8.2 Gauss elimination

Concepts

Gauss elimination
Downward pass
Upper triangular systems
Upward pass
Square nonsingular systems have unique solutions
Efficiency: counting arithmetic operations

The main algorithm considered in this chapter for solving linear systems $A\xi = \beta$ is called *Gauss elimination*. Its basic strategy is to replace the original system step by step with equivalent simpler ones until the resulting system can be analyzed very easily. Two systems are called *equivalent* if they have the same sets of solution vectors ξ. Just two kinds of operations are used to produce the simpler systems:

I) interchange two equations;
II) subtract from one equation a scalar multiple of another.

Obviously, operation (I) doesn't change the set of solution vectors: it produces an equivalent system. Here's an application of operation (II), from the example in Section 8.1:

$$\begin{cases} x_1 + 2x_2 = 3 & [1] \\ 4x_1 + 5x_2 = 6 & [2] \end{cases}$$

$$\begin{cases} x_1 + 2x_2 = 3 & [3] \\ -3x_2 = -6 & [4] = [2] - 4\cdot[1]\ . \end{cases}$$

In general, this operation has the following appearance:

$$\begin{cases} a_{i1}x_1 + \cdots + a_{in}x_n = b_i & [1] \\ a_{j1}x_1 + \cdots + a_{jn}x_n = b_j & [2] \end{cases}$$

$$(a_{j1} - ca_{i1})x_1 + \cdots + (a_{jn} - ca_{in})x_n = b_j - cb_i \qquad [3] = [2] - c\cdot[1] \ .$$

Clearly, any solution $x_1, \ldots, x_n$ of [1] and [2] satisfies [3]. On the other hand, [2] = [3] + $c\cdot$[1], hence any solution of [1] and [3] also satisfies [2]. Thus, operation (II) doesn't change the solution set; it produces an equivalent system.

Downward pass

The first steps of Gauss elimination, called the *downward pass*, convert the original system

$$\begin{cases} a_{11}x_1 + \cdots + a_{1n}x_n = b_1 \\ \quad\vdots \qquad\qquad \vdots \quad \vdots \\ a_{m1}x_1 + \cdots + a_{mn}x_n = b_m \end{cases}$$

into an equivalent upper triangular system

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1,m-1}x_{m-1} + a_{1,m}x_m + \cdots + a_{1,n}x_n = b_1 \\ \qquad a_{22}x_2 + \cdots + a_{2,m-1}x_{m-1} + a_{2,m}x_m + \cdots + a_{2,n}x_n = b_2 \\ \qquad\qquad\qquad\qquad \vdots \qquad\qquad \vdots \qquad\qquad \vdots \quad \vdots \\ \qquad\qquad\qquad a_{m-1,m-1}x_{m-1} + a_{m-1,m}x_m + \cdots + a_{m-1,n}x_n = b_{m-1} \\ \qquad\qquad\qquad\qquad\qquad\qquad a_{m,m}x_m + \cdots + a_{m,n}x_n = b_m \end{cases} .$$

A linear system is called *upper triangular* if $a_{ij} = 0$ whenever $i > j$.

Figure 8.2.1 contains pseudocode for the downward pass. The algorithm considers in turn the diagonal coefficients $a_{1,1}, \ldots, a_{n-1,n-1}$, called *pivots*. Lines (2) to (4) ensure that each pivot is nonzero, if possible. That is, if a diagonal coefficient is zero, you search

downward for a nonzero coefficient; if you find one, you interchange rows to make it the pivot. If you don't, then you proceed to the next equation. Nonzero pivots a_{kk} are used in Lines (6) and (7) to eliminate the unknown x_k from all equations below Equation k. This process clearly produces an equivalent upper triangular system.

```
1)  for (k = 1; k < min(m,n); ++k) {
2)    for (i = k; a_ik == 0 && i ≤ m; ++i);
3)    if (i ≤ m) {
4)      if (i > k) { interchange Equations [i] and [k]; }
5)      for (i = k + 1; i ≤ m; ++i) {
6)        M = a_ik/a_kk;
7)        subtract M times Equation [k] from Equation [i]; }}}
```

Figure 8.2.1 Pseudocode for the downward pass

Here's an example downward pass to convert a system of five equations in six unknowns $x_1, \ldots, x_6$ into an equivalent upper triangular system:

$$\begin{cases} 7x_1 + 3x_2 + 8x_3 - 15x_5 - 12x_6 = -18 & [1] \\ 7x_1 + 3x_2 + 8x_3 - 2x_4 - 10x_5 - 9x_6 = -15 & [2] \\ -14x_1 - 6x_2 - 16x_3 - 4x_4 + 30x_5 + 18x_6 = 28 & [3] \\ -7x_1 - 6x_3 - 2x_4 + 15x_5 + 10x_6 = 13 & [4] \\ 14x_1 + 3x_2 + 14x_3 + 4x_4 - 40x_5 - 31x_6 = -41 & [5] \end{cases}$$

$$\begin{cases} 7x_1 + 3x_2 + 8x_3 - 15x_5 - 12x_6 = -18 & [6] = [1] \\ - 2x_4 + 5x_5 + 3x_6 = 3 & [7] = [2] - [1] \\ - 4x_4 - 6x_6 = -8 & [8] = [3] + 2\cdot[1] \\ 3x_2 + 2x_3 - 2x_4 - 2x_6 = -5 & [9] = [4] + [1] \\ -3x_2 - 2x_3 + 4x_4 - 10x_5 - 7x_6 = -5 & [10] = [5] - 2\cdot[1] \end{cases}$$

$$
\left\{\begin{array}{rrrrrrll}
7x_1 + 3x_2 + 8x_3 & & & -15x_5 - 12x_6 & = & -18 & [11] = [6] \\
& 3x_2 + 2x_3 - 2x_4 & & - 2x_6 & = & - 5 & [12] = [9] \\
& & - 4x_4 & - 6x_6 & = & - 8 & [13] = [8] \\
& & - 2x_4 & + 5x_5 + 3x_6 & = & 3 & [14] = [7] \\
& - 3x_2 - 2x_3 + 4x_4 & & - 10x_5 - 7x_6 & = & - 5 & [15] = [10] \\
7x_1 + 3x_2 + 8x_3 & & & - 15x_5 - 12x_6 & = & -18 & [16] = [11] \\
& 3x_2 + 2x_3 - 2x_4 & & - 2x_6 & = & - 5 & [17] = [12] \\
& & - 4x_4 & - 6x_6 & = & - 8 & [18] = [13] \\
& & - 2x_4 & + 5x_5 + 3x_6 & = & 3 & [19] = [14] \\
& & 2x_4 & - 10x_5 - 9x_6 & = & -10 & [20] = [15] + [12] \\
7x_1 + 3x_2 + 8x_3 & & & - 15x_5 - 12x_6 & = & -18 & [21] = [16] \\
& 3x_2 + 2x_3 - 2x_4 & & - 2x_6 & = & - 5 & [22] = [17] \\
& & - 4x_4 & - 6x_6 & = & - 8 & [23] = [18] \\
& & - 2x_4 & + 5x_5 + 3x_6 & = & 3 & [24] = [19] \\
& & & - 5x_5 - 6x_6 & = & - 7 & [25] = [20] + [19] .
\end{array}\right.
$$

You can assess this algorithm's efficiency by counting the number of scalar arithmetic operations it requires. They're all in Lines (6) and (7) of the pseudocode in Figure 8.2.1. When you execute these for particular values of k and i, you need one division to evaluate M, then $n - k + 1$ subtractions and as many multiplications to subtract M times Equation $[k]$ from Equation $[i]$. (While the equations have $n + 1$ coefficients, you know the first k coefficients of the result of the subtraction are zero.) Thus the total number of scalar operations is

$$
\sum_{k=1}^{m-1} \sum_{i=k+1}^{m} \left[1 + 2(n - k + 1)\right] = \sum_{k=1}^{m-1} (m - k)(2n - 2k + 3)
$$

$$
= \sum_{k=1}^{m-1} \left[m(2n + 3) - (2m + 2n + 3)k + 2k^2\right]
$$

$$
= m(m - 1)(2n + 3) - (2m + 2n + 3)\sum_{k=1}^{m-1} k + 2\sum_{k=1}^{m-1} k^2
$$

$$= m(m-1)(2n+3) - (2m+2n+3)\frac{m(m-1)}{2} + 2\frac{m(m-1)(2m-1)}{6}$$

$$= m^2n - \tfrac{1}{3}m^3 + \text{lower-order terms}.$$

In particular, for large $m = n$, about $\frac{2}{3}n^3$ operations are required.

This operation count has two major consequences. First, solving large linear systems can require an excessive amount of time. For example, computing the steady state temperature distribution for the problem at the beginning of this section required solving a system of 25 equations, one for each interior cell. The result will be only a coarse approximation, because each cell is one square cm. To double the resolution—to use cells with a 0.5-cm edge—requires four times as many cells and equations, hence $4^3 = 64$ times as many operations. Second, solving large linear systems can require a huge number of scalar operations, each of which depends on the preceding results. This can produce excessive round-off error. For example, computing the temperature distribution just mentioned for a 0.5 cm. grid requires about 700,000 operations.

These large numbers justify spending considerable effort to minimize the use of Gauss elimination in numerical analysis applications. Unfortunately, there are few alternatives. Linear systems of special form—where nonzero coefficients are rare and occur in symmetric patterns—can sometimes be solved by special algorithms that are more efficient. For systems of general form there are algorithms that are somewhat more efficient than Gauss elimination, but only for extremely large systems. They are considerably more difficult to implement in software. Thus, round-off error in solving linear systems is often unavoidable. This subject has been studied in detail, particularly by Wilkinson [55].

Upward pass for nonsingular square systems

When you apply Gauss elimination to a system of m linear equations in n unknowns $x_1, \ldots, x_n$, the downward pass always yields an equivalent upper triangular system. This system may have a unique solution, infinitely many, or none at all. In one situation, you can easily determine a unique solution: namely, when $m = n$ and none of the diagonal coefficients of the upper triangular system is zero. Such a system is called *nonsingular*. For example, if the upper triangular system has the form

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 = b_1 & [1] \\ a_{22}x_2 + a_{23}x_3 + a_{24}x_4 = b_2 & [2] \\ a_{33}x_3 + a_{34}x_4 = b_3 & [3] \\ a_{44}x_4 = b_4 & [4] \end{cases}$$

with $a_{11}, a_{22}, a_{33}, a_{44} \neq 0$, then you can solve [4] for x_4 and substitute this value into [3], then solve that equation for x_3. You can substitute the x_3 and x_4 values into Equation [2], and solve that for x_2. Finally, Equation [1] would yield a value for x_1. This process is called the *upward pass* of Gauss elimination. In pseudocode, it has the form

$$\text{for } (k = n;\ k \geq 1;\ --k)$$

$$x_k = \frac{b_k - \sum_{j=k+1}^{n} a_{kj} x_j.}{a_{kk}}$$

For $k = n$ to 1, the assignment statement in the upward pass requires $n - k$ subtractions, $n - k$ multiplications, and one division. Thus, the total number of scalar arithmetic operations required is

$$\sum_{k=1}^{n} \left[2(n-k)+1\right] = 2\sum_{k=1}^{n}(n-k) + n = 2\sum_{k=0}^{n-1} k + n = 2\frac{(n-1)n}{2} + n$$
$$= n^2 + \text{lower order terms.}$$

Notice two facts about nonsingular square systems. First, their solutions are unique: each iteration of the for-loop in the upward pass pseudocode *determines* the value of one entry of the solution vector. Second, the nonsingularity criterion (no zero among the diagonal coefficients of the upper triangular system) doesn't involve the right-hand sides of the equations. Here's a summary of the preceding discussion of the downward and upward passes:

Consider a square linear system

$$\begin{cases} a_{11}x_1 + \cdots + a_{1n}x_n = b_1 \\ \quad\vdots \qquad\qquad\quad \vdots \quad \vdots \\ a_{n1}x_1 + \cdots + a_{nn}x_n = b_n \end{cases},$$

that is, $A\xi = \beta$ with

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \qquad \xi = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad \beta = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

Suppose the downward pass of Gauss elimination yields an upper triangular system

$$\begin{cases} a'_{11}x_1 + \cdots + a'_{1n}x_n = b'_1 \\ \qquad\qquad \ddots \qquad\quad \vdots \quad\; \vdots \\ \qquad\qquad\qquad\; a'_{nn}x_n = b'_n \end{cases}$$

with no zero among the diagonal coefficients a'_{kk}. Then this situation will occur for *any* coefficient vector β on the right hand side, and each of these systems has a unique solution ξ. Each solution may be computed by the upward pass, executed after the downward. For large n, the downward pass requires about $\frac{2}{3}n^3$ scalar arithmetic operations; the upward pass requires about n^2.