|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  |  | | --- | --- | --- | |  |  |  | | 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**

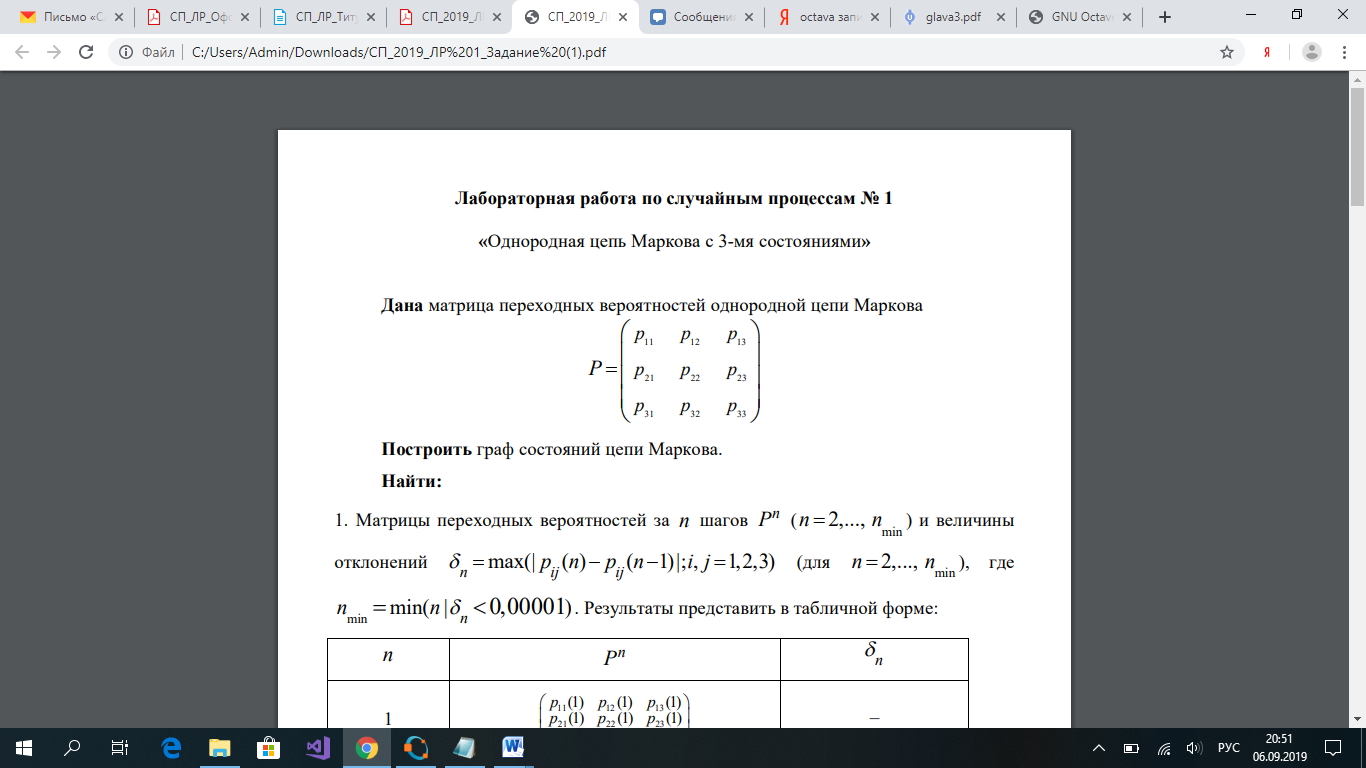
Выполнил:

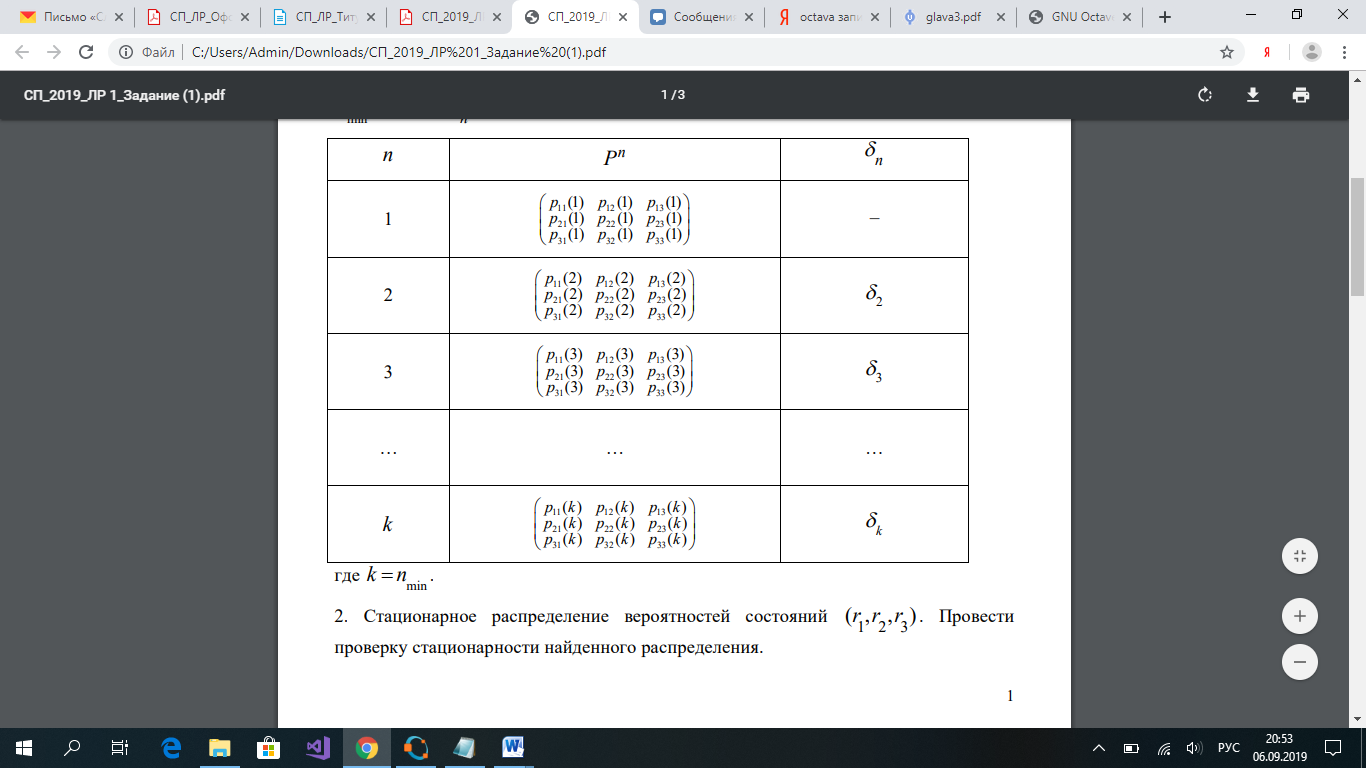
Студент 4-го курса

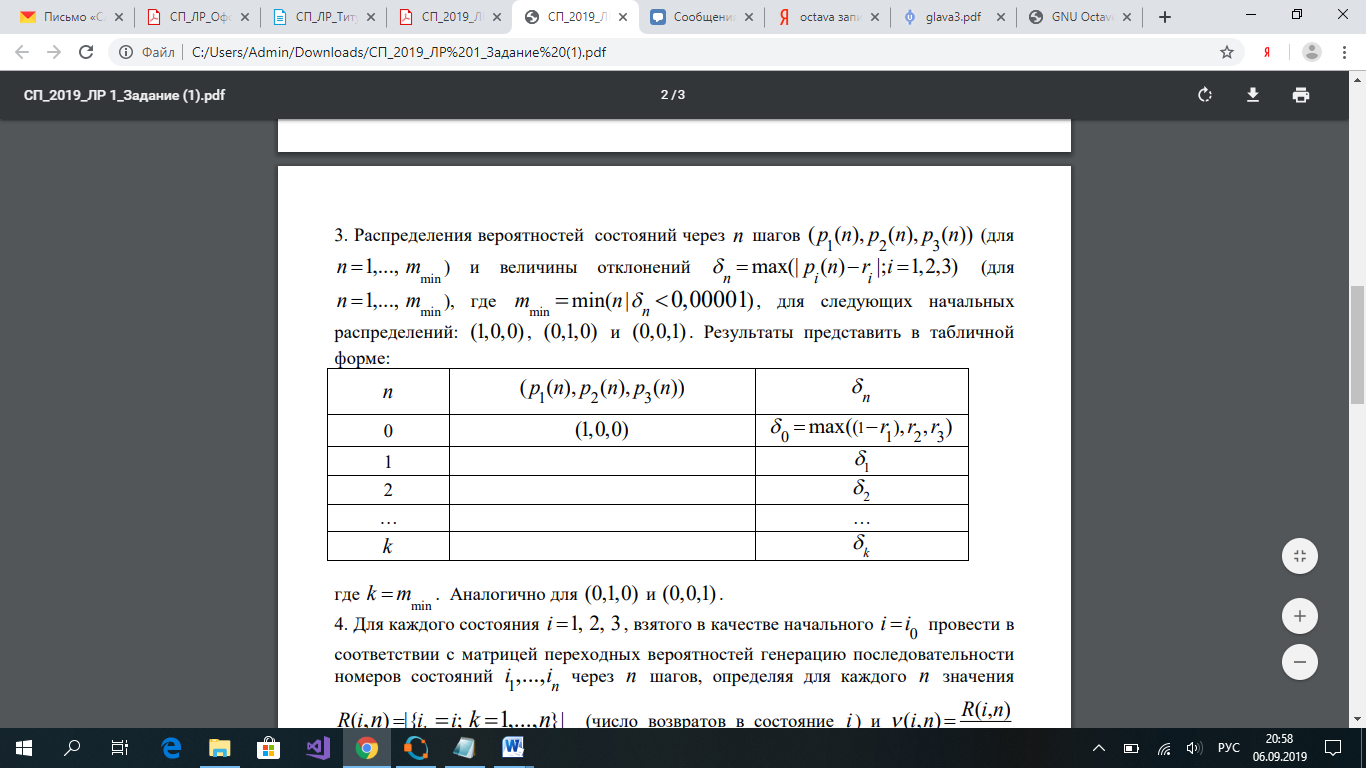
Жолковский Д.А.

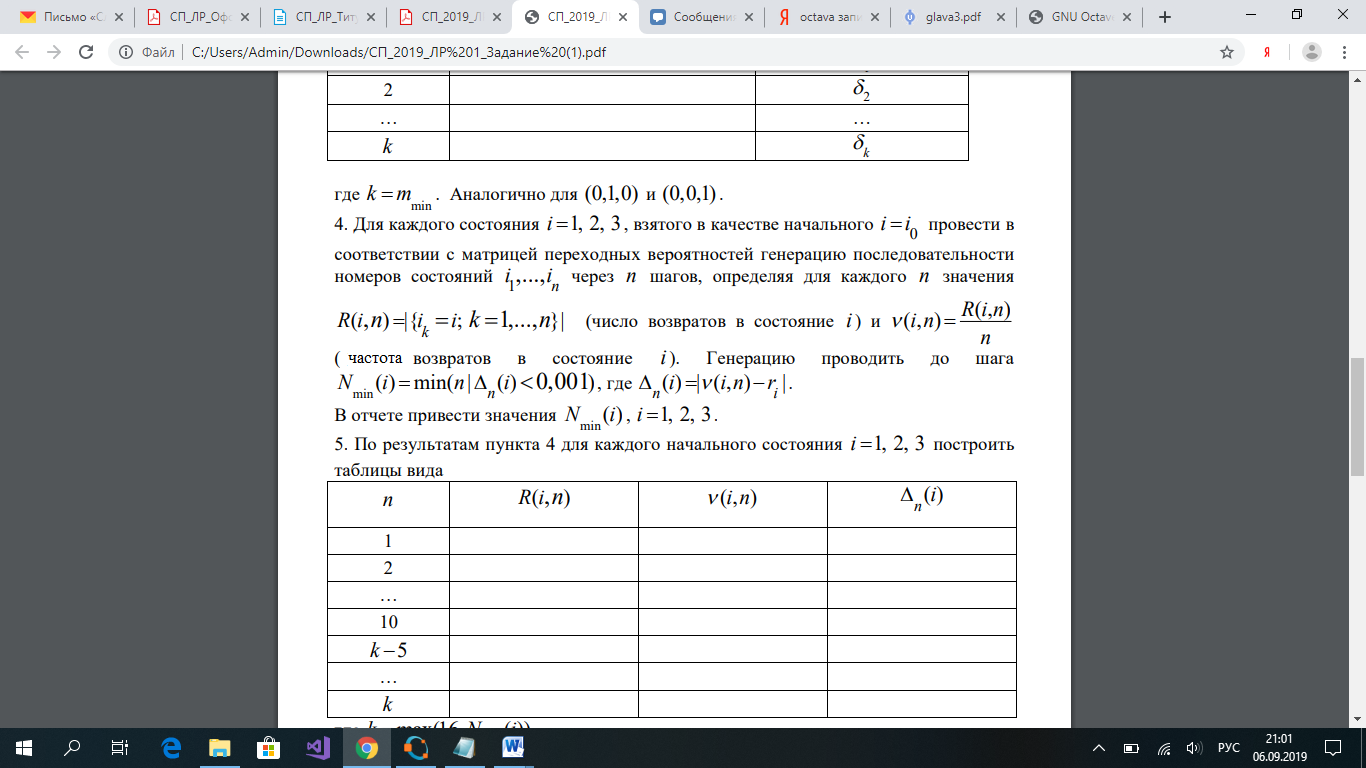
Группа: КМБО-01-16

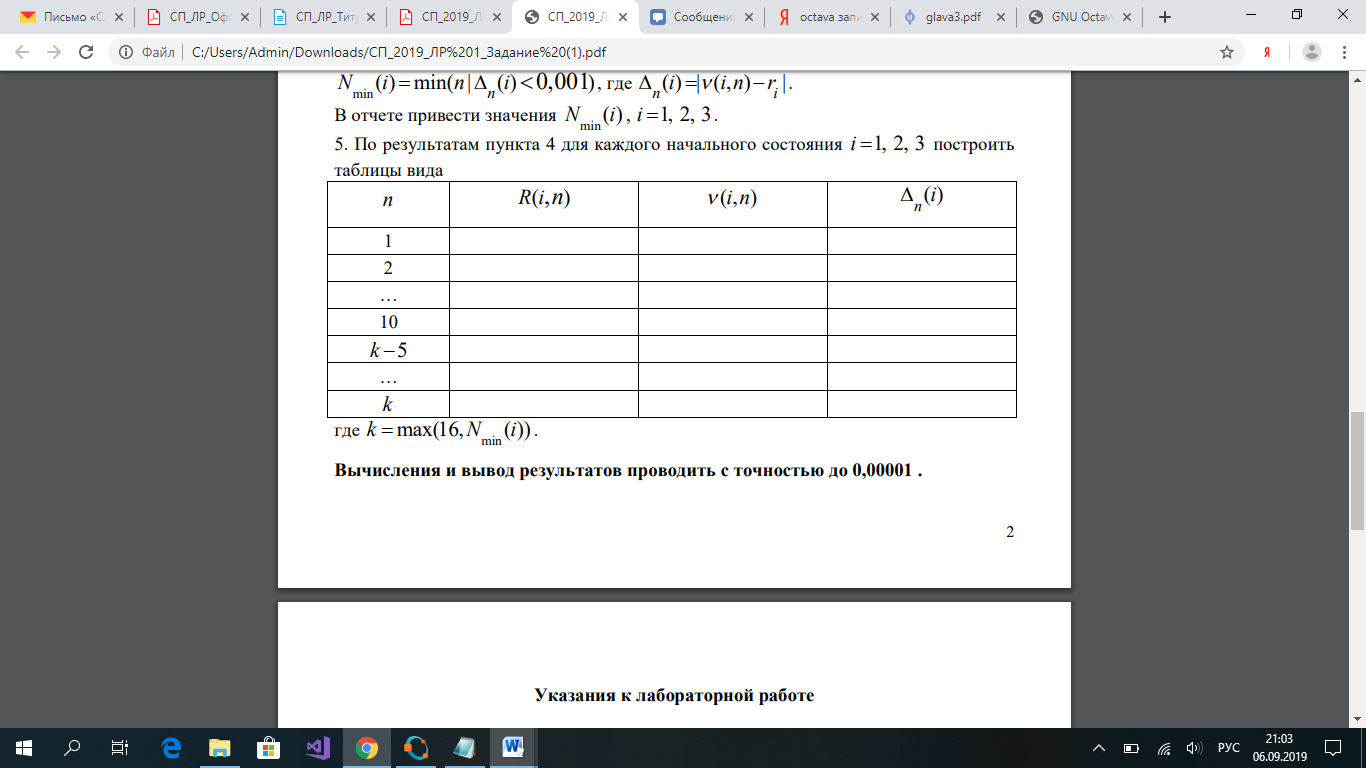
МОСКВА 2019











**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. .



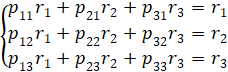
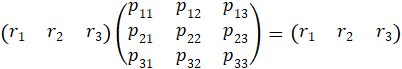
Распределение цепи Маркова называется *стационарным*, если оно останется неизменным на каждом шаге. Стационарное распределение удовлетворяет соотношению .



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

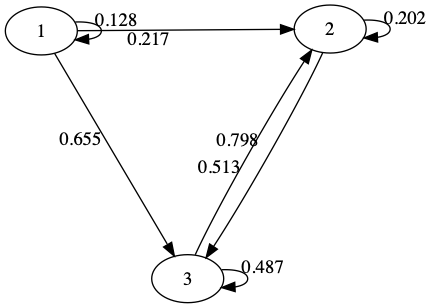
enumerate – проход по итерируемому объекту с перечислением индексов его элементов

**Результаты расчетов**

Вариант №10

Исходные данные:

Граф состояний цепи Маркова:



Задание №1:

|  |  |  |
| --- | --- | --- |
| *n* |  |  |
| 1 |  | - |
| 2 |  | 0.24818 |
| 3 |  | 0.07718 |
| 4 |  | 0.024 |
| 5 |  | 0.00747 |
| 6 |  | 0.00232 |
| 7 |  | 0.00072 |
| 8 |  | 0.00022 |
| 9 |  | 0.00007 |
| 10 |  | 0.00002 |
| 11 |  | 0.00001 |

Задание №2:

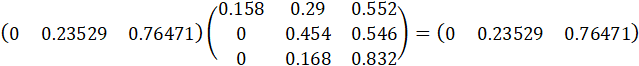
Стационарное распределение вероятностей состояний

0, 0.3913, 0.6087

Проверка стационарности найденного распределения:



0, 0.3913, 0.60870, 0.39131, 0.6087



Задание №3:

|  |  |  |
| --- | --- | --- |
| *n* |  |  |
| 0 | [1.0, 0.0, 0.0] | 1.0 |
| 1 | [0.128, 0.217, 0.655] | 0.1743 |
| 2 | [0.01638, 0.40762, 0.57599] | 0.00992 |
| 3 | [0.0021, 0.38138, 0.61652] | 0.00084 |
| 4 | [0.00027, 0.39377, 0.60596] | 0.00008 |
| 5 | [0.00003, 0.39046, 0.60951] | 0.0 |

|  |  |  |
| --- | --- | --- |
| *n* |  |  |
| 0 | [0.0, 1.0, 0.0] | 0.6087 |
| 1 | [0.0, 0.202, 0.798] | 0.1893 |
| 2 | [0.0, 0.45018, 0.54982] | 0.01831 |
| 3 | [0.0, 0.37299, 0.62701] | 0.00177 |
| 4 | [0.0, 0.397, 0.603] | 0.00017 |
| 5 | [0.0, 0.38953, 0.61047] | 0.0001 |
| 6 | [0.0, 0.39186, 0.60814] | 0.0 |

|  |  |  |
| --- | --- | --- |
| *n* |  |  |
| 0 | [0.0, 0.0, 1.0] | 0.3913 |
| 1 | [0.0, 0.513, 0.487] | 0.1217 |
| 2 | [0.0, 0.35346, 0.64654] | 0.01177 |
| 3 | [0.0, 0.40307, 0.59693] | 0.00114 |
| 4 | [0.0, 0.38764, 0.61236] | 0.00011 |
| 5 | [0.0, 0.39244, 0.60756] | 0.00001 |
| 6 | [0.0, 0.39095, 0.60905] | 0.0 |

Задание №4:



**Анализ результатов и выводы**

Стационарное распределение:



Строки матрицы :



(0.00003, 0.39046, 0.60951)



(0.0, 0.39186, 0.60814)

(0.0, 0.39095, 0.60905)



Вывод: с погрешностью 0.00084 значения векторов совпадают

Задание №5:

|  |  |  |  |
| --- | --- | --- | --- |
| *n* | R(1,n) | v(1,n) |  |
| 1 | 1 | 1.0 | 1.0 |
| 2 | 2 | 1.0 | 1.0 |
| 3 | 2 | 0.66667 | 0.66667 |
| 4 | 2 | 0.5 | 0.5 |
| 5 | 2 | 0.4 | 0.4 |
| 6 | 2 | 0.33333 | 0.33333 |
| 7 | 2 | 0.28571 | 0.28571 |
| 8 | 2 | 0.25 | 0.25 |
| 9 | 2 | 0.22222 | 0.22222 |
| 10 | 2 | 0.2 | 0.2 |
| … | … | … | … |
| 1996 | 2 | 0.001 | 0.001 |
| 1997 | 2 | 0.001 | 0.001 |
| 1998 | 2 | 0.001 | 0.001 |
| 1999 | 2 | 0.001 | 0.001 |
| 2000 | 2 | 0.001 | 0.001 |
| 2001 | 2 | 0.001 | 0.001 |

|  |  |  |  |
| --- | --- | --- | --- |
| *n* | R(2,n) | v(2,n) |  |
| 1 | 0 | 0.0 | 0.3913 |
| 2 | 0 | 0.0 | 0.3913 |
| 3 | 1 | 0.33333 | 0.05797 |
| 4 | 1 | 0.25 | 0.1413 |
| 5 | 2 | 0.4 | 0.0087 |
| 6 | 2 | 0.33333 | 0.05797 |
| 7 | 2 | 0.28571 | 0.10559 |
| 8 | 3 | 0.375 | 0.0163 |
| 9 | 4 | 0.44444 | 0.05314 |
| 10 | 4 | 0.4 | 0.0087 |
| … | … | … | … |
| 4268 | 1675 | 0.39246 | 0.00116 |
| 4269 | 1676 | 0.3926 | 0.0013 |
| 4270 | 1676 | 0.39251 | 0.00121 |
| 4271 | 1676 | 0.39241 | 0.00111 |
| 4272 | 1676 | 0.39232 | 0.00102 |
| 4273 | 1676 | 0.39223 | 0.00093 |

|  |  |  |  |
| --- | --- | --- | --- |
| *n* | R(3,n) | v(3,n) |  |
| 1 | 1 | 1.0 | 0.3913 |
| 2 | 1 | 0.5 | 0.1087 |
| 3 | 2 | 0.66667 | 0.05797 |
| 4 | 2 | 0.5 | 0.1087 |
| 5 | 2 | 0.4 | 0.2087 |
| 6 | 3 | 0.5 | 0.1087 |
| 7 | 3 | 0.42857 | 0.18013 |
| 8 | 4 | 0.5 | 0.1087 |
| 9 | 5 | 0.55556 | 0.05314 |
| 10 | 5 | 0.5 | 0.1087 |
| … | … | … | … |
| 184 | 111 | 0.60326 | 0.00544 |
| 185 | 112 | 0.60541 | 0.00329 |
| 186 | 112 | 0.60215 | 0.00655 |
| 187 | 113 | 0.60428 | 0.00442 |
| 188 | 114 | 0.60638 | 0.00232 |
| 189 | 115 | 0.60847 | 0.00023 |

**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')