Autor: Paweł Salwa

Użyłem funkcji timeit() do obliczenia czasu wykonania każdej z zaimplementowanych funkcji (musialem w niej importowac caly plik jeszcze raz dlatego jest duplikat w outpucie).
Po porównaniu wyników okazuje się, że najszybciej wykonała się metoda Gaussa- Seidela.

Output:

```
start
start
X[0] = 0.9999999997671694
X[1] = 2.0
X[2] = 0.9999999997671694
czas relaksacyjnej: 0.00014394434401765466

X[0] = 0.9999999998835847
X[1] = 1.9999999997671694
X[2] = 0.9999999998835847
czas jacobiego: 0.00011966103920713067

X[0] = 0.9999999999126885
X[1] = 1.9999999999563443
X[2] = 0.9999999999890861
czas gausSeidel: 7.197214290499687e-05

X[0] = 1.000000000059785
X[1] = 1.9999999998749103
X[2] = 0.999999999982343
czas succ OverRelaxation: 0.00011029880261048675

koniec  #tutaj mam zduplikowane funkcje (wywołane przez import w timeit() )
X[0] = 0.9999999997671694
X[1] = 2.0
X[2] = 0.9999999997671694
czas relaksacyjnej: 0.0001170279283542186

X[0] = 0.9999999998835847
X[1] = 1.9999999997671694
X[2] = 0.9999999998835847
czas jacobiego: 0.00010239941184408963

X[0] = 0.9999999999126885
X[1] = 1.9999999999563443
X[2] = 0.9999999999890861
czas gausSeidel: 6.699847290292382e-05

X[0] = 1.000000000059785
X[1] = 1.9999999998749103
X[2] = 0.999999999982343
czas succ OverRelaxation: 8.660065941512585e-05

koniec
```

Kod w pythonie:

```
# -*- coding: utf-8 -*-
"""
Created on Sat Feb 04 20:34:45 2017

@author: salwus
"""
#=================================================================
def relaksacyjna(M, b, gamma):
    N=len(M)
    D=[]
    X=[]

    for i in range(0,N):
        X.append(1.)
        D.append(1.)

    iterator=0

    end=False

    while(not end):
        end=True
        iterator +=1

        for i in range(0,N):
            X[i] = b[i]
            for j in range(0,N):
                X[i] -= M[i][j]*D[j]

            X[i] = D[i] + (gamma*X[i])

        for i in range(0,N):
            if( abs(D[i] - X[i]) > 0.000000001):
                end=False # tutaj sprawdza czy dokladnosc wyniku jest wystarczajaca
            D[i] = X[i]

    for i in range(0,N):
        print 'X[' + repr(i) + '] = ' + repr(X[i])
#=====================================================
def jacobiego(M,b):

    T = [[ 0., 0., 0.],
        [ 0., 0., 0.],
       [ 0., 0., 0.]]
    D=[]
    C=[]
    X=[]

    N=len(M)

    for i in range(0,N):
        D.append(0.)
        C.append(0.)
        X.append(0.)

    for i in range(0,N):
        D[i] = 1/M[i][i]
        C[i] = D[i]*b[i]
        D[i] *= -1


    for i in range(0,N):
        for j in range(0,N):
            if(j != i):
                T[i][j] = M[i][j] * D[i]
        D[i]=0.

    end = False

    while(not end):
        end = True

        for i in range(0,N):
            X[i] = C[i]
            for j in range(0,N):
                if(j != i):
                    X[i] += T[i][j]*D[j]
        for i in range(0,N):
            if ( abs(D[i] - X[i]) > 0.000000001):
                end = False
            D[i]=X[i]

    for i in range(0,N):
        print 'X[' + repr(i) + '] = ' + repr(X[i])
#====================================================
def gaussSeidel(M,b):

    T = [[ 0., 0., 0.],
        [ 0., 0., 0.],
     [ 0., 0., 0.]]

    D=[]
    C=[]
    X=[]
    N=len(M)

    for i in range(0,N):
        D.append(0.)
        C.append(0.)
        X.append(0.)

    for i in range(0,N):
        D[i] = 1/M[i][i]
        C[i] = D[i]*b[i]
        D[i] *= (-1)

    for i in range(0,N):
        for j in range(0,N):
            if(j != i):
                T[i][j] = M[i][j]*D[i]
        D[i] = 0

    end=False

    while(not end):
        end = True

        for i in range(0,N):
            X[i] = C[i]
            for j in range(0,N):
                if(j != i):
                    if(i > j):
                        X[i] += T[i][j]*X[j]

                    elif(i < j):
                        X[i] += T[i][j]*D[j]

        for i in range(0,N):
            if ( abs(D[i] - X[i]) > 0.000000001):
                end = False
            D[i]=X[i]

    for i in range(0,N):
        print 'X[' + repr(i) + '] = ' + repr(X[i])
#==============================================================
def sor(M,b,omega):

    D=[]
    X=[]

    N=len(M)

    for i in range(0,N):
        D.append(0.)
        X.append(0.)
    end=False

    while(not end):
        end = True

        for i in range(0,N):
            X[i] = b[i]
            for j in range(0,N):
                if(j != i):
                    if(i > j):
                        X[i] -= M[i][j]*X[j]

                    elif(i < j):
                        X[i] -= M[i][j]*D[j]
            X[i] =(1-omega)*D[i]+omega/M[i][i]*X[i]

        for i in range(0,N):
            if ( abs(D[i] - X[i]) > 0.000000001):
                end = False
            D[i]=X[i]

    for i in range(0,N):
        print 'X[' + repr(i) + '] = ' + repr(X[i])
#=====================================================


print 'start'

A = [[ 4.,-1., 0.],
    [-1., 4.,-1.],
    [ 0.,-1., 4.]]

B = [2.,
    6.,
    2.]

import timeit
#duplikuje mi tutaj output- nie wiem czemu-dlatego usunalem zduplikowana czesc w opisie dla przejzystosci
t = timeit.Timer(stmt="Salwa7.relaksacyjna(Salwa7.A, Salwa7.B,0.25)", setup="import Salwa7")
print "czas relaksacyjnej: " + repr(t.timeit(1))+ '\n\n'
del t
t = timeit.Timer(stmt="Salwa7.jacobiego(Salwa7.A, Salwa7.B)", setup="import Salwa7")
print "czas jacobiego: " + repr(t.timeit(1)) + '\n\n'
del t
t = timeit.Timer(stmt="Salwa7.gaussSeidel(Salwa7.A, Salwa7.B)", setup="import Salwa7")
print "czas gausSeidel: " + repr(t.timeit(1))+ '\n\n'
del t
t = timeit.Timer(stmt="Salwa7.sor(Salwa7.A, Salwa7.B, 1.25)", setup="import Salwa7")
print "czas succ OverRelaxation: " + repr(t.timeit(1))+ '\n\n'
del t
print 'koniec'
```

```
#zwykle wywolanie
#print 'relaksacyjna:'
#relaksacyjna(A,B, 0.25)
#print 'jacobi:'
#jacobiego(A,B)
#print 'gausSeidel:'
#gaussSeidel(A,B)
#print 'succ OverRelaxation:'
#sor(A,B,1.25)
```