Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**Отчёт по практическим работам**

Дисциплина: Теория вероятностей и математическая статистика

Выполнил студент гр. 3530901/10001 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.Л. Симоновский

(подпись)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ К.В. Никитин

(подпись)

“02” май 2023 г.

Санкт-Петербург

2023

Оглавление

[1. Задания 2](#_Toc135495324)

[2. Решение 2](#_Toc135495325)

[a. Задача 1.7 2](#_Toc135495326)

[b. Задача 2.6 3](#_Toc135495327)

[c. Задача 3.22 4](#_Toc135495328)

[d. Задача 4.23 5](#_Toc135495329)

[e. Задача 5.3 5](#_Toc135495330)

[3. Моделирование 5](#_Toc135495331)

[a. Задача 2.6 5](#_Toc135495332)

[b. Задача 3.22 7](#_Toc135495333)

[c. Задача 4.23 8](#_Toc135495334)

[d. Задача 5.3 10](#_Toc135495335)

[4. Ссылки 11](#_Toc135495336)

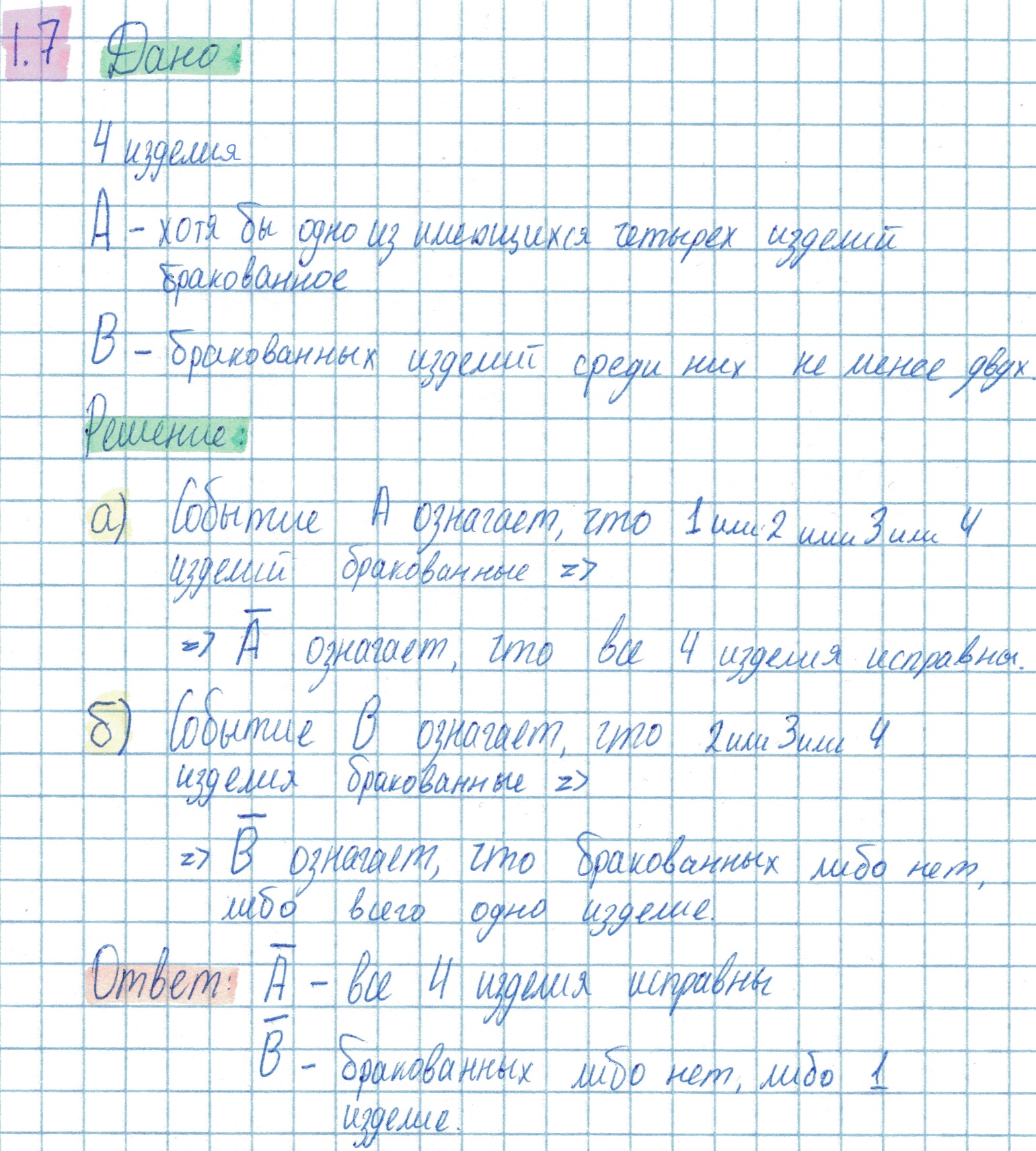
# Задания

Задания для теоретического решения: 1.7, 2.6, 3.22, 4.23, 5.3, 6.14, 7.16, 8.40, 9.20, 10.7, 11.16, 12.17, 13.1, 14.4, 15.6, 16.6, 17.8, 18.10, 19.4, 20.34, 21.10, 22.17, 23.12, 35.19, 36.25, 37.5, 38.17, 39.30.

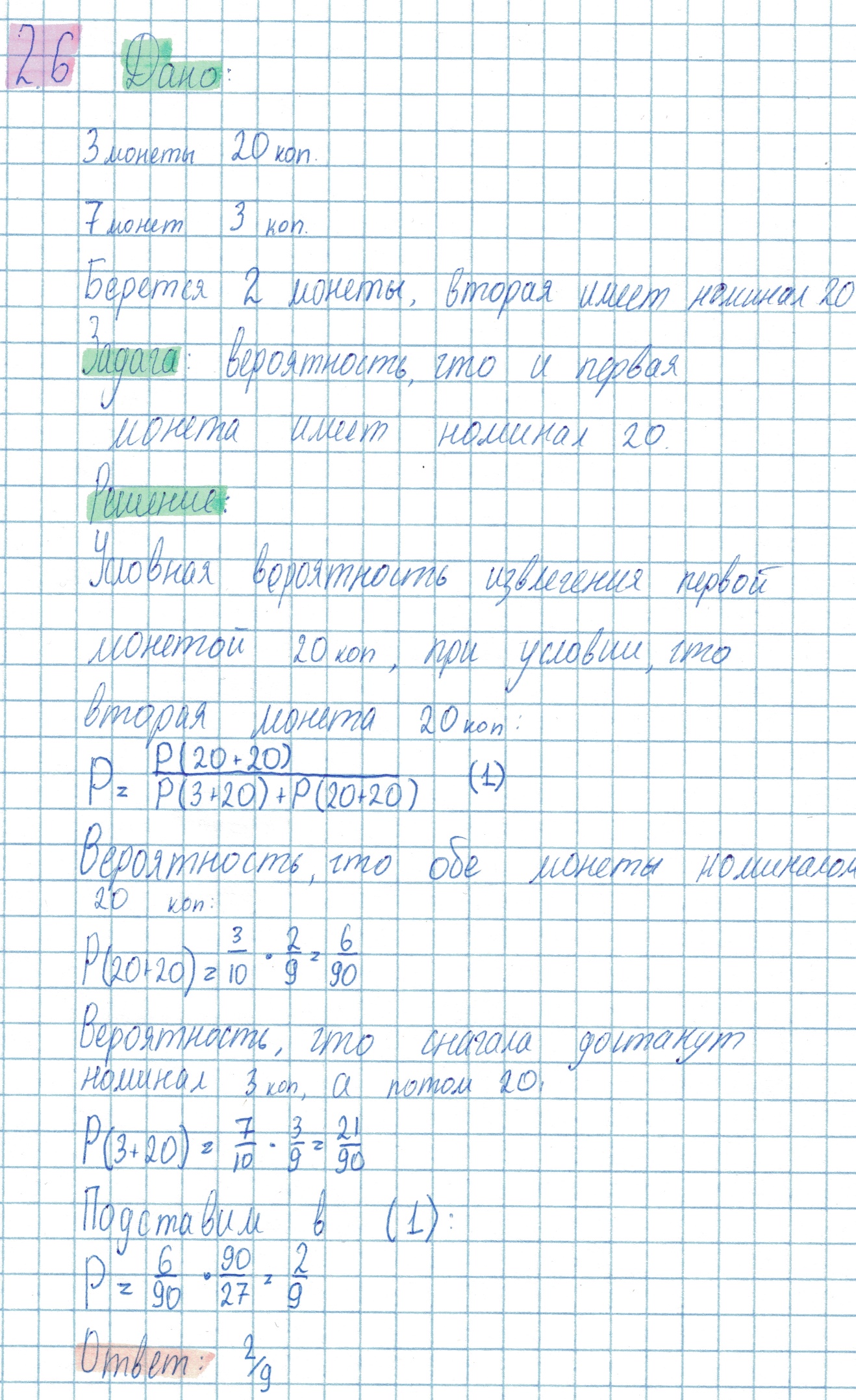
Задания для моделирования: 2.6, 3.22, 4.23, 5.3, 6.14, 7.16, 8.40, 9.20, 10.7, 11.16, 12.17, 13.1, 14.4, 15.6, 17.8, 19.4, 21.10, 22.17.

# Решение

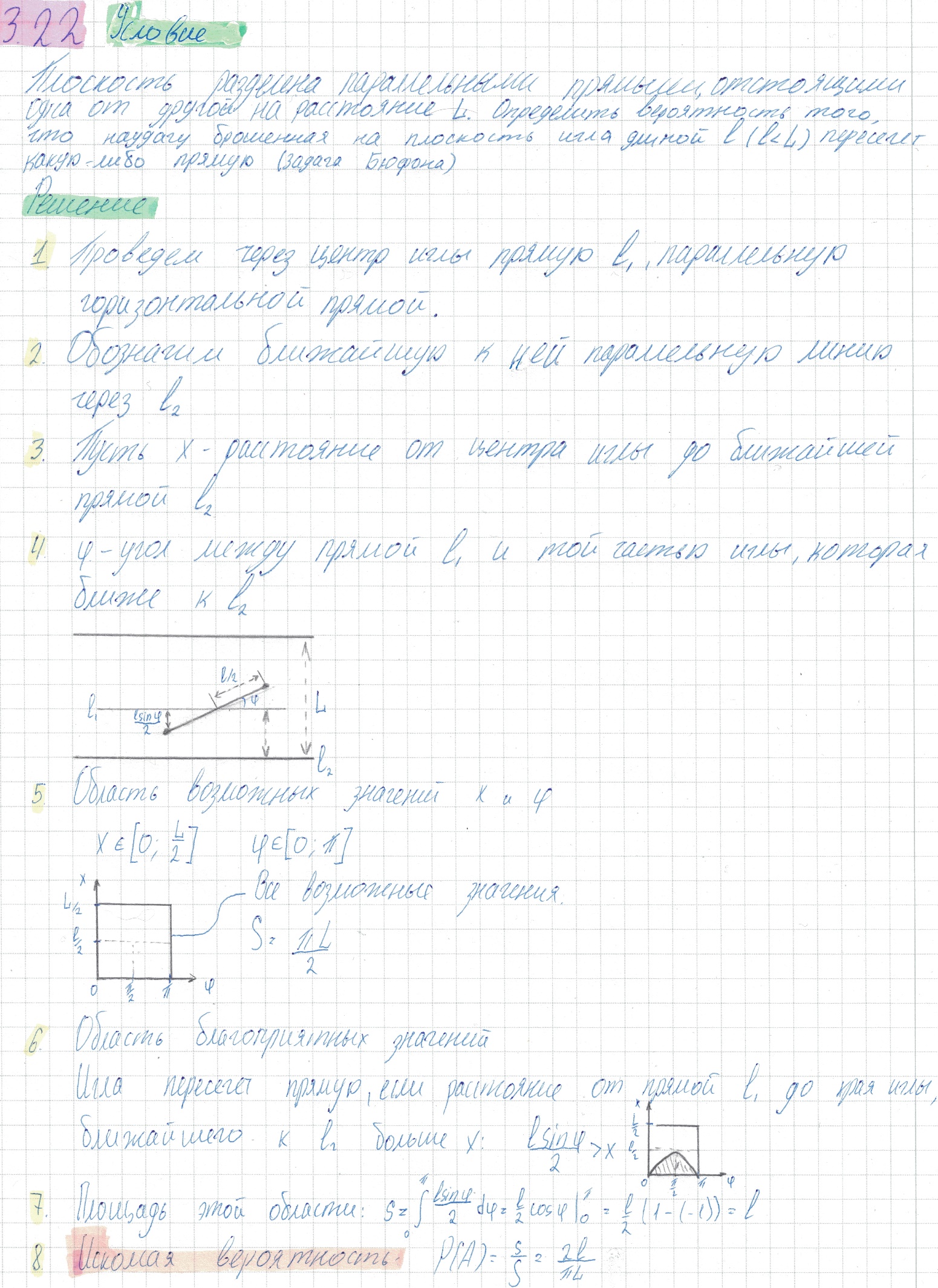
## Задача 1.7



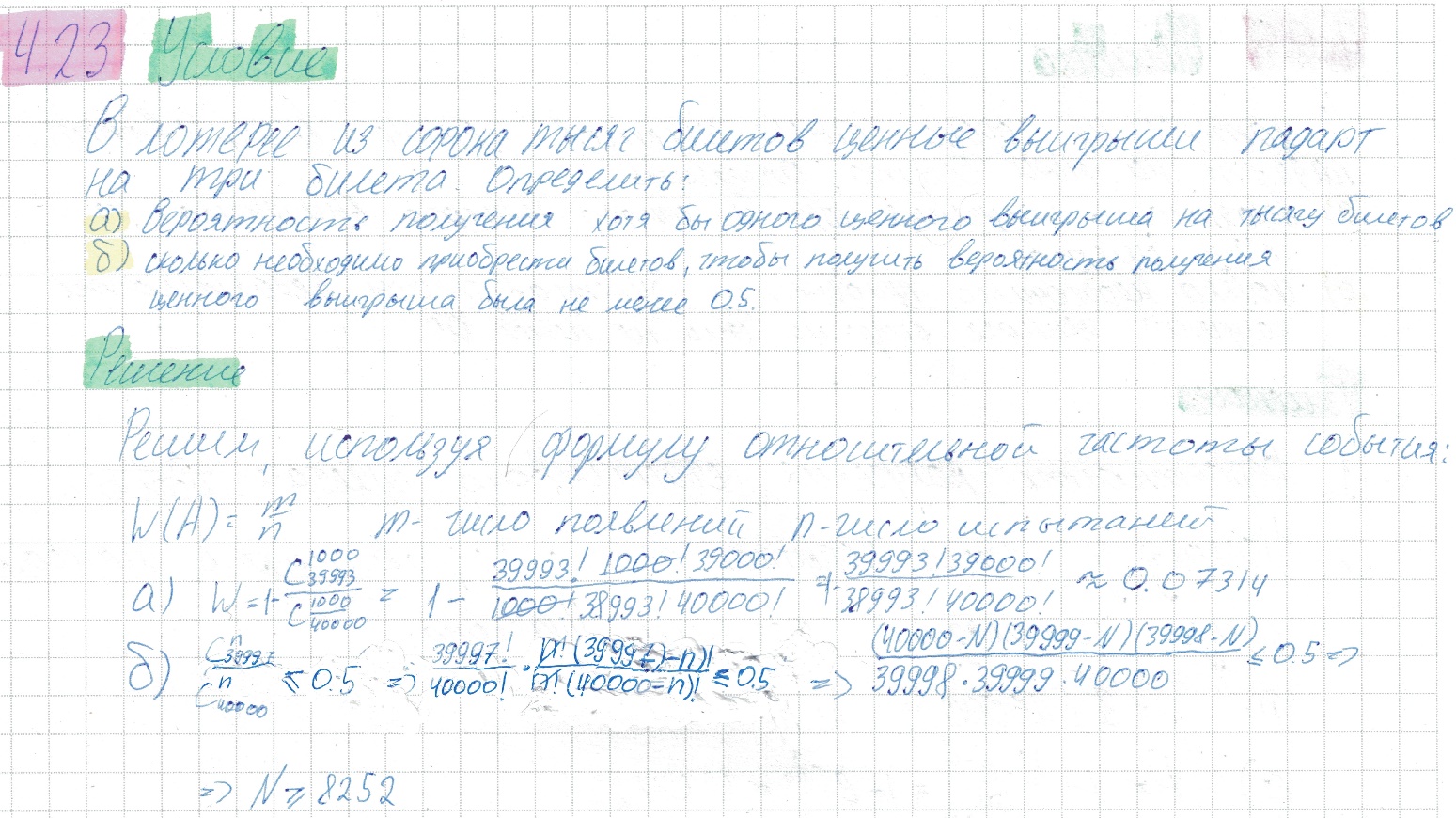
## Задача 2.6



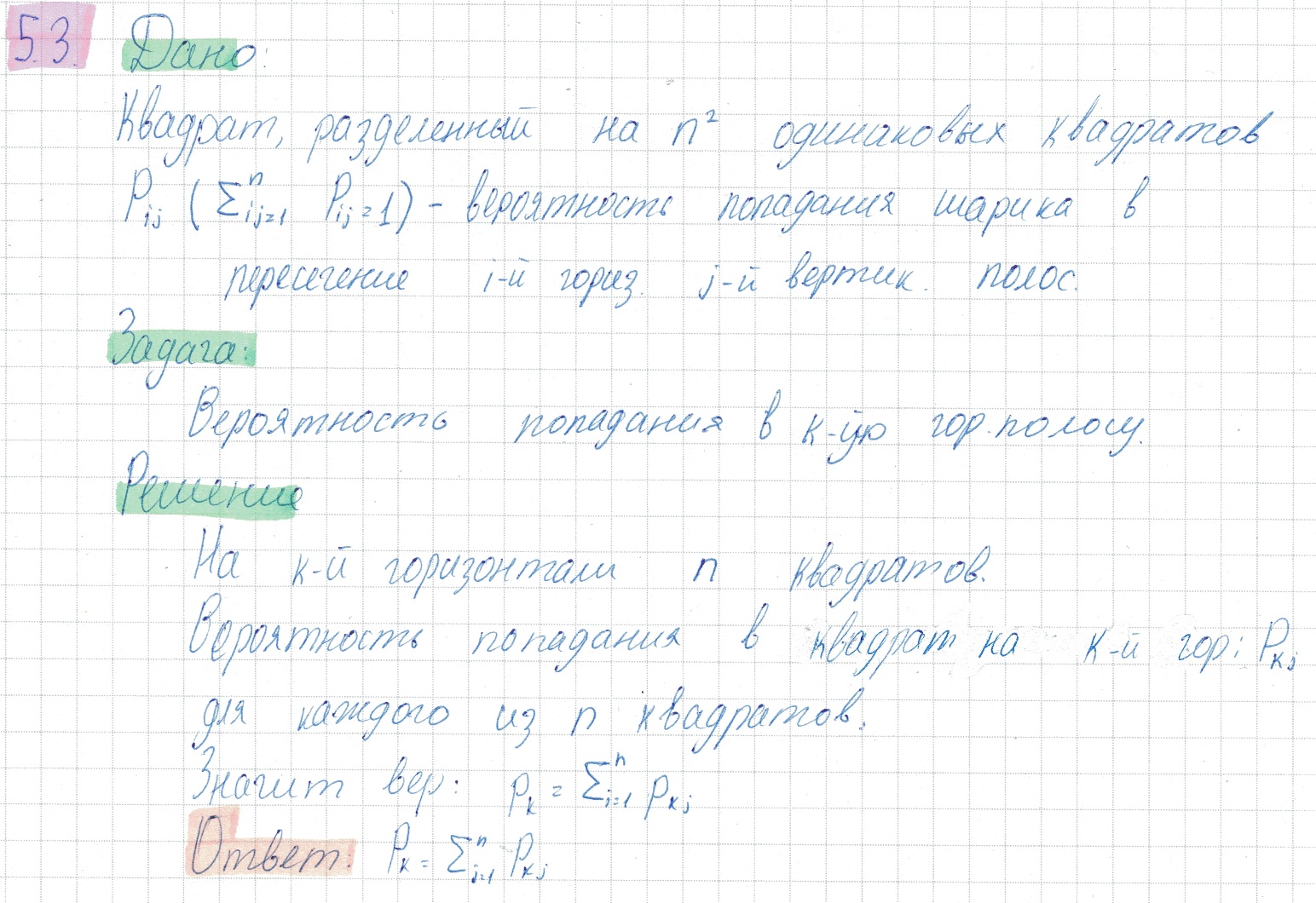
## Задача 3.22



## Задача 4.23



## Задача 5.3



# Моделирование

## Задача 2.6

**Условие:**

В кошельке лежат три монеты достоинством по 20 коп. и семь монет по 3 коп. Наудачу берется одна монета, а затем извлекается вторая монета, оказавшаяся монетой в 20 коп. Определить вероятность того, что и первая извлеченная монета имеет достоинство в 20 коп.

**Решение:**

Создадим функцию для получения количества 20-ок и 3-ек:

**def** get\_input\_data**():**

count\_20 **=** 3

count\_3 **=** 7

**return** count\_20**,** count\_3

Далее необходимо сделать функцию, которая будет возвращать результат броска, принимая на вход общее количество монет, количество 20-ок и 3-ек:

**def** get\_random\_coin**(**count\_20**,** count\_3**):**

"""

Получение результата доставания монетки

"""

x **=** randint**(**1**,** count\_3 **+** count\_20**)**

**if** x **<=** count\_20**:**

**return** 20

**return** 3

Также реализуем функцию одной итерации вытягивания двух монет, если вторая монета не 20-ка, вернем None, иначе результат первого броска:

**def** one\_iteration**(**count\_20**,** count\_3**):**

"""

Одна итерация вытягивания двух монет

"""

first\_coin **=** get\_random\_coin**(**count\_20**,** count\_3**)**

**if** first\_coin **==** 20**:**

count\_20 **-=** 1

**else:**

count\_3 **-=** 1

second\_coin **=** get\_random\_coin**(**count\_20**,** count\_3**)**

**if** second\_coin **!=** 20**:**

**return**

**return** 1 **if** first\_coin **==** 20 **else** 0

Ну и последнее – основной цикл программы на 1 000 000 итераций одиночной программы:

def main():

"""

Основной цикл, запускающий одну итерацию несколько раз

"""

count\_20, count\_3 = get\_input\_data()

iteration\_counter = 0

event\_counter = 0

while iteration\_counter < 1\_000\_000:

iteration = one\_iteration(count\_20, count\_3)

if iteration is None:

continue

event\_counter += iteration

iteration\_counter += 1

print(f'Количество 20-ок: {count\_20}, количество 3-ек: {count\_3}\n'

f'Количество попыток, когда второй раз выпала 20: {iteration\_counter}\n'

f'Количество выпадений двух 20 подряд: {event\_counter}\n'

f'Итоговая вероятность: {event\_counter / iteration\_counter}\n'

f'Ожидаемая вероятность: {(count\_20 - 1) / (count\_20 + count\_3 - 1)}')

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

main()

Выполним запуск программы и посмотрим на результат:

Количество 20-ок: 3, количество 3-ек: 7

Количество попыток, когда второй раз выпала 20: 1000000

Количество выпадений двух 20 подряд: 223004

Итоговая вероятность: 0.223004

Ожидаемая вероятность: 0.2222222222222222

Таким образом результат моделирования близок к результатам моделирования.

## Задача 3.22

**Условие:**

Плоскость разграфлена параллельными прямыми, отстоящими одна от другой на расстояние L. Определить вероятность того, что наудачу брошенная на плоскость игла длинной l (l<L) пересечет какую-либо прямую (задача Бюффона).

**Решение:**

Создадим функцию для получения начальных данных (расстояние между отстоящими прямыми и длина иглы):

**def** get\_input\_data**():**

"""

Начальные данные

"""

l **=** 3

L **=** 7

**return** l**,** L

Далее необходимо сделать функцию, которая будет возвращать результат броска иглы, как расстояние до ближайшей прямой и угол между прямой и «горизонтом».

**def** get\_random\_x\_fi**(**L**):**

"""

Получение результата броска иголки

"""

x **=** random**()** **\*** L **/** 2

fi **=** random**()** **\*** math**.**pi

**return** x**,** fi

Также реализуем функцию одной итерации броска иголки, которая будет возвращать результат броска и подставлять в условие попадания, полученное в ходе аналитического решения ():

**def** one\_iteration**(**L**,** l**):**

"""

Одна итерация

"""

x**,** fi **=** get\_random\_x\_fi**(**L**)**

**return** l **\*** math**.**sin**(**fi**)** **/** 2 **>** x

Ну и последнее – основной цикл программы на 1 000 000 итераций одиночной программы:

def main():

"""

Основной цикл, запускающий одну итерацию несколько раз

"""

l, L = get\_input\_data()

event\_counter = 0

count\_iterations = 1\_000\_000

for i in range(0, count\_iterations):

iteration = one\_iteration(L, l)

event\_counter += iteration

print(f'Расстояние между прямыми: {L}, длина прямой: {l}\n'

f'Количество падений иглы на прямую: {event\_counter}\n'

f'Количество падений иглы мимо прямой: {count\_iterations - event\_counter}\n'

f'Смоделированная вероятность падения иглы на прямую: {event\_counter / count\_iterations}\n'

f'Расчетная вероятность падения иглы на прямую: {2 \* l / (math.pi \* L)}')

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

main()

Выполним запуск программы и посмотрим на результат:

Расстояние между прямыми: 7, длина прямой: 3

Количество падений иглы на прямую: 273378

Количество падений иглы мимо прямой: 726622

Смоделированная вероятность падения иглы на прямую: 0.273378

Расчетная вероятность падения иглы на прямую: 0.272837045300392

Таким образом результат моделирования близок к результатам моделирования.

## Задача 4.23

**Условие:**

В лотерее из сорока тысяч билетов ценные выигрыши падают на три билета, определить:

1. Вероятность получения хотя бы одного ценного выигрыша на тысячу билетов
2. Сколько необходимо приобрести билетов, чтобы вероятность получения ценного выигрыша была не менее 0.5

**Решение:**

Создадим функцию для получения начальных данных (общее число билетов и количество выигрышных):

**def** get\_input\_data**():**

"""

Начальные данные

"""

N **=** 40000

win **=** 3

**return** N**,** win

Далее необходимо сделать функцию, которая будет возвращать результат одной покупки в лотерее, причем нужно учесть, что несколько одинаковых билетов быть не может, для этого воспользуемся set()

**def** one\_iteration**(**x**,** win**,** n**,** N**):**

"""

Одна покупка n билетов

"""

x **=** **set()**

**while** **len(**x**)** **<** n**:**

m **=** randint**(**0**,** N**)**

**if** m **<** win**:**

**return** **True**

x**.**add**(**m**)**

**return** **False**

Эта функция достаточно долгая, поэтому необходимо воспользоваться многопоточностью для ускорения подсчетов. Вот как будет выглядеть функция для вызова one\_iteration много раз:

**def** do\_iterations**(**N**,** win**,** n**,** count\_iterations**):**

"""

Функция для выполнения нескольких покупок

"""

**with** Pool**(**processes**=**8**)** **as** pool**:**

one\_iteration\_partial **=** partial**(**one\_iteration**,** win**=**win**,** n**=**n**,** N**=**N**)**

results **=** pool**.map(**one\_iteration\_partial**,** **range(**count\_iterations**))**

**return** results

Вместо 8 необходимо указать количество ядер процессора, которые вы собираетесь задействовать для расчетов.

В функции main() получим данные, используя get\_input\_data()

N**,** win **=** get\_input\_data**()**

После этого решим пункт a:

# Решение пункта a

count\_iterations **=** 10\_000

n **=** 1000

event\_counter **=** **sum(**do\_iterations**(**N**,** win**,** n**,** count\_iterations**))**

**print(**f'Пункт a:'

f'Количество билетов: {N}, количество победных: {win}\n'

f'Количество покупок с выигрышным билетом: {event\_counter}\n'

f'Количество покупок без выигрышного билета: {count\_iterations **-** event\_counter}\n'

f'Смоделированная вероятность получения билета: {event\_counter **/** count\_iterations}\n'

f'Расчетная вероятность получения билета: {1 **-** math**.**comb**(**N **-** win**,** n**)** **/** math**.**comb**(**N**,** n**)**}'**)**

Для решения пункта b уменьшим точность подсчетов до 1000. Считать будем вероятность от 1000 и до 20000 с шагом 500, чтоб получить график изменения погрешности:

count\_iterations **=** 1\_000

chance **=** **[]**

real\_chance **=** **[]**

points **=** **list(range(**1000**,** 20000**,** 500**))**

**for** n **in** points**:**

event\_counter **=** **sum(**do\_iterations**(**N**,** win**,** n**,** count\_iterations**))**

chance**.**append**(**event\_counter **/** count\_iterations**)**

real\_chance**.**append**(**1 **-** math**.**comb**(**N **-** win**,** n**)** **/** math**.**comb**(**N**,** n**))**

plt**.**plot**(**points**,** chance**,** label**=**'Model'**,** linestyle**=**'--'**,** color**=**'r'**,** marker**=**'o'**,** markersize**=**3**)**

plt**.**plot**(**points**,** real\_chance**,** label**=**'Real'**,** linestyle**=**'--'**,** color**=**'g'**,** marker**=**'o'**,** markersize**=**3**)**

plt**.**savefig**(**f"Chance.jpg"**)**

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

Выполним запуск программы и посмотрим на результат:

Пункт a:

Пункт a:Количество билетов: 40000, количество победных: 3

Количество покупок с выигрышным билетом: 737

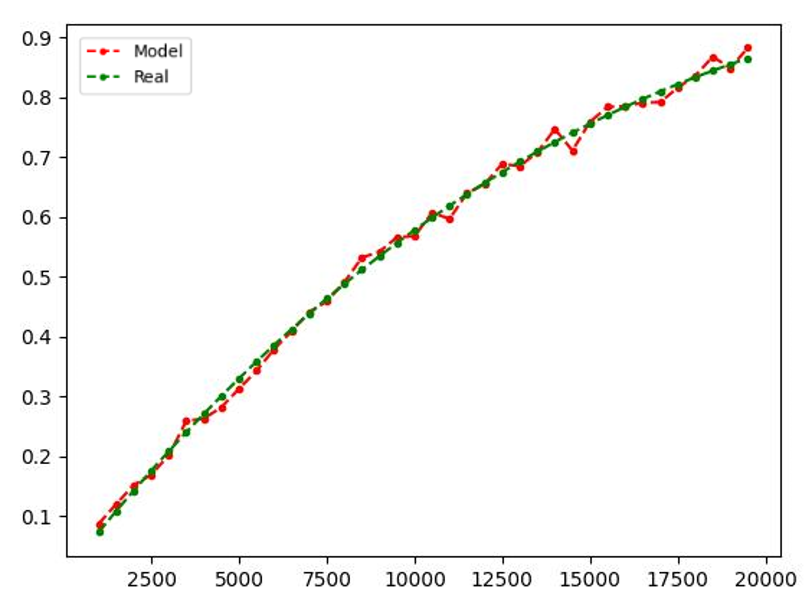
Количество покупок без выигрышного билета: 9263

Смоделированная вероятность получения билета: 0.0737

Расчетная вероятность получения билета: 0.07314240749538414

Видно, что результат моделирования близок к теоретическому.

Пункт b:



Как видно из графика искомое значение равно примерно 8110, что достаточно близко к ответу при теоретическом решении, точность можно повысить путем увеличения количества итераций.

## Задача 5.3

**Условие:**

Квадрат разделен на n2 одинаковых квадратов.

() – вероятность попадания шарика в пересечение i-й горизонтальной и j-й вертикальной полосы.

**Задача:**

Найти вероятность попадания в k-ю горизонтальную полосу.

**Решение:**

Создадим функцию для получения входных данных – в данной задаче это лишь размерность n:

def get\_input\_data():  
 # Размерность массива  
 n = 10  
 return n

Создадим массив вероятностей , сумма элементов этого массива n на n равна единице:

def generate\_array(n):  
 *"""  
 Создает массив случайных чисел, сумма которых равна 1, размерности n на n  
 """* random\_nums = np.random.rand(n, n)  
 total\_sum = np.sum(random\_nums)  
 result\_array = random\_nums / total\_sum  
 return result\_array

Получим входные данные и массив n на n, так же номер горизонтали k и количество итераций:

n = get\_input\_data()  
P = generate\_array(n)  
k = randint(0, n - 1)  
count\_in\_k = 0  
count\_iterations = 1000000

Создадим основной цикл программы:

for i in range(count\_iterations):  
 chance = random()  
 sum\_chance = 0  
 counter = 0  
 while chance > sum\_chance:  
 sum\_chance += P[counter // n][counter % n]  
 counter += 1  
 if (counter - 1) // n == k:  
 count\_in\_k += 1

Выведем итоговый результат:

print(f'Теоретическая вероятность: {np.sum(P[k, :])}\n'  
 f'Полученная вероятность: {count\_in\_k / count\_iterations}')

Полученный вывод:

Теоретическая вероятность: 0.098022

Полученная вероятность: 0.09798

Как видно из вывода программа работает корректно.

# Ссылки

Ссылка на репозиторий с моделированием: [github.com](https://github.com/DafterT/Probability_Theory)