CHAPTER 3

LINEAR SYSTEMS: ITERATIVE METHODS

SOLVED EXAMPLE PROBLEMS

for

NUMERICAL METHODS FOR SCIENTISTS AND ENGINEERS With Pseudocodes

By Zekeriya ALTAÇ

November 2024

EXAMPLE 3.1: Develop Tailored Iterative Algorithm

Consider the following system of linear equations:

$$\begin{bmatrix} d_1 & a_1 & & & a_n \\ b_2 & d_2 & a_2 & & \\ & \ddots & \ddots & \ddots & \\ & & b_{n-1} & d_{n-1} & a_{n-1} \\ b_1 & & & b_n & d_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-1} \\ c_n \end{bmatrix}$$

Develop an algorithm and write a pseudocode specifically to solve the given linear system by the Gauss-Seidel method.

SOLUTION:

The unknowns are easily isolated from the first to last iteration equations. The Gauss-Seidel iteration equations for this particular problem can be expressed as

$$\begin{aligned} x_1^{(p+1)} &= \left(c_1 - a_1 x_2^{(p)} - a_n x_n^{(p)}\right)/d_1 \\ x_2^{(p+1)} &= \left(c_2 - b_2 x_1^{(p+1)} + a_2 x_3^{(p)}\right)/d_2 \\ \vdots \quad &= \qquad \vdots \\ x_{n-1}^{(p+1)} &= \left(c_{n-1} - b_{n-1} x_{n-2}^{(p+1)} + a_{n-1} x_n^{(p)}\right)/d_{n-1} \\ x_n^{(p+1)} &= \left(c_n - b_1 x_1^{(p+1)} - b_n x_{n-1}^{(p+1)}\right)/d_n \end{aligned}$$

where the superscript (p) denotes the iteration step.

A pseudomodule (SOLVE_3D_SYSTEM) for solving the given linear system with the Gauss-Seidel method is presented below. As input, the module requires the number of equations (n), the diagonals of the coefficient matrix (**b**, **d**, and **a**), the rhs vector (**c**), an initial guess vector (**xo**), a convergence tolerance (ε), and an upper bound for the number of iterations ($maxit$). The outputs of the module are the solution vector (**x**), the total number of iterations performed ($iter$), and the norm of the displacement vector ($error$).

An initial guess for the solution vector, i.e., $xo_i = x_i^{(0)}$, should be prepared prior to calling the module. The code uses the initial guess to update each $x_i = x_i^{(p+1)}$ using the iteration equations presented above, moving sequentially from the first ($i = 1$) to last ($i = n$) equation. At the end of the iteration, a convergence check is carried out to see if the current estimate has converged by comparing the ℓ_2-norm of the displacement vector ($\delta = \|\mathbf{dx}^{(p)}\|_2$) with a small tolerance, i.e., $\delta < \varepsilon$. If convergence is achieved, the iteration is terminated; otherwise, the procedure is repeated until convergence is achieved or the maximum number of iterations is reached, in which case a warning to the user is issued. The code returns the current error and the number of iterations on exit.

It is important to note that this system of linear equations can be solved using a general code with full matrix storage because of a_n and b_1. However, we would have to provide the matrix, including the elements with zeros. Iterative methods allow the user to exclude the zero-valued elements when constructing the iteration equations. This not only reduces the memory requirement but also speeds up the iteration process. This is especially advantageous for very large systems with millions of variables.

Discussion: A code tailored and optimized specifically for certain systems or special matrices offers several advantages over generic implementations. These customizations are often applied to improve the method's efficiency, accuracy, and convergence for particular applications, such as solving large, sparse linear systems often found in engineering simulations and scientific computing.

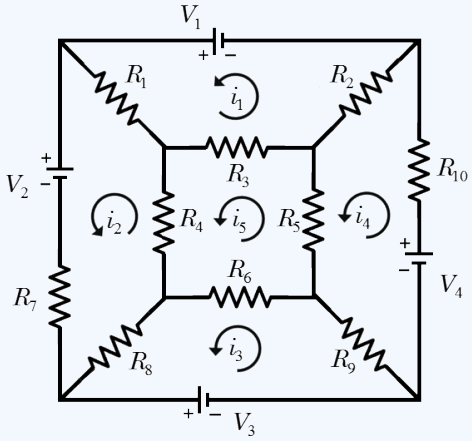
Tailored codes can be structured to take advantage of the computer's memory hierarchy, minimizing cache misses and improving memory locality. By customizing the algorithm for a specific matrix structure, the code can skip unnecessary computations, making each iteration faster. Also, tailored codes can manage floating-point precision more effectively by taking advantage of the matrix structure, reducing rounding errors, and improving accuracy, especially in systems prone to numerical instability.

A tailored code (such as Gauss-Seidel or any other iterative method) is essential in high-performance computing and specialized scientific applications, allowing engineers and scientists to solve large-scale problems faster, with greater accuracy, and with fewer computational resources.

Module SOLVE_3D_SYSTEM (n, $\mathbf{b}$, $\mathbf{d}$, $\mathbf{a}$, $\mathbf{c}$, $\mathbf{xo}$, $\mathbf{x}$, ε, $maxit$, $iter$, $error$)
\ DESCRIPTION: A pseudomodule to solve the system $\mathbf{Tx} = \mathbf{c}$ presented in Example 3.1
\ ON ENTRY
\ n :: Number of equations;
\ $\mathbf{b}$ :: Array of length n providing elements below diagonal;
\ $\mathbf{d}$ :: Array of length n providing diagonal elements;
\ $\mathbf{a}$ :: Array of length n providing elements above diagonal;
\ $\mathbf{c}$ :: Array of length n providing elements on the rhs;
\ $\mathbf{xo}$ :: Array of length n providing initial estimate for the solution;
\ ε :: Tolerance (accuracy in the solution) desired;
\ $maxit$ :: Maximum number of iterations;
\ ON EXIT
\ $\mathbf{x}$ :: Array of length n giving elements below diagonal;
\ $iter$ :: Number of iterations performed to be performed;
\ $error$ :: ℓ_2-norm of the error on exit.
\ USES
\ ENORM :: Function module computing Euclidean-norm of a vector (**Pseudocode 3.1**);
\ ABS :: Built-in absolute value function.
Declare: a_n, b_n, c_n, d_n, x_n, xo_n, dx \ **xo** denotes prior estimates
$p \leftarrow 0$ \ Initialize iteration counter
Repeat \ Start of the iteration loop
  $p \leftarrow p + 1$ \ Count iterations
  $x_1 \leftarrow (c_1 - a_1 * xo_2 - a_n * xo_n)/d_1$
  For $[i = 2, n-1]$ \ Sweep equations from row_2 to row_{n-1}
    $x_i \leftarrow (c_i - b_i * x_{i-1} - a_i * xo_{i+1})/d_i$
  End For \ End of inner loop over j
  $x_n \leftarrow (c_n - b_1 * x_1 - b_n * x_{n-1})/d_n$
  $\mathbf{dx} \leftarrow |\mathbf{x} - \mathbf{xo}|$ \ Find the displacement vector, $\mathbf{dx} = |\mathbf{x}^{(p+1)} - \mathbf{x}^{(p)}|$
  $\delta \leftarrow$ **ENORM**$(n, \mathbf{dx})$ \ Find the ℓ_2-norm of the displacement vector, $\delta = \|\mathbf{dx}^{(p)}\|_2$
Until $[\delta < \varepsilon$ **Or** $p = maxit]$ \ Terminate iteration if $\delta < \varepsilon$ and $p \leqslant maxit$
$error \leftarrow \delta$ \ Set current ℓ_2-norm as error
$iter \leftarrow p$ \ Set current p as $iter$
\ In the case of iteration limit reaching $maxit$ *with no* convergence
If $[p = maxit]$ **Then** \ Issue info and a warning
  Write: "Error=",$error$," iterations failed to converge after",$maxit$,"iterations"
End If
End Module SOLVE_3D_SYSTEM

EXAMPLE 3.2: Applying Jacobi method

Consider the following circuit:



The loop-current method yields the following equations:

Loop -1:
$$R_1(i_1-i_2)+R_3(i_1-i_5)+R_2(i_1-i_4)=V_1$$

Loop -2:
$$R_7 i_2+R_8(i_2-i_3)+R_4(i_2-i_5)+R_1(i_2-i_1)=-V_2$$

Loop -3:
$$R_9(i_3-i_4)+R_6(i_3-i_5)+R_8(i_3-i_2)=-V_3$$

Loop -4:
$$R_{10}i_4+R_2(i_4-i_1)+R_5(i_4-i_5)+R_9(i_4-i_3)=V_4$$

Loop -5:
$$R_3(i_5-i_1)+R_4(i_5-i_2)+R_6(i_5-i_3)+R_5(i_5-i_4)=0$$

Use the given data below to estimate the loop currents using the Jacobi method. Use $i_k^{(0)}=1$ as the initial guess and $\varepsilon=0.5\times10^{-4}$ for convergence tolerance. Given: $R_1=1\,\Omega$, $R_2=R_3=R_{10}=2\,\Omega$, $R_4=R_7=R_8=3\,\Omega$, $R_5=R_9=4\,\Omega$, $R_6=5\,\Omega$, $V_1=23\,V$, $V_2=18$ V, $V_3=24$ V, $V_4=48$ V.

SOLUTION:

Substituting the data provided leads to the following linear system:

$$\begin{bmatrix} 5 & -1 & 0 & -2 & -2 \\ -1 & 10 & -3 & 0 & -3 \\ 0 & -3 & 12 & -4 & -5 \\ -2 & 0 & -4 & 12 & -4 \\ -2 & -3 & -5 & -4 & 14 \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \\ i_3 \\ i_4 \\ i_5 \end{bmatrix} = \begin{bmatrix} 23 \\ -18 \\ -24 \\ 48 \\ 0 \end{bmatrix}$$

The Jacobi iteration equations are found as follows:

$$\begin{aligned}
i_1^{(p+1)} &= \frac{1}{5}\left(i_2^{(p)}+2i_4^{(p)}+2i_5^{(p)}+23\right),\\
i_2^{(p+1)} &= \frac{1}{10}\left(i_1^{(p)}+3i_3^{(p)}+3i_5^{(p)}-18\right),\\
i_3^{(p+1)} &= \frac{1}{12}\left(3i_2^{(p)}+4i_4^{(p)}+5i_5^{(p)}-24\right),\\
i_4^{(p+1)} &= \frac{1}{6}\left(i_1^{(p)}+2i_3^{(p)}+2i_5^{(p)}+24\right),\\
i_5^{(p+1)} &= \frac{1}{14}\left(2i_1^{(p)}+3i_2^{(p)}+5i_3^{(p)}+4i_4^{(p)}\right).
\end{aligned}$$

The first iteration yields

$$\begin{aligned}
i_1^{(1)} &= \frac{1}{5}\left(i_2^{(0)}+2i_4^{(0)}+2i_5^{(0)}+23\right)=\frac{1}{5}\left(1+2(1)+2(1)+23\right)=5.6\\
i_2^{(1)} &= \frac{1}{10}\left(i_1^{(0)}+3i_3^{(0)}+3i_5^{(0)}-18\right)=\frac{1}{10}\left(1+3(1)+3(1)-18\right)=-1.1\\
i_3^{(1)} &= \frac{1}{12}\left(3i_2^{(0)}+4i_4^{(0)}+5i_5^{(0)}-24\right)=\frac{1}{12}\left(3(1)+4(!)+5(1)-24\right)=-1\\
i_4^{(1)} &= \frac{1}{6}\left(i_1^{(0)}+2i_3^{(0)}+2i_5^{(0)}+24\right)=\frac{1}{6}\left(1+2(1)+2(1)+24\right)=4.833333\\
i_5^{(1)} &= \frac{1}{14}\left(2i_1^{(0)}+3i_2^{(0)}+5i_3^{(0)}+4i_4^{(0)}\right)=\frac{1}{14}\left(2(1)+3(1)+5(1)+4(1)\right)=1
\end{aligned}$$

which yields $\|\mathbf{i}^{(1)} - \mathbf{i}^{(0)}\|_\infty = \delta^{(1)} = 4.6$.

Table 3.1: Iteration progress of Example 3.2.

p	$i_1^{(p)}$	$i_2^{(p)}$	$i_3^{(p)}$	$i_4^{(p)}$	$i_5^{(p)}$	$\delta^{(p)}$	$\delta^{(p)}/\delta^{(p-1)}$
0	1	1	1	1	1		
1	5.6	-1.1	-1.	4.83333	1	4.6	
2	6.71333	-1.24	-0.24722	4.93333	1.58810	1.113333	0.24203
3	6.96057	-0.72641	-0.00385	5.56585	2.01456	0.632513	0.56813
4	7.48688	-0.50073	0.51308	5.83033	2.42758	0.526312	0.83210
5	7.80302	-0.16911	0.82975	6.22803	2.81131	0.397700	0.75564
6	8.18191	0.07262	1.20511	6.51419	3.15426	0.378896	0.95272
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
107	11.9994	2.99953	4.99939	9.99944	6.99938	5.56×10^{-5}	0.91711
108	11.9994	2.99957	4.99944	9.99949	6.99944	5.10×10^{-5}	0.91711
109	11.9995	2.99961	4.99949	9.99953	6.99948	4.68×10^{-5}	0.91711

This set of linear equations converged to an approximate solution in 109 iterations. Here, $\delta^{(p)}$ denotes the maximum absolute error at the p'th iteration step (i.e., $\delta^{(p)} = \|\mathbf{i}^{(p)} - \mathbf{i}^{(p-1)}\|_\infty$), and the ratio $\delta^{(p)}/\delta^{(p-1)}$ approaches the convergence rate for sufficiently large p; in this case, $\delta^{(p)}/\delta^{(p-1)} \to 0.91711$ as $p \to \infty$.

Discussion: In general, the convergence of iterative methods in solving linear systems is affected by several key factors that require considerable mathematical analysis, such as the condition number, structure, eigenvalue spread of the coefficient matrix, the choice of the iterative method, acceleration, and preconditioning.

The simplest and easiest test that can be applied to determine whether a linear system will converge with the Jacobi method is the *diagonal dominance* test. The Jacobi method converges if the coefficient matrix is *diagonally dominant*, meaning:

$$|a_{ii}| \geqslant \sum_{\substack{j=1 \\ i \neq j}}^{n} |a_{ij}| \text{ for } i = 1, 2, ..., n$$

for all i, i.e., for each row, the absolute value (also called *magnitude*) of the diagonal element is greater than or equal to the sum of the absolute values of the rest of the elements of that row. Notice that this criterion is satisfied for all rows of the given linear system.

The diagonal dominance is a *sufficient condition* for convergence, i.e., it is *not a necessary condition*. A linear system can still converge to its solution if the coefficient matrix is *not* diagonally dominant. In other words, the Jacobi method does *not* always require *strict diagonal dominance* for convergence. The diagonal dominance is a straightforward and easily checked criterion that guarantees convergence for the Jacobi method. However, other factors, like the spectral radius of the iteration matrix (i.e., $\rho(\mathbf{M}_J) < 1$) and the matrix's positive definiteness, can also lead to convergence even if a coefficient matrix is *not diagonally dominant*. In general, matrices that are symmetric positive definite allow the Jacobi method to converge even without diagonal dominance. This is because such matrices yield a spectral radius of less than 1.

EXAMPLE 3.3: Applying Gauss-Seidel and SOR Methods

Solve the linear system of equations given in Example 3.2 first with the Gauss-Seidel method and then apply the SOR method for a number of ω values to determine the ω_{opt}.

SOLUTION:

The Gauss-Seidel iteration equations are obtained as follows:

$$
\begin{aligned}
i_1^{(p+1)} &= \frac{1}{5}\left(i_2^{(p)} + 2i_4^{(p)} + 2i_5^{(p)} + 23\right), \\
i_2^{(p+1)} &= \frac{1}{10}\left(i_1^{(p+1)} + 3i_3^{(p)} + 3i_5^{(p)} - 18\right), \\
i_3^{(p+1)} &= \frac{1}{12}\left(3i_2^{(p+1)} + 4i_4^{(p)} + 5i_5^{(p)} - 24\right), \\
i_4^{(p+1)} &= \frac{1}{6}\left(i_1^{(p+1)} + 2i_3^{(p+1)} + 2i_5^{(p)} + 24\right), \\
i_5^{(p+1)} &= \frac{1}{14}\left(2i_1^{(p+1)} + 3i_2^{(p+1)} + 5i_3^{(p+1)} + 4i_4^{(p)}\right).
\end{aligned}
$$

The first iteration gives

$$
\begin{aligned}
i_1^{(1)} &= \frac{1}{5}\left(i_2^{(0)} + 2i_4^{(0)} + 2i_5^{(0)} + 23\right) = \frac{1}{5}\left(1 + 2(1) + 2(1) + 23\right) = 5.6 \\
i_2^{(1)} &= \frac{1}{10}\left(i_1^{(0)} + 3i_3^{(0)} + 3i_5^{(0)} - 18\right) = \frac{1}{10}\left(5.6 + 3(1) + 3(1) - 18\right) = -0.64 \\
i_3^{(1)} &= \frac{1}{12}\left(3i_2^{(0)} + 4i_4^{(0)} + 5i_5^{(0)} - 24\right) = \frac{1}{12}\left(3(-0.64) + 4(1) + 5(1) - 24\right) = -1.41 \\
i_4^{(1)} &= \frac{1}{6}\left(i_1^{(0)} + 2i_3^{(0)} + 2i_5^{(0)} + 24\right) = \frac{1}{6}\left(5.6 + 2(-1.41) + 2(1) + 24\right) = 4.79667 \\
i_5^{(1)} &= \frac{1}{14}\left(2i_1^{(0)} + 3i_2^{(0)} + 5i_3^{(0)} + 4i_4^{(0)}\right) \\
&= \frac{1}{14}\left(2(5.6) + 3(-0.64) + 5(-1.41) + 4(4.79667)\right) = 1.52976
\end{aligned}
$$

which yields $\|\mathbf{i}^{(1)} - \mathbf{i}^{(0)}\|_\infty = \delta^{(1)} = 4.6$. This procedure is repeated until the estimates converge to an approximate solution within a specified tolerance (in this case, $\delta^{(p)} < 5 \times 10^{-5}$). The iteration progress is depicted in Table 3.2.

Table 3.2: Iteration progress with Gauss-Seidel method.

p	$i_1^{(p)}$	$i_2^{(p)}$	$i_3^{(p)}$	$i_4^{(p)}$	$i_5^{(p)}$	$\delta^{(p)}$	$\delta^{(p)}/\delta^{(p-1)}$
0	1	1	1	1	1		
1	5.60000	-0.64000	-1.41000	4.79667	1.52976	4.600	
2	7.00257	-1.06381	-0.02966	5.66713	2.38099	1.403	0.30490
3	7.60648	-0.33395	0.79763	6.32729	3.10775	0.827	0.58984
4	8.30722	0.20234	1.45457	6.90531	3.72254	0.701	0.84702
5	8.89161	0.64230	2.01340	7.39392	4.23948	0.584	0.83396
6	9.38182	1.01405	2.48460	7.80499	4.67491	0.490	0.83885
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
58	11.99965	2.99974	4.99983	9.99971	6.99969	6.51×10^{-5}	0.84226
59	11.99971	2.99978	4.99986	9.99975	6.99974	5.49×10^{-5}	0.84226
60	11.99975	2.99981	4.99988	9.99979	6.99978	4.62×10^{-5}	0.84226

It is observed that the number of iterations required to satisfy the same convergence criterion decreased to 60 iterations with the Gauss-Seidel method. This corresponds to a saving of approximately 45% in the number of iterations (as well as cpu-time) compared to the Jacobi method, which required 109 iterations. We also note that the convergence rate, i.e., the ratio $\delta^{(p)}/\delta^{(p-1)} \to 0.84226$ as $p \to \infty$.

The SOR iteration equations take the following form:

$$
\begin{aligned}
i_1^{(p+1)} &= (1-\omega)\, i_1^{(p)} + \frac{\omega}{5}\left(i_2^{(p)} + 2i_4^{(p)} + 2i_5^{(p)} + 23\right)\\
i_2^{(p+1)} &= (1-\omega)\, i_2^{(p)} + \frac{\omega}{10}\left(i_1^{(p+1)} + 3i_3^{(p)} + 3i_5^{(p)} - 18\right)\\
i_3^{(p+1)} &= (1-\omega)\, i_3^{(p)} + \frac{\omega}{12}\left(3i_2^{(p+1)} + 4i_4^{(p)} + 5i_5^{(p)} - 24\right)\\
i_4^{(p+1)} &= (1-\omega)\, i_4^{(p)} + \frac{\omega}{6}\left(i_1^{(p+1)} + 2i_3^{(p+1)} + 2i_5^{(p)} + 24\right)\\
i_5^{(p+1)} &= (1-\omega)\, i_5^{(p)} + \frac{\omega}{14}\left(2i_1^{(p+1)} + 3i_2^{(p+1)} + 5i_3^{(p+1)} + 4i_4^{(p)}\right)
\end{aligned}
$$

The SOR iteration equations were solved for ω values from $\omega = 1$ to 1.9 with $\Delta\omega = 0.05$ steps. The numerical solutions were obtained using the SOR method for $\Delta\omega = 0.05$ steps from $\omega = 1$ to 1.90, and the number of iterations required for convergence versus ω was plotted and presented in Figure 3.1. The optimum value is $\omega \approx 1.45$ with 20 iterations (one third of that required by the Gauss-Seidel method).

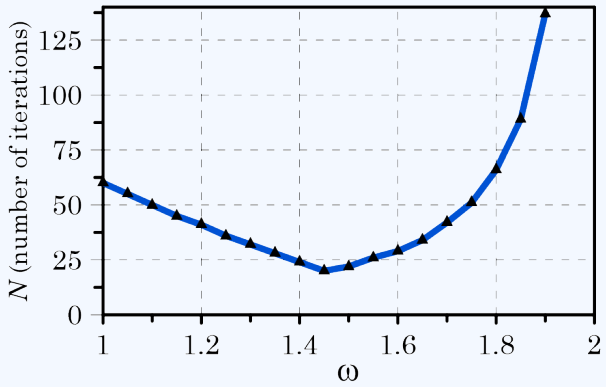


Figure 3.1: The variation of the number of iterations (N) with acceleration parameter (ω).

Discussion: The convergence of the SOR method depends primarily on the choice of ω (relaxation parameter), which plays a central role in shaping the convergence rate. The optimal choice of ω typically lies in the range $1 \leqslant \omega < 2$, and it reduces to the Gauss-Seidel method for $\omega = 1$. For $\omega > 1$, the method accelerates convergence, but if ω is chosen too large or too small, it may slow down convergence or even lead to divergence. This is because the SOR iteration matrix, $\mathbf{M}_\omega$, is a function of ω (*see* Eq. (3.17)). As a result, the spectral radius of $\mathbf{M}_\omega$ depends on the value of ω as well as the properties of the system (e.g., the matrix $\mathbf{A}$). If the coefficient matrix $\mathbf{A}$ is diagonally dominant or positive definite, the SOR method tends to converge faster. For symmetric positive definite matrices, SOR generally converges for appropriate values of ω. For non-symmetric or indefinite matrices, the convergence may be slower or the method may fail to converge.

EXAMPLE 3.4: Determine Iteration Matrix and Spectral Radius

Consider the linear system of equations given in Example 3.2. For the linear system in question, (a) find Jacobi and Gauss-Seidel iteration matrices; (b) estimate the spectral radius and convergence rates for each method and discuss their convergence behavior; and (c) determine the theoretical optimum relaxation parameter, compare it with the finding in **EXAMPLE 3.3** and discuss the result.

SOLUTION:

(a) We split the coefficient matrix as **A**=**-L**+**D**-**U**, where

$$\mathbf{U}=\begin{bmatrix}0&1&0&2&2\\0&0&3&0&3\\0&0&0&4&5\\0&0&0&0&4\\0&0&0&0&0\end{bmatrix},\quad \mathbf{L}=\begin{bmatrix}0&0&0&0&0\\1&0&0&0&0\\0&3&0&0&0\\2&0&4&0&0\\2&3&5&4&0\end{bmatrix},\quad \mathbf{D}=\begin{bmatrix}5&0&0&0&0\\0&10&0&0&0\\0&0&12&0&0\\0&0&0&12&0\\0&0&0&0&14\end{bmatrix}$$

The Jacobi and Gauss-Seidel iteration matrices are defined as

$$\mathbf{M}_J=(\mathbf{L}+\mathbf{U})\mathbf{D}^{-1}\qquad\text{and}\qquad \mathbf{M}_{GS}=(\mathbf{L}+\mathbf{D})\mathbf{U}$$

which yield

$$\mathbf{M}_J=\begin{bmatrix}0&\frac{1}{5}&0&\frac{2}{5}&\frac{2}{5}\\\frac{1}{10}&0&\frac{3}{10}&0&\frac{3}{10}\\0&\frac{1}{4}&0&\frac{1}{3}&\frac{5}{12}\\\frac{1}{6}&0&\frac{1}{3}&0&\frac{1}{3}\\\frac{1}{7}&\frac{3}{14}&\frac{5}{14}&\frac{2}{7}&0\end{bmatrix}\quad\text{and}\quad \mathbf{M}_{GS}=\begin{bmatrix}0&\frac{1}{5}&0&\frac{2}{5}&\frac{2}{5}\\0&\frac{1}{50}&\frac{3}{10}&\frac{1}{25}&\frac{17}{50}\\0&\frac{1}{200}&\frac{3}{40}&\frac{103}{300}&\frac{301}{600}\\0&\frac{7}{200}&\frac{1}{40}&\frac{163}{900}&\frac{1021}{1800}\\0&\frac{5}{112}&\frac{11}{112}&\frac{121}{504}&\frac{475}{1008}\end{bmatrix}$$

(b) To estimate the convergence rates of both methods, we need to find the spectral radius (maximum magnitude eigenvalue) of the iteration matrices. In order to determine the spectral radii, we will obtain the characteristic polynomials and then find the largest absolute roots (i.e., max. magnitude eigenvalues). In this regard, the characteristic polynomials and their roots are found as

$$\text{Jacobi:}\quad \lambda^5-\frac{4021\lambda^3}{6300}-\frac{2371\lambda^2}{12600}+\frac{13\lambda}{8400}+\frac{1}{2520}=0$$

$$\lambda=-0.528593,\ \lambda=-0.389631,\ \lambda=-0.0452801,\ \lambda=0.0463979,$$
$$\lambda=0.917106$$

$$\text{Gauss-Seidel:}\quad \lambda\left(\lambda^4-\frac{18833\lambda^3}{25200}-\frac{319\lambda^2}{5040}-\frac{13\lambda}{1008}-\frac{1}{1050}\right)=0$$

$$\lambda=0,\ \lambda=-0.0746727,\ \lambda=0.842262,\ \lambda=-0.01012\pm0.12264\,i$$

The spectral radius (the largest absolute eigenvalue) for the Jacobi and Gauss-Seidel iteration matrices are $\rho(\mathbf{M}_J)=0.917106$ and $\rho(\mathbf{M}_{GS})=0.842262$, respectively. (For complex eigenvalues, the magnitude or modulus is evaluated.) This means that both iterative methods will converge to the true solution as the number of iterations increases. Also note that $\rho(\mathbf{M}_{GS})\cong\rho^2(\mathbf{M}_J)$.

Notice that, in Examples 3.2 and 3.3, both Jacobi and Gauss-Seidel methods converge after a sufficient number of iterations, specifically 109 and 60, respectively. The spectral radii of the Jacobi and Gauss-Seidel methods were also estimated as 0.91711 and 0.84226 during the iterations, as shown in the last columns of Tables 3.1 and 3.2, respectively.

The theoretical rate of convergence of the methods depends on how close the spectral radii are to 1. The convergence rates, $R = -\log \rho(\mathbf{M})$, are found as 0.0376 and 0.07455 for the Jacobi and Gauss-Seidel methods, respectively. The Gauss-Seidel method converges faster than the Jacobi method since it has a smaller convergence rate or smaller spectral radius. Moreover, we can estimate the number of iteration steps required to reduce the maximum error by the factor 10 with $1/R$, which yields ≈ 27 and ≈ 14 for Jacobi and Gauss-Seidel methods. These values are pretty consistent with the total number of iterations.

For a numerical analyst, the rate of convergence in the optimum case is of interest. The following theoretical formula for the ω_{opt} is obtained by minimizing the spectral radius of the iteration matrix, $\rho(\mathbf{M}_\omega)$, in the context of diagonally dominant or symmetric positive definite matrices:

$$\omega_{\text{opt}} \approx \frac{2}{1 + \sqrt{1 - \rho(\mathbf{M}_{GS})}}$$

Note that $\rho^2(\mathbf{M}_J)$ is replaced with $\rho(\mathbf{M}_{GS})$ in the above equation. For this example, the estimation of the optimal value yields $\omega_{\text{opt}} \cong 1.43$, which is in excellent agreement with the value ($\approx$1.45) obtained by numerical experiments in Example 3.2.

Discussion: In this example, it was relatively easy to find the $\mathbf{M}_J$ and $\mathbf{M}_{GS}$ iteration matrices and their eigenvalues since the system of equations in question was small (i.e., 5×5). However, estimating the ω_{opt} in large systems using the foregoing analysis can be challenging. While the theoretical formulation for ω_{opt} is relatively straightforward, the task of calculating the largest or all eigenvalues of large linear systems, as encountered in many problems of engineering importance, is more difficult. The optimal acceleration value can be found by experimentation, as demonstrated in Example 3.3. Nevertheless, this approach can also be time-consuming, especially when a system is very large and requires many trials to determine ω_{opt}.

For very large problems, there are adaptive algorithms that can adjust ω during the iterations depending on the observed convergence behavior. For example, if convergence is slow, ω can be increased; if convergence starts to stall, ω can be decreased. Such adaptive approaches require monitoring the residual or the error reduction at each step. The optimum value estimation algorithm described in Section 3.3.2, however, adjusts ω only once. With this algorithm, a pre-specified number of iterations (usually $\approx$10-15) are carried out with $\omega = 1$, and the spectral radius of the Gauss-Seidel method is estimated. After calculating the optimal value using the formula above, subsequent iterations are carried out with the estimated ω value. Note that the estimate for the spectral radius of the Gauss-Seidel method after 6 iterations is found to be 0.83885 (*see* Table 3.2), which gives $\omega = 2/(1 + \sqrt{1 - 0.83885}) = 1.427$. This value is pretty good agreement with the optimal value calculated earlier, and the subsequent iterations can then be performed with $\omega = 1.427$. Here, the critical point is to estimate the spectral radius of the Gauss-Seidel iteration matrix with sufficient accuracy.

EXAMPLE 3.5: Convergence Properties of Linear Systems

Consider the following linear system of equations, whose coefficient matrix is *not* diagonally dominant. Determine if the system converges for the Jacobi, or Gauss-Seidel, or both methods.

$$\begin{bmatrix} 5 & -2 & 1 & -3 \\ -1 & 4 & -2 & 2 \\ 2 & -1 & 5 & -4 \\ -2 & 3 & 4 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 15 \\ -13 \\ 21 \\ -5 \end{bmatrix}$$

SOLUTION:

We split the coefficient matrix as **A**=**-L**+**D**-**U**, where

$$\mathbf{U} = \begin{bmatrix} 0 & 2 & -1 & 3 \\ 0 & 0 & 2 & -2 \\ 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{L} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 \\ 2 & -3 & -4 & 0 \end{bmatrix}, \quad \mathbf{D} = \begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$$

The Jacobi and Gauss-Seidel iteration matrices are constructed as follows:

$$\mathbf{M}_J = (\mathbf{L} + \mathbf{U})\mathbf{D}^{-1} \qquad \text{and} \qquad \mathbf{M}_{GS} = (\mathbf{L} + \mathbf{D})\mathbf{U}$$

which yield

$$\mathbf{M}_J = \begin{bmatrix} 0 & \frac{2}{5} & -\frac{1}{5} & \frac{3}{5} \\ \frac{1}{4} & 0 & \frac{1}{2} & -\frac{1}{2} \\ -\frac{2}{5} & \frac{1}{5} & 0 & \frac{4}{5} \\ \frac{1}{2} & -\frac{3}{4} & -1 & 0 \end{bmatrix} \quad \text{and} \quad \mathbf{M}_{GS} = \begin{bmatrix} 0 & \frac{2}{5} & -\frac{1}{5} & \frac{3}{5} \\ 0 & \frac{1}{10} & \frac{9}{20} & -\frac{7}{20} \\ 0 & -\frac{7}{50} & \frac{17}{100} & \frac{49}{100} \\ 0 & \frac{53}{200} & -\frac{243}{400} & \frac{29}{400} \end{bmatrix}$$

The characteristic polynomials and their roots are found as

$$\text{Jacobi:} \quad \lambda^4 - \frac{31\lambda^2}{200} + \frac{137\lambda}{400} - \frac{3}{25} = 0$$

$$\lambda = 0.35998, \ \lambda = -0.850597, \ \lambda = 0.24531 \pm 0.57596\, i$$

$$\text{Gauss-Seidel:} \quad \lambda\left(\lambda^3 - \frac{137\lambda^2}{400} + \frac{49\lambda}{100} - \frac{2}{25}\right) = 0$$

$$\lambda = 0, \ \lambda = 0.173657, \ \lambda = 0.08442 \pm 0.67346\, i$$

The spectral radius for the Jacobi and Gauss-Seidel iteration matrices are found to be $\rho(\mathbf{M}_J) = 0.8506$ and $\rho(\mathbf{M}_{GS}) = |\lambda|_{max} = 0.6787$, respectively. Since the spectral radii of both iteration matrices are less than 1, both methods will converge the solution of the system of equations with a sufficient number of iterations.

Discussion: The convergence criterion for a system of linear equations involves spectral radius of the iteration matrix **M**, i.e., $\rho(\mathbf{M})$. The spectral radius of an iteration matrix is defined as its largest absolute eigenvalue, i.e., $\rho(\mathbf{M}) = |\lambda|_{max}$. An iterative method is guaranteed to converge if the spectral radius of the iteration matrix is strictly less than 1; that is, $\rho(\mathbf{M}) < 1$. On the other hand, the criterion based on the *diagonal dominance* of the coefficient matrix **A** is the *sufficient condition* but *not a necessary condition*.

EXAMPLE 3.6: Implementing Conjugate Gradient Method

Consider the following system of linear equations: $\mathbf{Ax}=\mathbf{b}$, where

$$\mathbf{A} = \begin{bmatrix} 8 & 1 & 2 & 1 \\ 1 & 6 & 1 & 2 \\ 2 & 1 & 7 & 1 \\ 1 & 2 & 1 & 8 \end{bmatrix}, \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 14 \\ 19 \\ 18 \\ 1 \end{bmatrix}$$

Apply the Conjugate Gradient method (CGM) to find the approximate solution of the given linear system. Use $\mathbf{x}^{(0)}=\mathbf{0}$ for the initial value and $\|\mathbf{r}\|_2 < 10^{-5}$ as the convergence criterion.

SOLUTION:

We note that the coefficient matrix is *symmetric* and *positive definite*. Starting with the initial guess, $\mathbf{x}^{(0)} = \mathbf{0}$, the initial residual is found as

$$\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)} = [\,14 \quad 19 \quad 18 \quad 1\,]^T$$

Setting $\mathbf{d}^{(0)} = \mathbf{r}^{(0)}$, we find

$$\rho^{(0)} = (\mathbf{r}^{(0)}, \mathbf{r}^{(0)}) = 14^2 + 19^2 + 18^2 + 1^2 = 882$$
$$\mathbf{c}^{(0)} = \mathbf{A}\mathbf{d}^{(0)} = [\,168 \quad 148 \quad 174 \quad 78\,]^T$$
$$(\mathbf{d}^{(0)}, \mathbf{c}^{(0)}) = (14)(168) + (19)(148) + (18)(174) + (1)(78) = 8374$$

Next, we calculate the step length and the current estimate as

$$\alpha^{(0)} = \frac{\rho^{(0)}}{(\mathbf{d}^{(0)}, \mathbf{c}^{(0)})} = \frac{882}{8374} = 0.10533$$
$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \alpha^{(0)}\mathbf{d}^{(0)} = [\,1.4746 \quad 2.0012 \quad 1.8959 \quad 0.10533\,]^T$$

The residual associated with the current estimate is obtained as follows:

$$\mathbf{r}^{(1)} = \mathbf{r}^{(0)} - \alpha^{(0)}\mathbf{c}^{(0)} = [\,-3.6948 \quad 3.4118 \quad -0.3267 \quad -7.2154\,]^T$$

with $\rho^{(1)} = (\mathbf{r}^{(1)}, \mathbf{r}^{(1)}) = 77.461$ and $\|\mathbf{r}^{(1)}\|_2 = \sqrt{\rho^{(1)}} = 8.8012 > \varepsilon$. Then, the conjugation parameter is obtained as

$$\beta^{(0)} = \frac{\rho^{(1)}}{\rho^{(0)}} = \frac{77.461}{882} = 0.0878$$

This procedure is repeated in the same way until convergence is achieved in four iterations. The results are presented in Table 3.3. It should be noted that the ℓ_2-norm of the residual vector quickly converges to zero.

The convergence of the CG method depends on several factors, including the properties of the matrix $\mathbf{A}$, particularly its eigenvalue spectrum (the set of eigenvalues of $\mathbf{A}$). If the eigenvalues of the coefficient matrix are clustered, CGM will converge faster. On the other hand, if the eigenvalues are spread out (i.e., the matrix is *ill-conditioned*), the convergence rate becomes slower.

The condition number for symmetric and positive definite matrices is the ratio of the largest to the smallest eigenvalue (i.e., $\kappa(\mathbf{A}) = \lambda_{max}/\lambda_{min}$) and provides an insight on the eigenvalue spread of the coefficient matrix $\mathbf{A}$. The rate of convergence of the CGM is influenced by the condition number, $\kappa(\mathbf{A})$ (*see* Eq. (3.47)). In this example, the eigenvalues of $\mathbf{A}$, which is a symmetric and positive definite matrix, are λ =11.3592, 7.50839, 5.42642, and 4.70597, which are fairly spread out. The condition number is found to be $\kappa(\mathbf{A}) = 11.3592/4.70597 = 2.4138$.

Table 3.3: Iteration progress of CGM for the Example 3.6.

p	$\mathbf{x}^{(p)}$	$\mathbf{d}^{(p)}$	$\mathbf{r}^{(p)}$	$\rho^{(p)}$	$\alpha^{(p)}$	$\beta^{(p)}$	$\|\mathbf{r}^{(p)}\|_2$
0	$\begin{bmatrix}0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}14\\19\\18\\10\end{bmatrix}$	$\begin{bmatrix}14\\19\\18\\1\end{bmatrix}$	882	0.10533	0.0878	29.698
1	$\begin{bmatrix}1.4746\\2.0012\\1.8959\\0.1053\end{bmatrix}$	$\begin{bmatrix}-2.4652\\5.0804\\1.2541\\-7.1276\end{bmatrix}$	$\begin{bmatrix}-3.6948\\3.4118\\-0.3267\\-7.2154\end{bmatrix}$	77.461	0.1653	0.02645	8.8012
2	$\begin{bmatrix}1.0671\\2.8409\\2.1031\\-1.0727\end{bmatrix}$	$\begin{bmatrix}-0.57652\\1.06430\\-0.59123\\0.54129\end{bmatrix}$	$\begin{bmatrix}-0.51132\\0.92989\\-0.62440\\0.72979\end{bmatrix}$	2.0486	0.14464	0.00922	1.4313
3	$\begin{bmatrix}0.98372\\2.9948\\2.0176\\-0.99444\end{bmatrix}$	$\begin{bmatrix}0.08928\\0.02842\\-0.09669\\-0.03052\end{bmatrix}$	$\begin{bmatrix}0.09459\\0.01861\\-0.09124\\-0.03550\end{bmatrix}$	0.01888	0.18235	0	0.1374
4	$\begin{bmatrix}1\\3\\2\\-1\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\end{bmatrix}$	0	0	0	0

To find the condition number of large matrices, computing all eigenvalues or performing direct methods like QR decomposition can become costly in terms of both time and memory, especially if the matrix $\mathbf{A}$ is dense. However, rather than computing all eigenvalues, the largest and smallest eigenvalues can be obtained iteratively by the *Power method* (*see* Section 11.2). For large matrices, one may require efficient and fast modules to numerically compute the desired eigenvalues.

Discussion: The CGM, in the absence of truncation errors, will find the exact solution after at most n iterations, where n is the size of the matrix. This is a key result because, as observed in this example problems, the CGM is guaranteed to converge in a finite number of steps, unlike other iterative methods. For matrices where the eigenvalues are well-clustered together (i.e., $\kappa(\mathbf{A}) \to 1$), the CGM does converge quickly, even in fewer than n steps. However, if the coefficient matrix $\mathbf{A}$ is ill-conditioned, the method may lead to many iterations to achieve a sufficiently accurate solutions. In practice, a preconditioning procedure is often used to improve the convergence rate. A well chosen preconditioner can significantly reduce the condition number of the transformed system, leading to much faster convergence.

EXAMPLE 3.7: Preconditioning Ill-Conditioned Linear Systems

Consider the following system of linear equations (**Ax=b**):

$$\begin{bmatrix} 25 & 23 & 21 & 19 & 170 \\ 23 & 25 & 23 & 22 & 21 \\ 21 & 23 & 23 & 20 & 19 \\ 19 & 22 & 20 & 22 & 60 \\ 170 & 21 & 19 & 60 & 10021 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 174 \\ 20 \\ 20 \\ 55 \\ 10129 \end{bmatrix}$$

(a) Determine the condition number of matrix $\mathbf{A}$, (b) Employ a diagonal matrix preconditioner $\mathbf{D}$ where $d_{ii} = 1/a_{ii}$ to obtain $\mathbf{M} = \mathbf{DAD}^T$, and (c) calculate $\kappa(\mathbf{M})$.

SOLUTION:

(a) The eigenvalues of $\mathbf{A}$ are found as

$$\lambda = \{10024.3,\ 86.1195,\ 3.93298,\ 1.58865,\ 0.0204095\}$$

from which the spectral condition number is found as $\kappa(\mathbf{A}) = 10024.3/0.02041 = 491160$. The condition number is usually easier to compute using linear algebra modules. In this regard, the spectral norm can be computed efficiently using singular value decomposition (SVD) or through methods like the *Power iteration* or the Lanczos algorithm.

Other forms of the condition numbers can be estimated using ℓ_1, ℓ_∞, and ℓ_F-based norms, which are found to be $\kappa_1 = \kappa_\infty = 706020$ and $\kappa_F = 491225$. Computing the ℓ_1 and ℓ_∞-based norm of a matrix is straightforward, as it just requires summing the absolute values of the columns or rows. However, computing the same norms of the inverse matrix can be computationally expensive due to requiring matrix inversion, which is generally more expensive than just computing the norm of the matrix. The Frobenius norm is relatively easy to compute, as it only requires summing the squares of the entries. However, the norm of the inverse matrix still requires computing the $\mathbf{A}^{-1}$, which may be costly for large matrices. An approximate value for the condition number based on Frobenius norm can be estimated by Eq. (2.33), which can be efficiently implemented into any program (*see* Section 2.7).

(b) The preconditioner and the conditioned matrices become

$$\mathbf{D} = \begin{bmatrix} 1/\sqrt{25} & 0 & 0 & 0 & 0 \\ 0 & 1/\sqrt{25} & 0 & 0 & 0 \\ 0 & 0 & 1/\sqrt{23} & 0 & 0 \\ 0 & 0 & 0 & 1/\sqrt{22} & 0 \\ 0 & 0 & 0 & 0 & 1/\sqrt{10021} \end{bmatrix}$$

and

$$\mathbf{M} = \mathbf{DAD}^T = \begin{bmatrix} 1 & 0.92 & 0.875761 & 0.810163 & 0.339644 \\ 0.92 & 1 & 0.959166 & 0.938083 & 0.041956 \\ 0.875761 & 0.959166 & 1 & 0.889108 & 0.0395762 \\ 0.810163 & 0.938083 & 0.889108 & 1 & 0.127786 \\ 0.339644 & 0.041956 & 0.0395762 & 0.127786 & 1 \end{bmatrix}$$

which is also symmetrical.

(c) The eigenvalues of $\mathbf{M}$ are found as

$$\lambda = \{1238.76, 15.0536, 5.98462, 0.960842, 0.268462\}$$

from which we find $\kappa(\mathbf{M}) = 1238.76/0.268462 = 4614.3$, which is still a large value, but the condition number is reduced by a factor of about 100 compared to the original coefficient matrix $\mathbf{A}$.

Discussion: The essence of preconditioning boils down to finding the condition number of the original and preconditioned matrices (i.e., $\mathbf{A}$ and $\mathbf{M}$). Computing the spectral radius of a matrix can be challenging, especially for large or complex matrices. Speaking of *symmetric and positive definite matrices*, the condition number for relatively small matrices can be found directly by computing the eigenvalues using exact methods like the characteristic polynomial or *via* standard eigenvalue algorithms (*see* Chapter 11).

There are several strategies that one might want to consider to simplify the process of finding the eigenvalues. The QR decomposition or Jacobi diagonalization methods can give all of the eigenvalues and, hence, the condition number. For symmetric matrices, the computation is fast and generally does not require specialized software. On the other hand, the Power and inverse Power methods allow one to determine the largest and smallest magnitude eigenvalues very quickly.

In the computation of eigenvalues of large matrices, the numerical procedure is adversely affected by rounding errors or loss of precision. For instance, in iterative methods such as Power or inverse Power iteration, small perturbations or inappropriate initial guess vectors might lead to slower convergence. Furthermore, large matrices tend to be *ill-conditioned*, meaning that eigenvalue computations can be sensitive to numerical errors. Regularization or scaling might be necessary to improve stability in such cases as well.