|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  |  | | --- | --- | --- | |  |  |  | | 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

Петров С.В.

Группа: КМБО-03-17

МОСКВА 2020

**Задание**

**Построить** граф состояний цепи Маркова.

**Найти**:

1. Матрицы переходных вероятностей за n шагов и величины отклонений , где Результаты представить в табличной форме.
2. Стационарное распределение вероятностей состояний (). Провести проверку стационарности найденного распределения.
3. Распределения вероятностей состояний через n шагов () (*n* = *1,…,*), где *m* = *min*(*n* | < 0,00001), для следующих начальных распределений: (1, 0, 0), (0, 1, 0) и (0, 0, 1). Результаты представить в табличной форме.
4. Для каждого состояния *i* =1, 2, 3, взятого в качестве начального провести в соответствии с матрицей переходных вероятностей генерацию последовательности номеров состояний через n шагов, определяя для каждого n значения *R(i, n)* = |{}| (число возвратов в состояние *i*) и (относительная частота возвратов в состояние *i*). Генерацию проводить до шага  *= min* (*n|* ), где . В отчете привести значения *min*
5. По результатам пункта 4 для каждого начального состояния *i* = 1, 2, 3 построить таблицы. *k* = *max* (16, .

Вычисления проводить с точностью до 0,00001.

**Краткие теоретические сведения**

Последовательность с.в. называется цепью Маркова, если для произвольного набора (*k* = 3, 4,…) и любых справедливо равенство

Цепь Маркова называется однородной, если для всех *i, j* вероятности не зависят от *n*.

Распределение цепи Маркова называется стационарным, если оно остается неизменным на каждом шаге.

Если существует и , то распределение называется предельным.

Матрица переходных вероятностей цепи Маркова:

Свойства матрицы переходных вероятностей:

1. для всех *i* = 1, 2, …, *N*.

Общие формулы:

*P(n)* = ;

Вычисления проводились средствами языка программирования Python:

Numpy – модуль для научных вычислений

Copy – модуль для копирования объектов

**Результаты расчетов**

**Вариант 44**

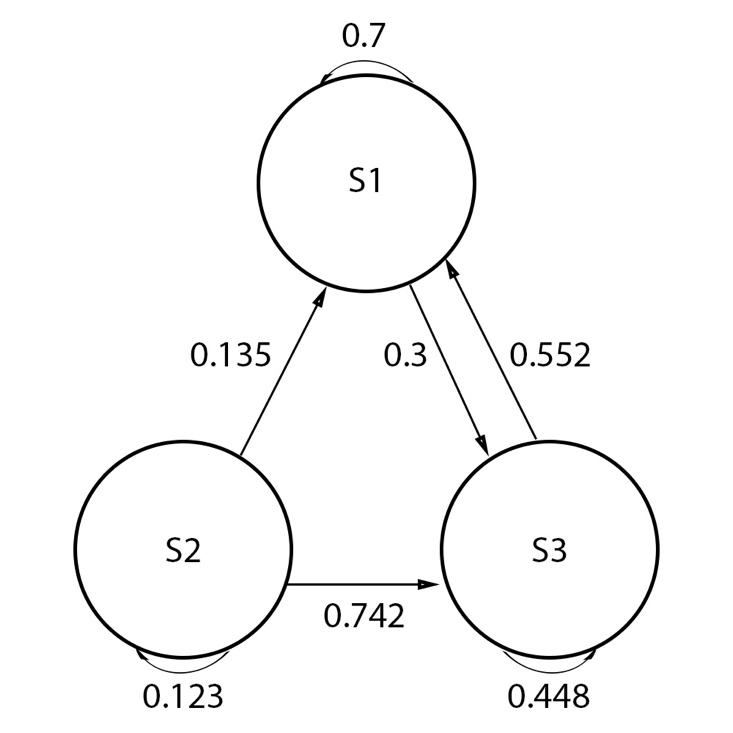


Рис. 1. Граф состояний цепи Маркова

1. Таблица 1

|  |  |  |
| --- | --- | --- |
| n |  |  |
| 1 |  | - |
| 2 |  | 0,385689 |
| 3 |  | 0,102064 |
| 4 |  | 0,020638 |
| 5 |  | 0,003735 |
| 6 |  | 0,000636 |
| 7 |  | 0,000104 |
| 8 |  | 0,00002 |
| 9 | ) | 0,000003 |

1. () = (0,427842, 0, 0,572158)

Проверка: () = () *P*

(0.427842, 0, 0.572158) \* () = (0.427842, 0, 0,572158)

1. Таблица 2

|  |  |  |
| --- | --- | --- |
| n | () |  |
| 0 | (1, 0, 0) | 0.352113 |
| 1 | (0.7, 0.0, 0.3) | 0.052113 |
| 2 | (0.6556, 0.0, 0.3444) | 0.007713 |
| 3 | (0.649029,0.0,0.3509712) | 0.001141 |
| 4 | (0.648056,0.0,0.351944) | 0.000169 |
| 5 | (0.647912,0.0,0.352088) | 0,000025 |
| 6 | (0.647891,0.0,0.352109) | 0,000004 |

Таблица 3

|  |  |  |
| --- | --- | --- |
| n | () |  |
| 0 | (0, 1, 0) | 0.352113 |
| 1 | (0.135, 0.123, 0.742) | 0.512887 |
| 2 | (0.520689,0.015129,0.464182) | 0.127198 |
| 3 | (0.622753,0.001861,0.375386) | 0.025134 |
| 4 | (0.643391, 0.000229, 0.35638) | 0.004496 |
| 5 | (0.647126,0,000028,0.352845) | 0.000761 |
| 6 | (0.647763,0,000003,0.352234) | 0.000124 |
| 7 | (0.647867,0,0.352132) | 0,00002 |
| 8 | (0.647884,0,0.352116) | 0,000003 |

Таблица 4

|  |  |  |
| --- | --- | --- |
| n | () |  |
| 0 | (0, 0, 1) | 0.352113 |
| 1 | (0.552, 0.0, 0.448) | 0.095887 |
| 2 | (0.633696,0.0,0.366304) | 0.014191 |
| 3 | (0.645787,0.0,0.354213) | 0.0021 |
| 4 | (0.647576,0.0,0.352424) | 0.000311 |
| 5 | (0.647841,0.0,0.352159) | 0,000046 |
| 6 | (0.647881,0.0,0.352119) | 0,000007 |



**Анализ результатов и выводы**

1. Стационарное распределение: () = (0,427842, 0, 0,572158)

Таблица 5.

|  |  |  |
| --- | --- | --- |
| () | () | *n* = |
| (1, 0, 0) | (0.647891,0,0.352109) | 6 |
| (0, 1, 0) | (0.647884,0,0.352116) | 8 |
| (0, 0, 1) | (0.647881,0,0.35212) | 6 |

1. Таблица 6. i = 1

|  |  |  |  |
| --- | --- | --- | --- |
| *n* | *R(i, n)* |  |  |
| 1 | 1 | 1.0 | 0,35211 |
| 2 | 2 | 1.0 | 0,35211 |
| 3 | 3 | 1.0 | 0,35211 |
| 4 | 4 | 1.0 | 0,35211 |
| 5 | 4 | 0.8 | 0,15211 |
| 6 | 5 | 0.83333 | 0,18544 |
| 7 | 6 | 0.85714 | 0,20925 |
| 8 | 7 | 0.875 | 0,22711 |
| 9 | 7 | 0.77778 | 0,12989 |
| 10 | 8 | 0.8 | 0,15211 |
| 11 | 8 | 0.72727 | 0,07938 |
| 12 | 8 | 0.66667 | 0,01878 |
| 13 | 9 | 0.69231 | 0,04442 |
| 14 | 10 | 0.71429 | 0,0664 |
| 15 | 10 | 0.66667 | 0,01878 |
| 16 | 10 | 0.625 | 0,02289 |
| 17 | 11 | 0.64706 | 0,00083 |

Таблица 7. i = 2

|  |  |  |  |
| --- | --- | --- | --- |
| *n* | *R(i, n)* |  |  |
| 1 | 0 | 0 | 0 |

Таблица 8. i = 3

|  |  |  |  |
| --- | --- | --- | --- |
| *n* | *R(i, n)* |  |  |
| 1 | 1 | 1.0 | 0,64789 |
| 2 | 2 | 1.0 | 0,64789 |
| 3 | 2 | 0.66667 | 0,14789 |
| 4 | 2 | 0.5 | 0.14789 |
| 5 | 3 | 0.6 | 0,24789 |
| 6 | 4 | 0.66667 | 0,31456 |
| 7 | 4 | 0.57143 | 0,21932 |
| 8 | 4 | 0.5 | 0.14789 |
| 9 | 5 | 0.55556 | 0,20345 |
| 10 | 6 | 0.6 | 0,24789 |
| … | … | … | … |
| 64 | 24 | 0.375 | 0,02289 |
| 65 | 24 | 0.36923 | 0,01712 |
| 66 | 24 | 0.36364 | 0,01153 |
| 67 | 24 | 0.35821 | 0,0061 |
| 68 | 24 | 0.35294 | 0,00083 |

**Литература**

1. Вентцель Е.С., Овчаров Л.А. Теория случайных процессов и ее инженерные приложения. — М.: Высшая школа, 2007.

2. Булинский А. В., А. Н. Ширяев А. Н. Теория случайных процессов: Учебник для вузов. — М.: ФИЗМАТЛИТ, 2005.

3. Лобузов А.А., Гумляева С.Д., Норин Н.В. Задачи по теории случайных процессов. — М.: МИРЭА, 1993.

4. Чжун Кай-Лай. Однородные цепи Маркова. — М.: Мир, 1964.  
5. Карлин С. Основы теории случайных процессов. — М.: Мир, 1971.

**Приложение**

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

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

**from** numpy **import** matmul

**from** math **import** fabs

**import** copy

variant = 60

matrix = np.array([[0.7, 0, 0.3], [0.135, 0.123, 0.742], [0.552, 0, 0.448]])

**def** convert\_float(num):

**return** "{:f}".format(float(num))

**def** getMatrixes1(matrix):

ans = [matrix]

deltas = []

matrix\_old = matrix

delta = 0.00001

delta\_k = 0

n = 1

**while** delta\_k > delta **or** n == 1:

delta\_k = 0

n += 1

matrix\_new = matrix\_power(matrix, n)

ans.append(np.array(matrix\_new))

**for** i **in** range(len(matrix)):

delta\_k = max(delta\_k, max([fabs(matrix\_new[i][k] - matrix\_old[i][k]) **for** k **in** range(len(matrix))]))

matrix\_old = matrix\_new

deltas.append(delta\_k)

**return** [ans, n, deltas, ans[-1]]

matrixes, n, deltas, final\_matrix = getMatrixes1(matrix)

**print**(n)

**for** i **in** matrixes:

**print**(i)

**print**()

**print**(deltas)

# r = [46 / 71, 0, 25 / 71]

tmp = copy.deepcopy(matrix)

tmp = np.transpose(tmp)

**for** i **in** range(3):

tmp[i][i] -= 1

tmp = np.append(tmp, [[1, 1, 1]], axis=0)

r = np.linalg.solve(tmp[1:4], [0,0,1])

**print**("r: ", r)

**def** getMatrixes3(matrix, vector, r):

ans = [vector]

n = 0

delta = 0.00001

delta\_k = 0

deltas = [max((1 - r[0]), r[1], r[2])]

**while** delta\_k > delta **or** n == 0:

n += 1

vector = matmul(vector, matrix)

ans.append(list(vector))

delta\_k = max([fabs(vector[i] - r[i]) **for** i **in** range(3)])

deltas.append(delta\_k)

**return** [ans, n, deltas]

vectors, m\_min, deltas = getMatrixes3(matrix, [1, 0, 0], r)

**print**("100:")

**print**(vectors)

**print**(m\_min)

**print**(deltas)

vectors, m\_min, deltas = getMatrixes3(matrix, [0, 1, 0], r)

**print**("010:")

**print**(vectors)

**print**(m\_min)

**print**(deltas)

vectors, m\_min, deltas = getMatrixes3(matrix, [0, 0, 1], r)

**print**("001:")

**print**(vectors)

**print**(m\_min)

**print**(deltas)

**def** task4(P, r):

n\_min = []

**for** i **in** range(3):

r\_i\_n = []

v\_i\_n = []

deltas = []

p\_0 = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])

P\_last = P

n = 0

R = 0

delta = 1.0

**while** delta > 0.001:

n += 1

tmp = p\_0[i].dot(P\_last)

P\_last = P\_last.dot(P)

alpha = np.random.random\_sample()

**if** alpha < tmp[0]:

**if** i == 0:

R += 1

**elif** alpha < tmp[0] + tmp[1]:

**if** i == 1:

R += 1

**elif** alpha <= 1:

**if** i == 2:

R += 1

v = R / n

delta = fabs(v - r[i])

**if** n <= 16:

r\_i\_n.append(R)

v\_i\_n.append(round(v, 5))

deltas.append(round(delta,5))

**print**("V: ", v\_i\_n)

**print**("R: ", r\_i\_n)

n\_min.append(n)

**print**('N\_min: ', n\_min)

task4(matrix, r)