbing up the tower on the right-hand side,

$$\begin{aligned} 1 &= 9 + (-2) \times 4 \\ &= 9 - 2 \times (13 - 1 \times 9) \\ &= (-2) \times 13 + 3 \times 9 \\ &= (-2) \times 13 + 3 \times (35 - 2 \times 13) \\ &= 3 \times 35 + (-8) \times 13 \end{aligned}$$

Before we prove Bézout's Lemma, let's do an example, with $a = 48$ and $b = -35$.

$$\begin{aligned} 48 &= 1 \times 35 + 13 & \qquad 13 &= 48 - 1 \times 35 \\ 35 &= 2 \times 13 + 9 & 9 &= 35 - 2 \times 13 \\ 13 &= 1 \times 9 + 4 & 4 &= 13 - 1 \times 9 \\ 9 &= 2 \times 4 + 1 & 1 &= 9 - 2 \times 4 \end{aligned}$$

Climbing up the tower on the right-hand side,

$$\begin{aligned} 1 &= 9 + (-2) \times 4 \\ &= 9 - 2 \times (13 - 1 \times 9) \\ &= (-2) \times 13 + 3 \times 9 \\ &= (-2) \times 13 + 3 \times (35 - 2 \times 13) \\ &= 3 \times 35 + (-8) \times 13 \\ &= 3 \times 35 - 8 \times (48 - 1 \times 35) \end{aligned}$$

Before we prove Bézout's Lemma, let's do an example, with $a = 48$ and $b = -35$.

$$\begin{aligned} 48 &= 1 \times 35 + 13 & \qquad 13 &= 48 - 1 \times 35 \\ 35 &= 2 \times 13 + 9 & 9 &= 35 - 2 \times 13 \\ 13 &= 1 \times 9 + 4 & 4 &= 13 - 1 \times 9 \\ 9 &= 2 \times 4 + 1 & 1 &= 9 - 2 \times 4 \end{aligned}$$

Climbing up the tower on the right-hand side,

$$\begin{aligned} 1 &= 9 + (-2) \times 4 \\ &= 9 - 2 \times (13 - 1 \times 9) \\ &= (-2) \times 13 + 3 \times 9 \\ &= (-2) \times 13 + 3 \times (35 - 2 \times 13) \\ &= 3 \times 35 + (-8) \times 13 \\ &= 3 \times 35 - 8 \times (48 - 1 \times 35) \\ &= -8 \times 48 + 11 \times 35 \end{aligned}$$

which is the solution $(x, y) = (-8, -11)$ that we saw earlier.

How do we prove Bézout's Lemma? We run Euclid's method.

How do we prove Bézout's Lemma? We run Euclid's method.

Proof.

Since $\gcd(a, b) = d$, we know that iterated divisions of the form

$$\begin{aligned} a &= q_0 b + r_0 \\ b &= q_1 r_0 + r_1 \\ r_0 &= q_2 r_1 + r_2 \\ &\vdots \end{aligned}$$

will eventually reach $r_{i-3} = q_{i-1} r_{i-2} + d$.

Proof.

Let's rewrite this as

$$d = r_{i-3} - q_{i-1} r_{i-2}.$$

Proof.

Let's rewrite this as

$$d = r_{i-3} - q_{i-1} r_{i-2}.$$

We know that $r_{i-4} = q_{i-2} r_{i-3} + r_{i-2}$, so using this to eliminate r_{i-2} we have

$$d = -q_{i-1} r_{i-4} + (1 - q_{i-1} q_{i-2}) r_{i-3}.$$

Proof.

Let's rewrite this as

$$d = r_{i-3} - q_{i-1} r_{i-2}.$$

We know that $r_{i-4} = q_{i-2} r_{i-3} + r_{i-2}$, so using this to eliminate r_{i-2} we have

$$d = -q_{i-1} r_{i-4} + (1 - q_{i-1} q_{i-2}) r_{i-3}.$$

Proceeding by induction, we can write d as a combination of r_{i-5} and r_{i-4}, then r_{i-6} and r_{i-5}, and so on until we write

$$d = xa + yb.$$

□

Proof.

Let's rewrite this as

$$d = r_{i-3} - q_{i-1} r_{i-2}.$$

We know that $r_{i-4} = q_{i-2} r_{i-3} + r_{i-2}$, so using this to eliminate r_{i-2} we have

$$d = -q_{i-1} r_{i-4} + (1 - q_{i-1} q_{i-2}) r_{i-3}.$$

Proceeding by induction, we can write d as a combination of r_{i-5} and r_{i-4}, then r_{i-6} and r_{i-5}, and so on until we write

$$d = xa + yb.$$

□

This uses an *algorithm* to prove an existence result.

We saw in our previous calculations that $48x - 35y = 1$ had a solution $(x, y) = (-8, -11)$. However, there are other solutions, such as $(x, y) = (-43, -59)$. How do we find them *all*? What is the general solution?

We saw in our previous calculations that $48x - 35y = 1$ had a solution $(x, y) = (-8, -11)$. However, there are other solutions, such as $(x, y) = (-43, -59)$. How do we find them *all*? What is the general solution?

Suppose we have a *particular solution* (x_p, y_p) satisfying $ax_p + by_p = 1$. If we had $(\tilde{x}, \tilde{y})$ such that $a\tilde{x} + b\tilde{y} = 0$, then

$$a(x_p + \tilde{x}) + b(y_p + \tilde{y}) = ax_p + by_p = 1$$

also. Similarly, if $a(x_p + \tilde{x}) + b(y_p + \tilde{y}) = 1$, then $a\tilde{x} + b\tilde{y} = 0$.

We saw in our previous calculations that $48x - 35y = 1$ had a solution $(x, y) = (-8, -11)$. However, there are other solutions, such as $(x, y) = (-43, -59)$. How do we find them *all*? What is the general solution?

Suppose we have a *particular solution* (x_p, y_p) satisfying $ax_p + by_p = 1$. If we had $(\tilde{x}, \tilde{y})$ such that $a\tilde{x} + b\tilde{y} = 0$, then

$$a(x_p + \tilde{x}) + b(y_p + \tilde{y}) = ax_p + by_p = 1$$

also. Similarly, if $a(x_p + \tilde{x}) + b(y_p + \tilde{y}) = 1$, then $a\tilde{x} + b\tilde{y} = 0$.

What are the solutions to the *homogeneous equation* $a\tilde{x} + b\tilde{y} = 0$? Exactly $(\tilde{x}, \tilde{y}) = n(-b, a)$ for $n \in \mathbb{Z}$!

We saw in our previous calculations that $48x - 35y = 1$ had a solution $(x, y) = (-8, -11)$. However, there are other solutions, such as $(x, y) = (-43, -59)$. How do we find them *all*? What is the general solution?

Suppose we have a *particular solution* (x_p, y_p) satisfying $ax_p + by_p = 1$. If we had $(\tilde{x}, \tilde{y})$ such that $a\tilde{x} + b\tilde{y} = 0$, then

$$a(x_p + \tilde{x}) + b(y_p + \tilde{y}) = ax_p + by_p = 1$$

also. Similarly, if $a(x_p + \tilde{x}) + b(y_p + \tilde{y}) = 1$, then $a\tilde{x} + b\tilde{y} = 0$.

What are the solutions to the *homogeneous equation* $a\tilde{x} + b\tilde{y} = 0$? Exactly $(\tilde{x}, \tilde{y}) = n(-b, a)$ for $n \in \mathbb{Z}$!

The general solution to $ax + by = c$ is thus

$$\{c(x_p, y_p) + n(-b, a) : n \in \mathbb{Z}\}.$$

Here is the whole algorithm for solving an LDE $ax + by = c$.

Step 1 Calculate $d = \gcd(a, b)$. If d does not divide c, stop; there are no solutions.

Here is the whole algorithm for solving an LDE $ax + by = c$.

Step 1 Calculate $d = \gcd(a, b)$. If d does not divide c, stop; there are no solutions.

Step 2 Divide both sides of the equation by d to get $\hat{a}x + \hat{b}y = \hat{c}$.

Here is the whole algorithm for solving an LDE $ax + by = c$.

Step 1 Calculate $d = \gcd(a, b)$. If d does not divide c, stop; there are no solutions.

Step 2 Divide both sides of the equation by d to get $\hat{a}x + \hat{b}y = \hat{c}$.

Step 3 Compute a *particular solution* (x_p, y_p) of $\hat{a}x + \hat{b}y = 1$.

Here is the whole algorithm for solving an LDE $ax + by = c$.

Step 1 Calculate $d = \gcd(a, b)$. If d does not divide c, stop; there are no solutions.

Step 2 Divide both sides of the equation by d to get $\hat{a}x + \hat{b}y = \hat{c}$.

Step 3 Compute a *particular solution* (x_p, y_p) of $\hat{a}x + \hat{b}y = 1$.

Step 4 Set the general solution to be

$$\left\{\hat{c}(x_p, y_p) + n(-\hat{b}, \hat{a}) : n \in \mathbb{Z}\right\}.$$

Let's see an example. Consider $192x - 140y = 12$.

Let's see an example. Consider $192x - 140y = 12$.

Step 1 $d = \gcd(192, 140) = 4$.

Let's see an example. Consider $192x - 140y = 12$.

Step 1 $d = \gcd(192, 140) = 4$.

Step 2 Dividing both sides by d, we get $48x - 35y = 3$.

Let's see an example. Consider $192x - 140y = 12$.

Step 1 $d = \gcd(192, 140) = 4$.

Step 2 Dividing both sides by d, we get $48x - 35y = 3$.

Step 3 Solving $48x_p - 35y_p = 1$, we get $(x_p, y_p) = (-8, -11)$.

Let's see an example. Consider $192x - 140y = 12$.

Step 1 $d = \gcd(192, 140) = 4$.

Step 2 Dividing both sides by d, we get $48x - 35y = 3$.

Step 3 Solving $48x_p - 35y_p = 1$, we get $(x_p, y_p) = (-8, -11)$.

Step 4 The general solution is thus

$$\begin{aligned} &\{3(-8, -11) + n(35, 48) : n \in \mathbb{Z}\} \\ =&\{(-24, -33) + n(35, 48) : n \in \mathbb{Z}\}. \end{aligned}$$

Section 3

The extended Euclidean algorithm

We saw that the quotients computed during Euclid's algorithm tell us how to solve

$$ax + by = \gcd(a, b).$$

We saw that the quotients computed during Euclid's algorithm tell us how to solve

$$ax + by = \gcd(a, b).$$

The idea of climbing up the tower of equations backwards is intuitively useful, but it's not so amenable to computer implementation.

We saw that the quotients computed during Euclid's algorithm tell us how to solve

$$ax + by = \gcd(a, b).$$

The idea of climbing up the tower of equations backwards is intuitively useful, but it's not so amenable to computer implementation.

There's a very clever modification of Euclid's algorithm that computes a particular solution to the LDE in one pass: the *extended Euclidean algorithm*.

We saw that the quotients computed during Euclid's algorithm tell us how to solve

$$ax + by = \gcd(a, b).$$

The idea of climbing up the tower of equations backwards is intuitively useful, but it's not so amenable to computer implementation.

There's a very clever modification of Euclid's algorithm that computes a particular solution to the LDE in one pass: the *extended Euclidean algorithm*.

This appears to have first been explained by Āryabhaṭa (476–550).

Recall that Euclid's algorithm constructs a sequence

$$r_{-2}, r_{-1}, r_0, r_1, \ldots, r_{i-1},$$

where $r_{i-1} = \gcd(a, b)$ and again we denote $r_{-2} = a$, $r_{-1} = b$.

Recall that Euclid's algorithm constructs a sequence

$$r_{-2}, r_{-1}, r_0, r_1, \ldots, r_{i-1},$$

where $r_{i-1} = \gcd(a, b)$ and again we denote $r_{-2} = a$, $r_{-1} = b$.

We introduce two new sequences

$$x_{-2}, x_{-1}, x_0, x_1, \ldots, x_{i-1},$$

$$y_{-2}, y_{-1}, y_0, y_1, \ldots, y_{i-1},$$

Recall that Euclid's algorithm constructs a sequence

$$r_{-2}, r_{-1}, r_0, r_1, \ldots, r_{i-1},$$

where $r_{i-1} = \gcd(a, b)$ and again we denote $r_{-2} = a$, $r_{-1} = b$.

We introduce two new sequences

$$x_{-2}, x_{-1}, x_0, x_1, \ldots, x_{i-1},$$

$$y_{-2}, y_{-1}, y_0, y_1, \ldots, y_{i-1},$$

and we will enforce the property that

$$ax_j + by_j = r_j, \quad j = -2, \ldots, i-1.$$

Recall that Euclid's algorithm constructs a sequence

$$r_{-2}, r_{-1}, r_0, r_1, \ldots, r_{i-1},$$

where $r_{i-1} = \gcd(a, b)$ and again we denote $r_{-2} = a$, $r_{-1} = b$.

We introduce two new sequences

$$x_{-2}, x_{-1}, x_0, x_1, \ldots, x_{i-1},$$

$$y_{-2}, y_{-1}, y_0, y_1, \ldots, y_{i-1},$$

and we will enforce the property that

$$ax_j + by_j = r_j, \quad j = -2, \ldots, i-1.$$

If we can enforce this, then we will have

$$ax_{i-1} + by_{i-1} = r_{i-1} = \gcd(a, b).$$

How do we enforce

$$ax_j + by_j = r_j, \quad j = -2, \ldots, i - 1?$$

How do we enforce

$$ax_j + by_j = r_j, \quad j = -2, \ldots, i-1?$$

Well, to begin, we should set

$$(x_{-2}, y_{-2}) = (1, 0), \quad (x_{-1}, y_{-1}) = (0, 1)$$

so that our property is enforced at the start.

How do we enforce

$$ax_j + by_j = r_j, \quad j = -2, \ldots, i-1?$$

Well, to begin, we should set

$$(x_{-2}, y_{-2}) = (1, 0), \quad (x_{-1}, y_{-1}) = (0, 1)$$

so that our property is enforced at the start.

Consider some step of Euclid's method,

$$r_j = q_{j+2} r_{j+1} + r_{j+2}.$$

If we know the expansions of r_j and r_{j+1} in terms of our 'basis' a and b, then we can work out the expansion of r_{j+2} too:

$$x_{j+2} = x_j - q_{j+2} x_{j+1},$$
$$y_{j+2} = y_j - q_{j+2} y_{j+1}.$$

Section 4

Euclid for polynomials

So far we've applied Euclid's method only to integers. It applies to other types of algebraic objects, too.

So far we've applied Euclid's method only to integers. It applies to other types of algebraic objects, too.

A *polynomial* p in $\mathbb{R}[x]$ of degree $d \in \mathbb{N}$ is an expression of the form

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_d x^d,$$

where all the a_i lie in the set of real numbers $\mathbb{R}$.

So far we've applied Euclid's method only to integers. It applies to other types of algebraic objects, too.

A *polynomial* p in $\mathbb{R}[x]$ of degree $d \in \mathbb{N}$ is an expression of the form

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_d x^d,$$

where all the a_i lie in the set of real numbers $\mathbb{R}$.

A root of p is a number $x \in \mathbb{C}$ satisfying $p(x) = 0$.

So far we've applied Euclid's method only to integers. It applies to other types of algebraic objects, too.

A *polynomial* p in $\mathbb{R}[x]$ of degree $d \in \mathbb{N}$ is an expression of the form

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_d x^d,$$

where all the a_i lie in the set of real numbers $\mathbb{R}$.

A root of p is a number $x \in \mathbb{C}$ satisfying $p(x) = 0$.

Recall: dividing $p(x)$ by $q(x)$ writes

$$p(x) = c(x)q(x) + r(x)$$

with quotient $c(x)$ and remainder $r(x)$, with $\deg(r) < \deg(q)$.

The polynomials $\mathbb{R}[x]$ form a *Euclidean domain*.

The polynomials $\mathbb{R}[x]$ form a *Euclidean domain*.

This is an algebraic structure R that can be equipped with a *Euclidean function*

$$f : R \setminus \{0\} \to \mathbb{N}$$

which is something that strictly decreases on division: given $a, b \in R$, there exist $q, r \in R$, such that

$$a = qb + r,$$

and either $r = 0$ or $f(r) < f(b)$.

The polynomials $\mathbb{R}[x]$ form a *Euclidean domain*.

This is an algebraic structure R that can be equipped with a *Euclidean function*

$$f : R \setminus \{0\} \to \mathbb{N}$$

which is something that strictly decreases on division: given $a, b \in R$, there exist $q, r \in R$, such that

$$a = qb + r,$$

and either $r = 0$ or $f(r) < f(b)$.

For the polynomials, the Euclidean function is

$$f(r) = \deg(r).$$

The polynomials $\mathbb{R}[x]$ form a *Euclidean domain*.

This is an algebraic structure R that can be equipped with a *Euclidean function*

$$f : R \setminus \{0\} \to \mathbb{N}$$

which is something that strictly decreases on division: given $a, b \in R$, there exist $q, r \in R$, such that

$$a = qb + r,$$

and either $r = 0$ or $f(r) < f(b)$.

For the polynomials, the Euclidean function is

$$f(r) = \deg(r).$$

We can generalise Euclid's method, greatest common divisors, Bézout's Lemma, and many other results to such domains.

We will study algorithms for finding roots of general (i.e. not necessarily polynomial) functions in the next lectures.

We will study algorithms for finding roots of general (i.e. not necessarily polynomial) functions in the next lectures.

For now, we focus on computing *common* roots of two polynomials p and q, of possibly different degrees. The common roots are $x \in \mathbb{C}$ such that $p(x) = q(x) = 0$.

We will study algorithms for finding roots of general (i.e. not necessarily polynomial) functions in the next lectures.

For now, we focus on computing *common* roots of two polynomials p and q, of possibly different degrees. The common roots are $x \in \mathbb{C}$ such that $p(x) = q(x) = 0$.

We do this by finding the roots of the *greatest common divisor* of p and q: the polynomial of largest degree that divides both p and q.

We will study algorithms for finding roots of general (i.e. not necessarily polynomial) functions in the next lectures.

For now, we focus on computing *common* roots of two polynomials p and q, of possibly different degrees. The common roots are $x \in \mathbb{C}$ such that $p(x) = q(x) = 0$.

We do this by finding the roots of the *greatest common divisor* of p and q: the polynomial of largest degree that divides both p and q.

A number a is a root of p iff $(x - a)$ divides p, which gives the link between common roots and common divisors.

Let's see an example of applying Euclid's method. Take

$$p(x) = x^4 + x^3 - 6x^2 + 5x - 1, \quad q(x) = x^3 + x^2 + 3x - 5.$$

Let's see an example of applying Euclid's method. Take

$$p(x) = x^4 + x^3 - 6x^2 + 5x - 1, \quad q(x) = x^3 + x^2 + 3x - 5.$$

We have

$$x^4 + x^3 - 6x^2 + 5x - 1 = (x)(x^3 + x^2 + 3x - 5) + (-9x^2 + 10x - 1)$$

Let's see an example of applying Euclid's method. Take

$$p(x) = x^4 + x^3 - 6x^2 + 5x - 1, \quad q(x) = x^3 + x^2 + 3x - 5.$$

We have

$$x^4 + x^3 - 6x^2 + 5x - 1 = (x)(x^3 + x^2 + 3x - 5) + (-9x^2 + 10x - 1)$$
$$x^3 + x^2 + 3x - 5 = \left(-\frac{1}{9}x - \frac{19}{81}\right)\left(-9x^2 + 10x - 1\right) + \frac{424}{81}\left(x - 1\right)$$

Let's see an example of applying Euclid's method. Take

$$p(x) = x^4 + x^3 - 6x^2 + 5x - 1, \quad q(x) = x^3 + x^2 + 3x - 5.$$

We have

$$\begin{aligned} x^4 + x^3 - 6x^2 + 5x - 1 &= (x)(x^3 + x^2 + 3x - 5) + (-9x^2 + 10x - 1) \\ x^3 + x^2 + 3x - 5 &= \left(-\frac{1}{9}x - \frac{19}{81}\right)\left(-9x^2 + 10x - 1\right) + \frac{424}{81}(x - 1) \\ -9x^2 + 10x - 1 &= -\frac{81}{424}(9x - 1)\frac{424}{81}(x - 1) + 0. \end{aligned}$$

Let's see an example of applying Euclid's method. Take

$$p(x) = x^4 + x^3 - 6x^2 + 5x - 1, \quad q(x) = x^3 + x^2 + 3x - 5.$$

We have

$$\begin{aligned} x^4 + x^3 - 6x^2 + 5x - 1 &= (x)(x^3 + x^2 + 3x - 5) + (-9x^2 + 10x - 1) \\ x^3 + x^2 + 3x - 5 &= \left(-\frac{1}{9}x - \frac{19}{81}\right)\left(-9x^2 + 10x - 1\right) + \frac{424}{81}(x - 1) \\ -9x^2 + 10x - 1 &= -\frac{81}{424}(9x - 1)\frac{424}{81}(x - 1) + 0. \end{aligned}$$

So $(x - 1)$ is the gcd, so $x = 1$ is their only common root:

$$p(1) = 0 = q(1)$$

.

We mention some interesting applications of Euclid's method for polynomials:

We mention some interesting applications of Euclid's method for polynomials:

1. A clever way to identify the multiple roots of a polynomial p is to compute the gcd of p and its derivative p'.

We mention some interesting applications of Euclid's method for polynomials:

1. A clever way to identify the multiple roots of a polynomial p is to compute the gcd of p and its derivative p'.

2. The sequence of remainders yielded by Euclid's method applied to p and p' can be used to compute its *Sturm sequence*. The number of times the Sturm sequence changes sign can be used to calculate how many real roots p has in any given interval (including $(-\infty, \infty)$).

Jacques Charles François Sturm, 1803–1855

Many interesting polynomials are defined via recurrence relations. Euclid's method can be used to deduce facts about these without calculating them.

Many interesting polynomials are defined via recurrence relations. Euclid's method can be used to deduce facts about these without calculating them.

Consider a family of polynomials $p_k(x)$ for $k \in \mathbb{N}$ given by

$$p_0(x) = 1, \quad p_1(x) = x,$$

and

$$p_k(x) = \alpha_k(x) \times p_{k-1}(x) + \beta_k \times p_{k-2}(x),$$

with $\deg \alpha_k = 1$ and $\beta_k \in \mathbb{R} \setminus \{0\}$.

Many interesting polynomials are defined via recurrence relations. Euclid's method can be used to deduce facts about these without calculating them.

Consider a family of polynomials $p_k(x)$ for $k \in \mathbb{N}$ given by

$$p_0(x) = 1, \quad p_1(x) = x,$$

and

$$p_k(x) = \alpha_k(x) \times p_{k-1}(x) + \beta_k \times p_{k-2}(x),$$

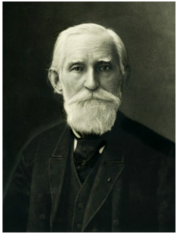
with $\deg \alpha_k = 1$ and $\beta_k \in \mathbb{R} \setminus \{0\}$.

Without specifying α_k or β_k, we can show that p_k and p_{k+1} have no common roots for $k \geq 1$.

Chebyshev polynomials

The main well-conditioned basis for polynomials used in practical computations:

$$T_0(x) = 1, \quad T_1(x) = x,$$
$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x).$$

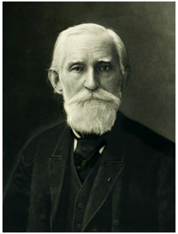


Pafnuty Chebyshev, 1821–1894

Chebyshev polynomials

The main well-conditioned basis for polynomials used in practical computations:

$$T_0(x) = 1, \quad T_1(x) = x,$$
$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x).$$



Pafnuty Chebyshev, 1821–1894

Laguerre polynomials

These describe the radial part of the solution of the Schrödinger equation for a one-electron atom:

$$L_0(x) = 1, \quad L_1(x) = -x + 1,$$
$$L_k(x) = \frac{2k+1-x}{k+1} L_{k-1}(x) - \frac{k}{k+1} L_{k-2}(x).$$

Edmond Laguerre, 1834–1886

Proof.

First note that $\deg p_k = k$, by induction.

Proof.

First note that $\deg p_k = k$, by induction. By the recursive formula, we have

$$p_{k+1}(x) = \alpha_{k+1}(x) \times p_k(x) + \beta_{k+1} \times p_{k-1}(x).$$

This is exactly the division of p_{k+1} by p_k, since $\beta_{k+1} \times p_{k-1}(x)$ is of lower degree than p_k.

Proof.

First note that $\deg p_k = k$, by induction. By the recursive formula, we have

$$p_{k+1}(x) = \alpha_{k+1}(x) \times p_k(x) + \beta_{k+1} \times p_{k-1}(x).$$

This is exactly the division of p_{k+1} by p_k, since $\beta_{k+1} \times p_{k-1}(x)$ is of lower degree than p_k. Up to a nonzero scalar (which doesn't change the roots), p_{k-1} is the remainder when p_{k+1} is divided by p_k.

Proof.

First note that $\deg p_k = k$, by induction. By the recursive formula, we have

$$p_{k+1}(x) = \alpha_{k+1}(x) \times p_k(x) + \beta_{k+1} \times p_{k-1}(x).$$

This is exactly the division of p_{k+1} by p_k, since $\beta_{k+1} \times p_{k-1}(x)$ is of lower degree than p_k. Up to a nonzero scalar (which doesn't change the roots), p_{k-1} is the remainder when p_{k+1} is divided by p_k.

Similarly, p_{k-2} is the remainder on division of p_k by p_{k-1}. Euclid's algorithm thus iterates until it terminates with

$$p_2(x) = \alpha_2(x) \times p_1(x) + \beta_2 p_0(x) = \alpha_2(x) \times x + \beta_2 \times 1,$$

so $\gcd(p_k, p_{k+1})$ is a nonzero constant (no roots). □

Computational Mathematics
Week 2: Rootfinding and fixed points

Patrick E. Farrell

University of Oxford

In the previous lecture we saw that we could use Euclid's method to compute the common roots of two polynomials p and q.

This, however, is very limited. We will want to find roots of general (not necessarily polynomial) functions $f : \mathbb{R} \to \mathbb{R}$.

In the previous lecture we saw that we could use Euclid's method to compute the common roots of two polynomials p and q.

This, however, is very limited. We will want to find roots of general (not necessarily polynomial) functions $f : \mathbb{R} \to \mathbb{R}$.

For this, we turn to *rootfinding* algorithms. There are many different ones, differing in efficiency, robustness, and applicability.

Rootfinding problem

Given $f : \mathbb{R} \to \mathbb{R}$, find $x^\star \in \mathbb{R}$ such that

$$f(x^\star) = 0.$$

Rootfinding problem

Given $f : \mathbb{R} \to \mathbb{R}$, find $x^\star \in \mathbb{R}$ such that

$$f(x^\star) = 0.$$

This problem shows up everywhere. For example, to solve an equation

$$f_1(x) = f_2(x),$$

find a root of $f(x) := f_1(x) - f_2(x)$.

Rootfinding problem

Given $f : \mathbb{R} \to \mathbb{R}$, find $x^\star \in \mathbb{R}$ such that

$$f(x^\star) = 0.$$

This problem shows up everywhere. For example, to solve an equation

$$f_1(x) = f_2(x),$$

find a root of $f(x) := f_1(x) - f_2(x)$.

Another use: if you want to calculate the decimal expansion of a number (like $\sqrt{2}$), set up a suitable equation, like

$$x^2 - 2 = 0$$

and apply a rootfinding algorithm.

The algorithms we meet here will have a different flavour than Euclid's method.

The algorithms we meet here will have a different flavour than Euclid's method.

Think back to some of the questions in Lecture 0:

- Does the algorithm terminate?
- Does the algorithm give the correct answer?
- How fast does the algorithm converge to the answer?
- How many operations does it take?

The algorithms we meet here will have a different flavour than Euclid's method.

Think back to some of the questions in Lecture 0:

- Does the algorithm terminate?
- Does the algorithm give the correct answer?
- How fast does the algorithm converge to the answer?
- How many operations does it take?

Euclid's method always terminated, always gave the exact answer, and did so in a very small number of operations.

The algorithms we meet here will have a different flavour than Euclid's method.

Think back to some of the questions in Lecture 0:

- Does the algorithm terminate?
- Does the algorithm give the correct answer?
- How fast does the algorithm converge to the answer?
- How many operations does it take?

Euclid's method always terminated, always gave the exact answer, and did so in a very small number of operations.

By contrast, rootfinding algorithms can only give *sequences* that converge to the root.

The algorithms we meet here will have a different flavour than Euclid's method.

Think back to some of the questions in Lecture 0:

- Does the algorithm terminate?
- Does the algorithm give the correct answer?
- How fast does the algorithm converge to the answer?
- How many operations does it take?

Euclid's method always terminated, always gave the exact answer, and did so in a very small number of operations.

By contrast, rootfinding algorithms can only give *sequences* that converge to the root.

Different algorithms will trade off termination, convergence speed, and operation count.

Section 2

Bisection

The first rootfinding algorithm we will meet is called the *bisection* method. It is based on the following theorem, a corollary of the Intermediate Value Theorem.

The first rootfinding algorithm we will meet is called the *bisection* method. It is based on the following theorem, a corollary of the Intermediate Value Theorem.

Bolzano's theorem (1817)

If $f : [a, b] \to \mathbb{R}$ *is continuous with* $f(a)f(b) < 0$, *then there exists* $x^\star \in (a, b)$ *with* $f(x^\star) = 0$.

The statement $f(a)f(b) < 0$ is just a fancy way of saying $f(a)$ and $f(b)$ have opposite signs.

Bernhard Bolzano, 1781–1848

The first rootfinding algorithm we will meet is called the *bisection* method. It is based on the following theorem, a corollary of the Intermediate Value Theorem.

Bolzano's theorem (1817)

If $f : [a, b] \to \mathbb{R}$ *is continuous with* $f(a)f(b) < 0$, *then there exists* $x^\star \in (a, b)$ *with* $f(x^\star) = 0$.

The statement $f(a)f(b) < 0$ is just a fancy way of saying $f(a)$ and $f(b)$ have opposite signs.

Bernhard Bolzano, 1781–1848

We evaluate f at $c = (a + b)/2$. We then have three possibilities:

1. $f(c) = 0$, so we are done!

The first rootfinding algorithm we will meet is called the *bisection* method. It is based on the following theorem, a corollary of the Intermediate Value Theorem.

Bolzano's theorem (1817)

If $f : [a, b] \to \mathbb{R}$ *is continuous with* $f(a)f(b) < 0$*, then there exists* $x^\star \in (a, b)$ *with* $f(x^\star) = 0$*.*

The statement $f(a)f(b) < 0$ is just a fancy way of saying $f(a)$ and $f(b)$ have opposite signs.

Bernhard Bolzano, 1781–1848

We evaluate f at $c = (a + b)/2$. We then have three possibilities:

1. $f(c) = 0$, so we are done!
2. $f(c)$ has the same sign as $f(a)$, so there exists a root in (c, b).

The first rootfinding algorithm we will meet is called the *bisection* method. It is based on the following theorem, a corollary of the Intermediate Value Theorem.

Bolzano's theorem (1817)

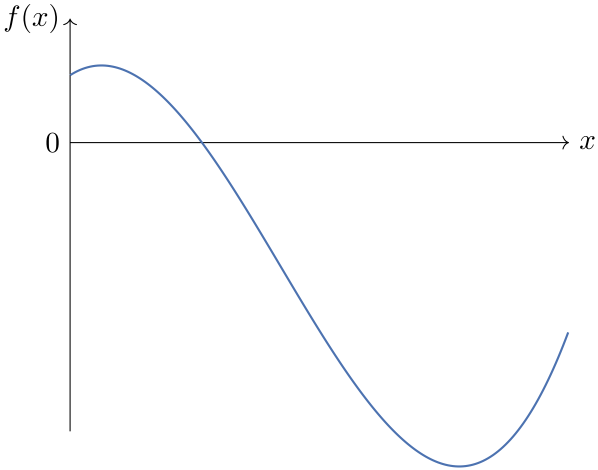
If $f : [a, b] \to \mathbb{R}$ *is continuous with* $f(a)f(b) < 0$*, then there exists* $x^\star \in (a, b)$ *with* $f(x^\star) = 0$*.*

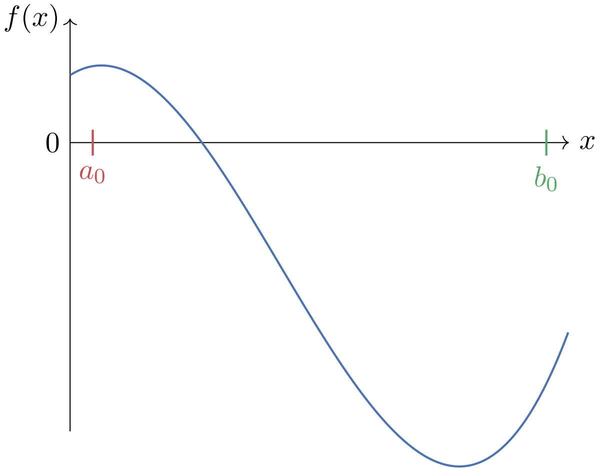
The statement $f(a)f(b) < 0$ is just a fancy way of saying $f(a)$ and $f(b)$ have opposite signs.

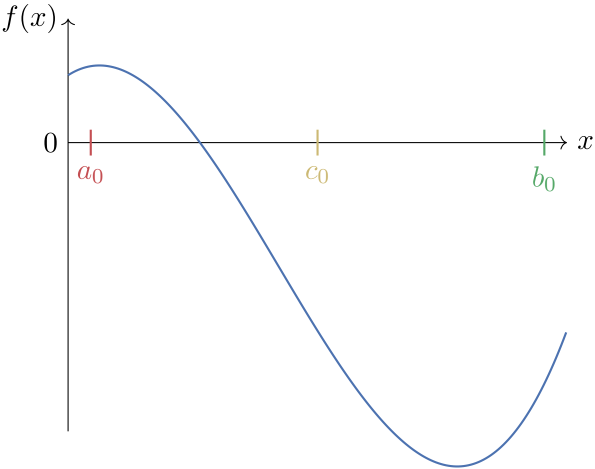
Bernhard Bolzano, 1781–1848

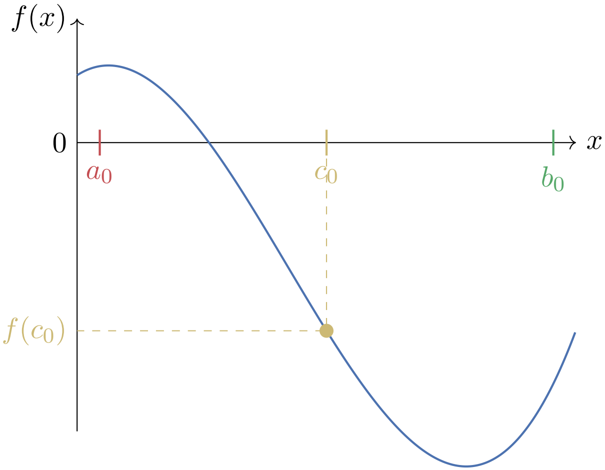
We evaluate f at $c = (a + b)/2$. We then have three possibilities:

1. $f(c) = 0$, so we are done!
2. $f(c)$ has the same sign as $f(a)$, so there exists a root in (c, b).
3. $f(c)$ has the same sign as $f(b)$, so there exists a root in (a, c).

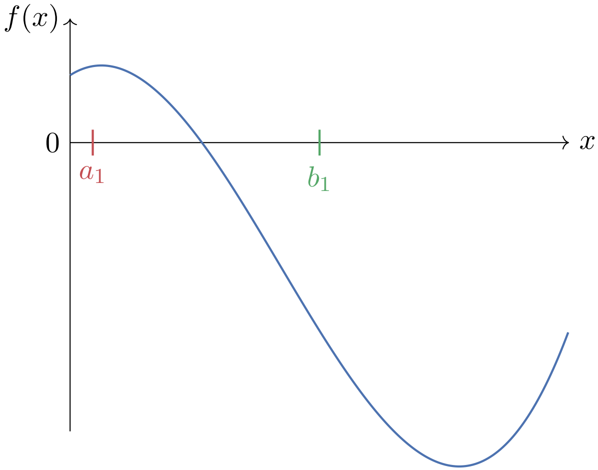

$f(x)$
0
x

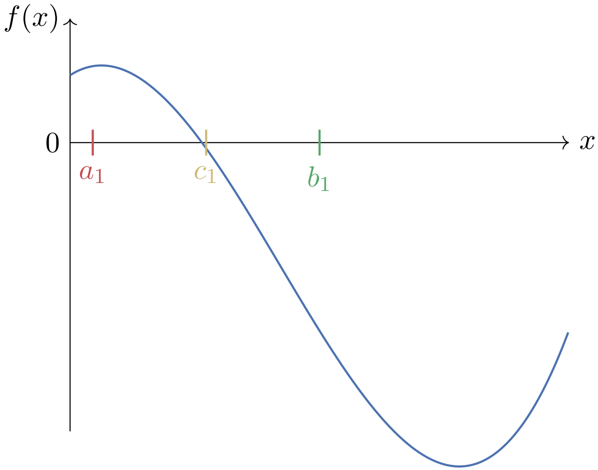

$f(x)$
0
x
a_0
b_0

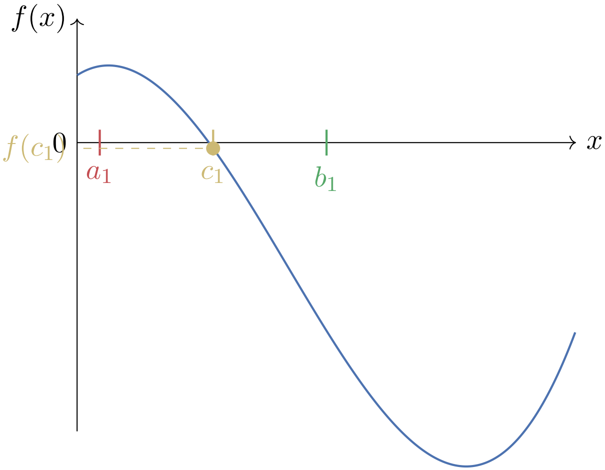

$f(x)$
0
x
a_0
c_0
b_0

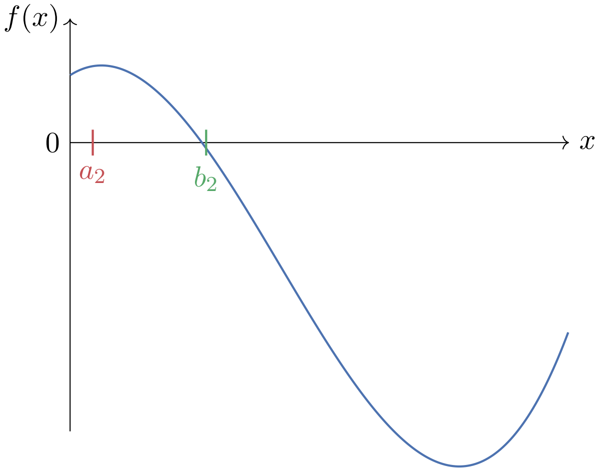


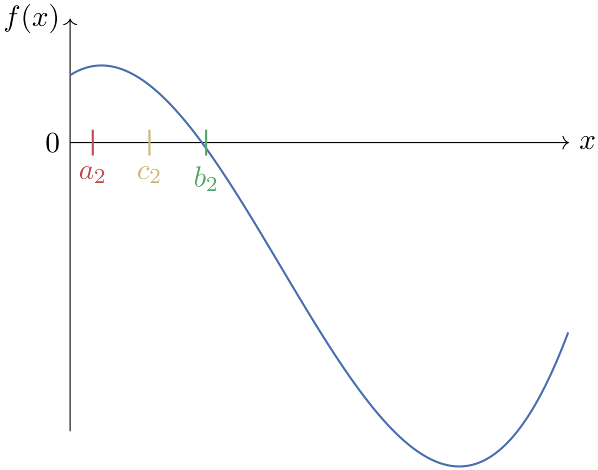
$f(x)$
0
x
a_0
c_0
b_0
$f(c_0)$


$f(x)$
0
x
a_1
b_1

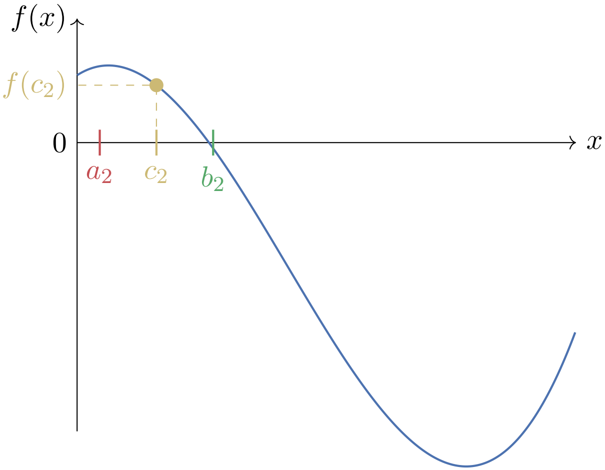

$f(x)$
0
x
a_1
c_1
b_1

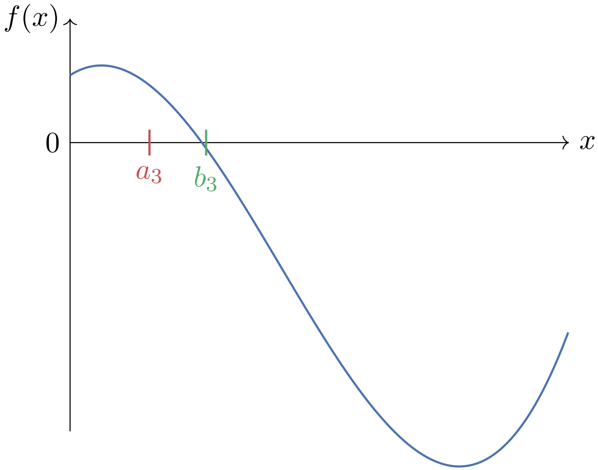

$f(x)$
x
0
$f(c_1)$
a_1
c_1
b_1

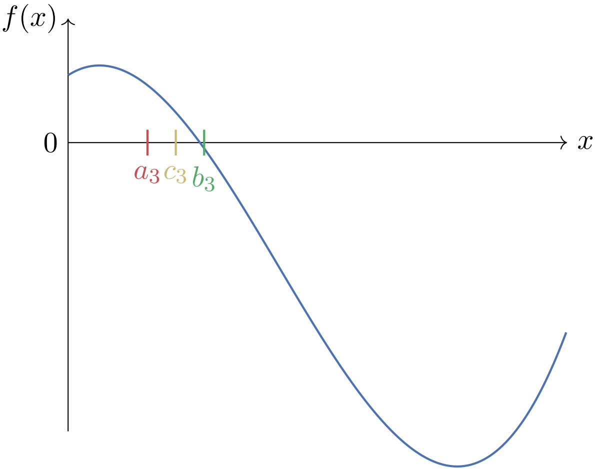

$f(x)$
0
x
a_2
b_2

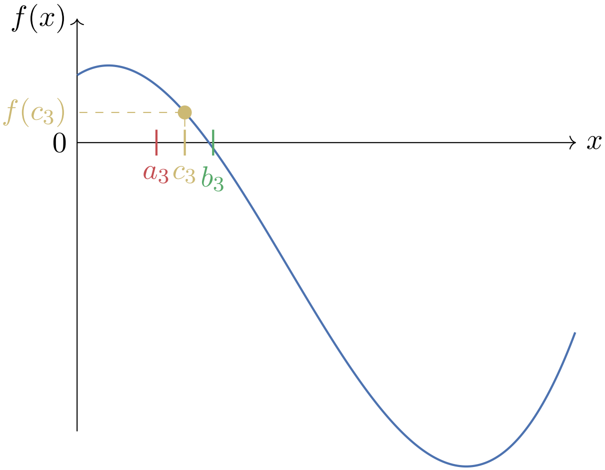


$f(x)$
0
x
a_2
c_2
b_2

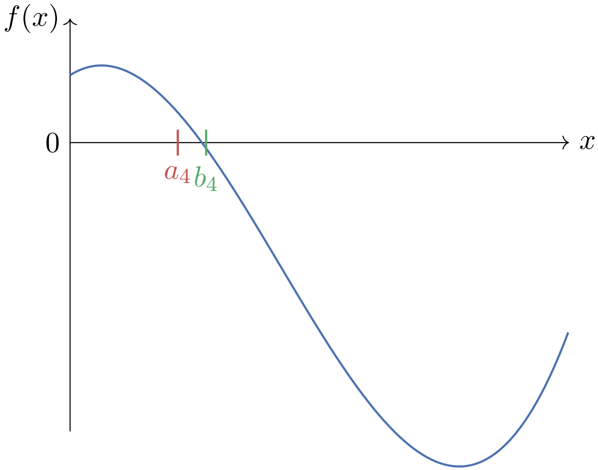

$f(x)$
$f(c_2)$
0
x
a_2
c_2
b_2


$f(x)$
0
x
a_3
b_3


$f(x)$
0
x
a_3
c_3
b_3



$f(x)$
$f(c_3)$
0
x
a_3
c_3
b_3


$f(x)$
0
x
a_4
b_4

Let's state this as an algorithm.

Assume $f : [a, b] \to \mathbb{R}$ is continuous, $f(a)f(b) < 0$, and $\mathrm{tol} > 0$.

Let's state this as an algorithm.

Assume $f : [a, b] \to \mathbb{R}$ is continuous, $f(a)f(b) < 0$, and $\mathrm{tol} > 0$.

---

function bisect(f, a, b, tol)
  while $|b - a|/2 > \mathrm{tol}$ **do**
    $c \leftarrow (a + b)/2$

Let's state this as an algorithm.

Assume $f : [a, b] \to \mathbb{R}$ is continuous, $f(a)f(b) < 0$, and $\mathrm{tol} > 0$.

---

function bisect(f, a, b, tol)
  while $|b - a|/2 > \mathrm{tol}$ **do**
    $c \leftarrow (a + b)/2$
    if $f(c) = 0$ **then return** c

Let's state this as an algorithm.

Assume $f : [a, b] \to \mathbb{R}$ is continuous, $f(a)f(b) < 0$, and $\mathrm{tol} > 0$.

---

function bisect(f, a, b, tol)
    while $|b - a|/2 > \mathrm{tol}$ **do**
        $c \leftarrow (a + b)/2$
        if $f(c) = 0$ **then return** c
        else if $f(c)f(b) < 0$ **then** $a = c$

Let's state this as an algorithm.

Assume $f : [a, b] \to \mathbb{R}$ is continuous, $f(a)f(b) < 0$, and $\mathrm{tol} > 0$.

---

function bisect(f, a, b, tol)
  while $|b - a|/2 > \mathrm{tol}$ **do**
    $c \leftarrow (a + b)/2$
    if $f(c) = 0$ **then return** c
    else if $f(c)f(b) < 0$ **then** $a = c$
    else if $f(a)f(c) < 0$ **then** $b = c$

Let's state this as an algorithm.

Assume $f : [a, b] \to \mathbb{R}$ is continuous, $f(a)f(b) < 0$, and $\mathrm{tol} > 0$.

---

function bisect(f, a, b, tol)
    while $|b - a|/2 > \mathrm{tol}$ **do**
        $c \leftarrow (a + b)/2$
        if $f(c) = 0$ **then return** c
        else if $f(c)f(b) < 0$ **then** $a = c$
        else if $f(a)f(c) < 0$ **then** $b = c$
        end if
    end while

Let's state this as an algorithm.

Assume $f : [a, b] \to \mathbb{R}$ is continuous, $f(a)f(b) < 0$, and $\mathrm{tol} > 0$.

```
function bisect(f, a, b, tol)
    while |b − a|/2 > tol do
        c ← (a + b)/2
        if f(c) = 0 then return c
        else if f(c)f(b) < 0 then a = c
        else if f(a)f(c) < 0 then b = c
        end if
    end while
    return (a + b)/2
end function
```

Let's state this as an algorithm.

Assume $f : [a, b] \to \mathbb{R}$ is continuous, $f(a)f(b) < 0$, and $\mathrm{tol} > 0$.

```
function bisect(f, a, b, tol)
    while |b − a|/2 > tol do
        c ← (a + b)/2
        if f(c) = 0 then return c
        else if f(c)f(b) < 0 then a = c
        else if f(a)f(c) < 0 then b = c
        end if
    end while
    return (a + b)/2
end function
```

Note this only uses the *sign* of the output of $f(x)$.

There's not much published information on the history of bisection. The earliest reference Prof. Hollings could find to it was in Cauchy's *Cours d'analyse* (1821).

Augustin-Louis Cauchy FRS
1789–1857

Lemma

The algorithm always terminates.

There's not much published information on the history of bisection. The earliest reference Prof. Hollings could find to it was in Cauchy's *Cours d'analyse* (1821).

Augustin-Louis Cauchy FRS
1789–1857

Lemma

The algorithm always terminates.

Proof.

In the k-th iteration of the while loop, either the function returns or it shrinks the interval by a factor of 2. For any $\mathrm{tol} > 0$, there exists $k \in \mathbb{N}$ such that $\mathrm{tol} < |b - a|/2^{k+1}$, so the algorithm must terminate. □

Let's do an example. Let's try to solve $x = \cos x$, so $f(x) = x - \cos x$.

Let's start with $[a, b] = [-10, 10]$. $f(-10) \approx -9.16$, $f(10) \approx 10.83$, so we're good to go.

Let's do an example. Let's try to solve $x = \cos x$, so $f(x) = x - \cos x$.

Let's start with $[a, b] = [-10, 10]$. $f(-10) \approx -9.16$, $f(10) \approx 10.83$, so we're good to go.

c	$f(c)$	$[a, b]$
0	-1	[0, 10]

Let's do an example. Let's try to solve $x = \cos x$, so $f(x) = x - \cos x$.

Let's start with $[a, b] = [-10, 10]$. $f(-10) \approx -9.16$, $f(10) \approx 10.83$, so we're good to go.

c	$f(c)$	$[a, b]$
0	-1	[0, 10]
5	4.71	[0, 5]

Let's do an example. Let's try to solve $x = \cos x$, so $f(x) = x - \cos x$.

Let's start with $[a, b] = [-10, 10]$. $f(-10) \approx -9.16$, $f(10) \approx 10.83$, so we're good to go.

c	$f(c)$	$[a, b]$
0	-1	[0, 10]
5	4.71	[0, 5]
2.5	3.30	[0, 2.5]

Let's do an example. Let's try to solve $x = \cos x$, so $f(x) = x - \cos x$.

Let's start with $[a, b] = [-10, 10]$. $f(-10) \approx -9.16$, $f(10) \approx 10.83$, so we're good to go.

c	$f(c)$	$[a, b]$
0	-1	[0, 10]
5	4.71	[0, 5]
2.5	3.30	[0, 2.5]
1.25	0.93	[0, 1.25]

Let's do an example. Let's try to solve $x = \cos x$, so $f(x) = x - \cos x$.

Let's start with $[a, b] = [-10, 10]$. $f(-10) \approx -9.16$, $f(10) \approx 10.83$, so we're good to go.

c	$f(c)$	$[a, b]$
0	-1	[0, 10]
5	4.71	[0, 5]
2.5	3.30	[0, 2.5]
1.25	0.93	[0, 1.25]
0.625	-0.185	[0.625, 1.25]

Let's do an example. Let's try to solve $x = \cos x$, so $f(x) = x - \cos x$.

Let's start with $[a, b] = [-10, 10]$. $f(-10) \approx -9.16$, $f(10) \approx 10.83$, so we're good to go.

c	$f(c)$	$[a, b]$
0	-1	[0, 10]
5	4.71	[0, 5]
2.5	3.30	[0, 2.5]
1.25	0.93	[0, 1.25]
0.625	-0.185	[0.625, 1.25]

The true solution is approximately $x \approx 0.739085$, so we're getting there, slowly.

Comments on bisection

- ✓ When it applies, it is guaranteed to converge.

Comments on bisection

- ✓ When it applies, it is guaranteed to converge.
- ✓ The method is very simple and very robust.

Comments on bisection

- ✓ When it applies, it is guaranteed to converge.
- ✓ The method is very simple and very robust.
- ✓ Its smoothness requirements are low (only continuity).

Comments on bisection

- ✓ When it applies, it is guaranteed to converge.
- ✓ The method is very simple and very robust.
- ✓ Its smoothness requirements are low (only continuity).
- ✗ The interval of interest only reduces by a factor of two each time.

Comments on bisection

- ✓ When it applies, it is guaranteed to converge.
- ✓ The method is very simple and very robust.
- ✓ Its smoothness requirements are low (only continuity).
- ✗ The interval of interest only reduces by a factor of two each time.
- ✗ It can be hard to find the initial points $[a, b]$ that bracket a root.

Comments on bisection

- ✓ When it applies, it is guaranteed to converge.
- ✓ The method is very simple and very robust.
- ✓ Its smoothness requirements are low (only continuity).
- ✗ The interval of interest only reduces by a factor of two each time.
- ✗ It can be hard to find the initial points $[a, b]$ that bracket a root.
- ✗ It is hard (but not impossible) to generalise to higher dimensions.

Comments on bisection

- ✓ When it applies, it is guaranteed to converge.
- ✓ The method is very simple and very robust.
- ✓ Its smoothness requirements are low (only continuity).
- ✗ The interval of interest only reduces by a factor of two each time.
- ✗ It can be hard to find the initial points $[a, b]$ that bracket a root.
- ✗ It is hard (but not impossible) to generalise to higher dimensions.
- ✗ It can never find roots of even multiplicity.

Comments on bisection

- ✓ When it applies, it is guaranteed to converge.
- ✓ The method is very simple and very robust.
- ✓ Its smoothness requirements are low (only continuity).
- ✗ The interval of interest only reduces by a factor of two each time.
- ✗ It can be hard to find the initial points $[a, b]$ that bracket a root.
- ✗ It is hard (but not impossible) to generalise to higher dimensions.
- ✗ It can never find roots of even multiplicity.

Definition (Multiplicity of a root)

A root $x^\star$ of a sufficiently differentiable $f(x)$ has multiplicity k if $f^{(n)}(x^\star) = 0$ for all $n < k$, and $f^{(k)}(x^\star) \neq 0$.

Comments on bisection

- ✓ When it applies, it is guaranteed to converge.
- ✓ The method is very simple and very robust.
- ✓ Its smoothness requirements are low (only continuity).
- ✗ The interval of interest only reduces by a factor of two each time.
- ✗ It can be hard to find the initial points $[a, b]$ that bracket a root.
- ✗ It is hard (but not impossible) to generalise to higher dimensions.
- ✗ It can never find roots of even multiplicity.

Definition (Multiplicity of a root)

A root $x^\star$ of a sufficiently differentiable $f(x)$ has multiplicity k if $f^{(n)}(x^\star) = 0$ for all $n < k$, and $f^{(k)}(x^\star) \neq 0$.

Later we will study other methods with different sets of advantages and disadvantages.

Section 3

Rate of convergence of a sequence

You've studied a great deal about *whether* sequences converge. Now let's consider: *how fast* do they converge?

You've studied a great deal about *whether* sequences converge. Now let's consider: *how fast* do they converge?

Definition (Linear convergence of a sequence)

Suppose $(x_i) \to x^\star$. We say the sequence converges linearly if there exists $\mu \in (0, 1)$ such that

$$\lim_{i \to \infty} \frac{|x_{i+1} - x^\star|}{|x_i - x^\star|} = \mu.$$

You've studied a great deal about *whether* sequences converge. Now let's consider: *how fast* do they converge?

Definition (Linear convergence of a sequence)

Suppose $(x_i) \to x^\star$. We say the sequence converges linearly if there exists $\mu \in (0, 1)$ such that

$$\lim_{i\to\infty} \frac{|x_{i+1} - x^\star|}{|x_i - x^\star|} = \mu.$$

In other words, asymptotically, moving one step along the sequence multiplies the error by a fixed $\mu < 1$. The μ is called the *rate of convergence*.

You've studied a great deal about *whether* sequences converge. Now let's consider: *how fast* do they converge?

Definition (Linear convergence of a sequence)

Suppose $(x_i) \to x^\star$. We say the sequence converges linearly if there exists $\mu \in (0, 1)$ such that

$$\lim_{i\to\infty} \frac{|x_{i+1} - x^\star|}{|x_i - x^\star|} = \mu.$$

In other words, asymptotically, moving one step along the sequence multiplies the error by a fixed $\mu < 1$. The μ is called the *rate of convergence*.

For bisection, the sequence of the midpoints of the intervals converges linearly with $\mu = 1/2$.

Can you go faster?

Can you go faster?

Definition (Superlinear convergence of a sequence)

Suppose $(x_i) \to x^\star$. We say the sequence converges superlinearly if

$$\lim_{i\to\infty} \frac{|x_{i+1} - x^\star|}{|x_i - x^\star|} = 0.$$

Can you go faster?

Definition (Superlinear convergence of a sequence)

Suppose $(x_i) \to x^\star$. We say the sequence converges superlinearly if

$$\lim_{i\to\infty} \frac{|x_{i+1} - x^\star|}{|x_i - x^\star|} = 0.$$

In other words, the sequence converges faster than any linear rate of convergence.

Can you go faster?

Definition (Superlinear convergence of a sequence)

Suppose $(x_i) \to x^\star$. We say the sequence converges superlinearly if

$$\lim_{i\to\infty} \frac{|x_{i+1} - x^\star|}{|x_i - x^\star|} = 0.$$

In other words, the sequence converges faster than any linear rate of convergence.

For example, the sequence

$$(\frac{1}{2^{2^n}}) = (\frac{1}{2}, \frac{1}{4}, \frac{1}{16}, \frac{1}{256}, \frac{1}{65535}, \dots) \to 0$$

has the ratio of successive terms going to zero too.

We can further classify superlinear convergence:

Definition (Order of convergence of a sequence)

Suppose $(x_i) \to x^\star$, superlinearly. The sequence converges with order q if

$$\lim_{i \to \infty} \frac{|x_{i+1} - x^\star|}{|x_i - x^\star|^q} = M$$

for some $M > 0$ (not necessarily $M < 1$).

We call $q = 2$ quadratic convergence, $q = 3$ cubic convergence, etc.

We can further classify superlinear convergence:

Definition (Order of convergence of a sequence)

Suppose $(x_i) \to x^\star$, superlinearly. The sequence converges with order q if

$$\lim_{i\to\infty} \frac{|x_{i+1} - x^\star|}{|x_i - x^\star|^q} = M$$

for some $M > 0$ (not necessarily $M < 1$).

We call $q = 2$ quadratic convergence, $q = 3$ cubic convergence, etc.

We will see rootfinding methods with orders of convergence $q = 2$ and $q = 3$. To develop these, we must first understand *fixed point iterations*.

Section 4

Fixed point iterations

So far we have considered rootfinding: find $x^\star \in \mathbb{R}$ such that $f(x^\star) = 0$.

So far we have considered rootfinding: find $x^\star \in \mathbb{R}$ such that $f(x^\star) = 0$.

It will be very useful to consider finding *fixed points*: given $g : [a, b] \to \mathbb{R}$, find $x^\star \in [a, b]$ such that $g(x^\star) = x^\star$.

So far we have considered rootfinding: find $x^\star \in \mathbb{R}$ such that $f(x^\star) = 0$.

It will be very useful to consider finding *fixed points*: given $g : [a, b] \to \mathbb{R}$, find $x^\star \in [a, b]$ such that $g(x^\star) = x^\star$.

We can translate between rootfinding problems and fixed point problems. For example, if you want to find the fixed points $g(x) = x$, then you can find the roots of $f(x) := g(x) - x$.

So far we have considered rootfinding: find $x^\star \in \mathbb{R}$ such that $f(x^\star) = 0$.

It will be very useful to consider finding *fixed points*: given $g : [a, b] \to \mathbb{R}$, find $x^\star \in [a, b]$ such that $g(x^\star) = x^\star$.

We can translate between rootfinding problems and fixed point problems. For example, if you want to find the fixed points $g(x) = x$, then you can find the roots of $f(x) := g(x) - x$.

Vice versa, if you have a rootfinding problem $f(x) = 0$, you could search for fixed points of $g(x) := f(x) + x$. There are other ways of transforming between them, of course.

So far we have considered rootfinding: find $x^\star \in \mathbb{R}$ such that $f(x^\star) = 0$.

It will be very useful to consider finding *fixed points*: given $g : [a, b] \to \mathbb{R}$, find $x^\star \in [a, b]$ such that $g(x^\star) = x^\star$.

We can translate between rootfinding problems and fixed point problems. For example, if you want to find the fixed points $g(x) = x$, then you can find the roots of $f(x) := g(x) - x$.

Vice versa, if you have a rootfinding problem $f(x) = 0$, you could search for fixed points of $g(x) := f(x) + x$. There are other ways of transforming between them, of course.

Transforming between the two problems is useful because there are powerful theorems that apply to finding fixed points. There's even a whole course, C4.6 Fixed Point Methods for Nonlinear PDEs, on this subject.

When can we show fixed points exist?

Theorem (Brouwer's fixed point theorem)

If $g : [a, b] \to [a, b]$ is continuous, then it has a fixed point.

Luitzen Brouwer, 1881–1966

When can we show fixed points exist?

Theorem (Brouwer's fixed point theorem)

If $g : [a, b] \to [a, b]$ is continuous, then it has a fixed point.

Luitzen Brouwer, 1881–1966

Warning (endomorphism)

Note that g must send $[a, b]$ to $[a, b]$, i.e. is an *endomorphism*. This result does *not* hold for general $g : [a, b] \to \mathbb{R}$, such as $g(x) = x + 1$.

Proof.

Since $g(x) \in [a, b]$, we have $a \leq g(x) \leq b$ for all $x \in [a, b]$. Thus $f(x) := g(x) - x$ has $f(a) \geq 0$ and $f(b) \leq 0$.

Proof.

Since $g(x) \in [a, b]$, we have $a \leq g(x) \leq b$ for all $x \in [a, b]$. Thus $f(x) := g(x) - x$ has $f(a) \geq 0$ and $f(b) \leq 0$.

If either inequality is an equality, we have a fixed point. So assume that $f(a) > 0$ and $f(b) < 0$.

Proof.

Since $g(x) \in [a, b]$, we have $a \leq g(x) \leq b$ for all $x \in [a, b]$. Thus $f(x) := g(x) - x$ has $f(a) \geq 0$ and $f(b) \leq 0$.

If either inequality is an equality, we have a fixed point. So assume that $f(a) > 0$ and $f(b) < 0$.

A root $x^\star$ of $f(x)$ thus exists in (a, b) by Bolzano's Theorem, with $g(x^\star) = x^\star$. □

That's not all! You can get uniqueness of the fixed point under stronger conditions.

That's not all! You can get uniqueness of the fixed point under stronger conditions.

Theorem

If $g : [a, b] \to [a, b]$ is differentiable with $|g'(x)| < 1$ for every $x \in (a, b)$, then g has a ***unique*** *fixed point in (a, b).*

That's not all! You can get uniqueness of the fixed point under stronger conditions.

Theorem

If $g : [a,b] \to [a,b]$ is differentiable with $|g'(x)| < 1$ for every $x \in (a,b)$, then g has a ***unique*** *fixed point in (a,b).*

Theorem (Mean value theorem, 1823)

If $g : [a,b] \to \mathbb{R}$ is differentiable, then there exists some $c \in (a,b)$ such that

$$g'(c) = \frac{g(b) - g(a)}{b - a}.$$

Augustin-Louis Cauchy FRS
1789–1857

Proof.

There must be at least one fixed point of g, since it is continuous.

Proof.

There must be at least one fixed point of g, since it is continuous.

Suppose p and q are two fixed points of g in (a, b), then we have

$$g(p) = p, \quad g(q) = q.$$

Proof.

There must be at least one fixed point of g, since it is continuous.

Suppose p and q are two fixed points of g in (a, b), then we have

$$g(p) = p, \quad g(q) = q.$$

Assume without loss of generality that $p < q$. Applying the MVT in $[p, q] \subset [a, b]$, we find that there exists $r \in (p, q)$ such that

$$g'(r) = \frac{g(q) - g(p)}{q - p} = \frac{q - p}{q - p} = 1.$$

Proof.

There must be at least one fixed point of g, since it is continuous.

Suppose p and q are two fixed points of g in (a, b), then we have

$$g(p) = p, \quad g(q) = q.$$

Assume without loss of generality that $p < q$. Applying the MVT in $[p, q] \subset [a, b]$, we find that there exists $r \in (p, q)$ such that

$$g'(r) = \frac{g(q) - g(p)}{q - p} = \frac{q - p}{q - p} = 1.$$

But $|g'(r)| < 1$ by assumption, a contradiction. □

Proof.

There must be at least one fixed point of g, since it is continuous.

Suppose p and q are two fixed points of g in (a, b), then we have

$$g(p) = p, \quad g(q) = q.$$

Assume without loss of generality that $p < q$. Applying the MVT in $[p, q] \subset [a, b]$, we find that there exists $r \in (p, q)$ such that

$$g'(r) = \frac{g(q) - g(p)}{q - p} = \frac{q - p}{q - p} = 1.$$

But $|g'(r)| < 1$ by assumption, a contradiction. □

How do we turn this into an algorithm?

Take $x_0 \in [a, b]$ and set $x_{i+1} = g(x_i)$!

Assume $g : [a, b] \to [a, b]$, and $x_0 \in [a, b]$.

Assume $g : [a, b] \to [a, b]$, and $x_0 \in [a, b]$.

---

function fixedpoint(g, x_0, tol)
    $x \leftarrow x_0$
    while $|g(x) - x| > \text{tol}$ **do**
        $x \leftarrow g(x)$
    end while

Assume $g : [a, b] \to [a, b]$, and $x_0 \in [a, b]$.

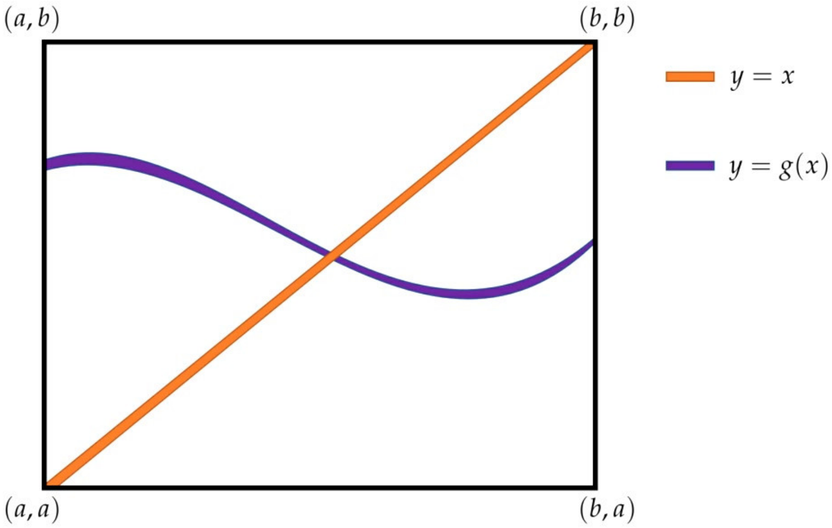
---

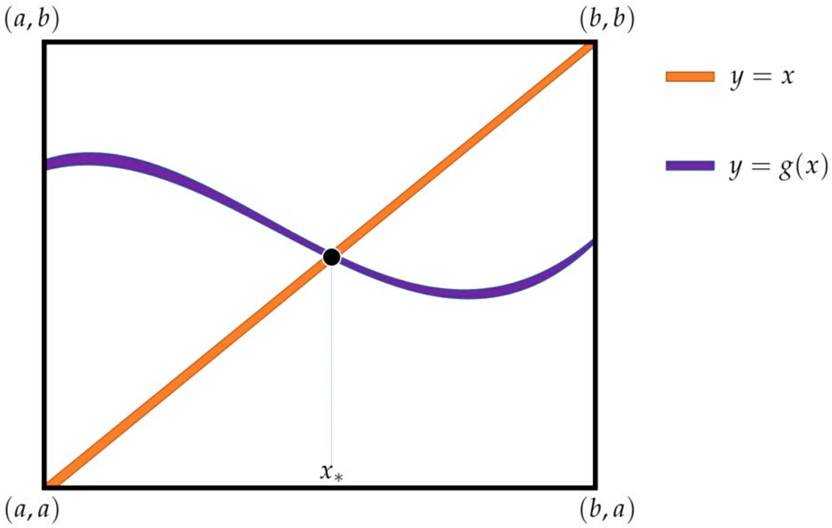
function fixedpoint(g, x_0, tol)
  $x \leftarrow x_0$
  while $|g(x) - x| > \text{tol}$ **do**
    $x \leftarrow g(x)$
  end while
  return $g(x)$
end function

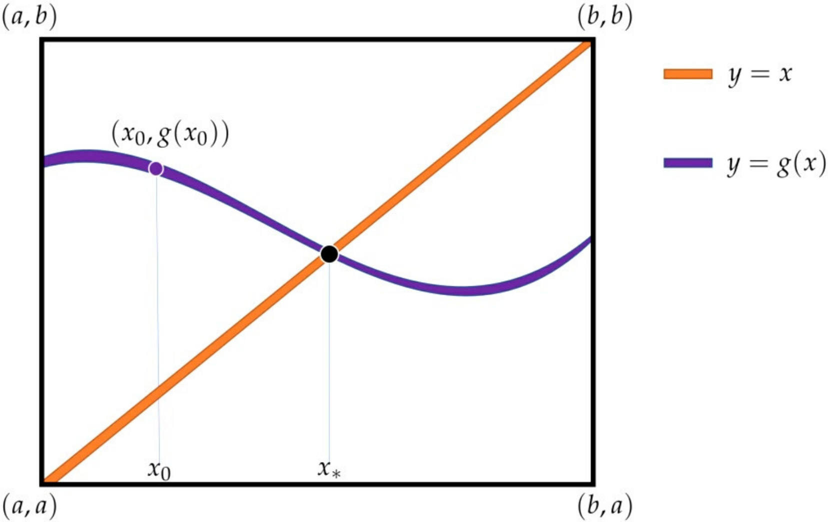
Assume $g : [a, b] \to [a, b]$, and $x_0 \in [a, b]$.

```
function fixedpoint(g, x_0, tol)
    x ← x_0
    while |g(x) − x| > tol do
        x ← g(x)
    end while
    return g(x)
end function
```

Our goal is to investigate when this converges.

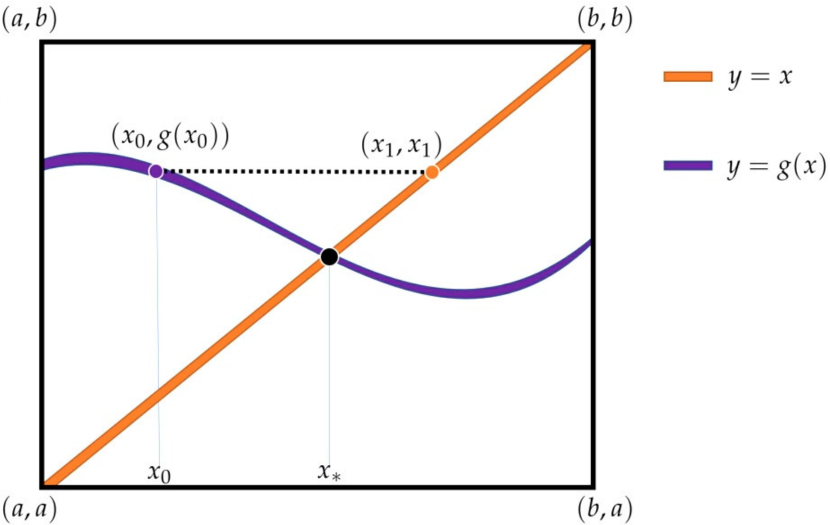

(a, b)
(b, b)
(a, a)
(b, a)
y = x
y = g(x)


(a, b)
(b, b)
$y = x$
$y = g(x)$
x_*
(a, a)
(b, a)

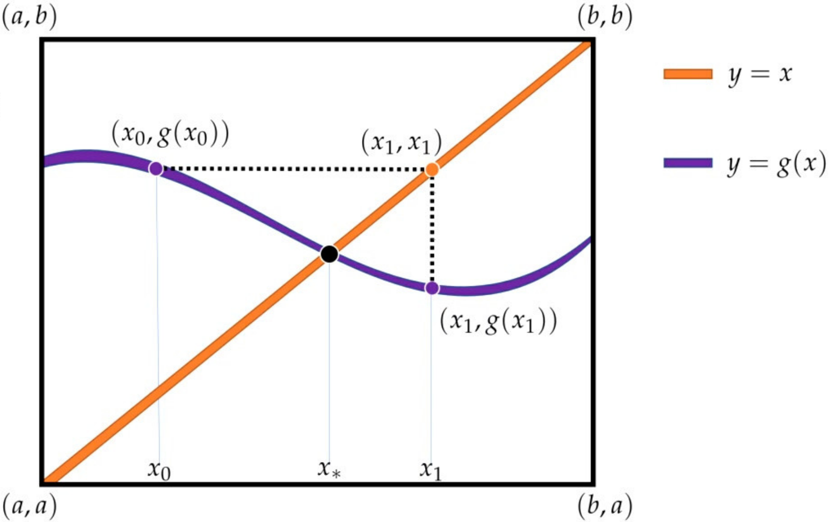


(a,b)
(b,b)
$(x_0, g(x_0))$
$y = x$
$y = g(x)$
x_0
x_*
(a,a)
(b,a)

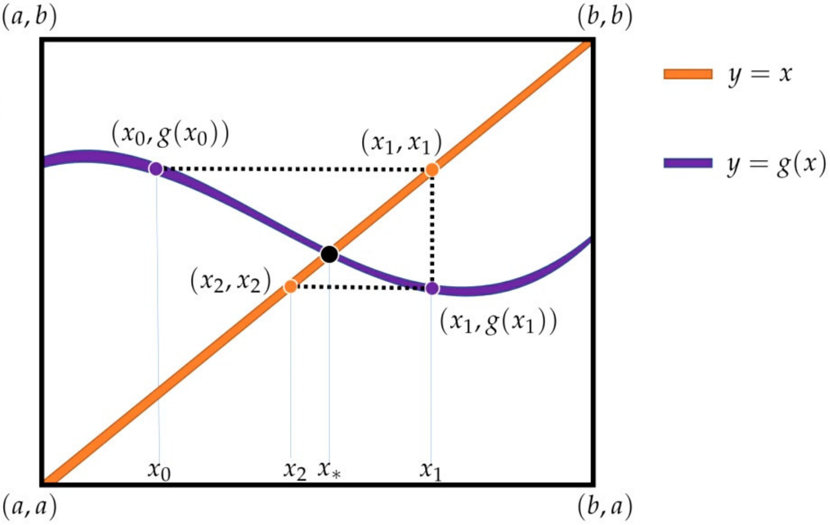
Set $x_{i+1} = g(x_i)$



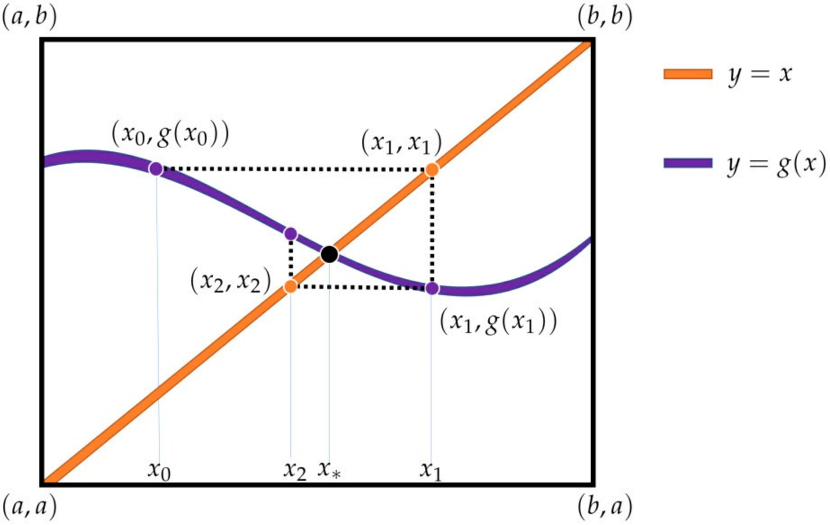
Set $x_{i+1} = g(x_i)$



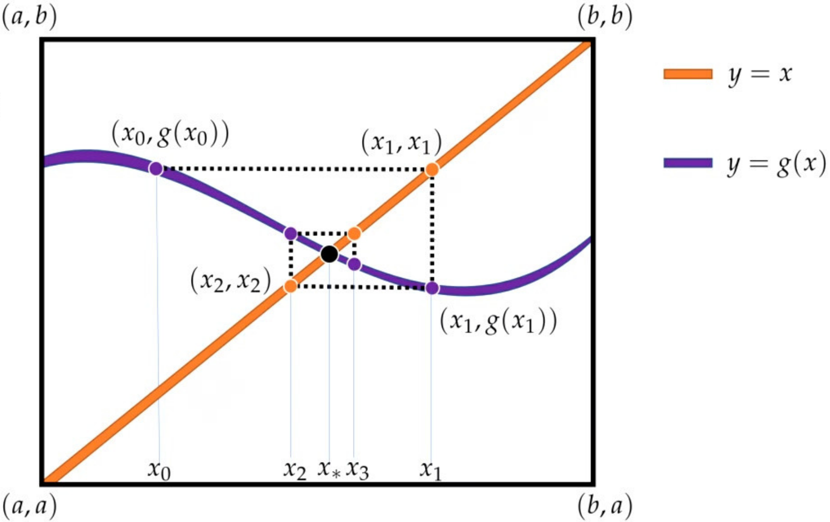
Set $x_{i+1} = g(x_i)$



Set $x_{i+1} = g(x_i)$



Set $x_{i+1} = g(x_i)$



Section 5

The contraction mapping theorem

Let's recall the setting. We have $g : [a, b] \to [a, b]$ with $|g'(x)| < 1$ for $x \in (a, b)$, and we want to find fixed points $x = g(x)$. We know that g has a unique fixed point $x^\star$.

We then proposed the iteration scheme: take any $x_0 \in [a, b]$, and set

$$x_{i+1} = g(x_i)$$

until convergence.

Let's recall the setting. We have $g : [a, b] \to [a, b]$ with $|g'(x)| < 1$ for $x \in (a, b)$, and we want to find fixed points $x = g(x)$. We know that g has a unique fixed point $x^\star$.

We then proposed the iteration scheme: take any $x_0 \in [a, b]$, and set

$$x_{i+1} = g(x_i)$$

until convergence.

Thus, we are checking $x_0, g(x_0), g(g(x_0)), \dots$ to find the unique fixed point.

Let's recall the setting. We have $g : [a, b] \to [a, b]$ with $|g'(x)| < 1$ for $x \in (a, b)$, and we want to find fixed points $x = g(x)$. We know that g has a unique fixed point $x^\star$.

We then proposed the iteration scheme: take any $x_0 \in [a, b]$, and set

$$x_{i+1} = g(x_i)$$

until convergence.

Thus, we are checking $x_0, g(x_0), g(g(x_0)), \ldots$ to find the unique fixed point.

This algorithm doesn't require derivatives. Can we devise conditions for convergence that don't require derivatives? We'll see this next.

Definition (Contraction)

A function $g : [a, b] \to [a, b]$ is called a *contraction* if there exists a constant $0 \le \gamma < 1$ such that

$$|g(x) - g(y)| \le \gamma |x - y|$$

for all $x, y \in [a, b]$.

Definition (Contraction)

A function $g : [a, b] \to [a, b]$ is called a *contraction* if there exists a constant $0 \leq \gamma < 1$ such that

$$|g(x) - g(y)| \leq \gamma |x - y|$$

for all $x, y \in [a, b]$.

Example

Any differentiable $g : [a, b] \to [a, b]$ with $|g'(x)| \leq \gamma < 1$ for $x \in (a, b)$ is a contraction. For $x, y \in [a, b]$, by the MVT there exists $c \in (x, y)$ such that

$$|g(x) - g(y)| = |g'(c)(x - y)| \leq \gamma |x - y|.$$

Definition (Contraction)

A function $g : [a, b] \to [a, b]$ is called a *contraction* if there exists a constant $0 \leq \gamma < 1$ such that

$$|g(x) - g(y)| \leq \gamma |x - y|$$

for all $x, y \in [a, b]$.

Example

Any differentiable $g : [a, b] \to [a, b]$ with $|g'(x)| \leq \gamma < 1$ for $x \in (a, b)$ is a contraction. For $x, y \in [a, b]$, by the MVT there exists $c \in (x, y)$ such that

$$|g(x) - g(y)| = |g'(c)(x - y)| \leq \gamma |x - y|.$$

Not all contractions are differentiable. For example,

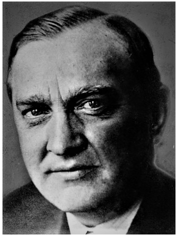
$$g(x) = |x|/2$$

is a contraction with $\gamma = 1/2$, but is not differentiable.

Contraction mapping theorem (1922)

If $g : [a, b] \to [a, b]$ *is a contraction, then it has a unique fixed point* $x^\star$*, and the iteration scheme* $x_{i+1} = g(x_i)$ *converges at least linearly to* $x^\star$ *for any* $x_0 \in [a, b]$.

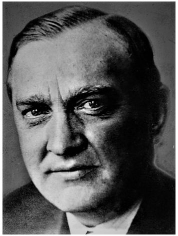
Banach proved his theorem on more general *complete metric spaces*.



Stefan Banach, 1892–1945

Contraction mapping theorem (1922)

If $g : [a, b] \to [a, b]$ *is a contraction, then it has a unique fixed point* $x^\star$*, and the iteration scheme* $x_{i+1} = g(x_i)$ *converges at least linearly to* $x^\star$ *for any* $x_0 \in [a, b]$.



Stefan Banach, 1892–1945

Banach proved his theorem on more general *complete metric spaces*.

Banach was a Pole who spent his entire academic career in Lwów (now Lviv).

Proof.

We prove the theorem in stages. First, we show g is continuous, and thus must have a fixed point.

Proof.

We prove the theorem in stages. First, we show g is continuous, and thus must have a fixed point.

If $\gamma = 0$ then $g(x) =$ const which is continuous, so assume $\gamma > 0$. Take arbitrary $\varepsilon > 0$ and choose $\delta = \varepsilon/\gamma$. Then if $|x - y| < \delta$, we have

$$|x - y| < \varepsilon/\gamma \implies \gamma|x - y| < \varepsilon,$$

and since $|g(x) - g(y)| \leq \gamma|x - y|$ by assumption, $|g(x) - g(y)| < \varepsilon$.

Proof.

We prove the theorem in stages. First, we show g is continuous, and thus must have a fixed point.

If $\gamma = 0$ then $g(x) =$ const which is continuous, so assume $\gamma > 0$. Take arbitrary $\varepsilon > 0$ and choose $\delta = \varepsilon/\gamma$. Then if $|x - y| < \delta$, we have

$$|x - y| < \varepsilon/\gamma \implies \gamma|x - y| < \varepsilon,$$

and since $|g(x) - g(y)| \leq \gamma|x - y|$ by assumption, $|g(x) - g(y)| < \varepsilon$.

We thus know by Brouwer's theorem that g must have a fixed point.

Proof.

We now show that the fixed point of g is unique. Suppose p and q are two fixed points of g. Then $g(p) = p$ and $g(q) = q$, so

$$|p - q| = |g(p) - g(q)| \leq \gamma |p - q|$$

and since $\gamma < 1$, this can only be satisfied if $|p - q| = 0$, so $p = q$.

Proof.

We now show convergence for arbitrary $x_0 \in [a, b]$. Recall that $x_i = g(x_{i-1})$ and consider

$$|x_i - x^\star| = |g(x_{i-1}) - g(x^\star)| \leq \gamma |x_{i-1} - x^\star|$$

Proof.

We now show convergence for arbitrary $x_0 \in [a, b]$. Recall that $x_i = g(x_{i-1})$ and consider

$$
\begin{aligned}
|x_i - x^\star| = |g(x_{i-1}) - g(x^\star)| &\le \gamma |x_{i-1} - x^\star| \\
&\le \gamma^2 |x_{i-2} - x^\star|
\end{aligned}
$$

Proof.

We now show convergence for arbitrary $x_0 \in [a, b]$. Recall that $x_i = g(x_{i-1})$ and consider

$$\begin{aligned} |x_i - x^\star| = |g(x_{i-1}) - g(x^\star)| &\leq \gamma |x_{i-1} - x^\star| \\ &\leq \gamma^2 |x_{i-2} - x^\star| \\ &\leq \gamma^i |x_0 - x^\star|. \end{aligned}$$

Proof.

We now show convergence for arbitrary $x_0 \in [a, b]$. Recall that $x_i = g(x_{i-1})$ and consider

$$
\begin{aligned}
|x_i - x^\star| = |g(x_{i-1}) - g(x^\star)| &\leq \gamma |x_{i-1} - x^\star| \\
&\leq \gamma^2 |x_{i-2} - x^\star| \\
&\leq \gamma^i |x_0 - x^\star|.
\end{aligned}
$$

Since $\gamma < 1$, $\gamma^i \to 0$, while $|x_0 - x^\star|$ is fixed. Thus

$$\lim_{i \to \infty} |x_i - x^\star| = 0,$$

i.e. $x_i \to x^\star$.

Proof.

We now show convergence for arbitrary $x_0 \in [a, b]$. Recall that $x_i = g(x_{i-1})$ and consider

$$\begin{aligned} |x_i - x^\star| = |g(x_{i-1}) - g(x^\star)| &\le \gamma |x_{i-1} - x^\star| \\ &\le \gamma^2 |x_{i-2} - x^\star| \\ &\le \gamma^i |x_0 - x^\star|. \end{aligned}$$

Since $\gamma < 1$, $\gamma^i \to 0$, while $|x_0 - x^\star|$ is fixed. Thus

$$\lim_{i \to \infty} |x_i - x^\star| = 0,$$

i.e. $x_i \to x^\star$. Since

$$\frac{|x_i - x^\star|}{|x_{i-1} - x^\star|} \le \gamma,$$

the convergence is at least linear with rate $\gamma < 1$. □

Let's review the conditions for our theorems.

Let's review the conditions for our theorems.

Existence of fixed point: $g : [a, b] \to [a, b]$ continuous.

Let's review the conditions for our theorems.

Existence of fixed point: $g : [a, b] \to [a, b]$ continuous.

Uniqueness of fixed point: g differentiable with $|g'(x)| < 1$ for all $x \in (a, b)$.

Let's review the conditions for our theorems.

Existence of fixed point: $g : [a,b] \to [a,b]$ continuous.

Uniqueness of fixed point: g differentiable with $|g'(x)| < 1$ for all $x \in (a,b)$.

Contraction mapping theorem: g a contraction, i.e. with $|g'(x)| \leq \gamma < 1$ for all $x \in (a,b)$ in the differentiable case.

Let's review the conditions for our theorems.

Existence of fixed point: $g : [a, b] \to [a, b]$ continuous.

Uniqueness of fixed point: g differentiable with $|g'(x)| < 1$ for all $x \in (a, b)$.

Contraction mapping theorem: g a contraction, i.e. with $|g'(x)| \leq \gamma < 1$ for all $x \in (a, b)$ in the differentiable case.

Let's explore some examples on the edges of these results.

First, let's consider

$$g : [0, 1] \to [0, 1], \quad g(x) = x.$$

This is differentiable but has $|g'(x)| = 1$. Clearly this has an infinite number of fixed points.

First, let's consider

$$g : [0,1] \to [0,1], \quad g(x) = x.$$

This is differentiable but has $|g'(x)| = 1$. Clearly this has an infinite number of fixed points.

You can have a unique fixed point of a differentiable function without being a contraction. An example is

$$g : [0,\pi] \to [0,1] \subset [0,\pi], \quad g : x \mapsto \sin x.$$

This has $|g'(x)| < 1$ for $x \in (0,\pi)$, so has a unique fixed point $x^\star = 0$. But it is not a contraction, since $g'(0) = \cos(0) = 1$; there is no $\gamma < 1$ such that $|g'(x)| \leq \gamma$ on $(0,\pi)$. The fixed point iteration converges, but so slowly as to be absolutely useless.

Section 6

Example

Suppose we wish to find the roots of $f(x) = x^2 - x - 1 = 0$. (Its roots are the golden ratio $\phi \approx 1.61834$ and its conjugate $-\phi^{-1} \approx -0.618034$.)

Suppose we wish to find the roots of $f(x) = x^2 - x - 1 = 0$. (Its roots are the golden ratio $\phi \approx 1.61834$ and its conjugate $-\phi^{-1} \approx -0.618034$.)

Let's manipulate f to recast the problem as a fixed point problem. There are many ways to do this.

Fixed point iteration A

$$x^2 - x - 1 = 0$$

Suppose we wish to find the roots of $f(x) = x^2 - x - 1 = 0$. (Its roots are the golden ratio $\phi \approx 1.61834$ and its conjugate $-\phi^{-1} \approx -0.618034$.)

Let's manipulate f to recast the problem as a fixed point problem. There are many ways to do this.

Fixed point iteration A

$$x^2 - x - 1 = 0 \implies x^2 = x + 1$$

Suppose we wish to find the roots of $f(x) = x^2 - x - 1 = 0$. (Its roots are the golden ratio $\phi \approx 1.61834$ and its conjugate $-\phi^{-1} \approx -0.618034$.)

Let's manipulate f to recast the problem as a fixed point problem. There are many ways to do this.

Fixed point iteration A

$$x^2 - x - 1 = 0 \implies x^2 = x + 1 \implies x = (x+1)/x =: g_A(x)$$

Suppose we wish to find the roots of $f(x) = x^2 - x - 1 = 0$. (Its roots are the golden ratio $\phi \approx 1.61834$ and its conjugate $-\phi^{-1} \approx -0.618034$.)

Let's manipulate f to recast the problem as a fixed point problem. There are many ways to do this.

Fixed point iteration A

$$x^2 - x - 1 = 0 \implies x^2 = x + 1 \implies x = (x+1)/x =: g_A(x)$$

Fixed point iteration B

$$x^2 - x - 1 = 0$$

Suppose we wish to find the roots of $f(x) = x^2 - x - 1 = 0$. (Its roots are the golden ratio $\phi \approx 1.61834$ and its conjugate $-\phi^{-1} \approx -0.618034$.)

Let's manipulate f to recast the problem as a fixed point problem. There are many ways to do this.

Fixed point iteration A

$$x^2 - x - 1 = 0 \implies x^2 = x + 1 \implies x = (x+1)/x =: g_A(x)$$

Fixed point iteration B

$$x^2 - x - 1 = 0 \implies x = x^2 - 1 =: g_B(x)$$

Suppose we wish to find the roots of $f(x) = x^2 - x - 1 = 0$. (Its roots are the golden ratio $\phi \approx 1.61834$ and its conjugate $-\phi^{-1} \approx -0.618034$.)

Let's manipulate f to recast the problem as a fixed point problem. There are many ways to do this.

Fixed point iteration A

$$x^2 - x - 1 = 0 \implies x^2 = x + 1 \implies x = (x+1)/x =: g_A(x)$$

Fixed point iteration B

$$x^2 - x - 1 = 0 \implies x = x^2 - 1 =: g_B(x)$$

Fixed point iteration C

$$x^2 - x - 1 = 0$$

Suppose we wish to find the roots of $f(x) = x^2 - x - 1 = 0$. (Its roots are the golden ratio $\phi \approx 1.61834$ and its conjugate $-\phi^{-1} \approx -0.618034$.)

Let's manipulate f to recast the problem as a fixed point problem. There are many ways to do this.

Fixed point iteration A

$$x^2 - x - 1 = 0 \implies x^2 = x + 1 \implies x = (x+1)/x =: g_A(x)$$

Fixed point iteration B

$$x^2 - x - 1 = 0 \implies x = x^2 - 1 =: g_B(x)$$

Fixed point iteration C

$$x^2 - x - 1 = 0 \implies x(x-1) = 1$$

Suppose we wish to find the roots of $f(x) = x^2 - x - 1 = 0$. (Its roots are the golden ratio $\phi \approx 1.61834$ and its conjugate $-\phi^{-1} \approx -0.618034$.)

Let's manipulate f to recast the problem as a fixed point problem. There are many ways to do this.

Fixed point iteration A

$$x^2 - x - 1 = 0 \implies x^2 = x + 1 \implies x = (x+1)/x =: g_A(x)$$

Fixed point iteration B

$$x^2 - x - 1 = 0 \implies x = x^2 - 1 =: g_B(x)$$

Fixed point iteration C

$$x^2 - x - 1 = 0 \implies x(x-1) = 1 \implies x = 1/(x-1) =: g_C(x)$$

Comment

Rootfinding with fixed point iteration doesn't typically rely on manual manipulation like this.

We'll see *generic* ways of transforming a rootfinding problem into a fixed point problem that work for very broad classes of functions.

If we run the fixed point iteration with $x_0 = 1.1$, we get

iteration	$g_A(x) = (x+1)/x$	$g_B(x) = x^2 - 1$	$g_C(x) = 1/(x-1)$
1	1.909091	0.210000	10.00000
2	1.523810	-0.955900	0.111111
3	1.656250	-0.086255	-1.125000
4	1.603774	-0.992560	-0.470588
5	1.623529	-0.014825	-0.680000
6	1.615942	-0.999780	-0.595238
7	1.618834	-0.000439	-0.626866
8	1.617729	-1.000000	-0.614679
9	1.618151	-0.000000	-0.619318
10	1.617989	-1.000000	-0.617544

If we run the fixed point iteration with $x_0 = 1.1$, we get

iteration	$g_A(x) = (x+1)/x$	$g_B(x) = x^2 - 1$	$g_C(x) = 1/(x-1)$
1	1.909091	0.210000	10.00000
2	1.523810	-0.955900	0.111111
3	1.656250	-0.086255	-1.125000
4	1.603774	-0.992560	-0.470588
5	1.623529	-0.014825	-0.680000
6	1.615942	-0.999780	-0.595238
7	1.618834	-0.000439	-0.626866
8	1.617729	-1.000000	-0.614679
9	1.618151	-0.000000	-0.619318
10	1.617989	-1.000000	-0.617544

Can we explain this?

Let's check if we can find γ and $[a,b]$ such that $g([a,b]) \subset [a,b]$ and $|g'(x)| \leq \gamma < 1$ on (a,b).

Case A: $g(x) = (x+1)/x$

Its derivative is $g'(x) = -1/x^2$. On $[a,b] = [1,2]$ this is increasing, but $g'(1) = -1$. So let's try $[a,b] = [1.1, 2]$. We then have $\gamma = |g'(1.1)| \approx 0.826 < 1$.

Let's check if we can find γ and $[a, b]$ such that $g([a, b]) \subset [a, b]$ and $|g'(x)| \leq \gamma < 1$ on (a, b).

Case A: $g(x) = (x + 1)/x$

Its derivative is $g'(x) = -1/x^2$. On $[a, b] = [1, 2]$ this is increasing, but $g'(1) = -1$. So let's try $[a, b] = [1.1, 2]$. We then have $\gamma = |g'(1.1)| \approx 0.826 < 1$.

We also need to check that $g([a, b]) \subset [a, b]$. $g(x) = 1 + 1/x$, so the function is decreasing on $[a, b]$. Checking, we find $g(1.1) = 1.9$ and $g(2) = 1.5$, so this is satisfied.

Let's check if we can find γ and $[a, b]$ such that $g([a, b]) \subset [a, b]$ and $|g'(x)| \leq \gamma < 1$ on (a, b).

Case A: $g(x) = (x + 1)/x$

Its derivative is $g'(x) = -1/x^2$. On $[a, b] = [1, 2]$ this is increasing, but $g'(1) = -1$. So let's try $[a, b] = [1.1, 2]$. We then have $\gamma = |g'(1.1)| \approx 0.826 < 1$.

We also need to check that $g([a, b]) \subset [a, b]$. $g(x) = 1 + 1/x$, so the function is decreasing on $[a, b]$. Checking, we find $g(1.1) = 1.9$ and $g(2) = 1.5$, so this is satisfied.

Banach's contraction mapping theorem thus applies.

Let's check if we can find γ and $[a, b]$ such that $g([a, b]) \subset [a, b]$ and $|g'(x)| \leq \gamma < 1$ on (a, b).

Case B: $g(x) = x^2 - 1$

Its derivative is $g'(x) = 2x$. We have $g'(\phi) \approx 3.23 > 1$ and $g'(-\phi^{-1}) \approx -1.23 < -1$. So there can be no interval containing the root that satisfies the criteria.

Let's check if we can find γ and $[a, b]$ such that $g([a, b]) \subset [a, b]$ and $|g'(x)| \leq \gamma < 1$ on (a, b).

Case C: $g(x) = 1/(x - 1)$

Its derivative is $g'(x) = -1/(x-1)^2$, with $g'(\phi) \approx -2.6 < -1$, and $g'(-\phi^{-1}) \approx -0.38$.
Taking $[a, b] = [-0.8, -0.4]$, we have g' is a decreasing function, and $\gamma = |g'(-0.4)| \approx 0.51$.

Let's check if we can find γ and $[a, b]$ such that $g([a, b]) \subset [a, b]$ and $|g'(x)| \leq \gamma < 1$ on (a, b).

Case C: $g(x) = 1/(x-1)$

Its derivative is $g'(x) = -1/(x-1)^2$, with $g'(\phi) \approx -2.6 < -1$, and $g'(-\phi^{-1}) \approx -0.38$. Taking $[a, b] = [-0.8, -0.4]$, we have g' is a decreasing function, and $\gamma = |g'(-0.4)| \approx 0.51$.

On $[-0.8, -0.4]$, g is a decreasing function, so we just need to check the endpoints. We have $g(-0.8) \approx -0.555$ and $g(-0.4) \approx -0.714$, so $g([a, b]) \subset [a, b]$.

Let's check if we can find γ and $[a, b]$ such that $g([a, b]) \subset [a, b]$ and $|g'(x)| \leq \gamma < 1$ on (a, b).

Case C: $g(x) = 1/(x-1)$

Its derivative is $g'(x) = -1/(x-1)^2$, with $g'(\phi) \approx -2.6 < -1$, and $g'(-\phi^{-1}) \approx -0.38$. Taking $[a, b] = [-0.8, -0.4]$, we have g' is a decreasing function, and $\gamma = |g'(-0.4)| \approx 0.51$.

On $[-0.8, -0.4]$, g is a decreasing function, so we just need to check the endpoints. We have $g(-0.8) \approx -0.555$ and $g(-0.4) \approx -0.714$, so $g([a, b]) \subset [a, b]$.

Banach's contraction mapping theorem thus applies.

Section 7

Termination criteria

In the statement of the algorithm we looped until $|g(x) - x| \leq \text{tol}$. This does not guarantee anything about the error $|x - x^\star|$! Can we do better?

In the statement of the algorithm we looped until $|g(x) - x| \leq \mathrm{tol}$. This does not guarantee anything about the error $|x - x^\star|$! Can we do better?

In the proof, we saw that $|x_i - x^\star| \leq \gamma^i |x_0 - x^\star|$. Since $x_0, x^\star \in [a, b]$, we can bound this by $\gamma^i |b - a|$.

In the statement of the algorithm we looped until $|g(x) - x| \leq \mathrm{tol}$. This does not guarantee anything about the error $|x - x^\star|$! Can we do better?

In the proof, we saw that $|x_i - x^\star| \leq \gamma^i |x_0 - x^\star|$. Since $x_0, x^\star \in [a, b]$, we can bound this by $\gamma^i |b - a|$.

Thus, to achieve a tolerance tol *on the error*, we choose i such that $\gamma^i \leq \mathrm{tol}/|b - a|$.

In the statement of the algorithm we looped until $|g(x) - x| \leq \mathrm{tol}$. This does not guarantee anything about the error $|x - x^\star|$! Can we do better?

In the proof, we saw that $|x_i - x^\star| \leq \gamma^i |x_0 - x^\star|$. Since $x_0, x^\star \in [a, b]$, we can bound this by $\gamma^i |b - a|$.

Thus, to achieve a tolerance tol *on the error*, we choose i such that $\gamma^i \leq \mathrm{tol}/|b - a|$.

This reminds us we want a contraction with a small γ: if $\gamma \approx 1$, we will require many iterations to converge.

In the statement of the algorithm we looped until $|g(x) - x| \leq \mathrm{tol}$. This does not guarantee anything about the error $|x - x^\star|$! Can we do better?

In the proof, we saw that $|x_i - x^\star| \leq \gamma^i |x_0 - x^\star|$. Since $x_0, x^\star \in [a, b]$, we can bound this by $\gamma^i |b - a|$.

Thus, to achieve a tolerance tol *on the error*, we choose i such that $\gamma^i \leq \mathrm{tol}/|b - a|$.

This reminds us we want a contraction with a small γ: if $\gamma \approx 1$, we will require many iterations to converge.

This is an *a priori* error estimate: we can compute it before ever doing any computations, or choosing x_0. What can we do if we know more?

From the contraction property, we know that

$$|x_i - x_{i-1}| \leq \gamma |x_{i-1} - x_{i-2}|$$

for $i > 2$. Take a fixed $J > i$. We can expand $|x_J - x_i|$ as

From the contraction property, we know that

$$|x_i - x_{i-1}| \leq \gamma |x_{i-1} - x_{i-2}|$$

for $i > 2$. Take a fixed $J > i$. We can expand $|x_J - x_i|$ as

$$|x_J - x_i| = |(x_J - x_{J-1}) + (x_{J-1} - x_{J-2}) + \cdots + (x_{i+1} - x_i)|$$

From the contraction property, we know that

$$|x_i - x_{i-1}| \leq \gamma |x_{i-1} - x_{i-2}|$$

for $i > 2$. Take a fixed $J > i$. We can expand $|x_J - x_i|$ as

$$\begin{aligned} |x_J - x_i| &= |(x_J - x_{J-1}) + (x_{J-1} - x_{J-2}) + \cdots + (x_{i+1} - x_i)| \\ &\leq |x_J - x_{J-1}| + |x_{J-1} - x_{J-2}| + \cdots + |x_{i+1} - x_i| \end{aligned}$$

From the contraction property, we know that

$$|x_i - x_{i-1}| \leq \gamma |x_{i-1} - x_{i-2}|$$

for $i > 2$. Take a fixed $J > i$. We can expand $|x_J - x_i|$ as

$$\begin{aligned}
|x_J - x_i| &= |(x_J - x_{J-1}) + (x_{J-1} - x_{J-2}) + \cdots + (x_{i+1} - x_i)| \\
&\leq |x_J - x_{J-1}| + |x_{J-1} - x_{J-2}| + \cdots + |x_{i+1} - x_i| \\
&\leq \gamma^{J-1}|x_1 - x_0| + \gamma^{J-2}|x_1 - x_0| + \cdots + \gamma^{i}|x_1 - x_0|
\end{aligned}$$

From the contraction property, we know that

$$|x_i - x_{i-1}| \leq \gamma |x_{i-1} - x_{i-2}|$$

for $i > 2$. Take a fixed $J > i$. We can expand $|x_J - x_i|$ as

$$\begin{aligned}
|x_J - x_i| &= |(x_J - x_{J-1}) + (x_{J-1} - x_{J-2}) + \cdots + (x_{i+1} - x_i)| \\
&\leq |x_J - x_{J-1}| + |x_{J-1} - x_{J-2}| + \cdots + |x_{i+1} - x_i| \\
&\leq \gamma^{J-1}|x_1 - x_0| + \gamma^{J-2}|x_1 - x_0| + \cdots + \gamma^{i}|x_1 - x_0| \\
&= \left(\gamma^{J-1} + \gamma^{J-2} + \cdots \gamma^{i}\right)|x_1 - x_0|
\end{aligned}$$

From the contraction property, we know that

$$|x_i - x_{i-1}| \le \gamma |x_{i-1} - x_{i-2}|$$

for $i > 2$. Take a fixed $J > i$. We can expand $|x_J - x_i|$ as

$$\begin{aligned}
|x_J - x_i| &= |(x_J - x_{J-1}) + (x_{J-1} - x_{J-2}) + \cdots + (x_{i+1} - x_i)| \\
&\le |x_J - x_{J-1}| + |x_{J-1} - x_{J-2}| + \cdots + |x_{i+1} - x_i| \\
&\le \gamma^{J-1}|x_1 - x_0| + \gamma^{J-2}|x_1 - x_0| + \cdots + \gamma^{i}|x_1 - x_0| \\
&= \left(\gamma^{J-1} + \gamma^{J-2} + \cdots \gamma^{i}\right)|x_1 - x_0| \\
&= \gamma^{i}\left(\gamma^{J-i-1} + \gamma^{J-i-2} + \cdots + \gamma + 1\right)|x_1 - x_0|.
\end{aligned}$$

From the contraction property, we know that

$$|x_i - x_{i-1}| \leq \gamma |x_{i-1} - x_{i-2}|$$

for $i > 2$. Take a fixed $J > i$. We can expand $|x_J - x_i|$ as

$$\begin{aligned}
|x_J - x_i| &= |(x_J - x_{J-1}) + (x_{J-1} - x_{J-2}) + \cdots + (x_{i+1} - x_i)| \\
&\leq |x_J - x_{J-1}| + |x_{J-1} - x_{J-2}| + \cdots + |x_{i+1} - x_i| \\
&\leq \gamma^{J-1}|x_1 - x_0| + \gamma^{J-2}|x_1 - x_0| + \cdots + \gamma^i|x_1 - x_0| \\
&= \left(\gamma^{J-1} + \gamma^{J-2} + \cdots \gamma^i\right)|x_1 - x_0| \\
&= \gamma^i\left(\gamma^{J-i-1} + \gamma^{J-i-2} + \cdots + \gamma + 1\right)|x_1 - x_0|.
\end{aligned}$$

In brackets we have the first few terms of the geometric series, which converges because $\gamma < 1$. Taking the limit $J \to \infty$, so $x_J \to x^\star$, we have

$$|x_i - x^\star| \leq \frac{\gamma^i}{1 - \gamma}|x_1 - x_0|.$$

From the contraction property, we know that

$$|x_i - x_{i-1}| \leq \gamma |x_{i-1} - x_{i-2}|$$

for $i > 2$. Take a fixed $J > i$. We can expand $|x_J - x_i|$ as

$$\begin{aligned}
|x_J - x_i| &= |(x_J - x_{J-1}) + (x_{J-1} - x_{J-2}) + \cdots + (x_{i+1} - x_i)| \\
&\leq |x_J - x_{J-1}| + |x_{J-1} - x_{J-2}| + \cdots + |x_{i+1} - x_i| \\
&\leq \gamma^{J-1}|x_1 - x_0| + \gamma^{J-2}|x_1 - x_0| + \cdots + \gamma^{i}|x_1 - x_0| \\
&= \left(\gamma^{J-1} + \gamma^{J-2} + \cdots \gamma^{i}\right)|x_1 - x_0| \\
&= \gamma^{i}\left(\gamma^{J-i-1} + \gamma^{J-i-2} + \cdots + \gamma + 1\right)|x_1 - x_0|.
\end{aligned}$$

In brackets we have the first few terms of the geometric series, which converges because $\gamma < 1$. Taking the limit $J \to \infty$, so $x_J \to x^\star$, we have

$$|x_i - x^\star| \leq \frac{\gamma^i}{1-\gamma}|x_1 - x_0|.$$

This is an *a posteriori* bound: you have to do some computation to use it.

Section 8

Another example

Example question

Find some $[a, b]$ so that $g(x) = e^{-x}$ has a unique fixed point in $[a, b]$.

Example question

Find some $[a, b]$ so that $g(x) = e^{-x}$ has a unique fixed point in $[a, b]$.

We need:

(i) $g : [a, b] \to [a, b]$, and

(ii) $|g'(x)| \leq \gamma < 1$ on $[a, b]$ for some γ.

Example question

Find some $[a, b]$ so that $g(x) = e^{-x}$ has a unique fixed point in $[a, b]$.

We need:

(i) $g : [a, b] \to [a, b]$, and

(ii) $|g'(x)| \leq \gamma < 1$ on $[a, b]$ for some γ.

So let's consider $g'(x)$. Calculating, we find $g'(x) = -e^{-x}$, so $|g'(x)| = |e^{-x}|$. This is 1 at $x = 0$ and strictly less than 1 for $x > 0$.

Example question

Find some $[a, b]$ so that $g(x) = e^{-x}$ has a unique fixed point in $[a, b]$.

We need:

(i) $g : [a, b] \to [a, b]$, and

(ii) $|g'(x)| \leq \gamma < 1$ on $[a, b]$ for some γ.

So let's consider $g'(x)$. Calculating, we find $g'(x) = -e^{-x}$, so $|g'(x)| = |e^{-x}|$. This is 1 at $x = 0$ and strictly less than 1 for $x > 0$.

Also note that $g(1) = e^{-1} < 1$, and $g(x)$ is decreasing, so $g : [0, 1] \to [0, 1]$.

Example question

Find some $[a, b]$ so that $g(x) = e^{-x}$ has a unique fixed point in $[a, b]$.

We need:

(i) $g : [a, b] \to [a, b]$, and

(ii) $|g'(x)| \le \gamma < 1$ on $[a, b]$ for some γ.

So let's consider $g'(x)$. Calculating, we find $g'(x) = -e^{-x}$, so $|g'(x)| = |e^{-x}|$. This is 1 at $x = 0$ and strictly less than 1 for $x > 0$.

Also note that $g(1) = e^{-1} < 1$, and $g(x)$ is decreasing, so $g : [0, 1] \to [0, 1]$.

We could thus take an interval with $a > 0$ but close and $b = 1$. Choosing $[a, b] = [1/10, 1]$ works fine. (The actual fixed point is $x^\star \approx 0.567143$.)

Continuing with the same example, how many iterations are required to get within 10^{-3} of the fixed point?

Continuing with the same example, how many iterations are required to get within 10^{-3} of the fixed point?

Our γ is $e^{-1/10} \approx 0.905$.

Continuing with the same example, how many iterations are required to get within 10^{-3} of the fixed point?

Our γ is $e^{-1/10} \approx 0.905$.

For the *a priori* bound, solving $\gamma^i < 0.001/0.9$ yields $i > 68$. (To achieve a tolerance of 10^{-6}, $i > 137$ is required.)

Continuing with the same example, how many iterations are required to get within 10^{-3} of the fixed point?

Our γ is $e^{-1/10} \approx 0.905$.

For the *a priori* bound, solving $\gamma^i < 0.001/0.9$ yields $i > 68$. (To achieve a tolerance of 10^{-6}, $i > 137$ is required.)

Let's imagine we start with a lucky guess $x_0 = 0.56$. How does the *a posteriori* bound look? In this case $x_1 \approx 0.57120906$, so we have

$$\frac{\gamma^i}{1-\gamma}|0.57120906 - 0.56| < \mathrm{tol},$$

which gives $i > 47$ for $\mathrm{tol} = 10^{-3}$ and $i > 116$ for $\mathrm{tol} = 10^{-6}$.

Section 9

Accelerating sequence convergence

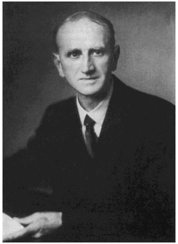
Suppose one has a sequence (x_i) that is linearly converging:

$$\lim_{i\to\infty} \frac{|x_{i+1} - x^\star|}{|x_i - x^\star|} = \mu,$$

with the property that for large enough i,

$$x_i - x^\star, \quad x_{i+1} - x^\star, x_{i+2} - x^\star$$

all have the same sign.



Alexander Aitken FRS FRSL, 1895–1967

Aitken's big idea: use the entries of (x_i) to make a new sequence $(\tilde{x}_i)$ that (hopefully) converges faster!

Assume that the asymptotic limits hold at iterations $i+1$, $i+2$, so that

$$x_{i+1} - x^\star \approx \mu(x_i - x^\star), \quad x_{i+2} - x^\star \approx \mu(x_{i+1} - x^\star).$$

Assume that the asymptotic limits hold at iterations $i+1$, $i+2$, so that

$$x_{i+1} - x^\star \approx \mu(x_i - x^\star), \quad x_{i+2} - x^\star \approx \mu(x_{i+1} - x^\star).$$

Equating the two expressions for μ and doing some algebra yields

$$x^\star \approx x_i - \frac{(x_{i+1} - x_i)^2}{x_{i+2} - 2x_{i+1} + x_i}$$

so we hope that the expression on the right gives a good approximation to the sequence limit.

Assume that the asymptotic limits hold at iterations $i+1$, $i+2$, so that

$$x_{i+1} - x^\star \approx \mu(x_i - x^\star), \quad x_{i+2} - x^\star \approx \mu(x_{i+1} - x^\star).$$

Equating the two expressions for μ and doing some algebra yields

$$x^\star \approx x_i - \frac{(x_{i+1} - x_i)^2}{x_{i+2} - 2x_{i+1} + x_i}$$

so we hope that the expression on the right gives a good approximation to the sequence limit.

Aitken thus defines

$$\tilde{x}_i = x_i - \frac{(x_{i+1} - x_i)^2}{x_{i+2} - 2x_{i+1} + x_i}$$

to yield a new, (hopefully) faster-converging sequence.

Aitken's acceleration is backed up by a theorem.

Aitken's theorem (1926)

Suppose (x_i) is linearly converging with all entries the same sign. Then

$$\lim_{i \to \infty} \frac{\tilde{x}_i - x^\star}{x_i - x^\star} = 0.$$

Aitken's acceleration is backed up by a theorem.

Aitken's theorem (1926)

Suppose (x_i) is linearly converging with all entries the same sign. Then

$$\lim_{i\to\infty} \frac{\tilde{x}_i - x^\star}{x_i - x^\star} = 0.$$

Consider Leibniz' formula for π:

$$\pi = 4\sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1}.$$

Set x_i to be the i^{th} partial sum.

To get π to 10 digits, Leibniz' formula requires about 5 billion terms; Aitken's acceleration $(\tilde{x}_i)$ of it requires about 1400.

Aitken's acceleration is backed up by a theorem.

Aitken's theorem (1926)

Suppose (x_i) is linearly converging with all entries the same sign. Then

$$\lim_{i\to\infty} \frac{\tilde{x}_i - x^\star}{x_i - x^\star} = 0.$$

Consider Leibniz' formula for π:

$$\pi = 4\sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1}.$$

Set x_i to be the i^{th} partial sum.

To get π to 10 digits, Leibniz' formula requires about 5 billion terms; Aitken's acceleration $(\tilde{x}_i)$ of it requires about 1400.

If you apply Aitken acceleration *again*, to yield $(\tilde{\tilde{x}}_i)$, you can get away with only 70 terms!

Computational Mathematics
Week 3: Newton's method

Patrick E. Farrell

University of Oxford

Let's consider rootfinding again:

$$\text{find } x^\star \in \mathbb{R} \text{ such that } f(x^\star) = 0.$$

Let's consider rootfinding again:

$$\text{find } x^\star \in \mathbb{R} \text{ such that } f(x^\star) = 0.$$

Since there are powerful theorems about fixed point problems, let's try to reformulate this as a fixed point problem:

$$\text{find } x^\star \in \mathbb{R} \text{ such that } x^\star = g(x^\star).$$

Let's consider rootfinding again:

$$\text{find } x^\star \in \mathbb{R} \text{ such that } f(x^\star) = 0.$$

Since there are powerful theorems about fixed point problems, let's try to reformulate this as a fixed point problem:

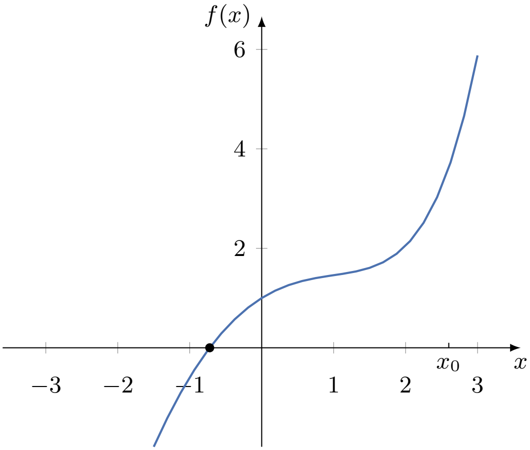
$$\text{find } x^\star \in \mathbb{R} \text{ such that } x^\star = g(x^\star).$$

How should we construct $g(x)$ from $f(x)$? One way we've seen is to set

$$g(x) = f(x) + x$$

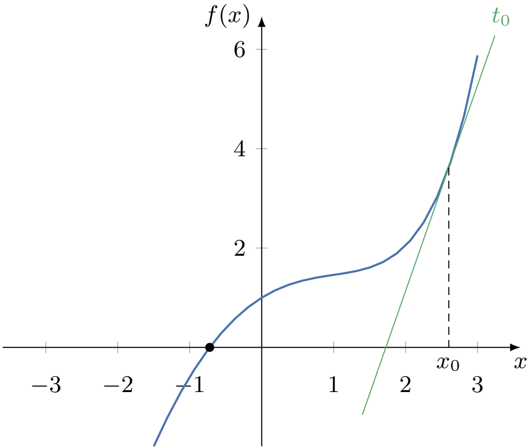
but we have no reason to think this is a contraction.

Here is a better way to construct $g(x)$.



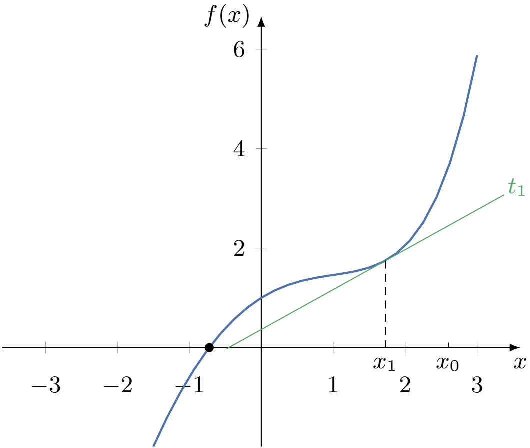
Start from an initial x_0.

Here is a better way to construct $g(x)$.



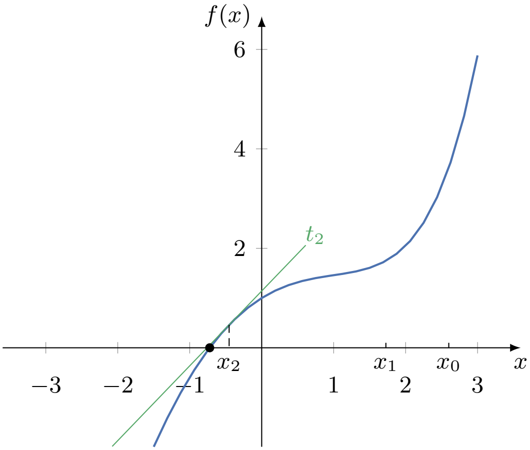
Build a *linear model* of the function.

Here is a better way to construct $g(x)$.



Set x_1 to be the root of the linear model.

Here is a better way to construct $g(x)$.



Repeat.

The tangent line joins $(x_i, f(x_i))$ and $(x_{i+1}, 0)$, so we can write its slope as

$$f'(x_i) = \frac{f(x_i) - 0}{x_i - x_{i+1}}$$

The tangent line joins $(x_i, f(x_i))$ and $(x_{i+1}, 0)$, so we can write its slope as

$$f'(x_i) = \frac{f(x_i) - 0}{x_i - x_{i+1}}$$

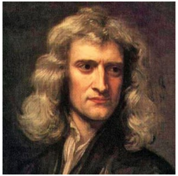
and solving for x_{i+1} yields

$$x_{i+1} = x_i - \left(f'(x_i)\right)^{-1} f(x_i).$$

Newton–Raphson method

$$x_{i+1} = g(x_i) \coloneqq x_i - \left(f'(x_i)\right)^{-1} f(x_i).$$

This is a generic way of constructing a fixed point problem $x = g(x)$ from a rootfinding problem $f(x) = 0$.

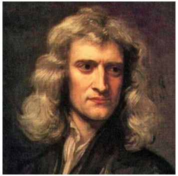


Isaac Newton FRS, 1643–1727

Newton–Raphson method

$$x_{i+1} = g(x_i) \coloneqq x_i - \left(f'(x_i)\right)^{-1} f(x_i).$$

This is a generic way of constructing a fixed point problem $x = g(x)$ from a rootfinding problem $f(x) = 0$.



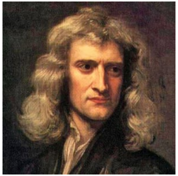
Isaac Newton FRS, 1643–1727

The special case of applying Newton's method for calculating square roots was known to the ancient Greeks in Alexandria (Heron's method, 60).

Newton–Raphson method

$$x_{i+1} = g(x_i) \coloneqq x_i - \left(f'(x_i)\right)^{-1} f(x_i).$$

This is a generic way of constructing a fixed point problem $x = g(x)$ from a rootfinding problem $f(x) = 0$.



Isaac Newton FRS, 1643–1727

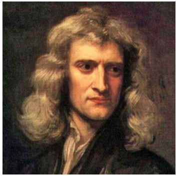
The special case of applying Newton's method for calculating square roots was known to the ancient Greeks in Alexandria (Heron's method, 60).

Taking $f(x) = x^2 - c$, we get

$$x_{i+1} = x_i - \frac{x_i^2 - c}{2x_i}$$

Newton–Raphson method

$$x_{i+1} = g(x_i) := x_i - \left(f'(x_i)\right)^{-1} f(x_i).$$



Isaac Newton FRS, 1643–1727

This is a generic way of constructing a fixed point problem $x = g(x)$ from a rootfinding problem $f(x) = 0$.

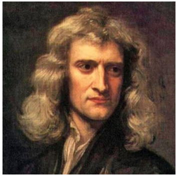
The special case of applying Newton's method for calculating square roots was known to the ancient Greeks in Alexandria (Heron's method, 60).

Taking $f(x) = x^2 - c$, we get

$$x_{i+1} = x_i - \frac{x_i^2 - c}{2x_i} = \frac{1}{2}\left(x_i + \frac{c}{x_i}\right).$$

Newton–Raphson method

$$x_{i+1} = g(x_i) \coloneqq x_i - \left(f'(x_i)\right)^{-1} f(x_i).$$



Isaac Newton FRS, 1643–1727

This is a generic way of constructing a fixed point problem $x = g(x)$ from a rootfinding problem $f(x) = 0$.

The special case of applying Newton's method for calculating square roots was known to the ancient Greeks in Alexandria (Heron's method, 60).

Taking $f(x) = x^2 - c$, we get

$$x_{i+1} = x_i - \frac{x_i^2 - c}{2x_i} = \frac{1}{2}\left(x_i + \frac{c}{x_i}\right).$$

The extension to computing p-th roots was known to Jamshīd al-Kāshī in Samarkand around 1427.

Isaac Newton (1669, published 1711) derived a complicated version of the method, only for polynomials. He didn't make the connection to calculus.

Isaac Newton (1669, published 1711) derived a complicated version of the method, only for polynomials. He didn't make the connection to calculus.

John Wallis (Savilian Chair of Geometry in Oxford) published the same method before Newton, in 1685. So we should probably call it the Wallis method!

John Wallis, 1616–1703

Isaac Newton (1669, published 1711) derived a complicated version of the method, only for polynomials. He didn't make the connection to calculus.

John Wallis (Savilian Chair of Geometry in Oxford) published the same method before Newton, in 1685. So we should probably call it the Wallis method!

John Wallis, 1616–1703

Joseph Raphson (1690) simplified the method, but still only applied it to polynomials.

Isaac Newton (1669, published 1711) derived a complicated version of the method, only for polynomials. He didn't make the connection to calculus.

John Wallis (Savilian Chair of Geometry in Oxford) published the same method before Newton, in 1685. So we should probably call it the Wallis method!

John Wallis, 1616–1703

Joseph Raphson (1690) simplified the method, but still only applied it to polynomials.

Thomas Simpson (1740) gave the modern description, using calculus, and applied it to general functions.

Newton–Raphson method

$$x_{i+1} = g(x_i) \coloneqq x_i - \left(f'(x_i)\right)^{-1} f(x_i).$$

Comments:

✓ If $f(x_i) = 0$, then $x_{i+1} = x_i$. So roots of f are fixed points of g.

Newton–Raphson method

$$x_{i+1} = g(x_i) \coloneqq x_i - \left(f'(x_i)\right)^{-1} f(x_i).$$

Comments:

- ✓ If $f(x_i) = 0$, then $x_{i+1} = x_i$. So roots of f are fixed points of g.
- ✗ Unlike bisection, we require f to be differentiable.

Newton–Raphson method

$$x_{i+1} = g(x_i) \coloneqq x_i - \left(f'(x_i)\right)^{-1} f(x_i).$$

Comments:

- ✓ If $f(x_i) = 0$, then $x_{i+1} = x_i$. So roots of f are fixed points of g.
- ✗ Unlike bisection, we require f to be differentiable.
- ✗ Moreover, we need $f'(x_i) \neq 0$ at every iterate.

Newton–Raphson method

$$x_{i+1} = g(x_i) \coloneqq x_i - \left(f'(x_i)\right)^{-1} f(x_i).$$

Comments:

- ✓ If $f(x_i) = 0$, then $x_{i+1} = x_i$. So roots of f are fixed points of g.
- ✗ Unlike bisection, we require f to be differentiable.
- ✗ Moreover, we need $f'(x_i) \neq 0$ at every iterate.
- ✓ If x_0 is close to $x^\star$, Newton's method usually converges *very fast*.

Newton–Raphson method

$$x_{i+1} = g(x_i) := x_i - \big(f'(x_i)\big)^{-1} f(x_i).$$

Comments:

- ✓ If $f(x_i) = 0$, then $x_{i+1} = x_i$. So roots of f are fixed points of g.
- ✗ Unlike bisection, we require f to be differentiable.
- ✗ Moreover, we need $f'(x_i) \neq 0$ at every iterate.
- ✓ If x_0 is close to $x^\star$, Newton's method usually converges *very fast*.
- ✗ If x_0 is far away, the method can diverge or get stuck in a cycle.

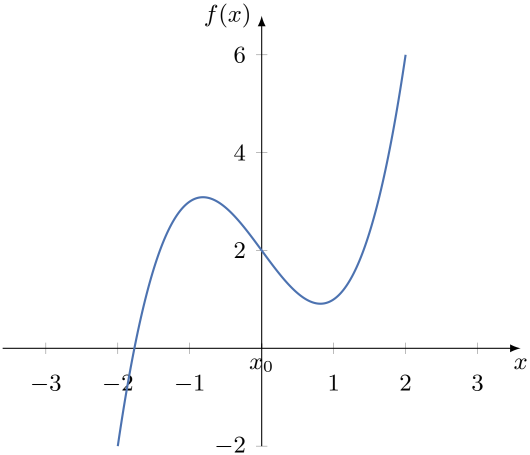
Newton–Raphson method

$$x_{i+1} = g(x_i) \coloneqq x_i - \big(f'(x_i)\big)^{-1} f(x_i).$$

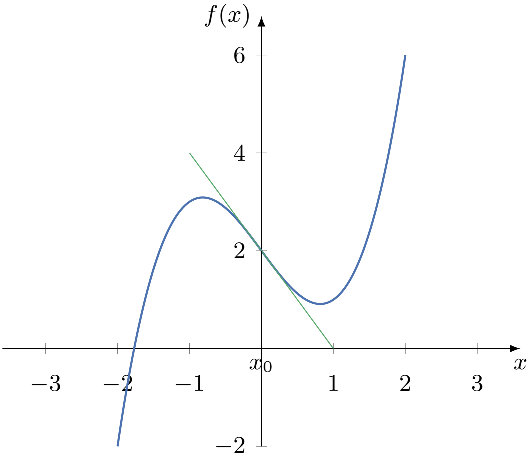
Comments:

- ✓ If $f(x_i) = 0$, then $x_{i+1} = x_i$. So roots of f are fixed points of g.
- ✗ Unlike bisection, we require f to be differentiable.
- ✗ Moreover, we need $f'(x_i) \neq 0$ at every iterate.
- ✓ If x_0 is close to $x^\star$, Newton's method usually converges *very fast*.
- ✗ If x_0 is far away, the method can diverge or get stuck in a cycle.
- ✓ Newton's method generalises elegantly to higher dimensions.

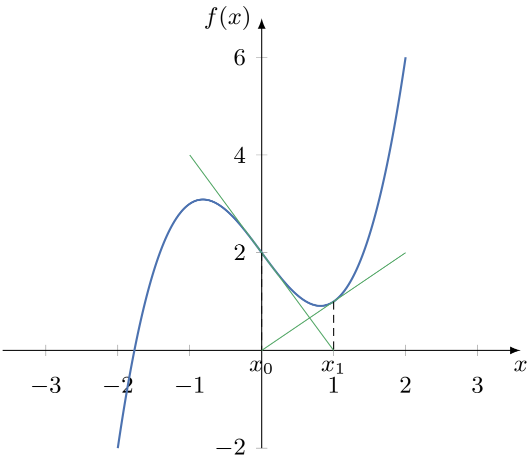
Consider $f(x) = x^3 - 2x + 2$ with $x_0 = 0$.



Consider $f(x) = x^3 - 2x + 2$ with $x_0 = 0$.



Consider $f(x) = x^3 - 2x + 2$ with $x_0 = 0$.



Even when it converges, Newton's method can behave in an unstable manner.

Even when it converges, Newton's method can behave in an unstable manner.

$$f(x) = (x-4)(x-1)(x+3), \quad x_0 = 2.352836327.$$

Even when it converges, Newton's method can behave in an unstable manner.

$$f(x) = (x-4)(x-1)(x+3), \quad x_0 = 2.352836327.$$

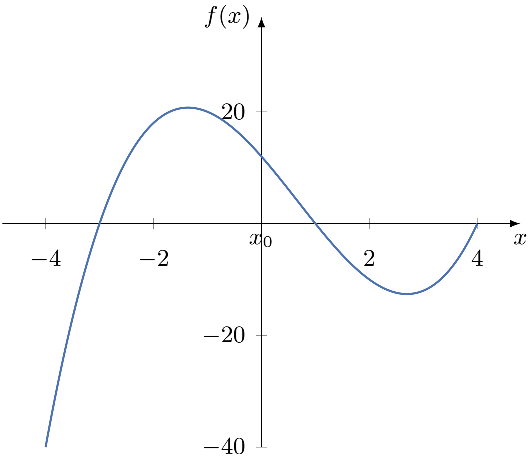
```
In [14]: newton(lambda x: (x-4)*(x-1)*(x+3),
                lambda x: 3*x**2 - 4*x - 11, 2.352836327, 1e-6)
Iteration  0: x = 2.352836e+00 f(x) = -1.192795e+01
Iteration  1: x = -7.829394e-01 f(x) = 1.890641e+01
Iteration  2: x = 2.352836e+00 f(x) = -1.192796e+01
Iteration  3: x = -7.829406e-01 f(x) = 1.890641e+01
...
Iteration  9: x = -8.476712e-01 f(x) = 1.927820e+01
Iteration 10: x = 2.687229e+00 f(x) = -1.259690e+01
Iteration 11: x = -1.449560e+02 f(x) = -3.086271e+06
Iteration 12: x = -9.643403e+01 f(x) = -9.143167e+05
...
Iteration 19: x = -5.622219e+00 f(x) = -1.670889e+02
Iteration 20: x = -4.050607e+00 f(x) = -4.271814e+01
Iteration 21: x = -3.265703e+00 f(x) = -8.235014e+00
Iteration 22: x = -3.023904e+00 f(x) = -6.756020e-01
Iteration 23: x = -3.000221e+00 f(x) = -6.196356e-03
Iteration 24: x = -3.000000e+00 f(x) = -5.385373e-07
Out[14]: -3.0000000192334735
```

Now change from $x_0 = 2.35283632\mathbf{7}$ to $x_0 = 2.35283632\mathbf{3}$.

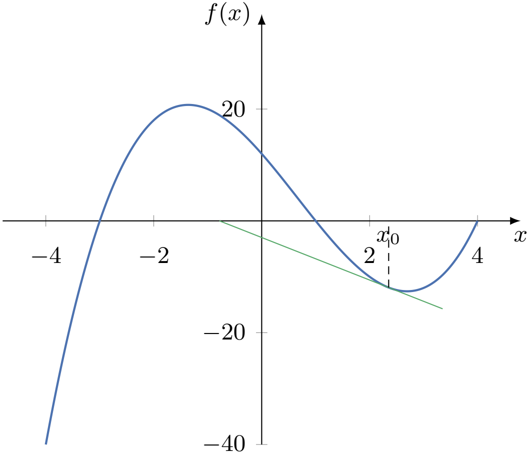
Now change from $x_0 = 2.35283632\mathbf{7}$ to $x_0 = 2.35283632\mathbf{3}$.

```
In [15]: newton(lambda x: (x-4)*(x-1)*(x+3),
                lambda x: 3*x**2 - 4*x - 11, 2.352836323, 1e-6)
Iteration  0: x = 2.352836e+00 f(x) = -1.192795e+01
Iteration  1: x = -7.829394e-01 f(x) = 1.890641e+01
Iteration  2: x = 2.352836e+00 f(x) = -1.192795e+01
Iteration  3: x = -7.829393e-01 f(x) = 1.890641e+01
Iteration  4: x = 2.352836e+00 f(x) = -1.192795e+01
Iteration  5: x = -7.829361e-01 f(x) = 1.890639e+01
Iteration  6: x = 2.352822e+00 f(x) = -1.192790e+01
Iteration  7: x = -7.828166e-01 f(x) = 1.890567e+01
Iteration  8: x = 2.352281e+00 f(x) = -1.192584e+01
Iteration  9: x = -7.783146e-01 f(x) = 1.887843e+01
Iteration 10: x = 2.332103e+00 f(x) = -1.184692e+01
Iteration 11: x = -6.205467e-01 f(x) = 1.781690e+01
Iteration 12: x = 1.799380e+00 f(x) = -8.442739e+00
Iteration 13: x = 8.042685e-01 f(x) = 2.379590e+00
Iteration 14: x = 9.981010e-01 f(x) = 2.279200e-02
Iteration 15: x = 9.999997e-01 f(x) = 3.591499e-06
Iteration 16: x = 1.000000e+00 f(x) = 8.926193e-14
Out[15]: 0.9999999999999926
```

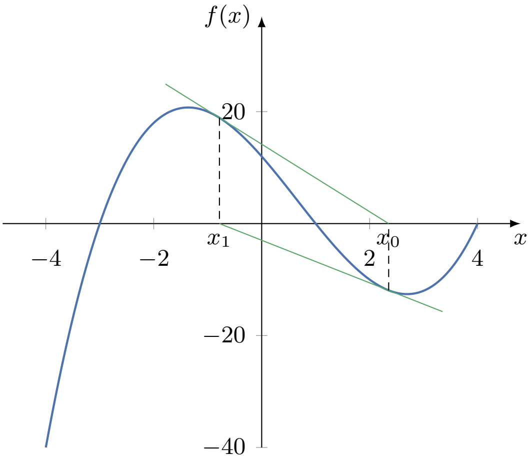
Let's draw the first iterations with $x_0 = 2.352836327$.



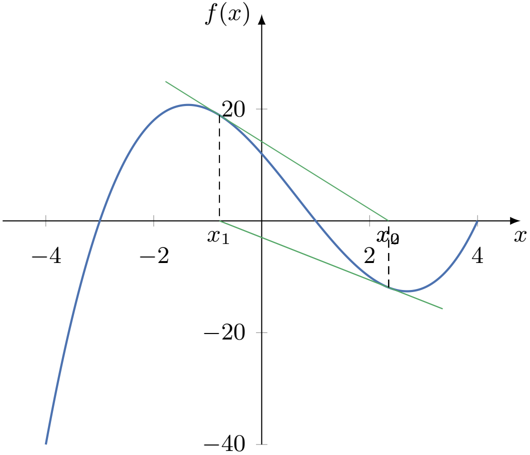
Let's draw the first iterations with $x_0 = 2.352836327$.



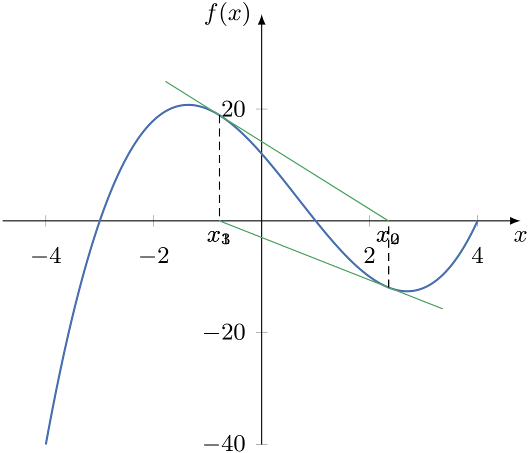
Let's draw the first iterations with $x_0 = 2.352836327$.



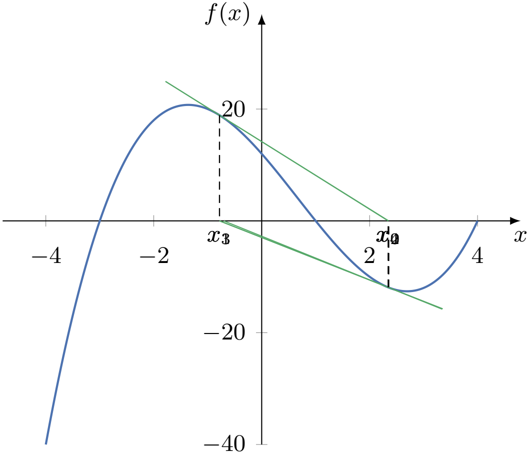
Let's draw the first iterations with $x_0 = 2.352836327$.



Let's draw the first iterations with $x_0 = 2.352836327$.



Let's draw the first iterations with $x_0 = 2.352836327$.



So why is Newton's method a good idea? Let's talk about general fixed point iterations $x_{i+1} = g(x_i)$ converging to $x^\star$ for a moment.

So why is Newton's method a good idea? Let's talk about general fixed point iterations $x_{i+1} = g(x_i)$ converging to $x^\star$ for a moment.

For a contraction g with contraction factor $\gamma < 1$, we know

$$|x_{i+1} - x^\star| \leq \gamma |x_i - x^\star|,$$

or in other words that we have linear convergence

$$\frac{|x_{i+1} - x^\star|}{|x_i - x^\star|} \leq \gamma < 1.$$

So why is Newton's method a good idea? Let's talk about general fixed point iterations $x_{i+1} = g(x_i)$ converging to $x^\star$ for a moment.

For a contraction g with contraction factor $\gamma < 1$, we know

$$|x_{i+1} - x^\star| \leq \gamma |x_i - x^\star|,$$

or in other words that we have linear convergence

$$\frac{|x_{i+1} - x^\star|}{|x_i - x^\star|} \leq \gamma < 1.$$

But Newton's method is special: under mild conditions, when x_i is close to $x^\star$ it will satisfy for some $K > 0$

$$\frac{|x_{i+1} - x^\star|}{|x_i - x^\star|^2} \leq K.$$

Recall that we called this *quadratic* convergence.

So why is Newton's method a good idea? Let's talk about general fixed point iterations $x_{i+1} = g(x_i)$ converging to $x^\star$ for a moment.

For a contraction g with contraction factor $\gamma < 1$, we know

$$|x_{i+1} - x^\star| \leq \gamma |x_i - x^\star|,$$

or in other words that we have linear convergence

$$\frac{|x_{i+1} - x^\star|}{|x_i - x^\star|} \leq \gamma < 1.$$

But Newton's method is special: under mild conditions, when x_i is close to $x^\star$ it will satisfy for some $K > 0$

$$\frac{|x_{i+1} - x^\star|}{|x_i - x^\star|^2} \leq K.$$

Recall that we called this *quadratic* convergence.

This is much, much faster: roughly speaking, the number of correct digits will *double* at each iteration!

So how can Newton achieve this quadratic convergence?

Recall the Taylor expansion of g around some point a:

$$g(x_i) = g(a) + (x_i - a)g'(a) + \frac{1}{2}(x_i - a)^2 g''(\zeta_i), \quad \text{some } \zeta_i \in (x_i, a).$$

So how can Newton achieve this quadratic convergence?

Recall the Taylor expansion of g around some point a:

$$g(x_i) = g(a) + (x_i - a)g'(a) + \frac{1}{2}(x_i - a)^2 g''(\zeta_i), \quad \text{some } \zeta_i \in (x_i, a).$$

What happens if we evaluate this around a fixed point $x^\star$ of g?

So how can Newton achieve this quadratic convergence?

Recall the Taylor expansion of g around some point a:

$$g(x_i) = g(a) + (x_i - a)g'(a) + \frac{1}{2}(x_i - a)^2 g''(\zeta_i), \quad \text{some } \zeta_i \in (x_i, a).$$

What happens if we evaluate this around a fixed point $x^\star$ of g?

$$g(x_i) = g(x^\star) + (x_i - x^\star)g'(x^\star) + \frac{1}{2}(x_i - x^\star)^2 g''(\zeta_i), \quad \text{some } \zeta_i \in (x_i, x^\star).$$

So how can Newton achieve this quadratic convergence?

Recall the Taylor expansion of g around some point a:

$$g(x_i) = g(a) + (x_i - a)g'(a) + \frac{1}{2}(x_i - a)^2 g''(\zeta_i), \quad \text{some } \zeta_i \in (x_i, a).$$

What happens if we evaluate this around a fixed point $x^\star$ of g?

$$g(x_i) = g(x^\star) + (x_i - x^\star)g'(x^\star) + \frac{1}{2}(x_i - x^\star)^2 g''(\zeta_i), \quad \text{some } \zeta_i \in (x_i, x^\star).$$

But $g(x_i) = x_{i+1}$ and $g(x^\star) = x^\star$, so

$$|x_{i+1} - x^\star| = |(x_i - x^\star)g'(x^\star) + \frac{1}{2}(x_i - x^\star)^2 g''(\zeta_i)|$$

So how can Newton achieve this quadratic convergence?

Recall the Taylor expansion of g around some point a:

$$g(x_i) = g(a) + (x_i - a)g'(a) + \frac{1}{2}(x_i - a)^2 g''(\zeta_i), \quad \text{some } \zeta_i \in (x_i, a).$$

What happens if we evaluate this around a fixed point $x^\star$ of g?

$$g(x_i) = g(x^\star) + (x_i - x^\star)g'(x^\star) + \frac{1}{2}(x_i - x^\star)^2 g''(\zeta_i), \quad \text{some } \zeta_i \in (x_i, x^\star).$$

But $g(x_i) = x_{i+1}$ and $g(x^\star) = x^\star$, so

$$\begin{aligned} |x_{i+1} - x^\star| &= \left|(x_i - x^\star)g'(x^\star) + \frac{1}{2}(x_i - x^\star)^2 g''(\zeta_i)\right| \\ &\leq |x_i - x^\star||g'(x^\star)| + \frac{1}{2}|x_i - x_\star|^2 \max_{s \in (x_i, x^\star)} |g''(s)|. \end{aligned}$$

So how can Newton achieve this quadratic convergence?

Recall the Taylor expansion of g around some point a:

$$g(x_i) = g(a) + (x_i - a)g'(a) + \frac{1}{2}(x_i - a)^2 g''(\zeta_i), \quad \text{some } \zeta_i \in (x_i, a).$$

What happens if we evaluate this around a fixed point $x^\star$ of g?

$$g(x_i) = g(x^\star) + (x_i - x^\star)g'(x^\star) + \frac{1}{2}(x_i - x^\star)^2 g''(\zeta_i), \quad \text{some } \zeta_i \in (x_i, x^\star).$$

But $g(x_i) = x_{i+1}$ and $g(x^\star) = x^\star$, so

$$\begin{aligned} |x_{i+1} - x^\star| &= |(x_i - x^\star)g'(x^\star) + \frac{1}{2}(x_i - x^\star)^2 g''(\zeta_i)| \\ &\leq |x_i - x^\star||g'(x^\star)| + \frac{1}{2}|x_i - x_\star|^2 \max_{s\in(x_i, x^\star)} |g''(s)|. \end{aligned}$$

If g has $g'(x^\star) = 0$, we would have quadratic convergence!

So how can Newton achieve this quadratic convergence?

Recall the Taylor expansion of g around some point a:

$$g(x_i) = g(a) + (x_i - a)g'(a) + \frac{1}{2}(x_i - a)^2 g''(\zeta_i), \quad \text{some } \zeta_i \in (x_i, a).$$

What happens if we evaluate this around a fixed point $x^\star$ of g?

$$g(x_i) = g(x^\star) + (x_i - x^\star)g'(x^\star) + \frac{1}{2}(x_i - x^\star)^2 g''(\zeta_i), \quad \text{some } \zeta_i \in (x_i, x^\star).$$

But $g(x_i) = x_{i+1}$ and $g(x^\star) = x^\star$, so

$$\begin{aligned} |x_{i+1} - x^\star| &= |(x_i - x^\star)g'(x^\star) + \frac{1}{2}(x_i - x^\star)^2 g''(\zeta_i)| \\ &\leq \frac{1}{2}|x_i - x_\star|^2 \max_{s \in (x_i, x^\star)} |g''(s)|. \end{aligned}$$

If g has $g'(x^\star) = 0$, we would have quadratic convergence!

Let's check when Newton's method does indeed satisfy $g'(x^\star) = 0$.

Let's check when Newton's method does indeed satisfy $g'(x^\star) = 0$.

Recall that

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

so (assuming $f \in C^2(\mathbb{R})$)

$$g'(x) = 1 - \left(\frac{[f'(x)]^2 - f(x)f''(x)}{[f'(x)]^2} \right)$$

Let's check when Newton's method does indeed satisfy $g'(x^\star) = 0$.

Recall that

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

so (assuming $f \in C^2(\mathbb{R})$)

$$\begin{aligned} g'(x) &= 1 - \left(\frac{[f'(x)]^2 - f(x)f''(x)}{[f'(x)]^2} \right) \\ &= \frac{f(x)f''(x)}{[f'(x)]^2}. \end{aligned}$$

Let's check when Newton's method does indeed satisfy $g'(x^\star) = 0$.

Recall that

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

so (assuming $f \in C^2(\mathbb{R})$)

$$\begin{aligned} g'(x) &= 1 - \left(\frac{[f'(x)]^2 - f(x)f''(x)}{[f'(x)]^2} \right) \\ &= \frac{f(x)f''(x)}{[f'(x)]^2}. \end{aligned}$$

If $f(x^\star) = 0$ and $f'(x^\star) \neq 0$, then $g'(x^\star) = 0$, and we do get quadratic convergence!

Let's check when Newton's method does indeed satisfy $g'(x^\star) = 0$.

Recall that

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

so (assuming $f \in C^2(\mathbb{R})$)

$$\begin{aligned} g'(x) &= 1 - \left(\frac{[f'(x)]^2 - f(x)f''(x)}{[f'(x)]^2} \right) \\ &= \frac{f(x)f''(x)}{[f'(x)]^2}. \end{aligned}$$

If $f(x^\star) = 0$ and $f'(x^\star) \neq 0$, then $g'(x^\star) = 0$, and we do get quadratic convergence!

If $f'(x^\star) = 0$, we have a multiple root, and we have to take the limit $x \to x^\star$ and use L'Hôpital's rule to evaluate the fraction.

Take-home message

Newton's method converges quadratically to isolated roots.

Take-home message

Newton's method converges quadratically to isolated roots.

If the root is not isolated, then one generally expects linear convergence, with the exact rate depending on details. For example, on the problem sheets you will prove that if

$$f'(x^\star) = 0, f''(x^\star) \neq 0$$

then one expects linear convergence with rate $1/2$.

Let's take an example. Let's look for the fixed point of $x = \cos x$. We tried this with bisection and it was slow.

The true answer is $x^\star \approx 0.739085133215161$.

Let's take an example. Let's look for the fixed point of $x = \cos x$. We tried this with bisection and it was slow.

The true answer is $x^\star \approx 0.739085133215161$.

Applying Newton's method to $f(x) := \cos x - x$ from $x_0 = 0$, we get

$$x_1 = 1 \qquad |x_1 - x^\star| = 2.6 \times 10^{-1}$$

Let's take an example. Let's look for the fixed point of $x = \cos x$. We tried this with bisection and it was slow.

The true answer is $x^\star \approx 0.739085133215161$.

Applying Newton's method to $f(x) := \cos x - x$ from $x_0 = 0$, we get

$$x_1 = 1 \qquad |x_1 - x^\star| = 2.6 \times 10^{-1}$$
$$x_2 = \underline{0.7}50363867840244 \qquad |x_2 - x^\star| = 1.1 \times 10^{-2}$$

Let's take an example. Let's look for the fixed point of $x = \cos x$. We tried this with bisection and it was slow.

The true answer is $x^\star \approx 0.739085133215161$.

Applying Newton's method to $f(x) := \cos x - x$ from $x_0 = 0$, we get

$$\begin{aligned} x_1 &= 1 & |x_1 - x^\star| &= 2.6 \times 10^{-1} \\ x_2 &= \underline{0.7}50363867840244 & |x_2 - x^\star| &= 1.1 \times 10^{-2} \\ x_3 &= \underline{0.739}112890911362 & |x_3 - x^\star| &= 2.8 \times 10^{-5} \end{aligned}$$

Let's take an example. Let's look for the fixed point of $x = \cos x$. We tried this with bisection and it was slow.

The true answer is $x^\star \approx 0.739085133215161$.

Applying Newton's method to $f(x) := \cos x - x$ from $x_0 = 0$, we get

$$\begin{aligned} x_1 &= 1 & |x_1 - x^\star| &= 2.6 \times 10^{-1} \\ x_2 &= \underline{0.7}50363867840244 & |x_2 - x^\star| &= 1.1 \times 10^{-2} \\ x_3 &= \underline{0.739}112890911362 & |x_3 - x^\star| &= 2.8 \times 10^{-5} \\ x_4 &= \underline{0.7390851333}85284 & |x_4 - x^\star| &= 1.7 \times 10^{-10} \end{aligned}$$

Let's take an example. Let's look for the fixed point of $x = \cos x$. We tried this with bisection and it was slow.

The true answer is $x^\star \approx 0.739085133215161$.

Applying Newton's method to $f(x) := \cos x - x$ from $x_0 = 0$, we get

$$
\begin{aligned}
x_1 &= 1 & |x_1 - x^\star| &= 2.6 \times 10^{-1} \\
x_2 &= \underline{0.7}50363867840244 & |x_2 - x^\star| &= 1.1 \times 10^{-2} \\
x_3 &= \underline{0.739}112890911362 & |x_3 - x^\star| &= 2.8 \times 10^{-5} \\
x_4 &= \underline{0.7390851333}85284 & |x_4 - x^\star| &= 1.7 \times 10^{-10} \\
x_5 &= \underline{0.739085133215161} = x_6 = \cdots .
\end{aligned}
$$

Let's do an exam question. Consider the question from 2017, Paper IV, Q7 (b):

> *The function*
>
> $$p(x) = 27x^3 - 27x^2 + 4$$
>
> *has a root* $\alpha = 2/3$.
>
> *Show that Newton's method to compute approximations to this root, with starting guess* x_0*, can be written as the iteration*
>
> $$x_{k+1} = g(x_k),$$
>
> *where you should find* g *explicitly. Prove or disprove that the sequence generated will converge to* α *for any* $x_0 \in [1/3, 1]$.

We write

$$g(x) = x - \frac{p(x)}{p'(x)}$$
$$= x - \frac{27x^3 - 27x^2 + 4}{81x^2 - 54x}$$

We write

$$\begin{aligned} g(x) &= x - \frac{p(x)}{p'(x)} \\ &= x - \frac{27x^3 - 27x^2 + 4}{81x^2 - 54x} \\ &= x - \frac{(3x-2)(9x^2 - 3x - 2)}{27x(3x-2)} \end{aligned}$$

We write

$$\begin{aligned} g(x) &= x - \frac{p(x)}{p'(x)} \\ &= x - \frac{27x^3 - 27x^2 + 4}{81x^2 - 54x} \\ &= x - \frac{(3x-2)(9x^2 - 3x - 2)}{27x(3x-2)} \\ &= \frac{2x}{3} + \frac{2}{27x} + \frac{1}{9}. \end{aligned}$$

We write

$$\begin{aligned} g(x) &= x - \frac{p(x)}{p'(x)} \\ &= x - \frac{27x^3 - 27x^2 + 4}{81x^2 - 54x} \\ &= x - \frac{(3x-2)(9x^2 - 3x - 2)}{27x(3x-2)} \\ &= \frac{2x}{3} + \frac{2}{27x} + \frac{1}{9}. \end{aligned}$$

To check whether the Newton sequence will converge, we investigate the conditions of Banach's contraction mapping theorem.

Let's check the conditions. We compute

$$g'(x) = \frac{2}{3} - \frac{2}{27x^2}, \quad g''(x) = \frac{4}{27x^3}.$$

Let's check the conditions. We compute

$$g'(x) = \frac{2}{3} - \frac{2}{27x^2}, \quad g''(x) = \frac{4}{27x^3}.$$

We see that $g''(x) > 0$ on $[1/3, 1]$ and hence $g'(x)$ is increasing on $[1/3, 1]$.

Let's check the conditions. We compute

$$g'(x) = \frac{2}{3} - \frac{2}{27x^2}, \quad g''(x) = \frac{4}{27x^3}.$$

We see that $g''(x) > 0$ on $[1/3, 1]$ and hence $g'(x)$ is increasing on $[1/3, 1]$. Evaluating at the endpoints,

$$g'(1/3) = 0, \quad g'(1) = 16/27 = \gamma.$$

Let's check the conditions. We compute

$$g'(x) = \frac{2}{3} - \frac{2}{27x^2}, \quad g''(x) = \frac{4}{27x^3}.$$

We see that $g''(x) > 0$ on $[1/3, 1]$ and hence $g'(x)$ is increasing on $[1/3, 1]$. Evaluating at the endpoints,

$$g'(1/3) = 0, \quad g'(1) = 16/27 = \gamma.$$

We now check that g is an endomorphism.

Let's check the conditions. We compute

$$g'(x) = \frac{2}{3} - \frac{2}{27x^2}, \quad g''(x) = \frac{4}{27x^3}.$$

We see that $g''(x) > 0$ on $[1/3, 1]$ and hence $g'(x)$ is increasing on $[1/3, 1]$. Evaluating at the endpoints,

$$g'(1/3) = 0, \quad g'(1) = 16/27 = \gamma.$$

We now check that g is an endomorphism.
Since $g'(x) \geq 0$ on $[1/3, 1]$, we know g is also increasing.

Let's check the conditions. We compute

$$g'(x) = \frac{2}{3} - \frac{2}{27x^2}, \quad g''(x) = \frac{4}{27x^3}.$$

We see that $g''(x) > 0$ on $[1/3, 1]$ and hence $g'(x)$ is increasing on $[1/3, 1]$. Evaluating at the endpoints,

$$g'(1/3) = 0, \quad g'(1) = 16/27 = \gamma.$$

We now check that g is an endomorphism.
Since $g'(x) \geq 0$ on $[1/3, 1]$, we know g is also increasing.

Checking at the endpoints,

$$g(1/3) = 5/9 \in [1/3, 1], \quad g(1) = 23/27 \in [1/3, 1].$$

Let's check the conditions. We compute

$$g'(x) = \frac{2}{3} - \frac{2}{27x^2}, \quad g''(x) = \frac{4}{27x^3}.$$

We see that $g''(x) > 0$ on $[1/3, 1]$ and hence $g'(x)$ is increasing on $[1/3, 1]$. Evaluating at the endpoints,

$$g'(1/3) = 0, \quad g'(1) = 16/27 = \gamma.$$

We now check that g is an endomorphism.
Since $g'(x) \geq 0$ on $[1/3, 1]$, we know g is also increasing.

Checking at the endpoints,

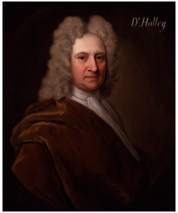
$$g(1/3) = 5/9 \in [1/3, 1], \quad g(1) = 23/27 \in [1/3, 1].$$

So the conditions of Banach's contraction mapping theorem are satisfied.

There are other fixed-point iterations for rootfinding.

Halley's method (1694)

$$x_{i+1} = g(x_i) := x_i - \frac{2f(x_i)f'(x_i)}{2[f'(x_i)]^2 - f(x_i)f''(x_i)}.$$

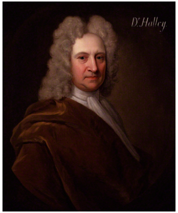


Edmund Halley FRS, 1656–1742

There are other fixed-point iterations for rootfinding.

Halley's method (1694)

$$x_{i+1} = g(x_i) \coloneqq x_i - \frac{2f(x_i)f'(x_i)}{2[f'(x_i)]^2 - f(x_i)f''(x_i)}.$$



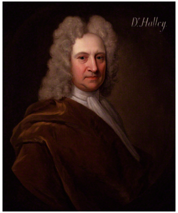
Edmund Halley FRS, 1656–1742

Halley was Savilian Professor of Geometry here in Oxford, after Wallis.

There are other fixed-point iterations for rootfinding.

Halley's method (1694)

$$x_{i+1} = g(x_i) := x_i - \frac{2f(x_i)f'(x_i)}{2[f'(x_i)]^2 - f(x_i)f''(x_i)}.$$



Edmund Halley FRS, 1656–1742

Halley was Savilian Professor of Geometry here in Oxford, after Wallis.

In a letter in 1712, Taylor wrote

> *While I was thinking of these things, I fell into a general method of applying Dr. Halley's Extraction of roots to all Problems ...And it is comprehended in this Theorem ....*

The theorem he proved was Taylor's theorem!

Brook Taylor FRS, 1685–1731

Section 2

The secant iteration

Halley's method uses more derivatives to get faster convergence.

Halley's method uses more derivatives to get faster convergence.

In practice, computing derivatives of your function might be very expensive. (Think of e.g. $f(x)$ as the evaluation of a climate model.)

Halley's method uses more derivatives to get faster convergence.

In practice, computing derivatives of your function might be very expensive. (Think of e.g. $f(x)$ as the evaluation of a climate model.)

The *secant* iteration makes the converse trade: no derivative evaluations, for (slightly) slower convergence.

The Newton iteration uses

$$x_{i+1} = g(x_i) = x - \left(f'(x_i)\right)^{-1} f(x_i)$$

but we don't want to code $f'(x)$.

The Newton iteration uses

$$x_{i+1} = g(x_i) = x - \left(f'(x_i)\right)^{-1} f(x_i)$$

but we don't want to code $f'(x)$.

The secant method approximates

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

with some previous data x_{i-1}.

The Newton iteration uses

$$x_{i+1} = g(x_i) = x - \left(f'(x_i)\right)^{-1} f(x_i)$$

but we don't want to code $f'(x)$.

The secant method approximates

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

with some previous data x_{i-1}.

This requires the user to supply both x_0 and x_{-1}.

The Newton iteration uses

$$x_{i+1} = g(x_i) = x - \left(f'(x_i)\right)^{-1} f(x_i)$$

but we don't want to code $f'(x)$.

The secant method approximates

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

with some previous data x_{i-1}.

This requires the user to supply both x_0 and x_{-1}.

Newton invented the secant method around the same time, but never published it.

The Newton iteration uses

$$x_{i+1} = g(x_i) = x - \left(f'(x_i)\right)^{-1} f(x_i)$$

but we don't want to code $f'(x)$.

The secant method approximates

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

with some previous data x_{i-1}.

This requires the user to supply both x_0 and x_{-1}.

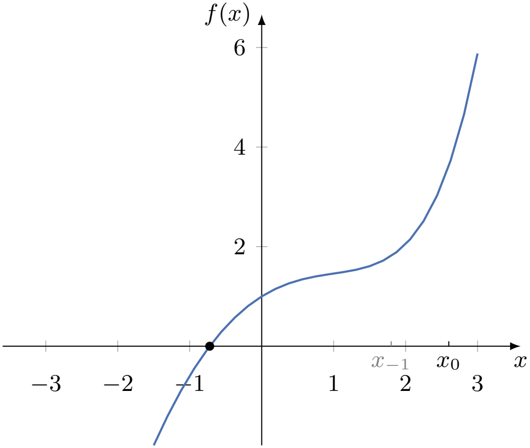
Newton invented the secant method around the same time, but never published it.

Both the ancient Egyptians and Babylonians used the secant method around 1800 BCE to solve equations like

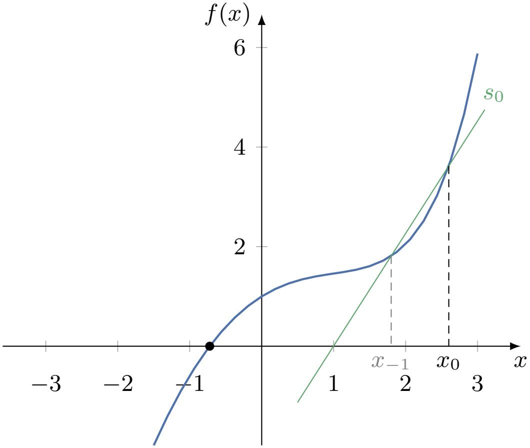
$$ax + b = c$$

since they didn't know how to move terms from one side to another!

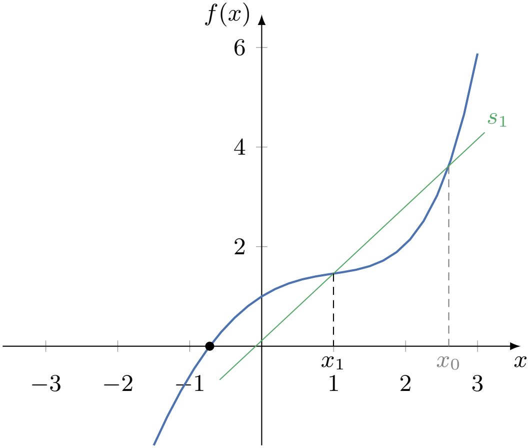
The secant iteration.



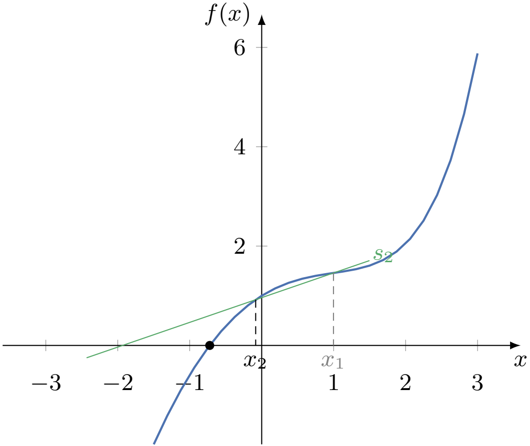
The secant iteration.



The secant iteration.



The secant iteration.



Interestingly, the secant iteration converges with order

$$\phi = \frac{1 + \sqrt{5}}{2} \approx 1.618034$$

so its convergence is superlinear, but not quite quadratic.

Interestingly, the secant iteration converges with order

$$\phi = \frac{1+\sqrt{5}}{2} \approx 1.618034$$

so its convergence is superlinear, but not quite quadratic.

The first proof to be found of this is by Terry Allen Jeeves in 1958, 300 years after Newton invented it!

Interestingly, the secant iteration converges with order

$$\phi = \frac{1+\sqrt{5}}{2} \approx 1.618034$$

so its convergence is superlinear, but not quite quadratic.

The first proof to be found of this is by Terry Allen Jeeves in 1958, 300 years after Newton invented it!

Comments on the secant method:

- ✗ The method requires more information to start, and depends sensitively on it.

Interestingly, the secant iteration converges with order

$$\phi = \frac{1+\sqrt{5}}{2} \approx 1.618034$$

so its convergence is superlinear, but not quite quadratic.

The first proof to be found of this is by Terry Allen Jeeves in 1958, 300 years after Newton invented it!

Comments on the secant method:

- ✗ The method requires more information to start, and depends sensitively on it.
- ✓ In principle the method can be applied to nondifferentiable functions.

Interestingly, the secant iteration converges with order

$$\phi = \frac{1+\sqrt{5}}{2} \approx 1.618034$$

so its convergence is superlinear, but not quite quadratic.

The first proof to be found of this is by Terry Allen Jeeves in 1958, 300 years after Newton invented it!

Comments on the secant method:

- ✗ The method requires more information to start, and depends sensitively on it.
- ✓ In principle the method can be applied to nondifferentiable functions.
- ▶ The generalisation to higher dimensions is different—leading to the quasi-Newton family of methods.

Section 3

Aitken acceleration of fixed-point iterations

Suppose our fixed-point iteration

$$x_{i+1} = g(x_i)$$

is only converging linearly.

Suppose our fixed-point iteration

$$x_{i+1} = g(x_i)$$

is only converging linearly.

We could apply Aitken acceleration, constructing

$$x_0, x_1, x_2, x_3, x_4, \ldots$$
$$\tilde{x}_0, \tilde{x}_1, \ldots.$$

Suppose our fixed-point iteration

$$x_{i+1} = g(x_i)$$

is only converging linearly.

We could apply Aitken acceleration, constructing

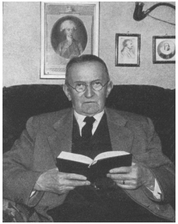
$$x_0, x_1, x_2, x_3, x_4, \ldots$$
$$\tilde{x}_0, \tilde{x}_1, \ldots.$$

The acceleration only goes one way: we don't re-use the accelerated values in the fixed-point iteration itself.

Steffensen's idea

Do two steps of fixed-point iteration, apply Aitken acceleration, then re-start the fixed-point iteration from there.

This *interleaves* the fixed-point iteration and acceleration.



Johan Frederik Steffensen, 1873–1961

Assume $g : [a, b] \to [a, b]$, and $x_0 \in [a, b]$.

Assume $g : [a, b] \to [a, b]$, and $x_0 \in [a, b]$.

---

function steffensen(g, x_0, tol)
  $x \leftarrow x_0$
  while $|g(x) - x| > \mathrm{tol}$ **do**
    $x_0 \leftarrow x$
    $x_1 \leftarrow g(x_0)$
    $x_2 \leftarrow g(x_1)$
    $x \leftarrow \left(x_0 x_2 - x_1^2\right) / (x_2 - 2x_1 + x_0)$
  end while
  return $g(x)$
end function

Assume $g : [a, b] \to [a, b]$, and $x_0 \in [a, b]$.

---

function steffensen(g, x_0, tol)
  $x \leftarrow x_0$
  while $|g(x) - x| > \mathrm{tol}$ **do**
    $x_0 \leftarrow x$
    $x_1 \leftarrow g(x_0)$
    $x_2 \leftarrow g(x_1)$
    $x \leftarrow \left(x_0 x_2 - x_1^2\right)/(x_2 - 2x_1 + x_0)$
  end while
  return $g(x)$
end function

If you organise the code properly, this requires two evaluations of g per iteration.

Assume $g : [a,b] \to [a,b]$, and $x_0 \in [a,b]$.

---

function steffensen(g, x_0, tol)
  $x \leftarrow x_0$
  while $|g(x) - x| > \mathrm{tol}$ **do**
    $x_0 \leftarrow x$
    $x_1 \leftarrow g(x_0)$
    $x_2 \leftarrow g(x_1)$
    $x \leftarrow \left(x_0 x_2 - x_1^2\right)/(x_2 - 2x_1 + x_0)$
  end while
  return $g(x)$
end function

If you organise the code properly, this requires two evaluations of g per iteration.

Does this really help?

Yes, it does, under certain conditions:

Steffensen's theorem (1933)

Suppose that $g(x)$ has a fixed point $x^\star$ with $g'(x^\star) \neq 1$. If there exists $\delta > 0$ such that $g \in C^3([x^\star - \delta, x^\star + \delta], \mathbb{R})$, then Steffensen's method gives quadratic convergence for any $x_0 \in [x^\star - \delta, x^\star + \delta]$.

This can achieve quadratic convergence, without derivatives!

Let's see two examples.

Let's see two examples.

We previously considered the fixed-point iteration

$$g(x) = \frac{x+1}{x}$$

for calculating the golden ratio ϕ.

Let's see two examples.

We previously considered the fixed-point iteration

$$g(x) = \frac{x+1}{x}$$

for calculating the golden ratio ϕ.

Fixed-point iteration requires 37 evaluations of g to get ϕ to 16 digits. Steffensen's method requires only 8!

Let's apply Newton's method to

$$f(x) = (x - 1)^2.$$

Let's apply Newton's method to

$$f(x) = (x-1)^2.$$

This gives

$$g(x) = x - \frac{(x-1)^2}{2x-2}.$$

Since $f'(1) = 0$, $g'(1) \neq 0$, and we only achieve linear convergence:

Let's apply Newton's method to

$$f(x) = (x-1)^2.$$

This gives

$$g(x) = x - \frac{(x-1)^2}{2x-2}.$$

Since $f'(1) = 0$, $g'(1) \neq 0$, and we only achieve linear convergence:

```
In [17]: newton(lambda x: (x-1)**2, lambda x: 2*x - 2, 0, 1e-4)
Iteration  0: x = 0.000000e+00 f(x) = 1.000000e+00
Iteration  1: x = 5.000000e-01 f(x) = 2.500000e-01
Iteration  2: x = 7.500000e-01 f(x) = 6.250000e-02
Iteration  3: x = 8.750000e-01 f(x) = 1.562500e-02
Iteration  4: x = 9.375000e-01 f(x) = 3.906250e-03
Iteration  5: x = 9.687500e-01 f(x) = 9.765625e-04
Iteration  6: x = 9.843750e-01 f(x) = 2.441406e-04
Iteration  7: x = 9.921875e-01 f(x) = 6.103516e-05
Out[17]: 0.9921875
```

Converging linearly, you say?

```
In [19]: steffensen(lambda x: x - (x-1)**2/(2*x-2), 2, 1e-12, exact=1)
Iterations  0: fixed point = 2.000000000000000e+00 error = 1.000000000000000e+00
Iterations  2: fixed point = 1.000000000000000e+00 error = 0.000000000000000e+00
```

Steffensen's method gets the answer exact to 16 digits in 2 iterations.

Section 4

Rootfinding for polynomials

We have seen general rootfinding methods that apply to many different kinds of functions.

We have seen general rootfinding methods that apply to many different kinds of functions.

Philosophical remark

When designing algorithms, we should always ask: have we used every piece of knowledge we have about the problem?

We have seen general rootfinding methods that apply to many different kinds of functions.

Philosophical remark

When designing algorithms, we should always ask: have we used every piece of knowledge we have about the problem?

For example, if we restrict ourselves to rootfinding for *polynomials*, can we make our algorithms better? The answer is yes.

Section 5

Horner's method

In the literature, Horner's method refers to two different things:

1. an efficient evaluation strategy for polynomials in the monomial basis;

In the literature, Horner's method refers to two different things:

1. an efficient evaluation strategy for polynomials in the monomial basis;
2. an iteration scheme for finding the roots of polynomials that combines Newton's method with the evaluation scheme.

In the literature, Horner's method refers to two different things:

1. an efficient evaluation strategy for polynomials in the monomial basis;
2. an iteration scheme for finding the roots of polynomials that combines Newton's method with the evaluation scheme.

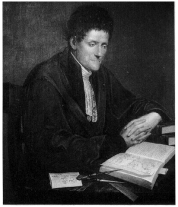
The evaluation scheme was known in medieval times to Qín Jiŭsháo (c. 1202–1261) and Sharaf al-Dīn al-Ṭūsī (c. 1135-1213), and later to Newton and Lagrange.

In the literature, Horner's method refers to two different things:

1. an efficient evaluation strategy for polynomials in the monomial basis;
2. an iteration scheme for finding the roots of polynomials that combines Newton's method with the evaluation scheme.

The evaluation scheme was known in medieval times to Qín Jiǔsháo (c. 1202–1261) and Sharaf al-Dīn al-Ṭūsī (c. 1135-1213), and later to Newton and Lagrange.

It's not clear that Horner, a schoolmaster in Bath, even invented the latter method that now bears his name. He was beaten to it by Paolo Ruffini in 1804 and Theophilus Holdred, a London watchmaker, in 1820. The method was published again by Horner in 1830.



Paolo Ruffini, 1765–1822

Let's consider Horner's two methods in order. Suppose we have a polynomial

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

with n large, e.g. $n = 10,000$. How should we evaluate $p(r)$ for $r \in \mathbb{R}$?

Let's consider Horner's two methods in order. Suppose we have a polynomial

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

with n large, e.g. $n = 10,000$. How should we evaluate $p(r)$ for $r \in \mathbb{R}$?

One way would be to evaluate all the terms in the sum separately, and add them up. This would require n additions and

$$0 + 1 + 2 + \cdots + n = \frac{n^2 + n}{2}$$

multiplications. Scaling like n^2 is bad!

Instead, a better way is to write

$$a_0 + a_1 x + \cdots + a_n x^n$$

Instead, a better way is to write

$$a_0 + a_1 x + \cdots + a_n x^n = a_0 +$$

Instead, a better way is to write

$$a_0 + a_1 x + \cdots + a_n x^n = a_0 + x\,($$

Instead, a better way is to write

$$a_0 + a_1 x + \cdots + a_n x^n = a_0 + x \left(a_1 + x\right.$$

Instead, a better way is to write

$$a_0 + a_1 x + \cdots + a_n x^n = a_0 + x\left(a_1 + x\left(a_2 + \cdots + x\left(a_{n-1} + x a_n\right)\cdots\right)\right).$$

Instead, a better way is to write

$$a_0 + a_1 x + \cdots + a_n x^n = a_0 + x\left(a_1 + x\left(a_2 + \cdots + x\left(a_{n-1} + x a_n\right)\cdots\right)\right).$$

This shares the evaluations of powers of x. It only requires n multiplications and n additions!

Instead, a better way is to write

$$a_0 + a_1x + \cdots + a_nx^n = a_0 + x\left(a_1 + x\left(a_2 + \cdots + x\left(a_{n-1} + xa_n\right)\cdots\right)\right).$$

This shares the evaluations of powers of x. It only requires n multiplications and n additions!

Algorithmically, to evaluate $p(r)$ for given $r \in \mathbb{R}$ we calculate

$$b_n := a_n$$

Instead, a better way is to write

$$a_0 + a_1 x + \cdots + a_n x^n = a_0 + x\left(a_1 + x\left(a_2 + \cdots + x\left(a_{n-1} + x a_n\right)\cdots\right)\right).$$

This shares the evaluations of powers of x. It only requires n multiplications and n additions!

Algorithmically, to evaluate $p(r)$ for given $r \in \mathbb{R}$ we calculate

$$\begin{aligned} b_n &:= a_n \\ b_{n-1} &:= a_{n-1} + b_n r \end{aligned}$$

Instead, a better way is to write

$$a_0 + a_1 x + \cdots + a_n x^n = a_0 + x\left(a_1 + x\left(a_2 + \cdots + x\left(a_{n-1} + x a_n\right)\cdots\right)\right).$$

This shares the evaluations of powers of x. It only requires n multiplications and n additions!

Algorithmically, to evaluate $p(r)$ for given $r \in \mathbb{R}$ we calculate

$$\begin{aligned} b_n &:= a_n \\ b_{n-1} &:= a_{n-1} + b_n r \\ &\vdots \\ b_i &:= a_i + b_{i+1} r \end{aligned}$$

Instead, a better way is to write

$$a_0 + a_1 x + \cdots + a_n x^n = a_0 + x\left(a_1 + x\left(a_2 + \cdots + x\left(a_{n-1} + x a_n\right)\cdots\right)\right).$$

This shares the evaluations of powers of x. It only requires n multiplications and n additions!

Algorithmically, to evaluate $p(r)$ for given $r \in \mathbb{R}$ we calculate

$$\begin{aligned}
b_n &:= a_n \\
b_{n-1} &:= a_{n-1} + b_n r \\
&\vdots \\
b_i &:= a_i + b_{i+1} r \\
&\vdots \\
b_1 &:= a_1 + b_2 r
\end{aligned}$$

Instead, a better way is to write

$$a_0 + a_1 x + \cdots + a_n x^n = a_0 + x\left(a_1 + x\left(a_2 + \cdots + x\left(a_{n-1} + x a_n\right)\cdots\right)\right).$$

This shares the evaluations of powers of x. It only requires n multiplications and n additions!

Algorithmically, to evaluate $p(r)$ for given $r \in \mathbb{R}$ we calculate

$$\begin{aligned}
b_n &:= a_n \\
b_{n-1} &:= a_{n-1} + b_n r \\
&\vdots \\
b_i &:= a_i + b_{i+1} r \\
&\vdots \\
b_1 &:= a_1 + b_2 r \\
b_0 &:= a_0 + b_1 r.
\end{aligned}$$

We then have $b_0 = p(r)$.

There's more to it than this, however.

There's more to it than this, however.

Theorem

Define the polynomial

$$Q(x) := b_n x^{n-1} + b_{n-1}x^{n-2} + \cdots + b_2 x + b_1.$$

Then

$$p(x) = (x - r)Q(x) + b_0.$$

There's more to it than this, however.

Theorem

Define the polynomial

$$Q(x) := b_n x^{n-1} + b_{n-1} x^{n-2} + \cdots + b_2 x + b_1.$$

Then

$$p(x) = (x - r)Q(x) + b_0.$$

Before proving this, note that indeed $p(r) = b_0$, and

$$p'(x) = Q(x) + (x - r)Q'(x),$$

There's more to it than this, however.

Theorem

Define the polynomial

$$Q(x) := b_n x^{n-1} + b_{n-1} x^{n-2} + \cdots + b_2 x + b_1.$$

Then

$$p(x) = (x - r)Q(x) + b_0.$$

Before proving this, note that indeed $p(r) = b_0$, and

$$p'(x) = Q(x) + (x - r)Q'(x),$$

so in particular

$$p'(r) = Q(r).$$

Proof.

Recall that $p(x) = a_0 + \cdots + a_n x^n$, $b_n = a_n$, and $b_i = a_i + b_{i+1} r$.

Expand

$$(x - r)Q(x) + b_0 =$$

Proof.

Recall that $p(x) = a_0 + \cdots + a_n x^n$, $b_n = a_n$, and $b_i = a_i + b_{i+1} r$.

Expand

$$(x - r)Q(x) + b_0 = (x - r)(b_n x^{n-1} + \cdots + b_1) + b_0$$

Proof.

Recall that $p(x) = a_0 + \cdots + a_n x^n$, $b_n = a_n$, and $b_i = a_i + b_{i+1} r$.

Expand

$$\begin{aligned}(x-r)Q(x) + b_0 &= (x-r)(b_n x^{n-1} + \cdots + b_1) + b_0 \\ &= x(b_n x^{n-1} + \cdots + b_1) - r(b_n x^{n-1} + \cdots + b_1) + b_0\end{aligned}$$

Proof.

Recall that $p(x) = a_0 + \cdots + a_n x^n$, $b_n = a_n$, and $b_i = a_i + b_{i+1} r$.

Expand

$$
\begin{aligned}
(x-r)Q(x) + b_0 &= (x-r)(b_n x^{n-1} + \cdots + b_1) + b_0 \\
&= x(b_n x^{n-1} + \cdots + b_1) - r(b_n x^{n-1} + \cdots + b_1) + b_0 \\
&= b_n x^n + (b_{n-1} - b_n r)x^{n-1} + \cdots + (b_0 - b_1 r)
\end{aligned}
$$

Proof.

Recall that $p(x) = a_0 + \cdots + a_n x^n$, $b_n = a_n$, and $b_i = a_i + b_{i+1} r$.

Expand

$$\begin{aligned}(x-r)Q(x) + b_0 &= (x-r)(b_n x^{n-1} + \cdots + b_1) + b_0 \\ &= x(b_n x^{n-1} + \cdots + b_1) - r(b_n x^{n-1} + \cdots + b_1) + b_0 \\ &= b_n x^n + (b_{n-1} - b_n r)x^{n-1} + \cdots + (b_0 - b_1 r) \\ &= a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0,\end{aligned}$$

since $a_i = b_i - b_{i+1} r$ for $i < n$. □

Proof.

Recall that $p(x) = a_0 + \cdots + a_n x^n$, $b_n = a_n$, and $b_i = a_i + b_{i+1} r$.

Expand

$$\begin{aligned}(x-r)Q(x) + b_0 &= (x-r)(b_n x^{n-1} + \cdots + b_1) + b_0 \\ &= x(b_n x^{n-1} + \cdots + b_1) - r(b_n x^{n-1} + \cdots + b_1) + b_0 \\ &= b_n x^n + (b_{n-1} - b_n r) x^{n-1} + \cdots + (b_0 - b_1 r) \\ &= a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0,\end{aligned}$$

since $a_i = b_i - b_{i+1} r$ for $i < n$. □

In the context of Newton's method applied to p, we have

$$x_{i+1} = x_i - \frac{p(x_i)}{p'(x_i)}$$

Proof.

Recall that $p(x) = a_0 + \cdots + a_n x^n$, $b_n = a_n$, and $b_i = a_i + b_{i+1} r$.

Expand

$$\begin{aligned}(x-r)Q(x) + b_0 &= (x-r)(b_n x^{n-1} + \cdots + b_1) + b_0 \\ &= x(b_n x^{n-1} + \cdots + b_1) - r(b_n x^{n-1} + \cdots + b_1) + b_0 \\ &= b_n x^n + (b_{n-1} - b_n r)x^{n-1} + \cdots + (b_0 - b_1 r) \\ &= a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0,\end{aligned}$$

since $a_i = b_i - b_{i+1} r$ for $i < n$. □

In the context of Newton's method applied to p, we have

$$x_{i+1} = x_i - \frac{p(x_i)}{p'(x_i)} = x_i - \frac{p(x_i)}{Q(x_i)}.$$

function horner($[a_0, \cdots, a_n]$, x_0, tol, maxit)

$x \leftarrow x_0$

function horner($[a_0, \cdots, a_n]$, x_0, tol, maxit)

$x \leftarrow x_0$

for $i = 1, \ldots, \text{maxit}$ **do**

$b \leftarrow a_n x + a_{n-1}$ # Horner eval for p

```
function horner([a_0, ..., a_n], x_0, tol, maxit)
    x ← x_0
    for i = 1, ..., maxit do
        b ← a_n x + a_{n-1}                      # Horner eval for p
        c ← a_n                                  # Horner eval for p'
```

function horner($[a_0, \cdots, a_n]$, x_0, tol, maxit)

$x \leftarrow x_0$

for $i = 1, \ldots, \mathrm{maxit}$ **do**

$b \leftarrow a_n x + a_{n-1}$ # Horner eval for p

$c \leftarrow a_n$ # Horner eval for p'

```
function horner([a_0, ..., a_n], x_0, tol, maxit)
    x ← x_0
    for i = 1, ..., maxit do
        b ← a_n x + a_{n-1}                          # Horner eval for p
        c ← a_n                                      # Horner eval for p'
        for k = n-1, n-2, ..., 1, 0 do
            c ← cx + b
```

```
function horner([a_0, ..., a_n], x_0, tol, maxit)
    x ← x_0
    for i = 1, ..., maxit do
        b ← a_n x + a_{n-1}                      # Horner eval for p
        c ← a_n                                  # Horner eval for p'
        for k = n-1, n-2, ..., 1, 0 do
            c ← cx + b
            b ← bx + a_i
        end for
```

function horner($[a_0, \cdots, a_n]$, x_0, tol, maxit)

  $x \leftarrow x_0$

  for $i = 1, \ldots, \text{maxit}$ **do**

    $b \leftarrow a_n x + a_{n-1}$ # Horner eval for p

    $c \leftarrow a_n$ # Horner eval for p'

    for $k = n-1, n-2, \ldots, 1, 0$ **do**

      $c \leftarrow cx + b$

      $b \leftarrow bx + a_i$

    end for

    if $|b| < \text{tol}$ **then** # success

      return x

    end if

```
function horner([a_0, ..., a_n], x_0, tol, maxit)
    x ← x_0
    for i = 1, ..., maxit do
        b ← a_n x + a_{n-1}                          # Horner eval for p
        c ← a_n                                      # Horner eval for p'
        for k = n-1, n-2, ..., 1, 0 do
            c ← cx + b
            b ← bx + a_i
        end for
        if |b| < tol then                            # success
            return x
        end if
        x ← x - b/c                                  # Newton update
    end for
end function
```

We can summarise with the following useful notation:

Definition (Big $\mathcal{O}$ notation)

For $g(n) > 0$, we say

$$f(n) = \mathcal{O}(g(n)) \text{ as } n \to \infty$$

if there exists $M > 0$ and $n_0 \in \mathbb{N}$ such that

$$|f(n)| \leq M g(n) \text{ for all } n \geq n_0.$$

We can summarise with the following useful notation:

Definition (Big $\mathcal{O}$ notation)

For $g(n) > 0$, we say

$$f(n) = \mathcal{O}(g(n)) \text{ as } n \to \infty$$

if there exists $M > 0$ and $n_0 \in \mathbb{N}$ such that

$$|f(n)| \leq Mg(n) \text{ for all } n \geq n_0.$$

The number of operations to evaluate a degree-n polynomial is:

- $\mathcal{O}(n^2)$ for the naïve way, but

We can summarise with the following useful notation:

Definition (Big $\mathcal{O}$ notation)

For $g(n) > 0$, we say
$$f(n) = \mathcal{O}(g(n)) \text{ as } n \to \infty$$
if there exists $M > 0$ and $n_0 \in \mathbb{N}$ such that
$$|f(n)| \leq Mg(n) \text{ for all } n \geq n_0.$$

The number of operations to evaluate a degree-n polynomial is:

- $\mathcal{O}(n^2)$ for the naïve way, but
- $\mathcal{O}(n)$ for Horner's evaluation scheme.

This is much, much better at high n!

In fact, Horner's scheme for evaluation has a nice optimality property:

Theorem

Any algorithm for evaluating an arbitrary polynomial must require at least n additions (Ostrowski, 1954) and at least n multiplications (Pan, 1966).

In fact, Horner's scheme for evaluation has a nice optimality property:

Theorem

Any algorithm for evaluating an arbitrary polynomial must require at least n additions (Ostrowski, 1954) and at least n multiplications (Pan, 1966).

Since Horner's scheme employs n additions and n multiplications, it is optimal (for arbitrary polynomials).

In fact, Horner's scheme for evaluation has a nice optimality property:

Theorem

Any algorithm for evaluating an arbitrary polynomial must require at least n additions (Ostrowski, 1954) and at least n multiplications (Pan, 1966).

Since Horner's scheme employs n additions and n multiplications, it is optimal (for arbitrary polynomials).

If you know you'll evaluate a polynomial many times on different inputs, it is possible to preprocess the polynomial into a representation that requires fewer operations (trading offline work for online work).

Section 6

More philosophical remarks

Horner's evaluation scheme exhibits an important principle:

Horner's evaluation scheme exhibits an important principle:

Philosophical remark

Equivalent expressions can have different algorithmic properties!

Horner's evaluation scheme exhibits an important principle:

Philosophical remark

Equivalent expressions can have different algorithmic properties!

In Horner's case, we had

$$a_0 + a_1 x + \cdots + a_n x^n = a_0 + x\left(a_1 + x\left(a_2 + \cdots + x\left(a_{n-1} + x a_n\right)\cdots\right)\right).$$

Horner's evaluation scheme exhibits an important principle:

Philosophical remark

Equivalent expressions can have different algorithmic properties!

In Horner's case, we had

$$a_0 + a_1 x + \cdots + a_n x^n = a_0 + x\left(a_1 + x\left(a_2 + \cdots + x\left(a_{n-1} + x a_n\right)\cdots\right)\right).$$

Algorithmic advances sometimes come by deriving an equivalent expression with better properties.

Think back to our list of questions we ask about algorithms:

- Does the algorithm terminate?
- Does the algorithm give the correct answer?
- How fast does the algorithm converge to the answer?
- How many operations does it take?

Think back to our list of questions we ask about algorithms:

- Does the algorithm terminate?
- Does the algorithm give the correct answer?
- How fast does the algorithm converge to the answer?
- How many operations does it take?

There's another very important question we might want to ask:

Think back to our list of questions we ask about algorithms:

- Does the algorithm terminate?
- Does the algorithm give the correct answer?
- How fast does the algorithm converge to the answer?
- How many operations does it take?

There's another very important question we might want to ask:

- Can we parallelise the algorithm?

Think back to our list of questions we ask about algorithms:

- Does the algorithm terminate?
- Does the algorithm give the correct answer?
- How fast does the algorithm converge to the answer?
- How many operations does it take?

There's another very important question we might want to ask:

- Can we parallelise the algorithm?

Every computer nowadays has multiple processing units. (My phone has 8.) Can we use them?

Here's another equivalent expression with different properties:

$$\begin{aligned} &a_0 + a_1 x + \cdots + a_n x^n \\ &\quad = \left(a_0 + a_2 x^2 + a_4 x^4 + \cdots\right) + \left(a_1 x + a_3 x^3 + a_5 x^5 + \cdots\right) \end{aligned}$$

Here's another equivalent expression with different properties:

$$\begin{aligned} & a_0 + a_1 x + \cdots + a_n x^n \\ & \quad = \left(a_0 + a_2 x^2 + a_4 x^4 + \cdots\right) + \left(a_1 x + a_3 x^3 + a_5 x^5 + \cdots\right) \\ & \quad = \left(a_0 + a_2 x^2 + a_4 x^4 + \cdots\right) + x\left(a_1 + a_3 x^2 + a_5 x^4 + \cdots\right) \end{aligned}$$

Here's another equivalent expression with different properties:

$$
\begin{aligned}
&a_0 + a_1 x + \cdots + a_n x^n \\
&\quad = \left(a_0 + a_2 x^2 + a_4 x^4 + \cdots\right) + \left(a_1 x + a_3 x^3 + a_5 x^5 + \cdots\right) \\
&\quad = \left(a_0 + a_2 x^2 + a_4 x^4 + \cdots\right) + x\left(a_1 + a_3 x^2 + a_5 x^4 + \cdots\right) \\
&\quad = p_1(x^2) + x p_2(x^2)
\end{aligned}
$$

Here's another equivalent expression with different properties:

$$
\begin{aligned}
&a_0 + a_1 x + \cdots + a_n x^n \\
&\quad = \left(a_0 + a_2 x^2 + a_4 x^4 + \cdots\right) + \left(a_1 x + a_3 x^3 + a_5 x^5 + \cdots\right) \\
&\quad = \left(a_0 + a_2 x^2 + a_4 x^4 + \cdots\right) + x \left(a_1 + a_3 x^2 + a_5 x^4 + \cdots\right) \\
&\quad = p_1(x^2) + x p_2(x^2)
\end{aligned}
$$

which we can evaluate in parallel with two independent runs of Horner's method.

Here's another equivalent expression with different properties:

$$\begin{aligned} & a_0 + a_1 x + \cdots + a_n x^n \\ & \quad = \left(a_0 + a_2 x^2 + a_4 x^4 + \cdots\right) + \left(a_1 x + a_3 x^3 + a_5 x^5 + \cdots\right) \\ & \quad = \left(a_0 + a_2 x^2 + a_4 x^4 + \cdots\right) + x\left(a_1 + a_3 x^2 + a_5 x^4 + \cdots\right) \\ & \quad = p_1(x^2) + x p_2(x^2) \end{aligned}$$

which we can evaluate in parallel with two independent runs of Horner's method.

More generally, if you have enough terms, you can break p up into $k+1$ polynomials $\{p_j\}_{j=0}^{k}$, each taking the monomial term x^i if

$$i \bmod (k+1) = j.$$

Section 7

Finding all roots of a polynomial

Horner's scheme is just a specialised variant of Newton's method. It finds roots one at a time.

Horner's scheme is just a specialised variant of Newton's method. It finds roots one at a time.

Once you have found a root $x^\star$ of $p_0(x)$, you can construct

$$p_1(x) = \frac{p_0(x)}{(x - x^\star)}$$

and apply the scheme again to p_1. Iterating in this way one can find all real roots, if you can construct good initial guesses.

Horner's scheme is just a specialised variant of Newton's method. It finds roots one at a time.

Once you have found a root $x^\star$ of $p_0(x)$, you can construct

$$p_1(x) = \frac{p_0(x)}{(x - x^\star)}$$

and apply the scheme again to p_1. Iterating in this way one can find all real roots, if you can construct good initial guesses.

Can we find them all at once, without fussing over guesses?

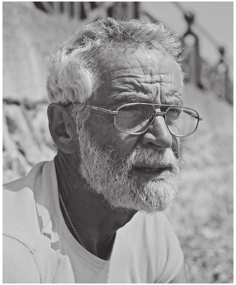
It turns out that we have *very* fast and powerful algorithms for computing the eigenvalues of diagonalisable matrices:

$$\text{for } A \in \mathbb{R}^{n\times n}, \text{ find all } \lambda_i, v_i \text{ s.t. } Av_i = \lambda v_i, \|v_i\|^2 = 1.$$

It turns out that we have *very* fast and powerful algorithms for computing the eigenvalues of diagonalisable matrices:

$$\text{for } A \in \mathbb{R}^{n\times n}, \text{ find all } \lambda_i, v_i \text{ s.t. } Av_i = \lambda v_i, \|v_i\|^2 = 1.$$

The algorithm is called the QR algorithm, invented independently by Francis (1959) and Kublanovskaya (1961). It is widely regarded as one of the ten most important algorithms of the 20^{th} century.



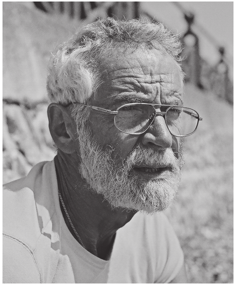
John Francis, 1934–

Vera Kublanovskaya, 1920–2012

It turns out that we have *very* fast and powerful algorithms for computing the eigenvalues of diagonalisable matrices:

$$\text{for } A \in \mathbb{R}^{n\times n}, \text{ find all } \lambda_i, v_i \text{ s.t. } Av_i = \lambda v_i, \|v_i\|^2 = 1.$$

The algorithm is called the QR algorithm, invented independently by Francis (1959) and Kublanovskaya (1961). It is widely regarded as one of the ten most important algorithms of the 20^{th} century.



John Francis, 1934–

Vera Kublanovskaya, 1920–2012

Given $p(x)$ of degree n, we want to construct an A with characteristic polynomial $p(x)$.

Given $p(x)$ of degree n, we want to construct an A with characteristic polynomial $p(x)$. Let

$$p(x) = a_0 + a_1 x + \cdots + x^n$$

be our (monic) polynomial. Then we can construct its *companion matrix*

$$C(a) := \begin{pmatrix} 0 & & & & -a_0 \\ 1 & 0 & & & -a_1 \\ 0 & 1 & 0 & & -a_2 \\ & & \ddots & & \vdots \\ & & & 1 & -a_{n-1} \end{pmatrix}.$$

Given $p(x)$ of degree n, we want to construct an A with characteristic polynomial $p(x)$. Let

$$p(x) = a_0 + a_1 x + \cdots + x^n$$

be our (monic) polynomial. Then we can construct its *companion matrix*

$$C(a) := \begin{pmatrix} 0 & & & & -a_0 \\ 1 & 0 & & & -a_1 \\ 0 & 1 & 0 & & -a_2 \\ & & \ddots & & \vdots \\ & & & 1 & -a_{n-1} \end{pmatrix}.$$

By construction, we have (proof is by induction):

$$\det(C(a) - \lambda I) = (-1)^n p(\lambda).$$

By applying the QR algorithm for eigenvalues to the companion matrix, we can find all roots in $\mathcal{O}(n^3)$ operations.

By applying the QR algorithm for eigenvalues to the companion matrix, we can find all roots in $\mathcal{O}(n^3)$ operations.

We previously saw that Newton's method can get stuck in a cycle for $p(x) = x^3 - 2x + 2$. No problem:

```
In [2]: np.roots([1, 0, -2, -2])
Out[2]:
array([ 1.76929235+0.j,
       -0.88464618+0.58974281j,
       -0.88464618-0.58974281j])
```

Section 8

Representing polynomials

Philosophical remark

Algorithms are usually tied to the *data structures* we use.

Philosophical remark

Algorithms are usually tied to the *data structures* we use.

For example, as mathematicians we might think of $p \in \Pi_n$, the vector space of degree-n polynomials. But Horner's method and the companion matrix *rely* on a particular representation of p, in the monomial basis $\{M_i\}$:

$$p(x) = \sum_{i=0}^{n} a_i M_i(x), \quad M_i(x) \coloneqq x^i.$$

Philosophical remark

Algorithms are usually tied to the *data structures* we use.

For example, as mathematicians we might think of $p \in \Pi_n$, the vector space of degree-n polynomials. But Horner's method and the companion matrix *rely* on a particular representation of p, in the monomial basis $\{M_i\}$:

$$p(x) = \sum_{i=0}^{n} a_i M_i(x), \quad M_i(x) := x^i.$$

The algorithms take in $[a_0, a_1, \ldots, a_n] \in \mathbb{R}^{n+1}$ to represent p.

Philosophical remark

Algorithms are usually tied to the *data structures* we use.

For example, as mathematicians we might think of $p \in \Pi_n$, the vector space of degree-n polynomials. But Horner's method and the companion matrix *rely* on a particular representation of p, in the monomial basis $\{M_i\}$:

$$p(x) = \sum_{i=0}^{n} a_i M_i(x), \quad M_i(x) := x^i.$$

The algorithms take in $[a_0, a_1, \ldots, a_n] \in \mathbb{R}^{n+1}$ to represent p.

A natural question to ask:

is the map $a \mapsto p$ *stable*?

If we make a perturbation δa to a, how big can the perturbation δp be? For the monomial basis $\{M_i\}$, the answer is *very very big*:

If we make a perturbation δa to a, how big can the perturbation δp be? For the monomial basis $\{M_i\}$, the answer is *very very big*:

Construct

$$p(x) = \prod_{i=1}^{20}(x - i), \quad x \in [0, 20],$$

then perturb its monomial coefficients by

$$\delta a = [0, -2^{-23}, 0, \ldots, 0].$$

James H. Wilkinson, 1919–1986

If we make a perturbation δa to a, how big can the perturbation δp be? For the monomial basis $\{M_i\}$, the answer is *very very big*:

Construct

$$p(x) = \prod_{i=1}^{20}(x-i), \quad x \in [0,20],$$

then perturb its monomial coefficients by

$$\delta a = [0, -2^{-23}, 0, \dots, 0].$$

James H. Wilkinson, 1919–1986

The resulting δp has

$$\|\delta p\|_\infty := \max\{|\delta p(x)| : x \in [0,20]\} \approx 6.25 \times 10^{17}$$

for a *stability constant* of

$$\frac{\|\delta p\|_\infty}{\|\delta a\|_\infty} \approx 5 \times 10^{24}.$$

Philosophical remark

Not all bases are equally good.

Philosophical remark

Not all bases are equally good.

For example, for $\varepsilon > 0$, the set

$$\{(1,0)^\top, (1,\varepsilon)^\top\}$$

is as much of a basis for $\mathbb{R}^2$ as

$$\{(1,0)^\top, (0,1)^\top\}.$$

Philosophical remark

Not all bases are equally good.

For example, for $\varepsilon > 0$, the set

$$\{(1,0)^\top, (1,\varepsilon)^\top\}$$

is as much of a basis for $\mathbb{R}^2$ as

$$\{(1,0)^\top, (0,1)^\top\}.$$

But you'd much rather compute with the latter than the former for small ε.

So what is a good basis for polynomials? An excellent choice on $[a, b]$ is

$$p(x) = \sum_{i=0}^{n} c_i T_i(\hat{x}(x)), \quad \hat{x} = \frac{2(x-a)}{(b-a)} - 1$$

where the *Chebyshev polynomials* $\{T_i : [-1, 1] \to [-1, 1]\}$ satisfy

$$T_0(\hat{x}) = 1, \quad T_1(\hat{x}) = \hat{x}, \quad T_{i+1}(\hat{x}) = 2\hat{x}T_i(\hat{x}) - T_{i-1}(\hat{x}).$$

The role of the $\hat{x}$ is to map the input interval $[a, b]$ to $[-1, 1]$.

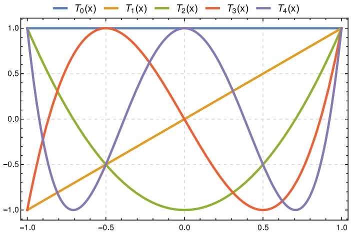
So what is a good basis for polynomials? An excellent choice on $[a, b]$ is

$$p(x) = \sum_{i=0}^{n} c_i T_i(\hat{x}(x)), \quad \hat{x} = \frac{2(x-a)}{(b-a)} - 1$$

where the *Chebyshev polynomials* $\{T_i : [-1, 1] \to [-1, 1]\}$ satisfy

$$T_0(\hat{x}) = 1, \quad T_1(\hat{x}) = \hat{x}, \quad T_{i+1}(\hat{x}) = 2\hat{x}T_i(\hat{x}) - T_{i-1}(\hat{x}).$$

The role of the $\hat{x}$ is to map the input interval $[a, b]$ to $[-1, 1]$.



Chebyshev polynomials. Credit: Glosser.ca, Wikipedia

Using this basis yields a *stable* map $c \mapsto p$. For Wilkinson's polynomial,

$$\|\delta p\|_\infty / \|\delta c\|_\infty \approx 1.$$

Using this basis yields a *stable* map $c \mapsto p$. For Wilkinson's polynomial,

$$\|\delta p\|_\infty / \|\delta c\|_\infty \approx 1.$$

Just as a polynomial p has a finite Chebyshev series, general functions f have infinite Chebyshev series. These expansions converge very, very fast:

Using this basis yields a *stable* map $c \mapsto p$. For Wilkinson's polynomial,

$$\|\delta p\|_\infty / \|\delta c\|_\infty \approx 1.$$

Just as a polynomial p has a finite Chebyshev series, general functions f have infinite Chebyshev series. These expansions converge very, very fast:

Theorem

Let $f : [a, b] \to \mathbb{R}$ be analytic with Chebyshev expansion

$$f(x) = \sum_{i=0}^{\infty} c_i T_i(x).$$

Then for a constant $C > 1$

$$\|f - p_n\|_\infty = \mathcal{O}(C^{-n}), \quad p_n(x) = \sum_{i=0}^{n} c_i T_i(x).$$

But we can't apply Horner's method for evaluation, or the companion matrix trick for rootfinding, to Chebyshev expansions!

But we can't apply Horner's method for evaluation, or the companion matrix trick for rootfinding, to Chebyshev expansions!

Our algorithms are tied to our choice of representation: to our choice of basis, or (from a CS perspective) the data structure we use.

But we can't apply Horner's method for evaluation, or the companion matrix trick for rootfinding, to Chebyshev expansions!

Our algorithms are tied to our choice of representation: to our choice of basis, or (from a CS perspective) the data structure we use.

Good news

For Chebyshev bases, analogous algorithms exist:

- ✓ the *second barycentric formula*, for $\mathcal{O}(n)$ evaluation, and
- ✓ the *colleague matrix*, for finding all roots with the QR algorithm.

These allow us to work with polynomials with degrees in the millions.

Computational Mathematics
Week 4: Higher-dimensional rootfinding

Patrick E. Farrell

University of Oxford

We have considered several algorithms for rootfinding over $\mathbb{R}$:

$$\text{given } f : \mathbb{R} \to \mathbb{R}, \text{ find } x^\star \in \mathbb{R} \text{ such that } f(x^\star) = 0.$$

We have considered several algorithms for rootfinding over $\mathbb{R}$:

$$\text{given } f : \mathbb{R} \to \mathbb{R}, \text{ find } x^\star \in \mathbb{R} \text{ such that } f(x^\star) = 0.$$

- bisection ($q = 1$, $\mu = 1/2$, when it applies)
- secant method ($q = \phi \approx 1.618$, usually)
- Newton's method ($q = 2$, usually)
- Halley's method ($q = 3$, usually)

In real life, most problems involve more than one variable. So let's consider

$$\text{given } F : \mathbb{R}^N \to \mathbb{R}^N, \text{ find } \mathbf{x}^\star \in \mathbb{R}^N \text{ such that } F(\mathbf{x}^\star) = \mathbf{0}.$$

In real life, most problems involve more than one variable. So let's consider

$$\text{given } F : \mathbb{R}^N \to \mathbb{R}^N, \text{ find } \mathbf{x}^\star \in \mathbb{R}^N \text{ such that } F(\mathbf{x}^\star) = \mathbf{0}.$$

Simpson extended Newton's method to this case in his 1740 book *Essays on Several Curious and Useful Subjects in Speculative and Mix'd Mathematicks, Illustrated by a Variety of Examples*.

Thomas Simpson, 1710–1761

Section 2

Derivation of Newton's method

The geometric pictures we had in one dimension don't naturally extend to higher dimensions. So first let's see another derivation of Newton's method in $\mathbb{R}$ that does extend.

The geometric pictures we had in one dimension don't naturally extend to higher dimensions. So first let's see another derivation of Newton's method in $\mathbb{R}$ that does extend.

Consider a Taylor expansion of f. We want to find $x_{i+1} = x_i + \delta x$:

$$f(x_i + \delta x) = f(x_i) + \delta x f'(x_i) + \text{higher-order terms.}$$

The geometric pictures we had in one dimension don't naturally extend to higher dimensions. So first let's see another derivation of Newton's method in $\mathbb{R}$ that does extend.

Consider a Taylor expansion of f. We want to find $x_{i+1} = x_i + \delta x$:

$$f(x_i + \delta x) = f(x_i) + \delta x f'(x_i) + \text{higher-order terms.}$$

We want to choose the update δx so that $f(x_i + \delta x) = 0$. Setting the left-hand side to zero, and dropping higher-order terms, we get

$$\delta x = -[f'(x_i)]^{-1} f(x_i), \quad x_{i+1} = x_i + \delta x,$$

which we recognise as Newton's scheme written in update form.

Taylor's theorem extends to higher dimensions, with the role of derivative f' replaced by the **Jacobian** matrix. If $F : \mathbb{R}^N \to \mathbb{R}^N$ looks like

$$F(\mathbf{x}) = F\begin{pmatrix} \mathbf{x}^1 \\ \mathbf{x}^2 \\ \vdots \\ \mathbf{x}^N \end{pmatrix} = \begin{pmatrix} F^1(\mathbf{x}^1, \ldots, \mathbf{x}^N) \\ F^2(\mathbf{x}^1, \ldots, \mathbf{x}^N) \\ \vdots \\ F^N(\mathbf{x}^1, \ldots, \mathbf{x}^N) \end{pmatrix},$$

then its Jacobian $DF : \mathbb{R}^N \to \mathbb{R}^{N \times N}$ is

Taylor's theorem extends to higher dimensions, with the role of derivative f' replaced by the **Jacobian** matrix. If $F : \mathbb{R}^N \to \mathbb{R}^N$ looks like

$$F(\mathbf{x}) = F \begin{pmatrix} \mathbf{x}^1 \\ \mathbf{x}^2 \\ \vdots \\ \mathbf{x}^N \end{pmatrix} = \begin{pmatrix} F^1(\mathbf{x}^1, \ldots, \mathbf{x}^N) \\ F^2(\mathbf{x}^1, \ldots, \mathbf{x}^N) \\ \vdots \\ F^N(\mathbf{x}^1, \ldots, \mathbf{x}^N) \end{pmatrix},$$

then its Jacobian $DF : \mathbb{R}^N \to \mathbb{R}^{N \times N}$ is

$$DF(\mathbf{a}) := \begin{bmatrix} \frac{\partial F^1}{x^1}(\mathbf{a}) & \frac{\partial F^1}{x^2}(\mathbf{a}) & \cdots & \frac{\partial F^1}{x^N}(\mathbf{a}) \\ \frac{\partial F^2}{x^1}(\mathbf{a}) & \frac{\partial F^2}{x^2}(\mathbf{a}) & \cdots & \frac{\partial F^2}{x^N}(\mathbf{a}) \\ \vdots & \vdots & & \vdots \\ \frac{\partial F^N}{x^1}(\mathbf{a}) & \frac{\partial F^N}{x^2}(\mathbf{a}) & \cdots & \frac{\partial F^N}{x^N}(\mathbf{a}) \end{bmatrix}.$$

Taylor expansions in higher dimensions look like

$$F(\mathbf{x}) = F(\mathbf{a}) + DF(\mathbf{a})(\mathbf{x} - \mathbf{a}) + \text{ higher-order terms.}$$

Taylor expansions in higher dimensions look like

$$F(\mathbf{x}) = F(\mathbf{a}) + DF(\mathbf{a})(\mathbf{x} - \mathbf{a}) + \text{ higher-order terms.}$$

Following the reasoning from one dimension,

$$F(\mathbf{x}_{i+1}) = F(\mathbf{x}_i + \delta\mathbf{x}) \approx F(\mathbf{x}_i) + DF(\mathbf{x}_i)\delta\mathbf{x},$$

Taylor expansions in higher dimensions look like

$$F(\mathbf{x}) = F(\mathbf{a}) + DF(\mathbf{a})(\mathbf{x} - \mathbf{a}) + \text{ higher-order terms.}$$

Following the reasoning from one dimension,

$$F(\mathbf{x}_{i+1}) = F(\mathbf{x}_i + \delta\mathbf{x}) \approx F(\mathbf{x}_i) + DF(\mathbf{x}_i)\delta\mathbf{x},$$

and optimistically setting $F(\mathbf{x}_{i+1}) = 0$, we get

$$\delta\mathbf{x} = -[DF(\mathbf{x}_i)]^{-1}F(\mathbf{x}_i), \quad \mathbf{x}_{i+1} = \mathbf{x}_i + \delta\mathbf{x}.$$

Taylor expansions in higher dimensions look like

$$F(\mathbf{x}) = F(\mathbf{a}) + DF(\mathbf{a})(\mathbf{x} - \mathbf{a}) + \text{ higher-order terms.}$$

Following the reasoning from one dimension,

$$F(\mathbf{x}_{i+1}) = F(\mathbf{x}_i + \delta\mathbf{x}) \approx F(\mathbf{x}_i) + DF(\mathbf{x}_i)\delta\mathbf{x},$$

and optimistically setting $F(\mathbf{x}_{i+1}) = 0$, we get

$$\delta\mathbf{x} = -[DF(\mathbf{x}_i)]^{-1}F(\mathbf{x}_i), \quad \mathbf{x}_{i+1} = \mathbf{x}_i + \delta\mathbf{x}.$$

In practice, we don't actually invert the matrix, but rather

$$\text{solve } DF(\mathbf{x}_i)\delta\mathbf{x} = -F(\mathbf{x}_i),$$

using e.g. an LU factorisation of the matrix.

Newton–Raphson method

$$\mathbf{x}_{i+1} = g(\mathbf{x}_i) := \mathbf{x}_i - \left(DF(\mathbf{x}_i)\right)^{-1} F(\mathbf{x}_i).$$

Comments:

✓ Still a fixed-point method.

Newton–Raphson method

$$\mathbf{x}_{i+1} = g(\mathbf{x}_i) := \mathbf{x}_i - (DF(\mathbf{x}_i))^{-1} F(\mathbf{x}_i).$$

Comments:

- ✓ Still a fixed-point method.
- ✓ Fixed points of g are roots of F.

Newton–Raphson method

$$\mathbf{x}_{i+1} = g(\mathbf{x}_i) \coloneqq \mathbf{x}_i - (DF(\mathbf{x}_i))^{-1} F(\mathbf{x}_i).$$

Comments:

- ✓ Still a fixed-point method.
- ✓ Fixed points of g are roots of F.
- ✗ We still require F to be differentiable.

Newton–Raphson method

$$\mathbf{x}_{i+1} = g(\mathbf{x}_i) := \mathbf{x}_i - \left(DF(\mathbf{x}_i)\right)^{-1} F(\mathbf{x}_i).$$

Comments:

- ✓ Still a fixed-point method.
- ✓ Fixed points of g are roots of F.
- ✗ We still require F to be differentiable.
- ✗ We now require DF to be invertible at every iterate.

Newton–Raphson method

$$\mathbf{x}_{i+1} = g(\mathbf{x}_i) \coloneqq \mathbf{x}_i - (DF(\mathbf{x}_i))^{-1} F(\mathbf{x}_i).$$

Comments:

- ✓ Still a fixed-point method.
- ✓ Fixed points of g are roots of F.
- ✗ We still require F to be differentiable.
- ✗ We now require DF to be invertible at every iterate.
- ✗ We have to solve linear systems (worse case $\mathcal{O}(N^3)$ operations).

Newton–Raphson method

$$\mathbf{x}_{i+1} = g(\mathbf{x}_i) := \mathbf{x}_i - (DF(\mathbf{x}_i))^{-1} F(\mathbf{x}_i).$$

Comments:

- ✓ Still a fixed-point method.
- ✓ Fixed points of g are roots of F.
- ✗ We still require F to be differentiable.
- ✗ We now require DF to be invertible at every iterate.
- ✗ We have to solve linear systems (worse case $\mathcal{O}(N^3)$ operations).
- ✓ Sometimes the linear systems can be solved in $\mathcal{O}(N)$ operations.

Newton–Raphson method

$$\mathbf{x}_{i+1} = g(\mathbf{x}_i) := \mathbf{x}_i - (DF(\mathbf{x}_i))^{-1} F(\mathbf{x}_i).$$

Comments:

- ✓ Still a fixed-point method.
- ✓ Fixed points of g are roots of F.
- ✗ We still require F to be differentiable.
- ✗ We now require DF to be invertible at every iterate.
- ✗ We have to solve linear systems (worse case $\mathcal{O}(N^3)$ operations).
- ✓ Sometimes the linear systems can be solved in $\mathcal{O}(N)$ operations.
- ✓ If x_0 is close to $x^\star$, Newton's method usually converges quadratically.

Newton–Raphson method

$$\mathbf{x}_{i+1} = g(\mathbf{x}_i) := \mathbf{x}_i - (DF(\mathbf{x}_i))^{-1} F(\mathbf{x}_i).$$

Comments:

- ✓ Still a fixed-point method.
- ✓ Fixed points of g are roots of F.
- ✗ We still require F to be differentiable.
- ✗ We now require DF to be invertible at every iterate.
- ✗ We have to solve linear systems (worse case $\mathcal{O}(N^3)$ operations).
- ✓ Sometimes the linear systems can be solved in $\mathcal{O}(N)$ operations.
- ✓ If x_0 is close to $x^\star$, Newton's method usually converges quadratically.
- ✗ If x_0 is far away, the method can diverge or get stuck in a cycle.

Newton–Raphson method

$$\mathbf{x}_{i+1} = g(\mathbf{x}_i) := \mathbf{x}_i - (DF(\mathbf{x}_i))^{-1} F(\mathbf{x}_i).$$

Comments:

- ✓ Still a fixed-point method.
- ✓ Fixed points of g are roots of F.
- ✗ We still require F to be differentiable.
- ✗ We now require DF to be invertible at every iterate.
- ✗ We have to solve linear systems (worse case $\mathcal{O}(N^3)$ operations).
- ✓ Sometimes the linear systems can be solved in $\mathcal{O}(N)$ operations.
- ✓ If x_0 is close to $x^\star$, Newton's method usually converges quadratically.
- ✗ If x_0 is far away, the method can diverge or get stuck in a cycle.
- ✓ Newton's method even generalises to infinite dimensions.

Section 3

Example

Let's do an example. Take

$$F(x,y) = \begin{pmatrix} xy + y^2 - 2 \\ x^3 y - 3x - 1 \end{pmatrix},$$

Let's do an example. Take

$$F(x,y) = \begin{pmatrix} xy + y^2 - 2 \\ x^3 y - 3x - 1 \end{pmatrix}, \text{ with } DF(x,y) = \begin{pmatrix} y & x + 2y \\ 3x^2 y - 3 & x^3 \end{pmatrix}.$$

Let's do an example. Take

$$F(x,y) = \begin{pmatrix} xy + y^2 - 2 \\ x^3y - 3x - 1 \end{pmatrix}, \text{ with } DF(x,y) = \begin{pmatrix} y & x + 2y \\ 3x^2y - 3 & x^3 \end{pmatrix}.$$

Starting at $\mathbf{x}_0 = (0,1)^\top$, we have to solve

$$\begin{pmatrix} 1 & 2 \\ -3 & 0 \end{pmatrix} \begin{pmatrix} \delta x \\ \delta y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Let's do an example. Take

$$F(x,y) = \begin{pmatrix} xy + y^2 - 2 \\ x^3y - 3x - 1 \end{pmatrix}, \text{ with } DF(x,y) = \begin{pmatrix} y & x + 2y \\ 3x^2y - 3 & x^3 \end{pmatrix}.$$

Starting at $\mathbf{x}_0 = (0,1)^\top$, we have to solve

$$\begin{pmatrix} 1 & 2 \\ -3 & 0 \end{pmatrix} \begin{pmatrix} \delta x \\ \delta y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

This yields $(\delta x, \delta y)^\top = (-1/3, 2/3)^\top$, so

$$\mathbf{x}_1 = \mathbf{x_0} + \delta\mathbf{x} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} -1/3 \\ 2/3 \end{pmatrix} = \begin{pmatrix} -1/3 \\ 5/3 \end{pmatrix}.$$

Repeating the procedure, the next iterates are

$$\mathbf{x}_2 = \begin{pmatrix} -0.357668 \\ 1.606112 \end{pmatrix},$$

Let's do an example. Take

$$F(x,y) = \begin{pmatrix} xy + y^2 - 2 \\ x^3 y - 3x - 1 \end{pmatrix}, \text{ with } DF(x,y) = \begin{pmatrix} y & x + 2y \\ 3x^2 y - 3 & x^3 \end{pmatrix}.$$

Starting at $\mathbf{x}_0 = (0,1)^\top$, we have to solve

$$\begin{pmatrix} 1 & 2 \\ -3 & 0 \end{pmatrix} \begin{pmatrix} \delta x \\ \delta y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

This yields $(\delta x, \delta y)^\top = (-1/3, 2/3)^\top$, so

$$\mathbf{x}_1 = \mathbf{x_0} + \delta\mathbf{x} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} -1/3 \\ 2/3 \end{pmatrix} = \begin{pmatrix} -1/3 \\ 5/3 \end{pmatrix}.$$

Repeating the procedure, the next iterates are

$$\mathbf{x}_2 = \begin{pmatrix} -0.357668 \\ 1.606112 \end{pmatrix}, \quad \mathbf{x}_3 = \begin{pmatrix} -0.357838 \\ 1.604407 \end{pmatrix},$$

Let's do an example. Take

$$F(x,y) = \begin{pmatrix} xy + y^2 - 2 \\ x^3y - 3x - 1 \end{pmatrix}, \text{ with } DF(x,y) = \begin{pmatrix} y & x + 2y \\ 3x^2y - 3 & x^3 \end{pmatrix}.$$

Starting at $\mathbf{x}_0 = (0,1)^\top$, we have to solve

$$\begin{pmatrix} 1 & 2 \\ -3 & 0 \end{pmatrix} \begin{pmatrix} \delta x \\ \delta y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

This yields $(\delta x, \delta y)^\top = (-1/3, 2/3)^\top$, so

$$\mathbf{x}_1 = \mathbf{x_0} + \delta\mathbf{x} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} -1/3 \\ 2/3 \end{pmatrix} = \begin{pmatrix} -1/3 \\ 5/3 \end{pmatrix}.$$

Repeating the procedure, the next iterates are

$$\mathbf{x}_2 = \begin{pmatrix} -0.357668 \\ 1.606112 \end{pmatrix}, \quad \mathbf{x}_3 = \begin{pmatrix} -0.357838 \\ 1.604407 \end{pmatrix}, \quad \mathbf{x}_4 = \begin{pmatrix} -0.357838 \\ 1.604406 \end{pmatrix}.$$

Section 4

Convergence

Definition (Norm of $\mathbf{x} \in \mathbb{R}^N$)

Given $\mathbf{x} \in \mathbb{R}^N$, we define its ∞-norm to be

$$\|\mathbf{x}\|_\infty := \max_{k=1,\ldots,N} |\mathbf{x}^k|.$$

Definition (Norm of $\mathbf{x} \in \mathbb{R}^N$)

Given $\mathbf{x} \in \mathbb{R}^N$, we define its ∞-norm to be

$$\|\mathbf{x}\|_\infty := \max_{k=1,\dots,N} |\mathbf{x}^k|.$$

Definition (Convergence of a vector-valued sequence)

We say $(\mathbf{x}_i) \to \mathbf{x}^\star$ in the $\|\cdot\|_\infty$ norm if

$$\lim_{i\to\infty} \|\mathbf{x}_i - \mathbf{x}^\star\|_\infty = 0.$$

Definition (Norm of $\mathbf{x} \in \mathbb{R}^N$)

Given $\mathbf{x} \in \mathbb{R}^N$, we define its ∞-norm to be

$$\|\mathbf{x}\|_\infty := \max_{k=1,\dots,N} |\mathbf{x}^k|.$$

Definition (Convergence of a vector-valued sequence)

We say $(\mathbf{x}_i) \to \mathbf{x}^\star$ in the $\|\cdot\|_\infty$ norm if

$$\lim_{i\to\infty} \|\mathbf{x}_i - \mathbf{x}^\star\|_\infty = 0.$$

Definition (Order of convergence of a sequence)

Suppose $(\mathbf{x}_i) \to \mathbf{x}^\star$. The sequence converges with order q if for some $M > 0$

$$\lim_{i\to\infty} \frac{\|\mathbf{x}_{i+1} - \mathbf{x}^\star\|_\infty}{\|\mathbf{x}_i - \mathbf{x}^\star\|_\infty^q} = M.$$

Assuming Newton's method converges, how fast does it converge? From our one-dimensional experience, we expect quadratic convergence to isolated roots.

Assuming Newton's method converges, how fast does it converge? From our one-dimensional experience, we expect quadratic convergence to isolated roots.

Theorem (Quadratic convergence of Newton's method)

Let $F \in C^2(\mathbb{R}^N, \mathbb{R}^N)$, i.e. F is continuous with all first and second partial derivatives continuous. Suppose $\mathbf{x}^\star \in \mathbb{R}^N$ is an isolated root of F, i.e. $F(\mathbf{x}^\star) = \mathbf{0}$ with $DF(\mathbf{x}^\star)$ nonsingular. Then if $\mathbf{x}_0$ is close enough to $\mathbf{x}^\star$, the Newton sequence will converge quadratically.

Assuming Newton's method converges, how fast does it converge? From our one-dimensional experience, we expect quadratic convergence to isolated roots.

Theorem (Quadratic convergence of Newton's method)

Let $F \in C^2(\mathbb{R}^N, \mathbb{R}^N)$, i.e. F is continuous with all first and second partial derivatives continuous. Suppose $\mathbf{x}^\star \in \mathbb{R}^N$ is an isolated root of F, i.e. $F(\mathbf{x}^\star) = \mathbf{0}$ with $DF(\mathbf{x}^\star)$ nonsingular. Then if $\mathbf{x}_0$ is close enough to $\mathbf{x}^\star$, the Newton sequence will converge quadratically.

The core of the proof is that the Jacobian matrix of the associated fixed-point iteration is zero at $\mathbf{x}^\star$.

Section 5

Affine covariance

Newton's method has an important property that becomes apparent in higher dimensions.

Newton's method has an important property that becomes apparent in higher dimensions.

Given $F : \mathbb{R}^N \to \mathbb{R}^N$, and $\mathbf{x}_0 \in \mathbb{R}^N$, we construct the sequence $\mathbf{x}_0, \mathbf{x}_1, \ldots$.

Newton's method has an important property that becomes apparent in higher dimensions.

Given $F : \mathbb{R}^N \to \mathbb{R}^N$, and $\mathbf{x}_0 \in \mathbb{R}^N$, we construct the sequence $\mathbf{x}_0, \mathbf{x}_1, \ldots$.

Now imagine that we change units or coordinate systems for our outputs F. Instead of solving $F(\mathbf{x}) = \mathbf{0}$, we want to solve $\tilde{F}(\mathbf{x}) = AF(\mathbf{x}) = \mathbf{0}$, where $A \in \mathbb{R}^{N\times N}$ is constant and nonsingular. Of course, this doesn't change the roots $\mathbf{x}^\star$.

Newton's method has an important property that becomes apparent in higher dimensions.

Given $F : \mathbb{R}^N \to \mathbb{R}^N$, and $\mathbf{x}_0 \in \mathbb{R}^N$, we construct the sequence $\mathbf{x}_0, \mathbf{x}_1, \ldots$.

Now imagine that we change units or coordinate systems for our outputs F. Instead of solving $F(\mathbf{x}) = \mathbf{0}$, we want to solve $\tilde{F}(\mathbf{x}) = AF(\mathbf{x}) = \mathbf{0}$, where $A \in \mathbb{R}^{N\times N}$ is constant and nonsingular. Of course, this doesn't change the roots $\mathbf{x}^\star$.

Theorem (Affine covariance)

Premultiplying F by a constant nonsingular $A \in \mathbb{R}^{N\times N}$ does not change the Newton sequence.

Let $\tilde{F}(\mathbf{x}) := AF(\mathbf{x})$. Newton's method applied to $\tilde{F}$ from $\mathbf{x}_0 = \tilde{\mathbf{x}}_0$ generates a sequence

$$\tilde{\mathbf{x}}_0, \tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \dots .$$

Proof.

For $i = 0$, we have $\mathbf{x}_i = \tilde{\mathbf{x}}_i$ by assumption.

Let $\tilde{F}(\mathbf{x}) := AF(\mathbf{x})$. Newton's method applied to $\tilde{F}$ from $\mathbf{x}_0 = \tilde{\mathbf{x}}_0$ generates a sequence

$$\tilde{\mathbf{x}}_0, \tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \ldots .$$

Proof.

For $i = 0$, we have $\mathbf{x}_i = \tilde{\mathbf{x}}_i$ by assumption.

Assume $\mathbf{x}_i = \tilde{\mathbf{x}}_i$ at iteration i. Then the Newton update for $\tilde{F}$ satisfies

$$-\delta\tilde{\mathbf{x}}_i = [D\tilde{F}(\tilde{\mathbf{x}}_i)]^{-1}\tilde{F}(\tilde{\mathbf{x}}_i) = [ADF(\mathbf{x}_i)]^{-1}AF(\mathbf{x}_i)$$

Let $\tilde{F}(\mathbf{x}) := AF(\mathbf{x})$. Newton's method applied to $\tilde{F}$ from $\mathbf{x}_0 = \tilde{\mathbf{x}}_0$ generates a sequence

$$\tilde{\mathbf{x}}_0, \tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \ldots .$$

Proof.

For $i = 0$, we have $\mathbf{x}_i = \tilde{\mathbf{x}}_i$ by assumption.

Assume $\mathbf{x}_i = \tilde{\mathbf{x}}_i$ at iteration i. Then the Newton update for $\tilde{F}$ satisfies

$$\begin{aligned} -\delta\tilde{\mathbf{x}}_i = [D\tilde{F}(\tilde{\mathbf{x}}_i)]^{-1}\tilde{F}(\tilde{\mathbf{x}}_i) &= [ADF(\mathbf{x}_i)]^{-1}AF(\mathbf{x}_i) \\ &= [DF(\mathbf{x}_i)]^{-1}A^{-1}AF(\mathbf{x}_i) \end{aligned}$$

Let $\tilde{F}(\mathbf{x}) := AF(\mathbf{x})$. Newton's method applied to $\tilde{F}$ from $\mathbf{x}_0 = \tilde{\mathbf{x}}_0$ generates a sequence

$$\tilde{\mathbf{x}}_0, \tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \ldots.$$

Proof.

For $i = 0$, we have $\mathbf{x}_i = \tilde{\mathbf{x}}_i$ by assumption.

Assume $\mathbf{x}_i = \tilde{\mathbf{x}}_i$ at iteration i. Then the Newton update for $\tilde{F}$ satisfies

$$\begin{aligned} -\delta\tilde{\mathbf{x}}_i = [D\tilde{F}(\tilde{\mathbf{x}}_i)]^{-1}\tilde{F}(\tilde{\mathbf{x}}_i) &= [ADF(\mathbf{x}_i)]^{-1}AF(\mathbf{x}_i) \\ &= [DF(\mathbf{x}_i)]^{-1}A^{-1}AF(\mathbf{x}_i) \\ &= [DF(\mathbf{x}_i)]^{-1}F(\mathbf{x}_i) \end{aligned}$$

Let $\tilde{F}(\mathbf{x}) := AF(\mathbf{x})$. Newton's method applied to $\tilde{F}$ from $\mathbf{x}_0 = \tilde{\mathbf{x}}_0$ generates a sequence

$$\tilde{\mathbf{x}}_0, \tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \ldots .$$

Proof.

For $i = 0$, we have $\mathbf{x}_i = \tilde{\mathbf{x}}_i$ by assumption.

Assume $\mathbf{x}_i = \tilde{\mathbf{x}}_i$ at iteration i. Then the Newton update for $\tilde{F}$ satisfies

$$\begin{aligned} -\delta\tilde{\mathbf{x}}_i = [D\tilde{F}(\tilde{\mathbf{x}}_i)]^{-1}\tilde{F}(\tilde{\mathbf{x}}_i) &= [ADF(\mathbf{x}_i)]^{-1}AF(\mathbf{x}_i) \\ &= [DF(\mathbf{x}_i)]^{-1}A^{-1}AF(\mathbf{x}_i) \\ &= [DF(\mathbf{x}_i)]^{-1}F(\mathbf{x}_i) = -\delta\mathbf{x}_i. \end{aligned}$$

Let $\tilde{F}(\mathbf{x}) := AF(\mathbf{x})$. Newton's method applied to $\tilde{F}$ from $\mathbf{x}_0 = \tilde{\mathbf{x}}_0$ generates a sequence

$$\tilde{\mathbf{x}}_0, \tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \ldots.$$

Proof.

For $i = 0$, we have $\mathbf{x}_i = \tilde{\mathbf{x}}_i$ by assumption.

Assume $\mathbf{x}_i = \tilde{\mathbf{x}}_i$ at iteration i. Then the Newton update for $\tilde{F}$ satisfies

$$\begin{aligned} -\delta\tilde{\mathbf{x}}_i = [D\tilde{F}(\tilde{\mathbf{x}}_i)]^{-1}\tilde{F}(\tilde{\mathbf{x}}_i) &= [ADF(\mathbf{x}_i)]^{-1}AF(\mathbf{x}_i) \\ &= [DF(\mathbf{x}_i)]^{-1}A^{-1}AF(\mathbf{x}_i) \\ &= [DF(\mathbf{x}_i)]^{-1}F(\mathbf{x}_i) = -\delta\mathbf{x}_i. \end{aligned}$$

Hence $\mathbf{x}_{i+1} = \tilde{\mathbf{x}}_{i+1}$, and the result follows by induction.

Let $\tilde{F}(\mathbf{x}) := AF(\mathbf{x})$. Newton's method applied to $\tilde{F}$ from $\mathbf{x}_0 = \tilde{\mathbf{x}}_0$ generates a sequence

$$\tilde{\mathbf{x}}_0, \tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \ldots .$$

Proof.

For $i = 0$, we have $\mathbf{x}_i = \tilde{\mathbf{x}}_i$ by assumption.

Assume $\mathbf{x}_i = \tilde{\mathbf{x}}_i$ at iteration i. Then the Newton update for $\tilde{F}$ satisfies

$$\begin{aligned} -\delta\tilde{\mathbf{x}}_i = [D\tilde{F}(\tilde{\mathbf{x}}_i)]^{-1}\tilde{F}(\tilde{\mathbf{x}}_i) &= [ADF(\mathbf{x}_i)]^{-1}AF(\mathbf{x}_i) \\ &= [DF(\mathbf{x}_i)]^{-1}A^{-1}AF(\mathbf{x}_i) \\ &= [DF(\mathbf{x}_i)]^{-1}F(\mathbf{x}_i) = -\delta\mathbf{x}_i. \end{aligned}$$

Hence $\mathbf{x}_{i+1} = \tilde{\mathbf{x}}_{i+1}$, and the result follows by induction.

We get exactly the same iterates $\mathbf{x}_0, \mathbf{x}_1, \ldots,$ whether we apply Newton to $F(\mathbf{x}) = \mathbf{0}$ or $AF(\mathbf{x}) = \mathbf{0}$.

Why does this matter?

Why does this matter?

Philosophical remark

Since Newton's method is affine covariant, *the conditions for any theorem guaranteeing its convergence* should also be affine covariant.

Why does this matter?

Philosophical remark

Since Newton's method is affine covariant, *the conditions for any theorem guaranteeing its convergence* should also be affine covariant.

This is not true of proofs found in many books!

Why does this matter?

Philosophical remark

Since Newton's method is affine covariant, *the conditions for any theorem guaranteeing its convergence* should also be affine covariant.

This is not true of proofs found in many books!

Moreover, any sensible strategy for globalising the convergence of Newton's method from poor initial guesses $\mathbf{x}_0$ must also preserve this property. This insight leads to the current state of the art for globalising Newton's method.

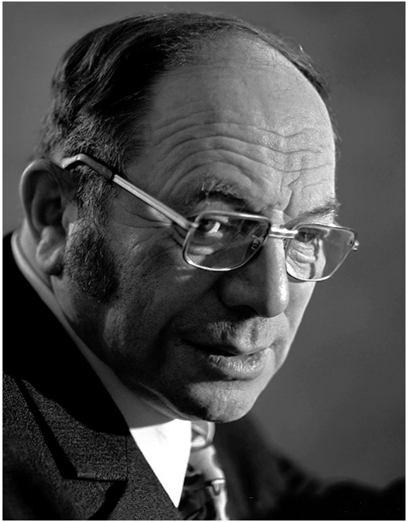
Peter Deuflhard, 1944–2019

Section 6

The Newton–Kantorovich theorem

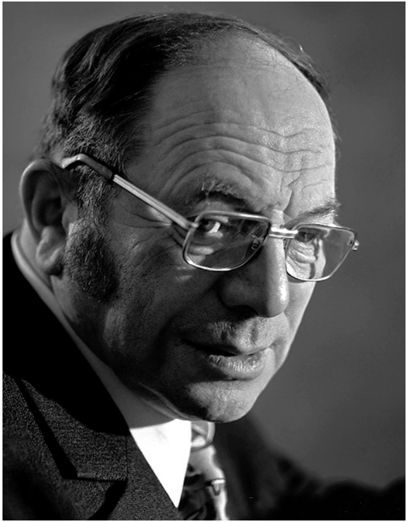
The generalisation of Newton's method to infinite-dimensional (Banach) spaces is called the *Newton–Kantorovich* algorithm.

- Invented linear programming when consulting for the Leningrad Plywood Trust.



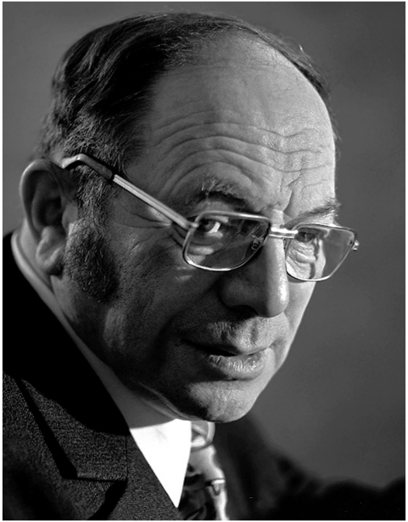
The generalisation of Newton's method to infinite-dimensional (Banach) spaces is called the *Newton–Kantorovich* algorithm.

- Invented linear programming when consulting for the Leningrad Plywood Trust.
- Instrumental in saving millions of lives during the siege of Leningrad.



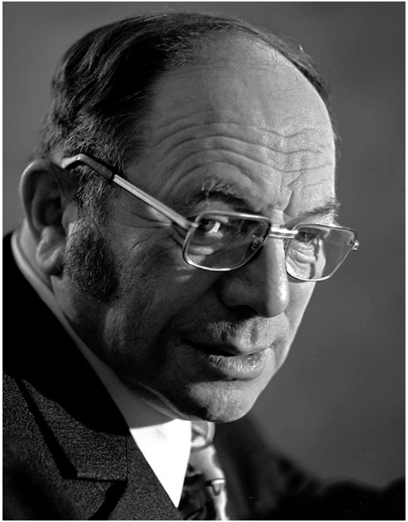
The generalisation of Newton's method to infinite-dimensional (Banach) spaces is called the *Newton–Kantorovich* algorithm.

- Invented linear programming when consulting for the Leningrad Plywood Trust.
- Instrumental in saving millions of lives during the siege of Leningrad.
- Involved in the Soviet nuclear bomb project.



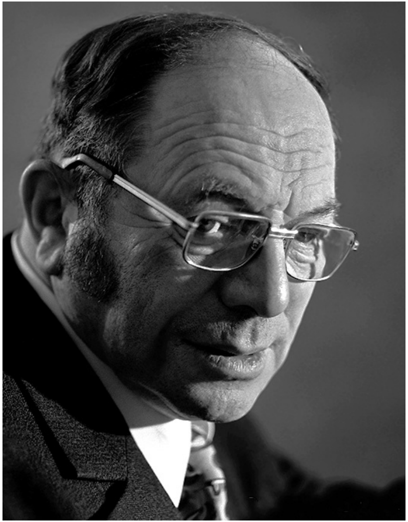
The generalisation of Newton's method to infinite-dimensional (Banach) spaces is called the *Newton–Kantorovich* algorithm.

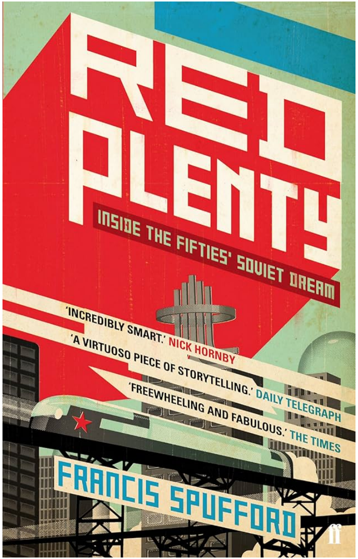
- Invented linear programming when consulting for the Leningrad Plywood Trust.
- Instrumental in saving millions of lives during the siege of Leningrad.
- Involved in the Soviet nuclear bomb project.
- Nearly sent to the gulag for "shadow prices".



The generalisation of Newton's method to infinite-dimensional (Banach) spaces is called the *Newton–Kantorovich* algorithm.

- Invented linear programming when consulting for the Leningrad Plywood Trust.
- Instrumental in saving millions of lives during the siege of Leningrad.
- Involved in the Soviet nuclear bomb project.
- Nearly sent to the gulag for "shadow prices".
- Pseudo-Nobel prize in Economics (1975).




RED PLENTY
INSIDE THE FIFTIES' SOVIET DREAM
'INCREDIBLY SMART.' NICK HORNBY
'A VIRTUOSO PIECE OF STORYTELLING.' DAILY TELEGRAPH
'FREEWHEELING AND FABULOUS.' THE TIMES
FRANCIS SPUFFORD

Kantorovich's theorem (1948) is an example of the application of the Banach contraction mapping theorem. It does not assume the existence of a solution: given certain conditions on the residual and initial guess, it proves the existence and local uniqueness of a solution, and the convergence of the Newton iteration.

Kantorovich's theorem (1948) is an example of the application of the Banach contraction mapping theorem. It does not assume the existence of a solution: given certain conditions on the residual and initial guess, it proves the existence and local uniqueness of a solution, and the convergence of the Newton iteration.

With a good initial guess, and great cleverness, it is possible to devise *computer-assisted proofs* of the existence of solutions to infinite-dimensional nonlinear problems.

Theorem (Kantorovich (1948) in finite dimensions)

Let $F \in C^1(\mathbb{R}^N, \mathbb{R}^N)$ *be the residual of our nonlinear problem, and let* $\mathbf{x}_0 \in \mathbb{R}^N$ *be an initial guess such that the Jacobian* $DF(\mathbf{x}_0)$ *is invertible. Let* $B(\mathbf{x}_0, r)$ *denote the open ball of radius* r *centred at* $\mathbf{x}_0$.

Assume that there exists a constant $r > 0$ *such that*

(1) $\|DF(\mathbf{x}_0)^{-1}F(\mathbf{x}_0)\| \leq \frac{r}{2}$,

Theorem (Kantorovich (1948) in finite dimensions)

Let $F \in C^1(\mathbb{R}^N, \mathbb{R}^N)$ *be the residual of our nonlinear problem, and let* $\mathbf{x}_0 \in \mathbb{R}^N$ *be an initial guess such that the Jacobian* $DF(\mathbf{x}_0)$ *is invertible. Let* $B(\mathbf{x}_0, r)$ *denote the open ball of radius* r *centred at* $\mathbf{x}_0$*.*

Assume that there exists a constant $r > 0$ *such that*

(1) $\|DF(\mathbf{x}_0)^{-1} F(\mathbf{x}_0)\| \leq \frac{r}{2}$,

(2) For all $\tilde{\mathbf{x}}, \mathbf{x} \in B(\mathbf{x}_0, r)$,

$$\|DF(\mathbf{x}_0)^{-1} \left(DF(\tilde{\mathbf{x}}) - DF(\mathbf{x})\right)\| \leq \frac{1}{r}\|\tilde{\mathbf{x}} - \mathbf{x}\|.$$

Theorem (Kantorovich (1948))

Then

(1) $DF(\mathbf{x})$ *is invertible at each* $\mathbf{x} \in B(\mathbf{x}_0, r)$.

Theorem (Kantorovich (1948))

Then

(1) $DF(\mathbf{x})$ *is invertible at each* $\mathbf{x} \in B(\mathbf{x}_0, r)$.

(2) The Newton sequence $(\mathbf{x}_i)_{i=0}^{\infty}$ *defined by*

$$\mathbf{x}_{i+1} = \mathbf{x}_i - DF(\mathbf{x}_i)^{-1} F(\mathbf{x}_i)$$

satisfies $\mathbf{x}_i \in B(\mathbf{x}_0, r)$ *for all* i, *and converges to a root* $\mathbf{x}^\star$ *of* F.

Theorem (Kantorovich (1948))

Then

(1) $DF(\mathbf{x})$ *is invertible at each* $\mathbf{x} \in B(\mathbf{x}_0, r)$.

(2) The Newton sequence $(\mathbf{x}_i)_{i=0}^{\infty}$ *defined by*

$$\mathbf{x}_{i+1} = \mathbf{x}_i - DF(\mathbf{x}_i)^{-1} F(\mathbf{x}_i)$$

satisfies $\mathbf{x}_i \in B(\mathbf{x}_0, r)$ *for all* i, *and converges to a root* $\mathbf{x}^\star$ *of* F.

(3) For each $i \geq 0$,

$$\|\mathbf{x}^\star - \mathbf{x}_i\| \leq \frac{r}{2^i}.$$

Theorem (Kantorovich (1948))

Then

(1) $DF(\mathbf{x})$ *is invertible at each* $\mathbf{x} \in B(\mathbf{x}_0, r)$.

(2) The Newton sequence $(\mathbf{x}_i)_{i=0}^{\infty}$ *defined by*

$$\mathbf{x}_{i+1} = \mathbf{x}_i - DF(\mathbf{x}_i)^{-1} F(\mathbf{x}_i)$$

satisfies $\mathbf{x}_i \in B(\mathbf{x}_0, r)$ *for all* i, *and converges to a root* $\mathbf{x}^\star$ *of* F.

(3) For each $i \geq 0$,

$$\|\mathbf{x}^\star - \mathbf{x}_i\| \leq \frac{r}{2^i}.$$

(4) The root $\mathbf{x}^\star$ *is locally unique, i.e.* $\mathbf{x}^\star$ *is the only root of* F *in the ball* $B\overline{(\mathbf{x}_0, r)}$.

Section 7

The Davidenko differential equation

Newton's method applied to $F(\mathbf{x}) = \mathbf{0}$ produces a sequence

$$\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots, \quad \mathbf{x}_i \in \mathbb{R}^N.$$

Newton's method applied to $F(\mathbf{x}) = \mathbf{0}$ produces a sequence

$$\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \dots, \quad \mathbf{x}_i \in \mathbb{R}^N.$$

Philosophical question

Is there a *curve* $\mathbf{x}(s), s \in [0, \infty)$, associated with this sequence?

Newton's method applied to $F(\mathbf{x}) = \mathbf{0}$ produces a sequence

$$\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots, \quad \mathbf{x}_i \in \mathbb{R}^N.$$

Philosophical question

Is there a *curve* $\mathbf{x}(s), s \in [0, \infty)$, associated with this sequence?

Yes. The *Davidenko differential equation* is

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}s} = -[DF(\mathbf{x})]^{-1} F(\mathbf{x}).$$

Victor Davidenko, 1914–1983

Newton's method applied to $F(\mathbf{x}) = \mathbf{0}$ produces a sequence

$$\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots, \quad \mathbf{x}_i \in \mathbb{R}^N.$$

Philosophical question

Is there a *curve* $\mathbf{x}(s), s \in [0, \infty)$, associated with this sequence?

Yes. The *Davidenko differential equation* is

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}s} = -[DF(\mathbf{x})]^{-1} F(\mathbf{x}).$$

Victor Davidenko, 1914–1983

The Newton iteration is the forward Euler discretisation of the Davidenko differential equation with $\Delta s = 1$:

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}s} \approx \frac{\mathbf{x}(s + \Delta s) - \mathbf{x}(s)}{\Delta s} = -[DF(\mathbf{x}(s))]^{-1} F(\mathbf{x}(s)).$$

Why is this useful?

Why is this useful?

Theorem

For any $\mathbf{x}_0 \in \mathbb{R}^N$, the solution curve of the Davidenko differential equation ends either at

- *a root $\mathbf{x}^\star$, or at*

Why is this useful?

Theorem

For any $\mathbf{x}_0 \in \mathbb{R}^N$, *the solution curve of the Davidenko differential equation ends either at*

- *a root* $\mathbf{x}^\star$, *or at*
- *a singular point of* DF.

Why is this useful?

Theorem

For any $\mathbf{x}_0 \in \mathbb{R}^N$*, the solution curve of the Davidenko differential equation ends either at*

- *a root* $\mathbf{x}^\star$*, or at*
- *a singular point of* DF*.*

This shows us that the tangent of the curve—the Newton update $[DF(\mathbf{x})]^{-1}F(\mathbf{x})$—is a special direction to go to find a root, even far away from a solution. It's just that it might be *too long*.

Why is this useful?

Theorem

For any $\mathbf{x}_0 \in \mathbb{R}^N$, the solution curve of the Davidenko differential equation ends either at

- *a root $\mathbf{x}^\star$, or at*
- *a singular point of DF.*

This shows us that the tangent of the curve—the Newton update $[DF(\mathbf{x})]^{-1}F(\mathbf{x})$—is a special direction to go to find a root, even far away from a solution. It's just that it might be *too long*.

You can use these ideas to build effective globalisation strategies for Newton's method.

Section 8

Newton fractals

One last beautiful idea about Newton's method in higher dimensions.

One last beautiful idea about Newton's method in higher dimensions.

Consider the problem

$$\text{find } z \in \mathbb{C} \text{ such that } z^3 - 1 = 0.$$

We could also think of this as a problem in $\mathbb{R}^2$.

One last beautiful idea about Newton's method in higher dimensions.

Consider the problem

$$\text{find } z \in \mathbb{C} \text{ such that } z^3 - 1 = 0.$$

We could also think of this as a problem in $\mathbb{R}^2$.

We know this has three solutions,

$$z = 1, \quad z = -1/2 + i\sqrt{3}/2, \text{ and } z = -1/2 - i\sqrt{3}/2.$$

Let's take a subset of the complex plane and colour each point as follows. For a given $z_0 \in \mathbb{C}$, we

1. run Newton's method with that initial guess,

One last beautiful idea about Newton's method in higher dimensions.

Consider the problem

$$\text{find } z \in \mathbb{C} \text{ such that } z^3 - 1 = 0.$$

We could also think of this as a problem in $\mathbb{R}^2$.

We know this has three solutions,

$$z = 1, \quad z = -1/2 + i\sqrt{3}/2, \text{ and } z = -1/2 - i\sqrt{3}/2.$$

Let's take a subset of the complex plane and colour each point as follows. For a given $z_0 \in \mathbb{C}$, we

1. run Newton's method with that initial guess,
2. colour the point according to which root it converges to,

One last beautiful idea about Newton's method in higher dimensions.

Consider the problem

$$\text{find } z \in \mathbb{C} \text{ such that } z^3 - 1 = 0.$$

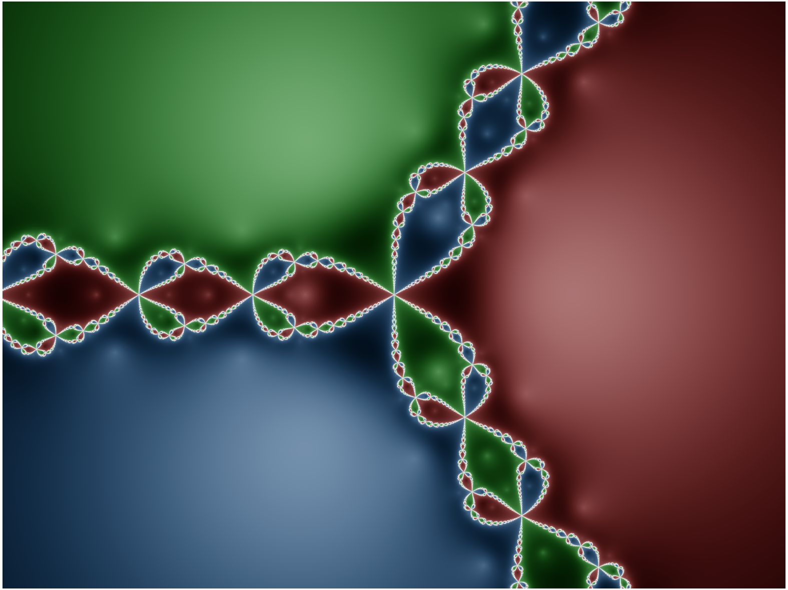
We could also think of this as a problem in $\mathbb{R}^2$.

We know this has three solutions,

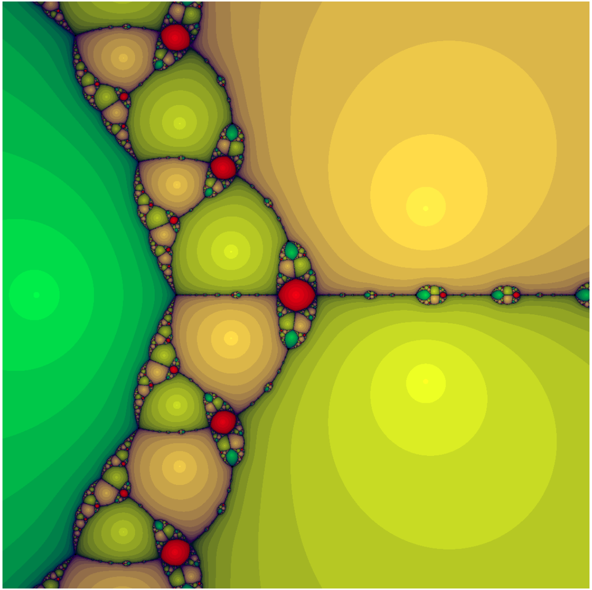
$$z = 1, \quad z = -1/2 + i\sqrt{3}/2, \text{ and } z = -1/2 - i\sqrt{3}/2.$$

Let's take a subset of the complex plane and colour each point as follows. For a given $z_0 \in \mathbb{C}$, we

1. run Newton's method with that initial guess,
2. colour the point according to which root it converges to,
3. shade the colour by how many iterations it took.



The Newton fractal for $z^3 - 1 = 0$.



The Newton fractal for $z^3 - 2z + 2 = 0$.

Some useful websites:

- https://attr.actor/snapshots/dxhdzbzwmylmtywj
- https://newtonfractal.starfree.app/
- https://www.youtube.com/watch?v=-RdOwhmqP5s

Section 9

Algorithms for optimisation problems

In this final lecture, we study how to apply rootfinding ideas to *optimisation*.

In this final lecture, we study how to apply rootfinding ideas to *optimisation*.

Optimisation is fundamental to applied mathematics and engineering. It is also the engine that powers machine learning.

In this final lecture, we study how to apply rootfinding ideas to *optimisation*.

Optimisation is fundamental to applied mathematics and engineering. It is also the engine that powers machine learning.

Nature optimizes. Physical systems tend to a state of minimum energy. The molecules in an isolated chemical system react with each other until the total potential energy of their electrons is minimized. Rays of light follow paths that minimize their travel time.

In this final lecture, we study how to apply rootfinding ideas to *optimisation*.

Optimisation is fundamental to applied mathematics and engineering. It is also the engine that powers machine learning.

Nature optimizes. Physical systems tend to a state of minimum energy. The molecules in an isolated chemical system react with each other until the total potential energy of their electrons is minimized. Rays of light follow paths that minimize their travel time.

The ideas in this lecture are further explored in ASO Calculus of Variations, B6.2 Optimisation for Data Science, and C6.2 Continuous Optimisation.

Optimisation studies how to find an input $\mathbf{x}^\star$ to a function f that achieves a minimal value. (If you want to maximise $f(\mathbf{x})$, just minimise $-f(\mathbf{x})$.)

Optimisation studies how to find an input $\mathbf{x}^\star$ to a function f that achieves a minimal value. (If you want to maximise $f(\mathbf{x})$, just minimise $-f(\mathbf{x})$.)

Let's consider the optimisation problem: given $f \in C^2(\mathbb{R}^N, \mathbb{R})$,

$$\text{find } \mathbf{x}^\star = \operatorname*{argmin}_{\mathbf{x}\in\mathbb{R}^N} f(\mathbf{x}).$$

We assume f is bounded below (e.g. $f(x) = -x^2$ has no min over $\mathbb{R}$).

Optimisation studies how to find an input $\mathbf{x}^\star$ to a function f that achieves a minimal value. (If you want to maximise $f(\mathbf{x})$, just minimise $-f(\mathbf{x})$.)

Let's consider the optimisation problem: given $f \in C^2(\mathbb{R}^N, \mathbb{R})$,

$$\text{find } \mathbf{x}^\star = \operatorname*{argmin}_{\mathbf{x}\in\mathbb{R}^N} f(\mathbf{x}).$$

We assume f is bounded below (e.g. $f(x) = -x^2$ has no min over $\mathbb{R}$).

We want an argument $\mathbf{x}^\star$ that satisfies $f(\mathbf{x}^\star) \leq f(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^N$.

Optimisation studies how to find an input $\mathbf{x}^\star$ to a function f that achieves a minimal value. (If you want to maximise $f(\mathbf{x})$, just minimise $-f(\mathbf{x})$.)

Let's consider the optimisation problem: given $f \in C^2(\mathbb{R}^N, \mathbb{R})$,

$$\text{find } \mathbf{x}^\star = \operatorname*{argmin}_{\mathbf{x}\in\mathbb{R}^N} f(\mathbf{x}).$$

We assume f is bounded below (e.g. $f(x) = -x^2$ has no min over $\mathbb{R}$).

We want an argument $\mathbf{x}^\star$ that satisfies $f(\mathbf{x}^\star) \leq f(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^N$.

This is usually too much to ask for, so instead we satisfy ourselves with *local minima* $\mathbf{x}^\star$ such that there is a neighbourhood $\mathcal{N}$ around $\mathbf{x}^\star$ so that

$$f(\mathbf{x}^\star) \leq f(\mathbf{x}) \text{ for all } \mathbf{x} \in \mathcal{N}.$$

A very profitable line of thinking is to identify conditions that are satisfied at local minima. These are called *optimality conditions*.

A very profitable line of thinking is to identify conditions that are satisfied at local minima. These are called *optimality conditions*.

In our case, the optimality conditions are that the *gradient* $g : \mathbb{R}^N \to \mathbb{R}^N$ is zero at a local minimiser:

$$g(\mathbf{x}^\star) := \nabla f(\mathbf{x}^\star) = Df(x^\star)^\top = \begin{pmatrix} \frac{\partial f}{\partial \mathbf{x}^1}(\mathbf{x}^\star) \\ \vdots \\ \frac{\partial f}{\partial \mathbf{x}^N}(\mathbf{x}^\star) \end{pmatrix} = \mathbf{0}.$$

A very profitable line of thinking is to identify conditions that are satisfied at local minima. These are called *optimality conditions*.

In our case, the optimality conditions are that the *gradient* $g : \mathbb{R}^N \to \mathbb{R}^N$ is zero at a local minimiser:

$$g(\mathbf{x}^\star) := \nabla f(\mathbf{x}^\star) = Df(x^\star)^\top = \begin{pmatrix} \frac{\partial f}{\partial \mathbf{x}^1}(\mathbf{x}^\star) \\ \vdots \\ \frac{\partial f}{\partial \mathbf{x}^N}(\mathbf{x}^\star) \end{pmatrix} = \mathbf{0}.$$

In other words, if we can find roots of g, we can find local minima of f!

A very profitable line of thinking is to identify conditions that are satisfied at local minima. These are called *optimality conditions*.

In our case, the optimality conditions are that the *gradient* $g : \mathbb{R}^N \to \mathbb{R}^N$ is zero at a local minimiser:

$$g(\mathbf{x}^\star) := \nabla f(\mathbf{x}^\star) = Df(x^\star)^\top = \begin{pmatrix} \frac{\partial f}{\partial \mathbf{x}^1}(\mathbf{x}^\star) \\ \vdots \\ \frac{\partial f}{\partial \mathbf{x}^N}(\mathbf{x}^\star) \end{pmatrix} = \mathbf{0}.$$

In other words, if we can find roots of g, we can find local minima of f!

...and local maxima, and saddle points: any point like these satisfying $g(\mathbf{x}) = \mathbf{0}$ is called a *critical point*.

To develop practical optimisation algorithms, we've already relaxed the problem twice:

$$\text{global minimisers} \subset \text{local minimisers} \subset \text{critical points.}$$

To develop practical optimisation algorithms, we've already relaxed the problem twice:

$$\text{global minimisers} \subset \text{local minimisers} \subset \text{critical points}.$$

We will have to be careful, when looking for critical points, to find only the local minimisers we're interested in.

To develop practical optimisation algorithms, we've already relaxed the problem twice:

$$\text{global minimisers} \subset \text{local minimisers} \subset \text{critical points.}$$

We will have to be careful, when looking for critical points, to find only the local minimisers we're interested in.

Local minimisers can be distinguished by studying the *second-order sufficiency conditions*. We won't see these.

To develop practical optimisation algorithms, we've already relaxed the problem twice:

$$\text{global minimisers} \subset \text{local minimisers} \subset \text{critical points}.$$

We will have to be careful, when looking for critical points, to find only the local minimisers we're interested in.

Local minimisers can be distinguished by studying the *second-order sufficiency conditions*. We won't see these.

Finding global minimisers is so hard that it is its own branch of study, *global* optimisation.

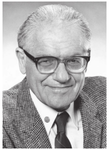
The model problem we're considering in this lecture is quite simplified. In most real optimisation problems, there are *constraints* on the solution:

$$
\begin{aligned}
\min_{\mathbf{x}\in\mathbb{R}^N} \quad & f(\mathbf{x}) \\
\text{subject to} \quad & c_i(\mathbf{x}) \geq 0, \quad i \in \mathcal{I}, \\
& c_e(\mathbf{x}) = 0, \quad i \in \mathcal{E}.
\end{aligned}
$$

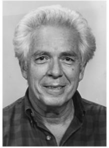
The model problem we're considering in this lecture is quite simplified. In most real optimisation problems, there are *constraints* on the solution:

$$
\begin{aligned}
\min_{\mathbf{x}\in\mathbb{R}^N} \quad & f(\mathbf{x}) \\
\text{subject to} \quad & c_i(\mathbf{x}) \geq 0, \quad i \in \mathcal{I}, \\
& c_e(\mathbf{x}) = 0, \quad i \in \mathcal{E}.
\end{aligned}
$$

For problems with constraints, the optimality conditions are no longer as simple as $\nabla f(\mathbf{x}) = 0$. The optimality conditions for the problem above are known as the *Karush–Kuhn–Tucker* conditions.



William Karush, 1917–1997



Harold Kuhn, 1925–2014

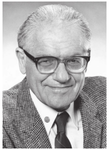
Albert Tucker, 1905–1995

The model problem we're considering in this lecture is quite simplified. In most real optimisation problems, there are *constraints* on the solution:

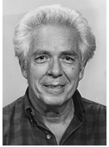
$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^N} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & c_i(\mathbf{x}) \geq 0, \quad i \in \mathcal{I}, \\ & c_e(\mathbf{x}) = 0, \quad i \in \mathcal{E}. \end{aligned}$$

For problems with constraints, the optimality conditions are no longer as simple as $\nabla f(\mathbf{x}) = 0$. The optimality conditions for the problem above are known as the *Karush–Kuhn–Tucker* conditions.

In this lecture we consider the unconstrained problem, since you need to understand that first to attack the constrained one!



William Karush, 1917–1997



Harold Kuhn, 1925–2014

Albert Tucker, 1905–1995

Section 10

Newton's method for optimisation

Let's see what Newton iteration on the gradient looks like. If we take the Jacobian of the gradient, we get the *Hessian matrix*:

$$Hf(\mathbf{a}) = D\nabla f(\mathbf{a}) := \begin{pmatrix} \frac{\partial^2 f}{\partial x^1 x^1}(\mathbf{a}) & \frac{\partial^2 f}{\partial x^1 x^2}(\mathbf{a}) & \cdots & \frac{\partial^2 f}{\partial x^1 x^N}(\mathbf{a}) \\ \frac{\partial^2 f}{\partial x^2 x^1}(\mathbf{a}) & \frac{\partial^2 f}{\partial x^2 x^2}(\mathbf{a}) & \cdots & \frac{\partial^2 f}{\partial x^2 x^N}(\mathbf{a}) \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x^N x^1}(\mathbf{a}) & \frac{\partial^2 f}{\partial x^N x^2}(\mathbf{a}) & \cdots & \frac{\partial^2 f}{\partial x^N x^N}(\mathbf{a}) \end{pmatrix}.$$

Let's see what Newton iteration on the gradient looks like. If we take the Jacobian of the gradient, we get the *Hessian matrix*:

$$Hf(\mathbf{a}) = D\nabla f(\mathbf{a}) := \begin{pmatrix} \frac{\partial^2 f}{\partial x^1 x^1}(\mathbf{a}) & \frac{\partial^2 f}{\partial x^1 x^2}(\mathbf{a}) & \cdots & \frac{\partial^2 f}{\partial x^1 x^N}(\mathbf{a}) \\ \frac{\partial^2 f}{\partial x^2 x^1}(\mathbf{a}) & \frac{\partial^2 f}{\partial x^2 x^2}(\mathbf{a}) & \cdots & \frac{\partial^2 f}{\partial x^2 x^N}(\mathbf{a}) \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x^N x^1}(\mathbf{a}) & \frac{\partial^2 f}{\partial x^N x^2}(\mathbf{a}) & \cdots & \frac{\partial^2 f}{\partial x^N x^N}(\mathbf{a}) \end{pmatrix}.$$

The Hessian is always symmetric for $f \in C^2(\mathbb{R}^N, \mathbb{R})$.

Let's see what Newton iteration on the gradient looks like. If we take the Jacobian of the gradient, we get the *Hessian matrix*:

$$Hf(\mathbf{a}) = D\nabla f(\mathbf{a}) := \begin{pmatrix} \frac{\partial^2 f}{\partial x^1 x^1}(\mathbf{a}) & \frac{\partial^2 f}{\partial x^1 x^2}(\mathbf{a}) & \cdots & \frac{\partial^2 f}{\partial x^1 x^N}(\mathbf{a}) \\ \frac{\partial^2 f}{\partial x^2 x^1}(\mathbf{a}) & \frac{\partial^2 f}{\partial x^2 x^2}(\mathbf{a}) & \cdots & \frac{\partial^2 f}{\partial x^2 x^N}(\mathbf{a}) \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x^N x^1}(\mathbf{a}) & \frac{\partial^2 f}{\partial x^N x^2}(\mathbf{a}) & \cdots & \frac{\partial^2 f}{\partial x^N x^N}(\mathbf{a}) \end{pmatrix}.$$

The Hessian is always symmetric for $f \in C^2(\mathbb{R}^N, \mathbb{R})$.

Applying Newton's method to find roots of $\nabla f(x)$, we get

$$\mathbf{x}_{i+1} = \mathbf{x}_i - [Hf(\mathbf{x}_i)]^{-1} \nabla f(\mathbf{x}_i)$$

Let's see what Newton iteration on the gradient looks like. If we take the Jacobian of the gradient, we get the *Hessian matrix*:

$$Hf(\mathbf{a}) = D\nabla f(\mathbf{a}) := \begin{pmatrix} \frac{\partial^2 f}{\partial x^1 x^1}(\mathbf{a}) & \frac{\partial^2 f}{\partial x^1 x^2}(\mathbf{a}) & \cdots & \frac{\partial^2 f}{\partial x^1 x^N}(\mathbf{a}) \\ \frac{\partial^2 f}{\partial x^2 x^1}(\mathbf{a}) & \frac{\partial^2 f}{\partial x^2 x^2}(\mathbf{a}) & \cdots & \frac{\partial^2 f}{\partial x^2 x^N}(\mathbf{a}) \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x^N x^1}(\mathbf{a}) & \frac{\partial^2 f}{\partial x^N x^2}(\mathbf{a}) & \cdots & \frac{\partial^2 f}{\partial x^N x^N}(\mathbf{a}) \end{pmatrix}.$$

The Hessian is always symmetric for $f \in C^2(\mathbb{R}^N, \mathbb{R})$.

Applying Newton's method to find roots of $\nabla f(x)$, we get

$$\mathbf{x}_{i+1} = \mathbf{x}_i - [Hf(\mathbf{x}_i)]^{-1}\nabla f(\mathbf{x}_i) = x_i - [Dg(\mathbf{x}_i)]^{-1}g(\mathbf{x}_i).$$

There's a nice geometric interpretation to this.

There's a nice geometric interpretation to this.

Suppose we're at iterate $\mathbf{x}_i$ and we'd like to minimise f. We don't know how, so we'll replace f with a local quadratic model:

$$f(\mathbf{x}_i + \delta\mathbf{x}) \approx m(\delta\mathbf{x}) := f(\mathbf{x}_i) + \nabla f(\mathbf{x}_i)^\top \delta\mathbf{x} + \frac{1}{2}\delta\mathbf{x}^\top H f(\mathbf{x}_i)\delta\mathbf{x}.$$

There's a nice geometric interpretation to this.

Suppose we're at iterate $\mathbf{x}_i$ and we'd like to minimise f. We don't know how, so we'll replace f with a local quadratic model:

$$f(\mathbf{x}_i + \delta\mathbf{x}) \approx m(\delta\mathbf{x}) := f(\mathbf{x}_i) + \nabla f(\mathbf{x}_i)^\top \delta\mathbf{x} + \frac{1}{2}\delta\mathbf{x}^\top Hf(\mathbf{x}_i)\delta\mathbf{x}.$$

We can decide what the update $\delta\mathbf{x}$ should be by solving $\nabla m(\delta\mathbf{x}) = 0$, which yields the update

$$\delta\mathbf{x} = -[Hf(\mathbf{x}_i)]^{-1}\nabla f(\mathbf{x}_i).$$

There's a nice geometric interpretation to this.

Suppose we're at iterate $\mathbf{x}_i$ and we'd like to minimise f. We don't know how, so we'll replace f with a local quadratic model:

$$f(\mathbf{x}_i + \delta\mathbf{x}) \approx m(\delta\mathbf{x}) := f(\mathbf{x}_i) + \nabla f(\mathbf{x}_i)^\top \delta\mathbf{x} + \frac{1}{2}\delta\mathbf{x}^\top H f(\mathbf{x}_i)\delta\mathbf{x}.$$

We can decide what the update $\delta\mathbf{x}$ should be by solving $\nabla m(\delta\mathbf{x}) = 0$, which yields the update

$$\delta\mathbf{x} = -[Hf(\mathbf{x}_i)]^{-1}\nabla f(\mathbf{x}_i).$$

So at every step, Newton's method for optimisation approximates the function with a paraboloid, and minimises that.

Section 11

Quasi-Newton methods

Problems with this:

1. We only want to find local *minimisers*.

Problems with this:

1. We only want to find local *minimisers*.

We often have *huge* $N \gg 1$, causing two more difficulties:

2. How do we store $Hf(\mathbf{x}_i)$? (Can't store a full/dense matrix.)
3. How do we solve $Hf(\mathbf{x}_i)\delta\mathbf{x} = -\nabla f(\mathbf{x}_i)$?

Problems with this:

1. We only want to find local *minimisers*.

We often have *huge* $N \gg 1$, causing two more difficulties:

2. How do we store $Hf(\mathbf{x}_i)$? (Can't store a full/dense matrix.)
3. How do we solve $Hf(\mathbf{x}_i)\delta\mathbf{x} = -\nabla f(\mathbf{x}_i)$?

It is often possible to overcome these issues by exploiting some *structure* in the problem. When minimising energy functions in physics, the matrix is usually *sparse*, which can sometimes be exploited to solve the linear system in time $\mathcal{O}(N)$ instead of $\mathcal{O}(N^3)$.

Problems with this:

1. We only want to find local *minimisers*.

We often have *huge* $N \gg 1$, causing two more difficulties:

2. How do we store $Hf(\mathbf{x}_i)$? (Can't store a full/dense matrix.)
3. How do we solve $Hf(\mathbf{x}_i)\delta\mathbf{x} = -\nabla f(\mathbf{x}_i)$?

It is often possible to overcome these issues by exploiting some *structure* in the problem. When minimising energy functions in physics, the matrix is usually *sparse*, which can sometimes be exploited to solve the linear system in time $\mathcal{O}(N)$ instead of $\mathcal{O}(N^3)$.

But for many problems no such nice structure exists (e.g. neural networks).

Problems with this:

1. We only want to find local *minimisers*.

We often have *huge* $N \gg 1$, causing two more difficulties:

2. How do we store $Hf(\mathbf{x}_i)$? (Can't store a full/dense matrix.)
3. How do we solve $Hf(\mathbf{x}_i)\delta\mathbf{x} = -\nabla f(\mathbf{x}_i)$?

It is often possible to overcome these issues by exploiting some *structure* in the problem. When minimising energy functions in physics, the matrix is usually *sparse*, which can sometimes be exploited to solve the linear system in time $\mathcal{O}(N)$ instead of $\mathcal{O}(N^3)$.

But for many problems no such nice structure exists (e.g. neural networks).

The standard practice is to modify the algorithm to

$$\mathbf{x}_{i+1} = \mathbf{x}_i - B_i^{-1}\nabla f(\mathbf{x}_i)$$

for carefully chosen matrices B_i. This is called a *quasi-Newton* scheme.

Here are some choices for B_i:

1. $B_i = Hf(\mathbf{x}_i)$. Newton again. Quadratic convergence, often impractical.

Here are some choices for B_i:

1. $B_i = Hf(\mathbf{x}_i)$. Newton again. Quadratic convergence, often impractical.
2. $B_i = I \in \mathbb{R}^{N\times N}$. Gradient descent. Linear convergence, very slow.

Here are some choices for B_i:

1. $B_i = Hf(\mathbf{x}_i)$. Newton again. Quadratic convergence, often impractical.
2. $B_i = I \in \mathbb{R}^{N \times N}$. Gradient descent. Linear convergence, very slow.

The choice most used in practice is the BFGS algorithm (1970).

Here are some choices for B_i:

1. $B_i = Hf(\mathbf{x}_i)$. Newton again. Quadratic convergence, often impractical.
2. $B_i = I \in \mathbb{R}^{N \times N}$. Gradient descent. Linear convergence, very slow.

The choice most used in practice is the BFGS algorithm (1970).

This builds up an approximation to the Hessian as the iterations proceed.

The BFGS approach demands that the symmetric matrix B_{i+1} satisfy

$$B_{i+1}(\mathbf{x}_{i+1} - \mathbf{x}_i) = \nabla f(\mathbf{x}_{i+1}) - \nabla f(\mathbf{x}_i).$$

which is the higher-order generalisation of the secant method.

The BFGS approach demands that the symmetric matrix B_{i+1} satisfy

$$B_{i+1}(\mathbf{x}_{i+1} - \mathbf{x}_i) = \nabla f(\mathbf{x}_{i+1}) - \nabla f(\mathbf{x}_i).$$

which is the higher-order generalisation of the secant method.

In one dimension, this secant condition is enough to approximate $f''(x_i)$. But in higher dimensions it is not; we have N equations, but $N(N+1)/2$ variables to define B_{i+1}. So how do we fill in the missing information?

The BFGS approach demands that the symmetric matrix B_{i+1} satisfy

$$B_{i+1}(\mathbf{x}_{i+1} - \mathbf{x}_i) = \nabla f(\mathbf{x}_{i+1}) - \nabla f(\mathbf{x}_i).$$

which is the higher-order generalisation of the secant method.

In one dimension, this secant condition is enough to approximate $f''(x_i)$. But in higher dimensions it is not; we have N equations, but $N(N+1)/2$ variables to define B_{i+1}. So how do we fill in the missing information?

BFGS proposed to choose, *among all symmetric matrices satisfying the secant condition, the one whose inverse is closest to* B_i^{-1}:

$$\begin{aligned} B_{i+1} = \underset{B \in \mathbb{R}^{N \times N}}{\text{argmin}} \quad & \|B^{-1} - B_i^{-1}\| \\ \text{subject to} \quad & B = B^\top, \\ & B(\mathbf{x}_{i+1} - \mathbf{x}_i) = \nabla f(\mathbf{x}_{i+1}) - \nabla f(\mathbf{x}_i). \end{aligned}$$

This means we now need to supply B_0.

The BFGS approach demands that the symmetric matrix B_{i+1} satisfy

$$B_{i+1}(\mathbf{x}_{i+1} - \mathbf{x}_i) = \nabla f(\mathbf{x}_{i+1}) - \nabla f(\mathbf{x}_i).$$

which is the higher-order generalisation of the secant method.

In one dimension, this secant condition is enough to approximate $f''(x_i)$. But in higher dimensions it is not; we have N equations, but $N(N+1)/2$ variables to define B_{i+1}. So how do we fill in the missing information?

BFGS proposed to choose, *among all symmetric matrices satisfying the secant condition, the one whose inverse is closest to* B_i^{-1}:

$$\begin{aligned} B_{i+1} = \underset{B \in \mathbb{R}^{N \times N}}{\text{argmin}} \quad & \|B^{-1} - B_i^{-1}\| \\ \text{subject to} \quad & B = B^\top, \\ & B(\mathbf{x}_{i+1} - \mathbf{x}_i) = \nabla f(\mathbf{x}_{i+1}) - \nabla f(\mathbf{x}_i). \end{aligned}$$

This means we now need to supply B_0. With the right choice of norm, this problem has an explicit solution for B_{i+1} and B_{i+1}^{-1}.

No matter the choice of B_i, we want to guarantee that

$$f(\mathbf{x}_{i+1}) < f(\mathbf{x}_i).$$

No matter the choice of B_i, we want to guarantee that

$$f(\mathbf{x}_{i+1}) < f(\mathbf{x}_i).$$

This will be achieved by ensuring that B_i is *positive definite*.

No matter the choice of B_i, we want to guarantee that

$$f(\mathbf{x}_{i+1}) < f(\mathbf{x}_i).$$

This will be achieved by ensuring that B_i is *positive definite*.

Definition (positive-definite)

A matrix $A \in \mathbb{R}^{N \times N}$ is said to be positive-definite if $\mathbf{x}^\top A\mathbf{x} > 0$ for all nonzero $\mathbf{x} \in \mathbb{R}^N$.

No matter the choice of B_i, we want to guarantee that

$$f(\mathbf{x}_{i+1}) < f(\mathbf{x}_i).$$

This will be achieved by ensuring that B_i is *positive definite*.

Definition (positive-definite)

A matrix $A \in \mathbb{R}^{N \times N}$ is said to be positive-definite if $\mathbf{x}^\top A \mathbf{x} > 0$ for all nonzero $\mathbf{x} \in \mathbb{R}^N$. This is equivalent to all of its eigenvalues being positive.

No matter the choice of B_i, we want to guarantee that

$$f(\mathbf{x}_{i+1}) < f(\mathbf{x}_i).$$

This will be achieved by ensuring that B_i is *positive definite*.

Definition (positive-definite)

A matrix $A \in \mathbb{R}^{N \times N}$ is said to be positive-definite if $\mathbf{x}^\top A\mathbf{x} > 0$ for all nonzero $\mathbf{x} \in \mathbb{R}^N$. This is equivalent to all of its eigenvalues being positive.

Diagonal matrices

A diagonal matrix A is positive-definite iff all of its diagonal entries are strictly positive. In this case,

$$\mathbf{x}^T A\mathbf{x} = A_{11}(\mathbf{x}^1)^2 + A_{22}(\mathbf{x}^2)^2 + \cdots + A_{NN}(\mathbf{x}^N)^2 > 0.$$

No matter the choice of B_i, we want to guarantee that

$$f(\mathbf{x}_{i+1}) < f(\mathbf{x}_i).$$

This will be achieved by ensuring that B_i is *positive definite*.

Definition (positive-definite)

A matrix $A \in \mathbb{R}^{N\times N}$ is said to be positive-definite if $\mathbf{x}^\top A\mathbf{x} > 0$ for all nonzero $\mathbf{x} \in \mathbb{R}^N$. This is equivalent to all of its eigenvalues being positive.

Diagonal matrices

A diagonal matrix A is positive-definite iff all of its diagonal entries are strictly positive. In this case,

$$\mathbf{x}^T A\mathbf{x} = A_{11}(\mathbf{x}^1)^2 + A_{22}(\mathbf{x}^2)^2 + \cdots + A_{NN}(\mathbf{x}^N)^2 > 0.$$

BFGS gives a positive-definite Hessian approximation, if B_0 is.

To ensure we satisfy

$$f(\mathbf{x}_{i+1}) < f(\mathbf{x}_i)$$

we modify the iteration

$$\mathbf{x}_{i+1} = \mathbf{x}_i - B_i^{-1} \nabla f(\mathbf{x}_i)$$

to use a *line search*.

To ensure we satisfy

$$f(\mathbf{x}_{i+1}) < f(\mathbf{x}_i)$$

we modify the iteration

$$\mathbf{x}_{i+1} = \mathbf{x}_i - B_i^{-1} \nabla f(\mathbf{x}_i)$$

to use a *line search*.

The basic idea is this. The direction $\mathbf{d}_i = -B_i^{-1} \nabla f(\mathbf{x}_i)$ might point towards a minimum, but we may overshoot if $\|\mathbf{d}_i\|$ gets too large. We fix this by adjusting the magnitude of the step.

To ensure we satisfy

$$f(\mathbf{x}_{i+1}) < f(\mathbf{x}_i)$$

we modify the iteration

$$\mathbf{x}_{i+1} = \mathbf{x}_i - B_i^{-1}\nabla f(\mathbf{x}_i)$$

to use a *line search*.

The basic idea is this. The direction $\mathbf{d}_i = -B_i^{-1}\nabla f(\mathbf{x}_i)$ might point towards a minimum, but we may overshoot if $\|\mathbf{d}_i\|$ gets too large. We fix this by adjusting the magnitude of the step.

Define

$$\phi_i(t) := f(\mathbf{x}_i + t\mathbf{d}_i)$$

and consider its derivative at $t = 0$:

To ensure we satisfy

$$f(\mathbf{x}_{i+1}) < f(\mathbf{x}_i)$$

we modify the iteration

$$\mathbf{x}_{i+1} = \mathbf{x}_i - B_i^{-1} \nabla f(\mathbf{x}_i)$$

to use a *line search*.

The basic idea is this. The direction $\mathbf{d}_i = -B_i^{-1} \nabla f(\mathbf{x}_i)$ might point towards a minimum, but we may overshoot if $\|\mathbf{d}_i\|$ gets too large. We fix this by adjusting the magnitude of the step.

Define

$$\phi_i(t) := f(\mathbf{x}_i + t\mathbf{d}_i)$$

and consider its derivative at $t = 0$:

$$\phi_i'(0) = \nabla f(\mathbf{x}_i + 0\mathbf{d}_i)^\top \mathbf{d}_i$$

To ensure we satisfy

$$f(\mathbf{x}_{i+1}) < f(\mathbf{x}_i)$$

we modify the iteration

$$\mathbf{x}_{i+1} = \mathbf{x}_i - B_i^{-1}\nabla f(\mathbf{x}_i)$$

to use a *line search*.

The basic idea is this. The direction $\mathbf{d}_i = -B_i^{-1}\nabla f(\mathbf{x}_i)$ might point towards a minimum, but we may overshoot if $\|\mathbf{d}_i\|$ gets too large. We fix this by adjusting the magnitude of the step.

Define

$$\phi_i(t) := f(\mathbf{x}_i + t\mathbf{d}_i)$$

and consider its derivative at $t = 0$:

$$\begin{aligned}\phi_i'(0) &= \nabla f(\mathbf{x}_i + 0\mathbf{d}_i)^\top \mathbf{d}_i \\ &= -\nabla f(\mathbf{x}_i)^T B_i^{-1}\nabla f(\mathbf{x}_i)\end{aligned}$$

To ensure we satisfy

$$f(\mathbf{x}_{i+1}) < f(\mathbf{x}_i)$$

we modify the iteration

$$\mathbf{x}_{i+1} = \mathbf{x}_i - B_i^{-1}\nabla f(\mathbf{x}_i)$$

to use a *line search*.

The basic idea is this. The direction $\mathbf{d}_i = -B_i^{-1}\nabla f(\mathbf{x}_i)$ might point towards a minimum, but we may overshoot if $\|\mathbf{d}_i\|$ gets too large. We fix this by adjusting the magnitude of the step.

Define

$$\phi_i(t) := f(\mathbf{x}_i + t\mathbf{d}_i)$$

and consider its derivative at $t = 0$:

$$\begin{aligned}\phi_i'(0) &= \nabla f(\mathbf{x}_i + 0\mathbf{d}_i)^\top \mathbf{d}_i \\ &= -\nabla f(\mathbf{x}_i)^T B_i^{-1}\nabla f(\mathbf{x}_i) \\ &< 0.\end{aligned}$$

Since $\phi_i'(0) < 0$, this means that there exists $t > 0$ such that

$$\phi_i(t) = f(\mathbf{x}_i + t\mathbf{d}_i) < f(\mathbf{x}_i) = \phi_i(0)$$

Since $\phi_i'(0) < 0$, this means that there exists $t > 0$ such that

$$\phi_i(t) = f(\mathbf{x}_i + t\mathbf{d}_i) < f(\mathbf{x}_i) = \phi_i(0)$$

so if we take a small enough step we will decrease f.

Since $\phi_i'(0) < 0$, this means that there exists $t > 0$ such that

$$\phi_i(t) = f(\mathbf{x}_i + t\mathbf{d}_i) < f(\mathbf{x}_i) = \phi_i(0)$$

so if we take a small enough step we will decrease f.

We therefore modify the algorithm to

$$\mathbf{x}_{i+1} = \mathbf{x}_i - t_i^\star B_i^{-1} \nabla f(\mathbf{x}_i),$$

where $t_i^\star$ is an (approximate) minimiser of $\phi(t)$.

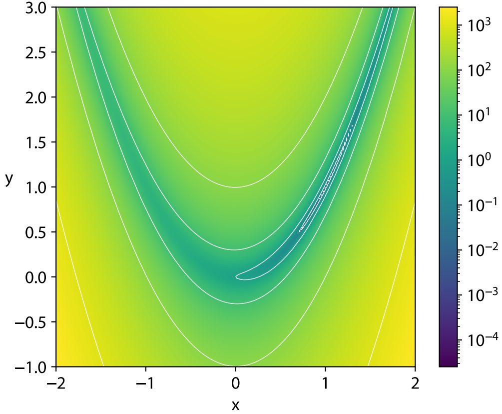
We end the course with a final example. Consider the problem

$$\text{find } (x,y)^\star = \underset{(x,y)\in\mathbb{R}^2}{\operatorname{argmin}} \; f(x,y) := 100(y-x^2)^2 + (1-x)^2.$$

We end the course with a final example. Consider the problem

$$\text{find } (x,y)^\star = \underset{(x,y)\in\mathbb{R}^2}{\operatorname{argmin}} \; f(x,y) := 100(y-x^2)^2 + (1-x)^2.$$

This is the *Rosenbrock* function and has unique minimiser $(x,y)^\star = (1,1)$.



We solve this from $(x_0, y_0) = (-1.2, 1)^\top$ with gradient descent, Newton's method, and BFGS, with a Wolfe line search, until $\|\nabla f(x)\| < 10^{-5}$.

We solve this from $(x_0, y_0) = (-1.2, 1)^\top$ with gradient descent, Newton's method, and BFGS, with a Wolfe line search, until $\|\nabla f(x)\| < 10^{-5}$.

Gradient descent	Newton's method	BFGS
1.827×10^{-4}	3.48×10^{-2}	1.70×10^{-3}
1.826×10^{-4}	1.44×10^{-2}	1.17×10^{-3}
1.824×10^{-4}	1.82×10^{-4}	1.34×10^{-4}
1.823×10^{-4}	1.17×10^{-8}	1.01×10^{-6}

$\|(x, y) - (x, y)^\star\|$ for the last 4 iterations.

We solve this from $(x_0, y_0) = (-1.2, 1)^\top$ with gradient descent, Newton's method, and BFGS, with a Wolfe line search, until $\|\nabla f(x)\| < 10^{-5}$.

Gradient descent	Newton's method	BFGS
1.827×10^{-4}	3.48×10^{-2}	1.70×10^{-3}
1.826×10^{-4}	1.44×10^{-2}	1.17×10^{-3}
1.824×10^{-4}	1.82×10^{-4}	1.34×10^{-4}
1.823×10^{-4}	1.17×10^{-8}	1.01×10^{-6}

$\|(x, y) - (x, y)^\star\|$ for the last 4 iterations.

Gradient descent took 5264 iterations, Newton's method 21, and BFGS 34.