Chapter 2

Linear Systems and the LU Decomposition

In Chapter 0, we discussed a variety of situations in which linear systems of equations $A\vec{x} = \vec{b}$ appear in mathematical theory and in practice. In this chapter, we tackle the basic problem head-on and explore numerical methods for solving such systems.

2.1 Solvability of Linear Systems

As introduced in §0.3.3, systems of linear equations like

$$\begin{aligned} 3x + 2y &= 6 \\ -4x + y &= 7 \end{aligned}$$

can be written in matrix form as in

$$\begin{pmatrix} 3 & 2 \\ -4 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 6 \\ 7 \end{pmatrix}.$$

More generally, we can write systems of the form $A\vec{x} = \vec{b}$ for $A \in \mathbb{R}^{m\times n}$, $\vec{x} \in \mathbb{R}^n$, and $\vec{b} \in \mathbb{R}^m$.

The solvability of the system must fall into one of three cases:

1. The system may not admit any solutions, as in:

   $$\begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}.$$

   This system asks that $x = -1$ and $x = 1$ simultaneously, obviously two incompatible conditions.

2. The system may admit a single solution; for instance, the system at the beginning of this section is solved by $(x, y) = (-{}^{8}/_{11}, {}^{45}/_{11})$.

3. The system may admit infinitely many solutions, e.g. $0\vec{x} = \vec{0}$. Notice that if a system $A\vec{x} = \vec{b}$ admits two solutions $\vec{x}_0$ and $\vec{x}_1$, then it automatically has infinitely many solutions of the form $c\vec{x}_0 + (1-c)\vec{x}_1$ for $c \in \mathbb{R}$, since

$$A(c\vec{x}_0 + (1-c)\vec{x}_1) = cA\vec{x}_0 + (1-c)A\vec{x}_1 = c\vec{b} + (1-c)\vec{b} = \vec{b}.$$

This linear system would be labeled *underdetermined*.

In general, the solvability of a system depends both on A and on $\vec{b}$. For instance, if we modify the unsolvable system above to be

$$\begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix},$$

then the system moves from having no solutions to infinitely many of the form $(1, y)$. In fact, every matrix A admits a right hand side $\vec{b}$ such that $A\vec{x} = \vec{b}$ is solvable, since $A\vec{x} = \vec{0}$ always can be solved by $\vec{x} \equiv \vec{0}$ regardless of A. Recall from §0.3.1 that matrix-vector multiplication can be viewed as linearly combining the columns of A with weights from $\vec{x}$. Thus, as mentioned in §0.3.3, we can expect $A\vec{x} = \vec{b}$ to be solvable exactly when $\vec{b}$ is in the column space of A.

In a broad way, the "shape" of the matrix $A \in \mathbb{R}^{m \times n}$ has considerable bearing on the solvability of $A\vec{x} = \vec{b}$. Recall that the columns of A are m-dimensional vectors. First, consider the case when A is "wide," that is, when it has more columns than rows ($n > m$). Each column is a vector in $\mathbb{R}^m$, so at most the column space can have dimension m. Since $n > m$, the n columns of A must then be linearly dependent; this implies that there exists an $\vec{x}_0 \neq \vec{0}$ such that $A\vec{x}_0 = \vec{0}$. Then, if we can solve $A\vec{x} = \vec{b}$ for $\vec{x}$, then $A(\vec{x} + \alpha\vec{x}_0) = A\vec{x} + \alpha A\vec{x}_0 = \vec{b} + \vec{0} = \vec{b}$, showing that there are actually infinitely many solutions $\vec{x}$ to $A\vec{x} = \vec{b}$. In other words, we have shown that *no wide matrix system admits a unique solution.*

When A is "tall," that is, when it has more rows than columns ($m > n$), then the n columns cannot span $\mathbb{R}^m$. Thus, there exists some vector $\vec{b}_0 \in \mathbb{R}^m \backslash \mathrm{col}\, A$. By definition, this $\vec{b}_0$ cannot satisfy $A\vec{x} = \vec{b}_0$ for *any* $\vec{x}$. In other words, *every tall matrix A admits systems $A\vec{x} = \vec{b}_0$ that are not solvable.*

Both of the situations above are far from favorable for designing numerical algorithms. For example, if a linear system admits many solutions we must first define *which* solution is desired from the user: after all, the solution $\vec{x} + 10^{31}\vec{x}_0$ might not be as meaningful as $\vec{x} - 0.1\vec{x}_0$. On the flip side, in the tall case even if $A\vec{x} = \vec{b}$ is solvable for a particular $\vec{b}$, any small perturbation $A\vec{x} = \vec{b} + \varepsilon\vec{b}_0$ is no longer solvable; this situation can appear simply because rounding procedures discussed in the last chapter can only approximate A and $\vec{b}$ in the first place.

Given these complications, in this chapter we will make some simplifying assumptions:

- We will consider only *square* $A \in \mathbb{R}^{n \times n}$.
- We will assume that A is *nonsingular*, that is, that $A\vec{x} = \vec{b}$ is solvable for any $\vec{b}$.

Recall from §0.3.3 that the nonsingularity condition is equivalent to asking that the columns of A span $\mathbb{R}^n$ and implies the existence of a matrix A^{-1} satisfying $A^{-1}A = AA^{-1} = I_{n \times n}$.

A misleading observation is to think that solving $A\vec{x} = \vec{b}$ is equivalent to computing the matrix A^{-1} explicitly and then multiplying to find $\vec{x} \equiv A^{-1}\vec{b}$. While this solution strategy certainly is valid, it can represent a considerable amount of overkill: after all, we're only interested in the n values in $\vec{x}$ rather than the n^2 values in A^{-1}. Furthermore, even when A is well-behaved it can be the case that writing A^{-1} yields numerical difficulties that can be bypassed.

2.2 Ad-Hoc Solution Strategies

In introductory algebra, we often approach the problem of solving a linear system of equations as an art form. The strategy is to "isolate" variables, iteratively writing alternative forms of the linear system until each line is of the form $x = const$.

In formulating systematic algorithms for solving linear systems, it is instructive to carry out an example of this solution process. Consider the following system:

$$\begin{aligned} y - z &= -1 \\ 3x - y + z &= 4 \\ x + y - 2z &= -3 \end{aligned}$$

In parallel, we can maintain a matrix version of this system. Rather than writing out $A\vec{x} = \vec{b}$ explicitly, we can save a bit of space by writing the "augmented" matrix below:

$$\left(\begin{array}{ccc|c} 0 & 1 & -1 & -1 \\ 3 & -1 & 1 & 4 \\ 1 & 1 & -2 & -3 \end{array}\right)$$

We can always write linear systems this way so long as we agree that the variables remain on the left hand side of the equations and the constants on the right.

Perhaps we wish to deal with the variable x first. For convenience, we may *permute* the rows of the system so that the third equation appears first:

$$\begin{aligned} x + y - 2z &= -3 \\ y - z &= -1 \\ 3x - y + z &= 4 \end{aligned} \qquad \left(\begin{array}{ccc|c} 1 & 1 & -2 & -3 \\ 0 & 1 & -1 & -1 \\ 3 & -1 & 1 & 4 \end{array}\right)$$

We can then *substitute* the first equation into the third to eliminate the $3x$ term. This is the same as scaling the relationship $x + y - 2z = -3$ by -3 and adding the result to the third equation:

$$\begin{aligned} x + y - 2z &= -3 \\ y - z &= -1 \\ -4y + 7z &= 13 \end{aligned} \qquad \left(\begin{array}{ccc|c} 1 & 1 & -2 & -3 \\ 0 & 1 & -1 & -1 \\ 0 & -4 & 7 & 13 \end{array}\right)$$

Similarly, to eliminate y from the third equation we can multiply the second equation by 4 and add the result to the third:

$$\begin{aligned} x + y - 2z &= -3 \\ y - z &= -1 \\ 3z &= 9 \end{aligned} \qquad \left(\begin{array}{ccc|c} 1 & 1 & -2 & -3 \\ 0 & 1 & -1 & -1 \\ 0 & 0 & 3 & 9 \end{array}\right)$$

We have now isolated z! Thus, we can scale the third row by $1/3$ to yield an expression for z:

$$\begin{aligned} x + y - 2z &= -3 \\ y - z &= -1 \\ z &= 3 \end{aligned} \qquad \left(\begin{array}{ccc|c} 1 & 1 & -2 & -3 \\ 0 & 1 & -1 & -1 \\ 0 & 0 & 1 & 3 \end{array}\right)$$

Now, we can substitute $z = 3$ into the other two equations to remove z from all but the final row:

$$\begin{aligned} x + y &= 3 \\ y &= 2 \\ z &= 3 \end{aligned} \qquad \left(\begin{array}{ccc|c} 1 & 1 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \end{array}\right)$$

Finally we make a similar substitution for y to complete the solve:

$$\begin{aligned} x &= 1 \\ y &= 2 \\ z &= 3 \end{aligned} \qquad \left(\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \end{array}\right)$$

This example might be somewhat pedantic, but looking back on our strategy yields a few important observations about how we can go about solving linear systems:

- We wrote successive systems $A_i\vec{x} = \vec{b}_i$ that can be viewed as simplifications of the original $A\vec{x} = \vec{b}$.
- We solved the system without ever writing down A^{-1}.
- We repeatedly used a few simple operations: scaling, adding, and permuting rows of the system.
- The same operations were applied to A and $\vec{b}$. If we scaled the k-th row of A, we also scaled the k-th row of $\vec{b}$. If we added rows k and ℓ of A, we added rows k and ℓ of $\vec{b}$.
- Less obviously, the steps of the solve did not depend on $\vec{b}$. That is, all of our decisions for how to solve were motivated by eliminating nonzero values in A rather than examining values in $\vec{b}$; $\vec{b}$ just came along for the ride.
- We terminated when we reduced the system to $I_{n\times n}\vec{x} = \vec{b}$.

We will use all of these general observations about solving linear systems to our advantage.

2.3 Encoding Row Operations

Looking back at the example in §2.2, we see that solving the linear system really only involved applying three operations: permutation, row scaling, and adding the scale of one row to another. In fact, we can solve *any* linear system this way, so it is worth exploring these operations in more detail.

2.3.1 Permutation

Our first step in §2.2 was to swap two of the rows in the system of equations. More generally, we might index the rows of a matrices using the numbers $1, \ldots, m$. Then, a *permutation* of those rows can be written as a function σ such that the list $\sigma(1), \ldots, \sigma(m)$ covers the same set of indices.

If $\vec{e}_k$ is the k-th standard basis function, then it is easy to see that the product $\vec{e}_k^\top A$ yields the k-th row of the matrix A. Thus, we can "stack" or concatenate these row vectors vertically to yield a matrix permuting the rows according to σ:

$$P_\sigma \equiv \left(\begin{array}{ccc} - & \vec{e}_{\sigma(1)}^\top & - \\ - & \vec{e}_{\sigma(2)}^\top & - \\ & \cdots & \\ - & \vec{e}_{\sigma(m)}^\top & - \end{array}\right)$$

That is, the product $P_\sigma A$ is exactly the matrix A with rows permuted according to σ.

Example 2.1 (Permutation matrices). *Suppose we wish to permute rows of a matrix in* $\mathbb{R}^{3\times 3}$ *with* $\sigma(1) = 2$, $\sigma(2) = 3$, *and* $\sigma(3) = 1$. *According to our formula we would have*

$$P_\sigma = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

From Example 2.1, we can see that P_σ has ones in positions of the form $(k, \sigma(k))$ and zeros elsewhere. The pair $(k, \sigma(k))$ represents the statement, "We would like row k of the output matrix to be row $\sigma(k)$ from the input matrix." Based on this description of a permutation matrix, it is easy to see that the inverse of P_σ is the transpose $P_\sigma^\top$, since this simply swaps the roles of the rows and columns – now we take row $\sigma(k)$ of the *input* and put it in row k of the *output*. In other words, $P_\sigma^\top P_\sigma = I_{m\times m}$.

2.3.2 Row Scaling

Suppose we write down a list of constants $a_1, \ldots, a_m$ and seek to scale the k-th row of some matrix A by a_k. This is obviously accomplished by applying the scaling matrix S_a given by:

$$S_a \equiv \begin{pmatrix} a_1 & 0 & 0 & \cdots \\ 0 & a_2 & 0 & \cdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_m \end{pmatrix}$$

Assuming that all a_k satisfy $a_k \neq 0$, it is easy to invert S_a by "scaling back:"

$$S_a^{-1} = S_{1/a} \equiv \begin{pmatrix} 1/a_1 & 0 & 0 & \cdots \\ 0 & 1/a_2 & 0 & \cdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1/a_m \end{pmatrix}$$

2.3.3 Elimination

Finally, suppose we wish to scale row k by a constant c and add the result to row ℓ. This operation may seem less natural than the previous two but actually is quite practical: It is the only one we need to combine equations from different rows of the linear system! We will realize this operation using an "elimination matrix" M such that the product MA applies this operation to matrix A.

Recall that the product $\vec{e}_k^\top A$ picks out the k-th row of A. Then, premultiplying by $\vec{e}_\ell$ yields a matrix $\vec{e}_\ell \vec{e}_k^\top A$, which is zero except the ℓ-th row is equal to the k-th of A.

Example 2.2 (Elimination matrix construction)**.** *Take*

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Suppose we wish to isolate the third row of $A \in \mathbb{R}^{3\times 3}$ and move it to row two. As discussed above, this operation is accomplished by writing:

$$\begin{aligned}
\vec{e}_2 \vec{e}_3^\top A &= \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \\
&= \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 7 & 8 & 9 \end{pmatrix} \\
&= \begin{pmatrix} 0 & 0 & 0 \\ 7 & 8 & 9 \\ 0 & 0 & 0 \end{pmatrix}
\end{aligned}$$

Of course, we multiplied right-to-left above but just as easily could have grouped the product as $(\vec{e}_2\vec{e}_3^\top)A$. The structure of this product is easy to see:

$$\vec{e}_2 \vec{e}_3^\top = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

We have succeeded in isolating row k and moving it to row ℓ. Our original elimination operation wanted to add c times row k to row ℓ, which we can now accomplish as the sum $A + c\vec{e}_\ell \vec{e}_k^\top A = (I_{n\times n} + c\vec{e}_\ell \vec{e}_k^\top)A$.

Example 2.3 (Solving a system)**.** *We can now encode each of our operations from Section 2.2 using the matrices we have constructed above:*

1. *Permute the rows to move the third equation to the first row:*

$$P = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

2. *Scale row one by -3 and add the result to row three: $E_1 = I_{3\times 3} - 3\vec{e}_3\vec{e}_1^\top$*

3. *Scale row two by 4 and add the result to row three: $E_2 = I_{3\times 3} + 4\vec{e}_3\vec{e}_2^\top$*

4. *Scale row three by $1/3$: $S = diag(1, 1, 1/3)$*

5. *Scale row three by 2 and add it to row one:* $E_3 = I_{3\times 3} + 2\vec{e}_1\vec{e}_3^\top$
6. *Add row three to row two:* $E_4 = I_{3\times 3} + \vec{e}_2\vec{e}_3^\top$
7. *Scale row three by -1 and add the result to row one:* $E_5 = I_{3\times 3} - \vec{e}_1\vec{e}_3^\top$

Thus, the inverse of A in Section 2.2 satisfies

$$A^{-1} = E_5E_4E_3SE_2E_1P.$$

Make sure you understand why these matrices appear in reverse *order!*

2.4 Gaussian Elimination

The sequence of steps chosen in Section 2.2 was by no means unique: There are many different paths that can lead to the solution of $A\vec{x} = \vec{b}$. Our steps, however, followed the strategy of *Gaussian elimination*, a famous algorithm for solving linear systems of equations.

More generally, let's say our system has the following "shape:"

$$\left(\begin{array}{c|c} A & \vec{b} \end{array}\right) = \left(\begin{array}{cccc|c} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{array}\right)$$

The algorithm proceeds in phases described below.

2.4.1 Forward Substitution

Consider the upper-left element of our matrix:

$$\left(\begin{array}{c|c} A & \vec{b} \end{array}\right) = \left(\begin{array}{cccc|c} \enclose{circle}{\times} & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{array}\right)$$

We will call this element our first *pivot* and will assume it is nonzero; if it is zero we can permute rows so that this is not the case. We first apply a scaling matrix so that the pivot equals one:

$$\left(\begin{array}{cccc|c} \enclose{circle}{1} & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{array}\right)$$

Now, we use the row containing the pivot to eliminate all other values underneath in the same column:

$$\left(\begin{array}{cccc|c} \enclose{circle}{1} & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{array}\right)$$

We now move our pivot to the next row and repeat a similar series of operations:

$$\left(\begin{array}{cccc|c} 1 & \times & \times & \times & \times \\ 0 & \textcircled{1} & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{array}\right)$$

Notice that a nice thing happens here. After the first pivot has been eliminated from all other rows, the first column is zero beneath row 1. This means we can safely add multiples of row two to rows underneath without affecting the zeros in column one.

We repeat this process until the matrix becomes *upper-triangular*:

$$\left(\begin{array}{cccc|c} 1 & \times & \times & \times & \times \\ 0 & 1 & \times & \times & \times \\ 0 & 0 & 1 & \times & \times \\ 0 & 0 & 0 & \textcircled{1} & \times \end{array}\right)$$

2.4.2 Back Substitution

Eliminating the remaining $\times$'s from the system is now a straight forward process. Now, we proceed in *reverse* order of rows and eliminate backward. For instance, after the first series of back substitution steps, we are left with the following shape:

$$\left(\begin{array}{cccc|c} 1 & \times & \times & 0 & \times \\ 0 & 1 & \times & 0 & \times \\ 0 & 0 & 1 & 0 & \times \\ 0 & 0 & 0 & \textcircled{1} & \times \end{array}\right)$$

Similarly, the second iteration yields:

$$\left(\begin{array}{cccc|c} 1 & \times & 0 & 0 & \times \\ 0 & 1 & 0 & 0 & \times \\ 0 & 0 & \textcircled{1} & 0 & \times \\ 0 & 0 & 0 & 1 & \times \end{array}\right)$$

After our final elimination step, we are left with our desired form:

$$\left(\begin{array}{cccc|c} \textcircled{1} & 0 & 0 & 0 & \times \\ 0 & 1 & 0 & 0 & \times \\ 0 & 0 & 1 & 0 & \times \\ 0 & 0 & 0 & 1 & \times \end{array}\right)$$

The right hand side now is the solution to the linear system $A\vec{x} = \vec{b}$.

2.4.3 Analysis of Gaussian Elimination

Each row operation in Gaussian elimination – scaling, elimination, and swapping two rows – obviously takes $O(n)$ time to complete, since you have to iterate over all n elements of a row (or two) of A. Once we choose a pivot, we have to do n forward- or back- substitutions into the rows below or above that pivot, resp.; this means the work for a single pivot in total is $O(n^2)$. In total, we choose one pivot per row, adding a final factor of n. Thus, it is easy enough to see that Gaussian elimination runs in $O(n^3)$ time.

One decision that takes place during Gaussian elimination that we have not discussed is the choice of *pivots*. Recall that we can permute rows of the linear system as we see fit before performing back- or forward- substitution. This operation is *necessary* to be able to deal with all possible matrices A. For example, consider what would happen if we did not use pivoting on the following matrix:

$$A = \begin{pmatrix} \textcircled{0} & 1 \\ 1 & 0 \end{pmatrix}$$

Notice that the circled element is exactly zero, so we cannot expect to divide row one by any number to replace that 0 with a 1. This does *not* mean the system is not solvable, it just means we must do *pivoting*, accomplished by swapping the first and second rows, to put a nonzero in that slot.

More generally, suppose our matrix looks like:

$$A = \begin{pmatrix} \textcircled{\varepsilon} & 1 \\ 1 & 0 \end{pmatrix},$$

where $0 < \varepsilon \ll 1$. If we do not pivot, then the first iteration of Gaussian elimination yields:

$$\tilde{A} = \begin{pmatrix} \textcircled{1} & 1/\varepsilon \\ 0 & -1/\varepsilon \end{pmatrix},$$

We have transformed a matrix A that looks nearly like a permutation matrix (in fact, $A^{-1} \approx A^\top$, a very easy way to solve the system!) into a system with potentially **huge** values $1/\varepsilon$.

This example shows that there are cases when we may wish to pivot even when doing so strictly speaking is not necessary. Since we are scaling by the reciprocal of the pivot value, clearly the most numerically stable options is to have a *large* pivot: Small pivots have large reciprocals, scaling numbers to large values in regimes that are likely to lose precision. There are two well-known pivoting strategies:

1. *Partial* pivoting looks through the current column and permutes rows of the matrix so that the largest absolute value appears on the diagonal.

2. *Full* pivoting iterates over the **entire** matrix and permutes both rows and columns to get the largest possible value on the diagonal. Notice that permuting columns of a matrix is a valid operation: it corresponds to changing the labeling of the variables in the system, or post-multiplying A by a permutation.

Example 2.4 (Pivoting)**.** *Suppose after the first iteration of Gaussian elimination we are left with the following matrix:*

$$\begin{pmatrix} 1 & 10 & -10 \\ 0 & \textcircled{0.1} & 9 \\ 0 & 4 & 6.2 \end{pmatrix}$$

If we implement partial pivoting, then we will look only in the second column and will swap the second and third rows; notice that we leave the 10 *in the first row since that one already has been visited by the algorithm:*

$$\begin{pmatrix} 1 & 10 & -10 \\ 0 & \textcircled{4} & 6.2 \\ 0 & 0.1 & 9 \end{pmatrix}$$

If we implement full pivoting, then we will move the 9*:*

$$\begin{pmatrix} 1 & -10 & 10 \\ 0 & \textcircled{9} & 0.1 \\ 0 & 6.2 & 4 \end{pmatrix}$$

Obviously full pivoting yields the best possible numerics, but the cost is a more expensive search for large elements in the matrix.

2.5 LU Factorization

There are many times when we wish to solve a sequence of problems $A\vec{x}_1 = \vec{b}_1, A\vec{x}_2 = \vec{b}_2, \ldots$. As we already have discussed, the steps of Gaussian elimination for solving $A\vec{x} = \vec{b}_k$ depend mainly on the structure of A rather than the values in a particular $\vec{b}_k$. Since A is kept constant here, we may wish to "remember" the steps we took to solve the system so that each time we are presented with a new $\vec{b}$ we do not have to start from scratch.

Solidifying this suspicion that we can move some of the $O(n^3)$ for Gaussian elimination into precomputation time, recall the *upper triangular* system resulting after the forward substitution stage:

$$\left(\begin{array}{cccc|c} 1 & \times & \times & \times & \times \\ 0 & 1 & \times & \times & \times \\ 0 & 0 & 1 & \times & \times \\ 0 & 0 & 0 & \textcircled{1} & \times \end{array}\right)$$

In fact, solving this system by back-substitution only takes $O(n^2)$ time! Why? Back substitution in this case is far easier thanks to the structure of the zeros in the system. For example, in the first series of back substitutions we obtain the following matrix:

$$\left(\begin{array}{cccc|c} 1 & \times & \times & 0 & \times \\ 0 & 1 & \times & 0 & \times \\ 0 & 0 & 1 & 0 & \times \\ \textcircled{0} & \textcircled{0} & \textcircled{0} & 1 & \times \end{array}\right)$$

Since we know that the (circled) values to the left of the pivot are zero by construction, we don't need to copy them explicitly. Thus, this step only took $O(n)$ time rather than $O(n^2)$ taken by forward substitution.

Now, our next pivot does a similar substitution:

$$\left(\begin{array}{cccc|c} 1 & \times & 0 & 0 & \times \\ 0 & 1 & 0 & 0 & \times \\ \textcircled{0} & \textcircled{0} & 1 & \textcircled{0} & \times \\ 0 & 0 & 0 & 1 & \times \end{array}\right)$$

Once again the zeros on both sides of the 1 do not need to be copied explicitly.

Thus, we have found:

Observation. *While Gaussian elimination takes $O(n^3)$ time, solving triangular systems takes $O(n^2)$ time.*

2.5.1 Constructing the Factorization

Recall from §2.3 that all the operations in Gaussian elimination can be thought of as pre-multiplying $A\vec{x} = \vec{b}$ by different matrices M to obtain an easier system $(MA)\vec{x} = M\vec{b}$. As we demonstrated in Example 2.3, from this standpoint each step of Gaussian elimination represents a system $(M_k \cdots M_2 M_1 A)\vec{x} = M_k \cdots M_2 M_1 \vec{b}$. Of course, explicitly storing these matrices M_k as $n \times n$ objects is overkill, but keeping this interpretation in mind from a theoretical perspective simplifies many of our calculations.

After the forward substitution phase of Gaussian elimination, we are left with an *upper triangular* matrix, which we can call $U \in \mathbb{R}^{n\times n}$. From the matrix multiplication perspective, we can write:

$$M_k \cdots M_1 A = U,$$

or, equivalently,

$$\begin{aligned} A &= (M_k \cdots M_1)^{-1} U \\ &= (M_1^{-1} M_2^{-1} \cdots M_k^{-1}) U \\ &\equiv LU, \text{ if we make the definition } L \equiv M_1^{-1} M_2^{-1} \cdots M_k^{-1}. \end{aligned}$$

We don't know anything about the structure of L yet, but we do know that systems of the form $U\vec{y} = \vec{d}$ are easier to solve since U is upper triangular. If L is equally nice, we could solve $A\vec{x} = \vec{b}$ in two steps, by writing $(LU)\vec{x} = \vec{b}$, or $\vec{x} = U^{-1}L^{-1}\vec{b}$:

1. Solve $L\vec{y} = \vec{b}$ for $\vec{y}$, yielding $\vec{y} = L^{-1}\vec{b}$.
2. Now that we have $\vec{y}$, solve $U\vec{x} = \vec{y}$, yielding $\vec{x} = U^{-1}\vec{y} = U^{-1}(L^{-1}\vec{b}) = (LU)^{-1}\vec{b} = A^{-1}\vec{b}$. We already know that this step only takes $O(n^2)$ time.

Our remaining task is to make sure that L has nice structure that will make solving $L\vec{y} = \vec{b}$ easier than solving $A\vec{x} = \vec{b}$. Thankfully–and unsurprisingly–we will find that L is *lower triangular* and thus can be solved using $O(n^2)$ forward substitution.

To see this, suppose for now that we do not implement pivoting. Then, each of our matrices M_k is either a scaling matrix or has the structure

$$M_k = I_{n\times n} + c\vec{e}_\ell \vec{e}_k^\top,$$

where $\ell > k$ since we only have carried out forward substitution. Remember that this matrix serves a specific purpose: Scale row k by c and add the result to row ℓ. This operation obviously is easy to undo: Scale row k by c and *subtract* the result from row ℓ. We can check this formally:

$$\begin{aligned} (I_{n\times n} + c\vec{e}_\ell \vec{e}_k^\top)(I_{n\times n} - c\vec{e}_\ell \vec{e}_k^\top) &= I_{n\times n} + (-c\vec{e}_\ell \vec{e}_k^\top + c\vec{e}_\ell \vec{e}_k^\top) - c^2 \vec{e}_\ell \vec{e}_k^\top \vec{e}_\ell \vec{e}_k^\top \\ &= I_{n\times n} - c^2 \vec{e}_\ell (\vec{e}_k^\top \vec{e}_\ell) \vec{e}_k^\top \\ &= I_{n\times n} \text{ since } \vec{e}_k^\top \vec{e}_\ell = \vec{e}_k \cdot \vec{e}_\ell, \text{ and } k \neq \ell \end{aligned}$$

So, the L matrix is the product of scaling matrices and matrices of the form $M^{-1} = I_{n\times n} + c\vec{e}_\ell \vec{e}_k^\top$ are lower triangular when $\ell > k$. Scaling matrices are diagonal, and the matrix M is lower triangular. You will show in Exercise 2.1 that the product of lower triangular matrices is lower triangular, showing in turn that L is lower triangular as needed.

We have shown that if it is possible to carry out Gaussian elimination of A without using pivoting, you can factor $A = LU$ into the product of lower- and upper-triangular matrices. Forward and back substitution each take $O(n^2)$ time, so if this factorization can be computed ahead of time the linear solve can be carried out faster than full $O(n^3)$ Gaussian elimination. You will show in Exercise 2.2 what happens when we carry out LU with pivoting; no major changes are necessary.

2.5.2 Implementing LU

A simple implementation of Gaussian elimination to solve $A\vec{x} = \vec{b}$ is straightforward enough to formulate. In particular, as we have discussed earlier, we can form the augmented matrix $(A \mid \vec{b})$ and apply row operations one at a time to this $n \times (n+1)$ block until it looks like $(I_{n\times n} A^{-1} \mid \vec{b})$. This process, however, is *destructive*, that is, in the end we care only about the last column of the augmented matrix and have kept no evidence of our solution path. Such behavior clearly is not acceptable for LU factorization.

Let's examine what happens when we multiply two elimination matrices:

$$(I_{n\times n} - c_\ell \vec{e}_\ell \vec{e}_k^\top)(I_{n\times n} - c_p \vec{e}_p \vec{e}_k^\top) = I_{n\times n} - c_\ell \vec{e}_\ell \vec{e}_k^\top - c_p \vec{e}_p \vec{e}_k^\top$$

As in our construction of the inverse of an elimination matrix, the product of the two $\vec{e}_i$ terms vanishes since the standard basis is orthogonal. This formula shows that after the pivot is scaled to 1, the product of the elimination matrices used to forward substitute for that pivot has the form:

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \textcircled{1} & 0 & 0 \\ 0 & \times & 1 & 0 \\ 0 & \times & 0 & 1 \end{pmatrix},$$

where the values $\times$ are the values used to eliminate the rest of the column. Multiplying matrices of this form together shows that the elements beneath the diagonal of L just come from coefficients used to accomplish substitution.

We can make one final decision to keep the elements along the diagonal of L in the LU factorization equal to 1. This decision is a legitimate one, since we can always post-multiply a L by a scaling matrix S taking these elements to 1 and write $LU = (LS)(S^{-1}U)$ without affecting the triangular pattern of L or U. With this decision in place, we can compress our storage of *both* L and U into a single $n \times n$ matrix whose upper triangle is U and which is equal to L beneath the diagonal; the missing diagonal elements of L are all 1.

We are now ready to write pseudocode for the simplest LU factorization strategy in which we do not permute rows or columns to obtain pivots:

```
// Takes as input an n-by-n matrix A[i,j]
// Edits A in place to obtain the compact LU factorization described above

for pivot from 1 to n {
    pivotValue = A[pivot,pivot]; // Bad assumption that this is nonzero!
    for eliminationRow from (pivot+1) to n { // Eliminate values beneath the pivot
        // How much to scale the pivot row to eliminate the value in the current
        // row; notice we're not scaling the pivot row to 1, so we divide by the
        // pivot value
        scale = A[eliminationRow,pivot]/pivotValue;

        // Since we /subtract/ scale times the pivot row from the current row
        // during Gaussian elimination, we /add/ it in the inverse operation
        // stored in L
        A[eliminationRow,pivot] = scale;

        // Now, as in Gaussian elimination, perform the row operation on the rest
        // of the row:  this will become U
        for eliminationCol from (pivot+1) to n
            A[eliminationRow,eliminationCol] -= A[pivot,eliminationCol]*scale;
    }
}
```

2.6 Problems

Problem 2.1. *Product of lower triangular things is lower triangular; product of pivot matrices looks right*

Problem 2.2. *Implement LU with pivoting*

Problem 2.3. *Non-square LU*