Systems of Equations - LU Factorization

We now shift to the next major topic: solving a linear system of equations $A\mathbf{x} = \mathbf{b}$.

Linear systems appear in models for traffic flow and other networks, Google search algorithms, computer graphics, quantum computing, Markov chains (probability)... In particular, it is often the case that we solve $A\mathbf{x} = \mathbf{b}_1$, followed by $A\mathbf{x} = \mathbf{b}_2$, then $A\mathbf{x} = \mathbf{b}_3$, and so on. That is, A is a "structure" matrix - doesn't really change - and b_i is a "loading vector" - forces coming in that change. For instance, this might be the case when testing designs. Our factorizations are largely designed to make later solves more efficient - avoid repeating work on A.

Formally, we call A the coefficient matrix, size $m \times n$, the variable vector is $\mathbf{x}$, size $n \times 1$, and $\mathbf{b}$ is the right-hand-side vector, size $m \times 1$.

Example 1. Write the following system of equations as an augmented matrix and solve.

$$\begin{aligned} 2x + 3y &= 0 \\ 4x + 2y &= 8 \end{aligned}$$

Remark. An *augmented matrix* contains the coefficients followed by a dashed line and the right-hand-side. That is:

$$\left[\begin{array}{cc:c} 2 & 3 & 0 \\ 4 & 2 & 8 \end{array}\right].$$

To solve, we use the process of *Gaussian elimination*. You can scale rows by any nonzero constant, add rows to other rows, and swap rows - though all our work in this handout will avoid row swaps. We will ultimately be separating the process into two steps - the factorization and the back substitution, and all the row scaling will wait until the back substitution.

The goal in Gaussian elimination is to reduce the coefficient matrix to the identity matrix; in the process the solution will appear on the right of the dashed lines. If we subtract twice the first row from the second, we get

$$\left[\begin{array}{cc:c} 2 & 3 & 0 \\ 0 & -4 & 8 \end{array}\right].$$

Then we can scale the second row by -4, which is

$$\left[\begin{array}{cc:c} 2 & 3 & 0 \\ 0 & 1 & -2 \end{array}\right].$$

Subtracting three times the second row from the first gives

$$\left[\begin{array}{cc:c} 2 & 0 & 6 \\ 0 & 1 & -2 \end{array}\right].$$

One more scaling

$$\left[\begin{array}{cc:c} 1 & 0 & 3 \\ 0 & 1 & -2 \end{array}\right]$$

gives the final vector $\mathbf{x} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$.

Example 2. Solve the following system.

$$\left[\begin{array}{rrr|r} 6 & 4 & 2 & 18 \\ -3 & -12 & 1 & 5 \\ 0 & 5 & 1 & -3 \end{array}\right]$$

Remark. Let's start by adding half of row one to row two. That gives

$$\left[\begin{array}{rrr|r} 6 & 4 & 2 & 18 \\ 0 & -10 & 2 & 14 \\ 0 & 5 & 1 & -3 \end{array}\right].$$

Then doing the same with row two and three, we get

$$\left[\begin{array}{rrr|r} 6 & 4 & 2 & 18 \\ 0 & -10 & 2 & 14 \\ 0 & 0 & 2 & 4 \end{array}\right].$$

Once we reach the bottom, we reverse the process starting with the last column and last row. Subtracting row three from rows one and two,

$$\left[\begin{array}{rrr|r} 6 & 4 & 0 & 14 \\ 0 & -10 & 0 & 10 \\ 0 & 0 & 2 & 4 \end{array}\right]$$

then adding $\frac{2}{5}$ row two to row one

$$\left[\begin{array}{rrr|r} 6 & 0 & 0 & 18 \\ 0 & -10 & 0 & 10 \\ 0 & 0 & 2 & 4 \end{array}\right].$$

A final scaling gives

$$\left[\begin{array}{rrr|r} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 2 \end{array}\right].$$

The row operations used during the process in this example can be stored for re-use in the LU factorization. The matrices L and U are so named because they are *lower triangular* and *upper triangular* matrices, meaning all entries above and below (respectively) the main diagonal are zeros. The matrix L should be *unit* lower triangular, meaning the diagonal entries are 1's. Then at each row elimination step, write the operation in the form $R_b - cR_p$ where R_p is the row containing the *pivot*, the number being used to introduce zeros. The pivot should be the leading nonzero term in its row. Then store the constant c in the corresponding entry in L.

For Example 2, we have U from the LU factorization when we reach

$$\left[\begin{array}{ccc|c} 6 & 4 & 2 & 18 \\ 0 & -10 & 2 & 14 \\ 0 & 0 & 2 & 4 \end{array}\right].$$

The row operations were $R_2 - (-\frac{1}{2})R_1$, so $(-\frac{1}{2})$ belongs in the second row first column of L; $R_3 - (0)R_1$, so 0 goes in the third row first column of L, and $R_3 - (-\frac{1}{2})R_2$ so $(-\frac{1}{2})$ goes in the second column third row of L. The final L is

$$L = \begin{bmatrix} 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 \\ 0 & -\frac{1}{2} & 1 \end{bmatrix}$$

and you can check: $A = LU$.

Example 3. For $A = \begin{bmatrix} 6 & 4 & 2 & 4 \\ -3 & -12 & 1 & 1 \\ 0 & 5 & 1 & 1 \end{bmatrix}$, the factorization goes like this.

1. Begin with $L = \begin{bmatrix} 1 & 0 & 0 \\ & 1 & 0 \\ & & 1 \end{bmatrix}$ and $U = \begin{bmatrix} & & \\ 0 & & \\ 0 & 0 & \end{bmatrix}$.

2. The first pivot must be the first element of the matrix, which is 6.

3. This pivot must be used to introduce a 0 to the first element of the second row. This is row operation $R_2 - (-\frac{1}{2})R_1$. The scalar in front of the pivot row is the element that goes into L at that location.

$$L = \begin{bmatrix} 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 \\ & & 1 \end{bmatrix} \qquad U = \begin{bmatrix} & & \\ 0 & & \\ 0 & 0 & \end{bmatrix}$$

4. Repeat for all rows below the pivot row. Since we do not need to introduce a 0 to the last row, we are effectively performing $R_3 - 0R_1$. We now have

$$L = \begin{bmatrix} 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 \\ 0 & & 1 \end{bmatrix} \qquad U = \begin{bmatrix} & & \\ 0 & & \\ 0 & 0 & \end{bmatrix}$$

5. The row that remains unchanged is copied as-is into the same row of U.

$$L = \begin{bmatrix} 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 \\ 0 & & 1 \end{bmatrix} \qquad U = \begin{bmatrix} 6 & 4 & 2 & 4 \\ 0 & & & \\ 0 & 0 & & \end{bmatrix}$$

6. A is now row equivalent to $\begin{bmatrix} 6 & 4 & 2 & 4 \\ 0 & -10 & 2 & 3 \\ 0 & 5 & 1 & 1 \end{bmatrix}$. The process begins again; -10 must be the pivot.

7. The next row operation is $R_3 - (-\frac{1}{2})R_2$.

$$L = \begin{bmatrix} 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 \\ 0 & -\frac{1}{2} & 1 \end{bmatrix} \qquad U = \begin{bmatrix} 6 & 4 & 2 & 4 \\ 0 & & & \\ 0 & 0 & & \end{bmatrix}$$

8. The second row of the row reduced A is remaining unchanged, so copy it into U.

$$L = \begin{bmatrix} 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 \\ 0 & -\frac{1}{2} & 1 \end{bmatrix} \qquad U = \begin{bmatrix} 6 & 4 & 2 & 4 \\ 0 & -10 & 2 & 3 \\ 0 & 0 & & \end{bmatrix}$$

9. Update A. $A = \begin{bmatrix} 6 & 4 & 2 & 4 \\ 0 & -10 & 2 & 3 \\ 0 & 0 & 2 & 2.5 \end{bmatrix}$

10. Copy the last unchanged row into U, completing the factorization.

$$L = \begin{bmatrix} 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 \\ 0 & -\frac{1}{2} & 1 \end{bmatrix} \qquad U = \begin{bmatrix} 6 & 4 & 2 & 4 \\ 0 & -10 & 2 & 3 \\ 0 & 0 & 2 & 2.5 \end{bmatrix}$$

□

Once you have an LU factorization, solving the system can be done through two steps of *back substitution*. First, augment L with b and solve for an intermediate result we'll call $\mathbf{y}$. Then augment U with y and solve to get the final answer.

Example 4. Let $A = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ -3 & 2 & 1 \end{bmatrix} \begin{bmatrix} 4 & 6 & 2 \\ 0 & 6 & 3 \\ 0 & 0 & 1 \end{bmatrix}$ and $\mathbf{b} = \begin{bmatrix} 8 \\ 7 \\ -17 \end{bmatrix}$. Solve $A\mathbf{x} = \mathbf{b}$.

Remark. First augment L with b:

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 8 \\ \frac{1}{2} & 1 & 0 & 7 \\ -3 & 2 & 1 & -17 \end{array}\right].$$

Subtract half of row one from row two, and add three times row one to row three.

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 8 \\ 0 & 1 & 0 & 3 \\ 0 & -2 & 1 & 7 \end{array}\right].$$

Now subtract two times the second row from the third row.

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 8 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 1 \end{array}\right].$$

Now augment U:

$$\left[\begin{array}{ccc|c} 6 & 4 & 2 & 8 \\ 0 & 6 & 3 & 3 \\ 0 & 0 & -1 & 1 \end{array}\right].$$

And work your way back.

$$\left[\begin{array}{ccc|c} 6 & 4 & 0 & 10 \\ 0 & 6 & 0 & 6 \\ 0 & 0 & -1 & 1 \end{array}\right]$$

$$\left[\begin{array}{ccc|c} 6 & 0 & 0 & 6 \\ 0 & 6 & 0 & 6 \\ 0 & 0 & -1 & 1 \end{array}\right]$$

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & -1 \end{array}\right]$$

On each repeated solve, we only do the back substitution, and don't have to recalculate L and U. How much work does this save?

When comparing algorithms, we primarily look at *operation counts*. That's the total number of operations (including $+, -, \times, \div, \sqrt{}$, etc.) the algorithm has to perform. Operation counts are a function of the size of the input. We'll assume from here on that A is square, that is, $n \times n$.

Now it is helpful to look at the algorithm in pseudocode form.

Algorithm 1 Computing an LU factorization - Classical Gaussian Elimination

```
for j = 1, ..., n-1 do
    for i = j+1, ..., n do
        c = a_ij / a_jj
        for k = j+1, ..., n do
            a_ik = a_ik - c(a_jk)
```

For each column j, we first scale to 1. That's $(n-j)$ divisions, one for each remaining row. Each of those $n-j$ rows has $n-j$ terms to be updated, each of which requires two operations: one multiplication and one subtraction. That results in a total of $2(n-j)^2$ operations per j. So the total operation count is:

$$\sum_{j=1}^{n-1} 2(n-j)^2 + (n-j).$$

For simplifying this, we're going to use a technique called re-indexing. That is, let k be the new index where each k is $n-j$. Then $k=1$ when $j=n-1$ and $k=n-1$ when $j=1$. So k also goes from 1 to $n-1$ and we have

$$\sum_{k=1}^{n-1} 2k^2 + k.$$

Then we can use formulas from calculus to evaluate

$$\begin{aligned} &2\frac{(n-1)(n)(2(n-1)+1)}{6} + \frac{(n-1)(n)}{2} \\ &= \frac{(n-1)(n)(2n-1)}{3} + \frac{(n-1)(n)}{2}. \end{aligned}$$

While we certainly can continue to simplify, what we're most concerned with is the largest term. The degree of this polynomial is what primarily determines how the algorithm scales as the matrices involved get larger.

In this case, the first term is $\frac{2n^3}{3}$, so we say that computing an LU factorization takes on the order of $\frac{2n^3}{3}$ operations. There is a specific notation called "big-O" notation $\mathcal{O}$ that is also convenient for this where the coefficients are also omitted. Saying that Classical Gaussian Elimination is an $\mathcal{O}(n^3)$ algorithm means that as n gets large, terms below cubic become negligible to the total running time of the algorithm, and the number of operations will grow like n^3.

Algorithm 2 Back Substitution

for i = n, . . . , 1 **do**
  for j=i+1,. . . , n **do**
    $b_i = b_i - a_{ij}(x_j)$
  $x_i = b_i/a_{ii}$

What is the operation count for back-substitution?

For each row i, there are $i-1$ rows to operate on. There are, however, only 2 operations per row, a multiply and a divide. So the total operation count is

$$\sum_{i=1}^{n} 2(i-1)+1 = \sum_{i=1}^{n} 2i - 1 = 2\frac{n(n+1)}{2} - n = n^2 + n - n = n^2.$$

Thus back-substitutions are a $\mathcal{O}(n^2)$ algorithm.

Example 5. Estimate the time required to carry out 1000 back-substitutions on a 5000 × 5000 system where one elimination takes 10 seconds.

Well, we know that an elimination is roughly $\frac{2(n^3)}{3} = \frac{2(5000)^3}{3}$ operations, so the computer is doing $\frac{2(5000)^3}{30}$ operations per second. Then we need to perform $(5000)^2$ operations, which gives $(5000)^2 \div \frac{2(5000)^3}{30} = \frac{15}{5000} = \frac{3}{1000} \approx .003$ seconds per back substitution. Then for 1000 eliminations we have about 3 seconds. That's right, 1000 back substitutions is faster than a single elimination!