Numerical Analysis (Math 128a)

Andrew Shi (Last Updated: 08/30/2024)

Contents

0 Chapter 0: Preface and Introduction

0.1 Preface

About These Notes

These notes were primarily written to synthesize some of the different perspectives on the material I have picked up as a result of taking this course and teaching it under various instructors at Berkeley over the course of a decade. It also serves as a place to collect many interesting and illustrative examples I wish I had been clever enough to think of myself.

These notes were also partly borne from frustrations I encountered while teaching. Students frequently complain they spend too much time in class trying to rapidly transcribe the contents from the board, which limits their ability to understand and participate in class (which I empathize with). I had hoped by providing a written record for their later reference, it could free students to participate more actively in the classroom and focus on understanding. Putting together these notes also allowed me to have a permanent and evolving record over time, which greatly reduced the amount of errors made in the classroom and helped me articulate my thoughts better.

What these notes are

These notes are primarily a reference for myself to lead discussion sections; significant work needs to be done before these can even be referred to as a complete set of *lecture* notes. The focus is more on big ideas and enlightenment with a well chosen example or two to drive home the point, which is the most I can ever hope to convey in a typical 50 minute discussion section.

What these notes are not

This is not a textbook or a primary reference. The topics of emphasis are totally uneven and far from comprehensive. The exposition is certainly riddled with typos (some egregious) and imprecise statements (some embarrassingly so). Rigor is not emphasized, and proofs and details are usually omitted with references made to more formal textbooks. These notes are in **very rough condition** and should only be used as a supplementary reference, and its contents and accuracy should be taken with a *serious grain of salt*.

0.1.1 Textbooks and Other Resources

While numerical analysis began to evolve rapidly with the advent of computers and continues to do so, the core material of this course has remain largely unchanged since the late 60s. Here are some references (most of which I cannot personally vouch for):

Most common:

- Numerical Analysis by Burden, Faires, and Burden (BFB): This is the typical textbook used at Berkeley and many other institutions. It is very large and covers more than

what could be taught in a two semester course. It is perfectly serviceable and similar in style to a calculus textbook, in the sense that each section works out an example of each type and there are many repetitive exercises with the solutions to odd-numbered ones in the back. However, the presentation can be quite dense at times, and the codes are written in a pseudocode that can be somewhat difficult to interpret. (Not to mention it is very expensive!)

- Numerical Analysis by Sauer: Covers topics in a similar fashion as Burden and Faires with a similar style of exercises. Has actual `MATLAB` codes. A useful alternative to Burden and Faires to find a corresponding, alternative explanation.

Old school references:

- Analysis of Numerical Methods by Isaacson and Keller
- Numerical Analysis: Mathematics of Scientific Computing by Kincaid and Cheney
- Introduction to Numerical Analysis by Stoer and Bulirsch
- Numerical Methods that Work by Acton

Relatively recent references:

- A Friendly Introduction to Numerical Analysis by Bradie
- Scientific Computing - An Introduction using Maple and MATLAB by Gander, Gander, and Kwok

Lecture notes:

- Prof. Masayuki Yano's lecture notes for AER336: Scientific Computing at the University of Toronto
- Prof. Leon Q. Brin lecture notes at Southern Connecticut State University
- Prof. Anthony Yeates' lecture notes for Numerical Analysis II at Durham University

0.1.2 Prerequisites

The formal prerequisites are Math 53 (Multivariable Calculus), Math 54 (Linear Algebra and Differential Equations), and basic programming skills. Here I describe in detail what I think is the necessary background for this course.

Calculus

No specific knowledge of Math 53 topics is required (besides some simple double integration), as this introductory course almost exclusively deals with functions of one variable $y = f(x)$ and ordinary differential equations $y' = f(t, y)$ (as opposed to partial differential equations). A solid grasp of all one-variable calculus topics from Math 1A is a must. Certain major

theorems (Mean Value Theorem, Intermediate Value Theorem) are reviewed at the beginning of this course. From Math 1B, basic integration techniques like integration by parts are useful. The single most important tool for this course is Taylor series expansions and polynomials and computing associated error bounds. The idea of Taylor approximation is central to many of the algorithms developed in this course. Your previous calculus courses might have already briefly mentioned some of the most basic numerical algorithms for rootfinding (Newton's method, Stewart §4.8), numerical integration (Trapezoidal rule, Stewart §7.7) and numerically solving ordinary differential equations (Euler's method, Stewart §9.3).

Differential Equations

The second to last unit of this course (§5 of BFB) spends about 4-5 weeks on the numerical solution of ordinary differential equations. This is usually the most conceptually challenging part of the course (which also spawns the most difficult programming assignments). Most institutions have a quarter or semester long course on ordinary differential equations focusing on exact solution techniques, which Berkeley splits up between Math 1B and Math 54. The ODE material covered in Math 54 is not specifically touched upon on this course. The exact solution techniques for first order ODEs (separation of variables, integrating factors, etc.) are not directly used except to obtain exact solutions for simple examples for comparison with the numerical (approximate) solution we will study. The most important takeaway is the idea of slope/direction fields and the qualitative behavior of ODEs, as these form the conceptual basis for many of our numerical methods. Unfortunately, this idea is usually lost among the "bag of tricks" learned for solving ODEs for most students. See Ten Lessons I wish I had learned before I started teaching differential equations by Gian-Carlo Rota.

Linear Algebra

The final two weeks of the course cover very basic numerical linear algebra (§6 of BFB), which is of much greater focus in Math 128B. The relevant linear algebra topics are roughly chapters 1-3 of Lay: vectors and matrices, solving systems of linear equations and row reduction, matrix algebra, inverses and determinants. These are also briefly reviewed in this course (§6.1, 6.3-6.5 of BFB). The notion of orthogonality and inner products (for both vectors and functions) also comes up. Students generally find this section to be very straightforward and a welcome respite right before final examinations.

Is this class going to have proofs? I heard it was all computation.

Of course proofs are important. This is still a math class after all, and proofs are the currency of mathematics. That being said, this is probably the first *applied* math class most of you are taking, and in particular it is a *computational* math class. The focus is on the development, analysis, and implementation of numerical algorithms to find fast and accurate solutions to basic problems in mathematics (basic meaning fundamental, not easy). It is true a significant portion of this course (and your grade) is focused on programming assignments and implementing and debugging these algorithms. The development and analysis is largely theoretical, and as such requires proof. While it is true that many homework questions will start with "compute" instead of "prove", a healthy attitude towards proofs and derivations

are important to get anything out of the course other than just a bag of tricks (not to mention a decent grade).

Computer Programming

This is a major sticking point for many students and where student background varies heavily. Students are pretty nervous about programming overall, and this is usually the main source of stress and reason students underperform in this course. All of our majors should really take programming courses, and if you could only take one it should be Math 124. If you already have programming experience in any other interpreted language (Python, R, etc.) you will find MATLAB very easy to pick up and will just need to spend an afternoon with any introductory MATLAB guide to work through the syntax.

For many semesters I have taught a Math 98 adjunct class on MATLAB where I cover the basics of MATLAB (and programming in general) needed for this class. The topics are: basic arithmetic operations, logicals, matrices and vectors, scripts and functions, control flow (if/else, for, while) and plotting. Debugging is criminally underemphasized in any programming class, from the basic philosophy (rubber duck debugging, writing good test cases, unit testing, etc.) to how to use the specific tools for your language (the MATLAB debugger is very helpful and relatively easy to use). The ability to debug successfully is what determines whether you can become an independent programmer, and also helps you avoid the packed and panicked office hours the day before a programming assignment is due.

0.1.3 Acknowledgments

I took Math 128a in Spring 2013 from Alexandre Chorin, and *it changed my life*. I took it very reluctantly, as I had just started to take upper division courses and it was the only class I could get into with my awful freshman Telebears time (MWF at 8am!). But without fail, I showed up at 7:55a every day and sat in the front of the classroom with great interest (I usually never sit in the front, that is, if I decided to show up at all). Prof. Chorin piqued my interest in the subject and was instrumental in my decision to pursue graduate studies in the field. I owe a great debt to him as do countless other students he has mentored over the years, and it is my great fortune that I could meet him at the end of his fifty year teaching career to jumpstart my own.

As a graduate student, I have had the good fortune to teach Math 128A under the supervision of Profs. John Strain, Jon Wilkening, Ming Gu, and Per-Olof Persson. I have been lucky to learn so much from them and benefit from their experience. In particular, I have frequently discussed pedagogy and teaching (among many other things) with Per-Olof Persson and have benefited from his mentorship. It has really been a great privilege to me to interact with the numerical analysts at Berkeley.

Students are often surprised to learn that their GSIs never receive any formal training in teaching before they enter the classroom. Much of our training is on the job and largely borne through our experiences in front of the classroom. There's nobody else in the classroom

besides me except for the students, so if I ever learned anything about teaching, it must have been from the students! I thank my previous 128a students, who despite being a captive audience, have largely been receptive towards my efforts in the classroom and are the reason I can justify spending time on these notes. I believe good teaching is *at least* as important as research, and as such I put a lot of effort into it and take great pride when I do a good job. I have learned a lot from my students (even when they didn't know they were teaching me anything), and I hope to continue to do so over what will hopefully be a very long career in research, teaching, and service.

0.2 Introduction

0.2.1 Course Topics

In this course, we are going to revisit some familiar problems from Math 1A, 1B and 54:

- Solution of nonlinear equations in one variable

$$f(x) = 0$$

- Interpolation and approximation theory

$$f(x) \approx P(x)$$

- Numerical differentiation

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

- Numerical integration

$$\int_a^b f(x)\ dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

- Numerical solution to ordinary differential equations (ODEs)

$$y' = f(t, y), \quad y(t_0) = y_0$$

- Numerical linear algebra

$$Ax = b$$

Numerical Approximation: We can plot a function like $f(x) = x + \cos(x) + e^x = 0$ and see it has one root. Our usual method of rootfinding to "do algebra and isolate x" does not work here. For most functions there does not exist a nice closed form for the root(s), but that doesn't mean they don't exist. Instead, we numerically approximate roots to as many decimal places as desired. Another prime example is the computation of definite integrals. The vast majority of functions (like e^{-x^2}) don't have antiderivatives in terms of elementary functions, so their definite integrals cannot be evaluated by finding the antiderivative and applying the fundamental theorem of calculus. But again, that doesn't mean these integrals don't exist, as there is clearly "area under the curve". So we approximate these numerically

– the simplest method is inspired by the definition of an integral, the Riemann sum.

Fast and accurate algorithms: We also consider how our usual algorithms behave in the presence of roundoff error due to the finite amount of storage space available on the computer. For example, given a system $Ax = b$, no matter how large, it can be solved by the row reduction methods from Math 54. But when done on a computer, each operation incurs some error which gradually compounds and can be catastrophic towards the accuracy of the final result. Another key issue when designing algorithms is speed: some algorithms converge much faster than others to the same result under different situations. This is a very important practical topic for algorithms, but of little emphasis in this course. As this is an introductory course, many of the codes you write will focus on correctness rather than speed, and the problems we deal with will be simple and small enough in scale so their implementations should run almost immediately on our personal computers.

Sources of Error

These are two major sources of error.

I. Discretization Error:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + error$$

$$\frac{du}{dx} \approx \frac{u(x+h) - u(x-h)}{2h} + error$$

This type of error comes from approximating the original problem by a discrete or truncated problem.

II. Roundoff Error:

There is no way to represent π or e exactly as a binary or decimal expansion on a computer with a finite number of digits. Error is also incurred as a result of doing computer arithmetic. Suppose you have a machine that can only store 16 digits numbers. Multiplying two 16-digit numbers gives a 32 digit number, and then you have to round the result to fit into 16 digits again.

Our course mostly focuses on discretization error, but we briefly discuss roundoff error as well (§1.2 of BFB).

0.2.2 Disasters Caused by Numerical Errors

From Douglas Arnold's page of the University of Minnesota:

Have you been paying attention in your numerical analysis or scientific computation courses? If not, it could be a costly mistake. Here are some real life examples of what can happen when numerical algorithms are not correctly applied.

The Patriot Missile failure, in Dharan, Saudi Arabia, on February 25, 1991 which resulted in 28 deaths, is ultimately attributable to poor handling of rounding errors.

The explosion of the Ariane 5 rocket just after lift-off on its maiden voyage off French Guiana, on June 4, 1996, was ultimately the consequence of a simple overflow.

The sinking of the Sleipner A offshore platform in Gandsfjorden near Stavanger, Norway, on August 23, 1991, resulted in a loss of nearly one billion dollars. It was found to be the result of inaccurate finite element analysis.

And finally, some more disasters caused by numerical errors. You should do a good job in this course, otherwise dollars and lives could be lost. I'll bet your Math 113 professor can't say the same ☺.

1 Chapter 1: Preliminaries

1.1 Basic Calculus

We review some basic definitions and theorems from calculus.

1.1.1 Basic Definitions

Definition 1.1 *A function f has the **limit** L at x_0 if for any $\epsilon > 0$, there exists $\delta > 0$ such that*

$$\forall x \text{ such that } |x - x_0| < \delta \implies |f(x) - L| < \epsilon.$$

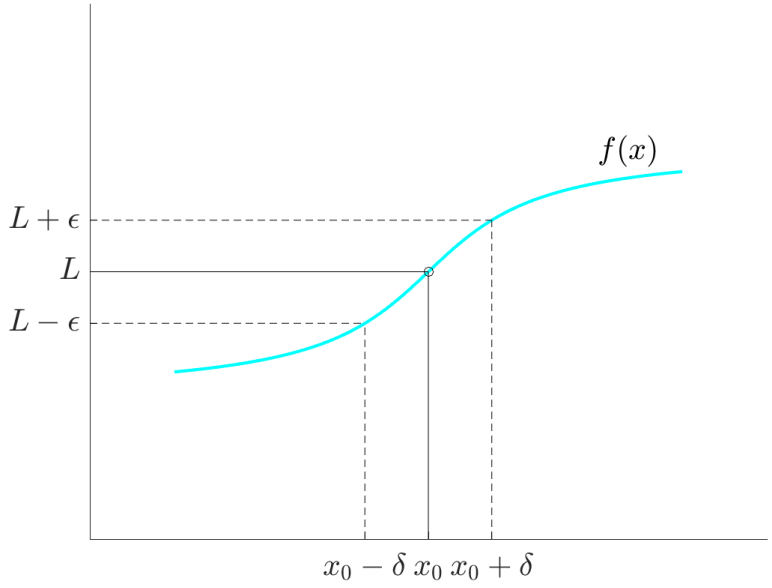


Figure 1: $\epsilon - \delta$ definition of a limit

This is a formalization of the intuitive notation that we can always get "as close as we want" to the limit L. Sometimes we write $\delta(\epsilon)$ in Definition 1.1 to emphasize that the the value of δ will depend on the value of ϵ.

Definition 1.2 *We say f is **continuous** at x_0 if*

$$\lim_{x \to x_0} f(x) = f(x_0).$$

The function f is continuous on the set X if it is continuous at each point in X. Informally, f is continuous on X if you can draw it without picking up your pencil. In this class, we usually take X to be $\mathbb{R}$ or some finite interval $[a, b]$.

Definition 1.3 *We say f is **differentiable** at x_0 if the following limit exists*

$$f'(x_0) \equiv \lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0}$$

The number $f'(x_0)$ is called the **derivative** of f at x_0. One interpretation of this number is that the derivative of f at x_0 is the slope of the tangent line to the graph of f at $(x_0, f(x_0))$.

Example 1.1: Definition of differentiability

Determine whether f is differentiable at $x = 0$ for the following functions:

$$a)\ f(x) = \begin{cases} x \sin \dfrac{1}{x} & x \neq 0 \\ 0 & x = 0 \end{cases} \qquad b)\ f(x) = \begin{cases} x^2 \sin \dfrac{1}{x} & x \neq 0 \\ 0 & x = 0 \end{cases}$$

Solution: We compute the limit from the definition:

$$\begin{aligned} \lim_{x \to 0} \frac{f(x) - f(0)}{x - 0} &= \lim_{x \to 0} \frac{x \sin \frac{1}{x}}{x} \\ &= \lim_{x \to 0} \sin \frac{1}{x} \end{aligned}$$

This limit does not exist as the function $\sin \dfrac{1}{x}$ oscillates between -1 and 1 an infinite number of times between 0 and any positive x value, no matter how small.

For b), the limit becomes $\lim\limits_{x \to 0} x \sin \dfrac{1}{x}$. Applying the squeeze theorem:

$$\begin{aligned} -1 \leq \sin \frac{1}{x} \leq 1 \implies & -x \leq x \sin \frac{1}{x} \leq x \\ \implies & \lim_{x \to 0} -x \leq \lim_{x \to 0} x \sin \frac{1}{x} \leq \lim_{x \to 0} x \\ \implies & 0 \leq \lim_{x \to 0} x \sin \frac{1}{x} \leq 0 \\ \implies & f'(0) = \lim_{x \to 0} x \sin \frac{1}{x} = 0. \end{aligned}$$

So f is in fact differentiable at 0, and $f'(0) = 0$.

1.1.2 Useful Theorems

Theorem 1.1 ***Differentiability implies continuity***
If the function f is differentiable at x_0, then f is continuous at x_0.

Proof: By assumption, the limit

$$f'(x_0) = \lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0}$$

exists. So we calculate $\lim_{x \to x_0} f(x) - f(x_0)$ hoping that it will be zero, since this would satisfy the definition of continuity of f at x_0.

$$\begin{aligned}
\lim_{x \to x_0} f(x) - f(x_0) &= \lim_{x \to x_0} f(x) - f(x_0) \left(\frac{x - x_0}{x - x_0} \right) \\
&= \lim_{x \to x_0} \left(\frac{f(x) - f(x_0)}{x - x_0} \right) (x - x_0) \\
&= f'(x_0) \cdot 0 = 0. \qquad \blacksquare
\end{aligned}$$

We commonly use the following notation:

$$\begin{aligned}
f \in C^r[a,b] &= r\text{-times continuously differentiable functions defined on the interval } [a,b] \\
&\quad (r = \infty \ OK) \\
C^0[a,b] = C[a,b] &= \text{continuous functions on } [a,b]
\end{aligned}$$

So this notation makes it clear that continuity is strictly weaker than differentiability.

As an example, the absolute value function $f(x) = |x|$ is continuous everywhere, but it is not differentiable due to the kink at $x = 0$. So we say that $f \in C^0$, but $f \notin C^1$.

While we're on the topic, it is a good exercise for the reader to formalize this notion of the "kink at $x = 0$" by repeating the calculation in the previous example by showing f is not differentiable at $x = 0$ using the limit definition. For this, you might find it useful to recall the piecewise definition of absolute value:

$$|x| = \begin{cases} x & x \geq 0 \\ -x & x < 0 \end{cases}$$

Theorem 1.2 ***Mean Value Theorem***
If $f \in C[a,b]$ and f is differentiable on (a,b) then

$$\exists c \in (a,b) \text{ such that } f'(c) = \frac{f(b) - f(a)}{b - a}.$$

The mean value of f on the interval (a,b) is $\dfrac{f(b) - f(a)}{b - a}$. The mean value theorem (Figure 2) simply says a function's derivative takes on its mean value at some point on the interval (a,b).

An interesting application (Figure 3): The California Highway Patrol can prove you were speeding without a radar gun. Your average speed on the interval is given by

$$\text{average speed } = \frac{x_2 - x_1}{t_2 - t_1}$$

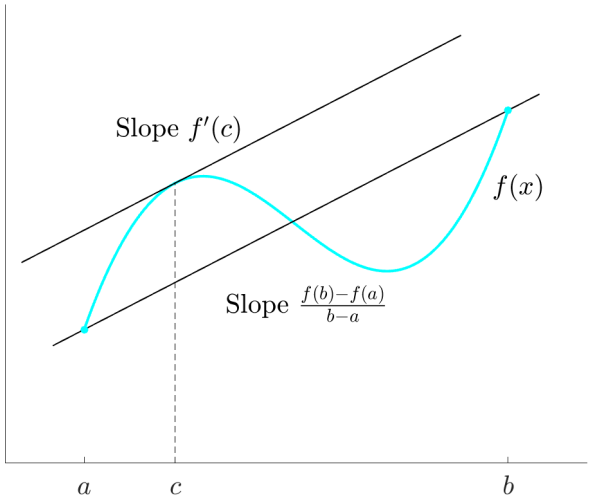


Figure 2: The mean value theorem

If you show up at x_2 sooner than

$$t_2 - t_1 < \frac{x_2 - x_1}{\text{speed limit}}$$

then

$$\text{speed limit} < \frac{x_2 - x_1}{t_2 - t_1} = \text{ average speed } = \text{ instantaneous speed at some } t_3 \in (t_1, t_2)$$

and you're busted (Figure 3)!

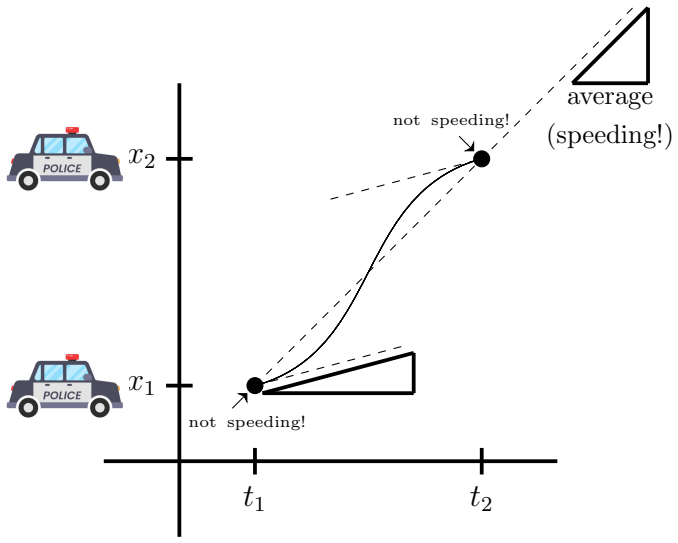


Figure 3: Speeding on the highway

Figure 4: The mean value theorem on a bridge in Beijing

Theorem 1.3 ***Rolle's Theorem***
Suppose $f \in C[a,b]$ *and* f *is differentiable on* (a,b) *If* $f(a) = f(b)$*, then* $\exists c \in (a,b)$ *such that* $f'(c) = 0$*.*

This is merely a special case of the mean value theorem when $a = b$.

Theorem 1.4 ***Intermediate Value Theorem***
If $f \in C[a,b]$ *and* K *is in between* $f(a)$ *and* $f(b)$*, then* $\exists c \in (a,b)$ *s.t.* $f(c) = K$*.*

In words, this just says that a continuous function on $[a,b]$ must attain all intermediate values between $f(a)$ and $f(b)$. We are frequently interested in the special case where $K = 0$, which would show that f has a root on the interval $[a,b]$.

Example 1.2: Application of the IVT

Consider the function $f(x) = x^3 + 2x + k$.

a. Show that $f(x)$ has at least one root, regardless of k.

b. Show that $f(x)$ has exactly one root, regardless of k.

Solution: Note that

$$\lim_{x\to\infty} f(x) = \infty \qquad \lim_{x\to-\infty} f(x) = -\infty$$

and by the Intermediate Value Theorem, since f is continuous there exists at least zero of f. Since $f'(x) = 3x^2 + 2 > 0$, the function is strictly increasing so there is only exactly one zero.

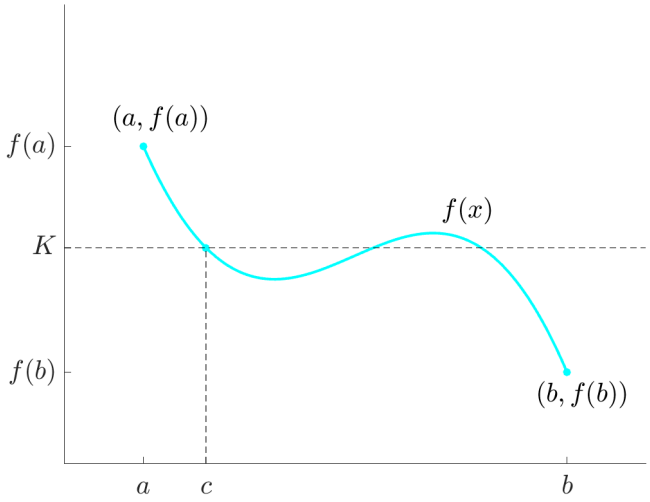


Figure 5: Intermediate Value Theorem

Here we get off easy because f happens to be increasing everywhere. In general, one would show this via Rolle's theorem as follows.

We know from a) there exists at least one root. Assume for the sake of contradiction there exists two distinct roots $x_1 \neq x_2$ such that $f(x_1) = f(x_2) = 0$. Then Rolle's Theorem says $\exists c \in (x_1, x_2)$ such that $f'(c) = 0$. But since $f'(x) = 3x^2 + 2 > 0$, this c cannot exist. So we have arrived at a contradiction and there cannot exist two distinct roots. So f has exactly one root.

Here is an interesting "real-life" application of the intermediate value theorem.

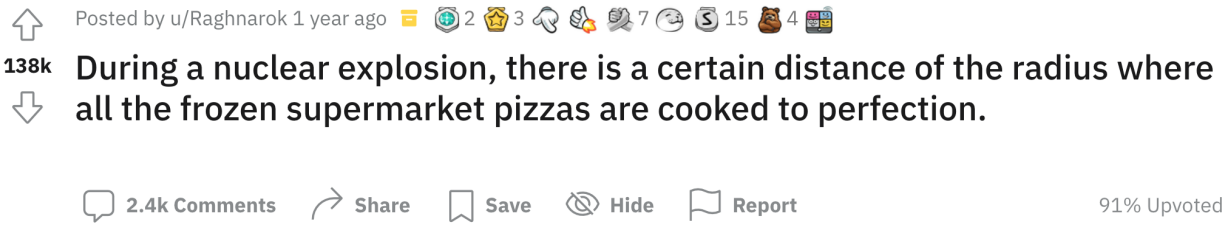


Figure 6: IVT in the real world

Theorem 1.5 ***Extreme Value Theorem***
If $f \in C[a,b]$ then $\exists\ c_1, c_2 \in [a,b]$ such that

$$\underbrace{f(c_1)}_{min} \leq f(x) \leq \underbrace{f(c_2)}_{max} \qquad \forall x \in [a,b].$$

So $f(x)$ attains its minimum and maximum on $[a,b]$ if it is continuous. If, in addition, f is differentiable on (a,b), then c_1 and c_2 occur either at the endpoints of $[a,b]$ or where $f'(x) = 0$.

So this theorem suggests an algorithm: To find the extrema of a function f which is continuous and piecewise differentiable on (a,b), we simply need to compare the values of $f(x)$ at

- the endpoints $x = a, x = b$ and at
- the critical points $x = c$ where $f'(c) = 0$ or $f'(c)$ does not exist.

Example 1.3: Finding extrema by the EVT

Find the extrema of $f(x) = xe^{-x}$ on $[0,2]$.

Solution: We find the critical points

$$f'(x) = e^{-x} - xe^{-x} = e^{-x}(1-x) = 0 \implies x = 1$$

So we test the points $x = 0, 1, 2$.

$$\begin{aligned} x = 0 &\implies f(0) = 0 \\ x = 1 &\implies f(1) = e^{-1} \\ x = 2 &\implies f(2) = 2e^{-2} \end{aligned}$$

So the local minimum is $(0,0)$ and the local maximum is $(1, e^{-1})$.

1.2 Taylor Series

1.2.1 Definition and Examples

Definition 1.4 *A **Taylor series** is a representation of a function as an infinite sum of terms that are calculated from the values of the function's derivatives at a single point.*

More concretely, given a function $f(x)$ and a real number x_0, we have that the Taylor series of $f(x)$ at x_0 is given by:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!}(x-x_0)^n = f(x_0)+f'(x_0)(x-x_0)+\frac{f''(x_0)}{2!}(x-x_0)^2+\frac{f^{(3)}(x_0)}{3!}(x-x_0)^3+\cdots$$

Definition 1.5 *The **Maclaurin series of** $f(x)$ is a special case of a Taylor series where $x_0 = 0$.*

Here are some common ones, which you should verify yourself directly from the definition.

$$\begin{aligned}
e^x &= \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots \\
\frac{1}{1-x} &= \sum_{n=0}^{\infty} x^n = 1 + x + x^2 + x^3 + \cdots \\
\sin(x) &= \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots \\
\cos(x) &= \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots
\end{aligned}$$

Comment 1: The Taylor series of $\frac{1}{1-x}$ should remind you of the formula for an infinite geometric sum, which only holds for $|x| < 1$.

Comment 2: From the Taylor series representations of $\sin(x)$ and $\cos(x)$, you can confirm that $\sin(0) = 0, \cos(0) = 1$ and that $\frac{d}{dx}\sin(x) = \cos(x)$, $\frac{d}{dx}\cos(x) = -\sin(x)$.

We are almost always interested in Maclaurin series, so much so that I will usually use the term Taylor series to mean the Maclaurin series. But sometimes we *have to* use a nonzero expansion point. For example, $f(x) = \ln(x)$ is undefined at $x = 0$. Here is the Taylor expansion of $f(x) = \ln(x)$ around $x_0 = 1$.

$$\ln(x) = \sum_{n=1}^{\infty} (-1)^{(n-1)} \frac{(x-1)^n}{n} = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \frac{(x-1)^4}{4} + \cdots$$

Building New Taylor Series from Old Ones

Say you want the Taylor expansion of the following functions:

$$f_1(x) = x^2 e^x \quad f_2(x) = e^{x^2} \quad f_3(x) = \arctan(x) \quad f_4(x) = \frac{e^x - 1}{x}$$

You could proceed directly with the definition and start taking derivatives, but this gets messy quickly. Or you could build these from more elementary Taylor series.

Multiplication and Division

Let's start with the Taylor series for e^x:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

Simply multiply by x^2:

$$f_1(x) = x^2 e^x = \sum_{n=0}^{\infty} \frac{x^{n+2}}{n!} = x^2 + x^3 + \frac{x^4}{2!} + \frac{x^5}{3!} + \cdots$$

See the example at the end of this section for a demonstration of division.

Substitution

Let's start with the Taylor series for e^x:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

Replace x by x^2 to get:

$$f_2(x) = e^{x^2} = \sum_{n=0}^{\infty} \frac{(x^2)^n}{n!} = 1 + x^2 + \frac{x^4}{2!} + \frac{x^6}{3!} + \cdots$$

Term by Term Differentiation and Multiplication

Recall that $\frac{d}{dx} \arctan(x) = \frac{1}{1+x^2}$. So we can start with the well known Taylor series for $\frac{1}{1-x}$:

$$\frac{1}{1-x} = \sum_{n=0}^{\infty} x^n = 1 + x + x^2 + x^3 + \cdots$$

Replace x by $-x^2$ to get the Taylor series for $\frac{1}{1+x^2}$:

$$\frac{1}{1+x^2} = \sum_{n=0}^{\infty} (-x^2)^n = 1 - x^2 + x^4 - x^6 + \cdots$$

Then integrate term by term to get the Taylor series for $\arctan(x)$:

$$f_3(x) = \arctan x = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{2n+1} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots$$

Only Expanding Part of the Function

Here we could just substitute in the Taylor series for e^x into the numerator instead of taking derivatives of the whole expression.

$$\begin{aligned} f_4(x) = \frac{e^x - 1}{x} &= \frac{1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots - 1}{x} \\ &= 1 + \frac{x}{2!} + \frac{x^2}{3!} + \frac{x^3}{4!} + \cdots \\ &= \sum_{n=0}^{\infty} \frac{x^n}{(n+1)!} \end{aligned}$$

Moral of the story: When it comes to potentially taking nasty derivatives, work smarter not harder. See if you can directly work with more elementary series.

Example 1.4: Taylor Series by Division

Find the first few terms of the Taylor expansion of $\tan(x)$:

a. directly by the definition.

b. by division of the series of $\sin(x)$ by $\cos(x)$.

Solution: These both seem equally unpleasant in different ways. Directly using the definition gives us:

$$\begin{aligned} f(x) = \tan(x) &\implies f(0) = 0 \implies a_0 = 0 \\ f'(x) = \sec^2(x) &\implies f'(0) = 1 \implies a_1 = \frac{1}{1!} = 1 \\ f''(x) = 2\sec^2(x)\tan(x) &\implies f''(0) = 0 \implies a_2 = 0 \\ f^{(3)}(x) = 2\sec^2(x)(2\tan^2(x) + \sec^2(x)) &\implies f^{(3)}(0) = 2 \implies a_3 = \frac{2}{3!} = \frac{1}{3} \\ &\vdots \end{aligned}$$

This will work out for as long as we want it to, but it will get nastier and nastier due to the differentiation of the subsequent terms become more and more complex. Now we try long division. Since:

$$\begin{aligned} \sin(x) &= x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots \\ \cos(x) &= 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots \end{aligned}$$

Long division goes as follows:

$$
\begin{array}{l|l}
 & x + \left(\frac{1}{2!} - \frac{1}{3!}\right) x^3 + [\frac{1}{2!}(\frac{1}{2!} - \frac{1}{3!}) + (\frac{1}{5!} - \frac{1}{4!})]x^5 + \dots \\
1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots & \;)x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots \\
 & \;\;\underline{x - \frac{x^3}{2!} + \frac{x^5}{4!} - \frac{x^7}{6!} + \cdots} \\
 & \;\;\left(\frac{1}{2!} - \frac{1}{3!}\right) x^3 + \left(\frac{1}{5!} - \frac{1}{4!}\right) x^5 + \left(\frac{1}{6!} - \frac{1}{7!}\right) x^7 + \cdots \\
 & \;\;\underline{\left(\frac{1}{2!} - \frac{1}{3!}\right) x^3 - \frac{1}{2!}\left(\frac{1}{2!} - \frac{1}{3!}\right) x^5 + \frac{1}{4!}\left(\frac{1}{2!} - \frac{1}{7!}\right) x^7 + \dots} \\
 & \;\;[\frac{1}{2!}(\frac{1}{2!} - \frac{1}{3!}) + (\frac{1}{5!} - \frac{1}{4!})]x^5 + [\left(\frac{1}{6!} - \frac{1}{7!}\right) - \frac{1}{4!}\left(\frac{1}{2!} - \frac{1}{7!}\right)]x^7 + \dots
\end{array}
$$

One advantage of long division in this case (so I claim) is that with a little effort in expanding to another term, you might be able to see the pattern to determine the general term a_n.

1.2.2 Some Properties and Technical Details

Where does this definition come from? A brief non-rigorous justification

Suppose you believe some function $f(x)$ can be represented as an infinite sum of monomials in the form:

$$f(x) = \sum_{n=0}^{\infty} a_n x^n = a_0 + a_1(x) + a_2(x^2) + a_3(x^3) + a_4(x^4) + \cdots$$

Now we need to find these coefficients. Plugging in $x = 0$ tells us that $a_0 = f(0)$.

Take the first derivative:

$$f'(x) = a_1 + 2a_2(x) + 3a_3(x^2) + 4a_4(x^3) + \cdots$$

and plug in $x = 0$ again to get $a_1 = f'(0)$.

Take the second derivative:

$$f''(x) = 2a_2 + 3 \cdot 2a_3(x) + 4 \cdot 3a_4(x^2) + 5 \cdot 4a_5(x^3) + \cdots$$

and plug in $x = 0$ again to get $a_2 = f''(0)/2$.

Take the third derivative:

$$f'''(x) = 3 \cdot 2a_3 + 4 \cdot 3 \cdot 2a_4(x) + 5 \cdot 4 \cdot 3a_5(x^2) + \cdots$$

and plug in $x = 0$ to get $a_3 = f'''(0)/3!$.

It is not hard to see that in general $a_n = \dfrac{f^{(n)}(0)}{n!}$, and more generally for the Taylor series centered around x_0, that $a_n = \dfrac{f^{(n)}(x_0)}{n!}$. What this shows is that *if* the monomial expansion of $f(x)$ exists, the coefficients must be given by the definition provided for Taylor series.

The Taylor expansion of a function is unique.

If $f(x) = \sum_{n=0}^{\infty} a_n x^n$, the choice of coefficients $\{a_n\}_{n=0}^{\infty}$ is unique. The proof is a standard one for uniqueness where you assume that $f(x) = \sum_{n=0}^{\infty} a_n x^n = \sum_{n=0}^{\infty} b_n x^n$, subtract the two series, and conclude that $a_n = b_n \ \forall n$, hence uniqueness.

Differentiability/Smoothness

In this class, we generally assume that $f(x)$ is sufficiently "smooth" and that we can take as many derivatives as we need.

How are a function and its Taylor series related?

You would hope that a function and its Taylor series always agree, but this is not always true. A real valued function that is equal to its Taylor series in an open interval around x_0 is known as an *analytic function* in that interval. In this class all of our functions are analytic, which means we don't have to worry about this. (This is discussed more in Math 185).

In particular, I want to know for what x can we trust that the function and its Taylor series are equal. For example, as mentioned earlier the Taylor series for $\frac{1}{1-x}$ only converges for $|x| < 1$. This is the **radius of convergence** of a Taylor series.

We won't worry too much about these things, but we do have to be careful in some cases. For the most part, we proceed recklessly.

1.2.3 Taylor Polynomials and Error Bounds

We are usually not interested in working with the infinite series, but rather its truncation after a finite number of terms. This is what we call a **Taylor polynomial**. Polynomials are a powerful tool when it comes to approximating arbitrary functions and are much easier to work with.

Let the nth Taylor polynomial $P_n(x)$ of $f(x)$ be the truncation of its Taylor series up to and including the nth degree. For example, for $f(x) = e^x$, $P_3(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!}$.

Warning 1: Don't get sloppy and write $f(x) = P_n(x)$. Either $f(x) \approx P_n(x)$ or $f(x) = P_n(x) + \ldots$.

Warning 2: Don't make the mistake that the Taylor series is a polynomial. Any finite truncation or subset of terms of the infinite series is a polynomial, but the entire series is not.

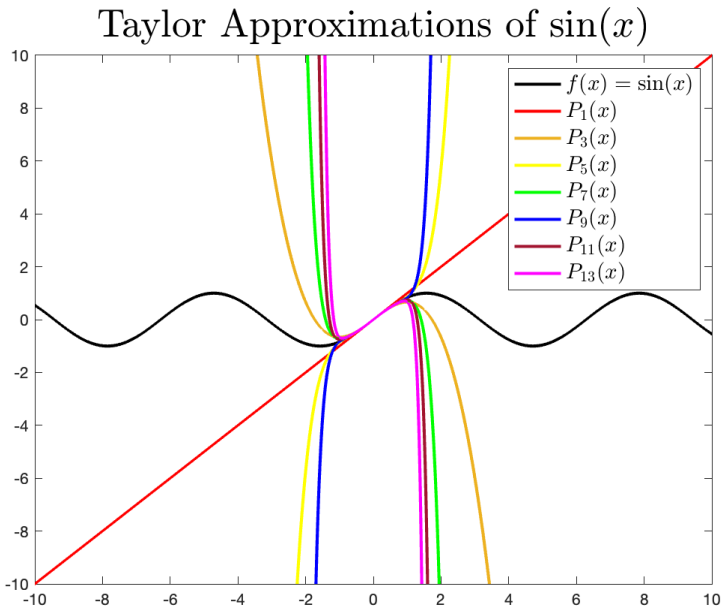


Figure 7: Taylor approximations of $\sin(x)$. Note that the approximation gets better as the degree improves around the expansion point.

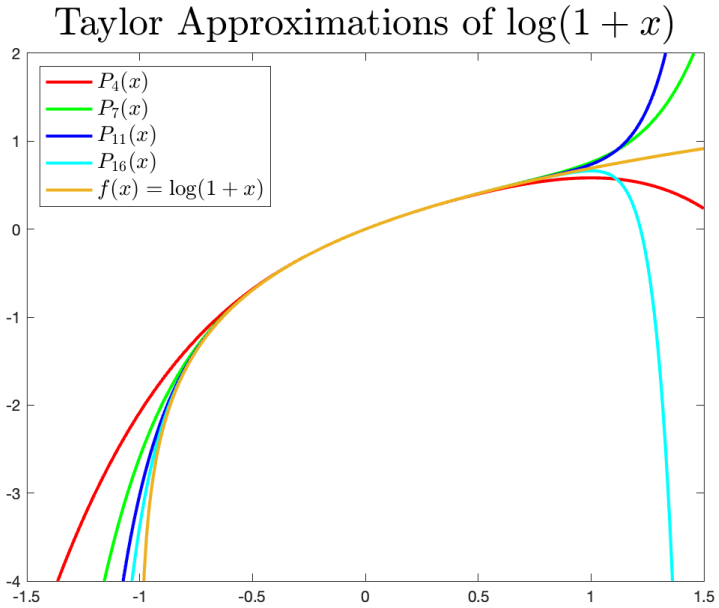


Figure 8: Taylor approximations of $\log(1+x)$. It is clear that outside the radius of convergence $|x| < 1$, the Taylor approximations diverge.

Remainder Term

So now we have $P_n(x)$, which is easier to work with than $f(x)$ and we hope still closely approximates $f(x)$. But in the process, we have discarded an infinite number (!!) of terms.

As budding numerical analysts, every time we make an approximation (in this case, by truncation) like this, we must ask ourselves *how good it is*.

We have $f(x) = P_n(x) + R_n(x)$, where the **remainder** $R_n(x)$ is defined by $R_n(x) \equiv f(x) - P_n(x)$. Note that the remainder is a function of x. By the mean value theorem (proof omitted), we have an expression for

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

where $\xi \in [x, x_0]$. This is saying for each point x, there is a $\xi \in [x, x_0]$ that makes definition hold with equality. But since we know nothing more about ξ, we generally try to maximize this expression for ξ on a given range of x to find an upper bound on the error.

Example 1.5: Computing Taylor Polynomial

Consider $f(x) = \frac{1}{x}$ and $x_0 = 2$. Then

$$P_3(x) = \frac{1}{2} - \frac{1}{4}(x-2) + \frac{1}{8}(x-2)^2 - \frac{1}{16}(x-2)^3.$$

Find an upper bound on the remainder on $[1, 3]$.

$$\max_{x\in[1,3]} |f(x) - P_3(x)| = \max_{x\in[1,3]} |R_3(x)|$$

Solution: Compute $R_3(x)$. Since $f^{(4)}(x) = \dfrac{24}{x^5}$,

$$R_3(x) = \frac{f^{(4)}(\xi)}{4!}(x-2)^4 = \frac{1}{\xi^5}(x-2)^4$$

Now we seek to maximize this quantity. We want to maximize over ξ and x, both of which live in $[1, 3]$ and are independent of each other. This quantity is maximized when $\xi = 1$ and $x = 3$, so $|R_5(x)| \leq 1$.

Example 1.6: Error Bound of Taylor Polynomials

Suppose you approximate $f(x) = \sin(x)$ by $P_3(x)$ around $x_0 = 0$. Find an upper bound on the error in estimating $f(3)$ by $P_3(3)$.

Solution: Since $P_3(x) = x - \frac{x^3}{3!}$,

$$R_3(x) = \frac{\sin(\xi)}{4!}x^4 \implies |R_3(x)| \leq \frac{1}{4!}3^4$$

using the fact that sin is always bounded by 1. But wait! It just so happens that $P_4(x) = P_3(x)$ here. So using the remainder term $R_4(x)$ gives us $|R_4(x)| \leq \frac{1}{5!}3^5$, which is a tighter bound.

1.2.4 Applications: Taylor in Action

In this section, we discuss a variety of ways to apply Taylor series and polynomials to familiar concepts from Calculus and other examples that have more of a numerical analysis bent.

Application 1: Computing Limits

Example 1.7: Limits with Taylor Series I

Evaluate $\lim_{x\to 0} \dfrac{\sin x - x}{x^3}$.

Solution: We plug in the Taylor series for $\sin x$:

$$\begin{aligned}\lim_{x\to 0} \frac{\sin x - x}{x^3} &= \lim_{x\to 0} \frac{\left(x - \frac{1}{3}x^3 + \frac{1}{5}x^5 - \frac{1}{7}x^7 + \dots\right) - x}{x^3} \\ &= \lim_{x\to 0} \frac{-\frac{1}{3!}x^3 + \frac{1}{5!}x^5 - \frac{1}{7!}x^7 + \dots}{x^3} \\ &= \lim_{x\to 0} -\frac{1}{3!} + \frac{1}{5!}x^2 - \frac{1}{7}x^4 + \dots = -\frac{1}{6}\end{aligned}$$

Your first instinct might have been to use L'Hopital's Rule, which would work fine. But that would be pretty messy for this next one...

Example 1.8: Limits with Taylor Series II

Evaluate $\lim_{x\to 0} \dfrac{\cos x - 1 + \frac{1}{2}x\sin x}{\ln(1+x)^4}$.

Solution: We plug in the Taylor series for $\sin x, \cos x, \ln(1+x)$:

$$\begin{aligned}\lim_{x\to 0} \frac{\cos x - 1 + \frac{1}{2}x\sin x}{\ln(1+x)^4} &= \lim_{x\to 0} \frac{[1 - \frac{1}{2}x^2 + \frac{1}{4!}x^4 + O(x^6)] - 1 + \frac{1}{2}x[x - \frac{1}{3!}x^3 + O(x^5)]}{[x + O(x^2)]^4} \\ &= \lim_{x\to 0} \frac{(\frac{1}{4!} - \frac{1}{2\times 3!})x^4 + O(x^6)}{[x + xO(x)]^4} \\ &= \lim_{x\to 0} \frac{(\frac{1}{4!} - \frac{1}{2\times 3!})x^4 + O(x^6)}{x^4[1 + O(x)]^4} \\ &= \lim_{x\to 0} \frac{(\frac{1}{4!} - \frac{1}{2\times 3!}) + O(x^2)}{[1 + O(x)]^4} = \frac{1}{4!} - \frac{1}{2\times 3!} = -\frac{1}{4!}\end{aligned}$$

Application 2: Evaluating Infinite Series

It is possible to use Taylor series to find the sums of certain infinite series.

Example 1.9: Infinite Sums by Taylor Series I

Find the sum of the following series:

$$\sum_{n=0}^{\infty} \frac{1}{n!} = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \dots.$$

Solution: Recall the Taylor series for e^x:

$$1 + \frac{1}{1!}x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \frac{1}{4!}x^4 + \dots = e^x.$$

The sum of the given series can be obtained by substituting in $x = 1$:

$$1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \dots = e.$$

In this example, note that we get a different series for every value of x that we plug in. For example,

$$1 + \frac{2}{1!} + \frac{2^2}{2!} + \frac{2^3}{3!} + \frac{2^4}{4!} + \dots = e^2.$$

and

$$1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \frac{1}{4!} - \dots = \sum_{n=0}^{\infty} \frac{(-1)^n}{n!} = e^{-1}$$

Admittedly the successful usage of this technique relies on a familiarity with the form of commonly encountered Taylor series and their variations, and more important, the good will of the instructor writing the problem so that the series can actually be easily summed by this elementary method!

Example 1.10: Infinite Sums by Taylor Series I

Find the sum of the following series:

$$(a) \quad 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \dots \qquad (b) \quad 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots$$

Solution: (a) Recall that

$$x - \frac{x}{2} + \frac{x}{3} - \frac{x}{4} + \frac{x}{5} - \dots = \ln(1 + x).$$

Substituting in $x = 1$ yields

$$x - \frac{x}{2} + \frac{x}{3} - \frac{x}{4} + \frac{x}{5} - \dots = \ln(2).$$

(b) Recall that

$$x - \frac{x}{3} + \frac{x}{5} - \frac{x}{7} + \frac{x}{9} - \dots = \tan^{-1}(x).$$

Substituting in $x = 1$ yields

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \tan^{-1}(x) = \frac{\pi}{4}.$$

This is known as the Gregory-Leibniz formula for computing π.

You might have studied something of a similar flavor (but more difficult) in the final week of Math 54 when Fourier Series were discussed. I once gave a Math 54 quiz that asked you to show the Fourier series of $f(x) = x^2$ on $-\pi < x < \pi$ is

$$f(x) \sim \frac{\pi^2}{3} + 4\sum_{n=1}^{\infty} \frac{(-1)^n}{n^2} \cos(nx)$$

and plugging in $x = 0$ and $x = \pi$ will allow you to show

$$\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n^2} = \frac{\pi^2}{12} \qquad \sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}.$$

One of the main motivations in this class is that it is too computationally expensive or even impossible to find an *exact* solution to the *exact* problem. So we have to settle for approximations.

There are two approaches:

1. We can find an approximate solution to the exact problem.
2. We can find an exact solution to an approximate problem.

Taylor series approaches not only give an approximation to the solution, but also an idea of how good it is via the error bound. This will be a recurring theme in this class.

Application 3: How much do we need to get "close enough"?

We are often concerned with doing the least amount of (computational) work to get a desired level of accuracy. This is a very "engineering" type of concern.

Example 1.11: Sufficient n for given accuracy

Let $f(x) = e^x, x_0 = 0$. Find n such that $|R_n(x)| \leq 10^{-6}$ for $x \in [0, \frac{1}{2}]$.

Solution: For $f(x) = e^x$,

$$R_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}x^{n+1} = \frac{e^{\xi}}{(n+1)!}x^{n+1}.$$

We know $\xi \in [0, \frac{1}{2}]$, and since e^x is monotonically increasing, this tells us $1 = e^0 \leq e^{\xi} \leq e^{\frac{1}{2}}$. This combined with the fact that $x \in [0, \frac{1}{2}]$ gives us that

$$|R_n(x)| \leq \frac{e^{1/2}}{(n+1)!}\left(\frac{1}{2}\right)^{(n+1)}.$$

Notice this upper bound is now independent of x because it holds for all $x \in [0, \frac{1}{2}]$. Now use a calculator to finish this problem. Make a table for some n:

n	max error
5	7.16×10^{-5}
6	5.11×10^{-6}
7	3.19×10^{-7}

So $n = 7$ is needed to achieve $|R_n(x)| \leq 10^{-6}$.

Comment: The bound gets simpler, but weaker the further right you move in x. Think about why this makes sense (remember our expansion point was $x_0 = 0$).

Application 4: Approximate the problem and solve exactly

Example 1.12: Upper bound for integration error

Estimate $\int_0^{0.2} e^{-x^2}\, dx$ by approximating e^{-x^2} with $P_5(x)$. Find an upper bound for the error $\epsilon = \int_0^{0.2} R_5(x)\, dx$.

Solution: The Taylor polynomial gives:

$$P_5(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!}$$

so

$$\int_0^{0.2} e^{-x^2}\, dx \approx \int_0^{0.2} P_5(x)\, dx = 0.2 - \frac{0.2^3}{3!} + \frac{0.2^5}{5!} \approx 0.197365333.$$

What about the remainder? $R_5(x) = \frac{f^{(5)}(\xi)}{5!}x^5$. But I don't want to take these derivatives! So I use an old trick and substitute in an easier Taylor Series:

$$g(y) \equiv e^y = 1 + y + \frac{y^2}{2} + \tilde{R}_2(y), \quad \tilde{R}_2(y) = \frac{e^\xi}{3!}y^3$$

$$f(x) \equiv e^{-x^2} = g(-x^2) = \underbrace{1 - x^2 + \frac{x^4}{2}}_{P_5(x)} \underbrace{- \frac{e^\xi}{6}x^6}_{R_5(x)}$$

Since $\xi \in [0, y] = [0, -x^2]$ where $x \in [0, 0.2]$, we have that $\xi \in [-0.04, 0]$. So

$$|R_5(x)| = \left|\frac{e^\xi}{6}x^6\right| \leq \frac{x^6}{6}.$$

This gives us that

$$|\epsilon| \leq \int_0^{0.2} |R_5(x)|\, dx \leq \int_0^{0.2} \frac{x^6}{6}\, dx = \frac{1}{6}(0.2)^7 = 2.1333 \times 10^{-6}.$$

1.3 Floating Point Arithmetic

1.3.1 Conversion Between Bases

We can convert numbers from our usual decimal (base 10) representation to any other integer base, such as binary (base 2).

Figure 9: S4E8 of Futurama.

Example 1.13: Binary to Decimal

Convert $-1.0110_2 \times 2^5$ from binary to decimal (base 10).

Solution: To illustrate, a number in our familiar decimal (base 10) representation is interpreted as the following:

$$4.27 \times 10^3 \text{ means } (4 \times 10^0 + 2 \times 10^{-1} + 7 \times 10^{-2}) \times 10^3$$

Accordingly, the binary (base 2) conversion is:

$$-1.0110_2 \times 2^5 \text{ means } -(1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4}) \times 2^5 = 44$$

In binary, each 0 or 1 is referred to as a **bit**, which also frequently represents off and on. Binary is very useful in the underlying hardware of a computer. Some other common representations are octal (base 8) and hexadecimal (base 16). In hexadecimal, we use the letters $A - F$ in place of the decimal numbers $10 - 15$:

Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

So a single hexadecimal digit can show 16 different values instead of the normal 10.

Example 1.14: Conversion between bases

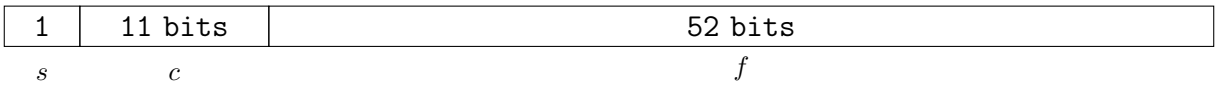
Verify the following conversions from decimal form.

$7C0_{16}$	hexadecimal
3700_8	octal
11111000000_2	binary
1984	decimal

1.3.2 Binary Machine Numbers

The computer approximates real numbers by binary floating-point numbers according to IEEE double precision, where each real number is represented by 64 bits.

IEEE double-precision (64 bits)



- s: sign
- c (characteristic): stored as a positive integer
- f (mantissa): stored as a fraction, $0 \leq f \leq 1$

Using this system gives us a floating-point number of the form

$$(-1)^s 2^{c-1023}(1+f).$$

Example 1.15: Conversion of machine number to decimal

Consider the machine number

```
0 10000000011 1011100100010000000000000000000000000000000000000000000000000
```

Convert this to decimal form.

Solution: The leftmost bit is $s = 0$, which indicates the number is positive. The next 11 bits, 10000000011, give the characteristic and are equivalent to the decimal number

$$c = 1 \cdot 2^{10} + 0 \cdot 2^9 + \cdots + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 1024 + 2 + 1 = 1027$$

The exponential part of the number is, therefore, $2^{1027-1023} = 2^4$. The final 52 bits specify that the mantissa is

$$f = 1 \cdot \left(\frac{1}{2}\right)^1 + 1 \cdot \left(\frac{1}{2}\right)^3 + 1 \cdot \left(\frac{1}{2}\right)^4 + 1 \cdot \left(\frac{1}{2}\right)^5 + 1 \cdot \left(\frac{1}{2}\right)^8 + 1 \cdot \left(\frac{1}{2}\right)^{12}.$$

So the machine number precisely represents the decimal number

$$\begin{aligned}(-1)^s 2^{c-1023}(1+f) &= (-1)^0 \cdot 2^{1027-1023}\left(1 + \left(\frac{1}{2} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{256} + \frac{1}{4096}\right)\right) \\ &= 27.56640625.\end{aligned}$$

The next largest machine number is:

```
0 10000000011 1011100100010000000000000000000000000000000000000000000000001
```

The next smallest machine number is:

```
0 10000000011 1011100100001111111111111111111111111111111111111111111111
```

There are certain special cases, summarized as:

$$x = \begin{cases} (-1)^s 2^{c-1023}(1+f) & 1 \le c \le 2046 \\ (-1)^s 2^{-1022} f & c = 0 \\ \pm\infty & c = 2047, f = 0 \\ \texttt{NaN} & c = 2047, f \neq 0 \end{cases}$$

Another well known representation is IEEE single-precision.

IEEE single-precision (32 bits)

1	8 bits	23 bits
s	c	f

Number represented:

$$x = \begin{cases} (-1)^s 2^{c-127}(1+f) & 1 \le c \le 254 \\ (-1)^s 2^{-126} f & c = 0 \\ \pm\infty, \texttt{NaN} & c = 255 \end{cases}$$

For ease of doing examples by hand, we also introduce a "Math 128a tiny precision".

Math 128A tiny precision (6 bits)

1	3 bits	2 bits
s	c	f

Number represented:

$$x = \begin{cases} (-1)^s 2^{c-3}(1+f) & 1 \le c \le 6 \\ (-1)^s 2^{-2} f & c = 0 \\ \pm\infty & c = 7, f = 0 \\ \texttt{NaN} & c = 7, f \neq 0 \end{cases}$$

Here is the tiny precision number line:

Example 1.16: Tiny precision to decimal

Convert the following binary tiny precision numbers to decimal form.

a. 0 101 01

b. 0 011 00

c. 1 001 10

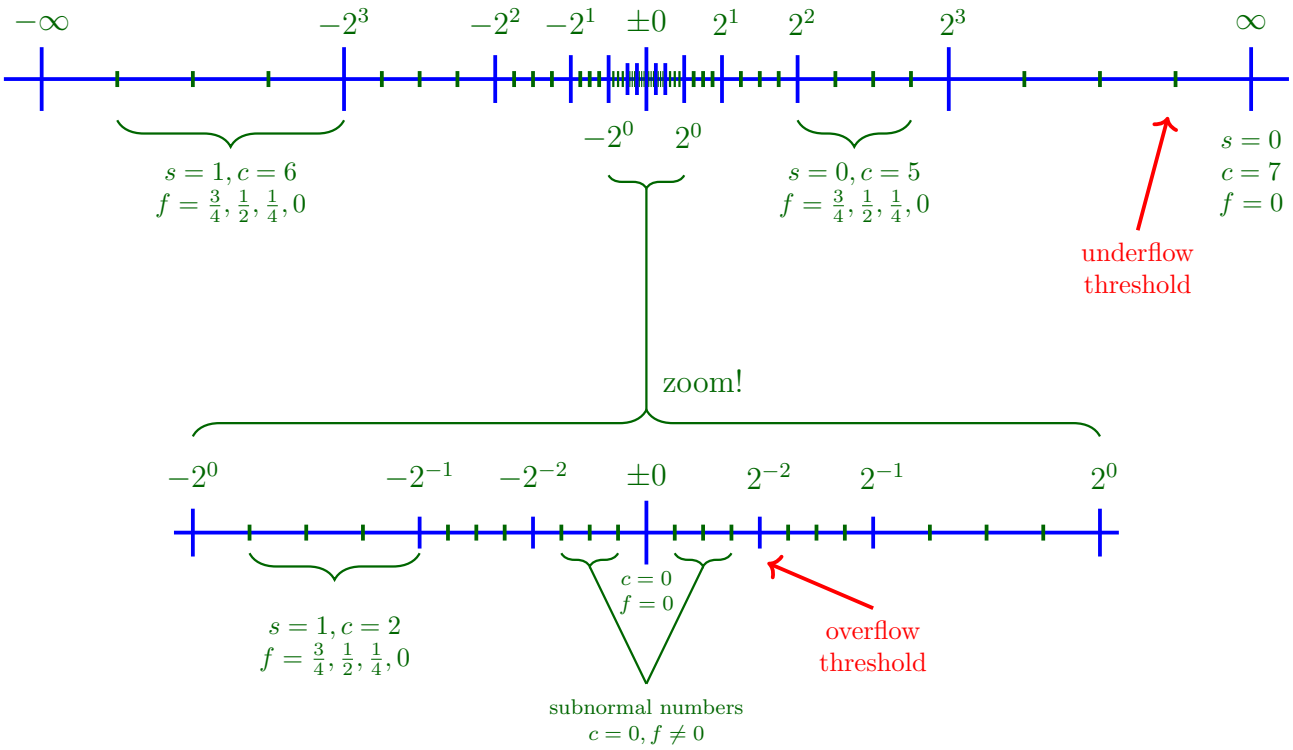


Figure 10: MATH 128a tiny precision number line

Solution:

$$0\ 101\ 01 \rightarrow \begin{cases} s = 0 \\ c = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 5 \\ f = 0 \times 2^{-1} + 1 \times 2^{-2} = \frac{1}{4} \end{cases}$$

$$0\ 011\ 00 \rightarrow 2^{3-3}(1+0) = 2^0 = 1$$

$$1\ 001\ 10 \rightarrow -2^{1-3}\left(1 + \frac{1}{2}\right) = -0.375$$

1.3.3 Rules of Computer Arithmetic

Definition 1.6 *If p^* approximates p, we define the* ***absolute*** *and* ***relative error*** *as:*

$$\textit{absolute error:} = |p - p^*|$$

$$\textit{relative error:} = \frac{|p - p^*|}{|p|} \quad \textit{(requires } p \neq 0\textit{)}$$

p^* approximates p to t significant digits if

$$\frac{|p - p^*|}{|p|} \leq 5 \times 10^{-t}$$

Example 1.17: Relative accuracy of π

Consider $\pi = 3.14159265.....$ Show that 3.1416 and 3.1415 approximates π to 6 and 5 digits, respectively.

Solution:

$$\frac{|\pi - 3.1416|}{\pi} = 2.34 \times 10^{-6} \leq 5 \times 10^{-6}$$
$$\frac{|\pi - 3.1415|}{\pi} = 2.9 \times 10^{-5} \leq 5 \times 10^{-5}$$

As another example, we can show that 0.9996 and 1.0004 both approximate 1 to 4 digits.

Rounding and Machine Epsilon

Rounding is the operation which replaces a real number x with a nearest floating point number $fl(x)$. In double precision, $\epsilon = 2^{-52}$ is machine precision (or machine epsilon). Machine epsilon gives an upper bound on the relative error due to rounding in floating point arithmetic. The key observation is that rounding commits relative error $\leq \epsilon$:

$$fl(x) = x(1 + \delta) \text{ where } |\delta| \leq \epsilon = 2^{-52}$$

or

$$\left| \frac{fl(x) - x}{x} \right| \leq \epsilon \text{ if } x \neq 0.$$

So

$$1 + \epsilon \neq 1$$

but

$$1 + \frac{\epsilon}{2} = 1$$

in floating point. Hence floating point arithmetic is not associative:

$$\left(\frac{\epsilon}{2} + 1\right) - 1 = 0$$
$$\frac{\epsilon}{2} + (1 - 1) = \frac{\epsilon}{2} \neq 0$$

In rounding, if there are two candidates, e.g. $1 + \frac{\epsilon}{2} \to 1$ or $1 + \epsilon$, then $fl(x)$ is the one with last bit 0, which is known as round to even.

Example 1.18: Rounding in binary

Round the following 5 bit binary numbers to 4 bits.

Solution:

$$\begin{aligned} 1.1111 &\to 2.000 \quad \text{(round up due to last bit even)} \\ 1.1110 &\to 1.111 \quad \text{(unambiguous)} \\ 1.1101 &\to 1.110 \quad \text{(round down due to last bit even)} \end{aligned}$$

When two binary floating point numbers are combined $(+, -, \times, \div)$, the result is usually not machine representable. IEEE guarantees correct rounding and "rounds to even" in case of ties.

Example 1.19: Addition in binary

$0\ 101\ 01 \oplus 0\ 010\ 00 = ?$

Solution: $2^{5-3}(1+\frac{1}{4}) + 2^{2-3}(1+0) = 5 + \frac{1}{2} = 5.5$. Comparing with the next largest and smallest available numbers that can be represented in Math 128a tiny precision, this gets rounded to six: $6 = 2^2(1+\frac{1}{2}) \leftrightarrow 0\ 101\ 10$, by round to even to make the last bit of f is zero.

Three Rules of Computer Arithmetic

Rule 1 *Floating point binary operations (and $\sqrt{\cdot}$) produce exact results, correctly rounded.*

The general technique to analyze sequences of operations by the golden rule. For example, subtraction is done by

$$x = a - b \implies \hat{x} = fl(\hat{a} - \hat{b})$$

This is done by rounding the inputs a, b, subtracting, and rounding the result. a and b are rounded by

$$\hat{a} = a(1+\delta_1), \quad \hat{b} = b(1+\delta_2), \quad |\delta_j| \le \epsilon$$

So

$$\begin{aligned} fl(\hat{a} - \hat{b}) &= (\hat{a} - \hat{b})(1+\delta_3) \\ &= a(1+\delta_1)(1+\delta_3) - b(1+\delta_2)(1+\delta_3) \\ &= a(1+2\delta_4) - b(1+2\delta_5) \\ &= a - b + 2a\delta_4 - 2b\delta_5 \end{aligned}$$

Where in the above calculation, we ignore the product of δ_i's as negligible. Computing the relative error gives us:

$$\left| \frac{\hat{a} - \hat{b} - (a-b)}{a-b} \right| \le \frac{(2|a| + 2|b|)\epsilon}{|a-b|}$$

The denominator could be trouble if a is close to b, in which case we cannot guarantee a small relative error in this sequence of only three floating point operations, which brings us to our next rule.

Rule 2 *Avoid subtracting approximately equal numbers, which results in severe cancellation error.*

Consider two nearly equal numbers:

$$fl(x) = 0.d_1 d_2 \dots d_p \alpha_{p+1} \alpha_{p+2} \dots \alpha_k \times 10^n$$
$$fl(y) = 0.d_1 d_2 \dots d_p \beta_{p+1} \beta_{p+2} \dots \beta_k \times 10^n$$

Subtracting of nearly equal numbers gives fewer digits of significance:

$$fl(fl(x) - fl(y)) = 0.\sigma_{p+1} \sigma_{p+2} \dots \sigma_k \times 10^{n-p}.$$

We illustrate how to avoid this issue with a simple examples.

Example 1.20: Expressions avoiding cancellation error

Consider

$$f(x) = \sqrt{4x^2 + x} - 2x$$

Find an alternative expression to avoid cancellation error for large x.

Solution: The issue here is that for large x

$$\sqrt{4x^2 + x} \approx \sqrt{4x^2} = 2x.$$

So to avoid this subtraction we multiply by the conjugate:

$$f(x) = (\sqrt{4x^2 + x} - 2x)\frac{\sqrt{4x^2 + x} + 2x}{\sqrt{4x^2 + x} + 2x} = \frac{\cancel{4x^2} + x - \cancel{4x^2}}{\sqrt{4x^2 + x} + 2x} = \frac{1}{\sqrt{4 + \frac{1}{x}} + 2}$$

This expression avoids cancellation error. You've done this exact same calculation in Math 1A for a very different reason: to evaluate $\lim_{x \to \infty} f(x) = \frac{1}{4}$.

And now with two more involved examples:

Example 1.21: Modified Quadratic Formula

Find the roots of $x^2 - 56x + 2 = 0$ by the quadratic formula, assuming you only have 5 digits of storage.

Solution: By the quadratic formula:

$$x = \frac{56 \pm \sqrt{56^2 - 8}}{2} = 28 \pm \sqrt{782} \equiv x_{\pm}.$$

Then the computed roots with 5 digits are

$$\sqrt{782} \approx 27.964 \implies x_+ \approx 55.962, x_- \approx 0.36.$$

The relative error of each root is given by:

$$\frac{|55.964 - x_+|}{|x_+|} = 4.698 \times 10^{-6} \implies 55.964 \text{ approximates } x_+ \text{ to six digits}$$
$$\frac{|0.036 - x_-|}{|x_-|} = 7.36 \times 10^{-3} \implies 0.36 \text{ approximates } x_- \text{ to only two digits}$$

A lot of cancellation occurs when subtracting 27.964 from 28. We would need to compute $\sqrt{782}$ to more digits to end up with 6 significant digits left over after subtracting 28. This difficulty can be avoided by rationalizing the denominator:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \times \frac{-b \mp \sqrt{b^2 - 4ac}}{-b \mp \sqrt{b^2 - 4ac}} = \frac{b^2 - (b^2 - 4ac)}{2a(-b \mp \sqrt{b^2 - 4ac})} = \frac{2c}{-b \mp \sqrt{b^2 - 4ac}}$$

to obtain a lesser known variant of the quadratic formula. Cancellation occurs when adding terms of the opposite sign or subtracting terms of the same sign. So we can avoid needing extra precision in intermediate calculations via:

$$b \geq 0 \implies x_+ = \frac{2c}{-b - \sqrt{b^2 - 4ac}}, x_- = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$
$$b < 0 \implies x_+ = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, x_- = \frac{2c}{-b + \sqrt{b^2 - 4ac}}$$

In the above example, $b = -56 < 0$, so we should use

$$x_- = \frac{2(2)}{56 + \sqrt{56^2 - 8}} = \frac{2}{28 + \sqrt{782}} = \frac{2}{28 + 27.964} = 0.03573726$$

which is good to six digits

$$\frac{|0.03573726 - x_-|}{|x_-|} = 4.707 \times 10^{-6}.$$

Example 1.22: A "computer counterexample" to Fermat's Last Theorem

Fermat's Last Theorem states that no three positive integers a, b, c satisfy the equation $a^n + b^n = c^n$ for an integer value of n greater than 2. Fermat originally stated this around 1637 in the margins of a book and infamously stated he had a proof that was too large to fit in the margin. The first successful proof was published in 1995 by Andrew Wiles.

Explain this apparent counterexample to the theorem.

```
>> myval = 844487^5 + 1288439^5 - 1318202^5
myval =
     0
>> myval == 0
ans =
  logical
   1
```

Solution: The problem here is that the calculations are being done in double-precision floating point and the difference between the sum of the first two terms and the third is smaller than the precision of this representation. In exact terms, we have:

$$844487^5 + 1288439^5 = 3980245235185639013055619497406$$
$$1318202^5 = 3980245235185639013290924656032$$

Giving a difference of

$$84487^5 + 1288439^5 - 1318202^5 = -235305158626$$

The finite precision of the floating point representation used truncates the decimal places before this difference can be seen.

```
>> format long
>> 844487^5 + 1288439^5
ans =
     3.980245235185639e+30
>> 1318202^5
ans =
     3.980245235185639e+30
```

So this is an example of Rule 2 where severe cancellation error occurs.

Example 1.23: Relative Error of Computer Addition

Consider the problem of adding up a list of numbers

$$S_n = \sum_{j=1}^{n} x_j$$

in the natural ordering. Assume the inputs x_j are already floating point numbers for simplicity. Compute the relative error.

Solution: We do the usual thing.

$$\begin{aligned}
fl(S_1) &= x_1 = S_1 \\
fl(S_2) = fl(x_1 + x_2) &= (x_1 + x_2)(1 + \delta_1) \\
&= S_2 + S_2\delta_1 \\
fl(S_3) = fl(fl(S_2) + x_3) &= (S_2 + S_2\delta_1 + x_3)(1 + \delta_2) \\
&= S_3 + S_2\delta_1 + S_3\delta_2 \\
fl(S_4) = fl(fl(S_3) + x_4) &= (S_3 + S_2\delta_1 + S_3\delta_2 + x_4)(1 + \delta_3) \\
&= S_4 + S_2\delta_1 + S_3\delta_2 + S_4\delta_3
\end{aligned}$$

The pattern seems clear, so the general form is

$$fl(S_n) = S_n + \sum_{j=2}^{n} S_j\delta_{j-1}$$

and the relative error is

$$\frac{|S_n - fl(S_n)|}{|S_n|} \leq \epsilon \frac{\sum_{j=2}^{n} |S_j|}{|S_n|}$$

This is a good relative error if no intermediate sums $\{S_j\}_{j=2}^{n}$ are larger than S_n. We need to keep intermediate quantities small.

Rule 3 *Keep intermediate quantities small.*

1.3.4 Examples

Example 1.24: Computer Arithmetic I

The first 11 digits of π are 3.1415926536. What is the absolute error in approximating $10^5\pi$ by 314159.27?

A 4.64×10^{-8}

B 5.36×10^{-8}

C 0.00464

D 0.00536

Solution:

$$p^* = 314159.27000$$
$$p = 314159.26536$$

So $|p^* - p| = 0.00464$.

Example 1.25: Computer Arithmetic II

Find the relative error in approximating $10^5\pi$ by 314159.27,

A 1.477×10^{-13}

B 1.477×10^{-8}

C 1.477

D 1.477×10^{8}

Solution:

$$\frac{|p^* - p|}{p} = \frac{4.64 \times 10^{-3}}{3.142 \times 10^5} = \frac{4.64}{3.142} \times 10^{-8}$$

Example 1.26: Computer Arithmetic III

Suppose you type `sin(314159.27)` in `MATLAB`. What will the result be? Assume the algorithm for $\sin(x)$ works by subtracting the nearest multiple of 2π from x and then using a Taylor expansion, with all calculations done in double-precision arithmetic.

A Impossible to estimate as roundoff errors are unpredictable.

B 0.0 (since $\sin(10^5\pi) = 0$)

C Close to my answer to Example 1.3.4A

D Close to the negative of my answer to Example 1.3.4A

E Close to my answer to Example 1.3.4B

F Close to the negative of my answer to Example 1.3.4B

Solution:

$$\sin(314159.27) = \sin(314159.27 - 10^5\pi) = \sin(0.00464)$$

And since $\sin(x) \approx x$ where x is small, the answer is C.

Example 1.27: Computer Arithmetic IV

Find the number encoded by the following bit patterns. In b and c, also find the binary representation. Here $\oplus, \ominus, \otimes, \oslash$ stand for correctly rounded floating-point arithmetic with "round-to-even" tie breaking.
a) `0 110 01` $\rightarrow x =$
b) `_ ___ __` $\rightarrow y = 1.0 \oslash x =$
c) `_ ___ __` $\rightarrow (1.0 \oslash y) \ominus x =$

Solution:
a) `0 110 01` $\rightarrow x = 2^3\left(1\frac{1}{4}\right) = 8 + 2 = 10$

b) Before rounding, $\frac{1}{x} = 0.1$. The nearest tiny precision numbers (from the number line) are $\frac{1}{8} = 0.125$ and $\frac{1}{16} = 0.0625$. So $\frac{1}{8} =$ `0 000 10`.

c) $(1.0 \oslash y) = 8$, which is representable in tiny-precision. So we have $(1.0 \oslash y) \ominus x = -2$, which is also representable. Since $s = 1, c = 4, f = 0$, we have `1 100 00`.

Example 1.28: Computer Arithmetic V

Compute the forward relative error when computing the inner product $x^T y$ of two n-vectors x and y by IEEE standard floating point arithmetic.

Solution: Given two vectors x, y, we can compute their inner product with the rules of floating point arithmetic in the natural order to get

$$fl(x^T y) = \sum x_j y_j (1 + (n - j + 1)\delta_j) \quad |\delta_j| \leq \epsilon$$

since each product $x_j y_j$ participates in $n - j$ subsequent additions. So the forward relative error is given by:

$$\begin{aligned} \frac{|fl(x^T y) - x^T y|}{|x^T y|} &= \frac{|\sum x_j y_j (1 + (n - j + 1)\delta_j) - \sum x_j y_j|}{|\sum x_j y_j|} \\ &= \frac{|\sum x_j y_j (n - j + 1)\delta_j|}{|\sum x_j y_j|} \\ &\leq \frac{n\epsilon \sum |x_j y_j|}{|\sum x_j y_j|} \end{aligned}$$

---

2 Chapter 2: Rootfinding

2.1 Basic Rootfinding Techniques

2.1.1 Bisection Method

Most of us probably discovered the bisection algorithm to calculate square roots in elementary school (at least, before all toddlers had iPads). Suppose I give you a 4 function calculator and ask you to approximate $\sqrt{19}$ as accurately as possible. Since $4^2 = 16$ and $5^2 = 25$, you know $\sqrt{19} \in [4, 5]$. You might consider $4.5^2 = 20.25$ next to deduce $\sqrt{19} \in [4, 4.5]$. Then you might further *bisect* this interval, notice $4.25^2 = 18.0625$, and that $\sqrt{19} \in [4.25, 4.5]$.

And then we proceed in this fashion until the interval becomes smaller and smaller after continual halving and we get an approximation to $\sqrt{19}$ to as many digits as we desire. And this is precisely the bisection algorithm for rootfinding applied to $f(x) = x^2 - 19$.

The algorithm

The bisection algorithm for rootfinding is based off the intermediate value theorem, which guarantees the *existence* of a root for a continuous function f on an interval $[a, b]$ by bracketing it.

1. If $f(a), f(b)$ have opposite signs ($f(a)f(b) < 0$), then $\exists p$ such that $f(p) = 0$ for $f(x)$ a continuous function. (IVT)
2. The distance from p to the midpoint $m = \frac{a+b}{2}$ is at most $\frac{b-a}{2}$ (half the length of the interval).

Let $m = \frac{a+b}{2}$ be the midpoint of $[a, b]$. In the exceedingly rare case $f(m) = 0$, then m is our root and we are done. But this pretty much never happens in floating point arithmetic for realistic problems.

Then $f(m)$ has the same sign as either $f(a)$ or $f(b)$. If $f(a)$ and $f(m)$ have the same sign, then we know the root is contained in the interval $[m, b]$, which has half the length of the original interval. Otherwise, $f(m)$ and $f(b)$ have the same sign, and we consider the interval $[a, m]$.

So starting from $[a, b]$, we produce a sequence of nested intervals

$$p \in [a_n, b_n] \subset [a_{n-1}, b_{n-1}] \subset \cdots \subset [a_1, b_1] = [a, b]$$

with $b_n - a_n \to 0$. Practically speaking, we continue this procedure until we reach some interval with length less than some user provided tolerance ϵ, and take the midpoint of that interval to be our estimate of the root.

Example 2.1: Application of Bisection

Approximate a root to $f(x) = x^2 + x - 4$ within $\epsilon = 1\text{e}{-4}$ by bisection.

Solution: We find an interval where the function takes on opposite signs by some trial and error, say $f(0) = -4$ and $f(4) = 16$. Since $f(x)$ is a polynomial and clearly continuous on this interval, the IVT guarantees the existence of a root on this interval and we can proceed with bisection.

Table 1: Iterations of Bisection

Iter (n)	a	$f(a)$	b	$f(b)$	p_n	$f(p_n)$
1	0	-4	4	16	2	2
2	0	-4	2	2	1	-2
3	1	-2	2	2	1.5	-0.25
4	1.5	-0.25	2	2	1.75	0.8125
...	...	...	...	...	...	...
...	...	...	...	...	...	...
16	1.5615234	-0.0001211	1.5616455	0.0003821	1.5615844	0.0001305
17	1.5615234	-0.0001211	1.5615844	0.0001305	1.5615539	0.0000047

Since $|a_{16} - b_{16}| = 1.2\text{e}{-4}$, we can take the midpoint of this interval to be the approximate of the root accurate to within $\epsilon = 1\text{e}{-4}$.

Comment: The assumption that f be continuous is important to invoke the IVT. For example, consider

$$f(x) = \begin{cases} 1 & x \leq 0 \\ -1 & x > 0 \end{cases}$$

Even though we can find an interval with opposite signs, there is no guarantee of a root in the interval (and there isn't one).

Sample code and practical considerations

Here is some sample code that implements bisection.

```
function [p, k] = bisection(f, a, b, tol)

   if f(a)*f(b) > 0; disp('interval does not have opposite signs'); end

   while true
      p = (a+b)/2;
      if abs(p - a) < tol || f(p) == 0, break; end
      if f(a)*f(p) > 0
         a = p;
      else
         b = p;
      end
   end
end
```

It is useful to modify the code to terminate after a maximum number of iterations to protect against some pathologies. An obvious problem is if the tolerance is too small (10^{-10}), which would cause the algorithm to run forever. E.g. if $a = 1$ and $b = 1 + \epsilon$ (machine ϵ), then $fl\left(\frac{a+b}{2}\right) = a$ correctly rounded, so $a \leftarrow a$ and no progress is made.

There is also the question of the convergence criteria. In the above code, the code converges if the length of the interval $[a, p]$ ever becomes less than a specified tolerance (ϵ) or the highly unlikely event that p is an exact root. In practice, convergence criteria are either

1. Proximity to the true unknown root p^*, $(p - p^* < \epsilon)$.
2. Residual based $(f(p) < \epsilon)$.

We will discuss this more in the next section on Newton's method.

2.1.2 Newton's Method

Setup: we want to solve $f(x) = 0$ and have a "good" initial guess x_0.

Interpretation 1: Algebraic Interpretation

We approximate $f(x) = 0$ by its first order Taylor expansion

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) = 0$$

which can be exactly solved. Algebra gives us

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

Call the result x_1 and repeat, and this is Newton's method:

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

In words: make a linear approximation of the original function, then solve exactly. Repeat until "done".

Interpretation 2: Graphical Interpretation

It turns out the above approach of approximation by first order Taylor expansion has a graphical interpretation: it is the approximation of $f(x)$ by its tangent line at the point x_0 (Figure 11).

Newton's method requires an $f(x)$ that is differentiable along with an initial guess for the root x_0. Newton's method is very sensitive to this initial guess (which we will illustrate later). A good initial guess is not always easy to find, and may depend on the setting of the specific problem, but in this class we will usually ignore this issue and take it for granted.

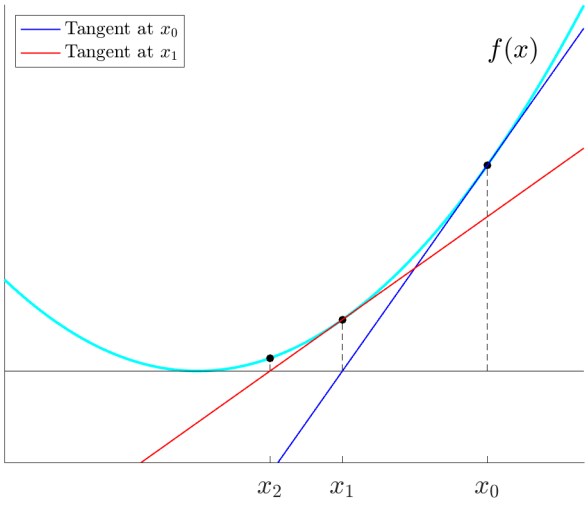


Figure 11: Graphical interpretation (tangent line) of Newton's method

Check your understanding: How many iterations should Newton's method take to converge for a linear function $f(x)$?

Example 2.2: Application of Newton's Method

Apply Newton's method to $f(x) = x^2 + x - 4$ and the initial guess $x_0 = 2$.

Solution: We also need the derivative $f'(x) = 2x+1$. Here are the first two Newton iterations:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 2 - \frac{2}{5} = 1.6$$
$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} \approx 1.6 - 0.038095238 = 1.56190476$$

Here we happen to know the true solution thanks to the quadratic formula

$$x^* = \frac{-1 + \sqrt{17}}{2} \approx 1.561552812808830292823358831810 8.$$

Table 2: Iterations of Newton's Method

Iter (n)	x_n
1	1.6
2	1.561904761904761904761904761904 8
3	1.561552842846145386561090949081 7
4	1.561552812808830292823358831810 8

Sample code and practical considerations

Here is some sample code that implements Newton's method.

```
f = @(x) x^2 - 2;
df = @(x) 2*x;
[p, n] = newton(f, df, 1, 1.0e-12)

function [p, n] = newton(f, df, p0, tol)
   for n = 1:100
      p = p0 - f(p0)/df(p0);
      if abs(p - p0) < tol, break; end
      p0 = p;
   end
end
```

We revisit the issue of the convergence criteria. Here the criteria is that if two subsequent iterations differ by less than a certain tolerance, then we call that the root. There is also a safeguard to stop after 100 iterations in the very real possibility that Newton's method diverges for the given input.

We note an issue with this convergence criteria - it could just mean the iteration has stalled, or just because you are close to the true root doesn't mean it "acts much like a root". For example, consider the example of $f(x) = e^{10^{20}x} - 1$. The true root is obviously $p^* = 0$, so $f(p^*) = 0$. Consider $p = 10^{-10}$. This is very close to the true root, but $f(p)$ results in overflow (∞). While this example is a bit extreme and artificial, it makes the argument for *residual* based convergence criterion, and shows how proximity to the root and driving the residual to zero – while usually closely correlated – can also be wildly unrelated.

2.1.3 Comparison of Bisection and Newton's Method

Speed

To solve $f(x) = x^2 + x - 4$, bisection needed 17 iterations and Newton's method needed only 4 iterations to converge. This is pretty typical behavior for the two algorithms. Bisection (linear convergence) is much slower than Newton's method (quadratic convergence). Empirically speaking, linear convergence results in one additional correct digit per iteration and quadratic convergence results in the number of correct digits doubling per iteration.

Robustness

While bisection is very slow, it is always guaranteed to find a root once the process begins. Compare this to Newton's method, which doesn't have such guarantees. For example, consider $f(x) = x^3 - 2x + 2, x_0 = 0$. One can easily show that $x_1 = 1, x_2 = 0$, and that Newton's method will just cycle back and forth forever, even though it is clear that this function has a root (Figure 12). We got unlucky with this starting point – if we would have chosen $x_0 = 0.5$,

we would have converged quickly to the root $x^* \approx -1.76929$.

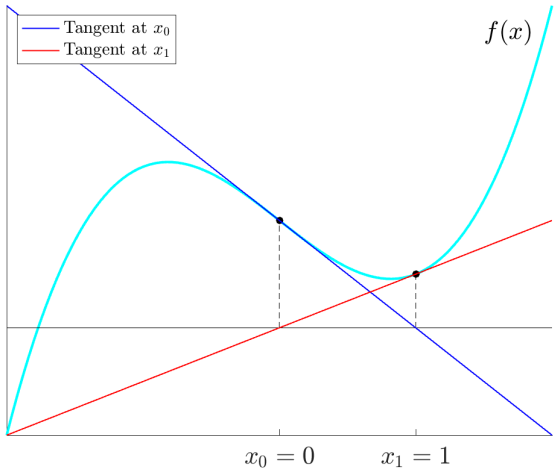


Figure 12: Newton's method oscillating

Now consider the functions $\arctan(x)$ and $x^{1/3}$ (Figure 13). Both of these functions seem qualitatively very similar and each have a single root at ($x^* = 0$). It should be clear that sufficiently far away from the origin, both functions are too "flat", and an initial guess for Newton's method there would shoot you off further and further away from the origin with alternating signs.

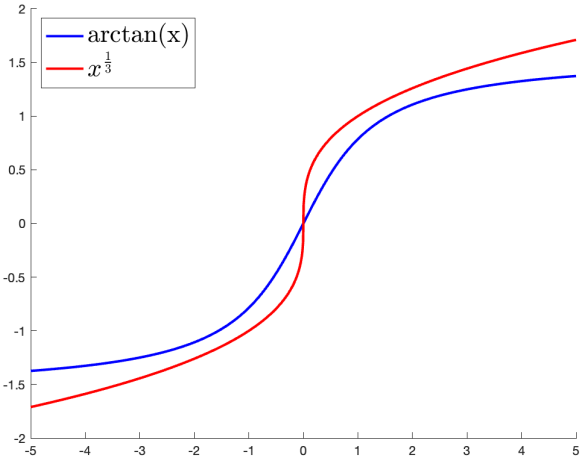


Figure 13: $\arctan(x)$ vs. $x^{1/3}$

In fact, it's even worse for $f(x) = x^{1/3}$. Put very imprecisely, it's too flat far away from the origin and it's too steep close to the origin. This results in Newton's method *never converging* for any initial guess except for exactly the root $p_0 = 0$. One can also show this

algebraically: plugging this function into the Newton iteration gives us

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{x^{1/3}}{\frac{1}{3}x^{-2/3}} = x_n - 3x_n = -2x_n.$$

So for any nonzero initial guess, Newton's method will diverge.

Now consider $f(x) = \arctan(x)$, which although shaped like $f(x) = x^{1/3}$, exhibits rather different behavior (Figure 14). For $|x_0| > 1.3917$, Newton's method will diverge. For $|x_0| < 1.3917$, Newton's method will converge. For $|x_0| = 1.3917$, Newton's method will oscillate forever. This demonstrates that Newton's method is a locally convergent method - it converges if we start "close enough" to the root.

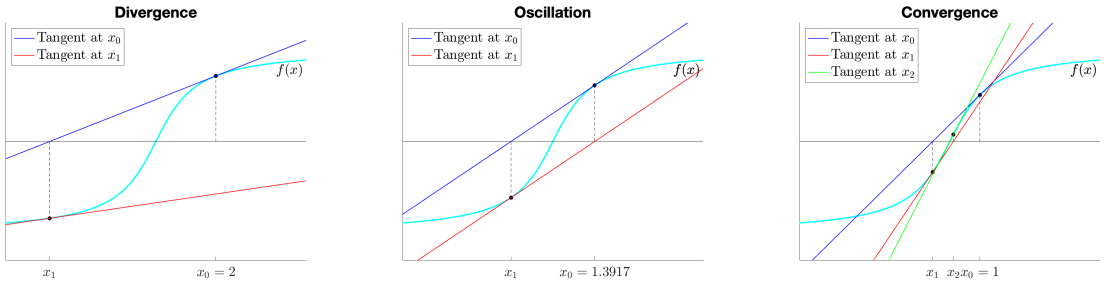


Figure 14: Effect of initial guesses on Newton's method for $\arctan(x)$

Function Evaluations vs. Derivatives

Bisection only requires function evaluations, but Newton's method also requires a first derivative. You may take this for granted in an academic setting, but in practical situations finding the derivative can be impractical or expensive compared to a function evaluation. One way to get around finding exact expressions for the derivative is to use a finite difference approximation for the derivative, which we will learn about in Chapter 4. It really just means take

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0)}{h} \text{ for small } h, \text{ like } h = 10^{-6}$$

Another workaround is to use the secant method (poor man's Newton method), which approximates the derivative by the slope of a secant line. Note this requires two starting guesses which are hopefully close to the root.

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

Generalization to High Dimensions

This is beyond the scope of the class, but worth mentioning because it is so useful in practical situations. Oftentimes you want to solve multiple equations in multiple variables. For example:

$$\vec{\mathbf{f}}(x_1, x_2, x_3) = \begin{bmatrix} f_1(x_1, x_2, x_3) \\ f_2(x_1, x_2, x_3) \\ f_3(x_1, x_2, x_3) \end{bmatrix} = \vec{\mathbf{0}}$$

The linear approximation we use in Newton's method has a natural generalization to higher dimensions (Math 53). The main difference is that instead of a derivative, we need a Jacobian matrix. Here is the linearization of $\vec{\mathbf{f}}(\vec{\mathbf{x}})$ around a point $\vec{\mathbf{x_0}}$.

$$\vec{\mathbf{f}}(\vec{\mathbf{x}}) = \vec{\mathbf{f}}(\vec{\mathbf{x_0}}) + D\vec{\mathbf{f}}(\vec{\mathbf{x_0}})(\vec{\mathbf{x}} - \vec{\mathbf{x_0}})$$

Where $D\vec{\mathbf{f}} = \begin{bmatrix} \frac{\partial f_1}{x_1} & \frac{\partial f_1}{x_2} & \frac{\partial f_1}{x_3} \\ \frac{\partial f_2}{x_1} & \frac{\partial f_2}{x_2} & \frac{\partial f_2}{x_3} \\ \frac{\partial f_3}{x_1} & \frac{\partial f_3}{x_2} & \frac{\partial f_3}{x_3} \end{bmatrix} = \left[\frac{\partial f_i}{\partial x_j}\right]_{ij}$.

Bisection would require you to do a search in each direction at each iteration. Considering bisection was slow to begin with, this additional ground to cover in the higher dimensions makes it prohibitively slow. This is an extremely important consideration in practice, and as a result bisection is rarely used in higher dimensions.

To sum up, here is a table.

Table 3: Bisection vs. Newton's Method

Bisection	Newton
Slow (Linear)	Fast (Quadratic)
Robust	Could Fly Off
Only Function Evaluations	Needs Derivative
Only for 1D	Works well for high-D

Failures of Both Methods

And finally, here some shortcomings of both methods.

- Finding all the roots:
  - Bisection: If the intermediate value theorem guarantees the existence of a root in an interval $[a, b]$ and there are multiple roots in that interval, there's nothing you can really do to find all of them consistently. You will just get one of them.
  - Newton's method: Different starting points converge to different roots (or sometimes not at all). The best you can do is try to find a good guess close to your root of interest and *hope* it converges to that.
- Repeated roots:
  - Bisection: If you have a function like $f(x) = (x-2)^2$, clearly $x = 2$ is a root, but the intermediate value theorem doesn't even apply here, so you can't use Bisection to find this root.

– Newton's method: is well known to be slower (no longer quadratic convergence) in the presence of repeated roots. There are some modifications you can make to combat this; for example if you know the order of the root is n, you can try the iteration

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

You can read more about this in §2.4 of BFB.

2.2 Fixed Point Iteration

2.2.1 Motivation

Consider some function continuous $g(x)$. Define a **fixed point** of $g(x)$ to be p such that $g(p) = p$.

Say we want to find a fixed point of a given $g(x)$. One obvious thing to do is to try fixed point iteration. Pick some starting value x_0, and continue to iterate $x_1 = g(x_0), x_2 = g(x_1), x_3 = g(x_2) \ldots x_{n+1} = g(x_n)$ until (hopefully) this converges to a fixed point.

This is all fine and good, but you should find this very strange. Sure, this is a very simple and straightforward idea, and it would be great if it worked. **But you should have no reason to expect that this would work in general!!** The following example is meant to illustrate this.

Consider the equation $f(x) = x^3 + 4x^2 - 10$. By the intermediate value theorem there exists a root in the interval $[1, 2]$. There are many ways to change the equation $f(x) = 0$ to a fixed point iteration of the form $x = g(x)$.

Here are 5 such examples. The first four are equivalent to $f(x) = 0$ by algebra and the fifth is Newton's method.

a) $x = g_1(x) = x - x^3 - 4x^2 + 10$

b) $x = g_2(x) = \left(\frac{10}{x} - 4x\right)^{1/2}$

c) $x = g_3(x) = \frac{1}{2}(10 - x^3)^{1/2}$

d) $x = g_4(x) = \left(\frac{10}{4+x}\right)^{1/2}$

e) $x = g_5(x) = x - \frac{x^3+4x^2-10}{3x^2+8x}$

Take $p_0 = 1.5$, Table 4 lists the results for each of the fixed point iterations above. If one of these fixed point iterations $g_i(x)$ converges to a fixed point, it must (by design) be a root of $f(x)$.

Note that these all behave very differently. This should convince you that finding a "good" fixed point iteration is no easy task. An important practical note for this class is that we **won't worry** about the problem of *finding* a good fixed point function. We will always take them as given and analyze them.

2.2.2 Two Main Theorems: Existence and Uniqueness

Say somebody gives you a fixed point iteration and a starting point, and you begin iterating. The first thing you would hope for is that your iteration will converge (question of existence). If it converges, the natural next question is whether the fixed point is unique.

Table 4: Fixed Point Iteration

Iter (n)	**(a)**	**(b)**	**(c)**	**(d)**	**(e)**
0	1.5	1.5	1.5	1.5	1.5
1	-0.875	0.8165	1.286953768	1.348399725	1.373333333
2	6.732	2.9969	1.402540804	1.367376372	1.365262015
3	-469.7	$(-8.65)^{1/2}$	1.345458374	1.364957015	1.365230014
4	1.03×10^8		1.375170253	1.365264748	1.365230013
5			1.360094193	1.365225594	
6			1.367846968	1.365230576	
7			1.363887004	1.365229942	
8			1.365916734	1.365230022	
9			1.364878217	1.365230012	
10			1.365410062	1.365230014	
15			1.365223680	1.365230013	
20			1.365230236		
25			1.365230006		
30			1.365230013		

Existence

Theorem 2.1 *If $g(x)$ is a continuous function that defines our fixed point iteration and there exists an interval $[a, b]$ such that $x \in [a, b] \implies g(x) \in [a, b]$, then $g(x)$ has at least one fixed point in $[a, b]$.*

Proof: We know that $g(a) > a$ and $g(b) < b$. Consider $h(x) \coloneqq g(x) - x$, which is continuous. Since $h(a) = g(a) - a > 0$ and $h(b) = g(b) - b < 0$, by the intermediate value theorem there exists some point $p \in [a, b]$ such that $h(p) = 0 \implies g(p) = p$.■

Uniqueness

Theorem 2.2 *If $g'(x)$ exists on the interval (a, b) and there exists some $k < 1$ such that $|g'(x)| \leq k \ \forall x \in (a, b)$, then there is exactly one fixed point in $[a, b]$.*

Proof: We proceed with the standard method for proving uniqueness by assuming two such fixed points exist and reach a contradiction. Suppose $p, q \in [a, b]$ are fixed points of $g(x)$. Then by the mean value theorem, there exists some $\xi \in (p, q) \subset [a, b]$ such that

$$g'(\xi) = \frac{g(p) - g(q)}{p - q}$$

Thus

$$|p - q| = |g(p) - g(q)| = |g'(\xi)||p - q| \leq k|p - q| < |p - q|$$

Since $|p-q| < |p-q|$ with strict inequality is absurd, we have reached a contradiction and the fixed point is unique.

2.2.3 Examples

Example 2.3: Fixed Point Iteration I

Consider the iteration

$$p_n = \frac{p_{n-1}^2 + 3}{5}, \quad n = 1, 2, 3...$$

a) Show this fixed point iteration converges to a unique point for any initial $p_0 \in [0, 1]$.

b) What value p does this iteration converge to?

c) Give a lower bound on the number of iterations n are required to obtain an absolute error $|p_n - p|$ less than 10^{-4} where $p_0 = 1$. (No numerical value needed, just give an expression for n).

Solution:

a) Let $g(x) \equiv \frac{x^2+3}{5}$. We want to show $g(x) \in [0, 1]$ for $x \in [0, 1]$. Since $g(0) = \frac{3}{5}$, $g(1) = \frac{4}{5}$, and g is an increasing function, this gives us existence. Since $g'(x) = \frac{2x}{5} \leq \frac{2}{5} = k < 1$ for $x \in [0, 1]$, this gives us uniqueness.

b) Let $\lim_{n\to\infty} p_n = p$. Taking limits on both sides:

$$\lim_{n\to\infty} p_n = \lim_{n\to\infty} \frac{p_{n-1}^2 + 3}{5} \implies p = \frac{p^2+3}{5}.$$

Solve this equation with the quadratic formula to get $p = \frac{5\pm\sqrt{13}}{2}$, and discard the root not in the interval $[0, 1]$. So this iteration converges to $p = \frac{5-\sqrt{13}}{2}$.

c) $10^{-4} = |p_n - p| \leq k^n \max\{1, 0\} = \left(\frac{2}{5}\right)^n \implies n \approx \frac{-4}{\log_{10}(2/5)}$

Example 2.4: Fixed Point Iteration II

Consider the iteration

$$p_n = 2 + \frac{1}{p_{n-1}}, \quad n = 1, 2, 3... \quad p_0 = 2$$

a) Show that $p_n \in [2, 2.5]$ $\forall n$ and that this iteration has a unique fixed point.

b) What value p does this iteration converge to?

Solution:

a) Since $p_n \geq 0\ \forall n$, this implies that $\frac{1}{p_n} \geq 0 \implies 2 + \frac{1}{p_n} \geq 2$. For the other side, since $p_n \geq 2$, this implies that $\frac{1}{p_n} \leq \frac{1}{2} \implies 2 + \frac{1}{p_n} \leq 2.5$.
Define $g(x) \equiv 2 + \frac{1}{x}$, $\max\limits_{[2,2.5]} |g'(x)| = \max\limits_{[2,2.5]} \dfrac{1}{x^2} \leq \dfrac{1}{4} = k < 1$.

b) Take limits and then $p = 1 \pm \sqrt{2}$. Discard the one that's negative so $p = 1 + \sqrt{2}$.

Example 2.5: Fixed Point Iteration III

Let $g(x) = \frac{1}{x} + \frac{x}{4}$.

a) Prove that $g(x)$ has a unique fixed point $p \in [1, 2]$.

b) Let $p_0 = \frac{3}{2}, p_{n+1} = g(p_n)$. How many iterations are required to guarantee $|p_n - p| \leq 10^{-7}$? (Write your answer as a ratio of logarithms).

Solution:

a) $g'(x) = -\frac{1}{x^2} + \frac{1}{4} = 0$ when $x = \pm 2$. So the only critical point in $[1, 2]$ is an endpoint. $g(1) = 1 + \frac{1}{4} = \frac{5}{4}$ (the max), and $g(2) = \frac{1}{2} + \frac{1}{2} = 1$ (the min). g maps $[1, 2]$ to $[1, 2]$ since both extrema are in this range.
$g''(x) = \frac{2}{x^3} > 0$, so $g'(x)$ is increasing. $g'(1) = -1 + \frac{1}{4} = -\frac{3}{4}, g'(2) = 0 \implies -\frac{3}{4} \leq g'(x) \leq 0$. So $|g'(x)| \leq \frac{3}{4}$ for $x \in [1, 2]$.

b) $|p_n - p| \leq k^n \max(p_0 - a, b - p_0) = \left(\frac{3}{4}\right)^n \left(\frac{1}{2}\right) \leq 10^{-7}$. This implies that $n \geq \frac{\ln(2 \times 10^{-7})}{\ln(3/4)}$

Comment: These are really all the same problem, but note that you can take different approaches on different $g(x)$.

Example 2.6: Fixed Point Iteration IV (pretty hard)

Let $g(x) = x - \tan(x)$ and consider the fixed point iteration $p_{n+1} = g(p_n)$. Show that if $|p_0| \leq \frac{\pi}{6}$, then $p_n \to 0$ *at least* quadratically.
(Useful Hints: $\frac{d}{dx}\tan(x) = \sec^2(x), 1 \leq \sec^2(x) \leq \frac{4}{3}$ for $|x| \leq \frac{\pi}{6}$, and that $\tan(\frac{\pi}{6}) = \frac{1}{\sqrt{3}} < \frac{\pi}{3}$).

Solution: Since $g(x) = x - \tan(x)$ and $g'(x) = 1 - \sec^2(x)$, the hint gives us $0 \leq \sec^2(x) - 1 \leq \frac{1}{3}$ for $-\frac{\pi}{6} \leq x \leq \frac{\pi}{6} \implies -\frac{1}{3} \leq g'(x) \leq 0$, which implies $g(x)$ is decreasing on $-\frac{\pi}{6} \leq x \leq \frac{\pi}{6}$. This gives us that $|g'(x)| \leq \frac{1}{3} < 1$.
Compute $g(-\frac{\pi}{6}) = -\frac{\pi}{6} - \tan(-\frac{\pi}{6}) = -\frac{\pi}{6} + \frac{1}{\sqrt{3}} < \frac{\pi}{6}$. Then compute $g(\frac{\pi}{6}) = \frac{\pi}{6} - \tan(\frac{\pi}{6}) = \frac{\pi}{6} - \frac{1}{\sqrt{3}} > -\frac{\pi}{6}$. Since $g(x)$ is decreasing, $g(x)$ maps $[-\frac{\pi}{6}, \frac{\pi}{6}]$ to itself.

So we have shown that if $|p_0| \leq \frac{\pi}{6}, p_n \to 0$.

As for the quadratically part, we have that $g(0) = 0$ and $g'(0) = 0$. This implies the convergence is *at least* quadratic. (See the first slide of the Week 4 Lecture notes for this theorem and the proof that connects this with the definition of order of convergence).

2.3 Order of Convergence

Here is the definition of rate of convergence (BFB §1.3 p34)

> Suppose $\{\beta_n\}_{n=1}^{\infty}$ is a sequence known to converge to zero and $\{\alpha_n\}_{n=1}^{\infty}$ converges to a number α. If a positive constant K exists with
>
> $$|\alpha_n - \alpha| \leq K|\beta_n|, \quad \text{for large } n,$$
>
> then we say that $\{\alpha_n\}_{n=1}^{\infty}$ converges to α with **rate, or order, of convergence** $O(\beta_n)$. (This expression is read "big oh of β_n".) It is indicated by writing $\alpha_n = \alpha + O(\beta_n)$. ■

Later on, we have the definition of order of convergence (BFB §2.4 p78).

> Suppose $\{p_n\}_{n=0}^{\infty}$ is a sequence that converges to p, with $p_n \neq p$ for all n. If positive constants λ and α exist with
>
> $$\lim_{n\to\infty} \frac{|p_{n+1} - p|}{|p_n - p|^{\alpha}} = \lambda,$$
>
> then $\{p_n\}_{n=0}^{\infty}$ **converges to p of order α, with asymptotic error constant λ.** ■
>
> **(i)** If $\alpha = 1$ (and $\lambda < 1$), the sequence is **linearly convergent**.
>
> **(ii)** If $\alpha = 2$, the sequence is **quadratically convergent**.

Ignore how the book says 'rate, or order, of convergence' in the first definition. We call the first one rate, and the second one order. Even with this slight ambiguity, it is usually clear from context which one a problem is asking for. Here is a good alternative explanation with a few examples: http://fourier.eng.hmc.edu/e176/lectures/NM/.

Examples

Example 2.7: Rate of Convergence

$$\alpha_n = \frac{2n + \sin(n)}{n^3}$$

What value α does this sequence converge to? What is the rate of convergence of this sequence? That is, what is the sequence β_n such that $\alpha_n = \alpha + O(\beta_n)$?

Solution: Clearly $\alpha = 0$. Now we need to find a sequence $\beta_n \to 0$ such that β_n satisfies the definition of rate of convergence.

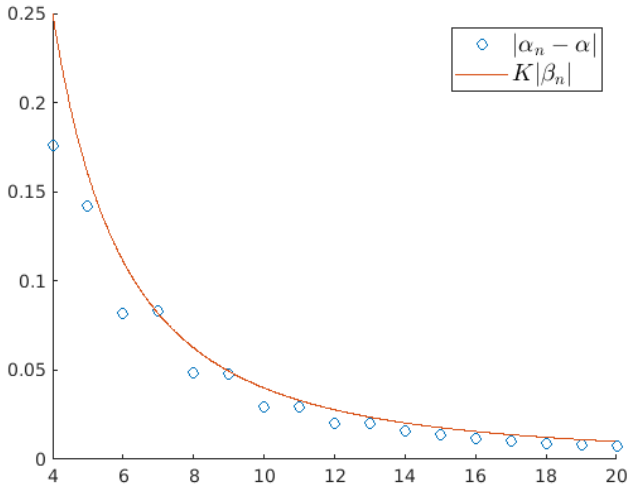
$$|\alpha_n - \alpha| = \left|\frac{2n + \sin(n)}{n^3}\right| \leq \frac{2n + n}{n^3} = \frac{3}{n^2}$$

Here we use the fact that $|\sin(n)| < 1 < n$. The desired sequence here is $\beta_n = \frac{1}{n^2}$.

Comment 1: Common sequences are $\beta_n = \frac{1}{n^p}$ where $p > 1$ or $\beta_n = \gamma^n$ where $\gamma < 1$.

Comment 2: The fact that we just need such a constant K to exist means that we don't care about the scaling of the RHS $K|\beta_n|$, we just care about the shape given by β_n.

Comment 3: The definition says this inequality must hold for large n, or more specifically, there must exist some N such that this holds for all $n > N$. This means we can have a sequence like the below (blue dots) that has values greater than $K|\beta_n|$ for some n, as long as for all $n > N$ for some N, $|\alpha_n - \alpha| \leq K|\beta_n|$.



Example 2.8: Rate of Convergence of bisection

Find the rate of convergence of bisection.

Solution: Let p_n be the sequence generated by the bisection algorithm and p be the root it converges to. For bisection, we have the upper bound

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

Since $|b-a|$ is just a constant, $\beta_n = \frac{1}{2^n}$.

Comment: Note that this upper bound lets us estimate the number of iterations we needed to reach a certain tolerance. Say I want to iterate until I get a root that is accurate to within $\epsilon = 1e-4$. Then I just need to solve for n such that $\frac{|b-a|}{2^n} < \epsilon$.

Now let's talk about Bisection's order of convergence. To develop some intuition, suppose that $\frac{|p_{n+1}-p|}{|p_n-p|^\alpha} = \lambda$ (even though this is actually only true in the limit). For linear convergence, where $\alpha = 1, \lambda < 1$, we have that

$$|p_n - p| = \lambda|p_{n-1} - p| = \cdots = \lambda^n|p_0 - p|$$

This means the error is cut by some factor of λ in each iteration. We say that Bisection is linearly convergent with asymptotic error constant $\lambda = 1/2$. We can see that the error in a linearly convergent method exhibits exponential decay.

For quadratic convergence, we have something that roughly looks like:

$$|p_n - p| = \lambda|p_{n-1} - p|^2 = \lambda(\lambda|p_{n-2} - p|^2)^2 = \lambda^3|p_{n-2} - p|^4 = \cdots \lambda^{2^n-1}|p_0 - p|^{2^n}$$

This exhibits *super*-exponential decay.

Example 2.9: Rate and order of convergence

Consider the sequence $\beta_n = \frac{1}{n^3}$. What is its rate of convergence and order of convergence?

Solution: This clearly has rate of convergence $O(\frac{1}{n^3})$, but since $\frac{\frac{1}{(n+1)^3}}{\frac{1}{n^3}} \to 1$, this sequence is not even linearly convergent!

Example 2.10: Example I

Consider the iteration

$$x_{k+1} = (x_k + 1/x_k)/2, \quad k = 0, 1, \ldots$$

Assume that $x_0 > 0$ and that the iteration converges.

1. What number does the iteration converge to?
2. What is the order of convergence?

Solution: Let $x = \lim_{k\to\infty} x_k$. Take limits of both sides and simplify to get the equation $x^2 = 1$. Discard the negative root (since the iterates are always positive) to get that $x = 1$.

To find the order of convergence, let's first try linear ($\alpha = 1$) and modify if needed. Proceeding with the definition, we have that

$$\begin{aligned}
\lim_{k\to\infty}\frac{|x_{k+1}-1|}{|x_k-1|} &= \lim_{k\to\infty}\frac{\left|\frac{x_k+\frac{1}{x_k}}{2}-1\right|}{|x_k-1|} \\
&= \lim_{k\to\infty}\frac{\left|x_k+\frac{1}{x_k}-2\right|}{2|x_k-1|} \\
&= \lim_{k\to\infty}\frac{|x_k^2+1-2x_k|}{2x_k|x_k-1|} \\
&= \lim_{k\to\infty}\frac{|x_k-1|}{2x_k} = 0
\end{aligned}$$

But this is not an acceptable constant. We require $0 < L < 1$. Let's try quadratic $\alpha = 2$. Then with a slight modification to to the above:

$$\begin{aligned}
\lim_{k\to\infty}\frac{|x_{k+1}-1|}{|x_k-1|^2} &= \lim_{k\to\infty}\frac{\left|\frac{x_k+\frac{1}{x_k}}{2}-1\right|}{|x_k-1|^2} \\
&= \lim_{k\to\infty}\frac{\left|x_k+\frac{1}{x_k}-2\right|}{2|x_k-1|^2} \\
&= \lim_{k\to\infty}\frac{|x_k^2+1-2x_k|}{2x_k|x_k-1|^2} \\
&= \lim_{k\to\infty}\frac{1}{2x_k} = \frac{1}{2}
\end{aligned}$$

So this iteration converges quadratically.

Example 2.11: Example II

Consider the methods for computing $21^{1/3}$, for $n = 1, 2, \ldots$

$$(a)\ p_{n+1} = p_n - \frac{p_n^3 - 21}{3p_n^2} \qquad (b)\ p_{n+1} = \left(\frac{21}{p_n}\right)^{1/2}$$

Assume that $p_0 > 0$ as that the iterations converge. What is the order of convergence for each method?

Solution: (a) This is just Newton's method applied to $f(x) = x^3 - 21$. We can verify that $p = 21^{1/3}$ is a simple root, so Newton's method converges quadratically.

(b) Method 1: Directly proceed by the definition.

$$\lim_{n\to\infty}\frac{p_{n+1}-p}{p_n-p}=\lim_{n\to\infty}\frac{\left|\left(\frac{21}{p_n}\right)^{1/2}-21^{1/3}\right|}{|p_n-21^{1/3}|}=\lim_{n\to\infty}\frac{\left|21^{1/2}-21^{1/3}p_n^{1/2}\right|}{|(p_n^{1/2}-21^{1/6})(p_n^{1/2}+21^{1/6})||p_n^{1/2}|}$$
$$=\lim_{n\to\infty}\frac{21^{1/3}\cancel{|p_n^{1/2}-21^{1/6}|}}{\cancel{|(p_n^{1/2}-21^{1/6})}(p_n^{1/2}+21^{1/6})||p_n^{1/2}|}$$
$$=\frac{21^{1/6}}{(21^{1/6}+21^{1/6})21^{1/6}}=\frac{1}{2}$$

Method 2: Check the theorem XYZ. Consider $g(p_n):=\left(\frac{21}{p_n}\right)^{1/2}$.

$$g'(p_n)=\frac{1}{2}\left(\frac{21}{p_n}\right)^{-1/2}\left(-\frac{21}{p_n^2}\right)=-\frac{\sqrt{21}}{2p_n^{3/2}}$$
$$|g'(p)|=\frac{1}{2}$$

Example 2.12: Example III

Find the rate of convergence of $\cos(2x)+2x\sin(x)$ as $x\to 0$.

Solution:

$$\cos(2x)+2x\sin(x)=\left(1-\frac{(2x)^2}{2!}+\frac{(2x)^4}{4!}+O(x^6)\right)+2x\left(x-\frac{x^3}{3!}+O(x^5)\right)$$
$$=1-2x^2+\frac{2}{3}x^4+2x^2-\frac{1}{3}x^4+O(x^6)$$
$$=1+O(x^4)$$

---

3 Chapter 3: Interpolation and Approximation

3.1 Background

3.1.1 Motivation

The idea behind polynomial interpolation: given $n+1$ data points

$$(x_0, y_0), (x_1, y_1), \ldots (x_n, y_n),$$

find a degree n polynomial P_n such that $P_n(x_i) = y_i,\ i = 0, 1, \ldots n$, that is, find a polynomial that *interpolates* the given data.

Practically speaking, the motivation is that we have some data generating process for which there typically doesn't exist a closed form function for, but we are still interested in having a functional form for approximation purposes and analysis. For example, consider

$$f(\text{year}) = \text{population}.$$

We can sample various points, but of course there is not a true underlying closed form. So we seek to approximate this function by a polynomial. The choice of using polynomials is motivated by the Weierstrass approximation theorem (Math 104):

Theorem 3.1 *Weierstrass Approximation Theorem: If $f \in C[a, b]$ and $\epsilon > 0$, $\exists P(x)$ such that $|P(x) - f(x)| < \epsilon\ \forall x \in [a, b]$.*

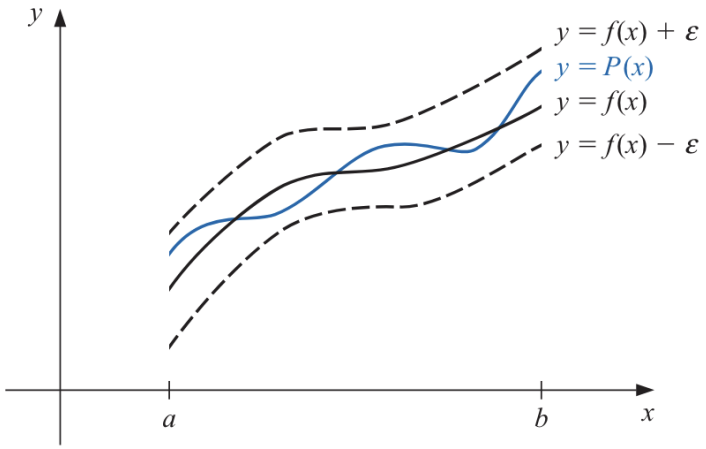


Figure 15: Uniform approximation of $f(x)$ by $p(x)$ on $[a, b]$

In English, this just says that we can always approximate a continuous function on $[a, b]$ as closely as we'd like (within ϵ). Like many theorems in real analysis, they establish the existence of such a wonderful approximating polynomial $P(x)$ but provide no insight on how to construct such a polynomial.

Taylor polynomials are not a good candidate for this, since they best approximate the function around an expansion point x_0 and diverge outside the radius of convergence, as seen in Figure 8. So we choose to construct interpolating polynomials to approximate these functions. These have their own shortcomings, which we will discuss later.

Given this unknown f from which we have data or we can sample from, we still want to do the usual operations on it (rootfinding, differentiation, integration, etc.) So the idea is to construct a polynomial P that (hopefully) approximates f well, which is now easy to work with. This extends our "approximate and solve" paradigm to

- Rootfinding: $P(x) = 0 \implies f(x) \approx 0$
- Differentiation: $P'(x) \approx f'(x)$
- Integration: $\int_0^x P(s)\ ds \approx \int_0^x f(s)\ ds$

since these operations are all easy to do on polynomials.

3.1.2 Facts about Polynomials

Theorem 3.2 *An nth degree polynomial with $n+1$ distinct zeros must be identically zero.*

Before we prove this, convince yourself this is true. For example, consider a first degree polynomial $f(x) = a_0 + a_1 x$. It's pretty obvious that the only linear polynomial that has more than one zero is the one corresponding to $a_0 = a_1 = 0$, which is just identically zero, $f(x) \equiv 0$. Now try drawing a quadratic with three distinct zeros, and the same logic extends for higher degree polynomials.

Proof: By the fundamental theorem of algebra, every degree n polynomial can be written as

$$P(x) = c_n(x - x_1)(x - x_2)\dots(x - x_n)$$

where $x_1, x_2, \dots x_n$ are the first n of these distinct zeros. In order for this polynomial to vanish at the $(n+1)$th zero, we must have $c_n = 0$, so the polynomial is identically zero.

Theorem 3.3 *Given $n+1$ distinct points $(x_0, y_0), (x_1, y_1), \dots (x_n, y_n)$, there **exists** a unique degree n interpolating polynomial $P_n(x) = a_n x^n + a_{n-1}x^{n-1} + \dots + a_1 x + a_0$.*

Proof: Before you took this class and learned about any interpolation formulas, in order to find the interpolating polynomial you would plug in each interpolation point (x_i, y_i) to the desired form of the polynomial $P_n(x)$ and solve for the coefficients.

$$
\begin{aligned}
a_n(x_0)^n + a_{n-1}(x_0)^{n-1} + a_{n-2}(x_0)^{n-2} + a_1 x_0 + a_0 &= y_0 \\
a_n(x_1)^n + a_{n-1}(x_1)^{n-1} + a_{n-2}(x_1)^{n-2} + a_1 x_1 + a_0 &= y_1 \\
a_n(x_2)^n + a_{n-1}(x_2)^{n-1} + a_{n-2}(x_2)^{n-2} + a_1 x_2 + a_0 &= y_2 \\
&\dots \\
a_n(x_n)^n + a_{n-1}(x_n)^{n-1} + a_{n-2}(x_n)^{n-2} + a_1 x_n + a_0 &= y_n
\end{aligned}
$$

Writing this system in matrix form:

$$
\begin{bmatrix}
1 & x_0 & x_0^2 & \dots & x_0^n \\
1 & x_1 & x_1^2 & \dots & x_1^n \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & x_n & x_n^2 & \dots & x_n^n
\end{bmatrix}
\begin{bmatrix}
a_0 \\ a_1 \\ \vdots \\ a_n
\end{bmatrix}
=
\begin{bmatrix}
y_0 \\ y_1 \\ \vdots \\ y_n
\end{bmatrix}
$$

This matrix is known as the Vandermonde matrix. This matrix is invertible iff the points x_i are distinct (which they are by assumption), which implies this system always has a unique solution.

We could also establish uniqueness by the standard method. Suppose $P_n(x)$ and $Q_n(x)$ are both interpolating polynomials of degree n through the same $n+1$ distinct points. So $P_n(x_i) = Q_n(x_i)$ for $i = 0, 1, \dots n$. Now consider the difference $R_n(x) \coloneqq P_n(x) - Q_n(x)$. $R_n(x)$ is an nth degree polynomial with $n+1$ distinct roots, hence this must be identically zero by Theorem 3.2. So $P_n(x) \equiv Q_n(x)$ and we have uniqueness.

Let's also consider and rule out the other cases. If we have $n+1$ points and we want to fit a polynomial strictly *greater* than degree n through these points, this is clearly not unique. For example if $n = 2$, with $n+1 = 3$, we can fit an infinite number of degree $n+1 = 3$ (and greater) polynomials through $n+1$ points. So we have, we have existence but not uniqueness. If we want to fit a polynomial with degree strictly *less* than degree n, this is generally not possible unless the points have some special structure (all lie on a line, etc). So in this case we don't even have existence.

degree	existence	uniqueness
$< n$		
n	✓	✓
$> n$	✓	

Main Takeaway: Given $n+1$ distinct points, the interpolating polynomial of degree n always exists and is unique.

3.2 Lagrange Interpolation

Now that we've discussed the motivation and theory behind interpolation, let's discuss how to actually construct this polynomial.

Example 3.1: Discovering the Lagrange Polynomial

Find the polynomial of degree at most 2 such that:

$$P(0) = 0, P(-1) = P(1) = 1$$

(Can you guess what this is by inspection?)

Nonsolution: As mentioned before, before this class you would take $P(x) = a_2x^2 + a_1x + a_0$, plug in those three points, get a system like the following

$$\begin{aligned} a_2 - a_1 + a_0 &= 1 \\ a_0 &= 0 \\ a_2 + a_1 + a_0 &= 1 \end{aligned}$$

and solve to get $P(x) = x^2$. For reasons we will see later, it is often useful to have the interpolating polynomial in the Lagrange form or Newton form instead of the standard form $P(x) = a_nx^n + a_{n-1}x^{n-1} + \cdots + a_1x + a_0$.

Solution: Let $x_0 = -1, x_1 = 0, x_2 = 1$. The Lagrange form suggests that we write it like

$$P(x) = L_0(x)f(x_0) + L_1(x)f(x_1) + L_2(x)f(x_2)$$

where we need to find polynomials $L_i(x)$ of degree 2 (or less) such that

$$L_i(x_j) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \tag{3.1}$$

Or to be more explicit in this simple case:

$$L_0(x_0) = 1, L_0(x_1) = 0, L_0(x_2) = 0$$

$$L_1(x_0) = 0, L_1(x_1) = 1, L_1(x_2) = 0$$

$$L_2(x_0) = 0, L_2(x_1) = 0, L_2(x_2) = 1$$

(3.1) says we want to design the Lagrange basis functions $L_i(x)$ to "turn on" at $x = x_i$ and "turn off" elsewhere. The key here is that the $L_i(x)$ are easy to construct manually. Isn't it strange that in order to construct a quadratic interpolant $P(x)$, we have to construct three quadratic interpolants $L_0(x), L_1(x), L_2(x)$?

Let's start with $L_0(x)$. The two conditions $L_0(x_1) = L_0(x_2) = 0$ lead us to guess $L_0(x) \propto (x - x_1)(x - x_2)$. Now we need to enforce that $L_0(x_0) = 1$, but we can't increase the

polynomial degree anymore since it is already of degree 2. So instead, to satisfy the last restriction, we normalize to obtain $L_0(x) = \dfrac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_1)}$. Proceed similarly for the others and we can write out the explicit form of the polynomial:

$$\begin{aligned} P(x) &= L_0(x)\underbrace{f(x_0)}_{1} + L_1(x)\underbrace{f(x_1)}_{0} + L_2(x)\underbrace{f(x_2)}_{1} \\ &= \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} \\ &= \frac{1}{2}x(x-1) + \frac{1}{2}x(x+1) \\ &= x^2 \end{aligned}$$

Comment 1: In general, there is no need to simplify the polynomial to standard form and leaving it in Lagrange form is fine. But here it was very easy to simplify and I wanted to explicitly show the results from the "Nonsolution" and "Solution" were equal..

Comment 2: This result should not be surprising. It's easy to guess at the outset that ***a*** quadratic interpolant could be $P(x) = x^2$, and by uniqueness (Theorem 3.3), this is in fact ***the*** quadratic interpolant.

Here is the general form of the Lagrange polynomial. Given f at $n+1$ distinct points $x_0, x_1, \ldots x_n$, then a unique polynomial $P(x)$ of degree at most n exists with

$$f(x_k) = P(x_k) \text{ for each } k = 0, 1, \ldots n$$

and is given by

$$P(x) = f(x_0)L_{n,0}(x) + \cdots + f(x_n)L_{n,n}(x) = \sum_{k=0}^{n} f(x_k)L_{n,k}(x)$$

where for each $k = 0, 1, \ldots n$

$$\begin{aligned} L_{n,k}(x) &= \frac{(x-x_0)(x-x_1)\cdots(x-x_{k-1})(x-x_{k+1})\cdots(x-x_n)}{(x_k-x_0)(x_k-x_1)\cdots(x_k-x_{k-1})(x_k-x_{k+1})\cdots(x_k-x_n)} \\ &= \prod_{\substack{i=0 \\ i\neq k}}^{n} \frac{(x-x_i)}{(x_k-x_i)}. \end{aligned}$$

Comparison of Taylor Polynomials and Interpolating Polynomials

We have now studied Taylor polynomials and Interpolating polynomials, and both of them seek to approximate the original $f(x)$. How are they different?

- The Taylor polynomial requires knowledge of the derivatives of $f(x)$ whereas the interpolating polynomial only utilizes function evaluations.

- $f(x)$ agrees with the Taylor polynomial at a single point (the expansion point x_0), whereas it agrees with the interpolating polynomial at $n+1$ points.
- We saw that the Taylor polynomial is only a good approximation of $f(x)$ in the radius of convergence centered at the expansion point, but we *hope* that the interpolating polynomial is a good approximation of $f(x)$ on the entire interval $[a, b]$ that contains the interpolation points.
  - Later we are going to see that the latter is easier said than done. Up until now, the interpolation points have always been given. The hard part is how to pick good interpolation points so that the resulting polynomial interpolant is indeed a good approximation.
- In summary:
  - Taylor: Input $a, f(a), f'(a), \ldots f^{(n)}(a)$. Output $T(t) \approx f(t)$, t near a.
  - Interpolant: Input $x_0, x_1, \ldots, x_n$ and $f(x_0), f(x_1) \ldots f(x_n)$ for some well chosen $x_i \in [a, b]$. Output: $P(t) \approx f(t)\ \forall t \in [a, b]$.

3.3 The Error Bound

Every time we approximate something, we should consider how good said approximation is by looking at error bounds. Let's attempt to deduce what the error estimate should look like.

First consider the constant case ($n = 0$). This can be treated by the mean value theorem:

$$f(x) - \underbrace{f(x_0)}_{P(x)} = f'(\xi)(x - x_0)$$

which shows the three properties of the error term:

- zero when f is constant, so error $\propto f'(\xi)$
- zero at $x = x_0$, so error $\propto (x - x_0)$
- $f(x)$ when $f(x) = x - x_0 \implies P(x) = 0$ and $f'(x) = 1$, so proportional to $1/f'(x)$.

We can't use Taylor expansion for the linear case ($n = 1$), but reason it should satisfy

- zero when f is linear, so error $\propto f''(\xi)$
- zero at $x = x_0, x_1$, so error $\propto (x - x_0)(x - x_1)$
- $f(x) = (x - x_0)(x - x_1) \implies P(x) = 0$ so $f(x) = \text{error} = Cf''(\xi)(x - x_0)(x - x_1) \implies C = \frac{1}{2!}$

Theorem 3.4 *Suppose $x_0, x_1, \ldots x_n \in [a, b]$ are $n + 1$ distinct points. Let $f \in C^{n+1}$ and P its polynomial interpolant at the points x_i. Then for each $x \in [a, b]$, there exists $\xi \in (a, b)$ such that*

$$f(x) = P(x) + \frac{f^{n+1}(\xi)}{(n+1)!}(x - x_0)(x - x_1)\cdots(x - x_n)$$

This is the analog of Taylor's theorem with remainder for polynomial interpolation. There is a short but tricky proof in the BFB that proceeds by generalized Rolle's theorem. Here I summarize the purpose each piece of the remainder formula plays and why it is "inevitable".

The "inevitability" of each piece

Consider the three pieces of the error term

$$f(x) - P(x) = \underbrace{\overbrace{f^{n+1}(\xi)}^{①}}_{②}\underbrace{(x - x_0)(x - x_1)\cdots(x - x_n)}_{③}$$

1. If the underlying function $f(x)$ we sample from is actually an nth degree polynomial, then the interpolating polynomial $P_n(x)$ on $n+1$ points must be exactly $f(x)$ by uniqueness, so the error must be zero everywhere. $f^{(n+1)}$ vanishes if f is a degree n polynomial, so the error is identically zero.

2. This is kind of subtle. Suppose $f(x) = x^{n+1}$. Then the LHS $f(x) - P(x)$ must be a polynomial of the form $x^{n+1} + \text{L.O.T}$ (lower order terms), so the error term on the RHS better look like that as well. Since (3) *already* looks like that, we need $\frac{(1)}{(2)}$ to not affect the leading coefficient (we need it to be 1). Since (1) becomes $(n+1)!$, we need (2) to act as a normalizing constant.

3. Since $f(x)$ and $P(x)$ agree at the $n+1$ interpolation points by construction, this product makes the error zero at those points.

The preceding arguments are by no means a rigorous argument, but rather provide a heuristic understanding for the error formula.

Examples

Example 3.2: Error Bound of Lagrange Polynomial I

Consider $f(x) = x^3$ and $x_0 = 1, x_1 = 2, x_2 = 3$. Estimate $f(1.5)$ with a linear interpolating polynomial and find an upper bound on your error.

Solution: We need to pick two points to construct our linear interpolant, and the most appropriate points are x_0 and x_1 because our point of interest $1.5 \in [x_0, x_1] = [1, 2]$. Constructing the Lagrange polynomial:

$$\begin{aligned} P(x) &= \frac{(x-x_1)}{(x_0-x_1)}\underbrace{f(x_0)}_{1} + \frac{(x-x_0)}{(x_1-x_0)}\underbrace{f(x_1)}_{8} \\ &= -(x-2) + (x-1)8 \\ &= 7x - 6 \end{aligned}$$

It's always a good idea to check your work by verifying that this polynomial does interpolate the original function as intended, so confirm$P(1) = f(1) = 1$ and $P(2) = f(2) = 8$. So our estimate is $f(1.5) \approx P(1.5) = 4.5$. Bounding the error term:

$$\begin{aligned} |P(x) - f(x)| &= \left| \frac{f^{(2)}(\xi)}{2!}(x-1)(x-2) \right| \\ &\leq \frac{|f^{(2)}(\xi)|}{2!}|(x-1)(x-2)| \end{aligned}$$

First we bound $|f^{(2)}(\xi)|$ where $\xi \in [1, 2]$. Since $f^{(2)}(x) = 6x, |f^{(2)}(\xi)| \leq 12$ for $\xi \in [1, 2]$. So

plug in $x = 1.5$ to get

$$\begin{aligned} |P(1.5) - f(1.5)| &\leq \frac{12}{2!} |(1.5-1)(1.5-2)| \\ &= 1.5 \end{aligned}$$

The true error is $|P(1.5) - f(1.5)| = |1.5^3 - 4.5| = 1.125$, which is less than our computed error bound.

Example 3.3: Error Bound of Lagrange Polynomial II

Construct the interpolant $P(x)$ to $f(x) = 2^x$ at $x = 0, 1, 2$ and use it to approximate $\sqrt{2}$. Compute the error estimate.

Solution:

$$\begin{aligned} P(x) &= f(x_0)\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} + f(x_1)\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} + f(x_2)\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} \\ &= 1\frac{(x-1)(x-2)}{(0-1)(0-2)} + 2\frac{(x-0)(x-2)}{(1-0)(1-2)} + 4\frac{(x-0)(x-1)}{(2-0)(2-1)} \end{aligned}$$

Then $P(1/2) = 1.375$.

Since $\omega(x) = (x-0)(x-1)(x-2)$, $\omega(1/2) = \frac{3}{8}$ and $f'''(x) = (\ln 2)^3 2^x \implies |f'''(x)| \leq 4(\log 2)^3$, the error estimate is given by

$$|f(x) - P(x)| \leq \frac{4(\log 2)^3}{6}\frac{3}{8} \approx 0.08.$$

The true error is $|\sqrt{2} - P(1/2)| = 0.039$.

3.4 Newton Divided Differences

Let's revisit our original problem: given $f(x)$ and $n+1$ distinct points $x_0, x_1, \dots x_n$, find a degree n interpolating polynomial $P(x)$. We have already discussed the Lagrange form:

$$P(x) = f(x_0)L_{n,0}(x) + \cdots + f(x_n)L_{n,n}(x) = \sum_{k=0}^{n} f(x_k)L_{n,k}(x)$$

Now let's consider an alternative form:

$$P(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_n(x - x_0)\cdots(x - x_{n-1})$$

The reason we like this form is that enforcing $P(x_i) = f(x_i)$ only relies on the first i terms, since the subsequent terms vanish. Let's compute the coefficients a_i.

$$\begin{aligned}
x = x_0 &\implies a_0 = f(x_0) \\
x = x_1 &\implies f(x_1) = \underbrace{a_0}_{f(x_0)} + a_1(x_1 - x_0) \implies a_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \\
x = x_2 &\implies f(x_2) = a_0 + a_1(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1) \\
&\implies a_2 = \frac{f(x_2) - a_0 - a_1(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)} \\
&\implies \text{(By the power of positive thinking, algebra, and some foresight ...)} \\
&\implies a_2 = \frac{\frac{f(x_2)-f(x_1)}{x_2-x_1} - \frac{f(x_1)-f(x_0)}{x_1-x_0}}{x_2 - x_0}
\end{aligned}$$

I claim there is a pattern. If we adopt the terminology below:

$$\begin{aligned}
f[x_i] &\coloneqq f(x_i) \\
f[x_i, x_{i+1}] &\coloneqq \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i} \\
f[x_i, x_{i+1}, x_{i+2}] &\coloneqq \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i} \\
&\cdots \\
f[x_i, x_{i+1}, \dots, x_{i+k}] &\coloneqq \frac{f[x_{i+1}, x_{i+2}, \dots x_{i+k}] - f[x_i, x_{i+1}, \dots x_{i+k-1}]}{x_{i+k} - x_i}
\end{aligned}$$

then we can rewrite the coefficients of the Newton polynomial as:

$$\begin{aligned}
a_0 &= f[x_0] \\
a_1 &= f[x_0, x_1] \\
a_2 &= f[x_0, x_1, x_2] \\
&\cdots \\
a_n &= f[x_0, x_1, x_2, \dots x_n].
\end{aligned}$$

We find these coefficients by setting up a divided difference table (Figure 16) where we compute all the coefficients recursively, and then the coefficients of the polynomial are the ones on the top diagonal.

x	$f(x)$	First divided differences	Second divided differences	Third divided differences
x_0	$f[x_0]$			
		$f[x_0,x_1]=\dfrac{f[x_1]-f[x_0]}{x_1-x_0}$		
x_1	$f[x_1]$		$f[x_0,x_1,x_2]=\dfrac{f[x_1,x_2]-f[x_0,x_1]}{x_2-x_0}$	
		$f[x_1,x_2]=\dfrac{f[x_2]-f[x_1]}{x_2-x_1}$		$f[x_0,x_1,x_2,x_3]=\dfrac{f[x_1,x_2,x_3]-f[x_0,x_1,x_2]}{x_3-x_0}$
x_2	$f[x_2]$		$f[x_1,x_2,x_3]=\dfrac{f[x_2,x_3]-f[x_1,x_2]}{x_3-x_1}$	
		$f[x_2,x_3]=\dfrac{f[x_3]-f[x_2]}{x_3-x_2}$		$f[x_1,x_2,x_3,x_4]=\dfrac{f[x_2,x_3,x_4]-f[x_1,x_2,x_3]}{x_4-x_1}$
x_3	$f[x_3]$		$f[x_2,x_3,x_4]=\dfrac{f[x_3,x_4]-f[x_2,x_3]}{x_4-x_2}$	
		$f[x_3,x_4]=\dfrac{f[x_4]-f[x_3]}{x_4-x_3}$		$f[x_2,x_3,x_4,x_5]=\dfrac{f[x_3,x_4,x_5]-f[x_2,x_3,x_4]}{x_5-x_2}$
x_4	$f[x_4]$		$f[x_3,x_4,x_5]=\dfrac{f[x_4,x_5]-f[x_3,x_4]}{x_5-x_3}$	
		$f[x_4,x_5]=\dfrac{f[x_5]-f[x_4]}{x_5-x_4}$		
x_5	$f[x_5]$			

Figure 16: Newton Divided Difference Table: Interpolating Polynomial

Example 3.4: Newton form

Let $f(x) = x^3$ and $x_0 = 0, x_1 = 1, x_2 = 2$. Construct the Newton form of the Interpolating polynomial:.

Solution: Let's populate the first two columns from the given data.

$$\begin{array}{lll} x_0 = 0 & f[x_0] = 0 & \\ & & f[x_0,x_1] \\ x_1 = 1 & f[x_1] = 1 & \qquad f[x_0,x_1,x_2] \\ & & f[x_1,x_2] \\ x_2 = 2 & f[x_2] = 8 & \end{array}$$

Compute the two divided differences we need from the recursive definition

$$f[x_0,x_1] = \frac{f[x_1]-f[x_0]}{x_1-x_0} = 1, \quad f[x_1,x_2] = \frac{f[x_2]-f[x_1]}{x_2-x_1} = 7$$

and populate the table.

$$\begin{array}{lll} x_0 = 0 & f[x_0] = 0 & \\ & & f[x_0,x_1] = 1 \\ x_1 = 1 & f[x_1] = 1 & \qquad f[x_0,x_1,x_2] \\ & & f[x_1,x_2] = 7 \\ x_2 = 2 & f[x_2] = 8 & \end{array}$$

Then compute the last coefficient:

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} = 3$$

Finish off the table:

$$\begin{array}{llll} x_0 = 0 & f[x_0] = 0 & & \\ & & f[x_0, x_1] = 1 & \\ x_1 = 1 & f[x_1] = 1 & & f[x_0, x_1, x_2] = 3 \\ & & f[x_1, x_2] = 7 & \\ x_2 = 2 & f[x_2] = 8 & & \end{array}$$

So we can read off our polynomial $P(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1)$ as $P(x) = x + 3x(x - 1)$. It is nice to check that $P(x_i) = f(x_i)$ for $i = 0, 1, 2$ as intended.

So this procedure seems to work, but why do this instead of Lagrange? By uniqueness, if I computed the Lagrange form I would have gotten the same polynomial. This also has the same error bound as the Lagrange form of the polynomial.

The advantage here is that if I "forgot" a point x_3, I can easily append it by computing a few additional entries in the divided difference table. This is unlike the Lagrange form where each of the basis functions L_i's have to be recomputed.

Let $x_3 = -2$:

$$\begin{array}{llllll} x_0 = 0 & f[x_0] = 0 & & & & \\ & & f[x_0, x_1] = 1 & & & \\ x_1 = 1 & f[x_1] = 1 & & f[x_0, x_1, x_2] = 3 & & \\ & & f[x_1, x_2] = 7 & & f[x_0, x_1, x_2, x_3] = 1 & \\ x_2 = 2 & f[x_2] = 8 & & f[x_1, x_2, x_3] = 1 & & \\ & & f[x_2, x_3] = 4 & & & \\ x_3 = -2 & f[x_3] = -8 & & & & \end{array}$$

So now our new polynomial looks like: $P(x) = x + 3x(x - 1) + x(x - 1)(x - 2)$, which is the old one with a new term added on. Clearly $P(x_i) = f(x_i)$ for $i = 0, 1, 2$ just like before, because the new term vanishes at those points. So we check that $P(x_3) = f(x_3)$ and we are done.

3.5 Hermite Polynomials

So far we have been studying the following problem: given $f(x)$ and $n+1$ distinct points $x_0, x_1, \ldots x_n$, find a degree n interpolating polynomial $P(x)$. Now we want our polynomial to not only match with the function values at the points, but we also want the first derivatives to match.

Definition: We say that $H(x)$ is the Hermite polynomial of $f(x)$ at $x_0, x_1, \ldots x_n$ if $H(x_i) = f(x_i)$ and $H'(x_i) = f'(x_i)$ at $i = 0, 1 \ldots n$. So there are $2(n+1)$ conditions to satisfy now.

We could use the idea we had when we "discovered" the Lagrange form, where

$$H(x) = \sum_{j=0}^{n} f(x_j) H_j(x) + \sum_{j=0}^{n} f'(x_j) \hat{H}_j(x)$$

where we need to pick $H_j(x)$ and $\hat{H}_j(x)$ so they each "turn on" (equal 1) when evaluated at the desired point and are zero otherwise. This is in principle doable, and explained in the textbook and lecture slides, but in practice is a huge pain to do. The winner will be construction via divided differences.

Clarification: Given $f(x)$ and points $x_0, x_1, \ldots, x_n$, we say that the interpolating polynomial $P(x)$ is the one that agrees with $f(x)$ at those points, i.e. $P(x_i) = f(x_i)$ for $i = 0, 1, \ldots n$. The interpolating polynomial can take two different forms (Lagrange or Newton). But the Hermite polynomial is the actual name of the polynomial that matches at both function values and derivatives. So you can construct either the Lagrange form of the Hermite polynomial or the Newton form of the Hermite polynomial. It's tempting think of Lagrange, Newton, and Hermite as all old dead European guys and hence different flavors of the same thing, but they are not directly comparable.

So now let's consider the Newton form. Let's start from first principles again:

Example 3.5: Hermite polynomial

Find the Hermite polynomial which agrees with $f(x) = x^4$ at $x_0 = 0$ and $x_1 = 1$.

Solution: If this was just a plain old Newton polynomial, we would look for something of the form:

$$P(x) = a_0 + a_1(x - x_0)$$

But now we need something of the form:

$$H(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)^2 + a_3(x - x_0)^2(x - x_1)$$

We need our polynomial to be a cubic since there are four constraints, and hence four degrees of freedom are needed to satisfy them.

But why assume this form? Because it has the same property as the Newton interpolating polynomial that you can impose all the constraints in order and it doesn't affect the following terms. Solving for the coefficients:

$$
\begin{aligned}
H(x_0) = f(x_0) &\implies a_0 = f(x_0) \\
H'(x_0) = f'(x_0) &\implies a_1 = f'(x_0) \text{ (Do you see the usefulness of the squared terms?)} \\
H(x_1) = f(x_1) &\implies f(x_1) = a_0 + a_1(x_1 - x_0) + a_2(x_1 - x_0)^2 \\
&\implies a_2 = \frac{f(x_1) - f(x_0) - f'(x_0)(x_1 - x_0)}{(x_1 - x_0)^2} \\
&\implies a_2 = \frac{\frac{f(x_1)-f(x_0)}{x_1-x_0} - f'(x_0)}{x_1 - x_0} \\
H'(x_1) = f'(x_1) &\implies a_3 = \frac{\frac{f'(x_1)-\frac{f(x_1)-f(x_0)}{x_1-x_0}}{x_1-x_0} - \frac{\frac{f(x_1)-f(x_0)}{x_1-x_0}-f'(x_0)}{x_1-x_0}}{x_1 - x_0} \text{ (lots of algebra)}
\end{aligned}
$$

This is fine and good, but I don't really see a pattern. Enter the divided differences table (Figure 17) again, but for Hermite polynomials.

z	$f(z)$	First divided differences	Second divided differences
$z_0 = x_0$	$f[z_0] = f(x_0)$		
		$f[z_0, z_1] = f'(x_0)$	
$z_1 = x_0$	$f[z_1] = f(x_0)$		$f[z_0, z_1, z_2] = \frac{f[z_1, z_2] - f[z_0, z_1]}{z_2 - z_0}$
		$f[z_1, z_2] = \frac{f[z_2] - f[z_1]}{z_2 - z_1}$	
$z_2 = x_1$	$f[z_2] = f(x_1)$		$f[z_1, z_2, z_3] = \frac{f[z_2, z_3] - f[z_1, z_2]}{z_3 - z_1}$
		$f[z_2, z_3] = f'(x_1)$	
$z_3 = x_1$	$f[z_3] = f(x_1)$		$f[z_2, z_3, z_4] = \frac{f[z_3, z_4] - f[z_2, z_3]}{z_4 - z_2}$
		$f[z_3, z_4] = \frac{f[z_4] - f[z_3]}{z_4 - z_3}$	
$z_4 = x_2$	$f[z_4] = f(x_2)$		$f[z_3, z_4, z_5] = \frac{f[z_4, z_5] - f[z_3, z_4]}{z_5 - z_3}$
		$f[z_4, z_5] = f'(x_2)$	
$z_5 = x_2$	$f[z_5] = f(x_2)$		

Figure 17: Newton Divided Difference Table: Hermite Polynomial

Let's look at this carefully and compare to the Newton table for the interpolating polynomial. The goal is still the same, we need to compute differences, and the top row will give the coefficients of the desired polynomial coefficients and the top row are the ones we want.

- First column (interpolation points) - now has duplicates of each point, and since each x_i shows up twice we label them as z's for clarity.
- Second column (function evaluations) - works the same as before.

- Third column (first divided differences) - is almost the same as before, but now in every other slot there is a derivative. Note we can't compute $f[z_0, z_1]$ like before because this is undefined, so we replace it with $f'(x_0)$. This might seem strange, but we showed via direct computation that this is indeed the right coefficient from above from a_1.
- Remaining columns (second divided differences and beyond) - Compare the a_2 I computed above to $f[z_0, z_1, z_2]$. Now that we have defined the column of first divided differences, this matches exactly.

So hopefully provides some justification for the procedure used to compute the Newton form of the Hermite polynomial. It is really quite similar and only requires slight modification.

We proceed by solving the previous example:
Solution:

$$\begin{array}{llllll} z_0 = 0 & f[z_0] = 0 & & & & \\ & & f[z_0, z_1] & & & \\ z_1 = 0 & f[z_1] = 0 & & f[z_0, z_1, z_2] & & \\ & & f[z_1, z_2] & & f[z_0, z_1, z_2, z_3] & \\ z_2 = 1 & f[z_2] = 1 & & f[z_1, z_2, z_3] & & \\ & & f[z_2, z_3] & & & \\ z_3 = 1 & f[z_3] = 1 & & & & \end{array}$$

Let's populate the first divided differences. So $f[z_0, z_1] = f'(x_0) = 0$ and $f[z_2, z_3] = f'(x_1) = 4$. The remaining entry is a regular difference quotient, $f[z_1, z_2] = \frac{f(x_1)-f(x_0)}{x_1-x_0} = 1$.

$$\begin{array}{llllll} z_0 = 0 & f[z_0] = 0 & & & & \\ & & f[z_0, z_1] = 0 & & & \\ z_1 = 0 & f[z_1] = 0 & & f[z_0, z_1, z_2] & & \\ & & f[z_1, z_2] = 1 & & f[z_0, z_1, z_2, z_3] & \\ z_2 = 1 & f[z_2] = 1 & & f[z_1, z_2, z_3] & & \\ & & f[z_2, z_3] = 4 & & & \\ z_3 = 1 & f[z_3] = 1 & & & & \end{array}$$

And then the rest of them are just regular divided differences, so:

$$\begin{array}{llllll} z_0 = 0 & f[z_0] = 0 & & & & \\ & & f[z_0, z_1] = 0 & & & \\ z_1 = 0 & f[z_1] = 0 & & f[z_0, z_1, z_2] = 1 & & \\ & & f[z_1, z_2] = 1 & & f[z_0, z_1, z_2, z_3] = 2 & \\ z_2 = 1 & f[z_2] = 1 & & f[z_1, z_2, z_3] = 3 & & \\ & & f[z_2, z_3] = 4 & & & \\ z_3 = 1 & f[z_3] = 1 & & & & \end{array}$$

So reading off the top row into the desired for the of the polynomial gets us:

$$H(x) = 0 + 0(x) + 1(x)^2 + 2(x)^2(x-1) = 2x^3 - x^2$$

Now confirm this satisfies the 4 conditions. Recall $f(x) = x^4$.

$$H(0) = f(0) = 0 \checkmark$$
$$H(1) = f(1) = 1 \checkmark$$
$$H'(0) = f'(0) = 0 \checkmark$$
$$H'(1) = f'(1) = 4 \checkmark$$

Suppose I want to add on a point $x_2 = -1$. You could just proceed like we did for the Newton example, and add one two rows at the bottom for $z_4 = z_5 = -1$ and fill out the table.

Now suppose I want to match the function value and first derivative at $x_0 = 0$, and the function value, first derivative, and second derivative at $x_1 = 1$. What would I do? You can formally justify this by writing out the polynomials and solving for the coefficients, but I claim you just need to add another $z_4 = 1$ and fill out some additional entries in the table.

$$\begin{array}{llllllll}
z_0 = 0 & f[z_0] = 0 & & & & & & \\
 & & f[z_0, z_1] = 0 & & & & & \\
z_1 = 0 & f[z_1] = 0 & & f[z_0, z_1, z_2] = 1 & & & & \\
 & & f[z_1, z_2] = 1 & & f[z_0, z_1, z_2, z_3] = 2 & & & \\
z_2 = 1 & f[z_2] = 1 & & f[z_1, z_2, z_3] = 3 & & f[z_0, z_1, z_2, z_3, z_4] & & \\
 & & f[z_2, z_3] = 4 & & f[z_1, z_2, z_3, z_4] & & & \\
z_3 = 1 & f[z_3] = 1 & & f[z_2, z_3, z_4] & & & & \\
 & & f[z_3, z_4] & & & & & \\
z_4 = 1 & f[z_4] = 1 & & & & & &
\end{array}$$

$f[z_3, z_4] = f[z_2, z_3]$. $f[z_2, z_3, z_4]$ is a bit tricky since we run into the same issue where it cannot be computed via divided difference. One might guess we let it be $f''(x_1)$, but it actually has to be $\frac{f''(x_1)}{2}$. (You can formally justify this by solving for the coefficients individually from the form of the polynomial). Then the rest of the table proceeds as usual.

$$\begin{array}{llllllll}
z_0 = 0 & f[z_0] = 0 & & & & & & \\
 & & f[z_0, z_1] = 0 & & & & & \\
z_1 = 0 & f[z_1] = 0 & & f[z_0, z_1, z_2] = 1 & & & & \\
 & & f[z_1, z_2] = 1 & & f[z_0, z_1, z_2, z_3] = 2 & & & \\
z_2 = 1 & f[z_2] = 1 & & f[z_1, z_2, z_3] = 3 & & f[z_0, z_1, z_2, z_3, z_4] = 1 & & \\
 & & f[z_2, z_3] = 4 & & f[z_1, z_2, z_3, z_4] = 3 & & & \\
z_3 = 1 & f[z_3] = 1 & & f[z_2, z_3, z_4] = 6 & & & & \\
 & & f[z_3, z_4] = 4 & & & & & \\
z_4 = 1 & f[z_4] = 1 & & & & & &
\end{array}$$

So we take our polynomial from before and tack on a single term at the end:

$$H(x) = 2x^3 - x^2 + x^2(x-1)^2 = x^4$$

We notice we recover the original polynomial now. This should not be surprising, since the original function was a polynomial of degree 4, and we attempted to construct a polynomial imposing 5 constraints that match this original function, and as a result must recover the original polynomial.

And finally, let me say something about the error form of the Hermite polynomial:

Theorem 3.5 *Suppose $x_0, x_1, \ldots x_n \in [a, b]$ are $n + 1$ distinct points. Let $f \in C^{2n+2}$ and P the Hermite polynomial at the points x_i. Then for each $x \in [a, b]$, there exists $\xi \in (a, b)$ such that*

$$f(x) = H(x) + \frac{f^{(2n+2)}(\xi)}{(2n+2)!}(x - x_0)^2(x - x_1)^2 \ldots (x - x_n)^2$$

4 Chapter 4: Numerical Differentiation and Integration

4.1 Finite Differences

Motivation

Suppose we are given $f(x)$ and a point x_0 and are interested in computing $f'(x_0)$. Typically we would proceed by computing the expression for $f'(x)$ via differentiation rules and plug in x_0. But suppose taking the derivative is too cumbersome, or the common real-world situation where we don't have access to an expression for $f(x)$ and only to some function evaluations of f. Many times you find that you don't have access to the form of $f(x)$ but can query it with any input, or another case is when you have access to f at some points $f(a), f(b), f(c)$ and want to estimate the derivative at some other point $f'(d)$. So we consider an alternative approach for differentiation.

Recall the limit definition of a derivative:

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}$$

Instead of taking the limit as $h \to 0$, just take h to be "small", and we have

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0)}{h} \text{ for } h \text{ small, say } h = 1\text{e}{-6} \tag{4.1}$$

This is known as a **finite difference approximations** to a derivative. Equation (4.1) is known as the forward difference approximation.

So the questions we want to answer are:

- How do you tell whether a finite difference approximation is "good"?
- How do we construct finite difference approximations?
- How do we pick h optimally?

4.1.1 Finite Difference Approximations

In addition to the forward difference approximation presented above, we also have the backward difference approximation:

$$f'(x_0) \approx \frac{f(x_0) - f(x_0 - h)}{h}. \tag{4.2}$$

There is also the centered difference approximation:

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0 - h)}{2h}. \tag{4.3}$$

Figure 18 shows the three approximations for some function.

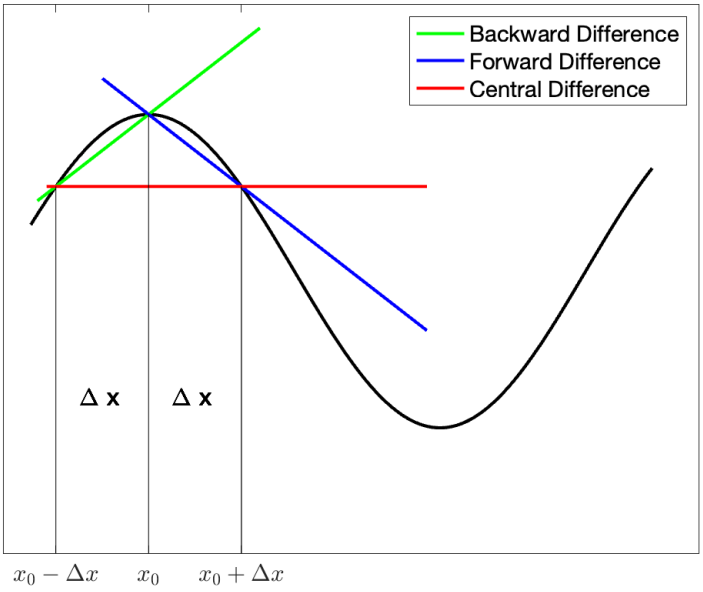


Figure 18: Forward, backward, and central finite difference approximations

The intuition is that the forward and backward difference approximations are comparable in accuracy since they use two points of data and there's no reason to believe "forward" is better than "backward" (or vice versa). Even though the centered difference approximation also uses only two points, we will see it is somehow "better" than the other two. In what sense this is "better" will be explained shortly.

First we need to make a slight modification to the familiar Taylor expansion.

We know that the Taylor expansion of $f(y)$ around y_0 looks like:

$$f(y) = f(y_0) + f'(y_0)(y - y_0) + \frac{f''(y_0)}{2!}(y - y_0)^2 + \frac{f^{(3)}(y_0)}{3!}(y - y_0)^3 + \cdots$$

We can do a change of variable by letting $y_0 = x$ and $y = x + h$. This implies that $y - y_0 = h$, and substitute this all in the above to get:

$$f(x + h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \cdots$$

So we can now use this slightly modified Taylor expansion to evaluate the asymptotic accuracy of our finite difference approximations (4.1) - (4.3):

Forward Difference:

$$\begin{aligned}\frac{f(x_0 + h) - f(x_0)}{h} &= \frac{\cancel{f(x_0)} + hf'(x_0) + \frac{h^2}{2!}f''(\xi) - \cancel{f(x_0)}}{h} \\ &= f'(x_0) + \frac{h}{2!}f''(\xi) = f'(x_0) + O(h)\end{aligned}$$

Backward Difference:

$$\frac{f(x_0) - f(x_0 - h)}{h} = \frac{\cancel{f(x_0)} - (\cancel{f(x_0)} - hf'(x_0) + \frac{h^2}{2!}f''(\xi))}{h}$$
$$= f'(x_0) - \frac{h}{2!}f''(\xi) = f'(x_0) + O(h)$$

Centered Difference:

$$\frac{f(x_0 + h) - f(x_0 - h)}{2h} = \frac{\cancel{f(x_0)} + hf'(x_0) + \cancel{\frac{h^2}{2!}f''(x_0)} + \frac{h^3}{3!}f'''(\xi_1) - (\cancel{f(x_0)} - hf'(x_0) + \cancel{\frac{h^2}{2!}f''(x_0)} - \frac{h^3}{3!}f'''(\xi_2))}{2h}$$
$$= f'(x_0) + \frac{h^2}{3!}f''(\xi_3) = f'(x_0) + O(h^2)$$

So we say that the forward and backward difference estimates are $O(h)$ estimates to $f'(x_0)$ and the centered difference estimate is an $O(h^2)$ estimate. Recall our discussion of rate of convergence of sequences – this means that as $h \to 0$ that the centered difference estimate converges faster to $f'(x_0)$ and hence is a better estimate asymptotically.

Comment: This does not mean for a given f, x_0 and h that the centered difference estimator is always more accurate from an absolute error perspective. Indeed the constants we hide in the big-O notation could certainly be large enough to make any one estimate worse than another. But when we talk about accuracy of finite-difference approximations, we are only concerned with asymptotic accuracy.

Example 4.1: Computing Finite Difference Approximations

Consider $f(x) = e^x, x_0 = 0.1, h = 0.1$. Compute the forward, backward, and centered finite difference approximations. Compute a bound on the error and compare with the true error.

Solution: Forward Difference:

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h} + \frac{h}{2}f''(\xi)$$
$$\approx \frac{e^{0.1} - e^0}{0.1} \approx 1.052$$
$$\frac{h}{2}\max_{\xi \in [0, 0.1]} f''(\xi) = \frac{0.1}{2}e^{0.1} \approx 0.0553$$

Backward Difference:

$$f'(x_0) = \frac{f(x_0) - f(x_0 - h)}{h} + \frac{h}{2}f''(\xi)$$
$$\approx \frac{e^0 - e^{-0.1}}{0.1} \approx 0.9516$$
$$\frac{h}{2}\max_{\xi \in [-0.1, 0]} f''(\xi) = \frac{0.1}{2}e^0 = 0.05$$

Centered Difference:

$$\begin{aligned} f'(x_0) &= \frac{f(x_0+h)-f(x_0-h)}{2h} + \frac{h^2}{3!}f''(\xi) \\ &\approx \frac{e^{0.1}-e^{-0.1}}{0.2} \approx 1.00167 \\ \frac{h^2}{3!}\max_{\xi\in[-0.1,0.1]} f''(\xi) &= \frac{0.1^2}{6}e^{0.1} \approx 0.00184195 \end{aligned}$$

The true value is given by $e^0 = 1$, so in all three cases we look at the true error vs. the computed bound

$$\begin{aligned} |e^0 - 1.052| &= 0.0534617 < 0.0553 \\ |e^0 - 0.9516| &= 0.0483742 < 0.05 \\ |e^0 - 1.00167| &= 0.0016675 < 0.00184195 \end{aligned}$$

Example 4.2: Accuracy of a Finite Difference Formula

Is the following finite difference an accurate approximation to $f'(x_0)$? If so, what is its rate of convergence?

$$f'(x_0) \overset{?}{\approx} \frac{3f(x_0) - 4f(x_0-h) + f(x_0-2h)}{2h}$$

Solution: Let's Taylor expand each component in the numerator around x_0:

$$\begin{aligned} f(x_0) &= f(x_0) \\ f(x_0-h) &= f(x_0) - hf'(x_0) + \frac{h^2}{2!}f''(x_0) - \frac{h^3}{3!}f'''(\xi_1) \\ f(x_0-2h) &= f(x_0) - 2hf'(x_0) + \frac{(2h)^2}{2!}f''(x_0) - \frac{(2h)^3}{3!}f'''(\xi_2) \end{aligned}$$

Plug back into the finite difference approximation:

$$\begin{aligned} &\frac{3f(x_0) - 4f(x_0-h) + f(x_0-2h)}{2h} \\ &= \frac{\cancel{3f(x_0)} - \cancel{4f(x_0)} + 4hf'(x_0) \cancel{-2h^2f''(x_0)} + \frac{2}{3}h^3f'''(\xi_1) + \cancel{f(x_0)} - 2hf'(x_0) + \cancel{2h^2f''(x_0)} - \frac{4}{3}h^3f'''(\xi_2)}{2h} \\ &= f'(x_0) + O(h^2) \end{aligned}$$

So this is a $O(h^2)$ approximation to $f'(x_0)$. Compare to the centered difference approximation which only uses two points for an $O(h^2)$ approximation.

So we've seen that Taylor series are the right tool to address our first question, how to tell whether a finite difference approximation is "good". They are also the right tool to construct finite difference approximations from data (function evaluations).

Example 4.3: Construction of a Finite Difference Formula

Given the values $f(x_0), f(x_0-h), f(x_0-2h)$, construct a finite difference approximation that equals $f'(x_0) + O(h^p)$ where p is large as possible.

Solution: We seek constants c_0, c_1, c_2 such that

$$c_0 f(x_0) + c_1 f(x_0 - h) + c_2 f(x_0 - 2h) = f'(x_0) + O(h^p)$$

Take the Taylor expansions from before and plug them into the LHS. We seek to kill the $f(x_0)$ and $f''(x_0)$ terms, but recover the $f'(x_0)$ term exactly.

$$\begin{aligned} c_0 f(x_0) + c_1 &\left[f(x_0) - h f'(x_0) + \frac{h^2}{2} f''(x_0) + \frac{h^3}{3!} f^{(3)}(\xi_1) \right] \\ + c_2 &\left[f(x_0) - 2h f'(x_0) + 2h^2 f''(x_0) + \frac{8h^3}{3!} f^{(3)}(\xi_2) \right] = f'(x_0) + O(h^p) \end{aligned}$$

This gives us the following linear system:

$$\begin{aligned} c_0 + c_1 + c_2 &= 0 \\ -c_1 - 2c_2 &= \frac{1}{h} \\ \frac{1}{2} c_1 + 2c_2 &= 0 \end{aligned}$$

We can solve this system to get $c_0 = \frac{3}{2h}, c_1 = -\frac{2}{h}, c_2 = \frac{1}{2h}$, which gives us the exact finite difference approximation from the previous example.

Comment: Can you instead construct an estimate for the second derivative $f''(x_0)$?

In general, if we have three data points and three constants, like in the above example, we can hope to satisfy three conditions. Try deriving the forward, backward and centered difference approximations yourself. Even though these each only have two constants, in the centered difference case we were able to get one extra order "for free" without explicitly enforcing it, due to symmetry.

Choice of h

Finally we discuss the choice of the parameter h. Consider the centered difference approximation to the first derivative with the exact error formula:

$$f'(x_0) = \frac{1}{2h} \left[f(x_0 + h) - f(x_0 - h) \right] - \frac{h^2}{6} f^{(3)}(\xi_1)$$

Suppose that in evaluating $f(x_0 + h)$ and $f(x_0 - h)$ we have roundoff errors $e(x_0 + h)$ and $e(x_0 - h)$. So our computed values $\tilde{f}(x_0 + h)$ and $\tilde{f}(x_0 - h)$ are related to the true values $f(x_0 + h)$ and $f(x_0 - h)$ by

$$f(x_0 \pm h) = \tilde{f}(x_0 \pm h) + e(x_0 \pm h).$$

So the total error in the approximation is

$$f'(x_0) - \frac{\tilde{f}(x_0+h) - \tilde{f}(x_0-h)}{2h} = \frac{e(x_0+h) - e(x_0-h)}{2h} - \frac{h^2}{6} f^{(3)}(\xi_1)$$

where the first part is due to round-off error and the second part due to truncation error. If we assume that the round-off errors $e(x_0 \pm h)$ are bounded by some number $\epsilon > 0$ and that the third derivative of f is bounded by a number $M > 0$, then

$$\left| f'(x_0) - \frac{\tilde{f}(x_0+h) - \tilde{f}(x_0-h)}{2h} \right| \leq \frac{\epsilon}{h} + \frac{h^2}{6} M.$$

So we have a tradeoff here. To reduce the truncation error $\frac{h^2}{6}M$, we must reduce h. But as h is reduced, the roundoff error $\frac{\epsilon}{h}$ grows. So this means we cannot choose $h = 10^{-50}$ as we might initially hope to approximate the original limit definition because practical considerations of roundoff error preclude this.

4.1.2 Numerical Differentiation via Polynomial Interpolation

In the previous section, we obtained finite difference formulas by directly constructing linear combinations of data points. In this section, we consider an alternative approach.

Polynomial Interpolation: Given $n+1$ data points $(x_0, y_0), (x_1, y_1), \ldots (x_n, y_n)$, find a degree n polynomial P_n that interpolates the $n+1$ points.
Numerical Differentiation: Given $n+1$ data points $(x_0, f(x_0)), (x_1, f(x_1)), \ldots (x_n, f(x_n))$, approximate $f'(x)$ by a linear combination of $f(x_0), f(x_1), \ldots f(x_n)$.

We connect these two concepts and show how the latter can be used to accomplish the former. This suggests a fairly obvious way to use the former to accomplish the latter: construct the polynomial interpolant $P_n(x)$ through $(x_0, f(x_0)), (x_1, f(x_1)), \ldots (x_n, f(x_n))$ and differentiate. So we are using $P_n'(x) \approx f'(x)$. We can also obtain the error term for the numerical differentiation formula by differentiating the error term from polynomial interpolation.

We repeat the first example by constructing the polynomial interpolant through $f(x_0), f(x_0 - h), f(x_0 - 2h)$:

$$\begin{aligned}
P_2(x) &= f(x_0)\frac{(x-(x_0-h))(x-(x_0-2h))}{(x_0-(x_0-h))(x_0-(x_0-2h))} + f(x_0-h)\frac{(x-x_0)(x-(x_0-2h))}{(x_0-h-x_0)(x_0-h-(x_0-2h))} \\
&\quad + f(x_0-2h)\frac{(x-x_0)(x-(x_0-h))}{(x_0-2h-x_0)(x_0-2h-(x_0-h))} \\
P_2'(x) &= f(x_0)\frac{x-(x_0-h)+x-(x_0-2h)}{2h^2} + f(x_0-h)\frac{x-x_0+x-(x_0-2h)}{-h^2} \\
&\quad + f(x_0-2h)\frac{x-x_0+x-(x_0-h)}{6h^2} \\
P_2'(x_0) &= \frac{3}{2h}f(x_0) - \frac{2}{h}f(x_0-h) + \frac{1}{2h}f(x_0-2h) \approx f'(x_0)
\end{aligned}$$

which matches the previous approach where you set up a linear system.

4.2 Richardson Extrapolation

In the previous section, we saw how we can get high order approximations to derivatives by using more data points. Here we will discuss an alternate way to get high order approximations by evaluating low-order approximations at various h and taking linear combinations of them.

Here is a motivating example: We have as a fact from calculus that

$$e = \lim_{h \to 0} (1+h)^{1/h}$$

So for a small h, we say that $N(h)$ is a good estimate for e.

$$N(h) \coloneqq (1+h)^{1/h} \approx e$$

It can be shown the following expansion

$$N(h) = e + K_1 h + K_2 h^2 + K_3 h^3 + \dots$$

holds for some constants $K_1, K_2, K_3, \dots$, so that $N(h)$ is an $O(h)$ approximation to e.

Let's evaluate this estimator at some h and $h/3$, so we have:

$$\begin{aligned} N(h) &= e + K_1 h + K_2 h^2 + K_3 h^3 + \dots \\ N\left(\frac{h}{3}\right) &= e + K_1 \left(\frac{h}{3}\right) + K_2 \left(\frac{h}{3}\right)^2 + K_3 \left(\frac{h}{3}\right)^3 + \dots \end{aligned}$$

If we wanted to eliminate the $O(h)$ term, we could take a linear combination as follows:

$$3N\left(\frac{h}{3}\right) - N(h) = 2e + K_2' h^2 + K_3' h^3$$

where K_2', K_3' areconstants. Dividing by 2:

$$\frac{3}{2} N\left(\frac{h}{3}\right) - \frac{1}{2} N(h) = e + K_2'' h^2 + K_3'' h^3.$$

We have obtained an $O(h^2)$ of estimate of e by taking a linear combination of two $O(h)$ estimates of e. This is the idea behind Richardson extrapolation.

The typical procedure for extrapolation is as follows: Given an estimator $N(h)$ for a quantity of interest M, evaluate at many values parameter values such as $h, \frac{h}{2}, \frac{h}{4}, \frac{h}{8}, \dots$ and take linear combinations of these in a systematic fashion to get high-order estimates of M (Figure 19).

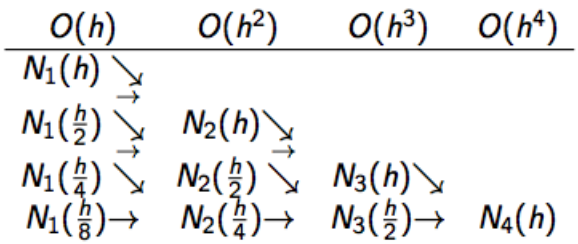


Figure 19: Systematic Richardson Extrapolation

To see this, let's suppose we have a first order estimate $N_1(h)$ of a quantity M (the subscript 1 means first order) and evaluate it at many different values:

$$
\begin{aligned}
N_1(h) &= M + K_1 h + K_2 h^2 + K_3 h^3 + K_4 h^4 + \dots \\
N_1\left(\frac{h}{2}\right) &= M + K_1\left(\frac{h}{2}\right) + K_2\left(\frac{h}{2}\right)^2 + K_3\left(\frac{h}{2}\right)^3 + K_4\left(\frac{h}{2}\right)^4 + \dots \\
N_1\left(\frac{h}{4}\right) &= M + K_1\left(\frac{h}{4}\right) + K_2\left(\frac{h}{4}\right)^2 + K_3\left(\frac{h}{4}\right)^3 + K_4\left(\frac{h}{4}\right)^4 + \dots \\
N_1\left(\frac{h}{8}\right) &= M + K_1\left(\frac{h}{8}\right) + K_2\left(\frac{h}{8}\right)^2 + K_3\left(\frac{h}{8}\right)^3 + K_4\left(\frac{h}{8}\right)^4 + \dots \\
N_1\left(\frac{h}{16}\right) &= M + K_1\left(\frac{h}{16}\right) + K_2\left(\frac{h}{16}\right)^2 + K_3\left(\frac{h}{16}\right)^3 + K_4\left(\frac{h}{16}\right)^4 + \dots
\end{aligned}
$$

Take linear combinations of adjacent evaluations of N_1 to obtain $O(h^2)$ estimates of M:

$$
\begin{aligned}
N_2(h) &= 2N_1\left(\frac{h}{2}\right) - N_1(h) = M - \frac{h^2}{2}K_2 + O(h^3) \\
N_2\left(\frac{h}{2}\right) &= 2N_1\left(\frac{h}{4}\right) - N_1\left(\frac{h}{2}\right) = M - \frac{h^2}{8}K_2 + O(h^3) \\
N_2\left(\frac{h}{4}\right) &= 2N_1\left(\frac{h}{8}\right) - N_1\left(\frac{h}{4}\right) = M - \frac{h^2}{32}K_2 + O(h^3) \\
N_2\left(\frac{h}{8}\right) &= 2N_1\left(\frac{h}{16}\right) - N_1\left(\frac{h}{8}\right) = M - \frac{h^2}{128}K_2 + O(h^3)
\end{aligned}
$$

Continue:

$$
\begin{aligned}
N_3(h) &= \frac{4}{3}N_2\left(\frac{h}{2}\right) - \frac{1}{3}N_2(h) = M + O(h^3) \\
N_3\left(\frac{h}{2}\right) &= \frac{4}{3}N_2\left(\frac{h}{4}\right) - \frac{1}{3}N_2\left(\frac{h}{2}\right) = M + O(h^3) \\
N_3\left(\frac{h}{4}\right) &= \frac{4}{3}N_2\left(\frac{h}{8}\right) - \frac{1}{3}N_2\left(\frac{h}{4}\right) = M + O(h^3)
\end{aligned}
$$

You can imagine how this can keep going in a systematic manner to get arbitrarily high-order estimates. To keep going forward you would need to explicitly find the coefficients for the h^3 term above that I hid in the big-O.

Example 4.4: Richardson Extrapolation: Simple Example

Assume a numerical approximation $N(h)$ to a quantity M satisfies

$$N(h) = M + 3h + 5h^2 + O(h^3) \text{ as } h \to 0$$

and we know three specific values

$$N(h) = N_1, N\left(\frac{h}{2}\right) = N_2, N\left(\frac{h}{3}\right) = N_3$$

Construct an approximation N_{123} such that

$$N_{123}(h) = M + O(h^3)$$

Solution: Plug the three values into the estimate N, the $O(h)$ estimate of M:

$$\begin{aligned} N_1 &= M + 3h + 5h^2 + O(h^3) \\ N_2 &= M + \frac{3}{2}h + \frac{5}{4}h^2 + O(h^3) \\ N_3 &= M + h + \frac{5}{9}h^2 + O(h^3) \end{aligned}$$

Since we seek constants c_0, c_1, c_2 such that $c_1N_1 + c_2N_2 + c_3N_3 = M + O(h^3)$, this gives rise to the system:

$$\begin{aligned} c_0 + c_1 + c_2 &= 1 \\ 3c_1 + \frac{3}{2}c_2 + c_3 &= 0 \\ 5c_1 + \frac{5}{4}c_2 + \frac{5}{9}c_3 &= 0 \end{aligned}$$

So we can solve this to get the constants $c_0 = 0.5, c_1 = -4, c_2 = 4.5$, and so we have the estimate

$$0.5N(h) - 4N\left(\frac{h}{2}\right) + 4.5N\left(\frac{h}{3}\right) = M + O(h^3)$$

Example 4.5: Richardson Extrapolation: High-Order formula

From the first order approximation to $f'(x)$:

$$N(h) := \frac{f(x+h) - f(x)}{h} = f'(x) + O(h)$$

use Richardson's extrapolation to find a 3-point 2nd order formula.

Solution: Taylor expansion:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + O(h^3)$$

First order approximation $N(h)$:

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

Richardson Extrapolation:

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

3 point 2nd order formula:

$$2N(h) - N(2h) = \frac{4f(x+h) - 3f(x) - f(x+2h)}{2h} = f'(x) + O(h^2)$$

4.3 Basic Quadrature

For motivation, recall the limit definition of a definite integral:

Definition 4.1 *Let $[a, b]$ be a closed interval on the real line and a partition of $[a, b]$ be a finite sequence $t_1, t_2, \ldots t_n$ such that*

$$a = x_0 \leq t_1 \leq x_1 \leq t_2 \leq x_2 \leq \cdots \leq x_{n-1} \leq t_n \leq x_n = b.$$

*For convenience we can take an equispaced partition so that $h = \frac{b-a}{n}$. Then a **Riemann sum** of a function with respect to this partition is given by*

$$\int_a^b f(x)\ dx = \lim_{n\to\infty} \frac{(b-a)}{n} \sum_{i=1}^{n} f(t_i)$$

What is this really saying? It says that we break up our domain into a bunch of equispaced pieces (of length $\frac{b-a}{n}$) and attempt to approximate the integral on each piece by a single rectangle. As the number of pieces goes to infinity (or equivalently the size of each piece goes to zero), our approximation approaches the true integral in the limit. This is the motivation of our simplest numerical integration schemes: break up the domain into finitely many pieces, and attempt to approximate the integral on each piece.

In considering the numerical approximation of a definite integral

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

We know f, but cannot find an analytical expression for its antiderivative so our usual notion of definite integration via antiderivatives and the fundamental theorem of calculus fail, for the simple reason that most elementary functions do not have antiderivatives.

Yet the value of this integral (interpreted as area under the curve) certainly still exists and we want to find it. What we do have access to is f at a finite number of points, so we seek to use these function evaluations to numerically approximate the integral I.

4.3.1 Basic Quadrature Rules

The goal is to find weights w_i and nodes (abscissas) x_i such that

$$\int_a^b f(x)\ dx \approx \sum_{i=0}^{n} w_i f(x_i). \tag{4.4}$$

Let's start off with the simplifying assumption that we don't break up the domain at all and try to evaluate the integral on the whole interval $[a, b]$. We use the same idea as in numerical differentiation: replace f by the interpolant P. The idea is that we will replace $f(x)$ with interpolating polynomials $P(x)$ of higher and higher degree, and since polynomials are easy

to integrate, use the result as an approximation for the integral of f.

The simplest thing we could do is replace our function with a degree 0 polynomial (a horizontal line) by interpolating through a single point. We could perhaps choose the left endpoint, the midpoint, or the right endpoint (Figure 20).

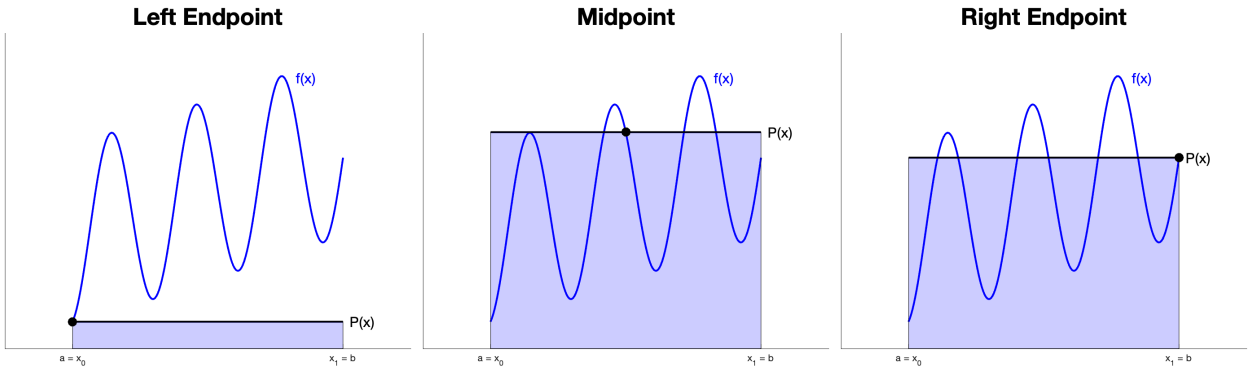


Figure 20: One-point integration rules: left, midpoint, and right.

The next thing we could do is interpolate a degree one polynomial. The natural choice choice is to pick the endpoints of the interval, which is known as the Trapezoid rule (Figure 21).

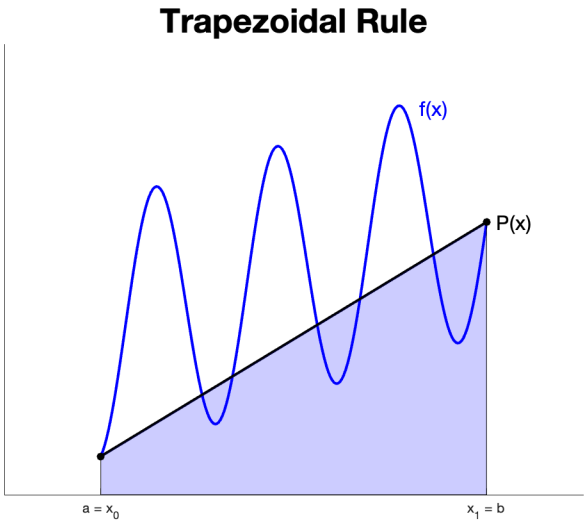


Figure 21: Trapezoid Rule

We could push this idea further and pick a degree two polynomial fit through three equispaced points (left endpoint, midpoint, and right endpoint), and this would be Simpson's rule (Figure 22).

And we can in principle push this idea as far as we wish with higher degree polynomials.

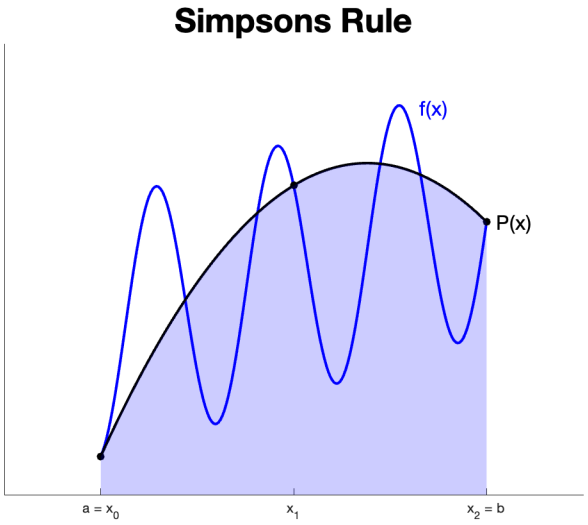


Figure 22: Simpsons' Rule

So our quadrature rule (4.4) becomes:

$$\int_a^b f(x)\ dx \approx \int_a^b P_n(x)\ dx = \int_a^b \sum_{i=0}^{n} f(x_i)L_i(x)\ dx = \sum_{i=0}^{n} w_i f(x_i)$$

where $w_i = \int_a^b L_i(x)\ dx$.

The next natural question is, how good are these approximations? Since we have an expression for error in polynomial interpolation:

$$f(x) = P_n(x) + \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-x_0)(x-x_1)\dots(x-x_n) \quad \xi \in (x_0, x_n),$$

we can simply integrate this expression from a to b w.r.t x to get an approximation error for these integral estimates

$$\int_a^b f(x)\ dx = \int_a^b P(x)\ dx + \int_a^b \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-x_0)(x-x_1)\dots(x-x_n)\ dx \quad \xi \in (x_0, x_n).$$

Let's do this for the Trapezoid Rule (note $x_0 = a, x_1 = b$).

$$f(x) = \underbrace{\frac{x-x_1}{x_0-x_1}f(x_0) + \frac{x-x_0}{x_1-x_0}f(x_1)}_{P(x)} + \underbrace{\frac{1}{2}f''(\xi)(x-x_0)(x-x_1)}_{R(x)}$$

Computing the integral of $P(x)$ gets us the quadrature rule (the Trapezoid rule):

$$\begin{aligned}\int_a^b P(x)\ dx &= \int_a^b \frac{x-x_1}{x_0-x_1}f(x_0) + \frac{x-x_0}{x_1-x_0}f(x_1)\ dx \\ &= \left[\frac{(x-x_1)^2}{2(x_0-x_1)}f(x_0) + \frac{(x-x_0)^2}{2(x_1-x_0)}f(x_1)\right]_{x_0}^{x_1} \\ &= \frac{(x_1-x_0)}{2}[f(x_0)+f(x_1)] \\ &= \frac{h}{2}(f(a)+f(b))\end{aligned}$$

Integrating the error term for interpolation gets us the error term for quadrature:

$$\begin{aligned}\int_a^b R(x)\ dx &= \frac{1}{2}f''(\xi)\int_a^b (x-x_0)(x-x_1)\ dx \\ &= \frac{1}{2}f''(\xi)\int_a^b (x^2-(x_0+x_1)x+x_0x_1)\ dx \\ &= \frac{1}{2}f''(\xi)\left[\frac{x^3}{3} - \frac{(x_1+x_0)}{2}x^2 - x_0x_1x\right]_{x_0}^{x_1} \\ &= -\frac{h^3}{12}f''(\xi)\end{aligned}$$

So the Trapezoid rule (with error) is given by

$$\int_a^b f(x)\ dx = \frac{h}{2}[f(x_0)+f(x_1)] - \frac{h^3}{12}f''(\xi).$$

You can follow an analogous procedure to get Simpson's rule (with error):

$$\int_a^b f(x)\ dx = \frac{h}{3}[f(x_0)+4f(x_1)+f(x_2)] - \frac{h^5}{90}f^{(4)}(\xi).$$

4.3.2 Composite Quadrature Rules

Instead of using one high-order polynomial, we use piecewise polynomial interpolation over the interval. The motivation for this is very similar to the one for cubic splines. We discuss how an integration rule (and its corresponding error term) for an entire interval can be transformed into a composite integration rule.

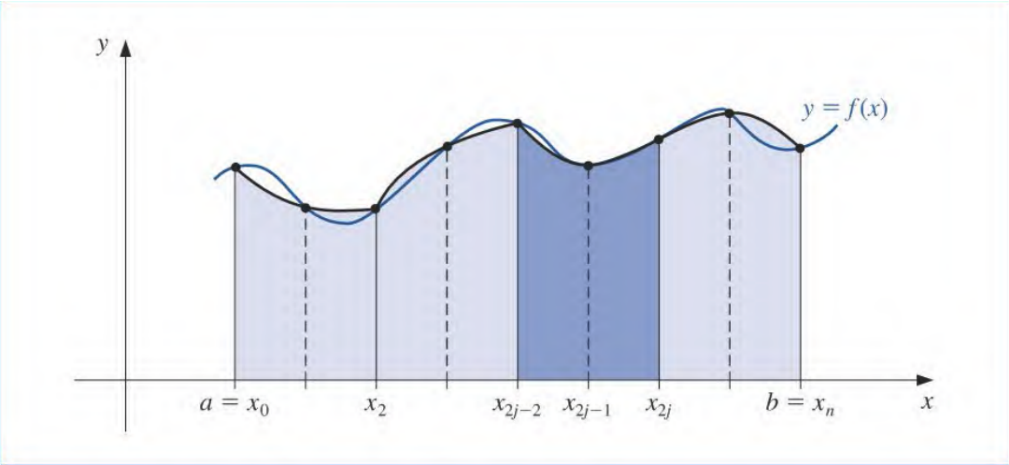
Recall Simpson's rule for the interval is given by:

$$\int_a^b f(x)\ dx \approx \frac{h}{3}\left[f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right]$$

This expression is made exact by inclusion of the error term:

$$\int_a^b f(x)\ dx = \frac{h}{3}\left[f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right] - \frac{h^5}{90}f^{(4)}(\xi) \quad \xi \in (a,b)$$

Let's derive the composite rule by dividing the domain into n equally spaced intervals and applying the quadrature rule on each consecutive pair of subintervals (because of this n must be even).



With $h = \frac{b-a}{n}$ and $x_j = a + jh, j = 0, 1, \ldots, n$, we have that:

$$\begin{aligned}
\int_a^b f(x)\ dx &= \int_{x_0}^{x_2} f(x)\ dx + \int_{x_2}^{x_4} f(x)\ dx + \cdots + \int_{x_{n-2}}^{x_n} f(x)\ dx \\
&= \sum_{j=1}^{n/2} \int_{x_{2j-2}}^{x_{2j}} f(x)\ dx \\
&= \sum_{j=1}^{n/2} \left(\frac{h}{3}[f(x_{2j-2}) + 4f(x_{2j-1}) + f(x_{2j})] - \frac{h^5}{90} f^{(4)}(\xi_j) \right) \quad \xi_j \in (x_{2j-2}, x_{2j}) \\
&= \frac{h}{3} \left[f(x_0) + 2 \sum_{j=1}^{n/2-1} f(x_{2j}) + 4 \sum_{j=1}^{n/2} f(x_{2j-1}) + f(x_n) \right] - \frac{h^5}{90} \sum_{j=1}^{n/2} f^{(4)}(\xi_j)
\end{aligned}$$

That expression in the square brackets simply states a quadrature rule with coefficients $[1, 4, 1]$ for a single interval becomes a composite quadrature rule with coefficients $[1, 4, 2, 4, 2, 4, \ldots, 2, 4, 2, 4, 1]$ on the partition of the interval.

The error form of the composite rule is given by

$$E(f) = -\frac{h^5}{90} \sum_{j=1}^{n/2} f^{(4)}(\xi_j) \quad \xi_j \in (x_{2j-2}, x_{2j})$$

We know by the extreme value theorem that

$$\begin{aligned}
\min_{x \in [a,b]} f^{(4)}(x) \leq f^{(4)}(\xi_j) \leq \max_{x \in [a,b]} f^{(4)}(x) = \frac{n}{2} \min_{x \in [a,b]} f^{(4)}(x) \leq \sum_{j=1}^{n/2} f^{(4)}(\xi_j) \leq \frac{n}{2} \max_{x \in [a,b]} f^{(4)}(x) \\
= \min_{x \in [a,b]} f^{(4)}(x) \leq \frac{2}{n} \sum_{j=1}^{n/2} f^{(4)}(\xi_j) \leq \max_{x \in [a,b]} f^{(4)}(x)
\end{aligned}$$

By the intermediate value theorem applied on $f^{(4)}$, there exists $\mu \in (a,b)$ such that

$$f^{(4)}(\mu) = \frac{2}{n}\sum_{j=1}^{n/2} f^{(4)}(\xi_j)$$

So this implies the error of the composite rule is given by:

$$E(f) = -\frac{h^5}{90}\sum_{j=1}^{n/2} f^{(4)}(\xi_j) = -\frac{h^5}{180} n f^{(4)}(\mu) = -\frac{(b-a)}{180}h^4 f^{(4)}(\mu)$$

Example 4.6: Composite Simpson's rule with error bound

Use the $n = 6$ Simpson's rule to evaluate $\int_0^\pi \sin(x)\,dx$. Then compute a bound on the error using $E = -\frac{b-a}{180}h^4 f^4(\xi)$.

Solution: Since $a = 0, b = \pi$, we have that $h = \frac{b-a}{n} = \frac{\pi}{6}$. So our integral estimate $S(h)$ is given by

$$\begin{aligned} S(h) &= \frac{h}{3}\begin{bmatrix}1 & 4 & 2 & 4 & 2 & 4 & 1\end{bmatrix}\begin{bmatrix} f(x_0) \\ f(x_1) \\ \cdots \\ f(x_6)\end{bmatrix} \\ &= \frac{\pi}{18}\left(0 + 4\left(\frac{1}{2}\right) + 2\left(\frac{\sqrt{3}}{2}\right) + 4(1) + 2\left(\frac{\sqrt{3}}{2}\right) + 4\left(\frac{1}{2}\right) + 0\right) \\ &= \frac{\pi}{18}(2 + \sqrt{3} + 4 + \sqrt{3} + 2) = \frac{\pi}{18}(8 + 2\sqrt{3}) = \frac{\pi}{9}(4 + \sqrt{3}) \end{aligned}$$

As for the error estimate, since $f(x) = \sin(x)$ and $f^{(4)}(x) = \sin(x)$. This gives us that

$$|E| = \left|-\frac{\pi}{180}\left(\frac{\pi}{6}\right)^4 \sin(\xi)\right| \leq \frac{\pi^5}{180(6^4)}$$

Example 4.7: Sufficient accuracy for Simpson's rule

Find an integer n such that the composite Simpson rule has an error $\leq 10^{-6}$ when used to approximate $\int_0^{1.8} \cos(x)\,dx$. (Simpson's rule error: $-\frac{(b-a)}{180}h^4 f^4(\mu)$ for some $\mu \in (a,b)$.

Solution:

$$|E| = \left|\frac{1.8}{180}h^4 f^4(\mu)\right| \leq \frac{h^4}{100} \leq 10^{-6}$$

which gives us

$$h^4 \leq 10^{-4} \implies h \leq 0.1 \implies \frac{1.8}{n} \leq 0.1 \implies \boxed{n \geq 18}$$

4.3.3 Degree of Precision

Definition 4.2 *The **degree of precision** (or accuracy) of a quadrature formula is the largest positive integer n such that the formula holds exactly for x^k for all $k = 0, 1, \ldots n$.*

In the case where the quadrature rule is given by replacing f with a polynomial interpolant P, the degree of precision of the rule is precisely the degree of the interpolant. For example, the trapezoid rule has degree of precision 1, and Simpson's rule has degree of precision 2.

Question: If we take a quadrature rule and extend it to a composite quadrature rule, does its degree of precision change?

Example 4.8: Find the degree of precision

Find the degree of precision of the quadrature formula:

$$\int_{-1}^{1} f(x)\, dx = f\left(-\frac{\sqrt{3}}{3}\right) + f\left(\frac{\sqrt{3}}{3}\right)$$

Solution: Let's start testing monomials of increasing powers:

$$\begin{aligned}
f = 1 &\implies 2 = 1 + 1 && \checkmark \\
f = x &\implies 0 = -\frac{\sqrt{3}}{3} + \frac{\sqrt{3}}{3} && \checkmark \\
f = x^2 &\implies \frac{2}{3} = \frac{1}{3} + \frac{1}{3} && \checkmark \\
f = x^3 &\implies 0 = -\frac{1}{3\sqrt{3}} + \frac{1}{3\sqrt{3}} && \checkmark \\
f = x^4 &\implies \frac{2}{5} = \frac{1}{9} + \frac{1}{9} && \times
\end{aligned}$$

Note all the integrals of the monomials of odd powers are automatically zero, because the integral of an odd function over a symmetric interval is zero. This quadrature rule has degree of precision 3, implying it is exact for all cubics, since every cubic is just a linear combination of $\{1, x, x^2, x^3\}$

Comment: Just because a quadrature rule holds for a higher power (or powers) does not mean it automatically holds for all lower powers. In particular, students frequently tend to forget to check if a quadrature rule holds for $f = 1$ first.

Example 4.9: Construct a quadrature rule with highest degree of precision

Given a quadrature rule of the form:

$$\int_{-1}^{1} f(x)\ dx = w_0 f(-1) + w_1 f(0) + w_2 f(1)$$

Find constants w_0, w_1, w_2 so that this rule has as high a degree of precision as possible. What is the resulting degree of precision?

Solution: We want this rule to be exact for as many monomial powers as possible starting with $f = 1$. So enforcing that gives us:

$$\text{Exact for } f = 1: \implies 2 = w_0 + w_1 + w_2$$

We might expect that since we have three constants, we can enforce three conditions, so:

$$\text{Exact for } f = x: \implies 0 = -w_0 + w_2$$

$$\text{Exact for } f = x^2: \implies \frac{2}{3} = w_0 + w_2$$

Solving this simple 3×3 system gives us $w_0 = \frac{1}{3}, w_1 = \frac{4}{3}, w_2 = \frac{1}{3}$, so our quadrature rule is

$$\int_{-1}^{1} f(x)\ dx = \frac{1}{3} f(-1) + \frac{4}{3} f(0) + \frac{1}{3} f(1)$$

By design this has degree of precision AT LEAST 2, let's just check it for cubics:

$$f = x^3 \implies 0 = -\frac{1}{3} + \frac{1}{3} \quad \checkmark$$

So by magic it does hold for $f = x^3$ even though we didn't design it that way. (The magic here is really symmetry). Now let's check $f = x^4$:

$$f = x^4 \implies \frac{2}{5} = \frac{1}{3} + \frac{1}{3} \quad \times$$

and the magic has run out. So the degree of precision of the quadrature rule is 3.

Example 4.10: Degree of precision of weighted quadrature rule

Let $w(x) = \sqrt{x}$. Find a numerical integration scheme

$$\int_{0}^{1} w(x) f(x)\ dx \approx w_0 f(0) + w_1 f(x_1)$$

that is exact for polynomials $f(x)$ of degree ≤ 2.

Solution: This is very similar to the last problem, except it now its a "weighted integral" with weight function $w(x)$. But nothing changes in how we approach this. We must find the three constants such that this rule is exact for $f = 1, x, x^2$.

Since we want to test this rule for $f(x) = x^k, k = 0, 1, 2$, the LHS always has the form:

$$\int_0^1 x^{k+\frac{1}{2}}\, dx = \frac{1}{k+3/2} x^{k+\frac{3}{2}}\Big|_0^1 = \frac{2}{2k+3}$$

$$\begin{aligned}
\text{Exact for } f = 1: &\implies \frac{2}{3} = w_0 + w_1 \\
\text{Exact for } f = x: &\implies \frac{2}{5} = w_1 x_1 \\
\text{Exact for } f = x^2: &\implies \frac{2}{7} = w_1 x_1^2
\end{aligned}$$

Solving this system gives us $w_0 = \frac{8}{75}, w_1 = \frac{14}{25}, x_1 = \frac{5}{7}$.

Example 4.11: A generalization of the previous

Let $p \in (-1, \infty)$ and $w(x) = x^p$. Find a numerical integration scheme

$$\int_0^1 w(x) f(x)\, dx \approx w_0 f(0) + w_1 f(x_1)$$

that is exact for polynomials $f(x)$ of degree ≤ 2.

Solution: This problem is a slight generalization to the previous one but the weight function seems to trip people up a bit.

Since we want to test this rule for $f(x) = x^k, k = 0, 1, 2$, the LHS always has the form:

$$\int_0^1 x^{k+p}\, dx = \frac{x^{k+p+1}}{k+p+1}\Big|_0^1 = \frac{1}{k+p+1}$$

$$\begin{aligned}
\text{Exact for } f = 1: &\implies \frac{1}{p+1} = w_0 + w_1 \\
\text{Exact for } f = x: &\implies \frac{1}{p+2} = w_1 x_1 \\
\text{Exact for } f = x^2: &\implies \frac{1}{p+3} = w_1 x_1^2
\end{aligned}$$

x_1 is the easiest to solve for first, and you get $x_1 = \frac{p+2}{p+3}$. Then solve for $w_1 = \frac{1}{x_1(p+2)} = \frac{p+3}{(p+2)^2}$. Then finally $w_0 = \frac{1}{p+1} - w_1 = \frac{1}{p+1} - \frac{(p+3)}{(p+2)^2}$. It is nice to check this reduces to the result in the previous question for $p = \frac{1}{2}$.

4.4 Gaussian Quadrature

Up until now, we have used equispaced quadrature nodes (Newton-Cotes) and their composite variants. These are certainly an obvious and simple choice of quadrature rules, but can we do better? A natural question is: "given a certain number of degrees of freedom (weights and nodes), what is the best we can do, in the sense of degree of precision?"

More formally, we can pose the question as follows. For an n point quadrature rule, possibly allowing for non-uniformly spaced nodes:

$$\int_a^b f(x)\ dx \approx \sum_{j=1}^{n} w_j f(x_j) \tag{4.5}$$

We wish to choose weights w_j and nodes x_j to maximize the order of the integration scheme. Heuristics suggest that since there are $2n$ degrees of freedom available, we could hope to choose them to integrate all polynomials of degree $\leq 2n-1$

$$P(x) = a_0 + a_1 x + \cdots + a_{2n-1}x^{2n-1}$$

exactly, which also have $2n$ degrees of freedom (or constraints). This optimal choice of nodes, which we will now derive, are known as Gaussian quadrature rules.

Let's first consider (4.5) in the setting of $n = 2$ and $a = -1, b = 1$. Note that this choice of lower and upper limits of integration is without loss of generality since a definite integral on an arbitrary interval $[a, b]$ can transformed to $[-1, 1]$ via a routine change of variables. So:

$$\int_{-1}^{1} f(x)\ dx \approx w_1 f(x_1) + w_2 f(x_2)$$

We want this to be true for $f(x) = x^k, k = 0, 1, 2, 3$. So the left hand side becomes

$$LHS = \int_{-1}^{1} x^k\ dx = \frac{x^{k+1}}{k+1}\Big|_{-1}^{1} = \begin{cases} 0 & k \text{ odd} \\ \frac{2}{k+1} & k \text{ even} \end{cases}$$

Evaluating the RHS and enforcing equality for $k = 0, 1, 2, 3$ gives us:

$$\begin{aligned} w_1 + w_2 &= 2 \quad (k=0) \\ w_1 x_1 + w_2 x_2 &= 0 \quad (k=1) \\ w_1 x_1^2 + w_2 x_2^2 &= \frac{2}{3} \quad (k=2) \\ w_1 x_1^3 + w_2 x_2^3 &= 0 \quad (k=3) \end{aligned}$$

Solving this system (which is a nontrivial task suggested as an exercise), gives us

$$w_1 = w_2 = 1, \qquad x_1 = -\frac{1}{\sqrt{3}}, x_2 = \frac{1}{\sqrt{3}}$$

Number of points, n	Points, x_i		Weights, w_i	
1	0		2	
2	$\pm\frac{1}{\sqrt{3}}$	±0.57735...	1	
3	0		$\frac{8}{9}$	0.888889...
	$\pm\sqrt{\frac{3}{5}}$	±0.774597...	$\frac{5}{9}$	0.555556...
4	$\pm\sqrt{\frac{3}{7}-\frac{2}{7}\sqrt{\frac{6}{5}}}$	±0.339981...	$\frac{18+\sqrt{30}}{36}$	0.652145...
	$\pm\sqrt{\frac{3}{7}+\frac{2}{7}\sqrt{\frac{6}{5}}}$	±0.861136...	$\frac{18-\sqrt{30}}{36}$	0.347855...
5	0		$\frac{128}{225}$	0.568889...
	$\pm\frac{1}{3}\sqrt{5-2\sqrt{\frac{10}{7}}}$	±0.538469...	$\frac{322+13\sqrt{70}}{900}$	0.478629...
	$\pm\frac{1}{3}\sqrt{5+2\sqrt{\frac{10}{7}}}$	±0.90618...	$\frac{322-13\sqrt{70}}{900}$	0.236927...

Figure 23: Gauss quadrature rules for up to $n = 5$ from Wikipedia

which is the Gaussian quadrature rule for $n = 2$.

The first few Gauss quadrature schemes over $[-1, 1]$ are given by Figure 23.

Setting up these equations and solving these equations can get tedious for large n. It turns out these nodes are the roots of the Legendre polynomials. Suppose that $x_1, x_2, \ldots, x_n$ are the roots of the nth Legendre polynomial $P_n(x)$ and then for each i, the numbers c_i are defined by

$$w_i = \int_{-1}^{1} \prod_{j=1, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}\, dx$$

So this set of rules are known as Gauss-Legendre quadrature rules. Other quadrature rules can be obtained from other families of orthogonal polynomials.

Example 4.12: Estimation via Gauss quadrature

Estimate the following integral with 2-point Gauss quadrature:

$$\int_0^{1/2} e^x\, dx$$

Solution: Since the Gauss quadrature rules are provided over the interval $[-1, 1]$, and we generally want to evaluate a definite integral over some arbitrary interval $[a, b]$, we have two choices. We can either a) transform the Gauss quadrature weights and nodes from the interval $[-1, 1]$ to $[a, b]$. b) transform the limits of integration from $[a, b]$ to $[-1, 1]$ (à la Math 1A) and I claim it is much simpler to do the latter.

As a reminder, two point Gauss quadrature is given by:

$$\int_{-1}^{1} f(x)\ dx \approx f\left(-\frac{1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right)$$

With the change of variables $t = 4x - 1, dt = 4\ dx$:

$$\begin{aligned}\int_0^{1/2} e^x\ dx &= \int_{-1}^{1} \exp\left(\frac{1}{4} + \frac{t}{4}\right)\frac{dt}{4} \\ &\approx \exp\left(\frac{1}{4} - \frac{\sqrt{3}}{12}\right)\frac{1}{4} + \exp\left(\frac{1}{4} + \frac{\sqrt{3}}{12}\right)\frac{1}{4} \approx 0.6487120\end{aligned}$$

As a check, the exact value of the integral is $e^{1/2} - e^0 = 0.6487213$. Not bad, Gauss.

4.5 Multiple Integrals

The basic approach to multiple integrals is to do 1D quadrature in each direction separately

$$I = \int_a^b \underbrace{\int_{c(x)}^{d(x)} f(x,y)\ dy}_{F(x)}\ dx \approx \sum_{i=1}^{n} c_i F(x_i)$$

where

$$F(x_i) \approx \sum_{j=1}^{n} d_{ij} f(x_i, y_{ij}).$$

This gives us the combined scheme of

$$I \approx \sum_{i=1}^{m}\sum_{j=1}^{n} \underbrace{c_i d_{ij}}_{w_{ij}} f(x_i, y_{ij})$$

where we can simply use any composite quadrature rule in an iterated manner. For example, for Trapezoid/Simpson's rule, letting the indices start at $i = 0, j = 0$, the spacing of the grid is given as $h = \frac{b-a}{m}, k_i = \frac{d(x_i)-c(x_i)}{n}$.

Example 4.13: Double Integrals

Use Simpson with $m = n = 2$ to estimate

$$\int_1^2 \int_0^x x^2 y\ dy\ dx$$

Solution: We can take two approaches to this problem. The first is to evaluate each integral individually:

$$\begin{aligned}
\int_1^2 \int_0^x x^2 y\ dy\ dx &\approx \int_1^2 \frac{\Delta y}{3}\left[f(x,0) + 4f\left(x, \frac{x}{2}\right) + f(x,x)\right]\ dx \\
&= \int_1^2 \frac{x}{6}\left[0 + 2x^3 + x^3\right]\ dx \\
&= \int_1^2 \frac{x^4}{2}\ dx \\
&\approx \frac{\Delta x}{3}\left[f(1) + 4f(1.5) + f(2)\right]. \\
&= \frac{1}{6}\left[\frac{1}{2} + \frac{81}{8} + 8\right] = \frac{149}{48} \approx 3.1041\bar{6}
\end{aligned}$$

Alternatively, we could draw the region of integration and deduce the spacing of the grid (Figure 24).

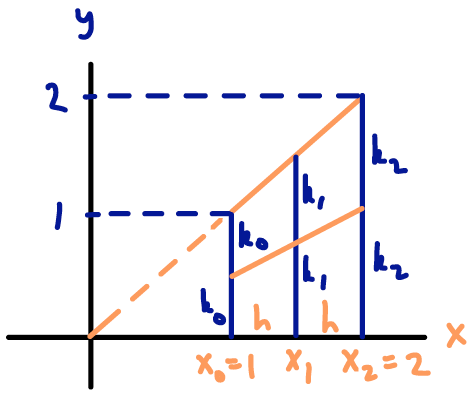


Figure 24: Region of integration for double integral

Then we evaluate the function at these gridpoints:

$$f(x_i, y_{ij}) = \begin{pmatrix} f(1,1) & f(\frac{3}{2},\frac{3}{2}) & f(2,2) \\ f(1,\frac{1}{2}) & f(\frac{3}{2},\frac{3}{4}) & f(2,1) \\ f(1,0) & f(\frac{3}{2},0) & f(2,0) \end{pmatrix} = \begin{pmatrix} 1 & \frac{27}{8} & 8 \\ \frac{1}{2} & \frac{27}{16} & 4 \\ 0 & 0 & 0 \end{pmatrix}$$

Putting it all together gives us

$$\begin{aligned} F_0 &= \frac{k_0}{3}\left[0 + 4\left(\frac{1}{2}\right) + 1\right] = \frac{1}{2} \\ F_1 &= \frac{k_1}{3}\left[0 + 4\left(\frac{27}{16}\right) + \frac{27}{8}\right] = \frac{81}{32} \\ F_2 &= \frac{k_2}{3}\left[0 + 4(4) + 8\right] = 8 \\ I &\approx \frac{h}{3}[F_0 + 4F_1 + F_2] == \frac{149}{48} \approx 3.10416\bar{6} \end{aligned}$$

We note for comparison purposes, that the true value of the integral is given by

$$\int_1^2 \int_0^x x^2 y \; dy \; dx = \int_1^2 \frac{x^2 y^2}{2}\Big|_0^x \; dx = \int_1^2 \frac{x^4}{2} = \frac{x^5}{10}\Big|_1^2 = 3.1.$$

4.6 Improper Integrals

Now we consider the class of improper integrals, where the integral of a function is either considered 1) on a finite interval where the function is unbounded, or 2) on an interval with one more infinite endpoints (Figure 25).

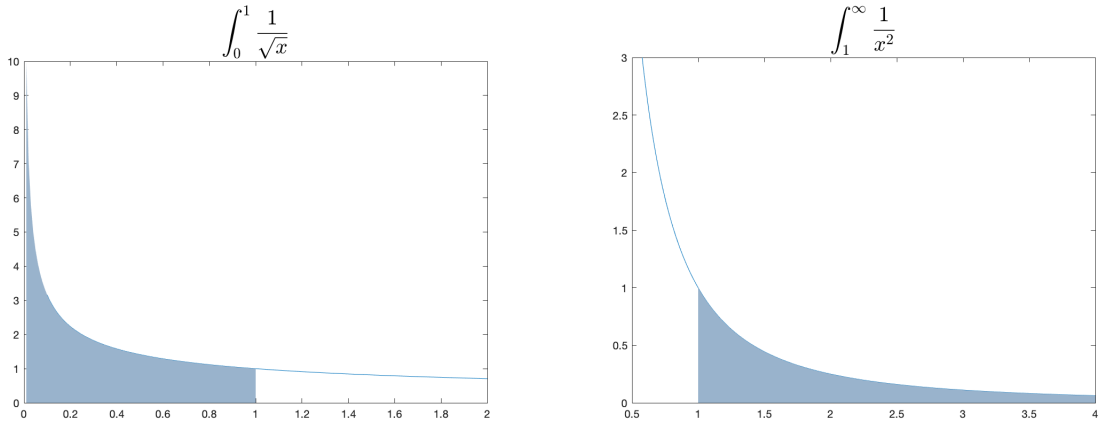


Figure 25: Two types of improper integrals

Both of these integrals are finite because the function blows up slowly enough or decays quickly enough so that the area is finite.

More generally, we consider integrals of the form

$$\int_a^b \frac{g(x)}{(x-a)^p}\,dx \tag{4.6}$$

which exists (and is finite) if $g \in C[a,b]$ and $p \in (0,1)$.

We rely on a key lemma that tells us that if $g(x) \in C^5[a,b]$ and $P_4(x) = \sum_{k=0}^{4} \frac{1}{k!} g^{(k)}(a)(x-a)^4$, then for $0 < p < 1$, we have

$$G(x) = \begin{cases} \frac{g(x)-P_4(x)}{(x-a)^p} & x > a \\ 0 & x = a \end{cases} \in C^4[a,b]$$

So the plan is to evaluate the integral (4.6) by subtracting and adding $P_4(x)$:

$$I = \int_a^b G(x)\,dx + \int_a^b \frac{P_4(x)}{(x-a)^p}\,dx = A + B$$

where we can use Simpson's rule for the first integral since $G \in C^4$ and the second integral can be computed analytically.

Example 4.14: Improper Integrals I

Compute $\int_0^1 \frac{e^{x^2}}{\sqrt{x}}\,dx$ with $n = 8$ for the Simpson calculation.

Solution: Setting up the problem:

$$P_4(x) = 1 + x^2 + \frac{x^4}{2}, G(x) = \frac{e^{x^2} - 1 - x^2 - \frac{x^4}{2}}{\sqrt{x}}$$

Computing the two pieces:

$$A = \int_0^1 G(x)\ dx = \frac{0.125}{3}\,[1\ 4\ 2\ 4\ 2\ 4\ 2\ 4\ 1] \begin{bmatrix} G(0) \\ G(\frac{1}{8}) \\ \vdots \\ G(1) \end{bmatrix} \approx 0.031466$$

$$B = \int_0^1 \frac{P_4(x)}{\sqrt{x}}\ dx = \int_0^1 \left(x^{-1/2} + x^{3/2} + \frac{1}{2}x^{7/2}\right)\ dx = \frac{113}{45} = 2.511111$$

Putting it together gives us:

$$\int_0^1 e^{x^2}\sqrt{x}\ dx \approx A + B = 2.5425770$$

For comparison purposes, Wolfram Alpha gives us

$$\int_0^1 e^{x^2}\sqrt{x}\ dx \approx 2.5425742$$

To deal with integrals over an infinite interval, we do a change of variables to transform the problem to one over a finite interval with a singularity, which reduces the problem to the previous case.

Example 4.15: Improper Integrals II

Approximate the following improper integral:

$$I = \int_1^\infty \frac{\sin\left(\frac{1}{x}\right)}{x^{3/2}}\ dx$$

Solution: Consider the change of variable $z = \frac{1}{x}, dz = -\frac{1}{x^2}\ dx$

$$\int_1^\infty \frac{\sin\left(\frac{1}{x}\right)}{x^{3/2}} = \int_0^1 \sin(z) z^{3/2} z^{-2}\ dz = \int_0^1 \frac{\sin(z)}{\sqrt{z}}\ dz$$

So we have converted this integral over an infinite interval to one over a finite interval with a singularity. We can proceed as before with a Taylor expansion.

$$\begin{aligned}\int_0^1 \frac{\sin(z)}{\sqrt{z}}\ dz &= \int_0^1 \frac{z - \frac{z^3}{6}}{\sqrt{z}}\ dz + \int_0^1 \frac{\sin(z) - \left(z - \frac{z^3}{6}\right)}{\sqrt{z}}\ dz \\ &\approx 0.61904761 + \int_0^1 \frac{\sin(z) - \left(z - \frac{z^3}{6}\right)}{\sqrt{z}}\ dz\end{aligned}$$

Since the first term was treated analytically, the second term can be treated with Composite Simpson's rule with $n = 16$:

$$\int_0^1 \frac{\sin(z) - \left(z - \frac{z^3}{6}\right)}{\sqrt{z}}\, dz \approx 0.00148900097$$

---

5 Chapter 5: Ordinary Differential Equations

Let's spend some time talking about ODEs (no numerical analysis yet). If you're anything like me, all the ODEs you "learned" were from the last thirds of Math 1B and Math 54, where you learned a bag of tricks to find analytical solutions to very special ODEs. In this section, let's refresh and review (and unlearn) some basic theory about ODEs and some analytical solution techniques. You might be interested in this note by Gian-Carlo Rota, a former professor at MIT, titled Ten Lessons I Wish I had Learned Before I Started Teaching Differential Equations.

Definition 5.1 *Consider a function $y(t)$, where y is a function of one variable t. Then an **ordinary differential equation (ODE)** is an equation with containing t, y and derivatives of y.*

Here are some examples:

1. $y' = 0$
2. $\frac{dy}{dt} = ye^t$
3. $y' = 3y + 4$
4. $y'' + 2y' + y = \sin(t)$
5. $y'''y' + e^y y + t\sin(y''y) = y^2 t^y$

Definition 5.2 *The **order of an ODE** is the order of the highest derivative of y that appears in the equation.*

So 1-3 are first order ODEs, 4 is a second order ODE, and 5 is a third order ODE. We will first focus on the study of first order ODEs and numerical methods to solve them. Specifically, we are interested in first order ODEs of the form:

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

For example, even though $(y')e^{y'} = y$ is a first order ODE, it cannot be written in this form.

It is worth contrasting these to PDEs (partial differential equations), which feature functions of multiple variables and their partial derivatives. For example, consider $u(x, t)$, a function of two variables. You might remember the heat equation from the final week of Math 54.

$$u_t(x, t) = u_{xx}(x, t)$$

where the subscripts denote partial derivatives. We won't discuss PDEs in this class (this is covered in 128B), but it is worth knowing that this exists for now.

5.1 Basic Theory of ODEs

5.1.1 Well-Posedness

Definition 5.3 *We say that $f(t, y)$ satisfies a **Lipschitz condition** in the variable y on a set $D \subset R^2$ if a constant $L > 0$ exists with*

$$|f(t, y_1) - f(t, y_2)| \leq L|y_1 - y_2|$$

for all $(t, y_1), (t, y_2) \in D$.

The constant L is called a **Lipschitz constant** for f on this set D.

Example 5.1: Applying the definition of Lipschitz continuity

Show that $f(t, y) = y \sin(t)$ is Lipschitz continuous on the interval $(t \times y) \in [0, \pi/6] \times [0, \infty)$.

Solution: Using the definition:

$$\begin{aligned} |f(t, y_1) - f(t, y_2)| &= |y_1 \sin(t) - y_2 \sin(t)| \\ &= |\sin(t)||y_1 - y_2| \\ &\leq 0.5|y_1 - y_2| \end{aligned}$$

So on the specified domain, this function is Lipschitz continuous with $L = 0.5$. For the domain of $(t \times y) \in [0, \infty) \times [0, \infty)$, this function is Lipschitz continuous with $L = 1$.

Here is an alternative characterization of the Lipschitz constant. Suppose $f(t, y)$ is defined on a convex set $D \in \mathbb{R}^2$. If a constant L exists such that

$$\left|\frac{\partial f}{\partial y}(t, y)\right| \leq L \quad \forall (t, y) \in D$$

then f is Lipschitz continuous on D with Lipschitz constant L.

Proof: By the Mean Value Theorem:

$$\begin{aligned} |f(t, y_1) - f(t, y_2)| &= \left|\frac{\partial f}{\partial y}(t, \xi)(y_1 - y_2)\right| \quad \xi \in (y_1, y_2) \\ &\leq L|y_1 - y_2| \end{aligned}$$

Of course, this can only be applied to $f(t, y)$ that are differentiable. For $f(t, y) = |y|$, you need to directly use Definition 5.3.

A subtle point is that the Lipschitz constant (or whether a function is Lipschitz continuous at all) depends on the domain of interest, just as it does with regular continuity. Consider the initial value problem

$$y'(t) = \sqrt{y} \quad y(0) = 0 \tag{5.1}$$

The function $f(y) = \sqrt{y}$ is not Lipschitz continuous at $y = 0$ since $f'(y) = \frac{1}{2\sqrt{y}} \to \infty$ as $y \to 0$. We cannot find a constant L so that the bound from the definition holds for all y_1, y_2 near 0. This ODE is Lipschitz continuous on any interval that does not include zero.

It is a big theorem in ODEs that Lipschitz continuity of $f(t, y)$ is a sufficient condition for the existence and uniqueness of solutions. Only continuity of $f(t, y)$ is required for the existence of a solution. The proof of this is omitted and standard fare in any ODE course (Math 123). Note that the IVP (5.1) does not have a unique solution, in fact it has two distinct solutions:

$$y(t) \equiv 0 \quad y(t) = \frac{1}{4}t^2$$

Theorem 5.1 *Suppose $D = \{(t, y) \mid a \le t \le b,\ -\infty < y < \infty\}$. If f is continuous and satisfies a Lipschitz condition in the variable y on the set D, then the initial-value problem*

$$y' = f(t, y), \qquad a \le t \le b, \qquad y(a) = \alpha$$

is well-posed.

This is the key theorem we need in order to justify use of numerical methods. If an IVP is well-posed:

- Solutions exist.
- Solutions are unique.
- Solutions behave well under perturbations (they depend continuous on f and α).

5.1.2 Basic Solution Techniques for First-Order ODEs

Let's consider the first order ODEs from the above examples. To solve an ODE means to find a function $y(t)$ that satisfies the equation. For example, if $y(t) = C$ where C is any constant, this is a solution to the ODE $y' = 0$. Let's consider two particular classes of first order ODEs.

Definition 5.4 *A **separable first-order ODE** is of the form:*

$$\frac{dy}{dt} = f(t, y) = g(y)h(t) \implies \frac{1}{g(y)}\, dy = h(t)\, dt$$

Once the variables are separated, you can integrate directly on both sides to solve.

Example 5.2: Separable ODE

Solve the ODE $\frac{dy}{dt} = ye^t$.

Solution: This is a separable ODE.

$$\begin{aligned}
\frac{dy}{dt} = ye^t &\implies \frac{1}{y}dy = e^t dt \\
&\implies \int \frac{1}{y}dy = \int e^t dt \\
&\implies \ln(y) = e^t + C \\
&\implies \boxed{y = e^{e^t+C} = Ce^{e^t}}
\end{aligned}$$

where C is some arbitrary constant. Note it is easy to plug the solution back into the original ODE to check it's correct.

Definition 5.5 *A **linear first-order ODE** is of the form:*

$$y' + p(t)y = g(t)$$

for some functions $p(t)$ and $g(t)$.

Example 5.3: First-Order ODE

Solve the ODE $y' = 3y + 4$.

Solution: This is a linear ODE. We need to multiply both sides of the equation by the integrating factor $I(t) = e^{\int p(t)\ dt} = e^{-3t}$ so the left hand side can be integrated exactly. If you want to review this, I suggest you check out Paul's Online Notes on Linear ODEs.

$$\begin{aligned}
y' - 3y = 4 &\implies e^{-3t}(y' - 3y) = 4e^{-3t} \\
&\implies (e^{-3t}y)' = 4e^{-3t} \\
&\implies e^{-3t}y = -\frac{4}{3}e^{-3t} + C \\
&\implies \boxed{y = -\frac{4}{3} + Ce^{3t}}
\end{aligned}$$

where C is some arbitrary constant. Note it is easy to plug the solution back into the original ODE to check it's correct.

Because of the arbitrary constant C, we can see that the ODE itself does not have a unique solution, but rather a family of solutions. What we need for a unique solution is an additional condition, which is the **initial condition**.

Definition 5.6 *An **IVP (Initial Value Problem)** is given by an ODE with an initial condition*

$$y' = f(t, y) \quad y(t_0) = y_0.$$

Example 5.4: Separable IVP

Solve the IVP $\frac{dy}{dt} = ye^t, \quad y(\ln 2) = e.$

Solution: We showed that $y(t) = Ce^{e^t}$, so let's enforce the initial condition to solve for C.

$$\begin{aligned} y = Ce^{e^t} &\implies e = Ce^{e^{\ln 2}} \\ &\implies e = Ce^2 \\ &\implies C = 1/e \end{aligned}$$

So the unique solution to this IVP is $\boxed{y = e^{e^t - 1}.}$ A different initial condition would result in a different solution curve.

But this is pretty much as far as we can go for exact solutions to first-order IVPs. For example, consider $y' = y^2 \cos(ty)$. This is neither separable nor linear, so how are we going to solve this? Let's switch gears to a different notion of solving an IVP.

5.1.3 Slope Fields

Let's consider the ODE $y' = f(t, y)$ and plot the associated **slope field (also known as direction field)**. What this means is you consider a grid of (t, y) values, and at each point on the grid, plot a line of unit length with slope $f(t, y)$. As an example, here is the slope field for the equation $y' = y$ (Figure 26).

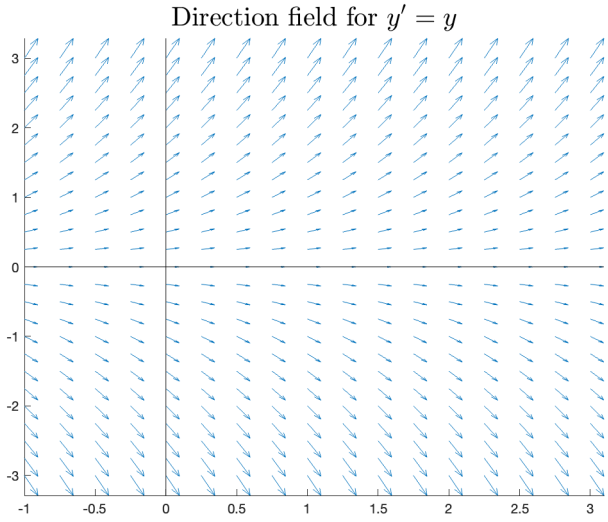


Figure 26: Slope Field for $y' = y$

So let's choose a few initial values $y_0 = -1, 0, 1$ and plot them on the direction field (Figure 27).

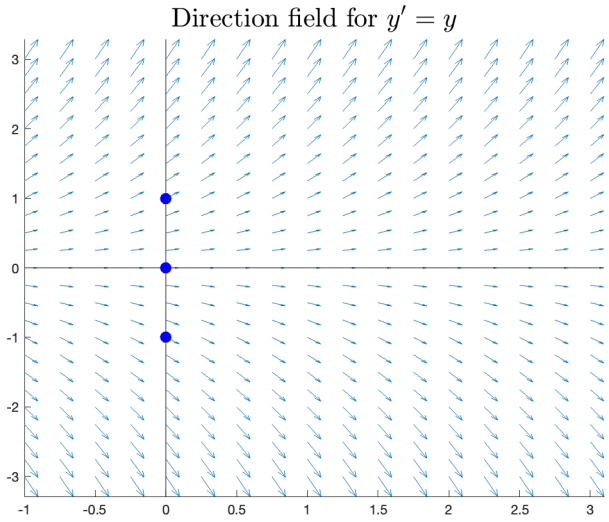


Figure 27: Slope Field for $y' = y$ with initial conditions $y_0 = -1, 0, 1$

At each point (t, y), the value of the slope of the line at that point is precisely the value for y', which tells you how y changes as a function of t. So given the direction field and an initial condition, we can actually trace the corresponding solution through that initial condition. This is how to qualitatively solve IVPs. These slope fields are a way of graphically representing the solutions of a first-order ODE without actually solving the equation. For this ODE, we can compare with the exact solution curves $y = e^t, y = 0$ and $y = -e^t$ (Figure 28).

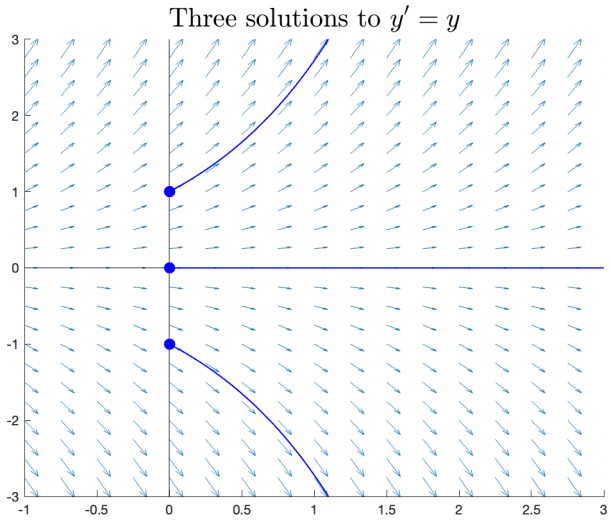


Figure 28: Slope Field for $y' = y$ with three particular solultions

So this is how we plan to solve ODEs that don't have analytical solutions. If we can draw

the direction field for $f(t, y)$, pick an initial condition and trace the curve, then that's the solution. It obviously exists and is there, there just may not be an analytical expression for this solution curve. So how do we plan to describe a curve without an analytical solution? The next best thing is a series of points along the curve.

Let's go back to $y' = y, y(0) = 1$. How are we going to find this series of points? We start at the point $(t_0, y_0) = (0, 1)$, and we know that $y'(t_0) = f(t_0, y_0) = 1$ at that point. So the value of the derivative (the instantaneous slope) is 1 at our initial condition. What we really need to do is follow it for an infinitesimally small time-step, but practically speaking we can't do that. So let's pick a finite time-step $h = 0.5$. So if we follow a slope of 1 for $h = 0.5$, that gives us the new point $(t_1, w_1) = (0.5, 1.5)$. We can now follow the slope at this new point (1.5) with $h = 0.5$ to get another point $(t_2, w_2) = (1, 2.25)$, and continue in this same manner (Figure 29).

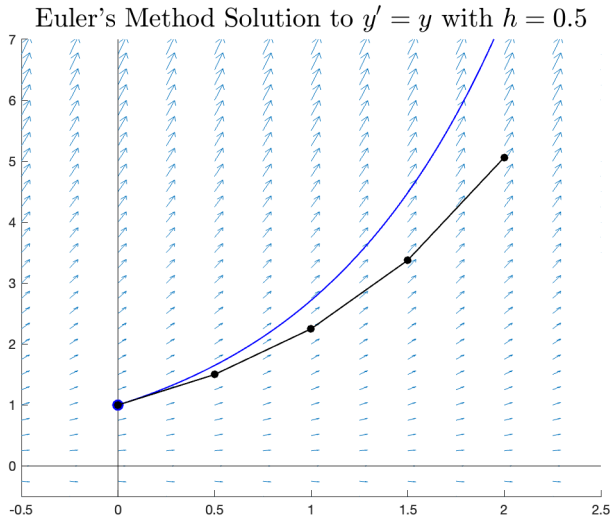


Figure 29: Euler's Method solution for $y' = y$ with $y_0 = 1$ and $h = 0.5$

We are pretty far off from the true solution by $T = 2$. At each time later than t_0, since our trajectory is already off from the true one, we are using the wrong slope to march forward and become even more and more off. Recall what we would have really liked is to use an "infinitesimally" small time-step, but we settled for a rather large timestep of $h = 0.5$. So let's make this even smaller and consider $h = 0.25$ and $h = 0.125$ (Figure 30).

These look much better! In fact it is seems plausible that as $h \to 0$, this series of points we generate will approach the true curve. This series of points we generated is precisely the numerical solution generated using Euler's method.

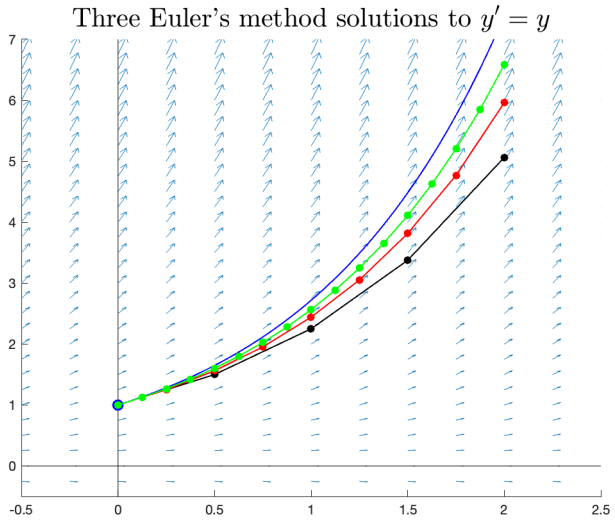


Figure 30: Euler's Method solution for $y' = y$ with $y_0 = 1$ and $h = 0.5, 0.25, 0.125$

5.2 Basic Numerical Methods

5.2.1 Euler's Method

Given an IVP

$$y' = f(t, y), \quad y(t_0) = y_0$$

Euler's method is given by

$$w_{n+1} = w_n + hf(t_n, w_n)$$

where h is a step-size chosen by the user. In the language of numerical differentiation, Euler's method is derived by using a forward difference to approximate $f'(t_n, y_n)$.

From the direction fields above, Euler's method is really a linear approximation between discrete times. We can intuitively see that Euler's method is convergent (under the right circumstances) – more specifically that as $h \to 0$, $w_n \to y(t_n)$, which will be proven later.

Example 5.5: Euler's Method

Given the IVP $y' = y, y(0) = 1$, approximate $y(2)$ using $h = 0.5$.

Solution: This is just the graph from earlier, but now let's use equations to do this:

$$w_1 = w_0 + h \underbrace{f(t_0, w_0)}_{w_0} = 1 + 0.5(1) = 1.5$$

$$w_2 = w_1 + h \underbrace{f(t_1, w_1)}_{w_1} = 1.5 + 0.5(1.5) = 2.25$$

$$w_3 = 2.25 + 0.5(2.25) = 3.375$$

$$w_4 = 5.0625$$

In fact, for this simple ODE we can see that $w_{n+1} = w_n + h\underbrace{f(t_n, w_n)}_{w_n} = (1+h)w_n$, meaning that the numerical solution at each time is just $(1+h)$ times the solution at the previous one.

5.2.2 Convergence and Upper Bound for Error

Consider the IVP

$$y' = f(t, y), \quad y(t_0) = y_0$$

and suppose we are interested in solving the problem from $t_0 < t < T$ for some final time T. Assume that f is continuous and satisfies the Lipschitz condition with some constant L. Furthermore, assume that $M = \max_{t\in[a,b]} |y''(t)|$. Let $w_0, w_1, \ldots w_N$ be the approximations generated by Euler's method for some positive integer N. Then for each $j = 0, 1, \ldots N$, we have that

$$|y(t_j) - w_j| \leq \frac{hM}{2L}\left(e^{L(t_j - t_0)} - 1\right)$$

where $h = \frac{(T-t_0)}{N}$ and $t_j = t_0 + jh$.

This theorem (proved in class/textbook) is somewhat of a mixed bag. The good news is that this shows Euler's method is convergent: as $h \to 0$ then $w_j \to y(t_j)$, which is the bare minimum we need from a numerical method to be useful. At first glance this proof for convergence seems to also give an upper bound for the error. But the problem with this bound, unfortunately, is that in an overwhelming majority of practical cases it is too large by many orders of magnitude.

Prof. Arieh Iserles of Oxford says in his textbook regarding this upper bound that "It falls in the broad category of statements like the distance between London and New York is less than 47 light years, which although manifestly true, fail to contribute significantly to the sum total of human knowledge". Regardless, let's do a problem with it.

Example 5.6: Upper Bound of Error in Euler's method

Let $f(t, y) = e^{-2y}$.
a) Find a Lipschitz condition L for f on the set $D = [0, 1] \times [0, \infty)$. Write the Lipschitz condition explicitly (and precisely) and justify your argument.
b) Compute $M = \max_{0\leq t\leq 1} |y''(t)|$
c) Find a bound on the stepsize h to ensure that the solution of the IVP $y' = e^{-2y}, y(0) = 0$ by Euler's method has error $|e_i| = |y(t_i) - w_i|$ bounded by 0.001 for $0 \leq t_i \leq 1$.

Solution: We need L so that

$$|f(t, y_1) - f(t, y_2)| \leq L|y_1 - y_2| \text{ for all } t \in [0, 1] \text{ and } y_1, y_2 \geq 0$$

By the MVT:

$$|f(t,y_1)-f(t,y_2)| = \left|\frac{\partial f}{\partial y}(t,\xi)(y_1-y_2)\right| \leq L|y_1-y_2|$$

With $L = \max\limits_{y\geq 0}\left|\dfrac{\partial f}{\partial y}\right| = \max\limits_{y\geq 0}|-2e^{-y}| = 2$. Since $y'' = e^{-2y}(-2y') = -2e^{-4y}$, we have $|y''(t)| \leq |-2e^{-4y}| \leq 2 = M$. By the upper bound on the error formula:

$$|e_i| = \frac{hM}{2L}\left(e^{Lt_i}-1\right) \leq \frac{hM}{2L}\left(e^{L}-1\right)$$

that last inequality since $t_i \leq 1$. Plugging in all the numbers we have that

$$|e_i| \leq \frac{h(2)}{(2)(2)}(e^2-1) \leq 0.001 \implies h \leq \frac{0.002}{e^2-1}$$

Local Truncation Error

A general one step method looks like the below:

$$w_{j+1} = w_j + h\phi(t_j, w_j)$$

where in the case of Euler's method, $\phi(t_j, w_j) = f(t_j, w_j)$.

We define the LTE τ_{j+1} as:

$$y_{j+1} = y_j + h\phi(t_j, y_j) + h\tau_{j+1}$$

which tells us by how much the true solution fails to satisfy the numerical scheme. This can also be interpreted as one-step error. Solving for the LTE τ_{j+1}:

$$\tau_{j+1} = \frac{y_{j+1}-y_j}{h} - \phi(t_j, y_j)$$

As an example, for Euler's method:

$$\begin{aligned}
\tau_{j+1} &= \frac{y_{j+1}-y_j}{h} - f(t_j,y_j) \\
&= \frac{\cancel{y_j} + hy_j' + \frac{h^2}{2!}y_j'' + \frac{h^3}{3!}y'''(\xi) - \cancel{y_j}}{h} - f(t_j,y_j) \quad \xi \in (t_j, t_{j+1}) \\
&= \cancel{y_j'} + \frac{h}{2}y_j'' + \frac{h^2}{3!}y'''(\xi) - \cancel{f(t_j,y_j)} \\
&= O(h)
\end{aligned}$$

Euler's method was prominently featured in the climax of the movie Hidden Figures (2016), based on the true story of a team of female African-American mathematicians who served a vital role in NASA during the early years of the U.S. space program. Here is a scene with one of the female mathematicians Katherine Goble Johnson (Taraji P. Henson) and NASA engineer Paul Stafford (Jim Parsons).

Figure 31: Johnson reminds a room full of rocket scientists about Euler's method.

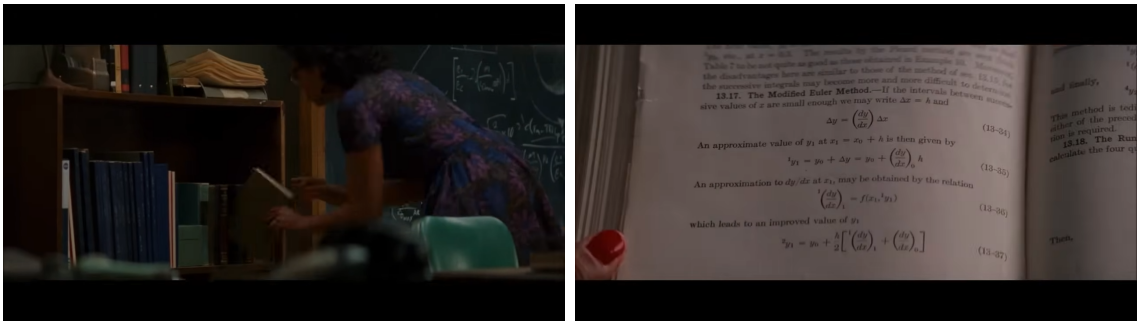


Figure 32: Johnson flips through her old Math 128a textbook.

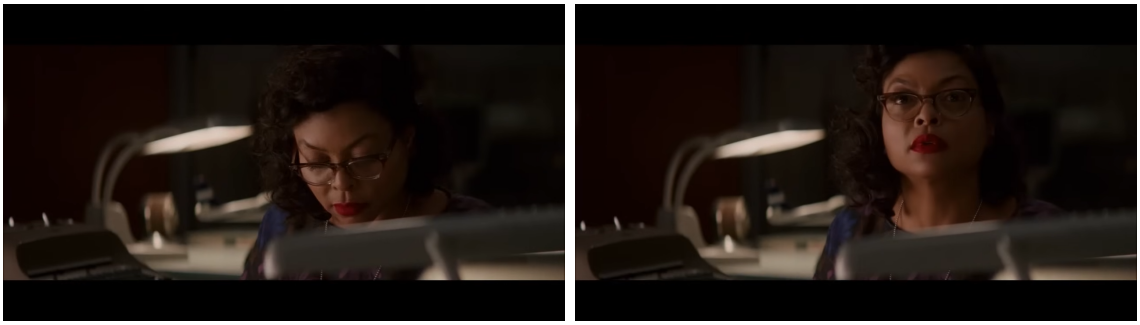


Figure 33: Johnson has her "aha" moment.

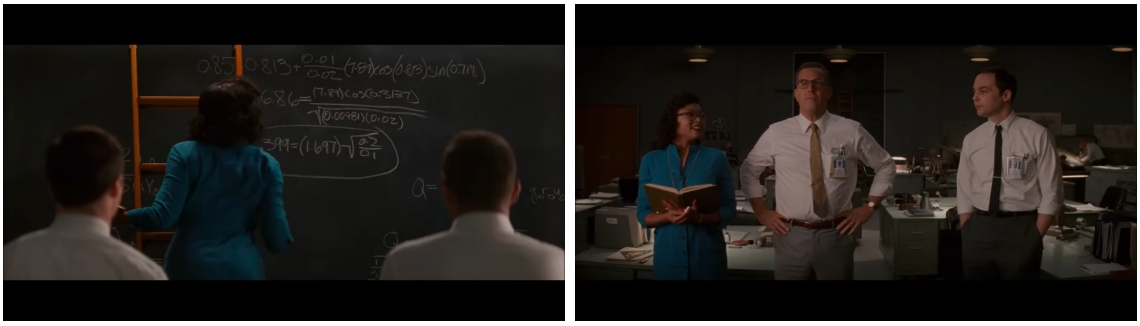


Figure 34: Johnson does one step of Euler's method on the board. They are pleased.

5.2.3 Higher-Order Taylor Methods

Consider the IVP:

$$y' = f(t, y), \quad y(a) = \alpha$$

Previously, we mentioned that Euler's method is a linear approximation between discrete time t_i and t_{i+1}. Now we extend this idea by approximate the solution between discrete times by a higher order Taylor polynomial of degree n at time t_i to arrive at the numerical solution at time t_{i+1} (Figure 35).

So the numerical scheme looks like:

$$\begin{aligned} w_0 &= \alpha \\ w_{i+1} &= w_i + hf(t_i, w_i) + \frac{h^2}{2!} f'(t_i, w_i) + \cdots + \frac{h^n}{n!} f^{(n-1)}(t_i, w_i) \end{aligned}$$

where the IVP gives us that $f^{(k-1)}(t_i, w_i) = y^{(k)}(t_i)$.

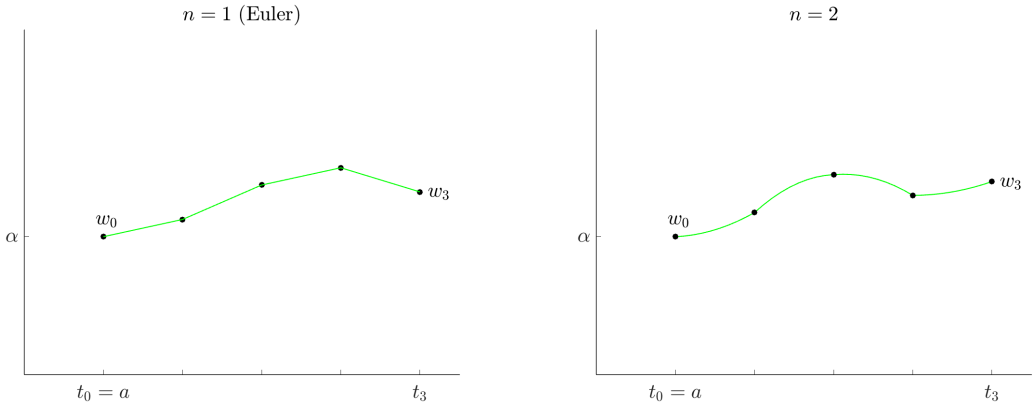


Figure 35: Taylor series method for $n = 1$ (Euler) and $n = 2$

This requires us to compute derivatives:

$$\begin{aligned} y' &= f(t, y) \\ y'' &= f_t(t, y) + f_y(t, y) f(t, y) \\ y''' &= f_{tt} + 2f_{ty} f + f_{yy} f^2 + f_y f_t + f_y^2 f \\ y^{(4)} &= \text{very ugly} \end{aligned}$$

Example 5.7: Taylor method

Derive the third order Taylor method to solve the ODE $y' = t + \sin(y)$.

Solution: Taking the necessary derivatives:

$$\begin{aligned} y' &= t + \sin y \\ y'' &= 1 + (\cos y)y' = 1 + (\cos y)(t + \sin y) \\ y''' &= (-\sin y)(t + \sin y)^2 + (\cos y)(1 + (\cos y)(t + \sin y)) \end{aligned}$$

So the scheme is given by

$$\begin{aligned} w_{i+1} = w_i + h[t_i + \sin w_i] + \frac{h^2}{2}[1 + (\cos w_i)(t_i + \sin w_i)] \\ + \cdots \frac{h^3}{6}[(-\sin w_i)(t_i + \sin w_i)^2 + (\cos w_i)(1 + (\cos w_i)(t_i + \sin w_i))] \end{aligned}$$

Taylor methods have the advantage of being very conceptually straightforward as a way to obtain accurate and high-order numerical methods, but they have three major drawbacks.

- The formulas for $f^{(k)}$ are complicated and hard to code.
- Even if you are able to work out all the formulas, they are often numerically unstable to evaluate. Example:

  $$f(t, y) = \frac{\sin y}{y}, \quad f' = \frac{y \cos y - \sin y}{y^2} \cdot \frac{\sin y}{y}$$

  which is bad near $y = 0$.
- Taylor methods are impractical for systems of differential equations. Consider $y' = f(t, y), y \in \mathbb{R}^d, f : \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}^d$. Then $\frac{\partial f}{\partial y}$ is a matrix when $d \geq 2$, $\frac{\partial^2 f}{\partial y^2}$ is a third rank tensor, etc.

In practice, these methods are rarely used in favor of multistep or Runge-Kutta methods. By construction, an nth order Taylor method has LTE $O(h^n)$.

5.3 Runge-Kutta Methods

5.3.1 Setup

Our motivation for Runge-Kutta methods is that we want to be able to construct numerical methods that have the arbitrarily high-order local truncation error (LTE) we can easily obtain from Taylor series methods without having to compute and evaluate derivatives of f. We only want to use direct function evaluations of f.

A general s-stage Runge-Kutta method has the form:

$$\begin{aligned} k_1 &= f\left(t_n + c_1 h, w_n + h\sum_{i=1}^{s} a_{1i}k_i\right) \\ k_2 &= f\left(t_n + c_2 h, w_n + h\sum_{i=1}^{s} a_{2i}k_i\right) \\ &\vdots \\ k_s &= f\left(t_n + c_s h, w_n + h\sum_{i=1}^{s} a_{si}k_i\right) \\ w_{n+1} &= w_n + h\sum_{i=1}^{s} b_i k_i. \end{aligned} \tag{5.2}$$

This looks rather intimidating. It's worth noting first that what defines the method is the choice of the coefficients a_{ij}, b_i and c_i. This is usually defined in what's called a **Butcher Table**.

$$\begin{array}{c|cccccc} c_1 & a_{11} & a_{12} & a_{13} & & & a_{1s} \\ c_2 & a_{21} & a_{22} & a_{23} & & & a_{2s} \\ c_3 & a_{31} & a_{32} & a_{33} & & & a_{3s} \\ \vdots & & \vdots & & & \ddots & \\ c_s & a_{s1} & a_{s2} & a_{s3} & & & a_{ss} \\ \hline & b_1 & b_2 & b_3 & & \cdots & b_s \end{array}$$

This is a much more convenient way to represent the Runge-Kutta method instead of writing out the individual $k_i's$.

Each of these k_i's are what's known as a **stage derivative**. We call it a derivative because they are evaluations of f at different times and positions, and f is "secretly" a derivative by the ODE $y' = f$. The idea is that we want to take some linear combination of these f's that are evaluated at these various times and positions that will give us the "right update" to march from w_n to w_{n+1}. Any (explicit) one step method has the form

$$w_{n+1} = w_n + h\phi(t_n, w_n)$$

where for a Runge-Kutta method, $\phi(t_n, w_n) = \sum_{i=1}^{s} b_i k_i$. The intuition is that we want $\phi(t_n, w_n)$ to capture the right "slope" between w_n and w_{n+1} so when we multiply it by h, it provides the right update $w_{n+1} - w_n$. This is still rather ambiguous at this point, so let's start from first principles and discuss the simplest Runge-Kutta methods and see how they fit in this framework.

5.3.2 Derivation of Simple Runge-Kutta Methods

Explicit Euler and Implicit Euler

Recall Euler's method: $w_{n+1} = w_n + hf(t_n, w_n)$. The idea we discussed previously with the direction fields in understanding Euler's method was that we just take $f(t_n, w_n)$ – the slope at the left endpoint – and march forward using that. So rewriting this as a Runge-Kutta method:

$$\begin{aligned} k_1 &= f(t_n, w_n) \\ w_{n+1} &= w_n + hk_1. \end{aligned}$$

So Euler's method is a one stage Runge-Kutta method with $a_{11} = 0, b_1 = 1, c_1 = 0$. Here is its Butcher table:

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}$$

Here is an obvious thought – if we are stepping from t_n to t_{n+1} and are only picking one point on the interval to proxy the slope over the entire interval, why the left endpoint? Shouldn't the one on the right be equally good (or bad) in the sense that it is equally representative of what goes on over the whole interval?

This leads us to implicit Euler's method. To clarify, the usual Euler's method goes by the name explicit Euler (or forward Euler). Here we introduce implicit Euler (or backward Euler):

$$\begin{aligned} k_1 &= f(t_{n+1}, w_{n+1}) \\ w_{n+1} &= w_n + hk_1. \end{aligned}$$

But this is not quite in the form of a Runge-Kutta method, because the second argument of the f evaluation in k_1 needs to be expressed as $w_n + \sum_{i=1}^{n} a_{1i} k_i$ for some coefficients a_{1i}. So we rather cleverly substitute the equation for the solution update in the second argument and write $t_{n+1} = t_n + h$ to get

$$\begin{aligned} k_1 &= f(t_n + h, w_n + hk_1) \\ w_{n+1} &= w_n + hk_1. \end{aligned}$$

Here is its Butcher table.

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}$$

Here it is written as a one liner:

$$w_{n+1} = w_n + hf(t_{n+1}, w_{n+1}).$$

Intuitively we consider explicit Euler and implicit Euler as "equally good" since the left or right endpoint should give equally good information about the entire interval. We will quantify this notion later when we see they have the same LTE.

But there's a good reason not to use this method – given the information (t_n, w_n) at time n, the next step w_{n+1} is only implicitly defined by the method! In order to march forward and get a value for w_{n+1}, we must solve a nonlinear equation using something like Newton's method.

Example 5.8: Implicit Euler

Consider the IVP:

$$y' = t + \sin(y) \quad y(0) = 1$$

Estimate $y(0.1)$ with $h = 0.1$ using Implicit Euler.

Solution: One step of Implicit Euler gives us:

$$\begin{aligned} w_1 &= w_0 + hf(\underbrace{t_1}_{0.1}, w_1) \\ &= 1 + (0.1)(0.1 + \sin w_1). \end{aligned}$$

So we need to set up a Newton iteration and iterate to convergence to get w_1. Define $F(x)$ to be:

$$\begin{aligned} F(x) &= x - 1 - (0.1)(0.1 + \sin(x)) = x - 0.1\sin(x) - 1.01 \\ F'(x) &= 1 - 0.1\cos(x). \end{aligned}$$

So the Newton iteration to solve $F(x) = 0$ gives us

$$x_{n+1} = x_n - \frac{x_n - 0.1\sin(x_n) - 1.01}{1 - 0.1\cos(x_n)}.$$

And we can iterate to convergence to get $w_1 = 1.09908$.

As of right now, it seems like there is no good reason to ever use an implicit method if we have an explicit method that is equally "as good" because of the additional cost of solving a nonlinear equation. We will see later in the chapter when we discuss stiffness why for some ODEs, we practically *have to* use implicit methods.

Midpoint Methods

So let's get back on track – we've already discussed explicit Euler and implicit Euler, which came from using the left and the right endpoints, respectively. Continuing down this same train of thought we could also use the midpoint to get the so called "midpoint method":

$$\begin{aligned} k_1 &= f\left(t_{n+\frac{1}{2}}, w_{n+\frac{1}{2}}\right) \\ w_{n+1} &= w_n + hk_1. \end{aligned}$$

But we can't actually use this method as given. We only have w_0 to start from the initial condition. In order to get to w_1, we need the value of $w_{1/2}$!

One way to do this is obtain $w_{1/2}$ by explicit Euler with a stepsize of $h/2$ as $w_{n+1/2} = w_n + \frac{h}{2}f(t_n, w_n)$. Here it is written as a Runge-Kutta method:

$$\begin{aligned} k_1 &= f\left(t_n, w_n\right) \\ k_2 &= f\left(t_n + \frac{h}{2}, w_n + \frac{h}{2}k_1\right) \\ w_{n+1} &= w_n + hk_2 \end{aligned}$$

Here it is as a one-liner:

$$w_{n+1} = w_n + hf\left(t_n + \frac{h}{2}, w_n + \frac{h}{2}f(t_n, w_n)\right).$$

Here is its Butcher table:

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1/2 & 1/2 & 0 \\ \hline & 0 & 1 \end{array}$$

This is the explicit midpoint method (or midpoint method with Euler predictor).

Here's another idea – instead of obtaining $w_{n+1/2}$ by Euler's method, let's use linear interpolation and let $w_{n+1/2} \approx \frac{1}{2}(w_n + w_{n+1})$?

This would give us the implicit midpoint method:

$$w_{n+1} = w_n + hf\left(t_n + \frac{h}{2}, \frac{1}{2}(w_n + w_{n+1})\right).$$

Let's write this as a Runge-Kutta method. This is quite challenging! This is a one-stage method, since there is only one f evaluation. Starting with the general form for a one-stage method:

$$\begin{aligned} k_1 &= f\left(t_n + c_1 h, w_n + a_{11} h k_1\right) \\ w_{n+1} &= w_n + hb_1 k_1. \end{aligned}$$

It's not hard to see that $b_1 = 1$ and $c_1 = \frac{1}{2}$. The question is how do we get the second argument of the f evaluation in the implicit midpoint method $\frac{1}{2}(w_n + w_{n+1})$ to match the second argument in the f evaluation for $k_1, (w_n + a_{11}hk_1)$?

Solve for k_1 in the equation that does the solution update:

$$w_{n+1} = w_n + hb_1k_1 \implies k_1 = \frac{w_{n+1} - w_n}{h}$$

and plug this into the second argument for f for k_1 to get

$$k_1 = f\left(t_n + \frac{1}{2}h, w_n + a_{11}hk_1\right) = f\left(t_n + \frac{1}{2}h, \underbrace{w_n + a_{11}(w_{n+1} - w_n)}_{a_{11}w_{n+1} + (1-a_{11})w_n}\right).$$

But now it's quite clear that $a_{11} = 1/2$, so the implicit midpoint method in Runge-Kutta form is:

$$\begin{aligned} k_1 &= f\left(t_n + \frac{1}{2}h, w_n + \frac{h}{2}k_1\right) \\ w_{n+1} &= w_n + hk_1 \end{aligned}$$

with Butcher table:

$$\begin{array}{c|c} \frac{1}{2} & \frac{1}{2} \\ \hline & 1 \end{array}$$

So that takes care of the one-point rules (left endpoint, right endpoint, and two different midpoint rules). The natural thing to consider next is a two-point rule. The simplest one that comes to mind is taking the average of the left and right endpoint. This is called the trapezoid rule:

$$w_{n+1} = w_n + \frac{h}{2}(f(t_n, w_n) + f(t_{n+1}, w_{n+1})).$$

When we study multistep methods we will see how this is connected (or rather derived from) to the trapezoid rule for integration. Let's write this as a Runge-Kutta method. Since there are two f evaluations we are going to need two stages. Here's my first attempt:

$$\begin{aligned} k_1 &= f\left(t_n, w_n\right) \\ k_2 &= f\left(t_{n+1}, w_{n+1}\right) \\ w_{n+1} &= w_n + \frac{h}{2}(k_1 + k_2). \end{aligned}$$

But this is not quite in Runge-Kutta form because of the w_{n+1}. So let's plug in the equation form the solution update into the second argument of k_2:

$$\begin{aligned} k_1 &= f\left(t_n, w_n\right) \\ k_2 &= f\left(t_n + h, w_n + \frac{h}{2}(k_1 + k_2)\right) \\ w_{n+1} &= w_n + \frac{h}{2}(k_1 + k_2). \end{aligned}$$

Beautiful. Here's the Butcher table:

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1/2 & 1/2 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

What rule comes next? We have discussed implicit and explicit versions of Euler's method and the midpoint method, and we just did the implicit trapezoid rule. So now we consider the explicit trapezoid rule. We can accomplish this by predicting w_{n+1} on the RHS by Euler's method:

$$w_{n+1} = w_n + \frac{1}{2}h(f(t_n, w_n) + f(t_{n+1}, w_n + hf(t_n, w_n))).$$

In Burden and Faires this is also known as modified Euler's method. Let's write this as a Runge-Kutta method:

$$\begin{aligned} k_1 &= f\left(t_n, w_n\right) \\ k_2 &= f\left(t_n + h, w_n + hk_1\right) \\ w_{n+1} &= w_n + \frac{h}{2}(k_1 + k_2) \end{aligned}$$

with Butcher table:

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & 1/2 & 1/2 \end{array}$$

Local Truncation Error

Now the question is how good are these methods, which we for now quantify by the local truncation error. Recall the LTE is the one-step error in the method assuming you have an exact solution up until that point. So for a one step method of the form:

$$w_{n+1} = w_n + h\phi(t_n, w_n).$$

Its LTE is defined by:

$$y_{n+1} = y_n + h\phi(t_n, y_n) + h\tau_{n+1} \implies \tau_{n+1} = \frac{y_{n+1} - y_n}{h} - \phi(t_n, y_n).$$

Let's compute the LTEs for each of these methods we've discussed using this definition.

(Implicit) Euler's method: $w_{n+1} = w_n + hf(t_{n+1}, w_{n+1})$

$$\begin{aligned}
\tau_{n+1} &= \frac{y_{n+1} - y_n}{h} - f(t_{n+1}, y_{n+1}) \\
&= \frac{\cancel{y_n} + hy'_n + \frac{h^2}{2!}y''_n + \frac{h^3}{3!}y'''(\xi) - \cancel{y_n}}{h} - \underbrace{f(t_{n+1}, y_{n+1})}_{y'_{n+1}} \quad \xi \in (t_n, t_{n+1}) \\
&= y'_n + \frac{h}{2}y''_n + \frac{h^2}{3!}y'''(\xi_1) - (y'_n + hy''_n + \frac{h^2}{2!}y^{(3)}(\xi_2)) \\
&= O(h)
\end{aligned}$$

This quantifies our intuition earlier that implicit and explicit Euler should be equally good (or bad).

For the remaining methods we need to be able to do Taylor expansions in 2-variables. Recall the form for 1D. The Taylor expansion of $f(x)$ around a point x_0 looks like:

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \frac{f^{(3)}(x_0)}{3!}(x - x_0)^3 + \dots$$

We can also write this as:

$$f(x + h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f^{(3)}(x) + \dots$$

For 2D, the Taylor Expansion of $f(x, y)$ around a point (x_0, y_0):

$$\begin{aligned}
f(x, y) = f(x_0, y_0) &+ \frac{\partial f}{\partial x}(x_0, y_0)(x - x_0) + \frac{\partial f}{\partial y}(x_0, y_0)(y - y_0) + \frac{1}{2}\frac{\partial^2 f}{\partial x^2}(x_0, y_0)(x - x_0)^2 \\
&+ \frac{\partial^2 f}{\partial x \partial y}(x_0, y_0)(x - x_0)(y - y_0) + \frac{1}{2}\frac{\partial^2 f}{\partial y^2}(x_0, y_0)(y - y_0)^2 + \dots
\end{aligned}$$

Similarly, this can be written as:

$$\begin{aligned}
f(x + \Delta x, y + \Delta y) = f(x, y) &+ \frac{\partial f}{\partial x}(x, y)\Delta x + \frac{\partial f}{\partial y}(x, y)\Delta y + \frac{1}{2}\frac{\partial^2 f}{\partial x^2}(x, y)(\Delta x)^2 \\
&+ \frac{\partial^2 f}{\partial x \partial y}(x, y)(\Delta x)(\Delta y) + \frac{1}{2}\frac{\partial^2 f}{\partial y^2}(x, y)(\Delta y)^2 + \dots
\end{aligned}$$

So now let's use this to calculate the LTE of the explicit midpoint method.

(Explicit) midpoint method: $w_{n+1} = w_n + hf\left(t_n + \frac{h}{2}, w_n + \frac{h}{2}f(t_n, w_n)\right)$

$$\begin{aligned}\tau_{n+1} &= \frac{y_{n+1}-y_n}{h} - f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}f(t_n,y_n)\right)\\ &= y_n' + \frac{h}{2!}y_n'' + \frac{h^2}{3!}y_n^{(3)} + O(h^3) - \Bigg[f(t_n,y_n) + \frac{\partial f}{\partial t}(t_n,y_n)\left(\frac{h}{2}\right)\\ &\quad + \frac{\partial f}{\partial y}(t_n,y_n)\left(\frac{h}{2}\right)f(t_n,y_n) + \frac{\partial^2 f}{\partial t^2}(t_n,y_n)\left(\frac{h}{2}\right)^2 + \frac{\partial^2 f}{\partial t\partial y}(t_n,y_n)\left(\frac{h}{2}\right)^2 f(t_n,y_n)\\ &\quad + \frac{\partial^2 f}{\partial y^2}(t_n,y_n)\left(\frac{h}{2}\right)^2 (f(t_n,y_n))^2 + O(h^3)\Bigg]\end{aligned}$$

Let me just stop here to marvel just how ugly this is before we start chipping away at it. We are going to need to translate y and its derivatives in terms of f (inside the square brackets) to start canceling terms out. Recall from our study of Taylor series methods we have that:

$$\begin{aligned}y' &= f(t,y)\\ y'' &= f_t(t,y) + f_y(t,y)\underbrace{f(t,y)}_{y'}\\ y^{(3)} &= f_{tt} + 2f_{ty}f + f_{yy}f^2 + f_y f_t + f_y^2 f.\end{aligned}$$

So from the first derivative we can see that the y_n' and the $f(t_n,y_n)$ cancel out in the LTE:

$$\begin{aligned}\tau_{n+1} &= \cancel{y_n'} + \frac{h}{2!}y_n'' + \frac{h^2}{3!}y_n^{(3)} + O(h^3) - \Bigg[\cancel{f(t_n,y_n)} + \frac{\partial f}{\partial t}(t_n,y_n)\left(\frac{h}{2}\right)\\ &\quad + \frac{\partial f}{\partial y}(t_n,y_n)\left(\frac{h}{2}\right)f(t_n,y_n) + \frac{\partial^2 f}{\partial t^2}(t_n,y_n)\left(\frac{h}{2}\right)^2 + \frac{\partial^2 f}{\partial t\partial y}(t_n,y_n)\left(\frac{h}{2}\right)^2 f(t_n,y_n)\\ &\quad + \frac{\partial^2 f}{\partial y^2}(t_n,y_n)\left(\frac{h}{2}\right)^2 (f(t_n,y_n))^2 + O(h^3)\Bigg].\end{aligned}$$

Actually, the terms from the second derivative match up too:

$$\begin{aligned}\tau_{n+1} &= \cancel{\frac{h}{2!}y_n''} + \frac{h^2}{3!}y_n^{(3)} + O(h^3) - \Bigg[\cancel{\frac{\partial f}{\partial t}(t_n,y_n)\left(\frac{h}{2}\right) + \frac{\partial f}{\partial y}(t_n,y_n)\left(\frac{h}{2}\right)f(t_n,y_n)}\\ &\quad + \frac{\partial^2 f}{\partial t^2}(t_n,y_n)\left(\frac{h}{2}\right)^2 + \frac{\partial^2 f}{\partial t\partial y}(t_n,y_n)\left(\frac{h}{2}\right)^2 f(t_n,y_n)\\ &\quad + \frac{\partial^2 f}{\partial y^2}(t_n,y_n)\left(\frac{h}{2}\right)^2 (f(t_n,y_n))^2 + O(h^3)\Bigg].\end{aligned}$$

But it doesn't take much effort to see our luck runs out when trying to cancel out the third derivative terms, so this means our LTE is $O(h^2)$. So comparing to the other one point methods, we see that this is actually better. This is consistent with when we studied Newton-Cotes integration rules, that the midpoint method was $O(h^2)$ and the left and right endpoint methods were only $O(h)$.

(Implicit) trapezoid rule: $w_{n+1} = w_n + \frac{h}{2}[f(t_n, w_n) + f(t_{n+1}, w_{n+1})]$

$$\begin{aligned}\tau_{n+1} &= \frac{y_{n+1} - y_n}{h} - \frac{1}{2}\left(\underbrace{f(t_n, y_n)}_{y'_n} + \underbrace{f(t_{n+1}, y_{n+1})}_{y'_{n+1}}\right) \\ &= y'_n + \frac{h}{2}y''_n + \frac{h^2}{3!}y^{(3)}_n + \frac{h^3}{4!}y^{(4)}(\xi_1) - \frac{1}{2}\left(y'_n + y'_n + hy''_n + \frac{h^2}{2!}y^{(3)}_n + \frac{h^3}{3!}y^{(3)}(\xi_2)\right) \\ &= O(h^2)\end{aligned}$$

Example 5.9: LTE of midpoint methods

Compute the LTE of the implicit and explicit midpoint methods.

5.3.3 More on Runge-Kutta Methods

Let's summarize – we have derived the simplest Runge-Kutta methods from purely geometric principles and (painfully) computed their LTEs. But what's the big picture here? The goal at the beginning was to get Runge-Kutta methods of arbitrarily high order, and it seems difficult to milk these simple geometric ideas much further.

Here is a 4th-order RK method:

$$\begin{aligned} k_1 &= f(t_n, w_n) \\ k_2 &= f(t_n + \frac{h}{2}, w_n + \frac{h}{2}k_1) \\ k_3 &= f(t_n + \frac{h}{2}, w_n + \frac{h}{2}k_2) \\ k_4 &= f(t_{n+1}, w_n + hk_3) \\ w_{n+1} &= w_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4). \end{aligned} \tag{5.3}$$

In fact this is frequently known as *the* 4th order RK method called RK4. It's pretty difficult to derive something like this from purely geometric principles, so you can think of how we get RK methods from algebraic techniques starting from the general form (5.2). In principle you could do Taylor expansions to solve for coefficients in order to get as high order a method as possible, but this gets cumbersome fast and practically speaking there are some advanced graph theory techniques used for this, which are outside the scope of this class and covered in Math 228a.

It is also worth making sure we know how to actually use an RK method to step forward:

Example 5.10: Runge-Kutta Order 4

Consider the IVP:

$$y' = t + y \quad y(0) = 1$$

Estimate $y(0.1)$ with $h = 0.1$ using RK4.

Solution:

$$k_1 = f(t_n, w_n) = f(0, 1) = 1$$
$$k_2 = f\left(t_n + \frac{h}{2}, w_n + \frac{h}{2}k_1\right) = f(0.05, 1.05) = 1.1$$
$$k_3 = f\left(t_n + \frac{h}{2}, w_n + \frac{h}{2}k_2\right) = f(0.05, 1.055) = 1.105$$
$$k_4 = f\left(t_{n+1}, w_n + hk_3\right) = f(0.1, 1.05525) = 1.15525$$
$$w_{n+1} = w_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) = 1 + \frac{(0.1)}{6}(1 + 2(1.1) + 2(1.105) + 1.11525) = \boxed{1.16400917}$$

Stage Derivatives vs. Stage Updates The below is how Burden and Faires presents RK4:

$$\begin{aligned} k_1 &= hf(t_n, w_n) \\ k_2 &= hf(t_n + \frac{h}{2}, w_n + \frac{1}{2}k_1) \\ k_3 &= hf(t_n + \frac{h}{2}, w_n + \frac{1}{2}k_2) \\ k_4 &= hf(t_{n+1}, w_n + k_3) \\ w_{n+1} &= w_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned} \tag{5.4}$$

This is clearly equivalent to (5.3), but the difference is that in (5.4) the definition of the k_i's have already absorbed the factor of h. In this convention, the k_i's are known as stage updates and now have the same units as the w's, as opposed to the stage derivatives of (5.3), because now we directly use a linear combination of them to update w_n to w_{n+1}.

Explicit vs. implicit methods For a method to be explicit, this requires $c_1 = 0$ and the A matrix in the Butcher table to be strictly lower triangular (0 on the diagonal and above). We see this means that each of the k_i's can be calculated in terms of all the previous ones, hence there is no need to solve any nonlinear equations for each stage and we can march forward explicitly. So the general form of an RK method (5.2) takes on the following general form for an explicit RK method:

$$
\begin{aligned}
k_1 &= f\left(t_n, w_n\right) \\
k_2 &= f\left(t_n + c_2 h, w_n + h a_{21} k_1\right) \\
k_3 &= f\left(t_n + c_3 h, w_n + h(a_{31} k_1 + a_{32} k_2)\right) \\
&\cdots \\
k_s &= f\left(t_n + c_s h, w_n + h \sum_{i=1}^{s-1} a_{si} k_i\right) \\
w_{n+1} &= w_n + h \sum_{i=1}^{s} b_i k_i.
\end{aligned}
\tag{5.5}
$$

5.4 Multistep Methods

Euler's method and Runge-Kutta methods are all one step methods; all you need to advance forward in time is one previous timestep. That is, to compute w_{i+1} you only need w_i. In a m-step multistep method (5.6), we also make use of m previously computed values of the solution $w_i, w_{i-1}, \ldots, w_{i+1-m}$

$$\begin{aligned} w_{i+1} = a_{m-1}w_i + a_{m-2}w_{i-1} + \cdots + a_0 w_{i+1-m} \\ + h[b_m f_{i+1} + b_{m-1}f_i + \cdots + b_0 f_{i+1-m}] \end{aligned} \tag{5.6}$$

where $f_j = f(t_j, w_j)$. When $b_m = 0$, this is an explicit method. Otherwise $b_m \neq 0$ and the method is implicit.

Advantages:

- Easy to construct high-order methods.
- Inexpensive - we reuse data we have already computed.
  - There is only one new f-evaluation per step, which is the expensive part.
- Theory is interesting (is that really an advantage?)

Disadvantages:

- Schemes are often not very stable and error constants can be large.
  - Have to take smaller steps to achieve the same error bounds as an RK method.
- Starting the method and changing the stepsize can be awkward.
- Unnatural. The theory of ODEs works like a 1-step method.
  - The equation and initial condition uniquely determine the solution. Past values are irrelevant and redundant.

5.4.1 Adams-Bashforth and Adams-Moulton Methods

The m-step Adams-Bashforth methods are an explicit family of methods of the form:

$$w_{i+1} = w_i + h[b_{m-1}f_i + \cdots + b_0 f_{i+1-m}]. \tag{5.7}$$

The method is derived by considering the following identity:

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} \underbrace{f(t, y(t))}_{y'(t)}\, dt$$

and approximating $f(t, y(t))$ by the interpolating polynomial $p(t)$ through $t_{i+1-m}, \ldots, t_i$.

We consider the case of $m = 3$. Constructing the Newton form of the interpolant:

$$\begin{array}{cccc} t_i & f_i & & \\ & & \frac{1}{h}\nabla f_i & \\ t_{i-1} & f_{i-1} & & \frac{1}{2h^2}\nabla^2 f_i \\ & & \frac{1}{h}\nabla f_{i-1} & \\ t_{i-2} & f_{i-2} & & \end{array}$$

$$p(t) = f_i + \frac{1}{h}\nabla f_i(t - t_i) + \frac{1}{2h^2}\nabla^2 f_i(t - t_i)(t - t_{i-1})$$

where

$$\begin{aligned} t_{i-1} &= t_i - h \\ \nabla f_i &= f_i - f_{i-1} \\ \nabla^2 f_i &= f_i - 2f_{i-1} + f_{i-2}. \end{aligned}$$

Integrating the polynomial $p(t)$ with the change of variable $t = t_i + hs, dt = h\ ds$:

$$\begin{aligned} \int_{t_i}^{t_{i+1}} p(t)\ dt &= \int_0^1 f_i + \frac{1}{h}\nabla f_j hs + \frac{1}{2h^2}\nabla^2 f_i(hs)(hs + h)\ ds \\ &= h\left[f_i + \nabla f_i \int_0^1 s\ ds + \frac{1}{2}\nabla^2 f_i \int_0^1 s(s+1)\ ds\right] \\ &= h\left[\left(1 + \frac{1}{2} + \frac{5}{12}\right) f_i - \left(\frac{1}{2} + \frac{5}{6}\right) f_{i-1} + \frac{5}{12} f_{i-2}\right] \end{aligned}$$

results in the method

$$w_{i+1} = w_i + \frac{h}{12}[23f_i - 16f_{i-1} + 5f_{i-2}].$$

The Adams-Moulton methods are constructed in a similar fashion, but we also interpolate through $f_{i+1} = f(t_{i+1}, w_{i+1})$, making them a class of implicit methods.

$$w_{i+1} = w_i + h[b_m f_{i+1} + b_{m-1} f_i + \cdots + b_0 f_{i+1-m}] \tag{5.8}$$

Compared to the m-step Adams-Bashforth methods, we:

- gain an extra order of accuracy.
- increase the stability region (in terms of absolute stability, see next section 5.5).
- but at the additional cost of solving an implicit equation for w_{i+1}.

As before, we consider the case of $m = 2$. Constructing the Newton form of the interpolant:

$$\begin{array}{cccc} t_{i+1} & f_{i+1} & & \\ & & \frac{1}{h}\nabla f_{i+1} & \\ t_i & f_i & & \frac{1}{2h^2}\nabla^2 f_{i+1} \\ & & \frac{1}{h}\nabla f_i & \\ t_{i-1} & f_{i-1} & & \end{array}$$

$$p(t) = f_{i+1} + \frac{1}{h}\nabla f_{i+1}(t - t_{i+1}) + \frac{1}{2h^2}\nabla^2 f_{i+1}(t - t_{i+1})(t - t_i)$$

Integrating the polynomial $p(t)$ with the change of variable $t = t_i + hs, dt = h\ ds$:

$$\begin{aligned}\int_{t_i}^{t_{i+1}} p(t)\ dt = \int_0^1 p(t_i + hs)h\ ds &= h\left[f_{i+1} + \nabla f_{i+1}\int_0^1 (s-1)\ ds + \frac{1}{2}\nabla^2 f_{i+1}\int_0^1 (s-1)s\ ds\right] \\ &= h\left[f_{i+1} - \frac{1}{2}\nabla f_{i+1} - \frac{1}{12}\nabla^2 f_{i+1}\right] \\ &= h\left[\left(1 - \frac{1}{2} - \frac{1}{12}\right)f_{i+1} + \left(\frac{1}{2} + \frac{1}{6}\right)f_i - \frac{1}{12}f_{i-1}\right] \\ &= h\left[\frac{5}{12}f_{i+1} + \frac{8}{12}f_i - \frac{1}{12}f_{i-1}\right]\end{aligned}$$

results in the method

$$w_{i+1} = w_i + \frac{h}{12}[5f_{i+1} + 8f_i - f_{i-1}].$$

5.4.2 Consistency of Multistep Methods

Given the IVP

$$y' = f(t, y), \quad y(t_0) = y_0,$$

the m-step multistep method

$$\begin{aligned}w_{i+1} = a_{m-1}w_i + a_{m-2}w_{i-1} + \cdots + a_0 w_{i+1-m} \\ + h[b_m f_{i+1} + b_{m-1}f_i + \cdots + b_0 f_{i+1-m}]\end{aligned}$$

has local truncation error (LTE)

$$\begin{aligned}\tau_{i+1}(h) = \frac{y(t_{i+1}) - a_{m-1}y(t_i) - \cdots - a_0 y(t_{i+1-m})}{h} \\ - [b_m f(t_{i+1}, y(t_{i+1})) + \cdots + b_0 f(t_{i+1-m}, y(t_{i+1-m}))].\end{aligned}$$

We say that a method is consistent if $\tau_{i+1}(h) = O(h^p), p \geq 1$, so that the local truncation error goes to zero faster than the step size h as $h \to 0$.

5.4.3 Stability of Multistep Methods

Stability means that small perturbations introduced along the way do not grow unconditionally. The easiest case is given by

$$y' = 0, \qquad y(0) = 0.$$

Stability will be defined from this case. So the generic scheme becomes:

$$\begin{aligned}w_{i+1} = a_{m-1}w_i + a_{m-2}w_{i-1} + \cdots + a_0 w_{i+1-m} \\ + h[b_m \underbrace{f_{i+1}}_{0} + \cdots + b_0 \underbrace{f_{i+1-m}}_{0}] = 0.\end{aligned}$$

For simplicity, set $i = n+m-1$ for a change of index to obtain the linear difference equation

$$w_{n+m} - a_{m-1}w_{n+m-1} - \cdots - a_1 w_{n+1} - a_0 w_n = 0.$$

Stability question: Suppose $w_0, \ldots, w_{m-1}$ are given, each bounded by ϵ. Under what conditions will the solution (with $f \equiv 0$) satisfy $|w_n| \leq C\epsilon$ for all $n \geq 0$?

This is necessary to keep w_n close to the unperturbed solution $y(t_n) = 0$ over any finite interval $0 \leq t_n \leq b$. Since h can be arbitrarily small, making large n's relevant on a finite interval, e.g. let $n \to \infty, h \to 0$ with $nh = t_n = \text{const}$. The space S of solutions of the LDE has dimension m. Stability ($|w_n| \leq C\epsilon$) is equivalent to requiring that all solutions remain bounded as $n \to \infty$.

The idea is to seek solutions of the form $w_n = \lambda^n$, where λ is a root of the polynomial

$$P(\lambda) = \lambda^m - a_{m-1}\lambda^{m-1} - \cdots - a_1\lambda - a_0.$$

Check:

$$\begin{aligned} & w_{n+m} - a_{m-1}w_{n+m-1} - \cdots - a_1 w_{n+1} - a_0 w_n \\ &= \lambda^{n+m} - a_{m-1}\lambda^{n+m-1} - \cdots - a_0\lambda^n \\ &= \lambda^n[P(\lambda)] = 0 \qquad \text{since } \lambda \text{ is a root of } P(\lambda). \end{aligned}$$

If the roots of $P(\lambda)$ are distinct, these sequences $\{\lambda_j^n\}_{n=0}^{\infty}$ are linearly independent in S for $j = 1, \ldots m$. If λ_j is a root of multiplicity μ_j, then $w_n = \lambda_j^n$ are solutions are the difference equation, and all solutions are linear combinations of these.

Example 5.11: Explicit solution of recurrence relation

Find an explicit expression for the nth term of the Fibonacci sequence:

$$w_{i+1} = w_i + w_{i-1} \qquad w_0 = 1, w_1 = 1$$

Solution: The recurrence relation is given by:

$$w_{n+2} - w_{n+1} - w_n = 0.$$

The characteristic polynomial and its roots are given by

$$P(\lambda) = \lambda^2 - \lambda - 1 \implies \lambda_1 = \frac{1+\sqrt{5}}{2} \approx 1.618, \lambda_2 = \frac{1-\sqrt{5}}{2} \approx -0.618.$$

The general solution is given by:

$$w_n = c_1\lambda_1^n + c_2\lambda_2^n.$$

Solving for the initial conditions:

$$\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ \lambda_1 & \lambda_2 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \implies \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \frac{1}{\lambda_2 - \lambda_1} \begin{pmatrix} \lambda_2 & -1 \\ -\lambda_1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{5}} \begin{pmatrix} \lambda_1 \\ -\lambda_2 \end{pmatrix}$$

$$\implies w_n = \frac{1}{\sqrt{5}}\lambda_1^{n+1} - \frac{1}{\sqrt{5}}\lambda_2^{n+1} \qquad n = 0, 1, 2, \ldots$$

which eventually grows like 1.618^{n+1}.

The root condition: all roots λ_j of $P(\lambda) = 0$ satisfy $|\lambda_j| \leq 1$. If $|\lambda_j| = 1$, then $\mu_j = 1$. In words, all roots on the unit circle have multiplicity 1.

Three notions of stability:

- A multistep method is strongly stable if $P(\lambda)$ satisfies the root condition and $\lambda = 1$ is the only root on the unit circle.
- Weakly stable: more than one distinct root on the unit circle.
- Unstable: root condition fails.

Definition 5.7 *A scheme is* ***consistent*** *if*

$$\lim_{h\to 0}\left(\max_{1\leq i\leq N}|\tau_i(h)|\right) = 0.$$

In practice, it suffices to show that $\tau_{i+1} = O(h^k)$ for some $k \geq 1$.

Definition 5.8 *A scheme is* ***convergent*** *if*

$$\lim_{h\to 0}\left(\max_{1\leq i\leq N}|w_i - y(t_i)|\right) = 0.$$

A scheme is stable if all solutions $\{w_n\}_{n=0}^{\infty}$ of the homogenous problem (with $f = 0$) remain bounded. So the point of this entire section is to show convergence of multistep methods, which is made possible by the following theorem.

Theorem 5.2 ***(Lax-Richtmeyer)*** *consistency + stability = convergence*

Example 5.12: Stability and Consistency of Multistep Methods

Consider the multistep method

$$w_{n+1} = 3w_n - 2w_{n-1} - hf(t_{n-1}, w_{n-1})$$

a) Is this method consistent? If so, what is the LTE?
b) Is this method stable?
c) Is this method convergent?

Solution: a) By the definition of LTE:

$$\begin{aligned} y_{n+1} &= 3y_n - 2_{n-1} - hf(t_{n-1}, y_{n-1}) + h\tau_{n+1} \\ \implies \tau_{n+1} &= \frac{y_{n+1} - 3y_n + 2y_{n-1}}{h} + \underbrace{f(t_{n-1}, y_{n-1})}_{y'_{n-1}}. \end{aligned}$$

Taylor expand each term around time t_n:

$$\begin{aligned} y_{n+1} &= y_n + hy'_n + \frac{h^2}{2!}y''_n + \frac{h^3}{3!}y'''_n + O(h^4) \\ y_n &= y_n \\ y_{n-1} &= y_n - hy'_n + \frac{h^2}{2!}y''_n - \frac{h^3}{3!}y'''_n + O(h^4) \end{aligned}$$

and plug them in to our expression for LTE, cancelling out like terms to get

$$\tau_{n+1} = \frac{h}{2}y''_n + O(h^2) = O(h).$$

b) Consider the characteristic polynomial:

$$\begin{aligned} w_{n+1} = 3w_n - 2w_{n-1} &\implies w_{n+1} - 3w_n + 2w_{n-1} = 0 \\ &\implies \lambda^2 - 3\lambda + 2 = 0 \\ &\implies \lambda = 1, 2. \end{aligned}$$

Since there is a root greater than 1, this method is unstable.

c) And hence not convergent.

5.5 Absolute Stability

5.5.1 Motivation

Now we turn to a a particular class of ODEs known as "stiff ODEs". This will answer the question as to why we would ever use implicit methods despite the additional cost of needing to solve nonlinear equations.

Let's consider the IVP:

$$y' = -1000y, \quad y(0) = y_0 > 0$$

The true solution for this IVP is easy to find and given by $y(t) = y_0 e^{-1000t}$, which exhibits exponential decay. Let's apply Euler's method $w_{n+1} = w_n + hf(t_n, w_n)$ to this IVP for some h:

$$w_{n+1} = w_n - 1000hw_n = (1 - 1000h)w_n.$$

So we can see that the numerical solution is obtained by simply multiplying the one at the previous time by $(1 - 1000h)$. Since the true solution exhibits exponential decay, at the very least we ask for our numerical solution to decay in absolute value (not even exponential decay like the true solution, we're not being that greedy), which gives us the restriction

$$|(1 - 1000h)| < 1 \implies -2 < -1000h < 0 \implies h < \frac{2}{1000}.$$

Typically, we like to pick our stepsize h based on accuracy concerns alone (based off LTE), but for this IVP we have an additional restriction on h here to make sure our numerical solution doesn't blow up.

Now apply implicit Euler's method $w_{n+1} = w_n - hf(t_{n+1}, w_{n+1})$ to this IVP:

$$\begin{aligned} w_{n+1} = w_n + h\lambda w_{n+1} &\implies (1 - 1000h)w_{n+1} = w_n \\ &\implies w_{n+1} = \frac{1}{1 - 1000h} w_j. \end{aligned}$$

Here $-1000h < 0$, so for any choice of of $h > 0$ the numerical solution will decay. So unlike the numerical solution from explicit Euler's method, there is no additional restriction on h. This is the payoff of the implicit method - they are practically necessary to solve what we call "stiff" equations like this one. There is no precise definition for what constitutes a **stiff equation**. Some common characterizations are:

- An equation for which certain numerical methods are numerically unstable unless the step size is taken to be extremely small.
- An equation for which it is practically necessary to use implicit methods.
- An equation which includes some terms which may lead to rapid variation in the solution (more obvious in the context of systems of ODEs).

Example 5.13: Prothero-Robinson Example

Consider the linear IVP:

$$y' = -L(y(t) - \varphi(t)) + \varphi'(t) \quad y(0) = 2$$

where $\varphi(t) = \cos(30t)$.
a) Solve the IVP exactly.
b) Write a script that will use Euler's method to solve the IVP for $0 \le t \le 1$ with $L = 10^k$ for $k = 1$ to 5. For each L use $h = 10^{-j}$ with $j = 1$ to 6. Tabulate the errors at the final time.
c) Repeat part b) with Implicit Euler. Comment on what you see.

Solution: a) Multiply by an integrating factor:

$$\begin{aligned} y' = -L(y(t) - \varphi(t)) + \varphi'(t) &\implies y' + Ly(t) = \varphi'(t) + L\varphi(t) \\ &\implies e^{Lt}y' + e^{Lt}Ly(t) = e^{Lt}\varphi' + Le^{Lt}\varphi \\ &\implies (e^{Lt}y)' = (e^{Lt}\varphi)' \\ &\implies e^{Lt}y = e^{Lt}\varphi(t) + C \\ &\implies y = \varphi(t) + Ce^{-Lt} \end{aligned}$$

Enforcing the initial condition $y(0) = 2$, we have

$$\begin{aligned} 2 &= \varphi(0) + C \\ &= 1 + C \\ \implies C &= 1 \end{aligned}$$

Therefore, the exact solution is: $\boxed{y(t) = \varphi(t) + (y_0 - 1)e^{-Lt} = \underbrace{\cos(30t)}_{\text{steady state}} + \underbrace{e^{-Lt}}_{\text{rapidly decaying}}}$

b) For Euler's method, the below shows the table of the error at $T = 1$ for various values of L and h.

L \ h	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
10^1	3.316	1.351×10^{-1}	1.272×10^{-2}	1.265×10^{-3}	1.264×10^{-4}	1.264×10^{-5}
10^2	2.598×10^{9}	2.026×10^{-3}	5.515×10^{-4}	5.833×10^{-5}	5.865×10^{-6}	5.866×10^{-7}
10^3	8.859×10^{19}	2.644×10^{95}	6.050×10^{-5}	5.647×10^{-6}	5.606×10^{-7}	5.626×10^{-8}
10^4	9.880×10^{29}	3.658×10^{199}	NaN	6.852×10^{-7}	6.806×10^{-8}	7.0429×10^{-9}
10^5	9.988×10^{39}	9.047×10^{299}	NaN	NaN	6.875×10^{-9}	9.275×10^{-10}

c) For implicit Euler, the below shows the table of the error at $T = 1$ for various values of L and h. (Results may vary depending on nonlinear solver in implicit Euler).

L \ h	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
10^1	0.879	0.119	1.256×10^{-2}	1.263×10^{-3}	1.264×10^{-4}	1.264×10^{-5}
10^2	0.212	9.054×10^{-3}	6.215×10^{-4}	5.903×10^{-5}	5.872×10^{-6}	5.871×10^{-7}
10^3	2.471×10^{-2}	1.138×10^{-4}	5.154×10^{-5}	5.557×10^{-6}	5.598×10^{-7}	5.578×10^{-8}
10^4	2.513×10^{-3}	2.329×10^{-5}	6.362×10^{-6}	6.763×10^{-7}	6.809×10^{-8}	6.572×10^{-9}
10^5	2.517×10^{-4}	2.449×10^{-6}	6.482×10^{-7}	6.883×10^{-8}	6.980×10^{-9}	4.580×10^{-10}

Some observations:

- Forward Euler requires smaller h as L gets large and the problem becomes more stiff. But backwards Euler has no such issues, demonstrating its attractive stability properties.
- For a given L, we can see that error scales with the mesh as expected, $O(h)$.
- These tables shows the point – for a given problem (a given L), you would like to just have to pick h based off the desired level of accuracy. However, for an explicit method applied to a stiff equation, you also have to make sure to pick h based on stability concerns.

Here are some accurate solutions for each L using $h = 10^{-6}$ (Figure 36). Note as L gets larger, the transient part of the solution (exponential decay) disappears faster.

5.5.2 Stability Functions

In fact, we can take any one-step method and apply it to the so called "test problem"

$$y' = \lambda y \quad \lambda < 0$$

to arrive at an expression of the form $w_{n+1} = Q(h\lambda)w_n$.

Definition 5.9 *For a given one step method, its* ***stability function*** *is known as the expression $Q(h\lambda)$ (frequently denoted by $R(z)$ where $z := h\lambda$) obtained by applying the method to the test equation $y = \lambda y$ to arrive at the expression $w_{n+1} = Q(h\lambda)w_n$*

So we have for explicit Euler's method that $Q(h\lambda) = R(z) = 1 + z$ and for implicit Euler's method that $R(z) = \frac{1}{1-z}$.

The numerical solution:

- Grows exponentially if $|Q(h\lambda)| > 1$.
- Remains bounded if $|Q(h\lambda)| = 1$.
- Converges to zero if $|Q(h\lambda)| < 1$.

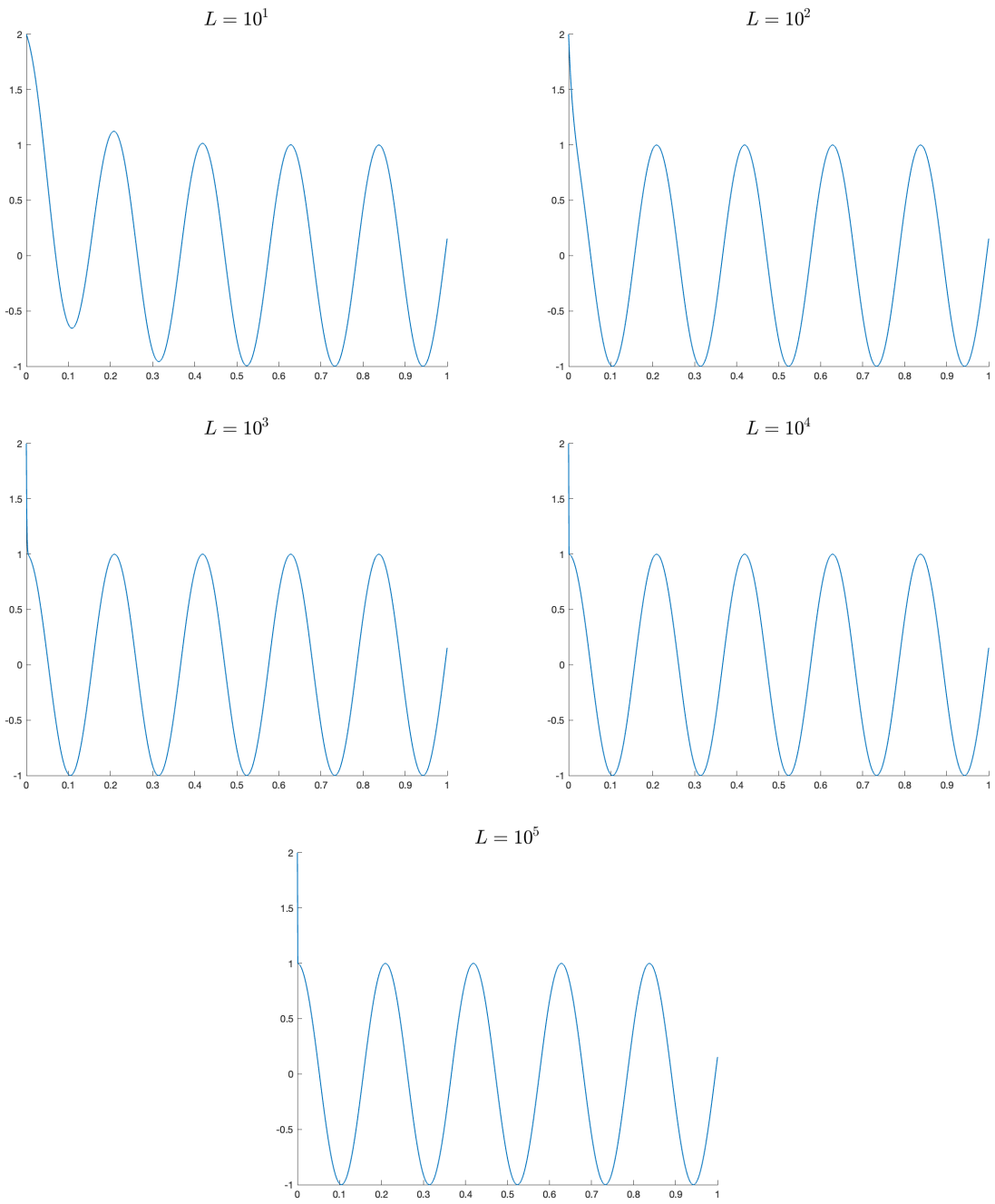


Figure 36: Numerical solution of Prothero-Robinson example for various L

Definition 5.10 *We say that the* ***region of absolute stability (RAS)*** *is:*

$$RAS = \{z \in \mathbb{C} : |Q(z)| < 1\}.$$

So the region of absolute stability for Euler's method is given by:

$$\text{RAS} = \{z \in \mathbb{C} : |1 + z| < 1\}.$$

It is given by the shaded portion in the complex plane.

This means that for a given λ in the problem for all h such that $h\lambda$ in the RAS, it will result in a numerical solution that converges to 0. Here's a concrete example: solving $y' = 100y$ requires a stepsize of $h > \frac{2}{100}$ to give a numerical solution that decays to zero.

For backward Euler:

$$\left|\frac{1}{1-z}\right| < 1 \implies |1-z| > 1 \implies \text{RAS} = \{z \in \mathbb{C} : |1-z| > 1\}.$$

Example 5.14: RAS of Implicit Trapezoid Method

Consider the implicit trapezoid method:

$$w_{n+1} = w_n + \frac{h}{2}[f(t_n, w_n) + f(t_{n+1}, w_{n+1})].$$

Find the stability function and the corresponding region of absolute stability.

Solution: Apply the method to the test equation $y' = \lambda y$:

$$\begin{aligned} w_{n+1} &= w_n + \frac{h}{2}[\lambda w_n + \lambda w_{n+1}] \\ &= w_n + \frac{h\lambda}{2} w_n + \frac{h\lambda}{2} w_{n+1} \\ \implies &\left(1 - \frac{h\lambda}{2}\right) w_{n+1} = \left(1 + \frac{h\lambda}{2}\right) w_n \\ \implies w_{n+1} &= \underbrace{\frac{1 + \frac{h\lambda}{2}}{1 - \frac{h\lambda}{2}}}_{Q(h\lambda)} w_n. \end{aligned}$$

Let's rewrite the stability function using $z = h\lambda$:

$$Q(h\lambda) = \frac{1 + \frac{h\lambda}{2}}{1 - \frac{h\lambda}{2}} = \frac{1 + \frac{z}{2}}{1 - \frac{z}{2}} = \frac{2+z}{2-z} = R(z).$$

Let's find the region of absolute stability. Rewrite $z = x + iy$:

$$|Q(z)| = \frac{|2 + x + iy|}{|2 - x - iy|} = \frac{(2+x)^2 + y^2}{(2-x)^2 + y^2} = \frac{\text{num}}{\text{denom}}.$$

So to have $|Q(z)| < 1$ we need num $<$ denom.

$$\begin{aligned} \text{num} &< \text{denom} \\ (2+x)^2 + y^2 &< (2-x)^2 + y^2 \\ (2+x)^2 &< (2-x)^2 \\ x^2 + 4x + 4 &< x^2 - 4x + 4 \\ 8x &< 0 \\ x &< 0. \end{aligned}$$

So the RAS is given by the the left half-plane.

Here is another way we could have done this without the calculation. We want to find

$$\{z \in \mathbb{C} : |2 + z| < |2 - z|\}$$

In English, this is the set of all points on the complex plane that are close to the point -2 than the point 2. But that set of points is precisely the left half-plane (not including the y-axis).

Definition 5.11 *We say a method is* ***A-stable*** *if its RAS includes the left half-plane.*

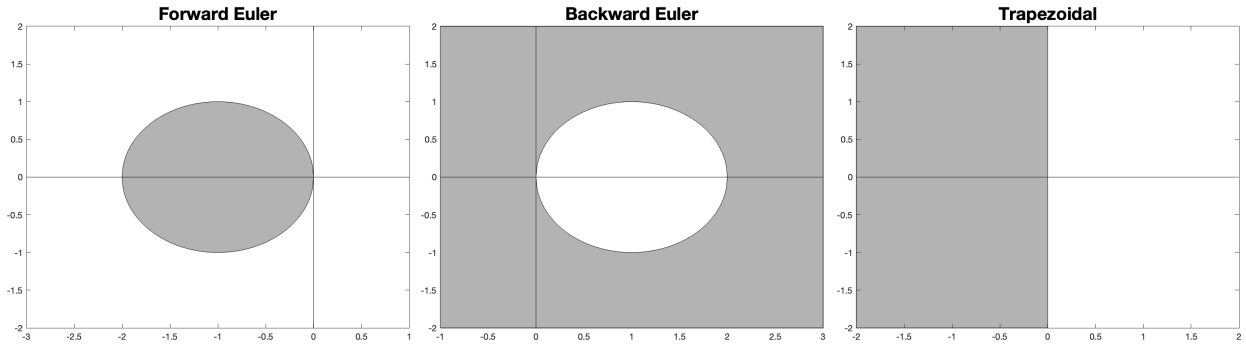


Figure 37: Regions of absolute stability (RAS)

So based off the Figure 37:

- Explicit Euler: Not A-stable.
- Implicit Euler: A-stable.
- Implicit Trapezoid Rule: A-stable (but just barely!)

The reason we like A-stable methods is that if the RAS contains the left half-plane, this means we can pick any stepsize and the resulting numerical solution of the test equation will decay to zero. With A-stable methods, the stepsize is controlled by the truncation error rather than by stability criteria. In other words, for stiff problems, implicit methods allow you to take much larger steps for a given error tolerance.

5.5.3 Examples

Example 5.15: Absolute Stability I

Compute the stability function $Q(h\lambda)$ – also known as $R(z)$ – of the following Runge-Kutta method, and determine if its A-stable.

$$w_{n+1} = w_n + \frac{h}{4} f(t_n, w_n) + \frac{3h}{4} f(t_n, w_n + \frac{2}{3} h f(t_n, w_n))$$

Solution: Apply the method to the test equation $y' = \lambda y$:

$$\begin{aligned} w_{n+1} &= w_n + \frac{h\lambda}{4} w_n + \frac{3h\lambda}{4}\left(w_n + \frac{2h\lambda}{3} w_n\right) \\ &= \left(1 + \underbrace{\frac{h\lambda}{4} + \frac{3h\lambda}{4}}_{h\lambda} + \frac{1}{2}(h\lambda)^2\right) w_n \\ &= \left(1 + z + \frac{z^2}{2}\right) w_n \end{aligned}$$

So the stability function is $R(z) = 1 + z + \frac{z^2}{2}$. This is clearly not A-stable since $R(-4) = 1 - 4 + 8 = 5$, so $|R(z)|$ is not bounded by 1 for $z \in \mathbb{C}^-$.

Example 5.16: Absolute Stability II

Consider the implicit midpoint method for solving $y' = f(t, y)$:

$$\begin{aligned} k_1 &= f\left(t_n + \frac{1}{2}h, w_n + \frac{1}{2}hk_1\right) \\ w_{n+1} &= w_n + hk_1 \end{aligned}$$

Compute the stability function $Q(z)$ for this scheme. Is the method A-stable?

Solution: Apply the method to the test equation $y' = \lambda y$:

$$k_1 = \lambda w_n + \frac{1}{2}h\lambda k_1 \implies \left(1 - \frac{1}{2}h\lambda\right) k_1 = \lambda w_n \implies k_1 = \frac{\lambda}{1 - \frac{1}{2}h\lambda} w_n$$

$$\implies w_{n+1} = \left(1 + \frac{h\lambda}{1 - \frac{1}{2}h\lambda}\right) w_n = \frac{1 + \frac{1}{2}h\lambda}{1 - \frac{1}{2}h\lambda} w_n = \frac{2 + h\lambda}{2 - h\lambda} w_n.$$

So $Q(z) = \frac{2+z}{2-z}$. This is A-stable as we showed earlier, since it's the same stability function as the Implicit Trapezoid Method.

Example 5.17: Absolute Stability III

Find the stability function of the fourth-order Runge-Kutta method. Is it A-stable?

$$\begin{aligned} k_1 &= hf(t_j, w_j) \\ k_2 &= hf(t_j + \frac{h}{2}, w_j + \frac{1}{2}k_1) \\ k_2 &= hf(t_j + \frac{h}{2}, w_j + \frac{1}{2}k_2) \\ k_2 &= hf(t_{j+1}, w_j + k_3) \\ w_{j+1} &= w_j + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned}$$

Solution: Details left to the reader. $R(z) = 1 + z + \frac{z^2}{2!} + \frac{z^3}{3!} + \frac{z^4}{4!}$ and not A-stable.

A final word on stiff equations and absolute stability: We have devoted a lot of energy to understanding how numerical methods will work on the simple problem

$$y' = \lambda y \quad y(0) = y_0 \quad \lambda < 0.$$

But why? Why did we spend so much effort and develop all these definitions based off a simple problem we can solve exactly?

The idea is that if our numerical method is to have any hope of performing well on a more complex stiff or nonlinear equation, at the very least it should perform will on this simple problem. In fact, all of these bounds on h we can get as a result of this are only a guarantee for this test problem. As such, these only give us a necessary (but far from sufficient) restriction when faced with a more general, difficult problem. For the more general class of nonlinear ODEs we want to solve numerically, we actually have no hope of developing any general theory, so this is in some sense the best we can do.

5.6 Higher-Order Equations and Systems

Most ODEs are not scalar first order equations, but all the numerical methods we have developed have only been demonstrated on such equations. For the more general cases of higher-order equations and systems, we can convert them to first-order systems and apply the methods we have developed.

Example 5.18: The Pendulum

Rewrite the equation for the pendulum as a first order system

$$\theta'' + \frac{g}{l}\sin(\theta) = 0, \quad \theta(0) = \omega_0,\ \theta'(0) = \omega_0$$

where the derivatives are w.r.t time.

Solution: In order to reduce this to a first order system, we can introduce a new variable $\omega \equiv \dot{\theta}$, and define $y(t) \equiv \begin{pmatrix} \theta(t) \\ \omega(t) \end{pmatrix}$ to rewrite the equation as

$$\frac{d}{dt}\begin{pmatrix} \theta \\ \omega \end{pmatrix} = \begin{pmatrix} \omega \\ -\frac{g}{l}\sin(\theta) \end{pmatrix}$$

or

$$\dot{y} = f(y), \quad f(y) = \begin{pmatrix} y_2 \\ -\frac{g}{l}\sin(y_1) \end{pmatrix} \quad y(0) = \begin{pmatrix} \theta_0 \\ \omega_0 \end{pmatrix}.$$

It is always possible to reduce a higher order equation to first order like this.

Now we consider the case of the nth order linear equation ODE.

$$a_n(t)u^{(n)}(t) + \cdots + a_0 u^{(0)}(t) = g(t), a_n(t) \neq 0.$$

Let $y'(t) \equiv \begin{pmatrix} u(t) & u'(t) & \cdots & u^{(n-1}(t) \end{pmatrix}^T$. Then

$$\begin{aligned} y_0'(t) &= y_1(t) \\ y_0'(t) &= y_1(t) \\ &\vdots \\ y_{n-1}'(t) &= \frac{g}{a_n} - \frac{a_0}{a_n}u^{(0)} - \cdots - \frac{a_{n-1}}{a_n}u^{(n-1)}. \end{aligned}$$

or written as an equivalent first order system, we have $y'(t) = A(t)y(t) + b(t)$ where

$$A(t) = \begin{pmatrix} 0 & 1 & & & \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ & & & 0 & 1 \\ -\frac{a_0(t)}{a_n(t)} & \cdots & \cdots & -\frac{a_{n-2}(t)}{a_n(t)} & -\frac{a_{n-1}(t)}{a_n(t)} \end{pmatrix}, \qquad b(t) = \begin{pmatrix} 0 \\ \vdots \\ \vdots \\ 0 \\ \frac{g(t)}{a_n(t)} \end{pmatrix}.$$

The general setup for first order systems is

$$y' = f(t, y), \quad f : D \to \mathbb{R}^d, \quad a \le t \le b$$

where

$$D \subset \mathbb{R} \times \mathbb{R}^d = \text{domain of } f.$$

Usually $D = [a, b] \times U$ where $U \subset \mathbb{R}^d$ is an open, convex set. The main difference is that we need to generalize the absolute value concept for scalars to vectors by the vector norm.

$$\|y\|_1 = \sum_{i=1}^{d} |y_i| \qquad \text{1-norm (Manhattan norm)}$$

$$\|y\|_2 = \sqrt{\sum_{i=1}^{d} |y_i|^2} \qquad \text{2-norm (Euclidean norm)}$$

$$\|y\|_\infty = \max_{1 \le i \le d} |y_i| \qquad \infty\text{-norm (max norm)}$$

f satisfies a Lipschitz condition in y if $\exists L$ such that

$$\|f(t, y) - f(t, z)\| \le L\|y - z\| \quad \forall (t, y), (t, z) \in D.$$

Any norm will do here. The Lipschitz condition will vary depending on the norm, but the conclusion of whether it is Lipschitz continuous or not does not by the equivalence of all norms in finite dimensions.

6 Chapter 6: Linear Algebra

We reach the final topic for this course, which is a (brief) overview of linear algebra. This is fairly unrelated to the previous few chapters, and is usually a welcome reprieve for students at the end of a long semester. The main problems of linear algebra are:

- Square systems: Given $A \in \mathbb{R}^{n \times n}, b \in \mathbb{R}^n$, solve $Ax = b$.
- Least squares problems: Given $A \in \mathbb{R}^{m \times n}$ $(m > n)$ and $b \in \mathbb{R}^m$, solve $\min_x ||Ax - b||_2$.
- Eigenvalue problems: Given $A \in \mathbb{R}^{n \times n}$, find $\lambda \in \mathbb{R}, x \in \mathbb{R}^n$ such that $Ax = \lambda x$.

These three problems were studied in Math 54, with the following algorithms developed to find their solution:

- Row reduction of the augmented matrix $[A \mid b]$.
- Forming and solving the normal equations $A^T Ax = A^T b$.
- Solving $\det(A - \lambda I) = 0$ to find the eigenvalues λ and row reduction of $[A - \lambda I \mid 0]$ to find the corresponding eigenvectors.

Just like everything else in this class, the basic premise of each topic is already familiar to you, but you expect me to tell you something new. In the previous chapter, we discussed how not all ODEs (in fact, most ODEs) cannot be solved with analytical solution techniques, so we must resort to numerical approximation. At first glance, it doesn't seem to be the case here for these linear algebra problems. For example, even if I give you a huge $100,000 \times 100,000$ linear system, while it may be tedious and time-consuming, in principle you could still imagine using the row reduction technique from Math 54 to find its solution. So it doesn't seem like we *have to* use a numerical approximation technique when it comes to the problems of linear algebra.

But the key issue with this huge $100,000 \times 100,000$ linear system is how row reduction can break down (catastrophically) in the presence of roundoff error when solved on a computer, which we demonstrate in this chapter. Another issue is how slow the "naive" approach from Math 54 can be for these problems, so we might also want develop some specialized approaches which depend on the specific structure of A, and also some numerical algorithms that can approximate the solution of $Ax = b$, analogous to fixed point iteration.

We begin with a brief review of topics which should be mostly familiar from Math 54, which leads up to the concept of pivoting to solve square systems. More advanced ideas in solving $Ax = b$ and the other two problems of linear algebra are considered in later courses such as the sequel to this course (Math 128B), and the graduate course entirely devoted to numerical linear algebra (Math 221).

6.1 Linear Algebra Review

Consider $A, B, C \in \mathbb{R}^{m\times n}, x \in \mathbb{R}^n, \lambda \in \mathbb{R}$.

The set of $m \times n$ matrices form a vector space. Namely the operations of matrix addition $(A + B)$ and multiplication by a scalar (λB) are computed entrywise.

A matrix-vector product is given by

$$Ax = \begin{pmatrix} a_{11} & a_{12} & \cdots & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} \sum_{j=1}^n a_{1j}x_j \\ \sum_{j=1}^n a_{2j}x_j \\ \vdots \\ \vdots \\ \sum_{j=1}^n a_{mj}x_j \end{pmatrix}$$

or more succinctly as

$$(Ax)_i = \sum_j a_{ij}x_j.$$

A matrix-matrix product can be expressed by

$$(AB)_{ij} = \sum_k a_{ik}b_{kj}.$$

Any linear transformation $T : \mathbb{R}^n \to \mathbb{R}^m$ can be expressed by left-multiplication of an $m \times n$ matrix. The columns of A are the images of the basis vectors e_j under T.

Any composition of linear operators can be interpreted more concretely as a matrix-matrix product.

Example 6.1: Matrix Representation of a Linear Transformation

Show the "cyclic shift" transformation $T : \mathbb{R}^n \to \mathbb{R}^n$

$$T \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} x_n \\ x_1 \\ \vdots \\ x_{n-2} \\ x_{n-1} \end{pmatrix}$$

is a linear transformation and find its matrix representation A.

Solution: Show the transformation satisfies the two properties of linearity: additivity and

homogeneity.

$$T\begin{pmatrix} x_1+y_1 \\ x_2+y_2 \\ \vdots \\ x_{n-1}+y_{n-1} \\ x_n+y_n \end{pmatrix} = \begin{pmatrix} x_n+y_n \\ x_1+y_1 \\ \vdots \\ x_{n-2}+y_{n-2} \\ x_{n-1}+y_{n-1} \end{pmatrix} = T\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} + T\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{pmatrix}$$

$$T\begin{pmatrix} cx_1 \\ cx_2 \\ \vdots \\ cx_{n-1} \\ cx_n \end{pmatrix} = \begin{pmatrix} cx_n \\ cx_1 \\ \vdots \\ cx_{n-2} \\ cx_{n-1} \end{pmatrix} = c\begin{pmatrix} x_n \\ x_1 \\ \vdots \\ x_{n-2} \\ x_{n-1} \end{pmatrix} = cT\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix}$$

Find the matrix representation by computing

$$T(e_1) = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, T(e_2) = \begin{pmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix}, \ldots, T(e_n) = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

which gives us

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

Some familiar properties of matrix multiplication that follow from the reals:

- $A(Bx) = (AB)x$
- $A(B+C) = AB + AC$
- $\lambda(AB) = (\lambda A)B = A(\lambda B)$

In the first property, the LHS is more efficient to evaluate since it requires only two matrix-vector multiplications compared to the RHS, which requires one matrix-matrix multiplication and one matrix-vector multiplication ($O(n)$ vs. $O(n^2)$).

In general, matrix multiplication is not commutative, that is

- $AB \neq BA$.

In fact, many properties from the real numbers do not hold for matrices, such as

- $AB = 0 \not\Longrightarrow A = 0$ or $B = 0$

For the case of square matrices ($m = n$), we have some important special cases.

- Identity matrix

$$I = \begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix}$$

- Diagonal matrices

$$D = \begin{pmatrix} d_{11} & & & & \\ & d_{22} & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & d_{nn} \end{pmatrix}$$

- Lower and upper triangular matrices

$$L = \begin{pmatrix} * & & & & \\ * & * & & & \\ * & * & \ddots & & \\ \vdots & \vdots & \ddots & \ddots & \\ * & * & \cdots & * & * \end{pmatrix} \qquad U = \begin{pmatrix} * & * & \cdots & * & * \\ & * & \cdots & * & * \\ & & \ddots & \ddots & * \\ & & & \ddots & \vdots \\ & & & & * \end{pmatrix}$$

6.1.1 Linear Systems of Equations

We first review the Gaussian elimination technique established in Math 54 for the solution of linear systems of the form $Ax = b$.

$$\begin{pmatrix} a_{11} & \cdots & \cdots & a_{1n} \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ a_{n1} & \cdots & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ \vdots \\ b_n \end{pmatrix} \iff \begin{aligned} a_{11}x_1 + \cdots + a_{1n}x_n &= b_1 \\ &\vdots \\ a_{n1}x_1 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

Example 6.2: Gaussian Elimination

Solve the following system by Gaussian elimination.

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

Solution: We proceed by using the first equation to cancel x_1 in the subsequent equations:

$$\begin{aligned} x_1 + x_2 &= 4 \\ x_2 + 3x_3 &= -3 \\ -x_2 - x_3 &= -4. \end{aligned}$$

Using the second equation to cancel x_2 in the subsequent equation:

$$\begin{aligned} x_1 + x_2 &= 4 \\ x_2 + 3x_3 &= -3 \\ 2x_3 &= -7. \end{aligned}$$

Solve by back substitution:

$$\begin{aligned} x_3 &= -\tfrac{7}{2} \\ x_2 &= -3 - 3x_3 = -3 + \tfrac{21}{2} = \tfrac{15}{2} \\ x_1 &= 4 - x_2 = -\tfrac{7}{2}. \end{aligned}$$

A less tedious way to represent the process is row reduction of the augmented matrix $[A \mid b]$.

$$\left(\begin{array}{ccc|c} 1 & 1 & 0 & 4 \\ 1 & 2 & 3 & 1 \\ 1 & 0 & -1 & 0 \end{array}\right) \sim \left(\begin{array}{ccc|c} 1 & 1 & 0 & 4 \\ 0 & 1 & 3 & -3 \\ 0 & -1 & -1 & -4 \end{array}\right) \sim \left(\begin{array}{ccc|c} 1 & 1 & 0 & 4 \\ 0 & 1 & 3 & -3 \\ 0 & 0 & 2 & -7 \end{array}\right) \sim \left(\begin{array}{ccc|c} 1 & 0 & 0 & -\frac{7}{2} \\ 0 & 1 & 0 & \frac{15}{2} \\ 0 & 0 & 1 & -\frac{7}{2} \end{array}\right)$$

The three elementary row operations below do not alter the solution:

- Transposing the order of two equations.
- Multiplying an equation by a nonzero constant.
- Adding a multiple of one row to another.

6.1.2 Matrix Inverses

Definition 6.1 *Consider a square matrix A. A is **invertible** if it has an inverse. That is, $\exists B$ such that $AB = BA = I$. We denote this inverse by A^{-1}. A is said to be **singular** if it is not invertible.*

In finite dimensions, the two properties we must check, that is $AA^{-1} = I, A^{-1}A = I$ are equivalent, that is, one implies the other. In other words, there is no distinction between the notion of a left inverse and a right inverse. A proof of this fact:

$$\begin{aligned} AB = I &\implies \text{columns of } B \text{ are linearly independent} \\ &\implies \text{columns of } B \text{ are a basis} \\ &\implies \exists C \text{ such that } BC = I \\ &\implies A = A(BC) = (AB)C = C \\ &\implies BA = I. \end{aligned}$$

Some basic properties of the inverse:

- A^{-1} is unique.
- A^{-1} is invertible.
- $(A^{-1})^{-1} = A$.

If A, B are invertible, then

- So is AB, which is given by $(AB)^{-1} = B^{-1}A^{-1}$.

Comment: It is computationally more efficient to solve $Ax = b$ by Gaussian elimination than by forming A^{-1} and applying it to b.

Now we discuss the procedure of computing the inverse of a matrix by hand.

Example 6.3: Matrix Inversion

Find A^{-1}:

$$A = \begin{pmatrix} 1 & 2 & -1 \\ 2 & 1 & 0 \\ -1 & 1 & 2 \end{pmatrix}$$

Solution: We seek a matrix B such that $AB = I$

$$\begin{aligned} AB &= \begin{pmatrix} 1 & 2 & -1 \\ 2 & 1 & 0 \\ -1 & 1 & 2 \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} \\ &= \begin{pmatrix} b_{11} + 2b_{21} - b_{31} & b_{12} + 2b_{22} - b_{32} & b_{13} + 2b_{23} - b_{33} \\ 2b_{11} + b_{21} & 2b_{12} + b_{22} & 2b_{13} + b_{23} \\ -b_{11} + b_{21} + 2b_{31} & -b_{12} + b_{22} + 2b_{32} & -b_{13} + b_{23} + 2b_{33} \end{pmatrix} = I \end{aligned}$$

which gives us three separate systems of three equations with three unknowns

$$\begin{array}{rrrr} b_{11} & +2b_{12} - & b_{31} & = 1 \\ 2b_{11} & + b_{21} & & = 0 \\ -b_{11} & + b_{21} & +2b_{31} & = 0 \end{array} \qquad \begin{array}{rrrr} b_{12} & +2b_{22} - & b_{32} & = 0 \\ 2b_{12} & + b_{22} & & = 1 \\ -b_{12} & + b_{22} & +2b_{32} & = 0 \end{array} \qquad \begin{array}{rrrr} b_{13} & +2b_{23} - & b_{33} & = 0 \\ 2b_{13} & + b_{23} & & = 0 \\ -b_{13} & + b_{23} & +2b_{33} & = 1. \end{array}$$

Note that the coefficients in each of the systems are the same; the only change in the systems occurs on the RHS, which are simply the standard basis vectors. So we can perform Gaussian elimination on the larger augmented matrix for all three systems simultaneously:

$$\left(\begin{array}{ccc|ccc} 1 & 2 & -1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 0 & 1 & 0 \\ -1 & 1 & 2 & 0 & 0 & 1 \end{array}\right) \sim \left(\begin{array}{ccc|ccc} 1 & 2 & -1 & 1 & 0 & 0 \\ 0 & -3 & 2 & -2 & 1 & 0 \\ 0 & 0 & 3 & -1 & 1 & 1 \end{array}\right)$$

and back-substitution is performed on each of the three right hand sides to obtain the inverse:

$$A^{-1} = \begin{pmatrix} -\frac{2}{9} & \frac{5}{9} & -\frac{1}{9} \\ \frac{4}{9} & -\frac{1}{9} & \frac{2}{9} \\ -\frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{pmatrix} = \frac{1}{9} \begin{pmatrix} -2 & 5 & -1 \\ 4 & -1 & 2 \\ -3 & 3 & 3 \end{pmatrix}.$$

So this proceeds as

$$(A \mid I) \to (U \mid B) \to (I \mid A^{-1})$$

where U is an upper triangular matrix and B is the matrix obtained by applying the same row operations to obtain U from A.

6.1.3 Matrix Transposes

Definition 6.2 *Consider an $m \times n$ matrix A. We denote its **transpose** as A^T, where the (i,j)th entry of the original matrix A becomes the (j,i)th entry of its transpose. This can be expressed by $A_{ij} = (A^T)_{ji}$.*

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & \cdots & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & \cdots & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & a_{ij} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & \cdots & \cdots & a_{mn} \end{pmatrix} \implies A^T = \begin{pmatrix} a_{11} & a_{21} & \cdots & \cdots & \cdots & \cdots & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & \cdots & \cdots & \cdots & \cdots & a_{m2} \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & a_{ij} & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & \cdots & \cdots & \cdots & \cdots & a_{mn} \end{pmatrix}$$

Definition 6.3 *A square matrix A is said to be **symmetric** if $A = A^T$.*

Some properties of transposes:

- $(A^T)^T = A$
- $(A + B)^T = A^T + B^T$
- $(A^{-1})^T = (A^T)^{-1}$
- $(AB)^T = B^T A^T$

Proving these properties requires us to consider the individual entries of the matrices. We provide a proof of the property $(AB)^T = B^T A^T$ by showing it holds for every entry.

Proof:

$$\begin{aligned} [(AB)^T]_{ij} = (AB)_{ji} &= \sum_{s=1}^{k} A_{js} B_{si} \\ &= \sum_{s=1}^{k} (A^T)_{sj} (B^T)_{is} = \sum_{s=1}^{k} (B^T)_{is} (A^T)_{sj} = (B^T A^T)_{ij} \end{aligned}$$

Matrix properties such as $A(BC) = (AB)C$ are proved similarly.

6.1.4 The Determinant

When we get to the determinant, I usually ask the class what they remember about the determinant from Math 54. The first answer is usually a hesitant, vague, and unconfident variant of the following:

- Given any square matrix, you can follow some algorithm to compute a single number associated with that matrix which is known as its determinant.

When asked what the significance of the determinant is, I usually get the answer:

- A square matrix A is invertible IFF $\det(A) \neq 0$.

And when asked for additional information about the determinant, I get a long list of properties associated with them, such as $\det(AB) = \det(A)\det(B)$ and also some "non"-properties such as $\det(A+B) = \det(A) + \det(B)$.

And of course, the calculation of determinants through cofactor expansion.

Example 6.4: Determinants by Cofactor Expansion

Find the determinant of

$$\begin{pmatrix} 1 & -4 & 1 \\ 0 & 1 & 0 \\ 1 & -1 & 2 \end{pmatrix}$$

Solution: Cofactor expansion along the first row:

$$\begin{vmatrix} 1 & -4 & 1 \\ 0 & 1 & 0 \\ 1 & -1 & 2 \end{vmatrix} = 1\begin{vmatrix} 1 & 0 \\ -1 & 2 \end{vmatrix} + 4\begin{vmatrix} 0 & 0 \\ 1 & 2 \end{vmatrix} + 1\begin{vmatrix} 0 & 1 \\ 1 & -1 \end{vmatrix} = 1(2) + 4(0) + 1(-1) = 1.$$

In this case, it would be simpler to do a cofactor expansion along the second row.

$$\begin{vmatrix} 1 & -4 & 1 \\ 0 & 1 & 0 \\ 1 & -1 & 2 \end{vmatrix} = 1\begin{vmatrix} 1 & 1 \\ 1 & 2 \end{vmatrix} = 1$$

There are many properties of determinants (most of which can be proved by cofactor expansion)

- $\det(A) = \det(A^T)$
- Switching any two rows (or columns) of the determinant changes the sign.
- If the elements of a row (or column) are multiplied by a constant, then the determinant gets multiplied by the same constant.
- Row reduction does not change the determinant.

Example 6.5: Properties of Determinants

If a 3×3 matrix A has $\det A = \frac{1}{2}$, find

(a) $\det(2A)$

(b) $\det(-A)$

(c) $\det(A^{-1})$

(d) $\det(A^T)$

(e) $\det(A^2)$

Solution:

(a) $\det(2A) = 2^3 \det(A) = 8\left(\frac{1}{2}\right) = 4$

(b) $\det(-A) = (-1)^3 \det(A) = -\det(A) = -\frac{1}{2}$

(c) $\det(A^{-1}) = \det(A)^{-1} = \left(\frac{1}{2}\right)^{-1} = 2$

(d) $\det(A^T) = \det(A)^T = \frac{1}{2}$

(e) $\det(A^2) = \det(AA) = \det(A)\det(A) = \left(\frac{1}{2}\right)^2 = \frac{1}{4}$

An important issue that remains unaddressed here is what the determinant actually *means*. As of right now it just seems like some arbitrary number assigned to a matrix that is calculated in a rather opaque way. And you would be right. I will not claim to shed any light on any of these very important issues here, except to refer you to an excellent 10 minute 3Blue1Brown video on the topic that does a very good job of doing so which motivates the determinant of a matrix A as a signed scaling factor of the linear transformation associated to the matrix A. This will also motivate the way the determinant is calculated as well as its common properties.

6.2 Pivoting

Acknowledgement: My treatment of this material is adapted from Sections 20 and 21 of Numerical Linear Algebra by Trefethen and Bau, which is easily one of the most well-written textbooks out there.

Suppose you want to solve the linear system $Ax = b$ where

$$A = \begin{pmatrix} 2 & 4 & 4 \\ -1 & 0 & 3 \\ 4 & 2 & -4 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}.$$

You would do this via Gaussian elimination by replacing rows of the augmented matrix $(A \mid b)$ with linear combinations of other rows. Your first instinct is probably to use $A_{21} = -1$ as the entry to "kill" all the other entries in the first column because it is easy to multiply and would allow you to avoid fractions. In that case, we would say that $A_{21} = -1$ is the **pivot** we chose for the first column.

But let's be systematic about it – think about how you would implement this on a computer (in exact arithmetic for now). The computer doesn't care about using a nice "round" number and doesn't see an advantage to choosing $A_{21} = -1$ as a pivot. So it would just start at the first entry of the diagonal at the top left A_{11} to eliminate the other entries below it, and continue with the entries in the $(2, 2)$ and $(3, 3)$ positions (which note, are no longer the original entries A_{22} and A_{33}). The row reduction would look like this:

$$\left(\begin{array}{ccc|c} * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{array}\right) \sim \left(\begin{array}{ccc|c} * & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{array}\right) \sim \left(\begin{array}{ccc|c} * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \end{array}\right).$$

And when the matrix is upper-triangular, we can easily back substitute to solve for x. We refer to this as **Gaussian elimination (without pivoting)**.

So what is pivoting and why do it? Why not always just use Gaussian elimination without pivoting? One obvious reason is sometimes you have to.

Consider the system $Ax = b$ where

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

Gaussian elimination (without pivoting) would fail immediately because $A_{11} = 0$, and we can't eliminate A_{21} with a multiple of 0. We would need to swap rows 1 and 2 before we could implement Gaussian elimination without pivoting.

The main (and more general) reason is to minimize roundoff error. It turns out we would like to select a pivot which has as large an absolute value as possible.

Consider the system

$$A = \begin{pmatrix} 1\text{e}{-20} & 1 \\ 1 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

We want to solve this with Gaussian elimination without pivoting in floating point arithmetic. As for as the computer can tell, the solution to this system is $x = [-1, 1]^T$.

Let's solve this system without pivoting. $(-1\text{e}{+20})R_1 + R_2 \to R_2$ in exact arithmetic gives us:

$$\left(\begin{array}{cc|c} 1\text{e}{-20} & 1 & 1 \\ 1 & 1 & 0 \end{array}\right) \sim \left(\begin{array}{cc|c} 1\text{e}{-20} & 1 & 1 \\ 0 & 1 - 1\text{e}{+20} & 0 - 1\text{e}{+20} \end{array}\right).$$

$1 - 1\text{e}{+20}$ cannot be represented exactly on the computer, so it will be rounded to the nearest floating point number. For simplicity, imagine the computer stores both $1 - 1\text{e}{+20}$ and $0 - 1\text{e}{+20}$ as $-1\text{e}{-20}$. So we are left with:

$$\left(\begin{array}{cc|c} 1\text{e}{-20} & 1 & 1 \\ 0 & -1\text{e}{+20} & -1\text{e}{+20} \end{array}\right).$$

Back substitution gives us $x_1 = 0$ and $x_2 = 1$. What happened?? This is extremely far off from the true solution.

The intuition is because we took a very small number as the pivot and had to scale it up by something very large (10^{20}) due the Gaussian elimination, it completely ruined the significance of the other entries.

Now let's pivot and see what happens. We swap the first and second row of the system

$$A = \begin{pmatrix} 1 & 1 \\ 1\text{e}{-20} & 1 \end{pmatrix} \quad b = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

and proceed with Gaussian elimination in exact arithmetic

$$\left(\begin{array}{cc|c} 1 & 1 & 0 \\ 1\text{e}{-20} & 1 & 1 \end{array}\right) \sim \left(\begin{array}{cc|c} 1 & 1 & 0 \\ 0 & 1 - 1\text{e}{-20} & 1 \end{array}\right).$$

This is stored as

$$\left(\begin{array}{cc|c} 1 & 1 & 0 \\ 0 & 1 & 1 \end{array}\right)$$

and back substitution gives us the correct solution $x_1 = -1, x_2 = 1$. Here we notice that the effects of the Gaussian elimination did not wreak havoc like in the above example. It all boils down to the fact that subtracting $1\text{e}{-20}$ from 1 does not really affect its significance after rounding, whereas subtracting $1\text{e}{+20}$ from 1 will ruin the significance of the original entry.

Of course, this is an extreme example, but it should illustrate the point that when doing Gaussian elimination, we want our pivots to be as large as possible. This minimizes the roundoff that builds up in row operations.

6.2.1 Partial Pivoting

The idea behind partial pivoting is for each column k, pick the largest remaining entry $|a_{ik}|$ as the pivot. This is known as **Gaussian elimination with partial pivoting (GEPP)**.

Example 6.6: Partial Pivoting

Row reduce the following augmented matrix with partial pivoting:

$$\left(\begin{array}{ccc|c} 1 & -2 & 0 & 4 \\ 4 & 8 & 0 & 0 \\ 0 & 1 & 2 & -1 \end{array}\right)$$

Solution: The second row contains the largest pivot, so we want to swap rows 1 and 2.

$$\left(\begin{array}{ccc|c} 1 & -2 & 0 & 4 \\ 4 & 8 & 0 & 0 \\ 0 & 1 & 2 & -1 \end{array}\right) \sim \left(\begin{array}{ccc|c} 4 & 8 & 0 & 0 \\ 1 & -2 & 0 & 4 \\ 0 & 1 & 2 & -1 \end{array}\right)$$

We proceed with row reduction on the first column.

$$\left(\begin{array}{ccc|c} 4 & 8 & 0 & 0 \\ 1 & -2 & 0 & 4 \\ 0 & 1 & 2 & -1 \end{array}\right) \sim \left(\begin{array}{ccc|c} 4 & 8 & 0 & 0 \\ 0 & -4 & 0 & 4 \\ 0 & 1 & 2 & -1 \end{array}\right)$$

We already have the largest entry in the pivot position for the second column, so no need for additional row swaps

$$\left(\begin{array}{ccc|c} 4 & 8 & 0 & 0 \\ 0 & -4 & 0 & 4 \\ 0 & 1 & 2 & -1 \end{array}\right) \sim \left(\begin{array}{ccc|c} 4 & 8 & 0 & 0 \\ 0 & -4 & 0 & 4 \\ 0 & 0 & 2 & 0 \end{array}\right)$$

and back substitution gives

$$x = \begin{pmatrix} 2 \\ -1 \\ 0 \end{pmatrix}.$$

For completeness, we also mention the idea of **complete pivoting**, where we move the largest remaining entry of the matrix to the pivot position (k, k), which requires swapping both columns and rows.

We have to keep track of the column swaps to untangle the components x_i of the solution at the end, since swapping two columns of A causes the corresponding components of the solution to switch positions as well.

Example 6.7: Complete Pivoting (OPTIONAL)

Row reduce the following augmented matrix with complete pivoting:

$$\left(\begin{array}{ccc|c} 1 & -2 & 0 & 4 \\ 4 & 8 & 0 & 0 \\ 0 & 1 & 2 & -1 \end{array}\right)$$

Solution: We want to use the entry $A_{22} = 8$ as our first pivot, so we first swap rows 1 and 2

$$\left(\begin{array}{ccc|c} 1 & -2 & 0 & 4 \\ 4 & 8 & 0 & 0 \\ 0 & 1 & 2 & -1 \end{array}\right) \sim \left(\begin{array}{ccc|c} 4 & 8 & 0 & 0 \\ 1 & -2 & 0 & 4 \\ 0 & 1 & 2 & -1 \end{array}\right)$$

and then we swap columns 1 and 2

$$\left(\begin{array}{ccc|c} 4 & 8 & 0 & 0 \\ 1 & -2 & 0 & 4 \\ 0 & 1 & 2 & -1 \end{array}\right) \sim \left(\begin{array}{ccc|c} 8 & 4 & 0 & 0 \\ -2 & 1 & 0 & 4 \\ 1 & 0 & 2 & -1 \end{array}\right).$$

But by swapping the columns, the solution to this system in terms of the original solution is given by

$$y = \begin{pmatrix} x_2 \\ x_1 \\ x_3 \end{pmatrix}.$$

Then we can proceed with Gaussian elimination as usual

$$\left(\begin{array}{ccc|c} 8 & 4 & 0 & 0 \\ -2 & 1 & 0 & 4 \\ 1 & 0 & 2 & -1 \end{array}\right) \sim \left(\begin{array}{ccc|c} 8 & 4 & 0 & 0 \\ 0 & 2 & 0 & 4 \\ 0 & -\frac{1}{2} & 2 & -1 \end{array}\right) \sim \left(\begin{array}{ccc|c} 8 & 4 & 0 & 0 \\ 0 & 2 & 0 & 4 \\ 0 & 0 & 2 & 0 \end{array}\right)$$

and back substitution gives us

$$y_3 = 0, y_2 = 2, y_1 = -1 \implies \begin{array}{l} x_1 = y_2 = 2 \\ x_2 = y_1 = -1 \\ x_3 = y_3 = 0. \end{array}$$

In practice, complete pivoting requires too much data movement and not enough floating point work to run efficiently on modern computers. As a result, this is rarely done, because the improvement in numerical stability is marginal compared to the significant increase in the amount of time to select pivots compared to partial pivoting. Partial pivoting is *usually* stable enough, and is used far more often than complete pivoting.

6.2.2 LU Factorization

In row reduction, the first step of your usual process is to eliminate entries below a_{11} (ignoring pivoting for now)

$$\begin{pmatrix} a_{11} & \cdots & \cdots & a_{1n} \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ a_{n1} & \cdots & \cdots & a_{nn} \end{pmatrix} \sim \begin{pmatrix} a_{11} & \cdots & \cdots & a_{1n} \\ 0 & * & \cdots & * \\ \vdots & \vdots & \ddots & \vdots \\ 0 & * & \cdots & * \end{pmatrix}.$$

More precisely, we replace row A_i of A by $A_i - \dfrac{a_{i1}}{a_{11}} A_1$ for $i = 2, \ldots n$. This is equivalent to left multiplication by a simple matrix:

$$\begin{pmatrix} 1 & & & & \\ -m_{21} & 1 & & & \\ -m_{31} & & 1 & & \\ \vdots & & & \ddots & \\ -m_{n1} & & & & 1 \end{pmatrix} \begin{pmatrix} — & A_1 & — \\ — & A_2 & — \\ — & A_3 & — \\ & \vdots & \\ — & A_n & — \end{pmatrix} = \begin{pmatrix} A_1 \\ -m_{21}A_1 + A_2 \\ -m_{31}A_1 + A_3 \\ \vdots \\ -m_{n1}A_1 + A_n \end{pmatrix}$$

where $m_{i1} = \dfrac{a_{i1}}{a_{11}}$.

Recall that if $C = AB$, then $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$ and

- ith row of C = linear comb. of the rows of B with coefficients from the ith row of A

$$\underbrace{\begin{pmatrix} — & — & a_1 & — & — \\ & & \vdots & & \\ a_{i1} & a_{i2} & \cdots & \cdots & a_{in} \\ & & \vdots & & \\ — & — & a_m & — & — \end{pmatrix}}_{A\ (m\times n)} \underbrace{\begin{pmatrix} — & b_1 & — \\ — & b_2 & — \\ & \vdots & \\ — & b_n & — \end{pmatrix}}_{B\ (n\times p)} = \underbrace{\begin{pmatrix} — & — & \times & — & — \\ & & \vdots & & \\ & & \sum_{j=1}^{n} a_{ij} b_j & & \\ & & \vdots & & \\ — & — & \times & — & — \end{pmatrix}}_{C\ (m\times p)}$$

- jth col of C = linear comb. of the cols of A with coefficients from the jth col of B

$$\underbrace{\begin{pmatrix} | & | & & | \\ a_1 & a_2 & \ldots & a_n \\ | & | & & | \end{pmatrix}}_{A\ (m\times n)} \underbrace{\begin{pmatrix} | & & b_{1j} & & | \\ | & & b_{2j} & & | \\ b_1 & \ldots & \vdots & \ldots & b_p \\ | & & \vdots & & | \\ | & & b_{nj} & & | \end{pmatrix}}_{B\ (n\times p)} = \underbrace{\begin{pmatrix} | & & & & | \\ \times & \ldots & \sum_{i=1}^{n} b_{ij} a_j & \ldots & \times \\ | & & & & | \end{pmatrix}}_{C\ (m\times p)}$$

So we can express row reduction of each column as a left multiplication by a single matrix.

Example 6.8: Matrix Representation of Row Operation

Suppose we want to perform $3R_1 + R_2 \to R_2$ on a 3×3 matrix A. Find a 3×3 matrix L such that LA will perform this row operation.

Solution:

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

Let's confirm this:

$$LA = \begin{pmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} = \begin{pmatrix} a & b & c \\ 3a+d & 3b+e & 3c+f \\ g & h & i \end{pmatrix}.$$

As another example, in order to do $-2R_2 + R_3 \to R_3$:

$$LA = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{pmatrix} \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g-2d & h-2e & i-2f \end{pmatrix}.$$

The reason we refer to this matrix that performs the row operation as L is because it will always be lower triangular. More specifically, it will be the identity matrix with the addition of a single non-zero entry below the diagonal in the column corresponding to the column of A we are row reducing. The reason it will always be lower triangular is because we are doing Gaussian elimination without pivoting where we are always replacing a lower row. For example, we would never do an operation like $3R_1 + R_3 \to R_1$.

Note that the order of the matrix multiplication matters: $LA \neq AL$.

Now suppose we are doing Gaussian elimination without pivoting on a 3×3 matrix A (assuming it won't fail).

$$A = \begin{pmatrix} * & * & * \\ * & * & * \\ * & * & * \end{pmatrix}$$

So left multiplication by two lower triangular matrices L_1 and L_2 will get us

$$L_2 L_1 A = \begin{pmatrix} * & * & * \\ 0 & * & * \\ 0 & * & * \end{pmatrix}$$

and we need another lower triangular matrix L_3 to finish the job

$$L_3 L_2 L_1 A = \begin{pmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \end{pmatrix} = U$$

which results in an upper triangular matrix U. So we have:

$$\begin{aligned}(L_3L_2L_1)A = U &\implies A = (L_3L_2L_1)^{-1}U \\ &\implies A = \underbrace{L_1^{-1}L_2^{-1}L_3^{-1}}_{L}U.\end{aligned}$$

Because the product of lower triangular matrices is lower triangular, and the inverse of a lower triangular matrix is lower triangular, $L = L_1^{-1}L_2^{-1}L_3^{-1}$ is also lower triangular (confirm these facts yourself). We have written $A = LU$ where L is lower triangular and U is upper triangular. This is known as an LU **factorization** of a matrix A.

This factorization follows immediately from Gaussian elimination, except for the additional step of computing the inverses of L_i and their subsequent product. But it turns out these are trivial to compute.

Example 6.9: Solving System with LU Factorization

Consider the system:

$$A = \begin{pmatrix}1 & 1 & 1\\ 2 & 3 & 5\\ 4 & 6 & 8\end{pmatrix} \quad b = \begin{pmatrix}-1\\ 1\\ 0\end{pmatrix}$$

a) Compute an LU factorization of A.
b) Use your LU factorization to solve $Ax = b$.

Solution: We need $-2R_1 + R_2 \to R_2$ and $-4R_1 + R_3 \to R_3$. These are accomplished by the left multiplication by the matrices L_1 and L_2, respectively:

$$L_1 = \begin{pmatrix}1 & 0 & 0\\ -2 & 1 & 0\\ 0 & 0 & 1\end{pmatrix} \quad L_2 = \begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ -4 & 0 & 1\end{pmatrix}.$$

So we have

$$A = \begin{pmatrix}1 & 1 & 1\\ 2 & 3 & 5\\ 4 & 6 & 8\end{pmatrix} \implies L_1A = \begin{pmatrix}1 & 1 & 1\\ 0 & 1 & 3\\ 4 & 6 & 8\end{pmatrix} \implies L_2L_1A = \begin{pmatrix}1 & 1 & 1\\ 0 & 1 & 3\\ 0 & 2 & 4\end{pmatrix}.$$

Now we need $-2R_2 + R_3 \to R_3$, which L_3 accomplishes

$$L_3L_2L_1A = \begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & -2 & 1\end{pmatrix}\begin{pmatrix}1 & 1 & 1\\ 0 & 1 & 3\\ 0 & 2 & 4\end{pmatrix} = \begin{pmatrix}1 & 1 & 1\\ 0 & 1 & 3\\ 0 & 0 & -2\end{pmatrix} = U.$$

So now we need to compute the factor $L = L_1^{-1}L_2^{-1}L_3^{-1}$. Our first "stroke of luck" is that it turns out these matrices L_i are easy to invert. The inverse of each matrix is just itself with the below diagonal entry negated.

$$L_1^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad L_2^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 4 & 0 & 1 \end{pmatrix} \quad L_3^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 2 & 1 \end{pmatrix}$$

Our second "stroke of luck" is that these matrices are extremely easy to multiply together.

$$L_1^{-1}L_2^{-1}L_3^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 4 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 2 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & 2 & 1 \end{pmatrix} = L.$$

At this point, it is a good idea to confirm your factorization is valid. That is, the L and U you obtained are indeed lower and upper triangular matrices and that $A = LU$.

Now we use this to solve $Ax = b$:

$$Ax = b \iff LUx = b \iff \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & 2 & 1 \end{pmatrix}}_{L} \underbrace{\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 3 \\ 0 & 0 & -2 \end{pmatrix}}_{U} x = \underbrace{\begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix}}_{b}.$$

Define $y \coloneqq Ux$ and do forward substitution to solve for $Ly = b$ to get

$$y = \begin{pmatrix} -1 \\ 3 \\ -2 \end{pmatrix}.$$

Now do back substitution on $Ux = y$ to get

$$x = \begin{pmatrix} -2 \\ 0 \\ 1 \end{pmatrix}.$$

Now we need to incorporate pivoting into this procedure and find a matrix that will do the row swaps needed for partial pivoting. This is done by a **permutation matrix**, which is obtained by permuting the rows (or columns) of an identity matrix of the appropriate size.

Example 6.10: Permutation Matrices

Suppose we have a 3×3 matrix

$$A = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}.$$

Find a permutation matrix that will swap the first and the third rows.

Solution: The permutation matrix which will do that is simply the identity matrix with the first and third rows swapped

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

which we obtained by swapping the first and third rows of the identity matrix. And we can check this has the desired result

$$PA = \begin{pmatrix} g & h & i \\ d & e & f \\ a & b & c \end{pmatrix}.$$

Note that the order matters – we always multiply from the left. $PA \neq AP$. Multiplying by the right has the effect of swapping columns.

Suppose we are doing Gaussian elimination with partial pivoting on a 3×3 matrix A

$$A = \begin{pmatrix} * & * & * \\ * & * & * \\ * & * & * \end{pmatrix}.$$

We first do the partial pivoting via left multiplication by P_1 and two row operations (represented by a single lower-triangular matrix L_1 by the "strokes of luck") to get

$$L_1 P_1 A = \begin{pmatrix} * & * & * \\ 0 & * & * \\ 0 & * & * \end{pmatrix}.$$

Move onto the second column to get

$$L_2 P_2 L_1 P_1 A = \begin{pmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \end{pmatrix} = U.$$

Then we can use the fact that $L_2 P_2 L_1 P_1 = L_2' L_1' P_2 P_1$ where

$$L_2' = L_2, L_1' = P_2 L_1 P_2^{-1}.$$

Note for a permutation matrix P, $P^T = P^{-1}$. You can confirm that L_1', L_2' are also lower triangular matrices, and the product of permutation matrices is a permutation matrix so we have

$$L_2' L_1' P_2 P_1 A = U \implies \underbrace{P_2 P_1}_{P} A = \underbrace{(L_2' L_1')^{-1}}_{L} U.$$

For a general $n \times n$ matrix A, we would have

$$L_{n-1} P_{n-1} L_{n-2} P_{n-2} \dots L_2 P_2 L_1 P_1 A = U.$$

which we could rewrite as

$$L_{n-1}P_{n-1}L_{n-2}P_{n-2}\dots L_2P_2L_1P_1 = L'_{n-1}L'_{n-2}\dots L'_1P_{n-1}P_{n-2}\dots P_1$$

where $L'_k = P_{n-1}\dots P_{k+1}L_kP_{k+1}^{-1}\dots P_{n-1}^{-1}$. For more details on this see §21 of Trefethen and Bau. This is the general case, but realistically you probably don't need to worry about anything larger than $n = 3$.

Example 6.11: Partial Pivoting for LU Factorization

Use the partial pivoting algorithm to factor $A = \begin{pmatrix} 2 & 4 & 4 \\ -1 & 0 & 3 \\ 4 & 2 & -4 \end{pmatrix}$.

Solution: First we need to interchange the first and the third rows since $A_{31} = 4$ is the pivot column. This is accomplished by the permutation matrix $P_1 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$.

So we have

$$P_1A = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}\begin{pmatrix} 2 & 4 & 4 \\ -1 & 0 & 3 \\ 4 & 2 & -4 \end{pmatrix} = \begin{pmatrix} 4 & 2 & -4 \\ -1 & 0 & 3 \\ 2 & 4 & 4 \end{pmatrix}.$$

Now we can begin the row reduction on PA. We want $\frac{1}{4}R_1 + R_2 \to R_2$ and $-\frac{1}{2}R_1 + R_3 \to R_3$, which are both accomplished by left multiplication by L_1

$$L_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -\frac{1}{2} & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{4} & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{4} & 1 & 0 \\ -\frac{1}{2} & 0 & 1 \end{pmatrix}.$$

This results in

$$L_1P_1A = \begin{pmatrix} 4 & 2 & -4 \\ 0 & \frac{1}{2} & 2 \\ 0 & 3 & 6 \end{pmatrix}.$$

Now we must permute again

$$P_2L_1P_1A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}\begin{pmatrix} 4 & 2 & -4 \\ 0 & \frac{1}{2} & 2 \\ 0 & 3 & 6 \end{pmatrix} = \begin{pmatrix} 4 & 2 & -4 \\ 0 & 3 & 6 \\ 0 & \frac{1}{2} & 2 \end{pmatrix}.$$

We perform $-\frac{1}{6}R_2 + R_3 \to R_3$ by left multiplication by L_2

$$L_2P_2L_1P_1A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{1}{6} & 1 \end{pmatrix}\begin{pmatrix} 4 & 2 & -4 \\ 0 & 3 & 6 \\ 0 & \frac{1}{2} & 2 \end{pmatrix} = \begin{pmatrix} 4 & 2 & -4 \\ 0 & 3 & 6 \\ 0 & 0 & 1 \end{pmatrix} = U.$$

So now we rewrite as

$$L_2P_2L_1P_1A = L_2\underbrace{P_2L_1P_2^{-1}}_{L'_1}P_2P_1A = U.$$

And compute (note that for a permutation matrix P, $P^{-1} = P^T$)

$$L_1' = P_2 L_1 P_2^T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{4} & 1 & 0 \\ -\frac{1}{2} & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 \\ \frac{1}{4} & 0 & 1 \end{pmatrix}.$$

And by the same strokes of luck as before, we have $PA = LU$ where

$$P = P_2 P_1 = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \quad L = L_1'^{-1} L_2^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ -\frac{1}{4} & \frac{1}{6} & 1 \end{pmatrix} \quad U = \begin{pmatrix} 4 & 2 & -4 \\ 0 & 3 & 6 \\ 0 & 0 & 1 \end{pmatrix}.$$

The $PA = LU$ has a simple interpretation as:

1. Permute the rows of A according to P.
2. Apply Gaussian elimination without pivoting to PA.

But the issue with this is that you cannot do this in practice because you don't know P ahead of time.

6.3 Special Types of Matrices

We discuss two types of matrices which show up frequently in applications for which Gaussian elimination can be performed without pivoting.

6.3.1 Diagonally Dominant Matrices

Definition 6.4 *An $n \times n$ matrix A is **diagonally dominant** when*

$$|a_{ii}| \geq \sum_{j \neq i}^{n} |a_{ij}| \text{ for } i = 1, 2, \ldots n.$$

In words this means that $|a_{ii}|$ is the largest element in each row for all i. If we replace the $\geq$ by $>$ in Definition 6.4, then we say A is strictly diagonally dominant. An example of a diagonally dominant matrix

$$\begin{pmatrix} -2 & 1 & & \\ 1 & -2 & 1 & \\ & 1 & -2 & 1 \\ & & 1 & -2 \end{pmatrix}.$$

Theorem 6.1 *A strictly diagonally dominant matrix is invertible.*

Proof: We proceed by contradiction. Suppose $\exists x \neq 0$ such that $Ax = 0$. Let k be chosen so $|x_k| = \max_j |x_j|$. Then

$$0 = (Ax)_k = a_{kk}x_k + \sum_{j \neq k} a_{kj}x_j.$$

So

$$\begin{aligned} a_{kk}x_k &= -\sum_{j \neq k} a_{kj}x_j \\ |a_{kk}x_k| &\leq \sum_{j \neq k} |a_{kj}||x_j| \\ |a_{kk}| &\leq \sum_{j \neq k} |a_{kj}| \underbrace{\left|\frac{x_j}{x_k}\right|}_{\leq 1} = \sum_{j \neq k} |a_{kj}| \end{aligned}$$

which contradicts strict diagonal dominance.

Theorem 6.2 *If A is strictly diagonally dominant, Gaussian elimination can be performed without pivoting.*

Proof: Omitted, but the main idea is since the "weight" of the matrix is concentrated on the diagonal, pivoting is no longer necessary.

6.3.2 Positive Definite Matrices

Definition 6.5 *A matrix A is **positive definite** (SPD) if it is symmetric and*

$$x^T Ax > 0 \qquad \forall x \in \mathbb{R}^n \backslash \{0\}$$

If we replace the $>$ by $\geq$ in Definition 6.5, then we say A is positive semidefinite. We are not generally interested in this property unless the matrix A is also symmetric.

Example 6.12: Definition of SPD

Show the following matrix is positive definite by the definition.

$$A = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}$$

Solution: Proceeding by the definition:

$$\begin{aligned} x^T Ax &= \begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \\ &= 2x_1^2 - 2x_1x_2 + 2x_2^2 - 2x_2x_3 + 2x_3^2 \\ &= x_1^2 + (x_1 - x_2)^2 + (x_2 - x_3)^2 + x_3^2 > 0 \end{aligned}$$

unless $x_1 = x_2 = x_3 = 0$.

Example 6.13: SPDness of a Matrix

For what α is the matrix A positive definite?

$$A = \begin{pmatrix} \alpha & 1 \\ 1 & \alpha \end{pmatrix}$$

Solution:

$$\begin{aligned} x^T Ax &= \begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} \alpha x_1 + x_2 \\ x_1 + \alpha x_2 \end{pmatrix} \\ &= \alpha x_1^2 + 2x_1x_2 + \alpha x_2^2 \\ &= (x_1 + x_2)^2 + (\alpha - 1)(x_1^2 + x_2^2). \end{aligned}$$

So clearly $x^T Ax > 0$ if $\alpha > 1$. And we can see that $x^T Ax \leq 0$ for $x_1 = -x_2$ if $\alpha \leq 1$. So we have that A is SPD iff $\alpha > 1$.

Theorem 6.3 *If A is SPD, then:*

- *A is invertible.*
- *$a_{ii} > 0$.*
- *The largest entry $|a_{ij}|$ has $i = j$.*
- *$|a_{ij}|^2 < a_{ii}a_{jj}$ if $i \neq j$.*

Proof:

- For any x such that $Ax = 0$: $Ax = 0 \implies x^TAx = 0 \implies x = 0$.
- $e_i^TAe_i = a_{ii} > 0$.
- Suppose not ($i \neq j$). Then

$$(e_i \pm e_j)^TA(e_i \pm e_j) = \begin{pmatrix} \cdots & 1 & \cdots & \pm 1 & \cdots \end{pmatrix} \begin{pmatrix} & & \\ & a_{ii} \quad a_{ij} & \\ & & \\ & a_{ji} \quad a_{jj} & \\ & & \end{pmatrix} = \begin{pmatrix} \vdots \\ 1 \\ \vdots \\ \pm 1 \\ \vdots \end{pmatrix} = a_{ii} \pm 2a_{ij} + a_{jj}.$$

  Choose sign $\pm$ so $a_{ii} - 2|a_{ij}| + a_{jj} = (a_{ii} - |a_{ij}|) + (a_{jj} - |a_{ij}|) \leq 0$, which is a contradiction.

- Let $x_j = \begin{cases} \alpha & k = i \\ 1 & k = j \\ 0 & \text{otherwise} \end{cases}$

  Then $0 < x^TAx = a_{ii}\alpha^2 + 2a_{ij}\alpha + a_{jj}$. This is a quadratic equation in α with no real roots, which has discriminant $4a_{ij}^2 - 4a_{ii}a_{jj} < 0$, as claimed.

Theorem 6.4 *If A is positive definite, Gaussian elimination can be performed without pivoting.*

Proof: Omitted.

It turns out the resulting U is related to L by $U = DL^T$, where D is a diagonal matrix defined by $d_{ii} = u_{ii}$. Conversely, if $A = LDL^T$ with D diagonal and $d_{ii} > 0$, then

$$y^TAy = (L^Ty)^TD(L^Ty) > 0$$

provided $y \neq 0$. Note L is invertible since it is unit lower triangular. This gives us that A is SPD iff $A = LDL^T, d_{ii} > 0$. It is clear D has a square root, so we can absorb $\sqrt{D}$ into L, so letting $\tilde{L} \coloneqq L\sqrt{D}$ gives us:

$$\tilde{L}\tilde{L}^T = L\sqrt{D}\sqrt{D}L^T = LDL^T = A.$$

This is known as the Cholesky factorization of A.

Definition 6.6 *A matrix A has a **Cholesky factorization** if it can be written as*

$$A = LL^T$$

where L is a lower triangular matrix.

Example 6.14: Cholesky Factorization

Let

$$A = \begin{pmatrix} 4 & 0 & -6 \\ 0 & 16 & -20 \\ -6 & -20 & 77 \end{pmatrix}.$$

Show that A is symmetric positive definite and find an upper triangular matrix R with positive diagonal entries $r_{ii} > 0$ such that $A = R^T R$.

Solution: Set

$$A = R^T R = \begin{pmatrix} r_{11} & 0 & 0 \\ r_{12} & r_{22} & 0 \\ r_{13} & r_{23} & r_{33} \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ 0 & r_{22} & r_{23} \\ 0 & 0 & r_{33} \end{pmatrix}$$

and solve successively for

$$\begin{aligned} r_{11} &= \sqrt{a_{11}} = 2 \\ r_{12} &= \frac{a_{12}}{r_{11}} = 0 \\ r_{13} &= \frac{a_{13}}{r_{11}} = -4 \\ r_{22} &= \sqrt{a_{22} - r_{12}^2} = 4 \\ r_{23} &= \frac{a_{23}}{r_{22}} = -5 \end{aligned}$$

and

$$r_{33} = \sqrt{a_{33} - r_{13}^2 - r_{23}^2} = 6.$$

Thus

$$R = \begin{pmatrix} 2 & 0 & -4 \\ 0 & 4 & -5 \\ 0 & 0 & 6 \end{pmatrix}$$

and since we were able to find a Cholesky factorization, A is symmetric positive definite.

There are many alternative characterizations of SPDness besides proceeding with the definition. A symmetric matrix A is positive definite if and only if:

- It has a Cholesky factorization ($A = LL^T$ for some L).

- Each of its leading principal submatrices has a positive determinant. (A leading principal submatrix of a matrix A consists of the first k rows and columns of the matrix.)
- All of its eigenvalues are positive.

It takes a certain level of judgment to know given a matrix A, which of the many characterizations is the "easiest" to use. But typically speaking, it is best to start with the direct definition when it comes to matrices that are only written in "general" form where you don't have access to the numerical entries. To illustrate what I mean:

Example 6.15: Definition of SPD (on a matrix without numerical entries)

Consider any matrix A. Show A^TA is SPD.

Solution: For any matrix A of size $m \times n$, A^TA is of size $n \times n$, so it is a square matrix.

$(A^TA)^T = A^T(A^T)^T = A^TA$, so it is clearly symmetric.

For any $\vec{x} \neq 0$, we have that

$$\vec{x}^T A^T Ax = (A\vec{x})^T(A\vec{x}) = \|A\vec{x}\|^2 \geq 0$$

It should be pretty clear that it would be pretty futile to attempt to say, calculate the eigenvalues or compute the leading principal submatrices of a matrix for which we don't even have access to the numerical entries of.

6.4 Operation Counts

We close with a brief discussion on operation counts. In choosing an algorithm to solve a problem, one important factor to consider is the speed. The traditional way to estimate the time an algorithm takes is to count the flops ("floating point operations") that it performs.

Let $A, B = \mathbb{R}^{n\times n}, x, y \in \mathbb{R}^n$. We count the number of flops associated with some common matrix operations.

Consider the dot product

$$x \cdot y = x^T y = \begin{pmatrix} x_1 & x_2 & \dots & x_n \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ x_n \end{pmatrix} = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n.$$

In order to form this quantity, it requires n multiplications to compute the terms $x_i y_i$ and $n-1$ additions to sum these terms. That gives us a total of $2n-1$ flops. We frequently only care about the leading term which illustrates how the cost scales as a function of the size of this problem, so we denote the operation count of taking a dot product as $O(n)$.

Now consider matrix vector multiplication

$$Ax = \begin{pmatrix} a_{11} & a_{12} & \cdots & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} \sum_{j=1}^n a_{1j} x_j \\ \sum_{j=1}^n a_{2j} x_j \\ \vdots \\ \vdots \\ \sum_{j=1}^n a_{mj} x_j \end{pmatrix}.$$

This is just n dot products; one for each row of A. So the operation count is $n(2n-1) = O(n^2)$. Continuing, the cost of matrix matrix multiplication is $O(n^3)$.

TODO: Compare (AB)x and A(Bx), Count for back substitution, compare solving Ax = b by inverting A vs LU.