Iterative Methods for Solving Systems of Linear Equations

Classic Iterative Methods

Basic Concept

Consider solving

$$\begin{bmatrix} 3 & 2 \\ 1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}.$$

This system has the exact solution $x_1 = x_2 = 1$.

Equivalently we can write the system as

$$\begin{cases} 3x_1 + 2x_2 = 5 \\ x_1 + 4x_2 = 5 \end{cases}$$

This implies that

$$\begin{cases} x_1 = \frac{1}{3}(5 - 2x_2) \\ x_2 = \frac{1}{4}(5 - x_1) \end{cases}$$

A naieve idea is to solve the system by

$$\begin{cases} x_1^{(k)} = \frac{1}{3}(5 - 2x_2^{(k-1)}) \\ x_2^{(k)} = \frac{1}{4}(5 - x_1^{(k-1)}) \end{cases}$$

that is, to use the iterative formulation

$$\begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \end{bmatrix} = \begin{bmatrix} \frac{1}{3} & 0 \\ 0 & \frac{1}{4} \end{bmatrix} \left(\begin{bmatrix} 5 \\ 5 \end{bmatrix} - \begin{bmatrix} 0 & 2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1^{(k-1)} \\ x_2^{(k-1)} \end{bmatrix} \right)$$

If we choose the initial guess $x_1^{(0)} = x_2^{(0)} = 0$, we would otain

$$\begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} = \begin{bmatrix} \frac{1}{3} & 0 \\ 0 & \frac{1}{4} \end{bmatrix} \left(\begin{bmatrix} 5 \\ 5 \end{bmatrix} - \begin{bmatrix} 0 & 2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} 1.6667 \\ 1.2500 \end{bmatrix}$$

$$\begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \end{bmatrix} = \begin{bmatrix} \frac{1}{3} & 0 \\ 0 & \frac{1}{4} \end{bmatrix} \left(\begin{bmatrix} 5 \\ 5 \end{bmatrix} - \begin{bmatrix} 0 & 2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1.6667 \\ 1.2500 \end{bmatrix} \right) = \begin{bmatrix} 0.8333 \\ 0.8333 \end{bmatrix}$$

By repeating the process, we have the following table

k	3	4	5	6	7
$x_1^{(k)}$	1.1111	0.9722	1.0185	0.9954	1.0031
$x_2^{(k)}$	1.0417	0.9722	1.0000	0.9954	1.0012

From this example, we observe that the basic idea is to split the coefficient matrix A into

$$A = M - (M - A),$$

for some matrix M, which is called the splitting matrix.

Here we assume that A and M are both nonsingular.

Then the original problem is rewritten in the equivalent form

$$Mx = (M - A)x + b.$$

This suggests an iterative process

$$x^{(k)} = (I - M^{-1}A)x^{(k-1)} + M^{-1}b \equiv Tx^{(k-1)} + c,$$

where T is usually called the iteration matrix. The initial vector $x^{(0)}$ can be arbitrary

Two criteria for choosing the splitting matrix M are

1. $x^{(k)}$ is easily computed. More precisely, the system $Mx^{(k)} = y$ is easy to solve;
2. the sequence $\{x^{(k)}\}$ converges rapidly to the exact solution.

Note that one way to achieve the second goal is to choose M so that M^{-1} approximate A^{-1}

Richard's Method

When we choose $M = I$ such that $A = I - (I - A)$, we obtain the iteration procedure

$$x^{(k)} = (I - A)x^{(k-1)} + b = x^{(k-1)} - Ax^{(k-1)} + b \equiv x^{(k-1)} + r^{(k-1)},$$

Algorithm **(Richard's Method)**
for $k = 1, 2, \ldots$ **do**
for $i = 1, 2, \ldots, n$ **do**

$$r_i^{(k-1)} = b_i - \sum_{j=1}^{n} a_{ij} x_j^{(k-1)}$$

$$x_i^{(k)} = x_i^{(k-1)} + r_i^{(k-1)}$$

end for
end for

Two n-vectors are required to implement this algorithm.

Jacobi Method

If we decompose the coefficient matrix A as

$$A = L + D + U,$$

where D is the diagonal part, L is the strictly lower triangular part, and U is the strictly upper triangular part, of A, and choose $M = D$, then we derive the iterative formulation for Jacobi method:

$$x^{(k)} = -D^{-1}(L + U)x^{(k-1)} + D^{-1}b.$$

With this method, the iteration matrix $T = -D^{-1}(L+U)$ and $c = D^{-1}b$. Each component $x_i^{(k)}$ can be computed by

$$x_i^{(k)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k-1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} \right) \Big/ a_{ii}.$$

Jacobi Method

$$x_i^{(k)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k-1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} \right) \Big/ a_{ii}.$$

Algorithm (Jacobi Method)

for $k = 1, 2, \ldots$ **do**

  for $i = 1, 2, \ldots, n$ **do**

$$x_i^{(k)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k-1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} \right) \Big/ a_{ii}$$

  end for

end for

Two n-vectors are required to implement this algorithm. A big advantage of Jacobi method is that $x_i^{(k)}, i = 1, \ldots, n$, can be computed in parallel at each iteration k.

Another important observation is that in Jacobi method, only the components of $x^{(k-1)}$ are used to compute $x^{(k)}$. When computing $x_i^{(k)}$ for $i > 1$, $x_1^{(k)}, \ldots, x_{i-1}^{(k)}$ have already been computed and are likely to be better approximations to the exact $x_1, \ldots, x_{i-1}$ than $x_1^{(k-1)}, \ldots, x_{i-1}^{(k-1)}$. It seems reasonable to compute $x_i^{(k)}$ using these most recently computed values. This improvement induce the Gauss-Seidel method which is to be discussed in the next subsection.

Gauss-Seidel Method

The Gauss-Seidel method sets $M = D + L$ and defines the iteration as

$$x^{(k)} = -(D + L)^{-1}Ux^{(k-1)} + (D + L)^{-1}b.$$

That is, Gauss-Seidel method uses $T = -(D + L)^{-1}U$ as the iteration matrix. The formulation above can be rewritten as

$$x^{(k)} = -D^{-1}\left(Lx^{(k)} + Ux^{(k-1)} - b\right).$$

Hence each component $x_i^{(k)}$ can be computed by

$$x_i^{(k)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)}\right) \Big/ a_{ii}.$$

Gauss-Seidel Method

$$x_i^{(k)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)}\right) \Big/ a_{ii}.$$

Algorithm **(Gauss-Seidel Method)**
for $k = 1, 2, \ldots$ **do**
for $i = 1, 2, \ldots, n$ **do**

$$x_i^{(k)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)}\right) \Big/ a_{ii}$$

end for
end for

At each iteration, since $x_i^{(k)}$ can not be computed until $x_1^{(k)}, \ldots, x_{i-1}^{(k)}$ are available, Gauss-Seidel method is not a parallel algorithm in nature. Moreover, only one n-vector is required for implementation in theory, but two are usually used in practice.

Successive Over Relaxation (SOR) Method

The successive over relaxation (SOR) method choose $M = \omega^{-1}(D+\omega L)$, where $0 < \omega < 2$ is called the relaxation parameter, and defines the iteration

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

Hence the iteration matrix $T = (D + \omega L)^{-1}((1-\omega)D - \omega U)$. Each component $x_i^{(k)}$ can be computed by the formulation

$$x_i^{(k)} = \omega \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} \right) \Big/ a_{ii} + (1-\omega) x_i^{(k-1)}.$$

The question of choosing a good relaxation parameter ω is a very complex topic.

Symmetric Successive Over Relaxation (SSOR) Method

In theory the symmetric successive over relaxation (SSOR) method chooses the splitting matrix $M = \frac{1}{\omega(2-\omega)}(D + \omega L)D^{-1}(D + \omega U)$ and iterates with the iteration matrix

$$T = (D + \omega U)^{-1} \left((1-\omega)D - \omega L\right) (D + \omega L)^{-1} \left((1-\omega)D - \omega U\right).$$

The idea is in fact to implement the SOR formulation twice, one forward and one backward, at each iteration. That is, SSOR method defines

$$\begin{aligned} (D + \omega L)x^{(k-\frac{1}{2})} &= \left((1-\omega)D - \omega U\right) x^{(k-1)} + \omega b \\ (D + \omega U)x^{(k)} &= \left((1-\omega)D - \omega L\right) x^{(k-\frac{1}{2})} + \omega b \end{aligned}$$

Symmetric Successive Over Relaxation (SSOR) Method

Each component $x_i^{(k)}$ is obtained by first computing

$$x_i^{(k-\frac{1}{2})} = \omega \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k-\frac{1}{2})} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} \right) \Big/ a_{ii} + (1-\omega) x_i^{(k)}$$

followed by

$$x_i^{(k)} = \omega \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-\frac{1}{2})} \right) \Big/ a_{ii} + (1-\omega) x_i^{(k-\frac{1}{2})}.$$

Jacobi Method

$$\begin{bmatrix} 25 & 5 & 1 \\ 64 & 8 & 1 \\ 144 & 12 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 106.8 \\ 177.2 \\ 279.2 \end{bmatrix}$$

$$a_1 = \frac{106.8 - 5a_2 - a_3}{25}$$

$$a_2 = \frac{177.2 - 64a_1 - a_3}{8}$$

$$a_3 = \frac{279.2 - 144a_1 - 12a_2}{1}$$

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 5 \end{bmatrix}$$

$$a_1 = \frac{106.8 - 5(2) - (5)}{25}$$
$$= 3.6720$$

$$a_2 = \frac{177.2 - 64\,(3.6720) - (5)}{8}$$
$$= -7.8150$$

$$a_3 = \frac{279.2 - 144\,(3.6720) - 12\,(-7.8510)}{1}$$
$$= -155.36$$

$$|\epsilon_a|_1 = \left|\frac{3.6720 - 1}{3.6720}\right| \times 100$$
$$= 72.76\%$$

$$|\epsilon_a|_2 = \left|\frac{-7.8510 - 2}{-7.8510}\right| \times 100$$
$$= 125.47\%$$

$$|\epsilon_a|_3 = \left|\frac{-155.36 - 5}{-155.36}\right| \times 100$$
$$= 103.22\%$$

At the end of the first iteration, the estimate of the solution vector is

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 3.6720 \\ -7.8510 \\ -155.36 \end{bmatrix}$$

and the maximum absolute relative approximate error is 125.47.

Iteration #2

The estimate of the solution vector at the end of Iteration #1 is

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 3.6720 \\ -7.8510 \\ -155.36 \end{bmatrix}$$

Now we get

$$\begin{aligned} a_1 &= \frac{106.8 - 5\,(-7.8510) - (-155.36)}{25} \\ &= 12.056 \\ a_2 &= \frac{177.2 - 64\,(12.056) - (-155.36)}{8} \\ &= -54.882 \\ a_3 &= \frac{279.2 - 144\,(12.056) - 12\,(-54.882)}{1} \\ &= -798.34 \end{aligned}$$

The absolute relative approximate error for each x_i then is

$$\begin{aligned} |\in_a|_1 &= \left|\frac{12.056 - 3.6720}{12.056}\right| \times 100 \\ &= 69.543\% \\ |\in_a|_2 &= \left|\frac{-54.882 - (-7.8510)}{-54.882}\right| \times 100 \\ &= 85.695\% \\ |\in_a|_3 &= \left|\frac{-798.34 - (-155.36)}{-798.34}\right| \times 100 \\ &= 80.540\% \end{aligned}$$

At the end of the second iteration the estimate of the solution vector is

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 12.056 \\ -54.882 \\ -798.54 \end{bmatrix}$$

and the maximum absolute relative approximate error is 85.695%.

and the maximum absolute relative approximate error is 85.695%.

Conducting more iterations gives the following values for the solution vector and the corresponding absolute relative approximate errors.

Iteration	a_1	$\lvert\epsilon_a\rvert_1\%$	a_2	$\lvert\epsilon_a\rvert_2\%$	a_3	$\lvert\epsilon_a\rvert_3\%$
1	3.672	72.767	−7.8510	125.47	−155.36	103.22
2	12.056	69.543	−54.882	85.695	−798.34	80.54
3	47.182	74.447	−255.51	78.521	−3448.9	76.852
4	193.33	75.595	−1093.4	76.632	−14440	76.116
5	800.53	75.85	−4577.2	76.112	−60072	75.963
6	3322.6	75.906	−19049	75.972	−249580	75.931

As seen in the above table, the solution estimates are not converging to the true solution of

$$a_1 = 0.29048$$

$$a_2 = 19.690$$

$$a_3 = 1.0857$$

$$Mx = (M - A)x + b.$$

$$x^{(k)} = (I - M^{-1}A)x^{(k-1)} + M^{-1}b \equiv Tx^{(k-1)} + c,$$

Richard's Method $M = I \qquad A = I - (I - A)$

$$x^{(k)} = (I - A)x^{(k-1)} + b = x^{(k-1)} - Ax^{(k-1)} + b \equiv x^{(k-1)} + r^{(k-1)},$$

Jacobi Method $M = D, \quad A = L + D + U,$

$$x^{(k)} = -D^{-1}(L + U)x^{(k-1)} + D^{-1}b.$$