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

Лабораторная работа по случайным процессам № 1

«Однородная цепь Маркова с 3-мя состояниями»

Дана матрица переходных вероятностей однородной цепи Маркова

$$P = \begin{pmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{pmatrix}$$

Построить граф состояний цепи Маркова.

Найти:

1. Матрицы переходных вероятностей за n шагов P^n $(n = 2,..., n_{\min})$ и величины отклонений $\delta_n = \max(|p_{ij}(n) - p_{ij}(n-1)|; i, j = 1,2,3)$ (для $n = 2,..., n_{\min}$), где $n_{\min} = \min(n \,|\, \delta_n < 0{,}00001)$. Результаты представить в табличной форме:

n	P^n	δ_n
1	$\begin{pmatrix} p_{11}(1) & p_{12}(1) & p_{13}(1) \\ p_{21}(1) & p_{22}(1) & p_{23}(1) \\ p_{31}(1) & p_{32}(1) & p_{33}(1) \end{pmatrix}$	–
2	$\begin{pmatrix} p_{11}(2) & p_{12}(2) & p_{13}(2) \\ p_{21}(2) & p_{22}(2) & p_{23}(2) \\ p_{31}(2) & p_{32}(2) & p_{33}(2) \end{pmatrix}$	δ_2
3	$\begin{pmatrix} p_{11}(3) & p_{12}(3) & p_{13}(3) \\ p_{21}(3) & p_{22}(3) & p_{23}(3) \\ p_{31}(3) & p_{32}(3) & p_{33}(3) \end{pmatrix}$	δ_3
…	…	…
k	$\begin{pmatrix} p_{11}(k) & p_{12}(k) & p_{13}(k) \\ p_{21}(k) & p_{22}(k) & p_{23}(k) \\ p_{31}(k) & p_{32}(k) & p_{33}(k) \end{pmatrix}$	δ_k

где $k = n_{\min}$.

2. Стационарное распределение вероятностей состояний (r_1, r_2, r_3). Провести проверку стационарности найденного распределения.

3. Распределения вероятностей состояний через n шагов $(p_1(n), p_2(n), p_3(n))$ (для $n=1,\ldots, m_{\min}$) и величины отклонений $\delta_n = \max(|p_i(n)-r_i|; i=1,2,3)$ (для $n=1,\ldots, m_{\min}$), где $m_{\min} = \min(n \mid \delta_n < 0{,}00001)$, для следующих начальных распределений: $(1,0,0)$, $(0,1,0)$ и $(0,0,1)$. Результаты представить в табличной форме:

n	$(p_1(n), p_2(n), p_3(n))$	δ_n
0	$(1,0,0)$	$\delta_0 = \max((1-r_1), r_2, r_3)$
1		δ_1
2		δ_2
…		…
k		δ_k

где $k = m_{\min}$. Аналогично для $(0,1,0)$ и $(0,0,1)$.

4. Для каждого состояния $i=1,\ 2,\ 3$, взятого в качестве начального $i=i_0$ провести в соответствии с матрицей переходных вероятностей генерацию последовательности номеров состояний $i_1,\ldots,i_n$ через n шагов, определяя для каждого n значения $R(i,n) = |\{i_k = i;\ k=1,\ldots,n\}|$ (число возвратов в состояние i) и $\nu(i,n) = \dfrac{R(i,n)}{n}$ (частота возвратов в состояние i). Генерацию проводить до шага $N_{\min}(i) = \min(n \mid \Delta_n(i) < 0{,}001)$, где $\Delta_n(i) = |\nu(i,n) - r_i|$.
В отчете привести значения $N_{\min}(i)$, $i=1,\ 2,\ 3$.

5. По результатам пункта 4 для каждого начального состояния $i=1,\ 2,\ 3$ построить таблицы вида

n	$R(i,n)$	$\nu(i,n)$	$\Delta_n(i)$
1			
2			
…			
10			
$k-5$			
…			
k			

где $k = \max(16, N_{\min}(i))$.

Вычисления и вывод результатов проводить с точностью до 0,00001 .

Краткие теоретические сведения

Рассмотрим последовательность случайных величин (с.в.) $\{X_n\}_{n=0}^{\infty}$, принимающих значения E_1, E_2, ...

Последовательность с.в. $\{X_n\}_{n=0}^{\infty}$ называется цепью Маркова, если для произвольного набора $i_1 < i_2 < i_3 < \dots < i_k$ $(k = 3, 4, \dots)$ и любых $E_{j_1}, \dots, E_{j_k}$ справедливо равенство
$P(X_{i_k} = E_{j_k} | X_{i_1} = E_{j_1}, \dots, X_{i_{k-1}} = E_{j_{k-1}}) = P(X_{i_k} = E_{j_k} | X_{i_{k-1}} = E_{j_{k-1}})$.

Цепь Маркова $\{X_n\}_{n=0}^{\infty}$ называется однородной, если для всех i и j вероятности $p_{ij} = P(X_{n+1} = E_j | X_n = E_i)$ не зависит от n.

Вероятности p_{ij} называются переходными, а матрица $P = ||p_{ij}||$ - матрицей переходных вероятностей цепи Маркова.

Матрица переходных вероятностей обладает следующими свойствами:

1. $p_{ij} \geq 0$;
2. $\sum_{j=1}^{N} p_{ij} = 1$ для всех $i = 1,2, \dots, N$.

Распределение $\overrightarrow{p^*}$ цепи Маркова называется *стационарным*, если оно останется неизменным на каждом шаге. Стационарное распределение $\overrightarrow{p^*}$ удовлетворяет соотношению $\overrightarrow{p^*} = \overrightarrow{p^*}P$.

Если существует $\lim_{n\to\infty} \vec{p}(n) = \vec{p}(\infty)$ и $\sum_i p_i(\infty) = 1$, то распределение $\vec{p}(\infty)$ называется *предельным*.

Общие формулы для нахождения стационарного распределения:
$\vec{r} = \vec{r}P$

$$(r_1 \quad r_2 \quad r_3)\begin{pmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{pmatrix} = (r_1 \quad r_2 \quad r_3)$$

$$\begin{cases} p_{11}r_1 + p_{21}r_2 + p_{31}r_3 = r_1 \\ p_{12}r_1 + p_{22}r_2 + p_{32}r_3 = r_2 \\ p_{13}r_1 + p_{23}r_2 + p_{33}r_3 = r_3 \end{cases}$$

Данная однородная система имеет ранг два. Добавим ещё одно условие: $r_1 + r_2 + r_3 = 1$. Заменим это условие в системе так, чтоб её ранг стал равен трём. Решением полученной системы будет стационарное распределение $\vec{r}$.

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

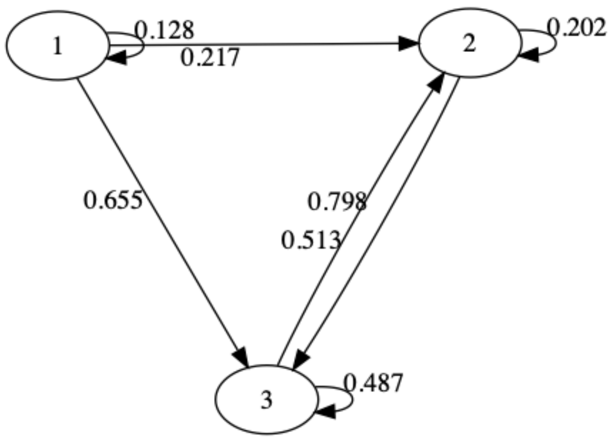
Результаты расчетов

Вариант №10

Исходные данные:

$$\begin{pmatrix} 0.128\ 0.217\ 0.655 \\ 0\ 0.202\ 0.798 \\ 0\ 0.513\ 0.487 \end{pmatrix}$$

Граф состояний цепи Маркова:



Задание №1:

n	P^n	δ_n
1	$\begin{pmatrix} 0.128, 0.217, 0.655 \\ 0.0, 0.202, 0.798 \\ 0.0, 0.513, 0.487 \end{pmatrix}$	-
2	$\begin{pmatrix} 0.01638, 0.40762, 0.57599 \\ 0.0, 0.45018, 0.54982 \\ 0.0, 0.35346, 0.64654 \end{pmatrix}$	0.24818
3	$\begin{pmatrix} 0.0021, 0.38138, 0.61652 \\ 0.0, 0.37299, 0.62701 \\ 0.0, 0.40307, 0.59693 \end{pmatrix}$	0.07718
4	$\begin{pmatrix} 0.00027, 0.39377, 0.60596 \\ 0.0, 0.397, 0.603 \\ 0.0, 0.38764, 0.61236 \end{pmatrix}$	0.024

5	$\begin{pmatrix} 0.00003, 0.39046, 0.60951 \\ 0.0, 0.38953, 0.61047 \\ 0.0, 0.39244, 0.60756 \end{pmatrix}$	0.00747
6	$\begin{pmatrix} 0.0, 0.39156, 0.60844 \\ 0.0, 0.39186, 0.60814 \\ 0.0, 0.39095, 0.60905 \end{pmatrix}$	0.00232
7	$\begin{pmatrix} 0.0, 0.39122, 0.60878 \\ 0.0, 0.39113, 0.60887 \\ 0.0, 0.39141, 0.60859 \end{pmatrix}$	0.00072
8	$\begin{pmatrix} 0.0, 0.39133, 0.60867 \\ 0.0, 0.39136, 0.60864 \\ 0.0, 0.39127, 0.60873 \end{pmatrix}$	0.00022
9	$\begin{pmatrix} 0.0, 0.3913, 0.6087 \\ 0.0, 0.39129, 0.60871 \\ 0.0, 0.39131, 0.60869 \end{pmatrix}$	0.00007
10	$\begin{pmatrix} 0.0, 0.39131, 0.60869 \\ 0.0, 0.39131, 0.60869 \\ 0.0, 0.3913, 0.6087 \end{pmatrix}$	0.00002
11	$\begin{pmatrix} 0.0, 0.3913, 0.6087 \\ 0.0, 0.3913, 0.6087 \\ 0.0, 0.39131, 0.60869 \end{pmatrix}$	0.00001

Задание №2:
Стационарное распределение вероятностей состояний
$(r_1, r_2, r_3) = (0, 0.3913, 0.6087)$

Проверка стационарности найденного распределения:
$\vec{r} = \vec{r}P$

$$(0, 0.3913, 0.6087) \ * \begin{pmatrix} 0.128\ 0.217\ 0.655 \\ 0\ 0.202\ 0.798 \\ 0\ 0.513\ 0.487 \end{pmatrix} = (0, 0.39131, 0.6087)$$

Задание №3:

n	$(p_1(n), p_2(n), p_3(n))$	δ_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	$(p_1(n), p_2(n), p_3(n))$	δ_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	$(p_1(n), p_2(n), p_3(n))$	δ_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:

$N_{min}(1) = 2001$
$N_{min}(2) = 4273$
$N_{min}(3) = 189$

Анализ результатов и выводы

Стационарное распределение: $(0.0, 0.3913, 0.6087)$

Строки матрицы P^{11}:

$$\begin{pmatrix} 0.0, 0.3913, 0.6087 \\ 0.0, 0.3913, 0.6087 \\ 0.0, 0.39131, 0.60869 \end{pmatrix}$$

$(p_1(6), p_2(6), p_3(6)$ $=$(0.00003, 0.39046, 0.60951)
$(p_1(7), p_2(7), p_3(7)$ $=$(0.0, 0.39186, 0.60814)
$(p_1(7), p_2(7), p_3(7)$ $=$(0.0, 0.39095, 0.60905)

Вывод: с погрешностью 0.00084 значения векторов совпадают

Задание №5:

n	R(1,n)	v(1,n)	$\Delta_n(1)$
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)	$\Delta_n(2)$
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)	$\Delta_n(3)$
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):
```

```python
    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'],
```

```python
                                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'\nВывод: с погрешностью {d} значения векторов совпадают')
```