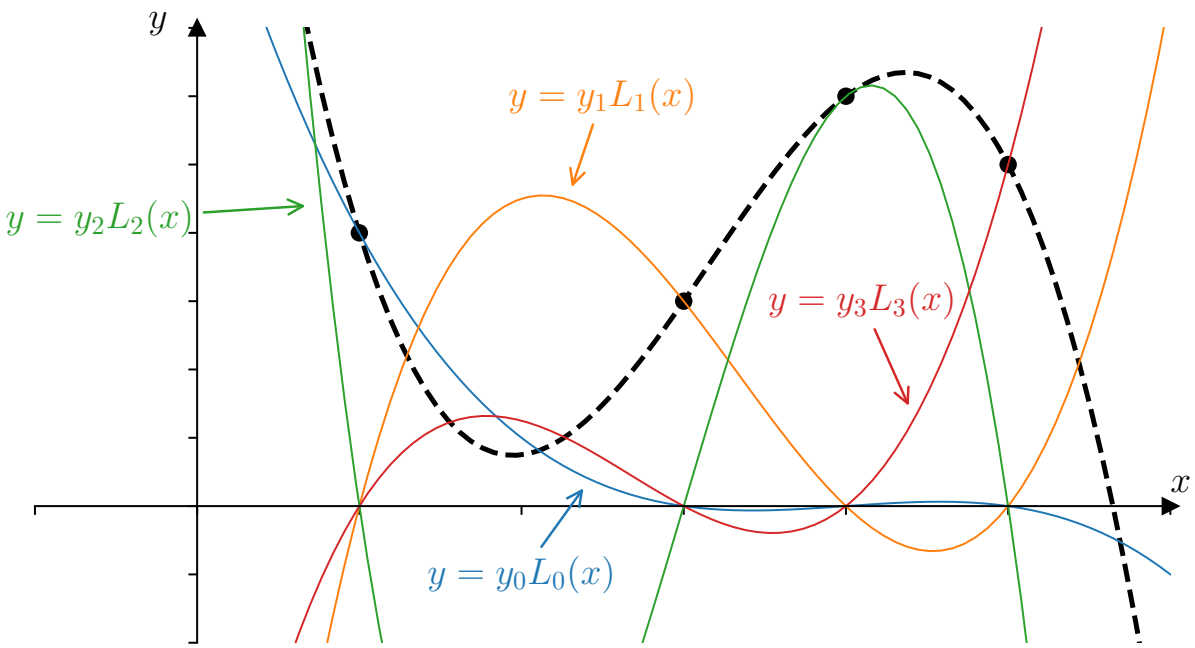
Incomplete Notes On Numerical Methods

Andrew Lehmann
July 15, 2024



Typeset in LaTeX 2_ε.

Contents

Chapter 1

Introduction

Consider some typical mathematics problems that you've seen so far in your mathematics education. For example, an algebra exercise

$$2x = x + 1 \quad \Longrightarrow \quad x = 1, \tag{1.1}$$

or to find the coordinates of the intersection of two lines

$$\left.\begin{aligned} y &= x + 1 \\ y &= 2x \end{aligned}\right\} \quad \Longrightarrow \quad \begin{aligned} x &= 1 \\ y &= 2 \end{aligned}$$

(I hope you noticed this was just a re-interpretation of the previous question). These two problems had solutions that were numbers, but we can have problems with solutions that are functions. For example the differential equations

$$\frac{dy}{dx} = y \quad \Longrightarrow \quad y(x) = Ae^x, \tag{1.2}$$

$$\frac{d^2y}{dx^2} - 2\frac{dy}{dx} + 2y = 0 \quad \Longrightarrow \quad y(x) = e^x \left(A\cos 2x + B\sin 2x\right), \tag{1.3}$$

which are in fact families of functions as solutions, for any constants A and B. Another familiar example is the solution of the quadratic equation

$$\alpha x^2 + \beta x + \gamma \quad \Longrightarrow \quad x = \frac{-\beta \pm \sqrt{\beta^2 - 4\alpha\gamma}}{2\alpha}. \tag{1.4}$$

All these previous examples have solutions that are in some sense simple. They are simple in that they have "closed form" solutions, meaning that you can write the solution as explicit numbers, constants representing abritrary numbers, functions like logs, exponentials, trigono-

metric functions, or as square roots. Such "simple" solutions are called *analytic* solutions. Now consider the following theorem:

ABEL-RUFFINI THEOREM. There is no solution "in radicals" for general 5th order or higher polynomials with arbitrary coefficients.

This theorem is also called "Abel's impossibility theorem" for good reason. A solution "in radicals" means a formula with nth roots in it, like the quadratic formula. Evariste Galois[1] found the simplest polynomial equation with no closed form expression:

$$x^5 - x - 1 = 0.$$

Example 1

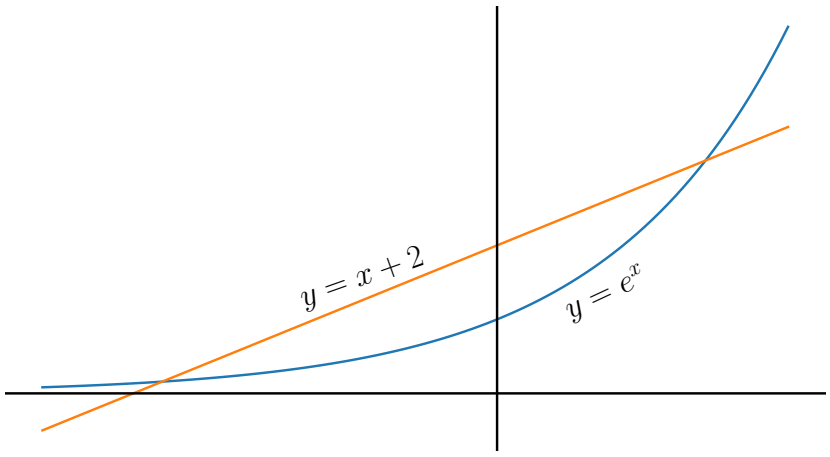


Figure 1.1: Two points of intersection.

The simple equation $e^x = x + 2$ has no analytic solution. You cannot rearrange it algebraically to end up with $x = \ldots$ for some closed form expression. However, it obviously

[1]Evariste Galois (1811-1832), born in Bourg-la-Reine, laid the foundations of group theory, a hugely important topic in abstract algebra with applications in quantum physics. He died at 21 years old in a duel.

has a solution (in fact 2). The solution is the intersection of two lines

$$y = e^x, \quad \text{and}$$
$$y = x + 2.$$

A simple sketch of these two lines, figure 1.1, shows that the two lines must cross at 2 locations. The point is that despite having no *analytic* solution, we will learn *numerical* methods to solving problems like this.

1.1 Square root of 2

How do you know $\sqrt{2} = 1.41421356\ldots$? It turns out there are many methods for finding its decimal expansion. We'll look at a few of these methods now to illustrate some basic principles of numerical analysis.

1.1.1 Heron's method

This method in fact works for computing the square root of any number. Say you want $\sqrt{N}$. Take a first guess $x_0 < N$. This guess could be bigger or smaller than $\sqrt{N}$. If it's smaller

$$x_0 < \sqrt{N} \quad \Longrightarrow \quad \frac{1}{x_0} > \frac{1}{\sqrt{N}} \tag{1.5}$$
$$\Longrightarrow \quad \frac{N}{x_0} > \sqrt{N} \tag{1.6}$$

Similarly

$$x_0 > \sqrt{N} \quad \Longrightarrow \quad \frac{N}{x_0} < \sqrt{N}. \tag{1.7}$$

In both cases the number we want, $\sqrt{N}$, is between x_0 and $\frac{N}{x_0}$. So let's take the next guess as the average of these two

$$x_1 = \text{average}(x_0, \frac{N}{x_0}). \tag{1.8}$$

This guess will constrain $\sqrt{N}$ into a smaller interval between x_1 and $\frac{N}{x_1}$. So we repeat this procedure as much as we want to converge on $\sqrt{N}$. So we have an iterative scheme for finding

the square root of any number, N,

$$x_{i+1} = \frac{x_i + N/x_i}{2} = \frac{x_i}{2} + \frac{N}{2x_i}. \tag{1.9}$$

In the next chapter we will look at *fixed-point analysis*, which studies when iterative schemes stabilise (or not) on fixed points. For example, consider the iterative scheme we just defined for $N = 2$, and set the new iterate to be equal to the previous iterate:

$$x_i = \frac{x_i}{2} + \frac{1}{x_i}. \tag{1.10}$$

Rearranging we see

$$x_i - \frac{x_i}{2} = \frac{1}{x_i} \tag{1.11}$$

$$\frac{x_i}{2} = \frac{1}{x_i} \tag{1.12}$$

$$x_i^2 = 2 \tag{1.13}$$

$$x_i = \sqrt{2} \tag{1.14}$$

Note that this does not mean the scheme will converge on $\sqrt{2}$. It means that if we happen to land on $\sqrt{2}$ on any iteration, then the scheme will stay there.

1.1.2 Theon of Smyrna's method

In this method, we develop an iteration scheme that converges on $\sqrt{2}$. This scheme is given by a ratio

$$x_i = \frac{p_i}{q_i} \quad \text{with} \quad \begin{matrix} p_{i+1} = p_i + 2q_i \\ q_{i+1} = p_i + q_i \end{matrix} \quad \text{and} \quad p_0 = q_0 = 1.$$

This method only works for computing $\sqrt{2}$

1.1.3 Comparison of the methods

Iteration	Theon	Heron
0	1.00000	1.00000
1	1.50000	1.50000
2	1.40000	1.41667
3	1.41667	1.41422
4	1.41379	1.41421
5	1.41429	1.41421
6	1.41420	1.41421
7	1.41422	1.41421
8	1.41421	1.41421
9	1.41421	1.41421

Table 1.1: Estimates of $\sqrt{2}$ for the 2 methods.

In the next chapter we will develop other more general methods, based on approximating the solution to the equation

$$x^2 - 2 = 0$$

with iterative methods that converge on the exact solution. This clearly the $\sqrt{2}$ that we want to approximate.

Chapter 2

Iterative rootfinding methods

Who cares about finding roots to equations? First, let's recall what this means. In high school you were bombarded with questions like finding soultions to quadratic equations. E.g.

$$x^2 + x - 2 = 0. \tag{2.1}$$

The quadratic formula gives you the two values of x

$$x = \frac{-\beta \pm \sqrt{\beta^2 - 5\alpha\gamma}}{2\alpha} \tag{2.2}$$

where in this problem $\alpha = 1$, $\beta = 1$ and $\gamma = -2$. A visual approach shows that the solutions are where the polynomial $y = x^2 + x - 2 = (x - 2)(x + 1)$ intersects with the x-axis:

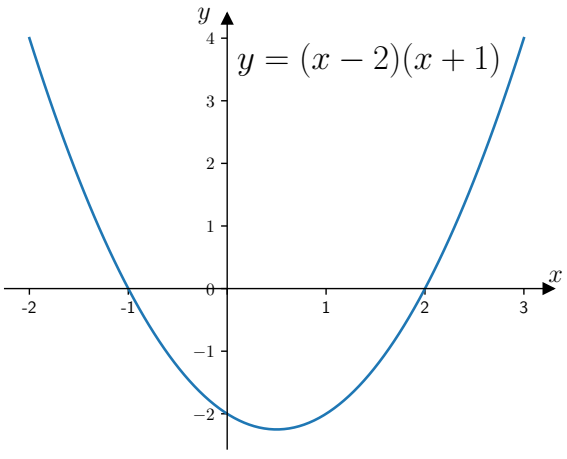


Figure 2.1: Quadratic $y = x^2 + x - 2$.

Or maybe you remember finding the intersection of two lines. E.g. for equations

$$y = x + 1$$
$$y = x^2 - 2$$

Visually we can intuit there will be 2 solutions

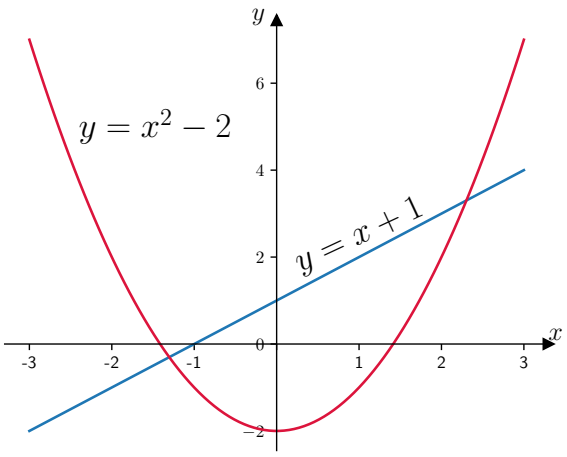


Figure 2.2: Intersection of the lines $y = x + 1$ and $y = x^2 - 2$.

Letting the two equations be simultaneously true leads to the quadratic $x^2 - x - 3 = 0$ and you can use the quadratic formula again.

As mentioned in the introduction, we may want to solve an equation with no analytic solution like $e^x = x + 2$. This can be visualised as the intersection of the lines $y = e^x$ and $y = x + 2$:

We can rearrange the equation so that $e^x - x - 2 = 0$. Now the solutions "zero" this equation.

As a final example, consider the equation $\sin x = \cos x$. Visually we can intuit that there'll be infinite solutions:

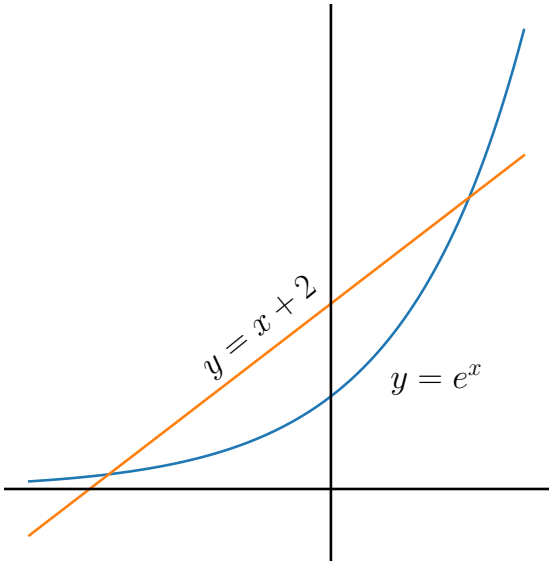


Figure 2.3: Intersection of the lines $y = e^x$ and $y = x + 2$.

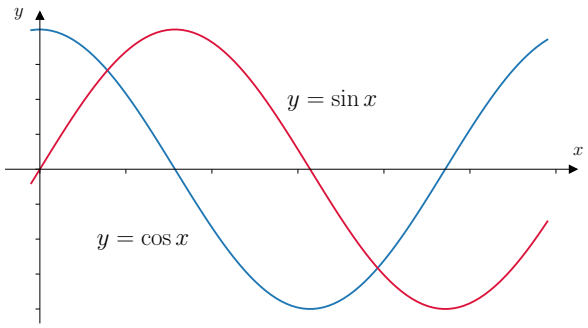


Figure 2.4: Intersection of the lines $y = \sin x$ and $y = \cos x$.

We can also rearrange this equation so that the solutions zero a new equation $\sin x - \cos x = 0$.

The point of all these examples is that in many cases we will have equations of the form $f(x) = 0$, even if we have to force equations to look that way. This is the form of the problems for this chapter, where we will develop techniques to find solutions, called the "zeros of f" or "roots of $f(x) = 0$".

2.1 Bisection

This method is quite simple, we start by making an interval surrounding a root. Then we cut this interval in half. The root must exist in one of the halves. Focus on that half, and cut it further in half. Rinse and repeat. This procedure is sketched out in figure 2.5.

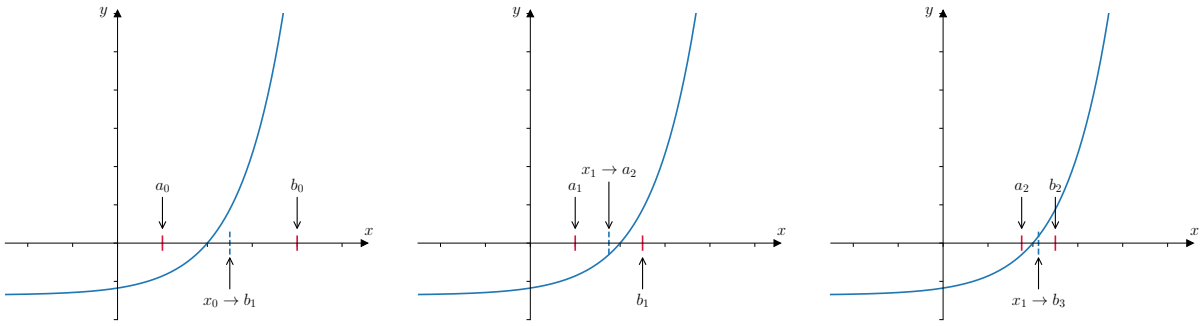


Figure 2.5: Sketch of 3 steps of the bisection method for rootfinding.

This method is guaranteed to converge on a root in the original interval, if one exists. So let's clearly define this algorithm. First we introduce a very useful theorem.

INTERMEDIATE VALUE THEOREM. Consider a function, $f(x)$, that is *continuous* on an interval $[a, b]$. Choose any value y such that $f(a) < y < f(b)$ or $f(b) < y < f(a)$. Then there exists an $x \in (a, b)$ such that $f(x) = y$.

For our purposes, we want to use this theorem to find a root. Roots occur, by definition, when $y = 0$. So we will want to find an a and b such that $f(a) < 0$ and $f(b) > 0$, or $f(a) > 0$ and $f(b) < 0$. We can condense both of these conditions (that the sign of f is different at a and b) into the relationship $f(a)f(b) < 0$. If we have $f(a)f(b) < 0$, then there exists an $x \in (a, b)$ such that $f(x) = 0$. That is, there is a root in the interval (a,b).

Example 2

Let $f(x) = \sin x - \cos x$. $\sin x$ and $\cos x$ are continuous functions for all x, and so f is a continuous function over all reals. Let's search for an interval containing a root by looking at x values that have clean answers with trigonometric functions.

$$f(0) = -1 \qquad\qquad f(\pi) = 1$$

So we have $f(0)f(\pi) = -1 < 0$. Hence by the intermediate value theorem, $f(x) = 0$ has a root in $(0, \pi)$.

Coming back to the bisection method, we can use the interval coming from the Intermediate Value Theorem as an initial interval, given we then know that a root is captured by it. So, the algorithm is as follows.

Bisection method for finding roots

1. Given a_0 and b_0 such that $f(a_0)f(b_0) < 0$, there is a root, $\xi \in (a_0, b_0)$.
2. Define the midpoint
$$x_k = \frac{a_k + b_k}{2}$$
3. If $f(x_k)f(a_k) < 0$, then the root $\xi \in (a_k, x_k)$. So let $a_{k+1} = a_k$ and $b_{k+1} = x_k$. Otherwise (that is, if $f(x_k)f(b_k) < 0$) the root $\xi \in (x_k, b_k)$ and we let $b_{k+1} = b_k$ and $a_{k+1} = x_k$.

The midpoint x_k is the series of values that will converge on the root, $x_k \to \xi$ as $k \to \infty$.

Example 3

To use the bisection method to approximate $\sqrt{2}$, as promised in the previous chapter, we start with the desire to find the number $x = \sqrt{2}$. Then

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

and we have a rootfinding problem, where if we define $f(x) = x^2 - 2$ then the number we want, $\sqrt{2}$, is a root of $f(x) = 0$. Let's use the intermediate value theorem to justify a search interval:

- $f(1) = 1 - 2 = -1 < 0$
- $f(2) = 4 - 2 = 2 > 0$
- $f(x)$ is continuous on $[1, 2]$

therefore by the interediate value theorem there exists a root, ξ ($=\sqrt{2}$), lying in the interval

$(1, 2)$. So we apply the bisection method with $a_0 = 1$ and $b_0 = 2$. Here are a few iterations

$$\begin{aligned}
x_0 = \frac{1+2}{2} = 1.5 &\implies f(x_0)f(a_0) = (1.5^2 - 2)(1^2 - 2) < 0 \\
&\implies \sqrt{2} \in (a_0, x_0) = (1, 1.5) \\
&\implies a_1 = 1 \quad \text{and} \quad b_1 = 1.5 \\
x_1 = \frac{1+1.5}{2} = 1.25 &\implies f(x_1)f(a_1) = (1.25^2 - 2)(1.5^2 - 2) > 0 \\
&\implies \sqrt{2} \in (x_1, b_1) = (1.25, 1.5) \\
&\implies a_2 = 1.25 \quad \text{and} \quad b_2 = 1.5 \\
x_2 = \frac{1.25+1.5}{2} = 1.375 &\implies f(x_2)f(a_2) = (1.25^2 - 2)(1.375^2 - 2) > 0 \\
&\implies \sqrt{2} \in (x_1, b_1) = (1.375, 1.5) \\
&\implies a_2 = 1.375 \quad \text{and} \quad b_2 = 1.5
\end{aligned}$$

and then $x_3 = 1.4375$ and so on.

Convergence of the bisection method

We know initially that the root, ξ, must be inside of the initial guesses for a and b. At each step of the method the interval size is halved, so that the midpoint cannot be a wrong approximation of the root by more than half of the current interval size:

$$\begin{aligned}
|x_0 - \xi| &\leq \frac{|b_0 - a_0|}{2} \\
|x_1 - \xi| &\leq \frac{1}{2}|b_1 - a_1| = \frac{|b_0 - a_0|}{2^2} \\
|x_2 - \xi| &\leq \frac{1}{2}|b_2 - a_2| = \frac{|b_0 - a_0|}{2^3} \\
&\vdots
\end{aligned}$$

You see the pattern, the error at iteration i, $\epsilon_i = |x_i - \xi|$, is limited by the original interval

$$\epsilon_k \leq \frac{|b_0 - a_0|}{2^{k+1}}.$$

The original interval is a fixed number, so it's pretty clear this error limits to zero as the iterations go to infinity

$$\lim_{k\to\infty} \epsilon_k = \lim_{k\to\infty} |x_k - \xi| = 0$$

which proves that x_i converges to the root.

It's natural to ask the inverse question "how many iterations must I perform in order to approximate the root to a given precision?" That is, given a required precision ϵ_g, when can we be sure that the error ϵ_i is less than ϵ. Well this will happen when

$$\begin{aligned} \frac{|b_0 - a_0|}{2^{k+1}} &\leq \epsilon_g \\ \implies k &\geq \frac{1}{\log 2} \log\left(\frac{|b_0 - a_0|}{\epsilon_g}\right) - 1. \end{aligned}$$

2.2 Fixed-point analysis

We can always rearrange the equation $f(x) = 0$ into the form $x - g(x) = 0$ for some newly defined function $g(x)$ that we'll call the *auxiliary function*.

Example 4
If we want to find the intersection points of $y = \sin x$ and $y = \cos x$, then we form

$$\sin x - \cos x = 0$$

so that $f(x) = \sin x - \cos x$. To rearrange, simply force it into the new form by doing nothing (adding zero)

$$x - x + \sin x - \cos x = 0$$

so that $g(x) = x - \sin x + \cos x$.

The auxiliary function let's us define an iterative scheme

$$x_{k+1} = g(x_k).$$

If there is a root of $f(x) = 0$, the new form gives $\xi - g(\xi) = 0 \implies \xi = g(\xi)$. Thinking

this through, the function g takes ξ and gives you back the same number. For this reason ξ is called a *fixed point of g*. *Fixed-points of auxiliary functions are roots of* $f(x) = 0$, so it's worth studying them to solve the original problem we have. If the iteration scheme gives a sequence of values x_1, x_2, x_3, ... that eventually land on ξ, then the sequence will remain unchanged, and the sequence will have converged on the root of $f(x) = 0$.

Now, how can we know if ξ actually exists? To explore this question, consider the following theorem:

BROUWER'S FIXED POINT THEOREM. Suppose we have a function, $g(x)$, that is continuous on an interval $[a, b]$. If the function is bounded by that interval, that is $g(x) \in [a, b]$, for any $x \in [a, b]$, then we are guaranteed the existence of a fixed point in the interval. That is, there exists a $\xi \in [a, b]$ such that $g(\xi) = \xi$.

This theorem is easy enough to understand with a visual demonstration. Consider the function sketched in figure 2.6. In this figure, can you put your pen on the left boundary of the dashed lined square, and draw a continuous line to the right boundary of the square (with $x \in [a, b]$ and $y \in [a, b]$), while staying inside the square and somehow avoid crossing the $y = x$ line? Of course you cannot. There will be *at least* 1 intersection point. At each of the intersection points, $(\xi_1, g(\xi_1))$, $(\xi_2, g(\xi_2))$, etc, you have the property that $\xi_k = g(\xi_k)$, because the intersection point is on both lines $y = x$ and $y = g(x)$ simultaneously.

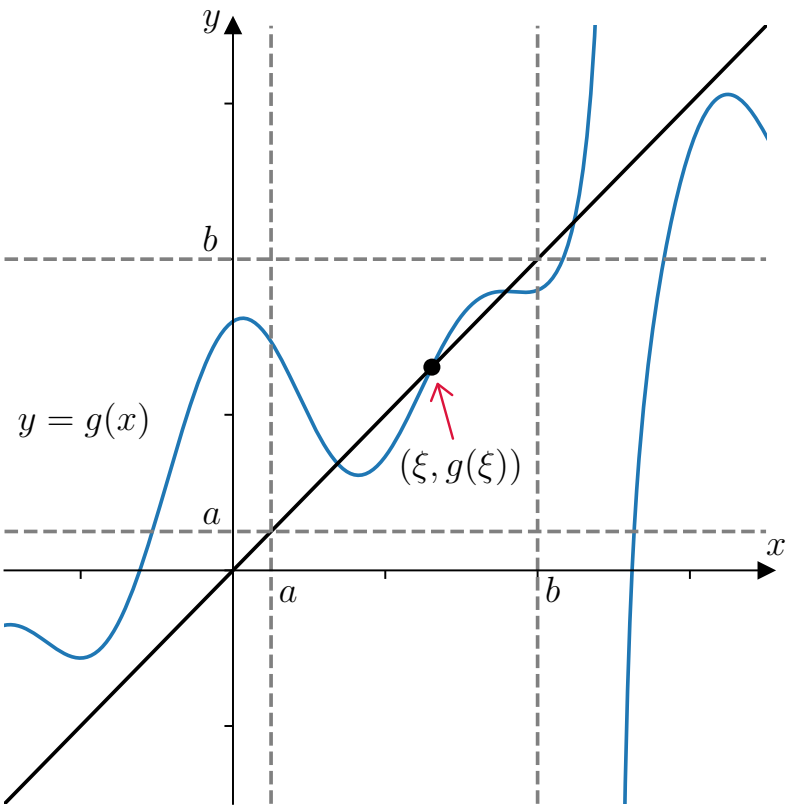


Figure 2.6: Some arbitrary function, $g(x)$, satisfying the conditions of Brouwer's theorem. One point of intersection of $y = g(x)$ and $y = x$ is highlighted.

Example 5

Let's consider the equation $e^x = x + 2$. Solving this equation is equivalent to finding the roots of $f(x) = e^x - x - 2 = 0$. Instead of adding zero as we did when we were being abstract, let's just take the logorithm of both sides of the equation we want to solve:

$$x = \ln(x + 2)$$

so that if we define $g(x) = \ln(x + 2)$ we have the form we want. Thus we define the iterative scheme

$$x_{k+1} = g(x_k) = \ln(x_k + 2).$$

We showed earlier with the Intermediate Value Theorem that a root exists in the interval

$[0, 3]$. So let's make an initial guess $x_0 = 1$, and see where the iteration scheme takes us:

$$\begin{array}{ll} k & x_k \\ 0 & 1 \\ 1 & \ln(1+2) = 1.09861\ldots \\ 2 & \ln(1.09861\cdots+2) = 1.13095\ldots \\ 3 & \ln(1.13095\cdots+2) = 1.14134\ldots \\ & \vdots \\ 7 & \ln(1.14604\cdots+2) = 1.14614\ldots \\ 8 & \ln(1.14614\cdots+2) = 1.14618\ldots \end{array}$$

After the 8th iteration, the result has stopped changing to 4 decimal places, so that's a good enough place to stop. You can see that you can get more accuracy if you just keep iterating. This point, $\xi \sim 1.1462$ is clearly a fixed point of $g(x)$.

Was this fixed point guaranteed by Brouwer's theorem? Let's consider the conditions of the theorem. We need the function to be continuous and enclosed in some square. Consider the interval in which we know there is a root $[0, 3]$

$$\begin{aligned} g(0) = \ln 2, \quad \text{and} \quad 1 < 2 < e \quad &\Longrightarrow \quad 0 < \ln 2 < 1 < 3 \\ g(3) = \ln 5, \quad \text{and} \quad 1 < 5 < e^3 \quad &\Longrightarrow \quad 0 < \ln 5 < 3. \end{aligned}$$

So we know the function starts between $[0, 3]$ and ends between $[0, 3]$. Now the next important fact is that the logarithm is a monotonically increasing function. So there can't be any weird oscillations in the function that let's the function leave the square (up or down) and come back to its end point in the square at $(3, \ln 5)$. This means a sketch of the function in the box looks like:

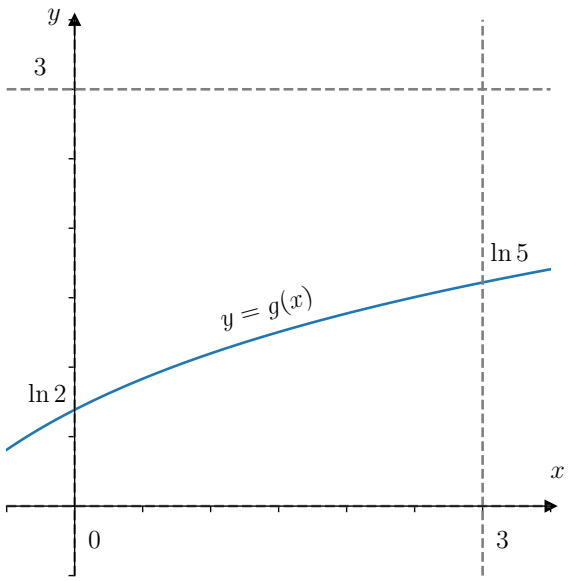


Figure 2.7: $g(x) = \ln(x + 2)$ remaining in a square.

Now, the definition of an auxiliary function is not unique. We could have rearranged the original equation differently

$$e^x = x + 2$$
$$x = e^x - 2$$

giving us a different auxiliary function $h(x) = e^x - 2$ so that $x = h(x)$. Now if we try to search in that same square, in the interval $[0, 3]$

$$h(0) = -1 < 0$$
$$h(3) = e^3 - 2 > 3.$$

So we can say that it is *not true* that $h(x) \in [0, 3]$ for every $x \in [0, 3]$. We can't satisfy one of the conditions in Brouwer's theorem, so we can't use it. However, if you sketch a graph of $y = h(x)$ and $y = x$, you will see the lines do in fact intersect in this square, and it definitely has a fixed-point in that region.

Note the important lesson from that example, we proved with Brouwer's theorem that there is a fixed point of g, meaning that if we can find this fixed point we have the root of

$f(x) = 0$. We could not find a fixed point of $h(x)$. If we never defined $g(x)$, and we only thought of the second formulation, our failure to satisfy Brouwer's thorem *does not* mean that there is no root of $f(x) = 0$ in that interval. Only that we don't know one way or the other. So the lesson is to either choose a different interval to try to find a square in which your $g(x)$ does satisfy Brouwer's theorem, or to find a different rearrangement of $f(x) = 0$ to define a $g(x)$ which works for us.

2.2.1 Fixed-point iteration figures

Let's now look at an important tool for visualising the fixed points: fixed-point iteration figures. The idea is simple:

1. Graph the function $g(x)$;
2. Graph the line $y = x$;
3. For each iterate graph:
   - the vertical line from $(x, y) = (x_k, x_k)$ to $(x, y) = (x_k, g(x_k))$.
   - the horizontal line from $(x, y) = (x_k, g(x_k))$ to $(x, y) = (x_{k+1}, x_{k+1})$ (i.e. until it intersects the $y = x$ lines).

An example of these steps is sketched in figure 2.8, where we represent the different iterations for the scheme $x_k = \log(x_k + 2)$.

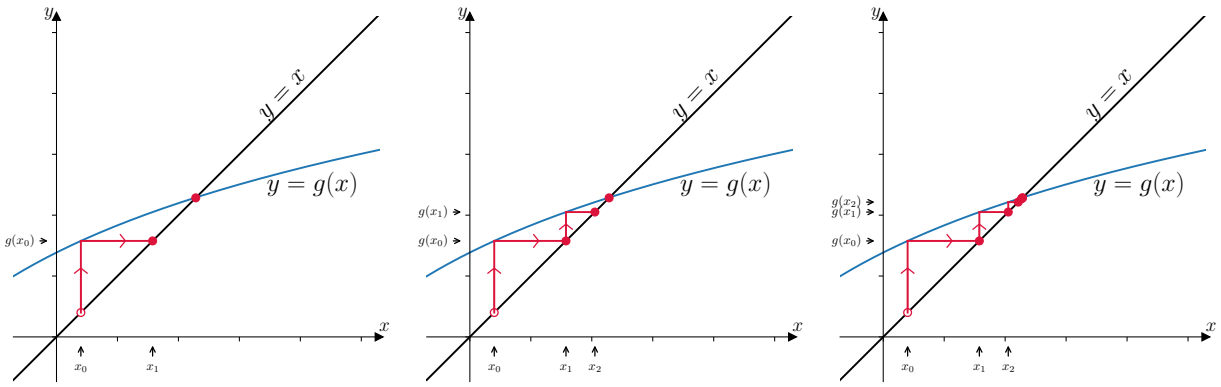


Figure 2.8: Fixed-point iteration figure step by step.

These iteration figures help us understand why an iterative scheme converges on a fixed-point, which we will call a *stable fixed-point*, or why it might diverge from a fixed-point, which we call an *unstable fixed-point*.

Example 6

Let's consider again the equation $e^x = x + 2$, but we rearrange to give the iterative scheme that did not satisfy Brouwer's theorem in the previous example:

$$x = h(x) = e^x - 2.$$

If you sketch the iteration figure accurately enough, shown in figure 2.9, you will see that no matter how close to the intersection point you initially guess x_0 either on the left or right, your lines will be "pushed" away from it.

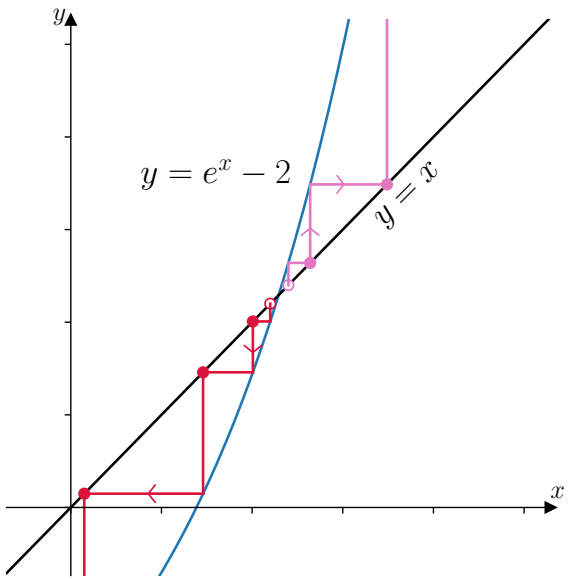


Figure 2.9: Fixed-point iteration figure for $h(x) = e^x - 2$.

So not only can we not use Brouwer's theorem (on the interval $[0, 3]$) for this fixed-point, we also suspect that its fixed point is *unstable*.

These iteration figures can give you a rough expectation for whether a fixed-point will be stable or unstable. The next theorem clarifies this point.

THEOREM: FIXED-POINT STABILITY. Consider an iteration scheme generated with the function $g(x)$ and one of its fixed-points ξ. ξ is a *stable fixed-point* if the gradient of nearby points is shallower than 1, i.e. $|g'(x \sim \xi)| < 1$, and it is an *unstable fixed-point* if the gradient of nearby points is steeper than 1, i.e. $|g'(x \sim \xi)| > 1$.

This theorem is visually displayed in figure 2.10. I think you can convince yourself that whenever you have a negative gradient the iterations will spiral, converging or diverging, around the fixed-point, and whenever you have a positive gradient the iterations will stay on one side, converging or diverging, of the fixed-point. Let's learn to use this stability theorem in practice with an example.

Example 7

For the function $f(x) = e^x - x - 2$, if we define $g(x) = \ln(x + 2)$, then fixed-points of g are roots of $f(x) = 0$. Let's consider the derivative near the fixed-point, that we previously proved must exist in the interval $[0, 3]$.

$$g'(x) = \frac{1}{x+2}$$

At the end points of the interval, the slopes are

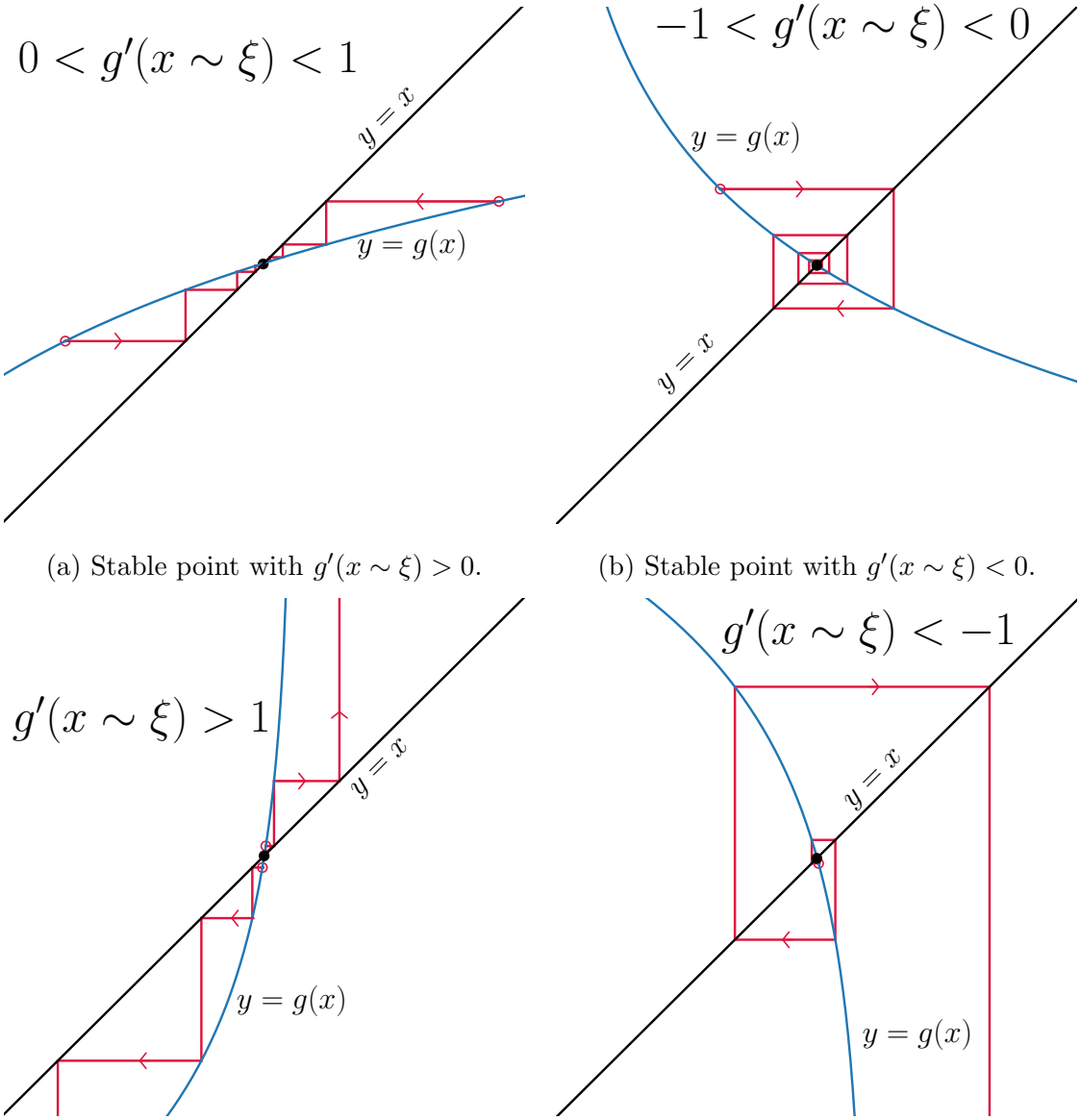
$$g'(0) = 1/2 < 1 \qquad\qquad g'(3) = 1/5 < 1.$$

Additionally, $g'(x)$ is clearly a monotonically decreasing function. Therefore $g'(x) \in [1/5, 1/2]$ for all $x \in [0, 3]$. Since we know the fixed-point is in this interval, we can confidently say $|g'(x)| < 1$ near the point ξ, and hence it is a stable fixed-point.

Now, there is a second fixed-point, we always knew that by looking at the sketch. We can localise it with the Intermediate Value Theorem. We have

$$f(-1) = e^{-1} - 1 = \frac{1-e}{e} < 0$$
$$f(-2) = e^{-2} > 0$$

and hence there must be a root of $f(x) = 0$ in the interval $[-2, -1]$, call it ξ_-. Remember, fixed-points of g are roots of $f(x) = 0$. This works both ways: roots of $f(x) = 0$ are fixed-points of g. But, for this particular representation, $g(x)$ is not defined at -2. However, the gradient has a one-sided limiting value, so we can consider the stability by looking at these



(a) Stable point with $g'(x \sim \xi) > 0$.

(b) Stable point with $g'(x \sim \xi) < 0$.

(c) Unstable point with $g'(x \sim \xi) > 0$.

(d) Unstable point with $g'(x \sim \xi) < 0$.

Figure 2.10: Stable and unstable fixed points (filled black circles). Open red circles are initial guesses.

slopes

$$g'(-1) = 1, \text{but also } \lim_{x \to -1^-} g(x) = 1^+$$
$$\lim_{x \to -2^+} g(x) = +\infty$$

These two facts, along with the monotonic nature of $g'(x)$, tell us that $|g'(x)| > 1$ at every point in the interval $[-2, -1]$. Therefore ξ_- is an unstable fixed-point.

This example proves fairly rigorously that $g(x) = \ln(x + 2)$ provides us with an iterative scheme to converge on the positive root of $f(x) = 0$, but cannot converge on the negative root. What if we look at the alternative iterative scheme we devised earlier by defining $h(x) = e^x - 2$?

Example 8

For the function $f(x) = e^x - x - 2$, if we define $h(x) = e^x - 2$, then fixed-points of h are roots of $f(x) = 0$. We know a fixed-point, ξ_-, exists in $[-2, -1]$, and the slopes at the end points are

$$h'(x) = e^x$$
$$h'(-2) = e^{-2} < 1$$
$$h'(-1) = e^{-1} < 1.$$

e^x is monotonically increasing, so the slopes must always be less than 1 in the interval. Hence $|h'(x)| < 1$ near the point ξ_- and the fixed-point is stable. Thus the iterative scheme

$$x_k = h(x_k) = e^{x_k} - 2$$

will converge on ξ_- given an initial guess that is close enough. So we can find this root, let's

guess $x_0 = -1$:

$$\begin{array}{ll} k & x_k \\ 0 & -1 \\ 1 & e^{-1} - 2 = -1.63212\ldots \\ 2 & e^{-1.63212\ldots} - 2 = -1.80449\ldots \\ 3 & e^{-1.80449\ldots} - 2 = -1.83544\ldots \\ & \vdots \\ 7 & e^{-1.84138\ldots} - 2 = -1.84140\ldots \\ 8 & e^{-1.84140\ldots} - 2 = -1.84141\ldots \end{array}$$

and so we see that the iterations of x_k get closer and closer to a number, which has stopped changing in the 4th decimal place by the 8th iteration. The second root of $f(x) = 0$ is thus $\xi_- \sim -1.8414$.

So we see that this alternative formulation of the iterative scheme let's us converge on the negative root. What does it do near the positive root? Looking at the slopes for the interval $[1, 3]$ (remember we know the fixed-point is at $\xi \sim 1.1462$)

$$h'(1) = e^1 > 1$$
$$h'(3) = e^3 > 1.$$

So with the monotonic fact, the slope must always be too steep near this fixed point. Hence it is unstable.

These two examples show that the stability of a fixed-point is a property of the auxiliary function, and not of the original function $f(x)$.

2.3 Newton-Raphson methods

Start with the general equation of an iterative scheme, $x = g(x)$, which is supposed to find roots of $f(x) = 0$. Previously we arbitrarily defined $g(x)$ by manipulation of $f(x)$. Here we will be more systematic.

Let $g(x) = x - \phi(x)f(x)$ for *any* function $\phi(x)$ as long as $0 < |\phi(x)| < \infty$ in an interval

$[a, b]$ containing a root, call it ξ. In other words, $\phi(x)$ must be continuous in this interval. Since ξ is a root, we have $f(\xi) = 0$ and hence

$$g(\xi) = \xi - \phi(\xi)\cancel{f(\xi)} = \xi.$$

This shows that any zero of f is a fixed point of this g auxiliary function. For the other direction

$$\begin{aligned} g(\xi) = \xi \quad &\implies \quad \xi - \phi(\xi)f(\xi) = \xi \\ &\implies \phi(\xi)f(\xi) = 0 \end{aligned}$$

and since $\phi(\xi) \neq 0$ in this interval, we must have $f(\xi) = 0$. So any fixed point of g is also a zero of f. Thus $x_{k+1} = g(x_k) = x_k - \phi(x_k)f(x_k)$ defines an iterative scheme for finding roots of $f(x) = 0$. Different choices of the function $\phi(x)$ give different methods. We will highlight some of these choices in increasing order of complexity.

2.3.1 Chord method

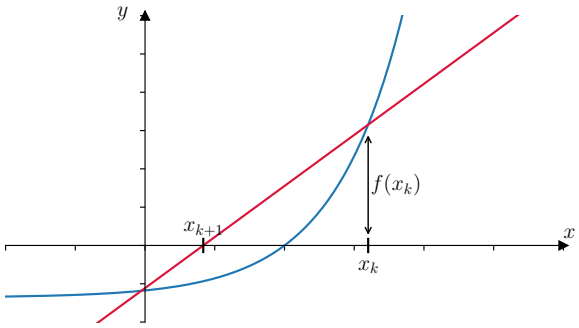
In the simplest method we simply choose a constant function $\phi(x) = \alpha \neq 0$, giving the auxiliary function

$$\boxed{g(x) = x - \alpha f(x) \quad \text{or} \quad x_{k+1} = x_k - \alpha f(x_k) \quad \text{(Chord method)}}$$

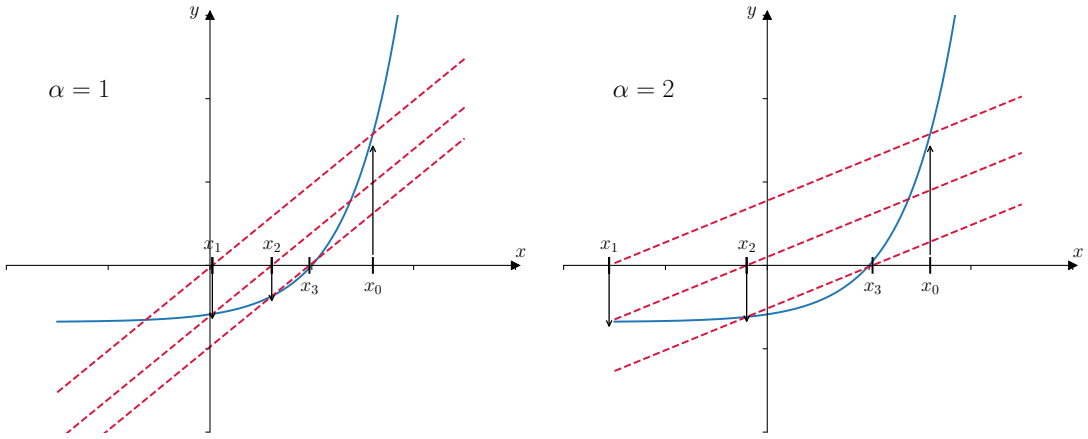
This iterative scheme can be rearranged to give the constant

$$\alpha = \frac{x_k - x_{k+1}}{f(x_k)} \implies \frac{1}{\alpha} = \frac{f(x_k) - 0}{x_k - x_{k+1}}$$

where we recognise that $1/\alpha$ is therefore the slope of a straight line with rise $f(x_k)$ and run $x_k - x_{k+1}$. This gives the following picture



showing that the iteration scheme gives an x-axis intercept as the next approximation of the root. If we take multiple iterations, and two different values of α, we see the following



showing that the choice of α fixes a slope that projects from the function (blue curve) at the previous approximation of the root, $(x_k, f(x_k))$, to the x-axis $(x_{k+1}, 0)$. This inspires the question, what is the optimal slope to choose in order to converge faster on the root? (Not to mention, do we know that this method will converge in all cases?)

2.3.2 Newton's method

Recall from the fixed-point analysis that the stability of a fixed point depends on the derivative of the auxiliary function near the fixed point. There was a criterion that required $|g'(x)| < 1$ for x near the root. Well it's not hard to see that if this derivative is 0 we will have immediate convergence.

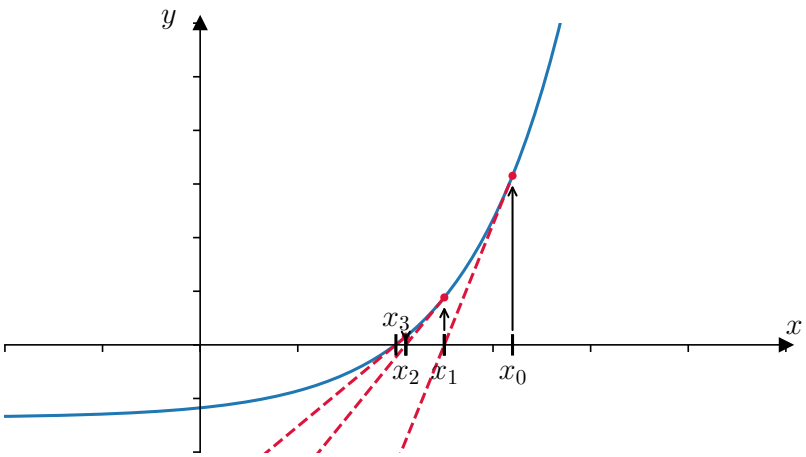
So take the Chord method, $x_{k+1} = g(x_k) = x_k - \alpha f(x_k)$ but imagine changing the constant

α at every iteration, forcing $g'(x_k)$ to be zero. This would give

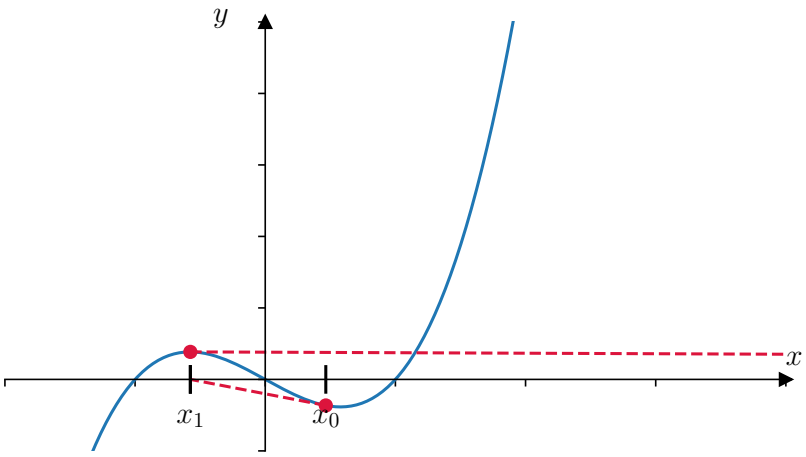
$$1 - \alpha_k f'(x_k) = 0 \quad \Longrightarrow \quad \alpha_k = \frac{1}{f'(x_k)}$$

which gives us a new iteration scheme

$$\boxed{x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \quad \text{(Newton's method)}}$$



Newton's method is a powerful method for finding roots, usually reaching a given precision faster than the chord method due to it's adaptive nature. There is, however, a problem. What happens if an iteration happens to land at a stationary point of the function $f(x_k)$? At this point the slope is zero, $f'(x_k) = 0$, and the next iteration is not defined. This situation is shown below



Example 9

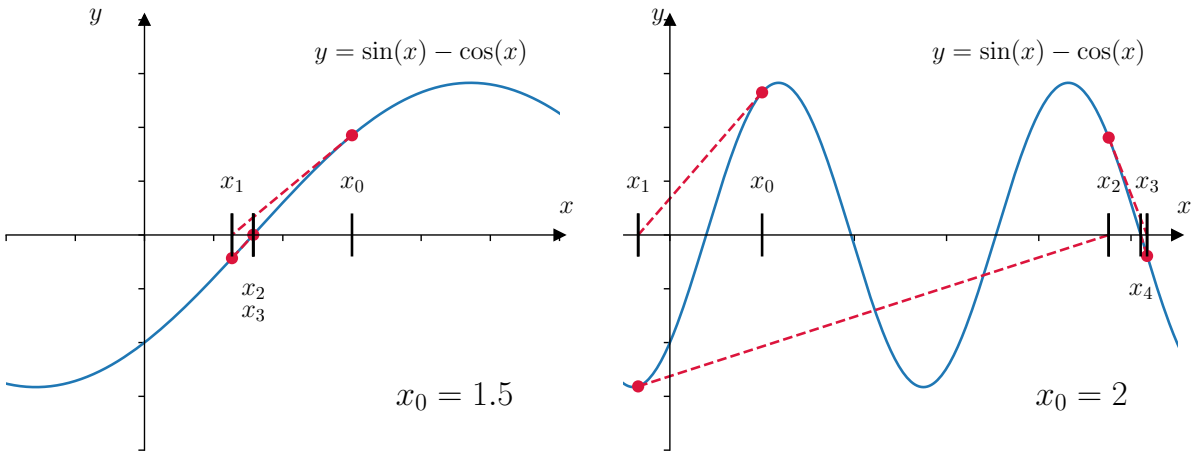
Let's use Newton's method to find some intersection points for $\sin x = \cos x$. We thus define $f(x) = \sin x - \cos x$ so that the intersection points are now zeros of this function f and we have forced a rootfinding problem. We need the derivative $f'(x) = \cos x + \sin x$ and the iterative scheme is therefore

$$x_{k+1} = x_k - \frac{\sin x_k - \cos x_k}{\cos x_k + \sin x_k}$$

This scheme is in fact well defined for any $x \in \mathbb{R}$ since $sinx - cosx$ is never zero. If we make an initial guess at $x_0 = 1.5$, the result stabilises in the 4th decimal place after just 4 iterations on $x_4 = 0.7854$. Whereas if we start at $x_0 = 2$ the result stabilises, to 4 decimal places, after 6 iterations, and at a different root! These iterations are shown in the following table:

k	x_k	x_k
0	1.5000	2.0000
1	0.6324	−0.6877
2	0.7866	9.5160
3	0.7854	10.3484
4	0.7854	10.2092
5	0.7854	10.2102
6	0.7854	10.2102

and on a figure



we can see how wildly the second initial guess jumps around.

2.3.3 Secant method

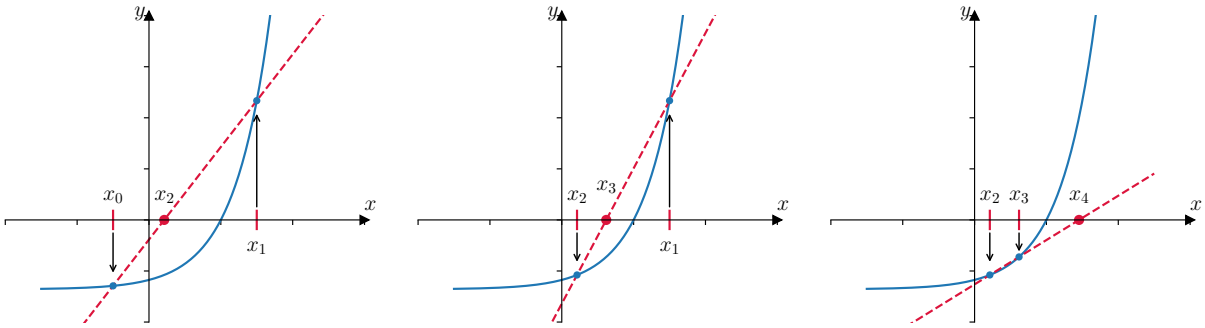
Newton's method requires the calculation of a derivative $f'(x)$ to define an iterative scheme. If the derivative is analytic (for example $(\cos x)' = \sin x$) then this is easy. But there can be situations where f is not known analytically, and so neither is its derivative. In the secant method we follow Newton's method but make an approximation for the derivative:

$$\boxed{x_{k+1} = x_k - \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} f(x_k) \quad \text{(Secant method)}}$$

We have approximated the derivate with

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

which will be justified in a later chapter. What you can recognise is that to calculated x_{k+1} we need the previous two iterations. So we must start with two initial guesses. The method is illustrated below for 3 iterations.



Notice that the approximation doesn't always stay on the same side of the function (blue curve), and in fact x_4 is even further away from the root than x_3. Sometimes it may take several iterations to settle down.

2.4 Comparison of methods

For any of the methods we have studied, consider the error at each iteration $\epsilon_k = |x_k - \xi|$ where ξ is a root. If we find that the following limit gives a constant

$$\lim_{k\to\infty} \frac{\epsilon_{k+1}}{\epsilon_k^p} = C$$

for some constant $C > 0$, then we say the method has *order of convergence* p. Note that p is not forced to be an integer. This characterizes a rate of convergence and therefore lets us compare which method will reach a certain precision faster than another.

Bisection method

We showed that the error in the bisection method is bounded

$$\epsilon_k \leq \frac{|b_0 - a_0|}{2^k}.$$

$$\lim_{k\to\infty} \left(\frac{\frac{|b_0-a_0|}{2^{k+1}}}{\frac{|b_0-a_0|}{2^k}} \right) = \lim_{k\to\infty} \frac{1}{2} = \frac{1}{2}$$

Newton method

Secant method

Chapter 3

Polynomial Interpolation

Imagine we were given a set of $n+1$ data points $\{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$ as shown in figure 3.1.

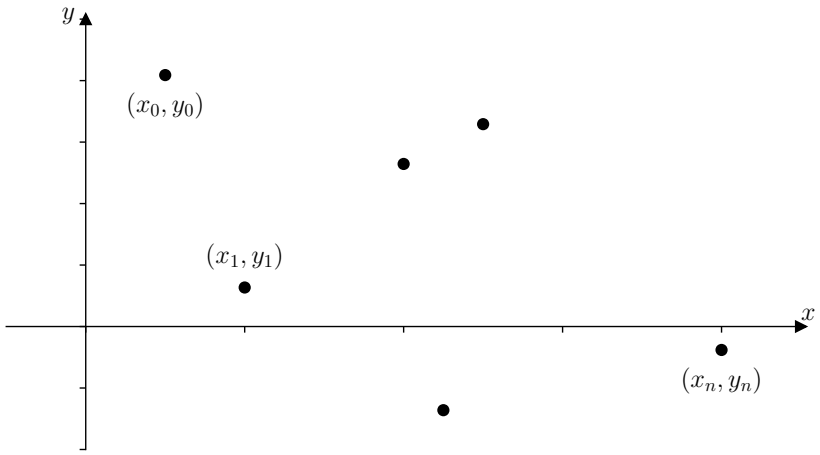


Figure 3.1: Scatter of $n+1$ points that we wish to fit.

We may want to estimate what happens between the points. In this case it will be useful to have a function that agrees with the given data, but is defined for *any* x whatsoever. In this chapter we will consider polynomials to fill this role. Recall the definition of a polynomial of degree n

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x_n^n = \sum_{k=0}^{n} a_k x^k.$$

As another motivation, we may be given a function $f(x)$ instead of data points. It can be computationally useful to estimate this function as a polynomial. Why? Maybe the function

doesn't have a closed form. Maybe it takes a lot of computational power to give $f(x)$ at any particular x. In such a case you can compute a finite set of points $\{(x_0, f(x_0)), (x_1, f(x_1)), \ldots, (x_n, f(x_n))\}$ that we can then use a polynomial to fit. The polynomial could then be used at a much reduced computational expense. On top of that, polynomials are analytically differentiable, which can be useful.

3.1 Piece-wise linear fit

We start with the simplest function that agrees with a set of data points: the *piece-wise linear* fit that simply connects all the data points by straight lines, as sketched in figure 3.2.

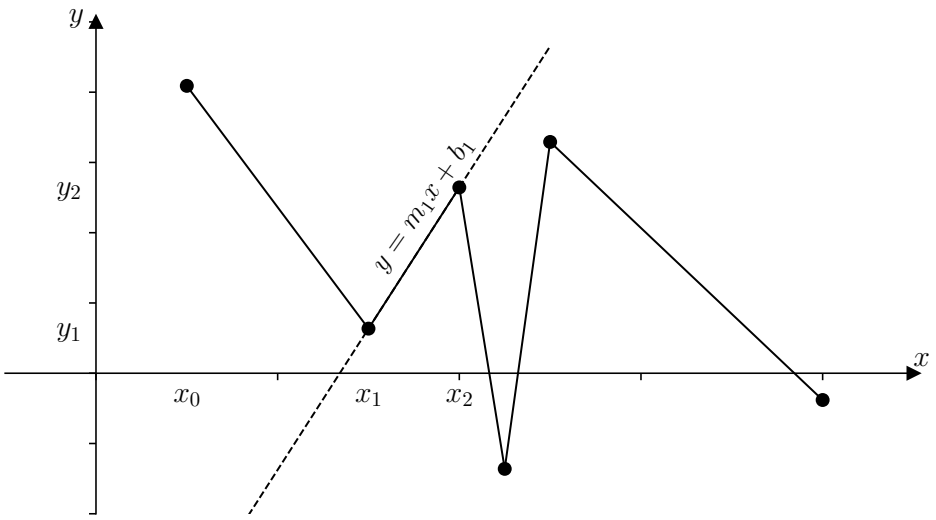


Figure 3.2: Sketch of a piece-wise linear fit to some data points. The straight line connecting the 2nd and 3rd data points is highlighted.

It's necessary to order the data points in increasing x coordinate, or else the resulting fit will be multi-valued, and thus not well defined. Then, for each pair of adjacent data points (x_k, y_k) and (x_{k+1}, y_{k+1}) we must define a straight line equation $y = m_k x + b_k$ for the interval between these points $x \in [x_k, x_{k+1}]$. The slope is given by "rise over run" for these two points

$$\boxed{m_k = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}}$$

and then we get the y-intercept by using the straight line equation at either of the two data

points

$$\begin{aligned} y(x_k) = y_k &= m_k x_k + b_k \\ \implies & \boxed{b_k = y_k - m_k x_k} \\ (\text{or } & b_k = y_{k+1} - m_k x_{k+1}). \end{aligned}$$

As defined above we have a function *between* the data points. In principle we can just extend the first straight line infinitely to the left, $(-\infty, x_0]$, and also the last straight line infinitely to the right, $[x_n, \infty)$, in order to have a function defined for all $x \in \mathbb{R}$. Usually this is not a good idea.

Example 10
Find the piece-wise linear fit to the data: $\{(1,1), (2,3), (3,2), (4,3)\}$.

For data points $(1,1)$, $(2,3)$:

$$\begin{aligned} m_1 &= \frac{3-1}{2-1} = 2 \\ b_1 &= 1 - 2 \times 1 = -1 \\ \implies & y = 2x - 1 \end{aligned}$$

For data points $(2,3)$, $(3,2)$:

$$\begin{aligned} m_1 &= \frac{2-3}{3-2} = -1 \\ b_1 &= 3 - -1 \times 2 = 5 \\ \implies & y = -x + 5 \end{aligned}$$

For data points $(3,2)$, $(4,3)$:

$$\begin{aligned} m_1 &= \frac{3-2}{4-3} = 1 \\ b_1 &= 2 - 1 \times 3 = -1 \\ \implies & y = x - 1 \end{aligned}$$

So we can define the function on any x between the first and last data point

$$y(x) = \begin{cases} 2x - 1 & \text{for } x \in [1, 2] \\ -x + 5 & \text{for } x \in [2, 3] \\ x - 1 & \text{for } x \in [3, 4] \end{cases}$$

3.2 Lagrangian interpolation

In this method, we fit a smooth polynomial through the scattering of points. Recall the figure 3.1, where we have $n + 1$ pairs of numbers (x_k, y_k). Let's start with a theorem

LAGRANGE INTERPOLATION THEOREM. Given the $n+1$ unique points (x_0, y_0), (x_1, y_1), ..., (x_n, y_n), there is a unique polynomial of degree at most n passing through each point. That is,

$$\exists p_n(x) \in \mathcal{P}_n$$
$$\text{such that}$$
$$p_n(x_k) = y_k \quad \forall k.$$

We'll spend the rest of this section constructing this unique polynomial, called the *Lagrange polynomial*.

First let's consider a couple of trivial cases. If we have just 1 point, (x_0, y_0), then the degree 0 polynomial

$$p_0(x) = y_0$$

will pass through the point. Though there are infinite straight lines that could go through this single point, this horizontal line is the only *degree 0 polynomial*. When we have 2 points (x_0, y_0) and (x_1, y_1) then the degree 1 polynomial Lagrange polynomial is just the straight

line joining the two

$$p_1(x) = mx + b$$
$$\text{with} \quad m = \frac{y_1 - y_0}{x_1 - x_0} \quad \text{and} \quad b = mx_0 - y_0.$$

These two trivial cases are shown in figure 3.3.

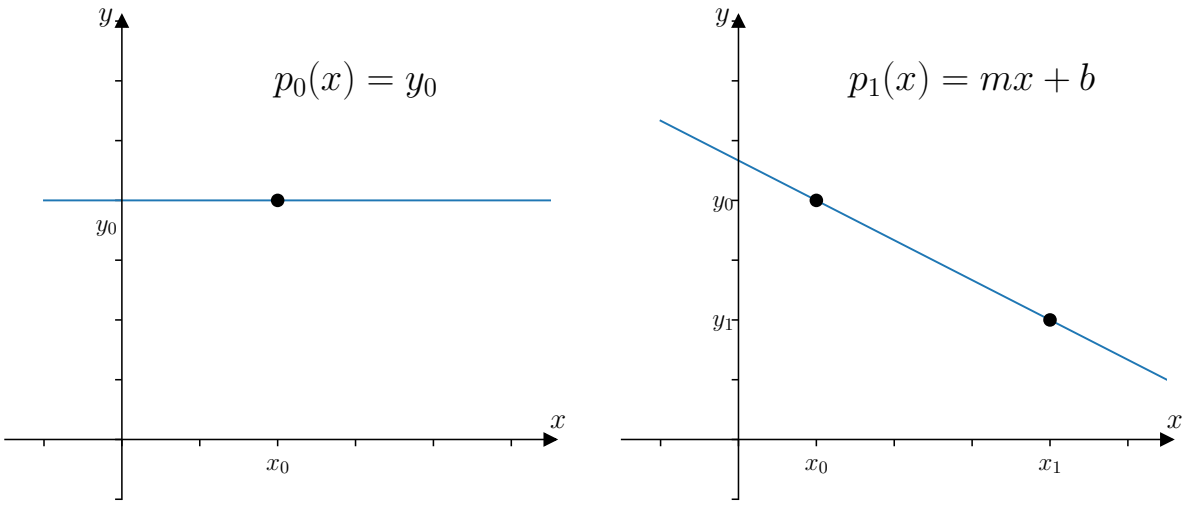


Figure 3.3: Trivial Lagrange polynomials for $n = 0$ and $n = 1$.

Ok, let's construct this unique polynomial for when we have more than 2 points. First, we choose one of the points, so (x_k, y_k), and construct a polynomial, call it $L_k(x)$, that passes through 1 at this x, and goes through zero at all the other positions x_i for $i \neq k$. That is, $L_k(x)$ must pass through the points $(x_k, 1)$ and $(x_i, 0)$ for all $i \neq k$. This polynomial is sketched in figure 3.4.

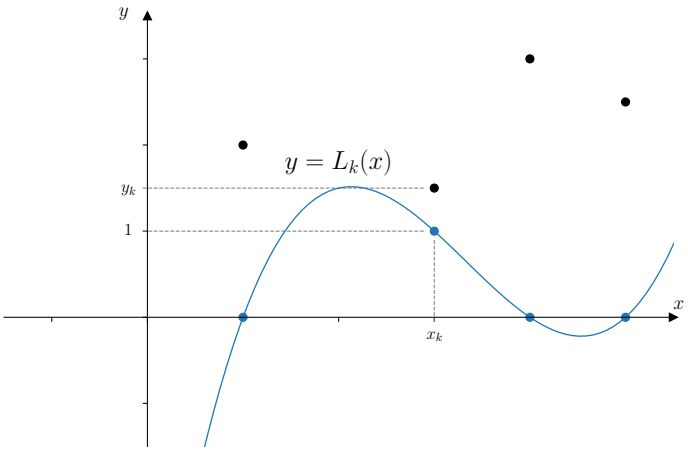


Figure 3.4: .

It's very simple to construct a polynomial that has zeros at chosen x values:

$$L_k(x) \propto (x - x_0)(x - x_1) \times \cdots \times (x - x_{k-1})(x - x_{k+1}) \times \cdots \times (x - x_n).$$

Notice we have excluded a term like $(x - x_k)$ to ensure that $L_k(x)$ *does not* equal 0 at $x = x_k$. Including the proportionality constant, we then have

$$L_k(x) = C_k \prod_{i \neq k} (x - x_i).$$

If you're not familiar with the "big pie" notation, it is the multiplicative version of the "big sigma" summation symbol Σ. Formally it is

$$\prod_{i=j}^{N} a_i = a_j \times a_{j+1} \times a_{j+2} \times \cdots \times a_{N-1} \times a_N.$$

It just means that we multiply the following expression by itself over the indices provided. Look at the previous expression for $L_k(x)$ to see what the $\prod$ symbol is replacing.

Now, we can determine the constant C_k by following our other desired property of $L_k(x)$,

that it passes through $(x_k, 1)$. This means

$$\begin{aligned} & L_k(x_k) = 1 \\ \implies & C_k \prod_{i \neq k} (x_k - x_i) = 1 \\ \implies & C_k = \frac{1}{\prod_{i \neq k}(x_k - x_i)} \end{aligned}$$

So now we can write the definition of $L_k(x)$ in terms of the coordinates we are trying to fit

$$L_k(x) = \frac{\prod_{i \neq k}(x - x_i)}{\prod_{i \neq k}(x_k - x_i)} = \prod_{i \neq k} \frac{x - x_i}{x_k - x_i}$$

Look at this function carefully. The denominator will be a multiplication of differences of known numbers, and so it will result in some number. The numerator is a function of x. We'll see in examples how this works in practice, but it really is just a number multipled by a polynomial, cleverly constructed to pass through certain points.

Now, this function passes through $(x_k, 1)$ and $(x_i, 0)$ for all $i \neq k$. If we multiply $L_k(x)$ by y_k, we will have a new polynomial that passes through (x_k, y_k) and doesn't change the other points that still must pass through zero. This, therefore, nicely gives us the properties we wanted from the beginning

$$y_k L_k(x) = \begin{cases} y_k & \text{if } x = x_k \\ 0 & \text{if } x = x_i \text{ for } i \neq k. \end{cases}$$

Now we only have to add up a series of these polynomials, giving us a new polynomial, that passes through every point we want. This is the *Lagrange polynomial*:

$$\boxed{p(x) = \sum_{k=0}^{n} y_k L_k(x)}$$

The components of the polynomial, $L_k(x)$, are called *Lagrange basis polynomials*. A sketch of a Lagrange polynomial and its basis polynomials is shown in figure 3.5.

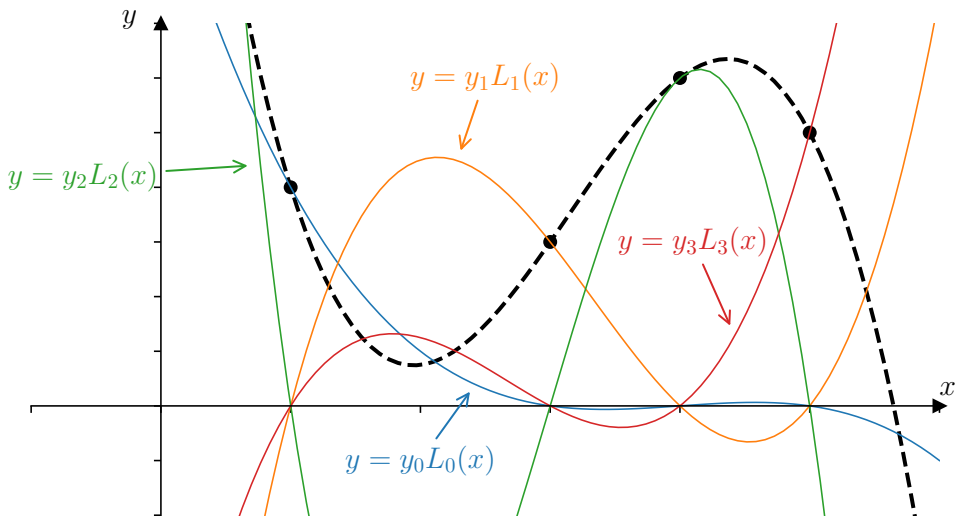


Figure 3.5: Terms of the Lagrange polynomial. Note that each coloured line passes through exactly 1 of the desired interpolation points, while passing through zero at the location of all the others. The dashed line is the addition of the other 4 lines, and thus passes through each interpolation point.

Example 11

Let's find the Lagrange polynomial that passes through the points in

$$\{(x_0, y_0), (x_1, y_1), (x_2, y_2), (x_3, y_3)\} = \{(1, 1), (2, 3), (3, 2), (4, 3)\}.$$

Since there are 4 points to fit, we form 4 basis polynomials

$$L_0(x) = \frac{(x-2)(x-3)(x-4)}{(1-2)(1-3)(1-4)} = -\frac{1}{6}(x-2)(x-3)(x-4)$$

$$L_1(x) = \frac{(x-1)(x-3)(x-4)}{(2-1)(2-3)(2-4)} = \frac{1}{2}(x-1)(x-3)(x-4)$$

$$L_2(x) = \frac{(x-1)(x-2)(x-4)}{(3-1)(3-2)(3-4)} = -\frac{1}{2}(x-1)(x-2)(x-4)$$

$$L_3(x) = \frac{(x-1)(x-2)(x-3)}{(4-1)(4-2)(4-3)} = \frac{1}{6}(x-1)(x-2)(x-3)$$

Stare at each polynomial until you recognise the patterns. In this first step, the denominator is a copy/paste of the numerator, but x is replaced with x_k where k is the index of the basis polynomial you are considering. Finally, we must add up these basis polynomials multipled by the y values of the fitting points

$$\begin{aligned} p(x) &= 1 \times L_0(x) + 3 \times L_1(x) + 2 \times L_2(x) + 4 \times L_3(x) \\ &= -\frac{1}{6}(x-2)(x-3)(x-4) + \frac{3}{2}(x-1)(x-3)(x-4) \\ &\quad - (x-1)(x-2)(x-4) + \frac{2}{3}(x-1)(x-2)(x-3) \end{aligned}$$

3.3 Newtonian interpolation

There is a weakness to the Lagrangian method for finding the interpolation polynomial. Consider a set of $n+1$ data points $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, so that the Lagrange polynomial is

$$p_n(x) = \sum_{k=0}^{n} y_k \prod_{i \neq k} \frac{x - x_i}{x_k - x_i}$$

what happens if we add data point (x_{n+1}, y_{n+1}), as pictured below?

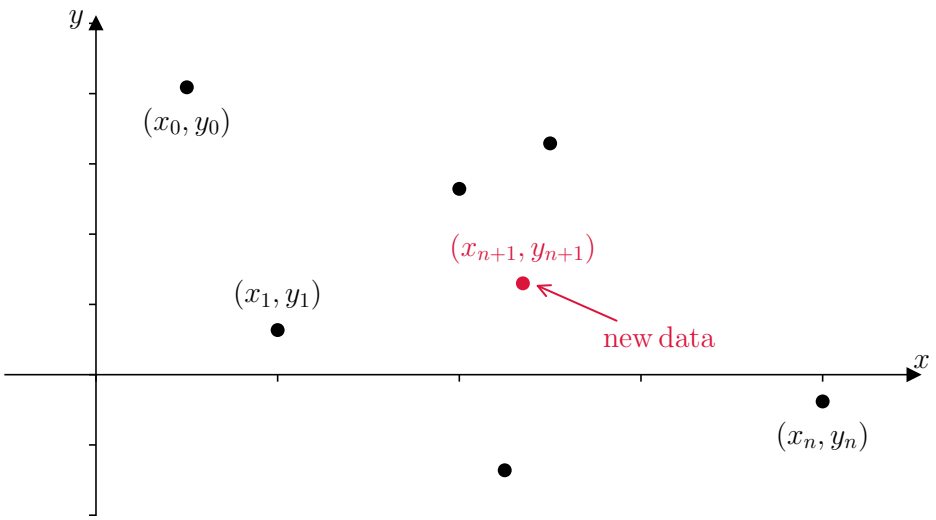


Figure 3.6

Well we would have to recompute all the Lagrange basis polynomials again to find p_{n+1}. This is obviously cumbersome, so we would like a method for *sequentially* building the unique interpolation polynomial out of the previous polynomial with 1 less point and therefore 1 lower degree. That is, we want a scheme

$$p_{n+1}(x) = p_n(x) + q_{n+1}(x), \quad q_{n+1}(x) \in \mathcal{P}_{n+1}. \tag{3.1}$$

Now, the new polynomial must pass through all of the old points, so

$$p_{n+1}(x_k) = y_k \quad \text{for } k = 0, 1, \ldots, n.$$

We also have that the old polynomial obviously passes through the old points

$$p_n(x_k) = y_k \quad \text{for } k = 0, 1, \ldots, n.$$

Hence equation 3.1 evaluated at one of the data points gives

$$\begin{aligned} p_{n+1}(x_k) &= p_n(x_k) + q_{n+1}(x_k) \\ y_k &= y_k + q_{n+1}(x_k). \end{aligned}$$

So we have the constraint

$$q_{n+1}(x_k) = 0 \quad \text{for } k = 0, 1, \ldots, n.$$

This means the general form of this extension polynomial is

$$q_{n+1}(x) = C_{n+1} \prod_{i=0}^{n} (x - x_i)$$

for some constant C_{n+1}. Now the new polynomial, p_{n+1} must also pass through the new datapoint (x_{n+1}, y_{n+1}), giving

$$p_{n+1}(x_{n+1}) = y_{n+1}$$

and hence

$$\begin{aligned} y_{n+1} &= p_n(x_{n+1}) + q_{n+1}(x_{n+1}) \\ \implies C_{n+1} \prod_{i=0}^{n} (x_{n+1} - x_i) &= y_{n+1} - p_n(x_{n+1}). \end{aligned}$$

This gives us an expression for this constant in terms of known values

$$\boxed{C_{n+1} = \frac{y_{n+1} - p_n(x_{n+1})}{\prod_{i=0}^{n}(x_{n+1} - x_i)}.}$$

And so the Newtonian interpolation polynomial is given recursively as:

$$\boxed{p_{n+1}(x) = p_n(x) + C_{n+1} \prod_{i=0}^{n} (x - x_i).} \tag{3.2}$$

Let's look at this expression sequentially. The zeroth term is simple, it's forced to be a 0 degree polynomial passing through the point (x_0, y_0). That is, it must be $p_0(x) = y_0$. This starts us off and let's us use equation 3.2

$$\begin{aligned} p_1(x) &= y_0 + C_1(x - x_0) \\ p_2(x) &= y_0 + C_1(x - x_0) + C_2(x - x_0)(x - x_1) \\ &\vdots \\ p_{n+1}(x) &= y_0 + C_1(x - x_0) + C_2(x - x_0)(x - x_1) + \cdots + C_{n+1}(x - x_0)(x - x_1) \times \cdots \times (x - x_n) \end{aligned}$$

Now, C_{n+1} as given above is horribly impractible. Let's demonstrate that by finding C_1 and C_2 in the general case before looking at a much better method!

$$\begin{aligned}
C_1 &= \frac{y_1 - p_0(x_1)}{x_1 - x_0} = \frac{y_1 - y_0}{x_1 - x_0} \\
C_2 &= \frac{y_2 - p_1(x_2)}{(x_2 - x_0)(x_2 - x_1)} \\
&= \frac{y_2 - (y_0 + C_1(x_2 - x_0))}{(x_2 - x_0)(x_2 - x_1)} \\
&= \frac{y_2 - y_0 - \frac{y_1 - y_0}{x_1 - x_0}(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)} \\
&= \frac{y_2 - y_0}{(x_2 - x_0)(x_2 - x_1)} - \frac{y_1 - y_0}{(x_1 - x_0)(x_2 - x_1)}
\end{aligned}$$

You see that the complexity builds up quickly. Try to write down C_3. But, there is a nice trick to quickly compute these coefficients, called *divided differences*.

First, notationally we will replace the C_{n+1} with

$$C_{n+1} = [y_0, \ldots, y_n, y_{n+1}].$$

This turns out to have a nice pattern involving recursive differences

$$\begin{aligned}
C_0 &= [y_0] = y_0 \\
C_1 &= [y_0, y_1] = \frac{y_1 - y_0}{x_1 - x_0} \\
C_2 &= [y_0, y_1, y_2] = \frac{[y_1, y_2] - [y_0, y_1]}{x_2 - x_0} \\
C_3 &= [y_0, y_1, y_2, y_3] = \frac{[y_1, y_2, y_3] - [y_0, y_1, y_2]}{x_3 - x_0} \\
&\vdots \\
C_{n+1} &= [y_0, \ldots, y_n, y_{n+1}] = \frac{[y_1, \ldots, y_{n+1}] - [y_0, \ldots, y_n]}{x_{n+1} - x_0}.
\end{aligned}$$

Before we make this clearer, let's check that it gives the right expression for C_2, that we

found earlier.

$$\begin{aligned}
C_2 &= \frac{[y_1,y_2]-[y_0,y_1]}{x_2-x_0} \\
&= \frac{\frac{y_2-y_1}{x_2-x_1}-\frac{y_1-y_0}{x_1-x_0}}{x_2-x_0} \\
&= \frac{y_2-y_1}{(x_2-x_1)(x_2-x_0)}-\frac{y_1-y_0}{(x_1-x_0)(x_2-x_0)} \\
&= \frac{y_2-y_0+y_0-y_1}{(x_2-x_1)(x_2-x_0)}-\frac{y_1-y_0}{(x_1-x_0)(x_2-x_0)} \\
&= \frac{y_2-y_0}{(x_2-x_1)(x_2-x_0)}-\frac{y_1-y_0}{(x_2-x_1)(x_2-x_0)}-\frac{y_1-y_0}{(x_1-x_0)(x_2-x_0)} \\
&= \frac{y_2-y_0}{(x_2-x_1)(x_2-x_0)}-\frac{y_1-y_0}{x_2-x_0}\left(\frac{1}{x_2-x_1}+\frac{1}{x_1-x_0}\right) \\
&= \frac{y_2-y_0}{(x_2-x_1)(x_2-x_0)}-\frac{y_1-y_0}{(x_2-x_1)(x_1-x_0)}.
\end{aligned}$$

It took some tricks but we got there, it's the same expression!

So, all we have done so far is replace one weird expression, the C_{n+1}, with another, $[y_0,\ldots,y_n]$. If we compute these new things the interpolation polynomial is given by

$$\begin{aligned}
p_{n+1}(x) = [y_0] + [y_0,y_1](x-x_0) + [y_0,y_1,y_2](x-x_0)(x-x_1) + \cdots \\
+ [y_0,\ldots,y_{n+1}](x-x_0)(x-x_1)\times\cdots\times(x-x_n).
\end{aligned}$$

Now we come to the heart of the method. These coefficients can be organised into a table that makes their computation actually clear:

Table of divided differences for Newtonian interpolation

x_0	$[y_0]$			
		$[y_0,y_1]$		
x_1	$[y_1]$		$[y_0,y_1,y_2]$	
		$[y_1,y_2]$		$[y_0,y_1,y_2,y_3]$
x_2	$[y_2]$		$[y_1,y_2,y_3]$	
		$[y_2,y_3]$		
x_3	$[y_3]$			
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

And of course it will become clearer what this table means with multiple usages.

Example 12
Given the data $\{(1,2),(2,2),(3,1),(4,3)\}$, determine the interpolation polynomial using the table of divided differences.

The table of divided differences is

x_k	y_k			
1	$\boxed{2}$			
		$\frac{2-2}{2-1} = \boxed{0}$		
2	2		$\frac{-1-0}{3-1} = \boxed{\text{-1/2}}$	
		$\frac{1-2}{3-2} = -1$		$\frac{\frac{3}{2}--\frac{1}{2}}{4-1} = \boxed{2/3}$
3	1		$\frac{2--1}{4-2} = \frac{3}{2}$	
		$\frac{3-1}{4-3} = 2$		
4	3			

In the boxes are the coefficients we need for the interpolation polynomial

$$\begin{aligned} p(x) &= 2 + 0(x-1) - \frac{1}{2}(x-1)(x-2) + \frac{2}{3}(x-1)(x-2)(x-3) \\ &= 2 - \frac{1}{2}x^2 + \frac{3}{2}x - 1 + \frac{2}{3}x^3 - 4x^2 + \frac{22}{3}x - 4 \\ &= -3 + \frac{53}{6}x - \frac{9}{2}x^2 + \frac{2}{3}x^3 \end{aligned}$$

Now let's do an example to see that *the order of the data doesn't matter*.

Example 13
Given the data $\{(1,2),(3,1),(4,3)\}$, construct the table of divided differences. Then add the datapoint $(2,2)$ to find the interpolation polynomial.

The table of divided differences for the 3 points is

x_k	y_k			
1	$\boxed{2}$			
		$\frac{1-2}{3-1} = \boxed{-1/2}$		
3	1		$\frac{2--\frac{1}{2}}{4-1} = \boxed{5/6}$	
		$\frac{3-1}{4-3} = 2$		
4	3			

In the boxes are the coefficients we need for the degree 2 polynomial fitting just these 3 points

$$\begin{aligned} p_2(x) &= 2 - \frac{1}{2}(x-1) + \frac{5}{6}(x-1)(x-3) \\ &= 2 - \frac{1}{2}x + \frac{1}{2} + \frac{5}{6}x^2 - \frac{10}{3}x + \frac{5}{2} \\ &= 5 - \frac{23}{6}x + \frac{5}{6}x^2 \end{aligned}$$

Now we add the new point $(2, 2)$ by simply attaching it to the previous table

x_k	y_k				
1	2				
		$\frac{1-2}{3-1} = -\frac{1}{2}$			
3	1		$\frac{2--\frac{1}{2}}{4-1} = \frac{5}{6}$		
		$\frac{3-1}{4-3} = 2$		$\frac{\frac{3}{2}-\frac{5}{6}}{2-1} = \boxed{2/3}$	
4	3		$\frac{\frac{1}{2}-2}{2-3} = \frac{3}{2}$		
		$\frac{2-3}{2-4} = \frac{1}{2}$			
2	2				

In the box is the coefficient we need to extend the degree 2 interpolation polynomial into the degree 3 polynomial that fits all 4 points

$$\begin{aligned} p_3(x) &= p_2(x) + \frac{2}{3}(x-1)(x-3)(x-4) \\ &= 5 - \frac{23}{6}x + \frac{5}{6}x^2 + \frac{2}{3}x^3 - \frac{16}{3}x^2 + \frac{38}{3}x - 8 \\ &= -3 + \frac{53}{6}x - \frac{9}{2}x^2 + \frac{2}{3}x^3 \end{aligned}$$

which is the same polynomial we found in the previous example! The order doesn't matter, and we can extend the polynomials one term at a time.

Chapter 4

Least-squares Extrapolation

Sometimes we don't want to fit data with a curve that is forced to pass through every point. We might instead assume there is a simple relation underneath the data and *noise* scatters the data around this relation. For example, see figure 4.1.

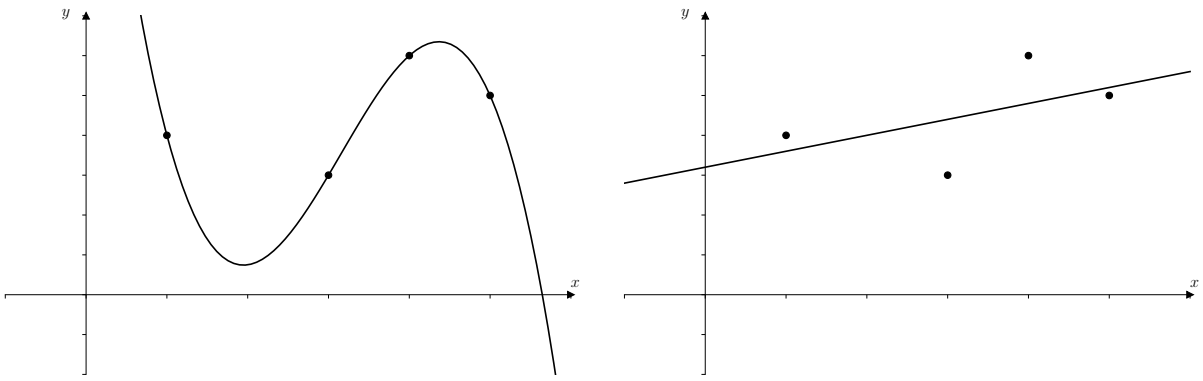


Figure 4.1: (Left) Lagrange polynomial passing through each point (right) simple line that roughly captures the trend of the data.

If we see a rough trend in the data, we may want to predict what will happen outside of the data range. The Lagrange polynomial is bad at this, giving massive errors outside of the range of the given data. So, Lagrange is good for *interpolation*, and in this chapter we will look at the method of least squares which is good for *extrapolation*.

4.1 Least-squares linear fitting

Say you have $n+1$ data points (x_k, y_k) for $k = 0, 1, 2, \ldots, n$. Imagine we want to *fit* this data with a straight line. We want to find the best line $y = mx + b$, where best means to

minimise the square of the *residuals*:

$$r_k = y_k - y(x_k).$$

Looking at figure 4.2 you can see that the residual is the vertical distance between a data point and a straight line. So minimising the residuals is a way to choose a line that is closest to all the data points in some way. We minimise the square so that it doesn't matter whether the residual is positive or negative.

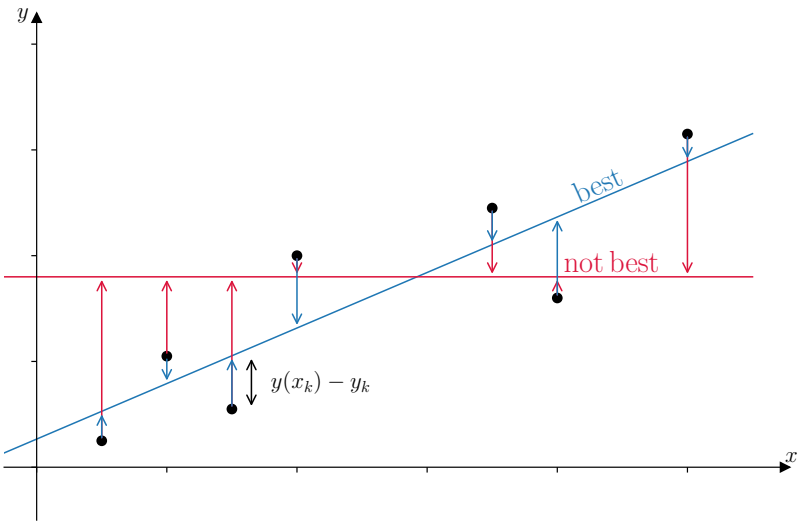


Figure 4.2: Two linear fits to the data points. The blue straight line minimises the collection of distances, $y(x_k) - y_k$, of each data point from the line. The red line is obviously worse.

What are we changing when we say we are minimising the (square of the) residuals? We are adjusting a straight line, which means we are changing the slope ,m, and the vertical offset, b. So. let's define the sum of the square of the residuals as a two variable function in the variables m and b

$$S(m, b) = \sum_{k=0}^{n} r_k^2 = \sum_{k=0}^{n} (y_k - y(x_k))^2 .$$

This function is general for any fitting curve $y(x)$, but in this chapter we only use linear fits, so we can define the function S as

$$S(m, b) = \sum_{k=0}^{n} (y_k - mx_k - b)^2 .$$

Recall that to find where a single-variable function, $f(x)$, is minimum (or maximum) you must find where its derivative is equal to zero. For multi-variate functions, we must find where its partial derivatives are zero. So we impose

$$\frac{\partial S}{\partial m} = 0$$
$$\frac{\partial S}{\partial b} = 0.$$

These partial derivatives are

$$\frac{\partial S}{\partial m} = \sum_{k=0}^{n} 2r_k x_k = \sum_{k=0}^{n} 2\left(y_k - mx_k - b\right) x_k$$
$$\frac{\partial S}{\partial b} = \sum_{k=0}^{n} 2r_k = \sum_{k=0}^{n} 2\left(y_k - mx_k - b\right)$$

and so setting these to be zero, we must simultaneously solve

$$\sum_{k=0}^{n} \left(y_k - mx_k - b\right) x_k = 0 \tag{4.1}$$
$$\sum_{k=0}^{n} \left(y_k - mx_k - b\right) = 0 \tag{4.2}$$

Example 14

Make a least-squares linear fit to the data: $\{(1,1), (2,3), (3,2), (5,3)\}$.

We want m and b for $y(x) = mx + b$. The residuals are

$$r_0 = 1 - m - b$$
$$r_1 = 3 - 2m - b$$
$$r_2 = 2 - 3m - b$$
$$r_3 = 3 - 5m - b$$

The function to minimise is

$$\begin{aligned} S(m,b) &= r_0^2 + r_1^2 + r_2^2 + r_3^2 \\ &= 1 - m - b - m + m^2 + mb - b + mb + b^2 + \\ &\quad 9 - 6m - 3b - 6m + 4m^2 + 2mb - 3b + 2mb + b^2 + \\ &\quad 4 - 6m - 2b - 6m + 9m^2 + 3mb - 2b + 3mb + b^2 + \\ &\quad 9 - 15m - 3b - 15m + 25m^2 + 5mb - 3b + 5mb + b^2. \end{aligned}$$

Finally, collecting all the terms we have

$$\boxed{S(m,b) = 23 - 56m - 18b + 22mb + 39m^2 + 4b^2}$$

The partial derivatives give

$$\begin{aligned} \frac{\partial S}{\partial m} &= -56 + 22b + 78m = 0 \implies b = \frac{56 - 78m}{22} \\ \frac{\partial S}{\partial b} &= -18 + 22m + 8b = 0 \implies b = \frac{18 - 22m}{8} \end{aligned}$$

This eliminates b, allowing us to determine m

$$\begin{aligned} \frac{56 - 78m}{22} &= \frac{18 - 22m}{8} \\ 8(56 - 78m) &= 22(18 - 22m) \\ 484m - 624m &= 396 - 448 \\ m &= \frac{52}{140} \sim 0.37 \end{aligned}$$

and now plug this into either of the 2 equations for b

$$b \sim \frac{56 - 78 \times 0.37}{22} \sim 1.23.$$

So we have the least squares linear fit to the data

$$y = 0.37x + 1.23.$$

In this previous example we expanded the squares of residuals *then* differentianted. It

turns out to be easier to differentiate *then* expand. Recall the partial derivative of S with respect to b gave us the condition

$$\sum_{k=0}^{n} (y_k - mx_k - b) = 0.$$

This can be rearranged to

$$\sum_{k=0}^{n} (y_k - mx_k) - \sum_{k=0}^{n} b = 0$$
$$\sum_{k=0}^{n} (y_k - mx_k) - (n+1)b = 0$$

so that we have an expression for b

$$\boxed{b = \frac{1}{n+1} \sum_{k=0}^{n} (y_k - mx_k)}$$

For the previous example with data $\{(1,1), (2,3), (3,2), (5,3)\}$, we have $n = 3$, and this gives

$$\begin{aligned} b &= \frac{1}{4} \left((1-m) + (3-2m) + (2-3m) + (3-5m) \right) \\ &= \frac{9-11m}{4} = \frac{18-22m}{8} \end{aligned}$$

which is the same equation we had, but now we reached it faster.

The partial derivative with respect to m gave us the condition

$$\sum_{k=0}^{n} x_k \left(y_k - mx_k - b \right) = 0.$$

For this previous example this gives

$$\begin{aligned} &1(1-m-b) + 2(3-2m-b) + 3(2-3m-b) + 5(3-5m-b) = 0 \\ &28 - 39m - 11b = 0. \end{aligned}$$

This is the second equation that we got earlier, but now we reached it faster. Taking the derivatives before expanding is simpler because it removes the need to expand squared expressions.

But we can push this algebra further to create even simpler expressions for b and m. So we start from the b expression we just derived

$$b = \frac{1}{n+1}\sum_{k=0}^{n} y_k - m\frac{1}{n+1}\sum_{k=0}^{n} x_k.$$

Now we note that the sum of a list of numbers, x_k, divided by how many numbers in that list is just the average value of the list, which we will denote $\bar{x}_k$. So we can write

$$b = \bar{y}_k - m\bar{x}_k. \tag{4.3}$$

We insert this into equation 4.1 and clean things up

$$\begin{aligned}
&\sum_{k=0}^{n} x_k \left(y_k - mx_k - b\right) = 0 \\
&\sum_{k=0}^{n} x_k \left(y_k - mx_k - \bar{y}_k + m\bar{x}_k\right) = 0 \\
&\sum_{k=0}^{n} \left(x_k \left(y_k - \bar{y}_k\right) + mx_k \left(\bar{x}_k - x_k\right)\right) = 0 \\
&\sum_{k=0}^{n} x_k \left(y_k - \bar{y}_k\right) + m\sum_{k=0}^{n} x_k \left(\bar{x}_k - x_k\right) = 0.
\end{aligned}$$

And this gives us an expression for m that is independent of b

$$\boxed{m = \frac{\sum_{k=0}^{n} x_k \left(\bar{y}_k - y_k\right)}{\sum_{k=0}^{n} x_k \left(\bar{x}_k - x_k\right)}}$$

which can be plugged into equation 4.3 to give an expression for b that is independent of m

$$\boxed{b = \bar{y}_k - \bar{x}_k \frac{\sum_{k=0}^{n} x_k \left(\bar{y}_k - y_k\right)}{\sum_{k=0}^{n} x_k \left(\bar{x}_k - x_k\right)}}$$

These last two expressions are not necessarily easier to use in hand calculations, but they are very easy to code. A computer can take averages very easily, and so those two expressions might be more useful for creating a least-squares computer program.

Example 15

Make a least-squares linear fit to the data: $\{(1,1), (2,3), (3,2), (5,3)\}$. Same as the previous

example, but with the new method.
We need the averages of the x_k and y_k:

$$\bar{x}_k = \frac{1+2+3+5}{4} = \frac{11}{4}$$
$$\bar{y}_k = \frac{1+3+2+3}{4} = \frac{9}{4}$$

So we use the formula to calculate the slope

$$m = \frac{1(\frac{9}{4}-1)+2(\frac{9}{4}-3)+3(\frac{9}{4}-2)+4(\frac{9}{4}-3)}{1(\frac{11}{4}-1)+2(\frac{11}{4}-2)+3(\frac{11}{4}-3)+5(\frac{11}{4}-5)} = \frac{-13/4}{-35/4} \sim 0.37$$

Now we use this in the formula for the offset

$$b = \bar{y}_k - m\bar{x}_k \sim \frac{9}{4} - 0.37 \times \frac{11}{4} \sim 1.23$$

So we have the same least squares linear fit to the data

$$y = 0.37x + 1.23.$$

You decide whether this method is simpler or not.

4.2 Least-squares non-linear fitting

We finish this chapter by showing that the linear fitting is more useful than you may realise. It can also be used to make non-linear fits (for example quadratic or exponential) by cleverly scaling the data first.

Chapter 5

Integration and Differentiation

Many functions are simple to integrate or differentiate by hand. You know the basic rules for trigonmetric functions, logs, exponentials, polynomials. But in some complicated (sometimes real-world) applications you may need to integrate or differentiate a function that has no closed-form expression. In this chapter we develop numerical methods to deal with these kinds of problems.

5.1 Numerical integration

The goal is to integrate some known function, $f(x)$, over an interval $[a, b]$, giving just 1 real value

$$I = \int_a^b f(x)dx.$$

That is, the find the area under its curve, as graphed in figure 5.1.

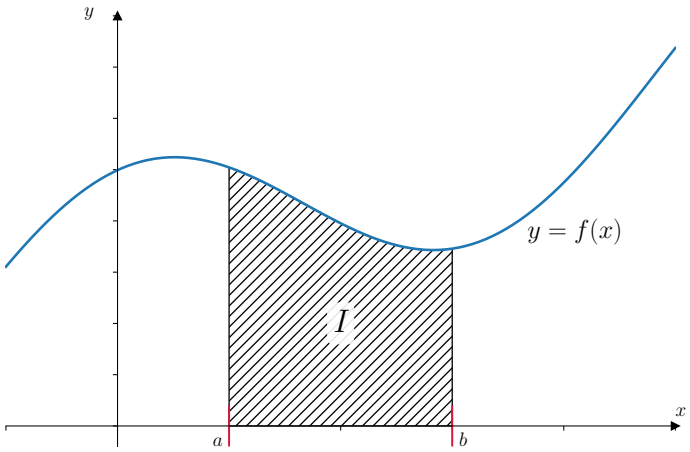


Figure 5.1: .

The method we will use is to first estimate $f(x)$ with the Lagrangian interpolation polynomial, $p_n(x)$, and integrate that instead. So we have the approximation for the integral

$$I \sim \int_a^b p_n(x)dx.$$

Figure 5.2 shows the general schema, where we interpolate $f(x)$ using $n+1$ evenly spaced points on $[a, b]$ (including the end points).

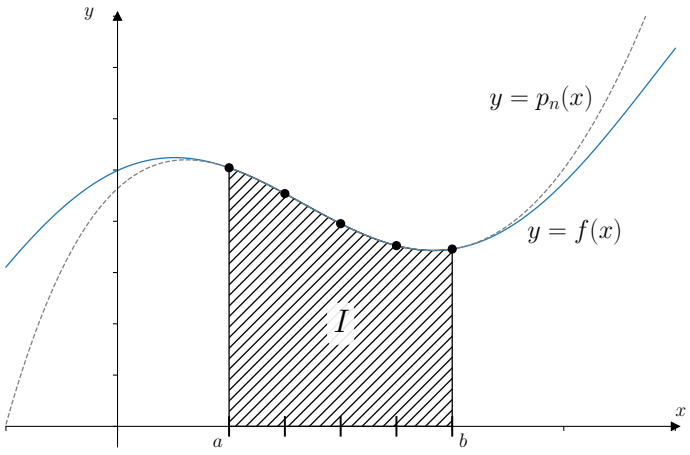


Figure 5.2: .

Recall from Chapter 3 that the interpolation polynomial through $n+1$ points is given by

a sum over the y values of the interpolation points multiplied by Lagrange basis polynomials

$$p(x) = \sum_{k=0}^{n} y_k L_k(x) = \sum_{k=0}^{n} y_k \prod_{i \neq k} \frac{x - x_i}{x_k - x_i}.$$

Since the interpolation points are given by a known function, we can set $y_k = f(x_k)$. The even spacing on $[a, b]$ also gives us the expressions for the x values of the interpolation points

$$x_k = a + k\frac{b-a}{n}, \quad k = 0, 1, 2, \ldots, n.$$

So the integral is approximated by

$$\begin{aligned} I &\sim \int_a^b \sum_{k=0}^{n} f(x_k) L_k(x) dx \\ &= \sum_{k=0}^{n} f(x_k) \int_a^b L_k(x) dx \\ &= \sum_{k=0}^{n} f(x_k) w_k \end{aligned}$$

where we have introduced the "quadrature weights"

$$w_k = \int_a^b L_k(x) dx.$$

This formalism of approximating an integral with evenly spaced interpolation points gives a family of formulae for different number of interpolation points, $n + 1$, called *Newton-Cotes formulae*. In this chapter we'll only look at $n = 1$ and $n = 2$, which give rise to the Trapezium and Simpson's rules, respectively.

5.1.1 Trapezium rule

If we choose $n = 1$, then we interpolate the function with $n + 1 = 2$ points. These are merely the endpoints of the interval, $x_0 = a$ and $x_1 = b$. Interpolating with a polynomial of degree $n = 1$ means a straight line connecting these two points. This integration approximation is graphed in figure 5.3.

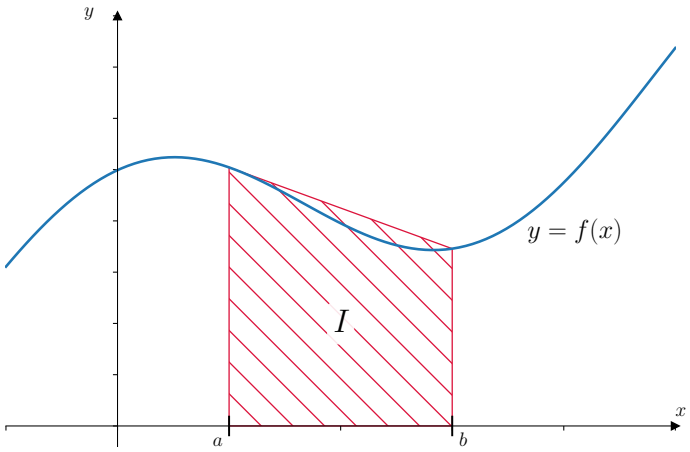


Figure 5.3: .

The Newton-Cotes formula for $n = 1$ is then

$$I \sim f(a)w_0 + f(b)w_1.$$

Let's derive expressions for the quadrature weights. The zeroth weight is

$$\begin{aligned} w_0 &= \int_a^b L_0(x)dx, \quad \text{and } L_0 = \frac{x-b}{a-b} \\ &= \frac{1}{a-b}\left(\frac{x^2}{2} - bx\right)\Big|_a^b \\ &= \frac{1}{a-b}\left(-\frac{b^2}{2} - \frac{a^2}{2} + ba\right) \\ &= -\frac{1}{2(a-b)}\left(a^2 - 2ab + b^2\right) \\ &= -\frac{1}{2(a-b)}(a-b)^2 \\ &= \frac{b-a}{2} \end{aligned}$$

and similarly the first weight is

$$\begin{aligned} w_1 &= \int_a^b L_1(x)dx, \quad \text{and } L_1 = \frac{x-a}{b-a} \\ &= \frac{1}{b-a}\left(\frac{x^2}{2} - ax\right)\Big|_a^b \\ exercice &= \frac{1}{2(b-a)}(b-a)^2 \\ &= \frac{b-a}{2}. \end{aligned}$$

The two weights are the same! Hence we have

Trapezium rule for approximating an integral

$$\int_a^b f(x)dx \sim \frac{b-a}{2}(f(a)+f(b)) \tag{5.1}$$

With a small rearrangement we can get an alternative geometric interpretation of this rule

$$I \sim \underbrace{\frac{f(a)+f(b)}{2}}_{\text{average height in the range}} \overbrace{(b-a)}^{\text{size of range}}.$$

This is simply the formula for the area of the rectangle shown below, with height equal to the average height of the function in the integration interval $[a, b]$.

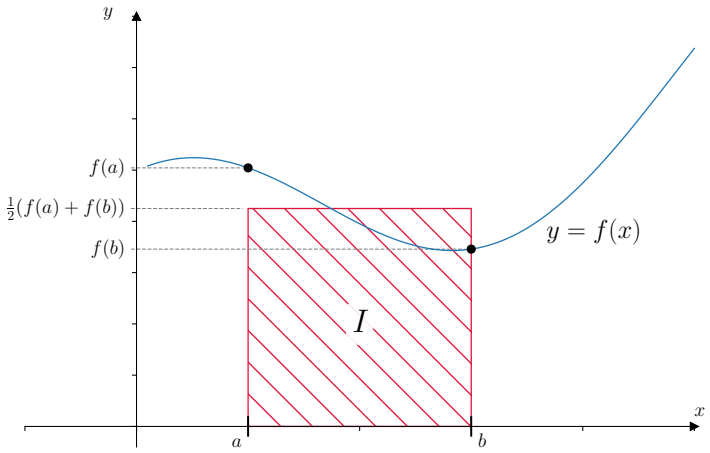


Figure 5.4: .

Example 16

Integrate $f(x) = e^x$ from 0 to 4 using the Trapezium rule with two (equal-sized) subdivisions.

The two subdivisions are the intervals $[0, 2]$ and $[2, 4]$. The areas are sketched out below

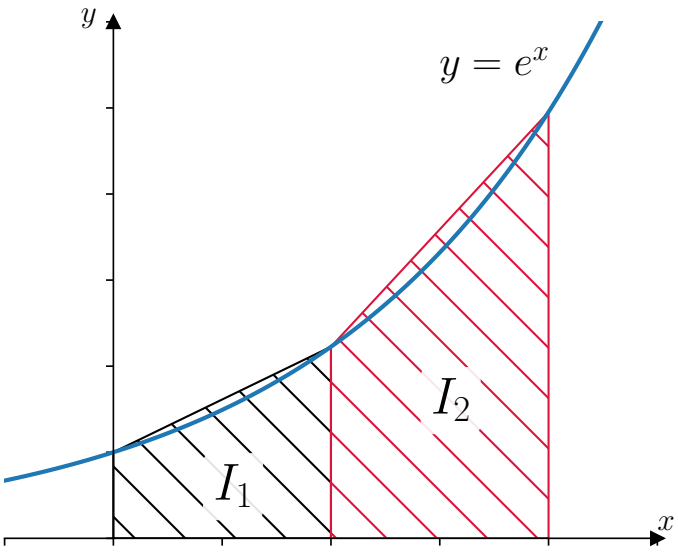


Figure 5.5: .

The two integrals are then

$$\int_0^2 e^x dx \sim I_1 = \frac{2-0}{2}\left(f(0) + f(2)\right) = e^0 + e^2 = 1 + e^2$$
$$\int_2^4 e^x dx \sim I_2 = \frac{4-2}{2}\left(f(2) + f(4)\right) = e^2 + e^4.$$

Therefore the approximation of the total integral is just the addition of these two parts

$$\int_0^4 e^x dx \sim 1 + 2e^2 + e^4 \sim 70.4.$$

5.1.2 Simpson's rule

Now we choose $n = 2$, so we interpolate the function with $n + 1 = 3$ points. These 3 points are the endpoints of the interval and the midpoint, $x_0 = a$, $x_1 = (a + b)/2$, and $x_2 = b$. Interpolating with a polynomial of degree $n = 2$ means a quadratic through the points. This integration approximation is graphed in figure 5.6.

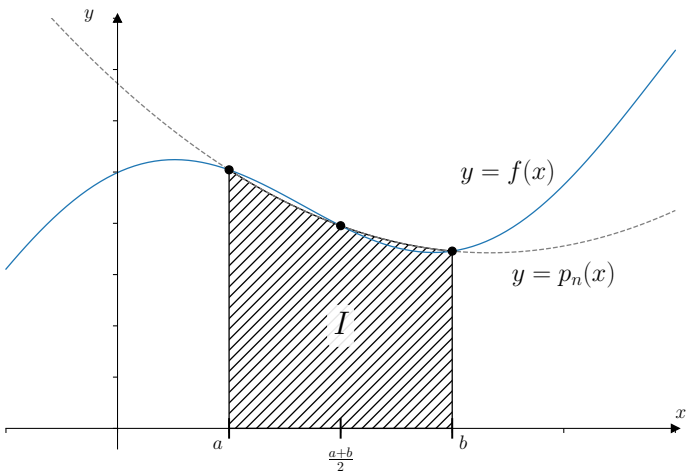


Figure 5.6: .

The Newton-Cotes formula for $n = 2$ is then

$$I \sim f(a)w_0 + f(\frac{a+b}{2})w_1 + f(b)w_2.$$

Let's derive expressions for the quadrature weights. Sometimes it will be convenient to use the midpoint $m = (a + b)/2$. Then the zeroth weight is

$$w_0 = \int_a^b L_0(x)dx, \quad \text{and } L_0 = \frac{(x-m)(x-b)}{(a-m)(a-b)}.$$

Let's integrate just the numerator for now

$$\begin{aligned}\int_a^b (x-m)(x-b)dx &= \left(\frac{x^3}{3} - \frac{m+b}{2}x^2 + mbx\right)\Big|_a^b \\ &= \frac{b^3}{3} - \frac{m+b}{2}b^2 + mb^2 - \frac{a^3}{3} + \frac{m+b}{2}a^2 - mba \\ &= \frac{b^3}{3} - \frac{ab^2}{4} - \frac{b^3}{4} - \frac{b^3}{2} + \frac{ab^2}{2} + \frac{b^3}{2} - \frac{a^3}{3} + \frac{a^3}{4} + \frac{ba^2}{4} + \frac{ba^2}{2} - \frac{ba^2+ab^2}{2} \\ &= \frac{1}{12}\left(b^3 - 3ab^2 + 3ba^2 - a^3\right) \\ &= \frac{1}{12}(b-a)^3 .\end{aligned}$$

The weight is then

$$\begin{aligned}w_0 &= \frac{1}{12}\frac{(b-a)^3}{(a-m)(a-b)} \\ &= \frac{1}{12}\frac{(b-a)^2}{m-a}\end{aligned}$$

This denominator is

$$m - a = \frac{a+b}{2} - a = \frac{b-a}{2}$$

and so the final expression for the weight is

$$w_0 = \frac{b-a}{6}.$$

It's not so difficult to show that the other two weights are

$$\begin{aligned}w_1 &= \frac{4}{6}(b-a), \\ w_2 &= \frac{b-a}{6}.\end{aligned}$$

And so we have

Simpson's rule for approximating an integral

$$\int_a^b f(x)dx \sim \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right). \tag{5.2}$$

Sometimes we use the distance between the interpolation points, $h = (b-a)/2$, and then

Simpson's rule is written

$$\int_a^b f(x)dx \sim \frac{h}{3}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right).$$

from which we get the alternative name *Simpson's 1/3 rule*.

Example 17

Integrate $f(x) = e^x$ from 0 to 4 using Simpson's rule with two (equal-sized) subdivisions. The two subdivisions are the intervals $[0, 2]$ and $[2, 4]$. The areas are sketched out below

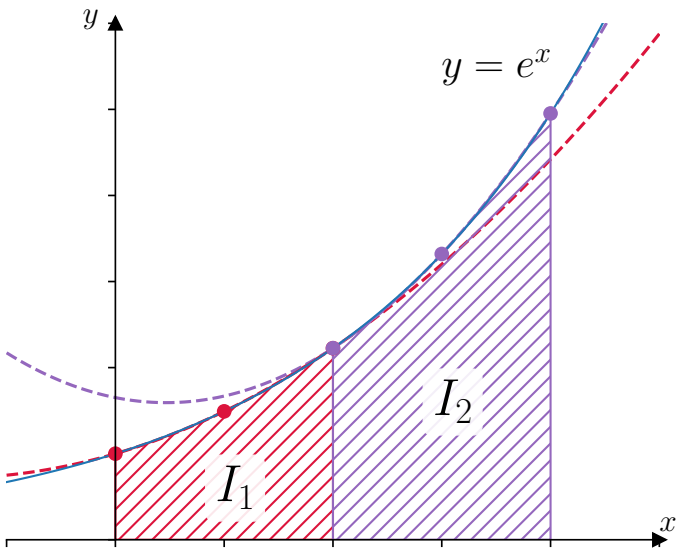


Figure 5.7: .

The two integrals are then

$$\int_0^2 e^x dx \sim I_1 = \frac{2-0}{6}\left(f(0) + 4f(1) + f(2)\right) = \frac{1}{3}\left(1 + 4e + e^2\right)$$
$$\int_2^4 e^x dx \sim I_2 = \frac{4-2}{6}\left(f(2) + 4f(3) + f(4)\right) = \frac{1}{3}\left(e^2 + 4e^3 + e^4\right).$$

Therefore the approximation of the total integral is just the addition of these two parts

$$\int_0^4 e^x dx \sim \frac{1}{3}\left(1 + 4e + 2e^2 + 4e^3 + e^4\right) \sim 53.9.$$

This integral can be solved analytically for comparison

$$\int_0^4 e^x dx = e^x|_0^4 = e^4 - 1 \sim 53.6.$$

So the Simpson's rule approximation is much better than the Trapezium rule result.

One final consideration: it's not necessarily the case that increasing the number of interpolation points increases the accuracy of the approximation. Consider the integral

$$\int_{-5}^{5} \frac{1}{1+x^2} dx.$$

Applying Newton-Cotes formulae with increasing n does not converge, and in fact eventually increases without bound. It is generally better to stick with Simpson's rule, but to increase the number of subdivisions, as shown below:

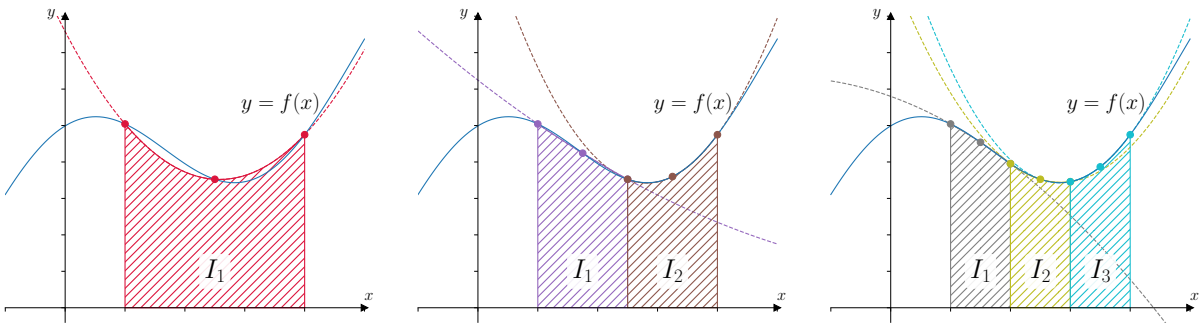


Figure 5.8: Simpson's rule using 1, 2 or 3 subdivisions.

5.1.3 Degree of precision

For an integration approximation scheme, the degree of precision of is the largest k such that the integration scheme applied to x^k gives the exact answer for $\int x^k\,dx$ which is of course $x^{k+1}/(k+1)$. That is, a scheme with degree of precision k will be exact when applied to

$$1 \quad x \quad x^2 \quad \cdots \quad x^k$$

but will fail to give the exact value when applied to x^{k+1}.

Example 18
Find the degree of precision for the Trapezium rule and for Simpson's rule.

For the Trapezium rule, when $f(x) = x^0$ we have

$$\begin{aligned}\int_a^b 1dx \sim \frac{b-a}{2}\left(f(a)+f(b)\right) &= \frac{b-a}{2}\left(1+1\right)\\ &= b-a\\ &= x\big|_a^b\end{aligned}$$

This is exact so we check the next power. For $f(x) = x^1$ we have

$$\begin{aligned}\int_a^b xdx &\sim \frac{b-a}{2}\left(a+b\right)\\ &= \frac{b^2-a^2}{2}\\ &= \frac{x^2}{2}\Big|_a^b\end{aligned}$$

This is exact so we check the next power. For $f(x) = x^2$ we have

$$\begin{aligned}\int_a^b x^2dx &\sim \frac{b-a}{2}\left(a^2+b^2\right)\\ &= \frac{b^3+ba^2-ab^2-a^3}{2}\\ &\neq \frac{x^3}{3}\Big|_a^b\end{aligned}$$

As $k = 1$ is the highest power for which the Trapezium rule gave the exact result, the degree of precision is 1.

For Simpson's rule, when $f(x) = x^0$ we have

$$\begin{aligned}\int_a^b 1dx \sim \frac{b-a}{6}\left(f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right) &= \frac{b-a}{6}\left(1+4+1\right).\\ &= b-a\\ &= x\big|_a^b\end{aligned}$$

This is exact so we check the next power. For $f(x) = x^1$ we have

$$\begin{aligned}\int_a^b x dx &\sim \frac{b-a}{6}\left(a + 4\left(\frac{a+b}{2}\right) + b\right)\\ &= \frac{b-a}{6}(3a+3b)\\ &= \frac{b^2-a^2}{2}\\ &= \left.\frac{x^2}{2}\right|_a^b\end{aligned}$$

This is exact so we check the next power. For $f(x) = x^2$ we have

$$\begin{aligned}\int_a^b x^2 dx &\sim \frac{b-a}{6}\left(a^2 + 4\left(\frac{a+b}{2}\right)^2 + b^2\right)\\ &= \frac{b-a}{6}\left(a^2 + 4\left(\frac{a^2+2ab+b^2}{4}\right) + b^2\right)\\ &= \frac{b-a}{3}\left(a^2+ab+b^2\right)\\ &= \frac{b^3-a^3}{3}\\ &= \left.\frac{x^3}{3}\right|_a^b\end{aligned}$$

This is exact so we check the next power. For $f(x) = x^3$ we have

$$\begin{aligned}\int_a^b x^3 dx &\sim \frac{b-a}{6}\left(a^3 + 4\left(\frac{a+b}{2}\right)^3 + b^3\right)\\ &= \frac{b-a}{6}\left(a^3 + 4\left(\frac{a^3+3a^2b+3ab^2+b^3}{8}\right) + b^3\right)\\ &= \frac{b-a}{6}\left(\frac{3}{2}a^3 + \frac{3}{2}a^2b + \frac{3}{2}ab^2 + \frac{3}{2}b^3\right)\\ &= \frac{b^4-a^4}{4}\\ &= \left.\frac{x^4}{4}\right|_a^b\end{aligned}$$

This is exact so we check the next power. For $f(x) = x^4$ we have

$$\begin{aligned}
\int_a^b x^4 dx &\sim \frac{b-a}{6}\left(a^4 + 4\left(\frac{a+b}{2}\right)^4 + b^4\right) \\
&= \frac{b-a}{6}\left(a^4 + 4\left(\frac{a^4+4a^3b+6a^2b^2+4ab^3+b^4}{16}\right) + b^4\right) \\
&= \frac{b-a}{24}\left(5a^4 + 4a^3b + 6a^2b^2 + 4ab^3 + 5b^4\right) \\
&= \frac{5b^5 + a^4b - 2a^3b^2 + 2a^2b^3 - ab^4 - 5a^5}{24} \\
&\neq \left.\frac{x^5}{5}\right|_a^b
\end{aligned}$$

As $k = 3$ is the highest power for which Simpson's rule gave the exact result, the degree of precision is 3.

5.2 Numerical differentiation

The goal is to find the slope of some known function, $f(x)$, at a point $x = x_k$, giving just 1 real value

$$m = f'(x_k)$$

as shown below.

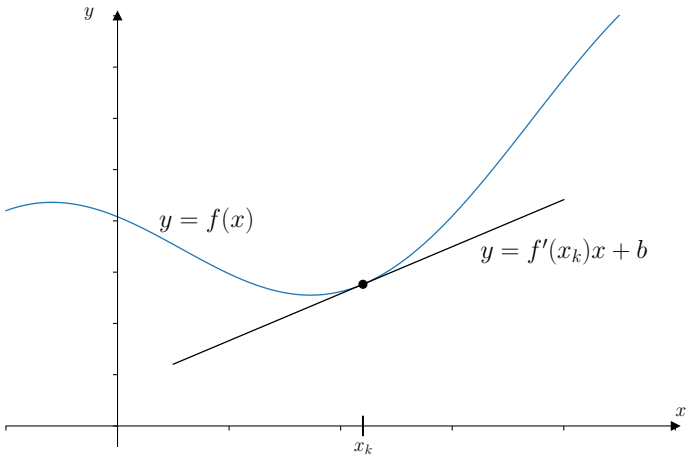


Figure 5.9: .

The method will be just like with integration, we first estimate the function with a Lagrange polynomial, $p_n(x)$ and differentiate that instead (and then evaluate the polynomial at x_k)! That is,

$$f'(x_k) \sim \frac{dp_n}{dx}(x_k).$$

We will here look at only $n = 1$, which gives the forward and backward finite difference schemes, and $n = 2$, which gives the centred finite difference scheme.

5.2.1 Forward and backward finite difference

For $n = 1$, we need $n + 1 = 2$ points of interpolation. Of course $x = x_k$ will be one of the points, but we have a free choice for the second point. If we choose the second point to be greater than x_k we get the forward finite difference scheme, whereas if we choose it to be less than x_k we get the backward finite difference scheme. The two schemes are illustrated below.

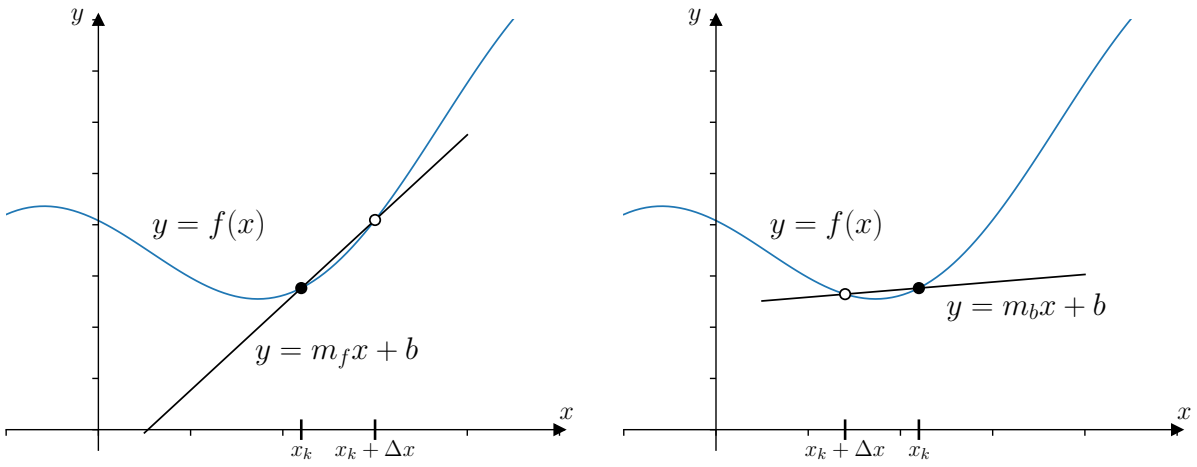


Figure 5.10: (Left) forward finite difference scheme and (right) backward finite difference scheme.

Consider the forward scheme, so that the interpolation positions are $x_0 = x_k$ and $x_1 = x_k + \Delta x$ for some small distance Δx. Then the Lagrange polynomial is given by

$$f(x) \sim p_1(x) = f(x_0)L_0(x) + f(x_1)L_1(x).$$

The Lagrange basis polynomials are

$$\begin{aligned} L_0(x) &= \frac{x - x_1}{x_0 - x_1} = \frac{x - x_k - \Delta x}{-\Delta x} \\ L_1(x) &= \frac{x - x_0}{x_1 - x_0} = \frac{x - x_k}{\Delta x}. \end{aligned}$$

So the derivative of f is approximated by

$$\begin{aligned} \frac{dp_1}{dx} &= f(x_0)\frac{dL_0}{dx} + f(x_1)\frac{dL_1}{dx} \\ &= f(x_0)\frac{1}{-\Delta x} + f(x_1)\frac{1}{\Delta x} \\ &= \frac{f(x_1) - f(x_0)}{\Delta x} \\ &= \frac{f(x_k + \Delta x) - f(x_k)}{\Delta x}. \end{aligned}$$

I hope a sense of familiarity is tickling your brain right now, because this is exactly the form

of the first principles definition of the derivative of a function at a point x_k

$$\frac{df}{dx}(x_k) = \lim_{\Delta x \to 0} \frac{f(x_k + \Delta x) - f(x_k)}{\Delta x}.$$

So this scheme converges on the true derivative as Δx becomes very small. So we have

Forward finite difference approximation of a derivative

$$f'(x_k) \sim \frac{f(x_k + \Delta x) - f(x_k)}{\Delta x}. \tag{5.3}$$

The formula for backward finite difference is quite straightforward (you should try to guess it before looking down)

Backward finite difference approximation of a derivative

$$f'(x_k) \sim \frac{f(x_k) - f(x_k - \Delta x)}{\Delta x}. \tag{5.4}$$

The accuracy of these approximations can be assessed using the Taylor expansion of f

$$f(x) = \sum_{n=0}^{\infty} f^{(n)}(a) \frac{(x-a)^n}{n!}.$$

We replace x with $x + \Delta x$ and use $a = x$

$$\begin{aligned} f(x + \Delta x) &= \sum_{n=0}^{\infty} f^{(n)}(x) \frac{(\Delta x)^n}{n!} \\ &= f(x) + f'(x)\Delta x + f''(x)\frac{(\Delta x)^2}{2} + \cdots \end{aligned}$$

Rearranging for the first derivative

$$f'(x) = \underbrace{\frac{f(x + \Delta x) - f(x)}{\Delta x}}_{\text{forward finite difference scheme}} - \underbrace{f''(x)\frac{\Delta x}{2}}_{\text{largest error term}} - \cdots$$

So we see that the largest error term is controlled by the second derivative (because further derivatives get multiplied by increasing powers of Δx, which becomes very small). This makes sense. Consider the graphs of the two functions below.

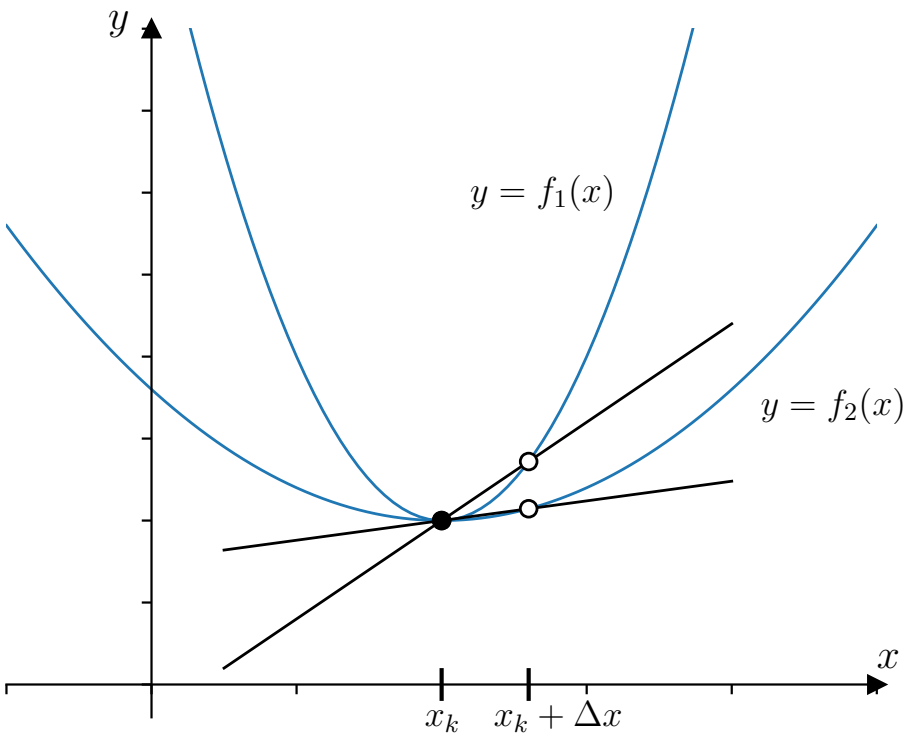


Figure 5.11: Two functions $f_1(x)$ and $f_2(x)$ with same first derivatives $f_1'(x_k) = f_2'(x_k)$ at $x = x_k$, but different second derivatives $f_1''(x_k) > f_2''(x_k)$.

Both of these functions have the same slope at $x = x_k$. But the function $f_1(x)$ has a larger second derivative than $f_2(x)$ at that position. You can see that the forward finite difference approximation to the derivative at this position, despite using the same Δx, is worse for the function with larger second derivative at the position.

Example 19

Given the data

x	4.5000	4.5025	4.5050	4.5075	4.5100
$f(x)$	405.563	406.472	407.383	408.296	409.210

Estimate estimate $f'(4.5050)$ by using forward finite difference with $\Delta x = 0.0025$ and backward finite difference with $\Delta x = 0.005$.

We are estimating the derivative at the position $x_k = 4.5050$. For forward finite difference,

$x_k + \Delta x = 4.5075$. So we have

$$\begin{aligned} f'(4.5050) &= \frac{f(4.5075) - f(4.5050)}{0.0025} \\ &= \frac{408.296 - 407.383}{0.0025} \\ &\sim 365.200 \end{aligned}$$

For backward finite difference with $\Delta x = 0.005$ we have $x_k - \Delta x = 4.5000$

$$\begin{aligned} f'(4.5050) &= \frac{f(4.5050) - f(4.5000)}{0.005} \\ &= \frac{407.383 - 405.563}{0.005} \\ &\sim 364.000 \end{aligned}$$

5.2.2 Centred finite difference

Now we consider $n = 2$, so that we have $n + 1 = 3$ interpolation points. The centred finite difference scheme uses the point of interest, at $x = x_k$, as one of the interpolation points, with the other two equidistant on either side, at $x = x_k \pm \Delta x$, as sketched below:

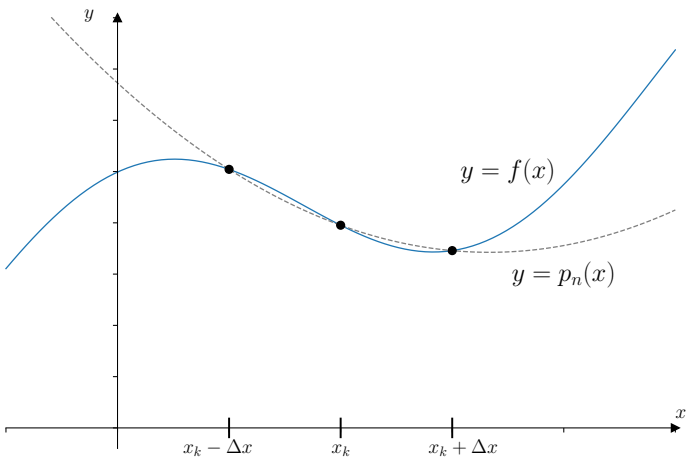


Figure 5.12: Lagrange polynomial of order 2 (quadratic) used for the centred finite difference approximation to the derivative of $f(x)$ at $x = x_k$.

So, the Lagrange polynomial will be given by

$$f(x) \sim p_2(x) = f(x_k - \Delta x)L_0(x) + f(x_k)L_1(x) + f(x_k + \Delta x)L_2(x)$$

Defining $x_0 = x_k - \Delta x$, $x_1 = x_k$ and $x_2 = x_k + \Delta x$, the basis polynomials are

$$\begin{aligned}
L_0(x) &= \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} = \frac{x^2 - (x_1 + x_2)x + x_1x_2}{(x_0 - x_1)(x_0 - x_2)} \\
L_1(x) &= \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} = \frac{x^2 - (x_0 + x_2)x + x_0x_2}{(x_1 - x_0)(x_1 - x_2)} \\
L_2(x) &= \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} = \frac{x^2 - (x_0 + x_1)x + x_0x_1}{(x_2 - x_0)(x_2 - x_1)}.
\end{aligned}$$

As we will differentiate $p_2(x)$, we need the derivatives of these basis polynomials

$$\begin{aligned}
L_0'(x) &= \frac{2x - (x_1 + x_2)}{(x_0 - x_1)(x_0 - x_2)} = \frac{2x - 2x_k - \Delta x}{2(\Delta x)^2} \\
L_1'(x) &= \frac{2x - (x_0 + x_2)}{(x_1 - x_0)(x_1 - x_2)} = \frac{2x - 2x_k}{-(\Delta x)^2} \\
L_2'(x) &= \frac{2x - (x_0 + x_1)}{(x_2 - x_0)(x_2 - x_1)} = \frac{2x - 2x_k + \Delta x}{2(\Delta x)^2}.
\end{aligned}$$

Then, we have the derivative of the Lagrange polynomial at the position $x = x_k$

$$\begin{aligned}
\frac{dp_2}{dx}(x_k) &= f(x_k - \Delta x)L_0'(x_k) + f(x_k)L_1'(x_k) + f(x_k + \Delta x)L_2'(x_k) \\
&= \frac{-f(x_k - \Delta x)}{2\Delta x} + 0 + \frac{+f(x_k + \Delta x)}{2\Delta x}
\end{aligned}$$

and so we get

Centred finite difference approximation of a derivative

$$f'(x_k) \sim \frac{f(x_k + \Delta x) - f(x_k - \Delta x)}{2\Delta x}. \tag{5.5}$$

The three methods are summarised in one figure below

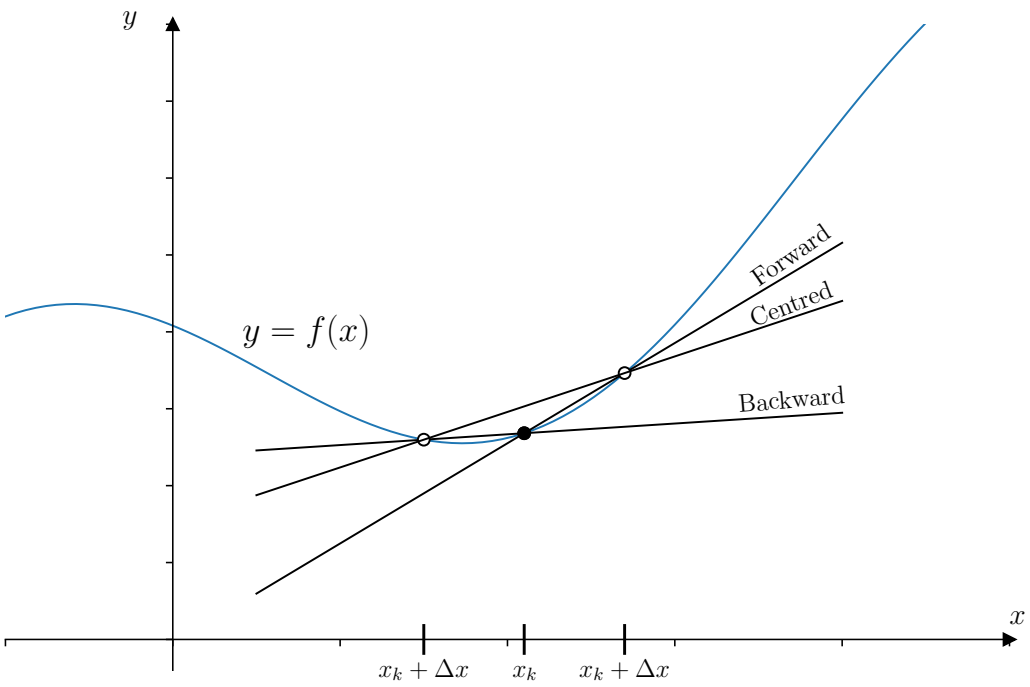


Figure 5.13: Forward, backward, and centred finite difference approximations to the derivative of $f(x)$ at the position $x = x_k$.

What about the accuracy of the centred scheme? Let's go to the Taylor series again:

$$f(x+\Delta x) = f(x) + f'(x)\Delta x + f''(x)\frac{(\Delta x)^2}{2} + f'''(x)\frac{(\Delta x)^3}{3!} + \cdots$$
$$f(x-\Delta x) = f(x) - f'(x)\Delta x + f''(x)\frac{(\Delta x)^2}{2} - f'''(x)\frac{(\Delta x)^3}{3!} + \cdots$$

Subtracting these two series removes the even terms

$$f(x+\Delta x) - f(x-\Delta x) = 2f'(x)\Delta x + 2f'''(x)\frac{(\Delta x)^3}{3!} + \cdots$$

So, rearranging for the derivative gives

$$f'(x) = \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x} - \underbrace{f'''(x)\frac{(\Delta x)^2}{3!}}_{\text{largest error term}} + \cdots$$

This methoad has the *third derivative* as the largest error term. So centred finite diffence is more accurate than backward or forward! We modeled the function with more Lagrange

polynomial interpolation points, so it's good to be rewarded for that work.

We have analysed the accuracy of these three methods using Taylor expansions. When this is possible for a scheme we define the *order of error* as power of the displacement Δx in the largest error term. Hence the backward and forward finite difference schemes are first order schemes and the centred finite difference scheme is a second order scheme.

Example 20

Given the data

x	4.5000	4.5025	4.5050	4.5075	4.5100
$f(x)$	405.563	406.472	407.383	408.296	409.210

Estimate estimate $f'(4.5050)$ by using centred finite difference with $\Delta x = 0.0025$ and with $\Delta x = 0.005$.

We are estimating the derivative at the position $x_k = 4.5050$. For $\Delta x = 0.0025$, the points we use are at $x = 4.5025$ and $x = 4.5075$. So we have

$$\begin{aligned} f'(4.5050) &= \frac{f(4.5075) - f(4.5025)}{2 \times 0.0025} \\ &= \frac{408.296 - 406.472}{0.05} \\ &\sim 364.800 \end{aligned}$$

For $\Delta x = 0.005$, the points we use are at $x = 4.5100$ and $x = 4.5000$. So we have

$$\begin{aligned} f'(4.5050) &= \frac{f(4.5100) - f(4.5000)}{2 \times 0.005} \\ &= \frac{409.210 - 405.563}{0.01} \\ &\sim 364.700 \end{aligned}$$

Chapter 6

Differential equations

In this chapter we will solve differential equations, so let's remind ourselves what they look like. A simple *first order* differential equation might look like

$$\frac{dy}{dx} = y.$$

The *order*, then, refers to the largest number of derivatives of y in this case. This equation has analytic solution

$$y(x) = Ae^x$$

for arbitrary constant A. A second order differential equation with constant coefficients could look like

$$m\frac{d^2f}{dt^2} + \beta\frac{df}{dt} + kf = \cos t$$

which has analytic solution

$$f(t) = A\cos\omega t + B\sin\omega t + C\cos t$$

for arbitrary constants A and B, and constant C as some mixture of the constants m, β, and k.

Now, these previous examples have analytic solutions because they were chosen to be simple. It turns out that most differential equations, in the sense of choosing any functions as driving terms or non-constant coefficients, do not have analytic solutions. Even an equation

as simple as

$$\frac{dy}{dx} = x - y^2$$

may not have an analytic solution.

In this chapter, we will restric ourselves to first order ordinary (1 variable) differential equations (ODEs) of the form

$$\frac{dy}{dx} = f(x, y)$$

because many higher order ODEs cn be broken down into coupled first order ODEs, and then the techniques we learn here can be applied. In addition to the differential equation, we specify initial conditions (x_0, y_0) that will constrain the solution from a family of functions down to exactly 1 function $y(x)$. Having an ODE paired with initial conditions is called an *initial value problem*. We could instead constrain the solution with 2 y values, y_A and y_B, to specify a *boundary value problem*, but we will not cover that here.

We introduce methods of finding the numerical approximation to the solution, in order of increasing accuracy but also increasing computational cost. These methods are the Euler method, explicit and implicit Trapezium methods, and Heun's method.

6.1 Euler method

Consider a flow field (you can imagine wind or water velocity vectors):

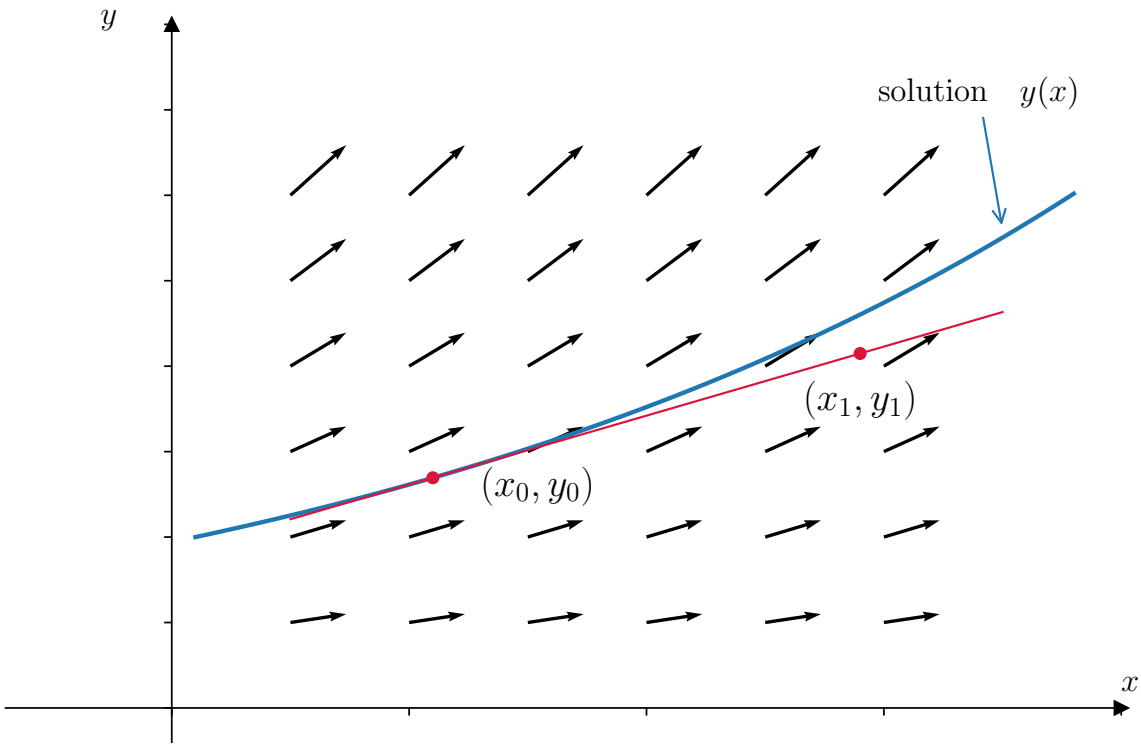


Figure 6.1: Vectors represent the slope function $f(x, y)$. Blue line is the true solution to the differential equation with particular initial conditions, whereas the red line is used for the Euler approximation of the next point in the numerical solution.

We want to find the path that a light ball would take if we drop it at the location (x_0, y_0). At each point in space, the background velocity is given by some function

$$\frac{dy}{dx} = f(x, y).$$

We can use the slope of the the solution at (x_0, y_0) to estimate the next y value, y_1, in the path after a step to x_1. The slope is given by the classic "rise over run" expression

$$m = \frac{y_1 - y_0}{x_1 - x_0}$$

and we set this slope equal to the derivative at (x_0, y_0)

$$m = \frac{dy}{dx}(x_0, y_0) = f(x_0, y_0).$$

Defining the stepsize $\Delta x = x_1 - x_0$ we then have

$$\frac{y_1 - y_0}{\Delta x} = f(x_0, y_0)$$
$$\implies y_1 = y_0 + \Delta x f(x_0, y_0)$$

So we compute a new y value from the old plus the derivative at the old position. This gives a general iterative scheme

Euler method for solving differential equations

$$y_{i+1} = y_i + \Delta x f(x_i, y_i).$$

This scheme is the simplest of the schemes we will look at, but it is also the least accurate.

Example 21
Given the initial value problem

$$\frac{dy}{dx} = -y, \quad y(0) = 1,$$

compute $y(0.1)$, $y(0.2)$, and $y(0.3)$ with the Euler method with stepsize $\Delta x = 0.1$.

The Euler method equation gives

$$y_{i+1} = y_i - y_i \Delta x = y_i(1 - \Delta x) = 0.9 y_i.$$

So each iteration of y values is just 10% smaller than the previous. For good measure let's compare this numerical solution with the analytic solution

$$y(x) = Ae^{-x}.$$

To satisfy the initial condition we must have

$$y(0) = A = 1.$$

Then we tabulate the desired values, where $y_{n,i}$ is the numerical solution and $y_a(x)$ is the analytic. We will keep only 3 decimal places at each stage of the computation. This mimics the finite precision of a computer, which of course can keep many more decimal places, which is itself a source of error independent of scheme.

i	x_i	$y_{n,i}$	$y_a(x_i)$
0	0	1	1
1	0.1	0.900	0.905
2	0.2	0.810	0.819
3	0.3	0.729	0.741

As you can see, the numerical solution is not so bad, it's correct to the first decimal place.

6.2 Trapezium method

The Trapezium rule of integration (see Chapter 5) will inspire a second method. Starting with the ODE

$$\frac{dy}{dx} = f(x, y)$$

we integrate both sides from positions $x = x_i$ to $x = x_{i+1}$

$$\int_{x_i}^{x_{i+1}} \frac{dy}{dx} dx = \int_{x_i}^{x_{i+1}} f(x, y) dx.$$

The left side can be solved with the fundamental theorem of calculus

$$\int_{x_i}^{x_{i+1}} \frac{dy}{dx} dx = y(x_{i+1}) - y(x_i)$$

and the right side can be approximated with the trapezium rule

$$\int_{x_i}^{x_{i+1}} f(x, y) dx \sim \frac{x_{i+1} - x_i}{2} \left(f(x_i, y(x_i)) + f(x_{i+1}, y(x_{i+1})) \right).$$

Using the notation $y(x_k) = y_k$ and $\Delta x = x_{i+1} - x_i$ we have

Trapezium method for solving differential equations

$$y_{i+1} = y_i + \frac{\Delta x}{2} \left(f(x_i, y_i) + f(x_{i+1}, y_{i+1}) \right).$$

Take a close look at this formula. You should be suspicious that y_{i+1} appears on both sides of the equation. We do not have method for calculating y_{i+1} based on previously known

values. However, sometimes the function $f(x, y)$ is simple enough that with a little algebraic manipulation we can find an equation with y_{i+1} equal to only previously known quantities. If we can do that, then we have developed an *explicit* Trapezium scheme. Otherwise we will have to be cleverer and develop an *implicit* Trapezium scheme. We'll look at an example which allows an explicit scheme before thinking about implicit schemes.

Example 22
Given the initial value problem

$$\frac{dy}{dx} = -y, \quad y(0) = 1,$$

compute $y(0.1)$, $y(0.2)$, and $y(0.3)$ with the Trapezium method with stepsize $\Delta x = 0.1$.
The Trapezium method equation gives

$$\begin{aligned} y_{i+1} &= y_i + \frac{\Delta x}{2}\left(-y_i - y_{i+1}\right) \\ y_{i+1} + \frac{\Delta x}{2} y_{i+1} &= y_i - \frac{\Delta x}{2} y_i \\ y_{i+1} &= y_i \frac{1 - \frac{\Delta x}{2}}{1 + \frac{\Delta x}{2}} \\ y_{i+1} &= \frac{0.95}{1.05} y_i. \end{aligned}$$

This is a simple explicit scheme! Let's tabulate the results, denoted $y_{T,i}$, comparing with the Euler solution, $y_{E,i}$, and the analytic solution, $y_a(x_i)$, given in the previous example. Also, let's keep 5 decimal places this time.

i	x_i	$y_{T,i}$	$y_{E,i}$	$y_a(x_i)$
0	0	1	1	1
1	0.1	0.90476	0.90000	0.90484
2	0.2	0.81859	0.81000	0.81873
3	0.3	0.74063	0.72900	0.74081

We see that the Trapezium method is more accurate than Euler. It is accurate to the third decimal place in this problem.

Now, if your ODE is more complicated you may not be able to make the algebraic ma-

nipulation. In this case we solve it with an implicit scheme. Define $z = y_{i+1}$ as the unknown we wish to find, while y_i, Δx, x_i and x_{i+1} are all known quantities. Then we can rearrange the Trapezium method equation to be in the form of a rootfinding problem

$$F_{i+1}(z) = z - y_i - \frac{\Delta x}{2}\left(f(x_i, y_i) + f(x_{i+1}, y_{i+1})\right) = 0.$$

So the quantity we want, y_{i+1}, is the root of $F_{i+1}(z) = 0$. This means we can use a rootfinding method (see Chapter 2)! It's most common to use Newton's method here. As rootfinding methods require an initial guess, we can even use the Euler method to give a fairly good first guess $z_0 = y_i + \Delta x f(x_i, y_i)$. So, this implicit method can be summarised by the following algorithm, for solving an ODE with initial condition $y_0 = y(x_0)$:

1. Choose a stepsize Δx;
   Suppose we've solved until (x_i, y_i)

2. Use Newton's method to find root

   - $F_{i+1}(z) = z - y_i - \frac{\Delta x}{2}\left(f(x_i, y_i) + f(x_{i+1}, y_{i+1})\right) = 0$
   - $z_{k+1} = z_k - F_{i+1}(x_k)/F'_{i+1}(x_k)$
     If the derivative of F_{i+1} can't be analytically found, then we could estimate it as in Chapter 5: e.g.,

     $$F'_{i+1}(x) \sim \frac{F_{i+1}(x + \Delta x') - F_{i+1}(x\Delta x')}{2\Delta x'}$$

     and to be more accurate we could choose $\Delta x' < \Delta x$, for example $\Delta x' = \Delta x/2$.
   - Iterate Newton's method until you satisfy a pre-chosen condition, for example until the 6th decimal stops changing.

3. The root, $z = y_{i+1} = y(x_i + \Delta x)$;

4. Go back to step 2 and repeat process as many times as you wish.

6.2.1 Heun's method

Instead of relying on luck, hoping that we happen to get an explicit method, we can sacrifice some accuracy to guarantee an explicit form. In this way we also avoid the computational complexity of an *implicit* scheme. Recall the Trapezium rule

$$y_{i+1} = y_i + \frac{\Delta x}{2}\left(f(x_i, y_i) + f(x_{i+1}, y_{i+1})\right).$$

In Heun's method we replace the y_{i+1} on the right side with an estimate $\tilde{y}_{i+1}$ given by the Euler scheme. That is,

Heun's method for solving differential equations

$$y_{i+1} = y_i + \frac{\Delta x}{2}\left(f(x_i, y_i) + f(x_{i+1}, \tilde{y}_{i+1})\right),$$
$$\text{with} \quad \tilde{y}_{i+1} = y_i + \Delta x f(x_i, y_i).$$

With this replacement we recover an explicit scheme, with y_{i+1} equal to a collection of known terms y_i, x_i and Δx. Let's see how this works.

Example 23
Given the initial value problem

$$\frac{dy}{dx} = -y, \quad y(0) = 1,$$

compute $y(0.1)$, $y(0.2)$, and $y(0.3)$ with Heun's method for stepsize $\Delta x = 0.1$.
Euler's method gives the first approximation of y_{i+1} as calculated in example 6.1

$$\tilde{y}_{i+1} = 0.9y_i.$$

This is inserted into the Trapezium rule

$$\begin{aligned} y_{i+1} &= y_i + \frac{\Delta x}{2}\left(f(x_i, y_i) + f(x_{i+1}, \tilde{y}_{i+1})\right) \\ &= y_i + \frac{0.1}{2}\left(-y_i + 0.9y_i\right) \\ &= y_i + y_i\frac{0.1}{2}\left(-0.1\right) \\ &= y_i - y_i 0.005 \\ y_{i+1} &= 0.905y_i. \end{aligned}$$

There we have our simple explicit scheme. Note the previous full Trapezium rule explicit scheme in the previous example gave

$$y_{i+1} = \frac{0.95}{1.05}y_i \sim 0.90476y_i$$

which is the same if we cutoff at 3 decimal places.

Let's tabulate the results, denoted $y_{H,i}$, comparing with the Trapezium solution, $y_{T,i}$, Euler solution, $y_{E,i}$, and the analytic solution, $y_a(x_i)$, given in the previous examples. Also, let's keep 5 decimal places this time.

i	x_i	$y_{H,i}$	$y_{T,i}$	$y_{E,i}$	$y_a(x_i)$
0	0	1	1	1	1
1	0.1	0.90500	0.90476	0.90000	0.90484
2	0.2	0.81903	0.81859	0.81000	0.81873
3	0.3	0.74122	0.74063	0.72900	0.74081

We see that Heun's method is less accurate than the Trapezium method, but more accurate than Euler. It is accurate to the second decimal place in this problem.

6.3 Reduction of second order differential equations

6.4 Summary

We have looked at four methods for computing numerical solutions to first order differential equations, the Euler method, explicit and implicit Trapezium methods, and Heun's method. The main take away message is the balancing act of accuracy and complexity. We always pay for more accuracy with complexity. In the explicit schemes, the end results were equations that were equally easy to iterate with a computer, that is, a computer would take the same amount of time. But we paid for it with complexity in the algebraic derivation of the schemes. In the implicit scheme, the algebraic derivation is simple, but the computational power is shifted to the implementation of the root-finding algorithm, and so it will generally take a computer longer to calculate the numerical solution than with the explicit schemes. However, the implicit schemes will be more flexible in the sense of being able to work on complicated functions that don't require you to do much algebraic work (often pen and paper) before throwing the problem at the computer.

Chapter 7

Matrix methods

Consider a system of N linear equations

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1N}x_N &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2N}x_N &= b_2 \\ &\vdots \\ a_{N1}x_1 + a_{N2}x_2 + \cdots + a_{NN}x_N &= b_N \end{aligned}$$

where the a_{ij} are constant coefficients, the b_i are also constants, and the x_i are the variables we are trying to determine. This system can be written more succinctly as a matrix equation:

$$\underbrace{\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}}_{X} = \underbrace{\begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{pmatrix}}_{B}$$

So, the goal is to find the column X. The obvious method to do this is to invert the matrix A, so that we can multiply both sides of the matrix equation by A^{-1}, giving the solution directly

$$X = A^{-1}B,$$

assuming that the inverse exists. This assumption is true if and only if A has non-zero determinant: $\det(A) \neq 0$.

7.1 Gaussian reduction

Let's quickly remind ourselves how Gaussian reduction works with an example. This example will also serve to define various useful terms.

Example 24
Consider the system

$$\begin{pmatrix} 2 & 1 & 2 \\ 1 & 3 & 3 \\ 3 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -1 \\ 2 \\ 0 \end{pmatrix}.$$

To simplify we work with the augmented matrix:

$$\left(\begin{array}{ccc|c} \boxed{2} & 1 & 2 & -1 \\ 1 & 3 & 3 & 2 \\ 3 & 1 & 1 & 0 \end{array}\right)$$

I have boxed the first element of the first row. At this step, this is the *pivot* or *pivot element*. We use a pivot to zero the rest of the column below it. In the first step the pivot will always be the leftmost uppermost non-zero element. Since row 2 has a 1 in the first column, we zero it by subtracting 1/2 of the first row (or, said differently, replacing row 2 with row 2 minus half of row 1).

$$\begin{array}{c} R_2 - \frac{1}{2}R_1 \\ \rightarrow \\ R_3 - \frac{3}{2}R_1 \end{array} \quad \left(\begin{array}{ccc|c} 2 & 1 & 2 & -1 \\ 0 & \boxed{5/2} & 2 & 5/2 \\ 0 & -1/2 & -2 & 3/2 \end{array}\right)$$

Now that the first column has been zeroed below the first pivot, we move on to the second column. So now the pivot is the element in the second column and second row, which is boxed above. [Note: If this element was zero, we would have to switch the second row with another row (not including the first) in order to get a nonzero pivot there. If there were no nonzero elements in the second row to move here, we'd be in trouble. In fact the system would be under-determined and we would have infinite solutions. We'll ignore this situation for the moment.] We use the new pivot to zero the rest of the column below it. We have to take the 3rd row, and subtract minus 1/5th of the 2nd row. That is, add 1/5th of the 2nd

row:

$$\begin{matrix} \\ \to \\ R_3 + \frac{1}{5}R_2 \end{matrix} \quad \left(\begin{array}{ccc|c} 2 & 1 & 2 & -1 \\ 0 & 5/2 & 2 & 5/2 \\ 0 & 0 & -8/5 & 2 \end{array}\right)$$

Now we end up with a matrix (everything left of the vertical bar) in *upper triangular form.* We unpack this starting from the bottom:

$$\begin{aligned} -\frac{8}{5}x_3 = 2 \quad &\implies \quad x_3 = -\frac{5}{4} \\ \frac{5}{2}x_2 + 2x_3 = \frac{5}{2} \quad &\implies \quad x_2 = \frac{2}{5}\left(\frac{5}{2} - 2x_3\right) = \frac{2}{5}\left(\frac{5}{2} + \frac{5}{2}\right) = 2 \\ 2x_1 + x_2 + 2x_3 = -1 \quad &\implies \quad x_1 = \frac{1}{2}\left(-1 - x_2 - 2x_3\right) = \frac{1}{2}\left(-1 - 2 + \frac{5}{2}\right) = -\frac{1}{4} \end{aligned}$$

So, the solution to this system is $(x_1, x_2, x_3) = (-1/4, 2, -5/4)$. Geometrically, are 3 equations were each an equation for a plane. The solution represents the point at which all 3 planes intersect.

Take note of this previous example. We will soon write down very general things to codify this process, and if you're ever lost on exactly what we've done, come back to this concrete example of the steps of Gaussian reduction. In this 3×3 example, everything we will do in the following paragraphs can be seen in practice.

Imagine now a general system of N equations and N unknowns, given by $AX = B$. At the n^{th} step of the reduction process we reach a matrix $A^{(n)}$. The matrix equation looks like

$$A^{(n)}X = b^{(n)} \to \underbrace{\left(\begin{array}{ccccccc|c} a_{11}^{(n)} & a_{12}^{(n)} & a_{13}^{(n)} & \cdots & a_{1n}^{(n)} & a_{1,n+1}^{(n)} & \cdots & b_1 \\ 0 & a_{22}^{(n)} & a_{23}^{(n)} & \cdots & a_{2n}^{(n)} & a_{2,n+1}^{(n)} & \cdots & b_2 \\ 0 & 0 & a_{33}^{(n)} & \cdots & a_{3n}^{(n)} & a_{3,n+1}^{(n)} & \cdots & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn}^{(n)} & a_{n,n+1}^{(n)} & \cdots & b_n \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & a_{N,n+1}^{(n)} & \cdots & b_N \end{array}\right)}_{\text{zeros under the diagonal for the first } n \text{ columns.}}$$

Our goal is to give concrete expressions of these coefficients, $a_{ij}^{(n)}$, as though we were going to code them in a computer program. Recall that for $a_{ij}^{(n)}$, this is the element in the ith

row and jth column. For notational consistency, let's rename the original matrices $A = A^{(0)}$ and $B = B^{(0)}$. So, let's do this slowly. In the first step,the $a_{11}^{(0)}$ element is the pivot that we use to zero the column below it. To do that we have to take an arbitrary row, R_k, and subtract from it the right amount, call it $C_k^{(1)}$, of row 1 to remove its first element $a_{k1}^{(0)}$, that is $R_k \to R_k - C_k^{(1)} R_i$. The first element is thus forced to be zero like so

$$a_{k1}^{(1)} = a_{k1}^{(0)} - C_k^{(1)} a_{11}^{(0)} = 0$$

which determines this constant $C_k^{(1)}$

$$C_k^{(1)} = \frac{a_{k1}^{(0)}}{a_{11}^{(0)}}.$$

So after the first step the matrix equation will look like

$$A^{(1)}X = b^{(1)} \to \left(\begin{array}{cccc|c} a_{11}^{(0)} & a_{12}^{(0)} & a_{13}^{(0)} & \cdots & a_{1n}^{(0)} & b_1 \\ 0 & a_{21}^{(0)} - \frac{a_{21}^{(0)}}{a_{11}^{(0)}} a_{11}^{(0)} & a_{22}^{(0)} - \frac{a_{21}^{(0)}}{a_{11}^{(0)}} a_{13}^{(0)} & \cdots & a_{2N}^{(0)} - \frac{a_{21}^{(0)}}{a_{11}^{(0)}} a_{1N}^{(0)} & b_2^{(0)} - \frac{a_{21}^{(0)}}{a_{11}^{(0)}} b_{1N}^{(0)} \\ 0 & a_{31}^{(0)} - \frac{a_{31}^{(0)}}{a_{11}^{(0)}} a_{11}^{(0)} & a_{32}^{(0)} - \frac{a_{31}^{(0)}}{a_{11}^{(0)}} a_{13}^{(0)} & \cdots & a_{3N}^{(0)} - \frac{a_{31}^{(0)}}{a_{11}^{(0)}} a_{1N}^{(0)} & b_3^{(0)} - \frac{a_{31}^{(0)}}{a_{11}^{(0)}} b_{1N}^{(0)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & a_{N1}^{(0)} - \frac{a_{31}^{(0)}}{a_{11}^{(0)}} a_{11}^{(0)} & a_{N2}^{(0)} - \frac{a_{31}^{(0)}}{a_{11}^{(0)}} a_{13}^{(0)} & \cdots & a_{NN}^{(0)} - \frac{a_{N1}^{(0)}}{a_{11}^{(0)}} a_{1N}^{(0)} & b_N^{(0)} - \frac{a_{31}^{(0)}}{a_{11}^{(0)}} b_{1N}^{(0)} \end{array}\right)$$

Now this matrix is tough to look at. I think it's easier to focus on the elements, $a_{ij}^{(0)}$, themselves. The easiest thing to say is that the first column, other than the first row, will be zero, so

$$a_{k1}^{(1)} = 0, \quad \text{for } k = 2, 3, 4, \ldots, N.$$

We can see also that the whole first row stays the same

$$a_{1k}^{(1)} = a_{1k}^{(0)}, \quad \text{for } k = 1, 2, 3, \ldots, N.$$

Now each of the rest of the elements are replaced by their old value minus a multiple (that we found earlier) of the first row, so

$$a_{ij}^{(1)} = a_{ij}^{(0)} - \frac{a_{i1}^{(0)}}{a_{11}^{(0)}} a_{1j}^{(0)}, \quad \text{for any } i, j = 2, 3, \ldots, N.$$

It's a good idea to stare at these last three equations until you convince yourself that you

understand them. One question you should ask is if you choose any possible i or j (between 1 and N), do one of these three equations apply? Finally, in the Gaussian reduction we also have to work on the B column in the augmented matrix. Nothing new, it's the same rules as above. The first element is unchanged

$$b_1^{(1)} = b_1^{(0)},$$

but all the rest have the same subtraction factor as in the matrix

$$b_i^{(1)} = b_i^{(0)} - \frac{a_{i1}^{(0)}}{a_{11}^{(0)}} b_i^{(0)}, \quad \text{for } i = 2, 3, \ldots, N.$$

Now let's look at what happens at step 2. We've zeroed the first column (below the top left element) and it remains untouched

$$a_{k1}^{(2)} = a_{k1}^{(1)}, \quad \text{for } k = 1, 2, 3, \ldots, N.$$

So now we need to zero the second column *below the diagonal*

$$a_{k2}^{(2)} = 0, \quad \text{for } k = 3, 4, 5, \ldots, N.$$

The first *and* second row remain the same

$$\begin{aligned} a_{1k}^{(2)} &= a_{1k}^{(1)}, \quad \text{for } k = 2, 3, 4, \ldots, N. \\ a_{2k}^{(2)} &= a_{2k}^{(1)}, \quad \text{for } k = 2, 3, 4, \ldots, N. \end{aligned}$$

And now we need to subtract a multiple of the second row from all the other rows (from row 3 downwards), by that multiple, $C_k^{(2)}$, that zerod the second row

$$C_k^{(2)} = \frac{a_{k2}^{(2)}}{a_{22}^{(1)}}.$$

So any element (not in the first two rows or columns) is replaced by

$$a_{ij}^{(2)} = a_{ij}^{(1)} - \frac{a_{i2}^{(1)}}{a_{22}^{(1)}} a_{2j}^{(1)}, \quad \text{for any } i, j = 3, 4, \ldots, N.$$

Now for the B column, the first two elements are unchanged

$$b_1^{(2)} = b_1^{(1)},$$
$$b_2^{(2)} = b_2^{(1)}$$

but all the rest have the same subtraction factor as in the matrix

$$b_i^{(2)} = b_i^{(1)} - \frac{a_{i2}^{(1)}}{a_{22}^{(1)}} b_i^{(1)}, \quad \text{for } i = 3, 4, \ldots, N.$$

OK, after 2 steps I think we can recognise the pattern. Now let's skip to the nth step. These will be the expressions for all the matrix elements, and B column terms, that could be used in a computer program to iteratively complete the Gaussian reduction procedure until we reach an upper triangular matrix.

n steps of Gaussian reduction: $A^{(n)}X = b^{(n)}$

At this step, the first n columns have been zeroed below the diagonal, and so those columns remain untouched on every row

$$a_{ij}^{(n)} = a_{ij}^{(n-1)}, \quad \text{for } i = 1, 2, 3, \ldots, N \quad \text{and } j = 1, 2, 3, \ldots, n.$$

Similarly, the first n rows remain the same rightwards of those first n columns

$$a_{ij}^{(n)} = a_{ij}^{(n-1)}, \quad \text{for } i = 1, 2, 3, \ldots, n \quad \text{and } j = n+1, n+2, n+3, \ldots, N.$$

The rest of the elements must subtract a multiple of the n row

$$a_{ij}^{(n)} = a_{ij}^{(n-1)} - \frac{a_{in}^{(n-1)}}{a_{nn}^{(n-1)}} a_{nj}^{(n-1)}, \quad \text{for any } i, j = n+1, n+2, n+3, \ldots, N.$$

This has assumed that $a_{nn}^{(n-1)} \neq 0$. If this element is zero, we need to switch the nth row with another row larger than n. If none of the lower rows have a non-zero element in this column, then we simply let $A^{(n)} = A^{(n-1)}$, and continue with the next step.
For the B column, the first n elements remain the same

$$b_i^{(n)} = b_i^{(n-1)}, \quad \text{for } i = 1, 2, 3, \ldots, n$$

and the remaining elements have the same subtraction factor as the A elements

$$b_i^{(n)} = b_i^{(n-1)} - \frac{a_{in}^{(n-1)}}{a_{nn}^{(n-1)}} b_i^{(n-1)}, \quad \text{for } i = n+1, n+2, n+3, \ldots, N.$$

Once we have reduced the matrix A to an upper triangular matrix, then we can determine the solution by unpacking the matrix equation back into explicit equations starting from the bottom of the matrix. This is called the method of ascent, which we detail in the next section.

7.1.1 Method of ascent and method of descent

If we have a matrix equation in upper triangular form

$$\left(\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \cdots & a_{1N} & b_1 \\ 0 & a_{22} & a_{23} & \cdots & a_{2N} & b_2 \\ 0 & 0 & a_{33} & \cdots & a_{3N} & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{NN} & b_N \end{array}\right)$$

we unpack the matrix equation starting from the bottom. The last line gives

$$a_{NN}x_N = b_N \quad \Longrightarrow \quad x_N = b_N/a_{NN}.$$

The second last line gives

$$\begin{aligned} a_{N-1,N-1}x_{N-1} + a_{N-1,N}x_N = b_{N-1} \quad \Longrightarrow \quad x_{N-1} &= \frac{1}{a_{N-1,N-1}}\left(b_{N-1} - a_{N-1,N}x_N\right) \\ &= \frac{1}{a_{N-1,N-1}}\left(b_{N-1} - a_{N-1,N}\frac{b_N}{a_{NN}}\right). \end{aligned}$$

We repeat this procedure succissively to determine every x_i. If we look at row i, we see a bunch of zeros before the x_i term

$$0x_1 + 0x_2 + \cdots + 0x_{i-1} + a_{ii}x_i + a_{i,i+1}x_{i+1} + \cdots + a_{i,N}x_N = b_i.$$

This allows us to see the general expression computing the solution

$$\boxed{x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{k=i+1}^{N} a_{ik}x_k\right).}$$

Be careful, this expression must be applied successively, *starting from the end*, $i = N$. This is the reason it is the method of ascent.

Now a related method is just the reverse of this. If you have a *lower triangular matrix* equation, then *starting from the top*, you will reach an equation like

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{i,i-1}x_{i-1} + a_{ii}x_i + 0x_{i+1} + \cdots + 0x_N = b_i.$$

The method of *descent* then gives the general expression

$$\boxed{x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{k=1}^{i-1} a_{ik} x_k \right).}$$

7.1.2 Computational problem with Gaussian reduction

To briefly summarise, with Gaussian reduction we solve a matrix equation $AX = B$ with successive steps representing row operations with the goal of achieving an upper triangular matrix. Once we reach this goal, we apply the method of ascent to find the solution X. But what if we merely slightly change the original equation to $AX = C$? If you look through the algorithm, you'll notice that we would need to start the whole thing again, because the rightmost column in the augmented matrix comes along for the ride. It needs to experience every step in order. So unless you want to save every row operation (which is certainly feasible, you only need to save the $C_k^{(n)}$ for every step), then we need to find a new method. We will look at 2 methods that bypass this problem in the next sections, the method of LU decomposition and the Jacobi iteration method.

7.2 LU decomposition

Decomposition

For a square matrix $A \in \mathcal{M}_{n\times n}(\mathbb{R})$ we seek to decompose it into a product of two matrices

$$A = LU$$

where the matrix L is an $n \times n$ lower triangular matrix and the matrix U is an $n \times n$ upper triangular matrix. We further make the assumption that L has only 1s along the diagonal. Both L and U are results of a Gaussian reduction process. For example, let A be the following 3×3 matrix and start by constructing L as follows

$$A = \begin{pmatrix} 3 & 1 & 2 \\ 1 & 2 & 2 \\ 2 & 1 & 3 \end{pmatrix} \quad \text{and} \quad L = \begin{pmatrix} 1 & 0 & 0 \\ \cdot & 1 & 0 \\ \cdot & \cdot & 1 \end{pmatrix}$$

We perform the Gaussian reduction and use the row operation to determine L.

$$\begin{array}{lcccc} R_2 - (1/3)\,R_1 & \to & A' = \begin{pmatrix} 3 & 1 & 2 \\ 0 & 5/3 & 4/3 \\ 0 & 1/3 & 5/3 \end{pmatrix} & \to & L = \begin{pmatrix} 1 & 0 & 0 \\ 1/3 & 1 & 0 \\ 2/3 & \cdot & 1 \end{pmatrix} \\ R_3 - (2/3)\,R_1 & & & & \\ & \to & A'' = \begin{pmatrix} 3 & 1 & 2 \\ 0 & 5/3 & 4/3 \\ 0 & 0 & 7/5 \end{pmatrix} & \to & L = \begin{pmatrix} 1 & 0 & 0 \\ 1/3 & 1 & 0 \\ 2/3 & 1/5 & 1 \end{pmatrix} \\ R_3 - (1/5)\,R_2 & & & & \end{array}$$

Now this last reduced matrix is exactly the upper triangular $U = A''$ we are looking for. So the decomposition is

$$A = \begin{pmatrix} 3 & 1 & 2 \\ 1 & 2 & 2 \\ 2 & 1 & 3 \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 1/3 & 1 & 0 \\ 2/3 & 1/5 & 1 \end{pmatrix}}_{L} \underbrace{\begin{pmatrix} 3 & 1 & 2 \\ 0 & 5/3 & 4/3 \\ 0 & 0 & 7/5 \end{pmatrix}}_{U}$$

What you can see in this example is that U is simply the final result of the Gaussian reduction, and L encodes the row operations performed in that reduction. We saw that did row operations

$$\begin{aligned} R_2 - (1/3)\,R_1 \\ R_3 - (2/3)\,R_1 \end{aligned}$$

and that gave us L entries $L_{21} = 1/3$ and $L_{31} = 2/3$. The indices of the rows in the operation give us the row/column indices for the entries in L. Generally, we then perform row operations

$$R_i - L_{ij} R_j$$

where these operations must use a pivot to zero the column below the pivot.

Usage

Why do we do this LU decomposition? In the end we are still trying to solve linear systems, which have matrix form

$$AX = B$$

where X is the column of n unknowns (variables) we are trying to solve for and B is a column of known values. The LU decomposition allows us to solve this system without taking an inverse. The algorithm runs as follows

$$AX = B \quad \Longrightarrow \quad (LU)X = B \quad \Longrightarrow \quad L(UX) \qquad\qquad = B$$

Now UX is an $n \times n$ matrix multiplied by a column of size n. So it must result in another column of size n, call it Y. That is, we have

$$L\,(UX) = B \quad \Longrightarrow \quad LY = B.$$

This last equation gives a lower triangular matrix, L, multplied by a column of unknowns, Y, equal to a column of known values B. That's really easy to solve! We use the method of *descent* to solve it, turning Y into a column of known values. But we earlier defined

$$UX = Y$$

which is also an easy problem to solve. This is an upper triangular matrix, U, multiplied by a column of unknowns, X, equal to a column of known values Y. Thus we use the method of *ascent* to solve it, giving us what we want, the variables in X. Let's use the previous example LU decomposition to see this in practice.

Example 25
Recall

$$A = \begin{pmatrix} 3 & 1 & 2 \\ 1 & 2 & 2 \\ 2 & 1 & 3 \end{pmatrix} = LU \quad \text{for} \quad L = \begin{pmatrix} 1 & 0 & 0 \\ 1/3 & 1 & 0 \\ 2/3 & 1/5 & 1 \end{pmatrix} \quad \text{and} \quad U = \begin{pmatrix} 3 & 1 & 2 \\ 0 & 5/3 & 4/3 \\ 0 & 0 & 7/5 \end{pmatrix}$$

Let's solve the system

$$\begin{pmatrix} 3 & 1 & 2 \\ 1 & 2 & 2 \\ 2 & 1 & 3 \end{pmatrix} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}}_{X} = \underbrace{\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}}_{B}$$

So we first use $LUX = B$ to define a new column of unknowns

$$Y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

such that $Y = UX$. This let's us write the equation $LY = B$, explicitly

$$\begin{pmatrix} 1 & 0 & 0 \\ 1/3 & 1 & 0 \\ 2/3 & 1/5 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

With the *method of descent* we then have

$$\begin{aligned} & y_1 = 1 \\ & (1/3)y_1 + y_2 = 2 \quad \Longrightarrow \quad y_2 = 5/3 \\ & (2/3)y_1 + (1/5)y_2 + y_3 = 2 \quad \Longrightarrow \quad y_3 = 2 \end{aligned}$$

Now we have determined the Y column, we have

$$UX = Y \quad \Longrightarrow \quad \begin{pmatrix} 3 & 1 & 2 \\ 0 & 5/3 & 4/3 \\ 0 & 0 & 7/5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 5/3 \\ 2 \end{pmatrix}$$

With the *method of ascent* we then have

$$\begin{aligned} & (7/5)x_3 = 2 \quad \Longrightarrow \quad x_3 = 10/7 \\ & (5/3)x_2 + (4/3)x_3 = 5/3 \quad \Longrightarrow \quad x_2 = -1/7 \\ & 3x_1 + x_2 + 2x_3 = 1 \quad \Longrightarrow \quad x_1 = -4/7 \end{aligned}$$

So that the unique solution to the system $AX = B$ is given by $(x_1,\, x_2,\, x_3) = (-4/7,\, -1/7,\, 10/7)$.

7.3 Iterative matrix methods

We still seek to solve linear systems of the form $AX = B$ for square matrix A. We split the matrix A into an addition of matrices

$$A = M + N$$

so that

$$\begin{aligned} & AX = B \\ \implies & (M+N)X = B \\ \implies & MX = B - NX \\ \implies & X = M^{-1}B - M^{-1}NX \end{aligned}$$

This last equation gives the iterative scheme

$$X^{(k+1)} = M^{-1}\left(B - NX^{(k)}\right).$$

The choices of M and N give rise to different iterative schemes, with different speeds of convergence. We will look at two sets of choices, giving the schemes of Jacobi iteration and Gauss-Seidel iteration.

7.3.1 Jacobi Iteration

In this scheme, we choose M to be the matrix of diagonal elements of A, called D, and we choose N to be the matrix of the rest of the elements, called O for "off-diagonal". Explicitly, these two matrices are defined as follows

$$D_{ij} = \begin{cases} A_{ij} & \text{when } i = j \\ 0 & \text{when } i \neq j \end{cases}$$

$$O_{ij} = \begin{cases} 0 & \text{when } i = j \\ A_{ij} & \text{when } i \neq j \end{cases}$$

The Jacobi iteration is therefore given by

$$X^{(k+1)} = D^{-1}\left(B - OX^{(k)}\right).$$

Example 26

Let's use Jacobi iteration on the system

$$\begin{cases} 3x - y + z = 2 \\ x - 2y - z = 1 \\ 2x + 2y + 4z = 3 \end{cases}$$

As a matrix equation this system is given by

$$\underbrace{\begin{pmatrix} 3 & -1 & 1 \\ 1 & -2 & -1 \\ 2 & 2 & 4 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} x \\ y \\ z \end{pmatrix}}_{X} = \underbrace{\begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix}}_{B}$$

Hence we split A with a diagonal and off-diagonal matrix

$$\begin{pmatrix} 3 & -1 & 1 \\ 1 & -2 & -1 \\ 2 & 2 & 4 \end{pmatrix} = \underbrace{\begin{pmatrix} 3 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 4 \end{pmatrix}}_{D} + \underbrace{\begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ 2 & 2 & 0 \end{pmatrix}}_{O}$$

So we have the Jacobi iteration scheme

$$\begin{aligned} \begin{pmatrix} x_{k+1} \\ y_{k+1} \\ z_{k+1} \end{pmatrix} &= \begin{pmatrix} 3 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 4 \end{pmatrix}^{-1} \left(\begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix} - \begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ 2 & 2 & 0 \end{pmatrix} \begin{pmatrix} x_k \\ y_k \\ z_k \end{pmatrix} \right) \\ &= \begin{pmatrix} 1/3 & 0 & 0 \\ 0 & -1/2 & 0 \\ 0 & 0 & 1/4 \end{pmatrix} \left(\begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix} - \begin{pmatrix} -y_k + z_k \\ x_k - z_k \\ 2x_k + 2y_k \end{pmatrix} \right) \\ &= \begin{pmatrix} 1/3 & 0 & 0 \\ 0 & -1/2 & 0 \\ 0 & 0 & 1/4 \end{pmatrix} \begin{pmatrix} 2 + y_k - z_k \\ 1 - x_k + z_k \\ 3 - 2x_k - 2y_k \end{pmatrix} \\ \begin{pmatrix} x_{k+1} \\ y_{k+1} \\ z_{k+1} \end{pmatrix} &= \begin{pmatrix} \frac{1}{3}(2 + y_k - z_k) \\ -\frac{1}{2}(1 - x_k + z_k) \\ \frac{1}{4}(3 - 2x_k - 2y_k) \end{pmatrix} \end{aligned}$$

With this example, I want to show that there is a much easier way to derive this iteration scheme, without ever thinking about diagonal and off-diagonal matrices. Let's start with the system of equations

$$\begin{cases} 3x - y + z = 2 \\ x - 2y - z = 1 \\ 2x + 2y + 4z = 3 \end{cases} \implies \begin{cases} x = \frac{1}{3}(2 + y - z) \\ y = -\frac{1}{2}(1 - x + z) \\ z = \frac{1}{4}(3 - 2x - 2y) \end{cases}$$

We used the first equation to isolate the x variable, the second equation to isolate the y variable and the third equation to isolate the z variable. Now we interprate these new 3 equations as giving an iterative scheme

$$\begin{cases} x_{k+1} = \dfrac{1}{3}(2 + y_k - z_k) \\ y_{k+1} = -\dfrac{1}{2}(1 - x_k + z_k) \\ z_{k+1} = \dfrac{1}{4}(3 - 2x_k - 2y_k) \end{cases}$$

and we have the same 3 equations that we found with the matrix manipulations, but in a fashion far easier and direct. If we make an initial guess of $(x_0, y_0, z_0) = (0, 0, 0)$, the first 5 iterations are shown below, making sure to keep only 4 decimal places at each iteration

k	x_k	y_k	z_k
0	0	0	0
1	0.6667	-0.5000	0.7500
2	0.2500	-0.5416	0.6666
3	0.2639	-0.7083	0.8958
4	0.1320	-0.8159	0.9722
5	0.0706	-0.9210	1.0919

It's not clear whether this converges or not at this stage.

7.3.2 Gauss-Seidel Iteration

In this scheme, we choose M to be the matrix of lower diagonal elements of A, called L, and we choose N to be the matrix of the rest of the elements, called U^* for the *strictly* upper

diagonal elements of A. Explicitly, these two matrices are defined as follows

$$L_{ij} = \begin{cases} A_{ij} & \text{when } i \geq j \\ 0 & \text{when } i < j \end{cases}$$

$$U^*_{ij} = \begin{cases} 0 & \text{when } i \geq j \\ A_{ij} & \text{when } i < j \end{cases}$$

Note that L contains the diagonal of A and U^* *does not contain the diagaonal* of A. The Gauss-Seidel iteration is therefore given by

$$X^{(k+1)} = L^{-1}\left(B - U^* X^{(k)}\right).$$

Example 27

Let's take up the previous system to compare the Gauss-Seidel iteration scheme to the Jacobi scheme for the same system. We thus have the matrix system

$$\underbrace{\begin{pmatrix} 3 & -1 & 1 \\ 1 & -2 & -1 \\ 2 & 2 & 4 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} x \\ y \\ z \end{pmatrix}}_{X} = \underbrace{\begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix}}_{B}$$

Hence we split A with a lower triangular and *strictly* upper triangular matrix

$$\begin{pmatrix} 3 & -1 & 1 \\ 1 & -2 & -1 \\ 2 & 2 & 4 \end{pmatrix} = \underbrace{\begin{pmatrix} 3 & 0 & 0 \\ 1 & -2 & 0 \\ 2 & 2 & 4 \end{pmatrix}}_{L} + \underbrace{\begin{pmatrix} 0 & -1 & 1 \\ 0 & 0 & -1 \\ 0 & 0 & 0 \end{pmatrix}}_{U^*}$$

So we have the Gauss-Seidel iteration scheme

$$\begin{pmatrix} x_{k+1} \\ y_{k+1} \\ z_{k+1} \end{pmatrix} = \begin{pmatrix} 3 & 0 & 0 \\ 1 & -2 & 0 \\ 2 & 2 & 4 \end{pmatrix}^{-1} \left(\begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix} - \begin{pmatrix} 0 & -1 & 1 \\ 0 & 0 & -1 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_k \\ y_k \\ z_k \end{pmatrix} \right) = \begin{pmatrix} 3 & 0 & 0 \\ 1 & -2 & 0 \\ 2 & 2 & 4 \end{pmatrix}^{-1} \begin{pmatrix} 2 + y_k - z_k \\ 1 + z_k \\ 3 \end{pmatrix}$$

Now we have an inverse that is a little bit more difficult than for a diagonal. Fortunately determinants of triangular matrices are always just the multiplication of the diagonal. Let's

use the cofactor form of the inverse

$$\begin{pmatrix} 3 & 0 & 0 \\ 1 & -2 & 0 \\ 2 & 2 & 4 \end{pmatrix}^{-1} = \frac{1}{\det(L)} \begin{pmatrix} \begin{vmatrix} -2 & 0 \\ 2 & 4 \end{vmatrix} & -\begin{vmatrix} 1 & 0 \\ 2 & 4 \end{vmatrix} & \begin{vmatrix} 1 & -2 \\ 2 & 2 \end{vmatrix} \\ -\begin{vmatrix} 0 & 0 \\ 2 & 4 \end{vmatrix} & \begin{vmatrix} 3 & 0 \\ 2 & 4 \end{vmatrix} & -\begin{vmatrix} 3 & 0 \\ 2 & 2 \end{vmatrix} \\ \begin{vmatrix} 0 & 0 \\ -2 & 0 \end{vmatrix} & -\begin{vmatrix} 3 & 0 \\ 1 & 0 \end{vmatrix} & \begin{vmatrix} 3 & 0 \\ 1 & -2 \end{vmatrix} \end{pmatrix}^T = \frac{-1}{24} \begin{pmatrix} -8 & 0 & 0 \\ -4 & 12 & 0 \\ 6 & -6 & -6 \end{pmatrix}$$

So we continue where we left off

$$\begin{aligned} \begin{pmatrix} x_{k+1} \\ y_{k+1} \\ z_{k+1} \end{pmatrix} &= \frac{-1}{24} \begin{pmatrix} -8 & 0 & 0 \\ -4 & 12 & 0 \\ 6 & -6 & -6 \end{pmatrix} \begin{pmatrix} 2 + y_k - z_k \\ 1 + z_k \\ 3 \end{pmatrix} \\ &= \begin{pmatrix} \frac{2}{3} + \frac{1}{3} y_k - \frac{1}{3} z_k \\ -\frac{1}{6} + \frac{1}{6} y_k - \frac{2}{3} z_k \\ \frac{1}{2} - \frac{1}{4} y_k \end{pmatrix} \end{aligned}$$

As with the Jacobi example, there is a much easier way to derive this iteration scheme, without ever thinking about lower triangular (and its inverse!) and upper triangular matrices. We do the same as with Jacobi, starting with the system of equations we isolate each variable

$$\begin{cases} 3x - y + z = 2 \\ x - 2y - z = 1 \\ 2x + 2y + 4z = 3 \end{cases} \implies \begin{cases} x = \frac{1}{3}(2 + y - z) \\ y = -\frac{1}{2}(1 - x + z) \\ z = \frac{1}{4}(3 - 2x - 2y) \end{cases}$$

As before we used the first equation to isolate the x variable, the second equation to isolate the y variable and the third equation to isolate the z variable. Now we interprate these new 3 equations as giving an iterative scheme, but using the latest possible information. That is, the first equation gives

$$x_{k+1} = \frac{1}{3}(2 + y_k - z_k)$$

Now a computer has this new iteration of x in it's memory, so let's use this immediately! The second equation gives the updated y

$$y_{k+1} = -\frac{1}{2}(1 - x_{k+1} + z_k)$$

Now we have new iterations of both x and y, which can be used in the computation of z

$$z_{k+1} = \frac{1}{4}\left(3 - 2x_{k+1} - 2y_{k+1}\right)$$

But these aren't the 3 equations we found earlier! Or are they? The first equation is the same. The equation for y_{k+1} contains an x_{k+1}, so insert that into it

$$y_{k+1} = -\frac{1}{2}\left(1 - \frac{1}{3}\left(2 + y_k - z_k\right) + z_k\right) = -\frac{1}{6} + \frac{1}{6}y_k - \frac{2}{3}z_k$$

Now this is the same as in the matrix. Insert these y_{k+1} and x_{k+1} into the equation for z_{k+1}

$$z_{k+1} = \frac{1}{4}\left(3 - 2\frac{1}{3}\left(2 + y_k - z_k\right) - 2\left(-\frac{1}{6} + \frac{1}{6}y_k - \frac{2}{3}z_k\right)\right) = \frac{1}{2} - \frac{1}{4}y_k$$

and so all three equations are the same. But! We should not do this rearrangement. The system built from the simple rearrangement

$$\begin{cases} x_{k+1} = \dfrac{1}{3}\left(2 + y_k - z_k\right) \\ y_{k+1} = -\dfrac{1}{2}\left(1 - x_{k+1} + z_k\right) \\ z_{k+1} = \dfrac{1}{4}\left(3 - 2x_{k+1} - 2y_{k+1}\right) \end{cases}$$

is easy enough to use and avoids an inversion of the matrix L. It also has a simple interpretation which is nicer from the perspective of programming the method: each equation uses the results of the previous equations. The computer has done the work and is storing the number, why not use it immediately?

If we make an initial guess of $(x_0, y_0, z_0) = (0, 0, 0)$, the first 5 iterations are shown below, making sure to keep only 4 decimal places at each iteration

k	x_k	y_k	z_k
0	0	0	0
1	0.6667	-0.1667	0.5000
2	0.4444	-0.5278	0.7917
3	0.2268	-0.7824	1.0278
4	0.0633	-0.9823	1.2095
5	-0.0639	-1.1367	1.3503

It's not clear whether this converges or not at this stage.

Convergence of matrix iterative schemes

Recall the general equation defining the iterative scheme

$$X^{(k+1)} = M^{-1}\left(B - NX^{(k)}\right) = M^{-1}B - M^{-1}NX^{(k)}.$$

We define a matrix

$$B_i = M^{-1}N$$

as the iteration matrix, so that

$$B_i = \begin{cases} B_J = D^{-1}O & \text{(Jacobi iteration matrix)} \\ B_{GS} = L^{-1}U^* & \text{(Gauss-Seidel iteration matrix)} \end{cases}$$

This iteration matrix can be used to determine whether the scheme will converge or not

THEOREM: Convergence of matrix iteration schemes.
A matrix iteration scheme will converge *if and only if* the maximum of the absolute value of the eigenvalues of B_i is less than 1. That is

$$\rho(B_i) < 1$$

where

$$\rho(B_i) = \max\left\{|\lambda_k| \quad | \quad \lambda_k \text{ eigenvalue of } B_i\right\}.$$