Jacobi's Iteration Method

MPHYCC-05 unit IV (Sem.-II)

Jacobi's Iteration method

Jacobi's method in numerical linear algebra is an iterative method to compute the solution of a strictly diagonally dominant system of linear equations. The method is named after Carl Gustav Jacob Jacobi. The method is a shorter version of the Jacobi transformation method of matrix diagonalization. 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

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

Further, the matrix A can be decomposed into a diagonal component (D), a strictly lower triangular part (L) and a strictly upper triangular component (U) as:

$$A=D + L + U$$

Where

$$D=\begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix} \quad L=\begin{bmatrix} 0 & 0 & \cdots & 0 \\ a_{21} & 0 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{bmatrix}$$

$$U = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

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

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

Where $X^{(k)}$ and $X^{(k+1)}$ are the k^{th} and $(k+1)^{th}$ iteration of X. The elements of $X^{(k+1)}$ can be computed using the element-based formula:

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

Thus new value of X is calculated after putting the previous iterative value of X in the above equation till the required accuracy is achieved. However, we overwrite, $X_i^{(k)}$ with $X_i^{(k+1)}$. The important thing is that the method should be converging for having a solution. And the sufficient but not necessary condition for the method to converge is that the matrix is strictly diagonally dominant. That means for each row the absolute value of diagonal term is greater than the sum of the absolute values of the other terms:

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

However, the method sometimes converges even if these conditions are not satisfied. Moreover, the standard convergence condition for any iterative method is defined in terms of the spectral radius, ρ() of the iteration matrix. And the spectral radius of the matrix is equal to the largest absolute value of its eigenvalues. The convergence condition for the method is when the spectral radius, ρ() of the iteration matrix $D^{-1}(L + U)$ is less than 1; thus

$$\rho\left(D^{-1}(L+U)\right) < 1$$

The method is simple and numerically robust and each iterations quite fast. However, the method is computing the independent variables of the linear equations in parallel and independent way. Therefore the method might require much iteration as well as good memory power of computational system. In-order to understand the process completely, we start with a motivational example as given below.

Example 1: Find the solution of the given set of equations using Jacobi's Iterative method.

$$2x_1 + x_2 = 11$$
$$5x_1 + 7x_2 = 13$$

Solution: Matrix form of equation is AX=B where

$$A=\begin{bmatrix}2 & 1\\ 5 & 7\end{bmatrix} \qquad B=\begin{bmatrix}11\\ 13\end{bmatrix} \qquad X=\begin{bmatrix}x_1\\ x_2\end{bmatrix}$$

Therefore, D, L, U, D^{-1} and $-D^{-1}(L+U)$ and $D^{-1}B$ are:

$$D=\begin{bmatrix}2 & 0\\ 0 & 7\end{bmatrix} \qquad L=\begin{bmatrix}0 & 0\\ 5 & 0\end{bmatrix} \qquad U=\begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}$$

Inverse of a diagonal matrix is a diagonal matrix and values of the diagonal elements are the reciprocal of the corresponding values of elements of the original matrix provided that each element of the diagonal is non zero. Therefore

$$D^{-1}=\begin{bmatrix}1/2 & 0\\ 0 & 1/7\end{bmatrix} \qquad -D^{-1}(L+U)=\begin{bmatrix}0 & -1/2\\ -5/7 & 0\end{bmatrix}$$

$$D^{-1}B=\begin{bmatrix}11/2\\ 13/7\end{bmatrix}$$

Thus according to Jacobi's Iteration method:

$$X^{(new)} = D^{-1}\left(B-(L+U)\,X^{(old)}\right)$$

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.

```
[5.5        1.85714286]
[ 4.57142857 -2.07142857]
[ 6.53571429 -1.40816327]
[ 6.20408163 -2.81122449]
[ 6.90561224 -2.57434402]
[ 6.78717201 -3.07543732]
[ 7.03771866 -2.99083715]
[ 6.99541858 -3.16979904]
[ 7.08489952 -3.1395847 ]
[ 7.06979235 -3.20349966]
[ 7.10174983 -3.19270882]
[ 7.09635441 -3.21553559]
[ 7.1077678  -3.21168172]
[ 7.10584086 -3.21983414]
[ 7.10991707 -3.21845776]
[ 7.10922888 -3.22136934]
[ 7.11068467 -3.22087777]
[ 7.11043889 -3.22191762]
[ 7.11095881 -3.22174206]
[ 7.11087103 -3.22211344]
[ 7.11105672 -3.22205074]
[ 7.11102537 -3.22218337]
[ 7.11109168 -3.22216098]
[ 7.11108049 -3.22220835]
[ 7.11110417 -3.22220035]
[ 7.11111111 -3.22222222]
```

Therefore, the solution of the above set of linear equations is: x_1=7.1111 & x_2 = -3.2222

Python script is given below to solve the set of linear equations using Jacobi's iteration method.

```
……..
import numpy as np
from scipy.linalg import solve

def Jacobi(A, B, x, n):
# computing the diagonal matrix
    D = np.diag(A)
```

```
# computing the sum of upper and
#lower triangular matrix (R=U+L=A-D)
   R = A - np.diagflat(D)
# computing new solution from old solution
   for i in range(n):
      x = (B - np.dot(R,x))/ D
      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 = Jacobi(A, B, x, n)
# Direct command that is solve to find the solution of a set of linear equations
 print ('Solution using the solve command:', 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:[3,5]
Enter guess of x:[0,0]
Enter the number of Iterations:25
[1.5        0.71428571]
[1.14285714 0.07142857]
[1.46428571 0.2244898 ]
[1.3877551  0.08673469]
[1.45663265 0.11953353]
[1.44023324 0.09001458]
[1.45499271 0.0970429 ]
[1.45147855 0.09071741]
[1.4546413  0.09222348]
[1.45388826 0.09086802]
```

[1.45456599 0.09119075]
[1.45440463 0.09090029]
[1.45454986 0.09096945]
[1.45451528 0.0909072]
[1.4545464 0.09092202]
[1.45453899 0.09090869]
[1.45454566 0.09091186]
[1.45454407 0.090909]
[1.4545455 0.09090968]
[1.45454516 0.09090907]
[1.45454546 0.09090922]
[1.45454539 0.09090909]
[1.45454546 0.09090912]
[1.45454544 0.09090909]
[1.45454545 0.0909091]
Solution using the solve command: [1.45454545 0.09090909]

Hence the solution is (1.4545, 0.0909)

……

Enter the matrix A:np.array([[5,-2,3],[-3,9,1],[2,-1,-7]])
Enter the matrix B:[-1,2,3]
Enter guess of x:[1,1,1]
Enter the number of Iterations:25
[-0.4   0.44444444 -0.28571429]
[0.14920635 0.12063492 -0.60634921]
[0.21206349 0.33932981 -0.4031746]
[0.17763668 0.33770723 -0.41645755]
[0.18495742 0.32770751 -0.42606198]
[0.18672019 0.33121492 -0.42254181]
[0.18601105 0.33141138 -0.42253922]
[0.18608808 0.33117471 -0.4227699]
[0.18613182 0.33122602 -0.42271408]
[0.18611885 0.33123439 -0.42270891]
[0.1861191   0.3312295 -0.42271381]
[0.18612009 0.33123012 -0.42271304]
[0.18611987 0.33123037 -0.42271285]
[0.18611986 0.33123027 -0.42271295]
[0.18611988 0.33123028 -0.42271294]
[0.18611987 0.33123029 -0.42271293]
[0.18611987 0.33123028 -0.42271293]

[0.18611987 0.33123028 -0.42271293]
[0.18611987 0.33123028 -0.42271293]
[0.18611987 0.33123028 -0.42271293]
[0.18611987 0.33123028 -0.42271293]
[0.18611987 0.33123028 -0.42271293]
[0.18611987 0.33123028 -0.42271293]
[0.18611987 0.33123028 -0.42271293]
[0.18611987 0.33123028 -0.42271293]
Solution using the solve command: [0.18611987 0.33123028 -0.42271293]

Hence the solution is (0.186, 0.331, -0.423)

……

Enter the matrix A:np.array([[20,1,-2],[3,20,-1],[2,-3,20]])
Enter the matrix B:[17,-18,25]
Enter guess of x:[0,0,0]
Enter the number of Iterations:25
[0.85 -0.9   1.25]
[1.02 -0.965 1.03]
[1.00125 -1.0015   1.00325]
[1.0004   -1.000025 0.99965]
[0.99996625 -1.0000775   0.99995625]
[0.9999995 -0.99999712 0.99999175]
[0.99999903 -1.00000034 1.00000048]
[1.00000007 -0.99999983 1.00000005]
[1.   -1.00000001 1.00000002]
[1. -1. 1.]
[1. -1. 1.]
[1. -1. 1.]
[1. -1. 1.]
[1. -1. 1.]
[1. -1. 1.]
[1. -1. 1.]
[1. -1. 1.]
[1. -1. 1.]
[1. -1. 1.]
[1. -1. 1.]
[1. -1. 1.]
[1. -1. 1.]
[1. -1. 1.]
[1. -1. 1.]
[1. -1. 1.]

Solution using the solve command: [1. -1. 1.]

Hence the solution is (1, -1, 1)

.....

Enter the matrix A:np.array([[10,2,-1],[1,8,3],[-2,-1,10]])
Enter the matrix B:[7,-4,9]
Enter guess of x:[0,0,0]
Enter the number of Iterations:25
[0.7 -0.5 0.9]
[0.89 -0.925 0.99]
[0.984 -0.9825 0.9855]
[0.99505   -0.9925625 0.99855]
[0.9983675 -0.9988375   0.99975375]
[0.99974288 -0.99970359 0.99978975]
[0.99991969 -0.99988902 0.99997822]
[0.99997562 -0.99998179 0.99999504]
[0.99999586 -0.99999509 0.99999695]
[0.99999871 -0.99999834 0.99999966]
[0.99999963 -0.99999971 0.99999991]
[0.99999993 -0.99999992 0.99999996]
[0.99999998 -0.99999997 0.99999999]
[0.99999999 -1.   1.   ]
[1. -1. 1.]
[1. -1. 1.]
[1. -1. 1.]
[1. -1. 1.]
[1. -1. 1.]
[1. -1. 1.]
[1. -1. 1.]
[1. -1. 1.]
[1. -1. 1.]
[1. -1. 1.]
[1. -1. 1.]
Solution using the solve command: [1. -1. 1.]

Hence the solution is (1, -1, 1)

……