Math 128A Notes

Kanyes Thaker

Spring 2020

0.1 Introduction

This document is an overview of Math 128A, Numerical Analysis, at UC Berkeley. These notes are largely based off of *Numerical Analysis* by Richard L. Burden and J. Douglas Faires and lectures by Dr. Per-Olof Persson. This is an introductory class to numerical methods, and these notes assume a good understanding of calculus and differential equations. Some linear algebra and analysis experience may be helpful, but is not necessary. These are not a replacement for lectures or discussions, but should provide a good enough overview to review for exams!

Contents

1 Mathematical Preliminaries and Error Analysis

We begin with a basic review of key concepts. There is very little explanation these first few pages; they're mostly here as a basic calculus review.

Key Topics

- 1.1: Limits, Continuity, Differentiability, Generalized Rolle's Theorem, Mean Value Theorem, Extreme Value Theorem, Intermediate Value Theorem, Riemann Integral, Taylor's Theorem
- 1.2: Round-off Error, Binary Machine Numbers, IEEE FLoating Point Arithmetic Standard 754-2008, Signed Bit, Exponent, Characteristic, Mantissa, Overflow, Underflow, Decimal Machine Numbers, Chopping and Rounding, Absolute and Relative Error
- 1.3: Algorithm, Big-O, Rate of Convergence

1.1 Review of Calculus

Limits of Functions

A function f defined on a set X of real number has the **limit** L at x_0, written

$$\lim_{x \longrightarrow x_0} f(x) = L,$$

if

$$\forall \varepsilon > 0,\ \exists \delta > 0 : |x - x_0| < \delta \implies |f(x) - L| < \varepsilon.$$

Continuity

Let f be defined on X and $x_0 \in X$. Then f is **continuous** at x_0 if

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

If f is continuous $\forall x \in X$, then f is **continuous on** X. We denote the set of all functions continuous on X as $C(X)$. The set of all continuous functions on an interval $[a, b]$ is $C[a, b]$.

Limits of Sequences

Let $\{x_n\}_{n=1}^{\infty}$ be an infinite sequence in $\mathbb{R}$. The sequence has **limit** x or **converges to** x if

$$\forall \varepsilon > 0,\ \exists N(\varepsilon) \in \mathbb{Z}^+ : n > N \implies |x_n - x| < \varepsilon.$$

If f is defined on X, $x_0 \in X$, then f is continuous at x_0 iff $\lim_{n\to\infty} f(x_n) = f(x_0)$ for $\{x_n\}_{n=1}^{\infty}$ converging to x_0.

1.1.1 Differentiability

Our numerical approximation methods rely on some assumptions of our functions. One such assumption is *smoothness*, which is characterized by the derivative.

Differentiability

A function f on an open interval containing x_0 is **differentiable** at x_0 if

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

exists. $f'(x_0)$ is the **derivative** of f at x_0; a function that has a derivative at each $x \in X$ is **differentiable on** X. The set of n-times differentiable functions is $C^n(X)$.

Rolle's Theorem

Suppose $f \in C[a, b]$ and f is differentiable on (a, b). If $f(a) = f(b)$, $\exists c \in (a, b) : f'(c) = 0$.

Mean Value Theorem

If $f \in C[a, b]$ and f is differentiable on (a, b), then

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

Extreme Value Theorem

If $f \in C[a,b]$, then $\exists c_1, c_2 \in [a,b] : f(c_1) \leq f(x) \leq f(c_2) \forall x \in [a,b]$. If f is differentiable on (a,b), then c_1 and c_2 appear either at the endpoints of $[a,b]$ or where $f' = 0$.

Generalized Rolle's Theorem

Suppose $f \in C^n[a,b]$. If $f(x) = 0$ at the $n+1$ points $a \leq x_0 < x_1 < ... < x_n \leq b$, then $\exists c \in (x_0, x_n) : f^{(n)}(c) = 0$.

Intermediate Value Theorem

If $f \in C[a,b]$ and K is between $f(a)$ and $f(b)$, then $\exists c \in (a,b) : f(c) = K$.

1.1.2 Integration

The other key calculus concept of this course is the Riemann integral.

Riemann Integration The **Riemann integral** of f on $[a,b]$ is the following limit, provided it exists:

$$\int_a^b f(x)dx = \lim_{\max \Delta x_i \to 0} \sum_{i=1}^{n} f(z_i)\Delta x_i,$$

where $x_0, ..., x_n$ satisfy $a = x_0 \leq x_1 \leq ... \leq x_n = b$, $\Delta x_i = x_i - x_{i-1}$, and z_i is arbitrarily chosen in $[x_{i-1}, x_i]$.

Weighted Mean Value Theorem for Integrals

Suppose $f \in C[a,b]$, the Riemann integral of g exists on $[a,b]$, and $g(x)$ maintains sign on $[a,b]$. Then

$$\exists c \in (a,b) : \int_a^b f(x)g(x)dx = f(c)\int_a^b g(x)dx.$$

1.1.3 Taylor Polynomials and Taylor Series

Computers have a finite amount of granularity with which they can approximate functions. To this end, we use our ability to create a polynomial that can approximate whatever function we want.

Taylor's Theorem

Suppose $f \in C^n[a,b]$, that $f^{(n+1)}$ exists on $[a,b]$, and $x_0 \in [a,b]$. $\forall x \in [a,b]$, $\exists \xi(x) \in [x_0, x]$ with

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

where

$$P_n(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k, \quad R_n(x) = \frac{f^{n+1}(\xi(x))}{(n+1)!}(x - x_0)^{n+1}.$$

P_n is the n**th Taylor Polynomial** of f about x_0, and R_n is the **remainder** or **truncation error**. The **Taylor series** of f about x_0 is $P_n, n \to \infty$. If $x_0 = 0$, we call this a **Maclaurin polynomial/Maclaurin series**.

1.2 Round-off Errors and Computer Arithmetic

Computers and other machines can't possibly represent every possible number, since this would require an infinite amount of resources. Instead, we make an approximation and interpret it with respect to the error in the calculation, the **round-off error**.

1.2.1 Binary Machine Numbers

The **IEEE Floating Point Arithmetic Standard 754-2008** determines the standards for binary floating point numbers, formats for data interchange, rounding algorithms, and exception handling. A 64-bit representation consists of a single **signed bit**, an 11-bit **exponent** known as the **characteristic**, and a 52-bit binary fraction known as the **mantissa**, denoted s, c, f respectively. We use this representation to compute the floating point value as

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

Numbers with magnitude less than 2^{-1022} are set to 0 as a result of **underflow** and numbers greater than $2^{1023}(2 - 2^{-52}$ will halt computation and result in **overflow**.

1.2.2 Decimal Machine Numbers

We typically represent decimal numbers in the form

$$\pm 0.d_1 d_2 ... d_k \times 10^n, \quad 1 \leq d_1 \leq 9, \quad 0 \leq d_i \leq 9.$$

Any (positive) real number in the range of the machine can be written as

$$y = 0.d_1 ... d_k d_{k+1} ... \times 10^n$$

We can "cut" the number at k digits ($fl(y)$) through one of two methods – we **chop** (remove all extra digits) after the kth digit, or we **round** (add $5 \times 10^{n-(k+1)}$ then chop).

1.2.3 Finite-Digit Arithmetic

In computer arithmetic, we are faced with errors as a result of compounding these floating-point approximations. We use the symbols $\oplus, \ominus, \otimes, \oslash$ to represent machine operations. We must use floating point approximation at every step of the computation, i.e.

$$x \oplus y = fl(fl(x) + fl(y)) \quad x \otimes y = fl(fl(x) \times fl(y))$$

$$x \ominus y = fl(fl(x) - fl(y)) \quad x \oslash y = fl(fl(x)/fl(y))$$

Absolute and Relative Error

Let p^* approximate p. Then the **absolute error** is

$$|p - p^*|,$$

and the **relative error** is

$$\frac{|p - p^*|}{|p|} \quad p \neq 0.$$

Note that rounding at every step can quickly lead to large losses in accuracy. For this reason, it is important to reduce the number of computations as much as possible.

1.3 Algorithms and Convergence

An **algorithm** is a procedure that describes a finite sequence of steps to e performed in a specified order. If you've taken any CS classes, you're already familiar with **big-O** notation and how it's applied to algorithms.

Rates of Convergence

Suppose $\{\beta_n\}_{n=1}^{\infty}$ converges to 0, and $\{\alpha_n\}_{n=1}^{\infty}$ converges to α. If there exists a positive K such that

$$|\alpha_n - \alpha| \leq K|\beta_n|,$$

then we say that the sequence (α_n) converges to α with **rate of convergence** $O(\beta_n)$. β_n is often in the form $\frac{1}{n^p}$.

2 Solutions of Equations in One Variable

Numerical methods can often approximate single-variable equations to a high degree of accuracy.

Key Topics

- 2.1: Root-finding Problems, Zeros, Bisection Method
- 2.2: Fixed Point, Functional Iteration
- 2.3: Newton-Raphson Method, Secant Method, *Regula Falsi*
- 2.4: Order of Convergence, Linear and Quadratic Convergence, Multiplicity
- 2.5: Aitken's Δ^2 Method, Forward Difference, Steffensen's Method

2.1 The Bisection Method

One of the most basic numerical approximation problems is the **root-finding problem**, where we find values of x for which $f(x) = 0$, the **roots** or **zeros** of f.

We begin with the **bisection method**. By the Intermediate Value Theorem, we know that there must be a root in the interval $[a, b]$ if $f(a)f(b) < 0$. The algorithm itself is simple:

PROCEDURE BISECTION(a, b, tol, N):
2. ASSERT $f(a)f(b) < 0$, SET $i = 1$
3. WHILE $i \leq N$ DO 4-7
   4. $p_0 = \frac{1}{2}(a + b)$
   5. IF $f(p_0)$ <tol THEN $p = p_i$ DONE.
   6. IF sgn($f(p_0)$) =sgn($f(a)$) $a = p_0, b = b$
      IF sgn($f(p_0)$) =sgn($f(b)$) $a = a, b = p_0$
   7. SET $i = i + 1$
8. UNSUCCESSFUL IN N ITERATIONS, DONE.

Bisection Theorem

Suppose $f \in C[a, b]$, $f(a)f(b) < 0$. Then the Bisection method generates a sequence (p_n) with

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

2.2 Fixed-Point Iteration

A **fixed point** for a function g is a point p such that $g(p) = p$ (we can think of this as saying that g intersects the line $y = x$ at p). Note the similarity to the above root-finding problem; if $g(x)$ has a fixed point at p then $f(x) = x - g(x)$ must have a zero at p. Likewise, if $f(x) = 0$ at p, then $g(x) = x + f(x)$ has a fixed point at p.

If $g \in C[a, b]$ and $g(x) \in [a, b]$ for all $x \in [a, b]$ then g has at least one fixed point in $[a, b]$. Additionally, if $g'(x)$ exists on (a, b) and

$$\exists k < 1 : |g'(x)| \leq k, \quad \forall x \in (a, b),$$

then there is exactly one fixed point in $[a, b]$. These conditions are sufficient but not necessary; it is possible for a unique fixed point to exist on an interval without satisfying the above condition. Observe this with $g(x) = 3^{-x}$.

2.2.1 Fixed-Point Iteration

As with root-finding, we have an iterative approach to approximating a fixed point. We choose an initial approximation p_0 and generate a sequence $\{p_n\}_{n=1}^{\infty}$ by letting $p_{n+1} = g(p_n)$. If this sequences converges to p and g is continuous, then $p = g(p)$. This technique is known as **fixed-point** or **functional** iteration.

`PROCEDURE FIXEDPOINT`(p_0, tol, N):
1. `SET` $i = 1$
2. `WHILE` $i \leq N$ `DO` 3-6
   3. `SET` $p = g(p_0)$
   4. `IF` $|p - p_0| <$tol `THEN` $p = p_0$ `DONE`.
   5. `SET` $i = i + 1$
   6. `SET` $p_0 = p$
7. `UNSUCCESSFUL IN N ITERATIONS, DONE`.

Fixed-Point Theorem

Let $g \in C[a, b]$ with $g(x) \in [a, b] \; \forall \; x \in [a, b]$. Suppose g' is defined on (a, b) and

$$\exists k \in \mathbb{R} : |g'(x)| \leq k \quad \forall x \in (a, b).$$

Then for any point $p_0 \in [a, b]$, the sequence $p_{n+1} = g(p_n)$ converges to the fixed point $p \in [a, b]$.

Corollary: If g satisfies the above hypotheses, then the bounds for the error involved in approximating p with p_n are

$$|p_n - p| \leq k^n \max\{p_0 - a, b - p_0\},$$

$$|p_n - p| \leq \frac{k^n}{1 - k}|p_1 - p_0| \quad \forall n \geq 1.$$

2.3 The Newton-Raphson Method and Extensions

The **Newton-Raphson method** (or **Newton's method**) is one of the most well-known numerical methods for root-finding problems.

2.3.1 Newton's Method

Suppose $f \in C^2[a,b]$. Let $p_0 \in [a,b]$ be an approximation to p such that $f'(p_0) \neq 0$ and $|p - p_0|$ is small. Then the first order Taylor polynomial about p_0 at p is

$$f(p) = f(p_0) + (p - p_0)f'(p_0) + \frac{1}{2}(p - p_0)^2 f''(\xi(p)).$$

We can set $f(p) = 0$, and additionally since $|p - p_0|$ is small we can say that $(p - p_0)^2$ goes to 0. Therefore, we have our approximation as

$$0 \approx f(p_0) + (p - p_0)f'(p_0) \implies p \approx p - \frac{f(p_0)}{f'(p_0)} \equiv p_1.$$

Newton's method starts with an approximation p_0 and generates the sequence $\{p_n\}_{n=0}^{\infty}$ as

$$p_n = p_{n-1} - \frac{f(p_0)}{f'(p_0)}, \quad n \geq 1.$$

The procedure is described below:

`PROCEDURE NEWTONS`(p_0, tol, N):
1. `SET` $i = 1$
2. `WHILE` $i \leq N$ `DO` 3-6:
   3. `SET` $p = p_0 - \frac{f(p_0)}{f'(p_0)}$
   4. `IF` $|p - p_0| <$tol `THEN` $p = p_0$ `DONE.`
   5. `SET` $i = i + 1$
   6. `SET` $p_0 = p$
7. `UNSUCCESSFUL IN N ITERATIONS, DONE.`

Newton's Method Theorem

Let $f \in C^2[a,b]$. If $p \in (a,b)$ such that $f(p) = 0$ and $f'(p) \neq 0$ then $\exists\, \delta > 0$ such that Newton's method generates $\{p_n\}_{n=1}^{\infty}$ converging to p for an initial approximation $p_0 \in [p - \delta, p + \delta]$.

2.3.2 The Secant Method

One of the major flaws with Newton's method is that we need to know f' at each approximation. Oftentimes, $f'(x)$ is significantly more complex than

$f(x)$. To avoid this issue, we use the definition of derivative:

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

We approximate p_{n-2} as being close to p_{n-1}. Then

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

We use this approximation in Newton's formula, yielding the **Secant method** with sequence

$$p_n = p_{n-1} - \frac{f(p_{n-1})(p_{n-1} - p_{n-2})}{f(p_{n-1}) - f(p_{n-2})}.$$

`PROCEDURE SECANT`(p_0, p_1, tol, N):
1. `SET` $i = 2$, $q_0 = f(p_0)$, $q_1 = f(p_1)$.
2. `WHILE` $i \leq N$ `DO` 3-6.
    3. `SET` $p = p_1 - q_1(p_1 - p_0)/(q_1 - q_0)$
    4. `IF` $|p - p_1| <$tol `THEN` $p = p_1$ `DONE`.
    5. `SET` $i = i + 1$
    6. `SET` $p_0 = p_1$, $q_0 = q_1$, $p_1 = p$, $q_1 = f(p)$
7. `UNSUCCESSFUL IN N ITERATIONS, DONE.`

2.3.3 *Regula Falsi*

The bisection method provides *root bracketing*, meaning that we can definitively narrow our error bound at each iteration (as we ensure that as we alter a and b the root is guaranteed to be contained between them). Newton's method and the Secant method do not guarantee root bracketing.

Regula Falsi, the **method of False Position**, generates approximations as with the Secant method but additionally ensures the root is bracketed between iterations. We use the signs of $f(p_{i-1})$ and $f(p_{i-2})$ to determine how to correctly bracket p_i, and assign indices accordingly.

`PROCEDURE FALSEPOSITION`(p_0, p_1, tol, N):
1. `SET` $i = 2$, $q_0 = f(p_0)$, $q_1 = f(p_1)$
2. `WHILE` $i \leq N$ `DO` 3-7:
    3. `SET` $p = p_1 - q_1(p_1 - p_0)/(q_1 - q_0)$

4. IF $|p - p_1| <$tol THEN $p = p_1$ DONE.
5. SET $i = i + 1$, $q = f(p)$
6. IF $(q)(q_1) < 0$ THEN SET $p_0 = p_1$, $q_0 = q_1$
7. SET $p_1 = p$, $q_1 = q$

8. UNSUCCESSFUL IN N ITERATIONS, DONE.

2.4 Error Analysis for Iterative Methods

Order of Convergence

Suppose $\{p_n\}_{n=0}^{\infty}$ is a sequence that converges to p with $p_n \neq p$. If $\lambda,\ \alpha > 0$ 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.**

If $\alpha = 1$ then the sequence is linearly convergent. If $\alpha = 2$ then the sequence is quadratically convergent.

Series that are quadratically convergent converge significantly more quickly than linearly convergent sequences.

Furthermore, we know that if we generate the sequence arbitrarily (as with the fixed point theorem) our convergence will be at most linear. Let $g \in C[a, b]$ be such that $g(x) \in [a, b]$. Suppose g' is continuous on (a, b) and $\exists\ k > 1$ where

$$|g'(x)| \leq k.$$

If $g'(p) \neq 0$ then for any $p_0 \neq p$ in $[a, b]$ the sequence $p_n = g(p_{n-1})$ converges only linearly to the fixed point.

Instead, we have a stronger condition that will guarantee a quadratically convergent sequence. Let p be a solution of $x = g(x)$. Suppose $g'(p) = 0$ and g'' is continuous with $|g''(x)| < M$ on an open interval I containing p. Then there exists $\delta > 0$ such that for $p_0 \in [p - \delta, p + \delta]$ the sequence defined by $p_n = g(p_{n-1})$ when $n \geq 1$, converges at least quadratically to p.

Moreover, for sufficiently large values of n,

$$|p_{n+1} - p| < \frac{M}{2}|p_n - p|^2.$$

Multiple Roots

A solution p of $f(x) = 0$ is a **zero of multiplicity** m of f if for $x \neq p$ we can write $f(x) = (x-p)^m q(x)$, where $\lim_{x \to p} q(x) \neq 0$. A **simple zero** of a function is one that has multiplicity 1.

Regardless of the multiplicity of the zero of f, we can use a modified version of Newton's method that is guaranteed to be quadratically convergent. To this end, we define

$$g(x) = x - \frac{f(x)f'(x)}{[f'(x)]^2 - f(x)f''(x)}.$$

Functional iteration on g will quadratically converge; however, we get significant rounding problems due to how close the elements of the denominator are to zero.

2.5 Accelerating Convergence

2.5.1 Aitken's Δ^2 Method

In 1926, mathematician Alexander Aitken determined a technique for constructing a sequence $\{\hat{p}_n\}_{n=0}^{\infty}$ that converges more rapidly to p than the original sequence $\{p_n\}_{n=0}^{\infty}$. The sequence in question is dtermined to be

$$\hat{p}_n = p_n - \frac{(p_{n+1} - p_n)^2}{p_{n+2} - 2p_{n+1} + p_n}.$$

The Δ in this notation arises from the following:

Forward Difference

For a given sequence $\{p_n\}_{n=0}^{\infty}$, the **forward difference** Δp_n is defined by

$$\Delta p_n = p_{n+1} - p_n.$$

Powers of Δ are defined recursively by

$$\Delta^k p_n = \Delta(\Delta^{k-1} p_n),$$

implying that $\Delta^2 p_n = (p_{n+2} - p_{n+1}) - (p_{n+1} - p_n)$. Then we can write Aitken's method as

$$\hat{p}_n = p_n - \frac{(\Delta p_n)^2}{\Delta^2 p_n}.$$

2.5.2 Steffensen's Method

Given a linearly convergent sequence obtained from fixed-point iteration, we can accelerate it to quadratic convergence using a modified version of Aitken's Δ^2 method. In particular, we assume that $\hat{p}_n$ is a better approximation to p than p_2.

$$p_0^{(0)},\ p_1^{(0)} = g(p_0^{(0)}),\ p_2^{(0)} = g(p_1^{(0)}),\ p_0^{(1)} = \{\Delta^2\}(p_0^{(0)}),\ p_1^{(1)} = g(p_0^{(1)}), \ldots$$

`PROCEDURE STEFFENSENS`(p_0, tol, N):

1. `SET` $i = 1$
2. `WHILE` $i \leq N$ `DO` 3-6:
   3. `SET` $p_1 = g(p_0),\ p_2 = g(p_1),\ p = p_0 - \frac{(p_1 - p_0)^2}{p_2 - 2p_1 + p_0}$
   4. `IF` $|p - p_0| <$tol `THEN OUTPUT` p, `DONE.`
   5. `SET` $i = i + 1$
   6. `SET` $p_0 = p$
7. `UNSUCCESSFUL IN N ITERATIONS, DONE.`

If $\Delta^2 p_n = 0$, then we use $p_2^{(n-1)}$ as our best approximation and proceed.

3 Interpolation and Polynomial Approximation

Measured data only gives us a snapshot of the reality of the world. We often want to be able to *interpret* data and get more useful information from it – "what does the trend look like outside the measured timeframe?"

"What would the measurement have been if we had recorded between these two time intervals?" Here, we explore methods of of interpolation to find a well-defined function that allows to answer these questions.

Key Topics

- 3.1: Algebraic Polynomials, Stone-Weierstrass Theorem, Lagrange Interpolating Polynomials
- 3.2 Divided Differences, Newton's Forward and Backward Differences
- 3.3 Osculating Polynomials, Hermite Polynomials
- 3.4 Piecewise-polynomial Approximations, Cubic Spline Interpolant, Natural and Clamped Cubic Splines

3.1 Interpolation and the Lagrange Polynomial

One particularly important class of functions is the class of **algebraic polynomials** mapping $\mathbb{R}$ to $\mathbb{R}$. These functions are particularly useful due to the following property (from real analysis):

Stone-Weierstrass Theorem

Suppose f is well-defined and continuous on $[a, b]$. For each $\varepsilon > 0$, $\exists\, P(x)$:

$$|f(x) - P(x)| < \varepsilon.$$

Note: don't confuse this for the similarly named **Bolzano-Weierstrass Theorem**, which states that every bounded sequence has a convergent subsequence.

Essentially, this means that for any function, we can find a polynomial approximating it to as granular of a degree as we want.

Note that we've already seen approximating polynomials in the form of Taylor polynomials. However, Taylor polynomials are centered around a certain value x_0, and begin to deviate significantly as we move further and further from x_0. For this reason, we don't typically use Taylor polynomials for approximation, only for error calculation purposes.

3.1.1 Lagrange Interpolating Polynomials

One of the simplest methods of interpolation was made popular by Joseph Louis Lagrange, the **nth Lagrange interpolating polynomials**. **Interpolation** means *agreeing with*; for a set of n points, we guarantee that our polynomial *must* pass through each of the points.

Lagrange Polynomial

The Lagrange polynomial given n values $x_0, ..., x_n$ and function values $f(x_0), ..., f(x_n)$ is a polynomial $P(x)$ such that $f(x_k) = P(x_k),\ k \in [0, n]$. This polynomial is

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

As per usual, we can calculate a remainder term:

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

As usual, $\xi(x)$ is unknown and in (a, b).

3.2 Divided Differences

The nth Lagrage polynomial agreeing with f at $n + 1$ points is unique, but we can represent it in different ways. Here, we seek to write $P_n(x)$ in the form

$$P_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \ldots + a_n(x - x_0)...(x - x_{n-1}).$$

We are now tasked with finding these constants $a_0, ..., a_n$. To do this, simply note that if we set $x = x_0$ then $P_n(x_0) = f(x_0) = a_0$. Likewise, for $x = x_1$ we have $P_n(x_1) = f(x_1) = f(x_0) + a_1(x_1 - x_0)$; this means that

$$a_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}.$$

More generally, the **zeroth divided difference** $f[x_i] = f(x_i)$. The **first divided difference**

$$f[x_i, x_{i+1}] = \frac{f[x_{i+1} - x_{i+2}] - f[x_i - x_{i+1}]}{x_{i+2} - x_i}.$$

Doing this recursively gives us the k**th divided difference**:

$$f[x_i, x_{i+1}, ..., x_{i+k-1}, x_{i+k}] = \frac{f[x_{i+k}, ..., x_{i+k}] - f[x_i, ..., x_{i+k-1}]}{x_{i+k} - x_i}$$

To illustrate how this works, we can use the following **divided differences table**:

$\underline{x}$	$\underline{f(x)}$	1st divided differences	2nd divided differences
x_0	$f[x_0]$		
		$f[x_0, x_1] = \frac{f[x_1]-f[x_0]}{x_1-x_0}$	
x_1	$f[x_1]$		$f[x_0, x_1, x_2] = \frac{f[x_1,x_2]-f[x_0,x_1]}{x_2-x_0}$
		$f[x_1, x_2] = \frac{f[x_2]-f[x_1]}{x_2-x_1}$	
x_2	$f[x_2]$		

Note that we can write this in terms of our forward difference Δ:

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

The coefficients of our interpolating polynomial are along the diagonal of the table. More precisely, the coefficients are $a_0 = f[x_0]$, $a_1 = f[x_0, x_1]$, $a_2 = f[x_0, x_1, x_2], ...$ and so on. This process of determining these values is known as **Newton's divided difference formula**.
We simplify this slightly with the following theorem: suppose that $f \in C^n[a, b]$ and $x_0, ..., x_n$ are distinct in $[a, b]$. Then there exists a number ξ in (a, b) with

$$f[x_0, ..., x_n]\frac{f^{(n)}(\varepsilon)}{n!}.$$

If we arrange our x_i's consecutively with equal spacing, we can simplify our notation. We use $h = x_{i+1} - x_i$ for each i, and let $x = x_0 + sh$. Then $x - x_i = sh - ih = (s - i)h$. In particular, we can write

$$P_n(x) = P_n(x_0 + sh) + f[x + 0] + \sum_{k=1}^{n} \binom{s}{k} k!h^k f[x_0, x_1, ..., x_k].$$

Newton's method here makes use of the **forward difference** Δ that we mentioned when discussing Aitken's method.

Newton's Forward Difference Method

For a list of consecutive x_i's evenly spaced with spacing h, and with $x = x_0 + sh$, the interpolating polynomial is

$$P_n(x) = f(x_0) + \sum_{k=1}^{n} \binom{s}{k} \Delta^k f(x_0).$$

3.2.1 Backward Differences

The forward difference method is most useful when the data point x we're trying to approximate is close to the top of the table, since the close it is to x_0 the more we use x_0 to determine our polynomial. However, when x is closer to x_n, we might want to use a polynomial that is more dependent on x_n than x_0. For this, we need the following definition.

Backward Difference

The **backward difference** ∇p_n is

$$\nabla p_n = p_n - p_{n-1}.$$

Higher powers are defined recursively by $\nabla^k p_n = \nabla(\nabla^{k-1} p_n)$

Using the same logic as the forward difference formula, we can define our divided differences in terms of this backward difference.

$$f[x_n, ..., x_{n-k}] = \frac{1}{k! h^k} \nabla^k f(x_n).$$

This gives us the following result:

Newton's Backward Difference Method

For a list of consecutive decreasing x_i's evenly spaced with spacing h and with $x = x_n + sh$ (note this means $s < 0$), the interpolating polynomial is given by

$$P_n(x) = f[x_n] + \sum_{k=1}^{n} (-1)^k \binom{-s}{k} \nabla^k f(x_n).$$

The divided differences are seen here:

$\underline{x}$	$\underline{f(x)}$	1st divided differences	2nd divided differences
x_0	$\underline{f[x_0]}$		
		$\underline{f[x_0,x_1] = \frac{f[x_1]-f[x_0]}{x_1-x_0}}$	
x_1	$f[x_1]$		$\underline{\overline{f[x_0,x_1,x_2] = \frac{f[x_1,x_2]-f[x_0,x_1]}{x_2-x_0}}}$
		$\overline{f[x_1,x_2] = \frac{f[x_2]-f[x_1]}{x_2-x_1}}$	
x_2	$\overline{f[x_2]}$		

Here, the overlines $\overline{f[x]}$ are the backward differences and the underlines $\underline{f[x]}$ are the forward differences. Note the 2nd divided differences coincide for both.

3.3 Hermite Interpolation

Lagrange and Taylor polynomials can be generalized by **osculating polynomials**, which attempt to not only match values but also derivatives of the original function.

Osculating Polynomial

Let $x_0, ..., x_n$ be $n+1$ distinct numbers in $[a,b]$, and for each i let $m_i \in \mathbb{Z}^+$. Suppose $f \in C^m[a,b]$ where $m = \max_i m_i$. The **osculating polynomial** approximating f is the polynomial $P(x)$ of least degree such that

$$\frac{d^k P(x_i)}{dx^k} = \frac{d^k f(x_i)}{dx^k}, \ \forall i \in [0,n], \ k \in [0, m_i].$$

Hermite polynomials are osculating polynomials where $m_i = 1$. For a function f, these polynomials agree with f at $x_0, ..., x_n$, and their first derivatives agree with f' at $x_0, ..., x_n$.

Hermite Polynomials

If $f \in C^1[a,b]$, and $x_0, ..., x_n \in [a,b]$ are distinct, the unique polynomial of least degree agreeing with f and f' at $x_0, ..., x_n$ is the **Hermite polynomial** of degree at most $2n+1$ given by

$$H_{2n+1}(x) = \sum_{j=0}^{n} f(x_j) H_{n,j}(x) + \sum_{j=0}^{n} f'(x_j) \hat{H}_{n,j}(x),$$

where for $L_{n,j}(x)$ denoting the jth Lagrange polynomial of degree n,

$$H_{n,j}(x) = [1 - 2(x - x_j)L'_{n,j}(x_j)]L^2_{n,j}(x), \quad \hat{H}_{n,j}(x) = (x - x_j)L^2_{n,j}(x).$$

3.3.1 Hermite Polynomials from Divided Differences

The above computation is extremely complex even when n is small. From our table of divided differences, we can see that the first divided difference bears a strong resemblance to the first derivative.

For this, we construct a new divided difference table with the following condition:

$$z_{2i} = z_{2i+1} = x_i.$$

Note that now $f[z_{2i}, z_{2i+1}]$ is undefined; we use $f'(x_i)$ as a substitute here. We can then just use Newton's Forward Difference method as usual (use the top row of the table) to get

$$H_{2n+1}(x) = f[z_0] + \sum_{k=1}^{2n+1} f[z_0, ..., z_k](x - z_0)...(x - z_{k-1}).$$

3.4 Cubic Spline Interpolation

High degree polynomials have highly irregular oscillations. Instead of trying to approximate using a single polynomial, it might be more appropriate to divide the interval into subintervals and construct a different approximation function for each subinterval; this is **piecewise-polynomial approximation**. The simplest piecewise polynomial approximation – a linear approximation – simply connects $(x_i, f(x_i))$ with $(x_{i+1}, f(x_{i+1}))$.

The most common piecewise-polynomial approximation uses cubic polynomials, and is called **cubic spline interpolation**. A cubic function has enough terms to where we can ensure continuous first and second derivatives on the interval; however, we do not guarantee that the interpolant matches derivatives with the function at any point.

A **cubic spline interpolant** S for a function f on $[a, b]$ with distinct nodes $a = x_0, ..., x_n = b$ satisfies the following conditions:

1. $S(x)$ is a cubic polynomial, denoted $S_j(x)$, on the subinterval $[x_j, x_{j+1}]$
2. $S_j(x_j) = f(x_j),\ S_j(x_{j+1}) = f(x_j)$
3. $S_{j+1}(x_{j+1}) = S_j(x_{j+1})$
4. $S'_{j+1}(x_{j+1}) = S'_j(x_{j+1})$
5. $S''_{j+1}(x_{j+1}) = S''_j(x_{j+1})$
6. Either $S''(x_n) = S''(x_0) = 0$ (natural spline or free spline) or $S'(x_n) = f'(x_n), S'(x_0) = f'(x_0)$ (clamped spline)

4 Numerical Differentiation and Integration

The previous section discusses how we may approximate complicated functions with polynomials. Differentiation and integration of complicated functions can be troublesome, tedious, and computationally intensive. Derivatives and integrals of polynomials are simple – here, we use polynomial approximations of functions to facilitate approximating their derivatives and integrals.

Key Topics

- 4.1: Derivative, $(n+1)$-point Formulas, Three-Point and Five-Point Formulas
- 4.2 Richardson's Extrapolation
- 4.3 Numerical Quadrature, Trapezoidal Rule, Simpson's Rule, Precision, Closed and Open Newton-Cotes Formulas,
- 4.4 Composite Trapezoidal Rule, Composite Simpson's Rule
- 4.5 Adaptive Quadratrue, Tolerance
- 4.6 Gaussian Quadrature, Legendre Polynomials
- 4.7 Composite Trapezoidal Rule for Double Integrals, Composite Simpson's Rule for Double Integrals, Non-Rectangular Regions
- 4.8 Improper Integrals, Singularity, Left and Right Endpoint Singularity, Infinite Singularity

4.1 Numerical Differentiation

Generally, the derivative of a function is

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

Computing this limit is difficult, however, due to issues with precision. We generally estimate it as follows, with $\xi \in [x_0, x_0+h]$:

$$f'(x_0) = \frac{f(x_0+h) - f(x_0)}{h} - \frac{h}{2} f''(\xi).$$

When $h > 0$ this is the **forward-difference formula**; otherwise it is the **backward-difference formula**. The error is bounded by $\frac{M|h|}{2}$, where M bounds $f''(x)$ on $[x_0, x_0+h]$.

The $(n+1)$**-point formula** for approximating $f'(x_j)$ is

$$f'(x_j) = \sum_{k=0}^{n} f(x_k) L_k'(x_j) + \frac{f^{(n+1)}(\xi(x_j))}{(n+1)!} \prod_{k=0, k \neq j}^{n} (x_j - x_k)$$

In general, the more points we use for our evaluation the better our approximation is. However, we generally don't want to use *too* many points because of the number of computations required.

4.1.1 Three-Point Formulas

Assume we have 3 evenly spaced nodes,

$$x_0, \quad x_1 = x_0 + h, \quad x_2 = x_0 + 2h.$$

Then, we have

$$L_0'(x) = \frac{2x - x_1 - x_2}{(x_0 - x_1)(x_0 - x_2)}, \; L_1'(x) = \frac{2x - x_0 - x_2}{(x_1 - x_0)(x_1 - x_2)}, \; L_2'(x) = \frac{2x - x_0 - x_1}{(x_2 - x_0)(x_2 - x_1)}.$$

Then our $(n+1)$-point formula gives us:

$$\begin{aligned}
f'(x_0) &= \frac{1}{h}\left[-\frac{3}{2}f(x_0) + 2f(x_1) - \frac{1}{2}f(x_2)\right] + \frac{h^2}{3}f^{(3)}(\xi_0), \\
f'(x_1) &= \frac{1}{h}\left[-\frac{1}{2}f(x_0) + \frac{1}{2}f(x_2)\right] - \frac{h^2}{6}f^{(3)}(\xi_1), \\
f'(x_2) &= \frac{1}{h}\left[\frac{1}{2}f(x_0) - 2f(x_1) + \frac{3}{2}f(x_2)\right] + \frac{h^2}{3}f^{(3)}(\xi_2).
\end{aligned}$$

For notational consistency, we typically write these in terms of x_0, where we substitute $x_0, x_0 + h, x_0 + 2h$ with $x_0 - h, x_0, x_0 + h$ and $x_0 - 2h, x_0 - h, x_0$ depending on context. From here, we get the following:

Three-Point Formulas

Three-Point Endpoint Formula:

$$f'(x_0) = \frac{1}{2h}[-3f(x_0) + 4f(x_0 + h) - f(x_0 + 2h)] + \frac{h^2}{3}f^{(3)}(\xi_0)$$

Three-Point Midpoint Formula:

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

We use the same concept to define the **five-point** formulas. Note that for both of these sets of formulas, we use h for the left endpoint and $-h$ for the right endpoint.

Five-Point Formulas

Five-Point Endpoint Formula:

$$f'(x_0) = \frac{1}{12h}[-25f(x_0) + 48f(x_0+h) - 36f(x_0+2h) + 16f(x_0+3h) - 3f(x_0+4h)] + \frac{h^4}{5}f^{(5)}(\xi)$$

Five-Point Midpoint Formula:

$$f'(x_0) = \frac{1}{12h}[f(x_0-2h) - 8f(x_0-h) + 8f(x_0+h) - f(x_0+2h)] + \frac{h^4}{30}f^{(5)}(\xi)$$

4.2 Richardson's Extrapolation

Richardson's Extrapolation generates high-accuracy results using low-order functions. Extrapolation can be used when we can define the error in terms of a parameter, usually the step size h.

For an approximation $N_1(h)$ approximating M, we can write the error as $M - N_1(h) = \sum_i K_i h^i$. Typically we use the lowest order term as a good approximation, so we can roughly say that $M - N_1(h) \approx K_1 h$. We want a way to eliminate all these low-order error terms.

We do this by combining the $N_1(h)$ formulas to produce $N_2(h)$ formulas with an error on $O(h^2)$ and so on. In particular, if we look at the following:

$$M = N_1(h) + K_1 h + K_2 h^2 + K_3 h^3 + ... \quad (1)$$

$$M = N_1\left(\frac{h}{2}\right) + K_1\left(\frac{h}{2}\right) + K_2\left(\frac{h}{2}\right)^2 + K_3\left(\frac{h}{2}\right)^3 + ... \quad (2)$$

we can linearly combine $2(2) - (1)$ to get

$$M = 2N_1\left(\frac{h}{2}\right) - N_1(h) + K_2\left(\frac{h^2}{2}\right) - K_2 h^2 + ...$$

We let

$$N_2 = 2N_1\left(\frac{h}{2}\right) - N_1(h)$$

In many cases, the truncation errors appear only with respect to even powers of h, which lets us get higher order errors much more quickly. In this case,

we can form a general structure:

$$N_j(h) = N_{j-1}\left(\frac{h}{2}\right) + \frac{N_{j-1}(h/2) - N_{j-1}(h)}{4^{j-1} - 1}$$

4.3 Elements of Numerical Integration

In elementary calculus, we can integrate relatively easily by using the antiderivative. However, many times the derivative either does not exist or is not easy to obtain. The method, then, of calculating $\int_a^b f(x)dx$ is known as **numerical quadrature**, using $\sum_{i=0}^{n} a_i f(x_i)$ to approximate $\int_a^b f(x)dx$.

Here, we discuss quadrature methods based in our formulas for polynomial interpolation discussed earlier. As an example, we can integrate the Lagrange interpolating polynomial:

$$\int_a^b f(x)dx = \int_a^b \sum_{i=0}^{n} f(x)L_i(x)dx + \int_a^b \prod_{i=0}^{n}(x - x_i)\frac{f^{(n+1)}(\xi(x))}{(n+1)!}dx$$

This simply becomes

$$\int_a^b f(x)dx = \sum_{i=0}^{n} f(x_i)\int_a^b L_i(x)dx + \frac{1}{(n+1)!}\int_a^b \prod_{i=0}^{n}(x - x_i)f^{(n+1)}(\xi(x))dx.$$

4.3.1 The Trapezoidal Rule

We once again take advantaged of having evenly spaced nodes. The trapezoidal rule is common in elementary calculus classes. Here, we see the intuition behind it by integrating a degree 1 Lagrange interpolator

$$P_1(x) = \frac{(x - x_1)}{(x_1 - x_0)}f(x_0) + \frac{(x - x_0)}{(x_1 - x_0)}f(x_1).$$

Simplifying leads to the following:

Trapezoidal Rule

For $x_0 = a$, $x_1 = b$, $h = b - a$, $x_0 < x_1$, the trapezoidal rule approximates the integral as

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

The error term goes to 0 when the second derivative of $f(x)$ is 0, or when $f(x)$ has degree at most 1.

4.3.2 Simpson's Rule

When we have access to the 2nd order Lagrange interpolator, we can approximate the integral with truncation error on $O(h^5)$, which is much better than the Trapezoidal rule!

Simpson's Rule

For three equally spaced points x_0, x_1, x_2 we approximate the integral of $f(x)$ as

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

For quadrature methods, we determine error by finding the polynomial class the formula belongs in. More precisely, the **degree of accuracy** or **precision** of a quadrature formula is the largest positive integer n such that the formula is *exact* for x^k, $0 \leq k \leq n$. For example, the Trapezoidal rule is exact for polynomials up to degree 1, and Simpson's rule is exact for polynomials up to degree 3.

4.3.3 Newton-Cotes Formulas

The $(n+1)$**-point closed Newton-Cotes formula** uses $n+1$ nodes equally spaced where $x_0 = a$, $x_n = b$, and $h = (b-a)/n$. It generalizes notions we've visited already with the Trapezoidal rule and Simpson's rule.

Closed Newton-Cotes Formula

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n} a_i f(x_i),$$

$$a_i = \int_{x_0}^{x_n} L_i(x)dx = \int_{x_0}^{x_n} \prod_{j=0, j\neq i}^{n} \frac{(x - x_j)}{(x_i - x_j)} dx.$$

The Trapezoidal rule is the same as the 1-point Newton-Cote's formula, and Simpson's rule is the 2-point Newton-Cote's formula.

The **open Newton-Cotes formulas** do not include $[a, b]$ as nodes. We then let $x_0 = a + h$ and $x_n = a - h$, so our bounds of integration are x_{-1} and x_{n-1}.

Open Newton-Cotes Formula

$$\int_a^b f(x)dx = \int_{x_{-1}}^{x_{n-1}} f(x)dx \approx \sum_{i=0}^{n} a_i f(x_i),$$

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

4.4 Composite Numerical Integration

Recall the issue with Hermite polynomials from the previous section; functions that oscillate heavily make it difficult to have evenly-spaced intervals. Additionally, Newton-Cotes is ultra computationally intensive over large integration intervals. Here, we discuss composite variants of the methods from above.

Composite Trapezoidal Rule

Let $f \in C^2[a, b]$, $h = (b - a)/n$, and $x_j = a + jh$ for each $j = 0, ..., n$. There exists $\mu \in (a, b)$ for which the **Composite Trapezoidal rule** for n subintervals can be written with its error term as

$$\int_a^b f(x)dx = \frac{h}{2}\left[f(a) + 2\sum_{i=1}^{n-1} f(x_j) + f(b)\right] - \frac{b-a}{12}h^2 f''(\mu).$$

Composite Simpson's Rule

Let $f \in C^4[a,b]$, n be even, $h = (b-a)/n$, and $x_j = a + jh$, for each $j = 0, ..., n$. There exists $\mu \in (a,b)$ for which **Composite Simpson's rule** for n subintervals can be written with its error term as

$$\int_a^b f(x)dx = \frac{h}{3}\left[f(a) + 2\sum_{j=1}^{(n/2)-1} f(x_{2j}) + 4\sum_{j=1}^{n/2} f(x_{2j-1}) + f(b)\right] - \frac{b-a}{180}h^4 f^{(4)}(\mu).$$

4.5 Adaptive Quadrature Methods

Not all functions are alike. Some vary widely on certain subsets of their domain and vary minimally on other subsets. For this reason, it's not always possible to determine equally spaced nodes for evaluating these functions. We want a way to vary step size in accordance with how much the function varies. These methods are known as **adaptive quadrature**.

Suppose we want to approximate $\int_a^b f(x)dx$ to a desired tolerance $\varepsilon > 0$. Using Simpson's rule, we calculate (with $h = (b-a)/2$) $\int_a^b f(x)dx = S(a,b) + \frac{h^5}{90}f^{(4)}(\xi)$. Here,

$$S(a,b) = \frac{h}{3}[f(a) + 4f(a+h) + f(b)].$$

From here, we calculate an approximation that doesn't use $f^{(4)}(\xi)$; we use composite Simpson's with $n = 4$ and $h = (b-a)/4$ to get

$$\int_a^b f(x)dx = S\left(a, \frac{a+b}{2}\right) + S\left(\frac{a+b}{2}, b\right) - \frac{1}{16}\left(\frac{h^5}{90}\right)f^{(4)}(\tilde{\xi}).$$

We assume that $f^{(4)}(\tilde{(}\xi)) \approx f^{(4)}(\xi)$, and setting our two Simpson's estimates equal to each other, we see that

$$\frac{h^5}{90}f^{(4)}(\xi) \approx \frac{16}{15}\left[S(a,b) - S\left(a, \frac{a+b}{2}\right) - S\left(\frac{a+b}{2}\right)\right].$$

This tells us that our Composite estimate is about *15 times* better than our normal estimate, so $S\left(a, \frac{a+b}{2}\right) - S\left(\frac{a+b}{2}\right)$ is a good estimate if

$$\left|S(a,b) - S\left(a, \frac{a+b}{2}\right) - S\left(\frac{a+b}{2}\right)\right| < 15\varepsilon.$$

When the above condition isn't met, we do the following. We divide $[a, b]$ into the subintervals $[a, (a+b)/2]$ and $[(a+b)/2, b]$. We do this error estimation procedure again, this time expecting a tolerance of $\varepsilon/2$ for the integral on each subinterval. If this condition is met, we sum the two approximations.

If one of the subintervals *fails* this second time, we split it in half again, this time expecting a tolerance of $\varepsilon/4$. We continue to do this until every portion is within the required tolerance.

4.6 Gaussian Quadrature

Newton-Cotes formulas involve integrating the interpolating degree n polynomial, leaving a residual on the order of the $n+1$st derivative. Newton-Cotes formulas use equally spaced intervals, which can decrease the accuracy of the approximation. For instance, the Trapezoidal rule interpolates the endpoints of the integrating domain with a 1st degree interpolator; however, this line isn't always the best line for approximating the integral.

Gaussian quadrature chooses *optimally spaced* evaluation points. We choose nodes x_i and coefficients c_i to minimize the error of

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

For example, we can determine c_1, c_2, x_1, x_2, for a polynomial of degree 3 or less, for $\int_{-1}^{1} f(x)dx \approx c_1 f(x_1) + c_2 f(x_2)$ by solving the system of equations

$$c_1 x_1^i + c_2 x_2^i = \int_{-1}^{1} x^i dx$$

for $i = 0, 1, 2, 3$ (polynomial of degree 3) which in this case yields

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

4.6.1 Legendre Polynomials

Instead of having to solve complex systems of equations to determine these coefficients, we use a special set of polynomials.

Legendre Polynomials

The **Legendre Polynomials** are a set of polynomials $\{P_0(x), P_1(x), ..., P_n(x), ...\}$ with the following properties:

1. For each n, $P_n(x)$ is a monic n-degree polynomial
2. $\int_{-1}^{1} P(x)P_n(x)dx = 0$ whenever $P(x)$ is of degree less than n

The first few Legendre polynomials are $P_0(x) = 1$, $P_1(x) = x$, $P_2(x) = x^2 - \frac{1}{3}$, $P_3(x) = x^3 - \frac{3}{5}$, $P_4(x) = x^4 - \frac{6}{7}x^2 + \frac{3}{35}$. These polynomials have distinct roots and all lie in $[-1, 1]$. The nodes needed for our Gaussian quadrature method to give exact results for any polynomial of degree less than $2n$ are the roots of the nth degree Legendre polynomial.

Legendre Polynomials and Gaussian Quadrature

Suppose $x_i, ..., x_n$ are the roots of the nth Legendre polynomial $P_n(x)$ and for each $i = 1, ..., n$ the numbers c_i are

$$c_i = \int_{-1}^{1} \prod_{j=1, j\neq i}^{n} \frac{x - x_j}{x_i - x_j} dx.$$

If $P_n(x)$ is any polynomial of degree $< 2n$, then

$$\int_{-1}^{1} P(x)dx = \sum_{i=1}^{n} c_i P(x_i).$$

Over an arbitrary interval $[a, b]$, we can use the change of variables

$$t = \frac{2x - a - b}{b - a} \iff x = \frac{1}{2}[(b - a)t + a + b]$$

to allow for our standard quadrature method;

$$\int_a^b f(x)dx = \int_{-1}^{1} f\left(\frac{(b - a)t + (b + a)}{2}\right)\left(\frac{b - a}{2}\right) dx.$$

4.7 Quadrature for Multiple Integrals

Consider the double integral over $R = \{(x, y)|x \in [a, b], y \in [c, d]\}$

$$\iint_R f(x, y)dA.$$

For this, we iterate the integral and apply our quadrature methods as normal.

$$\iint_R f(x,y)dA = \int_a^b \left(\int_c^d f(x,y)dy \right) dx.$$

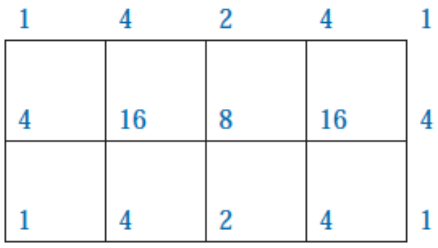
This looks complicated, but is just a byproduct of our nesting method. Here, **Composite Simpson's Rule for Double Integrals** looks like:

$$\begin{aligned}
\int_a^b \int_c^d f(x,y)dydx \approx \frac{hk}{9} \Bigg\{ & \Bigg[f(x_0,y_0) + 2 \sum_{i=1}^{(n/2)-1} f(x_{2i}y_0) \\
& + 4 \sum_{i=1}^{n/2} f(x_{2i-1},y_0) + f(x_n,y_0) \Bigg] \\
& + 2 \Bigg[\sum_{j=1}^{(m/2)-1} f(x_0,y_{2j}) + 2 \sum_{j=1}^{(m/2)-1} \sum_{i=1}^{(n/2)-1} f(x_{2i},y_{2i}) \\
& + 4 \sum_{j=1}^{(m/2)-1} \sum_{i=1}^{(n/2)} f(x_{2i-1},y_{2i}) + \sum_{j=1}^{(m/2)-1} f(x_n,y_{2j}) \Bigg] \\
& + 4 \Bigg[\sum_{j=1}^{m/2} f(x_0,y_{2j-1}) + 2 \sum_{j=1}^{m/2} \sum_{i=1}^{(n/2)-1} f(x_{2i},y_{2j-1} \\
& + 4 \sum_{j=1}^{m/2} \sum_{i=1}^{n/2} f(x_{2i-1},y_{2j-1}) + \sum_{j=1}^{m/2} f(x_n,y_{2j-1} \Bigg] \\
& + \Bigg[f(x_0,y_m) + 2 \sum_{i=1}^{(n/2)-1} f(x_{2i},y_m) + 4 \sum_{i=1}^{n/2} f(x_{2i-1},y_m) + f(x_n,y_m) \Bigg] \Bigg\} + E
\end{aligned}$$

Here, the error term E is

$$\begin{aligned}
E = & \frac{-k(b-a)h^4}{540} \Bigg[\frac{\partial^4 f}{\partial x^4}(\xi_0,y_0) + 2 \sum_{j=1}^{(m/2)-1} \frac{\partial^4 f}{\partial x^4}(\xi_{2j},y_{2j}) + 4 \sum_{j=1}^{m/2} \frac{\partial^4 f}{\partial x^4}(\xi_{2j-1},y_{2j-1}) \\
& + \frac{\partial^4}{\partial x^4}(\xi_m,y_m) \Bigg] - \frac{(d-c)k^4}{180} \int_a^b \frac{\partial^4 f}{\partial y^4}(x,\mu)dx.
\end{aligned}$$

More simply put; we use the following diagram:



where $w_{0,0}$ represents the bottom left corner. Then

$$\int_a^b \left(\int_c^d f(x,y)dy \right) dx = \frac{hk}{9} \sum_{i=0}^{n} \sum_{j=0}^{m} w_{i,j} f(x_i, y_j) + E$$

$$E = -\frac{(d-c)(b-a)}{180} \left[h^4 \frac{\partial^4 f}{\partial x^4}(\bar{\eta}, \bar{\mu}) + k^4 \frac{\partial^4 f}{\partial y^4}(\hat{\eta}, \hat{\mu}) \right]$$

4.7.1 Gaussian Quadrature for Double Integral Approximation

We apply Gaussian methods in the same iterative fashion, except now we must manipulate the bounds of both integrals rather than just one.

$$\int_a^b \int_c^d f(x,y)dydx \iff \int_{-1}^{1} \int_{-1}^{1} f\left(\frac{1}{2}[(b-a)u + a + b], \frac{1}{2}[(d-c)v + c + d] \right) dvdu$$

If the region we're integrating over is **non-rectangular**, i.e.

$$\int_a^b \int_{c(x)}^{d(x)} f(x,y)dydx,$$

we follow all the same procedures, the key difference being the step size; the step size in x is still $(b-a)/n$ but for y is $(d(x) - c(x))/m$. Any of the above methods naturally extends into more and more integrals, but becomes increasingly computationally expensive (but remember, still much less computationally expensive than trying to compute the integrals of these functions!)

4.8 Improper Integrals

Here we consider integrals where one or more integrands is unbounded.

4.8.1 Left Endpoint Singularity

If the left endpoint of an interval is unbounded, we call it a **left endpoint singularity**. We know from basic calculus that the integral with a left endpoint singularity is of the form

$$\int_a^b \frac{dx}{(x-a)^p},$$

which converges if and only if $0 < p < 1$. For this, we define

$$\int_a^b \frac{1}{(x-a)^p}dx = \lim_{M \to a^+} \frac{(x-a)^{1-p}}{1-p}\Big|_{x=M}^{x=b} = \frac{(b-a)^{1-p}}{1-p}.$$

The improper integral $\int_a^b f(x)$ exists if, for $0 < p < 1$, $\exists\, g(x)$ continuous on $[a, b]$ such that

$$f(x) = \frac{g(x)}{(x-a)^p}.$$

Using Composite Simpson's, assuming $g \in C^5[a, b]$, we construct the 4th degree Taylor polynomial of g about a, as $P_4(x) = g(a)+g'(x-a)+\frac{g''(a)}{2!}(x-a)^2 + \frac{g'''(a)}{3!}(x-a)^3 + \frac{g^(4)(a)}{4!}(x-a)^4$and rewrite our evaluation as

$$\int_a^b f(x)dx = \int_a^b \frac{P_4(x)}{(x-a)^p} + \frac{g(x)-P_4(x)}{(x-a)^p}dx$$

We can evaluate the first element of the integrand easily, since it's a polynomial. This is

$$\int_a^b \frac{P_4(x)}{(x-a)^p}dx = \sum_{k=0}^{4} \int_a^b \frac{g^{(k)}(a)}{a!}(x-a)^{k-p}dx = \sum_{k=0}^{4} \frac{g^{(k)}(a)}{k!(k+1-p)}(b-a)^{k+1-p}.$$

The second term requires us to define

$$G(x) = \begin{cases} \frac{g(x)-P_4(x)}{(x-a)^p} & a < x < b \\ 0 & x = a \end{cases}.$$

This is a continuous function on (a, b), meaning we can use composite Simpson's rule on this term, add it to our previous answer, and get an approximation accurate to within the error of Composite Simpson's.

In the case of **right endpoint singularity**, we simply turn it into a left-endpoint problem!

$$\int_a^b f(x)dx \iff \int_{-b}^{-a} f(-z)dz,$$

where $z = -x, dz = -dx$.

4.8.2 Infinite Singularity

The final case is where one of the limits of integration is infinite. This integral typically looks like

$$\int_a^\infty \frac{1}{x^p}dx.$$

For this we perform the subsitution

$$t = x^{-1}, \;\; dt = -x^{-2}dx \iff dx = -x^2dt = -t^{-2}dt.$$

Then

$$\int_a^\infty \frac{1}{x^p}dx = \int_0^{1/a} \frac{1}{t^{2-p}}dt,$$

which gives us an integral with a left endpoint singularity at 0! More generally, we turn the infinite singularity problem into a left endpoint singularity problem with

$$\int_a^\infty f(x)dx = \int_0^{1/a} t^{-2} f\left(\frac{1}{t}\right).$$

5 Initial-Value Problems for Ordinary Differential Equations

Many natural phenomena in chemistry, biology, physics, and economics can be described by equations that relate rates of change to state values. These, equations, of the form

$$a_0 \frac{d^n y(t)}{dt^n} + ... + a_{n-1}\frac{dy(t)}{dt} + a_n = g(t),$$

are known as **differential equations**.

Key Topics

- 5.1: Lipschitz Conditions, Convex Sets, Well-Posed Problems
- 5.2: Euler's Method, Mesh Points
- 5.3: Local Truncation Error, Taylor Method of Order n
- 5.4: Runge-Kutta Methods, Taylor Polynomials in Two Variables, Midpoint Method, Modified Euler Method, Higher-Order Runge-Kutta, Heun's Method
- 5.5: mth Order System of Differential Equations, Multivariate Lipschitz Condition
- 5.6 One-Step and Multistep Methods, Explicit and Implicit Methods, Adams-Bashforth Method, Adams-Moulton Method, Predictor-Corrector Methods, Adams-Bashforth-Moulton Method, Milne-Simpson Method
- 5.7 Consistency, Convergence, Stability, Root Condition, Characteristic Polynomial, Strong/Weak Stability
- Stiff Differential Equations, Transient Equations, Test Equation, Region of Stability

5.1 The Elementary Theory of Initial-Value Problems

Here, we explore methods for solving differential equations that involve a given initial condition. We typically do not solve these for actual polynomials; rather, we determine approximations of the solution at given points, and then use interpolation to determine intermediate values.

Lipschitz Conditions

A function $f(t, y)$ is said to satisfy a **Lipschitz condition** in the variable y on a set $D \subset \mathbb{R}^2$ if a constant $L > 0$ exists with

$$|f(t, y_1) - f(t, y_2)| \leq L|y_1 - y_2|,$$

whenever (t, y_1) and (t, y_2) are in D. L is then a **Lipschitz constant** for f.

Convex Set

A set $D \subset \mathbb{R}^2$ is **convex** if whenever (t_1, y_1) and (t_2, y_2) are in D, then $((1-\lambda)t_1 + \lambda t_2, (1-\lambda)y_1 + \lambda y_2)$ are in D for every $\lambda \in [0, 1]$.

In fact, if $f(t, y)$ is defined on a set D which is convex, then f satisfies a Lipschitz condition on D in the variable y with Lipschitz constant L if $\exists L > 0$:

$$\left|\frac{\partial f}{\partial y}(t, y)\right| \leq L, \quad \forall (t, y) \in D.$$

Let $D = \{(t, y) | a \leq t \leq b, y \in \mathbb{R}\}$ and let $f(t, y)$ be continuous on D. If f is Lipschitz on D in y, then

$$y'(t) = f(t, y), \quad a \leq t \leq b, \quad y(a) = \alpha$$

has a unique solution $y(t)$ for $a \leq t \leq b$.

Well-Posed Problem

The initial-value problem

$$\frac{dy}{dt} = f(t, y), \quad a \leq t \leq b, \quad y(a) = \alpha$$

is a **well-posed problem** if

1. A unique solution $y(t)$ to the problem exists, and
2. There exist constants $\varepsilon_0 > 0$ and $k > 0$ such that for any ε, with $\varepsilon_0 > \varepsilon > 0$, whenever $\delta(t)$ is continuous with $|\delta(t)| < \varepsilon$ for all $t \in [a, b]$, and when $|\delta_0| < \varepsilon$, the initial-value problem

   $$\frac{dz}{dt} = f(t, z) + \delta(t), \quad a \leq t \leq b, \quad z(a) = a + \delta_0$$

   has a unique solution $z(t)$ that satisfies

   $$|z(t) - y(t)| < k\epsilon, \quad \forall t \in [a, b].$$

The condition in (2) is known as the **perturbed problem**.

If $D = \{(t, y) | a \leq t \leq b, \quad y \in \mathbb{R}\}$ and f is continuous and is Lipschitz on D in y, then the initial value problem

$$\frac{dy}{dt} = f(t, y), \quad a \leq t \leq b, \quad y(a) = \alpha$$

is well-posed.

5.2 Euler's Method

Euler's approximation is the simplest technique for approximating initial-value problems. Euler's method approximates the well-posed initial-value problem

$$\frac{dy}{dt} = f(t, y), \quad a \leq t \leq b, \quad y(a) = \alpha$$

at a set of N discrete, equally-spaced **mesh points** in $[a, b]$, where the distance between mesh points is known as the **step size** and is denoted as h.

Euler's method constructs $w_i \approx y(t_i)$ for each $i = 1, 2, ..., N$ as follows:

$$\begin{aligned} w_0 &= \alpha, \\ w_{i+1} &= w_i + hf(t_i, w_i), \quad \forall i = 0, 1, ..., N \end{aligned}$$

Euler's method is typically not accurate enough to use in practice. However, it is a good enough approximation to warrant analyzing its error bound.

For the error, we rely on two computational truths:

1. $\forall x \geq -1, \forall m > 0, 0 \leq (1 + x)^m \leq e^{mx}$
2. If $s, t \in \mathbb{R}^+$, and $\{a_i\}_{i=0}^k$ is a sequence satisfying $a_0 \geq -t/s$ and

$$a_{i+1} \leq (1 + s)a_i + t, \quad \forall i = 0, 1, 2, ..., k - 1,$$

then

$$a_{i+1} \leq e^{(i+1)s} \left(a_0 + \frac{t}{s} \right) - \frac{t}{s}.$$

Then if f is continuous and Lipschitz in y on

$$D = \{(t, y) | a \leq t \leq b, \quad y \in \mathbb{R}\}$$

with Lipschitz constant L, and there is a constant M such that

$$|y''(t)| \leq M \quad \forall t \in [a, b]$$

where $y(t)$ is the solution to

$$y'(t) = f(t, y), \quad a \leq t \leq b, \quad y(a) = \alpha,$$

and if $w_0, ..., w_N$ are the Euler's method approximations for a positive integer N, then for each $i = 0, 1, ..., N$

$$|y(t_i) - w_i| \leq \frac{hM}{2L}[e^{L(t_i - a)} - 1].$$

If we don't know $y(t)$, we can use the multivariate chain rule to calculate

$$y''(t) = \frac{\partial f}{\partial t}(t, y(t)) + \frac{\partial f}{\partial y}(t, y(t)) \cdot f(t, y(t)).$$

5.3 Higher-Order Taylor Methods

In order to determine the accuracy of differential approximation methods, we use the **local truncation error**. For the initial problem

$$y' = f(t, y), \quad a \leq t \leq b, \quad y(a) = \alpha,$$

the difference method

$$\begin{aligned} w_0 &= \alpha \\ w_{i+1} &= w_i + h\phi(t_i, w_i), \quad i = 0, 1, ..., N \end{aligned}$$

has local truncation error

$$\tau_{i+1}(h) = \frac{y_{i+1} - y_i}{h} - \phi(t_i, y_i).$$

Taylor Method of Order n

Taylor's method of order n is the following difference equation:

$$\begin{aligned} w_0 &= \alpha \\ w_{i+1} &= w_i + hT^{(n)}(t_i, w_i) \end{aligned}$$

where

$$T^{(n)}(t_i, w_i) = f(t_i, w_i) + \frac{h}{2}f'(t_i, w_i) + \ldots + \frac{h^{n-1}}{n!}f^{(n-1)}(t_i, w_i).$$

Euler's method is Taylor's method of order 1.

If Taylor's method of order n is used to approximate the solution to

$$y' = f(t, y), \quad a \leq t \leq b, \quad y(a) = \alpha,$$

with step size h and if $y \in C^{n+1}[a, b]$ then the local truncation error is $\mathrm{O}(h^n)$.

5.4 Runge-Kutta Methods

Taylor's methods improve on Euler's method by allowing for high-order local truncation error. However, Taylor's methods require the computation and evaluations of derivatives of f which can be unwieldy.

Runge-Kutta methods have the high-order truncation error of Taylor polynomials without needing to compute derivatives of f. Before we begin, we establish the following:

Suppose $f(t, y)$ and all its partial derivatives of order less than or equal to $n + 1$ are continuous on $D = \{(t, y)|a \leq t \leq b, c \leq y \leq d\}$ and let $(t_0, y_0) \in D$. For every $(t, y) \in D$, there is ξ between t and t_0 and μ between y and y_0 with

$$f(t, y) = P_n(t, y) + R_n(t, y),$$

where:

$$P_n(t, y) = \sum_{i=0}^{n} \frac{1}{i!} \sum_{j=0}^{i} \binom{i}{j} (t - t_0)^{i-j} (y - y_0)^j \frac{\partial^i f}{\partial t^{i-j} \partial y^j}(t_0, y_0),$$

$$R_n(t, y) = \frac{1}{(n+1)!} \sum_{j=0}^{n+1} \binom{n+1}{j} (t - t_0)^{n+1-j} (y - y_0)^j \frac{\partial^{n+1} f}{\partial t^{n+1-j} \partial y^j}(\xi, \mu).$$

$P_n(t, y)$ is the n**th order Taylor polynomial in two variables** for f about (t_0, y_0) and $R_n(t, y)$ is the remainder term associated with $P_n(t, y)$.

5.4.1 Runge-Kutta Methods of Order Two

We first determine values for a_1, α_1, and β_1 with the property that $a_1 f(t + \alpha_1, y + \beta_1)$ approximates

$$T^{(2)}(t, y) = f(t, y) + \frac{h}{2} f'(t, y)$$

with error at most $O(h^2)$. Expanding $a_1 f(t + \alpha_1, y + \beta_1)$ using its Taylor polynomial

$$T^{(2)}(t, y) = f(t, y) + \frac{h}{2} \frac{\partial f}{\partial t}(t, y) + \frac{h}{2} \frac{\partial f}{\partial y}(t, y) \cdot f(t, y).$$

We determine through computation (not shown here) is $a_1 = 1, \alpha_1 = \frac{h}{2}, \beta_1 = \frac{h}{2} f(t, y)$.

Midpoint Method

The specific Runge-Kutta method of order 2 derived from the parameters above is known as the **midpoint method**.

$$w_0 = \alpha,$$
$$w_{i+1} = w_i + hf\left(t_i + \frac{h}{2}, w_i + \frac{h}{2}f(t_i, w_i)\right)$$

These 3 parameters are enough to match $T^{(2)}$. The most appropriate form for approximating

$$T^{(3)}(t,y) = f(t,y) + \frac{h}{2}f'(t,y) + \frac{h^2}{6}f''(t,y)$$

is

$$a_1 f(t,y) + a_2 f(t+\alpha_2, y+\delta_2 f(t,y)).$$

There is not enough flexibility here to match the term $\frac{h^2}{6}f''(t,y)$; therefore the best error we can get is $\mathrm{O}(h^2)$. However, with 4 parameters we have more freedom in our methods.

Modified Euler Method

The **modified Euler method** is defined as the following difference equation:

$$w_0 = \alpha,$$
$$w_{i+1} = w_i + \frac{h}{2}[f(t_i, w_i) + f(t_{i+1}, w_i + hf(t_i, w_i))]$$

5.4.2 Higher-Order Runge-Kutta Methods

We can approximate $T^{(3)}(t,y)$ can be approximated to an order of $\mathrm{O}(h^3)$ by an expression of the form

$$f(t+\alpha_1, y+\delta_1 f(t+\alpha_2, y+\delta_2 f(y,t))).$$

We disregard the calculations here (they get quite complicated) but we *can* find a difference equation in 4 parameters.

Heun's Method

The most common $O(h^3)$ approximation for $T^{(3)}(t, y)$ is **Heun's method**, given by the difference equation

$$w_0 = \alpha$$
$$w_{i+1} = w_i + \frac{h}{4}(f(t_i, w_i) + 3f(t_i + \frac{2h}{3}f(t_i + \frac{h}{3}, w_i + \frac{h}{3}f(t_i, w_i)))$$

it should be noted that 3rd order Runge-Kutta is not typically used.

The most common Runge-Kutta method is of order 4. We skip the derivation and focus on the difference equation:

$$\begin{aligned} w_0 &= \alpha \\ k_1 &= hf(t_i, w_i) \\ k_2 &= hf\left(t_i + \frac{h}{2}, w_i + \frac{1}{2}k_1\right) \\ k_3 &= hf\left(t_i + \frac{h}{2}, w_i + \frac{1}{2}k_2\right) \\ k_4 &= hf(t_{i+1}, w_i + k_3) \\ w_{i+1} &= w_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned}$$

Provided that $y(t)$ has five continuous derivatives, this method has local truncation error $O(h^4)$.

5.5 Higher-Order Equations and Systems of Differential Equations

Here, we we discuss techniques that solve systems of first-order differential equations that are a transformation of a higher-order differential equation.

An m**th order system** of first-order initial-value problems has the form

$$\begin{aligned}
\frac{du_1}{dt} &= f_1(t, u_1, u_2, ..., u_m), \\
\frac{du_2}{dt} &= f_2(t, u_1, u_2, ..., u_m), \\
&\vdots \\
\frac{du_m}{dt} &= f_m(t, u_1, u_2, ..., u_m),
\end{aligned}$$

with the initial conditions

$$u_1(a) = \alpha_1,\ u_2(a) = \alpha_2, ..., u_m(a) = \alpha_m.$$

Multivariate Lipschitz Condition

The function $f(t, y_1, ..., y_m)$ defined on the set

$$D = \{(t, u_1, ..., u_m) | a \leq t \leq b,\ u_i \in \mathbb{R} \forall i \in [1, m]\}$$

is said to satisfy a **Lipschitz condition** on D in $u_1, ..., u_m$ if $L > 0$ exists with

$$|f(t, u_1, ..., u_m) - f(t, z_1, ..., z_m)| \leq L \sum_{j=1}^{m} |u_j - z_j|,$$

for all $(t, u_1, ..., u_m)$ and $(t, z_1, ..., z_m)$ in D.

We discover the uniqueness of solutions from an analogous multivariate case as was discussed in section 5.1. Now, we can describe a variant of the 4th order Runge-Kutta method for systems of initial-value problems. Partition the interval $[a, b]$ into N subintervals with mesh points $t_j = a + hj$. Then let w_{ij} denote an approximation for $u_i(t_j)$ (the ith solution $u_i(t)$ at the jth

mesh point t_j). Then:

$$
\begin{aligned}
w_{i,0} &= \alpha_i \\
k_{1,i} &= hf_i(t_j, w_{1,j}, w_{2,j}, ..., w_{m,j}), \ \ \forall i \in [1, m] \\
k_{2,i} &= hf_i\left(t_j + \frac{h}{2}, w_{1,j} + \frac{1}{2}k_{1,1}, w_{2,j} + \frac{1}{2}k_{1,2}, ..., w_{m,j} + \frac{1}{2}k_{1,m}\right), \ \ \forall i \in [1, m] \\
k_{3,i} &= hf_i\left(t_j + \frac{h}{2}, w_{1,j} + \frac{1}{2}k_{2,1}, w_{2,j} + \frac{1}{2}k_{2,2}, ..., w_{m,j} + \frac{1}{2}k_{2,m}\right), \ \ \forall i \in [1, m] \\
k_{4,i} &= hf_i(t_j + h, w_{i,j} + k_{3,1}, w_{2,j} + k_{3,2}, ..., w_{m,j} + k_{3,m}), \ \ \forall i \in [1, m] \\
w_{i,j+1} &= w_{i,j} + \frac{1}{6}(k_{1,i} + 2k_{2,i} + 2k_{3,i} + k_{4,i}) \ \ \forall i \in [1, m]
\end{aligned}
$$

An mth order initial value problem

$$y^{(m)}(t) = f(t, y, y', ..., y^{(m-1)}), \ \ a \leq t \leq b,$$

with initial conditions $y(a) = \alpha_1, ..., y^{(m-1)}(a) = \alpha_m$ can be converted into a system of first-order equations. Let $y^{(i)}(t) = u_{i+1}(t)$, so $u_1(t) = y(t)$, $u_2(t) = y'(t)$, etc. so that $u_{i+1} = \frac{du_i}{dt}$. We can then form a system of equations as before using these assignments and solve using our above Runge-Kutta method for systems.

5.6 Multistep Methods

The methods we've seen so far to calculate w_{i+1} are **one-step methods**. This means that they only depend on mesh points t_i and perhaps t_{i+1}. However, remember that as j increases the error $|w_j - y_j|$ also increases. For that reason, it might be valuable to use *earlier* mesh points in our approximation, since the approximations at earlier mesh points are likely much more accurate than the approximation at the previous mesh point.

Consider a standard initial-value problem

$$y' = f(t, y), \ \ a \leq t \leq b \ \ y(a) = \alpha.$$

The m**-step multistep method** for solving this initial value problem is of the form

$$
\begin{aligned}
w_{i+1} = &a_{m-1}w_i + a_{m-2}w_{i-1} + ... + a_0 w_{i+1-m} \\
&+ h[b_m f(t_{i+1}, w_{i+1}) + b_{m-1}f(t_i, w_i) \\
&+ ... + b_0 f(t_{i+1-m}, w_{i+1-m})]
\end{aligned}
$$

Here, i takes on values from $m-1$ to $N-1$, and a_j and b_j are constants. When the value b_m is 0, the method is **explicit** or **closed** (w_{i+1} only appears on the LHS); otherwise, it is **implicit** or **open**. Here we explore two m-step multistep methods that were both developed to help model physical phenomena; fluid mechanics and ballistic equations.

Fourth-Order Adams-Bashforth Technique

For $i = 3, 4, ..., N-1$ the following is an explicit four-step method. For initial conditions $w_0 = \alpha_0,\ w_1 = \alpha_1,\ w_2 = \alpha_2,\ w_3 = \alpha_3$:

$$\begin{aligned} w_{i+1} =& w_i + \frac{h}{24}[55f(t_i, w_i) - 59f(t_{i-1}, w_{i-1}) \\ &+ 37f(t_{i-2}, w_{i-2}) - 9f(t_{i-3}, w_{i-3})] \end{aligned}$$

Fourth-Order Adams-Moulton Technique

For $i = 2, 3, ..., N-1$ the following is an implicit three-step method. For initial conditions $w_0 = \alpha_0,\ w_1 = \alpha_1,\ w_2 = \alpha_2$:

$$\begin{aligned} w_{i+1} =& w_i + \frac{h}{24}[9f(t_{i+1}, w_{i+1}) + 19f(t_i, w_i) \\ &- 5f(t_{i-1}, w_{i-1}) + f(t_{i-2}, w_{i-2})] \end{aligned}$$

Do not be alarmed by the need for several initial conditions; we typically use a different method (i.e. Runge-Kutta) to find approximations at the first few mesh points and then proceed with multistep methods from there on. Implicit methods are generally more accurate than explicit methods, but we cannot always solve the implicit equation for w_{i+1}.

In order to derive a more general m-step method, we begin with our initial value problem and integrate it over the interval $[t_i, t_{i+1}]$, so

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

Since we cannot take the integral of f directly (as this would require knowing the solution already), we replace f with the interpolating polynomial $P(t)$ interpolating the i points $(t_0, w_0), ..., (t_i, w_i)$. Then our approximation is $y(t_{i+1}) \approx w_i + \int_{t_i}^{t_{i+1}} P(t)dt$.

As an example, take the Adams-Bashforth method from earlier. Suppose

instead of the four-step method, we want a general m-step method. We use Newton's backward differences to find an interpolating polynomial, since it makes the most use of recently calculated data. From this, we get the following:

$$\begin{aligned}\int_{t_i}^{t_{i+1}} f(t, y(t))dt &\approx \int_{t_i}^{t_{i+1}} \sum_{k=0}^{m-1} (-1)^k \binom{-s}{k} \nabla^k f(t_i, y(t_i))dt \\ &= \sum_{k=0}^{m-1} \nabla^k f(t_i, y(t_i)) h(-1)^k \int_0^1 \binom{-s}{k} ds.\end{aligned}$$

Using a divided differences table and integral table, we can verify the formulation we have shown above, as well as find Adams-Bashforth methods of various orders.

Truncation Error

The local truncation error for a multistep method as defined previously is

$$\begin{aligned}\tau_{i+1}(h) =& \frac{y(t_{i+1}) - a_{m-1}y(t_i) - ... - a_0 y(t_{i+1-m})}{h} \\ & - [b_m f(t_{i+1}, y(t_{i+1})) + ... + b_0 f(t_{i+1-m}, y(t_{i+1-m}))]\end{aligned}$$

5.6.1 Predictor-Corrector Models

In practice, we rarely use implicit multistep methods in isolation because it is often difficult to solve for w_{i+1}. We usually use implicit methods to correct approximations made by an explicit method; this combination of an explicit predictor and an implicit corrector is called a **predictor-corrector method**. Typically, we'll take the output of an explicit model and use it as an approximation for w_{i+1} on the RHS of the implicit model. A classic example is to use Adams-Bashforth for the explicit portion and Adams-Moulton for the implicit; we call this the **Adams-Bashforth-Moulton method**.

Milne-Simpson Method

Another predictor-corrector model involves the following:
Milne's Method (explicit)

$$w_{i+1} = w_{i-3} + \frac{4h}{3}[2f(t_i, w_i) - f(t_{i-1}, w_{i-1}) + 2f(t_{i-2}, w_{i-2})]$$

Simpson's Method (implicit)

$$w_{i+1} = w_{i-1} + \frac{h}{3}[f(t_{i+1}, w_{i+1}) + 4f(t_i, w_i) + f(t_{i-1}, w_{i-1})]$$

5.7 Stability

Methods presented thusfar approximate the solution to an initial-value problem. However, these are obviously not a comprehensive set of methods; what determines whether not an initial-value estimation method is *good* or not?

5.7.1 One-Step Methods

Consistency

A one-step difference equation with local truncation error $\tau_i(h)$ at the ith step is **consistent** if

$$\lim_{h \to 0} \max_{1 \leq i \leq n} |\tau_i(h)| = 0.$$

Convergence

A one-step difference method is **convergent** with respect to the differential equation if

$$\lim_{h \to 0} \max_{1 \leq i \leq n} |w_i - y(t_i)| = 0.$$

The definition of convergence makes sense intuitively; as we get an infinite amount of granularity, we expect the absolute error to disappear. Consistency means something slightly different – it implies that as the step size approaches 0, the difference equation increasingly resembles the original differential equation.

Due to round-off errors, we often run into tricky situations even with convergent methods. Instead of directly testing the convergence, we test the notion of **stability**, where a minor perturbation to the input results in a minor perturbation to the output.

Stability

Suppose we approximate the initial value problem

$$y' = f(t, y), \quad a \leq t \leq b \;\; y(a) = \alpha$$

is approximated by

$$w_0 = \alpha$$
$$w_{i+1} = w_i + h\phi(t_i, w_i, h)$$

Suppose there exists a positive h_0 with ϕ continuous and Lipschitz in w with constant L on

$$D = \{(t, w, h) | a \leq t \leq b, \; -\infty < w < \infty, \; 0 \leq h \leq h_0\}.$$

This method is **stable**. It is convergent if and only if it is consistent, i.e.

$$\phi(t, y, 0) = f(t, y), \quad a \leq t \leq b.$$

Finally, if a function τ exists and, for each $i = 1, 2, ..., N$, the local truncation error $\tau_i(h)$ satisfies $|\tau_i(h)| \leq \tau(h)$ when $0 \leq h \leq h_0$, then

$$|y(t_i) - w_i| \leq \frac{\tau(h)}{L} e^{L(t_i - a)}.$$

5.7.2 Multistep Methods

Our definitions for consistency, convergence, and stability for multistep methods are similar to our definitions for one-step methods.

Convergence

A multistep difference method is **convergent** if the solution to the difference eqution approaches the solution to the differential equation as the step size approaches 0.

$$\lim_{h \longrightarrow 0} \max_{0 \leq i \leq N} |w_i - y(t_i)| = 0.$$

Consistency

Consistency is slightly different for multistep methods. Note that we have multiple starting values $w_i = \alpha_i$ for our multistep methods. In order for our multistep method to be consistent, the one-step method that generates the starting values must *also* be consistent. In other words, we require the following:

$$\lim_{h \longrightarrow 0} |\tau_i(h)| = 0, \;\; i = m, ..., N$$
$$\lim_{x \longrightarrow 0} |\alpha_i - y(t_i)| = 0, \;\; i = 1, ..., m-1$$

Stability is slightly more complicated for multistep methods. For the multistep described by the difference equation

$$w_0 = \alpha_0, w_1 = \alpha_1, ..., w_{m-1} = \alpha_{m-1}$$
$$w_{i+1} = a_{m-1}w_i + a_{m-2}w_{i-1} + ... + \alpha_0 w_{i+1-m} + hF(t_i, h, w_{i+1}, w_i, ..., w_{i+1-m}),$$

there exists a **characteristic polynomial**

$$P(\lambda) = \lambda^m - a_m\lambda^{m-1} - ... - \alpha_1\lambda - \alpha_0.$$

The stability of the multistep method depends on the roots of this polynomial. If $|\lambda_i| \leq 1$ for each i, and all the roots are simple roots, then the difference equation is said to satisfy the **root condition**. Keep in mind that the roots are in $\mathbb{C}$, not just in $\mathbb{R}$.

1. Methods that satisfy the root condition and have $\lambda = 1$ as the only root with magnitude 1 are **strongly stable**.

2. Methods that satisfy the root condition and have multiple roots with magnitude 1 are **weakly stable**.

3. Methods which do no not satisfy the root condition are **unstable**.

A multistep method is stable if and only if it satisfies the root condition. If the difference method is consistent with the differential equation, then the method is stable if and only if it is also convergent.

5.8 Stiff Differential Equations

Throughout this chapter, we have seen multiple methods for approximating the solutions to initial-value problems. All of the methods have an error term that depends on a (higher) derivative of the solution. If we can reasonably bound the derivative, this is completely fine. Even if the derivative grows with step size, we can keep it under control as long as the solution is growing as well.

The major issues arise when the derivative grows very fast and the solution does not. These equations are those whose solutions have a term of the form e^{-ct}, and are called **stiff differential equations**. This term is called a **transient term**. As t increases, the solution itself decays rapidly to 0. However, its nth derivative has magnitude $c^n e^{-ct}$, which decays much more slowly (or can even grow exponentially).

Fortunately, there is a simple way to know how a numerical method will behave when confronted with a stiff differential equation. The simplest exponential equation we can think of $y(t) = \alpha e^{\lambda t}$. Note that $y' = \lambda \alpha e^{\lambda t}$, or $y' = \lambda y$. In fact, we can simply test our numerical method on this equation, the **test equation**, with

$$y' = \lambda y, \quad y(0) = \alpha, \quad \lambda < 0.$$

The **steady-state** (the solution as t tends to infinity) is 0, so we have an easy way of knowing how to benchmark our methods.

In general, when we apply a one-step difference method to the test equation, we can get an equation of the form

$$w_{i+1} = Q(h\lambda)w_i.$$

The accuracy of the method depends on how well $Q(h\lambda)$ approximates $e^{h\lambda}$, and the error will be uncontrollable if $|Q(h\lambda)| > 1$.

When we apply a multi-step method to the test equation, we get

$$w_{j+1} = (1 - h\lambda b_m)w_{j+1} - (a_{m-1} + h\lambda b_{m-1})w_j - ... - (a_0 + h\lambda b_0)w_{j+1-m} = 0.$$

Rearranging terms, we form a different **characteristic polynomial**

$$Q(z, h\lambda) = (1 - h\lambda b_m)z^m - (a_{m-1} + h\lambda b + m - 1)z^{m-1} - ... - (a_0 + h\lambda b_0).$$

If we let $\beta_1, ..., \beta_m$ be the roots of this characteristic polynomial, we can rewrite it as

$$w_j = \sum_{k=0}^{m} c_k(\beta_k)^j, \quad j = 0, ..., N.$$

For stability, $|\beta_k| < 1$.

Now we have a way of knowing whether or not a method is unstable, i.e. whether or not it is only accurate for very small step sizes. If we determine a method is unstable, it would be beneficial to also know how much we need to reduce the step size by to get good solutions.

Region of Stability

The **region of stability** R for a one-step method is

$$R = \{h\lambda \in \mathbb{C} | \, |Q(h\lambda)| < 1\}.$$

For a multistep method, it is

$$R = \{h\lambda \in \mathbb{C} | \, |\beta_k| < 1\},$$

where β_k is a root of the characteristic polynomial $Q(z, h\lambda)$.

A method can only be applied to a stiff equation if $h\lambda$ is within the region of absolute stability, and this restricts the value of h, the step size. This criterion places a very strict limit on the magnitude of h. A method is **A-stable** if the entire left half-plane is contained within the region of stability. That is to say, the stability criterion is satisfied for all $h\lambda$ where $\mathfrak{Re}(h\lambda) < 0$. The only A-stable multistep method is the implicit trapezoidal method.

6 Direct Methods for Solving Linear Systems

Systems of linear equations have applications in a wide breadth of fields in science, mathematics, economics, and social sciences. We consider *direct methods* for solving a system of n linear equations in n variables, which

generally has form

$$\begin{aligned} E_1: \quad & a_{11}x_1 + a_{12}x_2 + ...a_{1n}x_n = b_1 \\ E_2: \quad & a_{21}x_1 + a_{22}x_2 + ...a_{2n}x_n = b_2 \\ & \vdots \\ E_n: \quad & a_{n1}x_1 + a_{n2}x_2 + ...a_{nn}x_n = b_n \end{aligned}$$

Key Topics

- 6.1: Triangular Form, Row-Reduction, Back-Substitution, Matrix, Vector, Augmented Matrix, Gaussian Elimination, Pivot
- 6.2: Partial Pivoting, Scaled Partial Pivoting
- 6.3: Matrix Equality, Vector Space, Matrix-Vector Product, Matrix-Matrix Product, Square Matrices, Diagonal Matrices, Identity Matrices, Upper and Lower Triangular Matrices, Singular Matrices, Matrix Inverses, Transpose
- 6.4: Determinant, Minor, Cofactor
- 6.5: **LU** Factorization, Permutation Matrices, Doolittle's Method
- 6.6: Diagonally Dominant Matrices, Positive Definiteness, Principle Submatrices, $\mathbf{LDL}^\top$ Decomposition, Cholesky Decomposition, Band Matrices, Tridiagonal Matrices, Crout's Method

6.1 Linear Systems of Equations

In order to simplify a system of equations, we use 3 simple operations.

1. Equation E_i can be multiplied by any nonzero constant λ, and substitute the result in place of E_i. We write $(\lambda E_i) \to (E_i)$.
2. We can multiply E_j by λ and add E_i, and substitute the result in place of E_i. We write $(E_i + \lambda E_j) \to (E_i)$.
3. We can swap the positions of E_i and E_j. We write $(E_i) \iff (E_j)$.

Our goal is to use the first equation to eliminate the first variable from all other equations, and the nth row to reduce the nth variable from all other

equations. In doing so, we use these operations to get a system of the form

$$\begin{aligned} E_1: \quad & a'_{11}x_1 + a'_{12}x_2 + ...a'_{1n}x_n = b'_1 \\ E_2: \quad & a'_{22}x_2 + ...a'_{2n}x_n = b'_2 \\ & \vdots \\ E_n: \quad & a'_{nn}x_n = b'_n \end{aligned}$$

This is called **reduced** or **triangular** form. We can then solve for the value of x_n in equation E_n and use it to solve for x_{n-1} in equation E_{n-1}. We continue this process until we have the solutions for each x. This process is called **back-substitution.**

6.1.1 Matrices and Vectors

We don't need to write the full system of equations at every step. Really, the only things that are important in this solving process are the coefficients of the values. We can replace the linear system by a n $n \times m$ **matrix**, a rectangular array of elements with n rows and m columns where the values at each position are important, as well as the position of each element. An $1 \times n$ matrix is a n**-dimensional row vector** and an $n \times 1$ matrix is an n**-dimensional column vector**. We can write the coefficients of the LHS of the equation in an $n \times n$ matrix:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$$

The values on the RHS can be written as a column vector:

$$\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

We can construct a single matrix that contains all of this information by constructing the following **augmented matrix** $[\mathbf{A}, \mathbf{b}]$**.**

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{array}\right]$$

We can use the same 3 operations for row-reduction and back-substitution, except now we operate on this matrix instead of on systems of equations. The actual logic of how these substitutions work is the same. Ultimately, we try to do this:

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{array}\right] \rightarrow \left[\begin{array}{cccc|c} a'_{11} & a'_{12} & \dots & a'_{1n} & b'_1 \\ 0 & a'_{22} & \dots & a'_{2n} & b'_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & a'_{nn} & b'_n \end{array}\right] \rightarrow \left[\begin{array}{cccc|c} a''_{11} & 0 & \dots & 0 & b''_1 \\ 0 & a''_{22} & \dots & 0 & b''_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & a''_{nn} & b''_n \end{array}\right]$$

This procedure is known as **Gaussian elimination**. Notice that this example breaks when we get a "0" as an element in one of the spots a_{ii} (the **main diagonal**) of the matrix. This 0 element is called a **pivot**, and we need a way to get around it to be able to make our algorithm robust. What we do is start from the pivot and look at the elements below it in the same column. If we find an element that is nonzero, we swap the row of the pivot and the row of the nonzero element and continue.

The total number of multiplications/divisions for Gaussian elimination is $\frac{n^3}{3} + n^2 - \frac{n}{3}$. The total number of additions/subtractions for Gaussian elimination is $\frac{n^3}{3} + \frac{n^2}{2} - \frac{5n}{6}$.

6.2 Pivoting Strategies

Sometimes we would like to swap rows even if we don't have a 0 pivot element. When the element we are using for elimination is very small in magnitude compared to another element in its column, we introduce round-off error which compounds though the elimination and back-substitution process.

Partial Pivoting

To counter the above issue, we can use **partial pivoting**. In this scenario, we examine element a_{kk} as our pivot element. We then determine the row p such that

$$|a_{pk}| = \max_{k \leq i \leq n} |a_{ik}|.$$

We then perform the operation $(E_k) \iff (E_p)$. In other words, we swap the "pivot" row with the row that has the largest element in that column. Note that we don't have to just do this with rows; if necessary, we can also do this with column vectors.

Partial pivoting is also not enough in all cases. If all elements in a row are consistently significantly larger than all elements in another, we can get the same compounding error inaccuracy as before. Instead, we introduced **scaled partial pivoting**. Here, we don't just swap in the row with the largest element in the appropriate column; we swap in the row with the largest element in the appropriate column *compared to the other elements in that row.* We find a scaling factor

$$s_i = \max_{i \leq j \leq n} |a_{ij}|.$$

If this scaling factor is 0, there is no solution. The row for row interchange is then

$$\frac{|a_{pk}|}{s_p} = \max_{k \leq i \leq n} \frac{|a_{ik}|}{s_k},$$

and we perform $(E_k) \iff (E_p)$.

6.3 Linear Algebra

Here we define some basic concepts in linear algebra.

Two matrices A and B are **equal** if they have the same number of rows and columns and the same elements in the same positions.

The **sum** of two $n \times m$ matrices $\mathbf{A}$ and $\mathbf{B}$ is an $n \times matrix$ $\mathbf{C}$ such that $c_{ij} = a_{ij} + b_{ij}$.

The **scalar multiplication** of a matrix $\mathbf{A}$ with a real number λ is a matrics

$\mathbf{C}$ with $c_{ij} = \lambda a_{ij}$. If we denote $\mathbf{O}$ as a matrix of all 0's and $-A$ as the matrix whose entries are $-a_{ij}$, we have enough information to define the set of all matrices as a **vector space** over the field of real numbers. The actual properties required for this are omitted here; they are covered in depth in Math110, Math104, and Math113.

A **matrix-vector** product of an $n \times m$ matrix $\mathbf{A}$ and an m-dimensional column vector $\mathbf{b}$ is an n-dimensional column vector $\mathbf{Ab}$ where the jth entry in $\mathbf{Ab}$ is $\sum_{i=1}^{m} a_{ji} b_i$.

A **matrix-matrix** product of an $n \times m$ matrix $\mathbf{A}$ and an $m \times p$ matrix $\mathbf{B}$ is the $n \times p$ matrix $\mathbf{C}$ where $c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}$. Note that by this definition, matrix-matrix multiplication is not commutative.

A **square** matrix has the same number of rows and columns. A **diagonal** matrix $\mathbf{A}$ is one such that $a_{ij} = 0$ if $i \neq j$. The **identity matrix I** is a diagonal matrix where all entries on the diagonal are 1. An **upper triangular** matrix has a 0 for all entries below the main diagonal; a **lower triangular** matrix has a 0 for all entires above the main diagonal.

A matrix $\mathbf{A}$ is **nonsingular** if there exists a matrix $\mathbf{A}^{-1}$ such that $\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$. $\mathbf{A}^{-1}$ is the **inverse** of $\mathbf{A}$. A matrix that has no inverse is called **singular**.

Matrix Inverses

For a nonsingular matrix $\mathbf{A}$:

1. $\mathbf{A}^{-1}$ is unique
2. $\mathbf{A}^{-1}$ is nonsingular and $(\mathbf{A}^{-1})^{-1} = \mathbf{A}$
3. If $\mathbf{B}$ is also nonsingular, then $(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$.

We can find the inverse of a matrix by solving the augmented matrix $[\mathbf{A}|\mathbf{I}]$ using Gaussian elimination.

The **transpose** of an $n \times m$ matrix $\mathbf{A}$, denoted $\mathbf{A}^\top$, is an $m \times n$ matrix where $a_{ij}^\top = a_{ji}$. A square matrix is **symmetric** if $\mathbf{A} = \mathbf{A}^\top$. If the following operations are possible, then $(\mathbf{A} + \mathbf{B})^\top = \mathbf{A}^\top + \mathbf{B}^\top$, $(\mathbf{AB})^\top = \mathbf{B}^\top\mathbf{A}^\top$, and $(\mathbf{A}^{-1})^\top = (\mathbf{A}^\top)^{-1}$.

6.4 Determinants

The **determinant** of a matrix is a value that allows us to determine the existence and uniqueness of the results of the corresponding linear system.

Determinant

For a square matrix $\mathbf{A}$, we define the following:

1. If $A = [a]$ is a 1×1 matrix, then $\det \mathbf{A} = a$.
2. for an $n \times n$ matrix with $n > 1$, the **minor** M_{ij} is the determinant of the $(n-1) \times (n-1)$ matrix that results from removing row i and column j from the original matrix.
3. The **cofactor** A_{ij} associated with M_{ij} is $A_{ij} = (-1)^{i+j} M_{ij}$.
4. The **determinant** of an $n \times n$ matrix A is then either of the following

$$\det \mathbf{A} = \sum_{j=1}^{n} a_{ij} A_{ij} \quad \text{for any row } i$$

$$\det \mathbf{A} = \sum_{i=1}^{n} a_{ij} A_{ij} \quad \text{for any column } j$$

We additionally list some properties of the determinant, and its relationship to the steps of Gaussian elimination.

1. If any row or column of A has only 0 as elements, then $\det \mathbf{A} = 0$.
2. If two rows or columns of $\mathbf{A}$ are the same then $\det \mathbf{A} = 0$.
3. If we perform a *row swap* $(E_i)(E_j)$ when reducing $\mathbf{A}$ to $\tilde{\mathbf{A}}$ then $\det \mathbf{A} = -\det \tilde{A}$.
4. If we perform a scalar multiplication $(\lambda E_i) \to (E_i)$, then $\det \tilde{\mathbf{A}} = \lambda \det \mathbf{A}$.
5. If $\mathbf{B}$ is also an $n \times n$ matrix, then $\det \mathbf{AB} = \det \mathbf{A} + \det \mathbf{B}$.
6. $\det \mathbf{A}^\top = \det \mathbf{A}$, and $\det \mathbf{A}^{-1} = (\det \mathbf{A})^{-1}$.
7. If $\mathbf{A}$ is upper triangular, lower triangular, or diagonal, then $\det \mathbf{A}$ is the product of the elements along the main diagonal.

Determinants give us key insight into nonsingularity. The following statements are all equivalent.

1. The equation $\mathbf{Ax} = 0$ has a unique solution $\mathbf{x} = 0$
2. $\mathbf{Ax} = \mathbf{b}$ has a unique solution for any n-dimensional column vector $\mathbf{b}$.
3. $\mathbf{A}^{-1}$ exists.
4. $\det \mathbf{A} \neq 0$.
5. Gaussian elimination with pivoting can be performed on $\mathbf{A}$.

6.5 Matrix Factorization

Solving a linear system using our elimination methods takes $O(n^3)$, as we saw previously. However, the back-substitution portion of an upper triangular matrix takes only $O(n^2)$. Here we explore a way to split a matrix into a lower triangular and upper triangular matrix for this purpose. Once we have these two matrices, which we denote $\mathbf{A} = \mathbf{LU}$,we can solve $\mathbf{LUx} = \mathbf{b}$ by letting $\mathbf{y} = \mathbf{Ux}$, solving $\mathbf{Ly} = \mathbf{b}$, and then solving for $\mathbf{x}$. This reduces the total time from $O(n^3)$ to $O(n^2)$.

In Gaussian elimination, we begin by eliminating all the values below the main diagonal. We do this through the operation $(E_j - \lambda_{j,i}E_i) \rightarrow (E_j)$, where $i < j$. The matrix $\mathbf{L}$ is then

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & \dots & 0 & 0 \\ -\lambda_{2,1} & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ -\lambda_{n,1} & -\lambda_{n,2} & \dots & -\lambda_{n,n-1} & 1 \end{bmatrix}$$

$\mathbf{U}$ is simply the upper triangular matrix we get from our normal row-reduction. This method is known as **Doolittle's Method** and requires 1s along the main diagonal of $\mathbf{L}$. Alternate methods include **Crout's method**, where we require 1s on the main diagonal of $\mathbf{U}$, and **Cholesky's method**, which requires $l_{ii} = u_{ii}$.

The above method assumes that Gaussian elimination can succeed without pivoting. If we need pivoting to successfully perform Gaussian elimination on $\mathbf{A}$, we multiply $\mathbf{A}$ by a matrix $\mathbf{P}$, a **permutation matrix**. The permutation matrix is an identity matrix with the pivoting operations performed

on it, such that $\mathbf{PA}$ does not require pivoting. We can then factor $\mathbf{PA}$ into $\mathbf{LU}$. Since $\mathbf{P}^{-1} = \mathbf{P}^\top$, $\mathbf{A} = \mathbf{P}^{-1}\mathbf{LU} = (\mathbf{P}^\top\mathbf{L})\mathbf{U}$. Note that while $\mathbf{U}$ is still upper triangular, but $\mathbf{P}^\top\mathbf{L}$ is not necessarily lower triangular.

6.6 Special Matrices

Diagonally Dominant Matrices

A matrix is **diagonally dominant** if the element in each row that is part of the main diagonal is greater than the sum of the other elements in the row.

$$|a_{ii}| \geq \sum_{j \neq i} |a_{ij}|$$

A diagonally dominant matrix is **strictly diagonally dominant** if

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|.$$

Strictly diagonally dominant matrices are nonsingular and do not require row interchanges during Gaussian elimination.

Positive Definite Matrices

A matrix $\mathbf{A}$ is **positive definite** if it is symmetric and for every nonzero n-dimensional vector $\mathbf{x}$, $\mathbf{x}^\top\mathbf{Ax} > 0$. This implies that $\mathbf{A}$ has an inverse, $a_{ii} > 0$ for all i, $\max a_{k,j} \leq \max a_{i,i}$, and $(a_{ij})^2 < a_{ii}a_{jj}$ for all i, j.

Leading Principle Submatrices

The kth **leading principle submatrix** of $\mathbf{A}$ is a $k \times k$ matrix containing the elements of the first k rows and k columns of $\mathbf{A}$. A symmetric matrix is only positive definite it all of its leading principle submatrices have positive determinants.

A symmetric matrix $\mathbf{A}$ is positive definite if and only if $\mathbf{Ax} = \mathbf{b}$ can be solved without row interchanges with a positive main diagonal. From this, we get two interesting facts:

1. $\mathbf{A}$ is positive definite if and only if it can be factored into $\mathbf{LDL}^\top$, where $\mathbf{L}$ is lower triangular with 1s on its main diagonal, and $\mathbf{D}$ is a diagonal matrix with only positive entries.

2. $\mathbf{A}$ is positive definite if and only if it can be factored into $\mathbf{L}\mathbf{L}^\top$, where $\mathbf{L}$ is lower triangular. This $\mathbf{L}$ is not the same $\mathbf{L}$ as in (1).

For (1), we can very simply pattern match to find the decomposition; the process is illustrated here for a 3×3 matrix

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{bmatrix} d_1 & 0 & 0 \\ 0 & d_2 & 0 \\ 0 & 0 & d_3 \end{bmatrix} \begin{bmatrix} 1 & l_{21} & l_{31} \\ 0 & 1 & l_{32} \\ 0 & 0 & 1 \end{bmatrix}$$
$$= \begin{bmatrix} d_1 & d_1 l_{21} & d_1 l_{31} \\ d_1 l_{21} & d_2 + d_1 l_{21}^2 & d_2 l_{32} + d_1 l_{21} l_{31} \\ d_1 l_{31} & d_1 l_{21} l_{31} + d_2 l_{32} & d_1 l_{31}^2 + d_2 l_{32}^2 + d_3 \end{bmatrix}$$

Equivalently, we can use Doolittle's method and perform Gaussian elimination on the above; the main diagonal of the upper triangular matrix in Gaussian elimination corresponds to $\mathbf{D}$. Our $\mathbf{L}\mathbf{L}^\top$ decomposition, also called **Cholesky decomposition**, follows a similar process.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & l_{31} \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{bmatrix}$$
$$= \begin{bmatrix} l_{11}^2 & l_{11} l_{21} & l_{11} l_{31} \\ l_{11} l_{21} & l_{21}^2 + l_{22}^2 & l_{21} l_{31} + l_{22} l_{32} \\ l_{11} l_{31} & l_{21} l_{31} + l_{22} l_{32} & l_{31}^2 + l_{32}^2 + l_{33}^2 \end{bmatrix}$$

Band Matrices

A **band matrix** is an $n \times n$ matrix where the only nonzero entries are on diagonals of the matrix. They are defined by two numbers, p and q, where p describes the number of diagonals above and including the main diagonal, and q describes the number of diagonals below and including the main diagonal. The value $w = p + q - 1$, the total number of diagonals with nonzero entries, is called the **bandwidth**. A band matrix with $p = q = 2$ is known as a **tridiagonal matrix**. An example of a tridiagonal matrix is shown below:

$$\begin{bmatrix} a_{11} & a_{21} & 0 & \dots & 0 \\ a_{21} & a_{22} & a_{23} & \dots & 0 \\ 0 & a_{32} & a_{33} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & a_{nn} \end{bmatrix}$$

A tridiagonal matrix also has a unique **LU** factorization. This method, known as **Crout's method**, allows us to decompose a tridiagonal matrix **A** into **L** and **U**, where **L** is a lower triangular band matrix (i.e. $p = 0$) and **U** is an upper triangular band matrix with 1s on its main diagonal.

$$\begin{aligned}\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} &= \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ 0 & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} 1 & u_{12} & 0 \\ 0 & 1 & u_{23} \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} l_{11} & l_{11}u_{12} & 0 \\ l_{21} & l_{21}u_{12} + l_{22} & l_{22}u_{23} \\ 0 & l_{32} & l_{32}u_{23} + l_{33} \end{bmatrix}\end{aligned}$$

If elements of the main diagonal of a tridiagonal matrix are larger in magnitude than the sum of the magnitudes of the other elements in the same row, and $a_i, a_{i-1}, a_{i+1} \neq 0$, the matrix is nonsingular. That is to say, if $|a_{11}| > |a_{12}|$, $|a_{ii}| \geq |a_{i,i+1}| + |a_{i,i-1}|$, and $|a_{nn}| > |a_{n,n-1}|$, with $a_i, a_{i-1}, a_{i+1} \neq 0$, the matrix is nonsingular and each l_{ii} is nonzero.