

FUNDAMENTALS OF AI-AUGMENTED ENGINEERING: A CONVERSATIONAL APPROACH

Taner Onsay, Ph.D. & Sebnem Onsay
Authors
August 2026 · Revision 1.0

© 2026 AI First Engineering

Problem

›

Model

›

Compute

›

Analyze

›

Validate

›

Improve

The engineering workflow you will use throughout this book

by Taner & Sebnem Onsay,

CHAPTER - 4

PART 3 · ≈ 50 MIN

Hunt It — Bisection

Every evaluation cuts the uncertainty in half. For the first time in this book, accuracy runs on a schedule you can write down before computing.

3

Problem



Model



Compute



Analyze



Validate

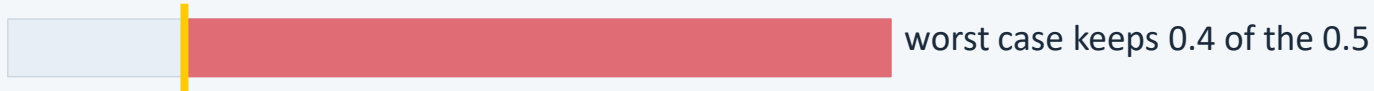


Improve

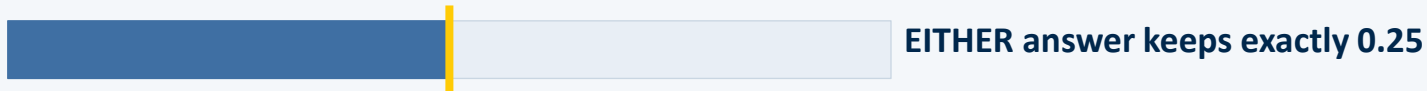


Ask the question that helps either way: the middle

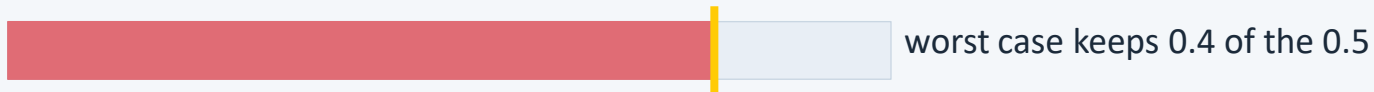
PROBE NEAR 20.0



PROBE THE MIDDLE



PROBE NEAR 20.5



WHY THE MIDDLE WINS

Wherever you probe, the sign of f keeps one piece and discards the other. **Probe off-center and the good outcome depends on luck. Probe the middle and half is gone — guaranteed, whichever way the sign falls.** Optimize the worst case, not the best hope — an engineering habit much older than computing.

colored bar = what the worst-case answer leaves you holding · gold line = where you spend the evaluation

An evaluation is a question you pay for — ask the one that helps either way. Halve the bracket. Then look at what you are holding: a new, certified bracket — half the size.

Problem



Model



Compute



Analyze



Validate



Improve



Bisection: halve, keep the sign change, repeat

```
● ● ● bisection — PSEUDOCODE

GET a, b with opposite signs of f
    # certified bracket (Part 2)
GET tol
    # from the requirement (Ch. 3)

WHILE (b - a) / 2 > tol
    SET m = (a + b) / 2
    IF f(m) has the same sign as f(a)
        SET a = m
    # right half keeps the crossing
    ELSE
        SET b = m
    # left half keeps the crossing

REPORT (a + b) / 2 ± (b - a) / 2
```

CHAPTER 2, REASSEMBLED

WHILE — a condition decides (your heater loop). IF / ELSE — the sign decision. $f(m)$ — your stopping-distance function, unchanged. **And the stop is Chapter 3's rule reborn: the KNOWN BOUND (half-width) meets the required tolerance.**

WHAT IS GENUINELY NEW

The algorithm's state is a **PAIR** $[a, b]$, not a single walker v . The search doesn't move forward; it closes in — and it never takes a step it later regrets.

Not one new statement — only a new strategy. You could have typed this in Chapter 2. Now you know why you would.

Problem



Model



Compute



Analyze



Validate



Improve



Three halvings, by hand

eval	midpoint m	d_stop(m) (m)	sign of f(m)	bracket kept	half- width
1	20.25	49.54	– (stops short)	[20.25, 20.5]	0.125
2	you	...	...	...	...
3	you	...	...	...	...

start: [20.0, 20.5] · d_stop(m) = $m + m^2/14$ · compare with 50 m — everything here is Chapter 2 arithmetic

HOW TO RUN A ROW

- 1 · $m = (a + b) / 2$
- 2 · one model run: d_stop(m)
- 3 · compare with 50 m → sign
- 4 · keep the sign-change half

The arithmetic is Chapter 2. The judgment — which half survives — is Chapter 4. That judgment is the exam question.

PREDICT first: after your two rows, how wide is the bracket — and what $\pm$ can you already guarantee?

Deliberately AI-free. Three rows on paper now buy you the whole method — and the exam.

Problem



Model



Compute



Analyze



Validate



Improve



The bracket after three halvings: [20.3125, 20.375]

eval	midpoint m	d_stop(m) (m)	sign of f(m)	bracket kept	half- width
1	20.25	49.54	– (stops short)	[20.25, 20.5]	0.125
2	20.375	50.028	+ (overruns)	[20.25, 20.375]	0.0625
3	20.3125	49.784	– (stops short)	[20.3125, 20.375]	0.03125

report: midpoint 20.344 ± 0.031 m/s — guaranteed · (the reference 20.368 sits inside, 0.024 away)

WHAT THREE EVALUATIONS BOUGHT

20.344 ± 0.031 — guaranteed. The step-0.1 march spent 204 evaluations, and its report actually landed 0.068 away.

TWO KINDS OF BOUND

The march's bound came from its step. This one comes from the bracket — you can point at both walls. **A promise, not an observation.**

Three model runs. A guaranteed ± 0.031 . The hunt is not faster walking — it is a different relationship with the evidence.

Problem



Model



Compute



Analyze



Validate



Improve



Can Excel express the SAME bisection algorithm?

You just ran three halvings by hand. Now make the algorithm visible before we automate it in Python.

iter	a	b	$m=(a+b)/2$	f(a)	f(m)	keep	new a	new b	$\pm$
1	20.0	20.5	20.25	–	–	RIGHT	20.25	20.5	0.125
2	?	?	?	?	?	?	?	?	?
3	?	?	?	?	?	?	?	?	?

DESIGN THE LOGIC

Which cells are calculations?
Which cell contains the decision?
Which values must be carried to the next row?

THE KEY QUESTION

Can the spreadsheet decide which half survives — without you manually editing a or b?

ALGORITHM FIRST.

Excel is not the method. Python is not the method. The bisection logic is the method.

Problem



Model



Compute



Analyze



Validate



Improve



The spreadsheet version of IF / ELSE

VISIBLE COMPUTATION

MIDPOINT	$m = (a + b) / 2$
SIGN TEST	$f(a) \times f(m) \leq 0 ?$
IF TRUE	new a = a · new b = m
IF FALSE	new a = m · new b = b
BOUND	$(\text{new } b - \text{new } a) / 2$

WHAT EXCEL ADDS

- Every row preserves the evidence.
- The bracket becomes visible data.
- The IF decision is exposed — not hidden in code.
- Copying the row creates repetition.
- The half-width shows convergence as it happens.

HAND → EXCEL → PYTHON

SAME ALGORITHM, NEW PLATFORM.

The computation changes form; the engineering logic does not.

Problem



Model



Compute



Analyze



Validate



Improve



What evidence does each row give you?

n	a	b	m	sign f(m)	bracket kept	$\pm$ bound
1	20.0000	20.5000	20.2500	–	[20.25, 20.50]	0.1250
2	20.2500	20.5000	20.3750	+	[20.25, 20.375]	0.0625
3	20.2500	20.3750	20.3125	–	[20.3125, 20.375]	0.03125

PREDICT

If we copy the logic two more rows, what will happen to the $\pm$ bound?

ANALYZE

Does the root estimate have to move monotonically toward the answer?

CONNECT

What will a WHILE loop automate that this table is doing row by row?

NOW AUTOMATE IT.

Once you understand the table, the Python loop is only a compact way to repeat it.

Problem



Model



Compute



Analyze



Validate



Improve



The bound is knowable before you run

THE GUARANTEE, WRITTEN DOWN

after n halvings: $\text{width} = W_0 / 2^n \rightarrow E_{\text{abs}} \leq W_0 / 2^{n+1}$

halvings n	0	1	2	3	4	5
guaranteed $\pm$ (from [20.0, 20.5])	0.25	0.125	0.0625	0.031	0.016	0.008

five halvings guarantee the accuracy the 2,037-evaluation march delivered

CHAPTER 3- STOPPING RULE, PERFECTED

Stop when the method's known bound meets the required tolerance — Chapter 3's honest rule. Bisection's bound halves on a fixed schedule, **so you can count the iterations before running.**

THE REQUIREMENT CHOOSES n

Need ± 0.008 ? Read it off the schedule: five halvings. Not hope, not habit — the tolerance picks the workload.

For the first time, you can promise accuracy in advance — and price it in evaluations. Accuracy becomes a schedule, not an experiment.

Chapter 3 asked: “is the next decimal worth 10× the computation?” — Chapter 4's price list reads differently. The next slides show how differently.





PREDICT

No head start this time: bracket $[0, 40 \text{ m/s}]$. Pure physics, zero knowledge.

The Chapter 3 march paid 2,037 evaluations for $E \approx 0.008$. To GUARANTEE that accuracy starting from a bracket eighty times wider than $[20.0, 20.5]$, how many model evaluations does bisection need in total? Commit a number: 50? 500? 2,000?

Certify the bracket first, with physics alone: $f(0) < 0$ — a parked car stops in time. $f(40) > 0$ — 144 km/h cannot stop in 50 m. Then we run it.

Problem

›

Model

›

Compute

›

Analyze

›

Validate

›

Improve



bisection.py — the hunt, automated

```

● ● ● bisection.py — Chapter 2's stopping_distance(), unchanged

def f(v):
    return stopping_distance(v, 1.0, 7.0) - 50.0
a, b = 0.0, 40.0          # physical bracket
fa, fb = f(a), f(b)       # 2 endpoint evaluations
assert fa * fb < 0
    # certify the sign change — Principle No. 2
tol = 0.01                # required half-width (m/s)

n = 0
while (b - a) / 2 > tol:
    m = (a + b) / 2
    fm = f(m)              # ONE new evaluation per halving
    n = n + 1
    if fa * fm <= 0:        # sign change on the left?
        b, fb = m, fm     # keep the left half — and its f
    else:
        a, fa = m, fm     # keep the right half — and its f
print(n, "halvings:", (a+b)/2, "+/-", (b-a)/2)

11 halvings: 20.361328125 +/- 0.009765625

```

READ IT LIKE AN ENGINEER

The **assert** is Principle No. 2 in code — **one line certifies the trap before the hunt**. `tol` comes from the requirement, never from habit (Chapter 3). And `fa`, `fm` are stored and reused — the algorithm never recomputes what it already knows: one new model run per halving.

INSPECT THE OUTPUT

The first midpoint of $[0, 40]$ is 20.0 — Chapter 2's old answer, reused as evidence. **And 17 printed digits against a ± 0.0098 bound? You know better: report 20.36 ± 0.01 .**

PREDICT → RUN → INSPECT — as always. The output shown is the real output of this exact program.

Problem



Model

**Compute**

Analyze



Validate



Improve



Two thousand evaluations — or about a dozen

THE MARCH · CHAPTER
3

2,037

model evaluations at step 0.01 —
actual error ≈ 0.008



THE HUNT · TODAY

11

halvings from $[0, 40]$ — guaranteed $\pm$
0.0098

THE PROMISE, KEPT

11 halvings (11 new midpoint evaluations) +
2 endpoint checks = 13 total model
evaluations. **Chapter 3 promised “a dozen
calculations instead of two thousand.”**
Count them.

guaranteed accuracy	the march (evaluations)	bisection halvings $[0, 40]$
± 0.1	204	8
± 0.01	2,037	11
± 0.001	20,368	15

READ THE SCALING — THE REAL LESSON

Each extra decimal costs the march $\times 10$. It costs bisection
+3 or 4 halvings — because every halving doubles the
accuracy. Multiplicative vs. additive cost: that asymmetry is
the entire story of numerical methods.

every bisection row also spends the same 2 endpoint checks — totals: $10 \cdot 13 \cdot 17$ model evaluations

The intelligent search buys accuracy with information, not effort. Every midpoint evaluation halves what remains. None is wasted. (These counts belong to THIS search — the asymmetry is what generalizes.)

Problem



Model



Compute



Analyze



Validate



Improve



Where does the first bracket come from?

1 · PHYSICS

Bounds you can defend without computing: 0 m/s certainly stops; 40 m/s (144 km/h) certainly cannot, on 50 m. Crude is fine — the whole 40-wide cage cost eleven halvings. Bisection forgives a generous bracket.

2 · A COARSE SWEEP

Your Chapter 2 search at a big step — five or ten evaluations — hands you a sign change to certify. The march is not retired. It is promoted to reconnaissance.

3 · A PLOT

Ask for the curve; look where it crosses. The engineer's eye finds a bracket faster than code — and sees HOW MANY crossings there are, which two endpoint signs can never tell you.

AND ALWAYS — CERTIFY BEFORE YOU TRUST

Whatever the source: evaluate f at both ends, look at the two signs, and only then let the algorithm run. One assert. Every time.

Establishing the bracket is engineering; the halving is clerical. Guess wisely, certify the signs, then let the algorithm do the boring part.

Problem



Model



Compute



Analyze



Validate



Improve



AI can now propose the method — you still sign for it



AI MAY...

- implement the bisection you specified
- run both methods and tabulate the comparison
- check your hand iterations against its own
- suggest candidate brackets — for your certification
- propose a method for the job — with reasons
- explain any method on demand — analyst and tutor at once



THE ENGINEER MUST...

- write the question as $f(x) = 0$ — and own what f means
- certify the sign change before trusting any guarantee
- set the tolerance from the engineering requirement
- confirm the conditions behind every claimed bound
- validate the root against the physics
- choose the method — and defend the choice, and the decision

A deeper analyst, the same constitution. A recommendation arrives with its contract attached — or it goes back. **AI assists. Human verifies. Engineer decides.**

Problem



Model



Compute



Analyze



Validate



Improve