CHAPTER 3

Linear Systems and the LU Decomposition

CONTENTS

WE commence our discussion of numerical algorithms by deriving ways to solve the linear system of equations $A\vec{x} = \vec{b}$. We will explore applications of these systems in Chapter 4, showing a variety of computational problems that can be approached by constructing appropriate A and $\vec{b}$ and solving for $\vec{x}$. Furthermore, solving a linear system will serve as a basic step in larger methods for optimization, simulation, and other numerical tasks considered in almost all future chapters. For these reasons, a thorough treatment and understanding of linear systems is critical.

3.1 SOLVABILITY OF LINEAR SYSTEMS

As introduced in §1.3.4, 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 linear systems in 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 $A\vec{x} = \vec{b}$ 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 enforces two incompatible conditions simultaneously: $x = -1$ and $x = 1$.

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., $0\vec{x} = \vec{0}$. If a system $A\vec{x} = \vec{b}$ admits two distinct solutions $\vec{x}_0$ and $\vec{x}_1$, then it automatically has infinitely many solutions of the form $t\vec{x}_0 + (1-t)\vec{x}_1$ for all $t \in \mathbb{R}$, since
$$A(t\vec{x}_0 + (1-t)\vec{x}_1) = tA\vec{x}_0 + (1-t)A\vec{x}_1 = t\vec{b} + (1-t)\vec{b} = \vec{b}.$$
Because it has multiple solutions, this linear system is labeled *underdetermined.*

The solvability of the system $A\vec{x} = \vec{b}$ depends both on A and on $\vec{b}$. For instance, if we modify the unsolvable system above to
$$\begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix},$$
then the system changes from having no solutions to infinitely many of the form $(1, y)$. 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 $\vec{x} = \vec{0}$ regardless of A.

For alternative intuition about the solvability of linear systems, recall from §1.3.1 that the matrix-vector product $A\vec{x}$ can be viewed as a linear combination of the columns of A with weights from $\vec{x}$. Thus, as mentioned in §1.3.4, $A\vec{x} = \vec{b}$ is 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 $A\vec{x} = \vec{b}$. 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 be linearly dependent; this implies that there exists a set of weights $\vec{x}_0 \neq \vec{0}$ such that $A\vec{x}_0 = \vec{0}$. 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:

No wide matrix system admits a unique solution.

When A is "tall," that is, when it has more rows than columns ($m > n$), then its n columns cannot possibly span the larger-dimensional $\mathbb{R}^m$. For this reason, 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* $\vec{x}$. That is:

For every tall matrix A, there exists a $\vec{b}_0$ such that $A\vec{x} = \vec{b}_0$ is not solvable.

The situations above are far from favorable for designing numerical algorithms. In the wide case, if a linear system admits many solutions, we must specify *which* solution is desired by the user. After all, the solution $\vec{x} + 10^{31}\vec{x}_0$ might not be as meaningful as $\vec{x} - 0.1\vec{x}_0$. In the tall case, even if $A\vec{x} = \vec{b}$ is solvable for a particular $\vec{b}$, a small perturbation $A\vec{x} = \vec{b} + \varepsilon\vec{b}_0$ may not be solvable. The rounding procedures discussed in the last chapter easily can move a tall system from solvable to unsolvable.

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 $\vec{b}$.

From §1.3.4, the nonsingularity condition ensures that the columns of A span $\mathbb{R}^n$ and implies the existence of a matrix A^{-1} satisfying $A^{-1}A = AA^{-1} = I_{n\times n}$. We will relax these conditions in subsequent chapters.

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 formula is valid mathematically, it can represent a considerable amount of overkill and potential for numerical instability for several reasons:

- The matrix A^{-1} may contain values that are difficult to express in floating-point precision, in the same way that $1/\varepsilon \to \infty$ as $\varepsilon \to 0$.
- It may be possible to tune the solution strategy both to A and to $\vec{b}$, e.g., by working with the columns of A that are the closest to $\vec{b}$ first. Strategies like these can provide higher numerical stability.
- Even if A is sparse, meaning it contains many zero values that do not need to be stored explicitly, or has other special structure, the same may not be true for A^{-1}.

We highlight this point as a common source of error and inefficiency in numerical software:

Avoid computing A^{-1} explicitly unless you have a strong justification for doing so.

3.2 AD-HOC SOLUTION STRATEGIES

In introductory algebra, we often approach the problem of solving a linear system of equations as a puzzle rather than as a mechanical exercise. The strategy is to "isolate" variables, iteratively simplifying individual equalities until each is of the form $x = \text{const}$. To formulate step-by-step algorithms for solving linear systems, it is instructive to carry out an example of this methodology with an eye for aspects that can be fashioned into a general technique.

We will consider the following system:

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

Alongside each simplification step, we will maintain a matrix system encoding the current state. Rather than writing out $A\vec{x} = \vec{b}$ explicitly, we save space using the augmented matrix

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

We can write linear systems this way so long as we agree that variable coefficients remain on the left of the line and the constants on the right.

Perhaps we wish to deal with the variable x first. For convenience, we can *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 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 scale 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! We 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 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 reveal the solution:

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

Revisiting the steps above yields a few observations about how to solve 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 elementary operations: scaling, adding, and permuting rows.
- 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 $\vec{b}$.
- The steps did not depend on $\vec{b}$. That is, all of our decisions were motivated by eliminating nonzero values in A; $\vec{b}$ just came along for the ride.
- We terminated when we reached the simplified system $I_{n\times n}\vec{x}=\vec{b}$.

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

3.3 ENCODING ROW OPERATIONS

Looking back at the example in §3.2, we see that solving $A\vec{x} = \vec{b}$ only involved three operations: permutation, row scaling, and adding a multiple of one row to another. We can solve *any* linear system this way, so it is worth exploring these operations in more detail.

A pattern we will see for the remainder of this chapter is the use of matrices to express row operations. For example, the following two descriptions of an operation on a matrix A are equivalent:

1. Scale the first row of A by 2.
2. Replace A with S_2A, where S_2 is defined by:

$$S_2 \equiv \begin{pmatrix} 2 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}.$$

When presenting the theory of matrix simplification, it is cumbersome to use words to describe each operation, so when possible we will encode matrix algorithms as a series of pre- and post-multiplications by specially designed matrices like S_2 above.

This description in terms of matrices, however, is a *theoretical* construction. Implementations of algorithms for solving linear systems should not construct matrices like S_2 explicitly. For example, if $A \in \mathbb{R}^{n\times n}$, it should take n steps to scale the first row of A by 2, but explicitly constructing $S_2 \in \mathbb{R}^{n\times n}$ and applying it to A takes n^3 steps! That is, we will show for notational convenience that row operations *can* be encoded using matrix multiplication, but they do not *have* to be encoded this way.

3.3.1 Permutation

Our first step in §3.2 was to swap two of the rows. More generally, we might index the rows of a matrix using the integers $1, \ldots, m$. A *permutation* of those rows can be written as a function $\sigma : \{1, \ldots, m\} \to \{1, \ldots, m\}$ such that $\{\sigma(1), \ldots, \sigma(m)\} = \{1, \ldots, m\}$, that is, σ maps every index to a different target.

If $\vec{e}_k$ is the k-th standard basis vector, the product $\vec{e}_k^\top A$ is the k-th row of the matrix A. We can "stack" or concatenate these row vectors vertically to yield a matrix permuting the rows according to σ:

$$P_\sigma \equiv \begin{pmatrix} - & \vec{e}_{\sigma(1)}^\top & - \\ - & \vec{e}_{\sigma(2)}^\top & - \\ & \vdots & \\ - & \vec{e}_{\sigma(m)}^\top & - \end{pmatrix}.$$

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

Example 3.1 (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 have

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

From Example 3.1, P_σ has ones in positions indexed $(k, \sigma(k))$ and zeros elsewhere. Reversing the order of each pair, that is, putting ones in positions indexed $(\sigma(k), k)$ and zeros elsewhere, undoes the effect of the permutation. Hence, the inverse of P_σ must be its transpose $P_\sigma^\top$. Symbolically, we write $P_\sigma^\top P_\sigma = I_{m\times m}$, or equivalently $P_\sigma^{-1} = P_\sigma^\top$.

3.3.2 Row Scaling

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

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

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

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

If any a_k equals zero, S_a is not invertible.

3.3.3 Elimination

Finally, suppose we wish to scale row k by a constant c and add the result to row ℓ; we will assume $k \neq \ell$. This operation may seem less natural than the previous two, but actually it is quite practical. In particular, 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 is the result of applying this operation to matrix A.

The product $\vec{e}_k^\top A$ picks out the k-th row of A. Pre-multiplying the result by $\vec{e}_\ell$ yields a matrix $\vec{e}_\ell \vec{e}_k^\top A$ that is zero except on its ℓ-th row, which is equal to the k-th row of A.

Example 3.2 (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:

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

We multiplied right to left above but just as easily could have grouped the product as $(\vec{e}_2\vec{e}_3^\top)A$. Grouping this way involves application of the matrix

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

We have succeeded in isolating row k and moving it to row ℓ. Our original elimination operation was to add c times row k to row ℓ, which we can now carry out using the sum $A + c\vec{e}_\ell\vec{e}_k^\top A = (I_{n\times n} + c\vec{e}_\ell\vec{e}_k^\top)A$. Isolating the coefficient of A, the desired elimination matrix is $M \equiv I_{n\times n} + c\vec{e}_\ell\vec{e}_k^\top$.

The action of M can be reversed: Scale row k by c and *subtract* the result from row ℓ. We can check this formally:

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

That is, $M^{-1} = I_{n\times n} - c\vec{e}_\ell\vec{e}_k^\top$.

Example 3.3 (Solving a system)**.** We can now encode each of our operations from Section 3.2 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:

$$E_1 = I_{3\times 3} - 3\vec{e}_3\vec{e}_1^\top = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -3 & 0 & 1 \end{pmatrix}.$$

3. Scale row two by 4 and add the result to row three:

$$E_2 = I_{3\times 3} + 4\vec{e}_3\vec{e}_2^\top = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 4 & 1 \end{pmatrix}.$$

4. Scale row three by $1/3$:

$$S = \text{diag}(1, 1, 1/3) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1/3 \end{pmatrix}.$$

5. Scale row three by 2 and add it to row one:

$$E_3 = I_{3\times 3} + 2\vec{e}_1\vec{e}_3^\top = \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

6. Add row three to row two:

$$E_4 = I_{3\times 3} + \vec{e}_2\vec{e}_3^\top = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}.$$

7. Scale row two by -1 and add the result to row one:

$$E_5 = I_{3\times 3} - \vec{e}_1\vec{e}_3^\top = \begin{pmatrix} 1 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Thus, the inverse of A in Section 3.2 satisfies

$$\begin{aligned} A^{-1} &= E_5E_4E_3SE_2E_1P \\ &= \begin{pmatrix} 1 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1/3 \end{pmatrix} \\ &\quad \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 4 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -3 & 0 & 1 \end{pmatrix}\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 1/3 & 1/3 & 0 \\ 7/3 & 1/3 & -1 \\ 4/3 & 1/3 & -1 \end{pmatrix}. \end{aligned}$$

Make sure you understand why these matrices appear in *reverse* order! As a reminder, we would not normally construct A^{-1} by multiplying the matrices above, since these operations can be implemented more efficiently than generic matrix multiplication. Even so, it is valuable to check that the theoretical operations we have defined are equivalent to the ones we have written in words.

3.4 GAUSSIAN ELIMINATION

The sequence of steps chosen in Section 3.2 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, used *Gaussian elimination*, a famous algorithm for solving linear systems of equations.

To introduce this algorithm, let's say our system has the following generic "shape":

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

Here, an $\times$ denotes a potentially nonzero value. Gaussian elimination proceeds in phases described below.

3.4.1 Forward-Substitution

Consider the upper-left element of the matrix:

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

We will call this element the 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 scale the first row by the reciprocal of the pivot so that the value in the pivot position is one:

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

Now, we use the row containing the pivot to eliminate all other values underneath in the same column using the strategy in §3.3.3:

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

At this point, the entire first column is zero below the pivot. We change the pivot label to the element in position $(2, 2)$ and repeat a similar series of operations to rescale the pivot row and use it to cancel the values underneath:

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

Now, our matrix begins to gain some structure. After the first pivot has been eliminated from all other rows, the first column is zero except for the leading one. Thus, any row operation involving rows two to m will not affect the zeros in column one. Similarly, after the second pivot has been processed, operations on rows three to m will not remove the zeros in columns one and two.

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

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

The method above of making a matrix upper triangular is known as *forward-substitution* and is detailed in Figure 3.1.

```
function FORWARD-SUBSTITUTION($A, \vec{b}$)
  ▷ Converts a system $A\vec{x} = \vec{b}$ to an upper-triangular system $U\vec{x} = \vec{y}$.
  ▷ Assumes invertible $A \in \mathbb{R}^{n\times n}$ and $\vec{b} \in \mathbb{R}^n$.

  $U, \vec{y} \leftarrow A, \vec{b}$                    ▷ $U$ will be upper triangular at completion
  for $p \leftarrow 1, 2, \ldots, n$                    ▷ Iterate over current pivot row $p$
    ▷ Optionally insert pivoting code here

    $s \leftarrow 1/u_{pp}$                 ▷ Scale row $p$ to make element at $(p, p)$ equal one
    $y_p \leftarrow s \cdot y_p$
    for $c \leftarrow p, \ldots, n : u_{pc} \leftarrow s \cdot u_{pc}$

    for $r \leftarrow (p+1), \ldots, n$                  ▷ Eliminate from future rows
      $s \leftarrow -u_{rp}$                         ▷ Scale row $p$ by $s$ and add to row $r$
      $y_r \leftarrow y_r + s \cdot y_p$
      for $c \leftarrow p, \ldots, n : u_{rc} \leftarrow u_{rc} + s \cdot u_{pc}$
  return $U, \vec{y}$
```

Figure 3.1 Forward-substitution without pivoting; see §3.4.3 for pivoting options.

3.4.2 Back-Substitution

Eliminating the remaining $\times$'s from the remaining upper-triangular system is an equally straightforward process proceeding in *reverse* order of rows and eliminating backward. After the first set 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 $A\vec{x} = \vec{b}$. Figure 3.2 implements this method of *back-substitution* in more detail.

3.4.3 Analysis of Gaussian Elimination

Each row operation in Gaussian elimination—scaling, elimination, and swapping two rows—takes $O(n)$ time to complete, since they iterate over all n elements of a row (or two) of A.

```
function Back-Substitution($U, \vec{y}$)
  ▷ Solves upper-triangular systems $U\vec{x} = \vec{y}$ for $\vec{x}$.

  $\vec{x} \leftarrow \vec{y}$                 ▷ We will start from $U\vec{x} = \vec{y}$ and simplify to $I_{n\times n}\vec{x} = \vec{x}$
  for $p \leftarrow n, n-1, \ldots, 1$                        ▷ Iterate backward over pivots
    for $r \leftarrow 1, 2, \ldots, p-1$                      ▷ Eliminate values above $u_{pp}$
      $x_r \leftarrow x_r - u_{rp}x_p/u_{pp}$
  return $\vec{x}$
```

Figure 3.2 Back-substitution for solving upper-triangular systems; this implementation returns the solution $\vec{x}$ to the system without modifying U.

Once we choose a pivot, we have to do n forward- or back-substitutions into the rows below or above that pivot, respectively; 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. Combining these counts, Gaussian elimination runs in $O(n^3)$ time.

One decision that takes place during Gaussian elimination meriting more discussion is the choice of pivots. We can permute rows of the linear system as we see fit before performing forward-substitution. This operation, called *pivoting*, 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:

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

The circled element is exactly zero, so we cannot scale row one by any value to replace that 0 with a 1. This does *not* mean the system is not solvable—although singular matrices are guaranteed to have this issue—but rather it means we must pivot by swapping the first and second rows.

To highlight a related issue, suppose A 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 ($A^{-1} \approx A^\top$, a very easy way to solve the system!) into a system with potentially **huge** values of the fraction $1/\varepsilon$. This example is one of many instances in which we should try to avoid dividing by vanishingly small numbers. In this way, there are cases when we may wish to pivot even when Gaussian elimination theoretically could proceed without such a step.

Since Gaussian elimination scales by the reciprocal of the pivot, the most numerically stable option is to have a *large* pivot. Small pivots have large reciprocals, which scale matrix elements to regimes that may 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 element in that column with the largest absolute value appears on the diagonal.

2. *Full pivoting* iterates over the **entire** matrix and permutes rows and columns to place the largest possible value on the diagonal. Permuting columns of a matrix is a valid operation after some added bookkeeping: it corresponds to changing the labeling of the variables in the system, or post-multiplying A by a permutation.

Full pivoting is more expensive computationally than partial pivoting since it requires iterating over the entire matrix (or using a priority queue data structure) to find the largest absolute value, but it results in enhanced numerical stability. Full pivoting is rarely necessary, and it is not enabled by default in common implementations of Gaussian elimination.

Example 3.4 (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; we leave the 10 in the first row since that row already has been visited during forward-substitution:

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

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

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

3.5 LU FACTORIZATION

There are many times when we wish to solve a sequence of problems $A\vec{x}_1 = \vec{b}_1, A\vec{x}_2 = \vec{b}_2, \ldots,$ where in each system the matrix A is the same. For example, in image processing we may apply the same filter encoded in A to a set of images encoded as $\vec{b}_1, \vec{b}_2, \ldots$. As we already have discussed, the steps of Gaussian elimination for solving $A\vec{x}_k = \vec{b}_k$ 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 cache the steps we took to solve the system so that each time we are presented with a new $\vec{b}_k$ we do not have to start from scratch. Such a caching strategy compromises between restarting Gaussian elimination for each $\vec{b}_i$ and computing the potentially numerically unstable inverse matrix A^{-1}.

Solidifying this suspicion that we can move some of the $O(n^3)$ expense for Gaussian elimination into precomputation time if we wish to reuse A, recall the *upper-triangular* system appearing after forward-substitution:

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

Unlike forward-substitution, solving this system by back-substitution only takes $O(n^2)$ time! Why? As implemented in Figure 3.2, back-substitution can take advantage of the structure of the zeros in the system. For example, consider the circled elements of the initial upper-triangular system:

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

Since we know that the (circled) values to the left of the pivot are zero by definition of an upper-triangular matrix, we do not need to scale them or copy them upward explicitly. If we ignore these zeros completely, this step of backward-substitution only takes n operations rather than the n^2 taken by the corresponding step of forward-substitution.

The next pivot benefits from a similar structure:

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

Again, the zeros on both sides of the one do not need to be copied explicitly.

A nearly identical method can be used to solve *lower*-triangular systems of equations via forward-substitution. Combining these observations, we have shown:

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

We will revisit the steps of Gaussian elimination to show that they can be used to factorize the matrix A as $A = LU$, where L is lower triangular and U is upper triangular, so long as pivoting is not needed to solve $A\vec{x} = \vec{b}$. Once the matrices L and U are obtained, solving $A\vec{x} = \vec{b}$ can be carried out by instead solving $LU\vec{x} = \vec{b}$ using forward-substitution followed by backward-substitution; these two steps combined take $O(n^2)$ time rather than the $O(n^3)$ time needed for full Gaussian elimination. This factorization also can be extended to a related and equally useful decomposition when pivoting is desired or necessary.

3.5.1 Constructing the Factorization

Other than full pivoting, from §3.3 we know 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 easier systems $(MA)\vec{x} = M\vec{b}$. As demonstrated in Example 3.3, from this standpoint, each step of Gaussian elimination brings a new system $(M_k \cdots M_2 M_1 A)\vec{x} = M_k \cdots M_2 M_1 \vec{b}$. 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 call $U \in \mathbb{R}^{n\times n}$. From the matrix multiplication perspective,

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

U is upper triangular by design, but we have not characterized the structure of L; our remaining task is to show that L is lower triangular. To do so, recall that in the absence of pivoting, each matrix M_i is either a scaling matrix or has the structure $M_i = I_{n\times n} + c\vec{e}_\ell \vec{e}_k^\top$, from §3.3.3, where $\ell > k$ since we carried out forward-substitution to obtain U. So, L is the product of scaling matrices and matrices of the form $M_i^{-1} = I_{n\times n} - c\vec{e}_\ell \vec{e}_k^\top$; these matrices are lower triangular since $\ell > k$. Since scaling matrices are diagonal, L is lower triangular by the following proposition:

Proposition 3.1. The product of two or more upper-triangular matrices is upper triangular, and the product of two or more lower-triangular matrices is lower triangular.

Proof. Suppose A and B are upper triangular, and define $C \equiv AB$. By definition of upper-triangular matrices, $a_{ij} = 0$ and $b_{ij} = 0$ when $i > j$. Fix two indices i and j with $i > j$. Then,

$$\begin{aligned} c_{ij} &= \sum_k a_{ik} b_{kj} \text{ by definition of matrix multiplication} \\ &= a_{i1} b_{1j} + a_{i2} b_{2j} + \cdots + a_{in} b_{nj}. \end{aligned}$$

The first $i - 1$ terms of the sum are zero because A is upper triangular, and the last $n - j$ terms are zero because B is upper triangular. Since $i > j$, $(i - 1) + (n - j) > n - 1$ and hence all n terms of the sum over k are zero, as needed.

If A and B are lower triangular, then $A^\top$ and $B^\top$ are upper triangular. By our proof above, $B^\top A^\top = (AB)^\top$ is upper triangular, showing that AB is again lower triangular. □

3.5.2 Using the Factorization

Having factored $A = LU$, we can solve $A\vec{x} = \vec{b}$ in two steps, by writing $(LU)\vec{x} = \vec{b}$, or equivalently $\vec{x} = U^{-1}L^{-1}\vec{b}$:

1. Solve $L\vec{y} = \vec{b}$ for $\vec{y}$, yielding $\vec{y} = L^{-1}\vec{b}$.
2. With $\vec{y}$ now fixed, solve $U\vec{x} = \vec{y}$ for $\vec{x}$.

Checking the validity of $\vec{x}$ as a solution of the system $A\vec{x} = \vec{b}$ comes from the following chain of equalities:

$$\begin{aligned} \vec{x} &= U^{-1}\vec{y} \text{ from the second step} \\ &= U^{-1}(L^{-1}\vec{b}) \text{ from the first step} \\ &= (LU)^{-1}\vec{b} \text{ since } (AB)^{-1} = B^{-1}A^{-1} \\ &= A^{-1}\vec{b} \text{ since we factored } A = LU. \end{aligned}$$

Forward- and back-substitution to carry out the two steps above each take $O(n^2)$ time. So, given the LU factorization of A, solving $A\vec{x} = \vec{b}$ can be carried out faster than full $O(n^3)$ Gaussian elimination. When pivoting is necessary, we will modify our factorization to include a permutation matrix P to account for the swapped rows and/or columns, e.g., $A = PLU$ (see Exercise 3.12). This minor change does not affect the asymptotic timing benefits of LU factorization, since $P^{-1} = P^\top$.

3.5.3 Implementing LU

The implementation of Gaussian elimination suggested in Figures 3.1 and 3.2 constructs U but not L. We can make some adjustments to factor $A = LU$ rather than solving a single system $A\vec{x} = \vec{b}$.

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 the construction of the inverse of an elimination matrix in §3.5.1, the remaining term vanishes by orthogonality of the standard basis vectors $\vec{e}_i$ since $k \neq p$. This formula shows that the product of elimination matrices used to forward-substitute a single pivot after it is scaled to 1 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 those used for forward-substitutions of the circled pivot. Products of matrices of this form performed in forward-substitution order combine the values below the diagonal, as demonstrated in the following example:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 0 & 1 & 0 \\ 4 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 5 & 1 & 0 \\ 0 & 6 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 7 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 5 & 1 & 0 \\ 4 & 6 & 7 & 1 \end{pmatrix}.$$

We constructed U by pre-multiplying A with a sequence of elimination and scaling matrices. We can construct L simultaneously via a sequence of *post*-multiplies by their inverses, starting from the identity matrix. These post-multiplies can be computed efficiently using the above observations about products of elimination matrices.

For any invertible diagonal matrix D, $(LD)(D^{-1}U)$ provides an alternative factorization of $A = LU$ into lower- and upper-triangular matrices. Thus, by rescaling we can decide to keep the elements along the diagonal of L in the LU factorization equal to 1. 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 LU factorization without pivoting, illustrated in Figure 3.3. This method extends the algorithm for forward-substitution by storing the corresponding elements of L under the diagonal rather than zeros. This method has three nested loops and runs in $O(n^3) \approx \frac{2}{3}n^3$ time. After precomputing this factorization, however, solving $A\vec{x} = \vec{b}$ only takes $O(n^2)$ time using forward- and backward-substitution.

3.6 EXERCISES

3.1 Can *all* matrices $A \in \mathbb{R}^{n\times n}$ be factored $A = LU$? Why or why not?

3.2 Solve the following system of equations using Gaussian elimination, writing the corresponding elimination matrix of each step:

$$\begin{pmatrix} 2 & 4 \\ 3 & 5 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \end{pmatrix}.$$

Factor the matrix on the left-hand side as a product $A = LU$.

```
function LU-FACTORIZATION-COMPACT($A$)
  ▷ Factors $A \in \mathbb{R}^{n\times n}$ to $A = LU$ in compact format.
  for $p \leftarrow 1, 2, \ldots, n$                    ▷ Choose pivots like in forward-substitution
    for $r \leftarrow p+1, \ldots, n$                   ▷ Forward-substitution row
      $s \leftarrow -a_{rp}/a_{pp}$           ▷ Amount to scale row $p$ for forward-substitution
      $a_{rp} \leftarrow -s$    ▷ $L$ contains $-s$ because it reverses the forward-substitution
      for $c \leftarrow p+1, \ldots, n$                 ▷ Perform forward-substitution
        $a_{rc} \leftarrow a_{rc} + s a_{pc}$
  return $A$
```

Figure 3.3 Pseudocode for computing the LU factorization of $A \in \mathbb{R}^{n\times n}$, stored in the compact $n \times n$ format described in §3.5.3. This algorithm will fail if pivoting is needed.

DH3.3 Factor the following matrix A as a product $A = LU$:

$$\begin{pmatrix} 1 & 2 & 7 \\ 3 & 5 & -1 \\ 6 & 1 & 4 \end{pmatrix}.$$

3.4 Modify the code in Figure 3.1 to include partial pivoting.

3.5 The discussion in §3.4.3 includes an example of a 2×2 matrix A for which Gaussian elimination without pivoting fails. In this case, the issue was resolved by introducing partial pivoting. If exact arithmetic is implemented to alleviate rounding error, does there exist a matrix for which Gaussian elimination fails unless *full* rather than partial pivoting is implemented? Why or why not?

3.6 Numerical algorithms appear in many components of simulation software for quantum physics. The Schrödinger equation and others involve *complex* numbers in $\mathbb{C}$, however, so we must extend the machinery we have developed for solving linear systems of equations to this case. Recall that a complex number $x \in \mathbb{C}$ can be written as $x = a + bi$, where $a, b \in \mathbb{R}$ and $i = \sqrt{-1}$. Suppose we wish to solve $A\vec{x} = \vec{b}$, but now $A \in \mathbb{C}^{n\times n}$ and $\vec{x}, \vec{b} \in \mathbb{C}^n$. Explain how a linear solver that takes only *real-valued* systems can be used to solve this equation.
Hint: Write $A = A_1 + A_2 i$, where $A_1, A_2 \in \mathbb{R}^{n\times n}$. Similarly decompose $\vec{x}$ and $\vec{b}$. In the end you will solve a $2n \times 2n$ real-valued system.

3.7 Suppose $A \in \mathbb{R}^{n\times n}$ is invertible. Show that A^{-1} can be obtained via Gaussian elimination on augmented matrix

$$\left(\begin{array}{c|c} A & I_{n\times n} \end{array} \right).$$

3.8 Show that if L is an invertible lower-triangular matrix, none of its diagonal elements can be zero. How does this lemma affect the construction in §3.5.3?

3.9 Show that the inverse of an (invertible) lower-triangular matrix is lower triangular.

3.10 Show that any invertible matrix $A \in \mathbb{R}^{n\times n}$ with $a_{11} = 0$ cannot have a factorization $A = LU$ for lower-triangular L and upper-triangular U.

3.11 Show how the LU factorization of $A \in \mathbb{R}^{n\times n}$ can be used to compute the determinant of A.

3.12 For numerical stability and generality, we incorporated pivoting into our methods for Gaussian elimination. We can modify our construction of the LU factorization somewhat to incorporate pivoting as well.

(a) Argue that following the steps of Gaussian elimination on a matrix $A \in \mathbb{R}^{n\times n}$ with partial pivoting can be used to write $U = L_{n-1}P_{n-1}\cdots L_2P_2L_1P_1A$, where the P_i's are permutation matrices, the L_i's are lower triangular, and U is upper triangular.

(b) Show that P_i is a permutation matrix that swaps rows i and j for some $j \geq i$. Also, argue that L_i is the product of matrices of the form $I_{n\times n} + c\vec{e}_k\vec{e}_i^\top$ where $k > i$.

(c) Suppose $j, k > i$. Show $P_{jk}(I_{n\times n} + c\vec{e}_k\vec{e}_i^\top) = (I_{n\times n} + c\vec{e}_j\vec{e}_i^\top)P_{jk}$, where P_{jk} is a permutation matrix swapping rows j and k.

(d) Combine the previous two parts to show that

$$L_{n-1}P_{n-1}\cdots L_2P_2L_1P_1 = L_{n-1}L'_{n-2}L'_{n-3}\cdots L'_1P_{n-1}\cdots P_2P_1,$$

where $L'_1, \ldots, L'_{n-2}$ are lower triangular.

(e) Conclude that $A = PLU$, where P is a permutation matrix, L is lower triangular, and U is upper triangular.

(f) Extend the method from §3.5.2 for solving $A\vec{x} = \vec{b}$ when we have factored $A = PLU$, without affecting the time complexity compared to factorizations $A = LU$.

3.13 ("Block LU decomposition") Suppose a square matrix $M \in \mathbb{R}^{n\times n}$ is written in block form as

$$M = \begin{pmatrix} A & B \\ C & D \end{pmatrix},$$

where $A \in \mathbb{R}^{k\times k}$ is square and invertible.

(a) Show that we can decompose M as the product

$$M = \begin{pmatrix} I & 0 \\ CA^{-1} & I \end{pmatrix}\begin{pmatrix} A & 0 \\ 0 & D - CA^{-1}B \end{pmatrix}\begin{pmatrix} I & A^{-1}B \\ 0 & I \end{pmatrix}.$$

Here, I denotes an identity matrix of appropriate size.

(b) Suppose we decompose $A = L_1U_1$ and $D - CA^{-1}B = L_2U_2$. Show how to construct an LU factorization of M given these additional matrices.

(c) Use this structure to define a recursive algorithm for LU factorization; you can assume $n = 2^\ell$ for some $\ell > 0$. How does the efficiency of your method compare with that of the LU algorithm introduced in this chapter?

3.14 Suppose $A \in \mathbb{R}^{n\times n}$ is *columnwise diagonally dominant*, meaning that for all i, $\sum_{j\neq i}|a_{ji}| < |a_{ii}|$. Show that Gaussian elimination on A can be carried out without pivoting. Is this necessarily a good idea from a numerical standpoint?

3.15 Suppose $A \in \mathbb{R}^{n\times n}$ is invertible and admits a factorization $A = LU$ with ones along the diagonal of L. Show that such a decomposition of A is unique.