Computers, Integrals, and Quadrature

Naqeeb Ali

UTM Math Club

March 26, 2026

Differentiation vs. Integration

	Differentiation	**Integration**
Requires	f differentiable	f integrable (weaker)
Regularity	f' can be worse than f	$\int f$ is smoother than f
Closed form	Algorithmic (chain rule)	Often impossible
Numerically	Amplifies noise	Smooths noise

Integration tends to be more well-conditioned than differentiation.

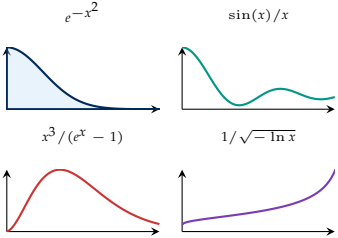
But We Still Need Algorithms

Most integrals in practice have no closed-form antiderivative.

But We Still Need Algorithms

Most integrals in practice have no closed-form antiderivative.

$$\int_0^1 e^{-x^2}\,dx \qquad \int_0^1 \frac{\sin x}{x}\,dx$$

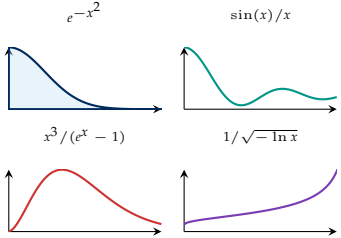


But We Still Need Algorithms

Most integrals in practice have no closed-form antiderivative.

$$\int_0^1 e^{-x^2}\,dx \qquad \int_0^1 \frac{\sin x}{x}\,dx$$

$$\int_0^{\pi/2} \sqrt{1 - k^2 \sin^2\theta}\,d\theta \quad \text{(elliptic)}$$



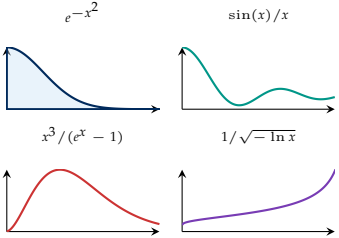
But We Still Need Algorithms

Most integrals in practice have no closed-form antiderivative.

$$\int_0^1 e^{-x^2}\,dx \qquad \int_0^1 \frac{\sin x}{x}\,dx$$

$$\int_0^{\pi/2} \sqrt{1 - k^2 \sin^2\theta}\,d\theta \quad \text{(elliptic)}$$

$$\int_0^{\infty} \frac{x^3}{e^x - 1}\,dx = \frac{\pi^4}{15} \quad \text{(Planck)}$$



But We Still Need Algorithms

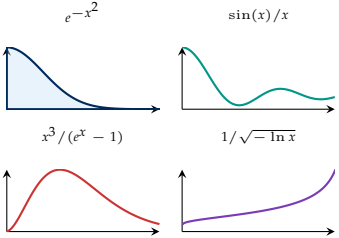
Most integrals in practice have no closed-form antiderivative.

$$\int_0^1 e^{-x^2}\,dx \qquad \int_0^1 \frac{\sin x}{x}\,dx$$

$$\int_0^{\pi/2} \sqrt{1 - k^2 \sin^2\theta}\,d\theta \quad \text{(elliptic)}$$

$$\int_0^{\infty} \frac{x^3}{e^x - 1}\,dx = \frac{\pi^4}{15} \quad \text{(Planck)}$$

$$\int_0^1 \frac{dx}{\sqrt{-\ln x}} = \sqrt{\pi}$$



But We Still Need Algorithms

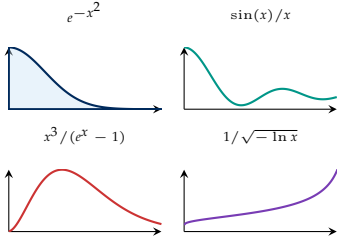
Most integrals in practice have no closed-form antiderivative.

$$\int_0^1 e^{-x^2}\,dx \qquad \int_0^1 \frac{\sin x}{x}\,dx$$

$$\int_0^{\pi/2} \sqrt{1 - k^2 \sin^2\theta}\,d\theta \quad \text{(elliptic)}$$

$$\int_0^{\infty} \frac{x^3}{e^x - 1}\,dx = \frac{\pi^4}{15} \quad \text{(Planck)}$$

$$\int_0^1 \frac{dx}{\sqrt{-\ln x}} = \sqrt{\pi}$$



These are encountered a lot in math, physics, statistics, and engineering. We need systematic numerical methods to numerically evaluate them.

The Abstract Problem

Definition 1 (Quadrature Rule)

A *quadrature rule* for approximating $\int_a^b f(x)\,dx$ is a pair of **nodes** $x_1, \ldots, x_n \in [a, b]$ and **weights** $w_1, \ldots, w_n \in \mathbb{R}$, giving

$$Q_n[f] \;=\; \sum_{i=1}^{n} w_i f(x_i) \;\approx\; \int_a^b f(x)\,dx.$$

The Abstract Problem

Definition 1 (Quadrature Rule)

A *quadrature rule* for approximating $\int_a^b f(x)\,dx$ is a pair of **nodes** $x_1, \ldots, x_n \in [a, b]$ and **weights** $w_1, \ldots, w_n \in \mathbb{R}$, giving

$$Q_n[f] \;=\; \sum_{i=1}^{n} w_i f(x_i) \;\approx\; \int_a^b f(x)\,dx.$$

Two design questions that determine everything:

The Abstract Problem

Definition 1 (Quadrature Rule)

A *quadrature rule* for approximating $\int_a^b f(x)\,dx$ is a pair of **nodes** $x_1, \ldots, x_n \in [a, b]$ and **weights** $w_1, \ldots, w_n \in \mathbb{R}$, giving

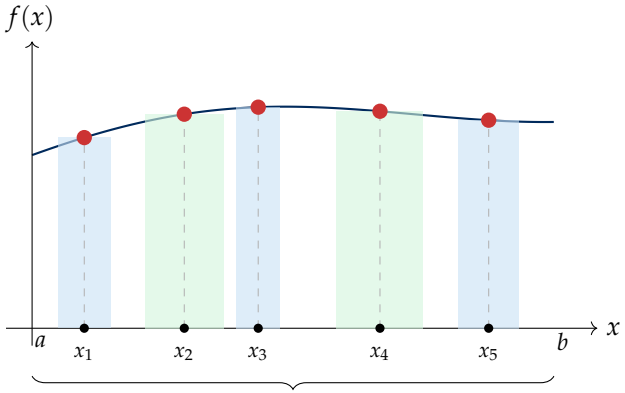
$$Q_n[f] \;=\; \sum_{i=1}^{n} w_i f(x_i) \;\approx\; \int_a^b f(x)\,dx.$$

Two design questions that determine everything:

Where to place nodes x_i ?

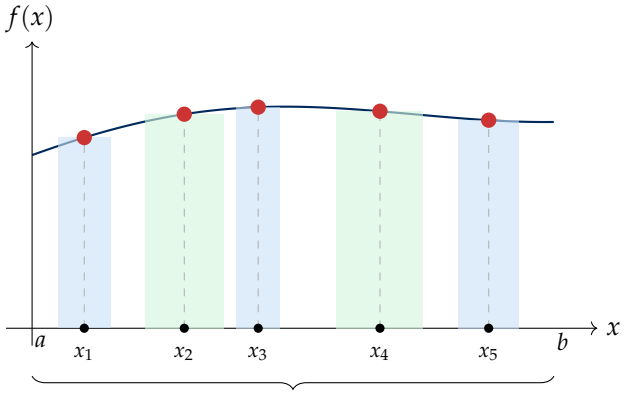
What weights w_i to use?

The Abstract Problem Visualised



Nodes at non-uniform spacing. We have a choice in nodes and weights!

The Abstract Problem Visualised



Nodes at non-uniform spacing. We have a choice in nodes and weights!

Each coloured rectangle has area $\approx w_i \cdot f(x_i)$. The sum of areas approximates $\int_a^b f(x)\,dx$.

Roadmap

1. **Why Numerical Integration?**
2. First Attempts — Riemann, Trapezoidal, Simpson's, Romberg
3. Gaussian Quadrature
4. Generalized Gaussian Quadrature: When Smoothness Breaks
5. Solving Laplace's Equation

Part 2

First Attempts at Quadrature

Riemann Sums $\rightarrow$ Trapezoidal $\rightarrow$ Simpson's

Riemann Sums

Choose a uniform partition $a = x_0 < x_1 < \cdots < x_n = b$ with n equal subintervals.

From the point of view of quadrature, the nodes x_i are equally spaced, and the weights w_i are all equal to $(b-a)/n$.

The three rules differ in *where within each subinterval* we sample f:

$$L_n = \frac{b-a}{n}\sum_{i=0}^{n-1} f(x_i), \qquad R_n = \frac{b-a}{n}\sum_{i=1}^{n} f(x_i), \qquad M_n = \frac{b-a}{n}\sum_{i=0}^{n-1} f\left(\frac{x_i + x_{i+1}}{2}\right)$$

Rule	Error (smooth f)	Why?
Left / Right	$O(n^{-1})$	One-sided evaluation
Midpoint	$O(n^{-2})$	Symmetry cancels linear error

$O(n^{-1})$: doubling n halves the error. $O(n^{-2})$: doubling n cuts the error by $\sim 4\times$. Much better!

Why Is Midpoint Better?

Focus on one panel $[x_i,\ x_{i+1}]$ of width $h = (b-a)/n$, with midpoint m_i. Substitute $u = x - m_i$:

$$\int_{x_i}^{x_{i+1}} f(x)\,dx \;=\; \int_{-h/2}^{h/2} f(m_i+u)\,du$$

Now Taylor-expand $f(m_i+u) = f(m_i) + f'(m_i)\,u + \frac{1}{2}f''(m_i)\,u^2 + \cdots$ and integrate term by term:

$$\int_{-h/2}^{h/2} f(m_i)\,du = hf(m_i)$$

$$f'(m_i)\int_{-h/2}^{h/2} u\,du = 0 \qquad \text{(odd function, symmetric interval)}$$

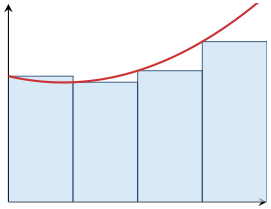
$$\tfrac{1}{2}f''(m_i)\int_{-h/2}^{h/2} u^2\,du = \frac{h^3}{24}f''(m_i)$$

So the per-panel error is $\dfrac{h^3}{24}f''(m_i) + O(h^5)$. Summing n panels gives total error $= O(n\cdot h^3) = O(n^{-2})$.

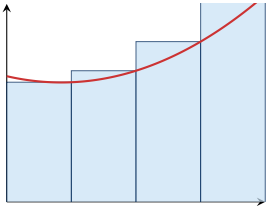
For left/right Riemann, the f' term does not cancel $\implies O(n^{-1})$.

Geometrically:

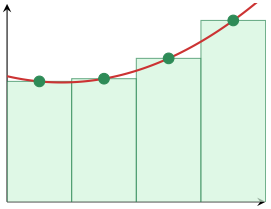
Left Riemann



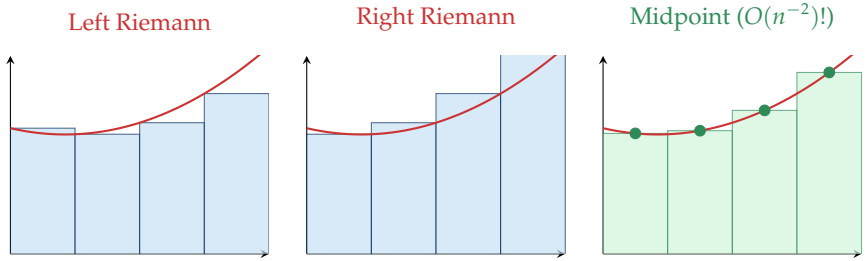
Right Riemann



Midpoint ($O(n^{-2})!$)



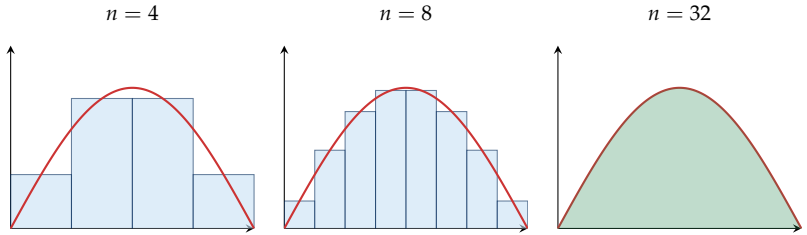
Geometrically:



Left/Right overshoot or undershoot systematically.
Midpoint benefits from error cancellation by symmetry around each subinterval centre.

What Does $O(n^{-2})$ Look Like?

Midpoint rule for $\int_0^{\pi} \sin x \, dx = 2$




```
>>> midpoint(sin, 0, pi, n)
n = 4: Q = 2.0523443060 error = 5.23e-02
n = 8: Q = 2.0129090856 error = 1.29e-02
n =16: Q = 2.0032163782 error = 3.22e-03
n =32: Q = 2.0008034163 error = 8.03e-04
```

Each doubling of n cuts error by $\sim 4\times$ $\implies$ exactly the $O(n^{-2})$ convergence.

From Riemann Sums to the Trapezoidal Rule

Can we do better than $O(n^{-2})$? Two natural ideas lead to the same rule.

Idea 1: Interpolate better.

Riemann sums approximate f as a constant on each panel (degree 0).
What if we fit a line through the two endpoints instead (degree 1)?
Integrating that line gives a trapezoid.

Idea 2: Combine and cancel.

Left endpoint overshoots, right endpoint undershoots (or vice versa).
Their errors have opposite signs:

$$L_n = I + \frac{c}{n} + O(n^{-2}), \qquad R_n = I - \frac{c}{n} + O(n^{-2})$$

Average them:

$$T_n \;=\; \frac{L_n + R_n}{2} \;=\; I + O(n^{-2}).$$

Averaging rectangles is the same as fitting a line.

The Trapezoidal Rule

With n panels, connect successive points by straight lines and integrate the trapezoids:

The Trapezoidal Rule

With n panels, connect successive points by straight lines and integrate the trapezoids:

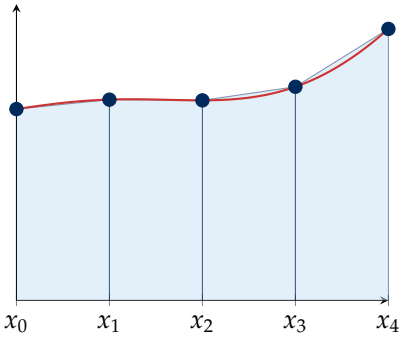
$$\boxed{T_n = \frac{b-a}{2n}\left[f(x_0) + 2f(x_1) + \cdots + 2f(x_{n-1}) + f(x_n)\right]}$$

The Trapezoidal Rule

With n panels, connect successive points by straight lines and integrate the trapezoids:

$$T_n = \frac{b-a}{2n}\left[f(x_0) + 2f(x_1) + \cdots + 2f(x_{n-1}) + f(x_n)\right]$$

Error: $O(n^{-2})$, same as midpoint.



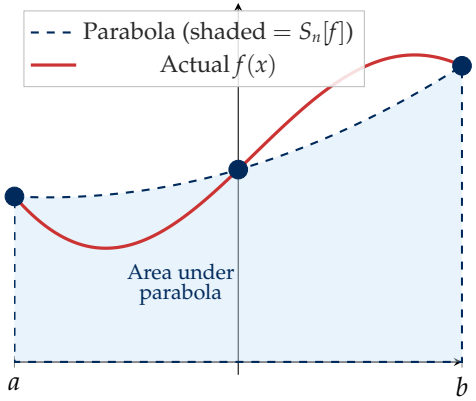
Simpson's Rule

Given three points $(a, f(a))$, $(\frac{a+b}{2}, f(\frac{a+b}{2}))$, $(b, f(b))$, there is a unique polynomial of degree ≤ 2 passing through them — the **Lagrange interpolating polynomial**.

Integrating this parabola exactly gives Simpson's 1/3 rule:

$$\int_a^b f(x)\,dx \;\approx\; \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$

Error: $O(n^{-4})$ — two orders better than trapezoidal.



Why $O(n^{-4})$? Simpson as a Combination

$O(n^{-4})$ seems too good for fitting a parabola. Where does it come from?

Midpoint and trapezoidal are both $O(n^{-2})$, but with opposite signs:

$$M_n = I + \frac{c}{n^2} + O(n^{-4})$$

$$T_n = I - \frac{2c}{n^2} + O(n^{-4})$$

The midpoint error is half the size and opposite in sign. Weigh them to cancel:

$$\boxed{S_n = \frac{2M_n + T_n}{3} = I + O(n^{-4})}$$

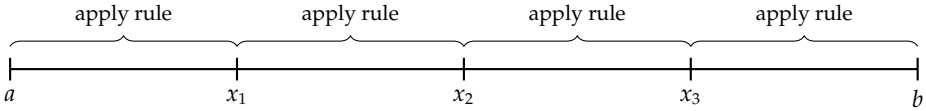
Expanding this gives exactly $\frac{b-a}{6}\big[f(a) + 4f(m) + f(b)\big]$ — the $1:4:1$ weights of Simpson's rule!

Composite Rules

In practice, we subdivide $[a, b]$ into n subintervals, apply the basic rule on each.

Composite Rules

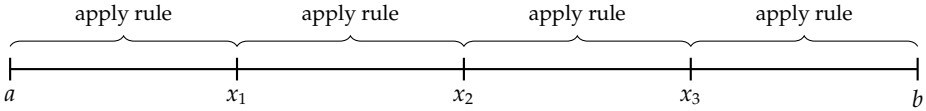
In practice, we subdivide $[a, b]$ into n subintervals, apply the basic rule on each.



Sum the results!

Composite Rules

In practice, we subdivide $[a, b]$ into n subintervals, apply the basic rule on each.



Sum the results!

Composite Rule	Error order	Error halving ratio
Trapezoidal	$O(n^{-2})$	$\div 4$ per doubling of n
Simpson's	$O(n^{-4})$	$\div 16$ per doubling of n

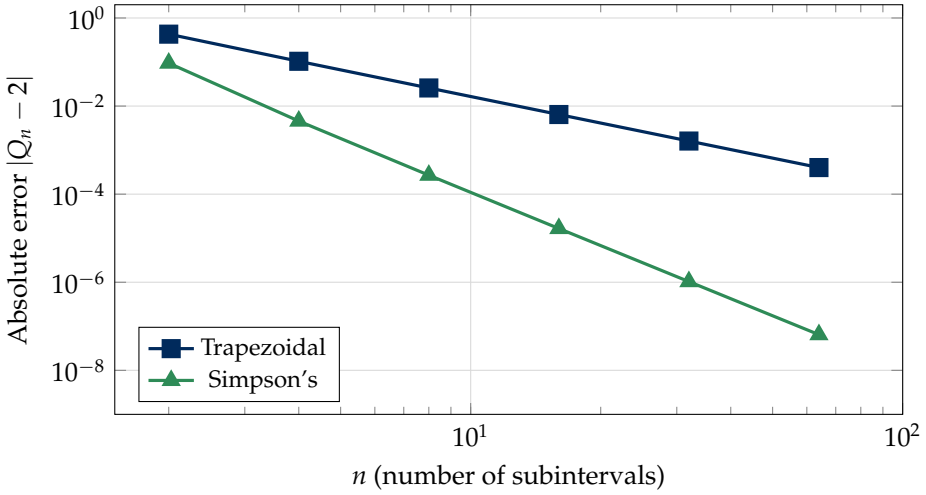
Convergence in Numbers: $\int_0^{\pi} \sin x \, dx = 2$

```
 Trapezoidal rule
n = 2: Q = 1.5707963268  error = 4.29e-01
n = 4: Q = 1.8961188979  error = 1.04e-01
n = 8: Q = 1.9742316019  error = 2.58e-02
n = 16: Q = 1.9935703438  error = 6.43e-03
n = 32: Q = 1.9983933610  error = 1.61e-03

Simpson's 1/3 rule
n = 2: Q = 2.0943951024  error = 9.44e-02
n = 4: Q = 2.0045597550  error = 4.56e-03
n = 8: Q = 2.0002691699  error = 2.69e-04
n = 16: Q = 2.0000165910  error = 1.66e-05
n = 32: Q = 2.0000010334  error = 1.03e-06
```

Trap: error $\div 4$ each doubling ($O(n^{-2})$). Simpson: error $\div 16$ each doubling ($O(n^{-4})$).

Convergence Plot: $\int_0^{\pi} \sin x \, dx = 2$



Straight lines on log-log = algebraic convergence. Steeper = faster.

Combining Estimates: Can We Keep Going?

We've been combining estimates to cancel leading errors:

- $T = \frac{1}{2}(L + R)$ cancelled $O(n^{-1}) \longrightarrow$ trapezoidal rule
- $S = \frac{1}{3}(2M + T)$ cancelled $O(n^{-2}) \longrightarrow$ Simpson's rule

Can we keep going? Yes. But we need to know the structure of the error.

The Euler–Maclaurin Formula

The Euler–Maclaurin formula gives the *exact* error expansion for the trapezoidal rule. The first few terms:

$$T_n - I \;=\; \frac{(b-a)^2}{12\,n^2}\big[f'(b) - f'(a)\big] \;-\; \frac{(b-a)^4}{720\,n^4}\big[f'''(b) - f'''(a)\big] \;+\; \cdots$$

The key observation: only even powers of $1/n$ appear. We can write

$$T_n \;=\; I + \frac{a_1}{n^2} + \frac{a_2}{n^4} + \frac{a_3}{n^6} + \cdots$$

where $a_1, a_2, \ldots$ depend on f and the interval, but not on n.

This is what makes the "combine and cancel" trick work systematically: if we compute T_n for two different values of n, the unknown constants are the same, so we can eliminate them one by one.

Richardson Extrapolation

Write out the expansion for T_n and T_{2n} (doubling the number of panels):

$$T_n = I + \frac{a_1}{n^2} + O(n^{-4}) \qquad\qquad T_{2n} = I + \frac{a_1}{4n^2} + O(n^{-4})$$

Multiply the second by 4 and subtract the first:

$$4\,T_{2n} - T_n \;=\; 3I + O(n^{-4})$$

$$\boxed{\frac{4\,T_{2n} - T_n}{3} \;=\; I + O(n^{-4})}$$

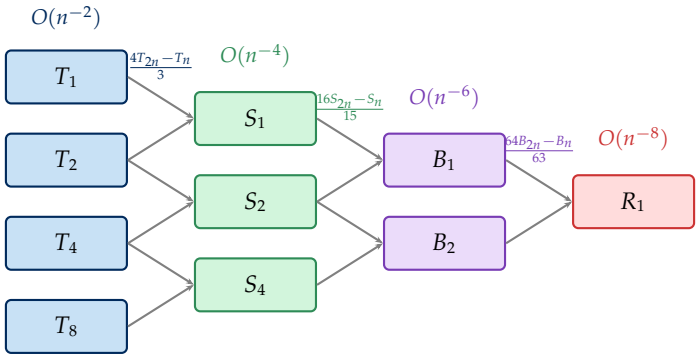
Two $O(n^{-2})$ estimates combine into one $O(n^{-4})$ estimate, with no extra function evaluations.

This is called Richardson extrapolation. It works for any method whose error has a predictable expansion in powers of $1/n$.

We can apply it again to cancel the n^{-4} term, then n^{-6}, and so on. This cascade is called Romberg integration.

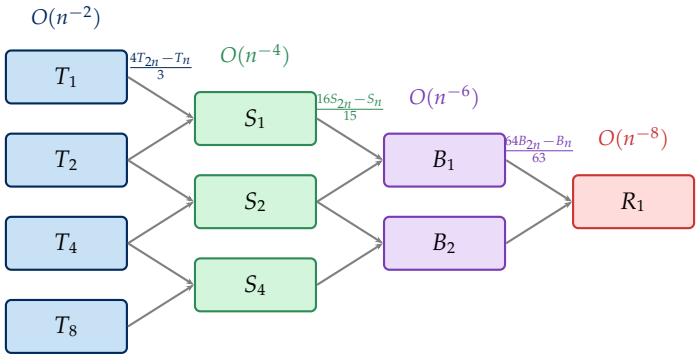
Romberg Integration: The Extrapolation Cascade

Keep cancelling: $n^{-4}, n^{-6}, n^{-8}, \ldots$



Romberg Integration: The Extrapolation Cascade

Keep cancelling: $n^{-4}, n^{-6}, n^{-8}, \ldots$



Each column cancels the next error term.

Free accuracy from data you already computed!

Romberg in Action: $\int_0^1 e^x\,dx = e - 1$

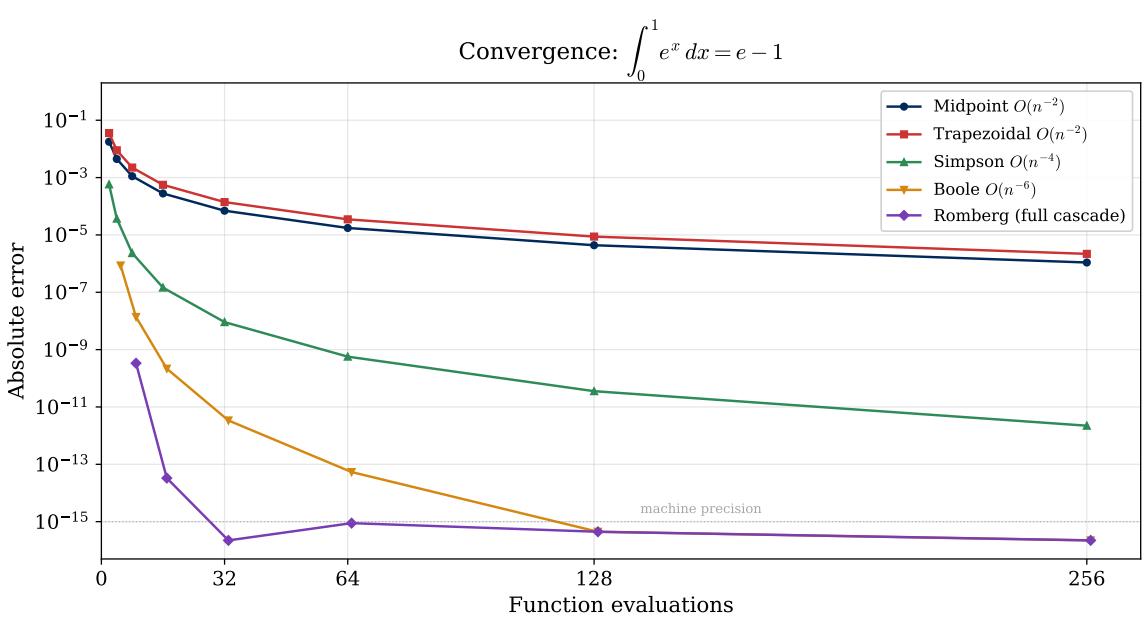
The trapezoidal nodes are nested: the n=8 grid contains the n=4, n=2, and n=1 grids as subsets. So T_1, T_2, T_4 come for free from the 9 evaluations needed for T_8.

```
 Trapezoidal (nested grids, 9 evaluations total)
T_1 = 1.859140914230  error = 1.41e-01 (2 evals)
T_2 = 1.753931092465  error = 3.56e-02 (3 evals)
T_4 = 1.727221904558  error = 8.94e-03 (5 evals)
T_8 = 1.720518592164  error = 2.24e-03 (9 evals)

Extrapolation (no new evaluations)
S_1 = 1.718861151877  error = 5.79e-04
S_2 = 1.718318841922  error = 3.70e-05
S_4 = 1.718284154700  error = 2.33e-06
B_1 = 1.718282687925  error = 8.59e-07
B_2 = 1.718281842218  error = 1.38e-08
R_1 = 1.718281828795  error = 3.35e-10
```

9 evaluations of f, error $\sim 10^{-10}$ — by reusing the same data.

Convergence Comparison: $\int_0^1 e^x\,dx = e - 1$



Part 3

Gaussian Quadrature

From Degree of Exactness to Optimal Node Placement

Degree of Exactness

Definition 2

A quadrature rule $Q_n[f] = \sum w_i f(x_i)$ is *exact* for a function g if $Q_n[g] = \int_a^b g(x)\,dx$. We say Q_n has **degree of exactness** (DoE) d if it is exact for every polynomial of degree $\leq d$, but not for some polynomial of degree $d+1$.

Example: trapezoidal on $[-1, 1]$:

$g(x)$	$\int_{-1}^{1} g\,dx$	Exact?
1	2	✓
x	0	✓
x^2	$\frac{2}{3}$	×

DoE $= 1$.

Rule	n	DoE	DoE$/n$
Midpoint	1	1	1.0
Trapezoidal	2	1	0.5
Simpson's	3	3	1.0

Can we do better than 1 per node?

Why Care about Exactness?

1. Exact for monomials $\Rightarrow$ exact for all polynomials of that degree, by linearity:

$$Q_n\Big[\sum_{k=0}^{d} a_k x^k\Big] = \sum_{k=0}^{d} a_k \underbrace{Q_n[x^k]}_{=\int x^k} = \int_a^b \sum_{k=0}^{d} a_k x^k \, dx$$

2. By the Weierstrass approximation theorem, every continuous function can be approximated by polynomials. So higher DoE $\Rightarrow$ faster convergence for smooth f.

Why Care about Exactness?

1. Exact for monomials $\Rightarrow$ exact for all polynomials of that degree, by linearity:

$$Q_n\Big[\sum_{k=0}^{d} a_k x^k\Big] = \sum_{k=0}^{d} a_k \underbrace{Q_n[x^k]}_{=\int x^k} = \int_a^b \sum_{k=0}^{d} a_k x^k \, dx$$

2. By the Weierstrass approximation theorem, every continuous function can be approximated by polynomials. So higher DoE $\Rightarrow$ faster convergence for smooth f.

The practical question: given n nodes, how do we choose them and the weights to maximize DoE?

Newton–Cotes: The Natural First Approach

Fix nodes to be equally spaced, then choose weights to maximise DoE.

Given $n+1$ nodes $x_0, x_1, \ldots, x_n$ with spacing $h = (b-a)/n$, the weights come from integrating the Lagrange interpolant:

$$w_i \;=\; \int_a^b \ell_i(x)\,dx \qquad \text{where} \quad \ell_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j}$$

Newton–Cotes: The Natural First Approach

Fix nodes to be equally spaced, then choose weights to maximise DoE.

Given $n+1$ nodes $x_0, x_1, \ldots, x_n$ with spacing $h = (b-a)/n$, the weights come from integrating the Lagrange interpolant:

$$w_i = \int_a^b \ell_i(x)\,dx \qquad \text{where} \quad \ell_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j}$$

In matrix form, this is a linear system:

$$\underbrace{\begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_0 & x_1 & \cdots & x_n \\ \vdots & \vdots & & \vdots \\ x_0^n & x_1^n & \cdots & x_n^n \end{pmatrix}}_{\text{Vandermonde Matrix}} \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{pmatrix} = \begin{pmatrix} \int 1\,dx \\ \int x\,dx \\ \vdots \\ \int x^n\,dx \end{pmatrix}$$

Newton–Cotes: The Natural First Approach

Fix nodes to be equally spaced, then choose weights to maximise DoE.

Given $n+1$ nodes $x_0, x_1, \ldots, x_n$ with spacing $h = (b-a)/n$, the weights come from integrating the Lagrange interpolant:

$$w_i \;=\; \int_a^b \ell_i(x)\,dx \qquad \text{where} \quad \ell_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j}$$

In matrix form, this is a linear system:

$$\underbrace{\begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_0 & x_1 & \cdots & x_n \\ \vdots & \vdots & & \vdots \\ x_0^n & x_1^n & \cdots & x_n^n \end{pmatrix}}_{\text{Vandermonde Matrix}} \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{pmatrix} = \begin{pmatrix} \int 1\,dx \\ \int x\,dx \\ \vdots \\ \int x^n\,dx \end{pmatrix}$$

Solve once and get a lookup table of weights.

Newton–Cotes Rules

The trapezoidal, Simpson, and Boole rules we have seen have all just been examples of this:

n	Name	Weights (scaled)	DoE
1	Trapezoidal	$\frac{h}{2}[1,\ 1]$	1
2	Simpson's 1/3	$\frac{h}{3}[1,\ 4,\ 1]$	3
3	Simpson's 3/8	$\frac{3h}{8}[1,\ 3,\ 3,\ 1]$	3
4	Boole's	$\frac{2h}{45}[7,\ 32,\ 12,\ 32,\ 7]$	5

Note: for even n, DoE $= n + 1$ (symmetry bonus). For odd n, DoE $= n$.

Why We Never Build the Polynomial

Q: If we're integrating the interpolant, don't we first need to construct it?

Why We Never Build the Polynomial

Q: If we're integrating the interpolant, don't we first need to construct it? **A:** No, since:

$$\int_a^b \underbrace{p_n(x)}_{\text{interpolant of } f} dx = \int_a^b \sum_{i=0}^{n} f(x_i)\,\ell_i(x)\,dx = \sum_{i=0}^{n} f(x_i) \underbrace{\int_a^b \ell_i(x)\,dx}_{=w_i}$$

Why We Never Build the Polynomial

Q: If we're integrating the interpolant, don't we first need to construct it? **A:** No, since:

$$\int_a^b \underbrace{p_n(x)}_{\text{interpolant of } f}\, dx \;=\; \int_a^b \sum_{i=0}^{n} f(x_i)\,\ell_i(x)\,dx \;=\; \sum_{i=0}^{n} f(x_i) \underbrace{\int_a^b \ell_i(x)\,dx}_{=w_i}$$

The integrals $\int \ell_i(x)\,dx$ depend only on the nodes, not on f. So:

1. Precompute (once): solve for $w_i = \int \ell_i\,dx$;
2. Then just compute $\sum w_i f(x_i)$.

Why We Never Build the Polynomial

Q: If we're integrating the interpolant, don't we first need to construct it? **A:** No, since:

$$\int_a^b \underbrace{p_n(x)}_{\text{interpolant of } f} dx = \int_a^b \sum_{i=0}^{n} f(x_i)\,\ell_i(x)\,dx = \sum_{i=0}^{n} f(x_i) \underbrace{\int_a^b \ell_i(x)\,dx}_{=w_i}$$

The integrals $\int \ell_i(x)\,dx$ depend only on the nodes, not on f. So:

1. Precompute (once): solve for $w_i = \int \ell_i\,dx$;
2. Then just compute $\sum w_i f(x_i)$.

The polynomial interpolation is implicit and baked into the weights.
For a new f, we never touch polynomials at all.

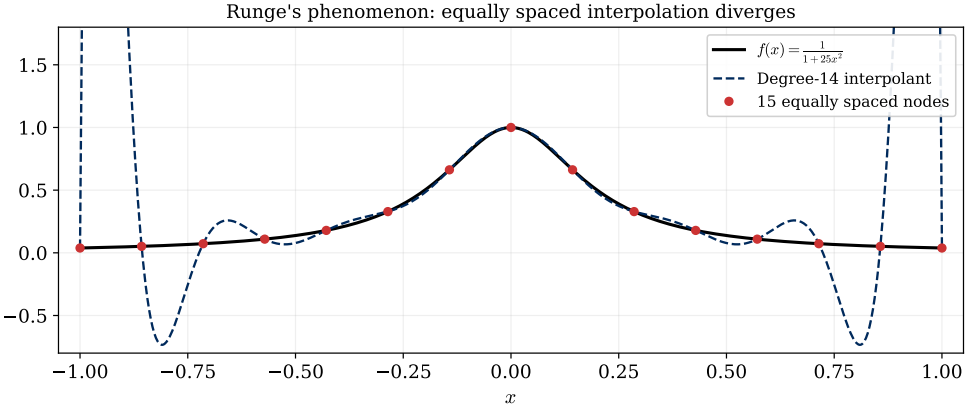
Newton–Cotes: What Goes Wrong at High Order

For $n \geq 8$, some weights become negative:

$$\text{8-point:} \quad \frac{4h}{14175}\left[989,\ 5888,\ -928,\ 10496,\ -4540,\ 10496,\ -928,\ 5888,\ 989\right]$$

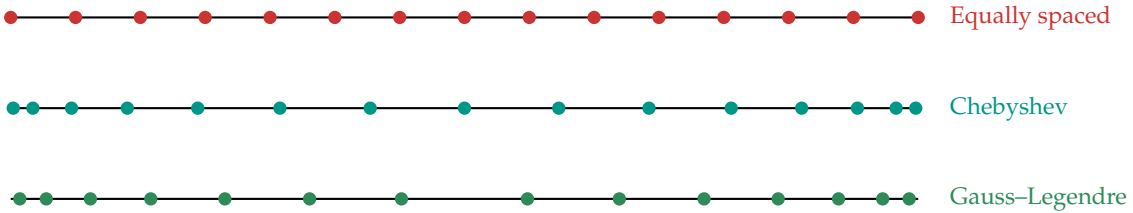
Consequences: $f \geq 0$ does not guarantee $Q_n[f] \geq 0$. Moreover, $\sum |w_i|$ grows, and often causes catastrophic cancellation.

One reason for this is that high-degree polynomial interpolation at equally-spaced nodes is unstable (Runge's phenomenon):



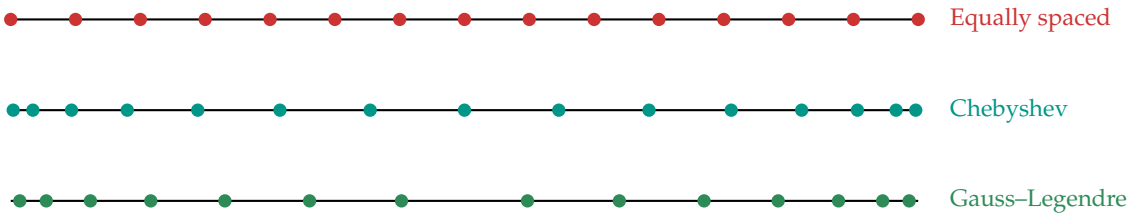
The Lesson: Node Placement Matters

Equally-spaced nodes are the wrong choice for high-order quadrature.



The Lesson: Node Placement Matters

Equally-spaced nodes are the wrong choice for high-order quadrature.



We need to cluster nodes near the endpoints. But how do we find the optimal positions?

Freeing the Nodes

Newton–Cotes fixes nodes equally spaced. What if we free the node positions too?

With N nodes we have $2N$ free parameters (nodes + weights). We may demand exactness for $1, x, \ldots, x^{2N-1}$:

$$\sum_{i=1}^{N} w_i x_i^k = \int_{-1}^{1} x^k \, dx \qquad \text{for } k = 0, 1, \ldots, 2N-1$$

This is $2N$ equations in $2N$ unknowns $\Rightarrow$ DoE $= 2N-1$.

Freeing the Nodes

Newton–Cotes fixes nodes equally spaced. What if we free the node positions too?

With N nodes we have $2N$ free parameters (nodes + weights). We may demand exactness for $1, x, \ldots, x^{2N-1}$:

$$\sum_{i=1}^{N} w_i x_i^k \;=\; \int_{-1}^{1} x^k \, dx \qquad \text{for } k = 0, 1, \ldots, 2N-1$$

This is $2N$ equations in $2N$ unknowns $\Rightarrow$ DoE $= 2N-1$.

$N = 1$: 2 unknowns (x_1, w_1). $w_1 = \int_{-1}^{1} 1 \, dx = 2$, $w_1 x_1 = \int_{-1}^{1} x \, dx = 0$.
$\Longrightarrow$ $x_1 = 0,\ w_1 = 2$. This is exactly the midpoint rule.

Freeing the Nodes

Newton–Cotes fixes nodes equally spaced. What if we free the node positions too?

With N nodes we have $2N$ free parameters (nodes + weights). We may demand exactness for $1, x, \ldots, x^{2N-1}$:

$$\sum_{i=1}^{N} w_i x_i^k \;=\; \int_{-1}^{1} x^k \, dx \qquad \text{for } k = 0, 1, \ldots, 2N-1$$

This is $2N$ equations in $2N$ unknowns $\Rightarrow$ DoE $= 2N-1$.

$N = 1$: 2 unknowns (x_1, w_1). $w_1 = \int_{-1}^{1} 1 \, dx = 2$, $w_1 x_1 = \int_{-1}^{1} x \, dx = 0$.
$\Longrightarrow$ $x_1 = 0,\ w_1 = 2$. This is exactly the midpoint rule.

$N = 2$: 4 unknowns (x_1, x_2, w_1, w_2).

$$w_1 + w_2 = 2 \qquad\qquad w_1 x_1 + w_2 x_2 = 0$$
$$w_1 x_1^2 + w_2 x_2^2 = \tfrac{2}{3} \qquad\qquad w_1 x_1^3 + w_2 x_2^3 = 0$$

$\Longrightarrow$ $x_{1,2} = \pm 1/\sqrt{3},\ \ w_{1,2} = 1$. Degree of exactness is 3, which is same as Simpson, but with only 2 points!

$N = 3$: The Nonlinear System

6 parameters $(x_1, x_2, x_3,\ w_1, w_2, w_3)$, match $\sum_i w_i\, x_i^k = \int_{-1}^{1} x^k\, dx$ for $k = 0, \ldots, 5$:

$$
\begin{aligned}
w_1 + w_2 + w_3 &= \int_{-1}^{1} 1\, dx \quad (k=0)\\
w_1 x_1 + w_2 x_2 + w_3 x_3 &= \int_{-1}^{1} x\, dx \quad (k=1)\\
w_1 x_1^2 + w_2 x_2^2 + w_3 x_3^2 &= \int_{-1}^{1} x^2\, dx \quad (k=2)\\
w_1 x_1^3 + w_2 x_2^3 + w_3 x_3^3 &= \int_{-1}^{1} x^3\, dx \quad (k=3)\\
w_1 x_1^4 + w_2 x_2^4 + w_3 x_3^4 &= \int_{-1}^{1} x^4\, dx \quad (k=4)\\
w_1 x_1^5 + w_2 x_2^5 + w_3 x_3^5 &= \int_{-1}^{1} x^5\, dx \quad (k=5)
\end{aligned}
$$

Linear in the w_i, but nonlinear in the x_i.

$N = 3$: The Nonlinear System

6 parameters $(x_1, x_2, x_3,\ w_1, w_2, w_3)$, match $\sum_i w_i\, x_i^k = \int_{-1}^{1} x^k\, dx$ for $k = 0, \ldots, 5$:

$$
\begin{aligned}
w_1 + w_2 + w_3 &= \int_{-1}^{1} 1\, dx \quad (k=0)\\
w_1 x_1 + w_2 x_2 + w_3 x_3 &= \int_{-1}^{1} x\, dx \quad (k=1)\\
w_1 x_1^2 + w_2 x_2^2 + w_3 x_3^2 &= \int_{-1}^{1} x^2\, dx \quad (k=2)\\
w_1 x_1^3 + w_2 x_2^3 + w_3 x_3^3 &= \int_{-1}^{1} x^3\, dx \quad (k=3)\\
w_1 x_1^4 + w_2 x_2^4 + w_3 x_3^4 &= \int_{-1}^{1} x^4\, dx \quad (k=4)\\
w_1 x_1^5 + w_2 x_2^5 + w_3 x_3^5 &= \int_{-1}^{1} x^5\, dx \quad (k=5)
\end{aligned}
$$

Linear in the w_i, but nonlinear in the x_i.

Solvable by Newton's method, but there is a shortcut. The optimal nodes turn out to be zeros of *orthogonal polynomials*.

The Optimal Nodes Are Roots of Special Polynomials

Collecting the solutions from the moment-matching systems:

N	Optimal nodes on $[-1, 1]$	DoE
1	0	1
2	$\pm 1/\sqrt{3} \approx \pm 0.5774$	3
3	$0,\ \pm\sqrt{3/5} \approx \pm 0.7746$	5

The Optimal Nodes Are Roots of Special Polynomials

Collecting the solutions from the moment-matching systems:

N	Optimal nodes on $[-1, 1]$	DoE
1	0	1
2	$\pm 1/\sqrt{3} \approx \pm 0.5774$	3
3	$0,\ \pm\sqrt{3/5} \approx \pm 0.7746$	5

These are exactly the roots of the *Legendre polynomials*:

$$P_1(x) = x, \qquad P_2(x) = \tfrac{1}{2}(3x^2 - 1), \qquad P_3(x) = \tfrac{1}{2}(5x^3 - 3x).$$

The Optimal Nodes Are Roots of Special Polynomials

Collecting the solutions from the moment-matching systems:

N	Optimal nodes on $[-1, 1]$	DoE
1	0	1
2	$\pm 1/\sqrt{3} \approx \pm 0.5774$	3
3	$0,\ \pm\sqrt{3/5} \approx \pm 0.7746$	5

These are exactly the roots of the *Legendre polynomials*:

$$P_1(x) = x, \qquad P_2(x) = \tfrac{1}{2}(3x^2 - 1), \qquad P_3(x) = \tfrac{1}{2}(5x^3 - 3x).$$

Where do these polynomials come from?

Orthogonal Polynomials

Define a weighted inner product on $[a, b]$ with weight $w(x) \geq 0$:

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

Orthogonal Polynomials

Define a weighted inner product on $[a, b]$ with weight $w(x) \geq 0$:

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

For $w(x) = 1$ on $[-1, 1]$, apply Gram–Schmidt to $\{1, x, x^2, x^3, \ldots\}$:

$$P_0 = 1, \qquad P_{n+1}(x) = x\, P_n(x) - \frac{\langle xP_n,\, P_n \rangle}{\langle P_n,\, P_n \rangle}\, P_n(x) - \frac{\langle xP_n,\, P_{n-1} \rangle}{\langle P_{n-1},\, P_{n-1} \rangle}\, P_{n-1}(x)$$

Orthogonal Polynomials

Define a weighted inner product on $[a, b]$ with weight $w(x) \geq 0$:

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

For $w(x) = 1$ on $[-1, 1]$, apply Gram–Schmidt to $\{1, x, x^2, x^3, \ldots\}$:

$$P_0 = 1, \qquad P_{n+1}(x) = x\, P_n(x) - \frac{\langle xP_n,\, P_n \rangle}{\langle P_n,\, P_n \rangle}\, P_n(x) - \frac{\langle xP_n,\, P_{n-1} \rangle}{\langle P_{n-1},\, P_{n-1} \rangle}\, P_{n-1}(x)$$

This gives us a three-term recurrence. Why only two terms? Multiplication by x raises degree by 1, so $\langle xP_n, P_k \rangle = 0$ for $k < n - 1$ by orthogonality. Only the last two terms survive.

Orthogonal Polynomials

Define a weighted inner product on $[a, b]$ with weight $w(x) \geq 0$:

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

For $w(x) = 1$ on $[-1, 1]$, apply Gram–Schmidt to $\{1, x, x^2, x^3, \ldots\}$:

$$P_0 = 1, \qquad P_{n+1}(x) = x\, P_n(x) - \frac{\langle xP_n,\, P_n \rangle}{\langle P_n,\, P_n \rangle}\, P_n(x) - \frac{\langle xP_n,\, P_{n-1} \rangle}{\langle P_{n-1},\, P_{n-1} \rangle}\, P_{n-1}(x)$$

This gives us a three-term recurrence. Why only two terms? Multiplication by x raises degree by 1, so $\langle xP_n, P_k \rangle = 0$ for $k < n - 1$ by orthogonality. Only the last two terms survive.

The resulting $\{P_n\}$ are the Legendre polynomials, satisfying:

$$\langle P_m,\, P_n \rangle \;=\; \int_{-1}^{1} P_m(x)\, P_n(x)\, dx \;=\; \frac{2}{2n+1}\, \delta_{mn}$$

Gaussian Quadrature: DoE $= 2N-1$

Theorem 3

Let $x_1, \ldots, x_N$ be the zeros of P_N, with weights $w_i = \int_{-1}^{1} \ell_i(x)\,dx$. Then the rule $\sum w_i f(x_i)$ has DoE $= 2N-1$, and:

1. *All weights $w_i > 0$*
2. *All nodes $x_i \in (-1, 1)$*
3. *No N-point rule can achieve DoE $\geq 2N$*

Gaussian Quadrature: DoE $= 2N-1$

Theorem 3

Let $x_1, \ldots, x_N$ be the zeros of P_N, with weights $w_i = \int_{-1}^{1} \ell_i(x)\,dx$. Then the rule $\sum w_i f(x_i)$ has DoE $= 2N-1$, and:

1. *All weights $w_i > 0$*
2. *All nodes $x_i \in (-1, 1)$*
3. *No N-point rule can achieve DoE $\geq 2N$*

Proof sketch. Let q be any polynomial of degree $\leq 2N-1$. Divide by P_N:

$$q(x) = \underbrace{P_N(x)}_{\deg N} \cdot \underbrace{s(x)}_{\deg \leq N-1} + \underbrace{r(x)}_{\deg \leq N-1}$$

$$\int_{-1}^{1} q\,dx = \underbrace{\int_{-1}^{1} P_N \cdot s\,dx}_{=0 \text{ (orthogonality)}} + \int_{-1}^{1} r\,dx. \qquad \sum w_i\, q(x_i) = \sum w_i [\underbrace{P_N(x_i)}_{=0} \cdot s(x_i) + r(x_i)] = \int_{-1}^{1} r\,dx. \quad \checkmark$$

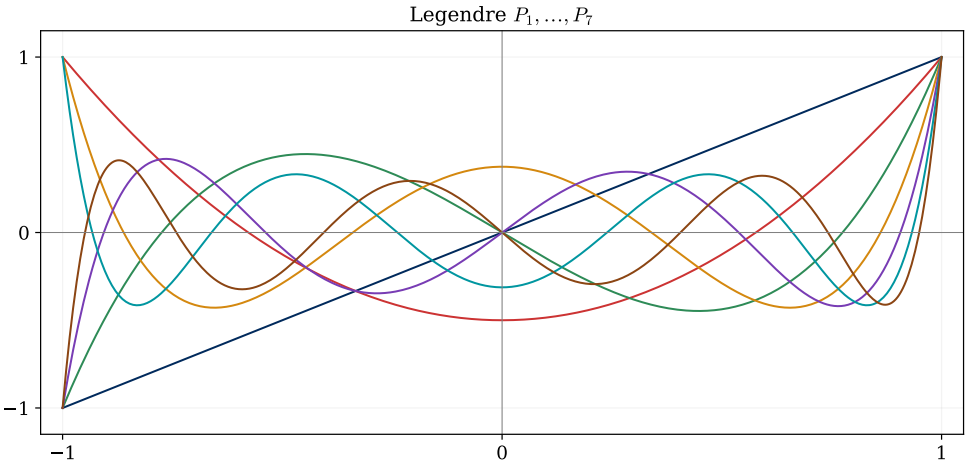
(The recurrence coefficients also form a tridiagonal matrix whose eigenvalues give the nodes — this is the Golub–Welsch algorithm.)

The Classical Families: Different $w(x)$, Different Rules

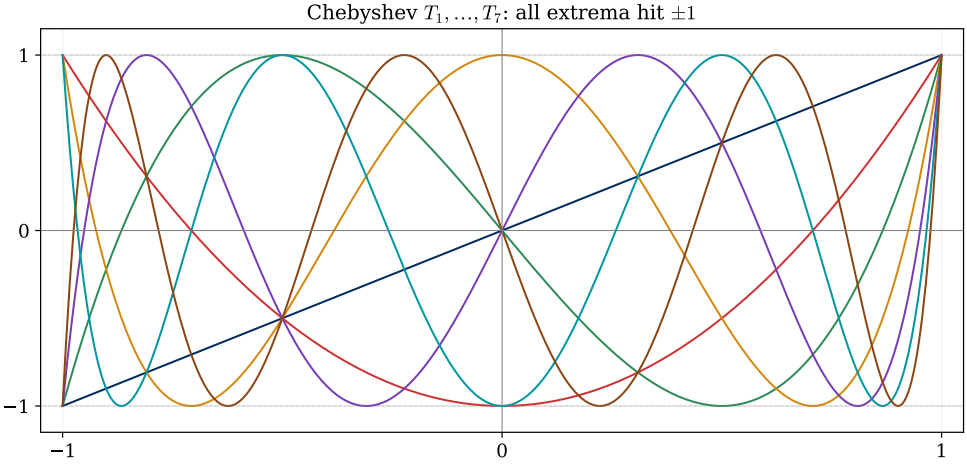
Changing the weight $w(x)$ in $\langle f, g \rangle_w$ gives different orthogonal polynomials, and hence different Gaussian quadrature rules:

Family	**Weight** $w(x)$	**Interval**	**Application**
Legendre	1	$[-1, 1]$	Default quadrature
Chebyshev I	$(1 - x^2)^{-1/2}$	$[-1, 1]$	Approx. theory, FFT
Jacobi	$(1-x)^{\alpha}(1+x)^{\beta}$	$[-1, 1]$	Generalises above
Hermite	e^{-x^2}	$(-\infty, \infty)$	Statistics

Legendre Polynomials $P_1, \ldots, P_7$



Chebyshev Polynomials $T_1, \ldots, T_7$



We say that T_n equioscillates, i.e., every extremum hits exactly ± 1.

Gauss–Legendre in Action: $\int_0^1 e^x\,dx = e - 1$

Transform $[-1,1] \to [0,1]$ via $x = \frac{t+1}{2}$, then $\int_0^1 f(x)\,dx = \frac{1}{2}\sum w_i f\left(\frac{x_i+1}{2}\right)$.

```
 Gauss--Legendre (N points, precomputed nodes & weights)
N = 1:  Q = 1.648721270700  error = 6.96e-02
N = 2:  Q = 1.717896378008  error = 3.85e-04
N = 3:  Q = 1.718281004373  error = 8.24e-07
N = 4:  Q = 1.718281827526  error = 9.33e-10
N = 5:  Q = 1.718281828458  error = 6.54e-13
```

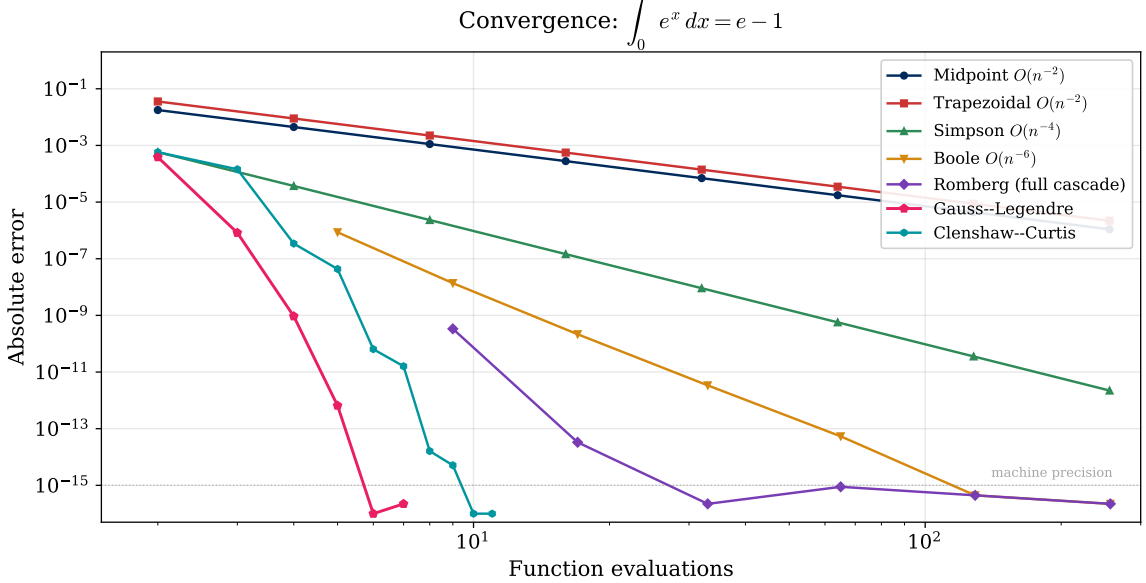
Gauss–Legendre in Action: $\int_0^1 e^x\,dx = e - 1$

Transform $[-1,1] \to [0,1]$ via $x = \frac{t+1}{2}$, then $\int_0^1 f(x)\,dx = \frac{1}{2}\sum w_i f\left(\frac{x_i+1}{2}\right)$.

```
 Gauss--Legendre (N points, precomputed nodes & weights)
N = 1:  Q = 1.648721270700  error = 6.96e-02
N = 2:  Q = 1.717896378008  error = 3.85e-04
N = 3:  Q = 1.718281004373  error = 8.24e-07
N = 4:  Q = 1.718281827526  error = 9.33e-10
N = 5:  Q = 1.718281828458  error = 6.54e-13
```

Compare with Romberg (slide 23): 9 evaluations for error $\sim 10^{-10}$.
GL reaches 10^{-13} with just **5 evaluations**.

Convergence Comparison: $\int_0^1 e^x\,dx = e - 1$



Clenshaw–Curtis: A Practical Alternative

Use Chebyshev nodes $x_k = \cos(k\pi/n)$ instead of GL nodes.

DoE is only n (vs. $2n-1$ for GL) — so why bother?

Clenshaw–Curtis: A Practical Alternative

Use Chebyshev nodes $x_k = \cos(k\pi/n)$ instead of GL nodes.

DoE is only n (vs. $2n-1$ for GL) — so why bother?

Three practical advantages:

1. Weights computed via FFT in $O(n \log n)$ (vs. $O(n^2)$ for GL)
2. Nested nodes: doubling n reuses all previous evaluations
3. Chebyshev nodes minimise $\max |\omega_n(x)|$ over $[-1, 1]$ — provably the best interpolation nodes (Lebesgue constant $\sim \log n$)

Clenshaw–Curtis: A Practical Alternative

Use Chebyshev nodes $x_k = \cos(k\pi/n)$ instead of GL nodes.

DoE is only n (vs. $2n-1$ for GL) — so why bother?

Three practical advantages:

1. Weights computed via FFT in $O(n \log n)$ (vs. $O(n^2)$ for GL)
2. Nested nodes: doubling n reuses all previous evaluations
3. Chebyshev nodes minimise $\max |\omega_n(x)|$ over $[-1, 1]$ — provably the best interpolation nodes (Lebesgue constant $\sim \log n$)

Theorem 4 (Trefethen, 2008)

For functions analytic near $[-1, 1]$, Gauss and Clenshaw–Curtis converge at the same exponential rate. The factor-of-2 difference in DoE is irrelevant in practice.

The "extra" polynomials GL integrates exactly (degrees $n+1$ through $2n-1$) have Chebyshev coefficients that are already negligibly small.

Error Estimation: Gauss–Kronrod

Problem: a quadrature rule gives you a number, but *how accurate is it*?

Error Estimation: Gauss–Kronrod

Problem: a quadrature rule gives you a number, but *how accurate is it*?

Gauss–Kronrod rules: embed an n-point Gauss rule inside a $(2n+1)$-point rule that **reuses all n evaluations**.

The difference $|G_n[f] - K_{2n+1}[f]|$ estimates the error.

Error Estimation: Gauss–Kronrod

Problem: a quadrature rule gives you a number, but *how accurate is it*?

Gauss–Kronrod rules: embed an n-point Gauss rule inside a $(2n+1)$-point rule that **reuses all n evaluations**.

The difference $|G_n[f] - K_{2n+1}[f]|$ estimates the error.

This is what library codes do:

- `scipy.integrate.quad`: Gauss–Kronrod (G10-K21 or G7-K15)
- MATLAB `integral`: adaptive Gauss–Kronrod
- Both use the error estimate to **adaptively subdivide** where accuracy is insufficient

Error Estimation: Gauss–Kronrod

Problem: a quadrature rule gives you a number, but *how accurate is it*?

Gauss–Kronrod rules: embed an n-point Gauss rule inside a $(2n+1)$-point rule that **reuses all n evaluations**.

The difference $|G_n[f] - K_{2n+1}[f]|$ estimates the error.

This is what library codes do:

- `scipy.integrate.quad`: Gauss–Kronrod (G10-K21 or G7-K15)
- MATLAB `integral`: adaptive Gauss–Kronrod
- Both use the error estimate to **adaptively subdivide** where accuracy is insufficient

Variants: Gauss–Lobatto (force both endpoints as nodes, DoE $= 2n-3$),
Gauss–Radau (force one endpoint, DoE $= 2n-2$).

Error Estimation: Gauss–Kronrod

Problem: a quadrature rule gives you a number, but *how accurate is it*?

Gauss–Kronrod rules: embed an n-point Gauss rule inside a $(2n+1)$-point rule that **reuses all n evaluations**.

The difference $|G_n[f] - K_{2n+1}[f]|$ estimates the error.

This is what library codes do:

- `scipy.integrate.quad`: Gauss–Kronrod (G10-K21 or G7-K15)
- MATLAB `integral`: adaptive Gauss–Kronrod
- Both use the error estimate to **adaptively subdivide** where accuracy is insufficient

Variants: Gauss–Lobatto (force both endpoints as nodes, DoE $= 2n-3$),
Gauss–Radau (force one endpoint, DoE $= 2n-2$).

For **smooth** integrands, the problem is solved: GL, CC, Gauss–Kronrod, and Romberg give us all the tools we need.

What happens when f isn't smooth?

Part 4

When Smoothness Breaks

Singular Integrands and Generalized Gaussian Quadrature

What Goes Wrong with Singular Integrands

$$\int_{-1}^{1} |x|^{\alpha}\, dx = \frac{2}{\alpha+1} \quad \text{with } N\text{-point Gauss–Legendre:}$$

```
 alpha = 4 (smooth), exact = 0.4000000000
n = 3:  error = 1.11e-16
n = 5:  error = 1.67e-16
n = 15:  error = 2.22e-16

alpha = 1 (kink), exact = 1.0000000000
n = 3:  error = 1.39e-01
n = 5:  error = 5.51e-02
n = 15:  error = 6.86e-03

alpha = 0.5 (unbounded deriv), exact = 1.3333333333
n = 3:  error = 3.55e-01
n = 5:  error = 1.80e-01
n = 15:  error = 3.79e-02
```

For smooth functions: machine precision in 3 points.
For singular functions: barely converges even at $n = 15$.

Three Strategies to Recover

1. **Substitution** — remove the singularity analytically
2. **Adaptive quadrature** — concentrate effort near trouble
3. **Generalised Gaussian quadrature** — build the singularity into the rule

Strategy 1: Change of Variables

If you know the singularity type, a substitution can remove it.

Example: $\displaystyle\int_0^1 x^{-1/2}\, g(x)\, dx$

Substitute $x = t^2$, $dx = 2t\, dt$:

$$\int_0^1 t^{-1} \cdot g(t^2) \cdot 2t\, dt \;=\; 2\int_0^1 g(t^2)\, dt$$

The integrand is now smooth, so we may apply standard GL.

Strategy 1: Change of Variables

If you know the singularity type, a substitution can remove it.

Example: $\displaystyle\int_0^1 x^{-1/2}\, g(x)\, dx$

Substitute $x = t^2$, $dx = 2t\, dt$:

$$\int_0^1 t^{-1} \cdot g(t^2) \cdot 2t\, dt \;=\; 2\int_0^1 g(t^2)\, dt$$

The integrand is now smooth, so we may apply standard GL.

Limitation: requires analytical knowledge of the singularity.

Strategy 2: Adaptive Quadrature

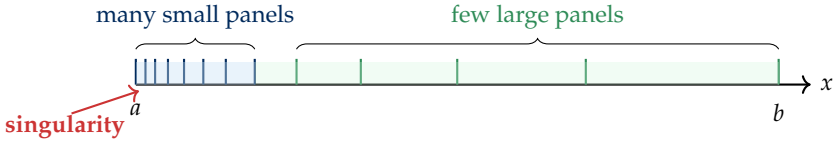
High Level Algorithm:

1. Apply a rule to $\int_a^b f\,dx$; estimate error via Gauss–Kronrod
2. If error $<$ tolerance, accept
3. Otherwise, split $[a, b]$ at the midpoint and recurse on each half

Strategy 2: Adaptive Quadrature

High Level Algorithm:

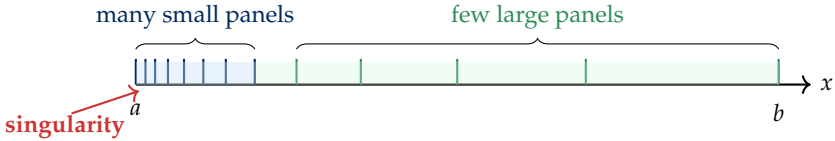
1. Apply a rule to $\int_a^b f\,dx$; estimate error via Gauss–Kronrod
2. If error $<$ tolerance, accept
3. Otherwise, split $[a, b]$ at the midpoint and recurse on each half



Strategy 2: Adaptive Quadrature

High Level Algorithm:

1. Apply a rule to $\int_a^b f\,dx$; estimate error via Gauss–Kronrod
2. If error $<$ tolerance, accept
3. Otherwise, split $[a, b]$ at the midpoint and recurse on each half



Automatic refinement concentrates effort near the singularity:

```
Adaptive G10--K21 for ∫_{-1}^{1} |x|^α dx:
alpha = 4   (smooth):  neval =  21
alpha = 1   (kink):    neval =  63
alpha = 0.5 (unbdd deriv):  neval = 483
```

It works, but singular integrands cost 10–20$\times$ more evaluations.

Strategy 3: Generalised Gaussian Quadrature

Substitution requires a different transform for each singularity, and is impractical when the same type of integral appears thousands of times.

Instead, replace the polynomial basis with one that includes the singularity:

$$\{1,\ x,\ x^2,\ \ldots\} \longrightarrow \{1,\ \log x,\ x,\ x\log x,\ x^2,\ x^2\log x,\ \ldots\}$$

Strategy 3: Generalised Gaussian Quadrature

Substitution requires a different transform for each singularity, and is impractical when the same type of integral appears thousands of times.

Instead, replace the polynomial basis with one that includes the singularity:

$$\{1,\ x,\ x^2,\ \ldots\} \longrightarrow \{1,\ \log x,\ x,\ x\log x,\ x^2,\ x^2\log x,\ \ldots\}$$

The generalisation rests on the following notion.

Definition 5 (Chebyshev System)

Functions $\{u_1, \ldots, u_m\} \subset L^1([a,b])$ form a *Chebyshev system* on (a,b) if for any m distinct points $x_1 < \cdots < x_m$ in (a,b),

$$\det \begin{pmatrix} u_1(x_1) & \cdots & u_1(x_m) \\ \vdots & \ddots & \vdots \\ u_m(x_1) & \cdots & u_m(x_m) \end{pmatrix} \neq 0.$$

The generalised Vandermonde matrix is always nonsingular. Integrability (rather than continuity) allows endpoint singularities like $\log x$ or x^α with $\alpha > -1$.

The Karlin–Studden Theorem

Theorem 6 (Karlin–Studden, 1966)

Let $\{u_1, \ldots, u_{2N}\}$ be a Chebyshev system on (a, b) and let $w \geq 0$ be an integrable weight function. Then there exists a unique N-point quadrature rule

$$\sum_{i=1}^{N} w_i f(x_i) \;=\; \int_a^b w(x) f(x)\, dx \qquad \text{for all } f \in \operatorname{span}\{u_1, \ldots, u_{2N}\}.$$

All weights $w_i > 0$ and all nodes $x_i \in (a, b)$.

The Karlin–Studden Theorem

Theorem 6 (Karlin–Studden, 1966)

Let $\{u_1, \ldots, u_{2N}\}$ be a Chebyshev system on (a, b) and let $w \geq 0$ be an integrable weight function. Then there exists a unique N-point quadrature rule

$$\sum_{i=1}^{N} w_i f(x_i) \;=\; \int_a^b w(x) f(x)\, dx \qquad \text{for all } f \in \operatorname{span}\{u_1, \ldots, u_{2N}\}.$$

All weights $w_i > 0$ and all nodes $x_i \in (a, b)$.

Same "Gaussian" property: N points, exact for $2N$ functions.

The Karlin–Studden Theorem

Theorem 6 (Karlin–Studden, 1966)

Let $\{u_1, \ldots, u_{2N}\}$ be a Chebyshev system on (a, b) and let $w \geq 0$ be an integrable weight function. Then there exists a unique N-point quadrature rule

$$\sum_{i=1}^{N} w_i f(x_i) \;=\; \int_a^b w(x) f(x)\, dx \qquad \text{for all } f \in \operatorname{span}\{u_1, \ldots, u_{2N}\}.$$

All weights $w_i > 0$ and all nodes $x_i \in (a, b)$.

Same "Gaussian" property: N points, exact for $2N$ functions.

Computing the nodes requires solving a nonlinear system via Newton's method with continuation from classical Gauss nodes (Ma–Rokhlin–Wandzura, 1996). The computation uses extended precision, but the resulting nodes and weights are stored as a lookup table.

Precompute once per singularity class, reuse for every integral of that type.

Precomputed GGQ Nodes and Weights (log-singular, $N = 8$)

```
 >>> ggq_nodes, ggq_weights = ggq_log(N=8)
>>> print(nodes, weights)
x_1 = 0.0010906939  w_1 = 0.0041243012
x_2 = 0.0154406535  w_2 = 0.0292703797
x_3 = 0.0694348621  w_3 = 0.0831140675
x_4 = 0.1874432443  w_4 = 0.1537216703
x_5 = 0.3733044213  w_5 = 0.2134976095
x_6 = 0.6004940137  w_6 = 0.2318702724
x_7 = 0.8168773397  w_7 = 0.1905362390
x_8 = 0.9628397593  w_8 = 0.0938654603
```

Nodes cluster near 0 (where $\log x \to -\infty$). All weights positive.

Computed once, stored, and reused for any $\int_0^1 f(x) \log x \, dx$.

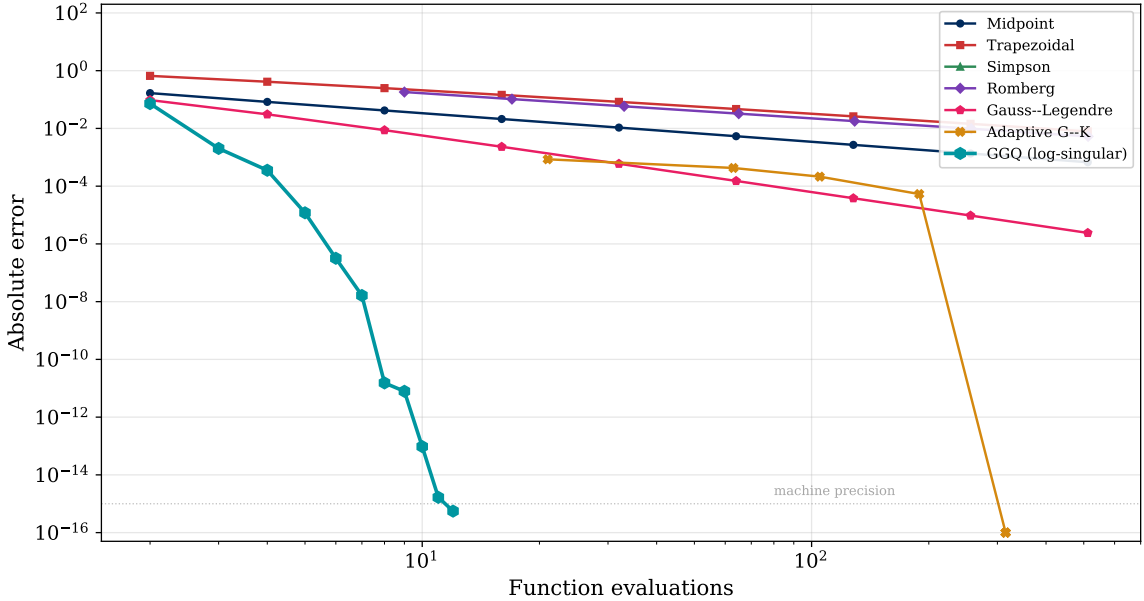
GGQ in Action: $\int_0^1 e^x \cos(2x) \cdot \log x \, dx$

```
 >>> gauss_legendre(f, 0, 1, N)
N = 4:  error = 3.08e-02
N = 6:  error = 1.49e-02
N = 8:  error = 8.71e-03
N = 10:  error = 5.72e-03
N = 12:  error = 4.04e-03

>>> ggq_log(f, N)
N = 4:  error = 3.55e-04
N = 6:  error = 3.18e-07
N = 8:  error = 1.53e-11
N = 10:  error = 9.55e-14
N = 12:  error = 5.55e-16
```

GL stalls at $\sim 10^{-3}$. GGQ reaches machine precision at $N = 12$.
Same number of evaluations — the singularity is handled exactly.

All Methods Compared: $\int_0^1 e^x \cos(2x) \cdot \log x \, dx$



Summary: Which Method for Which Integrand?

Integrand	Best method	Convergence
Smooth (analytic)	Gauss–Legendre / CC	Exponential
Smooth (finite regularity)	Gauss–Legendre	Algebraic $O(n^{-2k})$
Periodic on full period	Trapezoidal rule!	Exponential
Known singularity type	GGQ / substitution	Exponential
Unknown singularity	Adaptive (Gauss–Kronrod)	Varies

Summary: Which Method for Which Integrand?

Integrand	Best method	Convergence
Smooth (analytic)	Gauss–Legendre / CC	Exponential
Smooth (finite regularity)	Gauss–Legendre	Algebraic $O(n^{-2k})$
Periodic on full period	Trapezoidal rule!	Exponential
Known singularity type	GGQ / substitution	Exponential
Unknown singularity	Adaptive (Gauss–Kronrod)	Varies

In every case, the convergence rate is determined by how well the nodes are matched to the integrand.

Optimal nodes come from orthogonal polynomials — or, for singular integrands, from their generalisations via Chebyshev systems.

Part 5

Quadrature in Action

Every ODE Is an Integral Equation

Consider any initial value problem:

$$\dot{y} = f(t, y), \qquad y(0) = y_0$$

Every ODE Is an Integral Equation

Consider any initial value problem:

$$\dot{y} = f(t, y), \qquad y(0) = y_0$$

Integrate both sides from 0 to t:

$$y(t) = y_0 + \int_0^t f\big(s, y(s)\big)\, ds$$

Every ODE Is an Integral Equation

Consider any initial value problem:

$$\dot{y} = f(t, y), \qquad y(0) = y_0$$

Integrate both sides from 0 to t:

$$y(t) = y_0 + \int_0^t f\big(s, y(s)\big)\, ds$$

This looks like a quadrature problem — but $y(s)$ appears *inside* the integral. It is the very thing we are trying to find!

Every ODE Is an Integral Equation

Consider any initial value problem:

$$\dot{y} = f(t, y), \qquad y(0) = y_0$$

Integrate both sides from 0 to t:

$$y(t) = y_0 + \int_0^t f\big(s, y(s)\big)\, ds$$

This looks like a quadrature problem — but $y(s)$ appears *inside* the integral. It is the very thing we are trying to find!

The idea: apply an N-point quadrature rule to approximate the integral. The quadrature nodes become the points where we need y, giving a system of N equations in N unknowns.

Different quadrature rules lead to different ODE methods. Let's see how.

From Quadrature to ODE Solvers

Example: $\dot{y} = -y,\ y(0) = 1,$ single step h=1. Exact: $y(1) = e^{-1} \approx 0.3679$.

Left rectangle (Forward Euler):

$y(1) = 1 + 1 \cdot f(0,\ 1) = 1 + (-1) = 0.$ Only needs $y(0)$ — no system to solve. Error: 0.37

From Quadrature to ODE Solvers

Example: $\dot{y} = -y,\ y(0) = 1,$ single step h=1. Exact: $y(1) = e^{-1} \approx 0.3679$.

Left rectangle (Forward Euler):

$y(1) = 1 + 1 \cdot f(0,\, 1) = 1 + (-1) = 0.$ Only needs $y(0)$ — no system to solve. Error: 0.37

Trapezoidal rule (Implicit trapezoidal):

$y(1) = 1 + \frac{1}{2}\big[f(0,1) + f(1, y(1))\big] = 1 + \frac{1}{2}\big[-1 - y(1)\big]$

$y(1)$ appears on both sides! Solve: $y(1) = \frac{1}{3} \approx 0.333.$ Error: 0.035

From Quadrature to ODE Solvers

Example: $\dot{y} = -y,\ y(0) = 1,$ single step h=1. Exact: $y(1) = e^{-1} \approx 0.3679$.

Left rectangle (Forward Euler):

$y(1) = 1 + 1 \cdot f(0,\, 1) = 1 + (-1) = 0.$ Only needs $y(0)$ — no system to solve. Error: 0.37

Trapezoidal rule (Implicit trapezoidal):

$y(1) = 1 + \frac{1}{2}\big[f(0,1) + f(1, y(1))\big] = 1 + \frac{1}{2}\big[-1 - y(1)\big]$

$y(1)$ appears on both sides! Solve: $y(1) = \frac{1}{3} \approx 0.333.$ Error: 0.035

2-point Gauss–Legendre collocation:

Nodes: $s_1 \approx 0.211$, $s_2 \approx 0.789$. Unknowns: $Y_1 = y(s_1)$, $Y_2 = y(s_2)$.

Write the integral equation at each node, approximate by quadrature:

$$Y_k = 1 + \int_0^{s_k} (-y)\, ds \ \approx\ 1 - a_{k1}\, Y_1 - a_{k2}\, Y_2$$

(a_{kj}: quadrature weights for integrating to s_k.) This is a 2×2 linear system. Solve: $Y_1 \approx 0.814$, $Y_2 \approx 0.449$.

Then $y(1) = 1 - \frac{1}{2}(Y_1 + Y_2) \approx 0.368.$ Error: 5×10^{-4}

ODE Solvers Under the Hood

Every classical ODE solver is a quadrature rule applied to $y(t) = y_0 + \int_0^t f(s, y(s))\, ds$.

ODE solver	Underlying quadrature	Order
Forward Euler	Left rectangle	1
Backward Euler	Right rectangle	1
Implicit trapezoidal	Trapezoidal rule	2
Classical RK4	Simpson-like (composite)	4
GL collocation (s stages)	Gauss–Legendre (s nodes)	$2s$

ODE Solvers Under the Hood

Every classical ODE solver is a quadrature rule applied to $y(t) = y_0 + \int_0^t f(s, y(s))\, ds$.

ODE solver	Underlying quadrature	Order
Forward Euler	Left rectangle	1
Backward Euler	Right rectangle	1
Implicit trapezoidal	Trapezoidal rule	2
Classical RK4	Simpson-like (composite)	4
GL collocation (s stages)	Gauss–Legendre (s nodes)	$2s$

Explicit methods (Euler, RK4) evaluate f at points where y is already known — no system to solve.

Implicit methods (trapezoidal, GL collocation) require solving a system at each step — but achieve higher order for the same number of stages.

Laplace's Equation: A Fundamental PDE

Find u satisfying $\nabla^2 u = 0$ on a domain Ω, with prescribed values on the boundary $\Gamma = \partial\Omega$.

Laplace's Equation: A Fundamental PDE

Find u satisfying $\nabla^2 u = 0$ on a domain Ω, with prescribed values on the boundary $\Gamma = \partial\Omega$.

Physical meaning: the steady state in many settings.

Field	u represents	Boundary condition
Heat conduction	Temperature	Fixed temperature on walls
Electrostatics	Electric potential	Voltage on conductors
Fluid mechanics	Velocity potential	Flow around obstacles
Membrane	Displacement	Shape of wire frame

Laplace's Equation: A Fundamental PDE

Find u satisfying $\nabla^2 u = 0$ on a domain Ω, with prescribed values on the boundary $\Gamma = \partial\Omega$.

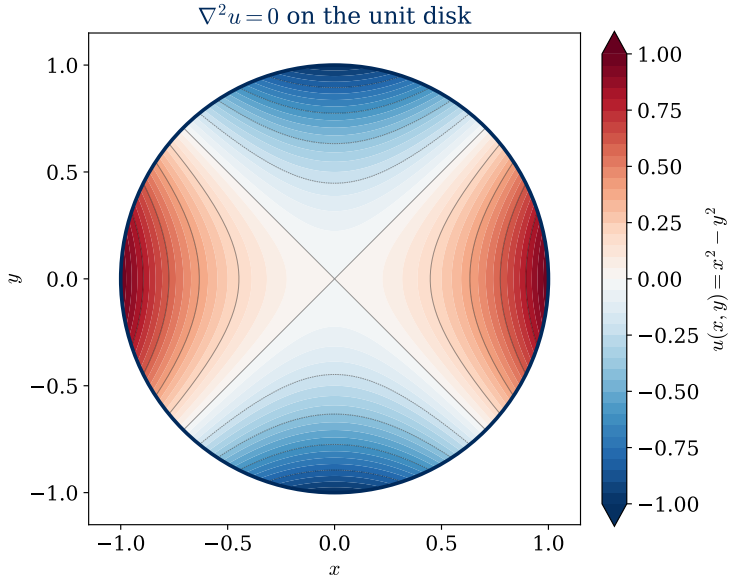
Physical meaning: the steady state in many settings.

Field	u represents	Boundary condition
Heat conduction	Temperature	Fixed temperature on walls
Electrostatics	Electric potential	Voltage on conductors
Fluid mechanics	Velocity potential	Flow around obstacles
Membrane	Displacement	Shape of wire frame

Key property (mean value): $u(\mathbf{x})$ = average of u on any circle around $\mathbf{x}$.

No interior maxima or minima — the "smoothest possible" interpolation.

A Harmonic Function



The smoothest interpolation of its boundary values.

The Standard Approach: Finite Element Method (FEM)

Mesh the *entire interior*, approximate u by piecewise polynomials.

The Standard Approach: Finite Element Method (FEM)

Mesh the *entire interior*, approximate u by piecewise polynomials.

Steps:

1. Triangulate Ω into elements
2. On each triangle: $u_h = a + bx + cy$
3. Assemble stiffness matrix K:

$$K_{ij} = \int_{\Omega} \nabla \phi_i \cdot \nabla \phi_j \, dA$$

(ϕ_i: piecewise linear basis function, $= 1$ at node i, $= 0$ at all others)

4. Apply boundary conditions, solve $Ku = f$

The Standard Approach: Finite Element Method (FEM)

Mesh the *entire interior*, approximate u by piecewise polynomials.

Steps:

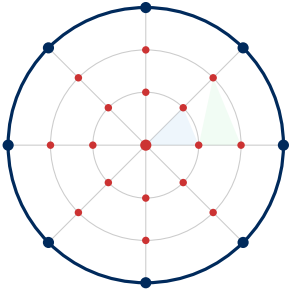
1. Triangulate Ω into elements
2. On each triangle: $u_h = a + bx + cy$
3. Assemble stiffness matrix K:

$$K_{ij} = \int_\Omega \nabla\phi_i \cdot \nabla\phi_j \, dA$$

(ϕ_i: piecewise linear basis function, $= 1$ at node i, $= 0$ at all others)

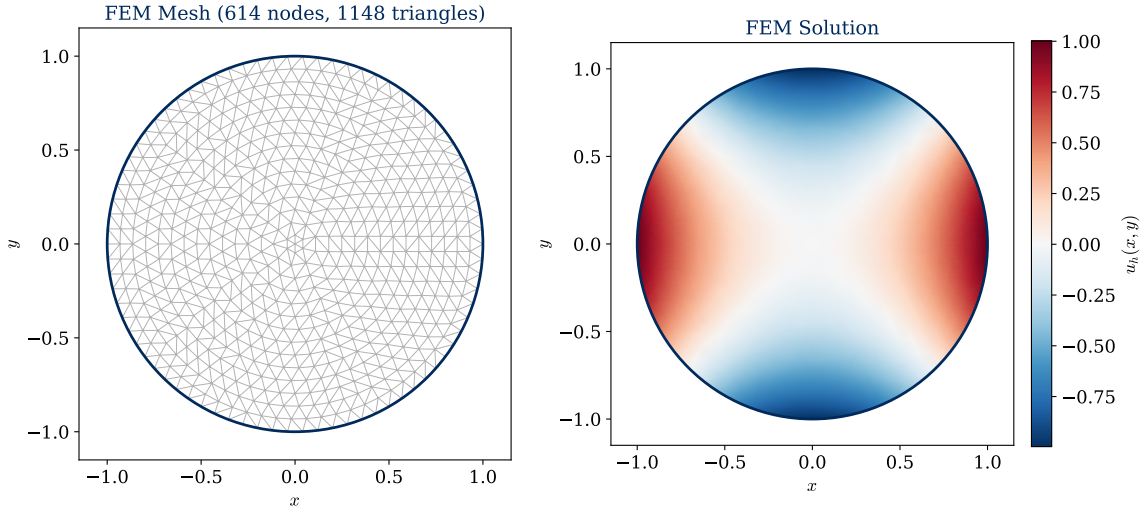
4. Apply boundary conditions, solve $Ku = f$

DOFs: $O(N^2)$ for a 2D domain.
K is usually very sparse, and hence solvable even for $N10^6$.



Must mesh the **whole** interior

FEM in Action



614 nodes, 1148 triangles — must fill the entire interior.

The Alternative: Boundary Element Method (BEM)

Laplace's equation has a Green's function:

$$G(\mathbf{x}, \mathbf{y}) = -\frac{1}{2\pi} \log |\mathbf{x} - \mathbf{y}| \qquad \text{(2D)}$$

The Alternative: Boundary Element Method (BEM)

Laplace's equation has a Green's function:

$$G(\mathbf{x}, \mathbf{y}) = -\frac{1}{2\pi} \log |\mathbf{x} - \mathbf{y}| \qquad \text{(2D)}$$

Represent u as a boundary integral (single-layer potential):

$$u(\mathbf{x}) = \int_\Gamma G(\mathbf{x}, \mathbf{y})\, \sigma(\mathbf{y})\, ds(\mathbf{y}), \qquad \mathbf{x} \in \Omega$$

Enforce $u = f$ on $\Gamma \Rightarrow$ solve an integral equation for the unknown density σ.

The Alternative: Boundary Element Method (BEM)

Laplace's equation has a Green's function:

$$G(\mathbf{x}, \mathbf{y}) = -\frac{1}{2\pi} \log |\mathbf{x} - \mathbf{y}| \qquad \text{(2D)}$$

Represent u as a boundary integral (single-layer potential):

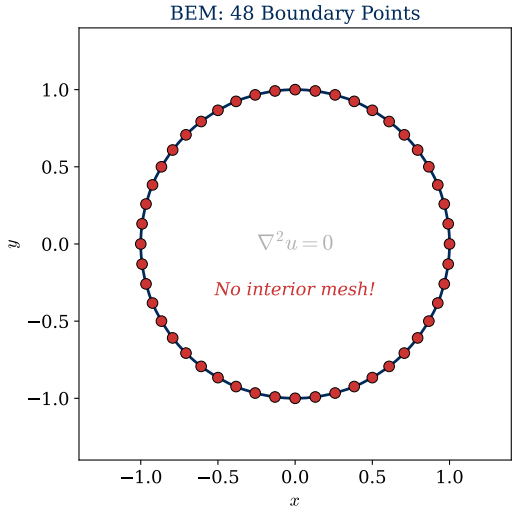
$$u(\mathbf{x}) = \int_\Gamma G(\mathbf{x}, \mathbf{y})\, \sigma(\mathbf{y})\, ds(\mathbf{y}), \qquad \mathbf{x} \in \Omega$$

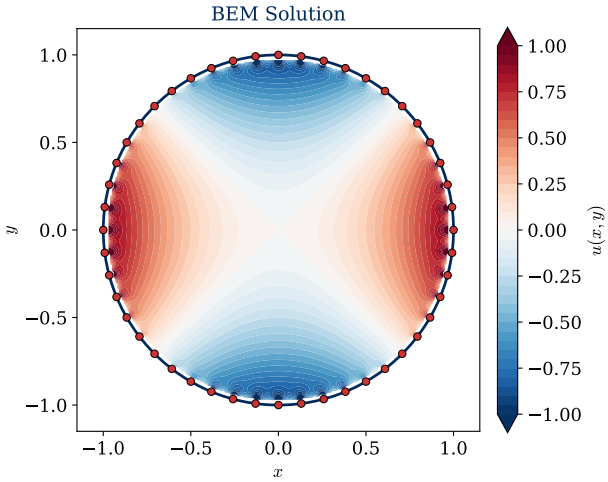
Enforce $u = f$ on $\Gamma \Rightarrow$ solve an integral equation for the unknown density σ.

FEM: mesh the interior
$O(N^2)$ DOFs in 2D

BEM: discretise the boundary
$O(N)$ DOFs

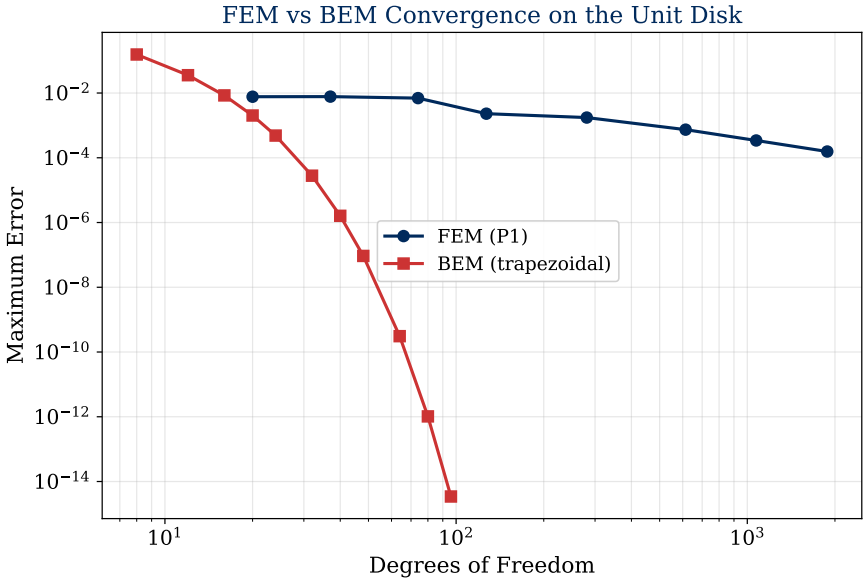
BEM in Action





Only 48 boundary points — no interior mesh needed.

FEM vs BEM: Convergence



FEM: algebraic. BEM: exponential — just add boundary points.

The Nyström Method: BIE $\rightarrow$ Linear System

Discretise the integral equation using quadrature:

$$\int_{\Gamma} G(\mathbf{x}_i, \mathbf{y})\, \sigma(\mathbf{y})\, ds(\mathbf{y}) \;\approx\; \sum_{j=1}^{N} \underbrace{w_j}_{\text{quad. weight}} \cdot G(\mathbf{x}_i, \mathbf{y}_j) \cdot \sigma(\mathbf{y}_j)$$

The Nyström Method: BIE $\rightarrow$ Linear System

Discretise the integral equation using quadrature:

$$\int_\Gamma G(\mathbf{x}_i, \mathbf{y})\, \sigma(\mathbf{y})\, ds(\mathbf{y}) \;\approx\; \sum_{j=1}^{N} \underbrace{w_j}_{\text{quad. weight}} \cdot G(\mathbf{x}_i, \mathbf{y}_j) \cdot \sigma(\mathbf{y}_j)$$

This gives a dense $N \times N$ linear system $A\boldsymbol{\sigma} = \mathbf{f}$, where $A_{ij} = w_j \cdot G(\mathbf{x}_i,\, \mathbf{y}_j)$.

The Nyström Method: BIE $\rightarrow$ Linear System

Discretise the integral equation using quadrature:

$$\int_\Gamma G(\mathbf{x}_i, \mathbf{y})\, \sigma(\mathbf{y})\, ds(\mathbf{y}) \;\approx\; \sum_{j=1}^{N} \underbrace{w_j}_{\text{quad. weight}} \cdot G(\mathbf{x}_i, \mathbf{y}_j) \cdot \sigma(\mathbf{y}_j)$$

This gives a dense $N \times N$ linear system $A\boldsymbol{\sigma} = \mathbf{f}$, where $A_{ij} = w_j \cdot G(\mathbf{x}_i,\, \mathbf{y}_j)$.

The quadrature nodes $\mathbf{y}_j$ (where we evaluate σ) and the collocation nodes $\mathbf{x}_i$ (where we enforce the equation) are the same points.

$\Rightarrow$ One set of N boundary points does everything.

The Nyström Method: BIE → Linear System

Discretise the integral equation using quadrature:

$$\int_\Gamma G(\mathbf{x}_i, \mathbf{y})\, \sigma(\mathbf{y})\, ds(\mathbf{y}) \;\approx\; \sum_{j=1}^{N} \underbrace{w_j}_{\text{quad. weight}} \cdot G(\mathbf{x}_i, \mathbf{y}_j) \cdot \sigma(\mathbf{y}_j)$$

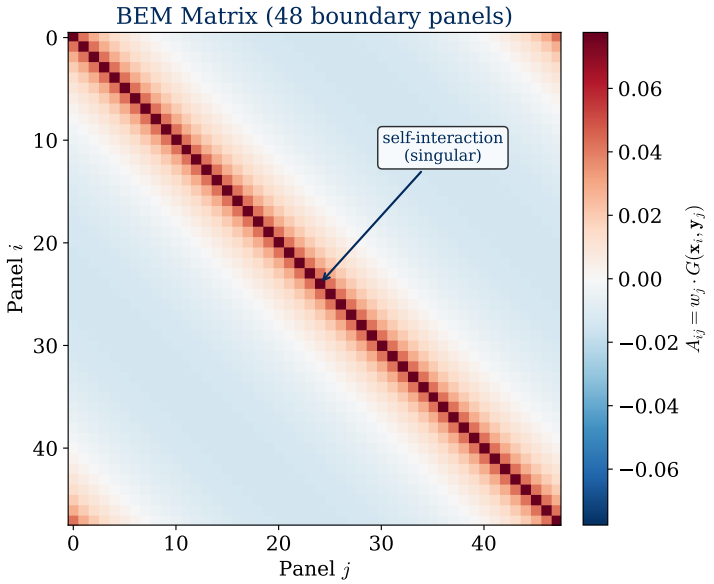
This gives a dense $N \times N$ linear system $A\boldsymbol{\sigma} = \mathbf{f}$, where $A_{ij} = w_j \cdot G(\mathbf{x}_i,\, \mathbf{y}_j)$.

The quadrature nodes $\mathbf{y}_j$ (where we evaluate σ) and the collocation nodes $\mathbf{x}_i$ (where we enforce the equation) are the same points.

$\Rightarrow$ One set of N boundary points does everything.

BEM's accuracy is entirely determined by its quadrature.
If the integrals are wrong, the matrix is wrong, the solution is wrong.

The BEM Matrix



Singularities live on and near the diagonal — this is where quadrature matters.

BEM: When Quadrature Becomes the Bottleneck

Each matrix entry $A_{ij} = \int_{\text{panel}_j} G(\mathbf{x}_i, \mathbf{y})\, ds$:

Far panels (i far from j):
Kernel is smooth $\Rightarrow$ standard GL works fine.

Self-interaction ($i = j$):
$\log|\mathbf{x} - \mathbf{y}| \to -\infty \Rightarrow$ singular!

Neighbouring panels:
Kernel sharply peaked $\Rightarrow$ nearly singular.

BEM: When Quadrature Becomes the Bottleneck

Each matrix entry $A_{ij} = \int_{\text{panel}_j} G(\mathbf{x}_i, \mathbf{y})\, ds$:

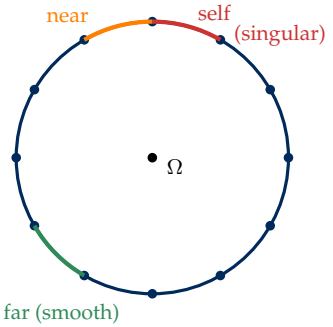
Far panels (i far from j):
Kernel is smooth $\Rightarrow$ standard GL works fine.

Self-interaction ($i = j$):
$\log|\mathbf{x} - \mathbf{y}| \to -\infty \Rightarrow$ singular!

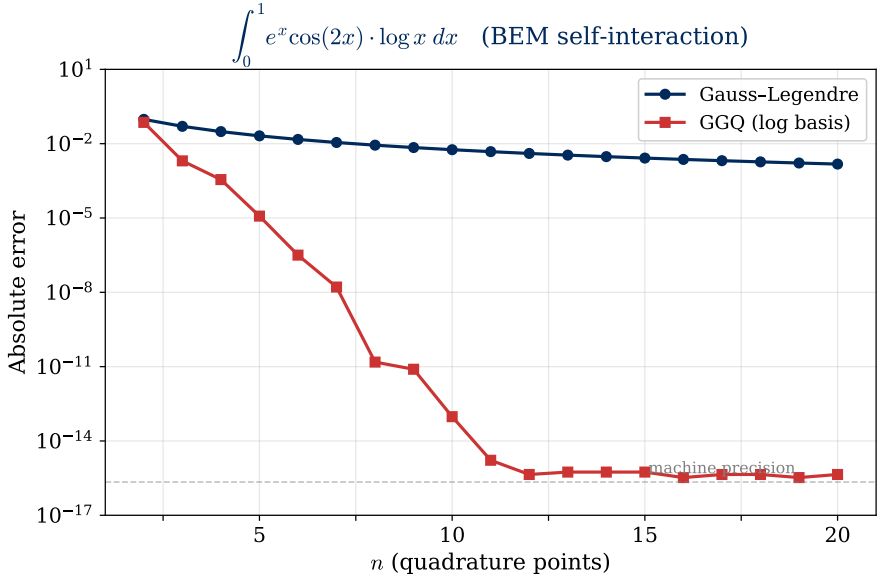
Neighbouring panels:
Kernel sharply peaked $\Rightarrow$ nearly singular.

For smooth closed curves, self-interactions can be handled analytically (i.e., solving by hand and just creating a function handle to evaluate the integral).

For general geometries, we need GGQ.



GL vs GGQ: A BEM Self-Interaction Integral



In more complex settings, e.g. domains with corners, intersections, and, electrode problems — log singularities appear in every BEM matrix entry.

We solve *thousands* of these integrals per problem.

The right quadrature, matched to the singularity, makes the difference between a method that stalls and one that reaches machine precision.

Thank you!

Questions?

UTM Math Club

Trefethen (2008), *Is Gauss Quadrature Better than Clenshaw–Curtis?*
Ma–Rokhlin–Wandzura (1996), *Generalised Gaussian Quadrature*
Huybrechs–Cools (2009), *GGQ for Singular Integrals*