Университет ИТМО

Вычислительная математика

Лабораторная работа №5

ФИО студента: Готовко Алексей Владимирович
Направление подготовки: 09.03.04 (СППО)
Учебная группа: P32101
ФИО преподавателей:
Малышева Т.А.
Рыбаков С.Д.

Санкт-Петербург
2023г.

1 Цель работы

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

2 Вычислительная реализация задачи

Точки по варианту:

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

$$X_1 = 1.121, X_2 = 1.482$$

2.1 Таблица конечных разностей

	y_i	Δy_i	$\Delta^2 y_i$	$\Delta^3 y_i$	$\Delta^4 y_i$	$\Delta^5 y_i$	$\Delta^6 y_i$
$x_0(x_{-3})$	0.2234	1.0204	0.0002	0.0132	-0.0368	0.2562	-1.2113
$x_1(x_{-2})$	1.2438	1.0206	0.0134	-0.0236	0.2194	-0.9551	
$x_2(x_{-1})$	2.2644	1.034	-0.0102	0.1958	-0.7357		
$x_3(x_0)$	3.2984	1.0238	0.1856	-0.5399			
$x_4(x_1)$	4.3222	1.2094	-0.3543				
$x_5(x_2)$	5.5316	0.8551					
$x_6(x_3)$	6.3867						

2.2 Интерполяция по Ньютону

Так как $X_1 = 1.121 < \frac{(x_6+x_0)}{2} = 1.55$, то будем интерполировать вперед.

$$t = \frac{X_1 - x_0}{h} = \frac{1.121 - 1.10}{0.15} = 0.14$$

$$N_6(X_1) = y_0 + t\Delta y_0 + \frac{t(t-1)}{2}\Delta^2 y_0 + \frac{t(t-1)(t-2)}{3!}\Delta^3 y_0 + \frac{t(t-1)(t-2)(t-3)}{4!}\Delta^4 y_0 + \\ + \frac{t(t-1)(t-2)(t-3)(t-4)}{5!}\Delta^5 y_0 + \frac{t(t-1)(t-2)(t-3)(t-4)(t-5)}{6!}\Delta^6 y_0 \approx 0.3932$$

2.3 Интерполяция по Гауссу

Так как $X_2 = 1.482 < \frac{(x_6+x_0)}{2} = 1.55$, то будем использовать вторую интерполяцию Гаусса.

$$t = \frac{X_2 - x_0}{h} = \frac{X_2 - a}{h} = \frac{1.482 - 1.55}{0.15} \approx -0.4533$$

$$P_6(X_2) = y_0 + t\Delta y_{-1} + \frac{t(t+1)}{2}\Delta^2 y_{-1} + \frac{(t+1)t(t-1)}{3!}\Delta^3 y_{-2} + \frac{(t+2)(t+1)t(t-1)}{4!}\Delta^4 y_{-2} + \\ + \frac{(t+2)(t+1)t(t-1)(t-2)}{5!}\Delta^5 y_{-3} + \frac{(t+3)(t+2)(t+1)t(t-1)(t-2)}{6!}\Delta^6 y_{-3} \approx 2.8306$$

3 Программная реализация задачи

```python
# interpolation_methods.py
from math import factorial

from interpolation_util import InterpolationData
from io_util import print_warn


def lagrange(data: InterpolationData, target_x):
    points = data.point_arr
    n = data.points_cnt

    val = 0
    for i in range(n):
        multiplier = 1
        for j in range(n):
            if i == j:
                continue
            multiplier *= (target_x - points[j].x) / (points[i].x - points[j].x)
        val += multiplier * points[i].y

    return val


def newton(data: InterpolationData, target_x):
    t: float
    is_backwards: bool
    if target_x < (data.max_x + data.min_x) / 2:
        is_backwards = False
        t = (target_x - data.min_x) / data.x_step
    else:
        is_backwards = True
        t = (target_x - data.max_x) / data.x_step

    val = 0
    t_multiplier = 1
    diff = data.differences

    idx = -1 if is_backwards else 0
    t_delta = 1 if is_backwards else -1
    for i in range(data.points_cnt):
        val += diff[i][idx] * t_multiplier
        t_multiplier = t_multiplier * (t + t_delta * i) / (i + 1)

    return val


def stirling(data: InterpolationData, target_x):
    n = data.points_cnt // 2
    t = (target_x - data.point_arr[n].x) / data.x_step

    if abs(t) > 0.25:
        print_warn(f"|t| = {round(abs(t), 4)} > 0.25, Stirling's interpolation may work improperly")

    diff = data.differences

    val = diff[0][n] + t * (diff[1][n] + diff[1][n - 1]) / 2 + t ** 2 / 2 * diff[2][n - 1]
    t_multiplier = t * (t ** 2 - 1) / 6
    iter_odd = True
```

```python

    i = 3
    for j in range(1, n):
        if iter_odd:
            dy = (diff[i][n - j - 1] + diff[i][n - j - 2]) / 2
        else:
            dy = diff[i][n - j - 1]

        val += dy * t_multiplier
        if iter_odd:
            t_multiplier *= t / i
        else:
            t_multiplier *= t ** 2 - (i + 1) ** 2 / i
        iter_odd = not iter_odd
        i += 1

    return val


def bessel_calc(u, n):
    if n == 0:
        return 1

    var = u
    for i in range(1, n // 2 + 1):
        var *= u - i

    for i in range(1, n // 2):
        var *= u + i

    return var


def bessel(data: InterpolationData, target_x):
    t = (target_x - data.point_arr[data.points_cnt // 2].x) / data.x_step

    if not 0.25 <= abs(t) <= 0.75:
        print_warn(f"|t| = {round(abs(t), 4)} not in [0.25, 0.75],"
                   "Bessel's interpolation may work improperly")

    n = data.points_cnt
    x = [p.x for p in data.point_arr]

    y = [[0 for _ in range(n)] for _ in range(n)]

    for i in range(n):
        y[i][0] = data.point_arr[i].y

    for i in range(1, n):
        for j in range(n - i):
            y[j][i] = y[j + 1][i - 1] - y[j][i - 1]

    val = (y[2][0] + y[3][0]) / 2

    if n % 2 == 1:
        k = n // 2
    else:
        k = n // 2 - 1

    u = (target_x - x[k]) / (x[1] - x[0])

```

```python
    for i in range(1, n):
        if i % 2 == 1:
            val += ((u - 0.5) * bessel_calc(u, i - 1) * y[k][i]) / factorial(i)
        else:
            val += (bessel_calc(u, i) * (y[k][i] + y[k - 1][i]) / (factorial(i) * 2))
            k -= 1

    return val

```

Полный код программы доступен по ссылке.

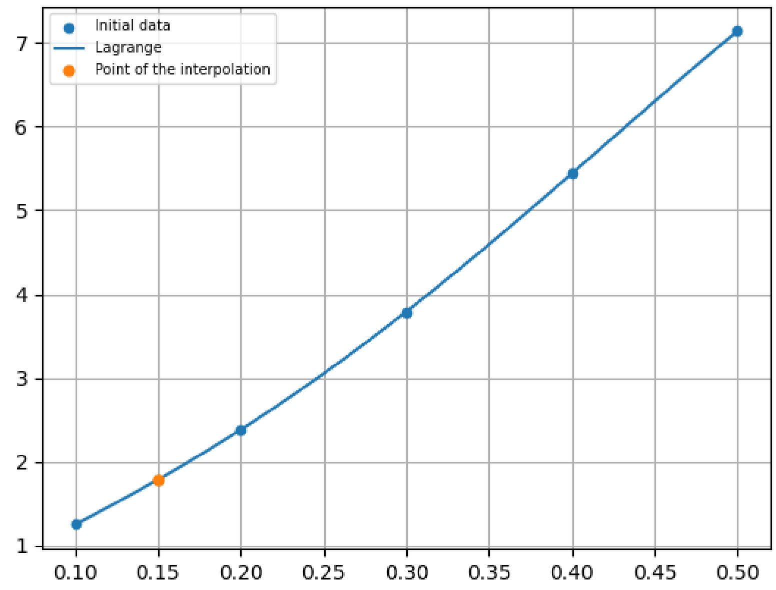
4 Результат работы программы

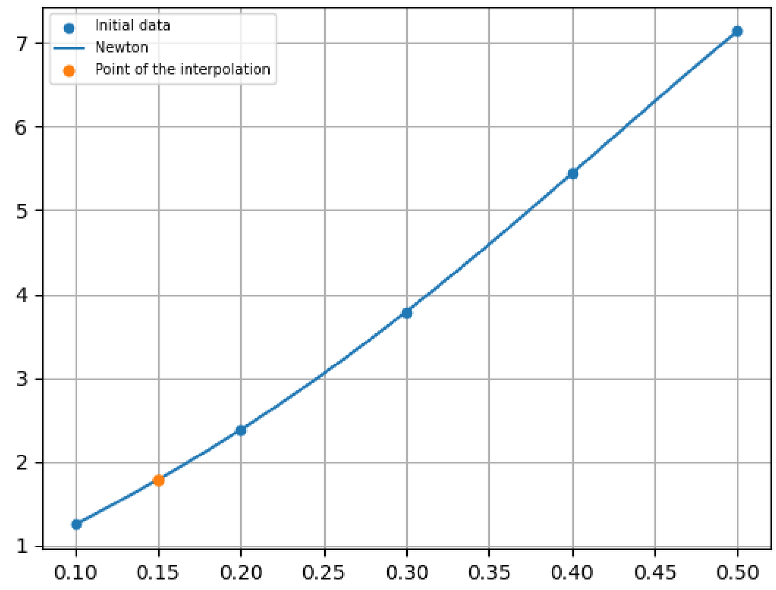
```
# Input:
# X     Y
# 0.1   1.25
# 0.2   2.38
# 0.3   3.79
# 0.4   5.44
# 0.5   7.14

Finite differences
-------------------------------------------------------------------------------------
|     |               |               |               |               |               |               |
|  i  |      yi       |      dyi      |     d2yi      |     d3yi      |     d4yi      |
|_____|_______________|_______________|_______________|_______________|_______________|
|  0  |     1.250     |     1.130     |     0.280     |    -0.040     |    -0.150     |
|  1  |     2.380     |     1.410     |     0.240     |    -0.190     |
|  2  |     3.790     |     1.650     |     0.050     |
|  3  |     5.440     |     1.700     |
|  4  |     7.140     |

|t| = 1.5 > 0.25, Stirling's interpolation may work improperly
|t| = 1.5 not in [0.25, 0.75], Bessel's interpolation may work improperly
Newton's method     :   1.7834
Lagrange's method   :   1.7834
Stirling's method   :   1.8009
Bessel's method     :   1.8039

```





5 Вывод

Для наиболее оптимального распределения человеческих ресурсов и поддержания количества нервных клеток в головном мозгу, настоятельно рекомендуется использовать готовые библиотеки, содержащие наиболее эффективные реализации алгоритмов, вместо самостоятельной реализации оных.

Иными словами, вообще лучше зачиллиться и не изобретать велосипед.