MATH50003 Numerical Analysis

Sheehan Olver

January 12, 2026

Contents

Chapter I

Calculus on a Computer

In this first chapter we explore the basics of mathematical computing and numerical analysis. In particular we investigate the following mathematical problems which can not in general be solved exactly:

1. Integration. General integrals have no closed form expressions. Can we instead use a computer to approximate the values of definite integrals? Numerical integration underpins much of modern scientific computing and simulations of physical systems modelled by partial differential equations.

2. Differentiation. Differentiating a formula as in calculus is usually algorithmic, however, it is often needed to compute derivatives without access to an underlying formula, eg, a function defined only in code. Can we use a computer to approximate derivatives? A very important application is in Machine Learning, where there is a need to compute gradients in training neural networks.

3. Root finding. There is no general formula for finding roots (zeros) of arbitrary functions, or even polynomials that are of degree 5 (quintics) or higher. Can we compute roots of general functions using a computer?

Each chapter is divided into sections that roughly correspond to individual lectures. In this chapter we investigate solving the above computational problems:

1. I.1 Rectangular rule: we review the rectangular rule for integration and deduce the *convergence rate* of the approximation. In the lab/problem sheet we investigate its implementation as well as extensions to the Trapezium rule.

2. I.2 Divided differences: we investigate approximating derivatives by a divided difference and again deduce the convergence rates. In the lab/problem sheet we extend the approach to the central differences formula and computing second derivatives. We also observe a mystery: the approximations may have significant errors in practice, and there is a limit to the accuracy.

3. I.3 Dual numbers: we introduce the algebraic notion of a *dual number* which allows the implementation of *forward-mode automatic differentiation*, a high accuracy alternative to divided differences for computing derivatives.

4. I.4 Newton's method: Newton's method is a basic approach for computing roots/zeros of a function. We use dual numbers to implement this algorithm.

Each week there are labs and problem sheets that further explore the mathematical material introduced in each section. The labs generally explore practical implementation and the impact of implementing methods in computer arithmetic. The problem sheets dig deeper into analysis of other methods and phenomena observed in the labs. The material introduced in the labs and problem sheets is also examinable so it's important to study these as well.

I.1 Rectangular rule

One possible definition for an integral is the limit of a Riemann sum, for example:

$$\int_a^b f(x)\mathrm{d}x = \lim_{n\to\infty} h\sum_{j=1}^{n} f(x_j)$$

where $x_j = a + jh$ are evenly spaced points dividing up the interval $[a, b]$, that is with the *step size* $h = (b-a)/n$. This suggests an algorithm known as the *(right-sided) rectangular rule* for approximating an integral: choose n large so that

$$\int_a^b f(x)\mathrm{d}x \approx h\sum_{j=1}^{n} f(x_j).$$

We will show that the error in approximation is bounded by C/n for some constant C. This can be expressed using "Big-O" notation:

$$\int_a^b f(x)\mathrm{d}x = h\sum_{j=1}^{n} f(x_j) + O(1/n).$$

In these notes we consider the "Analysis" part of "Numerical Analysis": we want to *prove* the convergence rate of the approximation, including finding an explicit expression for the constant C.

To tackle this question we consider the error incurred on a single panel (x_{j-1}, x_j), then sum up the errors on rectangles.

Now for a secret. There are only so many tools available in analysis (especially at this stage of your career), and one can make a safe bet that the right tool in any analysis proof is either (1) integration-by-parts, (2) geometric series or (3) Taylor series. In this case we use (1):

Lemma 1 ((Right-sided) Rectangular Rule error on one panel)**.** *Assuming f is differentiable on $[a, b]$ and its derivative is integrable we have*

$$\int_a^b f(x)\mathrm{d}x = (b-a)f(b) + \delta$$

where $|\delta| \le M(b-a)^2$ for $M = \sup_{a\le x\le b} |f'(x)|$.

Proof We write

$$\begin{aligned}\int_a^b f(x)\mathrm{d}x &= \int_a^b (x-a)'f(x)\mathrm{d}x = [(x-a)f(x)]_a^b - \int_a^b (x-a)f'(x)\mathrm{d}x \\ &= (b-a)f(b) + \underbrace{\left(-\int_a^b (x-a)f'(x)\mathrm{d}x\right)}_{\delta}.\end{aligned}$$

Recall that we can bound the absolute value of an integral by the supremum of the integrand times the width of the integration interval:

$$\left|\int_a^b g(x)\mathrm{d}x\right| \leq (b-a)\sup_{a\leq x\leq b}|g(x)|.$$

The lemma thus follows since

$$\begin{aligned}\left|\int_a^b (x-a)f'(x)\mathrm{d}x\right| &\leq (b-a)\sup_{a\leq x\leq b}|(x-a)f'(x)| \\ &\leq (b-a)\sup_{a\leq x\leq b}|x-a|\sup_{a\leq x\leq b}|f'(x)| \\ &\leq M(b-a)^2.\end{aligned}$$

■

Now summing up the errors in each panel gives us the error of using the Rectangular rule:

Theorem 1 (Rectangular Rule error)**.** *Assuming f is differentiable on $[a,b]$ and its derivative is integrable we have*

$$\int_a^b f(x)\mathrm{d}x = h\sum_{j=1}^n f(x_j) + \delta$$

where $|\delta| \leq M(b-a)h$ for $M = \sup_{a\leq x\leq b}|f'(x)|$, $h=(b-a)/n$ and $x_j = a+jh$.

Proof We split the integral into a sum of smaller integrals:

$$\int_a^b f(x)\mathrm{d}x = \sum_{j=1}^n \int_{x_{j-1}}^{x_j} f(x)\mathrm{d}x = \sum_{j=1}^n [(x_j - x_{j-1})f(x_j) + \delta_j] = h\sum_{j=1}^n f(x_j) + \underbrace{\sum_{j=1}^n \delta_j}_{\delta}$$

where δ_j, the error on each panel as in the preceding lemma, satisfies

$$|\delta_j| \leq (x_j - x_{j-1})^2 \sup_{x_{j-1}\leq x\leq x_j}|f'(x)| \leq Mh^2.$$

Thus using the triangular inequality we have

$$|\delta| = \left|\sum_{j=1}^n \delta_j\right| \leq \sum_{j=1}^n |\delta_j| \leq Mnh^2 = M(b-a)h.$$

■

Note a consequence of this lemma is that the approximation converges as $n\to\infty$ (i.e. $h\to 0$). In the labs and problem sheets we will consider the left-sided rule:

$$\int_a^b f(x)\mathrm{d}x \approx h\sum_{j=0}^{n-1} f(x_j).$$

We also consider the *Trapezium rule*. Here we approximate an integral by an affine function:

$$\int_a^b f(x)\mathrm{d}x \approx \int_a^b \frac{(b-x)f(a)+(x-a)f(b)}{b-a}\mathrm{d}x = \frac{b-a}{2}\left[f(a)+f(b)\right].$$

Subdividing an interval $a = x_0 < x_1 < \ldots < x_n = b$ and applying this approximation separately on each subinterval $[x_{j-1}, x_j]$, where $h=(b-a)/n$ and $x_j = a+jh$, leads to the approximation

$$\int_a^b f(x)\mathrm{d}x \approx \frac{h}{2}f(a) + h\sum_{j=1}^{n-1} f(x_j) + \frac{h}{2}f(b)$$

We shall see both experimentally and provably that this approximation converges faster than the rectangular rule.

I.1.1 Lab and problem sheet

In the lab, we explore the practical implementation of the right-sided rectangular rule and extensions to other rules like the left-sided rectangular rule and trapezium rule. We also see how linear convergence ($O(h) = O(1/n)$) can be deduced *experimentally*: by comparing an implementation of the rule to specific integrals with known formulæ we can compute the error, and determine its rate of decay visually by plotting it. In particular, we deduce that the Trapezium rule converges much faster to the true value of the integral than the other rules. In the problem sheet we explore the *analysis* of these other rules, proving that the Trapezium rule converges to the true integral at a faster quadratic ($O(h^2)$) error rate. This is a guarantee that the integral can be computed much more accurately for the same amount of work by taking into account the analysis, highlighting the important contribution of analysis in the construction of algorithms.

I.2 Divided Differences

Given a function, how can we approximate its derivative at a point? We consider an intuitive approach to this problem using *(Right-sided) Divided Differences*:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

Note by the definition of the derivative we know that this approximation will converge to the true derivative as $h \to 0$. But in numerical approximations we also need to consider the rate of convergence.

Now in the previous section I mentioned there are three basic tools in analysis: (1) integration-by-parts, (2) geometric series or (3) Taylor series. In this case we use (3):

Proposition 1 (divided differences error)**.** *Suppose that f is twice-differentiable on the interval $[x, x+h]$. The error in approximating the derivative using divided differences is*

$$f'(x) = \frac{f(x+h) - f(x)}{h} + \delta$$

where $|\delta| \leq Mh/2$ for $M = \sup_{x \leq t \leq x+h} |f''(t)|$.

Proof Follows immediately from Taylor's theorem: recall that

$$f(x+h) = f(x) + f'(x)h + \frac{f''(t)}{2}h^2$$

for some $t \in [x, x+h]$. Rearranging we get

$$f'(x) = \frac{f(x+h) - f(x)}{2} + \underbrace{\left(-\frac{f''(t)}{2h^2}\right)}_{\delta}.$$

We then bound:

$$|\delta| \leq \left|\frac{f''(t)}{2}h\right| \leq \frac{Mh}{2}.$$

■

Unlike the rectangular rule, the computational cost of computing the divided difference is independent of h! We only need to evaluate a function f twice and do a single division. Here we are assuming that the computational cost of evaluating f is independent of the point of evaluation. Later we will investigate the details of how computers work with numbers via floating point, and confirm that this is a sensible assumption.

In the lab we investigate the convergence rate of these approximations (in particular, that central differences is more accurate than standard divided differences) and observe that they too suffer from unexplained (for now) loss of accuracy as $h \to 0$. In the problem sheet we prove the theoretical convergence rate, which is never realised because of these errors.

I.2.1 Lab and problem sheet

In the labs and problem sheets we explore alternative versions of divided differences. Left-side divided differences evaluates to the left of the point where we wish to know the derivative:

$$f'(x) \approx \frac{f(x) - f(x-h)}{h}$$

and central differences evaluates both left and right:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}.$$

We can further arrive at an approximation to the second derivative by composing a left- and right-sided finite difference:

$$f''(x) \approx \frac{f'(x+h) - f'(x)}{h} \approx \frac{\frac{f(x+h)-f(x)}{h} - \frac{f(x)-f(x-h)}{h}}{h} = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}.$$

The lab explores these approximations *experimentally*, and we will observe that central differences converges much faster to the true value of the derivative as h becomes moderately small.

An important distinction between rectangular rules and divided difference is that the computational cost of divided differences is independent of h: we can choose h arbitrarily and the approximation will take the same amount of time. This raises a question: why not just set h ridiculously small so that the approximation is extremely accurate? Unfortunately, we will observe in the lab a serious issue: if h becomes too small, the error mysteriously starts to grow, and hence these rules do not actually converge to the true value of the derivatives! Thus there is a limitation to how accurate one can approximate a derivative using divided differences, an issue we will overcome in the next section by re-thinking derivatives in an algebraic way.

The problem sheet explores the *analysis* of divided difference rules, proving the precise theoretical convergence rates observed for moderately small h. This presents a bit of a conundrum: why does the theory say the method converges but in practice it diverges, and spectacularly so! This is a mystery that we will return to later, by understanding how computer arithmetic with real numbers works.

I.3 Dual Numbers

In this section we introduce a mathematically beautiful alternative to divided differences for computing derivatives: *dual numbers*. These are a commutative ring that *exactly* compute

derivatives, which when implemented on a computer gives very high-accuracy approximations to derivatives. They underpin forward-mode automatic differentation. Automatic differentiation is a basic tool in Machine Learning for computing gradients necessary for training neural networks.

Definition 1 (Dual numbers)**.** Dual numbers $\mathbb{D}$ are a commutative ring (over $\mathbb{R}$) generated by 1 and ϵ such that $\epsilon^2 = 0$, that is,

$$\mathbb{D} := \{a + b\epsilon \quad : \quad a, b \in \mathbb{R}, \quad \epsilon^2 = 0\}.$$

This is very much analoguous to complex numbers, which are a field generated by 1 and i such that $\mathrm{i}^2 = -1$, that is,

$$\mathbb{C} := \{a + b\mathrm{i} \quad : \quad a, b \in \mathbb{R}, \quad \mathrm{i}^2 = -1\}.$$

Compare multiplication of each number type which falls out of the rules of the generators:

$$\begin{aligned}(a + b\mathrm{i})(c + d\mathrm{i}) &= ac + (bc + ad)\mathrm{i} + bd\mathrm{i}^2 = ac - bd + (bc + ad)\mathrm{i},\\ (a + b\epsilon)(c + d\epsilon) &= ac + (bc + ad)\epsilon + bd\epsilon^2 = ac + (bc + ad)\epsilon.\end{aligned}$$

And just as we view $\mathbb{R} \subset \mathbb{C}$ by equating $a \in \mathbb{R}$ with $a + 0\mathrm{i} \in \mathbb{C}$, we can view $\mathbb{R} \subset \mathbb{D}$ by equating $a \in \mathbb{R}$ with $a + 0\epsilon \in \mathbb{D}$.

Conceptually, dual numbers can be thought of as introducing an infinitesimally small ϵ, where ϵ^2 is so small it is treated as zero. This is the intuitive reason they allow for differentiation of functions. But we do not need to appeal to this calculus-like interpretation, instead, their construction and relationship to differentiation can be accomplished using purely algebraic reasoning.

I.3.1 Differentiating polynomials

Polynomials evaluated on dual numbers are well-defined as they depend only on the operations $+$ and $*$. From the formula for multiplication of dual numbers we deduce that evaluating a polynomial at a dual number $a + b\epsilon$ tells us the derivative of the polynomial at a:

Theorem 2 (polynomials on dual numbers)**.** *Suppose p is a polynomial. Then*

$$p(a + b\epsilon) = p(a) + bp'(a)\epsilon$$

Proof

First consider $p(x) = x^n$ for $n \geq 0$. The cases $n = 0$ and $n = 1$ are immediate. For $n > 1$ we have by induction:

$$(a + b\epsilon)^n = (a + b\epsilon)(a + b\epsilon)^{n-1} = (a + b\epsilon)(a^{n-1} + (n - 1)ba^{n-2}\epsilon) = a^n + bna^{n-1}\epsilon.$$

For a more general polynomial

$$p(x) = \sum_{k=0}^{n} c_k x^k$$

the result follows from linearity:

$$p(a+b\varepsilon) = \sum_{k=0}^{n} c_k(a+b\epsilon)^k = c_0+\sum_{k=1}^{n} c_k(a^k+kba^{k-1}\epsilon) = \sum_{k=0}^{n} c_k a^k+b\sum_{k=1}^{n} c_k k a^{k-1}\epsilon = p(a)+bp'(a)\epsilon.$$

■

Example 1 (differentiating polynomial)**.** Consider computing $p'(2)$ where

$$p(x) = (x-1)(x-2) + x^2.$$

We can use dual numbers to differentiate, avoiding expanding in monomials or applying rules of differentiating:

$$p(2+\epsilon) = (1+\epsilon)\epsilon + (2+\epsilon)^2 = \epsilon + 4 + 4\epsilon = 4 + \underbrace{5}_{p'(2)}\epsilon.$$

I.3.2 Differentiating other functions

We can extend real-valued differentiable functions to dual numbers in a similar manner. First, consider a standard function with a Taylor series (e.g. cos, sin, exp, etc.)

$$f(x) = \sum_{k=0}^{\infty} f_k x^k$$

so that a is inside the radius of convergence. This leads naturally to a definition on dual numbers:

$$\begin{aligned} f(a+b\epsilon) &= \sum_{k=0}^{\infty} f_k (a+b\epsilon)^k = f_0 + \sum_{k=1}^{\infty} f_k (a^k + k a^{k-1} b\epsilon) = \sum_{k=0}^{\infty} f_k a^k + \sum_{k=1}^{\infty} f_k k a^{k-1} b\epsilon \\ &= f(a) + bf'(a)\epsilon. \end{aligned}$$

More generally, given a differentiable function (which may not have a Taylor series) we can extend it to dual numbers:

Definition 2 (dual extension)**.** Suppose a real-valued function $f : \Omega \to \mathbb{R}$ is differentiable in $\Omega \subset \mathbb{R}$. We can construct the *dual extension* $\underline{f} : \Omega + \epsilon\mathbb{R} \to \mathbb{D}$ by defining

$$\underline{f}(a+b\epsilon) := f(a) + bf'(a)\epsilon.$$

By viewing $\mathbb{R} \subset \mathbb{D}$, it is natural to reuse the notation f for the dual extension, hence when there's no chance of confusion we will identify $f(a+b\epsilon) \equiv \underline{f}(a+b\epsilon)$.

Thus, for basic functions we have natural extensions:

$$\begin{aligned} \exp(a+b\epsilon) &:= \exp(a) + b\exp(a)\epsilon && (a,b \in \mathbb{R}) \\ \sin(a+b\epsilon) &:= \sin(a) + b\cos(a)\epsilon && (a,b \in \mathbb{R}) \\ \cos(a+b\epsilon) &:= \cos(a) - b\sin(a)\epsilon && (a,b \in \mathbb{R}) \\ \log(a+b\epsilon) &:= \log(a) + \frac{b}{a}\epsilon && (a \in (0,\infty), b \in \mathbb{R}) \\ \sqrt{a+b\epsilon} &:= \sqrt{a} + \frac{b}{2\sqrt{a}}\epsilon && (a \in (0,\infty), b \in \mathbb{R}) \\ |a+b\epsilon| &:= |a| + b\,\mathrm{sign}a\,\epsilon && (a \in \mathbb{R}\backslash\{0\}, b \in \mathbb{R}) \end{aligned}$$

provided the function is differentiable at a. Note the last example does not have a convergent Taylor series (at 0) but we can still extend it where it is differentiable.

Going further, we can add, multiply, and compose such dual-extensions. And the beauty is these automatically satisfy the right properties to be dual-extensions themselves, thus

allowing for differentiation of complicated functions built from basic differentiable building blocks.

The following lemma shows that addition and multiplication in some sense "commute" with the dual-extension, hence we can recover the product rule from dual number multiplication:

Lemma 2 (addition/multiplication)**.** *Suppose $f, g : \Omega \to \mathbb{R}$ are differentiable for $\Omega \subset \mathbb{R}$ and $c \in \mathbb{R}$. Then for $a \in \Omega$ and $b \in \mathbb{R}$ we have*

$$\begin{aligned} \underline{f+g}(a+b\epsilon) &= \underline{f}(a+b\epsilon) + \underline{g}(a+b\epsilon) \\ \underline{cf}(a+b\epsilon) &= c\underline{f}(a+b\epsilon) \\ \underline{fg}(a+b\epsilon) &= \underline{f}(a+b\epsilon)\underline{g}(a+b\epsilon) \end{aligned}$$

Proof The first two are immediate due to linearity:

$$\begin{aligned} \underline{(f+g)}(a+b\epsilon) &= (f+g)(a) + b(f+g)'(a)\epsilon \\ &= (f(a) + bf'(a)\epsilon) + (g(a) + bg'(a)\epsilon) = \underline{f}(a+b\epsilon) + \underline{g}(a+b\epsilon), \\ \underline{cf}(a+b\epsilon) &= (cf)(a) + b(cf)'(a)\epsilon = c(f(a) + bf'(a)\epsilon) = c\underline{f}(a+b\epsilon). \end{aligned}$$

The last property essentially captures the product rule of differentiation:

$$\begin{aligned} \underline{fg}(a+b\epsilon) &= f(a)g(a) + b(f(a)g'(a) + f'(a)g'(a))\epsilon \\ &= (f(a) + bf'(a)\epsilon)(g(a) + bg'(a)\epsilon) = \underline{f}(a+b\epsilon)\underline{g}(a+b\epsilon). \end{aligned}$$

■

Furthermore composition recovers the chain rule:

Lemma 3 (composition)**.** *Suppose $f : \Gamma \to \mathbb{R}$ and $g : \Omega \to \Gamma$ are differentiable in $\Omega, \Gamma \subset \mathbb{R}$. Then*

$$\underline{(f \circ g)}(a+b\epsilon) = \underline{f}(\underline{g}(a+b\epsilon))$$

Proof Again it falls out of the properties of dual numbers:

$$\underline{(f \circ g)}(a+b\epsilon) = f(g(a)) + bg'(a)f'(g(a))\epsilon = \underline{f}(g(a) + bg'(a)\epsilon) = \underline{f}(\underline{g}(a+b\epsilon))$$

■

A simple corollary is that any function defined in terms of addition, multiplication, composition, etc. of basic functions with dual-extensions will be differentiable via dual numbers. In this following example we see a practical realisation of this, where we differentiate a function by just evaluating it on dual numbers, implicitly, using the dual-extension for the basic build blocks:

Example 2 (differentiating non-polynomial)**.** Consider differentiating $f(x) = \exp(x^2 + \cos x)$ at the point $a = 1$, where we automatically use the dual-extension of exp and cos. We can differentiate f by simply evaluating on the duals:

$$f(1+\epsilon) = \exp(1 + 2\epsilon + \cos 1 - \sin 1\epsilon) = \exp(1 + \cos 1) + \exp(1 + \cos 1)(2 - \sin 1)\epsilon.$$

Therefore we deduce that

$$f'(1) = \exp(1 + \cos 1)(2 - \sin 1).$$

I.3.3 Lab and problem sheet

In the lab we explore how one can turn this mathematical idea into a practical implementation on a computer, giving a basic version of *forward-mode automatic differentiation*. This is a concept that underpins machine learning, which uses *reverse-mode automatic differentiation* to compute gradients when performing stochastic gradient descent. In order to implement dual numbers, we will introduce the concept of a *type*: a data structure with fields. For example, we will implement a type `Rat` for representing rationals p/q, where the type has two fields (`p` and `q`). Basic arithmetic operations like `+` and `*` can be implemented to correctly do rational arithmetic. We will then create a new type that can represent a dual number $a + b\epsilon$, where the the type has two fields (`a` and `b`). By implementing basic arithmetic operations as well as more complicated functions like `exp` we can efficiently, and extremely accurately, compute derivatives of quite general functions.

In the problem sheet, we explore how dual numbers can also be used for pen-and-paper calculations of derivatives. This gives an alternative to traditional differentiation rules like chain and product rule, that while it is mathematically equivalent feels very different in practice. (I prefer it because it is much more algorithmic!) Make sure when doing the problem sheet to only use dual numbers and not fall back to the more traditional rules. We also see that one can extend the concept to a 2D-analogue of dual numbers, which allows for computation of gradients.

I.4 Newton's method

In school you may recall learning Newton's method: a way of approximating zeros/roots to a function by using a local approximation by an affine function. That is, approximate a function $f(x)$ locally around an initial guess x_0 by its first order Taylor series:

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0)$$

and then find the root of the right-hand side which is

$$f(x_0) + f'(x_0)(x - x_0) = 0 \Leftrightarrow x = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

We can then repeat using this root as the new initial guess. In other words we have a sequence of *hopefully* more accurate approximations:

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

Thus *if* we can compute derivatives, we can (sometimes) compute roots.

In terms of analysis, we can guarantee convergence provided our initial guess is accurate enough. The first step is the bound the error of an iteration in terms of the previous error:

Theorem 3 (Newton error)**.** *Suppose f is twice-differentiable in a neighbourhood B of r such that $f(r) = 0$, and f' does not vanish in B. Denote the error of the k-th Newton iteration as $\varepsilon_k := r - x_k$. If $x_k \in B$ then*

$$|\varepsilon_{k+1}| \leq M|\varepsilon_k|^2$$

where

$$M := \frac{1}{2} \sup_{x \in B} |f''(x)| \sup_{x \in B} \left| \frac{1}{f'(x)} \right|.$$

Proof Using Taylor's theorem we find that

$$0 = f(r) = f(x_k + \varepsilon_k) = f(x_k) + f'(x_k)\varepsilon_k + \frac{f''(t)}{2}\varepsilon_k^2.$$

for some $t \in B$ between r and x_k. Rearranging this we get an expression for $f(x_k)$ that tells us that

$$\varepsilon_{k+1} = r - \underbrace{x_{k+1}}_{x_k - f(x_k)/f'(x_k)} = \varepsilon_k + \frac{f(x_k)}{f'(x_k)} = -\frac{f''(t)}{2f'(x_k)}\varepsilon_k^2.$$

Taking absolute values of each side gives the result.

■

This result says that the error decays *quadratically*, which in this case means that the number of digits roughly doubles each iteration. That is, if the error and one step is about 10^{-3} then the error at the next step is about 10^{-6} and the step after about 10^{-12}: this is a drastic improvement! Hidden in this result is a guarantee of convergence provided x_0 is sufficiently close to r.

Corollary 1 (Newton convergence)**.** *If $x_0 \in B$ is sufficiently close to r then $x_k \to r$.*

Proof

Suppose $x_k \in B$ satisfies $|\varepsilon_k| = |r - x_k| \leq M^{-1}$. Then

$$|\varepsilon_{k+1}| \leq M|\varepsilon_k|^2 \leq |\varepsilon_k|,$$

hence $x_{k+1} \in B$. Thus from induction if x_0 satisfies the condition $|\varepsilon_0| < M^{-1}$ condition then $x_k \in B$ for all k and satisfies $|\varepsilon_k| \leq M^{-1}$. Thus we find (for large enough k)

$$|\varepsilon_k| \leq M|\varepsilon_{k-1}|^2 \leq M^3|\varepsilon_{k-2}|^4 \leq M^7|\varepsilon_{k-3}|^8 \leq \ldots \leq M^{2^k-1}|\varepsilon_0|^{2^k} = \frac{1}{M}(M|\varepsilon_0|)^{2^k}.$$

Provided x_0 satisfies the strict inequality $|\varepsilon_0| < M^{-1}$ this will go to zero as $k \to \infty$.

■

I.4.1 Lab and problem sheet

In the lab we explore using Newton's method for some simple root finding problems. We also see that automatic differentiation via dual numbers can be used effectively to compute the derivatives. This is in some sense a baby version of how Machine Learning algorithms train neural networks; but whilst Newton uses derivatives (or in higher-dimensions, gradients) to find roots of functions Machine Learning uses gradients to (very roughly) minimise functions that represent the error between a neural network and training data. Minimisation problems are very closely related to root finding problems (essentially the minima are associated with roots of the gradient) and there are specialised training algorithms in ML built on a randomised version of Newton's method.

In the problem sheet we see how the error bound for Newton iteration can be extended to the degenerate case where the second derivative also vanishes, but now we no longer achieve quadratic convergence, but it still decays exponentially with the number of iterations (which is called *linear convergence*).