32 Lecture #22: Wednesday, April 29th, 2026

32.1 The Gauss-Jordan elimination method

In this section, we study how elimination can be used to solve systems of linear equations. Consider the algebraic system $A\vec{x} = \vec{b}$. Our goal is to understand how Gaussian elimination can be applied to determine the solution of this system. In the usual school approach, one writes the system together with the right-hand side in the augmented matrix form

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

Then one applies elementary row operations (in fact, dividing the pivots until we reach the number 1, which is not what we want right now) until the left-hand side is reduced to the identity matrix. If this is possible, one obtains

$$\left(\begin{array}{cccc|c} 1 & 0 & \cdots & 0 & x_1^{(0)} \\ 0 & 1 & \cdots & 0 & x_2^{(0)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & x_n^{(0)} \end{array} \right)$$

from which we read the solution of the system directly as

$$\vec{x}_0 = \begin{pmatrix} x_1^{(0)} \\ x_2^{(0)} \\ \vdots \\ x_n^{(0)} \end{pmatrix}.$$

This is the **Gauss-Jordan elimination method** (GJM, for short). The algorithm is as follows.

Algorithm 32.1 (GJM: Gauss-Jordan elimination method for reducing a full-rank matrix to an extended diagonal matrix, with partial pivoting)**.** Given a matrix $C = (c_{i,j})_{i,j=1}^{n,m}$ of real numbers, where $m \geqslant n$, we have the following.

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 $j > k$ replace $c_{k,j}$ with

   $$\frac{c_{k,j}}{c_{k,k}}.$$

   Set $c_{k,k} = 1$.

   For all $i \neq k$ do the following:

   If $c_{i,k} \neq 0$, then for $j > k$ replace $c_{i,j}$ with

   $$c_{i,j} - c_{i,k}c_{k,j};$$

   then set $c_{i,k} = 0$.

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

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

However, if we do the computation complexity of GJM we get the following result.

Fact 32.2. Gauss–Jordan elimination applied to an $n \times (n + c)$ matrix requires at most

$$n^3 + \frac{1}{2}(4c - 3)n^2 + \frac{1}{2}(1 - 2c)n$$

operations.

Corollary 32.3. The computational complexity of GJM when reducing an $n \times n$ matrix or an $n \times (n+1)$ matrix is

$$n^3 + O(n^2).$$

In Numerical Analysis, therefore, this is not the preferred approach. Reducing the whole matrix all the way to the identity requires too many operations, and, as we have seen, the computational cost is of order n^3 as we have seen in Corollary 32.3. For large systems, this may lead to a very long running time, so this direct school-style procedure is not practical. We therefore need a more efficient approach for solving the system.

32.2 The back substitution method

Instead of applying Gauss–Jordan elimination, we use Gaussian elimination on the matrix $(A \mid \vec{b})$. This reduces the computational cost to about $\frac{2}{3}n^3$ operations, which is significantly less than the cost of the school-style approach. In this way, we transform the system into one of the form $(U \mid \vec{y})$, where U is upper triangular. However, the solution is no longer obtained directly from the right-hand side, so an additional step is still required.

Example 32.4. Consider a system in a matrix form

$$\left(\begin{array}{ccc|c} 2 & -6 & -2 & 0 \\ 1 & 1 & 0 & 1 \\ -2 & 0 & 1 & -1 \end{array}\right).$$

Notice that the left-hand side of the matrix is the matrix which appears in Example 31.1. We already know how it turns out to be an upper triangle matrix. By using the same operations with the rows and columns on the right-hand side, we get that

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

that is,

$$\begin{cases} x + y = 1, \\ 2y + z = 1, \\ 2z = 2. \end{cases}$$

We now solve by back substitution. From the third equation $z = 1$ and here we need one operation, that is, one flop. Substituting into the second equation, we get

$$y = \frac{1}{2} \cdot (1 - 1) = 0$$

and here we need 3 flops. Finally, from the first equation,

$$x = \frac{1}{1}(1 - 1 \cdot 0 - 0 \cdot 1) = 1$$

and here we need 5 flops.

This algorithm is called **back substitution** and we get see its algorithm in what follows.

Algorithm 32.5 (Solving an upper triangular system by back substitution)**.** Given a system $U\vec{x} = \vec{d}$, where the matrix U is square, regular and upper triangular, we find the solution $\vec{x}_0$ using the formulas

$$x_n = \frac{d_n}{u_{n,n}}$$

$$x_{n-1} = \frac{d_{n-1} - u_{n-1,n}x_n}{u_{n-1,n-1}}$$

$$x_{n-2} = \frac{d_{n-2} - u_{n-2,n}x_n - u_{n-2,n-1}x_{n-1}}{u_{n-2,n-2}}$$

$$\vdots$$

$$x_1 = \frac{d_1 - u_{1,n}x_n - u_{1,n-1}x_{n-1} - \cdots - u_{1,2}x_2}{u_{1,1}}$$

In general, for $k = n, n-1, \ldots, 1$ we calculate

$$x_k = \frac{1}{u_{k,k}} \left(d_k - \sum_{i=k+1}^{n} u_{k,i}x_i \right).$$

We can do the same by forward substitution as one can imagine.

Algorithm 32.6 (Solving a lower triangular system by forward substitution)**.** Given a system $L\vec{x} = \vec{d}$, where the matrix L is square, regular and lower triangular, we find the solution $\vec{x}_0$ using the formulas

$$x_1 = \frac{d_1}{l_{1,1}}$$

$$x_2 = \frac{d_2 - l_{2,1}x_1}{l_{2,2}}$$

$$x_3 = \frac{d_3 - l_{3,1}x_1 - l_{3,2}x_2}{l_{3,3}}$$

$$\vdots$$

$$x_n = \frac{d_n - l_{n,1}x_1 - l_{n,2}x_2 - \cdots - l_{n,n-1}x_{n-1}}{l_{n,n}}$$

In general, for $k = 1, 2, \ldots, n$ we calculate

$$x_k = \frac{1}{l_{k,k}} \left(d_k - \sum_{i=1}^{k-1} l_{k,i}x_i \right).$$

How much work does this require? In Example 32.4, we used $1 + 3 + 5$ flops in total. In general, for a system of size n, the cost of back substitution is obtained by summing the first n odd numbers:

$$1 + 3 + 5 + \cdots + (2n - 1) = n^2.$$

Therefore, back substitution requires on the order of n^2 operations, which is negligible compared with the $\frac{2}{3}n^3$ operations required by Gaussian elimination. This shows that solving the upper triangular system is relatively cheap, and that this approach is still much faster than the school-style method.

Fact 32.7. Any system $A\vec{x} = \vec{b}$ whose regular matrix A is upper (resp. lower) triangular can be solved by back (resp. forward) substitution with computational complexity n^2.

Example 32.8. In the Maple computation in Figure 136, we define the following matrix and the vector

$$A = \begin{pmatrix} 1 & -1 & 2 & -2 \\ 1 & -1 & 2 & 11 \\ -1 & 0 & -1 & 2 \\ 1 & -1 & 3 & -2 \end{pmatrix} \quad \text{and} \quad \vec{b} = \begin{pmatrix} -4 \\ 9 \\ 2 \\ -6 \end{pmatrix}.$$

We then solve the system $A\vec{x} = \vec{b}$ in two different ways. First, we apply Gauss-Jordan elimination to the matrix $(A \mid \vec{b})$. Maple reduces it directly to the form

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

from which the solution can be read immediately and it is given by

$$\vec{x} = \begin{pmatrix} 2 \\ 0 \\ -2 \\ 1 \end{pmatrix}.$$

Next, we apply Gaussian elimination to $(A \mid \vec{b})$. This produces an upper triangular system of the form

$$\left(\begin{array}{cccc|c} 1 & -1 & 2 & -2 & -4 \\ 0 & -1 & 1 & 0 & -2 \\ 0 & 0 & 1 & 0 & -2 \\ 0 & 0 & 0 & 13 & 13 \end{array} \right).$$

At this stage, the solution is not yet visible directly, so Maple applies back substitution and it gives once again $\vec{x}$ as before. Therefore, both methods produce the same solution, as expected. The difference is that Gauss-Jordan elimination gives the answer directly by reducing the matrix to the identity form, whereas Gaussian elimination stops at an upper triangular system and then requires an additional back substitution step.

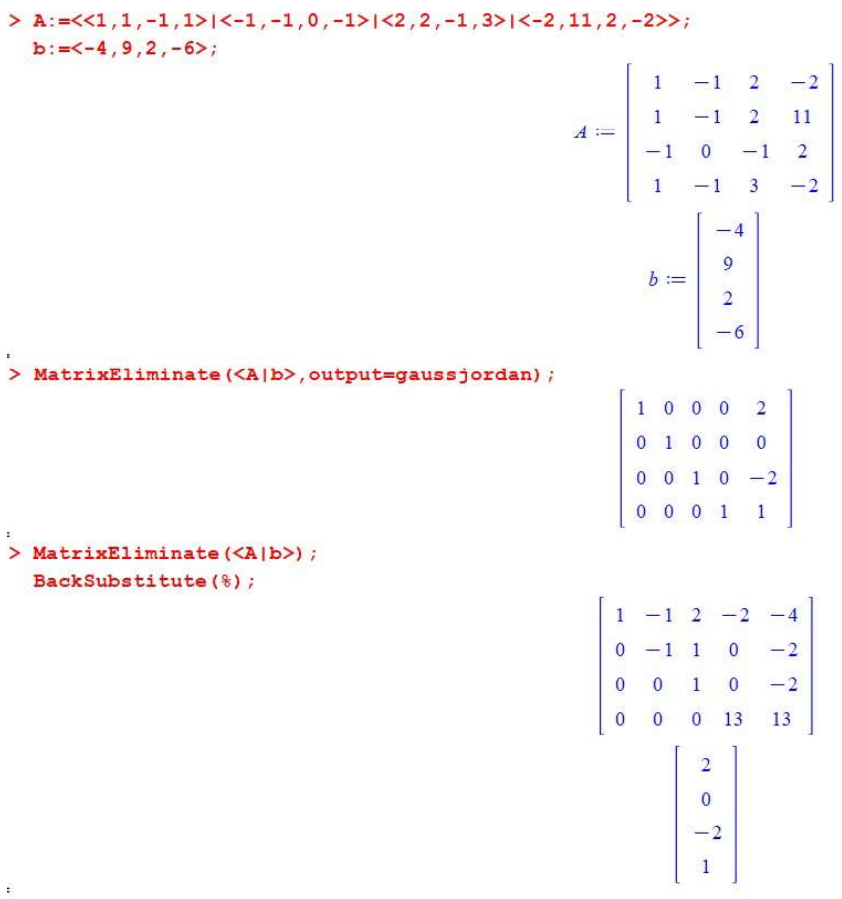

Figure 136: Gauss-Jordan elimination and Gaussian elimination methods.

Example 32.9. In the Maple experiment in Figure 137, we compare the running times of Gauss-Jordan elimination and Gaussian elimination on a random dense system of size 150×150. More precisely, we generate a random regular matrix A and a random vector $\vec{b}$, both with entries between -9.9 and 9.9, and then solve the system $A\vec{x} = \vec{b}$. First, we apply Gauss-Jordan elimination directly to the matrix $(A \mid \vec{b})$. Maple reports a running time of 5.984 seconds. Next, we apply Gaussian elimination to the same matrix. This produces an upper triangular system and Maple reports a running time of 2.407 seconds. After that, we solve the resulting triangular system by back substitution, which takes only 0.109 seconds. Therefore, the total time needed by Gaussian elimination together with back substitution is only

$$2.407 + 0.109 = 2.516 \text{ seconds}$$

which is still much smaller than the 5.984 seconds required by Gauss-Jordan elimination.

```
> n:=150:
  A:=RandomMatrix(n,generator=-9.9..9.9,density=1):
  while Rank(A)<n do A:=RandomMatrix(n,generator=-9.9..9.9,density=1) end do:
  b:=RandomVector(n,generator=-9.9..9.9,density=1):start:=time():Ael:=MatrixEliminate(<A|b>,output=
  gaussjordan):
  printf("GJM: % .3f\n",time()-start):
  start:=time():Ael:=MatrixEliminate(<A|b>):printf("GEM: % .3f  ",time()-start):
  start:=time():solG:=BackSubstitute(Ael):printf("BS:  % .3f\n",time()-start):
GJM:  5.984
GEM:  2.407  BS:   0.109
```

Figure 137: Speed comparison between the GEM and GJM for 150×150 matrices.

Repeating the same experiment for a random dense system of size 250×250, Maple gives a running time of 23.875 seconds for Gauss-Jordan elimination, while Gaussian elimination takes 13.360 seconds and the subsequent back substitution only 0.328 seconds. Hence, Gaussian elimination followed by back substitution remains clearly faster than Gauss-Jordan elimination, in agreement with the theoretical complexity estimates.

```
> n:=250:
  A:=RandomMatrix(n,generator=-9.9..9.9,density=1):
  while Rank(A)<n do A:=RandomMatrix(n,generator=-9.9..9.9,density=1) end do:
  b:=RandomVector(n,generator=-9.9..9.9,density=1):start:=time():Ael:=MatrixEliminate(<A|b>,output=
  gaussjordan):
  printf("GJM: % .3f\n",time()-start):
  start:=time():Ael:=MatrixEliminate(<A|b>):printf("GEM: % .3f  ",time()-start):
  start:=time():solG:=BackSubstitute(Ael):printf("BS:  % .3f\n",time()-start):
GJM:  23.875
GEM:  13.360  BS:   0.328
```

Figure 138: Speed comparison between the GEM and GJM for 250×250 matrices.