

FUNDAMENTALS OF AI-AUGMENTED ENGINEERING: A CONVERSATIONAL APPROACH

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

© 2026 AI First Engineering



The engineering workflow you will use throughout this book

CHAPTER - 9

PART 2 · ≈ 30 MIN

How Does Jacobi Actually Work?

Part 1 showed why large systems call for repeated correction. Part 2 builds the simplest such method — Jacobi — by hand, in pseudocode and in MATLAB, on a circuit whose answer we already know.

2

Problem

›

Model

›

Compute

›

Analyze

›

Validate

›

Improve



CONTENTS OF CHAPTER 9:

Chapter 8: a few coupled unknowns, solved directly. **Chapter 9:** far too many — solved by repeated correction, and checked.

1

Why do we need another way to solve equations?

From two unknowns to 100,000 · mostly-empty matrices · the 80 GB surprise · direct vs. iterative



2

How does Jacobi actually work? · you are here

Repeated correction by hand · using only the previous guess · pseudocode and compact MATLAB



3

When do iterations converge — and when do they fail?

Convergence and divergence · equation order · change vs. residual vs. error · stopping safely



4

Is Jacobi the only way? What happens at engineering scale?

Gauss-Seidel · a growing sparse network · LU in concept · what industrial solvers do



5

Can we trust the solver — and the AI?

Auditing AI-written solver code · ACCEPT / MODIFY / REJECT · a verification checklist

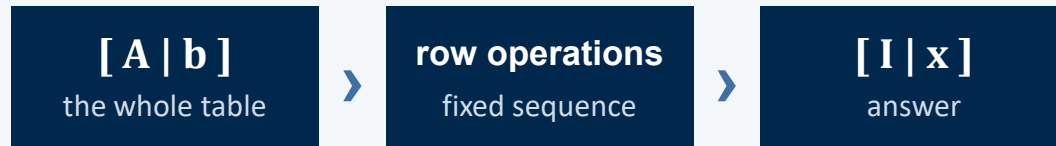


Course spiral: Ch 4–5 — one unknown, improved step by step until a stopping test is met (bisection, Newton) · Ch 8 — several coupled unknowns, solved directly · Ch 9 — the same kind of system, solved by repeated correction, and judged by evidence.



What If We Could Solve Without Elimination?

CHAPTER 8 · ELIMINATION



- Finishes after a known number of steps.
- Each step rewrites the table — and can fill in its zeros.

TODAY · REPEATED CORRECTION



- The equations are never rewritten — only the guess changes.
- Each pass needs only the nonzero coefficients.

PART 1 LEFT US WITH A QUESTION

Given a guess for every node voltage, how can the circuit's own equations produce a better guess — and how would we know it is better?

Repeated correction: keep the equations exactly as they are — and improve the guess instead.





Start With a Circuit We Already Trust

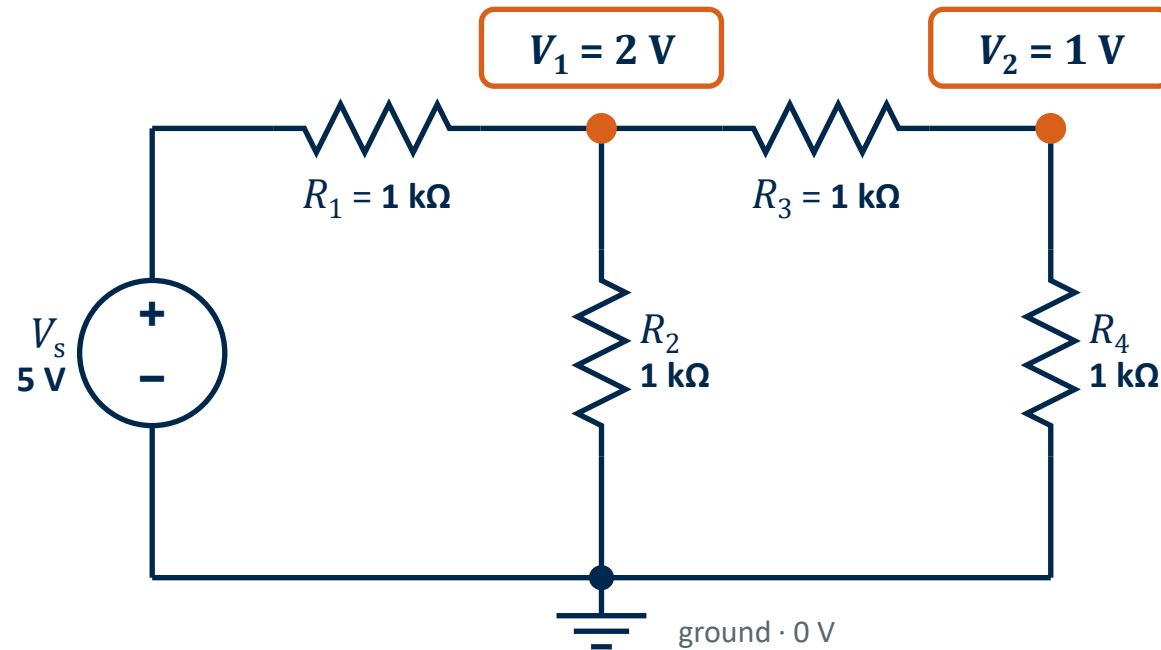


Figure 1. The Chapter 8 circuit: every resistor 1 kΩ (1 mS); the 5 V source behind R1 contributes 5 V ÷ 1 kΩ = 5 mA at node V1.

THE MODEL · KCL AT EACH NODE

$$3V_1 - V_2 = 5$$

$$-V_1 + 2V_2 = 0$$

A in mS

x in V

b in mA

$$\begin{bmatrix} 3 & -1 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$$

WHY THIS CIRCUIT?

- **Known answer: (2, 1) V** — Gauss-Jordan, $A \backslash b$ and substitution agree.
- Every iterate we compute today can be checked against the truth.

Test a new method on a problem you can already solve. Then you can tell whether it works — not just whether it runs.

Problem



Model



Compute



Analyze



Validate



Improve



Turn Equations Into Update Rules

1 · KCL EQUATION

$$3V_1 - V_2 = 5$$

$$-V_1 + 2V_2 = 0$$

2 · ISOLATE ITS UNKNOWN

$$\triangleright V_1 = (5 + V_2) / 3$$

$$\triangleright V_2 = V_1 / 2$$

3 · LABEL OLD AND NEW

$$\triangleright V_1^{(k+1)} = (5 + V_2^{(k)}) / 3$$

$$\triangleright V_2^{(k+1)} = V_1^{(k)} / 2$$

NEW · (k+1)

OLD · (k)

NOTATION

The superscript is an **iteration number**, not a power: $V_1^{(2)}$ means "V₁ after 2 corrections", not V₁ squared.

DIVIDE BY THE DIAGONAL

Each equation is solved for the unknown on its diagonal (3 mS, 2 mS). These formulas need every diagonal coefficient to be **nonzero**.

Jacobi update: each new value comes from its own equation and the previous iteration's values.

Problem



Model



Compute



Analyze



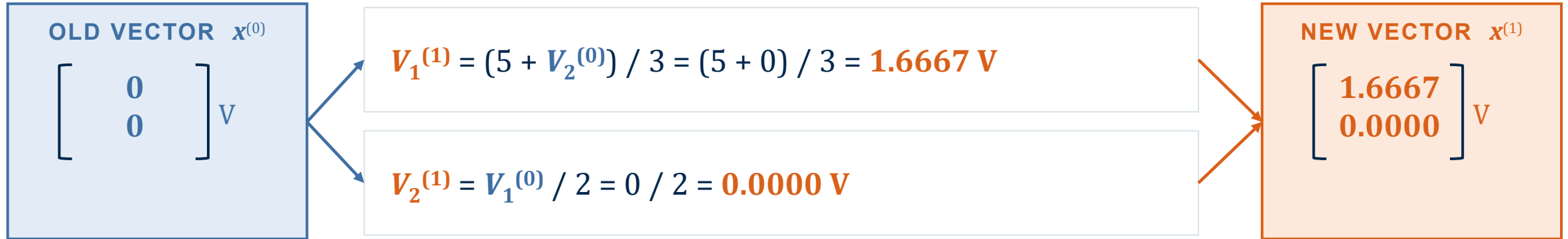
Validate



Improve



Start With a Guess — Even Zero



ASK THE ROOM

$V_2^{(1)}$ is still 0 V — even though $V_1^{(1)}$ is already 1.67 V.

Mistake — or the rule?

CHECK THE UNITS

$$V_1^{(1)} = (5 \text{ mA} + 1 \text{ mS} \times 0 \text{ V}) / 3 \text{ mS} \\ = \mathbf{1.6667 \text{ V}}$$

mA ÷ mS = V: every iterate is a node voltage.

One iteration = one full sweep: every unknown is recomputed once, from the previous vector.

Problem ›

Model ›

Compute ›

Analyze ›

Validate ›

Improve



One More Iteration: What Changes?

$k = 1 \rightarrow 2$ AND $k = 2 \rightarrow 3$

$$V_1^{(2)} = (5 + 0) / 3 = \mathbf{1.6667}$$

$$V_2^{(2)} = \mathbf{1.6667} / 2 = \mathbf{0.8333}$$

$$V_1^{(3)} = (5 + \mathbf{0.8333}) / 3 = \mathbf{1.9444}$$

$$V_2^{(3)} = \mathbf{1.6667} / 2 = \mathbf{0.8333}$$

WHAT CHANGES?

- V_1 and V_2 take turns moving.
- Each vector is closer to (2, 1) V — but never reaches it.

k	V_1 (V)	V_2 (V)
0	0.0000	0.0000
1	1.6667	0.0000
2	1.6667	0.8333
3	1.9444	0.8333
4	1.9444	0.9722
5	1.9907	0.9722
Ref.	2.0000	1.0000

Table 1. Jacobi iterates from (0, 0) V. Orange: value changed this iteration.

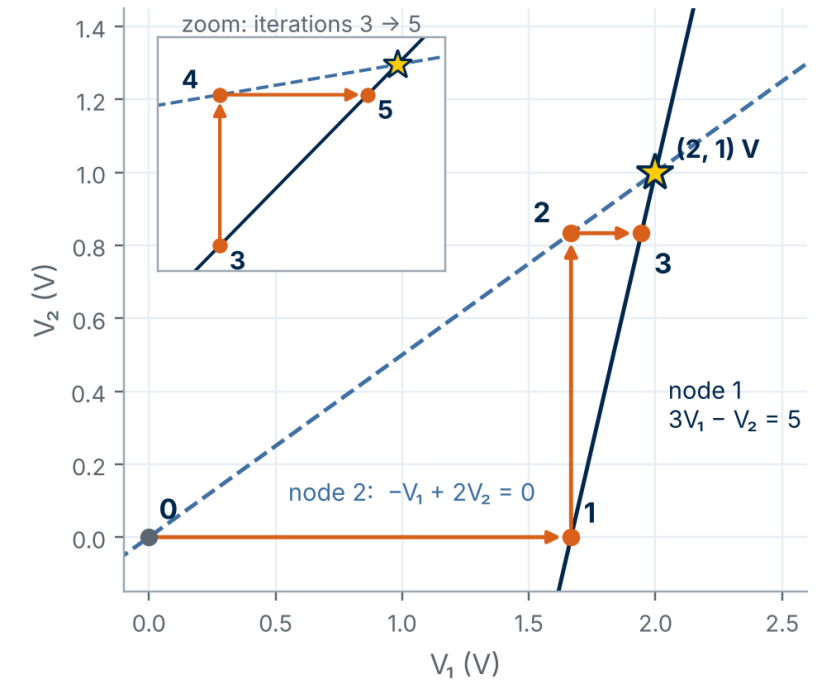


Figure 2. Iterates 0–5 in the (V_1 , V_2) plane; each lands on one KCL line.

Every new vector is built from the previous one — and each one is closer to the reference (2, 1) V.

Problem



Model

**Compute**

Analyze



Validate

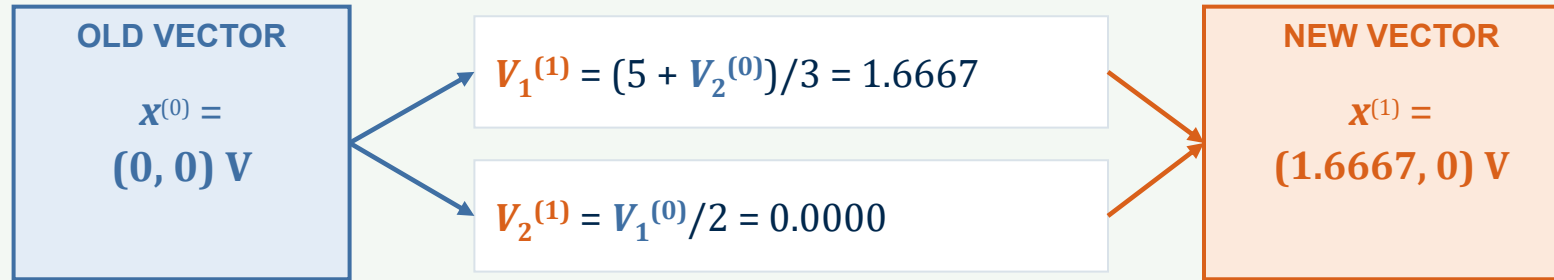


Improve

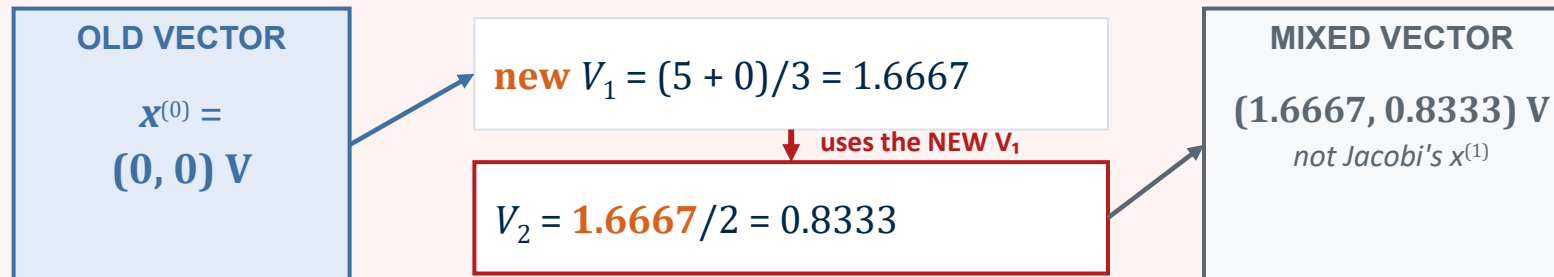


The Rule That Makes It Jacobi

JACOBI ✓ OLD VECTOR → COMPUTE ALL NEW VALUES → NEW VECTOR



NOT JACOBI ✗ UPDATE V_1 , THEN USE THE NEW V_1 FOR V_2



WHAT IS WRONG?

- The second sequence is **not Jacobi**. It resembles **Gauss-Seidel** (Part 4).
- It may still converge to $(2, 1) V$.
- So it is an **algorithm-identity error** — not necessarily a wrong-answer error.
- If you claim Jacobi, every number you report must come from Jacobi.

Jacobi: compute ALL new values from the OLD vector — only then replace it.

Problem



Model



Compute



Analyze



Validate



Improve



From Mathematics to Pseudocode

JACOBI · PSEUDOCODE

```

INPUT A, b, starting guess x, tol, max_iter
FOR k = 1, 2, ..., max_iter
    x_old ← x           save the OLD vector
    FOR each row i
        x(i) ← ( b(i) - Σ A(i,j) · x_old(j) ) / A(i,i)
                    sum over j ≠ i
    change ← max | x - x_old |
    IF any x(i) is not finite: STOP, error
    IF change < tol: leave the loop
END
OUTPUT x, k
  
```

FOR OUR 2 × 2 CIRCUIT

```

i = 1
x(1) ← (b(1) - A(1,2) · x_old(2)) / A(1,1)
        = (5 - (-1) · V2(k)) / 3
i = 2
x(2) ← (b(2) - A(2,1) · x_old(1)) / A(2,2)
        = (0 - (-1) · V1(k)) / 2
  
```

THREE PROMISES

- Every update reads only **x_old**.
- The loop always ends (max_iter).
- Inf or NaN stops it with an error.

Pseudocode first: decide WHAT the computer must do before deciding HOW to write it in MATLAB.

Problem



Model



Compute



Analyze



Validate



Improve



Now Make MATLAB Do the Repetition

jacobi_demo.m · MATLAB

```
A = [3 -1; -1 2];
b = [5; 0];
x = [0; 0];
tol = 1e-3;
max_iter = 100;
for k = 1:max_iter
    x_old = x;
    x(1) = (b(1) - A(1,2)*x_old(2))/A(1,1);
    x(2) = (b(2) - A(2,1)*x_old(1))/A(2,2);
    change = max(abs(x - x_old));
    if ~all(isfinite(x))
        error('Nonfinite iterate encountered.');
```

```
    end
    if change < tol
        break;
    end
end
disp(x)
disp(k)
```

READ THE CODE

- **x_old = x**: save the OLD vector
- Both updates read only **x_old**
- **change**: largest component change, in V
- **isfinite**: stop on Inf or NaN
- **break** once the change is below 1 mV

OUTPUT · VERIFIED RUN

```
1.9997
0.9999
10
```

x ≈ (1.9997, 0.9999) V
after **k = 10** iterations
A\b gives (2, 1) V

Same algorithm, three forms: hand calculation, pseudocode and MATLAB — line for line.

Problem



Model



Compute



Analyze



Validate



Improve



When Should We Stop?

1 · TOLERANCE

`change < tol`

Largest change between consecutive iterations below **1e-3 V = 1 mV**.

2 · ITERATION CAP

`k = 1:max_iter`

Guarantees the loop ends. If it ends at `max_iter`, the tolerance was **not** met — check **change**. The proper test is the actual final `change < tol`, ideally accompanied by an explicit convergence flag.

3 · FINITE VALUES

`isfinite(x)`

Stops on Inf or NaN (e.g. a zero diagonal). A slowly growing iterate can stay finite — the cap is the backstop.

CHANGE IS NOT ERROR

The run stopped at **k = 10**: the largest change was 0.643 mV, below 1 mV.

The true error was 0.257 mV — we know only because we have the reference (2, 1) V.

A 1 mV tolerance limits the change between iterations. It does not guarantee an error below 1 mV.

k	change (mV)	error vs (2, 1) (mV)
8	3.858	1.543
9	1.286	0.772
10	0.643	0.257

Table 2. Last three iterations of the verified run (largest component).

A stopping rule decides when to stop — not how close we are. Part 3 examines the gap.

Problem



Model



Compute



Analyze



Validate



Improve



Did We Get the Right Answer?

COMPUTED vs REFERENCE

Jacobi (k = 10): (1.99974, 0.99987) V

Reference (Ch 8): (2, 1) V

$$\mathbf{r} = \mathbf{A} \cdot \mathbf{x} - \mathbf{b} \quad \% \text{ mA}$$

$$\mathbf{err} = \mathbf{x} - [2; 1] \quad \% \text{ V}$$

THE RESIDUAL

How well does $\mathbf{x}$ satisfy the original equations? $\mathbf{A} \text{ (mS)} \times \mathbf{x} \text{ (V)} - \mathbf{b} \text{ (mA)} \rightarrow \mathbf{mA}$.

Measure	Compares	Units	This run
Iteration change	$\mathbf{x}^{(10)} \text{ vs } \mathbf{x}^{(9)}$	V	6.43×10^{-4}
Residual $\mathbf{r} = \mathbf{Ax} - \mathbf{b}$	$\mathbf{Ax} \text{ vs } \mathbf{b}$	mA	$(-6.43 \times 10^{-4}, 0)$
Reference error	$\mathbf{x} \text{ vs } (2, 1)$	V	$(-2.57, -1.29) \times 10^{-4}$

Table 3. Three measures from the verified run — different questions, different units.

THREE QUESTIONS — DON'T MIX THEM UP

- **Change:** has it stopped moving?
- **Residual:** does it satisfy the equations?
- **Error:** is it the true answer? (needs a reference)

RUNS ≠ CORRECT. Check the equations (residual) and, when you can, a trusted reference (error).

Problem



Model



Compute



Analyze



Validate



Improve



YOUR TURN

Spot the Hidden Mistake

A STUDENT'S "JACOBI" CODE

```
A = [3 -1; -1 2]; b = [5; 0];
x = [0; 0]; tol = 1e-3; max_iter = 100;
for k = 1:max_iter
    x_old = x;
    x(1) = (b(1) - A(1,2)*x_old(2))/A(1,1);
    x(2) = (b(2) - A(2,1)*x(1))/A(2,2);
    change = max(abs(x - x_old));
    if ~all(isfinite(x))
        error('Nonfinite iterate encountered.');
```

end

if change < tol

break;

end

end

IT RUNS — RESULT

x = (2.0000, 1.0000) k = 6

Close to the reference (2, 1) V.

Does this code implement Jacobi correctly? Why or why not?

≈ 2 MIN

- Point to the exact line.
- Is the answer wrong — or is something else wrong?
- What one change would make it Jacobi?

RUNS ≠ CORRECT. Correct means the code does what it claims — here, Jacobi.

Problem



Model



Compute



Analyze



Validate



Improve



What We Know — and What We Don't

EQUATIONS



UPDATE RULES



OLD VECTOR ONLY



STOP, THEN CHECK

WE KNOW

How to turn each equation into an update rule, and why every update must use the previous iteration's values.

WE CAN DO

Iterate by hand, write the pseudocode, run it in MATLAB, and check the result with a residual and a reference.

WE DON'T KNOW YET

Will Jacobi always approach the answer? How fast? And does a tiny change mean a tiny error?

NEXT — PART 3 · WHEN DO ITERATIONS CONVERGE — AND WHEN DO THEY FAIL?

Same circuit, same two equations — written in the other order. Will Jacobi still find (2, 1) V? Make a prediction.

A method that runs is not yet a method we can trust. Part 3 asks when Jacobi works — and how we would know.

Problem



Model



Compute



Analyze



Validate



Improve