MAT 460: Numerical Analysis I

James V. Lambers

October 29, 2013

Contents

Chapter 1

Mathematical Preliminaries and Error Analysis

1.1 Introduction

This course is the first in a two-course sequence on *numerical analysis*, the study of numerical algorithms for solving mathematical problems that arise in science and engineering. These numerical algorithms differ from the analytical methods that are presented in other mathematics courses in that they rely exclusively on the four basic arithmetic operations, addition, subtraction, multiplication and division, so that they can be implemented on a computer.

1.1.1 Error Analysis

First, we will discuss the most fundamental concepts of numerical analysis, which pertain to *error analysis*. Numerical algorithms must not only be *efficient*, but they must also be *accurate*, and *robust*. In other words, the solutions they produce are at best *approximate* solutions because an exact solution cannot be computed by analytical techniques. Furthermore, these computed solutions should not be too sensitive to the input data, because if they are, any error in the input can result in a solution that is essentially useless. Such error can arise from many sources, such as

- neglecting components of a mathematial model or making simplifying assumptions in the model,
- discretization error, which arises from approximating continuous functions by sets of discrete data points,

- convergence error, which arises from truncating a sequence of approximations that is meant to converge to the exact solution, to make computation possible, and
- roundoff error, which is due to the fact that computers represent real numbers approximately, in a fixed amount of storage in memory.

We will see that in some cases, these errors can be surprisingly large, so one must be careful when designing and implementing numerical algorithms.

1.1.2 Nonlinear Equations

The vast majority of equations, especially nonlinear equations, cannot be solved using analytical techniques such as algebraic manipulations or knowledge of trignometric functions. Therefore, *iterative* methods must instead be used to obtain an approximate solution. We will study a variety of such methods, which have distinct advantages and disadvantages. For example, some methods are guaranteed to produce a solution under reasonable assumptions, but they might do so slowly. On the other hand, other methods may produce a sequence of iterates that quickly converges to the solution, but may be unreliable for some problems.

1.1.3 Polynomial Interpolation and Approximation

Polynomials are among the easiest functions to work with, because it is possible to evaluate them, as well as perform operations from calculus on them, with great efficiency. For this reason, more complicated functions, or functions that are represented only by values on a discrete set of points in their domain, are often approximated by polynomials.

Such an approximation can be computed in various ways. We first consider *interpolation*, in which we construct a polynomial that agrees with the given data at selected points. While interpolation methods are efficient, they must be used carefully, because it is not necessarily true that a polynomial that agrees with a given function at certain points is a good approximation to the function elsewhere in its domain.

One remedy for this is to use *piecewise* polynomial interpolation, in which a low-degree polynomial, typically linear or cubic, is used to approximate data only on a given subdomain, and these polynomial "pieces" are "glued" together to obtain a piecewise polynomial approximation. This approach is also efficient, and tends to be more robust than standard polynomial interpolation, but there are disadvantages, such as the fact that a piecewise polynomial only has very few derivatives.

1.1.4 Numerical Differentiation and Integration

It is often necessary to approximate derivatives or integrals of functions that are represented only by values at a discrete set of points, thus making differentiation or integration rules impossible to use directly. Numerical techniques for these operations make use of polynomial interpolation by (implicitly) constructing a polynomial interpolant that fits the given data, and then applying differentiation or integration rules to the polynomial. We will see that by choosing the method of polynomial approximation judiciously, accurate results can be obtain with far greater accuracy and efficiency than one might expect.

1.2 Review of Calculus

Among the mathematical problems that can be solved using techniques from numerical analysis are the basic problems of differential and integral calculus:

- computing the instantaneous rate of change of one quantity with respect to another, which is a *derivative*, and
- computing the total change in a function over some portion of its domain, which is a *definite integral*.

Calculus also plays an essential role in the development and analysis of techniques used in numerical analysis, including those techniques that are applied to problems not arising directly from calculus. Therefore, it is appropriate to review some basic concepts from calculus before we begin our study of numerical analysis.

1.2.1 Limits

The basic problems of differential and integral calculus described in the previous paragraph can be solved by computing a sequence of approximations to the desired quantity and then determining what value, if any, the sequence of approximations approaches. This value is called a *limit* of the sequence. As a sequence is a function, we begin by defining, precisely, the concept of the limit of a function.

Definition *We write*

$$\lim_{x \to a} f(x) = L$$

if for any open interval I_1 containing L, there is some open interval I_2 containing a such that $f(x) \in I_1$ whenever $x \in I_2$, and $x \neq a$. We say that L is the **limit of** $f(x)$ **as** x **approaches** a.

We write

$$\lim_{x \to a^-} f(x) = L$$

if, for any open interval I_1 containing L, there is an open interval I_2 of the form (c, a), where $c < a$, such that $f(x) \in I_1$ whenever $x \in I_2$. We say that L is the **limit of $f(x)$ as x approaches a from the left**, *or the* **left-hand limit of $f(x)$ as x approaches a.**

Similarly, we write

$$\lim_{x \to a^+} f(x) = L$$

if, for any open interval I_1 containing L, there is an open interval I_2 of the form (a, c), where $c > a$, such that $f(x) \in I_1$ whenever $x \in I_2$. We say that L is the **limit of $f(x)$ as x approaches a from the right**, *or the* **right-hand limit of $f(x)$ as x approaches a.**

We can make the definition of a limit a little more concrete by imposing sizes on the intervals I_1 and I_2, as long as the interval I_1 can still be of arbitrary size. It can be shown that the following definition is equivalent to the previous one.

Definition *We write*

$$\lim_{x \to a} f(x) = L$$

if, for any $\epsilon > 0$, there exists a number $\delta > 0$ such that $|f(x) - L| < \epsilon$ whenever $0 < |x - a| < \delta$.

Similar definitions can be given for the left-hand and right-hand limits.

Note that in either definition, the point $x = a$ is specifically excluded from consideration when requiring that $f(x)$ be close to L whenever x is close to a. This is because the concept of a limit is only intended to describe the behavior of $f(x)$ *near* $x = a$, as opposed to its behavior *at* $x = a$. Later in this lecture we discuss the case where the two distinct behaviors coincide.

1.2.2 Limits at Infinity

The concept of a limit defined above is useful for describing the behavior of a function $f(x)$ as x approaches a *finite* value a. However, suppose that the function f is a *sequence*, which is a function that maps $\mathbb{N}$, the set of natural numbers, to $\mathbb{R}$, the set of real numbers. We will denote such a sequence by $\{f_n\}_{n=0}^{\infty}$, or simply $\{f_n\}$. In numerical analysis, it is sometimes necessary to determine the value that the terms of a sequence $\{f_n\}$ approach as $n \to \infty$. Such a value, if it exists, is not a limit, as defined previously. However, it is

natural to use the notation of limits to describe this behavior of a function. We therefore define what it means for a sequence $\{f_n\}$ to have a limit as n becomes infinite.

Definition *(Limit at Infinity) Let $\{f_n\}$ be a sequence defined for all integers not less than some integer n_0. We say that the* **limit of $\{f_n\}$ as n approaches ∞** *is equal to L, and write*

$$\lim_{n\to\infty} f_n = L,$$

if for any open interval I containing L, there exists a number M such that $f_n \in I$ whenever $x > M$.

Example Let the sequence $\{f_n\}_{n=1}^{\infty}$ be defined by $f_n = 1/n$ for every positive integer n. Then

$$\lim_{n\to\infty} f_n = 0,$$

since for any $\epsilon > 0$, no matter how small, we can find a positive integer n_0 such that $|f_n| < \epsilon$ for all $n \geq n_0$. In fact, for any given ϵ, we can choose $n_0 = \lceil 1/\epsilon \rceil$, where $\lceil x \rceil$, known as the *ceiling function*, denotes the smallest integer that is greater than or equal to x. □

1.2.3 Continuity

In many cases, the limit of a function $f(x)$ as x approached a could be obtained by simply computing $f(a)$. Intuitively, this indicates that f has to have a graph that is one continuous curve, because any "break" or "jump" in the graph at $x = a$ is caused by f approaching one value as x approaches a, only to actually assume a different value at a. This leads to the following precise definition of what it means for a function to be continuous at a given point.

Definition *(Continuity) We say that a function f is* **continuous at** *a if*

$$\lim_{x\to a} f(x) = f(a).$$

We also say that $f(x)$ has the **Direct Subsitution Property** *at $x = a$.*

We say that a function f is **continuous from the right** *at a if*

$$\lim_{x\to a^+} f(x) = f(a).$$

Similarly, we say that f is **continuous from the left** *at a if*

$$\lim_{x\to a^-} f(x) = f(a).$$

The preceding definition describes continuity at a single point. In describing where a function is continuous, the concept of continuity over an interval is useful, so we define this concept as well.

Definition *(Continuity on an Interval) We say that a function f is* **continuous on the interval** (a, b) *if f is continuous at every point in (a, b). Similarly, we say that f is continuous on*

1. $[a, b)$ *if f is continuous on (a, b), and continuous from the right at a.*
2. $(a, b]$ *if f is continuous on (a, b), and continuous from the left at b.*
3. $[a, b]$ *if f is continuous on (a, b), continuous from the right at a, and continuous from the left at b.*

In numerical analysis, it is often necessary to construct a continuous function, such as a polynomial, based on data obtained by measurements and problem-dependent constraints. In this course, we will learn some of the most basic techniques for constructing such continuous functions by a process called *interpolation*.

1.2.4 The Intermediate Value Theorem

Suppose that a function f is continuous on some closed interval $[a, b]$. The graph of such a function is a continuous curve connecting the points $(a, f(a))$ with $(b, f(b))$. If one were to draw such a graph, their pen would not leave the paper in the process, and therefore it would be impossible to "avoid" any y-value between $f(a)$ and $f(b)$. This leads to the following statement about such continuous functions.

Theorem *(Intermediate Value Theorem) Let f be continuous on $[a, b]$. Then, on (a, b), f assumes every value between $f(a)$ and $f(b)$; that is, for any value y between $f(a)$ and $f(b)$, $f(c) = y$ for some c in (a, b).*

The Intermediate Value Theorem has a very important application in the problem of finding solutions of a general equation of the form $f(x) = 0$, where x is the solution we wish to compute and f is a given continuous function. Often, methods for solving such an equation try to identify an interval $[a, b]$ where $f(a) > 0$ and $f(b) < 0$, or vice versa. In either case, the Intermediate Value Theorem states that f must assume every value between $f(a)$ and $f(b)$, and since 0 is one such value, it follows that the equation $f(x) = 0$ must have a solution somewhere in the interval (a, b).

We can find an approximation to this solution using a procedure called *bisection*, which repeatedly applies the Intermediate Value Theorem to smaller and smaller intervals that contain the solution. We will study bisection, and other methods for solving the equation $f(x) = 0$, in this course.

1.2.5 Derivatives

The basic problem of differential calculus is computing the instantaneous rate of change of one quantity y with respect to another quantity x. For example, y may represent the position of an object and x may represent time, in which case the instantaneous rate of change of y with respect to x is interpreted as the velocity of the object.

When the two quantities x and y are related by an equation of the form $y = f(x)$, it is certainly convenient to describe the rate of change of y with respect to x in terms of the function f. Because the instantaneous rate of change is so commonplace, it is practical to assign a concise name and notation to it, which we do now.

Definition *(Derivative) The* **derivative** *of a function $f(x)$ at $x = a$, denoted by $f'(a)$, is*

$$f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h},$$

provided that the above limit exists. When this limit exists, we say that f is **differentiable** *at a.*

Remark Given a function $f(x)$ that is differentiable at $x = a$, the following numbers are all equal:

- the derivative of f at $x = a$, $f'(a)$,
- the slope of the tangent line of f at the point $(a, f(a))$, and
- the instantaneous rate of change of $y = f(x)$ with respect to x at $x = a$.

This can be seen from the fact that all three numbers are defined in the same way. □

Many functions can be differentiated using differentiation rules such as those learned in a calculus course. However, many functions cannot be differentiated using these rules. For example, we may need to compute the instantaneous rate of change of a quantity $y = f(x)$ with respect to another quantity x, where our only knowledge of the function f that relates x and

y is a set of pairs of x-values and y-values that may be obtained using measurements. In this course we will learn how to approximate the derivative of such a function using this limited information. The most common methods involve constructing a continuous function, such as a polynomial, based on the given data, using interpolation. The polynomial can then be differentiated using differentiation rules. Since the polynomial is an approximation to the function $f(x)$, its derivative is an approximation to $f'(x)$.

1.2.6 Differentiability and Continuity

Consider a tangent line of a function f at a point $(a, f(a))$. When we consider that this tangent line is the limit of secant lines that can cross the graph of f at points on *either side* of a, it seems impossible that f can fail to be continuous at a. The following result confirms this: a function that is differentiable at a given point (and therefore has a tangent line at that point) *must* also be continuous at that point.

Theorem *If f is differentiable at a, then f is continuous at a.*

It is important to keep in mind, however, that the *converse* of the above statement, "if f is continuous, then f is differentiable", is not true. It is actually very easy to find examples of functions that are continuous at a point, but fail to be differentiable at that point. As an extreme example, it is known that there is a function that is continuous everywhere, but is differentiable *nowhere*.

Example The functions $f(x) = |x|$ and $g(x) = x^{1/3}$ are examples of functions that are continuous for all x, but are not differentiable at $x = 0$. The graph of the *absolute value function* $|x|$ has a sharp corner at $x = 0$, since the one-sided limits

$$\lim_{h\to 0^-} \frac{f(h) - f(0)}{h} = -1, \quad \lim_{h\to 0^+} \frac{f(h) - f(0)}{h} = 1$$

do not agree, but in general these limits must agree in order for $f(x)$ to have a derivative at $x = 0$.

The cube root function $g(x) = x^{1/3}$ is not differentiable at $x = 0$ because the tangent line to the graph at the point $(0, 0)$ is vertical, so it has no finite slope. We can also see that the derivative does not exist at this point by noting that the function $g'(x) = (1/3)x^{-2/3}$ has a vertical asymptote at $x = 0$. □

1.2.7 Riemann Sums and the Definite Integral

There are many cases in which some quantity is defined to be the product of two other quantities. For example, a rectangle of width w has uniform height h, and the area A of the rectangle is given by the formula $A = wh$. Unfortunately, in many applications, we cannot necessarily assume that certain quantities such as height are constant, and therefore formulas such as $A = wh$ cannot be used directly. However, they can be used indirectly to solve more general problems by employing the notation known as *integral calculus*.

Suppose we wish to compute the area of a shape that is not a rectangle. To simplify the discussion, we assume that the shape is bounded by the vertical lines $x = a$ and $x = b$, the x-axis, and the curve defined by some continuous function $y = f(x)$, where $f(x) \geq 0$ for $a \leq x \leq b$. Then, we can approximate this shape by n rectangles that have width $\Delta x = (b - a)/n$ and height $f(x_i)$, where $x_i = a + i\Delta x$, for $i = 0, \ldots, n$. We obtain the approximation

$$A \approx A_n = \sum_{i=1}^{n} f(x_i)\Delta x.$$

Intuitively, we can conclude that as $n \to \infty$, the approximate area A_n will converge to the exact area of the given region. This can be seen by observing that as n increases, the n rectangles defined above comprise a more accurate approximation of the region.

More generally, suppose that for each $n = 1, 2, \ldots$, we define the quantity R_n by choosing points $a = x_0 < x_1 < \cdots < x_n = b$, and computing the sum

$$R_n = \sum_{i=1}^{n} f(x_i^*)\Delta x_i, \quad \Delta x_i = x_i - x_{i-1}, \quad x_{i-1} \leq x_i^* \leq x_i.$$

The sum that defines R_n is known as a *Riemann sum*. Note that the interval $[a, b]$ need not be divided into subintervals of equal width, and that $f(x)$ can be evaluated at *arbitrary* points belonging to each subinterval.

If $f(x) \geq 0$ on $[a, b]$, then R_n converges to the area under the curve $y = f(x)$ as $n \to \infty$, provided that the widths of all of the subintervals $[x_{i-1}, x_i]$, for $i = 1, \ldots, n$, approach zero. This behavior is ensured if we require that

$$\lim_{n\to\infty} \delta(n) = 0, \quad \text{where} \quad \delta(n) = \max_{1\leq i\leq n} \Delta x_i.$$

This condition is necessary because if it does not hold, then, as $n \to \infty$, the region formed by the n rectangles will not converge to the region whose area

we wish to compute. If f assumes negative values on $[a, b]$, then, under the same conditions on the widths of the subintervals, R_n converges to the *net* area between the graph of f and the x-axis, where area below the x-axis is counted negatively.

We define the *definite integral* of $f(x)$ from a to b by

$$\int_a^b f(x)\,dx = \lim_{n\to\infty} R_n,$$

where the sequence of Riemann sums $\{R_n\}_{n=1}^{\infty}$ is defined so that $\delta(n) \to 0$ as $n \to \infty$, as in the previous discussion. The function $f(x)$ is called the *integrand*, and the values a and b are the *lower* and *upper limits of integration*, respectively. The process of computing an integral is called *integration*.

In this course, we will study the problem of computing an approximation to the definite integral of a given function $f(x)$ over an interval $[a, b]$. We will learn a number of techniques for computing such an approximation, and all of these techniques involve the computation of an appropriate Riemann sum.

1.2.8 Extreme Values

In many applications, it is necessary to determine where a given function attains its minimum or maximum value. For example, a business wishes to maximize profit, so it can construct a function that relates its profit to variables such as payroll or maintenance costs. We now consider the basic problem of finding a maximum or minimum value of a general function $f(x)$ that depends on a single independent variable x. First, we must precisely define what it means for a function to *have* a maximum or minimum value.

Definition *(Absolute extrema) A function f has a* **absolute maximum** *or* **global maximum** *at c if $f(c) \geq f(x)$ for all x in the domain of f. The number $f(c)$ is called the* **maximum value** *of f on its domain. Similarly, f has a* **absolute minimum** *or* **global minimum** *at c if $f(c) \leq f(x)$ for all x in the domain of f. The number $f(c)$ is then called the* **minimum value** *of f on its domain. The maximum and minimum values of f are called the* **extreme values** *of f, and the absolute maximum and minimum are each called an* **extremum** *of f.*

Before computing the maximum or minimum value of a function, it is natural to ask whether it is possible to determine in advance whether a function even has a maximum or minimum, so that effort is not wasted in trying to

solve a problem that has no solution. The following result is very helpful in answering this question.

Theorem *(Extreme Value Theorem) If f is continuous on $[a, b]$, then f has an absolute maximum and an absolute minimum on $[a, b]$.*

Now that we can easily determine whether a function has a maximum or minimum on a closed interval $[a, b]$, we can develop an method for actually finding them. It turns out that it is easier to find points at which f attains a maximum or minimum value in a "local" sense, rather than a "global" sense. In other words, we can best find the absolute maximum or minimum of f by finding points at which f achieves a maximum or minimum with respect to "nearby" points, and then determine which of these points is the absolute maximum or minimum. The following definition makes this notion precise.

Definition *(Local extrema) A function f has a* **local maximum** *at c if $f(c) \geq f(x)$ for all x in an open interval containing c. Similarly, f has a* **local minimum** *at c if $f(c) \leq f(x)$ for all x in an open interval containing c. A local maximum or local minimum is also called a* **local extremum**.

At each point at which f has a local maximum, the function either has a horizontal tangent line, or no tangent line due to not being differentiable. It turns out that this is true in general, and a similar statement applies to local minima. To state the formal result, we first introduce the following definition, which will also be useful when describing a method for finding local extrema.

Definition*(Critical Number) A number c in the domain of a function f is a* **critical number** *of f if $f'(c) = 0$ or $f'(c)$ does not exist.*

The following result describes the relationship between critical numbers and local extrema.

Theorem *(Fermat's Theorem) If f has a local minimum or local maximum at c, then c is a critical number of f; that is, either $f'(c) = 0$ or $f'(c)$ does not exist.*

This theorem suggests that the maximum or minimum value of a function $f(x)$ can be found by solving the equation $f'(x) = 0$. As mentioned previously, we will be learning techniques for solving such equations in this course. These techniques play an essential role in the solution of problems in which one must compute the maximum or minimum value of a function, subject to

constraints on its variables. Such problems are called *optimization problems*. Although we will not discuss optimization problems in this course, we will learn about some of the building blocks of methods for solving these very important problems.

1.2.9 The Mean Value Theorem

While the derivative describes the behavior of a function at a point, we often need to understand how the derivative influences a function's behavior on an interval. This understanding is essential in numerical analysis because, it is often necessary to approximate a function $f(x)$ by a function $g(x)$ using knowledge of $f(x)$ and its derivatives at various points. It is therefore natural to ask how well $g(x)$ approximates $f(x)$ away from these points.

The following result, a consequence of Fermat's Theorem, gives limited insight into the relationship between the behavior of a function on an interval and the value of its derivative at a point.

Theorem *(Rolle's Theorem) If f is continuous on a closed interval $[a, b]$ and is differentiable on the open interval (a, b), and if $f(a) = f(b)$, then $f'(c) = 0$ for some number c in (a, b).*

By applying Rolle's Theorem to a function f, then to its derivative f', its second derivative f'', and so on, we obtain the following more general result, which will be useful in analyzing the accuracy of methods for approximating functions by polynomials.

Theorem *(Generalized Rolle's Theorem) Let $x_0, x_1, x_2, \ldots, x_n$ be distinct points in an interval $[a, b]$. If f is n times differentiable on (a, b), and if $f(x_i) = 0$ for $i = 0, 1, 2, \ldots, n$, then $f^{(n)}(c) = 0$ for some number c in (a, b).*

A more fundamental consequence of Rolle's Theorem is the Mean Value Theorem itself, which we now state.

Theorem *(Mean Value Theorem) If f is continuous on a closed interval $[a, b]$ and is differentiable on the open interval (a, b), then*

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

for some number c in (a, b).

Remark The expression

$$\frac{f(b) - f(a)}{b - a}$$

is the slope of the secant line passing through the points $(a, f(a))$ and $(b, f(b))$. The Mean Value Theorem therefore states that under the given assumptions, the slope of this secant line is equal to the slope of the tangent line of f at the point $(c, f(c))$, where $c \in (a, b)$. $\square$

The Mean Value Theorem has the following practical interpretation: the average rate of change of $y = f(x)$ with respect to x on an interval $[a, b]$ is equal to the instantaneous rate of change y with respect to x at some point in (a, b).

1.2.10 The Mean Value Theorem for Integrals

Suppose that $f(x)$ is a continuous function on an interval $[a, b]$. Then, by the Fundamental Theorem of Calculus, $f(x)$ has an antiderivative $F(x)$ defined on $[a, b]$ such that $F'(x) = f(x)$. If we apply the Mean Value Theorem to $F(x)$, we obtain the following relationship between the integral of f over $[a, b]$ and the value of f at a point in (a, b).

Theorem *(Mean Value Theorem for Integrals) If f is continuous on $[a, b]$, then*

$$\int_a^b f(x)\,dx = f(c)(b - a)$$

for some c in (a, b).

In other words, f assumes its average value over $[a, b]$, defined by

$$f_{ave} = \frac{1}{b - a}\int_a^b f(x)\,dx,$$

at some point in $[a, b]$, just as the Mean Value Theorem states that the derivative of a function assumes its average value over an interval at some point in the interval.

The Mean Value Theorem for Integrals is also a special case of the following more general result.

Theorem *(Weighted Mean Value Theorem for Integrals) If f is continuous on $[a, b]$, and g is a function that is integrable on $[a, b]$ and does not change sign on $[a, b]$, then*

$$\int_a^b f(x)g(x)\,dx = f(c)\int_a^b g(x)\,dx$$

for some c in (a, b).

In the case where $g(x)$ is a function that is easy to antidifferentiate and $f(x)$ is not, this theorem can be used to obtain an estimate of the integral of $f(x)g(x)$ over an interval.

Example Let $f(x)$ be continuous on the interval $[a, b]$. Then, for any $x \in [a, b]$, by the Weighted Mean Value Theorem for Integrals, we have

$$\int_a^x f(s)(s-a)\,ds = f(c)\int_a^x (s-a)\,ds = f(c)\,\frac{(s-a)^2}{2}\bigg|_a^x = f(c)\frac{(x-a)^2}{2},$$

where $a < c < x$. It is important to note that we can apply the Weighted Mean Value Theorem because the function $g(x) = (x-a)$ does not change sign on $[a, b]$. □

1.2.11 Taylor's Theorem

In many cases, it is useful to approximate a given function $f(x)$ by a polynomial, because one can work much more easily with polynomials than with other types of functions. As such, it is necessary to have some insight into the accuracy of such an approximation. The following theorem, which is a consequence of the Weighted Mean Value Theorem for Integrals, provides this insight.

Theorem *(Taylor's Theorem) Let f be n times continuously differentiable on an interval $[a, b]$, and suppose that $f^{(n+1)}$ exists on $[a, b]$. Let $x_0 \in [a, b]$. Then, for any point $x \in [a, b]$,*

$$f(x) = P_n(x) + R_n(x),$$

where

$$\begin{aligned} P_n(x) &= \sum_{j=0}^{n} \frac{f^{(j)}(x_0)}{j!}(x-x_0)^j \\ &= f(x_0) + f'(x_0)(x-x_0) + \frac{1}{2}f''(x_0)(x-x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x-x_0)^n \end{aligned}$$

and

$$R_n(x) = \int_{x_0}^{x} \frac{f^{(n+1)}(s)}{n!}(x-s)^n\,ds = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x-x_0)^{n+1},$$

where $\xi(x)$ is between x_0 and x.

The polynomial $P_n(x)$ is the nth *Taylor polynomial* of f with *center* x_0, and the expression $R_n(x)$ is called the *Taylor remainder* of $P_n(x)$. When the center x_0 is zero, the nth Taylor polynomial is also known as the nth *Maclaurin polynomial*.

The final form of the remainder is obtained by applying the Mean Value Theorem for Integrals to the integral form. As $P_n(x)$ can be used to approximate $f(x)$, the remainder $R_n(x)$ is also referred to as the *truncation error* of $P_n(x)$. The accuracy of the approximation on an interval can be analyzed by using techniques for finding the extreme values of functions to bound the $(n+1)$-st derivative on the interval.

Because approximation of functions by polynomials is employed in the development and analysis of many techniques in numerical analysis, the usefulness of Taylor's Theorem cannot be overstated. In fact, it can be said that Taylor's Theorem is the Fundamental Theorem of Numerical Analysis, just as the theorem describing inverse relationship between derivatives and integrals is called the Fundamental Theorem of Calculus.

We conclude our discussion of Taylor's Theorem with some examples that illustrate how the nth-degree Taylor polynomial $P_n(x)$ and the remainder $R_n(x)$ can be computed for a given function $f(x)$.

Example If we set $n = 1$ in Taylor's Theorem, then we have

$$f(x) = P_1(x) + R_1(x)$$

where

$$P_1(x) = f(x_0) + f'(x_0)(x - x_0).$$

This polynomial is a linear function that describes the tangent line to the graph of f at the point $(x_0, f(x_0))$.

If we set $n = 0$ in the theorem, then we obtain

$$f(x) = P_0(x) + R_0(x),$$

where

$$P_0(x) = f(x_0)$$

and

$$R_0(x) = f'(\xi(x))(x - x_0),$$

where $\xi(x)$ lies between x_0 and x. If we use the integral form of the remainder,

$$R_n(x) = \int_{x_0}^{x} \frac{f^{(n+1)}(s)}{n!}(x - s)^n \, ds,$$

then we have

$$f(x) = f(x_0) + \int_{x_0}^{x} f'(s)\, ds,$$

which is equivalent to the Total Change Theorem and part of the Fundamental Theorem of Calculus. Using the Mean Value Theorem for integrals, we can see how the first form of the remainder can be obtained from the integral form. □

Example Let $f(x) = \sin x$. Then

$$f(x) = P_3(x) + R_3(x),$$

where

$$P_3(x) = x - \frac{x^3}{3!} = x - \frac{x^3}{6},$$

and

$$R_3(x) = \frac{1}{4!}x^4 \sin\xi(x) = \frac{1}{24}x^4 \sin\xi(x),$$

where $\xi(x)$ is between 0 and x. The polynomial $P_3(x)$ is the 3rd Maclaurin polynomial of $\sin x$, or the 3rd Taylor polynomial with center $x_0 = 0$.

If $x \in [-1, 1]$, then

$$|R_n(x)| = \left|\frac{1}{24}x^4 \sin\xi(x)\right| = \left|\frac{1}{24}\right| |x^4||\sin\xi(x)| \leq \frac{1}{24},$$

since $|\sin x| \leq 1$ for all x. This bound on $|R_n(x)|$ serves as an upper bound for the error in the approximation of $\sin x$ by $P_3(x)$ for $x \in [-1, 1]$. □

Example Let $f(x) = e^x$. Then

$$f(x) = P_2(x) + R_2(x),$$

where

$$P_2(x) = 1 + x + \frac{x^2}{2},$$

and

$$R_2(x) = \frac{x^3}{6}e^{\xi(x)},$$

where $\xi(x)$ is between 0 and x. The polynomial $P_2(x)$ is the 2nd Maclaurin polynomial of e^x, or the 2nd Taylor polynomial with center $x_0 = 0$.

If $x > 0$, then $R_2(x)$ can become quite large, whereas its magnitude is much smaller if $x < 0$. Therefore, one method of computing e^x using a

Maclaurin polynomial is to use the nth Maclaurin polynomial $P_n(x)$ of e^x when $x < 0$, where n is chosen sufficiently large so that $R_n(x)$ is small for the given value of x. If $x > 0$, then we instead compute e^{-x} using the nth Maclaurin polynomial for e^{-x}, which is given by

$$P_n(x) = 1 - x + \frac{x^2}{2} - \frac{x^3}{6} + \cdots + \frac{(-1)^n x^n}{n!},$$

and then obtaining an approximation to e^x by taking the reciprocal of our computed value of e^{-x}. □

Example Let $f(x) = x^2$. Then, for any real number x_0,

$$f(x) = P_1(x) + R_1(x),$$

where

$$P_1(x) = x_0^2 + 2x_0(x - x_0) = 2x_0x - x_0^2,$$

and

$$R_1(x) = (x - x_0)^2.$$

Note that the remainder does not include a "mystery point" $\xi(x)$ since the 2nd derivative of x^2 is only a constant. The linear function $P_1(x)$ describes the tangent line to the graph of $f(x)$ at the point $(x_0, f(x_0))$. If $x_0 = 1$, then we have

$$P_1(x) = 2x - 1,$$

and

$$R_1(x) = (x - 1)^2.$$

We can see that near $x = 1$, $P_1(x)$ is a reasonable approximation to x^2, since the error in this approximation, given by $R_1(x)$, would be small in this case. □

1.3 Roundoff Errors and Computer Arithmetic

In computing the solution to any mathematical problem, there are many sources of error that can impair the accuracy of the computed solution. The study of these sources of error is called *error analysis*, which will be discussed in the next section. In this section, we will focus on one type of error that occurs in all computation, whether performed by hand or on a computer: *roundoff error*.

This error is due to the fact that in computation, real numbers can only be represented using a finite number of digits. In general, it is not possible

to represent real numbers exactly with this limitation, and therefore they must be approximated by real numbers that *can* be represented using a fixed number of digits, which is called the *precision*. Furthermore, as we shall see, arithmetic operations applied to numbers that can be represented exactly using a given precision do not necessarily produce a result that can be represented using the same precision. It follows that if a fixed precision is used, then *every* arithmetic operation introduces error into a computation.

Given that scientific computations can have several sources of error, one would think that it would be foolish to compound the problem by performing arithmetic using fixed precision. As we shall see in this lecture, however, using a fixed precision is far more practical than other options, and, as long as computations are performed carefully, sufficient accuracy can still be achieved.

1.3.1 Floating-Point Numbers

We now describe a typical system for representing real numbers on a computer.

Definition *(Floating-point Number System) Given integers $\beta > 1$, $p \geq 1$, L, and $U \geq L$, a* **floating-point number system** $\mathbb{F}$ *is defined to be the set of all real numbers of the form*

$$x = \pm m\beta^E.$$

The number m is the **mantissa** *of x, and has the form*

$$m = \left(\sum_{j=0}^{p-1} d_j \beta^{-j} \right),$$

where each digit d_j, $j = 0, \ldots, p-1$ is an integer satisfying $0 \leq d_i \leq \beta - 1$. The number E is called the **exponent** *or the* **characteristic** *of x, and it is an integer satisfying $L \leq E \leq U$. The integer p is called the* **precision** *of $\mathbb{F}$, and β is called the* **base** *of $\mathbb{F}$.*

The term "floating-point" comes from the fact that as a number $x \in \mathbb{F}$ is multiplied by or divided by a power of β, the mantissa does not change, only the exponent. As a result, the decimal point shifts, or "floats," to account for the changing exponent.

Nearly all computers use a binary floating-point system, in which $\beta = 2$. In fact, most computers conform to the *IEEE standard* for floating-point

arithmetic. The standard specifies, among other things, how floating-point numbers are to be represented in memory. Two representations are given, one for *single-precision* and one for *double-precision*. Under the standard, single-precision floating-point numbers occupy 4 bytes in memory, with 23 bits used for the mantissa, 8 for the exponent, and one for the sign. IEEE double-precision floating-point numbers occupy eight bytes in memory, with 52 bits used for the mantissa, 11 for the exponent, and one for the sign.

Example Let $x = -117$. Then, in a floating-point number system with base $\beta = 10$, x is represented as

$$x = -(1.17)10^2,$$

where 1.17 is the mantissa and 2 is the exponent. If the base $\beta = 2$, then we have

$$x = -(1.110101)2^6,$$

where 1.110101 is the mantissa and 6 is the exponent. The mantissa should be interpreted as a string of binary digits, rather than decimal digits; that is,

$$\begin{aligned} 1.110101 &= 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} + 0 \cdot 2^{-5} + 1 \cdot 2^{-6} \\ &= 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{16} + \frac{1}{64} \\ &= \frac{117}{64} \\ &= \frac{117}{2^6}. \end{aligned}$$

□

1.3.2 Properties of Floating-Point Systems

A floating-point system $\mathbb{F}$ can only represent a finite subset of the real numbers. As such, it is important to know how large in magnitude a number can be and still be represented, at least approximately, by a number in $\mathbb{F}$. Similarly, it is important to know how small in magnitude a number can be and still be represented by a *nonzero* number in $\mathbb{F}$; if its magnitude is too small, then it is most accurately represented by zero.

Definition *(Underflow, Overflow) Let $\mathbb{F}$ be a floating-point number system. The smallest positive number in $\mathbb{F}$ is called the* **underflow level***, and it has the value*

$$UFL = m_{\min}\beta^L,$$

where L is the smallest valid exponent and $m_{\min}$ is the smallest mantissa. The largest positive number in $\mathbb{F}$ is called the overflow level*, and it has the value*

$$OFL = \beta^{U+1}(1 - \beta^{-p}).$$

The value of $m_{\min}$ depends on whether floating-point numbers are *normalized* in $\mathbb{F}$; this point will be discussed later. The overflow level is the value obtained by setting each digit in the mantissa to $\beta - 1$ and using the largest possible value, U, for the exponent.

It is important to note that the real numbers that can be represented in $\mathbb{F}$ are not equally spaced along the real line. Numbers having the same exponent are equally spaced, and the spacing between numbers in $\mathbb{F}$ decreases as their magnitude decreases.

Normalization

It is common to *normalize* floating-point numbers by specifying that the leading digit d_0 of the mantissa be nonzero. In a binary system, with $\beta = 2$, this implies that the leading digit is equal to 1, and therefore need not be stored. Therefore, in the IEEE floating-point standard, $p = 24$ for single precision, and $p = 53$ for double precision, even though only 23 and 52 bits, respectively, are used to store mantissas. In addition to the benefit of gaining one additional bit of precision, normalization also ensures that each floating-point number has a unique representation.

One drawback of normalization is that fewer numbers near zero can be represented exactly than if normalization is not used. Therefore, the IEEE standard provides for a practice called *gradual underflow*, in which the leading digit of the mantissa is allowed to be zero when the exponent is equal to L, thus allowing smaller values of the mantissa. In such a system, the number UFL is equal to β^{L-p+1}, whereas in a normalized system, UFL $= \beta^L$.

1.3.3 Rounding

A number that can be represented exactly in a floating-point system is called a *machine number*. Since only finitely many real numbers are machine numbers, it is necessary to determine how non-machine numbers are to be approximated by machine numbers. The process of choosing a machine number to approximate a non-machine number is called *rounding*, and the error introduced by such an approximation is called *roundoff error*. Given a

real number x, the machine number obtained by rounding x is denoted by $\text{fl}(x)$.

In most floating-point systems, rounding is achieved by one of two strategies:

- *chopping*, or *rounding to zero*, is the simplest strategy, in which the base-β expansion of a number is truncated after the first p digits. As a result, $\text{fl}(x)$ is the unique machine number between 0 and x that is nearest to x.
- *rounding to nearest*, or *rounding to even* sets $\text{fl}(x)$ to be the machine number that is closest to x in absolute value; if two numbers satisfy this property, then $\text{fl}(x)$ is set to be the choice whose last digit is even.

Example Suppose we are using a floating-point system with $\beta = 10$ (decimal), with $p = 4$ significant digits. Then, if we use chopping, or rounding to zero, we have $\text{fl}(2/3) = 0.6666$, whereas if we use rounding to nearest, then we have $\text{fl}(2/3) = 0.6667$. If $p = 2$ and we use rounding to nearest, then $\text{fl}(89.5) = 90$, because we choose to round so that the last digit is even when the original number is equally distant from two machine numbers. This is why the strategy of rounding to nearest is sometimes called *rounding to even*. □

Machine Precision

In error analysis, it is necessary to estimate error incurred in each step of a computation. As such, it is desirable to know an upper bound for the relative error introduced by rounding. This leads to the following definition.

Definition *(Machine Precision) Let $\mathbb{F}$ be a floating-point number system. The* **unit roundoff** *or* **machine precision**, *denoted by ϵ_{mach}, is the real number that satisfies*

$$\left|\frac{\text{fl}(x) - x}{x}\right| \leq \epsilon_{\text{mach}}$$

for any real number x such that $\text{UFL} < x < \text{OFL}$.

An intuitive definition of ϵ_{mach} is that it is the smallest positive number such that

$$\text{fl}\left(1 + \epsilon_{\text{mach}}\right) > 1.$$

The value of ϵ_{mach} depends on the rounding strategy that is used. If rounding toward zero is used, then $\epsilon_{\text{mach}} = \beta^{1-p}$, whereas if rounding to nearest is used, $\epsilon_{\text{mach}} = \frac{1}{2}\beta^{1-p}$.

It is important to avoid confusing ϵ_{mach} with the underflow level UFL. The unit roundoff is determined by the number of digits in the mantissa, whereas the underflow level is determined by the range of allowed exponents. However, we do have the relation that $0 < \text{UFL} < \epsilon_{\text{mach}}$.

Example The following table summarizes the main aspects of a general floating-point system and a double-precision floating-point system that uses a 52-bit mantissa and 11-bit exponent. For both systems, we assume that rounding to nearest is used, and that normalization is used.

	General	Double Precision
Form of machine number	$\pm m\beta^E$	$\pm 1.d_1d_2\cdots d_{52}2^E$
Precision	p	53
Exponent range	$L \leq E \leq U$	$-1023 \leq E \leq 1024$
UFL (Underflow Level)	β^L	2^{-1023}
OFL (Overflow Level)	$\beta^{U+1}(1-\beta^{-p})$	$2^{1025}(1-2^{-53})$
ϵ_{mach}	$\frac{1}{2}\beta^{1-p}$	2^{-53}

The Matlab functions `eps`, `realmax` and `realmin` return the values of ϵ_{mach}, OFL and UFL, respectively, on the system on which Matlab is being used. On a machine using an Intel Pentium processor, Matlab returns the following values when these functions are called:

Function	Value (5-digit decimal, rounded)	Value (Exact)
`eps`	2.2204×10^{-16}	2^{-52}
`realmax`	1.7977×10^{308}	$2^{1024}(1-2^{-53})$
`realmin`	2.2251×10^{-308}	2^{-1022}

The UFL and OFL values differ from the ones in the previous table because the exponent of -1023 (corresponding to all zeros in the exponent) is used to represent zero, since zero cannot be represented when using normalization, and the exponent of 1024 (corresponding to all ones in the exponent) is used to represent ∞ and $-\infty$, so that computation does not have to halt after overflow occurs. The value returned by `eps` is not the true value of ϵ_{mach}, because it can be shown by experimentation that

$$\epsilon_{\text{mach}} \leq 2^{-53} + 2^{-105},$$

which is much closer to the value that is given in the first table. □

1.3.4 Floating-Point Arithmetic

We now discuss the various issues that arise when performing *floating-point arithmetic*, or *finite-precision arithmetic*, which approximates arithmetic operations on real numbers.

When adding or subtracting floating-point numbers, it is necessary to shift one of the operands so that both operands have the same exponent, before adding or subtracting the mantissas. As a result, digits of precision are lost in the operand that is smaller in magnitude, and the result of the operation cannot be represented using a machine number. In fact, if x is the smaller operand and y is the larger operand, and $|x| < |y|\epsilon_{\text{mach}}$, then the result of the operation will simply be y (or $-y$, if y is to be subtracted from x), since the entire value of x is lost in rounding the result.

In multiplication or division, the operands need not be shifted, but the mantissas, when multiplied or divided, cannot necessarily be represented using only p digits of precision. The product of two mantissas requires $2p$ digits to be represented exactly, while the quotient of two mantissas could conceivably require infinitely many digits. Furthermore, overflow or underflow may occur depending on the exponents of the operands, since their sum or difference may lie outside of the interval $[L, U]$.

Because floating-point arithmetic operations are not exact, they do not follow all of the laws of real arithmetic. In particular, floating-point arithmetic is not associative; i.e., $x + (y + z) \neq (x + y) + z$ in floating-point arithmetic.

Absolute Error and Relative Error

Now that we have been introduced to some specific errors that can occur during computation, we introduce useful terminology for discussing such errors. Suppose that a real number $\hat{y}$ is an approximation to some real number y. For instance, $\hat{y}$ may be the closest number to y that can be represented using finite precision, or $\hat{y}$ may be the result of a sequence of arithmetic operations performed using finite-precision arithmetic, where y is the result of the same operations performed using exact arithmetic.

Definition *(Absolute Error, Relative Error) Let $\hat{y}$ be a real number that is an approximation to the real number y. The* **absolute error** *in $\hat{y}$ is*

$$E_{abs} = \hat{y} - y.$$

The **relative error** *in* $\hat{y}$ *is*

$$E_{rel} = \frac{\hat{y} - y}{y},$$

provided that y *is nonzero.*

The absolute error is the most natural measure of the accuracy of an approximation, but it can be misleading. Even if the absolute error is small in magnitude, the approximation may still be grossly inaccurate if the exact value y is even smaller in magnitude. For this reason, it is preferable to measure accuracy in terms of the relative error.

The magnitude of the relative error in $\hat{y}$ can be interpreted as a percentage of $|y|$. For example, if the relative error is greater than 1 in magnitude, then $\hat{y}$ can be considered completely erroneous, since the error is larger in magnitude as the exact value. Another useful interpretation of the relative error concerns *significant digits*, which are all digits excluding leading zeros. Specifically, if the relative error is at most β^{-p}, where β is an integer greater than 1, then the representation of $\hat{y}$ in base β has at least p correct significant digits.

It should be noted that the absolute error and relative error are often defined using absolute value; that is,

$$E_{abs} = |\hat{y} - y|, \quad E_{rel} = \left|\frac{\hat{y} - y}{y}\right|.$$

This definition is preferable when one is only interested in the magnitude of the error, which is often the case. If the sign, or direction, of the error is also of interest, then the first definition must be used.

Example Assume that we are using a floating-point system that uses base $\beta = 10$ (decimal), with 4 digits of precision and rounding to nearest. Then, if we add the numbers 0.4567×10^0 and 0.8580×10^{-2}, we obtain the exact result

$$x = 0.4567 \times 10^0 + 0.008530 \times 10^0 = 0.46523 \times 10^0,$$

which is rounded to

$$\text{fl}(x) = 0.4652 \times 10^0.$$

The absolute error in this computation is

$$E_{abs} = \text{fl}(x) - x = 0.4652 - 0.46523 = -0.00003,$$

while the relative error is

$$E_{rel} = \frac{\text{fl}(x) - x}{x} = \frac{0.4652 - 0.46523}{0.46523} \approx -0.000064484.$$

Using the same floating-point system, suppose that we multiply 0.4567×10^4 and 0.8530×10^{-2}. The exact result is

$$x = (0.4567 \times 10^4) \times (0.8530 \times 10^{-2}) = 0.3895651 \times 10^2 = 38.95651,$$

which is rounded to

$$\text{fl}(x) = 0.3896 \times 10^2 = 38.96.$$

The absolute error in this computation is

$$E_{abs} = \text{fl}(x) - x = 38.96 - 38.95651 = 0.00349,$$

while the relative error is

$$E_{rel} = \frac{\text{fl}(x) - x}{x} = \frac{38.96 - 38.95651}{38.95651} \approx 0.000089587.$$

We see that in this case, the relative error is smaller than the absolute error, because the exact result is larger than 1, whereas in the previous operation, the relative error was larger in magnitude, because the exact result is smaller than 1.

It should be noted that given a number such as 0.4567×10^4, the actual mantissa is the number 4.567 and the exponent is 3, since the mantissa m in a normalized floating-point system must satisfy $1 \leq m < \beta$, where β is the base. In this example, the numbers are written in such a way that all four significant digits are grouped together on the same side of the decimal point. The examples in the text use this format for examples involving decimal floating-point arithmetic, so it is used here as well for consistency. □

Example Suppose that the exact value of a computation is supposed to be 10^{-16}, and an approximation of 2×10^{-16} is obtained. Then the absolute error in this approximation is

$$E_{abs} = 2 \times 10^{-16} - 10^{-16} = 10^{-16},$$

which suggests the computation is accurate because this error is small. However, the relative error is

$$E_{rel} = \frac{2 \times 10^{-16} - 10^{-16}}{10^{-16}} = 1,$$

which suggests that the computation is completely erroneous, because by this measure, the error is equal in magnitude to the exact value; that is, the error is 100%. It follows that an approximation of zero would be just as accurate. This example, although an extreme case, illustrates why the absolute error can be a misleading measure of error. □

Cancellation

Subtraction of floating-point numbers presents a unique difficulty, in addition to the rounding error previously discussed. If the operands, after shifting exponents as needed, have leading digits in common, then these digits cancel and the first digit in which the operands do not match becomes the leading digit. However, since each operand is represented using only p digits, it follows that the result contains only $p - m$ correct digits, where m is the number of leading digits that cancel.

In an extreme case, if the two operands differ by less than ϵ_{mach}, then the result contains no correct digits; it consists entirely of roundoff error from previous computations. This phenomenon is known as *catastrophic cancellation*. Because of the highly detrimental effect of this cancellation, it is important to ensure that no steps in a computation compute small values from relatively large operands. Often, computations can be rearranged to avoid this risky practice.

Example Consider the quadratic equation

$$ax^2 + bx + c = 0,$$

which has the solutions

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}.$$

Suppose that $b > 0$. Then, in computing x_1, we encounter catastrophic cancellation if b is much larger than a and c, because this implies that $\sqrt{b^2 - 4ac} \approx b$ and as a result we are subtracting two numbers that are nearly equal in computing the numerator. On the other hand, if $b < 0$, we encounter this same difficulty in computing x_2.

Suppose that we use 4-digit rounding arithmetic to compute the roots of the equation

$$x^2 + 10,000x + 1 = 0.$$

Then, we obtain $x_1 = 0$ and $x_2 = -10,000$. Clearly, x_1 is incorrect because if we substitute $x = 0$ into the equation then we obtain the contradiction

$1 = 0$. In fact, if we use 7-digit rounding arithmetic then we obtain the same result. Only if we use at least 8 digits of precision do we obtain roots that are reasonably correct,

$$x_1 \approx -1 \times 10^{-4}, \quad x_2 \approx -9.9999999 \times 10^3.$$

A similar result is obtained if we use 4-digit rounding arithmetic but compute x_1 using the formula

$$x_1 = \frac{-2c}{b + \sqrt{b^2 - 4ac}},$$

which can be obtained by multiplying and dividing the original formula for x_1 by the *conjugate* of the numerator, $-b - \sqrt{b^2 - 4ac}$. The resulting formula is not susceptible to catastrophic cancellation, because an addition is performed instead of a subtraction. □

1.3.5 Other Arithmetic Systems

It is natural to ask whether it is feasible to use *variable-precision* arithmetic, in which the precision is automatically increased to represent results of operations exactly. Variable-precision arithmetic has been implemented in some software environments, such as Maple. Unfortunately, the increased accuracy is more than offset by the loss of efficiency. Not only are arithmetic operations more expensive due to the increased precision, but they are implemented in software, rather than hardware, because hardware only performs arithmetic in fixed precision. As a result, variable-precision arithmetic does not enjoy widespread use.

To aid in error estimation, some arithmetic systems perform *interval analysis* on arithmetic operations that are implemented using either fixed or variable precision. In interval analysis, the result of each arithmetic operation is represented not as a single floating-point number but as an interval $[a, b]$ such that the result that would be obtained using exact arithmetic lies within the interval $[a, b]$. This interval is used to determine the appropriate interval for future computations that use the associated result as an operand. In the case of variable-precision arithmetic, interval analysis can increase efficiency by providing a guide to precision adjustment: if the interval becomes too large, then precision should be increased, perhaps repeating computations in order to ensure sufficient accuracy.

1.4 Approximations in Numerical Analysis

Mathematical problems arising from scientific applications present a wide variety of difficulties that prevent us from solving them exactly. This has led to an equally wide variety of techniques for computing approximations to quantities occurring in such problems in order to obtain approximate solutions. In this lecture, we will describe the types of approximations that can be made, and learn some basic techniques for analyzing the accuracy of these approximations.

1.4.1 Sources of Approximation

Suppose that we are attempting to solve a particular instance of a problem arising from a mathematical model of a scientific application. We say that such a problem is *well-posed* if it meets the following criteria:

- The problem has a unique solution.
- A small perturbation in the problem data results in a small perturbation in the solution; i.e., the solution *depends continuously* on the data.

By the first condition, the process of solving a well-posed problem can be seen to be equivalent to the evaluation of some function f at some known value x, where x represents the problem data. Since, in many cases, knowledge of the function f is limited, the task of computing $f(x)$ can be viewed, at least conceptually, as the execution of some (possibly infinite) sequence of steps that solves the underlying problem for the data x. The goal in numerical analysis is to develop a finite sequence of steps, i.e., an algorithm, for computing an approximation to the value $f(x)$.

There are two general types of error that occur in the process of computing this approximation to $f(x)$:

1. *data error* is the error in the data x. In reality, numerical analysis involves solving a problem with *approximate* data $\hat{x}$. The exact data is often unavailable because it must be obtained by measurements or other computations that fail to be exact due to limited precision. In addition, data may be altered in order to simplify the solution process.

2. *computational error* refers to the error that occurs when attempting to compute $f(\hat{x})$. Effectively, we must approximate $f(\hat{x})$ by the quantity $\hat{f}(\hat{x})$, where $\hat{f}$ is a function that approximates f. This approximation

may be the result of *truncation*, which occurs when it is not possible to evaluate f exactly using a finite sequence of steps, and therefore a finite sequence that evaluates f approximately must be used instead. This particular source of computational error will be discussed in this lecture. Another source of computational error is roundoff error, which was discussed in the previous lecture.

1.4.2 Basics of Error Analysis

Intuitively, it is not difficult to conclude that any scientific computation can include several approximations, each of which introduces error in the computed solution. Therefore, it is necessary to understand the effects of these approximations on accuracy. The study of these effects is known as *error analysis*.

Error analysis will be a recurring theme in this course. In this lecture, we will introduce some basic concepts that will play a role in error analyses of specific algorithms in later lectures.

Forward Error and Backward Error

Suppose that we compute an approximation $\hat{y} = \hat{f}(x)$ of the value $y = f(x)$ for a given function f and given problem data x. Before we can analyze the accuracy of this approximation, we must have a precisely defined notion of error in such an approximation. We now provide this precise definition.

Definition *(Forward Error) Let x be a real number and let $f : \mathbb{R} \to \mathbb{R}$ be a function. If $\hat{y}$ is a real number that is an approximation to $y = f(x)$, then the* **forward error** *in $\hat{y}$ is the difference $\Delta y = \hat{y} - y$. If $y \neq 0$, then the* **relative forward error** *in $\hat{y}$ is defined by*

$$\frac{\Delta y}{y} = \frac{\hat{y} - y}{y}.$$

Clearly, our primary goal in error analysis is to obtain an estimate of the forward error Δy. Unfortunately, it can be difficult to obtain this estimate directly.

An alternative approach is to instead view the computed value $\hat{y}$ as the *exact* solution of a problem with modified data; i.e., $\hat{y} = f(\hat{x})$ where $\hat{x}$ is a perturbation of x.

Definition *(Backward Error) Let x be a real number and let $f : \mathbb{R} \to \mathbb{R}$ be a function. Suppose that the real number $\hat{y}$ is an approximation to $y = f(x)$,*

and that $\hat{y}$ is in the range of f; that is, $\hat{y} = f(\hat{x})$ for some real number $\hat{x}$. Then, the quantity $\Delta x = \hat{x} - x$ is the **backward error** *in $\hat{y}$. If $x \neq 0$, then the* **relative forward error** *in $\hat{y}$ is defined by*

$$\frac{\Delta x}{x} = \frac{\hat{x} - x}{x}.$$

The process of estimating Δx is known as *backward error analysis*. As we will see, this estimate of the backward error, in conjunction with knowledge of f, can be used to estimate the forward error.

As discussed in the previous lecture, floating-point arithmetic does not follow the laws of real arithmetic. This tends to make forward error analysis difficult. In backward error analysis, however, real arithmetic is employed, since it is assumed that the computed result is the exact solution to a modified problem. This is one reason why backward error analysis is sometimes preferred.

In analysis of roundoff error, it is assumed that $\text{fl}(x \texttt{ op } y) = (x \texttt{ op } y)(1+\delta)$, where `op` is an arithmetic operation and δ is an unknown constant satisfying $|\delta| \leq \epsilon_{\text{mach}}$. From this assumption, it can be seen that the relative error in $\text{fl}(x \texttt{ op } y)$ is $|\delta|$. In the case of addition, the relative backward error in each operand is also $|\delta|$.

1.4.3 Sensitivity and Conditioning

In most cases, the goal of error analysis is to obtain an estimate of the forward relative error $(f(\hat{x}) - f(x))/f(x)$, but it is often easier to instead estimate the relative backward error $(\hat{x} - x)/x$. Therefore, it is necessary to be able to estimate the forward error in terms of the backward error. The following definition addresses this need.

Definition *(Condition Number) Let x be a real number and let $f : \mathbb{R} \to \mathbb{R}$ be a function. The* **absolute condition number***, denoted by κ_{abs}, is the ratio of the magnitude of the forward error to the magnitude of the backward error,*

$$\kappa_{abs} = \frac{|f(\hat{x}) - f(x)|}{|\hat{x} - x|}.$$

If $f(x) \neq 0$, then the **relative condition number** *of the problem of computing $y = f(x)$, denoted by κ_{rel}, is the ratio of the magnitude of the relative forward error to the magnitude of the relative backward error,*

$$\kappa_{rel} = \frac{|(f(\hat{x}) - f(x))/f(x)|}{|(\hat{x} - x)/x|} = \frac{|\Delta y/y|}{|\Delta x/x|}.$$

Intuitively, either condition number is a measure of the change in the solution due to a change in the data. Since the relative condition number tends to be a more reliable measure of this change, it is sometimes referred to as simply the *condition number*.

If the condition number is large, e.g. much greater than 1, then a small change in the data can cause a disproportionately large change in the solution, and the problem is said to be *ill-conditioned* or *sensitive*. If the condition number is small, then the problem is said to be *well-conditioned* or *insensitive*.

Since the condition number, as defined above, depends on knowledge of the exact solution $f(x)$, it is necessary to estimate the condition number in order to estimate the relative forward error. To that end, we assume, for simplicity, that $f : \mathbb{R} \to \mathbb{R}$ is differentiable and obtain

$$\begin{aligned} \kappa_{rel} &= \frac{|x\Delta y|}{|y\Delta x|} \\ &= \frac{|x(f(x+\Delta x) - f(x))|}{|f(x)\Delta x|} \\ &\approx \frac{|xf'(x)\Delta x|}{|f(x)\Delta x|} \\ &\approx \left|\frac{xf'(x)}{f(x)}\right|. \end{aligned}$$

Therefore, if we can estimate the backward error Δx, and if we can bound f and f' near x, we can then bound the condition number and obtain an estimate of the relative forward error.

Of course, the condition number is undefined if the exact value $f(x)$ is zero. In this case, we can instead use the absolute condition number. Using the same approach as before, the absolute condition number can be estimated using the derivative of f. Specifically, we have $\kappa_{abs} \approx |f'(x)|$.

1.4.4 Conditioning, Stability and Accuracy

Determining the condition, or sensitivity, of a problem is an important task in the error analysis of an algorithm designed to solve the problem, but it does not provide sufficient information to determine whether an algorithm will yield an accurate approximate solution.

Recall that the condition number of a function f depends on, among other things, the absolute forward error $f(\hat{x}) - f(x)$. However, an algorithm for evaluating $f(x)$ actually evaluates a function $\hat{f}$ that approximates f,

producing an approximation $\hat{y} = \hat{f}(x)$ to the exact solution $y = f(x)$. In our definition of backward error, we have assumed that $\hat{f}(x) = f(\hat{x})$ for some $\hat{x}$ that is close to x; i.e., our approximate solution to the original problem is the exact solution to a "nearby" problem.

This assumption has allowed us to define the condition number of f independently of any approximation $\hat{f}$. This independence is necessary, because the sensitivity of a problem depends solely on the problem itself and not any algorithm that may be used to approximately solve it.

Is it always reasonable to assume that any approximate solution is the exact solution to a nearby problem? Unfortunately, it is not. It is possible that an algorithm that yields an accurate approximation for given data may be unreasonably sensitive to perturbations in that data. This leads to the concept of a *stable algorithm*: an algorithm applied to a given problem with given data x is said to be *stable* if it computes an approximate solution that is the exact solution to the same problem with data $\hat{x}$, where $\hat{x}$ is a small perturbation of x.

It can be shown that if a problem is well-conditioned, and if we have a stable algorithm for solving it, then the computed solution can be considered accurate, in the sense that the relative error in the computed solution is small. On the other hand, a stable algorithm applied to an ill-conditioned problem cannot be expected to produce an accurate solution.

1.4.5 Convergence

Many algorithms in numerical analysis are *iterative methods* that produce a sequence $\{\alpha_n\}$ of approximate solutions which, ideally, converges to a limit α that is the exact solution as n approaches ∞. Because we can only perform a finite number of iterations, we cannot obtain the exact solution, and we have introduced computational error.

If our iterative method is properly designed, then this computational error will approach zero as n approaches ∞. However, it is important that we obtain a sufficiently accurate approximate solution using as few computations as possible. Therefore, it is not practical to simply perform enough iterations so that the computational error is determined to be sufficiently small, because it is possible that another method may yield comparable accuracy with less computational effort.

The total computational effort of an iterative method depends on both the effort per iteration and the number of iterations performed. Therefore, in order to determine the amount of computation that is needed to attain a given accuracy, we must be able to measure the error in α_n as a function of

n. The more rapidly this function approaches zero as n approaches ∞, the more rapidly the sequence of approximations $\{\alpha_n\}$ converges to the exact solution α, and as a result, fewer iterations are needed to achieve a desired accuracy. We now introduce some terminology that will aid in the discussion of the convergence behavior of iterative methods.

Definition *(Big-O Notation) Let f and g be two functions defined on a domain $D \subseteq \mathbb{R}$ that is not bounded above. We write that $f(n) = O(g(n))$ if there exists a positive constant c such that*

$$|f(n)| \leq c|g(n)|, \quad n \geq n_0,$$

for some $n_0 \in D$.

As sequences are functions defined on $\mathbb{N}$, the domain of the natural numbers, we can apply big-O notation to sequences. Therefore, this notation is useful to describe the rate at which a sequence of computations converges to a limit.

Definition *(Rate of Convergence) Let $\{\alpha_n\}_{n=1}^{\infty}$ and $\{\beta_n\}_{n=1}^{\infty}$ be sequences that satisfy*

$$\lim_{n\to\infty} \alpha_n = \alpha, \quad \lim_{n\to\infty} \beta_n = 0,$$

where α is a real number. We say that $\{\alpha_n\}$ converges to α with **rate of convergence** *$O(\beta_n)$ if $\alpha_n - \alpha = O(\beta_n)$.*

We say that an iterative method converges rapidly, in some sense, if it produces a sequence of approximate solutions whose rate of convergence is $O(\beta_n)$, where the terms of the sequence β_n approach zero rapidly as n approaches ∞. Intuitively, if two iterative methods for solving the same problem perform a comparable amount of computation during each iteration, but one method exhibits a faster rate of convergence, then that method should be used because it will require less overall computational effort to obtain an approximate solution that is sufficiently accurate.

Example Consider the sequence $\{\alpha_n\}_{n=1}^{\infty}$ defined by

$$\alpha_n = \frac{n+1}{n+2}, \quad n = 1, 2, \ldots$$

Then, we have

$$\lim_{n\to\infty} \alpha_n = \lim_{n\to\infty} \frac{n+1}{n+2}\frac{1/n}{1/n}$$

$$\begin{aligned} &= \lim_{n\to\infty} \frac{1+1/n}{1+2/n} \\ &= \frac{\lim_{n\to\infty}(1+1/n)}{\lim_{n\to\infty}(1+2/n)} \\ &= \frac{1+\lim_{n\to\infty} 1/n}{1+\lim_{n\to\infty} 2/n} \\ &= \frac{1+0}{1+0} \\ &= 1. \end{aligned}$$

That is, the sequence $\{\alpha_n\}$ converges to $\alpha = 1$. To determine the rate of convergence, we note that

$$\alpha_n - \alpha = \frac{n+1}{n+2} - 1 = \frac{n+1}{n+2} - \frac{n+2}{n+2} = \frac{-1}{n+2},$$

and since

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

for any positive integer n, it follows that

$$\alpha_n = \alpha + O\left(\frac{1}{n}\right).$$

On the other hand, consider the sequence $\{\alpha_n\}_{n=1}^{\infty}$ defined by

$$\alpha_n = \frac{2n^2+4n}{n^2+2n+1}, \quad n = 1, 2, \ldots$$

Then, we have

$$\begin{aligned} \lim_{n\to\infty} \alpha_n &= \lim_{n\to\infty} \frac{2n^2+4n}{n^2+2n+1}\frac{1/n^2}{1/n^2} \\ &= \lim_{n\to\infty} \frac{2+4/n}{1+2/n+1/n^2} \\ &= \frac{\lim_{n\to\infty}(2+4/n)}{\lim_{n\to\infty}(1+2/n+1/n^2)} \\ &= \frac{2+\lim_{n\to\infty} 4/n}{1+\lim_{n\to\infty}(2/n+1/n^2)} \\ &= 2. \end{aligned}$$

That is, the sequence $\{\alpha_n\}$ converges to $\alpha = 2$. To determine the rate of convergence, we note that

$$\alpha_n - \alpha = \frac{2n^2 + 4n}{n^2 + 2n + 1} - 2 = \frac{2n^2 + 4n}{n^2 + 2n + 1} - \frac{2n^2 + 4n + 2}{n^2 + 2n + 1} = \frac{-2}{n^2 + 2n + 1},$$

and since

$$\left| \frac{-2}{n^2 + 2n + 1} \right| = \left| \frac{2}{(n+1)^2} \right| \leq \left| \frac{2}{n^2} \right|$$

for any positive integer n, it follows that

$$\alpha_n = \alpha + O\left(\frac{1}{n^2}\right).$$

□

We can also use big-O notation to describe the rate of convergence of a *function* to a limit.

Example Consider the function $f(h) = 1 + 2h$. Since this function is continuous for all h, we have

$$\lim_{h \to 0} f(h) = f(0) = 1.$$

It follows that

$$f(h) - f_0 = (1 + 2h) - 1 = 2h = O(h),$$

so we can conclude that as $h \to 0$, $1 + 2h$ converges to 1 of order $O(h)$. □

Example Consider the function $f(h) = 1 + 4h + 2h^2$. Since this function is continuous for all h, we have

$$\lim_{h \to 0} f(h) = f(0) = 1.$$

It follows that

$$f(h) - f_0 = (1 + 4h + 2h^2) - 1 = 4h + 2h^2.$$

To determine the rate of convergence as $h \to 0$, we consider h in the interval $[-1, 1]$. In this interval, $|h^2| \leq |h|$. It follows that

$$\begin{aligned} |4h + 2h^2| &\leq |4h| + |2h^2| \\ &\leq |4h| + |2h| \\ &\leq 6|h|. \end{aligned}$$

Since there exists a constant C (namely, 6) such that $|4h+2h^2| \leq C|h|$ for h satisfying $|h| \leq h_0$ for some h_0 (namely, 1), we can conclude that as $h \to 0$, $1 + 4h + 2h^2$ converges to 1 of order $O(h)$.

In general, when $f(h)$ denotes an approximation that depends on h, and

$$f_0 = \lim_{h \to 0} f(h)$$

denotes the exact value, $f(h) - f_0$ represents the absolute error in the approximation $f(h)$. When this error is a polynomial in h, as in this example and the previous example, the rate of convergence is $O(h^k)$ where k is the smallest exponent of h in the error. This is because as $h \to 0$, the smallest power of h approaches zero more slowly than higher powers, thereby making the dominant contribution to the error.

By contrast, when determining the rate of convergence of a sequence $\{\alpha_n\}$ as $n \to \infty$, the *highest* power of n determines the rate of convergence. As powers of n are negative if convergence occurs at all as $n \to \infty$, and powers of h are positive if convergence occurs at all as $h \to 0$, it can be said that for either types of convergence, it is the exponent that is closest to zero that determines the rate of convergence. □

Example Consider the function $f(h) = \cos h$. Since this function is continuous for all h, we have

$$\lim_{h \to 0} f(h) = f(0) = 1.$$

Using Taylor's Theorem, with center $h_0 = 0$, we obtain

$$f(h) = f(0) + f'(0)h + \frac{f''(\xi(h))}{2}h^2,$$

where $\xi(h)$ is between 0 and h. Substituting $f(h) = \cos h$ into the above, we obtain

$$\cos h = 1 - (\sin 0)h + \frac{-\cos \xi(h)}{2}h^2,$$

or

$$\cos h = 1 - \frac{\cos \xi(h)}{2}h^2.$$

Because $|\cos x| \leq 1$ for all x, we have

$$|\cos h - 1| = \left| -\frac{\cos \xi(h)}{2}h^2 \right| \leq \frac{1}{2}h^2,$$

so we can conclude that as $h \to 0$, $\cos h$ converges to 1 of order $O(h^2)$. □

Chapter 2

Solution of Equations in One Variable

2.1 Nonlinear Equations

To this point, we have only considered the solution of linear equations. We now explore the much more difficult problem of solving nonlinear equations of the form

$$f(\mathbf{x}) = \mathbf{0},$$

where $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}^m$ can be any known function. A solution $\mathbf{x}$ of such a nonlinear equation is called a *root* of the equation, as well as a *zero* of the function f.

2.1.1 Existence and Uniqueness

For simplicity, we assume that the function $f : \mathbb{R}^n \to \mathbb{R}^m$ is continuous on the domain under consideration. Then, each equation $f_i(\mathbf{x}) = 0$, $i = 1, \ldots, m$, defines a hypersurface in $\mathbb{R}^m$. The solution of $f(\mathbf{x}) = 0$ is the intersection of these hypersurfaces, if the intersection is not empty. It is not hard to see that there can be a unique solution, infinitely many solutions, or no solution at all.

For a general equation $f(\mathbf{x}) = \mathbf{0}$, it is not possible to characterize the conditions under which a solution exists or is unique. However, in some situations, it is possible to determine existence analytically. For example, in one dimension, the Intermediate Value Theorem implies that if a continuous function $f(x)$ satisfies $f(a) \leq 0$ and $f(b) \geq 0$ where $a < b$, then $f(x) = 0$ for some $x \in (a, b)$.

Similarly, it can be concluded that $f(x) = 0$ for some $x \in (a, b)$ if the function $(x - z)f(x) \geq 0$ for $x = a$ and $x = b$, where $z \in (a, b)$. This condition can be generalized to higher dimensions. If $S \subset \mathbb{R}^n$ is an open, bounded set, and $(\mathbf{x} - \mathbf{z})^T f(\mathbf{x}) \geq 0$ for all $\mathbf{x}$ on the boundary of S and for some $\mathbf{z} \in S$, then $f(\mathbf{x}) = \mathbf{0}$ for some $\mathbf{x} \in S$. Unfortunately, checking this condition can be difficult in practice.

One useful result from calculus that can be used to establish existence and, in some sense, uniqueness of a solution is the *Inverse Function Theorem*, which states that if the Jacobian of f is nonsingular at a point $\mathbf{x}_0$, then f is invertible near $\mathbf{x}_0$ and the equation $f(\mathbf{x}) = \mathbf{y}$ has a unique solution for all $\mathbf{y}$ near $f(\mathbf{x}_0)$.

If the Jacobian of f at a point $\mathbf{x}_0$ is singular, then f is said to be *degenerate* at $\mathbf{x}_0$. Suppose that $\mathbf{x}_0$ is a solution of $f(\mathbf{x}) = \mathbf{0}$. Then, in one dimension, degeneracy means $f'(x_0) = 0$, and we say that x_0 is a *double root* of $f(x)$. Similarly, if $f^{(j)}(x_0) = 0$ for $j = 0, \ldots, m-1$, then x_0 is a root of multiplicity m. We will see that degeneracy can cause difficulties when trying to solve nonlinear equations.

2.1.2 Sensitivity

Recall that the absolute condition number of a function $f(x)$ is approximated by $|f'(x)|$. In solving a nonlinear equation in one dimension, we are trying to solve an inverse problem, where the forward problem is the evaluation of f at $x = 0$. It follows that the condition number for solving $f(x) = 0$ is approximately $1/|f'(x_0)|$, where x_0 is the solution. This discussion can be generalized to higher dimensions, where the condition number is measured using the norm of the Jacobian.

2.2 The Bisection Method

Suppose that $f(x)$ is a continuous function that changes sign on the interval $[a, b]$. Then, by the Intermediate Value Theorem, $f(x) = 0$ for some $x \in [a, b]$. How can we find the solution, knowing that it lies in this interval?

The method of *bisection* attempts to reduce the size of the interval in which a solution is known to exist. Suppose that we evaluate $f(m)$, where $m = (a + b)/2$. If $f(m) = 0$, then we are done. Otherwise, f must change sign on the interval $[a, m]$ or $[m, b]$, since $f(a)$ and $f(b)$ have different signs. Therefore, we can cut the size of our search space in half, and continue this process until the interval of interest is sufficiently small, in which case

we must be close to a solution. The following algorithm implements this approach.

Algorithm (Bisection) Let f be a continuous function on the interval $[a, b]$ that changes sign on (a, b). The following algorithm computes an approximation p^* to a number p in (a, b) such that $f(p) = 0$.

for $j = 1, 2, \ldots$ **do**
  $p_j = (a + b)/2$
  if $f(p_j) = 0$ **or** $b - a$ is sufficiently small **then**
    $p^* = p_j$
    return p^*
  end
  if $f(a)f(p_j) < 0$ **then**
    $b = p_j$
  else
    $a = p_j$
  end
end

At the beginning, it is known that (a, b) contains a solution. During each iteration, this algorithm updates the interval (a, b) by checking whether f changes sign in the first half (a, p_j), or in the second half (p_j, b). Once the correct half is found, the interval (a, b) is set equal to that half. Therefore, at the beginning of *each* iteration, it is known that the current interval (a, b) contains a solution.

The test $f(a)f(p_j) < 0$ is used to determine whether f changes sign in the interval (a, p_j) or (p_j, b). This test is more efficient than checking whether $f(a)$ is positive and $f(p_j)$ is negative, or vice versa, since we do not care which value is positive and which is negative. We only care whether they have different signs, and if they do, then their product must be negative.

In comparison to other methods, including some that we will discuss, bisection tends to converge rather slowly, but it is also guaranteed to converge. These qualities can be seen in the following result concerning the accuracy of bisection.

Theorem *Let f be continuous on $[a, b]$, and assume that $f(a)f(b) < 0$. For each positive integer n, let p_n be the nth iterate that is produced by the bisection algorithm. Then the sequence $\{p_n\}_{n=1}^{\infty}$ converges to a number p in (a, b) such that $f(p) = 0$, and each iterate p_n satisfies*

$$|p_n - p| \leq \frac{b - a}{2^n}.$$

It should be noted that because the nth iterate can lie anywhere within the interval (a, b) that is used during the nth iteration, it is possible that the error bound given by this theorem may be quite conservative.

Example We seek a solution of the equation $f(x) = 0$, where

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

Because $f(1) = -1$ and $f(2) = 1$, and f is continuous, we can use the Intermediate Value Theorem to conclude that $f(x) = 0$ has a solution in the interval $(1, 2)$, since $f(x)$ must assume every value between -1 and 1 in this interval.

We use the method of *bisection* to find a solution. First, we compute the midpoint of the interval, which is $(1 + 2)/2 = 1.5$. Since $f(1.5) = -0.25$, we see that $f(x)$ changes sign between $x = 1.5$ and $x = 2$, so we can apply the Intermediate Value Theorem again to conclude that $f(x) = 0$ has a solution in the interval $(1.5, 2)$.

Continuing this process, we compute the midpoint of the interval $(1.5, 2)$, which is $(1.5+2)/2 = 1.75$. Since $f(1.75) = 0.3125$, we see that $f(x)$ changes sign between $x = 1.5$ and $x = 1.75$, so we conclude that there is a solution in the interval $(1.5, 1.75)$. The following table shows the outcome of several more iterations of this procedure. Each row shows the current interval (a, b) in which we know that a solution exists, as well as the midpoint of the interval, given by $(a + b)/2$, and the value of f at the midpoint. Note that from iteration to iteration, only one of a or b changes, and the endpoint that changes is always set equal to the midpoint.

a	b	$m = (a + b)/2$	$f(m)$
1	2	1.5	-0.25
1.5	2	1.75	0.3125
1.5	1.75	1.625	0.015625
1.5	1.625	1.5625	-0.12109
1.5625	1.625	1.59375	-0.053711
1.59375	1.625	1.609375	-0.019287
1.609375	1.625	1.6171875	-0.0018921
1.6171875	1.625	1.62109325	0.0068512
1.6171875	1.62109325	1.619140625	0.0024757
1.6171875	1.619140625	1.6181640625	0.00029087

The correct solution, to ten decimal places, is 1.6180339887, which is the number known as the *golden ratio*. □

2.3 Fixed-point Iteration

A nonlinear equation of the form $f(x) = 0$ can be rewritten to obtain an equation of the form

$$g(x) = x,$$

in which case the solution is a *fixed point* of the function g. This formulation of the original problem $f(x) = 0$ will leads to a simple solution method known as *fixed-point iteration*. Before we describe this method, however, we must first discuss the questions of existence and uniqueness of a solution to the modified problem $g(x) = x$. The following result answers these questions.

Theorem *Let g be a continuous function on the interval $[a, b]$. If $g(x) \in [a, b]$ for each $x \in [a, b]$, then g has a fixed point in $[a, b]$. Furthermore, if g is differentiable on (a, b) and there exists a constant $k < 1$ such that*

$$|g'(x)| \leq k, \quad x \in (a, b),$$

then g has exactly one fixed point in $[a, b]$.

Given a continuous function g that is known to have a fixed point in an interval $[a, b]$, we can try to find this fixed point by repeatedly evaluating g at points in $[a, b]$ until we find a point x for which $g(x) = x$. This is the essence of the method of fixed-point iteration, the implementation of which we now describe.

Algorithm (Fixed-Point Iteration) Let g be a continuous function defined on the interval $[a, b]$. The following algorithm computes a number $x^* \in (a, b)$ that is a solution to the equation $g(x) = x$.

Choose an initial guess x_0 in $[a, b]$.
for $k = 0, 1, 2, \ldots$ **do**
    $x_{k+1} = g(x_k)$
    if $|x_{k+1} - x_k|$ is sufficiently small **then**
        $x^* = x_{k+1}$
        return x^*
    end
end

Under what circumstances will fixed-point iteration converge to the exact solution x^*? If we denote the error in x_k by $e_k = x_k - x^*$, we can see from Taylor's Theorem and the fact that $g(x^*) = x^*$ that $e_{k+1} \approx g'(x^*)e_k$.

Therefore, if $|g'(x^*)| \leq \rho$, where $\rho < 1$, then fixed-point iteration is *locally convergent*; that is, it converges if x_0 is chosen sufficiently close to x^*. This leads to the following result.

Theorem *(Fixed-Point Theorem) Let g be a continuous function on the interval $[a, b]$. If $g(x) \in [a, b]$ for each $x \in [a, b]$, and if there exists a constant $\rho < 1$ such that*

$$|g'(x)| \leq \rho, \quad x \in (a, b),$$

then the sequence of iterates $\{x_k\}_{k=0}^{\infty}$ converges to the unique fixed point x^* of g in $[a, b]$, for any initial guess $x_0 \in [a, b]$.

It can be seen from the preceding discussion why $g'(x)$ must be bounded away from 1 on (a, b), as opposed to the weaker condition $|g'(x)| < 1$ on (a, b). If $g'(x)$ is allowed to approach 1 as x approaches a point $c \in (a, b)$, then it is possible that the error e_k might not approach zero as k increases, in which case fixed-point iteration would not converge.

In general, when fixed-point iteration converges, it does so at a rate that varies inversely with the constant ρ that bounds $|g'(x)|$. This constant is called the *asymptotic error constant*. In the extreme case where derivatives of g are equal to zero at the solution x^*, the method can converge much more rapidly. We will discuss convergence behavior of various methods for solving nonlinear equations later in this chapter.

Often, there are many ways to convert an equation of the form $f(x) = 0$ to one of the form $g(x) = x$, the simplest being $g(x) = x - \phi(x)f(x)$ for any function ϕ. However, it is important to ensure that the conversion yields a function g for which fixed-point iteration will converge.

Example We use fixed-point iteration to compute a fixed point of $g(x) = \cos x$ in the interval $[0, 1]$. Since $|\cos x| \leq 1$ for all x, and $\cos x \geq 0$ on $[0, \pi/2]$, and $\pi/2 > 1$, we know that $\cos x$ maps $[0, 1]$ into $[0, 1]$. Since $\cos x$ is continuous for all x, we can conclude that $\cos x$ has a fixed point in $[0, 1]$. Because $g'(x) = -\sin x$, and $|-\sin x| \leq |-\sin 1| < 1$ on $[0, 1]$, we can also conclude that this fixed point is unique.

To use fixed-point iteration, we first choose an initial guess x_0 in $[0, 1]$. As discussed above, fixed-point iteration will converge for any initial guess, so we choose $x_0 = 0.5$. The table on page 4 shows the outcome of several iterations, in which we compute $x_{k+1} = \cos x_k$ for $k = 0, 1, 2, \ldots$. As the table shows, it takes nearly 30 iterations to obtain the fixed point to five decimal places, and there is considerable oscillation in the first iterations

before a reasonable approximate solution is obtained. This oscillation is shown in Figure 2.1.

As x_k converges, it can be seen from the table that the error is reduced by a factor of roughly 2/3 from iteration to iteration. In other words, fixed-point iteration, in this case, converges even more slowly than the bisection method. This suggests that $\cos x$ is a relatively poor choice for the iteration function $g(x)$ to solve the equation $g(x) = \cos x$. □

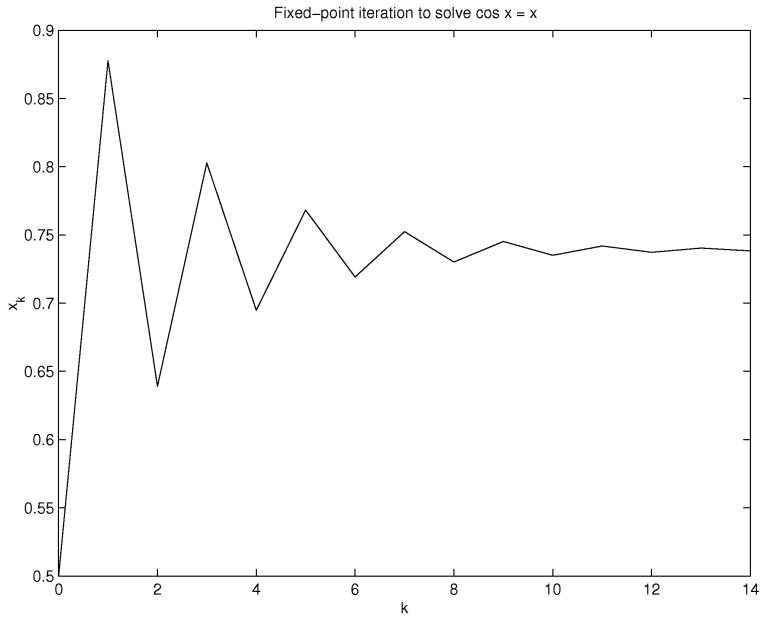

Figure 2.1: Fixed-point iteration applied to the equation $\cos x = x$, with $x_0 = 0.5$.

As previously discussed, a common choice for a function $g(x)$ to use with Fixed-point Iteration to solve the equation $f(x) = 0$ is a function of the form $g(x) = x - \phi(x)f(x)$, where $\phi(x)$ is nonzero. Clearly, the simplest choice of $\phi(x)$ is a constant function $\phi(x) \equiv \lambda$, but it is important to choose λ so that Fixed-point Iteration with $g(x)$ will converge.

Suppose that x^* is a solution of the equation $f(x) = 0$, and that f is continuously differentiable in a neighborhood of x^*, with $f'(x^*) = \alpha > 0$. Then, by continuity, there exists an interval $[x^* - \delta, x^* + \delta]$ containing x^* on which $m \leq f'(x) \leq M$, for positive constants m and M. It follows that for

any choice of a positive constant λ,

$$1 - \lambda M \leq 1 - \lambda f'(x) \leq 1 - \lambda m.$$

By choosing

$$\lambda = \frac{2}{M+m},$$

we obtain

$$1 - \lambda M = -k, \quad 1 - \lambda m = k, \quad k = \frac{M-m}{M+m},$$

which satisfies $0 < k < 1$. Therefore, if we define $g(x) = x - \lambda f(x)$, we have $|g'(x)| \leq k < 1$ on $[x^* - \delta, x^* + \delta]$.

Furthermore, if $|x - x^*| \leq \delta$, then, by the Mean Value Theorem,

$$|g(x) - x^*| = |g(x) - g(x^*)| = |g'(\xi)||x - x^*| < \delta,$$

and therefore g maps the interval $[x^* - \delta, x^* + \delta]$ into itself. We conclude that the Fixed-point Theorem applies, and Fixed-point Iteration converges to x^* for any choice of x_0 in $[x^* - \delta, x^* + \delta]$, with asymptotic error constant $|1 - \lambda\alpha| \leq k$.

In summary, if f is continuously differentiable in a neighborhood of a root x^* of $f(x) = 0$, and $f(x^*)$ is nonzero, then there exists a constant λ such that Fixed-point Iteration with $g(x) = x - \lambda f(x)$ converges to x^* for x_0 chosen sufficiently close to x^*. This approach to Fixed-point Iteration, with a constant ϕ, is known as *relaxation*.

Convergence can be accelerated by allowing λ to vary from iteration to iteration. Intuitively, an effective choice is to try to minimize $|g'(x)|$ near x^* by setting $\lambda = 1/f'(x_k)$, for each k, so that $g'(x_k) = 1 - \lambda f'(x_k) = 0$. This results in convergence with an asymptotic error constant of 0, which indicates much more rapid convergence. We will make this concept more precise later in this chapter.

2.4 Newton's Method

In the previous lecture, we developed a simple method, bisection, for approximately solving the equation $f(x) = 0$. Unfortunately, this method, while guaranteed to find a solution on an interval that is known to contain one, is not practical because of the large number of iterations that are necessary to obtain an accurate approximate solution.

To develop a more effective method for solving this problem of computing a solution to $f(x) = 0$, we can address the following questions:

- Are there cases in which the problem easy to solve, and if so, how do we solve it in such cases?
- Is it possible to apply our method of solving the problem in these "easy" cases to more general cases?

In this course, we will see that these questions are useful for solving a variety of problems. For the problem at hand, we ask whether the equation $f(x) = 0$ is easy to solve for any particular choice of the function f. Certainly, if f is a linear function, then it has a formula of the form $f(x) = m(x-a)+b$, where m and b are constants and $m \neq 0$. Setting $f(x) = 0$ yields the equation

$$m(x - a) + b = 0,$$

which can easily be solved for x to obtain the unique solution

$$x = a - \frac{b}{m}.$$

We now consider the case where f is not a linear function. Using Taylor's theorem, it is simple to construct a linear function that approximates $f(x)$ near a given point x_0. This function is simply the first Taylor polynomial of $f(x)$ with center x_0,

$$P_1(x) = f(x_0) + f'(x_0)(x - x_0).$$

This function has a useful geometric interpretation, as its graph is the tangent line of $f(x)$ at the point $(x_0, f(x_0))$.

We can obtain an approximate solution to the equation $f(x) = 0$ by determining where the linear function $P_1(x)$ is equal to zero. If the resulting value, x_1, is not a solution, then we can repeat this process, approximating f by a linear function near x_1 and once again determining where this approximation is equal to zero. The resulting algorithm is *Newton's method*, which we now describe in detail.

Algorithm (Newton's Method) Let $f : \mathbb{R} \to \mathbb{R}$ be a differentiable function. The following algorithm computes an approximate solution x^* to the equation $f(x) = 0$.

Choose an initial guess x_0
for $k = 0, 1, 2, \ldots$ **do**
    if $f(x_k)$ is sufficiently small **then**
        $x^* = x_k$

```
        return x*
    end
    x_{k+1} = x_k - f(x_k)/f'(x_k)
    if |x_{k+1} - x_k| is sufficiently small then
        x* = x_{k+1}
        return x*
    end
end
```

When Newton's method converges, it does so very rapidly. However, it can be difficult to ensure convergence, particularly if $f(x)$ has horizontal tangents near the solution x^*. Typically, it is necessary to choose a starting iterate x_0 that is close to x^*. As the following result indicates, such a choice, if close enough, is indeed sufficient for convergence.

Theorem *(Convergence of Newton's Method) Let f be twice continuously differentiable on the interval $[a, b]$, and suppose that $f(c) = 0$ and $f'(c) = 0$ for some $c \in [a, b]$. Then there exists a $\delta > 0$ such that Newton's Method applied to $f(x)$ converges to c for any initial guess x_0 in the interval $[c - \delta, c + \delta]$.*

Example We will use of Newton's Method in computing $\sqrt{2}$. This number satisfies the equation $f(x) = 0$ where

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

Since $f'(x) = 2x$, it follows that in Newton's Method, we can obtain the next iterate x_{n+1} from the previous iterate x_n by

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{x_n^2 - 2}{2x_n} = x_n - \frac{x_n^2}{2x_n} + \frac{2}{2x_n} = \frac{x_n}{2} + \frac{1}{x_n}.$$

We choose our starting iterate $x_0 = 1$, and compute the next several iterates as follows:

$$\begin{aligned} x_1 &= \frac{1}{2} + \frac{1}{1} = 1.5 \\ x_2 &= \frac{1.5}{2} + \frac{1}{1.5} = 1.41666667 \\ x_3 &= 1.41421569 \\ x_4 &= 1.41421356 \\ x_5 &= 1.41421356. \end{aligned}$$

Since the fourth and fifth iterates agree in to eight decimal places, we assume that 1.41421356 is a correct solution to $f(x) = 0$, to at least eight decimal places. The first two iterations are illustrated in Figure 2.2. □

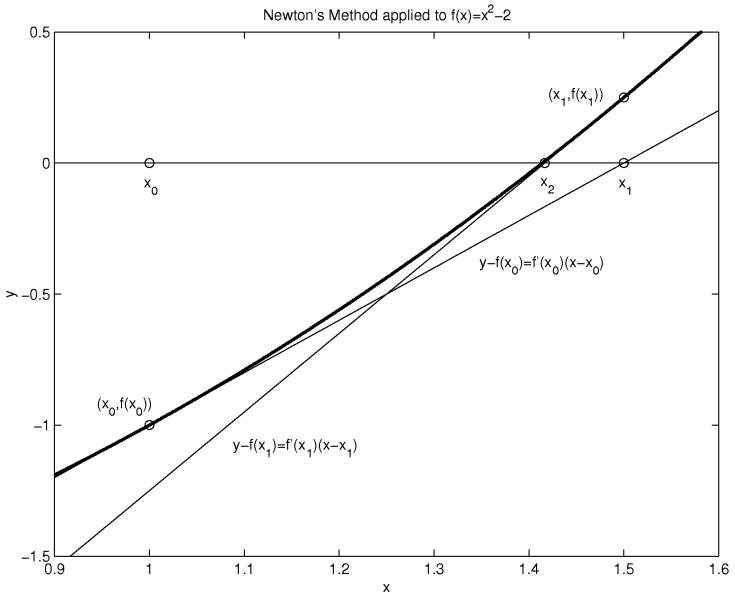

Figure 2.2: Newton's Method applied to $f(x) = x^2 - 2$. The bold curve is the graph of f. The initial iterate x_0 is chosen to be 1. The tangent line of $f(x)$ at the point $(x_0, f(x_0))$ is used to approximate $f(x)$, and it crosses the x-axis at $x_1 = 1.5$, which is much closer to the exact solution than x_0. Then, the tangent line at $(x_1, f(x_1))$ is used to approximate $f(x)$, and it crosses the x-axis at $x_2 = 1.41\bar{6}$, which is already very close to the exact solution.

Example Newton's Method can be used to compute the reciprocal of a number *a without performing any divisions*. The solution, $1/a$, satisfies the equation $f(x) = 0$, where

$$f(x) = a - \frac{1}{x}.$$

Since

$$f'(x) = \frac{1}{x^2},$$

it follows that in Newton's Method, we can obtain the next iterate x_{n+1}

from the previous iterate x_n by

$$x_{n+1} = x_n - \frac{a - 1/x_n}{1/x_n^2} = x_n - \frac{a}{1/x_n}^2 + \frac{1/x_n}{1/x_n^2} = 2x_n - ax_n^2.$$

Note that no divisions are necessary to obtain x_{n+1} from x_n. This iteration was actually used on older IBM computers to implement division in hardware.

We use this iteration to compute the reciprocal of $a = 12$. Choosing our starting iterate to be 0.1, we compute the next several iterates as follows:

$$\begin{aligned} x_1 &= 2(0.1) - 12(0.1)^2 = 0.08 \\ x_2 &= 2(0.12) - 12(0.12)^2 = 0.0832 \\ x_3 &= 0.0833312 \\ x_4 &= 0.08333333333279 \\ x_5 &= 0.08333333333333. \end{aligned}$$

We conclude that 0.08333333333333 is an accurate approximation to the correct solution.

Now, suppose we repeat this process, but with an initial iterate of $x_0 = 1$. Then, we have

$$\begin{aligned} x_1 &= 2(1) - 12(1)^2 = -10 \\ x_2 &= 2(-6) - 12(-6)^2 = -1220 \\ x_3 &= 2(-300) - 12(-300)^2 = -17863240 \end{aligned}$$

It is clear that this sequence of iterates is not going to converge to the correct solution. In general, for this iteration to converge to the reciprocal of a, the initial iterate x_0 must be chosen so that $0 < x_0 < 2/a$. This condition guarantees that the next iterate x_1 will at least be positive. The contrast between the two choices of x_0 are illustrated in Figure 2.3. □

2.4.1 The Secant Method

One drawback of Newton's method is that it is necessary to evaluate $f'(x)$ at various points, which may not be practical for some choices of f. The *secant method* avoids this issue by using a finite difference to approximate the derivative. As a result, $f(x)$ is approximated by a *secant line* through two points on the graph of f, rather than a tangent line through one point on the graph.

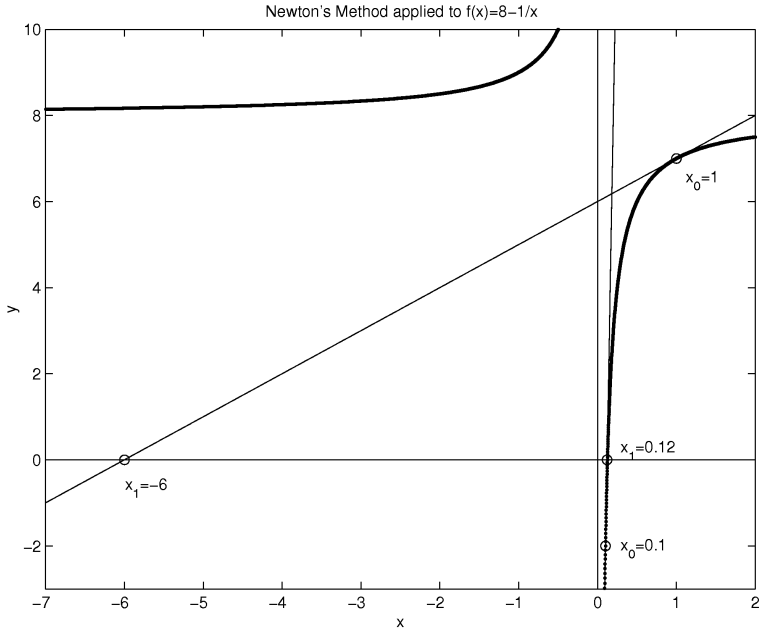

Figure 2.3: Newton's Method used to compute the reciprocal of 8 by solving the equation $f(x) = 8 - 1/x = 0$. When $x_0 = 0.1$, the tangent line of $f(x)$ at $(x_0, f(x_0))$ crosses the x-axis at $x_1 = 0.12$, which is close to the exact solution. When $x_0 = 1$, the tangent line crosses the x-axis at $x_1 = -6$, which causes searcing to continue on the wrong portion of the graph, so the sequence of iterates does not converge to the correct solution.

Since a secant line is defined using two points on the graph of $f(x)$, as opposed to a tangent line that requires information at only one point on the graph, it is necessary to choose two initial iterates x_0 and x_1. Then, as in Newton's method, the next iterate x_2 is then obtained by computing the x-value at which the secant line passing through the points $(x_0, f(x_0))$ and $(x_1, f(x_1))$ has a y-coordinate of zero. This yields the equation

$$\frac{f(x_1) - f(x_0)}{x_1 - x_0}(x_2 - x_0) + f(x_0) = 0$$

which has the solution

$$x_2 = x_0 - \frac{f(x_0)(x_1 - x_0)}{f(x_1) - f(x_0)}$$

which can be rewritten as follows:

$$\begin{aligned} x_2 &= x_0 - \frac{f(x_0)(x_1 - x_0)}{f(x_1) - f(x_0)} \\ &= x_0 \frac{f(x_1) - f(x_0)}{f(x_1) - f(x_0)} - \frac{f(x_0)(x_1 - x_0)}{f(x_1) - f(x_0)} \\ &= \frac{x_0(f(x_1) - f(x_0)) - f(x_0)(x_1 - x_0)}{f(x_1) - f(x_0)} \\ &= \frac{x_0 f(x_1) - x_0 f(x_0) - x_1 f(x_0) + x_0 f(x_0)}{f(x_1) - f(x_0)} \\ &= \frac{x_0 f(x_1) - x_1 f(x_0)}{f(x_1) - f(x_0)}. \end{aligned}$$

This leads to the following algorithm.

Algorithm (Secant Method) Let $f : \mathbb{R} \to \mathbb{R}$ be a continuous function. The following algorithm computes an approximate solution x^* to the equation $f(x) = 0$.

Choose two initial guesses x_0 and x_1
for $k = 1, 2, 3, \ldots$ **do**
    if $f(x_k)$ is sufficiently small **then**
        $x^* = x_k$
        return x^*
    end
    $x_{k+1} = \frac{x_{k-1} f(x_k) - x_k f(x_{k-1})}{f(x_k) - f(x_{k-1})}$
    if $|x_{k+1} - x_k|$ is sufficiently small **then**
        $x^* = x_{k+1}$
        return x^*
    end
end

Like Newton's method, it is necessary to choose the starting iterate x_0 to be reasonably close to the solution x^*. Convergence is not as rapid as that of Newton's Method, since the secant-line approximation of f is not as accurate as the tangent-line approximation employed by Newton's method.

Example We will use the Secant Method to solve the equation $f(x) = 0$, where $f(x) = x^2 - 2$. This method requires that we choose two initial iterates x_0 and x_1, and then compute subsequent iterates using the formula

$$x_{n+1} = x_{n-1} - \frac{f(x_{n-1})(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}, \quad n = 1, 2, 3, \ldots.$$

We choose $x_0 = 1$ and $x_1 = 1.5$. Applying the above formula, we obtain

$$\begin{aligned} x_2 &= 1.4 \\ x_3 &= 1.41379310344828 \\ x_4 &= 1.41421568627451. \end{aligned}$$

As we can see, the iterates produced by the Secant Method are converging to the exact solution $x^* = \sqrt{2}$, but not as rapidly as those produced by Newton's Method. □

2.4.2 Safeguarded Methods

It is natural to ask whether it is possible to combine the rapid convergence of methods such as Newton's method with "safe" methods such as bisection that are guaranteed to converge. This leads to the concept of *safeguarded methods*, which maintain an interval within which a solution is known to exist, as in bisection, but use a method such as Newton's method to find a solution within that interval. If an iterate falls outside this interval, the safe procedure is used to refine the interval before trying the rapid method.

An example of a safeguarded method is the *method of Regula Falsi*, which is also known as the *method of false position*. It is a modification of the secant method in which the two initial iterates x_0 and x_1 are chosen so that $f(x_0) \cdot f(x_1) < 0$, thus guaranteeing that a solution lies between x_0 and x_1. This condition also guarantees that the next iterate x_2 will lie between x_0 and x_1, as can be seen by applying the Intermediate Value Theorem to the secant line passing through $(x_0, f(x_0))$ and $(x_1, f(x_1))$.

It follows that if $f(x_2) \neq 0$, then a solution must lie between x_0 and x_2, or between x_1 and x_2. In the first scenario, we use the secant line passing through $(x_0, f(x_0))$ and $(x_2, f(x_2))$ to compute the next iterate x_3. Otherwise, we use the secant line passing through $(x_1, f(x_1))$ and $(x_2, f(x_2))$. Continuing in this fashion, we obtain a sequence of smaller and smaller intervals that are guaranteed to contain a solution, as in bisection, but the sizes of these intervals converges to zero much more rapidly.

In fact, regula falsi converges almost as rapidly as the secant method, while ensuring that a solution will be found. We now describe the full algorithm, which is nearly identical to the bisection method. The only substantial difference is that the current interval is divided more judiciously, using the secant line instead of a simple bisection.

Algorithm (Method of Regula Falsi) Let $f : \mathbb{R} \to \mathbb{R}$ be a continuous function that changes sign on the interval (a, b). The following algorithm

computes an approximate solution x^* to the equation $f(x) = 0$.

repeat
    $c = \frac{af(b)-bf(a)}{f(b)-f(a)}$
    if $f(c) = 0$ **or** $b - a$ is sufficiently small **then**
        $x^* = c$
        return x^*
    end
    if $f(a) \cdot f(c) < 0$ **then**
        $b = c$
    else
        $a = c$
    end
end

Example We use the Method of Regula Falsi (False Position) to solve $f(x) = 0$ where $f(x) = x^2 - 2$. First, we must choose two initial guesses x_0 and x_1 such that $f(x)$ changes sign between x_0 and x_1. Choosing $x_0 = 1$ and $x_1 = 1.5$, we see that $f(x_0) = f(1) = -1$ and $f(x_1) = f(1.5) = 0.25$, so these choices are suitable.

Next, we use the Secant Method to compute the next iterate x_2 by determining the point at which the secant line passing through the points $(x_0, f(x_0))$ and $(x_1, f(x_1))$ intersects the line $y = 0$. We have

$$\begin{aligned} x_2 &= x_0 - \frac{f(x_0)(x_1 - x_0)}{f(x_1) - f(x_0)} \\ &= 1 - \frac{(-1)(1.5 - 1)}{0.25 - (-1)} \\ &= 1 + \frac{1.5 - 1}{0.25 + 1} \\ &= 1 + \frac{0.5}{1.25} \\ &= 1.4. \end{aligned}$$

Computing $f(x_2)$, we obtain $f(1.4) = -0.04 < 0$. Since $f(x_2) < 0$ and $f(x_1) > 0$, we can use the Intermediate Value Theorem to conclude that a solution exists in the interval (x_2, x_1). Therefore, we compute x_3 by determining where the secant line through the points $(x_1, f(x_1))$ and $f(x_2, f(x_2))$ intersects the line $y = 0$. Using the formula for the Secant Method, we ob-

tain

$$\begin{aligned} x_3 &= x_1 - \frac{f(x_1)(x_2 - x_1)}{f(x_2) - f(x_1)} \\ &= 1.5 - \frac{(0.25)(1.4 - 1.5)}{-0.04 - 0.25} \\ &= 1.41379. \end{aligned}$$

Since $f(x_3) < 0$ and $f(x_2) < 0$, we do not know that a solution exists in the interval (x_2, x_3). However, we do know that a solution exists in the interval (x_3, x_1), because $f(x_1) > 0$. Therefore, instead of proceeding as in the Secant Method and using the Secant line determined by x_2 and x_3 to compute x_4, we use the secant line determined by x_1 and x_3 to compute x_4.

By the Intermediate Value Theorem, and the fact that this secant line intersects the graph of $f(x)$ at $x = x_1$ and $x = x_3$, we can conclude that x_4 must fall between x_3 and x_1, so then we can continue looking for a solution in the interval (x_3, x_4) or the interval (x_4, x_1), depending on whether $f(x_4)$ is positive or negative. In this sense, the Method of Regula Falsi is very similar to the Bisection Method. The only difference is in how we divide the interval (a, b) that is known to contain a solution. □

2.5 Error Analysis for Iterative Methods

In general, nonlinear equations cannot be solved in a finite sequence of steps. As linear equations can be solved using *direct methods* such as Gaussian elimination, nonlinear equations usually require *iterative methods*. In iterative methods, an approximate solution is refined with each iteration until it is determined to be sufficiently accurate, at which time the iteration terminates. Since it is desirable for iterative methods to converge to the solution as rapidly as possible, it is necessary to be able to measure the speed with which an iterative method converges.

To that end, we assume that an iterative method generates a sequence of iterates $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots$ that converges to the exact solution $\mathbf{x}^*$. Ideally, we would like the error in a given iterate $\mathbf{x}_{k+1}$ to be much smaller than the error in the previous iterate $\mathbf{x}_k$. For example, if the error is raised to a power greater than 1 from iteration to iteration, then, because the error is typically less than 1, it will approach zero very rapidly. This leads to the following definition.

Definition *(Rate of Convergence) Let $\{\mathbf{x}_k\}_{k=0}^{\infty}$ be a sequence in $\mathbb{R}^n$ that converges to $\mathbf{x}^* \in \mathbb{R}^n$ and assume that $\mathbf{x}_k \neq \mathbf{x}^*$ for each k. We say that the*

rate of convergence *of* $\{\mathbf{x}_k\}$ *to* $\mathbf{x}^*$ *is of* **order** r*, with* **asymptotic error constant** C*, if*

$$\lim_{k\to\infty} \frac{\|\mathbf{x}_{k+1} - \mathbf{x}^*\|}{\|\mathbf{x}_k - \mathbf{x}^*\|^r} = C,$$

where $r \geq 1$ *and* $C > 0$*.*

If $r = 1$, we say that convergence is *linear*. If $r = 2$, then the method converges *quadratically*, and if $r = 3$, we say it converges *cubically*, and so on.

Of course, we cannot use the error $\mathbf{e}_k = \mathbf{x}_k - \mathbf{x}^*$ to determine the rate of convergence experimentally because we do not know the solution $\mathbf{x}^*$. Instead, we can use the relative change in successive iterations, but it is advisable to also compute $\|f(\mathbf{x}_k)\|$ after each iteration to ensure that the iterates are actually converging to a solution of $f(\mathbf{x}) = \mathbf{0}$.

In the remainder of this lecture, we will focus on the case where f is a scalar-valued function of a single variable. We will examine the various methods we have discussed for solving $f(x) = 0$ and attempt to determine their rate of convergence analytically. In this case, we can work with the error $e_k = x_k - x^*$, where x_k is the kth iterate that is computed using the given method, and x^* is the exact solution to which the sequence of iterates $\{x_k\}$ converges.

2.5.1 Bisection

For this method, it is easier to determine the rate of convergence if we use a different measure of the error in each iterate x_k. Since each iterate is contained within an interval $[a_k, b_k]$ where $b_k - a_k = 2^{-k}(b - a)$, with $[a, b]$ being the original interval, it follows that we can bound the error $x_k - x^*$ by $e_k = b_k - a_k$. Using this measure, we can easily conclude that bisection converges linearly, with asymptotic error constant 1/2.

2.5.2 Fixed-point Iteration

Suppose that we are using Fixed-point Iteration to solve the equation $g(x) = x$, where g is continuously differentiable on an interval $[a, b]$ Starting with the formula for computing iterates in Fixed-point Iteration,

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

we can use the Mean Value Theorem to obtain

$$e_{k+1} = x_{k+1} - x^* = g(x_k) - g(x^*) = g'(\xi_k)(x_k - x^*) = g'(\xi_k)e_k,$$

where ξ_k lies between x_k and x^*.

It follows that if g maps $[a, b]$ into itself, and $|g'(x)| \leq k < 1$ on (a, b) for some constant k, then for any initial iterate $x_0 \in [a, b]$, Fixed-point Iteration converges linearly with asymptotic error constant $|g'(x^*)|$, since, by the definition of ξ_k and the continuity of g',

$$\lim_{k\to\infty} \frac{|e_{k+1}|}{|e_k|} = \lim_{k\to\infty} |g'(\xi_k)| = |g'(x^*)|.$$

Recall that the conditions we have stated for linear convergence are nearly identical to the conditions for g to have a unique fixed point in $[a, b]$. The only difference is that now, we also require g' to be continuous on $[a, b]$.

Now, suppose that in addition to the previous conditions on g, we assume that $g'(x^*) = 0$, and that g is twice continuously differentiable on $[a, b]$. Then, using Taylor's Theorem, we obtain

$$e_{k+1} = g(x_k) - g(x^*) = g'(x^*)(x_k - x^*) + \frac{1}{2}g''(\xi_k)(x_k - x^*)^2 = \frac{1}{2}g''(\xi_k)e_k^2,$$

where ξ_k lies between x_k and x^*. It follows that for any initial iterate $x_0 \in [a, b]$, Fixed-point Iteration converges at least quadratically, with asymptotic error constant $|g''(x^*)/2|$. This discussion implies the following general result.

Theorem *Let $g(x)$ be a function that is n times continuously differentiable on an interval $[a, b]$. Furthermore, assume that $g(x) \in [a, b]$ for $x \in [a, b]$, and that $|g'(x)| \leq k$ on (a, b) for some constant $k < 1$. If the unique fixed point x^* in $[a, b]$ satisfies*

$$g'(x^*) = g''(x^*) = \cdots = g^{(n-1)}(x^*) = 0,$$

then for any $x_0 \in [a, b]$, Fixed-point Iteration converges to x^* of order n, with asymptotic error constant $|g^{(n)}(x^*)/n!|$.

2.5.3 Newton's Method

Using the same approach as with Fixed-point Iteration, we can determine the convergence rate of Newton's Method applied to the equation $f(x) = 0$, where we assume that f is continuously differentiable near the exact solution x^*, and that f'' exists near x^*. Using Taylor's Theorem, we obtain

$$\begin{aligned} e_{k+1} &= x_{k+1} - x^* \\ &= x_k - \frac{f(x_k)}{f'(x_k)} - x^* \end{aligned}$$

$$\begin{aligned}
&= e_k - \frac{f(x_k)}{f'(x_k)} \\
&= e_k - \frac{1}{f'(x_k)}\left[f(x^*) - f'(x_k)(x^* - x_k) - \frac{1}{2}f''(\xi_k)(x_k - x^*)^2\right] \\
&= e_k + \frac{1}{f'(x_k)}\left[f'(x_k)(x^* - x_k) + \frac{1}{2}f''(\xi_k)(x_k - x^*)^2\right] \\
&= e_k + \frac{1}{f'(x_k)}\left[-f'(x_k)e_k + \frac{1}{2}f''(\xi_k)e_k^2\right] \\
&= e_k - e_k + \frac{f''(\xi_k)}{2f'(x_k)}e_k^2 \\
&= \frac{f''(\xi_k)}{2f'(x_k)}e_k^2
\end{aligned}$$

where ξ_k is between x_k and x^*. We conclude that if $f'(x^*) \neq 0$, then Newton's Method converges quadratically, with asymptotic error constant $|f''(x^*)/2f'(x^*)|$. It is easy to see from this constant, however, that if $f'(x^*)$ is very small, or zero, then convergence can be very slow or may not even occur.

Example Suppose that Newton's Method is used to find the solution of $f(x) = 0$, where $f(x) = x^2 - 2$. We examine the error $e_k = x_k - x^*$, where $x^* = \sqrt{2}$ is the exact solution. We have

k	x_k	$\lvert e_k\rvert$
0	1	0.41421356237310
1	1.5	0.08578643762690
2	1.41666666666667	0.00245310429357
3	1.41421568627457	0.00000212390141
4	1.41421356237469	0.00000000000159

We can determine analytically that Newton's Method converges quadratically, and in this example, the asymptotic error constant is $|f''(\sqrt{2})/2f'(\sqrt{2})| \approx 0.35355$. Examining the numbers in the table above, we can see that the number of correct decimal places approximately doubles with each iteration, which is typical of quadratic convergence. Furthermore, we have

$$\frac{|e_4|}{|e_3|^2} \approx 0.35352,$$

so the actual behavior of the error is consistent with the behavior that is predicted by theory. □

2.5.4 The Secant Method

The convergence rate of the Secant Method can be determined using a result, which we will not prove here, stating that if $\{x_k\}_{k=0}^{\infty}$ is the sequence of iterates produced by the Secant Method for solving $f(x) = 0$, and if this sequence converges to a solution x^*, then for k sufficiently large,

$$|x_{k+1} - x^*| \approx S|x_k - x^*||x_{k-1} - x^*|$$

for some constant S.

We assume that $\{x_k\}$ converges to x^* of order α. Then, dividing both sides of the above relation by $|x_k - x^*|^\alpha$, we obtain

$$\frac{|x_{k+1} - x^*|}{|x_k - x^*|^\alpha} \approx S|x_k - x^*|^{1-\alpha}|x_{k-1} - x^*|.$$

Because α is the rate of convergence, the left side must converge to a positive constant C as $k \to \infty$. It follows that the right side must converge to a positive constant as well, as must its reciprocal. In other words, there must exist positive constants C_1 and C_2

$$\frac{|x_k - x^*|}{|x_{k-1} - x^*|^\alpha} \to C_1, \quad \frac{|x_k - x^*|^{\alpha-1}}{|x_{k-1} - x^*|} \to C_2.$$

This can only be the case if there exists a nonzero constant β such that

$$\frac{|x_k - x^*|}{|x_{k-1} - x^*|^\alpha} = \left(\frac{|x_k - x^*|^{\alpha-1}}{|x_{k-1} - x^*|}\right)^\beta,$$

which implies that

$$1 = (\alpha - 1)\beta \quad \text{and} \quad \alpha = \beta.$$

Eliminating β, we obtain the equation

$$\alpha^2 - \alpha - 1 = 0,$$

which has the solutions

$$\alpha_1 = \frac{1 + \sqrt{5}}{2} \approx 1.618, \quad \alpha_2 = \frac{1 - \sqrt{5}}{2} \approx -0.618.$$

Since we must have $\alpha > 1$, the rate of convergence is 1.618.

2.6 Accelerating Convergence

Suppose that a sequence $\{x_k\}_{k=0}^{\infty}$ converges linearly to a limit x^*, in such a way that if k is sufficiently large, then $x_k - x^*$ has the same sign; that is, $\{x_k\}$ converges *monotonically* to x^*. It follows from the linear convergence of $\{x_k\}$ that for sufficiently large k,

$$\frac{x_{k+2} - x^*}{x_{k+1} - x^*} \approx \frac{x_{k+1} - x^*}{x_k - x^*}.$$

Solving for x^* yields

$$x^* \approx x_k - \frac{(x_{k+1} - x_k)^2}{x_{k+2} - 2x_{k+1} + x_k}.$$

Therefore, we can construct an alternative sequence $\{\hat{x}_k\}_{k=0}^{\infty}$, where

$$\hat{x}_k = x_k - \frac{(x_{k+1} - x_k)^2}{x_{k+2} - 2x_{k+1} + x_k},$$

that also converges to x^*. This sequence has the following desirable property.

Theorem *Suppose that the sequence $\{x_k\}_{k=0}^{\infty}$ converges linearly to a limit x^* and that for k sufficiently large, $(x_{k+1} - x^*)(x_k - x^*) > 0$. Then, if the sequence $\{\hat{x}_k\}_{k=0}^{\infty}$ is defined by*

$$\hat{x}_k = x_k - \frac{(x_{k+1} - x_k)^2}{x_{k+2} - 2x_{k+1} + x_k}, \quad k = 0, 1, 2, \ldots,$$

then

$$\lim_{k \to \infty} \frac{\hat{x}_k - x^*}{x^k - x^*} = 0.$$

In other words, the sequence $\{\hat{x}_k\}$ converges to x^* more rapidly than $\{x_k\}$ does.

If we define the *forward difference operator* Δ by

$$\Delta x_k = x_{k+1} - x_k,$$

then

$$\Delta^2 x_k = \Delta(x_{k+1} - x_k) = (x_{k+2} - x_{k+1}) - (x_{k+1} - x_k) = x_{k+2} - 2x_{k+1} + x_k,$$

and therefore $\hat{x}_k$ can be rewritten as

$$\hat{x}_k = x_k - \frac{(\Delta x_k)^2}{\Delta^2 x_k}, \quad k = 0, 1, 2, \ldots$$

For this reason, the method of accelerating the convergence of $\{x_k\}$ by constructing $\{\hat{x}_k\}$ is called *Aitken's Δ^2 Method*.

A slight variation of this method, called *Steffensen's Method*, can be used to accelerate the convergence of Fixed-point Iteration, which, as previously discussed, is linearly convergent. The basic idea is as follows:

1. Choose an initial iterate x_0
2. Compute x_1 and x_2 using Fixed-point Iteration
3. Use Aitken's Δ^2 Method to compute $\hat{x}_0$ from x_0, x_1, and x_2
4. Repeat steps 2 and 3 with $x_0 = \hat{x}_0$.

The principle behind Steffensen's Method is that $\hat{x}_0$ is thought to be a better approximation to the fixed point x^* than x_2, so it should be used as the next iterate for Fixed-point Iteration.

Example We wish to find the unique fixed point of the function $f(x) = \cos x$ on the interval $[0, 1]$. If we use Fixed-point Iteration with $x_0 = 0.5$, then we obtain the following iterates from the formula $x_{k+1} = g(x_k) = \cos(x_k)$. All iterates are rounded to five decimal places.

$$\begin{aligned} x_1 &= 0.87758 \\ x_2 &= 0.63901 \\ x_3 &= 0.80269 \\ x_4 &= 0.69478 \\ x_5 &= 0.76820 \\ x_6 &= 0.71917. \end{aligned}$$

These iterates show little sign of converging, as they are oscillating around the fixed point.

If, instead, we use Fixed-point Iteration with acceleration by Aitken's Δ^2 method, we obtain a new sequence of iterates $\{\hat{x}_k\}$, where

$$\begin{aligned} \hat{x}_k &= x_k - \frac{(\Delta x_k)^2}{\Delta^2 x_k} \\ &= x_k - \frac{(x_{k+1} - x_k)^2}{x_{k+2} - 2x_{k+1} + x_k}, \end{aligned}$$

for $k = 0, 1, 2, \ldots$. The first few iterates of this sequence are

$$\hat{x}_0 = 0.73139$$

$$\begin{aligned}\hat{x}_1 &= 0.73609\\ \hat{x}_2 &= 0.73765\\ \hat{x}_3 &= 0.73847\\ \hat{x}_4 &= 0.73880.\end{aligned}$$

Clearly, these iterates are converging much more rapidly than Fixed-point Iteration, as they are not oscillating around the fixed point, but convergence is still linear.

Finally, we try Steffensen's Method. We begin with the first three iterates of Fixed-point Iteration,

$$x_0^{(0)} = x_0 = 0.5, \quad x_1^{(0)} = x_1 = 0.87758, \quad x_2^{(0)} = x_2 = 0.63901.$$

Then, we use the formula from Aitken's Δ^2 Method to compute

$$x_0^{(1)} = x_0^{(0)} - \frac{(x_1^{(0)} - x_0^{(0)})^2}{x_2^{(0)} - 2x_1^{(0)} + x_0^{(0)}} = 0.73139.$$

We use this value to restart Fixed-point Iteration and compute two iterates, which are

$$x_1^{(1)} = \cos(x_0^{(1)}) = 0.74425, \quad x_2^{(1)} = \cos(x_1^{(1)}) = 0.73560.$$

Repeating this process, we apply the formula from Aitken's Δ^2 Method to the iterates $x_0^{(1)}$, $x_1^{(1)}$ and $x_2^{(1)}$ to obtain

$$x_0^{(2)} = x_0^{(1)} - \frac{(x_1^{(1)} - x_0^{(1)})^2}{x_2^{(1)} - 2x_1^{(1)} + x_0^{(1)}} = 0.739076.$$

Restarting Fixed-point Iteration with $x_0^{(2)}$ as the initial iterate, we obtain

$$x_1^{(2)} = \cos(x_0^{(2)}) = 0.739091, \quad x_2^{(2)} = \cos(x_1^{(2)}) = 0.739081.$$

The most recent iterate $x_2^{(2)}$ is correct to five decimal places.

Using all three methods to compute the fixed point to ten decimal digits of accuracy, we find that Fixed-point Iteration requires 57 iterations, so x_58 must be computed. Aitken's Δ^2 Method requires us to compute 25 iterates of the modified sequence $\{\hat{x}_k\}$, which in turn requires 27 iterates of the sequence $\{x_k\}$, where the first iterate x_0 is given. Steffensen's Method requires us to compute $x_2^{(3)}$, which means that only 11 iterates need to be computed. □

k	x_k	$g(x_k)$
0	0.50000000000000	0.87758256189037
1	0.87758256189037	0.63901249416526
2	0.63901249416526	0.80268510068233
3	0.80268510068233	0.69477802678801
4	0.69477802678801	0.76819583128202
5	0.76819583128202	0.71916544594242
6	0.71916544594242	0.75235575942153
7	0.75235575942153	0.73008106313782
8	0.73008106313782	0.74512034135144
9	0.74512034135144	0.73500630901484
10	0.73500630901484	0.74182652264325
11	0.74182652264325	0.73723572544223
12	0.73723572544223	0.74032965187826
13	0.74032965187826	0.73824623833223
14	0.73824623833223	0.73964996276966
15	0.73964996276966	0.73870453935698
16	0.73870453935698	0.73934145228121
17	0.73934145228121	0.73891244933210
18	0.73891244933210	0.73920144413580
19	0.73920144413580	0.73900677978081
20	0.73900677978081	0.73913791076229
21	0.73913791076229	0.73904958059521
22	0.73904958059521	0.73910908142053
23	0.73910908142053	0.73906900120401
24	0.73906900120401	0.73909599983575
25	0.73909599983575	0.73907781328518
26	0.73907781328518	0.73909006398825
27	0.73909006398825	0.73908181177811
28	0.73908181177811	0.73908737057104
29	0.73908737057104	0.73908362610348
30	0.73908362610348	0.73908614842288

Chapter 3

Interpolation and Polynomial Approximation

3.1 Interpolation

Calculus provides many tools that can be used to understand the behavior of functions, but in most cases it is necessary for these functions to be continuous or differentiable. This presents a problem in most "real" applications, in which functions are used to model relationships between quantities, but our only knowledge of these functions consists of a set of discrete data points, where the data is obtained from measurements. Therefore, we need to be able to construct continuous functions based on discrete data.

The problem of constructing such a continuous function is called *data fitting*. In this lecture, we discuss a special case of data fitting known as *interpolation*, in which the goal is to find a linear combination of n known functions to fit a set of data that imposes n constraints, thus guaranteeing a unique solution that fits the data exactly, rather than approximately. The broader term "constraints" is used, rather than simply "data points", since the description of the data may include additional information such as rates of change or requirements that the fitting function have a certain number of continuous derivatives.

When it comes to the study of functions using calculus, polynomials are particularly simple to work with. Therefore, in this course we will focus on the problem of constructing a polynomial that, in some sense, fits given data. We first discuss some algorithms for computing the unique polynomial $p_n(x)$ of degree n that satisfies $p_n(x_i) = y_i$, $i = 0, \ldots, n$, where the points (x_i, y_i) are given. The points $x_0, x_1, \ldots, x_n$ are called *interpolation points*. The

polynomial $p_n(x)$ is called the *interpolating polynomial* of the data (x_0, y_0), (x_1, y_1), ..., (x_n, y_n). At first, we will assume that the interpolation points are all distinct; this assumption will be relaxed in a later lecture.

3.1.1 Computing the Interpolating Polynomial

If the interpolation points $x_0, \ldots, x_n$ are distinct, then the process of finding a polynomial that passes through the points (x_i, y_i), $i = 0, \ldots, n$, is equivalent to solving a system of linear equations $A\mathbf{x} = \mathbf{b}$ that has a unique solution. However, different algorithms for computing the interpolating polynomial use a different A, since they each use a different basis for the space of polynomials of degree $\leq n$.

The most straightforward method of computing the interpolation polynomial is to form the system $A\mathbf{x} = \mathbf{b}$ where $b_i = y_i$, $i = 0, \ldots, n$, and the entries of A are defined by $a_{ij} = p_j(x_i)$, $i, j = 0, \ldots, n$, where $x_0, x_1, \ldots, x_n$ are the points at which the data $y_0, y_1, \ldots, y_n$ are obtained, and $p_j(x) = x^j$, $j = 0, 1, \ldots, n$. The basis $\{1, x, \ldots, x^n\}$ of the space of polynomials of degree $n+1$ is called the *monomial basis*, and the corresponding matrix A is called the *Vandermonde matrix* for the points $x_0, x_1, \ldots, x_n$. Unfortunately, this matrix can be ill-conditioned, especially when interpolation points are close together.

In *Lagrange interpolation*, the matrix A is simply the identity matrix, by virtue of the fact that the interpolating polynomial is written in the form

$$p_n(x) = \sum_{j=0}^{n} y_j \mathcal{L}_{n,j}(x),$$

where the polynomials $\{\mathcal{L}_{n,j}\}_{j=0}^{n}$ have the property that

$$\mathcal{L}_{n,j}(x_i) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}.$$

The polynomials $\{\mathcal{L}_{n,j}\}$, $j = 0, \ldots, n$, are called the *Lagrange polynomials* for the interpolation points x_0, x_1, ..., x_n. They are defined by

$$\mathcal{L}_{n,j}(x) = \prod_{k=0, k\neq j}^{n} \frac{x - x_k}{x_j - x_k}.$$

As the following result indicates, the problem of polynomial interpolation can be solved using Lagrange polynomials.

Theorem *Let $x_0, x_1, \ldots, x_n$ be $n+1$ distinct numbers, and let $f(x)$ be a function defined on a domain containing these numbers. Then the polynomial defined by*

$$p_n(x) = \sum_{j=0}^{n} f(x_j)\mathcal{L}_{n,j}$$

is the unique polynomial of degree n that satisfies

$$p_n(x_j) = f(x_j), \quad j = 0, 1, \ldots, n.$$

The polynomial $p_n(x)$ is called the *interpolating polynomial* of $f(x)$. We say that $p_n(x)$ *interpolates* $f(x)$ at the points $x_0, x_1, \ldots, x_n$.

Example We will use Lagrange interpolation to find the unique polynomial $p_3(x)$, of degree 3 or less, that agrees with the following data:

i	x_i	y_i
0	-1	3
1	0	-4
2	1	5
3	2	-6

In other words, we must have $p_3(-1) = 3$, $p_3(0) = -4$, $p_3(1) = 5$, and $p_3(2) = -6$.

First, we construct the Lagrange polynomials $\{\mathcal{L}_{3,j}(x)\}_{j=0}^{3}$, using the formula

$$\mathcal{L}_{n,j}(x) = \prod_{i=0, i\neq j}^{3} \frac{(x - x_i)}{(x_j - x_i)}.$$

This yields

$$\begin{aligned}
\mathcal{L}_{3,0}(x) &= \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)} \\
&= \frac{(x-0)(x-1)(x-2)}{(-1-0)(-1-1)(-1-2)} \\
&= \frac{x(x^2-3x+2)}{(-1)(-2)(-3)} \\
&= -\frac{1}{6}(x^3-3x^2+2x) \\
\mathcal{L}_{3,1}(x) &= \frac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)}
\end{aligned}$$

$$
\begin{aligned}
&= \frac{(x+1)(x-1)(x-2)}{(0+1)(0-1)(0-2)} \\
&= \frac{(x^2-1)(x-2)}{(1)(-1)(-2)} \\
&= \frac{1}{2}(x^3-2x^2-x+2) \\
\mathcal{L}_{3,2}(x) &= \frac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)} \\
&= \frac{(x+1)(x-0)(x-2)}{(1+1)(1-0)(1-2)} \\
&= \frac{x(x^2-x-2)}{(2)(1)(-1)} \\
&= -\frac{1}{2}(x^3-x^2-2x) \\
\mathcal{L}_{3,3}(x) &= \frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)} \\
&= \frac{(x+1)(x-0)(x-1)}{(2+1)(2-0)(2-1)} \\
&= \frac{x(x^2-1)}{(3)(2)(1)} \\
&= \frac{1}{6}(x^3-x).
\end{aligned}
$$

By substituting x_i for x in each Lagrange polynomial $\mathcal{L}_{3,j}(x)$, for $j = 0, 1, 2, 3$, it can be verified that

$$\mathcal{L}_{3,j}(x_i) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}.$$

It follows that the Lagrange interpolating polynomial $p_3(x)$ is given by

$$
\begin{aligned}
p_3(x) &= \sum_{j=0}^{3} y_j \mathcal{L}_{3,j}(x) \\
&= y_0\mathcal{L}_{3,0}(x) + y_1\mathcal{L}_{3,1}(x) + y_2\mathcal{L}_{3,2}(x) + y_3\mathcal{L}_{3,3}(x) \\
&= (3)\left(-\frac{1}{6}\right)(x^3-3x^2+2x) + (-4)\frac{1}{2}(x^3-2x^2-x+2) + (5)\left(-\frac{1}{2}\right)(x^3-x^2-2x) + \\
&\quad (-6)\frac{1}{6}(x^3-x) \\
&= -\frac{1}{2}(x^3-3x^2+2x) + (-2)(x^3-2x^2-x+2) - \frac{5}{2}(x^3-x^2-2x) - (x^3-x)
\end{aligned}
$$

$$\begin{aligned} &= \left(-\frac{1}{2} - 2 - \frac{5}{2} - 1\right) x^3 + \left(\frac{3}{2} + 4 + \frac{5}{2}\right) x^2 + (-1 + 2 + 5 + 1)\, x - 4 \\ &= -6x^3 + 8x^2 + 7x - 4. \end{aligned}$$

Substituting each x_i, for $i = 0, 1, 2, 3$, into $p_3(x)$, we can verify that we obtain $p_3(x_i) = y_i$ in each case. □

3.1.2 Interpolation Error

In some applications, the interpolating polynomial $p_n(x)$ is used to fit a known function $f(x)$ at the points $x_0, \ldots, x_n$, usually because $f(x)$ is not feasible for tasks such as differentiation or integration that are easy for polynomials, or because it is not easy to evaluate $f(x)$ at points other than the interpolation points. In such an application, it is possible to determine how well $p_n(x)$ approximates $f(x)$. From Taylor's theorem, we obtain the following result.

Theorem *(Interpolation error) If f is $n+1$ times continuously differentiable on $[a, b]$, and $p_n(x)$ is the unique polynomial of degree n that interpolates $f(x)$ at the $n + 1$ distinct points $x_0, x_1, \ldots, x_n$, then for each $x \in [a, b]$,*

$$f(x) - p_n(x) = \prod_{j=0}^{n} (x - x_j) \frac{f^{(n+1)}(\xi(x))}{(n+1)!},$$

where $\xi(x) \in [a, b]$.

It is interesting to note that the error closely resembles the Taylor remainder $R_n(x)$.

Is it possible to choose the interpolation points so that the error is minimized? To answer this question, we introduce the *Chebyshev polynomials*

$$T_k(x) = \cos(k \cos^{-1}(x)),$$

which satisfy the three-term recurrence relation

$$T_{k+1}(x) = 2xT_k(x) - T_{k-1}(x), \quad T_0(x) \equiv 1, \quad T_1(x) \equiv x.$$

These polynomials have the property that $|T_k(x)| \leq 1$ on the interval $[-1, 1]$, while they grow rapidly outside of this interval. Furthermore, the roots of these polynomials lie within the interval $[-1, 1]$. Therefore, if the interpolation points $x_0, x_1, \ldots, x_n$ are chosen to be the images of the roots of the

$(n+1)$st-degree Chebyshev polynomial under a linear transformation that maps $[-1,1]$ to $[a,b]$, then it follows that

$$\left|\prod_{j=0}^{n}(x-x_j)\right| \leq \frac{1}{2^n}, \quad x \in [a,b].$$

Therefore, the error in interpolating $f(x)$ by an nth-degree polynomial is determined entirely by $f^{(n+1)}$.

3.1.3 Neville's Method

While the Lagrange polynomials are easy to compute, they are difficult to differentiate or integrate. Furthermore, if new interpolation points are added, all of the Lagrange polynomials must be recomputed. Unfortunately, it is not uncommon, in practice, to add to an existing set of interpolation points. It may be determined after computing the kth-degree interpolating polynomial $p_k(x)$ of a function $f(x)$ that $p_k(x)$ is not a sufficiently accurate approximation of $f(x)$ on some domain. Therefore, an interpolating polynomial of higher degree must be computed, which requires additional interpolation points.

To address these issues, we consider the problem of computing the interpolating polynomial *recursively*. More precisely, let $k > 0$, and let $p_k(x)$ be the polynomial of degree k that interpolates the function $f(x)$ at the points $x_0, x_1, \ldots, x_k$. Ideally, we would like to be able to obtain $p_k(x)$ from polynomials of degree $k-1$ that interpolate $f(x)$ at points chosen from among $x_0, x_1, \ldots, x_k$. The following result shows that this is possible.

Theorem *Let n be a positive integer, and let $f(x)$ be a function defined on a domain containing the $n+1$ distinct points $x_0, x_1, \ldots, x_n$, and let $p_n(x)$ be the polynomial of degree n that interpolates $f(x)$ at the points $x_0, x_1, \ldots, x_n$. For each $i = 0, 1, \ldots, n$, we define $p_{n-1,i}(x)$ to be the polynomial of degree $n-1$ that interpolates $f(x)$ at the points $x_0, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$. If i and j are distinct nonnegative integers not exceeding n, then*

$$p_n(x) = \frac{(x-x_j)p_{n-1,j}(x) - (x-x_i)p_{n-1,i}(x)}{x_i - x_j}.$$

This result leads to an algorithm called *Neville's Method* that computes the value of $p_n(x)$ at a given point using the values of lower-degree interpolating polynomials at x. We now describe this algorithm in detail.

Algorithm Let $x_0, x_1, \ldots, x_n$ be distinct numbers, and let $f(x)$ be a function defined on a domain containing these numbers. Given a number x^*, the following algorithm computes $y^* = p_n(x^*)$, where $p_n(x)$ is the nth interpolating polynomial of $f(x)$ that interpolates $f(x)$ at the points $x_0, x_1, \ldots, x_n$.

for $j = 0$ **to** n **do**
    $Q_j = f(x_j)$
end
for $j = 1$ **to** n **do**
    for $k = n$ **to** j **do**
        $Q_k = [(x - x_k)Q_{k-1} - (x - x_{k-j})Q_k]/(x_{k-j} - x_k)$
    end
end
$y^* = Q_n$

At the jth iteration of the outer loop, the number Q_k, for $k = n, n-1, \ldots, j$, represents the value at x of the polynomial that interpolates $f(x)$ at the points $x_k, x_{k-1}, \ldots, x_{k-j}$.

In the next lecture, we will use the preceding theorem to compute the polynomial $p_n(x)$ itself, rather than its value at a given point. This will yield an alternative method of constructing the interpolating polynomial that is more suitable for tasks such as differentiation, integration, and inclusion of additional interpolation points.

3.2 Divided Differences

In the previous lecture, we learned how to compute the value of an interpolating polynomial at a given point, using Neville's Method. However, the theorem that serves as the basis for Neville's Method can easily be used to compute the interpolating polynomial itself. The basic idea is to represent interpolating polynomials using the *Newton form*, which uses linear factors involving the interpolation points, instead of monomials of the form x^j.

Recall that if the interpolation points $x_0, \ldots, x_n$ are distinct, then the process of finding a polynomial that passes through the points (x_i, y_i), $i = 0, \ldots, n$, is equivalent to solving a system of linear equations $A\mathbf{x} = \mathbf{b}$ that has a unique solution. The matrix A is determined by the choice of basis for the space of polynomials of degree n or less. Each entry a_{ij} of A is the value of the jth polynomial in the basis at the point x_i.

In *Newton interpolation*, the matrix A is upper triangular, and the basis functions are defined to be the set $\{\mathcal{N}_j(x)\}_{j=0}^n$, where

$$\mathcal{N}_0(x) = 1, \quad \mathcal{N}_j(x) = \prod_{k=0}^{j-1}(x - x_k), \quad j = 1, \ldots, n.$$

The advantage of Newton interpolation is that the interpolating polynomial is easily updated as interpolation points are added, since the basis functions $\{\mathcal{N}_j(x)\}$, $j = 0, \ldots, n$, do not change from the addition of the new points.

The coefficients c_j of the Newton interpolating polynomial

$$p_n(x) = \sum_{j=0}^{n} c_j \mathcal{N}_j(x)$$

are given by

$$c_j = f[x_0, \ldots, x_j]$$

where $f[x_0, \ldots, x_j]$ denotes the *divided difference* of $x_0, \ldots, x_j$. The divided difference is defined as follows:

$$\begin{aligned} f[x_i] &= y_i, \\ f[x_i, x_{i+1}] &= \frac{y_{i+1} - y_i}{x_{i+1} - x_i}, \\ f[x_i, x_{i+1}, \ldots, x_{i+k}] &= \frac{f[x_{i+1}, \ldots, x_{i+k}] - f[x_i, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}. \end{aligned}$$

This definition implies that for each nonnegative integer j, the divided difference $f[x_0, x_1, \ldots, x_j]$ only depends on the interpolation points $x_0, x_1, \ldots, x_j$ and the value of $f(x)$ at these points. It follows that the addition of new interpolation points does not change the coefficients $c_0, \ldots, c_n$. Specifically, we have

$$p_{n+1}(x) = p_n(x) + \frac{y_{n+1} - p_n(x_{n+1})}{\mathcal{N}_{n+1}(x_{n+1})}\mathcal{N}_{n+1}(x).$$

This ease of updating makes Newton interpolation the most commonly used method of obtaining the interpolating polynomial.

The following result shows how the Newton interpolating polynomial bears a resemblance to a Taylor polynomial.

Theorem *Let f be n times continuously differentiable on $[a, b]$, and let $x_0, x_1, \ldots, x_n$ be distinct points in $[a, b]$. Then there exists a number $\xi \in [a, b]$ such that*

$$f[x_0, x_1, \ldots, x_n] = \frac{f^{(n)}(\xi)}{n!}.$$

3.2.1 Computing the Newton Interpolating Polynomial

We now describe in detail how to compute the coefficients $c_j = f[x_0, x_1, \ldots, x_j]$ of the Newton interpolating polynomial $p_n(x)$, and how to evaluate $p_n(x)$ efficiently using these coefficients.

The computation of the coefficients proceeds by filling in the entries of a *divided-difference table*. This is a triangular table consisting of $n+1$ columns, where n is the degree of the interpolating polynomial to be computed. For $j = 0, 1, \ldots, n$, the jth column contains $n - j$ entries, which are the divided differences $f[x_k, x_{k+1}, \ldots, x_{k+j}]$, for $k = 0, 1, \ldots, n - j$.

We construct this table by filling in the $n + 1$ entries in column 0, which are the trivial divided differences $f[x_j] = f(x_j)$, for $j = 0, 1, \ldots, n$. Then, we use the recursive definition of the divided differences to fill in the entries of subsequent columns. Once the construction of the table is complete, we can obtain the coefficients of the Newton interpolating polynomial from the first entry in each column, which is $f[x_0, x_1, \ldots, x_j]$, for $j = 0, 1, \ldots, n$.

In a practical implementation of this algorithm, we do not need to store the entire table, because we only need the first entry in each column. Because each column has one fewer entry than the previous column, we can overwrite all of the other entries that we do not need. The following algorithm implements this idea.

Algorithm (Divided-Difference Table) Given n distinct interpolation points $x_0, x_1, \ldots, x_n$, and the values of a function $f(x)$ at these points, the following algorithm computes the coefficients $c_j = f[x_0, x_1, \ldots, x_j]$ of the Newton interpolating polynomial.

```
for i = 0, 1, ..., n do
    d_i = f(x_i)
end
for i = 1, 2, ..., n do
    for j = n, n − 1, ..., i do
        d_j = (d_j − d_{j−1})/(x_j − x_{j−i})
    end
end
```

Example We will use Newton interpolation to construct the third-degree polynomial $p_3(x)$ that fits the data

i	x_i	$f(x_i)$
0	-1	3
1	0	-4
2	1	5
3	2	-6

In other words, we must have $p_3(-1) = 3$, $p_3(0) = -4$, $p_3(1) = 5$, and $p_3(2) = -6$.

First, we construct the *divided-difference table* from this data. The divided differences in the table are computed as follows:

$$\begin{aligned}
f[x_0] &= f(x_0) \\
&= 3 \\
f[x_1] &= f(x_1) \\
&= -4 \\
f[x_2] &= f(x_2) \\
&= 5 \\
f[x_3] &= f(x_3) \\
&= -6 \\
f[x_0, x_1] &= \frac{f[x_1] - f[x_0]}{x_1 - x_0} \\
&= \frac{-4 - 3}{0 - (-1)} \\
&= -7 \\
f[x_1, x_2] &= \frac{f[x_2] - f[x_1]}{x_2 - x_1} \\
&= \frac{5 - (-4)}{1 - 0} \\
&= 9 \\
f[x_2, x_3] &= \frac{f[x_3] - f[x_2]}{x_3 - x_2} \\
&= \frac{-6 - 5}{2 - 1} \\
&= -11 \\
f[x_0, x_1, x_2] &= \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} \\
&= \frac{9 - (-7)}{1 - (-1)} \\
&= 8
\end{aligned}$$

$$\begin{aligned} f[x_1, x_2, x_3] &= \frac{f[x_2, x_3] - f[x_1, x_2]}{x_3 - x_1} \\ &= \frac{-11 - 9}{2 - 0} \\ &= -10 \\ f[x_0, x_1, x_2, x_3] &= \frac{f[x_1, x_2, x_3] - f[x_0, x_1, x_2]}{x_3 - x_0} \\ &= \frac{-10 - 8}{2 - (-1)} \\ &= -6 \end{aligned}$$

The resulting divided-difference table is

$x_0 = -1$	$f[x_0] = 3$			
		$f[x_0, x_1] = -7$		
$x_1 = 0$	$f[x_1] = -4$		$f[x_0, x_1, x_2] = 8$	
		$f[x_1, x_2] = 9$		$f[x_0, x_1, x_2, x_3] = -6$
$x_2 = 1$	$f[x_2] = 5$		$f[x_1, x_2, x_3] = -10$	
		$f[x_2, x_3] = -11$		
$x_3 = 2$	$f[x_3] = -6$			

It follows that the interpolating polynomial $p_3(x)$ can be obtained using the *Newton divided-difference formula* as follows:

$$\begin{aligned} p_3(x) &= \sum_{j=0}^{3} f[x_0, \ldots, x_j] \prod_{i=0}^{j-1} (x - x_i) \\ &= f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \\ &\quad f[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2) \\ &= 3 - 7(x + 1) + 8(x + 1)x - 6(x + 1)x(x - 1). \end{aligned}$$

We see that Newton interpolation produces an interpolating polynomial that is in the *Newton form*, with centers $x_0 = -1$, $x_1 = 0$, and $x_2 = 1$. □

Once the coefficients have been computed, we can use *nested multiplication* to evaluate the resulting interpolating polynomial, which is represented using the *Newton divided-difference formula*

$$p_n(x) = \sum_{j=0}^{n} c_j \mathcal{N}_j(x)$$

$$\begin{aligned} &= \sum_{j=0}^{n} f[x_0, x_1, \ldots, x_j] \prod_{i=0}^{j-1} (x - x_i) \\ &= f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \\ &\quad f[x_0, x_1, \ldots, x_n](x - x_0)(x - x_1) \cdots (x - x_{n-1}). \end{aligned}$$

Algorithm (Nested Multiplication) Given n distinct interpolation points $x_0, x_1, \ldots, x_n$ and the coefficients $c_j = f[x_0, x_1, \ldots, x_j]$ of the Newton interpolating polynomial $p_n(x)$, the following algorithm computes $y = p_n(x)$ for a given real number x.

$y = c_n$
for $i = n - 1, n - 2, \ldots, 0$ **do**
    $y = c_i + (x - x_i)y$
end

It can be seen that this algorithm closely resembles *Horner's Method*, which is a special case of nested multiplication that works with the power form of a polynomial, whereas nested multiplication works with the more general Newton form.

Example Consider the interpolating polynomial obtained in the previous example,

$$p_3(x) = 3 - 7(x + 1) + 8(x + 1)x - 6(x + 1)x(x - 1).$$

We will use *nested multiplication* to write this polynomial in the *power form*

$$p_3(x) = b_3x^3 + b_2x^2 + b_1x + b_0.$$

This requires repeatedly applying nested multiplication to a polynomial of the form

$$p(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + c_3(x - x_0)(x - x_1)(x - x_2),$$

and for each application it will perform the following steps,

$$\begin{aligned} b_3 &= c_3 \\ b_2 &= c_2 + (z - x_2)b_3 \\ b_1 &= c_1 + (z - x_1)b_2 \\ b_0 &= c_0 + (z - x_0)b_1, \end{aligned}$$

where, in this example, we will set $z = 0$ each time.

The numbers b_0, b_1, b_2 and b_3 computed by the algorithm are the coefficients of $p(x)$ in the Newton form, with the centers x_0, x_1 and x_2 changed to z, x_0 and x_1; that is,

$$p(x) = b_0 + b_1(x - z) + b_2(x - z)(x - x_0) + b_3(x - z)(x - x_0)(x - x_1).$$

It follows that $b_0 = p(z)$, which is why this algorithm is the preferred method for evaluating a polynomial in Newton form at a given point z.

It should be noted that the algorithm can be derived by writing $p(x)$ in the *nested form*

$$p(x) = c_0 + (x - x_0)[c_1 + (x - x_1)[c_2 + (x - x_2)c_3]]$$

and computing $p(z)$ as follows:

$$\begin{aligned} p(z) &= c_0 + (z - x_0)[c_1 + (z - x_1)[c_2 + (z - x_2)c_3]] \\ &= c_0 + (z - x_0)[c_1 + (z - x_1)[c_2 + (z - x_2)b_3]] \\ &= c_0 + (z - x_0)[c_1 + (z - x_1)b_2] \\ &= c_0 + (z - x_0)b_1 \\ &= b_0. \end{aligned}$$

Initially, we have

$$p(x) = 3 - 7(x + 1) + 8(x + 1)x - 6(x + 1)x(x - 1),$$

so the coefficients of $p(x)$ in this Newton form are

$$c_0 = 3, \quad c_1 = -7, \quad c_2 = 8, \quad c_3 = -6,$$

with the centers

$$x_0 = -1, \quad x_1 = 0, \quad x_2 = 1.$$

Applying nested multiplication to these coefficients and centers, with $z = 0$, yields

$$\begin{aligned} b_3 &= -6 \\ b_2 &= 8 + (0 - 1)(-6) \\ &= 14 \\ b_1 &= -7 + (0 - 0)(14) \\ &= -7 \\ b_0 &= 3 + (0 - (-1))(-7) \\ &= -4. \end{aligned}$$

It follows that

$$\begin{aligned} p(x) &= -4 + (-7)(x-0) + 14(x-0)(x-(-1)) + (-6)(x-0)(x-(-1))(x-0) \\ &= -4 - 7x + 14x(x+1) - 6x^2(x+1), \end{aligned}$$

and the centers are now 0, -1 and 0.

For the second application of nested multiplication, we have

$$p(x) = -4 - 7x + 14x(x+1) - 6x^2(x+1),$$

so the coefficients of $p(x)$ in this Newton form are

$$c_0 = -4, \quad c_1 = -7, \quad c_2 = 14, \quad c_3 = -6,$$

with the centers

$$x_0 = 0, \quad x_1 = -1, \quad x_2 = 0.$$

Applying nested multiplication to these coefficients and centers, with $z = 0$, yields

$$\begin{aligned} b_3 &= -6 \\ b_2 &= 14 + (0-0)(-6) \\ &= 14 \\ b_1 &= -7 + (0-(-1))(14) \\ &= 7 \\ b_0 &= -4 + (0-0)(7) \\ &= -4. \end{aligned}$$

It follows that

$$\begin{aligned} p(x) &= -4 + 7(x-0) + 14(x-0)(x-0) + (-6)(x-0)(x-0)(x-(-1)) \\ &= -4 + 7x + 14x^2 - 6x^2(x+1), \end{aligned}$$

and the centers are now 0, 0 and -1.

For the third and final application of nested multiplication, we have

$$p(x) = -4 + 7x + 14x^2 - 6x^2(x+1),$$

so the coefficients of $p(x)$ in this Newton form are

$$c_0 = -4, \quad c_1 = 7, \quad c_2 = 14, \quad c_3 = -6,$$

with the centers

$$x_0 = 0, \quad x_1 = 0, \quad x_2 = -1.$$

Applying nested multiplication to these coefficients and centers, with $z = 0$, yields

$$\begin{aligned} b_3 &= -6 \\ b_2 &= 14 + (0 - (-1))(-6) \\ &= 8 \\ b_1 &= 7 + (0 - 0)(8) \\ &= 7 \\ b_0 &= -4 + (0 - 0)(7) \\ &= 1. \end{aligned}$$

It follows that

$$\begin{aligned} p(x) &= -4 + 7(x - 0) + 8(x - 0)(x - 0) + (-6)(x - 0)(x - 0)(x - 0) \\ &= -4 + 7x + 8x^2 - 6x^3, \end{aligned}$$

and the centers are now 0, 0 and 0. Since all of the centers are equal to zero, the polynomial is now in the power form.

It should be noted that this is not the most efficient way to convert the Newton form of $p(x)$ to the power form. To see this, we observe that after one application of nested multiplication, we have

$$\begin{aligned} p(x) &= b_0 + b_1(x - z) + b_2(x - z)(x - x_0) + b_3(x - z)(x - x_0)(x - x_1) \\ &= b_0 + (x - z)[b_1 + b_2(x - x_0) + b_3(x - x_0)(x - x_1)]. \end{aligned}$$

Therefore, we can apply nested multiplication to the second-degree polynomial

$$q(x) = b_1 + b_2(x - x_0) + b_3(x - x_0)(x - x_1),$$

which is the quotient obtained by dividing $p(x)$ by $(x - z)$. Because

$$p(x) = b_0 + (x - z)q(x),$$

it follows that once we have changed all of the centers of $q(x)$ to be equal to z, then all of the centers of $p(x)$ will be equal to z as well. In summary, we can convert a polynomial of degree n from Newton form to power form by applying nested multiplication n times, where the jth application is to a polynomial of degree $n - j + 1$, for $j = 1, 2, \ldots, n$.

Since the coefficients of the appropriate Newton form of each of these polynomials of successively lower degree are computed by the nested multiplication algorithm, it follows that we can implement this more efficient procedure simply by proceeding exactly as before, except that during the jth application of nested multiplication, we do not compute the coefficients $b_0, b_1, \ldots, b_{j-2}$, because they will not change anyway, as can be seen from the previous computations. For example, in the second application, we did not need to compute b_0, and in the third, we did not need to compute b_0 and b_1. □

3.3 Interpolation Using Equally Spaced Points

Suppose that the interpolation points $x_0, x_1, \ldots, x_n$ are equally spaced; that is, $x_i = x_0 + ih$ for some positive number h. In this case, the Newton interpolating polynomial can be simplified, since the denominators of all of the divided differences can be expressed in terms of the spacing h. If we recall the *forward difference operator* Δ, defined by

$$\Delta x_k = x_{k+1} - x_k,$$

where $\{x_k\}$ is any sequence, then the divided differences $f[x_0, x_1, \ldots, x_k]$ are given by

$$f[x_0, x_1, \ldots, x_k] = \frac{1}{k!h^k}\Delta^k f(x_0).$$

The interpolating polynomial can then be described by the *Newton forward-difference formula*

$$p_n(x) = f[x_0] + \sum_{k=1}^{n} \left(\begin{array}{c} s \\ k \end{array} \right) \Delta^k f(x_0),$$

where the new variable s is related to x by

$$s = \frac{x - x_0}{h},$$

and the *extended binomial coefficient* $\left(\begin{array}{c} s \\ k \end{array} \right)$ is defined by

$$\left(\begin{array}{c} s \\ k \end{array} \right) = \frac{s(s-1)(s-2)\cdots(s-k+1)}{k!},$$

where k is a nonnegative integer.

Example We will use the *Newton forward-difference formula*

$$p_n(x) = f[x_0] + \sum_{k=1}^{n} \binom{s}{k} \Delta^k f(x_0)$$

to compute the interpolating polynomial $p_3(x)$ that fits the data

i	x_i	$f(x_i)$
0	-1	3
1	0	-4
2	1	5
3	2	-6

In other words, we must have $p_3(-1) = 3$, $p_3(0) = -4$, $p_3(1) = 5$, and $p_3(2) = -6$. Note that the interpolation points $x_0 = -1$, $x_1 = 0$, $x_2 = 1$ and $x_3 = 2$ are equally spaced, with spacing $h = 1$.

To apply the forward-difference formula, we define $s = (x - x_0)/h = x + 1$ and compute

$$\begin{aligned}
\binom{s}{1} &= s \\
&= x + 1, \\
\binom{s}{2} &= \frac{s(s-1)}{2} \\
&= \frac{x(x+1)}{2}, \\
\binom{s}{3} &= \frac{s(s-1)(s-2)}{6} \\
&= \frac{(x+1)x(x-1)}{6}, \\
f[x_0] &= f(x_0) \\
&= 3, \\
\Delta f(x_0) &= f(x_1) - f(x_0) \\
&= -4 - 3 \\
&= -7, \\
\Delta^2 f(x_0) &= \Delta(\Delta f(x_0)) \\
&= \Delta[f(x_1) - f(x_0)] \\
&= [f(x_2) - f(x_1)] - [f(x_1) - f(x_0)] \\
&= f(x_2) - 2f(x_1) + f(x_0)
\end{aligned}$$

$$\begin{aligned}
&= 5 - 2(-4) + 3, \\
&= 16 \\
\Delta^3 f(x_0) &= \Delta(\Delta^2 f(x_0)) \\
&= \Delta[f(x_2) - 2f(x_1) + f(x_0)] \\
&= [f(x_3) - f(x_2)] - 2[f(x_2) - f(x_1)] + [f(x_1) - f(x_0)] \\
&= f(x_3) - 3f(x_2) + 3f(x_1) - f(x_0) \\
&= -6 - 3(5) + 3(-4) - 3 \\
&= -36.
\end{aligned}$$

It follows that

$$\begin{aligned}
p_3(x) &= f[x_0] + \sum_{k=1}^{3} \left(\begin{array}{c} s \\ k \end{array} \right) \Delta^k f(x_0) \\
&= 3 + \left(\begin{array}{c} s \\ 1 \end{array} \right) \Delta f(x_0) + \left(\begin{array}{c} s \\ 1 \end{array} \right) \Delta^2 f(x_0) + \left(\begin{array}{c} s \\ 2 \end{array} \right) \Delta^3 f(x_0) \\
&= 3 + (x+1)(-7) + \frac{x(x+1)}{2} 16 + \frac{(x+1)x(x-1)}{6}(-36) \\
&= 3 - 7(x+1) + 8(x+1)x - 6(x+1)x(x-1).
\end{aligned}$$

Note that the forward-difference formula computes the same form of the interpolating polynomial as the Newton divided-difference formula. □

If we define the *backward difference operator* ∇ by

$$\nabla x_k = x_k - x_{k-1},$$

for any sequence $\{x_k\}$, then we obtain the *Newton backward-difference formula*

$$p_n(x) = f[x_n] + \sum_{k=1}^{n} (-1)^k \left(\begin{array}{c} -s \\ k \end{array} \right) \nabla^k f(x_n),$$

where $s = (x - x_n)/h$, and the preceding definition of the extended binomial coefficient applies.

Example We will use the *Newton backward-difference formula*

$$p_n(x) = f[x_n] + \sum_{k=1}^{n} (-1)^k \left(\begin{array}{c} -s \\ k \end{array} \right) \nabla^k f(x_n)$$

to compute the interpolating polynomial $p_3(x)$ that fits the data

i	x_i	$f(x_i)$
0	-1	3
1	0	-4
2	1	5
3	2	-6

In other words, we must have $p_3(-1) = 3$, $p_3(0) = -4$, $p_3(1) = 5$, and $p_3(2) = -6$. Note that the interpolation points $x_0 = -1$, $x_1 = 0$, $x_2 = 1$ and $x_3 = 2$ are equally spaced, with spacing $h = 1$.

To apply the backward-difference formula, we define $s = (x - x_3)/h = x - 2$ and compute

$$\begin{aligned}
\binom{-s}{1} &= -s \\
&= -(x-2), \\
\binom{-s}{2} &= \frac{-s(-s-1)}{2} \\
&= \frac{s(s+1)}{2} \\
&= \frac{(x-2)(x-1)}{2}, \\
\binom{-s}{3} &= \frac{-s(-s-1)(-s-2)}{6} \\
&= -\frac{s(s+1)(s+2)}{6} \\
&= -\frac{(x-2)(x-1)x}{6}, \\
f[x_3] &= f(x_3) \\
&= -6, \\
\nabla f(x_3) &= f(x_3) - f(x_2) \\
&= -6 - 5 \\
&= -11, \\
\nabla^2 f(x_3) &= \nabla(\nabla f(x_3)) \\
&= \Delta[f(x_3) - f(x_2)] \\
&= [f(x_3) - f(x_2)] - [f(x-2) - f(x_1)] \\
&= f(x_3) - 2f(x_2) + f(x_1) \\
&= -6 - 2(5) + -4, \\
&= -20
\end{aligned}$$

$$\begin{aligned}
\nabla^3 f(x_3) &= \nabla(\nabla^2 f(x_3)) \\
&= \nabla[f(x_3) - 2f(x_2) + f(x_1)] \\
&= [f(x_3) - f(x_2)] - 2[f(x_2) - f(x_1)] + [f(x_1) - f(x_0)] \\
&= f(x_3) - 3f(x_2) + 3f(x_1) - f(x_0) \\
&= -6 - 3(5) + 3(-4) - 3 \\
&= -36.
\end{aligned}$$

It follows that

$$\begin{aligned}
p_3(x) &= f[x_3] + \sum_{k=1}^{3} (-1)^k \binom{-s}{k} \nabla^k f(x_3) \\
&= -6 - \binom{-s}{1} \nabla f(x_3) + \binom{-s}{1} \nabla^2 f(x_3) - \binom{-s}{2} \nabla^3 f(x_3) \\
&= -6 - [-(x-2)](-11) + \frac{(x-2)(x-1)}{2}(-20) - \frac{-(x-2)(x-1)x}{6}(-36) \\
&= -6 - 11(x-2) - 10(x-2)(x-1) - 6(x-2)(x-1)x.
\end{aligned}$$

Note that the backward-difference formula does not compute the same form of the interpolating polynomial $p_n(x)$ as the Newton divided-difference formula. Instead, it computes a different Newton form of $p_n(x)$ given by

$$\begin{aligned}
p_n(x) &= \sum_{j=0}^{n} f[x_j, x_{j+1}, \ldots, x_n] \prod_{i=j+1}^{n} (x - x_i) \\
&= f[x_n] + f[x_{n-1}, x_n](x - x_n) + f[x_{n-2}, x_{n-1}, x_n](x - x_{n-1})(x - x_n) + \\
&\quad \cdots + f[x_0, x_1, \ldots, x_n](x - x_1)(x - x_2) \cdots (x - x_{n-1})(x - x_n).
\end{aligned}$$

The divided-differences $f[x_j, \ldots, x_n]$, for $j = 0, 1, \ldots, n$, can be obtained by constructing a divided-difference table as in the first example. These divided differences appear in the bottom row of the table, whereas the divided differences used in the forward-difference formula appear in the top row. □

3.4 Osculatory Interpolation

Suppose that the interpolation points are perturbed so that two neighboring points x_i and x_{i+1}, $0 \leq i < n$, approach each other. What happens to the interpolating polynomial? In the limit, as $x_{i+1} \to x_i$, the interpolating polynomial $p_n(x)$ not only satisfies $p_n(x_i) = y_i$, but also the condition

$$p_n'(x_i) = \lim_{x_{i+1} \to x_i} \frac{y_{i+1} - y_i}{x_{i+1} - x_i}.$$

It follows that in order to ensure uniqueness, the data must specify the value of the derivative of the interpolating polynomial at x_i.

In general, the inclusion of an interpolation point x_i k times within the set $x_0, \ldots, x_n$ must be accompanied by specification of $p_n^{(j)}(x_i)$, $j = 0, \ldots, k-1$, in order to ensure a unique solution. These values are used in place of divided differences of identical interpolation points in Newton interpolation.

Interpolation with repeated interpolation points is called *osculatory interpolation*, since it can be viewed as the limit of distinct interpolation points approaching one another, and the term "osculatory" is based on the Latin word for "kiss".

3.4.1 Hermite Interpolation

In the case where each of the interpolation points $x_0, x_1, \ldots, x_n$ is repeated exactly once, the interpolating polynomial for a differentiable function $f(x)$ is called the *Hermite polynomial* of $f(x)$, and is denoted by $p_{2n+1}(x)$, since this polynomial must have degree $2n+1$ in order to satisfy the $2n+2$ constraints

$$p_{2n+1}(x_i) = f(x_i), \quad p'_{2n+1}(x_i) = f'(x_i), \quad i = 0, 1, \ldots, n.$$

The Hermite polynomial can be described using Lagrange polynomials and their derivatives, but this representation is not practical because of the difficulty of differentiating and evaluating these polynomials. Instead, one can construct the Hermite polynomial using a divided-difference table, as discussed previously, in which each entry corresponding to two identical interpolation points is filled with the value of $f'(x)$ at the common point. Then, the Hermite polynomial can be represented using the Newton divided-difference formula.

Example We will use Hermite interpolation to construct the third-degree polynomial $p_3(x)$ that fits the data

i	x_i	$f(x_i)$	$f'(x_i)$
0,1	0	0	1
2,3	1	0	1

In other words, we must have $p_3(0) = 0$, $p'_3(0) = 1$, $p_3(1) = 0$, and $p'_3(1) = 1$. To include the values of $f'(x)$ at the two distinct interpolation points, we repeat each point once, so that the number of interpolation points, including repetitions, is equal to the number of constraints described by the data.

First, we construct the *divided-difference table* from this data. The divided differences in the table are computed as follows:

$$
\begin{aligned}
f[x_0] &= f(x_0) \\
&= 0 \\
f[x_1] &= f(x_1) \\
&= 0 \\
f[x_2] &= f(x_2) \\
&= 0 \\
f[x_3] &= f(x_3) \\
&= 0 \\
f[x_0, x_1] &= \frac{f[x_1] - f[x_0]}{x_1 - x_0} \\
&= f'(x_0) \\
&= 1 \\
f[x_1, x_2] &= \frac{f[x_2] - f[x_1]}{x_2 - x_1} \\
&= \frac{0 - 0}{1 - 0} \\
&= 0 \\
f[x_2, x_3] &= \frac{f[x_3] - f[x_2]}{x_3 - x_2} \\
&= f'(x_2) \\
&= 1 \\
f[x_0, x_1, x_2] &= \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} \\
&= \frac{0 - 1}{1 - 0} \\
&= -1 \\
f[x_1, x_2, x_3] &= \frac{f[x_2, x_3] - f[x_1, x_2]}{x_3 - x_1} \\
&= \frac{1 - 0}{1 - 0} \\
&= 1 \\
f[x_0, x_1, x_2, x_3] &= \frac{f[x_1, x_2, x_3] - f[x_0, x_1, x_2]}{x_3 - x_0} \\
&= \frac{1 - (-1)}{1 - 0}
\end{aligned}
$$

$$= \quad 2$$

Note that the values of the derivative are used whenever a divided difference of the form $f[x_i, x_{i+1}]$ is to be computed, where $x_i = x_{i+1}$. This makes sense because

$$\lim_{x_{i+1} \to x_i} f[x_i, x_{i+1}] = \lim_{x_{i+1} \to x_i} \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} = f'(x_i).$$

The resulting divided-difference table is

$x_0 = 0$	$f[x_0] = 0$			
		$f[x_0, x_1] = 1$		
$x_1 = 0$	$f[x_1] = 0$		$f[x_0, x_1, x_2] = -1$	
		$f[x_1, x_2] = 0$		$f[x_0, x_1, x_2, x_3] = 2$
$x_2 = 1$	$f[x_2] = 0$		$f[x_1, x_2, x_3] = 1$	
		$f[x_2, x_3] = 1$		
$x_3 = 1$	$f[x_3] = 0$			

It follows that the interpolating polynomial $p_3(x)$ can be obtained using the *Newton divided-difference formula* as follows:

$$\begin{aligned} p_3(x) &= \sum_{j=0}^{3} f[x_0, \ldots, x_j] \prod_{i=0}^{j-1} (x - x_i) \\ &= f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \\ &\quad f[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2) \\ &= 0 + (x - 0) + (-1)(x - 0)(x - 0) + 2(x - 0)(x - 0)(x - 1) \\ &= x - x^2 + 2x^2(x - 1). \end{aligned}$$

We see that Hermite interpolation, using divided differences, produces an interpolating polynomial that is in the *Newton form*, with centers $x_0 = 0$, $x_1 = 0$, and $x_2 = 1$. □

The error in the Hermite polynomial can be obtained using the formula for the error in the Lagrange polynomial. Specifically, to obtain the error in the Hermite polynomial $p_{2n+1}(x)$ that interpolates $f(x)$ and $f'(x)$ at the $n+1$ points $x_0, x_1, \ldots, x_n$, we use the Lagrange error formula of degree $2n+1$ with $2n+2$ *distinct* points $z_0, z_1, \ldots, z_{2n+1}$, and then, for $i = 0, 1, \ldots, n$, we let z_{2i} and z_{2i+1} approach x_i. The result is

$$f(x) - p_{2n+1}(x) = \frac{f^{(2n+2)}(\xi(x))}{(2n+2)!} \prod_{j=0}^{n} (x - x_j)^2,$$

where $\xi(x)$ lies in an interval containing $x_0, x_1, \ldots, x_n$.

3.5 Piecewise Polynomial Interpolation

If the number of data points is large, then polynomial interpolation becomes problematic since high-degree interpolation yields oscillatory polynomials, when the data may fit a smooth function.

Example Suppose that we wish to approximate the function $f(x) = 1/(1+x^2)$ on the interval $[-5, 5]$ with a tenth-degree interpolating polynomial that agrees with $f(x)$ at 11 equally-spaced points $x_0, x_1, \ldots, x_{10}$ in $[-5, 5]$, where $x_j = -5 + j$, for $j = 0, 1, \ldots, 10$. Figure 3.1 shows that the resulting polynomial is not a good approximation of $f(x)$ on this interval, even though it agrees with $f(x)$ at the interpolation points. The following MATLAB session shows how the plot in the figure can be created.

```
>> % create vector of 11 equally spaced points in [-5,5]
>> x=linspace(-5,5,11);
>> % compute corresponding y-values
>> y=1./(1+x.^2);
>> % compute 10th-degree interpolating polynomial
>> p=polyfit(x,y,10);
>> % for plotting, create vector of 100 equally spaced points
>> xx=linspace(-5,5);
>> % compute corresponding y-values to plot function
>> yy=1./(1+xx.^2);
>> % plot function
>> plot(xx,yy)
>> % tell MATLAB that next plot should be superimposed on
>> % current one
>> hold on
>> % plot polynomial, using polyval to compute values
>> % and a red dashed curve
>> plot(xx,polyval(p,xx),'r--')
>> % indicate interpolation points on plot using circles
>> plot(x,y,'o')
>> % label axes
>> xlabel('x')
>> ylabel('y')
>> % set caption
>> title('Runge''s example')
```

In general, it is not wise to use a high-degree interpolating polynomial and equally-spaced interpolation points to approximate a function on an interval

$[a, b]$ unless this interval is sufficiently small. For example, one can interpolate this same function $f(x) = 1/(1 + x^2)$ with reasonable accuracy on the interval $[-1, 1]$ with a tenth-degree polynomial and equally-spaced interpolation points (try it).

In general, when one is able to choose the interpolation points arbitrarily, the *Chebyshev points*, to be discussed later, yield much greater accuracy. The example shown in Figure 3.1 is a well-known example of the difficulty of high-degree polynomial interpolation using equally-spaced points, and it is known as *Runge's example.* □

If the fitting function is only required to have a few continuous derivatives, then one can construct a *piecewise polynomial* to fit the data.

We now precisely define what we mean by a piecewise polynomial.

Definition *(Piecewise polynomial) Let $[a, b]$ be an interval that is divided into subintervals $[x_i, x_{i+1}]$, where $i = 0, \ldots, n-1$, $x_0 = a$ and $x_n = b$. A* **piecewise polynomial** *is a function $p(x)$ defined on $[a, b]$ by*

$$p(x) = p_i(x), \quad x_i \leq x \leq x_{i+1}, \quad i = 0, 1, \ldots, n-1,$$

where, for $i = 0, 1, \ldots, n-1$, each function $p_i(x)$ is a polynomial defined on $[x_i, x_{i+1}]$. The **degree** *of $p(x)$ is the maximum degree of each polynomial $p_i(x)$, for $i = 0, 1, \ldots, n-1$.*

It is essential to note that by this definition, a piecewise polynomial defined on $[a, b]$ is equal to some polynomial on each subinterval $[x_i, x_{i+1}]$ of $[a, b]$, for $i = 0, 1, \ldots, n-1$, but a different polynomial may be used for each subinterval.

Typically, piecewise polynomials are used to fit smooth functions, and therefore are required to have a certain number of continuous derivatives. This requirement imposes additional constraints on the piecewise polynomial, and therefore the degree of the polynomials used on each subinterval must be chosen sufficiently high to ensure that these constraints can be satisfied.

3.5.1 Cubic Spline Interpolation

A *spline* is a piecewise polynomial of degree k that has $k - 1$ continuous derivatives. The most commonly used spline is a *cubic spline*, which we now define.

Definition *(Cubic Spline) Let $f(x)$ be function defined on an interval $[a, b]$, and let $x_0, x_1, \ldots, x_n$ be $n + 1$ distinct points in $[a, b]$, where $a = x_0 < x_1 <$*

$\cdots < x_n = b$. *A* **cubic spline**, *or* **cubic spline interpolant**, *is a piecewise polynomial $s(x)$ that satisifes the following conditions:*

1. *On each interval $[x_{i-1}, x_i]$, for $i = 1, \ldots, n$, $s(x) = s_i(x)$, where $s_i(x)$ is a cubic polynomial.*
2. *$s(x_i) = f(x_i)$ for $i = 0, 1, \ldots, n$.*
3. *$s(x)$ is twice continuously differentiable on (a, b).*
4. *Either of the following boundary conditions are satisfied:*
   (a) *$s''(a) = s''(b) = 0$, which is called* **free** *or* **natural boundary conditions**, *and*
   (b) *$s'(a) = f'(a)$ and $s'(b) = f'(b)$, which is called* **clamped boundary conditions**.

If $s(x)$ satisfies free boundary conditions, we say that $s(x)$ is a **natural spline**. *The points $x_0, x_1, \ldots, x_n$ are called the* **nodes** *of $s(x)$.*

Clamped boundary conditions are often preferable because they use more information about $f(x)$, which yields a spline that better approximates $f(x)$ on $[a, b]$. However, if information about $f'(x)$ is not available, then free boundary conditions must be used instead.

3.5.2 Constructing Cubic Splines

Suppose that we wish to construct a cubic spline interpolant $s(x)$ that fits the given data (x_0, y_0), $(x_1, y_1), \ldots, (x_n, y_n)$, where $a = x_0 < x_1 < \cdots < x_n = b$, and $y_i = f(x_i)$, for some known function $f(x)$ defined on $[a, b]$. From the preceding discussion, this spline is a piecewise polynomial of the form

$$s(x) = s_i(x) = d_i(x-x_i)^3 + c_i(x-x_i)^2 + b_i(x-x_i) + a_i, \quad i = 0, 1, \ldots, n-1, \quad x_i \leq x \leq x_{i+1}.$$

That is, the value of $s(x)$ is obtained by evaluating a different cubic polynomial for each subinterval $[x_i, x_{i+1}]$, for $i = 0, 1, \ldots, n-1$.

We now use the definition of a cubic spline to construct a system of equations that must be satisfied by the coefficients a_i, b_i, c_i and d_i for $i = 0, 1, \ldots, n-1$. We can then compute these coefficients by solving the system. Because $s(x)$ must fit the given data, we have

$$a_i = y_i, \quad i = 0, 1, \ldots, n-1.$$

If we define $h_i = x_{i+1} - x_i$, for $i = 0, 1, \ldots, n-1$, and define $a_n = y_n$, then the requirement that $s(x)$ is continuous at the interior nodes implies that we must have $s_i(x_{i+1}) = s_{i+1}(x_{i+1})$ for $i = 0, 1, \ldots, n-2$. Furthermore, because $s(x)$ must fit the given data, we must also have $s(x_n) = s_{n-1}(x_n) = y_n$. These conditions lead to the constraints

$$d_i h_i^3 + c_i h_i^2 + b_i h_i + a_i = a_{i+1}, \quad i = 0, 1, \ldots, n-1.$$

To ensure that $s(x)$ has a continuous first derivative at the interior nodes, we require that $s_i'(x_{i+1}) = s_{i+1}'(x_{i+1})$ for $i = 0, 1, \ldots, n-2$, which imposes the constraints

$$3d_i h_i^2 + 2c_i h_i + b_i = b_{i+1}, \quad i = 0, 1, \ldots, n-2.$$

Similarly, to enforce continuity of the second derivative at the interior nodes, we require that $s_i''(x_{i+1}) = s_{i+1}''(x_{i+1})$ for $i = 0, 1, \ldots, n-2$, which leads to the constraints

$$3d_i h_i + c_i = c_{i+1}, \quad i = 0, 1, \ldots, n-2.$$

There are $4n$ coefficients to determine, since there are n cubic polynomials, with 4 coefficients each. However, we have only prescribed $4n-2$ constraints, so we must specify 2 more in order to determine a unique solution. If we use free boundary conditions, then these constraints are

$$\begin{aligned} c_0 &= 0, \\ 3d_{n-1}h_{n-1} + c_{n-1} &= 0. \end{aligned}$$

On the other hand, if we use clamped boundary conditions, then our additional constraints are

$$\begin{aligned} b_0 &= z_0, \\ 3d_{n-1}h_{n-1}^2 + 2c_{n-1}h_{n-1} + b_{n-1} &= z_n, \end{aligned}$$

where $z_i = f'(x_i)$ for $i = 0, 1, \ldots, n$. In the next lecture, we will learn how to use these equations to construct cubic spline interpolants.

3.5.3 Constructing Cubic Splines, cont'd

Having determined our constraints that must be satisfied by $s(x)$, we can set up a system of linear equations $A\mathbf{x} = \mathbf{b}$ based on these constraints, and then solve this system to determine the coefficients a_i, b_i, c_i, d_i for $i =$

$0, 1, \ldots, n-1$. In the case of free boundary conditions, A is an $(n+1)\times(n+1)$ matrix is defined by

$$A = \begin{bmatrix} 1 & 0 & 0 & \cdots & \cdots & 0 \\ h_0 & 2(h_0+h_1) & h_1 & \ddots & & \vdots \\ 0 & h_1 & 2(h_1+h_2) & h_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ 0 & \cdots & \cdots & 0 & 0 & 1 \end{bmatrix}$$

and the $(n+1)$-vectors $\mathbf{x}$ and $\mathbf{b}$ are

$$\mathbf{x} = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 0 \\ \frac{3}{h_1}(a_2 - a_1) - \frac{3}{h_0}(a_1 - a_0) \\ \vdots \\ \frac{3}{h_{n-1}}(a_n - a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1} - a_{n-2}) \\ 0 \end{bmatrix},$$

where $c_n = s''(x_n)$.

In the case of clamped boundary conditions, we have

$$A = \begin{bmatrix} 2h_0 & h_0 & 0 & \cdots & \cdots & 0 \\ h_0 & 2(h_0+h_1) & h_1 & \ddots & & \vdots \\ 0 & h_1 & 2(h_1+h_2) & h_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ 0 & \cdots & \cdots & 0 & h_{n-1} & 2h_{n-1} \end{bmatrix}$$

and

$$\mathbf{x} = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} \frac{3}{h_0}(a_1 - a_0) - 3z_0 \\ \frac{3}{h_1}(a_2 - a_1) - \frac{3}{h_0}(a_1 - a_0) \\ \vdots \\ \frac{3}{h_{n-1}}(a_n - a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1} - a_{n-2}) \\ 3z_n - \frac{3}{h_{n-1}}(a_n - a_{n-1}) \end{bmatrix}.$$

Once the coefficients $c_0, c_1, \ldots, c_n$ have been determined, the remaining coefficients can be computed as follows:

1. The coefficients $a_0, a_1, \ldots, a_n$ have already been defined by the relations $a_i = y_i$, for $i = 0, 1, \ldots, n$.

2. The coefficients $b_0, b_1, \ldots, b_{n-1}$ are given by

$$b_i = \frac{1}{h_i}(a_{i+1} - a_i) - \frac{h_i}{3}(2c_i + c_{i+1}), \quad i = 0, 1, \ldots, n-1,$$

3. The coefficients $d_0, d_1, \ldots, d_{n-1}$ can be obtained using the constraints

$$3d_i h_i + c_i = c_{i+1}, \quad i = 0, 1, \ldots, n-1.$$

Example We will construct a cubic spline interpolant for the following data on the interval $[0, 2]$.

j	x_j	y_j
0	0	3
1	1/2	−4
2	1	5
3	3/2	−6
4	2	7

The spline, $s(x)$, will consist of four pieces $\{s_j(x)\}_{j=0}^3$, each of which is a cubic polynomial of the form

$$s_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3, \quad j = 0, 1, 2, 3.$$

We will impose free, or natural, boundary conditions on this spline, so it will satisfy the conditions $s''(0) = s''(2) = 0$, in addition to the "essential" conditions imposed on a spline: it must fit the given data and have continuous first and second derivatives on the interval $[0, 2]$.

These conditions lead to the following system of equations that must be solved for the coefficients c_0, c_1, c_2, c_3, and c_4, where $c_j = s''(x_j)/2$ for $j = 0, 1, \ldots, 4$. We define $h = (2 - 0)/4 = 1/2$ to be the spacing between the interpolation points.

$$\begin{aligned}
c_0 &= 0 \\
\frac{h}{3}(c_0 + 4c_1 + c_2) &= \frac{y_2 - 2y_1 + y_0}{h} \\
\frac{h}{3}(c_1 + 4c_2 + c_3) &= \frac{y_3 - 2y_2 + y_1}{h} \\
\frac{h}{3}(c_2 + 4c_3 + c_4) &= \frac{y_4 - 2y_3 + y_2}{h} \\
c_4 &= 0.
\end{aligned}$$

Substituting $h = 1/2$ and the values of y_j, and also taking into account the boundary conditions, we obtain

$$\begin{aligned} \frac{1}{6}(4c_1 + c_2) &= 32 \\ \frac{1}{6}(c_1 + 4c_2 + c_3) &= -40 \\ \frac{1}{6}(c_2 + 4c_3) &= 48 \end{aligned}$$

This system has the solutions

$$c_1 = 516/7, \quad c_2 = -720/7, \quad c_3 = 684/7.$$

Using the relation $a_j = y_j$, for $j = 0, 1, 2, 3$, and the formula

$$b_j = \frac{a_{j+1} - a_j}{h} - \frac{h}{3}(2c_j + c_{j+1}), \quad j = 0, 1, 2, 3,$$

we obtain

$$b_0 = -184/7, \quad b_1 = 74/7, \quad b_2 = -4, \quad b_3 = -46/7.$$

Finally, using the formula

$$d_j = \frac{c_{j+1} - c_j}{3h}, \quad j = 0, 1, 2, 3,$$

we obtain

$$d_0 = 344/7, \quad d_1 = -824/7, \quad d_2 = 936/7, \quad d_3 = -456/7.$$

We conclude that the spline $s(x)$ that fits the given data, has two continuous derivatives on $[0, 2]$, and satisfies natural boundary conditions is

$$s(x) = \begin{cases} \frac{344}{7}x^3 - \frac{184}{7}x^2 + 3 & \text{if } x \in [0, 0.5] \\ -\frac{824}{7}(x - 1/2)^3 + \frac{516}{7}(x - 1/2)^2 + \frac{74}{7}(x - 1/2) - 4 & \text{if } x \in [0.5, 1] \\ \frac{936}{7}(x - 1)^3 - \frac{720}{7}(x - 1)^2 - 4(x - 1) + 5 & \text{if } x \in [1, 1.5] \\ -\frac{456}{7}(x - 3/2)^3 + \frac{684}{7}(x - 3/2)^2 - \frac{46}{7}(x - 3/2) - 6 & \text{if } x \in [1.5, 2] \end{cases}.$$

The graph of the spline is shown in Figure 3.2. □

3.5.4 Well-Posedness and Accuracy

For both boundary conditions, the system $A\mathbf{x} = \mathbf{b}$ has a unique solution, which leads to the following results.

Theorem *Let $x_0, x_1, \ldots, x_n$ be $n+1$ distinct points in the interval $[a, b]$, where $a = x_0 < x_1 < \cdots < x_n = b$, and let $f(x)$ be a function defined on $[a, b]$. Then f has a unique cubic spline interpolant $s(x)$ that is defined on the nodes $x_0, x_1, \ldots, x_n$ that satisfies the natural boundary conditions $s''(a) = s''(b) = 0$.*

Theorem *Let $x_0, x_1, \ldots, x_n$ be $n+1$ distinct points in the interval $[a, b]$, where $a = x_0 < x_1 < \cdots < x_n = b$, and let $f(x)$ be a function defined on $[a, b]$ that is differentiable at a and b. Then f has a unique cubic spline interpolant $s(x)$ that is defined on the nodes $x_0, x_1, \ldots, x_n$ that satisfies the clamped boundary conditions $s'(a) = f'(a)$ and $s'(b) = f'(b)$.*

The following result provides insight into the accuracy with which a cubic spline interpolant $s(x)$ approximates a function $f(x)$.

Theorem *Let f be four times continuously differentiable on $[a, b]$, and assume that $|f^{(4)}(x)| \leq M$ on $[a, b]$ for some constant M. Let $s(x)$ be the unique clamped cubic spline interpolant of $f(x)$ on the nodes $x_0, x_1, \ldots, x_n$, where $a = x_0 < x_1 < \cdots < x_n < b$. Then for $x \in [a, b]$,*

$$|f(x) - s(x)| \leq \frac{5M}{384} \max_{0 \leq i \leq n-1} h_i^4,$$

where $h_i = x_{i+1} - x_i$.

3.5.5 B-splines

An alternative method of computing splines to fit given data involves constructing a basis for the vector space of splines defined on the interval $[a, b]$, and then solving a system of linear equations for the coefficients of the desired spline in this basis. The basis functions are known as *B-splines*, where the letter B is due to the fact that these splines form a basis, and the fact that they tend to have bell-shaped graphs.

One advantage of using B-splines is that the system of linear equations that must be solved for the coefficients of a spline in the basis is banded, and therefore can be solved very efficiently. Furthermore, because each B-spline has compact support, it follows that a change in the data value y_i only causes the coefficients of a few B-splines to be changed, whereas in

cubic spline interpolation, such a change forces all of the coefficients of each polynomial $s_i(x)$ to be recomputed.

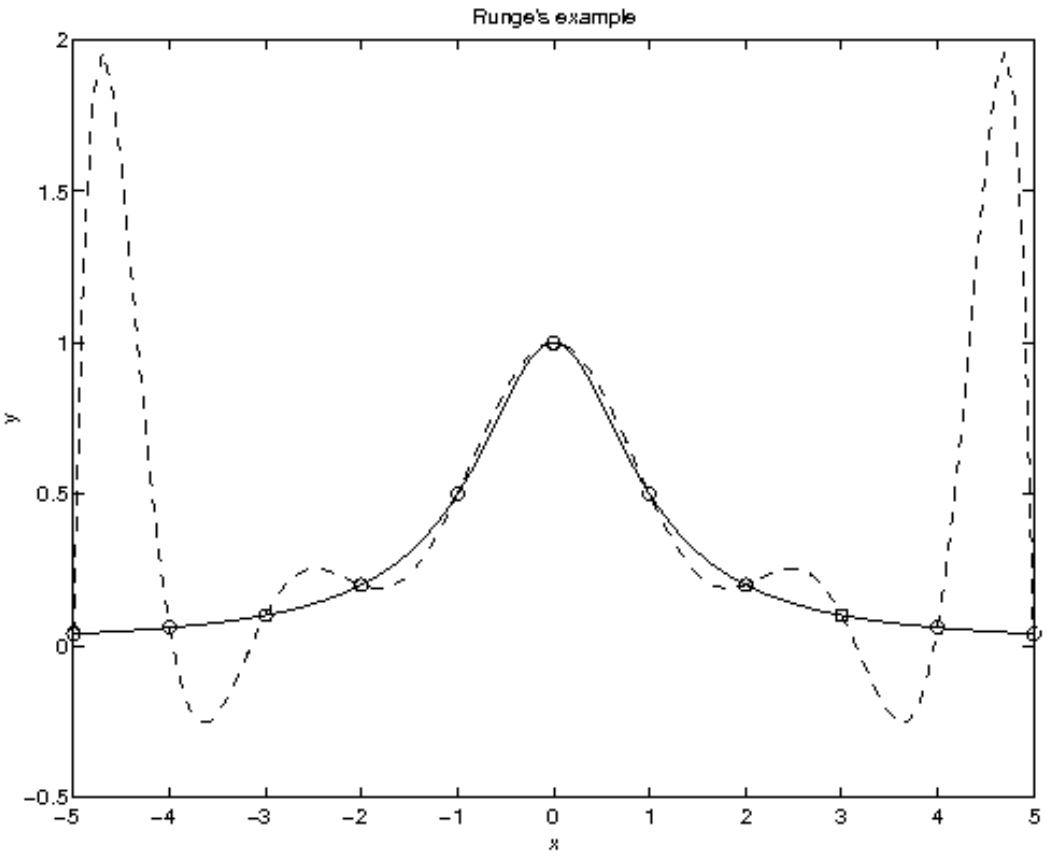

Figure 3.1: The function $f(x) = 1/(1 + x^2)$ (solid curve) cannot be interpolated accurately on $[-5, 5]$ using a tenth-degree polynomial (dashed curve) with equally-spaced interpolation points. This example that illustrates the difficulty that one can generally expect with high-degree polynomial interpolation with equally-spaced points is known as *Runge's example*.

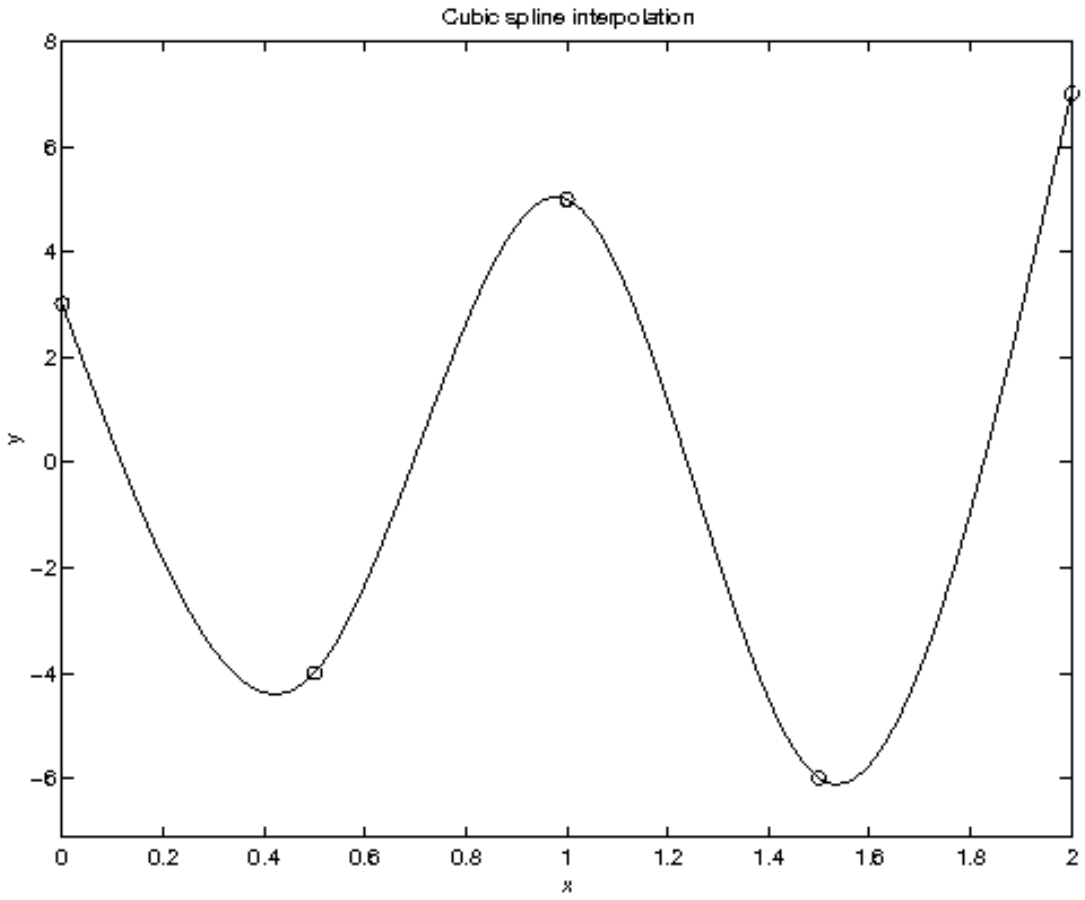

Figure 3.2: Cubic spline that passing through the points $(0, 3)$, $(1/2, -4)$, $(1, 5)$, $(2, -6)$, and $(3, 7)$.

Chapter 4

Numerical Differentiation and Integration

4.1 Numerical Differentiation

We now discuss the other fundamental problem from calculus that frequently arises in scientific applications, the problem of computing the derivative of a given function $f(x)$.

4.1.1 Finite Difference Approximations

Recall that the derivative of $f(x)$ at a point x_0, denoted $f'(x_0)$, is defined by

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}.$$

This definition suggests a method for approximating $f'(x_0)$. If we choose h to be a small positive constant, then

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0)}{h}.$$

This approximation is called the *forward difference formula*.

To estimate the accuracy of this approximation, we note that if $f''(x)$ exists on $[x_0, x_0 + h]$, then, by Taylor's Theorem, $f(x_0 + h) = f(x_0) + f'(x_0)h + f''(\xi)h^2/2$, where $\xi \in [x_0, x_0 + h]$. Solving for $f'(x_0)$, we obtain

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h} - \frac{f''(\xi)}{2}h,$$

so the error in the forward difference formula is $O(h)$. We say that this formula is *first-order accurate*.

The forward-difference formula is called a *finite difference approximation* to $f'(x_0)$, because it approximates $f'(x)$ using values of $f(x)$ at points that have a small, but finite, distance between them, as opposed to the definition of the derivative, that takes a limit and therefore computes the derivative using an "infinitely small" value of h. The forward-difference formula, however, is just one example of a finite difference approximation. If we replace h by $-h$ in the forward-difference formula, where h is still positive, we obtain the *backward-difference formula*

$$f'(x_0) \approx \frac{f(x_0) - f(x_0 - h)}{h}.$$

Like the forward-difference formula, the backward difference formula is first-order accurate.

If we average these two approximations, we obtain the *centered-difference formula*

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0 - h)}{2h}.$$

To determine the accuracy of this approximation, we assume that $f'''(x)$ exists on the interval $[x_0 - h, x_0 + h]$, and then apply Taylor's Theorem again to obtain

$$f(x_0 + h) = f(x_0) + f'(x_0)h + \frac{f''(x_0)}{2}h^2 + \frac{f'''(\xi_+)}{6}h^3,$$

$$f(x_0 - h) = f(x_0) - f'(x_0)h + \frac{f''(x_0)}{2}h^2 - \frac{f'''(\xi_-)}{6}h^3,$$

where $\xi_+ \in [x_0, x_0+h]$ and $\xi_- \in [x_0-h, x_0]$. Subtracting the second equation from the first and solving for $f'(x_0)$ yields

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h} - \frac{f'''(\xi_+) + f'''(\xi_-)}{12}h^2.$$

By the Intermediate Value Theorem, $f'''(x)$ must assume every value between $f'''(\xi_-)$ and $f'''(\xi_+)$ on the interval (ξ_-, ξ_+), including the average of these two values. Therefore, we can simplify this equation to

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h} - \frac{f'''(\xi)}{6}h^2,$$

where $\xi \in [x_0-h, x_0+h]$. We conclude that the centered-difference formula is *second-order accurate*. This is due to the cancellation of the terms involving $f''(x_0)$.

Example Consider the function

$$f(x) = \frac{\sin^2\left(\frac{\sqrt{x^2+x}}{\cos x - x}\right)}{\sin\left(\frac{\sqrt{x}-1}{\sqrt{x^2+1}}\right)}.$$

Our goal is to compute $f'(0.25)$. Differentiating, using the Quotient Rule and the Chain Rule, we obtain

$$\begin{aligned} f'(x) &= \frac{2\sin\left(\frac{\sqrt{x^2+x}}{\cos x - x}\right)\cos\left(\frac{\sqrt{x^2+x}}{\cos x - x}\right)\left[\frac{2x+1}{2\sqrt{x^2+1}(\cos x - x)} + \frac{\sqrt{x^2+1}(\sin x + 1)}{(\cos x - x)^2}\right]}{\sin\left(\frac{\sqrt{x}-1}{\sqrt{x^2+1}}\right)} - \\ &\quad \frac{\sin\left(\frac{\sqrt{x^2+x}}{\cos x - x}\right)\cos\left(\frac{\sqrt{x}-1}{\sqrt{x^2+1}}\right)\left[\frac{1}{2\sqrt{x}\sqrt{x^2+1}} - \frac{x(\sqrt{x}-1)}{(x^2+1)^{3/2}}\right]}{\sin^2\left(\frac{\sqrt{x}-1}{\sqrt{x^2+1}}\right)}. \end{aligned}$$

Evaluating this monstrous function at $x = 0.25$ yields $f'(0.25) = -9.066698770$.

An alternative approach is to use a *centered difference* approximation,

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

Using this formula with $x = 0.25$ and $h = 0.005$, we obtain the approximation

$$f'(0.25) \approx \frac{f(0.255) - f(0.245)}{0.01} = -9.067464295,$$

which has absolute error 7.7×10^{-4}. While this complicated function must be evaluated twice to obtain this approximation, that is still much less work than using differentiation rules to compute $f'(x)$, and then evaluating $f'(x)$, which is much more complicated than $f(x)$. □

While Taylor's Theorem can be used to derive formulas with higher-order accuracy simply by evaluating $f(x)$ at more points, this process can be tedious. An alternative approach is to compute the derivative of the interpolating polynomial that fits $f(x)$ at these points. Specifically, suppose we want to compute the derivative at a point x_0 using the data

$$(x_{-j}, y_{-j}), \ldots, (x_{-1}, y_{-1}), (x_0, y_0), (x_1, y_1), \ldots, (x_k, y_k),$$

where j and k are known nonnegative integers, $x_{-j} < x_{-j+1} < \cdots < x_{k-1} < x_k$, and $y_i = f(x_i)$ for $i = -j, \ldots, k$. Then, a finite difference formula for $f'(x_0)$ can be obtained by analytically computing the derivatives of the

Lagrange polynomials $\{\mathcal{L}_{n,i}(x)\}_{i=-j}^{k}$ for these points, where $n = j + k + 1$, and the values of these derivatives at x_0 are the proper weights for the function values $y_{-j}, \ldots, y_k$. If $f(x)$ is $n+1$ times continuously differentiable on $[x_{-j}, x_k]$, then we obtain an approximation of the form

$$f'(x_0) = \sum_{i=-j}^{k} y_i \mathcal{L}'_{n,i}(x_0) + \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=-j, i\neq 0}^{k} (x_0 - x_i),$$

where $\xi \in [x_{-j}, x_k]$.

Among the best-known finite difference formulas that can be derived using this approach is the second-order-accurate three-point formula

$$f'(x_0) = \frac{-3f(x_0) + 4f(x_0 + h) - f(x_0 + 2h)}{2h} + \frac{f'''(\xi)}{3}h^2, \quad \xi \in [x_0, x_0 + 2h],$$

which is useful when there is no information available about $f(x)$ for $x < x_0$. If there is no information available about $f(x)$ for $x > x_0$, then we can replace h by $-h$ in the above formula to obtain a second-order-accurate three-point formula that uses the values of $f(x)$ at x_0, $x_0 - h$ and $x_0 - 2h$.

Another formula is the five-point formula

$$f'(x_0) = \frac{f(x_0 - 2h) - 8f(x_0 - h) + 8f(x_0 + h) - f(x_0 + 2h)}{12h} + \frac{f^{(5)}(\xi)}{30}h^4, \quad \xi \in [x_0 - 2h, x_0 + 2h],$$

which is fourth-order accurate. The reason it is called a five-point formula, even though it uses the value of $f(x)$ at four points, is that it is derived from the Lagrange polynomials for the five points $x_0 - 2h, x_0 - h, x_0, x_0 + h$, and $x_0 + 2h$. However, $f(x_0)$ is not used in the formula because $\mathcal{L}'_{4,0}(x_0) = 0$, where $\mathcal{L}_{4,0}(x)$ is the Lagrange polynomial that is equal to one at x_0 and zero at the other four points.

If we do not have any information about $f(x)$ for $x < x_0$, then we can use the following five-point formula that actually uses the values of $f(x)$ at five points,

$$f'(x_0) = \frac{-25f(x_0) + 48f(x_0 + h) - 36f(x_0 + 2h) + 16f(x_0 + 3h) - 3f(x_0 + 4h)}{12h} + \frac{f^{(5)}(\xi)}{5}h^4,$$

where $\xi \in [x_0, x_0 + 4h]$. As before, we can replace h by $-h$ to obtain a similar formula that approximates $f'(x_0)$ using the values of $f(x)$ at $x_0, x_0 - h, x_0 - 2h, x_0 - 3h$, and $x_0 - 4h$.

The strategy of differentiating Lagrange polynomials to approximate derivatives can be used to approximate higher-order derivatives. For example, the second derivative can be approximated using a centered difference

formula,

$$f''(x_0) \approx \frac{f(x_0+h) - 2f(x_0) + f(x_0-h)}{h^2},$$

which is second-order accurate.

Example We will construct a formula for approximating $f'(x)$ at a given point x_0 by interpolating $f(x)$ at the points x_0, x_0+h, and x_0+2h using a second-degree polynomial $p_2(x)$, and then approximating $f'(x_0)$ by $p_2'(x_0)$. Since $p_2(x)$ should be a good approximation of $f(x)$ near x_0, especially when h is small, its derivative should be a good approximation to $f'(x)$ near this point.

Using Lagrange interpolation, we obtain

$$p_2(x) = f(x_0)\mathcal{L}_{2,0}(x) + f(x_0+h)\mathcal{L}_{2,1}(x) + f(x_0+2h)\mathcal{L}_{2,2}(x),$$

where $\{\mathcal{L}_{2,j}(x)\}_{j=0}^2$ are the Lagrange polynomials for the points x_0, $x_1 = x_0+h$ and $x_2 = x_0+2h$. Recall that these polynomials satisfy

$$\mathcal{L}_{2,j}(x_k) = \delta_{jk} = \begin{cases} 1 & \text{if } j = k \\ 0 & \text{otherwise} \end{cases}.$$

Using the formula for the Lagrange polynomials,

$$\mathcal{L}_{2,j}(x) = \prod_{i=0, i\neq j}^{2} \frac{(x-x_i)}{(x_j-x_i)},$$

we obtain

$$\begin{aligned}
\mathcal{L}_{2,0}(x) &= \frac{(x-(x_0+h))(x-(x_0+2h))}{(x_0-(x_0+h))(x_0-(x_0+2h))} \\
&= \frac{x^2-(2x_0+3h)x+(x_0+h)(x_0+2h)}{2h^2}, \\
\mathcal{L}_{2,1}(x) &= \frac{(x-x_0)(x-(x_0+2h))}{(x_0+h-x_0)(x_0+h-(x_0+2h))} \\
&= \frac{x^2-(2x_0+2h)x+x_0(x_0+2h)}{-h^2}, \\
\mathcal{L}_{2,2}(x) &= \frac{(x-x_0)(x-(x_0+h))}{(x_0+2h-x_0)(x_0+2h-(x_0+h))} \\
&= \frac{x^2-(2x_0+h)x+x_0(x_0+h)}{2h^2}.
\end{aligned}$$

It follows that

$$\begin{aligned}
\mathcal{L}'_{2,0}(x) &= \frac{2x - (2x_0 + 3h)}{2h^2} \\
\mathcal{L}'_{2,1}(x) &= -\frac{2x - (2x_0 + 2h)}{h^2} \\
\mathcal{L}'_{2,2}(x) &= \frac{2x - (2x_0 + h)}{2h^2}
\end{aligned}$$

We conclude that $f'(x_0) \approx p'_2(x_0)$, where

$$\begin{aligned}
p'_2(x_0) &= f(x_0)\mathcal{L}'_{2,0}(x_0) + f(x_0 + h)\mathcal{L}'_{2,0}(x_0) + f(x_0 + 2h)\mathcal{L}'_{2,0}(x_0) \\
&\approx f(x_0)\frac{-3}{2h} + f(x_0 + h)\frac{2}{h} + f(x_0 + 2h)\frac{-1}{2h} \\
&\approx \frac{3f(x_0) + 4f(x_0 + h) - f(x_0 + 2h)}{2h}.
\end{aligned}$$

Using Taylor's Theorem to write $f(x_0+h)$ and $f(x_0+2h)$ in terms of Taylor polynomials centered at x_0, it can be shown that the error in this approximation is $O(h^2)$, and that this formula is exact when $f(x)$ is a polynomial of degree 2 or less. □

In a practical implementation of finite difference formulas, it is essential to note that roundoff error in evaluating $f(x)$ is bounded independently of the spacing h between points at which $f(x)$ is evaluated. It follows that the roundoff error in the approximation of $f'(x)$ actually *increases* as h decreases, because the errors incurred by evaluating $f(x)$ are divided by h. Therefore, one must choose h sufficiently small so that the finite difference formula can produce an accurate approximation, and sufficiently large so that this approximation is not too contaminated by roundoff error.

4.1.2 Automatic Differentiation

If the function $f(x)$ can be represented as a sequence of computations, each consisting of the evaluation of some differentiable function with differentiable arguments, then a similar sequence of computations representing $f'(x)$ can be obtained using *automatic differentiation*.

Suppose that the evaluation of $f(x)$ consists of the evaluation of simpler differentiable functions $t_1, \ldots, t_n$, where t_1 only depends on x, and t_j, for $j > 1$, depends on x and $t_1, \ldots, t_{j-1}$, with $t_n(x, t_1, \ldots, t_{n-1})$ representing the desired value $f(x)$. Automatic differentiation proceeds by computing the derivatives of each function t_j, $j = 1, \ldots, n$, to obtain a new sequence

of functions t'_j, $j = 1, \ldots, n$, where t'_j depends on $t_1, \ldots, t_j$ and $t'_1, \ldots, t'_{j-1}$, and $f'(x) = t'_n(x, t_1, \ldots, t_n, t'_1, \ldots, t'_n)$.

This process, while conceptually simple, can be very difficult to implement since much optimization of the generated derivative is necessary. Fortunately, techniques used in compiler design are very useful in building a practical automatic differentiation procedure. An additional benefit of automatic differentiation is that its analysis of $f(x)$ can be used to estimate its sensitivity to perturbations.

4.2 Richardson Extrapolation

We have seen that the accuracy of methods for computing derivatives of a function $f(x)$ depends on the spacing between points at which f is evaluated, and that the approximation tends to the exact value as this spacing tends to 0.

Suppose that a uniform spacing h is used. We denote by $F(h)$ the approximation computed using the spacing h, from which it follows that the exact value is given by $F(0)$. Let p be the order of accuracy in our approximation; that is,

$$F(h) = a_0 + a_1 h^p + O(h^r), \quad r > p,$$

where a_0 is the exact value $F(0)$. Then, if we choose a value for h and compute $F(h)$ and $F(h/q)$ for some positive integer q, then we can neglect the $O(h^r)$ terms and solve a system of two equations for the unknowns a_0 and a_1, thus obtaining an approximation that is rth order accurate. If we can describe the error in this approximation in the same way that we can describe the error in our original approximation $F(h)$, we can repeat this process to obtain an approximation that is even more accurate.

This process of extrapolating from $F(h)$ and $F(h/q)$ to approximate $F(0)$ with a higher order of accuracy is called *Richardson extrapolation*. In a sense, Richardson extrapolation is similar in spirit to Aitken's Δ^2 method, as both methods use assumptions about the convergence of a sequence of approximations to "solve" for the exact solution, resulting in a more accurate method of computing approximations.

Example Consider the function

$$f(x) = \frac{\sin^2\left(\frac{\sqrt{x^2+x}}{\cos x - x}\right)}{\sin\left(\frac{\sqrt{x}-1}{\sqrt{x^2+1}}\right)}.$$

Our goal is to compute $f'(0.25)$ as accurately as possible. Using a centered difference approximation,

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

with $x = 0.25$ and $h = 0.01$, we obtain the approximation

$$f'(0.25) \approx \frac{f(0.26) - f(0.24)}{0.02} = -9.06975297890147,$$

which has absolute error 3.0×10^{-3}, and if we use $h = 0.005$, we obtain the approximation

$$f'(0.25) \approx \frac{f(0.255) - f(0.245)}{0.01} = -9.06746429492149,$$

which has absolute error 7.7×10^{-4}. As expected, the error decreases by a factor of approximately 4 when we halve the step size h, because the error in the centered difference formula is of $O(h^2)$.

We can obtain a more accurate approximation by applying *Richardson Extrapolation* to these approximations. We define the function $N_1(h)$ to be the centered difference approximation to $f'(0.25)$ obtained using the step size h. Then, with $h = 0.01$, we have

$$N_1(h) = -9.06975297890147, \quad N_1(h/2) = -9.06746429492149,$$

and the exact value is given by $N_1(0) = -9.06669877124279$. Because the error in the centered difference approximation satisfies

$$N_1(h) = N_1(0) + K_1 h^2 + K_2 h^4 + K_3 h^6 + O(h^8),$$

where the constants K_1, K_2 and K_3 depend on the derivatives of $f(x)$ at $x = 0.25$, it follows that the new approximation

$$N_2(h) = N_1(h/2) + \frac{N_1(h/2) - N_1(h)}{2^2 - 1} = -9.06670140026149,$$

has fourth-order accuracy. Specifically, if we denote the exact value by $N_2(0)$, we have

$$N_2(h) = N_2(0) + \tilde{K}_2 h^4 + \tilde{K}_3 h^6 + O(h^8),$$

where the constants $\tilde{K}_2$ and $\tilde{K}_3$ are independent of h.

Now, suppose that we compute

$$N_1(h/4) = \frac{f(x+h/4) - f(x-h/4)}{2(h/4)} = \frac{f(0.2525) - f(0.2475)}{0.005} = -9.06689027527046,$$

which has an absolute error of 1.9×10^{-4}, we can use extrapolation again to obtain a second fourth-order accurate approximation,

$$N_2(h/2) = N_1(h/4) + \frac{N_1(h/4) - N_1(h/2)}{3} = -9.06669893538678,$$

which has absolute error of 1.7×10^{-7}. It follows from the form of the error in $N_2(h)$ that we can use extrapolation on $N_2(h)$ and $N_2(h/2)$ to obtain a *sixth*-order accurate approximation,

$$N_3(h) = N_2(h/2) + \frac{N_2(h/2) - N_2(h)}{2^4 - 1} = -9.06669877106180,$$

which has an absolute error of 1.8×10^{-10}. □

4.3 Integration

Numerous applications call for the computation of the integral of some function $f : \mathbb{R} \to \mathbb{R}$ over an interval $[a, b]$,

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

In some cases, $I(f)$ can be computed by applying the Fundamental Theorem of Calculus and computing

$$I(f) = F(b) - F(a),$$

where $F(x)$ is an *antiderivative* of f. Unfortunately, this is not practical if an antiderivative of f is not available. In such cases, numerical techniques must be employed instead.

4.4 Well-Posedness

The integral $I(f)$ is defined using sequences of *Riemann sums*

$$R_n = \sum_{i=1}^{n} f(\xi_i)(x_{i+1} - x_i), \quad x_i \leq \xi_i \leq x_{i+1},$$

where $a = x_1 < x_2 < \cdots < x_n = b$. If any such sequence $\{R_n\}_{n=1}^{\infty}$ converges to the same finite value R as $n \to \infty$, then f is said to be *Riemann integrable* on $[a, b]$, and $I(f) = R$. A function is Riemann integrable if and only if it is bounded and continuous except on a set of measure zero. For such functions, the problem of computing $I(f)$ has a unique solution, by the definition of $I(f)$.

To determine the sensitivity of $I(f)$, we define the ∞-norm of a function $f(x)$ by

$$\|f\|_\infty = \max_{x \in [a,b]} |f(x)|$$

and let $\hat{f}$ be a perturbation of f that is also Riemann integrable. Then the condition number of the problem of computing $I(f)$ can be approximated by

$$\begin{aligned}
\frac{|I(\hat{f}) - I(f)|/|I(f)|}{\|\hat{f} - f\|_\infty / \|f\|_\infty} &= \frac{|I(\hat{f} - f)|/|I(f)|}{\|\hat{f} - f\|_\infty / \|f\|_\infty} \\
&\leq \frac{I(|\hat{f} - f|)/|I(f)|}{\|\hat{f} - f\|_\infty / \|f\|_\infty} \\
&\leq \frac{(b-a)\|\hat{f} - f\|_\infty / |I(f)|}{\|\hat{f} - f\|_\infty / \|f\|_\infty} \\
&\leq (b-a)\frac{\|f\|_\infty}{|I(f)|},
\end{aligned}$$

from which it follows that the problem is fairly well-conditioned in most cases, since, if $I(f)$ is small, we should then use the absolute condition number, which is bounded by $(b - a)$. Similarly, perturbations of the endpoints a and b do not lead to large perturbations in $I(f)$, in most cases.

4.5 Numerical Quadrature

Clearly, if f is a Riemann integrable function and $\{R_n\}_{n=1}^{\infty}$ is any sequence of Riemann sums that converges to $I(f)$, then any particular Riemann sum R_n can be viewed as an approximation of $I(f)$. However, such an approximation is usually not practical since a large value of n may be necessary to achieve sufficient accuracy.

Instead, we use a *quadrature rule* to approximate $I(f)$. A quadrature rule is a sum of the form

$$Q_n(f) = \sum_{i=1}^{n} f(x_i) w_i,$$

where the points x_i, $i = 1, \ldots, n$, are called the *nodes* of the quadrature rule, and the numbers w_i, $i = 1, \ldots, n$, are the *weights*. We say that a quadrature rule is *open* if the nodes do not include the endpoints a and b, and *closed* if they do.

The objective in designing quadrature rules is to achieve sufficient accuracy in approximating $I(f)$, for any Riemann integrable function f, while using as few nodes as possible in order to maximize efficiency. In order to determine suitable nodes and weights, we consider the following questions:

- For what functions f is $I(f)$ easy to compute?
- Given a general Riemann integrable function f, can $I(f)$ be approximated by the integral of a function g for which $I(g)$ is easy to compute?

One class of functions for which integrals are easily evaluated is the class of polynomial functions. If we choose n nodes $x_1, \ldots, x_n$, then any polynomial $p_{n-1}(x)$ of degree $n-1$ can be written in the form

$$p_{n-1}(x) = \sum_{i=1}^{n} p_{n-1}(x_i)\mathcal{L}_{n-1,i}(x),$$

where $\mathcal{L}_{n-1,i}(x)$ is the ith Lagrange polynomial for the points $x_1, \ldots, x_n$. It follows that

$$\begin{aligned} I(p_{n-1}) &= \int_a^b p_{n-1}(x)\,dx \\ &= \int_a^b \sum_{i=1}^{n} p_{n-1}(x_i)\mathcal{L}_{n-1,i}(x)\,dx \\ &= \sum_{i=1}^{n} p_{n-1}(x_i)\left(\int_a^b \mathcal{L}_{n-1,i}(x)\,dx\right) \\ &= \sum_{i=1}^{n} p_{n-1}(x_i)w_i \\ &= Q_n(p_{n-1}) \end{aligned}$$

where

$$w_i = \int_a^b \mathcal{L}_{n-1,i}(x)\,dx, \quad i = 1, \ldots, n,$$

are the weights of a quadrature rule with nodes $x_1, \ldots, x_n$.

Therefore, *any* n-point quadrature rule computes $I(f)$ *exactly* when f is a polynomial of degree less than n. For a more general function f, we

can use this quadrature rule to approximate $I(f)$ by $I(p_{n-1})$, where p_{n-1} is the polynomial that interpolates f at the points $x_1, \ldots, x_n$. Quadrature rules that use the weights defined above for given nodes $x_1, \ldots, x_n$ are called *interpolatory* quadrature rules. We say that an interpolatory quadrature rule has *degree of accuracy* n if it integrates polynomials of degree n exactly, but is not exact for polynomials of degree $n+1$.

If the weights w_i, $i = 1, \ldots, n$, are nonnegative, then the quadrature rule is stable, as its absolute condition number can be bounded by $(b-a)$, which is the same absolute condition number as the underlying integration problem. However, if any of the weights are negative, then the condition number can be arbitrarily large.

In the remainder of this section, we discuss specific interpolatory quadrature rules.

4.5.1 Newton-Cotes Quadrature

The family of *Newton-Cotes* quadrature rules consists of interpolatory quadrature rules in which the nodes are equally spaced points within the interval $[a, b]$. The most commonly used Newton-Cotes rules are

- The *Midpoint Rule*, which is an open rule with one node, is defined by

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

  It is of degree one, and it is based on the principle that the area under $f(x)$ can be approximated by the area of a rectangle with width $b-a$ and height $f(m)$, where $m = (a+b)/2$ is the midpoint of the interval $[a, b]$.

- The *Trapezoidal Rule*, which is a closed rule with two nodes, is defined by

$$\int_a^b f(x)\,dx \approx \frac{b-a}{2}[f(a) + f(b)].$$

  It is of degree one, and it is based on the principle that the area under $f(x)$ from $x = a$ to $x = b$ can be approximated by the area of a trapezoid with heights $f(a)$ and $f(b)$ and width $b-a$.

- *Simpson's Rule*, which is a closed rule with three nodes, is defined by

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

It is of degree three, and it is derived by computing the integral of the quadratic polynomial that interpolates $f(x)$ at the points a, $(a+b)/2$, and b.

Example Let $f(x) = x^3$, $a = 0$ and $b = 1$. We have

$$\int_a^b f(x)\,dx = \int_0^1 x^3\,dx = \left.\frac{x^4}{4}\right|_0^1 = \frac{1}{4}.$$

Approximating this integral with the Midpoint Rule yields

$$\int_0^1 x^3\,dx \approx (1-0)\left(\frac{0+1}{2}\right)^3 = \frac{1}{8}.$$

Using the Trapezoidal Rule, we obtain

$$\int_0^1 x^3\,dx \approx \frac{1-0}{2}[0^3 + 1^3] = \frac{1}{2}.$$

Finally, Simpson's Rule yields

$$\int_0^1 x^3\,dx \approx \frac{1-0}{6}\left[0^3 + 4\left(\frac{0+1}{2}\right)^3 + 1^3\right] = \frac{1}{6}\left[0 + 4\frac{1}{8} + 1\right] = \frac{1}{4}.$$

That is, the approximation of the integral by Simpson's Rule is actually exact, which is expected because Simpson's Rule is of degree three. On the other hand, if we approximate the integral of $f(x) = x^4$ from 0 to 1, Simpson's Rule yields 5/24, while the exact value is 1/5. Still, this is a better approximation than those obtained using the Midpoint Rule (1/16) or the Trapezoidal Rule (1/2). □

In general, the degree of accuracy of Newton-Cotes rules can easily be determined by expanding the integrand $f(x)$ in a Taylor series. This technique can be used to show that n-point Newton-Cotes rules with an odd number of nodes have degree n, which is surprising since, in general, interpolatory n-point quadrature rules have degree $n-1$. This extra degree of accuracy is due to the cancellation of the high-order error terms in the Taylor expansion used to determine the error. Such cancellation does not occur with Newton-Cotes rules that have an even number of nodes.

Unfortunately, Newton-Cotes rules are not practical when the number of nodes is large, due to the inaccuracy of high-degree polynomial interpolation using equally spaced points. Furthermore, for $n \geq 11$, n-point Newton-Cotes rules have at least one negative weight, and therefore such rules can be ill-conditioned.

4.5.2 Clenshaw-Curtis Quadrature

An alternative to Newton-Cotes rules are *Clenshaw-Curtis rules*, whose nodes are determined by Chebyshev polynomials. The roots of the nth Chebyshev polynomial yield the n-point *classical* Clenshaw-Curtis rule, while the interior extrema of the nth Chebyshev polynomial yields the $(n-1)$-point *practical* Clenshaw-Curtis rule. The latter rule is called practical because it has the property that the nodes of the $(2n-1)$-point rule include the nodes of the n-point rule. Such a sequence of quadrature rules that reuses nodes is said to be *progressive*.

The Clenshaw-Curtis rules have several advantages over Newton-Cotes rules:

- Their weights are always positive, making them much more stable than Newton-Cotes rules.
- Unlike Newton-Cotes rules, approximate integrals computed using n-point Clenshaw-Curtis rules converge to the exact integral as $n \to \infty$, from which it follows that such rules are much more accurate than n-point Newton-Cotes rules for sufficiently large n.
- The weights can be computed very efficiently by taking advantage of the three-term recurrence relation satisfied by Chebsyshev polynomials.

The main drawback of Clenshaw-Curtis rules is that n-point rules always have degree $n-1$.

The error in any interpolatory quadrature rule defined on an interval $[a,b]$, such as a Newton-Cotes rule or a Clenshaw-Curtis rule can be obtained by computing the integral from a to b of the error in the polynomial interpolant on which the rule is based.

For the Trapezoidal Rule, which is obtained by integrating a linear polynomial that interpolates the integrand $f(x)$ at $x=a$ and $x=b$, this approach to error analysis yields

$$\int_a^b f(x)\,dx - \frac{b-a}{2}[f(a)+f(b)] = \int_a^b \frac{f''(\xi(x))}{2}(x-a)(x-b)\,dx,$$

where $\xi(x)$ lies in $[a,b]$ for $a \le x \le b$. The function $(x-a)(x-b)$ does not change sign on $[a,b]$, which allows us to apply the Weighted Mean Value Theorem for Integrals and obtain a more useful expression for the error,

$$\int_a^b f(x)\,dx - \frac{b-a}{2}[f(a)+f(b)] = \frac{f''(\eta)}{2}\int_a^b (x-a)(x-b)\,dx = -\frac{f''(\eta)}{12}(b-a)^3,$$

where $a \leq \eta \leq b$. Because the error depends on the second derivative, it follows that the Trapezoidal Rule is exact for any linear function.

A similar approach can be used to obtain expressions for the error in the Midpoint Rule and Simpson's Rule, although the process is somewhat more complicated due to the fact that the functions $(x - m)$, for the Midpoint Rule, and $(x - a)(x - m)(x - b)$, for Simpson's Rule, where in both cases $m = (a + b)/2$, change sign on $[a, b]$, thus making the Weighted Mean Value Theorem for Integrals more difficult to apply. The result of the analysis is that for the Midpoint RUle,

$$\int_a^b f(x)\,dx - (b-a)f\left(\frac{a+b}{2}\right) = \frac{f''(\eta)}{24}(b-a)^3,$$

and for Simpson's Rule,

$$\int_a^b f(x)\,dx - \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right] = -\frac{f^{(4)}(\eta)}{90}\left(\frac{b-a}{2}\right)^5,$$

where, in both cases, η is some point in $[a, b]$.

It follows that the Midpoint Rule is exact for any linear function, just like the Trapezoidal Rule, even though it uses one less interpolation point, because of the cancellation that results from choosing the midpoint of $[a, b]$ as the interpolation point. Similar cancellation causes Simpson's Rule to be exact for polynomials of degree three or less, even though it is obtained by integrating a quadratic interpolant over $[a, b]$.

4.6 Composite Quadrature

When using a quadrature rule to approximate $I(f)$ on some interval $[a, b]$, the error is proportional to h^r, where $h = b - a$ and r is some positive integer. Therefore, if the interval $[a, b]$ is large, it is advisable to divide $[a, b]$ into smaller intervals, use a quadrature rule to compute the integral of f on each subinterval, and add the results to approximate $I(f)$. Such a scheme is called a *composite quadrature rule*.

It can be shown that the approximate integral obtained using a composite rule that divides $[a, b]$ into n subintervals will converge to $I(f)$ as $n \to \infty$, provided that the maximum width of the n subintervals approaches zero, and the quadrature rule used on each subinterval has a degree of at least zero. It should be noted that using closed quadrature rules on each subinterval improves efficiency, because the nodes on the endpoints of each subinterval, except for a and b, are shared by two quadrature rules. As a

result, fewer function evaluations are necessary, compared to a composite rule that uses open rules with the same number of nodes.

We will now state some of the most well-known composite quadrature rules. In the following discussion, we assume that the interval $[a, b]$ is divided into n subintervals of equal width $h = (b-a)/n$, and that these subintervals have endpoints $[x_{i-1}, x_i]$, where $x_i = a+ih$, for $i = 0, 1, 2, \ldots, n$. Given such a partition of $[a, b]$, we can compute $I(f)$ using the *Composite Midpoint Rule*

$$\int_a^b f(x)\,dx \approx 2h[f(x_1) + f(x_3) + \cdots + f(x_{n-1})], \quad n \text{ is even},$$

the *Composite Trapezoidal Rule*

$$\int_a^b f(x)\,dx \approx \frac{h}{2}[f(x_0) + 2f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-1}) + f(x_n)],$$

or the *Composite Simpson's Rule*

$$\int_a^b f(x)\,dx \approx \frac{h}{3}[f(x_0)+4f(x_1)+2f(x_2)+4f(x_3)+\cdots+2f(x_{n-2})+4f(x_{n-1})+f(x_n)],$$

for which n is required to be even, as in the Composite Midpoint Rule.

To obtain the error in each of these composite rules, we can sum the errors in the corresponding basic rules over the n subintervals. For the Composite Trapezoidal Rule, this yields

$$\begin{aligned} E_{trap} &= \int_a^b f(x)\,dx - \frac{h}{2}\left[f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n)\right] \\ &= -\sum_{i=1}^{n} \frac{f''(\eta_i)}{12}(x_i - x_{i-1})^3 \\ &= -\frac{h^3}{12}\sum_{i=1}^{n} f''(\eta_i) \\ &= -\frac{h^3}{12} n f''(\eta) \\ &= -\frac{f''(\eta)}{12}(b-a)h^2, \end{aligned}$$

where, for $i = 1, \ldots, n$, η_i belongs to $[x_{i-1}, x_i]$, and $a \leq \eta \leq b$. The replacement of $\sum_{i=1}^{h} f''(\eta_i)$ by $nf''(\eta)$ is justified by the Intermediate Value Theorem, provided that $f''(x)$ is continuous on $[a, b]$. We see that the Composite Trapezoidal Rule is *second-order accurate*. Furthermore, its *degree of*

accuracy, which is the highest degree of polynomial that is guaranteed to be integrated exactly, is the same as for the basic Trapezoidal Rule, which is one.

Similarly, for the Composite Midpoint Rule, we have

$$E_{mid} = \int_a^b f(x)\,dx - 2h\sum_{i=1}^{n/2} f(x_{2i-1}) = \sum_{i=1}^{n/2} \frac{f''(\eta_i)}{24}(2h)^3 = \frac{f''(\eta)}{6}(b-a)h^2.$$

Although it appears that the Composite Midpoint Rule is less accurate than the Composite Trapezoidal Rule, it should be noted that it uses about half as many function evaluations. In other words, the Basic Midpoint Rule is applied $n/2$ times, each on a subinterval of width $2h$. Rewriting the Composite Midpoint Rule in such a way that it uses n function evaluations, each on a subinterval of width h, we obtain

$$\int_a^b f(x)\,dx = h\sum_{i=1}^{n} f\left(x_{i-1} + \frac{h}{2}\right) + \frac{f''(\eta)}{24}(b-a)h^2,$$

which reveals that the Composite Midpoint Rule is generally more accurate.

Finally, for the Composite Simpson's Rule, we have

$$E_{simp} = -\sum_{i=1}^{n/2} \frac{f^{(4)}(\eta_i)}{90}h^5 = -\frac{f^{(4)}(\eta)}{180}(b-a)h^4,$$

because the Basic Simpson Rule is applied $n/2$ times, each on a subinterval of width $2h$. We conclude that the Simpson's Rule is *fourth-order accurate*.

Example We wish to approximate

$$\int_0^1 e^x\,dx$$

using composite quadrature, to 3 decimal places. That is, the error must be less than 10^{-3}. This requires choosing the number of subintervals, n, sufficiently large so that an upper bound on the error is less than 10^{-3}.

For the Composite Trapezoidal Rule, the error is

$$E_{trap} = -\frac{f''(\eta)}{12}(b-a)h^2 = -\frac{e^\eta}{12n^2},$$

since $f(x) = e^x$, $a = 0$ and $b = 1$, which yields $h = (b-a)/n = 1/n$. Since $0 \le \eta \le 1$, and e^x is increasing, the factor e^η is bounded above by $e^1 = e$. It follows that $|E_{trap}| < 10^{-3}$ if

$$\frac{e}{12n^2} < 10^{-3} \quad \Rightarrow \quad \frac{1000e}{12} < n^2 \quad \Rightarrow \quad n > 15.0507.$$

Therefore, the error will be sufficiently small provided that we choose $n \geq 16$.

On the other hand, if we use the Composite Simpson's Rule, the error is

$$E_{simp} = -\frac{f^{(4)}(\eta)}{180}(b-a)h^4 = -\frac{e^\eta}{180n^4}$$

for some η in $[0,1]$, which is less than 10^{-3} in absolute value if

$$n > \left(\frac{1000e}{180}\right)^{1/4} \approx 1.9713,$$

so $n = 2$ is sufficient. That is, we can approximate the integral to 3 decimal places by setting $h = (b-a)/n = (1-0)/2 = 1/2$ and computing

$$\int_0^1 e^x\,dx \approx \frac{h}{3}[e^{x_0} + 4e^{x_1} + e^{x_2}] = \frac{1/2}{3}[e^0 + 4e^{1/2} + e^1] \approx 1.71886,$$

whereas the exact value is approximately 1.71828. □

4.7 Romberg Integration

Richardson extrapolation is not only used to compute more accurate approximations of derivatives, but is also used as the foundation of a numerical integration scheme called *Romberg integration*. In this scheme, the integral

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

is approximated using the Composite Trapezoidal Rule with step sizes $h_k = (b-a)2^{-k}$, where k is a nonnegative integer. Then, for each k, Richardson extrapolation is used $k-1$ times to previously computed approximations in order to improve the order of accuracy as much as possible.

More precisely, suppose that we compute approximations $T_{1,1}$ and $T_{2,1}$ to the integral, using the Composite Trapezoidal Rule with one and two subintervals, respectively. That is,

$$\begin{aligned} T_{1,1} &= \frac{b-a}{2}[f(a) + f(b)] \\ T_{2,1} &= \frac{b-a}{4}\left[f(a) + 2f\left(\frac{a+b}{2}\right) + f(b)\right]. \end{aligned}$$

Suppose that f has continuous derivatives of all orders on $[a, b]$. Then, the Composite Trapezoidal Rule, for a general number of subintervals n, satisfies

$$\int_a^b f(x)\,dx = \frac{h}{2}\left[f(a) + 2\sum_{j=1}^{n-1} f(x_j) + f(b)\right] + \sum_{i=1}^{\infty} K_i h^{2i},$$

where $h = (b-a)/n$, $x_j = a + jh$, and the constants $\{K_i\}_{i=1}^{\infty}$ depend only on the derivatives of f. It follows that we can use Richardson Extrapolation to compute an approximation with a higher order of accuracy. If we denote the exact value of the integral by $I(f)$ then we have

$$\begin{aligned} T_{1,1} &= I(f) + K_1 h^2 + O(h^4) \\ T_{2,1} &= I(f) + K_1 (h/2)^2 + O(h^4) \end{aligned}$$

Neglecting the $O(h^4)$ terms, we have a system of equations that we can solve for K_1 and $I(f)$. The value of $I(f)$, which we denote by $T_{2,2}$, is an improved approximation given by

$$T_{2,2} = T_{2,1} + \frac{T_{2,1} - T_{1,1}}{3}.$$

It follows from the representation of the error in the Composite Trapezoidal Rule that $I(f) = T_{2,2} + O(h^4)$.

Suppose that we compute another approximation $T_{3,1}$ using the Composite Trapezoidal Rule with 4 subintervals. Then, as before, we can use Richardson Extrapolation with $T_{2,1}$ and $T_{3,1}$ to obtain a new approximation $T_{3,2}$ that is fourth-order accurate. Now, however, we have two approximations, $T_{2,2}$ and $T_{3,2}$, that satisfy

$$\begin{aligned} T_{2,2} &= I(f) + \tilde{K}_2 h^4 + O(h^6) \\ T_{3,2} &= I(f) + \tilde{K}_2 (h/2)^4 + O(h^6) \end{aligned}$$

for some constant $\tilde{K}_2$. It follows that we can apply Richardson Extrapolation to these approximations to obtain a new approximation $T_{3,3}$ that is *sixth-order* accurate. We can continue this process to obtain as high an order of accuracy as we wish. We now describe the entire algorithm.

Algorithm (Romberg Integration) Given a positive integer J, an interval $[a, b]$ and a function $f(x)$, the following algorithm computes an approximation to $I(f) = \int_a^b f(x)\,dx$ that is accurate to order $2J$.

$h = b - a$
for $j = 1, 2, \ldots, J$ **do**
  $T_{j,1} = \frac{h}{2}\left[f(a) + 2\sum_{j=1}^{2^{j-1}-1} f(a + jh) + f(b)\right]$ (Composite Trapezoidal Rule)
  for $k = 2, 3, \ldots, j$ **do**
    $T_{j,k} = T_{j,k-1} + \frac{T_{j,k-1} - T_{j-1,k-1}}{4^{k-1}-1}$ (Richardson Extrapolation)
  end
  $h = h/2$
end

It should be noted that in a practical implementation, $T_{j,1}$ can be computed more efficiently by using $T_{j-1,1}$, because $T_{j-1,1}$ already includes more than half of the function values used to compute $T_{j,1}$, and they are weighted correctly relative to one another. It follows that for $j > 1$, if we split the summation in the algorithm into two summations containing odd- and even-numbered terms, respectively, we obtain

$$\begin{aligned} T_{j,1} &= \frac{h}{2}\left[f(a) + 2\sum_{j=1}^{2^{j-2}} f(a + (2j-1)h) + 2\sum_{j=1}^{2^{j-2}-1} f(a + 2jh) + f(b)\right] \\ &= \frac{h}{2}\left[f(a) + 2\sum_{j=1}^{2^{j-2}-1} f(a + 2jh) + f(b)\right] + \frac{h}{2}\left[2\sum_{j=1}^{2^{j-2}} f(a + (2j-1)h)\right] \\ &= \frac{1}{2}T_{j-1,1} + h\sum_{j=1}^{2^{j-2}} f(a + (2j-1)h). \end{aligned}$$

Example We will use *Romberg integration* to obtain a sixth-order accurate approximation to

$$\int_0^1 e^{-x^2}\,dx,$$

an integral that *cannot* be computed using the Fundamental Theorem of Calculus. We begin by using the Trapezoidal Rule, or, equivalently, the Composite Trapezoidal Rule

$$\int_a^b f(x)\,dx \approx \frac{h}{2}\left[f(a) + \sum_{j=1}^{n-1} f(x_j) + f(b)\right], \quad h = \frac{b-a}{n}, \quad x_j = a + jh,$$

with $n = 1$ subintervals. Since $h = (b-a)/n = (1-0)/1 = 1$, we have

$$R_{1,1} = \frac{1}{2}[f(0) + f(1)] = 0.68393972058572,$$

which has an absolute error of 6.3×10^{-2}.

If we bisect the interval $[0, 1]$ into two subintervals of equal width, and approximate the area under e^{-x^2} using two trapezoids, then we are applying the Composite Trapezoidal Rule with $n = 2$ and $h = (1-0)/2 = 1/2$, which yields

$$R_{2,1} = \frac{0.5}{2}[f(0) + 2f(0.5) + f(1)] = 0.73137025182856,$$

which has an absolute error of 1.5×10^{-2}. As expected, the error is reduced by a factor of 4 when the step size is halved, since the error in the Composite Trapezoidal Rule is of $O(h^2)$.

Now, we can use Richardson Extrapolation to obtain a more accurate approximation,

$$R_{2,2} = R_{2,1} + \frac{R_{2,1} - R_{1,1}}{3} = 0.74718042890951,$$

which has an absolute error of 3.6×10^{-4}. Because the error in the Composite Trapezoidal Rule satisfies

$$\int_a^b f(x)\,dx = \frac{h}{2}\left[f(a) + \sum_{j=1}^{n-1} f(x_j) + f(b)\right] + K_1 h^2 + K_2 h^4 + K_3 h^6 + O(h^8),$$

where the constants K_1, K_2 and K_3 depend on the derivatives of $f(x)$ on $[a, b]$ and are independent of h, we can conclude that $R_{2,1}$ has fourth-order accuracy.

We can obtain a second approximation of fourth-order accuracy by using the Composite Trapezoidal Rule with $n = 4$ to obtain a third approximation of second-order accuracy. We set $h = (1 - 0)/4 = 1/4$, and then compute

$$R_{3,1} = \frac{0.25}{2}\left[f(0) + 2[f(0.25) + f(0.5) + f(0.75)] + f(1)\right] = 0.74298409780038,$$

which has an absolute error of 3.8×10^{-3}. Now, we can apply Richardson Extrapolation to $R_{2,1}$ and $R_{3,1}$ to obtain

$$R_{3,2} = R_{3,1} + \frac{R_{3,1} - R_{2,1}}{3} = 0.74685537979099,$$

which has an absolute error of 3.1×10^{-5}. This significant decrease in error from $R_{2,2}$ is to be expected, since both $R_{2,2}$ and $R_{3,2}$ have fourth-order accuracy, and $R_{3,2}$ is computed using half the step size of $R_{2,2}$.

It follows from the error term in the Composite Trapezoidal Rule, and the formula for Richardson Extrapolation, that

$$R_{2,2} = \int_0^1 e^{-x^2}\,dx + \tilde{K}_2 h^4 + O(h^6), \quad R_{2,2} = \int_0^1 e^{-x^2}\,dx + \tilde{K}_2 \left(\frac{h}{2}\right)^4 + O(h^6).$$

Therefore, we can use Richardson Extrapolation with these two approximations to obtain a new approximation

$$R_{3,3} = R_{3,2} + \frac{R_{3,2} - R_{2,2}}{2^4 - 1} = 0.74683370984975,$$

which has an absolute error of 9.6×10^{-6}. Because $R_{3,3}$ is a linear combination of $R_{3,2}$ and $R_{2,2}$ in which the terms of order h^4 cancel, we can conclude that $R_{3,3}$ is of sixth-order accuracy. □

4.8 Adaptive Quadrature

Composite rules can be used to implement an *automatic* quadrature procedure, in which the all of the subintervals of $[a, b]$ are continually subdivided until sufficient accuracy is achieved. However, this approach is impractical since small subintervals are not necessary in regions where the integrand is smooth.

An alternative is *adaptive quadrature*. Adaptive quadrature is a technique in which the interval $[a, b]$ is divided into n subintervals $[a_j, b_j]$, for $j = 0, 1, \ldots, n-1$, and a quadrature rule, such as the Trapezoidal Rule, is used on each subinterval to compute

$$I_j(f) = \int_{a_j}^{b_j} f(x)\,dx,$$

as in any composite quadrature rule. However, in adaptive quadrature, a subinterval $[a_j, b_j]$ is subdivided if it is determined that the quadrature rule has not computed $I_j(f)$ with sufficient accuracy.

To make this determination, we use the quadrature rule on $[a_j, b_j]$ to obtain an approximation I_1, and then use the corresponding composite rule on $[a_j, b_j]$, with two subintervals, to compute a second approximation I_2. If I_1 and I_2 are sufficiently close, then it is reasonable to conclude that these two approximations are accurate, so there is no need to subdivide $[a_j, b_j]$. Otherwise, we divide $[a_j, b_j]$ into two subintervals, and repeat this process on these subintervals. We apply this technique to all subintervals, until

we can determine that the integral of f over each one has been computed with sufficient accuracy. By subdividing only when it is necessary, we avoid unnecessary computation and obtain the same accuracy as with composite rules or automatic quadrature procedures, but with much less computational effort.

We now describe an algorithm for adaptive quadrature. This algorithm uses the Trapezoidal Rule to integrate over intervals, and intervals are subdivided as necessary into two subintervals of equal width. The algorithm uses a data structure called a *stack* in order to keep track of the subintervals over which f still needs to be integrated. A stack is essentially a list of elements, where the elements, in this case, are subintervals. An element is added to the stack using a *push* operation, and is removed using a *pop* operation. A stack is often described using the phrase "last-in-first-out," because the most recent element to be pushed onto the stack is the first element to be popped. This corresponds to our intuitive notion of a stack of objects, in which objects are placed on top of the stack and are removed from the top as well.

Algorithm (Adaptive Quadrature) Given a function $f(x)$ that is integrable on an interval $[a, b]$, the following algorithm computes an approximation I to $I(f) = \int_a^b f(x)\,dx$ that is accurate to within $(b-a)TOL$, where TOL is a given error tolerance.

S is an empty stack
$push(S, [a, b])$
$I = 0$
while S is not empty **do**
    $[a, b] = pop(S)$ (the interval $[a, b]$ on top of S is removed from S)
    $I_1 = [(b-a)/2][f(a) + f(b)]$ (Trapezoidal Rule)
    $m = (a+b)/2$
    $I_2 = [(b-a)/4][f(a) + 2f(m) + f(b)]$ (Composite Trapezoidal Rule with 2 subintervals)
    if $|I_1 - I_2| < 3(b-a)TOL$ **then**
        $I = I + I_2$ (from error term in Trapezoidal Rule, $|I(f) - I_2| \approx \frac{1}{3}|I_1 - I_2|$)
    else
        $push(S, [a, m])$
        $push(S, [m, b])$
    end
end

Throughout the execution of the loop in the above algorithm, the stack S contains all intervals over which f still needs to be integrated to within

the desired accuracy. Initially, the only such interval is the original interval $[a, b]$. As long as intervals remain in the stack, the interval on top of the stack is removed, and we attempt to integrate over it. If we obtain a sufficiently accurate result, then we are finished with the interval. Otherwise, the interval is bisected into two subintervals, both of which are pushed on the stack so that they can be processed later. Once the stack is empty, we know that we have accurately integrated f over a collection of intervals whose union is the original interval $[a, b]$, so the algorithm can terminate.

Example We will use *adaptive quadrature* to compute the integral

$$\int_0^{\pi/4} e^{3x} \sin 2x \, dx$$

to within $(\pi/4)10^{-4}$.

Let $f(x) = e^{3x} \sin 2x$ denote the integrand. First, we use Simpson's Rule, or, equivalently, the Composite Simpson's Rule with $n = 2$ subintervals, to obtain an approximation I_1 to this integral. We have

$$I_1 = \frac{\pi/4}{6}[f(0) + 4f(\pi/8) + f(\pi/4)] = 2.58369640324748.$$

Then, we divide the interval $[0, \pi/4]$ into two subintervals of equal width, $[0, \pi/8]$ and $[\pi/8, \pi/4]$, and integrate over each one using Simpson's Rule to obtain a second approximation I_2. This is equivalent to using the Composite Simpson's Rule on $[0, \pi/4]$ with $n = 4$ subintervals. We obtain

$$\begin{aligned} I_2 &= \frac{\pi/8}{6}[f(0) + 4f(\pi/16) + f(\pi/8)] + \frac{\pi/8}{6}[f(\pi/8) + 4f(3\pi/16) + f(\pi/4)] \\ &= \frac{\pi/16}{3}[f(0) + 4f(\pi/16) + 2f(\pi/8) + 4f(3\pi/16) + f(\pi/4)] \\ &= 2.58770145345862. \end{aligned}$$

Now, we need to determine whether the approximation I_2 is sufficiently accurate. Because the error in the Composite Simpson's Rule is $O(h^4)$, where h is the width of each subinterval used in the rule, it follows that the actual error in I_2 satisfies

$$|I_2 - I(f)| \approx \frac{1}{15}|I_2 - I_1|,$$

where $I_f(f)$ is the exact value of the integral of f.

We find that the relation

$$|I_2 - I(f)| \approx \frac{1}{15}|I_2 - I_1| < \frac{\pi}{4}10^{-4}$$

is *not* satisfied, so we must divide the interval $[0, \pi/4]$ into two subintervals of equal width, $[0, \pi/8]$ and $[\pi/8, \pi/4]$, and use the Composite Simpson's Rule with these smaller intervals in order to achieve the desired accuracy.

First, we work with the interval $[0, \pi/8]$. Proceeding as before, we use the Composite Simpson's Rule with $n = 2$ and $n = 4$ subintervals to obtain approximations I_1 and I_2 to the integral of $f(x)$ over this interval. We have

$$I_1 = \frac{\pi/8}{6}[f(0) + 4f(\pi/16) + f(\pi/8)] = 0.33088926959519.$$

and

$$\begin{aligned} I_2 &= \frac{\pi/16}{6}[f(0) + 4f(\pi/32) + f(\pi/16)] + \frac{\pi/16}{6}[f(\pi/16) + 4f(3\pi/32) + f(\pi/8)] \\ &= \frac{\pi/32}{3}[f(0) + 4f(\pi/32) + 2f(\pi/16) + 4f(3\pi/32) + f(\pi/8)] \\ &= 0.33054510467064. \end{aligned}$$

Since these approximations satisfy the relation

$$|I_2 - I(f)| \approx \frac{1}{15}|I_2 - I_1| < \frac{\pi}{8}10^{-4},$$

where $I(f)$ denotes the exact value of the integral of f over $[0, \pi/8]$, we have achieved sufficient accuracy on this interval and we do not need to subdivide it further. The more accurate approximation I_2 can be included in our approximation to the integral over the original interval $[0, \pi/4]$.

Now, we need to achieve sufficient accuracy on the remaining subinterval, $[\pi/4, \pi/8]$. As before, we compute the approximations I_1 and I_2 of the integral of f over this interval and obtain

$$I_1 = \frac{\pi/8}{6}[f(\pi/8) + 4f(3\pi/16) + f(\pi/4)] = 2.25681218386343.$$

and

$$\begin{aligned} I_2 &= \frac{\pi/16}{6}[f(\pi/8) + 4f(5\pi/32) + f(3\pi/16)] + \frac{\pi/16}{6}[f(3\pi/16) + 4f(7\pi/32) + f(\pi/4)] \\ &= \frac{\pi/32}{3}[f(\pi/8) + 4f(5\pi/32) + 2f(3\pi/16) + 4f(7\pi/32) + f(\pi/4)] \\ &= 2.25801455892266. \end{aligned}$$

Since these approximations do not satisfy the relation

$$|I_2 - I(f)| \approx \frac{1}{15}|I_2 - I_1| < \frac{\pi}{8}10^{-4},$$

where $I(f)$ denotes the exact value of the integral of f over $[\pi/8, \pi/4]$, we have not achieved sufficient accuracy on this interval and we need to subdivide it into two subintervals of equal width, $[\pi/8, 3\pi/16]$ and $[3\pi/16, \pi/4]$, and use the Composite Simpson's Rule with these smaller intervals in order to achieve the desired accuracy.

The discrepancy in these two approximations to the integral of f over $[\pi/4, \pi/8]$ is larger than the discrepancy in the two approximations of the integral over $[0, \pi/8]$ because even though these intervals have the same width, the derivatives of f are larger on $[\pi/8, \pi/4]$, and therefore the error in the Composite Simpson's Rule is larger.

We continue the process of adaptive quadrature on the interval $[\pi/8, 3\pi/16]$. As before, we compute the approximations I_1 and I_2 of the integral of f over this interval and obtain

$$I_1 = \frac{\pi/16}{6}[f(\pi/8) + 4f(5\pi/32) + f(3\pi/16)] = 0.72676545197054.$$

and

$$\begin{aligned} I_2 &= \frac{\pi/32}{6}[f(\pi/8) + 4f(9\pi/64) + f(5\pi/32)] + \frac{\pi/32}{6}[f(5\pi/32) + 4f(11\pi/64) + f(3\pi/16)] \\ &= \frac{\pi/64}{3}[f(\pi/8) + 4f(9\pi/64) + 2f(5\pi/32) + 4f(11\pi/64) + f(3\pi/16)] \\ &= 0.72677918153379. \end{aligned}$$

Since these approximations satisfy the relation

$$|I_2 - I(f)| \approx \frac{1}{15}|I_2 - I_1| < \frac{\pi}{16}10^{-4},$$

where $I(f)$ denotes the exact value of the integral of f over $[\pi/8, 3\pi/16]$, we have achieved sufficient accuracy on this interval and we do not need to subdivide it further. The more accurate approximation I_2 can be included in our approximation to the integral over the original interval $[0, \pi/4]$.

Now, we work with the interval $[3\pi/16, \pi/4]$. Proceeding as before, we use the Composite Simpson's Rule with $n = 2$ and $n = 4$ subintervals to obtain approximations I_1 and I_2 to the integral of $f(x)$ over this interval. We have

$$I_1 = \frac{\pi/16}{6}[f(3\pi/16) + 4f(7\pi/32) + f(\pi/4)] = 1.53124910695212.$$

and

$$I_2 = \frac{\pi/32}{6}[f(3\pi/16) + 4f(13\pi/64) + f(7\pi/32)] + \frac{\pi/32}{6}[f(7\pi/32) + 4f(15\pi/64) + f(\pi/4)]$$

$$\begin{aligned} &= \frac{\pi/64}{3}[f(3\pi/16) + 4f(13\pi/64) + 2f(7\pi/32) + 4f(15\pi/64) + f(\pi/4)] \\ &= 1.53131941583939. \end{aligned}$$

Since these approximations satisfy the relation

$$|I_2 - I(f)| \approx \frac{1}{15}|I_2 - I_1| < \frac{\pi}{16}10^{-4},$$

where $I(f)$ denotes the exact value of the integral of f over $[3\pi/16, \pi/4]$, we have achieved sufficient accuracy on this interval and we do not need to subdivide it further. The more accurate approximation I_2 can be included in our approximation to the integral over the original interval $[0, \pi/4]$.

We conclude that the integral of $f(x)$ over $[0, \pi/4]$ can be approximated by the sum of our approximate integrals over $[0, \pi/8]$, $[\pi/8, 3\pi/16]$, and $[3\pi/16, \pi/4]$, which yields

$$\begin{aligned} \int_0^{\pi/4} e^{3x} \sin 2x \, dx &\approx 0.33054510467064 + 0.72677918153379 + 1.53131941583939 \\ &\approx 2.58864370204382. \end{aligned}$$

Since the exact value is 2.58862863250716, the absolute error is -1.507×10^{-5}, which is less in magnitude than our desired error bound of $(\pi/4)10^{-4} \approx 7.854 \times 10^{-5}$. This is because on each subinterval, we ensured that our approximation was accurate to within 10^{-4} times the width of the subinterval, so that when we added these approximations, the total error in the sum would be bounded by 10^{-4} times the width of the union of these subintervals, which is the original interval $[0, \pi/4]$. □

Adaptive quadrature can be very effective, but it should be used cautiously, for the following reasons:

- The integrand is only evaluated at a few points within each subinterval. Such sampling can miss a portion of the integrand whose contribution to the integral can be misjudged.
- Regions in which the function is not smooth will still only make a small contribution to the integral if the region itself is small, so this should be taken into account to avoid unnecessary function evaluations.
- Adaptive quadrature can be very inefficient if the integrand has a discontinuity within a subinterval, since repeated subdivision will occur. This is unnecessary if the integrand is smooth on either side of the discontinuity, so subintervals should be chosen so that discontinuities occur between subintervals whenever possible.

4.9 Gaussian Quadrature

In Lecture 26, we learned that a Newton-Cotes quadrature rule with n nodes has degree at most n. Therefore, it is natural to ask whether it is possible to select the nodes and weights of an n-point quadrature rule so that the rule has degree greater than n. *Gaussian quadrature rules* have the surprising property that they can be used to integrate polynomials of degree $2n-1$ exactly using only n nodes.

Gaussian quadrature rules can be constructed using a technique known as *moment matching*. For any nonnegative integer k, the k^{th} moment is defined to be

$$\mu_k = \int_a^b x^k \, dx.$$

For given n, our goal is to select weights and nodes so that the first $2n$ moments are computed exactly; i.e.,

$$\mu_k = \sum_{i=0}^{n-1} w_i x_i^k, \quad k = 0, 1, \ldots, 2n-1.$$

Since we have $2n$ free parameters, it is reasonable to think that appropriate nodes and weights can be found. Unfortunately, this system of equations is nonlinear, so we cannot be certain that a solution exists.

Suppose $g(x)$ is a polynomial of degree $2n-1$. For convenience, we will write $g \in \mathcal{P}_{2n-1}$, where, for any natural number k, $\mathcal{P}_k$ denotes the space of polynomials of degree at most k. We shall show that there exist weights $\{w_i\}_{i=0}^{n-1}$ and nodes $\{x_i\}_{i=0}^{n-1}$ such that

$$\int_a^b g(x) \, dx = \sum_{i=0}^{n-1} w_i g(x_i).$$

Furthermore, for more general functions, $G(x)$,

$$\int_a^b G(x) \, dx = \sum_{i=0}^{n-1} w_i G(x_i) + E[G]$$

where

1. x_i are real, distinct, and $a < x_i < b$ for $i = 0, 1, \ldots, n-1$.
2. The weights $\{w_i\}$ satisfy $w_i > 0$ for $i = 0, 1, \ldots, n-1$.

3. The error $E[G]$ satisfies $E[G] = \frac{G^{(2n)}(\xi)}{(2n)!} \int_a^b \prod_{i=0}^{n-1}(x - x_i)^2\,dx$.

Notice that this method is exact for polynomials of degree $2n-1$ since the error functional $E[G]$ depends on the $(2n)^{th}$ derivative of G.

To prove this, we shall construct an *orthonormal family* of polynomials $\{q_i(x)\}_{i=0}^n$ so that

$$\int_a^b q_r(x)q_s(x)\,dx = \begin{cases} 0 & r \neq s, \\ 1 & r = s. \end{cases}$$

This can be accomplished using the fact that such a family of polynomials satisfies a *three-term recurrence relation*

$$\beta_j q_{j+1}(x) = (x-\alpha_j)q_j(x) - \beta_{j-1}q_{j-1}(x), \quad q_0(x) = (b-a)^{-1/2}, \quad q_{-1}(x) = 0,$$

where

$$\alpha_j = \int_a^b x q_j(x)^2\,dx, \quad \beta_j = \int_a^b x q_{j+1}(x)q_j(x)\,dx.$$

We choose the nodes $\{x_i\}$ to be the roots of the n^{th}-degree polynomial in this family. Next, we construct the interpolant of degree $n-1$, denoted $p_{n-1}(x)$, of $g(x)$ through the nodes:

$$L_{n-1}(x) = \sum_{i=0}^{n-1} g(x_i)\mathcal{L}_{n-1,i}(x),$$

where, for $i = 0, \ldots, n-1$, $\mathcal{L}_{n-1,i}(x)$ is the ith Lagrange polynomial for the points $x_0, \ldots, x_{n-1}$. We shall now look at the interpolation error function

$$e(x) = g(x) - p_{n-1}(x).$$

Clearly, since $g \in \mathcal{P}_{2n-1}$, $e \in \mathcal{P}_{2n-1}$. Since $e(x)$ has roots at each of the roots of $q_n(x)$, we can factor e so that

$$e(x) = q_n(x)r(x),$$

where $r \in \mathcal{P}_{n-1}$. It follows from the fact that $q_n(x)$ is orthogonal to *any* polynomial in $\mathcal{P}_{n-1}$ that the integral of g can then be written as

$$\begin{aligned} I(g) &= \int_a^b p_{n-1}(x)\,dx + \int_a^b q_n(x)r(x)\,dx \\ &= \int_a^b p_{n-1}(x)\,dx \end{aligned}$$

$$\begin{aligned} &= \int_a^b \sum_{i=0}^{n-1} g(x_i)\mathcal{L}_{n-1,i}(x)\,dx \\ &= \sum_{i=0}^{n-1} g(x_i) \int_a^b \mathcal{L}_{n-1,i}(x)\,dx \\ &= \sum_{i=0}^{n-1} g(x_i) w_i \end{aligned}$$

where

$$w_i = \int_a^b \mathcal{L}_{n-1,i}(x)\,dx, \quad i = 0, 1, \ldots, n-1.$$

For a more general function $G(x)$, the error functional $E[g]$ can be obtained from the expression for the interpolation error presented in Lecture 16.

It is easy to show that the weights are positive. Since the interpolation basis functions $\mathcal{L}_{n-1,i}$ belong to $\mathcal{P}_{n-1}$, it follows that $\mathcal{L}_{n-1,i}^2 \in \mathcal{P}_{2n-2}$, and therefore

$$0 < \int_a^b \mathcal{L}_{n-1,i}^2(x)\,dx = \sum_{j=0}^{n-1} w_j \mathcal{L}_{n-1,i}^2(x_j) = w_i.$$

Also, we can easily obtain qualitative bounds on the error. For instance, if we know that the even derivatives of g are positive, then we know that the quadrature rule yields an upper bound for $I(g)$. Similarly, if the even derivatives of g are negative, then the quadrature rule gives a lower bound.

Example We will use *Gaussian quadrature* to approximate the integral

$$\int_0^1 e^{-x^2}\,dx.$$

The particular Gaussian quadrature rule that we will use consists of 5 nodes x_1, x_2, x_3, x_4 and x_5, and 5 weights w_1, w_2, w_3, w_4 and w_5. To determine the proper nodes and weights, we use the fact that the nodes and weights of a 5-point Gaussian rule for integrating over the interval $[-1, 1]$ are given by

i	Nodes $r_{5,i}$	Weights $c_{5,i}$
1	0.9061798459	0.2369268850
2	0.5384693101	0.4786286705
3	0.0000000000	0.5688888889
4	−0.5384693101	0.4786286705
5	−0.9061798459	0.2369268850

To obtain the corresponding nodes and weights for integrating over $[-1, 1]$, we can use the fact that in general,

$$\int_a^b f(x)\,dx = \int_{-1}^1 f\left(\frac{b-a}{2}t + \frac{a+b}{2}\right)\frac{b-a}{2}\,dt,$$

as can be shown using the change of variable $x = [(b-a)/2]t + (a+b)/2$ that maps $[a, b]$ into $[-1, 1]$. We then have

$$\begin{aligned}\int_a^b f(x)\,dx &= \int_{-1}^1 f\left(\frac{b-a}{2}t + \frac{a+b}{2}\right)\frac{b-a}{2}\,dt \\ &\approx \sum_{i=1}^5 f\left(\frac{b-a}{2}r_{5,i} + \frac{a+b}{2}\right)\frac{b-a}{2}c_{5,i} \\ &\approx \sum_{i=1}^5 f(x_i)w_i,\end{aligned}$$

where

$$x_i = \frac{b-a}{2}r_{5,i} + \frac{a+b}{2}, \quad w_i = \frac{b-a}{2}c_{5,i}, \quad i = 1, \ldots, 5.$$

In this example, $a = 0$ and $b = 1$, so the nodes and weights for a 5-point Gaussian quadrature rule for integrating over $[0, 1]$ are given by

$$x_i = \frac{1}{2}r_{5,i} + \frac{1}{2}, \quad w_i = \frac{1}{2}c_{5,i}, \quad i = 1, \ldots, 5,$$

which yields

i	Nodes x_i	Weights w_i
1	0.95308992295	0.11846344250
2	0.76923465505	0.23931433525
3	0.50000000000	0.28444444444
4	0.23076534495	0.23931433525
5	0.04691007705	0.11846344250

It follows that

$$\begin{aligned}\int_0^1 e^{-x^2}\,dx &\approx \sum_{i=1}^5 e^{-x_i^2}w_i \\ &\approx 0.11846344250e^{-0.95308992295^2} + 0.23931433525e^{-0.76923465505^2} + \\ &\quad 0.28444444444e^{-0.5^2} + 0.23931433525e^{-0.23076534495^2} + \\ &\quad 0.11846344250e^{-0.04691007705^2} \\ &\approx 0.74682412673352.\end{aligned}$$

Since the exact value is 0.74682413281243, the absolute error is -6.08×10^{-9}, which is remarkably accurate considering that only fives nodes are used. □

The high degree of accuracy of Gaussian quadrature rules make them the most commonly used rules in practice. However, they are not without their drawbacks:

- They are not progressive, so the nodes must be recomputed whenever additional degrees of accuracy are desired. An alternative is to use *Gauss-Kronrod rules*. A $(2n + 1)$-point Gauss-Kronrod rule uses the nodes of the n-point Gaussian rule. For this reason, practical quadrature procedures use both the Gaussian rule and the corresponding Gauss-Kronrod rule to estimate accuracy.

- Because the nodes are the roots of a polynomial, they must be computed using traditional root-finding methods, which are not always accurate. Errors in the computed nodes lead to lost degrees of accuracy in the approximate integral. In practice, however, this does not normally cause significant difficulty.

Often, variations of Gaussian quadrature rules are used in which one or more nodes are prescribed. For example, *Gauss-Radau rules* are rules in which either of the endpoints of the interval $[a, b]$ are chosen to be a node, and n additional nodes are determined by a procedure similar to that used in Gaussian quadrature, resulting in a rule of degree $2n$. In *Gauss-Lobatto rules*, both endpoints of $[a, b]$ are nodes, with n additional nodes chosen in order to obtain a rule of degree $2n+1$. It should be noted that Gauss-Lobatto rules are closed, whereas Gaussian rules are open.

4.10 Multiple Integrals

4.10.1 Double Integrals

As many problems in scientific computing involve two-dimensional domains, it is essential to be able to compute integrals over such domains. Such integrals can be evaluated using the following strategies:

- If a two-dimensional domain Ω can be decomposed into rectangles, then the integral of a function $f(x, y)$ over Ω can be computed by evaluating integrals of the form

$$I(f) = \int_a^b \int_c^d f(x, y)\, dy\, dx.$$

Then, to evaluate $I(f)$, one can use a *Cartesian product rule*, whose nodes and weights are obtained by combining one-dimensional quadrature rules that are applied to each dimension. For example, if functions of x are integrated along the line between $x = a$ and $x = b$ using nodes x_i and weights w_i, for $i = 1, \ldots, n$, and if functions of y are integrated along the line between $y = c$ and $y = d$ using nodes y_i and weights z_i, for $i = 1, \ldots, m$, then the resulting Cartesian product rule

$$Q_{n,m}(f) = \sum_{i=1}^{n} \sum_{j=1}^{m} f(x_i, y_j) w_i z_j$$

has nodes (x_i, y_j) and corresponding weights $w_i z_j$ for $i = 1, \ldots, n$ and $j = 1, \ldots, m$.

- If the domain Ω can be described as the region between two curves $y_1(x)$ and $y_2(x)$ for $x \in [a, b]$, then we can write

$$I(f) = \int \int_{\Omega} f(x, y)\, dA$$

as an *iterated integral*

$$I(f) = \int_a^b \int_{y_1(x)}^{y_2(x)} f(x, y)\, dy\, dx$$

which can be evaluated by applying a one-dimensional quadrature rule to compute the *outer integral*

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

where $g(x)$ is evaluated by using a one-dimensional quadrature rule to compute the *inner integral*

$$g(x) = \int_{y_1(x)}^{y_2(x)} f(x, y)\, dy.$$

- For various simple regions such as triangles, there exist *cubature rules* that are not combinations of one-dimensional quadrature rules. Cubature rules are more direct generalizations of quadrature rules, in that they evaluate the integrand at selected nodes and use weights determined by the geometry of the domain and the placement of the nodes.

It should be noted that all of these strategies apply to certain special cases. The first algorithm capable of integrating over a general two-dimensional domain was developed by Lambers and Rice. This algorithm combines the second and third strategies described above, decomposing the domain into subdomains that are either triangles or regions between two curves.

Example We will use the Composite Trapezoidal Rule with $m = n = 2$ to evaluate the double integral

$$\int_0^{1/2} \int_0^{1/2} e^{y-x}\,dy\,dx.$$

The Composite Trapezoidal Rule with $n = 2$ subintervals is

$$\int_a^b f(x)\,dx \approx \frac{h}{2}\left[f(a) + 2f\left(\frac{a+b}{2}\right) + f(b)\right], \quad h = \frac{b-a}{n}.$$

If $a = 0$ and $b = 1/2$, then $h = (1/2 - 0)/2 = 1/4$ and this simplifies to

$$\int_0^{1/2} f(x)\,dx \approx \frac{1}{8}[f(0) + 2f(1/4) + f(1/2)].$$

We first use this rule to evaluate the "single" integral

$$\int_0^{1/2} g(x)\,dx$$

where

$$g(x) = \int_0^1 e^{y-x}\,dy.$$

This yields

$$\begin{aligned}
\int_0^{1/2} \int_0^{1/2} e^{y-x}\,dy\,dx &= \int_0^{1/2} g(x)\,dx \\
&\approx \frac{1}{8}[g(0) + 2g(1/4) + g(1/2)] \\
&\approx \frac{1}{8}\left[\int_0^{1/2} e^{y-0}\,dy + 2\int_0^{1/2} e^{y-1/4}\,dy + \int_0^{1/2} e^{y-1/2}\,dy\right].
\end{aligned}$$

Now, to evaluate each of these integrals, we use the Composite Trapezoidal Rule in the y-direction with $m = 2$. If we let k denote the step size in the

y-direction, we have $k = (1/2 - 0)/2 = 1/4$, and therefore we have

$$\begin{aligned}
\int_0^{1/2}\int_0^{1/2} e^{y-x}\,dy\,dx &\approx \frac{1}{8}\left[\int_0^{1/2} e^{y-0}\,dy + 2\int_0^{1/2} e^{y-1/4}\,dy + \int_0^{1/2} e^{y-1/2}\,dy\right] \\
&\approx \frac{1}{8}\left[\frac{1}{8}\left[e^{0-0} + 2e^{1/4-0} + e^{1/2-0}\right] + \right. \\
&\quad 2\frac{1}{8}\left[e^{0-1/4} + 2e^{1/4-1/4} + e^{1/2-1/4}\right] + \\
&\quad \left.\frac{1}{8}\left[e^{0-1/2} + 2e^{1/4-1/2} + e^{1/2-1/2}\right]\right] \\
&\approx \frac{1}{64}\left[e^0 + 2e^{1/4} + e^{1/2}\right] + \\
&\quad \frac{1}{32}\left[e^{-1/4} + 2e^0 + e^{1/4}\right] + \\
&\quad \frac{1}{64}\left[e^{-1/2} + 2e^{-1/4} + e^0\right] \\
&\approx \frac{3}{32}e^0 + \frac{1}{16}e^{-1/4} + \frac{1}{64}e^{-1/2} + \frac{1}{16}e^{1/4} + \frac{1}{64}e^{1/2} \\
&\approx 0.25791494889765.
\end{aligned}$$

The exact value, to 15 digits, is 0.255251930412762. The error is 2.66×10^{-3}, which is to be expected due to the use of few subintervals, and the fact that the Composite Trapezoidal Rule is only second-order-accurate. □

Example We will use the Composite Simpson's Rule with $n = 2$ and $m = 4$ to evaluate the double integral

$$\int_0^1 \int_x^{2x} x^2 + y^3\,dy\,dx.$$

In this case, the domain of integration described by the limits is not a rectangle, but a triangle defined by the lines $y = x$, $y = 2x$, and $x = 1$. The Composite Simpson's Rule with $n = 2$ subintervals is

$$\int_a^b f(x)\,dx \approx \frac{h}{3}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right], \quad h = \frac{b-a}{n}.$$

If $a = 0$ and $b = 1$, then $h = (1 - 0)/2 = 1/2$, and this simplifies to

$$\int_0^{1/2} f(x)\,dx \approx \frac{1}{6}[f(0) + 4f(1/2) + f(1)].$$

We first use this rule to evaluate the "single" integral

$$\int_0^1 g(x)\,dx$$

where

$$g(x) = \int_x^{2x} x^2 + y^3\,dy.$$

This yields

$$\begin{aligned}
\int_0^1 \int_x^{2x} x^2 + y^3\,dy\,dx &= \int_0^1 g(x)\,dx \\
&\approx \frac{1}{6}[g(0) + 4g(1/2) + g(1)] \\
&\approx \frac{1}{6}\left[\int_0^0 0^2 + y^3\,dy + 4\int_{1/2}^1 \left(\frac{1}{2}\right)^2 + y^3\,dy + \int_1^2 1^2 + y^3\,dy\right].
\end{aligned}$$

The first integral will be zero, since the limits of integration are equal. To evaluate the second and third integrals, we use the Composite Simpson's Rule in the y-direction with $m = 4$. If we let k denote the step size in the y-direction, we have $k = (2x - x)/4 = x/4$, and therefore we have $k = 1/8$ for the second integral and $k = 1/4$ for the third. This yields

$$\begin{aligned}
\int_0^1 \int_x^{2x} x^2 + y^3\,dy\,dx &\approx \frac{1}{6}\left[4\int_{1/2}^1 \left(\frac{1}{2}\right)^2 + y^3\,dy + \int_1^2 1^2 + y^3\,dy\right] \\
&\approx \frac{1}{6}\left\{4\frac{1}{24}\left[\left(\frac{1}{4} + \left(\frac{1}{2}\right)^3\right) + 4\left(\frac{1}{4} + \left(\frac{5}{8}\right)^3\right) + 2\left(\frac{1}{4} + \left(\frac{3}{4}\right)^3\right) + \right.\right. \\
&\quad \left.\left. 4\left(\frac{1}{4} + \left(\frac{7}{8}\right)^3\right) + \left(\frac{1}{4} + 1^3\right)\right] + \frac{1}{12}\left[\left(1 + 1^3\right) + 4\left(1 + \left(\frac{5}{4}\right)^3\right) + \right. \right. \\
&\quad \left.\left. 2\left(1 + \left(\frac{3}{2}\right)^3\right) + 4\left(1 + \left(\frac{7}{4}\right)^3\right) + \left(1 + 2^3\right)\right]\right\} \\
&\approx 1.03125.
\end{aligned}$$

The exact value is 1. The error 3.125×10^{-2} is rather large, which is to be expected due to the poor distribution of nodes through the triangular domain of integration. A better distribution is achieved if we use $n = 4$ and $m = 2$, which yields the much more accurate approximation of 1.001953125. □

4.10.2 Higher Dimensions

In more than two dimensions, generalizations of quadrature rules are not practical, since the number of function evaluations needed to attain sufficient accuracy grows very rapidly as the number of dimensions increases. An alternative is the *Monte Carlo method*, which samples the integrand at n randomly selected points and attempts to compute the mean value of the integrand on the entire domain. The method converges rather slowly but its convergence rate depends only on n, not the number of dimensions.

4.11 Improper Integrals

An *improper integral* is an integral in which either of the limits of integration, or the integrand itself, is unbounded. Although such integrals can not be defined as limits as Riemann sums, it is sometimes possible to use other limiting processes to obtain well-defined values.

We first consider the integral

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

The value of this integral is given by

$$I(f) = \lim_{b\to\infty} \int_a^b f(x)\,dx,$$

provided f is integrable on any finite interval $[a, b]$, and the above limit exists and is finite. If it does, then, for any $\epsilon > 0$, there exists a value of b such that

$$\left| I(f) - \int_a^b f(x)\,dx \right| = \left| \int_b^\infty f(x)\,dx \right| < \epsilon.$$

Therefore, $I(f)$ can be approximated by evaluating the finite integral obtained by choosing a sufficiently large value for b. An alternative approach is to use a transformation of the integration variable to obtain an integral that has finite limits.

Example We will try to approximate the improper integral

$$\int_1^\infty \frac{1}{(1+x^2)^3}\,dx.$$

There are two ways in which this can be accomplished. The first involves using the definition of an integral over an infinite interval,

$$\int_a^\infty f(x)\,dx = \lim_{b\to\infty} \int_a^b f(x)\,dx,$$

just as the definition of a derivative or an integral can be used to approximate these quantities. Specifically, we drop the limit, and assume

$$\int_a^\infty f(x)\,dx \approx \int_a^b f(x)\,dx,$$

where the value b is chosen in order to achieve sufficient accuracy. Certainly, the larger b is, the more accurate this approximation will be.

We can determine an appropriate value of b using the fact that

$$\int_a^\infty f(x)\,dx = \int_a^b f(x)\,dx + \int_b^\infty f(x)\,dx,$$

and bounding $f(x)$, for large x, by a function that can be integrated analytically from b to ∞ using the Fundamental Theorem of Calculus. This allows us to obtain a bound for the error we are introducing by truncating the interval of integration at b. In this case, we note that

$$\frac{1}{(1+x^2)^3} < \frac{2}{x^6}$$

for all real x. Therefore,

$$\int_b^\infty \frac{1}{(1+x^2)^3}\,dx < \int_b^\infty \frac{2}{x^6}\,dx = \frac{2}{5}x^{-5}\bigg|_b^\infty = \frac{2}{5b^5}.$$

Therefore, we can ensure that the integral from b to ∞ is sufficiently small simply by choosing b to be sufficiently large. Then, we can approximate the integral from a to b using any quadrature rule.

The second approach is to use a change of variable to obtain a new integral over a finite interval. In this case, if we define $t = x^{-1}$, then $t = 1$ when $x = 1$, and t approaches 0 as x approaches ∞, and therefore

$$\int_1^\infty \frac{1}{(1+x^2)^3}\,dx = -\int_1^0 \frac{1}{(1+t^{-2})^3}\left(-\frac{1}{t^2}\right)\,dt = \int_0^1 \frac{t^{-2}}{(1+t^{-2})^3}\,dt.$$

If we multiply the numerator and denominator by t^6, we obtain

$$\int_0^1 \frac{t^6 t^{-2}}{t^6(1+t^{-2})^3}\,dt = \int_0^1 \frac{t^4}{[t^2(1+t^{-2})]^3}\,dt = \int_0^1 \frac{t^4}{(t^2+1)^3}\,dt.$$

We can approximate this integral using any quadrature rule, such as the Composite Simpson's Rule with $n = 4$. Our step size h is given by $h = (1 - 0)/4 = 1/4$, and therefore we have

$$\begin{aligned}\int_1^\infty \frac{1}{(1+x^2)^3}\,dx &= \int_0^1 \frac{t^4}{(t^2+1)^3}\,dt \\ &\approx \frac{1}{12}\left[\frac{0^4}{(0^2+1)^3} + 4\frac{(1/4)^4}{((1/4)^2+1)^3} + 2\frac{(1/2)^4}{((1/2)^2+1)^3} + 4\frac{(3/4)^4}{((3/4)^2+1)^3} + \frac{1^4}{(1^2+1)^3}\right] \\ &\approx 0.04448355532940.\end{aligned}$$

The exact value, to 14 significant digits, is 0.044524311274043, so the absolute error is -4.08×10^{-5}. The relative error, however, is much larger, due to the small magnitude of the exact value; it is approximately 1×10^{-3}. □

Next, we consider the integral

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

where the integrand $f(x)$ has a singularity at some point $c \in [a, b]$, and is therefore unbounded at c. We can define the value of $I(f)$ to be

$$I(f) = \lim_{\gamma \to c^-} \int_a^\gamma f(x)\,dx + \lim_{\delta \to c^+} \int_\delta^b f(x)\,dx,$$

provided f is integrable on the indicated intervals, and the limits exist and are finite. Such an integral is most efficiently evaluated by using a transformation of the integration variable to obtain an integrand that does not have a singularity, or by subtracting the integral of an analytically integrable function that has the same singularity as $f(x)$.

Example Use the Composite Simpson's Rule with $n = 4$ to evaluate the integral

$$\int_0^1 \frac{e^x}{x^{1/4}}\,dx.$$

This is an *improper integral*, due to the singularity at $x = 0$. To work around this problem, we can rewrite the integral as

$$\int_0^1 \frac{e^x}{x^{1/4}}\,dx = \int_0^1 \frac{e^x - p_n(x)}{x^{1/4}}\,dx + \int_0^1 \frac{p_n(x)}{x^{1/4}}\,dx,$$

where $p_n(x)$ is any Taylor polynomial for e^x centered at $x = 0$. Because the error term for the Composite Simpson's Rule is only valid when the

integrand is four times continuously differentiable, we choose $n = 4$, which yields

$$\int_0^1 \frac{e^x}{x^{1/4}}\,dx = \int_0^1 \frac{e^x - 1 - x - x^2/2 - x^3/6 - x^4/24}{x^{1/4}}\,dx + \int_0^1 \frac{1 + x + x^2/2 + x^3/6 + x^4/24}{x^{1/4}}\,dx.$$

To evaluate the first integral, we define

$$G(x) = \begin{cases} \frac{e^x - p_4(x)}{x^{1/4}} & \text{if } 0 < x \le 1, \\ 0 & \text{if } x = 0. \end{cases}.$$

Then $G(x) \in C^4[0, 1]$, and

$$\int_0^1 \frac{e^x - p_4(x)}{x^{1/4}}\,dx = \int_0^1 G(x)\,dx.$$

We can approximate this integral using the Composite Simpson's Rule

$$\int_a^b f(x)\,dx \approx \frac{h}{3}\left[f(a) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \cdots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(b)\right],$$

with $n = 4$ and $h = \frac{1-0}{4} = 1/4$, which yields

$$\begin{aligned} \int_0^1 G(x)\,dx &\approx \frac{1/4}{3}[G(0) + 4G(0.25) + 2G(0.5) + 4G(0.75) + G(1)] \\ &\approx 0.00163047386619. \end{aligned}$$

The second integral can be evaluated analytically. We have

$$\begin{aligned} \int_0^1 \frac{p_4(x)}{x^{1/4}}\,dx &= \int_0^1 \frac{1 + x + x^2/2 + x^3/6 + x^4/24}{x^{1/4}}\,dx \\ &= \frac{4}{3}x^{3/4} + \frac{4}{7}x^{7/4} + \frac{2}{11}x^{11/4} + \frac{2}{45}x^{15/4} + \frac{1}{114}x^{19/4}\Big|_0^1 \\ &= \frac{4}{3} + \frac{4}{7} + \frac{2}{11} + \frac{2}{45} + \frac{1}{114} \\ &= 2.13979646084909, \end{aligned}$$

and therefore

$$\begin{aligned} \int_0^1 \frac{e^x}{x^{1/4}}\,dx &= \int_0^1 \frac{e^x - p_4(x)}{x^{1/4}}\,dx + \int_0^1 \frac{p_4(x)}{x^{1/4}}\,dx \\ &\approx 0.00163047386619 + 2.13979646084909 \\ &\approx 2.14142693471528. \end{aligned}$$

which has an error of -5.331×10^{-5}. □

Index