4 Gaussian Elimination

4.1 Simultaneous linear equations

Consider the system of linear equations

$$\left.\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & = & b_2 \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ a_{n1}x_1 & + & a_{n2}x_2 & + & \cdots & + & a_{nn}x_n & = & b_n \end{array}\right\} \tag{4.1}$$

which is to be solved for $x_1, \ldots, x_n$. Notice that the number of equations, n, is equal to the number of unknowns, so it is reasonable to expect that there should usually be a unique solution. Such systems arise in many different applications, and it is important to have an efficient method of solution, especially when n is large.

The system (4.1) can be written more succinctly, using the summation convention, as

$$a_{ij}x_j = b_i,$$

or alternatively, using matrix notation, as

$$\boldsymbol{Ax} = \boldsymbol{b}, \tag{4.2}$$

where $\boldsymbol{A}$ is an $n \times n$ matrix with entries $\{a_{ij}\}$, $\boldsymbol{x} = (x_1, \ldots, x_n)^T$ and $\boldsymbol{b} = (b_1, \ldots, b_n)^T$. If $\boldsymbol{A}$ is invertible, we can formally multiply (4.2) through by $\boldsymbol{A}^{-1}$ to obtain

$$\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}.$$

Example 4.1 *The system*

$$\begin{array}{ccccccc} -x_1 & + & x_2 & + & 2x_3 & = & 1, \\ 3x_1 & - & x_2 & + & x_3 & = & 1, \\ -x_1 & + & 3x_2 & + & 4x_3 & = & 1, \end{array} \tag{4.3}$$

may be written as $\boldsymbol{Ax} = \boldsymbol{b}$, *where*

$$\boldsymbol{A} = \begin{pmatrix} -1 & 1 & 2 \\ 3 & -1 & 1 \\ -1 & 3 & 4 \end{pmatrix}, \quad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

The inverse of this $\boldsymbol{A}$ *was found in* §2. *The solution may thus be determined via*

$$\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b} = \frac{1}{10}\begin{pmatrix} -7 & 2 & 3 \\ -13 & -2 & 7 \\ 8 & 2 & -2 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \frac{1}{10}\begin{pmatrix} -2 \\ -8 \\ 8 \end{pmatrix},$$

which gives $x_1 = -1/5$, $x_2 = -4/5$, $x_3 = 4/5$.

The problem with this approach is that, when n is large, it is extremely time-consuming to calculate $\boldsymbol{A}^{-1}$ using determinants. In this section we discuss the method of Gaussian elimination, which provides a much more efficient algorithm for solving systems like (4.1).

4.2 Doing it by hand

In practice, one would go about solving a system like (4.3) by eliminating the variables one at a time until just one remains. Then the other variables would be determined by back-substitution. Gaussian elimination is a formal procedure for doing this, which we illustrate with an example.

Example 4.2 *Consider again the system (4.3). We eliminate the variables one at a time as follows.*

1. *Eliminate x_1 from the second and third equations by subtracting suitable multiples of the first equation (-3 and 1 respectively). This results in the new system*

$$\begin{array}{rcrcrcl} -x_1 & + & x_2 & + & 2x_3 & = & 1, \\ & & 2x_2 & + & 7x_3 & = & 4, \\ & & 2x_2 & + & 2x_3 & = & 0. \end{array} \tag{4.4}$$

2. *Subtract a suitable multiple (here 1) of the second equation from the third to eliminate x_2:*

$$\begin{array}{rcrcrcl} -x_1 & + & x_2 & + & 2x_3 & = & 1, \\ & & 2x_2 & + & 7x_3 & = & 4, \\ & & & - & 5x_3 & = & -4. \end{array} \tag{4.5}$$

3. *Now we can solve by* back-substitution. *The third equation gives $x_3 = 4/5$; then the second gives $2x_2 = 4 - 7 \times 4/5 \Rightarrow x_2 = -4/5$. Finally, from the first equation, $-x_1 = 1 - (-4/5) - 2 \times 4/5 \Rightarrow x_1 = -1/5$.*

The elimination process just described is performed by applying so-called *elementary operations* to the equations. These comprise

1. swapping two equations;

2. multiplying an equation by a nonzero constant;

3. adding a multiple of one equation to another equation.

The crucial point is that, when any combination of these operations is performed upon a system of equations, the result is an *equivalent* system, in the sense that it has the same set of solutions. Thus the systems (4.3), (4.4) and (4.5) are all equivalent. The idea of Gaussian elimination is to use elementary operations to reduce the original system to an equivalent one which is in "triangular" form, and can then readily be solved by back-substitution.

4.3 The augmented matrix; elementary row operations

In practice, when carrying out this procedure on a general matrix equation of the form (4.2), it is useful to define the so-called *augmented matrix* $\widetilde{\boldsymbol{A}}$, which consists of $\boldsymbol{A}$ with the right-hand side $\boldsymbol{b}$ tacked on as an extra column:

$$\widetilde{\boldsymbol{A}} = (\boldsymbol{A}\,\boldsymbol{b}).$$

Thus each n-dimensional system of the form (4.1) corresponds to an $n\times(n+1)$ augmented matrix, and to each elementary operation, which may be performed upon a system of equations, corresponds an *elementary row operation*, which may be performed upon the augmented matrix $\widetilde{\boldsymbol{A}}$:

1. swapping two rows;
2. multiplying a row by a nonzero constant;
3. adding a multiple of one row to another row.

In Gaussian elimination we perform a combination of these operations such as to reduce the augmented matrix to a triangular form, known as *echelon form.*

Example 4.3 *Consider the system*

$$\begin{array}{rcrcrcl} 3x_1 & - & 4x_2 & + & 5x_3 & = & -1, \\ -3x_1 & + & 2x_2 & + & x_3 & = & 1, \\ 6x_1 & + & 8x_2 & - & x_3 & = & 35, \end{array}$$

whose corresponding augmented matrix is

$$\begin{pmatrix} \boxed{3} & -4 & 5 & -1 \\ -3 & 2 & 1 & 1 \\ 6 & 8 & -1 & 35 \end{pmatrix}.$$

The first operation is to use the first equation to eliminate x_1 from the other two. This corresponds to subtracting appropriate multiples of row one from rows two and three so as to eliminate their first column. The boxed number 3 is called the pivot*: it must be divided into the first entry of each subsequent row to determine the appropriate multiple of row one to be subtracted. Here $-3/3 = -1$ and $6/3 = 2$, so we subtract $(-1)\times$(row one) from (row two) and $2\times$(row one) from (row three), resulting in*

$$\begin{pmatrix} 3 & -4 & 5 & -1 \\ 0 & \boxed{-2} & 6 & 0 \\ 0 & 16 & -11 & 37 \end{pmatrix}.$$

Next we use row two to eliminate the second element from row three. Here the pivot is -2; $16/(-2) = -8$ so we subtract $(-8)\times$(row two) from (row three) two obtain

$$\begin{pmatrix} 3 & -4 & 5 & -1 \\ 0 & -2 & 6 & 0 \\ 0 & 0 & 37 & 37 \end{pmatrix}.$$

This is now in echelon form which completes the Gaussian elimination. It remains to perform back substitution [**exercise**] *which gives the solution $x_3 = 1$, $x_2 = 3$, $x_1 = 2$.*

Let us reiterate the process just described:

1. use multiples of row 1 to eliminate column 1 from rows 2, 3, … ;
2. use multiples of row 2 to eliminate column 2 from rows 3, … ;
3. … and so on.

Note once more the role of the pivot in determining the right multiple to be subtracted at each stage. In particular, *the pivot must be nonzero* for the process to work.

4.4 Pivoting

Example 4.4 *The system*

$$\begin{array}{rcrcrcr} x_1 & + & x_2 & + & x_3 & = & -1, \\ 2x_1 & + & 2x_2 & + & 5x_3 & = & -8, \\ 4x_1 & + & 6x_2 & + & 8x_3 & = & -14, \end{array}$$

corresponds to the augmented matrix

$$\begin{pmatrix} \boxed{1} & 1 & 1 & -1 \\ 2 & 2 & 5 & -8 \\ 4 & 6 & 8 & -14 \end{pmatrix}.$$

First use the pivot 1 *to clear out column one:*

$$\begin{pmatrix} 1 & 1 & 1 & -1 \\ 0 & \boxed{0} & 3 & -6 \\ 0 & 2 & 4 & -10 \end{pmatrix}.$$

Now we appear to be in trouble, since 0 *cannot be used as a pivot: it does not allow us to clear out the row below. The remedy to this difficulty is simply to switch rows* 2 *and* 3*:*

$$\begin{pmatrix} 1 & 1 & 1 & -1 \\ 0 & \boxed{2} & 4 & -10 \\ 0 & 0 & 3 & -6 \end{pmatrix}.$$

Now the pivot is nonzero, and indeed the matrix is in echelon form and ready for back-substitution.

The process of swapping two rows as just described is called *pivoting.* This suggests the following modification of the procedure outlined above:

1. check that the first pivot is nonzero;
   (a) if it *is* zero, find a row beneath whose first entry is nonzero and swap it with row 1;
2. use multiples of row 1 to eliminate column 1 from rows 2, 3, … ;
3. check that the second pivot is nonzero;

   (a) if it *is* zero, find a row beneath whose second entry is nonzero and swap it with row 2;

4. use multiples of row 2 to eliminate column 2 from rows 3, . . . ;

5. . . . and so on.

4.5 When pivoting fails

Notice that the procedure outlined above fails if we encounter a zero entry in the pivot position, but there is nothing nonzero with which to swap it.

Example 4.5 *The system*

$$\begin{array}{rcrcrcr} x_1 & + & x_2 & + & x_3 & = & -1, \\ 2x_1 & + & 2x_2 & + & 5x_3 & = & -8, \\ 4x_1 & + & 4x_2 & + & 8x_3 & = & -14, \end{array} \tag{4.6}$$

corresponds to the augmented matrix

$$\begin{pmatrix} \boxed{1} & 1 & 1 & -1 \\ 2 & 2 & 5 & -8 \\ 4 & 4 & 8 & -14 \end{pmatrix}.$$

As before, begin by using the first pivot, 1*, to eliminate the first column:*

$$\begin{pmatrix} 1 & 1 & 1 & -1 \\ 0 & \boxed{0} & 3 & -6 \\ 0 & 0 & 4 & -10 \end{pmatrix}. \tag{4.7}$$

Now the entry in the pivot position is zero, and the situation cannot be remedied by pivoting.

In fact the system (4.6) has *no solution*; the final two equations in (4.7) read $3x_3 = -6$ and $4x_3 = -10$, and are therefore inconsistent. In general, if no nonzero pivot can be found, this corresponds to the original matrix $\boldsymbol{A}$ being singular, *i.e.* $|\boldsymbol{A}| = 0$.

As an aside, we should point out that $\boldsymbol{A}$ being singular does not necessarily imply that $\boldsymbol{Ax} = \boldsymbol{b}$ has no solution.

Example 4.6 *If we adjust the right-hand side of* (4.6) *by changing the* -14 *to* -12*, the augmented matrix*

$$\begin{pmatrix} 1 & 1 & 1 & -1 \\ 2 & 2 & 5 & -8 \\ 4 & 4 & 8 & -12 \end{pmatrix}$$

is reduced by Gaussian elimination to

$$\begin{pmatrix} 1 & 1 & 1 & -1 \\ 0 & 0 & 3 & -6 \\ 0 & 0 & 4 & -8 \end{pmatrix}.$$

Now the final two equations both give $x_3 = -2$ *and the first implies* $x_1 + x_2 = 1$*, i.e. there is an infinity of solutions.*

This situation, though, is nongeneric: any other value apart from -12 in this last entry would have given no solution. Therefore, when performing Gaussian elimination in practice (using real arithmetic with finite precision) it is safest simply to stop the calculation if $\boldsymbol{A}$ appears to be singular; the system almost certainly has no solution.

4.6 Implementation of Gaussian elimination

Now we show how one could write a computer program to carry out Gaussian elimination. We illustrate the algorithm by means of a flowchart in figure 1. It is straightforward (exercise 4.1) to translate this into (*e.g.*) `Fortran90`.

The main loop variable i goes from 1 to $n-1$; at each stage the first step is to check whether the pivot entry a_{ii} is zero. If it is, we pass to the pivoting subroutine, otherwise the program proceeds directly to the elimination stage. In the pivoting subroutine, j looks at each row in turn below the i^{th} row until it finds one with a nonzero pivot entry. The program then swaps rows i and j and returns to the elimination stage. If j reaches n and no nonzero pivot entry has been located, then the matrix is singular.

The purpose of the elimination stage is to use the i^{th} row to eliminate the i^{th} column. Thus k cycles through all the rows below the i^{th} one; for each, r calculates the multiple of row i that must be subtracted from row k. When k reaches n, elimination of the i^{th} column is completed, and so i can be incremented. When i reaches n, Gaussian elimination is finished, the matrix is in echelon form, and back-substitution may proceed.

4.7 Operations count

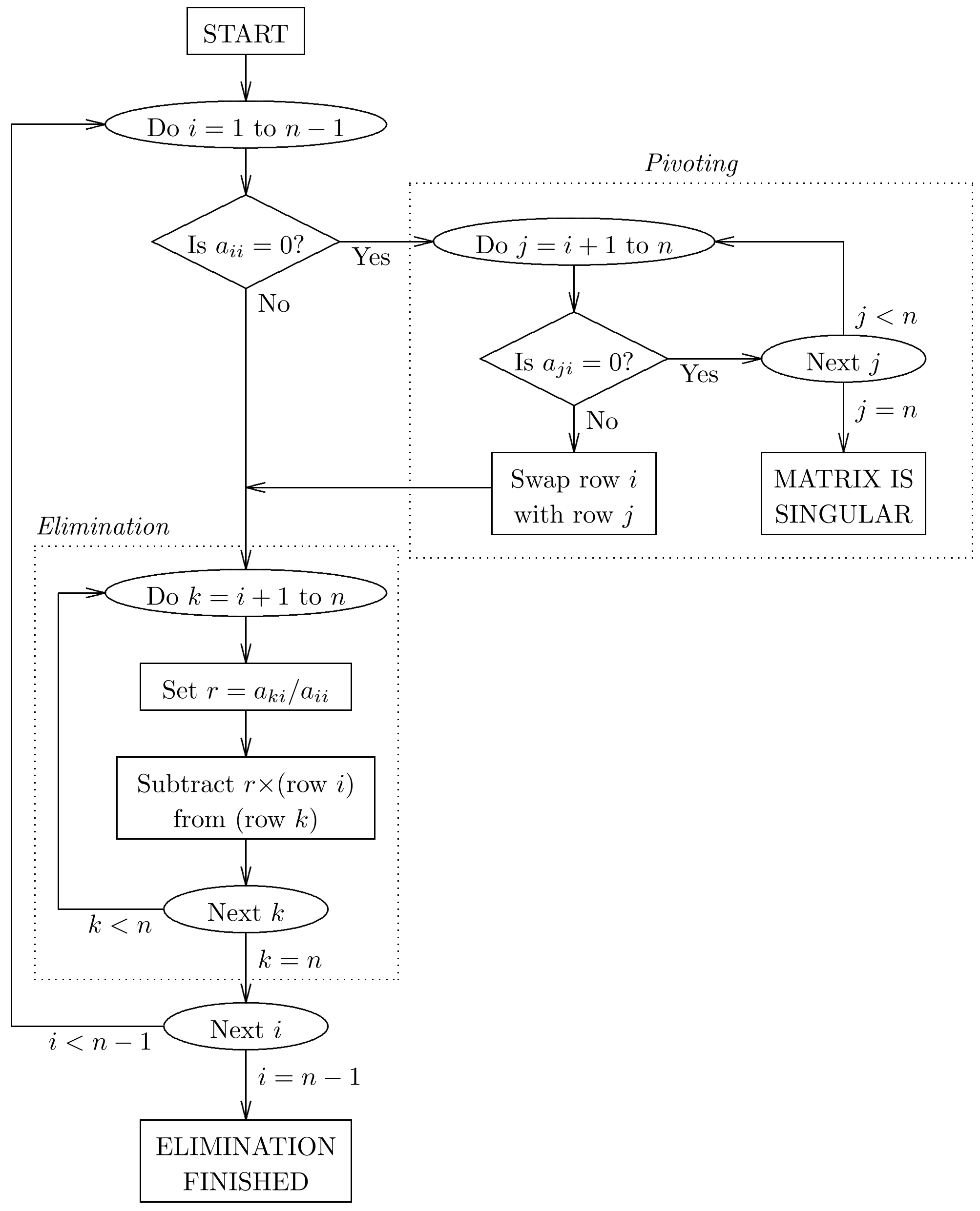
When evaluating any algorithm, the following criteria should be borne in mind:

1. ease of programming;
2. effect of rounding errors;
3. storage space required;
4. computing time required.

Gaussian elimination is straightforward to program, and rounding errors may be controlled by the method of *partial pivoting*, described below. It requires the $n \times (n+1)$ augmented matrix to be stored, which may be inefficient if n is very large, but most of the coefficients of $\boldsymbol{A}$ are zero. Such systems are called *sparse*, and some routines that may be more efficient for large, sparse systems are discussd in §6.

Here we address the question of computing time. An estimate of the time it will take to perform a calculation may be obtained by counting the number of arithmetic operations involved. It is standard to count each multiplication and each division as a single operation, but to ignore additions and subtractions (which require significantly less processor time).

Flowchart for the Gaussian elimination method



The first step in Gaussian elimination is to use the top row to eliminate the first column from each subsequent row. For each, we have to carry out 1 division, to calculate the multiple of row 1 to be substracted (*i.e.* the ratio r in figure 1), followed by n multiplication-subtractions. This makes $n+1$ operations for each of the $n-1$ rows below the top one, leading to a total of $(n-1)(n+1) = n^2 - 1$ operations needed to eliminate the first column.

Next we continue to the second row. Effectively we must perform the same operation as that just described, but with an $(n-1) \times n$ matrix instead of $n \times (n+1)$. Thus the number of operations needed to eliminate column 2 is $(n-1)^2 - 1$. Therefore, to complete the elimination through all n rows, the total number of operations required is[1]

$$N_n = \sum_{k=1}^{n} \left(k^2 - 1\right) = \frac{n(n+1)(2n+1)}{6} - n = \frac{n(n-1)(2n+5)}{6}.$$

Of particular interest is the behaviour of N_n when n is large:

$$N_n = \frac{n^3}{3} + \frac{n^2}{2} - \frac{5n}{6} \sim \frac{n^3}{3}$$

since $n^3 \gg n^2$ or n for large n. Thus an adequate estimate of the number of operations required for Gaussian elimination of an n-dimensional system is $n^3/3$.

This should be compared with the $n!$ operations needed to find the determinant of the same system. It should be clear that Gaussian elimination is vastly preferable to solving using determinants if n is at all large. For example, a 100×100 system requires fewer than 10^6 operations, which a 1000MHz machine could perform in less than a millisecond.

4.8 Back-substitution

Now suppose the elimination process is complete and the augmented matrix is in echelon form. Thus the reduced system takes the form

$$\begin{array}{ccccccccccc}
a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1(n-1)}x_{n-1} & + & a_{1n}x_n & = & b_1, \\
 & & a_{22}x_2 & + & \cdots & + & a_{2(n-1)}x_{n-1} & + & a_{2n}x_n & = & b_2, \\
 & & & & \ddots & & \vdots & & \vdots & & \vdots \\
 & & & & & & a_{(n-1)(n-1)}x_{n-1} & + & a_{(n-1)n}x_n & = & b_{n-1}, \\
 & & & & & & & & a_{nn}x_n & = & b_n.
\end{array}$$

The next stage is back-substitution, of which the first step is to solve the final equation for x_n:

$$x_n = -b_n/a_{nn}.$$

[1]Here we use the identity

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

Then with x_n known, the penultimate equation gives

$$x_{n-1} = \left(b_{n-1} - a_{(n-1)n}x_n\right)/a_{(n-1)(n-1)}.$$

By continuing in a similar fashion we may generate in turn $x_{n-2}, \ldots, x_2, x_1$; the algorithm for back substitution may thus be written in the following form.

```
• START
• Do $k = n$ to 1 step $(-1)$
    • Set $x_k = \frac{1}{a_{kk}}\left(b_k - \sum_{j=k+1}^{n} a_{kj}x_j\right)$
• End do
• END
```

Now let us count the number of operations required for back substitution. To solve for x_n requires just one division, while one multplication-subtraction followed by one division are needed to solve for x_{n-1}. In general, then, at the k^{th} step, k operations are needed, giving rise to a total of

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2} \sim \frac{n^2}{2}$$

when n is large. The main point is that this is negligible compared to the number of operations used in the elimination stage.

4.9 Partial pivoting

We have seen that it may be necessary to swap two rows if a diagonal entry, which we intend to use as a pivot, turns out to be zero. However, when implementing the procedure on a computer with finite-precision arithmetic, testing whether a quantity is equal to zero is unsafe. In any case, inaccuracy may arise if the pivot is too small, so it is prudent at each stage to choose the row with the largest possible entry in the pivot position, and use that as the pivot.

Example 4.7 *Consider the augmented matrix*

$$\begin{pmatrix} \boxed{0.0001} & 1 & 1 \\ 1 & 1 & 2 \end{pmatrix}, \tag{4.8}$$

in which the pivot entry, although nonzero, is small. The effect of this is that we have to use a large multiple, 10000, in eliminating the first column, resulting in

$$\begin{pmatrix} 0.0001 & 1 & 1 \\ 0 & -9999 & -9998 \end{pmatrix}.$$

Now back-substitution gives $x_2 = 9998/9999$*, and if only three decimal places are stored, this is approximated by* $x_2 = 1.000$*. Then we find* $0.0001x_1 = 0.000$*, that is* $x_1 = 0.000$*, which is the wrong answer.*

However if, on the grounds that the pivot entry is too small, we first swap rows:

$$\begin{pmatrix} \boxed{1} & 1 & 2 \\ 0.0001 & 1 & 1 \end{pmatrix},$$

then Gaussian elimination leads to

$$\begin{pmatrix} 1 & 1 & 2 \\ 0 & 0.9999 & 0.9998 \end{pmatrix}.$$

Now back-substitution results in (to three decimal places) $x_2 = 1.000$*,* $x_1 = 1.000$*, which is correct.*

Errors, like those just illustrated, which arise from the fact that a computer can only store real numbers with a finite precision are called *rounding errors*.

Actually, the argument for selecting the largest possible pivot entry is still slightly unsound, since we can multiply a whole row through by any factor we like to make the pivot entry bigger, without alleviating the problem. For example, multiplication of the first row of (4.8) by 10000 results in

$$\begin{pmatrix} \boxed{1} & 10000 & 10000 \\ 1 & 1 & 2 \end{pmatrix}.$$

Now the pivot entry in the first row looks fine, but the same loss of accuracy occurs if Gaussian elimination is attempted without pivoting [**exercise**].

What one should really look for, when deciding whether or not to swap rows, is the size of the pivot entry compared with the other entries in the same row. In the method of *partial pivoting*, we choose at each stage the row which maximises (in absolute value) the ratio between the pivot entry and the largest entry in the same row (apart from the last one, *i.e.* the right-hand side). The row which maximises this so-called *pivot ratio* is then switched into the pivot position.

Example 4.8 [Kreyszig] *Consider the augmented matrix*

$$\begin{pmatrix} 3 & -4 & 5 & -1 \\ -3 & 2 & 1 & 1 \\ 6 & 8 & -1 & 35 \end{pmatrix}.$$

In row one the largest entry (apart from the last) is 5*, giving a pivot ratio* $3/5$*. The corresponding ratios for rows two and three are* $3/3$ *and* $6/8$*, of which that for row two is the biggest. We therefore swap row two into the pivot position,*

$$\begin{pmatrix} -3 & 2 & 1 & 1 \\ 3 & -4 & 5 & -1 \\ 6 & 8 & -1 & 35 \end{pmatrix},$$

and then eliminate column one:

$$\begin{pmatrix} -3 & 2 & 1 & 1 \\ 0 & -2 & 6 & 0 \\ 0 & 12 & 1 & 37 \end{pmatrix}.$$

Now the 6 *is the largest entry in row two, giving a pivot ratio of* 2/6*, compared with* 12/1 *for row three. So we switch row three into the pivot position:*

$$\begin{pmatrix} -3 & 2 & 1 & 1 \\ 0 & 12 & 1 & 37 \\ 0 & -2 & 6 & 0 \end{pmatrix},$$

and eliminate to give

$$\begin{pmatrix} -3 & 2 & 1 & 1 \\ 0 & 12 & 1 & 37 \\ 0 & 0 & 37/6 & 37/6 \end{pmatrix},$$

which is now ready for back-substitution.

Note that, as well as avoiding unnecessary rounding errors, this procedure automatically swaps zero pivots where necessary and possible (*i.e.* it replaces the *pivoting* stage in figure 1).

Partial pivoting may be written as an algorithm as follows.

```
• START
• Do $i = 1$ to $n - 1$
    • Set $temp_1 = 0.0$
    • Set $m = 0$
    • Do $j = 1$ to $n$
        • Set $temp_2 = \max_{k=i,\dots,n} |a_{jk}|$
        • If $|a_{ji}| > temp_1 \times temp_2$ then
            · Set $temp_2 = |a_{ji}|/temp_1$
            · Set $m = j$
        • End if
    • End do
    • If $m \neq i$ then
        • Swap row $i$ with row $m$
    • End if
    • ELIMINATE COLUMN $i$
• End do
• ELIMINATION FINISHED
```

This fits inside the main loop of figure 1, at the point where we are about to use row i to eliminate column i. We start by initialising a temporary real variable $temp_1$ and an integer m; these will be used to store the maximum pivot ratio found so far, and the row in which it occurs. Now j cycles through all the rows below and including the

i^{th} one, and in each such row, $temp_2$ calculates the largest entry. Then the pivot ratio for that row is given by $|a_{ji}|/temp_2$; we check to see whether this is greater than $temp_1$. If so, row j holds the largest pivot ratio found so far, so $temp_1$ and m are replaced accordingly. When j has run through every row, m tells us which row has the largest pivot ratio; if m is not equal to i, we swap rows i and m. Now we have the best possible row in the pivoting position, and are ready to continue with elimination.

Finally, we mention that there is an even more complicated procedure, called *total pivoting*, which involves swapping columns as well as rows to maximise the pivot ratio at each stage. This is even safer, so far as rounding errors are concerned. However, it is also considerably more expensive in terms of computer time, and since partial pivoting usually gives adequate accuracy, is rarely used.

Exercises

4.1. Write a program in `Fortran90` to solve an arbitrary linear system of size n. The program should

(a) ask the user to input the size n of the system and the $n \times (n+1)$ augmented matrix;

(b) perform Gaussian elimination, with partial pivoting, on the matrix to reduce it to echelon form and print out the reduced matrix;

(c) perform back-substitution on the reduced matrix to solve the system, and print out the solution.

Test your program on the systems from examples 4.1, 4.3 and 4.4.