Федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

Лабораторная работа 5: «Интерполяция функции»
«Вычислительная математика»
вариант 15

Выполнила:
Сенина Мария Михайловна
группа P3112
Преподователь:
Малышева Татьяна Алексеевна

ITMO UNIVERSITY

2022г.
г.Санкт-Петербург

1 Цель работы

Решить задачу интерполяции, найти значения функции при заданных значениях аргумента, отличных от узловых точек. Для исследования использовать:

1. многочлен Лагранжа;
2. многочлен Ньютона;
3. многочлен Гаусса.

2 Порядок выполнения работы

2.1 Для исследования использовать

- многочлен Лагранжа
- многочлен Ньютона

2.2 Вычислительная реализация задачи:

1. Используя первую или вторую интерполяционную формулу Ньютона, первую или вторую интерполяционную формулу Гаусса вычислить значения функции при данных значениях аргумента (для значения X_1 и $_2$, см. табл. 1 - 4).
2. Построить таблицу конечных разностей.
3. Подробные вычисления привести в отчете.

2.3 Программная реализация задачи:

1. Исходные данные задаются в виде: а) набора данных (таблицы x,y), б) на основе выбранной функции (например, sinx).
2. Вычислить приближенное значение функции для заданного значения аргумента, введенного с клавиатуры, указанными методами (см. табл.5).
3. Построить графики заданной функции с отмеченными узлами интерполяции и интерполяционного многочлена Ньютона/Гаусса (разными цветами).

3 Рабочие формулы

3.1 Вычилительная часть

```
x = [1.10, 1.25, 1.40, 1.55, 1.70, 1.85, 2.00]
y = [0.2234, 1.2438, 2.2644, 3.2984, 4.3222, 5.3516, 6.3867]
x1 = 1.875
x2 = 1.575
```

Вычислим конечные разности по рекурентной формуле:

$$f(x_0, x_1) = \frac{f(x_1) - f(x_0)}{x_1 - x_0}, f(x_i, x_{i+1}) = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}, f(x_i, x_{i+1}) = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}$$

$$f(x_i, x_{i+1}, ..., x_{i+k}) = \frac{f(x_{i+1}, ..., x_{i+k}) - f(x_i, ..., x_{k-1})}{x_{i+1} - x_i} \tag{1}$$

	y_0	Δ	Δ^2	Δ^3	Δ^4	Δ^5	Δ^6
x_0	0.2234	1.0204	0.0002	0.0132	-0.0368	0.0762	-0.1313
x_1	1.2438	1.0206	0.0134	-0.0236	0.0394	-0.0551	0.0000
x_2	2.2644	1.0340	-0.0102	0.0158	-0.0157	0.0000	0.0000
x_3	3.2984	1.0238	0.0056	0.0001	0.0000	0.0000	0.0000
x_4	4.3222	1.0294	0.0057	0.0000	0.0000	0.0000	0.0000
x_5	5.3516	1.0351	0.0000	0.0000	0.0000	0.0000	0.0000
x_6	6.3867	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

Исходя из таблицы строим многочлен Ньютона для первой точки $x_1 = 1.875$ по формуле:

$$N_n(x) = y_n + t\Delta y_{n-1} + \frac{t(t+1)}{2!}\Delta^2 y_{n-2} + \cdots + \frac{t(t+1)(t+n-1)}{n!}\Delta^n y_0, t = \frac{(x-x_n)}{h}$$

$$N_6(1.875) = 6.3867 + (-0.833) * 1.035/1 + (-0.139) * 0.006/2+$$

$$(-0.162) * 0.0/6 + (-0.351) * -0.016/24 + (-1.112) * -0.055/120+$$

$$(-4.632) * -0.131/720 = 5.5253$$

Используем обратную формулу Ньютона, потому что точка ближе к концу отрезка.

Аналогично вычисляем значение в точке $x_2 = 1.575$:

$$N_6(1.575) = 6.3867 + (-2.833) * 1.035/1 + (5.194) * 0.006/2+$$

$$(-4.329) * 0.0/6 + (-0.721) * -0.016/24 + (-0.842) * -0.055/120+$$

$$(-1.824) * +(-0.131)/720 = 3.4698$$

3.2 Программная часть

Формула Ньютона:

Чтобы вычислить интрополяционное значение функции в точке, если интервалы между точками не одинаковые (не равны константе)

$$N_n(x) = f(x_0) + \sum_{k=1}^{n} f(x_0, x_1, ...x_k)\prod_{j=0}^{k-1}(x - x_j)$$

Формула Ньютона: Чтобы вычислить интрополяционное значение функции в точке, если интервалы между точками одинаковые

Интрополяция вперёд:

$$N_n(x) = y_n + t\Delta y_{n-1} + \frac{t(t-1)}{2!}\Delta^2 y_{n-2} + \cdots + \frac{t(t+1)(t-n+1)}{n!}\Delta^n y_0, t = \frac{(x-x_0)}{h}$$

Интрополяция назад:

$$N_n(x) = y_n + t\Delta y_{n-1} + \frac{t(t+1)}{2!}\Delta^2 y_{n-2} + \cdots + \frac{t(t+1)(t+n-1)}{n!}\Delta^n y_0, t = \frac{(x-x_n)}{h}$$

Чтобы интрополировать многочлен с помощью метода Лагранжа, нужно вычислить многочлен Лагранжа:

$$L_n(x) = \sum_{i=0}^{n} y_i \prod_{j=0, j\neq i}^{n}(\frac{x - x_j}{x_i - x_j})$$

4 Листинг программы

Ссылка репозиторий с кодом всей программы

4.1 Метод Ньютона

```python
def newton_interpolation(x, y, point):

    if check_h_is_const(x):
        result = newton_const_h(x, y, point)
    else:
        result = newton_not_const_h(x, y, point)
    return result

#----------------CONST H----------------
def newton_const_h(x, y, point):
    n = len(x)
    h = abs(x[1] - x[0])
    finite_differences = count_finite_differences_const_h(x, y)
    if point <= x[n // 2]: # if point is closer to beginning we use FIRST Formula (forward)
        x0 = search_for_x0(x, point)
        t = (point - x[x0]) / h

        result = finite_differences[x0][0]
        for i in range(1, n):
            result += (t_forward(t, i) * finite_differences[x0][i]) / factorial(i)
    else:                  # if point is closer to beginning we use SECOND Formula (back)
        t = (point - x[n - 1]) / h

        result = finite_differences[n - 1][0]
        for i in range(1, n):
            result += (t_back(t, i) * finite_differences[n - i - 1][i]) / factorial(i)

    return result

def count_finite_differences_const_h(x, y):
    n = len(x)
    finite_differences = [[0]*n for _ in range(n)]
    for i in range(n):
        finite_differences[i][0] = y[i]

    k = 1
    while k <= n:
        for i in range(n-k):
            finite_differences[i][k] = finite_differences[i+1][k-1] - finite_differences[i][k-1]
        k += 1
    return finite_differences

def t_forward(t_0, i):
    t = t_0
    for j in range (1, i):
        t*= t_0 - j
    return t

def t_back(t_0, i):
    t = t_0
    for j in range (1, i):
        t*= t_0 + j
    return t

def search_for_x0(x, point):
    n = len(x)
```

```
    x0 = n - 1
    for i in range(n):
        if point <= x[i]:
            x0 = i - 1
            break
    if x0 < 0:
        x0 = 0
    return x0

#----------------NOT CONST H----------------
def newton_not_const_h(x, y, point):
    n = len(x)
    finite_differences = count_finite_differences_not_const_h(x, y)
    [print(finite_differences[i]) for i in range(n)] #removeme
    #f(x_0) + sum_{k=1}^n f(x_0, x_1, ... x_k) * \prod_{i = 0}^{k-1}(x-x_i)
    return y[0] + sum([finite_differences[0][k]* sum([point - x[i] for i in range(k - 1)]) for k in range(n)])

def count_finite_differences_not_const_h(x, y):
    n = len(x)
    finite_differences = [[0]*n for _ in range(n)]
    print()
    for i in range(n):
        finite_differences[i][0] = y[i]
    k = 1
    while k <= n:
        for i in range(n - k):
            finite_differences[i][k] = (finite_differences[i + 1][k - 1] - finite_differences[i][k - 1]) / (x[i + k] - x[i])
        k += 1
    return finite_differences


# Method to check if we work with data with equally spaced nodes or not
def check_h_is_const(x):
    h = abs(x[1]-x[0])
    e = 0.0001 #verification accuracy
    for i in range(len(x) - 1):
        if abs(abs(x[i+1] - x[i]) - h) > e:
            return False
    return True
```

4.2 Метод Лагранжа

```
def lagrange_interpolation(x, y, point):
    result = 0
    n = len(x)
    for i in range(n):
        c1 = c2 = 1
        for j in range(n):
            if i != j:
                c1 *= point - x[j]
                c2 *= x[i] - x[j]
        result += y[i] * c1 / c2
    return result
```

5 Примеры и результаты работы программы

5.1 Пример ввода данных таблицей

Ввод:

```
How you would like to enter data?
1—————————————————— table
2—————————————————— choose function
1
Enter points like this:
x_1 y_1
x_2 y_2
...
Type '#' to finish.
1 2
2 4
3 9
4 16
5 9
6 4
#
Entered data: [(1.0, 2.0), (2.0, 4.0), (3.0, 9.0), (4.0, 16.0), (5.0, 9.0), (6.0, 4.0)]
Choose interpolation method:
1—————————————————— Newton
2—————————————————— Lagrange
3—————————————————— both
3
Enter point to interpolate:
2.5
```

Вывод:

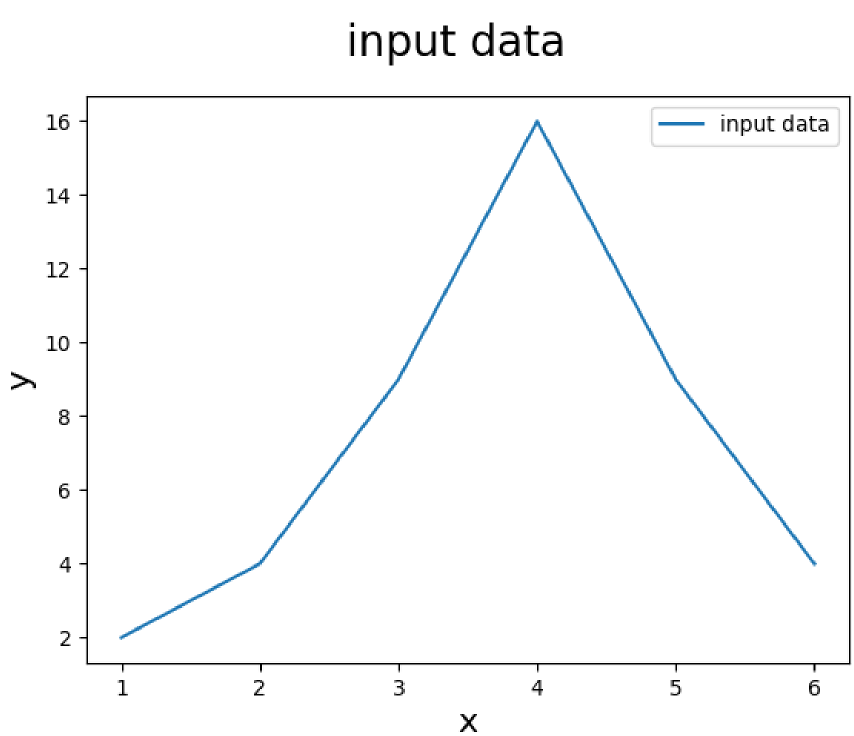

Рис. 1: График, чтобы было удобнее выбирать точку, в которой мы хотим интрополировать функцию

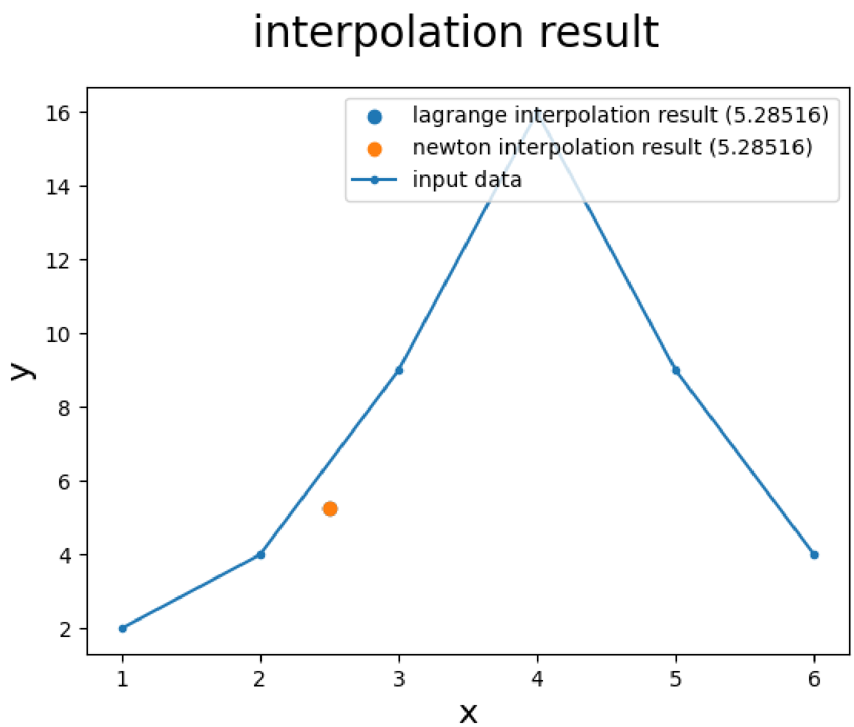

Рис. 2: Предсказание значения для конкретной точки

5.2 Пример ввода данных функцией

```
How you would like to enter data?
1 ——————————————— table
2 ——————————————— choose function
0 6
Value has be integer! Try again from new line...
2
Choose function:
1 ——————————————— x^2
2 ——————————————— 1/x
3 ——————————————— sin(x)
3
Enter segment's bounds (enter the values separated by a space):
0 7
Select the number of interpolation nodes:
10
Entered data: [(0.0, 0.0), (0.7, 0.644217687237691), (1.4, 0.9854497299884601), (2.0999999999999996, 0.8632093666488739), (2.8, 0
Choose interpolation method:
1 ——————————————— Newton
2 ——————————————— Lagrange
3 ——————————————— both
3
Enter point to interpolate:
3
```

Выюод:

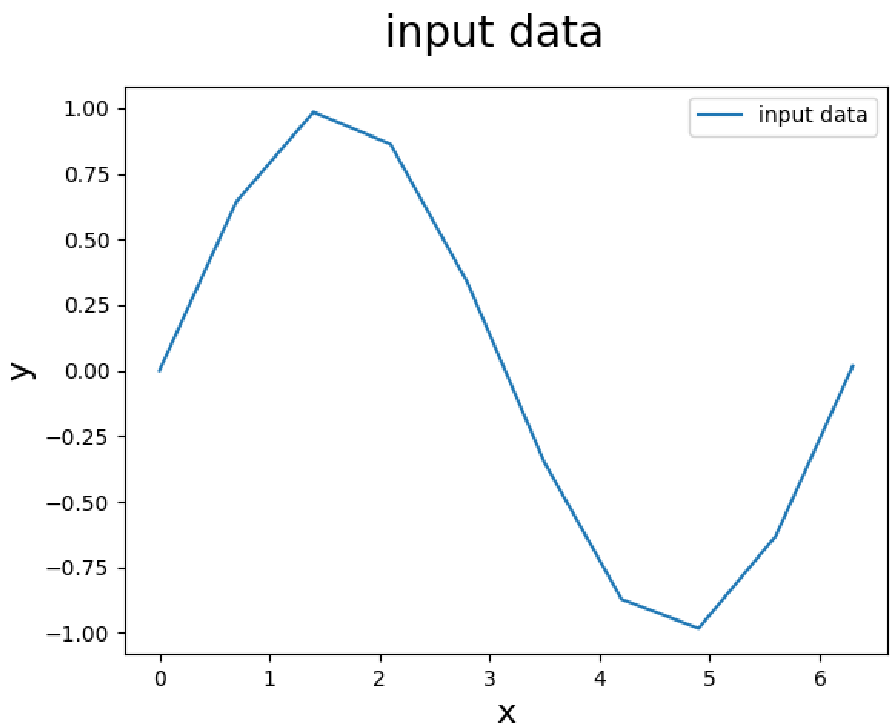

Рис. 3: График, чтобы было удобнее выбирать точку, в которой мы хотим интраполировать функцию

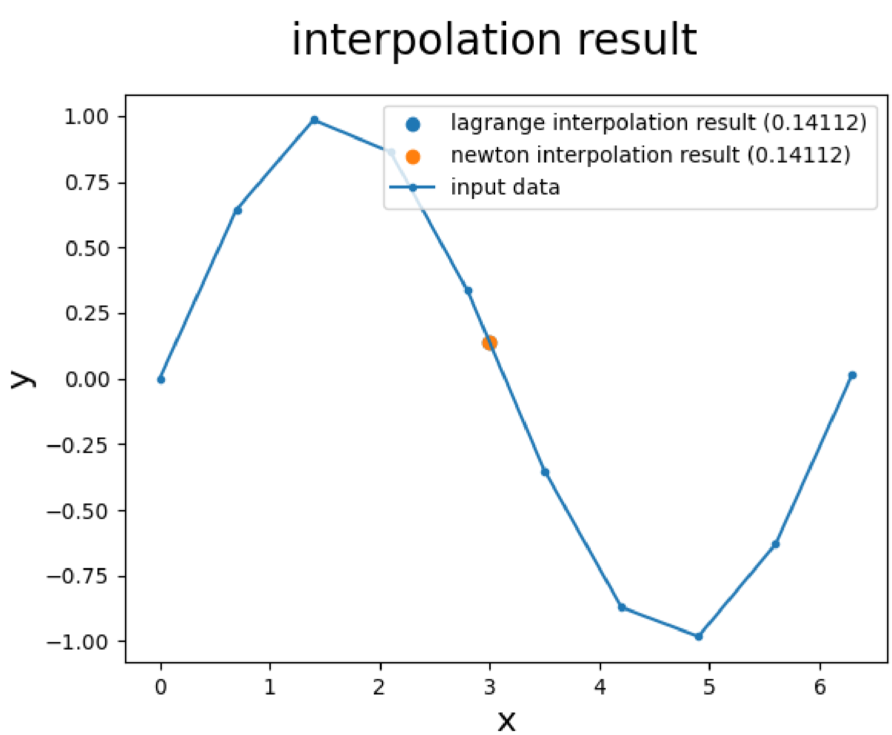

Рис. 4: Предсказание значения для конкретной точки

6 Выводы

В этой лабораторной работе я научилась интраполировать функции заданные в табличном виде. Разобралась с тем, как применять метод Лагранжа, а так же метод Ньютона для равностоящих узлов и для случая, когда расстояния между точками в таблице не одинаковые.