МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

РЕАЛИЗАЦИЯ И АНАЛИЗ МОДЕЛИ БАРАБАШИ—АЛЬБЕРТ ГЕНЕРАЦИИ СЛУЧАЙНОГО ГРАФА

КУРСОВАЯ РАБОТА

студента 2 курса 211 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета КНиИТ
Козырева Юрия Дмитриевича

Научный руководитель
зав. каф., к. ф.-м. н., доцент _____ С. В. Миронов

Заведующий кафедрой
к. ф.-м. н., доцент _____ С. В. Миронов

Саратов 2020

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Наука о графах — одно из приложений к весьма красивой и интересной науке — науке о случайных графах.

В повседневной жизни для решения многих задач часто используются случайные графы. Случайные графы нашли практическое применение во всех областях, где нужно смоделировать сложные сети — известно большое число случайных моделей графов, отражающих разнообразные типы сложных сетей в различных областях. Случайные графы применяются при моделировании и анализе биологических и социальных систем, сетей, а также при решении многих задач класса NP.

Случайные графы впервые определены венгерскими математиками Эрдёшем и Реньи в книге 1959 года «On Random Graphs» [1] и независимо Гильбертом в его статье «Random graphs» [2].

Случайный граф — общий термин для обозначения вероятностного распределения графов [3]. Их можно описать просто распределением вероятности или случайным процессом, создающим эти графы.

Теория сучайных графов находится на стыке комбинаторики, теории графов и теории вероятностей. В основе ее лежит глубокая идея о том, что мощные инструменты современной теории вероятностей должны поспособствовать более верному осознанию природы графа, призваны помочь решению многих комбинаторных и теоретико-графовых задач [4].

С математической точки зрения случайные графы необходимы для ответа на вопрос о свойствах типичных графов.

Модель Барабаши—Альберт является одной из наиболее популярных и хорошо изученных моделей случайных графов.

Целью настоящей работы является изучение процесса построения случайного графа на примере модели Барабаши—Альберт. Для достижения этой цели необходимо решить следующие задачи.

— рассмотреть алгоритм Барабаши—Альберт для построения случайного графа;
— реализовать алгоритм Барабаши—Альберт на некотором языке программирования;
— провести анализ закона распределения степеней вершин графа, построенного по алгоритму Барабаши—Альберт.

1 Модели построения случайного графа

Существуют различные модели построения случайных графов.

1.1 Модель Эрдеша—Ренье

Модель Ердеша—Ренье является одной из первых моделей случайного графа. Граф построенный по этой модели представляет собой совокупность множества вершин $V = \{1, ..., n\}$ и множества рёбер E, состоящего из рёбер полного графа K_n построенного на множестве V, выбранных по схеме Бернулли. Таким образом образуется случайный граф $G = (V, E)$. Формально выражаясь, мы имеем вероятностное пространство

$$G(n, p) = (\Omega_n, F_n, P_{n,p}),$$

в котором: n — количество вершин, p — вероятность появления новго ребра, F_n — сигма-множество,$|\Omega_n| = 2^N$ — множество возможных рёбер, $P_{n,p}(G) = p^{|E|}q^{\binom{n}{2}} - |E|$ — вероятностная мера. Таким образом, в модели Эрдеша—Реньи каждое ребро независимо от других ребер входит в случайный граф с вероятностью p. Модель Эрдеша—Реньи на данный момент является самой изученной моделью случайных графов [5].

1.2 Модель Барабаши—Альберт

Модель Барабаши—Альберт является одной из первых моделей веб-графов. Веб-граф представляет собой ориентированный мульти-граф, вершинами в котором являются какие-либо конкретные структурные единицы в Интернете: речь может идти о страницах, сайтах, хостах, владельцах и пр. Для определенности будем считать, что вершинами веб-графа служат именно сайты. А рёбрами соединяются вершины, между которыми имеются ссылки.

Также Барабаши и Альберт была предложена модель предпочтительного присоединения [6], основная идея которой заключается в том, что при присоединении к графу новой вершины проводится некоторое количество рёбер от добавленной вершины к уже существующим, при этом вероятность появления ребра межу новой вершиной и какой-то конкретной вершиной пропорциональна степени данной вершины. Однако в своих работах Барабаши и Альберт никак не конкретизировали, какую именно из этих моделей они предлагают рассматривать.

1.3 Модель Боллобаша—Риодана

Одной из наиболее удачных и часто используемых моделей предпочтительного присоединения является модель Боллобаша—Риодана. Существуют две основных и, по сути, совпадающих модификации этой модели. В одной дается динамическое, а в другой статическое описание случайности [7].

1.3.1 Динамическая модификация

В данной модификации при добавлении n-ной вершины проводятся n новых рёбер, при этом рёбра могут быть кратными, а также петлями и даже кратными рёбрами, при создании графа с единственной вершиной проводится петля в этой точке [5]. Таким образом вероятность появления ребра $(n,\ i),\ \ i \in [0,\ n-1]$ равна $\frac{\deg\ i}{2n-1}$, где $degi$ — количество уже проведенных рёбер из вершины n в вершину i. Очевидно, что распределение вероятностей задано корректно, поскольку

$$\sum_{i=1}^{n-1} \frac{deg\ i}{2n-1} + \frac{1}{2n-1} = \frac{2n-2}{2n-1} + \frac{1}{2n-1} = 1.$$

1.3.2 Статическая модификация, или LCD-модель

Данная модель основывается на объекте называемом линейной хордовой диаграммой (LCD). Для построения данного объекта требуется зафиксировать на оси абсцисс $2n$ точек $1, \ldots, 2n$, разбить их на пары и соединить элементы каждой пары дугой, лежащей в верхней полуплоскости. Количество различных диаграмм равно

$$l_n = \frac{(2n)!}{2^n n!}.$$

По каждой диаграмме строится граф с n вершинами и n ребрами по следующему алгоритму:

— Идти слева направо по оси абсцисс пока не встретится правый конец какой-либо дуги, пусть позиция этой точки равна i_k

— Последовательность $i_{k-1}+1, i_k$ объявляется списком смежности для k-той вершины, $i_0 = 0$

— Если $k < n$, k увеличивается на 1, переход на шаг (1).

При построении модели LCD случайно выбирается одна из возможных LCD и вероятность каждой диаграммы равной $\frac{1}{l_n}$, где l_n – общее число диаграмм. Графы построенные по такой модели имеют те же свойства, что и

графы построенные по динамической модификации схемы Боллобаша—Риодана. Модель Чунг-Лу Пусть нам задано некоторое конечное множество вершин $V = v_1, \ldots, v_n$ и степень каждой вершины d_i, $i = \overline{1, n}$. Генерация графа $G = (V, E)$ происходит следующим образом:

- Формируем множество L, состоящее из $i \cdot d$ копий $i \cdot v$ для каждого i от 1 до n.
- Задаем случайные паросочетания на множестве L.
- Для вершин u и v из Vs количество ребер в графе G , соединяющее их, равно числу паросочетаний между копиями u и v в L [8].

Сгенерированный таким образом граф соответствует степенной модели $P(a, b)$, описывающей графы, для которых:

$$|\{v| \deg v = \ x\}| = \frac{e^{\alpha}}{e^{\beta}}.$$

2 Реализация модели Барабаши—Альберт

В ходе выполнения курсовой работы была реализована модель Барабаши—Альберт. Все расчёты производились на компьютере с процессором Intel core i5-8265U и 16 ГБ оперативной памяти. Модель реализована на Python 3.9.1 с помощью библиотек networkx [9], random и numpy.random.

Реализованная модель представляет собой модель растущего случайного графа предпочтительного связывания. Были реализованы две модификации: в одной из них на каждом шагу добавляется фиксированное количество m рёбер, а во второй, количество новых рёбер на каждой итерации определяется распределением Пуассона с параметром m. В эксперементах в качестве параметра m подставляются числа из множества $\{1, 2, 5, 7, 10\}$, каждый вариант графа строится на 10000 вершин. Так как получаемые графы случайны, то в ходе экперемента каждый граф сроится десять раз, и строятся графики зависимости количества вершин от степени по средним значениям.

2.1 Реализация стандартной модели Барабаши—Альберт

Реализация стандартной модели Барабаши—Альберт состоит в следующем. Сначала создаётся полный граф из m вершин с помощью команды `nx.complete_graph(m)` . Затем в графе создаются $n - m$ вершин, но рёбра ещё не проводятся. Далее создаются и инициализируются вспомогательные массивы `nodes` , `used` и `degrees` , хранящие список присоединённых к графу вершин, информацию о том использованы они или нет и степени вершин, соответственно. Затем в цикле добавляются ребра, как представлено в следующем коде.

```
    for i in range(m, n):
        if not o :
            conections = []
            j = 0
            while j < m:
                choice = random.choices(nodes, weights = degrees, k = 1)
                choosen = choice[0]
                if not used[choosen]:
                    G.add_edge(i, choosen)
                    j += 1
                    conections.append(choosen)
                    used[choosen] = True
```

Здесь описан цикл по неприсоединённым вершинам (см. стр. 1). Для каждой вершины, с помощью функции `random.choices()`, выбираются m различных вершин с которыми будет соединена новая вершина (см. стр. 6). Для того чтобы не было кратных ребер создаётся массив `connections`, хранящий все выбранные вершины (см. стр. 3, 11).

Следующим шагом с помощью массива `connections` исправляются массивы `used` и `degrees`. И новая вершина добавляется в массив `nodes`.

```
for j in range(m):
  used[conections[j]] = False
  degrees[conections[j] - 1] += 1
  ...
nodeCount += 1
  nodes.append(nodeCount)
  degrees.append(m)
```

Случай добавления первой вершины рассматривается отдельно: если флаг `o` поднят, добавляется грань между нулевым и первым узлами.

```
        else:
            G.add_edge(0, 1)
            o = False
        nodeCount += 1
        nodes.append(nodeCount)
        degrees.append(m)
    return G
```

Далее по такому же алгоритму присоединяются другие вершины. Таким образом, получается алгоритм построения модели сучайного графа:

```
    def my_bag(n, m):
      G = nx.complete_graph(m)
      for i in range(m, n):
          G.add_node(i)
      nodeCount = m
      o = True
      nodes = []
      degrees = []
```

```
        used = []
        for j in range(n):
            used.append(False)
        for i in range(m):
            nodes.append(i)
            degrees.append(m)
        for i in range(m, n):
            if not o :
                conections = []
                j = 0
                while j < m:
                    choice = random.choices(nodes, weights = degrees, k = 1)
                    choosen = choice[0]
                    if not used[choosen]:
                        G.add_edge(i, choosen)
                        j += 1
                        conections.append(choosen)
                        used[choosen] = True
                for j in range(m):
                    used[conections[j]] = False
                    degrees[conections[j] - 1] += 1
            else:
                G.add_edge(0, 1)
                o = False
            nodeCount += 1
            nodes.append(nodeCount)
            degrees.append(m)
        return G
```

2.2 Реализация модели Барабаши—Альберт со случайным числом добавляемых ребер на каждой итерации

Основное отличие второй модификации от стандартной модели заключается в том, что на каждом шаге значение m выбирается заново по распределению Пуассона, с помощью функции `numpy.random.poisson()` . Остальные манипуляции производятся аналогично:

```
  def my_bag_poisson(n, m):
    m0 = numpy.random.poisson(m)
        G = nx.complete_graph(m0)
```

```
        for i in range(m0, n):
            G.add_node(i)
        nodeCount = m0
        o = True
        nodes = []
        degrees = []
        used = []
        for j in range(n):
          used.append(False)
        for i in range(m0):
            nodes.append(i)
            degrees.append(m0)
        mi = numpy.random.poisson(m0, n)
        for i in range(m0, n):
            if not o :
                conections = []
                j = 0
                while j < min(mi[i], nodeCount):
                    choice = random.choices(nodes, weights = degrees, k = 1)
                    choosen = choice[0]
                    if not used[choosen]:
                        G.add_edge(i, choosen)
                        j += 1
                        conections.append(choosen)
                        used[choosen] = True
                for j in range(min(mi[i], nodeCount)):
                    used[conections[j]] = False
                    degrees[conections[j] - 1] += 1
            else:
                G.add_edge(0, 1)
                o = False
            nodeCount += 1
            nodes.append(nodeCount)
            degrees.append(m)
        return G
```

2.3 Вывод и представление данных для анализа

Данная реализация содержит фрагмент кода, отвечающий за отображение данных о построенном графе для дальнейшего анализа. Для этого используются библиотеки matplotlib, matplotlib.pyplot [10] и pylab. На первом шаге создаются

десять графов для анализа, с помощью одной из двух описанных подпрограмм, а также создаются массивы `c` и `x` по которым будет строиться график зависимости коллчества вершин от степени.

```
    x = []
    c = []
    for i in range(n):
        c.append(0)
        x.append(i)
```

Данные о количестве вершин каждой степени записываются в массив `c`. Затем каждый элемент этого массива делится на 10, для того чтобы найти среднее значение по десяти графам.

```
    for i in range(10):
        print(str(i) + ',', end = ' ')
        bag = my_bag(n = p, m = m[j])
        for k in bag.adjacency():
            c[len(k[1])]+=1
    for i in range(n):
        c[i]/=10
```

Затем, при помощи команды `pylab.loglog()` строится график по полученным данным, массивы `c` и `x` сохраняются в текстовый файл.

```
    fname = "F:\\CSW\\results\\data_s_" + str(m[j]) + ".txt"
    f = open(fname, "w")
    f.write("x\ty\n")
    for i in range(n):
        f.write(str(i) + '\t' + str(c[i]) + '\n')
    pylab.loglog (x, c, color='red', marker='.', linestyle='-',
    ↪ linewidth=0.05, markersize=0.5)
    plt.savefig('F:\\CSW\\results\\plot_s_' + str(m[j]) + '.png')
    plt.clf()
    print('\n');
```

В результате получается фрагмент отвечающий за вывод данных:

```
n = 10000;
p = n
m = [1, 2, 5, 7, 10]
print('static:')
for j in range(5):
    print(str(m[j])+ '):', end = '\t')
    x = []
    c = []
    for i in range(n):
        c.append(0)
        x.append(i)
    for i in range(10):
        print(str(i) + ',', end = ' ')
        bag = my_bag(n = p, m = m[j])
        for k in bag.adjacency():
            c[len(k[1])]+=1
    for i in range(n):
        c[i]/=10
    fname = "F:\\CSW\\results\\data_s_" + str(m[j]) + ".txt"
    f = open(fname, "w")
    f.write("x\ty\n")
    for i in range(n):
        f.write(str(i) + '\t' + str(c[i]) + '\n')
    pylab.loglog (x, c, color='red', marker='.', linestyle='-',
    ↪ linewidth=0.05, markersize=0.5)
    plt.savefig('F:\\CSW\\results\\plot_s_' + str(m[j]) + '.png')
    plt.clf()
    print('\n');

print('poisson:')
for j in range(5):
    print(str(m[j])+ '):', end = '\t')
    x = []
    c = []
    for i in range(n):
        c.append(0)
        x.append(i)
    for i in range(10):
        print(str(i) + ',', end = ' ')
        bag = my_bag_poisson(n = p, m = m[j])
        for k in bag.adjacency():
```

```
                c[len(k[1])]+=1
        for i in range(n):
            c[i]/=10
        fname = "F:\\CSW\\results\\data_p_" + str(m[j]) + ".txt"
        f = open(fname, "w")
        f.write("x\ty\n")
        for i in range(n):
            f.write(str(i) + '\t' + str(c[i]) + '\n')
        pylab.loglog (x, c, color='red', marker='.', linestyle='-',
        ↪  linewidth=0.05, markersize=0.5)
        plt.savefig('F:\\CSW\\results\\plot_p_' + str(m[j]) + '.png')
        plt.clf()
        print('\n');
```

В результате выполнения программы создается текстовый файл, содержащий данные о графе и график распределения степеней основанный на этих данных. Следующий код представляет пример такого вывода.

```
x  y
0  4629.4
1  1031.2
2  1053.4
3  892.6
4  703.4
5  506.1
6  339.7
7  217.9
8  143.1
9  97.6
10  64.0
11  44.3
12  36.1
13  28.2
14  23.8
15  16.6
16  14.2
17  12.8
18  12.9
19  9.8
20  8.1
```

```
21  7.1
22  8.1
23  6.5
24  3.7
25  4.8
26  3.4
27  3.1
28  3.5
29  3.1
```

Примеры вывода графика распределения степеней приведены на Рис. 1, 2.

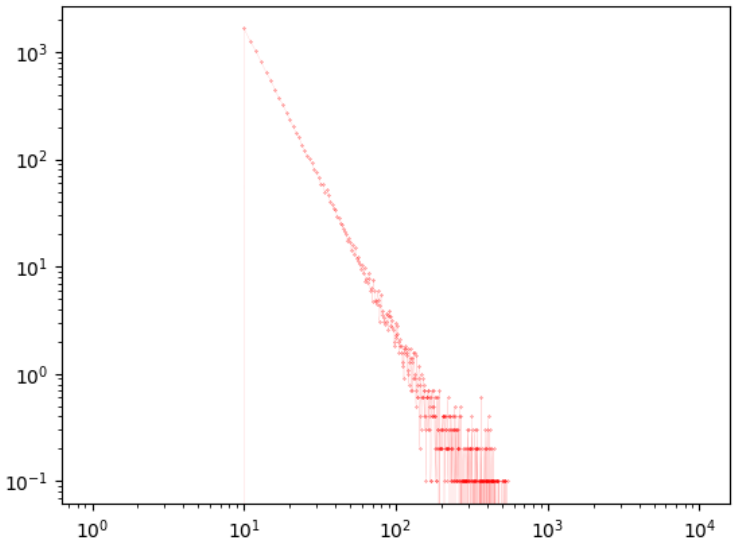

Рисунок 1 – График построенный программой
для стандартной модели

Полный код программы приведен в приложении Б.

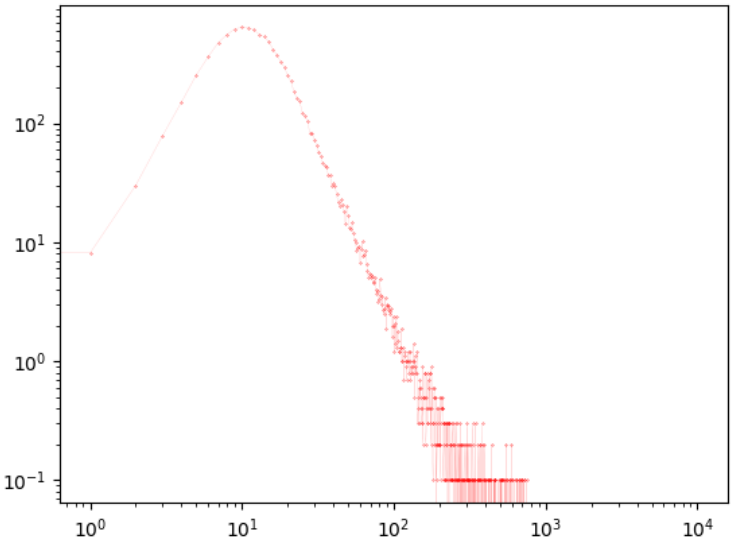

Рисунок 2 – График построенный программой
для модифицированной модели

3 Анализ

Так как модель Барабаши—Альберт подходит для описания реальных систем, в том числе сеть Интернет то распределение степеней вершин в этой модели должно соответствовать распределению степеней в реальных сетях.

Например, график распределения степеней при взаимодействии белков, представленный на рис. 3 (сиреневые точки) [11], соответствует графику функции $y = x^{-\gamma}$. В логорифмической системе координат эта функция выглядит как убывающая линейная функция с коэфициентом $-\gamma$, так как $log(y) = -\gamma log(x)$. В большинстве случаев коэфициент $\gamma = 3$.

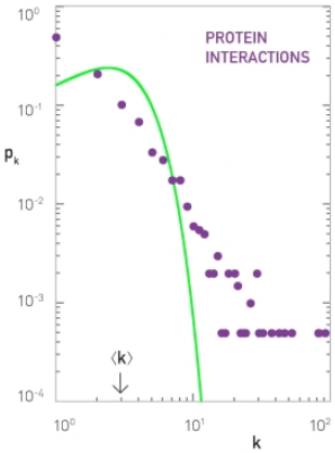

Рисунок 3 – Распределение степеней при взаимодействии белков

В нашей программной реализации для наборов выходных данных были построены графики для исследования закона распределения степеней и подбора числа γ в случае, если распределение степеней представляет собой степенную функцию.

На графиках для стандартной модели Барабаши—Альберт (рис. 4–6, 16–20) хорошо заметна та же закономерность. При этом можно заметить, что значение γ не зависит от значения m.

Многие другие структуры в реальном мире не соотвесвуют степенному распределению. Например, на рис. 7 [11] видно, что начальный участок графика распределения степеней узлов в сети Интернет представляет собой показательную функцию, которая затем переходит в степенную функцию с $\gamma = 3$. Такое распределение называется смешанным, оно представляет собой сочетание степенного и показательного законов. [11]

На графиках модифицированной модели Барабаши—Альберт(рис. 8-10,

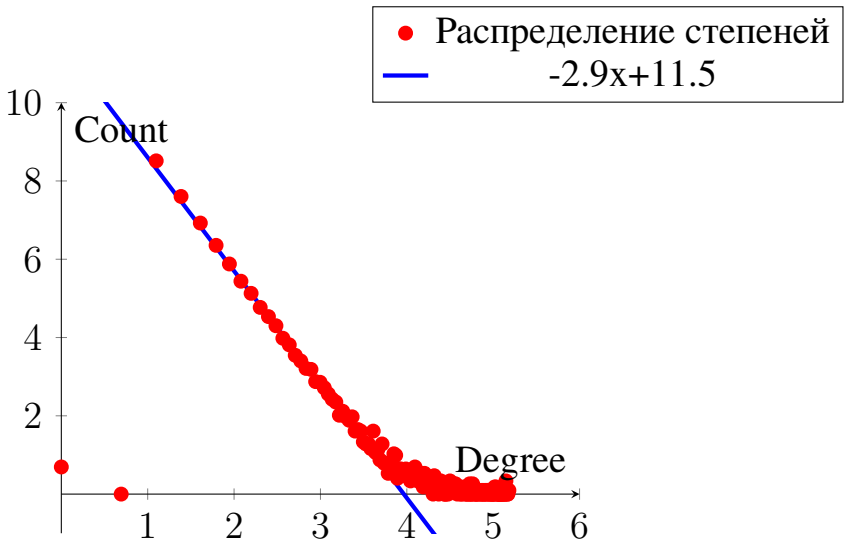

Рисунок 4 – Стандартная модель Барабаши—Альберт, $m = 2$

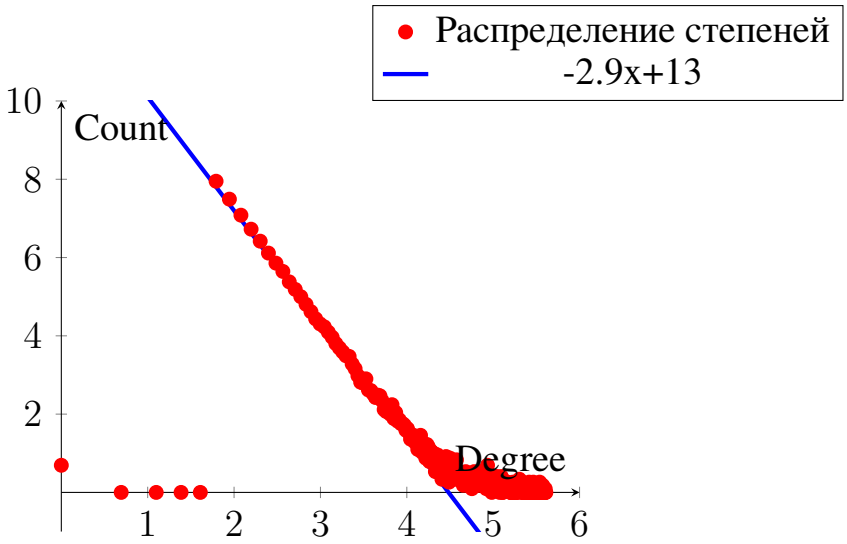

Рисунок 5 – Стандартная модель Барабаши—Альберт, $m = 5$

11-15 в приложении A) можно обнаружить описанное распределение. И γ всё так же равно 2.9 независимо от показателя m.

Все построенные графики приведены в приложении A

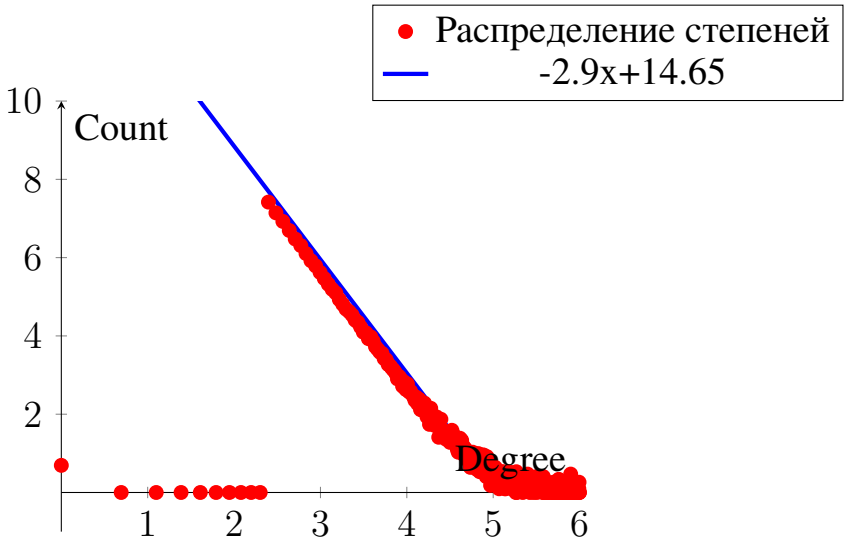

Рисунок 6 – Стандартная модель Барабаши—Альберт, $m = 10$

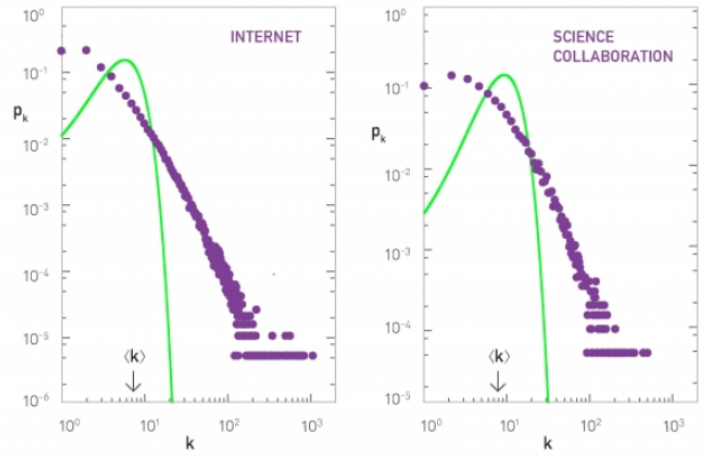

Рисунок 7 – Распределение степеней в сети Интернет и при научной кооперации

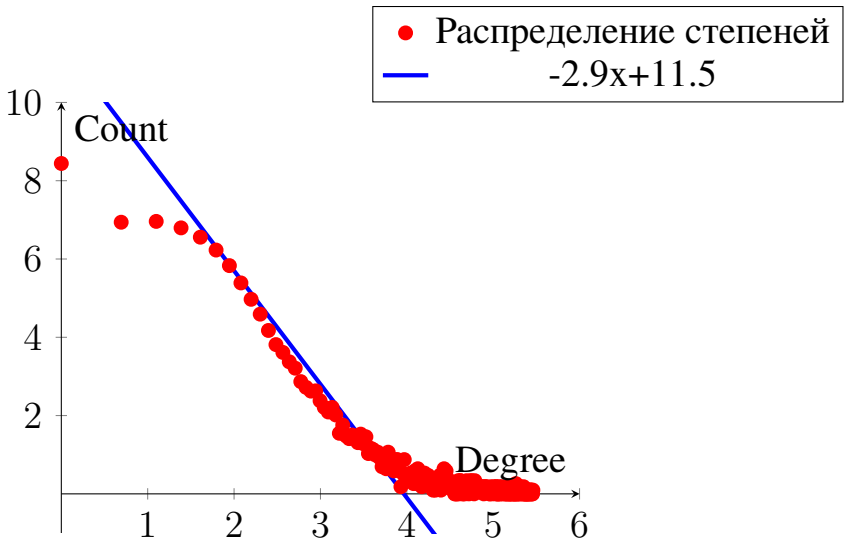

Рисунок 8 – Модифицированная модель Барабаши—Альберт, $m = 1$

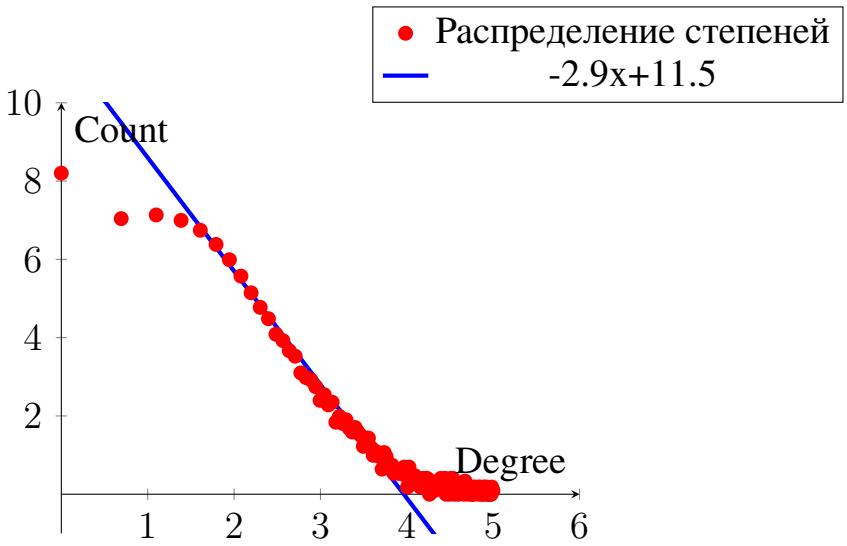

Рисунок 9 – Модифицированная модель Барабаши—Альберт, $m = 2$

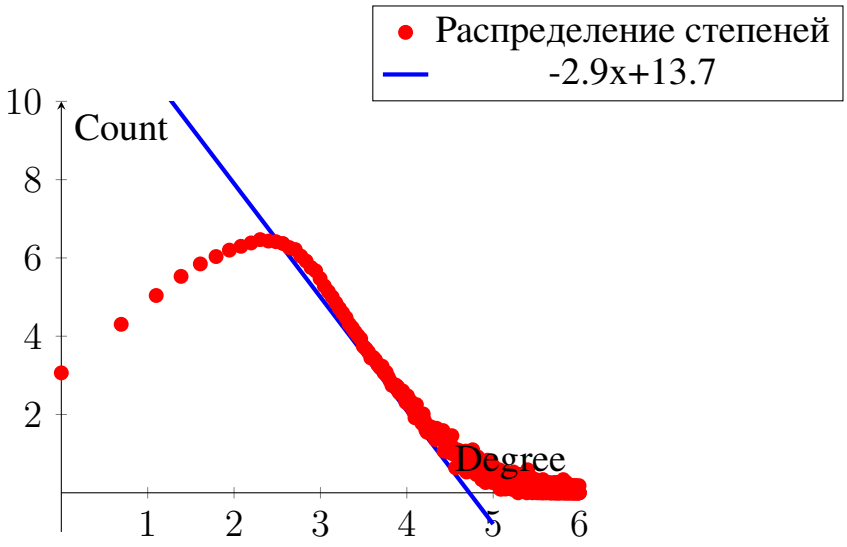

Рисунок 10 – Модифицированная модель Барабаши—Альберт, $m = 7$

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были изучены различные модели генерации случайных графов, и проведены исследования стандартной модели Барабаши—Альберт и её модификации со случайным количеством новых вершин, подчиняющимся распределению Пуасона. Проведенные эксперименты показали, что модификация алгоритма Барабаши—Альберт, в которой количество новых вершин подчиняется распределению Пуасона, распределение степеней полученного графа - смешанное, и что не зависимо от модификации и от выбора числа m полученное число гамма от степенного распределения равно 2.9. Также были изучены различные модули языка программирования Python такие как: networkx(для работы с графами), random и numpy.random(для работы со случайными величинами), а также matplotlib, matplotlib.pyplot и pylab(для построения и отображения графиков).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Erdös, P.* On random graphs i / P. Erdös, A. Rényi // *Publicationes Mathematicae Debrecen*. — 1959. — Vol. 6. — Pp. 290–297.

2 *Gilbert, E. N.* Random graphs / E. N. Gilbert // *Annals of Mathematical Statistics*. — 1959. — Vol. 30, no. 4. — Pp. 1141–1144.

3 Случайные графы [Электронный ресурс]. — URL: https://neerc.ifmo.ru/wiki/index.php?title=%D0%A1%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D1%8B%D0%B5_%D0%B3%D1%80%D0%B0%D1%84%D1%8B (Дата обращения 18.05.2021). Загл. с экр. Яз. рус.

4 ScienceHub 04: Теория случайных графов [Электронный ресурс]. — URL: https://habr.com/ru/company/postnauka/blog/201416/ (Дата обращения 21.05.2021). Загл. с экр. Яз. рус.

5 *Райгородский, А.* Модели случайных графов / А. Райгородский // *ТРУДЫ МФТИ*. — 2010. — Т. 2, № 4. — С. 130.

6 *Albert, R.* Statistical mechanics of complex networks / R. Albert, A.-L. Barabasi // *Reviews of Modern Physics*. — 2002. — Vol. 74, no. 1. — P. 47.

7 *Райгородский, А.* Модели случайных графов и их применени / А. Райгородский. — Москва, М.: Издательство МЦНМО, 2011.

8 *Берновски, М.* Случайные графы, модели и генераторы безмасштабных графов. / М. Берновски, Н. Кузюрин // *Труды Института системного программирования РАН*. — 2012. — Т. 22, № 4. — С. 419.

9 NetworkX [Электронный ресурс]. — URL: https://networkx.org/ (Дата обращения 16.05.2021). Загл. с экр. Яз. англ.

10 matplotlib.pyplot [Электронный ресурс]. — URL: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html (Дата обращения 20.05.2021). Загл. с экр. Яз. англ.

11 Network Science by Albert-Laszlo Barabasi [Электронный ресурс]. — URL: http://networksciencebook.com/chapter/4#advanced-a (Дата обращения 20.04.2021). Загл. с экр. Яз. англ.

ПРИЛОЖЕНИЕ А

Графики распределения степеней

В данном приложении приведены все построенные графы.

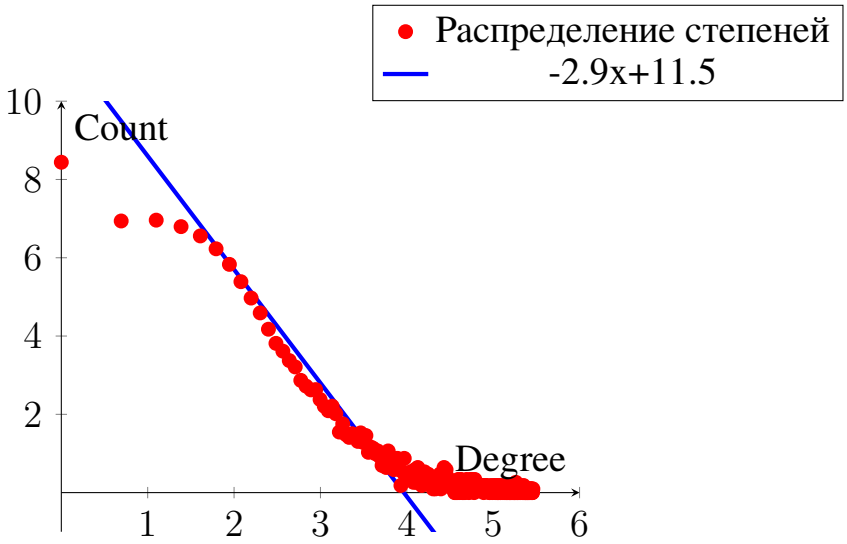

Рисунок 11 – Модифицированная модель Барабаши—Альберт, $m = 1$

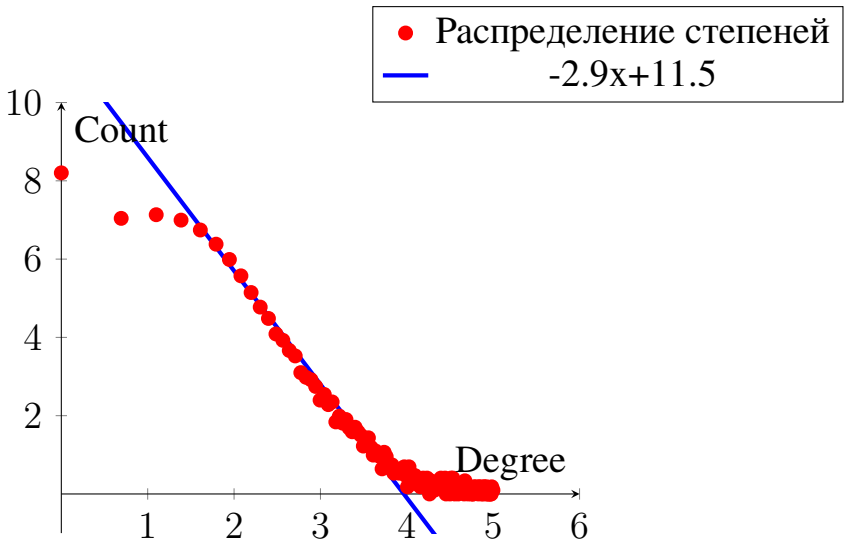

Рисунок 12 – Модифицированная модель Барабаши—Альберт, $m = 2$

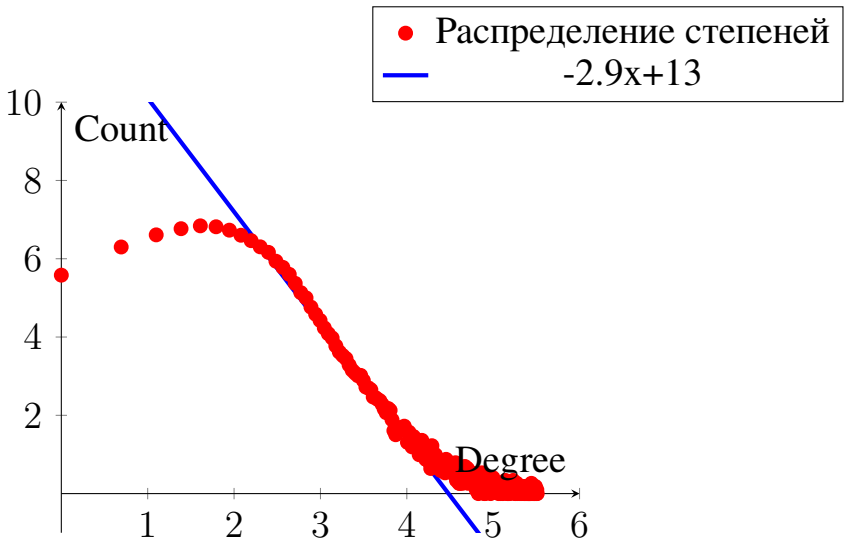

Рисунок 13 – Модифицированная модель Барабаши—Альберт, $m = 5$

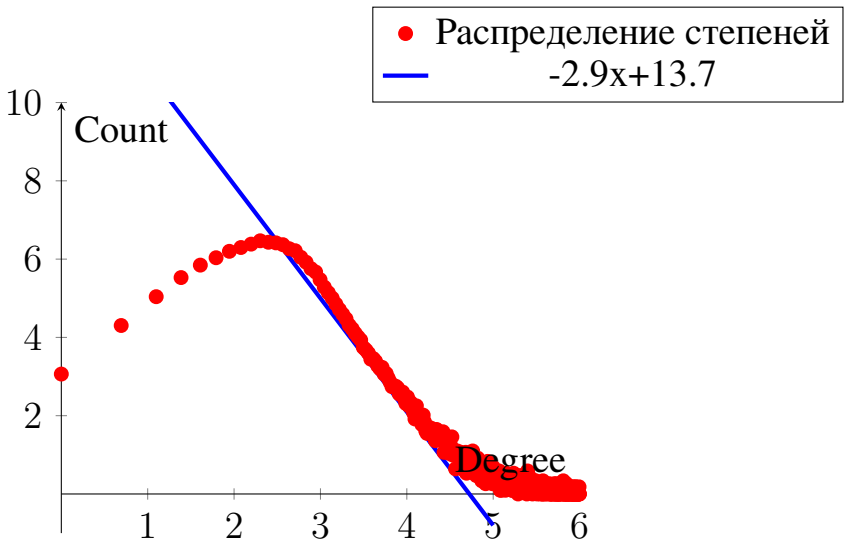

Рисунок 14 – Модифицированная модель Барабаши—Альберт, $m = 7$

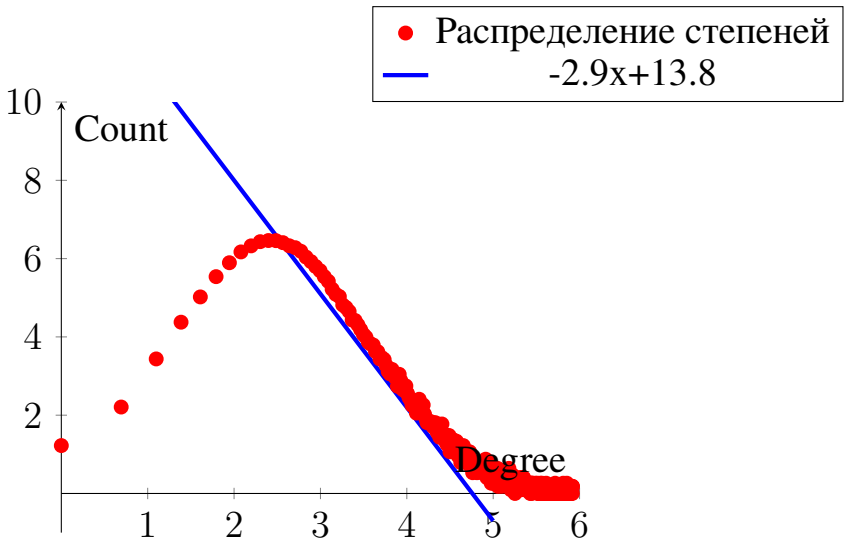

Рисунок 15 – Модифицированная модель Барабаши—Альберт, $m = 10$

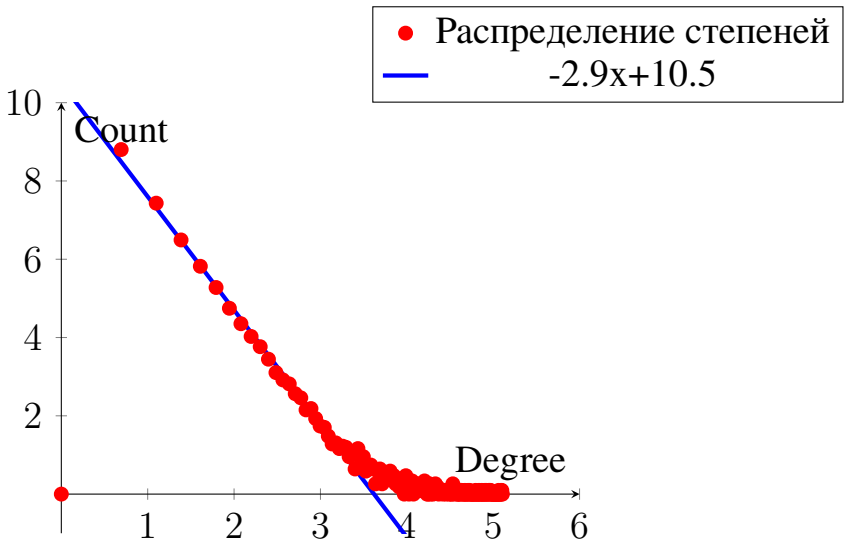

Рисунок 16 – Стандартная модель Барабаши—Альберт, $m = 1$

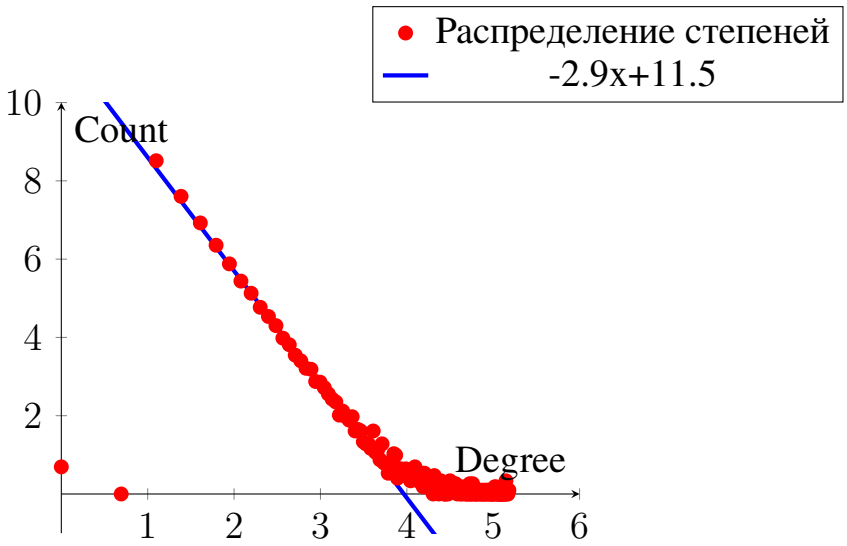

Рисунок 17 – Стандартная модель Барабаши—Альберт, $m = 2$

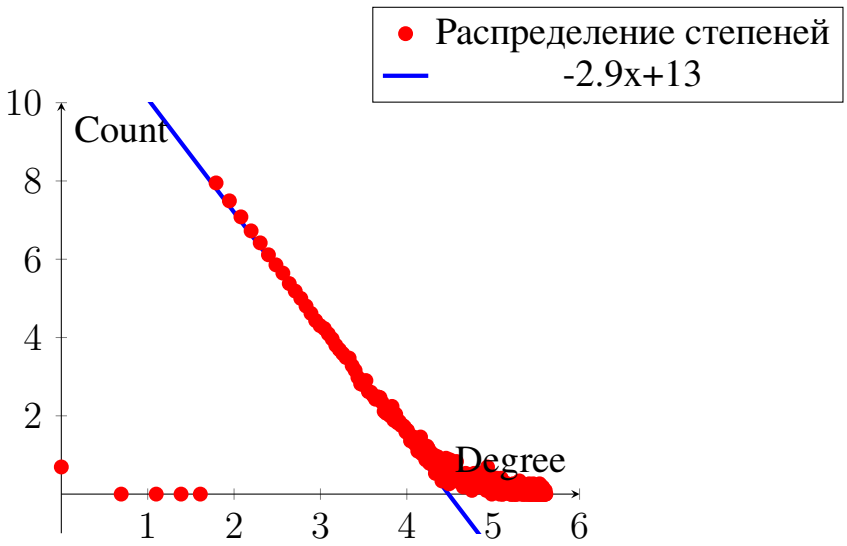

Рисунок 18 – Стандартная модель Барабаши—Альберт, $m = 5$

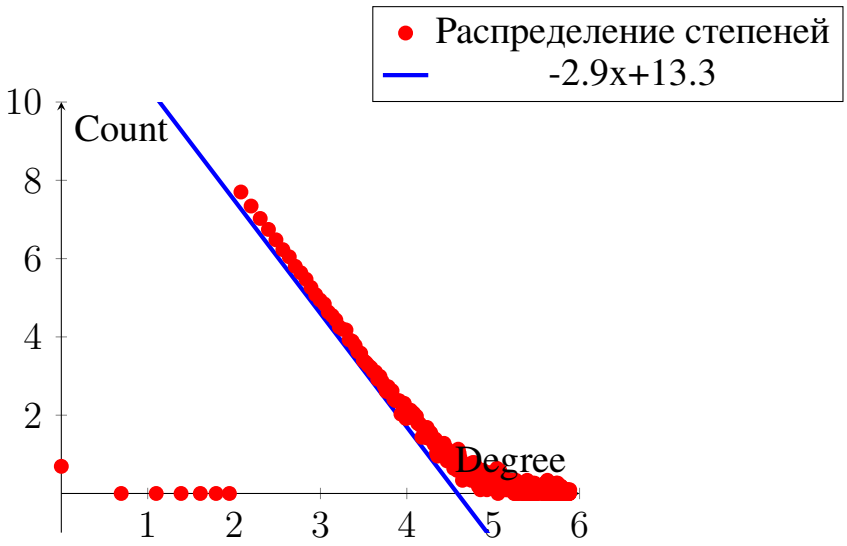

Рисунок 19 – Стандартная модель Барабаши—Альберт, $m = 7$

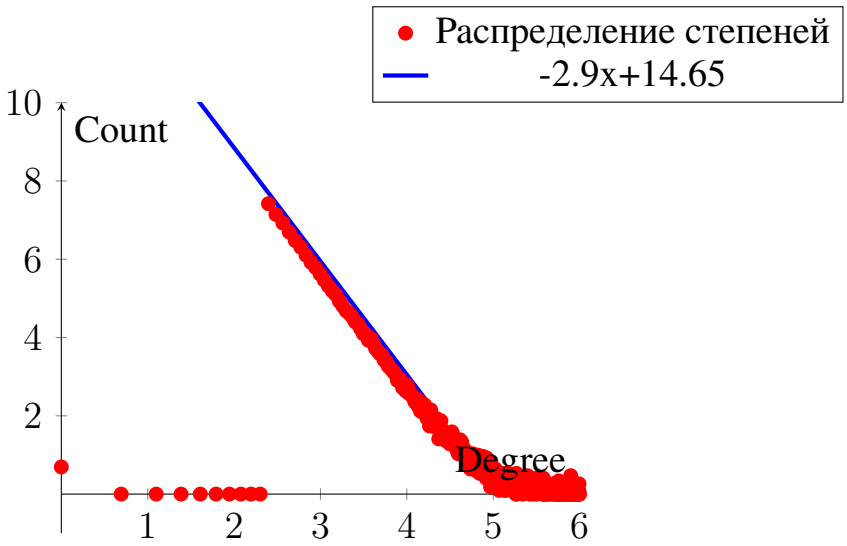

Рисунок 20 – Стандартная модель Барабаши—Альберт, $m = 10$

ПРИЛОЖЕНИЕ Б
Текст программы

В этом приложении приведён полный текст реализации модели Барабаши—Альберт.

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import networkx as nx
import math
import numpy.random
import random
import pylab

def my_bag_poisson(n, m):
    m0 = numpy.random.poisson(m)
    G = nx.complete_graph(m0)
    for i in range(m0, n):
        G.add_node(i)
    nodeCount = m0
    o = True
    nodes = []
    degrees = []
    used = []
    for j in range(n):
        used.append(False)
    for i in range(m0):
        nodes.append(i)
        degrees.append(m0)
    mi = numpy.random.poisson(m0, n)
    for i in range(m0, n):
        if not o :
            conections = []
            j = 0
            while j < min(mi[i], nodeCount):
                choice = random.choices(nodes, weights = degrees, k = 1)
                choosen = choice[0]
                if not used[choosen]:
                    G.add_edge(i, choosen)
                    j += 1
                    conections.append(choosen)
```

```
                    used[choosen] = True
            for j in range(min(mi[i], nodeCount)):
                used[conections[j]] = False
                degrees[conections[j] - 1] += 1
        else:
            G.add_edge(0, 1)
            o = False
        nodeCount += 1
        nodes.append(nodeCount)
        degrees.append(m)
    return G


def my_bag(n, m):
    G = nx.complete_graph(m)
    for i in range(m, n):
        G.add_node(i)
    nodeCount = m
    o = True
    nodes = []
    degrees = []
    used = []
    for j in range(n):
        used.append(False)
    for i in range(m):
        nodes.append(i)
        degrees.append(m)
    for i in range(m, n):
        if not o :
            conections = []
            j = 0
            while j < m:
                choice = random.choices(nodes, weights = degrees, k = 1)
                choosen = choice[0]
                if not used[choosen]:
                    G.add_edge(i, choosen)
                    j += 1
                    conections.append(choosen)
                    used[choosen] = True
            for j in range(m):
                used[conections[j]] = False
```

```
77                  degrees[conections[j] - 1] += 1
78          else:
79              G.add_edge(0, 1)
80              o = False
81          nodeCount += 1
82          nodes.append(nodeCount)
83          degrees.append(m)
84      return G
85
86
87  n = 10000;
88  p = n
89  m = [1, 2, 5, 7, 10]
90  print('static:')
91  for j in range(5):
92      print(str(m[j])+ '):', end = '\t')
93      x = []
94      c = []
95      for i in range(n):
96          c.append(0)
97          x.append(i)
98      for i in range(10):
99          print(str(i) + ',', end = ' ')
100         bag = my_bag(n = p, m = m[j])
101         for k in bag.adjacency():
102             c[len(k[1])]+=1
103     for i in range(n):
104         c[i]/=10
105     fname = "F:\\CSW\\results\\data_s_" + str(m[j]) + ".txt"
106     f = open(fname, "w")
107     f.write("x\ty\n")
108     for i in range(n):
109         f.write(str(i) + '\t' + str(c[i]) + '\n')
110     pylab.loglog (x, c, color='red', marker='.', linestyle='-',
        ↪ linewidth=0.05, markersize=0.5)
111     plt.savefig('F:\\CSW\\results\\plot_s_' + str(m[j]) + '.png')
112     plt.clf()
113     print('\n');
114
115 print('poisson:')
116 for j in range(5):
```

```
117     print(str(m[j])+ '):', end = '\t')
118     x = []
119     c = []
120     for i in range(n):
121         c.append(0)
122         x.append(i)
123     for i in range(10):
124         print(str(i) + ',', end = ' ')
125         bag = my_bag_poisson(n = p, m = m[j])
126         for k in bag.adjacency():
127             c[len(k[1])]+=1
128     for i in range(n):
129         c[i]/=10
130     fname = "F:\\CSW\\results\\data_p_" + str(m[j]) + ".txt"
131     f = open(fname, "w")
132     f.write("x\ty\n")
133     for i in range(n):
134         f.write(str(i) + '\t' + str(c[i]) + '\n')
135     pylab.loglog (x, c, color='red', marker='.', linestyle='-',
        ↪ linewidth=0.05, markersize=0.5)
136     plt.savefig('F:\\CSW\\results\\plot_p_' + str(m[j]) + '.png')
137     plt.clf()
138     print('\n');
```