

```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Fri Jun 11 00:03:50 2021
4
5  @author: SESWARAN
6  """
7  ### Library Import
8  import numpy as np
9  from math import log10,floor
10 from numpy.linalg import *
11
12 import sys
13 ### Significant digit round up
14 def round_sig(x, sig=4):
15     return round(x, sig-int(floor(log10(abs(x))))-1)
16
17
18 ### Get User Input & Generate Random 3X3 Matrix
19 print('Number of unknowns = 3. i.e. 3X3 matrix')
20 unknowns = 3
21 elementDtype = int(input('Enter number of data type for martix elements (int = 1/
float = 2):'))
22 sigDigits = 4
23 print('Enter the number of significant digits (d)=4')
24 diagDomYes = input('Make it diagonally dominant if it is already not (y/n):')
25 # Generate Random martix
26 if elementDtype == 2:
27     matrixA = np.random.rand(unknowns,unknowns+1)
28     # significant digit reduction
29     for i in range(unknowns):
30         for j in range (unknowns+1):
31             matrixA[i][j] = round_sig(matrixA[i][j],sigDigits)
32     del i,j
33
34 elif elementDtype == 1:
35     rangeRandom = int(input('Enter max value range for array values (e.g. 10):'))
36     matrixA = np.random.randint(rangeRandom,size=(unknowns,unknowns+1))
37 else:
38     print('Incorrect Data type entered. Enter Correct Data type!!')
39
40 print('\nGenerated Linear system equations are:')
41 print(f'{matrixA[0][0]}x + {matrixA[0][1]}y + {matrixA[0][2]}z = {matrixA[0][3]}')
42 print(f'{matrixA[1][0]}x + {matrixA[1][1]}y + {matrixA[1][2]}z = {matrixA[1][3]}')
43 print(f'{matrixA[2][0]}x + {matrixA[2][1]}y + {matrixA[2][2]}z = {matrixA[2][3]}')
44 ### Check diagonal dominance
45
46 matrixA_Coeff = matrixA[:,-1]
47 D = np.diag(np.abs(matrixA_Coeff)) # Find diagonal coefficients
48 S = np.sum(np.abs(matrixA_Coeff), axis=1) - D # Find row sum without diagonal
49 if np.all(D > S):
50     print('\nmatrix is diagonally dominant')
51     # Print Matrix
52     print('Linear system equations are:')
53     print(f'{matrixA[0][0]}x + {matrixA[0][1]}y + {matrixA[0][2]}z = {matrixA[0][3]}')
54     print(f'{matrixA[1][0]}x + {matrixA[1][1]}y + {matrixA[1][2]}z = {matrixA[1][3]}')
55     print(f'{matrixA[2][0]}x + {matrixA[2][1]}y + {matrixA[2][2]}z = {matrixA[2][3]}')
56     diagDom = True
57 else:
58     print('\nNOT diagonally dominant, converting to diagonal dominance')
59     diagDom = False
60     if diagDomYes == 'y' or diagDomYes == 'Y':
61         # make it diagonally dominant
62         for i in range(3):
63             d = matrixA[i][i]
64             matrixA[i][i] = sum(matrixA[i]) - d + 1
65             # Print Matrix
66             print('\nNew Linear system equations are:')
67             print(f'{matrixA[0][0]}x + {matrixA[0][1]}y + {matrixA[0][2]}z = {matrixA[0][
3]}')
68             print(f'{matrixA[1][0]}x + {matrixA[1][1]}y + {matrixA[1][2]}z = {matrixA[1][
3]}')
69             print(f'{matrixA[2][0]}x + {matrixA[2][1]}y + {matrixA[2][2]}z = {matrixA[2][

```

```

3]})
70
71
72
73 #diagonally dominant form
74 print('\nDiagonally dominant form:')
75 print(f'x = {matrixA[0][3]}/{matrixA[0][0]} - {matrixA[0][1]}y/{matrixA[0][0]} - {
matrixA[0][2]}z/{matrixA[0][0]}')
76 print(f'y = {matrixA[1][3]}/{matrixA[1][1]} - {matrixA[1][0]}x/{matrixA[1][1]} - {
matrixA[1][2]}z/{matrixA[1][1]}')
77 print(f'z = {matrixA[2][3]}/{matrixA[2][2]} - {matrixA[2][0]}x/{matrixA[2][2]} - {
matrixA[2][1]}y/{matrixA[2][2]}')
78
79
80
81
82 %% Functions - Jacobi Method
83
84 xF = lambda x,y,z: (matrixA[0][3]/matrixA[0][0]) - (matrixA[0][1]*y/matrixA[0][0]) -
(matrixA[0][2]*z/matrixA[0][0])
85 yF = lambda x,y,z: (matrixA[1][3]/matrixA[1][1]) - (matrixA[1][0]*x/matrixA[1][1]) -
(matrixA[1][2]*z/matrixA[1][1])
86 zF = lambda x,y,z: (matrixA[2][3]/matrixA[2][2]) - (matrixA[2][0]*x/matrixA[2][2]) -
(matrixA[2][1]*y/matrixA[2][2])
87 print('\nGauss Jacobi Method')
88 # Initial setup
89 x0 = 0
90 y0 = 0
91 z0 = 0
92 count = 1
93
94 e = 0.01 # 1% error
95 e1 = 1
96 e2 = 1
97 e3 = 1
98 print('\nCount\tx\ty\tz\n')
99
100 condition = True
101
102 while condition:
103     x1 = xF(x0,y0,z0)
104     y1 = yF(x0,y0,z0)
105     z1 = zF(x0,y0,z0)
106     print('%d\t%.4f\t%.4f\t%.4f\n' %(count, x1,y1,z1))
107     if count>1:
108         e1 = abs((x1-x0)/x1)
109         e2 = abs((y1-y0)/y1)
110         e3 = abs((z1-z0)/z1)
111
112
113     count += 1
114     x0 = x1
115     y0 = y1
116     z0 = z1
117
118     condition = e1>e and e2>e and e3>e
119
120 %% Functions - Seidel Method
121
122 xF = lambda x,y,z: (matrixA[0][3]/matrixA[0][0]) - (matrixA[0][1]*y/matrixA[0][0]) -
(matrixA[0][2]*z/matrixA[0][0])
123 yF = lambda x,y,z: (matrixA[1][3]/matrixA[1][1]) - (matrixA[1][0]*x/matrixA[1][1]) -
(matrixA[1][2]*z/matrixA[1][1])
124 zF = lambda x,y,z: (matrixA[2][3]/matrixA[2][2]) - (matrixA[2][0]*x/matrixA[2][2]) -
(matrixA[2][1]*y/matrixA[2][2])
125 print('\nGauss Seidel Method')
126 # Initial setup
127 x0 = 0
128 y0 = 0
129 z0 = 0
130 count = 1
131

```

```
132 e = 0.01 # 1% error
133 e1 = 1
134 e2 = 1
135 e3 = 1
136 print('\nCount\tx\ty\tz\n')
137
138 condition = True
139
140 while condition:
141     x1 = xF(x0,y0,z0)
142     y1 = yF(x1,y0,z0)
143     z1 = zF(x1,y1,z0)
144     print('%d\t%.4f\t%.4f\t%.4f\n' %(count, x1,y1,z1))
145     if count>1:
146         e1 = abs((x1-x0)/x1)
147         e2 = abs((y1-y0)/y1)
148         e3 = abs((z1-z0)/z1)
149
150
151     count += 1
152     x0 = x1
153     y0 = y1
154     z0 = z1
155
156     condition = e1>e and e2>e and e3>e
157
158
```