**ОТЧЁТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3**

**МЕТОДЫ ЧИСЛЕННОГО ИНТЕГРИРОВАНИЯ**

**(Вариант 9)**

*Выполнил студент 3 курса МОиАИС*

*Рубайло Егор*

***Постановка задачи:*** изучение различных методов вычисления определенных интегралов, практическое интегрирование функций на ЭВМ. Применить полученные знания на функции **xln(x).**

I. Вычислить приближенно с заданной точностью интеграл    
по формулам прямоугольников (левых, правых, центральных), трапеций и Симпсона. Величину шага определить с помощью двойного пересчета.

II. Определить относительную погрешность вычислений каждого метода по формуле: , где *I* – точное значение интеграла;  – приближенное.

***Результаты расчетов***

Расчёты были проведены приведёнными в таблице методами. Решение искали на интервале [2, 6]. Точность всех вычислений принимаем за . За точное значение интеграла примем 22.8653.

**Итоговая таблица**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод решения | Полученное решение | Ширина шага | Количество итераций | Погрешность |
| {%tr for method in records %} | | | | |
| {{loop.index}}. {{method}} | {{records[method][‘result’]}} | {{records[method][‘h’]}} | {{records[method][‘steps’]}} | {{records[method][‘err’]}} |
| {%tr endfor %} | | | | |

**Выводы:** Метод Симпсона показывает выдающуюся скорость сходимости. Приближенным значением интеграла является .

Все исходные тексты программ приводятся в Приложении

**ПРИЛОЖЕНИЕ**

***Программы нахождения значения интеграла всеми способами***

# Вариант 9

# x\*ln(x) e=10\*\*(-4) [2, 6]

import numpy as np

from util import meta

@meta(name="Метод левых прямоугольников")

def rectangular\_left(f, a, b, e=10 \*\* (-4), steps\_start=10):

span = b - a

N = steps\_start

convergence = False

h\_n = span / N

res\_n = h\_n \* np.sum([

f(a + i \* h\_n)

for i in range(N)

])

while not convergence:

h\_2n = span / (2 \* N)

res\_2n = h\_2n \* np.sum([

f(a + i \* h\_2n)

for i in range(2 \* N)

])

convergence = abs(res\_2n - res\_n) <= e

N \*= 2

res\_n = res\_2n

return res\_2n, h\_2n, N \* 2

@meta(name="Метод правых прямоугольников")

def rectangular\_right(f, a, b, e=10 \*\* (-4), steps\_start=10):

span = b - a

N = steps\_start

convergence = False

h\_n = span / N

res\_n = h\_n \* np.sum([

f(a + i \* h\_n)

for i in range(1, N + 1)

])

while not convergence:

h\_2n = span / (2 \* N)

res\_2n = h\_2n \* np.sum([

f(a + i \* h\_2n)

for i in range(1, 2 \* N + 1)

])

convergence = abs(res\_2n - res\_n) <= e

N \*= 2

res\_n = res\_2n

return res\_2n, h\_2n, N \* 2

@meta(name="Метод средних прямоугольников")

def rectangular\_middle(f, a, b, e=10 \*\* (-4), steps\_start=10):

span = b - a

N = steps\_start

convergence = False

h\_n = span / N

res\_n = h\_n \* np.sum([

f(a + i \* h\_n + h\_n / 2)

for i in range(N)

])

while not convergence:

h\_2n = span / (2 \* N)

res\_2n = h\_2n \* np.sum([

f(a + i \* h\_2n + h\_2n / 2)

for i in range(2 \* N)

])

convergence = abs(res\_2n - res\_n) <= e

N \*= 2

res\_n = res\_2n

return res\_2n, h\_2n, N \* 2

@meta(name="Метод трапеций")

def trapezoid(f, a, b, e=10 \*\* (-4), steps\_start=10):

span = b - a

N = steps\_start

convergence = False

median = (f(a) + f(b)) / 2

h\_n = span / N

res\_n = h\_n \* (median + np.sum([

f(a + i \* h\_n)

for i in range(1, N)

]))

while not convergence:

h\_2n = span / (2 \* N)

res\_2n = h\_2n \* (median + np.sum([

f(a + i \* h\_2n)

for i in range(1, 2 \* N)

]))

convergence = abs(res\_2n - res\_n) <= e

N \*= 2

res\_n = res\_2n

return res\_2n, h\_2n, N \* 2

@meta(name="Метод Симпсона")

def simpson\_integrator(f, a, b, e=10 \*\* (-4), steps\_start=10):

convergence = False

span = b - a

N = steps\_start

h\_n = span / N

x\_n = np.fromfunction(

lambda i: a + h\_n \* i,

(N,)

)

s1\_n = 2 \* np.sum([

f(x\_n[2 \* i])

for i in range(1, (N // 2) - 1)

])

s2\_n = 4 \* np.sum([

f(x\_n[2 \* i - 1])

for i in range(1, N // 2)

])

res\_n = h\_n / 3 \* (f(a) + s1\_n + s2\_n + f(b))

while not convergence:

h\_2n = span / (2 \* N)

x\_2n = np.fromfunction(

lambda i: a + h\_2n \* i,

(2 \* N,)

)

s1\_2n = 2 \* np.sum([

f(x\_2n[2 \* i])

for i in range(1, N)

])

s2\_2n = 4 \* np.sum([

f(x\_2n[2 \* i - 1])

for i in range(1, N + 1)

])

res\_2n = h\_2n / 3 \* (f(a) + s1\_2n + s2\_2n + f(b))

convergence = abs(res\_2n - res\_n) <= e

N \*= 2

res\_n = res\_2n

return res\_2n, h\_2n, N \* 2