

# Вычмат лаба4

## "Численное интегрирование"

Вариант 19

Б9122-02.03.01сцт 2 группа

### Цель

- 1) Найти точное значение интеграла  $I^* = \int_a^b f(x)dx$
- 2) Получить формулу трапеций для численного интегрирования из формулы Ньютона-Котеса
- 3) Исследовать порядок аппроксимации метода трапеций. Получить теоретическую оценку для  $R_n$
- 4) Провести вычислительный эксперимент на методе трапеций и сделать вывод о поведении ошибки
- 5) Провести вычислительный эксперимент для всех квадратурных формул и сделать вывод об эффективности метода трапеций
- 6) Сделать общий вывод

### Ход работы

#### ▼ 1) Точное значение интеграла

Моему варианту соответствует функция  $f(x) = x^2 + \lg(x)$ ,  
отрезок интегрирования  $[0.4, 0.9]$

Применяя формулу интегрирования по частям на логарифме получаем:

$$\int_{0.4}^{0.9} [x^2 + \lg(x)] dx = \frac{x^3}{3} \Big|_{0.4}^{0.9} + \frac{x \ln x - x}{\ln 10} \Big|_{0.4}^{0.9} =$$

$$= \frac{0.9^3 - 0.4^3}{3} + \frac{0.9 \ln 0.9 - 0.4 \ln 0.4 - 0.5}{\ln 10} = 0.122513687679$$

## ▼ 2) Вывод формулы трапеций

Исходный интеграл можно представить как сумму интегралов при заданном узловом разбиении:

$$I = \int_a^b f(x) dx = \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x) dx$$

Формула трапеций подразумевает использование двух узлов, потому для каждого кусочка формула Ньютона-Котеса примет вид:

$$\sum_{k=0}^1 f(x_{i+k}) \int_{x_i}^{x_{i+1}} \prod_{j=0, j \neq k}^1 \frac{x - x_{i+j}}{x_{i+k} - x_{i+j}} dx \quad (*)$$

Преобразуем её:

$$\begin{aligned} (*) &= f(x_i) \int_{x_i}^{x_{i+1}} \prod_{j=0, j \neq 0}^1 \frac{x - x_{i+j}}{x_i - x_{i+j}} dx + f(x_{i+1}) \int_{x_i}^{x_{i+1}} \prod_{j=0, j \neq 1}^1 \frac{x - x_{i+j}}{x_{i+1} - x_{i+j}} dx = \\ &= f(x_i) \int_{x_i}^{x_{i+1}} \frac{x - x_{i+1}}{x_i - x_{i+1}} dx + f(x_{i+1}) \int_{x_i}^{x_{i+1}} \frac{x - x_i}{x_{i+1} - x_i} dx = \\ &= \frac{f(x_i)}{-h} \int_{x_i}^{x_{i+1}} (x - x_{i+1}) dx + \frac{f(x_{i+1})}{h} \int_{x_i}^{x_{i+1}} (x - x_i) dx = \\ &= \frac{f(x_i)}{-h} \left( \frac{x_{i+1}^2 - x_i^2}{2} - x_{i+1}h \right) + \frac{f(x_{i+1})}{h} \left( \frac{x_{i+1}^2 - x_i^2}{2} - x_ih \right) = \\ &= \frac{f_{i+1} - f_i}{h} \frac{x_{i+1}^2 - x_i^2}{2} + f_i x_{i+1} - f_{i+1} x_i = \\ &= \frac{f_{i+1} - f_i}{h} \frac{(x_{i+1} - x_i)(x_{i+1} + x_i)}{2} + f_i x_{i+1} - f_{i+1} x_i = \\ &= \frac{f_{i+1} - f_i}{2} (x_{i+1} + x_i) + f_i x_{i+1} - f_{i+1} x_i = \\ &= \frac{f_{i+1} x_{i+1} + f_{i+1} x_i - f_i x_{i+1} - f_i x_i}{2} + f_i x_{i+1} - f_{i+1} x_i = \frac{f_{i+1} + f_i}{2} (x_{i+1} - x_i) = \frac{f_{i+1} + f_i}{2} h \end{aligned}$$

Таким образом получаем формулу трапеций:

$$I = \sum_{i=0}^{n-1} \frac{f_{i+1} + f_i}{2} h$$

▼ 3) Исследование порядка аппроксимации и нахождение теор. оценки

$$\begin{aligned} \int_a^b R_n(x) dx &= \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} \frac{f''(\xi)}{2!} (x-x_0)(x-x_1) dx = \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} \frac{f''(\xi)}{2!} (x-x_i)(x-x_i-h) dx = \\ &= \frac{f''(\xi)}{2!} \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} (x-x_i)^2 - h(x-x_i) dx = \frac{f''(\xi)}{2!} \sum_{i=0}^{n-1} \int_0^h t^2 - ht dt = \frac{f''(\xi)}{2!} \sum_{i=0}^{n-1} \left[ \frac{h^3}{3} - h \frac{h^2}{2} \right] = \frac{f''(\xi)}{12} nh^3 = \frac{f''(\xi)(b-a)}{12} h^2 \end{aligned}$$

$h^2 \rightarrow$  порядок аппроксимации 2

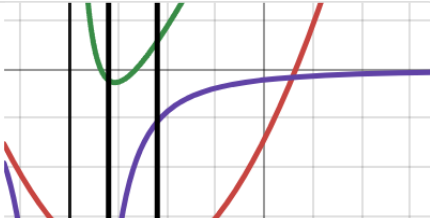
$$\begin{aligned} \int_a^b R_n(x) dx &= \int_{0.4}^{0.9} (x-0.4)(x-0.9) dx = \int_{0.4}^{0.9} (x-0.4)(x-0.4-0.5) dx = \\ &= \int_{0.4}^{0.9} (x-0.4)^2 dx - \frac{1}{2} \int_{0.4}^{0.9} (x-0.4) dx = \int_0^{0.5} t^2 dt - \frac{1}{2} \int_0^{0.5} t dt = \frac{1}{3} \left( \frac{1}{8} - 0 \right) - \frac{1}{4} \left( \frac{1}{4} - 0 \right) = -\frac{1}{48} \\ \int_a^b R_n(x) dx &= \frac{2 - \frac{1}{0.9^2 \ln 10}}{2} \left( -\frac{1}{48} \right) = -0.0152482705517 \end{aligned}$$

Вывод производной и все остальные выкладки были проведены в десмосе:

**Desmos | Graphing Calculator**

Explore math with our beautiful, free online graphing calculator. Graph functions, plot points, visualize algebraic equations, add sliders, animate graphs, and more.

<https://www.desmos.com/calculator/erzhmucf3>



▼ 4) Выч. эксперимент и вывод

N	2 <sup>i</sup>	I	$\Delta I$	$\partial I$	R_n	ratio
1	2	0.06578781259591092	0.056725875083089086	46.30166323269725	0.003812067637925	-
2	4	0.09258365400421101	0.029930033674788997	24.429950842071733	0.00095301690948125	0.5276257727350888
3	8	0.10734748857483842	0.015166199104161587	12.379187494460846	0.0002382542273703125	0.5067217521020215
4	16	0.11489104765029372	0.007622640028706287	6.2218680811228895	5.956355684257812e-05	0.502607144766723
5	32	0.11869346967697415	0.003820218002025852	3.1181968924445926	1.489088921064453e-05	0.5011673104907485
6	64	0.12060146170926198	0.0019122259607380295	1.560826390883182	3.7227223026611327e-06	0.5005541486700448
7	128	0.12155705804547837	0.0009566296335216384	0.7808349023238272	9.306805756652832e-07	0.5002701817990133
8	256	0.12203524522521009	0.0004784424537899179	0.39052163301417575	2.326701439163208e-07	0.5001334236622264
9	512	0.12227443473061075	0.00023925294838925137	0.19528670871137413	5.81675359790802e-08	0.5000663015876646
10	1024	0.12239405329780201	0.00011963438119799719	0.09764980833117444	1.454188399477005e-08	0.500033048718625
11	2048	0.12245386851464136	5.981916435864043e-05	0.04882651521793511	3.6354709986925124e-09	0.5000164982643123
12	4096	0.12248377760383555	2.99100751644521e-05	0.02441366000084819	9.088677496731281e-10	0.5000082412574159
13	8192	0.12249873251830985	1.495516069015268e-05	0.0120693048546292	2.2721693741828202e-10	0.5000041159350471
14	16384	0.12250621006797664	7.47761102336264e-06	0.006103490283432529	5.6804234354570506e-11	0.5000020513511647
15	32768	0.12250994886591263	3.7388130873783343e-06	0.0030517513252677985	1.4201058588642626e-11	0.5000010131172898

Видно, что ошибка имеет тенденцию раза так в четыре уменьшаться с увеличением количества узлов. Убывание происходит монотонно. Сама по себе она всегда маленькая, что говорит о неплохой точности метода трапеций.

Последний же столбец показывает, что отношение ошибок на соседних итерациях стремится к половине. Исходя из этого столбца и факта уменьшения ошибки в 4 раза следует, что порядок аппроксимации действительно 2.

#### ▼ 5) Второй выч. эксперимент и вывод

I	$\Delta I$	$\partial I$	R_n
0.12247638183020487	3.730584879513277e-05	0.030450351713253783	2.8531867804200002e-05
0.12252649095611044	1.2803277110437494e-05	0.010450487086784586	2.8531867804200002e-05
0.12253965112406927	2.596344506926307e-05	0.021192281092126043	7.62413527583e-11
0.12250143639315823	1.2251285841771709e-05	0.009999932312764507	1.5248270551699998e-10
0.12252294634514116	9.258666141154226e-06	0.007557250391003655	9.03601217877e-11
0.122513687679			

В таблице сверху вниз по порядку расписаны результаты эксперимента для квадратурных формул по порядку, представленному в pdf файле

Из результатов видно, что метод трапеций обладает довольно хорошей точностью при количестве узлов 10к. Однако формула Симпсона оказывается наиболее точной

#### ▼ 6) Общий вывод

Видно, что все квадратурные формулы при достаточно большом разбиении дают неплохую точность при вычислении интеграла. Однако формула Симпсона является

самой точной среди рассмотренных пяти методов. Наименее точной вышла формула левых прямоугольников

Формула трапеций, в свою очередь, на втором месте по точности. Её реализовать и вывести довольно просто, что я собственно и проделал в этой работе

В первом выч. эксперименте мы брали  $n$  равную степени двойки и рассматривали отношение ошибок на соседних итерациях. В моих вычислениях вышло, что соотношение стремится к  $1/2$ . Это может быть обусловлено тем фактом, что каждый раз количество узлов удваивалось. Как следствие, ошибка монотонно уменьшается в 4 раза, т.к. порядок аппроксимации 2

#### ▼ Листинг кринж кода

```
import numpy as np

def f(x):
    return x*x + np.log10(x)

I = 0.122513687679

brd = [0.4, 0.9]

def T_method(n):
    xs = np.linspace(*brd, n)
    ys = f(xs)

    h = (brd[1]-brd[0])/n
    S = 0

    for i in range(n-1): S += h * (ys[i+1] + ys[i])/2

    return S
```

```

ns = [(2<<_) for _ in range(15)]
Is = [T_method(ns[0])]
DeltaIs = [abs(I - Is[0])]
deltaIs = [DeltaIs[0] / I * 100]
Rs = [0.0152482705517 / ns[0] / ns[0]]
ratio = ['-']

# да это кринж
print("№.....2^i.....I.....ΔI.....∂I.....R_n.....ratio")
print("="*130)
print(1, ns[0], Is[0], DeltaIs[0], deltaIs[0], Rs[0], ratio[0], sep='\t')

for i in range(1, 15):
    Is.append(T_method(ns[i]))
    DeltaIs.append(abs(I - Is[i]))
    deltaIs.append(DeltaIs[i] / I * 100)
    Rs.append(0.0152482705517 / ns[i] / ns[i])
    ratio.append(DeltaIs[i] / DeltaIs[i-1])

    print(i+1, ns[i], Is[i], DeltaIs[i], deltaIs[i], Rs[i], ratio[i], sep='\t')

```

```

def L_rect(n):
    xs = np.linspace(*brd, n)
    ys = f(xs)

    h = (brd[1]-brd[0])/n
    S = 0

    for i in range(n-1): S += h * ys[i]

    return S

def R_rect(n):
    xs = np.linspace(*brd, n)
    ys = f(xs)

    h = (brd[1]-brd[0])/n
    S = 0

    for i in range(n-1): S += h * ys[i+1]

    return S

def C_rect(n):
    xs = np.linspace(*brd, n)
    ys = f(xs)

    h = (brd[1]-brd[0])/n
    S = 0

    for i in range(n//2): S += h * ys[2*i-1]

    return S*2 # я n делю на 2, значит шаг возрастает вдвое

def Simp(n):
    xs = np.linspace(*brd, n)
    ys = f(xs)

    h = (brd[1]-brd[0])/n
    S = 0

    for i in range(n//2): S += ys[2*i-2] + 4*ys[2*i-1] + ys[2*i]

    return S * h / 3

N = 10000
Is = [L_rect(N), R_rect(N), C_rect(N), T_method(N), Simp(N)]

```

```

... h = (brd[1]-brd[0])/n
... S = 0

... for i in range(n//2): S += ys[2*i-2] + 4*ys[2*i-1] + ys[2*i]

... return S * h / 3

N = 10000
Is = [L_rect(N), R_rect(N), C_rect(N), T_method(N), Simp(N)]
DeltaIs = [abs(I - Is[_]) for _ in range(len(Is))]
deltaIs = [DeltaIs[_] / I * 100 for _ in range(len(Is))]
Rs = [0.285318678042/N, 0.285318678042/N, 0.00762413527583/N, 0.0152482705517/N, 0.00903601217877/N]

print("I..... $\Delta I$ ..... $\partial I$ .....R_n")
print('='*100)
for i in range(len(Is)):
    print(Is[i], DeltaIs[i], deltaIs[i], Rs[i], sep='\t')
print('='*100)
print(I)

```