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 $(n=2,\ldots, n_{\min})$ и величины отклонений $\delta_n=\max(|p_{ij}(n)-p_{ij}(n-1)|;i,j=1,2,3)$ (для $n=2,\ldots, n_{\min}$), 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 $n = 1,..., m_{\min}$) и величины отклонений $\delta_n = \max(|p_i(n) - r_i|; i = 1,2,3)$ (для $n = 1,..., m_{\min}$), где $m_{\min} = \min(n | \delta_n < 0{,}00001)$, 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 $i_1,...,i_n$ через n шагов, определяя для каждого n значения $R(i,n) = |\{i_k = i;\ k = 1,...,n\}|$ (число возвратов в состояние i) и $\nu(i,n) = \dfrac{R(i,n)}{n}$ (частота возвратов в состояние i). Генерацию проводить до шага $N_{\min}(i) = \min(n | \Delta_n(i) < 0{,}001)$, 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 $i_1 < i_2 < i_3 < \cdots < i_k$ $(k = 3,4,\dots)$ и любых $E_{j_1},\dots,E_{j_k}$ справедливо равенство
$P\left(X_{i_k} = E_{j_k} \middle| X_{i_1} = E_{j_1},\dots,X_{i_{k-1}} = E_{j_{k-1}}\right) = P\left(X_{i_k} = E_{j_k} \middle| X_{i_{k-1}} = E_{j_{k-1}}\right)$.

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

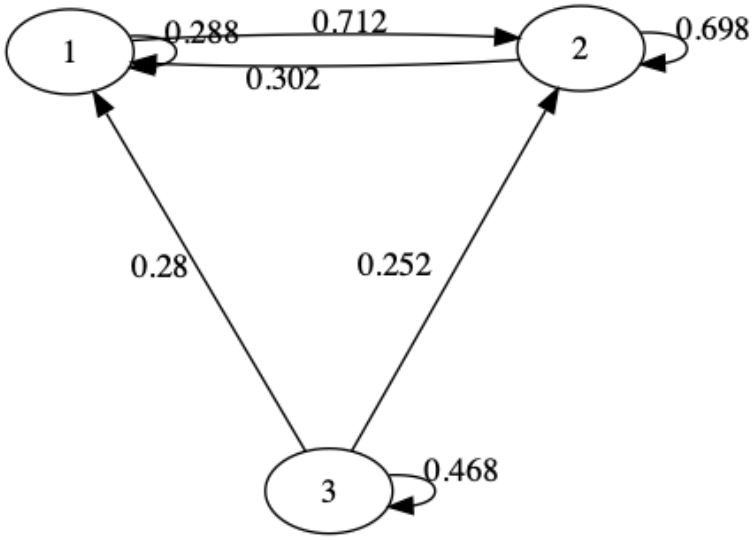
Результаты расчетов

Вариант №9

Исходные данные:

$$\begin{pmatrix} 0.288 & 0.712 & 0 \\ 0.302 & 0.698 & 0 \\ 0.28 & 0.252 & 0.468 \end{pmatrix}$$

Граф состояний цепи Маркова:



Задание №1:

n	P^n	δ_n
1	$\begin{pmatrix} 0.288 & 0.712 & 0.0 \\ 0.302 & 0.698 & 0.0 \\ 0.28 & 0.252 & 0.468 \end{pmatrix}$	-
2	$\begin{pmatrix} 0.29797 & 0.70203 & 0.0 \\ 0.29777 & 0.70223 & 0.0 \\ 0.28778 & 0.49319 & 0.21902 \end{pmatrix}$	0.24898
3	$\begin{pmatrix} 0.29783 & 0.70217 & 0.0 \\ 0.29783 & 0.70217 & 0.0 \\ 0.29315 & 0.60434 & 0.1025 \end{pmatrix}$	0.11652
4	$\begin{pmatrix} 0.29783 & 0.70217 & 0.0 \\ 0.29783 & 0.70217 & 0.0 \\ 0.29564 & 0.65639 & 0.04797 \end{pmatrix}$	0.05453

5	$\begin{pmatrix} 0.29783\ 0.70217\ 0.0 \\ 0.29783\ 0.70217\ 0.0 \\ 0.29681\ 0.68074\ 0.02245 \end{pmatrix}$	0.02552
6	$\begin{pmatrix} 0.29783\ 0.70217\ 0.0 \\ 0.29783\ 0.70217\ 0.0 \\ 0.29735\ 0.69214\ 0.01051 \end{pmatrix}$	0.01194
7	$\begin{pmatrix} 0.29783\ 0.70217\ 0.0 \\ 0.29783\ 0.70217\ 0.0 \\ 0.29761\ 0.69748\ 0.00492 \end{pmatrix}$	0.00559
8	$\begin{pmatrix} 0.29783\ 0.70217\ 0.0 \\ 0.29783\ 0.70217\ 0.0 \\ 0.29773\ 0.69997\ 0.0023 \end{pmatrix}$	0.00262
9	$\begin{pmatrix} 0.29783\ 0.70217\ 0.0 \\ 0.29783\ 0.70217\ 0.0 \\ 0.29778\ 0.70114\ 0.00108 \end{pmatrix}$	0.00122
10	$\begin{pmatrix} 0.29783\ 0.70217\ 0.0 \\ 0.29783\ 0.70217\ 0.0 \\ 0.29781\ 0.70169\ 0.0005 \end{pmatrix}$	0.00057
11	$\begin{pmatrix} 0.29783\ 0.70217\ 0.0 \\ 0.29783\ 0.70217\ 0.0 \\ 0.29782\ 0.70194\ 0.00024 \end{pmatrix}$	0.00027
12	$\begin{pmatrix} 0.29783\ 0.70217\ 0.0 \\ 0.29783\ 0.70217\ 0.0 \\ 0.29783\ 0.70206\ 0.00011 \end{pmatrix}$	0.00013
13	$\begin{pmatrix} 0.29783\ 0.70217\ 0.0 \\ 0.29783\ 0.70217\ 0.0 \\ 0.29783\ 0.70212\ 5e-05 \end{pmatrix}$	6e-05
14	$\begin{pmatrix} 0.29783\ 0.70217\ 0.0 \\ 0.29783\ 0.70217\ 0.0 \\ 0.29783\ 0.70215\ 2e-05 \end{pmatrix}$	3e-05
15	$\begin{pmatrix} 0.29783\ 0.70217\ 0.0 \\ 0.29783\ 0.70217\ 0.0 \\ 0.29783\ 0.70216\ 1e-05 \end{pmatrix}$	1e-05

16	$\begin{pmatrix} 0.29783\ 0.70217\ 0.0 \\ 0.29783\ 0.70217\ 0.0 \\ 0.29783\ 0.70216\ 1e-05 \end{pmatrix}$	1e-05

Задание №2:
Стационарное распределение вероятностей состояний
$(r_1, r_2, r_3) = (0.29783, 0.70217, 0)$

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

$$(0.29783, 0.70217, 0) * \begin{pmatrix} 0.288\ 0.712\ 0 \\ 0.302\ 0.698\ 0 \\ 0.28\ 0.252\ 0.468 \end{pmatrix} = (0.29783, 0.70217, 0)$$

Задание №3:

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

$N_{min}(1) = 64$
$N_{min}(2) = 275$
$N_{min}(3) = 1001$

Анализ результатов и выводы

Стационарное распределение: $(0.29783, 0.70217, 0)$

Строки матрицы P^{16}:

$$\begin{pmatrix} 0.29783\ 0.70217\ 0.0 \\ 0.29783\ 0.70217\ 0.0 \\ 0.29783\ 0.70216\ 0.00001 \end{pmatrix}$$

$(p_1(3), p_2(3), p_3(3) = (0.29797, 0.70203, 0.0)$
$(p_1(3), p_2(3), p_3(3) = (0.29777, 0.70223, 0.0)$
$(p_1(10), p_2(10), p_3(10) = (0.29778, 0.70114, 0.00108)$

Вывод: с погрешность 0.00107 значения векторов совпадают.

Задание №5:

n	R(1,n)	v(1,n)	$\Delta_n(1)$
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)	$\Delta_n(2)$
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)	$\Delta_n(3)$
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'\nВывод: с погрешностью {d} значения векторов совпадают')
```