Chapter 2

Root Finding

2.1 Introduction

In this chapter the following topics will be discussed

1. Review some topics from Real Analysis
2. The Bisection method.
3. Newton's method.
4. The Secant Method
5. Fixed Points

Definition 2.1.1. *Throughout this course we will use the notation*

$$f(x) \in C([a,b]), \tag{2.1}$$

to mean that $f(x)$ is a continuous real-valued function, defined on the interval $[a,b]$. Similarly the notation

$$f(x) \in C^1((a,b)), \tag{2.2}$$

means that $f(x)$ has a continuous first derivative on the open interval (a,b).

For brevity, we we usually just say that $f(x)$ is continuous on $[a,b]$ and differentiable on (a,b).

Definition 2.1.2. *Suppose that $f(x) \in C([a,b])$. Suppose that there exists a number $\xi \in [a,b]$ such that*

$$f(\xi) = 0,$$

we then say that ξ is a root of $f(x)$.

Finding ξ is not trivial in general, even for simple-looking functions. For example, consider the relatively simple looking equation

$$f(x) = x^5 - 4x - 2.$$

It can be shown that the problem $f(x) = 0$ has no solutions that can be expressed in terms of radicals. In fact there is no general formula for the solution of an arbitrary real valued polynomial, which further implies that there is no general solution to the equation $f(x) = 0$ for an arbitrary continuous real-valued function $f(x)$. The questions that we must ask ourselves are

1. How can we decide whether such an equation possesses a solution within the set of real numbers, and
2. If a solution does exist, how can it be found?

Root finding is equivalent to finding the solution to any equation in one variable. For example, suppose that we wish to find $x \in [a,b]$ such that $h(x) = g(x)$ for $g, h \in C([a,b])$, then this is equivalent to finding a root of $f(x) = 0$, where $f(x) = h(x) - g(x) \in C([a,b])$.

The purpose of this chapter is to answer these questions. The aim is to develop simple numerical techniques that allow an approximate solution of the equation $f(x) = 0$ to be found, where $f(x)$ is a real valued function that is defined and continuous on a closed bounded interval on the real axis. The methods discussed in this chapter are iterative in nature, and when implemented they produce sequences of numbers that, under

favourable conditions, converge to the required solution.

2.2 Bisection method

Before we go into detail regarding the detail of these methods, we first remind ourselves a some result from Real Analysis.

Theorem 2.2.1 (Intermediate Value Theorem)**.** *Suppose that* $f(x) \in C([a,b])$*. Then* $f(x)$ *is bounded on* $[a,b]$ *and if* y *is any number such that*

$$\min_{x\in[a,b]} f(x) \leq y \leq \max_{y\in[a,b]} f(x)$$

then there exists $\xi \in [a,b]$ *such that* $f(\xi) = y$*.*

The next theorem presented here is one from which we will derive our first numerical method:

Theorem 2.2.2. *Let* $f(x)$ *be a continuous real-valued function on* $[a,b]$*. Further assume that* $f(a)f(b) \leq 0$*; Then, there exists a number* $\xi \in [a,b]$ *such that* $f(\xi) = 0$*.*

Proof. If $f(a) = 0$ or $f(b) = 0$ then clearly $\xi = a$ or $\xi = b$ respectively, and the proof is complete.

Now suppose that $f(a)f(b) \neq 0$, then $f(a)f(b) < 0$; Then either

1. $f(a) < 0$ and $f(b) > 0$, or
2. $f(a) > 0$ and $f(b) < 0$.

Assuming either of the above, this means that zero lies in the open interval whose endpoints are $f(a)$ and $f(b)$. By the Intermediate Value Theorem (Theorem 2.2.1) there exists a number ξ in the open interval (a,b) such that $f(\xi) = 0$. □

This result leads to our first algorithm, and this method is known is the bisection method.

Suppose that $f(x)$ satisfies the conditions laid out in Theorem 2.2.2. Then we may define the bisection root-finding algorithm as follows:

1. Let $i = 0$ and define $a_0 = a$, $b_0 = b$
2. Let $\xi_i = \frac{1}{2}(a_i + b_i)$
3. If $b_i - a_i < TOL$ then $\xi = \xi_i$, STOP.
4. Else let

$$(a_{i+1}, b_{i+1}) = \begin{cases} (a_i, \xi_i) & \text{if } f(\xi_i)f(a_i) > 0 \\ (\xi_i, b_i) & \text{if } f(\xi_i)f(b_i) < 0 \end{cases}$$

5. $i = i + 1$. go to (2);

The conditions laid out in step 3 are known as stopping criterion, which tells the algorithm to stop under certain conditions. On implementation a tolerance parameter TOL would be defined, and the stopping condition $(b_i - a_i < TOL)$ indicates that the length of the *interval* in which the root is contained is to within an acceptable value. A typical value of TOL might be 10^{-4} or 10^{-6}, for example. Smaller values of the tolerance parameter mean that more iterations would need to be performed in order for the stopping criteria to be satisfied.

The bisection method may seem a crude method, but it has some important advantages. One particularly important propert is that it always converges to a solution.

Question 2.2.1. Show that the Bisection Method is globally convergent.

Solution 2.2.1. At each iteration the interval

$$I_k = [a_k, b_k],$$

is divided into two halves. Denote $|I_k|$ as being the length of the interval $[a_k, b_k]$ for $k \geq 0$, and therefore $I_0 = [a, b]$ and $|I_0| = b - a$. Note too that

$$|I_k| = \frac{|I_{k-1}|}{2} = \frac{|I_{k-2}|}{2^2} = \cdots = \frac{|I_0|}{2^k} = \frac{b-a}{2^k}. \tag{2.3}$$

Now since the root ξ lies in I_k for all k, i.e.

$$\xi \in I_k \quad \forall k \geq 1,$$

this means that the absolute error on the kth iteration ϵ_k, defined as

$$\epsilon_k = |\xi_k - \xi|$$

is bounded by the length of the interval, i.e.

$$0 \leq \epsilon_k \leq |I_k|.$$

and due to 2.3 we have

$$0 \leq \epsilon_k \leq |I_k| = \frac{b-a}{2^k}$$

and since $2^{-k} \to 0$ as $k \to \infty$ we must conclude that $\epsilon_k \to 0$. Hence

$$\lim_{k \to \infty} \xi_k = \xi,$$

which proves the convergence of the method. □

As well as guaranteed convergence, another advantage with the bisection method is the relatively simple conditions that f is simply continuous and $f(a)f(b) < 0$. No further

conditions required, such as he differentiability of $f(x)$ or even bounds on the derivative. Once an interval $[a, b]$ is found, it is possible to guarantee convergence to a solution, and that after k iterations the solution ξ will lie in an interval of length $\frac{1}{2^k}(b-a)$. So in actual fact the solution is very robust. However one drawback is that the rate of convergence is significantly slower when compared other methods.

Question 2.2.2 (The Bisection Method)**.** Show that the function

$$f(x) = x^3 + 4x^2 - 10,$$

has a root ξ in the closed interval $[1, 2]$, and use the bisection method to find the value of ξ correct to 5 decimal places

Solution 2.2.2. A Matlab code implementation of the solution can be found in section 2.7.3. The code will be demonstrated in the lecture, and you should ensure that you understand the code and are able to write similar code in a language of your choosing.

A subset of the results given from running the algorithm are shown in table 2.2.2. From these results we note a few interesting points:

- The true value of the root is $\xi = 1.365230013$ to nine decimal places.
- The algorithm achieves the solution correct to 5 decimal places in 17 iterations, which is indeed rather slow. In fact the solution given by the 17th iteration is $\xi_{17} = 1.3652267456$
- Note that ξ_9 is actually a better approximation to the root than ξ_{13}. So a good opportunity to use a good intermediate solution was missed.

i	a_i	b_i	ξ_i	$f(\xi_i)$
1	1.0	2.0	1.5	2.37500e+00
2	1.0	1.5	1.25	-1.79688e+00
3	1.25	1.5	1.375	1.62109e-01
4	1.25	1.375	1.3125	-8.48389e-01
5	1.3125	1.375	1.34375	-3.50983e-01
6	1.34375	1.375	1.359375	-9.64088e-02
7	1.359375	1.375	1.3671875	3.23558e-02
8	1.359375	1.3671875	1.36328125	-3.21500e-02
9	1.36328125	1.3671875	1.365234375	7.20248e-05
10	1.36328125	1.3652343750	1.3642578125	-1.60467e-02
11	1.3642578125	1.3652343750	1.3647460938	-7.98926e-03
12	1.3647460938	1.3652343750	1.3649902344	-3.95910e-03
13	1.3649902344	1.3652343750	1.3651123047	-1.94366e-03

Table 2.1: Bisection Method Applied to 2.2.2

The approximations, ξ_i are known as *iterates*. As numerical analysts, when we look at a numerical scheme we are essentially interested in two questions. These are

1. Does the solution converge to the true root? i.e.

$$\lim_{i\to\infty}(\xi_i) = \xi$$

2. And secondly, how *fast* does it converge if indeed it does converge?

In the case of the Bisection Method, the convergence analysis is quite straightforward. In terms of how fast it converges, it is quite straightforward to show that the Bisection Method converges *at least linearly*. A rigorous definition will be given in the next section.

2.3 Some Definitions

We have loosely discussed some notions of convergence and rate of convergence. We now aim to make these definitions more rigorous.

Definition 2.3.1. *Suppose that we have a sequence ξ_k such that*

$$\lim_{k\to\infty} \xi_k = \xi.$$

We say that the sequence (ξ_k) converges to ξ ***at least linearly*** *if there exists a null sequence (i.e. converging to zero) of positive real numbers ϵ_k , and $\lambda \in (0,1)$ such that*

$$|\xi_k - \xi| \leq \epsilon_k, \quad k = 0, 1, 2, \ldots \qquad \textit{and} \qquad \lim_{k\to\infty} \frac{\epsilon_{k+1}}{\epsilon_k} = \lambda.$$

1. If definition 2.3.1 holds with $\lambda = 0$, then the convergence is faster than linear convergence and we say that the sequence (ξ_k) converges to ξ **superlinearly**.
2. If definition 2.3.1 holds with $\lambda \in (0,1)$ and $|\xi_k - \xi| = \epsilon_k$ (note the equality here), we say that the sequence (ξ_k) converges to ξ **linearly**.
3. If definition 2.3.1 holds with $\lambda = 1$ and $|\xi_k - \xi| = \epsilon_k$ (note the equality here), the convergence is slower than linear convergence and we say that the sequence (ξ_k) converges to ξ **sublinearly**.

The words 'at least' here refer to the fact that we only have the inequality $|\xi_k - \xi| \leq \epsilon_k$, which in many cases is all that may be ascertained, as is the case with the bisection method as we'll see.

Some texts call λ the *asymptotic error constant*. In the linear case, other texts identify $\rho = -\log_{10} \lambda$ as the *asymptotic rate of convergence*. This is the number of iterations required to achieve one more significant figure of accuracy.

Example 2.3.1. *Show that the Bisection Method converges at least linearly*

Solution 2.3.1. It was noted in question 2.2.1 that one such sequence that bounds $|\xi_k - \xi|$ is the **maximum value of the absolute error** ϵ_k, which it itself is bounded via

$$0 < |\xi_k - \xi| \leq \epsilon_k \leq \frac{b-a}{2^k}.$$

Therefore we have

$$\frac{\epsilon_{k+1}}{\epsilon_k} \leq \frac{b-a}{2^{k+1}} \times \frac{2^k}{b-a} = \frac{1}{2} \to \frac{1}{2} \quad \text{as} \quad k \to \infty,$$

and since $1/2 \in (0,1)$ and the above *inequality* is satisfied, we can conclude that the Bisection Method converges **at least linearly**.

Definition 2.3.2. *Suppose that*

$$\lim_{k\to\infty} \xi_k = \xi.$$

We say that the sequence (ξ_k) *converges to* ξ ***with at least order*** $q > 1$ *if there exists a sequence* (ϵ_k) *of positive real numbers converging to zero, and* $\lambda > 0$ *such that*

$$|\xi_k - \xi| \leq \epsilon_k, \qquad k = 0, 1, 2, \ldots \quad \text{and} \quad \lim_{k\to\infty} \frac{\epsilon_{k+1}}{\epsilon_k^q} = \lambda.$$

1. If definition 2.3.2 holds with $|\xi_k - \xi| = \epsilon_k$, then the sequence (ξ_k) converges to ξ **with order** q.

2. In particular, if $q = 2$ and $|\xi_k - \xi| = \epsilon_k$ we say that (ξ_k) converges to ξ **quadratically.**

Notice that unlike linear convergence, all we require for λ is $\lambda > 0$. The reason for this is because if $q > 1$ then 2.3.2 implies suitably rapid convergence of the sequence (ξ_k), irrespective of the size of λ.

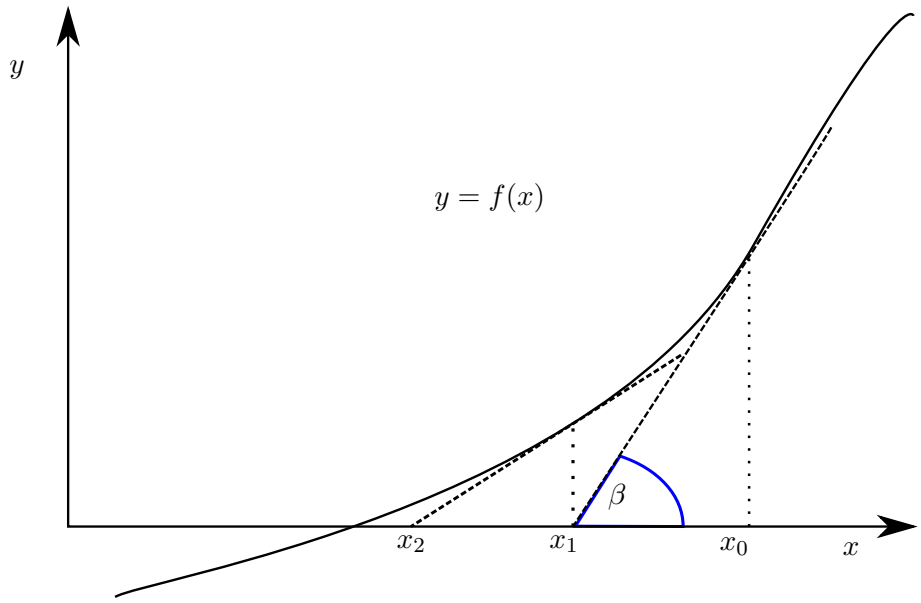

Figure 2.1: Diagram Illustrating Newton's Method

2.4 Newton's Method

We begin from an intuitive understanding of Newton's method. Consider figure 2.4. Suppose we wish to seek the solution to $f(x) = 0$, where $f(x)$ is a real, continuous and differentiable function. Suppose that we have an initial guess to the root which we call x_0. Then let x_1 (our improved solution) be the intersection between the tangent to the curve $y = f(x)$ at x_0 and the x axis. We have

$$\tan\beta = f'(x_0) = \frac{f(x_0)}{x_0 - x_1},$$

i.e.

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

If we were to repeat the process again we would get

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)},$$

which implies the iterative procedure

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}. \tag{2.4}$$

Definition 2.4.1. *Newton's method for the solution of $f(x) = 0$ is defined as*

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \tag{2.5}$$

for some given x_0. We assume that $f'(x_k) \neq 0$ for all $k \geq 0$.

We now wish to show that Newton's method converges quadratically under certain conditions. For that, we need to use Taylor's theorem. Throughout this discussion we shall use the notation

$$f^{(n)}(x) = \frac{d^n}{dx^n} f(x).$$

We first remind ourselves of Taylor's theorem.

Theorem 2.4.1 (Taylor's Theorem)**.** *Suppose that n is a non-negative integer and that $f \in C^n[a,b]$, meaning that f is a real-valued function, and f and all its derivatives up to $f^{(n)}$ are defined and continuous on the interval $[a,b]$. Suppose additionally that $f \in C^{n+1}(a,b)$, meaning that $f^{(n)}$ is differentiable on the open interval (a,b). Now choose $c \in [a,b]$. Then, for each $x \in [a,b]$, there exists $\xi = \xi(x)$ in the interval (a,b) such that*

$$f(x) = f(c) + (x-c)f'(c) + \cdots + \frac{(x-c)^n}{n!} f^{(n)}(c) + \frac{(x-c)^{n+1}}{(n+1)!} f^{(n+1)}(\xi).$$

Proof. The proof of Taylor's theorem, although a vitally important mathematical proof, is outside the scope of this course. □

Next comes a very important result, which is the proof of the quadratic convergence of Newton's method.

Theorem 2.4.2 (Convergence of Newton's Method)**.** *Suppose that f has a continuous second derivative on the closed interval $I_\delta = [\xi - \delta, \xi + \delta], \delta > 0$ such that $f(\xi) = 0$ and*

$f''(\xi) \neq 0$. *Suppose that there exists a real constant,* A*, such that*

$$\frac{|f''(x)|}{|f'(y)|} \leq A \qquad \forall x, y \in I_\delta. \tag{2.6}$$

If $|\xi - x_0| \leq h$*, where* $h = \min\{\delta, 1/A\}$ *then the sequence* (x_i) *defined by Newton's method converges quadratically to* ξ*.*

Proof. Suppose that $|\xi - x_k| \leq h = \min\{\delta, 1/A\}$ so that $x_k \in I_\delta$. Then from Taylor's Theorem (Theorem 2.4.1), expanding about the point x_k we have

$$0 = f(\xi) = f(x_k) + (\xi - x_k)f'(x_k) + \frac{1}{2}(\xi - x_k)^2 f''(\eta_k), \tag{2.7}$$

where η_k is some value contained within the interval $|\xi - x_k|$ and therefore in the interval I_δ.

Now using the formula for Newton's method given by equation (2.4) we can write the $f(x_k)$ term as

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

and on substitution to the above we get

$$\xi - x_{k+1} = -\frac{1}{2}(\xi - x_k)^2 \frac{f''(\eta_k)}{f'(x_k)}. \tag{2.8}$$

We now establish bounds on the above:

$$\begin{aligned} |\xi - x_{k+1}| &= \left| -\frac{1}{2}(\xi - x_k)^2 \frac{f''(\eta_k)}{f'(x_k)} \right| \\ &= \frac{1}{2}|(\xi - x_k)||(\xi - x_k)| \left| \frac{f''(\eta_k)}{f'(x_k)} \right| \\ &\leq \frac{1}{2}|(\xi - x_k)|\frac{1}{A} \times A \\ &= \frac{1}{2}|(\xi - x_k)|, \end{aligned}$$

where we have used both the assumption on the bounds of the derivative quotient given

by equation (2.6), and the fact that $|\xi - x_k| \leq 1/A$ by assumption. This has now led us to

$$|\xi - x_{k+1}| \leq \frac{1}{2}|\xi - x_k|.$$

Now since we assumed that $|\xi - x_0| \leq h$, by induction we have

$$|\xi - x_{k+1}| \leq 2^{-k}h$$

for all k. Since $2^{-k} \to 0$ as $k \to \infty$ this means that

$$|\xi - x_k| \to 0 \quad \text{as} \quad k \to \infty$$

thus proving that the sequence (x_k) converges to ξ.

Now, η_k lies in the interval $|\xi - x_k|$, and therefore (η_k) converges to ξ as $k \to \infty$. Since f' and f'' have been assumed to be continuous on I_δ then we must have

$$\lim_{k\to\infty} \left(\frac{f''(\eta_k)}{f'(x_k)} \right) = \left(\frac{f''(\xi)}{f'(\xi)} \right),$$

and therefore from (2.8) and the above we get

$$\lim_{k\to\infty} \frac{|x_{k+1} - \xi|}{|x_k - \xi|^2} = \left| \frac{f''(\xi)}{2f'(\xi)} \right|,$$

and this, according to Definition 2.3.2, implies that the sequence (x_k) generated by Newton's method converges quadratically to ξ with

$$\lambda = \left| \frac{f''(\xi)}{2f'(\xi)} \right| \in (0, A/2].$$

□

The theorem presented here requires that $f'(x) \neq 0$ for all $x \in I_\delta$, because otherwise

the quantity A is unbounded in the neighbourhood of ξ. Other convergence results, on larger intervals, are possible, although with different assumptions about the behaviour of the derivatives, e.g. if f' and f'' are positive on the interval under consideration, then Newton's method converges quadratically for any starting value, x_0 in the interval. The proof of this is similar to that of Theorem 2.4.2.

When $f'(\xi) = 0$ (i.e. when f has a double root), the convergence is linear. Proof of this is left as an exercise. Contrarily, if $f'(\xi) \neq 0$ but $f''(\xi) = 0$, then it's possible to show that the convergence is *cubic*.

Theorem 2.4.2 shows that if x_0 is close enough to ξ, then the Newton iteration will converge to it. However, when multiple roots are present and when x_0 is not close to any of them, the root eventually found by Newton's method is not necessarily the closest to x_0.

2.4.1 A Demonstration of Newton's Method

We now progress to using Newton's method to solve a practical problem using Newton's method

Question 2.4.1 (Newton's Method)**.** Show that the function

$$f(x) = x^3 + 4x^2 - 10,$$

has a root ξ in the closed interval $[1, 2]$, and use Newton's method to find the value of ξ correct to 9 decimal places. Compare the result with those from the Bisection method in Question 2.2.2.

Solution 2.4.1. A Matlab code implementation of the solution can be found in section 2.7.2. The code will be demonstrated in the lecture, and you should ensure that you understand the code and how to get it to run.

i	ξ_i	$f(\xi_i)$
1	1.50	2.37500e+00
2	1.3733333333	1.34345e-01
3	1.3652620149	5.28461e-04
4	1.3652300139	8.29055e-09
5	1.3652300134	0.00000e+00

Table 2.2: Newton's Method Applied to 2.4.1

The results given from running the algorithm are shown in table 2.4.1. From these results we note a few interesting points:

- The starting value was $x_0 = 1.5$, though any value sufficiently close to the root will do. A dangerous starting value would be one that is close to a turning point!
- The procedure took only five iterations in total to achieve an accuracy of nine decimal places. This is remarkably quick compared to the Bisection method, which would need 34 iterations to achieve the same level of accuracy (try this for yourself).

2.5 Secant method

Newton's method requires us to calculate derivatives of f. Practically, this is often quite difficult. One solution to this problem is to approximate the derivative using a finite difference quotient, i.e. If two points x_k and x_{k+1} are sufficiently close, then $f'(x_k)$ may be approximated by

$$f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}. \tag{2.9}$$

Replacing the $f'(x_k)$ term in 2.4 with the finite-difference expression above yields a new method where no derivatives need be calculated. This method is defined below

Definition 2.5.1 (The Secant Method)**.** *The Secant Method is defined by the iterative procedure*

$$x_{k+1} = x_k - f(x_k)\left(\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}\right), \tag{2.10}$$

where x_0 and x_1 are two initial values that have to be given. It is implicitly assumed that

i	ξ_i	$f(\xi_i)$
1	1.2631578947	-1.60227e+00
2	1.3388278388	-4.30365e-01
3	1.3666163947	2.29094e-02
4	1.3652119026	-2.99068e-04
5	1.3652300011	-2.03168e-07
6	1.3652300134	1.80478e-12
7	1.3652300134	0.00000e+00

Table 2.3: Secant Method Applied to Question 2.5.1

$f(x_k) - f(x_{k-1}) \neq 0$ *for all* $k \geq 0$.

If $f'(\xi) \neq 0$. The secant method converges at least linearly. In fact, it's possible to show that the order of convergence is $(1 + \sqrt{5})/2$ - the *golden ratio*.

Comparing Newton's method with the secant method, we see that Newton's method converges faster (order 2 against $\alpha \approx 1.6$). However, Newton's method requires the evaluation of both f and f' at each iteration step. The secant method on the other hand only requires the evaluation of f. Therefore, occasionally the Secant Method is faster in practice.

2.5.1 Practical Example of Secant Method

Question 2.5.1 (Secant Method)**.** Use the Secant method to find a root for the function $f(x)$

$$f(x) = x^3 + 4x^2 - 10,$$

where the root ξ lies in the closed interval $[1, 2]$, and use the method to find the value of ξ correct to 9 decimal places. Compare the result with those from the Bisection method in Question 2.2.2 and Newton's method in 2.4.1.

Solution 2.5.1. A Matlab code implementation of the solution can be found in section 2.7.3. The code will be demonstrated in the lecture.

The results given from running the algorithm are shown in table 2.5.1. From these

results we note a few interesting points:

- The starting values were chosen as $x_0 = 1.5, x_1 = 2$. It is not necessary for the root to be bracketed in between x_0 and x_1.
- The procedure took only seven iterations in total to achieve an accuracy of nine decimal places, not many more than Newton's method.

2.6 Fixed points

The equation $f(x) = 0$ can always be rewritten as $g(x) = x$ for some function, $g(x)$. For example we can always write $g(x) = x + f(x)$, although this of course may not be the only way.

Any ξ that satisfies $g(\xi) = \xi$ is called a *fixed point* of g.

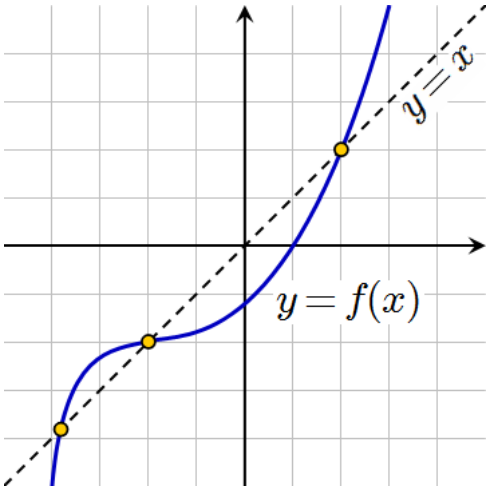

Figure 2.2: A function with three fixed points

It is clear that an alternative to solving $f(x) = 0$ would be to solve $x = g(x)$, or in other words to find some iterative process that converges to a fixed point of $g(x)$. In order to find fixed points of functions, we first need to establish some conditions on which a fixed point will exists for a function. The next theorem describes the situation where a fixed point does indeed exist, and provides a proof of the existence.

Theorem 2.6.1 (Brouwer's Fixed Point Theorem)**.** *Suppose that $g(x)$ is a continuous function defined on the closed interval $[a, b]$. Now let $g(x) \in [a, b]$ for all $x \in [a, b]$. Then There exists some real number ξ such that $g(\xi) = \xi$, i.e. ξ is a fixed point of the function g.*

Before we dive into the proof figure 2.6 gives you an idea of the type of function we're

dealing with.

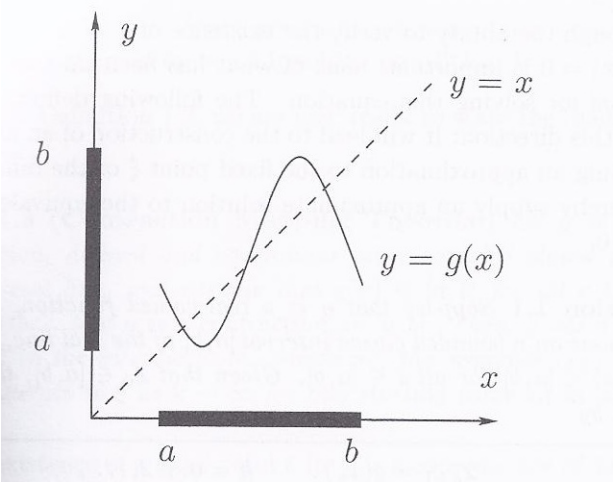

Figure 2.3: A sketch of the type of function described in theorem 2.6.1 (*Image taken from Suli, An Introduction to Numerical Analysis*). This function has three fixed points in $[a, b]$, characterised by the intersection with the line $y = x$.

Proof. Let $f(x) = x - g(x)$. Then $f(a) = a - g(a) \leq 0$ since $g(a) \in [a, b]$. Similarly $f(b) = b - g(b) \geq 0$ since $g(b) \in [a, b]$. Consequently $f(a)f(b) \leq 0$, and by theorem 2.2.2 this means that there exists a value $\xi \in [a, b]$ such that $f(\xi) = 0$. Hence $\xi - g(\xi) = 0$ or $g(\xi) = \xi$. □

The next question that we must ask is, how do we find fixed points? One approach is (of course) to iterate: For this, the following lemma is useful

Lemma 2.6.2. *Let $g : \mathbb{R} \to \mathbb{R}$ be a function that is continuous on a section of the real line $[a, b]$. Now choose $x_0 \in \mathbb{R}$ and define a sequence*

$$x_{n+1} = g(x_n). \tag{2.11}$$

If the sequence (x_n) converges to some limit $\xi = \lim_{n\to\infty} x_n$ and g is continuous in a neighbourhood of ξ then $g(\xi) = \xi$.

Proof. The proof of this lemma is straightforward. We have assumed that ξ is the limit

of the sequence (x_n), and so we have

$$\xi = \lim_{n\to\infty} x_{n+1} = \lim_{n\to\infty} g(x_n) = g\left(\lim_{n\to\infty} x_n\right) = g(\xi)$$

where the second inequality comes from equation 2.11 and the third inequality is a consequence of the continuity of g. [1] □

Example 2.6.3. *Lets use this to try and find a root of the function*

$$f(x) = e^x - 2x - 1.$$

Now since $f(1) < 0$ *and* $f(2) > 0$ *By Theorem 2.2.2 this means that a root must exist in* $[1, 2]$, *i.e.*

$$\exists \quad \xi \in (a, b) \quad \textit{such that} \quad f(\xi) = 0.$$

Now, we are attempting to find a solution to $f(x) = 0$, *we can transform this into an equivalent fixed point problem by writing* $f(x) = 0$ *in the form* $x = g(x)$ *as*

$$\begin{aligned} & x = \ln(2x + 1), \\ \textit{or} \quad & x = \frac{e^x - 1}{2}. \end{aligned}$$

This implies two separate fixed-point type iterative processes:

$$x_{n+1} = \ln(2x_n + 1), \tag{2.12}$$

$$x_{n+1} = \frac{e^{x_n} - 1}{2}. \tag{2.13}$$

Since we know that the root lies in the interval $[1, 2]$ *let us take* $x_0 = 1.5$ *and see what happens. The results are shown in table 2.6.3.*

So the process defined by 2.12 clearly converges to the required fixed point, but the process defined by 2.13 rapidly diverges. Clearly, from this we can conclude that simply rewriting

[1] i.e. if $g(x)$ is continuous at $x = c$ then by definition $\lim_{x\to c} g(x) = g(c)$

n	$x_{n+1} = \ln(2x_n + 1)$	$x_{n+1} = (e^{x_n} - 1)/2$
0	1.5	1.5
1	1.38629436	1.74084454
2	1.32776143	2.35107853
3	1.29623914	4.74844242
4	1.27884229	57.20219636
5	1.26910993	34799117997714600000000000.
6	1.26362374	#NUM!

Table 2.4: Iterative Method Applied to Question 2.6.3. These results were obtained using Microsoft Excel

$f(x) = 0$ as $x = g(x)$ and iterating is not enough to guarantee a solution. We need some additional criteria on the function $g(x)$ to ensure convergence of the sequence to the required fixed point.

A sufficient condition for the convergence of the sequence (x_n) is provided via our next result, which is a modification of the Brouwer theorem suggested earlier. This results essentially suggests that the sequence (x_n) will converge to the required fixed point provided that the mapping g is a contraction.

Definition 2.6.1 (Contraction)**.** *Suppose that $g \in C([a, b])$. Then g is a contraction on $[a, b]$ if*

$$|g(x) - g(y)| \leq L\,|x - y| \quad \forall x, y \in [a, b], \tag{2.14}$$

for some constant, L, satisfying $0 < L < 1$.

The term contraction stems from the fact that if equation 2.14 holds then the distance $|g(x) - g(y)|$ between the image points $g(x)$ and $g(y)$ is at least $1/L$ times smaller than the distance $|x - y|$ between the points x and y. If 2.14 holds for L being *any* positive real number, then this is usually referred to as a Lipschitz condition.

With the above definition we are now ready to discuss the main result of this section on fixed points.

Theorem 2.6.4 (Contraction Mapping Theorem)**.** *Let $g \in C([a, b])$ and assume that g*

is a contraction and that $g : [a, b] \to [a, b]$ *for all* $x \in [a, b]$. *Then* g *has a unique fixed point,* $\xi \in [a, b]$. *Moreover, the sequence,* $x_{n+1} = g(x_n)$ *converges to* ξ *for any starting* $x_0 \in [a, b]$

Proof. For the existence of the fixed point, we simply note that this is a straightforward application of the Brouwer fixed point theorem.

For uniqueness, we prove this by contradiction using the fact that the mapping is assumed to be a contraction. Suppose that there exists a second fixed point $\eta \in [a, b]$. Then

$$|\xi - \eta| = |g(\xi) - g(\eta)| \leq L\,|\xi - \eta|\,,$$

i.e.

$$|\xi - \eta| \leq L\,|\xi - \eta| \quad \Longleftrightarrow \quad (1 - L)|\xi - \eta| \leq 0$$

but since $1 - L > 0$ we must have $|\xi - \eta| = 0$ or $\xi = \eta$. Hence uniqueness of the fixed point is proved.

For convergence, we use induction and again use the fact that the mapping is a contraction. Let x_0 be any value within $[a, b]$, and we shall prove that the sequence $x_{n+1} = g(x_n)$ converges to ξ. According to our definitions we have

$$|x_k - \xi| = |g(x_{k-1}) - g(\xi)| \leq L\,|x_{k-1} - \xi| \quad \text{for} \quad k \geq 1,$$

where the first equality is due to the definition of our iteration, and the inequality is by the definition of a contraction, i.e.

$$|x_k - \xi| \leq L\,|x_{k-1} - \xi| \quad \text{for} \quad k \geq 1.$$

We may therefore deduce by induction that

$$|x_k - \xi| \leq L^k |x_0 - \xi| \quad \text{for} \quad k \geq 1.$$

Since $0 < L < 1$ it follows that $L^k \to 0$ as $k \to \infty$ and therefore $|x_k - \xi| \to 0$ as $k \to \infty$, indicating that the sequence converges to the required fixed point. □

We are now in a position to explain why the iterative procedure 2.12 outlined in question 2.6.3 will converge for any starting value contained within the interval $[1, 2]$. However first we recall a result from Real Analysis, namely the *Mean Value Theorem*.

Theorem 2.6.5 (Mean Value Theorem)**.** *Suppose that $f(x)$ is continuous on $[a, b]$ and differentiable on (a, b). Then, there exists $c \in (a, b)$ such that*

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

.

Proof. This is an elementary result from real analysis, and its proof is not part of this course. This result will just be assumed here. □

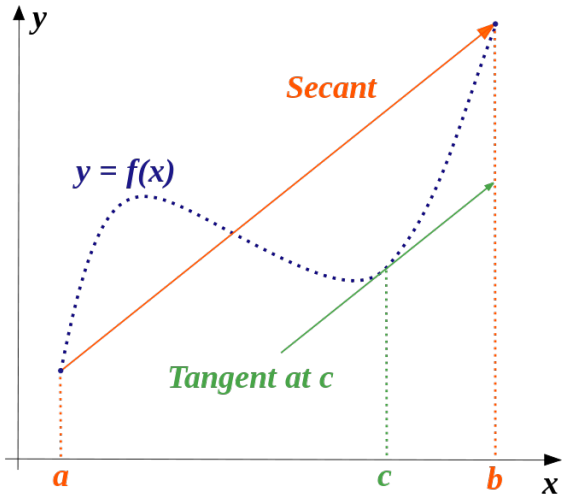

Figure 2.4: Illustration of the mean value theorem

We now use the contraction mapping theorem theorem and the mean value theorem to prove that the first iterative process defined in 2.6.3 will converge for any starting value in $[1, 2]$.

Example 2.6.6. *Consider the function $f(x) = e^x - 2x - 1$ as defined in question 2.6.3. Recall from this example that $f(x)$ has a root ξ in the interval $[1, 2]$, and that ξ is a fixed point of the function $g(x)$ defined by*

$$g(x) = \ln(2x + 1)$$

as in (2.12). It is possible to show that this iterative process will in fact converge for any starting value $x_0 \in [a, b]$.

We note that the function is continuous on $[1, 2]$ and differentiable on $(1, 2)$. By the mean value theorem for any $x, y \in [1, 2]$ we have

$$\left|\frac{g(x) - g(y)}{x - y}\right| = |g'(\eta)| \quad \iff \quad |g(x) - g(y)| = |x - y||g'(\eta)|, \tag{2.15}$$

for some η that lies between x and y.

Differentiating g gives

$$g'(x) = \frac{2}{2x + 1}, \quad g''(x) = -\frac{4}{(2x + 1)^2},$$

and since $g''(x) < 0$ for all x in the interval $[1, 2]$ this means that $g'(x)$ is monotonically decreasing in the interval $[1, 2]$. Hence we have

$$g'(1) \geq g'(\eta) \geq g'(2)$$

i.e.

$$g'(x) \in [2/5, 2/3] \quad \forall x \in [1, 2].$$

Thus, we can deduce from 2.15 that

$$|g(x) - g(y)| \leq \frac{2}{3}|x - y|,$$

i.e. $L = 2/3$ *The function* $g(x)$ *is a contraction on* $[1, 2]$ *and therefore according to the contraction mapping theorem the sequence* (x_n) *will converge for any starting value in the interval* $[1, 2]$.

We now consider what's known as a 'local' version of the contraction mapping theorem.

Theorem 2.6.7. *Suppose that we have a real continuous function $g(x)$ defined on $[a, b]$, and assume that $g(x) \in [a, b]$ for all $x \in [a, b]$. Let ξ be a fixed point of g, i.e. $g(\xi) = \xi$, and the existence of ξ is guaranteed by theorem 2.6.1, and assume further that $g(x)$ has a continuous derivative in some neighbourhood of ξ with $|g'(\xi)| < 1$. Then the sequence (x_k) given by $x_{k+1} = g(x_k)$ converges to ξ as $k \to \infty$, provided that x_0 is sufficiently close to ξ.*

Sketch of Proof. The proof is left as an exercise, but here is an outline:

- Use the continuity of g' to show that $|g'(\xi)| \leq L < 1$ in a small neighbourhood of ξ.
- Then we consider $x_{k+1} - \xi = g(k_i) - g(\xi)$ and by applying the mean value theorem.

□

Definition 2.6.2. *Suppose that $g \in C([a, b])$ and that ξ is a fixed point of g (i.e. $g(\xi) = \xi$). If the sequence (x_i) defined by $x_{i+1} = g(x_i)$ converges to ξ for all x_0 in a neighbourhood of ξ then ξ is a stable fixed point. Conversely, if (x_i) does not converge for any x_0 in **any** neighbourhood of ξ other than $x_0 = \xi$, then ξ is an unstable fixed point.*

Note that fixed points may be neither stable, nor unstable.

The next theorem will ultimately explain why the sequence 2.13 in example (2.6.3) failed.

Theorem 2.6.8. *If g is continuous and has a continuous derivative in the neighbourhood of one of its fixed points, ξ, and $|g'(\xi)| > 1$. Then the sequence (x_k) defined by $x_{k+1} = g(x_k)$ does not converge to ξ for any starting value of $x_0 \neq \xi$, unless the sequence reaches ξ in a finite number of steps.*

Proof. Suppose that $x_0 \neq \xi$. Then since $|g'(\xi)| > 1$ by assumption it is possible to construct an interval $I_\delta = [\xi - \delta, \xi + \delta]$ such that $|g'(x)| \geq L$ for all $x \in I_\delta$ and $L > 1$. Then, if the sequence does not reach ξ in a finite number of steps, then suppose that there is a $k \geq 0$ such that $0 < |x_k - \xi| < \delta$; then

$$|x_{k+1} - \xi| = |g(x_k) - g(\xi)| = \left|(x_k - \xi)g'(\eta_k)\right| \geq L\,|x_k - \xi|$$

for some η_k between x_k and ξ. We note here that the second equality was obtained using the mean value theorem. If $x_{k+1} \in I_\delta$ then the same argument gives

$$|x_{k+2} - \xi| \geq L\,|x_{k+1} - \xi| \geq L^2\,|x_k - \xi|\,.$$

Since $L > 1$, eventually one of the members of the sequence $x_{k+1}, x_{k+1}, \ldots$ will fall outside the interval I_δ. Hence there is no value $k = k(\delta)$ such that $|x_k - \xi| \leq \delta$ for all $k \geq k_0$. Hence, x_k does not converge to ξ. □

Finally we return to the example given in 2.6.3. Consider once again $g(x) = (e^x - 1)/2$. Differentiating this gives

$$g'(x) = \frac{1}{2}e^x, \quad g''(x) = \frac{1}{2}e^x$$

and since $g''(x) > 0$ for all $x \in [1, 2]$ the derivative $g'(x)$ is monotonically increasing for all $x \in [1, 2]$. Hence

$$g'(x) \in \left[\frac{1}{2}e, \frac{1}{2}e^2\right]$$

and $|g'(\eta)| > 1$ for all $\eta \in [1, 2]$. Hence the sequence defined by $x_{k+1} = g(x_k)$ will not converge to ξ for any starting value in $[1, 2]$.

2.7 Appendix: Matlab Code for Chapter 2

2.7.1 Bisection Method Code

There are two files required. The first is `bisect.m` and the second is `f.m`. Place these two m-files in a folder and change the Matlab working directory so that it corresponds with the folder that contains the files. The procedure may now be run by typing `bisect(1,2,0.00001)` from the Matlab command line.

```
%%%%%%%%%%%%%
%% bisect.m %%
%%%%%%%%%%%%%

function xb=bisect(a,b,tol)
% Matlab function bisect.m
% Solves f(x) = 0 using the bisection method

% Inputs:
% a = left endpoint of interval
% b = right endpoint of interval
% tol = tolerance for stopping bisection method
%
% Required m-file:
% f.m is m-file for function f(x)
% Outputs:
% Root by bisection
%%

i=0; % set iteration count to zero
```

```
fa=f(a);
fb=f(b);

% First check the signs at the end of the interval

if fa*fb > 0
  error('Root not in bracket');
end

% begin while loop

while (b-a)>tol
  i=i+1;
  xi=a+0.5*(b-a);
  % We need to computes the value of the function at the new point xb
  % to see how close to the root we are.
  f_xi = f(xi);
  % table_array(i,:) = [i,a,b,xi,f_xi];
  % fprintf('\n i    a    b     xi     f(xi)')
  % fprintf('\n n = %i Solution = %15.10e error = %15.10e\n',i,xi, f_xi);

  if(i==1)
    %% We print the table headers on the first loop
    fprintf('\n i      a           b          xi_i           f(xi_i)');
  end
    %% Print the progress to the screen
  fprintf('\n %i %8.10f %8.10f %8.10f %15.5e\n',i,a,b,xi, f_xi);

  if fa*f_xi <0; % root lies in [a,xi_i]
```

```
    b=xi;
    fb=f_xi;
  else % if the above criteria is not satisfied then the root must lie in [xi_i,b]
    a=xi;
    fa=f_xi;
  end
end

% print the final results
format long e
fprintf('\n\n Computed Solution = %8.10f \n\n',xi);
fprintf(' Iteration count = %i \n\n',i);

%%%%%%%%%%%%%%%%%
%% End bisect.m %%
%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%
%% f.m %%
%%%%%%%%%%%%%%

function y=f(x)
y= x^3 + 4*x^2-10;

%%%%%%%%%%%%%%
%% End  f.m %%
%%%%%%%%%%%%%%
```

2.7.2 Newton Method Code

For this three files are required. The first is `Newton.m` ,the second is `f.m`, which is exactly the same file as in section 2.7.3, and the third is `df.m`,which is the file that contains the definition of the derivative.

As with the Bisection Method, place these two m-files in a folder and change the Matlab working directory so that it corresponds with the folder that contains the files. The procedure may now be run by typing `newton(2,0.000000001)`from the Matlab command line.

```
function xb=newton(x,tol)
% Matlab function newton.m
% Solves f(x) = 0 using the Newton's method

% Inputs:
% x0 = initial value
% tol = tolerance for stopping Newton's method
```

```
%
% Required m-files:
% f.m is m-file for function f(x)
% df.m is m-file for df(x)/dx
% Outputs:
% Root by Newton's Method
%%

i=0; % set iteration count to zero


%% diff is used to compute the difference between two
%% consecutive values. Any large number (or even inf!)
%% will do as an initial value

diff = inf;

% begin while loop
while diff > tol
  i = i+1;
  %% Store the previous value of x
  x_old = x;
  %% Now update x using Newton's formula
  x = x_old - f(x_old)/df(x_old);
  %% Compute the value of the function at the new point for comparison
  f_xi = f(x);
  %% Compute the absolute difference between the two subsequent values
  diff = abs(x_old-x);
```

```
  if(i==1)
    %% We print the table headers on the first loop
    fprintf('\n i  xi_i            f(xi_i)');
  end
    %% Print the progress to the screen
  fprintf('\n %i %8.10f %15.5e\n',i,x,f_xi);

end

% print the final results
format long e
fprintf('\n\n Computed Solution = %8.10f \n\n',x);
fprintf(' Iteration count = %i \n\n',i);
```

```
%% The function derivative
function y=df(x)
y= 3*x^2 + 8*x;
```

2.7.3 Secant Method Code

There are two files required. The first is `Secant.m` and the second is `f.m`.which has already been given. Place these two m-files in a folder and change the Matlab working directory so that it corresponds with the folder that contains the files. The procedure may now be run by typing `secant(1.5,2,0.00000001)` from the Matlab command line.

```
function xb=secant(x0,x1,tol)
% Matlab function secant.m
% Solves f(x) = 0 using the Secant method

% Inputs:
% x0,x1 = initial values
% tol = tolerance for stopping method
%
% Required m-files:
% f.m is m-file for function f(x)
% Outputs:
% Root by Secant Method
%%

i=0; % set iteration count to zero
```

```
%% diff is used to compute the difference between two
%% consecutive values. Any large number (or even inf!)
%% will do as an initial value

diff = inf;

% begin while loop
while diff > tol
  i = i + 1;
  %% Store the previous value of x
  %% Now update x using Secant Formula
  x2 = x1 - f(x1)*(x1-x0)/(f(x1)-f(x0));
  %% Compute the value of the function at the new point for comparison
  f_xi = f(x2);
  %% Compute the absolute difference between the two subsequent values
  diff = abs(x2-x1);

  x0 = x1;
  x1 = x2;

  if(i==1)
    %% We print the table headers on the first loop
    fprintf('\n i  xi_i            f(xi_i)');
  end
    %% Print the progress to the screen
  fprintf('\n %i %8.10f %15.5e\n',i,x2,f_xi);

end
```

```
% print the final results
format long e
fprintf('\n\n Computed Solution = %8.10f \n\n',x2);
fprintf(' Iteration count = %i \n\n',i);
```

Chapter 3

Ordinary Differential Equations (Part I)

3.1 Introduction

- Sets of ordinary differential equations occur regularly within mathematical models.
- Examples include models forecasting population growth, ecology (such as predator-prey systems), engineering, physics, economics and within many other fields.
- A small subset of these problems has an explicit *analytical solution*.
- Other solutions that are sometimes possible involve quasi-analytical solutions. These are equations to which, although the exact analytical solution is not possible, a very accurate approximation (which made be considered to be quasi-analytic) is possible under certain conditions[1].
- However most ODE systems cannot be solved analytically and require a full numerical solution. Solutions to systems of ODEs is the main focus of this chapter.

[1]See Asymptotic Methods, covered in Methods 5, and 4th year course M302 Asymptotic Methods and Boundary Layer Theory

3.2 Some Useful Definitions and Examples

Throughout this course we spend a lot of time discussing the relative sizes of functions compared to other functions. For this purpose, we need a useful notation, and thus we introduce **the order symbols**, O and o

Definition 3.2.1. *The ordering symbol O (Big O) may be defined as follows. Suppose that $\phi(x)$ and $\psi(x)$ are complex valued functions, and suppose that x_0 is a limit point of a set R which does not necessarily belong to R. Then one may say that $\psi(x)$* ***is of the order*** *$\phi(x)$ and write*

$$\psi = O(\phi) \quad in \quad R$$

to mean $\exists$ a constant A (independent of x) such that

$$|\psi| \leq A|\phi| \quad \forall x \in R.$$

Also, one may write

$$\psi = O(\phi) \quad as \quad x \to x_0$$

in some neighbourhood Δ, if $\exists A$ such that

$$|\psi| \leq A|\phi| \quad \forall x \in \Delta \cap R.$$

An alternative and commonly used notation for ψ is of order the order ϕ is tilde notation

$$\psi(x) \sim \phi(x),$$

and due to the compactness of this notation it will also be employed throughout this course.

Example 3.2.1.

$$\sin x = O(x), \quad \text{as} \quad x \to 0,$$
$$\cos x = O(1), \quad \text{as} \quad x \to 0.$$

Definition 3.2.2. *The ordering symbol o (Little o) may be defined as follows. One may say* $\psi(x)$ ***is much smaller than*** $\phi(x)$*, and write*

$$\psi = o(\phi) \quad as \quad x \to x_0$$

to mean $\forall \epsilon > 0 \quad \exists$ *a neighbourhood* Δ_ϵ *of* x_0 *such that*

$$|\psi| \leq A\epsilon|\phi| \quad \forall x \in \Delta_\epsilon \cap R$$

Note that provided $\phi \neq 0$ in R then this is equivalent to saying:

$$\psi = o(\phi) \quad \text{as} \quad x \to x_0$$

if

$$\frac{\psi}{\phi} \to 0 \quad \text{as} \quad x \to x_0.$$

Some times the symbol $\ll$ is used instead of o. The symbol $\ll$ is usually read as "much less than".

Definition 3.2.3 (Initial Value Problem)**.** *Consider the first order ordinary differential equation*

$$y'(t) = f(t, y),$$

where $y(t)$ *a real-valued function of the real variable* t*,* f *is real-valued function of both* y *and* t*, and*

$$y'(t) \equiv \frac{dy}{dt}.$$

Suppose that we wish to seek $y(t)$ for all values of $t \geq t_0$. We can specify an initial condition

$$y(t_0) = y_0,$$

and putting this all together we have

$$y'(t) = f(t, y), \quad t \geq t_0, \quad y(t_0) = y_0. \tag{3.1}$$

which constitutes the definition of an initial value problem

We will often use the acronym IVP to mean Initial Value Problem.

We now define what we mean by a function being Lipschitz continuous. This was touched briefly in chapter 2, but we go into more detail here. It is a useful definition to know in general, and it will prove useful within this chapter.

Definition 3.2.4. *(Lipschitz Continuity) Suppose that the function f is a real-valued function on the closed interval $[a, b]$ of the real line. Then f is said to be locally Lipschitz continuous on $[a, b]$ is there exists a positive constant L such that*

$$|f(x) - f(y)| \leq L\,|x - y| \quad \forall x, y \in [a, b], \tag{3.2}$$

and the constant L is refereed to as the Lipschitz constant. Likewise, a function that is globally Lipschitz continuous satisfies the same property as the above, with the one exception that

$$|f(x) - f(y)| \leq L\,|x - y| \quad \forall x, y \in \mathbb{R} \tag{3.3}$$

If $0 < L < 1$ then the function $f(x)$ is a contraction on $[a, b]$, which we have sen before.

Another way of thinking of a Lipschitz continuous function is to rearrange (3.2) as

$$\frac{|f(x) - f(y)|}{|x - y|} \leq L \quad \forall x, y \in [a, b],$$

i.e. the absolute value of the gradient of the line connecting the points $(x, f(x))$ and $(y, f(y))$ is bounded by L for all x, y in the interval.

Let's give a few examples for the sake of clarity: An example of a function that is locally Lipschitz continuous is

$$f(x) = x^2, \quad x \in [0, 1],$$

since the absolute value of the derivative of this function is bounded by 2 on this interval. However the function

$$f(x) = x^2, \quad x \in (-\infty, \infty),$$

is not globally Lipschitz because the derivative approaches infinity as $x \to \pm\infty$.

An example of a function that is both locally and globally Lipschitz is

$$f(x) = \sqrt{x^2 + 1},$$

since the function is differentiable everywhere, and the absolute value of the derivative is bounded by 1. Thus, for this function the Lipschitz constant $L = 1$.

An example of a function that is not locally Lipschitz continuous is

$$f(x) = \sqrt{x}, \quad x \in [0, 1],$$

since the derivative of the function becomes infinite as x approaches zero, and so the gradient is unbounded on this interval.

We now define the following, which will prove useful

Definition 3.2.5. *(Autonomous ODE) When $f = f(y)$, i.e. f has no explicit dependence on the independent variable t, we say that the first order ODE* (**??**) *is* autonomous.

Autonomous ODEs are usually easier to solve as they allow for separation of variables.

You may be forgiven for thinking that such an equation will always have a unique solution. However we can show through simple examples that this is not always the case. To appreciate this consider the following example.

Example 3.2.1. *(Example of an IVP that does not have a unique solution) Consider the initial value problem*

$$y' = |y|^{\alpha}, \quad y = y(t), \quad t \geq 0, \quad y(0) = 0, \quad \alpha > 0.$$

First suppose that $\alpha \in (0, 1)$. It is straightforward to verify via differentiation that the solution to the above on the interval $[0, \infty)$ is

$$y(x; c) = \begin{cases} (1-\alpha)^{\frac{1}{1-\alpha}}(x-c)^{\frac{1}{1-\alpha}}, & c \leq x < \infty \\ 0, \quad 0 \leq x \leq c, \end{cases}$$

where $y = y(x; c)$ denotes that the function y is dependent upon the variable x and is parameterised by some value $c \geq 0$. So in fact this IVP has an family of solutions $\{y(x; c)\}$ infinite in number and parameterised by c.

In contrast with the above, if $\alpha \in [1, \infty)$ the above initial value problem has a unique solution given by $y(x) \equiv 0$.

On the basis of what we have seen above we introduce the notion of a well-posed problem.

Definition 3.2.6. *(Well-posed IVP) We say that an initial value problem is* well posed *for $t_0 \leq t \leq t_{\max}$ if the following conditions are satisfied:*

1. *A solution exists*
2. *The said solution is unique*
3. *The solution depends continuously on f*

Informally, the last condition in the above definition means that if we change f by a little bit, then the solution does not change by a lot. Problems that are not well-posed are said to be ill-posed.

Using our new terminology we can now say that the IVP given in example 3.2.1 was ill-posed for $\alpha \in (0, 1)$ and well-posed for $\alpha \in [1, \infty)$.

Notice however that even in the case $\alpha \in (0, 1)$ in the last example there was an interval of the real line where the solution was indeed unique. This in fact always true, provided that the function f is Lipschitz continuous in y and continuous in t. This leads us to our next theorem, known as the Picard-Lindelöf Theorem, or more dramatically The Fundemental Theorem of ODEs. A statement of the theorem is given below

Theorem 3.2.2. *(Picard-Lindelöf Theorem) Consider initial value problem*

$$y'(t) = f(t, y(t)), \quad t \geq t_0, \quad y(t_0) = y_0. \tag{3.4}$$

Suppose that f is Lipschitz continuous in y and continuous in t. Then there exists some $\epsilon > 0$ such that the solution $y(t)$ to the IVP in unique in the interval $[t_0 - \epsilon, t_0 + \epsilon]$.

Proof. (Sketch of proof) The proof is obtained by transforming the differential equation and applying fixed point theory discussed in the previous chapter. By integrating both

sides of equation 3.4 the solution $y(t)$ must also satisfy the integral equation

$$y(t) - y_0 = \int_{t_0}^{t} f(s, y(s))\mathrm{d}s.$$

The proof of existence of the solution is obtained by successive approximations (known in this context as Picard iterations). We set up an iterative process

$$\phi_{k+1} = y_0 + \int_{t_0}^{t} f(s, \phi_k(s))\mathrm{d}s, \quad \phi_0(t) = y_0.$$

It can then be shown, using the Banach fixed point theorem that the sequence of Picard iterates ϕ_k converges to the required solution, and that the solution is unique in the required interval.

We won't provide a formal proof here. A nice explanation of the proof can be found here

`http://proofwiki.org/wiki/Picard's_Existence_Theorem`

□

So we can use the Picard-Lindelöf Theorem to determine whether a particular IVP will have a unique solution in *some* interval. However the last example (example 3.2.1)appeared to indicate that in order for the solution of the IVP to be unique throughout the entire domain, certain growth conditions of the function $f(t, y)$ with respect to y (i.e. the second variable) must be met. The precise hypotheses on the function f guaranteeing the existence of a unique solution to the IVP are given by *Picard's theorem.* A statement of Picard's theorem is given below:

Picard's Theorem gives sufficient conditions for an IVP to be well-posed.

Theorem 3.2.3 (Picard's Theorem)**.** *Suppose that* $f(t, y) \in C([t_0, t_{\max}] \times [y_0 - C, y_0 + C])$

satisfies $|f(t, y_0)| \leq K$ *and a Lipschitz condition in its second variable, i.e.*

$$|f(t, u) - f(t, v)| \leq L\,|u - v| \qquad \forall u, v \in [y_0 - C, y_0 + C]$$

for all $t \in [t_0, t_{\max}]$. *Assume further that*

$$C \geq \frac{K}{L}\left(e^{L(t_{\max} - t_0)} - 1\right),$$

Then there exists a unique function $y \in C^1([t_0, t_{\max}])$ *such that* $y(t_0) = y_0$ *and* $y' = f(t, y)$ *for all* $t \in [t_0, t_{\max}]$. *Moreover,*

$$|y(t) - y_0| \leq C.$$

Proof. The proof is quite long, and not given here. The interested reader should consult Suli's book. □

One important point to note here is that Picard's theorem gives a sufficient but not necessary condition for a unique solution. Consider for example the initial value problem

$$y' = y^2, \quad y(0) = 1.$$

It can be shown that this function does not satisfy the hypotheses for Picard's theorem, although a unique solution can clearly be obtained via integration, resulting in

$$y = \frac{1}{1 - t}, \quad 0 \leq t \leq 1,$$

and this is not continuous or differentiable on any interval $[t_0, t_{max}]$ for $t_{max} \geq 1$.

Throughout this chapter we will assume that all the ODEs that we look at satisfy the conditions for Picard's theorem, and thus a unique solution defined for the entire domain will always exist.

3.3 Euler's Method

To find a solution to the IVP on the interval $[t_0, t_{\max}]$, we first specify *mesh points*, $t_n = t_0 + nh$, for $n = 0, 1, ..., N$ and where $h = (t_{\max} - t_0)/N$ is called the *step size* or *mesh size*. For each t_n, we will seek an approximation y_n to $y(t_n)$, the value of the analytical solution at the point t_n. These values y_n are evaluated in succession for each $n = 0, 1, 2, \ldots N$

The initial condition for $y(t_0)$ is already helpfully called y_0. Suppose that we know y_n, how should we choose y_{n+1}?

Using Taylor's theorem we may expand about the point t_n as

$$y(t_n + h) = y(t_n) + hf(t_n, y(t_n)) + \mathrm{O}(h^2), \tag{3.5}$$

where $\mathrm{O}(h^2)$ is read as 'of the order h^2' (for a formal definition see appendix 3.2). So, if y_n is a good approximation to $y(t_n)$, and h is small, $y_n + hf(t_n, y_n)$ ought to be a good approximation to $y(t_{n+1})$. On that basis, we to form our first numerical method for the solution of an IVP we replace $y(t_n)$ with y_n and discard the $\mathrm{O}(h^2)$ terms, which means we arrive at Euler's Method, defined formally below.

Definition 3.3.1. *Given $y_0 = y(t_0)$, the iteration:*

$$y_{n+1} = y_n + hf(t_n, y_n)$$

for $n = 0, 1, ..N - 1$ is known as Euler's Method.

Example 3.3.1. *Lets look at an example. Consider the IVP*

$$\frac{dy}{dt} = f(t, y) = y + t,$$

for $t \in [0, 1]$, where $y(0) = 0$.

The exact solution is

$$y(t) = e^t - t - 1.$$

Lets see how well Euler's method does against the exact solution. The Euler method was implemented for $N = 10$ and $N = 100$ and the results obtained are shown in the following figures.

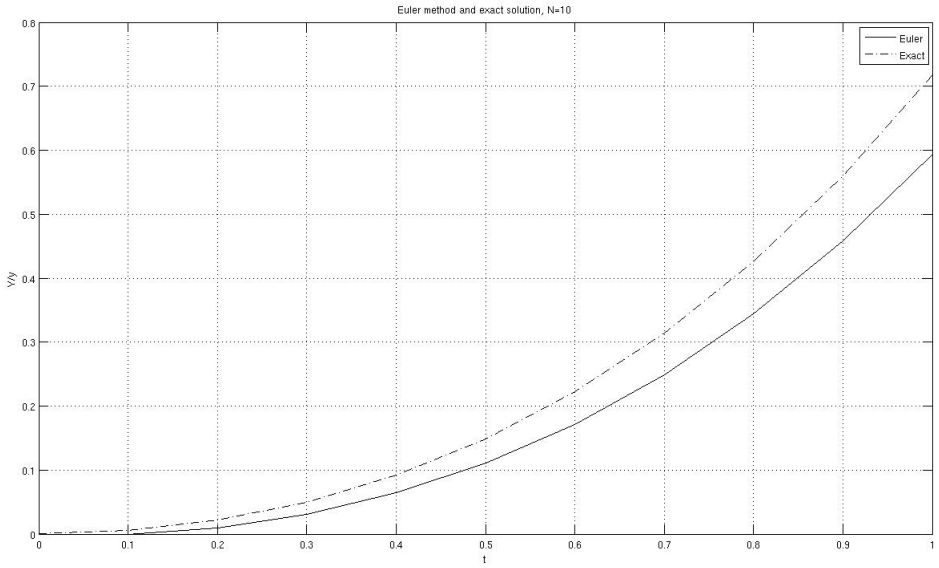

Figure 3.1: Euler's Method for Example 3.3.1 with $N = 10$

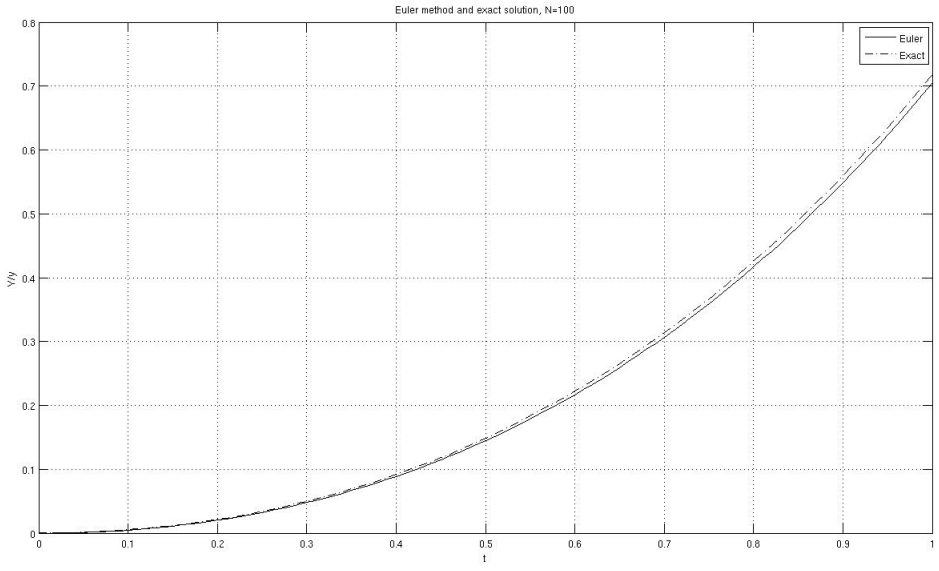

Figure 3.2: Euler's Method for Example 3.3.1 with $N = 100$

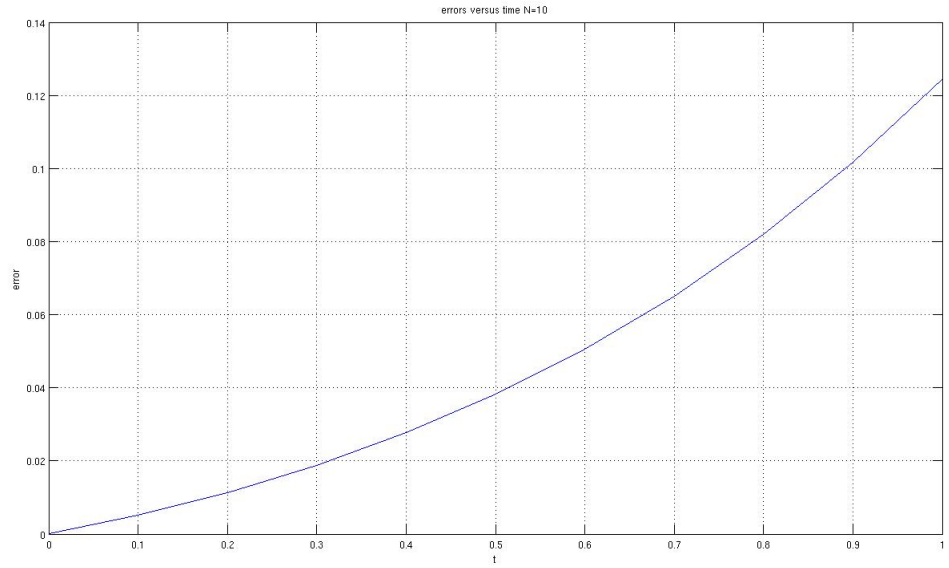

Figure 3.3: Euler's Method for Example 3.3.1 with $N = 10$

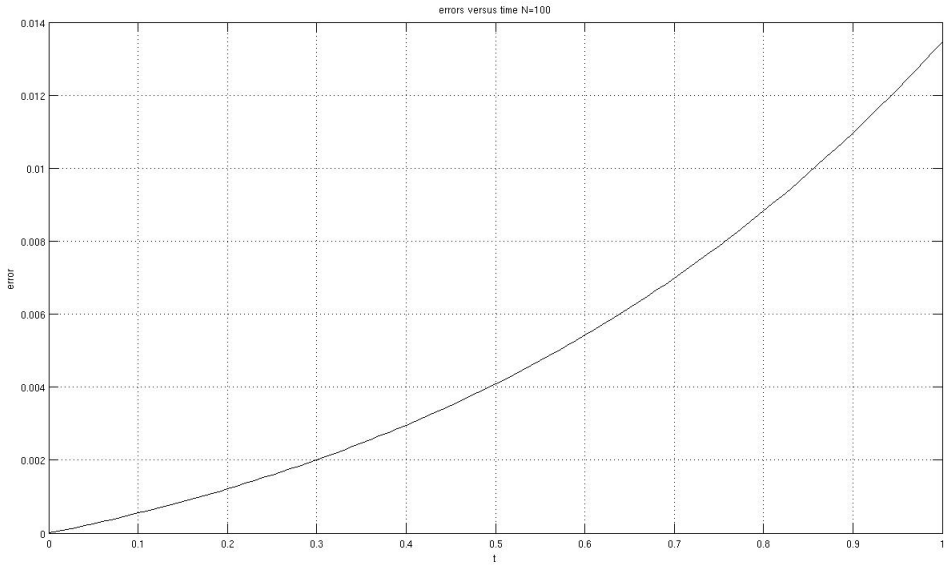

Figure 3.4: Euler's Method for Example 3.3.1 with $N = 100$

Two things appear to be apparent from the results below (as well as runs with even larger values of N):

1. As we march forward in time, the absolute error of the Euler solution increases.
2. Increasing the number of steps (or equivalently reducing the step/mesh size) appears to give better results.

We now proceed to formally analyse the error.

3.3.1 Error Analysis of Euler's Method

In order to analyse Euler's method, we first define a general one-step method as

$$y_{n+1} = y_n + h\Phi(t_n, y_n; h), \qquad n = 0, 1, \ldots, N-1, \quad y(t_0) = y_0, \tag{3.6}$$

where $\Phi(,,)$ is a continuous function of its variable. In the case of Euler's method, we have $\Phi(t_n, y_n; h) = f(t_n, y_n)$.

We define the *global error* of a numerical method e_n as

$$e_n = y(t_n) - y_n,$$

which is the difference between the exact value of the function y at the toint t_n and the value obtained using the one-step method.

The *truncation error*, T_n as

$$T_n = \frac{y(t_{n+1}) - y(t_n)}{h} - \Phi(t_n, y(t_n); h). \tag{3.7}$$

What we'd like to do is obtain a bound on the global error e_n for a general one-step method. As it turns out it is possible to bound the error e_n for a general one-step method in terms of the truncation error, as indicated in the next theorem. However first we need

a lemma:

Lemma 3.3.2. *Suppose that*

$$|e_{n+1}| \leq \alpha\,|e_n| + \beta, \tag{3.8}$$

where α and β are positive. Then, if $|e_0| = 0$,

$$|e_n| \leq \frac{\beta}{\alpha - 1}(\alpha^n - 1) \tag{3.9}$$

for all $n \geq 0$.

Proof. Clearly it's true for $n = 0$ since $e_0 = 0$ and

$$|e_0| \leq \frac{\beta}{\alpha - 1}(\alpha^0 - 1) = 0.$$

For the assumption step, assume that for some $n = k$

$$|e_k| \leq \frac{\beta}{\alpha - 1}(\alpha^k - 1),$$

which means that we are required to prove that

$$|e_{k+1}| \leq \frac{\beta}{\alpha - 1}(\alpha^{k+1} - 1).$$

This is quite straightforward. From equation (3.8) we have

$$\begin{aligned} |e_{k+1}| &\leq \alpha\,|e_k| + \beta \\ &\leq \alpha\frac{\beta}{\alpha - 1}(\alpha^k - 1) + \beta \\ &= \frac{\beta}{\alpha - 1}(\alpha^{k+1} - 1), \end{aligned}$$

which is precisely what we wanted. So, by induction, equation (3.9) is true for all n. □

Theorem 3.3.3. *Consider the general one step method* (3.6) *where, in addition to being a continuous function of its arguments Φ is assumed to satisfy a Lipschitz condition with*

respect to its second argument, i.e $\exists L_\Phi \in \mathbb{R}, L_\Phi > 0$ *such that for all* (t,u) *and* (t,v) *in the rectangle*

$$D = \{(t,y) : t_0 \le t \le t_{max}, |y - y_0| \le C\}$$

we have that

$$|\Phi(t,u,h) - \Phi(t,v,h)| \le L_\Phi |u - v| . \tag{3.10}$$

Then, assuming that $|y_n - y_0| \le C$ *for* $n = 1, 2, 3, \ldots N$ *it follows that*

$$|e_n| \le \frac{T}{L_\Phi}\left(e^{L_\Phi(t_n - t_0)} - 1\right) \quad n = 1, 2, 3 \ldots N, \tag{3.11}$$

where

$$T = \max_{0 \le n \le N-1} |T_n| .$$

Proof. Rewrite the truncation error T_n (3.7) as

$$y(t_{n+1}) = y(t_n) + h\Phi(t_n, y(t_n); h) + hT_n$$

and subtracting the general definition of a one-step method (3.6) from the above we obtain

$$e_{n+1} = e_n + h\left[\Phi(t_n, y(t_n), h) - \Phi(t_n, y_n, h)\right] + hT_n.$$

Then, since $y(t_n)$ and y_n both belong to D, the Lipschitz condition (3.10) gives

$$\begin{aligned}
|e_{n+1}| &= |e_n + h\left[\Phi(t_n, y(t_n), h) - \Phi(t_n, y_n, h)\right] + hT_n| \\
&\le |e_n| + h\,|\Phi(t_n, y(t_n), h) - \Phi(t_n, y_n, h)| + h\,|T_n| \\
&\le |e_n| + hL_\Phi\,|y(t_n) - y_n| + h\,|T_n| \\
&= |e_n| + hL_\Phi\,|e_n| + h\,|T_n| , \quad n = 0, 1, 2, \ldots N - 1.
\end{aligned}$$

That is

$$|e_{n+1}| \leq (1 + hL_\Phi)\,|e_n| + h\,|T_n| \quad n = 0, 1, 2, \ldots N-1.$$

It then follows from the induction argument in lemma 3.3.2 that

$$|e_n| \leq \frac{T}{L_\Phi}\left[(1 + hL_\Phi)^n - 1\right], n = 0, 1, 2, \ldots N,$$

since $e_0 = 0$ (i.e. an initial condition is always known). Then observing that

$$1 + hL_\Phi \leq \exp(hL_\Phi),$$

and $nh = t_n - t_0$ gives the result

$$|e_n| \leq \frac{T}{L_\Phi}\left(e^{L_\Phi(t_n - t_0)} - 1\right) \quad n = 1, 2, 3 \ldots N,$$

where

$$T = \max_{0 \leq n \leq N-1} |T_n|\,.$$

□

We now look to apply this result to Euler's method. For Euler's method the truncation error is

$$T_n = \frac{y(t_{n+1}) - y(t_n)}{h} - f(t_n, y(t_n)) = \frac{y(t_{n+1}) - y(t_n)}{h} - y'(t_n),$$

which is essentially the difference between a (forward) finite-difference approximation to the derivative at t_n and the exact value of the derivative of y at t_n.

So by Taylor's theorem in the form (assuming that f has a continuous second derivative in the domain)

$$y(t_{n+1}) = y(t_n) + hy'(t_n) + \frac{1}{2}h^2 y''(\xi_n), \quad t_n < \xi_n < t_{n+1},$$

and substituting this into our expression for the truncation error (3.7) we obtain

$$T_n = \frac{1}{2}hy''(\xi_n), \quad \text{for some} \quad \xi_n \in (t_n, t_{n+1}).$$

Now let

$$M_2 = \max_{\zeta \in [t_0, t_{max}]} \left| y''(\zeta) \right|,$$

and so

$$T = \frac{1}{2}hM_2.$$

Then inserting these into equation (3.11) yields

$$|e_n| \leq \frac{1}{2}M_2 \left(\frac{e^{L(t_n - t_0)} - 1}{L} \right) h, \quad n = 0, 1, \ldots N. \tag{3.12}$$

where we note that for Euler's method $\Phi \equiv f$, and $L_\Phi = L$ is the Lipschitz constant for f.

Example 3.3.4. *Consider the IVP*

$$y' = \tan^{-1}(y), \quad y(0) = y_0$$

where y_0 is some given real number. Find a bound for the global error $e_n = y(t_n) - y_n$.

In order to do this we need to determine the constants L and M_2 in equation (3.12)*. On this occasion*

$$f(t, y) = \tan^{-1}(y)$$

and so by the mean value theorem

$$|f(t, u) - f(t, v)| = \left| \frac{\partial f}{\partial y}(t, \eta)(u - v) \right| = \left| \frac{\partial f}{\partial y}(t, \eta) \right| |(u - v)|,$$

where η lies between u and v. In our case we have

$$\left|\frac{\partial f}{\partial y}(t,y)\right| = \frac{1}{1+y^2} \leq 1,$$

and therefore $L = 1$. To find the constant M_2 we need a bound on y'', which is easily obtained by differentiating the differential equation with respect to t, i.e:

$$y'' = \frac{d}{dt}(\tan^{-1}(y)) = \frac{1}{1+y^2}\frac{dy}{dt} = \frac{1}{1+y^2}\tan^{-1}(y).$$

Therefore we may conclude that

$$|y''| \leq M_2 = \frac{1}{2}\pi.$$

Now finally inserting the values of L and M_2 into (3.12) we have

$$|e_n| \leq \frac{1}{4}\pi(e^{t_n} - 1)h, \quad n = 0, 1, \ldots N.$$

Thus, if we were to specify a tolerance `TOL`*, we could ensure that the error between the (unknown) analytical solution and our Euler approximation by specifying a step size h such that*

$$h \leq \frac{4}{\pi(e^{t_{max}-1})}\texttt{TOL}$$

and for such an h we would have $|y(t_n) - y_n| \leq$ `TOL` *for all $n = 0, 1, 2, \ldots N$ as required. Actually as it turns out this estimate for h is five times smaller than is actually required: Taking for example $y_0 = 1$ and $t_{max} = 1$ our bound implies that the tolerence* `TOL` $= 0.01$ *will be achieved with $h \leq 0.0074$, for which we would need $N \geq 135$. Actually using $N = 27$ gives a result using Euler's method which is just inside this tolerance. So the bound gave a estimate for h that is five times smaller than what was required.*

3.3.2 Consistencey and Convergence

Recall that for a general one-step method we have

$$y_{n+1} = y_n + h\Phi(t_n, y_n; h),$$

and that for the Euler method

$$y_{n+1} = y_n + hf(t_n, y_n),$$

Theorem 3.3.3 shows that if the truncation error tends to zero as $h \to 0$, then so does the global error. Consider the truncation error for a general one-step method once more:

$$T_n = \frac{y(t_{n+1}) - y(t_n)}{h} - \Phi(t_n, y(t_n); h).$$

The truncation error (also known as the *local error*) is the error committed by one step of the method, assuming that no prior errors have been made in earlier steps. Suppose we let $n \to \infty, h \to 0$ in such a way that $t_n \to t \in [t_0, t_{max}]$ then we have

$$\lim_{n\to\infty} T_n = y'(t) - \Phi(t, y(t); 0),$$

since

$$\lim_{n\to\infty, h\to 0} \frac{y(t_{n+1}) - y(t_n)}{h} = y'(t)$$

and this prompts the following definition:

Definition 3.3.2. *A numerical method* (3.6) *is consistent with the differential equation* (3.1) *if*

$$\Phi(t, y; 0) = f(t, y)$$

Now consider any point $t \in [t_0, t_{max}]$. Suppose that $h \to 0$ and choose $n(h)$ such that

$t_n \to t$. We have assumed that both Φ and y' are continuous, so

$$\lim_{h\to 0} T_{n(h)} = y'(t) - \Phi(t, y; 0) = f(t, y) - f(t, y)$$

In other words, *if a numerical method is consistent, and the conditions for Theorem 3.3.3 are satisfied (i.e. Φ is Lipschitz) then the truncation errors, and hence the global errors, tend to zero as $h \to 0$*

Definition 3.3.3. *A numerical method is said to have order of accuracy p for a given problem if there exist constants C and h_0 such that the truncation error T_n satisfies*

$$|T_n| \leq Ch^p \qquad \forall \quad 0 \leq h \leq h_0,$$

for all n, and $p \in \mathbb{Z}$ is its largest possible value.

We can also write this as $|T_n| = O(h^p)$.

For the Euler method we have

$$y_{n+1} = y_n + hf(t_n, y_n) + \mathrm{O}(h^2),$$

meaning that the Euler method is *first order accurate.*

3.4 Runge-Kutta methods

- Euler's method is only first order accurate.
- Nonetheless it is simple and cheap to implement, cheap because we only require a single evaluation of the function f at each step.
- Runge-Kutta methods achieve higher accuracy but sacrifice efficiency by evaluating $f(.,.)$ at intermediate points between $(t_n, y(t_n)$ and $(t_{n+1}, y(t_{n+1})$.

Lets consider a general numerical method given by

$$y_{n+1} = y_n + h(ak_1 + bk_2)$$

where

$$k_1 = f(t_n, y_n), \qquad k_2 = f(t_n + \alpha h, y_n + \beta h k_1),$$

where the parameters a, b, α, β are to be determined.

- Note that the Euler method is a member of this family of methods, with $a = 1$ and $b = 0$.
- However we now wish to seek methods that are *at least* second order accurate.

In other words, using our general one-step method definition

$$\Phi(t_n, y_n; h) = af(t_n, y_n) + bf(t_n + \alpha h, y_n + \beta h f(t_n, y_n))$$

Can we pick a, b, α, β so that this is a good numerical scheme?

- First note that for a method from this family to be consistent we must have $a + b = 1$, because

$$as \quad n \to \infty, h \to 0, \quad \Phi \to af(t, y(t)) + bf(t, y(t)) \equiv f(t, y(t)) \iff a + b = 1.$$

- Further conditions on the parameters are found by attempting to maximise the order of the method.

To get further information, we're going to need Taylor's theorem for functions of 2 variables. Consider a twice differentiable function $f(x, y)$. We have:

$$f(x + \xi, y + \eta) = f(x, y) + \xi f_x(x, y) + \eta f_y(x, y) + \frac{1}{2!}\left(\xi^2 f_{xx} + 2\xi\eta f_{xy} + \eta^2 f_{yy}\right) + O(h^3)$$

Lets apply this to Φ to obtain:

$$\Phi(t_n, y(t_n); h) = af + b(f + \alpha h f_t + \beta h f f_y + \tfrac{1}{2}\alpha^2 h^2 f_{tt} + \alpha\beta h^2 f f_{ty} + \tfrac{1}{2}\beta^2 h^2 f^2 f_{yy} + O(h^3))$$

Hence the truncation error,

$$\begin{aligned} T_n &= \frac{y(t_n + h) - y(t_n)}{h} - \Phi(t_n, y(t_n); h) \\ &= f + \tfrac{1}{2}h(f_t + f f_y) + \tfrac{1}{6}h^2(f_{tt} + 2f_{ty}f + f_{yy}f^2 + f_y(f_t + f f_y)) \\ &\quad - \left((a+b)f + hb(\alpha f_t + \beta f f_y) + h^2(\tfrac{1}{2}\alpha^2 f_{tt} + \alpha\beta f f_{ty} + \frac{1}{2}\beta^2 f^2 f_{yy}\right) + O(h^3) \end{aligned}$$

Consistency means that the term $f - (a+b)f$ is equal to zero. The coefficient of the term in h is

$$\tfrac{1}{2}(f_t + f f_y) - (b\alpha f_t + b\beta f f_y)$$

This will be equal to zero for all functions, f if (and only if) $b\alpha = b\beta = \frac{1}{2}$. The method is therefore second-order accurate if

$$\beta = \alpha, \qquad b = \frac{1}{2\alpha}, \qquad a = 1 - \frac{1}{2\alpha}$$

The truncation error then becomes

$$T_n = h^2\left((\tfrac{1}{6} - \tfrac{\alpha}{4})(f_{tt} + f_{yy}f^2) + (\tfrac{1}{3} - \tfrac{\alpha}{2})f f_{ty} + \tfrac{1}{6}(f_t f_y + f f_y^2)\right) + O(h^3) \tag{3.13}$$

- Choosing $\alpha = \frac{1}{2}$ gives the *modified Euler Method* (also known as the 'mid-point' method).

$$y_{n+1} = y_n + hf\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hf(t_n, y_n)\right). \tag{3.14}$$

- Choosing $\alpha = 1$ gives the *improved Euler method*

$$y_{n+1} = y_n + \frac{1}{2}h\left[f(t_n, y_n) + f(t_n + h, y_n + hf(t_n, y_n))\right]. \tag{3.15}$$

- Note that both these methods are also variously referred to as "the"modified Euler method.

- It is easily verified using (3.13) that the Truncation error for the these two methods are given by

$$T_n = \frac{1}{6}h^2\left[f_y(f_t + f_y f) + \frac{1}{4}(f_{tt} + 2f_{ty}f + f_{yy}f^2)\right] + \mathrm{O}(h^3)$$
$$T_n = \frac{1}{6}h^2\left[f_y(f_t + f_y f) - \frac{1}{2}(f_{tt} + 2f_{ty}f + f_{yy}f^2)\right] + \mathrm{O}(h^3)$$

  the derivation of which is left as an exercise.

These schemes are examples of (explicit) *Runge-Kutta* methods (pronounced Run-Ga-Kutta). General (explicit) s-stage Runge-Kutta schemes have the form

$$\begin{aligned} y_{n+1} &= y_n + h\sum_{j=1}^{s} b_j k_j \\ k_1 &= f(t_n, y_n) \\ k_2 &= f(t_n + c_2 h, y_n + h a_{21} k_1) \\ &\vdots \\ k_s &= f\left(t_n + c_s h, y_n + h(a_{s1} + \ldots a_{s,s-1}k_{s-1})\right) \end{aligned}$$

We have seen examples of two 2-stage explicit Runge-Kutta methods that have order 2 (and seen that the order cannot be higher than 2). There exist 3-stage Runge-Kutta methods with order 3 and a widely used 4-stage method with order 4. Interestingly, to get to order 5, one needs to take 6 stages.

One of the most frequently used methods of the Runge-Kutta family is often known as the classical fourth-order Runge-Kutta Method, or RK4. This is given by

$$y_{n+1} = y_n + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4), \tag{3.16}$$

with

$$\begin{aligned} k_1 &= f(x_n, y_n), \\ k_2 &= f(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1), \\ k_3 &= f(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_2), \\ k_4 &= f(x_n + h, y_n + hk_3). \end{aligned}$$

Here k_2 and k_3 represent approximations to the derivative y' at the points on the curve, intermediate between $(t_n, y(t_n))$ and $(t_{n+1}, y(t_{n+1}))$

3.5 Implicit methods

From Riemmann integration, we know that we can approximate an integral

$$\int_a^b f(t)dt \sim \frac{b-a}{2}(f(b) + f(a)) \tag{3.17}$$

This is known as the *trapezium rule* and we will see a more general example later on. It motivates a new kind of numerical method to solve our ODE

$$\begin{aligned} y(t_{n+1}) - y(t_n) &= \int_{t_n}^{t_{n+1}} y'(t)dt \\ &= \int_{t_n}^{t_{n+1}} f(t, y(t))dt \\ &\sim \frac{h}{2}(f(t_{n+1}, y(t_n)) + f(t_n, y(t_n))). \end{aligned}$$

Hence we could write

$$y_{n+1} = y_n + \frac{h}{2}(f(t_n, y_n) + f(t_{n+1}, y_{n+1})) \tag{3.18}$$

This is known as the *trapezium rule method*. The tricky thing here is that the next iterate, y_{n+1}, appears on both sides of (3.18).

When this occurs, we say that the method is *implicit*. If $\frac{\partial f}{\partial y}$ is easy to compute, we can calculate y_{n+1} using Newton's method. The obvious starting point is from Euler's method: $y_n + hf(t_n, y_n)$.

Why would we do this? There are two reasons. Firstly, we obtain improved accuracy. Secondly, the method is more stable.

3.6 Systems of ODEs

We often want to solve a system of ordinary differential equations

$$\begin{aligned} \frac{dy_1}{dt} &= f_1(t, y_1, \ldots, y_m) \qquad y_1(t_0) = y_{t,0} \\ &\vdots \\ \frac{dy_m}{dt} &= f_m(t, y_1, \ldots, y_m) \qquad y_m(t_0) = y_{m,0} \end{aligned}$$

We typically write this as

$$\frac{d\mathbf{y}}{dt} = \mathbf{f}(t, \mathbf{y}) \qquad \mathbf{y}(t_0) = \mathbf{y}_0 \tag{3.19}$$

where $\mathbf{y}(t) = (y_1(t), \ldots, y_m(t))$ is a vector-valued function of time.

The solution methods and analysis of systems of ordinary differential equations are essentially the same as for a single ODE. For example, Euler's method becomes

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}(t, \mathbf{y}).$$

Chapter 4

Ordinary Differential Equations (Part II)

4.1 Multi-step methods

We saw in the last chapter that the RK4 method gives an improvement over the Euler method in terms of accuracy, but more computational effort id required, because four evaluations of f were required per step, which is perhaps (some might argue) slightly more than necessary.

At the end of the first section on ODEs, we briefly saw the trapezium rule method, which we motivated via the idea of approximating the integral in the expression

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(t, y(t))dt \tag{4.1}$$

using the trapezium rule.

Now that we know how to use high order quadratures, we can generalise this. For

example, we could use Simpson's rule to approximate the integral

$$y(t_{n+1}) = y(t_{n-1}) + \int_{t_{n-1}}^{t_{n+1}} f(t, y(t))dt.$$

Simpson's rule states that

$$\int_a^b \mathrm{d}x = \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right) + \mathrm{O}((b-a)^5),$$

thus giving for the integral

$$\int_{t_{n-1}}^{t_{n+1}} f(t, y(t))\mathrm{d}t \approx \frac{h}{3}\left(f(t_{n-1}) + 4f(t_n) + f(t_{n+1})\right).$$

to obtain the method

$$y_{n+1} = y_{n-1} + \frac{1}{3}h(f(t_{n-1}, y_{n-1}) + 4f(t_n, y_n) + f(t_{n+1}, y_{n+1})).$$

The right-hand side involves the terms y_{n-1}, so this is not a one-step method, it is a *multistep method*. It is also an implicit method.

Definition 4.1.1. *Given a set of equally spaced mesh points, t_n, with stepsize h, the general linear k-step method is given by*

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = h\sum_{j=0}^{k} \beta_j f(t_{n+j}, y_{n+j})$$

where the coefficints α_j and β_j are real constants

We assume that $\alpha_k \neq 0$. When $\beta_k = 0$, the method is *explicit*, when $\beta_k \neq 0$, the method is *implicit*

It is convenient to write $f_k = f(t_k, y_k)$

By applying Lagrange interpolation to (4.1) at the nodes $t_{n+1}, t_n, t_{n-1}, \ldots t_{n+1-k}$, we

obtain the (implicit) Adams-Bashforth methods, of which the first few are:

$$
\begin{aligned}
y_{n+1} &= y_n + hf_n \\
y_{n+2} &= y_n + \frac{1}{2}h(3f_{n+1} - f_n) \\
y_{n+3} &= y_n + \frac{1}{12}h(23f_{n+2} - 16f_{n+1} + 5f_n) \\
y_{n+4} &= y_{n+3} + \frac{1}{24}h(55f_{n+3} - 59f_{n+2} + 37f_{n+1} - 9f_n)
\end{aligned}
$$

Doing the same procedure, but using the nodes $t_n, t_{n-1} \dots t_{n-k}$, gives the Adams-Moulton methods. For example, here is the 3-step implicit A-M method

$$y_{n+3} = y_{n+2} + \frac{1}{24}h(9f_{n+3} + 19f_{n+2} - 5f_{n+1} - f_n)$$

There are systematic ways of generating these method, but we do not discuss these here.

4.2 An excursion into Linear Algebra

In many practical situations, solving ODEs and PDEs requires knowledge of techniques from linear algebra. In the next section we recap some of the ideas from linear algebra, with a focus on some of the more important ideas.

We denote the space of $m \times n$ (real-valued) matrices as $\mathbb{R}^{m,n}$ and the space of column vectors of length n as $\mathbb{R}^n$. If $A \in \mathbb{R}^{m,n}$, we denote the entry in the ith row and jth column as A_{ij}. Similarly, the ith entry of a vector, $\mathbf{x}$, is x_i.

Suppose that we know some $A \in \mathbb{R}^{n,n}$ and $b \in \mathbb{R}^n$. The equation

$$A\mathbf{x} = \mathbf{b} \tag{4.2}$$

is a system of linear equations in the unknowns $x \in \mathbb{R}^n$. This can also be written in the

expanded form

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n} &= b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n} &= b_2 \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn} &= b_n
\end{aligned}
$$

From linear algebra, we know that (4.2) has a unique solution for all $b \in \mathbb{R}^n$ if and only if $\det(A) \neq 0$.

Suppose we are given A and $\mathbf{b}$, how should we calculate $\mathbf{x}$?

From linear algebra, we might attempt to use Cramer's Rule, which states that

$$x_i = \frac{D_i}{D}, \quad i = 1, 2, \ldots n, \tag{4.3}$$

where $D = \det(A)$ and D_i is the $n \times n$ determinant obtained by replacing the ith column of D with the entries of $\mathbf{b}$. Assuming that $D \neq 0$, all we need to do to solve for each x_i is to evaluate $n+1$ determinants $D, D_1, \ldots D_n$, each of size $n \times n$ and substitute to (4.3). involves the computation of determinants of M.

This method relies heavily on the computation of determinants. The next question is, how expensive is it to compute a determinant?

Let's denote d_n as the total number of arithmetic operations requird to compute the determinant of an $n \times n$ matrix. It can be shown (left as an exercise) that

$$d_n = \mathrm{O}(en!) \quad \text{as} \quad n \to \infty,$$

and thus the total number of arithmetic operations required in order to employ Cramer's rule is about $(n+1)d_n \sim \mathrm{O}(e(n+1!))$, which is a lot! Consider for example a 100×100 system ($n = 100$). A solution via Cramer's rule would take about $e101! = 2.56 \times 10^{160}$ operations, which on a computer capable of teraflop speeds would take around 8.11×10^{140} years to complete, or to put another way, about 10^{130} times longer than the universe has been in existence! Clearly a more efficient approach is needed.

In fact for even moderately sized systems (e.g. $n = 10$), Cramer's Rule is the wrong numerical algorithm to use to compute solutions to $Mx = b$.

Fortunately, we can do a lot better than this. We also recall from linear algebra the technique of Gaussian elimination.

Example 4.2.1. *Demo of Gaussian Elimination*
Consider the system of linear equations

$$\begin{aligned} x_1 + x_2 + x_3 &= 6, \\ 2x_1 + 4x_2 + 2x_3 &= 16, \\ -x_1 + 5x_2 - 4x_3 &= -3. \end{aligned}$$

Written in matrix form we have

$$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 4 & 2 \\ -1 & 5 & -4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 6 \\ 16 \\ -3 \end{pmatrix}.$$

We start by subtracting two lots of the first row from the second, and adding the first

row to the third, to give the equivalent system

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 2 & 0 \\ 0 & 6 & -3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 6 \\ 4 \\ 3 \end{pmatrix},$$

noting the newly created zeros in the first column. Next we add three lots of the second row to the third, giving the system

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 2 & 0 \\ 0 & 0 & -3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 6 \\ 4 \\ 9 \end{pmatrix},$$

yielding the equivalent system of linear equations

$$\begin{aligned} x_1 + x_2 + x_3 &= 6, \\ 2x_2 &= 4, \\ -3x_3 &= -9, \end{aligned}$$

which can be easily solved via back-substitution beginning with $x_3 = 3$.

As it turns out, for an $n \times n$ system of equations, the technique of Gaussian elimination requires approximately $\frac{2}{3}n^3$ arithmetic operations to get to a solution, significantly less than the $O((n+1)!)$ that was given by Cramer's rule. So for a 100×100 system, approximately 0.67×10^6 arithmetic operations would be required, which would take about 10^{-6} seconds on a computer capable of Teraflop speeds, a lot less than the $O(10^{140})$ *years* needed if we were to use Cramer's rule.

We now introduce some definitions that will prove useful later in our study of differential equations.

Definition 4.2.1. *If* $M \in \mathbb{R}^{m,n}$ *satisfies* $M_{ij} = 0$ *for* $i < j$ *then* M *is a lower triangular matrix.*

Also, a matrix is unit lower triangular if it is triangular and the diagonal entries are all equal to unity, that is $M_{ii} = 1$ for $i = 1, 2, \ldots n$.

Similarly, if $M_{ij} = 0$ for $i > j$, M is an upper triangular matrix, and M is unit upper triangular if it is upper triangular and its diagonal entries are equal to unity.

Theorem 4.2.2. *The lower (upper) triangular matrices form a ring, in particular:*

1. *The product of two lower (upper) triangular matrices is a lower (upper) triangular matrix.*
2. *The product of two lower (upper) triangular matrices is a lower (upper) triangular matrix.*
3. *A lower (upper) triangular matrix is non-singular iff all diagonal elements are non-zero.*
4. *The inverse of a non-singular lower (upper) triangular matrix is a lower (upper) triangular matrix.*
5. *The inverse of a unit lower (upper) triangular matrix is a unit lower (upper) triangular matrix.*

Proof. The proof of these theorems is left as an exercise, in particular 1,2,3,5 are simple to show. Part 4 is proved by induction. □

The operations of Gaussian elimination can be thought of as a series of linear operations, each of which is represented as pre-multiplication lower triangular matrix. We see that we can write

$$L_N L_{N-1} \ldots L_2 L_1 A = U, \quad N = \frac{1}{2} n(n-1),$$

where the L_i are lower triangular matrices and U is an upper triangular matrix. We can see, from Gaussian elimination, that computing the factors, L_i is not expensive. Thus we have

$$L^{-1} A = U, \quad L^{-1} = L_N L_{N-1} \ldots L_2 L_1$$

giving the so-called LU factorisation of A as

$$A = LU$$

and L and L^{-1} are both lower triangular by the theorems for lower and upper triangular matricies, and solving (4.2) therefore requires us to compute

$$Ux = L^{-1}b.$$

The final problem of inverting U can be done cheaply via back substitution, which we also saw in the Gaussian elimination demo. Note that, in practice, a little extra care is required to avoid zero diagonal elements (as this would give a singular L). We also have to make sure to minimise rounding error. This process is known as *LU decomposition*, which we won't discuss in detail here (except in the case of so-called tri-diagonal matricies, see later). Note also that once L and U have been computed, finding x only costs about $2n^2 - n$ operations.

Other factorisations are also possible. If M is symmetric, then the *Cholesky decomposition* $M = LL^t$ is cheaper than LU. For over or underdetermined linear systems, where, instead of solving $Mx = b$, we attempt to find an x that minimises $\|Mx - b\|$, it is sometimes useful to compute $M = QR$, where Q is orthogonal and R is upper triangular.

The good news is that implementations of these algorithms are readily available in your favourite programming platform:

MATLAB: `x = A \ b`

Python: `x = numpy.linalg.solve(A,b)`

C++: Look at `http://eigen.tuxfamily.org/dox/TutorialLinearAlgebra.html`

Fortran: `http://www.netlib.org/linpack/`

4.2.1 Tri-diagonal Matricies and the Thomas Algorithm

The numerical solution to many second-order boundary value problems

$$-y'' + r(x)y = f(x), \qquad a < x < b \tag{4.4}$$

with the boundary conditions

$$y(a) = A, \quad y(b) = B,$$

can be found by discretizing the above system and writing as a *tri-diagonal system.* What is a tr-diagonal system? Let's define it:

Definition 4.2.2. *Suppose $n \geq 3$. A matrix $T \in \mathbb{R}^{n\times n}$ is said to be tri-diagonal if it has non-zero elements on the main diagonal and the two adjacent diagonals, i.e*

$$t_{ij} = 0 \quad if \quad |i-j| > 1, \quad i,j \in \{1,2,\dots n\}.$$

For a tri-diagonal matrix T the LU decomposition of T is such that both L and T have only two non-zero elements in each row. If we write

$$T = \begin{pmatrix} b_1 & c_1 & 0 & & & \\ a_2 & b_2 & c_2 & & & \\ 0 & a_3 & b_3 & & & \\ \dots & \dots & \dots & \dots & \dots & \dots \\ & & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & & a_n & b_n \end{pmatrix}$$

Then, if we seek $T = LU$, where

$$L = \begin{pmatrix} 1 & & & & \\ l_2 & 1 & & & \\ & l_3 & 1 & & \\ \dots & \dots & \dots & \dots & \dots \\ & & & l_n & 1 \end{pmatrix},$$

and

$$U = \begin{pmatrix} u_1 & v_1 & & & \\ & u_2 & v_2 & & \\ \dots & \dots & \dots & \dots & \\ & & & u_{n-1} & v_{n-1} \\ & & & & u_n \end{pmatrix}$$

and we have defined $a_1 = c_n = 0$ for convenience. Multiplying L and U shows that $v_j = c_j$ and that the elements l_j and u_j can be calculated from

$$l_j = \frac{a_j}{u_{j-1}}, \quad u_j = b_j - l_j c_{j-1}, \quad j = 2, 3, \dots n, \tag{4.5}$$

starting with $u_1 = b_1$.

Suppose that we wish to solve a system $T\boldsymbol{x} = \boldsymbol{b}$, where T is tri-diagonal and non-singular. Having already calculated the elements of L and U in the LU factorisation of T, the forward and back substitution are both very simple. Let $\boldsymbol{y} = U\boldsymbol{x}$ then we have $L\boldsymbol{y} = \boldsymbol{b}$ which gives (from forward substitution)

$$\begin{aligned} y_1 &= b_1, \\ y_j &= b_j - l_j y_{j-1}, \quad j = 2, 3, \dots n, \end{aligned}$$

and finally from $U\boldsymbol{x} = \boldsymbol{y}$ we get (via back substitution)

$$x_n = y_n/u_n,$$
$$x_j = (y_j - v_j x_{j+1})/u_j, \quad j = n-1, n-2, \ldots 1.$$

Factorisation of LU takes approximately $3n$ operations, and the forward and back substitutions require around $5n$ operations, so the whole process takes around $8n$ operations, so order n, and therefore the amount of work required is far less than for a full matrix which is of the order $2/3n^3$. The method described here is known as the Thomas algorithm[1]

This method will prove very useful in the numerical solution of boundary value problems.

[1] actually it is a minor variant of it, but in essence it is more or less the same.

4.3 Boundary Value Problems

So far, we have only explicitly described first order initial value problems. This is already quite powerful, for example, from calculus, we know that given a higher order ODE, e.g.

$$y'' + p(t)y' + q(t)y = f(t)$$

with initial conditions, $y(0) = a$, $y'(0) = b$, we could introduce the new function, $u = y'$ and obtain a system of ODEs:

$$\begin{aligned} y' &= u \\ u' &= f(t) - p(t)u - q(t)y \end{aligned}$$

with initial conditions $y(0) = a$, $u(0) = b$, which we already know techniques to solve.

However, what if rather than initial conditions, we want to impose boundary conditions?

We will consider the following model problem:

$$-y'' + r(x)y = f(x), \qquad a < x < b \tag{4.6}$$

with the boundary conditions

$$y(a) = A, \quad y(b) = B.$$

We will assume that $r(x) \geq 0$.

We will construct a numerical approximation to the solution on a uniform mesh of points, $x_j = a + jh$, $j = 0, 1, \ldots n$, $h = (b - a)/n$.

Definition 4.3.1. *The* central difference, δ_h *of* y *is defined as*

$$\delta_h y(x) = y(x + \frac{1}{2}h) - y(x - \frac{1}{2}h) \tag{4.7}$$

Higher order central differences are defined recursively, by $\delta_h^{m+1} y(x) = \delta_h(\delta_h^m y(x))$

In particular, we have the second central difference,

$$\begin{aligned}\delta_h^2 y(x) &= \delta_h y(x + \frac{1}{2}h) + \delta_h y(x - \frac{1}{2}h) \\ &= y(x+h) - 2y(x) + y(x-h)\end{aligned}$$

Theorem 4.3.1. *Suppose that* $y \in C^4([x-h, x+h])$*, then there exists* $\xi \in (x-h, x+h)$ *such that*

$$\frac{\delta_h^2 y(x)}{h^2} = y''(x) + \frac{1}{12}h^2 y^{(4)}(\xi)$$

Sketch of proof. Write out Taylor's theorem for $y(x \pm h)$ with the remainder as the h^4 term. Then add things up, divide by h^2 and apply the Intermediate Value Theorem to the 2 Taylor remainders to give ξ. □

Now lets try and solve (4.6). Write Y_j for our numerical approximation to $y(x_j)$ and $r_j = r(x_j)$ and $f_j = f(x_j)$, we will approximate the differential equation by

$$-\frac{\delta_h^2 Y_j}{h^2} + r_j Y_j = f_j, \qquad j = 1, 2, \ldots n-1 \tag{4.8}$$

This gives $n-1$ linear algebraic equations in the $n-1$ unknowns, $Y_1, Y_2, \ldots Y_{n-1}$. We also need values for Y_0 and Y_n. These are given by the boundary conditions $Y_0 = A$, $Y_n = B$.

We can write this in matrix form as

$$M\mathbf{Y} = \mathbf{g}$$

where $M \in \mathbb{R}^{(n-1)\times(n-1)}$ is tridiagonal, with

$$M_{jj} = 2/h^2 + r_j, \qquad M_{jj-1} = M_{jj+1} = \frac{-1}{h^2}$$

and

$$g_1 = f_1 + A/h^2, \quad g_{n-1} = f_{n-1} + B/h^2, \quad g_j = f_j, j = 2, 3, \ldots n-2$$

.

If $r > 0$ then M is positive definite (by Gersgorin's theorem) and so invertible. In fact, M is positive definite even for $r \geq 0$. Tridiagonal matrices are easy to factorise. Write

$$M = \begin{pmatrix} b_1 & c_1 & 0 & & & \\ a_2 & b_2 & c_2 & & & \\ 0 & a_3 & b_3 & & & \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ & & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & & a_n & b_n \end{pmatrix}$$

Then, if we seek $M = LU$, where

$$L = \begin{pmatrix} 1 & & & & \\ l_2 & 1 & & & \\ & l_3 & 1 & & \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ & & & l_n & \end{pmatrix}$$

and

$$U = \begin{pmatrix} u_1 & v_1 & & & \\ & u_2 & v_2 & & \\ \dots & \dots & \dots & \dots & \\ & & & u_{n-1} & v_{n-1} \\ & & & & u_n \end{pmatrix}$$

Then $v_j = c_j$, $j = 1 \dots n-1$ and $l_j = a_k/u_{j-1}$, $u_j = b_j - l_j c_{j-1}$ $j = 2 \dots n$. N.B. this assumes that u_j is never zero (or small). Fortunately, the positive-definiteness of M guarantees this.

... some examples

Definition 4.3.2. *The* truncation error *of the central difference approximation* (4.8) *is*

$$T_j = \frac{\delta^2 y(x_j)}{h^2} + r_j y(x_j) - f(j), \qquad j = 1, 2, \dots n-1$$

where y is the exact solution of (4.6)

Theorem 4.3.2. *Suppose that y is the solution to* (4.6) *and $y \in C^{(4)}([a,b])$. Then the truncation error at each point, T_j satisfies*

$$T_j = -\frac{1}{12} h^2 y^{(4)}(\xi_j)$$

for some $\xi_j \in (x_{j-1}, x_{j+1})$. Furthermore,

$$T_j \leq T = \frac{1}{12} h^2 M_4$$

where $M_4 = \|y^{(4)}\|_\infty$.

Outline of proof. Use Theorem 4.3.1, and the fact that y solves (4.6) □

Define the *global error* as

$$e_j = y(x_j) - Y_j$$

Just as before, we want to go from a truncation error to a global error.

We start with a lemma

Lemma 4.3.3 (A maximum principle)**.** *Suppose that a_j, b_j, c_j, $j = 0, 1, \ldots, n$ are positive real numbers such that $b_j > a_j + c_j$, and suppose that u_j are real numbers such that*

$$-a_j u_{j-1} + b_j u_j - c_j u_{j+1} \leq 0, \qquad j = 1, 2, \ldots n-1$$

Then $u_j \leq K$, $j = 0, 1, \ldots, n$ where $K = \max\{u_0, u_n, 0\}$

Proof. Let $u_m = \max\{u_0, u_1, \ldots u_n\}$. If $m = 0$, $m = n$, or $u_m < 0$, we're done. So $u_m \geq u_{m-1}$ and $u_m \geq u_{m+1}$. Hence

$$\begin{aligned} b_m u_m &\leq a_m u_{m-1} + c_m u_{m+1} \\ &\leq a_m u_m + c_m u_m \qquad (*) \\ &\leq b_m u_m \end{aligned}$$

since $u_m > 0$. Hence $u_{m-1} = u_m = u_{m+1}$ (Note that (*) can only give equality if $u_{m-1} = u_m = u_{m+1}$). Now apply the same logic to u_{m-1} and u_{m+1}, repeat until $u_m = u_n$ or $u_m = u_0$. □

It's going to be helpful to work with the operator:

$$L(u)_j = -\frac{\delta_h^2 u_j}{h^2} + r_j u_j.$$

$$L(y)_j = f_j + T_j$$
$$L(Y)_j = f_j$$

So $L(e)_j = T_j$

Essentially, estimating the global error amounts to establishing a bound on the inverse of L. Note that $r_j \geq 0$, so L satisfies the conditions for the maximum principle.

Theorem 4.3.4. *Suppose that y is the solution of (4.6) and $y \in C^{(4)}([a,b])$ and that Y_j is the solution of (4.8). Then*

$$\max_{0\leq j\leq n} |y(x_j) - Y_j| \leq \frac{1}{96}h^2(b-a)^2 M_4$$

Proof. Define

$$\phi_j = C((2j-n)^2 - n^2)h^2 \qquad j = 0, 1, \ldots, n$$

where C is a constant.

$$\begin{aligned} L(\phi_j) &= -C((2j-2-n)^2 - 2(2j-n)^2 + (2j+2-n)^2) + r_j\phi_j \\ &= -8C + r_j\phi_j \end{aligned}$$

Hence $L(e_j + \phi_j) = T_j - 8C + r_j\phi_j$. Now choose $C = T/8$, so $L(e_j + \phi_j) \leq 0$. Now, $e_0 + \phi_0 = e_n + \phi_n = 0$, so by the maximum principle, $e_j + \phi_j \leq 0$ for $j = 0, 1, \ldots n$. Hence

$$e_j \leq \phi_k \leq Cn^2h^2 = \frac{1}{8}(b-a)^2T = \frac{1}{96}h^2(b-a)^2M_4$$

Finish things off by applying the same argument to $L(-e_j + \phi_j)$. □

The boundary conditions $y(a) = A$ and $y(b) = B$, where we specify the value of the solution are called *Dirichlet* boundary conditions. Sometimes we want to specify a boundary condition involving a derivative instead, e.g.

$$y'(a) = A$$

or

$$y'(a) - \alpha y(a) = A. \tag{4.9}$$

The first of these, just involving the derivative is known as a *Neumann* boundary condition. The second, which involves both the derivative and the value (provided that $\alpha \neq 0$ is called a *Robin* boundary condition.

Example 4.3.5 (The heat equation)**.** *The temperature of a bar can be described using the heat equation,*

$$u_t = ku_{xx} + f(t,x), \quad x \in [a,b], \quad t \geq 0.$$

where $k > 0$ is the thermal diffusivity *and $f(t,x)$ describes some heat source along the length of the bar. If the end-points of the bar are at a and b, then the Dirichlet conditions, $u(a) = A(t)$ and $u(b) = B(t)$ describe a situation where the ends of the bar are fixed in such a way that they vary with time (in a manner that is known). If A and B are constants, then the ends of the bar are held at some fixed temperature.*

If $f_t = A_t = B_t = 0$ and the bar is left, its temperature will eventually reach a steady state, described by

$$u''(x) = -\frac{f(x)}{k}, \qquad u(a) = A, \quad u(b) = B,$$

since intuitively we can see that for large t

$$u_t \to 0 \quad as \quad t \to \infty.$$

Suppose that rather than holding one end of the bar at a constant temperature, we insulate it instead, to mean that there is no heat flux through one end of the bar. Suppose that we decide to insulate the end at $x = a$. To model this, we use the Neumann boundary condition, given by

$$u_x(a,t) = 0, \tag{4.10}$$

and this would normally be combined with an initial condition such as

$$u(x,0) = h(x).$$

So we now have a derivative boundary condition at one of the boundaries, and if we want to solve the system numerically this derivative boundary condition would need to be modelled somehow.

Holding the bar at a constant temperature requires a perfectly conducting heat reservoir, i.e. we need do be able to get heat energy into and out of the bar at an infinitely high rate. In practice, this isn't possible. The Robin boundary condition is used to model a situation where this heat flux *is dependent on the temperature of the bar, e.g.*

$$u_x(a,t) = \alpha(A - u(a,t)), \tag{4.11}$$

which describes a situation where the heat flux is proportional to the difference between the temperature of the bar and a constant, A.

We could use the central difference approach to model the Robin condition, (4.11) as

$$\frac{\delta_h u(a,t)}{h} + \alpha u(a,t) = \alpha A,$$

which would lead to the discretisation

$$(U_{1/2} + U_{-1/2})/h + \alpha U_0 = \alpha A.$$

Now unfortunately, this involves values of Y that we don't know anything about. However instead, we use

$$\frac{\delta_{2h} u(a,t)}{2h} + \alpha u(a,t) = \alpha A$$

which leads to

$$\frac{U_1 - U_{-1}}{2h} + \alpha U_0 = \alpha A. \tag{4.12}$$

When we considered the Dirichlet problem, U_0 was prescribed, but now it is an unknown,

so if we were solving (4.6), we could use (4.8) to obtain

$$-\frac{U_1 - 2U_0 + U_{-1}}{h^2} + r_0 U_0 = f_0. \tag{4.13}$$

We would now have $n+1$ equations in $n+1$ unknowns. Alternatively, we could combine (4.12) and (4.13) to eliminate Y_{-1}:

$$\left(\frac{2(1-\alpha h)}{h^2} + r_0\right) U_0 - \frac{2}{h^2} U_1 = f_0 + \frac{2}{h} A.$$

In either case, we are, again, left with a tridiagonal system. With certain conditions on r and α, the system is straightforward to solve and an error analysis is also possible.

So far we have just considered linear ODEs. Lets look at what would happen with a non-linear ODE.

Example 4.3.6. *The brachistochrone problem is to find the path between 2 points such that a point gliding under the force of gravity travels between the 2 points in the shortest time.*

Solutions of the brachistochrone problem are given by the ODE

$$2yy'' + y'^2 + 1 = 0 \tag{4.14}$$

Where y is height below the starting point. For this example, lets suppose that we start at $(0,0)$ and what to travel to $(1,-1)$. So, our boundary conditions are $y(0) = 0$, and $y(1) = 1$.

If we attempted to discretise this using central differencing, we'd get:

$$2Y_i \frac{(Y_{i-1} - 2Y_i + Y_{i+1})}{h^2} + \frac{(Y_1 - Y_{-1})^2}{4h^2} + 1 = 0 \qquad i = 1, \ldots n-1$$

This is not a linear equation in the Y_i, so we can't just use matrix algebra to solve it. In fact, there are things that can be done with central differencing, but lets look at a

different approach.

Matlab example ...

In the MATLAB example, we defined a family of solutions, $y_S(x)$ that solve (4.14) with the initial conditions $y_S(0) = 0$, $y'_S(0) = S$. We then used a root-finding algorithm to find S such that $y_S(1) = 1$. This approach is called a *shooting method*.

Chapter 5

Numerical Integration / Quadrature

This chapter concerns itself exclusively with the problem of attempting to evalutae a definite intehral of the form

$$\int_a^b f(x)\mathrm{d}x,$$

where f is a continuous real-valued function defined on the interval $[a, b]$. Problems involving integration occur in many fields. However in general, integration is a difficult topic. Most integrals indefinite cannot be evaluated analytically, and thus the answer to any definite counterpart cannot be found exactly. For example, try to persuade Mathematica to evaluate

$$\int \exp(\sin(\cos x))dx$$

analytically. It won't be able to. However, ask Mathematica to evaluate

$$\int_1^{2000} \exp(\sin(\cos x))dx,$$

and mysteriously the answer 1514.780678 will appear. How was this achieved?

The answer of course is that whilst an analytical solution to the indefinite case may be impossible, on many occasions a solution to the definite problem can be achieved using numerical techniques. It is some of these techniques that we will study in this chapter.

In this section, we will see that *polynomial interpolation* can help us to integrate. Polynomial interpolation involves approximating a function $f(x)$ by a suitable n'th order polynomial $p_n(x)$, and since polynomials are easy to integrate the integration of $p_n(x)$ should give us a good approximation to the integral of $f(x)$, i.e.

$$\int_a^b f(x)\mathrm{d}x \approx \int_a^b p_n(x)\mathrm{d}x.$$

When a function is not known (or is too expensive to compute) for every value in its domain, we can still attempt to find an approximation to it. We know that smooth functions are well approximated by the first terms in their Taylor series. Truncating a Taylor series gives a polynomial, so it's natural to try to use polynomials to perform a more general approximation of a function.

5.1 Lagrange interpolation

Suppose that a set of values, y_i are given at a set of distinct real points, $x_0, \ldots, x_n$. The nth degree polynomial, p_n that satisfies $p_n(x_i) = y_i$ is called the *Lagrange interpolation polynomial*. We will see that this polynomial exists and is unique. We will then show that if we take $p_n(x_i) = f(x_i)$, it is possible to derive an error bound for $|p(x) - f(x)|$.

Definition 5.1.1. *We will denote the space of all polynomials of maximum degree n as $\mathcal{P}_n$*

i.e. $\mathcal{P}_2$ contains all polynomials that are of quadratic order or less.

Lemma 5.1.1. *Suppose that $n \geq 1$. There exist polynomials, $L_k \in P_n$ such that*

$$L_k(x_i) = \begin{cases} 1, & i = k \\ 0, & i \neq k \end{cases} \tag{5.1}$$

for all $i, k = 0, \ldots, n$. Moreover the polynomial given by

$$p_n(x) = \sum_{k=0}^{n} L_k(x) y_k, \tag{5.2}$$

satisfies the above interpolation conditions, i.e. $p_n \in \mathcal{P}_n$ and $p_n(x_i) = y_i$.

Proof. For each fixed $k, 0 \leq k \leq n$, L_k is required to have n zeros at the points x_i, $0 \leq i \leq n, i \neq k$; thus L_k is of the form

$$L_k(x) = C_k \prod_{i=0, i\neq k}^{n} (x - x_i),$$

where C_k is some constant to be determined. We know that we require the function should satisfy $L_k(x_k) = 1$, which we can use to find C_k. Using this gives

$$C_k = \prod_{i=0, i\neq k}^{n} \frac{1}{(x_k - x_i)},$$

and thus we have

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

Since the function $p_n(x)$ defined by (5.2) is a linear combination of polynomials $L_k \in \mathcal{P}_n, k = 0, 1, 2, ..., n$, then $p_n \in \mathcal{P}_n$. Finally $p_n(x_i) = y_i, i = 0, 1, 2, ..., n$ is a trivial consequence of using (5.1) in (5.2).

□

Theorem 5.1.2 (Lagrange's Interpolation Theorem)**.** *Assume that $n \geq 0$. Let $x_i, i = 0, \ldots, n$ be distinct real numbers and suppose that y_i, $i = 0, \ldots, n$ are real numbers.*

Then, there exists a unique polynomial, $p_n(x) \in \mathcal{P}_n$ such that

$$p_n(x_i) = y_i, \qquad i = 0, \ldots, n$$

Proof. The existence of such a polynomial follows immediately from the previous lemma, since we have

$$p_n(x) = \sum_{k=0}^{n} L_k(x) y_k,$$

which satisfies the condition that $p_n(x_i) = y_i$ as previously described. Hence we are left to prove that p_n is a unique polynomial in $\mathcal{P}_n$ satisfying this property. We do this via contradiction. Suppose that there exists another distinct polynomial $q_n \in \mathcal{P}_n$ that satisfies $q_n(x_i) = y_i, \quad i = 0, 1, ..., n$. Then $p_n - q_n \in \mathcal{P}_n$ and $p_n - q_n$ must have $n + 1$ distinct roots for all $x_i, \quad i = 0, 1, 2, ..., n$. However since it is a polynomial of degree $\leq n$ it cannot have more than n distinct roots unless it is identically zero. If follows that

$$p_n(x) - q_n(x) \equiv 0,$$

which violates our original assumption that p_n and q_n are distinct. Hence there exists only one polynomial $p_n(x) \in \mathcal{P}_n$ that satisfies (5.1.2).

□

Definition 5.1.2. *Given the data described in Theorem 5.1.2, the polynomial,*

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

is called the Lagrange interpolation polynomial. The numbers, x_i, are the interpolation points.

Definition 5.1.3. *Given a function, $f \in C([a, b])$ and a set of points $x_i \in [a, b]$, $i = 0, \ldots, n$, the polynomial*

$$p_n(x) = \sum_{k=0}^{n} L_k(x) f(x_k)$$

is the Lagrange interpolation polynomial of degree n with interpolation points x_i for the

function, f.

Example 5.1.3. *Consider the function $f : x \mapsto e^x$. We shall construct a Lagrange interpolation polynomial of degree two for this function on the interval $[-1, 1]$ with interpolation points $x_0 = -1, x_1 = 0, x_2 = 1$.*

As $n = 2$ we have

$$L_0(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} = \frac{1}{2}x(x - 1).$$

Similarly we can show that

$$L_1(x) = 1 - x^2, \qquad L_2(x) = \frac{1}{2}x(x + 1).$$

Therefore we have

$$p_2(x) = \frac{1}{2}x(x - 1)e^{-1} + (1 - x^2)e^0 + \frac{1}{2}x(x + 1)e^1,$$

which, after some simplifications gives

$$p_2(x) = 1 + x \sinh 1 + x^2(\cosh 1 - 1). \tag{5.3}$$

An important point to note is that although the values of $p_n(x)$ will concide with $f(x)$ at the points x_i, the values of the interpolation polynomial may differ significantly from the function whenever $x \neq x_i$. The next question that we attempt to answer is: Can we bound or estimate the size of the interpolation error $f(x) - p_n(x)$, assuming that f is sufficiently smooth?

Theorem 5.1.4. *Suppose that p_n is the Lagrange interpolation polynomial for a function*

$f \in C^{n+1}([a,b])$. Then, given any $x \in [a,b]$, there exists $\xi = \xi(x) \in (a,b)$ such that

$$f(x) - p_n(x) = \frac{f^{n+1}(\xi)}{(n+1)!}\pi_{n+1}(x)$$

where $\pi_{n+1}(x) = (x - x_0)\dots(x - x_n)$. Moreover

$$|f(x) - p_n(x)| \leq \frac{M_{n+1}}{(n+1)!}\left|\pi_{n+1}(x)\right|, \tag{5.4}$$

where

$$M_{n+1} = \max_{\zeta \in [a,b]} \left|f^{(n+1)}(\zeta)\right|.$$

Proof. Left as an exercise.

□

5.2 Quadrature Rules

Let $x_i \in [a,b]$ be a set of distinct points. The Lagrange interpolant p_n of f at these points is

$$p_n(x) = \sum_{k=0}^{n} L_k(x) f(x_k),$$

where

$$L_k(x) = \prod_{i=0, i\neq k} \frac{x - x_i}{x_k - x_i},$$

as before. So therefore we have

$$\int_a^b f(x)dx \approx \int_a^b p_n(x)\mathrm{d}x = \int_a^b \sum_{k=0}^{n} L_k(x) f(x_k)\mathrm{d}x = \sum_{k=0}^{n} w_k f(x_k), \tag{5.5}$$

where

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

The values, w_k are referred to as *quadrature weights* and x_i as *quadrature points*. When the x_i are equally spaced, equation (5.5) is known as the *Newton-Cotes formula* of order n.

5.2.1 The Trapezium Rule

When $n = 1$, we obtain the *trapezium rule*. We take $n = 1, x_0 = a, x_1 = b$; The Lagrange interpolation polynomial of degree one is simply

$$\begin{aligned} p_1(x) &= L_0 f(a) + L_1 f(b) \\ &= \frac{x-b}{a-b} f(a) + \frac{x-a}{b-a} f(b) \\ &= \frac{1}{b-a} \left[(b-x) f(a) + (x-a) f(b) \right]. \end{aligned}$$

We now integrate $p_1(x)$ from a to b to yield

$$\int_a^b f(x) \mathrm{d}x \approx \frac{b-a}{2} \left[f(a) + f(b) \right]. \tag{5.6}$$

Equation (5.6) is known as the Trapezium Rule. Its name stems from the fact that expression on the right hand side of the above equation is the area of a trapezium with vertices $(a, 0), (b, 0), (a, f(a)), (b, f(b))$.

5.2.2 Simpson's Rule

We can obtain a more sophisticated quadrature rule by setting $n = 2$. When we do this, we obtain *Simpson's rule*:

We must first define three equally spaced points x_0, x_1, x_2, and thus we have

$$x_0 = a, \quad x_1 = \frac{1}{2}(a+b), \quad x_2 = b$$

In calculating the required quadrature weights we have

$$\begin{aligned} w_0 &= \int_a^b L_0(x)dx \\ &= \int_a^b \frac{(x-(a+b)/2)(x-b)}{(a-(a+b)/2)(a-b)}dx \\ &= \int_{-1}^1 \frac{t(t-1)}{2}\frac{b-a}{2}dt \\ &= \frac{1}{6}(b-a), \end{aligned}$$

where we have made use of the transformation

$$x = \frac{b-a}{2}t + \frac{b+a}{2}.$$

For the second quadrature weight w_1 we have

$$\begin{aligned} w_1 &= \int_a^b L_1(x)dx \\ &= \int_{-1}^1 \frac{(t-1)(t+1)}{-1}\frac{(b-a)}{2}dt \\ &= \left(-\frac{2}{3}+2\right)\frac{(b-a)}{2} = \frac{2}{3}(b-a). \end{aligned}$$

We can see that $w_2 = w_0$ by symmetry. Hence we obtain

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

Note that the weights w depend only on the x_k, not on f, so they can be precomputed. In fact, the weights can be precomputed for a normalised interval $[-1, 1]$.

Next, we define the quadrature error $E_n(f)$ as follows:

$$E_n(f) = \int_a^b f(x)dx - \sum_{k=0}^{n} w_k f(x_k)$$

.

Theorem 5.2.1. *Suppose that $f \in C^{n+1}([a,b])$, then*

$$\left| E_n(f) \leq \frac{M_{n+1}}{(n+1)!} \int_a^b |\pi_{n+1}(x)|\, dx \right|$$

where $M_{n+1} = \|f^{(n+1)}\|_\infty$ and $\pi_{n+1}(x) = (x - x_0) \dots (x - x_n)$

Proof. Left as an exercise. Use Lagrange interpolation error estimate, theorem. 5.1.4 □

We now introduce an example of how to apply this for the trapezium rule:

$$\begin{aligned} |E_1(f)| &\leq \frac{M_2}{2} \int_a^b |(x-a)(x-b)|\, dx \\ &= \frac{M_2}{2} \left(\frac{b-a}{2} \right)^3 \int_{-1}^1 (1-t^2) dt \\ &= \frac{(b-a)^3}{12} M_2 \end{aligned}$$

Similarly, for Simpson's rule, it is possible to show that

$$|E_2| \leq \frac{(b-a)^4}{196} M_3,$$

although in actual fact, the error for Simpson's rule is better than this.

Suppose that we divide the interval $[a,b]$ into m equal subintervals, each of width $h = (b-a)/m$. We can write

$$\int_a^b f(x) dx = \sum_{i=1}^m \int_{x_{i-1}}^{x_i} f(x) dx$$

where $x_i = a + ih$

If we choose to evaluate each of these integrals using the trapezium rule, we obtain the *composite trapezium rule*, defined as follows:

Definition 5.2.1. *The composite trapezium rule is defined as*

$$\int_a^b f(x)dx \approx h\left(\frac{1}{2}f(x_0) + f(x_1) + \cdots + f(x_{m-1}) + \frac{1}{2}f(x_m)\right).$$

Lets try and obtain an expression for the error of the composite trapezium rule:

$$\begin{aligned} E_1(f) &= \int_a^b f(x)dx - h\left(\frac{1}{2}f(x_0) + f(x_1) + \cdots + f(x_{m-1}) + \frac{1}{2}f(x_m)\right) \\ &= \sum_{i=1}^{m}\left[\int_{x_{i-1}}^{x_i} f(x)\mathrm{d}x - \frac{1}{2}h(f(x_{i-1}) + f(x_i))\right]. \end{aligned}$$

So we have

$$\begin{aligned} |E_1(f)| &\leq \frac{h^3}{12}\sum_{i=1}^{m}\max_{\xi\in[x_{i-1},x_i]}\left|f''(\xi)\right| \\ &\leq \frac{(b-a)^3}{12m^3}mM_2 = \frac{(b-a)^3}{12m^2}M_2 \end{aligned}$$

where $M_2 = \|f\|_\infty$

We can do the same kind of thing to generate the composite Simpson rule. Here, we assume that m is even and use Simpson's rule to integrate each interval, $[x_{i-2}, x_i]$.

Definition 5.2.2. *The composite Simpson rule is defined as*

$$\begin{aligned} \int_a^b f(x)dx = \frac{h}{3}\Big(&f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \ldots \\ &+ 2f(x_{m-2} + 4f(x_{m-1}) + f(x_m)\Big). \end{aligned}$$

Define the error for the composite Simpson rule as

$$\begin{aligned} E_2(f) = \int_a^b f(x)dx - \frac{h}{3}\Big(&f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \ldots \\ &+ 2f(x_{m-2}) + 4f(x_{m-1}) + f(x_m)\Big). \end{aligned}$$

Following the same principle as for the composite trapezium rule, we could use the estimate for $|E_2|$ to obtain

$$E_2(f) \leq \frac{(b-a)^4}{196m^3} M_3.$$

The example showed that the Simpson rule might have order $p = 4$. Lets try and prove this:

Theorem 5.2.2. *Suppose that $f \in C^4([a,b])$. Then*

$$\int_a^b f(x)dx - \frac{b-a}{6}(f(a) + 4f((a+b)/2) + f(b)) = -\frac{(b-a)^5}{2880} f^{(4)}(\xi) \tag{5.7}$$

for some $\xi \in (a,b)$

Proof. Make the change of variable, $x = (a+b)/2 + t(b-a)/2$ and write $F(t) = f(x)$. Clearly, $F \in C^4([-1,1])$ and

$$\begin{aligned} &\int_a^b f(x)dx - \frac{b-a}{6}(f(a) + 4f((a+b)/2) + f(b)) \\ &= \frac{b-a}{2} \int_{-1}^1 F(\tau)d\tau - \frac{1}{3}(F(-1) + 4F(0) + F(1)). \end{aligned} \tag{5.8}$$

Now define

$$G(t) = \int_{-t}^t F(\tau)d\tau - \frac{t}{3}(F(-t) + 4F(0) + F(t))$$

So the right hand side of (5.8) is just $\frac{1}{2}(b-a)G(1)$.

The remainder of the proof is devoted to showing that $\frac{1}{2}(b-a)G(1)$ is equal to the right hand side of (5.7).

So, what we're interested in is $G(1)$. Define the auxiliary function,

$$H(t) = G(t) - t^5 G(1)$$

Now, $G(0) = 0$

$G'(t) = F(t) + F(-t) - \frac{1}{3}(F(-t) + 4F(0) + F(t)) - \frac{t}{3}(-F'(-t) + F'(t))$, so $G'(0) = 0$

$G''(t) = F'(t) - F'(-t) - \frac{1}{3}(-F'(-t) + F'(t)) - \frac{1}{3}(-F'(-t) + F'(t)) - \frac{t}{3}(F''(-t) + F''(t))$, so $G''(0) = 0$

Therefore, $H(0) = H(1) = 0$.

$H'(t) = G'(t) + 5t^4 G(1)$, so $H'(0) = G'(0) = 0$

$H''(t) = G''(t) + 20t^3 G(1)$, so $H''(0) = 0$.

This means that we can apply Rolle's theorem to H to see that there exists $\xi_1 \in (0, 1)$ such that $H'(\xi_1) = 0$. Then apply it again to see that there exists $\xi_2 \in (0, \xi_1)$ such that $H''(\xi_2) = 0$ and then again to get $\xi_3 \in (0, \xi_2)$ such that $H'''(\xi_3) = 0$.

Now, $G'''(t) = \frac{1}{3}(F''(t) + F''(-t)) - \frac{1}{3}(F''(-t) + F''(t)) - \frac{t}{3}(-F'''(-t) + F'''(t)) = \frac{t}{3}(-F'''(-t) + F'''(t))$, so

$$0 = H'''(\xi_3) = -\frac{\xi_3}{3}(F'''(\xi_3) - F'''(-\xi_3)) - 60\xi_3^2 G(1)$$

The Mean value theorem means that there exists $\xi_4 \in (-\xi_3, \xi_3)$ such that $F'''(\xi_3) - F'''(-\xi_3) = 2\xi_3 F^{(4)}(\xi_4)$.

Hence

$$\begin{aligned} 0 &= -\frac{2\xi_3^2}{3} F^{(4)}(\xi_4) - 60\xi_3^2 G(1) \\ \Rightarrow G(1) &= -\frac{1}{90} F^{(4)}(\xi_4) \\ &= \frac{(b-a)^4}{90 \cdot 2^4} f^{(4)}(\xi_4). \end{aligned}$$

Hence, we finally have the result. □

Using this result and using the composite Simpson rule gives:

Corollary 5.2.3.

$$|\mathcal{E}_2(f)| \leq \frac{(b-a)^5}{2880m^4} M_4.$$

5.2.3 The Mid-point rule

A slightly different quadrature rule (derived using a different type of interpolation) is the *mid-point rule*, defined as

$$\int_a^b f(x)\mathrm{d}x \approx (b-a) f\left(\frac{a+b}{2}\right), \tag{5.9}$$

and the subsequent composite midpoint rule is given by

$$\int_a^b f(x)\mathrm{d}x \approx h \sum_{i=0}^{n-1} f\left(\frac{x_i + x_{i+1}}{2}\right). \tag{5.10}$$

As an exercise, try and show that

$$\int_a^b f(x)\mathrm{d}x - h \sum_{i=0}^{n-1} f\left(\frac{x_i + x_{i+1}}{2}\right) = \mathrm{O}(h^2).$$