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

ИНСТИТУТ КИБЕРНЕТИКИ

КАФЕДРА ВЫСШЕЙ МАТЕМАТИКИ

Лабораторная работа 2

 по курсу «**Случайные процессы**»

Тема: **Марковский процесс с непрерывным временем и пятью состояниями**

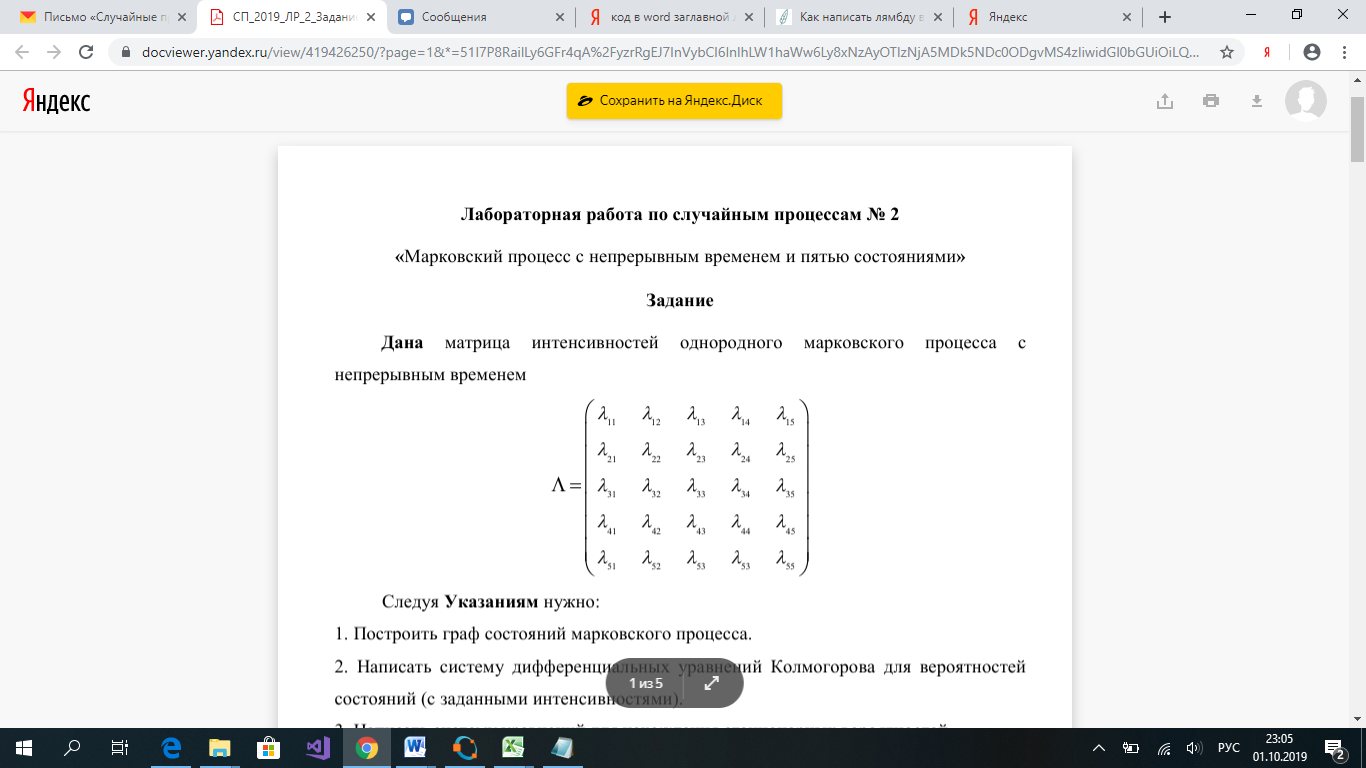
Выполнил:

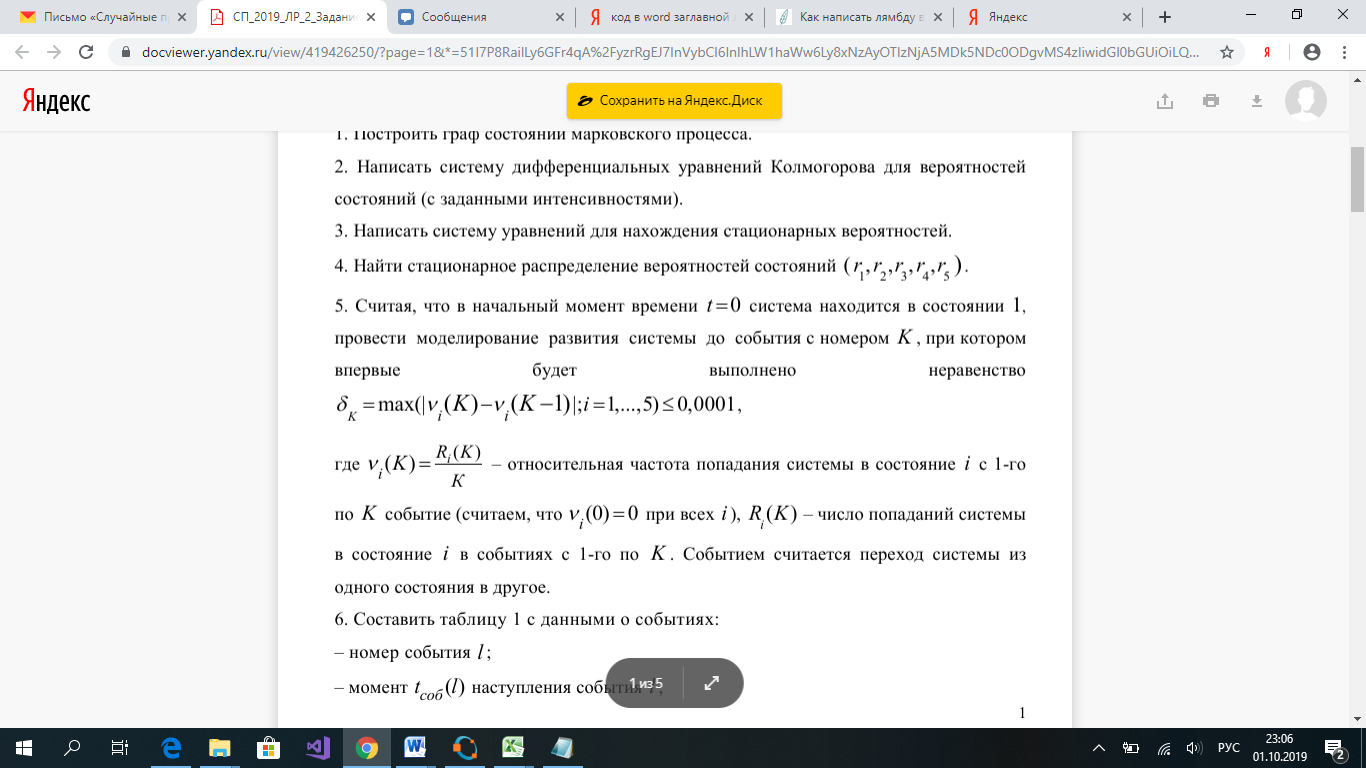
Студент 4-го курса

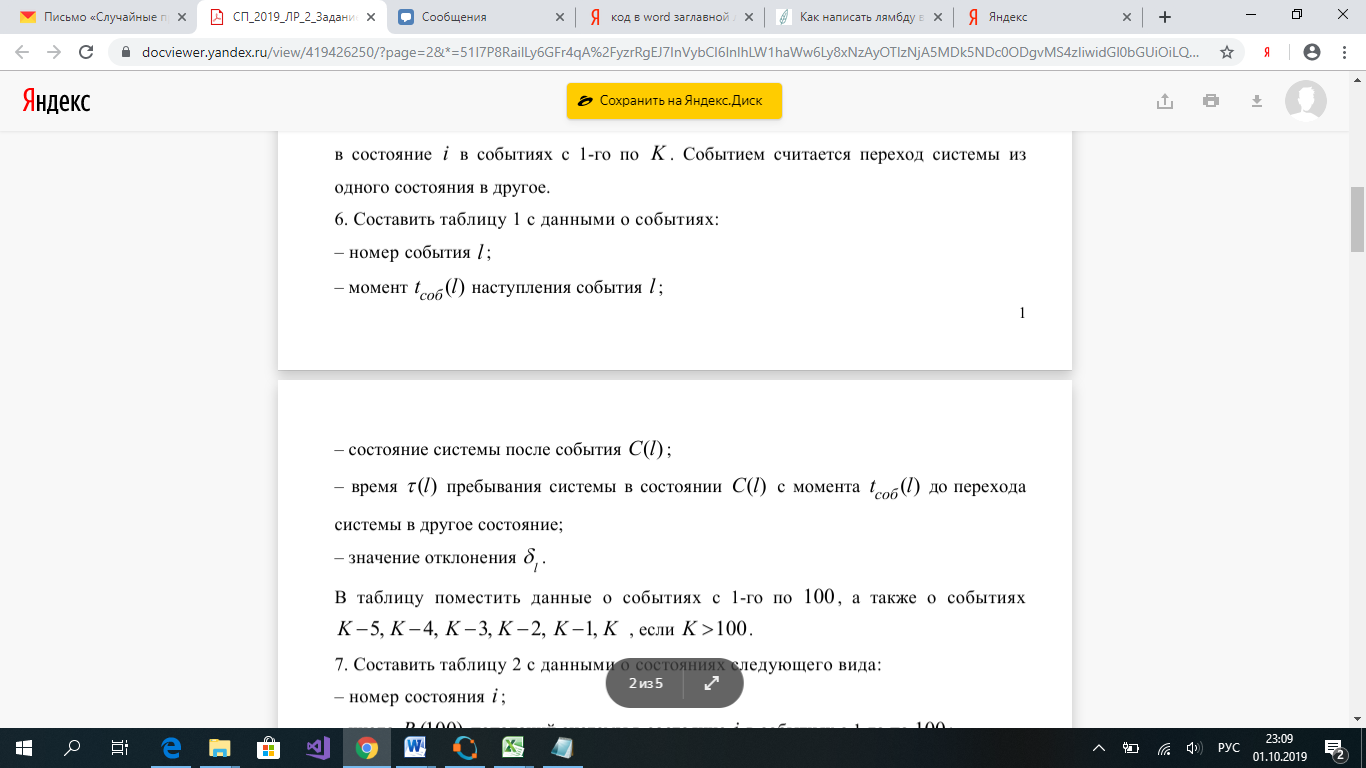
Жолковский Д.А.

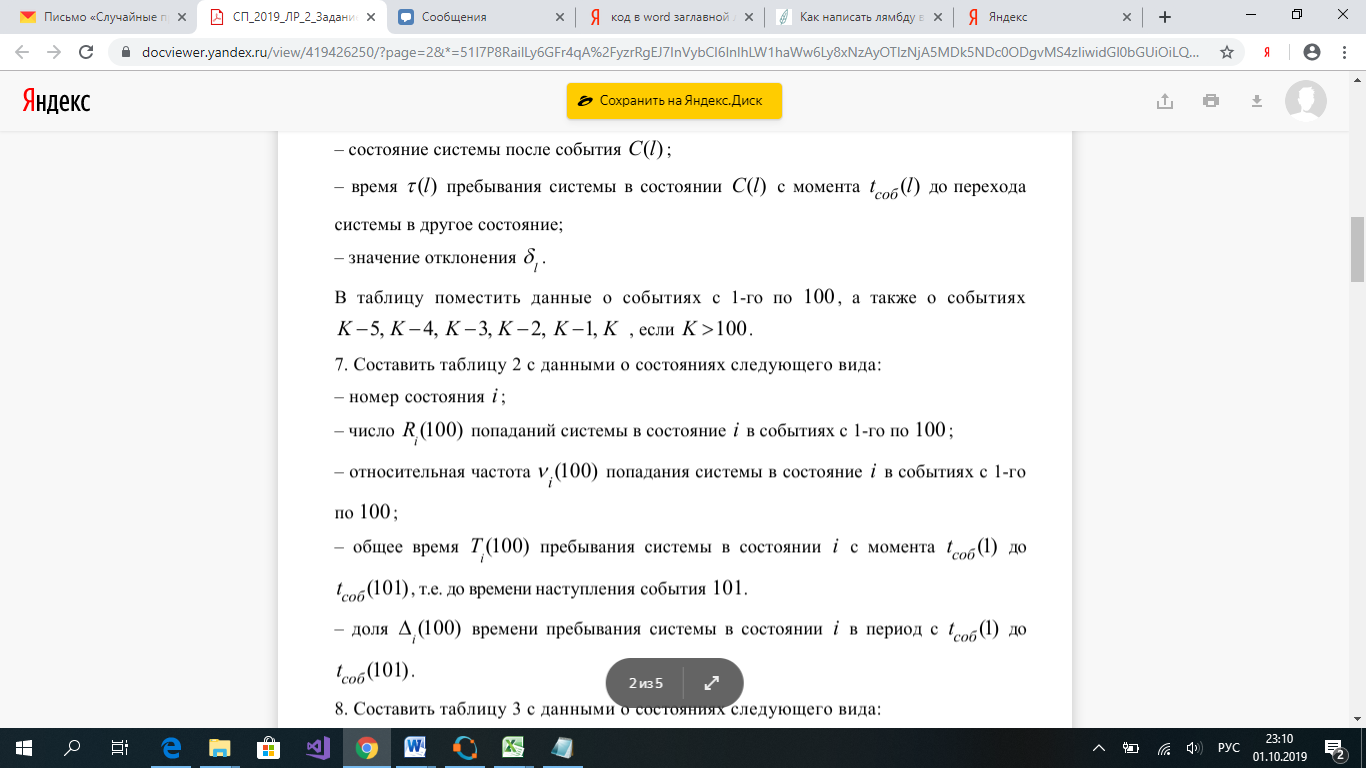
Группа: КМБО-01-16

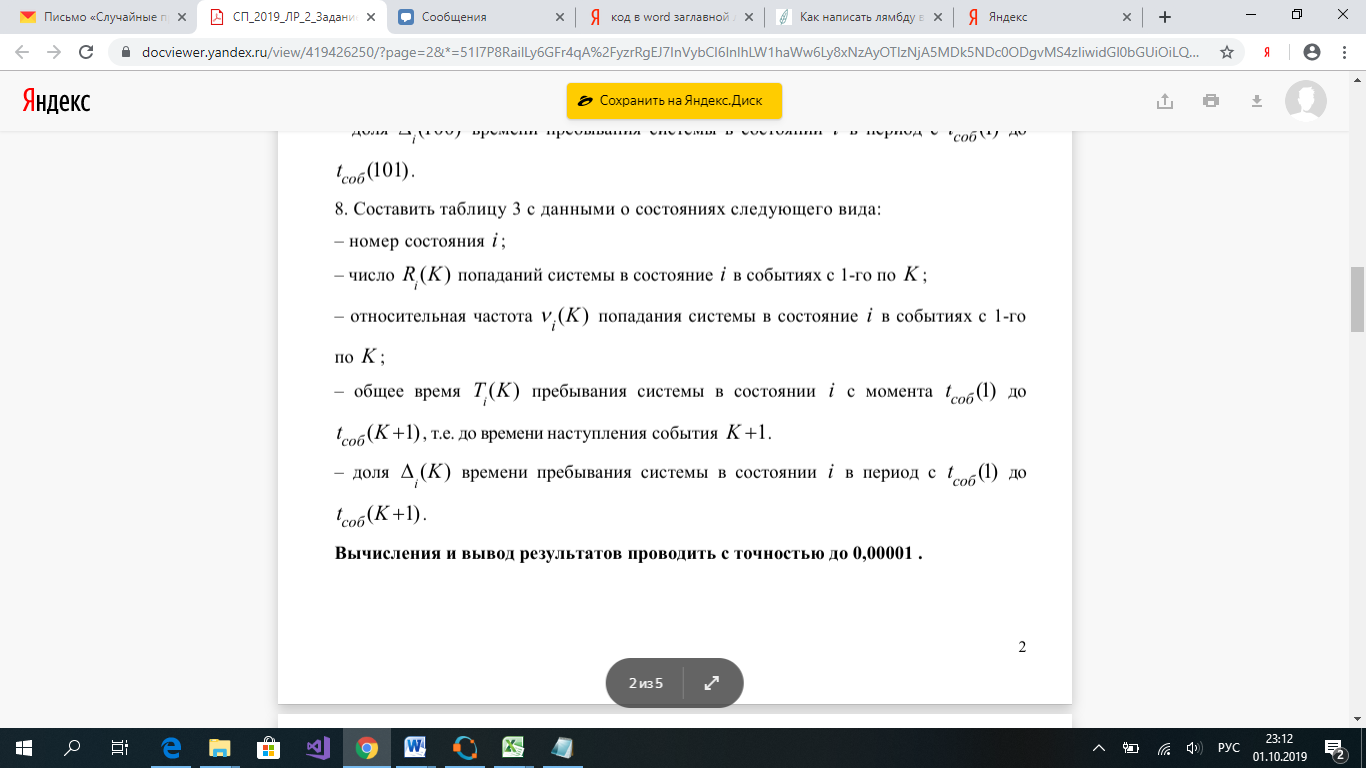
МОСКВА 2019











**Краткие теоретические сведения**

Случайный процесс называется марковским, если для любого целого неотрицательного m, любых моментов времени , любого набора состояний выполнено равенство



Процесс называется однородным (по времени), если условная вероятность перехода из состояния в состояние не зависит от s.



.



- матрица вероятностей перехода за время t.



Предполагаем, что переходные вероятности дифференцируемы в нуле:, при



- матрица интенсивностей (плотность вероятностей) перехода.



Ориентированный граф состояний, вершины которого (обозначаемых прямоугольниками) служат состояния системы, а стрелками обозначены возможные непосредственные переходы из состояния в состояние При этом каждой стрелке приписана соответствующая плотность вероятности перехода.



Пределы, если они существуют, называются *предельными* (или финитными) вероятностями состояний.



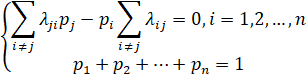
Распределение вероятностей состояний, которое не зависит от времени для любых и любых *i*=1,2,… называется стационарным.



дифференциальные уравнения Колмогорова для вероятностей состояний:



Формулы для нахождения стационарного распределения:



**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)

math.log – натуральный логарифм.

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

random.uniform – рандомное число в заданном диапазоне.

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

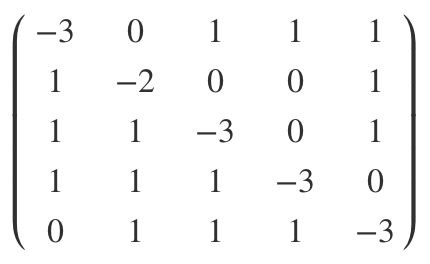
.tolist – метод преобразования матрицы в список

enumerate – проход по итерируемому объекту с перечислением индексов его элементов

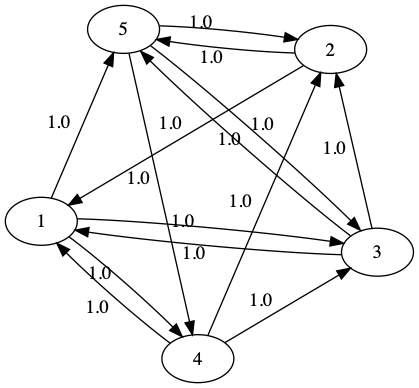
**Результаты расчетов**

Вариант №10

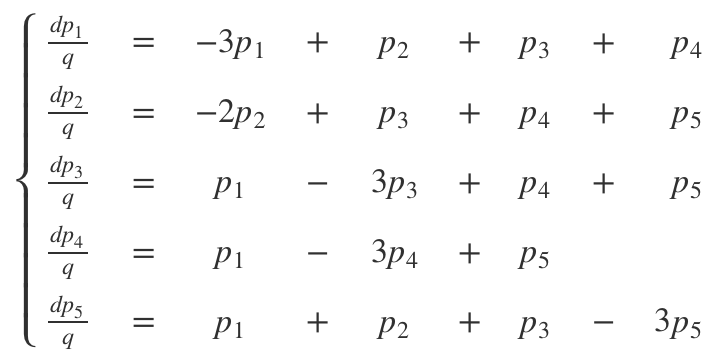
Исходные данные:



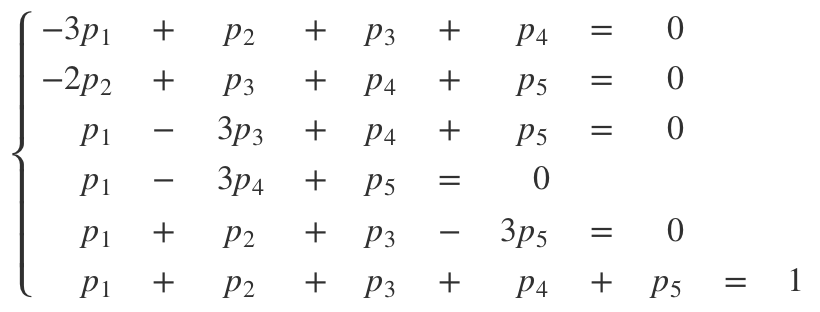
1 Построить граф состояний марковского процесса.



2 Написать систему дифференциальных уравнений Колмогорова для вероятностей состояний (с заданными интенсивностями).



3 Написать систему уравнений для нахождения стационарных вероятностей.



3 Найти стационарное распределение вероятностей состояний.

(, , , , ) = 



5.

K = 7536

6 Таблица 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *l* |  |  |  |  |
| 1 | 0.10426 | 4 | 0.10426 | 1.0 |
| 2 | 0.2008 | 1 | 0.09655 | 0.5 |
| 3 | 1.23922 | 3 | 1.03842 | 0.33333 |
| 4 | 1.57812 | 2 | 0.33891 | 0.25 |
| 5 | 1.76707 | 5 | 0.18895 | 0.2 |
| 6 | 2.6551 | 2 | 0.88803 | 0.13333 |
| 7 | 2.67188 | 1 | 0.01677 | 0.11905 |
| 8 | 2.68582 | 5 | 0.01394 | 0.10714 |
| 9 | 3.11975 | 4 | 0.43393 | 0.09722 |
| 10 | 4.27323 | 2 | 1.15348 | 0.07778 |
| 11 | 5.22844 | 1 | 0.95522 | 0.07273 |
| 12 | 5.40541 | 3 | 0.17697 | 0.07576 |
| 13 | 5.79365 | 1 | 0.38824 | 0.05769 |
| 14 | 6.28093 | 4 | 0.48727 | 0.06044 |
| 15 | 7.36766 | 1 | 1.08673 | 0.04762 |
| 16 | 7.44824 | 4 | 0.08058 | 0.05 |
| 17 | 7.49037 | 1 | 0.04213 | 0.04044 |
| 18 | 7.98521 | 3 | 0.49484 | 0.04902 |
| 19 | 8.29031 | 2 | 0.3051 | 0.04386 |
| 20 | 8.29682 | 1 | 0.00651 | 0.03421 |
| 21 | 8.45618 | 4 | 0.15937 | 0.0381 |
| 22 | 8.55346 | 1 | 0.09728 | 0.0303 |
| 23 | 8.75084 | 3 | 0.19738 | 0.03755 |
| 24 | 9.13865 | 5 | 0.38781 | 0.03804 |
| 25 | 9.16094 | 3 | 0.02229 | 0.03333 |
| 26 | 9.23959 | 1 | 0.07864 | 0.02615 |
| 27 | 9.25729 | 3 | 0.0177 | 0.02991 |
| 28 | 9.61411 | 1 | 0.35682 | 0.02381 |
| 29 | 9.99357 | 4 | 0.37946 | 0.02833 |
| 30 | 10.26583 | 2 | 0.27226 | 0.02874 |
| 31 | 10.58967 | 1 | 0.32384 | 0.02151 |
| 32 | 10.96439 | 3 | 0.37472 | 0.0252 |
| 33 | 11.10245 | 5 | 0.13806 | 0.02746 |
| 34 | 11.33324 | 4 | 0.23078 | 0.02406 |
| 35 | 11.86942 | 1 | 0.53619 | 0.01933 |
| 36 | 12.03478 | 3 | 0.16536 | 0.02222 |
| 37 | 12.23818 | 2 | 0.20339 | 0.02327 |
| 38 | 13.05795 | 1 | 0.81978 | 0.01778 |
| 39 | 13.4287 | 3 | 0.37075 | 0.02024 |
| 40 | 13.46175 | 5 | 0.03305 | 0.02244 |
| 41 | 14.56527 | 4 | 1.10352 | 0.02012 |
| 42 | 14.90506 | 1 | 0.33979 | 0.01626 |
| 43 | 14.9943 | 3 | 0.08924 | 0.01827 |
| 44 | 15.06153 | 5 | 0.06723 | 0.02008 |
| 45 | 15.29744 | 2 | 0.23591 | 0.01919 |
| 46 | 15.57072 | 1 | 0.27328 | 0.01498 |
| 47 | 15.75318 | 5 | 0.18246 | 0.0185 |
| 48 | 15.83652 | 4 | 0.08334 | 0.01729 |
| 49 | 15.95432 | 3 | 0.1178 | 0.01616 |
| 50 | 16.07373 | 5 | 0.11941 | 0.01714 |
| 51 | 16.11317 | 3 | 0.03944 | 0.01529 |
| 52 | 16.26931 | 2 | 0.15614 | 0.01659 |
| 53 | 16.7008 | 1 | 0.43149 | 0.01343 |
| 54 | 16.73736 | 5 | 0.03656 | 0.01572 |
| 55 | 16.79465 | 4 | 0.05728 | 0.01515 |
| 56 | 17.44621 | 2 | 0.65156 | 0.01526 |
| 57 | 18.36836 | 1 | 0.92216 | 0.01253 |
| 58 | 18.52733 | 5 | 0.15897 | 0.01452 |
| 59 | 18.66129 | 3 | 0.13396 | 0.01344 |
| 60 | 18.71636 | 2 | 0.05507 | 0.01412 |
| 61 | 19.15755 | 5 | 0.44119 | 0.01366 |
| 62 | 19.26242 | 3 | 0.10487 | 0.01269 |
| 63 | 19.46152 | 5 | 0.1991 | 0.01306 |
| 64 | 19.92669 | 3 | 0.46517 | 0.01215 |
| 65 | 20.34197 | 1 | 0.41528 | 0.0113 |
| 66 | 20.45863 | 5 | 0.11666 | 0.01235 |
| 67 | 20.67491 | 2 | 0.21628 | 0.01266 |
| 68 | 21.55426 | 5 | 0.87935 | 0.01185 |
| 69 | 22.09469 | 3 | 0.54043 | 0.0113 |
| 70 | 22.33453 | 1 | 0.23984 | 0.01056 |
| 71 | 22.72426 | 4 | 0.38972 | 0.01207 |
| 72 | 22.75547 | 1 | 0.03121 | 0.01017 |
| 73 | 22.79879 | 4 | 0.04332 | 0.01161 |
| 74 | 23.68042 | 1 | 0.88163 | 0.00981 |
| 75 | 23.70738 | 4 | 0.02696 | 0.01117 |
| 76 | 23.80737 | 1 | 0.09999 | 0.00947 |
| 77 | 24.47661 | 4 | 0.66924 | 0.01077 |
| 78 | 24.77537 | 1 | 0.29876 | 0.00916 |
| 79 | 24.8052 | 4 | 0.02984 | 0.01039 |
| 80 | 25.04211 | 2 | 0.2369 | 0.01076 |
| 81 | 25.50438 | 5 | 0.46227 | 0.01019 |
| 82 | 25.67298 | 4 | 0.1686 | 0.00994 |
| 83 | 25.67748 | 1 | 0.0045 | 0.00867 |
| 84 | 26.23584 | 4 | 0.55836 | 0.00961 |
| 85 | 26.37587 | 3 | 0.14003 | 0.00952 |
| 86 | 26.61812 | 1 | 0.24224 | 0.00834 |
| 87 | 28.63848 | 4 | 2.02037 | 0.00922 |
| 88 | 29.19838 | 1 | 0.55989 | 0.0081 |
| 89 | 29.24356 | 4 | 0.04518 | 0.00894 |
| 90 | 29.25479 | 2 | 0.01123 | 0.00961 |
| 91 | 30.65169 | 5 | 1.3969 | 0.00916 |
| 92 | 31.06137 | 4 | 0.40968 | 0.0086 |
| 93 | 32.03611 | 2 | 0.97473 | 0.00923 |
| 94 | 32.03685 | 5 | 0.00074 | 0.00881 |
| 95 | 32.64346 | 4 | 0.60661 | 0.00829 |
| 96 | 32.88051 | 3 | 0.23705 | 0.00855 |
| 97 | 33.06953 | 2 | 0.18902 | 0.00881 |
| 98 | 33.10747 | 1 | 0.03794 | 0.00747 |
| 99 | 33.21964 | 5 | 0.11218 | 0.00835 |
| 100 | 33.33982 | 2 | 0.12017 | 0.00848 |
| … | … | … | … | … |
| 7531 | 2732.24217 | 2 | 0.10237 | 0.00011 |
| 7532 | 2734.54975 | 5 | 2.30758 | 0.0001 |
| 7533 | 2735.02643 | 2 | 0.47668 | 0.00011 |
| 7534 | 2735.12814 | 1 | 0.10171 | 0.0001 |
| 7535 | 2735.50864 | 3 | 0.3805 | 0.00011 |
| 7536 | 2736.49479 | 5 | 0.98615 | 0.0001 |

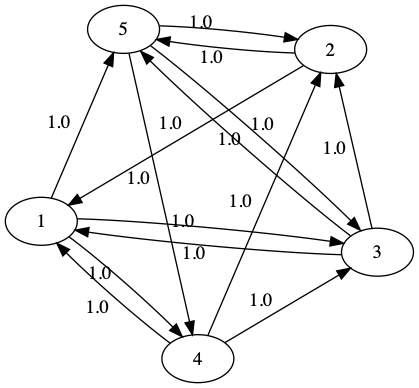
7 Таблица 2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *i* |  |  |  |  |
| 1 | 27 | 0.27 | 417.81534 | 0.25003 |
| 2 | 16 | 0.16 | 280.44736 | 0.16783 |
| 3 | 18 | 0.18 | 264.49004 | 0.15828 |
| 4 | 21 | 0.21 | 385.94054 | 0.23096 |
| 5 | 18 | 0.18 | 322.35339 | 0.19291 |

8 Таблица 3.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *i* |  |  |  |  |
| 1 | 1586 | 0.21046 | 2133529.99512 | 0.20982 |
| 2 | 1465 | 0.1944 | 1940723.53619 | 0.19086 |
| 3 | 1506 | 0.19984 | 2056189.17699 | 0.20221 |
| 4 | 1121 | 0.14875 | 1519501.19484 | 0.14943 |
| 5 | 1858 | 0.24655 | 2518412.73536 | 0.24767 |

**Анализ результатов и выводы**



K=7536

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *i* | 1 | 2 | 3 | 4 | 5 |
|  | 0.19608 | 0.26797 | 0.18301 | 0.13725 | 0.21569 |
|  | 0.27 | 0.16 | 0.18 | 0.21 | 0.18 |
|  | 0.21046 | 0.1944 | 0.19984 | 0.14875 | 0.24655 |
|  | 0.25003 | 0.16783 | 0.15828 | 0.23096 | 0.19291 |
|  | 0.20982 | 0.19086 | 0.20221 | 0.14943 | 0.24767 |

**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**

**from** **IPython.display** **import** Latex

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

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

**from** **random** **import** choices, uniform

**from** **math** **import** log

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

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

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

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

**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

# ### Методы

# 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)

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

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

**else**:

res = []

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

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

**return** tuple(res)

**return** rounder

np.set\_printoptions(precision=**5**, suppress=True) # formatter={'float': '{: 0.5f}'.format})

pd.options.display.float\_format = '{:,.5f}'.format

**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')

**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')

**def** **matrix2latex**(matr, toint=False):

start = r'$$\begin{pmatrix} '

end = r' \end{pmatrix}$$'

**if** **not** toint:

body = r' **\\** '.join([r' & '.join([str(x) **for** x **in** matr[i]]) **for** i **in** range(matr.shape[**0**])])

**else**:

body = r' **\\** '.join([r' & '.join([str(int(x)) **for** x **in** matr[i]]) **for** i **in** range(matr.shape[**0**])])

**return** start + body + end

**def** **ode\_system2latex**(P, toint=False):

matr = P.T

start = r'\[\left\{\begin{array}{ r @{{}={}} r >{{}}c<{{}} r >{{}}c<{{}} r } '

end = r' \end{array}\right.\]'

frac = **lambda** i: r'\frac{dp\_' + r'{0}'.format(i+**1**) + r'}{q}' + r'&=&'

check\_first = **lambda** arr, j: j == [i **for** i, x **in** enumerate(arr) **if** x != **0**][**0**]

**def** **element**(k, j, is\_first):

**if** **not** is\_first:

st = r'&+&' **if** k > **0** **else** r'&-&'

**else**:

st = r''

**if** **not** toint:

**if** abs(k) != **1**:

bd = r'{0}'.format(abs(k)) **if** **not** is\_first **else** r'{0}'.format(k)

**else**:

bd = r''

**if** is\_first:

bd = r'' **if** k > **0** **else** r'-'

**else**:

**if** abs(k) != **1**:

bd = r'{0}'.format(abs(int(k))) **if** **not** is\_first **else** r'{0}'.format(int(k))

**else**:

bd = r''

**if** is\_first:

bd = r'' **if** k > **0** **else** r'-'

nd = r'p\_{0}'.format(j+**1**)

**return** st + bd + nd

body = [frac(i) + r' '.join([element(matr[i][j], j, check\_first(matr[i], j)) **for** j **in** range(matr.shape[**1**]) **if** matr[i][j] != **0**]) + r'**\\**'

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

**return** start + r' '.join(body) + end

**def** **stationary\_system2latex**(P, toint=False):

matr = P.T

start = r'\[\left\{\begin{array}{ r @{{}={}} r >{{}}c<{{}} r >{{}}c<{{}} r } '

end = r' \end{array}\right.\]'

check\_first = **lambda** arr, j: j == [i **for** i, x **in** enumerate(arr) **if** x != **0**][**0**]

**def** **element**(k, j, is\_first):

**if** **not** is\_first:

st = r'&+&' **if** k > **0** **else** r'&-&'

**else**:

st = r''

**if** **not** toint:

**if** abs(k) != **1**:

bd = r'{0}'.format(abs(k)) **if** **not** is\_first **else** r'{0}'.format(k)

**else**:

bd = r''

**if** is\_first:

bd = r'' **if** k > **0** **else** r'-'

**else**:

**if** abs(k) != **1**:

bd = r'{0}'.format(abs(int(k))) **if** **not** is\_first **else** r'{0}'.format(int(k))

**else**:

bd = r''

**if** is\_first:

bd = r'' **if** k > **0** **else** r'-'

nd = r'p\_{0}'.format(j+**1**)

**return** st + bd + nd

body = [r' '.join([element(matr[i][j], j, check\_first(matr[i], j)) **for** j **in** range(matr.shape[**1**]) **if** matr[i][j] != **0**]) + r' &=& 0' + r'**\\**'

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

body += [r' '.join([element(**1.0**, j, check\_first(np.ones(matr.shape[**1**]), j)) **for** j **in** range(matr.shape[**1**])]) + r' &=& 1' + r'**\\**']

**return** start + r' '.join(body) + end

**@round\_output**

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

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

B = np.zeros(P.shape[**0**])

B[-**1**] = **1.**

x = solve(A, B)

**return** x

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

**return** choices(population=range(**5**), weights=P[current\_state\_id])[**0**]

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

weights = [(i, (-**1.** / x) \* log(uniform(**1e-10**, **1.**)))

**for** i, x **in** enumerate(P[current\_state\_id]) **if** x > **0.**]

**return** sorted(weights, key=**lambda** x: x[**1**])[**0**]

**@round\_output**

**def** **generate\_n\_states**(P, stationary, dm=**1e-4**):

matr = [[x **if** x > **0** **else** **0.** **for** x **in** P[i]] **for** i **in** range(P.shape[**0**])]

matr = [[x / sum(matr[i]) **for** x **in** matr[i]] **for** i **in** range(P.shape[**0**])]

matr = np.array(matr)

current\_state\_id = **0**

prev\_state\_id = current\_state\_id

d = **1.**

t = **0**

K = **0**

v = [np.zeros(**5**)]

r = [np.zeros(**5**)]

tkidq = [[**0**, **0**, **0**, **0**, **0**]]

**while** d > dm:

prev\_state\_id = current\_state\_id

current\_state\_id, cur\_q = generate\_time\_and\_next\_state(current\_state\_id, P)

t += cur\_q

**if** current\_state\_id == prev\_state\_id:

**print**('cur = prev')

**continue**

K += **1**

cur\_r = copy(r[-**1**])

cur\_r[current\_state\_id] = cur\_r[current\_state\_id] + **1**

r.append(cur\_r)

cur\_v = copy(v[-**1**])

cur\_v = cur\_r / K #(tkid[-1][1]+1)

v.append(cur\_v)

d = max(v[-**1**] - v[-**2**])

tkidq.append([t, K, current\_state\_id+**1**, d, cur\_q])

**return** (np.array(tkidq), np.array(r), np.array(v))

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

parser = createParser()

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

# ### 0) Подготовка

P = np.genfromtxt(namespace.path, comments="%")

display(Latex(matrix2latex(P, True)))

# ### 1) Построить граф

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

# ### 2) СДУ Колмогорова

display(Latex(ode\_system2latex(P, True)))

# ### 3) СУ для стационарных вероятностей

display(Latex(stationary\_system2latex(P, True)))

# ### 4) Стационарное распределение

stationary = get\_stationary\_d(P)

display(Latex(matrix2latex(stationary.reshape(**1**, -**1**))))

# ### 5) Генерация

tkidq, R, v = generate\_n\_states(P, stationary)

**print**(f'K = {int(tkidq[-1][1])}')

# ### 6) Таблица 1

table = np.concatenate(

(np.array([tkidq[**1**:**101**, **1**], tkidq[**1**:**101**, **0**], tkidq[**1**:**101**, **2**], tkidq[**1**:**101**, **4**], tkidq[**1**:**101**, **3**]]).T,

np.array([tkidq[-**6**:, **1**], tkidq[-**6**:, **0**], tkidq[-**6**:, **2**], tkidq[-**6**:, **4**], tkidq[-**6**:, **3**]]).T)

)

table1 = pd.DataFrame(table, columns=['l', 't', 'C', 'tau', 'delta'])

table1.l = table1.l.astype(int)

table1.C = table1.C.astype(int)

table1 = table1.set\_index('l')

table1.to\_csv('Data/table1.csv', encoding='utf-8', sep=';')

# ### 7) Таблица 2

table = []

**for** i **in** range(**5**):

table.append([i+**1**, R[**100**][i], v[**100**][i],

sum([t **for** t, \_, state, \_, \_ **in** tkidq[**1**:**101**] **if** state-**1** == i]),

sum([t **for** t, \_, state, \_, \_ **in** tkidq[**1**:**101**] **if** state-**1** == i]) / sum(tkidq[**1**:**101**, **0**])

])

table = np.round(np.array(table), **5**)

table2 = pd.DataFrame(table, columns=['i', 'R', 'v', 'T', 'delta'])

table2.i = table2.i.astype(int)

table2.R = table2.R.astype(int)

table2 = table2.set\_index('i')

table2.to\_csv('Data/table2.csv', encoding='utf-8', sep=';')

# ### 8) Таблица 3

table = []

**for** i **in** range(**5**):

table.append([i+**1**, R[-**1**][i], v[-**1**][i],

sum([t **for** t, \_, state, \_, \_ **in** tkidq[**1**:] **if** state-**1** == i]),

sum([t **for** t, \_, state, \_, \_ **in** tkidq[**1**:] **if** state-**1** == i]) / sum(tkidq[**1**:, **0**])

])

table = np.round(np.array(table), **5**)

table3 = pd.DataFrame(table, columns=['i', 'R', 'v', 'T', 'delta'])

table3.i = table3.i.astype(int)

table3.R = table3.R.astype(int)

table3 = table3.set\_index('i')

table3.to\_csv('Data/table3.csv', encoding='utf-8', sep=';')

# ### Анализ

table = []

**for** i **in** range(**5**):

table.append([stationary[i], v[**100**][i], v[-**1**][i],

sum([t **for** t, \_, state, \_, \_ **in** tkidq[**1**:**101**] **if** state-**1** == i]) / sum(tkidq[**1**:**101**, **0**]),

sum([t **for** t, \_, state, \_, \_ **in** tkidq[**1**:] **if** state-**1** == i]) / sum(tkidq[**1**:, **0**])

])

table = np.round(np.array(table).T, **5**)

table4 = pd.DataFrame(table, columns=['1', '2', '3', '4', '5'])

table4.to\_csv('Data/table4.csv', encoding='utf-8', sep=';')