|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

**ОТЧЕТ**

*к лабораторной работе №2*

*По курсу: «Моделирование»*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ***ИУ7И-76Б*** |  |  | **Нгуен Ф. С.** |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | |  | | --- | | **Рудаков И.В.** | |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

*Москва, 2021 г.*

Оглавление

[**I.** **Теоретическая часть** 3](#_Toc88611859)

[**II.** **Результаты** 5](#_Toc88611860)

[Эксперимент I (Система имеет 4 состояния) 5](#_Toc88611861)

[Эксперимент II (Система имеет 5 состояния) 7](#_Toc88611862)

[**III.** **Код Программы** 9](#_Toc88611863)

1. **Теоретическая часть**

Случайный процесс, протекающий в системе S, называется марковским, если он обладает следующим свойством: для каждого момента времени 𝑡0 вероятность любого состояния системы в будущем (при 𝑡 > 𝑡0) зависит только от ее состояния в настоящем (при 𝑡 = 𝑡0) и не зависит от того, когда и каким образом система пришла в это состояние. Вероятностью i-го состояния называется вероятность 𝑝𝑖(𝑡) того, что в момент t система будет находиться в состоянии 𝑆𝑖. Для любого момента t сумма вероятностей всех состояний равна единице.

Уравнения Колмогорова в общем виде:

|  |  |
| --- | --- |
|  | (1) |

Имея в распоряжении размеченный граф состояний, можно найти все вероятности состоянийp\_i (t)как функции времени. Для этого составляются и решаются так называемые уравнения Колмогорова особого вида дифференциальные уравнения, в которых неизвестными функциями являются вероятности состояний.

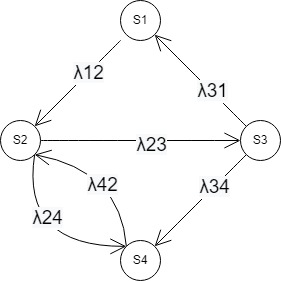
Правило:

* в левой части каждого из уравнений стоит производная вероятности i-ого состояния;
* в правой части содержится столько членов, сколько стрелок связано с данным состоянием;
* если стрелка направлена из состояния, соответствующий член имеет знак «минус», если в состояние — знак «плюс»;
* каждый член равен произведению интенсивности, соответствующей данной стрелке, и вероятности того состояния, из которого выходит стрелка.

Если в уравнениях Колмагорова приравнять производные к нулю, то получим систему уравнений, описывающих стационарный режим.

Для поиска решений необходимо добавить уравнение нормировки:

|  |  |
| --- | --- |
|  | (2) |

Пример

# Результаты

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Данные |  | S1 | S2 | S3 | S4 |
| S1 | **0** | **2** | **0** | **0** |
| S2 | **0** | **0** | **2** | **3** |
| S3 | **3** | **0** | **0** | **1** |
| S4 | **0** | **4** | **0** | **0** |
| Результаты | P | **0.24** | **0.32** | **0.16** | **0.28** |
| T1 | **2.199** | **2.899** | **3.074** | **3.376** |
| T2 | **1.612** | **0958** | **0.536** | **1.359** |

## Эксперимент I (Система имеет 4 состояния)

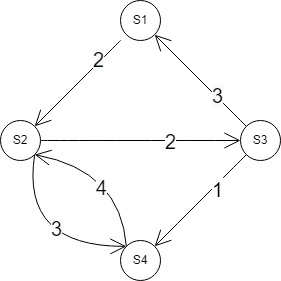


Рисунок 1. граф связей и интенсивностей системы Эксперимент I

T1 - Время стабилизации, при начальных условиях P1 =1, P2,3,4 = 0

T2 - Время стабилизации, при начальных условиях P1,2,3,4 = 1/4

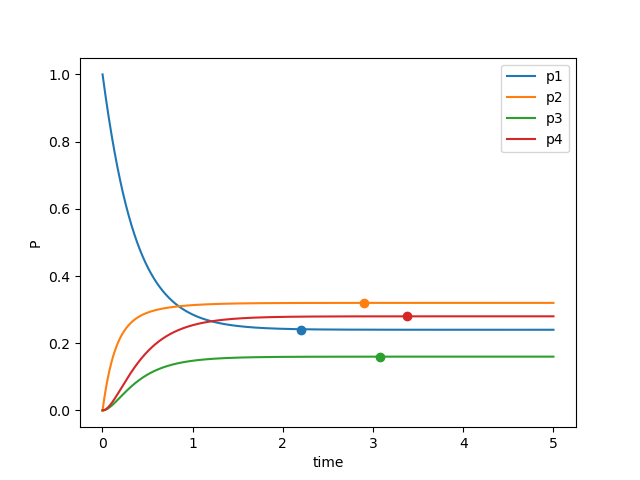


Рисунок 2. графики вероятностей состояний как функции времени, при начальных условиях P1 =1, P2,3,4 = 0

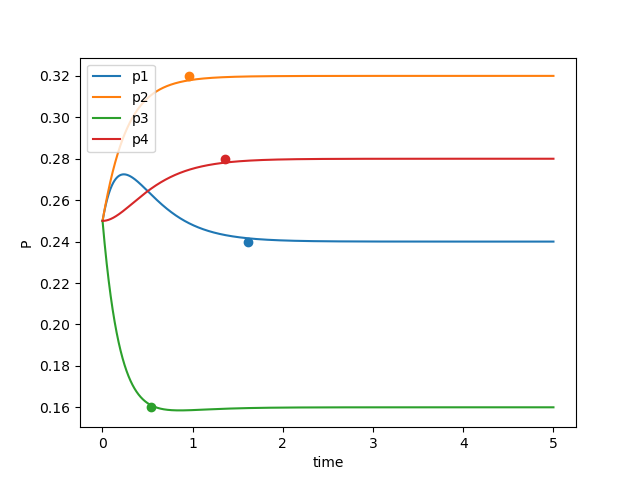


Рисунок 3. графики вероятностей состояний как функции времени, при начальных условиях P1,2,3,4 = 1/4

## Эксперимент II (Система имеет 5 состояния)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Данные |  | S1 | S2 | S3 | S4 | S5 |
| S1 | **0** | **1** | **1** | **2** | **0** |
| S2 | **0** | **0** | **0** | **0** | **0.5** |
| S3 | **0** | **2** | **0** | **1** | **0** |
| S4 | **0** | **0** | **3** | **0** | **1.5** |
| S5 | **2** | **0** | **0** | **2** | **0** |
| Результаты | P | **0.057** | **0.603** | **0.122** | **0.103** | **0.114** |
| T1 | **1.494** | **4.734** | **3.334** | **2.880** | **3.292** |
| T2 | **1.525** | **4.631** | **3.221** | **3.773** | **0.911** |

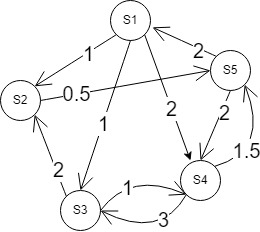


Рисунок 4. граф связей и интенсивностей системы Эксперимент II

T1 - Время стабилизации, при начальных условиях P1 =1, P2,3,4,5 = 0

T2 - Время стабилизации, при начальных условиях P1,2,3,4,5 = 1/5

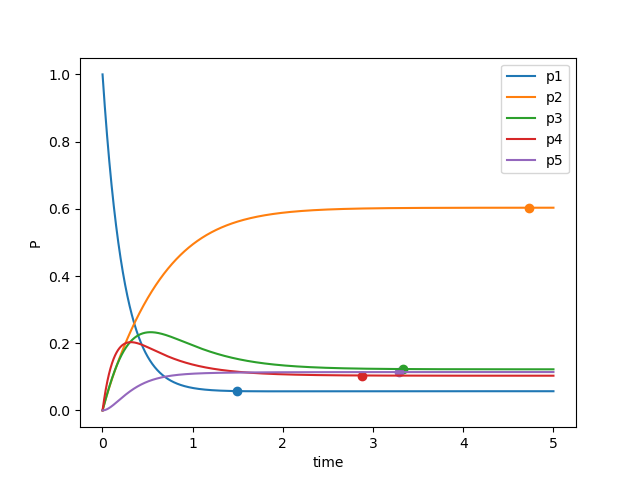


Рисунок 5. графики вероятностей состояний как функции времени, при начальных условиях P1 = 1, P2,3,4,5 = 0

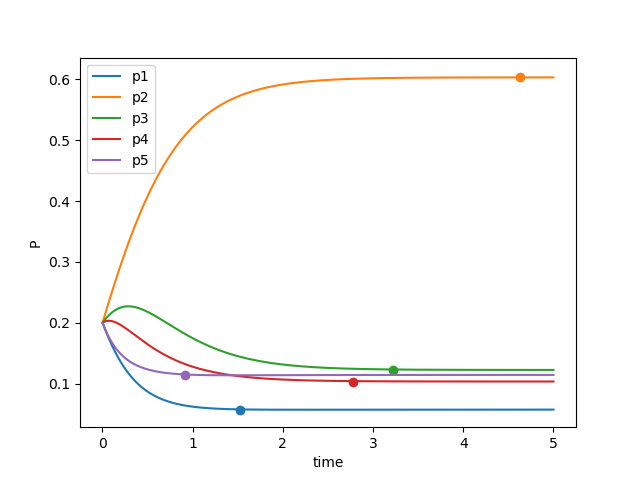


Рисунок 6. графики вероятностей состояний как функции времени, при начальных условиях P1,2,3,4,5 = 1/5

1. **Код Программы**

**Main.py**

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

**import** random

**import** matplotlib**.**pyplot **as** plt

**import** numpy

PRECISION **=** 5

TIME\_DELTA **=** 1e-3

MAGIC\_NUM **=** 10

**def** dps**(**matrix**,** P**):**

n **=** len**(**matrix**)**

res **=** **[**sum**(**

**[**

P**[**j**]** **\*** **(-**sum**(**matrix**[**i**])** **+** matrix**[**i**][**i**])** **if** i **==** j **else** P**[**j**]** **\*** matrix**[**j**][**i**]** **for** j **in** range**(**n**)**

**]**

**)**

**for** i **in** range**(**n**)]**

**return** **[**i **\*** TIME\_DELTA **for** i **in** res**]**

**def** calcStabilizationTimes**(**matrix**,** start\_P**,** limit\_P**):**

n **=** len**(**matrix**)**

current\_time **=** 0

current\_P **=** start\_P**.**copy**()**

stabilizationTimes **=** **[**0 **for** i **in** range**(**n**)]**

total\_lambda\_sum **=** sum**([**sum**(**i**)** **for** i **in** matrix**])** **\*** MAGIC\_NUM

cool\_eps **=** **[**p**/**total\_lambda\_sum **for** p **in** limit\_P**]**

**while** **not** all**(**stabilizationTimes**):**

curr\_dps **=** dps**(**matrix**,** current\_P**)**

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

**if** **(not** stabilizationTimes**[**i**]** **and** curr\_dps**[**i**]** **<=** 1e-7 **and**

abs**(**current\_P**[**i**]** **-** limit\_P**[**i**])** **<=** cool\_eps**[**i**]):**

stabilizationTimes**[**i**]** **=** current\_time

current\_P**[**i**]** **+=** curr\_dps**[**i**]**

current\_time **+=** TIME\_DELTA

**return** stabilizationTimes

**def** calcPOverTime**(**matrix**,** start\_P**,** end\_time**):**

n **=** len**(**matrix**)**

current\_time **=** 0

current\_P **=** start\_P**.**copy**()**

POverTime **=** **[]**

times **=** **[]**

**while** current\_time **<** end\_time**:**

POverTime**.**append**(**current\_P**.**copy**())**

curr\_dps **=** dps**(**matrix**,** current\_P**)**

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

current\_P**[**i**]** **+=** curr\_dps**[**i**]**

current\_time **+=** TIME\_DELTA

times**.**append**(**current\_time**)**

**return** times**,** POverTime

**def** buildCoeffMatrix**(**matrix**):**

matrix **=** numpy**.**array**(**matrix**)**

n **=** len**(**matrix**)**

res **=** numpy**.**zeros**((**n**,** n**))**

**for** state **in** range**(**n **-** 1**):**

**for** col **in** range**(**n**):**

res**[**state**,** state**]** **-=** matrix**[**state**,** col**]**

**for** row **in** range**(**n**):**

res**[**state**,** row**]** **+=** matrix**[**row**,** state**]**

**for** state **in** range**(**n**):**

res**[**n **-** 1**,** state**]** **=** 1

**return** res

**def** buildAugmentationMatrix**(**count**):**

res **=** **[**0 **for** i **in** range**(**count**)]**

res**[**count **-** 1**]** **=** 1

**return** numpy**.**array**(**res**)**

**def** solve**(**matrix**):**

coeffMatrix **=** buildCoeffMatrix**(**matrix**)**

augmentationMatrix **=** buildAugmentationMatrix**(**len**(**matrix**))**

**return** numpy**.**linalg**.**solve**(**coeffMatrix**,** augmentationMatrix**)**

**def** graphPOverTime**(**P**,** stabilizationTime**,** times**,** POverTime**):**

**for** i\_node **in** range**(**len**(**POverTime**[**0**])):**

plt**.**plot**(**times**,** **[**i**[**i\_node**]** **for** i **in** POverTime**])**

plt**.**scatter**(**stabilizationTime**[**i\_node**],** P**[**i\_node**])**

plt**.**legend**([**'p{}'**.**format**(**i**+**1**)** **for** i **in** range**(**len**(**P**))])**

plt**.**xlabel**(**'time'**)**

plt**.**ylabel**(**'P'**)**

plt**.**show**()**

**def** random\_matrix**(**size**):**

**return** **[**

**[**round**(**random**.**random**(),** PRECISION**)** **if** i **!=** j **else** 0.0 **for** j **in** range**(**size**)]**

**for** i **in** range**(**size**)**

**]**

**def** output**(**title**,** caption**,** data**):**

**print(**title**)**

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

**print(**caption **+** str**(**i**),** round**(**fabs**(**data**[**i**]),** PRECISION**))**

**print()**

**def** getPreDefineI**(**i**):**

**if** i **==** 3**:**

**return** **[[**0**,** 2**,** 0**],**

**[**1**,** 0**,** 0**],**

**[**0**,** 1**,** 1**]]**

**if** i **==** 4**:**

**return** **[[**0**,** 2**,** 0**,** 0**],**

**[**0**,** 0**,** 2**,** 3**],**

**[**3**,** 0**,** 0**,** 1**],**

**[**0**,** 4**,** 0**,** 0**]]**

**elif** i **==** 5**:**

**return** **[[**0**,** 1**,** 1**,** 2**,** 0**],**

**[**0**,** 0**,** 0**,** 0**,** 0.5**],**

**[**0**,** 2**,** 0**,** 1**,** 0**],**

**[**0**,** 0**,** 3**,** 0**,** 1.5**],**

**[**2**,** 0**,** 0**,** 2**,** 0**]]**

**def** getStartP**(**n**,** all\_equal**=True):**

**if** all\_equal**:**

**return** **[**1**/**n**]** **\*** n

**else:**

res **=** **[**0**]** **\*** n

res**[**0**]** **=** 1

**return** res

**if** \_\_name\_\_ **==** '\_\_main\_\_'**:**

n **=** 5

#I = random\_matrix(n)

I **=** getPreDefineI**(**n**)**

start\_P **=** getStartP**(**n**,** **True)**

P **=** solve**(**I**)**

output**(**'P:'**,** 'p'**,** P**)**

stabilizationTime **=** calcStabilizationTimes**(**I**,** start\_P**,** P**)**

times**,** POverTime **=** calcPOverTime**(**I**,** start\_P**,** 5**)**

output**(**'T:'**,** 't'**,** stabilizationTime**)**

graphPOverTime**(**P**,** stabilizationTime**,** times**,** POverTime**)**