- # Gauss Jacobi method

S No	Matrix	Tolerance	Total No. Iterations	Root
1	83x + 11y – 4z = 95 7x +52y + 13z = 104 3x + 8y + 29z = 71	0.01	5	X = 1.05874766 Y = 1.37180339 Z = 1.96550715
		0.001	7	X = 1.05800232 Y = 1.3675643 Z = 1.96202466
		0.0001	9	X = 1.05793371 Y = 1.36719964 Z = 1.96171606
2	8x – 3y + 2z = 45 4x + 11y – z = 71 6x + 3y + 12z = 35	0.01	7	X = 7.3977288 Y = 3.61766817 Z = -1.6785675
		0.001	9	x = 7.40110315 y = 3.60984686 z = -1.68693335
		0.0001	11	x = 7.40029077 y = 3.61017725 z = -1.6860157
3	10.0x + -5.0y + -3.0z = 3.0 4.0x + -10.0y + 3.0z = -3.0 1.0x + 6.0y + 10.0z = -3.0	0.01	5	X = 0.28056 Y = 0.26238 Z = -0.48729
		0.001	8	X = 0.28774125 Y = 0.26851257 Z = -0.48975093
		0.0001	11	X = 0.2870932 Y = 0.26795769 Z = -0.48949285

- **Gauss Siedel method**

S No	Matrix	Tolerance	Total No. Iterations	Root
1	83x + 11y – 4z = 95 7x +52y + 13z = 104 3x + 8y + 29z = 71	0.01	4	X = 1.05175649 Y = 1.36926293 Z = 1.96174576
		0.001	5	X = 1.0576517 Y = 1.36718737 Z = 1.96170848
		0.0001	6	X = 1.05792497 Y = 1.3671599 Z = 1.96168779
2	8x – 3y + 2z = 45 4x + 11y – z = 71 6x + 3y + 12z = 35	0.01	5	X = 7.39687049 Y = 3.61077985 Z = -1.68446354
		0.001	6	x = 7.40015833 y = 3.61044574 z = -1.68602393
		0.0001	7	x = 7.40042314 y = 3.61020759 z = -1.6860968
3	10.0x + -5.0y + -3.0z = 3.0 4.0x + -10.0y + 3.0z = -3.0 1.0x + 6.0y + 10.0z = -3.0	0.01	4	X = 0.2829972 Y = 0.26641608 Z = -0.48814937
		0.001	5	X = 0.28676323 Y = 0.26826048 Z = -0.48963261
		0.0001	6	X = 0.28724046 Y = 0.2680064 Z = -0.48952789

```
import numpy as np

ITERATION_LIMIT = 1000

# initialize the matrix

A = np.array([[10., -5., -3.],

 [4., -10., 3.],

 [1., 6., 10.]])

# initialize the RHS vector

b = np.array([3.,-3.,-3.])

for i in range(A.shape[0]):

    row = ["{0:3g}*x{1}".format(A[i, j], j + 1) for j in range(A.shape[1])]

    print("[{0}] = [{1:3g}]".format(" + ".join(row), b[i]))

    x = np.zeros_like(b)

for it_count in range(1, ITERATION_LIMIT):

    x_new = np.zeros_like(x)

    print("Iteration {0}: {1}".format(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, atol=0.01, rtol=0.):

        break

    x = x_new

print("Solution: {0}".format(x))

print(it_count)

error = np.dot(A, x) - b

print("Error: {0}".format(error))
```

```
[ 10*x1 +  -5*x2 +  -3*x3] = [  3]
[  4*x1 + -10*x2 +   3*x3] = [ -3]
[  1*x1 +   6*x2 +  10*x3] = [ -3]
Iteration 1: [0. 0. 0.]
Iteration 2: [ 0.3    0.42  -0.582]
Iteration 3: [ 0.3354    0.25956  -0.489276]
Iteration 4: [ 0.2829972   0.26641608 -0.48814937]
Solution: [ 0.2829972   0.26641608 -0.48814937]
4
Error: [-0.0376603  0.0033799  0.       ]
```