
CS/MATH 3414 — Numerical Methods

Lecture 4 Linear Systems of Equations, Part 1

Virginia Tech Department of Computer Science, Fall 2026

Instructor: Sumeet Khatri

Date: September 15, 2026

1	Introduction	2
2	Gaussian elimination	4
2.1	Algorithm	8
2.2	What changes for singular or rectangular systems?	10
3	Gaussian elimination with scaled partial pivoting	16
3.1	Why Gaussian elimination can fail	16
3.2	Scaled partial pivoting	17
4	How much computation is required?	23
4.1	The arithmetic model and asymptotic notation	23
4.2	Vector operations and matrix multiplication	24
4.3	Deriving the cost of Gaussian elimination	28
A	Linear algebra background	31
A.1	Vectors, matrices, products, and transposes	31
A.2	Inner products, lengths, and matrix norms	33
A.3	Span, independence, bases, and dimension	35
A.4	Elementary, permutation, and triangular matrices	35
A.5	Column space, nullspace, and rank	36
A.6	Consistency and all exact solutions	37
A.7	Dense matrices, sparse matrices, and storage	39
A.8	A reference table of common operations and their costs	40

Learning Objectives

- Understand and explain the concept of a linear system of equation and its importance in scientific and engineering applications.
- Explain the steps of Gaussian elimination in general terms as an algorithm.
- Understand the arithmetic and storage costs of matrix operations and Gaussian elimination, and reuse factors for repeated right-hand sides.

Primary References

- [1] T. A. Driscoll and R. J. Braun, *Fundamentals of Numerical Computation*, Includes Python, Julia, and MATLAB examples (Society for Industrial and Applied Mathematics, Philadelphia, PA, 2022), <https://fncbook.com/>.
- [2] T. Sauer, *Numerical Analysis*, Third (Pearson, 2018), <https://www.pearson.com/en-ca/subject-catalog/p/numerical-analysis/P200000006340/9780134696454>.

- [3] R. L. Burden, J. D. Faires, and A. M. Burden, *Numerical Analysis*, 10th ed. (Cengage Learning, 2016), <https://www.cengage.com/c/numerical-analysis-10e-burden/9781305253667/?filterBy=Student>.
- [4] E. W. Cheney and D. R. Kincaid, *Numerical Mathematics and Computing*, 7th ed. (Brooks/Cole, Cengage Learning, 2013), <https://web.ma.utexas.edu/CNA/NMC7/>.
- [5] L. N. Trefethen and D. Bau III, *Numerical Linear Algebra* (Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997), <https://epubs.siam.org/doi/book/10.1137/1.9780898719574>.
- [6] G. Strang, *Introduction to Linear Algebra*, 5th ed. (Wellesley–Cambridge Press, Wellesley, MA, 2016), <https://math.mit.edu/~gs/linearalgebra/ila5/indexila5.html>.
- [7] S. Axler, *Linear Algebra Done Right*, 4th ed. (Springer, 2026), <https://linear.axler.net/LADR4e.pdf>.
- [8] V. Strassen, “Gaussian elimination is not optimal”, *Numerische Mathematik* **13**, 354 (1969).
- [9] A. Fawzi, M. Balog, A. Huang, et al., “Discovering faster matrix multiplication algorithms with reinforcement learning”, *Nature* **610**, 47 (2022).
- [10] A. Novikov, N. Vũ, M. Eisenberger, et al., “AlphaEvolve: A coding agent for scientific and algorithmic discovery”, *arXiv:2506.13131* (2025).
- [11] E. Dupont, M. Eisenberger, B. Kozlovskii, et al., “Improving the matrix multiplication exponent with modern optimization and AlphaEvolve”, *arXiv:2608.16884* (2026).
- [12] J. Huang, T. M. Smith, G. M. Henry, and R. A. van de Geijn, “Strassen’s Algorithm Reloaded”, in *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis* (2016), pp. 690–701, *arXiv:1605.01078*.
- [13] J.-G. Dumas, C. Pernet, and A. Sedoglavic, “A non-commutative algorithm for multiplying 4x4 matrices using 48 non-complex multiplications”, *arXiv:2506.13242* (2025).
- [14] J.-G. Dumas, C. Pernet, A. Sedoglavic, and P. Tichavský, “Fast matrix multiplication via recursive $\langle 4 \times 4 \times 4 : 48 \rangle$ algorithms into practice”, *arXiv:2609.12027* (2026).
- [15] I. E. Kaporin, “Finding Complex-Valued Solutions of Brent Equations Using Nonlinear Least Squares”, *Computational Mathematics and Mathematical Physics* **64**, 1881 (2024).

Note I: Much of the text here comes directly from, or is a summarized form of, the text in the above two references.

Note II: See Appendix A for a review of linear algebra.

1 Introduction

One of the fundamental problems in many scientific and engineering applications is to solve an algebraic linear system. A linear system asks for numbers that satisfy several linear equations at once. In a network model these numbers might be unknown flows; in a discretized differential equation they might be temperatures at grid points; in a data-fitting problem they might be model coefficients. Although the applications differ, the central problem always the same form:

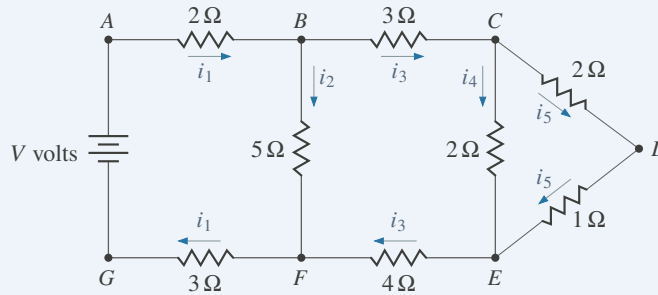
$$A\mathbf{x} = \mathbf{b}, \quad A \in \mathbb{R}^{m \times n}, \quad \mathbf{x} \in \mathbb{R}^n, \quad \mathbf{b} \in \mathbb{R}^m. \quad (1.1)$$

Here m counts equations and n counts unknowns. The entries of A and $\mathbf{b}$ are given; the entries of $\mathbf{x}$ are to be determined. Unless stated otherwise, all matrices and vectors in this note are real. The transpose is denoted by

$A^\top$. We will mostly consider the square case, $m = n$, in which the coefficient matrix is non-singular, meaning that it has an inverse. (The more general case will be dealt with in the next lecture.)

The system $A\mathbf{x} = \mathbf{b}$ may or may not have a solution, and if it has a solution, it may or may not be unique. Gaussian elimination is the standard method for solving the linear system by using a calculator or a computer. This method is almost undoubtedly familiar to you from linear algebra, since it is the simplest way of solving a system by hand. When the system has no solution, other approaches are used, such as linear least squares, which we will encounter in the future.

Example 1 (An electrical circuit). Kirchoff's laws of electrical circuits state that both the net flow of current through each junction and the net voltage drop around each closed loop of a circuit are zero. Suppose that a potential of V volts is applied between the points A and G in the circuit and that i_1, i_2, i_3, i_4 , and i_5 represent the current flow as shown in the diagram below.



Using G as a reference point, Kirchoff's laws imply that the currents satisfy the following system of linear equations:

$$5i_1 + 5i_2 + 0i_3 + 0i_4 + 0i_5 = V, \quad (1.2)$$

$$0i_1 + 0i_2 + i_3 - i_4 - i_5 = 0, \quad (1.3)$$

$$0i_1 + 0i_2 + 0i_3 + 2i_4 - 3i_5 = 0, \quad (1.4)$$

$$i_1 - i_2 - i_3 + 0i_4 + 0i_5 = 0, \quad (1.5)$$

$$0i_1 + 5i_2 - 7i_3 - 2i_4 + 0i_5 = 0 \quad (1.6)$$

Thus, this is a system of $n = m = 5$, and the given A and $\mathbf{b}$ are

$$A = \begin{pmatrix} 5 & 5 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & -1 \\ 0 & 0 & 0 & 2 & -3 \\ 1 & -1 & -1 & 0 & 0 \\ 0 & 5 & -7 & -2 & 0 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} V \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \quad (1.7)$$

In applied mathematics and in many applications, it can be a daunting task for even the largest and fastest computers to solve accurately extremely large systems involving thousands or millions of unknowns. Some important questions are the following:

- How do we store such large systems in the computer?
- How do we know that the computed answers are correct?

- What is the precision of the computed results?
- Can the algorithm fail?
- How long will it take to compute the answers?
- What is the asymptotic operation count of the algorithm?
- Will the algorithm be unstable for certain systems?
- Can instability be controlled by pivoting? (Permuting the order of rows of the matrix when doing Gaussian elimination is called **pivoting**.) Which strategy for pivoting should be used?
- How do we know whether the matrix is ill-conditioned and whether the answers are accurate?

2 Gaussian elimination

Gaussian elimination transforms a linear system into an upper triangular form that is easier to solve. We will mostly consider the square case, $m = n$, in which the coefficient matrix is non-singular, meaning that it has an inverse. (The more general case will be dealt with in the next lecture.) Rather than trying to isolate every variable simultaneously, we transform the equations until the last equation contains one unknown, the preceding equation contains two, and so on.

The augmented matrix $(A \mid \mathbf{b})$ records the coefficients and right-hand side together. Three elementary operations on its rows preserve the solution set (let R_i denote the rows of the augmented matrix):

1. Exchange two equations: $R_i \leftrightarrow R_j$.
2. Multiply an equation by a nonzero scalar: $R_i \leftarrow cR_i$, with $c \neq 0$.
3. Add a multiple of one equation to another: $R_i \leftarrow R_i + cR_j$, with $i \neq j$.

Each operation is reversible. Applying its inverse recovers the original system, so a vector solves the transformed equations if and only if it solves the original equations. Notice that a row operation must also be applied to the corresponding entry of $\mathbf{b}$. Modifying only the coefficient matrix generally changes the problem.

Proposition 1 (Equivalence under elementary row operations). Let $A \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$. Suppose $(\tilde{A} \mid \tilde{\mathbf{b}})$ is obtained from $(A \mid \mathbf{b})$ by a finite sequence of the three elementary row operations, with every row scaling nonzero. Then

$$\{\mathbf{x} : A\mathbf{x} = \mathbf{b}\} = \{\mathbf{x} : \tilde{A}\mathbf{x} = \tilde{\mathbf{b}}\}. \quad (2.1)$$

This conclusion does not require a square matrix or a consistent system.

Proof. Exchanging rows only changes the order in which the equations are listed. An equation $a_{i1}x_1 + \cdots + a_{in}x_n = b_i$ is equivalent to the equation obtained by multiplying both sides by $c \neq 0$, since division by c recovers it. For a row addition, every solution of the old equations satisfies the sum of equation i and c times equation j . Conversely, equation j is unchanged, so subtracting its c multiple from the new equation i recovers the old equation. Thus each individual operation preserves the solution set in both directions. Repeating this argument through the finite sequence proves the statement.

Equivalently, each row operation is left multiplication by an invertible elementary matrix. If their product is E , the transformed system is $E A \mathbf{x} = E \mathbf{b}$. Multiplication by E^{-1} gives the original system. The same E must multiply both A and $\mathbf{b}$; otherwise this equivalence does not hold. Multiplying a row by zero would make E noninvertible and can erase a constraint. ■

Key Idea

Gaussian elimination removes variables from successive equations without changing the solution set. It stops at a triangular system; back substitution then recovers the unknowns.

Example 2 (A warm-up 2×2 example). Consider the system

$$\begin{aligned}2x + y &= 5, \\4x + 3y &= 11.\end{aligned}\tag{2.2}$$

Intuitively, the way we would solve this is simply by expressing one variable in terms of the other and then substituting. Solving the second equation for x gives

$$x = \frac{11 - 3y}{4}.\tag{2.3}$$

Substituting this expression into the first equation eliminates x , leaving an equation involving only y :

$$2\left(\frac{11 - 3y}{4}\right) + y = 5.\tag{2.4}$$

Multiplying both sides by 2 and simplifying, we obtain

$$\begin{aligned}11 - 3y + 2y &= 10, \\-y &= -1, \\y &= 1.\end{aligned}\tag{2.5}$$

We now substitute $y = 1$ into our expression for x :

$$x = \frac{11 - 3(1)}{4} = \frac{8}{4} = 2.\tag{2.6}$$

Thus, the solution is $x = 2$ and $y = 1$. We can check these values in both original equations:

$$\begin{aligned}2(2) + 1 &= 5, \\4(2) + 3(1) &= 11.\end{aligned}\tag{2.7}$$

Gaussian elimination formalizes this concept of eliminating one of the variables and then solving for the rest to larger systems.

Example 3 (A warm-up 3×3 example). Consider now the following 3×3 example:

$$\begin{aligned}2x_1 + x_2 - x_3 &= 8, \\-3x_1 - x_2 + 2x_3 &= -11, \\-2x_1 + x_2 + 2x_3 &= -3.\end{aligned}\tag{2.8}$$

To start, subtract $-3/2$ times the first row from the second row and subtract -1 times the first row from the third row:

$$\left(\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ -3 & -1 & 2 & -11 \\ -2 & 1 & 2 & -3 \end{array} \right) \rightarrow \left(\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ 0 & 1/2 & 1/2 & 1 \\ 0 & 2 & 1 & 5 \end{array} \right). \quad (2.9)$$

For example, the new second right-hand side is $-11 - (-3/2)8 = 1$. Then, subtracting 4 times the second row from the third gives

$$\left(\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ 0 & 1/2 & 1/2 & 1 \\ 0 & 0 & -1 & 1 \end{array} \right). \quad (2.10)$$

The last row yields $x_3 = -1$. Substituting into the second gives $\frac{1}{2}x_2 + \frac{1}{2}(-1) = 1$, so $x_2 = 3$. The first then gives $2x_1 + 3 - (-1) = 8$, so $x_1 = 2$. Therefore

$$\mathbf{x} = \begin{pmatrix} 2 \\ 3 \\ -1 \end{pmatrix}. \quad (2.11)$$

Substitution into all three original equations verifies the answer.

Let us now go through the simplest form of Gaussian elimination, as shown in Example 3 above, a bit more formally. We will introduce several terms along the way.

Let us consider the following different example that has four equations and four unknowns:

$$\begin{array}{rrcr} 6x_1 & -2x_2 & +2x_3 & +4x_4 & = & 16 \\ 12x_1 & -8x_2 & +6x_3 & +10x_4 & = & 26 \\ 3x_1 & -13x_2 & +9x_3 & +3x_4 & = & -19 \\ -6x_1 & +4x_2 & +x_3 & -18x_4 & = & -34 \end{array} \quad (2.12)$$

First phase: forward elimination

In the first phase of the Gaussian elimination procedure, certain multiples of the first equation are subtracted from the second, third, and fourth equations so as to eliminate x_1 from these equations. Thus, we want to create 0's as coefficients for each x_1 below the first (where 12, 3, and -6 now stand). It is clear that we should subtract 2 times the first equation from the second. (This multiplier is simply the quotient $\frac{12}{6}$.) Likewise, we should subtract $\frac{1}{2}$ times the first equation from the third. (Again, this multiplier is just $\frac{3}{6}$.) Finally, we should subtract -1 times the first equation from the fourth. When all of this has been done, the result is

$$\begin{array}{rrcr} 6x_1 & -2x_2 & +2x_3 & +4x_4 & = & 16 \\ & -4x_2 & +2x_3 & +2x_4 & = & -6 \\ & -12x_2 & +8x_3 & +x_4 & = & -27 \\ & 2x_2 & +3x_3 & -14x_4 & = & -18 \end{array} \quad (2.13)$$

Note that the first equation was not altered in this process, although it was used to produce the 0 coefficients in the other equations. In this context, it is called the **pivot equation**, and the coefficient 6 and/or the variable x_1 in the pivot is sometimes referred to as the **pivot (element)**.

As stated in general earlier, notice also that systems (2.12) and (2.13) are *equivalent* in the following technical sense: any solution of (2.12) is also a solution of (2.13), and vice versa. This follows at once from the fact that if equal quantities are added to equal quantities, the resulting quantities are equal. One can get system (2.12) from system (2.13) by adding 2 times the first equation to the second, and so on.

In the second step of the process, we mentally ignore the first equation and the first column of coefficients. This leaves a system of three equations with three unknowns. The same process is now repeated using the top equation in the smaller system as the current pivot equation. Thus, we begin by subtracting 3 times the second equation from the third. (The multiplier is just the quotient $\frac{-12}{-4}$.) Then we subtract $-\frac{1}{2}$ times the second equation from the fourth. After doing the arithmetic, we arrive at

$$\begin{array}{rrrrr} 6x_1 & -2x_2 & +2x_3 & +4x_4 & = & 16 \\ & -4x_2 & +2x_3 & +2x_4 & = & -6 \\ & & 2x_3 & -5x_4 & = & -9 \\ & & 4x_3 & -13x_4 & = & -21 \end{array} \quad (2.14)$$

The final step consists of subtracting 2 times the third equation from the fourth. The result is the so-called **upper-triangular system**

$$\begin{array}{rrrrr} 6x_1 & -2x_2 & +2x_3 & +4x_4 & = & 16 \\ & -4x_2 & +2x_3 & +2x_4 & = & -6 \\ & & 2x_3 & -5x_4 & = & -9 \\ & & & -3x_4 & = & -3 \end{array} \quad (2.15)$$

This system is equivalent to system (2.12). This completes the first phase (sometimes referred to as *forward elimination*) in the Gaussian algorithm.

Second phase: back substitution

The second phase (*back substitution*) solves System (2.15) for the unknowns *starting at the bottom*. Thus, from the fourth equation, we obtain the last unknown

$$x_4 = \frac{-3}{-3} = 1. \quad (2.16)$$

Putting $x_4 = 1$ in the third equation gives us

$$2x_3 - 5 = -9, \quad (2.17)$$

and we find the next to last unknown

$$x_3 = \frac{-4}{2} = -2, \quad (2.18)$$

and so on. The solution is

$$x_1 = 3, \quad x_2 = 1, \quad x_3 = -2, \quad x_4 = 1. \quad (2.19)$$

2.1 Algorithm

Let us generalize the above example into an algorithm. We write the $n \times n$ system as follows:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{i1} & a_{i2} & a_{i3} & \cdots & a_{in} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_i \\ \vdots \\ b_n \end{pmatrix}. \quad (2.20)$$

Operations between *equations* correspond to operations between *rows* of the matrix.

Forward elimination

1st step. The first of these steps uses the first equation to produce $n - 1$ zeros as coefficients for each x_1 in all but the first equation. This is done by subtracting appropriate multiples of the first equation from the others. In this process, we refer to the first equation as the first **pivot equation** and to a_{11} as the first **pivot element**. For each of the remaining equations ($2 \leq i \leq n$), we compute

$$\begin{cases} a_{ij} \leftarrow a_{ij} - \left(\frac{a_{i1}}{a_{11}} \right) a_{1j} & (1 \leq j \leq n), \\ b_i \leftarrow b_i - \left(\frac{a_{i1}}{a_{11}} \right) b_1. \end{cases} \quad (2.21)$$

The symbol $\leftarrow$ indicates a **replacement**. Thus, the content of the memory location allocated to a_{ij} is replaced by $a_{ij} - (a_{i1}/a_{11})a_{1j}$, and so on. This is accomplished by the following line of pseudocode:

$$a_{ij} \leftarrow a_{ij} - (a_{i1}/a_{11})a_{1j}. \quad (2.22)$$

Note that the quantities (a_{i1}/a_{11}) are the **multipliers**. The new coefficient of x_1 in the i th equation becomes 0 because $a_{i1} - (a_{i1}/a_{11})a_{11} = 0$.

2nd step. After the first step, the system will be of the form

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22} & a_{23} & \cdots & a_{2n} \\ 0 & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & a_{i2} & a_{i3} & \cdots & a_{in} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & a_{n2} & a_{n3} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_i \\ \vdots \\ b_n \end{pmatrix}. \quad (2.23)$$

From here on, we do not alter the first equation, nor do we alter any of the coefficients for x_1 (since a multiplier times 0 subtracted from 0 is still 0). Thus, we can mentally ignore the first row and the first column

and repeat the process on the smaller system. With the second equation as the **pivot equation**, we compute for each remaining equation ($3 \leq i \leq n$)

$$\begin{cases} a_{ij} \leftarrow a_{ij} - \left(\frac{a_{i2}}{a_{22}} \right) a_{2j} & (2 \leq j \leq n), \\ b_i \leftarrow b_i - \left(\frac{a_{i2}}{a_{22}} \right) b_2. \end{cases} \quad (2.24)$$

kth step. Just prior to the k th step in the forward elimination, the system appears as follows:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & \cdots & \cdots & a_{1n} \\ 0 & a_{22} & a_{23} & \cdots & \cdots & \cdots & a_{2n} \\ 0 & 0 & a_{33} & \cdots & \cdots & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & & & \vdots \\ 0 & 0 & 0 & \cdots & a_{kk} & \cdots & a_{kn} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{ik} & \cdots & a_{in} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nk} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_k \\ \vdots \\ x_i \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_k \\ \vdots \\ b_i \\ \vdots \\ b_n \end{pmatrix}. \quad (2.25)$$

Here, a “wedge” of 0 coefficients has been created, and the first k equations have been processed and are now fixed. Using the k th equation as the **pivot equation**, we select multipliers to create 0’s as coefficients below the a_{kk} coefficient.

$$\begin{cases} a_{ij} \leftarrow a_{ij} - \left(\frac{a_{ik}}{a_{kk}} \right) a_{kj} & (k \leq j \leq n), \\ b_i \leftarrow b_i - \left(\frac{a_{ik}}{a_{kk}} \right) b_k. \end{cases} \quad (2.26)$$

Backward substitution

At the beginning of the back substitution phase, the linear system is of the form

$$\begin{array}{rcll} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots & \cdots + & a_{1n}x_n & = b_1 \\ a_{22}x_2 + a_{23}x_3 + \cdots & \cdots + & a_{2n}x_n & = b_2 \\ a_{33}x_3 + \cdots & + & a_{3n}x_n & = b_3 \\ \vdots & \vdots & \vdots & \\ a_{ii}x_i + a_{i,i+1}x_{i+1} + \cdots & \cdots + & a_{in}x_n & = b_i \\ \vdots & \vdots & \vdots & \\ a_{n-1,n-1}x_{n-1} + a_{n-1,n}x_n & & & = b_{n-1} \\ & & a_{nn}x_n & = b_n \end{array} \quad (2.27)$$

where the a_{ij} ’s and b_i ’s are *not* the original ones from the original system, but instead are the ones that have been altered by the elimination process.

The back substitution starts by solving the n th equation for x_n :

$$x_n = \frac{b_n}{a_{nn}}. \quad (2.28)$$

Then, using the $(n - 1)$ th equation, we solve for x_{n-1} :

$$x_{n-1} = \frac{1}{a_{n-1,n-1}} (b_{n-1} - a_{n-1,n}x_n). \quad (2.29)$$

We continue working upward, recovering each x_i by the formula

$$x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{j=i+1}^n a_{ij}x_j \right), \quad i = n - 1, n - 2, \dots, 1. \quad (2.30)$$

2.2 What changes for singular or rectangular systems?

So far, back substitution has recovered one unknown from each successive equation. This works particularly neatly for a square system with a nonzero pivot in every column. What happens if there are more equations than unknowns, fewer equations than unknowns, or equations that repeat information already present in the system?

The elementary row operations still preserve the solution set. What changes is the shape of the reduced system and what we can conclude from it. Throughout this discussion, we work in exact arithmetic and consider

$$A\mathbf{x} = \mathbf{b}, \quad A \in \mathbb{R}^{m \times n}, \quad \mathbf{x} \in \mathbb{R}^n, \quad \mathbf{b} \in \mathbb{R}^m. \quad (2.31)$$

The matrix has m rows because there are m equations, and n columns because there are n unknowns. The word *singular* refers to a square matrix that is not invertible. For a rectangular matrix, the relevant questions concern its rank and the compatibility of the equations with the right-hand side.

Definition 2 (Consistent and inconsistent linear systems). A *solution* of $A\mathbf{x} = \mathbf{b}$ is a single vector $\mathbf{x} \in \mathbb{R}^n$ whose entries satisfy *all* m equations simultaneously. The *solution set* is

$$\mathcal{S} = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{b}\}. \quad (2.32)$$

- The system is **consistent** if it has at least one solution; equivalently, $\mathcal{S} \neq \emptyset$.
- The system is **inconsistent** if it has no solution; equivalently, $\mathcal{S} = \emptyset$.

The same choice of the unknowns must satisfy every equation. Finding a different solution for each equation separately does not establish that the system is consistent.

Consistency is a question of *existence*, not *uniqueness*. A consistent system may have exactly one solution or infinitely many solutions. For example, the one-equation system $x_1 + x_2 = 1$ is consistent: $(x_1, x_2) = (1, 0)$ is a solution. It is not uniquely solvable, since every pair $(x_1, x_2) = (1 - t, t)$, with $t \in \mathbb{R}$, is also a solution.

In contrast, the system

$$\begin{aligned} x_1 + x_2 &= 1, \\ x_1 + x_2 &= 2 \end{aligned} \quad (2.33)$$

is inconsistent: the same sum cannot equal both 1 and 2. Subtracting the first equation from the second gives the contradiction $0 = 1$. Each equation individually has solutions, but they have no solution in common.

Thus, we first ask whether a solution exists. Only if it does do we ask whether it is unique or whether there are free choices remaining. Consistency is a property of the complete system, including $\mathbf{b}$; it cannot in general be decided from A alone. Here, “solution” means an exact solution. A vector that approximately satisfies the equations does not make an inconsistent system consistent.

Keep track of the pivot row and pivot column separately

For a general system, elimination puts the coefficient matrix in *row echelon form*: each nonzero coefficient row starts with a pivot, successive pivots lie farther to the right, and any zero coefficient rows lie at the bottom. The pivots need not lie on the main diagonal.

Suppose we are looking for a pivot in coefficient column j , starting at the current row p .

1. If the current entry is zero but a nonzero entry occurs below it in column j , exchange rows to bring a nonzero entry into the pivot position. Apply the exchange to the entire augmented matrix.
2. Once a nonzero pivot is available, eliminate the entries below it. Then advance to the next row and the next column.
3. If *every* entry in column j from row p downward is zero, skip that column. Advance to column $j + 1$, but keep the same current row p : this row may still supply a pivot farther to the right.

Continue until all coefficient columns have been considered or all rows have supplied pivots. We never divide by a zero candidate pivot. Skipping a column does not mean deleting a variable or setting it to zero.

The number of pivots in the coefficient part is the rank $r = \text{rank}(A)$, with $0 \leq r \leq \min(m, n)$. A *pivot variable* corresponds to a pivot column. In a consistent system, the remaining variables are *free variables*: we may choose their values and then use back substitution to determine the pivot variables.

Example 4 (Dependent equations: a singular square system with infinitely many solutions or none). Consider

$$A = \begin{pmatrix} 1 & 1 \\ 2 & 2 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 3 \\ c \end{pmatrix}. \quad (2.34)$$

The second coefficient row is twice the first, so A is singular. The operation $R_2 \leftarrow R_2 - 2R_1$ gives

$$\left(\begin{array}{cc|c} 1 & 1 & 3 \\ 2 & 2 & c \end{array} \right) \longrightarrow \left(\begin{array}{cc|c} 1 & 1 & 3 \\ 0 & 0 & c-6 \end{array} \right). \quad (2.35)$$

If $c = 6$, the last row is $0 = 0$, and the system is consistent. There is only one independent equation, $x_1 + x_2 = 3$. Choose $x_2 = t$ freely and solve for x_1 :

$$x_1 = 3 - t, \quad x_2 = t, \quad t \in \mathbb{R}. \quad (2.36)$$

Thus, the complete solution set is

$$\mathbf{x} = \begin{pmatrix} 3 \\ 0 \end{pmatrix} + t \begin{pmatrix} -1 \\ 1 \end{pmatrix}, \quad t \in \mathbb{R}. \quad (2.37)$$

Setting $t = 0$ gives one solution, but it does not give all solutions. Geometrically, the two original equations describe the same line.

If $c \neq 6$, the last row instead requires $0 = c - 6$, which is impossible, so the system is inconsistent. For example, $c = 7$ gives $0 = 1$. The two original lines are parallel and distinct. In both cases, $\text{rank}(A) = 1$; the right-hand side determines whether the dependent equations agree or contradict one another.

The missing second pivot prevents us from recovering a unique solution by ordinary triangular back substitution. Elimination still provides the correct conclusion: either infinitely many solutions or no solution.

Example 5 (A rectangular system). Consider

$$\begin{aligned} x_1 + 2x_2 - x_3 &= 3, \\ 2x_1 + 4x_2 + x_3 &= 9. \end{aligned} \tag{2.38}$$

Eliminating the first entry of the second row gives

$$\left(\begin{array}{ccc|c} 1 & 2 & -1 & 3 \\ 2 & 4 & 1 & 9 \end{array} \right) \xrightarrow{R_2 \leftarrow R_2 - 2R_1} \left(\begin{array}{ccc|c} 1 & 2 & -1 & 3 \\ 0 & 0 & 3 & 3 \end{array} \right). \tag{2.39}$$

The next diagonal entry is zero, but the second row is not zero. We skip coefficient column 2 and use the entry 3 in column 3 as the next pivot. The pivot variables are x_1 and x_3 ; the free variable is x_2 . In particular, the fact that column 2 is a nonpivot column does not require all of its entries to be zero.

The last equation gives $x_3 = 1$. Setting $x_2 = t$, the first equation gives

$$x_1 + 2t - 1 = 3, \quad \text{so} \quad x_1 = 4 - 2t. \tag{2.40}$$

Hence

$$\mathbf{x} = \begin{pmatrix} 4 - 2t \\ t \\ 1 \end{pmatrix} = \begin{pmatrix} 4 \\ 0 \\ 1 \end{pmatrix} + t \begin{pmatrix} -2 \\ 1 \\ 0 \end{pmatrix}, \quad t \in \mathbb{R}. \tag{2.41}$$

There are two pivots and three unknowns, leaving one free parameter. The two planes intersect in a line. Notice that a pivot variable need not be a fixed constant: x_1 depends on the chosen value of x_2 . It is determined uniquely *once the free variable has been chosen*.

Here $\text{rank}(A) = 2 = m$, so the matrix has *full row rank*. It is not rank deficient, even though the system has infinitely many solutions.

Back substitution when the pivots are not on the diagonal. Suppose the transformed coefficient matrix is $\tilde{A}$, its right-hand side is $\tilde{\mathbf{b}}$, and its r pivot columns are $j_1 < j_2 < \dots < j_r$. Once consistency has been checked, assign values to the nonpivot variables. Then work upward through the pivot rows:

$$x_{j_i} = \frac{1}{\tilde{a}_{i,j_i}} \left(\tilde{b}_i - \sum_{\ell=j_i+1}^n \tilde{a}_{i\ell} x_\ell \right), \quad i = r, r-1, \dots, 1. \tag{2.42}$$

An empty sum is interpreted as zero. Every variable on the right is either a free variable whose value has already been chosen or a pivot variable recovered from a lower row. The divisor is the nonzero pivot $\tilde{a}_{i,j_i}$, which need not be a diagonal entry.

Example 6 (A rectangular systems: extra equations test compatibility). Consider

$$\begin{aligned}x_1 + x_2 &= 3, \\2x_1 - x_2 &= 0, \\3x_1 &= c.\end{aligned}\tag{2.43}$$

Apply $R_2 \leftarrow R_2 - 2R_1$ and $R_3 \leftarrow R_3 - 3R_1$:

$$\left(\begin{array}{cc|c}1 & 1 & 3 \\2 & -1 & 0 \\3 & 0 & c\end{array}\right) \longrightarrow \left(\begin{array}{cc|c}1 & 1 & 3 \\0 & -3 & -6 \\0 & -3 & c-9\end{array}\right).\tag{2.44}$$

Then $R_3 \leftarrow R_3 - R_2$ gives

$$\left(\begin{array}{cc|c}1 & 1 & 3 \\0 & -3 & -6 \\0 & 0 & c-3\end{array}\right).\tag{2.45}$$

There is a pivot in each of the two coefficient columns, but we must still inspect the last row.

- If $c = 3$, the last row is redundant. Back substitution gives $x_2 = 2$ and $x_1 = 1$, so there is a unique exact solution.
- If $c \neq 3$, the last row is contradictory. There is no exact solution, despite the pivot in every coefficient column.

This condition can also be seen before elimination: the left-hand side of the third equation is the sum of the first two left-hand sides. Its right-hand side must therefore equal $3 + 0 = 3$.

Here $\text{rank}(A) = 2 = n$, so A has *full column rank*. This guarantees *at most one* solution. Existence still depends on whether all the equations agree at that solution. Geometrically, the first two lines intersect at $(1, 2)$; the third line must pass through the same point.

Rectangular matrices can also have dependent rows or columns

A matrix has *full rank* if $\text{rank}(A) = \min(m, n)$, and is *rank deficient* otherwise. Shape alone does not tell us the rank. In particular, more unknowns than equations do not guarantee consistency, and more equations than unknowns do not guarantee uniqueness.

Example 7 (A rectangular system can be inconsistent). For two equations in three unknowns, consider

$$\left(\begin{array}{ccc|c}1 & -1 & 2 & 2 \\3 & -3 & 6 & c\end{array}\right) \xrightarrow{R_2 \leftarrow R_2 - 3R_1} \left(\begin{array}{ccc|c}1 & -1 & 2 & 2 \\0 & 0 & 0 & c-6\end{array}\right).\tag{2.46}$$

The coefficient rank is 1, less than $\min(2, 3) = 2$. If $c = 7$, the last equation is $0 = 1$. Having three unknowns does not help: no choices can make contradictory equations hold simultaneously. If $c = 6$, the system is consistent. Both x_2 and x_3 are free, so

$$x_2 = s, \quad x_3 = t, \quad x_1 = 2 + s - 2t, \quad s, t \in \mathbb{R}. \quad (2.47)$$

There are $3 - 1 = 2$ free parameters. The solution set is a plane, whereas the preceding full-row-rank wide example had a line of solutions.

Example 8 (A rectangular system can have infinitely many solutions). Consider the three equations represented by

$$\left(\begin{array}{cc|c} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{array} \right). \quad (2.48)$$

The operations $R_2 \leftarrow R_2 - 2R_1$ and $R_3 \leftarrow R_3 - 3R_1$ give

$$\left(\begin{array}{cc|c} 1 & 2 & 3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{array} \right). \quad (2.49)$$

All three original equations express the same restriction, $x_1 + 2x_2 = 3$. Although $m = 3 > 2 = n$, there is only one pivot, so

$$x_1 = 3 - 2t, \quad x_2 = t, \quad t \in \mathbb{R}. \quad (2.50)$$

The extra equations add no independent information. This tall matrix is rank deficient, and the consistent system has infinitely many solutions.

The general classification

The examples separate two questions: *Are the equations compatible?* If they are, *how many variables remain free?* The two ranks answer these questions precisely.

Theorem 1 (Classification by coefficient and augmented rank). Let $A \in \mathbb{R}^{m \times n}$, $m, n \geq 1$, let $\mathbf{b} \in \mathbb{R}^m$, and set $r = \text{rank}(A)$. In exact arithmetic:

1. The system is consistent if and only if $\text{rank}((A \mid \mathbf{b})) = r$.
2. If the system is consistent, it has exactly $n - r$ free parameters. It has a unique solution when $r = n$, and infinitely many solutions when $r < n$.
3. If $\text{rank}((A \mid \mathbf{b})) > r$, the augmented rank is $r + 1$ and the system has no solution. Elimination reveals a row $0 = c$ with $c \neq 0$.

Proof. The system is solvable exactly when $\mathbf{b}$ is a linear combination of the columns of A . Appending such a column does not change the rank. If $\mathbf{b}$ is not in the column space, it adds one independent column, so the augmented rank is $r + 1$.

Row operations preserve both ranks and the exact solution set. In echelon form, there are r coefficient pivots. If no contradictory row occurs, we may choose the $n - r$ nonpivot variables freely and recover all pivot variables by back substitution. Every choice gives one solution, and distinct choices give distinct solution vectors. If the augmented rank exceeds the coefficient rank, row echelon form of the full augmented matrix has an extra pivot in the appended column, and its row asserts $0 = c$ with $c \neq 0$. ■

A pivot in the appended column does not correspond to another unknown: it records a contradiction. Count pivots in the coefficient matrix when determining the number of free variables.

Key Idea

- **Square and singular:** $r < n$. A consistent system has infinitely many solutions; an inconsistent system has none.
- **Wide, $m < n$:** since $r \leq m < n$, a unique solution is impossible. A consistent system has at least $n - m$ free parameters. If $r = m$, the system is consistent for every right-hand side: there are no zero coefficient rows left to impose compatibility conditions.
- **Tall, $m > n$:** all three outcomes are possible. Full column rank gives uniqueness *provided the system is consistent*. With rank deficiency, a consistent system has free parameters.

For a consistent system, let $\mathbf{x}_p$ be one particular solution. The complete solution set can also be written as

$$\{\mathbf{x} : A\mathbf{x} = \mathbf{b}\} = \{\mathbf{x}_p + \mathbf{z} : A\mathbf{z} = \mathbf{0}\} = \mathbf{x}_p + \ker(A). \quad (2.51)$$

Indeed, adding a nullspace vector does not change $A\mathbf{x}$, and the difference of any two solutions lies in the nullspace. The free parameters describe these directions of nonuniqueness. For the homogeneous system $\mathbf{b} = \mathbf{0}$, consistency is automatic.

What if there is no exact solution? Elimination has still answered the exact-solution question. If the application instead asks for the best fit, a common new objective is

$$\min_{\mathbf{x} \in \mathbb{R}^n} \|A\mathbf{x} - \mathbf{b}\|_2^2. \quad (2.52)$$

We cannot obtain this solution by simply discarding contradictory rows. Row operations preserve exact equations, but generally change the least-squares objective: multiplying an equation by 100 multiplies its squared residual by 10^4 . Least squares and the pseudoinverse will address best fits and the selection of a particular solution when solutions are not unique.

Exact zeros versus computed small numbers. The classification above assumes exact arithmetic. In floating-point arithmetic, rounding can turn a mathematically zero entry into a small nonzero number. Conversely, a small nonzero entry may carry information that is difficult to distinguish from numerical error. Decisions about numerical rank therefore require a scale-sensitive tolerance; a single small pivot is not, by itself, a reliable test of singularity.

3 Gaussian elimination with scaled partial pivoting

3.1 Why Gaussian elimination can fail

To see why the naive Gaussian elimination algorithm is unsatisfactory, consider the following system:

$$\begin{aligned} 0x_1 + x_2 &= 1, \\ x_1 + x_2 &= 2. \end{aligned} \tag{3.1}$$

Our algorithm from the previous section would attempt to subtract some multiple of the first equation from the second to produce a 0 as the coefficient for x_1 in the second equation. This, of course, is impossible, so the algorithm fails if $a_{11} = 0$.

If a numerical procedure actually fails for some values of the data, then the procedure is probably untrustworthy for values of the data *near* the failing values. To test this dictum, consider the system

$$\begin{aligned} \varepsilon x_1 + x_2 &= 1, \\ x_1 + x_2 &= 2, \end{aligned} \tag{3.2}$$

in which ε is a small number different from 0. Now, our Gaussian elimination algorithm works, and after forward elimination it produces the system

$$\begin{aligned} \varepsilon x_1 + x_2 &= 1, \\ (1 - \varepsilon^{-1})x_2 &= 2 - \varepsilon^{-1}. \end{aligned} \tag{3.3}$$

(This comes from doing $R_2 \leftarrow R_2 - \frac{1}{\varepsilon}R_1$.) In the back substitution, the finite-precision arithmetic can behave as follows:

$$x_2 = \frac{2 - \varepsilon^{-1}}{1 - \varepsilon^{-1}} \approx 1, \quad x_1 = \varepsilon^{-1}(1 - x_2) \approx 0. \tag{3.4}$$

But ε^{-1} may be large! So if this calculation is performed by a computer that has a fixed word length, then for small values of ε , both $(2 - \varepsilon^{-1})$ and $(1 - \varepsilon^{-1})$ would be computed as $-\varepsilon^{-1}$. In other words, for values of ε sufficiently close to 0, the computer calculates x_2 as 1 and then x_1 as 0. Since the *correct* solution is

$$x_1 = \frac{1}{1 - \varepsilon} \approx 1, \quad x_2 = \frac{1 - 2\varepsilon}{1 - \varepsilon} \approx 1, \tag{3.5}$$

the relative error in the computed solution for x_1 is extremely large: 100%.

Reordered sample 2×2 systems. Actually, the naive Gaussian elimination algorithm works well on systems (3.1) and (3.2) if the equations are first permuted:

$$\begin{aligned} x_1 + x_2 &= 2, \\ 0x_1 + x_2 &= 1, \end{aligned} \tag{3.6}$$

and

$$\begin{aligned} x_1 + x_2 &= 2, \\ \varepsilon x_1 + x_2 &= 1. \end{aligned} \tag{3.7}$$

System (3.6) is easily solved obtaining $x_2 = 1$ and $x_1 = 2 - x_2 = 1$. Moreover, System (3.7) becomes

$$\begin{aligned} x_1 + x_2 &= 2, \\ (1 - \varepsilon)x_2 &= 1 - 2\varepsilon \end{aligned} \quad (3.8)$$

after forward elimination. Then from the back substitution, the solution is computed as

$$x_2 = \frac{1 - 2\varepsilon}{1 - \varepsilon} \approx 1, \quad x_1 = 2 - x_2 \approx 1. \quad (3.9)$$

Notice that we do *not* have to rearrange the equations in the system: it is necessary only to select a different pivot row. The difficulty in system (3.2) is *not* due simply to ε being small, but rather to its being small relative to other coefficients in the same row. To verify this, consider

$$\begin{aligned} x_1 + \varepsilon^{-1}x_2 &= \varepsilon^{-1}, \\ x_1 + x_2 &= 2. \end{aligned} \quad (3.10)$$

System (3.10) is mathematically equivalent to (3.2). The naive Gaussian elimination algorithm fails here! It produces the triangular system

$$\begin{aligned} x_1 + \varepsilon^{-1}x_2 &= \varepsilon^{-1}, \\ (1 - \varepsilon^{-1})x_2 &= 2 - \varepsilon^{-1}, \end{aligned} \quad (3.11)$$

and then, in the back substitution, it produces the erroneous result

$$x_2 = \frac{2 - \varepsilon^{-1}}{1 - \varepsilon^{-1}} \approx 1, \quad x_1 = \varepsilon^{-1} - \varepsilon^{-1}x_2 \approx 0. \quad (3.12)$$

This situation can be resolved by interchanging the two equations in system (3.10):

$$\begin{aligned} x_1 + x_2 &= 2, \\ x_1 + \varepsilon^{-1}x_2 &= \varepsilon^{-1}. \end{aligned} \quad (3.13)$$

Now the naive Gaussian elimination algorithm can be applied, resulting in the system

$$\begin{aligned} x_1 + x_2 &= 2, \\ (\varepsilon^{-1} - 1)x_2 &= \varepsilon^{-1} - 2. \end{aligned} \quad (3.14)$$

The solution is

$$x_2 = \frac{\varepsilon^{-1} - 2}{\varepsilon^{-1} - 1} \approx 1, \quad x_1 = 2 - x_2 \approx 1, \quad (3.15)$$

which is the correct solution to the working precision.

3.2 Scaled partial pivoting

These simple examples should make it clear that the *order* in which we treat the equations significantly affects the accuracy of the elimination algorithm in the computer. In the naive Gaussian elimination algorithm, we use the first equation to eliminate x_1 from the equations that follow it. Then we use the second equation to eliminate x_2 from the equations that follow it, and so on. The order in which the equations are used as pivot equations is the **natural order** $\{1, 2, \dots, n\}$. Note that the last equation (equation number n) is *not* used as an operating equation in the natural ordering: at no time are multiples of it subtracted from other equations in the naive algorithm.

Types of Pivoting. From the previous examples, it is clear that a strategy is needed for selecting new pivots at each stage in Gaussian elimination. Perhaps the best approach is **complete pivoting**, which involves searches over all entries in the submatrices for the largest entry in absolute value and then interchanges rows and columns to move it into the pivot position. This would be quite expensive, since it involves a great amount of searching and data movement. However, searching just the first column in the submatrix at each stage accomplishes most of what is needed (avoiding small or zero pivots). This is **partial pivoting**, and it is the most common approach. It does not involve an examination of the elements in the rows, since it looks only at column entries. We advocate a strategy that simulates a scaling of the row vectors and then selects as a pivot element the relatively largest entry in a column. Also, rather than interchanging rows to move the desired element into the pivot position, we use an indexing array to avoid the data movement. This procedure is not as expensive as complete pivoting, and it goes beyond partial pivoting to include an examination of all elements in the original matrix. Of course, other strategies for selecting pivot elements could be used.

Scaled Partial Pivoting. The Gaussian elimination algorithm we develop uses the equations in an order that is determined by the actual system being solved. For instance, if the algorithm were asked to solve System (3.1) or (3.2), the order in which the equations would be used as pivot equations would *not* be the natural order $\{1, 2\}$, but rather $\{2, 1\}$. This order is automatically determined by the computer program. The order in which the equations are employed is denoted by the row vector $(\ell_1 \ell_2 \cdots \ell_n)$, where ℓ_n is *not* actually being used in the forward elimination phase. Here, the ℓ_i are integers from 1 to n in a possibly different order. We call $\ell = (\ell_1 \ell_2 \cdots \ell_n)$ the **index vector**. The strategy to be described now for determining the index vector is termed **scaled partial pivoting** (SPP).

Scale Vector. At the beginning, a **scale factor** must be computed for each equation in the system. We define

$$s_i = \max_{1 \leq j \leq n} |a_{ij}|, \quad 1 \leq i \leq n. \quad (3.16)$$

These n numbers are recorded in the **scale vector** $s = (s_1 s_2 \cdots s_n)$.

In starting the forward elimination process, we do *not* arbitrarily use the first equation as the pivot equation. Instead, we use the equation for which the ratio $|a_{i,1}|/s_i$ is greatest. Let ℓ_1 be the first index for which this ratio is greatest. Now appropriate multiples of equation ℓ_1 are subtracted from the other equations to create 0's as coefficients for each x_1 except in the pivot equation.

Example 9 (Simple partial pivoting and row scaling). In certain situations, Gaussian elimination with the simple partial pivoting strategy may lead to an incorrect solution. Consider the augmented matrix

$$\left(\begin{array}{cc|c} 2 & 2c & 2c \\ 1 & 1 & 2 \end{array} \right), \quad (3.17)$$

where $c > 1$ is a parameter that can take on very large numerical values and the variables are x and y . The first row is selected as the pivot row by choosing the larger number in the first column. Since the multiplier is $1/2$, one step in the row reduction process brings us to

$$\left(\begin{array}{cc|c} 2 & 2c & 2c \\ 0 & 1-c & 2-c \end{array} \right). \quad (3.18)$$

Now suppose that we are working with a computer of limited word length. So in this computer, we obtain $1 - c \approx -c$ and $2 - c \approx -c$. Consequently, the computer returns these numbers:

$$\left(\begin{array}{cc|c} 2 & 2c & 2c \\ 0 & -c & -c \end{array} \right). \quad (3.19)$$

Thus, we obtain $y = 1$ and $x = 0$ as the solution; whereas the exact solution is

$$x = \frac{c}{c-1} \approx 1, \quad y = \frac{c-2}{c-1} \approx 1. \quad (3.20)$$

On the other hand, Gaussian elimination with scaled partial pivoting selects the second row as the pivot row. The scaling constants are $s = (2c \ 1)$. The larger of the two ratios for selecting the pivot row from $\{2/(2c), 1\}$ is the second one. Now the multiplier is 2, and one step in the row reduction process brings us to

$$\left(\begin{array}{cc|c} 0 & 2c-2 & 2c-4 \\ 1 & 1 & 2 \end{array} \right). \quad (3.21)$$

On our computer of limited word length, we find $2c-2 \approx 2c$ and $2c-4 \approx 2c$. Consequently, the computer returns these numbers:

$$\left(\begin{array}{cc|c} 0 & 2c & 2c \\ 1 & 1 & 2 \end{array} \right). \quad (3.22)$$

Now we obtain $y = 1$ and $x = 1$, the correct solution to the working precision in this large- c regime.

Steps of Gaussian elimination with scaled partial pivoting

Step 1. The best way of keeping track of the indices is as follows: At the beginning, define the index vector ℓ to be $(\ell_1 \ \ell_2 \ \cdots \ \ell_n) = (1 \ 2 \ \cdots \ n)$. In Step 1, select j to be the first index associated with the largest ratio in the set:

$$\left\{ \frac{|a_{\ell_i,1}|}{s_{\ell_i}} : 1 \leq i \leq n \right\}. \quad (3.23)$$

Now interchange ℓ_j with ℓ_1 in the index vector ℓ . Next, use multipliers

$$\frac{a_{\ell_i,1}}{a_{\ell_1,1}} \quad (3.24)$$

times row ℓ_1 , and subtract from equations ℓ_i for $2 \leq i \leq n$. It is important to note that only entries in ℓ are being interchanged and *not* the equations. This eliminates the time-consuming and unnecessary process of moving the coefficients of equations around in the computer memory!

Step 2. In Step 2, the ratios

$$\left\{ \frac{|a_{\ell_i,2}|}{s_{\ell_i}} : 2 \leq i \leq n \right\} \quad (3.25)$$

are scanned. If j is the first index for the largest ratio, interchange ℓ_j with ℓ_2 in ℓ . Then multipliers

$$\frac{a_{\ell_i,2}}{a_{\ell_2,2}} \quad (3.26)$$

times equation ℓ_2 are subtracted from equations ℓ_i for $3 \leq i \leq n$.

Step k . At Step k , select j to be the first index corresponding to the largest of the ratios,

$$\left\{ \frac{|a_{\ell_i,k}|}{s_{\ell_i}} : k \leq i \leq n \right\}, \quad (3.27)$$

and interchange ℓ_j and ℓ_k in index vector ℓ . Then multipliers

$$\frac{a_{\ell_i,k}}{a_{\ell_k,k}} \quad (3.28)$$

times pivot equation ℓ_k are subtracted from equations ℓ_i for $k+1 \leq i \leq n$.

Notice that the scale factors are *not* changed after each pivot step. Intuitively, one might think that after each step in the Gaussian algorithm, the remaining (modified) coefficients should be used to recompute the scale factors instead of using the original scale vector. Of course, this could be done, but it is generally believed that the extra computations involved in this procedure are not worthwhile for the majority of linear systems.

Example 10 (Comparing pivoting strategies in five-digit arithmetic). Solve this system of linear equations:

$$\begin{aligned} 0.0001x + y &= 1, \\ x + y &= 2 \end{aligned} \quad (3.29)$$

using in order no pivoting, partial pivoting, and scaled partial pivoting. Carry at most five significant digits of precision (rounding) to see how finite precision computations and roundoff errors can affect the calculations.

Solution. By direct substitution, it is easy to verify that the true solution is $x = 1.0001$ and $y = 0.99990$ to five significant digits.

No Pivoting. For no pivoting, the first equation in the original system is the pivot equation, and the multiplier is $1/0.0001 = 10000$. Multiplying the first equation by this multiplier and subtracting the second equation from the result, the necessary calculations are $(10000)(0.0001) - 1 = 0$, $(10000)(1) - 1 = 9999$, and $(10000)(1) - 2 = 9998$. The new system of equations is

$$\begin{aligned} 0.0001x + y &= 1, \\ 9999y &= 9998. \end{aligned} \quad (3.30)$$

From the second equation, we obtain $y = 9998/9999 \approx 0.99990$. Using this result and the first equation, we find $0.0001x = 1 - y = 1 - 0.99990 = 0.0001$ and $x = 0.0001/0.0001 = 1$. Notice that we have lost the right-most significant digit in the correct value of x !

Partial pivoting. We repeat the solution using partial pivoting in the original system. Examining the first column of x coefficients $(0.0001, 1)$, we see that the second is larger, so the second equation is used as the pivot equation. We can interchange the two equations, obtaining

$$\begin{aligned} x + y &= 2, \\ 0.0001x + y &= 1. \end{aligned} \quad (3.31)$$

The multiplier is $0.0001/1 = 0.0001$. This multiple of the first equation is subtracted from the second equation. The calculations are $(-0.0001)(1) + 0.0001 = 0$, $1 - (0.0001)(1) = 0.99990$, and $1 - (0.0001)(2) = 0.99980$. The new system of equations is

$$\begin{aligned} x + y &= 2, \\ 0.99990y &= 0.99980. \end{aligned} \quad (3.32)$$

We obtain $y = 0.99980/0.99990 \approx 0.99990$. Now, using the pivot equation $x + y = 2$ and this value, we find $x = 2 - y = 2 - 0.99990 = 1.0001$. Both computed values of x and y are correct to five significant digits!

Scaled partial pivoting. We repeat the solution using scaled partial pivoting on the original system. Since the scaling constants are $s = (1 \ 1)$ and the ratios for determining the pivot equation are $(0.0001/1, 1/1)$, the second equation is now the pivot equation. We do not actually interchange the equations, but work with an index array $\ell = (2 \ 1)$ that tells us to use the second equation as the first pivot equation. The rest of the calculations are the same as those for partial pivoting with the results correct to five significant digits. We cannot promise that scaled partial pivoting is better than partial pivoting in all cases, but it clearly has some advantages. For example, suppose that we want to force the first equation in the original system to be the pivot equation and multiply it by a large number such as 20,000, obtaining

$$\begin{aligned} 2x + 20000y &= 20000, \\ x + y &= 2. \end{aligned} \quad (3.33)$$

Partial pivoting ignores the fact that the coefficients in the first equation differ by orders of magnitude and selects the first equation as the pivot equation. However, scaled partial pivoting uses the scaling constants $s = (20000 \ 1)$, and the ratios for determining the pivot equations are $(2/20000, 1/1)$. Scaled partial pivoting continues to select the second equation as the pivot equation!

Example 11 (A larger numerical example). Let us now consider a more concrete example:

$$\begin{pmatrix} 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \\ 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} -19 \\ -34 \\ 16 \\ 26 \end{pmatrix}. \quad (3.34)$$

The index vector is $\ell = (1 \ 2 \ 3 \ 4)$ at the beginning. The scale vector does not change throughout the procedure and is $s = (13 \ 18 \ 6 \ 12)$.

Step 1. To determine the first pivot row, we look at four ratios:

$$\left\{ \frac{|a_{\ell_i,1}|}{s_{\ell_i}} : i = 1, 2, 3, 4 \right\} = \left\{ \frac{3}{13}, \frac{6}{18}, \frac{6}{6}, \frac{12}{12} \right\} \approx \{0.23, 0.33, 1.0, 1.0\}. \quad (3.35)$$

We select the index j as the *first* occurrence of the largest value of these ratios. In this example, the largest of these occurs for the index $j = 3$. So row three is to be the pivot equation in Step $k = 1$ of the elimination process. In the index vector ℓ , entries ℓ_k and ℓ_j are interchanged so that the new index vector

is $\ell = (3 \ 2 \ 1 \ 4)$. Thus, the pivot equation is ℓ_k , which is $\ell_1 = 3$. Now appropriate multiples of the third equation are subtracted from the other equations so as to create 0's as coefficients for x_1 in each of those equations. Explicitly, $\frac{1}{2}$ times row three is subtracted from row one, -1 times row three is subtracted from row two, and 2 times row three is subtracted from row four. The result is

$$\begin{pmatrix} 0 & -12 & 8 & 1 \\ 0 & 2 & 3 & -14 \\ 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} -27 \\ -18 \\ 16 \\ -6 \end{pmatrix}. \quad (3.36)$$

Step 2. In Step $k = 2$, we use the index vector $\ell = (3 \ 2 \ 1 \ 4)$ and scan the ratios corresponding to rows two, one, and four:

$$\left\{ \frac{|a_{\ell_i,2}|}{s_{\ell_i}} : i = 2, 3, 4 \right\} = \left\{ \frac{2}{18}, \frac{12}{13}, \frac{4}{12} \right\} \approx \{0.11, 0.92, 0.33\}, \quad (3.37)$$

looking for the largest value. We find that the largest is the second ratio, and we therefore set $j = 3$ and interchange ℓ_k with ℓ_j in the index vector. Thus, the index vector becomes $\ell = (3 \ 1 \ 2 \ 4)$. The pivot equation for Step 2 in the elimination is now row one, and $\ell_2 = 1$. Next, multiples of the first equation are subtracted from the second equation and the fourth equation. The appropriate multiples are $-\frac{1}{6}$ and $\frac{1}{3}$, respectively. The result is

$$\begin{pmatrix} 0 & -12 & 8 & 1 \\ 0 & 0 & 13/3 & -83/6 \\ 6 & -2 & 2 & 4 \\ 0 & 0 & -2/3 & 5/3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} -27 \\ -45/2 \\ 16 \\ 3 \end{pmatrix}. \quad (3.38)$$

Step 3. The final Step $k = 3$ is to examine the ratios corresponding to rows two and four:

$$\left\{ \frac{|a_{\ell_i,3}|}{s_{\ell_i}} : i = 3, 4 \right\} = \left\{ \frac{13/3}{18}, \frac{2/3}{12} \right\} \approx \{0.24, 0.06\} \quad (3.39)$$

with the index vector $\ell = (3 \ 1 \ 2 \ 4)$. The larger value is the first, so we set $j = 3$. Since this is Step $k = 3$, interchanging ℓ_k with ℓ_j leaves the index vector unchanged, $\ell = (3 \ 1 \ 2 \ 4)$. The pivot equation is row two and $\ell_3 = 2$, and we subtract $-\frac{2}{13}$ times the second equation from the fourth equation.

So the forward elimination phase ends with the final system

$$\begin{pmatrix} 0 & -12 & 8 & 1 \\ 0 & 0 & 13/3 & -83/6 \\ 6 & -2 & 2 & 4 \\ 0 & 0 & 0 & -6/13 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} -27 \\ -45/2 \\ 16 \\ -6/13 \end{pmatrix}. \quad (3.40)$$

The order in which the pivot equations were selected is displayed in the final index vector $\ell = (3 \ 1 \ 2 \ 4)$. Now, reading the entries in the index vector from the last to the first, we have the order in which the back substitution is to be performed. The solution is obtained by using equation $\ell_4 = 4$ to determine x_4 , and

then equation $\ell_3 = 2$ to find x_3 , and so on. Carrying out the calculations, we have

$$\begin{aligned} x_4 &= \frac{1}{-6/13} [-6/13] = 1, \\ x_3 &= \frac{1}{13/3} [(-45/2) + (83/6)(1)] = -2, \\ x_2 &= \frac{1}{-12} [-27 - 8(-2) - 1(1)] = 1, \\ x_1 &= \frac{1}{6} [16 + 2(1) - 2(-2) - 4(1)] = 3. \end{aligned} \tag{3.41}$$

Hence, the solution is $\mathbf{x} = (3 \ 1 \ -2 \ 1)$.

4 How much computation is required?

4.1 The arithmetic model and asymptotic notation

We distinguish three questions: how many arithmetic operations an algorithm performs, how much storage it requires, and how long its implementation takes to run. These quantities are related, but they are not interchangeable. An operation count must specify both the algorithm and the computational model [1, 5].

Unless stated otherwise, matrices and vectors are real, dimensions are positive integers, and arithmetic uses a fixed floating-point precision. We count one scalar addition, subtraction, multiplication, or division as one *floating-point operation*, abbreviated *flop*. A multiplication followed by an addition counts as two flops, even if the processor executes them using one fused multiply-add instruction. Comparisons, absolute values, assignments, indexing, and memory transfers are excluded from these counts and discussed separately when relevant. Square roots, if required, must also be accounted for separately.

This is a unit-cost arithmetic model. It does not measure the bit cost of operations on integers or exact rational numbers whose representations grow during the computation. Furthermore, one complex multiplication is not one real flop: its usual expansion requires four real multiplications and two real additions or subtractions.

Definition 3 (Asymptotic upper, lower, and tight bounds). Let $f, g : \mathbb{N} \rightarrow [0, \infty)$, with $g(n) > 0$ for all sufficiently large n .

1. We write $f(n) = O(g(n))$ if there exist constants $C > 0$ and $n_0 \in \mathbb{N}$ such that

$$f(n) \leq Cg(n) \quad \text{for every } n \geq n_0. \tag{4.1}$$

This is an asymptotic upper bound.

2. We write $f(n) = \Omega(g(n))$ if there exist constants $c > 0$ and $n_0 \in \mathbb{N}$ such that

$$cg(n) \leq f(n) \quad \text{for every } n \geq n_0. \tag{4.2}$$

This is an asymptotic lower bound.

3. We write $f(n) = \Theta(g(n))$ if there exist constants $c, C > 0$ and $n_0 \in \mathbb{N}$ such that

$$cg(n) \leq f(n) \leq Cg(n) \quad \text{for every } n \geq n_0. \tag{4.3}$$

This is a tight asymptotic bound.

The constants must be independent of n . For a signed remainder $r(n)$, the notation $r(n) = O(g(n))$ means $|r(n)| \leq Cg(n)$ eventually. Strictly speaking, $O(g)$ denotes a class of functions; the conventional notation $f = O(g)$ means membership in that class.

An upper bound need not be tight. For example, $n^2 = O(n^3)$ is true, but $n^2 = \Theta(n^3)$ is false. Also, the symbols O , Ω , and Θ do not themselves mean worst case, best case, or average case. If costs depend on the entries of the input, one must first specify which cost function is being bounded. A worst-case cost at size n , for instance, is the largest cost over all admissible inputs of that size.

Definition 4 (Lower-order terms and asymptotic equivalence). For an eventually positive function g , we write $f(n) = o(g(n))$ if

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0. \quad (4.4)$$

Equivalently, for every $\varepsilon > 0$ there is an n_0 such that $|f(n)| \leq \varepsilon g(n)$ whenever $n \geq n_0$. We write $f(n) \sim g(n)$ if

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 1. \quad (4.5)$$

Thus, for $c > 0$, a statement $T(n) = cn^p + O(n^{p-1})$ implies both $T(n) \sim cn^p$ and $T(n) = \Theta(n^p)$. It supplies a leading coefficient that $\Theta(n^p)$ alone does not specify. The converse implications need not hold.

Example 12 (An exact count, its order, and its leading term). Suppose an algorithm uses $T(n) = 2n^3 - n^2$ flops. For every $n \geq 1$,

$$n^3 \leq 2n^3 - n^2 \leq 2n^3, \quad (4.6)$$

so $T(n) = \Theta(n^3)$. Moreover,

$$\frac{T(n)}{2n^3} = 1 - \frac{1}{2n} \rightarrow 1, \quad (4.7)$$

so $T(n) \sim 2n^3$. The statement $T(n) = O(n^4)$ is also true, but gives a less informative upper bound.

For functions of several dimensions, bounds must hold uniformly over the stated range: their constants cannot depend on those dimensions. We will often give exact expressions to make this dependence explicit.

4.2 Vector operations and matrix multiplication

Adding two length- n vectors takes n additions. Their dot product $\mathbf{x}^\top \mathbf{y} = \sum_{j=1}^n x_j y_j$ requires n multiplications and $n - 1$ additions when the sum starts with its first product: $2n - 1$ flops in total. The update $\mathbf{y} \leftarrow \alpha \mathbf{x} + \mathbf{y}$ takes $2n$ flops. An outer product $\mathbf{u} \mathbf{v}^\top$, with $\mathbf{u} \in \mathbb{R}^m$ and $\mathbf{v} \in \mathbb{R}^n$, requires mn multiplications. Matrix addition and scalar multiplication likewise require one operation per entry.

For $A \in \mathbb{R}^{m \times n}$, the classical dense computation of $A\mathbf{x}$ uses m dot products of length n . Hence

$$T_{A\mathbf{x}} = m(2n - 1) = 2mn - m = \Theta(mn). \quad (4.8)$$

If a sparse representation instead visits only nonzeros and accumulates each contribution into an initially zero output vector, the count is $2 \text{nnz}(A)$ flops. Including output initialization and indexing gives $O(\text{nnz}(A) + m)$ total work in the corresponding unit-cost model.

For $A \in \mathbb{R}^{m \times k}$ and $B \in \mathbb{R}^{k \times n}$, their product $C = AB$ is an $m \times n$ matrix with entries

$$c_{ij} = \sum_{\ell=1}^k a_{i\ell} b_{\ell j}, \quad 1 \leq i \leq m, \quad 1 \leq j \leq n. \quad (4.9)$$

The classical algorithm computes mn dot products of length k . With the same summation convention,

$$T_{AB} = mn(2k - 1) = 2mnk - mn = \Theta(mnk). \quad (4.10)$$

Indeed, $mnk \leq T_{AB} \leq 2mnk$ for all positive dimensions. If k stays fixed, the cost is $\Theta(mn)$; a cubic count applies when all three dimensions are n .

For two $n \times n$ matrices, the classical algorithm uses n^3 multiplications and $n^2(n - 1)$ additions:

$$T_{\text{classical}}(n) = 2n^3 - n^2 \sim 2n^3. \quad (4.11)$$

This is a tight count for this algorithm, not a lower bound on all algorithms for matrix multiplication. In NumPy, $A @ B$ denotes matrix multiplication, whereas $A * B$ multiplies entries element by element.

Example 13 (Why the order of multiplication matters). Let A be 1000×10 , B be 10×1000 , and x have length 1000. In exact arithmetic, $(AB)x = A(Bx)$. Forming AB first costs 19,000,000 flops and produces one million entries. Multiplying that result by x costs another 1,999,000 flops, for a total of 20,999,000.

Computing Bx first costs 19,990 flops. Multiplying the resulting ten-entry vector by A costs 19,000 flops. The total is 38,990 flops, with only ten intermediate entries. The two floating-point evaluation orders can nevertheless produce slightly different answers because rounding does not preserve associativity.

Strassen's algorithm: fewer block multiplications

For even n , partition A and B into four square blocks:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}. \quad (4.12)$$

Each block has size $(n/2) \times (n/2)$. The usual block formula $C_{ij} = A_{i1}B_{1j} + A_{i2}B_{2j}$ requires eight block multiplications. Strassen's construction uses the following seven [8]:

$$\begin{aligned} M_1 &= (A_{11} + A_{22})(B_{11} + B_{22}), \\ M_2 &= (A_{21} + A_{22})B_{11}, \\ M_3 &= A_{11}(B_{12} - B_{22}), \\ M_4 &= A_{22}(B_{21} - B_{11}), \\ M_5 &= (A_{11} + A_{12})B_{22}, \\ M_6 &= (A_{21} - A_{11})(B_{11} + B_{12}), \\ M_7 &= (A_{12} - A_{22})(B_{21} + B_{22}). \end{aligned} \quad (4.13)$$

The output blocks are recovered by

$$\begin{aligned} C_{11} &= M_1 + M_4 - M_5 + M_7, & C_{12} &= M_3 + M_5, \\ C_{21} &= M_2 + M_4, & C_{22} &= M_1 - M_2 + M_3 + M_6. \end{aligned} \quad (4.14)$$

For example, expanding $M_3 + M_5$ cancels the terms involving $A_{11}B_{22}$ and gives $A_{11}B_{12} + A_{12}B_{22} = C_{12}$. The formulas preserve the order of factors, so they remain valid for matrix blocks even though matrix multiplication is not commutative.

These particular formulas use eighteen block additions or subtractions. Thus, if $n = 2^s$ and recursion continues to scalar multiplication, their real arithmetic count satisfies

$$T(n) = 7T(n/2) + 18(n/2)^2, \quad T(1) = 1. \quad (4.15)$$

At level j , there are 7^j subproblems, each of size $n/2^j$. Summing the work over the recursion tree gives

$$\begin{aligned} T(n) &= 7^s + \frac{18}{4}n^2 \sum_{j=0}^{s-1} \left(\frac{7}{4}\right)^j \\ &= 7n^{\log_2 7} - 6n^2 = \Theta\left(n^{\log_2 7}\right). \end{aligned} \quad (4.16)$$

The exponent $\log_2 7 \approx 2.8074$ is smaller than 3. The exact expression above applies to these formulas and this recursion convention; variants and practical stopping rules have different constants.

For an arbitrary positive n , pad both matrices with zeros to size $N = 2^{\lceil \log_2 n \rceil}$. Since $n \leq N < 2n$ and the desired product is the leading $n \times n$ block of the padded product, padding changes only a constant factor in the asymptotic bound.

Recent discoveries and their significance

The results below concern exact algebraic formulas and arithmetic complexity. An exact identity still incurs rounding error when evaluated in floating-point arithmetic. We distinguish *bilinear products* from total flops: a bilinear product multiplies a linear combination of entries of A by a linear combination of entries of B . Forming those combinations and combining the products requires additional operations.

Small formulas and the arithmetic in which they are valid. Two recursive levels of Strassen's algorithm multiply 4×4 matrices using $7^2 = 49$ bilinear products, compared with $4^3 = 64$ in the classical construction.

In 2022, AlphaTensor found a 4×4 formula using 47 products over the field $\mathbb{F}_2 = \{0, 1\}$, where addition and multiplication are performed modulo 2; for example, $1 + 1 = 0$. This is not a 47-product formula for ordinary real arithmetic [9].

In 2025, AlphaEvolve found a block-recursive formula using 48 products over the complex numbers [10]. The count improves the 49-product recursive Strassen construction. Historically, Kaporin had also reported a 48-product complex formula in 2024 [15]; it is therefore inappropriate to describe the AlphaEvolve result without qualification as the first such improvement since 1969.

Subsequently, Dumas, Pernet, and Sedoglavic obtained a 48-product formula with rational coefficients that supports block recursion and can be evaluated over the real numbers [13]. This matters for the present real-flop model: one need not implement complex arithmetic merely to use a 48-product construction. The AI systems participate in discovering formulas; the resulting algorithms can be executed without running an AI model.

Why a 4×4 result applies to general matrices. Partition each large matrix into a 4×4 grid of blocks and use a block-recursive 48-product formula. Each block product can itself be computed by the same procedure. If $n = 4^s$, the arithmetic recurrence has the form

$$T_{48}(n) = 48T_{48}(n/4) + \Theta(n^2), \quad T_{48}(n) = \Theta\left(n^{\log_4 48}\right). \quad (4.17)$$

Here $\log_4 48 \approx 2.7925$. For the specified recursions, the bilinear-product counts are:

n	Classical	Strassen	48-product recursion
4	64	49	48
16	4096	2401	2304
64	262144	117649	110592

The table omits additions and fixed-coefficient scalings. Padding to the next power of four handles arbitrary sizes with a bounded multiplicative overhead. Practical implementations can instead combine recursion with conventional multiplication and separate treatment of boundary blocks.

Reducing the number of products is only one part of algorithm design. A September 2026 preprint gives a real-applicable 48-product formula requiring 216 additional additions, subtractions, or constant scalings [14]. For its direct recursive evaluation, these counts give

$$T(n) = 48T(n/4) + 216(n/4)^2, \quad T(1) = 1, \quad (4.18)$$

and hence, for powers of four,

$$T(n) = \frac{31}{4}n^{\log_4 48} - \frac{27}{4}n^2. \quad (4.19)$$

The leading coefficient is $31/4 = 7.75$. Improving the additions and scalings can therefore improve the leading coefficient without changing either the 48-product count or its exponent. These are arithmetic statements, not universal runtime comparisons.

The matrix multiplication exponent. Fix a field $\mathbb{F}$ and count exact scalar arithmetic operations. For an algorithm $\mathcal{A}$ that multiplies every pair of $n \times n$ matrices over $\mathbb{F}$, let $T_{\mathcal{A}}(n)$ denote its worst-case arithmetic cost. Define

$$\omega_{\mathbb{F}} = \inf \{ \tau \geq 0 : \text{there exists } \mathcal{A} \text{ with } T_{\mathcal{A}}(n) = O(n^{\tau}) \}. \quad (4.20)$$

The field and arithmetic model must be specified. In the usual real/complex setting the exponents agree, since conversions between real and complex multiplication require only a constant-factor increase in arithmetic; we write this common exponent as ω . An infimum need not be attained. Its definition guarantees an $O(n^{\omega+\varepsilon})$ algorithm for each fixed $\varepsilon > 0$, not necessarily an $O(n^{\omega})$ algorithm.

Strassen gives the upper bound $\omega \leq \log_2 7$, and the real 48-product formulas give $\omega \leq \log_4 48$. Neither assertion claims that its right-hand side is the optimal exponent. An August 2026 preprint reports the stronger bound

$$\omega < 2.371177, \quad (4.21)$$

improving the preceding bound 2.371339 [11]. This result uses a more elaborate asymptotic construction and computer-assisted optimization with rigorous verification. It is not obtained merely by repeating the 48-product formula. An upper bound also does not establish a matching lower bound or determine the exact value of ω .

These discoveries show that classical cubic multiplication is not arithmetically optimal. Small block formulas provide reusable building blocks, while exponent bounds describe possibilities at asymptotically large sizes. Faster multiplication can also accelerate suitably reformulated algorithms for related linear-algebra problems, including matrix inversion under appropriate hypotheses [8]. It does not follow that the particular classical elimination algorithm studied here ceases to have cubic cost.

4.3 Deriving the cost of Gaussian elimination

We count the operations used to solve one system $A\mathbf{x} = \mathbf{b}$ by the procedures in Sections 2 and 3. Assume that A is a nonsingular $n \times n$ matrix and that every pivot used as a divisor is nonzero. The calculation consists of forward elimination, applied to both the coefficients and the right-hand side, followed by back substitution.

Counting conventions. Count each scalar addition, subtraction, multiplication, or division as one floating-point operation (flop). A multiplication followed by a subtraction counts as two flops, even if the hardware combines them into one instruction. We count absolute values, comparisons, index operations, and data movement separately. We use the full update ranges of the algorithm, without shortcuts for coefficients or multipliers that happen to be zero or ± 1 .

Each elimination multiplier is computed once and reused throughout its row update. In back substitution, the computed numerator is divided by the diagonal coefficient in one operation. These conventions specify the implementation whose exact cost we count; algebraically equivalent implementations can have different lower-order terms.

Forward elimination

At stage k , where $1 \leq k \leq n - 1$, there are $n - k$ rows below the pivot row. For each such row i , first compute and save

$$m_{ik} = \frac{a_{ik}}{a_{kk}}, \quad i = k + 1, \dots, n. \quad (4.22)$$

This requires one division per row. The multiplier must be saved before the entry a_{ik} is overwritten.

The updates displayed in Section 2 are then

$$\begin{aligned} a_{ij} &\leftarrow a_{ij} - m_{ik}a_{kj}, & j = k, \dots, n, \\ b_i &\leftarrow b_i - m_{ik}b_k. \end{aligned} \quad (4.23)$$

Notice that the coefficient range starts at $j = k$. Thus, in each row, the formula updates $n - k + 1$ coefficients, including the coefficient being eliminated, and one right-hand-side entry. Each update costs one multiplication and one subtraction.

Writing $q = n - k$, the cost at stage k is therefore

$$\begin{aligned} T_k &= \underbrace{q}_{\text{multiplier divisions}} + \underbrace{2q(q+1)}_{\text{coefficient updates}} + \underbrace{2q}_{\text{right-hand-side updates}} \\ &= 2q^2 + 5q. \end{aligned} \quad (4.24)$$

Summing over all $n - 1$ stages gives

$$\begin{aligned}
T_{\text{forward}}(n) &= \sum_{q=1}^{n-1} (2q^2 + 5q) \\
&= \frac{n(n-1)(2n-1)}{3} + \frac{5n(n-1)}{2} \\
&= \frac{2}{3}n^3 + \frac{3}{2}n^2 - \frac{13}{6}n.
\end{aligned} \tag{4.25}$$

The cubic term comes from updating the coefficients in the successively smaller submatrices. The multiplier divisions and right-hand-side updates together contribute only quadratic work.

Back substitution

After elimination, a_{ij} and b_i denote the updated entries. Starting with the last equation and working upward, we compute

$$x_i = \frac{b_i - \sum_{j=i+1}^n a_{ij}x_j}{a_{ii}}, \quad i = n, n-1, \dots, 1, \tag{4.26}$$

where the sum is zero when $i = n$. In row i , start an accumulator at b_i and subtract the $n - i$ products $a_{ij}x_j$. This requires $n - i$ multiplications and $n - i$ subtractions. One final division gives x_i . Consequently,

$$T_{\text{back}}(n) = \sum_{i=1}^n (2(n-i) + 1) = n(n-1) + n = n^2. \tag{4.27}$$

The notation $a_{ii}^{-1}(\dots)$ in Section 2 is evaluated here as a single division. Computing a reciprocal and then multiplying would instead use two operations for that final step.

Combining forward elimination and back substitution, the total arithmetic cost, before adding any pivot-selection arithmetic, is

$$\begin{aligned}
T_{\text{GE}}(n) &= T_{\text{forward}}(n) + T_{\text{back}}(n) \\
&= \frac{2}{3}n^3 + \frac{5}{2}n^2 - \frac{13}{6}n.
\end{aligned} \tag{4.28}$$

The additional cost of scaled partial pivoting

Section 3 chooses the pivot equations using the fixed scale vector s and the index vector ℓ . Changing the pivot order does not change the number of rows or entries updated at each stage. It therefore leaves the elimination and back-substitution counts above unchanged. We must, however, count the work needed to choose that order.

Computing the scale vector. For each original row, compute

$$s_i = \max_{1 \leq j \leq n} |a_{ij}|, \quad i = 1, \dots, n. \tag{4.29}$$

This uses n^2 absolute values and $n(n-1)$ comparisons, assuming each row maximum is initialized with its first candidate. The scale factors are computed only once. They remain fixed as elimination proceeds, and

the equations themselves are not divided by the scale factors. Thus, this initialization requires no additions, subtractions, multiplications, or divisions under our counting convention. Since A is nonsingular, each s_i is positive.

Selecting the pivots. At stage k , form the $n - k + 1$ candidate ratios

$$\frac{|a_{\ell_i, k}|}{s_{\ell_i}}, \quad i = k, \dots, n. \quad (4.30)$$

Computing each ratio once requires $n - k + 1$ divisions and the same number of absolute values. Finding their maximum requires $n - k$ comparisons. There are only $n - 1$ pivot searches: after these searches, just one equation remains.

The total additional division count is

$$R(n) = \sum_{k=1}^{n-1} (n - k + 1) = \frac{n(n+1)}{2} - 1. \quad (4.31)$$

The pivot searches also require

$$\sum_{k=1}^{n-1} (n - k) = \frac{n(n-1)}{2} \quad (4.32)$$

comparisons. Together with scale initialization, this gives $3n(n-1)/2$ comparisons and $n^2 + R(n)$ absolute values. At most $n - 1$ exchanges of two entries in ℓ are needed. As described in Section 3, the coefficient rows themselves are not moved.

Thus, with explicit ratio divisions, the total arithmetic cost of scaled partial pivoting, elimination, and back substitution is

$$\begin{aligned} T_{\text{GE(SPP)}}(n) &= T_{\text{GE}}(n) + R(n) \\ &= \frac{2}{3}n^3 + 3n^2 - \frac{5}{3}n - 1. \end{aligned} \quad (4.33)$$

Absolute values, comparisons, and index operations are additional to this flop count. This extra pivot-selection work is $O(n^2)$ under a model that assigns constant cost to each such operation.

A common implementation saving

The update $a_{ik} \leftarrow a_{ik} - m_{ik}a_{kk}$ produces zero in exact arithmetic. An implementation can assign $a_{ik} = 0$ directly and apply the arithmetic update only for $j = k + 1, \dots, n$. This saves one multiplication and one subtraction for every eliminated entry, or

$$2 \sum_{k=1}^{n-1} (n - k) = n(n-1) \quad (4.34)$$

flops overall. With this convention, the multiplier divisions and coefficient updates alone cost

$$T_{\text{coeff}}(n) = \sum_{q=1}^{n-1} (q + 2q^2) = \frac{2}{3}n^3 - \frac{1}{2}n^2 - \frac{1}{6}n. \quad (4.35)$$

The right-hand-side updates still cost $n(n-1)$ flops, and back substitution still costs n^2 flops. The corresponding total costs are therefore

$$\begin{aligned}\tilde{T}_{\text{GE}}(n) &= T_{\text{GE}}(n) - n(n-1) = \frac{2}{3}n^3 + \frac{3}{2}n^2 - \frac{7}{6}n, \\ \tilde{T}_{\text{GE(SPP)}}(n) &= T_{\text{GE(SPP)}}(n) - n(n-1) = \frac{2}{3}n^3 + 2n^2 - \frac{2}{3}n - 1.\end{aligned}\tag{4.36}$$

These different lower-order terms reflect whether the known zeros are assigned directly or computed using the displayed update rule.

Key Idea

For both procedures, including the right-hand side and back substitution, the leading arithmetic cost is

$$T(n) = \frac{2}{3}n^3 + O(n^2) = \Theta(n^3).\tag{4.37}$$

Forward elimination accounts for the cubic term. Back substitution and the extra work of scaled partial pivoting are quadratic. Consequently, doubling the number of equations approximately multiplies the arithmetic work by eight for sufficiently large n . Scaled partial pivoting improves the choice of pivots at a cost that does not change this leading term. A flop count measures arithmetic work, however; it does not by itself determine execution time or guarantee numerical accuracy.

A Linear algebra background

This appendix develops the algebraic facts used in the main text. All vectors and matrices are real unless stated otherwise, and vectors are columns. The aim is to make the dimensions, hypotheses, and reasoning visible, so that an algebraic identity can be used correctly in a numerical method. The results below include proofs, including the spectral theorem and the construction of the singular value decomposition. A first course in linear algebra provides further perspectives and examples [5, 6].

A.1 Vectors, matrices, products, and transposes

A vector $\mathbf{x} \in \mathbb{R}^n$ is an ordered list of n real numbers, with component x_j in position j . A matrix $A \in \mathbb{R}^{m \times n}$ has m rows and n columns; a_{ij} is its entry in row i , column j . Addition acts entry by entry and requires equal dimensions. Scalar multiplication multiplies every entry by the scalar. The zero vector and zero matrix have all entries zero; their shapes are determined by context.

The standard coordinate vector $\mathbf{e}_j \in \mathbb{R}^n$ has a 1 in position j and zeros elsewhere. The identity matrix $\mathbb{1}_n = (\mathbf{e}_1 \cdots \mathbf{e}_n)$ leaves every vector unchanged. For $A \in \mathbb{R}^{m \times n}$, both $\mathbb{1}_m A = A$ and $A \mathbb{1}_n = A$, with different identity sizes on the two sides when $m \neq n$.

What does $A\mathbf{x}$ mean?

If $A \in \mathbb{R}^{m \times n}$ and $\mathbf{x} \in \mathbb{R}^n$, then $A\mathbf{x} \in \mathbb{R}^m$ and

$$(A\mathbf{x})_i = \sum_{j=1}^n a_{ij}x_j, \quad i = 1, \dots, m.\tag{A.1}$$

Recap (continued)

Writing $A = (\mathbf{a}_1 \cdots \mathbf{a}_n)$ in terms of its columns gives

$$A\mathbf{x} = x_1\mathbf{a}_1 + \cdots + x_n\mathbf{a}_n. \quad (\text{A.2})$$

Thus solving $A\mathbf{x} = \mathbf{b}$ means finding coefficients that express $\mathbf{b}$ as a linear combination of the columns of A . The map takes inputs in $\mathbb{R}^n$ to outputs in $\mathbb{R}^m$, and is linear: $A(\alpha\mathbf{x} + \beta\mathbf{y}) = \alpha A\mathbf{x} + \beta A\mathbf{y}$.

If $B \in \mathbb{R}^{n \times p}$, then $AB \in \mathbb{R}^{m \times p}$ and

$$(AB)_{ij} = \sum_{k=1}^n a_{ik}b_{kj}, \quad 1 \leq i \leq m, \quad 1 \leq j \leq p. \quad (\text{A.3})$$

The inner dimensions must agree. The j th column of AB is A times the j th column of B . Consequently, $(AB)\mathbf{x} = A(B\mathbf{x})$: apply B first, then A . To verify this directly, expand the i th component as $\sum_{j=1}^p \sum_{k=1}^n a_{ik}b_{kj}x_j$ and reverse the two finite sums. The same calculation proves associativity of matrix multiplication. Distributivity follows by distributing each scalar product inside these sums. Multiplication generally does not commute, even when both orders exist.

Example 14 (A product as a combination of columns). Let

$$A = \begin{pmatrix} 1 & 2 & 0 \\ -1 & 1 & 3 \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix}. \quad (\text{A.4})$$

Computing row dot products or combining columns gives the same answer:

$$A\mathbf{x} = \begin{pmatrix} 2 - 2 + 0 \\ -2 - 1 + 3 \end{pmatrix} = 2 \begin{pmatrix} 1 \\ -1 \end{pmatrix} - \begin{pmatrix} 2 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}. \quad (\text{A.5})$$

For a separate square example, let $B = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ and $C = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$. Then

$$BC = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}, \quad CB = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}. \quad (\text{A.6})$$

Both products exist, but their entries differ. The order of elimination matrices and permutation matrices therefore matters.

The transpose $A^\top \in \mathbb{R}^{n \times m}$ interchanges rows and columns: $(A^\top)_{ji} = a_{ij}$. In particular,

$$(A^\top)^\top = A, \quad (A + B)^\top = A^\top + B^\top, \quad (AB)^\top = B^\top A^\top. \quad (\text{A.7})$$

The first two formulas follow entry by entry. For the last, the (i, j) entry of the left side is $\sum_{k=1}^n a_{jk}b_{ki}$; the same entry of the right side is $\sum_{k=1}^n b_{ki}a_{jk}$, which is equal because the entries are scalars. The reversal of matrix order is essential and makes the dimensions agree.

A.2 Inner products, lengths, and matrix norms

The Euclidean inner product of two vectors of length n is

$$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^\top \mathbf{y} = \sum_{j=1}^n x_j y_j. \quad (\text{A.8})$$

It is symmetric, linear in each argument, and satisfies $\mathbf{x}^\top \mathbf{x} \geq 0$ with equality only for $\mathbf{x} = 0$. Two vectors are *orthogonal* if their inner product is zero. Their Euclidean length is $\|\mathbf{x}\|_2 = \sqrt{\mathbf{x}^\top \mathbf{x}}$; a vector with length one is a unit vector. A family is *orthonormal* when its vectors are unit vectors and distinct members are orthogonal.

Proposition 5 (Cauchy–Schwarz and Pythagoras). For vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$,

$$|\mathbf{x}^\top \mathbf{y}| \leq \|\mathbf{x}\|_2 \|\mathbf{y}\|_2. \quad (\text{A.9})$$

If the vectors are orthogonal, then

$$\|\mathbf{x} + \mathbf{y}\|_2^2 = \|\mathbf{x}\|_2^2 + \|\mathbf{y}\|_2^2. \quad (\text{A.10})$$

Proof. If $\mathbf{y} = 0$, the first inequality holds with both sides zero. Otherwise set $c = (\mathbf{x}^\top \mathbf{y}) / (\mathbf{y}^\top \mathbf{y})$. Expanding a nonnegative squared length gives

$$0 \leq \|\mathbf{x} - c\mathbf{y}\|_2^2 = \|\mathbf{x}\|_2^2 - \frac{(\mathbf{x}^\top \mathbf{y})^2}{\|\mathbf{y}\|_2^2}. \quad (\text{A.11})$$

Multiply by the positive number $\|\mathbf{y}\|_2^2$ and take square roots. For the second claim, expand $\|\mathbf{x} + \mathbf{y}\|_2^2 = \|\mathbf{x}\|_2^2 + 2\mathbf{x}^\top \mathbf{y} + \|\mathbf{y}\|_2^2$; orthogonality removes the cross term. ■

A *norm* is a function measuring vector size. It is nonnegative and vanishes only at zero, satisfies $\|c\mathbf{x}\| = |c|\|\mathbf{x}\|$, and obeys the triangle inequality $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$. The norms used in this note are

$$\|\mathbf{x}\|_1 = \sum_{j=1}^n |x_j|, \quad \|\mathbf{x}\|_2 = \left(\sum_{j=1}^n |x_j|^2 \right)^{1/2}, \quad \|\mathbf{x}\|_\infty = \max_{1 \leq j \leq n} |x_j|. \quad (\text{A.12})$$

Nonnegativity and homogeneity follow directly from the definitions. The triangle inequality for the 1- and infinity norms follows by applying $|x_j + y_j| \leq |x_j| + |y_j|$ in each coordinate. For the 2-norm, Cauchy–Schwarz bounds the cross term in the preceding expansion by $2\|\mathbf{x}\|_2 \|\mathbf{y}\|_2$, yielding $\|\mathbf{x} + \mathbf{y}\|_2^2 \leq (\|\mathbf{x}\|_2 + \|\mathbf{y}\|_2)^2$.

Definition 6 (Induced matrix norms). For $p \in \{1, 2, \infty\}$ and $A \in \mathbb{R}^{m \times n}$, define

$$\|A\|_p = \max_{\mathbf{x} \neq 0} \frac{\|A\mathbf{x}\|_p}{\|\mathbf{x}\|_p} = \max_{\|\mathbf{x}\|_p=1} \|A\mathbf{x}\|_p. \quad (\text{A.13})$$

This is the largest amplification of a vector by A . The input norm is on $\mathbb{R}^n$, and the output norm of the same type is on $\mathbb{R}^m$. The Frobenius norm is a different matrix norm:

$$\|A\|_F = \left(\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2 \right)^{1/2}. \quad (\text{A.14})$$

It treats the entries as one long vector.

The two maxima in the induced-norm definition agree because rescaling a nonzero vector to unit length does not change the ratio. They are attained: a continuous function on the finite-dimensional compact unit sphere attains its maximum. We use this familiar extreme-value fact again in the spectral theorem proof below.

Proposition 7 (Entrywise formulas and norm inequalities). For a real $m \times n$ matrix,

$$\|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^m |a_{ij}|, \quad \|A\|_\infty = \max_{1 \leq i \leq m} \sum_{j=1}^n |a_{ij}|. \quad (\text{A.15})$$

For conformable matrices and vectors and $p \in \{1, 2, \infty\}$,

$$\|A\mathbf{x}\|_p \leq \|A\|_p \|\mathbf{x}\|_p, \quad \|AB\|_p \leq \|A\|_p \|B\|_p. \quad (\text{A.16})$$

The induced norms are matrix norms, and $\|AB\|_F \leq \|A\|_F \|B\|_F$.

Proof. For the 1-norm, sum the componentwise triangle inequalities:

$$\|A\mathbf{x}\|_1 \leq \sum_{i=1}^m \sum_{j=1}^n |a_{ij}| |x_j| = \sum_{j=1}^n \left(\sum_{i=1}^m |a_{ij}| \right) |x_j| \leq \left(\max_j \sum_{i=1}^m |a_{ij}| \right) \|\mathbf{x}\|_1. \quad (\text{A.17})$$

This proves an upper bound on $\|A\|_1$. Choose $\mathbf{x} = \mathbf{e}_{j_*}$, where column j_* has maximal absolute column sum. It has unit 1-norm, and $A\mathbf{x}$ is that column, so the upper bound is attained.

For the infinity norm, each row satisfies $|(A\mathbf{x})_i| \leq (\sum_{j=1}^n |a_{ij}|) \|\mathbf{x}\|_\infty$. Choose a row i_* of maximal absolute row sum and take $x_j = 1$ when $a_{i_*j} \geq 0$ and $x_j = -1$ when $a_{i_*j} < 0$. Then $\|\mathbf{x}\|_\infty = 1$ and $(A\mathbf{x})_{i_*} = \sum_{j=1}^n |a_{i_*j}|$, proving equality.

The bound on $A\mathbf{x}$ is the definition when $\mathbf{x} \neq 0$ and is trivial when $\mathbf{x} = 0$. Applying it twice gives $\|AB\mathbf{x}\|_p \leq \|A\|_p \|B\|_p \|\mathbf{x}\|_p$; maximize over unit vectors to obtain submultiplicativity. Positivity and homogeneity of the induced norm follow from those of the vector norm. If $A \neq 0$, some column $A\mathbf{e}_j$ is nonzero, so $\|A\|_p > 0$. Finally, $\|(A+B)\mathbf{x}\|_p \leq (\|A\|_p + \|B\|_p) \|\mathbf{x}\|_p$ proves its triangle inequality.

For the Frobenius product inequality, let $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times \ell}$. Apply Cauchy–Schwarz to each row–column dot product:

$$\sum_{i=1}^m \sum_{j=1}^{\ell} |(AB)_{ij}|^2 \leq \sum_{i=1}^m \sum_{j=1}^{\ell} \left(\sum_{k=1}^n |a_{ik}|^2 \right) \left(\sum_{k=1}^n |b_{kj}|^2 \right) = \|A\|_F^2 \|B\|_F^2. \quad (\text{A.18})$$

Taking square roots proves the claim. ■

Every induced norm of $\mathbb{1}_n$ is 1, while $\|\mathbb{1}_n\|_F = \sqrt{n}$. Thus for $n > 1$ the Frobenius norm is not induced by any single vector norm on $\mathbb{R}^n$. It remains a useful matrix norm. The SVD will also give $\|A\|_2 \leq \|A\|_F \leq \sqrt{r} \|A\|_2$ for rank r . Different norms measure different aspects of an error, so the norm must be specified when reporting accuracy or a condition number.

A.3 Span, independence, bases, and dimension

A *linear combination* of vectors $\mathbf{v}_1, \dots, \mathbf{v}_k$ is $\sum_{j=1}^k c_j \mathbf{v}_j$ with real coefficients. Their span is

$$\text{span}\{\mathbf{v}_1, \dots, \mathbf{v}_k\} = \left\{ \sum_{j=1}^k c_j \mathbf{v}_j : c_1, \dots, c_k \in \mathbb{R} \right\}. \quad (\text{A.19})$$

A *subspace* $S \subseteq \mathbb{R}^n$ contains zero and is closed under addition and scalar multiplication, equivalently under linear combinations. For example, $\text{span}\{(1, 2)^\top\}$ is a line through the origin.

The family is *linearly independent* if $\sum_{j=1}^k c_j \mathbf{v}_j = 0$ forces all $c_j = 0$. If a nontrivial combination equals zero, choose a nonzero coefficient and solve for its vector: that vector is a combination of the others and is redundant in forming the span. For example, $(1, 2)^\top$ and $(2, 4)^\top$ are dependent because the second is twice the first.

A *basis* of S is an independent family spanning S . Coordinates in a basis exist by spanning and are unique: subtracting two coordinate representations gives a zero combination, so independence forces the coordinates to agree. The vectors $\mathbf{e}_1, \dots, \mathbf{e}_n$ are a basis of $\mathbb{R}^n$ because $\mathbf{x} = \sum_{j=1}^n x_j \mathbf{e}_j$ uniquely.

Lemma 8 (Finite-dimensional basis facts). An independent family in a subspace can be extended to a basis of that subspace. Any two bases of a subspace have the same number of vectors. This number is its *dimension*; the zero subspace has dimension zero.

Proof. First suppose $\mathbf{v}_1, \dots, \mathbf{v}_k$ are independent and $\mathbf{w}_1, \dots, \mathbf{w}_\ell$ span a subspace containing them. We show $k \leq \ell$ by successive replacement. Express $\mathbf{v}_1$ using the $\mathbf{w}$'s. At least one coefficient is nonzero, since $\mathbf{v}_1 \neq 0$; solving for that $\mathbf{w}$ lets us replace it by $\mathbf{v}_1$ without changing the span. Suppose i replacements have produced a spanning family consisting of $\mathbf{v}_1, \dots, \mathbf{v}_i$ and $\ell - i$ remaining $\mathbf{w}$'s. If $i < k$, the expression for $\mathbf{v}_{i+1}$ in this family must use a remaining $\mathbf{w}$ with nonzero coefficient. Otherwise it would lie in the span of the first i independent vectors, a contradiction. Solve for that $\mathbf{w}$ and replace it. This cannot continue after all ℓ positions are filled, so $k \leq \ell$.

Starting with an independent family in S , add a vector of S outside its span whenever one exists. Each addition preserves independence: a dependence with a nonzero coefficient of the new vector would express it using the previous ones. There can be at most n independent vectors in $\mathbb{R}^n$ by the preceding argument with its standard basis. Thus this process terminates with an independent spanning family of S , a basis. Finally, comparing two bases, each as an independent family against the other as a spanning family, gives both inequalities between their sizes. The sizes are equal. ■

A.4 Elementary, permutation, and triangular matrices

An *inverse* of a square matrix E is a matrix F of the same shape with $EF = FE = \mathbb{1}$; the notation is $F = E^{-1}$. An *elementary matrix* is obtained by applying one elementary row operation to an identity matrix. Multiplying A on the left by it performs that row operation on A .

Proposition 9 (Elementary row operations are invertible). Swapping two rows, multiplying a row by a nonzero scalar, and adding a multiple of one row to a different row are each represented by an invertible matrix. Therefore, for any sequence of such operations represented by E ,

$$A\mathbf{x} = \mathbf{b} \iff E A \mathbf{x} = E \mathbf{b}. \quad (\text{A.20})$$

Proof. Work with m rows. Swapping rows i, k is represented by the identity with these rows interchanged; applying the swap twice restores every row, so this matrix is its own inverse. Multiplying row i by $c \neq 0$ is represented by the diagonal matrix with diagonal entry c in position i and ones elsewhere. Replacing c by $1/c$ gives its inverse.

For $i \neq k$, the update $R_i \leftarrow R_i - \ell R_k$ is represented by

$$E = \mathbb{1}_m - \ell \mathbf{e}_i \mathbf{e}_k^\top. \quad (\text{A.21})$$

Indeed, $\mathbf{e}_k^\top A$ extracts row k , and left multiplication by $\mathbf{e}_i$ places that row in position i with zeros elsewhere. Since $\mathbf{e}_k^\top \mathbf{e}_i = 0$, the matrix $N = \mathbf{e}_i \mathbf{e}_k^\top$ satisfies $N^2 = 0$. Hence

$$(\mathbb{1}_m - \ell N)(\mathbb{1}_m + \ell N) = \mathbb{1}_m - \ell^2 N^2 = \mathbb{1}_m, \quad (\text{A.22})$$

and the reversed product is also $\mathbb{1}_m$. Thus $E^{-1} = \mathbb{1}_m + \ell N$.

A product $E_s \cdots E_1$ of elementary matrices is inverted by undoing the operations in reverse order, using $E_1^{-1} \cdots E_s^{-1}$. Multiplying an equation by E gives the forward implication; multiplying the transformed equation by E^{-1} gives the reverse implication. ■

This proves why row operations must affect the right-hand side as well as the coefficient matrix. It also shows why the sign stored as a multiplier in an L factor is positive ℓ when the elimination operation subtracts ℓR_k : the inverse elementary matrix contains $+\ell$.

A *permutation matrix* has exactly one 1 in each row and column and zeros elsewhere. Left multiplication reorders rows; right multiplication reorders columns. Its columns are distinct standard coordinate vectors, so $P^\top P = \mathbb{1}$ and $PP^\top = \mathbb{1}$, giving $P^{-1} = P^\top$. A product of row swaps is a permutation matrix, with the rightmost swap applied first. For example,

$$P = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad (\text{A.23})$$

exchanges two rows. Although this example has $P = P^\top$, a general permutation matrix need not equal its transpose. Applying P to both A and $\mathbf{b}$ preserves the matching of coefficients and right-hand sides.

A square matrix U is *upper triangular* if $u_{ij} = 0$ for $i > j$; L is *lower triangular* if $l_{ij} = 0$ for $i < j$. A triangular matrix is *unit triangular* if every diagonal entry is 1. A diagonal matrix has zeros whenever $i \neq j$.

A.5 Column space, nullspace, and rank

Four spaces describing the same linear map

For $A \in \mathbb{R}^{m \times n}$, define

$$\begin{aligned} \text{Col}(A) &= \{A\mathbf{x} : \mathbf{x} \in \mathbb{R}^n\} \subseteq \mathbb{R}^m, \\ \text{Null}(A) &= \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{0}\} \subseteq \mathbb{R}^n, \\ \text{Row}(A) &= \text{Col}(A^\top) \subseteq \mathbb{R}^n, \\ \text{Null}(A^\top) &= \{\mathbf{y} \in \mathbb{R}^m : A^\top \mathbf{y} = \mathbf{0}\} \subseteq \mathbb{R}^m. \end{aligned}$$

The alternative notations $\text{col}(A)$ and $\text{ker}(A)$ mean $\text{Col}(A)$ and $\text{Null}(A)$, respectively. The column space lists attainable outputs; the nullspace lists input directions erased by A . The row space is the span of the rows regarded as vectors in the input space. The rank is $\text{rank}(A) = \dim \text{Col}(A)$.

Each displayed set is a subspace. For the column space, a combination of $A\mathbf{x}$ and $A\mathbf{y}$ is $A(\alpha\mathbf{x} + \beta\mathbf{y})$. For the nullspace, if $A\mathbf{x} = A\mathbf{y} = 0$, then $A(\alpha\mathbf{x} + \beta\mathbf{y}) = 0$. The other two cases follow by replacing A with $A^\top$.

Theorem 2 (Rank, pivots, and rank-nullity). For $A \in \mathbb{R}^{m \times n}$ with rank r , the row-space dimension equals r , and any row echelon form has exactly r pivots. In particular, $0 \leq r \leq \min(m, n)$ and

$$\dim \text{Null}(A) = n - r, \quad \dim \text{Null}(A^\top) = m - r. \quad (\text{A.24})$$

Proof. First row-reduce A to an echelon matrix $R = EA$, where E is invertible by Proposition 9. A linear relation among columns of R has the form $E\mathbf{A}\mathbf{x} = 0$, equivalent to $A\mathbf{x} = 0$. Thus a set of corresponding columns is independent for R exactly when it is independent for A , and the column ranks agree. Every row of R is a combination of rows of A ; using $A = E^{-1}R$ proves the converse inclusion of row spaces. Thus the row spaces themselves agree.

Suppose R has s nonzero rows, with pivot columns $j_1 < \dots < j_s$. These rows are independent: in a zero combination, column j_1 first forces the coefficient of row 1 to vanish, column j_2 then forces the coefficient of row 2 to vanish, and so on. They also span its row space, because all other rows are zero. Therefore the row dimension is s . Its s pivot columns are independent: restricted to the first s rows they form an upper triangular $s \times s$ matrix with nonzero diagonal, and back substitution in a zero combination forces all coefficients to vanish. No more than s columns can be independent, since all columns have zero entries below row s and lie in an s -dimensional coordinate subspace. Consequently the column rank is also s , establishing $s = r$ and the equality of row and column ranks.

To prove rank-nullity without merely counting variables, choose a basis $\mathbf{z}_1, \dots, \mathbf{z}_d$ of $\text{Null}(A)$ and extend it to a basis $\mathbf{z}_1, \dots, \mathbf{z}_d, \mathbf{w}_1, \dots, \mathbf{w}_{n-d}$ of $\mathbb{R}^n$, using Lemma 8. The vectors $A\mathbf{w}_1, \dots, A\mathbf{w}_{n-d}$ span $\text{Col}(A)$ because applying A to any basis expansion kills all the $\mathbf{z}$ terms. They are independent: if $\sum_{j=1}^{n-d} c_j A\mathbf{w}_j = 0$, then $\sum_{j=1}^{n-d} c_j \mathbf{w}_j$ lies in the nullspace and is a combination of the $\mathbf{z}$'s. Independence of the complete basis forces every $c_j = 0$. They therefore form a column-space basis, so $r = n - d$. Applying the same argument to $A^\top$, whose rank is also r by the row-column equality just proved, gives its nullity $m - r$. ■

Full column rank means $r = n$; full row rank means $r = m$. For rectangular matrices these are different conditions. In the homogeneous system $A\mathbf{x} = 0$, echelon form gives another concrete reading of rank-nullity: there are $n - r$ free variables. Set one free variable to 1 and the rest to 0, solve for the pivot variables, and repeat. The resulting vectors span the nullspace, and their distinct free-coordinate patterns make them independent.

A.6 Consistency and all exact solutions

Theorem 3 (Existence and uniqueness for a linear system). Let $A \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$, and $r = \text{rank}(A)$. The following conditions are equivalent:

1. The system $A\mathbf{x} = \mathbf{b}$ has an exact solution.
2. $\mathbf{b} \in \text{Col}(A)$.
3. $\text{rank}((A \mid \mathbf{b})) = \text{rank}(A)$.

If $\mathbf{x}_0$ is one solution, the complete solution set is

$$\{\mathbf{x}_0 + \mathbf{z} : \mathbf{z} \in \text{Null}(A)\}. \quad (\text{A.25})$$

A consistent system has a unique solution exactly when $r = n$. Otherwise it has infinitely many solutions parametrized by $n - r$ free real coefficients.

Proof. By (A.2), a solution is exactly a choice of coefficients expressing $\mathbf{b}$ in the span of the columns of A . This proves the first equivalence. If $\mathbf{b}$ belongs to that span, appending it as a column does not change the span and hence does not change its dimension. Conversely, if $\mathbf{b}$ is outside the column space, choose a basis $\mathbf{c}_1, \dots, \mathbf{c}_r$ of that space. The extended family $\mathbf{c}_1, \dots, \mathbf{c}_r, \mathbf{b}$ is independent: a zero combination with a nonzero coefficient of $\mathbf{b}$ would put $\mathbf{b}$ back in the column space, while one with zero coefficient of $\mathbf{b}$ forces all remaining coefficients to vanish by independence. This family spans the augmented column space, so its rank is $r + 1$. This proves the third equivalence and explains why a single appended column can increase rank by at most one.

If $A\mathbf{x}_0 = \mathbf{b}$ and $A\mathbf{z} = 0$, then $A(\mathbf{x}_0 + \mathbf{z}) = \mathbf{b}$, proving that every displayed vector is a solution. Conversely, if $A\mathbf{x} = \mathbf{b}$, subtraction gives $A(\mathbf{x} - \mathbf{x}_0) = 0$, so every solution has the displayed form. Uniqueness therefore holds exactly when the nullspace is $\{0\}$, equivalently when $r = n$ by rank–nullity. If it has basis $\mathbf{z}_1, \dots, \mathbf{z}_{n-r}$, every solution is uniquely written $\mathbf{x}_0 + \sum_{j=1}^{n-r} t_j \mathbf{z}_j$. For $n - r > 0$, changing even one coefficient through all real values gives infinitely many solutions. ■

A nonempty solution set is an *affine set*, a translate of a subspace. It need not contain zero. A homogeneous system $A\mathbf{x} = 0$ is always consistent and has the nullspace itself as its solution set. These are rank statements, not just dimension comparisons: an overdetermined system can be consistent and unique, while an underdetermined system can be inconsistent. If an underdetermined system is consistent, $r \leq m < n$ prevents uniqueness.

For a two-variable example,

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix}, \quad \text{Col}(A) = \text{span} \left\{ \begin{pmatrix} 1 \\ 2 \end{pmatrix} \right\}, \quad \text{Null}(A) = \text{span} \left\{ \begin{pmatrix} -2 \\ 1 \end{pmatrix} \right\}. \quad (\text{A.26})$$

Its rank and nullity are both 1. Consistency requires $b_2 = 2b_1$; when this holds, all solutions are $(b_1 - 2t, t)^\top$ for $t \in \mathbb{R}$.

Example 15 (The same matrix, with consistent and inconsistent data). Consider

$$A = \begin{pmatrix} 1 & 2 & 0 \\ 0 & 0 & 1 \\ 1 & 2 & 1 \end{pmatrix}. \quad (\text{A.27})$$

Its third row is the sum of the first two rows, so consistency requires $b_3 = b_1 + b_2$. Its first and third columns are independent, and its second column is twice its first, so $\text{rank}(A) = 2$. For $\mathbf{b} = (3, 1, 4)^\top$, the

equations reduce to $x_1 + 2x_2 = 3$ and $x_3 = 1$. The solutions are

$$\mathbf{x} = \begin{pmatrix} 3 \\ 0 \\ 1 \end{pmatrix} + t \begin{pmatrix} -2 \\ 1 \\ 0 \end{pmatrix}, \quad t \in \mathbb{R}. \quad (\text{A.28})$$

The second vector spans the null space. For $\mathbf{b} = (3, 1, 5)^\top$, elimination instead produces $0 = 1$, so no exact solution exists. The vector $\mathbf{y} = (-1, -1, 1)^\top$ gives another certificate: $A^\top \mathbf{y} = 0$ but $\mathbf{y}^\top \mathbf{b} = 1$. If an exact solution existed, then $\mathbf{y}^\top \mathbf{b} = \mathbf{y}^\top A \mathbf{x} = 0$, a contradiction.

A.7 Dense matrices, sparse matrices, and storage

Definition 10 (Nonzero count and density). For $A = (a_{ij}) \in \mathbb{R}^{m \times n}$, define

$$\text{nnz}(A) = \#\{(i, j) : 1 \leq i \leq m, 1 \leq j \leq n, a_{ij} \neq 0\}, \quad \rho(A) = \frac{\text{nnz}(A)}{mn}. \quad (\text{A.29})$$

Here $\#$ denotes the number of elements of a finite set. The number $\rho(A) \in [0, 1]$ is the *density* of A .

A *dense storage format* reserves a scalar location for every one of the mn entries, including any zeros. A *dense algorithm* generally processes all these positions without exploiting a zero pattern. Consequently, a matrix with many zeros may still incur a dense algorithm's full cost.

A *sparse storage format* records nonzero values and information about their locations, allowing suitable algorithms to avoid operations on omitted zeros. For example, compressed sparse row storage records values, column indices, and row pointers, requiring $O(\text{nnz}(A) + m)$ storage locations. In an actual data structure, explicitly stored zeros may make the number of stored entries larger than the mathematical quantity $\text{nnz}(A)$.

The adjective *sparse* usually means that sufficiently few entries are nonzero for such a representation to be useful; there is no universal density threshold. For a precise asymptotic statement, we call a family $A_n \in \mathbb{R}^{n \times n}$ *asymptotically sparse* when

$$\text{nnz}(A_n) = o(n^2), \quad \text{equivalently} \quad \rho(A_n) \longrightarrow 0. \quad (\text{A.30})$$

A family with $\text{nnz}(A_n) = \Theta(n^2)$ has a nonzero count of dense order, even if its density is a small fixed percentage. These asymptotic statements and the practical choice of storage are different distinctions.

Example 16 (A tridiagonal family). A matrix is *tridiagonal* when $a_{ij} = 0$ for $|i - j| > 1$. If every entry on its three permitted diagonals is nonzero, then

$$\text{nnz}(A_n) = n + 2(n - 1) = 3n - 2, \quad \rho(A_n) = \frac{3n - 2}{n^2} \longrightarrow 0. \quad (\text{A.31})$$

Storing just these diagonals requires $\Theta(n)$ scalars, while storing the same matrix in a dense array requires $\Theta(n^2)$ scalars. An appropriate tridiagonal solver can also use $\Theta(n)$ arithmetic; its pivot conditions or pivoting strategy must still be respected.

Sparsity here concerns exact zeros in the represented problem. Discarding a small nonzero coefficient changes that problem and requires a separate approximation-error analysis.

A.8 A reference table of common operations and their costs

The following table collects the classical dense costs. Here A, B have the displayed dimensions, q counts right-hand sides, and the QR row assumes $m \geq n$. Details of inversion, QR, and the SVD are developed later. A transpose can be represented as a view in some array libraries, but physically copying the transposed entries requires moving mn numbers.

Operation	Arithmetic or work	Interpretation
Vector addition, length n	n flops	Entrywise operation
Vector dot product, length n	$2n - 1$ flops	$\Theta(n)$
Vector update $\mathbf{y} \leftarrow \alpha \mathbf{x} + \mathbf{y}$	$2n$ flops	Scale and add
Outer product, lengths m, n	mn flops	$m \times n$ output
Add or scale an $m \times n$ matrix	mn flops	Entrywise operation
Copy or explicitly transpose an $m \times n$ matrix	$\Theta(mn)$ data moves	No scalar arithmetic required
$A\mathbf{x}, A \in \mathbb{R}^{m \times n}$	$2mn - m$ flops	$\Theta(mn)$
$AB, A : m \times k, B : k \times n$	$2mnk - mn$ flops	Classical product
Triangular solve, order n	n^2 flops	$n^2 - n$ if unit diagonal
LU factorization, order n	$\frac{2}{3}n^3 + O(n^2)$	Pivot search adds quadratic work
Solve with LU already available	$2n^2 - n$ flops	Per right-hand side
Factor and solve q right-hand sides	$\frac{2}{3}n^3 + 2qn^2 + O(n^2 + qn)$	Reuse one factorization
Cholesky factorization, order n	$\frac{1}{3}n^3 + O(n^2)$	Symmetric positive definite input; n square roots in addition
Inverse via LU and n general solves	$\frac{8}{3}n^3 + O(n^2)$	This specific unoptimized method
Determinant from available LU	$O(n)$ additional arithmetic	Product of pivots and permutation sign
Householder QR, $m \geq n$	$2mn^2 - \frac{2}{3}n^3 + O(mn)$	Q stored implicitly
Economy SVD, $m \times n$	$O(mn \min(m, n))$	Standard dense fixed-precision algorithms
Explicit pseudoinverse from economy SVD	$O(mn \min(m, n))$ overall	Includes factorization and assembly
Matrix 1-, ∞ -, or Frobenius norm	$\Theta(mn)$ work	Frobenius norm includes a square root
Matrix spectral norm by economy SVD	$O(mn \min(m, n))$	This method; estimates can be cheaper

For an $m \times n$ matrix of full column rank with $m \geq n$, an unblocked LU-type reduction of the coefficient matrix has leading work $mn^2 - n^3/3 + O(mn + n^2)$: sum one division and two arithmetic operations per updated entry over shrinking rectangular submatrices. This describes the elimination phase; rectangular least squares is a different optimization problem and is ordinarily handled by QR or SVD, not by treating a remaining inconsistent row as an exact equation.

More explicitly, for this full-column-rank tall case, stage k has $m - k$ multipliers and $(m - k)(n - k)$ trailing entries. Including the last column's multipliers, the coefficient work is

$$\sum_{k=1}^n ((m - k) + 2(m - k)(n - k)) = mn^2 - \frac{n^3}{3} - \frac{n^2}{2} - \frac{n}{6}. \quad (\text{A.32})$$

The $k = n$ term vanishes in the square case and counts only the remaining last-column eliminations in a genuinely tall matrix. Setting $m = n$ recovers (4.35). A wide matrix requires a different choice of summation limit; the displayed formula assumes $m \geq n$ throughout.

Cholesky saves work because symmetry allows one triangular half of each trailing submatrix to be updated. With q rows remaining, there are $q(q + 1)/2$ stored entries to update, each with two flops, in addition to q divisions and a square root for the pivot. The leading sum $\sum_{q=1}^{n-1} q^2$ is $\frac{1}{3}n^3 + O(n^2)$; the additional divisions and the n square roots do not change that leading term.

The determinant row assumes the factors already exist: $\det(A) = \det(P) \prod_{i=1}^n u_{ii}$ when $PA = LU$. Computing a determinant from a new dense matrix still requires a cubic factorization by this approach. Products of many pivots can overflow or underflow, so software often returns the sign and logarithm of the absolute determinant. A determinant is also a poor numerical test of closeness to singularity: scaling A changes $\det(A)$ by a power without necessarily changing its condition number.