31 Lecture #21: Tuesday, April 28th, 2026

31.1 Systems of linear equations

In this section, we study systems of linear equations. Note that we are not dealing with *differential* equations, but with ordinary linear equations of the kind already familiar from elementary algebra (from school and from Linear Algebra). Our main interest is in the case of a large system of equations that we want to solve efficiently. A natural first step is to rewrite the system in matrix form, which allows us to apply systematic algebraic methods such as Gaussian elimination.

In practical applications, one often encounters very large systems of linear equations, far too large to solve by hand, so one relies on computer implementations of elimination methods; this is the realm of Numerical Analysis, where the goal is not only to obtain a solution, but also to do so efficiently and with attention to rounding errors and stability, unlike the exact hand computations usually performed in high school.

Before we begin the study of systems of linear equations, let us briefly recall how the elimination method is applied to a simple matrix. A **pivot** is the leading nonzero entry used at each step of Gaussian elimination to eliminate the entries below it in the same column. The method goes like this:

- Use elementary row operations to eliminate the entries below the first pivot.
- Move to the next column and repeat the elimination process below the next pivot.
- Continue until the matrix is in upper triangular form.

However, in Numerical Analysis, one **does not** usually employ all elementary row operations in Gaussian elimination in the same way as in hand computations. For a person working by hand, it is often convenient to scale a row so that the pivot becomes 1, but in numerical computations this is not always desirable, since such operations may destroy structures that are useful later, for instance in matrix factorizations. Instead, we typically rely on two basic operations. First, if necessary, we may interchange two rows in order to place a suitable entry in the pivot position. Second, once the pivot has been chosen, we eliminate the entries below it by subtracting from each lower row an appropriate multiple of the pivot row.

Example 31.1. Let us apply elimination in the following matrix by using just the allowed operations. First, we interchange the first and second rows in order to place a more convenient pivot and we denote it by $R_1 \leftrightarrow R_2$. Next, we eliminate the entry below the pivot in the first column. To do this, we replace the second row by $R_2 - 2R_1$, which makes the first entry of the second row equal to 0. The notation for this is $R_2 - 2R_1 \to R_2$. Finally, we eliminate the other entry below the same pivot. Since the first entry of the third row is -2, we replace the third row by $R_3 - (-2)R_1 = R_3 + 2R_1$, use the notation $R_3 - (-2)R_1 \to R_3$ and obtain another zero in the first column. Visually, we have the following steps (notice that what we are doing now is Step 2 in Algorithm 31.2 with $l_{2,1} = 2$ and $l_{3,1} = -2$)

$$\begin{pmatrix} 2 & -6 & -2 \\ 1 & 1 & 0 \\ -2 & 0 & 1 \end{pmatrix} \xrightarrow{R_1 \leftrightarrow R_2} \begin{pmatrix} 1 & 1 & 0 \\ 2 & -6 & -2 \\ -2 & 0 & 1 \end{pmatrix} \xrightarrow{R_2 - 2R_1 \to R_2} \begin{pmatrix} 1 & 1 & 0 \\ 0 & -8 & -2 \\ -2 & 0 & 1 \end{pmatrix} \xrightarrow{R_3 - (-2)R_1 \to R_3} \begin{pmatrix} 1 & 1 & 0 \\ 0 & -8 & -2 \\ 0 & 2 & 1 \end{pmatrix}.$$

We now focus on the submatrix formed by the last two rows and the last two columns. Since the entry 2 is more convenient than -8, we interchange R_2 and R_3 so that the new pivot in the second column is 2.

Next, we eliminate the entry below this pivot. Since $-8 = (-4) \cdot 2$ (in the second step of Algorithm 31.2, this means that $l_{3,2} = -4$), we replace the third row by $R_3 - (-4)R_2 = R_3 + 4R_2$, which makes the entry below the pivot equal to 0. In this way, the matrix is transformed into upper triangular form as follows

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & -8 & -2 \\ 0 & 2 & 1 \end{pmatrix} \overset{R_2 \leftrightarrow R_3}{\longrightarrow} \begin{pmatrix} 1 & 1 & 0 \\ 0 & 2 & 1 \\ 0 & -8 & -2 \end{pmatrix} \overset{R_3 - (-4)R_2 \to R_3}{\longrightarrow} \begin{pmatrix} 1 & 1 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 2 \end{pmatrix}.$$

If the matrix were larger than 3×3, we would continue by using the same ideas as before.

31.2 Gaussian elimination method: error and numerical stability

The algorithm of this method is given in the following lines.

Algorithm 31.2 (GEM: Gauss elimination method for reducing a full rank matrix to upper-triangular form, with partial pivoting)**.** Given a matrix $C = (c_{i,j})_{i,j=1}^{n,m}$ of real numbers, where $m \geqslant n$, we follow the steps.

0. Set $k = 1$.

1. If $c_{i,k} = 0$ for all $i = k, \ldots, n$, then $\operatorname{rank}(A) < n$. The algorithm fails and stops.

   Otherwise, do the "pivoting". Among the rows $i = k, \ldots, n$, choose the row k' that has the largest possible value of $|c_{i,k}|$. If $k' \neq k$, exchange rows k and k'. The (new) number $c_{k,k}$ is called a "pivot". Continue with step 2.

2. For $i = k+1, \ldots, n$ do the following:

   If $c_{i,k} \neq 0$, then let

   $$l_{i,k} = \frac{c_{i,k}}{c_{k,k}},$$

   set $c_{i,k} = 0$ and for $j > k$ replace $c_{i,j}$ with

   $$c_{i,j} - l_{i,k} c_{k,j}.$$

3. If $k < n$, increase k by one and go back to step 1.

   Otherwise the algorithm stops.

The output is the matrix $(c_{i,j})_{i,j=1}^{n,m}$.

Note: The output has the form

$$\begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,n-1} & c_{1,n} & \cdots & c_{1,m} \\ 0 & c_{2,2} & \cdots & c_{2,n-1} & c_{2,n} & \cdots & c_{2,m} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & c_{n-1,n-1} & c_{n-1,n} & \cdots & c_{n-1,m} \\ 0 & 0 & \cdots & 0 & c_{n,n} & \cdots & c_{n,m} \end{pmatrix}.$$

Gaussian elimination is closely related to the computation of the determinant: once the matrix has been reduced to upper triangular form, the determinant is obtained as the product of the diagonal entries, multiplied by -1 each time two rows have been interchanged during the elimination process. In Example 31.1, we have that

$$\det\begin{pmatrix} 2 & -6 & -2 \\ 1 & 1 & 0 \\ -2 & 0 & 1 \end{pmatrix} = \det\begin{pmatrix} 1 & 1 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 2 \end{pmatrix} = (-1)^2 \times (1 \cdot 2 \cdot 2) = 4.$$

This means that the determinant of the matrix in absolute value *does not change* after applying the Gauss elimination method.

In Example 31.1 no error was introduced, since all computations were carried out exactly and the method produced the desired upper triangular matrix. However, this does not mean that Gaussian elimination is generally error-free: in exact arithmetic it is an exact method, but in Numerical Analysis, when computations are performed in floating-point arithmetic, which is much closer to reality, rounding errors may still appear. For instance, if a coefficient such as $\sqrt{2}$ is replaced by a decimal approximation, say 1.4142, then each subsequent elimination step is performed with this inexact value rather than with the exact irrational number. As the computation proceeds, these small discrepancies may accumulate and even be amplified (see Example 31.3), especially when subtracting nearly equal numbers, which can lead to a catastrophic loss of accuracy. This shows that, without suitable precautions, the numerical stability of the method may be very poor.

Example 31.3. To illustrate the effect of rounding errors in practice, we performed a numerical experiment in Maple. We generated a random matrix A of size 100×100 with integer entries between -9 and 9, repeating the process until a full-rank matrix was obtained (see Figure 134). We then computed its determinant exactly and compared it with the determinant obtained indirectly from Gaussian elimination. Namely as the product of the diagonal entries of the upper triangular matrix produced by the elimination process as we have done before. Recall Step 2 in its algorithm which requires to calculate $l_{i,k} = \frac{c_{i,k}}{c_{k,k}}$; if the pivot $c_{k,k}$ is small, the multiplier $l_{i,k}$ can be large and then the update subtracts two potentially large, nearby equal numbers, so many significant digits can cancel. The resulting entry may have only a few trustworthy digits and later that bad entry can itself become a pivot, and so on.

```
> n:=100:
  A:=RandomMatrix(n,generator=-9..9):
  while Rank(A)<n do A:=RandomMatrix(n,generator=-9..9) end do:
  Ae:=MatrixEliminate(A,pivoting=euclidean):
  det:=abs(mul(Ae[k,k],k=1..n)):
  Ae:=MatrixEliminate(evalf(A),pivoting=minimal):
  abs((abs(mul(Ae[k,k],k=1..n))-abs(det))/det)/10^(1-Digits);
                                                    8309.107372
```

Figure 134: A 100×100 matrix on Maple and the huge error produced using the Gauss elimination method.

The numerical result obtained (and it took around 4 to 5 seconds to provide the result) was

$$8309.107372$$

which means that the error is more than 8000 times larger than the working precision. This is a very large amplification of roundoff errors.

Can we in fact control this imprecisions? Let us notice the following. Each time we use the pivot in row k to create a zero in row i, we do not only propagate the exact entry of the pivot row, but also the error already contained in it. Observe the matrix below. Thus, if the pivot row carries an error term E, then after applying the elimination factor $l_{i,k}$, this error is transferred to the i-th row as $-l_{i,k} \cdot E$. In this way, Gaussian elimination not only creates zeros, but may also propagate and amplify previously existing errors.

$$\begin{pmatrix} \cdots & \cdots & \cdots \\ \vdots & & \vdots \\ \text{arrow k} & \star_{pivot} & \bullet + E \\ \vdots & \vdots & \vdots \\ \text{arrow i} & \bullet_{possible\ 0} & \bullet - l_{i,k} \cdot E \\ \vdots & & \vdots \\ \cdots & \cdots & \cdots \end{pmatrix}$$

This means that, in order to reduce as much as possible the accumulation of errors, we would like the multipliers $l_{i,k}$ to be as small as possible. From Step 2 of Algorithm 31.2, we see that these multipliers are given by

$$l_{i,k} = \frac{c_{i,k}}{c_{k,k}}.$$

Is it possible to make these numbers as small as possible? Since $c_{k,k}$ is the pivot, this means that we should choose the pivots as large as possible in absolute value (see the first step in Algorithm 31.2). This procedure is called **partial pivoting**.

Example 31.4. We now reconsider the matrix

$$A = \begin{pmatrix} 2 & -6 & -2 \\ 1 & 1 & 0 \\ -2 & 0 & 1 \end{pmatrix}$$

and apply Gaussian elimination with partial pivoting, that is, at each step we choose as pivot the entry of largest absolute value in the current column among the rows below. In the first column, the entries are 2, 1, and -2, so the largest absolute value is 2. Thus, we may keep the first row as pivot row. We then eliminate the entries below it as follows

$$\begin{pmatrix} 2 & -6 & -2 \\ 1 & 1 & 0 \\ -2 & 0 & 1 \end{pmatrix} \xrightarrow{R_2 - \frac{1}{2}R_1 \to R_2} \begin{pmatrix} 2 & -6 & -2 \\ 0 & 4 & 1 \\ -2 & 0 & 1 \end{pmatrix} \xrightarrow{R_3 - (-1)R_1 \to R_3} \begin{pmatrix} 2 & -6 & -2 \\ 0 & 4 & 1 \\ 0 & -6 & -1 \end{pmatrix}.$$

Since $|-6| > |4|$, partial pivoting tells us to interchange the second and third rows and we get

$$\begin{pmatrix} 2 & -6 & -2 \\ 0 & 4 & 1 \\ 0 & -6 & -1 \end{pmatrix} \xrightarrow{R_2 \leftrightarrow R_3} \begin{pmatrix} 2 & -6 & -2 \\ 0 & -6 & -1 \\ 0 & 4 & 1 \end{pmatrix}.$$

Finally, we eliminate the entry below the new pivot

$$\begin{pmatrix} 2 & -6 & -2 \\ 0 & -6 & -1 \\ 0 & 4 & 1 \end{pmatrix} \xrightarrow{R_3 - \left(-\frac{2}{3}\right)R_2 \to R_3} \begin{pmatrix} 2 & -6 & -2 \\ 0 & -6 & -1 \\ 0 & 0 & \frac{1}{3} \end{pmatrix}.$$

Remark 31.5. In practice, carrying out partial pivoting by hand is often inconvenient, since it quickly leads to many fractions. For this reason, in written exercises or tests, students may instead be allowed to choose more convenient pivots, even if they are not the largest possible ones.

Example 31.6. We may also compare two different pivoting strategies through a numerical experiment in Maple (see Figure 135). As before, we generate a random matrix 100×100 with integer entries between -9 and 9, and we repeat the process until the matrix has full rank. First, we apply Gaussian elimination with standard pivoting and use the product of the diagonal entries of the resulting triangular matrix as a reference value for the determinant. Then we repeat the elimination twice: once with minimal pivoting and once with partial pivoting. In each case, we again approximate the determinant by the product of the diagonal entries. The quantity computed at the end compares the error obtained with minimal pivoting to the error obtained with partial pivoting. The result 35.5 shows that the error produced by minimal pivoting is 35.5 times larger than the error produced by partial pivoting. Since this experiment is based on a random generation of a 100×100 matrix, it can be repeated many times, and one consistently observes that this new approach has a better chance of producing smaller errors.

```
> n:=100:
  A:=RandomMatrix(n,generator=-9..9):
  while Rank(A)<n do A:=RandomMatrix(n,generator=-9..9) end do:
  Ae:=MatrixEliminate(A,pivoting=euclidean):
  det:=abs(mul(Ae[k,k],k=1..n)):
  A:=evalf(A):
  Ae:=MatrixEliminate(A,pivoting=minimal):
  err:=abs(mul(Ae[k,k],k=1..n))-det:
  Ae:=MatrixEliminate(A,pivoting=partial):
  abs(err/(abs(mul(Ae[k,k],k=1..n))-det));
                                                    35.50000000
```

Figure 135: Comparison of two pivoting strategies: minimal pivoting and partial pivoting.

31.3 Gaussian elimination method: computational complexity (speed)

We now turn to the computational cost of Gaussian elimination. The speed of the method is an important issue, since it gives a clearer idea of its practical performance on large systems. To illustrate this, the following table reports the average time required to apply Gaussian elimination to a random $n \times n$ matrix in Maple. The averages were obtained by generating several random matrices of size $n \times n$ and measuring the time T needed to perform the elimination. The results are summarized below.

n	T
100	3 sec
300	2 min
500	16 min
1000	5 hours

The table shows that the computational cost grows very rapidly as the size of the matrix increases. Even a moderate increase in n leads to a dramatic increase in the running time, which makes clear that Gaussian elimination becomes expensive quite quickly for large systems. In real-life applications, matrices may be far larger than the ones in the previous table; for instance, problems coming from discretizations in science or engineering may easily lead to systems of size $10^6 \times 10^6$. If we use the data above to make a rough

estimate, the growth appears to be approximately cubic in n. Thus, the required time would be roughly $5 \cdot 10^9$ hours, which is approximately

$$\frac{5 \cdot 10^9}{24 \cdot 365} \approx 5.7 \times 10^5$$

years. This means that Maple would need on the order of several hundred thousand years, which clearly shows that a direct use of Gaussian elimination becomes completely impractical at that scale.

In Numerical Analysis, the **computational complexity** of an algorithm measures the amount of computational work needed to execute it, usually as a function of the size of the input. This work is typically estimated by counting the number of elementary arithmetic operations performed by the method. We now ask how many such operations are required by Gaussian elimination. In other words, we want to determine the computational complexity of this method by answering how many basic operations one needs to get from the original matrix to the upper triangular form.

Typically, a general matrix has a squared matrix $n \times n$ as its core with some extra columns as the following matrix

$$\left(\begin{array}{cccc|c} a_{11} & \cdots & \cdots & a_{1n} & \\ \vdots & & & \vdots & \\ \vdots & & & \vdots & c \text{ columns more} \\ a_{n1} & \cdots & \cdots & a_{nn} & \end{array}\right)$$

We always assume that the $n \times n$ core of this matrix is **regular**, that is, it is invertible, or equivalently, its determinant is nonzero. Our aim is to apply Gaussian elimination to this $n \times n$ block in order to transform it into an upper triangular matrix.

Now let us assume that some elimination steps have already been carried out and that we still need to create k more zeros below the pivot $\star_{\text{pivot}}$. After completing this step, we are left with a $(k+1) \times (k+1)$ submatrix in which $\star_{\text{pivot}}$ is the leading pivot, as illustrated below.

$$\left(\begin{array}{cccccc|c} * & \cdots & \cdots & \cdots & \cdots & \cdots & \\ 0 & \ddots & & & & \vdots & \\ \vdots & & \star_{\text{pivot}} & \star_1 & \cdots & \star_k & \\ 0 & \cdots & \star_1 & \ddots & & \vdots & \\ \vdots & & \vdots & & \ddots & \vdots & c \text{ columns more} \\ 0 & \cdots & \vdots & \bullet & \cdots & \cdots & \\ \vdots & & \vdots & \vdots & & \vdots & \\ 0 & \cdots & \vdots & \bullet & \cdots & \cdots & \\ 0 & \cdots & \star_k & \bullet & \cdots & \cdots & \end{array}\right)$$

This is the general setting we shall consider. Our next question is: how many operations are required to create the k zeros below the pivot $\star_{\text{pivot}}$? Here we assume that no further row interchanges are performed, and we focus only on the operations needed to eliminate the entries below the pivot. Since there are k entries below the pivot, we need to perform k elimination steps. For each such step, we first compute the multiplier $l_{i,k}$, which requires one operation. Then, for each of the remaining $k+c$ entries in the row, we must update the entry according to the formula $c_{i,j} - l_{i,k}c_{k,j}$. Each such update involves one multiplication and one subtraction, hence $2(k+c)$ operations. Therefore, at this stage, the total number of operations is

$$k\Big(1 + 2(k+c)\Big).$$

These operations are usually called **flops** (floating-point operations). Summing over all elimination stages, we obtain

$$\sum_{k=1}^{n-1} k\Big(1+2(k+c)\Big) = \sum_{k=1}^{n-1}(k+2k^2+2ck) = 2\sum_{k=1}^{n-1} k^2 + \sum_{k=1}^{n-1} k + 2c\sum_{k=1}^{n-1} k.$$

Recall now that

$$\sum_{p=1}^{n} p = \frac{n(n+1)}{2}, \qquad \sum_{p=1}^{n} p^2 = \frac{n(n+1)(2n+1)}{6}.$$

Therefore,

$$\begin{aligned}
\sum_{k=1}^{n-1}(k+2k^2+2ck) &= 2\sum_{k=1}^{n-1} k^2 + \sum_{k=1}^{n-1} k + 2c\sum_{k=1}^{n-1} k \\
&= 2\cdot\frac{(n-1)n(2n-1)}{6} + \frac{(n-1)n}{2} + 2c\cdot\frac{(n-1)n}{2} \\
&= \frac{1}{3}(2n^3-3n^2+n) + \frac{1}{2}(n^2-n) + c(n^2-n) \\
&= \frac{2}{3}n^3 - \frac{1}{2}n^2 - \frac{1}{6}n + cn^2 - cn \\
&= \frac{2}{3}n^3 + \left(c-\frac{1}{2}\right)n^2 - \left(c+\frac{1}{6}\right)n.
\end{aligned}$$

Therefore, we have the following result.

Fact 31.7. Gaussian elimination applied to an $n \times (n+c)$ matrix requires at most

$$\frac{2}{3}n^3 + \left(c-\frac{1}{2}\right)n^2 - \left(c+\frac{1}{6}\right)n$$

operations.

In particular, when Gaussian elimination is applied to an $n \times n$ matrix, the dominant term in the number of operations is $\frac{2}{3}n^3$, and therefore the computational complexity is of order n^3. In other words, we have the following consequence of the previous result.

Corollary 31.8. The computational complexity of GEM when reducing an $n \times n$ matrix or an $n \times (n+1)$ matrix is

$$\frac{2}{3}n^3 + O(n^2).$$

Notice, however, that the leading term in the computational complexity is proportional to n^3, doubling the size of the matrix means replacing n by $2n$, and therefore the running time is multiplied approximately by $(2)^3 = 8$. Thus, if the size of the matrix is doubled, we should expect the algorithm to take about eight times longer to finish, which is quite a lot.

We then can say that

- to obtain numerically reliable results, Gaussian elimination should be performed with partial pivoting, since choosing pivots as large as possible in absolute value helps control the multipliers and reduces the propagation of rounding errors, and
- even with good numerical stability, Gaussian elimination can be very expensive for large matrices, because its computational complexity grows like n^3; thus, even on a very powerful computer, the running time may become prohibitively large.