МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

ЛАБОРАТОРНАЯ РАБОТА №2

‘ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА’

Вариант №22

Студент:
Самарина Арина Анатольевна, Суржицкий Арсений Арсентьевич
Группа P3266
Преподаватель:
Машина Екатерина Александровна

Санкт-Петербург, 2024

1. Цели работы

Изучить численные методы нахождения определенных интегралов, выполнить программную реализацию методов.

2. Описание метода, расчётные формулы

Формула Ньютона - Котеса

Простой прием построения квадратурных формул состоит в том, что подынтегральная функция $f(x)$ заменяется на отрезке $[a, b]$ **интерполяционным многочленом Лагранжа** $L_n(x)$, совпадающий с $f(x)$ в узлах интерполяции $x_0, x_1, \dots, x_n \in [a, b]$. Полином Лагранжа имеет вид:

$$L_n(x) = \sum_{i=0}^{n} f(x_i) L_n^i(x), \qquad n = 0, 1, 2 \dots$$

где $L_n^i(x)$ - коэффициенты Лагранжа (полиномы степени *n*):

$$L_n^i(x) = \frac{(x - x_0)(x - x_1) \dots (x - x_{i-1})(x - x_{i+1}) \dots (x - x_n)}{(x_i - x_0)(x_i - x_1) \dots (x_i - x_{i-1})(x_i - x_{i+1}) \dots (x_i - x_n)}$$

Если полином Лагранжа «близок» к $f(x)$, то интегралы от них тоже должны быть «близки»:

$$\int_a^b f(x)dx \approx \int_a^b L_n(x)dx = \sum_{i=0}^{n} f(x_i) \int_a^b L_n^i(x)dx$$

Коэффициенты Котеса: $c_n^i = \int_a^b L_n^i(x)dx$

Формула Ньютона-Котеса порядка n:

$$\int_a^b \boldsymbol{f(x)dx} \approx \int_a^b \boldsymbol{L_n(x)dx} = \sum_{i=0}^{n} \boldsymbol{f(x_i)c_n^i}$$

Метод прямоугольников

На каждом шаге интегрирования функция аппроксимируется полиномом нулевой степени – отрезком, параллельным оси абсцисс. Площадь криволинейной трапеции приближенно заменяется площадью многоугольника, составленного из *n*- прямоугольников. Таким образом, вычисление определенного интеграла сводится к нахождению суммы *n*- элементарных прямоугольников.

$$\int_a^b f(x)dx \approx S_n = \sum_{i=1}^{n} f(\xi_i)\Delta x_i$$

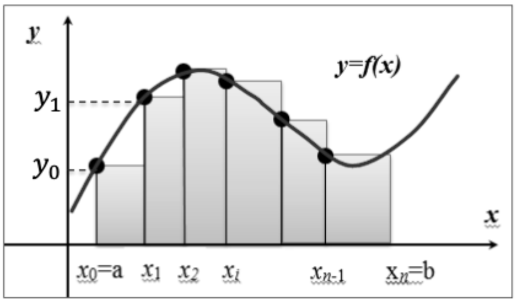
Различают метод левых, правых и средних прямоугольников.

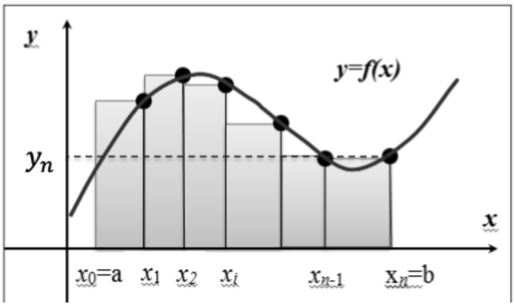
В качестве точек ξ_i могут выбираться левые ($\xi_i = x_{i-1}$) или правые ($\xi_i = x_i$) границы отрезков, получим формулы левых и правых прямоугольников.

Обозначим:

$f(x_i) = y_i$, $f(a) = y_0$, $f(b) = y_n$

$\Delta x_i = x_i - x_{i-1} = h_i$





$\int_a^b f(x)dx \approx h_1y_0 + h_2y_1 + \cdots + h_ny_{n-1} = \sum_{i=1}^{n} h_i\, y_{i-1}$ - левые прямоугольники

$\int_a^b f(x)dx \approx h_1y_1 + h_2y_2 + \cdots + h_ny_n = \sum_{i=1}^{n} h_iy_i$ - правые прямоугольники

При $h_i = h = \frac{b-a}{n} = const$:

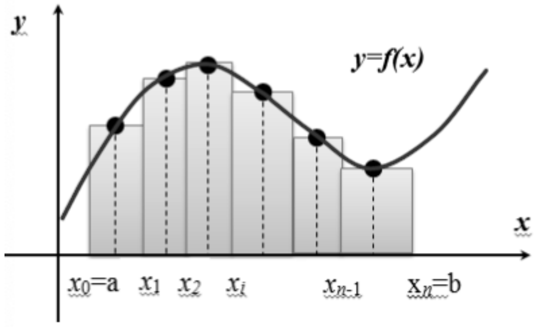
$$\int_a^b \boldsymbol{f(x)dx = h\sum_{i=1}^{n} y_{i-1}}$$

$$\int_a^b \boldsymbol{f(x)dx = h\sum_{i=1}^{n} y_i}$$

Для аналитически заданных функций более точным является использование значений в средних точках элементарных отрезков (полуцелых узлах):

$$\int_a^b f(x)dx \approx \sum_{i=1}^n h_i\, f(x_{i-1/2})$$

$$x_{i-1/2} = \frac{x_{i-1} + x_i}{2} = x_{i-1} + \frac{h_i}{2}, i = 1, 2, \dots n$$



При $h_i = h = \frac{b-a}{n} = const$:

$$\int_a^b \boldsymbol{f(x)dx = h \sum_{i=1}^{n} f(x_{i-1/2})}$$

Метод трапеций

Подынтегральную функцию на каждом отрезке $[x_i; x_{i+1}]$ заменяют интерполяционным многочленом первой степени:

$$f(x) \approx \varphi_i(x) = a_i x + b$$

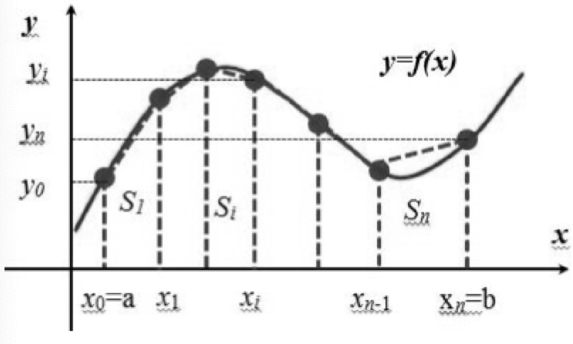
Используют линейную интерполяцию, т.е. график функции $y = f(x)$ представляется в виде ломаной, соединяющий точки (x_i, y_i) . Площадь всей фигуры (криволинейной трапеции):

$$S_{общ} = S_1 + S_2 + \dots + S_n = \frac{y_0 + y_1}{2}h_1 + \frac{y_1 + y_2}{2}h_2 + \dots + \frac{y_{n-1} + y_n}{2}h_n$$

$$y_0 = f(a), \quad y_n = f(b), \quad y_i = f(x_i), \quad h_i = x_i - x_{i-1}$$

Складывая все эти равенства, получаем формулу трапеций для численного интегрирования:

$$\int_a^b f(x)dx = \frac{1}{2}\sum_{i=1}^{n} h_i(y_{i-1} + y_i)$$



При $h_i = h = \frac{b-a}{n} = const$ формула трапеций:

$$\int_a^b \boldsymbol{f(x)dx = h \cdot \left(\frac{y_0 + y_n}{2} + \sum_{i=1}^{n-1} y_i\right)}$$

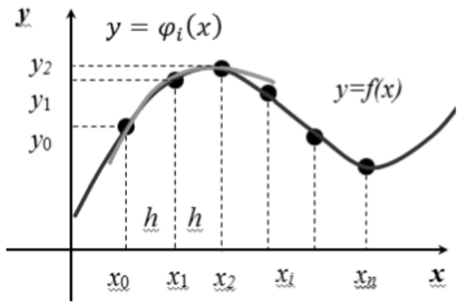
или $$\int_a^b \boldsymbol{f(x)dx = \frac{h}{2} \cdot \left(y_0 + y_n + 2\sum_{i=1}^{n-1} y_i\right)}$$

Разобьем отрезок интегрирования [*a, b*] на четное число *n* равных частей с шагом *h*. На каждом отрезке $[x_0, x_2], [x_2, x_4], \ldots, [x_{i-1}, x_{i+1}], \ldots, [x_{n-2}, x_n]$ подынтегральную функцию заменим интерполяционным многочленом второй степени:

$$f(x) \approx \varphi_i(x) = a_i x^2 + b_i x + c_i, \quad x_{i-1} \le x \le x_{i+1}$$

Коэффициенты этих квадратных трехчленов могут быть найдены из условий равенства многочлена и подынтегральной функции в узловых точках.

В качестве $\varphi_i(x)$ можно принять интерполяционный многочлен Лагранжа второй степени, проходящий через точки $(x_{i-1}, y_{i-1}), (x_i, y_i), (x_{i+1}, y_{i+1})$.



3. Вычислительная часть

Интеграл для вычислений:

22	$\int_3^5 (2x^3 - 3x^2 + 4x - 22)dx$

Точное значение интеграла = 162

По формуле Ньютона-Котеса при n = 6:

I = (**41*** (**b** − **a**) / **840**) * f(3)=1,66 + (2**16***(**b** − **a**) / **840**) * f(3,33)=18,07 + (**27***(**b** − **a**) / **840**) * f(3,66)=21,32 + (**272***(**b** − **a**) / **840**) * f(3,99)=68,75 + (**27***(**b** − **a**) / **840**) * f(4,32)= 75,21 + (2**16***(**b** − **a**) / **840**) * f(4.65)=143,52 + (**41*** (**b** − **a**) / **840**) * f(5)=160,41

R = |162-160,41| = 1,59

По формуле средних прямоугольников при n = 10:

h = (5 - 3) / 10 = 0.2

$x_i =$ 3,2; 3,4; 3,6; 3,8; 4,0; 4,2; 4,4; 4,6; 4,8; 5,0

$x_i + \frac{h}{2}$= 3.3, 3.5, 3.7, 3.9, 4.1, 4.3, 4.5, 4.7, 4.9

I = 0.2 * 788.15 = 157,63

R = |162-157,63| = 4,37

По формуле трапеций при n = 10:

h = (5 - 3) / 10 = 0.2

x[i] = 3.2, 3.4, 3.6, 3.8, 4, 4.2, 4.4, 4.6, 4.8, 5

y[i] = 25.62, 35.53, 46,83, 59,62, 74, 90.1, 107.9, 127.6, 149.26, 173

I = 0,2 * ((17+173)/2 + (25662 + 35,53 + 46,83 + 59,62 + 74 + 90,1 + 107,9 + 127,6 + 147,26) = 162,29

R = |162-162.29| = 0,29

По формуле Симпсона при n = 10:

h = (5 - 3) / 10 = 0.2

x[i] = 3.2, 3.4, 3.6, 3.8, 4, 4.2, 4.4, 4.6, 4.8, 5

y[i] = 25.62, 35.53, 46.83, 59.62, 74, 90.1, 107.9, 127.6, 149.26, 173

I = 0,2 / 3 * (25.62+ 4 * (35.53+ 59.62+ 90.1+ 127.6) + 2 * (46.83+ 74+ 107.9+ 149.26) + 173) ≈ 147.1
R = |162-147.1| = 14,9

4. Листинг программы

Main.py
```
from numerical_integration import *
import sys


def main():
    numerical_integration()


if __name__ == "__main__":
    main()
```

numerical_integration.py
```
import sys
import math
from validate_input import input_variables


def numerical_integration():

    quation, method, left_border, right_border, inaccuracy, parts =
input_variables()
    switch_command = {
        1: Rectangle_method_left,
        2: Rectangle_method_centre,
        3: Rectangle_method_right,
        4: trapezoidal_method,
        5: simpson_method,
        6: exit,
    }
```

```
    switch_command.get(method, exit)(
        quation, left_border, right_border, inaccuracy, parts
    )


def Rectangle_method_left(quation, left_border, right_border,
inaccuracy, parts):
    I = 99999

    while I > inaccuracy and parts < 10000000:
        i = 1
        h = (right_border - left_border) / parts
        h2 = (right_border - left_border) / (parts // 2)
        integral = 0
        integral_2 = 0

        for i in range(parts // 2):
            integral_2 += integrand(quation, left_border + (i - 1) *
h2)
        integral_2 *= h2
        i = 1
        for i in range(parts):
            integral += integrand(quation, left_border + (i - 1) * h)
        integral *= h
        I = abs((integral_2 - integral) / (2**2 - 1))
        parts *= 2
    if integral < float("inf"):
        print(f"S = {integral}\nParts = {parts//2}\nInnacuary = {I}")
    else:
        print("The integral doesn't converge.")


def Rectangle_method_centre(quation, left_border, right_border,
inaccuracy, parts):
    I = 99999

    while I > inaccuracy and parts < 10000000:
        i = 1
        h = (right_border - left_border) / parts
        h2 = (right_border - left_border) / (parts // 2)
        integral = 0
        integral_2 = 0

        for i in range(parts // 2):
            integral_2 += integrand(quation, left_border + (i - 1 + h2
/ 2) * h2)
        integral_2 *= h2
        i = 1
        for i in range(parts):
            integral += integrand(quation, left_border + (i - 1 + h /
2) * h)
        integral *= h
        I = abs((integral_2 - integral) / (2**2 - 1))
        parts *= 2
    if integral < float("inf"):
        print(f"S = {integral}\nParts = {parts//2}\nInnacuary = {I}")
```

```python
    else:
        print("The integral doesn't converge.")


def Rectangle_method_right(quation, left_border, right_border,
inaccuracy, parts):
    I = 99999

    while I > inaccuracy and parts < 10000000:
        i = 1
        h = (right_border - left_border) / parts
        h2 = (right_border - left_border) / (parts // 2)
        integral = 0
        integral_2 = 0

        for i in range(parts // 2):
            integral_2 += integrand(quation, left_border + (i) * h2)
        integral_2 *= h2
        i = 1
        for i in range(parts):
            integral += integrand(quation, left_border + (i) * h)
        integral *= h
        I = abs((integral_2 - integral) / (2**2 - 1))
        parts *= 2
    if integral < float("inf"):
        print(f"S = {integral}\nParts = {parts//2}\nInnacuary = {I}")
    else:
        print("The integral doesn't converge.")


def trapezoidal_method(quation, left_border, right_border, inaccuracy,
parts):
    I = 99999

    while I > inaccuracy and parts < 10000000:
        i = 1
        h = (right_border - left_border) / parts
        h2 = (right_border - left_border) / (parts // 2)
        integral = 0.5 * (
            integrand(quation, left_border) + integrand(quation,
right_border)
        )
        integral_2 = 0.5 * (
            integrand(quation, left_border) + integrand(quation,
right_border)
        )

        for i in range(parts // 2):
            integral_2 += integrand(quation, left_border + i * h2)
        integral_2 *= h2
        i = 1
        for i in range(parts):
            integral += integrand(quation, left_border + i * h)
        integral *= h
        I = abs((integral_2 - integral) / (2**2 - 1))
        parts *= 2
```

```python
    if integral < float("inf"):
        print(f"S = {integral}\nParts = {parts//2}\nInnacuary = {I}")
    else:
        print("The integral doesn't converge.")


def simpson_method(quation, left_border, right_border, inaccuracy,
parts):
    I = 99999

    while I > inaccuracy and parts < 10000000:
        # i = 1
        h = (right_border - left_border) / parts
        h2 = (right_border - left_border) / (parts // 2)
        integral = 0
        integral_2 = 0
        x_values = [left_border + i * h for i in range(parts + 1)]
        x_values_2 = [left_border + i * h2 for i in range(parts // 2 +
1)]

        integral_2 = integrand(quation, left_border) +
integrand(quation, right_border)
        for i in range(1, parts // 2, 2):
            integral_2 += 4 * integrand(quation, x_values_2[i])
        for i in range(2, parts // 2 - 1, 2):
            integral_2 += 2 * integrand(quation, x_values_2[i])
        integral_2 *= h2 / 3

        integral = integrand(quation, left_border) +
integrand(quation, right_border)
        for i in range(1, parts, 2):
            integral += 4 * integrand(quation, x_values[i])
        for i in range(2, parts - 1, 2):
            integral += 2 * integrand(quation, x_values[i])
        integral *= h / 3
        I = abs((integral_2 - integral) / (2**2 - 1))
        parts *= 2
    if integral < float("inf"):
        print(f"S = {integral}\nParts = {parts//2}\nInnacuary = {I}")
    else:
        print("The integral doesn't converge.")


def integrand(quation, x):
    if quation == 1:
        try:
            return 1 / x
        except ZeroDivisionError:
            print(f"Infinity breaking point at {x}")
            exit()
    if quation == 2:
        return x**2
    if quation == 3:
        return x
```

validate_input.py

```python
import sys
import math
from validate_input import input_variables


def numerical_integration():

    quation, method, left_border, right_border, inaccuracy, parts = 
input_variables()
    switch_command = {
        1: Rectangle_method_left,
        2: Rectangle_method_centre,
        3: Rectangle_method_right,
        4: trapezoidal_method,
        5: simpson_method,
        6: exit,
    }
    switch_command.get(method, exit)(
        quation, left_border, right_border, inaccuracy, parts
    )


def Rectangle_method_left(quation, left_border, right_border, 
inaccuracy, parts):
    I = 99999

    while I > inaccuracy and parts < 10000000:
        i = 1
        h = (right_border - left_border) / parts
        h2 = (right_border - left_border) / (parts // 2)
        integral = 0
        integral_2 = 0

        for i in range(parts // 2):
            integral_2 += integrand(quation, left_border + (i - 1) * 
h2)
        integral_2 *= h2
        i = 1
        for i in range(parts):
            integral += integrand(quation, left_border + (i - 1) * h)
        integral *= h
        I = abs((integral_2 - integral) / (2**2 - 1))
        parts *= 2
    if integral < float("inf"):
        print(f"S = {integral}\nParts = {parts//2}\nInnacuary = {I}")
    else:
        print("The integral doesn't converge.")


def Rectangle_method_centre(quation, left_border, right_border, 
inaccuracy, parts):
    I = 99999

    while I > inaccuracy and parts < 10000000:
        i = 1
        h = (right_border - left_border) / parts
```

```
        h2 = (right_border - left_border) / (parts // 2)
        integral = 0
        integral_2 = 0

        for i in range(parts // 2):
            integral_2 += integrand(quation, left_border + (i - 1 + h2
/ 2) * h2)
        integral_2 *= h2
        i = 1
        for i in range(parts):
            integral += integrand(quation, left_border + (i - 1 + h /
2) * h)
        integral *= h
        I = abs((integral_2 - integral) / (2**2 - 1))
        parts *= 2
    if integral < float("inf"):
        print(f"S = {integral}\nParts = {parts//2}\nInnacuary = {I}")
    else:
        print("The integral doesn't converge.")


def Rectangle_method_right(quation, left_border, right_border,
inaccuracy, parts):
    I = 99999

    while I > inaccuracy and parts < 10000000:
        i = 1
        h = (right_border - left_border) / parts
        h2 = (right_border - left_border) / (parts // 2)
        integral = 0
        integral_2 = 0

        for i in range(parts // 2):
            integral_2 += integrand(quation, left_border + (i) * h2)
        integral_2 *= h2
        i = 1
        for i in range(parts):
            integral += integrand(quation, left_border + (i) * h)
        integral *= h
        I = abs((integral_2 - integral) / (2**2 - 1))
        parts *= 2
    if integral < float("inf"):
        print(f"S = {integral}\nParts = {parts//2}\nInnacuary = {I}")
    else:
        print("The integral doesn't converge.")


def trapezoidal_method(quation, left_border, right_border, inaccuracy,
parts):
    I = 99999

    while I > inaccuracy and parts < 10000000:
        i = 1
        h = (right_border - left_border) / parts
        h2 = (right_border - left_border) / (parts // 2)
        integral = 0.5 * (
```

```
            integrand(quation, left_border) + integrand(quation,
right_border)
        )
        integral_2 = 0.5 * (
            integrand(quation, left_border) + integrand(quation,
right_border)
        )

        for i in range(parts // 2):
            integral_2 += integrand(quation, left_border + i * h2)
        integral_2 *= h2
        i = 1
        for i in range(parts):
            integral += integrand(quation, left_border + i * h)
        integral *= h
        I = abs((integral_2 - integral) / (2**2 - 1))
        parts *= 2
    if integral < float("inf"):
        print(f"S = {integral}\nParts = {parts//2}\nInnacuary = {I}")
    else:
        print("The integral doesn't converge.")


def simpson_method(quation, left_border, right_border, inaccuracy,
parts):
    I = 99999

    while I > inaccuracy and parts < 10000000:
        # i = 1
        h = (right_border - left_border) / parts
        h2 = (right_border - left_border) / (parts // 2)
        integral = 0
        integral_2 = 0
        x_values = [left_border + i * h for i in range(parts + 1)]
        x_values_2 = [left_border + i * h2 for i in range(parts // 2 +
1)]

        integral_2 = integrand(quation, left_border) +
integrand(quation, right_border)
        for i in range(1, parts // 2, 2):
            integral_2 += 4 * integrand(quation, x_values_2[i])
        for i in range(2, parts // 2 - 1, 2):
            integral_2 += 2 * integrand(quation, x_values_2[i])
        integral_2 *= h2 / 3

        integral = integrand(quation, left_border) +
integrand(quation, right_border)
        for i in range(1, parts, 2):
            integral += 4 * integrand(quation, x_values[i])
        for i in range(2, parts - 1, 2):
            integral += 2 * integrand(quation, x_values[i])
        integral *= h / 3
        I = abs((integral_2 - integral) / (2**2 - 1))
        parts *= 2
    if integral < float("inf"):
        print(f"S = {integral}\nParts = {parts//2}\nInnacuary = {I}")
```

```
    else:
        print("The integral doesn't converge.")


def integrand(quation, x):
    if quation == 1:
        try:
            return 1 / x
        except ZeroDivisionError:
            print(f"Infinity breaking point at {x}")
            exit()
    if quation == 2:
        return x**2
    if quation == 3:
        return x
```

5. Вывод:

В ходе выполнения работы мы познакомились с численными методами решения определенных интегралов, научились решать их вручную и с помощью программы.