Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ *Робототехники и комплексной автоматизации*

КАФЕДРА *Системы автоматизированного проектирования (РК-6)*

ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

по дисциплине: «Вычислительная математика»

Студент	Тетерин Никита Евгеньевич
Группа	РК6-54б
Тип задания	лабораторная работа
Тема лабораторной работы	Использование аппроксимаций для численной оптимизации (вариант 5)

Студент _____ **Тетерин Н.Е.**
подпись, дата *фамилия, и.о.*

Преподаватель _____ **Соколов А.П.**
подпись, дата *фамилия, и.о.*

Оценка _____

Москва, 2021 г.

Оглавление

Задание на лабораторную работу

Реализовать алгоритмы для приближенного вычисления интеграла методами составных формул Симпсона и трапеций, воспользоваться ими для определения полного времени движения точки по кривой наискорейшего спуска. Воспользовавшись кусочно-линейной интерполяцией, решить задачу аппроксимации этой кривой.

Цель выполнения лабораторной работы

Цель выполнения лабораторной работы – имплементировать алгоритмы составных формул трапеции и Симпсона и применить их для приближения функционала полного времени движения по плоской кривой для брахистохроны.

Выполненные задачи

1. Разработать алгоритм для составной формулы трапеций
2. Разработать алгоритм для составной формулы Симпсона
3. Определить константы из граничных условий
4. Определить аналитически значение функционала времени
5. Построение графиков зависимости абсолютной погрешности численного интегрирования от шага интегрирования
6. Анализ графиков погрешности
7. Приближение функционала времени путем кусочно-линейной интерполяции подынтегрального выражения с дальнейшим численным интегрированием и анализ абсолютной погрешности результата с построением полученной поверхности

1. Составные формулы трапеций и Симпсона

Пусть имеется функция $f(x)$ на отрезке $[a; b]$, интеграл $\int_a^b f(x)dx$ которой на данном отрезке требуется определить. Выражения для составных формул трапеций и Симпсона получаются путем разбиения этого отрезка на *чётное* число n подотрезков с последующим применением соответствующей формулы на каждом подотрезке. Тогда для $f \in C^4[a; b]$, $\mathrm{h} = (b-a)/n$, $x_i = a + (i-1)h$, где $i = 1, \dots, n+1$ можно получить составную формулу Симпсона (1.1) и трапеций (1.2). Стоит отдельно отметить, что составная формула трапеций *нечувствительна к четности* числа элементов: интуитивно это объясняется тем, что для каждого применения формулы Симпсона требуется 3 точки и, соответственно, 2 подотрезка, в то время как для применения формулы трапеций только 2 точки и один подотрезок соответственно (так, при четном числе подотрезков или, что то же самое, нечетном числе узлов не получится записать исходный интеграл в виде суммы интегралов на подотрезках). Говоря более строго, записать исходный интеграл в виде суммы интегралов на подотрезках необходимо для дальнейшего применения *первой теоремы о промежуточном значении* для оценки остаточного члена формулы.

$$(1.1)\quad \int_a^b f(x)dx = \frac{h}{3}\left[f(x_1) + 2\sum_{i=1}^{n/2-1} f(x_{2i+1}) + 4\sum_{i=1}^{n/2} f(x_{2i}) + f(x_{n+1})\right] - \frac{(b-a)h^4}{180} f^{(4)}(\xi)$$

$$(1.2)\quad \int_a^b f(x)dx = \frac{h}{2}\left[f(x_1) + 2\sum_{i=2} n f(x_i) + f(x_{n+1})\right] - \frac{(b-a)h^2}{12} f^{(2)}(\xi),\ \ \xi \in [a; b]$$

Ещё один важный момент заключается в *порядке точности* этих формул. Вообще говоря, выражение для остаточного члена является справедливым только для достаточно гладких функций (имеющих 4 непрерывные производные в случае формулы Симпсона). Этот факт стоит помнить, поскольку, как будет показано далее, фактический порядок точности примененных формул в данной задаче не будет совпадать с аналитическим.

2. Алгоритмы для составных формул численного интегрирования

Исходный код для данной ЛР был написан на ЯП *Python* и в тексте данного отчета будут периодически появляться листинги имплементированных методов. *Python* предоставляет простой интерфейс для работы с коллекциями и разными типами данных, при этом такие библиотеки, как *numpy*, *scipy*, *matplotlib* и *plotly* помогают легко выстроить этапы разработки и создать репрезентативные иллюстрации.

Исходный код алгоритмов данного раздела находится в модулях *simpson.py* и *trapezoid.py*, он немногословен, поэтому представлен на листинге 2.1. Поскольку функции в python являются объектами, не составляет труда передать интегрируемую функцию в качестве аргумента. Далее необходимо сформировать коллекцию из указанного числа узлов и, собственно, воспользоваться выражением для квадратуры, просуммировав нужное число раз значение функции в данных узлах. Разумеется, выражение для остаточного члена здесь отсутствует, его величина будет оценена далее.

simpson.py

```python
import numpy as np

def composite_simpson(a: float, b: float, n: int, f) -> float:
    if a > b: raise ValueError
    if n < 3: raise ValueError
    if f is None: raise ValueError

    if n % 2 == 0: n -= 1

    h = (b - a) / (n - 1)
    x_range = np.linspace(a, b, n)

    def sum_terms():
        yield x_range[0]
        for i in range(1, n - 1):
            if i % 2 == 0:
                yield x_range[i]
                yield x_range[i]
            if i % 2 == 1:
                yield x_range[i]
                yield x_range[i]
                yield x_range[i]
                yield x_range[i]
        yield x_range[-1]

    approximation = (h / 3) * sum([f(term) for term in sum_terms()])
    return approximation
```

trapezoid.py

```python
import numpy as np

def composite_trapezoid(a: float, b: float, n: int, f) -> float:
    if a > b: raise ValueError
    if n < 2: raise ValueError
    if f is None: raise ValueError

    x_range = np.linspace(a, b, n)
    h = (b - a) / (n - 1)

    def sum_terms():
        yield x_range[0]
        for i, item in enumerate(x_range[1:-1], 1):
            yield item
            yield item
        yield x_range[-1]

    approximation = (h / 2) * sum([f(term) for term in sum_terms()])
    return approximation
```

Листинг 2.1. Функции для численного интегрирования для составных формул Симпсона и трапеций

В данной реализации используется генератор (функция *sum_terms*), который позволяет уменьшить потребление памяти и улучшить читаемость кода. Первоначальная проверка

переданных аргументов бросает исключение *ValueError* в случае некорректно заданных интервала интегрирования, числа узлов и подынтегральной функции. Сущность квадратуры заключена в функции *sum*. Также стоит обратить внимание на строку 8: вообще говоря, составная формула Симпсона не работает для четных n. Но, если вдруг в функцию было передано четное число узлов (такое происходило, например, во время численного интегрирования для коллекции значений n, заданных в виде равноотстоящих в логарифмическом пространстве чисел, *np.logspace* или *np.geomspace*), оно уменьшается на 1 (однако возможен вариант и увеличения на 1), это можно будет заметить, например, на рис. 4.1.

График абсолютной ошибки для данных функций показан на рис. 2.1. В качестве подынтегрального выражения для данного теста была выбрана экспонента, поскольку она является бесконечно гладкой. Можно заключить, что фактический порядок точности совпадает с аналитическим (4 для составной формулы Симпсона и 2 для составной формулы трапеций), то есть методы реализованы правильно.

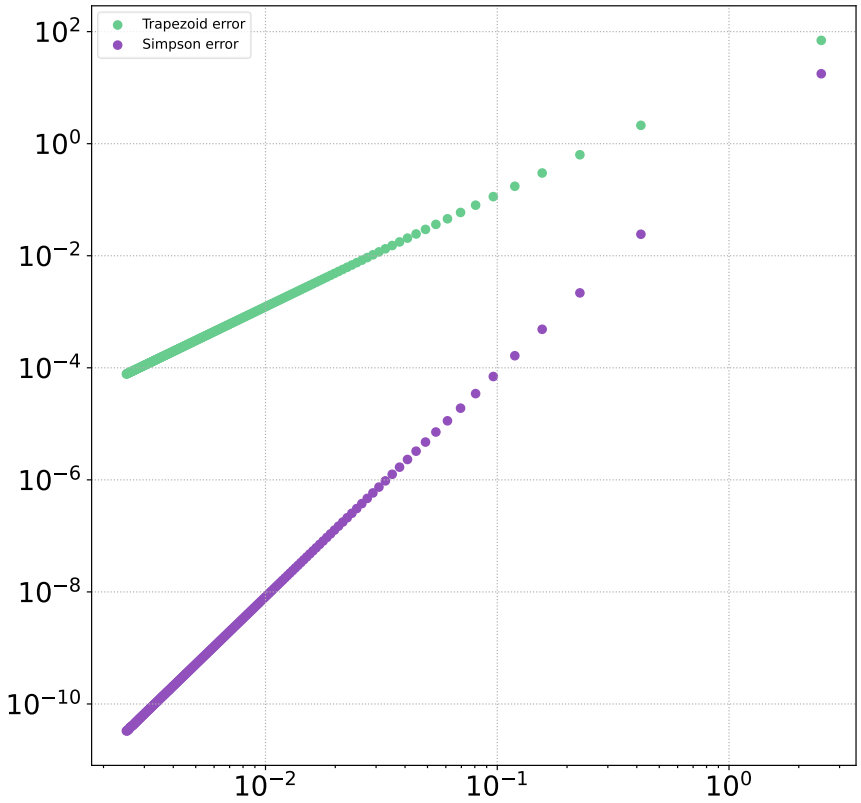

Рис. 2.1. Зависимость абсолютной погрешности численного интегрирования экспоненты для составных формул трапеций и Симпсона

Порядок точности на log-log графике можно определить как тангенс угла наклона графика ошибки. Так, например, при уменьшении величины шага на 1 порядок абсолютная погрешность составной формулы Симпсона уменьшилась на 4 порядка. Аналогично рассчитанный для составной формулы трапеций порядок точности составляет 2.

3. Определение констант из граничного условия

По условию задачи требуется рассчитать значение функционала времени (3.1), соответствующего полному времени движения точки по кривой, на которую наложены граничные условия: начало кривой соответствует началу координат, а конец – точке с координатами $(x, y) = (a, y_a)$. В рамках данной задачи ось ординат направлена вертикально вниз, $a = 2, y_a = 1$.

$$\mathcal{F}[y] = \int_0^a \sqrt{\frac{1 + \left(y'(x)\right)^2}{2gy(x)}}\,dx \tag{3.1}$$

$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \frac{C}{2}\begin{bmatrix} 2t - sin(2t) \\ 1 - cos(2t) \end{bmatrix} \tag{3.2}$$

Методами вариационного исчисления (а именно путем применения *теоремы Эйлера-Лагранжа*) можно показать, что экстремуму (минимуму) данного функционала соответствует параметрическим образом заданная функция (3.2), которая описывает *циклоиду*, соединяющую две описанные выше точки, при этом в начале координат она перпендикулярна оси ординат (производная стремится к $+\infty$). При этом $t \in [0; T]$, а константы C, T находятся из граничного условия (координат конечной точки). Проще всего сделать это численно, в данном случае использовался метод *scipy.optimize.root* библиотеки *scipy*. На вход ему необходимо подать вектор-функцию, корень которой требуется найти, и вектор с начальным приближением решения (вектор здесь содержит 2 компоненты, C и T. Исходный код функции, вычисляющей константы из граничного условия, находится на листинге 3.1.

optimizer.py

```python
from scipy.optimize import root
import numpy as np

def find_constants(endpoint: tuple):
    xa, ya = endpoint

    def boundary_conds(interest):
        C, T = interest

        f = [
            C * (T - 0.5 * np.sin(2*T)) - xa,
            C * (0.5 - 0.5 * np.cos(2*T)) - ya
        ]
        return f

    solution = root(boundary_conds, [0.9, 1.0])
    return solution.x

```

Листинг 3.1.
Функция для определения констант циклоиды из граничного условия

Таким образом получаем:

$$C = 1.034399843373137, \qquad T = 1.754184384262122$$

4. Аппроксимация функционала времени

Для применения составных формул трапеций и Симпсона требуется иметь набор равноотстоящих узлов на оси переменной интегрирования. Ключевая проблема данной задачи состоит в переходе от интегрирования зависимости $y(x)$ к параметризованным функциям $y(t)$ и $x(t)$ и интегрировании по параметру. Для этого проведем некоторые алгебраические преобразования над (3.2). В первую очередь получим выражения для производных y'_t, x'_t:

$$(4.1) \quad y'_t = C/2 \cdot [1 - cos(2t)]' = Csin(2t)dt$$
$$(4.2) \quad x'_t = \mathrm{C}/2 \cdot [2t - sin(2t)]' = \mathrm{C}[1 - cos(2t)]\mathrm{dt}$$

Из них имеем:

$$(4.3) \quad y'(x) = \frac{dy}{dx} = \frac{dy}{dt} \cdot \frac{dt}{dx} =$$
$$= \frac{Csin(2t)}{C[1 - cos(2t)]} = \frac{sin(2t)}{1 - cos(2t)}$$

$$(4.4) \quad \mathcal{F}[y] = \frac{1}{\sqrt{2g}} \int_0^{t_a} \sqrt{\frac{1 + \left(y'(x)\right)^2}{\frac{1}{2}C[1 - cos(2t)]}} C[1 - cos(2t)]dt =$$
$$= \frac{1}{\sqrt{2g}} \int_0^{t_a} \sqrt{\frac{1 - 2cos(2t) + cos^2(2t) + sin^2(2t)}{1 - cos(2t)} 2C}\, dt =$$
$$= \frac{1}{\sqrt{2g}} \int_0^{t_a} \sqrt{\frac{2(1 - cos(2t)}{1 - cos(2t)} 2C}\, dt = \sqrt{\frac{2C}{g}} \int_0^{t_a} dt$$

При этом верхний предел интегрирования по t соответствует T, а нижний 0. Таким образом, аналитически рассчитанное значение функционала (3.1) равно

$$(4.5) \quad \mathcal{F}[y_{br}] = \sqrt{\frac{2C}{g}}(T - 0) = \sqrt{\frac{2C}{g}} T$$

В дальнейшем при определении абсолютной погрешности численного интегрирования в качестве выражения для точного значения будет использоваться (4.5) (с поправкой на нижний предел интегрирования, это будет оговорено отдельно).

Требуется произвести численное интегрирование для значений $n \in [3; 9999]$. Для этого в модуле *error.py* было имплементированы классы *BrachistochroneErrorComputer* и *BrachistochroneNodeProvider*. Первый получает такие данные для моделирования, как верхняя и нижняя границы множества значений n, координаты конечной точки (a, y_a) (для определения констант), с помощью методов последнего получает значения констант и функцию для подынтегрального выражения относительно t. Далее для неё производится численное интегрирование. Результат сравнивается с точным значением и сохраняется для последующего отображения на графике. Первоначально процесс несколько отличался: сперва формировалась коллекция узлов параметра t, на её основе формировались коллекции той же размерности для значений функций (3.2), (4.2-3), на их основе уже набор значений подынтегрального выражения из (3.1). Далее производился выбор необходимого числа узлов параметра и подынтегрального выражения и в таком виде передавался в функцию численного интегрирования, в которой уже

производилось суммирование необходимых значений. Впоследствии архитектура была доработана: лишние наборы узлов не генерировались, а побочные вариации функций численного интегрирования удалены. Всё это было возможно благодаря тому, что *python* поддерживает концепцию функций первого класса, и, в частности, они могут быть замыканиями (closures). *Замыкание* – это понятие функционального программирования, используемое в контексте вложенных функций. Функция может быть названа замыканием, если она имеет доступ к т. н. *enclosing scope*, то есть контексту, в котором она объявлена (Разрешение имен в python происходит по принципу LEGB: Local-Enclosing-Global-BuiltIn). В контексте данной задачи это свойство может оказаться полезным по той причине, что функция подынтегрального выражения может быть сформирована с использованием нескольких вспомогательных функций и впоследствии передана в качестве параметра в функцию численного интегрирования. Исходный код фрагментов, задействующих данное замечательное свойство, представлен на листинге 4.1.

```
122
123 class BrachistochroneNodeProvider(ABC):
124
125     @staticmethod
126     def get_constants(x_a, y_a):
127         log.debug(msg=f'Computes arbitrary constants of model (C and T)')
128         log.debug(msg=f'The endpoint is ({x_a},{y_a})')
129
130         C, T = find_constants((x_a, y_a))
131         log.info(msg=f'Found constants: C = {C}, T = {T}')
132         return C, T
133
134     @staticmethod
135     def get_parametrized_funcs(C):
136         # the funcs tuple contains x(t), y(t), y'(x), x'(t) expressions
137         # these are required to compute the integrand value for given t
138         log.debug(msg=f'Sets functions for x and y for parameter t')
139         funcs = (
140             lambda t: C * (t - 0.5 * np.sin(2*t)),
141             lambda t: C * (0.5 - 0.5 * np.cos(2*t)),
142             lambda t: np.sin(2*t) / (1 - np.cos(2*t)),
143             lambda t: C * (1 - np.cos(2*t))
144         )
145         return funcs
146
147     @staticmethod
148     def get_integrand_func(C):
149         _, yt, ydx, xdt = BrachistochroneNodeProvider.get_parametrized_funcs(C=C)
150         integrand_f = lambda t: np.sqrt((1 + ydx(t)**2) / yt(t)) * xdt(t)
151         return integrand_f
```

Листинг 4.1. Методы get_parametrized_funcs и get_integrand_func, задействующие концепции замыканий и лямбда-выражений

Отметим также такой факт, что подынтегральное выражение суть есть тождественная константа, что неудивительно, ведь циклоида удовлетворяет условию, исходящему из физического смысла кривой наискорейшего спуска (принципа Ферма для движения света в неоднородной среде), а именно

$$\frac{v}{sin(\alpha)} = const$$

Здесь v обозначает скорость света в данный момент времени, а α – угол, который составляет касательная (направление мгновенной скорости) с нормалью (осью ординат) в этот момент времени.

В ходе расчетов было выявлено, что не представляется возможным производить интегрирование на всём интервале $[0; T]$, поскольку в этом случае придется столкнуться с

неприятной мелочью: *производная функции* $y(x)$, присутствующая в подынтегральном выражении, *не определена в 0*, а значение правого предела составляет $+\infty$, в результате чего применять численное интегрирование нельзя. По этой причине в качестве нижней границы интервала применения составных формул численного интегрирования принимались различные значения от 1e-7 до 1e-4. На рис. 4.1 представлены графики абсолютной погрешности аппроксимации для разных типов данных для случая, когда аналитическое значение рассчитывалось на всём интервале. Можно наблюдать, что присутствует неустранимая погрешность порядка 1e-8 (нижняя граница в этом случае принималась 1e-7). Она несколько выше для случая одинарной точности, чем для двойной. В случае вычислений с половинной точностью размер сетки оказался слишком маленьким, чтобы вместить все значащие разряды, что приводило к *underflow*. При вычислениях задействующих float128 отчетливо прослеживается неустранимая погрешность, связанная с различными отрезками аналитического и численного интегрирования. В дальнейшем при вычислении точного значения использовался тот же диапазон, в результате чего получилась зависимость, представленная на рис. 4.2.

Обратим внимание на порядок точности методов: судя по графику, он *одинаковый для обеих составных формул* и *равен 1*. Почему так получается*?* Судя по всему, вследствие *гладкости* полученной функции: экспонента, чей интеграл рассчитывался численно в п.2, *бесконечно гладкая*, в то время как полученная циклоида только *непрерывная*, что делает формулу (1.1) нерабочей, а остаточный член $\propto O(h)$, поскольку у составной формулы Симпсона порядок точности на 1 ниже, чем у стандартной. То же происходит и в случае составной формулы трапеций, необходимым условием для применения первой теоремы о среднем значении, произведенного в процессе её вывода, является дважды дифференцируемость. Можно резюмировать, что применение таких методов численного дифференцирования оказывается на настолько удачным в подобных случаях недостаточно гладких функций. Если бы подынтегральная функция имела 2 непрерывные производные, последовательным решением было бы применение именно составной формулы трапеций (или средних), поскольку её вычисления несколько проще, чем в составной формуле Симпсона, при одинаковом итоговом порядке точности.

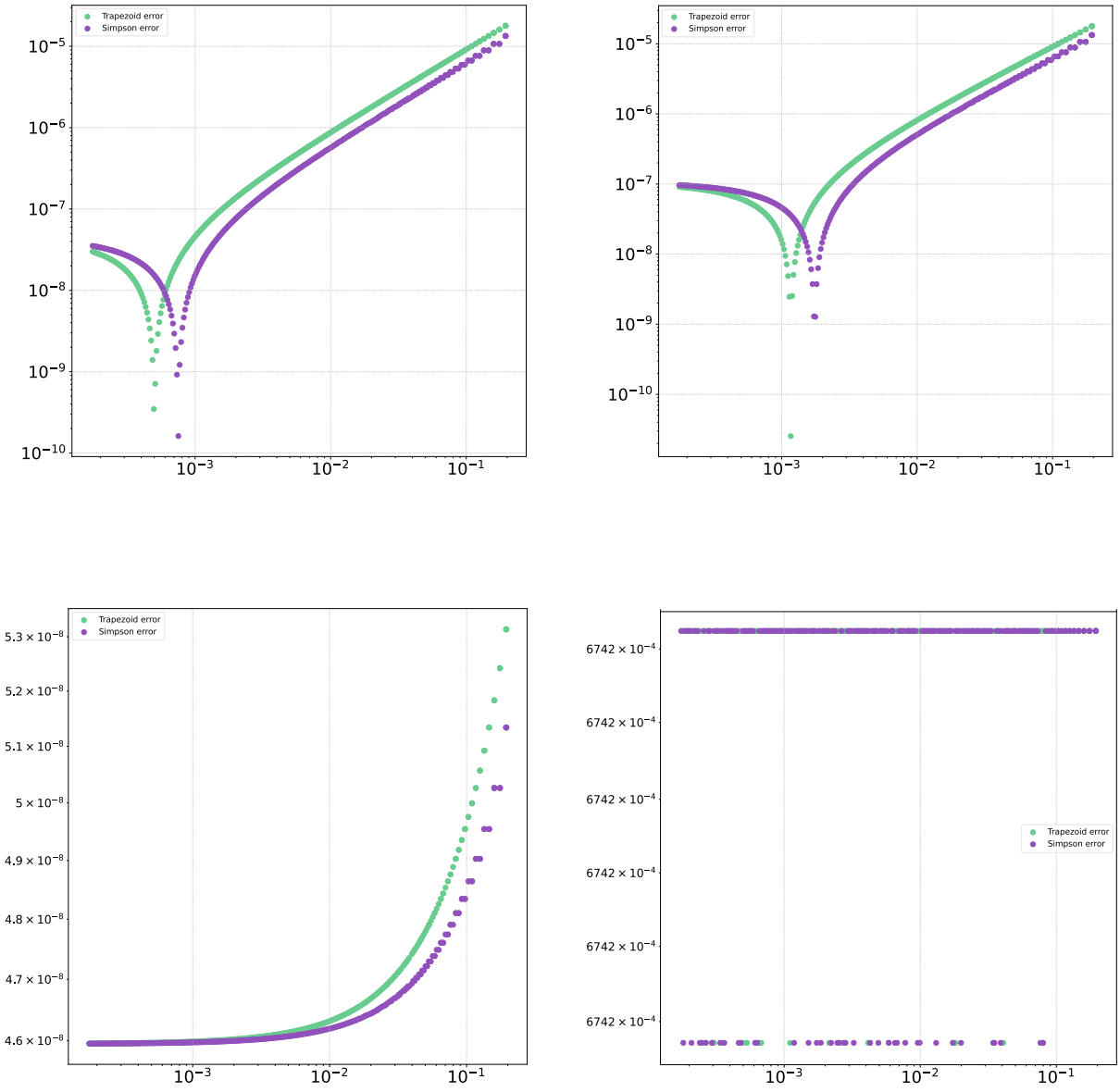

Рис 4.1. Ошибка численного интегрирования для различных типов данных (различные диапазоны для численного и аналитического интегрирования). Первая строка, слева направо: np.float64, float32; вторая строка, слева направо: float128, float16; фиолетовые круги соответствуют составной формуле Симпсона, зеленые – составной формуле трапеций.

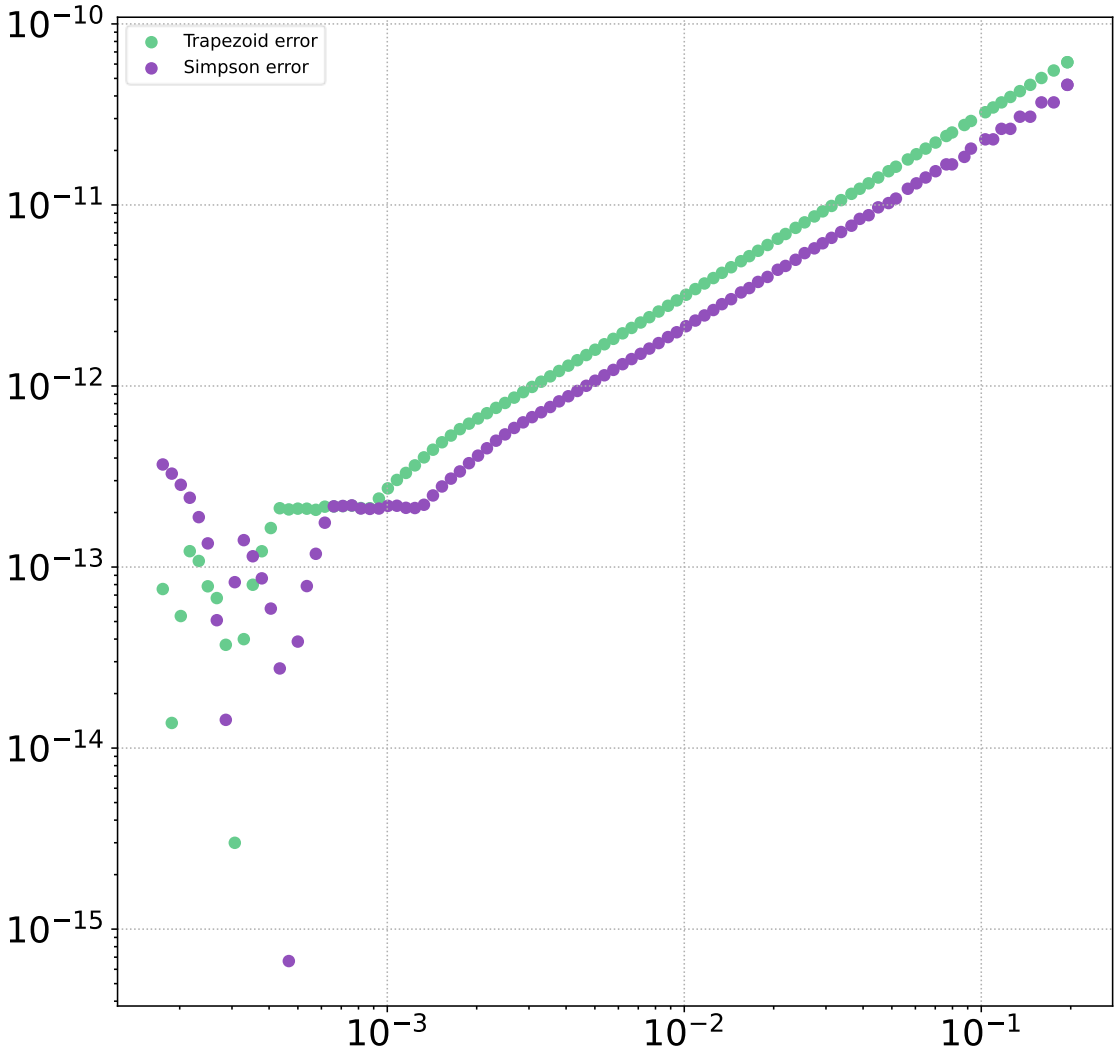

Рис. 4.2. Зависимость абсолютной погрешности аппроксимации для вычислений двойной точности (np.float64) и одинаковом интервале аналитического и численного интегрирования (для t в диапазоне от 1e-4 до T). Наблюдается одинаковый порядок точности, равный 1.

К вопросу об оптимальном шаге: численное интегрирование является вычислительно устойчивым, поэтому уменьшать шаг можно, пока не произойдет преобладание ошибок округления, связанных с машинной арифметикой. В рамках данной задачи такой оптимальный шаг составляет порядка 1e-3 для *float64* (см. рис. 4.2). По рис. 4.1 можно подумать, что далее происходит нелинейное уменьшение погрешности, однако это может свидетельствовать о преобладании ошибок округления, и, например, при изменении ГУ привести к большей погрешности.

5. Аппроксимация кривой наискорейшего спуска

В данной части произошло неприятное недопонимание условия задачи: было сделано ложное предположение, что аппроксимация предполагает кусочно-линейную интерполяцию зависимости (3.2) с последующим анализом зависимости абсолютной ошибки численного интегрирования от собственно шага интегрирования. Вообще говоря, какой смысл в подобном эксперименте, когда мы заранее знаем форму кривой? Тем не менее, результаты наблюдений будут представлены, и они будут вполне ожидаемыми. Но сперва проведем более строгий вывод системы уравнений, задающих условие минимизации функционала (3.1) при условии, что отрезок оси независимой переменной в нем, соответствующий ГУ, разделен на n равных подотрезков, на каждом из которых искомая функция заменяется на линейную. Другими словами, для (3.1) требуется найти y^* такую, что:

$$(5.1) \quad y^* = \mathrm{a}r\underset{y}{g\min}\,\mathcal{F}[y]$$

при этом введем следующую кусочно-заданную функцию $S(x)$ (кусочно-линейную аппроксимацию):

$$(5.2) \quad S_i(x) = \alpha_i + \beta_i x, \qquad x \in [x_i; x_{i+1}], i \in [1; n-1]$$

Таким образом, общее число неизвестных параметров составляет $2(n-1)$, и (3.1) можно переписать в следующем виде:

$$(5.3) \quad \mathcal{F}[y^*] \approx \frac{1}{\sqrt{2g}} \sum_{i=1}^{n-1} \int_{x_i}^{x_{i+1}} \sqrt{\frac{1+(\beta_i)^2}{\alpha_i + \beta_i x}}\, dx =$$

$$= \sum_{i=1}^{n-1} \left(\frac{1+\beta_i^2}{2g}\right)^{1/2} \int_{x_i}^{x_{i+1}} \frac{dx}{\sqrt{\alpha_i + \beta_i x}} = H(\alpha, \beta)$$

Здесь аргументами вектор-функции H являются вектора, образованные всеми значениями весов кусочно-линейной функции S. Условие минимума тогда можно записать в следующем виде, с учетом условия сопрягаемости соседних элементов S:

$$(5.4) \quad \begin{cases} \dfrac{\partial H}{\partial \alpha_i} = 0 \\ \dfrac{\partial H}{\partial \beta_i} = 0 \\ S_1(0) = 0 \\ S_{n-1}(a) = y_a \\ S_i(x_{i+1}) = S_{i+1}(x_{i+1}) \Rightarrow \alpha_i + \beta_i x_{i+1} = \alpha_{i+1} + \beta_{i+1} x_{i+1} \end{cases} \qquad (i = 1, \dots, n-1)$$

Произведем вывод выражений для частных производных в системе (5.4):

$$(5.5)\quad \frac{\partial H}{\partial \alpha_p} = \sum_{i=1}^{n-1}\left(\frac{1+\beta_i^2}{2g}\right)^{1/2}\int_{x_i}^{x_{i+1}}\frac{\partial}{\partial \alpha_p}\frac{dx}{\sqrt{\alpha_i+\beta_i x}} = \sum_{i=1}^{n-1}\frac{\partial J_i}{\partial \alpha_p} \Rightarrow \left\{\frac{\partial J_i}{\partial \alpha_p} \neq 0 \Leftrightarrow p=i\right\} \Rightarrow$$

$$\Rightarrow -\frac{1}{2}\left(\frac{1+\beta_p^2}{2g}\right)^{1/2}\int_{x_p}^{x_{p+1}}\left(\alpha_p+\beta_p x\right)^{(-3/2)}dx$$

$$(5.6)\quad \frac{\partial H}{\partial \beta_p} = \sum_{i=1}^{n-1}\frac{\partial}{\partial \beta_p}\left[\left(\frac{1+\beta_i^2}{2g}\right)^{1/2}\int_{x_i}^{x_{i+1}}\frac{dx}{\sqrt{\alpha_i+\beta_i x}}\right] = \sum_{i=1}^{n-1}\frac{\partial J_i}{\partial \beta_p} \Rightarrow \left\{\frac{\partial J_i}{\partial \beta_p} \neq 0 \Leftrightarrow p=i\right\} \Rightarrow$$

$$\Rightarrow \frac{\partial H}{\partial \beta_p} = \frac{\partial}{\partial \beta_p}\left[\left(\frac{1+\beta_p^2}{2g}\right)^{1/2}\cdot\int_{x_p}^{x_{p+1}}\frac{dx}{\sqrt{\alpha_p+\beta_p x}}\right] = \int_{x_p}^{x_{p+1}}\left[\frac{\partial}{\partial \beta_p}\left(\frac{1+\beta_p^2}{(\alpha_p+\beta_p x)2g}\right)^{1/2}dx\right] =$$

$$= \frac{1}{2}\int_{x_p}^{x_{p+1}}\left(\frac{1+\beta_p^2}{(\alpha_p+\beta_p x)2g}\right)^{-1/2}\frac{2\beta_p 2g(\alpha_p+\beta_p x)-2gx(1+\beta_p^2)}{4g^2(\alpha_p+\beta_p x)^2}dx =$$

$$= \frac{1}{2}\int_{x_p}^{x_{p+1}}\left(\frac{1+\beta_p^2}{(\alpha_p+\beta_p x)2g}\right)^{-1/2}\frac{(2\beta_p\alpha_p+2\beta_p^2 x-x-x\beta_p^2)}{2g(\alpha_p+\beta_p x)^2}dx =$$

$$= \frac{1}{2}\int_{x_p}^{x_{p+1}}\left[\frac{\alpha_p+\beta_p x}{(1+\beta_p^2)2g}\right]^{1/2}\frac{2\beta_p\alpha_p+\beta_p^2 x-x}{(\alpha_p+\beta_p x)^2}dx$$

Перепишем (5.4) с учетом полученных выражений, опустив ненулевые константные члены (так, множитель перед знаком интеграла, содержащий β_p в частной производной по α_p можно опустить, поскольку $\beta_p \in \mathbb{R}$):

$$(5.7)\quad \begin{cases} \dfrac{\partial H}{\partial \alpha_i} = \displaystyle\int_{x_i}^{x_{i+1}}(\alpha_i+\beta_i x)^{(-3/2)}dx = 0 \\ \dfrac{\partial H}{\partial \beta_i} = \displaystyle\int_{x_i}^{x_{i+1}}\left[\frac{\alpha_i+\beta_i x}{(1+\beta_i^2)2g}\right]^{1/2}\frac{2\beta_i\alpha_i+\beta_i^2 x-x}{(\alpha_i+\beta_i x)^2}dx = 0 \quad (i=1,\dots,n-1) \\ \alpha_1 = 0 \\ S_{n-1}(a) = y_a \\ \alpha_i+\beta_i x_{i+1} = \alpha_{i+1}+\beta_{i+1}x_{i+1} \end{cases}$$

Проблема заключается в том, что решить данную систему не удалось (а именно, записать условие нулей частных производных: после аналитического вычисления интегралов оказывалось, что для выполнения равенства должно выполняться $x_i = x_{i+1}$!). Тем не менее, её вывод был помещен в этот раздел чтобы, возможно, можно было найти ошибки в постановке задачи и преобразованиях. Не исключено, что при вычислении частных производных была допущена неприятная арифметическая ошибка.

Теперь будут описаны результаты вычислительного эксперимента, во время которого на кусочно-линейную функцию была заменена сперва, собственно, брахистохрона для записи $y(x)$, а затем в записи $int(t)$, где int обозначает подынтегральное выражение в (3.1). Первичная проверка работоспособности интерполяции была проведена на функции $x^2 - 5x + 3$, графики расположены на рис. 5.1. Можно наблюдать, что аппроксимация производится корректно. График кривой наискорейшего спуска и её аппроксимации с помощью модели кусочно-линейной интерполяции, а также производных обеих зависимостей представлены на рис. 5.2-4.

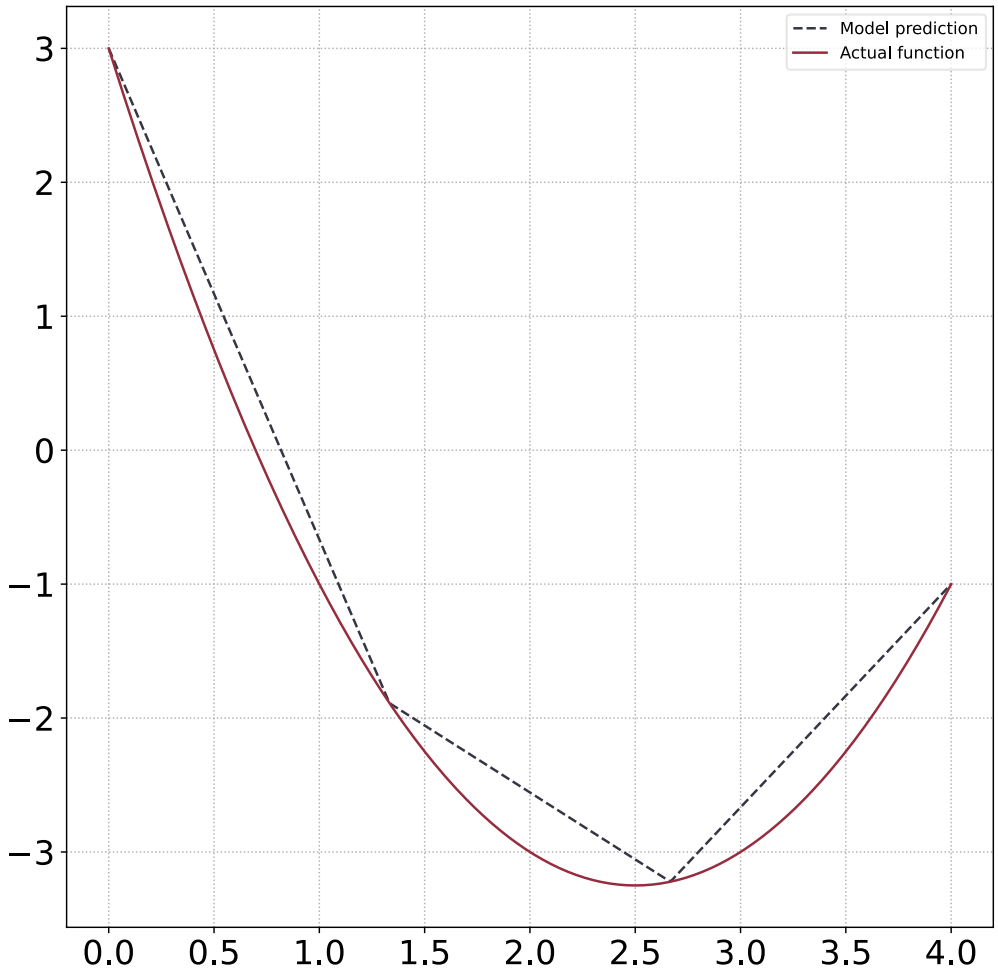

Рис. 5.1. График функции $x^2 - 5x + 3$ и её кусочно-линейной интерполяции по 4 узлам.

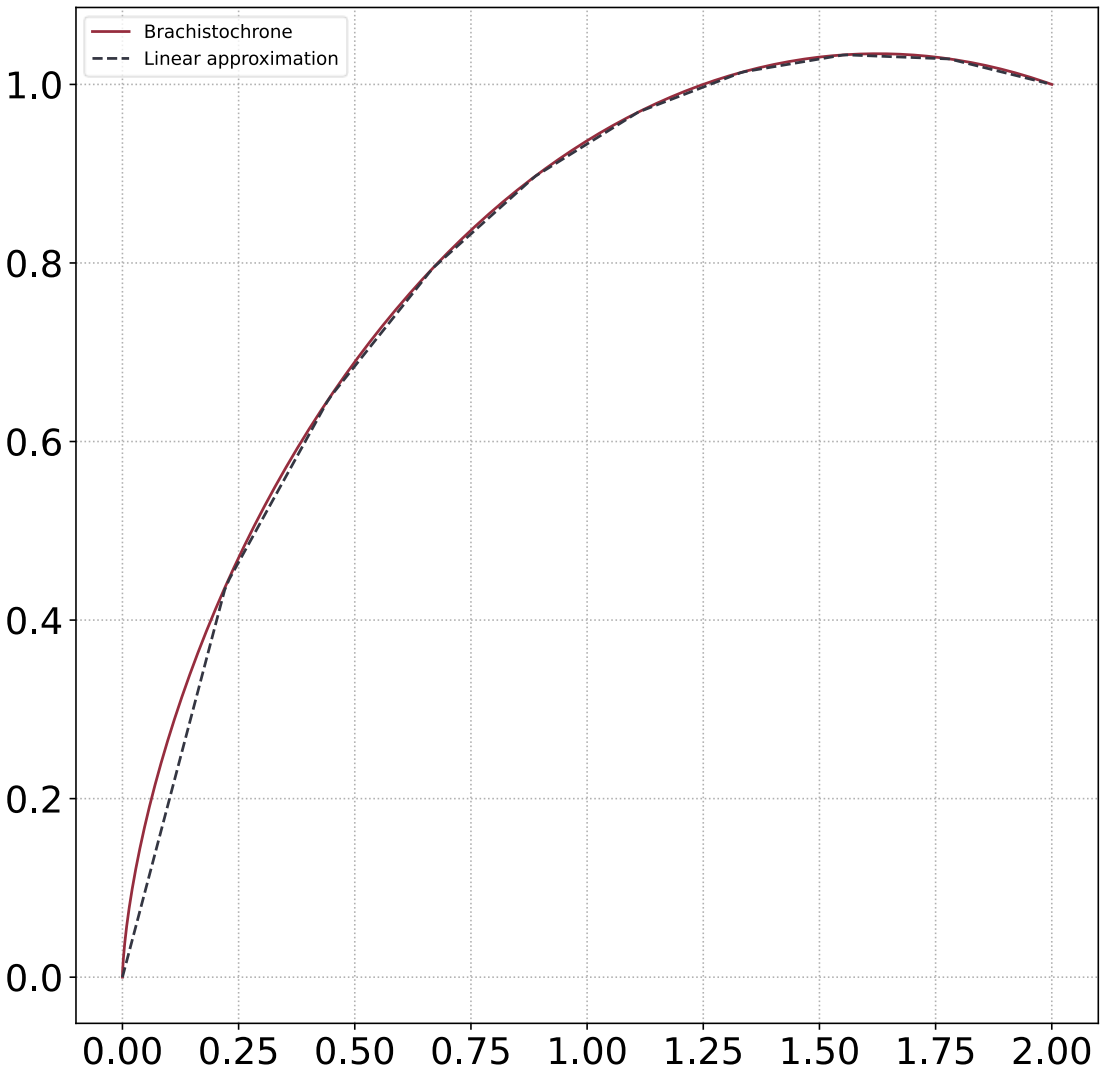

Рис. 5.2. Сравнение графиков кривой наискорейшего спуска и её кусочно-линейной аппроксимации, задействующей 10 узлов.

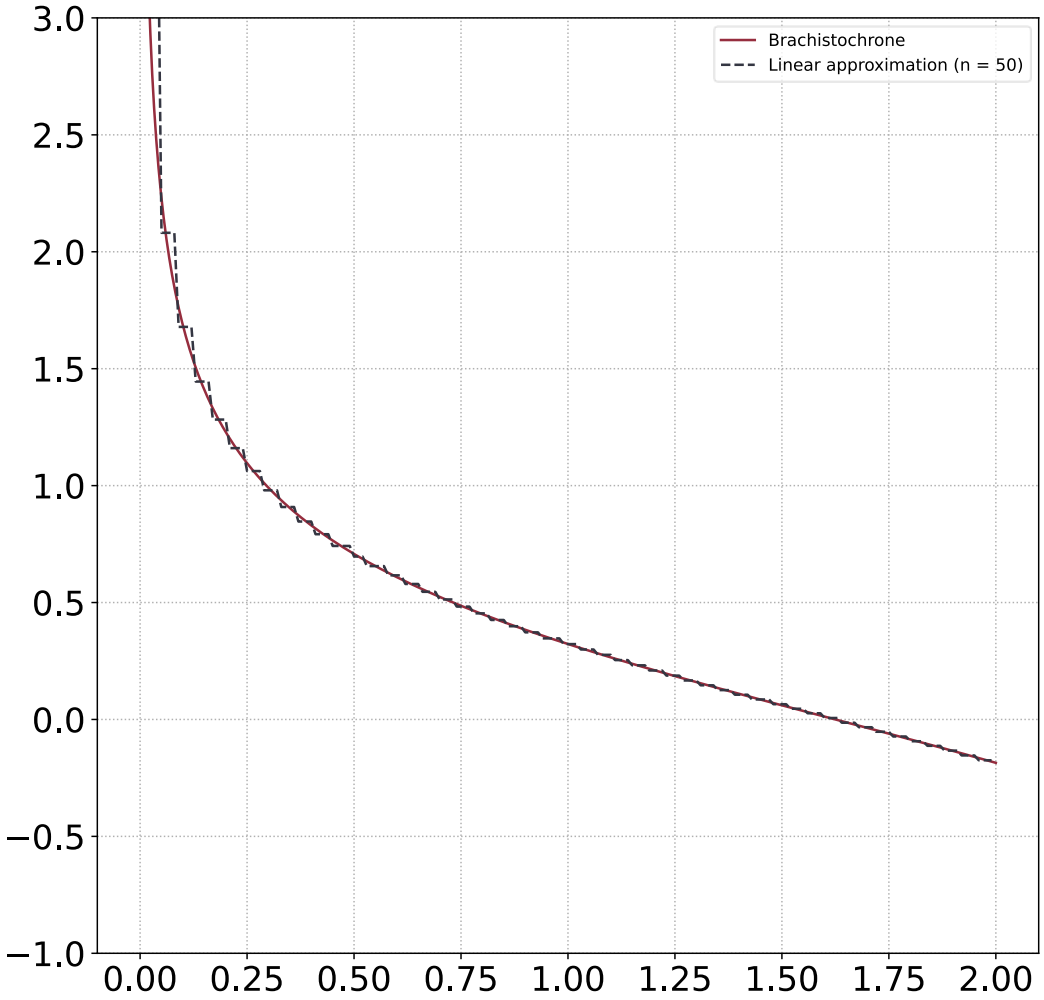

Рис. 5.3. Сравнение производных кривой наискорейшего спуска и её кусочно-линейной аппроксимации, рассчитанной для 50 узлов. Можно наблюдать «ступенчатый» характер производной интерполянта, соответствующий его линейности.

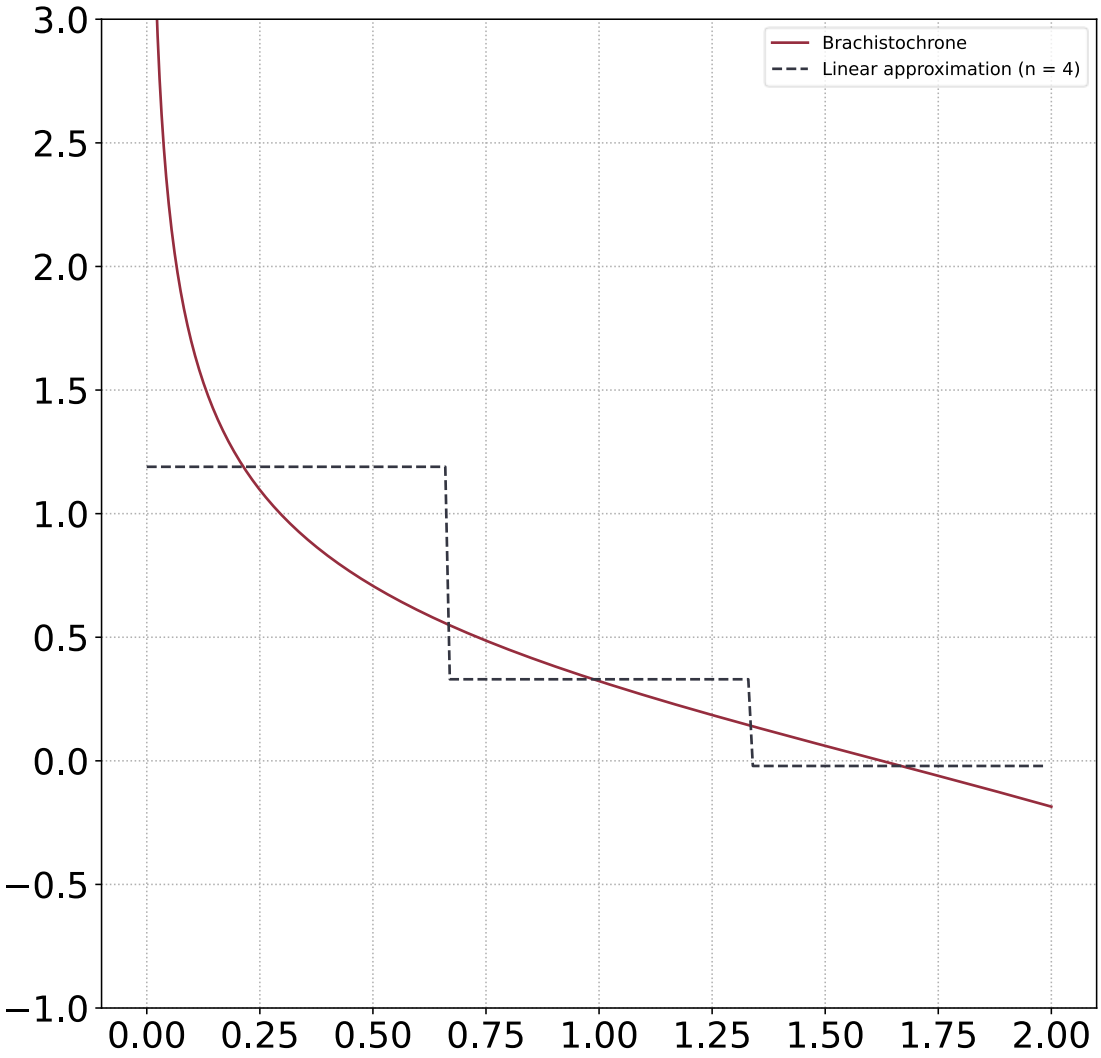

Рис. 5.4. Сравнение производных кривой наискорейшего спуска и её кусочно-линейной аппроксимации, рассчитанной для 4 узлов. При малом количестве последних точность приближения производной страдает из-за малого $(n-1)$ *числа «ступенек».*

Проблема кусочно-линейной интерполяции заключается в том, что в результирующей формуле (3.1) фигурирует *не только сама функция* $y(x)$, но и её *производная*. Здесь следует остановиться подробнее: при кусочно-линейном способе производная интерполянта суть есть «ступенчатая» функция, высота каждого участка соответствует величине производной на этом участке. В случае с кривой наискорейшего спуска возникает проблема: её производная при приближении к нулю справа, как уже было сказано, *стремится к* $+\infty$. При этом при отдалении от 0 по оси абсцисс её величина стремительно *понижается*, пока вовсе не пересечет ось абсцисс. Но производная кусочно-линейной функции имеет *константное значение* на всем

промежутке между двумя первыми узлами! Поэтому при вычислении подынтегральной функции ошибка, накапливаемая в значении $y'(x)$, давала о себе знать очень значительно. От этого неприятного эффекта можно избавиться, *увеличивая нижнюю границу* численного интегрирования, а, следовательно, и вычисления производной, пока нижняя граница по x не составит порядка 1e-2. Причем, при увеличении нижней границы пределов интегрирования на порядок абсолютная погрешность также снижалась примерно на порядок. Стоит понимать важный момент, заключающийся в том, что нижняя граница интегрирования по t в общем случае не равна нижней границе по x! При подобной операции снижается погрешность для вычисленного на том же диапазоне точного значения, но увеличивается неустранимая погрешность для вычисленного на всем интервале точного значения функционала, из чего следует вывод о сомнительности применимости подобного метода. Поверхность абсолютной погрешности для различных конфигураций узлов интегрирования и интерполяции представлена на рис. 5.5-6. Можно наблюдать паразитные осцилляции, природа которых не вполне ясна, их происхождение может быть связано со спецификой расчета производных. После неудовлетворительных результатов этого эксперимента была произведена интерполяция зависимости подынтегрального выражения в (3.1) от параметра циклоиды t. Результат был вполне ожидаемым и представлен на рис. 5.7. Заметно, что наименьшая величина ошибки достигается при равном числе узлов интерполирования и численного интегрирования (в таком случае не происходит вычисления приближенного значения функции вовсе), разумеется, при увеличении их числа порядок погрешности снижается. При уменьшении числа узлов интерполяции при фиксированном числе узлов интегрирования ошибка закономерно растет (интерполянт попросту хуже описывает исходную функцию), но при увеличении немного возрастает и остается примерно постоянной (узлы уже не совпадают, но их становится не меньше, погрешность квадратуры при этом не уменьшается). Зеркальная ситуация образуется при фиксировании числа узлов интерполяции и варьировании числа узлов численного интегрирования: при уменьшении числа последних ошибка так же закономерно растет, поскольку само интегрирование становится менее точным, однако, когда число у. интегрирования превышает число у. интерполяции, ошибка не уменьшается, ведь интерполяция сама вносит неустранимую погрешность в вычисленное значение.

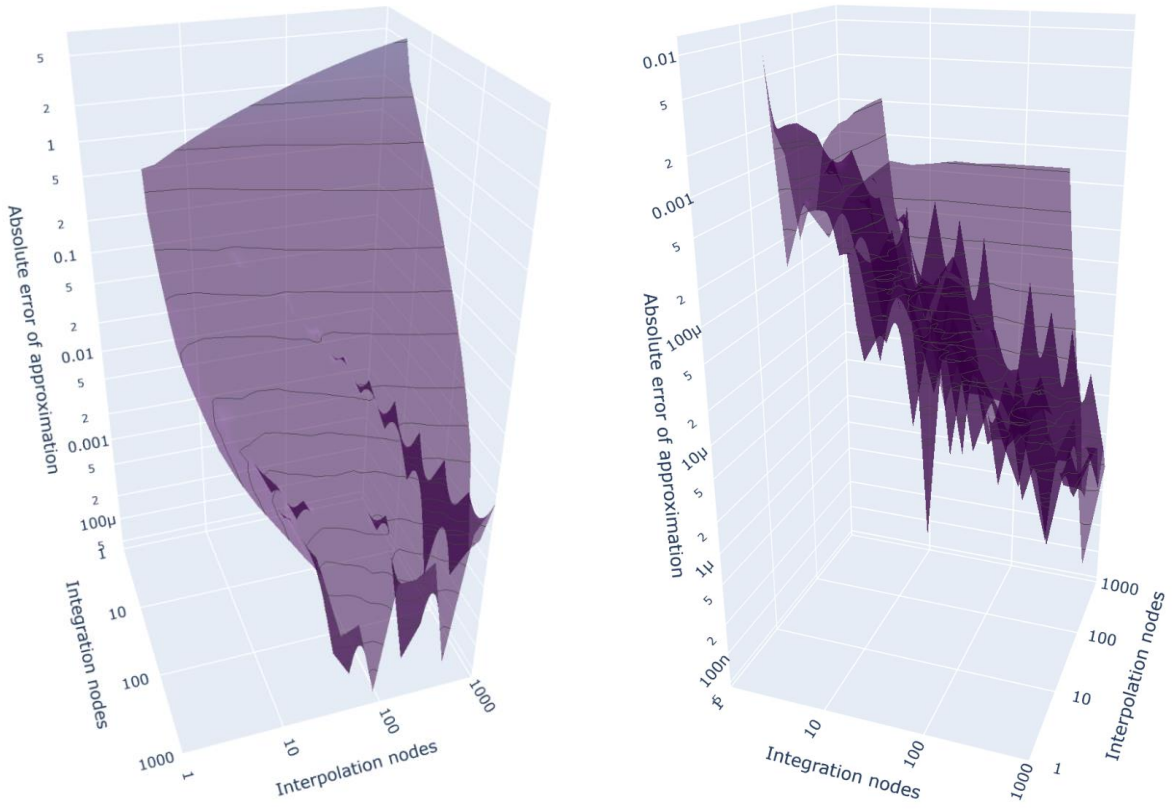

Рис. 5.5. Зависимость абсолютной погрешности аппроксимации функционала (3.1) от числа узлов интегрирования составной формулой Симпсона и кусочно-линейной интерполяции $y(x)$ и нижней границе интервала численного и аналитического интегрирования 1e-1 (слева) и 1e0 (справа). В последнем случае снизился порядок погрешности, но усилились паразитные осцилляции.

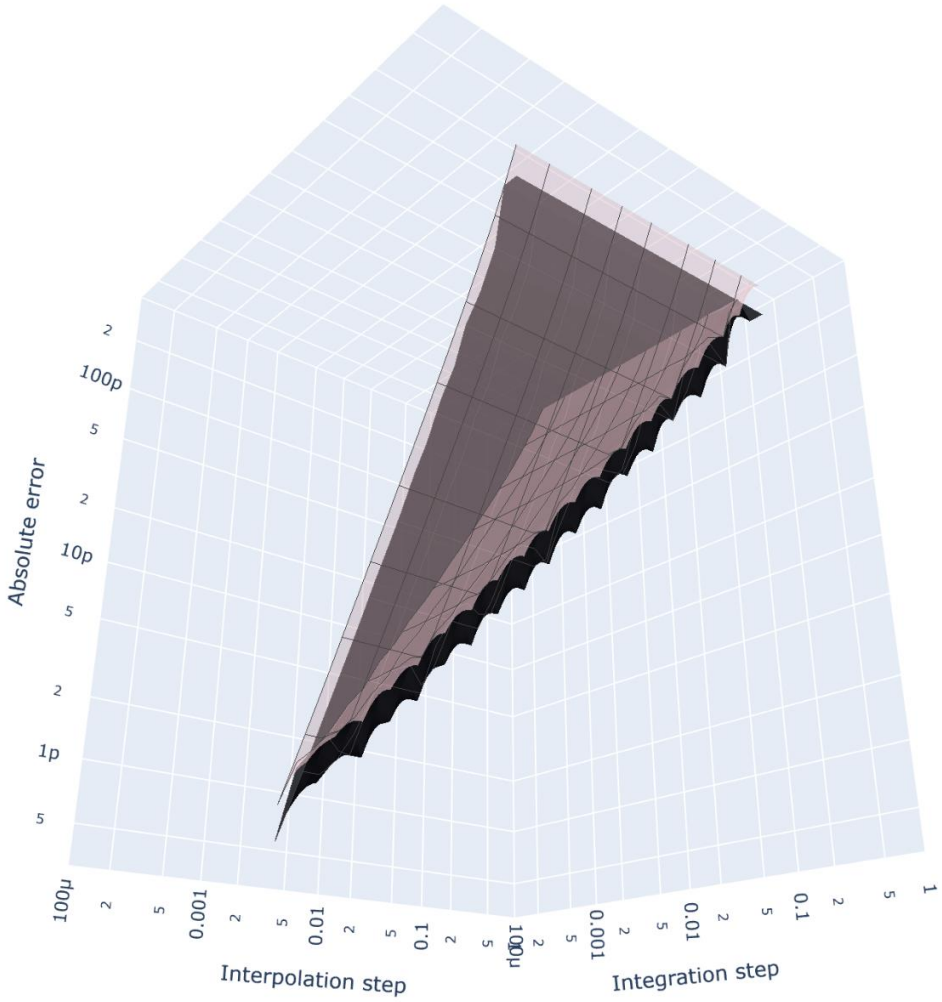

Рис. 5.6. Зависимость абсолютной погрешности аппроксимации функционала (3.1) от шага интерполяции по t и численного интегрирования составными формулами трапеций (светлая поверхность) и Симпсона (темная поверхность).

Заключение

Хочется отметить, что работа получилась многослойная и потребовала задействование знаний из различных областей изученного материала. Первый умозрительный вывод заключался в несовпадении порядков точности составных формул численного интегрирования (трапеции и Симпсона), но было обнаружено, что они не равны аналитическим и составляют 1, что объясняется недостаточной гладкостью интегрируемых функций, что делает формулу для остаточного члена неприменимой. Оптимальный шаг численного интегрирования, однако, согласуется с теорией и определяется из места начала преобладания ошибок машинной арифметики. Что касается продвинутой части данной работы, она производит обманчивое и неоднозначное впечатление, и выполнить её в полном соответствии с требованиями не удалось. На поверку решение задачи аппроксимации кривой наискорейшего спуска оказалось на порядок труднее, чем предполагалось. В любом случае, практика работы с библиотеками *plotly* и *scipy* уже оправдывает приложенные усилия.

Список использованных источников

1. **Соколов А. П., Першин А. Ю.** Инструкция по выполнению лабораторных работ (общая). // кафедра «Системы автоматизированного проектирования» МГТУ им. Н. Э. Баумана, Москва, 2021.
2. **Першин А. Ю.** Лекции по вычислительной математике. // Кафедра РК6 (Системы автоматизированного проектирования) МГТУ им. Н. Э. Баумана, 2020.
3. **Higham, Nicholas J.** Accuracy, and stability of numerical algorithms // University of Manchester, Manchester, England, 2002.
4. Официальная документация к библиотеке визуализации Plotly. Веб-ресурс. // **URL:** https://plotly.com/graphing-libraries/ [Дата обращения: 22.10.2021]
5. 5 levels of understanding closures in Python. Yang Zhou on Medium. Веб-ресурс. // **URL:** https://medium.com/techtofreedom/5-levels-of-understanding-closures-in-python-a0e1212baf6d [Дата обращения: 12.10.2021]