MATH50003 Numerical Analysis

Sheehan Olver

January 16, 2025

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 use a computer to approximate the values of definite integrals?

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 to determine the "right" weights in a neural network.

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?

In this chapter we discuss:

1. I.1 Rectangular rule: we review the rectangular rule for integration and deduce the *converge 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 implemention 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.

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

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

In the lab we explore practical implementation of this approximation, and observe 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:

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

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.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 approimxations 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.

So why not just set h ridiculously small? In the lab we explore this question and observe that there are significant errors introduced in the numerical realisation of this algorithm. We will return to the question of understanding these errors after learning floating point numbers.

There are 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:

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

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

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

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.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:

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

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.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. The lab will explore using dual numbers to accomplish this task. This is in some sense a baby version of how Machine Learning algorithms train neural networks; but where Newton uses derivatives (or in higher-dimensions, gradients) to find roots of functions Machine Learning uses gradients to roughly minimise functions that represent the error between a neural network and training data.

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.

■

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 $k \to \infty$.

■

Appendix A

Asymptotics and Computational Cost

We introduce Big-O, little-o and asymptotic notation and see how they can be used to describe computational cost.

A.1 Asymptotics as $n \to \infty$

Big-O, little-o, and "asymptotic to" are used to describe behaviour of functions at infinity.

Definition 3 (Big-O)**.**

$$f(n) = O(\phi(n)) \qquad (\text{as } n \to \infty)$$

means $\left|\frac{f(n)}{\phi(n)}\right|$ is bounded for sufficiently large n. That is, there exist constants C and N_0 such that, for all $n \geq N_0$, $|\frac{f(n)}{\phi(n)}| \leq C$.

Definition 4 (little-O)**.**

$$f(n) = o(\phi(n)) \qquad (\text{as } n \to \infty)$$

means $\lim_{n\to\infty} \frac{f(n)}{\phi(n)} = 0$.

Definition 5 (asymptotic to)**.**

$$f(n) \sim \phi(n) \qquad (\text{as } n \to \infty)$$

means $\lim_{n\to\infty} \frac{f(n)}{\phi(n)} = 1$.

Example 3 (asymptotics with n)**.** 1.

$$\frac{\cos n}{n^2 - 1} = O(n^{-2})$$

as

$$\left|\frac{\frac{\cos n}{n^2-1}}{n^{-2}}\right| \leq \left|\frac{n^2}{n^2-1}\right| \leq 2$$

for $n \geq N_0 = 2$.

2.

$$\log n = o(n)$$

as $\lim_{n\to\infty} \frac{\log n}{n} = 0$.

3.
$$n^2 + 1 \sim n^2$$
as $\frac{n^2+1}{n^2} \to 1$.

Note we sometimes write $f(O(\phi(n)))$ for a function of the form $f(g(n))$ such that $g(n) = O(\phi(n))$.

We have some simple algebraic rules:

Proposition 2 (Big-O rules)**.**
$$O(\phi(n))O(\psi(n)) = O(\phi(n)\psi(n)) \qquad (as\ n \to \infty)$$
$$O(\phi(n)) + O(\psi(n)) = O(|\phi(n)| + |\psi(n)|) \qquad (as\ n \to \infty).$$

Proof See any standard book on asymptotics, eg F.W.J. Olver, Asymptotics and Special Functions. ■

A.2 Asymptotics as $x \to x_0$

We also have Big-O, little-o and "asymptotic to" at a point:

Definition 6 (Big-O)**.**
$$f(x) = O(\phi(x)) \qquad (\text{as } x \to x_0)$$
means $|\frac{f(x)}{\phi(x)}|$ is bounded in a neighbourhood of x_0. That is, there exist constants C and r such that, for all $0 \le |x - x_0| \le r$, $|\frac{f(x)}{\phi(x)}| \le C$.

Definition 7 (little-O)**.**
$$f(x) = o(\phi(x)) \qquad (\text{as } x \to x_0)$$
means $\lim_{x\to x_0} \frac{f(x)}{\phi(x)} = 0$.

Definition 8 (asymptotic to)**.**
$$f(x) \sim \phi(x) \qquad (\text{as } x \to x_0)$$
means $\lim_{x\to x_0} \frac{f(x)}{\phi(x)} = 1$.

Example 4 (asymptotics with x)**.**
$$\exp x = 1 + x + O(x^2) \qquad \text{as } x \to 0$$
since $\exp x = 1 + x + \frac{\exp t}{2}x^2$ for some $t \in [0, x]$ and
$$\left|\frac{\frac{\exp t}{2}x^2}{x^2}\right| \le \frac{3}{2}$$
provided $x \le 1$.

A.3 Computational cost

We will use Big-O notation to describe the computational cost of algorithms. Consider the following simple sum

$$\sum_{k=1}^{n} x_k^2$$

which we might implement as:

```
function sumsq(x)
    n = length(x)
    ret = 0.0
    for k = 1:n
        ret = ret + x[k]^2
    end
    ret
end
```

```
sumsq (generic function with 1 method)
```

Each step of this algorithm consists of one memory look-up (`z = x[k]`), one multiplication (`w = z*z`) and one addition (`ret = ret + w`). We will ignore the memory look-up in the following discussion. The number of CPU operations per step is therefore 2 (the addition and multiplication). Thus the total number of CPU operations is $2n$. But the constant 2 here is misleading: we didn't count the memory look-up, thus it is more sensible to just talk about the asymptotic complexity, that is, the *computational cost* is $O(n)$.

Now consider a double sum like:

$$\sum_{k=1}^{n} \sum_{j=1}^{k} x_j^2$$

which we might implement as:

```
function sumsq2(x)
    n = length(x)
    ret = 0.0
    for k = 1:n
        for j = 1:k
            ret = ret + x[j]^2
        end
    end
    ret
end
```

```
sumsq2 (generic function with 1 method)
```

Now the inner loop is $O(1)$ operations (we don't try to count the precise number), which we do k times for $O(k)$ operations as $k \to \infty$. The outer loop therefore takes

$$\sum_{k=1}^{n} O(k) = O\left(\sum_{k=1}^{n} k\right) = O\left(\frac{n(n+1)}{2}\right) = O(n^2)$$

operations.