```python
# -*- coding: utf-8 -*-
"""
Created on Fri Jun 11 00:03:50 2021

@author: SESWARAN
"""
#%% Library Import
import numpy as np
from math import log10,floor
from numpy.linalg import *

import sys
#%% Significant digit round up
def round_sig(x, sig=4):
    return round(x, sig-int(floor(log10(abs(x))))-1)


#%% Get User Input & Generate Random 3X3 Matrix
print('Number of unknowns = 3. i.e. 3X3 matrix')
unknowns = 3
elementDtype =  int(input('Enter number of data type for martix elements (int = 1/ float = 2):'))
sigDigits = 4
print('Enter the number of significant digits (d)=4')
diagDomYes =  input('Make it diagonally dominant if it is already not (y/n):')
# Generate Randon martix
if elementDtype == 2:
    matrixA = np.random.rand(unknowns,unknowns+1)
    # significant digit reduction
    for i in range(unknowns):
        for j in range (unknowns+1):
            matrixA[i][j] = round_sig(matrixA[i][j],sigDigits)
    del i,j

elif elementDtype == 1:
    rangeRandom = int(input('Enter max value range for array values (e.g. 10):'))
    matrixA = np.random.randint(rangeRandom,size=(unknowns,unknowns+1))
else:
    print('Incorrect Data type entered. Enter Correct Data type!!')

print('\nGenerated Linear system equations are:')
print(f'{matrixA[0][0]}x + {matrixA[0][1]}y + {matrixA[0][2]}z = {matrixA[0][3]}')
print(f'{matrixA[1][0]}x + {matrixA[1][1]}y + {matrixA[1][2]}z = {matrixA[1][3]}')
print(f'{matrixA[2][0]}x + {matrixA[2][1]}y + {matrixA[2][2]}z = {matrixA[2][3]}')
 #%% Check diagnonal dominance

matrixA_Coeff = matrixA[:,:-1]
D = np.diag(np.abs(matrixA_Coeff)) # Find diagonal coefficients
S = np.sum(np.abs(matrixA_Coeff), axis=1) - D # Find row sum without diagonal
if np.all(D > S):
    print('\nmatrix is diagonally dominant')
    # Print Matrix
    print('Linear system equations are:')
    print(f'{matrixA[0][0]}x + {matrixA[0][1]}y + {matrixA[0][2]}z = {matrixA[0][3]}')
    print(f'{matrixA[1][0]}x + {matrixA[1][1]}y + {matrixA[1][2]}z = {matrixA[1][3]}')
    print(f'{matrixA[2][0]}x + {matrixA[2][1]}y + {matrixA[2][2]}z = {matrixA[2][3]}')
    diagDom = True
else:
    print('\nNOT diagonally dominant, converting to diagonal dominance')
    diagDom = False
    if diagDomYes == 'y' or diagDomYes == 'Y':
        # make it diagnolly dominant
        for i in range(3):
            d = matrixA[i][i]
            matrixA[i][i] = sum(matrixA[i]) - d + 1
            # Print Matrix
        print('\nNew Linear system equations are:')
        print(f'{matrixA[0][0]}x + {matrixA[0][1]}y + {matrixA[0][2]}z = {matrixA[0][3]}')
        print(f'{matrixA[1][0]}x + {matrixA[1][1]}y + {matrixA[1][2]}z = {matrixA[1][3]}')
        print(f'{matrixA[2][0]}x + {matrixA[2][1]}y + {matrixA[2][2]}z = {matrixA[2][
```

```
        3]}')



#diagonally dominant form
print('\nDiagonally dominant form:')
print(f'x = {matrixA[0][3]}/{matrixA[0][0]} - {matrixA[0][1]}y/{matrixA[0][0]} - {matrixA[0][2]}z/{matrixA[0][0]}')
print(f'y = {matrixA[1][3]}/{matrixA[1][1]} - {matrixA[1][0]}x/{matrixA[1][1]} - {matrixA[1][2]}z/{matrixA[1][1]}')
print(f'z = {matrixA[2][3]}/{matrixA[2][2]} - {matrixA[2][0]}x/{matrixA[2][2]} - {matrixA[2][1]}y/{matrixA[2][2]}')




#%% Functions - Jacobi Method

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])
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])
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])
print('\nGauss Jacobi Method')
# Initial setup
x0 = 0
y0 = 0
z0 = 0
count = 1

e = 0.01 # 1% error
e1 = 1
e2 = 1
e3 = 1
print('\nCount\tx\ty\tz\n')

condition = True

while condition:
    x1 = xF(x0,y0,z0)
    y1 = yF(x0,y0,z0)
    z1 = zF(x0,y0,z0)
    print('%d\t%0.4f\t%0.4f\t%0.4f\n' %(count, x1,y1,z1))
    if count>1:
        e1 = abs((x1-x0)/x1)
        e2 = abs((y1-y0)/y1)
        e3 = abs((z1-z0)/z1)


    count += 1
    x0 = x1
    y0 = y1
    z0 = z1

    condition = e1>e and e2>e and e3>e

#%% Functions - Seidel Method

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])
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])
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])
print('\nGauss Seidel Method')
# Initial setup
x0 = 0
y0 = 0
z0 = 0
count = 1

```

```python
e = 0.01 # 1% error
e1 = 1
e2 = 1
e3 = 1
print('\nCount\tx\ty\tz\n')

condition = True

while condition:
    x1 = xF(x0,y0,z0)
    y1 = yF(x1,y0,z0)
    z1 = zF(x1,y1,z0)
    print('%d\t%0.4f\t%0.4f\t%0.4f\n' %(count, x1,y1,z1))
    if count>1:
        e1 = abs((x1-x0)/x1)
        e2 = abs((y1-y0)/y1)
        e3 = abs((z1-z0)/z1)


    count += 1
    x0 = x1
    y0 = y1
    z0 = z1

    condition = e1>e and e2>e and e3>e
```