|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  |  | | --- | --- | --- | |  |  |  | | Министерство образования и науки РОССИЙСКОЙ ФЕДЕРАЦИИ | | |  Федеральное государственное бюджетное образовательное учреждение высшего образования  **«МИРЭА – Российский технологический университет»** | |
|  | |
|  | |
|  |  |

ИНСТИТУТ КИБЕРНЕТИКИ

КАФЕДРА ВЫСШЕЙ МАТЕМАТИКИ

Лабораторная работа 1

 по курсу «**Случайные процессы**»

Тема: **Однородная цепь** **Маркова с тремя состояниями**

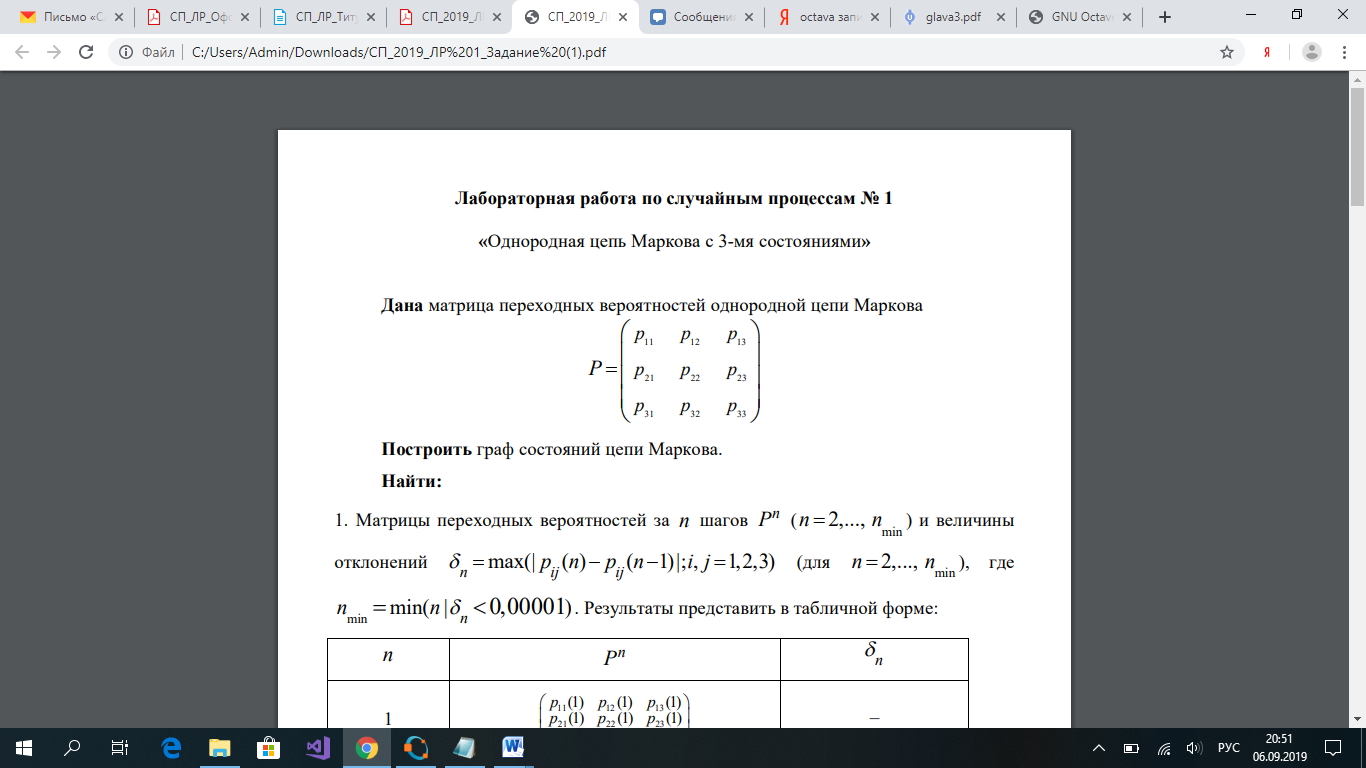
Выполнил:

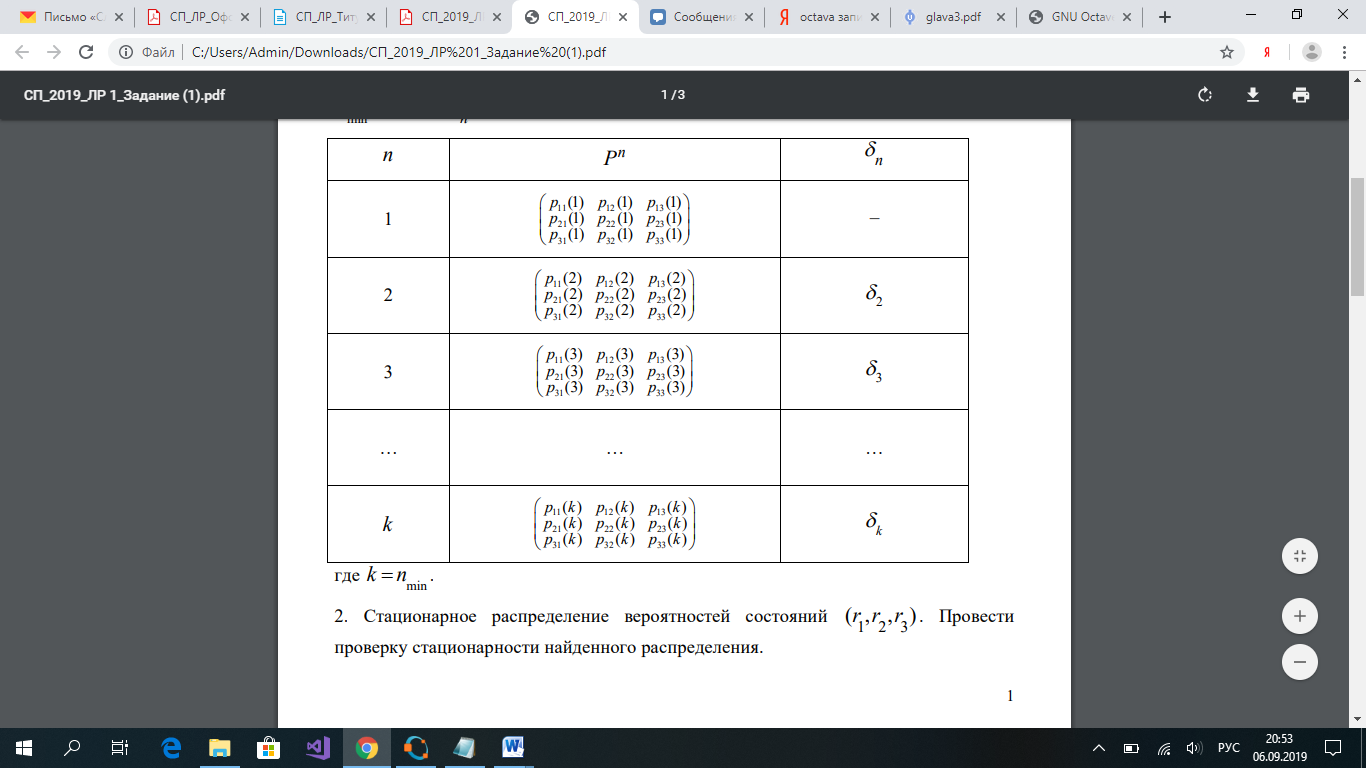
Студент 4-го курса

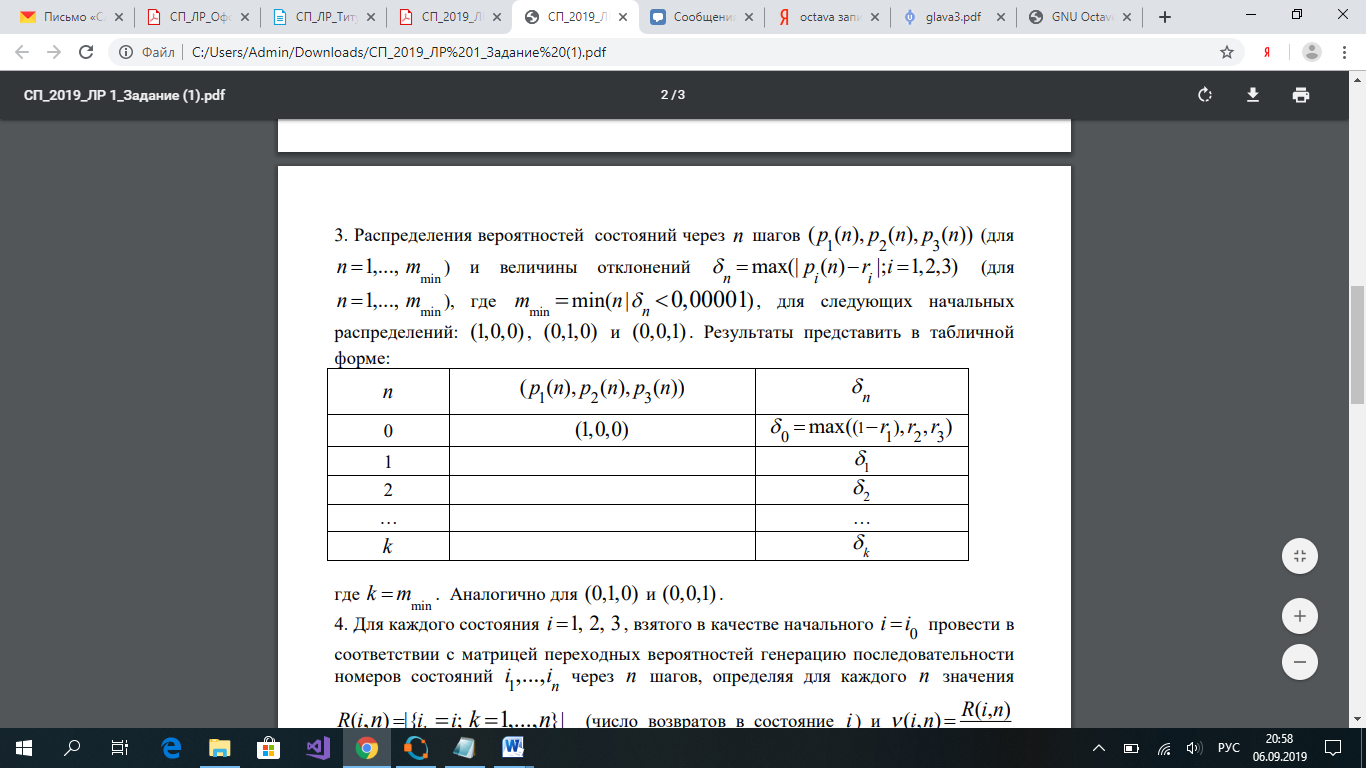
Жолковский Д.А.

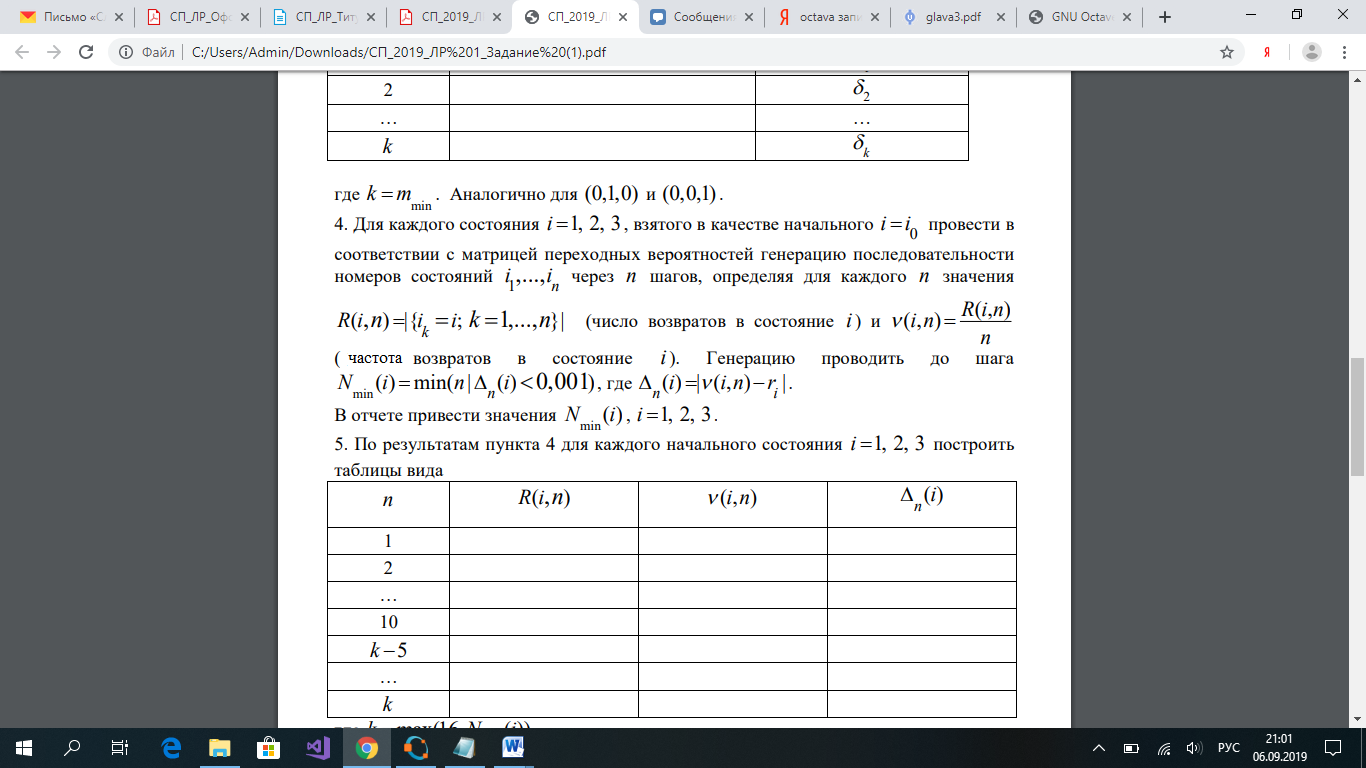
Группа: КМБО-01-16

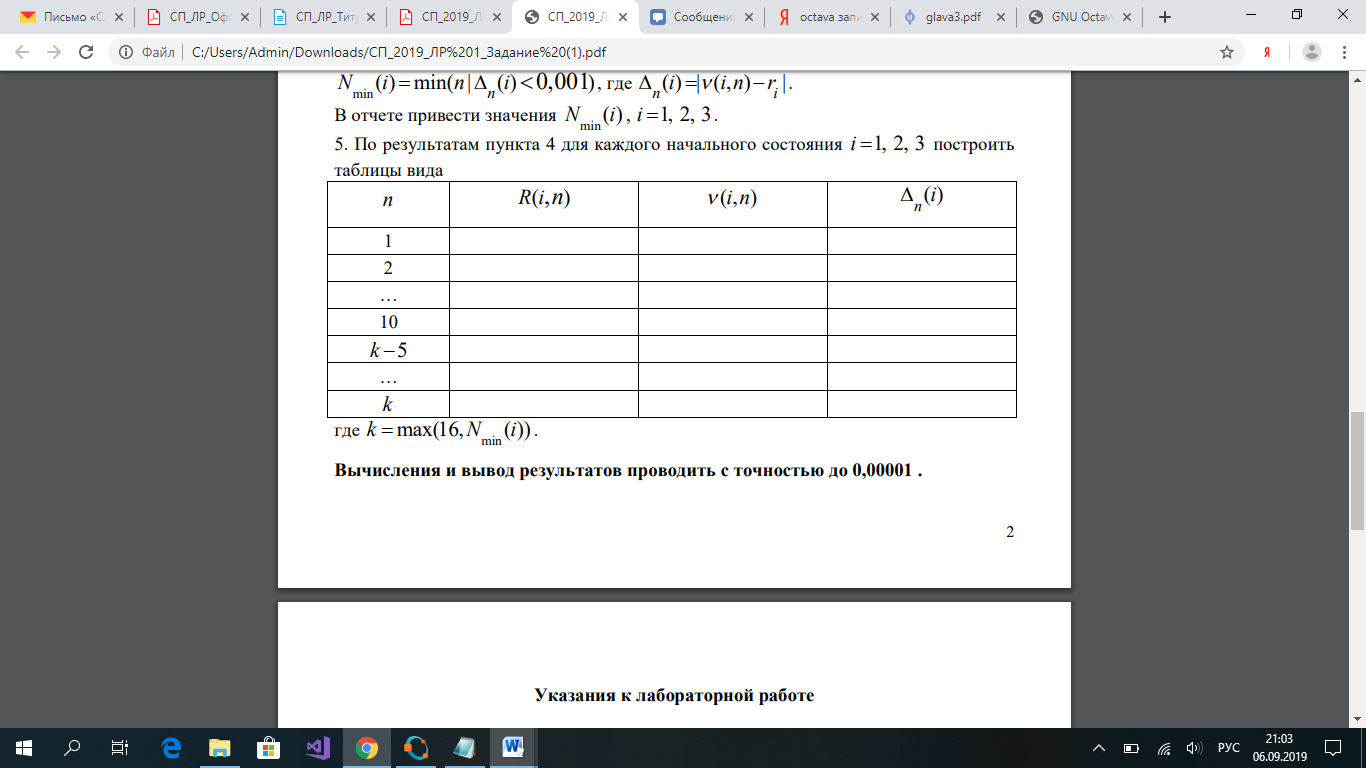
МОСКВА 2019











**Краткие теоретические сведения**

Рассмотрим последовательность случайных величин (с.в.) , принимающих значения , , ….



Последовательность с.в. называется цепью Маркова, если для произвольного набора и любых справедливо равенство



Цепь Маркова называется однородной, если для всех вероятности не зависит от .



Вероятности называются переходными, а матрица - матрицей переходных вероятностей цепи Маркова.



Матрица переходных вероятностей обладает следующими свойствами:

1. ;



1. .



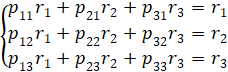
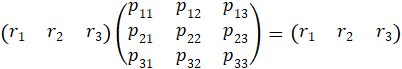
Распределение цепи Маркова называется *стационарным*, если оно останется неизменным на каждом шаге. Стационарное распределение удовлетворяет соотношению .



Если существует и , то распределение называется *предельным*.



Общие формулы для нахождения стационарного распределения:



Данная однородная система имеет ранг два. Добавим ещё одно условие: . Заменим это условие в системе так, чтоб её ранг стал равен трём. Решением полученной системы будет стационарное распределение .



**Средства высокоуровневого интерпретируемого языка программирования Python, которые использованы в программе расчета**

numpy.linalg.matrix\_power – возведение матрицы в натуральную степень

numpy.linalg.solve – решение слау

numpy.round – округление всех элементов матрицы

numpy.concatenate – конкатенация матриц

numpy.identity – единичная матрица

numpy.ones – матрица, целиком состоящая из единиц

numpy.max – максимальный элемент в матрице

numpy.absolute – модуль матрицы

numpy.set\_printoptions – форматирование выхода (например, без экспоненциальной формы)

.shape – получение размерности матрицы

.reshape – изменение размерности матрицы

yield – создание части генерируемой последовательности (используется вместо return для генераторов)

random.choices – выбор одного из элементов с возможностью задания весов (вероятностей)

tuple – преобразование массива в кортеж

itertools.product – проход по всем комбинациям

copy.copy – безопасное копирование переменной

inspect.isgeneratorfunction – проверка, является ли функция генератором

range – генератор массива

networkx.MultiDiGraph – создание графа

@... – декоратор функции, A @ B – матричное произведение (A \* B) – скалярное

pandas.DataFrame – создание таблицы (удобный формат для записи и чтения)

parser.parse\_args – парсинг командной строки

.tolist – метод преобразования матрицы в список

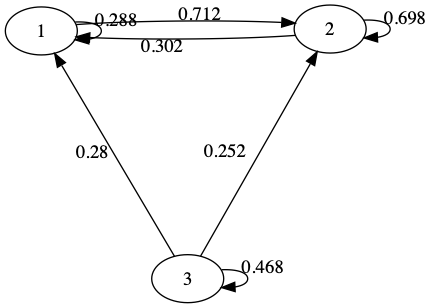
enumerate – проход по итерируемому объекту с перечислением индексов его элементов

**Результаты расчетов**

Вариант №9

Исходные данные:

Граф состояний цепи Маркова:



Задание №1:

|  |  |  |
| --- | --- | --- |
| *n* |  |  |
| 1 |  | - |
| 2 |  | 0.24898 |
| 3 |  | 0.11652 |
| 4 |  | 0.05453 |
| 5 |  | 0.02552 |
| 6 |  | 0.01194 |
| 7 |  | 0.00559 |
| 8 |  | 0.00262 |
| 9 |  | 0.00122 |
| 10 |  | 0.00057 |
| 11 |  | 0.00027 |
| 12 |  | 0.00013 |
| 13 |  | 6e-05 |
| 14 |  | 3e-05 |
| 15 |  | 1e-05 |
| 16 |  | 1e-05 |

Задание №2:

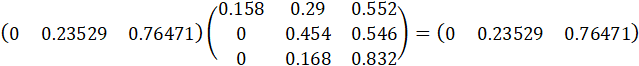
Стационарное распределение вероятностей состояний

0.29783, 0.70217, 0

Проверка стационарности найденного распределения:



0.29783, 0.70217, 00.29783, 0.70217, 0



Задание №3:

|  |  |  |
| --- | --- | --- |
| *n* |  |  |
| 0 | [1.0, 0.0, 0.0] | 0.70217 |
| 1 | [0.288, 0.712, 0.0] | 0.00983 |
| 2 | [0.29797, 0.70203, 0.0] | 0.0 |

|  |  |  |
| --- | --- | --- |
| *n* |  |  |
| 0 | [0.0, 1.0, 0.0] | 0.29783 |
| 1 | [0.302, 0.698, 0.0] | 0.00417 |
| 2 | [0.29777, 0.70223, 0.0] | 0.0 |

|  |  |  |
| --- | --- | --- |
| *n* |  |  |
| 0 | [0.0, 0.0, 1.0] | 1.0 |
| 1 | [0.28, 0.252, 0.468] | 0.468 |
| 2 | [0.28778, 0.49319, 0.21902] | 0.1025 |
| 3 | [0.29315, 0.60434, 0.1025] | 0.02245 |
| 4 | [0.29564, 0.65639, 0.04797] | 0.00492 |
| 5 | [0.29681, 0.68074, 0.02245] | 0.00108 |
| 6 | [0.29735, 0.69214, 0.01051] | 0.00024 |
| 7 | [0.29761, 0.69748, 0.00492] | 5e-05 |
| 8 | [0.29773, 0.69997, 0.0023] | 1e-05 |
| 9 | [0.29778, 0.70114, 0.00108] | 0.0 |

Задание №4:



**Анализ результатов и выводы**

Стационарное распределение: 0.29783, 0.70217, 0



Строки матрицы :



(0.29797, 0.70203, 0.0)



(0.29777, 0.70223, 0.0)



(0.29778, 0.70114, 0.00108)



Вывод: с погрешность 0.00107 значения векторов совпадают.

Задание №5:

|  |  |  |  |
| --- | --- | --- | --- |
| *n* | R(1,n) | v(1,n) |  |
| 1 | 0 | 0.0 | 0.29783 |
| 2 | 0 | 0.0 | 0.29783 |
| 3 | 1 | 0.33333 | 0.0355 |
| 4 | 1 | 0.25 | 0.04783 |
| 5 | 1 | 0.2 | 0.09783 |
| 6 | 2 | 0.33333 | 0.0355 |
| 7 | 2 | 0.28571 | 0.01212 |
| 8 | 2 | 0.25 | 0.04783 |
| 9 | 2 | 0.22222 | 0.07561 |
| 10 | 3 | 0.3 | 0.00217 |
| … | … | … | … |
| 59 | 16 | 0.27119 | 0.02664 |
| 60 | 16 | 0.26667 | 0.03116 |
| 61 | 17 | 0.27869 | 0.01914 |
| 62 | 17 | 0.27419 | 0.02364 |
| 63 | 18 | 0.28571 | 0.01212 |
| 64 | 19 | 0.29688 | 0.00095 |

|  |  |  |  |
| --- | --- | --- | --- |
| *n* | R(2,n) | v(2,n) |  |
| 1 | 1 | 1.0 | 0.29783 |
| 2 | 2 | 1.0 | 0.29783 |
| 3 | 3 | 1.0 | 0.29783 |
| 4 | 4 | 1.0 | 0.29783 |
| 5 | 5 | 1.0 | 0.29783 |
| 6 | 6 | 1.0 | 0.29783 |
| 7 | 7 | 1.0 | 0.29783 |
| 8 | 7 | 0.875 | 0.17283 |
| 9 | 8 | 0.88889 | 0.18672 |
| 10 | 8 | 0.8 | 0.09783 |
| … | … | … | … |
| 270 | 191 | 0.70741 | 0.00524 |
| 271 | 191 | 0.7048 | 0.00263 |
| 272 | 192 | 0.70588 | 0.00371 |
| 273 | 192 | 0.7033 | 0.00113 |
| 274 | 193 | 0.70438 | 0.00221 |
| 275 | 193 | 0.70182 | 0.00035 |

|  |  |  |  |
| --- | --- | --- | --- |
| *n* | R(3,n) | v(3,n) |  |
| 1 | 1 | 1.0 | 1.0 |
| 2 | 1 | 0.5 | 0.5 |
| 3 | 1 | 0.33333 | 0.33333 |
| 4 | 1 | 0.25 | 0.25 |
| 5 | 1 | 0.2 | 0.2 |
| 6 | 1 | 0.16667 | 0.16667 |
| 7 | 1 | 0.14286 | 0.14286 |
| 8 | 1 | 0.125 | 0.125 |
| 9 | 1 | 0.11111 | 0.11111 |
| 10 | 1 | 0.1 | 0.1 |
| … | … | … | … |
| 996 | 1 | 0.001 | 0.001 |
| 997 | 1 | 0.001 | 0.001 |
| 998 | 1 | 0.001 | 0.001 |
| 999 | 1 | 0.001 | 0.001 |
| 1000 | 1 | 0.001 | 0.001 |
| 1001 | 1 | 0.001 | 0.001 |

**Список литературы**

1. Лобузов А.А., Гумляева С.Д., Норин Н.В. Задачи по теории случайных процессов. — М.: МИРЭА, 1993.
2. Булинский А. В., А. Н. Ширяев А. Н. Теория случайных процессов: Учебник для вузов. — М.: ФИЗМАТЛИТ, 2005
3. Вентцель Е. С., Овчаров Л. А. Теория случайных процессов и ее инженерные приложения: Учеб. пособие для вузов. — М.: Высшая школа, 2007.

**Приложение (Листинг программы)**

#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

**import** **sys**

**import** **argparse**

**import** **numpy** **as** **np**

**from** **numpy.linalg** **import** matrix\_power, solve

**import** **pandas** **as** **pd**

**from** **random** **import** choices

**from** **itertools** **import** product

**from** **copy** **import** copy

**from** **inspect** **import** isgeneratorfunction

**import** **networkx** **as** **nx**

**from** **networkx.drawing.nx\_agraph** **import** to\_agraph

**def** **createParser**():

parser = argparse.ArgumentParser()

parser.add\_argument('-p', '--path', default='Data/input.csv', type=str)

**return** parser

# Декоратор для округления возвращаемых значений других функций

**def** **round\_output**(fn):

rn = **5**

**if** isgeneratorfunction(fn):

**def** **rounder**(\*args, \*\*kwargs):

**for** output **in** fn(\*args, \*\*kwargs):

**if** type(output) != tuple:

**yield** np.round(output, rn)

**else**:

res = []

**for** element **in** output:

res.append(np.round(element, rn))

**yield** tuple(res)

**return** rounder

**else**:

**def** **rounder**(\*args, \*\*kwargs):

output = fn(\*args, \*\*kwargs)

**return** np.round(output, rn)

**return** rounder

**def** **M2W**(P):

graph = {(str(i+**1**), str(j+**1**)): round(P[i][j], **5**)

**for** i, j **in** product(range(P.shape[**0**]), repeat=**2**) **if** P[i][j] > **0.**}

**return** graph

**def** **plot\_graph**(P, path='graph'):

graph = M2W(P)

G=nx.MultiDiGraph()

**for** edge **in** graph:

G.add\_edge(edge[**0**], edge[**1**])

G.graph['edge'] = {'arrowsize': '1', 'splines': 'curved'}

G.graph['graph'] = {'scale': '3'}

A = to\_agraph(G)

A.layout('neato')#neato, dot, twopi, circo, fdp, nop, wc, acyclic, gvpr, gvcolor, ccomps, sccmap, tred, sfdp, unflatten

**for** pair **in** graph:

edge = A.get\_edge(pair[**0**], pair[**1**])

edge.attr['label'] = str(graph[pair]) + " "

A.draw(f'{path}.png')

**@round\_output**

**def** **n\_delta\_powers\_from\_2**(P, dm=**1e-5**):

Pn = copy(P)

d = **2** \* dm

**while** d >= dm:

d = np.max(np.absolute(Pn @ P - Pn))

Pn = Pn @ P

**yield** Pn, d

**def** **save\_table\_csv**(table, path, columns, index, convert=True):

df = pd.DataFrame(np.array(table).T **if** convert **else** table,

index=index,

columns=columns)

df.to\_csv(path+'.csv', sep=';', encoding='utf-8')

**@round\_output**

**def** **get\_stationary\_d**(P):

A = np.concatenate(((P.T - np.identity(**3**))[:-**1**], np.ones((**1**, **3**))), axis=**0**)

B = np.array([**0**, **0**, **1**])

x = solve(A, B)

**return** x

**@round\_output**

**def** **p\_distributions**(start, P, stationary, dm=**1e-5**):

Pn = copy(P)

x = copy(start)

d = np.max(np.absolute(x - stationary))

**while** d >= dm:

d = np.max(np.absolute(x @ Pn - stationary))

x = start @ Pn

Pn = Pn @ P

**yield** x, d

**@round\_output**

**def** **p\_distribution**(start, P, stationary, n):

Pn = copy(P)

**for** i **in** range(n):

x = start @ Pn

Pn = Pn @ P

**return** x

**def** **generate\_next\_state**(current\_state\_id, P):

**return** choices(population=[**0**, **1**, **2**], weights=P[current\_state\_id])[**0**]

**@round\_output**

**def** **generate\_n\_states**(start\_state\_id, P, stationary, dm=**1e-3**):

current\_state\_id = start\_state\_id

count\_start\_state = **0**

N = dm \* **2**

n = **0**

**while** N >= dm:

current\_state\_id = generate\_next\_state(current\_state\_id, P)

n += **1**

count\_start\_state += **1** **if** current\_state\_id == start\_state\_id **else** **0**

v = count\_start\_state / n

N = abs(v - stationary[start\_state\_id])

**yield** count\_start\_state, v, N

**if** \_\_name\_\_ == '\_\_main\_\_':

parser = createParser()

namespace = parser.parse\_args(sys.argv[**1**:])

# ### Подготовка

np.set\_printoptions(suppress=True)

df = pd.read\_csv(namespace.path, encoding='utf-8', sep=';', dtype=np.float)

P = df.values

# #### 0) Построение графа

plot\_graph(P, 'Data/test')

# #### 1) Матрицы переходных вероятностей

ndp = [[np.round(P, **5**).tolist(), None]]

**for** Pn, d **in** n\_delta\_powers\_from\_2(P):

ndp.append([Pn.tolist(), d])

save\_table\_csv(ndp, 'Data/ndp',

columns=['Pn', 'd'],

index=range(**1**, len(ndp)+**1**),

convert=False)

**print**(f'n\_min = {len(ndp)}')

# #### 2) Стационарное распределение вероятностей

x = get\_stationary\_d(P)

**print**(x)

**print**(np.round(x @ P, **5**) == x)

# #### 3) Распределения вероятностей состояний через n шагов

starts = np.identity(**3**)

distr = [[[starts[i].tolist(), np.max(np.absolute(starts[i] - x))]] **for** i **in** range(starts.shape[**0**])]

**for** i **in** range(starts.shape[**0**]):

**for** p, d **in** p\_distributions(starts[i], P, x):

distr[i].append([p.tolist(), d])

**for** i, dist **in** enumerate(distr):

save\_table\_csv(dist, f'Data/distr{i}',

columns=['(p1(n), p2(n), p3(n))', 'd(n)'],

index=range(len(dist)),

convert=False)

# #### 4) и 5) Генерация последовательности номеров состояний через n шагов

generates = [[], [], []]

**for** i, generation **in** enumerate(generates):

**for** R, v, d **in** generate\_n\_states(i, P, x):

generation.append((R, v, d))

**for** i, generation **in** enumerate(generates):

**if** len(generation) > **16**:

save\_table\_csv(generation[**0**:**10**]+generation[-**6**:], f'Data/generation{i}',

columns=['R', 'v', 'd'],

index=list(range(**1**, **11**)) + list(range(len(generation)-**5**, len(generation)+**1**)),

convert=False)

**else**:

save\_table\_csv(generation, f'Data/generation{i}',

columns=['R', 'v', 'd'],

index=range(len(generation)),

convert=False)

**for** i, generation **in** enumerate(generates):

**print**(f'Nmin for state {i+1} = {len(generation)}')

# #### Анализ результатов и выводы

**print**(f'Стационарное распределение: {tuple(x)}**\n**')

k = len(ndp)

**print**(f'Строки Матрицы P^{k}:')

**print**(np.round(matrix\_power(P, k), **5**))

**for** dist **in** distr:

**print**(f'(p1({len(dist)}), p2({len(dist)}), p3({len(dist)})) = {tuple(dist[-1][0])}')

pn = np.concatenate((np.array(distr[**0**][-**1**][**0**]).reshape(**1**, -**1**),

np.array(distr[**1**][-**1**][**0**]).reshape(**1**, -**1**),

np.array(distr[**2**][-**1**][**0**]).reshape(**1**, -**1**)))

d = np.max(np.abs(np.round(matrix\_power(P, k), **5**) - pn))

**print**(f'**\n**Вывод: с погрешностью {d} значения векторов совпадают')