

Lecture 01

§1.1, 1.2 Taylor Series and Convergence

T. Takahashi

Department of Mathematics and Statistics
University of New Mexico



We will answer these questions:

- How are quantities such as $\ln(2)$, $\sqrt{7}$, and e^3 calculated on the computer?
- How much error can we expect when using Taylor series approximations?
- What is “convergence rate” for a numerical algorithm?
- How can we differentiate algebraic convergence versus geometric convergence?



Math on the Computer



A Brief History of Computation

- Before 1950s: CPUs only perform integer arithmetic
- Floating point operations were slow, expensive, and needed *software*
- 1960s: **math coprocessors** introduced a separate chip for floating point operations called **floating-point unit (FPU)**
- 1980s: CPUs all have an integrated FPU

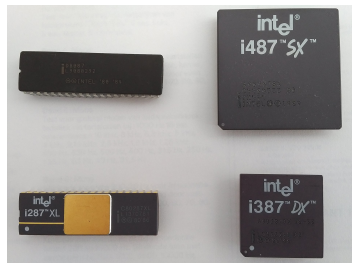


Figure: Early floating-point coprocessor

Key Idea

Computers can handle calculations like $4 + 2.1$ or 2.1^2 at the assembly level.



What Can a Computer Actually Do?

- A CPU (with an FPU) can perform:

$+$, $-$, $\times$, $\div$

- These operations are implemented directly in hardware
- It can also handle:
 - floating point numbers
 - basic arithmetic efficiently



What It Cannot Do

- The CPU does **not** have built-in formulas for:

$$\sin(x), \quad e^x, \quad \ln(x)$$

- These functions are called **transcendental functions** - cannot be expressed using a finite combination of:

$$+, \quad -, \quad \times, \quad \div, \quad \text{powers}$$

- These are not computed directly in hardware - **need software**



What It Cannot Do

- The CPU does **not** have built-in formulas for:

$$\sin(x), \quad e^x, \quad \ln(x)$$

- These functions are called **transcendental functions** - cannot be expressed using a finite combination of:

$$+, \quad -, \quad \times, \quad \div, \quad \text{powers}$$

- These are not computed directly in hardware - **need software**

Question

So how does your calculator compute $\sin(0.5)$?



Taylor Series Approximation



Polynomial Functions

Definition: Polynomials

A **polynomial** is a mathematical expression consisting of variables raised to non-negative integer powers and multiplied by **coefficients** (which are constants). These terms are then combined using addition and subtraction.

- Polynomial $P(x)$ of degree n has the form

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0,$$

where $a_n, a_{n-1}, \dots, a_1, a_0$ are constants, and x is the variable

- The **degree** n is the highest power and they are given specific names

Linear $p_1(x) = a_1 x + a_0$

Quadratic $p_2(x) = a_2 x^2 + a_1 x + a_0$

Cubic $p_3(x) = a_3 x^3 + a_2 x^2 + a_1 x + a_0$



Taylor Series

Question

If CPUs can only do basic arithmetic, then how do they calculate transcendentals: e^x , $\cos(x)$, $\sqrt{x}$?

Taylor Series

The Taylor series expansion of $f(x)$ at the point $x = c$ is given by

$$\begin{aligned} f(x) &= f(c) + (x - c)f'(c) + \frac{(x - c)^2}{2!}f''(c) + \cdots + \frac{(x - c)^n}{n!}f^{(n)}(c) + \cdots \\ &= \sum_{k=0}^{\infty} \frac{(x - c)^k}{k!}f^{(k)}(c) \end{aligned}$$



Taylor Example

Example (e^x)

Find the Taylor series expansion of $y = e^x$ about $c = 0$.

- All derivatives of $y = e^x$ is simply e^x
- Thus $\frac{(x-c)^n}{n!} f^{(n)}(c) = \frac{(x)^n}{n!} e^0$
- Using the Taylor series expansion:

$$\begin{aligned} e^x &= 1 + (x-0) \cdot 1 + \frac{(x-0)^2}{2} \cdot 1 + \dots \\ &= 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \dots \end{aligned}$$



Truncated Taylor Series

Question

Can you add an infinite series on the computer?

- No, you can't. You have to *truncate*.
- **Truncate** means cutting off the series at some finite integer n
- Example with $n = 3$

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \dots$$

truncate ↓

$$e^x \approx \underbrace{1 + x + \frac{x^2}{2!} + \frac{x^3}{3!}}_{\text{composed of only } +, \times, \div}$$

Key Idea

Computing a transcendental function like e^x can be done entirely with basic arithmetic because they can be approximated with polynomials.

Taylor Approximation

Implementation

Example

Approximate e^2 with Taylor series using $c = 0$.

- Using 0^{th} order Taylor series: $e^2 \approx 1$ does not give a good fit.
- Using 1^{st} order Taylor series: $e^2 \approx 1 + 2$ gives a better fit.
- Using 2^{nd} order Taylor series: $e^2 \approx 1 + 2 + 2^2/2$ gives an ever better fit

```
1 x=2;  
2 pn=0;  
3 for j=0:15  
4     pn = pn + (x^j)/factorial(j);  
5     err = exp(2)-pn  
6     pause  
7 end
```



Our First Numerical Method

Method: Taylor Approximation

- **Given:** Smooth function $f(x)$, cut off integer n , center of approximation c
 - **Output:** n^{th} degree polynomial approximation $p_n(x) \approx f(x)$
 - **Idea:** Transcendental functions are approximated with polynomials which can be computed entirely with additions and multiplications.
-
- Numerical Methods will be shown in a green text box
 - The integer n is called a **parameter** of the method
 - As n increases,
 - Approximation improves
 - Calculation cost increases



Our First Numerical Method

Method: Taylor Approximation

- **Given:** Smooth function $f(x)$, cut off integer n , center of approximation c
 - **Output:** n^{th} degree polynomial approximation $p_n(x) \approx f(x)$
 - **Idea:** Transcendental functions are approximated with polynomials which can be computed entirely with additions and multiplications.
-
- Numerical Methods will be shown in a green text box
 - The integer n is called a **parameter** of the method
 - As n increases,
 - Approximation improves
 - Calculation cost increases

Question

How do we choose n ?

Error Estimates



Exact vs. Approximation

Taylor Series Expansion (exact)

$$f(x) = f(c) + (x - c)f'(c) + \frac{(x - c)^2}{2!}f''(c) + \cdots + \frac{(x - c)^n}{n!}f^{(n)}(c) + \cdots$$

↕ same

Taylor Series with Remainder (exact)

$$f(x) = f(c) + (x - c)f'(c) + \frac{(x - c)^2}{2!}f''(c) + \cdots + \frac{(x - c)^n}{n!}f^{(n)}(c) + \frac{(x - c)^{n+1}}{(n + 1)!}f^{(n+1)}(\xi),$$

where ξ is an unknown number between x and c .

↓ approx.

Taylor Series Approximation (Truncated)

$$f(x) \approx p_n(x) = f(c) + (x - c)f'(c) + \frac{(x - c)^2}{2!}f''(c) + \cdots + \frac{(x - c)^n}{n!}f^{(n)}(c)$$



Definition: Error

Suppose $f(x_0)$ is approximated with the quantity $p_n(x_0)$. Then the **absolute error** (or **error**) is defined to be

$$\begin{aligned} E_n &= |\text{true} - \text{approx.}| \\ &= |f(x_0) - p_n(x_0)| \end{aligned}$$

- In real life, we don't know the true value so we must find an **error estimate** M satisfying

$$E = |f(x_0) - p_n(x_0)| < E_{max}$$

- Mathematicians calculate M by considering the *worst-case scenario* for the approximation



Taylor Approximation Error

- Consider the following approximation with $n = 2$:

$$e^x = \underbrace{1 + x + \frac{x^2}{2!}}_{p_2 \text{ approximation}} + \underbrace{\frac{x^3}{3!} + \dots}_{R_3 \text{ remainder}} \quad (1)$$

- The error would be

$$E_2 = |e^x - p_2(x)| = |R_3(x)|$$

Taylor's Inequality (Recall from Calculus II)

If we have a Taylor expansion of the form

$$f(x) = p_n(x) + R_n(x)$$

then we have the following error estimate

$$E_n(x) = |R_n(x)| = \left| \frac{(x-c)^{n+1}}{(n+1)!} f^{(n+1)}(\xi) \right| \leq \left| \frac{M(x-c)^{n+1}}{(n+1)!} \right|,$$

where M is a bound $|f^{(n+1)}(x)| \leq M$ for all x .

Error vs. Estimate vs. Tolerance

- **Error** E_n = the precise error that cannot be calculated
- **Error Estimate** $E_{n,max}$ = upper bound error which *can* be calculated
- **User Tolerance** δ = is a threshold for error that the user is okay with

Strategy

- Approximate $E_{n,max}$
- Set up inequality $E_{n,max} < \delta$
- Find parameters to make inequality true



Using the Taylor Inequality

Part 1

Example

How many Taylor terms are required to approximate e^2 with an error less than 2×10^{-8} ?

- We know

$$f^{(n)}(x) = e^x$$

- According to the Taylor inequality

$$E_n = |f(x) - p_n(x)| = \left| \frac{(x - c)^{n+1}}{(n + 1)!} f^{(n+1)}(\xi) \right|$$

- Let $c = 0$ and $x = 2$

$$E_n = \left| \frac{2^{n+1} e^{\xi}}{(n + 1)!} \right|$$

- ξ is unknown, but we do know its range $\xi \in [0, 2]$



Using the Taylor Inequality

Part 2

$$E_n = \left| \frac{2^{n+1}e^\xi}{(n+1)!} \right|$$

- What is the worst case scenario for the error if $\xi \in [0, 2]$?

$$e^\xi \leq e^2 =: M$$

- We have found our error estimate

$$E_n \leq \left| \frac{2^{n+1}e^\xi}{(n+1)!} \right| \leq \frac{2^{n+1}e^2}{(n+1)!} =: E_{n,max}$$

- Next, force the error estimate to be smaller than the threshold

$$\begin{aligned} E_{n,max} &\leq \delta \\ \frac{2^{n+1}e^2}{(n+1)!} &\leq 2 \times 10^{-8} \end{aligned}$$



Using the Taylor Inequality

Part 3

- Now to find the smallest integer n such that

$$\frac{2^{n+1}e^2}{(n+1)!} < 2 \times 10^{-8}$$

- By brute force... $n = 16$

$$\frac{2^{16+1}e^2}{(16+1)!} \approx 2.31 \times 10^{-9}$$

- Answer:** $n = 16$ is enough to guaranty an error less than 2×10^{-8}

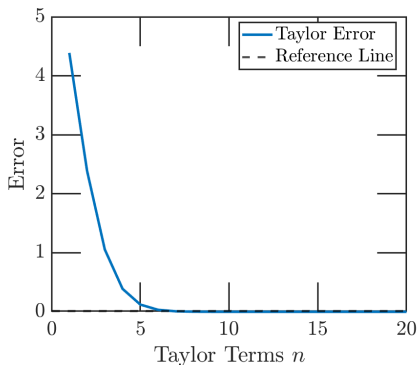
Choosing n

Choosing n for the Taylor approximation depends on the desired tolerance level and the function $f(x)$.



Taylor Error Plot

Regular Plot



- The Error goes down *too quickly*
- Can't see what is happening on a regular linear plot
- This is NOT a good plot

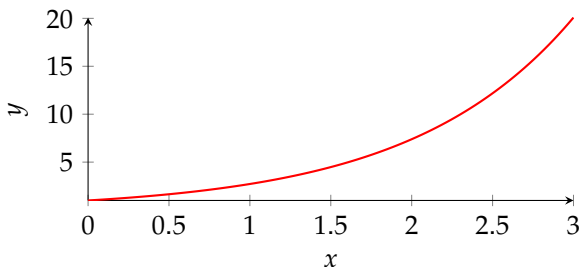


Log Plots



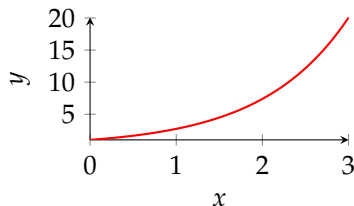
Four Scales

- Consider the function $y = e^x$ for $x > 0$
- There are 4 different scales to plot this function
 - x linear, y linear
 - x logarithm, y liner
 - x linear, y logarithm
 - x logarithm, y logarithm
- This is $y = e^x$ on a standard Linear-Linear scale

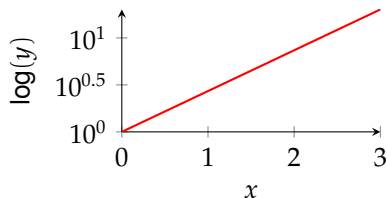


Effect of Logarithmic Scales

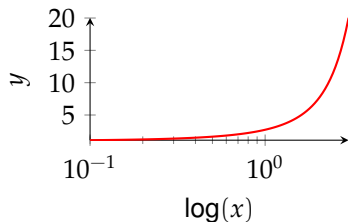
Linear-Linear



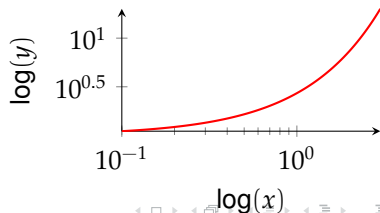
Linear-Log



Linear-Log



Log-Log



Choosing the Right Plot

- Suppose we have data (x_i, y_i)
- The choice of plot depends on how the data **scales**

Guidelines

- If y spans multiple orders of 10 $\Rightarrow$ use log scale for y
- If x spans multiple orders of 10 $\Rightarrow$ use log scale for x
- If both x and y span many orders of magnitude $\Rightarrow$ use **log-log scale**

Key Idea

Choose the scale based on how the data grows.



Example

Example

Consider the following data:

x	2	4	6	8	10	12	14
y	2	67	100	1,003	20,008	5,600	480

What is the most appropriate way to plot this data?



Example

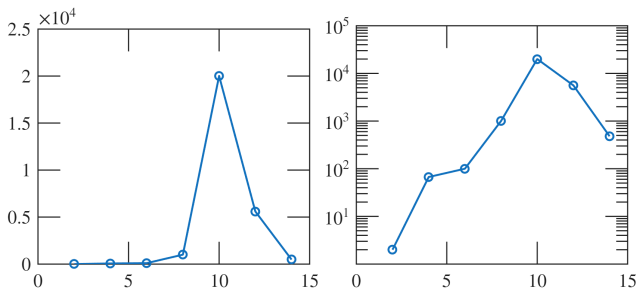
Example

Consider the following data:

x	2	4	6	8	10	12	14
y	2	67	100	1,003	20,008	5,600	480

What is the most appropriate way to plot this data?

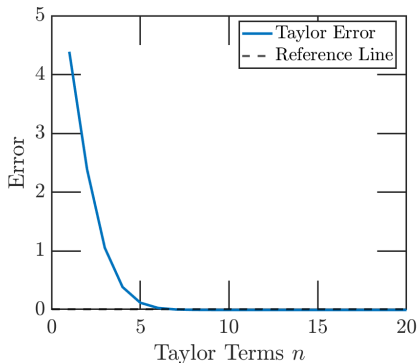
Since the y data ranges from 10^0 to 10^5 , we use log-linear



Revisit Taylor Error Plot

Example

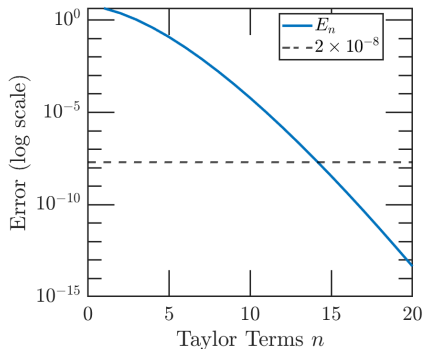
How many Taylor terms are required to approximate e^2 with an error less than 2×10^{-8} ?



Data is plotted on a linear-linear scale



A Better Plot



- This is the same data on a **log-linear** scale
- Clearly see that $n = 15$ is the threshold

Key Idea

Log scale reveals dynamics that span over *multiple orders of magnitude*.

Big O Notation



Definition: Big-O

Let $g(n)$ be a function of n . Then define

$$\mathcal{O}(g(n)) = \{f(n) \mid \exists c, n_0 > 0 \text{ such that } \forall n \geq n_0, 0 \leq f(n) \leq cg(n)\}$$

- $f(n) \in \mathcal{O}(g(n))$ represents an asymptotic upper bound on $f(n)$ up to a constant
- Example: $f(n) = 3\sqrt{n} + 2\log n + 8n + 85n^2 \in \mathcal{O}(n^2)$
- Example: If n is the size of an array, bubble sort's cost is $\mathcal{O}(n^2)$
- Example: Merge sort's cost is $\mathcal{O}(n \log n)$



Cost Scaling

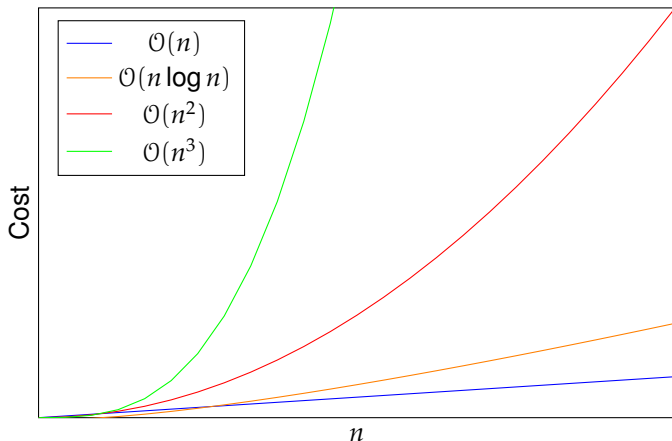
- Big O is used to describe algorithmic cost with respect to n
- What is n ?
 - size of an array
 - size of a matrix
 - number of iterations
- Big O notation tells us how well the algorithm cost scales
 - Ex 1: $\mathcal{O}(n^3)$ - scales very poorly
 - Ex 2: $\mathcal{O}(n)$ - scales efficiently

Key Idea

In terms of cost, it is better if $f(n) = \mathcal{O}(g(n))$ grows slower



Algorithm Cost



- The more slowly the curve rises, the better the algorithm
- $\mathcal{O}(n)$ cost is very cheap
- $\mathcal{O}(n^3)$ cost is very expensive



Cost Scaling Example

Example

Suppose a sorting algorithm costs $\mathcal{O}(n^2)$ to sort an array of length n . If it takes 2 minutes to sort an array of length k , how long will it take to sort an array of length $2k$?

- We let the cost of sorting k elements be

$$T_1 \approx Ck^2$$

- Then the cost of sorting $2k$ elements will be

$$T_2 \approx C(2k)^2 = 4Ck^2$$

- Then the ratio of T_2 to T_1 is

$$T_2/T_1 = 4Ck^2/Ck^2 = 4$$

so the runtime will be *four times* longer, or

about 8 minutes



Convergence



Sequence

Definition: Sequence

A **sequence** is an ordered list of numbers

$$a_1, a_2, a_3, \dots$$

usually generated by some rule.

- Example: $a_n = \frac{1}{2^n}$

$$\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \dots$$

Definition: Convergence

A sequence **converges** if its terms approach a limiting value.

- Notice that

$$\lim_{n \rightarrow \infty} \frac{1}{2^n} = 0$$



Error Sequence

- Suppose we want to compute certain values, e.g. $u = e^2$
- You use an iterative algorithm to approximate u

$$u_1, u_2, u_3, \dots,$$

e.g.

$$u_n = 1 + 2 + \frac{2^2}{2!} + \dots + \frac{2^n}{n!}$$

Definition: Error Sequence

The error sequence for an iterative algorithm is

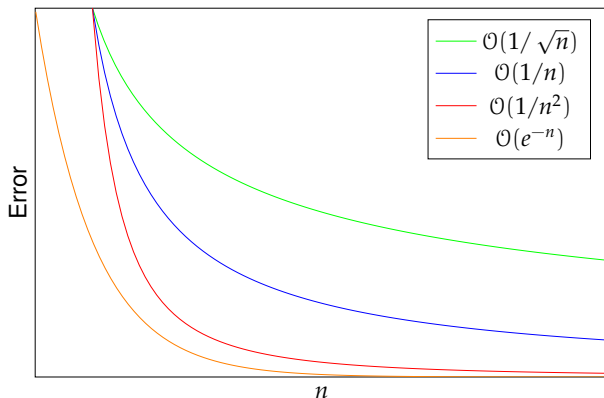
$$e_n = |u - u_n|$$

Key Idea

A successful algorithm should satisfy

$$e_n \rightarrow 0.$$

Algorithm Error



- The error sequence can converge to zero at various speeds
- We use Big O to describe the convergence rate
- The faster the curve plummets to zero, the better the algorithm



Definition: Algebraic Convergence

The **Algebraic Index of Convergence** α is the largest number for which

$$\lim_{n \rightarrow \infty} |e_n| n^\alpha < \infty$$

where e_n is the sequence of error. Alternatively, α is the algebraic index if

$$e_n \sim \mathcal{O}(1/n^\alpha)$$

We say that the error is decaying with **algebraic convergence**.



Exponential Convergence (J. P. Boyd)

- There are cases when error e_n decreases faster than algebraically

Definition: Exponential Convergence

If the algebraic index α is unbounded (i.e. a_n decrease faster than $1/n^\alpha$ for any finite α), then the sequence converges **exponentially** (a.k.a. **spectrally**).

Alternatively, the sequence converges exponentially if for constants q and β

$$e_n \sim \mathcal{O}(e^{-qn^\beta})$$

where β is the **exponential index of convergence** and

$$\beta = \lim_{n \rightarrow \infty} \frac{\log |\log(|e_n|)|}{\log(qn)}$$



Rates of Exponential Convergence (J. P. Boyd)

- Exponential convergence can be broken into 3 different categories

Definition:

A sequence a_n has **supergeometric**, **geometric**, or **subgeometric** if

$$\lim_{n \rightarrow \infty} \log(|e_n|)/n = \begin{cases} \infty, & \text{supergeometric} \\ \text{constant}, & \text{geometric} \\ 0, & \text{subgeometric} \end{cases}$$

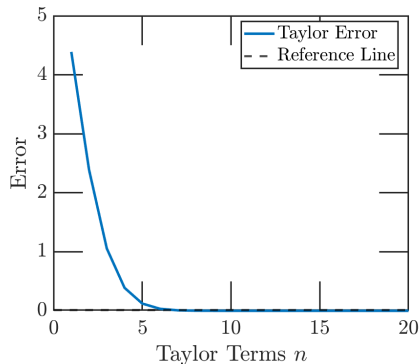
or, alternatively,

$$e_n \sim \mathcal{O}(e^{-(n/j) \log(n)}) : \text{supergeometric}$$

$$e_n \sim \mathcal{O}(e^{-qn}) : \text{geometric}$$

$$\beta < 1 : \text{subgeometric}$$

How can we tell?



- We can plot e_n versus n and clearly see that errors goes down
- We cannot tell by eye how quickly it goes down

Question

How can we distinguish if the error is going down algebraically or exponentially?

How to identify Algebraic Convergence

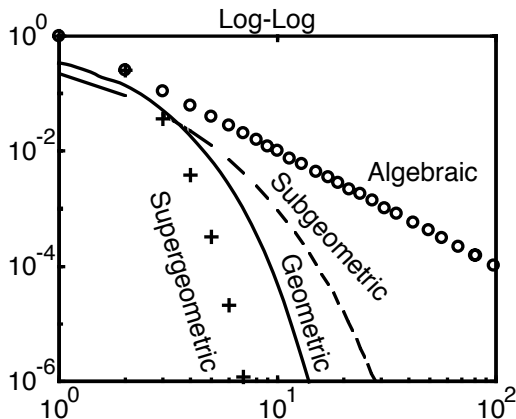


Figure 2.6: Same as previous figure except that the graph is log-log: the degree of the spectral coefficient n is now plotted on a logarithmic scale, too.

Error vs. n will show up as a straight line on a **log-log plot**.



Proof for Algebraic Convergence

- Suppose the error converges algebraically $E = \mathcal{O}(n^{-\alpha})$
- For large n we have

$$E = Cn^{-\alpha}$$

- Take log both sides

$$\log(E) = \log(Cn^{-\alpha})$$

- Using the properties of logarithms

$$\log(E) = \log(Cn^{-\alpha})$$

$$\log(E) = \log(n^{-\alpha}) + \log(C)$$

$$\log(E) = -\alpha \log(n) + \log(C)$$

Key Idea

- ▶ If E is algebraic then $\log(E)$ vs. $\log(n)$ is a straight line on a **log-log plot**.
- ▶ The slope reveals the algebraic index α .

How to Identify Geometric Convergence

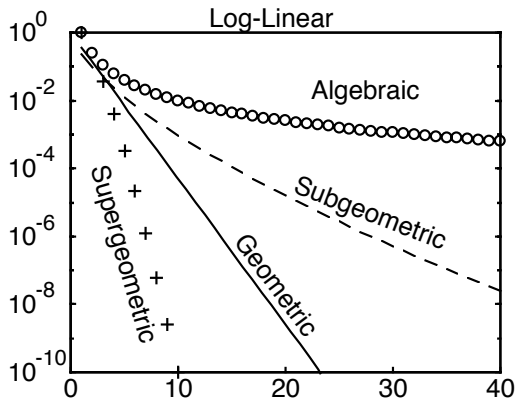


Figure 2.5: $\log |a_n|$ versus n for four rates of convergence. Circles: algebraic convergence, such as $a_n \sim 1/n^2$. Dashed: subgeometric convergence, such as $a_n \sim \exp(-1.5 n^{2/3})$. Solid: geometric convergence, such as $\exp(-\mu n)$ for any positive μ . Pluses: supergeometric, such as $a_n \sim \exp(-n \log(n))$ or faster decay.

Error vs. n will show up as a straight line on a **log-linear plot**.



Proof for Geometric Convergence

- Suppose the error converges geometrically $E = \mathcal{O}(e^{-qn})$
- For large n we have

$$E = Cn^{-n}$$

- Take log both sides

$$\log(E) = \log(Cn^{-qn})$$

- Using the properties of logarithms

$$\log(E) = \log(Ce^{-qn})$$

$$\log(E) = \log(e^{-qn}) + \log(C)$$

$$\log(E) = -qn \log(e) + \log(C)$$

$$\log(E) = -qn + \log(C)$$

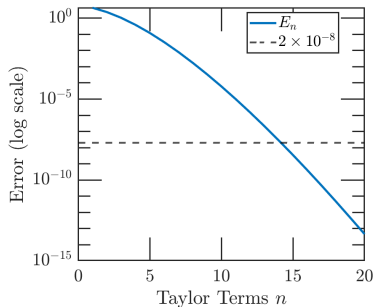
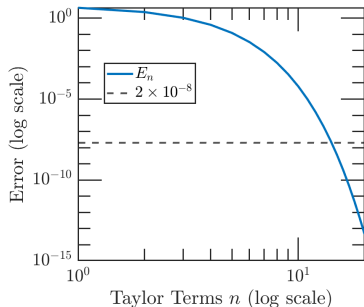
Key Idea

If E is geometric then $\log(E)$ vs. n is a straight line on a **log-linear plot**.

Taylor Error Convergence

Example

Based on the plotted data, determine how the convergence rate of Taylor approximations.



- Algebraic ✗ - not straight on the log-log plot
- Geometric ✗ - not straight on the log-linear plot
- Supergeometric ✓

