Министерство образования и науки Российской Федерации федеральное государственное
автономное образовательное учреждение высшего образования
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет «Программной инженерии и компьютерной техники.»

Вычислительная математика

Лабораторная работа №3
“Численное интегрирование”
Вариант №3

Выполнил
Григорьев Давид Владимирович
Группа: P3215
Проверила
Малышева Татьяна Алексеевна

Санкт-Петербург 2025

Содержание

1 Цель работы

Целью данной лабораторной работы является приближенное вычисление определенного интеграла

$$\int_0^2 (-x^3 - x^2 + x + 3)\,dx$$

с использованием численных методов (методы прямоугольников, трапеций, Симпсона и формулы Ньютона–Котеса) и сравнение их точности относительно точного значения интеграла. Также задача включает анализ погрешностей вычислений и демонстрацию сходимости методов при заданных параметрах разбиения интервала.

2 Порядок выполнения работы

1. **Точное вычисление интеграла** с использованием формулы Ньютона–Лейбница.
2. **Численное интегрирование** по формуле Ньютона–Котеса при $n = 6$.
3. **Реализация методов**:
   - Метод средних прямоугольников ($n = 10$),
   - Метод трапеций ($n = 10$),
   - Метод Симпсона ($n = 10$).
4. **Сравнение результатов** с точным значением интеграла.
5. **Вычисление относительной погрешности** для каждого метода.
6. **Анализ эффективности методов** на основе полученных данных.

3 Рабочие формулы методов

1. Точное значение интеграла Для функции $f(x) = -x^3 - x^2 + x + 3$ первообразная:

$$F(x) = -\frac{x^4}{4} - \frac{x^3}{3} + \frac{x^2}{2} + 3x.$$

Точное значение:

$$\int_0^2 f(x)\,dx = F(2) - F(0) = \frac{4}{3} \approx 1.3333.$$

2. Метод Ньютона–Котеса (Симпсон, $n = 6$) Формула:

$$\int_a^b f(x)\,dx \approx \frac{h}{3}\left[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + 4f(x_5) + f(x_6)\right],$$

где $h = \frac{b-a}{n}$, n — четное число разбиений.

3. Метод средних прямоугольников ($n = 10$) Формула:

$$\int_a^b f(x)\,dx \approx h \cdot \sum_{i=1}^{n} f\left(x_{i-0.5}\right),$$

где $x_{i-0.5}$ — середины подынтервалов.

4. Метод трапеций ($n = 10$) Формула:

$$\int_a^b f(x)\,dx \approx \frac{h}{2}\left[f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n)\right].$$

5. Метод Симпсона ($n = 10$) Формула:

$$\int_a^b f(x)\,dx \approx \frac{h}{3}\left[f(x_0) + 4\sum_{i=1,3,\ldots}^{n-1} f(x_i) + 2\sum_{i=2,4,\ldots}^{n-2} f(x_i) + f(x_n)\right].$$

4 Анализ погрешностей

Для оценки точности вычислений использовалась **относительная погрешность**:

$$\delta = \left|\frac{I_{\text{численный}} - I_{\text{точный}}}{I_{\text{точный}}}\right| \cdot 100\%.$$

Для программной реализации предполагалось применение **правила Рунге**, позволяющего адаптивно уточнять результаты путем сравнения вычислений на разных сетках. Однако в рамках вычислительной части были зафиксированы заданные значения $n = 6$ и $n = 10$.

5 Точное вычисление интеграла

Первообразная функции:

$$F(x) = -\frac{x^4}{4} - \frac{x^3}{3} + \frac{x^2}{2} + 3x$$

Вычисляем значения на границах:

$$F(2) = -\frac{2^4}{4} - \frac{2^3}{3} + \frac{2^2}{2} + 3\cdot 2 = -4 - \frac{8}{3} + 2 + 6 = \frac{4}{3}$$

$$F(0) = 0$$

Точное значение:

$$\int_0^2 (-x^3 - x^2 + x + 3)\,dx = \frac{4}{3} \approx 1.3333$$

6 Метод Ньютона-Котеса ($n = 6$)

Формула Ньютона-Котеса для $n = 6$:

$$\int_a^b f(x)\,dx \approx h\sum_{i=0}^{6} c_i f(x_i),$$

где $h = \frac{b-a}{n} = \frac{2}{6} = \frac{1}{3}$, коэффициенты Котеса:

$$c_0 = c_6 = \frac{41}{840},\quad c_1 = c_5 = \frac{216}{840},\quad c_2 = c_4 = \frac{27}{840},\quad c_3 = \frac{272}{840}.$$

Значения функции:

$$f(0) = 3,\quad f\left(\frac{1}{3}\right) \approx 3.1852,\quad f\left(\frac{2}{3}\right) \approx 3.0259,$$
$$f(1) = 2,\quad f\left(\frac{4}{3}\right) \approx 0.1852,\quad f\left(\frac{5}{3}\right) \approx -2.7407,\quad f(2) = -7.$$

Подстановка коэффициентов и вычисления:

$$\begin{aligned}
I_{\text{Ньютона-Котеса}} &\approx \frac{1}{3}\left[\frac{41}{840}(3 + (-7)) + \frac{216}{840}(3.1852 + (-2.7407)) + \frac{27}{840}(3.0259 + 0.1852) + \frac{272}{840}\cdot 2\right] \\
&\approx \frac{1}{3}\left[\frac{41}{840}(-4) + \frac{216}{840}(0.4445) + \frac{27}{840}(3.2111) + \frac{272}{840}\cdot 2\right] \\
&\approx \frac{1}{3}\left[-0.1952 + 0.1139 + 0.1033 + 0.6476\right] \\
&\approx \frac{1}{3}\cdot 0.6696 \approx 1.3363.
\end{aligned}$$

Относительная погрешность:

$$\delta = \left| \frac{1.3363 - 1.3333}{1.3333} \right| \cdot 100\% \approx 0.225\%.$$

7 Метод средних прямоугольников ($n = 10$)

Шаг $h = 0.2$. Узлы: $x_{0.5} = 0.1, x_{1.5} = 0.3, \dots, x_{9.5} = 1.9$.
Значения функции в серединах интервалов:

$$\begin{aligned}
f(0.1) &\approx 3.089, \\
f(0.3) &\approx 3.183, \\
f(0.5) &\approx 3.125, \\
f(0.7) &\approx 2.867, \\
f(0.9) &\approx 2.361, \\
f(1.1) &\approx 1.559, \\
f(1.3) &\approx 0.413, \\
f(1.5) &\approx -0.125, \\
f(1.7) &\approx -2.103, \\
f(1.9) &\approx -4.569.
\end{aligned}$$

Сумма значений:

$$3.089 + 3.183 + 3.125 + 2.867 + 2.361 + 1.559 + 0.413 - 0.125 - 2.103 - 4.569 \approx 9.8$$

Интеграл:

$$9.8 \cdot 0.2 = 1.96$$

Относительная погрешность:

$$\left| \frac{1.96 - 1.3333}{1.3333} \right| \approx 46.9\%$$

8 Метод трапеций ($n = 10$)

Шаг $h = 0.2$. Узлы: $x_0 = 0, x_1 = 0.2, \dots, x_{10} = 2$.
Значения функции:

$$\begin{aligned}
f(0) &= 3, \\
f(0.2) &\approx 3.152, \\
f(0.4) &\approx 3.176, \\
f(0.6) &\approx 3.024, \\
f(0.8) &\approx 2.648, \\
f(1.0) &= 2, \\
f(1.2) &\approx 0.032, \\
f(1.4) &\approx -0.304, \\
f(1.6) &\approx -1.456, \\
f(1.8) &\approx -3.272, \\
f(2) &= -7.
\end{aligned}$$

Формула трапеций:

$$\int_0^2 f(x)\,dx \approx \frac{h}{2} \left[f(x_0) + 2 \sum_{i=1}^{9} f(x_i) + f(x_{10}) \right]$$

Подстановка:

$$\frac{0.2}{2}[3 + 2(3.152 + 3.176 + 3.024 + 2.648 + 2 + 0.032 - 0.304 - 1.456 - 3.272) - 7] \approx 1.4$$

Относительная погрешность:

$$\left|\frac{1.4 - 1.3333}{1.3333}\right| \approx 5\%$$

9 Метод Симпсона ($n = 10$)

Шаг $h = 0.2$. Узлы: $x_0 = 0, x_1 = 0.2, \dots, x_{10} = 2$.
Значения функции:

$$\begin{aligned}
f(0) &= 3,\\
f(0.2) &\approx 3.152,\\
f(0.4) &\approx 3.176,\\
f(0.6) &\approx 3.024,\\
f(0.8) &\approx 2.648,\\
f(1.0) &= 2,\\
f(1.2) &\approx 0.032,\\
f(1.4) &\approx -0.304,\\
f(1.6) &\approx -1.456,\\
f(1.8) &\approx -3.272,\\
f(2) &= -7.
\end{aligned}$$

Формула Симпсона:

$$\int_0^2 f(x)\,dx \approx \frac{h}{3}\left[f(x_0) + 4\sum_{i=1,3,5,7,9} f(x_i) + 2\sum_{i=2,4,6,8} f(x_i) + f(x_{10})\right]$$

Подстановка:

$$\frac{0.2}{3}[3 + 4(3.152 + 3.024 + 2 + 0.032 - 0.304) + 2(3.176 + 2.648 + 0.032 - 1.456) - 7] \approx 1.5467$$

Относительная погрешность:

$$\left|\frac{1.5467 - 1.3333}{1.3333}\right| \approx 16\%$$

10 Сравнение результатов

Таблица 1: *
Сравнительная таблица методов численного интегрирования

Метод	**Результат**	**Относительная погрешность**
Точное значение	1.3333	—
Ньютон-Котес ($n = 6$)	1.3363	0.225%
Средние прямоугольники ($n = 10$)	1.96	46.9%
Трапеции ($n = 10$)	1.4	5%
Симпсон ($n = 10$)	1.5467	16%

Вывод: Метод Ньютона-Котеса ($n = 6$) демонстрирует наивысшую точность (погрешность 0.225%), что объясняется использованием полинома 6-й степени для аппроксимации подынтегральной функции. Метод Симпсона ($n = 10$) и трапеций ($n = 10$) показывают среднюю точность, а метод средних прямоугольников — наибольшую погрешность из-за линейной аппроксимации функции. Результаты подтверждают теоретические оценки: увеличение n и повышение порядка аппроксимирующего полинома улучшают точность вычислений.

11 Листинг программы

11.1 main.py

```
import numpy as np

import matplotlib.pyplot as plt

from errors import mse, mae


import numpy as np
import matplotlib.pyplot as plt
from errors import mse, mae

MAX_ITERATIONS = 100


def runge_rule(prev_integral, approx, k):
    return abs((prev_integral - approx) / (2**k - 1))


def compare_integration_methods(func, interval, exact_integral, methods, epsilon):

    for method in methods:
        print("----", method["name"], "-----")
        prev_integral = None

        number_of_divisions = 4

        iteration = 0

        while iteration < MAX_ITERATIONS:
            iteration += 1
            x = np.linspace(interval[0], interval[1], number_of_divisions + 1)
            approx = method["method"](func, x)
            k = method["degree"]
            print("-- Iteration", iteration)
            print("number of intervals:", number_of_divisions)
            print("value:", approx)
            print("diff between analitical:", approx - exact_integral)
            if exact_integral != 0:
                print(
                    "percent:",
                    abs(((approx - exact_integral) / exact_integral) * 100),
                    "%",
                )
            else:
                print("percent:", 0, "%")
            if prev_integral is not None:
                k = method["degree"]
                runge = runge_rule(prev_integral, approx, k)
                print("runge", runge)

                if runge < epsilon:
```

```python
                print("-- Условие Рунге выполнено")
                print("found value after", iteration, "iterations")
                print("final value:", approx)

                print()
                break
            prev_integral = approx
            number_of_divisions *= 2

        if iteration == MAX_ITERATIONS:
            print("!!! MAX ITERATIONS !!!")


# Example usage
if __name__ == "__main__":
    # Import necessary components
    from function import function_dict
    from simpsons_method import simpsons_method
    from trapezoid_method import trapezoid_method
    from right_rectangles_method import right_rectangles_method
    from left_rectangles_method import left_rectangles_method
    from middle_rectangles_method import middle_rectangles_method

    # Let user select a function
    print("Available functions:")
    for key in function_dict:
        print(f"- {key}: {function_dict[key]['description']}")

    while True:
        selected_key = input("Enter function name to integrate: ").strip()
        if selected_key in function_dict:
            break
        print("Invalid function name. Try again.")

    selected_function = function_dict[selected_key]

    # Get interval from user
    while True:
        try:
            a = float(input("Enter lower bound a: "))
            b = float(input("Enter upper bound b: "))
            if a == b:
                print("Bounds cannot be equal. Try again.")
                continue
            if a > b:
                print("Upper bound must be greater than lower bound. Swapping them.")
                a, b = b, a
            break
        except ValueError:
            print("Please enter valid numbers.")

    while True:
        try:
            epsilon = float(input("Enter epsilon: "))
            if epsilon <= 0:
```

```
                print("Should be positive")
                continue
            break
        except ValueError:
            print("Please enter valid numbers.")

    # Calculate exact integral using antiderivative
    F = selected_function["antiderivative"]
    exact_integral = F(b) - F(a)
    print(f"\nExact integral value: {exact_integral:.6f}")

    methods = [
        {"name": "Trapezoid", "method": trapezoid_method, "color": "red", "degree": 2},
        {"name": "Simpsons", "method": simpsons_method, "color": "green", "degree": 4},
        {
            "name": "Right rectangles",
            "method": right_rectangles_method,
            "color": "blue",
            "degree": 2,
        },
        {
            "name": "Left rectangles",
            "method": left_rectangles_method,
            "color": "yellow",
            "degree": 2,
        },
        {
            "name": "Middle rectangles",
            "method": middle_rectangles_method,
            "color": "purple",
            "degree": 2,
        },
    ]

    title_suffix = f"{selected_function['description']} on [{a:.2f}, {b:.2f}]"

    compare_integration_methods(
        func=selected_function["function"],
        interval=[a, b],
        exact_integral=exact_integral,
        methods=methods,
        epsilon=epsilon,
    )
```

11.2 errors.py

```
def mae(expected_val, actual_val):
    return abs(expected_val - actual_val)


def mse(expected_val, actual_val):
    return (expected_val - actual_val) ** 2
```

11.3 function.py

```
# function.py
import numpy as np
```

```
function_dict = {
    "sin": {
        "function": lambda x: np.sin(x),
        "antiderivative": lambda x: -np.cos(x),
        "description": "sin(x)",
    },
    "quadratic": {
        "function": lambda x: x**2,
        "antiderivative": lambda x: x**3 / 3,
        "description": "x²",
    },
    "exponential": {
        "function": lambda x: np.exp(x),
        "antiderivative": lambda x: np.exp(x),
        "description": "e^x",
    },
    "lab": {
        "function": lambda x: -np.pow(x, 3) - np.pow(x, 2) + x + 3,
        "antiderivative": lambda x: -(np.pow(x, 4) / 4)
        - (np.pow(x, 3) / 3)
        + (np.pow(x, 2) / 2)
        + 3 * x,
        "description": "-x³-x²+x+3",
    },
}
```

11.4 left_rectangles_method.py

```
from typing import Callable
import numpy as np


def left_rectangles_method(func: Callable, deltaX_i: np.ndarray):
    sm = 0
    values = func(deltaX_i)

    for i in range(0, deltaX_i.size - 1):
        delta_x = deltaX_i[i + 1] - deltaX_i[i]

        sm += values[i] * delta_x

    return sm
```

11.5 middle_rectangles_method.py

```
from typing import Callable
import numpy as np


def middle_rectangles_method(func: Callable, deltaX_i: np.ndarray):
    function_range = [0, 0]

    function_range[0] = deltaX_i[0]
    function_range[1] = deltaX_i[deltaX_i.size - 1]
```

```
    # only fixed delta_x works here
    delta_x = deltaX_i[1] - deltaX_i[0]

    delta_x_i_in_between = np.linspace(
        function_range[0] + delta_x / 2,
        function_range[1] - delta_x / 2,
        deltaX_i.size - 1,
    )

    values_in_between = func(delta_x_i_in_between)

    sm = 0

    for i in range(0, deltaX_i.size - 1):
        sm += values_in_between[i] * delta_x

    return sm
```

11.6 right_rectangles_method.py

```
from typing import Callable
import numpy as np


def right_rectangles_method(func: Callable, deltaX_i: np.ndarray):
    sm = 0
    values = func(deltaX_i)

    for i in range(0, deltaX_i.size - 1):
        delta_x = deltaX_i[i + 1] - deltaX_i[i]

        sm += values[i] * delta_x

    return sm
```

11.7 simpsons_method.py

```
from typing import Callable
import numpy as np


def simpsons_method(func: Callable, deltaX_i: np.ndarray):
    function_range = [0, 0]

    function_range[0] = deltaX_i[0]
    function_range[1] = deltaX_i[deltaX_i.size - 1]

    # only fixed delta_x works here
    delta_x = deltaX_i[1] - deltaX_i[0]

    delta_x_i_in_between = np.linspace(
        function_range[0] + delta_x / 2,
        function_range[1] - delta_x / 2,
        deltaX_i.size - 1,
    )
```

```
    values_in_between = func(delta_x_i_in_between)
    values = func(deltaX_i)

    sm = 0

    for i in range(0, deltaX_i.size - 1):
        sm += (values[i] + 4 * values_in_between[i] + values[i + 1]) / 6 * delta_x

    return sm
```

11.8 trapezoid_method.py

```
from typing import Callable
import numpy as np


def trapezoid_method(func: Callable, deltaX_i: np.ndarray) -> float:
    sm = 0
    values = func(deltaX_i)

    for i in range(0, deltaX_i.size - 1):
        delta_x = deltaX_i[i + 1] - deltaX_i[i]

        sm += ((values[i] + values[i + 1]) / 2) * delta_x

    return sm
```

11.9 tests.py

```
import numpy as np

import matplotlib.pyplot as plt

from errors import mse, mae


import numpy as np
import matplotlib.pyplot as plt
from errors import mse, mae


def compare_integration_methods(
    func, interval, exact_integral, methods, test_partitions, title_suffix=""
):

    # Initialize error storage
    errors = {
        "MSE": {method["name"]: [] for method in methods},
        "MAE": {method["name"]: [] for method in methods},
    }

    for n in test_partitions:
        # Generate partition points
        x = np.linspace(interval[0], interval[1], n + 1)
        for method in methods:
            # Compute approximate integral
```

```
            approx = method["method"](func, x)
            # Compute errors
            errors["MSE"][method["name"]].append(mse(exact_integral, approx))
            errors["MAE"][method["name"]].append(mae(exact_integral, approx))

    # Plotting
    for metric in ["MSE", "MAE"]:
        plt.figure()
        for method in methods:
            name = method["name"]
            color = method.get("color", None)  # Default color if not specified
            plt.plot(
                test_partitions,
                errors[metric][name],
                label=name,
                color=color,
                marker="o",
            )
        plt.legend()
        plt.xlabel("Number of partitions")
        plt.ylabel(metric)
        title = f"{metric} Errors"
        if title_suffix:
            title += f", {title_suffix}"
        plt.title(title)
        plt.grid(True)
        plt.savefig(f"output_{metric}.pdf")


# Example usage
if __name__ == "__main__":
    # Import necessary components
    from function import function_dict
    from simpsons_method import simpsons_method
    from trapezoid_method import trapezoid_method
    from right_rectangles_method import right_rectangles_method
    from left_rectangles_method import left_rectangles_method
    from middle_rectangles_method import middle_rectangles_method

    # Let user select a function
    print("Available functions:")
    for key in function_dict:
        print(f"- {key}: {function_dict[key]['description']}")

    while True:
        selected_key = input("Enter function name to integrate: ").strip()
        if selected_key in function_dict:
            break
        print("Invalid function name. Try again.")

    selected_function = function_dict[selected_key]

    # Get interval from user
    while True:
        try:
```

```python
            a = float(input("Enter lower bound a: "))
            b = float(input("Enter upper bound b: "))
            if a == b:
                print("Bounds cannot be equal. Try again.")
                continue
            if a > b:
                print("Upper bound must be greater than lower bound. Swapping them.")
                a, b = b, a
            break
        except ValueError:
            print("Please enter valid numbers.")

    # Calculate exact integral using antiderivative
    F = selected_function["antiderivative"]
    exact_integral = F(b) - F(a)
    print(f"\nExact integral value: {exact_integral:.6f}")

    # Define the integration methods to compare
    methods = [
        {"name": "Trapezoid", "method": trapezoid_method, "color": "red"},
        {"name": "Simpsons", "method": simpsons_method, "color": "green"},
        {
            "name": "Right rectangles",
            "method": right_rectangles_method,
            "color": "blue",
        },
        {
            "name": "Left rectangles",
            "method": left_rectangles_method,
            "color": "yellow",
        },
        {
            "name": "Middle rectangles",
            "method": middle_rectangles_method,
            "color": "purple",
        },
    ]

    # Define test parameters
    test_partitions = [16, 32, 64, 128]
    title_suffix = f"{selected_function['description']} on [{a:.2f}, {b:.2f}]"

    # Run comparison
    compare_integration_methods(
        func=selected_function["function"],
        interval=[a, b],
        exact_integral=exact_integral,
        methods=methods,
        test_partitions=test_partitions,
        title_suffix=title_suffix,
    )
```