Университет ИТМО

Вычислительная математика

Лабораторная работа №3

ФИО студента: Готовко Алексей Владимирович
Направление подготовки: 09.03.04 (СППО)
Учебная группа: P32101
ФИО преподавателей:
Малышева Т.А.
Рыбаков С.Д.

Санкт-Петербург
2023г.

1 Цель работы

Найти приближенное значение определенного интеграла с требуемой точностью различными численными методами.

2 Вычислительная реализация задачи

Интеграл по варианту:

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

2.1 Точное значение

$$\int_0^2 (-x^3 - x^2 + x + 3)\, dx = \left(-\frac{1}{4}x^4 - \frac{1}{3}x^3 + \frac{1}{2}x^2 + 3x \right) \Bigg|_0^2 = \frac{4}{3} - 0 = \frac{4}{3}$$

2.2 Формула Ньютона-Котеса

При $n = 5$:

$$\begin{aligned}\int_0^2 (-x^3 - x^2 + x + 3)\, dx &\approx \sum_{i=0}^{5} f(x_i) c_n^i \\ &= f(0)c_5^0 + f(0.4)c_5^1 + f(0.8)c_5^2 + f(1.2)c_5^3 + f(1.6)c_5^4 + f(2)c_5^5 \\ &= 3\frac{19 \cdot 2}{288} + 3.176\frac{75 \cdot 2}{288} + 2.648\frac{50 \cdot 2}{288} + 1.032\frac{50 \cdot 2}{288} - 2.056\frac{75 \cdot 2}{288} - 7\frac{19 \cdot 2}{288} \approx 1.333\end{aligned}$$

Погрешность:

$$\frac{|1.333 - 1.333|}{1.333} = 0$$

2.3 Формула средних прямоугольников

При $n = 10$:

$$\int_0^2 (-x^3 - x^2 + x + 3)\, dx \approx \sum_{i=1}^{10} \frac{2}{10} f\left(x_{i-1} + \frac{2}{10} \right) \approx 1.359$$

Погрешность:

$$\frac{|1.333 - 1.359|}{1.333} \approx 0.019$$

2.4 Формула трапеций

При $n = 10$:

$$\int_0^2 (-x^3 - x^2 + x + 3)\, dx \approx 0.1 \cdot \left(f(x_0) + f(x_{10}) + 2\sum_{i=1}^{9} f(x_i) \right) \approx 1.279$$

Погрешность:

$$\frac{|1.333 - 1.279|}{1.333} \approx 0.040$$

2.5 Формула Симпсона

При $n = 10$:

$$\int_0^2 (-x^3 - x^2 + x + 3)\, dx \approx \frac{1}{15}\left[(f(x_0) + 4 \cdot (f(x_1) + f(x_3) + \cdots + f(x_9)) + 2 \cdot (f(x_2) + f(x_4) + \cdots + f(x_8)) + f(x_{10})) \right] \approx 1.333$$

Погрешность:

$$\frac{|1.333 - 1.333|}{1.333} = 0$$

3 Программная реализация задачи

```python
def rectangle_calculate(method_type, func, left, part_len, i):
    if method_type == RectangleMethodType.LEFT:
        return func.calculate(left + part_len * i) * part_len
    if method_type == RectangleMethodType.RIGHT:
        return func.calculate(left + part_len * (i + 1)) * part_len
    if method_type == RectangleMethodType.MIDDLE:
        return func.calculate(left + part_len * i + part_len / 2) * part_len


def rectangles(method_type, func, left, right, eps):
    res = validate_boundaries_and_epsilon(left, right, eps)
    if res.is_error:
        return res

    prev_int = 0
    partitions_cnt = 4
    interval_len = right - left
    part_len = interval_len / partitions_cnt
    cur_int = 0

    while True:
        for i in range(partitions_cnt):
            cur_int += rectangle_calculate(method_type, func, left, part_len, i)

        if abs(cur_int - prev_int) < eps:
            break

        prev_int = cur_int
        partitions_cnt *= 2
        part_len = interval_len / partitions_cnt
        cur_int = 0

    res.answer = cur_int
    res.partition_cnt = partitions_cnt
    res.accuracy = abs(cur_int - prev_int)

    return res


def trapezoid_calculate(func, left, part_len, i, j):
    return 0.5 * part_len * (func.calculate(left + i * part_len) + func.calculate(left + j * part_len))


def trapezoids(func, left, right, eps):
    res = validate_boundaries_and_epsilon(left, right, eps)
    if res.is_error:
        return res

    prev_int = 0
    partitions_cnt = 4
    interval_len = right - left
    part_len = interval_len / partitions_cnt
    cur_int = 0

    while True:
        for i in range(1, partitions_cnt):
            j = i - 1
            cur_int += trapezoid_calculate(func, left, part_len, i, j)
```

```python

        if abs(cur_int - prev_int) < eps:
            break

        prev_int = cur_int
        partitions_cnt *= 2
        part_len = interval_len / partitions_cnt
        cur_int = 0

    res.answer = cur_int
    res.partition_cnt = partitions_cnt
    res.accuracy = abs(cur_int - prev_int)

    return res


def simpson_calculate(part_len, odd_y, even_y, y_n):
    return part_len / 3 * (even_y[0] + 4 * sum(odd_y) + 2 * sum(even_y[1:]) + y_n)


def simpson(func, left, right, eps):
    res = validate_boundaries_and_epsilon(left, right, eps)
    if res.is_error:
        return res

    prev_int = 0
    partitions_cnt = 4
    interval_len = right - left
    part_len = interval_len / partitions_cnt

    while True:
        y = [func.calculate(i * part_len + left) for i in range(partitions_cnt)]
        odd_y = [y[i] for i in range(1, partitions_cnt, 2)]
        even_y = [y[i] for i in range(2, partitions_cnt - 1, 2)]
        y_n = y[-1]

        cur_int = simpson_calculate(part_len, odd_y, even_y, y_n)

        if abs(cur_int - prev_int) < eps:
            break

        prev_int = cur_int
        partitions_cnt *= 2
        part_len = interval_len / partitions_cnt

    res.answer = cur_int
    res.partition_cnt = partitions_cnt
    res.accuracy = abs(cur_int - prev_int)

    return res

```

Полный код программы доступен по ссылке.

4 Вывод

Для наиболее оптимального распределения человеческих ресурсов и поддержания количества нервных клеток в головном мозгу, настоятельно рекомендуется использовать готовые библиотеки, содержащие наиболее эффективные реализации алгоритмов, вместо самостоятельной реализации оных.

Иными словами, вообще лучше зачиллиться и не изобретать велосипед.