

# CS 4220: Numerical Analysis

## Gaussian elimination

David Bindel

2026-02-06

### Introduction

For the next few lectures, we will build tools to solve linear systems. Our main tool will be the factorization  $PA = LU$ , where  $P$  is a permutation,  $L$  is a unit lower triangular matrix, and  $U$  is an upper triangular matrix. As we will see, the Gaussian elimination algorithm learned in a first linear algebra class implicitly computes this decomposition; but by thinking about the decomposition explicitly, we find other ways to organize the computation.

### Triangular solves

Suppose that we have computed a factorization  $PA = LU$ . How can we use this to solve a linear system of the form  $Ax = b$ ? Permuting the rows of  $A$  and  $b$ , we have

$$PAx = LUx = Pb,$$

and therefore

$$x = U^{-1}L^{-1}Pb.$$

So we can reduce the problem of finding  $x$  to two simpler problems:

1. Solve  $Ly = Pb$
2. Solve  $Ux = y$

We assume the matrix  $L$  is unit lower triangular (diagonal of all ones + lower triangular), and  $U$  is upper triangular, so we can solve linear systems with  $L$  and  $U$  involving forward and backward substitution.

As a concrete example, suppose

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 2 & 1 \end{bmatrix}, \quad d = \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix}$$

To solve a linear system of the form  $Ly = d$ , we process each row in turn to find the value of the corresponding entry of  $y$ :

1. Row 1:  $y_1 = d_1$
2. Row 2:  $2y_1 + y_2 = d_2$ , or  $y_2 = d_2 - 2y_1$
3. Row 3:  $3y_1 + 2y_2 + y_3 = d_3$ , or  $y_3 = d_3 - 3y_1 - 2y_2$

More generally, the *forward substitution* algorithm for solving unit lower triangular linear systems  $Ly = d$  looks like

```
function forward_subst_unit(L, d)
    y = copy(d)
    n = length(d)
    for i = 2:n
        y[i] = d[i] - L[i,1:i-1]'*y[1:i-1]
    end
    y
end
```

Similarly, there is a *backward substitution* algorithm for solving upper triangular linear systems  $Ux = d$

```
function backward_subst(U, d)
    x = copy(d)
    n = length(d)
    for i = n:-1:1
        x[i] = (d[i] - U[i,i+1:n]'*x[i+1:n])/U[i,i]
    end
    x
end
```

Each of these algorithms takes  $O(n^2)$  time.

## Gaussian elimination by example

Let's start our discussion of  $LU$  factorization by working through these ideas with a concrete example:

$$A = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 10 \end{bmatrix}.$$

To eliminate the subdiagonal entries  $a_{21}$  and  $a_{31}$ , we subtract twice the first row from the second row, and thrice the first row from the third row:

$$A^{(1)} = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 10 \end{bmatrix} - \begin{bmatrix} 0 \cdot 1 & 0 \cdot 4 & 0 \cdot 7 \\ 2 \cdot 1 & 2 \cdot 4 & 2 \cdot 7 \\ 3 \cdot 1 & 3 \cdot 4 & 3 \cdot 7 \end{bmatrix} = \begin{bmatrix} 1 & 4 & 7 \\ 0 & -3 & -6 \\ 0 & -6 & -11 \end{bmatrix}.$$

That is, the step comes from a rank-1 update to the matrix:

$$A^{(1)} = A - \begin{bmatrix} 0 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 4 & 7 \end{bmatrix}.$$

Another way to think of this step is as a linear transformation  $A^{(1)} = M_1 A$ , where the rows of  $M_1$  describe the multiples of rows of the original matrix that go into rows of the updated matrix:

$$M_1 = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -3 & 0 & 1 \end{bmatrix} = I - \begin{bmatrix} 0 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} = I - \tau_1 e_1^T.$$

Similarly, in the second step of the algorithm, we subtract twice the second row from the third row:

$$\begin{bmatrix} 1 & 4 & 7 \\ 0 & -3 & -6 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 4 & 7 \\ 0 & -3 & -6 \\ 0 & -6 & -11 \end{bmatrix} = \left( I - \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \right) A^{(1)}.$$

More compactly:  $U = (I - \tau_2 e_2^T) A^{(1)}$ .

Putting everything together, we have computed

$$U = (I - \tau_2 e_2^T)(I - \tau_1 e_1^T)A.$$

Therefore,

$$A = (I - \tau_1 e_1^T)^{-1} (I - \tau_2 e_2^T)^{-1} U = LU.$$

Now, note that

$$(I - \tau_1 e_1^T)(I + \tau_1 e_1^T) = I - \tau_1 e_1^T + \tau_1 e_1^T - \tau_1 e_1^T \tau_1 e_1^T = I,$$

since  $e_1^T \tau_1$  (the first entry of  $\tau_1$ ) is zero. Therefore,

$$(I - \tau_1 e_1^T)^{-1} = (I + \tau_1 e_1^T)$$

Similarly,

$$(I - \tau_2 e_2^T)^{-1} = (I + \tau_2 e_2^T)$$

Thus,

$$L = (I + \tau_1 e_1^T)(I + \tau_2 e_2^T).$$

Now, note that because  $\tau_2$  is only nonzero in the third element,  $e_1^T \tau_2 = 0$ ; thus,

$$\begin{aligned} L &= (I + \tau_1 e_1^T)(I + \tau_2 e_2^T) \\ &= (I + \tau_1 e_1^T + \tau_2 e_2^T + \tau_1 (e_1^T \tau_2) e_2^T) \\ &= I + \tau_1 e_1^T + \tau_2 e_2^T \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 2 & 1 \end{bmatrix}. \end{aligned}$$

The final factorization is

$$A = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 10 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 4 & 7 \\ 0 & -3 & -6 \\ 0 & 0 & 1 \end{bmatrix} = LU.$$

The subdiagonal elements of  $L$  are easy to read off: for  $i > j$ ,  $l_{ij}$  is the multiple of row  $j$  that we subtract from row  $i$  during elimination. This means that it is easy to read off the subdiagonal entries of  $L$  during the elimination process.

## Basic LU factorization

Let's generalize our previous algorithm and write a simple code for  $LU$  factorization. We will leave the issue of pivoting to a later discussion. We'll start with a purely loop-based implementation:

```
# Overwrites a copy of A with L and U
#
function my_lu(A)

  A = copy(A)
  m, n = size(A)
  L = UnitLowerTriangular(A) # View on A for tracking multipliers
```

```

U = UpperTriangular(A)      # Upper triangular view on A

for j = 1:n-1
    for i = j+1:n

        # Figure out multiple of row j to subtract from row i
        L[i,j] = A[i,j]/A[j,j]

        # Subtract off the appropriate multiple
        for k = j+1:n
            A[i,k] -= L[i,j]*A[j,k]
        end
    end
end

L, U
end

```

We can write the two innermost loops more concisely in terms of a *Gauss transformation*  $M_j = I - \tau_j e_j^T$ , where  $\tau_j$  is the vector of multipliers that appear when eliminating in column  $j$ :

```

#
# Overwrites a copy of A with L and U
#
function my_lu2(A)

    A = copy(A)
    m, n = size(A)
    L = UnitLowerTriangular(A) # View on A for tracking multipliers
    U = UpperTriangular(A)      # Upper triangular view on A

    for j = 1:n-1

        # Form vector of multipliers
        L[j+1:n,j] ./= A[j,j]

        # Apply Gauss transformation
        A[j+1:n,j+1:n] -= L[j+1:n,j]*A[j,j+1:n]'

    end

```

```
L, U  
end
```

## Problems to ponder

1. What is the complexity of the Gaussian elimination algorithm?
2. Describe how to find  $A^{-1}$  using Gaussian elimination. Compare the cost of solving a linear system by computing and multiplying by  $A^{-1}$  to the cost of doing Gaussian elimination and two triangular solves.
3. Consider a parallelepiped in  $\mathbb{R}^3$  whose sides are given by the columns of a 3-by-3 matrix  $A$ . Interpret  $LU$  factorization geometrically, thinking of Gauss transformations as shearing operations. Using the fact that shear transformations preserve volume, give a simple expression for the volume of the parallelepiped.