$$x = \frac{-6-(y+z+u)}{5}$$

となる．他の未知数も，

$$\begin{bmatrix} y= & \frac{2-x-z-u}{3} \\ z= & \frac{9-(x-2y+u)}{9} \\ u= & \end{bmatrix}$$

となる．適当に初期値(x_0, y_0, z_0, u_0)をとり，下側の方程式に代入すると，得られた出力(x_1, y_1, z_1, u_1)はより精解に近い値となる．これを繰り返すことによって精解が得られる．これをヤコビ(Jacobi)法と呼び，係数行列の対角要素が非対角要素にくらべて大きいときに適用できる．多くの現実の問題ではこの状況が成り立っている．

Gauss-Seidel法はJacobi法の高速版である．n番目の解の組が得られた後に一度に次の解の組に入れ替えるのではなく，得られた解を順次改良した解として使っていく．これにより，収束が早まる．以下にはヤコビ法のコードを示した．x1[i]の配列を変数に換えるだけで，Gauss-Seidel法となる．

In [14]:
```
# Jacobi iterative method for solving Ax=b by bob
import numpy as np
np.set_printoptions(precision=6, suppress=True)

A=np.array([[5,1,1,1],[1,3,1,1],[1,-2,-9,1],[1,3,-2,5]])
b=np.array([-6,2,-7,3])
n=4
x0=np.zeros(n)
x1=np.zeros(n)

for iter in range(0, 20):
    for i in range(0, n):
        x1[i]=b[i]
        for j in range(0, n):
            x1[i]=x1[i]-A[i][j]*x0[j]
        x1[i]=x1[i]+A[i][i]*x0[i]
        x1[i]=x1[i]/A[i][i]
    for j in range(0, n):
        x0[j]=x1[j]
    print(x0)
```

```
[-1.2      0.666667  0.777778  0.6     ]
[-1.608889  0.607407  0.562963  0.751111]
[-1.584296  0.764938  0.54749   0.782519]
[-1.618989  0.751429  0.518705  0.676892]
[-1.589405  0.807797  0.506116  0.680422]
[-1.598867  0.800956  0.497269  0.635649]
[-1.586775  0.821983  0.492763  0.638108]
[-1.590571  0.818635  0.489707  0.621271]
[-1.585923  0.826531  0.488159  0.622816]
[-1.587501  0.824982  0.487092  0.61653 ]
[-1.585721  0.82796   0.486563  0.617348]
[-1.586374  0.82727   0.48619   0.614993]
[-1.585691  0.828397  0.486009  0.615389]
[-1.585959  0.828098  0.485878  0.614503]
[-1.585696  0.828526  0.485817  0.614684]
[-1.585805  0.828398  0.485771  0.61435 ]
[-1.585704  0.828561  0.48575   0.61443 ]
[-1.585748  0.828508  0.485734  0.614304]
[-1.585709  0.82857   0.485727  0.614338]
[-1.585727  0.828548  0.485721  0.61429 ]
```

もう少しpython(numpy)の機能を使うとおしゃれな書き方ができる．以下は，ネットから拾ってきたcode. 上(by bob)とやってることの本質的なところは同じですが，numpyのindexを自動的に判断する機能を使って簡潔に書いています．こういうのを読める，書けるようになれば，あなたもpython使い．私はまだpythonに馴染めてなくて，いちいち頭の中で治しています．．．

In [15]:
```
# https://www.quantstart.com/articles/Jacobi-Method-in-Python-and-NumPy
from numpy import array, zeros, diag, diagflat, dot

np.set_printoptions(precision=6, suppress=True)

def jacobi(A,b,N=25,x=None):
    if x is None:
        x = zeros(len(A[0]))

    D = diag(A)
    R = A - diagflat(D)
    # Iterate for N times
    for i in range(N):
        x = (b - dot(R,x)) / D
        print(x)
    return x

A = array([[5,1,1,1],[1,3,1,1],[1,-2,-9,1],[1,3,-2,5]])
b = array([-6,2,-7,3])

sol = jacobi(A,b,N=10)
```

```
[-1.2      0.666667  0.777778  0.6     ]
[-1.608889  0.607407  0.562963  0.751111]
[-1.584296  0.764938  0.54749   0.782519]
[-1.618989  0.751429  0.518705  0.676892]
[-1.589405  0.807797  0.506116  0.680422]
[-1.598867  0.800956  0.497269  0.635649]
[-1.586775  0.821983  0.492763  0.638108]
[-1.590571  0.818635  0.489707  0.621271]
[-1.585923  0.826531  0.488159  0.622816]
[-1.587501  0.824982  0.487092  0.61653 ]
```

課題

行列AをLU分解

次の行列AをLU分解し，上・下三角行列を求めよ．さらに連立方程式の解を求めよ．

$$\begin{bmatrix} x+4y+3z \\ x-2y+z \\ 2x-2y-z \end{bmatrix} = \begin{bmatrix} 11 \\ 11 \\ 11 \end{bmatrix}$$

pivot付きのLU分解

次の連立方程式の拡大行列をLU分解せよ．pivot操作が必要となる．P.A = L.Uを確かめよ．