Университет ИТМО
МФ КТиУ, Ф ПИиКТ

Лабораторная работа №5
Дисциплина «Вычислительная математика»

Интерполяция функции

Выполнил
Галлямов Камиль Рустемович

Преподаватель:
Машина Екатерина
Алексеевна

г. Санкт-Петербург
2024 г.

Вычислительная реализация задачи

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

n = 6

Таблица конечных разностей:

x_i	1.10	1.25	1.40	1.55	1.70	1.85	2.00
y_i	0.2234	1.2438	2.2644	3.2984	4.3222	5.3516	6.3867
Δy_i	1.0204	1.0206	1.034	1.0238	1.0294	1.0351	-
$\Delta^2 yi$	0.0002	0.0134	-0.0102	0.0056	0.0057	-	-
$\Delta^3 yi$	0.0132	-0.0236	0.0158	0.0001	-	-	-
$\Delta^4 yi$	-0.0368	0.0394	-0.0157	-	-	-	-
$\Delta^5 yi$	0.0762	-0.0551	-	-	-	-	-
$\Delta^6 yi$	-0.1313	-	-	-	-	-	-

$$\Delta^k y_i = \Delta^{k-1} y_{i+1} - \Delta^{k-1} y_i$$

$X_1 = 1.121$

$x_0 <= X_1 <= x_1$ => первая интерполяционная формула Ньютона

h = 1.25 – 1.10 = 0.15

t = (x – x0) / h = (x – 1.10) / 0.15

$$N_n(\mathrm{x}) = y_0 + t\Delta y_0 + \frac{\mathrm{t}(\mathrm{t}-1)}{2!}\Delta^2 y_0 + \frac{\mathrm{t}(\mathrm{t}-1)(\mathrm{t}-2)}{3!}\Delta^3 y_0 + \frac{\mathrm{t}(\mathrm{t}-1)(\mathrm{t}-2)(\mathrm{t}-3)}{4!}\Delta^4 y_0 + \frac{\mathrm{t}(\mathrm{t}-1)(\mathrm{t}-2)(\mathrm{t}-3)(\mathrm{t}-4)}{5!}\Delta^5 y_0 + \frac{\mathrm{t}(\mathrm{t}-1)(\mathrm{t}-2)(\mathrm{t}-3)(\mathrm{t}-4)(\mathrm{t}-5)}{6!}\Delta^6 y_0$$

x = 1.121=> t = (1.121 – 1.10) / 0.15 = 0.14

$$N_6(1.121) = 0.2234 + 0.14 * 1.0204 + \frac{0.14(0.14-1)}{2}0.0002 + \frac{0.14(0.14-1)(0.14-2)}{6}0.0132 + \frac{0.14(0.14-1)(0.14-2)(0.14-3)}{24}(-0.0368) + \frac{0.14(0.14-1)(0.14-2)(0.14-3)(0.14-4)}{120}0.0762 + \frac{0.14(0.14-1)(0.14-2)(0.14-3)(0.14-4)(0.14-5)}{720}(-0.1313)$$

=0.2234+0.142856+0+0.00049+0.00098+0.00156+0.00219 = **0.3715**

$X_2 = 1.482$
$a = x_3 = 1.55$
$x_2 < X_2 < x_3 \Rightarrow X_2 < a \Rightarrow$ вторая интерполяционная формула Гаусса

$h = 1.25 – 1.10 = 0.15$
$t = (x – a) / h = (x – 1.55) / 0.15$

Таблица конечных разностей:

i	-3	-2	-1	0	1	2	3
x_i	1.10	1.25	1.40	1.55	1.70	1.85	2.00
y_i	0.2234	1.2438	2.2644	3.2984	4.3222	5.3516	6.3867
Δy_i	1.0204	1.0206	1.034	1.0238	1.0294	1.0351	-
$\Delta^2 y_i$	0.0002	0.0134	-0.0102	0.0056	0.0057	-	-
$\Delta^3 y_i$	0.0132	-0.0236	0.0158	0.0001	-	-	-
$\Delta^4 y_i$	-0.0368	0.0394	-0.0157	-	-	-	-
$\Delta^5 y_i$	0.0762	-0.0551	-	-	-	-	-
$\Delta^6 y_i$	-0.1313	-	-	-	-	-	-

$$G_n(\mathrm{x}) = y_0 + t\Delta y_{-1} + \frac{\mathrm{t}(\mathrm{t}+1)}{2!}\Delta^2 y_{-1} + \frac{\mathrm{t}(\mathrm{t}+1)(\mathrm{t}-1)}{3!}\Delta^3 y_{-2} + \frac{\mathrm{t}(\mathrm{t}+2)(\mathrm{t}+1)(\mathrm{t}-1)}{4!}\Delta^4 y_{-2} + \frac{\mathrm{t}(\mathrm{t}+2)(\mathrm{t}+1)(\mathrm{t}-1)(\mathrm{t}-2)}{5!}\Delta^5 y_{-3} + \frac{\mathrm{t}(\mathrm{t}+3)(\mathrm{t}+2)(\mathrm{t}+1)(\mathrm{t}-1)(\mathrm{t}-2)}{6!}\Delta^6 y_{-3}$$

$x = 1.482 \Rightarrow t = (1.482 – 1.55) / 0.15 = -0.453$

$$G_6(\mathrm{x}) = 3.2984 - 0.453 * 1.034 + \frac{-0.453(-0.453+1)}{2}(-0.0102) + \frac{-0.453(-0.453+1)(-0.453-1)}{6}(-0.0236) + \frac{-0.453(-0.453+2)(-0.453+1)(-0.453-1)}{24}(-0.0394) + \frac{-0.453(-0.453+2)(-0.453+1)(-0.453-1)(-0.453-2)}{120}0.0762 + \frac{-0.453(-0.453+3)(-0.453+2)(-0.453+1)(-0.453-1)(-0.453-2)}{720}(-0.1313)$$

$= 3.2984 - 0.4684 + 0.00126 - 0.00141 - 0.00091 - 0.00087 + 0.00063 =$ **2.8287**

Программная реализация задачи

```python
from functools import reduce
from math import factorial

from matplotlib import pyplot as plt


def calc_lagrange_polynomial(xs, ys):
    n = len(xs) - 1
    f = lambda x: sum([ys[i] *
                       reduce(lambda a, b: a * b,
                              [(x - xs[j]) / (xs[i] - xs[j])
                               for j in range(n + 1) if i != j])
                       for i in range(n + 1)])
    return f


def calc_newton_divided_difference_polynomial(xs, ys):
    div_difs = []
    div_difs.append(ys[:])
    n = len(xs) - 1
    for k in range(1, n + 1):
        new = []
        last = div_difs[-1][:]
        for i in range(n - k + 1):
            new.append((last[i + 1] - last[i]) / (xs[i + k] - xs[i]))
        div_difs.append(new[:])
    print("divided differences:")
    for row in div_difs:
        print(*map(lambda a: round(a, 5), row), sep='\t')
    print('-' * 30)
    f = lambda x: ys[0] + sum([
        div_difs[k][0] * reduce(lambda a, b: a * b,
                                [x - xs[j] for j in range(k)])
        for k in range(1, n + 1)])
    return f


def calc_newton_finite_difference_polynomial(xs, ys):
    fin_difs = []
    fin_difs.append(ys[:])
    n = len(xs) - 1
    for k in range(1, n + 1):
        last = fin_difs[-1][:]
        fin_difs.append(
            [last[i + 1] - last[i] for i in range(n - k + 1)])
    print("finite differences:")
    for row in fin_difs:
        print(*map(lambda a: round(a, 5), row), sep='\t')
    print('-' * 30)
    h = xs[1] - xs[0]
    f = lambda x: ys[0] + sum([
        reduce(lambda a, b: a * b,
               [(x - xs[0]) / h - j for j in range(k)])
        * fin_difs[k][0] / factorial(k)
        for k in range(1, n + 1)])
    return f


def calc_gauss_polynomial(xs, ys):
    n = len(xs) - 1
    alpha_ind = n // 2
    fin_difs = []
    fin_difs.append(ys[:])
```

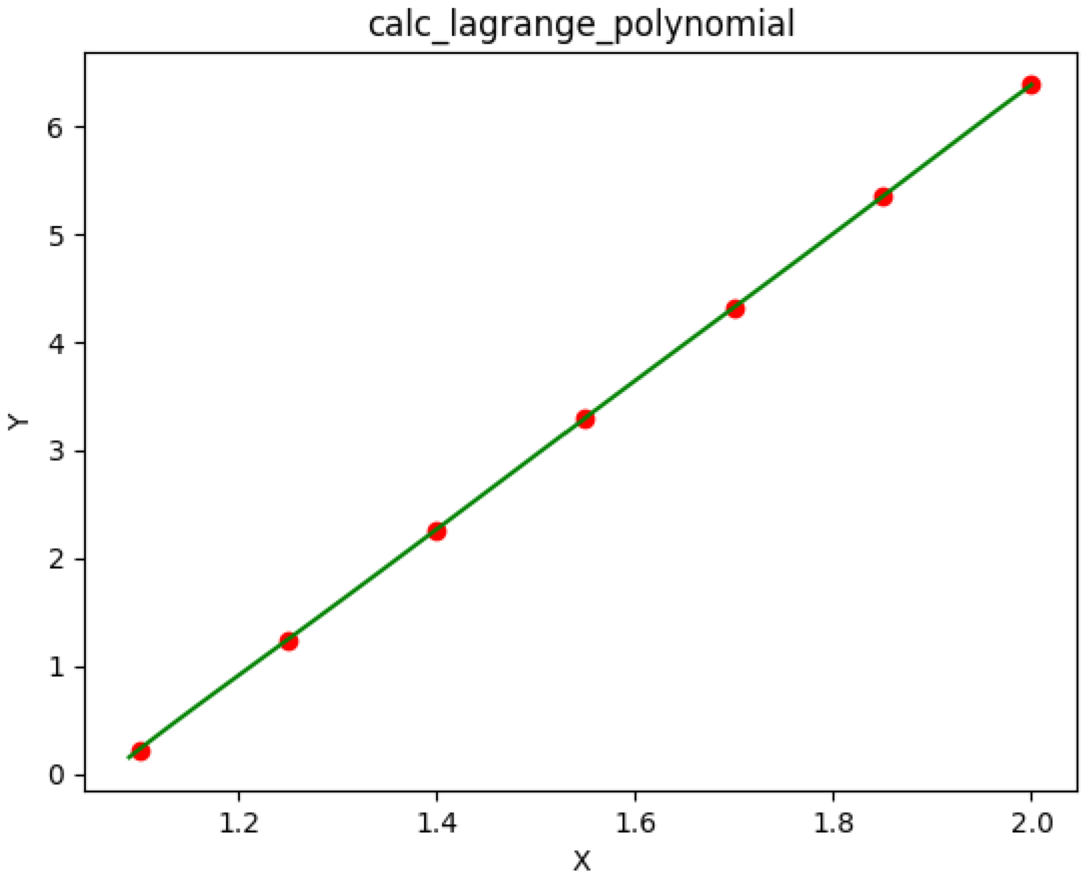
```
    for k in range(1, n + 1):
        last = fin_difs[-1][:]
        fin_difs.append(
            [last[i + 1] - last[i] for i in range(n - k + 1)])

    h = xs[1] - xs[0]
    dts1 = [0, -1, 1, -2, 2, -3, 3, -4, 4]
    f1 = lambda x: ys[alpha_ind] + sum([
        reduce(lambda a, b: a * b,
               [(x - xs[alpha_ind]) / h + dts1[j] for j in range(k)])
        * fin_difs[k][len(fin_difs[k]) // 2] / factorial(k)
        for k in range(1, n + 1)])
    f2 = lambda x: ys[alpha_ind] + sum([
        reduce(lambda a, b: a * b,
               [(x - xs[alpha_ind]) / h - dts1[j] for j in range(k)])
        * fin_difs[k][len(fin_difs[k]) // 2 - (1 - len(fin_difs[k]) % 2)] / factorial(k)
        for k in range(1, n + 1)])
    return lambda x: f1(x) if x > xs[alpha_ind] else f2(x)


def calc_stirling_polynomial(xs, ys):
    n = len(xs) - 1
    alpha_ind = n // 2
    fin_difs = []
    fin_difs.append(ys[:])

    for k in range(1, n + 1):
        last = fin_difs[-1][:]
        fin_difs.append(
            [last[i + 1] - last[i] for i in range(n - k + 1)])

    h = xs[1] - xs[0]
    dts1 = [0, -1, 1, -2, 2, -3, 3, -4, 4]
    f1 = lambda x: ys[alpha_ind] + sum([
        reduce(lambda a, b: a * b,
               [(x - xs[alpha_ind]) / h + dts1[j] for j in range(k)])
        * fin_difs[k][len(fin_difs[k]) // 2] / factorial(k)
        for k in range(1, n + 1)])
    f2 = lambda x: ys[alpha_ind] + sum([
        reduce(lambda a, b: a * b,
               [(x - xs[alpha_ind]) / h - dts1[j] for j in range(k)])
        * fin_difs[k][len(fin_difs[k]) // 2 - (1 - len(fin_difs[k]) % 2)] / factorial(k)
        for k in range(1, n + 1)])
    return lambda x: (f1(x) + f2(x)) / 2


def calc_bessel_polynomial(xs, ys):
    n = len(xs) - 1
    alpha_ind = n // 2
    fin_difs = []
    fin_difs.append(ys[:])

    for k in range(1, n + 1):
        last = fin_difs[-1][:]
        fin_difs.append(
            [last[i + 1] - last[i] for i in range(n - k + 1)])

    h = xs[1] - xs[0]
    dts1 = [0, -1, 1, -2, 2, -3, 3, -4, 4, -5, 5]
    f = lambda x: (ys[alpha_ind] + ys[alpha_ind]) / 2 + sum([
        reduce(lambda a, b: a * b,
               [(x - xs[alpha_ind]) / h + dts1[j] for j in range(k)])
        * fin_difs[k][len(fin_difs[k]) // 2] / factorial(2 * k) +
        ((x - xs[alpha_ind]) / h - 1 / 2) *
```

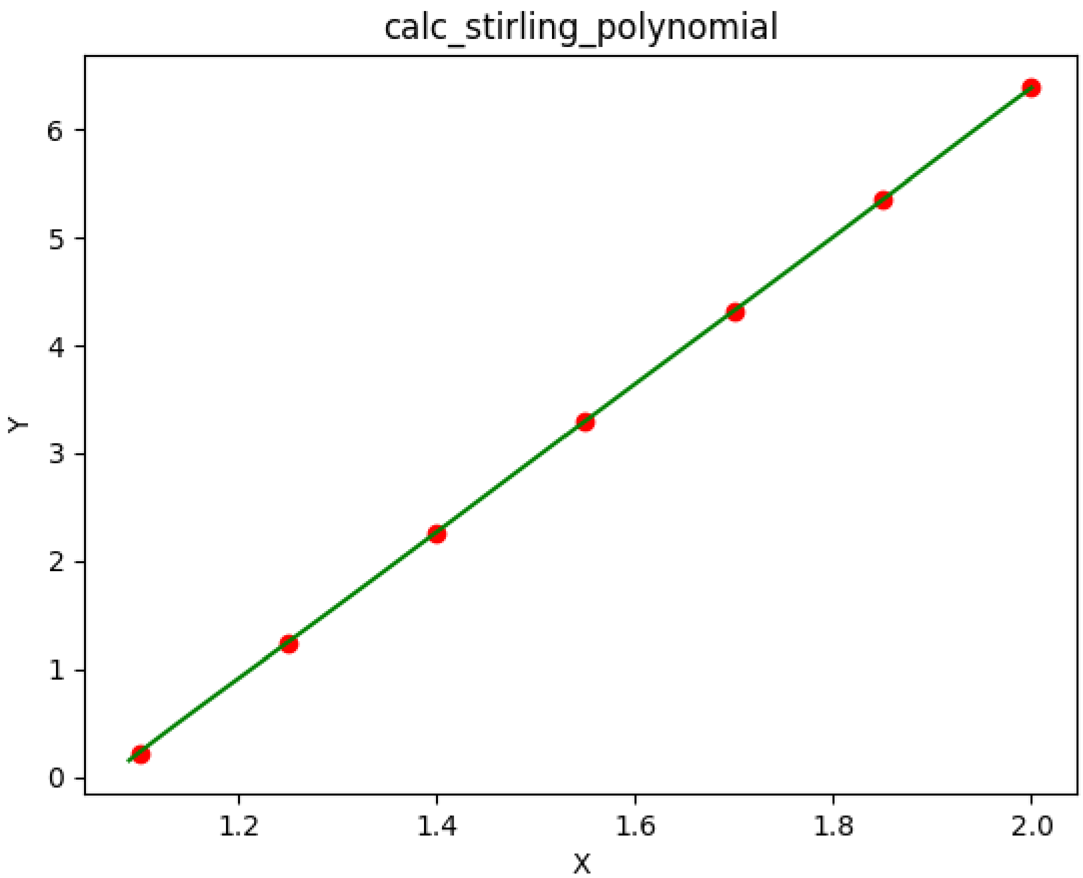
```
        reduce(lambda a, b: a * b,
               [(x - xs[alpha_ind]) / h + dts1[j] for j in range(k)])
        * fin_difs[k][len(fin_difs[k]) // 2] / factorial(2 * k + 1)
        for k in range(1, n + 1)])
    return f


def draw_plot(a, b, func, dx=0.01):
    xs, ys = [], []
    a -= dx
    b += dx
    x = a
    while x <= b:
        xs.append(x)
        ys.append(func(x))
        x += dx
    plt.plot(xs, ys, 'g')


def main(xs, ys, x):
    methods = [calc_lagrange_polynomial,
               calc_newton_divided_difference_polynomial,
               calc_newton_finite_difference_polynomial,
               calc_gauss_polynomial,
               calc_stirling_polynomial,
               calc_bessel_polynomial]
    for method in methods:
        # для гаусса и стирлинга нечётное число узлов должно быть
        if (method is calc_gauss_polynomial or method is calc_stirling_polynomial) \
                and len(xs) % 2 == 0:
            continue
        # для бесселя чётное число узлов должно быть
        if method is calc_bessel_polynomial and len(xs) % 2 == 1:
            continue
        print(method.__name__)

        P = method(xs, ys)

        plt.title(method.__name__)

        draw_plot(xs[0], xs[-1], P)
        for i in range(len(xs)):
            plt.scatter(xs[i], ys[i], c='r')
        plt.xlabel("X")
        plt.ylabel("Y")
        plt.show()

        print(f'P({x}) = {P(x)}')
        print('-' * 60)


if __name__ == '__main__':
    mode = 0
    if mode == 0:
        xs = [1.1, 1.25, 1.4, 1.55, 1.7, 1.85, 2]
        ys = [0.2234, 1.2438, 2.2644, 3.2984, 4.3222, 5.3516, 6.3867]
        x = 1.121
        # x = 1.482
    elif mode == 1:
        xs = list(map(float, input('input xs: ').split()))
        ys = list(map(float, input('input ys: ').split()))
        x = float(input('input x: '))
    elif mode == 2:
        with open('test2.txt') as f:
            xs = list(map(float, f.readline().strip().split()))
```

```python
        ys = list(map(float, f.readline().strip().split()))
        x = float(f.readline().strip())
elif mode == 3:
    print('functions: ')
    print('1. x ^ 2 - 3 * x')
    print('2. x ^ 5')
    func_number = int(input('input 1 or 2: '))
    f = lambda x: x ** 2 - 3 * x if func_number == 1 else x ** 5
    n = int(input('input n: '))
    x0 = float(input('input first x: '))
    xn = float(input('input last x: '))
    h = (xn - x0) / (n - 1)
    xs = [x0 + h * i for i in range(n)]
    ys = list(map(f, xs))
    x = float(input('input x: '))
else:
    # xs = [0.15, 0.2, 0.33, 0.47]
    xs = [0.15, 0.2, 0.25, 0.3]
    ys = [1.25, 2.38, 3.79, 5.44]
    x = 0.22
main(xs, ys, x)
```

Тестовые данные

```python
xs = [1.1, 1.25, 1.4, 1.55, 1.7, 1.85, 2]
ys = [0.2234, 1.2438, 2.2644, 3.2984, 4.3222, 5.3516, 6.3867]
```




calc_stirling_polynomial
Y
X
0
1
2
3
4
5
6
1.2
1.4
1.6
1.8
2.0

```
calc_lagrange_polynomial
P(1.121) = 0.37147968132677955
------------------------------------------------------------
calc_newton_divided_difference_polynomial
divided differences:
0.2234  1.2438  2.2644  3.2984  4.3222  5.3516  6.3867
6.80267 6.804   6.89333 6.82533 6.86267 6.90067
0.00444 0.29778 -0.22667    0.12444 0.12667
0.65185 -1.16543    0.78025 0.00494
-3.02881    3.2428  -1.29218
8.36214 -6.04664
-16.00975
------------------------------
P(1.121) = 0.3714796813267792
------------------------------------------------------------
calc_newton_finite_difference_polynomial
finite differences:
0.2234  1.2438  2.2644  3.2984  4.3222  5.3516  6.3867
1.0204  1.0206  1.034   1.0238  1.0294  1.0351
0.0002  0.0134  -0.0102 0.0056  0.0057
0.0132  -0.0236 0.0158  0.0001
-0.0368 0.0394  -0.0157
0.0762  -0.0551
-0.1313
------------------------------
P(1.121) = 0.37147968132678
------------------------------------------------------------
calc_gauss_polynomial
P(1.121) = 0.37147968132677844
------------------------------------------------------------
calc_stirling_polynomial
P(1.121) = 0.37147968132677844
------------------------------------------------------------
```

Вывод

В ходе лабораторной работы я познакомился с интерполяцией функции разными методами (линейная, квадратичная, многочлен Лагранжа, многочлен Ньютона, многочлены Гаусса, Стирлинга и Бесселя).

Линейная и квадратичная интерполяция – простые методы, но неточные.

Многочлен Лагранжа – хороший метод, но много вычислений. Малая погрешность при небольших n, с изменением числа узлов все вычисления заново.

Многочлен Ньютона с разделёнными разностями – хороший метод. Используется для неравноотстоящих узлов. При добавлении новых узлов первые члены многочлена остаются неизменными.

Многочлен Ньютона с конечными разностями – хороший метод. Используется для равноотстоящих узлов. При добавлении новых узлов первые члены многочлена остаются неизменными. Есть формулы для интерполирования вперёд и назад. Можно использовать для экстраполирования (но будут бОльшие погрешности).