```python
# coding: utf-8

import numpy as np

eps = .0000001
n = 5
h = 1./n

A = np.array([[((i*h)**2 + 1) / h**2 - i / 2. for i in range(n+1)] for k in range(2*n+1)])
B = np.array([[((k*h)**2 + 1) / h**2 + k / 2. for i in range(n+1)] for k in range(2*n+1)])
D = np.array([[((i*h)**2 + 1) / h**2 + i / 2. for i in range(n+1)] for k in range(2*n+1)])
C = np.array([[((k*h)**2 + 1) / h**2 - k / 2. for i in range(n+1)] for k in range(2*n+1)])
E = np.array([[2 / h**2 * ((i*h)**2 + (k*h)**2 + 2) for i in range(n+1)] for k in range(2*n+1)])
F = np.array([[(k*h * (i*h - 1))**2 for i in range(n+1)] for k in range(2*n+1)])

A[0] = [((i*h)**2 + 1) / (2*h) - i*h / 4 for i in range(n+1)]
D[0] = [((i*h)**2 + 1) / (2*h) + i*h / 4 for i in range(n+1)]
B[0] = [0 for i in range(n+1)]
C[0] = [1 / h for i in range(n+1)]
E[0] = [((i*h)**2 + 1) / h + 1 for i in range(n+1)]

U_next = np.array([[ -F[i][k] / E[i][k] for k in range(n+1)] for i in range (2*n+1)])
U = np.zeros([2*n+1,n+1])

def classic(ar, ar_next):
    iter = 0
    U = np.array(ar) ; U_next = np.array(ar_next)
    while (np.max(abs(U_next - U)) > eps):
        U = [[U_next[i][k] for k in range(n+1)] for i in range(2*n+1)]
        for i in range(n+1):
            U_next[i][0] = 0
            for k in range(1,n):
                U_next[i][k] = (A[i][k] * U[i-1][k] + B[i][k] * U[i][k-1] + C[i][k] * U[i][k+1] +
                D[i][k] * U[i+1][k] - F[i][k]) / E[i][k]
            U_next[i][n] = 1
        for i in range(n+1,2*n+1):
            for k in range(i-n):
                U_next[i][k] = 0
            U_next[i][i-n] = (i-n)*h
            for k in range(i-n+1,n):
                U_next[i][k] = (A[i][k] * U[i-1][k] + B[i][k] * U[i][k-1] + C[i][k] * U[i][k+1] +
                D[i][k] * U[i+1][k] - F[i][k]) / E[i][k]
            U_next[i][n] = 1
        iter += 1
    print 'Итераций:', iter
    print np.flipud(U_next)

def Zeydel(ar, ar_next):
    iter = 0
    U = np.array(ar) ; U_next = np.array(ar_next)
    while (np.max(abs(U_next - U)) > eps):
        U = [[U_next[i][k] for k in range(n+1)] for i in range(2*n+1)]
        for i in range(n+1):
            U_next[i][0] = 0
            for k in range(1,n):
                U_next[i][k] = (A[i][k] * U_next[i-1][k] + B[i][k] * U_next[i][k-1] + C[i][k] * U[i]
                [k+1] + D[i][k] * U[i+1][k] - F[i][k]) / E[i][k]
            U_next[i][n] = 1
        for i in range(n+1,2*n+1):
            for k in range(i-n):
                U_next[i][k] = 0
            U_next[i][i-n] = (i-n)*h
            for k in range(i-n+1,n):
                U_next[i][k] = (A[i][k] * U_next[i-1][k] + B[i][k] * U_next[i][k-1] + C[i][k] * U[i]
                [k+1] + D[i][k] * U[i+1][k] - F[i][k]) / E[i][k]
            U_next[i][n] = 1
        iter += 1
    print 'Итераций:', iter
    print np.flipud(U_next)

def upper_relax(ar, ar_next, omega, it=False):
    iter = 0
    U = np.array(ar) ; U_next = np.array(ar_next)
    while (np.max(abs(U_next - U)) > eps):
```

```
71          U = [[U_next[i][k] for k in range(n+1)] for i in range(2*n+1)]
72          for i in range(n+1):
73              U_next[i][0] = 0
74              for k in range(1,n):
75                  tmp = (A[i][k] * U_next[i-1][k] + B[i][k] * U_next[i][k-1] + C[i][k] * U[i][k+1] +
                    D[i][k] * U[i+1][k] - F[i][k]) / E[i][k]
76                  U_next[i][k] = U[i][k] + omega * (tmp - U[i][k])
77              U_next[i][n] = 1
78          for i in range(n+1,2*n+1):
79              for k in range(i-n):
80                  U_next[i][k] = 0
81              U_next[i][i-n] = (i-n)*h
82              for k in range(i-n+1,n):
83                  tmp = (A[i][k] * U_next[i-1][k] + B[i][k] * U_next[i][k-1] + C[i][k] * U[i][k+1] +
                    D[i][k] * U[i+1][k] - F[i][k]) / E[i][k]
84                  U_next[i][k] = U[i][k] + omega * (tmp - U[i][k])
85              U_next[i][n] = 1
86          iter += 1
87      if it:
88          return iter
89      else:
90          print 'Итераций:', iter
91          print np.flipud(U_next)
92
93  def omega_search():
94      omega0 = .95; omega = 1.
95      while (omega <= 2):
96          omega0 = omega
97          omega += .01
98          if (upper_relax(U, U_next, omega, True) - upper_relax(U, U_next, omega0, True) > 0):
99              return omega0
100     return omega
101
102 print 'Метод простых итераций:'
103 classic(U, U_next)
104 print
105 print 'Метод Зейделя:'
106 Zeydel(U, U_next)
107 print
108 print 'Метод верхней релаксации:'
109 omega = omega_search()
110 print omega
111 upper_relax(U, U_next, omega)
```

```
Метод простых итераций:
Итераций: 116
[[ 0.          0.          0.          0.          0.          1.        ]
 [ 0.          0.          0.          0.          0.8         1.        ]
 [ 0.          0.          0.          0.6         0.7906219   1.        ]
 [ 0.          0.          0.4         0.58091508  0.78113064  1.        ]
 [ 0.          0.2         0.37444771  0.56681684  0.77489199  1.        ]
 [ 0.          0.17632812  0.3658433   0.56599163  0.77617108  1.        ]
 [ 0.          0.18683601  0.38528522  0.58614456  0.78892881  1.        ]
 [ 0.          0.22536507  0.43933748  0.63372001  0.81616293  1.        ]
 [ 0.          0.31547408  0.5475433   0.71734211  0.85943195  1.        ]
 [ 0.          0.52945756  0.74438221  0.84379265  0.91568224  1.        ]
 [ 0.          1.08813255  1.068672    1.00676466  0.97146017  1.        ]]

Метод Зейделя:
Итераций: 65
[[ 0.          0.          0.          0.          0.          1.        ]
 [ 0.          0.          0.          0.          0.8         1.        ]
 [ 0.          0.          0.          0.6         0.79062191  1.        ]
 [ 0.          0.          0.4         0.5809151   0.78113066  1.        ]
 [ 0.          0.2         0.37444776  0.56681692  0.77489204  1.        ]
 [ 0.          0.17632819  0.36584344  0.56599177  0.77617118  1.        ]
 [ 0.          0.18683613  0.38528544  0.58614481  0.78892896  1.        ]
 [ 0.          0.22536524  0.43933778  0.63372031  0.81616314  1.        ]
 [ 0.          0.31547431  0.54754364  0.71734247  0.85943217  1.        ]
 [ 0.          0.52945783  0.7443826   0.84379296  0.91568243  1.        ]
 [ 0.          1.08813299  1.06867234  1.00676489  0.97146026  1.        ]]

Метод верхней релаксации:
1.36
Итераций: 25
[[ 0.          0.          0.          0.          0.          1.        ]
 [ 0.          0.          0.          0.          0.8         1.        ]
 [ 0.          0.          0.          0.6         0.79062191  1.        ]
 [ 0.          0.          0.4         0.5809151   0.78113066  1.        ]
 [ 0.          0.2         0.37444777  0.56681693  0.77489205  1.        ]
 [ 0.          0.17632821  0.36584347  0.56599181  0.7761712   1.        ]
 [ 0.          0.18683617  0.3852855   0.58614486  0.788929    1.        ]
 [ 0.          0.22536531  0.43933788  0.63372041  0.81616319  1.        ]
 [ 0.          0.31547442  0.54754379  0.71734259  0.85943223  1.        ]
 [ 0.          0.52945802  0.74438278  0.8437931   0.9156825   1.        ]
 [ 0.          1.08813332  1.06867254  1.00676499  0.9714603   1.        ]]
```