Предварительные импорты

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.optimize import minimize_scalar, root_scalar
from math import floor, comb
from tqdm.notebook import trange
```

Задача 1

Однажды в Самарканде турист заказывал Яндекс-такси. На десятом заказе впервые приехал таксист, который уже раньше приезжал к туристу. Для упрощения предположим, что все n таксистов Самарканда всегда на работе и приезжают равновероятно.

[5] Постройте график функции правдоподобия как функции от общего количества такси

n. Найдите оценку числа n методом максимального правдоподобия.

Рассуждение:

На десятом заказе впервые приехал таксист, который уже раньше приезжал к туристу -> первые девять заказов были приняты девятью разными таксистами.

Итоговое событие: P(x|n) = P1(девять разных таксистов на девяти вызовах) * P2(один из девяти предыдущих)

$$P1(x \vee n) = \frac{A_n^k}{n^k}$$

$$P2(x \vee n) = \frac{k}{n}$$

$$P(x \vee n) = \frac{(n-1)\dots(n-8)9}{n^9}$$

```
def logP(n):
    arr = [0] * 9
    for i in range(8):
        arr[i] = (n-i-1) / n
    arr[-1] = 9 / n
    return np.sum(np.log(arr))

def logP_dev(n):
    arr = [0] * 9
```

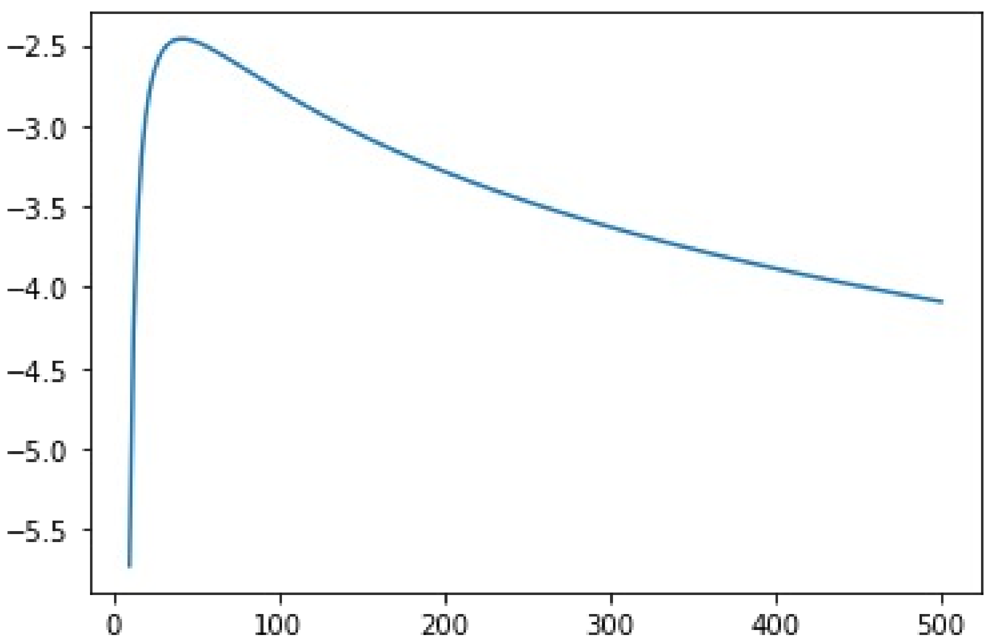
```
    for i in range(8):
        arr[i] = 1/(n-i-1)
    arr[-1] = -9/n
    return np.sum(arr)


P_vec = np.vectorize(logP)
lP_vec = np.vectorize(logP_dev)
```

$$L(n \vee x) = \sum_{i=1}^{8} ln\left(\frac{n-i}{n}\right) + ln\left(\frac{9}{n}\right)$$

$$L(n \vee x) = \sum_{i=1}^{8} ln(n-i) + ln(9) - 9\,ln(n)$$

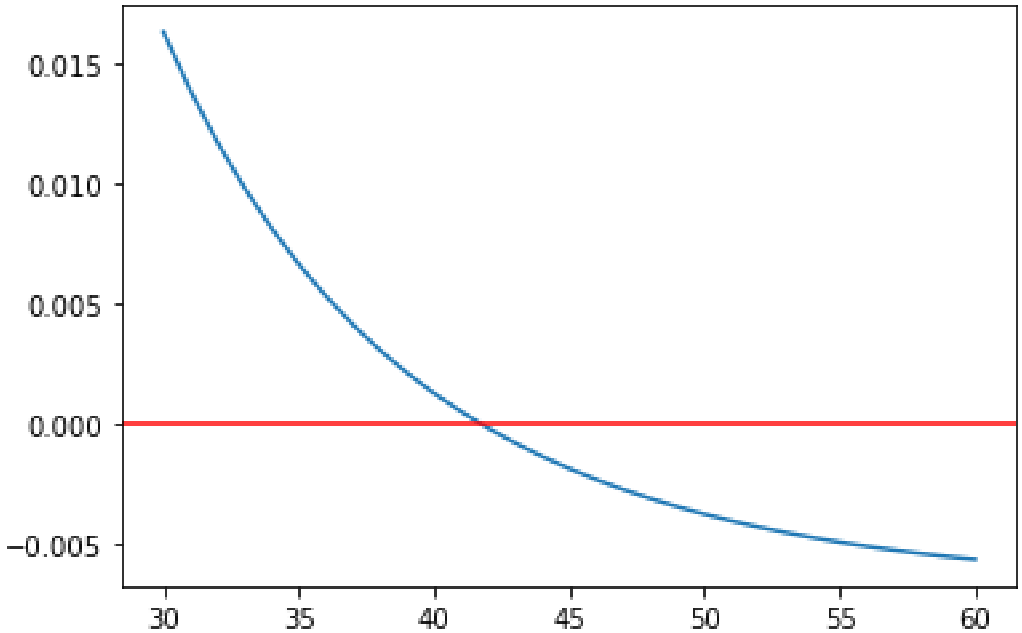
$$L'(n \vee x) = \sum_{i=1}^{8} \frac{1}{n-i} - \frac{9}{n} = 0$$

```
#график логарифма вероятности
x = np.linspace(10, 500, 200)
y = P_vec(x)

plt.plot(x, y);
```




```
#график производной функции правдопободия для поиска максимума
x = np.linspace(30, 60, 31)
y = lP_vec(x)

plt.plot(x, y);
plt.axhline(0, c='r')
```

```
<matplotlib.lines.Line2D at 0x1f86092d2b0>
```




```
root_scalar(logP_dev, bracket=(30,60)).root
```

```
41.77139840710066
```

Ответ: n=42

[5] Постройте график математического ожидания номера заказа, на котором происходит первый повторный приезд, как функции от общего количества такси n. Найдите оценку числа n методом моментов.

$$P1(x \vee k, n) = \frac{A_n^{k-1}}{n^{k-1}}$$

$$P2(x \vee k, n) = \frac{k-1}{n}$$

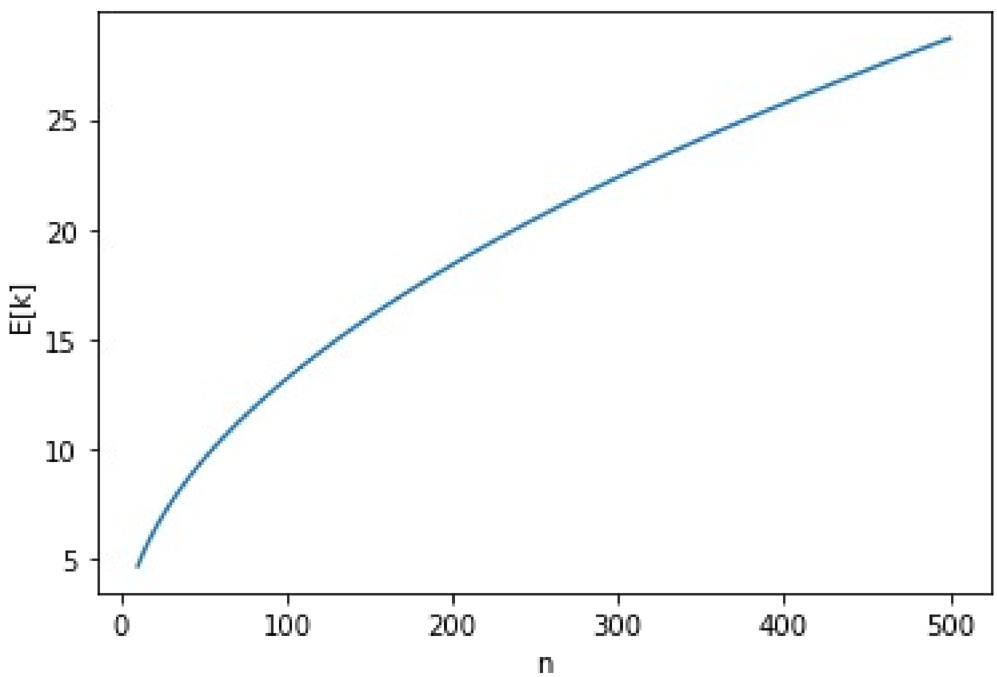
$$P(x \vee k, n) = \frac{(n-1)\dots(n-k+2)(k-1)}{n^{k-1}}, k = \{2..n+1\}$$

Объяснение: k-й заказ оказался повторным, если k-1 заказов оказались разными, а k-й заказ принял один из k-1 ранних водителей

Очевидно, k не может быть меньше 2-х (два заказа принял один и тот же водитель) и больше n+1 (n первых заказов приняли все возможные таксисты)

```
def En(n):
    n = int(n)
    r = 0
```

```
    P = 1
    for k in range(2,n+2):
        P *= (n-k+2)/n
        r += k*P*(k-1)/n
    return r

start=10
end=500
iters = end-start+1
res = np.zeros(iters)
x = np.linspace(start, end, iters, dtype=np.int32)
for n in range(start, end+1):
    E = 0
    res[n-start] = En(n)

plt.plot(x, res);
plt.ylabel('E[k]');
plt.xlabel('n');
```



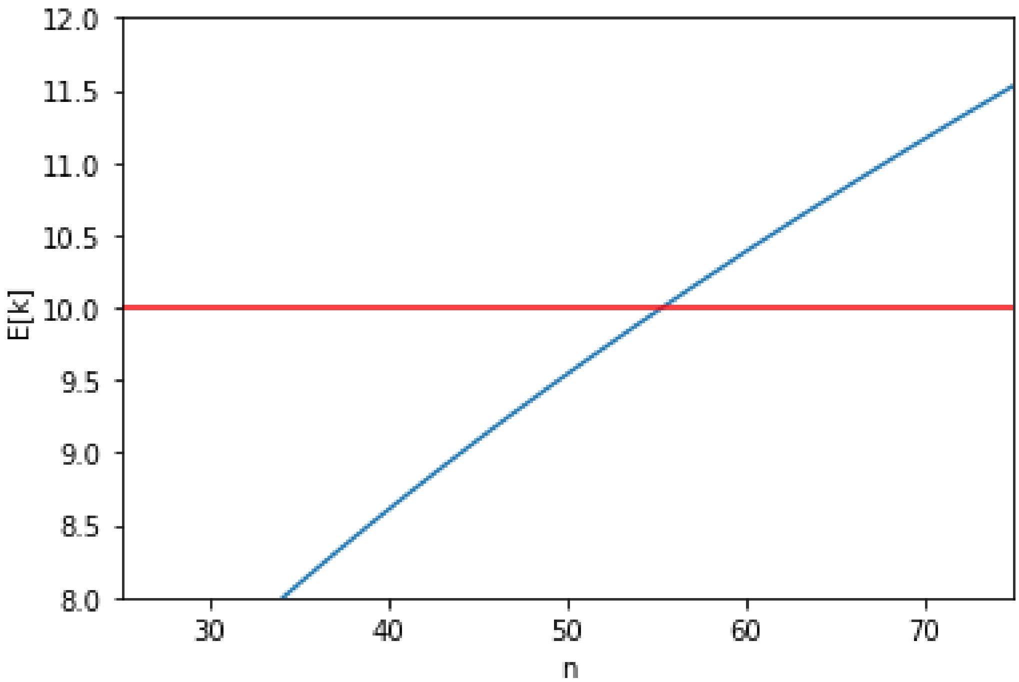

```
En(1)
En(2)
```

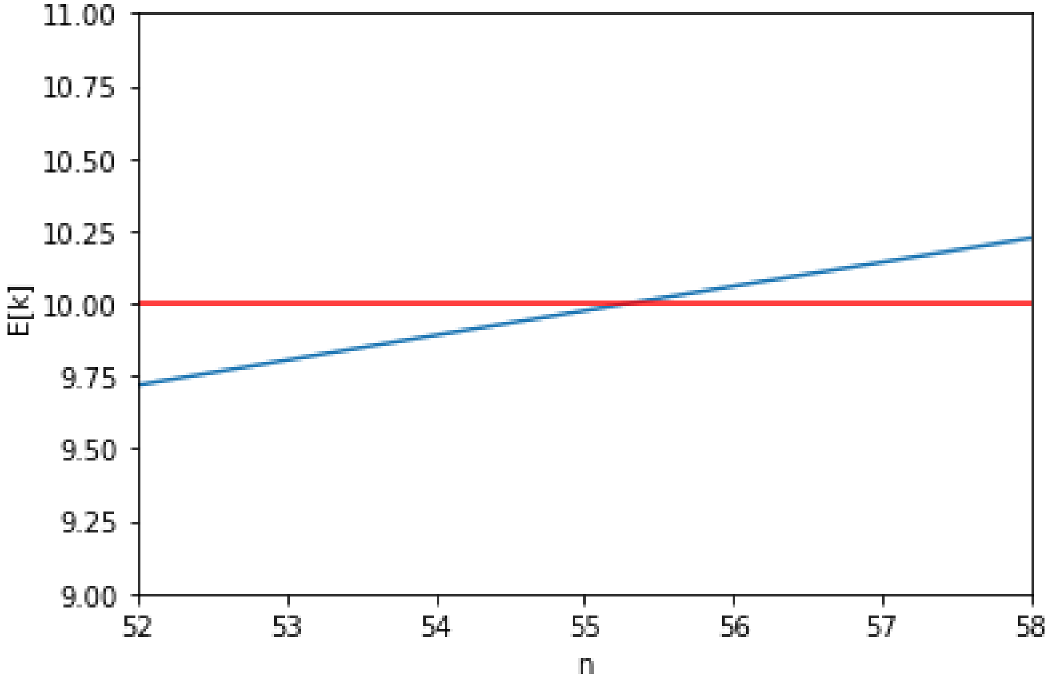
2.5

Считая E[k] = 10, из условия задачи, получим

```
plt.plot(x, res);
plt.xlim(25,75)
plt.ylim(8,12)
plt.ylabel('E[k]')
```

```
plt.xlabel('n')
plt.axhline(10, c='r');
plt.show()

plt.plot(x, res);
plt.xlim(52,58)
plt.ylim(9,11)
plt.ylabel('E[k]')
plt.xlabel('n')
plt.axhline(10, c='r');
```






```
root_scalar(lambda x: En(x) - 10, bracket=(10,100))
```

```
      converged: True
           flag: 'converged'
 function_calls: 47
     iterations: 46
           root: 55.999999999999744
```

Ответ: n=55

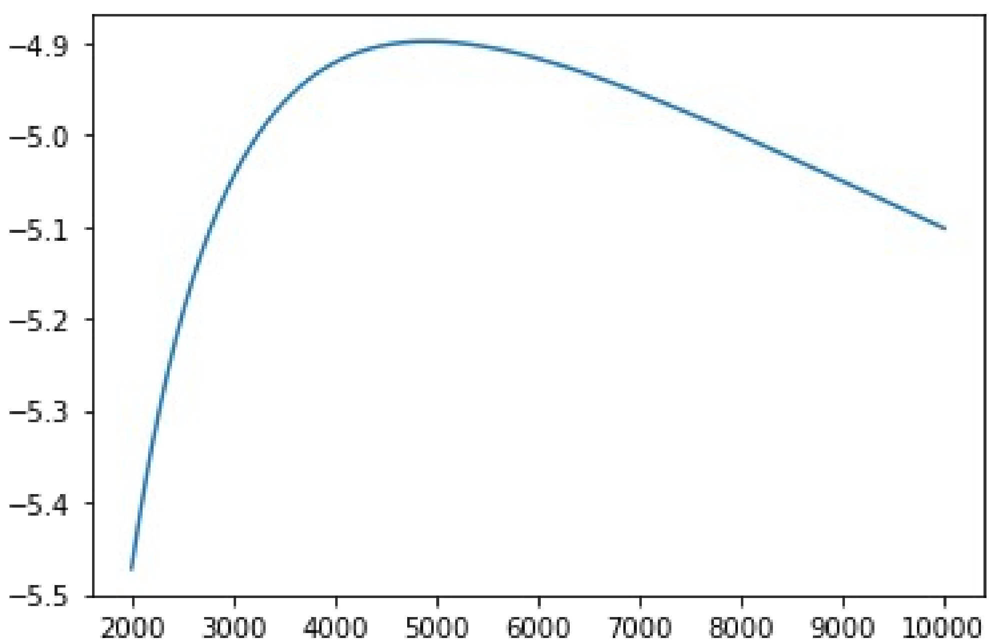
[15] Предположим, что настоящее n равно 100. Проведя 10000 симуляций вызовов такси до первого повторного, рассчитайте 10000 оценок методом моментов и 10000 оценок методом максимального правдоподобия. Постройте гистограммы для оценок двух методов. Оцените смещение, дисперсию и среднеквадратичную ошибку двух методов.

```
def simulate(n):
    hist = np.zeros(100, dtype=np.int32)
    it = 0
    while it <= 100:
        i = np.random.randint(0,100)
        it += 1
        hist[i] += 1
        if hist[i] > 1:
            return it
    return it

def LogDerMP(k,n):
    k = int(k)
```

```
    n = int(n)
    arr = np.zeros(k-1)
    for i in range(k-2):
        arr[i] = 1 / (n-i-1)
    arr[-1] = -(k-1) / n
    return np.sum(arr)

def LogMP(k,n):
    k = int(k)
    n = int(n)
    arr = np.zeros(k-1)
    for i in range(k-2):
        arr[i] = (n-i-1) / n
    arr[-1] = (k-1) / n
    return np.sum(np.log(arr))



lP_vec = np.vectorize(LogMP)
lD_vec = np.vectorize(LogDerMP)
```

Найдём максимумы оценок методов для k=100

```
#график логарифма вероятности
x = np.linspace(2000, 10000, 200)
y = lP_vec(100, x)

plt.plot(x, y);
#plt.axhline(0)
```

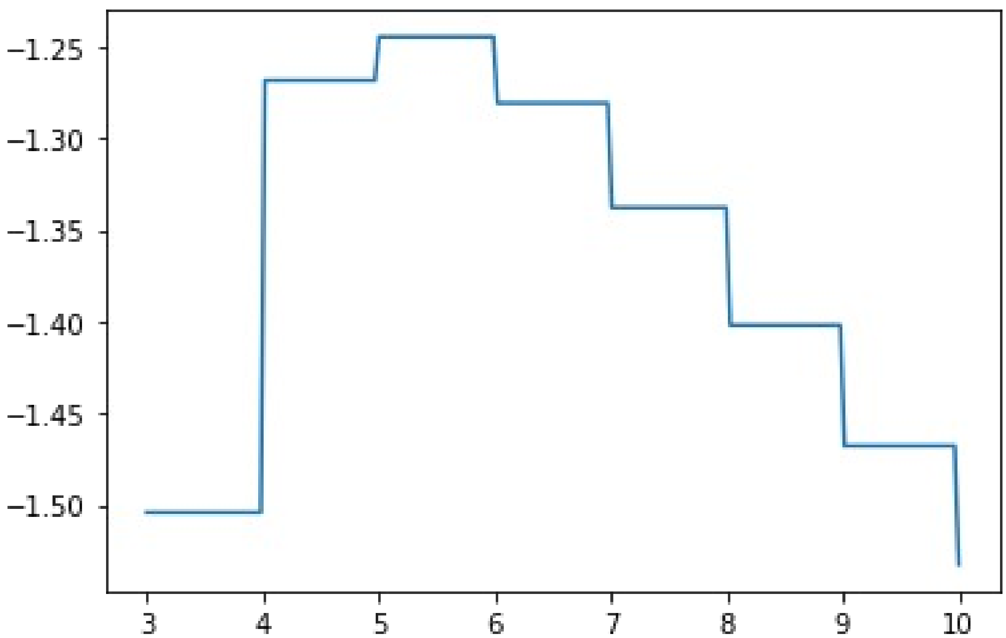


```
En(7000)
```

```
105.52769471030268
```

Эмпирически было найдено, что для метода макс. правдоподобия при k=2,3 n=1,2 соответственно, дальше пик смещается от минимально возможных значений вправо, что позволяет рассмотреть края области для метода root_scalar

```
#график логарифма вероятности
x = np.linspace(3, 10, 200)
y = lP_vec(4, x)

plt.plot(x, y);
#plt.axhline(0)
```

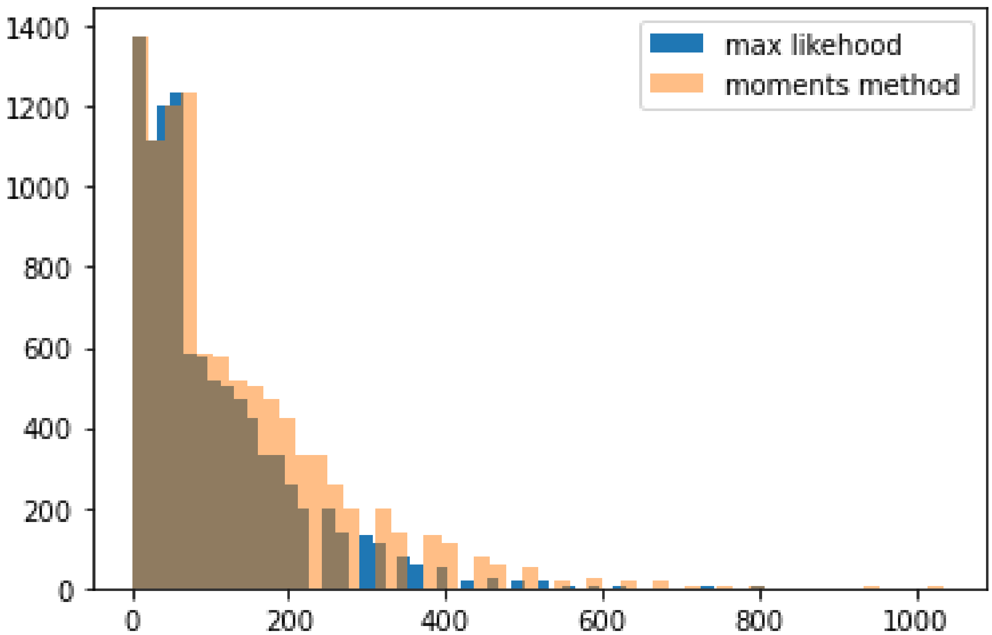



```
np.random.seed(1234)
n = 100
k = 10000
res = np.zeros(k)
mmp = np.zeros(k)
mm = np.zeros(k)
for i in trange(k):
    res[i] = simulate(n) #эксперимент
    #print(res[i])
    if res[i] < 4:
        mmp[i] = res[i]-1
    else:
        resMMP = root_scalar(lambda nx: LogDerMP(res[i],nx), 
bracket=(res[i]-1,7000)) #метод макс правдоподобия
        mmp[i] = resMMP.root
```

```
    resMM = root_scalar(lambda nx: En(nx) - res[i], bracket=(res[i]-
1,7000))
    mm[i] = resMM.root
```

{"model_id":"e1b2cd89bb5d4c989a9b42c6ddff8f81","version_major":2,"version_minor":0}

```
plt.hist(mmp, bins=50, label='max likehood')
plt.hist(mm, bins=50, label='moments method', alpha=0.5)
plt.legend();
```




```
print('Метод макс. правдоподобия:')
print(f'\tСмещение: {mmp.mean() - 100}')
print(f'\tДисперсия: {mmp.var()}')
print(f'\tMSE error: {((mmp-100)**2).mean()}')

print('Метод моментов:')
print(f'\tСмещение: {mm.mean() - 100}')
print(f'\tДисперсия: {mm.var()}')
print(f'\tMSE error: {((mm-100)**2).mean()}')
```

```
Метод макс. правдоподобия:
	Смещение: -2.7311000000002394
	Дисперсия: 8452.361992790004
	MSE error: 8459.820900000002
Метод моментов:
	Смещение: 26.47390000000007
	Дисперсия: 13995.078118789992
	MSE error: 14695.945499999994
```

Задача 2

Однажды в Самарканде турист заказывал Яндекс-такси. На десятом заказе он обнаружил, что у таксистов было 6 разных имён. Для упрощения предположим, что все n имён среди таксистов встречаются равновероятно и независимо от поездки к поездке.

а) [5] Постройте график функции правдоподобия как функции от общего количества имён

n. Найдите оценку числа n методом максимального правдоподобия.

Рассуждение:

k - число заказов

l - число имён за k заказов

n - число всех имён

$ C^l_n $ - выбрать l имён из n

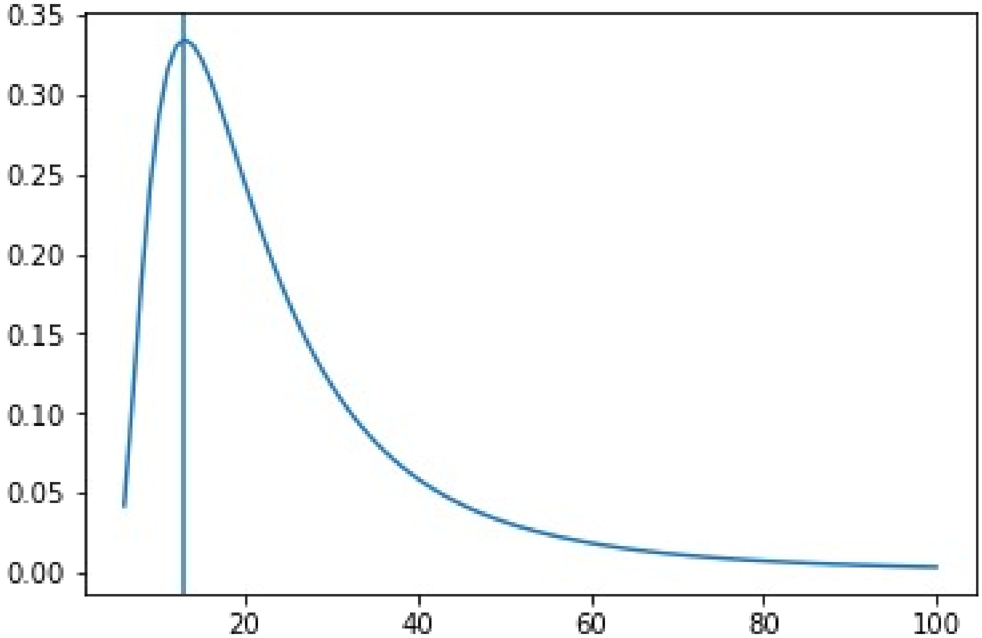
$ C^{l-1}_{k-1} $ - распределелить l имен по k заказов ($x_{k1}+\ldots+x_{kl}=k$, чтобы каждое имя использовалось минимум один раз)

$ C^{n-1}_{k+n-1} $ - число всех возможных подборов имен ($x_1+\ldots+x_n=k$)

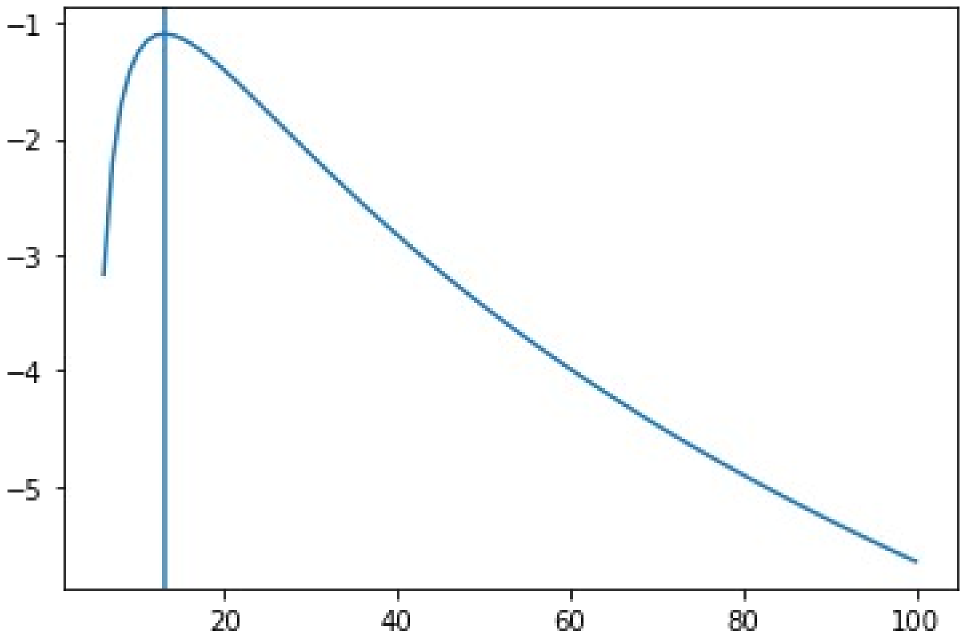
$$P(k,l,n)=\frac{C_n^l * C_{k-1}^{l-1}}{C_{k+n-1}^{n-1}}$$

```
def P(n,k,l):
    return comb(n,l) * comb(k-1, l-1) / comb(k+n-1, n-1)

def sumlog(arr):
    return np.sum(np.log(arr))

def logP(n,k,l):
    return sumlog(np.arange(n-l+1, n+1, dtype=np.int32)) - \
            sumlog(np.arange(1, l+1, dtype=np.int32)) + \
            sumlog(np.arange(k-l+1, k, dtype=np.int32)) - \
            sumlog(np.arange(1, l, dtype=np.int32)) - \
            sumlog(np.arange(k+1, k+n, dtype=np.int32)) + \
            sumlog(np.arange(1, n, dtype=np.int32))


P2_vec = np.vectorize(P)
lP2_vec = np.vectorize(logP)

k=10
l=6
ns = np.linspace(6,100,95, dtype=np.int32)
```

```
y = P2_vec(ns, k, l);
plt.plot(ns, y);
plt.axvline(13);
```




```
k=10
l=6
ns = np.linspace(6,100,95, dtype=np.int32)

y = lP2_vec(ns, k, l);
plt.plot(ns, y);
plt.axvline(13);
```




```
floor(minimize_scalar(lambda xn: -P(int(xn), k, l), bounds=(10,20),
method='bounded').x)
```

13

```
floor(minimize_scalar(lambda xn: -logP(int(xn), k, l), bounds=(10,20),
method='bounded').x)
```

13

Ответ: n=13

[5] Постройте график математического ожидания числа разных имён у 10 таксистов, как функции от общего количества имён n. Найдите оценку числа n методом моментов.

$$P(k,l,n)=\frac{C_n^l * C_{k-1}^{l-1}}{C_{k+n-1}^{n-1}}$$

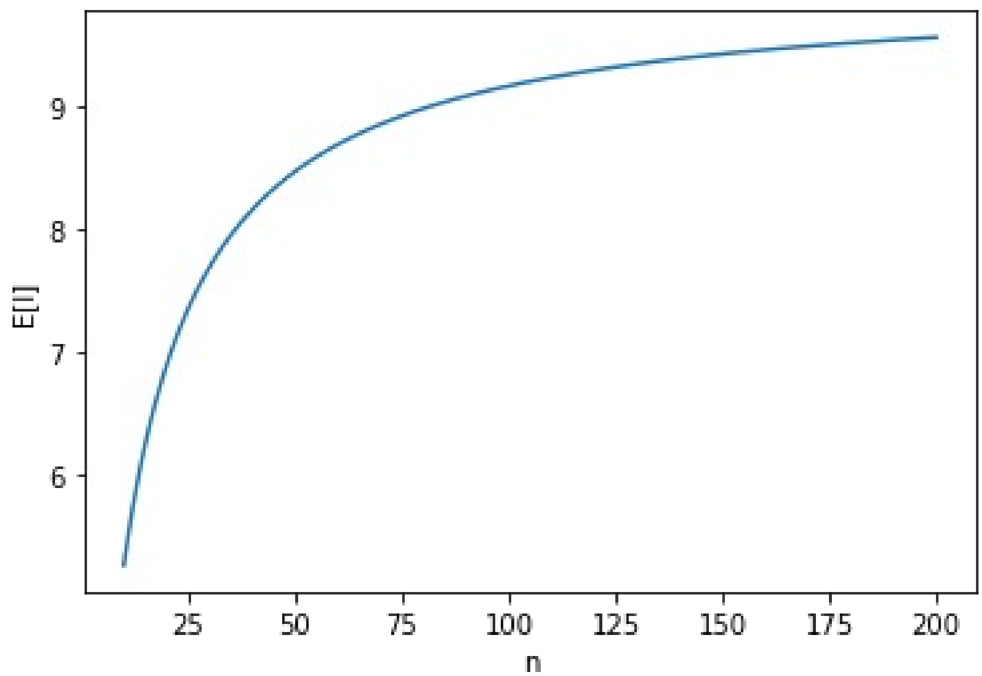
$$C_n^l=\frac{(n-l)\dots n}{l!}$$

$$C_{k-1}^{l-1}=\frac{(k-l+1)\dots(k-1)}{(l-1)!}$$

$$C_{k+n-1}^{n-1}=\frac{(k+1)\dots(k+n-1)}{(n-1)!}$$

```
def En(n):
    r = 0
```

```
    for l in range(1,11):
        P0 = P(n, 10, l)
        r += l*P0
    return r

En_vec = np.vectorize(En)

ns = np.linspace(10,200,191, dtype=np.int32)

y = En_vec(ns)

plt.plot(ns,y);
plt.ylabel('E[l]');
plt.xlabel('n');
```




```
root_scalar(lambda xn: En(int(xn))-6, bracket=(10,25))
```

```
      converged: True
           flag: 'converged'
 function_calls: 43
     iterations: 42
           root: 14.00000000000119
```

Ответ: n=14

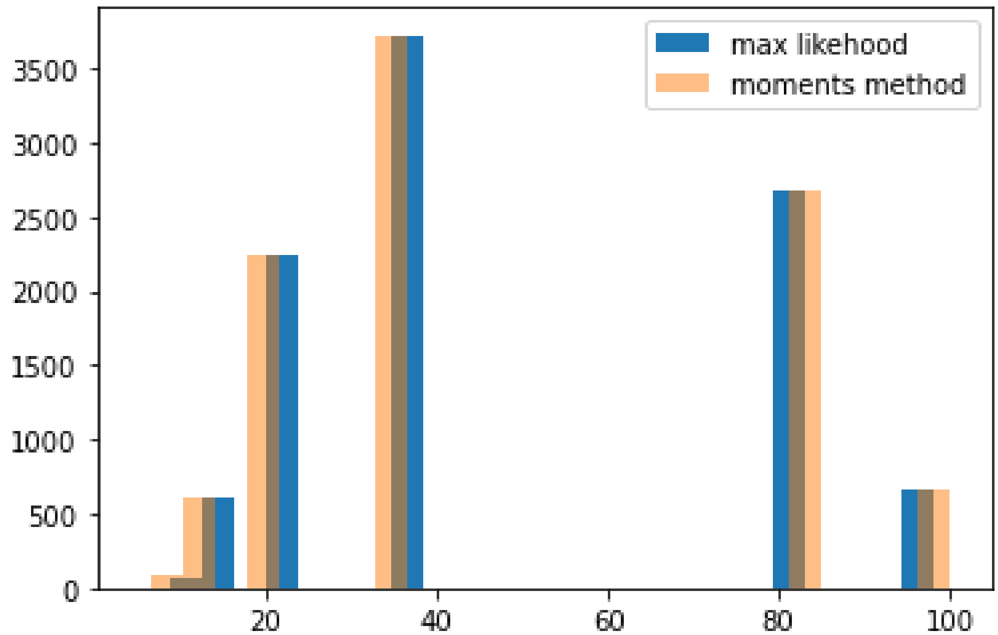
[15] Предположим, что настоящее n равно 20. Проведя 10000 симуляций десяти вызовов такси, рассчитайте 10000 оценок методом моментов и 10000 оценок методом максимального правдоподобия. Постройте гистограммы для оценок двух методов. Оцените смещение, дисперсию и

среднеквадратичную ошибку двух методов. Update 2023-06-07: если по выборке в симуляциях оценка метода моментов или метода максимального правдоподобия стремится к бесконечности и, строго говоря, не существует, то можно ограничить её сверху большим числом, например, 100.

```
def simulate(n):
    hist = np.zeros(n, dtype=np.int32)
    for i in range(10):
        x = np.random.randint(0,n)
        hist[x] += 1
    return np.sum(hist > 0)

np.random.seed(1234)
n = 20
k = 10000
res = np.zeros(k, dtype=np.int32)
mmp = np.zeros(k)
mm = np.zeros(k)
for i in trange(k):
    res[i] = int(simulate(n)) #эксперимент
    try:
        resMMP = minimize_scalar(lambda xn: -P(int(xn), 10, res[i]),
bounds=(res[i],100), method='bounded') #метод макс правдоподобия
        resMMP = floor(resMMP.x)
    except ValueError as ve:
        resMMP = 100
    mmp[i] = resMMP
    try:
        resMM = root_scalar(lambda xn: En(int(xn))-res[i],
bracket=(res[i],100)).root
    except ValueError as ve:
        resMM = 100
    mm[i] = resMM
```

{"model_id":"ed03af25ce184585b92e0ce62eae3ed4","version_major":2,"version_minor":0}

```
plt.hist(mmp, bins=25, label='max likehood')
plt.hist(mm, bins=25, label='moments method', alpha=0.5)
plt.legend();
```




```
print('Метод макс. правдоподобия:')
print(f'\tСмещение: {mmp.mean() - 20}')
print(f'\tДисперсия: {mmp.var()}')
print(f'\tMSE error: {((mmp-20)**2).mean()}')

print('Метод моментов:')
print(f'\tСмещение: {mm.mean() - 20}')
print(f'\tДисперсия: {mm.var()}')
print(f'\tMSE error: {((mm-20)**2).mean()}')
```

```
Метод макс. правдоподобия:
	Смещение: 26.363300000000002
	Дисперсия: 764.3911131100001
	MSE error: 1459.4147
Метод моментов:
	Смещение: 27.77608282251608
	Дисперсия: 779.6932332059512
	MSE error: 1551.2040101692244
```

Задача 3

Иноагент Иннокентий по 20 наблюдениям строит 95%-й доверительный интервал для математического ожидания несколькими способами: классический асимптотический нормальный интервал, с помощью наивного бутстрэпа, с помощью бутстрэпа t-статистики.

[15] Для каждого способа с помощью 10000 симуляций оцените вероятность того, что номинально 95%-й доверительный интервал

фактически накрывает математическое ожидание, если наблюдения распределены экспоненциально с интенсивностью 1.

```
def classic(arr):
    N = arr.shape[0]
    m = arr.mean()
    SE = arr.std() / np.sqrt(N)
    return (m - 1.96 * SE, m + 1.96 * SE)

def naive_bootstrap(arr):
    N = 1000
    m = np.zeros(N)
    j = arr.shape[0]
    for i in range(N):
        arr_j = np.random.choice(arr, j)
        m[i] = arr_j.mean()
    return (np.quantile(m, 0.025), np.quantile(m, 0.975))

def t_bootstrap(arr):
    L = arr.shape[0]
    m = arr.mean()
    SE = arr.std() / np.sqrt(L)
    N = 1000
    t = np.zeros(N)
    j = arr.shape[0]
    for i in range(N):
        arr_j = np.random.choice(arr, j)
        t[i] = (arr_j.mean() - m) / (arr_j.std() / np.sqrt(j))
    return (m + np.quantile(t, 0.025) * SE, m + np.quantile(t, 0.975)
* SE)
```

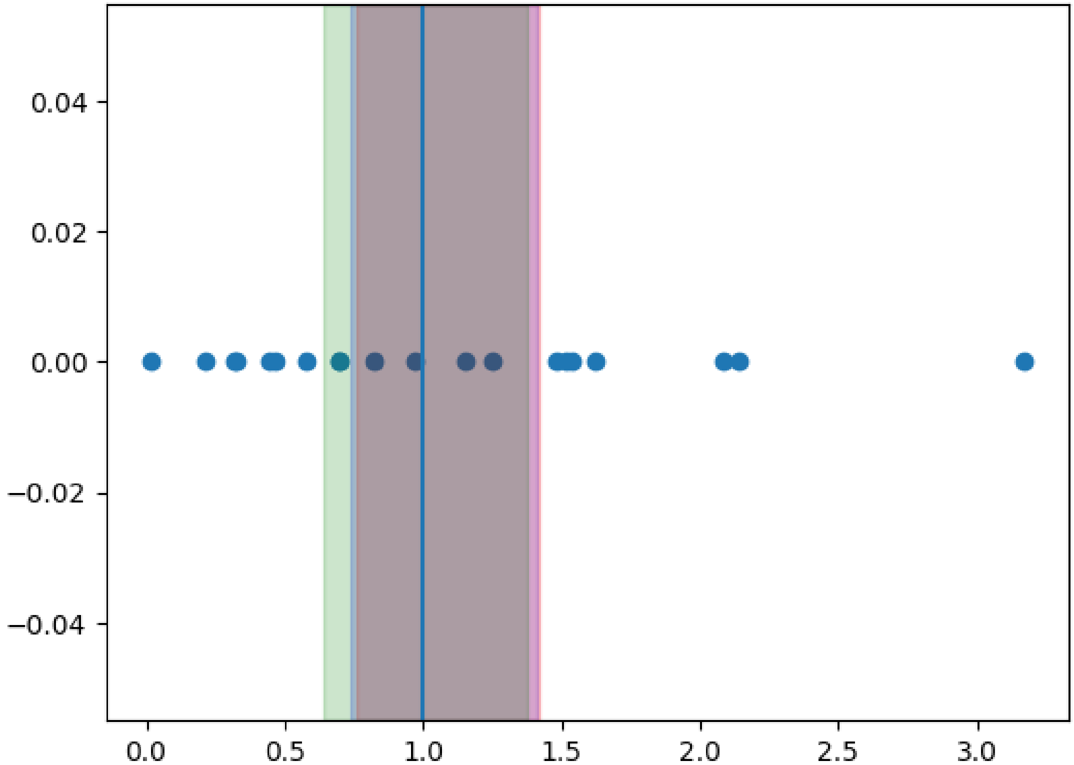
Проверка

```
np.random.seed(1234)
arr = np.random.exponential(size=20)
true_E = 1

res_c = classic(arr)
res_nb = naive_bootstrap(arr)
res_tb = t_bootstrap(arr)

print(res_c, res_nb, res_tb)
```

```
(0.7372248919323703, 1.4122594782884972) (0.7614281757111883,
1.4164114386083948) (0.6442988968106178, 1.376029272687912)
```

```
plt.scatter(arr, [0] * 20)
plt.axvline(true_E)
plt.axvspan(*res_c, alpha=0.2, color='b')
plt.axvspan(*res_nb, alpha=0.2, color='r')
plt.axvspan(*res_tb, alpha=0.2, color='g')
```

```
<matplotlib.patches.Polygon at 0x7f46e83c6ce0>
```



Симуляция

```
def isbetween(x, a):
    return a[0] < x < a[1]

def simulate(gen):
    arr = gen(20)
    true_E = 1
    res_c = classic(arr)
    res_c = min(res_c), max(res_c)
    res_nb = naive_bootstrap(arr)
    res_nb = min(res_nb), max(res_nb)
    res_tb = t_bootstrap(arr)
    res_tb = min(res_tb), max(res_tb)
    return isbetween(1, res_c), isbetween(1, res_nb), isbetween(1,
res_tb)

def big_simulate(N, gen):
    c_stat = np.zeros(N, dtype=np.bool_)
    nb_stat = np.zeros(N, dtype=np.bool_)
    tb_stat = np.zeros(N, dtype=np.bool_)
    for i in trange(N):
        c_stat[i], nb_stat[i], tb_stat[i] = simulate(gen)
    return c_stat.mean(), nb_stat.mean(), tb_stat.mean()
```

```
np.random.seed(1234)
big_simulate(10000, lambda n: np.random.exponential(scale=1, size=n))
```

{"model_id":"9cb71eeb824740fabbe61fa393b2577e","version_major":2,"version_minor":0}

```
(0.8916, 0.8955, 0.8841)
```

Результаты незначительно отличаются в пределах одного процента, t-bootstrap показался результат слабее остальных

```
np.random.seed(1234)
big_simulate(10000, lambda n: np.random.standard_t(df=3, size=n))
```

{"model_id":"93abb1c058d849b8a6d9c03939d22d4e","version_major":2,"version_minor":0}

```
(0.1802, 0.1838, 0.242)
```

Метод t-bootstrap показал себя сильно лучше остальных методов в случае t-распределения. Очевидно, что использование квантилей исследуемого распределения улучшает точность покрытия интервала bootstrap.

Задача 4

Проверьте гипотезу о том, что ожидаемые результаты экзамена по теории вероятностей тех, у кого фамилия начинается с гласной буквы и с согласной буквы, равны. В качестве альтернативной гипотезы возьмите гипотезу о неравенстве.

Достанем выборку студентов, пришедших на экзамен и разделим их на студентов с фамилией на гласную (класс 1) и согласную (класс 0)

```
data = pd.read_csv('Exam.csv', skiprows=5)
data = data[['Unnamed: 1', 'Unnamed: 74']].rename(columns={'Unnamed: 1':'Name', 'Unnamed: 74': 'Mark'})
data
```

```
            Name    Mark
0      Репенкова      16
1      Ролдугина  неявка
2         Сафина      19
3        Сидоров      26
4       Солоухин      21
..           ...     ...
327     Сенников      19
328           Ся  неявка
329       Сятова  неявка
330   Темиркулов  неявка
```

```
331      Эшмеев      16

[332 rows x 2 columns]
```

```
data.Mark.unique()
```

```
array(['16', 'неявка', '19', '26', '21', '22', '20', '17', '27', '29',
       '23', '24', '25', '10', '8', '15', '18', '28', '12', '13',
'11',
       '14', '4', '9', '5', '6', '7', '30'], dtype=object)
```

```
data = data[data['Mark'] != 'неявка']
data['Mark'] = data['Mark'].apply(int)
```

```
<ipython-input-5-79f9a5a13bc5>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  data['Mark'] = data['Mark'].apply(int)
```

```
def classify(s):
    vowel = 'УЕЁЫАОЭЯИЮ'
    s0 = s[0]
    return int(s0 in vowel)
```

```
data['class'] = data['Name'].apply(classify)
```

```
<ipython-input-6-55ecd944202c>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  data['class'] = data['Name'].apply(classify)
```

```
data[data['class'] == 1]
```

	Name	Mark	class
17	Адилхан	25	1
18	Алексанян	26	1
32	Охотин	25	1
45	Аврамчук	29	1
46	Авсеенко	26	1
47	Адамокова	20	1
48	Адамцева	19	1
49	Азаров	24	1
50	Алексеева	25	1

```
51       Афанасьев    28       1
67          Иванов    16       1
68         Иванова    16       1
85         Осенева    15       1
113  Абдугаффорова    17       1
114        Амреева     4       1
126       Ермишова    23       1
143       Овчарова    21       1
144    Осиновскова    13       1
158         Ускова    16       1
166          Ягжов    19       1
167       Яковлева     6       1
168             Ян    15       1
169      Янковская     8       1
170       Агамалов    13       1
171         Акимов    23       1
172     Амбросимов    11       1
173     Асаналиева     7       1
174       Асонкова    12       1
184      Есауленко    20       1
188         Иванов    20       1
189       Исмаилов    17       1
220        Уначева    18       1
221        Ушатова    20       1
227       Яковлева    17       1
228         Ямкова    11       1
229       Адмайкин    21       1
230         Алиева     5       1
240        Ермошин    13       1
270         Янышен    13       1
271        Яхьяева    13       1
272      Абдулаева    21       1
300        Ермаков    22       1
331         Эшмеев    16       1
```

[5] Используйте тест Уэлча.

```
data[data['class'] == 0]['Mark'].var(), data[data['class'] == 1]
['Mark'].var()
```

```
(33.72049689440993, 39.39202657807308)
```

Дисперсии выборок разные, учитываем это в аргументах теста

```
from scipy.stats import ttest_ind
a = data[data['class'] == 0]['Mark']
b = data[data['class'] == 1]['Mark']
ttest_ind(a, b, equal_var=False)
```

```
Ttest_indResult(statistic=0.8646536949457263,
pvalue=0.3909901940797269)
```

Критерий не отвергает нулевую гипотезу

[5] Используйте наивный бутстрэп.

```
def nb_hypothesis(arr1, arr2):
    N = 1000
    m = np.zeros(N)
    j1, j2 = arr1.shape[0], arr2.shape[0]
    for i in range(N):
        arr1_j = np.random.choice(arr1, j1)
        arr2_j = np.random.choice(arr2, j2)
        m[i] = arr1_j.mean() - arr2_j.mean()
    return (np.quantile(m, 0.025), np.quantile(m, 0.975))

np.random.seed(1234)
res = nb_hypothesis(a,b)
res
```

```
(-0.9651875172350391, 2.7861223458038404)
```

Ноль входит в доверительный интервал разниц между средними повторных подвыборок наивного бутстрапа => их различие статистически не значимо

[5] Используйте бутстрэп t-статистики.

```
def tb_hypothesis(arr1, arr2):
    m = arr1.mean() - arr2.mean()
    SE = np.sqrt(arr1.var() / arr1.shape[0] + arr2.var() / 
arr2.shape[0])
    N = 1000
    t = np.zeros(N)
    j1, j2 = arr1.shape[0], arr2.shape[0]
    for i in range(N):
        arr1_j = np.random.choice(arr1, j1)
        arr2_j = np.random.choice(arr2, j2)
        t[i] = (arr1_j.mean() - arr2_j.mean()- m) / \
          np.sqrt(arr1_j.var() / arr1_j.shape[0] + arr2_j.var() / 
arr2_j.shape[0])
    return (m + np.quantile(t, 0.025) * SE, m + np.quantile(t, 0.975) 
* SE)

np.random.seed(1234)
res = tb_hypothesis(a,b)
res
```

```
(-1.2126137791705514, 2.8960508732029213)
```

Ноль входит в доверительный интервал бутстрапа t-статистики => их различие статистически не значимо

Задача 8

Самым полезным ресурсом для подготовки к контрольным был youtube-канал Math Meth, а именно записи семинаров Пильника Н.П. Очень помогло разобраться с темами, которые с первого раза оказались непонятны и была необходиомсть пересмотреть семинар.