In [2]:
```
# Initialize Otter
import otter
grader = otter.Notebook("PW2.ipynb")
```

--- assignment_name: PW2 requirements: requirements.txt solutions_pdf: true generate: pdf_via_html: true zips: true ---

In [3]:
```
%pip show otter-grader
```

```
Name: otter-grader
Version: 6.1.6
Summary: A Python and R autograding solution
Home-page:
Author: Christopher Pyles
Author-email: cpyles@berkeley.edu
License: BSD-3-Clause
Location: /Users/scobca/PycharmProjects/itmo-math-statistics/.venv/lib/python3.
9/site-packages
Requires: click, dill, fica, ipylab, ipython, ipywidgets, jinja2, jupytext, nbco
nvert, nbformat, pandas, python-on-whales, pyyaml, requests, wrapt
Required-by:
Note: you may need to restart the kernel to use updated packages.
```

Измерение мира: ошибки и аппроксимации в вычислительной математике

Инструкция для студентов

- Выполните все упражнения, помеченные **YOUR CODE HERE** или *ВАШ КОММЕНТАРИЙ ЗДЕСЬ*.
- Убедитесь, что ваш код является оригинальным и не сгенерирован инструментами искусственного интеллекта.
- Прокомментируйте свой код, чтобы объяснить свои рассуждения.
- Отправьте заполненный блокнот в соответствии с приведенными рекомендациями Moodle до конца занятия.

In [4]:
```
%pip install numpy
%pip install matplotlib
```

```
Requirement already satisfied: numpy in ./.venv/lib/python3.9/site-packages (2.
0.2)

[notice] A new release of pip is available: 23.2.1 -> 26.0.1
[notice] To update, run: pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: matplotlib in ./.venv/lib/python3.9/site-packages
(3.9.4)
Requirement already satisfied: contourpy>=1.0.1 in ./.venv/lib/python3.9/site-pa
ckages (from matplotlib) (1.3.0)
Requirement already satisfied: cycler>=0.10 in ./.venv/lib/python3.9/site-packag
es (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in ./.venv/lib/python3.9/site-p
ackages (from matplotlib) (4.60.2)
Requirement already satisfied: kiwisolver>=1.3.1 in ./.venv/lib/python3.9/site-p
ackages (from matplotlib) (1.4.7)
Requirement already satisfied: numpy>=1.23 in ./.venv/lib/python3.9/site-package
s (from matplotlib) (2.0.2)
Requirement already satisfied: packaging>=20.0 in ./.venv/lib/python3.9/site-pac
kages (from matplotlib) (26.0)
Requirement already satisfied: pillow>=8 in ./.venv/lib/python3.9/site-packages
(from matplotlib) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in ./.venv/lib/python3.9/site-pa
ckages (from matplotlib) (3.3.2)
Requirement already satisfied: python-dateutil>=2.7 in ./.venv/lib/python3.9/sit
e-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: importlib-resources>=3.2.0 in ./.venv/lib/python
3.9/site-packages (from matplotlib) (6.5.2)
Requirement already satisfied: zipp>=3.1.0 in ./.venv/lib/python3.9/site-package
s (from importlib-resources>=3.2.0->matplotlib) (3.23.0)
Requirement already satisfied: six>=1.5 in ./.venv/lib/python3.9/site-packages
(from python-dateutil>=2.7->matplotlib) (1.17.0)

[notice] A new release of pip is available: 23.2.1 -> 26.0.1
[notice] To update, run: pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

In [5]:
```
import numpy as np
```

Содержание

1. Введение

Добро пожаловать в это исследование ошибок и приближений в вычислительной математике, дополненное увлекательной историей измерения мира, рассказанной в книге Кена Алдера "Мера всех вещей (The Measure of All Things.)".

- "После Французской революции два астронома, Пьер Мешен и Жан-Батист Деламбр, предприняли трудную экспедицию для измерения меридиана по всей Франции от Дюнкерка до Барселоны. Их миссия: определить метр как одну десятимиллионную часть расстояния от Северного полюса до экватора, установив универсальную единицу измерения, закрепленную в природе".

Этот поиск был сопряжен с трудностями, которые подчеркивают критическую важность понимания ошибок при измерениях и управления ими — тема, которая сегодня находит глубокий отклик в вычислительной математике.

Астрономы использовали триангуляцию для измерения дуги меридиана. Они измеряли углы между точками и использовали эти измерения для вычисления расстояний. В число используемых ими инструментов входили теодолиты и цепи, которые имели свои ограничения.

Ученые были потрясены разнообразием мер и весов, которые они видели вокруг себя. В XVIII веке меры отличались не только от нации к нации, но и внутри самих наций. Это разнообразие затрудняло коммуникации и торговлю, а также государственное управление. Это также затрудняло ученым сравнивать свои результаты с результатами их коллег. Один англичанин, путешествовавший по Франции накануне революции, обнаружил, что тамошнее разнообразие причиняет ему мучения. “Во Франции, - жаловался он, - бесконечная запутанность принимаемых мер превосходит всякое понимание. Они различаются не только в каждой провинции, но и в каждом округе и почти в каждом городе. . ..”По оценкам современников, под прикрытием примерно восьмисот наименований Древний режим Франции использовал ошеломляющие 250 000 различных единиц веса и мер.

---

2. Понимание ошибок и приближений

2.1 Исторический контекст

- **Проблемы миссии**: Расхождения в измерениях из-за особенностей местности, погоды и ограниченного оборудования.
- **Дилемма Мешена**: Он обнаружил ошибки в своих измерениях, но боролся с их раскрытием, иллюстрируя глубокое влияние, которое ошибки могут оказать на научную целостность и прогресс.

2.2 Типы погрешностей

- **Абсолютная погрешность**: Разница между измеренным или приблизительным значением и истинным значением.
- **Относительная погрешность**: Абсолютная погрешность, деленная на истинное значение, часто выражается в процентах.

Напишите какие ещё типы погрешностей Вы знаете и как они могли бы быть полезны в этой миссии

In [7]:
```
# Напишите ваш ответ между тройными кавычками ниже.
# Это позволит нам автоматически проверить наличие ответа и красиво отобразить его.

t1_ans = '''
Мы обсуждали систематические погрешности, которые возникают, например, из-за неправильной калиб
ровки инструмента измерения, случайные погрешности, возникновение которых мы как правило не мож
ем предсказать, различные инструментальные погрешности, связанные с неточностью самих приборов из
мерения. Также шла речь о приведенной погрешности.
'''

# Записываем в файл для автопроверки
with open("t1.txt", "w", encoding="utf-8") as f:
    f.write(t1_ans)

# Красиво отображаем ответ студента прямо под ячейкой
from IPython.display import display, Markdown
display(Markdown(t1_ans))
```

Мы обсуждали систематические погрешности, которые возникают, например, из-за неправильной калибровки инструмента измерения, случайные погрешности, возникновение которых мы как правило не можем предсказать, различные инструментальные погрешности, связанные с неточностью самих приборов измерения.

In [ ]:
```
grader.check("t1")
```

2.3 Источники ошибок

1. **Человеческие ошибки**: Ошибки при вводе данных, расчетах или методологические ошибки.
2. **Инструментальные ошибки**: Ограничения или неисправности измерительных приборов.
3. **Вычислительные ошибки**: Ошибки, вызванные численными методами или компьютерными арифметическими ограничениями.

Напишите какие ещё источники погрешностей Вы знаете

In [8]:
```
# Напишите ваш ответ между тройными кавычками ниже.
# Это позволит нам автоматически проверить наличие ответа и красиво отобразить его.

t2_ans = '''
В курсе физики мы обсуждали погрешности, связанные с изменением состояния внешней среды (изменение температуры, влажности, давления, отхождение от идеальных условий), если проводятся какие-то измерения с электроприборами могут возникать ошибки из-за перебоев в сети
'''

# Записываем в файл для автопроверки
with open("t2.txt", "w", encoding="utf-8") as f:
    f.write(t2_ans)

# Красиво отображаем ответ студента прямо под ячейкой
from IPython.display import display, Markdown
display(Markdown(t2_ans))
```

В курсе физики мы обсуждали погрешности, связанные с изменением состояния внешней среды (изменение температуры, влажности, давления, отхождение от идеальных условий), если проводятся какие-то измерения с электроприборами могут возникать ошибки из-за перебоев в сети

In [ ]:
```
grader.check("t2")
```

Ошибки в измерениях неизбежны. Деламбер и Мешейн столкнулись с ошибками, связанными с точностью приборов, факторами окружающей среды и человеческими ошибками. Им удалось снизить погрешность до 0,2 миллиметра, что удивительно, учитывая обстоятельства.

Подсчитайте среднее значение и стандартное отклонение. Для этого можно использовать стандратные функции numpy: np.mean и np.std

In [14]:
```
true_distance = 10000

# Simulated measurements with random errors
np.random.seed(42)
measurements = true_distance + np.random.normal(0, 0.2, 100)

mean_measurement = np.mean(measurements)
std_deviation = np.std(measurements)
print(f"Mean Measurement: {mean_measurement} meters")
print(f"Standard Deviation: {std_deviation} meters")
```

```
Mean Measurement: 9999.979230696523 meters
Standard Deviation: 0.18072323532890344 meters
```

In [ ]:
```
grader.check("t3")
```

3. Значимые цифры и точность

Понимание значимых цифр имеет решающее значение для представления точности измерений и вычислений.

- **Значимые цифры**: Цифры, которые несут значение, влияющее на точность измерений.
- **Правила определения значимых цифр**:
  - Ненулевые цифры всегда являются значимыми.
  - Любые нули между значащими цифрами являются значимыми.
  - Начальные нули не являются значимыми.
  - Конечные нули в десятичном числе являются значимыми.

---

4. Упражнения на вычисления

Эти упражнения помогут вам лучше понять ошибки и приближения при использовании Python и популярных библиотек, таких как NumPy и SciPy.

Упражнение 1. Абсолютные и относительные погрешности

Контекст: Представьте, что вы воспроизводите измерения Мешена и Деламбра с помощью современных инструментов.

Задача:

- Вычислить абсолютные и относительные погрешности серии измерений.
- Интерпретировать значимость этих ошибок.

Инструкции:

- **Данные**:

In [27]:
```
# True values (in meters)
true_lengths = np.array([1000, 2000, 3000, 4000, 5000])

# Measured values (in meters)
measured_lengths = np.array([995, 1998, 2995, 3990, 5005])
```

- **Вычислите абсолютную погрешность**:

In [30]:
```
# используем функцию из numpy для поиска абсолютной погрешности
absolute_errors = np.abs(true_lengths - measured_lengths)  # Replace with your ca
lculation
print(absolute_errors)
```

```
[ 5  2  5 10  5]
```

In [ ]:
```
grader.check("t4")
```

- **Вычислите относительную погрешность (в процентах)**:

In [31]:
```
# посчитали относительную погрешность и переводим ее в процентики
relative_errors = (absolute_errors / true_lengths) * 100  # Replace with your cal
culation
print(relative_errors)
```

```
[0.5        0.1        0.16666667 0.25       0.1        ]
```

In [ ]:
```
grader.check("t5")
```

Упражнение 2. Аппроксимация ряда Тейлора

Контекст: Аппроксимация сложных функций и анализ ошибок усечения.

Задача:

- Аппроксимация экспоненциальной функции с использованием ряда Тейлора.
- Вычисление ошибки усечения по сравнению с функцией math.exp в Python.

Инструкции:

- **Импорт необходимых библиотек**:

In [32]:
```
import math
```

- **Определите функцию экспоненты, аппроксимированную через ряд Тейлора**:

Напомним, что ряд Тейлора для экспоненциальной функции выглядит следующим образом:

$$e^x = \sum_{n=0}^{\inf} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots$$

In [33]:
```
def taylor_exp(x, n_terms):
    res = 0
    for n in range (n_terms):
        res += x**n / math.factorial(n)

    return res
```

In [ ]:
```
grader.check("t6")
```

- **Вычислите абсолютную погрешность**:

In [35]:
```
x_value = 1
n_terms = 5

approx_value = taylor_exp(x_value, n_terms)
true_value = math.exp(x_value)
truncation_error = abs(true_value - approx_value) # считаем абсолютную погрешность данных
print(truncation_error)
```

```
0.009948495125712054
```

In [ ]:
```
grader.check("t7")
```

Упражнение 3: Арифметика с плавающей точкой

Контекст: Исследуйте, как компьютеры обрабатывают действительные числа и связанные с ними ошибки.

Задача:

- Исследуйте влияние машинного эпсилона на вычисления с плавающей точкой.
- Анализ результатов операций, связанных с числами, сильно отличающимися по величине.

Инструкции:

- **Машинный эпсилон**:

In [36]:
```
import sys
epsilon = sys.float_info.epsilon
print(f"Machine Epsilon: {epsilon}")
```

```
Machine Epsilon: 2.220446049250313e-16
```

In [ ]:
```
grader.check("t7")
```

- **Вычислительный эксперимент**:

1. Сложите очень большое и маленькое число
2. Вычтите из результата большое число

In [37]:
```
# Large and small numbers
large_number = 1e16
small_number = 1

# Addition
result = large_number + small_number
print("Result of addition:", result)

# Subtraction
diff = result - large_number
print("Difference after subtraction:", diff)
```

```
Result of addition: 1e+16
Difference after subtraction: 0.0
```

In [ ]:
```
grader.check("t8")
```

Анализ:

- Объясните, почему `diff` может быть не равен `small_number`.

In [38]:
```
# Напишите ваш ответ между тройными кавычками ниже.
# Это позволит нам автоматически проверить наличие ответа и красиво отобразить его.

t9_ans = '''
При сложении "1e16 + 1" единичка теряется, тк большое число занимает полностью место выделенное системой для float_64, а при помощи вычитания мы как раз и видим результат такой потери
'''

# Записываем в файл для автопроверки
with open("t9.txt", "w", encoding="utf-8") as f:
    f.write(t9_ans)

# Красиво отображаем ответ студента прямо под ячейкой
from IPython.display import display, Markdown
display(Markdown(t9_ans))
```

при сложении "1e16 + 1" единичка теряется, тк большое число занимает полностью место выделенное системой для float_64, а при помощи вычитания мы как раз и видим результат такой потери

In [ ]:
```
grader.check("t9")
```

5. Современные численные методы

В этом разделе мы рассмотрим, как наиболее популярные численные методы, такие как метод наименьших квадратов и моделирование методом Монте-Карло, могли бы существенно помочь Мешену и Деламбру в их миссии. Эти методы решают многие проблемы, с которыми они столкнулись при измерении меридиана.

5.1 Как современные методы могли бы помочь Мешену и Деламбру

5.1.1 Метод наименьших квадратов

- **Контекст**: Мешен и Деламбр собрали многочисленные данные астрономических наблюдений и наземных измерений, которые были подвержены различным ошибкам и неопределенности.
- **Потенциальные выгоды**:
  - **Подгонка данных**: Метод наименьших квадратов мог бы помочь им привести свои измерения в соответствие с наилучшей возможной моделью, сведя к минимуму влияние случайных ошибок.
  - **Минимизация ошибок**: Статистически минимизируя сумму квадратов остатков (различий между наблюдаемыми и рассчитанными значениями), они могли бы повысить точность своих конечных результатов.
  - **Обработка выбросов**: Это позволило бы им идентифицировать и смягчить влияние точек данных с выбросами, которые могли бы исказить их расчеты.
- **Историческая параллель**:
  - Хотя метод был официально утвержден после их экспедиции (Гаусс разработал его в начале 19 века), применение его принципов могло бы значительно повысить точность их измерений.

Иллюстрирующий Пример:

Предположим, что они провели несколько измерений одного и того же расстояния с небольшими отклонениями:

Measured Distances (km): 100.5, 101.0, 99.8, 100.2, 100.7 Используя метод наименьших квадратов, они могли бы определить наилучшую оценку истинного расстояния, минимизировав сумму квадратов различий между каждым измерением и расчетным значением.

Примечание:

Метод наименьших квадратов был независимо разработан двумя математиками: Карлом Фридрихом Гауссом и Адриеном-Мари Лежандром.

- Часто считается, что Карл Фридрих Гаусс (Германия) изобрел этот метод примерно в 1795 году, когда ему было всего 18 лет. Он использовал его для изучения положения звезд и других небесных тел.
- Адриен-Мари Лежандр (Франция) опубликовал свою версию метода в 1805 году.

Неясно, были ли Пьер Мешен и Жан-Батист Деламбр непосредственно знакомы с методом наименьших квадратов во время своей работы по измерению дуги меридиана для вычисления метра. Их работа была проведена в конце 18 века, примерно в то же время, когда Карл Фридрих Гаусс и Адриен-Мари Лежандр разрабатывали этот метод. Однако Мешен и Деламбр в первую очередь сосредоточились на геодезии и практических аспектах своих измерений, а не на статистических методах.

5.1.2 Моделирование методом Монте-Карло

- **Контекст**: Миссия включала сложные измерения в различных условиях, что приводило к неопределенности, которую было трудно точно определить количественно.
- **Потенциальные выгоды**:
  - **Анализ распространения ошибок**: Моделирование методом Монте-Карло могло бы позволить им смоделировать, как погрешности измерений распространяются в ходе их расчетов.
  - **Вероятностное моделирование**: Смоделировав огромное количество сценариев со случайными вариациями в измерениях, они могли бы оценить распределение вероятностей конечной длины меридиана.
  - **Принятие решений в условиях неопределенности**: Это обеспечило бы статистическую основу для оценки надежности полученных результатов и принятия обоснованных решений о принятии или пересмотре результатов измерений.
- **Историческая параллель**:
  - Хотя метод Монте-Карло был разработан в 20 веке, концепция использования случайной выборки для понимания неопределенностей соответствует задачам, с которыми они столкнулись.

Иллюстрирующий Пример:

Представьте, что вы моделируете тысячи возможных результатов измерений с учетом всех известных ошибок. В результате получается колоколообразная кривая, показывающая наиболее вероятное значение длины меридиана и связанную с ним неопределенность.

Упражнение 4: Метод наименьших квадратов

Контекст: Минимизация ошибок при подборе данных, что имеет решающее значение при научных измерениях.

Задача:

- Используйте метод наименьших квадратов, чтобы сопоставить линию с предоставленными данными, имитируя измерения, аналогичные тем, которые были проведены Мешейном и Деламбром.
- Вычислите и интерпретируйте ошибки подгонки.

Инструкции:

- **Приведенные данные**:

In [39]:
```
# Simulated measurements (distance vs. angle)
distance = np.array([0, 100, 200, 300, 400, 500])  # in kilometers
angle_measurements = np.array([0.0, 0.9, 1.8, 2.7, 3.6, 4.5])  # in degrees with
some measurement errors

# Introduce measurement errors
angle_measurements += np.random.normal(0, 0.05, size=angle_measurements.shape)
```

Примечание: Предполагается, что истинная взаимосвязь равна `angle = 0.009 * distance` .

- **Выполните линейную регрессию**:

Необходимо построить график линейной функции $f(x_i) = b_0 + b_1 * x_i$ и рассчитать для каждой точки величину отклонения (разница между значением найденной функции и величиной исходных данных *angle_measurements*), т.е. задача сводится к нахождению коэффициентов b_0 и b_1. Из условия выше $b_1 = angle = 0.009 * distance$

Дополнительно про линейную регрессию в Python можно почитать тут (https://realpython.com/linear-regression-in-python/) или тут (https://proglib.io/p/linear-regression). Помимо упоминаемых в статье scikit-learn и statsmodel можно использовать и различные функции numpy, например, можно поэкспериментировать с np.linalg.lstsq (https://numpy.org/doc/stable/reference/generated/numpy.linalg.lstsq.html) или np.polyfit (https://numpy.org/doc/stable/reference/generated/numpy.polyfit.html)

In [43]:
```
slope, intercept = np.polyfit(distance, angle_measurements, 1)

print(f"Slope: {slope}")
print(f"Intercept: {intercept}")
```

```
Slope: 0.009134508915356112
Intercept: -0.05644591805082641
```

In [ ]:
```
grader.check("t10")
```

- **Вычислите ошибки подгонки**: Здесь необходимо посчитать разницу между значениями полученной линейной функции и значениями исходных данных. Сделать это можно:

1. подставив значения slope и intercept в формулу $f(x_i) = b_0 + b_1 * x_i$ и вычислив значения этой функции в каждой точке x;
2. посчитать разницу между полученными значениями линейной функции (аппроксимации) и значениями исходных данных;
3. возвести каждое значение разницы в квадрат и сложить, чтобы узнать величину отклонения для всей линейной функции - это позволит говорить о том, на сколько удачно выбрана аппроксимирующая функция.

$\sum_{x_i} \Delta y_i = \sum_{x_i} (y_i - f(x_i))^2$

In [44]:
```
# Predicted values
angle_pred = intercept + slope * distance

# Residuals
residuals = angle_measurements - angle_pred

# Sum of squares of residuals
ss_res = np.sum(residuals ** 2)

print("Sum of squares of residuals:", ss_res)
```

```
Sum of squares of residuals: 0.0015361037247457723
```

In [ ]:
```
grader.check("t11")
```

- **Интерпретация**:
  - Как Вы думаете, как минимизация суммы квадратов невязок повышает точность измерений?
  - Объясните, как этот метод мог бы помочь устранить систематические ошибки при измерении по меридиану.

In [46]:
```
# Напишите ваш ответ между тройными кавычками ниже.
# Это позволит нам автоматически проверить наличие ответа и красиво отобразить его.

t12_ans = '''
Минимизация суммы квадратов невязок помогает построить график "усредненной" прямой, которая лучше всего сможет описать все точки данных

При использовании такого метода погрешности случайных измерений взаимно компенсируются при возведении в квадрат, а систематические ошибки автоматически включаются в свободный член регрессии (intercept) сохраняя точность оценки наклона
'''

# Записываем в файл для автопроверки
with open("t12.txt", "w", encoding="utf-8") as f:
    f.write(t12_ans)

# Красиво отображаем ответ студента прямо под ячейкой
from IPython.display import display, Markdown
display(Markdown(t12_ans))
```

Минимизация суммы квадратов невязок помогает построить график "усредненной" прямой, которая лучше всего сможет описать все точки данных

При использовании такого метода погрешности случайных измерений взаимно компенсируются при возведении в квадрат, а систематические ошибки автоматически включаются в свободный член регрессии (intercept) сохраняя точность оценки наклона

In [ ]:
```
grader.check("t12")
```

Упражнение 5: Моделирование методом Монте-Карло

Контекст: Понимание распространения ошибок с использованием вероятностных методов.

Задача:

- Смоделировать измерение длины меридиана с неопределенностями, используя моделирование методом Монте-Карло.
- Проанализировать распределение вычисленных длин метров на основе ошибок измерений.

Инструкции:

- **Константы**:

In [47]:
```
# True meridian length (from pole to equator) in meters
true_meridian_length = 10_000_000  # Definition of the meter

# Number of simulations
num_simulations = 10000
```

- **Имитируйте ошибки измерений**:

In [48]:
```
# Define measurement error (standard deviation in meters)
measurement_error_sd = 10  # Simulating instrument and observational errors
```

- **Функция моделирования методом Монте-Карло**: Реализуйте функцию, которая будет симулировать большое количество измерений одной и той же величины путём добавления к ней случаных отклонений.

In [50]:
```
def simulate_meridian_measurements(num_simulations, true_length, error_sd):

    # Монте-Карло симуляция измерений расстояния по меридиану
    true_angle = 0.009 * true_length  # Истинный угол: 0.009
    return true_angle + np.random.normal(0, error_sd, num_simulations)
```

In [ ]:
```
grader.check("t13")
```

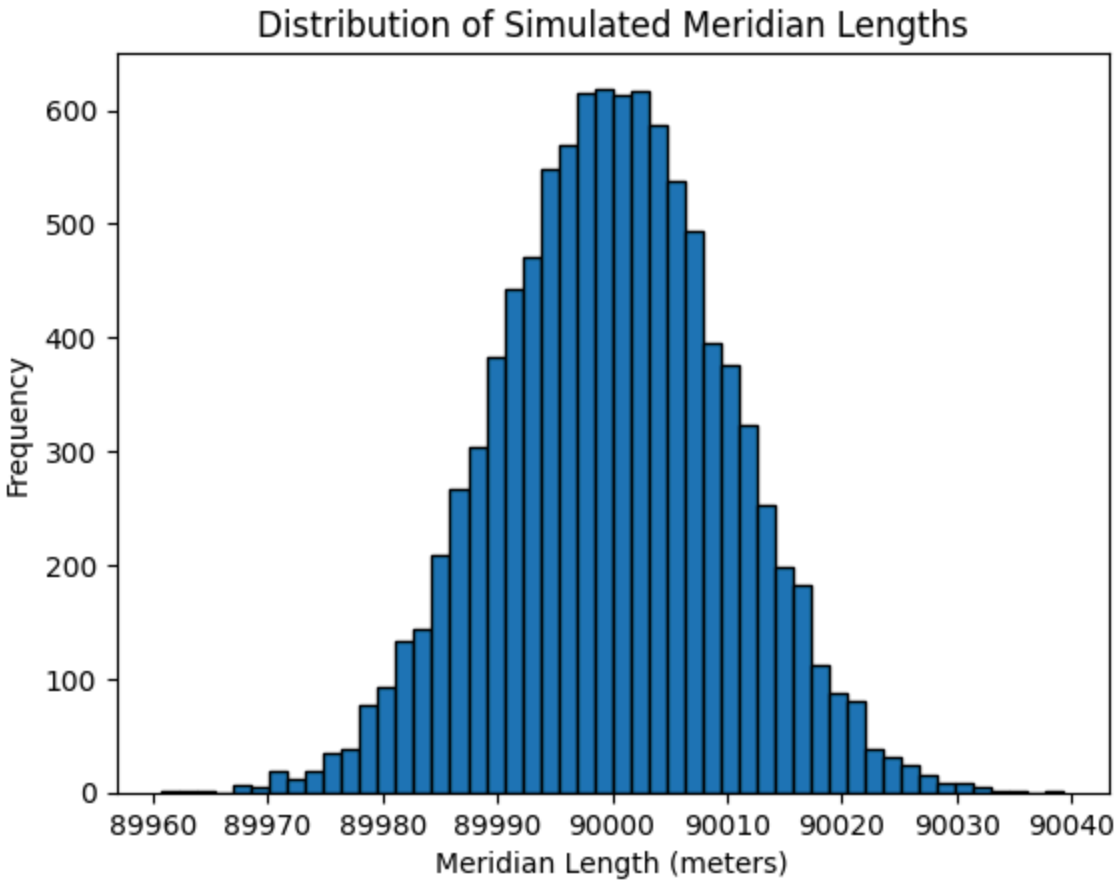
- **Выполните моделирование и проанализируйте результаты**:

In [51]:
```
# Simulate measurements
simulated_lengths = simulate_meridian_measurements(num_simulations, true_meridian
_length, measurement_error_sd)

# Compute statistical information
mean_length = np.mean(simulated_lengths)
std_length = np.std(simulated_lengths)
print(f"Mean Simulated Length: {mean_length} meters")
print(f"Standard Deviation: {std_length} meters")
```

```
Mean Simulated Length: 89999.98837466161 meters
Standard Deviation: 10.039300889910256 meters
```

- **Визуализация**:

In [52]:
```
import matplotlib.pyplot as plt

# Plot histogram of simulated lengths
plt.hist(simulated_lengths, bins=50, edgecolor='black')
plt.title('Distribution of Simulated Meridian Lengths')
plt.xlabel('Meridian Length (meters)')
plt.ylabel('Frequency')
plt.show()
```



- **Интерпретация**:
  - Объясните, как моделирование позволяет получить представление о неопределенности и вариативности измерений.
  - Порассуждайте, как этот метод мог бы помочь в оценке достоверности полученных результатов.

In [54]:
```
# Напишите ваш ответ между тройными кавычками ниже.
# Это позволит нам автоматически проверить наличие ответа и красиво отобразить его.

t14_ans = '''
Моделирование таким методом позволяет нам увидеть полную картину всех возможных результатов изме
рений, а не только среднее значение, мы как бы можем более широко рассмотреть выборку

Говоря о практичеком применении, мы получаем больше данных для логических рассуждений, можно п
осмотреть на распределение и валидировать его, мы способны оценить качество проделанной работы
'''

# Записываем в файл для автопроверки
with open("t14.txt", "w", encoding="utf-8") as f:
    f.write(t14_ans)

# Красиво отображаем ответ студента прямо под ячейкой
from IPython.display import display, Markdown
display(Markdown(t14_ans))
```

Моделирование таким методом позволяет нам увидеть полную картину всех возможных результатов измерений, а не только среднее значение, мы как бы можем более широко рассмотреть выборку

Говоря о практичеком применении, мы получаем больше данных для логических рассуждений, можно посмотреть на распределение и валидировать его, мы способны оценить качество проделанной работы

In [ ]:
```
grader.check("t14")
```

---

6. Продвинутые темы по теории погрешностей

В этом разделе мы рассмотрим передовые концепции, связанные с ошибками и приближениями, связав их с исторической миссией Пьера Мешена и Жан-Батиста Деламбра по измерению дуги меридиана между Дюнкерком и Барселоной. Эти темы углубят ваше понимание того, как могут распространяться ошибки, как возникают несоответствия и как сложные математические концепции, такие как фракталы и теория хаоса, могут быть связаны с измерениями и аппроксимацией.

6.1 Общая формула погрешности

Исторический контекст:

Пьер Мешен и Жан-Батист Деламбр стремились к максимальной точности своих измерений, определяя измерительный прибор на основе доли земного меридиана. Понимание и применение общей формулы погрешности имело бы решающее значение для количественной оценки неопределенностей в их измерениях.

Теория:

Общая формула погрешности позволяет оценить максимально возможную погрешность в функции, которая зависит от нескольких переменных, каждая из которых имеет свою собственную погрешность измерения. Для функции $f(x_1, x_2, \ldots, x_n)$ суммарный дифференциал дает приблизительное значение ошибки в f:

$$\Delta f \approx \left| \frac{\partial f}{\partial x_1} \Delta x_1 \right| + \left| \frac{\partial f}{\partial x_2} \Delta x_2 \right| + \cdots + \left| \frac{\partial f}{\partial x_n} \Delta x_n \right|$$

Эта формула позволяет нам оценить, как ошибки во входных переменных влияют на конечный результат.

Применение к измерению по меридиану:

Длина измеряемой дуги меридиана зависит от углов и расстояний, каждое из которых имеет свои собственные погрешности измерения. Применение общей формулы погрешности помогло бы оценить общую неопределенность при определении показаний счетчика.

Упражнение 6: Общая формула ошибки

Задача:

- Для упрощенной модели рассмотрим расчет расстояния D между двумя точками с использованием триангуляции:

$$D = \frac{H}{\sin(\theta)}$$

где:

- H - это базовое измерение (расстояние между двумя известными точками).
- θ - измеренный угол.

Учитывая погрешности измерений в H и θ, используйте общую формулу для определения погрешности в D.

Инструкции:

- **Данные**:

In [ ]:
```
H = 10000  # meters
theta_deg = 30  # degrees
delta_H = 5     # measurement error in H (meters)
delta_theta_deg = 0.01  # measurement error in theta (degrees)
```

- **Преобразование угла в радианы**:

In [ ]:
```
theta = np.radians(theta_deg)
delta_theta = np.radians(delta_theta_deg)
```

- **Определите функцию $D(H, \theta)$**:

In [ ]:
```
def D(H, theta):
    return H / np.sin(theta)
```

- **Вычислите частные производные**:

In [ ]:
```
# Compute partial derivative with respect to H
def partial_D_H(H, theta):
    return 1 / np.sin(theta)


# Compute partial derivative with respect to theta
def partial_D_theta(H, theta):
    return -H * np.cos(theta) / (np.sin(theta))**2
```

- **Вычислите абсолютную погрешность в D, используя общую формулу погрешности**:

In [ ]:
```
# YOUR CODE HERE
# Calculate partial derivatives at the given values
partial_H = ...
partial_theta = ...

# Calculate the total error
delta_D = ...

print(f"Estimated Error in D: {delta_D:.2f} meters")
```

In [ ]:
```
grader.check("t15")
```

- **Вычислите относительную погрешность**:

In [ ]:
```
# YOUR CODE HERE
relative_error = ...
print(f"Relative Error: {relative_error * 100:.4f}%")

relative_error = delta_D / D(H, theta)
print(f"Relative Error: {relative_error * 100:.4f}%")
```

In [ ]:
```
grader.check("t16")
```

6.2 Обратная задача теории ошибок

Исторический контекст:

Во время своей экспедиции Мешену и Деламбру часто приходилось отступать от результатов наблюдений, чтобы сделать вывод о начальных условиях или скорректировать измерения, особенно когда они имели дело с накопленными ошибками.

Теория:

Обратная задача теории ошибок заключается в нахождении исходных входных значений с учетом результатов измерений и модели влияния ошибок на эти измерения. Это имеет решающее значение в таких областях, как астрономия и геодезия, где широко распространены косвенные измерения.

Применение для измерения по меридиану:

Предположим, что на некоторые измерения повлияли постоянные ошибки, вызванные неправильной калибровкой прибора. Решение обратной задачи помогло бы скорректировать записанные значения, чтобы они соответствовали истинным измерениям.

Упражнение 7: Обратная задача теории ошибок

Задача:

- Задано измеренное расстояние D_{measured}, на которое влияет неизвестная систематическая ошибка ϵ, и модель, которая связывает их:

$$D_{\text{measured}} = D_{\text{true}}(1 + \epsilon)$$

- Если вы знаете D_{measured} и имеете приблизительное значение ϵ, вычислите D_{true}.

Инструкции:

- **Данные**:

In [ ]:
```
D_measured = 10050  # meters
epsilon_estimate = 0.005  # estimated systematic error (0.5%)
```

- **Вычислите истинное расстояние**:

In [ ]:
```
# YOUR CODE HERE
D_true = ...
print(f"Estimated True Distance: {D_true:.2f} meters")
```

In [ ]:
```
grader.check("t17")
```

6.3 Теория несоответствий (discrepancies)

Исторический контекст:

несоответствия между ожидаемыми и наблюдаемыми измерениями были обычным явлением во время экспедиции "Меридиан". Понимание и анализ этих расхождений помогли бы усовершенствовать методы измерений и исправить ошибки.

Теория:

Теория несоответствий предполагает изучение распределения различий между наблюдаемыми данными и ожидаемыми значениями, часто для выявления закономерностей, смещений или случайных ошибок.

Применение к измерениям по меридиану:

Проанализировав расхождения в угловых измерениях или расстояниях, Мешен и Деламбр могли бы выявить систематические ошибки или несоответствия в своих данных.

Упражнение 8: Анализ распределения несоответствий

Задание:

- Учитывая набор ожидаемых и наблюдаемых угловых измерений, вычислите расхождения и проанализируйте их распределение.

Инструкции:

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

- **Данные**:

In [ ]:
```
expected_angles_deg = np.array([30, 45, 60, 75, 90])  # degrees
observed_angles_deg = np.array([29.98, 44.95, 60.05, 75.02, 89.97])  # degrees
discrepancies = observed_angles_deg - expected_angles_deg
```

- **Постройте график распределения несоответствий**:

In [ ]:
```
plt.hist(discrepancies, bins=5, edgecolor='black')
plt.title('Discrepancies in Angle Measurements')
plt.xlabel('Discrepancy (degrees)')
plt.ylabel('Frequency')
plt.show()
```

- **Проанализируйте результаты**:
  - Распределены ли расхождения случайным образом?
  - Есть ли признаки систематической ошибки?

In [ ]:
```
# Напишите ваш ответ между тройными кавычками ниже.
# Это позволит нам автоматически проверить наличие ответа и красиво отобразить его.

t18_ans = '''
*ВАШ КОММЕНТАРИЙ ЗДЕСЬ*
'''

# Записываем в файл для автопроверки
with open("t18.txt", "w", encoding="utf-8") as f:
    f.write(t18_ans)

# Красиво отображаем ответ студента прямо под ячейкой
from IPython.display import display, Markdown
display(Markdown(t18_ans))
```

In [ ]:
```
grader.check("t18")
```

6.4 Парадокс береговой линии

Исторический контекст:

Хотя парадокс береговой линии был официально описан гораздо позже, задача измерения неправильных форм была актуальна во времена Мешена и Деламбра, особенно при картографировании береговой линии Франции или неровной местности, что могло повлиять на их базовые измерения.

Теория:

Парадокс береговой линии связан с парадоксальным наблюдением о том, что длина береговой линии может сильно варьироваться в зависимости от используемой шкалы измерений. Это связано с тем, что береговые линии похожи на фракталы и имеют сложность в нескольких масштабах.

Применение к измерениям по меридианам:

Понимание сложности естественных границ могло бы помочь оценить ограничения методов измерения и необходимость в согласованных масштабах.

Упражнение 9: Исследуем парадокс береговой линии

Задача:

- Смоделируйте измерение фрактальной береговой линии, используя различные шкалы измерений, и понаблюдайте, как меняется общая длина.

Инструкции:

- **Определите функцию для генерации кривой Коха**:

In [ ]:
```
def koch_curve(order, scale=1):
    def recursive_koch(points, order):
        if order == 0:
            return points
        else:
            new_points = []
            for i in range(len(points) - 1):
                p0 = points[i]
                p4 = points[i + 1]
                delta = (p4 - p0) / 3
                p1 = p0 + delta
                p3 = p0 + 2 * delta
                angle = np.pi / 3
                rotation = np.array([[np.cos(angle), -np.sin(angle)],
                                     [np.sin(angle),  np.cos(angle)]])
                p2 = p1 + rotation.dot(delta)
                new_points.extend([p0, p1, p2, p3])
            new_points.append(points[-1])
            return recursive_koch(np.array(new_points), order - 1)
    initial_points = np.array([[0, 0], [scale, 0]])
    return recursive_koch(initial_points, order)
```

- **Измерьте длину с помощью различных масштабов**:

In [ ]:
```
scales = [1, 0.5, 0.25, 0.125]  # Measurement scales
lengths = []

for s in scales:
    order = int(np.log2(1 / s))
    points = koch_curve(order, scale=1)
    # Calculate segment lengths
    distances = np.sqrt(np.sum(np.diff(points, axis=0)**2, axis=1))
    total_length = np.sum(distances)
    lengths.append(total_length)
```

- **Постройте график длины в зависимости от масштаба**:

In [ ]:
```
plt.figure()
plt.plot(scales, lengths, marker='o')
plt.xlabel('Measurement Scale')
plt.ylabel('Total Length')
plt.title('Coastline Length vs. Measurement Scale')
plt.gca().invert_xaxis()
plt.show()
```

- **Проанализируйте результаты**:
  - Опишите, как изменяется измеренная длина в зависимости от шкалы измерений.
  - Соотнесите это с трудностями, связанными с точными измерениями во время экспедиции.

In [ ]:
```
# Напишите ваш ответ между тройными кавычками ниже.
# Это позволит нам автоматически проверить наличие ответа и красиво отобразить его.

t19_ans = '''
*ВАШ КОММЕНТАРИЙ ЗДЕСЬ*
'''

# Записываем в файл для автопроверки
with open("t19.txt", "w", encoding="utf-8") as f:
    f.write(t19_ans)

# Красиво отображаем ответ студента прямо под ячейкой
from IPython.display import display, Markdown
display(Markdown(t19_ans))
```

In [ ]:
```
grader.check("t19")
```

6.5 Основы теории фракталов

Исторический контекст:

Несмотря на то, что теория фракталов была разработана спустя столетия после экспедиции на меридиан, в мире природы существуют фрактальные структуры, которые могли повлиять на точность измерений.

Теория:

Фракталы - это сложные структуры, которые проявляют самоподобие в разных масштабах. Они имеют нецелые (фрактальные) размеры, что усложняет традиционные методы измерения.

Применение для измерения по меридианам:

Неровная местность и объекты с фрактальными характеристиками могут затруднить точное измерение расстояний с использованием прямолинейных приближений.

Упражнение 10: Генерация и измерение фракталов

Задача:

- Сгенерируйте простой фрактал (например, треугольник Серпинского) и вычислите его периметр и площадь на разных уровнях итерации.

Инструкции:

- **Определите функцию для генерации треугольника Серпинского**:

In [ ]:
```
def sierpinski_triangle(order, points):
    if order == 0:
        return [points]
    else:
        triangles = []
        mid_points = [
            (points[0] + points[1]) / 2,
            (points[1] + points[2]) / 2,
            (points[2] + points[0]) / 2
        ]
        triangles += sierpinski_triangle(order - 1, [points[0], mid_points[0], mi
d_points[2]])
        triangles += sierpinski_triangle(order - 1, [mid_points[0], points[1], mi
d_points[1]])
        triangles += sierpinski_triangle(order - 1, [mid_points[2], mid_points
[1], points[2]])
        return triangles
```

- **Рассчитайте периметр и площадь**:

In [ ]:
```
import matplotlib.pyplot as plt

initial_points = np.array([[0, 0], [1, 0], [0.5, np.sqrt(3)/2]])

orders = [0, 1, 2, 3]
perimeters = []
areas = []

for order in orders:
    triangles = sierpinski_triangle(order, initial_points)
    perimeter = 0
    area = 0
    for tri in triangles:
        # Calculate side lengths
        a = np.linalg.norm(tri[0] - tri[1])
        b = np.linalg.norm(tri[1] - tri[2])
        c = np.linalg.norm(tri[2] - tri[0])
        s = (a + b + c) / 2
        # Calculate perimeter and area
        perimeter += (a + b + c)
        area += np.sqrt(s * (s - a) * (s - b) * (s - c))
    perimeters.append(perimeter)
    areas.append(area)
```

- **Постройте график с результатами**:

In [ ]:
```
plt.figure()
plt.plot(orders, perimeters, marker='o')
plt.xlabel('Order')
plt.ylabel('Perimeter')
plt.title('Perimeter vs. Order in Sierpinski Triangle')
plt.show()

plt.figure()
plt.plot(orders, areas, marker='o')
plt.xlabel('Order')
plt.ylabel('Area')
plt.title('Area vs. Order in Sierpinski Triangle')
plt.show()
```

- **Проанализируйте результаты**:
  - Понаблюдайте, как увеличивается периметр, в то время как площадь уменьшается или остается постоянной.
  - Обсудите последствия для точности измерений.

In [ ]:
```
# Напишите ваш ответ между тройными кавычками ниже.
# Это позволит нам автоматически проверить наличие ответа и красиво отобразить его.

t20_ans = '''
*ВАШ КОММЕНТАРИЙ ЗДЕСЬ*
'''

# Записываем в файл для автопроверки
with open("t20.txt", "w", encoding="utf-8") as f:
    f.write(t20_ans)

# Красиво отображаем ответ студента прямо под ячейкой
from IPython.display import display, Markdown
display(Markdown(t20_ans))
```

In [ ]:
```
grader.check("t20")
```

6.6 Основы теории хаоса

Исторический контекст:

Хотя теория хаоса возникла в 20 веке, чувствительность к начальным условиям была концепцией, которая могла повлиять на повторные измерения во время экспедиции.

Теория:

Теория хаоса изучает системы, которые очень чувствительны к начальным условиям, что приводит к кажущемуся случайным и непредсказуемым поведению в детерминированных системах.

Применение к измерениям по меридиану:

Небольшие ошибки в начальных измерениях или условиях могут привести к значительным отклонениям в результатах, что подчеркивает необходимость в точности.

Упражнение 11: Чувствительность к начальным условиям

Задача:

- Исследуйте простую хаотическую систему, такую как логистическая карта, чтобы проиллюстрировать чувствительность к начальным условиям.

Инструкции:

- **Определите функцию логистической карты**:

In [ ]:
```
def logistic_map(r, x):
    return r * x * (1 - x)
```

- **Промоделируйте логистическую карту для двух близких начальных условий**:

In [ ]:
```
r = 3.99  # Parameter value in chaotic regime
x0 = 0.5
x1 = x0 + 1e-8  # Slightly different initial condition

iterations = 100
x_values_0 = [x0]
x_values_1 = [x1]

for i in range(iterations):
    x0 = logistic_map(r, x0)
    x1 = logistic_map(r, x1)
    x_values_0.append(x0)
    x_values_1.append(x1)
```

- **Постройте график по результатам**:

In [ ]:
```
plt.figure()
plt.plot(range(iterations + 1), x_values_0, label='x0')
plt.plot(range(iterations + 1), x_values_1, label='x1')
plt.xlabel('Iteration')
plt.ylabel('x')
plt.title('Sensitivity to Initial Conditions in Logistic Map')
plt.legend()
plt.show()
```

- **Проанализируйте результаты**:
  - Обсудите, как небольшие различия в начальных условиях приводят к большим расхождениям.
  - Объясните это важностью точности измерений во время экспедиции.

In [ ]:
```
# Напишите ваш ответ между тройными кавычками ниже.
# Это позволит нам автоматически проверить наличие ответа и красиво отобразить его.

t21_ans = '''
*ВАШ КОММЕНТАРИЙ ЗДЕСЬ*
'''

# Записываем в файл для автопроверки
with open("t21.txt", "w", encoding="utf-8") as f:
    f.write(t21_ans)

# Красиво отображаем ответ студента прямо под ячейкой
from IPython.display import display, Markdown
display(Markdown(t21_ans))
```

In [ ]:
```
grader.check("t21")
```

Понимание ошибок и управление ими является краеугольным камнем вычислительной математики, точно так же, как это было важно для Мешена и Деламбра в их стремлении дать определение счетчику. С помощью этого блокнота вы изучили:

- Типы и источники ошибок.
- Методы вычисления и минимизации ошибок.
- Как современные численные методы могли бы помочь историческим научным исследованиям.
- Какое глубокое влияние ошибки могут оказать на научные достижения.

По мере продвижения в учебе и карьере умение распознавать ошибки и устранять их будет иметь неоценимое значение.

---

7. Источники

- Alder, K. (2002). *The Measure of All Things: The Seven-Year Odyssey and Hidden Error That Transformed the World*. Free Press.
- NumPy, Matplotlib and SciPy documentation.
- Articles on numerical methods and error analysis.
- Gauss, C. F. (1809). *Theoria motus corporum coelestium in sectionibus conicis solem ambientium*.
- Mandelbrot, B. (1982). *The Fractal Geometry of Nature*. W.H. Freeman.
- Gleick, J. (1987). *Chaos: Making a New Science*. Viking.

Создание отправляемого решения для Moodle

Пожалуйста, убедитесь в том, что запустили все ячейки в ноутбуке, прежде чем запускать следующую ячейку, чтобы все Ваши результаты попали в отправляемое решение. Эта ячейка сгенерирует zip-архив для загрузки в Moodle. **Пожалуйста, сохраните файл и не очищайте вывод ячеек перед запуском следующей ячейки.**

In [55]:

```
import os
from otter.export import export_notebook
from zipfile import ZipFile

# 1. Запуск тестов (опционально, если хотите видеть результат перед экспортом)
grader.check_all()

# 2. Экспорт в PDF через HTML (явно указываем тип)
# nb_path - путь к текущему ноутбуку. Обычно Otter находит его сам, но можно указать явно
pdf_path = export_notebook("PW2.ipynb", exporter_type="html")

# 3. Создание ZIP архива
submission_filename = "submission.zip"
text_outputs = [fn for fn in os.listdir("./")
                if any(fn.endswith(ext) for ext in ["txt"])]
with ZipFile(submission_filename, 'w') as zipf:
    zipf.write("PW2.ipynb")  # Сам ноутбук
    zipf.write(pdf_path)     # PDF файл
    for f in text_outputs:
        zipf.write(f)

print(f"Submission file created: {submission_filename}")
```

```
---------------------------------------------------------------------------
Error                                     Traceback (most recent call last)
File ~/PycharmProjects/itmo-math-statistics/.venv/lib/python3.9/site-packages/nb
convert/exporters/webpdf.py:110, in WebPDFExporter.run_playwright.<locals>.main
(temp_file)
    109 try:
--> 110     browser = await chromium.launch(
    111         handle_sigint=False, handle_sigterm=False, handle_sighup=False,
args=args
    112     )
    113 except Exception as e:

File ~/PycharmProjects/itmo-math-statistics/.venv/lib/python3.9/site-packages/pl
aywright/async_api/_generated.py:14547, in BrowserType.launch(self, executable_p
ath, channel, args, ignore_default_args, handle_sigint, handle_sigterm, handle_s
ighup, timeout, env, headless, proxy, downloads_path, slow_mo, traces_dir, chrom
ium_sandbox, firefox_user_prefs)
  14456 """BrowserType.launch
  14457
  14458 Returns the browser instance.
   (...)
  14543 Browser
  14544 """
  14546 return mapping.from_impl(
> 14547     await self._impl_obj.launch(
  14548         executablePath=executable_path,
  14549         channel=channel,
  14550         args=mapping.to_impl(args),
  14551         ignoreDefaultArgs=mapping.to_impl(ignore_default_args),
  14552         handleSIGINT=handle_sigint,
  14553         handleSIGTERM=handle_sigterm,
  14554         handleSIGHUP=handle_sighup,
  14555         timeout=timeout,
  14556         env=mapping.to_impl(env),
  14557         headless=headless,
  14558         proxy=proxy,
  14559         downloadsPath=downloads_path,
  14560         slowMo=slow_mo,
  14561         tracesDir=traces_dir,
  14562         chromiumSandbox=chromium_sandbox,
  14563         firefoxUserPrefs=mapping.to_impl(firefox_user_prefs),
  14564     )
  14565 )

File ~/PycharmProjects/itmo-math-statistics/.venv/lib/python3.9/site-packages/pl
aywright/_impl/_browser_type.py:97, in BrowserType.launch(self, executablePath,
channel, args, ignoreDefaultArgs, handleSIGINT, handleSIGTERM, handleSIGHUP, tim
eout, env, headless, proxy, downloadsPath, slowMo, tracesDir, chromiumSandbox, f
irefoxUserPrefs)
     93 normalize_launch_params(params)
     94 browser = cast(
     95     Browser,
     96     from_channel(
---> 97         await self._channel.send(
     98             "launch", TimeoutSettings.launch_timeout, params
     99         )
    100     ),
    101 )
    102 browser._connect_to_browser_type(
    103     self, str(tracesDir) if tracesDir is not None else None
    104 )
```

```
File ~/PycharmProjects/itmo-math-statistics/.venv/lib/python3.9/site-packages/playwright/_impl/_connection.py:69, in Channel.send(self, method, timeout_calculator, params, is_internal, title)
     61 async def send(
     62     self,
     63     method: str,
   (...)
     67     title: str = None,
     68 ) -> Any:
---> 69     return await self._connection.wrap_api_call(
     70         lambda: self._inner_send(method, timeout_calculator, params, False),
     71         is_internal,
     72         title,
     73     )

File ~/PycharmProjects/itmo-math-statistics/.venv/lib/python3.9/site-packages/playwright/_impl/_connection.py:559, in Connection.wrap_api_call(self, cb, is_internal, title)
    558 except Exception as error:
--> 559     raise rewrite_error(error, f"{parsed_st['apiName']}: {error}") from None
    560 finally:

Error: BrowserType.launch: Executable doesn't exist at /Users/scobca/Library/Caches/ms-playwright/chromium_headless_shell-1208/chrome-headless-shell-mac-arm64/chrome-headless-shell
╔═══════════════════════════════════════════════════════════╗
║ Looks like Playwright was just installed or updated.      ║
║ Please run the following command to download new browsers:║
║                                                           ║
║     playwright install                                    ║
║                                                           ║
║ <3 Playwright Team                                        ║
╚═══════════════════════════════════════════════════════════╝

The above exception was the direct cause of the following exception:

RuntimeError                              Traceback (most recent call last)
File ~/PycharmProjects/itmo-math-statistics/.venv/lib/python3.9/site-packages/otter/export/exporters/via_html.py:50, in PDFViaHTMLExporter.convert_notebook(cls, nb_path, dest, **kwargs)
     49 try:
---> 50     pdf, _ = nbconvert.export(exporter, nb)
     51 except RuntimeError as e:
     52     # Replace nbconvert's error about installing chromium since their flag can't be passed
     53     # to Otter.

File ~/PycharmProjects/itmo-math-statistics/.venv/lib/python3.9/site-packages/nbconvert/exporters/base.py:86, in export(exporter, nb, **kw)
     85 if isinstance(nb, NotebookNode):
---> 86     output, resources = exporter_instance.from_notebook_node(nb, resources)
     87 elif isinstance(nb, (str,)):

File ~/PycharmProjects/itmo-math-statistics/.venv/lib/python3.9/site-packages/nbconvert/exporters/webpdf.py:179, in WebPDFExporter.from_notebook_node(self, nb, resources, **kw)
    178 self.log.info("Building PDF")
--> 179 pdf_data = self.run_playwright(html)
```

```
    180 self.log.info("PDF successfully created")

File ~/PycharmProjects/itmo-math-statistics/.venv/lib/python3.9/site-packages/nb
convert/exporters/webpdf.py:168, in WebPDFExporter.run_playwright(self, html)
    167 try:
--> 168     pdf_data = pool.submit(asyncio.run, main(temp_file)).result()
    169 finally:
    170     # Ensure the file is deleted even if playwright raises an exception

File /Library/Developer/CommandLineTools/Library/Frameworks/Python3.framework/Ve
rsions/3.9/lib/python3.9/concurrent/futures/_base.py:445, in Future.result(self,
timeout)
    444 elif self._state == FINISHED:
--> 445     return self.__get_result()
    446 else:

File /Library/Developer/CommandLineTools/Library/Frameworks/Python3.framework/Ve
rsions/3.9/lib/python3.9/concurrent/futures/_base.py:390, in Future.__get_result
(self)
    389 try:
--> 390     raise self._exception
    391 finally:
    392     # Break a reference cycle with the exception in self._exception

File /Library/Developer/CommandLineTools/Library/Frameworks/Python3.framework/Ve
rsions/3.9/lib/python3.9/concurrent/futures/thread.py:52, in _WorkItem.run(self)
     51 try:
---> 52     result = self.fn(*self.args, **self.kwargs)
     53 except BaseException as exc:

File /Library/Developer/CommandLineTools/Library/Frameworks/Python3.framework/Ve
rsions/3.9/lib/python3.9/asyncio/runners.py:44, in run(main, debug)
     43         loop.set_debug(debug)
---> 44     return loop.run_until_complete(main)
     45 finally:

File /Library/Developer/CommandLineTools/Library/Frameworks/Python3.framework/Ve
rsions/3.9/lib/python3.9/asyncio/base_events.py:642, in BaseEventLoop.run_until_
complete(self, future)
    640     raise RuntimeError('Event loop stopped before Future completed.')
--> 642 return future.result()

File ~/PycharmProjects/itmo-math-statistics/.venv/lib/python3.9/site-packages/nb
convert/exporters/webpdf.py:120, in WebPDFExporter.run_playwright.<locals>.main
(temp_file)
    119     await playwright.stop()
--> 120     raise RuntimeError(msg) from e
    122 page = await browser.new_page()

RuntimeError: No suitable chromium executable found on the system. Please use '-
-allow-chromium-download' to allow downloading one,or install it using `playwrig
ht install chromium`.

During handling of the above exception, another exception occurred:

RuntimeError                              Traceback (most recent call last)
Cell In[55], line 10
      6 grader.check_all()
      8 # 2. Экспорт в PDF через HTML (явно указываем тип)
      9 # nb_path - путь к текущему ноутбуку. Обычно Otter находит его сам, но можно ук
азать явно
---> 10 pdf_path = export_notebook("PW2.ipynb", exporter_type="html")
```

```
     12 # 3. Создание ZIP архива
     13 submission_filename = "submission.zip"

File ~/PycharmProjects/itmo-math-statistics/.venv/lib/python3.9/site-packages/otter/export/__init__.py:43, in export_notebook(nb_path, dest, exporter_type, **kwargs)
     40     pdf_name = os.path.splitext(nb_path)[0] + ".pdf"
     42 Exporter = get_exporter(exporter_type=exporter_type)
---> 43 Exporter.convert_notebook(nb_path, pdf_name, **kwargs)
     45 return pdf_name

File ~/PycharmProjects/itmo-math-statistics/.venv/lib/python3.9/site-packages/otter/export/exporters/via_html.py:55, in PDFViaHTMLExporter.convert_notebook(cls, nb_path, dest, **kwargs)
     51 except RuntimeError as e:
     52     # Replace nbconvert's error about installing chromium since their flag can't be passed
     53     # to Otter.
     54     if "--allow-chromium-download" in str(e):
---> 55         raise RuntimeError(
     56             "No suitable version of chromium was found; please install chromium by running "
     57             "'playwright install chromium'"
     58         )
     59     raise e
     61 pdf_path = os.path.splitext(dest)[0] + ".pdf"

RuntimeError: No suitable version of chromium was found; please install chromium by running 'playwright install chromium'
```

Теперь полученный zip архив можно загрузить в [Moodle (https://learning.cosm-lab.science/moodle/mod/assign/view.php?id=336)](https://learning.cosm-lab.science/moodle/mod/assign/view.php?id=336).

Спасибо за энтузиазм и работу на занятии! Работы будут в скором времени проверены, а результаты выставлены в [google-журнал (https://docs.google.com/spreadsheets/d/1SLSdgPLtW2mugeKOu_Z6VGyn94W1yEalbYACdwIpzTw/edit?usp=sharing)](https://docs.google.com/spreadsheets/d/1SLSdgPLtW2mugeKOu_Z6VGyn94W1yEalbYACdwIpzTw/edit?usp=sharing).