

ДОДАТОК А

Тексти програм

```
# -*- coding: utf-8 -*-
"""
Created on Sat Dec 7 23:56:07 2024

@author: Admin
"""
import numpy as np
import networkx as nx
from networkx.utils import py_random_state
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy.special import gamma, gammaln

def _random_subset(seq, m, rng):
    """Return m unique elements from seq.
    This differs from random.sample which can return repeated
    elements if seq holds repeated elements.
    Note: rng is a random.Random or numpy.random.RandomState
    instance.
    """
    targets = set()
    while len(targets) < m:
        x = rng.choice(seq)
        targets.add(x)
    return targets

@py_random_state("seed")
def homol(n, m0=2, lam=1.4, beta=0, seed=None):
    G = nx.complete_graph(m0)
    m = nx.number_of_edges(G)
    nx.set_node_attributes(G, 0, 'color')
    # List of existing nodes, with nodes repeated once for each
    adjacent edge
    repeated_nodes = [[n for n, d in G.degree() for _ in range(d)]]
    flattened_list = repeated_nodes[0].copy()
    colors = 1
    # Start adding the other n - m0 nodes.
    source = len(G)
    ncol = [source]
    gam = gamma(lam)
    while source < n:
        dEdges = source ** (lam-1) / gam # m * (source - 1 + lam) /
source
        m = int(dEdges)
        m += seed.random() < (dEdges - m)
        if seed.random() < np.log(source-m0+2) ** -3:
            G.add_node(source, color=colors)
            # Choose m unique nodes from the flattened list of nodes
            targets = _random_subset(flattened_list, m, seed)
            for k in targets:
                c = G.nodes[k]['color']
                repeated_nodes[c].append(k)
```

```

        repeated_nodes.extend([[source] * m])
        ncol.append(1)
        colors += 1
    else:
        # col = seed.choice(range(colors))
        # col = seed.choice(range(colors), weights =
np.array(ncol)/sum(ncol))
        # col = np.random.choice(range(colors), p =
np.array(ncol)/sum(ncol))
        k = seed.choice(flattened_list)
        col = G.nodes[k]['color']
        G.add_node(source, color=col)
        # Choose m unique nodes of the colors=col
        if ncol[col] < m:
            targets = set(repeated_nodes[col]) | \
                _random_subset(flattened_list, m - ncol[col],
seed)
            for k in targets:
                c = G.nodes[k]['color']
                repeated_nodes[c].append(k)
        else:
            targets = _random_subset(repeated_nodes[col], m,
seed)
            repeated_nodes[col].extend(targets)
            repeated_nodes[col].extend([source] * m)
            ncol[col] += 1

        G.add_edges_from(zip([source] * m, targets))
        flattened_list.extend(targets)
        flattened_list.extend([source] * m)
        source += 1
    node_by_colors = [set(x) for x in repeated_nodes]
    return G, node_by_colors

@py_random_state("seed")
def homo2(n, m, p, seed=None, initial_graph=None, *,
create_using=None):
    G = nx.complete_graph(m+1, create_using)
    nx.set_node_attributes(G, 0, 'color')
    # List of existing nodes, with nodes repeated once for each
adjacent edge
    repeated_nodes = [[n for n, d in G.degree() for _ in range(d)]]
    flattened_list = repeated_nodes[0].copy()
    colors = 1
    # Start adding the other n - m0 nodes.
    source = len(G)
    ncol = [source]
    while source < n:
        if seed.random() < np.log(source-m+2) ** -2.5:
            G.add_node(source, color=colors)
            # Choose m unique nodes from the flattened list of nodes
            targets = _random_subset(flattened_list, m, seed)
            for k in targets:
                c = G.nodes[k]['color']
                repeated_nodes[c].append(k)
            repeated_nodes.extend([[source] * m])
            ncol.append(1)

```

```

        colors += 1
    else:
        col = seed.choice(range(colors))
        # col = seed.choice(range(colors), weights =
np.array(ncol)/sum(ncol))
        # col = np.random.choice(range(colors), p =
np.array(ncol)/sum(ncol))
        k = seed.choice(flattened_list)
        # col = G.nodes[k]['color']
        G.add_node(source, color=col)
        # Choose m unique nodes of the colors=col
        if ncol[col] < m:
            targets = set(repeated_nodes[col]) | \
                _random_subset(flattened_list, m - ncol[col],
seed)
            for k in targets:
                c = G.nodes[k]['color']
                repeated_nodes[c].append(k)
        else:
            targets = _random_subset(repeated_nodes[col], m,
seed)
            repeated_nodes[col].extend(targets)
            repeated_nodes[col].extend([source] * m)
            ncol[col] += 1

        G.add_edges_from(zip([source] * m, targets))
        flattened_list.extend(targets)
        flattened_list.extend([source] * m)
        source += 1
    node_by_colors = [set(x) for x in repeated_nodes]
    return G, node_by_colors

@py_random_state("seed")
def homo3(n, m0=2, lam=1.4, beta=0, seed=None):
    G = nx.complete_graph(m0)
    m = nx.number_of_edges(G)
    nx.set_node_attributes(G, 0, 'color')
    # List of existing nodes, with nodes repeated once for each
adjacent edge
    repeated_nodes = [[n for n, d in G.degree() for _ in range(d)]]
    flattened_list = repeated_nodes[0].copy()
    colors = 1
    # Start adding the other n - m0 nodes.
    source = len(G)
    ncol = [source]
    gam = gamma(lam)
    while source < n:
        dEdges = source ** (lam-1) / gam if lam > 1 else m0
        m = int(dEdges)
        m += seed.random() < (dEdges - m)
        m_in = int(beta * dEdges + 0.5)
        m_new = m - m_in
        #if seed.random() < 1500*np.exp(-source-m0+2): #500*(source-
m0+2)**-1.5:
            #if source % 100 == 0 : #in
[5,10,20,40,80,160,320,640,1280,2560,5120] :
            #if seed.random() < colors / source / 3 :

```

```

if seed.random() < colors ** -3 :
    G.add_node(source, color=colors)
    # Choose m unique nodes from the flattened list of nodes
    targets = _random_subset(flattened_list, m, seed)
    for k in targets:
        c = G.nodes[k]['color']
        repeated_nodes[c].append(k)
    repeated_nodes.extend([[source] * m])
    ncol.append(1)
    colors += 1
    m_new = m
    m_in = 0
    tar1 = set(list(targets))
    tar2 = {}
else:
    #col = seed.choice(range(colors))
    k = seed.choice(flattened_list)
    col = G.nodes[k]['color']
    G.add_node(source, color=col)
    # Choose m unique nodes of the colors=col
    if ncol[col] < m + m_in:
        targets = set(repeated_nodes[col]) | \
            _random_subset(flattened_list,
                           m + m_in - ncol[col], seed)
        for k in targets:
            c = G.nodes[k]['color']
            repeated_nodes[c].append(k)
        tar1 = set(list(targets)[:m_new])
        tar2 = set(list(targets)[m_new:])
    else:
        tar1 = _random_subset(repeated_nodes[col], m_new,
seed)
        tar2 = _random_subset(repeated_nodes[col], 2 * m_in,
seed)

        repeated_nodes[col].extend(tar1)
        repeated_nodes[col].extend(tar2)
        repeated_nodes[col].extend([source] * m_new)
        ncol[col] += 1
    # Adding m edges
    G.add_edges_from(zip(list(tar2)[:m_in], list(tar2)[m_in:]))
    G.add_edges_from(zip([source] * m_new, tar1))
    flattened_list.extend(tar1)
    flattened_list.extend(tar2)
    flattened_list.extend([source] * m_new)
    source += 1
node_by_colors = [set(x) for x in repeated_nodes]
return G, node_by_colors

def func(x, a, b):
    return a * x ** (-b)

def func1(x, a, b):
    return a * np.exp(gammaln(x) - gammaln(x+b))

def func2(x, a, b):
    return a - b * x

# rng = np.random.default_rng(1234)

```

```

# G, node_by_colors = homo1(10000, m0=2, lam=1.4)
# G, node_by_colors = homo2(10000, 5, 0.002)
(lam,beta) = 1.3,0.3
#G, node_by_colors = homo3(10000, m0=2, lam=lam, beta=beta)
(lam,beta) = 1,0
n_b_c = list()
samples = 20
for i in range(samples):
    G, node_by_colors = homo3(1000, m0=5, lam=lam, beta=beta)
    n_b_c.append(node_by_colors)
pw = 1-lam*(1-beta)/2
edges = nx.number_of_edges(G)
deg_color = [(G.degree(node), data['color']) for node, data in
G.nodes.data()]
deg_color = np.array(sorted(deg_color, key=lambda k: k[0],
reverse=True))

xdata = np.arange(1, len(deg_color)+1)
ydata = deg_color[:, 0]
popt, _ = curve_fit(func, xdata, ydata, bounds=(0, [np.inf, 1.]))

plt.figure()
plt.set_cmap('tab10')
ax = plt.axes()
ax.set_xscale('log')
ax.set_yscale('log')
ax.scatter(xdata, ydata, c=deg_color[:, 1])
ax.plot(xdata, func(xdata, *popt), 'r-', label='fit: b=%6.4f' %
popt[1])
ax.legend()
ax.grid()
plt.sca(ax)
plt.show()

#cond3 = np.array([nx.conductance(G, part) for part in
node_by_colors])
#c3 = np.sum(cond3)

ncol = [len(x) for x in node_by_colors]

x1 = np.arange(1, len(ncol)+1)
y1 = np.array(sorted(ncol, reverse=True))
pc1, _ = curve_fit(func, x1, y1) # , bounds=(0, [np.inf, 10.]))
# , bounds=(0, [np.inf, 10.]))
pc2, _ = curve_fit(func2, np.log10(x1), np.log10(y1))
pc2[0] = 10 ** pc2[0]

plt.figure()
plt.set_cmap('tab10')
ax = plt.axes()
ax.set_xscale('log')
ax.set_yscale('log')
ax.scatter(x1, y1, c=range(len(y1)))
# ax.plot(x1, func(x1, *pc1), 'k-', label='fit: b=%6.4f' % pc1[1])
ax.plot(x1, func(x1, *pc2), 'r-', label='fit: b=%6.4f' % pc2[1])
ax.legend()
ax.grid()
plt.sca(ax)

```

```

plt.show()

ncol_arr = [[len(x) for x in nbc] for nbc in n_b_c]
plt.figure()
plt.set_cmap('tab10')
ax = plt.axes()
for ncol in ncol_arr :
    x1 = np.arange(1, len(ncol)+1)
    y1 = np.array(sorted(ncol, reverse=True))
    pc1, _ = curve_fit(func, x1, y1)
    pc2, _ = curve_fit(func2, np.log10(x1), np.log10(y1))
    pc2[0] = 10 ** pc2[0]

    ax.set_xscale('log')
    ax.set_yscale('log')
    ax.scatter(x1, y1, c=range(len(y1)))

ax.plot(x1, func(x1, *pc2), 'r-', label='fit: b=%6.4f' % pc2[1])

#ax.legend()
ax.grid()
plt.sca(ax)
plt.show()

#mod = nx.community.modularity(G, node_by_colors)
#(cov, perf) = nx.community.partition_quality(G, node_by_colors)

```

ДОДАТОК Б

Відомість кваліфікаційної роботи

Позначення					Найменування		Дод. відомості	
					Текстові документи			
1.					Пояснювальна записка		68 с.	