

Objectives:

- Revisit Gaussian Elimination.
- Pathogenic cases.
- Solving sparse sets of equations.
- **Materials:** Gerald and Wheatley, Chapter 2; Forsythe, et al., section 3.4; Golub and Ortega, Chapters 3.1 and 4.

Summary of Gaussian Elimination

The **augmented coefficient matrix**, A , is made up of the multipliers of x_j on the left-hand-side, and the b_i from the right-hand-side of the set of n linear equations, $\left[\sum_{j=1, n} a_{ij} x_j \right] = b_i$. The “augmented matrix” is an $n \times n + 1$ matrix of $[a_{i1} a_{i2} \cdots a_{in} | b_i]$.

Gaussian elimination proceeds by transforming the augmented coefficient matrix using a series of **elementary row operations** (see below) into an upper triangular matrix. Gaussian elimination requires $n^3/3 + n^2 - n/3$ operations.

Valid Elementary Row Operations

1. Multiply any row of the augmented coefficient matrix by a constant.
2. Add a multiple of one row to a multiple of any other row.
3. Interchange any two rows.

General Gaussian Elimination without Partial Pivoting

```
*** Forward reduction
for k = 1, ..., n-1
  for i = k+1, k+2, ..., n
    l[i,k] = a[i,k]/a[k,k]
    for j = k+1, k+2, ..., n
      a[i,j] = a[i,j] - l[i,k]*a[k,j]
    end j loop
    b[i] = b[i] - l[i,k]*b[k]
  end i loop
end k loop
*** Back substitution
for k = n, n-1, ..., 1
```

```

    for j = k+1, k+2, ..., n
        b[k] = b[k] - a[k,j]*b[j]
    end j loop
    b[k] = b[k] / a[k,k]
end k loop
*** now the solution, x[k], is stored in b[k]

```

Algorithms often store $l[i, k]$ in the original $a[i, k]$ parts of the matrix; $a[i, k]$ would be zeroed by Gaussian elimination, so information is not lost from this. New values are written over the original $a[i, j]$'s, so the original A matrix is lost. You can find FORTRAN versions in LINPACK. Matlab uses a Gaussian elimination routine when you use the back-slash, or the command "mldivide".

Partial Pivoting

Gaussian elimination routines often add a partial pivoting step to minimize rounding error. It is usually accomplished by exchanging rows to minimize $l[i, k] = a[i, k]/a[k, k]$.

Scaling

When the magnitude of the elements of the augmented matrix in one column are much larger (or smaller) than those in other columns, you should scale the system to minimize rounding errors. Simply multiply the values in the offending columns so that they have similar magnitudes. Remember the multipliers used, so that the results can be re-scaled after Gaussian elimination is used to solve the scaled system (see Gerald and Wheatley, page 136, for an example).

LU Factorization

Gaussian elimination transforms the problem of $Ax = b$ into $Ux = b'$. If you accept that there is a lower triangular matrix, L, such that $LU = A$, then the Gaussian elimination method is similar to solving the set of problems: $Lb' = b$; $Ux = b'$. If A is non-singular, the lower triangular matrix does exist and can be constructed using the l_{ij} elements formed in the Gaussian elimination algorithm above, and then putting 1's along the diagonal. Sometimes the problem requires solving $Ax = b$ for several different right-hand-sides. In that case you should retain the L and U matrices, and reuse them for each new right-hand side. The intermediate problem of finding b' such that $Lb' = b$ is efficiently solved once you have L, and then the backward elimination is redone for the new b'.

Gauss - Jordan Elimination

A method that transforms the matrix A into the identity matrix I_n is called **Gauss - Jordan elimination**. It is similar to Gaussian elimination, except the coefficients above the diagonal are zeroed as well as the elements below

the diagonal. The back-substitution step is then not needed; but the method requires more operations ($n^3/2 + n^2 - 7n/2$) than Gaussian elimination.

Pathology in Linear Systems

Not all sets of linear equations have unique solutions. If the number of equations is less than the number of unknowns, then there may exist an infinity of solutions (or no solutions). If the number of equations (n) is greater than the number of unknowns (m), then the system may or may not have a solution. If you can find a solution for a subset of m of the equations in the m -unknowns, you can then test to see if the remaining equations are satisfied. If they are- then they were redundant. If they are not satisfied, then the system is **inconsistent**, and no solution exists.

Even when the number of equations equals the number of unknowns, the system can be inconsistent (no solutions) or redundant (infinite solutions). If this is the case, Gaussian elimination will result in a row that has zeros for the left-hand-side part of the augmented coefficient matrix, and either zero (for a redundant system) or some number (for an inconsistent) system for the right-hand-side part of the augmented coefficient matrix. If an $n \times n$ system does not have a unique solution, then the matrix A is **singular**, otherwise, it is **non-singular**. See Gerald and Wheatley, page 150, for a table summarizing this material. A matrix, A , will be singular if there exists a vector x , where $x_i \neq 0$, and $Ax = 0$. If the only vector that can be multiplied by A to get the zero vector is zero, then A is non-singular.

A system of linear equations can theoretically have a unique solution, but still be **ill-conditioned** so that finding the solution is difficult. Such systems are “close to being singular”. Geometrically, some of lines that form the system are close to being parallel in an ill-conditioned system, so that small changes in solution coefficients (from rounding or measurement error) lead to large differences in the solution that is obtained.

Banded Systems

A banded matrix of order $p+q+1$ has zeros for all elements $a_{i,j}$ where $j > i+p$ or $j < i-q$. So, everything is zero except for elements on the p sub-diagonals, the main-diagonal, and the q super-diagonals. When this is the case, Gaussian elimination requires fewer steps than if the full matrix needed to be considered. Many (most?) problems that arise from using finite differences to solve partial differential equations involve solving a set of linear equations that contain 3 (1-D problems), 5 (2-D problems), or 7 (3-D problems) non-zero diagonals, and are as such, banded matrices. The solution algorithm for these is derived from the Gaussian elimination routine above. The difference is that the inner loops (the i , and the j loops) do not need to evaluate “ $a[i,j] = a[i,j] - l[i,k]*a[k,j]$ ” for the situations where $l[i,k] = 0$ or $a[k,j] = 0$. Those loops are therefore only called

p , and q times for each $k = 1, \dots, n$. The loops would therefore be changed to be “for $i = k + 1, k + 2, \dots, k + p$ ”, and “for $j = k + 1, k + 2, \dots, k + q$ ”. If $p \ll n$ this presents a substantial savings, with operation count $\approx np^2 - \frac{2}{3}p^3$. Partial pivoting is not recommended for banded matrices, because exchanging rows will usually increase the band-width of the matrix.

Gaussian Elimination for Tri-diagonal Systems

Recall that in order to find the cubic-spline of a data set, we derived a set of n -linear equations of the form $a_{i,i-1}x_{i-1} + a_{i,i}x_i + a_{i,i+1}x_{i+1} = b_i$. In matrix form, this would be $Ax = b$ where A is an $n \times n$ matrix, and x and b are both $n \times 1$ column vectors. The matrix A is tri-diagonal, in that $a_{ij} = 0$ for all $|i - j| > 1$. The problem of solving tri-diagonal systems frequently arises in scientific computing, for example, when solving one-dimensional transport or mixing problems, the solution will result from a statement that relates the value at one grid - point (x_i) to those at it's neighboring grid points, $x_{i\pm 1}$.

Solving tri-diagonal systems is an efficient procedure using a specialized version of Gaussian Elimination. Assume that you store your tri-diagonal matrix, and right-hand side in three $n \times 1$ vectors e , d , and c . The matrix, A , is then:

$$\begin{array}{cccccccc}
 d[1] & c[1] & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\
 e[2] & d[2] & c[2] & 0 & 0 & \dots & 0 & 0 & 0 \\
 0 & e[3] & d[3] & c[3] & 0 & \dots & 0 & 0 & 0 \\
 \cdot & \cdot & & & & & \cdot & \cdot & \cdot \\
 \cdot & \cdot & & & & & \cdot & \cdot & \cdot \\
 \cdot & \cdot & & & & & \cdot & \cdot & \cdot \\
 0 & 0 & 0 & 0 & 0 & & e[n-1] & d[n-1] & c[n-1] \\
 0 & 0 & 0 & 0 & 0 & & 0 & e[n] & d[n] .
 \end{array}$$

To solve the tridiagonal system $Ax=b$;

```

*** Gaussian Elimination for Tri-diagonal systems
for k = 1, ..., n-1
    l[k+1] = e[k+1]/d[k]
    b[k+1] = b[k+1] - l[k+1]*b[k]
    d[k+1] = d[k+1] - l[k+1]*c[k]
end k loop

x[n] = b[n]/d[n]
for k = 1, ..., n-1
    x[n-k] = ( b[n-k] - c[n-k]*x[n-k+1] )/d[n-k]
end k loop

```

Again, the solution, $x[i]$ can be stored in $b[i]$, and the $l[k+1]$ could be conveniently stored in the $e[k+1]$, if desired. The operation count for this algorithm is of $O(n)$.