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

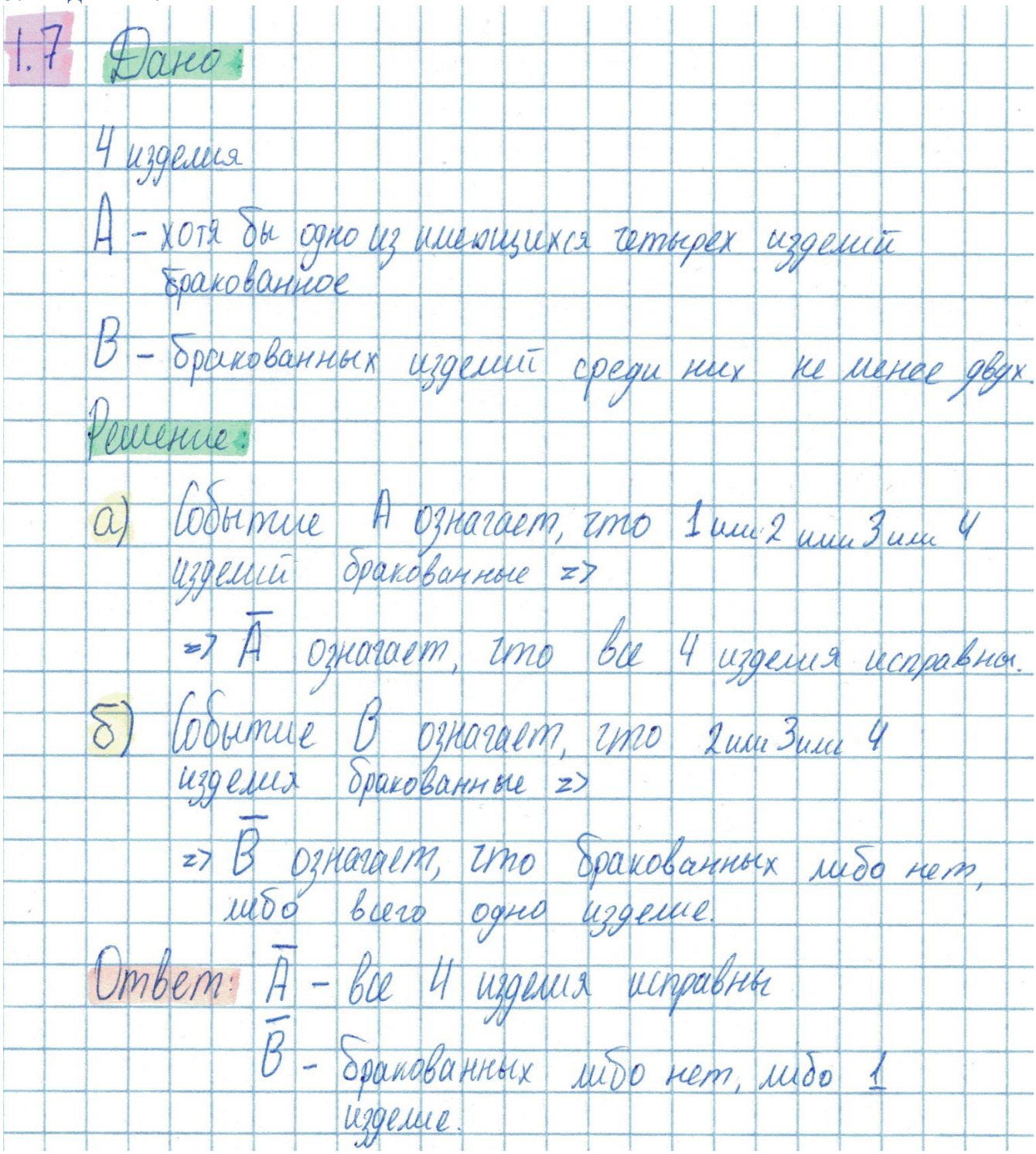
1. Задания

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.

2. Решение

a. Задача 1.7



b. Задача 2.6

2.6 Дано:

3 монеты 20 коп.

7 монет 3 коп.

Берется 2 монеты, вторая имеет номинал 20

Задача: вероятность, что и первая монета имеет номинал 20.

Решение:

Условная вероятность извлечения первой монетой 20 коп, при условии, что вторая монета 20 коп:

$$P = \frac{P(20+20)}{P(3+20)+P(20+20)} \quad (1)$$

Вероятность, что обе монеты номиналом 20 коп:

$$P(20+20) = \frac{3}{10} \cdot \frac{2}{9} = \frac{6}{90}$$

Вероятность, что сначала достанут номинал 3 коп, а потом 20:

$$P(3+20) = \frac{7}{10} \cdot \frac{3}{9} = \frac{21}{90}$$

Подставим в (1):

$$P = \frac{6}{90} \cdot \frac{90}{27} = \frac{2}{9}$$

Ответ: 2/9

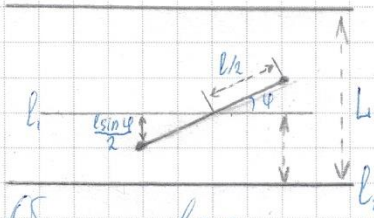
с. Задача 3.22

3.22 **Условие**

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

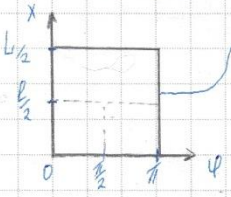
Решение

1. Проведем через центр иглы прямую l_1, параллельную горизонтальной прямой.
2. Обозначим ближайшую к ней параллельную линию через l_2
3. Пусть x - расстояние от центра иглы до ближайшей прямой l_2
4. φ - угол между прямой l_1 и той частью иглы, которая ближе к l_2



5. Область возможных значений x и φ

$x \in [0; \frac{L}{2}]$ $\varphi \in [0; \pi]$

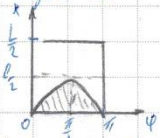


Все возможные значения

$S = \frac{\pi L}{2}$

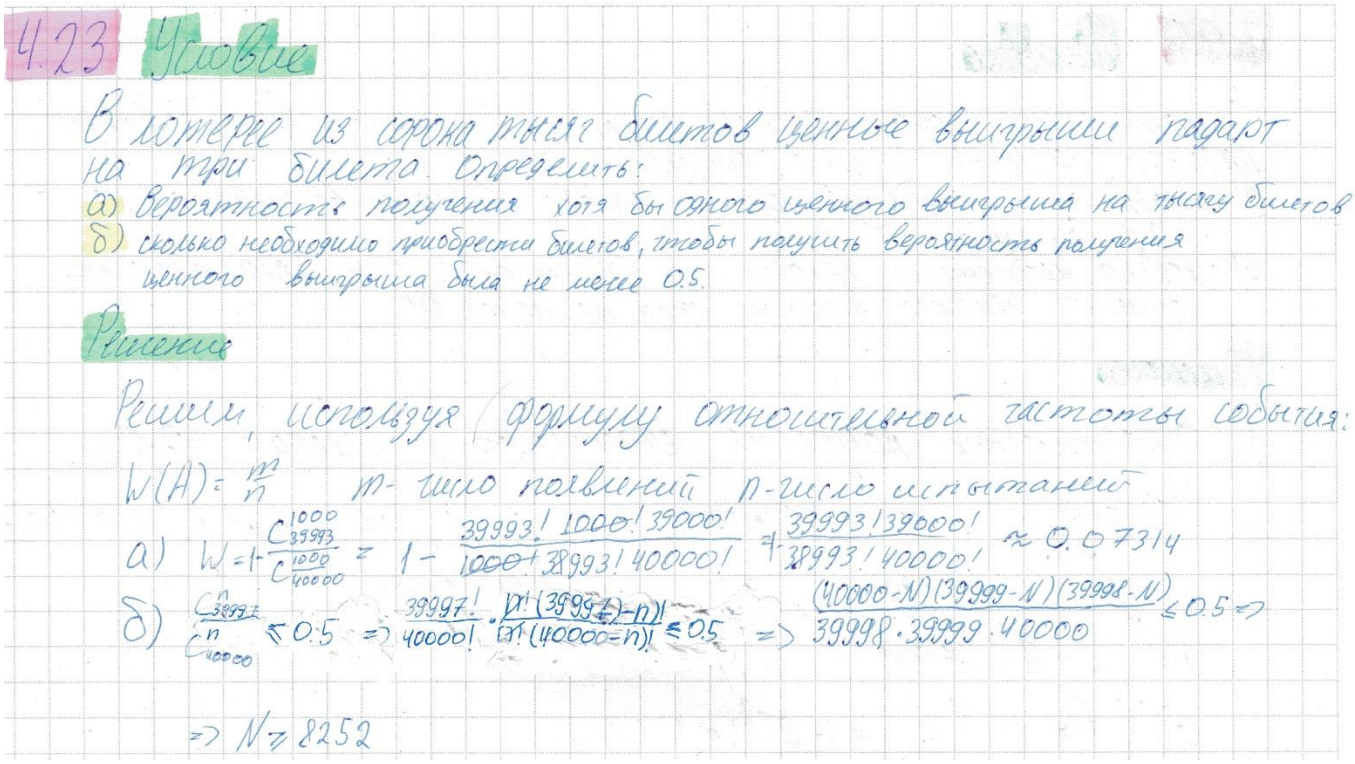
6. Область благоприятных значений

Игла пересечет прямую, если расстояние от прямой l_1 до края иглы, ближайшего к l_2 больше x: $\frac{l \sin\varphi}{2} > x$

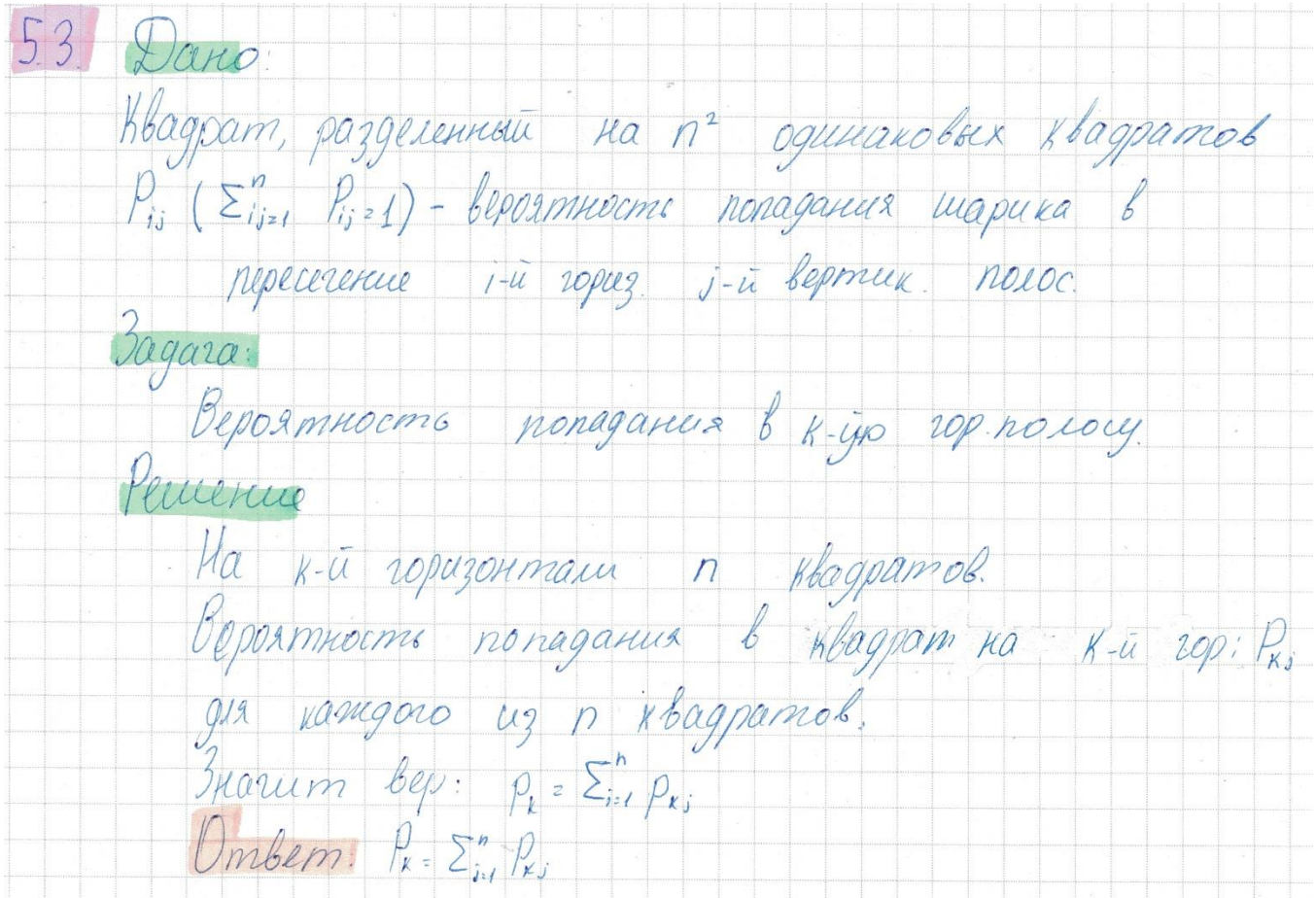


7. Площадь этой области: $S = \int_0^{\pi} \frac{l \sin\varphi}{2} d\varphi = \frac{l}{2} \cos\varphi \Big|_0^{\pi} = \frac{l}{2}(1-(-1)) = l$
8. **Искомая вероятность:** $P(A) = \frac{S}{S} = \frac{2l}{\pi L}$

d. Задача 4.23



e. Задача 5.3



3. Моделирование

a. 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.

b. 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 ($\frac{lsin(fi)}{2} > x$):

```
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.

c. Задача 4.23

Условие:

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

a) Вероятность получения хотя бы одного ценного выигрыша на тысячу билетов
b) 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)}')
```

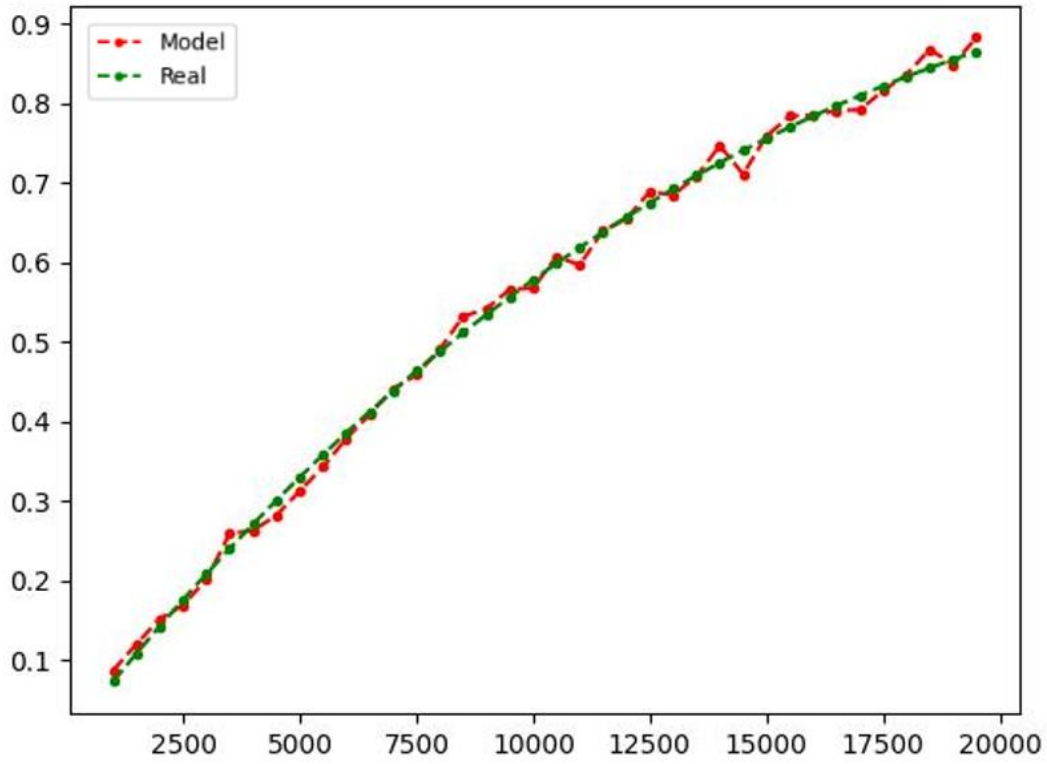
Для решения пункта b уменьшим точность подсчетов до 1000. Считать будем вероятность от 1000 и до 20000 с шагом 500, чтоб получить график изменения погрешности:

```
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.

d. Задача 5.3

Условие:

Квадрат разделен на n^2 одинаковых квадратов.

P_{ij} $(\sum_{j=1}^{n} P_{kj} = 1)$ – 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 P_{ij}, 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.

4. Ссылки

Ссылка на репозиторий с моделированием: [github.com]