Российский Государственный Университет нефти и газа имени И.М.Губкина

Кафедра «Прикладной математики и компьютерного моделирования»

Лабораторная работа №3:

«Моделирование и анализ Марковского процесса»

Выполнила работу:

студент гр. АМ 20-06

Коротченя И. С.

Проверил:

Кузнецов П. В.

Москва   
2023 г.

Постановка задачи

1. Требуется смоделировать *K* реализаций Марковского процесса, определенных матрицей интенсивностей перехода Λ. Процесс может находиться в одном из трех состояний: {0, 1, 2}.   
Для всех реализаций процесса установить ограничение по времени .

2. Построить траектории реализаций. Для заданного состояния проверить выполнение центральной предельной теоремы (ЦПТ) и закона больших чисел (ЗБЧ).

3. По полученным траекториям построить эмпирическую матрицу интенсивностей переходов.

**Исходные данные:**

K = 100 – количество реализаций.

T = 100 – длина реализации.

– матрица интенсивности переходов

**Ход работы:**

Случайый процесс называется марковским, если для каждого момента времени t вероятность любого сосотояния системы в будущем зависит только от его состояния в настоящем.

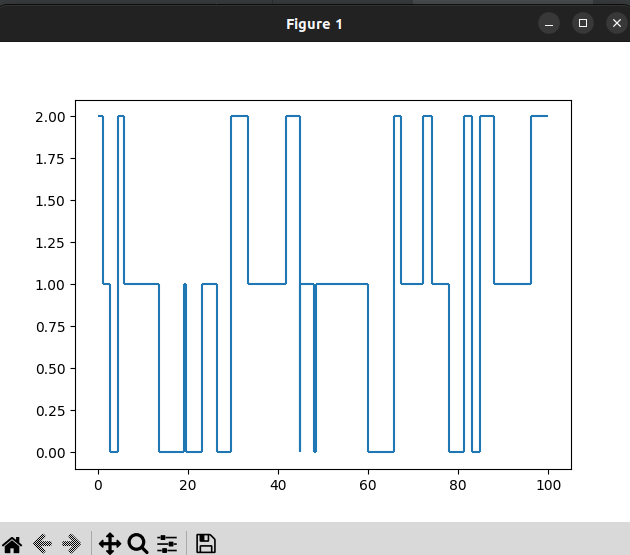
Случйное время пребывания в состоянии имеет показательное распределение с параметром

Матрица интенсивности, представляет собой матрицу, элементы который равны:

В момент скачка вероятность перехода равна:

, где

Изобразим одну из реализаций марковского процесса, заданную таблицей интенсивностей.

 Стационарные вероятности

Теоретически стационарные вероятности рассчитываются с помощью системы формул:

где , вектор стационарных вероятностей

- условие нормировки

Эмпирические:

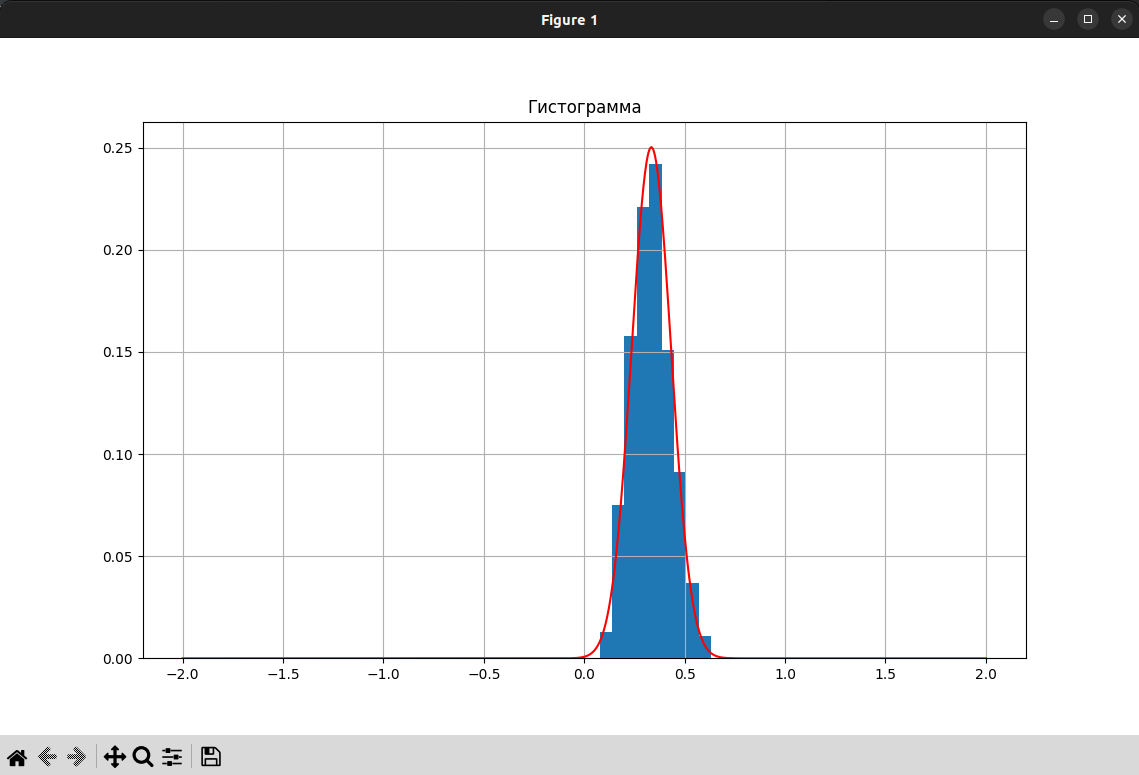
Теоретические стационарные вероятности: 0.33333333, 0.44444444,

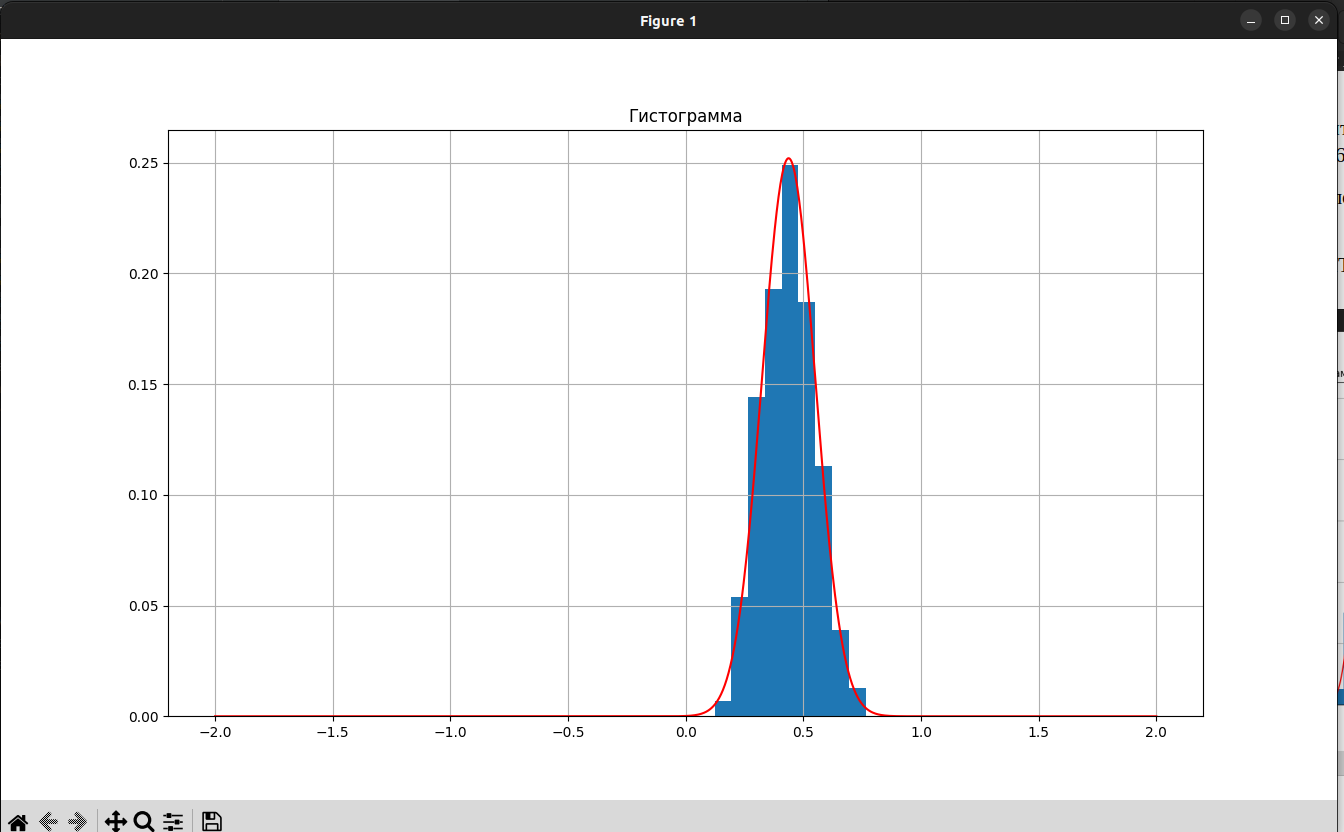
0.22222222

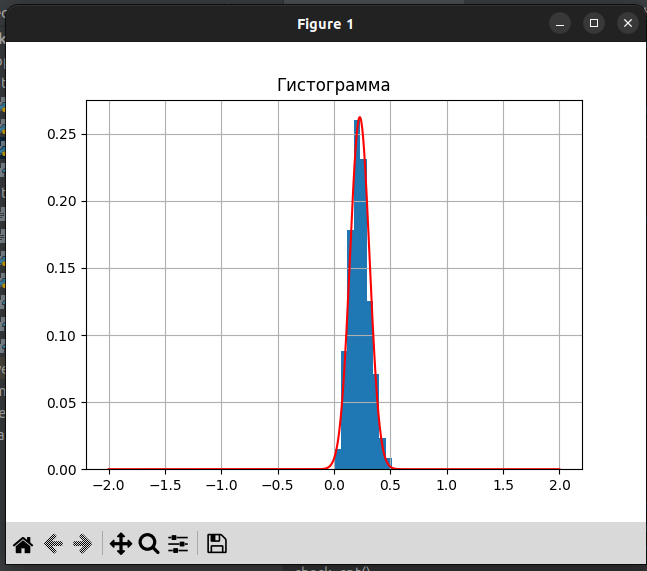
Эмпирические стационарные вероятности: 0.3159603618914241, 0.4682130567088925, 0.21582658139968336*.*

Можно сделать вывод, что ЗБЧ выполняется.

Проверим условия выполнения ЦПТ:







import math  
from matplotlib import pyplot as plt  
import numpy as np  
from scipy import stats  
  
  
class State:  
 def \_\_init\_\_(self, my\_lambda, name):  
 self.name = name  
 self.my\_lambda = my\_lambda  
 self.another\_states = []  
  
 def stay(self):  
 return np.random.exponential(1/self.my\_lambda)  
  
 def add\_another\_state(self, state, ver):  
 self.another\_states.append((state, ver))  
  
 def choice\_new\_state(self):  
 num = np.random.random()  
 if num <= self.another\_states[0][1]:  
 return self.another\_states[0][0]  
 return self.another\_states[1][0]  
  
 def \_\_repr\_\_(self):  
 return self.name  
  
  
class MarkovProcces:  
 def \_\_init\_\_(self, ver\_mat, lamdas, t, do\_draw=False):  
 self.do\_draw = do\_draw  
 self.ver\_mat = ver\_mat  
 self.max\_time = t  
 self.now\_time = 0  
 self.all\_time = []  
 self.all\_states = []  
 self.states = [State(lamdas[0], '0'), State(lamdas[1], '1'), State(lamdas[2], '2')]  
 self.now\_state = np.random.choice(self.states)  
 self.add\_vers\_to\_states()  
 self.start\_process()  
  
 def add\_vers\_to\_states(self):  
 for i in range(3):  
 can\_be = list(range(3))  
 can\_be.pop(i)  
 for j in can\_be:  
 self.states[i].add\_another\_state(self.states[j], self.ver\_mat[i][j])  
  
 def draw(self):  
 y = list(map(int, [i.name for i in self.all\_states]))  
 x = self.all\_time  
  
 for i in range(1, len(x)):  
 plt.hlines(y[i - 1], x[i-1], x[i])  
 plt.vlines(x[i], y[i - 1], y[i])  
  
 plt.hlines(y[-1], x[-1], self.max\_time)  
 plt.show()  
  
  
 def start\_process(self):  
 while True:  
 self.all\_time.append(self.now\_time)  
 self.all\_states.append(self.now\_state)  
 self.now\_time += self.now\_state.stay()  
 if self.now\_time > self.max\_time:  
 break  
 self.now\_state = self.now\_state.choice\_new\_state()  
 if self.do\_draw:  
 self.draw()  
  
 def calculate\_all\_time(self):  
 states = list(map(int, [i.name for i in self.all\_states]))  
 res = {i: 0 for i in range(3)}  
 for i in range(len(self.all\_time) - 1):  
 res[states[i]] += self.all\_time[i+1] - self.all\_time[i]  
 res[states[-1]] += self.max\_time - self.all\_time[-1]  
 return res  
  
  
def make\_ver\_matrix(in\_mat):  
 ver\_mat = []  
 for i in range(3):  
 temp = []  
 for j in range(3):  
 if i == j:  
 temp.append(0)  
 else:  
 temp.append(in\_mat[i, j] / -in\_mat[i, i])  
 ver\_mat.append(temp)  
 return ver\_mat  
  
  
def make\_markov(t, in\_mat, draw):  
 ver\_mat = make\_ver\_matrix(in\_mat)  
 MarkovProcces(ver\_mat, [-in\_mat[0, 0], -in\_mat[1, 1], -in\_mat[2, 2]], t, do\_draw=draw)  
  
  
def make\_static\_ver(transition\_matrix, t=1000):  
 Q = transition\_matrix  
 # решаем уравнение Q \* pi = 0 для pi  
 w, v = np.linalg.eig(Q.T)  
 pi = v[:, np.isclose(w, 0)]  
 # нормируем вектор pi  
 pi = pi / pi.sum()  
 print('Теоретические стационарные вероятности:', pi)  
  
 ver\_mat = make\_ver\_matrix(transition\_matrix)  
 proc = MarkovProcces(ver\_mat, [-transition\_matrix[0, 0], -transition\_matrix[1, 1], -transition\_matrix[2, 2]], t=t, do\_draw=False)  
 emper = proc.calculate\_all\_time()  
 for i in emper.keys():  
 emper[i] /= t  
 print('Эмпирические стационарные вероятности: ', emper.values())  
  
  
def check\_cpt(transition\_matrix, t=100):  
 ver\_mat = make\_ver\_matrix(in\_mat)  
 all\_emper\_vers = {0: [], 1: [], 2: []}  
 for i in range(1000):  
 proc = MarkovProcces(ver\_mat, [-in\_mat[0, 0], -in\_mat[1, 1], -in\_mat[2, 2]], t, do\_draw=False)  
 emper = proc.calculate\_all\_time()  
 for k in emper.keys():  
 emper[k] /= t  
 for k in all\_emper\_vers:  
 all\_emper\_vers[k].append(emper[k])  
 draw\_hist(all\_emper\_vers[0])  
 draw\_hist(all\_emper\_vers[1])  
 draw\_hist(all\_emper\_vers[2])  
  
  
  
def draw\_hist(sample):  
 fig = plt.figure()  
 ax = fig.add\_subplot(111)  
  
 n\_bins = 10  
 intervals = np.linspace(min(sample), max(sample), n\_bins) # начало каждого интервала  
 frequency = np.array([len([value for value in sample if intervals[i] <= value < intervals[i + 1]]) for i in range(n\_bins-1)])  
 frequency = frequency / len(sample) # частоты  
 labels = [(intervals[i] + intervals[i + 1]) / 2 for i in range(n\_bins-1)] # середины столбцов  
 width = labels[1] - labels[0] # ширина одного столбца  
  
 ax.bar(labels, frequency, width)  
  
 mu, std = stats.norm.fit(sample)  
 print(mu, std)  
 x = np.linspace(-2, 2, num=1000) # создаем массив значений x  
 y = stats.norm.pdf(x, mu, std) \* width # вычисляем плотность вероятности стандартного нормального распределения  
 plt.plot(x, y, c='r') # строим график  
  
 ax.set\_title('Гистограмма')  
 plt.grid()  
 plt.show()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 in\_mat = np.array([[-0.3, 0.1, 0.2],  
 [0.15, -0.2, 0.05],  
 [0.15, 0.25, -0.4]])  
 #make\_markov(t=100, in\_mat=in\_mat, draw=True)  
 # make\_static\_ver(in\_mat, t=10000)  
 # check\_cpt(in\_mat)