

Вычмат лаба2

“Интерполирование функции с помощью интерполяционных формул с конечными разностями”

Вариант 19

Б9122-02.03.01сцт 2 группа

Цель

1. Построить таблицу конечных разностей по значениям табличной функции
2. По соответствующим интерполяционным формулам вычислить значения функции в заданных узлах
3. Оценить максимум и минимум для $f^{(n+1)}(x)$
4. Проверить на выполнение равенство $\min R_n < R_n(z) < \max R_n$
где $R_n(z) = L_n(z) - f(z)$, z - заданный узел
5. Вывод

Ход работы

- ▼ 1. Табличка конечных разностей

Имеем функцию $f(x) = x^2 + \lg(x)$ и отрезок $[0.4, 0.9]$

Определяю функцию, границы, количество узлов для разбиения, шаг и два списка по аналогии с предыдущей лабой

```
import numpy as np

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

brds = [0.4, 0.9]
n = 9; nbuf = n-1
h = (brds[1] - brds[0]) / n

xs = np.linspace(brds[0], brds[1], n)
ys = [f(_) for _ in xs]
```

Переменная nbuf нужна для построения таблички конечных разностей

Поскольку табличка треугольная, удобнее собирать список списков с уменьшением длины

```
table = [ys]
nods = [0.43, 0.67, 0.86]

for i in range(n-1):
    table.append([table[i][j+1] - table[i][j] for j in range(nbuf)])
    nbuf -= 1
```

Сама табличка будет выглядеть так:

```

.. [-0.23794, -0.12098, -0.00422, 0.11416, 0.23541, 0.36044, 0.48993, 0.62439, 0.76424]

[0.11696, 0.11677, 0.11838, 0.12125, 0.12503, 0.12949, 0.13446, 0.13985]

[-0.00019, 0.00161, 0.00287, 0.00378, 0.00446, 0.00498, 0.00539]

[0.00181, 0.00126, 0.00091, 0.00068, 0.00052, 0.00041]

[-0.00055, -0.00035, -0.00023, -0.00016, -0.00011]

[0.0002, 0.00012, 7e-05, 5e-05]

[-9e-05, -5e-05, -3e-05]

[4e-05, 2e-05]

[-2e-05]

```

Важно уточнить, что здесь табличка “транспонирована” и каждый порядок конечных разностей располагается строками, а не столбцами

▼ 2. Нахождение значений

Первое, с чего я начинаю - это определение t

Делаю я это незамысловатым образом чисто по определению

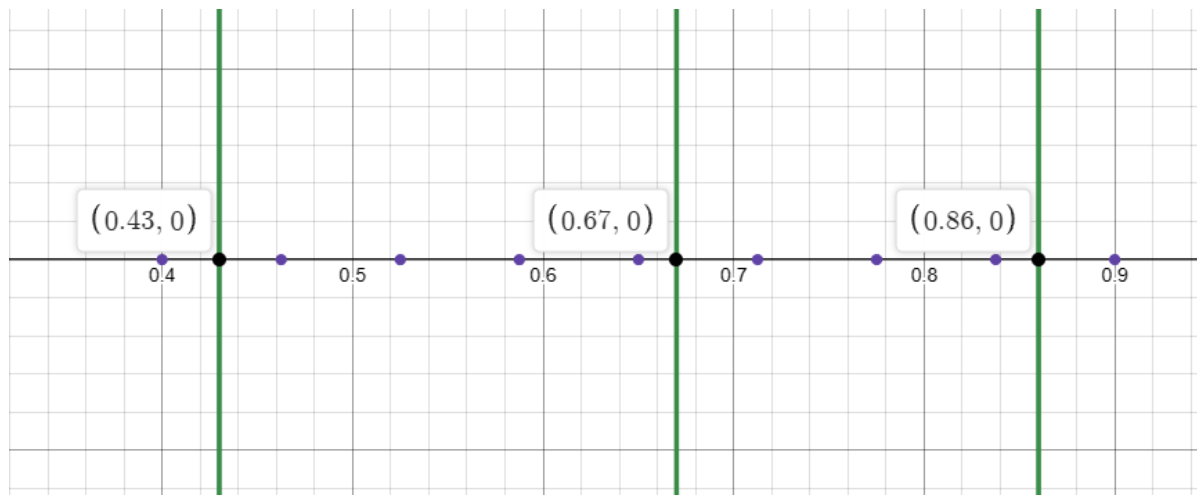
```

def find_t(x_звездочка, x_нулевое):
    ... return (x_звездочка - x_нулевое) / h

```

Далее необходимо определить, какие формулы использовать для вычисления

Я в десмосе вывел картину разбиения и положения данных мне точек:



Очевидно, что для первого узла нужно использовать Ньютона-вперёд, а для последнего - Ньютона-назад

Несложно определить и со вторым узлом: ближайшая к нему точка - ровно серединная точка разбиения, которая между прочим находится левее, откуда следует использование Гаусса-1

Теперь, когда определено, что юзать, можно писать реализации:

```

def Newton_forward(t):
    S = table[0][0]
    t_buf = t

    for i in range(1, n):
        S += table[i][0] * t_buf
        t_buf *= (t_buf-1) / (i+1)

    return S

def Newton_backward(t):
    S = table[0][-1]
    t_buf = t

    for i in range(1, n):
        S += table[i][-1] * t_buf
        t_buf *= (t_buf+1) / (i+1)

    return S

def Gauss_1(t):
    S = table[0][n//2]
    t_buf = t

    for i in range(1, n):
        S += table[i][len(table[i])//2] * t_buf
        t_buf *= (
            t_buf + int((i+1)/2) * (
                -1 if (i%2!=0) else 1
            ) ) / (i+1)

    return S

```

Далее вычисляем непосредственно значения функции (в порядке первая-последняя-вторая)

```
t = find_t(nods[0], brds[0])
print(t, f(nods[0]), Newton_forward(t), "\t", R1 := Newton_forward(t)-f(nods[0]))
[55] ✓ 0.0s
... 0.5399999999999995 -0.1816315444204135 -0.17466828605765913 0.006963258362754354

t = find_t(nods[-1], brds[1])
print(t, f(nods[-1]), Newton_backward(t), "\t", R3 := Newton_backward(t)-f(nods[-1]))
[56] ✓ 0.0s
... -0.7200000000000006 0.6740984512435676 0.6629948806154334 -0.011103570628134163

t = find_t(nods[1], xs[len(xs)//2])
print(t, f(nods[1]), Gauss_1(t), "\t", R2 := Gauss_1(t)-f(nods[1]))
[57] ✓ 0.0s
... 0.36000000000000003 0.2749748027008265 0.2799616264437464 0.004986823742919866
```

Видим, что t во всех случаях удовлетворяет соответствующим ограничениям


▼ 3. Оценка точек экстремума

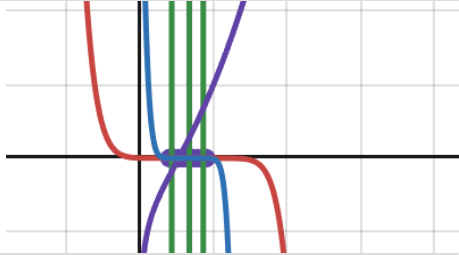
Как и в прошлый раз, большую часть выкладок я сделал в десмосе, потому что это банально удобнее. Там я снова вывел k -ую производную для своей функции, но т.к. в нашем случае порядок производной связан с количеством точек разбиения, там вылезла "магическая константа", которую я не увидел смысла вычислять программно

Собственно, вот ссылка на десмос:

a

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/a82wj8d9c2>



В код же пошло это:

```
def f_10th_derivative(x):  
    return -157596.781593 / (x**10)  
  
l = np.linspace(brds[0], brds[1], 100)  
f_l = f_10th_derivative(l)  
  
maxf = max(f_l)  
minf = min(f_l)
```

Здесь я определяю псевдонепрерывный интервал и уже ручками ищу точки экстремума

Далее я вывожу все значения из списка и анализирую ситуацию

```
f_l, maxf, minf
✓ 0.0s

(array([-1.50296003e+09, -1.32573146e+09, -1.17122128e+09, -1.03628944e+09,
        -9.18260671e+08, -8.14851242e+08, -7.24107922e+08, -6.44357056e+08,
        -5.74161936e+08, -5.12287071e+08, -4.57668164e+08, -4.09386834e+08,
        -3.66649303e+08, -3.28768376e+08, -2.95148189e+08, -2.65271271e+08,
        -2.38687548e+08, -2.15004984e+08, -1.93881601e+08, -1.75018666e+08,
        -1.58154854e+08, -1.43061255e+08, -1.29537086e+08, -1.17405999e+08,
        -1.06512916e+08, -9.67212849e+07, -8.79107224e+07, -7.99749688e+07,
        -7.28201205e+07, -6.63630961e+07, -6.05303051e+07, -5.52564909e+07,
        -5.04837229e+07, -4.61605184e+07, -4.22410766e+07, -3.86846086e+07,
        -3.54547523e+07, -3.25190587e+07, -2.98485425e+07, -2.74172872e+07,
        -2.52020978e+07, -2.31821958e+07, -2.13389501e+07, -1.96556399e+07,
        -1.81172455e+07, -1.67102631e+07, -1.54225419e+07, -1.42431385e+07,
        -1.31621893e+07, -1.21707963e+07, -1.12609261e+07, -1.04253202e+07,
        -9.65741504e+06, -8.95127079e+06, -8.30150815e+06, -7.70325170e+06,
        -7.15207946e+06, -6.64397778e+06, -6.17530103e+06, -5.74273553e+06,
        -5.34326722e+06, -4.97415267e+06, -4.63289310e+06, -4.31721108e+06,
        -4.02502949e+06, -3.75445275e+06, -3.50374978e+06, -3.27133877e+06,
        -3.05577340e+06, -2.85573044e+06, -2.66999849e+06, -2.49746793e+06,
        -2.33712174e+06, -2.18802723e+06, -2.04932857e+06, -1.92023999e+06,
        -1.80003967e+06, -1.68806417e+06, -1.58370333e+06, -1.48639575e+06,
        -1.39562459e+06, -1.31091378e+06, -1.23182460e+06, -1.15795251e+06,
        -1.08892433e+06, -1.02439563e+06, -9.64048353e+05, -9.07588642e+05,
        -8.54744895e+05, -8.05265943e+05, -7.58919409e+05, -7.15490199e+05,
        -6.74779132e+05, -6.36601673e+05, -6.00786783e+05, -5.67175862e+05,
        -5.35621780e+05, -5.05987987e+05, -4.78147700e+05, -4.51983155e+05]),
-451983.1554477576,
-1502960029.5352926)
```

А ситуация такая: у меня минимум и максимум - граничные точки

В десмосе я записал это так

$$\frac{f^{(10)}(0.4) - \text{минимум}}{f^{(10)}(0.9) - \text{максимум}}$$

Еще хочется отметить колоссальные значения по модулю. Это обуславливается всё той же причиной с рядом Тейлора для логарифма в

точке 0

Далее выведем функции для последующей оценки и увидим некоторую странность

$$R_9(x) = \frac{f^{(9+1)}}{(9+1)!} \cdot \prod_{i=1}^9 (x - L[i]);$$

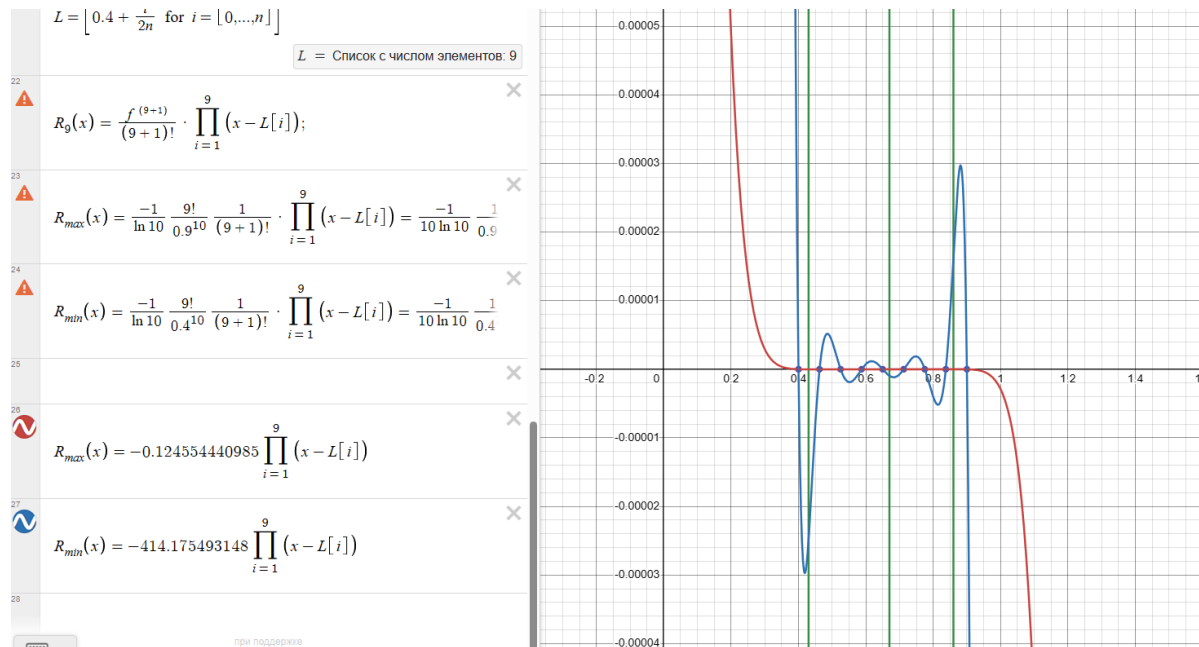
$$R_{max}(x) = \frac{-1}{\ln 10} \frac{9!}{0.9^{10}} \frac{1}{(9+1)!} \cdot \prod_{i=1}^9 (x - L[i]) = \frac{-1}{10 \ln 10} \frac{1}{0.9^{10}} \cdot \prod_{i=1}^9 (x - L[i]) = -0.124554440985 \prod_{i=1}^9 (x - L[i])$$

$$R_{min}(x) = \frac{-1}{\ln 10} \frac{9!}{0.4^{10}} \frac{1}{(9+1)!} \cdot \prod_{i=1}^9 (x - L[i]) = \frac{-1}{10 \ln 10} \frac{1}{0.4^{10}} \cdot \prod_{i=1}^9 (x - L[i]) = -414.175493148 \prod_{i=1}^9 (x - L[i])$$

С первого взгляда может показаться, что тут явная ошибка, однако причиной такого взрывного роста служит функция $\frac{1}{x^{10}}$. Именно она руинит всю картину за счёт стремительного роста в бесконечность близ точки 0

▼ 4. Проверка равенства

Если отрисовать графики оценочных функций, то получится ещё более странная вещь: функция для оценки минимума принимает большие значения, чем функция для максимума



И нетрудно уже на этом этапе сделать вывод, что проверяемое условие выполняться конечно же не будет

```
print( Rn(nods[0], "min"), R1, Rn(nods[0], "max") )
print( Rn(nods[1], "min"), R2, Rn(nods[1], "max") )
print( Rn(nods[2], "min"), R3, Rn(nods[2], "max") )
```

[60] ✓ 0.0s

```
... -0.0002481297718360579 0.006963258362754354 -7.461973374612624e-08
-9.544190214609834e-06 0.004986823742919866 -2.870211532323167e-09
0.00016381868163907053 -0.011103570628134163 4.926497258307828e-08
```

▼ 5. Вывод

В первую очередь хочется отметить, что относительная ошибка вышла не настолько большой, как я предполагал изначально:

```
print(R1 / 0.1816315444204135 * 100)
print(R2 / 0.2749748027008265 * 100)
print(-R3 / 0.6740984512435676 * 100)
39] ✓ 0.0s
.. 3.8337274425398524
   1.81355662189366
   1.6471734370032223
```

Наибольшая ошибка вышла для первого значения, потому что оно само по себе довольно маленькое, а значит отклонения от него будут сразу же видны

Проверяемое неравенство не выполняется. Тому может послужить гигантский разброс в экстремумах производной логарифма. Как я уже не раз писал здесь и в предыдущей лабе, ряд Тейлора для логарифма в нуле не существует, как раз из-за такого поведения производных