ЛАБОРАТОРНАЯ РАБОТА №6

«ИССЛЕДОВАНИЕ ДИНАМИЧЕСКИХ СЕТЕЙ БАЙЕСА»

1. Цель работы

Исследование методов точного и приближенного вероятностного вывода с использованием динамических сетей Байеса, приобретение навыков программирования интеллектуальных агентов, знания которых представляются условными высказываниями с определенной степенью уверенности.

2. Постановка задачи

В лабораторной работе необходимо создать Пакман-агента, который используют сенсоры для обнаружения невидимых призраков. Такой агент, кроме поиска одиночных призраков сможет охотиться с высокой эффективностью на группы из нескольких движущихся призраков.

**Задание 0 (0 баллов). Класс DiscreteDistribution**

Класс DiscreteDistribution, определенный в inference.py, используется для работы с дискретными распределениями. Этот класс является разновидностью словаря Python, где ключами являются дискретные элементы распределения, а значения ключей равны вероятностям (степени уверенности в возможном значении ключа).

В задании необходимо дописать недостающие методы этого класса: normalize и sample. Метод normalize нормализует значения распределения, таким образом, чтобы сумма всех значений была равна единице. Метод sample формирует случайную выборку из распределения в соответствии с алгоритмом, описанным п. 6.2.5.

**Задание 1 (2 балла). Вероятность наблюдения**

В этом задании необходимо реализовать метод getObservationProb базового класса InferenceModule, определяемого в файле inference.py. Метод должен принимать на вход наблюдение (зашумленное значение расстояния до призрака noisyDistance), позицию Пакмана pacmanPosition, позицию призрака ghostPosition, позицию тюремной камеры для призрака jailPosition и возвращать вероятность наблюдения noisyDistance для заданных положений Пакмана и призрака:

P(noisyDistance | pacmanPosition, ghostPosition).

По сути метод реализует модель наблюдения (восприятия) СММ.

**Задание 2 (3 балла). Точный вывод на основе наблюдений**

В этом задании необходимо реализовать метод observeUpdate класса ExactInference, определяемого в файле inference.py. Метод обновляет распределение степеней уверенности агента в отношении позиций призрака, оцениваемых на основе данных, поступающих от сенсоров Пакмана. Необходимо реализовать онлайн-обновление степеней уверенности в соответствии с (6.19) при получении нового наблюдения observation. Метод observeUpdate должен обновлять степени уверенности для каждой возможной позиции призрака после получения наблюдения. Необходимо циклически выполнять обновления для всех значений переменной self.allPositions, которая включает в себя все легальные позиции призрака, а также специальную тюремную позицию. Степени уверенности представляются вероятностями того, что призрак находится в определенной позиции, и хранятся в виде объекта DiscreteDistribution в поле с именем self.beliefs, которое необходимо обновлять.

**Задание 3 (3 балла). Точный вывод во времени**

В предыдущем задании было реализовано обновление распределения степеней доверия на основе наблюдений. К счастью, наблюдения Пакмана — не единственный источник информации о том, где может быть призрак. Пакман также знает, как может двигаться призрак, а именно, что призрак не может пройти сквозь стену или более чем через одну ячейку за один временной шаг.

Представим следующий сценарий, в котором имеется один призрак. Пакман получает серию наблюдений, которые указывают на то, что призрак очень близко, но затем поступает одно наблюдение, которое указывает, что призрак очень далеко. Наблюдение, указывающие на то, что призрак находится очень далеко, вероятно, является результатом сбоя сенсора. Предварительное знание Пакманом правил движения призрака может снизить влияние этого наблюдения, поскольку Пакман знает, что призрак не может далеко переместиться за один шаг.

В этом задании необходимо реализовать метод elapseTime класса ExactInference. Метод elapseTime должен обновлять степени доверия для каждой возможной новой позиции призрака по истечении одного временного шага в соответствии с (6.18). При этом агент имеет доступ к распределению действий призрака через self.getPositionDistribution.

**Задание 4 (2 балла). Полное тестирование точного вывода**

Теперь, когда Пакман знает, как использовать свои априорные знания о поведении призраков и свои наблюдения, он готов эффективно выслеживать призраков. В задании необходимо будет совместно использовать разработанные методы observUpdate и elapseTime, а также реализовать простую стратегию жадной охоты. В простой стратегии жадной охоты Пакман предполагает, что призрак находится в наиболее вероятной позиции поля игры в соответствии с его степенью уверенности, и поэтому он движется к ближайшему призраку. До этого момента Пакман выбирал допустимое действие случайно.

Реализуйте метод ChooseAction класса GreedyBustersAgent в файле bustersAgents.py. Ваш агент должен сначала найти наиболее вероятную позицию каждого непойманного призрака, а затем выбрать действие, которое ведет к ближайшему призраку. Чтобы найти расстояние между любыми двумя позициями pos1 и pos2, используйте метод self.distancer.getDistance(pos1, pos2). Чтобы найти следующую позицию после выполнения действия используйте вызов:

successorPosition = Actions.getSuccessor(position, action)

Вам предоставляется список LivingGhostPositionDistributions, элементы которого представляют распределения степеней уверенности о позициях каждого из еще непойманных призраков.

При правильной реализации ваш агент должен выиграть игру в тесте q4/3 gameScoreTest со счетом выше 700 очков как минимум в 8 из 10 раз.

**Задание 5 (2 балла). Инициализация приближенного вывода**

В нижеследующих заданиях (5,6 и 7) необходимо реализовать приближенный вероятностный вывод, основанный на алгоритме фильтрации частиц для отслеживания одного призрака.

В данном задании реализуйте методы initializeUniformly и getBeliefDistribution класса ParticleFilter в файле inference.py. Частица представляется позицией призрака. В результате применения метода initializeUniformly частицы должны быть равномерно (не случайным образом) распределены по допустимым позициям.

Метод getBeliefDistribution получает список частиц и отображает позиции частиц в соответствующее распределение вероятностей, представляемое в виде объекта DiscreteDistribution. Метод должен возвращать нормализованное распределение.

**Задание 6 (3 балла). Приближенный вывод: обновление на основе наблюдения**

Необходимо реализовать метод observеUpdate класса ParticleFilter в файле inference.py. Метод осуществляет обновление на основе наблюдения в соотвествии с алгоритом, описанным в п. 6.2.8. Наблюдение — это зашумленное манхеттенское расстояние до отслеживаемого призрака. Метод должен выполнять выборку из нормализованного распределения весов частиц и формировать новый список частиц self.particles. Вес частицы — это вероятность наблюдения с учетом положения Пакмана и местоположения частицы.

Имеется специальный случай, который необходимо учесть. Когда все частицы получают нулевой вес, список частиц следует повторно инициализировать, вызвав initializeUniformly.

**Задание 7 (3 балла). Приближенный вывод: обновление во времени**

Реализуйте метод elapseTime класса ParticleFilter в файле inference.py. Метод должен сформировать новый список частиц self.particles с учетом изменения состояний игры во времени. Используйте алгоритм обновления во времени, описанный в п. 6.2.8.

**Задание 8 (1 балл). Инициализация при совместной фильтрации частиц**

В задании рассматривается случай, когда имеется несколько призраков. Поскольку модели перехода призраков больше не являются независимыми, все призраки должны отслеживаться совместно с использованием динамической сети Байеса (ДСБ), которая является обобщением СММ.

ДСБ с двумя призраками (a и b) изображена на рисунке 6.4. На рисунке скрытые переменные G представляют положения призраков, а переменные свидетельств E представляют собой зашумленные расстояния до каждого из призраков. Представленную структуру ДСБ можно распространить на большее количество призраков.

В заданиях ниже необходимо реализовать алгоритм вывода, основанный на фильтрации частиц, который одновременно отслеживает несколько призраков.

Каждая частица (полная выборка на временном шаге) представляется кортежем позиций призраков, показывающим, где призраки находятся в данный момент.

Предоставляемый вам программный код уже подготовлен для извлечения маргинальных распределений по каждому призраку с помощью алгоритма совместного отслеживания призраков, который вы реализуете.

В данном задании завершите определение метода initializeUniformly класса JointParticleFilter в файле inference.py. Метод должен обеспечить начальное равномерное распределение частиц. Как и в задании 5, частицы хранятся в списке частиц self.particles.

**Задание 9 (3 балла). Обновление на основе наблюдения при совместной фильтрации частиц**

В задании необходимо реализовать метод оbservUpdate класса JointParticleFilter файла inference.py. Метод должен обеспечивать взвешивание и повторное сэмплирование всех частиц с учетом правдоподобия наблюдаемого расстояния до каждого из призраков. Метод аналогичен одноименному методу класса ParticleFilter, но обеспечивает обработку наблюдений для нескольких призраков.

Также реализация метода должна обрабатывать особый случай, когда все частицы получают нулевой вес. В этом случае список частиц self.particles следует воссоздать из априорного распределения, вызвав initializeUniformly.

**Задание 10 (3 балла). Обновление во времени при совместной фильтрации частиц**

В задании необходимо завершить определение метода elapseTime класса JointParticleFilter в файле inference.py, чтобы корректно выполнять ресэмплирование частиц в ДСБ совместного отслеживания призраков. В частности, необходимо учитывать, что каждый призрак перемещается в новую позицию, обусловленную позициями всех призраков на предыдущем временном шаге.

3. Ход работы

**Задание 0 (0 баллов). Класс DiscreteDistribution**

Класс DiscreteDistribution, определенный в inference.py, используется для работы с дискретными распределениями. Этот класс является разновидностью словаря Python, где ключами являются дискретные элементы распределения, а значения ключей равны вероятностям (степени уверенности в возможном значении ключа).

В задании необходимо дописать недостающие методы этого класса: normalize и sample. Метод normalize нормализует значения распределения, таким образом, чтобы сумма всех значений была равна единице. Метод sample формирует случайную выборку из распределения в соответствии с алгоритмом, описанным п. 6.2.5

Листинг 1 – Код класса DiscreteDistribution

class DiscreteDistribution(dict):

"""

Класс для работы с распределением,

представляемым в виде словаря c

набором значений ключей и соответсвующих вероятностей

"""

def \_\_getitem\_\_(self, key):

self.setdefault(key, 0)

return dict.\_\_getitem\_\_(self, key)

def copy(self):

"""

Возвращает копию распределения

"""

return DiscreteDistribution(dict.copy(self))

def argMax(self):

"""

Возвращает ключ с наибольшим значением

"""

if len(self.keys()) == 0:

return None

all = list(self.items())

values = [x[1] for x in all]

maxIndex = values.index(max(values))

return all[maxIndex][0]

def total(self):

"""

Возвращает сумму всех значений вероятностей

"""

return float(sum(self.values()))

def normalize(self):

"""

Нормализуйте распределение таким образом, чтобы суммарное значение

всех вероятностей ключей равнялось 1. Сотношение значений для всех

ключей должно остаться прежним. В случае, когда суммарное значение

равно 0, ничего не делайте.

Тесты:

>>> dist = DiscreteDistribution()

>>> dist['a'] = 1

>>> dist['b'] = 2

>>> dist['c'] = 2

>>> dist['d'] = 0

>>> dist.normalize()

>>> list(sorted(dist.items()))

[('a', 0.2), ('b', 0.4), ('c', 0.4), ('d', 0.0)]

>>> dist['e'] = 4

>>> list(sorted(dist.items()))

[('a', 0.2), ('b', 0.4), ('c', 0.4), ('d', 0.0), ('e', 4)]

>>> empty = DiscreteDistribution()

>>> empty.normalize()

>>> empty

{}

"""

"\*\*\* ВСТАВЬТЕ ВАШ КОД СЮДА \*\*\*"

total = float(self.total())

if total !=0:

for key in self.keys():

self[key]=self[key]/ total

# raiseNotDefined()

def sample(self):

"""

Формирует случайную выборку по распределению, представляемому

в виде словаря, и возвращает ключ,

соответствующий случайной выборке.

Тесты:

>>> dist = DiscreteDistribution()

>>> dist['a'] = 1

>>> dist['b'] = 2

>>> dist['c'] = 2

>>> dist['d'] = 0

>>> N = 100000.0

>>> samples = [dist.sample() for \_ in range(int(N))]

>>> round(samples.count('a') \* 1.0/N, 1) # proportion of 'a'

0.2

>>> round(samples.count('b') \* 1.0/N, 1)

0.4

>>> round(samples.count('c') \* 1.0/N, 1)

0.4

>>> round(samples.count('d') \* 1.0/N, 1)

0.0

"""

"\*\*\* ВСТАВЬТЕ ВАШ КОД СЮДА \*\*\*"

items = sorted (self.items())

distribution = [i[1] for i in items]

item\_values = [i[0] for i in items]

random\_choice =random.random()

i,total =0,distribution[0]

#

if self.total()!=1:

self.normalize()

while random\_choice > total:

i +=1

total +=distribution[i]

return item\_values[i]

#raiseNotDefined()

Были проведены тесты с помощью автооценивателя. Результаты на рисунке 1.

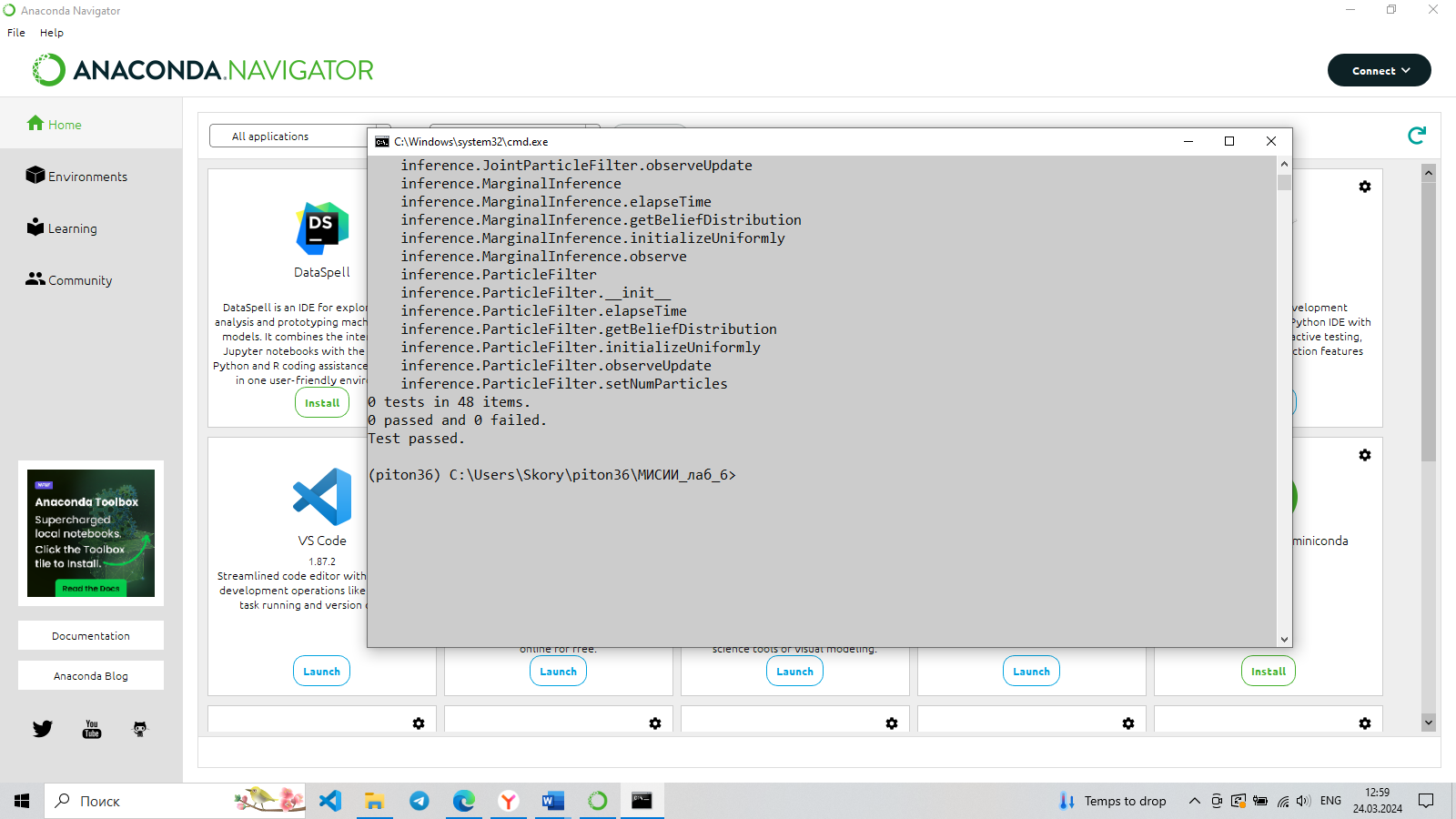


Рисунок 1 – Прохождения тестов для задания 0

**Задание 1 (2 балла). Вероятность наблюдения**

В этом задании необходимо реализовать метод getObservationProb базового класса InferenceModule, определяемого в файле inference.py. Метод должен принимать на вход наблюдение (зашумленное значение расстояния до призрака noisyDistance), позицию Пакмана pacmanPosition, позицию призрака ghostPosition, позицию тюремной камеры для призрака jailPosition и возвращать вероятность наблюдения noisyDistance для заданных положений Пакмана и призрака:

P(noisyDistance | pacmanPosition, ghostPosition).

По сути метод реализует модель наблюдения (восприятия) СММ.

Листинг 2 – Код метода getObservationProb

def getObservationProb(self, noisyDistance, pacmanPosition, ghostPosition, jailPosition):

if ghostPosition == jailPosition:

# 。 If the noise distance between Pacman and the ghost is None, the probability of a noise distance being None is 1

if noisyDistance == None:

return 1.0

# , Pacman and ghost noise distance is not None, the probability is 0

else:

return 0.0

#If the ghost is not in the ghost jail, the probability that the noise distance between Pacman and the ghost is None is 0.

if noisyDistance == None :

return 0.0

#If the noise distance between Pacman and the ghost is not None, obtain the probabilityDistributions[noisyDistance][trueDistance] probability distribution value according to the noise distance and the actual distance.

return busters.getObservationProbability(noisyDistance, manhattanDistance(pacmanPosition, ghostPosition))

С помощью автооценивания была проверена правильность написания кода. Результат представлен на рисунке 2.

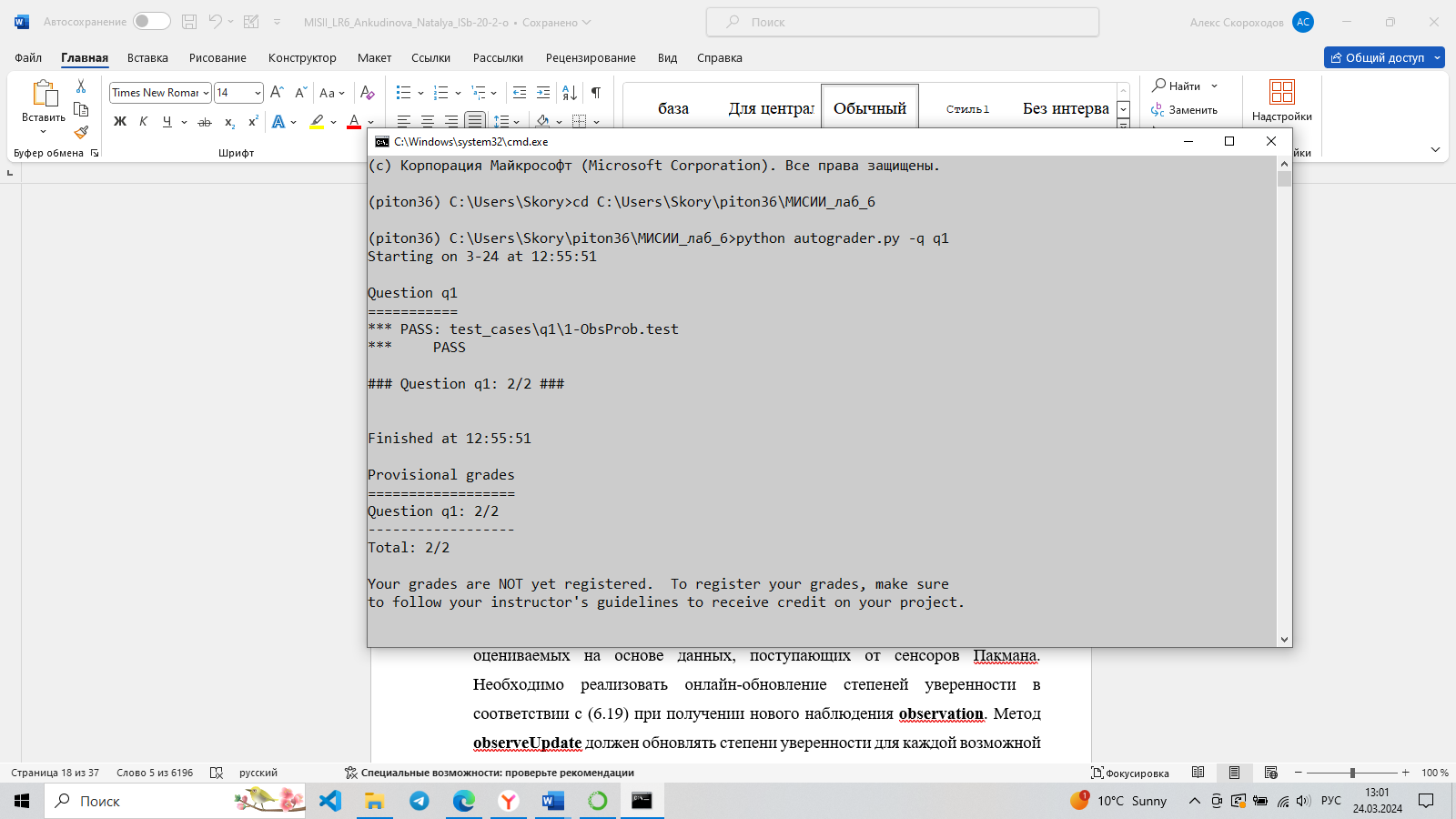


Рисунок 2 – Прохождение тестов для задания 1

**Задание 2 (3 балла). Точный вывод на основе наблюдений**

В этом задании необходимо реализовать метод observeUpdate класса ExactInference, определяемого в файле inference.py. Метод обновляет распределение степеней уверенности агента в отношении позиций призрака, оцениваемых на основе данных, поступающих от сенсоров Пакмана. Необходимо реализовать онлайн-обновление степеней уверенности в соответствии с (6.19) при получении нового наблюдения observation. Метод observeUpdate должен обновлять степени уверенности для каждой возможной позиции призрака после получения наблюдения. Необходимо циклически выполнять обновления для всех значений переменной self.allPositions, которая включает в себя все легальные позиции призрака, а также специальную тюремную позицию. Степени уверенности представляются вероятностями того, что призрак находится в определенной позиции, и хранятся в виде объекта DiscreteDistribution в поле с именем self.beliefs, которое необходимо обновлять.

Листинг 3 – Код метода observeUpdate

def observeUpdate(self, observation, gameState):

pacmanPosition=gameState.getPacmanPosition()

jailPosition=self.getJailPosition()

positions=self.allPositions

noisyDistance=observation

for possibleGhostPos in positions:

self.beliefs[possibleGhostPos] = self.getObservationProb(noisyDistance, pacmanPosition, possibleGhostPos, jailPosition)\*self.beliefs[possibleGhostPos]

self.beliefs.normalize()

self.beliefs.normalize()

С помощью автооценивания была проверена правильность написания кода. Результат представлен на рисунке 3.

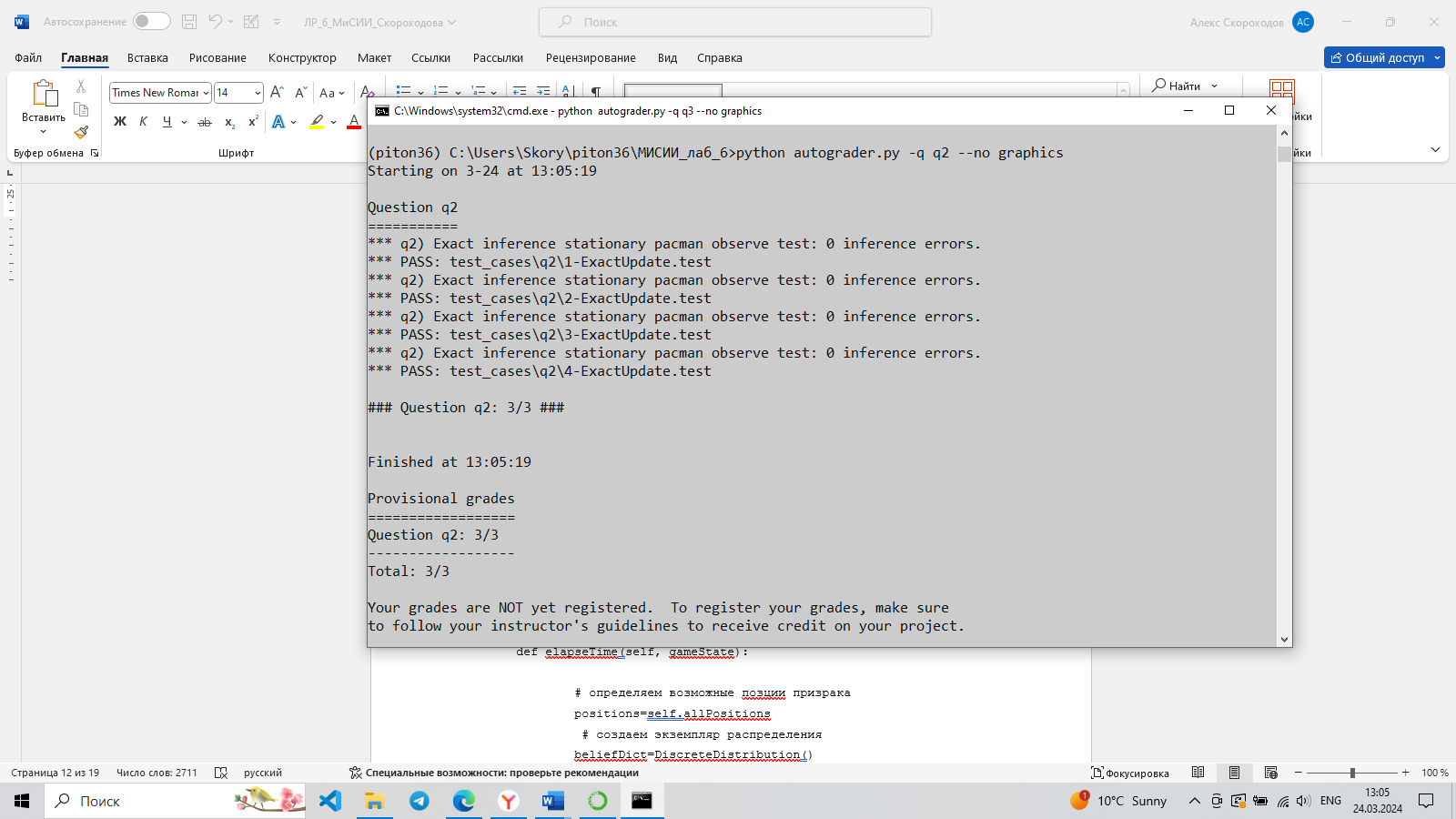


Рисунок 3 – Прохождение тестов для задания 2

**Задание 3 (3 балла). Точный вывод во времени**

В предыдущем задании было реализовано обновление распределения степеней доверия на основе наблюдений. К счастью, наблюдения Пакмана — не единственный источник информации о том, где может быть призрак. Пакман также знает, как может двигаться призрак, а именно, что призрак не может пройти сквозь стену или более чем через одну ячейку за один временной шаг.

Представим следующий сценарий, в котором имеется один призрак. Пакман получает серию наблюдений, которые указывают на то, что призрак очень близко, но затем поступает одно наблюдение, которое указывает, что призрак очень далеко. Наблюдение, указывающие на то, что призрак находится очень далеко, вероятно, является результатом сбоя сенсора. Предварительное знание Пакманом правил движения призрака может снизить влияние этого наблюдения, поскольку Пакман знает, что призрак не может далеко переместиться за один шаг.

В этом задании необходимо реализовать метод elapseTime класса ExactInference. Метод elapseTime должен обновлять степени доверия для каждой возможной новой позиции призрака по истечении одного временного шага в соответствии с (6.18). При этом агент имеет доступ к распределению действий призрака через self.getPositionDistribution.

Листинг 4 – Код метода elapseTime

def elapseTime(self, gameState):

# определяем возможные позции призрака

positions=self.allPositions

# создаем экземпляр распределения

beliefDict=DiscreteDistribution()

# выполняем итерации по всем возможным позициям призрака

for ghostPos in positions:

# определяем распределение новых позиций призрака

# по предыдущей позиции ghostPos

newPosDist = self.getPositionDistribution(gameState, ghostPos)

# для всех элементов распределения newPosDist

for newPos, prob in newPosDist.items():

# обновляем степени доверия возможных новых позиций

beliefDict[newPos]=beliefDict[newPos]+self.beliefs[ghostPos]\*prob

# нормализуем распределение

beliefDict.normalize

# сохраняем обновленное распределение

self.beliefs=beliefDict

Были проведены тесты с помощью автооценивателя. Результаты на рисунке 4.

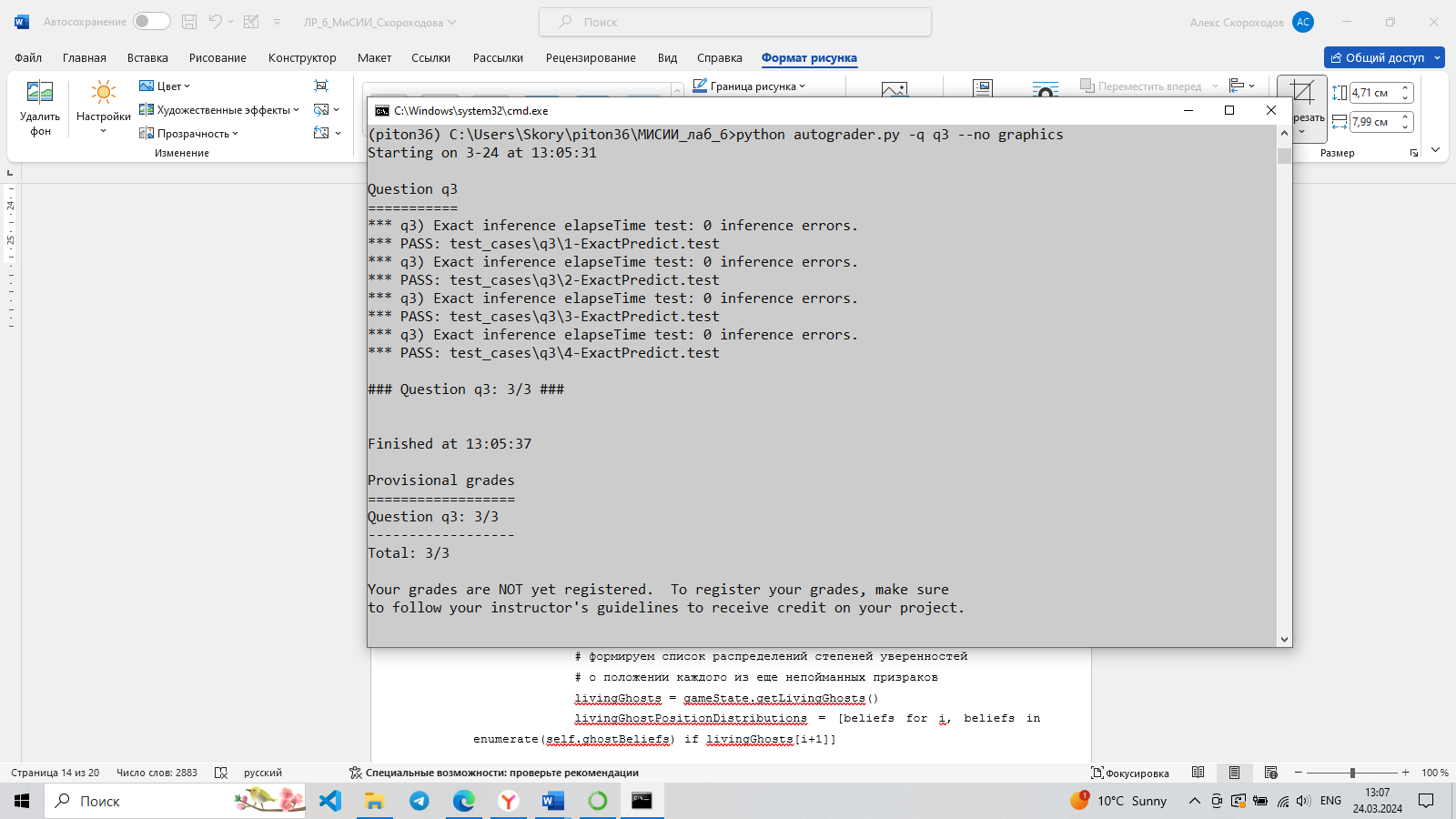


Рисунок 4 – Прохождения тестов для задания 3

**Задание 4 (2 балла). Полное тестирование точного вывода**

Теперь, когда Пакман знает, как использовать свои априорные знания о поведении призраков и свои наблюдения, он готов эффективно выслеживать призраков. В задании необходимо будет совместно использовать разработанные методы observUpdate и elapseTime, а также реализовать простую стратегию жадной охоты. В простой стратегии жадной охоты Пакман предполагает, что призрак находится в наиболее вероятной позиции поля игры в соответствии с его степенью уверенности, и поэтому он движется к ближайшему призраку. До этого момента Пакман выбирал допустимое действие случайно.

Реализуйте метод ChooseAction класса GreedyBustersAgent в файле bustersAgents.py. Ваш агент должен сначала найти наиболее вероятную позицию каждого непойманного призрака, а затем выбрать действие, которое ведет к ближайшему призраку. Чтобы найти расстояние между любыми двумя позициями pos1 и pos2, используйте метод self.distancer.getDistance(pos1, pos2). Чтобы найти следующую позицию после выполнения действия используйте вызов:

successorPosition = Actions.getSuccessor(position, action)

Вам предоставляется список LivingGhostPositionDistributions, элементы которого представляют распределения степеней уверенности о позициях каждого из еще непойманных призраков.

При правильной реализации ваш агент должен выиграть игру в тесте q4/3- gameScoreTest со счетом выше 700 очков как минимум в 8 из 10 раз.

Листинг 5 – Код метода ChooseAction класса GreedyBustersAgent

def chooseAction(self, gameState):

"""

Сначала вычисляет наиболее вероятную позицию каждого призрака,

который еще не был пойман. Затем выбирает действие, перемещающее

Пакмана к ближайшему призраку (в соответствии с mazeDistance).

"""

# определяем позицию Пакмана

pacmanPosition = gameState.getPacmanPosition()

# формируем список допустимых действий Пакмана

legal = [a for a in gameState.getLegalPacmanActions()]

# формируем список распределений степеней уверенностей

# о положении каждого из еще непойманных призраков

livingGhosts = gameState.getLivingGhosts()

livingGhostPositionDistributions = [beliefs for i, beliefs in enumerate(self.ghostBeliefs) if livingGhosts[i+1]]

"\*\*\* ВСТАВЬТЕ ВАШ КОД СЮДА\*\*\*"

ghostMaxProb=[]

# итерации по всем призракам

for g in livingGhostPositionDistributions:

# создаем список наиболее вероятных позиций призраков

ghostMaxProb.append(g.argMax())

minDist=[]

# итерации по всем допустимым действиям Пакмана

for action in legal:

# находим следующую позицию после действия action

successorPosition = Actions.getSuccessor(pacmanPosition, action)

# для всех наиболее вероятных позиций призраков из ghostMaxProb

for ghostPos in ghostMaxProb:

# создаем список расстояний от Пакмана до призрака

minDist.append((action, self.distancer.getDistance(successorPosition, ghostPos)))

# находим минимальное расстояние до призрака

minGhostDist=min([d for act, d in minDist])

# находим действие act ведущее в сторону ближайшего призрака

for act, d in minDist:

if d==minGhostDist:

return act

Были проведены тесты с помощью автооценивателя. Результаты на рисунке 5.

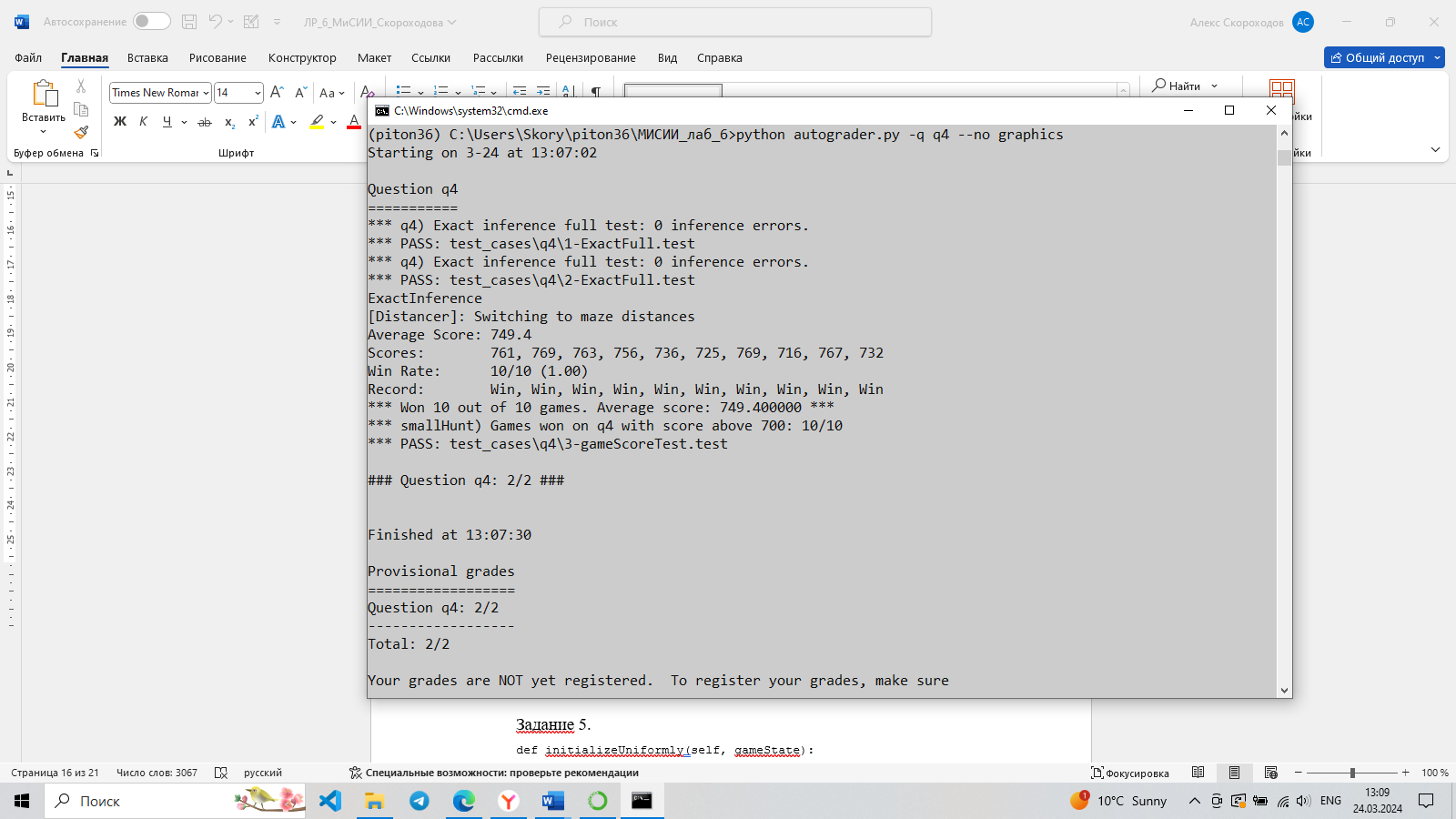


Рисунок 5 – Прохождения тестов для задания 4

**Задание 5 (2 балла). Инициализация приближенного вывода**

В нижеследующих заданиях (5,6 и 7) необходимо реализовать приближенный вероятностный вывод, основанный на алгоритме фильтрации частиц для отслеживания одного призрака.

В данном задании реализуйте методы initializeUniformly и getBeliefDistribution класса ParticleFilter в файле inference.py. Частица представляется позицией призрака. В результате применения метода initializeUniformly частицы должны быть равномерно (не случайным образом) распределены по допустимым позициям.

Метод getBeliefDistribution получает список частиц и отображает позиции частиц в соответствующее распределение вероятностей, представляемое в виде объекта DiscreteDistribution. Метод должен возвращать нормализованное распределение.

Листинг 6 – Код класса ParticleFilter

def initializeUniformly(self, gameState):

self.particles = []

num = 0

while num < self.numParticles:

for p in self.legalPositions:

if num >= self.numParticles: break

self.particles.append(p)

num += 1

def getBeliefDistribution(self):

answer = util.Counter()

for particle in self.particles:

answer[particle] += 1

answer.normalize()

return answer

Были проведены тесты с помощью автооценивателя. Результаты на рисунке 6.

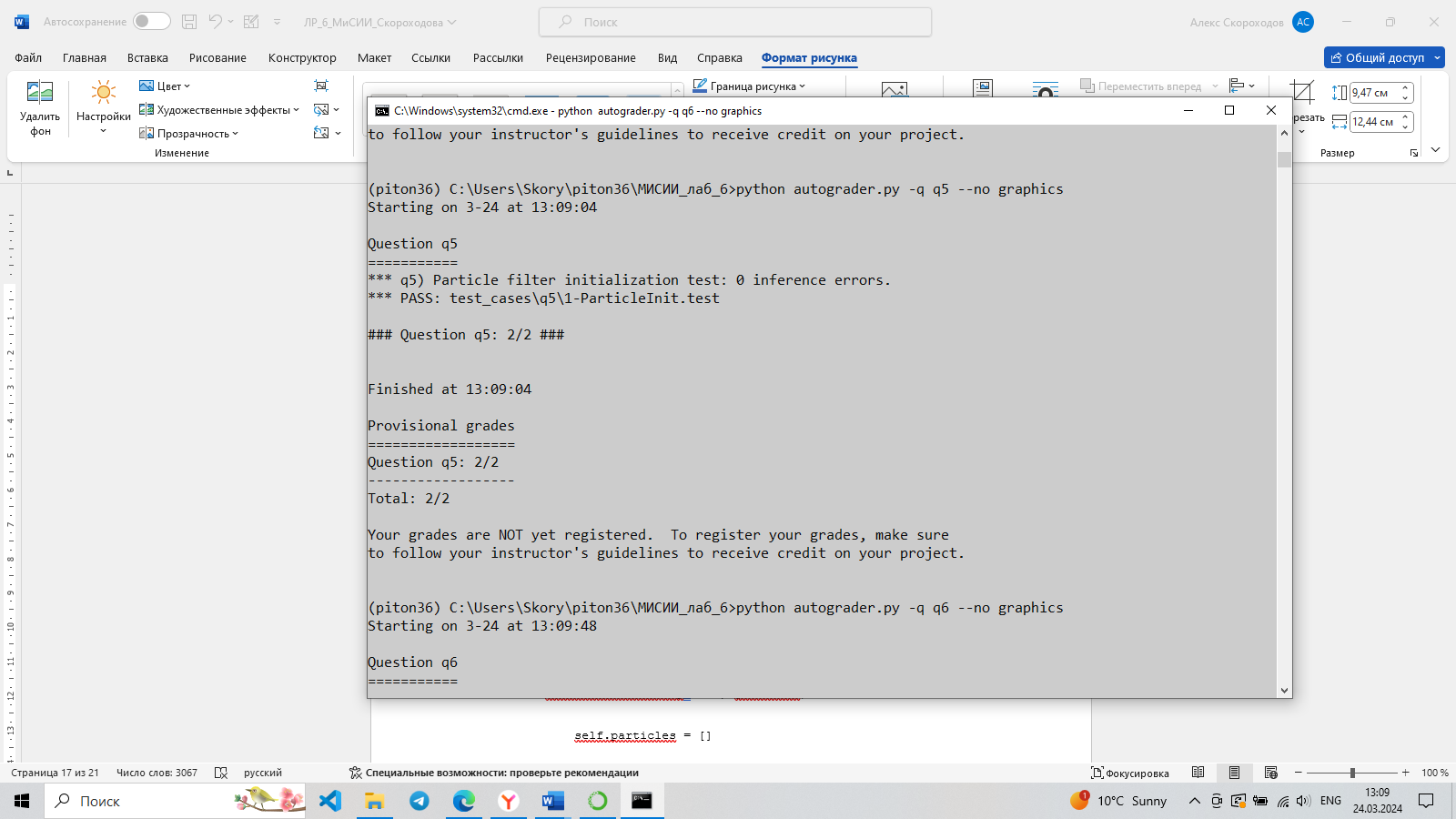


Рисунок 6 – Прохождения тестов для задания 5

**Задание 6 (3 балла). Приближенный вывод: обновление на основе наблюдения**

Необходимо реализовать метод observUpdate класса ParticleFilter в файле inference.py. Метод осуществляет обновление на основе наблюдения в соответствии с алгоритмом, описанным в п. 6.2.8. Наблюдение - это зашумленное манхеттенское расстояние до отслеживаемого призрака. Метод должен выполнять выборку из нормализованного распределения весов частиц и формировать новый список частиц self.particles. Вес частицы — это вероятность наблюдения с учетом положения Пакмана и местоположения частицы.

Имеется специальный случай, который необходимо учесть. Когда все частицы получают нулевой вес, список частиц следует повторно инициализировать, вызвав initializeUniformly.

Листинг 7 – Код метода observUpdate

def observeUpdate(self, observation, gameState):

weights=DiscreteDistribution()

resample=[]

# определяем позиции Пакмана и тюрьмы

pacmanPosition=gameState.getPacmanPosition()

jailPosition=self.getJailPosition()

# для каждой позиции частицы

for pos in self.particles:

# определяем степень уверенности наблюдения при заданных

# pacmanPosition, pos, jailPosition и аккумулируем в виде веса

weights[pos]+=self.getObservationProb(observation, pacmanPosition, pos, jailPosition)

# если частицы получают нулевой вес

if weights.total()==0:

# то инициализаируем повторно список частиц

self.initializeUniformly(gameState)

# иначе

else:

# нормазизуем распределение весов

weights.normalize()

# формируем список частиц путем выборки из распределния весов

self.particles = [weights.sample() for \_ in range(int(self.numParticles))]

#raiseNotDefined()

Были проведены тесты с помощью автооценивателя. Результаты на рисунке 7.

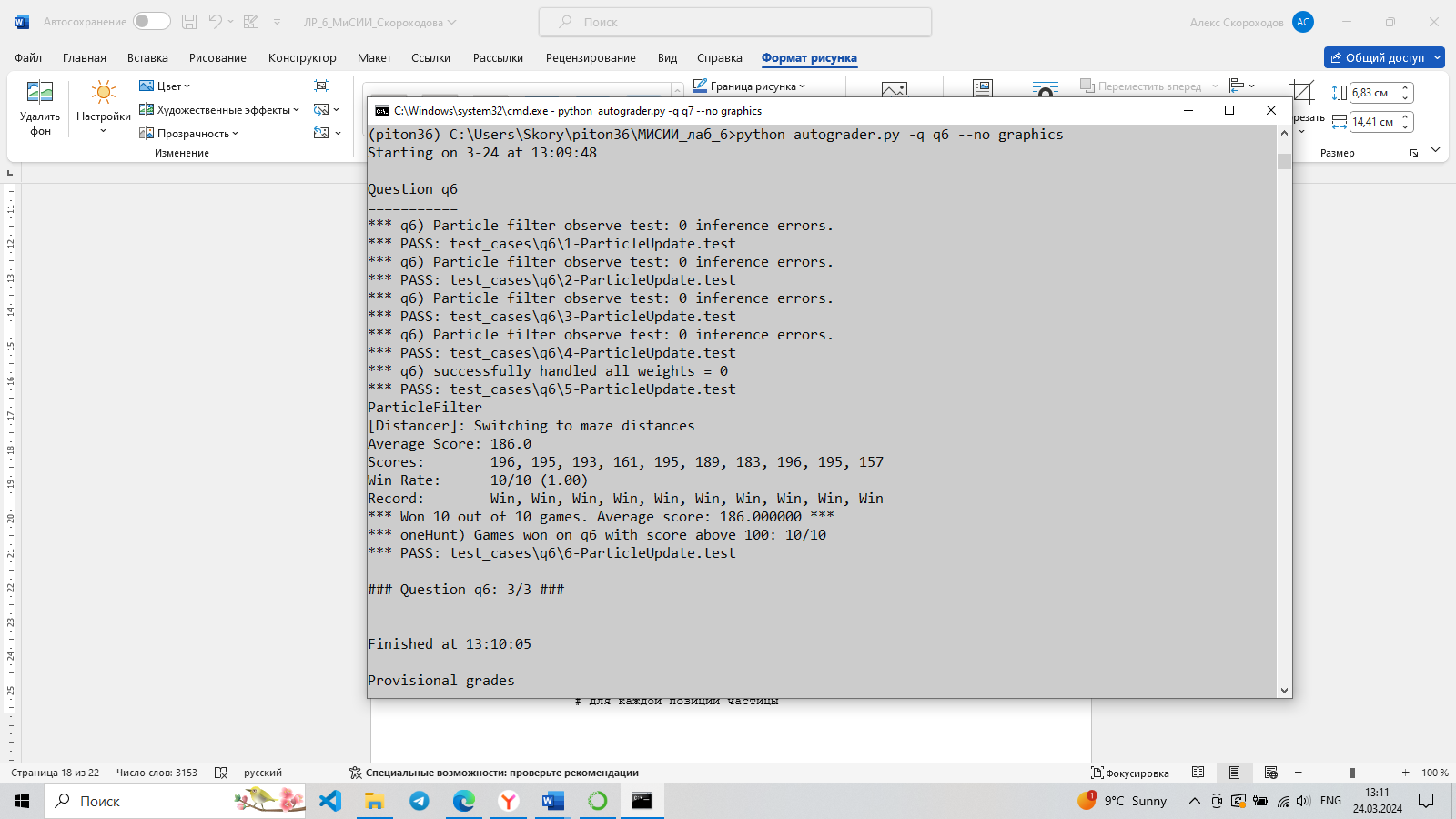


Рисунок 7 – Прохождения тестов для задания 6

**Задание 7 (3 балла). Приближенный вывод: обновление во времени**

Реализуйте метод elapseTime класса ParticleFilter в файле inference.py. Метод должен сформировать новый список частиц self.particles с учетом изменения состояний игры во времени. Используйте алгоритм обновления во времени, описанный в п. 6.2.8.

Листинг 8 – Код метода elapseTime

def elapseTime(self, gameState):

elapseDist=DiscreteDistribution()

# определяем выборку в виде списка позиций частиц

sample=self.particles

# для каждой позиции частицы

for pos in sample:

# находим распределение в следующей возможной позиции

newPosDist = self.getPositionDistribution(gameState, pos)

# для всех элементов распределения newPosDist

for newPos, prob in newPosDist.items():

# обновляем степени доверия возможных новых позиций

elapseDist[newPos]+=prob

# нормализуем распределение

elapseDist.normalize()

# формируем новый список частиц путем выборки из распределния

self.particles=[elapseDist.sample() for \_ in range(int(self.numParticles))]

#raiseNotDefined()

Были проведены тесты с помощью автооценивателя. Результаты на рисунке 8.

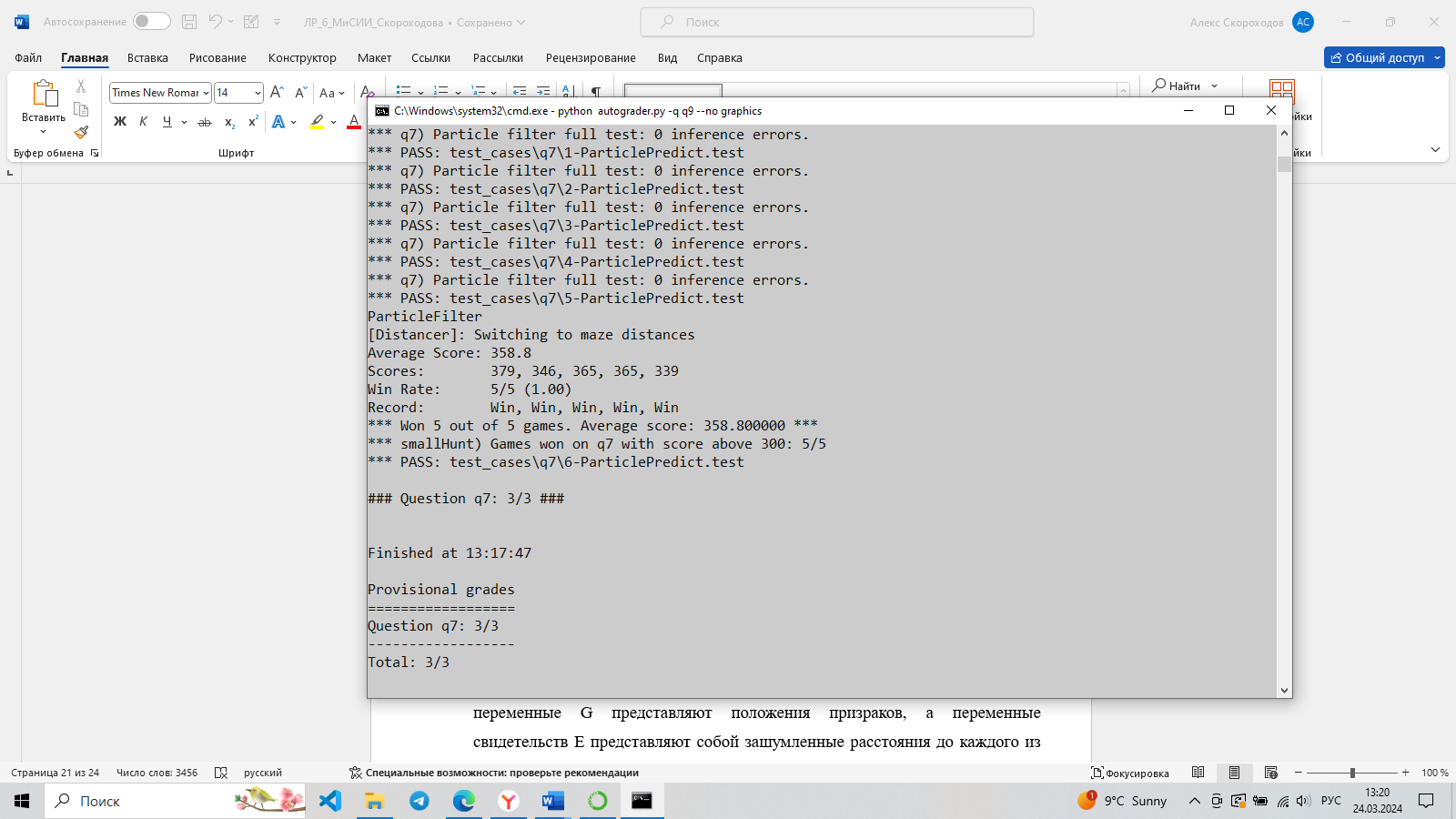


Рисунок 8 – Прохождения тестов для задания 7

**Задание 8 (1 балл). Инициализация при совместной фильтрации частиц**

В задании рассматривается случай, когда имеется несколько призраков. Поскольку модели перехода призраков больше не являются независимыми, все призраки должны отслеживаться совместно с использованием динамической сети Байеса (ДСБ), которая является обобщением СММ. ДСБ с двумя призраками (a и b) изображена на рисунке 6.4. На рисунке скрытые переменные G представляют положения призраков, а переменные свидетельств E представляют собой зашумленные расстояния до каждого из призраков. Представленную структуру ДСБ можно распространить на большее количество призраков.

В заданиях ниже необходимо реализовать алгоритм вывода, основанный на фильтрации частиц, который одновременно отслеживает несколько призраков. Каждая частица (полная выборка на временном шаге) представляется кортежем позиций призраков, показывающим, где призраки находятся в данный момент. Предоставляемый вам программный код уже подготовлен для извлечения маргинальных распределений по каждому призраку с помощью алгоритма совместного отслеживания призраков, который вы реализуете.

В данном задании завершите определение метода initializeUniformly класса JointParticleFilter в файле inference.py. Метод должен обеспечить начальное равномерное распределение частиц. Как и в задании 5, частицы хранятся в списке частиц self.particles.

Листинг 9 – Код метода initializeUniformly

def initializeUniformly(self, gameState):

#self.particles = []

self.particles = []

permutations = list(itertools.product(self.legalPositions, repeat=self.numGhosts))

random.shuffle(permutations)

size = self.numParticles

i = 0

while i < size:

for particle in permutations:

self.particles.append(particle)

i += 1

Были проведены тесты с помощью автооценивателя. Результаты на рисунке 9.

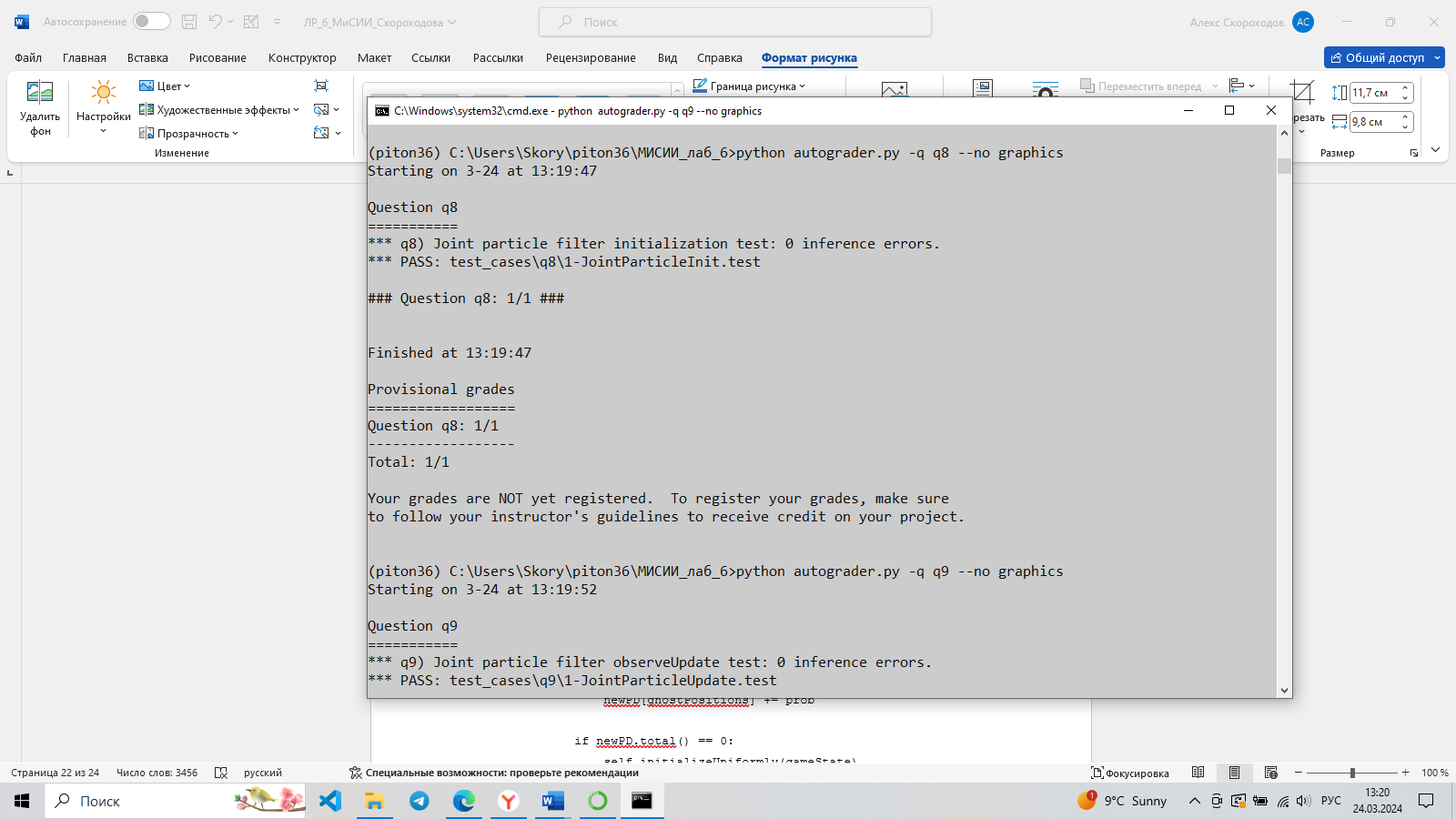


Рисунок 9 – Прохождения тестов для задания 8

**Задание 9 (3 балла). Обновление на основе наблюдения при совместной фильтрации частиц**

В задании необходимо реализовать метод оbservUpdate класса JointParticleFilter файла inference.py. Метод должен обеспечивать взвешивание и повторное сэмплирование всех частиц с учетом правдоподобия наблюдаемого расстояния до каждого из призраков. Метод аналогичен одноименному методу класса ParticleFilter, но обеспечивает обработку наблюдений для нескольких призраков.

Также реализация метода должна обрабатывать особый случай, когда все частицы получают нулевой вес. В этом случае список частиц self.particles следует воссоздать из априорного распределения, вызвав initializeUniformly.

Листинг 10 – Код метода оbserveUpdate

def observeUpdate(self, observation, gameState):

pacmanPosition = gameState.getPacmanPosition()

newPD = DiscreteDistribution()

for ghostPositions in self.particles:

prob = 1

for i in range(self.numGhosts):

prob \*= self.getObservationProb(observation[i], pacmanPosition, ghostPositions[i],self.getJailPosition(i))

newPD[ghostPositions] += prob

if newPD.total() == 0:

self.initializeUniformly(gameState)

else:

newPD.normalize()

self.particles = [newPD.sample() for \_ in range(self.numParticles)]

Были проведены тесты с помощью автооценивателя. Результаты на рисунке 10.

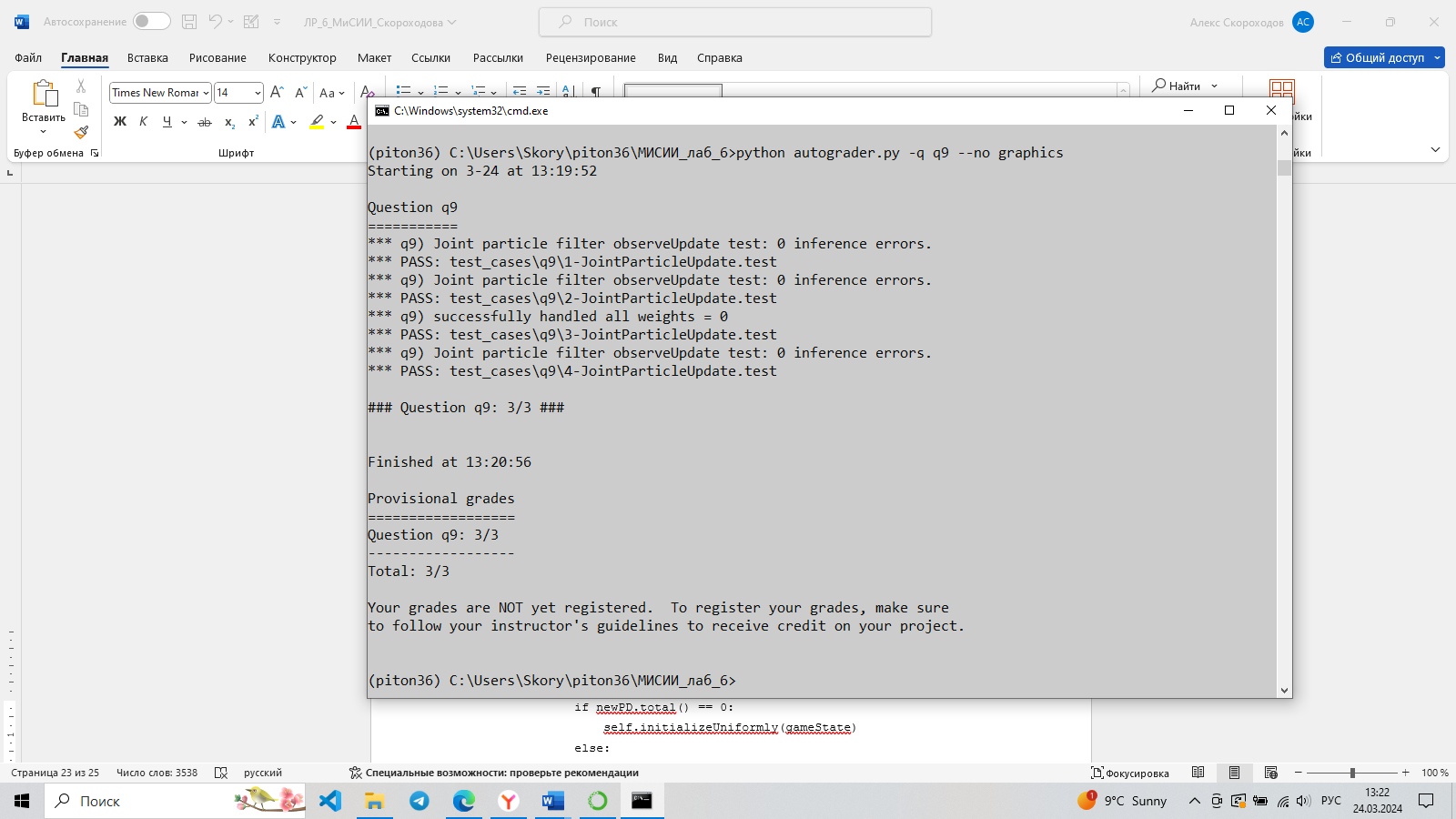


Рисунок 10 – Прохождения тестов для задания 9

**Задание 10 (3 балла). Обновление во времени при совместной фильтрации частиц**

В задании необходимо завершить определение метода elapseTime класса JointParticleFilter в файле inference.py, чтобы корректно выполнять ресэмплирование частиц в ДСБ совместного отслеживания призраков. В частности, необходимо учитывать, что каждый призрак перемещается в новую позицию, обусловленную позициями всех призраков на предыдущем временном шаге.

Листинг 11 – Код метода elapseTime

def elapseTime(self, gameState):

"""

Sample each particle's next state based on its current state and the

gameState.

"""

newParticles = []

for oldParticle in self.particles:

newParticle = list(oldParticle) # A list of ghost positions

# now loop through and update each entry in newParticle...

"\*\*\* YOUR CODE HERE \*\*\*"

for i in range(self.numGhosts):

newPosDist = self.getPositionDistribution(gameState, newParticle, i, self.ghostAgents[i])

newParticle[i] = newPosDist.sample()

"""\*\*\* END YOUR CODE HERE \*\*\*"""

newParticles.append(tuple(newParticle))

self.particles = newParticles

Были проведены тесты с помощью автооценивателя. Результаты на рисунке 11.

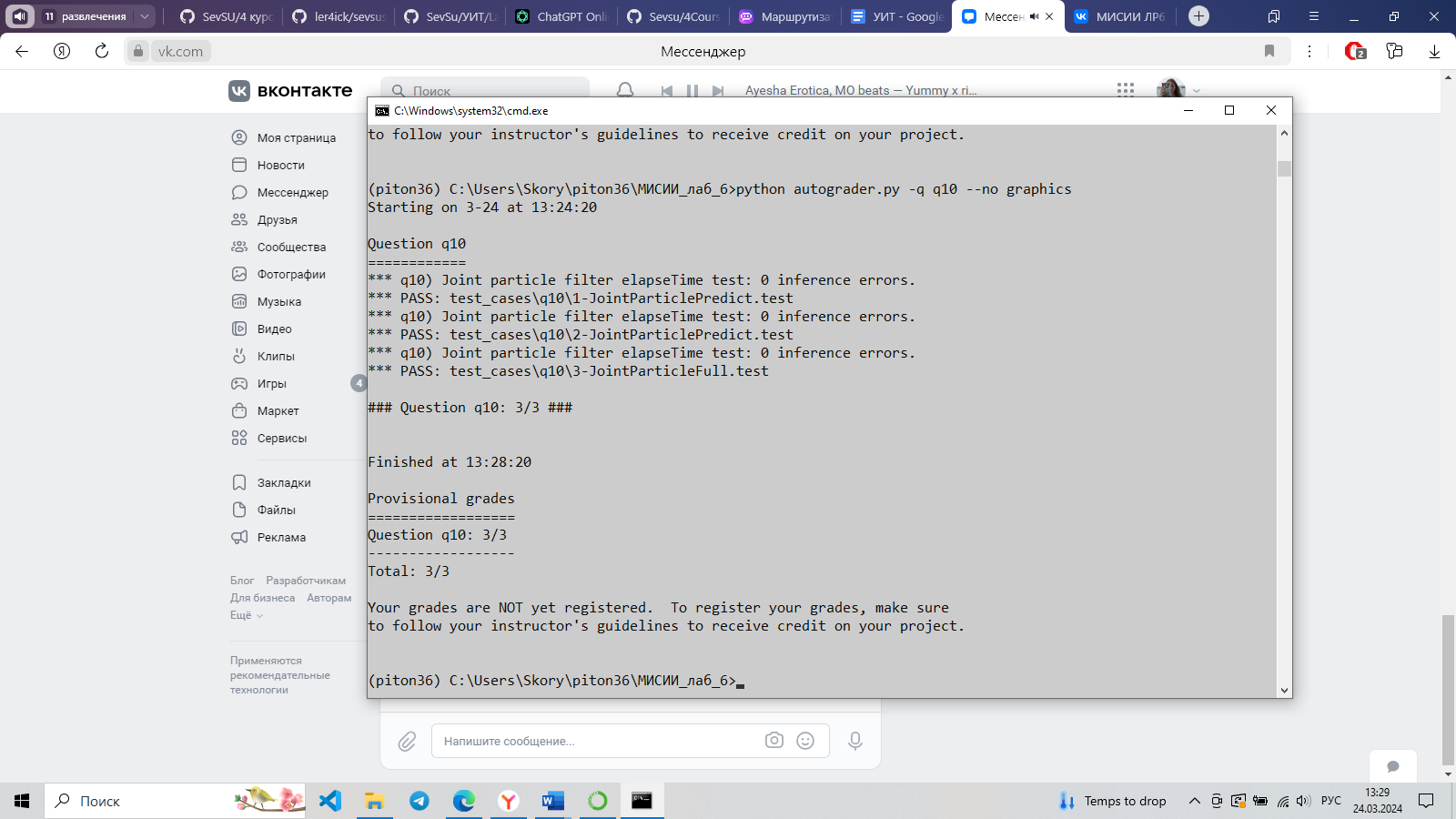


Рисунок 11 – Прохождения тестов для задания 10

4. Выводы

В ходе выполнения лабораторной работы были исследованы методы точного и приближенного вероятностного вывода с использованием динамических сетей Байеса, приобретены навыки программирования интеллектуальных агентов, знания которых представляются условными высказываниями с определенной степенью уверенности.

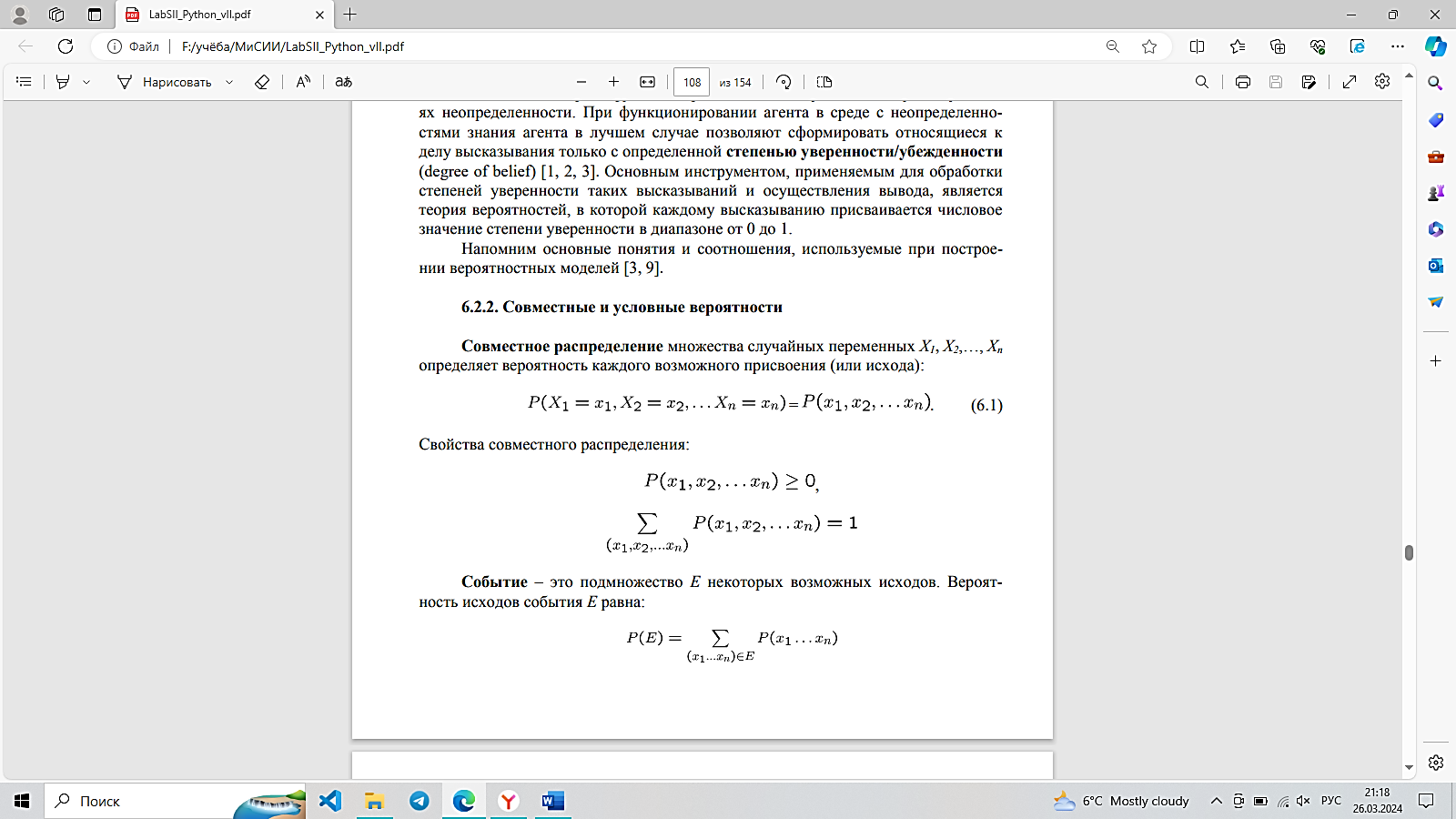
**5. Контрольные вопросы**

**5.1 Объясните, что понимают под степенью уверенности высказываний?**

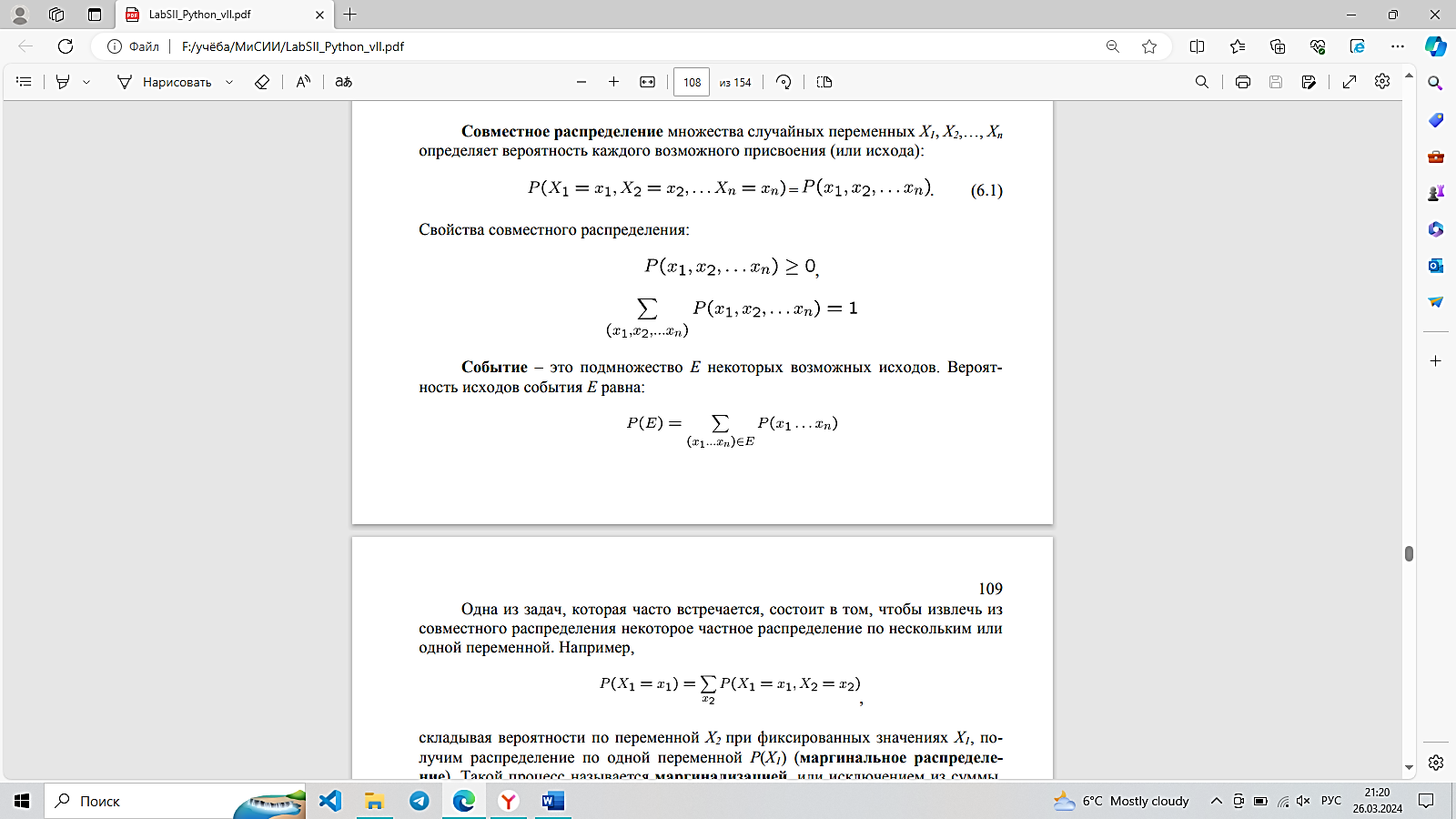
Степень уверенности высказывания указывает на ***уровень доверия*** или ***вероятность того, что данное высказывание является истинным или верным.*** Она представляет собой меру убежденности в правильности или достоверности информации, выраженной в высказывании.

Основным инструментом, применяемым для обработки степеней уверенности таких высказываний и осуществления вывода, является теория вероятностей, в которой каждому высказыванию присваивается числовое значение степени уверенности в диапазоне от 0 до 1.

**5.2. Что понимают под совместным распределением случайных переменных? Свойства совместного распределения?**



**5.3. Что понимают под событием?**



Вероятность любого события можно вычислить с помощью совместного распределения:

 Вероятность того, что «жарко И солнечно»?

 Вероятность того, что «жарко»?

 Вероятность того, что «жарко ИЛИ солнечно»?

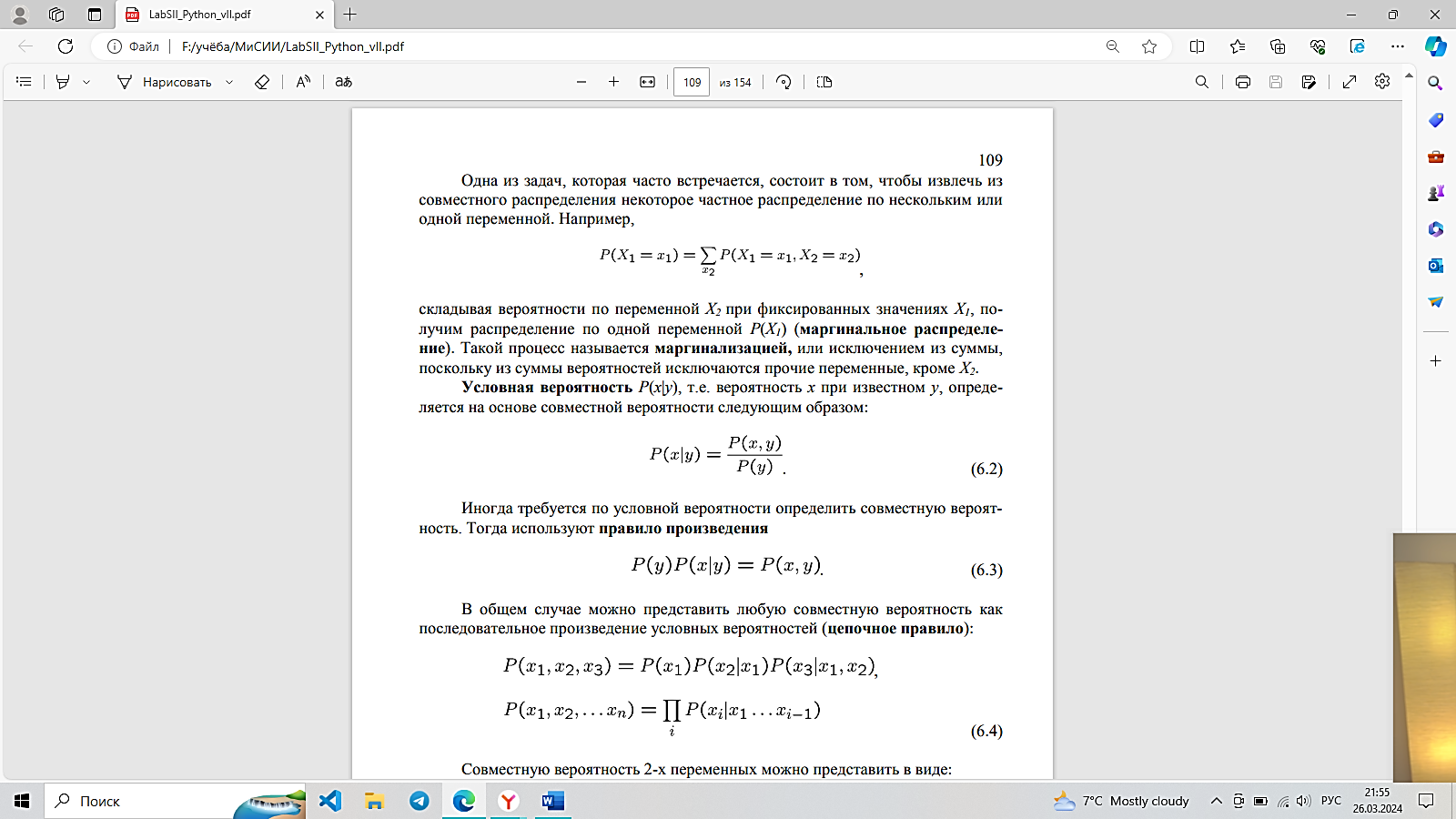
**5.4. Что такое маргинальное распределение? Как его получить из совместного распределения случайных переменных?**

Маргинальное распределение — это распределение вероятностей для одной или нескольких случайных переменных в совместном распределении случайных переменных. Оно предоставляет информацию о вероятностях отдельных значений каждой переменной, независимо от остальных переменных.

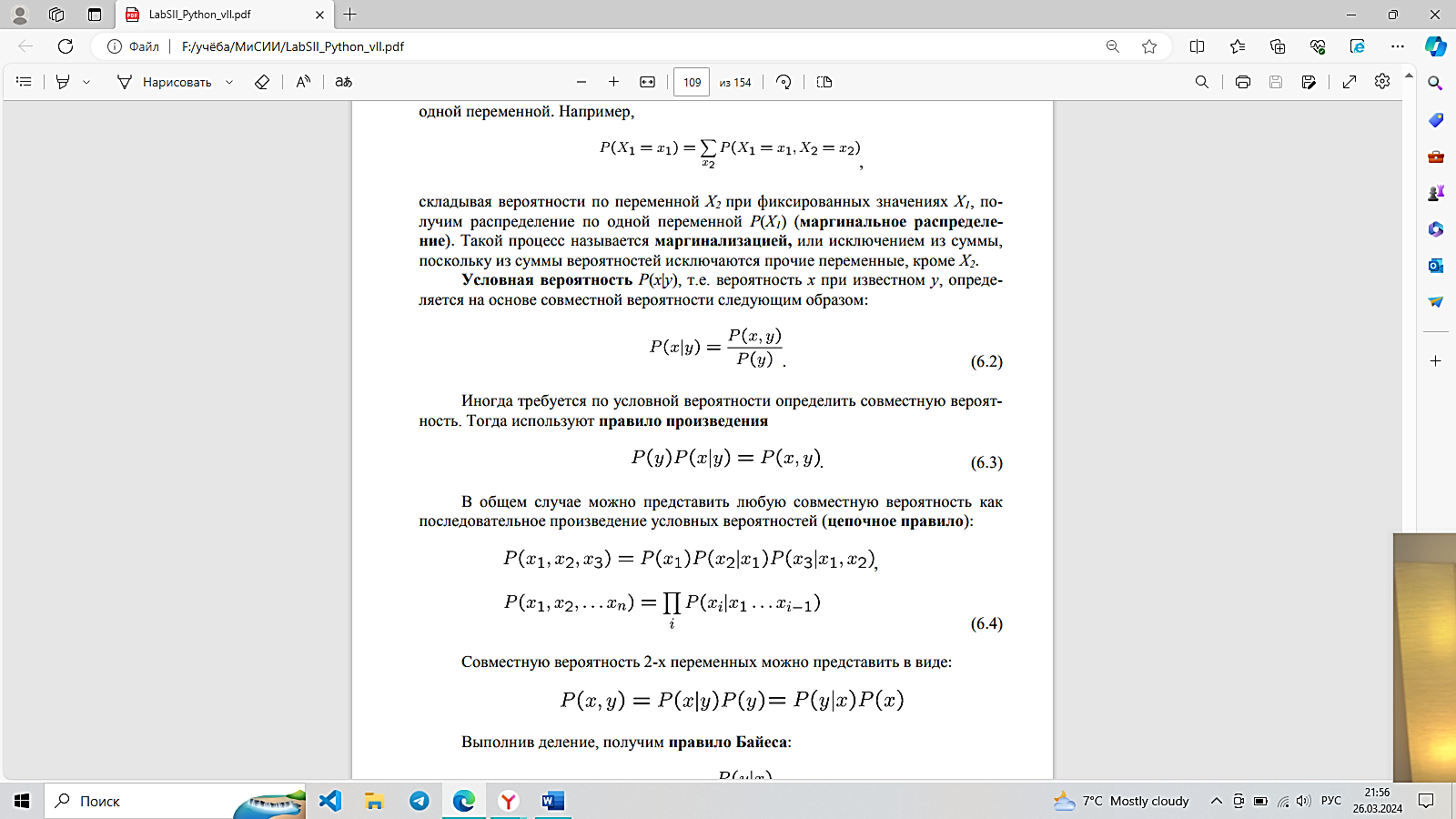
Маргинальные распределения соответствуют субтаблицам, которые получают из совместного распределения путем исключения переменных в ходе суммирования;

Маргинализация – «схлопывание» строк путем суммирования.

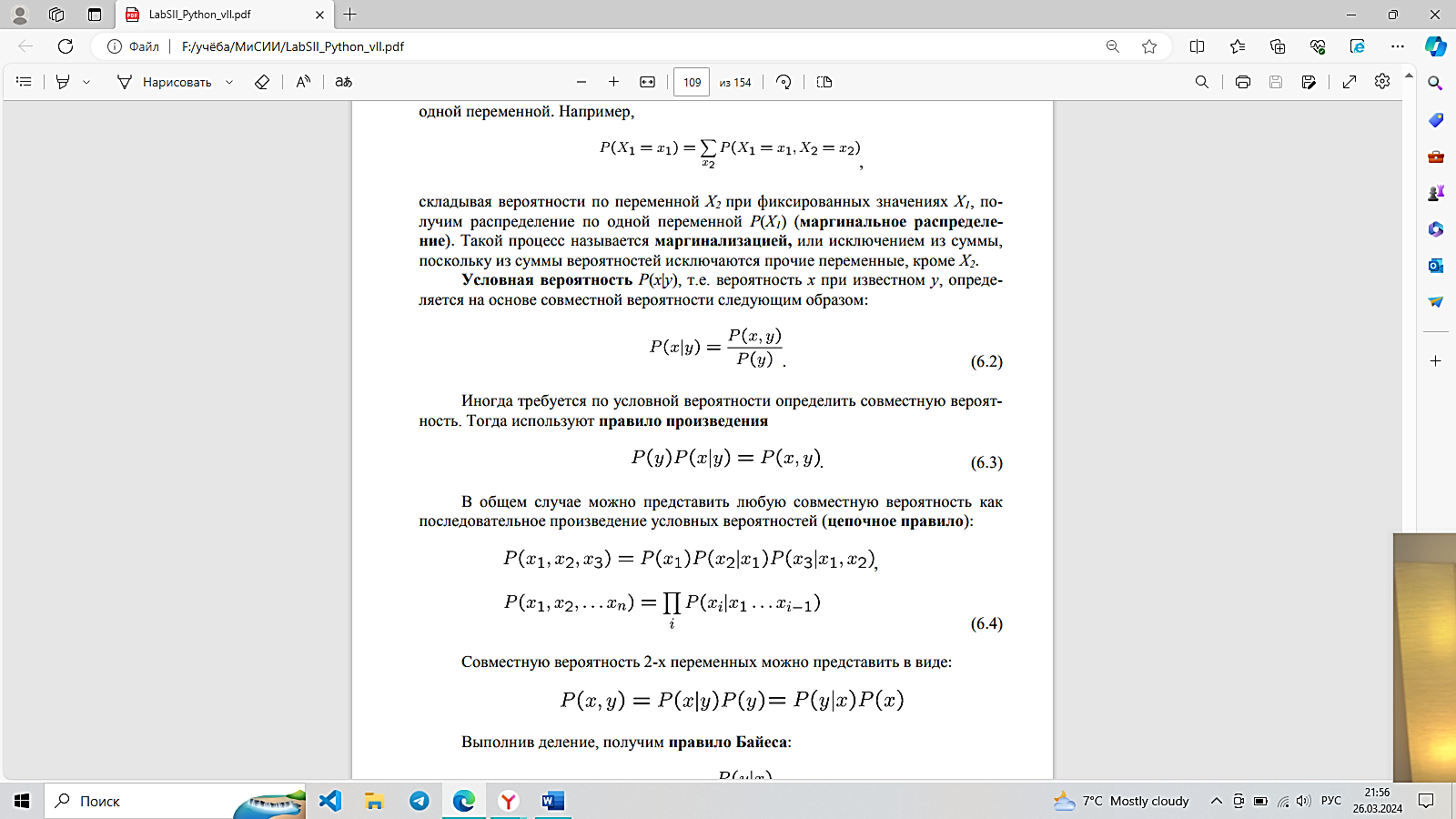
**5.5. Запишите формулу условной вероятности x при известном y.**



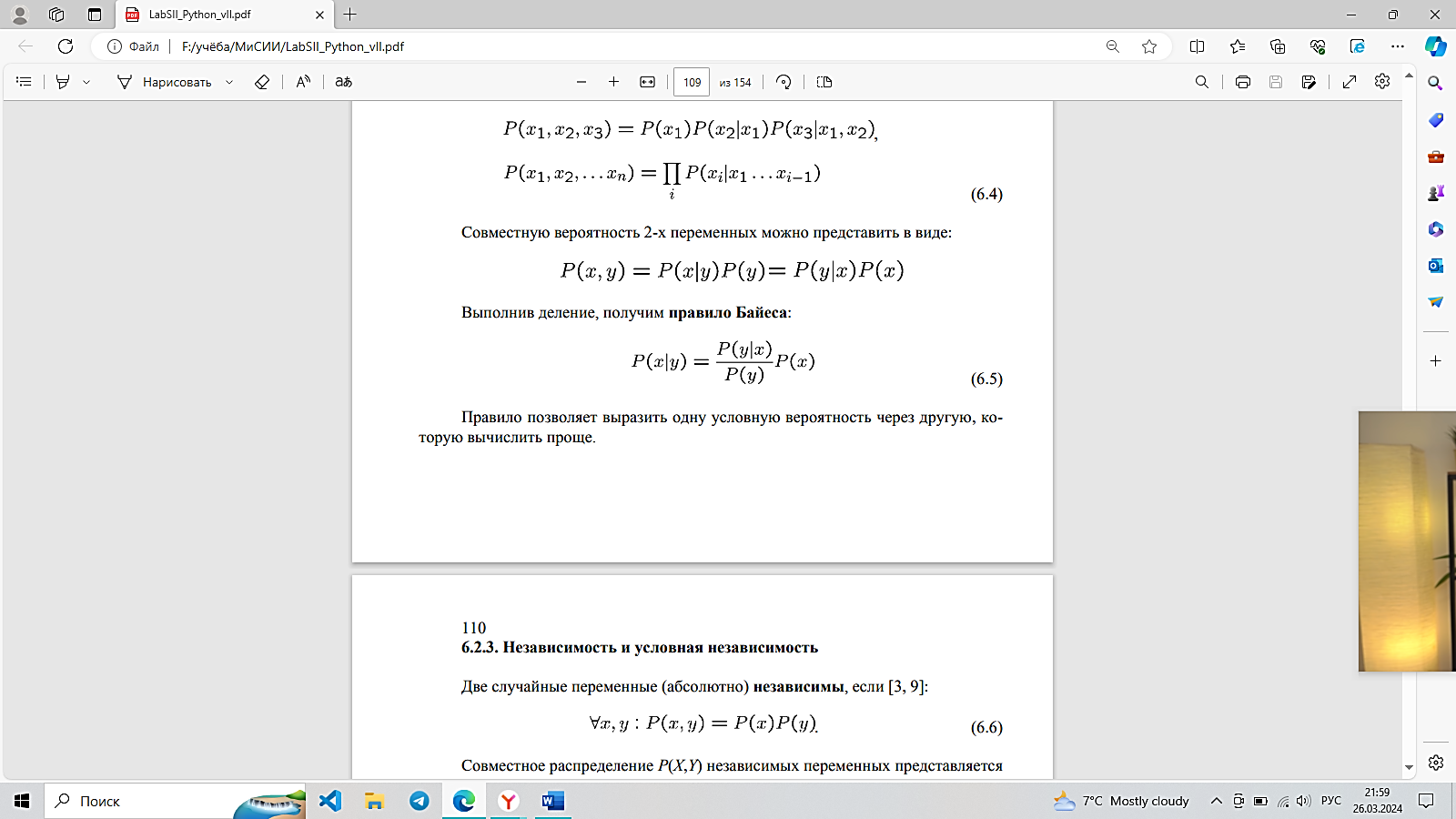
**5.6. Запишите правило произведения для 2-х переменных**.



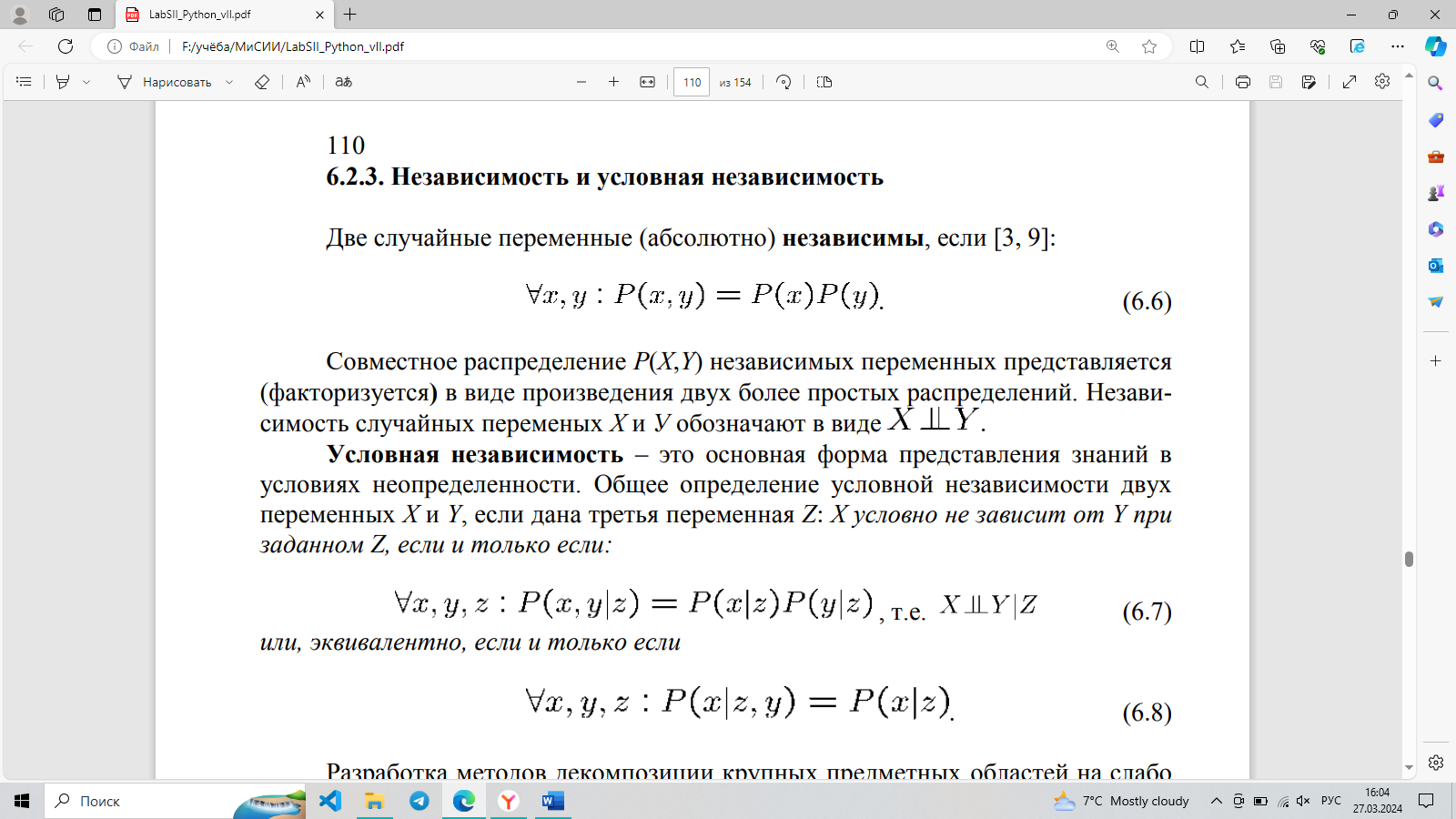
**5.7. Запишите цепочное правило.**



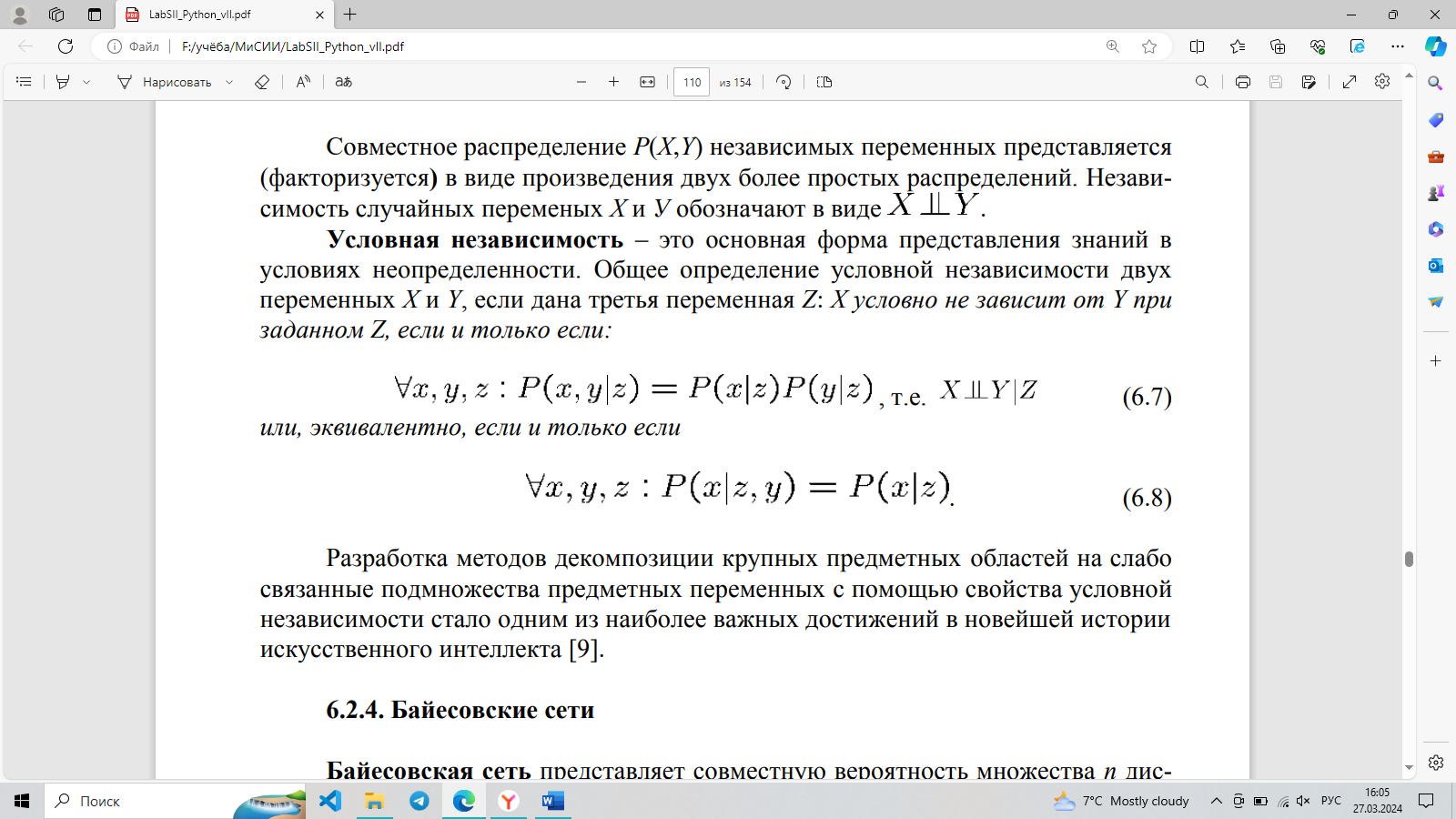
**5.8. Запишите правило Байеса и объясните его.**



**5.9. Определите понятие независимости 2-х случайных переменных.**



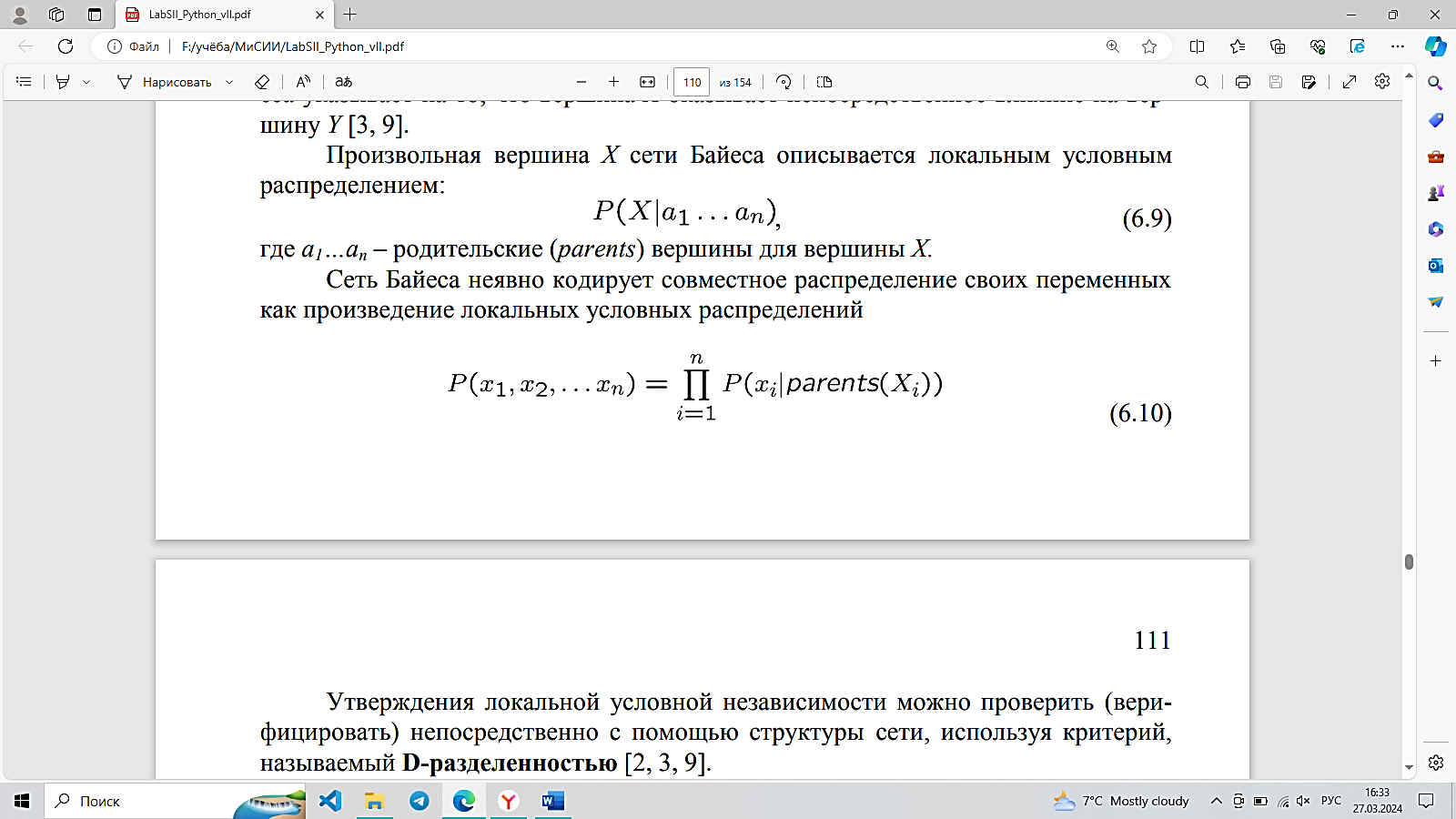
**5.10. Определите понятие условной независимости 2-х случайных переменных при заданной третьей переменной.**



**5.11. Определите понятие сети Байеса.**

Байесовская сеть представляет ***совместную вероятность множества n дискретных случайных переменных Х1, Х2,…, Хn в форме направленного ациклического графа.*** Каждая вершина графа представляется случайной переменной, с которой связана таблица условных вероятностей, содержащая вероятность каждого состояния переменной с учетом её родителей в графе. Ребра графа обозначают взаимодействия переменных. Интуитивный смысл стрелки вдоль ребра сети Байеса указывает на то, что вершина X оказывает непосредственное влияние на вершину Y.

**5.12. Запишите выражение полного совместного распределения для сети Байеса и объясните его.**



**5.13. Сформулируйте критерии D-разделенности для различных триплетов подсетей Байеса.**

Утверждения локальной условной независимости можно проверить (верифицировать) непосредственно с помощью структуры сети, используя критерий, называемый D-разделенностью [2, 3, 9].

D-разделенность - это концепция, связанная с графическими моделями, такими как сети Байеса. Она указывает на независимость между наборами переменных в графе, при условии некоторого набора других переменных.

Для различных триплетов подсетей Байеса (три переменные, связанные между собой), критерии D-разделенности могут быть сформулированы следующим образом:

1. X независимо от Z при условии Y:

Это означает, что переменная X статистически независима от переменной Z, при условии, что переменная Y известна. Формально, это можно записать как P(X, Z | Y) = P(X | Y) \* P(Z | Y), где \* обозначает произведение и P(A | B) обозначает условную вероятность A при условии B.

2. X независимо от Y при условии Z:

Это означает, что переменная X статистически независима от переменной Y, при условии, что переменная Z известна. Формально, это можно записать как P(X, Y | Z) = P(X | Z) \* P(Y | Z).

3. Y независимо от Z при условии X:

Это означает, что переменная Y статистически независима от переменной Z, при условии, что переменная X известна. Формально, это можно записать как P(Y, Z | X) = P(Y | X) \* P(Z | X).

**5.14. Что понимают под точным и приближенным вероятностным выводом?**

Представление предметной области в виде модели байесовской сети позволяет осуществлять вероятностные логические выводы. Вероятностный вывод в сетях Байеса предполагает вычисление вероятностей переменных запроса через вероятности других известных переменных. Известны алгоритмы ***точного*** вероятностного вывода **(например, вывод с использованием метода прямого (полного) перебора или метода исключения переменных),** которые характеризуются большой вычислительной сложностью. Точный вероятностный вывод в больших многосвязных сетях, часто является неосуществимым. Поэтому были разработаны методы ***приближенного*** вероятностного вывода, основанные на формировании случайных выборок из распределений. Вероятностный вывод на основе выборок происходит быстрее, чем вычисление ответа на запрос, например, методом исключения переменных. Точность вывода зависит от количества формируемых выборок.

**5.15. Сформулируйте алгоритм формирования случайной выборки из заданного распределения.**

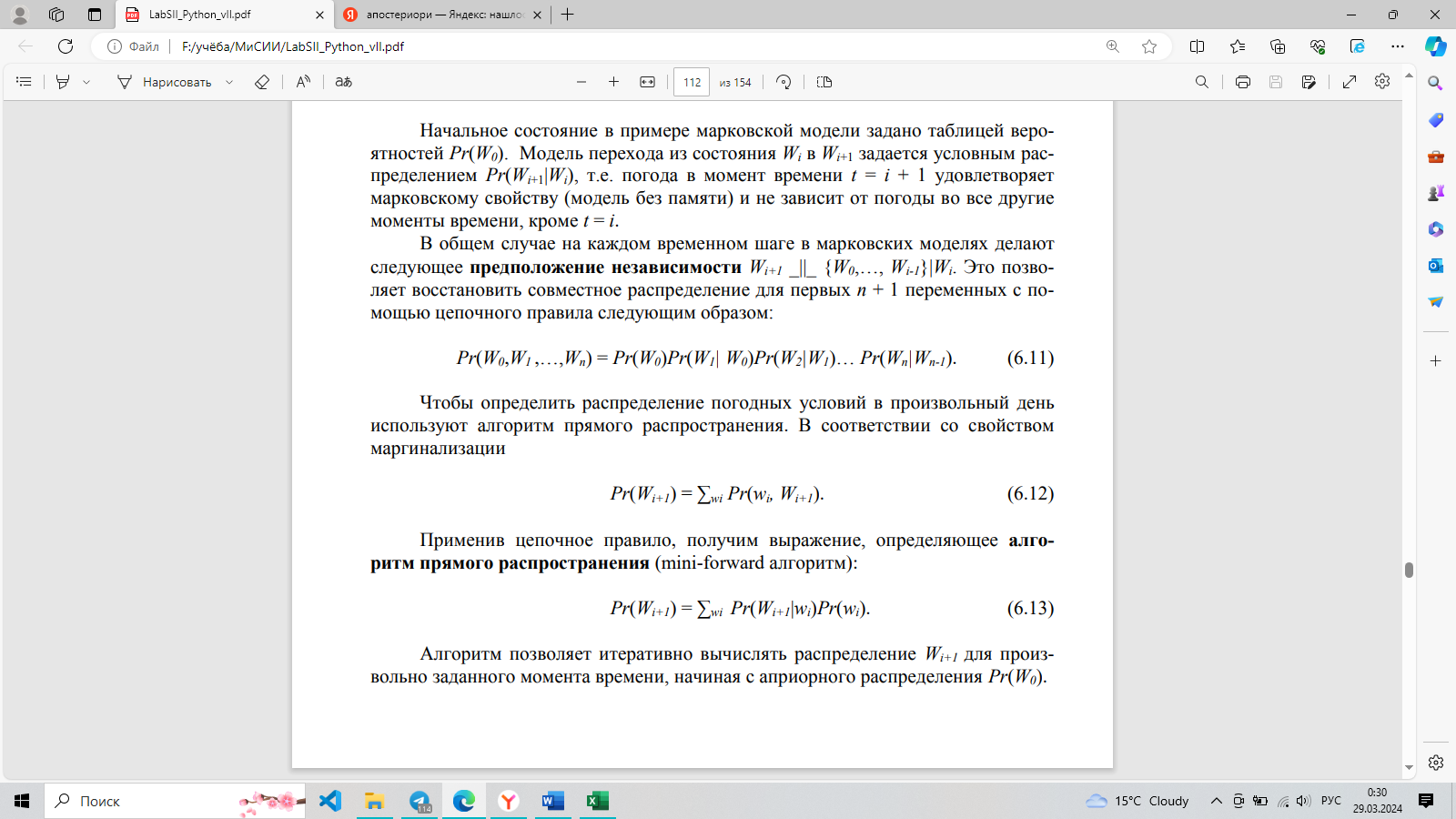
Алгоритм формирования выборки из заданного дискретного распределения можно представить в виде 2-х шагов:

1. Получить случайное число u из равномерного распределения в интервале [0, 1). Например, можно использовать функцию random() языка Пайтон;

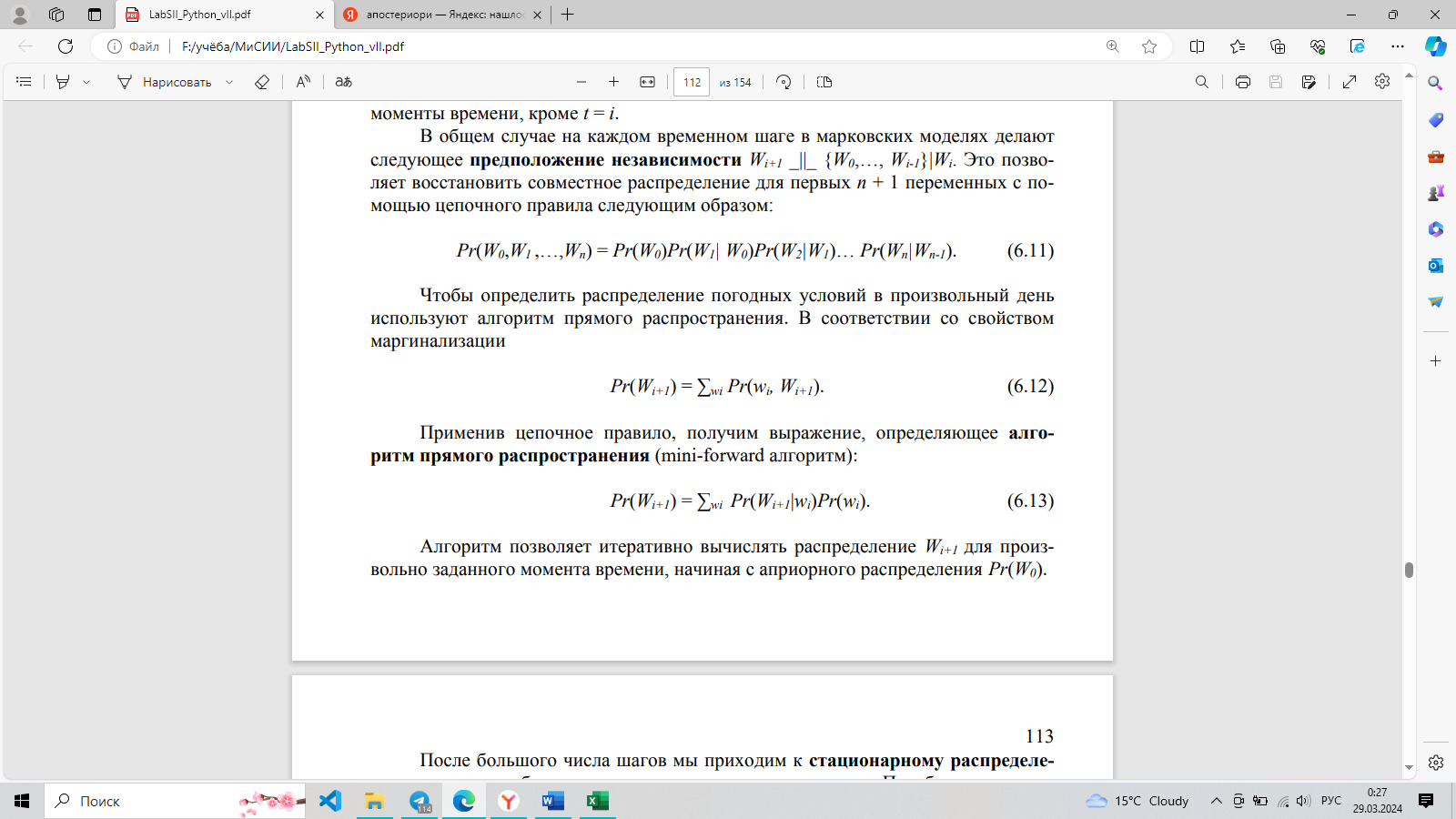
2. Преобразовать это значение u в выборочное значение дискретной случайной переменной с учетом заданного распределения, связав u с некоторым диапазоном, ширина которого равна задаваемой распределением вероятности.

**5.16. Приведите пример марковской модели. Какие предположения независимости используют в марковской модели?**

Марковскую модель можно рассматривать как аналог байесовской сети в виде внутренне связанной структуры бесконечной длины, зависящей от времени.



**5.17. Определите алгоритм прямого распространения для марковской модели.**



**5.18. Определите понятие скрытой марковской модели.**

Основная идея СММ состоит в том, что, хотя мы не можем непосредственно наблюдать скрытые состояния, мы наблюдаем результаты, порождаемые этими состояниями в виде наблюдаемых переменных. Задача состоит в том, чтобы оценить параметры СМЦ на основе наблюдаемых данных и использовать модель для прогнозирования скрытых состояний или генерации новых последовательностей.

Скрытые марковские модели (СММ) описываются с помощью двух вероятностных процессов: скрытого процесса смены состояний цепи Маркова и наблюдаемых значений свидетельств, формируемых при смене состояний.

В качестве примера на рисунке 6.2 изображена скрытая модель Маркова для моделирования погоды. В отличие от обычной марковской модели, СММ содержит два типа узлов: скрытые узлы Wi , которые являются переменными состояния и представляют погоду в i-ый день, и наблюдаемые узлы Fi , которые представляют переменные, называемые свидетельствами (наблюдениями). В рассматриваемом примере свидетельства Fi представляют прогноз погоды в i-ый день.

**5.19. Сформулируйте задачу фильтрации для СММ.**

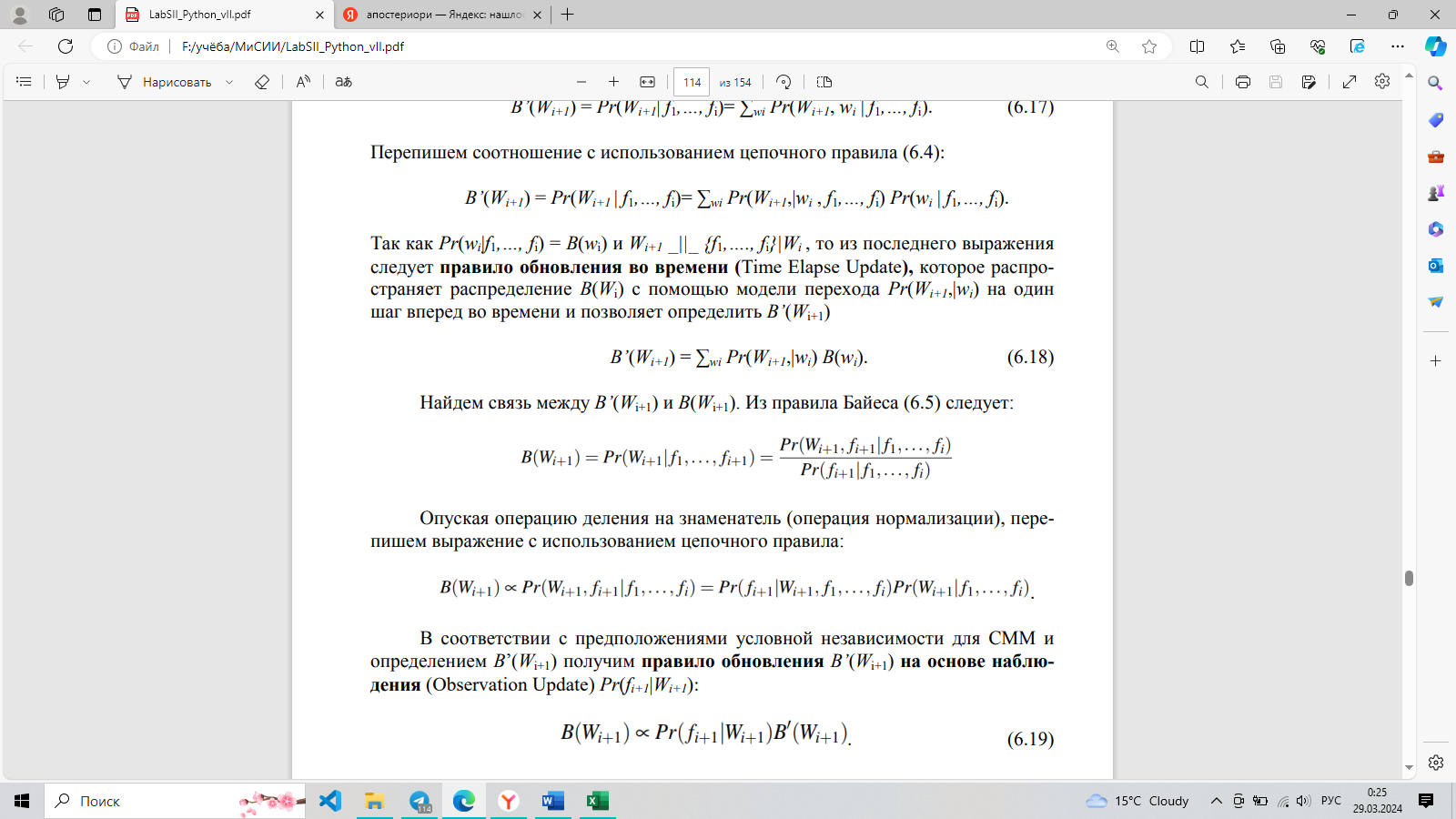
Одна из задач, решаемая с помощью модели СММ и называемая фильтрацией или мониторингом, заключается в вычислении апостериорных распределений скрытых переменных состояний в текущий момент времени по значениям всех полученных к этому моменту свидетельств.

**5.20. Какие предположения независимости используют в СММ?**

**Предположение о скрытой Марковской цепи**: СММ предполагает, что скрытая Марковская цепь, которая генерирует наблюдаемые данные, удовлетворяет свойству Марковской зависимости. Это означает, что вероятность состояния в момент времени t+1 зависит только от состояния в момент времени t и не зависит от предыдущих состояний. Это называется свойством первого порядка Марковской цепи.

**Предположение о независимости наблюдаемых переменных**: СММ предполагает, что каждая наблюдаемая переменная зависит только от соответствующего скрытого состояния и не зависит от других наблюдаемых переменных. Это означает, что наблюдаемые переменные в каждый момент времени являются независимыми.

**5.21. Сформулируйте правило обновления во времени и правило на основе наблюдений для СММ.**



**5.22. Сформулируйте алгоритм прямого распространения для СММ.**

Объединение полученных правил дает итерационный алгоритм, известный как алгоритм прямого распространения для СММ (аналог mini-forward алгоритма для обычной марковской модели). Алгоритм включает два отдельных шага:

1. Обновление B’(W i+1) по B(Wi) на одном шаге во времени;

2. Обновление B(Wi) на основе наблюдения, т.е. определение B(Wi+1) по B’(W i+1).

**5.23. Что такое фильтрация частиц применительно к СММ?**

Точный вывод с использованием алгоритма прямого распространения СММ характеризуется большой вычислительной сложностью. В этом случае, аналогично сетям Байеса, используют приближенные методы вывода, основанные на формировании случайных выборок из распределений [3].

Применение к СММ процедур, аналогичных байесовскому сэмплированию (взятию выборок), называется фильтрацией частиц и включает в себя моделирование движения набора частиц через граф состояний для аппроксимации распределения вероятности (доверий) рассматриваемой случайной величины в требуемый момент времени. Частица в этом случает представляет возможное выборочное значение случайной величины. При этом вместо хранения полных таблиц вероятностей, отображающих каждое состояние в вероятность, хранят список из n частиц, в котором каждая частица может находиться в одном из d состояний.

**5.24. Сформулируйте правило обновления во времени и правило на основе наблюдений, применяемые при фильтрации частиц.**

Обновление во времени (Time Elapse Update) — обновление выборочного значения каждой частицы в соответствии с моделью переходной вероятности. Для частицы в состоянии ti выполняется случайная выборка обновленного значения из переходного распределения Pr(Ti+1 | ti).

Обновление на основе наблюдения (Observation Update). Этот этап немного сложнее. Здесь используется модель сенсора Pr(Fi |Ti) для взвешивания каждой частицы в соответствии с вероятностью, определяемой наблюдаемым свидетельством и состоянием частицы. В частности, частице в состоянии ti при свидетельстве fi , поступающим от некоторого сенсора, присваивается вес Pr(fi |ti).

Алгоритм обновления на основе наблюдений следующий:

1. Рассчитайте веса всех частиц в соответствии с Pr(fi |ti);

2. Вычислите суммарный вес каждого состояния;

3. Если сумма всех весов во всех состояниях равна 0, повторно инициализируйте все частицы;

4. Иначе нормализуйте распределение по отношению к суммарному весу и выполните выборки из этого распределения.