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

1.7 **Дано:**

4 изделия

A – хотя бы одно из имеющихся четырех изделий бракованное

B – бракованных изделий среди них не менее двух

Решение:

а) Событие A означает, что 1 или 2 или 3 или 4 изделий бракованные =>

=> $\bar{A}$ означает, что все 4 изделия исправны.

б) Событие B означает, что 2 или 3 или 4 изделия бракованные =>

=> $\bar{B}$ означает, что бракованных либо нет, либо всего одно изделие.

Ответ: $\bar{A}$ – все 4 изделия исправны

$\bar{B}$ – бракованных либо нет, либо 1 изделие.

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: $2/9$

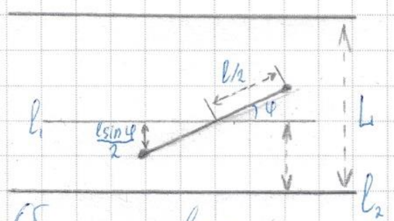
c. Задача 3.22

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 Бюфона)

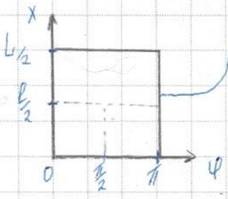
Решение

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

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

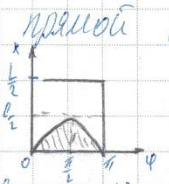


Все возможные значения.

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

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

4.23 Условие

В лотерее из сорока тысяч билетов ценные выигрыши падают на три билета. Определить:

а) вероятность получения хотя бы одного ценного выигрыша на тысячу билетов

б) сколько необходимо приобрести билетов, чтобы получить вероятность получения ценного выигрыша была не менее 0.5.

Решение

Решим, используя формулу относительной частоты события:

$W(A) = \frac{m}{n}$ m - число появлений n - число испытаний

а) $W = 1 - \frac{C_{39993}^{1000}}{C_{40000}^{1000}} = 1 - \frac{39993!\,\cancel{1000!}\,39000!}{\cancel{1000!}\,38993!\,40000!} = 1 - \frac{39993!\,39000!}{38993!\,40000!} \approx 0.07314$

б) $\frac{C_{39997}^{n}}{C_{40000}^{n}} \leq 0.5 \Rightarrow \frac{39997!}{40000!} \cdot \frac{n!\,(39997-n)!}{n!\,(40000-n)!} \leq 0.5 \Rightarrow \frac{(40000-N)(39999-N)(39998-N)}{39998 \cdot 39999 \cdot 40000} \leq 0.5 \Rightarrow$

$\Rightarrow N \geq 8252$

e. Задача 5.3

5.3. Дано:

Квадрат, разделенный на n^2 одинаковых квадратов

P_{ij} $(\sum_{i,j=1}^{n} P_{ij} = 1)$ - вероятность попадания шарика в пересечение i-й гориз. j-й вертик. полос.

Задача:

Вероятность попадания в k-ую гор. полосу.

Решение

На k-й горизонтали n квадратов.

Вероятность попадания в квадрат на k-й гор: P_{kj} для каждого из n квадратов.

Значит вер: $P_k = \sum_{j=1}^{n} P_{kj}$

Ответ: $P_k = \sum_{j=1}^{n} P_{kj}$

f. Задача 6.14

6.14 Дано:

В ящике 15 теннисных мячей, 9 из них новые.

Для первой игры наугад берутся три мяча.

После игры они возвращаются в ящик.

Для 2 игры наугад берутся 3 мяча.

Найти:

Вероятность, что все м и, взятые для 2 игры новые.

Решение:

В коробке: 15 мячей: 6 старых, 9 новых.

Гипотезы:

H_1 = Для 1 игры 3 нов. мяча. Станет 9 старых и 6 новых мячей.

H_2 = Для 1 игры 2 новых и 1 стар мяч. Станет 8 старых и 7 новых мяч.

H_3 = Для 1 игры 1 новых и 2 стар. мяч. Станет 7 стар и 8 нов. мяч.

H_4 = Для 1 игры 3 старых мяч. Станет 6 стар и 9 нов. мяч.

Вероятность гипотез:

$P(H_1) = \frac{C_9^3}{C_{15}^3} = \frac{84}{455}$ $P(H_2) = \frac{C_9^2 C_6^1}{C_{15}^3} = \frac{216}{455}$

$P(H_3) = \frac{C_9^1 C_6^2}{C_{15}^3} = \frac{135}{455}$ $P(H_4) = \frac{C_6^3}{C_{15}^3} = \frac{20}{455}$

A - для 2 игры 3 новых мяча

$P(A|H_1) = \frac{C_6^3}{C_{15}^3} = \frac{20}{455}$ $P(A|H_2) = \frac{C_7^3}{C_{15}^3} = \frac{35}{455}$

$P(A|H_3) = \frac{C_8^3}{C_{15}^3} = \frac{56}{455}$ $P(A|H_4) = \frac{C_9^3}{C_{15}^3} = \frac{84}{455}$

Вероятность A:

$A = P(A|H_1)P(H_1) + P(A|H_2)P(H_2) + P(A|H_3)P(H_3) + P(A|H_4)P(H_4) = 0.089$

Ответ: 0.089

g. Задача 7.16

7.16 **Дано:**

n – студентов

n_k $(k=1,2,3)$ – на k-ом году

Найти:

Среди наудачу выбранных 2-х студентов один учится дольше второго.

Какова вероятность, что этот студент учится 3 год.

Решение

$P(H_1)=\frac{n_2}{n-1}$ – студент, который учится дольше, учится 2 год

$P(H_2)=\frac{n_3}{n-1}$ – студент, который учится дольше учится 3 год

A – один из студентов учится дольше другого.

$P(A|H_1)=\frac{n_1}{n-1}$ – чтобы 1 студ. меньше 2, кот. учится 2 год необх. выбирать среди 1 года

$P(A|H_2)=\frac{n_1+n_2}{n-1}$ – чтоб 1 студ. меньше 2, котор. уч. 3 год необх. выбирать среди 1 или 2 года.

$$P(H_2|A)=\frac{P(A|H_2)\cdot P(H_2)}{P(A|H_1)\cdot P(H_1)+P(A|H_2)\cdot P(H_2)}=\frac{\frac{n_1+n_2}{n-1}\cdot\frac{n_3}{n-1}}{\frac{n_1}{n-1}\cdot\frac{n_2}{n-1}+\frac{n_1+n_2}{n-1}\cdot\frac{n_3}{n-1}}=$$

$$=\frac{(n_1+n_2)\cdot n_3}{n_1\cdot n_2+(n_1+n_2)\cdot n_3}$$ – вероятность, что среди двух выбранных студентов (один из которых уч. дольше) один учится третий год.

Ответ: $\frac{(n_1+n_2)\cdot n_3}{n_1\cdot n_2+(n_1+n_2)n_3}$

h. Задача 8.40

8.40. **Дано:**

Ящик: 20 белых 2 черных

Найти:

Шар извлекается n раз по 1 и возвр.

Определить мин. число n, чтоб вер. черного шара > 0.5

Решение:

Воспользуемся формулой: $n \geq \frac{\ln(1-P)}{\ln(1-p)}$, где

$P = 0.5$ – треб. шанс

$p = \frac{2}{22}$ – шанс. черного шара

$\Rightarrow n \geq 7.27 \Rightarrow n = 8$

Ответ: 8

i. Задача 9.20

9.20 **Дано:**

n игральных костей.

Найти:

а) Вероятность, что Σ очков на верхних гранях $=$ заданному числу m

б) не больше m

Решение

$P_j = \frac{1}{k}$ $j = \overline{1, k}$ тогда производ. ф-я суммы ном. появивш. событий при n исп.:

$G(n) = \frac{1}{k^n}(u + \ldots + u^k)^n$

а) Вероятность P_m - сумма равна m - коэф. при u^m в разл. $G(n) = \frac{u^n}{6^n}(1 + u + \ldots + u^5)^n = \frac{u^n}{6^n} \cdot \left(\frac{1-u^6}{1-u}\right) =$

$= \frac{u^n}{6^n} \sum_{k=0}^{n} C_n^k (-1)^k u^{6k} \cdot \sum_{s=0}^{\infty} C_{n+s+1}^{n-1} u^s \Rightarrow P_m = \frac{1}{6^n}\left[C_{m-1}^{n-1} - C_n^1 C_{m-7}^{n-1} + C_n^2 C_{m-13}^{n-1} \ldots\right]$

Суммируем, пока $n + 6k \leq m$

б) $R_m = \sum_{k=n}^{m} P_k = \frac{1}{6^n}\left[(1 + C_n^{n-1} + \ldots + C_{m-1}^{n-1}) - C_n^1 (C_{m-7}^{n-1} + C_{n-6}^{n-1} + \ldots + C_{m-7}^{n-1}) + \ldots\right] = \frac{1}{6}\left[C_m^n - C_n^1 C_{m-?}^n + \ldots\right]$

при $n = 10$ $m = 20$ $10 + 6k \leq 20 \Rightarrow k = 0$ $k = 1$

$P_{20} = \frac{1}{6^{10}}\left[\frac{19!}{9! \cdot 10!} - 10 \cdot \frac{13!}{9!4!}\right] \approx 0.0014$ $\quad P_{20} = \frac{1}{6^{10}}\left[\frac{20!}{10!10!} - 10 \cdot \frac{14!}{10!4!}\right] \approx 0.0029$

Ответ: а) 0.0014 б) 0.0029

j. Задача 10.7

10.7. Условие:

Подаются сигналы на вкл. прибора

Интервал сигналов: 5с

Интервал для вкл: 16с

Шанс вкл. $\frac{1}{2}$

Найти:

Ряд распределения

Производящую функцию.

Решение.

Первые сигн. дойдут только через 16с =>

$\Rightarrow P(1) = P(2) = P(3) = 0$

После подачи 4 сигнала через 1 сек дойдет 1 сигнал =>

$\Rightarrow P(4) = p = \frac{1}{2}$

Аналогично:

$P(5) = pq = \frac{1}{4} \quad P(6) = pq^2 = \frac{1}{8} \Rightarrow P(k) = pq^{k-4} = 2^{3-k} \quad k \geq 4$

Получим геометр. распр. со смещ. центром;

Найдем производ. ф-цию:

$$G(u) = \sum_{k=4}^{\infty} P(k)u^k = \sum_{k=4}^{\infty} 2^{3-k} u^k = \frac{u^4}{2} \sum_{k=0}^{\infty} \left(\frac{u}{2}\right)^k = \frac{u^4}{2} \lim_{N\to\infty} \frac{1-\left(\frac{u}{2}\right)^N}{1-\frac{u}{2}} = \frac{u^4}{2-u}$$

Ответ: $P(k) = \begin{cases} 0 & k = 1,2,3 \\ 2^{3-k} & k \geq 4 \end{cases}$; $G(u) = \frac{u^4}{2-u}$

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:

```python
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:

```python
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:

```python
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:

```python
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()
```

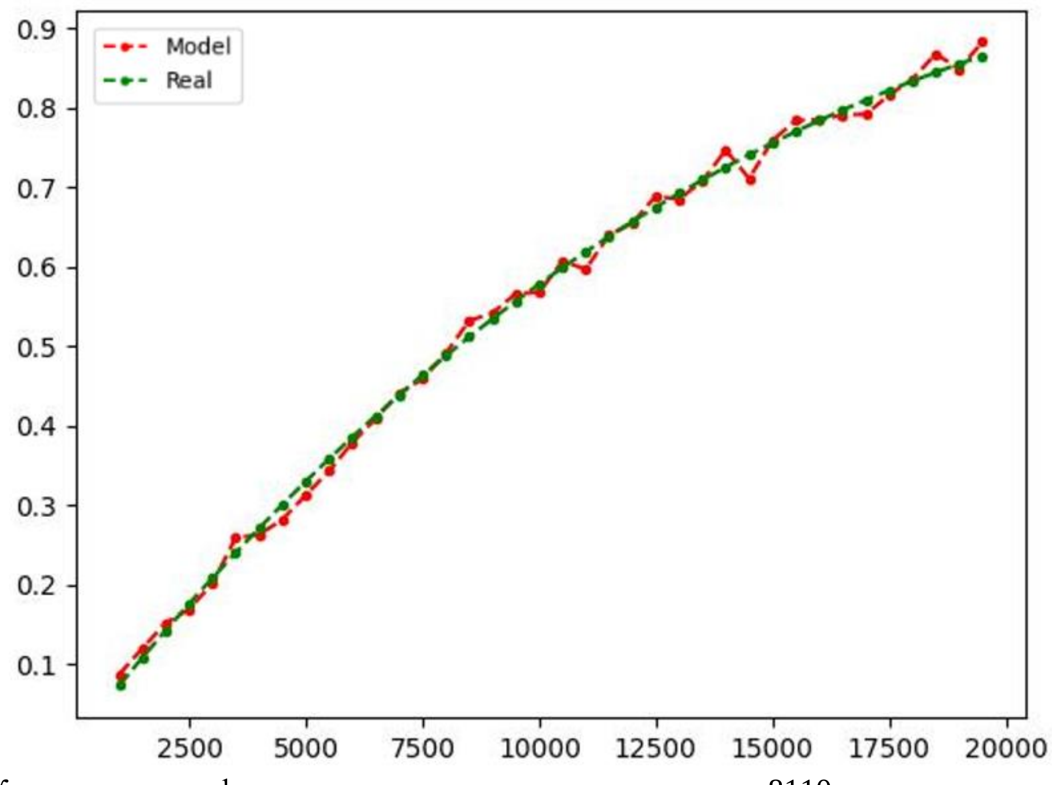
Выполним запуск программы и посмотрим на результат:

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:

```python
def get_input_data():
    # Размерность массива
    n = 10
    return n
```

Создадим массив вероятностей P_{ij}, сумма элементов этого массива n на n равна единице:

```python
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:

```python
n = get_input_data()
P = generate_array(n)
k = randint(0, n - 1)
count_in_k = 0
count_iterations = 1000000
```

Создадим основной цикл программы:

```python
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:

```python
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.

e. Задача 6.14

Условие:

В ящике находятся 15 теннисных мячей, из которых 9 новых. Для первой игры наугад берутся три мяча, которые после игры возвращаются в ящик. Для второй игры также наугад берутся три мяча.

Задача:

Найти вероятность того, что все мячи, взятые для второй игры, новые.

Решение:

Создадим функцию для получения входных данных – в данной задаче это состав коробки и количество вытаскиваемых шаров:

```python
def get_input_data():
    box = [1] * 9 + [0] * 6
    count_to_taken = 3
    return box, count_to_taken
```

Далее создадим симуляцию одной игры:

```python
def simulate_game(box, count_to_taken):
    random.shuffle(box)
    first_game = random.sample(box, count_to_taken)  # выбираем 3 мяча для первой игры
    for ball in first_game:
        box.remove(ball)  # удаляем выбранные мячи из ящика
    box.extend([0] * count_to_taken)
    second_game = random.sample(box, count_to_taken)  # выбираем 3 мяча для второй игры
    return all(ball == 1 for ball in second_game)  # проверяем, все ли мячи новые
```

Напишем основную функцию, вызывая симуляцию множество раз:

```python
def main():
    box, count_to_taken = get_input_data()
    num_experiments = 1000000  # количество экспериментов
    num_successes = 0  # количество успешных экспериментов

    for _ in range(num_experiments):
        if simulate_game(box.copy(), count_to_taken):
            num_successes += 1

    probability = num_successes / num_experiments
    print(f'Вероятность того, что все мячи для второй игры будут новыми: {probability}\n'
          f'Теоретическая вероятность: 0.089 для коробки 9 новых мячей и 6 старых')
```

Полученный вывод:

```
Вероятность того, что все мячи для второй игры будут новыми: 0.089452
Теоретическая вероятность: 0.089 для коробки 9 новых мячей и 6 старых
```

Как видно результат моделирования совпадает с теоретическими ожиданиями.

f. Задача 7.16

Условие:

n – студентов

n_k (k = 1, 2, 3) – на k-ом году обучения

Задача:

Наудачу берут 2 студента, один из которых учится дольше второго.

Какова вероятность, что этот студент учится 3-й год

Решение:

Создадим функцию для получения начальных данных – сколько студентов на каждом году обучения:

```python
def get_input_data():
    n1 = 3
    n2 = 4
    n3 = 3
    return n1, n2, n3
```

После этого создадим основной цикл, где выбираем двух студентов случайным образом:

```python
first_student = random.randint(0, n - 1)
second_student = random.randint(0, n - 1)
while students[second_student] == students[first_student]:
    second_student = random.randint(0, n - 1)
if students[first_student] == 3 or students[second_student] == 3:
    count += 1
```

Будем делать это множество раз:

```
n = n1 + n2 + n3
print("n =", n)
print("n1 =", n1)
print("n2 =", n2)
print("n3 =", n3)
N = 100000
count = 0
students = [1 for _ in range(n1)] + [2 for _ in range(n1, n1 + n2)] + [3 for _ in range(n1 +
n2, n)]
for j in range(N):
    first_student = random.randint(0, n - 1)
    second_student = random.randint(0, n - 1)
    while students[second_student] == students[first_student]:
        second_student = random.randint(0, n - 1)
    if students[first_student] == 3 or students[second_student] == 3:
        count += 1
```

Выведем результат на экран:

```
print('Вероятность того, что среди двух выбранных студентов\n'
      '(один из которых учится дольше другого) один учится третий год:\n'
      f'Моделирование: P = {count / N:.05f}\n'
      f'Аналитически: P = {((n1 + n2) * n3) / (n1 * n2 + (n1 + n2) * n3):.05f}')
```

Полученный вывод:

```
n = 10
n1 = 3
n2 = 4
n3 = 3
Вероятность того, что среди двух выбранных студентов
(один из которых учится дольше другого) один учится третий год:
Моделирование: P = 0.63178
Аналитически: P = 0.63636
```

Видно, что полученный результат вполне соответствует ожиданиям.

g. Задача 8.40

Условие:

Ящик: 20 белых 2 черных шара.

Задача:

Шар извлекается n раз по 1 и возвращается. Определить минимальное число n, чтоб вероятность черного шара была больше 0.5

Решение:

Создадим функцию для получения начальных данных – сколько студентов на каждом году обучения:

```
def get_input_data():
    count_black = 2
    count_white = 20
    return count_white, count_black
```

Далее сделаем функцию для получения одного мяча:

```
def get_one_black_ball(count_black, count_white):
    return random.randint(1, count_black + count_white) <= count_black
```

Будем вытаскивать мячи до тех пор, пока не встретим черный. Если встретили черный, записали, каким именно он выпал:

```
count_white, count_black = get_input_data()
try_to_get = 1_000_000
max_n = 200
count_black_mas = np.zeros(max_n)
n_now = 1
count_iterations = 0
for i in range(try_to_get):
    if get_one_black_ball(count_black, count_white):
        count_black_mas[n_now - 1] += 1
        count_iterations += 1
        n_now = 0
    n_now += 1
```

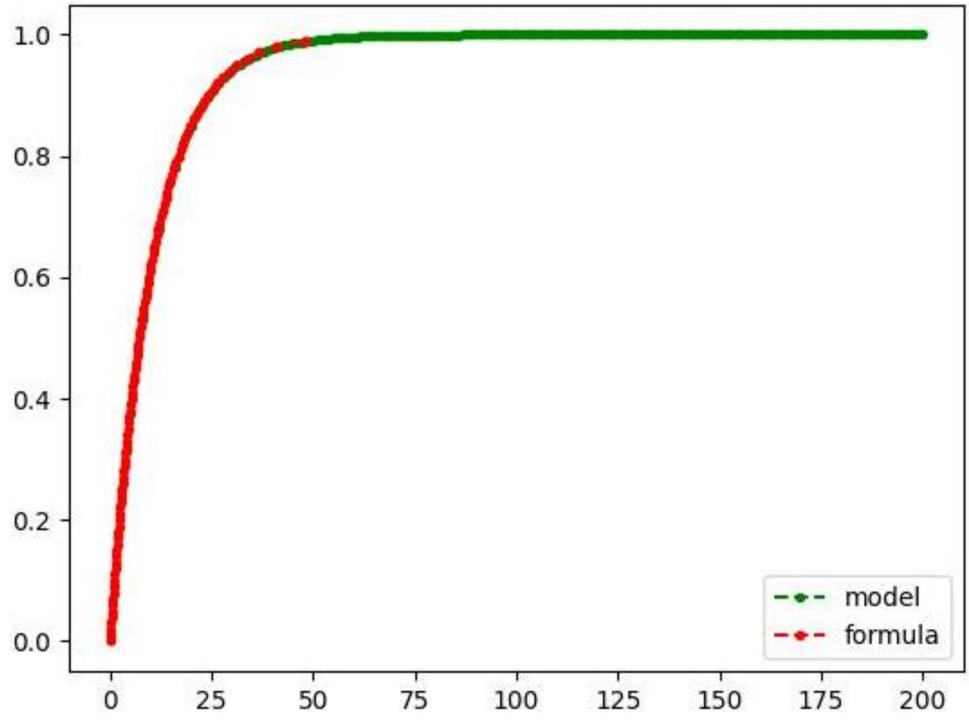
Основываясь на этом, получим шансы выпадения:

```
chance = count_black_mas
for i in range(1, max_n):
    chance[i] += chance[i - 1]
chance = chance / count_iterations
```

Выведем график, сравнивающий данные при теоретическом решении и полученные в ходе моделирования:

```
formula = [log(1 - i / 100) / log(1 - count_black / (count_white + count_black)) for i in
range(0, 100)]
plt.plot(range(1, max_n + 1), chance, label='model', linestyle='--', color='g', marker='o',
markersize=3)
plt.plot(formula, [i / 100 for i in range(0, 100)], label='formula', linestyle='--',
color='r', marker='o',
         markersize=3)
plt.legend()
plt.savefig("pictures/8.40.Chance.jpg")
plt.show()
```

Полученный график:



Как можно видеть, графики совпадают, что значит, что результат моделирования соответствует теоретическом решению.

h. Задача 9.20

Условие:

Дано n игральных костей

Задача:

Найти вероятность того, что сумма очков на верхних гранях равна заданному числу m; не больше m.

Решение:

Зададим начальные данные:

```
############
MAX_NUM = 5_000_000
n = 10
m = 20
############
```

Создадим модель, а в качестве точно решения оставим число:

```
def model():
    sumPoints = 0
    for i in range(n):
        sumPoints += random.randint(1, 6)
    return sumPoints <= m


def solve():
    return 0.0029
```

Проведем симуляцию:

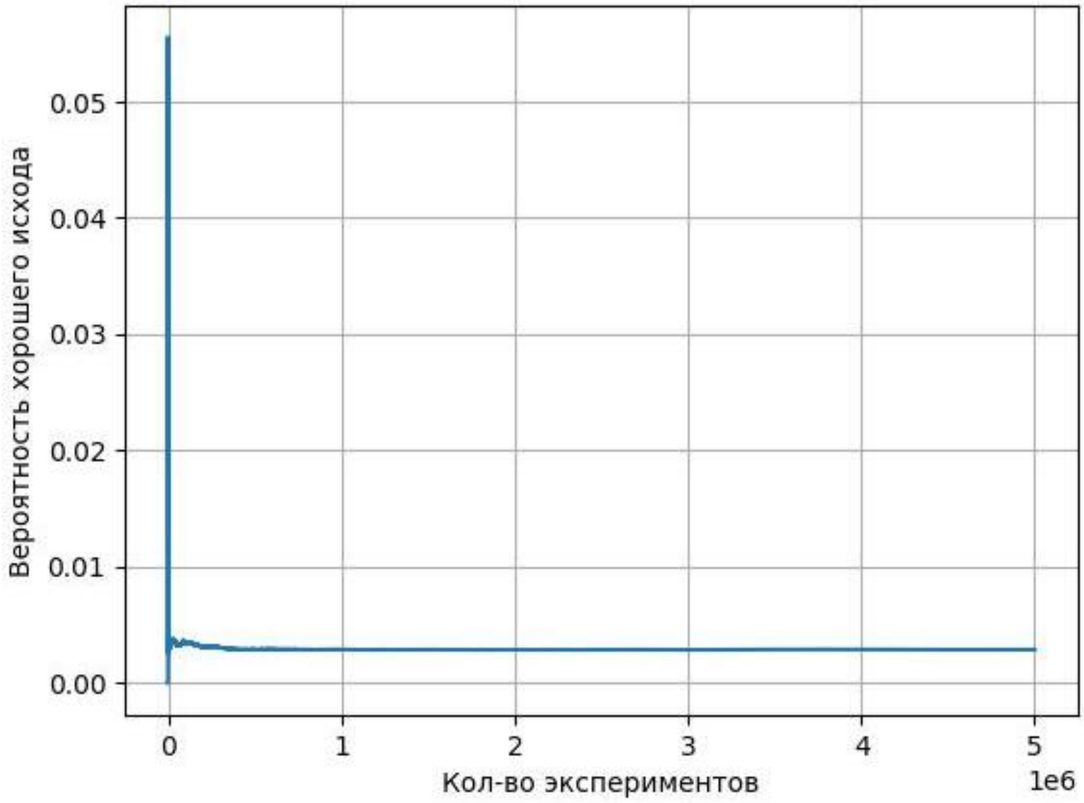
```
def main():
    allResults = 0
    goodResults = 0
    x = []
    y = []
    moda = {}
    while allResults != MAX_NUM:
        if model():
            goodResults += 1
        allResults += 1
        x.append(allResults)
        value = goodResults / allResults
        y.append(value)
        if moda.get(value) is None:
            moda[value] = 0
        moda[value] += 1
```

Выведем полученные результаты на экран в виде графика и в таблицы:

```
def print_table_with_graph(x, y, modaOfY, analytic, name):
    values = []
    for item in modaOfY.items():
        values.append(item)
    values.sort(key=lambda v: v[1], reverse=True)
    print(f'Аналитически ответ: {analytic}')
    th = ['МОДА', 'ЗНАЧЕНИЕ', 'ПОГРЕШНОСТЬ']
    table = PrettyTable(th)
    for v in values[:3]:
        table.add_row([v[1], "{0:.10f}".format(v[0]), "{0:.6f}".format(abs(analytic -
v[0]))])
    print(table)
    plt.xlabel('Кол-во экспериментов', color='black')
    plt.ylabel('Вероятность хорошего исхода', color='black')
    plt.grid(True)
    plt.plot(x, y)
    plt.savefig(name)
    plt.show()
```

Полученный вывод:

```
Аналитически ответ: 0.0029
+------+-------------+------------+
| МОДА |   ЗНАЧЕНИЕ   | ПОГРЕШНОСТЬ |
+------+-------------+------------+
| 248  | 0.0028818444 |   0.000018  |
| 196  | 0.0028901734 |   0.000010  |
| 158  | 0.0028735632 |   0.000026  |
+------+-------------+------------+
```



Как видно из таблицы и графика, ответ совпадает с аналитическим.

i. Задача 10.7

Условие:

Подаются сигналы на включение прибора.

Интервал сигналов: 5 секунд

Интервал для включения: 16 секунд

Шанс для включения: ½

Найти:

Ряд распределения

Производящую функцию

Решение:

Напишем функцию для получения входных данных:

```
def get_input_data():
    probability = 1 / 2
    time_signal_interval = 5
    time_delay = 16
    return probability, time_signal_interval, time_delay
```

Напишем функцию для генерации сигнала:

```
def generate_signals(probability, time_interval, delay, break_time):
    time = 0
    count_signals = 1
    while time + delay <= break_time:
        if np.random.random() < probability:
            return time + delay, count_signals + 3
        count_signals += 1
        time += time_interval
    return -1
```

Это одна итерация симуляции.

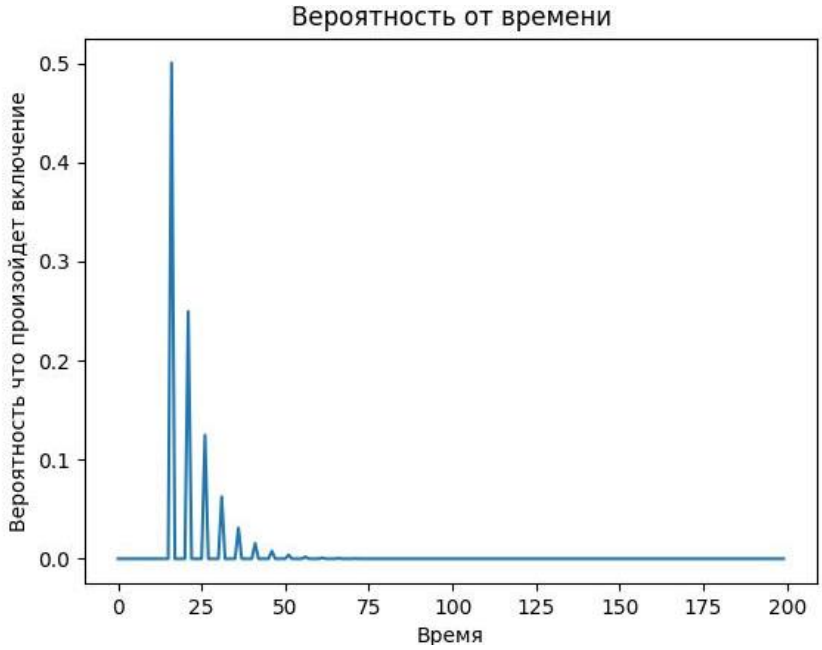
Напишем функцию для многократного запуска симуляции:

```
def simulate(num_trials, probability, time_interval, delay, break_time):
    signal_times = [0] * break_time
    count_signals_mas = [0] * break_time
    counter = 0
    for _ in range(num_trials):
        signal_time, count_signals = generate_signals(probability, time_interval, delay,
break_time)
        if signal_time != -1:
            counter += 1
            signal_times[signal_time] += 1
            count_signals_mas[count_signals] += 1

    return signal_times, count_signals_mas, counter
```

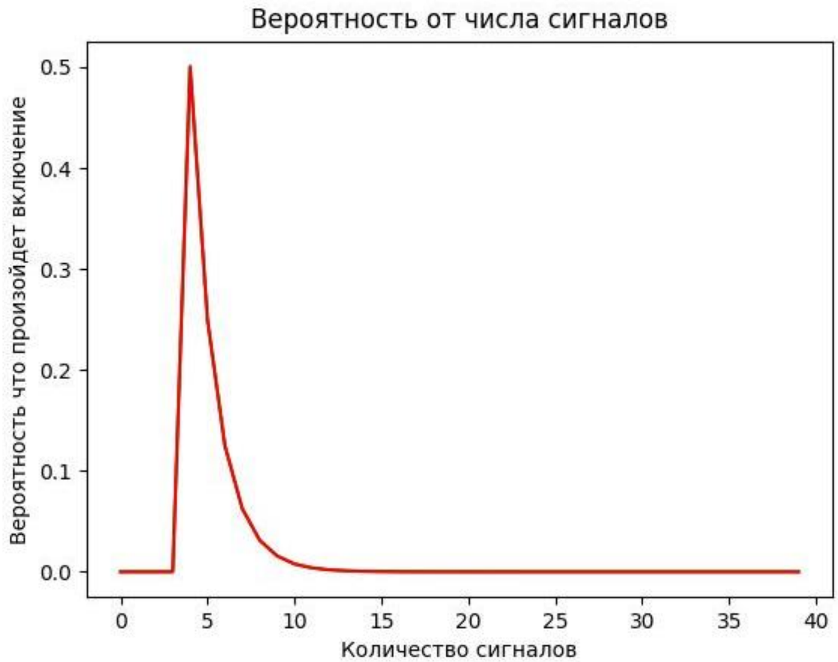
Выведем на экран график зависимости включения устройства от времени и включения устройства от числа сигналов:

```
def main():
    count_simulations = 1_000_000
    break_time = 200
    probability, time_signal_interval, time_delay = get_input_data()
    signals, count_signals_mas, count_signals = simulate(count_simulations, probability,
time_signal_interval,
                                                         time_delay, break_time)
    plt.plot(range(break_time), [signal / count_signals for signal in signals])
    plt.title('Вероятность от времени')
    plt.xlabel('Время', color='black')
    plt.ylabel('Вероятность что произойдет включение', color='black')
    plt.savefig('pictures/10.7.Chance_1.jpg')
    plt.show()

    plt.plot(range(break_time // time_signal_interval),
             [count / count_signals for count in count_signals_mas[:break_time //
time_signal_interval]], color='g')
    plt.plot(range(break_time // time_signal_interval),
             [0, 0, 0, 0, *[2 ** (3 - i) for i in range(4, break_time //
time_signal_interval)]], color='r')
    plt.title('Вероятность от числа сигналов')
    plt.xlabel('Количество сигналов', color='black')
    plt.ylabel('Вероятность что произойдет включение', color='black')
    plt.savefig('pictures/10.7.Chance_2.jpg')
    plt.show()
```

Полученные графики:



На этом рисунке видно, что первые 16 секунд устройства не включались, а потом включались с интервалом в 5 секунд. Чем дальше, тем меньше включений т.к. включения происходили до этого.



На этом графике видно, что результат полностью совпадает с ожидаемым, полученным в ходе теоретического решения.

4. Ссылки

Ссылка на репозиторий с моделированием: [github.com]