Iterative Methods (continued). SOR method.

We have considered the Jacobi and Gauss-Seidel iterative methods for solving linear systems. It can be shown that the Gauss-Seidel method will always converge if the Jacobi method converges , and will do so more rapidly. An infinity of iterative methods can be derived, by simply writing A = S-T, where S is non-singular. Then we get

$$(S-T)x = b \Rightarrow Sx = Tx + b$$

$$\Rightarrow x = S^{-1}Tx + S^{-1}b$$

Introducing an index on both sides we get the following iterative method:

$$x^{(n+1)} = S^{-1}Tx^{(n)} + S^{-1}b \qquad (1)$$

The different methods arise by choosing different matrices S and T.

For the Jacobi method: S = D and T = – (L+U).
For Gauss-Seidel method: S = L+D and T = – U.

Another method which is similar to the Gauss-Seidel (GS) method, but which converges faster is called the **successive over-relaxation** (SOR) method.

We can rewrite the algorithm for the GS method in the equivalent form

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

by adding and subtracting $x_i^{(n)}$ from the right-hand side of (2). The residual vector is defined by

$$r = b - Ax \qquad (3)$$

Ideally the residual should be zero, this means we have the exact solution. The term which we add to $x_i^{(n)}$ to get $x_i^{(n+1)}$ is exactly the increment that relaxes the residual to zero, for the current (ith) equation. The method of relaxing the residual to zero is due to a British engineer Richard Southwell. A modification of his method leads to the *over-relaxation* method.

Over-relaxation can be applied to Gauss-Seidel if we add to $x_i^{(n)}$ some multiple of the second term. It can be shown that this multiple should never be more than 2 in magnitude (to avoid divergence) and the optimum over-relaxation factor, denoted by ω lies between 1.0 and 2.0. Our iteration equations take this form , where ω is the over-relaxation factor.

$$x_i^{(n+1)} = x_i^{(n)} + \frac{\omega}{a_{i,i}}(b_i - \sum_{j=1}^{i-1} a_{i,j}x_j^{(n+1)} - \sum_{j=i}^{N} a_{i,j}x_j^{(n)}), \qquad \text{i=1,…,N} \qquad (4)$$

When ω =1, the method reduces to the Gauss-Seidel method.
The table below shows how the convergence rate is influenced by the value of ω for the system:

$$\begin{pmatrix} -4 & 1 & 1 & 1 \\ 1 & -4 & 1 & 1 \\ 1 & 1 & -4 & 1 \\ 1 & 1 & 1 & -4 \end{pmatrix} x = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \qquad (5)$$

starting with an initial estimate of x=0. The exact solution is x = (x_1,x_2,x_3,x_4) = (-1,-1,-1,-1).

ω	No. of iterations to reach error < 10^{-5}.
1.0	24
1.1	18
1.2	13
1.3	11
1.4	14
1.5	18
1.6	24
1.7	35
1.8	55

The error estimate would be of the form $\left\| x^{(n+1)} - x^{(n)} \right\| < 10^{-5}$. You can use the norm function to do this, for the 2-norm (Euclidean) for example, or any other norm.

Exercise: Write a MATLAB program using the SOR method for the above problem. Make ω a variable, so you can change its value, to see how the convergence is affected.

For certain problems the optimum ω can be analytically calculated. E.g. for finite difference approximations of Poisson's equation on a uniform rectangular mesh.
Poisson's equation is one of the most important equations in engineering/physics/computational mathematics: It is given by

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \rho(x,y), \qquad u(x,y) = u_{0,} \text{ on the boundary } \partial\Omega \text{ of the region } \Omega$$

in 2 space dimensions.

In Matrix notation the SOR iteration is given by:

$$\begin{aligned} Dx^{(k+1)} &= Dx^{(k)} + \omega(b - Ax) \\ &= Dx^{(k)} + \omega(b - (L + D + U)x) \\ &= Dx^{(k)} + \omega(b - Lx^{(k+1)} - Dx^{(k)} - Ux^{(k)}) \\ (D + \omega L)x^{(k+1)} &= [(1-\omega)D - \omega U]x^{(k)} + \omega b \qquad \text{or} \end{aligned}$$

$$x^{(k+1)} = (D + \omega L)^{-1}[(1-\omega)D - \omega U]x^{(k)} + (D + \omega L)^{-1}\omega b \qquad (6)$$

The SOR iteration matrix is thus given by

$$M_{SOR} = (D + \omega L)^{-1}[(1-\omega)D - \omega U] \qquad (7)$$

Frequently for problems in 1 dimension the linear systems arising from finite difference methods applied to differential equations are tridiagonal. In this case these iterative methods can be made very efficient, as each row has only 3 nonzero elements, c.f. calculation of cubic splines.

In addition the LU decomposition and solution of a tridiagonal system, is very efficient as both the L and U matrices have only 2 nonzero diagonals. The solution of a system of order N requires only 7N operations. For a full matrix, LU requires of order $N^3/3$ operations.