Systems of Linear Equations

Consider the following set of n equations:

$$\begin{array}{ccccccccccc} a_{11}x_1 & + & a_{12}x_2 & + \cdots + & a_{1j}x_j & + \cdots + & a_{1n}x_n & = b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + \cdots + & a_{2j}x_j & + \cdots + & a_{2n}x_n & = b_2 \\ & & \cdot & & \cdot & \cdot & & \cdot \\ a_{i1}x_1 & + & a_{i2}x_2 & + \cdots + & a_{ij}x_j & + \cdots + & a_{in}x_n & = b_i \\ & & \cdot & & \cdot & \cdot & & \cdot \\ a_{n1}x_1 & + & a_{n2}x_2 & + \cdots + & a_{nj}x_j & + \cdots + & a_{nn}x_n & = b_n \end{array} \tag{1}$$

The a's on the left hand side of each equation are n^2 given coefficients. The b's on the right are n given numbers. The x's are unknowns. Clearly a_{ij} is the coefficient of the j^{th} unknown in equation i. Each equation is *linear* in the x's; that is, they appear with exponent one, multiplied by scalar constants. The task is to assign numerical values to the n unknowns so that the all the equations are satisfied simultaneously.

If we store the coefficients in the matrix

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ & & & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix},$$

and let $\underline{x}$ denote the (column) vector of unknowns and $\underline{b}$, the vector of right-hand sides (we underline to distinguish vectors from scalars), then (1) may be expressed more succinctly as

$$A\underline{x} = \underline{b}\,. \tag{2}$$

By the definition of matrix multiplication of vectors, $A\underline{x}$ is the vector whose i^{th} component is $\sum_{i=1}^{n} a_{ij}x_j$, the left-hand side of the i^{th} equation in (1).

Given A and $\underline{b}$ ($n^2 + n$ inputs), (2) asks for vectors $\underline{x}$ that are mapped by A (via matrix multiplication) to $\underline{b}$. These are the solutions of the system. It is possible that there are *no* solutions, a single, unique solution, or an infinite number of solutions. The data defining the system in (1) or (2) may be stored in the *augmented coefficient matrix*

$$A' \equiv (A|\underline{b}) = \left(\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ & & & & \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{array}\right). \tag{3}$$

1. **Gaussian Elimination and Backsolving:** This is the main computational method for solving linear systems. Gaussian elimination operates on (3) in a sequence of "steps". After each step we have a new system with the same solution set as (3). The final system, the *reduced form* of (3) is

$$C' \equiv (C|\underline{d}) = \left(\begin{array}{ccccc|c} c_{11} & c_{12} & \cdots & c_{1,n-1} & c_{1n} & d_1 \\ 0 & c_{22} & \cdots & c_{2,n-1} & c_{2n} & d_2 \\ & & & & & \\ 0 & 0 & \cdots & c_{n-1,n-1} & c_{n-1,n} & d_{n-1} \\ 0 & 0 & \cdots & 0 & c_{nn} & d_n \end{array}\right). \tag{4}$$

Notice that in column j, all the coefficients are zero in the rows below row j.

The "steps" are done by the following algorithm, GEBS. The inputs are n, the size of the system, and A and $\underline{b}$, the data describing the system; the output is the solution vector $\underline{x}$.

GEBS(n, A , $\underline{b}$; $\underline{x}$)

- **FOR** $j = 1$ **TO** $n-1$ **DO** [process column j]
  - **IF** $a_{kj} = 0, k = j, \ldots, n$ **THEN STOP (no unique solution), ELSE**
    - $i \leftarrow \min(k \geq j : a_{kj} \neq 0)$
    - **IF** $i \neq j$ **SWAP ROW**$_i \leftrightarrow$ **ROW**$_j$ [now $a_{jj} \neq 0$]
  - **ENDIF**
  - **FOR** $k = j+1$ **TO** n **DO** [eliminate x_j from equation k]
    - $c \leftarrow a_{kj}/a_{jj}$ [this is the pivot value]
    - $\boxed{\mathbf{Row}_k \leftarrow \mathbf{Row}_k - c * (\mathbf{Row}_j)}$ [pivot step]
  - **ENDFOR**
- **ENDFOR**

At this point we have arrived at the reduced system in (4) with the guarantee that $c_{jj} \neq 0$, $j < n$ (the c_{jj} are the pivots). If also $c_{nn} \neq 0$ the system has a unique solution. The last equation in (4) ($c_{nn}x_n = d_n$) implies that $x_n = d_n/c_{nn}$. The remaining x's are found by *backsolving*, as follows: The k^{th} equation in (4) is

$$c_{kk}x_k + c_{k,k+1}x_{k+1} + \cdots + c_{kn}x_n = d_k\,.$$

If we knew the solution values of $x_{k+1}, \ldots, x_n$, they may be used in the above equation to evaluate

$$x_k = (d_k - (c_{k,k+1}x_{k+1} + \cdots + c_{k,n}x_n))/c_{kk}\,, \tag{5}$$

the solution value of x_k. Thus, starting with x_n, we iterate (5) for $k = n-1, n-2, \ldots, 1$.

- **IF** $c_{nn} = 0$ **THEN STOP (if** $d_n \neq 0$**, NO solution); ELSE, an infinite number**
  - **ELSE** $x_n \leftarrow d_n/c_{nn}$
  - **FOR** $k = n-1$ **DOWNTO 1 DO**
    - $x_k \leftarrow (d_k - \sum_{i=k+1}^{n} c_{ki}x_i)/c_{kk}$
  - **ENDFOR**
- **ENDIF**

In fact the system (1) has a unique solution if and only if Gaussian elimination arrives at (4) and $c_{nn} \neq 0$.

2. **Gauss-Jordan Reduction** This is a sequence of "steps" which produces as a final, reduced form of (3), the system

$$C' \equiv (C|\underline{d}) = \left(\begin{array}{ccccc|c} c_{11} & 0 & \cdots & 0 & 0 & d_1 \\ 0 & c_{22} & \cdots & 0 & 0 & d_2 \\ & & \cdots & & & \\ 0 & 0 & \cdots & c_{n-1,n-1} & 0 & d_{n-1} \\ 0 & 0 & \cdots & 0 & c_{nn} & d_n \end{array}\right). \tag{6}$$

The Gauss-Jordan reduction steps are the same as those in Gaussian elimination except the outer **FOR** loop runs from 1 to n and the inner FOR loop is **FOR** $k = 1$ **TO** n, $k \neq j$. At this point we have (6) and $c_{ii} \neq 0$, $i = 1, \ldots, n$. Note that this coefficient matrix is *diagonal*, so equation i is $c_{ii}x_i = d_i$. Therefore, after Gauss-Jordan reduction, we can solve with

$$\underline{\text{FOR}}\ \mathbf{i = 1}\ \underline{\text{TO}}\ \mathbf{n}\ \underline{\text{DO}}\ \{\mathbf{x_i = d_i/c_{ii}}\}\ \underline{\text{ENDFOR}}.$$

3. **Pivoting Strategy:** Simple examples show that GEBS (and Gauss-Jordan) can produce very large roundoff errors when carried out in k-digit arithmetic (see handout on this topic). One class of remedies is based on pivoting strategies. They look for a different pivot row because the one currently in the j^{th} position is expected to generate roundoff errors. The candidates for the pivot row to be used to eliminate x_j are rows j through n.

   - **Partial Pivoting:** The partial pivoting strategy says that the best row for eliminating x_j will have the largest value of $|a_{jj}|$ (because the pivot value is $c = a_{kj}/a_{jj}$). Thus instead of the **IF** in lines 3-6 of GEBS we use
     - find $i : |a_{ij}| = \max(|a_{kj}|,\ k = j, \ldots, n)$
     - **IF** $|a_{ij}| = 0$ **THEN STOP (no unique sol.), ELSE**
     - **IF** $i \neq j$ **SWAP ROW**$_i \leftrightarrow$ **ROW**$_j$ $\qquad [\ a_{jj} \neq 0]$

     The same change may be added to Gauss-Jordan reduction. We still seek candidates for the pivot row from rows j through n, even though with Gauss-Jordan, this pivot is used in *all* rows $\neq j$.
   - **Scaled Partial Pivoting:** In multiplying $c * (\text{ROW}_j)$ to eliminate x_j from equation i we have

$$\frac{a_{ij}}{a_{jj}} * (\underbrace{(\overbrace{0, \ldots, 0}^{j-1}, a_{jj}, a_{j,j+1}, \ldots, a_{jn} | b_j)}_{\text{ROW}_j} = a_{ij} * (\overbrace{0, \ldots, 0}^{j-1}, 1, \frac{a_{j,j+1}}{a_{jj}}, \ldots, \frac{a_{jn}}{a_{jj}} | \frac{b_j}{a_{jj}}).$$

     Scaled-partial pivoting wants *all* the ratios to be small so a_{ij} wont be multiplied by too large a number.

     We look for the row that best achieves this as follows: For each candidate row ($k = j, \ldots, n$) we compute its SIZE, or good-scaling-coefficient by

$$\sigma_k = \begin{cases} \infty & \text{if } a_{kj} = 0 \\ \max(|a_{k,\ell}|, \ell = j+1, \ldots, n)/|a_{kj}| & \text{otherwise} \end{cases}$$

     We take row i as the pivot row if it is the best scaled row; i.e.,

$$\sigma_i \equiv \min(\sigma_k, k = j, \ldots, n) < \infty.$$

     If $\sigma_i = \infty$ there is no unique solution. The same strategy also applies to Gauss-Jordan reduction.

4. **Cost of Gaussian Elimination and Backsolving:** We will count the number of * and / steps used in the above procedures (the number of + and - steps could be counted in the same way).

Suppose we have already processed columns 1 through $j-1$. We will compute the cost of the FOR $k = j+1$ TO n loop. For each k, one divide step is needed to compute $c = a_{kj}/a_{jj}$. Then we multiply each entry of row j by c (before subtracting the product from row k), as follows

$$c * \underbrace{(\overbrace{0, \ldots, 0}^{j-1}, a_{jj}, a_{j,j+1}, \ldots, a_{jn} | b_j)}_{\text{ROW}_j} : \tag{7}$$

the first $j-1$ entries are 0 because we have already processed columns 1 through $j-1$. There is no need to multiply c by 0 (we know the answer). There is no need to multiply c by a_{jj} (we know the product is a_{kj}). Therefore $n-j+1$ multiplications are needed for (7), and, adding the division used to compute c, a total of $n-j+2$ multiply or divide steps (we will just say "steps") were performed in eliminating x_j from equation k. Since the FOR k loop is executed $n-j$ times, the cost to eliminate x_j from all equations below the j^{th} is $(n-j)(n-j+2)$ steps. Summing from $j = 1, \ldots, n-1$ the cost of Gaussian elimination is

$$\text{GE} = \sum_{j=1}^{n-1} (n-j)(n-j+2),$$

which may be simplified to $(2n^3 + 3n^2 - 5n)/6$. Equation (5) has 1 division and $n-k$ multiplications, so the FOR $k = n-1$ in backsolving costs $\sum_{k=1}^{n-1} n-k+1$ steps. Adding the one division in $x_n = d_n/c_{nn}$, the total work in backsolving is

$$\text{BS} = n(n+1)/2,$$

and so the total cost to obtain the solution is

$$\text{GEBS} = \frac{n^3 - n}{3} + n^2 \tag{8}$$

multiply/divide steps. Observe that the work increases as the *cube* of n, the size of the system. This means if you double the size of a system you would work eight times as hard. Also note that the coefficient of n^3 is 1/3. One would expect the running time of any decent implementation of GEBS to grow as the *cube* of the number of equations being solved.

Since partial pivoting does no multiplications or divisions, it may be included at no cost (actually $n(n-1)/2$ comparisons are performed, but when n is large, this will have no noticeable effect on the running time). Scaled partial pivoting does less than $n(n+1)/2$ division steps. However it also makes about $n^3/3$ comparisons which would increase the running time by a constant factor.

5. **Cost of Gauss-Jordan Reduction:** As with Gaussian elimination, the cost of the pivot step to eliminate x_j from an equation is $n-j+2$. Here this is done in $n-1$ equations. Therefore the reduction costs

$$GJ = \sum_{j=1}^{n} (n-1)(n-j+2) = (n-1)[\frac{(n+1)(n+2)}{2} - 1],$$

steps, and adding the n divisions needed to solve the reduced system, we find that the total is

$$\frac{n^3 - n}{2} + n^2\,; \tag{9}$$

this means that for large n Gauss-Jordan is 50% more costly (or time-consuming) than Gaussian elimination. The added costs of pivot strategies are the same as with Gaussian elimination.

6. **Computing Inverses:** Suppose A^{-1} exists. Gauss-Jordan reduction is a sequence of operations that transforms the matrix A in (3) to the matrix C in (6), and no $c_{ii} = 0$. If we now perform n more row operations on C (divide row_i by c_{ii}, $i = 1, \ldots, n$) we will have reduced A to I. We can denote this as

$$[E_k E_{k-1} \cdots E_1]A = I.$$

The matrices E_i are elementary matrices, each performing one of the row operations in the above reduction process. Thus $E_k E_{k-1} \cdots E_1 = A^{-1}$ which shows that the sequence of row operations that reduces $A \to I$ also reduces $I \to A^{-1}$.

A counting argument shows that the inverse may be computed in this way with n^3 * or / operations.

7. **LU Factorization:** Suppose we reduce A in (3) to the matrix

$$U = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1,n-1} & c_{1n} \\ 0 & c_{22} & \cdots & c_{2,n-1} & c_{2n} \\ & & & & \\ 0 & 0 & \cdots & 0 & c_{nn} \end{pmatrix}$$

by Gaussian elimination using only pivot steps $E_1, \ldots, E_m$, and no row swaps. Then

$$U = E_k E_{k-1} \cdots E_1 A\,.$$

If we write

$$L = E_1^{-1} E_2^{-1} \cdots E_k^{-1}\,,$$

then

$$LU = (E_1^{-1} E_2^{-1} \cdots E_k^{-1})(E_k E_{k-1} \cdots E_1 A) = A. \tag{10}$$

To obtain $L = (\ell_{ij})$, start with $L = I$. Then, during the Gaussian elimination, if E_i is the pivot step

$$\text{row}_k \leftarrow \text{row}_k - c(\text{row}_j)\,,\ k > j\,, \tag{11}$$

we change L by $\ell_{kj} \leftarrow c$. Therefore

$$L = \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 & 0 \\ \ell_{21} & 1 & \cdots & 0 & 0 & 0 \\ & & & & & \\ \ell_{n-1,1} & \ell_{n-1,2} & \cdots & \ell_{n-1,n-2} & 1 & 0 \\ \ell_{n1} & \ell_{n2} & \cdots & \ell_{n,n-2} & \ell_{n,n-1} & 1 \end{pmatrix}$$

is (unit) lower-triangular and it was obtained without any multiply or divide steps. The equation $A = LU$ is called the LU factorization of A, obtained with a cost of $(n^3 - n)/3$ operations. It is not necessary to maintain two separate matrices for L and U because of the pattern of 0's. Thus when the pivot in (11) is done (on columns j to n of A), a zero is created at a_{kj}. In the *compact notation* we would store $\ell_{kj} = c$ at this location. Thus, when Gaussian

elimination is completed, L is stored under the diagonal of A (except $\ell_{ii} = 1$) and U is stored on, and above the diagonal.

Once A has been factored, it is easy to solve $A\underline{x} = \underline{b}$. Note that $A\underline{x} = (LU)\underline{x} = L(U\underline{x}) = L\underline{y}$, where we write $\underline{y}$ for $U\underline{x}$. First we solve $L\underline{y} = \underline{b}$ for $\underline{y}$. This is called *forward substitution* due to the shape of L. Then we solve $U\underline{x} = \underline{y}$ for $\underline{x}$ by backsolving. The cost is n^2 ($n(n-1)/2$ for forward substitution and $n(n+1)/2$ for backsolving).

8. **LUP Factorization:** It is possible that A cannot be factored as in (10), even if A^{-1} exists. However for every invertible A there are matrices L and U as in (10), and an n-vector $\underline{p}$ (the components are a permutation of the first n integers), such that

$$LU = A(\underline{p})\,. \tag{12}$$

The right hand side of (12) denotes A with its rows permuted according to $\underline{p}$. This is called an LUP factorization of A.

To obtain it we reduce A to U, now allowing row swaps (e.g., as required by some pivoting strategy). We maintain L and U in compact notation as A. We keep $\underline{p}$ as an extra column. Initially $p_i = i$; i.e., $\underline{p}$ is the identity permutation. Pivots are handled as in the previous section. They do not change $\underline{p}$. When we swap $\text{row}_j \leftrightarrow \text{row}_k$, j being the pivot row in U and $k > j$, we do the same swap in L and also in $\underline{p}$. Thus in compact notation with $\underline{p}$ as an augmented column, the entire j^{th} and k^{th} rows of A' are swapped (all $n+1$ columns).

Suppose we have an LUP factorization of A and want to solve $A\underline{x} = \underline{b}$. This system is equivalent to $A(\underline{p})\underline{x} = \underline{b}(\underline{p})$. Since $LU = A(\underline{p})$ we first solve $L\underline{y} = \underline{b}(\underline{p})$ for $\underline{y}$ by forward substitution and then backsolve $U\underline{x} = \underline{y}$ for $\underline{x}$.