Iterative Methods for Linear Systems.

We now consider iterative methods for solving linear systems of equations. In certain cases these methods are preferred over the direct methods – when the coefficient matrix is large and sparse (has many zeros) they may be more rapid. They may be more economical in terms of core storage requirements of a computer. The simplest method is called the **Jacobi** method. We illustrate the method by a simple example.

$$\begin{aligned} 8x_1 + x_2 - x_3 &= 8 \\ 2x_1 + x_2 + 9x_3 &= 12 \\ x_1 - 7x_2 + 2x_3 &= -4 \end{aligned} \qquad (1)$$

The solution is $x_1=1$, $x_2=1$, $x_3 = 1$. We begin our iterative scheme by solving each equation for one of the variables, choosing, when possible, to solve for the variable with largest coefficient:

$$\begin{aligned} x_1 &= (8 - x_2 + x_3)/8 = 1 - 0.125x_2 + 0.125x_3 && \text{(from 1st eqn)} \\ x_2 &= (-4 - x_1 - 2x_3)/(-7) = 0.571 + 0.143x_1 + 0.286x_3 && \text{(from 3rd eqn)} \\ x_3 &= (12 - 2x_1 - x_2)/9 = 1.333 - 0.222x_1 - 0.111x_2 && \text{(from 2nd eqn)} \end{aligned} \qquad (2)$$

We begin with some initial approximation to the value of the variables. Each component might be taken to be zero, for example). Substituting these into the RHS of the set (2) generates new approximations that are closer to the true value. The new values are then substituted into the RHS to generate a second approximation, and the process is repeated until successive values of each of the variables are sufficiently close, i.e. we use some convergence criterion. For the set of equations given above we get:

		1st	2nd	3th	4th	5th	6th	7th
x_1	0	1.000	1.095	0.995	0.993	1.002	1.001	1.000
x_2	0	0.571	1.095	1.026	0.990	0.998	1.001	1.000
x_3	0	1.333	1.048	0.969	1.000	1.004	1.001	1.000

We may write the original system (1) as Ax=b and write the system matrix A as

$$A = L + D + U \qquad (3)$$

where L is the strictly lower part, D is the main diagonal and U is the strictly upper triangular part of A. Then

$(L + D + U)x = b \Rightarrow Dx = -(L + U)x + b$ or

$$x = -D^{-1}(L + U)x + D^{-1}b \qquad (4)$$

We introduce a subscript n+1 on the LHS and n on the RHS to get

$$x^{(n+1)} = -D^{-1}(L + U)x^{(n)} + D^{-1}b \qquad (5)$$

The superscript refers to the iteration number, and $x^{(0)}$ is the initial guess. The matrix $-D^{-1}(L + U)$ is known as the iteration matrix.

This method is called the **Jacobi method**, also called the method of "simultaneous displacements".

Algorithm for Jacobi Iteration.
To solve a system of N linear equations, rearrange the rows so that the diagonal elements have magnitudes as large as possible, relative to the magnitudes of other coefficients in the same row. Define the rearranged system as Ax=b. Beginning with the vector $x^{(0)}$ as an initial approximation to the solution, compute each component of $x^{(n+1)}$ for i=1,2,…,N by:

$$x_i^{(n+1)} = (b_i - \sum_{\substack{j=1,\\ j\neq i}}^{N} a_{i,j} x_j^{(n)}) / a_{i,i}, \qquad i = 1,2,\ldots,N \tag{6}$$

A sufficient condition for convergence is that

$$|a_{i,i}| \geq \sum_{\substack{j=1,\\ j\neq i}}^{N} |a_{i,j}|, \qquad i = 1,2,\ldots,N \tag{7}$$

When this is true, $x^{(n)}$ will converge to the solution no matter what the initial vector (i.e., guess $x^{(0)}$) is.
A matrix with property (7) is said to be a **diagonally dominant** matrix. These matrices arise in the finite difference approximation of differential equations.

Gauss-Seidel Method.

The x-values of the next iterate are not all calculated "simultaneously" when we perform the Jacobi method. In the above, we calculated the second estimate of x_1 before we did the x_2, and the new values of both x_1 and x_2 were available before we improved the value of x_3. Usually the new values are better than the old, and should be used in preference to the older (poorer) values. When this is done the method is called the Gauss-Seidel method.

We write the equations as for Jacobi in the form of (2). We then proceed to improve each x-value in turn, using always the most recent approximations to the values of the other variables. The rate of convergence is more rapid, as shown by reworking the same examples above.

		1st	2nd	3th	4th	5th
x_1	0	1.000	1.041	0.997	1.001	1.000
x_2	0	0.714	1.014	0.996	1.000	1.000
x_3	0	1.032	0.990	1.002	1.000	1.000

These values were computed using equations (2) in the following way:

$$\begin{aligned} x_1^{(n+1)} &= 1 - 0.125x_2^{(n)} + 0.125x_3^{(n)} && \text{(from 1st eqn)} \\ x_2^{(n+1)} &= 0.571 + 0.143x_1^{(n+1)} + 0.286x_3^{(n)} && \text{(from 3rd eqn)} \\ x_3^{(n+1)} &= 1.333 - 0.222x_1^{(n+1)} - 0.111x_2^{(n+1)} && \text{(from 2nd eqn)} \end{aligned} \tag{8}$$

beginning with $x^{(0)} = (0,0,0)$.

Similar to equation (3) we can more formally write the Gauss-Seidel method as:
$(L + D + U)x = b \Rightarrow (D + L)x = -Ux + b$ or

$$x = -(D + L)^{-1}Ux + (D + L)^{-1}b \tag{9}$$

We introduce a subscript n+1 on the LHS and n on the RHS to get

$$x^{(n+1)} = -(D+L)^{-1}Ux^{(n)} + (D+L)^{-1}b \qquad (10)$$

The matrix $-(D+L)^{-1}U$ is known as the Gauss-Seidel iteration matrix.

Algorithm for Gauss-Seidel Iteration.
Beginning with the vector $x^{(0)}$ as an initial approximation to the solution, compute each component of $x^{(n+1)}$ for i =1,2,…,N by:

$$x_i^{(n+1)} = (b_i - \sum_{j=1}^{i-1} a_{i,j}x_j^{(n+1)} - \sum_{j=i+1}^{N} a_{i,j}x_j^{(n)}) / a_{i,i}, \qquad i = 1,2,\ldots,N$$

A sufficient condition for convergence is that A be diagonally dominant.
In general a necessary and sufficient condition for convergence is that the *spectral radius* of the iteration matrix is less than 1.

The spectral radius $\rho(M)$ is defined as the maximum of the absolute values of the eigenvalues of the iteration matrix M, i.e.,

$$\rho(M) = \max_i |\lambda_i(M)| ,$$

where $\lambda_i(M)$ is an eigenvalue of M.

The MATLAB function **eig**(A) returns the eigenvalues of A.