# Вычмат лаба5

## “Интерполяция сплайнами”

Вариант 19

Б9122-02.03.01сцт 2 группа

## Цель

1) Закодить генерацию сплайна с помощью метода моментов

2) Построить табличку абсолютных и относительных ошибок (зависимость от количества узлов)

3) Построить график зависимости абсолютной ошибки от количества узлов

4) Сделать вывод о поведении ошибки

## Ход работы

▼ 1) Генерация сплайна

Я выбрал второе краевое условие, т.к. при нём матрица системы выглядит наиболее простой

Второе условие подразумевает использование второй производной, потому объявляю её вместе со своей функцией:

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

brd = [0.4, 0.9]

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

def df(x):
    return 2*x + 0.434294481903/x

def ddf(x): # для 2-го краевого условия
    return 2 - 0.434294481903/x/x

```

Сплайн по факту есть условная функция, которая на каждом узловом отрезке принимает новые коэффициенты для кубического полинома. Значит необходимо получить три массива значений. Для этого существует ***метод монотонной прогонки***

Его я вывел в отдельную функцию:

```python

def Schplein_build():

    # === Метод монотонной прогонки и его комплектующие ================= #

    C = [0] + [hs[i] / (hs[i] + hs[i+1]) for i in range (0,n-1)] + [0]
    A = [0] + [hs[i+1] / (hs[i] + hs[i+1]) for i in range (0,n-1)] + [0]
    B = [1] + [2]*(n-1) + [1]

    F = [ddf(brd[0])] + [6/(hs[i] + hs[i+1]) * ((ys[i+1]-ys[i])/hs[i+1] \
        - (ys[i]-ys[i-1])/hs[i]) for i in range(0, n-1)] + [ddf(brd[1])]
    
    α = [-C[0] / B[0]]
    β = [F[0] / B[0]]

    M = [ddf(brd[1])]

    for i in range(0, n-1):
        α.append(-C[i] / (α[i]*A[i] + B[i]))
        β.append((F[i] - β[i]*A[i]) / (α[i]*A[i] + B[i]))

    for i in range(0, n-1):
        M.append(α[n-i-1]*M[i] + β[n-i-1])
    M.append(ddf(brd[0]));  M = M[::-1]

    # ================================================================= #

    a = M[:-1:]
    b = [(M[i+1] - M[i])/hs[i+1] for i in range(0, n)]
    c = [(ys[i+1] - ys[i])/hs[i+1] - hs[i+1]/6 * (2*M[i] + M[i+1]) for i in range(0, n)]

    return [a, b, c]

```

Т.к. практически вся матрица состоит из нулей, мне достаточно хранить три списка элементов матрицы. По двум боковым диагоналям располагаются коэффициенты, задействующие шаги, по главной диагонали - только двойки с единицами на концах. (По лекциям Татьяны Владимировны)

Ещё нужно хранить список свободных коэффициентов, которые в результате преобразований для метода моментов и второго краевого условия принимают монструозный вид. В начале и в конце списка значения вторых производных функции с концов данного отрезка

Я кстати не сказал о том, что функция есть $f(x) = x^2 + \lg x$, отрезок - $[0.4,\ 0.9]$

Затем применяется обратный ход монотонной прогонки, т.к. максимально “удобно” формулы построены на основе последующих значений, которые мне как-бы не известны

Для него я итеративно вычисляю коэффициенты $\alpha$ и $\beta$. На их основе уже и прогоняем значения моментов в цикле

Поскольку ход был обратным, список нужно перевернуть, что благо в питоне делается крайне легко. Затем по формулам из методички собираем массив коэффициентов a, b и c

Функция возвращает этот массив массивов, который затем присваивается глобальной переменной, используемой непосредственно в функции самого сплайна:

```

def Schplein(x, i):
    return ys[i] + L[2][i]*(x-xs[i]) + L[0][i]*(x-xs[i])**2 /2 + L[1][i]*(x-xs[i])**3 /6

```

Несмотря на то, что я писал код для общего случая, сетку я всё равно беру равномерной

Хочу показать работу кода на примере 20-узлового сплайна:

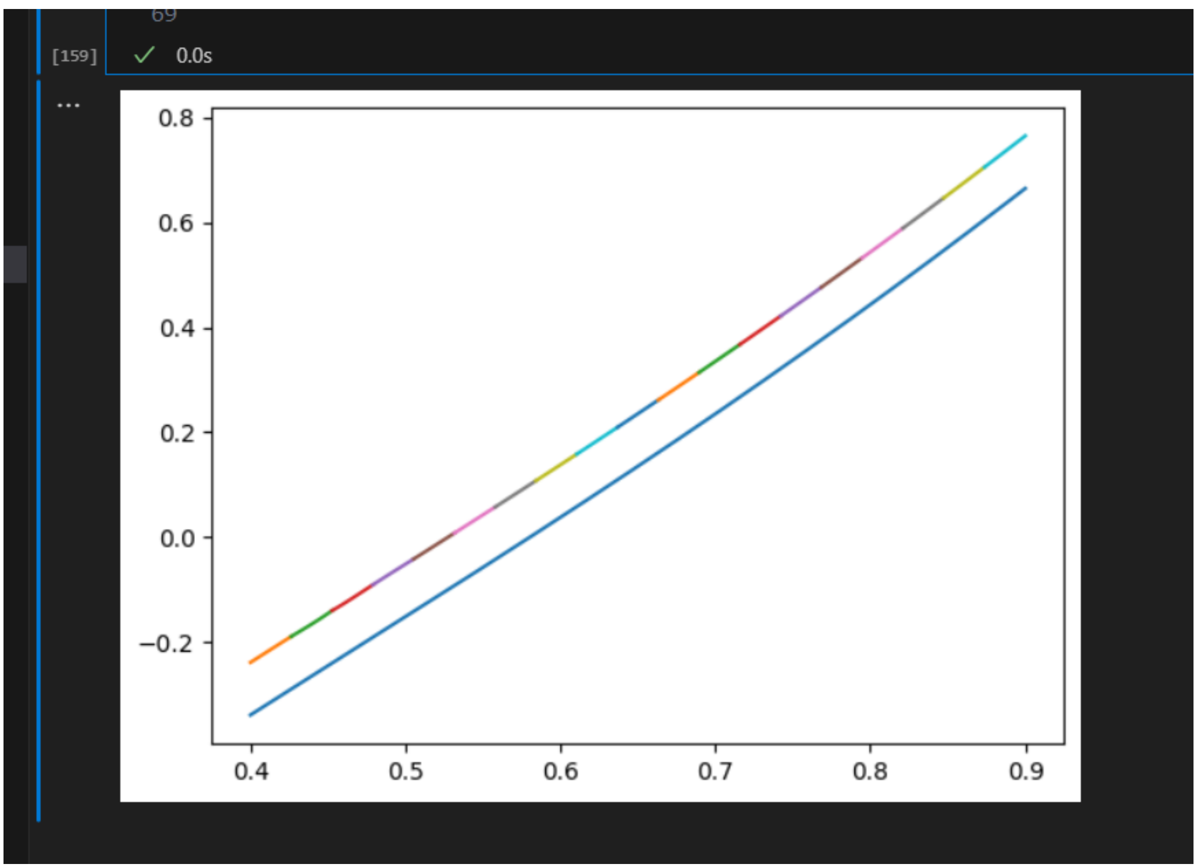
```python
xs = np.linspace(*brd, 20)
n = len(xs)-1
hs = [abs(xs[_] - xs[_-1]) for _ in range(0,n+1)]
ys = [f(_) for _ in xs]

L = Schplein_build()
# plt.ylim(-1/4,1)

x__ = np.linspace(*brd, 200)
plt.plot(x__, f(x__)-0.1)

for i in range(n):
    xl = np.linspace(xs[i], xs[i+1], 10)
    yl = Schplein(xl, i)

    plt.plot(xl, yl)
```



Синий график ниже - это исходная функция, смещенная на 0.1 вниз

Получается такая вот лакричная конфета

▼ 2) Таблица ошибок

Скрин для листинга кода:

```python
def norm(lst):
    return max(list(map(np.fabs, lst)))

ns = [3,5,10,20,30,40,55,70,85,100]
Ds = []; ds = []

print("n	Δ                       δ")
print("="*55)

for ni in ns:
    
    yspl = []
    xs = np.linspace(*brd, ni)
    n = ni-1
    hs = [abs(xs[_] - xs[_-1]) for _ in range(0,n+1)]
    ys = [f(_) for _ in xs]

    L = Schplein_build()

    for i in range(n):
        xl = np.linspace(xs[i], xs[i+1], 10)
        yl = Schplein(xl, i)

        yspl += [*yl]
    
    other_ys = f(np.linspace(*brd, n * 10))

    NormSpl = norm(np.array(yspl) - other_ys)
    NormFunc = norm(other_ys)

    Ds.append(NormSpl); ds.append(NormSpl/NormFunc*100)

    print(ni, NormSpl, NormSpl/NormFunc*100, sep='\t')

```

Поскольку каждый раз я увеличиваю количество узлов, мне необходимо перестраивать массив значений для просчёта норм функций. На самом деле в моём случае этого можно не делать, достаточно взять самый последний кусок сплайна и посчитать норму на нём. Я так не сделал в силу общего случая, на который я писал код.

Результат выполнения кода:

```
n       Δ                       δ
=======================================================
3       0.02657652239531516     3.477498577619378
5       0.020834876998188034    2.7262127846661173
10      0.011019545698680688    1.4418912272709108
20      0.005629692032292688    0.7366368610433391
30      0.00377302924772549     0.4936952866562626
40      0.002836397613686592    0.37113842512731626
55      0.0020664905636801345   0.2703972283871234
70      0.0016252243827840074   0.21265820243057781
85      0.0013392201969057993   0.17523497847408387
100     0.0011388024326822732   0.1490106109797188
```

Видно, что ошибки стабильно жмутся к нулю

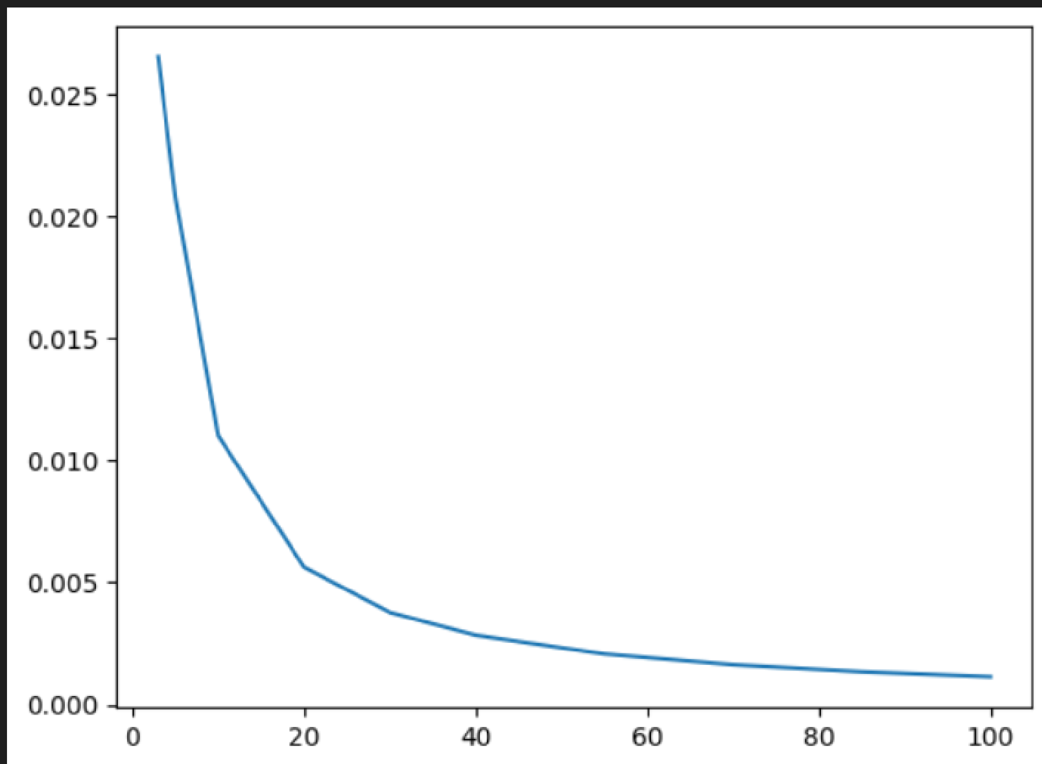
Мне стало интересно, что будет происходить на миллионе узлов. Собсна всё как и ожидалось:

```
100     0.0011388024326822732   0.1490106109797188
1000000 1.1412744782557382e-07  1.4933407448023497e-05
```

▼ 3) График зависимости

```
[167]
plt.plot(ns, Ds)
# plt.plot(ns, ds)
✓ 0.0s

[<matplotlib.lines.Line2D at 0x2bd95f8d910>]
```




Взяв вычисленные ошибки из предыдущего пункта видим, что ошибка имеет некий гиперболический характер изменения

▼ 4) Вывод об ошибке

Как я уже написал в предыдущем пункте, ошибка принимает гиперболический характер изменения. Это обусловлено тем, что сплайны так явно не обладают делением на малые значения, как тот же полином Лагранжа. В нём с увеличением количества узлов их разница стремилась к нулю, что по сути вызывало деление на бесконечно малое значение.

Увеличение точности с увеличением количества узлов в сплайнах обеспечивает некую полную непрерывность функции за счёт множества условий, которые мы накладываем на интерполяционную функцию. Устремление количества узлов в “бесконечность” порождает тот факт, что интерполяция происходит по факту на бесконечно малых масштабах, на которых даже интерполяция прямыми работает отменно. Потому неудивительно, что ошибка так резко уходит к нулю.