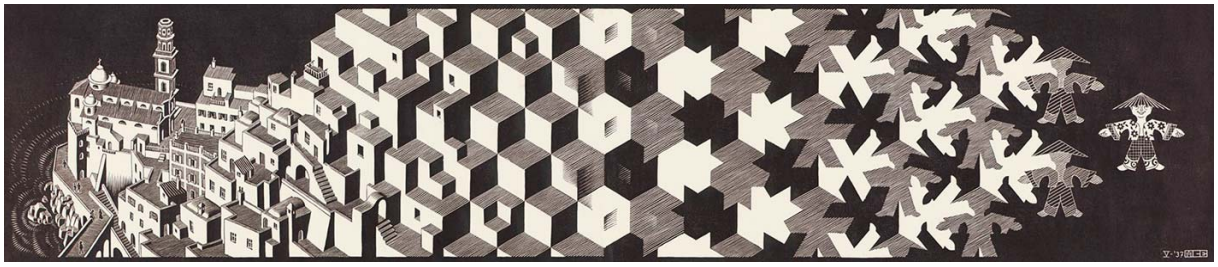


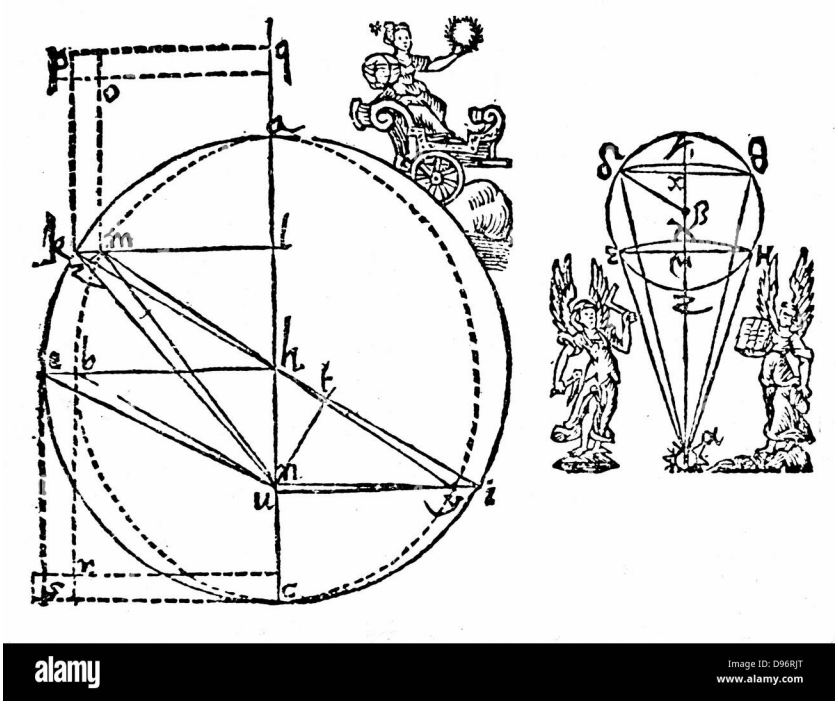
Numerical Analysis

Math 327 Lecture notes

Subhadip Chowdhury, PhD

THE COLLEGE OF WOOSTER

Spring 2022



Kepler's illustration to explain his discovery of the elliptical orbit of Mars.

From Johannes Kepler's 'Astronomia Nova', 1609, (ETH Bibliothek). In this text Kepler derives his famous equation that solves two-body orbital motion,

$$M = E - e \sin E,$$

where M (the mean anomaly) and e (the eccentricity) are known, and one solves for E (the eccentric anomaly). This vital problem spurred the development of algorithms for solving nonlinear equations.

Contents

Introduction

[*] We model our world with continuous mathematics. Whether our interest is natural science, engineering, even finance and economics, the models we most often employ are functions of real variables. The equations can be linear or nonlinear, involve derivatives, integrals, combinations of these and beyond. The tricks and techniques one learns in algebra and calculus for solving such systems *exactly* cannot tackle the complexities that arise in serious applications. Exact solution may require an intractable amount of work; worse, for many problems, it is impossible to write an exact solution using elementary functions like polynomials, roots, trig functions, and logarithms.

This course tells a marvelous success story. Through the use of clever algorithms, careful analysis, and speedy computers, we can construct *approximate* solutions to these otherwise intractable problems with remarkable speed. Trefethen[†] defines numerical analysis to be

'the study of algorithms for the problems of continuous mathematics'.

This course takes a tour through many such algorithms, sampling a variety of techniques suitable across many applications. We aim to assess alternative methods based on both accuracy and efficiency, to discern well-posed problems from ill-posed ones, and to see these methods in action through computer implementation.

Perhaps the importance of numerical analysis can be best appreciated by realizing the impact its disappearance would have on our world. The space program would evaporate; aircraft design would be hobbled; weather forecasting would again become the stuff of soothsaying and almanacs. The ultrasound technology that uncovers cancer and illuminates the womb would vanish. Google couldn't rank web pages. Even the letters you are reading, whose shapes are specified by polynomial curves, would suffer. (Several important exceptions involve discrete, not continuous, mathematics: combinatorial optimization, cryptography and gene sequencing.)

On one hand, we are interested in *complexity*: we want algorithms that minimize the number of calculations required to compute a solution. But we are also interested in the *quality* of approximation: since we do not obtain exact solutions, we must understand the accuracy of our answers. Discrepancies arise from approximating a complicated function by a polynomial, a continuum by a discrete grid of points, or the real numbers by a finite set of floating point numbers. Different algorithms for the same problem will differ in the quality of their answers and the labor required to obtain those answers; we will learn how to evaluate algorithms according to these criteria.

Numerical analysis forms the heart of 'scientific computing' or 'computational science and engineering,' fields that also encompass the high-performance computing technology that makes our algorithms practical for problems with millions of variables, visualization techniques that illuminate the data sets that emerge from these computations, and the applications that motivate them.

Though numerical analysis has flourished in the past seventy years, its roots go back centuries, where approximations were necessary in celestial mechanics and, more generally, 'natural philosophy'. Science,

[*] This section is adapted from Lecture Notes by Prof. Mark Embree at Virginia Tech.

[†] **trefethen**.

commerce, and warfare magnified the need for numerical analysis, so much so that the early twentieth century spawned the profession of 'computers', people who conducted computations with hand-crank desk calculators. But numerical analysis has always been more than mere number-crunching, as observed by Alston Householder in the introduction to his *Principles of Numerical Analysis*, published in 1953, the end of the human computer era:

> The material was assembled with high-speed digital computation always in mind, though many techniques appropriate only to ''hand'' computation are discussed. … How otherwise the continued use of these machines will transform the computer's art remains to be seen. But this much can surely be said, that their effective use demands a more profound understanding of the mathematics of the problem, and a more detailed acquaintance with the potential sources of error, than is ever required by a computation whose development can be watched, step by step, as it proceeds.

Thus the *analysis* component of 'numerical analysis' is essential. We rely on tools of classical real analysis, such as continuity, differentiability, Taylor expansion, and convergence of sequences and series.

Matrix computations play a fundamental role in numerical analysis. Discretization of continuous variables turns calculus into algebra. Algorithms for the fundamental problems in linear algebra are covered in MATH 211. This course is a prerequisite for Math 327; when the methods we discuss this semester connect to matrix techniques, we will provide pointers.

Chapter 1 | Interpolation

The idea behind interpolation is something you've practiced since Kindergarten: Connecting the dots. Now that we're all grown up, we call our "dots" data points in the xy–plane, and we connect them with very specific kinds of curves.

In the late 1950's, engineers at two rival French automakers (Paul de Casteljau at Citroen and Pierre Bézier at Renault) used a new way of interpolating data points to generate curves that could be manipulated easily to design automobiles. Their method was a trade secret until Bézier publicized the concept in 1962, which is why we now call the resulting objects "Bézier curves". The same methods are now commonly used not only in industrial design, but in computer graphics. In particular, most printers use these methods to generate fonts. Here's a schematic* that illustrates how the outline of an "G" might be drawn by interpolating 23 data points.

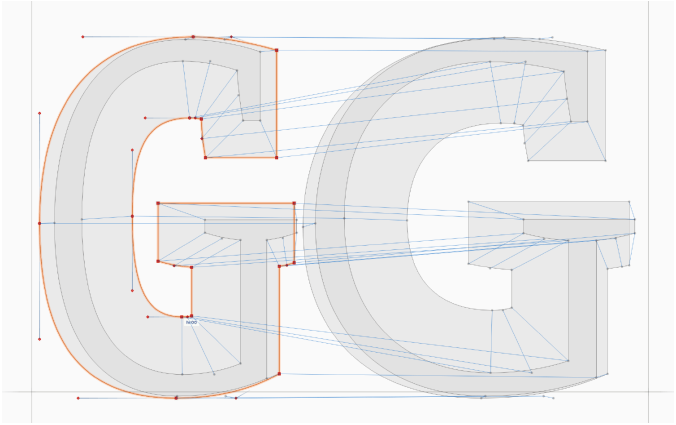


Figure 1.1: Font Curve Mathematics

Notice that the sides are all straight lines and ellipses in this case. Linear functions are first-degree polynomials, and conic sections are second degree. The reason that auto designs and printer fonts look much better than the above is because they use higher-degree polynomials in the interpolation.

§1.1 Polynomial Interpolation: definitions and notations

Definition 1.1

The set of continuous functions that map from $[a,b] \subseteq \mathbb{R}$ to $\mathbb{R}$ is denoted by $C[a,b]$. The set of continuous functions whose first r derivatives are also continuous on $[a,b]$ is denoted by $C^r[a,b]$. (Note that $C^0[a,b] \equiv C[a,b]$.)

Definition 1.2

The set of polynomials of degree n or less is denoted by $\mathcal{P}_n$.

*Taken from https://www.lucasfonts.com/learn/interpolation-curve-technicalities

Note that $C[a,b]$, $C^r[a,b]$ (for any $a < b, r \geq 0$) and $\mathcal{P}_n$ are **vector spaces** of functions (since linear combinations of such functions maintain continuity and polynomial degree).

Question 1.

What is the dimension of the vector space $\mathcal{P}_n$ (over $\mathbb{R}$)?

As a consequence, every element p_n in $\mathcal{P}_n$ can be **uniquely** determined by $(n+1)$ constants $c_i \in \mathbb{R}$ for $(0 \leq i \leq n)$, as we can write

$$p_n(x) = c_0 + c_1 x + c_2 x^2 + \ldots + c_n x^n \tag{1.1}$$

Hence $\mathcal{P}_n$ is an $n+1$ dimensional subspace of $C[a,b]$.

Working with polynomials is especially convenient in a computer as we only need to store the $n+1$ coefficients. Operations such as taking the derivative or integrating f are also easier. The idea in this chapter is to then find a polynomial that approximates a general function $f \in C[a,b]$. The following theorem due to Weierstrass says that this is always doable!

Theorem 1.3: Weierstrass Approximation Theorem

For any $f \in C[a,b]$ and any $\varepsilon > 0$, there exists a polynomial $p_n(x)$ of finite degree such that

$$\max_{x \in [a,b]} |f(x) - p_n(x)| \leq \varepsilon.$$

In this chapter, we look for a suitable polynomial p_n by using the technique of **interpolation**.

1.1.1 The Polynomial Interpolation Problem

Let's start with an example.

Question 2.

Consider the data set

$$S = \{(0,1), (1,2), (2,5), (3,11)\}.$$

If we want to fit a polynomial to this data then we can use a cubic function (which has 4 parameters) to match the data perfectly. Why is a cubic polynomial the best choice?

The polynomial interpolation problem can be stated as:

Given $f \in C[a,b]$ and $n+1$ points $\{x_j\}_{j=0}^n$ satisfying

$$a \leq x_0 < x_1 < \cdots < x_n \leq b,$$

determine some $p_n \in \mathcal{P}_n$ such that

$$p_n(x_j) = f(x_j) \quad \text{for } j = 0, \ldots, n.$$

The points $f(x_j)$ are usually called *nodes* and the polynomial p_n is called a *f-interpolating polynomial*.

It shall become clear why we require $n+1$ points $x_0, \ldots, x_n$, and no more, to determine a degree-n polynomial p_n. (You know the $n = 1$ case well: two points determine a unique line.) If the number of data points were smaller, we could construct infinitely many degree-n interpolating polynomials. Were it larger, there would in general be no degree-n interpolant.

As numerical analysts, we seek answers to the following questions:

- Does such a polynomial $p_n \in \mathcal{P}_n$ exist?
- If so, is it unique?
- Does $p_n \in \mathcal{P}_n$ behave like $f \in \mathrm{C}[a,b]$ at points $x \in [a,b]$ when $x \neq x_j$ for $j = 0, \ldots, n$?
- How can we compute $p_n \in \mathcal{P}_n$ efficiently on a computer?
- If we want to add a new interpolation point x_{n+1}, can we easily adjust p_n to give an interpolating polynomial p_{n+1} of one higher degree?
- How should the interpolation points $\{x_j\}$ be chosen?

Regarding this last question, we should note that, in practice, we are not always able to choose the interpolation points as freely as we might like. For example, our 'continuous function $f \in \mathrm{C}[a,b]$ ' could actually be a discrete list of previously collected experimental data, and we are stuck with the values $\{x_j\}_{j=0}^n$ at which the data was measured. In chapter 3 we will see an alternative approach, appropriate for noisy data, where the overall error $|f(x) - p_n(x)|$ is minimised, without requiring p_n to match f at specific points.

§I.2 Constructing Interpolating Polynomials in Monomial Basis

Every polynomial $p_n(x)$ in the vector space $\mathcal{P}_n$ can be written as a linear combination of the basis functions $1, x, x^2, \ldots, x^n$; these basis functions are called *monomials*.

To construct the polynomial interpolant to f , we merely need to determine the proper values for the coefficients $c_0, c_1, \ldots, c_n$ in eq. (1.1). This reduces to solving the linear system

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{bmatrix}, \tag{1.2}$$

which we denote as $\mathrm{A}\vec{c} = \vec{f}$. Matrices of this form, called Vandermonde matrices, arise in a wide range of applications. Provided all the interpolation points $\{x_j\}$ are distinct, one can show that this matrix is invertible.*

Hence, fundamental properties of linear algebra allow us to confirm that there is **exactly one** degree-n polynomial (whose coefficients are given by $\mathrm{A}^{-1}\vec{f}$), that interpolates f at the given $n+1$ distinct interpolation points.

Theorem 2.4

For any set of $n+1$ different points $\{a \le x_0, x_1, \ldots, x_n \le b\}$ and any function $f \in \mathrm{C}[a,b]$, there exists a unique interpolating polynomial of degree less than or equal to n.

We may also prove uniqueness by the following straightforward argument.

Proof of theorem 4.

[Uniqueness Part.] Suppose that in addition to p_n there is another interpolating polynomial q_n. Then the degree-n polynomial $r_n = p_n - q_n$ has $n+1$ roots x_i for $0 \le i \le n$. From the Fundamental theorem of Algebra, this is possible only if $r_n(x) = 0$. ■

Note: The unique polynomial through $n+1$ points may have degree $< n$. This happens when $c_n = 0$ in the solution eq. (1.2).

One way to solve the above linear system in eq. (1.2) is Gaussian elimination, which we will come back to in section 2.2. However, this is computationally inefficient (taking $\mathcal{O}(n^3)$ operations). In practice, we choose a different basis for p_n. There are two common choices, due to Lagrange and Newton. We will discuss the Lagrange basis in next section.

*In fact, the determinant is given by $\prod_{0 \le i < j \le n} (x_i - x_j)$.

Question 3. Lab

Write a script that accepts an array of ordered pairs (where each x value is unique) and builds a Vandermonde interpolation polynomial.

(a) Use your code to build the Vandermonde interpolation polynomial that interpolates the function $f(x) = cos(2\pi x)$ *with* $(n+1)$ *points that are linearly spaced on the interval* $x \in [0, 2]$. *Repeat the experiment with* $n = 4, n = 6, n = 8$. *Make a plot for each value of n. What do you observe?*

(b) Vandermonde interpolation is relatively easy to conceptualize and code, but there is an inherent problem. Use your Vandemonde interploation code to create a plot where the horizontal axis is the order of the interpolating polynomial and the vertical axis is the ratio of the maximum eigenvalue to the minimum eigenvalue of the Vandemonde matrix $\left|\frac{\lambda_{max}}{\lambda_{min}}\right|$. *What does this plot tell you about Vandermonde interpolation for high-order polynomials? You can use the same model function as part (a).*

§I.3 Constructing interpolants in the Lagrange basis

Observe that the solution for c_i in eq. (1.2) gives us a way to write $p_n(x)$ as a linear combination of the basis $\{x^j\}_{j=0}^n$ of $\mathcal{P}_n$. However, we might want to choose a different basis for P_n if it simplifies the calculation.

Lagrange's method uses a special basis of polynomials $\{\ell_j\}$ such that

$$\ell_j(x_i) = \begin{cases} 0 & \text{if } j \neq i \\ 1 & \text{if } j = i \end{cases} \tag{1.3}$$

Definition 3.5

The Lagrange polynomials $\{\ell_j\}$, for $0 \leq j \leq n$, are defined as

$$\ell_j(x) = \prod_{\substack{i=0 \\ i \neq j}}^{n} \frac{x - x_i}{x_j - x_i}$$

Question 4.

Verify that ℓ_j satisfies eq. (1.3) and $p_n(x_i) = f(x_i)$ for $0 \leq i \leq n$.

Question 5.

Explain why this approach to constructing basis polynomials leads to a diagonal matrix A in the equation $\mathrm{A}\vec{c} = \vec{f}$ for the coefficients.

In other words, the coefficients in this basis are just the function values,

$$p_n(x) = \sum_{j=0}^{n} f(x_j)\ell_j(x) \qquad \text{and} \qquad \vec{c} = \vec{f}$$

Question 6.

Is the Lagrange interpolation polynomial built from a given set of data points same as the Vandermonde interpolation polynomial for the same data?

Note: The Lagrange form of the interpolating polynomial is easy to write down, but expensive to evaluate since all of the ℓ_k must be computed. Moreover, changing any of the nodes means that the ℓ_k must all be recomputed from scratch, and similarly for adding a new node (moving to higher degree).

Question 7. Lab

Compute the quadratic interpolating polynomial to $f(x) = \cos x$ with nodes $\{-\pi/4, 0, \pi/4\}$ using the Lagrange basis analytically.

Then use your favorite mathematical or programming software to plot the interpolating polynomial and the function in the same picture.

§I.4 Interpolation Error Bounds

Whenever we approximate a function $f(x)$ with a (usually) simpler function, we can create the error function associated with the approximation by taking the difference between the two functions. For instance, the Taylor polynomial of degree n at $x = a$

$$T_n(x) := f(a) + f'(a)(x-a) + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n$$

approximates $f(x)$ with error function equal to the Taylor remainder term

$$f(x) - T_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-a)^{n+1},$$

where the unidentified point ξ between x and a can change for different values of x.* Now we'll look more carefully at the error involved in f-interpolation.

I.4.1 Errors in f–interpolation

From theorem 4, we know that there is a unique polynomial $p_n(x)$ of degree less than or equal to n interpolating $f \in C[a,b]$ at the set of points $\{x_0, x_1, \ldots, x_n\}$. The error function associated with this "approximation" of $f(x)$ is then simply

$$E_n(x) = f(x) - p_n(x) \tag{1.4}$$

Example 4.6

You should have found that the quadratic interpolant for $f(x) = \cos x$ with nodes $\{-\pi/4, 0, \pi/4\}$ using the Lagrange basis was $p_2(x) = \frac{16}{\pi^2}\left(\frac{1}{\sqrt{2}} - 1\right)x^2 + 1$. So the error is

$$E_2(x) = \left|\cos x - \frac{16}{\pi^2}\left(\frac{1}{\sqrt{2}} - 1\right)x^2 - 1\right|$$

■ **Question 8.** **Lab**

Plot the error function for example 6. What is $E_n(x_i)$? Is this true in general for any function $f(x)$?

This means that we can repeatedly apply the Factorization Theorem to factor out terms like $(x - x_i)$ from the error function $E_n(x)$, to get the expression

$$E_n(x) = (x-x_0)(x-x_1)(x-x_2)\ldots(x-x_n)r(x) = r(x)\prod_{i=0}^{n}(x-x_i) \tag{1.5}$$

for some "remainder" function $r(x)$. To determine $r(x)$, we are going to do something that may seem a little strange at this point. We'll define a new function $W(x)$ that is the difference between our two expressions (1.4) and (1.5) for the error function $E_n(x)$, with a slight twist. Since we already know the

*This can be proved using the Mean Value Theorem. Look up a proof online or from a calculus textbook if you are interested.

value of $E_n(x_i) = 0$, we choose any other point $\hat{x} \neq x_i$ in $[a,b]$ as the input to the remainder function $r(\hat{x})$, and we leave the other variables x in $E_n(x)$ as they are:

$$W(x) = \overbrace{f(x) - p_n(x)}^{E_n(x) \text{ via } (1.4)} - \overbrace{(x-x_0)(x-x_1)\cdots(x-x_n)\,r(\hat{x})}^{\text{"twisted" } \hat{E}_n(x) \text{ via } (1.5)}. \tag{1.6}$$

■ Question 9.

□

(a) What is $W(x_i)$ equal to?

(b) What is $W(\hat{x})$ equal to?

We conclude that W has at least $n+2$ zeroes in any interval containing the set of points $\{\hat{x}, x_0, x_1, \ldots, x_n\}$. We'll also apply a general fact about "interlacing" zeroes.

Theorem 4.7: Rolle's Theorem

If f is continuous on $[a,b]$ and differentiable on (a,b), with $f(a) = f(b) = 0$, then there exists $\xi \in (a,b)$ with $f'(\xi) = 0$.

If we apply Rolle's theorem to each consecutive pair of zeroes of W from the list $\{\hat{x}, x_0, x_1, \ldots, x_n\}$, we conclude that there are at least $n+1$ zeroes of the derivative function $W'(x)$ in the interval $[a,b]$ (since there is a zero between each consecutive pair of $n+2$ points).

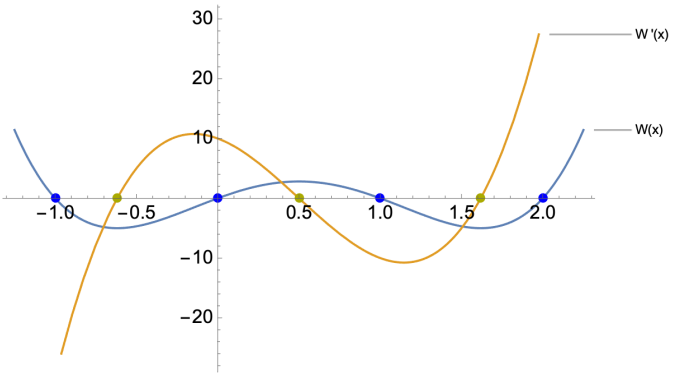


Figure 1.2: Interlacing Zeroes

If we now apply Rolle's theorem again to each consecutive pair of the $n+1$ zeroes of the derivative function $W'(x)$, we conclude that there are n zeroes of the second-derivative function $W''(x)$. We can continue in this manner to deduce that there is at least one zero ξ of the $(n+1)$st derivative function $W^{(n+1)}(x)$.

■ Question 10.

Group

Use $W(x) = f(x) - p_n(x) - (x-x_0)(x-x_1)\cdots(x-x_n)\,r(\hat{x})$ to evaluate $W^{(n+1)}(\xi)$. Then simplify and solve $W^{(n+1)}(\xi) = 0$ to find $r(\hat{x})$.

HINT: only the highest power in the last term matters.

We conclude the following essential result.

Theorem 4.8: Interpolation Error Formula (Cauchy)

Suppose $f \in C^{n+1}[a,b]$ and let $p_n \in \mathcal{P}_n$ denote the f–interpolating polynomial with nodes $\{x_0, x_1, \ldots, x_n\}$ for distinct points $x_j \in [a,b]$. Then for every $x \in [a,b]$ there exists $\xi \in [a,b]$ such that

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

From this formula we can conclude that the worst error over $[a,b]$ can be estimated by the following bound:

$$\max_{x\in[a,b]} |f(x) - p_n(x)| \leq \left(\max_{\xi\in[a,b]} \frac{\left|f^{(n+1)}(\xi)\right|}{(n+1)!} \right) \left(\max_{x\in[a,b]} \prod_{j=0}^{n} \left|x - x_j\right| \right). \tag{1.7}$$

This theorem is the foundation for the convergence theory of piecewise polynomial approximation (section 1.6) and interpolatory quadrature rules for definite integrals (**??**).

Example 4.9

Let's apply the interpolation error bound formula to $f(x) = \sin x$ on $[-5, 5]$. Note that

$$\max_{\xi\in[-5,5]} \left|f^{(n+1)}(\xi)\right| = 1$$

and

$$\max_{x\in[-5,5]} \prod_{j=0}^{n} \left|x - x_j\right| = 10^{n+1}$$

Hence,

$$\max_{x\in[-5,5]} |f(x) - p_n(x)| = \frac{10^{n+1}}{(n+1)!}$$

which does $\to 0$ as $n \to \infty$.

1.4.2 Conclusions

Here are some conclusions we can draw from theorem 8.

- **"Flatter" $f \implies$ better fit:** The smaller the higher-order derivatives of f are on $[a,b]$, the better the interpolation will fit f. As a special case, any polynomial f will eventually (i.e., for n big enough) be fit exactly by an f–interpolating polynomial.
- **Better fit near center:** The error $E_n(x)$ is smaller if x is centered within $[a,b]$, since that makes the product $(x - x_0)(x - x_1)(x - x_2)\ldots(x - x_n)$ relatively smaller.
- **Extrapolation is Dangerous:** Using $p_n(x)$ to approximate $f(x)$ for values of x outside the smallest interval containing the original interpolation points $\{x_0, x_1, \ldots, x_n\}$ causes relatively larger errors, since the product $(x - x_0)(x - x_1)(x - x_2)\ldots(x - x_n)$ is relatively larger.

Note: An interesting feature of the interpolation bound is the polynomial $w(x) = (x - x_0)(x - x_1)(x - x_2)\dots(x - x_n)$. This quantity plays an essential role in approximation theory, and also a closely allied subdiscipline of complex analysis called *potential theory*. Naturally, one might wonder what choice of points $\{x_j\}$ minimizes $|w(x)|$ - we will revisit this question in the next section. For now, we simply note that the points that minimize $|w(x)|$ over $[a, b]$ are called *Chebyshev points*, which are clustered more densely at the ends of the interval $[a, b]$.

Question 11.

Verify that

$$p_1(x) = f(x_0) + \left[\frac{f(x_1) - f(x_0)}{x_1 - x_0}\right](x - x_0)$$

is the unique polynomial of degree ≤ 1 *interpolating* f *at the (distinct) points* x_0 *and* x_1*, and identify what* p_1 *approaches as the interpolating point* x_1 *approaches* x_0*. Explain your thinking.*

Question 12.

Suppose that p_n *is the polynomial of degree* $\leq n$ *that interpolates* $f(x)$ *at the distinct points* $x_0, x_1, \dots, x_n$*. Develop a conjecture about what* p_n *approaches as the interpolating points* $x_1, \dots, x_n$ *all approach* x_0 *simultaneously, and explain your thinking.*

Hint: use the definition of $\mathrm{E}_n(x)$ *and the alternative formula we developed for it in this section.*

§I.5 Lab Assignment I: Runge's Phenomenon and Chebyshev Nodes

You might expect polynomial interpolation to *converge* as $n \to \infty$. Surprisingly, this is not the case if you take equally-spaced nodes x_i. This was shown by Runge in a famous 1901 paper.

Question 13. □

Use your favorite mathematical or programming software to define Runge's function

$$f(x) = \frac{1}{1+25x^2}$$

Question 14. □

Write code/script to create a list of pairs (data-points) $\{(x_i, f(x_i))\}$ for $(n+1)$ evenly spaced points x_i between -1 and 1. For example, for $n = 4$, the points x_i should be 0.5 apart.

- *Your input should be n.*
- *Depending on the programming language you might need separate arrays for x_i values and $f(x_i)$ values for the next part.*

Use `Table` for Mathematica, `numpy.linspace` for Python.

Question 15. □

Define the f–interpolating polynomial $p_n(x)$ corresponding to the lists of nodes.

Check here for Mathematica and here for Python code for Interpolating Polynomial.

Question 16. □

Plot f and the polynomial p_i on the same graph, for $i = 4, 10, 20$, and describe what you observe.

Question 17. □

Use this to predict how the plot of an analogous f–interpolating polynomial p_{100} would compare (to that of f).

Question 18. □

Make a list plot of $\max_{x\in[-1,1]} |f(x) - p_n(x)|$ along the vertical axis vs. n along the horizontal axis for $n = 0, 1, 2, \ldots, 25$. Use this to discuss the nature of convergence of $p_n(x)$ as $n \to \infty$.

What you are seeing here is referred to as ''Runge phenomenon'', named after the German mathematician and physicist Carl David Tolmé Runge who first explored this example.* This is not an issue of numerical instability but rather a fatal flaw associated with uniformly spaced interpolation points.

The problem is (largely) coming from the polynomial

$$w(x) = (x - x_0)(x - x_1)(x - x_2)\ldots(x - x_n)$$

that showed up in the expression of $E_n(x)$ in the last section. Check that the problem is mostly occurring near the ends of the interval, so it would be logical to put more nodes there to reduce the error. A good

*For a proof see http://math.stackexchange.com/questions/775405/.

choice was first proposed by the Russian mathematician Pafnuty Chebyshev (1821-1894). The idea is as follows:

- Draw a semicircle above the closed interval on which you are interpolating ($[-1,1]$ in this case).
- Pick $(n+2)$ equally spaced points, including the end points ($(-1,0)$ and $(1,0)$ in this case), along the semicircle (i.e. same arc length between each point).
- Then choose the midpoints of each of these arcs to get a list of $(n+1)$ points.
- Project the $(n+1)$ points on the semicircle down to the interval. Use these projected points for the interpolation.

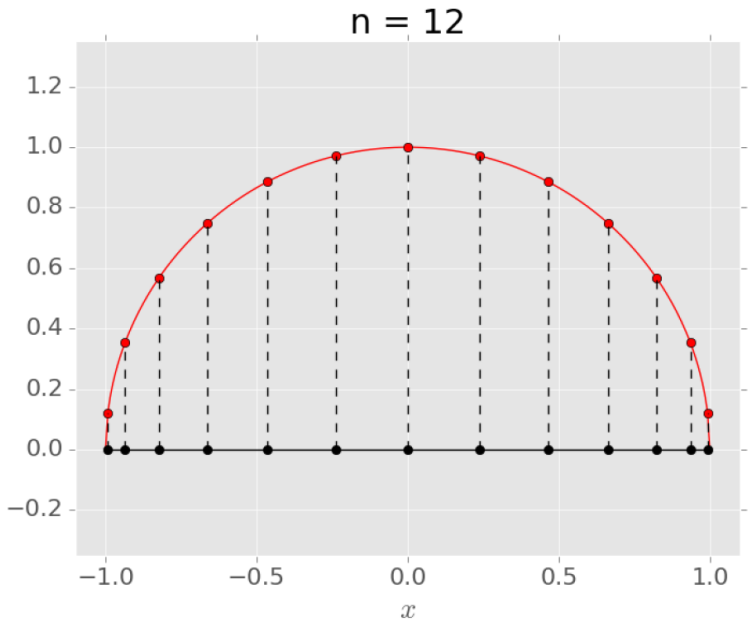


Figure 1.3: Chebyshev Nodes for $n = 12$

Question 19.

Show that the Chebyshev nodes are given by

$$x_j = \cos\frac{(2j+1)\pi}{2(n+1)}, \qquad j = 0,\dots,n. \tag{1.8}$$

HINT: Find the polar coordinates of the points on the circle first. Try an example with $n = 3$.

Although these values might seem arbitrary, they are actually directly related to the Chebyshev polynomial which is defined as

$$T_n(x) := \cos[n\arccos(x)]$$

The fact that T_n is a degree n polynomial can be proved by induction, you can skip that part.*

Question 20.

Prove that the Chebyshev nodes in eq. (1.8) are the roots of the Chebyshev polynomial $T_{n+1}(x)$.

*For example, $T_2(x) = 2x^2 - 1$ because $\cos(2\theta) = 2\cos^2\theta - 1$.

Although we will not prove it here, one can show that the Chebyshev nodes are the best possible choice for minimizing the error.

Theorem 5.10: Chebyshev Interpolation

Let $x_0, x_1, \ldots, x_n \in [-1,1]$ be distinct. Then $\max_{x\in[-1,1]} |w(x)|$ is minimized if

$$w(x) = \frac{1}{2^n} \mathrm{T}_{n+1}(x)$$

Question 21.

Define the f–interpolating polynomial $q_n(x)$ corresponding to the new set of $(n+1)$ Chebyshev interpolation points (note that these are not equally-spaced).

Question 22.

Plot f and the polynomial q_i on the same graph, for $i = 4, 10, 20$, and describe what you observe.

Question 23.

Make a list plot of $\max_{x\in[-1,1]} |f(x) - q_n(x)|$ along the vertical axis vs. n along the horizontal axis for $n = 0, 1, 2, \ldots, 25$. Use this to discuss the nature of convergence for $q_n(x)$ as $n \to \infty$. Compare what you observe to Runge's phenomenon for equally-spaced interpolation points.

§1.6 Spline Interpolation

Last time we saw that Runge's phenomenon can be fixed by using interpolation points (at the zeroes of Chebyshev polynomials) that aren't equally spaced.

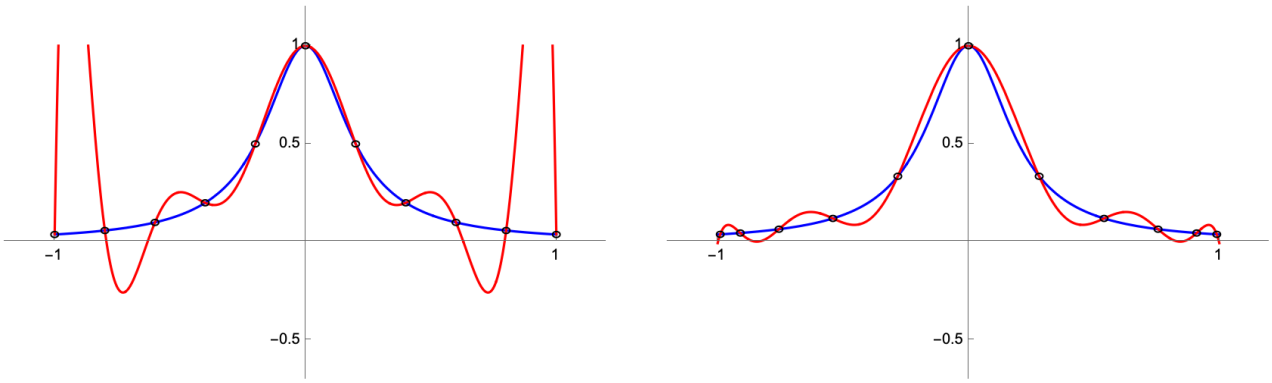


Figure 1.4: Interpolating Polynomial with equally spaced nodes vs. Chebyshev nodes

The interpolating polynomial on the left wiggles too much near the ends of the interpolation interval to match Runge's function (which doesn't wiggle at all there). This "over-wiggling" is fixed on the right by using more interpolation points near the ends of the interval (and fewer interpolation points in the middle where the match is good).

Another way to improve the fit of the interpolating function is the basis for modern computer aided design (CAD). This approach doesn't require special interpolation points or high-order polynomials, but instead uses pieces of simple polynomials between interpolation points.

1.6.1 Splines

Long before computers, ship-builders used thin flexible strips of wood called "splines" to draw curves as they were designing ships. The Romanian mathematician Isaac Jacob Schoenberg is credited with using this same term to describe the piecewise polynomial curves that are now fundamental to computer-aided design. The nice thing about polynomial splines is that they are relatively easy to describe as functions, and yet they can assume an extraordinary range of shapes. Think about trying to come up with a simple function whose graph matches some outline you want. This kind of "inverse problem" is solved very nicely by polynomial splines.

1.6.2 Linear Spline Interpolation

The simplest kind of spline uses linear polynomials to interpolate between each pair of adjacent interpolation points. Figure 1.5 shows images of the linear spline interpolation of Runge's function using 5 and 9 equally-spaced interpolation points. These are clearly better global (i.e. on the entire interval) approximations of Runge's function than the high-order (single) polynomials we saw in Runge's phenomenon.

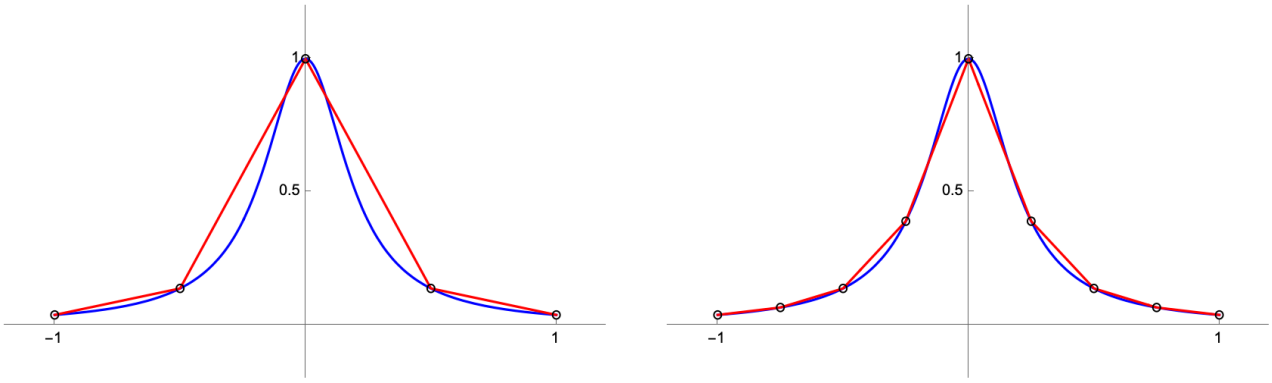


Figure 1.5

Definition 6.11: General Formula

Given $n+1$ data points $\left\{\left(x_j, f_j\right)\right\}_{j=0}^{n}$, we need to construct n linear polynomials $\left\{s_j\right\}_{j=1}^{n}$ such that

$$s_j\left(x_{j-1}\right) = f_{j-1}, \quad \text{and} \quad s_j\left(x_j\right) = f_j$$

for each $j = 1, \ldots, n$. It is simple to write down a formula for these polynomials (compare to question 11)

$$s_j(x) = f_j - \frac{\left(x_j - x\right)}{\left(x_j - x_{j-1}\right)}\left(f_j - f_{j-1}\right).$$

Each s_j is valid on $x \in \left[x_{j-1}, x_j\right]$, and the interpolant $\mathrm{S}(x)$ is defined as $\mathrm{S}(x) = s_j(x)$ for $x \in \left[x_{j-1}, x_j\right]$.

Accuracy of Linear Splines

For a function $\varphi \in \mathrm{C}^1[a,b]$, the L^2 norm of φ is defined as

$$\|\varphi\|_2 = \sqrt{\int_a^b (\varphi(x))^2 \, \mathrm{d}x}$$

We will come back to further discussion regarding L^2 norm in the future. For now, we note the following:

Theorem 6.12

If $\sigma(x)$ is a function that interpolates $f(x)$ at $n+1$ distinct points $x_0 < x_2 < \ldots < x_n$ and if $\|\sigma'\|_2$ is finite on $[x_0, x_n]$ then,

$$\|\mathrm{S}_\mathrm{L}'\|_2 \leq \|\sigma'\|_2$$

where S_L is the linear spline interpolating f at the same points. Thus the linear spline is the "flattest" among all splines.

Question 24. Group

Write a careful argument proving the inequality from theorem 12.

HINT: Start with the difference function $h(x) = \sigma(x) - S_L(x)$ on a generic subinterval $[x_i, x_{i+1}]$. Use the definition of $\|.\|_2$ norm to show that

$$\|\sigma'\|_2^2 = \|h'\|_2^2 + \|S_L'\|_2^2 + 2\int_{x_i}^{x_{i+1}} h'(x)S_L'(x)\,dx.$$

Use the fact that the slope $S_L'(x)$ is constant on $[x_i, x_{i+1}]$ (why?) to simplify the last term.

One problem with a linear splines can be that they are not necessarily differentiable where two different linear pieces meet. Still, these functions are easy to construct and cheap to evaluate, and can be very useful despite their simplicity. Another problem is that we might need a huge number of linear pieces to reproduce a curved shape. To get around these issues, we can use quadratic or cubic splines instead.

1.6.3 Cubic Splines

The idea behind cubic spline interpolation is to use cubic pieces to interpolate adjacent interpolation points. Cubic functions are defined by 4 coefficients whereas linear functions are defined by only 2.

$$s_i(x) = c_3x^3 + c_2x^2 + c_1x + c_0$$

These extra coefficients allow us to choose cubic pieces that match each other better and/or match the function f better. For instance, we can ensure that the derivatives and the second derivatives of adjacent cubic pieces match at the intersection points, so that the resulting spline will be C^2 everywhere.

The cubic spine requirements can be written as:

$$\begin{aligned} &s_j\left(x_{j-1}\right) = f\left(x_{j-1}\right), && j = 1,\ldots,n; \\ &s_j\left(x_j\right) = f\left(x_j\right), && j = 1,\ldots,n; \\ &s_j'\left(x_j\right) = s_{j+1}'\left(x_j\right), && j = 1,\ldots,n-1; \\ &s_j''\left(x_j\right) = s_{j+1}''\left(x_j\right), && j = 1,\ldots,n-1. \end{aligned} \tag{1.9}$$

Question 25. □

Count the total number of requirements (i.e. equations) and the total number of free variables (the coefficients).

What does this tell you about the number of possible solutions to $S(x)$?

So far, we thus have an *underdetermined system*, and there will be infinitely many choices for the function $S(x)$ that satisfy the constraints.

There are several canonical ways to add two extra constraints that uniquely define S :

- **Natural Boundary Conditions:** requires $S''(x_0) = S''(x_n) = 0$;
- **Clamped Boundary Conditions:** specifies exact values for $S'(x_0)$ and $S'(x_n)$, or assumes both are zero;

- **Not-a-knot Boundary Conditions:*** requires S‴ to be continuous at x_1 and x_{n-1}. This forces $s_1 = s_2$ and $s_{n-1} = s_n$.

Natural cubic splines are a popular choice for they can be shown, in a precise sense, to minimize curvature over all the other possible splines. They also model the physical origin of splines, where beams of wood extend straight (i.e., zero second derivative) beyond the first and final 'ducks.'

Question 26.

Suppose S_3 *is the natural cubic spline interpolant to* $\{(x_j, f_j)\}_{j=0}^n$, *and* σ *is any* $C^2[x_0, x_n]$ *function that also interpolates the same data. Then show that*

$$\|S_3''\|_2 \leq \|\sigma''\|_2.$$

Hint: Similar to the proof of theorem 12, first show that

$$\|\sigma''\|_2^2 = \|S_3''\|_2^2 + \|(\sigma'' - S_3'')\|_2^2 + 2\int_{x_0}^{x_n} (\sigma'' - S_3'')S_3'' \, dx.$$

Use integration by parts and the fact that S_3 *is a cubic polynomial on each interval* $[x_i, x_{i+1}]$ *to show that the last term simplifies to*

$$\int_{x_0}^{x_n} (\sigma''(x) - S_3''(x))\, S_3''(x)\, dx = \Big((\sigma'(x_n) - S_3'(x_n))\, S_3''(x_n) \Big) - \Big((\sigma'(x_0) - S_3'(x_0))\, S_3''(x_0) \Big).$$

Use the fact that we are using the natural boundary condition to conclude that above term must be zero.

Question 27. Lab

In this problem, we will compare the Not-a-knot and clamped boundary conditions.

(a) Define the function $f(x) = \sin(20x) + e^{5x/2}$.

(b) Define a `list` *of data points* $\{x_i, f(x_i)\}$ *using the* `Table` *command as* x_i *ranges from* 0 *to* 1 *with* 0.1 *increment intervals.*

(c) Interpolate the data via piecewise-cubic splines, using the Not-a-knot boundary conditions (this is the default).

In `Mathematica`, *you can use* `ResourceFunction["CubicSplineInterpolation"]`. *See this page for usage examples. Alternately, see here for* `Python`.

(d) Next consider clamped boundary conditions. First, analytically compute the derivative at the two endpoints. Then interpolate the data with these values as the boundary conditions.

In Mathematica, use `{{"Clamped",f'[0]},{"Clamped",f'[1]}}` *as an optional argument. In Python, we pass a third argument into cubic spline interpolation package, see* `bc_type`.

(e) The absolute error for an interpolant $p(x)$ *is defined as* $|f(x) - p(x)|$. *Plot the absolute errors for the two boundary condition methods on the same set of axes over the interval* $[0, 1]$. *Which one did a better job of estimating the function near the endpoints? Does this match your expectations? Why or why not?*

*In spline parlance, the interpolation nodes $\{x_j\}_{j=0}^n$ are called *knots*.

Chapter 2 | Linear Equations

This chapter is concerned with techniques for solving a linear system of the form

$$\mathrm{A}\vec{\boldsymbol{x}} = \vec{\boldsymbol{b}} \tag{2.1}$$

where A is an $n \times n$ square matrix with elements a_{ij} , and $\vec{\boldsymbol{x}}, \vec{\boldsymbol{b}}$ are $n \times 1$ vectors. We will focus on solving eq. (2.1) both *accurately* and *efficiently*.

Note: Although this seems like a conceptually easy problem (just use Gaussian elimination!), it is actually a hard one when n gets large. Nowadays, linear systems with $n = 1$ million arise routinely in computational problems. And even for small n there are some potential pitfalls, as we will see.

Let's start by recalling some background facts from Linear Algebra:

- A^{T} is the transpose of A, so $(a^{\mathrm{T}})_{ij} = a_{ji}$.
- A is symmetric if $\mathrm{A} = \mathrm{A}^{\mathrm{T}}$.
- A is non-singular iff there exists a solution $\vec{\boldsymbol{x}} \in \mathbb{R}^n$ for every $\vec{\boldsymbol{b}} \in \mathbb{R}^n$.
- A is non-singular iff $\det(\mathrm{A}) \neq 0$.
- A is non-singular iff there exists a unique inverse A^{-1} such that $\mathrm{A}\mathrm{A}^{-1} = \mathrm{A}^{-1}\mathrm{A} = \mathrm{I}_n$.

It follows that eq. (2.1) has a unique solution iff A is non-singular, given by $\vec{\boldsymbol{x}} = \mathrm{A}^{-1}\vec{\boldsymbol{b}}$. As such many algorithms are based on the idea of rewriting eq. (2.1) in a form where the matrix is easier to invert. Easiest to invert are diagonal matrices, followed by orthogonal matrices (where $\mathrm{A}^{-1} = \mathrm{A}^{\mathrm{T}}$). However, the most common method for solving $\mathrm{A}\vec{\boldsymbol{x}} = \vec{\boldsymbol{b}}$ transforms the system to triangular form.

§2.1 Triangular Systems

If the matrix A is triangular, then $\mathrm{A}\vec{\boldsymbol{x}} = \vec{\boldsymbol{b}}$ is straightforward to solve. Let's take a look at an example for an *upper triangular* matrix U of the form

$$\mathrm{U} = \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & u_{nn} \end{pmatrix}$$

Example 1.13: Solving $\mathrm{U}\vec{\boldsymbol{x}} = \vec{\boldsymbol{b}}$ for $n = 4$

The system is

$$\begin{pmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} \iff \begin{aligned} u_{11}x_1 + u_{12}x_2 + u_{13}x_3 + u_{14}x_4 &= b_1 \\ u_{22}x_2 + u_{23}x_3 + u_{24}x_4 &= b_2, \\ u_{33}x_3 + u_{34}x_4 &= b_3, \\ u_{44}x_4 &= b_4. \end{aligned}$$

We can just solve step-by-step:

$$x_4 = \frac{b_4}{u_4}, \quad x_3 = \frac{b_3 - u_{34}x_4}{u_{33}}, \quad x_2 = \frac{b_2 - u_{23}x_3 - u_{24}x_4}{u_{22}}, \quad x_1 = \frac{b_1 - u_{12}x_2 - u_{13}x_3 - u_{14}x_4}{u_{11}}.$$

Since $\det \mathrm{U} = \prod_{i=1}^{4} u_{ii} \neq 0$ when a solution exists, we know that $u_{11}, u_{22}, u_{33}, u_{44}$ are all non-zero, hence the formula for x_i are well-defined.

In general,

Theorem 1.14

An upper-triangular system $\mathrm{U}\vec{\boldsymbol{x}} = \vec{\boldsymbol{b}}$ *may be solved by backward substitution*

$$x_j = \frac{b_j - \sum_{k=j+1}^{n} u_{jk}x_k}{u_{jj}}, \quad j = n, \ldots, 1$$

Question 28.

Write a script/code that performs backward substitution algorithm for a given upper triangular matrix U *and a* $\vec{\boldsymbol{b}}$*. Your code should work for any n (say, up to* 500*).*

Use the code to solve the system

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

§2.2 Elimination Algorithms

If our matrix A is not triangular, we can try to transform it to triangular form.

Elimination methods for solving linear systems involve three elementary row operations on the matrix equation version of a linear system in order to transform the system to an upper triangular form $U\vec{x} = \vec{y}$. The three operations are:

(a) multiply any row by a constant

(b) add any pair of rows

(c) rearrange the order of the rows

where ''row'' means the entire equation represented by the row of the matrix A and the corresponding components of the vectors $\vec{x}$ and $\vec{b}$. Linear algebra theorem tells us that these operations won't change the solution $\vec{x}$, but they will change the matrix A and the right-hand side $\vec{b}$.

2.2.1 Gaussian Elimination

The first kind of elimination we'll consider has been known for a very long time. For instance, it appears in Chapter 8 of a Chinese math text ''The Nine Chapters on the Mathematical Art'' dating to at least 10 BC, and authored by generations of now anonymous scholars. The same method was later ''invented'' independently and published in 1809 by the German mathematician and astronomer Carl Friedrich Gauss. Since the history of mathematics is essentially ''a fable agreed upon'' (as Napoleon might have said*), Gauss gets his name on this method.

The first stage of Gaussian elimination is to convert the matrix into upper-triangular form (all zeroes below the diagonal) like the one in (2). This process is sometimes called ''forward elimination'' because it ''eliminates'' (turns to zero) entries under the diagonal (i.e., the ''forward'' part of the matrix).

Forward Elimination

Here's an example that doesn't already have an upper-triangular matrix.

$$\begin{bmatrix} 3 & 0 & 1 \\ 0 & -1 & 2 \\ 2 & -2 & 4 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 4 \\ 10 \end{bmatrix}$$

add $-\frac{2}{3}$ times first row to third row

$$\begin{bmatrix} 3 & 0 & 1 \\ 0 & -1 & 2 \\ 0 & -2 & \frac{10}{3} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 4 \\ 6 \end{bmatrix}$$

add -2 times second row to third row

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

The second stage of Gaussian elimination is to use ''back-substitution'' to solve for the variables.

*or not, that's also a fable we agree upon.

Back-substitution

Now the system is upper triangular, and we can follow up with back-substitution similar to last section. The general idea is to work our way up from the bottom row; solving the equations represented by each row as we go.

Since the vector $\vec{x}$ doesn't change during the process of *forward elimination*, we can encode the process on the matrix $[\mathrm{A}|\vec{b}]$ where A is **augmented** by the vector $\vec{b}$. For our example above, the augmented matrix is:

$$[\mathrm{A}|\vec{b}] = \left[\begin{array}{ccc|c} 3 & 0 & 1 & 6 \\ 0 & -1 & 2 & 4 \\ 2 & -2 & 4 & 10 \end{array}\right] \tag{2.2}$$

Algorithm for Gaussian Elimination

Let $\mathrm{A}^{(1)} = \mathrm{A}$ and $\vec{b}^{(1)} = \vec{b}$. Then for each k from 1 to $n-1$, compute a new matrix $\mathrm{A}^{(k+1)}$ and right-hand side $\vec{b}^{(k+1)}$ by the following procedure:

Step 1. Define the row multipliers

$$m_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}, \quad i = k+1, \ldots, n.$$

Step 2. Use these to remove the unknown x_k from equations $k+1$ to n, leaving

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - m_{ik} a_{kj}^{(k)}, \quad b_i^{(k+1)} = b_i^{(k)} - m_{ik} b_k^{(k)}, \quad i = k+1, \ldots, n, \quad j = k, \ldots, n.$$

The final matrix $\mathrm{A}^{(n)} = \mathrm{U}$ will then be upper triangular.

Step 3. The upper-triangular system $\mathrm{A}^{(n)}\vec{x} = \vec{b}^{(n)}$ may be solved by *backward substitution*

$$x_j = \frac{b_j^{(n)} - \sum_{k=j+1}^{n} a_{jk}^{(n)} x_k}{a_{jj}^{(n)}}, \quad j = n, n-1, n-2, \ldots, 1$$

Note that this procedure will work provided $a_{kk}^{(k)} \neq 0$ for every k. We will worry about this later (section 2.3). This will involve the 'swapping' elementary row operation.

Question 29. □

Take a look ahead to question 33 and start thinking about how to write a computer program or script that does Gaussian elimination for a linear systems of any size (say, up to $n = 500$) .

2.2.2 Gauss-Jordan elimination

This method gets its name because it follows Gaussian elimination through the forward elimination stage, but then - instead of back-substitution - uses elementary row operations to complete ''reverse elimination''

until the matrix is transformed into zeroes above the diagonal too. As a final step, Gauss-Jordan elimination divides through to get ones along the diagonal, so the solution is simply the resulting right-side target vector. A German geodesist (studying measurements of the earth) named Wilhelm Jordan came up with this twist in 1887 (the same year Clasen published the same idea!), and it has the nice property that the inverse matrix A^{-1} can be explicitly computed via the process by applying exactly the same operations on the matrix [A|I] where A is augmented by the $n \times n$ identity matrix I. *

We'll show how this works by revisiting the upper triangular matrix (2.2) we already generated by forward elimination in the preceding section.

Reverse Elimination

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

add $\frac{3}{2}$ times the last row to the first

$$\begin{bmatrix} 3 & 0 & 0 \\ 0 & -1 & 2 \\ 0 & 0 & -\frac{2}{3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \\ -2 \end{bmatrix}$$

add 3 times the last row to the second

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

Divide to get ones on the diagonal

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

divide first row by 3 , second by –1, and third by $-\frac{2}{3}$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

From here, it is trivial to generate the solution $x_1 = 1, x_2 = 2$, and $x_3 = 3$.

2.2.3 Comparing Computational Effort

Since elimination methods take a finite number of steps to solve a system of linear equations, they are often compared by counting the individual operations needed to solve a general $n \times n$ system (n equations for n unknowns). In general, the amount of time required to perform a multiplication or division on a computer is approximately the same and is considerably greater than that required to perform an addition or subtraction. So we have a couple of different ways of computing the number of operations.

*There is another description of Gauss-Jordan elimination where reverse elimination is performed on each column at the same time as forward elimination, however this is less efficient than the version we're using here.

(a) total individual operations (multiplications/divisions and additions/subtractions of numbers).

(b) total individual multiplications/divisions only

(c) total individual additions/subtractions only

Note: We do not count additions/subtractions or multiplications/divisions of zeroes created along the steps of the process.

Example 2.15: Total number of operations required for backward substitution.

Consider each x_j. We have

$$\begin{aligned}
&j = n \to 1 \text{ division} \\
&j = n-1 \to 1 \text{ division } + [1 \text{ subtraction } + 1 \text{ multiplication }] \\
&j = n-2 \to 1 \text{ division } + 2 \times [1 \text{ subtraction } + 1 \text{ multiplication }] \\
&\vdots \\
&j = 1 \to 1 \text{ division } + (n-1) \times [1 \text{ subtraction } + 1 \text{ multiplication }]
\end{aligned}$$

So the total number of operations required is

$$\sum_{j=1}^{n} (1 + 2(n-j)) = n^2$$

So solving a triangular system by backward substitution takes n^2 operations.

Note:

- We say that the computational complexity of the algorithm is n^2.
- In practice, this is only a rough estimate of the computational cost, because reading from and writing to the computer's memory also take time. This can be estimated given a "memory model", but this depends on the particular computer.

Question 30. Group

Evaluate how many additions/subtractions are necessary to perform Gauss-Jordan elimination on a general $n \times n$ linear system. You should break your work into two steps:

(a) Find the number of additions/subtractions for forward elimination.

(b) Find the number of additions/subtractions for reverse elimination.

§2.3 Pivoting

Gaussian elimination will fail if we ever hit a zero on the diagonal. But this does not mean that the matrix A is singular. Consider the following example,

Example 3.16

The system

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

obviously has solution $x_1 = x_2 = x_3 = 1$ (the matrix has determinant -6). But Gaussian elimination will fail because $a_{11}^{(1)} = 0$, so we cannot calculate m_{21} and m_{31}. However, we could avoid the problem by changing the order of the equations to get the equivalent system

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

Now there is no problem with Gaussian elimination (actually the matrix is already upper triangular). Alternatively, we could have rescued Gaussian elimination by swapping columns:

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

Swapping rows or columns is called pivoting. It is needed if the ''pivot'' element is zero, as in the above example. But it is also used to reduce rounding error.

■ Question 31. □

Apply Gaussian elimination to the system

$$0.003x_1 + 59.14x_2 = 59.17$$
$$5.291x_1 - 6.130x_2 = 46.78$$

using the program you wrote for question 33.

If you used Mathematica to write your program, change the first equation to

$$0.003 \times 10^{-14}x_1 + 59.14x_2 = 59.14 + 0.003 \times 10^{-13}.$$

This is because the default precision in Mathematica is a bit 'too good'.

Above example shows how difficulties can arise when the pivot element $a_{kk}^{(k)}$ is small relative to the entries $a_{ij}^{(k)}$, for $k \le i \le n$ and $k \le j \le n$. To avoid this problem, the simplest strategy is to select an element in the same column that is below the diagonal and has the largest absolute value; specifically, we determine the smallest $p \ge k$ such that

$$\left|a_{pk}^{(k)}\right| = \max_{k \le i \le n} \left|a_{ik}^{(k)}\right|$$

and exchange the kth and the pth row. This method, known as **partial pivoting**, dramatically improves the stability of Gaussian elimination.

Question 32.

Apply Gaussian elimination with partial pivoting to the system

$$0.003x_1 + 59.14x_2 = 59.17$$
$$5.291x_1 - 6.130x_2 = 46.78$$

using the program you wrote for question 33.

If you used Mathematica to write your program, change the first equation to

$$0.003 \times 10^{-14}x_1 + 59.14x_2 = 59.14 + 0.003 \times 10^{-13}.$$

It should give the correct answer this time.

Note:

- Gaussian elimination without pivoting is unstable: rounding errors can accumulate.
- The ultimate accuracy is obtained by full pivoting, where both the rows and columns are swapped to bring the largest element possible to the diagonal.
- If it is not possible to rearrange the columns or rows to remove a zero from position $a_{kk}^{(k)}$ then A is singular.

§2.4 Lab Assignment 2: Elimination Methods

Question 33.

Write a function or script that will solve a linear systems of any size (say, up to $n = 500$) by Gaussian elimination.

- *The program should have an input parameter called* `method` *or something similar which has three possible values. Include (perhaps by an* `if - else` *structure) the ability to switch between the following three possibilities.*
  - *If method is 0, the algorithm should not be using pivoting. Make sure to catch any division by zero in such situations and display an error.*
  - *If method is 1, pivoting is done only when the pivot element is zero.*
  - *If method is 2, partial pivoting is performed at each step.*

Question 34.

Solve the augmented matrix system $A\vec{x} = \vec{b}$ below analytically (by hand).

$$A = \begin{pmatrix} \varepsilon & 1 & 2 \\ 1 & 1 & 0 \\ 2\varepsilon & 2 & -3 \end{pmatrix}, \vec{b} = \begin{pmatrix} 5 \\ 2 \\ -4 \end{pmatrix}$$

Question 35.

Use your code to solve the matrix system from question 34 for $\varepsilon = 0.001$, with and without partial pivoting. Calculate the error vector in your computed answer, by subtracting the computed solution (in question 33) from the exact solution obtained in question 34. To get a scalar error value from the error vector, we can compute its 2-norm, the Euclidean distance from the origin in n-dimensional space. This can be done via `numpy.linalg.norm` *in Python or* `norm` *in Octave.*

Question 36.

Regardless of the methodology used, some systems of equations are very difficult to solve with a high degree of accuracy, due to properties of the coefficient matrix. One example is a Hilbert matrix, whose entries are given by the formula

$$H_{ij} = \frac{1}{i + j - 1}.$$

(a) Find the (scalar) error associated with solving a system involving the 6×6 Hilbert matrix. For simplicity, set up your system so that the exact solution is $[1\,2\,3\,4\,5\,6]^T$, computing the associated vector $\vec{b}$.

(b) Does partial pivoting help in this example?

Question 37.

Hilbert matrices are known to be difficult to work with in numerical linear algebra settings, in that they induce a high degree of error in floating-point computations. The condition number is one measure of how "bad" a matrix is in this aspect. For a square matrix A, we can define the condition number $\kappa_*(A)$ as $\|A\|_*\|A^{-1}\|_*$, for some matrix norm $\|\cdot\|_*$.

(a) *Compute the condition number κ of the coefficient matrix from question 34 and also for the Hilbert matrix.*

See here for Python code, here for Octave code, and here for Mathematica. The default is the l^2-norm (see theorem 26).

(b) *Comment briefly on the relationship between the condition number and the observed levels of error in each case.*

§2.5 Norms of Vectors and Matrices

2.5.1 Vector Norm

To measure the error when the solution is a vector, as opposed to a scalar, we usually summarize the error in a single number called a norm.

Definition 5.17

A vector norm on $\mathbb{R}^n$ is a real-valued function that satisfies

$$\|\vec{x}+\vec{y}\| \le \|\vec{x}\|+\|\vec{y}\| \quad \text{for every } \vec{x},\vec{y}\in\mathbb{R}^n, \tag{N1}$$

$$\|\alpha\vec{x}\| = |\alpha|\|\vec{x}\| \quad \text{for every } \vec{x}\in\mathbb{R}^n \text{ and every } \alpha\in\mathbb{R}, \tag{N2}$$

$$\|\vec{x}\| \ge 0 \quad \text{for every } \vec{x}\in\mathbb{R}^n \text{ and } \|\vec{x}\|=0 \Longrightarrow \vec{x}=\vec{0}. \tag{N3}$$

Property (N1) is called *the triangle inequality.*

Example 5.18

Let $\vec{x} = \langle x_1, x_2, \dots, x_n\rangle$. There are three common examples of vector norm:

(a) The ℓ_2-norm

$$\|\vec{x}\|_2 := \sqrt{\sum_{k=1}^{n} x_k^2} = \sqrt{\vec{x}^{\mathrm{T}}\vec{x}}.$$

This is just the usual Euclidean length of $\vec{x}$.

(b) The ℓ_1-norm

$$\|\vec{x}\|_1 := \sum_{k=1}^{n} |x_k|$$

This is sometimes known as the taxicab or Manhattan norm, because it corresponds to the distance that a taxi has to drive on a rectangular grid of streets to get to $\vec{x}\in\mathbb{R}^2$ from the origin.

(c) The $\ell_\infty - norm$

$$\|x\|_\infty := \max_{k=1,\dots,n} |x_k|$$

This is sometimes known as the maximum norm.

The norms in the example above are all special cases of the ℓ_p-norm,

$$\|\vec{x}\|_p = \left(\sum_{k=1}^{n} |x_k|^p\right)^{1/p}$$

which is a norm for any real number $p \ge 1$. Increasing p means that more and more emphasis is given to the maximum element $|x_k|$.

Question 38.

Consider the vectors $\vec{a} = (1,-2,3)^T$, $\vec{b} = (2,0,-1)^T$, *and* $\vec{c} = (0,1,4)^T$. *Evaluate their* ℓ_1-, ℓ_2-, *and* $\ell_\infty-$*norms.*

You should be able to observe that for the same vector $\vec{x}$, the norms satisfy the ordering $\|\vec{x}\|_1 \geq \|\vec{x}\|_2 \geq \|\vec{x}\|_\infty$, but two vectors may be ordered differently by different norms.

Question 39.

Sketch the "unit circles" $S_p = \left\{\vec{x} \in \mathbb{R}^2 : \|\vec{x}\|_p = 1\right\}$ *for* $p = 1, 2, \infty$.

2.5.2 Matrix Norm

We also use norms to measure the "size" of matrices. Since the set $\mathbb{R}^{n\times n}$ of $n \times n$ matrices with real entries is a vector space, we could just use a vector norm on this space. But usually we add an additional axiom.

Definition 5.19

A matrix norm is a real-valued function $\|\cdot\|$ on $\mathbb{R}^{n\times n}$ that satisfies:

$$\|A+B\| \leq \|A\| + \|B\| \quad \text{for every } A, B \in \mathbb{R}^{n\times n} \tag{M1}$$
$$\|\alpha A\| = |\alpha|\|A\| \quad \text{for every } A \in \mathbb{R}^{n\times n} \text{ and every } \alpha \in \mathbb{R}, \tag{M2}$$
$$\|A\| \geq 0 \quad \text{for every } A \in \mathbb{R}^{n\times n} \text{ and } \|A\| = 0 \Rightarrow A = 0, \tag{M3}$$
$$\|AB\| \leq \|A\|\|B\| \quad \text{for every } A, B \in \mathbb{R}^{n\times n}. \tag{M4}$$

The new axiom (M4) is called **consistency**. We usually want this additional axiom because matrices are more than just vectors. Some books call this a submultiplicative norm and define a "matrix norm" to satisfy just (M1), (M2), (M3), perhaps because (M4) only works for square matrices.

Note: If we treat a matrix as a big vector with n^2 components, then the ℓ_2-norm is called the Frobenius norm of the matrix:

$$\|A\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}^2}$$

This norm is rarely used in numerical analysis because it is not induced by any vector norm (as we are about to define).

The most important matrix norms are so-called induced or *operator norms*. Remember that A is a linear map on $\mathbb{R}^n$, meaning that it maps every vector to another vector. So we can measure the size of A by how much it can stretch vectors with respect to a given vector norm.

Definition 5.20

if $\|\cdot\|_p$ is a vector norm, then the induced or operator norm is defined as

$$\|A\|_p := \sup_{\vec{x}\neq 0} \frac{\|A\vec{x}\|_p}{\|\vec{x}\|_p} = \max_{\|\vec{x}\|_p=1} \|A\vec{x}\|_p.$$

Question 40.

Use property (M2) to prove that the two definitions are equivalent.

Example 5.21

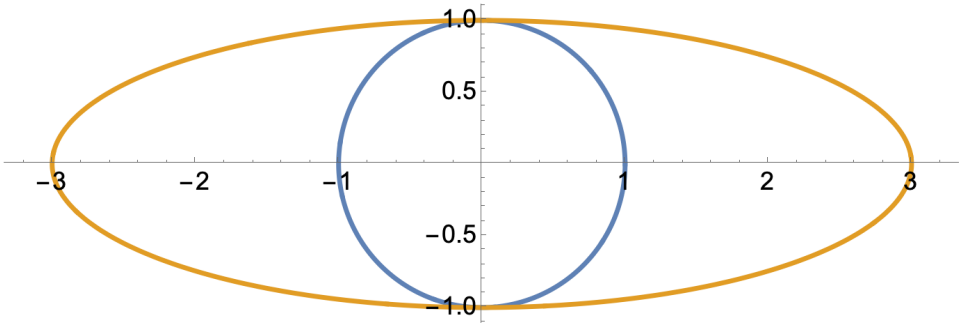
Let

$$A = \begin{pmatrix} 0 & 3 \\ 1 & 0 \end{pmatrix}.$$

In the ℓ_2-norm, a unit vector in $\mathbb{R}^2$ has the form $\vec{x} = (\cos\theta, \sin\theta)^T$, so the image of the unit circle is

$$A\vec{x} = \begin{pmatrix} 3\sin\theta \\ \cos\theta \end{pmatrix}.$$

This is illustrated below:



The induced matrix norm is the maximum stretching of this unit circle, which is

$$\|A\|_2 = \max_{\|\vec{x}\|_2=1} \|A\vec{x}\|_2 = \max_{\theta} \left(9\sin^2\theta + \cos^2\theta\right)^{1/2} = \max_{\theta} \left(1 + 8\sin^2\theta\right)^{1/2} = 3.$$

Theorem 5.22

The induced norm corresponding to any vector norm is a matrix norm.

Properties (M1)-(M3) for an induced norm follow from the fact that the vector norm satisfies (N1)-(N3).

Question 41.

Prove (M4) for an induced norm using the following steps:

(a) Show that $\|A\vec{y}\| \leq \|A\|\|\vec{y}\|$ *for any matrix* $A \in \mathbb{R}^{n\times n}$ *and any vector* $\vec{y} \in \mathbb{R}^n$.

(b) Taking $\vec{y} = \mathrm{B}\vec{x}$ *for some* $\vec{x}$ *with* $\|\vec{x}\| = 1$*, show that*

$$\|\mathrm{AB}\| \leq \|\mathrm{A}\|\|\mathrm{B}\|.$$

It is cumbersome to compute the induced norms from their definition, but fortunately there are some very useful alternative formulae.

Theorem 5.23

The matrix norms induced by the ℓ_1*-norm and* ℓ_∞*-norm satisfy*

$$\|\mathrm{A}\|_1 = \max_{1\leq j\leq n} \sum_{i=1}^{n} |a_{ij}|, \quad \textit{(maximum column sum)}$$

$$\|\mathrm{A}\|_\infty = \max_{1\leq i\leq n} \sum_{j=1}^{n} |a_{ij}|. \quad \textit{(maximum row sum)}$$

Proof of theorem 23.

We will prove the result for the ℓ_1-norm first. Starting from the definition of the ℓ_1 vector norm, we have

$$\|\mathrm{A}\vec{x}\|_1 = \sum_{i=1}^{n} \left| \sum_{j=1}^{n} a_{ij}x_j \right| \leq \sum_{i=1}^{n} \sum_{j=1}^{n} |a_{ij}||x_j| = \sum_{j=1}^{n} |x_j| \sum_{i=1}^{n} |a_{ij}|.$$

If we let

$$c = \max_{j=1,\ldots,n} \sum_{i=1}^{n} |a_{ij}|$$

then

$$\|\mathrm{A}\vec{x}\|_1 \leq c\|\vec{x}\|_1 \implies \|\mathrm{A}\|_1 \leq c. \tag{2.3}$$

Now let m be the column where the maximum sum is attained. If we choose y to be the vector with components $y_k = \delta_{km}$, then we have $\|\mathrm{A}\vec{y}\|_1 = c$. Since $\|\vec{y}\|_1 = 1$, we must have that

$$\max_{\|x\|_1=1} \|\mathrm{A}x\|_1 \geq \|\mathrm{A}y\|_1 = c \implies \|\mathrm{A}\|_1 \geq c. \tag{2.4}$$

The only way to satisfy both eq. (2.3) and eq. (2.4) is if $\|\mathrm{A}\|_1 = c$. ■

■ Question 42. □

Prove theorem 23 for the ℓ_∞*-norm.*

■ Question 43. □

Find $\|\mathrm{A}\|_1$ *and* $\|\mathrm{A}\|_\infty$ *for the matrix*

$$\mathrm{A} = \begin{pmatrix} -7 & 3 & -1 \\ 2 & 4 & 5 \\ -4 & 6 & 0 \end{pmatrix}$$

2.5.3 Using Matrix Norm to measure effect of small perturbations

In practice, often times the output vector $\vec{b}$ has small perturbations due to computation errors. In this section, we wish to find how we can use matrix norms to measure the accuracy of the solution $\vec{x}$ compared to the accuracy of $\vec{b}$.

Suppose a system of equation $A\vec{x} = \vec{b}$ and a second system of equations obtained by altering the right-hand side:

$$A(\vec{x} + \delta\vec{x}) = \vec{b} + \delta\vec{b}$$

We think of $\delta\vec{b}$ as being the error in $\vec{b}$ and $\delta\vec{x}$ as being the resulting error in $\vec{x}$, although we need not make any assumptions that the errors are small.

Question 44. □

Suppose A *is nonsingular. Show that*

$$\frac{\|\delta\vec{x}\|}{\|\vec{x}\|} \leq \|A\| \cdot \|A^{-1}\| \frac{\|\delta\vec{b}\|}{\|\vec{b}\|}$$

for any matrix norm $\|.\|$.

Definition 5.24

The condition number of a nonsingular matrix A relative to a matrix norm $\|.\|$ is

$$\kappa(A) = \|A\| \cdot \|A^{-1}\|$$

We conclude that the relative error in the solution is small only if $\kappa(A)$ is small; which is what you should have observed in question 37. When $\kappa(A)$ is close to 1, we say that A is well-conditioned, and if $\kappa(A)$ is significantly greater than 1, we say that A is ill-conditioned,

Question 45. □

Consider the n-by-n upper triangular matrix A *with elements*

$$a_{ij} = \begin{cases} -1, & i < j \\ 1, & i = j \\ 0, & i > j \end{cases}$$

Show that the condition number κ_1 *relative to the* ℓ_1 *norm is equal to* $n2^{n-1}$.

2.5.4 Eigenvalues and Spectral Radius

What about the matrix norm induced by the ℓ_2-norm? This turns out to be related to the eigenvalues of A. Recall that $\lambda \in \mathbb{C}$ is an eigenvalue of A with associated eigenvector $\vec{u}$ if $A\vec{u} = \lambda\vec{u}$.

Definition 5.25

The *spectral radius* $\rho(A)$ of A is the maximum $|\lambda|$ over all eigenvalues λ of A.

Theorem 5.26

The matrix norm induced by the ℓ_2-norm satisfies

$$\|A\|_2 = \sqrt{\rho(A^T A)}.$$

As a result the ℓ_2-norm is sometimes known as the spectral norm.

Example 5.27

For our matrix

$$A = \begin{pmatrix} 0 & 3 \\ 1 & 0 \end{pmatrix},$$

we have

$$A^T A = \begin{pmatrix} 0 & 1 \\ 3 & 0 \end{pmatrix}\begin{pmatrix} 0 & 3 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 9 \end{pmatrix}.$$

We see that the eigenvalues of $A^T A$ are $\lambda = 1, 9$, so $\|A\|_2 = \sqrt{9} = 3$ (as we calculated earlier).

Theorem 5.28

If A is an $n \times n$ matrix, then $\rho(A) \leq \|A\|$ for any induced norm $\|.\|$.

Proof of theorem 28.

If $A\vec{x} = \lambda\vec{x}$ and $\|\vec{x}\| = 1$, then $|\lambda| = \|A\vec{x}\| \leq \|A\|$. ■

We include the proof of theorem 26 for the sake of completion, but you can skip it for the purpose of the current course.

Proof of theorem 26.

We want to show that

$$\max_{\|x\|_2 = 1} \|Ax\|_2 = \max\left\{\sqrt{|\lambda|} : \lambda \text{ eigenvalue of } A^T A\right\}.$$

For A real, $A^T A$ is symmetric, so has real eigenvalues $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ with corresponding orthonormal eigenvectors $\vec{u}_1, \ldots, \vec{u}_n$ in $\mathbb{R}^n$. (Orthonormal means that $\vec{u}_j^T \vec{u}_k = \delta_{jk}$.) Note also that all of the eigenvalues are non-negative, since

$$A^T A\vec{u}_1 = \lambda_1 \vec{u}_1 \quad \Longrightarrow \lambda_1 = \frac{\vec{u}_1^T A^T A \vec{u}_1}{\vec{u}_1^T \vec{u}_1} = \frac{\|A\vec{u}_1\|_2^2}{\|\vec{u}_1\|_2^2} \geq 0.$$

So we want to show that $\|A\|_2 = \sqrt{\lambda_n}$. The eigenvectors form a basis, so every vector $\vec{x} \in \mathbb{R}^n$ can be

expressed as a linear combination $\vec{\boldsymbol{x}} = \sum_{k=1}^{n} \alpha_k \vec{\boldsymbol{u}}_k$. Therefore

$$\|\mathrm{A}\vec{\boldsymbol{x}}\|_2^2 = \vec{\boldsymbol{x}}^{\mathrm{T}}\mathrm{A}^{\mathrm{T}}\mathrm{A}\vec{\boldsymbol{x}} = \vec{\boldsymbol{x}}^{\mathrm{T}} \sum_{k=1}^{n} \alpha_k \lambda_k \vec{\boldsymbol{u}}_k = \sum_{j=1}^{n} \alpha_j \vec{\boldsymbol{u}}_j^{\mathrm{T}} \sum_{k=1}^{n} \alpha_k \lambda_k \vec{\boldsymbol{u}}_k = \sum_{k=1}^{n} \alpha_k^2 \lambda_k,$$

where the last step uses orthonormality of the $\vec{\boldsymbol{u}}_k$. It follows that

$$\|\mathrm{A}\vec{\boldsymbol{x}}\|_2^2 \leq \lambda_n \sum_{k=1}^{n} \alpha_k^2.$$

But if $\|x\|_2 = 1$, then $\|x\|_2^2 = \sum_{k=1}^{n} \alpha_k^2 = 1$, so $\|\mathrm{A}x\|_2^2 \leq \lambda_n$. To show that the maximum of $\|\mathrm{A}x\|_2^2$ is equal to λ_n, we can choose x to be the corresponding eigenvector $x = u_n$. In that case, $\alpha_1 = \ldots = \alpha_{n-1} = 0$ and $\alpha_n = 1$, so $\|\mathrm{A}\vec{\boldsymbol{x}}\|_2^2 = \lambda_n$. ■

§2.6 Iterative Methods

So far, we have studied elimination methods for solving linear systems $A\vec{x} = \vec{b}$, which converted the systems via elementary row operations until the solutions were easy to deduce. These solution methods are familiar from a course on Linear Algebra, but they can be very computationally expensive. In particular, some zero entries in the original matrix (which don't require storage or processing) may be replaced by non-zero entries as the method proceeds; necessitating more storage and processing at subsequent steps. This time, we'll consider a new kind of solution method for linear systems that avoids this problem by maintaining the original matrix structure.

The basic idea behind *iteration methods* for solving linear systems is to start with an initial approximation $\vec{x}^{(0)}$ to the solution $\vec{x}$ and generate a sequence of vectors

$$\left\{\vec{x}^{(0)}, \vec{x}^{(1)}, \vec{x}^{(2)} \ldots, \vec{x}^{(k)}, \ldots\right\}$$

that converges to $\vec{x}$. To generate the sequence, we'll use some iteration vector-function $g(\vec{x}) = T\vec{x} + \vec{c}$ where T is a (square) matrix and $\vec{c}$ is a vector (of the same dimension as $\vec{x}$) satisfying

$$\vec{x} = g(\vec{x}) \quad \Longleftrightarrow \quad A\vec{x} = \vec{b} \tag{2.5}$$

The iterative methods for linear system consist of starting with an initial approximation $\vec{x}^{(0)}$ and repeatedly calculating

$$\vec{x}^{(k+1)} = T\vec{x}^{(k)} + \vec{c}$$

until there is ''sufficiently small change'' between the input and the output vectors. For example, we can require that

$$\frac{\|\vec{x}^{(k+1)} - \vec{x}^{(k)}\|_*}{\|\vec{x}^{(k+1)}\|_*} \leq \varepsilon$$

for some prescribed error tolerance ε and some convenient norm $\|.\|_*$, usually the l_∞ norm.

Note: At this point, you might ask why should the sequence $\vec{x}^{(0)}, \vec{x}^{(1)}, \vec{x}^{(2)} \ldots, \vec{x}^{(k)}, \ldots$ converge? In fact, it may not always converge. We will come back to this question at the end of this section.

■ Question 46. □

Let D *be a diagonal matrix. Show that if*

$$T = D^{-1}(D - A) \quad \textit{and} \quad \vec{c} = D^{-1}\vec{b}, \tag{2.6}$$

then condition 2.5 is satisfied.

2.6.1 The Jacobi Method

The following version of an iteration method was named after the German mathematician Carl Gustav Jacob Jacobi who proposed a (more complicated) variant of it in the mid-1800's. The Jacobi method uses the formula from eq. (2.6) with the matrix D consisting of only the diagonal entries of the matrix A in the linear system:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \implies D = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix} \tag{2.7}$$

Note: Diagonal matrices are very easy to invert (as long as the diagonal entries a_{ii} of A are nonzero), the inverse D^{-1} in this case is the diagonal matrix whose entries are the inverses a_{ii}^{-1} of their counterparts in D.

2.6.2 Gauss-Seidel Method

The next iteration method we'll consider is usually credited to Gauss and another German mathematician, Philipp Ludwig von Seidel (though it is also sometimes credited to Liebmann, or simply called the method of "successive displacement"). The Gauss-Seidel method uses

$$T = -(L+D)^{-1}U \quad \text{and} \quad \vec{c} = (L+D)^{-1}\vec{b}, \tag{2.8}$$

in terms of the same matrix D as in the Jacobi method, and the upper (U) and lower (L) triangular parts of A left over when the diagonal D is removed:

$$U = \begin{bmatrix} 0 & a_{1,2} & \cdots & a_{1,n} \\ 0 & 0 & \cdots & a_{2,n} \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & a_{n-1,n} \\ 0 & 0 & \cdots & 0 \end{bmatrix} \quad L = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ a_{2,1} & 0 & \cdots & 0 \\ \cdot & \cdot & \cdots & \cdot \\ a_{n-1,1} & \cdot & \cdots & \cdot \\ a_{n,1} & \cdots & a_{n,n-1} & 0 \end{bmatrix} \tag{2.9}$$

■ **Question 47.** □

Show that eq. (2.8) gives an iteration vector-function satisfying condition 2.5.

2.6.3 Convergence Criteria

To study the convergence of general iteration techniques, we need to analyze the formula

$$\vec{x}^{(k+1)} = T\vec{x}^{(k)} + \vec{c}$$

where $\vec{x}^{(0)}$ is arbitrary. We will mention two necessary and sufficient conditions for the convergence, without proof.

Theorem 6.29

For any $\vec{\boldsymbol{x}}^{(0)} \in \mathbb{R}^n$, *the sequence* $\left\{\vec{\boldsymbol{x}}^{(k)}\right\}_{k=0}^{\infty}$ *defined by*

$$\vec{\boldsymbol{x}}^{(k)} = \mathrm{T}\vec{\boldsymbol{x}}^{(k-1)} + \mathbf{c}, \quad \text{for each } k \geq 1,$$

converges to the unique solution of $\vec{\boldsymbol{x}} = \mathrm{T}\vec{\boldsymbol{x}} + \mathbf{c}$ *if and only if* $\rho(\mathrm{T}) < 1$.

Definition 6.30

A square matrix A is said to be *diagonally dominant* if

$$|a_{ii}| \geq \sum_{j \neq i} |a_{ij}| \quad \text{for all } i$$

where a_{ij} denotes the entry in the ith row and jth column.

If the inequality is strict, we say A is *strictly diagonally dominant*.

One can show that a strictly diagonally dominant is nonsingular. Gaussian elimination algorithm can be performed for such a matrix without any pivoting, and the computations remain stable with respect to the growth of round-off errors. The following theorem gives a sufficient (but not necessary) condition for convergence of Jacobi and Gauss-Seidel algorithms.

Theorem 6.31

If A *is strictly diagonally dominant, then for any choice of* $\vec{\boldsymbol{x}}^{(0)}$, *both the Jacobi and the Gauss-Seidel methods give sequences* $\left\{\vec{\boldsymbol{x}}^{(k)}\right\}_{k=0}^{\infty}$ *that converge to the unique solution of* $\mathrm{A}\vec{\boldsymbol{x}} = \vec{\boldsymbol{b}}$.

Question 48.

Let $\vec{\boldsymbol{x}}$ *be a fixed point of* g. *Show that* $\vec{\boldsymbol{x}} - \vec{\boldsymbol{x}}^{(k)} = \mathrm{T}^k\left(\vec{\boldsymbol{x}} - \vec{\boldsymbol{x}}^{(0)}\right)$. *Conclude that* $\|\mathrm{T}\| < 1$ *is a sufficient condition for* $\|\vec{\boldsymbol{x}} - \vec{\boldsymbol{x}}^{(k)}\| \to 0$.

□

§2.7 Lab Assignment 3: Stationary Iterative Methods

Question 49.

Write a script or function that approximates the solution to the system $A\vec{x} = \vec{b}$ *using the Jacobi Method. The inputs should be an* $n \times n$ *matrix* A*, an* n*-dimensional vector* $\vec{b}$*, a starting vector* $\vec{x}_0$*, an error tolerance* ε*, and a maximum number of iterations* N*. The outputs should be either an approximate solution to the system* $A\vec{x} = \vec{b}$ *or an error message, along with the number of iterations completed. Additionally, it would be wise to build in functionality that allows you to optionally print the current estimated solution value at each iteration.*

Question 50.

Make a copy of your above code and modify it slightly, so that it approximates the solution using the Gauss-Seidel Method, with the same inputs and outputs.

Question 51.

Consider the system $\begin{pmatrix} 3 & 1 & -1 \\ 1 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \vec{x} = \begin{pmatrix} 0 \\ 5 \\ -5 \end{pmatrix}$.

(a) *Solve the above system analytically, using methods from prior weeks.*

(b) *Why should the Jacobi method converge to the solution for the above system?*

(c) *Test your Jacobi code for the above problem, using several combinations (of your choosing) of starting vector* $\vec{x}_0$ *and tolerance* ε*, then report the results (final output and number of iterations.) Make sure that the maximum number of iterations* N *is sufficient to allow convergence, as it should eventually converge for all starting vectors.*

Question 52.

Consider the system $\begin{pmatrix} -1 & 1 & 2 \\ 6 & -1 & 5 \\ 68.5 & -28.5 & -1 \end{pmatrix} \vec{x} = \begin{pmatrix} 1 \\ 0 \\ -20.5 \end{pmatrix}$*. The coefficient matrix is not invertible, and this system has infinitely many solutions, including* $\begin{pmatrix} -4 \\ -9 \\ 3 \end{pmatrix}$*. The spectral radius of the iteration matrix* $D^{-1}(D - A)$ *used in the Jacobi method is* 1.

(a) *Test your Jacobi code using several combinations of starting vector* $\vec{x}_0$*, tolerance* ε*, and max number of iterations* N*. For each, report your inputs, whether it successfully converged, and if so, the number of iterations. If the Jacobi method failed, report the last few candidate vectors* $\vec{x}_j$ *before the iteration cap was reached.*

(b) *Repeat part (a) with your Gauss-Seidel code, for the same combinations.*

(c) *Compare the results of these two methods.*

Question 53.

Consider the system $\begin{pmatrix} 1 & 2 & -1 \\ 2 & 1 & 1 \\ -1 & 0 & 1 \end{pmatrix} \vec{x} = \begin{pmatrix} 3 \\ 0 \\ 3 \end{pmatrix}$, *whose solution is* $\begin{pmatrix} -2 \\ 3 \\ 1 \end{pmatrix}$.

(a) *Explain why the Jacobi method should fail for this system. If this requires any matrix-related computations, you don't need to do these by hand.*

(b) *As in question 52.a, test your Jacobi code and report the results.*

(c) *Similarly, test your Gauss-Seidel code and report the results.*

(d) *Compare the methods. Though neither worked, does one appear to have performed worse than the other? How can you tell?*

■ Question 54. □

Consider the system $\begin{pmatrix} 1 & \gamma - 1 \\ \gamma - 1 & 1 \end{pmatrix} \vec{x} = \begin{pmatrix} \gamma \\ \gamma \end{pmatrix}$, *whose solution is* $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$.

(a) *Find the spectral radius of the iteration matrix* $D^{-1}(D - A)$ *by analytic methods, e.g. via the characteristic polynomial, as learned in MATH 211.*

(b) *Using the initial vector* $\vec{x}_0 = \vec{0}$, *tolerance* $\varepsilon = 10^{-8}$, *and a maximum number of iterations* N *large enough to assure convergence, run your Jacobi code for parameter values* $\gamma = 2^{-2},\ 2^{-4}, 2^{-6}$, *and* 2^{-8}. *Based on the numbers of iterations required for each to converge, make a prediction for the number of iterations needed for* $\gamma = 2^{-10}$, *then test that prediction.*

Chapter 3 | Least-squares approximation

§3.1 Discrete Least Squares

Interpolation connects data pairs using polynomial functions, where the degree of the polynomials is chosen so that all the pairs can be matched exactly. However, real-world data tend to be inaccurate, in which case exact matching is not necessarily desirable. Instead, we might know (or suspect) that a general type of function (not necessarily polynomial) underlies the data, and we want to choose a particular function of this type to represent the data. Identifying a particular "model" function for the data would allow us not only to predict the behavior of the system at points not represented in the collected dataset, but to study the system more generally. This process of identifying a particular model function for a set of data is called **data fitting** (or ''regression''), and we usually choose the model function that ''best'' fits the data in some sense.

We consider the problem using the following example.

Example 1.32

The temperature in Wooster, Ohio on Feb 15, 2022 was measured to be as in table 3.1. The data is also available as a csv file in Moodle for easier importing.

Knowing that daily temperatures are periodic with respect to the hour of day, we make an educated guess that a good choice to model the daily temperature would be a function that has a period of 24 hours.

Since we know that sine and cosine functions are periodic with period 2π, we make a guess that the temperature function is of the form

$$f(t) = a + b\sin\left(\frac{\pi}{12}t\right) + c\cos\left(\frac{\pi}{12}t\right)$$

Data-fitting with this model function means finding the coefficients a, b, and c so that the ''error" between observed and simulated data is the minimum.

Here's a typical way of calculating the ''error''. Given the data set $\{(t_i, \mathrm{T}_i)\}$, we can calculate the Cumulative Square Error $\mathrm{CSE}(a,b,c)$ as

$$\mathrm{CSE}(a,b,c) = \sum_{i=1}^{n} (\mathrm{T}_i - f(t_i))^2$$

This is question about optimization, and can be solved using multivariable calculus! We will see in the next section a way to find the best (a,b,c) using Linear Algebra.

Time	Temperature
2022-02-15 00:00:00	12.7
2022-02-15 01:00:00	12.7
2022-02-15 02:00:00	10.9
2022-02-15 03:00:00	12.7
2022-02-15 04:00:00	10.9
2022-02-15 05:00:00	10.9
2022-02-15 06:00:00	12.7
2022-02-15 07:00:00	10.9
2022-02-15 08:00:00	14.5
2022-02-15 09:00:00	21.7
2022-02-15 10:00:00	25.3
2022-02-15 11:00:00	27.1
2022-02-15 12:00:00	28.9
2022-02-15 13:00:00	30.7
2022-02-15 14:00:00	30.7
2022-02-15 15:00:00	32.5
2022-02-15 16:00:00	32.5
2022-02-15 17:00:00	32.5
2022-02-15 18:00:00	28.9
2022-02-15 19:00:00	28.9
2022-02-15 20:00:00	29.5
2022-02-15 21:00:00	31.3
2022-02-15 22:00:00	33.1
2022-02-15 23:00:00	34.9

Table 3.1

Question 55. Lab

(a) *Use Mathematica's* `Fit` *command or Python's* `scipy.optimize.curve_fit` *to find the periodic function* $f(t)$ *of the above form which has the least CSE with the temperature data.*

(b) Use `ListPlot` *(or equivalent command) to plot the temperature data, and then overlay the plot of the periodic function on the temperature data.*

(c) Now find the periodic function of the form

$$g(t) = a + b\sin\left(\frac{\pi}{12}t\right) + c\cos\left(\frac{\pi}{12}t\right) + d\sin\left(\frac{\pi}{24}t\right) + e\cos\left(\frac{\pi}{24}t\right)$$

which has the least cumulative squared-error with the temperature data, and show its graph on the same image as the overlay from the preceding problem. Compare how the two different periodic models fit the data.

3.1.1 Linear Least-Squares

The problem in example 32 can be recast as solving a linear system:

$$\begin{bmatrix} 1 & \sin\left(\frac{\pi}{12}t_1\right) & \cos\left(\frac{\pi}{12}t_1\right) \\ 1 & \sin\left(\frac{\pi}{12}t_2\right) & \cos\left(\frac{\pi}{12}t_2\right) \\ . & . & . \\ . & . & . \\ 1 & \sin\left(\frac{\pi}{12}t_9\right) & \cos\left(\frac{\pi}{12}t_9\right) \end{bmatrix}_{9\times 3} \begin{bmatrix} a \\ b \\ c \end{bmatrix}_{3\times 1} = \begin{bmatrix} T_1 \\ T_2 \\ . \\ . \\ T_9 \end{bmatrix}_{9\times 1} \tag{3.1}$$

This time however, the matrix is not square (it has 9 rows and 3 columns), which means there are 9 equations for 3 unknowns. Unless 6 of these equations are redundant (which they aren't for the data we have), the system is inconsistent and has no solution. That's why we didn't try to solve the system directly, but instead tried to find the combination of coefficients (a, b, c) that minimized the cumulative squared-error.

Normal Equation

Consider the linear system $A\vec{x} = \vec{b}$ where A be a $m \times n$ matrix with $m > n$ and $\vec{b}$ is a $m \times 1$ vector. Recall that the space Col(A) is a subspace of $\mathbb{R}^m$, defined as

$$\text{Col}(A) = \{A\vec{x} \mid \vec{x} \in \mathbb{R}^n\}$$

If $\vec{b} \notin \text{Col}(A)$, the system is inconsistent and doesn't have a solution. In such case, we we can still try to find an $\vec{x}$ that will minimize the "distance" of $A\vec{x}$ to $\vec{b}$ as follows. We define a **least-square solution** $\vec{x}^*$ to be an $\vec{x} \in \mathbb{R}^n$ such that

$$\|A\vec{x}^* - \vec{b}\|_2 \leq \|A\vec{x} - \vec{b}\|_2$$

for all $\vec{x} \in \mathbb{R}^n$. In other words, $\vec{x}^*$ is such that $\vec{b}^* = A\vec{x}^*$ is the point in Col(A) that is closest to $\vec{b}$.

Question 56. □

Check that $\|A\vec{x} - \vec{b}\|^2$ *for the system (3.1) is equal to the* CSE *from example 32.*

Geometrically speaking, $\vec{b}^* = A\vec{x}^*$ is the orthogonal projection of $\vec{b}$ onto Col(A) i.e we can write $\vec{b}$ as $\vec{b}^* + \vec{b}^\perp$ where

$$\vec{b}^* = \text{proj}_{\text{Col}(A)}\,\vec{b}$$

and

$$\vec{b}^{\perp} = \vec{b} - \vec{b}^{*} \text{ is perpendicular to } \mathrm{Col}(A).$$

In other words,

$$\vec{b} - \vec{b}^{*} \in (\mathrm{Col}(A))^{\perp}.$$

Question 57.

For the following problem, recall that matrix multiplication is nothing but a bunch of dot products!

(a) First show that $(\mathrm{Col}(A))^{\perp} = \mathrm{Nul}(A^{T})$.

(b) Using part (a), conclude that

$$A^{T}A\vec{x}^{*} = A^{T}\vec{b}. \tag{3.2}$$

Equation (3.2) above is called the **Normal Equation** for $A\vec{x} = \vec{b}$.

Conclusion

Any time we wish to fit a linear combination of functions of any kind (e.g. trigonometric, polynomial, exponential) to known data, we can simply build the corresponding linear system in the form $A\vec{x} = \vec{b}$ (with $\vec{x}$ representing the unknown coefficients of the functions) and solve the normal equation. This is a neat trick because the matrix $A^{T}A$ in the normal equation is always square (and symmetric); so we can use algorithms from the previous chapter to easily calculate the inverse.

Note that if $A\vec{x} = \vec{b}$ is consistent but has more than one solution, all of the solutions give $\vec{b}^{*} = \vec{b}$. When $A\vec{x} = \vec{b}$ is not consistent, the Normal Equation 3.2 is still consistent by definition. However it may have more than one solution i.e. there may be more than one best approximation. If the matrix A has full rank (i.e., there are as many linearly independent rows as there are components in $\vec{x}$), then the normal equation has a unique solution

$$\vec{x}^{*} = \left(A^{T}A\right)^{-1}A^{T}\vec{b}.$$

This section of the text on solving over determined systems is really a teaser for a bit of higher-level statistics, data science, and machine learning! The normal equations and solving systems via projections is the starting point of many modern machine learning algorithms. For more information on this sort of problem look into taking some statistics, data science, and/or machine learning courses.

Question 58.

Find the least-squares straight line fit to the data $f(-3) = f(0) = 0, f(6) = 2$ *by solving the Normal equation.*

Question 59. Optional

Let A *be an* $m \times n$ *matrix.*

(a) Show that $\mathrm{Nul}(A) = \mathrm{Nul}(A^{T}A)$.

HINT: It is easy to show that $\mathrm{Nul}(A) \subseteq \mathrm{Nul}(A^{T}A)$. *To show the opposite containment* $\mathrm{Nul}(A^{T}A) \subseteq \mathrm{Nul}(A)$, *use orthogonal complements.*

(b) Show that $\mathrm{rank}(A) = \mathrm{rank}(A^{T}A)$.

Hence if $\mathrm{Nul}(A) = \{0\}$, *we get* $\mathrm{Nul}(A^{T}A) = \{0\}$, *and* $A^{T}A$ *is invertible.*

3.1.2 Nonlinear to Linear

When we're trying to fit a nonlinear combination of functions to known data, we can't use the same approach as with linear combinations. In general, this is a hard problem that we discuss further in the next chapter. However, in some instances, it is possible to convert nonlinear problems into a linear form. Here's an example to demonstrate the process.

Example: Pipe Organ

In this example, we want to rebuild an antique pipe-organ from the remaining 5 of the original 12 pipes. Each pipe can produce a single pitch (frequency of sound), and based on this, we know that there is a formula of the type $y = ab^x$ for the diameter y of a pipe relative to its number x in the line of 12 ordered by diameter-size. We want to determine the best-fit values of a and b that model the pipe-diameters in our pipe-organ. The numbers x and measured diameters of the corresponding five remaining pipes are:

pipe number x	pipe diameter $d(x)$
2	7.8125
4	7.375
5	7.1875
9	6.4375
10	6.25

You can easily check that there are no exact values of a and b that fits the data exactly.

Question 60. **Lab**

Apply natural log to both sides of $f(x) = ab^x$, and reformulate the problem as a linear system $A\vec{x} = \vec{b}$.

(a) Find the corresponding normal equation and use Mathematica's `LinearSolve` *command to solve it.*

(b) Use your result to estimate the 12th pipe diameter.

§3.2 Continuous Least Squares

We can use the normal equation 3.2 from last section to find the best polynomial approximation $p_n(x)$ of a function $f(x)$ (which is a linear combination of monomials), by minimising the sum of squares of errors at $m > n$ nodes. In this section, we will see how to find an approximating polynomial that minimizes the error over all $x \in [a,b]$.

To formulate the problem, we need to define a way to 'measure' the error first, analogous to ℓ_2 norm in the discrete case. We will start by introducing a formal structure on $C[a,b]$.

Definition 2.33: Inner Product

We can define an inner product of the functions $f, g \in C[a,b]$ as

$$\langle f, g \rangle = \int_a^b f(x)g(x)w(x)\,dx.$$

for any choice of weight function $w(x)$ that is positive, continuous, and integrable on (a,b).

Note: The purpose of the weight function will be to assign varying degrees of importance to errors on different portions of the interval. This will be relevant later when talking about quadrature.

The inner product satisfies the following basic axioms:

- $\langle \alpha f + g, h \rangle = \alpha \langle f, h \rangle + \langle g, h \rangle$ for all $f, g, h \in C[a,b]$ and all $\alpha \in \mathbb{R}$;
- $\langle f, g \rangle = \langle g, f \rangle$ for all $f, g \in C[a,b]$;
- $\langle f, f \rangle \geq 0$ for all $f \in C[a,b]$.

With this inner product we associate the 'L^2 norm' that satisfies (N1), (N2), and (N3) properties of vector norms.

$$\|f\|_2 := \sqrt{\langle f, f \rangle} = \left(\int_a^b f(x)^2 w(x)\,dx \right)^{1/2}.$$

The 'L' stands for Lebesgue, coming from the fact that this inner product can be generalized from $C[a,b]$ to the set of all functions that are square-integrable, in the sense of Lebesgue integration. By restricting our attention to continuous functions, we dodge the measure theoretic complexities.

Example 2.34

For the weight function $w(x) = 1$ on $[0,1]$, we have

$$\|x\| = \sqrt{\langle x, x \rangle} = \sqrt{\int_0^1 x^2\,dx} = \frac{1}{\sqrt{3}}$$

The continuous least squares problem is to find the polynomial $p_n \in \mathcal{P}_n$ that minimizes $\|p_n - f\|_2$. The analogue of the normal equations is the following.

Theorem 2.35

Given $f \in C[a,b]$, the polynomial $p^* \in \mathcal{P}_n$ minimizes $\|q - f\|_2$ among all $q \in \mathcal{P}_n$ if and only if

$$\langle p^* - f, q\rangle = 0 \quad \text{for all } q \in \mathcal{P}_n$$

We will not prove this theorem. Note that we are again setting the ''error'' function orthogonal to the subspace P_n of $C[a,b]$.

Question 61.

Find the least squares polynomial $p_1(x) = c_0 + c_1 x$ that best approximates $f(x) = \sin(\pi x)$ on the interval $[0,1]$ with weight function $w(x) = 1$.

Hint: Why is it enough to find a p_1 such that $\langle p_1 - f, 1\rangle = 0 = \langle p_1 - f, x\rangle$?

In the previous question, we performed the minimization in the monomial basis. Our experience with interpolation suggests that different choices for the basis may yield approximation algorithms with superior numerical properties. Thus, we will next develop the form of the approximating polynomial in an arbitrary basis.

Suppose $\{\varphi_k\}_{k=0}^n$ is a basis for $\mathcal{P}_n$. In this basis, the least square solution $p^* \in \mathcal{P}_n$ can be written as

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

where $c_0, \ldots, c_n$ are the unknown coefficients to be found. We can find c_k by using theorem 35.

$$\begin{aligned} &\langle p^* - f, \varphi_k\rangle = 0 \\ \implies &\langle p^*, \varphi_k\rangle = \langle f, \varphi_k\rangle \\ \implies &c_0\langle \varphi_0, \varphi_k\rangle + c_1\langle \varphi_1, \varphi_k\rangle + \ldots + c_n\langle \varphi_n, \varphi_k\rangle = \langle f, \varphi_k\rangle \end{aligned}$$

for $k = 0, 1, \ldots, n$. In matrix form, the system of equations look like

$$\begin{bmatrix} \langle \varphi_0, \varphi_0\rangle & \langle \varphi_0, \varphi_1\rangle & \cdots & \langle \varphi_0, \varphi_n\rangle \\ \langle \varphi_1, \varphi_0\rangle & \langle \varphi_1, \varphi_1\rangle & \cdots & \langle \varphi_1, \varphi_n\rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \varphi_n, \varphi_0\rangle & \langle \varphi_n, \varphi_1\rangle & \cdots & \langle \varphi_n, \varphi_n\rangle \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} \langle f, \varphi_0\rangle \\ \langle f, \varphi_1\rangle \\ \vdots \\ \langle f, \varphi_n\rangle \end{bmatrix},$$

which we denote $G\vec{c} = \vec{b}$. The matrix G is called the Gram matrix. This matrix is obviously time-consuming to calculate in general, so we ask ourselves what choice of φ_k will make the computation simpler.

Note: Suppose we apply this method on the interval $[a,b] = [0,1]$ with weight $w(x) = 1$ and with the monomial basis $\varphi_k = x^k$. We find that

$$\langle x^i, x^j \rangle = \int_0^1 x^{i+j} = \frac{1}{i+j+1}$$

Thus the Gram matrix in this case is the notorious Hilbert matrix. It is exceptionally difficult to obtain accurate solutions with this matrix in floating point arithmetic, reflecting the fact that the monomials are a poor basis for $\mathcal{P}_n$ on $[0,1]$.

3.2.1 Orthogonal Polynomials

We say that two polynomials f and g are orthogonal to each other with respect to an inner product if $\langle f, g \rangle = 0$.

Definition 2.36

An *orthogonal family* of polynomials associated with the inner product is a set $\{\varphi_0, \varphi_1, \ldots, \varphi_n, \ldots\}$ where each φ_k is a polynomial of degree exactly k and the polynomials are mutually orthogonal i.e.

$$\langle \varphi_j, \varphi_k \rangle = 0 \text{ for } j \neq k.$$

Note: The orthogonality condition implies that each φ_k is orthogonal to all polynomials of degree less than k. (why?)

Question 62. □

Show that a family of orthogonal polynomials $\{\varphi_0, \varphi_1, \ldots, \varphi_n\}$ will form a basis of $\mathcal{P}_n$. There are two steps:

(a) Check that the family is linearly independent using the orthogonality condition.

(b) Prove that each of the monomial basis vectors $\{1, x, x^2, \ldots, x^n\}$ can be written as a linear combination of $\varphi_0, \varphi_1, \ldots, \varphi_n$. This will prove that they span $\mathcal{P}_n$.

In this basis, the Gram matrix transforms to a diagonal matrix! That makes solving for $\vec{c}$ much easier - so the only difficulty is finding an orthogonal basis to begin with. Fortunately, a similar process to the Gram-Schmidt algorithm from Linear Algebra is still applicable, we only need to replace dot product with inner products.

Theorem 2.37: Three Term Recurrence for Orthogonal Polynomials

Given a weight function $w(x)$, a real interval $[a,b]$, and an associated inner product

$$\langle f, g \rangle = \int_a^b f(x)g(x)w(x)\,dx$$

a family of (monic) orthogonal polynomials $\{\varphi_k\}_{k=0}^n$ can be generated as follows:

$$\begin{aligned}
\varphi_0(x) &= 1 \\
\varphi_1(x) &= x - \frac{\langle x\varphi_0, \varphi_0\rangle}{\langle \varphi_0, \varphi_0\rangle} \\
\varphi_2(x) &= x\varphi_1(x) - \frac{\langle x\varphi_1, \varphi_1\rangle}{\langle \varphi_1, \varphi_1\rangle}\varphi_1(x) - \frac{\langle x\varphi_1, \varphi_0\rangle}{\langle \varphi_0, \varphi_0\rangle}\varphi_0(x) \\
\varphi_k(x) &= x\varphi_{k-1}(x) - \frac{\langle x\varphi_{k-1}, \varphi_{k-1}\rangle}{\langle \varphi_{k-1}, \varphi_{k-1}\rangle}\varphi_{k-1}(x) - \frac{\langle x\varphi_{k-1}, \varphi_{k-2}\rangle}{\langle \varphi_{k-2}, \varphi_{k-2}\rangle}\varphi_{k-2}(x) \qquad \textit{for } k \geq 2
\end{aligned}$$

Question 63.

Use a basis of orthogonal polynomials to find the least squares polynomial $p_1 = c_0 + c_1 x$ that approximates $f(x) = \sin(\pi x)$ on the interval $[0,1]$ with weight function $w(x) = 1$.

Example 2.38: Legendre Polynomials

There are several different important families of orthogonal polynomials which are useful in Numerical Analysis. One example is the family of *Legendre polynomials* named after the French mathematician Adrien-Marie Legendre.[a] These are generated by the inner product with $w(x) \equiv 1$ on the interval $(-1, 1)$.

$$\begin{aligned}
\mathrm{L}_0(x) &= 1 \\
\mathrm{L}_1(x) &= x - \frac{\langle x\mathrm{L}_0, \mathrm{L}_0\rangle}{\langle \mathrm{L}_0, \mathrm{L}_0\rangle} = x - \frac{\int_{-1}^{1} x\,\mathrm{d}x}{\int_{-1}^{1} 1\,\mathrm{d}x} = x \\
\mathrm{L}_k(x) &= x\mathrm{L}_{k-1}(x) - \frac{\langle x\mathrm{L}_{k-1}, \mathrm{L}_{k-1}\rangle}{\langle \mathrm{L}_{k-1}, \mathrm{L}_{k-1}\rangle}\mathrm{L}_{k-1}(x) - \frac{\langle x\mathrm{L}_{k-1}, \mathrm{L}_{k-2}\rangle}{\langle \mathrm{L}_{k-2}, \mathrm{L}_{k-2}\rangle}\mathrm{L}_{k-2}(x) \qquad \text{for } k \geq 2
\end{aligned}$$

Now note that in the Gram–Schmidt process, for all k,

$$\langle x\mathrm{L}_{k-1}(x), \mathrm{L}_{k-1}\rangle = 0,$$

since if L_{k-1} is even, $x\mathrm{L}_{k-1}$ will be odd (or vice versa). So the recurrence relation gets simplified to two terms.

Traditionally, the Legendre polynomials are also normalized so that $\mathrm{L}_k(1) = 1$. After some simplifications, one can show

$$\mathrm{L}_k(x) = \frac{(2k-1)x\mathrm{L}_{k-1}(x) - (k-1)\mathrm{L}_{k-2}(x)}{k} \qquad \text{for } k \geq 2.$$

You can find graphs of Legendre polynomials (up to $n = 5$) below in fig. 3.1.

[a] Exercise: Find a good picture of Legendre.

Figure 3.1: Graphs of Legendre polynomials (up to $n = 5$) from Wikipedia

Question 64.

Compute $\mathrm{L}_2(x)$ *and verify that it is orthogonal to all polynomials* $c_0 + c_1 x$ *of lower degree.*

Conclusion

With this procedure, we can easily increase the degree of the approximating polynomial. To increase the degree by one, simply add

$$\frac{\langle f, \varphi_{n+1} \rangle}{\langle \varphi_{n+1}, \varphi_{n+1} \rangle} \varphi_{n+1}$$

Instead, if we were to use a non-orthogonal basis (e.g. the monomial basis), to increase the degree of the approximation we would need to solve a new $(n+2)$-by-$(n+2)$ linear system to find the coefficients of the least squares approximation.

Chapter 4 | Nonlinear Equations

The solution of nonlinear equations has been a motivating challenge throughout the history of numerical analysis. We opened these notes with mention of Kepler's equation, $M = E - e \sin E$, where M and e are known, and E is sought. One can view this E as the solution of the equation $f(E) = 0$, where

$$f(x) = M - x + e \sin x$$

This is a simple equation in one variable, $s \in \mathbb{R}$, and it turns out that it is not particularly difficult to solve. Other examples are complicated by nastier nonlinearities, multiple solutions (in which case one might like to find them all), ill-conditioned zeros (where $f(x) \approx 0$ for x far from the true zeros of f), solutions in the complex plane, and expensive f evaluations.

Optimization is closely allied to the solution of nonlinear equations, since we find extrema of $F : \mathbb{R} \to \mathbb{R}$ by solving

$$F'(x) = 0.$$

When $F : \mathbb{R}^n \to \mathbb{R}$, we seek $x \in \mathbb{R}^n$ that solves

$$\nabla F(\vec{x}) = \vec{0},$$

a nonlinear system of n equations in n variables. Handling $n > 1$ variables leads to more sophisticated theory and algorithms. Optimization problems become more difficult when additional constraints are imposed upon $\vec{x} \in \mathbb{R}^n$,

$$\min_{\vec{x} \in S} F(\vec{x}).$$

where S is the set of feasible solutions (e.g., vectors with nonnegative entries). When F is linear in $\vec{x}$ and $\vec{x}$ is constrained by simple inequalities, we have the important field of *linear programming*. When the set S constrains $\vec{x}$ to take discrete values (e.g., integers), the optimization problem becomes exceptionally difficult. Such combinatorial optimization or integer programming problems connect closely to the study of NP-hard problems in computer science.

In this course, we only have time to look at a few of the simplest algorithms for solving nonlinear equations. These will give some brief introduction to some of the over-arching themes in this important field. Further study is highly recommended!

§4.1 Bracketing Algorithms for Root Finding

Given a function $f : \mathbb{R} \to \mathbb{R}$, we seek a point $x^* \in \mathbb{R}$ such that $f(x^*) = 0$. This x^* is called a root of the equation $f(x) = 0$, or simply a zero of f. At first, we only require that f be continuous a interval $[a, b]$ of the real line, $f \in C[a, b]$, and that this interval contains the root of interest. The function f could have many different roots; we will only look for one. In practice, f could be quite complicated* (e.g., evaluation of a parameter-dependent integral or differential equation) that is expensive to evaluate (e.g., requiring minutes, hours, ...), so we seek algorithms that will produce a solution that is accurate to high precision while keeping evaluations of f to a minimum.

*In some applications, like polynomial root-finding, we know f has multiple zeros, and we might want to find all of them. This particular case is complicated by the fact that the zeros could be located in the complex plane, $x^* \in \mathbb{C}$.

Definition 1.39

We call the interval $[a,b]$ a zero-bracket of f if it contains a zero of f.

The first algorithms we study require the user to specify a finite interval $[a_0,b_0]$, a zero-bracket, such that $f(a_0)$ and $f(b_0)$ differ in sign, $f(a_0)f(b_0)<0$. Since f is continuous, the **intermediate value theorem** guarantees that f has at least one root x^* in the bracket, $x^*\in(a_0,b_0)$.

Question 65.

Draw a picture of what the intermediate value theorem says graphically.

4.1.1 Bisection Method

Question 66.

Design a zero-finding method that uses exactly one step to identify a smaller zero-bracket $[a_{k+1},b_{k+1}]$ inside a current zero-bracket $[a_k,b_k]$.

Modify your method if necessary to ensure that, starting from the zero-bracket $[a_0,b_0]$, it estimates a zero within $\frac{b_0-a_0}{2^n}$ of a true zero after n steps.

Advantages and Disadvantages

The bisection method cuts the error in half at each iteration, *independent of the behavior of f*. So, for example, Reduction of the initial bracket width by ten orders of magnitude would require roughly $\log_2 10^{10}\approx 33$ iterations. If f is fast to evaluate, this convergence will be pretty quick; moreover, since the algorithm only relies on our ability to compute the sign of $f(x)$ accurately, the algorithm is robust to strange behavior in f (such as local minima).

On the other hand, we do need to find an initial zero bracket to begin the search which can be done by checking incremental values of the function with small step sizes, but might fail due to "bad guess". Also, the bisection method cannot reliably estimate zeroes with even multiplicity. Finally, it has the serious disadvantage of only working in one dimension.

Question 67. Lab

The coding part of the following question is part of the next lab assignment (lab assignment 4.3).

Write a script/code to implement the bisection method. The inputs are

- *f - the function*
- *a - the lower limit of the initial zero-bracket*
- *b - the upper limit of the initial zero-bracket*
- *δ - an (optional) tolerance for the accuracy of the root*

If δ is not specified, you will need to provide a default value. You can either use a `While` *loop, or manually iterate the recursion. The output should be only* 1 *single number: the root.*

(a) Apply bisection to estimate a zero of the function

$$f(x) = xe^{-x} - 0.16064$$

between 0 and 1. Stop at the first estimate that is within 2^{-14} of a true zero.

(b) Apply bisection to find each of the two solutions to the equation

$$\sin x + x^2 = 2\ln x + 5$$

in the interval $[0, 5]$. Stop at the first estimate that is within 2^{-14} of a true zero.

HINT: Graph the function (use `Desmos`*) first to get the initial zero-brackets for each case.*

■ Question 68. □

*We say that a zero of a continuous function $f(x)$ is **simple**, if the graph of f passes through the zero on the x-axis by crossing from one side of the x-axis in the xy-plane to the other.*

(a) Give an example of a function that doesn't have any simple zero.

(b) Consider a general continuous function $f(x)$ with only simple zeroes in the interval (a, b). If $f(a) > 0 > f(b)$, sketch a generic picture and use it to explain why f must have an odd number of zeroes in (a, b).

4.1.2 Secant-Bracket

A simple adjustment to bisection can often yield much quicker convergence. Like bisection, this method starts with a zero-bracket $[a, b]$ and generates a new zero-bracket by testing a point inside the current zero-bracket $[a, b]$. However, instead of using the midpoint as in bisection, now we use the x-intercept of the secant line through the points $(a, f(a))$ and $(b, f(b))$ on the graph of f.

■ Question 69. □

Write the equation of the line connecting the points $(a, f(a))$ and $(b, f(b))$.

The motivation for this idea comes from the fact that the secant line is the linear polynomial approximation of f that interpolates the values at the endpoints. The expectation is that the x-intercept of this secant line approximation will occur closer to the endpoint of the current zero-bracket nearest the zero of f; which should give a better new zero-bracket than the one we would get from bisection.

■ Question 70. □

Use question 69 to find a formula for the x–intercept x_{sec} of the secant line. Use this to design an algorithm that generates a smaller zero-bracket $[a_{k+1}, b_{k+1}]$ inside a current zero-bracket $[a_k, b_k]$.

The secant-bracket method uses the same data $f(a_k)$ and $f(b_k)$ as bisection, but uses more information from that data to construct a zero-estimate; namely, the numerical values $f(a_k)$ and $f(b_k)$ instead of merely the signs of $f(a_k)$ and $f(b_k)$. This extra information is what gives the method its potential performance advantage.

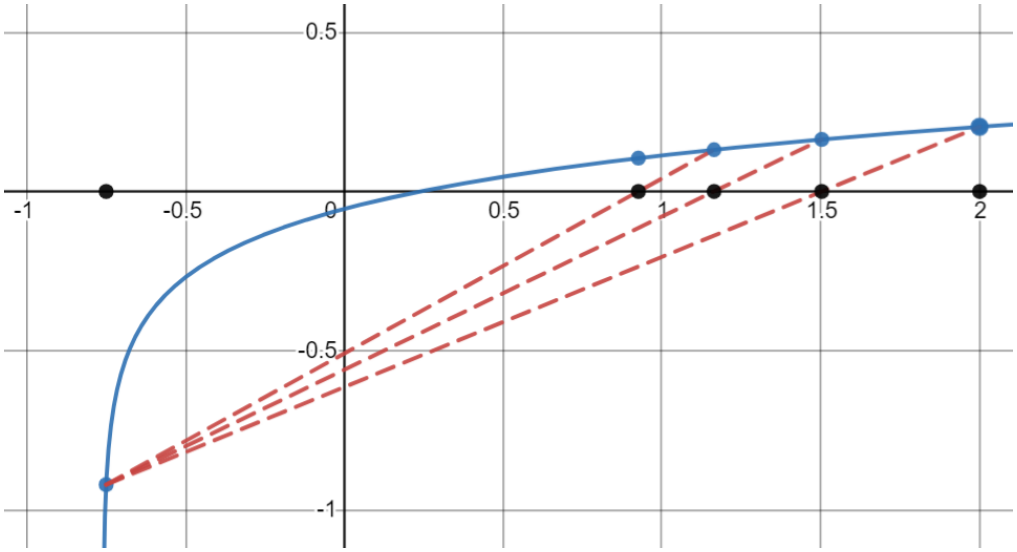
Secant-bracket by (m)any other name(s)

The secant-bracket method is sometimes called ''inverse linear interpolation'' because the secant line interpolates f at the points a and b, and because we're solving the ''inverse'' problem of finding the input x_{sec} that generates an output of zero from the secant line. It is also sometimes called *false position*, or the Latin version of this *regula falsi*.

Troubles with secant-bracket

- **Unreliable Speed** - In many cases, the secant-bracket method is faster than any of the methods we've seen so far. Unfortunately, in some cases it is much slower. For instance, the secant-bracket method will be very slow to identify a zero which occurs in an interval over which the graph transitions from steep to flat since the secant lines will be very poor approximations of the graph.

  The example below starts with a zero-bracket of $[-0.75, 2]$ and takes two steps of the secant-bracket method to generate the zero-brackets $[-0.75, 1.503]$, $[-0.75, 1.164]$, and $[-0.75, 0.926]$.



  The zero-estimate after the third step of the secant-bracket method is at 0.926. The bisection method from the same starting zero-bracket would generate a much better zero-estimate of 0.625 after only one step. This illustrates a general principle of numerical methods that you typically don't get something for nothing. In the case of the secant-bracket method, (potentially) improved speed is balanced by the uncertainty about whether this improved speed is actually achieved.

- **Zero-bracket** - Like bisection, the secant-bracket method needs to start with a zero-bracket. Unlike bisection, the length of the zero-brackets generated by the secant-bracket method do not necessarily shrink to zero. This means that we cannot use the size of the zero-bracket to gauge the accuracy of our zero-estimate. Another problem occurs whenever the x-intercepts of the secant lines and the actual graph of f approach the zero from opposite sides along the x-axis.

We will call a function f *convex* on $[a, b]$, if the secant line (linear interpolant) between any two points in $[a, b]$ is always above the graph. In other words,

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y)$$

for all $x, y \in [a, b]$ and $\lambda \in [0, 1]$. Although this definition looks weird, we already know a class of functions from calculus that are convex.*

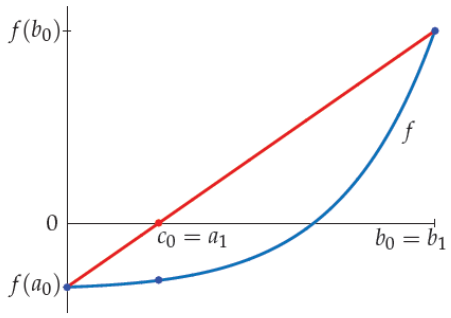
Proposition 1.40

If $f \in C^2[a, b]$, and $f'' \geq 0$ for all $x \in [a, b]$, then f is a convex function on $[a, b]$.

This observation plays a key role in the proof of the next theorem, which guarantees convergence of secant-bracket for convex functions.

Theorem 1.41

Suppose f is a convex function on $[a_0, b_0]$ with $a_0 < b_0$ and $f(a_0) < 0 < f(b_0)$. Then the secant-bracket method converges: either $f(x_{sec}) = 0$ at some stage $k \geq 0$, or $x_{sec} \to x^* \in [a_0, b_0]$ with $f(x^*) = 0$.



Proof of theorem 41.

We will give sketch of the proof. Let c_k denote the x-intercept at step k. Observe that either $f(c_k) = 0$ or $f(c_k) < 0$ implying

$$[a_{k+1}, b_{k+1}] = [c_k, b_k].$$

Then $\{c_k\}$ forms an increasing sequence bounded above by b_0, hence it must converge to some γ with $f(\gamma) \leq 0$. Check that γ must be a root of f. ■

Question 71. □

Write a script/function that implements the Regula Falsi method. Then use it to find the root of one of the equations from question 67 and compare the number of steps required to achieve the same amount of accuracy.

*Recall that in calculus, we call a function concave up if $f'' > 0$ and concave down if $f'' < 0$.

§4.2 Fixed Point Iteration

We have studied two bracketing methods for finding zeros of a function, bisection and *regula falsi*. These methods have certain virtues (most importantly, they always converge), but some exploratory evaluations of f might be necessary to determine an initial bracket. Moreover, while these methods appear to converge fairly quickly, when f is expensive to compute (e.g., requiring solution of a differential equation) or many systems must be solved (e.g., to evaluate $\sqrt{c}$ as a zero of $f(x) = x^2 - c$ for many values of A), every objective function evaluation counts. The next few sections describe algorithms that can converge more quickly that bracketing methods — provided we have a sufficiently good initial estimate of the root.

4.2.1 Newton's Method

None of the zero-finding methods we've seen so far has been famous enough to warrant attaching someone's name to it! The next method was first published in 1690 by the English mathematician Joseph Raphson, but a version of it was developed a few years earlier by Isaac Newton (who published it later in 1736). Because of this muddled history, the method is sometimes called the ''Newton-Raphson'' method. We'll use the shorter (and more traditional) title Newton's method.

Newton's method is based purely on local information at the current solution estimate, x_k. Whereas the bracketing methods only required that f be continuous, Newton's method needs $f \in C^1(\mathbb{R})$; to analyze the algorithm, we will further require $f \in C^2(\mathbb{R})$. This latter condition means that f can be expanded in a Taylor series centered at the approximate root x_k:

$$f(x^*) = f(x_k) + f'(x_k)(x^* - x_k) + \frac{1}{2}f''(\xi)(x^* - x_k)^2$$

where x^* is the exact solution, $f(x^*) = 0$, and ξ is between x_k and x^*. If we ignore the error term in this series, and we have a linear model for f

$$0 = f(x^*) \approx f(x_k) + f'(x_k)(x^* - x_k) \implies x^* \approx x_k - \frac{f(x_k)}{f'(x_k)}$$

■ **Question 72.** □

Check that $0 = f(x_k) + f'(x_k)(x - x_k)$ *is the equation of the tangent line to* $f(x)$ *at* $x = x_k$.

So we can formulate an iterative method a follows:

- Given a zero-estimate x_k, draw a tangent line to $f(x)$ at that point.
- Find where the tangent line intersects the x axis (we already found it).
- Use this intersection as the next zero-estimate x_{k+1}.

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

Repeat until satisfied that the zero-estimate is accurate enough (i.e. relative error is smaller than the tolerance).

The major drawback of Newton's method is that it demands the additional effort of computing the derivative.

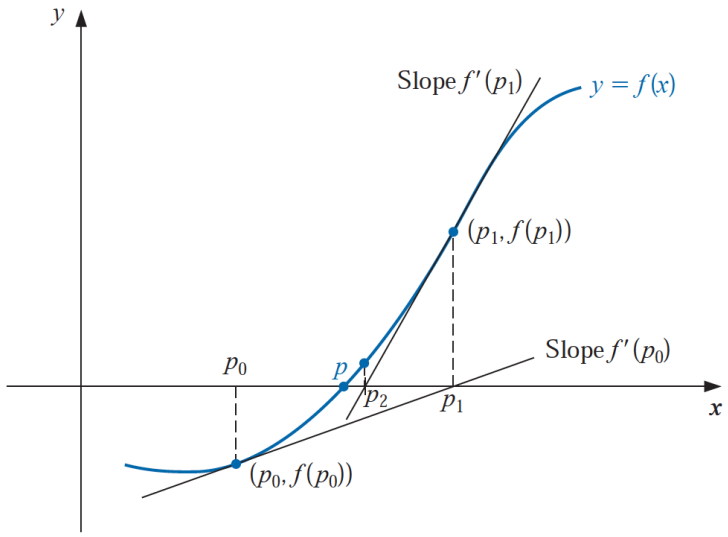


Figure 4.1

Cool facts about Newton's method

- Newton's method is used by computers to do simple division.
- Newton's method is used by computers to do square roots.
- Newton's method can find complex zeroes of polynomials if the original zero-estimate x_0 is a complex number.

Question 73.

Group

Given a fixed $c > 0$:

(a) For what simple function $f(x)$ will Newton's method estimate $\frac{1}{c}$ without using its value?

(Note $f(x) = \frac{1}{c} - x$ doesn't work because it requires us to know the value $\frac{1}{c}$ beforehand.)

(b) For what simple function $f(x)$ will Newton's method estimate $\sqrt{c}$ without using its value?

(c) In both cases, graph f to decide which original zero-estimates x_0 will work (i.e., cause the method to converge to the desired zero). Not all initial estimates work.

Question 74.

The coding part of this question is part of the next lab assignment (lab assignment 4.3).

Create a new script/function called to implement Newton's method. Your inputs are

- *the function f*
- *an initial guess,*
- *and an optional error tolerance.*

You don't need to set aside any code for calculating the derivative, you can do that analytically by hand.

Question 75.

Group

There are several reasons why Newton's method could fail. Work with your partners to come up with a list of reasons. Support each of your reasons with a sketch or an example.

Question 76.

Use Newton's method to estimate the zeroes of

$$g(x) = (x-1)^2(x+1)$$

by choosing original guesses exactly 0.2 *units to the right of each of the true zeroes. Identify the minimal number of steps to find each zero within* 0.0001*, and use the graph of* g *to explain the differences.*

Question 77.

Consider the function $f(x) = \sqrt[3]{x}$*. This function is differentiable everywhere except at* $x = 0$*. Find* $f'(x)$ *and rewrite* x_{k+1} *as a function of* x_k*. What does this tell us about the Newton iterations?*

§4.3 Lab Assignment 4: Root Finding

Write codes for two root-finding algorithms: bisection and Newton's method. Note that the Newton's Method code will require not only the function $f(x)$ but also its derivative $f'(x)$ which you may compute analytically and supply to the code. In each case, you will need to specify appropriate stopping criteria.

> **Note:** One of the failures of Newton's Method is that it requires a division by $f'(x_k)$. If $f'(x_k)$ is zero then the algorithm completely fails. Make sure to put an `if` statement to catch instances when Newton's Method fails in this way.

Question 78.

Use each algorithm to find the roots of $f(x) = \sin x - e^{-x}$ *to within the accuracy of* 10^{-5}, *in the interval* $[0, 5]$. *For Newton's method, you will need to choose appropriate initial zero-estimates* x_0 *for each root.*

Question 79.

Find the root of $f(x) = x^3 - 2x - \frac{3}{2}$ *which lies between* -1 *and* 2, *using both methods. For Newton's Method, try the values* $-1, 0, 1$, *and* 2 *for* x_0. *Explain any surprising results analytically (Both looking at the sequence* x_k *and plotting the function may prove beneficial.)*

Question 80.

Use one or both of your programs to find the point on the curve $y = 1/x$ *which lies closest to the* $(2, 1)$.

HINT: *This will be the point where the distance between the point and the curve is at a minimum, so optimization techniques from first-semester calculus apply.*

§4.4 Comparing Convergence

In the previous sections, we introduced a notion of convergence for zero-finding methods to indicate when the zero-estimates become arbitrarily close to a zero. We also saw that zero-finding methods can diverge (i.e. fail to converge) in a variety of ways. In this section, we'll explore computational effort as a means of comparing different converging methods.

4.4.1 Computational Effort

There are generally two components which contribute to the computational effort involved in a converging method: (1) The number of computations per step, and (2) the progress made per step.

Computations per step

Let's look at the number of computations per step for some of our methods so far:

- Bisection computes the midpoint $\frac{a+b}{2}$ of the current zero-bracket, and then evaluates the function f there.
- The secant-bracket method computes the x-intercept of the secant line

$$x_{\text{sec}} = a - f(a)\frac{b-a}{f(b)-f(a)},$$

  and then computes the function evaluation $f(x_{\text{sec}})$. (Recall that we have already evaluated $f(b)$ and $f(a)$ at preceding steps, so those don't count as additional computations when identifying x_{sec} at this step.)
- Newton's method computes the x-intercept

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

  of the tangent line, which involves not only the evaluation of the function at x_n but also its derivative there (which can involve even more computation if we don't have specific information about the derivative).

This list is arranged in ascending order of the number of computations per step; so, by this measure, bisection is the ''best'' method and Newton's method is the ''worst''.

Progress per step

In practice, zero-finding methods are stopped when an acceptable level of accuracy has been achieved, even if the current zero-estimate isn't exactly a zero. Since we don't know every practitioner's acceptable level of accuracy beforehand, we must accommodate arbitrarily strict levels of accuracy and thus arbitrarily large numbers of steps. As a result, our focus is not on counting steps, but on determining how much progress a step of a method makes toward a zero.

4.4.2 Comparing Convergence

To measure improvements in zero-estimates, we first consider the absolute error e_n associated with the n-th zero-estimate x_n of the zero z:

$$e_n = |x_n - z|.$$

It follows that the method converges (i.e., the zero-estimates x_n approach arbitrarily close to the zero z) if and only if the sequence of errors converges to zero: $e_n \to 0$. Thus, we know that the bisection method converges since its errors satisfy

$$e_n \leq \frac{b-a}{2^n}$$

This kind of diminishing bound on the errors is often called a ''rate'' of convergence.

Question 81. Lab

This problem is about analyzing (using a plot) the error between the approximate solution that the bisection method gives you and the exact solution to the equation. This is a bit of a funny thing! Stop and think about this for a second: if you know the exact solution to the equation then why are you solving it numerically in the first place!?!? However, whenever you build an algorithm you need to test it on problems where you actually do know the answer so that you can can be somewhat sure that it isn't giving you nonsense. Furthermore, analysis like this tells us how fast the algorithm is expected to perform.

Let's solve the very simple equation $x^2 - 2 = 0$ for x to get the solution $x = \sqrt{2}$ with the bisection method.

(a) Plot the absolute error e_n on the vertical axis and the iteration number n on the horizontal axis. The absolute error should follow an exponentially decreasing trend.

Observe that it isn't a completely smooth curve since we will have some jumps in the accuracy just due to the fact that sometimes the root will be near the midpoint of the interval and sometimes it won't be.

(b) Next plot $\log_2 e_n$ vs n. Your plot should look approximately linear (because of the observation made above). Use the `Fit` *command from Mathematica to find the slope of the best-fit straight line.*

As we observed in the plot above, $\frac{e_{n+1}}{e_n}$ is not always less than 1. However, if the error quotients are all **eventually** bounded above by a fixed constant $c < 1$

$$\frac{e_{n+1}}{e_n} \leq c < 1$$

we know that the errors will always shrink by (at least) a factor of c. Another way to say this is that we require

$$\limsup \frac{e_{n+1}}{e_n} < 1$$

where lim sup of a sequence is defined as the smallest upper bound of of limits of all possible sub-sequences.

Example 4.42

lim sup of the sequence $\{1, 2, 1, 2, 1, 2, \ldots\}$ is 2. lim sup of the sequence $\{\sin n\}_{n \in \mathbb{N}}$ is 1.

Order of Convergence

We've just seen that an eventual linear bound-relationship between errors, $e_{n+1} \leq ce_n$ for $c < 1$, ensures that the errors eventually shrink by at least the factor c. But what about a non-linear relationship like $e_{n+1} \leq ce_n^m$ for some power $m > 1$?

We can rewrite the non-linear relationship as follows:

$$e_{n+1} \leq ce_n^m = \underbrace{ce_n^{m-1}}_{c_n} e_n$$

where c_n is the "reduction factor" at the n–th step. Since $m > 1$ and $e_n \to 0$, we know that the reduction factors $c_n = ce_n^{m-1}$ will eventually be smaller than any fixed constant associated with linear convergence (no matter how large the constant c in the non-linear relationship). Moreover, the larger m the better, since then the reduction factors $c_n = ce_n^{m-1}$ shrink more quickly to zero.

This leads us to the following definition.

Definition 4.43: Order of Convergence

The order of convergence of a convergent sequence of zero-estimates $\{x_n\}$ is the highest power $m > 1$ for which error quotient $\frac{e_{n+1}}{e_n^\mu}$ is eventually bounded above.

$$m = \sup\left\{\mu \;\middle|\; \limsup_{n\to\infty} \frac{e_{n+1}}{e_n^\mu} < \infty\right\}$$

Notice that any finite eventual upper bound will do in this case, and that higher values of m indicate faster convergence. We call the (finite) value of the lim sup of the error quotient the *limiting constant*.

Definition 4.44

we say that a method has *linear convergence* when there is no $m > 1$ with a finite lim sup, but $m = 1$ has a limiting constant less than one, i.e.

$$\limsup_{n\to\infty} \frac{e_{n+1}}{e_n} = c, \qquad \text{with } 0 < c < 1$$

If $c = 0$ then the convergence is called *superlinear*.

Question 82.

Suppose that a sequence recursively defined as $x_{k+1} = g(x_k)$ converges to x_* as $k \to \infty$. Recall that $x_* = g(x_*)$, i.e. x_* is a fixed point of g. Let g' be continuous in the neighbourhood x_*. Prove the following claims.

(a) If $|g'(x_*)| \neq 0$ then the convergence will be linear with rate $\lambda = |g'(x_*)|$.

(b) If $|g'(x_*)| = 0$ then the convergence will be superlinear.

The special case of $m = 2$ is similarly called *quadratic convergence* since it corresponds to a quadratic bound-relationship like $e_{n+1} \leq ce_n^2$ between errors.

In the next subsection, we'll show Newton's method often has (at least) quadratic convergence. It can also be shown that zero-estimates from the secant-bracket method always exhibit an order of convergence of at least the "golden ratio" $m = \frac{1+\sqrt{5}}{2}$. Since this value is less than 2, Newton's method typically has faster

convergence than the secant-bracket method. Note that this is the opposite relationship to what we saw with the computations per step; illustrating again that a balance between pros and cons is typical of most particular methods.

Quadratic convergence for Newton's method

Suppose $\{x_n\}_{n\in\mathbb{N}}$ are the iterative zero-estimates for the actual zero z using Newton's method. The key to showing quadratic convergence here is the second order Taylor series of f.

$$f(x) = \overbrace{f(x_n) + f'(x_n)(x - x_n)}^{\text{Taylor polynomial at } x_n} + \overbrace{\frac{f''(\xi_n)}{2}(x - x_n)^2}^{\text{remainder}}$$

where ξ_n is some unidentified point between x_n and x.

Question 83.

Group

Justify the following steps.

(a) $0 = f(x_n) + f'(x_n)(z - x_n) + \dfrac{f''(\xi_n)}{2}(z - x_n)^2$

(b) $x_{n+1} - z = \dfrac{f''(\xi_n)}{2f'(x_n)}(z - x_n)^2$

(c) $\limsup \dfrac{e_{n+1}}{e_n^2} \leq \left|\dfrac{f''(z)}{2f'(z)}\right|$

The assumptions that make this all work are

(a) f is twice-continuously differentiable (i.e. it has continuous first- and second-derivative functions), i.e. $f \in C^2$,

(b) the zero-estimates converge to the exact zero $x_n \to z$, and

(c) $f'(z) \neq 0$ (which automatically ensures that $f'(x_n) \neq 0$ for x_n close enough to z since $f \in C^2$.

Under these conditions, the zero-estimates from Newton's method are guaranteed by the above argument to exhibit at least quadratic convergence.

Question 84.

In this question, we will continue question 81 with Newton's method and analyze the order of convergence.

(a) Suppose we are finding the root of the function $f(x) = x^2 - 3$. Store the Newton iterates $\{x_n\}$ in an array and create a plot of $X = \ln e_n$ vs. $Y = \ln e_{n+1}$. Use the appropriate command to find a best fit straight line through these data points and find its slope. With this graphical support, test for quadratic convergence.

(b) Does your rate change based on your initial guess? Again, please show some plots to make your case.

(c) Repeat the same for the function $f(x) = 13x - 1$.

The case of Multiple Roots

Definition 4.45

We say that the function $f \in C^m[a,b]$ has a zero of multiplicity m at p in (a,b) iff

$$0 = f(p) = f'(p) = f''(p) = \cdots = f^{(m-1)}(p), \quad \text{but} \quad f^{(m)}(p) \neq 0.$$

Question 85.

Show that the function $f(x) = e^x - x - 1$ has a zero of multiplicity 2 at $x = 0$. Then create a table of zero-approximates and plot them to show that the order of convergence using Newton's method is not quadratic in this case.

§4.5 Lab Assignment 5: Nonlinear Systems and Basin of Attraction

Newton's method generalizes to higher-dimensional problems where we want to find $\vec{x} \in \mathbb{R}^m$ that satisfies $\vec{f}(\vec{x}) = 0$ for some function $\vec{f} : \mathbb{R}^m \to \mathbb{R}^m$.

To see how it works, take $m = 2$ so that $\vec{x} = (x_1, x_2)^{\mathrm{T}}$ and $\vec{f} = [f_1(\vec{x}), f_2(\vec{x})]^{\mathrm{T}}$. Taking the linear terms in Taylor's theorem for two variables gives

$$0 \approx f_1(\vec{x}_{k+1}) \approx f_1(\vec{x}_k) + \left.\frac{\partial f_1}{\partial x_1}\right|_{\vec{x}_k} (x_{1,k+1} - x_{1,k}) + \left.\frac{\partial f_1}{\partial x_2}\right|_{\vec{x}_k} (x_{2,k+1} - x_{2,k}),$$
$$0 \approx f_2(\vec{x}_{k+1}) \approx f_2(\vec{x}_k) + \left.\frac{\partial f_2}{\partial x_1}\right|_{\vec{x}_k} (x_{1,k+1} - x_{1,k}) + \left.\frac{\partial f_2}{\partial x_2}\right|_{\vec{x}_k} (x_{2,k+1} - x_{2,k}).$$

In matrix form, we can write

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix} = \begin{pmatrix} f_1(\vec{x}_k) \\ f_2(\vec{x}_k) \end{pmatrix} + \begin{pmatrix} \partial f_1/\partial x_1(\vec{x}_k) & \partial f_1/\partial x_2(\vec{x}_k) \\ \partial f_2/\partial x_1(\vec{x}_k) & \partial f_2/\partial x_2(\vec{x}_k) \end{pmatrix} \begin{pmatrix} x_{1,k+1} - x_{1,k} \\ x_{2,k+1} - x_{2,k} \end{pmatrix}$$

The matrix of partial derivatives is called the *Jacobian matrix* $\mathrm{J}(\vec{x}_k)$, so (for any m) we have

$$\vec{0} = f(\vec{x}_k) + \mathrm{J}(\vec{x}_k)(\vec{x}_{k+1} - \vec{x}_k).$$

To derive Newton's method, we rearrange this equation for $\vec{x}_{k+1}$,

$$\mathrm{J}(\vec{x}_k)(\vec{x}_{k+1} - \vec{x}_k) = -f(\vec{x}_k) \quad \Longrightarrow \quad \vec{x}_{k+1} = \vec{x}_k - \mathrm{J}^{-1}(\vec{x}_k) f(\vec{x}_k).$$

> **Note:** If $m = 1$, then $\mathrm{J}(x_k) = \frac{\partial f}{\partial x}(x_k)$, and $\mathrm{J}^{-1} = 1/\mathrm{J}$, so this reduces to the scalar Newton's method.

Question 86.

Code Newton's Method for nonlinear systems of equations. You will need two functions/scripts:

- *one to compute the function $\vec{f}$, and*
- *the other to compute the Jacobian matrix (based on partial derivative formulas that you manually supply for a given problem).*

The inverse of the Jacobian needs to be dealt with carefully. We typically don't calculate inverses directly in numerical analysis, but since we have some other tools to do the work we can think of it as follows:

- *We need the vector $\vec{b} = \mathrm{J}^{-1}(\vec{x}_k) f(\vec{x}_k)$.*
- *The vector $\vec{b}$ is the same as the solution to the equation $\mathrm{J}(\vec{x}_k)\vec{b} = f(\vec{x}_k)$ at each iteration of Newton's Method.*
- *Therefore we can so a relatively fast linear solve (use a built-in solver or your code from Lab 2) to find $\vec{b}$.*
- *The Newton iteration becomes*

$$\vec{x}_{k+1} = \vec{x}_k - \vec{b}.$$

Question 87.

Test your code using the following problem that involves finding an intersection point of a sphere, cylinder, and paraboloid in 3-D space.

$$\vec{\mathbf{F}}(\vec{x}) = \begin{bmatrix} x_1^2 + x_2^2 + x_3^2 - 1 \\ x_1^2 + x_3^2 - \frac{1}{4} \\ x_1^2 + x_2^2 - 4x_3 \end{bmatrix}$$

Starting from an initial estimate of $x = \langle 1, 1, 1 \rangle$*, you should obtain convergence to roughly*

$$\langle 0.4408, 0.8660, 0.2361 \rangle$$

within a few iterations.

Question 88.

The nonlinear system below has four solutions. Use a graphing tool (e.g. DESMOS) to get initial estimates, and then Newton's Method to find each to ***five*** *decimal places.*

$$\begin{aligned} f_1(x,y) = x^2 + 4y^2 - 16 &= 0 \\ f_2(x,y) = xy^2 - 4 &= 0 \end{aligned}$$

4.5.1 Domain of Convergence

Definition 5.46

Given a root finding algorithm, the set of initial guesses (conditions) that, after iterations, converge to a fixed point, is called the domain of convergence.

In general, the domain of convergence is composed of multiple regions separated by boundaries which can be smooth or fractal curves. Determining the domain of convergence analytically is very difficult, so we use random sampling instead.

In the exercise below, we will try to plot the domain of convergence for the root of the nonlinear system of equation below (which has a single unique solution)

$$\begin{aligned} f_1(x,y) = x^3y - y - 2x^3 + 16 &= 0 \\ f_2(x,y) = x - y^2 + 1 &= 0 \end{aligned}$$

- First draw the graphs of the two functions to make an initial guess of the intersection point. You can use DESMOS or similar graphing tools for this part.
- Randomly sample a point within a reasonably sized 2D box. Choose $[-10, 10] \times [-10, 10]$ for this exercise, but allow the user to specify the size at runtime.
- Write a program that will repeatedly apply Newton's method to this starting point and keep track of whether the process converges to a specified tolerance within a fixed number of iterations. The values of tolerance and number of iteration should be specified as input variables.

- If the algorithm converged, color the starting point red (or some other color of your choosing). Otherwise color it white (or just don't color it).
- Repeat this process multiple times with random sampling each time, so that you get a colored picture of the 2D box showing where the starting points converge to.

Question 89. □

Write the code for the program with the given system running 1000 *or more repetitions with random initial points from* $[-10,10]\times[-10,10]$*, and using a tolerance of* 10^{-5} *in the* ℓ_∞ *norm. Vary the maximum number of iterations allowed (use* $nmax = 5, 10, 20,$ *and* 100*). Discuss the resulting convergence regions you find, including one or more relevant graphs.*

Your output should look similar to fig. 4.2.

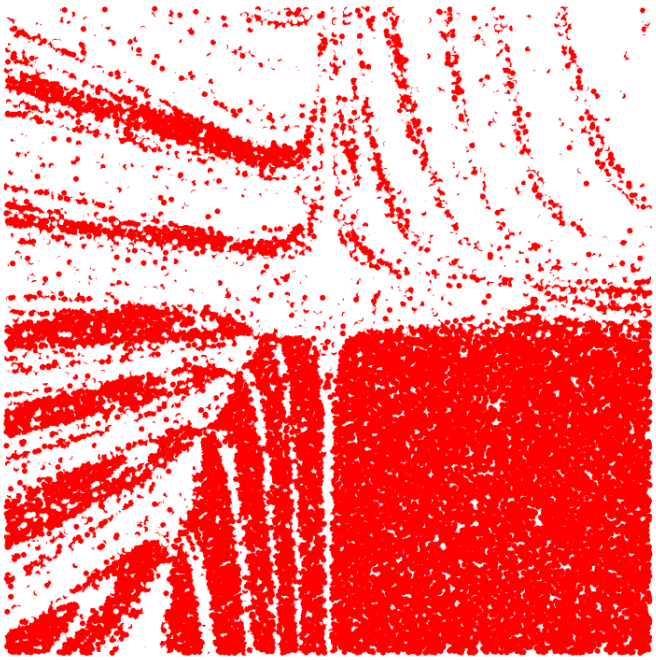


Figure 4.2: Convergence Region for 50000 random points, 20 iteration each

4.5.2 Basin of Attraction (Optional)

Consider a differentiable function $\vec{f}$ with several roots. Given a starting $\vec{x}$ value we should be able to apply Newton's Method to that starting point and we will converge to one of the roots (so long as you aren't in one of the special cases discussed earlier in the last chapters). It stands to reason that starting points near each other should all end up at the same root, and for some functions this is true. However, it is not true in general.

Definition 5.47

A **basin of attraction** for a root is the set of $\vec{x}$ values that converges to that root under Newton iterations.

In the next problem, you will produce a colored plot showing the basins of attraction for the given function. Do this as follows:

- Draw the graphs of the two functions to make an initial guess of the intersection points. [There should be three roots for the system below.]
- Assign each of the roots a different color. [Choose Blue, Green, Red.]
- Randomly sample a point within a reasonably sized 2D box. [Choose $[-1,1]\times[-1,1]$.]
- Write a program that will repeatedly apply Newton's method to this starting point and keep track of whether convergence to a specified tolerance occurred within a set number of iterations. [Set tolerance to be 10^{-5}.]
- Color the starting point according to the root that it converges to (you can figure it out by calculating ℓ_∞–distances from the roots, and finding the minimum). Choose a different color depending on the root.
- If the program diverges after the max number of iterations, color it Black.
- Repeat this process many many times with different starting points so that you get a colored picture of the 2D box showing where the starting points converge to. [Repeat at least 1000 times.]

The set of points that are all the same color will be the basin of attraction for the root associated with that color.

Question 90.

Use this new program on the system below running 1000 *or more repetitions with random initial points from* $[-1,1]\times[-1,1]$, *and using a tolerance of* 10^{-5} *in the* ℓ_∞ *norm. Vary the maximum number of iterations allowed (use* $nmax = 5, 10, 20,$ *and* 100*). Attach plots for the basins of attraction.*

$$\begin{aligned} f_1(x,y) = x^3 - 3xy^2 - 1 &= 0 \\ f_2(x,y) = 3x^2y - y^3 &= 0 \end{aligned}$$

Note: This system of equation is equivalent to the complex equation $z^3 = 1$ for $z = x + iy \in \mathbb{C}$.

Your output should look similar to fig. 4.3.

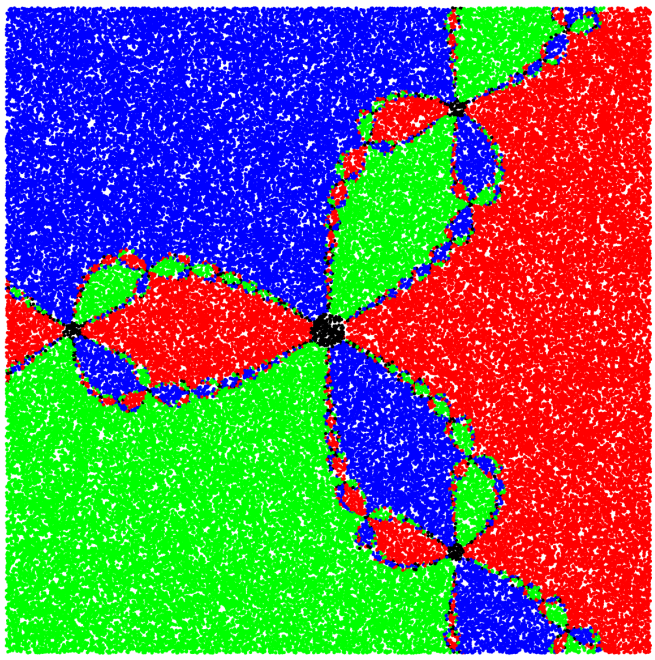


Figure 4.3: Basin of attraction for 10000 random points

§4.6 The Secant Method: A Quasi-Newton Method

Newton's Method has second-order (quadratic) convergence and, as such, will perform faster than the Bisection and Regula-Falsi methods. However, Newton's Method requires that you have a function and a derivative of that function. The conundrum here is that sometimes the derivative is cumbersome or impossible to obtain but you still want to have the great quadratic convergence exhibited by Newton's method.

Recall that Newton's method is

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

If we replace $f'(x_n)$ with an approximation of the derivative then we may have a method that is close to Newton's method in terms of convergence rate but is less troublesome to compute.

Definition 6.48

Any method that replaces the derivative in Newton's method with a numerical approximation is called a *Quasi-Newton Method*.

Recall that by definition,

$$f'(x_{n-1}) = \lim_{x \to x_{n-1}} \frac{f(x) - f(x_{n-1})}{x - x_{n-1}}$$

The first, and most obvious, way to approximate the derivative is just to *use the slope of a secant line* instead of the slope a tangent line in the Newton iteration. If we choose two starting points that are quite close to each other then the slope of the secant line through those points will be approximately the same as the slope of the tangent line.

Thus, assuming x_{n-2} is close to x_{n-1}, we can write

$$f'(x_{n-1}) \approx \frac{f(x_{n-2}) - f(x_{n-1})}{x_{n-2} - x_{n-1}} = \frac{f(x_{n-1}) - f(x_{n-2})}{x_{n-1} - x_{n-2}}$$

Note: We will learn better ways to approximate the derivative in the next chapter.

Using this approximation for $f'(x_{n-1})$ in Newton's formula gives

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

This is called the **Secant Method**.

4.6.1 Similarities and differences from secant-bracket method

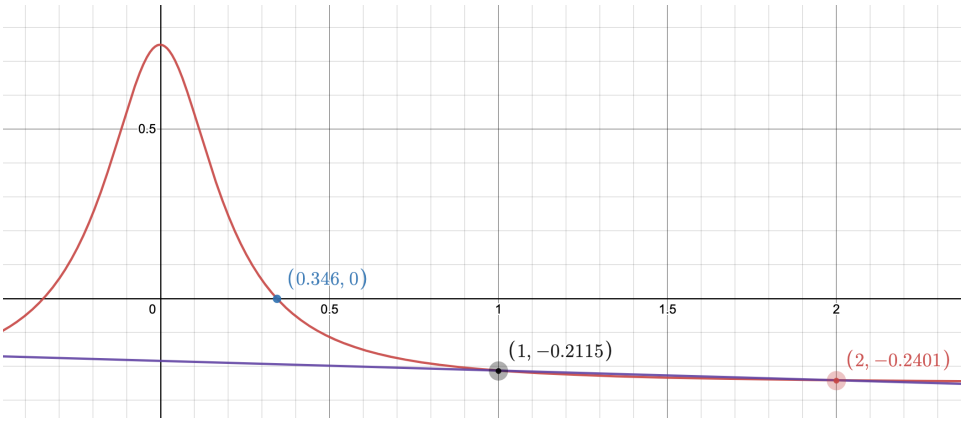
Both secant-bracket method and the secant method approximate f by a line that joins two points on the graph of f, but the secant method requires no initial bracket for the root.

Indeed, the zero-brackets generated by the secant-bracket method do not necessarily help us gauge how close our zero-estimate is to a zero, since the lengths of these zero-brackets don't necessarily shrink to zero.

In secant method, the user simply provides two starting points x_0 and x_1 with no stipulation about the signs of $f(x_0)$ and $f(x_1)$. As a consequence, there is no guarantee that the method will converge: as in Newton's method, a poor initial guess can lead to divergence!

Example 6.49

In the illustration below, we used a first pair of zero-estimates 1 and 2 whose corresponding points on the graph are along a section where the graph is asymptotic to the horizontal line $y = -0.25$. The secant line is nearly horizontal, so the next zero-estimate from the secant method will be very far to the left. Subsequent secant lines are also nearly horizontal, and the method diverges as the corresponding zero-estimates alternate between approaching $-\infty$ and ∞ (and completely avoid both of the zeroes ± 0.346 of this function).



Note: Another example of Quasi-Newton method (which we will not explore too too much) is Steffensen's method. In this method, the derivative $f'(x_n)$ is replaced by

$$\frac{f(x_n + f(x_n)) - f(x_n)}{f(x_n)}$$

4.6.2 Rate of Convergence

Clearly, the secant method convergence slower than the Newton's method, but it is still significantly faster than the bracketing algorithms (provided it does converge).

Theorem 6.50

Suppose x_* is a root of the nonlinear equation $f(x_*) = 0$. If $f'(x_*) \neq 0$, then the secant method converges for choice of x_1 and x_2 sufficiently close to x_*, and the order of convergence is $\varphi = \dfrac{1+\sqrt{5}}{2}$, the golden ratio.

Note: This illustrates that orders of convergence does not necessarily have to be integers.

Proof of theorem 50.

Recall that the linear interpolant to f at nodes x_k and x_{k-1} is given by

$$p(x) = f(x_k) + \frac{x - x_k}{x_{k-1} - x_k}\left(f(x_{k-1}) - f(x_k)\right)$$

The formula for the error for interpolants is _____ which implies

$$\mathrm{E}_1(x) = \frac{f''(\xi)}{2} e_k e_{k-1}$$

for some ξ between x_{k-1} and x_k. Then substituting $x = x_*$, we get,

$$f(x_*) = p(x_*) + \mathrm{E}_1(x_*) = f(x_k) + \frac{x_* - x_k}{x_{k-1} - x_k}\left(f(x_{k-1}) - f(x_k)\right) + \frac{f''(\xi)}{2} e_k e_{k-1}$$

On the other hand, in secant method, $p(x_{k+1}) = 0$ by definition. So

$$0 = p(x_{k+1}) = f(x_k) + \frac{x_{k+1} - x_k}{x_{k-1} - x_k}\left(f(x_{k-1}) - f(x_k)\right)$$

Subtract the two equations to conclude that

$$e_{k+1}\frac{f(x_{k-1}) - f(x_k)}{x_{k-1} - x_k} = \frac{f''(\xi)}{2} e_k e_{k-1} \implies e_{k+1} = \frac{f''(\xi)}{2f'(\eta)} e_k e_{k-1}$$

for some $\eta \in (x_{k-1}, x_k)$ by Mean Value Theorem.

As $x_k \to x_*$, both ξ and η also converge to x_*. Hence,

$$e_{k+1} \approx \frac{f''(x_*)}{2f'(x_*)} e_k e_{k-1} = \mathrm{C} e_k e_{k-1}$$

Now suppose r is the order of convergence so that for all j, we have

$$e_j \approx \mathrm{M} e_{j-1}^r$$

for some constant M. Then $e_{k-1} \approx \mathrm{M}^{-1/r} e_k^{1/r}$ and we can rewrite above equation as

$$e_{k+1} \approx \mathrm{C}\mathrm{M}^{-1/r} e_k^{1+1/r}$$

At the same time,

$$e_{k+1} \approx \mathrm{M} e_k^r$$

So equating the exponents, we get

$$r = 1 + \frac{1}{r} \implies r^2 - r - 1 = 0 \implies r = \frac{1 + \sqrt{5}}{2}.$$

■

Chapter 5 | Numerical Differentiation

We learned all about derivatives and their important applications in calculus, and we also learned rules for how to "symbolically compute" derivatives of some basic functions (i.e., write the symbols that represent the derivative function). However, in practice we often encounter functions as only a list of input/output pairs; and these functions cannot be symbolically differentiated because we don't have a symbolic representation of the function in the first place.

We also learned in calculus about differential equations and some of their important applications, we you even learned how to "symbolically solve" some differential equations. However, most differential equations cannot be solved symbolically.

The key to making progress in both of these situations is to develop good numerical approximations to derivatives of functions. One avenue for these approximations comes from the definition of the derivative as the limit of a difference-quotient.

§5.1 Difference-Quotients

Remember how the derivative was defined as the limit of difference-quotients?

$$f'(a) = \lim_{h\to 0} \frac{f(a+h) - f(a)}{h} \tag{5.1}$$

This definition automatically implies that the *difference-quotient* without the limit

$$\boxed{\mathrm{D}_{2\mathrm{F}}(h) := \frac{f(a+h) - f(a)}{h}} \tag{5.2}$$

is an approximation to the derivative $f'(a)$, and that smaller values of h eventually lead toward a perfect approximation. Recall that the approximation $\mathrm{D}_{2\mathrm{F}}(h)$ measures the slope of the secant line connecting the two points $(a, f(a))$ and $(a+h, f(a+h))$ on the graph of f.

Question 91.

Draw a picture to demonstrate the difference quotient.

The limit in the definition (5.1) of the derivative is typically taken from both sides of a; that using smaller and smaller values of positive or negative h. However, when approximating derivatives with difference-quotients, it is convenient for making comparisons to always use positive h. For this reason we define another difference-quotient approximation

$$\boxed{\mathrm{D}_{2\mathrm{B}}(h) := \frac{f(a) - f(a-h)}{h}} \tag{5.3}$$

called the **backward-difference** approximation, to distinguish it from the **forward-difference** approximation $\mathrm{D}_{2\mathrm{F}}(h)$ we defined in eq. (5.2).

Question 92.

Draw another picture to demonstrate the new difference quotient.

Note: The "backward" and "forward" terminology refer to the fact that $D_{2B}(h)$ for positive values of h uses x-coordinates $a-h$ to the left of the base point at $x=a$, and $D_{2F}(h)$ for positive values of h uses x-coordinates $a+h$ the right of the base point. Left is "backward" on the x-axis and right is "forward".

5.1.1 Theoretical Errors

As we can see from the definition of the derivative (5.1), as long as the function f is differentiable at a, both of the derivative approximations $D_{2F}(h)$ or $D_{2B}(h)$ should converge to the value $f'(a)$ as h is decreased toward zero. In order to compare these (and other convergent) derivative approximations, we analyze the error between $f'(a)$ and the approximation as a function of h :

$$E_{2F}(h) = f'(a) - D_{2F}(h) \qquad \text{and} \qquad E_{2B}(h) = f'(a) - D_{2B}(h).$$

Notice that these errors don't employ absolute values, and so can be negative.

Definition 1.51: Big "$\mathcal{O}$" notation

We write a function $f(h)$ as $\mathcal{O}(h^m)$(for an integer $m \geq 1$) if $f(h)$ can be written as an infinite series of the form

$$f(h) = C_m h^m + C_{m+1} h^{m+1} + \dots \tag{5.4}$$

in terms of some constant coefficients C_m, C_{m+1}, etc.

For our purposes here, the importance of this property is that if $E(h) = \mathcal{O}(h^m)$, then $E(h)$ diminishes like a multiple of h^m as h diminishes. The key principle here is that the lowest power m dominates the infinite series (5.4) when h is small; and small values of h are precisely where we expect our best approximations.

Note: This notation simplifies algebraic manipulations because it represents a category and not a particular element of a category. For instance, we have for any constant multiple $c \neq 0$ that

$$c\mathcal{O}(h^m) = \mathcal{O}(h^m), \quad \mathcal{O}(h^m) + \mathcal{O}(h^m) = \mathcal{O}(h^m), \quad \mathcal{O}((ch)^m) = \mathcal{O}(h^m),$$

because the corresponding infinite series (5.4) all still have lowest exponent m.

Question 93.

Use the Taylor series of f centered at a

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \dots \tag{5.5}$$

to show that $E_{2F}(h)$ is $\mathcal{O}(h)$.

§5.2 Better Approximation

An even better derivative approximation would have $\mathcal{O}\left(h^2\right)$ error, since then halving the value of h should give approximately one-quarter the error. One way to try to generate such an approximation is to average the two approximations $\mathrm{D}_{2\mathrm{F}}(h)$ and $\mathrm{D}_{2\mathrm{B}}(h)$:

$$\mathrm{D}_{3\mathrm{C}}(h) = \frac{\mathrm{D}_{2\mathrm{F}}(h) + \mathrm{D}_{2\mathrm{B}}(h)}{2} = \frac{f(a+h) - f(a-h)}{2h} \tag{5.6}$$

Question 94. □

Use the Taylor series (5.5) at $x = a+h$ and then $x = a-h$ to show that $\mathrm{E}_{3\mathrm{C}}(h) = \mathrm{D}_{3\mathrm{C}}(h) - f'(a)$ is $\mathcal{O}(h^2)$.

The approximation $\mathrm{D}_{3\mathrm{C}}(h)$ (5.6) is called the **centered-difference approximation** because it measures the slope of the secant line through $(a-h, f(a-h))$ and $(a+h, f(a+h))$ and the point a is "centered" between $a-h$ and $a+h$.

Question 95. □

Draw a third picture to demonstrate this new difference quotient.

5.2.1 Richardson Extrapolation

We can construct even better approximations by cleverly manipulating the ones we already have. The particular approach we'll investigate is credited to the English mathematician Lewis Fry Richardson, who introduced the idea in the early 20th century (Richardson was also a meteorologist who was one of the first to use mathematical modeling in weather forecasting).

The idea behind Richardson extrapolation is to use error series associated with an approximation to compare the approximation at h and at $2h$ (or some other multiple of h). As an example, we'll Richardson extrapolate our approximation $\mathrm{D}_{3\mathrm{C}}(h)$ to $f'(a)$ whose associated error $\mathrm{E}_{3\mathrm{C}}(h)$ is $\mathcal{O}\left(h^2\right)$:

$$\mathrm{E}_{3\mathrm{C}}(h) = \mathrm{C}_2 h^2 + \mathcal{O}\left(h^3\right)$$

Plugging $2h$ into this equation yields

$$\begin{aligned} \mathrm{E}_3(2h) &= \mathrm{C}_2(2h)^2 + \mathcal{O}\left((2h)^3\right) \\ &= 4\mathrm{C}_2 h^2 + \mathcal{O}\left(h^3\right) \end{aligned}$$

Notice that the first term in the error series at $2h$ is 4 times the first term in the error series at h; so that $4\mathrm{E}_{3\mathrm{C}}(h) - \mathrm{E}_{3\mathrm{C}}(2h)$ is $\mathcal{O}\left(h^3\right)$.

Question 96. □

Construct an weighted average $\widetilde{\mathrm{D}_{3\mathrm{C}}}(h)$ of $\mathrm{D}_{3\mathrm{C}}(h)$ and $\mathrm{D}_{3\mathrm{C}}(2h)$ that guarantees at least $\mathcal{O}\left(h^3\right)$ error.

Note: There is nothing special about taking $2h$; we could have taken $3h$ or even $h/2$, and modified the formula accordingly. But $2h$ is usually convenient.

The cost of improvement

Evidently, Richardson extrapolation can be used to improve the approximation. Of course, there is usually a price to pay for improvement, and we can see what the price is by looking carefully at our Richardson Extrapolation of the centered-difference approximation D_{3C} :

$$\widetilde{D_{3C}}(h) = \frac{f(a-2h) - 8f(a-h) + 8f(a+h) - f(a+2h)}{12h}.$$

It is clear that to compute this approximation to the derivative $f'(a)$, we need to make four function evaluations (at $a+2h, a+h, a-h$, and $a-2h$) as opposed to the two function evaluations (at $a-h$ and $a+h$) needed to compute the centered-difference approximation $D_{3C}(h)$.

Note: The forward- and backward-difference approximations effectively need only one function evaluation (since $f(a)$ doesn't change as h does), which balances the relatively poor $\mathcal{O}(h)$ error relationship. As we've seen before in this class; you never get something (improved approximation) for nothing (since you need to compute additional function evaluations).

5.2.2 Another Way: Interpolation

Another way to approximate derivatives $f'(a)$ would be to first approximate the function f, then to differentiate the approximation. We've already developed f-interpolating polynomials as a kind of approximation to a function f, so let's explore how this process works out in that case.

Recall that the Lagrange interpolating polynomial interpolating the pairs $(a, f(a))$ and $(a+h, f(a+h))$ is given by

$$p_1(x) = \frac{x-(a+h)}{a-(a+h)} f(a) + \frac{x-a}{(a+h)-a} f(a+h)$$

Question 97. □

Check that $p_1'(a) = D_{2F}(h)$.

Thus apparently, this is not "another" way at all. We've simply reproduced the forward-difference approximation $D_{2F}(h)$ we developed earlier. In similar fashion, we can reproduce the backward-difference approximation $D_{2B}(h)$ (by interpolating the pairs $(a-h, f(a-h))$ and $(a, f(a))$) and the centered-difference approximation $D_{3C}(h)$ (by interpolating the nodes $(a-h, f(a-h))$ and $(a+h, f(a+h))$).

If we keep increasing the number of nodes for interpolation and move the nodes forward or backward, we end up with higher order approximations for the derivative. We will list a couple of formula for the sake of completion. All of these can be derived from the proper Lagrange polynomial.

Interpolation Nodes	Approximation formula	Error
$a, a+h, a+2h$	$\mathrm{D}_{3\mathrm{F}}(h) = \frac{1}{2h}[-3f(a)+4f(a+h)-f(a+2h)]$	$\mathcal{O}(h^2)$
$a-2h, a-h, a+h, a+2h$	$\mathrm{D}_{5\mathrm{C}}(h) = \frac{1}{12h}[f(a-2h)-8f(a-h)+8f(a+h)-f(a+2h)]$	$\mathcal{O}(h^4)$
$a, a+h, a+2h, a+3h, a+4h$	$\mathrm{D}_{5\mathrm{F}}(h) = \frac{1}{12h}[-25f(a)+48f(a+h)-36f(a+2h)$ $+16f(a+3h)-3f(a+4h)]$	$\mathcal{O}(h^4)$

Question 98.

(a) *Give an approximation to $f'(a)$ which evaluates f at the points $a, a-h$ and $a-2h$ and with at least $\mathcal{O}\left(h^2\right)$ error.*

Hint: You can either work from the Lagrange interpolating polynomial at those three nodes. Or you can use $\mathrm{D}_{2\mathrm{B}}(h)$ and use Richardson extrapolation.

(b) *Use Richardson extrapolation on your approximation above to create an approximation to $f'(a)$ which evaluates f at the points $a, a-h, a-2h$, and $a-4h$ and with at least $\mathcal{O}\left(h^3\right)$ error.*

Question 99.

(a) *Describe how you can create an approximation to $f'(a)$ with at least $\mathcal{O}\left(h^{m+1}\right)$ error from an approximation $\mathrm{D}(h)$ to $f'(a)$ with $\mathcal{O}(h^m)$ error.*

(b) *Explain why this approximation with $\mathcal{O}\left(h^{m+1}\right)$ error is better than the one with $\mathcal{O}(h^m)$ error, and discuss the costs associated with such improvement.*

Question 100.

Find an $\mathcal{O}(h^4)$ approximation of $f'(a)$ which evaluates f at the points $a-h, a, a+h, a+2h$, and $a+3h$.

There are two ways to answer this question:

- You can start from the Lagrange interpolating polynomial at these nodes. Then take the derivative, and evaluate at $x = a$.
- Alternately, start from the Taylor series of f centered at a. You need to find a linear combination of $f(a-h), f(a), f(a+h), f(a+2h)$, and $f(a+3h)$ so that the difference-quotient differs from the actual Taylor series of $f'(x)$ by at most $\mathcal{O}(h^4)$. This will require that you solve a linear system of five equations.

You may use a computer algebra system (e.g. Mathematica) to automate any steps (such as construction and generation of Lagrange polynomial or solution of a system of equations), but should export the code and include it in your solution.

§5.3 Rounding Error Instability

If we generate data from a method with error satisfying $\mathrm{E}(h) = \mathcal{O}(h^m)$, we should see the ratios $\mathrm{E}(h)/h^m$ approach a constant as h diminishes. This follows from the expansion

$$\mathrm{E}(h) = \mathrm{C}_m h^m + \mathcal{O}\left(h^{m+1}\right)$$

since dividing both sides by h^m gives

$$\frac{\mathrm{E}(h)}{h^m} = \mathrm{C}_m + \mathcal{O}(h),$$

which should approach the coefficient C_m as h diminishes. However, this is usually not the case for the reason described below.

One very dangerous component of numerical derivative approximations is their tendency to be subject to "rounding" errors. These arise because computers store only finite significant digits (Mathematica typically uses 16) to represent numbers that may not be exactly represented in this way. Any terms beyond the stored digits are rounded (or truncated), which creates rounding errors.

It is particularly important to pay attention to round-off error when approximating derivatives. To illustrate the situation, let's examine the centered-difference formula

$$f'(a) = \frac{1}{2h}\left[f(a+h) - f(a-h)\right] + \mathcal{O}(h^2)$$

more closely. Suppose that in evaluating $f(a+h)$ and $f(a-h)$ we encounter round-off errors $e(a+h)$ and $e(a-h)$. Then our computations actually use the values $\tilde{f}(a+h)$ and $\tilde{f}(a-h)$, which are related to the true values $f(a+h)$ and $f(a-h)$ by

$$f(a+h) = \tilde{f}(a+h) + e(a+h) \quad \text{and} \quad f(a-h) = \tilde{f}(a-h) + e(a-h).$$

The total error in the approximation,

$$f'(a) - \frac{\tilde{f}(a+h) - \tilde{f}(a-h)}{2h} = \frac{e(a+h) - e(a-h)}{2h} + \mathcal{O}(h^2)$$

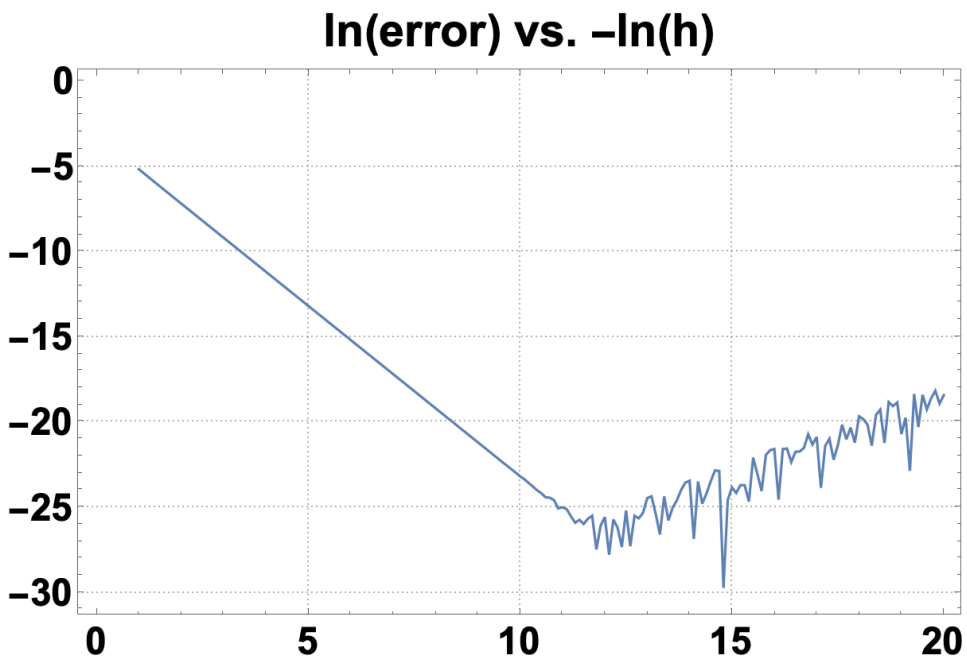
is due both to round-off error, the first part, and to truncation error. If we assume that the round-off errors $e(a \pm h)$ are bounded by some number $\varepsilon > 0$, then

$$|\mathrm{E}_{3\mathrm{C}}(h)| = \left| f'(a) - \frac{\tilde{f}(a+h) - \tilde{f}(a-h)}{2h} \right| \le \frac{\varepsilon}{h} + \mathcal{O}(h^2)$$

To reduce the truncation error, we need to reduce h. But as h is reduced, the roundoff error ε/h grows. In practice, then, it is seldom advantageous to let h be too small, because in that case the round-off error will dominate the calculations.

Example 3.52

Suppose we wish to approximate $f(x) = \ln(x)$ at $x = 2$. Here is a plot of $\ln(\mathrm{E}_{3\mathrm{C}}(h))$ vs $\ln(h)$ for $h \in (e^{-20}, e^{-1})$. Observe how once h is small enough, rounding error takes over and the error in the computed derivative starts to increase again.



Question 101. Lab

Write a program that will repeatedly approximate a derivative of $f(x) = \sin(x)$ *at* $a = \pi/6$*, for different values of the step size* h*, using* D_{3C}, D_{3F}, D_{5C}*, and* D_{5F}.

Vary h *from* 10^{-20} *to* 10^{-1}*, with many points in between, at least several hundred points. For each* h*, find the associated absolute error. You might find it helpful to change* h *by some multiplicative factor slightly smaller than* 1 *at each iteration.*

Your program should further make a log-log graph of the errors, with the $\log_{10}(h)$ *on the horizontal axis and* $\log_{10}$ *of the error on the vertical axis. Given that our* h *values are small,* $\log(h) < 0$*, so plotting* $-\log(h)$ *on the horizontal axis might make the graph more intuitive to read, but this is not required. Your loop should run enough times so that you see a significant qualitative change in the graph at a certain point similar to the example above.*

(a) *Produce two plots, one with the error results for both three-point formulas, and the other with the results of the five-point formulas. Include these in your write-up, making sure to label the graphs (or add a legend).*

(b) *For each method, the error steadily decreases to a certain point, and then changes erratically, overall increasing slightly as* h *continues to decrease. Report the (approximate) optimal* h *value and associated minimal error for each method. The optimal* h *value is where the behavior qualitatively changes, ignoring any "lucky" results from the erratic portion of the graph.*

(c) *Each of your plots should include a pair of parallel lines. Estimate the (integer) slopes of these lines, and interpret them, given that they come from log-log plots.*

Chapter 6 | Numerical Integration

Just as you learned about derivatives in calculus, you also learned about definite integrals $\int_a^b f(x)\,dx$ and some of their important applications. You should have noticed that it is often more difficult to symbolically integrate than it is to symbolically differentiate functions (e.g., integration by parts versus product rule); and that many functions cannot be symbolically integrated.

For example, one of the important integral in probability and statistics is the **error function** (which showed up in your take-home exam):

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\,dt.$$

It turns out that there is no closed form formula for the indefinite integral. Then how does a computer evaluate values for this function?

Another area where integrals show up is least squares approximations discussed in section 3.2. To find the optimal solution we had to compute integrals for the inner products

$$\langle f, \varphi_j \rangle = \int_a^b f(x)\varphi_j(x)\,dx$$

that form the right-hand side of the Gram matrix equation $\mathbf{G}\vec{c} = \vec{b}$. There are many other applications that require evaluation of definite integrals; integrals involving multiple variables pose additional challenges.

So this chapter will be about developing algorithms that approximate such integrals quickly and accurately. The difficulties associated with symbolic integration are so serious that you have already seen some numerical methods for approximating definite integrals in calculus (Riemann Sums). We'll explore these in more detail now, and introduce a few new ideas along the way.

§6.1 Quadrature using Riemann Sums

Since we want to sound more fancy than a calculus course, we will use the exotic new term quadrature to refer to numerically approximating definite integrals. This term is derived from the Latin ''quadra'', meaning ''dining table'', which has no connection whatever to numerical integration (unless you happen to be doing your homework at a dining table). The term ''quadrature'' in math historically refers to finding the area of a geometric figure in the plane by dividing it into shapes of known area (e.g., rectangles or triangles), and many of the numerical integration methods from calculus do just that.

Question 102. Riemann Sums

Write a code to approximate an integral with Riemann sums. Your function/script should accept the following inputs: a function, a lower bound, an upper bound, the number of subintervals, and an optional input that allows the user to designate whether they want left, right, or midpoint rectangles. Test your code on several functions

for which you know the integral. You should write your code without any loops (use the inbuilt `np.sum` *or* `Sum` *command).*

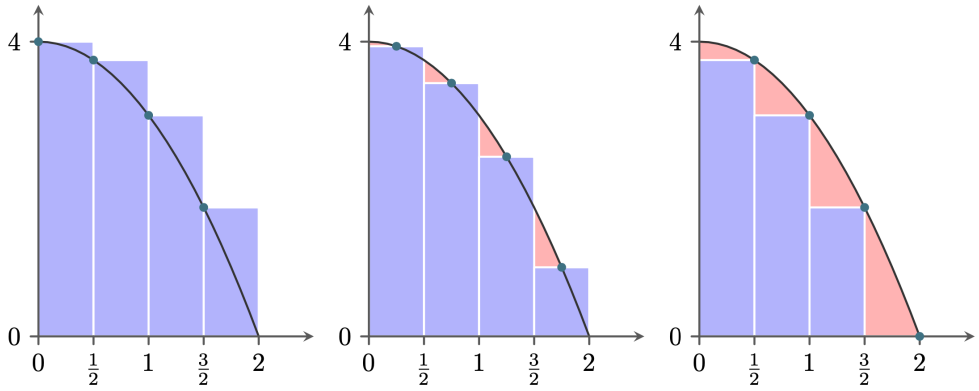


Figure 6.1: Left-aligned Riemann sums, midpoint-aligned Riemann sums, and right-aligned Riemann sums

■ Question 103. □

(a) *What happens if you have a monotonic increasing function? Will the right endpoint or left endpoint method overestimate?*

(b) *How about a monotonic decreasing function? Will the right endpoint or left endpoint method overestimate?*

Note: While writing the formula for the Riemann sums, you might have observed that each of the methods approximates the definite integral by a weighted average of the function values at an ordered set of points $\{x_0, x_1, \ldots, x_n\}$:

$$\int_a^b f(x)\,\mathrm{d}x \approx \sum_{i=0}^{n} w_i f(x_i)$$

where the coefficients w_i are the weights. Often (but not always), quadrature rules use only points x_i in the interval $[a, b]$. All quadrature methods are essentially about choosing the x_i and w_i in an efficient way.

Here's a way to get a better approximation. To motivate the idea consider fig. 6.2. It is plain to see that trapezoids will make better approximations than rectangles at least in this particular case. Another way to think about using trapezoids, however, is to see the top side of the trapezoid as a secant line connecting two points on the curve. As the width of the subintervals get arbitrarily small, the secant lines become better and better approximations for tangent lines and are hence arbitrarily good approximations for the curve. For these reasons it seems like we should investigate how to systematically approximate definite integrals via trapezoids.

■ Question 104. Trapezoid Rule

Consider a single trapezoid approximating the area under a curve. From geometry we recall that the area of a

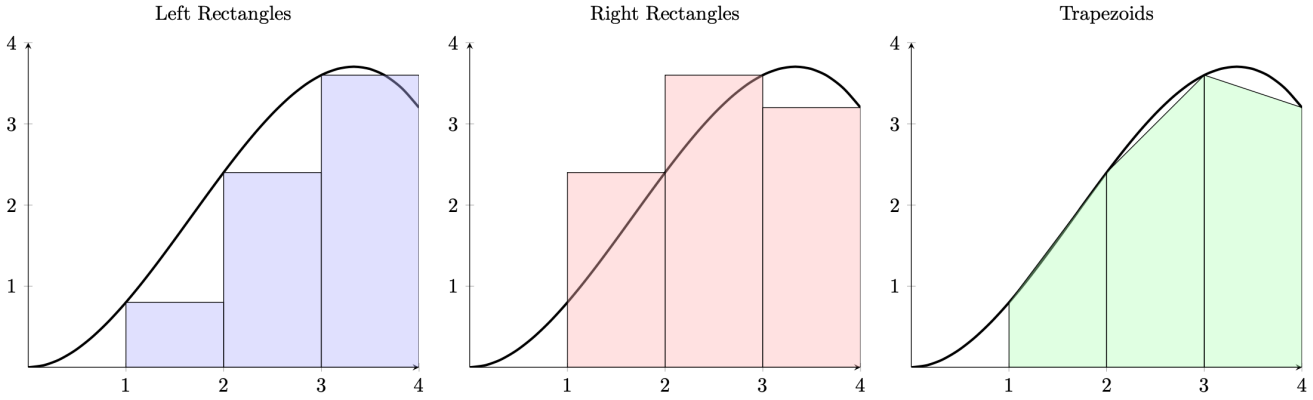


Figure 6.2: Using trapezoids to approximate a definite integral.

trapezoid is

$$\mathrm{A} = \frac{1}{2}(a+b)h$$

where a and b are length of two parallel sides and h is the distance between the two sides. The function shown in fig. 6.2 is $f(x) = \dfrac{x^2(5-x)}{2}$.

(a) Find the area of the third shaded region by hand (using the area formula for trapezoids above above) as an approximation to $\displaystyle\int_1^4 f(x)\,\mathrm{d}x$ *using* $h = 1$.

(b) Write code to give the trapezoidal rule approximation for the definite integral $\displaystyle\int_a^b f(x)\,\mathrm{d}x$. *Test your code on the function* $f(x)$ *and other examples where you know the definite area.*

§6.2 Interpolatory Quadrature

You may have noticed that the trapezoid rule essentially integrates the linear f–interpolation of the endpoints over each subinterval (i.e., it integrates the **linear spline**). It may not have occurred to you that the other three rules (left-endpoint, right-endpoint, and midpoint) can be considered as integration of different 0–degree interpolations (i.e., constants) on each subinterval (so they integrate a kind of pseudo-spline where the endpoints don't necessarily connect).

What happens if we consider an higher degree spline? As we will see in lab assignment 6.3, this indeed reduces the error – sometimes by an even greater margin than we might expect.

6.2.1 Simpson's Rule

Let's first consider the case of quadratic splines. We will start simple: consider a subdivision of $[a,b]$ into only two subintervals. So we are essentially taking the integral of the quadratic interpolant at the nodes $x_0 = a, x_1 = \dfrac{a+b}{2}$, and $x_2 = b$.

Using the Lagrange basis,

$$\mathrm{P}_2(x) = f_0 \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} + f_1 \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} + f_2 \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}$$

where $f_i = f(x_i)$. Let's integrate our interpolant to find an approximation of the integral:

$$\int_{x_0}^{x_2} f(x)\,\mathrm{d}x \approx f_0 \int_{x_0}^{x_2} \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}\,\mathrm{d}x + f_1 \int_{x_0}^{x_2} \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}\,\mathrm{d}x + f_2 \int_{x_0}^{x_2} \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}\,\mathrm{d}x$$

Let's evaluate the integrals using even spacing and a change of variables $h = \dfrac{b-a}{2}$, $t = \dfrac{x-x_0}{h}$, $\mathrm{d}t = \dfrac{\mathrm{d}x}{h}$, $x_1 = x_0 + h$ and $x_2 = x_0 + 2h$, so that:

$$\int_{x_0}^{x_2} f(x)\,\mathrm{d}x = \int_0^2 f(t)h\,\mathrm{d}t \approx f_0 \int_0^2 \frac{h^2(t-1)(t-2)}{(-h)(-2h)} h\,\mathrm{d}t + f_1 \int_0^2 \frac{h^2(t)(t-2)}{(h)(-h)}\,h\,\mathrm{d}t + f_2 \int_0^2 \frac{h^2(t)(t-1)}{(2h)(h)}\,h\,\mathrm{d}t$$

$$= \frac{hf_0}{2} \int_0^2 (t^2 - 3t + 2)\,\mathrm{d}t - hf_1 \int_0^2 (t^2 - 2t)\,\mathrm{d}t + \frac{hf_2}{2} \int_0^2 (t^2 - t)\,\mathrm{d}t$$

$$= \frac{hf_0}{2}\left(\frac{t^3}{3} - \frac{3t^2}{2} + 2t\Big|_0^2\right) - hf_1\left(\frac{t^3}{3} - t^2\Big|_0^2\right) + \frac{hf_2}{2}\left(\frac{t^3}{3} - \frac{t^2}{2}\Big|_0^2\right)$$

$$= \frac{hf_0}{2}\left(\frac{8}{3} - \frac{12}{2} + 4\right) - hf_1\left(\frac{8}{3} - 4\right) + \frac{hf_2}{2}\left(\frac{8}{3} - \frac{4}{2}\right)$$

$$= \frac{h}{3}(f_0 + 4f_1 + f_2)$$

The last formula is known as **Simpson's Rule** for approximating definite integrals. Because Simpson's rule captures part of the concavity of an underlying function, it is generally a better approximate than the trapezoid rule.

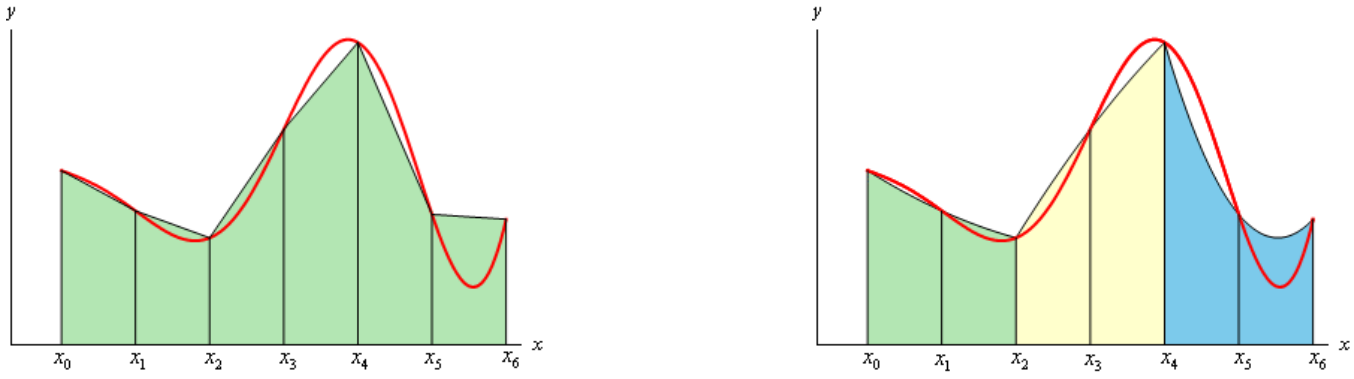


Figure 6.3: Linear vs. Quadratic Spline

Composite Simpson's Rule

If we want to use Simpson's rule for more finer partitions, we must first make sure that the number of intervals is even, say, 2M with $h = \dfrac{b-a}{2\mathrm{M}}$ (why?). Then we can write

$$\begin{aligned}\int_a^b f(x)\,\mathrm{d}x &= \sum_{k=1}^{\mathrm{M}} \int_{x_{2k-2}}^{x_{2k}} f(x)\,\mathrm{d}x \approx \sum_{k=1}^{\mathrm{M}} \frac{h}{3}\left[f(x_{2k-2}) + 4f(x_{2k-1}) + f(x_{2k})\right] \\ &= \frac{h}{3}\left[f_0 + 4f_1 + 2f_2 + 4f_3 + 2f_4 + \cdots + 2f_{2\mathrm{M}-2} + 4f_{2\mathrm{M}-1} + f_{2\mathrm{M}}\right] \\ &= \underbrace{\frac{h}{3}(f_0 + f_{2\mathrm{M}}) + \frac{2h}{3}\sum_{k=1}^{\mathrm{M}-1} f_{2k} + \frac{4h}{3}\sum_{k=1}^{\mathrm{M}} f_{2k-1}}_{\text{Composite Simpson's Rule}}\end{aligned}$$

Question 105.

Write a code for composite Simpson's rule.

Question 106.

Use your code to find the smallest number of equally-spaced subintervals that produces a composite Simpson's approximation of

$$\int_0^b x^2 \sin(2x)\,\mathrm{d}x$$

within an error tolerance of 0.001 *(Hint: Use Mathematica's* `NIntegrate` *command or Python's* `quad` *from scipy.integrate to get the "true" value of the integral.)*

(a) where b is the first zero of the integrand $x^2\sin(2x)$ to the right of zero.

(b) where b is the second zero of the integrand to the right of zero.

6.2.2 Newton-Cotes Quadrature

What happens if we consider a polynomial interpolant of higher order instead of a spline? Quadrature rules based on integrating the polynomial interpolant of higher degree is known as **Newton-Cotes quadrature**. This will increase the number of evaluations of f (often very costly), but hopefully will significantly decrease the error. This is essentially true until $n = 8$. After that point, negative weights appear in the Newton-Cotes formulas, and rounding error can become a significant feature of the calculations. For this reason, the Newton-Cotes formulas beyond $n = 7$ are rarely used.

Using `Mathematica`, the Newton-Cotes quadrature formula for degree 3 and 4 were computed to be as follows:

$$n = 3 \quad \to \quad (b-a)\frac{f(a) + 3f\left(a + \frac{b-a}{3}\right) + 3f\left(a + 2\left(\frac{b-a}{3}\right)\right) + f(b)}{8} \quad (\text{“Simpson's } \frac{3}{8} \text{ rule”})$$

$$n = 4 \quad \to \quad (b-a)\frac{7f(a) + 32f\left(a + \frac{b-a}{4}\right) + 12f\left(a + 2\left(\frac{b-a}{4}\right)\right) + 32f\left(a + 3\left(\frac{b-a}{4}\right)\right) + 7f(b)}{90}$$

Note: Here is another example to emphasize why high-degree Newton–Cotes rules can be a bad idea. Recall that Runge's function $f(x) = \dfrac{1}{1+x^2}$ gave a nice example for which the polynomial interpolant at uniformly spaced points over $[-1, 1]$ fails to converge uniformly to f. This fact suggests that Newton–Cotes quadrature will also fail to converge as the degree of the interpolant grows, and indeed it does. On the other hand, integrating the interpolant at Chebyshev nodes, called **Clenshaw–Curtis quadrature**, does indeed converge.

6.2.3 Degree of Exactness

Observe that if $f \in \mathcal{P}_n$, its polynomial interpolant $p_n \in \mathcal{P}_n$ is exactly f, and so the exact integral of p_n is the same thing as the exact integral of f. However, there are special cases where a degree-n interpolant will exactly integrate polynomials of higher degree. This motivates the next definition.

Definition 2.53

An interpolatory quadrature rule has degree of exactness m if for all $f \in \mathcal{P}_m$,

$$\int_a^b f(x)\mathrm{d}x = \sum_{j=0}^{n} w_j f\left(x_j\right)$$

Clearly a degree-n quadrature rule has degree of exactness $m \geq n$. Thus it is particularly interesting to see circumstances in which this degree of exactness is exceeded.

■ **Question 107.** □

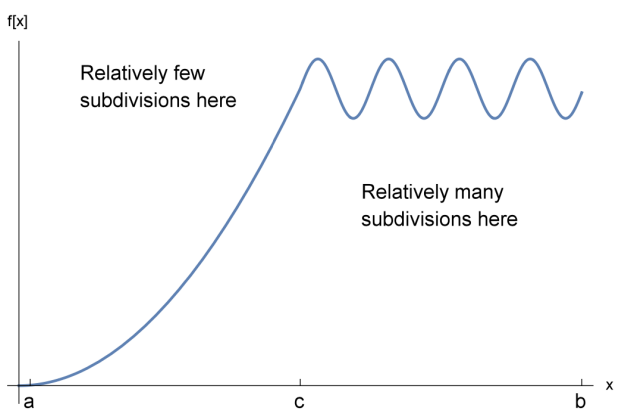
Check that although the Simpson's rule is based on quadratic interpolants, it is exact for cubic polynomials.

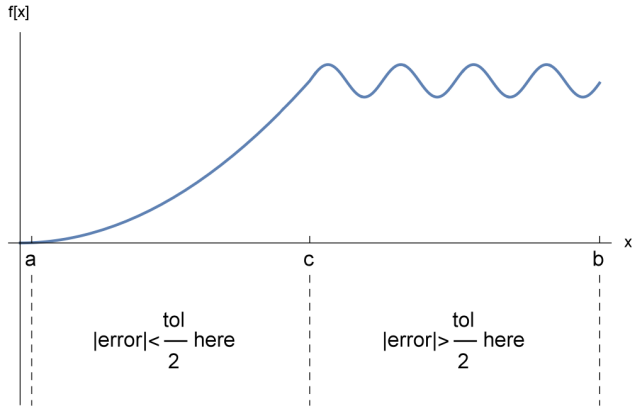
We will see later that similar results also hold for higher even degree Newton-Cotes quadratures.

6.2.4 Adaptive Quadrature

We now understand that the approximations given by different quadrature rules can be improved differently by taking smaller subdivisions. However, smaller subdivisions take more computational effort, so in practice we want to be careful about how we do this. One idea that is used by most commercial numerical integration solvers is called adaptive quadrature, because the size of the subdivisions are adapted as the method progresses.

The basic idea behind adaptive quadrature is to use more subdivisions only on the parts of the interval $[a,b]$ where necessary. For instance, if a function is relatively flat and straight over one section of the interval $[a,b]$, we might not need many subdivisions to get an accurate approximation of the integral. If that same function is very wavy over some other section of the interval $[a,b]$, more subdivisions may be needed there.





In practice, we don't have an image like this to help us to decide where more subdivisions may be needed. Instead, we decide where to do more subdivisions by cleverly comparing local error estimates.

Algorithm Pseudocode

- Compute the approximate area using Simpson's rule.

$$S(a,b) = \frac{(b-a)}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$

- Halve the interval and compute $S\left(a, \frac{a+b}{2}\right)$ and $S\left(\frac{a+b}{2}, b\right)$ Estimate the error,

$$\text{error } = \frac{1}{15}\left|\left(S\left(a, \frac{a+b}{2}\right) + S\left(\frac{a+b}{2}, b\right)\right) - S(a,b)\right|$$

- If the error is less than some specified tolerance $= tol$, we are done, otherwise recursively compute each of $S\left(a, \frac{a+b}{2}\right)$ and $S\left(\frac{a+b}{2}, b\right)$ with tolerance $= \frac{tol}{2}$.

Adaptive quadrature starts with the approximation on the interval $[a,b]$, and breaks that into two subdivisions $[a,c]$ and $[c,b]$; usually using the midpoint $c = \frac{b+a}{2}$. The local errors on each subdivision $[a,c]$ and $[c,b]$ are then compared to the local error on the original interval $[a,b]$ in a way that allows us to estimate

whether the approximation from the two subdivisions is within some tolerance (that we choose) of the actual value of the integral; all without knowing the actual value of the integral!

Exploration Activity

There are lots of other quadrature techniques that we do not have time to go over in this course, such as:

- **Gaussian quadrature** (uses orthogonal polynomials to find best fit approximation and then integrate that) -

  Suppose that $x_0, x_1, \ldots, x_n$ are roots of the nth Legendre polynomial $\mathrm{P}_n(x)$ and that for each $i = 0, 1, \ldots, n$ the numbers w_i are defined by

  $$w_i = \int_{-1}^{1} \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} dx = \int_{-1}^{1} \ell_i(x) dx$$

  Then

  $$\int_{-1}^{1} f(x) dx = \sum_{i=0}^{n} w_i f(x_i)$$

  where $f(x)$ is any polynomial of degree less or equal to $2n + 1$.

- **Monte Carlo quadrature** (uses a random sampling based statistical approach) -

  In this method, we compute the integral of $f : \mathbb{R}^d \to \mathbb{R}, d \geqslant 1$ by generating n random points in $\mathbb{R}^d$ and use the approximation,

  $$\iint \ldots \int_{\Omega} f(x_1, x_2, \ldots, x_d)\, dx_1 dx_2 \ldots dx_d \approx \text{volume}(\Omega) * \frac{\sum_{i=1}^{n} f(\mathbf{z_i})}{n}$$

  where $\mathbf{z_i}$ are randomly chosen values from $\mathbb{R}^d$.

- Romberg integration (uses Richardson Extrapolation on Trapezium rule) etc.

Please feel free to choose one of these as your final project topic.

§6.3 Lab Assignment 6 - Quadrature Error Analysis

6.3.1 Experimental Analysis

For the first part of this lab, you will be using various quadrature rules we have studied so far to approximate a "test problem" whose exact result is known. In this case, we will consider the definite integral

$$\int_0^{\pi/2} \sin x \, dx$$

Recall that computational error includes both methodological error (e.g. from Taylor series remainder terms) and round-off error (inexact arithmetic & storage on computers.)

Question 108. □

For each of the five different quadrature rules below, approximate the integral for $n = 10$, $n = 100$, and $n = 1000$ subintervals, compute the corresponding errors, and indicate how the error appears to depend on the value of $h = \frac{b-a}{n}$ for each method. In particular, does the data appear to show errors of type $\mathcal{O}(h), \mathcal{O}(h^2), \mathcal{O}(h^3)$, or $\mathcal{O}(h^4)$?

Use tables to organize your results.

(a) Right-endpoint Riemann Sum.

(b) Left-endpoint Riemann Sum.

(c) Midpoint Riemann Sum.

(d) Composite Trapezoid Rule.

(e) Composite Simpson's Rule.

6.3.2 Theoretical Error Analysis of Trapezoid Rule

Our basic approach will be to analyze the "local" error associated with a single subinterval of a quadrature rule, and then propagate that error across all subintervals to establish a "global" error relationship on the entire interval of integration $[a, b]$.

Local Error

All of our quadrature rules so far have used n subdivisions of length $h = \frac{b-a}{n}$ of the interval of integration $[a, b]$. The endpoints defining these subintervals are thus

$$x_0 = a, x_1 = a + h, \ldots, x_i = a + ih, \ldots, x_n = b$$

so that a typical subinterval can be represented as $[x_i, x_i + h]$.

Question 109.

Trapezoid Rule

Recall that the Trapezoid rule uses linear interpolant between successive points. Suppose the linear f–interpolant on $[x_i, x_{i+1}]$ is $\mathrm{T}_i(x)$. We will define

$$\mathrm{E}_i(h) = \int_{x_i}^{x_i+h} (f(x) - \mathrm{T}_i(x))\, \mathrm{d}x$$

Show that

$$\mathrm{E}_i(h) = \int_{x_i}^{x_i+h} \frac{f^{(2)}(\xi)}{2!}(x - x_i)(x - x_i - h)\, \mathrm{d}x$$

for some $\xi \in [x_i, x_i + h]$.

Warning: Observe that ξ is not a constant, its value depends on x. As such, we can't just pull it outside the integral.

At this point, we will need to use the generalized Mean Value Theorem for Integrals. A proof of this result can be found in a standard Calculus textbook.

Theorem 3.54

Let f and g be continuous real functions on the closed interval $[a, b]$ such that:

$$(\forall x \in [a,b])(g(x) \geq 0) \qquad \text{or} \qquad (\forall x \in [a,b])(g(x) \leq 0)$$

Then there exists a real number $k \in [a, b]$ such that:

$$\int_a^b f(x)g(x)\mathrm{d}x = f(k)\int_a^b g(x)\mathrm{d}x$$

Now note that in the interval $[x_i, x_i + h]$, the product $(x - x_i)(x - x_i - h)$ is always negative. Hence using the GMVT for integrals, we can write

$$\mathrm{E}_i(h) = f^{(2)}(\eta) \int_{x_i}^{x_i+h} \frac{(x - x_i)(x - x_i - h)}{2}\, \mathrm{d}x$$

for some $\eta \in [x_i, x_i + h]$.

Question 110.

Use a change of variable to integrate the last integral and show that $\mathrm{E}_i(h) = \mathcal{O}(h^3)$.

Global Error

To propagate local errors across all the subintervals of $[a, b]$, we simply add all the local errors together. Since there are $n = \frac{b-a}{h}$ subintervals, we have the general relationship that

$$\mathrm{E}(h) \approx \mathcal{O}(n\mathrm{E}_i(h)) = \mathcal{O}\left(\frac{b-a}{h}\mathrm{E}_i(h)\right) = \mathcal{O}\left(\frac{\mathrm{E}_i(h)}{h}\right) = \frac{\mathcal{O}(\mathrm{E}_i(h))}{h}$$

which means that we typically lose a power of h going from local to global errors. Notice that the global error $E(h)$ could be better than this if there happens to be cancellation of local errors.

Note: Make sure your experimental observations in question 108 matches the theoretical observations above.

6.3.3 Theoretical Error Analysis for Simpson's Rule

Let's try the same technique with Simpson's rule. However, this time we need $2n$ subintervals to begin. Suppose $S_i(x)$ is the quadratic interpolant at the nodes x_i, x_{i+1}, and x_{i+2}. Then we can write

$$E_i(h) = \int_{x_i}^{x_{i+2}} (f(x) - S_i(x))\ dx = \int_{x_i}^{x_{i+2}} \frac{f^{(3)}(\xi)}{3!}(x-x_i)(x-x_{i+1})(x-x_{i+2})\ dx$$

Now we run into a problem because the product $(x-x_i)(x-x_{i+1})(x-x_{i+2})$ is not always positive or negative over the interval $[x_i, x_{i+2}]$. In fact, because of symmetry, you can easily check that

$$\int_{x_i}^{x_i+2h} (x-x_i)(x-x_i-h)(x-x_i-2h)\ dx = 0$$

So clearly we can't use the GMVT and pull out a $f^{(3)}(\eta)$ term as last time.

To fix this issue, we will use a clever trick! Suppose we were to ask what is the cubic interpolant through four nodes x_i, x_{i+1}, x_{i+1}, and x_{i+2} (i.e. the middle node is counted twice).

Question 111. □

Explain why the third order interpolant $\widetilde{S}_i(x)$ through four nodes x_i, x_{i+1}, x_{i+1}, and x_{i+2} must be the exact same polynomial as the second order interpolant $S_i(x)$ through three nodes x_i, x_{i+1}, and x_{i+2}.

So we replace $S_i(x)$ with $\widetilde{S}_i(x)$ and rewrite

$$E_i(h) = \int_{x_i}^{x_{i+2}} (f(x) - \widetilde{S}_i(x))\ dx = \int_{x_i}^{x_{i+2}} \frac{f^{(4)}(\xi)}{4!}(x-x_i)(x-x_{i+1})(x-x_{i+1})(x-x_{i+2})\ dx$$

and voila! Suddenly the product is a function of x that is negative everywhere on the interval $[x_i, x_{i+2}]$.

Question 112. **Simpson's Rule**

Use the GMVT for integrals and a change of variables to show that for Simpson's rule $E_i(h) = \mathcal{O}(h^5)$.

6.3.4 Theoretical Error Analysis for General Newton-Cotes

We will complete this section with the following analogous result for Newton-Cotes quadrature. A proof can be found at this linked JSTOR article.

Suppose we have $n+1$ equally-spaced interpolation points

$$x_i := a + i\left(\frac{b-a}{n}\right) = a + ih \text{ for } i = 0, 1, \ldots, n,$$

where n represents the degree of the f-interpolating polynomial $p_n(x)$. If

$$\mathrm{E}(h) = \int_a^b (f(x) - p_n(x))\,\mathrm{d}x,$$

then using similar techniques as above, we can show that

- When n is odd, $\mathrm{E}(h) = \mathcal{O}(h^{n+2})$.
- When n is even, $\mathrm{E}(h) = \mathcal{O}(h^{n+3})$.

Chapter 7 | Differential Equations

The final segment of the course addresses techniques for approximating the solution of an *ordinary differential equation*, and *initial value problems* in particular.

Differential equations play the dominant role in applied mathematics. Most systems or phenomenon in nearly every corner of physics, chemistry, biology, engineering, finance, and beyond, are concerned with how a quantity changes with respect to another variable, often the time variable (we will denote this as t). So that the mathematical description of the model involves the rate of change of a variable with respect to time which can be mathematically represented using differential equations. A solution of the model produces a state of the system at certain points in time: the past, present or future.

However, most of these mathematical models of real-world situations are too complicated to have an analytical solution. For many practical problems involving nonlinearities, one cannot write down an explicit closed-form solution; however, approximate solutions to these equations are easier to obtain via numerical methods. These are often derived from the techniques of interpolation, approximation, and quadrature studied earlier in the course.

§7.1 Review of Differential Equations

A **differential equation** is any equation containing the derivative of one or more dependent variables, with respect to one or more independent variables. A solution to such an equation is a function (not a number) which, when substituted into the original equation, creates a true statement.

Example 1.55

First consider the simple, essential model problem

$$y'(t) = \lambda y(t),$$

which says that the rate of change is proportional to the quantity at any given moment. This equation shows up everywhere from radioactive decay to compound bank interest. This equation has the exact solution $y(t) = Ce^{\lambda t}$, where the constant C is determined by the initial data (t_0, y_0).

Here's a numerical example: the function $y(t) = 3e^{-0.25t}$ is a solution to the differential equation $y' = -0.25y$ with initial condition $y(0) = 3$. We can verify this by substituting $y(t)$ into the differential equation.

Note: Observe that 'solving' an equation involving a derivative will in some form require an anti-derivative, resulting in an arbitrary constant. The extra initial condition allows us to provide a value for this constant by choosing a fixed starting point. If the differential equation involves derivatives of higher order, we need multiple anti-differentiation, and as such, we need more initial conditions to find the value of each arbitrary constant.

7.1.1 Scalar Initial Value Problems

A standard scalar initial value problem takes the form

Given: $y'(t) = f(t, y(t))$, with $y(t_0) = y_0$

Determine: $y(t)$ for all $t \geq t_0$

We will not go over the analytical methods for solving a differential equation in this course, you are welcome to take a DE class to learn more on that area. Instead, let's focus on how we might be able to find a numerical approximation of the solution.

To that end, we start by discretizing the time variable and look for our solution at a discrete number of points. More precisely, we approximate our true solution at these points. The effectiveness of the algorithm will be measured by its ability to approximate such solutions on a large range of problems. What if we want to find the solution between two points for which we have a numerical solution? We will just use Interpolation!

7.1.2 Lipschitz Condition

Since, we will be approximating a continuous solution with a discrete amount of data, on a computer, we should be always concerned with round-off error. More precisely, we need to know whether small changes in the statement of the problem introduce correspondingly small changes in the solution.

The following notion of continuity is helpful for this purpose.

Definition 1.56

A function $f(t, y)$ is said to satisfy a Lipschitz condition in the variable y on a set $D \subset \mathbb{R}^2$ if there exists some constant $L > 0$ such that,

$$\left|f(t, y_2) - f(t, y_1)\right| \leq L\left|y_2 - y_1\right|, \quad (t, y_1), (t, y_2) \in D$$

The constant L is called a Lipschitz constant for f.

■ **Question 113.** □

If $\dfrac{\partial f}{\partial y}$ *is continuous at a given point, then show that* f *is Lipschitz in* y *in the neighbourhood.*

7.1.3 Existence and Uniqueness Theorem

We are now ready to describe a class of initial-value problems that can be solved numerically.

Theorem 1.57: Picard's Theorem

Let $D = [t_0, T] \times [y_0 - C, y_0 + C]$ *for some* $\varepsilon > 0$, *and let* $f(t, y)$ *be continuous on* D. *Additionally, suppose* f *is bounded above by some constant* K *and* f *is Lipschitz in* y *on the set* D *with Lipschitz constant* L.

Then for sufficiently large value of C *(that depends on* K *and* L*), the initial value problem has a* ***unique*** *solution* $y(t) \in C^1[t_0, T]$ *with* $|y(t) - y_0| \leq C$ *for all* $t \in [t_0, T]$.

Note: In particular, if $C = \infty$, and the functions $f(t,y)$ and $\dfrac{\partial f}{\partial y}$ are continuous, then the IVP has a unique solution.

7.1.4 Well-posed Problems

Once we have answered the question of Existence and Uniqueness, we still need to determine whether small changes in the statement of a particular problem will induce small changes in the solution. The following definition characterizes problems for which numerical solution is feasible.

Definition 1.58

The initial value problem is said to be **well-posed** if a unique solution $y(t)$ exists, and depends continuously on the data y_0 and $f(t,y)$.

Fortunately, Picard's theorem further guarantees that the IVP is well-posed.

Theorem 1.59

Assuming the same conditions as the last theorem, for any $\varepsilon > 0$, *there exists a positive constant* $k(\varepsilon)$, *such that whenever* $|\varepsilon_0| < \varepsilon$ *and* $\delta(t)$ *is continuous with* $|\delta(t)| < \varepsilon$ *on* $[t_0, T]$, *a unique solution,* $z(t)$ *to*

$$\frac{dz}{dt} = f(t,z) + \delta(t), \quad t_0 \leq t \leq T, z(t_0) = y_0 + \varepsilon_0$$

(which is the perturbed problem with errors $\delta(t)$ *and* ε_0*) exists with*

$$|z(t) - y(t)| < k(\varepsilon)\varepsilon,$$

for all $a \leq t \leq b$.

§7.2 One-step Methods

Consider an IVP of the form

$$y'(t) = f(t, y(t)), \qquad \text{with } y(t_0) = y_0$$

Then by the fundamental theorem of Calculus,

$$y(t) = y(t_0) + \int_{t_0}^{t} y' \, \mathrm{d}t = y(t_0) + \int_{t_0}^{t} f(\tau, y(\tau)) \, \mathrm{d}\tau$$

Instead of trying to compute the value of $y(t)$ for all $t \in [t_0, \mathrm{T}]$, we will start by we choosing a set of evenly spaced points in time

$$t_0 < t_1 < t_2 < \ldots < t_{\mathrm{N}} = \mathrm{T}$$

where $t_{k+1} - t_k = h = \dfrac{\mathrm{T} - t_0}{\mathrm{N}}$, and $t_k = t_0 + kh$. We will compute $y(t)$ only at the points $t = t_k$. So we are essentially interested in finding algorithms that starts from $y_k = y(t_k)$ and finds

$$y(t_{k+1}) = y(t_k + h) = y(t_k) + \int_{t_k}^{t_k+h} f(t, y) \, \mathrm{d}t$$

From the last chapter, we know a couple of different ways of numerically approximating the last integral.

7.2.1 Taylor Series Methods

Since we know the function $y(t)$ at one input t_k and we know its derivative $y'(t)$ everywhere, we could consider its Taylor series at the known point t_k:

$$y(t) = y(t_k) + (t - t_k) y'(t_k) + \frac{(t - t_k)^2}{2} y''(c)$$

But the differential equation tells us that $y'(t_k) = f(t_k, y_k)$ so that:

$$y(t) = y(t_k) + (t - t_k) f(t_k, y_k) + \frac{(t - t_k)^2}{2} y''(c)$$

Therefore, we can compute $y(t_k + h) = y(t_{k+1})$ from:

$$y(t_{k+1}) = y(t_k) + \underbrace{(t_{k+1} - t_k)}_{=h} f(t_k, y_k) = y(t_k) + hf(t_k, y_k)$$

with a local truncation error of $\mathcal{O}(h^2)$. This is **Euler's method.**

The general formula for Euler's method

$$y_{k+1} = y_k + f(t_i, y_i)h \quad \text{and } t_{k+1} = t_i + h$$

produces a list of pairs (t_k, y_k) that approximate points on the graph of the exact solution $y(t)$.

Question 114.

Write code to implement Euler's method for initial value problems. Your function should accept as input a function $f(t,y)$, an initial condition, a start time, an end time, and the value of $h = \Delta t$. The output should be vectors for t and y that you can easily plot to show the numerical solution.

Note that the overall error for Euler's method is $\mathcal{O}(h)$, however, errors build up at each step, growing exponentially. As such, Eulers' method is highly inaccurate and rarely used in practice.

Question 115.

Use Euler's method with $h = 0.25, 0.125, 0.0625$ to estimate $y(1)$ for $y' = 2t + y, y(0) = 3$ which has the exact solution $y = 5e^t - 2t - 2$. Check that the numerical error varies approximately linearly with h.

Higher-order Taylor series methods

In theory, Taylor series methods with $\mathcal{O}(h^m)$ global error can be obtained for any power of m by truncating the Taylor series farther and farther out. For instance, a second-order Taylor series method would use an update formula like

$$y(t_{k+1}) = y(t_k) + hf(t_k, y_k) + \frac{y''(t_k)}{2}h^2$$

to compute the next approximate y-value. However, in order to evaluate the term $y''(t_k)$, we need to compute the second-derivative y'' symbolically. Higher-order Taylor series methods than this would need symbolic computations of even higher-order derivativeswhich become very difficult for complicated examples. Because of this, difference-quotient approximations are typically used in place of any second-order (and higher) derivatives in the Taylor series.

$$y''(t_k) = \frac{y'(t_{k+1}) - y'(t_k)}{h} + \mathcal{O}(h) = \frac{f(t_{k+1}, y_{k+1}) - f(t_k, y_k)}{h} + \mathcal{O}(h)$$

This results in $\mathcal{O}(h^3)$ local error and $\mathcal{O}(h^2)$ global error.

Question 116.

Consider the problem of evaluating of the integral

$$y(t) = \int_0^t \frac{1}{\sqrt{1+x^3}}\,\mathrm{d}x$$

(a) Convert this problem into an IVP of the form

$$y'(t) = f(t, y(t)), \quad y(0) = 0.$$

(b) Solve the IVP by approximating $y(2)$ with 10 subdivisions using the second-order Taylor series method described above.

7.2.2 Implicit One-Step Methods

Next let's take a look at some other ways to improve Euler's method. One way to think of Euler's method is that when you are approximating the integral $\int_{t_k}^{t_k+h} f(t,y)\,\mathrm{d}t$, we are using left-endpoint Riemann sum

with a single rectangle. What happens if we approximate the quadrature instead with right-endpoint Riemann sum instead?

We get the equation

$$\boxed{y_{k+1} = y_k + hf(t_{k+1}, y_{k+1})}$$

called the **backward Euler method**. Because y_{k+1} depends on the value $f(t_{k+1}, y_{k+1})$, this scheme is called an **implicit method**; to compute y_{k+1}, one needs to solve a (generally nonlinear) system of equations (using Newton's method etc.), which is rather more involved than the simple update required for the forward Euler method.

Similarly, if we instead use the Trapezoid quadrature rule, we will get

$$\boxed{y_{k+1} = y_k + \frac{h}{2}\Big(f(t_k, y_k) + f(t_{k+1}, y_{k+1})\Big)}$$

Like the backward Euler method, the trapezoid rule is implicit. To obtain a similar explicit method, replace y_{k+1} by its approximation from the explicit Euler method:

$$\widetilde{y}_{k+1} = y_k + hf(t_k, y_k)$$

which results in

$$\boxed{y_{k+1} = y_k + \frac{h}{2}\Big(f(t_k, y_k) + f(t_{k+1}, \widetilde{y}_{k+1})\Big) = y_k + \frac{h}{2}\Big(f(t_k, y_k) + f\Big(t_k + h, y_k + hf(t_k, y_k)\Big)\Big)}$$

This is called **Heun's method** or the **improved Euler's method**.

Predictor-Corrector Methods: The above trick of turning the implicit method to an explicit method gives a class of algorithms in numerical analysis known as predictor–corrector methods. All such algorithms proceed in two steps:

- The initial, ''prediction'' step, starts from a function fitted to the function-values and derivative-values at a preceding set of points to extrapolate (''anticipate'') this function's value at a subsequent, new point.
- The next, ''corrector'' step refines the initial approximation by using the *predicted* value of the function and *another method* to interpolate that unknown function's value at the **same** subsequent point.

We will see some more examples in the next section.

7.2.3 Runge-Kutta Methods

Warning: We will switch to t_i and y_i instead of t_k and y_k for this section because the letter k will be used to denote a standard quantity. So, assume that we are trying to find y_{i+1} starting from y_i.

Let's start by restating Heun's method in a form that is more easier to read. Define

$$k_1 = f(t_i, y_i)$$
$$k_2 = f(t_i + h, y_i + hk_1)$$

So k_1 is the slope of $y(t)$ at the beginning of the interval $[t_i, t_{i+1}]$, and k_2 corresponds to the slope of the solution one would get by taking one Euler step with stepsize h starting from (t_i, y_i).

The improved Euler's method, which is also known as the classical Runge-Kutta method of order 2 (or RK2) now says

$$y_{i+1} = y_i + \frac{h}{2}\big(k_1 + k_2\big)$$

Additional function evaluations can deliver increasingly accurate explicit one-step methods, an important family of which are known as Runge–Kutta methods.

General Explicit Runge-Kutta methods

General explicit Runge-Kutta methods are of the form

$$y_{i+1} = y_i + h\sum_{j=1}^{n} b_j k_j$$

with

$$\begin{aligned} k_1 &= f\left(t_i, y_i\right) \\ k_2 &= f\left(t_i + c_2 h, y_i + h a_{21} k_1\right) \\ k_3 &= f\left(t_i + c_3 h, y_i + h a_{31} k_1 + h a_{32} k_2\right) \\ &\vdots \\ k_n &= f\left(t_i + c_n h, y_i + h\sum_{j=1}^{n-1} a_{n,j} k_j\right) \end{aligned}$$

Determination of the coefficients is rather complicated and requires multivariate Taylor series computations, which is beyond the scope of this course. We will look at some specific cases below.

To specify a particular method, one needs to provide the integer n (the number of stages), and the coefficients a_{lj}, b_l and c_l. The matrix $\mathrm{A} = [a_{lj}]$ is called the Runge–Kutta matrix, while the b_l and c_l are known as the weights and the nodes. These data are usually arranged in a mnemonic device, known as a Butcher tableau:

$$\begin{array}{c|c} \vec{c} & \mathrm{A} \\ \hline & \vec{b}^{\,\mathrm{T}} \end{array}$$

where $c_1 = 0$ and $a_{lj} = 0$ for $i \leq j$.

Question 117.

Check that the improved Euler's method above corresponds to the tableau

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}.$$

In general, the Butcher tableau for a RK2 method looks like

$$\begin{array}{c|cc} 0 & 0 & 0 \\ \alpha & \alpha & 0 \\ \hline & \left(1 - \frac{1}{2\alpha}\right) & \frac{1}{2\alpha} \end{array}.$$

The case of $\alpha = \frac{1}{2}$ is called the **Midpoint method**. The case of $\alpha = \frac{2}{3}$ is called the **Ralston method**. All of these methods require 2 stages per time step and have a local truncation error of $\mathcal{O}(h^3)$ which leads to a global error of $\mathcal{O}(h^2)$.

Classical RK4

The classical fourth order Runge-Kutta method is given by the Butcher tableau

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6}. \end{array}$$

It requires four evaluations of the function f per time step, and has a local truncation error of $\mathcal{O}(h^5)$.

Question 118.

Write down the classical RK4 in an algorithm form and write a code to implement it. Test the problem on several differential equations where you know the solution.

Here's some pseudocode to help you get started:

Code:

```
def rk4(f,x0,t0,tmax,dt):
    t = # build the time array
    x = # initialize an empty array for the x values
    x[0] = # build the initial condition
    # On the next line: be careful about how far you're looping
    for i in range( ??? ):
    # The interesting part of the code goes here.
    # Start from t[i] and x[i]. Then Define k1, k2, k3, k4
    # and use them to find x[i+1]
    return t, x

f = lambda t, x: # the definition of the function goes here

x0 = # initial condition
t0 = 0
tmax = # your choice
dt = # pick something reasonable
t, x = rk4( ??? , ??? , ??? , ??? , ??? )
# plot t vs. x
```

Question 119.

Use RK4 method with $h = 0.1, 0.05, 0.025$ to estimate $y(2)$ for the IVP

$$y'(t) = \frac{2y}{t} + t^2 e^t, \quad y(1) = 0$$

which has the exact solution $y = t^2(e^t - e)$. Find how the numerical error varies with h.

Question 120.

Consider RK4 applied to a general initial value problem of the form

$$y'(t) = f(t) \quad \text{with } y(t_0) = y_0.$$

In other words, there is no dependence of f on y. Show that in this case the classical fourth-order Runge-Kutta method corresponds to Simpson's quadrature rule.

Computational Comparison of Runge-Kutta methods

The main computational effort in applying the Runge-Kutta methods is the evaluation of f. In the second-order methods, the local truncation error is $O(h^2)$, and the cost is two function evaluations per step. The Runge-Kutta method of order four requires 4 evaluations per step, and the local truncation error is $O(h^4)$. Butcher has established the relationship between the number of evaluations per step and the order of the local truncation error shown in table 7.1. This table indicates why the methods of order less than five with smaller step size are used in preference to the higher-order methods using a larger step size.

Evaluations per step	2	3	4	$5 \le n \le 7$	$8 \le n \le 9$	$10 \le n$
Best possible local truncation error	$O(h^2)$	$O(h^3)$	$O(h^4)$	$O(h^{n-1})$	$O(h^{n-2})$	$O(h^{n-3})$

Table 7.1

One measure of comparing the lower-order Runge-Kutta methods is described as follows - the Runge-Kutta method of order four requires four evaluations per step, whereas Euler's method requires only one evaluation. Hence if the Runge-Kutta method of order four is to be superior it should give more accurate answers than Euler's method with one-fourth the step size. Similarly, if the Runge-Kutta method of order four is to be superior to the second-order Runge-Kutta methods, which require two evaluations per step, it should give more accuracy with step size h than a second-order method with step size $h/2$.

Question 121.

Approximate the value of $y(1)$ for the IVP

$$y' = y - t^2 + 1, \quad 0 \le t \le 1, \quad y(0) = 0.5$$

using Euler's method with $h = 0.025$, the Midpoint method with $h = 0.05$, and the Runge-Kutta fourth-order method with $h = 0.1$. Compare with the exact value of $y(1)$ to decide which method gives the most accurate result.

§7.3 Multistep Methods