WIKIPEDIA
The Free Encyclopedia

Gauss–Seidel method

In numerical linear algebra, the **Gauss–Seidel method**, also known as the **Liebmann method** or the **method of successive displacement**, is an iterative method used to solve a system of linear equations. It is named after the German mathematicians Carl Friedrich Gauss and Philipp Ludwig von Seidel. Though it can be applied to any matrix with non-zero elements on the diagonals, convergence is only guaranteed if the matrix is either strictly diagonally dominant,[1] or symmetric and positive definite. It was only mentioned in a private letter from Gauss to his student Gerling in 1823.[2] A publication was not delivered before 1874 by Seidel.[3]

Description

Let $\mathbf{Ax} = \mathbf{b}$ be a square system of n linear equations, where:

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

When $\mathbf{A}$ and $\mathbf{b}$ are known, and $\mathbf{x}$ is unknown, the Gauss–Seidel method can be used to iteratively approximate $\mathbf{x}$. The vector $\mathbf{x}^{(0)}$ denotes the initial guess for $\mathbf{x}$, often $\mathbf{x}_i^{(0)} = 0$ for $i = 1, 2, \ldots, n$. Denote by $\mathbf{x}^{(k)}$ the k-th approximation or iteration of $\mathbf{x}$, and by $\mathbf{x}^{(k+1)}$ the approximation of $\mathbf{x}$ at the next (or $k+1$-th) iteration.

Matrix-based formula

The solution is obtained iteratively via

$$\mathbf{L}\mathbf{x}^{(k+1)} = \mathbf{b} - \mathbf{U}\mathbf{x}^{(k)},$$

where the matrix $\mathbf{A}$ is decomposed into a lower triangular component $\mathbf{L}$, and a strictly upper triangular component $\mathbf{U}$ such that $\mathbf{A} = \mathbf{L} + \mathbf{U}$.[4] More specifically, the decomposition of A into L_* and U is given by:

$$\mathbf{A} = \underbrace{\begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}}_{\mathbf{L}} + \underbrace{\begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}}_{\mathbf{U}}.$$

Why the matrix-based formula works

The system of linear equations may be rewritten as:

$$\begin{aligned}\mathbf{A}\mathbf{x} &= \mathbf{b} \\ (\mathbf{L}+\mathbf{U})\mathbf{x} &= \mathbf{b} \\ \mathbf{L}\mathbf{x}+\mathbf{U}\mathbf{x} &= \mathbf{b} \\ \mathbf{L}\mathbf{x} &= \mathbf{b}-\mathbf{U}\mathbf{x}\end{aligned}$$

The Gauss–Seidel method now solves the left hand side of this expression for $\mathbf{x}$, using the previous value for $\mathbf{x}$ on the right hand side. Analytically, this may be written as

$$\mathbf{x}^{(k+1)} = \mathbf{L}^{-1}\left(\mathbf{b}-\mathbf{U}\mathbf{x}^{(k)}\right).$$

Element-based formula

However, by taking advantage of the triangular form of $\mathbf{L}$, the elements of $\mathbf{x}^{(k+1)}$ can be computed sequentially for each row i using forward substitution:[5]

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

Notice that the formula uses two summations per iteration which can be expressed as one summation $\sum_{j\neq i} a_{ij}x_j$ that uses the most recently calculated iteration of x_j. The procedure is generally continued until the changes made by an iteration are below some tolerance, such as a sufficiently small residual.

Discussion

The element-wise formula for the Gauss–Seidel method is related to that of the (iterative) Jacobi method, with an important difference:

In Gauss-Seidel, the computation of $\mathbf{x}^{(k+1)}$ uses the elements of $\mathbf{x}^{(k+1)}$ that have already been computed, and only the elements of $\mathbf{x}^{(k)}$ that have not been computed in the $(k+1)$-th iteration. This means that, unlike the Jacobi method, only one storage vector is required as elements can be overwritten as they are computed, which can be advantageous for very large problems.

However, unlike the Jacobi method, the computations for each element are generally much harder to implement in parallel, since they can have a very long critical path, and are thus most feasible for sparse matrices. Furthermore, the values at each iteration are dependent on the order of the original equations.

Gauss-Seidel is the same as successive over-relaxation with $\omega = 1$.

Convergence

The convergence properties of the Gauss–Seidel method are dependent on the matrix $\mathbf{A}$. Namely, the procedure is known to converge if either:

- $\mathbf{A}$ is symmetric positive-definite,[6] or
- $\mathbf{A}$ is strictly or irreducibly diagonally dominant.[7]

The Gauss–Seidel method may converge even if these conditions are not satisfied.

Golub and Van Loan give a theorem for an algorithm that splits $\mathbf{A}$ into two parts. Suppose $\mathbf{A} = \mathbf{M} - \mathbf{N}$ is nonsingular. Let $r = \rho(\mathbf{M}^{-1}\mathbf{N})$ be the spectral radius of $\mathbf{M}^{-1}\mathbf{N}$. Then the iterates $\mathbf{x}^{(k)}$ defined by $\mathbf{M}\mathbf{x}^{(k+1)} = \mathbf{N}\mathbf{x}^{(k)} + \mathbf{b}$ converge to $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ for any starting vector $\mathbf{x}^{(0)}$ if $\mathbf{M}$ is nonsingular and $r < 1$.[8]

Algorithm

Since elements can be overwritten as they are computed in this algorithm, only one storage vector is needed, and vector indexing is omitted. The algorithm goes as follows:

```
algorithm Gauss-Seidel method is
    inputs: A, b
    output: φ

    Choose an initial guess φ to the solution
    repeat until convergence
        for i from 1 until n do
            σ ← 0
            for j from 1 until n do
                if j ≠ i then
                    σ ← σ + a_ij φ_j
                end if
            end (j-loop)
            φ_i ← (b_i - σ) / a_ii
        end (i-loop)
        check if convergence is reached
    end (repeat)
```

Examples

An example for the matrix version

A linear system shown as $\mathbf{A}\mathbf{x} = \mathbf{b}$ is given by:

$$\mathbf{A} = \begin{bmatrix} 16 & 3 \\ 7 & -11 \end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} 11 \\ 13 \end{bmatrix}.$$

Use the equation

$$\mathbf{x}^{(k+1)} = \mathbf{L}^{-1}\left(\mathbf{b} - \mathbf{U}\mathbf{x}^{(k)}\right)$$

in the form

$$\mathbf{x}^{(k+1)} = \mathbf{T}\mathbf{x}^{(k)} + \mathbf{c}$$

where:

$$\mathbf{T} = -\mathbf{L}^{-1}\mathbf{U} \quad \text{and} \quad \mathbf{c} = \mathbf{L}^{-1}\mathbf{b}.$$

Decompose $\mathbf{A}$ into the sum of a lower triangular component $\mathbf{L}$ and a strict upper triangular component U:

$$\mathbf{L} = \begin{bmatrix} 16 & 0 \\ 7 & -11 \end{bmatrix} \quad \text{and} \quad \mathbf{U} = \begin{bmatrix} 0 & 3 \\ 0 & 0 \end{bmatrix}.$$

The inverse of $\mathbf{L}$ is:

$$\mathbf{L}^{-1} = \begin{bmatrix} 16 & 0 \\ 7 & -11 \end{bmatrix}^{-1} = \begin{bmatrix} 0.0625 & 0.0000 \\ 0.0398 & -0.0909 \end{bmatrix}.$$

Now find:

$$\mathbf{T} = -\begin{bmatrix} 0.0625 & 0.0000 \\ 0.0398 & -0.0909 \end{bmatrix} \begin{bmatrix} 0 & 3 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0.000 & -0.1875 \\ 0.000 & -0.1194 \end{bmatrix},$$

$$\mathbf{c} = \begin{bmatrix} 0.0625 & 0.0000 \\ 0.0398 & -0.0909 \end{bmatrix} \begin{bmatrix} 11 \\ 13 \end{bmatrix} = \begin{bmatrix} 0.6875 \\ -0.7439 \end{bmatrix}.$$

With $\mathbf{T}$ and $\mathbf{c}$ the vectors $\mathbf{x}$ can be obtained iteratively.

First of all, choose $\mathbf{x}^{(0)}$, for example

$$\mathbf{x}^{(0)} = \begin{bmatrix} 1.0 \\ 1.0 \end{bmatrix}.$$

The closer the guess to the final solution, the fewer iterations the algorithm will need.

Then calculate:

$$\mathbf{x}^{(1)} = \begin{bmatrix} 0.000 & -0.1875 \\ 0.000 & -0.1193 \end{bmatrix} \begin{bmatrix} 1.0 \\ 1.0 \end{bmatrix} + \begin{bmatrix} 0.6875 \\ -0.7443 \end{bmatrix} = \begin{bmatrix} 0.5000 \\ -0.8636 \end{bmatrix}.$$

$$\mathbf{x}^{(2)} = \begin{bmatrix} 0.000 & -0.1875 \\ 0.000 & -0.1193 \end{bmatrix} \begin{bmatrix} 0.5000 \\ -0.8636 \end{bmatrix} + \begin{bmatrix} 0.6875 \\ -0.7443 \end{bmatrix} = \begin{bmatrix} 0.8494 \\ -0.6413 \end{bmatrix}.$$

$$\mathbf{x}^{(3)} = \begin{bmatrix} 0.000 & -0.1875 \\ 0.000 & -0.1193 \end{bmatrix} \begin{bmatrix} 0.8494 \\ -0.6413 \end{bmatrix} + \begin{bmatrix} 0.6875 \\ -0.7443 \end{bmatrix} = \begin{bmatrix} 0.8077 \\ -0.6678 \end{bmatrix}.$$

$$\mathbf{x}^{(4)} = \begin{bmatrix} 0.000 & -0.1875 \\ 0.000 & -0.1193 \end{bmatrix} \begin{bmatrix} 0.8077 \\ -0.6678 \end{bmatrix} + \begin{bmatrix} 0.6875 \\ -0.7443 \end{bmatrix} = \begin{bmatrix} 0.8127 \\ -0.6646 \end{bmatrix}.$$

$$\mathbf{x}^{(5)} = \begin{bmatrix} 0.000 & -0.1875 \\ 0.000 & -0.1193 \end{bmatrix} \begin{bmatrix} 0.8127 \\ -0.6646 \end{bmatrix} + \begin{bmatrix} 0.6875 \\ -0.7443 \end{bmatrix} = \begin{bmatrix} 0.8121 \\ -0.6650 \end{bmatrix}.$$

$$\mathbf{x}^{(6)} = \begin{bmatrix} 0.000 & -0.1875 \\ 0.000 & -0.1193 \end{bmatrix} \begin{bmatrix} 0.8121 \\ -0.6650 \end{bmatrix} + \begin{bmatrix} 0.6875 \\ -0.7443 \end{bmatrix} = \begin{bmatrix} 0.8122 \\ -0.6650 \end{bmatrix}.$$

$$\mathbf{x}^{(7)} = \begin{bmatrix} 0.000 & -0.1875 \\ 0.000 & -0.1193 \end{bmatrix} \begin{bmatrix} 0.8122 \\ -0.6650 \end{bmatrix} + \begin{bmatrix} 0.6875 \\ -0.7443 \end{bmatrix} = \begin{bmatrix} 0.8122 \\ -0.6650 \end{bmatrix}.$$

As expected, the algorithm converges to the solution:

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} \approx \begin{bmatrix} 0.8122 \\ -0.6650 \end{bmatrix}$$

.

In fact, the matrix A is strictly diagonally dominant, but not positive definite.

Another example for the matrix version

Another linear system shown as $\mathbf{Ax} = \mathbf{b}$ is given by:

$$\mathbf{A} = \begin{bmatrix} 2 & 3 \\ 5 & 7 \end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} 11 \\ 13 \end{bmatrix}.$$

Use the equation

$$\mathbf{x}^{(k+1)} = \mathbf{L}^{-1}(\mathbf{b} - \mathbf{U}\mathbf{x}^{(k)})$$

in the form

$$\mathbf{x}^{(k+1)} = \mathbf{T}\mathbf{x}^{(k)} + \mathbf{c}$$

where:

$$\mathbf{T} = -\mathbf{L}^{-1}\mathbf{U} \quad \text{and} \quad \mathbf{c} = \mathbf{L}^{-1}\mathbf{b}.$$

Decompose $\mathbf{A}$ into the sum of a lower triangular component $\mathbf{L}$ and a strict upper triangular component $\mathbf{U}$:

$$\mathbf{L} = \begin{bmatrix} 2 & 0 \\ 5 & 7 \end{bmatrix} \quad \text{and} \quad \mathbf{U} = \begin{bmatrix} 0 & 3 \\ 0 & 0 \end{bmatrix}.$$

The inverse of $\mathbf{L}$ is:

$$\mathbf{L}^{-1} = \begin{bmatrix} 2 & 0 \\ 5 & 7 \end{bmatrix}^{-1} = \begin{bmatrix} 0.500 & 0.000 \\ -0.357 & 0.143 \end{bmatrix}.$$

Now find:

$$\mathbf{T} = -\begin{bmatrix} 0.500 & 0.000 \\ -0.357 & 0.143 \end{bmatrix}\begin{bmatrix} 0 & 3 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0.000 & -1.500 \\ 0.000 & 1.071 \end{bmatrix},$$

$$\mathbf{c} = \begin{bmatrix} 0.500 & 0.000 \\ -0.357 & 0.143 \end{bmatrix}\begin{bmatrix} 11 \\ 13 \end{bmatrix} = \begin{bmatrix} 5.500 \\ -2.071 \end{bmatrix}.$$

With $\mathbf{T}$ and $\mathbf{c}$ the vectors $\mathbf{x}$ can be obtained iteratively.

First of all, we have to choose $\mathbf{x}^{(0)}$, for example

$$\mathbf{x}^{(0)} = \begin{bmatrix} 1.1 \\ 2.3 \end{bmatrix}$$

Then calculate:

$$\mathbf{x}^{(1)} = \begin{bmatrix} 0 & -1.500 \\ 0 & 1.071 \end{bmatrix} \begin{bmatrix} 1.1 \\ 2.3 \end{bmatrix} + \begin{bmatrix} 5.500 \\ -2.071 \end{bmatrix} = \begin{bmatrix} 2.050 \\ 0.393 \end{bmatrix}.$$

$$\mathbf{x}^{(2)} = \begin{bmatrix} 0 & -1.500 \\ 0 & 1.071 \end{bmatrix} \begin{bmatrix} 2.050 \\ 0.393 \end{bmatrix} + \begin{bmatrix} 5.500 \\ -2.071 \end{bmatrix} = \begin{bmatrix} 4.911 \\ -1.651 \end{bmatrix}.$$

$$\mathbf{x}^{(3)} = \cdots.$$

In a test for convergence we find that the algorithm diverges. In fact, the matrix $\mathbf{A}$ is neither diagonally dominant nor positive definite. Then, convergence to the exact solution

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} = \begin{bmatrix} -38 \\ 29 \end{bmatrix}$$

is not guaranteed and, in this case, will not occur.

An example for the equation version

Suppose given n equations and a starting point $\mathbf{x}_0$. At any step in a Gauss-Seidel iteration, solve the first equation for x_1 in terms of $x_2, \ldots, x_n$; then solve the second equation for x_2 in terms of x_1 just found and the remaining $x_3, \ldots, x_n$; and continue to x_n. Then, repeat iterations until convergence is achieved, or break if the divergence in the solutions start to diverge beyond a predefined level.

Consider an example:

$$\begin{aligned} 10x_1 - x_2 + 2x_3 &= 6, \\ -x_1 + 11x_2 - x_3 + 3x_4 &= 25, \\ 2x_1 - x_2 + 10x_3 - x_4 &= -11, \\ 3x_2 - x_3 + 8x_4 &= 15. \end{aligned}$$

Solving for x_1, x_2, x_3 and x_4 gives:

$$\begin{aligned} x_1 &= x_2/10 - x_3/5 + 3/5, \\ x_2 &= x_1/11 + x_3/11 - 3x_4/11 + 25/11, \\ x_3 &= -x_1/5 + x_2/10 + x_4/10 - 11/10, \\ x_4 &= -3x_2/8 + x_3/8 + 15/8. \end{aligned}$$

Suppose $(0, 0, 0, 0)$ is the initial approximation, then the first approximate solution is given by:

$$\begin{aligned} x_1 &= 3/5 = 0.6, \\ x_2 &= (3/5)/11 + 25/11 = 3/55 + 25/11 = 2.3272, \\ x_3 &= -(3/5)/5 + (2.3272)/10 - 11/10 = -3/25 + 0.23272 - 1.1 = -0.9873, \\ x_4 &= -3(2.3272)/8 + (-0.9873)/8 + 15/8 = 0.8789. \end{aligned}$$

Using the approximations obtained, the iterative procedure is repeated until the desired accuracy has been reached. The following are the approximated solutions after four iterations.

x_1	x_2	x_3	x_4
0.6	2.32727	−0.987273	0.878864
1.03018	2.03694	−1.01446	0.984341
1.00659	2.00356	−1.00253	0.998351
1.00086	2.0003	−1.00031	0.99985

The exact solution of the system is $(1, 2, -1, 1)$.

An example using Python and NumPy

The following iterative procedure produces the solution vector of a linear system of equations:

```
import numpy as np

ITERATION_LIMIT = 1000

# initialize the matrix
A = np.array(
    [
        [10.0, -1.0, 2.0, 0.0],
        [-1.0, 11.0, -1.0, 3.0],
        [2.0, -1.0, 10.0, -1.0],
        [0.0, 3.0, -1.0, 8.0],
    ]
)
# initialize the RHS vector
b = np.array([6.0, 25.0, -11.0, 15.0])

print("System of equations:")
for i in range(A.shape[0]):
    row = [f"{A[i,j]:3g}*x{j+1}" for j in range(A.shape[1])]
    print("[{0}] = [{1:3g}]".format(" + ".join(row), b[i]))

x = np.zeros_like(b, np.float_)
for it_count in range(1, ITERATION_LIMIT):
    x_new = np.zeros_like(x, dtype=np.float_)
    print(f"Iteration {it_count}: {x}")
    for i in range(A.shape[0]):
        s1 = np.dot(A[i, :i], x_new[:i])
        s2 = np.dot(A[i, i + 1 :], x[i + 1 :])
        x_new[i] = (b[i] - s1 - s2) / A[i, i]
    if np.allclose(x, x_new, rtol=1e-8):
        break
    x = x_new

print(f"Solution: {x}")
error = np.dot(A, x) - b
print(f"Error: {error}")
```

Produces the output:

```
System of equations:
[ 10*x1 +  -1*x2 +   2*x3 +   0*x4] = [  6]
[ -1*x1 +  11*x2 +  -1*x3 +   3*x4] = [ 25]
[  2*x1 +  -1*x2 +  10*x3 +  -1*x4] = [-11]
[  0*x1 +   3*x2 +  -1*x3 +   8*x4] = [ 15]
Iteration 1: [ 0.  0.  0.  0.]
Iteration 2: [ 0.6         2.32727273 -0.98727273  0.87886364]
Iteration 3: [ 1.03018182  2.03693802 -1.0144562   0.98434122]
Iteration 4: [ 1.00658504  2.00355502 -1.00252738  0.99835095]
Iteration 5: [ 1.00086098  2.00029825 -1.00030728  0.99984975]
Iteration 6: [ 1.00009128  2.00002134 -1.00003115  0.9999881 ]
Iteration 7: [ 1.00000836  2.00000117 -1.00000275  0.99999922]
Iteration 8: [ 1.00000067  2.00000002 -1.00000021  0.99999996]
Iteration 9: [ 1.00000004  1.99999999 -1.00000001  1.        ]
Iteration 10: [ 1.  2. -1.  1.]
```

```
Solution: [ 1.  2. -1.  1.]
Error: [  2.06480930e-08  -1.25551054e-08   3.61417563e-11   0.00000000e+00]
```

Program to solve arbitrary number of equations using Matlab

The following code uses the formula

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)} \right), \quad \begin{matrix} i = 1, 2, \ldots, n \\ k = 0, 1, 2, \ldots \end{matrix}$$

```
function x = gauss_seidel(A, b, x, iters)
    for i = 1:iters
        for j = 1:size(A,1)
            x(j) = (b(j) - sum(A(j,:)'.*x) + A(j,j)*x(j)) / A(j,j);
        end
    end
end
```

See also

- Conjugate gradient method
- Gaussian belief propagation
- Iterative method: Linear systems
- Kaczmarz method (a "row-oriented" method, whereas Gauss-Seidel is "column-oriented." See, for example, this paper (https://arxiv.org/abs/1507.05844).)
- Matrix splitting
- Richardson iteration

Notes

1. Sauer, Timothy (2006). *Numerical Analysis* (2nd ed.). Pearson Education, Inc. p. 109. ISBN 978-0-321-78367-7.
2. Gauss 1903, p. 279; direct link (http://gdz.sub.uni-goettingen.de/en/dms/loader/img/?PPN=PPN23601515X&DMDID=DMDLOG_0112&LOGID=LOG_0112&PHYSID=PHYS_0286).
3. Seidel, Ludwig (1874). "Über ein Verfahren, die Gleichungen, auf welche die Methode der kleinsten Quadrate führt, sowie lineäre Gleichungen überhaupt, durch successive Annäherung aufzulösen" (https://www.biodiversitylibrary.org/item/110049#page/621/mode/1up) [On a process for solving by successive approximation the equations to which the method of leas squares leads as well as linear equations generally]. *Abhandlungen der Mathematisch-Physikalischen Klasse der Königlich Bayerischen Akademie der Wissenschaften* (in Germa **11** (3): 81–108.
4. Golub & Van Loan 1996, p. 511.
5. Golub & Van Loan 1996, eqn (10.1.3)
6. Golub & Van Loan 1996, Thm 10.1.2.
7. Bagnara, Roberto (March 1995). "A Unified Proof for the Convergence of Jacobi and Gauss-Seidel Methods". *SIAM Review*. **37** (1): 93–97. CiteSeerX 10.1.1.26.5207 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.26.5207). doi:10.1137/1037008 (https://doi.org/10.1137%2F1037008). JSTOR 2132758 (https://www.jstor.org/stable/2132758).
8. Golub & Van Loan 1996, Thm 10.1.2

References

- Gauss, Carl Friedrich (1903), *Werke* (in German), vol. 9, Göttingen: Königlichen Gesellschaft der Wissenschaften.
- Golub, Gene H.; Van Loan, Charles F. (1996), *Matrix Computations* (3rd ed.), Baltimore: Johns Hopkins, ISBN 978-0-8018-5414-9.
- Black, Noel & Moore, Shirley. "Gauss-Seidel Method" (https://mathworld.wolfram.com/Gauss-SeidelMethod.html). *MathWorld*.

This article incorporates text from the article Gauss-Seidel_method (http://www.cfd-online.com/Wiki/Gauss-Seidel_method) on CFD-Wiki (http://www.cfd-online.com/Wiki/Main_Page) that is under the GFDL license.

External links

- "Seidel method" (https://www.encyclopediaofmath.org/index.php?title=Seidel_method), *Encyclopedia of Mathematics*, EMS Press, 2001 [1994]
- Gauss–Seidel from www.math-linux.com (http://www.math-linux.com/spip.php?article48)
- Gauss–Seidel (http://numericalmethods.eng.usf.edu/topics/gauss_seidel.html) From Holistic Numerical Methods Institute
- Gauss Siedel Iteration from www.geocities.com (https://web.archive.org/web/20090614041201/http://www.geocities.com/rsrirang2001/Mathematics/NumericalMethods/gsiedel/gsiedel.htm)
- The Gauss-Seidel Method (http://www.netlib.org/linalg/html_templates/node14.html#figgs)
- Bickson (https://arxiv.org/abs/0901.4192)
- Matlab code (https://web.archive.org/web/20090812100731/http://matlabdb.mathematik.uni-stuttgart.de/gauss_seidel.m?MP_ID=406)
- C code example (http://adrianboeing.blogspot.com/2010/02/solving-linear-systems.html)