Autor: Paweł Salwa

Niezaimplementowana funkcja z metodą Rayleigha.
Potęgowa() okazała się szybsza od qr() dla danej w zadaniu macierzy (patrząc na czas trwania funkcji)

Output:
wynik potegowej:
L[0] = 8.548512847334699
L[1] = -4.574087224171323
L[2] = 0.025574372634318318
0.000160620809652 <-czas wykonania potegowa()

wynik qr:
L[0] = 8.548512851978701
L[1] = -4.5740872246130255
L[2] = 0.02557437263431808
0.00112346795834 <-czas wykonania qr()

Kod w pythonie (Salwa8.py):

```
# -*- coding: utf-8 -*-
"""
Created on Sun Feb 05 14:01:32 2017

@author: salwus
"""
from potegowa import potegowa

from qr import qr

import timeit

print "wynik potegowej:"
print timeit.timeit( potegowa, number=1 )  ,"<-czas wykonania potegowa()"
print "\nwynik qr:"
print timeit.timeit( qr, number=1 ) ,"<-czas wykonania qr() "
```

Kod w pythonie (potegowa.py):

```
# -*- coding: utf-8 -*-
"""
Created on Sun Feb 05 14:56:18 2017

@author: salwus
"""

from math import sqrt
```

```
def potegowa():
	M = [[1.,2.,3.],
	    [2.,4.,5.],
	    [3.,5.,-1.]]

	N=len(M)
	X=[]
	temp2=[]
	X_prev=[]
	Z=[]
	Lambda = 0.
	Evectors=[[0.,0.,0.],[0.,0.,0.],[.0,0.,0.]]
	Eigenvalue= []

	for i in range(0,N):
		X.append(0.)
		temp2.append(0.)
		X_prev.append(1.)
		Z.append(0.)
		Eigenvalue.append(0.)
#		for j in range(0,N):pass
#			Evectors[i][j].append(0.)

	for L in range(0,N):
		while(True):
			for i in range(0,N):
				X[i] =0
				for j in range(0,N):
					X[i] += M[i][j]*X_prev[j]


			for k in range(0,L): # deflacja
				temp = 0
				for i in range(0,N):
					temp += Evectors[i][k]*X[i]
					temp2[i] = Evectors[i][k]
				for i in range(0,N):
					temp2[i] *= temp
				for i in range(0,N):
				  X[i] -= temp2[i]

			for i in range(0,N):
			  Z[i]= X[i]

			X_norm = 0
			for i in range(0,N):
				X_norm += X[i]*X[i]
			X_norm = sqrt(X_norm)

			for i in range(0,N):
				X[i] /= X_norm
```

```
			if ( abs( Lambda - X_norm ) < 0.00000001):
				break
			for i in range(0,N):
				X_prev[i] = X[i]
			Lambda = X_norm

		X_norm = 0
		for i in range(0,N):
			X_norm += Z[i]*X_prev[i]
		Eigenvalue[L] = X_norm

		for i in range(0,N):
			Evectors[i][L] = X[i]
		for i in range(0,N):
			X_prev[i] = 1



	for i in range(0,N):
		print 'L[' + repr(i) + '] = ' + repr(Eigenvalue[i])
```

Kod w pythonie (qr.py):

```
# -*- coding: utf-8 -*-
"""
Created on Sun Feb 05 14:58:19 2017

@author: salwus
"""
from math import sqrt
from copy import copy
def qr():
	M = [[1.,2.,3.],
	    [2.,4.,5.],
	    [3.,5.,-1.]]
	Q = [[0.,0.,0.], #nieelastyczne, ale proste
	    [0.,0.,0.],
	    [0.,0.,0.]]
	R = [[0.,0.,0.],
	    [0.,0.,0.],
	    [0.,0.,0.]]

	N=len(M)

	Lambda=[]
	Prev=[]
	for i in range(0,N):
		Lambda.append(0.)
		Prev.append(1.)
```

```
	M,Q,R=qrTMP(M,Q,R)
	end = True

	while(end):
		end = False
		for i in range(0,N):
			for j in range(0,N):
				M[i][j] = 0
				for k in range(0,N):
					M[i][j] += R[i][k]*Q[k][j]

		for i in range(0,N):
			Lambda[i] = M[i][i]

		for i in range(0,N):
			if(abs(Prev[i] - Lambda[i]) > 0.00000001):
				end = True

		for i in range(0,N):
			Prev[i] = Lambda[i]
		M,Q,R=qrTMP(M,Q,R)

	for i in range(0,N):
		print 'L[' + repr(i) + '] = ' + repr(Lambda[i])
```

```
#=====================================================
def qrTMP(M,Q,R):
	N=len(M)
	a=[]
	M_vec=[]

	for i in range(0,N):
		a.append(M[i][0])
		M_vec.append(0.)
	for i in range(0,N):#jendostkowa
		for j in range(0,N):
			if(j==i):
				Q[i][j]=1.
			else:
				Q[i][j]=0.
#========================
	for ite in range(0,N-1):
		norm =0
```

```
        for i in range(ite,N):
            norm += a[i]*a[i]
        norm = sqrt(norm)
        a[ite] -= norm
        norm =0
        for i in range(ite,N):
            norm += a[i]*a[i]
        norm = sqrt(norm)
        for i in range(ite,N):
            a[i] /= norm
        for i in range(0,N):#jendostkowa
            for j in range(0,N):
                if(j==i):
                    R[i][j]=1
                else:
                    R[i][j]=0
        for i in range(ite,N):
            for j in range(ite,N):
                R[i][j]-= 2*a[i]*a[j]
        for i in range(ite,N):
            a[i] = 0
            for j in range(0,N):
                a[i] += M[j][ite+1]*R[i][j]

        for i in range(0,N):
            for j in range(0,N):
                for k in range(0,N):
                    if (j==0):
                        M_vec[k] = Q[i][k]
                    if (k==0):
                        Q[i][j] = M_vec[k]*R[k][j]
                    else:
                        Q[i][j] += M_vec[k]*R[k][j]
#========================

    Q = transp(Q)

    for i in range(0,N):
            for j in range(0,N):
                R[i][j] = 0
                for k in range(0,N):
                    R[i][j] += Q[i][k]*M[k][j]
    Q = transp(Q)
    return M,Q,R
#====================================================


def transp(X):
    Y = [[0.,0.,0.],
         [0.,0.,0.],
         [0.,0.,0.]]
    N = len(X)
```

```
for i in range(0,N):
	for j in range(0,N):
		Y[i][j] = X[i][j]


for i in range(0,N):
	for j in range(0,N):
		Y[i][j] = X[j][i]
return Y
```