In numerical linear algebra, the Jacobi method (or Jacobi iterative method[1]) is an algorithm for determining the solutions of a diagonally dominant system of linear equations. Each diagonal element is solved for, and an approximate value is plugged in. The process is then iterated until it converges. This algorithm is a stripped-down version of the Jacobi transformation method of matrix diagonalization. The method is named after Carl Gustav Jacob Jacobi.

### Description

Let

\[ A \mathbf{x} = \mathbf{b} \]

be a square system of \( n \) linear equations, where:

\[
A = \begin{bmatrix}
ad_{11} & a_{12} & \cdots & a_{1n} \\
a_{21} & a_{22} & \cdots & a_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
a_{n1} & a_{n2} & \cdots & a_{nn}
\end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix}
x_1 \\
x_2 \\
\vdots \\
x_n
\end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix}
b_1 \\
b_2 \\
\vdots \\
b_n
\end{bmatrix}.
\]

Then \( A \) can be decomposed into a diagonal component \( D \), and the remainder \( R \):

\[
A = D + R \quad \text{where} \quad D = \begin{bmatrix}
ad_{11} & 0 & \cdots & 0 \\
0 & a_{22} & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & a_{nn}
\end{bmatrix} \quad \text{and} \quad R = \begin{bmatrix}
0 & a_{12} & \cdots & a_{1n} \\
a_{21} & 0 & \cdots & a_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
a_{n1} & a_{n2} & \cdots & 0
\end{bmatrix}.
\]

The solution is then obtained iteratively via

\[
\mathbf{x}^{(k+1)} = D^{-1} (\mathbf{b} - R \mathbf{x}^{(k)}),
\]
where $\mathbf{x}^{(k)}$ is the $k$th approximation or iteration of $\mathbf{x}$ and $\mathbf{x}^{(k+1)}$ is the next or $k+1$ iteration of $\mathbf{x}$. The element-based formula is thus:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left( b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \ldots, n.$$ 

The computation of $x_i^{(k+1)}$ requires each element in $\mathbf{x}^{(k)}$ except itself. Unlike the Gauss–Seidel method, we can’t overwrite $x_i^{(k)}$ with $x_i^{(k+1)}$, as that value will be needed by the rest of the computation. The minimum amount of storage is two vectors of size $n$.

**Algorithm**

```plaintext
Input: initial guess $\mathbf{z}^{(0)}$ to the solution, (diagonal dominant) matrix $\mathbf{A}$, right-hand side vector $\mathbf{b}$, convergence criterion
Output: solution when convergence is reached
Comments: pseudocode based on the element-based formula above

$k = 0$

while convergence not reached do
  for $i := 1$ step until $n$ do
    for $j := 1$ step until $n$ do
      if $j \neq i$ then
        $\sigma = \sigma + a_{ij} z_j^{(k)}$
      end
    end
    $x_i^{(k+1)} = \frac{1}{a_{ii}} (b_i - \sigma)$
  end
  $k = k + 1$
end
```

**Convergence**

The standard convergence condition (for any iterative method) is when the spectral radius of the iteration matrix is less than 1:

$$\rho(D^{-1}R) < 1.$$ 

A sufficient (but not necessary) condition for the method to converge is that the matrix $\mathbf{A}$ is strictly or irreducibly diagonally dominant. Strict row diagonal dominance means that for each row, the absolute value of the diagonal term is greater than the sum of absolute values of other terms:

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|.$$ 

The Jacobi method sometimes converges even if these conditions are not satisfied.

**Example**

A linear system of the form $\mathbf{Ax} = \mathbf{b}$ with initial estimate $\mathbf{z}^{(0)}$ is given by

$$\mathbf{A} = \begin{bmatrix} 2 & 1 \\ 5 & 7 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 11 \\ 13 \end{bmatrix} \quad \text{and} \quad \mathbf{z}^{(0)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$ 

We use the equation $\mathbf{x}^{(k+1)} = D^{-1} (\mathbf{b} - \mathbf{Rx}^{(k)})$, described above, to estimate $\mathbf{x}$. First, we rewrite the equation in a more convenient form $D^{-1} (\mathbf{b} - \mathbf{Rx}^{(k)}) = T \mathbf{x}^{(k)} + \mathbf{C}$, where $T = -D^{-1} \mathbf{R}$ and $\mathbf{C} = D^{-1} \mathbf{b}$. Note that $\mathbf{R} = \mathbf{L} + \mathbf{U}$ where $\mathbf{L}$ and $\mathbf{U}$ are the strictly lower and upper parts of $\mathbf{A}$. From the known values
\[ D^{-1} = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/7 \end{bmatrix}, \quad L = \begin{bmatrix} 0 & 0 \\ 5 & 0 \end{bmatrix} \quad \text{and} \quad U = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}. \]

we determine \( T = -D^{-1}(L + U) \) as

\[ T = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/7 \end{bmatrix} \left\{ \begin{bmatrix} 0 & 0 \\ -5 & 0 \end{bmatrix} + \begin{bmatrix} 0 & -1 \\ 0 & 0 \end{bmatrix} \right\} = \begin{bmatrix} 0 & -1/2 \\ -5/7 & 0 \end{bmatrix}. \]

Further, \( C \) is found as

\[ C = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/7 \end{bmatrix} \begin{bmatrix} 11 \\ 13 \end{bmatrix} = \begin{bmatrix} 11/2 \\ 13/7 \end{bmatrix}. \]

With \( T \) and \( C \) calculated, we estimate \( x^{(1)} \) as \( x^{(1)} = Tx^{(0)} + C \):

\[ x^{(1)} = \begin{bmatrix} 0 & -1/2 \\ -5/7 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 11/2 \end{bmatrix} = \begin{bmatrix} 5.0 \\ 8/7 \end{bmatrix} \approx \begin{bmatrix} 5 \\ 1.143 \end{bmatrix}. \]

The next iteration yields

\[ x^{(2)} = \begin{bmatrix} 0 & -1/2 \\ -5/7 & 0 \end{bmatrix} \begin{bmatrix} 5.0 \\ 8/7 \end{bmatrix} + \begin{bmatrix} 11/2 \\ 13/7 \end{bmatrix} = \begin{bmatrix} 69/14 \\ -12/7 \end{bmatrix} \approx \begin{bmatrix} 4.929 \\ -1.714 \end{bmatrix}. \]

This process is repeated until convergence (i.e., until \( \|Ax^{(n)} - b\| \) is small). The solution after 25 iterations is

\[ x = \begin{bmatrix} 7.111 \\ -3.222 \end{bmatrix}. \]

**Another example**

Suppose we are given the following linear system:

\[
\begin{align*}
10x_1 - x_2 + 2x_3 &= 6, \\
-x_1 + 11x_2 - x_3 + 3x_4 &= 25, \\
2x_1 - x_2 + 10x_3 - x_4 &= -11, \\
3x_2 - x_3 + 8x_4 &= 15.
\end{align*}
\]

If we choose \((0, 0, 0, 0)\) as the initial approximation, then the first approximate solution is given by

\[
\begin{align*}
x_1 &= (6 + 0 - (2 \ast 0))/10 = 0.6, \\
x_2 &= (25 + 0 - (3 \ast 0))/11 = 25/11 = 2.2727, \\
x_3 &= (-11 - (2 \ast 0) + 0 + 0)/10 = -1.1, \\
x_4 &= (15 - (3 \ast 0) + 0)/8 = 1.875.
\end{align*}
\]

Using the approximations obtained, the iterative procedure is repeated until the desired accuracy has been reached. The following are the approximated solutions after five iterations.
The exact solution of the system is (1, 2, -1, 1).

An example using Python and Numpy

The following numerical procedure simply iterates to produce the solution vector

```python
import numpy as np

ITERATION_LIMIT = 1000

# initialize the matrix
A = np.array([[10., -1., 2., 0.],
              [-1., 11., -1., 3.],
              [2., -1., 10., -1.],
              [0.0, 3., -1., 8.]])

# initialize the RHS vector
b = np.array([6., 25., -11., 15.])

# prints the system
print("System:")
for i in range(A.shape[0]):
    row = "{}*x{}".format(A[i, j], j+1) for j in range(A.shape[1])
    print(" + " + " " + row, "=" + b[i])

print()

x = np.zeros_like(b)
for it_count in range(ITERATION_LIMIT):
    print("Current solution:", x)
    x_new = np.zeros_like(x)
    for i in range(A.shape[0]):
        s1 = np.dot(A[i, :i], x[:i])
        s2 = np.dot(A[i, i+1:], x[i+1:])[i]
        x_new[i] = (b[i] - s1 - s2) / A[i, i]
    if np.allclose(x, x_new, atol=1e-10, rtol=0.):
        break
    x = x_new
print("Solution:")
print(x)
error = np.dot(A, x) - b
print("Error:")
print(error)
```

Produces the output:

```
System:
10.0*x1 + -1.0*x2 + 2.0*x3 + 0.0*x4 = 6.0
-1.0*x1 + 11.0*x2 + -1.0*x3 + 3.0*x4 = 25.0
2.0*x1 + -1.0*x2 + 10.0*x3 + -1.0*x4 = -11.0
0.0*x1 + 3.0*x2 + -1.0*x3 + 8.0*x4 = 15.0

Current solution: [ 0.  0.  0.  0.]
Current solution: [ 0.6  2.27272727 -1.1  1.875]
Current solution: [ 1.04727273 1.71590909 -0.80522727  0.88522727]
Current solution: [ 0.93263636  1.71590909 -0.80522727  0.88522727]
Current solution: [ 1.01519876  1.71590909 -0.80522727  0.88522727]
Current solution: [ 1.01519876  1.71590909 -0.80522727  0.88522727]
Current solution: [ 0.99812847  2.01141476 -1.0102859   1.02135051]
Current solution: [ 1.0001186   1.9986703  -0.99982814  0.99978598]
```

The output shows the iterations of the solution process, starting from an initial guess and converging to the exact solution.
**Weighted Jacobi method**

The weighted Jacobi iteration uses a parameter \( \omega \) to compute the iteration as

\[
x^{(k+1)} = \omega D^{-1} (b - Rx^{(k)}) + (1 - \omega) x^{(k)}
\]

with \( \omega = 2/3 \) being the usual choice\(^2\).

**Recent developments**

In 2014, a refinement of the algorithm, called *scheduled relaxation Jacobi (SRJ) method*, was published\(^1\)[3]. The new method employs a schedule of over- and under-relaxations and provides performance improvements for solving elliptic equations discretized on large two- and three-dimensional Cartesian grids. The described algorithm applies the well-known technique of polynomial (Chebyshev) acceleration to a problem with a known spectrum distribution that can be classified either as a multi-step method or a one-step method with a non-diagonal preconditioner. However, none of them are Jacobi-like methods.

Improvements published\(^4\) in 2015.

**See also**

- Gauss–Seidel method
- Successive over-relaxation
- Iterative method § Linear systems
- Gaussian Belief Propagation
- Matrix splitting

**References**


**External links**

This article incorporates text from the article Jacobi method on CFD-Wiki that is under the GFDL license.

Black, Noel; Moore, Shirley; and Weisstein, Eric W. "Jacobi method". MathWorld.

Jacobi Method from www.math-linux.com

Numerical matrix inversion


This page was last edited on 12 November 2017, at 18:34.

Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.