What LU computes, and at what cost

The factorization costs n once. Everything below reuses it, and the cost column is the reason the decomposition exists at all — each task afterwards is cheaper than redoing the elimination.

4
TASKS

THE FACTORIZATION ITSELF

1

1 Compute PA = LU

once per matrix

n flops

Gaussian elimination with the multipliers kept rather than discarded — L holds what was subtracted, U holds what remained, P records the row swaps. The elimination was being done anyway; LU is the observation that its byproducts are worth storing.

REUSING IT — COST PER USE

3

2 Solve a system

forward solve $L\mathbf{y} = P\mathbf{b}$, then back solve $U\mathbf{x} = \mathbf{y}$

$2n$ per right-hand side

Two triangular solves, each n. The saving arrives on the second right-hand side: a fresh elimination would cost n again, while reusing the factors costs n — which is why LU is what a solver stores when the matrix is fixed and $\mathbf{b}$ varies.

3 Determinant

s = number of row swaps in P

$\det A = (-1)^{s} \prod u$

Multiply the diagonal of U and correct the sign for the swaps — $O(n)$ once the factorization exists. Cofactor expansion is $O(n!)$ for the same number, which is the gap between a definition and a method.

4 Inverse

solve $A\mathbf{x} = \mathbf{e}_i$ for each i

$2n$ total

n solves at $2n$ each. Worth doing only when the inverse is genuinely wanted as an object — if the goal is solving systems, the triangular solves above are three times cheaper and numerically better behaved.

The economics are the whole argument. One elimination at n, then n per right-hand side — so ten systems sharing a matrix cost barely more than one. Compare recomputing from scratch each time, and the saving is a factor of n; compare **inverting**, and LU is three times cheaper and more accurate besides.