Университет ИТМО
МФ КТиУ, Ф ПИиКТ

Лабораторная работа №3
Дисциплина «Вычислительная математика»

Численное интегрирование

Выполнил
Аскаров Эмиль Рамилевич

Преподаватель:
Машина Екатерина
Алексеевна

г. Санкт-Петербург
2024 г.

Вычислительная реализация задачи

Точное вычисление:

$I_{точн} = \int_0^2(-x^3 - x^2 - 2x + 1)dx = \left(-\frac{x^4}{4} - \frac{x^3}{3} - x^2 + x\right)\Big|_0^2 =$
= (- 16 / 4 – 8 / 3 - 4 + 2) – 0 = -4 – 8 / 3 - 4 + 2 = -6 – 8 / 3 = **-26 / 3 = -8.667**

По формуле Ньютона – Котеса при n = 6:

h = (b – a) / n = (2 – 0) / 6 = 1 / 3

i	0	1	2	3	4	5	6
x_i	0	1/3	2/3	1	4/3	5/3	2
y_i	1	0.185	-1.074	-3	-5.815	-9.741	-15
c_6^i	0.098	0.514	0.064	0.648	0.064	0.514	0.098

$I_{cotes} = \sum_{i=0}^{n} c_n^i * f(x_i)$ = 0.098 * 1 + 0.514 * 0.185 + 0.064 * (-1.074) + 0.648 * (-3) + 0.064 * (-5.815) + 0.514 * (-9.741) + 0.098 * (-15) = -8.669

R = | I_{cotes} - $I_{точн}$ | = 0.002 => 0.023%

По формуле средних прямоугольников при n = 10:

h = (b – a) / n = 0.2

i	0	1	2	3	4	5	6	7	8	9	10
x_i	0	0.2	0.4	0.6	0.8	1	1.2	1.4	1.6	1.8	2
$(x_i + x_{i-1}) / 2$		0.1	0.3	0.5	0.7	0.9	1.1	1.3	1.5	1.7	1.9
$f((x_i + x_{i-1}) / 2)$		0.789	0.283	-0.375	-1.233	-2.339	-3.741	-5.487	-7.625	-10.203	-13.269

$I_{сред} = h * \sum_{i=1}^{n} f\left(\frac{(x_{i-1}+x_i)}{2}\right)$ = 0.2 * (-43.989) = **-8.64**

R = | $I_{сред}$ - $I_{точн}$ | = 0.027 => 0.31%

По формуле трапеций при n = 10:
h = (b – a) / n = 0.2

i	0	1	2	3	4	5	6	7	8	9	10
x_i	0	0.2	0.4	0.6	0.8	1	1.2	1.4	1.6	1.8	2
y_i	1	0.552	-0.024	-0.776	-1.752	-3	-4.568	-6.504	-8.856	-11.672	-15

$I_{трап} = h * (\frac{y_0+y_n}{2} + \sum_{i=1}^{n-1} y_i)$ = 0.2 * (-7 – 36.6) = **-8.72**

R = | $I_{трап}$ - $I_{точн}$ | = 0.053 => 0.61%

По формуле Симпсона при n = 10:
h = (b – a) / n = 0.2

i	0	1	2	3	4	5	6	7	8	9	10
x_i	0	0.2	0.4	0.6	0.8	1	1.2	1.4	1.6	1.8	2
y_i	1	0.552	-0.024	-0.776	-1.752	-3	-4.568	-6.504	-8.856	-11.672	-15

$$I_{симп} = \frac{h}{3} * (y_0 + 4 * (y_1 + y_3 + \dots + y_{n-1}) + 2 * (y_2 + y_4 + \dots + y_{n-2}) + y_n) =$$

= 0.2 / 3 * (1 + 4 * (-21.4) + 2 * (-30.2) - 15) = **-8.668**

R = | $I_{трап}$ - $I_{симп}$ | = 0.001 => 0.01%

Программная реализация задачи

```
def get_break(f, a, b, step):
    try:
        f(a)
    except ArithmeticError:
        return a
    try:
        f(b)
    except ArithmeticError:
        return b
    x = a
    while x <= b:
        try:
            f(x)
            x += step
        except ArithmeticError:
            return x

def left_rectangles(f, a, b, n):
    h = (b - a) / n
    xs = [a + i * h if i != n else b for i in range(n + 1)]
    return h * sum([f(xs[i]) for i in range(n)])


def right_rectangles(f, a, b, n):
    h = (b - a) / n
    xs = [a + i * h if i != n else b for i in range(n + 1)]
    return h * sum([f(xs[i]) for i in range(1, n + 1)])


def middle_rectangles(f, a, b, n):
    h = (b - a) / n
    xs = [a + i * h if i != n else b for i in range(n + 1)]
    return h * sum([f((xs[i - 1] + xs[i]) / 2) for i in range(1, n + 1)])


def trapecia(f, a, b, n):
    h = (b - a) / n
    xs = [a + i * h if i != n else b for i in range(n + 1)]
    ys = [f(x) for x in xs]

    return h * ((ys[0] + ys[n]) / 2 +
                sum([ys[i] for i in range(1, n)]))


def simpson(f, a, b, n):
    h = (b - a) / n
    xs = [a + i * h if i != n else b for i in range(n + 1)]
    ys = [f(x) for x in xs]
    return h / 3 * (ys[0] +
                    4 * sum([ys[i] for i in range(1, n, 2)]) +
                    2 * sum([ys[i] for i in range(2, n - 1, 2)]) +
```

```python
                    ys[n])


def read_number(s: str):
    while True:
        try:
            return float(input(s))
        except Exception:
            continue


eps = 0.001


def compute(func, a, b, eps, method):
    n = 4
    i0 = method(func, a, b, n)
    i1 = method(func, a, b, n * 2)
    while abs(i1 - i0) > eps:
        n *= 2
        i0 = i1
        i1 = method(func, a, b, n * 2)
    return i1, n


def additional():
    print('f1 = 1 / x')
    print('f2 = 1 / (3 - 4 * x) ** (1 / 5)')
    print('f3 = 1 / (x + 1) ** 0.5')
    functions = [lambda x: 1 / x, lambda x: 1 / (3 - 4 * x) ** (1 / 5), lambda x: 1 / (x
+ 1) ** 0.5]
    bps = [(0, False), (0.75, True), (-1, True)]

    while True:
        try:
            ind = int(input('Введите номер функции: ')) - 1
            f = functions[ind]
            bp = bps[ind]
            break
        except Exception:
            continue

    a = read_number('Введите нижнюю границу интегрирования (a): ')
    b = read_number('Введите верхнюю границу интегрирования (b): ')

    methods = [left_rectangles, right_rectangles, middle_rectangles, trapecia, simpson]
    method_name = ["Левых прямоугольников",
                   "Правых прямоугольников",
                   "Средних прямоугольников",
                   "Трапеции",
                   "Симпсона"]
    # undefined = get_break(f, a, b, eps)
    eps2 = 0.00001
    for name, method in zip(method_name, methods):
        if not bp[1] and a <= bp[0] <= b:
            print("Интеграл не сходится на заданном интервале")
            break
        if bp[0] == a:
            res, _ = compute(f, a + eps2, b, eps, method)
        elif bp[0] == b:
            res, _ = compute(f, a, b - eps2, eps, method)
        elif a <= bp[0] <= b:
            res = compute(f, a, bp[0] - eps2, eps, method)[0] + compute(f, bp[0] + eps2,
b, eps, method)[0]
        else:
            res, _ = compute(f, a, b, eps, method)
        if isinstance(res, complex):
            print("Интеграл не определен на заданном интервале")
            break
        print(f'{name} I = {res:.3f}')
```

```python
def main():
    print('f1 = x ** 2')
    print('f2 = x')
    print('f3 = -x ** 3 - x ** 2 - 2 * x + 1')
    functions = [lambda x: x ** 2, lambda x: x, lambda x: -x ** 3 - x ** 2 - 2 * x + 1]

    while True:
        try:
            f = functions[int(input('Введите номер функции: ')) - 1]
            break
        except Exception:
            continue

    a = read_number('Введите нижнюю границу интегрирования (a): ')
    b = read_number('Введите верхнюю границу интегрирования (b): ')

    methods = [left_rectangles, right_rectangles, middle_rectangles, trapecia, simpson]
    method_name = ["Левых прямоугольников",
                   "Правых прямоугольников",
                   "Средних прямоугольников",
                   "Трапеции",
                   "Симпсона"]
    for name, method in zip(method_name, methods):
        res, n = compute(f, a, b, eps, method)
        print(f'{name} I = {res:.3f} n = {n}')


if __name__ == '__main__':
    additional()
```

Тестовые данные

```
➜  lab3 git:(main) ✗ python3 main.py
f1 = 1 / x
f2 = 1 / (3 - 4 * x) ** (1 / 5)
f3 = 1 / (x + 1) ** 0.5
Введите номер функции: 2
Введите нижнюю границу интегрирования (a): 0
Введите верхнюю границу интегрирования (b): 0.75
Левых прямоугольников I = 0.751
Правых прямоугольников I = 0.753
Средних прямоугольников I = 0.751
Трапеции I = 0.753
Симпсона I = 0.753
```

```
➜  lab3 git:(main) ✗ python3 main.py
f1 = x ** 2
f2 = x
f3 = -x ** 3 - x ** 2 - 2 * x + 1
Введите номер функции: 2
Введите нижнюю границу интегрирования (a): 0 10
Введите нижнюю границу интегрирования (a): 0
Введите верхнюю границу интегрирования (b): 10
Левых прямоугольников I = 49.994 n = 4096
Правых прямоугольников I = 50.006 n = 4096
Средних прямоугольников I = 50.000 n = 4
Трапеции I = 50.000 n = 4
Симпсона I = 50.000 n = 4
```

Вывод

В ходе лабораторной работы я познакомился с численными методами решения определённых интегралов.

Метод прямоугольников – частный случай метода Ньютона-Котеса. Имеет три модификации (левые, средние, правые прямоугольники). Средние – самая лучшая модификация. Из плюсов – простота в понимании и в реализации, из минусов – не такой точный.

Метод трапеций - частный случай метода Ньютона-Котеса. Тоже простой и не такой точный (но точнее прямоугольников).

Метод Симпсона – частный случай метода Ньютона-Котеса. Более сложный в понимании и имеет более сложную формулу, но зато является довольно точным.

Метод Ньютона-Котеса – общий случай перечисленных выше методов. Использует более общие и сложные формулы, не такой простой в программной реализации.

Квадратурная формула Гаусса – позволяет повысить порядок точности методов за счёт специального выбора узлов интегрирования. Состоит из двух этапов: 1) свести интеграл к интегралу с пределами [-1, 1]; 2) вычислить по специальной формуле (сумма значений подынтегральной функции в специальных точках, умноженных на весовые коэффициенты). Является более сложным в понимании и в программной реализации.