Федеральное агентство связи

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Кафедра вычислительных систем

**Лабораторная работа №3**

«Рандомизированная цепь Маркова»

Выполнил:

студент гр. МГ-211 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Бурдуковский И.А./

подпись

Проверил:

Профессор

кафедры ВС \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Родионов А.С./

Новосибирск

2023 г.

Оглавление

[Задание 3](#_Toc133933349)

[Выполнение работы 4](#_Toc133933350)

# Задание

Требуется построить две два-стохастические матрицы, проверить их на два-стохастичность. Запустить рандомизированную цепь Маркова и получить некоторое количество сгенерированных на интервалах значений. Продемонстрировать разное поведение для двух матриц на графиках. Для убедительности построить график автокорреляционной функции.

# Выполнение работы

В соответствии с заданием была создана python программа. При запуске она генерирует две два-стохастических матрицы.

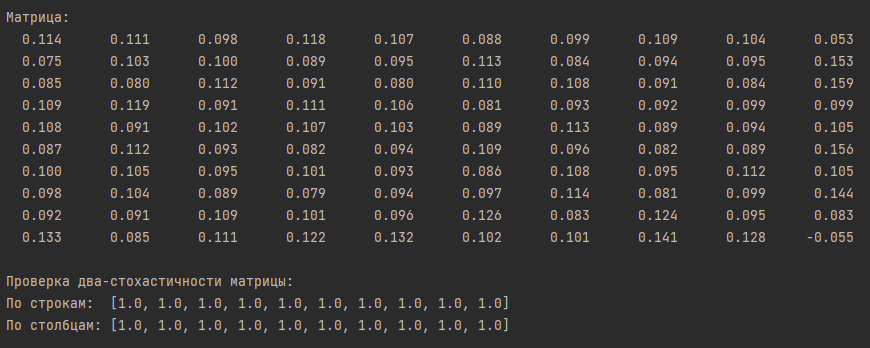


Рис. 1 – Первая матрица

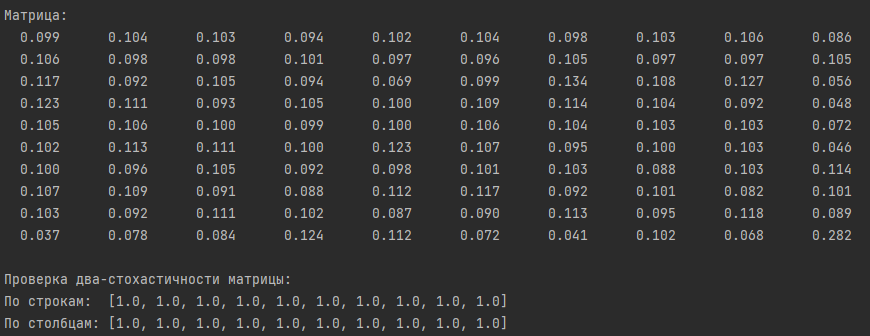


Рис. 2 – Вторая матрица

Затем отрабатывает рандомизированная цепь Маркова:

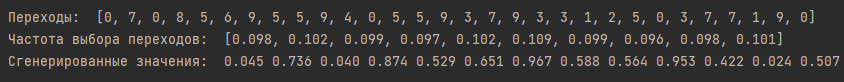


Рис. 3 – Для первой матрицы

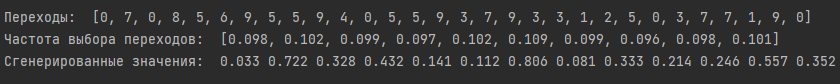
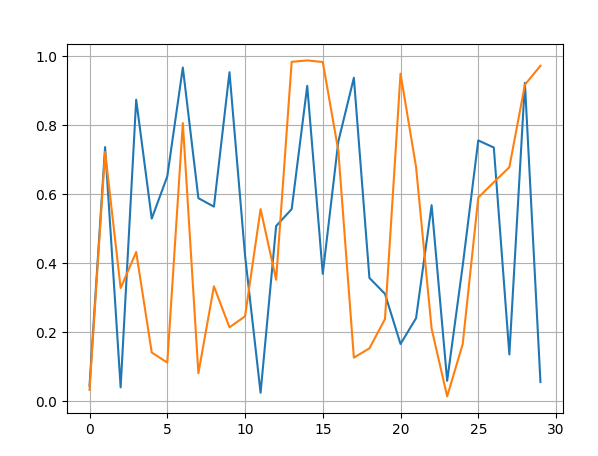
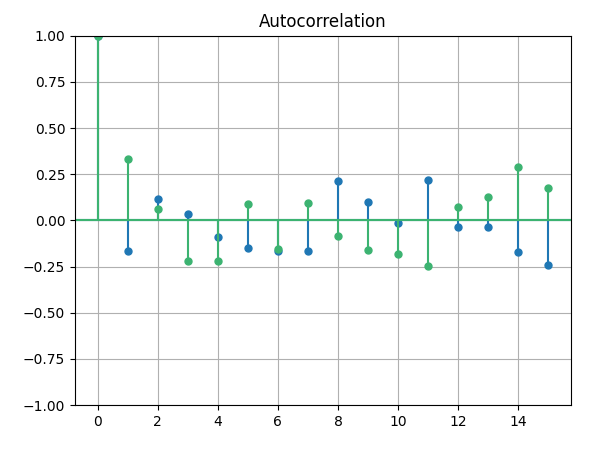


Рис. 4 –Для второй матрицы

Если вывести значения двух два-стохастических матриц на график, то можно пронаблюдать их ожидаемо различное поведение.



Для убедительности, построим график автокорреляционной функции для этих же значений матриц.



# Листинг

import random  
import numpy as np  
from statsmodels.graphics.tsaplots import plot\_acf  
import matplotlib.pyplot as plt  
  
n = 10  
size = 100  
show\_size = 30  
shift = 0  
  
  
def normalization(matrix, rowsum):  
 for i in range(n):  
 rowSum2 = 0  
 for j in range(n):  
 if j == n - 1:  
 matrix [i][j] = 1 - rowSum2  
 break  
  
 matrix [i][j] = matrix [i][j] / rowsum[i]  
 rowSum2 += matrix [i][j]  
 return matrix  
  
  
def check\_matrix(matrix):  
 print('Проверка два-стохастичности матрицы:')  
 rowMultySum = [round(sum(e), 1) for e in matrix]  
 print('По строкам: ', rowMultySum)  
  
 colsum = []  
 for column in range(len(matr[0])):  
 t = 0  
 for row in matrix:  
 t += row[column]  
 colsum.append(round(t, 1))  
 print('По столбцам:', colsum)  
  
  
def Chain(matrix):  
 partLen = 1 / n  
 selectedNodes = []  
 freq = [0 for j in range(n)]  
 generatedset = []  
  
 currentNode = 0  
 freq[currentNode] += 1  
 selectedNodes.append(currentNode)  
  
 beginborder = currentNode \* partLen  
 endborder = beginborder + partLen  
 generatedset.append(random.uniform(beginborder, endborder))  
  
 for i in range(size - 1):  
 nextNodeProb = random.uniform(0, 1)  
 cumulativeProb = 0  
 for j in range(len(matrix [currentNode])):  
 cumulativeProb += matrix [currentNode][j]  
  
 if j == len(matrix [currentNode]) - 1:  
 cumulativeProb = 1  
  
 if nextNodeProb <= cumulativeProb:  
 currentNode = j  
 freq[currentNode] += 1  
 selectedNodes.append(currentNode)  
  
 beginborder = currentNode \* partLen  
 endborder = beginborder + partLen  
 generatedset.append(random.uniform(beginborder, endborder))  
 break  
 print()  
  
 nodesFreq = []  
 for i in freq:  
 nodesFreq.append(i / sum(freq))  
  
 return selectedNodes, nodesFreq, generatedset, matr  
  
  
def optimization(matrix):  
 for i in range(len(matrix)):  
 rowSum = 0  
 for j in range(len(matrix [i]) - 1):  
 rowSum += matrix [i][j]  
 matrix [i][len(matrix) - 1] = 1 - rowSum  
  
 for j in range(len(matrix [0])):  
 colSum = 0  
 for i in range(len(matrix) - 1):  
 colSum += matrix[i][j]  
 matrix[len(matrix[0]) - 1][j] = 1 - colSum  
  
 return matr  
  
  
def print\_matrix(matrix):  
 print('\nМатрица:')  
 for i in range(n):  
 for j in range(n):  
 print("%7.3f" % (matrix[i][j]), end=' ')  
 print()  
 print()  
  
  
def generate\_two\_stochastic\_matrix():  
 matrix1 = [[random.randint(0, 100) for i in range(n)] for j in range(n)]  
 rowSum1 = [sum(e) for e in matrix1]  
  
 matrix1 = normalization(matrix1, rowSum1)  
 # print\_matrix(matrix1)  
  
 matrix2 = [[random.randint(0, 100) for i in range(n)] for j in range(n)];  
 rowSum2 = [sum(e) for e in matrix2]  
  
 matrix2 = normalization(matrix2, rowSum2)  
 # print\_matrix(matrix2)  
  
 matrix2\_transparent = [[matrix2[j][i] for j in range(len(matrix2))] for i in range(len(matrix2[0]))]  
 matrix2 = matrix2\_transparent  
 # print\_matrix(matrix2)  
  
 two\_stochastic\_matrix = [[0 for i in range(n)] for j in range(n)]  
 for i in range(len(matrix1)):  
 for j in range(len(matrix2[0])):  
 for k in range(len(matrix2)):  
 two\_stochastic\_matrix[i][j] += matrix1[i][k] \* matrix2[k][j]  
 two\_stochastic\_matrix = optimization(two\_stochastic\_matrix)  
  
 return two\_stochastic\_matrix  
  
  
def get\_Markov\_chain():  
  
 matrix = generate\_two\_stochastic\_matrix()  
 print\_matrix(matrix)  
 check\_matrix(matrix)  
 return Chain(matrix)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
  
 selectedNodes1, freq1, set1, multyRes1 = get\_Markov\_chain()  
  
 print('Переходы: ', selectedNodes1[0:show\_size])  
 print('Частота выбора переходов: ', [round(i, 3) for i in freq1])  
 print('Сгенерированные значения: ', end=' ')  
 for i in range(show\_size):  
 print("%5.3f" % set1[i], end=' ')  
 print()  
 print()  
  
 fig, ax1 = plt.subplots()  
  
 plot\_acf(np.array(set1[shift:shift + show\_size]), ax=ax1, alpha=None)  
  
 selectedNodes2, freq2, set2, multyRes2 = get\_Markov\_chain()  
  
 print('Переходы: ', selectedNodes1[0:show\_size])  
 print('Частота выбора переходов: ', [round(i, 3) for i in freq1])  
 print('Сгенерированные значения: ', end=' ')  
 for i in range(show\_size):  
 print("%5.3f" % set2[i], end=' ')  
 print()  
 print()  
  
 plot\_acf(np.array(set2[shift:shift + show\_size]), ax=ax1, alpha=None, color='mediumseagreen', vlines\_kwargs={"colors": 'mediumseagreen'})  
 plt.grid()  
 plt.show()  
  
 print()  
 plt.plot(range(show\_size), set1[shift:shift + show\_size])  
 plt.plot(range(show\_size), set2[shift:shift + show\_size])  
 plt.grid()  
 plt.show()