Gauss-Seidel Iteration Method

MPHYCC-05 unit IV (Sem.-II)

Gauss-Seidel Iteration Method

Gauss–Seidel method is an iterative method to solve a set of linear equations and very much similar to Jacobi's method. This method is also known as Liebmann method or the method of successive displacement. The name successive displacement is because the second unknown is determined from the first unknown in the current iteration, the third unknown is determined from the first and second unknowns. This method was developed by German mathematicians Carl Friedrich Gauss and Philipp Ludwig von Seidel. The method can be applied to any matrix with non-zero diagonal elements. However, the convergence is only possible if the matrix is either diagonally dominant, or symmetric & positive definite. The process adopted by this method for solving a set of linear equations is as follows.

Let us assume a set of linear equations in the matrix form is as follows:

$$AX = B$$

Where

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdot & \cdot & \cdot\, a_{1n} \\ a_{21} & a_{22} & \cdot & \cdot & \cdot\, a_{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdot & \cdot & \cdot\, a_{nn} \end{bmatrix} \quad X = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{bmatrix} \quad B = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Further, the matrix A can be decomposed into a lower triangular part (L_*) and an strict upper triangular component (U) as:

$$A = L_* + U$$

Where

$$L_* = \begin{bmatrix} a_{11} & 0 & . & 0 & . & 0 \\ a_{21} & a_{22} & . & . & . & 0 \\ \vdots & \vdots & & & & \vdots \\ a_{n1} & a_{n2} & . & . & . & a_{nn} \end{bmatrix} \qquad U = \begin{bmatrix} 0 & a_{12} & . & . & . & a_{1n} \\ 0 & 0 & . & . & . & a_{2n} \\ \vdots & \vdots & & & & \vdots \\ 0 & 0 & . & . & . & 0 \end{bmatrix}$$

And equation AX=B can be written as

$$(L* + U)X = B$$

Therefore, the solution can be obtained iteratively via using the following relation:

$$X^{(k+1)} = L_*^{-1}\left(B - U\,X^{(k)}\right)$$

Where $X^{(k)}$ and $X^{(k+1)}$ are the kth and (k+1)th iteration of X.

The elements of $X^{(k+1)}$ can be computed sequentially using the element-based forward substitution formula:

$$X_i^{(k+1)} = \frac{1}{a_{ii}}\left(B_i - \sum_{j=1}^{i-1} a_{ij}\,X_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}\,X_j^{(k)}\right),$$

$$i = 1, 2, \dots, n$$

Thus the calculation of $X^{(k+1)}$ is using the value of $X^{(k+1)}$ that have already computed and only those value of $X^{(k)}$ that have not been calculated in the (k+1) iteration. Hence, only one storage vector is required because the elements can be overwritten as they computed which is opposite to Jacobi method. We know that in Jacobi method the computations for each element can be done in parallel but the parallel computation is not possible in the Gauss-Seidel method.

The process adopted by this method can be understood completely by starting with the following motivational example.

Example 1: Find the solution of given set of equations using Gauss-Siedel Iterative method.

$$16x_1 + 3x_2 = 11$$
$$7x_1 - 11x_2 = 13$$

Solution: Matrix form of equation is AX=B where

$$A=\begin{bmatrix}16 & 3\\ 7 & -11\end{bmatrix} \qquad B=\begin{bmatrix}11\\ 13\end{bmatrix} \qquad X=\begin{bmatrix}x_1\\ x_2\end{bmatrix}$$

Therefore, L_*, L_*^{-1}, $L_*^{-1}B$ and $-L_*^{-1}U$ are:

$$L_*=\begin{bmatrix}16 & 0\\ 7 & -11\end{bmatrix} \quad U=\begin{bmatrix}0 & 3\\ 0 & 0\end{bmatrix}$$

Using Gauss-Jordan method, we can find the inverse of the matrix as:

$$L_*=\begin{bmatrix}16 & 0\\ 7 & -11\end{bmatrix} \qquad I=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$$

R1→ 7R1
R2→ 16R2
$$\begin{bmatrix}112 & 0\\ 112 & -176\end{bmatrix} \qquad \begin{bmatrix}7 & 0\\ 0 & 16\end{bmatrix}$$

R2→ R2-R1
$$\begin{bmatrix}112 & 0\\ 0 & -176\end{bmatrix} \qquad \begin{bmatrix}7 & 0\\ -7 & 16\end{bmatrix}$$

R1→ R1/112
R2→ R2/-176
$$\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} \qquad \begin{bmatrix}0.0625 & 0\\ 0.0398 & -0.0909\end{bmatrix}$$

Therefore $$L_*^{-1}=\begin{bmatrix}0.0625 & 0\\ 0.0398 & -0.0909\end{bmatrix}$$

$$L_*^{-1}B=\begin{bmatrix}0.6875\\ -0.7439\end{bmatrix} \quad -L_*^{-1}U=\begin{bmatrix}0 & -0.1875\\ 0 & -0.1194\end{bmatrix}$$

Thus according to Gauss-Siedel Iteration method:

$$X^{(new)} = L_*^{-1}\left(B - U\,X^{(old)}\right)$$

Let's assume that the initial guess for solution is:

$$X=\begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

We found that the solution is converges even for 25 number of iterations. Out put after each iteration are given below.

```
[ 0.6875     -0.74431818]
[ 0.82705966 -0.65550749]
[ 0.81040765 -0.66610422]
[ 0.81239454 -0.66483984]
[ 0.81215747 -0.6649907 ]
[ 0.81218576 -0.6649727 ]
[ 0.81218238 -0.66497485]
[ 0.81218278 -0.66497459]
[ 0.81218274 -0.66497462]
[ 0.81218274 -0.66497462]
[ 0.81218274 -0.66497462]
[ 0.81218274 -0.66497462]
[ 0.81218274 -0.66497462]
[ 0.81218274 -0.66497462]
[ 0.81218274 -0.66497462]
[ 0.81218274 -0.66497462]
[ 0.81218274 -0.66497462]
[ 0.81218274 -0.66497462]
[ 0.81218274 -0.66497462]
[ 0.81218274 -0.66497462]
[ 0.81218274 -0.66497462]
[ 0.81218274 -0.66497462]
[ 0.81218274 -0.66497462]
[ 0.81218274 -0.66497462]
[ 0.81218274 -0.66497462]
[ 0.81218274 -0.66497462]
```

Therefore, the solution of the above set of linear equations is: $x_1=0.8122$ & $x_2 = -0.6650$

Python script is given below to solve the set of linear equations using Gauss Seidel iteration method.

……………………………………………………………………………..................................

```
import numpy as np
from scipy.linalg import solve
def GaussSeidel(A, B, x, n):
   L=np.tril(A)
   U=A-L
```

```
    for i in range(n):
        x = np.dot(np.linalg.inv(L), B - np.dot(U, x))
        print(x)
    return x

'''___Main___'''

A = eval(input('Enter the matrix A:'))
# as np.array([[a11,a12],[a21,a22]])
B = eval(input('Enter the matrix B:'))# as [b1,b2]
x = eval(input('Enter guess of x:'))  # as [x1,x2]
n = eval(input('Enter the number of Iterations:'))
x = GaussSeidel(A, B, x, n)
print ('Solution using the solve syntax:\n', solve(A, B))
```
……………………………………………………………………………..................................

After compiling the above python script and for the different matrices we have the following out puts:

……………………………………………………………………………………………..

```
Enter the matrix A:np.array([[2,1],[3,7]])
Enter the matrix B:[4,3]
Enter guess of x:[0,0]
Enter the number of Iterations:25
[ 2.         -0.42857143]
[ 2.21428571 -0.52040816]
[ 2.26020408 -0.54008746]
[ 2.27004373 -0.54430446]
[ 2.27215223 -0.5452081 ]
[ 2.27260405 -0.54540174]
[ 2.27270087 -0.54544323]
[ 2.27272161 -0.54545212]
[ 2.27272606 -0.54545403]
[ 2.27272701 -0.54545443]
[ 2.27272722 -0.54545452]
[ 2.27272726 -0.54545454]
[ 2.27272727 -0.54545454]
[ 2.27272727 -0.54545455]
[ 2.27272727 -0.54545455]
[ 2.27272727 -0.54545455]
[ 2.27272727 -0.54545455]
[ 2.27272727 -0.54545455]
[ 2.27272727 -0.54545455]
[ 2.27272727 -0.54545455]
[ 2.27272727 -0.54545455]
[ 2.27272727 -0.54545455]
[ 2.27272727 -0.54545455]
[ 2.27272727 -0.54545455]
[ 2.27272727 -0.54545455]
Solution using the solve syntax:
 [ 2.27272727 -0.54545455]
Hence solution is [ 2.27272727 -0.54545455]
```

……………………………………………………………………………….................................

```
Enter the matrix A:np.array([[2,1],[5,7]])
Enter the matrix B:[11,15]
Enter guess of x:[0,0]
Enter the number of Iterations:25
[ 5.5         -1.78571429]
[ 6.39285714 -2.42346939]
[ 6.71173469 -2.65123907]
[ 6.82561953 -2.73258538]
[ 6.86629269 -2.76163764]
[ 6.88081882 -2.77201344]
[ 6.88600672 -2.77571909]
[ 6.88785954 -2.77704253]
[ 6.88852127 -2.77751519]
[ 6.88875759 -2.777684  ]
[ 6.888842   -2.77774428]
[ 6.88887214 -2.77776582]
[ 6.88888291 -2.77777351]
[ 6.88888675 -2.77777625]
[ 6.88888813 -2.77777723]
[ 6.88888862 -2.77777758]
[ 6.88888879 -2.77777771]
[ 6.88888885 -2.77777775]
[ 6.88888888 -2.77777777]
[ 6.88888888 -2.77777777]
[ 6.88888889 -2.77777778]
[ 6.88888889 -2.77777778]
[ 6.88888889 -2.77777778]
[ 6.88888889 -2.77777778]
[ 6.88888889 -2.77777778]
Solution using the solve syntax:
 [ 6.88888889 -2.77777778]
Hence solution is [ 6.88888889 -2.77777778]
```

…………………………………………………………………………………………

```
Enter the matrix A:np.array([[5,-1,3],[-3,9,1],[2,-1,-7]])
Enter the matrix B:[-3,5,7]
Enter guess of x:[1,1,1]
Enter the number of Iterations:25
[-1.          0.11111111 -1.3015873 ]
[ 0.2031746   0.76790123 -1.05165029]
[ 0.18457042  0.73392906 -1.0521126 ]
[ 0.17805337  0.73180808 -1.05367162]
[ 0.17856459  0.73215171 -1.05357465]
[ 0.17857513  0.73214445 -1.0535706 ]
[ 0.17857125  0.7321427  -1.05357146]
[ 0.17857142  0.73214286 -1.05357143]
[ 0.17857143  0.73214286 -1.05357143]
[ 0.17857143  0.73214286 -1.05357143]
[ 0.17857143  0.73214286 -1.05357143]
[ 0.17857143  0.73214286 -1.05357143]
[ 0.17857143  0.73214286 -1.05357143]
[ 0.17857143  0.73214286 -1.05357143]
[ 0.17857143  0.73214286 -1.05357143]
[ 0.17857143  0.73214286 -1.05357143]
[ 0.17857143  0.73214286 -1.05357143]
[ 0.17857143  0.73214286 -1.05357143]
[ 0.17857143  0.73214286 -1.05357143]
[ 0.17857143  0.73214286 -1.05357143]
[ 0.17857143  0.73214286 -1.05357143]
[ 0.17857143  0.73214286 -1.05357143]
[ 0.17857143  0.73214286 -1.05357143]
[ 0.17857143  0.73214286 -1.05357143]
[ 0.17857143  0.73214286 -1.05357143]
Solution using the solve syntax:
 [ 0.17857143  0.73214286 -1.05357143]
Hence solution is [ 0.17857143  0.73214286 -1.05357143]
```

………………………………………………………………………