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 a strictly diagonally dominant system of linear equations.

Procedure

1. Choose an **n X n** matrix

   Otherwise- Show Pop up – please select the number of rows equal to the number of Columns

2. Verify that the magnitude of the diagonal item in each row of the matrix is greater than or equal to the sum of the magnitudes of all other (non-diagonal) values in that row so that

   $$|a_{ii}| \geq \sum_{j \neq i} |a_{ij}| \quad \text{for all } i$$

   Otherwise- Show Pop up – entered matrix is not diagonally dominant

   Ensure that all of the diagonal elements are non-zero as well.

   $$\boldsymbol{a_{ii} \neq 0}$$

   Otherwise- Show Pop up – all of the diagonal elements must be non-zero

3. Decompose the given matrix **A** into a lower triangular matrix **L***, and a strictly upper triangular matrix **U**:

Let's assume, A linear system of the form **Ax = b**

$$L^* = \begin{bmatrix} a11 & 0 & \cdots & 0 \\ a21 & a22 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots \\ an1 & an2 & \cdots & ann \end{bmatrix};$$

$$U = \begin{bmatrix} 0 & a12 & \cdots & a1n \\ 0 & 0 & \dots & a2n \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix};$$

An upper triangular matrix differs significantly from a strictly upper triangular matrix in that an upper triangular matrix contains all zero elements below the diagonal while a strictly upper triangular matrix contains all zeroes on the diagonal also.

4. Now we've to choose a column matrix $\mathbf{x}^{(0)}$ of size **n X 1** containing only 1s

$$\mathbf{x}^{(0)} = \begin{bmatrix} x1 \\ x2 \\ \vdots \\ xn \end{bmatrix}, \text{ where, } x_1 = x_2 = x_3 = . . . = x_n = 1$$

5. The system of linear equations can be rewritten as

Ax = b
Or, (L* + U)x = b
Or, L*x + Ux = b
Or, L*x = b – Ux

Firstly, we can only guess starting points of **x**, then go to the next iteration and put the updated value of **x** as shown in the example below

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

Repeat the iteration process until either the value of $\mathbf{x}^{(k)}$ converges or the error, **(Ax – b)** becomes very tiny. For greater precision, you can choose a threshold value of **1e-8 or 1e-10**. Stop the iteration procedure if the value of the error, **(Ax - b),** is less than **1e-8 or 1e-10**. Additionally, you can initially set the program's total iterations to 1000 at the beginning.

Example

Use the iterative Gauss-Seidel method to solve the problem. There are equations

$$16x + 3y = 11$$
$$7x - 11y = 13$$

Taking values from the aforementioned equations, we obtain,

A = $\begin{bmatrix} 16 & 3 \\ 7 & -11 \end{bmatrix}$; b = $\begin{bmatrix} 11 \\ 13 \end{bmatrix}$;

Let's initialize, $x^{(0)}$ = $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$;

Then calculate,

1st iteration

$x^{(1)} = L*^{-1}(b – Ux^{(0)}))$
$= L*^{-1}b - L*^{-1}Ux^{(0)}$

$$= \begin{bmatrix} 16 & 0 \\ 7 & -11 \end{bmatrix}^{-1} * \begin{bmatrix} 11 \\ 13 \end{bmatrix} - \begin{bmatrix} 16 & 0 \\ 7 & -11 \end{bmatrix}^{-1} * \begin{bmatrix} 0 & 3 \\ 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.0625 & 0 \\ 0.0398 & -0.0909 \end{bmatrix} * \begin{bmatrix} 11 \\ 13 \end{bmatrix} - \begin{bmatrix} 0.0625 & 0 \\ 0.0398 & -0.0909 \end{bmatrix} * \begin{bmatrix} 0 & 3 \\ 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.6875 \\ -0.7439 \end{bmatrix} - \begin{bmatrix} 0 & -0.1875 \\ 0 & -0.1194 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.5000 \\ -0.8636 \end{bmatrix}$$

2nd Iteration

$x^{(2)} = L*^{-1}(b – Ux^{(1)}))$
$= L*^{-1}b - L*^{-1}Ux^{(1)}$

$$= \begin{bmatrix} 16 & 0 \\ 7 & -11 \end{bmatrix}^{-1} * \begin{bmatrix} 11 \\ 13 \end{bmatrix} - \begin{bmatrix} 16 & 0 \\ 7 & -11 \end{bmatrix}^{-1} * \begin{bmatrix} 0 & 3 \\ 0 & 0 \end{bmatrix} * \begin{bmatrix} 0.5000 \\ -0.8636 \end{bmatrix}$$

$$= \begin{bmatrix} 0.8494 \\ -0.6413 \end{bmatrix}$$

3rd Iteration

$x^{(3)} = L*^{-1}(b – Ux^{(2)}))$
$= L*^{-1}b - L*^{-1}Ux^{(2)}$
$= \begin{bmatrix} 0.8077 \\ -0.6678 \end{bmatrix}$

4th Iteration

$x^{(4)} = L*^{-1}(b – Ux^{(3)}))$
$= L*^{-1}b - L*^{-1}Ux^{(3)}$
$= \begin{bmatrix} 0.8127 \\ -0.6646 \end{bmatrix}$

5th Iteration

$x^{(5)} = L*^{-1}(b – Ux^{(4)}))$
$= L*^{-1}b - L*^{-1}Ux^{(4)}$
$= \begin{bmatrix} 0.8121 \\ -0.6650 \end{bmatrix}$

6th Iteration

$x^{(6)} = L*^{-1}(b – Ux^{(5)}))$
$= L*^{-1}b - L*^{-1}Ux^{(5)}$
$= \begin{bmatrix} 0.8122 \\ -0.6650 \end{bmatrix}$

7th Iteration

$x^{(7)} = L*^{-1}(b – Ux^{(6)}))$
$= L*^{-1}b - L*^{-1}Ux^{(6)}$
$= \begin{bmatrix} 0.8122 \\ -0.6650 \end{bmatrix}$

As expected, the algorithm converges to the exact solution

Now the approximate value of x and y are 0.8122 and -0.6650, respectively.

(for the programming perspective)
Repeat the iteration process until either the value of $\mathbf{x}^{(k)}$ converges or the error, **(Ax – b)** becomes very tiny. For greater precision, you can choose a threshold value of **1e-8 or 1e-10**. Stop the iteration procedure if the value of the error, (Ax - b), is less than 1e-8 or 1e-10. Additionally, you can initially set the program's total iterations to 1000 at the beginning.