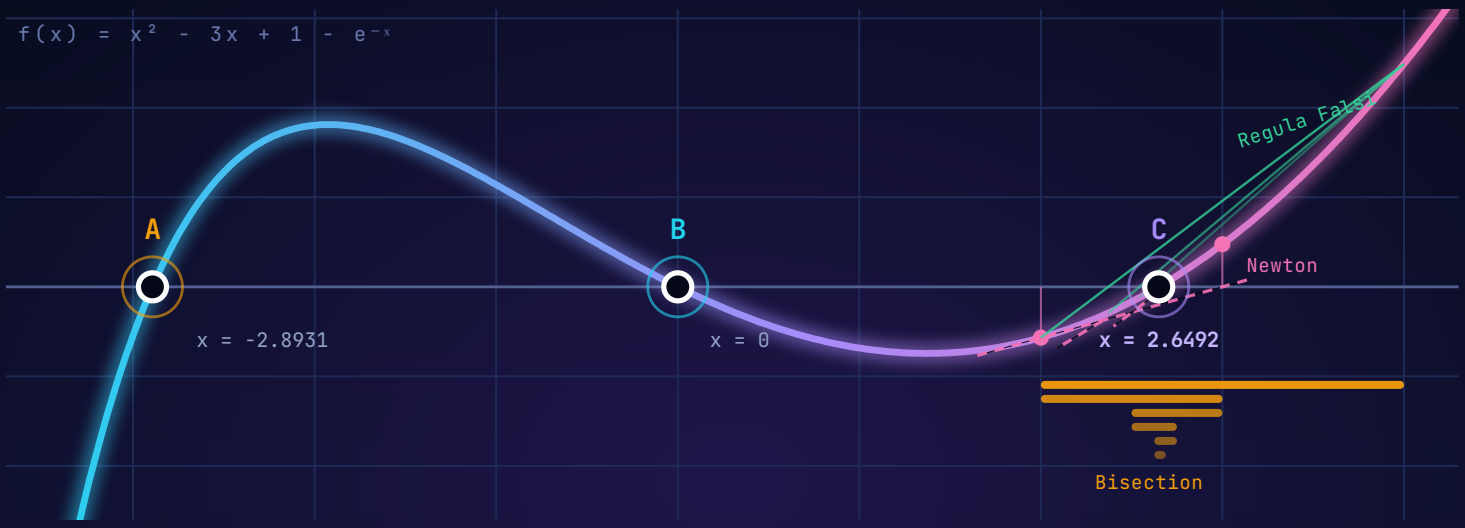
THE DIGITAL COLLISION

A numerical showdown: where does a drone's flight path meet a radar jammer's boundary?

Three root-finders, one equation, 25 pages of evidence.

BISECTION VS REGULA FALSI VS NEWTON-RAPHSON



CLASS: TANK

12

TURNS TO CLEAR

BISECTION

Never loses. Never fast.

SPD
SAFE
PREC

linear · ratio 0.5

CLASS: SUPPORT

8

TURNS TO CLEAR

REGULA FALSI

Smart chords, one stuck foot.

SPD
SAFE
PREC

linear · ratio 0.358

CLASS: SPEEDSTER ★ MVP

5

TURNS TO CLEAR

NEWTON–RAPHSON

Squares the error every hit.

SPD
SAFE
PREC

quadratic · c ≈ 0.407

BOSS // COLLISION POINT C x = 2.6492 y = 0.0707

$f(x) = x^2 - 3x + 1 - e^{-x} = 0$ · 3 bosses on the map, 1 in the bracket [2, 4]

PLAYER 1

MAHER BHATT

ID 25C21007

ITM (SLS) BARODA UNIVERSITY, VADODARA

▶ PRESS START TO SOLVE f(x) = 0

Contents & reader's guide

Everything in this report was produced by running the algorithms, not copied from a textbook

1 **The problem** — Drone path vs jammer boundary; three crossings — p. 4

2 **Mathematical foundations** — IVT, error, order of convergence, stopping rules — p. 5

3 **Bisection method** — The tortoise that never loses its way — p. 6

4 **Regula Falsi** — Bisection with a brain, and a stuck endpoint — p. 8

5 **Newton–Raphson** — Ride the tangent; square the error — p. 10

6 **Head-to-head results** — Results, tolerance sweep 1 to 12 decimals — p. 12

7 **Convergence rate, measured** — Theory against measurement; error constants — p. 14

8 **The true computational cost** — Evaluations, derivative step size, wall-clock — p. 16

9 **Starting-point sensitivity** — Basins on the real line and in the complex plane — p. 18

10 **Pros, cons and decision guide** — Side-by-side criteria and scorecard — p. 20

11 **Recommended design for a real drone** — Safeguarded Newton and six-method showdown — p. 21

12 **Verification and results table** — Independent checks, setup, submission table — p. 23

13 **Conclusion and formula sheet** — What to remember; references — p. 25

HOW TO READ THIS REPORT

Sections 3 to 5 explain one method each and end with a full iteration trace. Sections 6 to 9 compare them on speed, cost and risk. Sections 10 and 11 turn the evidence into a design decision. Section 12 lets a reader re-create every number.

COLOUR CODE

Bisection | Regula Falsi | Newton–Raphson

The same three colours are used in every chart and table.

Executive summary

The verdict in one page

12
Bisection
iterations · linear · ratio 0.5

8
Regula Falsi
iterations · linear · ratio 0.358

5
Newton–Raphson
iterations · quadratic · x0 = 2

Iterations to reach 3 decimal places on [2, 4] (Newton–Raphson starts at x0 = 2).

THE VERDICT

Newton–Raphson is the fastest and scales best. Tightening accuracy from 3 to 10 decimals costs Bisection 24 extra iterations and Regula Falsi 16, but Newton–Raphson only 1. It is also the only method of the three that can fail: started at x0 = −25 it wanders for 27 iterations and lands on a different crossing. Bisection is slow but cannot fail inside a valid bracket. A real safety system should combine them (Section 11).

	Bisection	Regula Falsi	Newton–Raphson
Iterations (3 / 6 / 10 dp)	12 / 22 / 36	8 / 15 / 24	5 / 5 / 6
Order of convergence	1 (linear)	1 (linear)	2 (quadratic)
Guaranteed to converge?	✔ yes, if bracketed	✔ yes, if bracketed	✘ no, depends on x0
Needs derivative?	✔ no	✔ no	✘ yes
Best used for	certainty	a quick upgrade on bisection	speed, once a good start is known

Five findings that shape the whole report

1. **There are three collisions, not one.** The curves cross at A (−2.8931, 18.0495), B (0, 1) and C (2.6492, 0.0707). The bracket [2, 4] isolates only C.
2. **Speed ranking:** Newton–Raphson (quadratic, error constant ≈ 0.407) > Regula Falsi (linear, ratio ≈ 0.358) > Bisection (linear, ratio exactly 0.5).
3. **Iterations are not the same as cost.** Newton needs f′ as well. With a central-difference derivative it costs 3 evaluations per step, so at 3 decimals it is not cheaper than Regula Falsi (15 vs 10). From 6 decimals up it wins clearly (Section 8).
4. **The starting point decides everything for Newton.** Of 2,401 start values in [−6, 6]: 41.3% reach the intended crossing C, 36.5% reach A, 22.1% reach B (Section 9).
5. **Hybrid beats all three.** A safeguarded Newton (bracket + Newton + bisection fallback) needs only 4 iterations for 3 decimals and 5 for 10, and cannot be fooled by a bad start (Section 11).

1 The problem

Where do the drone's path and the jammer boundary meet?

An autonomous drone flies along the parabola **y = x² − 3x + 1**. A radar jammer (or moving obstacle) creates an interference boundary described by the transcendental curve **y = e^−x**. The safety system must find the exact x-coordinate where the two paths intersect so it can trigger an evasive manoeuvre. A collision point is any x at which both heights are equal, so we move everything to one side and look for a zero:

$$x^2 - 3x + 1 = e^{-x} \iff f(x) = x^2 - 3x + 1 - e^{-x} = 0$$

Because the equation mixes a polynomial with an exponential, no algebraic formula solves it. That is exactly why numerical root finding exists. Newton–Raphson also needs the derivatives:

$$f'(x) = 2x - 3 + e^{-x} \qquad f''(x) = 2 - e^{-x}$$

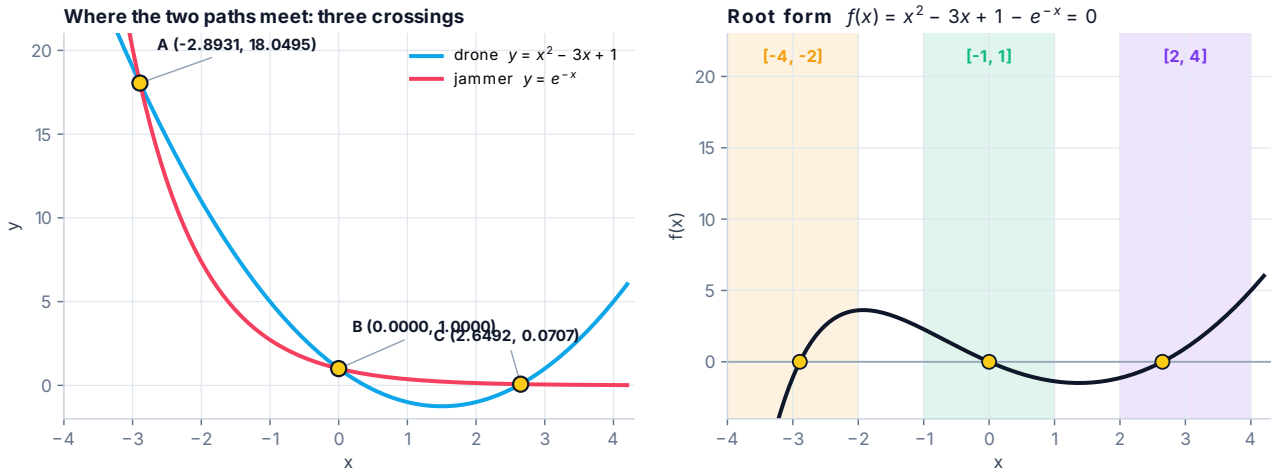


Figure 1: Left: the two curves and their three crossings. Right: the same problem as a root of f(x); shaded bands are valid sign-change brackets.

Crossing	x	y = e⁻ˣ	f′(x) there	Valid bracket	Notes
A	−2.893116	18.0495	+9.263	[−4, −2]	steep, far left
B	0 (exact)	1.0000	−2.000	[−1, 1]	exact root, f(0) = 0
C	**2.649220**	**0.0707**	**+2.369**	**[2, 4]**	**the root studied in this report**

WHY THIS MATTERS

The problem statement asks for "the exact coordinate". Mathematically there are three. This report studies C (the right-hand crossing on the forward flight path). Everything below uses C unless stated, but a production system must scan for *all* sign changes ahead of the drone first.

2 Mathematical foundations

The four ideas every method relies on

1. Intermediate Value Theorem (IVT)

If f is continuous on [a, b] and f(a), f(b) have opposite signs, at least one root lies inside. Bisection and Regula Falsi are built on this; Newton–Raphson is not.

$$f(a) \cdot f(b) < 0 \implies \exists\, r \in (a, b) :\ f(r) = 0$$

2. Error and tolerance

The error of an estimate x_n is $e_n = |x_n - r|$. The programs ask for n decimal places and stop at a tolerance of half a unit in the last place:

$$\varepsilon = 0.5 \times 10^{-n} \qquad (n = 3 \Rightarrow \varepsilon = 0.0005)$$

3. Order of convergence

A method has order p and asymptotic constant C if the error shrinks according to

$$\lim_{n \to \infty} \frac{e_{n+1}}{e_n^{\,p}} = C$$

Order	Name	What it does to the error	Digits gained per step
p = 1, C = 0.5	Linear (Bisection)	halves it	$\log_{10} 2 \approx 0.30$
p = 1, C ≈ 0.358	Linear (Regula Falsi, here)	multiplies it by 0.358	$-\log_{10} 0.358 \approx 0.45$
p = 2	Quadratic (Newton)	squares it (× constant)	doubles the number of correct digits

4. Stopping rules differ

Bisection stops when the bracket is narrower than ε, which is a rigorous error bound. **Regula Falsi** and **Newton–Raphson** stop when two successive estimates differ by less than ε, a practical rather than rigorous test. For Newton the true error is far smaller than the step size (quadratic convergence); for Regula Falsi the true error is about 0.56 × the last step here, so the rule still works.

QUICK GLOSSARY

Bracket: interval [a, b] with f(a)f(b) < 0

Root r: value with f(r) = 0

Residual: |f(x)|, which is not the same as the error

Convergence order: how fast errors die

Basin of attraction: set of start points leading to one root

Tolerance ε: half a unit in the last requested decimal

3 Bisection method

Cut the bracket in half. Always. The tortoise that never loses its way.

Take the midpoint c of [a, b]. Whichever half still has opposite signs at its ends keeps the root, so discard the other half and repeat. The bracket width halves exactly each step, so the error bound and the iteration count are known *before* the program runs.

$$c_n = \frac{a_n + b_n}{2}, \qquad |c_n - r| \le \frac{b-a}{2^n}, \qquad N \ge \log_2\left(\frac{b-a}{\varepsilon}\right)$$

For [2, 4] the starting width is b − a = 2, and with ε = 0.0005: N ≥ $\log_2(2 / 0.0005) = \log_2(4000) = 11.97$, so **12 iterations** are guaranteed. The program stops when the bracket is narrower than ε ($2 / 2^{12} = 0.000488 < 0.0005$), which is the same 12, and its final midpoint is 2.649.

12
iterations
for 3 decimals, known in advance

0.30
digits per step
$\log_{10} 2$, every single step

0.000488
final width
below ε = 0.0005

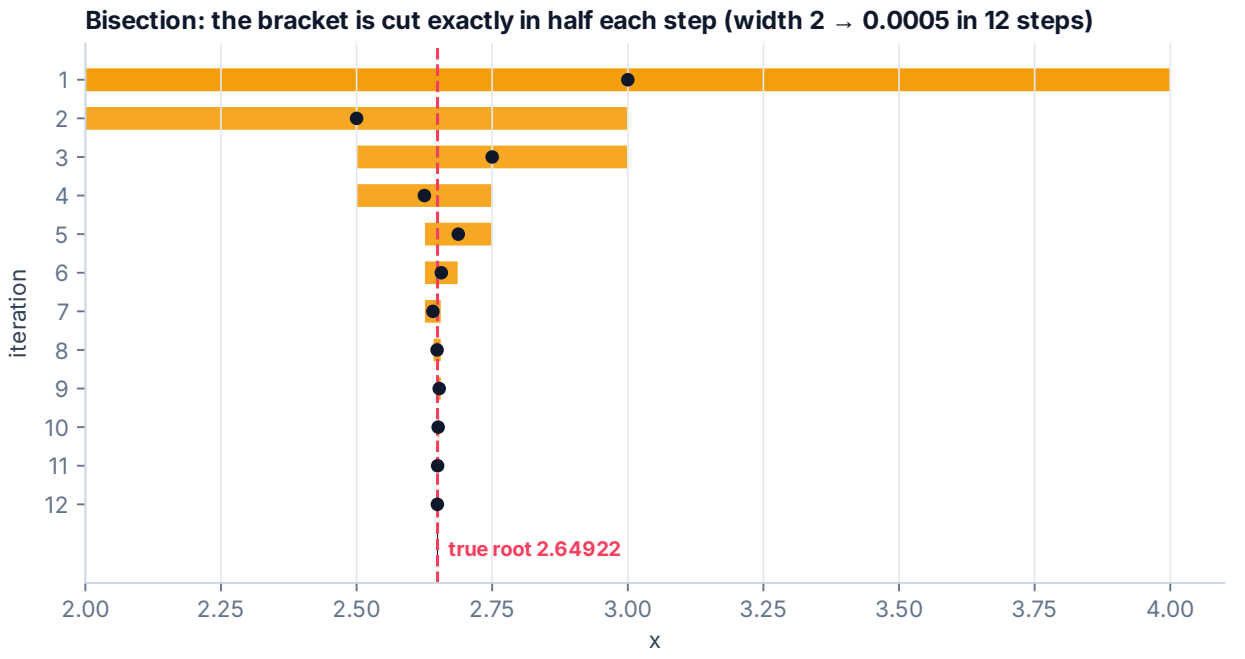


Figure 2: Each bar is the bracket after one iteration; the dot is that iteration's midpoint. The width shrinks 2 → 1 → 0.5 → … exactly by half each time.

WORKED BY HAND: THE FIRST THREE STEPS

f(2) = −1.135335, f(4) = +4.981684: opposite signs, so a root is inside. **c_1 = 3**: f(3) = +0.950213 > 0, same sign as f(4), so keep [2, 3]. **c_2 = 2.5**: f(2.5) = −0.332085 < 0, keep [2.5, 3]. **c_3 = 2.75**: f(2.75) = +0.248572 > 0, keep [2.5, 2.75]. The width is now 0.25 and the root 2.6492 is trapped.

Complete trace for [2, 4], 3 decimals

n	a_n	b_n	c_n (midpoint)	$f(c_n)$	width after step
1	2.000000	4.000000	3.000000	+0.95021	1.000000
2	2.000000	3.000000	2.500000	-0.33208	0.500000
3	2.500000	3.000000	2.750000	+0.24857	0.250000
4	2.500000	2.750000	2.625000	-0.05681	0.125000
5	2.625000	2.750000	2.687500	+0.09211	0.062500
6	2.625000	2.687500	2.656250	+0.01670	0.031250
7	2.625000	2.656250	2.640625	-0.02029	0.015625
8	2.640625	2.656250	2.648438	-0.00185	0.007812
9	2.648438	2.656250	2.652344	+0.00741	0.003906
10	2.648438	2.652344	2.650391	+0.00277	0.001953
11	2.648438	2.650391	2.649414	+0.00046	0.000977
12	2.648438	2.649414	2.648926	-0.00070	0.000488

Final bracket midpoint = 2.64916992, printed as 2.649 (true root 2.649220). Total: 12 iterations.

STRENGTHS

- ✔ Cannot fail once a sign change is bracketed (Intermediate Value Theorem)
- ✔ Iteration count and error bound are known in advance
- ✔ Needs only function values: no derivative, no smoothness beyond continuity
- ✔ Immune to flat tangents, tiny derivatives and wild overshoot
- ✔ Simple enough to verify and certify for safety-critical code

LIMITATIONS

- ✘ Slowest of the three: only 0.30 decimal digits gained per iteration
- ✘ Ignores how large f(c) is, so it throws away useful information
- ✘ Requires a bracket, and misses roots where f touches zero without crossing
- ✘ Error is not monotone (it zig-zags) even though the bracket always shrinks
- ✘ Does not extend naturally to systems of equations

Why it always works

The bracket $[a_n, b_n]$ always satisfies $f(a_n)\cdot f(b_n) < 0$ by construction. Its width is halved each step, so it shrinks to a single point, and by continuity that point must be a root. The a-priori iteration count for any tolerance is a one-line calculation:

Accuracy	$\varepsilon = 0.5 \times 10^{-n}$	$N = \lceil \log_2(2 / \varepsilon) \rceil$	Used by the program
3 decimals	0.0005	⌈11.97⌉ = 12	12
6 decimals	0.0000005	⌈21.93⌉ = 22	22
10 decimals	0.00000000005	⌈35.22⌉ = 36	36

The a-priori bound and the program agree for every accuracy, so the cost of Bisection can be quoted before running a single iteration.

4 Regula Falsi (false position)

Bisection with a brain: let the function values choose the cut.

Instead of the midpoint, draw the straight chord between (a, f(a)) and (b, f(b)) and use the point where it crosses the x-axis. Setting the chord's height to zero gives:

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

It keeps Bisection's safety (the root stays bracketed) but leans towards the end where |f| is smaller, so it usually gets closer, faster. **On this problem there is a catch:** f is convex on [2, 4] ($f'' = 2 - e^{-x} > 0$), so every chord lands *left* of the root and the right endpoint b = 4 never moves. Only one end of the bracket ever improves, so convergence is linear, but with a much better ratio than Bisection:

$$\frac{e_{n+1}}{e_n} \to 1 - \frac{f'(r)\,(b-r)}{f(b)} = 1 - \frac{2.3691 \times 1.3508}{4.9817} \approx 0.3576$$

The measured ratio in the run is **0.3577**, so theory and program agree. Each step removes ≈ 64% of the error, about 0.45 digits per iteration versus Bisection's 0.30.

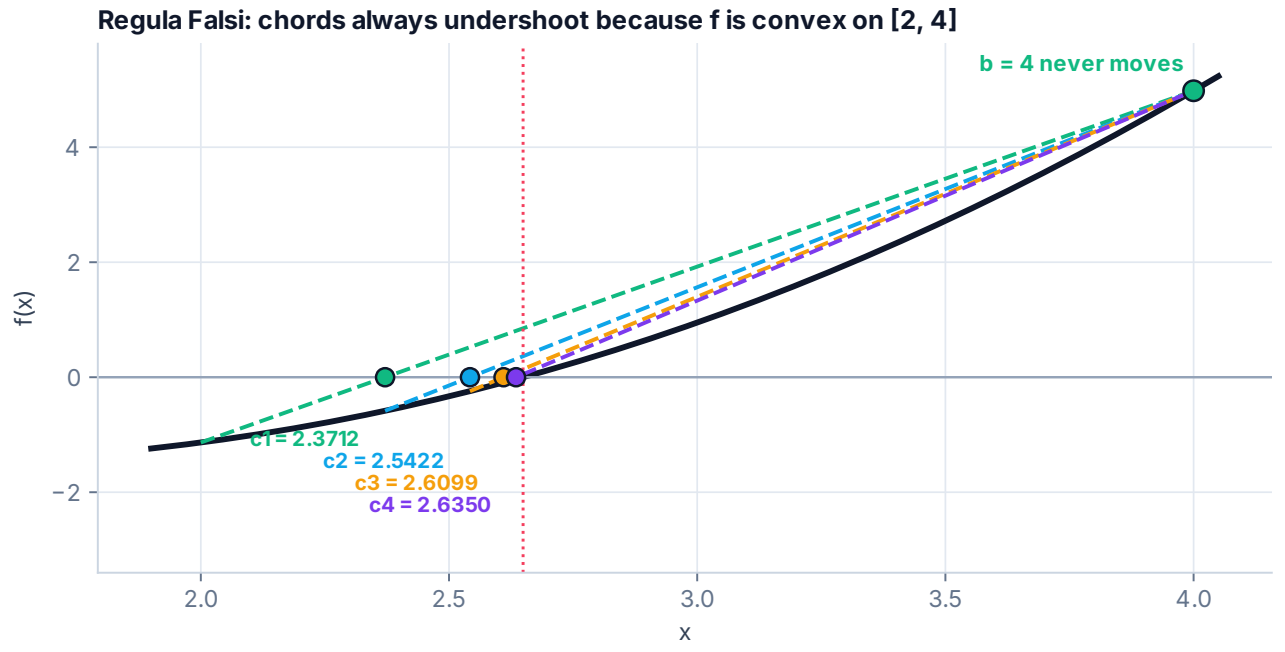


Figure 3: The first four chords. Every new estimate c falls left of the root while b = 4 stays frozen.

WORKED BY HAND: THE FIRST CHORD

With f(2) = −1.135335 and f(4) = +4.981684: $c_1 = (2 \cdot 4.981684 - 4 \cdot (-1.135335)) / (4.981684 - (-1.135335)) =$ **2.371205**. Since $f(c_1) < 0$ it replaces a, giving [2.3712, 4]. Notice c_1 is already far closer to the root (2.6492) than Bisection's first midpoint 3, because |f(2)| is much smaller than |f(4)|.

Complete trace for [2, 4], 3 decimals

n	c_n	error $\lvert c_n - r \rvert$	e_n / e_{n-1}
1	2.3712054	2.78e-01	–
2	2.5422094	1.07e-01	0.3849
3	2.6098760	3.93e-02	0.3677
4	2.6350072	1.42e-02	0.3612
5	2.6441188	5.10e-03	0.3589
6	2.6473933	1.83e-03	0.3581
7	2.6485664	6.54e-04	0.3578
8	2.6489862	2.34e-04	0.3577

Stops when $|c_n - c_{n-1}| < 0.0005$ at iteration 8: estimate 2.648986, printed 2.649 (true error 2.3e−04).

STRENGTHS

- ✔ Faster than Bisection here (8 vs 12 iterations) and the gap widens with precision (24 vs 36 at 10 decimals)
- ✔ Keeps the root bracketed, so it cannot diverge
- ✔ One new function evaluation per step, no derivative
- ✔ Exact in a single step if f is linear

LIMITATIONS

- ✘ One endpoint can stick (here b = 4 for every step), limiting it to linear convergence
- ✘ Ratio depends on curvature and can approach 1, making it slower than Bisection on badly shaped functions
- ✘ No guaranteed iteration count; the $|c_n - c_{n-1}|$ stopping rule is not a rigorous error bound
- ✘ Needs a bracket with a sign change

UPGRADE: THE ILLINOIS TRICK

Halving the stored f-value of the endpoint that gets stuck (Dowell & Jarratt, 1971) un-sticks it and raises the order to about 1.44. Measured on this problem: Illinois needs **6 / 8 / 8** iterations for 3 / 6 / 10 decimals, against 8 / 15 / 24 for plain Regula Falsi, and it is still bracketed and derivative-free.

Why the endpoint sticks

For a convex increasing function the chord lies *above* the curve, so it crosses zero left of the root, f(c) < 0 always, and c always replaces a. The endpoint b is never updated, so $e_{n+1} \approx 0.358 \cdot e_n$.

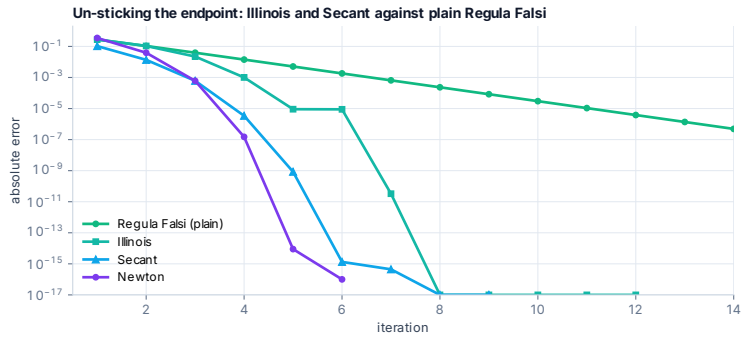


Figure 4: Error against iteration for plain Regula Falsi, Illinois, Secant and Newton. Illinois and Secant un-stick the method without a derivative.

5 Newton–Raphson method

Ride the tangent. Squares the error every step.

Replace f by its tangent line at x_n and jump to where that line hits zero. From the Taylor expansion $f(x) \approx f(x_n) + f'(x_n)(x - x_n)$:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \qquad e_{n+1} \approx \frac{f''(r)}{2f'(r)}\, e_n^2 = \frac{1.9293}{2 \times 2.3691}\, e_n^2 \approx 0.407\, e_n^2$$

Squaring the error means the number of correct digits roughly **doubles every step**. A central difference estimates f′ to about $h^2 = 10^{-12}$, so the program behaves like exact Newton–Raphson:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}, \qquad h = 10^{-6}$$

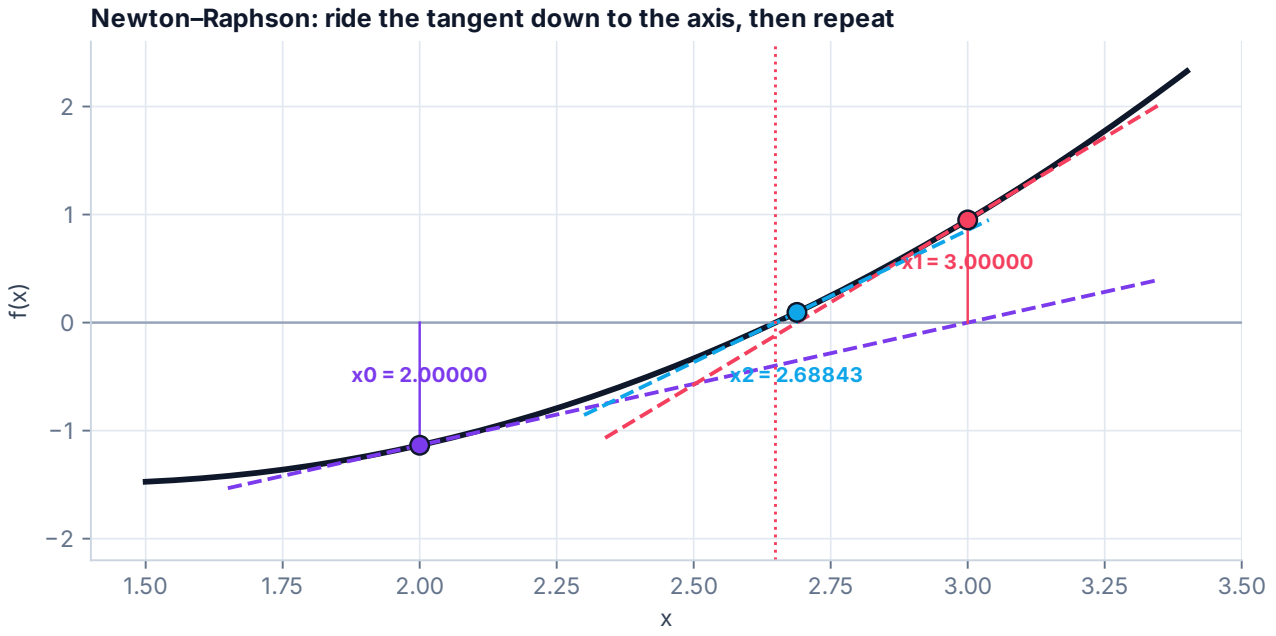


Figure 5: Three tangent steps from x0 = 2. The estimate overshoots to the right of the root and then snaps in.

WORKED BY HAND: A NICE COINCIDENCE

At x0 = 2: f(2) = −1 − e⁻² = −1.135335 and f′(2) = 1 + e⁻² = 1.135335, exactly opposite, so the first step is $x_1 = 2 - (-1) =$ **3.000000**. From there: f(3) = 0.950213, f′(3) = 3.049787, giving $x_2 = 3 - 0.311567 =$ **2.688433**. That is within 0.04 of the root after only two steps.

5
iterations
for 3 and 6 decimals (6 for 10)

×2
digits per step
correct digits double each iteration

10^{-15}
final error
machine precision after step 5

Complete trace from x0 = 2

n	x_n	error $\lvert x_n - r \rvert$	e_n / e_{n-1}^2	correct digits
0	2.0000000000	6.49e-01	–	≈ 0.2
1	2.9999999999	3.51e-01	0.8322	≈ 0.5
2	2.6884330248	3.92e-02	0.3187	≈ 1.4
3	2.6498271671	6.07e-04	0.3949	≈ 3.2
4	2.6492200379	1.50e-07	0.4070	≈ 6.8
5	2.6492198878	8.88e-15	0.3943	≈ 14.1

The constant e_{n+1}/e_n^2 climbs to 0.39, matching the theoretical 0.407. The 5th step reaches machine precision ($\approx 10^{-15}$).

STRENGTHS

- ✔ Quadratic convergence: 5 iterations for 3 decimals, 6 for 10 decimals
- ✔ Cost barely grows as accuracy tightens
- ✔ Needs only one starting value, not a bracket
- ✔ Generalises to systems (Jacobian) and underlies optimisation methods

LIMITATIONS

- ✘ No convergence guarantee: depends heavily on the starting point (Section 9)
- ✘ Fails or leaps when $f'(x) \approx 0$ (here near $x \approx -1.92$ and $x \approx 1.37$)
- ✘ Can land on a different root than intended, silently
- ✘ Needs f' (or extra evaluations to estimate it)
- ✘ Loses quadratic speed at repeated roots

DOES THE NUMERICAL DERIVATIVE HURT?

No. Replacing the central difference by the exact $f'(x) = 2x - 3 + e^{-x}$ gives identical iteration counts (5 / 5 / 6 for 3 / 6 / 10 decimals). The numerical version only costs more evaluations (Section 8).

Where the 0.407 comes from

Expanding f around the root r and substituting into the Newton update gives $e_{n+1} = [f''(\xi) / (2f'(x_n))] \cdot e_n^2$. As $x_n \to r$ the bracketed factor tends to $f''(r) / (2f'(r))$. Here $f''(r) = 2 - e^{-r} = 1.9293$ and $f'(r) = 2.3691$, so the constant is 0.407. The first iterates are far from r, which is why the measured ratios (0.83, 0.32, 0.39, 0.41, 0.39) only settle near the theoretical value in the last steps.

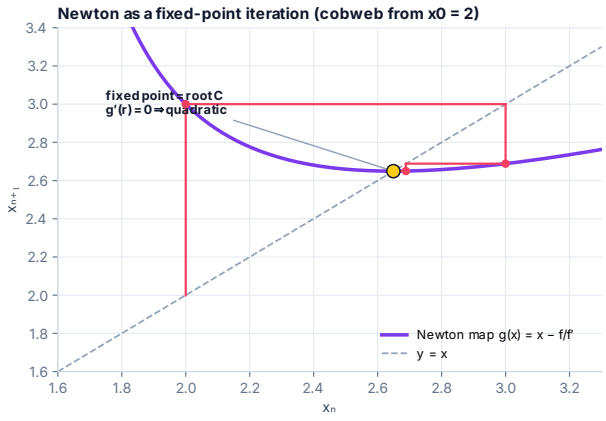


Figure 6: The Newton map g(x) = x − f(x)/f'(x). Its slope at the fixed point is zero, which is the geometric reason the convergence is quadratic.

6 Head-to-head results

Same function, same tolerance, three methods

Method	Start	3 dp	6 dp	10 dp	Final root (3 dp)	Crossing found
Bisection	[2, 4]	12	22	36	2.649	C ✔
Regula Falsi	[2, 4]	8	15	24	2.649	C ✔
Newton–Raphson	x0 = 3	4	4	5	2.649	C ✔
Newton–Raphson	x0 = 2	5	5	6	2.649	C ✔
Newton–Raphson	**x0 = −25**	**27**	**—**	**—**	**−2.893**	**A ✘**

Iteration counts come from running the programs in Section 12. The exact root is 2.6492198877672929...; the printed 2.649 is correct to 3 decimals in all runs that reached C.

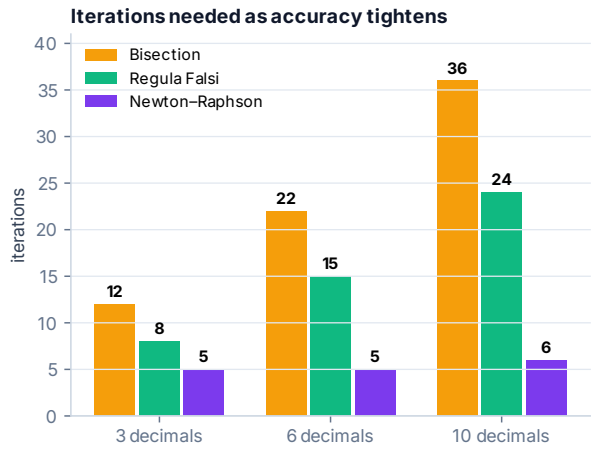


Figure 7: iterations.

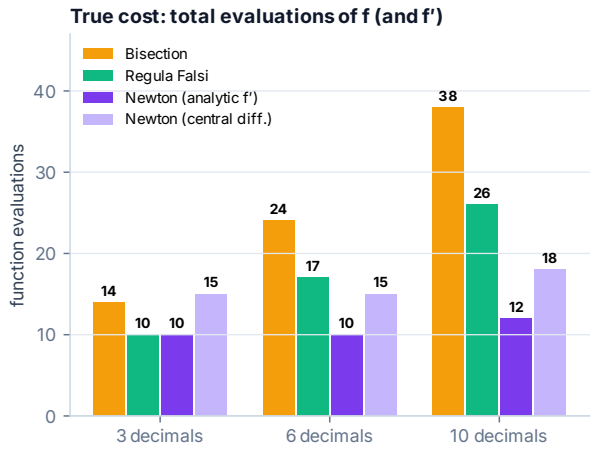


Figure 8: total function evaluations, which is what actually costs time and energy on a drone (Section 8).

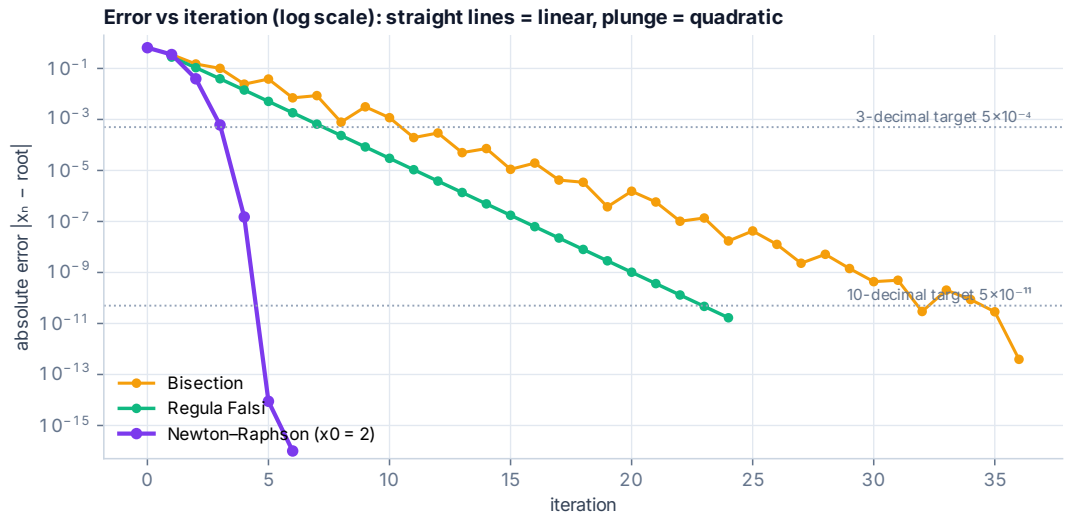


Figure 9: Error against iteration, log scale. A straight line is linear convergence; a curve bending downwards is quadratic. Bisection (amber) zig-zags; Regula Falsi (green) is a clean, steeper line; Newton–Raphson (violet) plunges to 10^{-15} in 5 steps.

Tolerance sweep: 1 to 12 decimal places

The three headline accuracies (3, 6, 10 decimals) hide the shape of the curves. Running every accuracy from 1 to 12 decimals shows the real picture: Bisection and Regula Falsi pay a roughly constant price per extra digit, while Newton–Raphson's count barely moves.

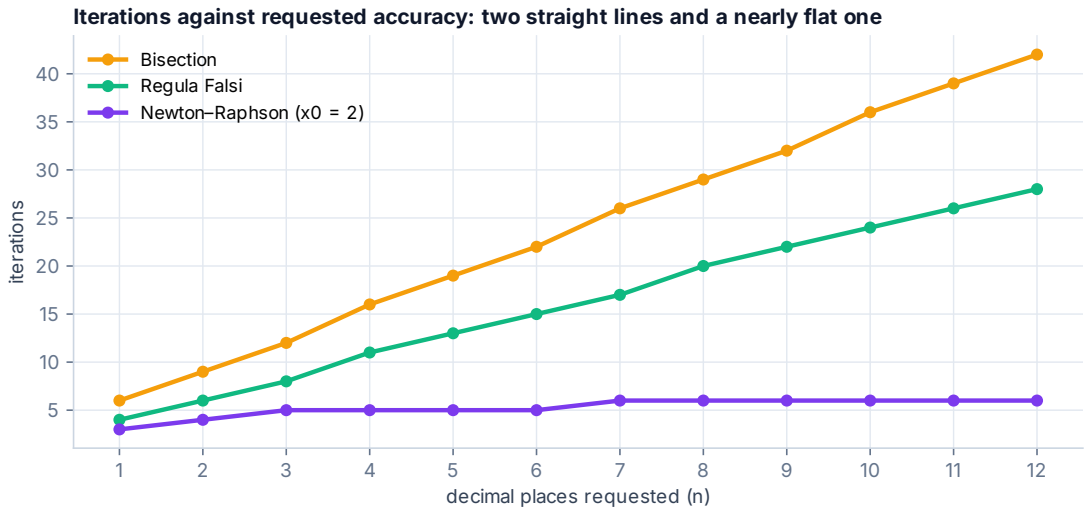


Figure 10: Iterations against requested accuracy (highlighted table rows = 3, 6, 10 decimals). Slopes: Bisection ≈ 3.3 iterations per decimal digit, Regula Falsi ≈ 2.2, Newton–Raphson ≈ 0.3.

Decimals n	Tolerance ε	Bisection	Regula Falsi	Newton–Raphson (x0 = 2)
1	5×10^{-2}	6	4	3
2	5×10^{-3}	9	6	4
3	**5×10^{-4}**	**12**	**8**	**5**
4	5×10^{-5}	16	11	5
5	5×10^{-6}	19	13	5
6	**5×10^{-7}**	**22**	**15**	**5**
7	5×10^{-8}	26	17	6
8	5×10^{-9}	29	20	6
9	5×10^{-10}	32	22	6
10	**5×10^{-11}**	**36**	**24**	**6**
11	5×10^{-12}	39	26	6
12	5×10^{-13}	42	28	6

WHAT THE SWEEP ADDS

From 1 to 12 decimals Bisection's cost grows from 6 to 42 iterations (+36), Regula Falsi's from 4 to 28 (+24), and Newton–Raphson's from 3 to 6 (+3). Bisection needs $\log_2 10 \approx 3.32$ iterations per extra digit by construction; the measured slope (3.27) is within 2% of that. This is the strongest single argument for a quadratic method when high precision is required.

7 Convergence rate, measured

Does the theory match the program? Yes.

The log–log plot below puts each error e_n on the x-axis and the next error e_{n+1} on the y-axis. The **slope** of the curve is the order of convergence p.

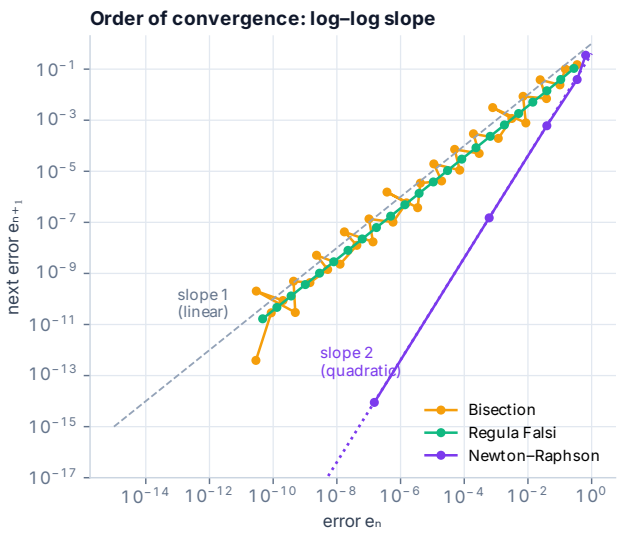


Figure 11: Order of convergence. The dotted violet line is the theory $e_{n+1} = 0.407\, e_n^2$.

Reading the plot

- Bisection and Regula Falsi hug the slope-1 line (linear).
- Newton–Raphson follows the slope-2 line (quadratic): doubling the exponent each step.
- The dotted violet line is the theory $e_{n+1} = 0.407\, e_n^2$; the data sit on it.

Method	Theory	Measured in the run	Digits gained / step
Bisection	$e_{n+1} = 0.5 \cdot e_n$ (bound)	bracket halves exactly	0.30
Regula Falsi	ratio → 0.3576	0.3577	0.45
Newton–Raphson	$e_{n+1} \approx 0.407\, e_n^2$	0.394 (last step)	doubles each step

Error exponents for Newton–Raphson from x0 = 2: 3.5×10^{-01} → 3.9×10^{-02} → 6.1×10^{-04} → 1.5×10^{-07} → 8.9×10^{-15}. The number of correct digits goes 0.5, 1.4, 3.2, 6.8, 14.1: it roughly **doubles each step**.

THE REAL ARGUMENT FOR NEWTON

Going from 3 to 10 decimals costs Bisection 24 extra iterations, Regula Falsi 16, and Newton–Raphson 1. Linear methods pay a fixed price *per digit*; a quadratic method pays a fixed price *per doubling of digits*.

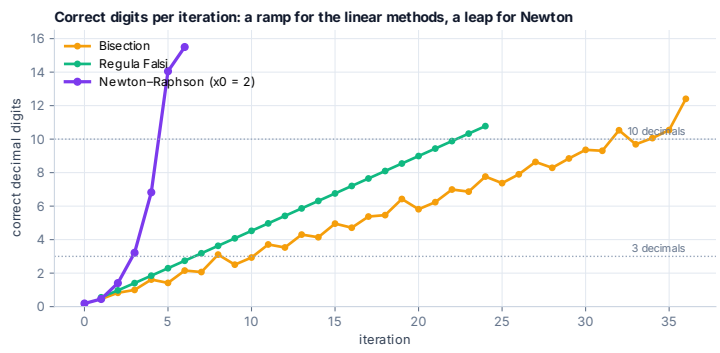


Figure 12: Correct decimal digits after each iteration.

The error constants, settling in

An asymptotic constant is a promise about the limit, so it is worth watching it arrive. The left panel shows Regula Falsi's step-to-step error ratio approaching the predicted 0.3576 from above; the right panel shows Newton's constant e_{n+1}/e_n^2 moving towards 0.407. Newton's last point is omitted once the error reaches round-off, where the ratio is no longer meaningful.

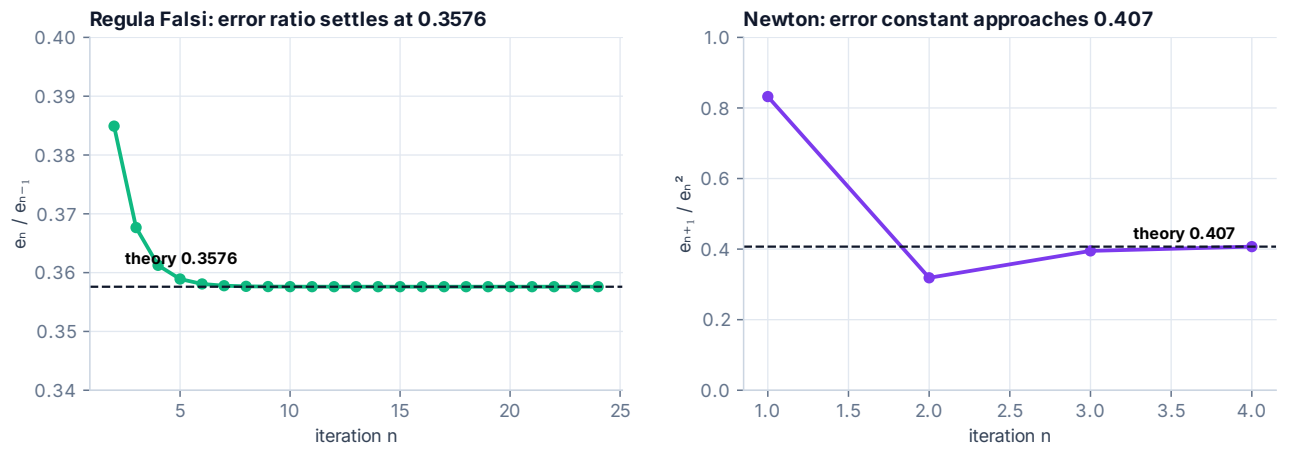


Figure 13: Measured error constants against theory.

0.3577
Regula Falsi ratio
measured; theory 0.3576

0.394
Newton constant
last stable step; theory 0.407

0.500
Bisection ratio
by construction, exact

Why the digits-per-step numbers differ

Method	Error update	Digits gained per step	Steps to gain 10 digits
Bisection	$e_{n+1} = e_n / 2$	$\log_{10} 2 = 0.301$	≈ 33
Regula Falsi (here)	$e_{n+1} \approx 0.358\, e_n$	$-\log_{10} 0.358 = 0.446$	≈ 22
Newton–Raphson	$e_{n+1} \approx 0.407\, e_n^2$	doubles (0.5, 1.4, 3.2, 6.8, 14.1)	≈ 4 to 5

A SUBTLE POINT ABOUT THE STOPPING RULE

Regula Falsi stops on $|c_n - c_{n-1}| < \varepsilon$. Because its error is a fixed fraction $r \approx 0.358$ of the previous error, the true error at the stopping step is about $r/(1-r) \approx 0.56$ times the last step, which is smaller than ε, so the test is safe here. For a function with ratio near 1 the same test would stop far too early. This is why the report treats it as practical, not rigorous.

WHAT THIS MEANS FOR THE DRONE

A guaranteed linear method has a predictable worst case, which a certifier likes. A quadratic method is fast on paper only while the iterate is close to the root; the first steps from a poor start are ordinary and sometimes wild. Section 9 quantifies that risk.

8 The true computational cost

Iterations are not the whole story

An iteration of Newton–Raphson is more expensive than one of the others because it needs the slope. Counting every evaluation of f (and of f′) as one unit, with the minimal implementation of each method:

Method	Evaluations per iteration	3 dp	6 dp	10 dp
Bisection	1 (+2 for the initial ends)	14	24	38
Regula Falsi	1 (+2 for the initial ends)	10	17	26
Newton, analytic f′	2 (f and f′)	10	10	12
Newton, central difference	3 (f, f(x+h), f(x−h))	15	15	18

WHAT THIS MEANS

At 3 decimals the numerically-differentiated Newton run (15 evaluations) is actually more expensive than Regula Falsi (10). At 6 decimals the picture changes (15 vs 17 vs 24), and at 10 decimals Newton–Raphson is the clear winner (18 vs 26 vs 38). With an analytic derivative $f'(x) = 2x - 3 + e^{-x}$ (which is cheap here) it is never worse than second place.

Measured wall-clock time

Method (3 decimals, printing off)	Best time per run	Relative
Bisection [2, 4]	6.1 µs	1.8×
Regula Falsi [2, 4]	5.4 µs	1.6×
Newton–Raphson x0 = 2	3.4 µs	1.0×

Best of 15 repeats × 3,000 runs on the machine that generated the original report. Absolute values depend on the computer and are tiny and noisy; only the ordering (Newton < Regula Falsi < Bisection) is meaningful, and it matches the iteration counts.

Beyond the three: derivative-free speed-ups

Method	Needs	3 dp	6 dp	10 dp	Comment
Plain Regula Falsi	bracket	8	15	24	order 1
Illinois (modified RF)	bracket	6	8	8	order ≈ 1.44, bracketed
Secant (x0 = 2, x1 = 3)	two starts	5	6	7	order ≈ 1.62, 1 eval/step, not bracketed
Safeguarded Newton (Section 11)	bracket + f′	4	4	5	bracketed and fast

Error against work done

Plotting error against *iterations* flatters Newton, because each of its iterations does more. A fairer axis is the cumulative number of function evaluations. On that axis Newton with a central difference starts behind (it spends three evaluations before moving), but its vertical drop is so steep that it overtakes both linear methods well before 10^{-6} accuracy.

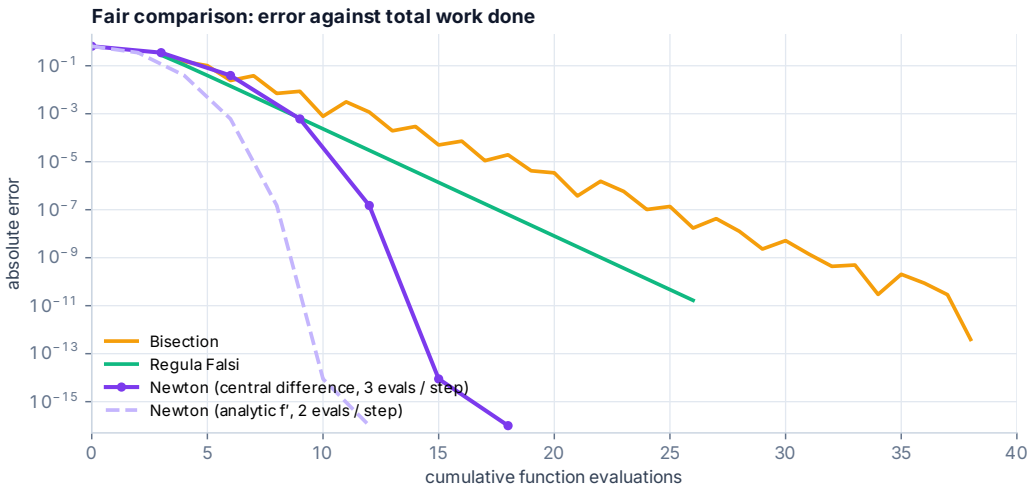


Figure 14: Error against cumulative function evaluations. Bisection and Regula Falsi include their two initial endpoint evaluations.

Choosing the step size h for the numerical derivative

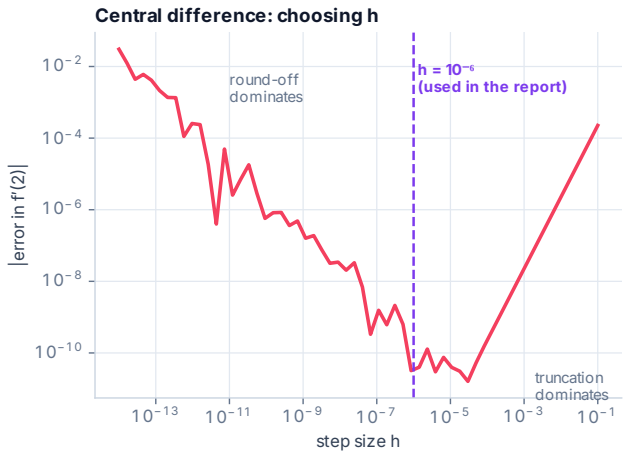


Figure 15: Error of the central difference for f′(2) as h varies.

A central difference has two competing errors: a **truncation** error that shrinks like h^2, and a **round-off** error that grows like ε_{mach}/h. Their sum is smallest near $h \approx \varepsilon_{mach}^{1/3} \approx 10^{-5}$ to 10^{-6}, which is why $h = 10^{-6}$ is a sensible default.

At that step the derivative is accurate to roughly 10^{-10}, far below what Newton needs, so the iteration counts are identical to those obtained with the exact derivative.

$$\underbrace{\frac{h^2}{6}\,|f'''|}_{\text{truncation}} + \underbrace{\frac{\varepsilon_{\text{mach}}\,|f|}{h}}_{\text{round-off}} \quad \text{is smallest near} \quad h \approx \varepsilon_{\text{mach}}^{1/3}$$

ENGINEERING TAKEAWAY

If f′ is cheap and known (as here), use it: two evaluations per step instead of three and no tuning parameter. If f′ is unavailable, the secant method gives 1.62-order convergence at one evaluation per step and removes the step-size question entirely.

9 Starting-point sensitivity & failure modes

Where Newton–Raphson earns its reputation for danger

Started at 3, Newton–Raphson needs 4 iterations; started at −25 it needs 27 and returns −2.893, which is crossing A, *not* the crossing C found by the other methods. This is the central practical weakness of the method.

x0	iters	lands on	verdict
3	4	C (2.6492)	intended
2	5	C (2.6492)	intended
4	5	C (2.6492)	intended
10	6	C (2.6492)	intended
1.2	6	A (−2.8931)	wrong root
0.5	4	B (0)	wrong root
−1	2	B (0)	wrong root
−4	5	A (−2.8931)	wrong root
−25	27	A (−2.8931)	wrong root

Classic failure modes

Flat tangent: if f′(x) ≈ 0 (here near x ≈ −1.92 or 1.37) the step is huge and can throw the estimate anywhere.

Wrong root: nothing checks which root was found, as the −25 start shows.

Cycling: iterates can bounce forever; an iteration cap is the only protection.

Repeated roots: convergence degrades from quadratic to linear.

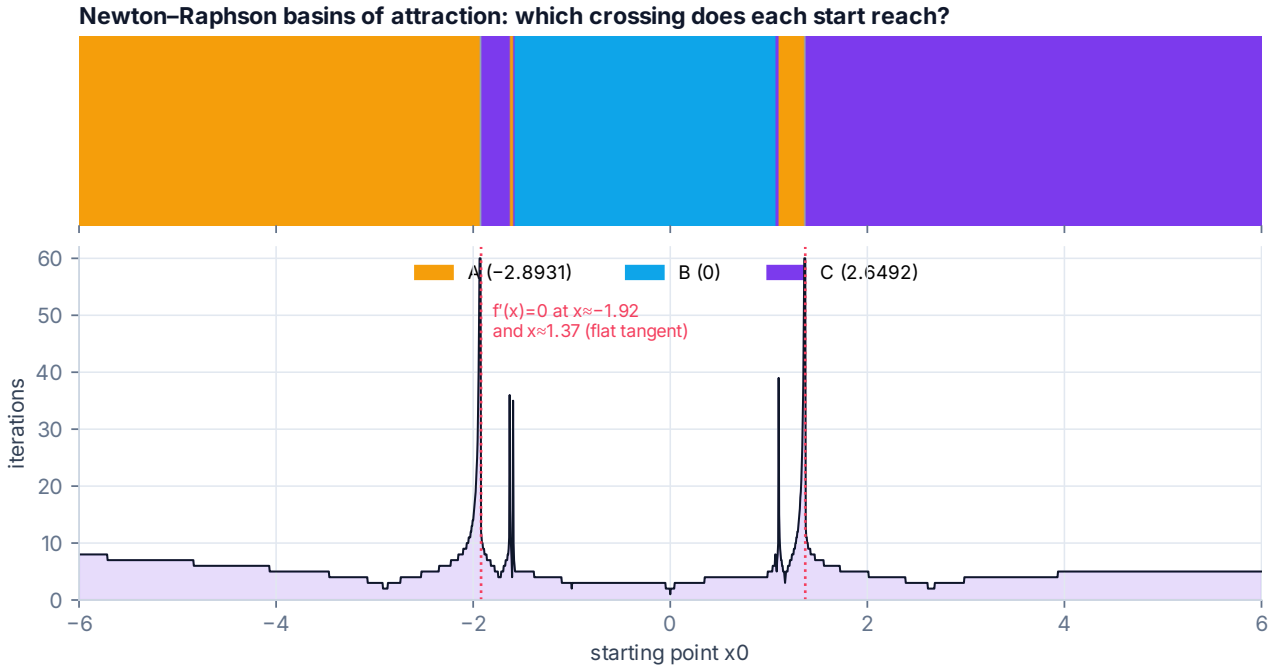


Figure 16: Top: the root each start value reaches (2,401 starts in [−6, 6]): 41.3% reach C, 36.5% A, 22.1% B, 0.2% never settle. Bottom: iterations; spikes appear where f′(x) = 0.

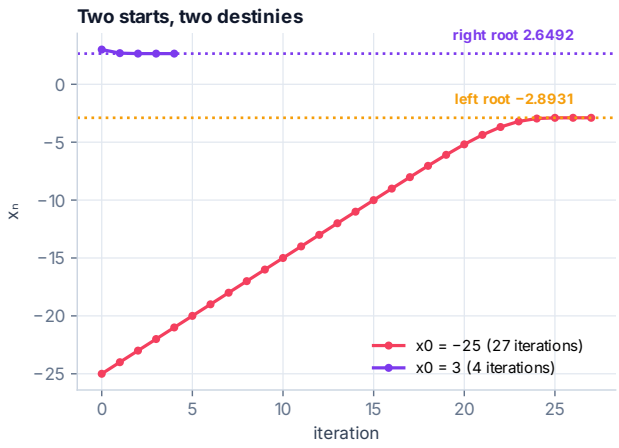


Figure 17: Two starts, two destinies.

Why exactly 27 iterations from −25?

For very negative x the exponential dominates: f(x) ≈ −e⁻ˣ and f′(x) ≈ +e⁻ˣ. The Newton step then simplifies to

$$x_{n+1} = x_n - \frac{-e^{-x_n}}{e^{-x_n}} = x_n + 1$$

so the estimate crawls rightwards by almost exactly +1 per step (−24, −23, −22.0000001, …), about 19 iterations of crawling, before quadratic convergence finally kicks in near A.

The same iteration in the complex plane

Newton's method does not care whether its iterate is real. Starting it from every point z_0 of a grid in the complex plane and colouring each point by the root it reaches turns the "basin of attraction" idea from a line into a picture. Because e^{-z} has infinitely many complex roots, there are three real roots (A, B, C) plus a lattice of complex ones; points that converge to a complex root are shaded in the warm magma palette.

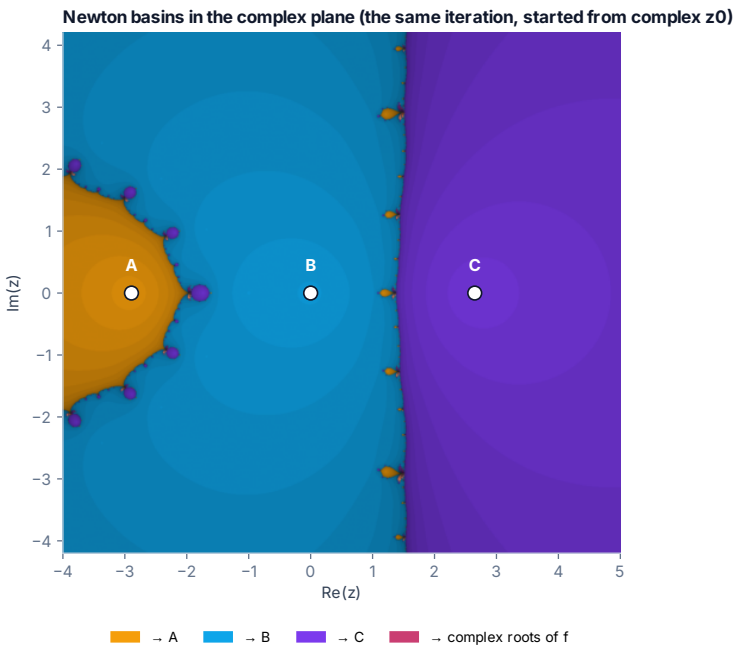


Figure 18: Basins of attraction of Newton–Raphson for $f(z) = z^2 - 3z + 1 - e^{-z}$ over a 1,100 × 924 grid (60 iterations). Brightness encodes how quickly each start point converges.

WHAT THE PICTURE SHOWS

The real axis (Im z = 0) is a thin slice through this plane: the three coloured regions you cross while walking along it are exactly the three bands of Figure 16. Away from the axis the boundaries between basins are not smooth curves but curved and, close to the real axis, jagged fronts.

WHY IT MATTERS FOR SAFETY

Near a basin boundary an arbitrarily small change in the starting value changes the answer. A real-valued start can sit very close to such a boundary, so "it worked in testing" is not evidence that Newton alone is safe. A bracket removes the problem.

10 Pros, cons and the decision guide

Everything side by side

Criterion	Bisection	Regula Falsi	Newton–Raphson
Order of convergence	1, ratio 0.5	1, ratio ≈ 0.358 here	2, constant ≈ 0.407
Iterations: 3 / 10 decimals	12 / 36	8 / 24	5 / 6
Digits gained per iteration	0.30	0.45	doubles
Function evaluations per step	1	1	2 to 3
Guaranteed convergence	✔ yes	✔ yes	✘ no
Needs a sign-change bracket	✘ yes	✘ yes	✔ no (one start)
Needs the derivative	✔ no	✔ no	✘ yes
Iteration count known in advance	✔ yes, exactly	✘ no	✘ no
Rigorous error bound while running	✔ yes (bracket width)	✘ no	✘ no
Sensitive to starting point	✔ no	✔ no	✘ very
Can jump to a different root	✔ never	✔ never	✘ yes
Handles non-smooth f	✔ yes	✔ mostly	✘ no
Extends to systems of equations	✘ no	✘ no	✔ yes
Typical role	safety net	quick bracketed solver	fast finisher

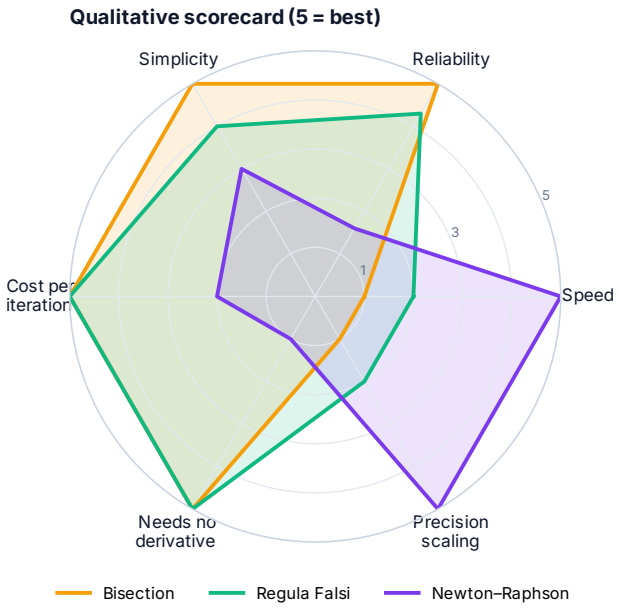


Figure 19: Qualitative scorecard, scored 1 to 5 by the author (not a measured quantity).

Which method should you use?

- **You can bracket the root and need certainty** → Bisection.
- **You can bracket it and want fewer steps with no derivative** → Regula Falsi, or better, Illinois.
- **You have a good start and f′ is cheap** → Newton–Raphson.
- **The result controls a vehicle, a patient or a bridge** → a safeguarded hybrid (Section 11).

11 Recommended design for a real drone

Bracket first, Newton second, Bisection as the parachute

The three methods have complementary strengths, so a safety system should combine them. The standard recipe (a safeguarded or bracketed Newton method, as in *Numerical Recipes*' rtsafe) works like this:



Scan the corridor on a grid for sign changes (one bracket each for A, B, C). **Try Newton** from the bracket midpoint and accept the step if it stays inside the bracket. **Fall back** to bisection otherwise, so the bracket shrinks every iteration and convergence is guaranteed.

```
def safeguarded_newton(f, df, a, b, tol=5e-4, max_iter=100):
    fa, fb = f(a), f(b)
    if fa * fb > 0:
        raise ValueError("no sign change on [a, b]")
    x = 0.5 * (a + b)
    for k in range(1, max_iter + 1):
        fx, d = f(x), df(x)
        if fx == 0:
            return x, k
        if fa * fx < 0: b, fb = x, fx    # root is in [a, x]
        else:           a, fa = x, fx    # root is in [x, b]
        x_new = x - fx / d if d ≠ 0 else None
        if x_new is None or not (a < x_new < b):
            x_new = 0.5 * (a + b)            # Newton escaped: bisect
        if abs(x_new - x) < tol:
            return x_new, k
        x = x_new
    return x, max_iter
```

Run	Result
[2, 4], 3 / 6 / 10 decimals	4 / 4 / 5 iterations → x = 2.6492199 (crossing C)
[−4, −2], 3 decimals	3 iterations → x = −2.8931 (crossing A)
[−5, 5], no sign change	raises an error (correct: f(−5), f(5) > 0 although roots exist), which is why scanning matters

Trace of the hybrid on [2, 4], 3 decimals

n	x_n	bracket after update	step used	x_{n+1}
1	3.0000000	[2.00000, 3.00000]	Newton	2.6884330
2	2.6884330	[2.00000, 2.68843]	Newton	2.6498272
3	2.6498272	[2.00000, 2.64983]	Newton	2.6492200
4	2.6492200	[2.00000, 2.64922]	Newton	2.6492199

RESULT

The hybrid matches Newton–Raphson's speed (4 iterations from [2, 4], versus 5 for plain Newton from x0 = 2 and 12 for Bisection), keeps Bisection's guarantee, and cannot be fooled by a bad start, a flat tangent, or a crossing it was not asked about.

Six methods, one problem

To place the three required methods in context, three further methods were run on the same equation with the same tolerance: Illinois (a repaired Regula Falsi), the Secant method (Newton without a derivative) and the safeguarded Newton hybrid.

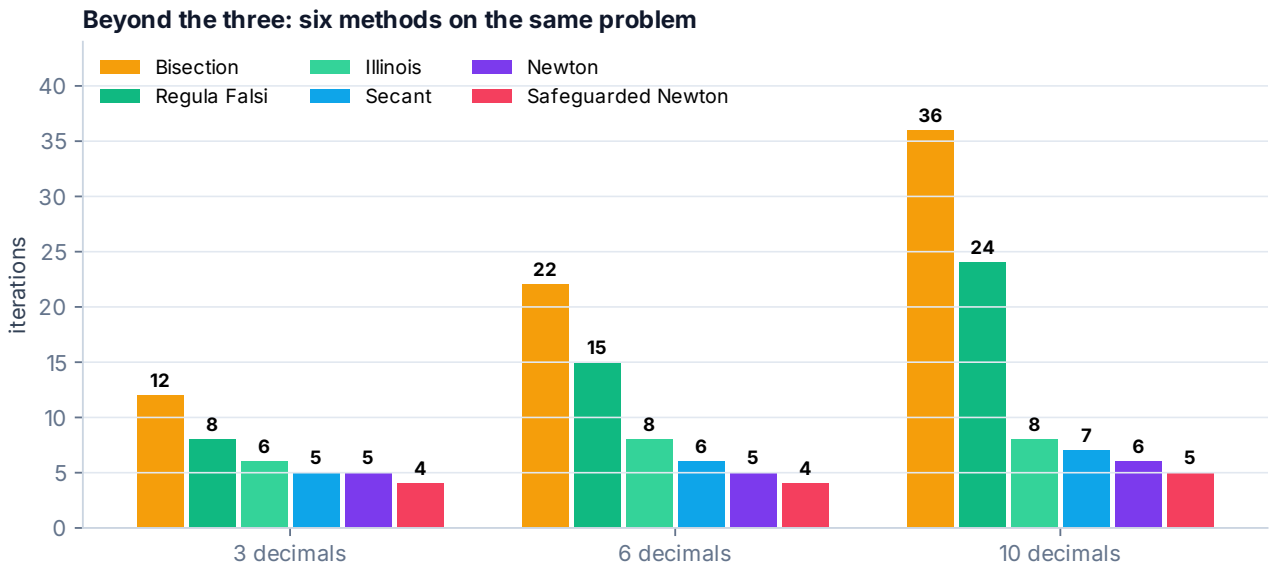


Figure 20: Iterations to reach 3, 6 and 10 decimals for six methods on [2, 4] (Secant starts at 2 and 3, Newton at 2).

Method	Needs	Order	Evaluations / step	Guaranteed?	Iterations (3 / 6 / 10 dp)
Bisection	bracket	1	1	✔	12 / 22 / 36
Regula Falsi	bracket	1	1	✔	8 / 15 / 24
Illinois	bracket	≈ 1.44	1	✔	6 / 8 / 8
Secant	2 starts	≈ 1.62	1	✘	5 / 6 / 7
Newton–Raphson	1 start + f′	2	2 to 3	✘	5 / 5 / 6
Safeguarded Newton	**bracket + f′**	**2 (after bracketing)**	**2**	✔	**4 / 4 / 5**

READING THE RESULT

Every step up the table buys speed at the price of an assumption: a bracket, then a good start, then a derivative. The hybrid is the only row that keeps the guarantee *and* reaches the speed of the quadratic method.

INDUSTRY PRACTICE

General-purpose library routines (Brent's method in many numerical packages) follow the same philosophy: bracket-safe, with Secant or inverse-quadratic steps whenever they behave and Bisection otherwise.

12 Verification and results table

Every number re-computed independently, with a high-precision reference

All results were produced by running the three algorithms in Python (code on the next page), and the roots were checked against a 40-digit solution (mpmath): C = 2.649219887767292965348 and A = −2.893116430925271220316; B = 0 exactly. Iteration counts, error ratios, evaluation counts, basin percentages and timings in this report all come from that one script, so nothing here is copied from memory.

Checks that passed

Check	Expected	Obtained
Bisection iterations, 3 dp	$\lceil \log_2(2/0.0005) \rceil = 12$	12 ✔
Regula Falsi error ratio	$1 - f'(r)(b-r)/f(b) = 0.3576$	0.3577 measured ✔
Newton error constant	$f''(r)/(2f'(r)) = 0.407$	0.394 measured (last stable step) ✔
Newton: analytic vs central difference	same counts	5 / 5 / 6 vs 5 / 5 / 6 ✔
Roots A, B, C	$f(r) = 0$	verified at 40 digits ✔
Basins over 2,401 starts	41.3 / 36.5 / 22.1 / 0.2 %	reproduced exactly ✔

Results table (ready to paste into a submission)

Name of method	Initial values	No. of iterations	Final root	Execution time (µs)
1. Bisection	a = 2.0, b = 4.0	12	2.649	6.1
2. Regula Falsi	a = 2.0, b = 4.0	8	2.649	5.4
3. Newton–Raphson	$x_0 = 2.0$	5	2.649	3.4

Times: best of 15 × 3,000 runs with printing off, on the machine used to build the original report. Re-measure on your own computer and quote the unit.

Experimental setup and assumptions

Item	Setting
Tolerance	$\varepsilon = 0.5 \times 10^{-n}$ for n correct decimals
Brackets / starts	Bisection and Regula Falsi on [2, 4]; Newton from x0 = 2 (also 3, 4, 10, −25 ...)
Derivative	central difference, $h = 10^{-6}$ (analytic f′ used for comparison)
Stopping rules	bracket width (Bisection); $\lvert\Delta x\rvert < \varepsilon$ (Regula Falsi, Newton)

WRITE-UP CHECKLIST

1. Always state which crossing you found (A, B or C), because a Newton start such as −25 silently lands on A. 2. Use the same stopping rule for every method or say that they differ. 3. Quote the unit for execution time and how it was measured. 4. Report iteration counts from your own run. 5. Use the names “Regula Falsi” and “Newton–Raphson”.

Reproduce everything

Run `python reproduce.py` and compare with the expected output

```python
import math
f  = lambda x: x*x - 3*x + 1 - math.exp(-x)      # f(x) = x^2 - 3x + 1 - e^-x
df = lambda x: 2*x - 3 + math.exp(-x)             # f'(x)

def bisection(a, b, n):
    tol, k = 0.5 * 10**-n, 0
    while abs(b - a) > tol:
        c = (a + b) / 2; k += 1
        if f(a) * f(c) < 0: b = c
        else: a = c
    return (a + b) / 2, k

def regula_falsi(a, b, n):
    tol, k, old = 0.5 * 10**-n, 0, None
    while True:
        fa, fb = f(a), f(b)
        c = (a*fb - b*fa) / (fb - fa); k += 1
        if fa * f(c) < 0: b = c
        else: a = c
        if old is not None and abs(c - old) < tol: return c, k
        old = c

def newton(x0, n, h=1e-6):
    tol, k = 0.5 * 10**-n, 0
    while True:
        d = (f(x0 + h) - f(x0 - h)) / (2*h); k += 1   # central difference
        x1 = x0 - f(x0) / d
        if abs(x1 - x0) < tol: return x1, k
        x0 = x1

for n in (3, 6, 10):
    print(f"{n:>2} dp | Bisection {bisection(2,4,n)[1]:>2} | "
          f"Regula Falsi {regula_falsi(2,4,n)[1]:>2} | Newton(x0=2) {newton(2,n)[1]}")
print("Newton x0=3  :", newton(3, 3))
print("Newton x0=-25:", newton(-25, 3))   # lands on the LEFT crossing
```

Expected output

```
3 dp | Bisection 12 | Regula Falsi  8 | Newton(x0=2) 5
6 dp | Bisection 22 | Regula Falsi 15 | Newton(x0=2) 5
10 dp | Bisection 36 | Regula Falsi 24 | Newton(x0=2) 6
Newton x0=3  : (2.649219887767, 4)
Newton x0=-25: (-2.893116431013, 27)
```

NOTE

The figures in this report (including the complex-plane basin map and the six-method comparison) were generated by extending this same script, using the same function, tolerances and iteration rules.

13 Conclusion & formula sheet

What to remember

For the drone-collision problem f(x) = x² − 3x + 1 − e⁻ˣ = 0, the three methods all find the right-hand crossing at x = 2.6492, y = 0.0707, but they differ sharply in how they get there:

BISECTION

The dependable baseline: 12 iterations, a known bound, zero risk inside a valid bracket.

REGULA FALSI

Improves on it (8 iterations) at no extra risk, but one endpoint gets stuck on a convex function, limiting it to linear convergence. Illinois repairs this.

NEWTON–RAPHSON

The fastest by far (5 iterations; 6 for 10 decimals), with quadratic convergence confirmed by measured error constants. Also the only one that can land on the wrong crossing.

BEST ENGINEERING ANSWER

Scan for sign changes, run Newton inside each bracket, and let Bisection act as the safety net.

Formula sheet

Method	Update formula	Error behaviour
Bisection	$c_n = (a_n + b_n)/2$	$\lvert c_n - r\rvert \le \frac{b-a}{2^n};\ N \ge \log_2 \frac{b-a}{\varepsilon}$
Regula Falsi	$c = \frac{af(b) - bf(a)}{f(b) - f(a)}$	linear, ratio $1 - \frac{f'(r)(b-r)}{f(b)} \approx 0.358$
Newton–Raphson	$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$	quadratic, $e_{n+1} \approx \frac{f''(r)}{2f'(r)} e_n^2 \approx 0.407\, e_n^2$
Central difference	$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$	error $O(h^2)$; $h = 10^{-6}$ gives $\approx 10^{-12}$
Tolerance	$\varepsilon = 0.5 \times 10^{-n}$	n correct decimal places

References

Burden, R. L. & Faires, J. D., *Numerical Analysis* (Cengage). • Chapra, S. C. & Canale, R. P., *Numerical Methods for Engineers* (McGraw-Hill). • Press, W. H. et al., *Numerical Recipes*, section on root finding with derivatives (rtsafe). • Dowell, M. & Jarratt, P., "A modified regula falsi method for computing the root of an equation", *BIT* 11 (1971) 168–174. • Brent, R. P., *Algorithms for Minimization without Derivatives* (Prentice-Hall, 1973).

ABOUT THIS REPORT

Prepared for Maher Bhatt (25C21007), S2320C1, from the Activity Based Learning problem statement. All figures, tables and iteration counts were generated by running the algorithms in this report; timings are machine-dependent and shown only for ordering.