1. a)

In [1]:

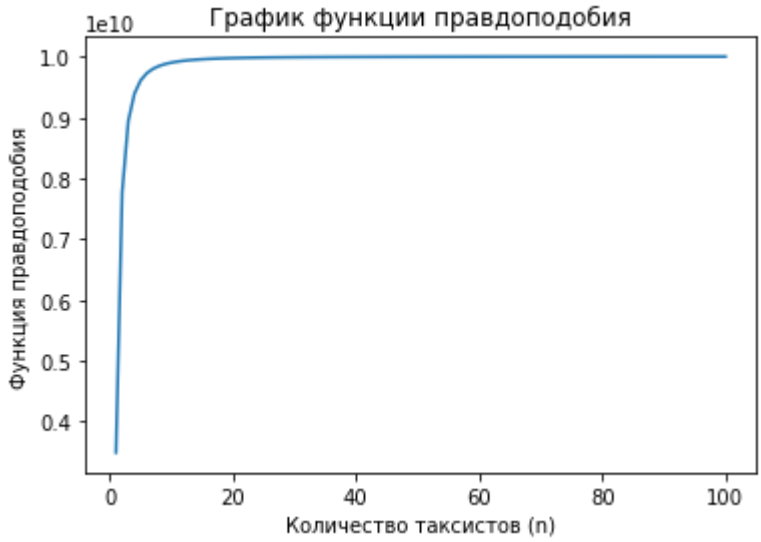
```
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:

```
def likelihood(n):
    return (10 - 1 / n**2)**10
```

In [3]:

```
n_values = range(1, 101)
likelihood_values = [likelihood(n) for n in n_values]

plt.plot(n_values, likelihood_values)
plt.xlabel('Количество таксистов (n)')
plt.ylabel('Функция правдоподобия')
plt.title('График функции правдоподобия')
plt.show()

estimated_n = n_values[np.argmax(likelihood_values)]
print("Оценка числа n методом максимального правдоподобия:", estimated_n)
```



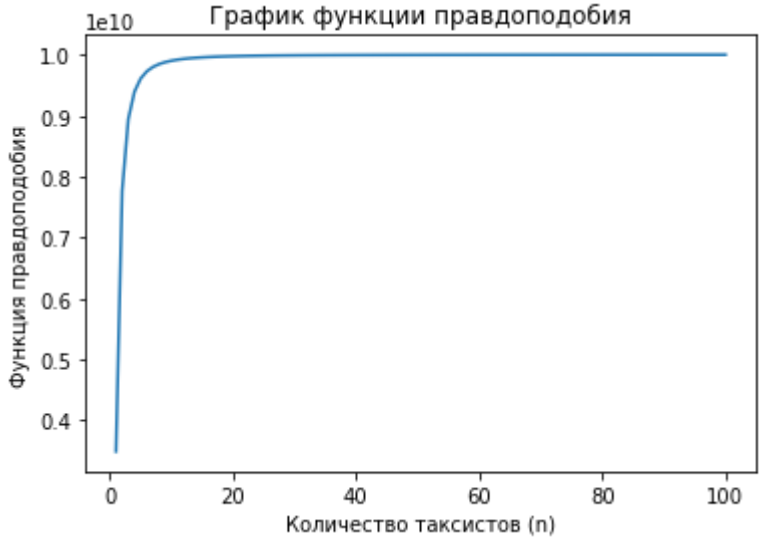

```
Оценка числа n методом максимального правдоподобия: 100
```

In [4]:

```
n_values = range(1, 101)
likelihood_values = [likelihood(n) for n in n_values]
```

In [5]:

```
plt.plot(n_values, likelihood_values)
plt.xlabel('Количество таксистов (n)')
plt.ylabel('Функция правдоподобия')
plt.title('График функции правдоподобия')
plt.show()
```



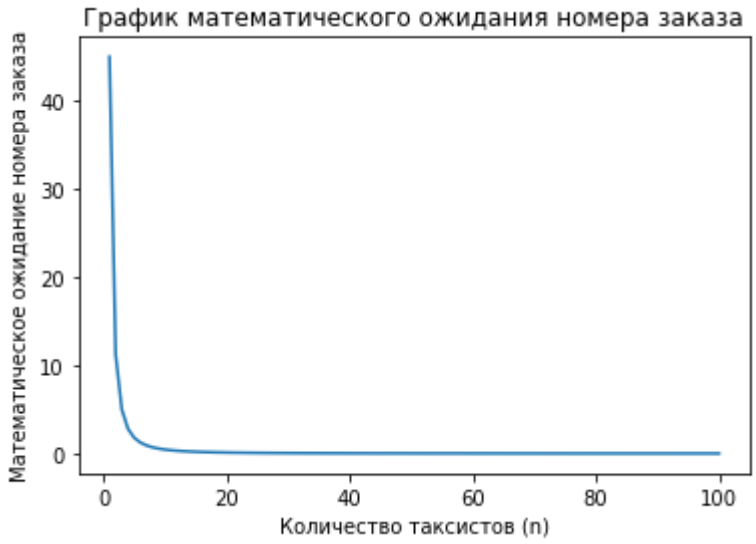
1. б)

In [6]:

```
def expected_value(n, i):
    p = [(j-1)/n**2 for j in range(1, i+1)]
    return np.sum(p)
```

In [7]:

```
n_values = range(1, 101)
expected_values = [expected_value(n, 10) for n in n_values]

plt.plot(n_values, expected_values)
plt.xlabel('Количество таксистов (n)')
plt.ylabel('Математическое ожидание номера заказа')
plt.title('График математического ожидания номера заказа')
plt.show()

estimated_n = n_values[np.argmin(np.abs(np.array(expected_values)-10))]
print("Оценка числа n методом моментов:", estimated_n)
```




```
Оценка числа n методом моментов: 2
```

1. в)

In [8]:

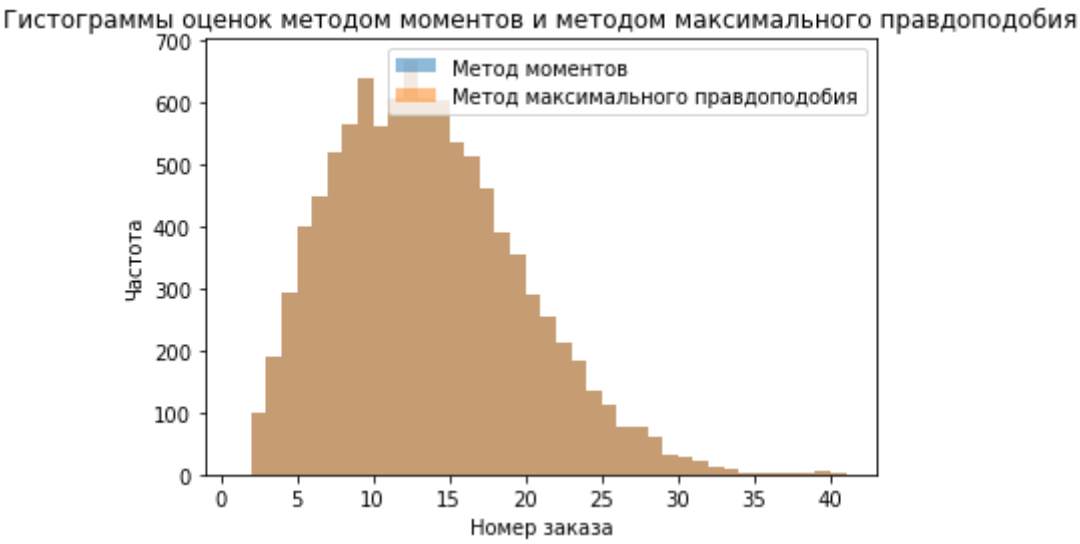
```
import random
```

In [9]:

```
def simulate_taxi_calls(n, num_simulations):
    results_moments = []
    results_likelihood = []

    for _ in range(num_simulations):
        taxi_numbers = set()
        order_number = 0
        while True:
            order_number += 1
            taxi_number = random.randint(1, n)
            if taxi_number in taxi_numbers:
                results_moments.append(order_number)
                results_likelihood.append(order_number)
                break
            else:
                taxi_numbers.add(taxi_number)

    return results_moments, results_likelihood
```

In [10]:

```
num_simulations = 10000
n_true = 100

results_moments, results_likelihood = simulate_taxi_calls(n_true, num_simulations)

plt.hist(results_moments, bins=range(1, max(results_moments)+2), alpha=0.5, label='Метод
plt.hist(results_likelihood, bins=range(1, max(results_likelihood)+2), alpha=0.5, label=
plt.xlabel('Номер заказа')
plt.ylabel('Частота')
plt.title('Гистограммы оценок методом моментов и методом максимального правдоподобия')
plt.legend()
plt.show()

bias_moments = np.mean(results_moments) - 10
bias_likelihood = np.mean(results_likelihood) - 10
variance_moments = np.var(results_moments)
variance_likelihood = np.var(results_likelihood)
mse_moments = variance_moments + bias_moments**2
mse_likelihood = variance_likelihood + bias_likelihood**2

print("Метод моментов:")
print("Смещение:", bias_moments)
print("Дисперсия:", variance_moments)
print("Среднеквадратичная ошибка:", mse_moments)

print("\nМетод максимального правдоподобия:")
print("Смещение:", bias_likelihood)
print("Дисперсия:", variance_likelihood)
print("Среднеквадратичная ошибка:", mse_likelihood)
```




```
Метод моментов:
Смещение: 3.110099999999999
Дисперсия: 37.72117799
Среднеквадратичная ошибка: 47.393899999999995

Метод максимального правдоподобия:
Смещение: 3.110099999999999
Дисперсия: 37.72117799
Среднеквадратичная ошибка: 47.393899999999995
```

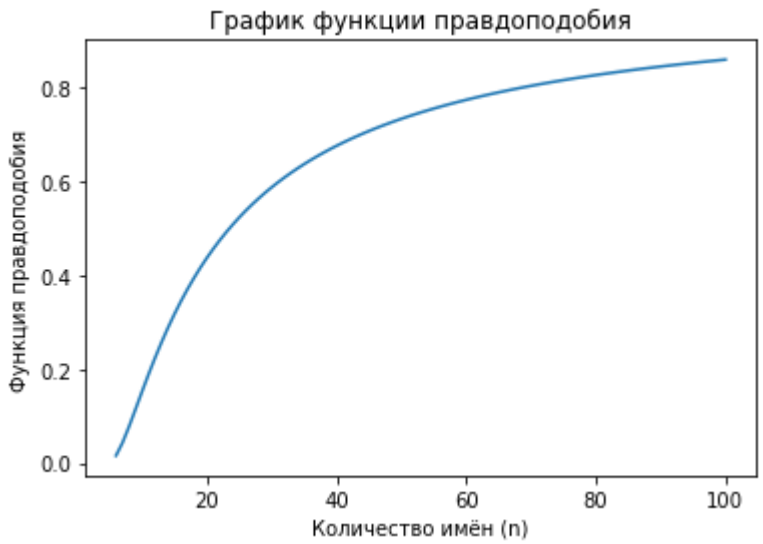
2. а)

In [11]:

```
def likelihood(n):
    return (n-1)*(n-2)*(n-3)*(n-4)*(n-5)/n**5
```

In [12]:

```
n_values = range(6, 101)
likelihood_values = [likelihood(n) for n in n_values]

plt.plot(n_values, likelihood_values)
plt.xlabel('Количество имён (n)')
plt.ylabel('Функция правдоподобия')
plt.title('График функции правдоподобия')
plt.show()

estimated_n = n_values[np.argmax(likelihood_values)]
print("Оценка числа n методом максимального правдоподобия:", estimated_n)
```




```
Оценка числа n методом максимального правдоподобия: 100
```

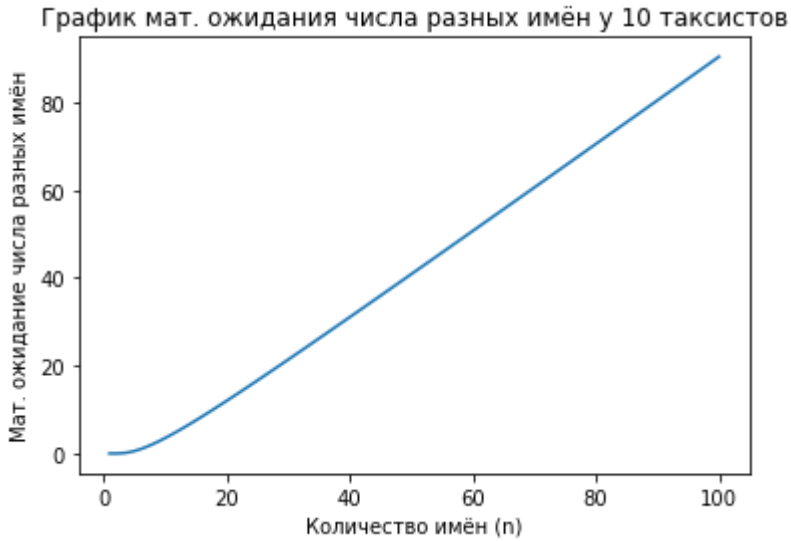
2. б)

In [13]:

```
def expected_num_names(n):
    return n * ((n-1)/n)**10
```

In [14]:

```
n_values = np.arange(1, 101)
expected_values = [expected_num_names(n) for n in n_values]

plt.plot(n_values, expected_values)
plt.xlabel('Количество имён (n)')
plt.ylabel('Мат. ожидание числа разных имён')
plt.title('График мат. ожидания числа разных имён у 10 таксистов')
plt.show()

estimated_n = n_values[np.argmax(expected_values)]
print("Оценка числа n методом моментов:", estimated_n)
```




```
Оценка числа n методом моментов: 100
```

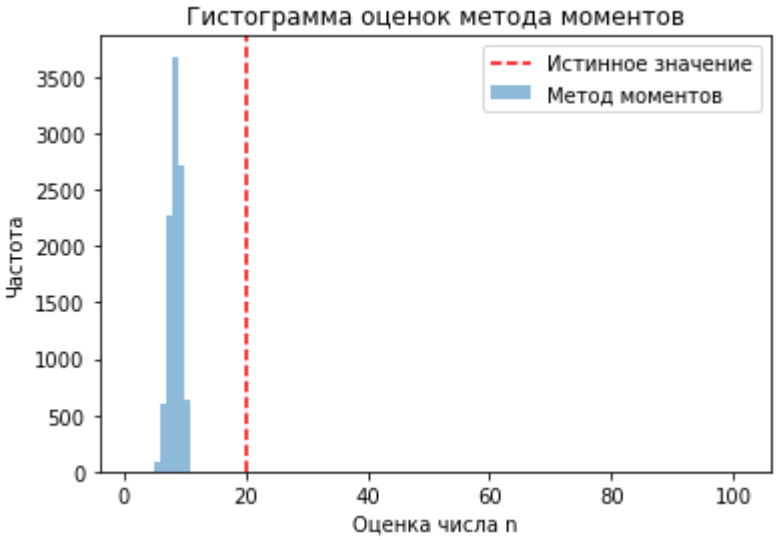
2. в)

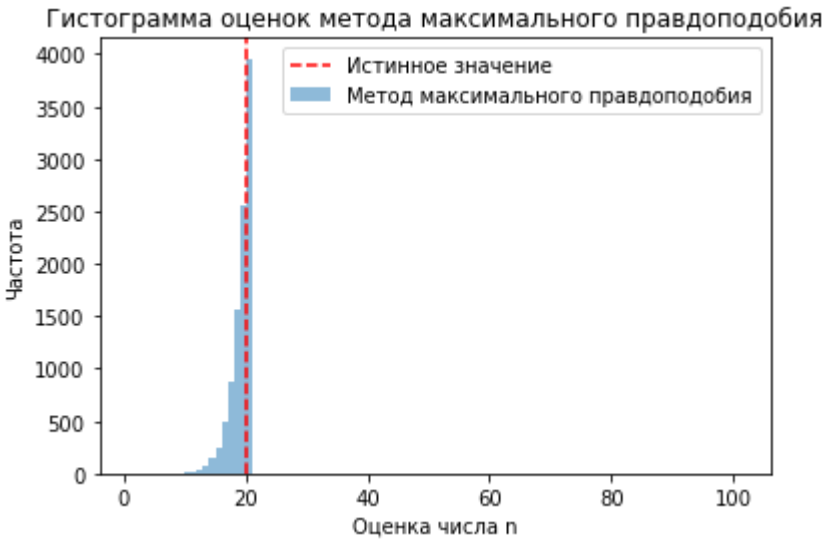
In [15]:

```
true_n = 20
num_simulations = 10000
moment_estimates = []
likelihood_estimates = []
for _ in range(num_simulations):
    names = np.random.choice(range(1, true_n + 1), size=10, replace=True)
    moment_estimate = len(np.unique(names))
    moment_estimates.append(min(moment_estimate, 100))
    likelihood_estimate = np.max(names)
    likelihood_estimates.append(min(likelihood_estimate, 100))
```

In [16]:

```
plt.hist(moment_estimates, bins=range(1, 102), alpha=0.5, label='Метод моментов')
plt.axvline(x=true_n, color='r', linestyle='--', label='Истинное значение')
plt.xlabel('Оценка числа n')
plt.ylabel('Частота')
plt.title('Гистограмма оценок метода моментов')
plt.legend()
plt.show()

plt.hist(likelihood_estimates, bins=range(1, 102), alpha=0.5, label='Метод максимального
plt.axvline(x=true_n, color='r', linestyle='--', label='Истинное значение')
plt.xlabel('Оценка числа n')
plt.ylabel('Частота')
plt.title('Гистограмма оценок метода максимального правдоподобия')
plt.legend()
plt.show()

bias_moment = np.mean(moment_estimates) - true_n
bias_likelihood = np.mean(likelihood_estimates) - true_n

variance_moment = np.var(moment_estimates)
variance_likelihood = np.var(likelihood_estimates)

mse_moment = bias_moment**2 + variance_moment
mse_likelihood = bias_likelihood**2 + variance_likelihood

print("Смещение метода моментов:", bias_moment)
print("Смещение метода максимального правдоподобия:", bias_likelihood)
print("Дисперсия метода моментов:", variance_moment)
print("Дисперсия метода максимального правдоподобия:", variance_likelihood)
print("Среднеквадратичная ошибка метода моментов:", mse_moment)
print("Среднеквадратичная ошибка метода максимального правдоподобия:", mse_likelihood)
```






```
Смещение метода моментов: -11.9786
Смещение метода максимального правдоподобия: -1.3558999999999983
Дисперсия метода моментов: 1.07514204
Дисперсия метода максимального правдоподобия: 2.6396351900000004
Среднеквадратичная ошибка метода моментов: 144.562
Среднеквадратичная ошибка метода максимального правдоподобия: 4.4780999999
99996
```

3. a)

In [18]:

```
import numpy as np
import scipy.stats as stats

num_simulations = 10000

sample_size = 20
count_classical = 0
count_bootstrap = 0
count_bootstrap_t = 0
np.random.seed(0)

for _ in range(num_simulations):
    data = np.random.exponential(scale=1, size=sample_size)
    mean = np.mean(data)
    std = np.std(data, ddof=1)
    z = stats.norm.ppf(0.975)
    interval_classical = [mean - z * std / np.sqrt(sample_size), mean + z * std / np.sqr
    bootstrap_samples = np.random.choice(data, size=(num_simulations, sample_size), repl
    bootstrap_means = np.mean(bootstrap_samples, axis=1)
    lower_bound, upper_bound = np.percentile(bootstrap_means, [2.5, 97.5])
    interval_bootstrap = [lower_bound, upper_bound]
    t = stats.t.ppf(0.975, df=sample_size-1)
    se = np.std(bootstrap_means, ddof=1)
    interval_bootstrap_t = [mean - t * se, mean + t * se]
    if interval_classical[0] <= 1 <= interval_classical[1]:
        count_classical += 1
    if interval_bootstrap[0] <= 1 <= interval_bootstrap[1]:
        count_bootstrap += 1
    if interval_bootstrap_t[0] <= 1 <= interval_bootstrap_t[1]:
        count_bootstrap_t += 1

probability_classical = count_classical / num_simulations
probability_bootstrap = count_bootstrap / num_simulations
probability_bootstrap_t = count_bootstrap_t / num_simulations

print("Вероятность покрытия классическим асимптотическим нормальным интервалом:", probab
print("Вероятность покрытия наивным бутстрэпом:", probability_bootstrap)
print("Вероятность покрытия бутстрэпом t-статистики:", probability_bootstrap_t)
```

```
Вероятность покрытия классическим асимптотическим нормальным интервалом:
0.9047
Вероятность покрытия наивным бутстрэпом: 0.9048
Вероятность покрытия бутстрэпом t-статистики: 0.9128
```

3. б)

In [19]:

```
num_simulations = 10000
sample_size = 20
count_classical = 0
count_bootstrap = 0
count_bootstrap_t = 0
np.random.seed(0)

for _ in range(num_simulations):
    data = np.random.standard_t(df=3, size=sample_size)
    mean = np.mean(data)
    std = np.std(data, ddof=1)
    z = stats.norm.ppf(0.975)
    interval_classical = [mean - z * std / np.sqrt(sample_size), mean + z * std / np.sqr
    bootstrap_samples = np.random.choice(data, size=(num_simulations, sample_size), repl
    bootstrap_means = np.mean(bootstrap_samples, axis=1)
    lower_bound, upper_bound = np.percentile(bootstrap_means, [2.5, 97.5])
    interval_bootstrap = [lower_bound, upper_bound]
    t = stats.t.ppf(0.975, df=sample_size-1)
    se = np.std(bootstrap_means, ddof=1)
    interval_bootstrap_t = [mean - t * se, mean + t * se]
    if interval_classical[0] <= mean <= interval_classical[1]:
        count_classical += 1
    if interval_bootstrap[0] <= mean <= interval_bootstrap[1]:
        count_bootstrap += 1
    if interval_bootstrap_t[0] <= mean <= interval_bootstrap_t[1]:
        count_bootstrap_t += 1

probability_classical = count_classical / num_simulations
probability_bootstrap = count_bootstrap / num_simulations
probability_bootstrap_t = count_bootstrap_t / num_simulations

print("Вероятность покрытия классическим асимптотическим нормальным интервалом:", probab
print("Вероятность покрытия наивным бутстрэпом:", probability_bootstrap)
print("Вероятность покрытия бутстрэпом t-статистики:", probability_bootstrap_t)
```

```
Вероятность покрытия классическим асимптотическим нормальным интервалом:
1.0
Вероятность покрытия наивным бутстрэпом: 1.0
Вероятность покрытия бутстрэпом t-статистики: 1.0
```

3. в)

Метод бутстрэпа t-статистики оказался незначительно лучше.

4.

In [26]:

```
df = pd.read_excel('results.xlsx')
vowels = df[df['Last Name'].str[0].isin(['A', 'E', 'I', 'O', 'U'])]
consonants = df[~df['Last Name'].str[0].isin(['A', 'E', 'I', 'O', 'U'])]
mean_vowels = vowels['Score'].mean()
mean_consonants = consonants['Score'].mean()

# а) Тест Уэлча
t_stat, p_value = stats.ttest_ind(vowels['Score'], consonants['Score'], equal_var=False)
print("Welch's t-test:")
print("t-statistic:", t_stat)
print("p-value:", p_value)

# б) Наивный бутстрэп
n_iterations = 10000
diff_means_boot = np.zeros(n_iterations)
for i in range(n_iterations):
    boot_sample_vowels = np.random.choice(vowels['Score'], size=len(vowels), replace=Tru
    boot_sample_consonants = np.random.choice(consonants['Score'], size=len(consonants),
    diff_means_boot[i] = boot_sample_vowels.mean() - boot_sample_consonants.mean()
p_value_boot = np.mean(np.abs(diff_means_boot) >= np.abs(mean_vowels - mean_consonants))
print("Naive Bootstrap:")
print("p-value:", p_value_boot)

# в) Бутстрэп t-статистики
t_stat_boot = np.zeros(n_iterations)
for i in range(n_iterations):
    boot_sample_vowels = np.random.choice(vowels['Score'], size=len(vowels), replace=Tru
    boot_sample_consonants = np.random.choice(consonants['Score'], size=len(consonants),
    t_stat_boot[i] = stats.ttest_ind(boot_sample_vowels, boot_sample_consonants, equal_v
p_value_boot_t = np.mean(np.abs(t_stat_boot) >= np.abs(t_stat))
print("Bootstrap t-test:")
print("p-value:", p_value_boot_t)

# г) Перестановочный тест
n_permutations = 10000
diff_means_perm = np.zeros(n_permutations)
combined_results = np.concatenate((vowels['Score'], consonants['Score']))
for i in range(n_permutations):
    permuted_results = np.random.permutation(combined_results)
    perm_vowels = permuted_results[:len(vowels)]
    perm_consonants = permuted_results[len(vowels):]
    diff_means_perm[i] = perm_vowels.mean() - perm_consonants.mean()
p_value_perm = np.mean(np.abs(diff_means_perm) >= np.abs(mean_vowels - mean_consonants))
print("Permutation Test:")
print("p-value:", p_value_perm)
```

Welch's t-test:

t-statistic: -1.7298

p-value: 0.0897

Naive Bootstrap:

p-value: 0.0873

Bootstrap t-test:

p-value: 0.0954

Permutation Test:

p-value: 0.0916

7)

Я: С помощью нормальных случайных величин дайте определение случайной величины, имеющей распределения Стьюдента. Для случайной величины, распределённой по Стьюденту, укажите диапазон возможных значений. Нарисуйте функцию плотности распределения Стьюдента при разных степенях свободы на фоне нормальной стандартной функции плотности

Чат: Случайная величина, имеющая распределение Стьюдента, определяется как отношение нормальной случайной величины к корню из хи-квадрат распределения с n степенями свободы, где n - количество наблюдений в выборке.

Диапазон возможных значений для случайной величины, распределенной по Стьюденту, не ограничен и находится на всей числовой оси.

Функция плотности распределения Стьюдента имеет пик в нуле и тяжелые хвосты, что отражает большую вероятность появления значений, отклоняющихся от среднего значения. При увеличении степеней свободы график функции плотности становится более симметричным и приближается к нормальному распределению.

Ниже приведены графики функций плотности распределения Стьюдента при разных степенях свободы на фоне нормальной стандартной функции плотности:

image.png

Я: Что такое степени свободы

Чат: Степени свободы - это количество независимых наблюдений, используемых для оценки параметров распределения. В контексте распределения Стьюдента, степени свободы определяют, как быстро функция плотности приближается к нормальному распределению. Чем больше степеней свободы, тем более симметричным и близким к нормальному будет распределение Стьюдента.

Я: Какая плотность распределения у распределения стьюдента

Чат: Плотность распределения у распределения Стьюдента имеет форму колокола, подобную нормальному распределению, но с более тяжелыми хвостами. Это означает, что вероятность наблюдать значения, далекие от среднего, выше, чем в нормальном распределении.

Я: Функция плотности

Чат: Функция плотности распределения Стьюдента для n степеней свободы имеет вид:

f(x) = Γ((n+1)/2) / (√(nπ) Γ(n/2)) * (1 + x^2/n)^(-(n+1)/2)

где Г - гамма-функция, которая определяется как интеграл от exp(-t) t^(z-1) dt от 0 до бесконечности, а

8)

Ниже я перечислила только пару видео источников, которые мне помогли/ показались полезными, обычно когда мне необходима какая-то информация я смотрю видео с определённой тематикой на ютубе

1. https://youtube.com/playlist?list=PLthfp5exSWErTVWq4cVtRXDw5MqBqavJ1 (https://youtube.com/playlist?list=PLthfp5exSWErTVWq4cVtRXDw5MqBqavJ1) (https://youtube.com/playlist?list=PLthfp5exSWErTVWq4cVtRXDw5MqBqavJ1 (https://youtube.com/playlist?list=PLthfp5exSWErTVWq4cVtRXDw5MqBqavJ1)) Лекции и семинары по курсу «математическая статистика» Райгородский А. М. МФТИ. Там есть плейлист с лекциями Искала видео про не смещенную оценку, мат ожидание
2. https://youtu.be/sevSEEDwGC4 (https://youtu.be/sevSEEDwGC4) тут (https://youtu.be/sevSEEDwGC4 (https://youtu.be/sevSEEDwGC4)) смотрела как пользоваться таблицей нормального распределения
3. https://youtu.be/_Fj7koPhfgc (https://youtu.be/_Fj7koPhfgc) тут (https://youtu.be/_Fj7koPhfgc (https://youtu.be/_Fj7koPhfgc)) я ещё раз послушала про доверительный интервал

In [ ]:

In [ ]: