10.001: System of Linear Equations, Part 2

R. Sureshkumar

November 5, 1996

1 Invariant Operations and Gaussian Elimination

Here, we will discuss certain operations on a system of equations which do not alter the solution to them. Such operations are called invariant operations, since they do not disturb the solution vector of the original problem. The basic idea involved here is once again that of linear dependence, i.e., we can replace one or more of the n equations by a linear combination of a set of the remaining equations, without changing the solution vector. Hence, multiplication of any of the equations with a non-zero scalar, addition of two or more equations etc. are invariant operations. As we perform operations on the coefficents of the equations, an identical set of operations has to be performed on the appropriate component of the right hand side vector as well. Hence, it is convenient if we append the right hand side vector to the coefficient matrix as an $n+1$th column. The resulting $n \times (n+1)$ matrix is called the *augmented* matrix.

Let's consider an example. The system of equations is given by $x+2y+z = 8$, $3x+4y+2z = 17$ and $5x + 6y + z = 20$. By adding the right hand side vector as the 4th column onto the coefficient matrix, we get the augmented matrix $\tilde{\mathbf{A}}$, given below.

$$\tilde{\mathbf{A}} = \begin{pmatrix} 1 & 2 & 1 & 8 \\ 3 & 4 & 2 & 17 \\ 5 & 6 & 1 & 20 \end{pmatrix}.$$

The augmented matrix now represents the entire system of equations, each row representing an equation. For instance, the first row means $1x + 2y + 1z = 8$. Now let's replace Row 2 with $-3\times$ Row 1 + Row 2 and Row 3 with $-5\times$ Row 1 + Row 3. This gives

$$\tilde{\mathbf{A}}_1 = \begin{pmatrix} 1 & 2 & 1 & 8 \\ 0 & -2 & -1 & -7 \\ 0 & -4 & -4 & -20 \end{pmatrix}.$$

Now, we do the following operation: $-2\times$ Row 2 + Row 3 = Row 3. This gives

$$\tilde{\mathbf{A}_2} = \begin{pmatrix} 1 & 2 & 1 & 8 \\ 0 & -2 & -1 & -7 \\ 0 & 0 & -2 & -6 \end{pmatrix}.$$

Now, the augmented matrix $\tilde{\mathbf{A}_2}$ tells us that (read from bottom to top) $-2z = -6$; $-2y - z = -7$ and $x + 2y + z = 8$. This is easily solved to give

$$\begin{aligned} z &= -6/(-2) = 3 \\ y &= (-7+3)/(-2) = 2 \\ z &= (8 - 3 - 2 \times 2)/1 = 1, \end{aligned} \tag{1}$$

which is indeed the correct solution. The procedure mentioned above is one of the basic methods for the solution of a linear system of equations, known as *Gaussian elimination* followed by *back substitution*. Here, the procedure we followed to come up with the upper triangular coefficient matrix (corresponding to $\tilde{\mathbf{A}_2}$) is called Gaussian elimination. Solution by back substitution refers to the procedure illustrated in Eq. 1.

Note that by doing the operations Row 1 = Row 1 + Row 2 and Row 2 = Row 2 - (Row 3)/2 followed by the operation Row i = Row i/(diagonal element of Row i), on $\tilde{\mathbf{A}_2}$ will result in the diagonal system

$$\tilde{\mathbf{A}_3} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \end{pmatrix},$$

which directly gives us the solution without the back substitution step. This latter approach, which involves a direct reduction of the coefficient matrix to a unit (identity) matrix is the basis for the *Gauss-Jordan* algorithm.

In the following section, we would generalize the algorithm discussed above.

2 The Gaussian Elimination Algorithm

The generalization of the algorithm we discussed in the previous section for an $n \times n$ system, $\mathbf{A}.\mathbf{x} = \mathbf{b}$ can be given as follows:
Step 1: Forward Reduction

$$\begin{aligned} &\text{For } k = 1, \cdots, n-1 \\ &\quad \text{For } i = k+1, \cdots, n \\ &\qquad l_{ik} = a_{ik}/a_{kk} \end{aligned}$$

$$
\begin{aligned}
&\text{For } j = k+1, \cdots, n \\
&\quad a_{ij} = a_{ij} - l_{ik} a_{kj} \\
&\text{End of loop over } j \\
&\quad b_i = b_i - l_{ik} b_k \\
&\text{End of loop over } i \\
&\text{End of loop over } k.
\end{aligned} \tag{2}
$$

Step 2: Back Substitution

$$
\begin{aligned}
&\text{For } k = n \cdots, 1 \\
&\quad x_k = b_k \\
&\text{For } i = k+1, \cdots, n \\
&\quad x_k = x_k - a_{ki} x_i \\
&\text{End of loop over } i \\
&\quad x_k = x_k / a_{kk} \\
&\text{End of loop over } k.
\end{aligned} \tag{3}
$$

The algorithm above assumes that the diagonal entries, a_{kk}, are all non-zero. We will discuss later on a modification to the algorithm when this happens. Also notice that the algorithm overwrites the coefficient matrix and the right hand side vector every time a row operation is performed. This helps reduce the storage requirements, which is significant for large values of n.

2.1 Number of Operations Required for the Gaussian Elimination Algorithm

We now estimate the number of arithmetic operations needed to compute the solution $\mathbf{x}$ using the Gaussian elimination algorithm. The majority of the computational load (we assume n is large) is concentrated in reducing the coefficient matrix using the statement $a_{ij} = a_{ij} - l_{ik} a_{kj}$ in Step 1 above. This requires one addition and one multiplication, so in the j loop of Step 1, we have $n-k$ additions. The i loop repreats this $n-k$ times. So, the total number of additions over the i, j and k loops is given by (verify the steps by yourself)

$$
\begin{aligned}
\sum_{k=1}^{n-1} (n-k)^2 &= \sum_{k=1}^{n-1} k^2 \\
&= \frac{2n^3 - 3n^2 + n}{6} \\
&\approx \frac{n^3}{3}, \text{ for large values of } n.
\end{aligned} \tag{4}
$$

In other words, the CPU (central processing unit) time requirement for the Gaussian elimination algorithm scales with the cube of the number of unknowns. For this reason, numerical analysts refer to it simply as a n-cube method. The implication is that if we can solve at best a system of n equations on a a computer we currently have access to in say a time of T units, the solution of a $10n$ system on the same computer will take (assuming memory requirements are sufficient) $1000T$ units of time. This is not a very desirable feature for a computational method and in the parlance of numerical analysts, the algorithm has rather poor *scalability*. This observation has prompted a great deal of research effort to develop algorithms with better scalability for the solution of large systems of linear equations. Such methods consists of techniques which take into account any special structre of the coefficient matrix, iterative techniques, use of preconditioner matrices and acceleration techniques, a discussion of which is beyond the scope of our course.

2.2 Potential Errors in Gaussian Elimination

2.2.1 How do we deal with the $a_{kk} = 0$ situation?

In the algorithm discussed above, we assumed that the diagonal elements do not vanish at any stage in the elimination process. However, it is common to encounter diagonal entries which is zero or close to zero. We need to modify our algorithm to circumvent such difficulties. This can be simply achieved by interchanging the row with 0 diagonal element with another row below it which has a non-zero element in the same column as the 0 element. This idea is illustrated through the following example.

Consider the 3×3 system of equations: $x + y + z = 3$, $x + y = 2$ and $y + z = 2$. The augmented matrix is

$$\tilde{\mathbf{A}} = \begin{pmatrix} 1 & 1 & 1 & 3 \\ 1 & 1 & 0 & 2 \\ 0 & 1 & 1 & 2 \end{pmatrix}.$$

Now, Row 2 = Row 2 - Row 1 gives us

$$\tilde{\mathbf{A}_1} = \begin{pmatrix} 1 & 1 & 1 & 3 \\ 0 & 0 & -1 & -1 \\ 0 & 1 & 1 & 2 \end{pmatrix}.$$

Evidently, $a_{22} = 0$ implying a break down of the algorithm. i.e., l_{32} (see Step 1 of the Gaussian elimination algorithm) is not defined or infinite. We can overcome this difficulty by interchanging row 2 and row 3 to give (this does not affect the solution, we are simply changing the order in which we write the equations)

$$\tilde{\mathbf{A}_2} = \begin{pmatrix} 1 & 1 & 1 & 3 \\ 0 & 1 & 1 & 2 \\ 0 & 0 & -1 & -1 \end{pmatrix}.$$

Now, we can obtain the solution by back substitution.

2.3 Rounding Error and Instability

The rounding error we discussed in relation to the numerical integration rules are of importance in Gaussian elimination as well. The first type of error occurs due to the accumulation of rounding errors during a large number of arithmetic operations. For instance, if $n = 1000$, then the number of operations (see Eq. 4) is of the order 10^9, hence, even in double precision computations, the error could br large. Remember that this type of error accumulation was demonstrated in realtion the numerical integration.

However, a more serious problem involves catastrophic rounding errors, leading to *numerical instability*. Let's analyze the following 2×2 system to understand this. Consider the system of equations $-10^{-5}x + y = 1$ and $2x + y = 0$ with the exact solution $x = -0.49999975...$ and $y = 0.999995$. Suppose we are solving this system of equations on a computer which has a precision of four decimal digits using the Gaussian elimination procedure, i.e., the representation of any real constant is given by $0.*****\times 10^p$. The augmented matrix to start with is given by

$$\tilde{\mathbf{A}} = \begin{pmatrix} -0.1000 \times 10^{-4} & 1 & 1 \\ 2 & 1 & 0 \end{pmatrix}.$$

If we now apply the Gaussian elimination procedure, it results in the following computations:
$l_{21} = a_{21}/a_{11} = -0.2 \times 10^6$ which is exact;
$a_{22} = 0.1 \times 10^1 - (-0.2 \times 10^6)(0.1 \times 10^1) = 0.2 \times 10^6$ which is approximate since the computer can store values only upto four decimal places (the exact answer here is 0.200001×10^6;
$b_2 = -(-0.2 \times 10^6)(0.1 \times 10^1) = 0.2 \times 10^6$ which is exact. Hence, we have the following system:

$$\tilde{\mathbf{A}_1} = \begin{pmatrix} -0.1000 \times 10^{-4} & 1 & 1 \\ 0 & 0.2 \times 10^6 & 0.2 \times 10^6 \end{pmatrix}.$$

giving us the (wrong) solution $x = 0$ and $y = 1$. Indeed, we could have dealt with the error in y since the answer is correct to the precision of the computer, but as for the value of x, the error is a factor of 50000 larger than the precison of the machine.

The origin of this problem may be traced to the large value of l_{21}, which forced, in the calculation of new a_{22} as $a_{22,new} = a_{22,old} - l_{21} * a_{12} = 0.1 \times 10^1 - (-0.2 \times 10^6)(0.1 \times 10^1)$, the first term ($a_{22,old}$) to be neglected. Indeed, by rewriting the system as

$$\tilde{\mathbf{A}_{rearranged}} = \begin{pmatrix} 2 & 1 & 0 \\ -0.1000 \times 10^{-4} & 1 & 1 \end{pmatrix},$$

we can avoid this problem, $l_{21} = -0.5 \times 10^{-5}$ in this case. Show that Guassian elimination of this system leads to an answer correct within the precision of the system.

The concept illustrated above can be generalized in an algorithm where we keep all the multipliers (l_{ik}) less than or equal to one in absolute value. This procedure is known as *partial pivoting*: at the kth stage of the elimination process, an interchange of rows is made, if necessary, to place in the main diagonal position the element of largest absolute value from the kth column on or below the main diagonal. This strategy, combined with the row interchange procedure when a 0 diagonal element is encountered, gives rise to a more complete and robust Gaussian elimination algorithm as below: only Step 1 (see 2) is changed, the back substitution step, i.e., Step 2 (see 3), remains the same.

Forward Reduction with Partial Pivoting

For $k = 1, \cdots, n-1$
Find $m \geq k$ such that $|a_{mk}| = \max\{|a_{ik}| : i \geq k\}$.
If $a_{mk} = 0$ then the $\mathbf{A}$ is singular, no unique solution, so stop.
Else interchange a_{kj} and a_{mj}, $j = k, k+1, \cdots, n$; interchange b_k and b_m.
For $i = k+1, k+2, \cdots, n$
$l_{ik} = a_{ik}/a_{kk}$
For $j = k+1, k+2, \cdots, n$
$a_{ij} = a_{ij} - l_{ik}a_{kj}$
End j loop
$b_i = b_i - l_{ik}b_k$
End i loop
End k loop.