Вычмат лаба1

“Интерполяционный многочлен Лагранжа”

Вариант 19

Б9122-02.03.01сцт

Цель

1. Построить интерполяционный многочлен Лагранжа для функции из варианта
2. Построить таблицу абсолютной и относительной погрешностей и остаточного

члена для каждой n

3. Построить график зависимостей $\Delta f_n(n)$ и $r_n(n)$
4. Сделать вывод

Ход работы

▼ 1. Построение полинома

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

Для начала я определяю количество точек n, данный интервал и линейное пространство для узлов интерполяции (по сути просто список значений от 0.4 до 0.9 с шагом в 1/2n

```
import matplotlib.pyplot as plt
import numpy as np

n = 10
borders = [0.4, 0.9] # интервал определяю тут
xs = np.linspace(borders[0], borders[1], n) # линейное пространство для ПЛ
```

Далее определяю данную мне функцию через нампаевский логарифм, а также определяю список значений функции в узлах интерполяции

```
def f(x): # функцию соответственно тут
    return x*x + np.log10(x)
```

```
#xs = [0.43, 0.53, 0.67, 0.86]; n = len(xs)
ys = [f(_) for _ in xs]
pts = [(xs[i], ys[i]) for i in range(n)]
# ласт список нужен был только для отрисовки точек, не более
```

Всё это было программной подводкой к самому полиному, который я определяю так:

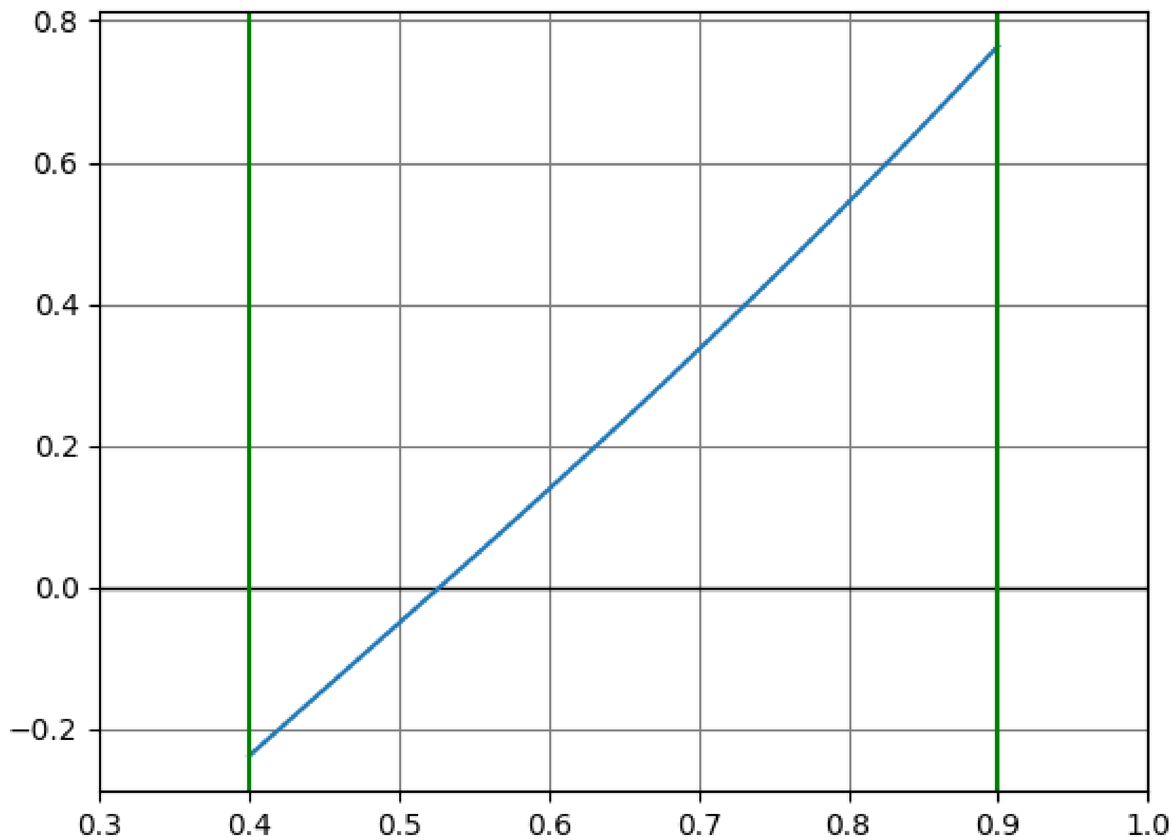
```
# лагранж момент

def Lagrange(x, n): # туть он определяется

    rez = 0

    for i in range(n):
        Prod = 1

        for j in range(n):
            if (i != j): Prod *= (x - xs[j]) / (xs[i] - xs[j])

        rez += ys[i] * Prod
        Prod = 1

    return rez
```

Я решил использовать глобальные массивы иксов и игреков

После чего я непосредственно строю полином и рисую график функции:

```
# построение графиков исходной функции и полинома лагранжа

l = np.linspace(borders[0], borders[1], 1000)
plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)
plt.grid(True, color='gray')
plt.xlim(0.3, 1)

#for p in pts: plt.scatter(p[0], p[1], color='red', zorder=10)

y_fl = f(l)
y_Ll = Lagrange(l, n) # воть тут строится ПЛ

plt.plot(l, y_fl)
#plt.plot( l, y_Ll )
```

```
plt.axvline(x=0.4, color="green")
plt.axvline(x=0.9, color="green")
```



Буквально первой же строчкой я определяю новый линспэйс с ещё большим разбиением

Он нужен чтобы создать псевдонепрерывность для последующего применения нормы для функции, которая оперирует с непрерывным интервалом.

▼ 2. Табличька

Прежде чем строить таблицу, очевидно нужно получить данные. Данные базируются на двух функциях, которые я определяю отдельно:

```
# норма

def norm(lst):
```

```
    return max(list(map(np.fabs, lst)))


# остаточный член-оценка
# https://www.desmos.com/calculator/iq0ma4i7ge

def r(n):
    return 0.542868102379 * (5**(n+1)) / (n+1)


# ошибки

DeltaErr = norm(y_Ll - y_fl) # абслютная

deltaerr = DeltaErr / norm(y_fl) * 100 # относительная
```

Теоретическую оценку (ака остаточный член) я решил отдельно выразить, используя Desmos. Там я вывел n-ую производную для исходной функции, которую подставил в норму. Норму раскрыл по определению и по свойствам модуля, после чего вычислил супремум. Ссылочка на вычисления

Но т.к. код работает с конкретным n, я решил его малясь переписать и на выходе получил такой маленький кодик:

```
# полевые условия

borders = [0.4, 0.9]
ns = [3, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

DeltaErr = {}; deltaerr = {}
rs = {}

print("n\tΔf\t\t\tδf\t\t\tr")
print("="*80)

for n in ns:

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

    l = np.linspace(borders[0], borders[1], 1000)

    y_fl = f(l)
```

```
    y_Ll = Lagrange(l, n)

    DeltaErr[n] = norm(y_Ll - y_fl)
    deltaerr[n] = DeltaErr[n] / norm(y_fl) * 100
    rs[n] = r(n)

    print(n, DeltaErr[n], deltaerr[n], rs[n], sep="\t")
```

Который выводит:

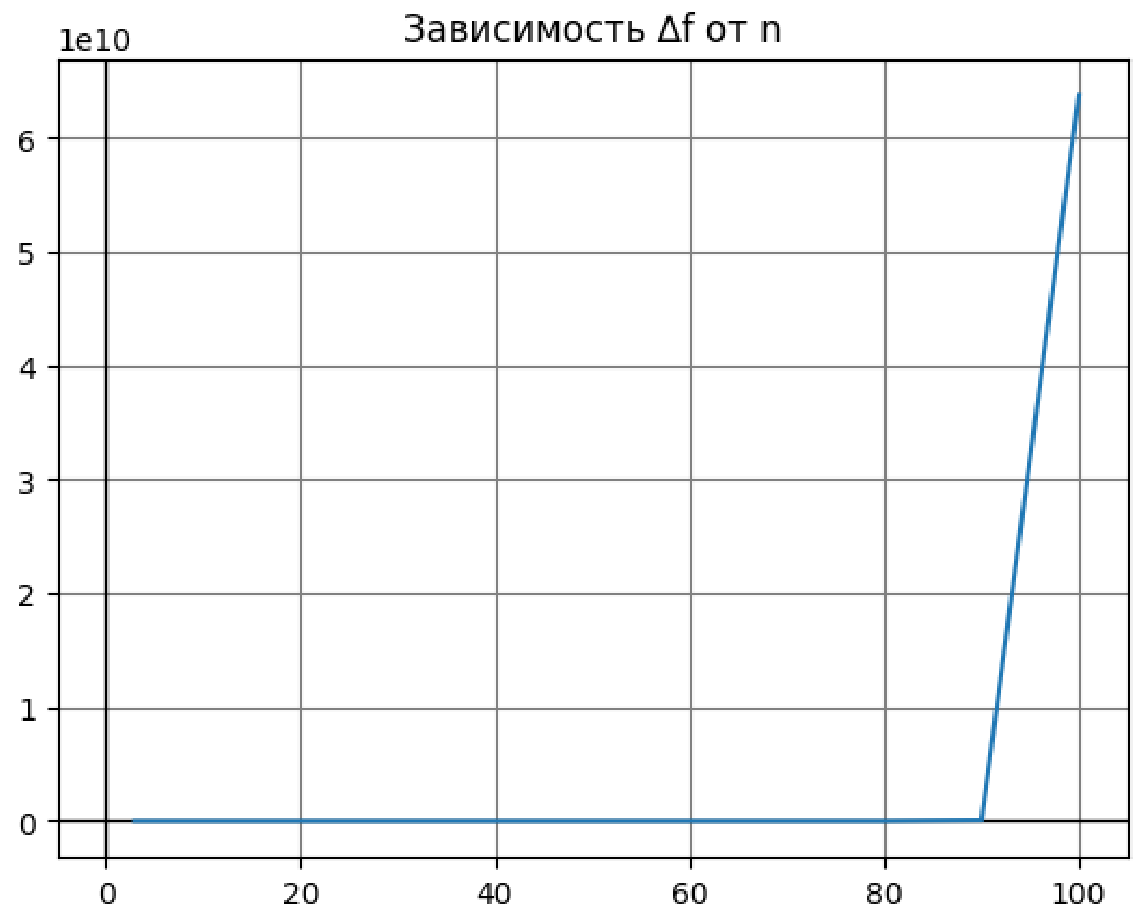
```
[83]
...  n       Δf                      δf                      r
     ===============================================================================
     3       0.004231214614800596    0.5536481630555677      84.82314099671875
     5       0.0001306298979678877   0.017092728597853366    1413.719016611979
     10      7.534018961474764e-08   9.858152181304314e-06   2409748.3237704188
     20      3.467226505904364e-13   4.536814509896921e-11   12326651581786.928
     30      2.4368418394260516e-10  3.188571440777096e-08   8.154601915874668e+19
     40      1.1635587585101526e-07  1.5224993953343161e-05  6.0211666204295617e+26
     50      5.837802913699619e-05   0.007638678615224421    4.7270954242802546e+33
     60      0.07644614939803152     10.002865380272432      3.859532957196033e+40
     70      100.68610238702482      13174.627313114483      3.238219498087561e+47
     80      56860.06988574314       7440055.89632245        2.7719133917367955e+54
     90      76568544.3865855        10018880583.17495       2.4094800044071997e+61
     100     63790678470.08707       8346915760664.261       2.1200367458332219e+68
```

▼ 3. Графики зависимостей

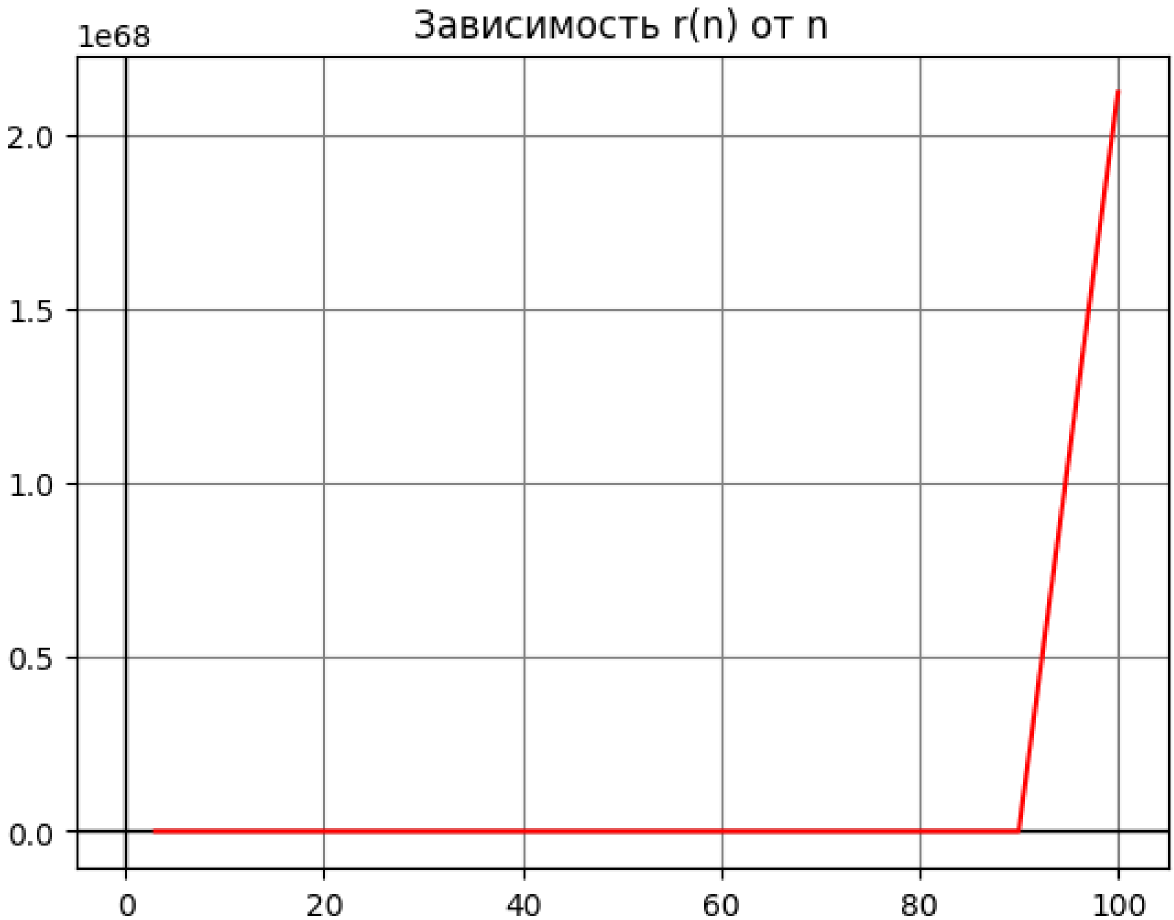
Для построения графиков зависимостей я использую словари, созданные в переписанном кодике. Мне стало лень вводить индексацию, как-то её связывать с пробегом по n в ns

Собственно вот кодики и их результаты:

```
plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)
plt.grid(True, color='gray')

plt.title('Зависимость Δf от n')
plt.plot(ns, list(DeltaErr.values()))
```




```
plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)
plt.grid(True, color='gray')

plt.title('Зависимость r(n) от n')
plt.plot(ns, list(rs.values()), color='red')
```



▼ 4. Выводы

По графикам (да и по табличке тоже) видно, что ошибки возрастают. Теперь более подробно о поведении практической и теоретической ошибок.

Практическая ошибка Δf убывает до n=20, принимая сравнительно маленькие значения, после чего начинает значительно возрастать и при сотне точек значение становится крайне большим. Такое дикое накопление ошибки происходит из-за того, что xs[i] - xs[j] (находясь в знаменателе) начинает принимать значения, стремящиеся к нулю, т.к. точки с увеличением n становятся всё ближе и ближе к друг другу. Ну а по базовым свойствам математики 1 / маленькое = большое.

Теоретическая ошибка r(n) имеет тенденцию показательного роста (исходя из выведенной в десмосе формулы https://www.desmos.com/calculator/iq0ma4i7ge). Это происходит по причине стремления функции x^(-t) к бесконечности при |x| <= 1 (ряд Тейлора для логарифма в нуле не сходится, т.е. не существует). Мой промежуток так вообще всецело лежит левее единицы, потому неудивительно что r(n) возрастает с количеством точек.