Національний технічний університет України
«Київський політехнічний інститут»
Факультет інформатики і обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота №5
«Випадкові процеси»

Виконав:
Студент групи ІВ-71
Мазан Я. В.
Залікова книжка №7109
Перевірив:
доцент Марковський О. П.

м. Київ
2018 р.

Варіант: ПМ-2, 3 стани

Код програми:

```
from MarkovProcesses import *
G = SemiMarkovProcess([[2,2,2],
                       [2,0,0.1],
                       [0,0.01,0.5]])
# F = ContiniousMarkovProcess([[2,2,2],
#                             [2,0,0.1],
#                             [0,0.01,0.5]])
print("Stationary probabilities of semi markov process: ".ljust(65) +
str(G.stationary_probabilities()))
print("Simulation of semi markov process: ".ljust(65) + str(G.simulate(10000)))
# print("\nStationary probabilities of identical continious markov process: ".ljust(65)
+ str(F.static_conditions()))

import random
import math
import numpy as np
class SemiMarkovProcess:
    def __init__(self, matrix: "two-dimensional double array"):
        self.relation_matrix = matrix
        self.equation = None
    def toMarcovProcess(self):
        #defining size of support matrix and position of "basic" nodes in it
        size = len(self.relation_matrix)
        index = 0
        self.real_nodes = list()
        for i in self.relation_matrix:
            self.real_nodes.append(index)
            for j in i:
                if j != 0:
                    size+=1
                    index+=1
            index+=1
        #filling the additional matrix with values
        additional_matrix = [[0 for i in range(size)] for j in range(size)]
        for i in range(len(self.relation_matrix)):
            row = self.real_nodes[i]
            column = self.real_nodes[i]
            for j in range(len(self.relation_matrix[0])):
                if self.relation_matrix[i][j] != 0:
                    additional_matrix[row][column+1] = 2 * self.relation_matrix[i][j]
                    additional_matrix[column+1][self.real_nodes[j]] = 2 *
self.relation_matrix[i][j]
                    column += 1
        # for i in self.relation_matrix:
        #     print(i)
        #
        # print("\n")
        # print(real_nodes)
        # for i in additional_matrix:
        #     print(i)
        return ContiniousMarkovProcess(additional_matrix, self.real_nodes)
    def stationary_probabilities(self):
        F = self.toMarcovProcess()
        return F.toSemiMarcovProcess()
    def simulate(self, tests_num):
        F = self.toMarcovProcess()
        return F.toSemiMarcovProcess(F.simulate(tests_num,0))
# additional continious markov process matrix used for calculation stationary
probabilities of any semi markov process
```

```python
class ContiniousMarkovProcess:
    def __init__(self, matrix, base_nodes = None):
        default = list(range(len(matrix)))
        self.base_matrix = matrix
        self.base_nodes = base_nodes if not base_nodes == None else default
    def static_conditions(self):
        equation = [[0 for i in range(len(self.base_matrix))] for j in range(len(self.base_matrix))]
        for i in range(len(equation)-1):
            s = 0
            for j in range(len(equation)):
                s+=self.base_matrix[i][j]
                equation[i][i] = -s
                if self.base_matrix[j][i] != 0:
                    equation[i][j] = self.base_matrix[j][i]
        for i in range(len(equation)):
            equation[len(equation)-1][i] = 1
        # for i in equation:
        #     print(i)
        free_members = [0 if i < len(self.base_matrix)-1 else 1 for i in range(len(self.base_matrix))]
        # solve the equations system and round the result to four digits after point
        c = list(map(lambda a: round(a,4), np.linalg.solve(equation, free_members)))
        # print(c)
        return c
    def simulate(self, tests_num, initial_state):
        system_time = 0
        time_in_states = [0 for i in range(len(self.base_matrix))]
        state = initial_state
        for iteration in range(tests_num):
            transition_times = [-1.0/x*math.log(random.random()) if x != 0 else float("inf") for x in self.base_matrix[state]]
            transition_time = min(transition_times)
            next_state = transition_times.index(transition_time)
            time_in_states[state]+=transition_time
            system_time+=transition_time
            state = next_state
        stationary_probabilities = [round(i/system_time,4) for i in time_in_states]
        # print(stationary_probabilities)
        return stationary_probabilities
    def toSemiMarcovProcess(self, conditions = []):
        p_i = self.static_conditions() if conditions == [] else conditions
        stationary_probabilities = [0] * len(p_i)
        for i in range(len(p_i)):
            # defining which p_i probabilities in additional matrix add to corresponding base node total probability
            less_than_i = list(filter(lambda a: a <= i, self.base_nodes))
            related_node = max(less_than_i)
            stationary_probabilities[related_node]+=p_i[i]
        # getting rid of non-base cells
        stationary_probabilities = [round(j,4) for (i,j) in enumerate(stationary_probabilities) if i in self.base_nodes]
        return stationary_probabilities
```