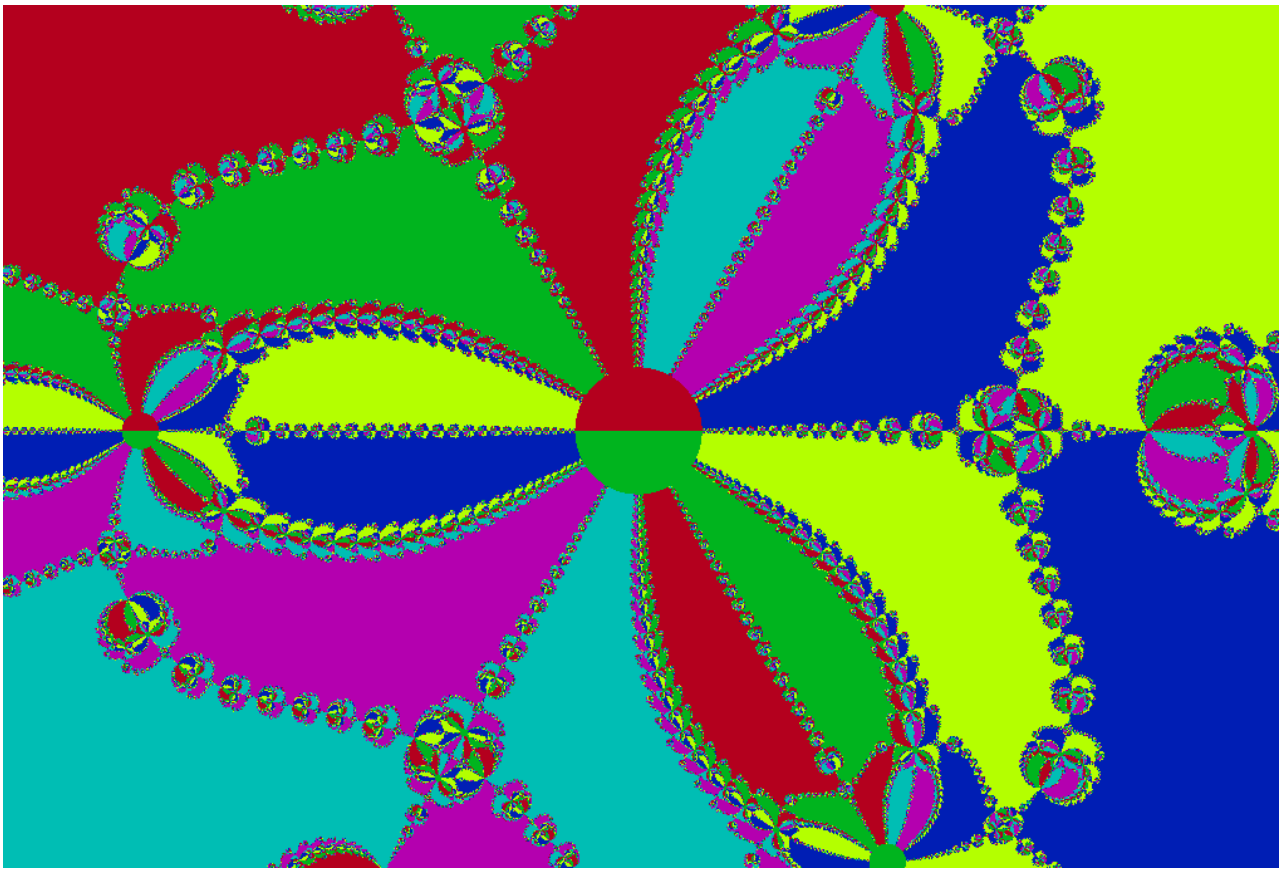
Semester: Fall 2022
Instructor: Lessard, Jean-Philippe

Numerical Analysis



Course Content. Error analysis. Numerical solutions of equations by iteration. Interpolation. Numerical differentiation and integration. Introduction to numerical solutions of differential equations.

Summary of Contents

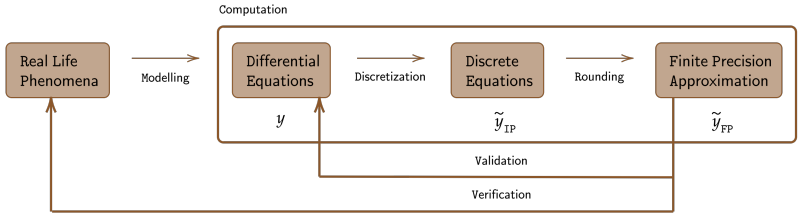
Topic 1. Error Sources and Floating Point Numbers

Many problems in science and engineering are described using continuous variables and then modelled by differential equations.

Fluid Mechanics	Laws of Motion
Electromagnetism	Maxwell's Equations
Stock Pricing	Black-Scholes Equation
Machine Learning	Neural Networks

Since computers have finite resources, we must approximate solutions to these problems in finite precision. This involves,

Validation	- Accurate approximations of solutions
	- Efficiency in costs
	- Stability during computation
Verification	- Accurate models of real-world phenomenon



We will use the following notation,

x	Exact input value
$\tilde{x}$	Approximate input value
f	Exact output value
$\tilde{f}_{\mathrm{FP}}$	Finite-precision approximation

Rounding and Discretization

Definition (Discretization Error)**.** The **discretization error** is an infinite-precision measurement of the difference between the exact output and the numerical output,

$$\left|f(x) - \tilde{f}_{\mathrm{IP}}(x)\right|$$

Definition (Rounding Error)**.** The **rounding error** is an infinite-precision measurement of the difference between numerical outputs with infinite and finite precision,

$$\left|\tilde{f}_{\mathrm{IP}}(x) - \tilde{f}_{\mathrm{FP}}(x)\right|$$

Remark. Error sources from computation can be bounded by,

$$\begin{aligned}\text{Error} &:= |f(x) - \tilde{f}(\tilde{x})| \\ &\leq \underbrace{|f(x) - f(\tilde{x})|}_{\text{Data Error}} + \underbrace{|f(\tilde{x}) - \tilde{f}_{\texttt{FP}}(\tilde{x})|}_{\text{Computation Error}}\end{aligned}$$

Assuming no data error, i.e., $x = \tilde{x}$, we have,

$$\begin{aligned}|f(\tilde{x}) - \tilde{f}_{\texttt{FP}}(x)| &= |f(x) - \tilde{f}_{\texttt{FP}}(x)| \\ &\leq \underbrace{\left|f(x) - \tilde{f}_{\texttt{IP}}(x)\right|}_{\text{Discretization Error}} + \underbrace{\left|\tilde{f}_{\texttt{IP}}(x) - \tilde{f}_{\texttt{FP}}(x)\right|}_{\text{Rounding Error}}\end{aligned}$$

as our bound on the computation error.

Example 1: Computation Error

Let $f(x) = \sin(x)$. We will approximate f using,

$$\begin{aligned}\tilde{f}_{\texttt{IP}}(x) &= x \text{ in infinite precision} \\ \tilde{f}_{\texttt{FP}}(x) &= x \text{ with the first four non-zero digits}\end{aligned}$$

What is the discretization and rounding error at $x = 1/9$?

We begin with the following computations,

$$\begin{aligned}f(1/9) &= \sin(1/9) = 0.11088262851\ldots \\ \tilde{f}_{\texttt{IP}}(1/9) &= 1/9 = 0.11111111111\ldots \\ \tilde{f}_{\texttt{FP}}(1/9) &= 0.1111\end{aligned}$$

allowing us to conclude for the choice of $x = 1/9$ that,

1. The **discretization error** is defined by $\left|\sin(x) - \tilde{f}_{\texttt{IP}}(x)\right|$. So,

$$|\sin(1/9) - 1/9| = 0.0002284826\ldots$$

2. The **rounding error** is defined by $\left|\tilde{f}_{\texttt{IP}}(x) - \tilde{f}_{\texttt{FP}}(x)\right|$. So,

$$|1/9 - 0.1111| = 0.0000111111\ldots$$

3. The **computation error** is defined by $|f(x) - \tilde{f}_{\texttt{FP}}(x)|$. So,

$$\begin{aligned}\left|f(1/9) - \tilde{f}_{\texttt{FP}}(1/9)\right| &= 0.00021737149 \\ &\leq 0.0002284826 + 0.0000111111 \\ &= \text{Discretization Error} + \text{Rounding Error}\end{aligned}$$

Measuring Error

Consider a non-zero real number a and its approximation $\tilde{a}$.

Definition (Absolute Error)**.** The **absolute error** is the difference between measured or inferred value $\tilde{x}$ and the actual value of x,

$$|x - \tilde{x}|$$

Remark. The absolute error is inadequate due to the fact that it does not give any details regarding the importance of the error.

Definition (Relative Error)**.** The **relative error** is the ratio of the absolute error of the measurement to the actual value of x,

$$\frac{1}{|x|} \cdot |x - \tilde{x}|$$

The relative error is defined to be 1 when $x = 0$.

Remark. This allows us to determine the magnitude of the absolute error in terms of the actual size of the measurement.

Example 2: Absolute and Relative Error

Let $f(x) = \sqrt{x}$. We will approximate f using,

$$\tilde{f}_{\mathsf{FP}}(x) = \sqrt{x} \text{ with the first two non-zero digits}$$

What is the absolute and relative error at $x = 2$?

We begin with the following computations,

$$f(2) = \sqrt{2} = 1.41421356237\ldots$$
$$\tilde{f}_{\mathsf{FP}}(2) = 1.4$$

allowing us to conclude for the choice of $x = 2$ that,

1. The **absolute error** is defined by $|f(x) - \tilde{f}_{\mathsf{FP}}(x)|$. So,

$$|f(2) - \tilde{f}(2)| = 0.01421356237\ldots$$

2. The **relative error** is defined by $\frac{1}{|a|} \cdot |a - \tilde{a}|$. So,

$$\frac{|f(2) - \tilde{f}(2)|}{|f(2)|} = 0.01005050633\ldots \approx 1\%$$

However, multiplying $x = 2$ by 10^6 gives an absolute error of,

$$\left| f\left(2 \times 10^6\right) - \tilde{f}\left(2 \times 10^6\right) \right| = 14.21356237\ldots$$

while the relative error remains at $\approx 1\%$.

Floating-Point Numbers

We can express a non-zero real number a in base β as,

$$\begin{aligned} a &= \pm\left(a_n \times \beta^n + \ldots + a_1 \times \beta^1 + a_0 \times \beta^0 + a_{-1} \times \beta^{-1} + \ldots\right) \\ &= \pm\left(a_n \cdots a_1 a_0 \cdot a_{-1} \ldots\right)_\beta \\ &= \pm\left(d_0 \cdot d_1 d_2 \ldots\right)_\beta \times \beta^E \end{aligned}$$

where d_0 is the first non-zero digit of a and E is called the exponent.

Remark. There are two methods that can be used to approximate a,

1. **Truncating** d_{p-1} after p digits
2. **Rounding** d_{p-1} based on the next digit d_p

Definition (Floating-Point Number System). The **floating-point number system** consists of the following parameters,

β	Base
p	Digits of Precision
$[L, U]$	Exponent Range

Given β, p, L, and U, we can represent a as,

$$\begin{aligned} fl(a) &:= \pm\left(d_0 + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \ldots + \frac{d_{p-1}}{\beta^{p-1}}\right) \times \beta^E \\ &= \pm\underbrace{(d_0 \cdot d_1 \ldots d_{p-1})_\beta}_{\text{Mantissa}} \times \beta^E \end{aligned}$$

where $0 \leq d_i \leq \beta - 1$ for $i \in [p-1]$ and $0 < d_0$ with $L \leq E \leq U$.

Example 3: Exploring Floating-Point Numbers

How many unique decimal floating-point numbers are represented with 4 digits of precision and $[L, U] = [-2, 1]$?

Observe that the largest and smallest floating-point numbers are 99.99 and 0.01, respectively. Suppose that $a \neq 0$. Then,

$$a = \pm (d_0 \cdot d_1 d_2 d_3)_{10} \times 10^E$$

There are 2 choices for the sign, 9 choices for d_0, 10 choices for each of d_1, d_2, d_3, and 4 choices for E. This gives a total of,

$$2 \times 9 \times 10 \times 10 \times 10 \times 4 + 1 = 72001$$

unique numbers. We add 1 for the case where $a = 0$.

The exponent is stored as an unsigned value, so biasing is used to represent the full range of small and large numbers within this convention.

Definition (IEEE754 Standard). Let $a \neq 0$. With s and m as the sign bit and mantissa bits, respectively, we can express a as,

$$fl(a) = (-1)^s \cdot (1.m)_2 \times 2^{e - \texttt{offset}}$$

The **IEEE754 standard** for storing a is,

$fl(a) =$

Sign Bit	Exponent Bits	Mantissa Bits

In **single precision**, `offset` $= 127$, and,

$$fl(a) = \boxed{s} \boxed{e_1 e_2 \dots e_8} \boxed{d_1 d_2 \dots d_{23}}$$

In **double precision**, `offset` $= 1023$, and,

$$fl(a) = \boxed{s} \boxed{e_1 e_2 \dots e_{11}} \boxed{d_1 d_2 \dots d_{52}}$$

Remark. The offset of a floating-point number is,

$$2^{k-1} - 1$$

where k is the number of bits in the exponent.

The exponent E in our floating-point number system is an integer.

Example 4: Single-Precision Floating Point Number

Express $(123)_{10}$ as a single-precision floating point number in binary. That is, $(123)_{10}$ as $(-1)^s (1.m)_2 \times 2^{e-127}$.

Solving for s, m, and e,

1. $s = 0$ because 123 is a positive number

2. $123 \in [2^6, 2^7] = [64, 128]$ so $6 = e - 127$ and $e = 133$
3. Converting $e = (133)_{10}$ to binary,

$$(133)_{10} = 1 \times 2^7 + 1 \times 2^2 + 1 \times 2^0 = (10000101)_2$$

4. To find the mantissa, we solve for m in,

$$m = \frac{123}{2^6} - 1 = \frac{59}{64} = (0.921875)_{10} = (0.111011)_2$$

Combining the previous four steps,

$$(123)_{10} = \begin{array}{|c|c|c|} \hline 0 & 10000101 & 111011000 \cdots 000 \\ \hline \end{array}$$

In general, every operation performed with floating-point numbers generates rounding error, which propagates with long computations.

Definition (Machine Precision). The **machine precision** ϵ_{mach} is an upper bound on the relative approximation error due to rounding in floating point arithmetic. That is, $\forall x \neq 0$,

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

Corollary. $\epsilon_{\text{mach}} = \beta^{1-k}$ for truncating after k digits in base β.

Arithmetic Operations

Given a machine epsilon ϵ, the IEEE 754 standard requires that,

1. $fl(x) = x(1 + \delta)$, where $|\delta| \leq \epsilon$
2. $fl(x \odot y) = (x \odot y)(1 + \delta)$ for $\odot \in \{+, -, \times, /\}$ and $|\delta| \leq \epsilon$
3. $fl(x \odot y) = fl(y \odot x)$ for $\odot \in \{+, x\}$

Example 5: Error Propagation with Addition and Subtraction

Let $\tilde{x} = fl(x) = x\,(1 + \delta_1)$ and $\tilde{y} = fl(y) = y\,(1 + \delta_2)$. Then,

$$\tilde{x} \pm \tilde{y} = x\,(1 + \delta_1) \pm y\,(1 + \delta_2) = x \pm y + x\delta_1 \pm y\delta_2$$

This implies that $(\tilde{x} \pm \tilde{y}) - (x \pm y) = x\delta_1 \pm y\delta_2$. Specifically,

1. The **absolute error** from addition and subtraction has,

$$|(\tilde{x} \pm \tilde{y}) - (x \pm y)| \leq |x|\,|\delta_1| + |y|\,|\delta_2| \leq (|x| + |y|) \cdot \epsilon$$

2. The **relative error** from addition and subtraction has,

$$\frac{|(\tilde{x} \pm \tilde{y}) - (x \pm y)|}{|x \pm y|} \leq \frac{|x| + |y|}{|x \pm y|} \cdot \epsilon$$

Observe that the relative error can become large when,

$$|x \pm y| \approx 0$$

We say that **underflow** occurs when $fl(a)$ is too small to be represented and that **overflow** occurs when $fl(a)$ is too large to be represented.

Topic 2. Iterative Methods for Nonlinear Systems of Equations

Review of Calculus

We write $C([a,b]) := C^0([a,b])$ for the set of continuous functions on $[a,b]$. Moreover, we say that f is a **smooth** function on $[a,b]$ if $f \in C^\infty([a,b])$.

Let $C^n(X)$ be the set of functions with n continuous derivatives on X.

Theorem 1 (Intermediate Value Theorem)**.** Suppose $f \in C[a,b]$. Then,

$$\exists c \in (a,b) \text{ such that } f(c) = y$$

for all $f(a) \leq y \leq f(b)$.

Theorem 2 (Rolle's Theorem)**.** Suppose $f \in C[a,b]$ and f is differentiable on (a,b). If $f(a) = f(b)$, then $\exists c \in (a,b)$ such that,

$$f'(c) = 0$$

Theorem 3 (Mean Value Theorem)**.** Suppose $f \in C[a,b]$ and f is differentiable on (a,b). Then $\exists c \in (a,b)$ with,

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

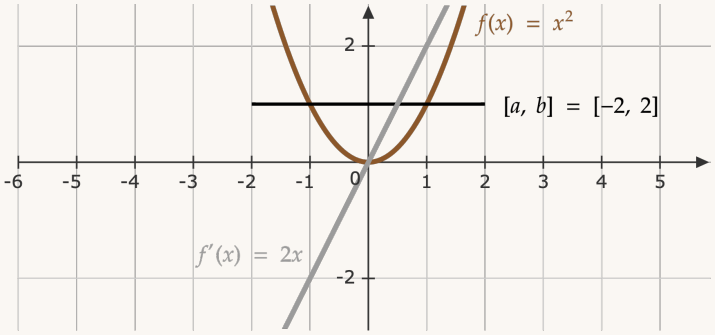
Theorem 4 (Extreme Value Theorem)**.** Suppose $f \in C[a,b]$. Then,

$$\exists c_1, c_2 \in [a,b] \text{ such that } f(c_1) \leq f(x) \leq f(c_2)$$

for all $x \in [a,b]$. That is, f attains both a maximum and a minimum.

Corollary. Suppose f is differentiable on (a,b). Then the numbers c_1 and c_2 occur either at the endpoints of $[a,b]$ or where f' is zero.

Example 6: $f(x) = x^2$



Intermediate Value Theorem

$4 = f(-2) \leqslant 0 \leqslant f(2) = 4$
$\implies \exists\, c \in (-2, 2)\, s.t.\ f(c) = 0$

Rolle's Theorem

$f(-2) = f(2) = 4$
$\implies \exists\, c \in (-2, 2)\, s.t.\ f'(c) = 0$

Mean Value Theorem

$$\exists\, c \in (-2, 2)\, s.t.\ f'(c) = \frac{f(2) - f(-2)}{2 - (-2)} = 0$$

Theorem 5 (Taylor's Theorem)**.** Suppose $f \in C^n([a, b])$ and $f^{(n+1)}$ exists on the interval $[a, b]$. Let $x_0 \in [a, b]$. Then $\forall x \in [a, b]$,

$$\exists \xi(x) \in (x_0, x) \text{ s.t. } f(x) = P_n(x) + R_n(x)$$

where,

$$P_n(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!} (x - x_0)^k$$

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

Remark. $P_n(x)$ is called the n**th Taylor polynomial** for f about x_0, and $R_n(x)$ is called the **truncation error** associated with $P_n(x)$.

Common Taylor Series include,

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n}$$

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

$$(1+x)^k = \sum_{n=0}^{\infty} \binom{n}{k} x^n$$

$$\frac{1}{1-x} = \sum_{k=0}^{\infty} x^k \quad |x| < 1$$

$$\log(1+x) = \sum_{k=0}^{\infty} \frac{(-1)^k x^{k+1}}{k+1} \quad |x| < 1$$

Example 7: Taylor Polynomials and Error Bounds

To find the second Taylor polynomial of,

$$g(x) = \sqrt{1+x} - 1$$

about $x_0 = 0$,

1. $g(x) = (1+x)^{1/2} - 1$ and $g(0) = 0$
2. $g'(x) = \frac{1}{2}(1+x)^{-1/2}$ and $g'(0) = \frac{1}{2}$
3. $g''(x) = -\frac{1}{4}(1+x)^{-3/2}$ and $g''(0) = -\frac{1}{4}$
4. $g'''(x) = \frac{3}{8}(1+x)^{-5/2}$ and $|g'''(\xi(x))| = \frac{3}{8} \cdot \frac{1}{|1+\xi(x)|^{\frac{5}{2}}} \leq \frac{3}{8}$

Thus, $P_2(x) = \frac{x}{2} - \frac{x^2}{8}$ with the error,

$$|g(x) - P_2(x)| = |R_2(x)| \leq \frac{1}{16}$$

In comparison, we can approximate $g(0.0001)$ using $P_2(0.001)$ with 5-digit truncation. This gives a relative error of $\approx 0.001\%$.

Remark. It is sometimes easier to use Taylor series expansion when asked to compute the n-th Taylor polynomial for a function f.

Convergence Order

Definition (Q-Convergence). A sequence $\{\mathbf{x}_k\}_k$ in $\mathbb{R}^n$ converges to $\mathbf{x}^* \in \mathbb{R}^n$ if $\|\mathbf{x}_k - \mathbf{x}^*\| \to 0$ as $k \to \infty$. Suppose $\mathbf{x}_k \neq \mathbf{x}^*$ for sufficiently large k. Then $\{\mathbf{x}_k\}_k$ converges to $\mathbf{x}^*$ at **order** p with an asymptotic error constant $C > 0$ whenever,

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

By convention, we have the following,

- If $p = 1$, the converge is called Q**-linear**
- If $p = 2$, the converge is called Q**-quadratic**
- If $p > 1$, the converge is called Q**-superlinear**

Example 8: The Significance of p

Suppose $\mathbf{x}_k$ converges to $\mathbf{x}^*$ with order p. There exists $r \in (0,1)$ and constants $C_1, C_2 > 0$ so that for $p = 1$ and $p > 2$,

1. The error is reduced by a factor of r between $\mathbf{x}_k$ and $\mathbf{x}_{k+1}$

$$(Linear) \quad C_1 \cdot r^k \leq \|\mathbf{x}_k - \mathbf{x}^*\| \leq C_2 \cdot r^k$$

2. The error is reduced by a factor of r^{p^k} between $\mathbf{x}_k$ and $\mathbf{x}_{k+1}$

$$(Superlinear) \quad C_1 \cdot r^{p^k} \leq \|\mathbf{x}_k - \mathbf{x}^*\| \leq C_2 \cdot r^{p^k}$$

Moreover, the order p is unique.

Remark. Suppose $\mathbf{x}_k$ converges to $\mathbf{x}^*$ with order p and A.E.C $C > 0$. It was left as an exercise to prove that,

$$\lim_{k\to\infty} \frac{\|\mathbf{x}_{k+1} - \mathbf{x}^*\|}{\|\mathbf{x}_k - \mathbf{x}^*\|^q} = \begin{cases} 0, \text{ if } q < p \\ C, \text{ if } q = p \\ \infty, \text{ if } q > p \end{cases} \tag{1}$$

The improvement in digits per iteration depends on the convergence order,

1. In the linear case, the digit improvement rate is constant
2. In the quadratic case, the digit improvement rate doubles
3. In the superlinear case, the digit improvement rate increases

Example 9: Convergence Order

The following sequences $\{x_k\}_k$ converge to zero,

1. $x_k := 2^{-i}$ is linear with A.E.C = $\frac{1}{2}$
2. $x_k := 2^{-2i+1}$ is linear with A.E.C = $\frac{1}{4}$
3. $x_k := 2^{-2^i}$ is quadratic with A.E.C = 1

The notion of Q-convergence may be insufficient to describe a sequence when the limit does not exist. We require a weaker notion,

Definition (R-Linear Convergence). Let $\{\mathbf{x}_k\}_k$ be a sequence that converges to $\mathbf{x}^* \in \mathbb{R}^n$. We say that $\{\mathbf{x}_k\}$ converges **R-linearly** if there is a non-negative real sequence $\{a_k\}_{k=0}^{\infty} \subset \mathbb{R}$ converging Q-linearly to zero so that for sufficiently large k,

$$\|\mathbf{x}_k - \mathbf{x}^*\| \leq a_k$$

Given $f : \mathbb{R}^n \to \mathbb{R}^n$, we are often required to find solutions,

$$\mathbf{x} \in \mathbb{R}^n \text{ such that } f(\mathbf{x}) = \mathbf{0}$$

Example 10: Nonlinear Equations with Known Solutions

1. $f(x) = ax^2 + bx + c$ is solved using the quadratic formula
2. $f(x) = x - e^{-x}$ is solved using the Lambert W function

To use the Lambert W function, remark that $y = xe^x$ if and only if $W(y) = x$.

In general, nonlinear systems of equations may have one, two, or even infinitely-many roots. We want to develop a method for approximating these roots by rapidly converging sequences of points.

Bisection Method

We will make use of the following fact,

> **Lemma 1.** Suppose $f \in C([a,b])$ and,
>
> $$\text{sign}(f(a)) \neq \text{sign}(f(b))$$
>
> then $\exists \xi \in (a,b)$ satisfying $f(\xi) = 0$.

The basic idea of the **bisection method** is to,

1. Assume that $\text{sign}(f(a)) \neq \text{sign}(f(b))$ on the interval $[a,b]$
2. Define a variable for the midpoint of the interval, $m = a + b/2$
3. If $\text{sign}(f(a)) \neq \text{sign}(f(m))$, consider the new interval $[a,m]$
4. Otherwise, consider the interval $[m,b]$
5. Repeat the previous steps until the interval length is small.

> **Remark.** The bisection method will return a root, if one exists, but it will not specify which root it has returned.

We want to know **how quickly the bisection method converges**.

> **Theorem 6.** The number of iterations k to reach an accuracy of ϵ is,
>
> $$\frac{\log(b-a) - \log\epsilon}{\log 2} \leq k$$
>
> using the bisection method.

Proof. We will use the following notation,

1. a_k and b_k are the left and right endpoints of the k-th interval
2. x_k is the midpoint of the interval $[a_k, b_k]$

Algorithm 1: The Bisection Method

```
function bisection(a, b, tol)
    // Assume f is defined
    // and that sign(f(a)) ≠ sign(f(b))
    while b − a > tol do
        m ← (a + b)/2
        if sign(f(a)) ≠ sign(f(b)) then
            b ← m
        else
            a ← m
    return m
```

Observe that,

$$|x^* - x_k| \leq b_k - a_k \leq \frac{1}{2}\left(b_{k-1} - a_{k-1}\right)$$

By iterating the recurrence, we obtain that,

$$|x^* - x_k| \leq \left(\frac{1}{2}\right)^k (b_0 - a_0) = \underbrace{\frac{b-a}{2^k}}_{\mathbf{a}_k}$$

Since $\{\mathbf{a}_k\}$ converges Q-linearly to zero, the bisection method converges R-linearly. Using the monotonicity of the logarithm function,

$$\frac{b-a}{2^k} \leq \epsilon \iff \frac{b-a}{\epsilon} \leq 2^k$$

implies that,

$$\frac{\log\left(\frac{b-a}{\epsilon}\right)}{\log 2} \leq k \iff \frac{\log(b-a) - \log\epsilon}{\log 2} \leq k$$

□

Fixed Point Iteration

> **Definition** (Fixed Point). A point $x^* \in \mathbb{R}$ is a **fixed point** for a function $g : \mathbb{R} \to \mathbb{R}$ if the condition $g(x^*) = x^*$ holds.

Fixed points and roots of functions can be related as follows.

1. If x^* is a root of f, then x^* is a fixed point of $g(x) := x + f(x)$.
2. If x^* is a fixed point of g, then x^* is a root of $f(x) := g(x) - x$

Note that there are many other choices for relating $g(x)$ and $f(x)$.

Theorem 7 (Fixed Point Theorem)**.** Suppose $g \in C([a,b])$.

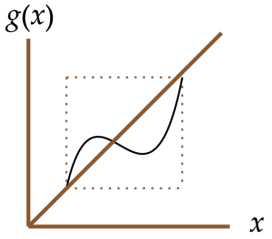
1. If $a \leq g(x) \leq b$ for all $x \in [a,b]$, then g has a fixed point,
$$x^* \in [a,b]$$
2. If g is differentiable on (a,b) and $\exists L \in (0,1)$ so that,
$$|g'(x)| \leq L$$
for all $x \in (a,b)$, then there is at most one fixed point in $[a,b]$

Proof. We will prove each claim in turn.

1. **(Existence)** Suppose $g(a) = a$ or $g(b) = b$. Then a or b is a fixed point of g. Otherwise, $a < g(a) < b$ since $a \leq g(x) \leq b$ by assumption. Define a function $f(x) := x - g(x)$. We know that $f(a) = a - g(a) < 0$ and $f(b) = b - g(b) > 0$. By the Intermediate Value Theorem, there exists $\xi \in (a,b)$ so that $f(\xi) = 0$. That is, $\xi = g(\xi)$ is a fixed point of g.
2. **(Uniqueness)** Suppose g is differentiable on (a,b) and $\exists L \in (0,1)$ so that $|g'(x)| \leq L$. Assume that $x^*, y^* \in [a,b]$ are fixed points. By the Mean Value Theorem, there exists $\xi \in (x^*, y^*)$ such that,
$$|g'(\xi)| = \left|\frac{g(y^*) - g(x^*)}{y^* - x^*}\right| = \left|\frac{y^* - x^*}{y^* - x^*}\right| = 1$$
This contradicts that $|g'(x)| \leq L < 1$ for all $x \in (a,b)$.

□

Visualization of the existence criteria for the fixed point theorem.



Definition (Fixed Point Iteration)**.** Let $x_0 \in [a,b]$. The **fixed point iteration sequence** is the sequence defined inductively by,

$$\begin{aligned} x_1 &= g(x_0) \\ x_2 &= g(x_1) = g(g(x_0)) \\ &\vdots \\ x_k &= g(x_{k-1}) = \underbrace{g(\ldots(g(x_0))\ldots)}_{k \text{ times}} \end{aligned}$$

We will establish conditions for the convergence of the fixed point iteration sequence in the theorem that follows.

Theorem 8 (Fixed Point Iteration Theorem)**.** Suppose g satisfies the hypothesis of the Fixed Point Theorem with a unique fixed point x^*.

1. $|x_k - x^*| \leq L^k \cdot |x_0 - x^*|$
2. $|x_k - x^*| \leq L^k \cdot \max\{|x_0 - a|, |x_0 - b|\}$

for $0 \leq L < 1$ and $\{x_k\}$ defined as the fixed point iteration sequence. Observe that (1) states that x_k converges R-linearly to x^*.

Proof. To prove (1),

$$|x_k - x^*| = |g(x_{k-1}) - g(x^*)|$$

by definition of $\{x_k\}$ and the fact that $x^* = g(x^*)$. By the Mean Value Theorem, there exists $\xi \in (x_{k-1}, x^*)$ such that,

$$\begin{aligned} |x_k - x^*| &= |g(x_{k-1}) - g(x^*)| \\ &= |g'(\xi_{k-1})| \, |x_{k-1} - x^*| \\ &\leq L|x_{k-1} - x^*| \end{aligned}$$

Iterating the recurrence gives that,

$$|x_k - x^*| \leq L^k |x_0 - x^*| \to 0 \text{ as } k \to \infty$$

To prove (2), we use that,

$$|x_0 - x^*| \leq \max\{|x_0 - a|, |x_0 - b|\}$$

because x_0 and x^* are in the interval $[a, b]$. □

Algorithm 2: Fixed Point Iteration

```
function FPI(a, b, tol)
    // Assume g is defined and that it satisfies the
       fixed point theorem hypothesis
    x ← x_0
    x' ← g(x)
    while |x' − x| > tol do
        x ← x'
        x' ← g(x)
    return x'
```

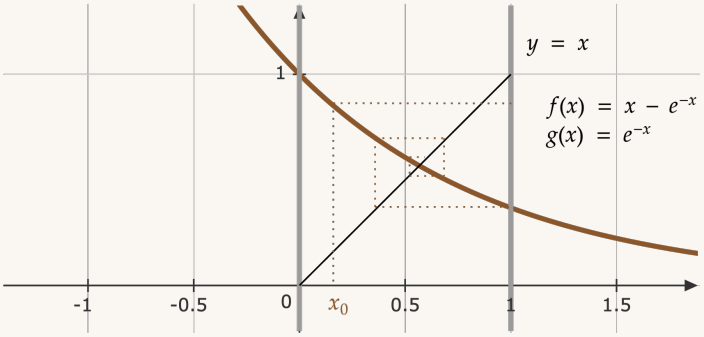
We could have used,

$$\frac{|x_k - x_{k-1}|}{|x_k|} < \texttt{tol}$$

$$|x_k - g(x_k)| < \texttt{tol}$$

as our stopping criteria.

Example 11: Finding the root of $f(x) = x - e^{-x}$



We want to know **how quickly the fixed point method converges**.

Theorem 9. Under the following conditions,

1. g satisfies the hypothesis of the Fixed Point Theorem
2. $g' \in C([a,b])$ with $g'(x^*) \neq 0$

Then, $\{x_k\}_k$ converges Q-linearly with A.E.C $|g'(x^*)|$.

Proof. Since $x^* = g(x^*)$ and $x_{k+1} = g(x_k)$, then by the Mean Value Theorem $\exists \xi_k \in (x_k, x^*)$ such that,

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

by the Mean Value Theorem. Taking $k \to \infty$, we obtain $x_k \to x^*$ and $\xi_k \to x^*$. By the continuity of g', we conclude that,

$$\lim_{k\to\infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|} = \lim_{k\to\infty} |g'(\xi_k)| = |g'(x^*)|$$

with $0 < |g'(x^*)| < 1$ implying Q-linear convergence. □

The continuity of g' justifies us in bringing the limit inside.

Remark. If $g'(x^*) = 0$, then the fixed point iteration might converge faster than linearly because a smaller A.E.C. gives faster convergence.

Theorem 10. Suppose $g \in C^p([a,b])$ has a fixed point x^*. If the fixed point iteration of g converges, $g^{(i)}(x^*) = 0$ for all $i \in [p-1]$, and $g^{(p)}(x^*) \neq 0$, then the fixed point iteration converges at order p.

Proof. This is left as an exercise on Assignment 1. □

Newton's Method

Previously, we saw that,

1. $g'(x^*) \neq 0 \implies$ fixed point iteration converges linearly
2. $g'(x^*) = 0 \implies$ fixed point iteration may converge at higher order

If $f'(x^*) = 0$, then Newton's Method is undefined or converges linearly.

The goal of **Newton's Method** is to construct a fixed point function g to find a root of f at quadratic order. To do this, we define,

$$g(x) = x + \phi(x) f(x)$$

for some unknown function $\phi(x)$. This ensures that,

$$f(x^*) = 0 \implies g(x^*) = x^*$$

We require that $g'(x^*) = 0$ whenever $f(x^*) = 0$ for the fixed point iteration to be of quadratic order. This is equivalent to,

$$0 = g'(x^*) = 1 + \phi'(x^*) \underbrace{f(x^*)}_{=0} + \phi(x^*) f'(x^*)$$

which suggests that ϕ is of the form,

$$\phi(x^*) = -\frac{1}{f'(x^*)}$$

with the fixed point function,

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

Therefore, we define the fixed point iteration,

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

given some x_0 chosen in advance.

Example 12: Recovering the Babylonian Sequence

Given $a > 0$, we can compute $\sqrt{a}$ using Newton's method and

$$f(x) = x^2 - a$$
$$f'(x) = 2x$$

We will iterate the recurrence,

$$x_{k+1} = x_k - \frac{x_k^2 - a}{2x_k}$$

to recover the Babylonian sequence.

Newton's Method is not guaranteed to converge for every choice of $x_0 \in [a, b]$.

The following theorem establishes that Newton's Method will converge if the initial choice x_0 is sufficiently close to x^*.

> **Theorem 11** (Local Convergence Theorem)**.** Let $f \in C^2[a, b]$ and $x^* \in [a, b]$ be a root of f with $f'(x^*) \neq 0$. There exists $\delta > 0$ such that Newton's Method converges quadratically $\forall x_0 \in [x^* - \delta, x^* + \delta]$.

We can **generalize Newton's method** to functions in n variables,

$$\begin{aligned} x_k &\to \mathbf{x}_k \\ f(x_k) &\to f(\mathbf{x}_k) \\ f'(x_k) &\to J_f(\mathbf{x}_k) \end{aligned}$$

where J_f is the **Jacobian matrix**. Recall that,

$$J_{\mathbf{f}}(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \dots & \frac{\partial f_1}{\partial x_n}(\mathbf{x}) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(\mathbf{x}) & \dots & \frac{\partial f_n}{\partial x_n}(\mathbf{x}) \end{pmatrix} \quad \text{for} \quad f(x) = \begin{pmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_n(\mathbf{x}) \end{pmatrix}$$

We select $\mathbf{x}_0$ in advance, and then we iterate as follows,

$$J_{\mathbf{f}}(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k) = -\mathbf{f}(\mathbf{x}_k)$$

In practice, we can first solve for $\mathbf{s}_k := \mathbf{x}_{k+1} - \mathbf{x}_k$ in,

$$J_{\mathbf{f}}(\mathbf{x}_k)\,\mathbf{s}_k = -\mathbf{f}(\mathbf{x}_k)$$

and then update $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$.

Newton's method can be computationally intensive in higher dimensions because we must compute $O(n^2)$ derivatives per iteration.

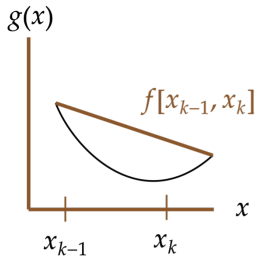
Secant Method

One disadvantage of Newton's Method is that evaluating $f'(x)$ can be costly or difficult. This motivates the **secant method**, which instead approximates f' using a **secant line** between $f(x_k)$ and $f(x_{k-1})$.

$$f[x_{k-1}, x_k] = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

The basic idea is to pick x_0 and x_1 and then iteratively define,

$$x_{k+1} = x_k - \frac{f(x_k)}{f[x_{k-1}, x_k]}$$



In order to converge, the Secant Method requires two initial guesses x_0 and x_1 which are sufficiently close to x^*.

Definition (Lagrange Interpolation Error). Let $f \in C^2[a,b]$ and $z_0, z_1 \in [a,b]$. There exists $\xi(x)$ between z_0, z_1 such that,

$$f(x) = \underbrace{f(z_0) + f[z_1, z_0](x - z_0)}_{f_1(x)} + \underbrace{\frac{f''(\xi(x))}{2!}(x - z_0)(x - z_1)}_{R_1(x)}$$

for all $x \in [a,b]$.

Remark. The Lagrange Interpolation Error is analogous to Taylor's Theorem with the derivative replaced by the secant. $f_1(x)$ represents $P_1(x)$ and $R_1(x)$ represents the remainder term.

Theorem 12. The secant method converges superlinearly, but slower than quadratic. Its convergence order is equal to the **golden ratio**.

Proof. We can re-write the secant method at the k-th iteration as,

$$0 = f(x_k) + f[x_{k-1}, x_k](x_{k+1} - x_k)$$

By the Mean Value Theorem, there exists η_k such that,

$$0 = f(x_k) + f'(\eta_k)(x_{k+1} - x_k)$$

By definition of the Lagrange Interpolation Error with,

$$z_0 = x_k \qquad z_1 = x_{k-1} \qquad x = x^*$$

we obtain the following,

$$0 = f(x_k) + f[x_{k-1}, x_k](x^* - x_k) + \frac{f''(\xi(x^*))}{2}(x^* - x_k)(x^* - x_{k-1})$$

Subtracting the two equations gives that,

$$x^* - x_{k+1} = A_k(x^* - x_k)(x^* - x_{k-1}) \text{ where } A_k = \frac{f''(\xi(x^*))}{2f'(\eta_k)}$$

If a sequence $\{x_k\}_k$ converges to x^* at order p, then for large k there exist constants $C_1, C_2 > 0$ and $r \in (0,1)$ such that,

$$C_1 r^{p^k} \leq |x^* - x_k| \leq C_2 r^{p^k}$$

Consequently,

$$\begin{aligned} C_1 r^{p^{k+1}} &\leq |x^* - x_{k+1}| \\ &= |A_k| \, |x^* - x_k| \, |x^* - x_{k-1}| \\ &\leq |A_k| \left(C_2 r^{p^k} \right) \left(C_2 r^{p^{k-1}} \right) \end{aligned}$$

Moreover,

$$\begin{aligned} |A_k| \left(C_1 r^{p^k} \right) \left(C_1 r^{p^{k-1}} \right) &\leq |A_k| \, |x^* - x_k| \, |x^* - x_{k-1}| \\ &= |x^* - x_{k+1}| \leq C_2 r^{p^{k+1}} \end{aligned}$$

Re-arranging gives that,

$$0 < \frac{C_1}{|A_k| \, C_2^2} \leq r^{p^k + p^{k-1} - p^{k+1}} \leq \frac{C_2}{|A_k| \, C_1^2} \text{ where } |A_k| \to \frac{|f''(x^*)|}{2 \, |f'(x^*)|}$$

since $r \in (0, 1)$, these inequalities require that,

$$0 = p^k + p^{k-1} - p^{k+1} = p^{k-1} \left(p + 1 - p^2 \right)$$

Solving gives the golden ratio,

$$p = \frac{1 + \sqrt{5}}{2} \approx 1.61803 \ldots$$

□

Summary

We saw the following numerical methods,

Method	**Convergence Criteria**	**Order**
Bisection	$f \in C[a, b]$ and $x_0 \in [a, b]$	Linear
Fixed Point	- Fixed Point Theorem (1) - Fixed Point Theorem (2) - $g'(x^*) \neq 0$	Linear
Newton	- $f'(x^*) \neq 0$ - $f''(x^*) \neq 0$ - x_0 close to x^*	Quadratic
Secant	- $f'(x^*) \neq 0$ - $f''(x^*) \neq 0$ - x_0 close to x^*	Golden Ratio

Topic 3. Interpolation and Polynomial Approximation

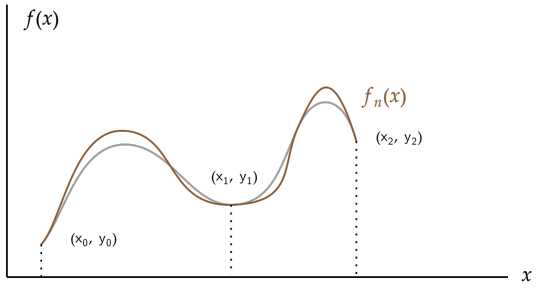
Defining Interpolation Problems

Given a set of data points $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=0}^n \subseteq \mathbb{R}^m \times \mathbb{R}^m$, we want to find a function $\mathbf{f}_n : \mathbb{R}^m \to \mathbb{R}^m$ so that $\mathbf{f}_n(\mathbf{x}_i) = \mathbf{y}_i$ for $0 \leq i \leq m$. For simplicity, we will first consider the case where $m = 1$.

In contrast to curve fitting, these equations are not satisfied exactly. They are satisfied in the least square sense.

> **Definition** (Interpolant). Given a sequence of points $\{y_i\}_{i=0}^n$ sampled from an unknown function f, we call f_n the **interpolant** of f.

Our objective is to obtain the interpolant f_n of f and to evaluate it on data points that we have not yet observed. For instance,



There are infinitely many ways to select the interpolant, so we will focus on choices of f_n which have desirable approximation properties, e.g.,

1. Polynomials
2. Piece-wise Polynomials
3. Trigonometric Functions

Since linear equations are easier to solve than non-linear ones, we will choose our interpolant to be a span by a set of functions,

$$f_n(x) = \sum_{i=1}^{N} c_j \cdot \phi_j(x)$$

where $\{\phi_j(x)\}_{j=0}^N$ is a set of linearly independent **basis functions**.

> **Definition** (Linear Independence). A set of functions $\{\phi_j(x)\}_{j=0}^N$ is called **linearly independent** on an interval $[a, b]$ if,
>
> $$\sum_{i=1}^{N} c_j \cdot \phi_j(x) = 0 \text{ on } [a, b] \quad \Longrightarrow \quad c_j = 0 \quad \forall j \in [N]$$
>
> Otherwise, we call the set of functions **linearly dependent**.

There are **three cases** to consider,

1. If $N > n$, then f_n exists but is not unique
2. If $N < n$, then f_n does not exist

3. If $N = n$, then f_n exists and is unique

Suppose that we are given a set of data points $\{(x_i, y_i)\}_{i=0}^{n}$ on an interval $[a, b]$ of real numbers. With $\{\phi_j(x)\}_{j=0}^{N}$ as a set of basis functions on $[a, b]$, observe that the coefficients c_j satisfy that,

$$y_i = f_n(x_i) = \sum_{i=1}^{N} c_j \cdot \phi_j(x)$$

which we can write in matrix form as follows,

$$\underbrace{\begin{pmatrix} \phi_0(x_0) & \phi_1(x_0) & \dots & \phi_n(x_0) \\ \phi_0(x_1) & \phi_1(x_1) & \dots & \phi_n(x_1) \\ \vdots & \vdots & & \vdots \\ \phi_0(x_n) & \phi_1(x_n) & \dots & \phi_n(x_n) \end{pmatrix}}_{:=\mathbf{A}} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

Remark. $\mathbf{A}$ is invertible since $\{\phi_i(x)\}_{i=0}^{N}$ are basis functions, i.e., the columns of $\mathbf{A}$ are linearly independent.

Definition (Condition Number)**.** If $\mathbf{A}$ is invertible, then the **condition number** of $\mathbf{A}$ with respect to the matrix norm $\|\cdot\|$ is,

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

Otherwise $\kappa(\mathbf{A}) := \infty$.

Remark. For invertible $\mathbf{A}$, the condition number $\kappa(\mathbf{A})$ satisfies that,

1. If $\kappa(\mathbf{A}) = \kappa(\mathbf{A}^{-1})$, then $1 \leq \kappa(\mathbf{A}) < \infty$ since

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

2. $\kappa(c\mathbf{A}) = \kappa(\mathbf{A})$ for any constant $c \neq 0$

$\mathbf{A}$ is called **well-conditioned** if $\kappa(\mathbf{A})$ is close to 1 since small changes in $\mathbf{b}$ imply small changes in $\mathbf{x}$. Otherwise, $\mathbf{A}$ is called **ill-conditioned** and small changes in $\mathbf{b}$ may result in large changes in $\mathbf{x}$.

The norm of a matrix $\mathbf{A}$ measures how much the mapping induced by that matrix can stretch vectors.

A matrix $\mathbf{A}$ is singular if and only if its determinant $\det(\mathbf{A})$ is 0.

Example 13: Ill-Conditioned Matrices

The following matrix is **ill-conditioned** in the ℓ_∞ norm.

$$\mathbf{A} = \begin{pmatrix} 0.123 & 0.456 \\ 0.789 & 2.92507 \end{pmatrix}$$

$$\mathbf{A}^{-1} = -2.5641 \times 10^6 \begin{pmatrix} 2.92507 & -0.456 \\ -0.789 & 0.123 \end{pmatrix}$$

We will compute $\|\mathbf{A}\|_\infty$ and $\|\mathbf{A}^{-1}\|_\infty$.

$$\begin{aligned} \|\mathbf{A}\|_\infty &= \max\{0.123 + 0.456, 0.789 + 2.92507\} \\ &= \max\{0.579, 3.71407\} = 3.71407 \\ \|\mathbf{A}^{-1}\|_\infty &= 2.5641 \times 10^6 \cdot \max\{2.92507 + 0.456, 0.789 + 0.123\} \\ &= 2.5641 \times 10^6 \cdot \max\{3.3811, 0.9120\} = 8.6695 \times 10^6 \end{aligned}$$

It follows that $\kappa(\mathbf{A}) = \|\mathbf{A}\|_\infty \cdot \|\mathbf{A}^{-1}\|_\infty = 3.2199 \times 10^7$

The value of $\kappa(\mathbf{A})$ depends on the matrix norm, but these are all technically related due to norm equivalences in $\mathbb{R}^n$.

$$\|\mathbf{A}\|_p = \sup_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p}$$

$$\|\mathbf{A}\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{m} |a_{ij}|$$

$$\|\mathbf{A}\|_\infty = \max_{1 \leq i \leq m} \sum_{j=1}^{n} |a_{ij}|$$

If $\mathbf{A}$ and $\mathbf{B}$ are matrices and x is a vector, then the following hold,

$$\|\mathbf{AB}\| \leq \|\mathbf{A}\|\|\mathbf{B}\|$$
$$\|\mathbf{A}x\| \leq \|\mathbf{A}\|\|x\|$$

Monomial Basis Functions

Consider a set of data points $\{(x_i, y_i)\}_{i=0}^n$. We want the interpolant to be a degree n polynomial $f_n(x) = \sum_{j=0}^n c_j x^j$, so the first natural choice of basis functions is the **monomial basis** $\phi_j(x) = x^j$ for $j \in [n]$. The resulting linear system for interpolation is then,

$$\underbrace{\begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^n \end{pmatrix}}_{:=\mathbf{V}} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

which is called the **Vandermonde matrix**.

> **Remark.** In general, $\mathbf{V}$ is **ill-conditioned** since,
>
> $$\det(\mathbf{V}) = \prod_{i \neq j} (x_i - x_j) \approx 0 \text{ if any } x_i \approx x$$
>
> which implies $\kappa(\mathbf{A}) \to \infty$. The proof is by induction, using the fact that $\det \mathbf{A} = a_{11} \cdot \det(\mathbf{A}_{11}) + a_{12} \cdot \det(\mathbf{A}_{12}) + \ldots + a_{1n} \cdot \det(\mathbf{A}_{1n})$

This motivates our study of the **Lagrange basis functions**.

The **Kronecker's delta** δ_{ij} is defined,

$$\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

Lagrange Basis Functions

Consider a set of data points $\{(x_i, y_i)\}_{i=0}^n$. The **Lagrange basis functions** $\{\ell_j(x)\}_{j=0}^n$ are the n-th order polynomials satisfying,

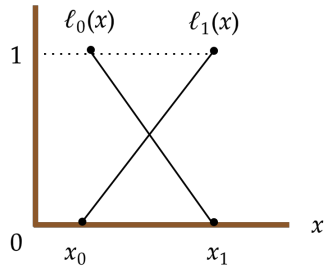
$$\ell_j(x_i) = \delta_{ji} = \begin{cases} 0, \text{ if } j \neq i \\ 1, \text{ if } j = i \end{cases}$$

with the corresponding linear system,

$$\underbrace{\begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix}}_{\mathbf{A}} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix} \Rightarrow c_i = y_i$$

for interpolation. The matrix $\mathbf{A}$ is **well-conditioned** with $\kappa(\mathbf{A}) = 1$. We will see two types of Lagrange basis functions,

1. Linear Lagrange Polynomials
2. Quadratic Lagrange Polynomials

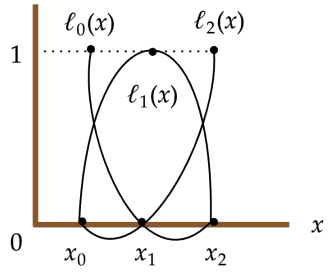


Suppose that $n = 1$. We will define the **linear Lagrange polynomial**. Given $\{x_0, x_1\}$, we want to find linear polynomials $\ell_0(x)$ and $\ell_1(x)$ so that $\ell_j(x_i) = \delta_{ji}$. Without loss of generality, let $\ell_0(x) = c(x - x_1)$ so that $\ell_0(x_1) = 0$. Thus, $1 = \ell_0(x_0) = c(x_0 - x_1)$ and,

$$\ell_0(x) = \frac{x - x_1}{x_0 - x_1}$$
$$\ell_1(x) = \frac{x - x_0}{x_1 - x_0}$$

Suppose that $n = 2$. We will define the **quadratic Lagrange polynomial**. Given $\{x_0, x_1, x_2\}$, we want to find $\ell_0(x)$, $\ell_1(x)$, and $\ell_2(x)$ so that $\ell_i(x_i) = \delta_{ii}$. As in the linear case, we set $\ell_0(x) = c(x - x_1)(x - x_2)$ to obtain that $\ell_0(x_1) = 0 = \ell_0(x_2)$. This gives $1 = \ell_0(x_0) = c(x_0 - x_1)(x_0 - x_2)$ and therefore,

$$\ell_0(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)}$$
$$\ell_1(x) = \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)}$$
$$\ell_2(x) = \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}$$



Theorem 13 (Lagrange Basis Formulas)**.** Given a sequence $\{x_i\}_{i=0}^n$, the j-th **Lagrange basis function** is the n-th degree polynomial,

$$\ell_j(x) = \frac{(x-x_0)\cdots(x-x_{j-1})\,(x-x_{j+1})\cdots(x-x_n)}{(x_j-x_0)\cdots(x_j-x_{j-1})\,(x_j-x_{j+1})\cdots(x_j-x_n)}$$

which gives the interpolant,

$$f_n = \sum_{j=0}^{n} y_j \ell_j(x) = \sum_{j=0}^{n} y_j \cdot \prod_{\substack{0\le k\le n \\ k\neq j}} \frac{x-x_k}{x_j-x_k}$$

Proof. Check that $\ell_i(x_i) = \delta_{ii}$. □

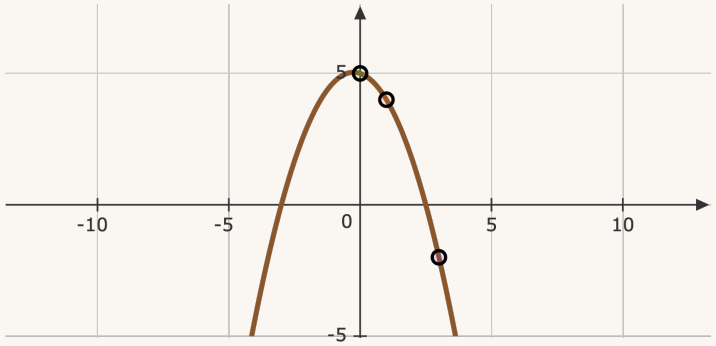
Example 14: Lagrange Interpolation

We want to find the Lagrange polynomial for $\{(0,5),(1,4),(3,-2)\}$. Let $x_0 = 0$, $x_1 = 1$, and $x_2 = 3$. Then,

$$f_2(x) = y_0\ell_0(x) + y_1\ell_1(x) + y_2\ell_2(x) = \frac{1}{3}\left(-2x^2 - x + 15\right)$$

Observe that f_n interpolates on these three points,

$$f_2(0) = \frac{15}{3} = 5$$
$$f_2(1) = \frac{-2-1+15}{3} = \frac{12}{3} = 4$$
$$f_2(3) = \frac{-18-3+15}{3} = \frac{-6}{3} = -2$$



Newton Basis Functions

Consider a set of data points $\{(x_i, y_i)\}_{i=0}^n$. The **Newton basis functions** $\{n_j(x)\}_{j=0}^n$ are the j-th order polynomials defined as follows,

$$n_j(x) = \prod_{k=0}^{j-1} (x - x_k)$$

The resulting linear system for interpolation is,

$$\begin{pmatrix} 1 & 0 & \dots & 0 \\ 1 & x_1 - x_0 & & 0 \\ 1 & x_2 - x_0 & & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n - x_0 & \dots & \prod_{k=0}^{n-1} (x_n - x_k) \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

and, as with Lagrange interpolation, it is **well-conditioned**.

> **Remark.** This matrix is lower triangular, so we can use **forward substitution** to find the coefficients c_j.
>
> $$c_0 = y_0 =: f[x_0]$$
> $$c_1 = \frac{y_1 - c_0}{x_1 - x_0} = \frac{y_1 - y_0}{x_1 - x_0} = \frac{f[x_1] - f[x_0]}{x_1 - x_0} =: f[x_0, x_1]$$
> $$c_2 = \frac{1}{x_2 - x_1} \left(f[x_0, x_2] - f[x_0, x_1] \right) =: f[x_1, x_0, x_2] = f[x_0, x_1, x_2]$$

Definition (Newton's Divided Differences). Given a set of data points $\{(x_i, y_i)\}_{i=0}^n$, the k-th order **divided difference** of f is,

$$\begin{aligned} k = 0: &\quad f[x_i] := y_i \\ k > 0: &\quad f[x_i, \dots, x_{i+k}] := \frac{f[x_{i+1}, \dots, x_{i+k}] - f[x_i, \dots, x_{i+k-1}]}{x_{i+k} - x_i} \end{aligned}$$

Example 15: Newton's Divided Differences

Fix x_0, x_1, and x_2. Then,

$$f[x_0, x_1] = \frac{y_1 - y_0}{x_1 - x_0}$$
$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} = \frac{\frac{y_2 - y_1}{x_2 - x_1} - \frac{y_1 - y_0}{x_1 - x_0}}{x_2 - x_0}$$

> **Remark.** By induction on forward substitution, it can shown that the coefficients c_j in Newton basis are just $f[x_0, \dots, x_j]$

It can be shown that $f[x_{\sigma(1)}, x_{\sigma(2)}, \dots, x_{\sigma(n)}]$ is equal to $f[x_0, x_1, \dots, x_n]$ for any permutation $\sigma : \{0, \dots, n\} \to \{0, \dots n\}$. Thus, we can always reorder the arguments with increasing indices.

Theorem 14 (Newton's Divided Difference Interpolation Formula)**.** Given a set of data points $\{(x_i, y_i)\}_{i=0}^{n}$, the polynomial interpolation can be expressed using the Newton basis as,

$$\begin{aligned} f_n(x) &= f[x_0] + \sum_{k=1}^{n} f[x_0, \ldots, x_k] \prod_{i=0}^{k-1} (x - x_i) \\ &= f_{n-1}(x) + f[x_0, \ldots, x_n] \prod_{i=0}^{n-1} (x - x_i) \end{aligned}$$

The second statement of Newton's Divided Difference Interpolation Formula can be useful if additional data points are added incrementally.

Example 16: Revisiting our Lagrange Interpolation Example

Given $\{(0,5),(1,4),(3,-2)\}$, we found that

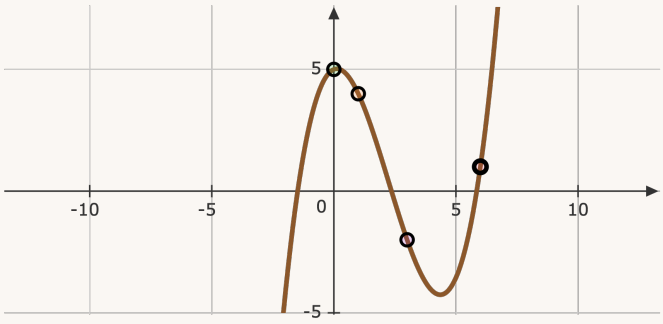
$$f_2(x) = \frac{1}{3}\left(-2x^2 - x + 15\right)$$

If we add one data point $(6,1)$, then $f_3(x)$ can be easily computed using Newton's Divided Difference Interpolation Formula. We need only compute $f[0,1,3,6]$:

$$\begin{aligned} f_3(x) &= f_2(x) + f[0,1,3,6](x-0)(x-1)(x-3) \\ &= \frac{11}{45}x^3 - \frac{74}{45}x^2 + \frac{18}{45}x + 5 \end{aligned}$$

To do this, we will maintain a table:

x_i	$f[x_i]$	$f[x_i, x_{i+1}]$	$f[x_i, x_{i+1}, x_{i+2}]$	$f[x_i, x_{i+1}, x_{i+2}, x_{i+3}]$
0	5	$\frac{4-5}{1-0} = -1$		
			$\frac{-3-(-1)}{3-0} = -\frac{2}{3}$	
1	4			$\frac{\left(\frac{4}{5}\right)-\left(-\frac{2}{3}\right)}{6-0} = \frac{11}{45}$
		$\frac{-2-4}{3-1} = -3$		
3	−2		$\frac{1-(-3)}{6-1} = \frac{4}{5}$	
		$\frac{1-(-2)}{6-3} = 1$		
6	1			



Lagrange and Newton basis functions are preferred because they are well-conditioned, but all three interpolation methods give the same polynomial satisfying $f_n(x_i) = y_i$ for all $0 \le i \le n$.

Error Analysis

We want to **quantify how well** $f_n(x)$ approximates $f(x)$ on the interval $[a,b]$. This error depends on the choice of $\{x_0, x_1, \cdots, x_n\}$, as well as the derivatives of f, and it is captured by the following theorem.

Lemma 2 (Generalized Rolle's Theorem)**.** If $g \in C^{n+1}([a,b])$ and $g = 0$ on $n+2$ distinct points $\{x_0, \ldots, x_n, x_{n+1}\}$ in $[a,b]$, then $\exists \xi$ between two points in $\{x_0, \ldots, x_n, x_{n+1}\}$ so that $g^{(n+1)}(\xi) = 0$.

Theorem 15 (Lagrange Interpolation Error)**.** Let $f \in C^{n+1}([a,b])$. Fix $x_1, \cdots x_n \in [a,b]$. For all $x \in [a,b]$, $\exists \xi(x) \in (a,b)$ so that,

$$f(x) = f_n(x) + \underbrace{\frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x-x_0)\cdots(x-x_n)}_{\texttt{Error}}$$

Proof. Let $x \in [a,b]$. If $x = x_i$ for all $i \in [n]$, then $f(x_i) = f_n(x_i) + 0$ and the statement holds. Assume that this is not the case.

1. **(Step 1)** Construct a function g with $n+2$ distinct roots using f and f_n. For fixed $x \in [a,b]$ and a dummy variable t,

$$g(t) := f(t) - f_n(t) - (f(x) - f_n(x)) \underbrace{\frac{(t-x_0)\cdots(t-x_n)}{(x-x_0)\cdots(x-x_n)}}_{\texttt{Denominator}\neq 0}$$

2. **(Step 2)** If $t = x_i$, then for any $i \in [n]$, we have that,

$$g(x_i) = f(x_i) - f_n(x_i) - (f(x) - f_n(x)) \times 0$$

   Moreover, if $t = x$, then,

$$g(x) = f(x) - f_n(x) - (f(x) - f_n(x)) \times 1$$

   That is, $g = 0$ on $n+2$ distinct points $\{x_0, \ldots, x_n, x\}$. By the generalized Rolle's Theorem, $g^{(n+1)}(\xi(x)) = 0$ for some $\xi(x)$.

3. **(Step 3)** Computing $g^{(n+1)}(t)$,

$$\begin{aligned} g^{(n+1)}(t) =& f^{(n+1)}(t) - f_n^{(n+1)}(t) \\ & - (f(x) - f_n(x)) \frac{d^{(n+1)}}{dt^{(n+1)}} \left(\frac{(t-x_0)\cdots(t-x_n)}{(x-x_0)\cdots(x-x_n)} \right) \end{aligned}$$

and simplifying at $t = \xi(x)$ gives,

$$\begin{aligned} 0 &= g^{(n+1)}(\xi(x)) \\ &= f^{(n+1)}(\xi(x)) - (f(x) - f_n(x)) \frac{(n+1)!}{(x-x_0)\cdots(x-x_n)} \end{aligned}$$

which can be re-arranged to obtain the result,

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

□

In general, the value of M_{n+1} depends on how fast f is changing on the interval $[a,b]$. The product terms of the upper bound will depend on how $\{x_0, \cdots, x_n\}$ are chosen on $[a,b]$.

Corollary. We can bound the error as follows,

$$\begin{aligned} |f(x) - f_n(x)| &= \left| \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x-x_0)\cdots(x-x_n) \right| \\ &\leq \frac{M_{n+1}}{(n+1)!} \max_{x\in[a,b]} |(x-x_0)\cdots(x-x_n)| \end{aligned}$$

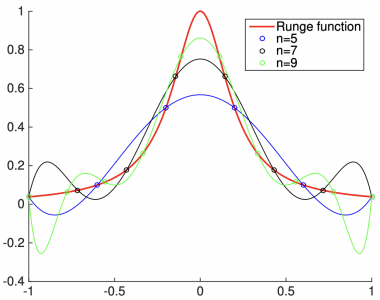
since $\left|f^{(n+1)}(\xi(x))\right| \leq \max_{x\in[a,b]} \left|f^{(n+1)}(x)\right| =: M_{n+1}$.

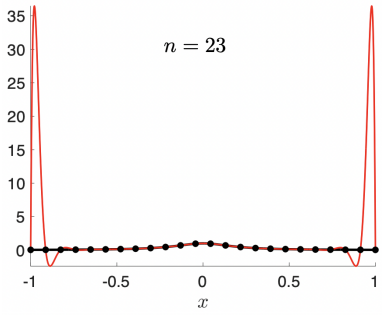
> **Remark.** If f is a polynomial of degree less than or equal to n, i.e., $f^{(n+1)}(x) = 0$, then the interpolation is exact.

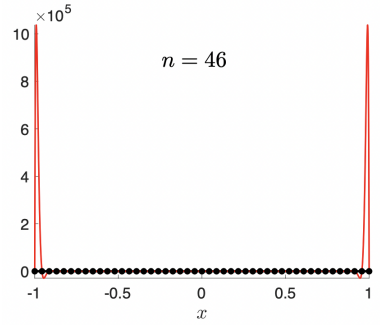
For a fixed function f, we want to pick $\{x_0, \cdots, x_n\}$ to minimize the error. A natural choice is to select points with a **uniform spacing** $h := \frac{b-a}{n}$. Specifically, we will define $x_i := a + ih$ for $i \in [n]$.

> **Theorem 16** (Error Bound for Uniformly-Spaced Points)**.**
>
> $$|f(x) - f_n(x)| \leq M_{n+1}\frac{h^{n+1}}{4(n+1)} = \frac{M_{n+1}(b-a)^{n+1}}{4n^{n+1}(n+1)}$$

Proof. The proof is left as an exercise on Assignment 2. □







Runge's Phenomenon

High degree interpolation does not always imply high accuracy. Consider the function $f(x) = 1/(1+25x^2)$ on $[-1,1]$ shown in the margin. There exist some $x \in (-1,1)$ such that $\lim_{n\to\infty} |f(z) - f_n(z)| = \infty$. We will formulate a minimization problem that allows us to interpolate on points with **non-uniform spacing**. We want to minimize,

$$\max_{x\in[a,b]} |\underbrace{(x-x_0)\cdots(x-x_n)}_{=x^{n+1}+a_nx^n+\cdots+a_0}|$$

since we have no control over M_{n+1}. Write,

$$\mathbb{P}_n = \{p(x) = a_n x^n + \cdots + a_0 : a_i \in \mathbb{R}\}$$

for the set of all polynomials of degree n with real coefficients, and,

$$\tilde{\mathbb{P}}_n = \{p(x) \in \mathbb{P}_n : a_n = 1\}$$

for the set of all **monic** polynomials of degree n. Let

$$(x - x_0) \cdots (x - x_n) \in \tilde{\mathbb{P}}_{n+1}$$

and consider the interval $[-1, 1]$, which we can transform to $[a, b]$ via

$$x := \frac{(b-a)t + (b+a)}{2} \in [a, b]$$

for $t \in [-1, 1]$. We want to find $\tilde{q}(t) \in \tilde{\mathbb{P}}_{n+1}$ so that

$$\max_{t\in[-1,1]} |\tilde{q}(t)| \leq \max_{t\in[-1,1]} |\tilde{p}(t)| \qquad (\star)$$

for all $\tilde{p}(t) \in \tilde{\mathbb{P}}_{n+1}$. The unique monic polynomial solving our minimization problem $(\star)$ is defined by the roots of the polynomial:

$(\star)$ is called the **optimal monic polynomial problem**, and the unique monic polynomial that solves it is the **Chebyshev polynomial**.

Definition (Chebyshev Polynomial). The **Chebyshev polynomial** T_n is the n-th degree polynomial defined on $[-1, 1]$ by,

$$T_n(t) = \cos(n \arccos(t))$$

Recall the cosine angle sum identities,

$$\sin(\alpha + \beta) = \sin\alpha\cos\beta + \cos\alpha\sin\beta$$
$$\cos(\alpha + \beta) = \cos\alpha\cos\beta - \sin\alpha\sin\beta$$
$$\sin(\alpha - \beta) = \sin\alpha\cos\beta - \cos\alpha\sin\beta$$
$$\cos(\alpha - \beta) = \cos\alpha\cos\beta + \sin\alpha\sin\beta$$

Remark. Computing T_n for $n = 0, 1, 2$,

$$T_0(t) = \cos(0) = 1$$
$$T_1(t) = \cos(\arccos(t)) = t$$
$$T_2(t) = \cos(2\underbrace{\arccos(t)}_{\theta}) = \cos(2\theta) = 2\cos^2\theta - 1 = 2t^2 - 1$$

where the result for T_n follows by trigonometric identities. With T_0 constant, this gives the following recurrence relation,

$$T_2(t) = 2tT_1(t) - T_0(t)$$

which, by induction, generalizes to,

$$\begin{aligned} T_{n+1}(t) &= 2tT_n(t) - T_{n-1}(t) \\ &\in 2t\left(2^{n-1}t^n + \mathbb{P}_{n-2}\right) + \mathbb{P}_{n-1} \subset 2^n t^{n+1} + \mathbb{P}_{n-1} \end{aligned}$$

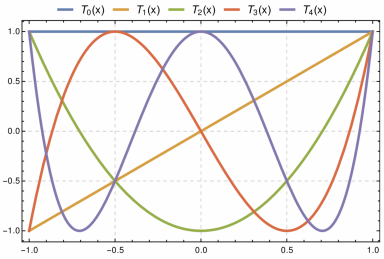
Remark. The zeroes of T_{n+1} are,

$$t_k = \cos\left(\frac{(2k+1)\pi}{2(n+1)}\right) \quad k = 0, 1, \cdots, n$$

The extremal points T_{n+1} are,

$$z_k = \cos\left(\frac{k\pi}{n+1}\right) \quad k = 0, 1, \cdots, n+1$$

with values $T_{n+1}(z_k) = (-1)^k$.



Theorem 17 (Chebyshev Equioscillation)**.** The polynomial,

$$\frac{T_{n+1}(t)}{2^n}$$

is the unique monic polynomial that solves our minimization problem $(\star)$. Specifically, for any $\tilde{p}_{n+1}(x) \in \tilde{\mathbb{P}}_{n+1}$,

$$\frac{1}{2^n} = \max_{t\in[-1,1]} \left|\frac{T_{n+1}(t)}{2^n}\right| \leq \max_{t\in[-1,1]} |\tilde{p}_{n+1}(t)|$$

Proof. Assume that there exists another monic polynomial,

$$\tilde{p}_{n+1}(x) \neq \frac{T_{n+1}(t)}{2n}$$

with a smaller maximum absolute value,

$$\max_{t\in[-1,1]} |\tilde{p}_{n+1}(t)| < \max_{t\in[-1,1]} \left|\frac{T_{n+1}(t)}{2^n}\right| = \frac{1}{2^n}$$

Define a polynomial $Q(t)$ as follows,

$$Q(t) := \frac{T_{n+1}(t)}{2^n} - \tilde{p}_{n+1}(t)$$

T_{n+1} and $\tilde{p}_{n+1}$ have $a_{n+1} = 1$, so $Q(t)$ is a polynomial of order n. Evaluating Q on all extremal points z_k of T_{n+1} for all $k \in [n+1]$,

$$Q(z_k) = \frac{(-1)^k}{2^n} - \tilde{p}_{n+1}(z_k) \begin{cases} \geq \frac{1}{2^n} - |\tilde{p}_{n+1}(z_k)| > 0, & \text{if } k \text{ even} \\ \leq -\frac{1}{2^n} + |\tilde{p}_{n+1}(z_k)| < 0, & \text{if } k \text{ odd} \end{cases}$$

By the Intermediate Value Theorem, it follows that $Q(t)$ has a zero between $[z_k, z_{k+1}]$ for $k = 0, \cdots, n$. This implies that $Q(t)$ has $n+1$ distinct zeroes, and consequently that $Q(t)$ must be

the zero polynomial (since $Q(t)$ is of degree n). Thus,

$$\tilde{p}_{n+1}(t) = \frac{T_{n+1}(t)}{2^n}$$

□

Returning to our previous example, we see that interpolating on,

$$t_k = \cos\left(\frac{(2k+1)\pi}{2(n+1)}\right)$$

produces a polynomial interpolant with no unbounded oscillations.

Spline Interpolation

There are **four** main drawbacks of polynomial interpretation.

1. On uniformly-spaced points (e.g., data collected over time intervals), higher-order interpolation leads to unbounded oscillations
2. Special data is needed to interpolate using the Chebyshev method
3. We have a lack of control and accuracy of the error,

$$|f(x) - f_n(x)| \leq C_n h^n$$

since C_n depends on $f^{(n)}$ on $[a, b]$ and may grow for large n

4. Changing one data point affects the interpolant globally

The solution to these drawbacks is to use **splines**. Formally,

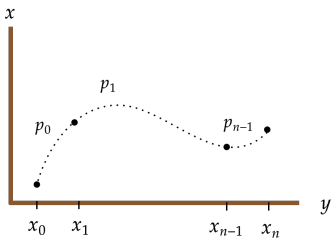
Definition (Spline). Consider a set of points $\{(x_i, y_i)\}_{i=0}^n$ with

$$a = x_0 < x_1 < \cdots < x_{n-1} < x_n = b$$

A piece-wise polynomial interpolant $S(x)$, called a **spline**, has

- $S(x) = p_i(x)$ on $[x_i, x_{i+1})$ for $0 \leq i \leq n-1$
- $p_i(x)$ is a polynomial of some fixed degree
- $S(x_i) = y_i$, that is, S interpolates the data

The points x_i are called **breakpoints** or **knots**



We will see three methods, each of which has a complexity in $\mathcal{O}(n)$, for spline interpolation. These are summarized in the table below,

Spline Interpolant	Locality	Error	Smoothness
Piecewise constant	Single sub-intervals	$\mathcal{O}(h)$	Bounded on $[a, b]$
Piecewise linear	Two sub-intervals	$\mathcal{O}(h^2)$	$C[a, b]$
Piecewise cubic	Neighboring sub-intervals	$\mathcal{O}(h^4)$	$C^2[a, b]$

Piece-wise constant splines are defined as follows,

1. $p_i(x) = a_i$ are polynomial of zeroth degree for $0 \leq i \leq n-1$
2. $S(x_n) = a_n$ so $n+1$ unknowns need to be determined

For each interval $[x_i, x_{i+1})$ where $0 \leq i \leq n-1$,

$$y_i = S(x_i) = p_i(x_i) = a_i \Rightarrow a_i = y_i$$

and the right-most endpoint is a_n. It follows that,

$$S(x) = \begin{cases} y_i, & x \in [x_i, x_{i+1}) \\ y_n, & x = x_n \end{cases}$$

> **Remark.** The **local interpolation error** is,
>
> $$|f(x) - S(x)| \leq \frac{M_1}{1!} \cdot \max_{x \in [x_{i-1}, x_i]} |x - x_{i-1}| = M_1 \cdot h_{i-1}$$
>
> since $S(x)$ is a polynomial of zeroth degree on each $[x_i, x_{i+1})$. Letting $h := \max_{1 \leq i \leq n} h_{i-1}$, the **global interpretation error** is,
>
> $$\max_{x \in [a,b]} |f(x) - S(x)| \leq \mathcal{O}(h)$$

If **continuity** is an important feature for our model, then we can use a **linear polynomial** on each interval. That is, $p_i(x) := a_i + b_i(x - x_i)$ on the interval $[x_i, x_{i+1})$ for each $0 \leq i \leq n-1$. We impose,

1. (Interpolation Condition) $y_i = S(x_i) = p_i(x_i) = a_i \Rightarrow a_i = y_i$ to ensure that our curve passes through our data points.
2. (Continuity Condition) For $1 \leq i \leq n-1$, we want $S\left(x_i^-\right) = S\left(x_i^+\right)$ at each x_i. Applying the definition of p_i, this implies that,

   $$y_{i-1} + b_{i-1}(x_i - x_{i-1}) = p_{i-1}(x_i) = S\left(x_i^-\right) = S\left(x_i^+\right) = p_i(x_i) = y_i$$

   which gives that,

   $$b_{i-1} = \frac{y_i - y_{i-1}}{x_i - x_{i-1}} = f[x_{i-1}, x_i]$$

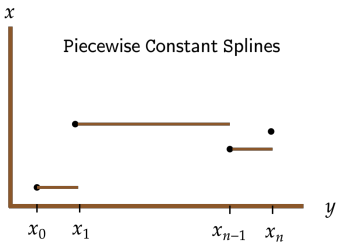
   and at the right-most end-point,

   $$y_n = S(x_n) = p_{n-1}(x_n) = y_{n-1} + b_{n-1}(x_n - x_{n-1})$$

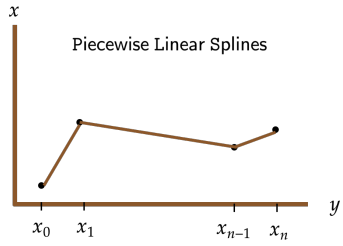
In summary, we have,

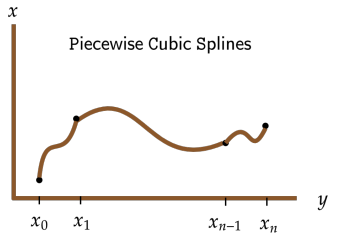
$$S(x) = \begin{cases} y_i + f[x_i, x_{i+1}](x - x_i), & x \in [x_i, x_{i+1}) \\ y_{n-1} + f[x_{n-1}, x_n](x - x_{n-1}), & x \in [x_{n-1}, x_n] \end{cases}$$

with the property that changes to y_i will only affect $S(x)$ on the intervals $[x_{i-1}, x_i)$ and $[x_i, x_{i+1})$. Our **error analysis** for piece-wise linear splines is similar to piece-wise constant splines.



> **Remark.** The **local interpolation error** is,
>
> $$|f(x) - S(x)| \leq \frac{M_2}{2!} \underbrace{\max_{x \in [x_{i-1}, x_i]} |(x - x_{i-1})(x - x_i)|}_{\text{maximized at midpoint}} = \frac{M_2}{8} h_{i-1}^2$$
>
> Letting $h := \max_{1 \leq i \leq n} h_{i-1}$, the **global interpolation error** is,
>
> $$\max_{x \in [a,b]} |f(x) - S(x)| \leq \mathcal{O}(h^2)$$





The drawback associated with piece-wise linear splines is that many applications require continuity in the first and second derivatives. This leads us to define **cubic piece-wise splines**. We will use a **cubic polynomial** on each interval. That is, $p_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$ for each $0 \leq i \leq n - 1$. We impose,

1. (Interpolation Condition) $y_i = S(x_i) = p_i(x_i) = a_i \Rightarrow a_i = y_i$ to ensure that the curve passes through our data points. We pick $y_n = S(x_n) = p_{n-1}(x_n)$ at the right-most end-point.
2. (Continuity Condition # A) For $1 \leq i \leq n - 1$, $S(x_i^-) = S(x_i^+)$
3. (Continuity Condition # B) For $1 \leq i \leq n - 1$, $S'(x_i^-) = S'(x_i^+)$
4. (Continuity Condition # C) For $1 \leq i \leq n - 1$, $S''(x_i^-) = S''(x_i^+)$

Conditions (2), (3), and (4) involve $n - 1$ equations, while (1) requires $n + 1$. In total, this is $4n - 2$ equations with $4n$ unknowns. We require two more conditions to determine an interpolate uniquely:

1. (Free Boundary Condition) We assume that,

$$f''(x_0) = 0 \quad \text{and} \quad f''(x_n) = 0$$

$$\implies S''(x_0) = 0 \text{ and } S''(x_n) = 0$$

The resultant $S(x)$ is called the **natural spline**

2. (Clamped Boundary Condition) We collect two additional data points z_0 and z_n to specify the first derivative at the endpoints,

$$S'(x_0) = z_0 \quad \text{and} \quad S'(x_n) = z_n$$

   The resultant $S(x)$ is called the **complete spline**

3. (Not-a-Knot Condition) We assume that,

$$p_0'''(x_1) = p_1'''(x_1) \text{ and } p_{n-2}'''(x_{n-1}) = p_{n-1}'''(x_{n-1})$$

   The resultant $S(x)$ is called the **not-a-knot spline**

The **not-a-knot** end condition means that, at the first and last interior break, even the third derivative is continuous (up to round-off error).

Given these different boundary conditions, we need to find the coefficients a_i, b_i, c_i, and d_i to construct our polynomial. We write,

$$\begin{aligned} p_i(x) &= a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 \\ p_i'(x) &= b_i + 2c_i(x - x_i) + 3d_i(x - x_i)^2 \\ p_i''(x) &= 2c_i + 6d_i(x - x_i) \end{aligned}$$

Recall that $h_i = x_{i+1} - x_i$ for $i \in [n-1]$ since our intervals are not uniformly spaced. The **interpolation condition** completely determines the constant terms a_i since for all n equations,

$$y_i = S(x_i) = p_i(x_i) = a_i \Rightarrow a_i = y_i$$

Plugging in h_i and using our **right end-point condition**,

$$\begin{aligned} &y_n = S(x_n) = p_{n-1}(x_n) = y_{n-1} + b_{n-1}h_{n-1} + c_{n-1}h_{n-1}^2 + d_{n-1}h_{n-1}^3 \\ &\Rightarrow f[x_{n-1}, x_n] = b_{n-1} + c_{n-1}h_{n-1} + d_{n-1}h_{n-1}^2 \end{aligned}$$

We will use **# A** to obtain,

$$\begin{aligned} f[x_{i-1}, x_i] &= b_{i-1} + c_{i-1}h_{i-1} + d_{i-1}h_{i-1}^2 \\ f[x_{n-1}, x_n] &= b_{n-1} + c_{n-1}h_{n-1} + d_{n-1}h_n^2 \end{aligned}$$

 # B to obtain,

$$\begin{aligned} b_i &= b_{i-1} + 2c_{i-1}h_{i-1} + 3d_{i-1}h_{i-1}^2 \\ b_n &= b_{n-1} + 2c_{n-1}h_{n-1} + 3d_{n-1}h_{n-1}^2 \end{aligned}$$

and **# C** to obtain,

$$\begin{aligned} d_{i-1} &= \frac{c_i - c_{i-1}}{3h_{i-1}} \\ d_{n-1} &= \frac{c_n - c_{n-1}}{3h_{n-1}} \end{aligned}$$

We have $4n$ equations with two additional auxiliary variables given by boundary conditions. It remains to solve b_i in terms of a_i and c_i. Substituting the expressions of d_{i-1} into the equations for **# B**,

$$b_{i-1} = f[x_{i-1}, x_i] - \frac{h_{i-1}}{3}(c_i + 2c_{i-1})$$

We can find equations for c_i by substituting these expressions for b_{i-1} and d_{i-1} into the equations for **# B**. Re-arranging the result,

$$h_{i-1}c_{i-1} + 2\left(h_i + h_{i-1}\right)c_i + h_i c_{i+1} = 3\left(f\left[x_i, x_{i+1}\right] - f\left[x_{i-1}, x_i\right]\right)$$

This is a **tridiagonal linear system** of $n-1$ equations with $n-1$ equations and $n+1$ unknowns. The cubic spline boundary conditions will give the remaining two equations. In summary,

The system takes $\mathcal{O}(n)$ to solve.

Example 17: Natural Cubic Spline

We have $c_0 = 0 = c_n$, producing the $(n-1) \times (n-1)$ system,

$$\mathbf{A} \cdot \mathbf{c} = \mathbf{b}$$

where $\mathbf{A}$ is the matrix,

$$\begin{pmatrix} 2\left(h_0 + h_1\right) & h_1 & & & \\ h_1 & 2\left(h_1 + h_2\right) & & & \\ & \ddots & \ddots & \ddots & \\ & & 2\left(h_{n-3} + h_{n-2}\right) & h_{n-2} & \\ & & h_{n-2} & 2\left(h_{n-2} + h_{n-1}\right) & \end{pmatrix}$$

and $\mathbf{c}$ and $\mathbf{b}$ are given by,

$$\begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-2} \\ c_{n-1} \end{pmatrix} \qquad \begin{pmatrix} 3\left(f\left[x_1, x_2\right] - f\left[x_0, x_1\right]\right) \\ 3\left(f\left[x_2, x_3\right] - f\left[x_1, x_2\right]\right) \\ \vdots \\ 3\left(f\left[x_{n-2}, x_{n-1}\right] - f\left[x_{n-1}, x_{n-2}\right]\right) \\ 3\left(f\left[x_{n-1}, x_n\right] - f\left[x_{n-2}, x_{n-1}\right]\right) \end{pmatrix}$$

Example 18: Clamped Cubic Spline

Combining equations for c_i gives the $(n-1) \times (n-1)$ system,

$$\mathbf{A} \cdot \mathbf{c} = \mathbf{b}$$

where $\mathbf{A}$ is the matrix,

$$\begin{pmatrix} 2h_0 & h_0 & & & \\ h_0 & 2\left(h_0 + h_1\right) & h_1 & & \\ & \ddots & \ddots & \ddots & \\ & & h_{n-2} & 2\left(h_{n-2} + h_{n-1}\right) & h_{n-1} \\ & & & h_{n-1} & 2h_{n-1} \end{pmatrix}$$

and $\mathbf{c}$ and $\mathbf{b}$ are given by,

$$\begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-2} \\ c_{n-1} \end{pmatrix} \quad \begin{pmatrix} 3\left(f\left[x_0, x_1\right] - b_0\right) \\ 3\left(f\left[x_1, x_2\right] - f\left[x_0, x_1\right]\right) \\ \vdots \\ 3\left(f\left[x_{n-1}, x_n\right] - f\left[x_{n-2}, x_{n-1}\right]\right) \\ 3\left(b_n - f\left[x_{n-1}, x_n\right]\right) \end{pmatrix}$$

Remark. Writing down the tridiagonal linear system for the **not-a-knot cubic** spline is left as an exercise.

Since the c_i coefficients are related through a linear system, a change in data for cubic splines is technically **non-local**. However, if only y_i is changed, it can be shown that the largest change occurs at c_i, with changes in other c_j decaying as a function of $|i-j|$. Thus,

$$\max_{x\in[a,b]} |f(x) - S(x)| \leq CM_4h^4 = \mathcal{O}\left(h^4\right)$$

where $C > 0$ depends on the type of boundary conditions imposed.

The accuracy for cubic splines is comparable to Lagrange interpolation, but Runge's phenomenon is avoided on equally-spaced points.

Hermite Interpolation

We have seen how to interpolate function values using a polynomial interpolant. However, we may want to impose additional conditions, e.g., incorporating a specified slope at each point. This is the motivation for **Hermite interpolation**, which considers data on derivative values for the polynomial interpolant.

Definition (Hermite Interpolation)**.** Given a set of data points $\{(x_i, y_i, z_i)\}_{i=0}^n$ with slope values z_i, **Hermite interpolation** finds a polynomial f_{2n+1} so that $f_{2n+1}(x_i) = y_i$ and $f'_{2n+1}(x_i) = z_i$.

Remark. We select a degree $2n+1$ polynomial because,

- The number of coefficients that we require is $2n+2$
- The number of data points required to interpolate is $n+1$ (function values) plus $n+1$ (derivative values)

We call f_{2n+1} the **Hermite interpolant**.

We will not use monomial basis functions, because we saw that they were ill-conditioned. Instead, we will introduce,

Definition (Hermite Basis Functions)**.** The **Hermite basis functions** $h_k(x)$ and $\hat{h}_k(x)$ are degree $2n+1$ polynomials that satisfy,

$$h_k(x_i) = \begin{cases} 0, \text{ if } i \neq k \\ 1, \text{ if } i = k \end{cases} \quad \text{and} \quad \hat{h}'_k(x_i) = \begin{cases} 0, \text{ if } i \neq k \\ 1, \text{ if } i = k \end{cases}$$

for all $i \in [n]$. Moreover,

$$h'_k(x_i) = 0 \quad \text{and} \quad \hat{h}_k(x_i) = 0$$

These requirements are similar to the ones that we imposed for Lagrange basis functions. We write the Hermite basis functions as,

$$h_k(x) = \left(1 - 2(x - x_k)\,\ell'_k(x_k)\right)\ell_k^2(x)$$
$$\hat{h}_k(x) = (x - x_k)\,\ell_k^2(x)$$

where $\ell_k(x)$ are the k-th Lagrange basis functions of order n. We will not see the derivation for this proof. Now, we can express the Hermite interpolant as a linear combination of $h_k(x)$ and $\hat{h}_k(x)$,

$$f_{2n+1}(x) = \sum_{k=0}^{n} a_k h_k(x) + \sum_{k=0}^{n} b_k \hat{h}_k(x)$$

$\{h_0(x), \ldots, h_n(x), \hat{h}_0(x), \ldots, \hat{h}_n(x)\}$ form a set of linear independent functions, i.e., they form a basis for $\mathbb{P}_{2n+1}$.

To find the coefficients a_k and b_k, observe that,

$$y_i = f_{2n+1}(x_i) = \sum_{k=0}^{n} a_k \underbrace{h_k(x_i)}_{=0 \text{ if } k \neq i} + \sum_{k=0}^{n} b_k \underbrace{\hat{h}_k(x_i)}_{=0} = a_i$$

$$z_i = f'_{2n+1}(x_i) = \sum_{k=0}^{n} a_k \underbrace{h'_k(x_i)}_{=0} + \sum_{k=0}^{n} b_k \underbrace{\hat{h}'_k(x_i)}_{=0 \text{ if } k \neq i} = b_i$$

since $f_{2n+1}(x)$ interpolates y_i, z_i on x_i.

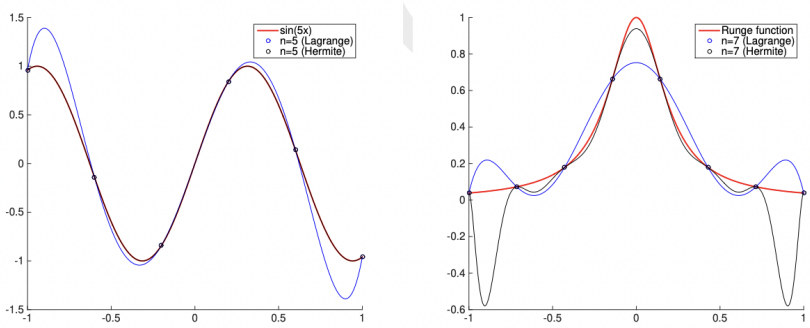
Similar constructions as the Newton basis for Lagrange interpolation can be constructed for Hermite interpolation.

Theorem 18 (Hermite Interpolation Error)**.** If $f \in C^{2n+1}([a,b])$, then

$$f(x) = f_{2n+1}(x) + \frac{f^{(2n+2)}(\xi(x))}{(2n+2)!}(x - x_0)^2 \cdots (x - x_n)^2$$

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

Since Hermite interpolation requires knowledge of the derivative values, it can be more accurate that Lagrange interpolation. However, Runge's phenomenon may still occur for Hermite interpolation.



Topic 4. Numerical Differentiation and Integration

We will approximate the derivative $\mathbf{D}_f$ and the integral $\mathbf{I}_f$ of a real-valued function $f : \mathbb{R} \to \mathbb{R}$ restricted to $[a, b]$. Recall that,

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

for every point $x_0 \in [a, b]$. In this way, $\mathbf{D}(f)(x_0)$ is said to be determined by **local information** about f around x_0. In contrast,

$$\mathbf{I}_f|_a^b = \lim_{n \to \infty} \sum_{i=1}^{n} f(x_i)\,\Delta x \quad \text{where} \quad \Delta x := \frac{b-a}{n}$$

is determined by **global information** about f on $[a, b]$.

We will use the following notation,

$\mathbf{D}(f) :=$ exact derivative of f
$\mathbf{D}_h(f) :=$ approximate derivative of f
$\mathbf{I}(f) :=$ exact integral of f
$\mathbf{I}_h(f) :=$ approximate integral of f

Definition (Degree of Accuracy). We say that,

1. $\mathbf{D}_h(f)$ has **degree of accuracy** of p if p is the largest positive integer with $\mathbf{D}(x^i) = \mathbf{D}_h(x^i)$ for any x and $i = 0, \cdots p$
2. $\mathbf{I}_h(f)$ has **degree of accuracy** of p if p is the largest positive integer with $\mathbf{I}(x^i) = \mathbf{I}_h(x^i)$ for any x and $i = 0, \cdots p$

Numerical Differentiation

The simplest method for approximating $\mathbf{D}(f)$ is to use Taylor's remainder theorem. Define a partition of the interval $[a, b]$ using

$$x_k = a + kh \quad \text{where} \quad h := \frac{b-a}{n}$$

and expand f about x_k. There exists $\xi(x) \in (x, x_k)$ such that,

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

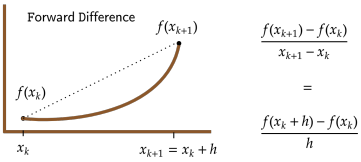
Setting $x = x_{k+1}$ gives that,

$$f(x_{k+1}) = f(x_k) + f'(x_k)\,h + \frac{f''(\xi(x_{k+1}))}{2}h^2$$

which we can re-arrange to obtain,

$$\underbrace{f'(x_k)}_{\mathbf{D}(f(x_k))} = \underbrace{\frac{f(x_{k+1}) - f(x_k)}{h}}_{\mathbf{D}_h(f(x_k))} + \underbrace{\frac{-f''(\xi(x_{k+1}))}{2}h}_{\mathcal{O}(h)}$$

From this, we obtain the **forward difference formula**. Similarly, the**backward difference formula** is obtained by setting $x = x_{k-1}$.



Definition (Forward Difference)**.** The **forward difference** is,

$$\mathbf{D}_h(f(x_k)) := \frac{f(x_{k+1}) - f(x_k)}{h}$$

Definition (Backward Difference)**.** The **backward difference** is,

$$\mathbf{D}_h(f(x_k)) := \frac{f(x_k) - f(x_{k-1})}{h}$$

The error term that we derived

$$-\frac{h}{2} \cdot f''(\xi(x_{k+1}))$$

is proportional to f''. This implies that $\mathbf{D}_h(f(x^i))$ is exact for $i \in \{0,1\}$, that is, has a degree of accuracy of 1.

Definition (Second Derivative Central Difference)**.** We define,

$$\mathbf{D}_h^2(f(x_k)) := \frac{f(x_{k+1}) - 2f(x_k) + f(x_{k-1})}{h^2}$$

as the **second derivative central difference**.

We expand f up to the fourth order using Taylor's Theorem around x_k. This gives the **second derivative central difference**,

$$\begin{aligned} f(x) =& f(x_k) + f'(x_k)(x - x_k) + \frac{f''(x_k)}{2}(x - x_k)^2 \\ &+ \frac{f^{(3)}(x_k)}{3!}(x - x_k)^3 + \frac{f^{(4)}(\xi(x))}{4!}(x - x_k)^4 \end{aligned}$$

Evaluating f at $x = x_{k+1}$ gives,

$$f(x_{k+1}) = f(x_k) + f'(x_k)h + \frac{f''(x_k)}{2}h^2 + \frac{f^{(3)}(x_k)}{3!}h^3 + \frac{f^{(4)}(\xi(x_{k+1}))}{4!}h^4$$

Evaluating f at $x = x_{k-1}$ gives,

$$f(x_{k-1}) = f(x_k) - f'(x_k)h + \frac{f''(x_k)}{2}h^2 - \frac{f^{(3)}(x_k)}{3!}h^3 + \frac{f^{(4)}(\xi(x_{k-1}))}{4!}h^4$$

Adding these two formulas and re-arranging gives,

$$\underbrace{f''(x_k)}_{\mathbf{D}^2(f(x_k))} = \underbrace{\frac{f(x_{k+1}) - 2f(x_k) + f(x_{k-1})}{h^2}}_{\mathbf{D}_h^2(f(x_k))} - \underbrace{\frac{f^{(4)}(\xi(x_{k+1})) + f^{(4)}(\xi(x_{k-1}))}{4!}h^2}_{\mathcal{O}(h^2)}$$

Expanding f up the third order would have given an $\mathcal{O}(h)$ error term.

The error term that we derived is proportional to $f^{(4)}$. This implies that $\mathbf{D}_h^2(f(x^i))$ is exact for $i \in \{0,1,2,3\}$, that is, has a degree of accuracy of 3. We can simplify the error term as follows,

> **Remark.** There exists ξ_k between $\xi(x_{k-1})$ and $\xi(x_{k+1})$ so that,
>
> $$f^{(4)}(\xi_k) = \frac{f^{(4)}(\xi(x_{k+1})) + f^{(4)}(\xi(x_{k-1}))}{2}$$
>
> **Proof.** Apply the Intermediate Value Theorem. □

We want to design more accurate finite difference approximations. Our current approximation schemes are a linear combination of the function values $f(x_{k-1}), f(x_k), f(x_{k+1})$. The natural extension is to use more function values at different points:

$$\mathbf{D}_h(f(x_k)) = \sum_{i=-L}^{R} a_i \cdot f(x_{k+i})$$

where the neighbouring points x_{k+i} of x_k form the **stencil** of $\mathbf{D}_h(f)$. The constants a_i are the coefficients associated with the stencil. We will derive the finite difference approximation for $\mathbf{D}(f)$ using

$$\mathbf{D}_h f(x) = af(x+h) + bf(x) + cf(x-h)$$

and maximizing for degree of accuracy. There are three parameters, a, b, and c, so we can achieve a degree of accuracy of two.

$$0 = \mathbf{D}(1) = \mathbf{D}_h(1) = a + b + c \Rightarrow a + b + c = 0$$

Next, we have that,

$$\begin{aligned} 1 = \mathbf{D}(x) = \mathbf{D}_h(x) &= a(x+h) + bx + c(x-h) \\ &= \underbrace{(a+b+c)}_{=0}x + (a-c)h \Rightarrow a - c = \frac{1}{h} \end{aligned}$$

Finally,

$$\begin{aligned} 2x = \mathbf{D}(x^2) = \mathbf{D}_h(x^2) &= a(x+h)^2 + bx^2 + c(x-h)^2 \\ &= \underbrace{(a+b+c)}_{=0}x^2 + \underbrace{(a-c)h}_{1} \cdot 2x + (a+c)h^2 \\ &\Rightarrow a + c = 0 \end{aligned}$$

This gives the **central difference** formula,

$$\mathbf{D}_h(f(x)) := \frac{f(x+h) - f(x-h)}{2h}$$

which has a degree of accuracy of 2. Expand using Taylor's Theorem around x and evaluate $x+h$ and $x-h$. For some $\xi^{\pm} \in (x, x \pm h)$,

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

Subtracting these two equations and re-arranging gives,

$$\underbrace{f'(x)}_{\mathbf{D}f(x)} = \underbrace{\frac{f(x+h) - f(x-h)}{2h}}_{\mathbf{D}_h f(x)} + \underbrace{\frac{f^{(3)}(\xi^+) + f^{(3)}(\xi^-)}{12}h^2}_{\mathcal{O}(h^2)}$$

which we can simplify by the Intermediate Value Theorem to,

$$\frac{h^2}{6} \cdot f^{(3)}(\xi(x))$$

Example 19: Caveat in Numerical Differentiation

Consider the central difference approximation $\mathbf{D}_h(f(x))$ for,

$$f(x) = \sin(x)$$

h	$\lvert\mathbf{D}f(\pi) - \mathbf{D}_h f(\pi)\rvert$
10^0	0.158529015192104
10^{-1}	0.00166583353171768
10^{-2}	1.66665833548629e − 05
10^{-3}	1.66666768497414e − 07
10^{-4}	1.66455649264208e − 09
10^{-5}	1.01152419773598e − 11
10^{-6}	1.39611433525033e − 10
10^{-7}	1.63658042673376e − 09
⋮	⋮
10^{-12}	8.89005823410116e − 05

Errors that occur when computing $f(x+h)$ and $f(x-h)$ are amplified by the division by h because h tends to ϵ_{mach}.

Due to round-off and discretization, $\tilde{f}$ approximates f up to ϵ.

$$|\tilde{f}(x+h) - f(x+h)| \leq \epsilon$$
$$|\tilde{f}(x-h) - f(x-h)| \leq \epsilon$$

Therefore, the actual error can be estimated in three parts,

$$\begin{aligned}
|f'(x) - \mathbf{D}_h\tilde{f}(x)| &= \left| f'(x) - \frac{\tilde{f}(x+h) - \tilde{f}(x-h)}{2h} \right| \\
&= \left| \left(f'(x) - \frac{f(x+h) - f(x-h)}{2h} \right) \right. \\
&\quad \left. + \left(\frac{f(x+h) - \tilde{f}(x+h)}{2h} \right) + \left(\frac{\tilde{f}(x-h) - f(x-h)}{2h} \right) \right| \\
&\leq \left| \frac{f^{(3)}(\xi(x))}{3!} \cdot h^2 \right| + \frac{\epsilon}{2h} + \frac{\epsilon}{2h} \\
&\leq \frac{M_3}{6} \cdot h^2 + \frac{\epsilon}{h} \nrightarrow 0 \text{ as } h \to 0
\end{aligned}$$

by applying the triangle inequality. We want to determine how small h can be before the error dominates. Applying standard results from single-variable calculus, it is left as an exercise to show that

$$\frac{M_3}{6} \cdot h^2 + \frac{\epsilon}{h}$$

is minimized at the point,

$$h^* = \left(\frac{3\epsilon}{M_3} \right)^{\frac{1}{3}}$$

In summary,

Finite Difference Method	Error	Degree of Accuracy
Forward and Backward Difference	$\mathcal{O}(h)$	1
Central Difference	$\mathcal{O}\left(h^2\right)$	2
Second Derivative Central Difference	$\mathcal{O}\left(h^2\right)$	3

Numerical Integration

Numerical integration is called **numerical quadrature**. The first idea will be to approximate integrals using Lagrange interpolants. This approach is called **Newton-Cotes quadrature**. As usual, we will begin by defining a partition of the interval $[a, b]$ using

$$x_k = a + kh \quad \text{where} \quad h := \frac{b-a}{n}$$

By the Lagrange Interpolation Theorem, there is $\xi \in [a, b]$ such that,

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

Integrating both sides on $[a,b]$ gives,

$$\begin{aligned}\mathbf{I}(f) &= \int_a^b f(x)dx \\ &= \underbrace{\int_a^b f_n(x)dx}_{\mathbf{I}_h(f)} + \underbrace{\int_a^b \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x-x_0)\cdots(x-x_n)\,dx}_{\mathbf{E}_n(f)}\end{aligned}$$

The **midpoint rule** is used if $n = 0$, in which case $f_0(x)$ is the constant function. We define $f_0(x)$ to be the midpoint value of f,

$$f_0(x) = f\left(\frac{a+b}{2}\right)$$

which implies that,

$$\mathbf{I}_h(f) = \int_a^b f_0(x)dx = \int_a^b f\left(\frac{a+b}{2}\right)dx = (b-a)\cdot f\left(\frac{a+b}{2}\right)$$

Definition (Midpoint Rule)**.** The **midpoint rule** is,

$$\mathbf{I}_h(f) = (b-a)\cdot f\left(\frac{a+b}{2}\right)$$

The **trapezoid rule** is used if $n = 1$, in which case we use Newton's Divided Difference Formula with $x_0 = a$ and $x_1 = b$,

$$f_1(x) = f(a) + \frac{f(b)-f(a)}{b-a}\cdot(x-a)$$

which implies that

$$\begin{aligned}\mathbf{I}_h(f) &= \int_a^b f_1(x)dx = \int_a^b \left(f(a) + \frac{f(b)-f(a)}{b-a}(x-a)\right)dx \\ &= (b-a)\cdot f(a) + \frac{f(b)-f(a)}{b-a}\frac{(x-a)^2}{2}\bigg|_a^b = (b-a)\cdot\frac{f(a)+f(b)}{2}\end{aligned}$$

Definition (Trapezoidal Rule)**.** The **trapezoid rule** is,

$$\mathbf{I}_h(f) = (b-a)\cdot\frac{f(a)+f(b)}{2}$$

Simpson's rule is used if $n = 2$, in which case we use Newton's Divided Difference Formula with the parameters,

$$m = \frac{a+b}{2} \qquad x_0 = a \qquad x_1 = m \qquad x_2 = b$$

to obtain the formula,

$$f_2(x) = f(a) + f[a,m](x-a) + f[a,m,b](x-a)(x-m)$$

where

$$f[a,m] = \frac{2(f(m)-f(a))}{b-a} \quad \text{and} \quad f[a,m,b] = \frac{2(f(b)-2f(m)+f(a))}{(b-a)^2}$$

Integrating by parts gives,

$$\begin{aligned}\mathbf{I}_h(f) &= \int_a^b f_2(x)dx \\ &= \int_a^b (f(a)+f[a,m](x-a)+f[a,m,b](x-a)(x-m))dx \\ &= (b-a)f(a)+f[a,m]\frac{(b-a)^2}{2}+f[a,m,b]\frac{(b-a)^3}{12}\end{aligned}$$

Definition (Simpson's Rule)**. Simpson's rule** is,

$$\mathbf{I}_h(f) = (b-a)\cdot\frac{f(a)+4f(m)+f(b)}{6}$$

As with numerical differentiation, we need to determine the degree of accuracy of Newton-Cotes quadrature. By definition, we check the largest p so that $\mathbf{I}(x^i) = \mathbf{I}_h(x^i)$ for $i \in [p]$.

$$\mathbf{E}_n(f) := \int_a^b \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x-x_0)\cdots(x-x_n)\,dx$$

is our error, which gives $\mathbf{E}_n\left(x^i\right) = 0$ for all $i = 0,\ldots,n$. It follows that $n+1$ point Newton-Cotes quadrature has degree of accuracy $\geq n$.

Lemma 3 (Weighted Mean Value Theorem for Integrals)**.** If $h \in C([a,b])$ and g does not change sign on $[a,b]$, then for $\eta \in (a,b)$,

$$\int_a^b h(x)g(x)dx = h(\eta)\int_a^b g(x)dx$$

Proposition 1. For the Trapezoidal rule ($n=1$),

$$\mathbf{E}_1(f) = \int_a^b \underbrace{\frac{f''(\xi(x))}{2}}_{h(x)}\underbrace{(x-a)(x-b)}_{g(x)}dx$$

Proof. Define g and h as follows,

$$\begin{aligned}h(x) &= \frac{f''(\xi(x))}{2} \\ g(x) &= (x-a)(x-b)\end{aligned}$$

Since g does not change sign on $[a,b]$, we have,

$$\begin{aligned}\mathbf{E}_1(f) &= h(\eta)\int_a^b (x-a)(x-b)dx \\ &= -f''(\xi(\eta))\frac{(b-a)^3}{12} \\ &= \mathcal{O}\left((b-a)^3\right)\end{aligned}$$

by the Weighted Mean Value Theorem. The error term is proportional to $f^{(2)}$, so the trapezoidal rule has degree of accuracy equal to 1. □

Similar calculations can be done for the Midpoint and Simpson rules.

Proposition 2. For the Midpoint rule ($n = 0$),

$$\mathbf{E}_0(f) = -\frac{f''(\xi(\eta))}{24}(b-a)^3 = \mathcal{O}\left((b-a)^3\right)$$

that is, we have a degree of accuracy of 1.

Proposition 3. For the Simpson rule ($n = 2$),

$$\mathbf{E}_2(f) = -\frac{f^{(4)}(\xi(\eta))}{90}\left(\frac{b-a}{2}\right)^5 = \mathcal{O}\left((b-a)^5\right)$$

that is, we have a degree of accuracy of 3.

Consider an even number n. The $(n+1)$ point Newton-Cotes quadrature has degree of accuracy of $n+1$ due to cancellations in the error term.

Gauss Quadrature

Newton-Cotes quadrature is a linear combination of $f(x_k)$,

$$\mathbf{I}_h(f) = c_0 f(a) + c_1 f(a+h) + \cdots + c_n f(b) = \sum_{k=0}^{n} c_k \cdot f(a+kh)$$

In this subsection, we will no longer restrict x_k to be uniformly spaced. This will produce quadratures with a higher order of accuracy. For simplicity, we will look at integration over $t \in [-1,1]$.

Example 20: Two Point Gauss Quadrature

We want to find w_0, w_1, t_0, and t_1 so that the quadrature of the form $\mathbf{I}_h(f) = w_0 \cdot f(t_0) + w_1 \cdot f(t_1)$ has the highest degree of

accuracy. With 4 parameters, we can satisfy

$$\begin{aligned} 2 &= \int_{-1}^{1} 1dt = I(1) = I_h(1) = w_0 + w_1 \\ 0 &= \int_{-1}^{1} tdt = I(t) = I_h(t) = w_0 t_0 + w_1 t_1 \\ \frac{2}{3} &= \int_{-1}^{1} t^2 dt = I(t^2) = I_h(t^2) = w_0 t_0^2 + w_1 t_1^2 \\ 0 &= \int_{-1}^{1} t^3 dt = I(t^3) = I_h(t^3) = w_0 t_0^3 + w_1 t_1^3 \end{aligned}$$

up to degree of accuracy of 3. Solving the 4 equations gives,

$$w_0 = 1 = w_1 \text{ and } -t_0 = \frac{1}{\sqrt{3}} = t_1$$

Thus, the two point Gauss quadrature on $[-1, 1]$,

$$\mathbf{I}_h(f) = f\left(-\frac{1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right)$$

has degree of accuracy up to 3.

In general, the goal has been to design approximations $\mathbf{I}_h(f)$ that maximize the degree of accuracy with respect to $\mathbf{I}_f$. We can decompose $\mathbf{I}_f$ as $\mathbf{I}_f = \mathbf{I}_h(f) + \mathbf{E}_n(f)$. Letting the integral be a linear combination of n function values $f(t_k)$,

$$\mathbf{I}_h(f) = \sum_{k=0}^{n-1} w_k \cdot f(t_k)$$

where t_k are called **Gauss points**.

Gauss quadrature has $2n$ parameters for n points. This implies a degree of accuracy of $2n - 1$.

> **Remark.** Relative to Newton-Cotes, Gauss quadrature is more accurate for the same number of function evaluations.

Example 21: Common Gauss Quadrature on $[-1, 1]$

n	w_k	t_k	Degree of Accuracy
1	2	0	1
2	$1, 1$	$-\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}$	3
3	$\frac{5}{9}, \frac{8}{9}, \frac{5}{9}$	$-\sqrt{\frac{3}{5}}, 0, \sqrt{\frac{3}{5}}$	5

Definition (Legendre Polynomials). The **Legendre polynomials** $P_k(x)$ are defined recursively for $k \in \mathbb{N}$ by the base case,

$$P_0(x) = 1$$
$$P_1(x) = x$$

and the recursion formula,

$$P_{k+1}(x) := x\left(\frac{2k+1}{k+1}\right)P_k(x) - \left(\frac{k}{k+1}\right)P_{k-1}(x)$$

Theorem 19. For an n-point Gauss quadrature,

1. The Gauss points $\{t_k\}_{k=0}^{n-1}$ are given by the zeros of $P_n(x)$
2. The weights $\{w_k\}_{k=0}^{n-1}$ are given by

$$w_j = \int_{-1}^{1} \ell_j(x)dx$$

where $\ell_j(x)$ is the j-th Lagrange function interpolating $\{t_k\}_{k=0}^{n-1}$.

To obtain the highest degree of accuracy with n points, we select t_k to be the zeroes of the Legendre polynomials.

Example 22: Two Point Gauss Quadrature

We will recover t_k and w_k for a two point Gauss quadrature.

$$P_2(x) = \frac{3}{2}xP_1(x) - \frac{1}{2}P_0(x) = \frac{3}{2}x^2 - \frac{1}{2} = \frac{3}{2}\left(x^2 - \frac{1}{3}\right)$$

which implies that $-t_0 = \frac{1}{\sqrt{3}} = t_1$, Moreover,

$$\ell_{0,1}(x) = \frac{1 \mp \sqrt{3}x}{2} \Rightarrow w_{0,1} = \int_{-1}^{1} \frac{1 \mp \sqrt{3}x}{2}dx = 1$$

which implies that $w_0 = 1 = w_1$.

We can generalize our analysis from $[-1,1]$ to intervals $[a,b]$ by changing coordinates. We use the formula for $x(t)$ given by,

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

We obtain the formula,

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

so the n point Gauss quadrature becomes,

$$\mathbf{I}_h(f) = \frac{b-a}{2}\cdot\sum_{k=0}^{n-1} w_k \cdot f\left(\frac{b-a}{2}t_k + \frac{b+a}{2}\right)$$

We will derive an error term for the Gauss quadrature.

Theorem 20. Denote the n point Gauss quadrature as,

$$\mathbf{I}_h(f) = \frac{b-a}{2} \cdot \sum_{k=0}^{n-1} w_k \cdot f\left(x\left(t_k\right)\right)$$

where $x(t) = \frac{b-a}{2}t + \frac{b+a}{2}$. Write,

$$\mathbf{I}(f) = \int_a^b f(x)dx$$

If $f \in C^{(2n)}([a,b])$, then for $\xi \in (a,b)$,

$$\mathbf{I}(f) = \mathbf{I}_h(f) + \frac{f^{(2n)}(\xi)}{(2n)!} \int_a^b \left(z - x\left(t_0\right)\right)^2 \cdots \left(z - x\left(t_{n-1}\right)\right)^2 dz$$

This is consistent with a degree of accuracy of $2n-1$ for the n point Gauss quadrature.

Composite Quadrature

We have looked at Newton-Cotes or Gauss quadrature over a general interval $[a,b]$. We found that the error is $\mathcal{O}\left((b-a)^p\right)$ for some positive integer p. To reduce the quadrature error, we will subdivide $[a,b]$ into smaller intervals of uniform length. Observe,

$$\begin{aligned}\mathbf{I}(f) &= \int_a^b f(x)dx = \sum_{k=0}^{n-1} \int_{x_k}^{x_{k+1}} f(x)dx \\ &= \sum_{k=0}^{n-1} \underbrace{\mathbf{I}_{h,k}(f)}_{\substack{\texttt{quadrature on} \\ [x_k, x_{k+1}]}} + \underbrace{\mathbf{E}_{h,k}(f)}_{\substack{\texttt{local error on} \\ [x_k, x_{k+1}]}}\end{aligned}$$

This is the main idea behind **composite quadrature**,

$$\mathbf{I}_{h,\mathsf{c}}(f) := \sum_{k=0}^{n-1} \mathbf{I}_{h,k}(f)$$
$$\mathbf{E}_{h,\mathsf{c}}(f) := \sum_{k=0}^{n-1} \mathbf{E}_{h,k}(f)$$

Example 23: Composite Midpoint Rule

For $\xi_k \in (x_k, x_{k+1})$ on each $[x_k, x_{k+1}]$,

$$\int_{x_k}^{x_{k+1}} f(x)dx = \underbrace{hf\left(x_{k+\frac{1}{2}}\right)}_{\mathbf{I}_{h,k}(f)} - \underbrace{f''\left(\xi_k\right)\frac{h^3}{24}}_{\mathbf{E}_{h,k}(f)}$$

where $x_{k+\frac{1}{2}} := \frac{x_k + x_{k+1}}{2}$. Putting this together,

$$\begin{aligned}\mathbf{I}(f) &= h\sum_{k=0}^{n-1} f\left(x_{k+\frac{1}{2}}\right) - \frac{h^3}{24}\sum_{k=0}^{n-1} f''\left(\xi_k\right) \\ &= \underbrace{h\sum_{k=0}^{n-1} f\left(x_{k+\frac{1}{2}}\right)}_{\mathbf{I}_{h,c}} - \underbrace{\frac{(b-a)}{24} f''(\xi)h^2}_{\mathbf{E}_{h,c}}\end{aligned}$$

where the composite error is in $\mathcal{O}\left(h^2\right)$.

Proposition 4. Suppose $f \in C^p([a,b])$ and $\xi_k \in [a,b]$ for $k \in [n-1]$. Using the Intermediate Value Theorem, we can show that

$$\frac{1}{n}\sum_{k=0}^{n-1} f^{(p)}\left(\xi_k\right) = f^{(p)}(\xi)$$

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

Proof. This is left as an exercise. □

Example 24: Composite Trapezoidal and Simpson Rule

We can derive the **Composite Trapezoidal Rule**,

$$\mathbf{I}(f) = \underbrace{\frac{h}{2}\sum_{k=0}^{n-1}\left(f\left(x_k\right) + f\left(x_{k+1}\right)\right)}_{\mathbf{I}_{h,c}} - \underbrace{\frac{(b-a)}{12} f''(\xi)h^2}_{\mathbf{E}_{h,c}}$$

as well as the **Composite Simpson's Rule**,

$$\mathbf{I}(f) = \underbrace{\frac{h}{6}\sum_{k=0}^{n-1}\left(f\left(x_k\right) + 4f\left(x_{k+\frac{1}{2}}\right) + f\left(x_{k+1}\right)\right)}_{\mathbf{I}_{h,c}} - \underbrace{\frac{(b-a)}{90} f''(\xi)h^4}_{\mathbf{E}_{h,c}}$$

The error terms are in $\mathcal{O}\left(h^2\right)$ and $\mathcal{O}\left(h^4\right)$, respectively.

In summary,

Newton-Cotes	Composite Error	Degree of Accuracy
Midpoint	$\mathcal{O}(h^2)$	1
Trapezoidal	$\mathcal{O}\left(h^2\right)$	1
Simpson	$\mathcal{O}\left(h^4\right)$	3

Topic 5. Numerical Methods for Initial Value Problems

Defining Initial Value Problems

We can write the initial value problem,

$$\begin{cases} y'(t) = f(t, y(t)) \\ y(0) = y_0 \end{cases}$$

for $t \in [0, T]$ as

$$y(t) = y_0 + \int_0^t \underbrace{f(s, y(s))}_{y'(s)} \, ds$$

using the Fundamental Theorem of Calculus. In many cases, we cannot obtain an explicit formula for $y(t)$. Using numerical methods, f can be approximated by a reasonably smooth function.

We will choose a uniformly spaced step size $h = T/N$ to step forward on each $t_n = nh$ by solving a discrete set of algebraic equations for y_n that approximates the value of $y(t_n)$. This discrete set of algebraic equations is called a **discretization**. One way to find a discretization is to approximate y' by a finite difference.

$$\begin{cases} y'(t) = f(t, y(t)) \\ y(0) = y_0 \end{cases} \quad \to \quad \begin{cases} \mathbf{D}_h(y(t_n)) = f(t_n, y_n) \\ y_0 \text{ is given} \end{cases}$$

Definition (Explicit Discretization). A discretization is **explicit** if y_{n+1} can be solved for explicitly. Otherwise, it is called **implicit**.

Remark. Implicit methods require root finding at each time step t_n to solve for y_{n+1}. This makes them costly computationally, but they generally have better stability properties.

Definition (k-Step). A discretization is called **k-step** if it requires knowing k previous steps $y_n, \cdots, y_{n-k+1}$ to compute y_{n+1}.

Methods based on Numerical Differentiation

Example 25: Explicit and Implicit Discretization

We have seen the following approximations for $\mathbf{D}_h(y(t_n))$:

1. The **forward difference** formula is a 1 step explicit method

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

2. The **backward difference** formula is 2 step implicit method

$$y_{n+1} = y_n + hf\left(t_{n+1}, y_{n+1}\right)$$

3. The **central difference** formula is 2 step explicit method

$$y_{n+1} = y_{n-1} + 2hf\left(t_n, y_n\right)$$

Example 26: Studying Population Dynamics

Let $N(t)$ be the number of individuals in a given population at time t. Consider the differential equation,

$$\frac{dN}{dt} = \frac{r}{k}(K - N(t)) \cdot N(t)$$

for a positive **rate** r and a **carrying capacity** k of the environment. Consider the initial value problem,

$$\begin{cases} y' = \frac{1}{5}(10 - y)y \\ y_0 = 5 \end{cases}$$

We will use Euler's Method with the Forward Difference:

$$y_{n+1} := y_n + h \cdot \frac{1}{5}(10 - y_n) \cdot y_n$$
$$y_0 = 5$$

which we can solve in Matlab as follows,

```
r = 2; % growth rate
K = 10 % carrying capacity
N0 = 5; % initial condition
h = .01 % step size
N = 300 % step count

% memory allocation
y = zeros(1, N+1);
y(1) = N0;

% forward euler
for n = 1:N
  y(n+1) = y(n) + h*(1/5)*(10-y(n))*y(n)
end
```

Methods based on Numerical Integration

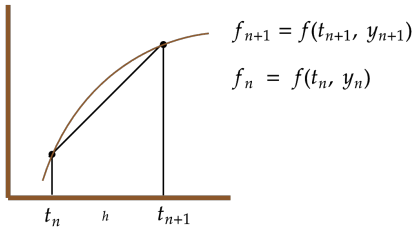
Rather than discretizing the derivative, we can discretize the integral,

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(s, y(s))ds$$

leads to the following initial value problem,

$$\begin{cases} y_{n+1} = y_n + \mathbf{I}_{h,n}(f) \\ y_0 \text{ is given} \end{cases}$$

One simple strategy for approximating the integral is to use the 1 step implicit **trapezoidal method**. We can visualize it as follows,



$$\int_{t_n}^{t_{n+1}} f(s, y(s))\, ds \approx \int_{t_n}^{t_{n+1}} t_n + \frac{f_{n+1} - f_n}{h}(s - t_n)\, dt$$

Definition (Trapezoidal Method)**.** The **trapezoidal method** is,

$$y_{n+1} = y_n + \frac{h}{2} \cdot (f_{n+1} + f_n)$$

which is 1-step implicit.

We will see two alternatives to this method,

Definition (Adam-Bashford)**.** The **Adam-Bashford** method is,

$$y_{n+1} = y_n + \frac{h}{2} \cdot (3f_n - f_{n-1})$$

which is 2 step explicit.

Definition (Adam-Moulton)**.** The **Adam-Moulton** method is,

$$y_{n+1} = y_n + \frac{h}{12} \cdot (5f_{n+1} + 8f_n - f_{n-1})$$

which is 2 step implicit.

Example 27: Newton's Law of Cooling

Given a constant $k > 0$, the change in temperature $T(t)$ of an object due to conduction is as follows,

$$\begin{cases} T'(0) = -k(T(t) - T_{\text{env}}) \\ T(0) = T_0 \end{cases}$$

where T_{env} is the environment temperature, and T_0 is the initial temperature. The exact solution is,

$$T(t) = (T_0 - T_{\text{env}})e^{-kt} + T_{\text{env}}$$

We can use the three 1-step methods to solve our system,

1. Forward Euler's Method gives,

$$T_{n+1} = T_n + h\left(-k\left(T_n - T_{\text{env}}\right)\right)$$

2. Backward Euler's Method gives,

$$T_{n+1} = T_n + h\left(-k\left(T_{n+1} - T_{\text{env}}\right)\right)$$

3. The Trapezoidal Method gives,

$$T_{n+1} = T_n + \frac{h}{2}\left(-k\left(T_{n+1} - T_{\text{env}}\right) - k\left(T_n - T_{\text{env}}\right)\right)$$

To compare the accuracy of each method, we look at the error

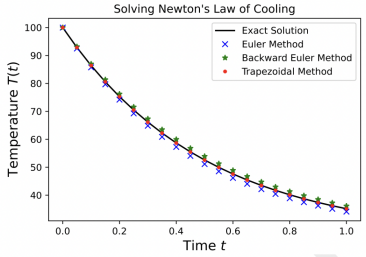
$$|T_N(1) - T(1)|$$

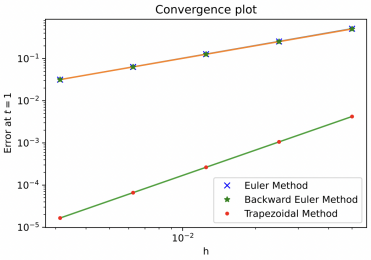
N increases, equivalently, as h decreases.

h	Forward	Backward	Trapezoidal
0.05	1.0318971	0.9981259	0.0169282
0.025	0.5117345	0.5032799	0.0042299
0.0125	0.2548108	0.2526965	0.0010574
0.00625	0.1271411	0.1266125	0.0002643

We can estimate the order $\mathcal{O}\left(h^p\right)$ of each method using the slope of the log-log plot in the error versus h. We get that,

1. Forward and Backward Euler have a slope of 1, i.e., $\mathcal{O}(h)$
2. The Trapezoidal Method has a slope of 2, i.e., $\mathcal{O}(h^2)$





Error Analysis

We want to **quantify the error** in approximating initial value problems as the step size approaches 0. To do this, we will re-write our k-step discretization scheme as the zero of a map ϕ,

$$\phi_h(t_n, y_{n+1}, y_n, \cdots, y_{n-k+1}) = 0$$

which is satisfied for the unknown y_{n+1}.

Definition (Local Truncation Error)**.** The **local truncation error** is,

$$\tau_h(t_n) := \Phi_h(t_n, y(t_{n+1}), y(t_n), \ldots, y(t_{n-k+1}))$$

Remark. In general, $\tau_h(t_n) = \mathcal{O}(h^p)$ for some power p.

Example 28: Local Truncation Error for Euler's Method

We will bound the local truncation error for Euler's method. Let y be a solution to the initial value problem. Then,

$$\begin{aligned}
\tau_h(t_n) &:= \Phi_h(t_n, y(t_{n+1}), y(t_n)) \\
&= y(t_{n+1}) - y(t_n) - hf(t_n, y(t_n)) \\
&= \left(y(t_n) + y'(t_n)h + \frac{y''(\xi_n)}{2}h^2\right) - y(t_n) - hf(t_n, y(t_n)) \\
&= h\underbrace{\left(y'(t_n) - f(t_n, y(t_n))\right)}_{=0} + \frac{y''(\xi_n)}{2}h^2
\end{aligned}$$

for $\xi_n \in (t_n, t_{n+1})$ obtained by Taylor's Theorem. Define,

$$M_2 := \max_{t\in[0,T]} |y''(t)|$$

It follows that,

$$|\tau_h(t_n)| \leq \frac{M_2}{2}h^2$$

and therefore $\tau_h(t_n) = \mathcal{O}(h^2)$.

Definition (Global Truncation Error)**.** The **global truncation error** is,

$$\frac{\tau_h}{h} = \frac{1}{h} \cdot \max_{0\leq n\leq N} |\tau_h(t_n)|$$

Remark. A method is of order p if,

$$\frac{\tau_h}{h} = \mathcal{O}\left(h^p\right)$$

This condition holds if and only if $\tau_h\left(t_n\right) = \mathcal{O}\left(h^{p+1}\right)$.

Definition (Consistency). A method is **consistent** if,

$$\lim_{h\to 0} \frac{\tau_h}{h} = 0$$

Remark. If a method is of order $p > 0$, then it is consistent.

Example 29: Consistency of Euler's Method

We determined in the previous example that,

$$\left|\tau_h\left(t_n\right)\right| \leq Ch^2$$

for some $C > 0$ independent of t_n. Thus,

$$\frac{\tau_h}{h} = \max_{0\leq n\leq N} \frac{\left|\tau_h\left(t_n\right)\right|}{h} = \mathcal{O}(h)$$

so Euler's method is of order 1 and hence consistent.

Definition (Convergence). A method is called **convergent** if,

$$\lim_{h\to 0} \max_{0\leq n\leq N} |\underbrace{y(t_n) - y_n}_{e_n}| = 0$$

where e_n denotes the error at time t_n.

Theorem 21 (Convergence of Euler's Method)**.** Let $f : [0, T] \times \mathbb{R} \to \mathbb{R}$ be continuous in t and continuously differentiable in y. If there exists $L > 0$ such that

$$\left|\frac{\partial f}{\partial z}(t, y)\right| \leq L \forall (t, y) \in [0, T] \times \mathbb{R}$$

then the Euler method converges.

Proof. By definition of Euler's method,

$$(\star) \quad y_{n+1} = y_n + hf\left(t_n, y_n\right)$$

and the local trunction error is,

$$\tau_h(t_n) = y(t_{n+1}) - y(t_n) - hf(t_n, y(t_n))$$

Re-arranging for $y(t_{n+1})$ gives that,

$$y(t_{n+1}) = y(t_n) + hf(t_n, y(t_n)) + \tau_h(t_n)$$

Subtracting by $(\star)$ gives that,

$$e_{n+1} = e_n + h(f(t_n, y(t_n)) - f(t_n, y_n)) + \tau_h(t_n)$$

By the Mean Value Theorem, $\exists \xi_n \in (y(t_n), t_n)$ such that,

$$f(t_n, y(t_n)) - f(t_n, y_n) = \frac{\partial f}{\partial z}(t_n, \xi_n)(y(t_n) - y_n)$$

The following inequalities hold by assumption and by definition,

$$\left|\frac{\partial f}{\partial z}(t, z)\right| \leq L \qquad |\tau_h(t_h)| \leq \tau_h$$

It follows that,

$$\begin{aligned}
|e_{n+1}| &\leq |e_n| + hL|e_n| + \tau_h \\
&= (1 + hL)|e_n| + \tau_h \\
&\leq (1 + hL)((1 + hL)|e_{n-1}| + \tau_h) + \tau_h \\
&= (1 + hL)^2 |e_{n-1}| + (1 + (1 + hL))\tau_h \\
&\leq \dots
\end{aligned}$$

Repeating inductively,

$$|e_{n+1}| \leq (1 + hL)^{n+1}|e_0| + (1 + (1 + hL) + \dots + (1 + hL)^n)\tau_h$$

where $|e_0| = 0$. Summing the geometric series,

$$\sum_{i=0}^{n}(1 + hL)^i = \frac{(1 + hL)^{n+1} - 1}{(1 + hL) - 1}$$

so that our bound can be simplified as follows,

$$\begin{aligned}
|e_{n+1}| &\leq \frac{(1 + hL)^{n+1} - 1}{hL}\tau_h \\
&\leq \frac{e^{(n+1)hL} - 1}{L}\frac{\tau_h}{h} \quad \text{since } 1 + x \leq e^x \text{ for all } x \in \mathbb{R} \\
&\leq \frac{e^{Nhl} - 1}{L}\frac{\tau_h}{h} \quad \text{since } 0 \leq n + 1 \leq N
\end{aligned}$$

Using the fact that $T = n \cdot h$,

$$|e_{n+1}| \leq \underbrace{\frac{e^{TL} - 1}{L}}_{C} \cdot \frac{\tau_h}{h}$$

It follows that Euler's method is consistent since C is a constant that does not depend on h. That is, as $h \to 0$,

$$|e_{n+1}| \leq C\frac{\tau_h}{h} \to 0$$

□

While all 1 step methods are convergent, we will see that not every consistent k-step method converges. We require some additional notion of **stability** to guarantee convergence for $k \geq 2$ step methods.

Definition (Linear Multistep Method). A k-step method is a **linear multistep method** if the discretization can be written as

$$y_{n+1} + \sum_{i=0}^{k-1} a_{k-1-i} y_{n-i} = h\left(b_k f_{n+1} + \sum_{i=0}^{k-1} b_{k-1-i} f_{n-i}\right)$$

where a_i and b_i are constants and $f_i = f(t_i, y_i)$. Equivalently,

$$\Phi_h := y_{n+1} + \sum_{i=0}^{k-1} a_{k-1-i} y_{n-i} - h\left(b_k f_{n+1} + \sum_{i=0}^{k-1} b_{k-1-i} f_{n-i}\right) = 0$$

Remark. The discretization Φ_h of a k-step linear multistep method is linear in $y_{n+1}, \ldots, y_{n-k+1}$ and $f_{n+1}, \ldots, f_{n-k+1}$.

Remark. If $b_k = 0$, then the method is explicit. Otherwise,

$$y_{n+1} + \sum_{i=0}^{k-1} a_{k-1-i} y_{n-i} = h\left(\sum_{i=0}^{k-1} b_{k-1-i} f_{n-i}\right)$$

and the method is implicit.

Example 30: Central Difference Method

Pick $a_0 = -1$, $a_1 = 0$, $b_0 = 0$, $b_1 = 2$, and $b_2 = 0$. This gives,

$$y_{n+1} = y_{n-1} + 2hf_n$$

Example 31: Adam-Bashford Method

Pick $a_0 = 0, a_1 = -1, b_0 = -\frac{1}{2}, b_1 = \frac{3}{2}$ and $b_2 = 0$. This gives,

$$y_{n+1} = y_n + \frac{h}{2}\left(3f_n - f_{n-1}\right)$$

Example 32: Adam-Moulton Method

Pick $a_0 = 0, a_1 = -1, b_0 = -\frac{1}{12}, b_1 = \frac{2}{3}$ and $b_2 = \frac{5}{12}$. This gives,

$$y_{n+1} = y_n + \frac{h}{12}\left(5f_{n+1} + 8f_n - f_{n-1}\right)$$

Using Taylor's Theorem, we can derive a set of **order** and **consistency** conditions for k-step linear multistep methods. These are:

Theorem 22 (Consistency Conditions)**.** A k-step linear multistep method is **consistent** if and only if we have:

$$\sum_{i=0}^{k-1} a_i = -1$$
$$\sum_{i=0}^{k} b_i = k + \sum_{i=0}^{k-1} i \cdot a_i$$

Theorem 23 (Order Conditions)**.** A k-step linear multistep method is of **order** p if and only for $q = 1, \ldots, p$ we have:

$$\sum_{i=0}^{k-1} a_i = -1$$
$$q\sum_{i=0}^{k} i^{q-1} \cdot b_i = k^q + \sum_{i=0}^{k-1} i^q \cdot a_i$$

Not all consistent linear multistep methods are convergent.

Example 33: Consistent but Divergent Linear Multistep Methods

Consider the following 2 step explicit linear multistep method,

$$y_{n+1} + 4y_n - 5y_{n-1} = h\left(4f_n + 2f_{n-1}\right)$$

The constants a_i and b_i are determined as follows,

$$a_1 = 4, a_0 = -5 \text{ and } b_2 = 0, b_1 = 4, b_0 = 2$$

This method is of order 3 by the order conditions and hence it

is consistent. We will apply it to the IVP

$$\begin{cases} y'(t) = -y(t) \\ y(0) = 1 \end{cases}$$

which we know has the following solution

$$y(t) = e^{-t}$$

for all $t \geq 0$. Using Euler's method to initialize y_1, we obtain an error that tends to infinity as $h \to 0$. This is shown below,

h	$\lvert y(1) - y_N \rvert$
0.2	11.7
0.1	6.51×10^3
0.05	2.37×10^8

Therefore, consistency is insufficient for convergence.

The previous example motivates the need for aa stronger condition that consistency to guarantee convergence. We require some idea of the stability of the system, which we will formalize using the concept of **zero-stability**. We begin with basic definitions.

Definition (Characteristic Polynomial). Let $\{a_i\}$ be the coefficients of a linear multistep method. The **characteristic polynomial** is,

$$p(\lambda) = \lambda^k + a_{k-1}\lambda^{k-1} + \dots a_1\lambda + a_0$$

Definition (Zero-Stability). Let $\{\lambda_1, \cdots, \lambda_k\}$ be the roots of the characteristic polynomial. The conditions for **zero-stability** are,

1. $|\lambda_i| \leq 1$
2. $|\lambda_i| = 1 \implies \lambda_i \neq \lambda_j$ for all $j \neq i$

That is, all roots have modulus less than or equal to 1 and all roots of modulus equal to 1 are distinct.

Example 34: Central Difference Method

Since $a_0 = -1$ and $a_1 = 0$,

$$p(\lambda) = \lambda^2 - 1 = (\lambda - 1)(\lambda + 1)$$

and zero-stability holds.

Example 35: Divergent Method

Since $a_0 = -5$ and $a_1 = 4$,

$$p(\lambda) = \lambda^2 + 4\lambda - 5 = (\lambda + 5)(\lambda - 1)$$

and zero-stability does not hold.

Theorem 24 (Convergence of Linear Multistep Methods)**.** Consistent linear multistep methods converge $\iff$ zero-stability holds.

Corollary. If a consistent linear multistep method converges, then it converges at the same order as the consistency error.

In summary,

1. Consistency implies convergence for 1 step methods
2. Consistency and the order conditions imply convergence for linear multistep methods
3. Consistent linear multistep methods converge if and only if zero-stability holds

Example 36: Adam-Bashford

We want to find the order of the Adam-Bashford method and show that it converges. This method is 2 step explicit,

$$y_{n+1} = y_n + \frac{h}{2}\left(3f_n - f_{n-1}\right)$$

Re-arranging gives,

$$y_{n+1} - y_n = \frac{h}{2}\left(3f_n - f_{n-1}\right)$$

and hence the coefficients are $a_0 = 0$, $a_1 = -1$, $b_0 = -\frac{1}{2}$, $b_1 = \frac{3}{2}$, and $b_2 = 0$. The characteristic polynomial is,

$$p(\lambda) = \lambda^2 - \lambda + 0 = \lambda(\lambda - 1)$$

which implies zero-stability. By the convergence theorem for linear multistep methods, it is sufficient so prove consistency. While this can be done using the conditions, we will prove it directly from the definitions. Since,

$$\Phi_h = y_{n+1} - y_n - \frac{h}{2}\left(3f_n - f_{n-1}\right) = 0$$

the local truncation error $\tau_h(t_n)$ is,

$$= y(t_{n+1}) - y(t_n) - \frac{h}{2}\left(3\underbrace{f(t_n, y(t_n))}_{=y'(t_n)} - \underbrace{f(t_{n-1}, y(t_{n-1}))}_{=y'(t_{n-1})}\right)$$

where $y(t)$ is the exact solution to the IVP. Applying Taylor's Theorem to y and y' around t_n gives that,

$$y(t_{n+1}) = y(t_n) + y'(t_n)h + \frac{y''(t_n)}{2}h^2 + \frac{y^{(3)}(\xi_n)}{3!}h^3$$

$$y'(t_{n-1}) = y'(t_n) - y''(t_n)h + \frac{y^{(3)}(\eta_n)}{2}h^2$$

for $\xi_n \in (t_n, t_{n+1})$ and $\nu_n \in (t_{n-1}, t_n)$. Thus,

$$\tau_h(t_n) = \frac{1}{12}\left(2y^{(3)}(\xi_n) + 3y^{(3)}(\eta_n)\right)h^3$$

Define a constant $M_3 = \max_{t\in[0,T]}\left|y^{(3)}(t)\right|$ so that,

$$|\tau_h(t_n)| \leq \frac{5M_3}{12}h^3$$

The consistency error is,

$$\frac{\tau_h}{h} = \max_{0\leq n\leq N}\frac{|\tau_h(t_n)|}{h} \leq \frac{5M_3}{12}h^2 = \mathcal{O}\left(h^2\right) \to 0 \text{ as } h \to 0$$

from which we conclude that Adam-Bashford is consistent of order 2 and zero-stable. This is sufficient for convergence.

Case Study: A-Stability

The zero-stability condition is useful in concluding that a consistent linear multi-step method converges in the limit as $h \to 0$. In practice, we want to work with a fixed non-zero h. For efficiency, we want to take h to be as large as possible. However, there are restrictions on h that arise based on the methods that is being used.

Consider the following problem. Fix $\lambda \in \mathbb{C}$ with $\text{Re}(\lambda) < 0$.

$$\begin{cases} y'(t) = \lambda y(t) \\ y(0) = y_0 \end{cases} \tag{2}$$

has $y(t) = y_0 \cdot e^{\lambda t}$ as an exact solution. We call the problem **stiff** if $\text{Re}(\lambda) \ll 0$. Observe that $y \to 0$ as $t \to \infty$, but numerical solutions do not always have this property. We will apply 1 step methods, beginning with Euler's method, to see why this is the case.

Example 37: Applying Euler's Method to (1)

Iterating the recurrence that we obtain by definition:

$$\begin{aligned} y_{n+1} &= y_n + h\lambda y_n \\ &= (1+h\lambda)y_n \\ &= (1+h\lambda)^2 y_{n-1} \\ &\cdots \\ &= (1+h\lambda)^{n+1} y_0 \end{aligned}$$

We require that $|1+h\lambda| < 1$ for $y_n \to 0$ as $n \to \infty$ when h is

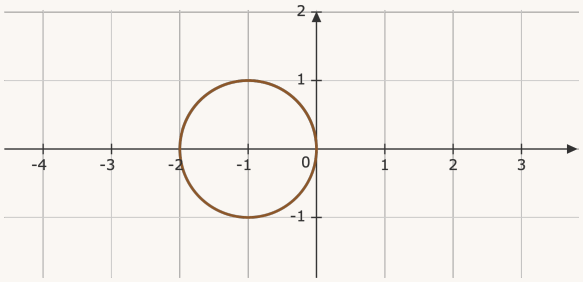
fixed. This condition is equivalent to the condition that,

$$(1+x)^2 + y^2 < 1$$

where $\lambda h = x + iy$. Observe that,

$$|1 + h\lambda| = |1 + x + iy| \leq |1 + x| + |iy| = (1+x)^2 + y^2$$

This region forms a circle in $\mathbb{R}^2$,



If $\lambda \in \mathbb{R}$, then we require

$$|1 + h\lambda| < 1 \iff -2 < h\lambda < 0 \iff 0 < h < -\frac{2}{\lambda}$$

so the stiffer the problem, the smaller h is required to be.

Example 38: Applying Backward Euler to (1)

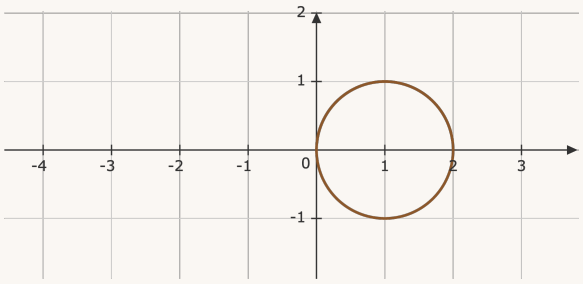
Iterating the recurrence that we obtain by definition:

$$\begin{aligned} y_{n+1} &= y_n + h\lambda y_{n+1} \\ &= \frac{y_n}{1 - h\lambda} \\ &\dots \\ &= \frac{y_0}{(1 - \lambda h)^{n+1}} \end{aligned}$$

For fixed h, we require the same condition,

$$|1 - h\lambda| > 1 \iff (1-x)^2 + y^2 > 1$$

to have $y_n \to 0$ as $n \to \infty$. This region forms a circle in $\mathbb{R}^2$,



If $\lambda \in \mathbb{R}$, then we require

$$|1 - h\lambda| > 1 \iff h\lambda < 0 \text{ or } h\lambda > 2$$

but $h\lambda < 0$ is satisfied since

$$\text{Re}(\lambda) < 0$$

so there is no restriction on h.

Example 39: Applying Trapezoidal to (1)

We obtain by definition that,

$$y_{n+1} = y_n + \frac{h}{2} \cdot (f_{n+1} + f_n)$$

where $f(t_n, y_n) = \lambda \cdot y_n$. Re-arranging and iterating,

$$y_{n+1} = \frac{y_n \left(1 + \frac{h}{2}\lambda\right)^{n+1}}{\left(1 - \frac{h}{2}\lambda\right)}$$

If $h\lambda$ satisfies the inequality,

$$1 + \frac{h\lambda}{2} < 1 - \frac{h\lambda}{2}$$

that is $h < 0$ then $y_n \to 0$ as $n \to \infty$. If $\lambda \in \mathbb{R}$, then $\text{Re}(\lambda) < 0$ is satisfied. Hence, there is no restriction on the size of h.

These examples on explicit and implicit 1 step methods motivate the following definition of A-**stability** for problem (1):

Definition (A-Stability). Let h be fixed. If y_n is the solution at t_n of a method applied to (1), then the region of A-**stability** is

$$R = \left\{ h\lambda \in \mathbb{C} \mid \lim_{n\to\infty} y_n \to 0, \text{ for a fixed } h \right\} \subseteq \mathbb{C}$$

Remark. An A-stable method is one with no restrictions on h.

Example 40: A-Stability

1. Euler's method is not A-stable
2. All explicit methods are not A-stable
3. Backward Euler and the Trapezoidal method are A-stable

For stiff problems, use an A-stable method or non-A stable method with a large region R in the left half of the complex plane.

Implicit methods tend to have larger regions of A-stability.

A-stability applies to equations other than (1). We begin with definitions. Consider a one-dimensional autonomous system,

$$y'(t) = f(y(t)) \tag{3}$$

An **equilibrium solution** of (2) is a solution $y^* \in \mathbb{R}$ such that $f(y^*) = 0$. We call y^* **asymptotically stable** if $f'(y^*) < 0$.

$$y' = f(y) = \underbrace{f(y^*)}_{=0} + f'(y^*)(y - y^*) + \frac{f''(\eta)}{2}(y - y^*)^2$$

for some η guaranteed by Taylor's remainder theorem. For $\tilde{y}(t) := y(t) - y^*$, we can approximate the IVP as,

$$\tilde{y}' \approx f'(y^*)\,\tilde{y}$$

which is the same form as (1) with $\lambda = f'(y^*)$. This last step can be justified by the **linearization theorem** from dynamical systems.

Remark. If y^* is asymptotically stable equilibrium, then A-stable methods will compute the correct y^* without restrictions to h. In contrast, a method which is not A-stable can lead to unbounded growh or oscillations near Y^* if h is not small enough.

Example 41: A-Stability

Suppose that a method is A-stable. We may be tempted to take h large enough to compute $y(T)$ with one time step.

1. The error is still $\mathcal{O}(h^p)$
2. Since all A-stable methods are implicit, root finding methods may not converge if h is too large

Predictor-Corrector Methods

While implicit methods have notable stability properites, one main drawback is the requirement that we use root finding methods. In

this section, we will explore **predictor-corrector** as a potential solution to this. The basic idea will be to use an explicit method to predict y_{n+1}, prior to correct y_{n+1} using the implicit method.

Definition (Improved Euler Method)**.** The **I-Euler method** is,

$$y_{n+1} = y_n + \frac{h}{2}\left(f\left(t_n, y_n\right) + f\left(t_{n+1}, \tilde{y}_{n+1}\right)\right)$$

which is obtained by using the Euler method to predict y_{n+1},

$$\tilde{y}_{n+1} = y_n + hf\left(t_n, y_n\right) \qquad \text{(predict)}$$

$$y_{n+1} = y_n + \tfrac{h}{2}\left(f\left(t_n, y_n\right) + f\left(t_{n+1}, \tilde{y}_{n+1}\right)\right) \qquad \text{(correct)}$$

in the original trapezoidal method,

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

Substituting $\tilde{y}_{n+1}$,

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

This is a 1 step method, so it suffices to show consistency in order to show convergence. It is not multistep, so we cannot use the order conditions. We will compute the consistency error.

$$\Phi_h = y_{n+1} - y_n - \frac{h}{2}\left(f\left(t_n, y_n\right) + f\left(t_{n+1}, y_n + hf\left(t_n, y_n\right)\right)\right)$$

Let y be the exact solution to the IVP. So the local truncation error is,

$$y\left(t_{n+1}\right) - y\left(t_n\right) - \frac{h}{2}\left(f\left(t_n, y\left(t_n\right)\right) + f\left(t_{n+1}, y\left(t_n\right) + hf\left(t_n, y\left(t_n\right)\right)\right)\right)$$

which we can simplify as follows,

$$y'\left(t_n\right)h + \frac{y''\left(t_n\right)}{2}h^2 + \mathcal{O}\left(h^3\right) - \frac{h}{2}\left(f\left(t_n, y\left(t_n\right)\right) + f\left(t_{n+1}, y\left(t_n\right) + hf\left(t_n, y\left(t_n\right)\right)\right)\right)$$

Since $y' = f(t, y)$, differentiating in t implies that,

$$y'' = f_t(t, y) + f_y(t, y)y' = f_t(t, y) + f_y(t, y)f(t, y)$$

Applying Taylor's Theorem to f about the point (t, y),

$$f(t + h, y + ch) = f(t, y) + f_t(t, y)h + f_y(t, y)ch + \mathcal{O}(h^2)$$

and setting $t = t_n$, $y = y(t_n)$, and $c = f(t_n, y(t_n))$ gives,

$$f\left(t_{n+1}, y\left(t_n\right) + hf\left(t_n, y\left(t_n\right)\right)\right)$$

which evaluates to,

$$f(t_n, y(t_n)) + f_t(t_n, y(t_n))\, h + f_y(t_n, y(t_n))\, f(t_n, y(t_n))\, h + \mathcal{O}(h^2)$$

Simplifying $\tau_h(t_n)$ using this expression tells us that $\tau_h(t_n) = \mathcal{O}(h^3)$, therefore, $\tau_h = \mathcal{O}(h^2)$. Hence, the method is of order 2.

Runge-Kutta Methods

Explicit 1 step methods are fast and easy to implement, but they have low order accuracy. One idea is to generalize to higher order and take fractional time steps that allow us to compute y_{n+1} in stages. This is the idea behind **Runge-Kutta (RK)** methods. The general form of an s-stage explicit RK method is,

$$\begin{aligned}
k_1 &= f(t_n + c_1 h, y_n) \\
k_2 &= f(t_n + c_2 h, y_n + h a_{2,1} k_1) \\
k_3 &= f(t_n + c_3 h, y_n + h(a_{3,1} k_1 + a_{3,2} k_2)) \\
&\vdots \\
k_s &= f(t_n + c_s h, y_n + h(a_{s,1} k_1 + a_{s,2} k_2 + \cdots + a_{s,s-1} k_{s-1})) \\
y_{n+1} &= y_n + h(b_1 k_1 + \cdots + b_s k_s)
\end{aligned}$$

The coefficients b_i are called **weights**, c_i are called **nodes**, k_i are stages, and the entries $a_{i,j}$ form the **RK matrix**.

> **Remark** (Butcher Tableau)**.** Runge-Kutta methods are specified by
>
> $$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^T \end{array}$$
>
> where $\mathbf{c}$ is the column vector of nodes, $\mathbf{b}^T$ is the row vector of weights, and A is the RK matrix.

> **Corollary.** A must be strictly lower triangular for the explicit RK method. Otherwise, the RK methods are implicit.

Example 42: Improved Euler as an RK Method

I-Euler method can be written as a 2 stage explicit RK method:

$$\begin{cases} k_1 = f(t_n, y_n) \\ k_2 = f(t_n + h, y_n + h k_1), \\ y_{n+1} = y_n + h\left(\frac{1}{2} k_1 + \frac{1}{2} k_2\right) \end{cases}$$

Arranged in the RK Tableau, this becomes,

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & 1/2 & 1/2 \end{array}$$

Example 43: 4 Stage Explicit Runge-Kutta Method

The most popular Runge-Kutta method is the following,

$$\begin{aligned} k_1 &= f(t_n, y_n) \\ k_2 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right) \\ k_3 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2\right) \\ k_4 &= f(t_n + h, y_n + hk_3) \\ y_{n+1} &= y_n + h\left(\frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4\right) \end{aligned}$$

Arranged in the RK Tableau, this becomes,

$$\begin{array}{c|cccc} 0 & & & & \\ 1/2 & 1 & & & \\ 1/2 & 0 & 1/2 & & \\ 1 & 0 & 0 & 1 & \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}$$

It can be shown that this method is of order 4.

RK4 is not A-stable, but both RK4 and I-Euler have larger regions of A-stability than previous explicit method, e.g., Euler and Adam-Bashford.

First-Order Systems of ODEs

We can generalize methods for solving first-order initial value problems to systems. Consider the system of first-order ODEs,

$$\begin{cases} \mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t)) \\ \mathbf{y}(0) = \mathbf{y}_0 \end{cases}$$

Euler's method can be written as,

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

and the Backward Euler method can be written as,

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

For implicit methods, root finding methods are required to solve the system. These include fixed point iteration and Newton's method.

Linear multistep methods can also be applied to first order systems, where consistency and zero-stability results will hold. For example, A-stability can be applied to an autonomous system,

$$y' = f(y)$$

where near an equilibrium solution $\mathbf{y}^*$,

$$\mathbf{y}' \approx J_F\left(\mathbf{y}^*\right)\left(\mathbf{y} - \mathbf{y}^*\right)$$

Since the Jacobian $J_F\left(\mathbf{y}^*\right)$ is a matrix, the stiffness of the system will be determined by the eigenvalues of $J_F\left(\mathbf{y}^*\right)$ with negative real components. Predictor-corrector and Runge-Kutta methods also generalize. For instance, the I-Euler method can be generalized as,

$$\begin{cases} \mathbf{k}_1 = \mathbf{f}\left(t_n, \mathbf{y}_n\right) \\ \mathbf{k}_2 = \mathbf{f}\left(t_n + h, \mathbf{y}_n + h\mathbf{k}_1\right) \\ \mathbf{y}_{n+1} = \mathbf{y}_n + h\left(\frac{1}{2}\mathbf{k}_1 + \frac{1}{2}\mathbf{k}_2\right) \end{cases}$$

The next example will show how we can write a higher-order IVP as a first-order system by introducing the appropriate variables.

Example 44: Solving Higher-Order IVPs

Consider the second-order IVP,

$$\begin{cases} x''(t) = f\left(t, x(t), x'(t)\right) \\ x(0) = a \\ x'(0) = b \end{cases}$$

Define the following variables,

$$\begin{aligned} &y := x' \\ &x'' = f\left(t, x, x'\right) \iff y' = f(t, x, y) \end{aligned}$$

Hence, the second-order IVP can be written as,

$$\mathbf{x}'(t) := \begin{pmatrix} x'(t) \\ y'(t) \end{pmatrix} = \begin{pmatrix} y(t) \\ f(t, x(t), y(t)) \end{pmatrix} =: \mathbf{f}(t, \mathbf{x}(t))$$

$$\mathbf{x}(0) := \begin{pmatrix} x(0) \\ y(0) \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix} =: \mathbf{x}_0$$