Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

Дисциплина: «Вычислительная математика»

Лабораторная работа №4
Вариант: Метод интерполяции кубическими сплайнами

Выполнил: Кизилов Степан Александрович,
группа P32312
Преподаватель: Перл Ольга Вячеславовна

Санкт-Петербург, 2023 год

1 Описание метода

Идея в том, чтобы получить кусочно-заданную функцию f(x) на $[a, b]$, разбитой на участки согласно влодным значениям значениям: $[x_i, x_{i+1}]$.
Кроме того на функцию накладываются определённые условия:

- $f(x_i) = y_i$: функция принимает заданные значения на заданных точках
- функция f(x) должна быть дважды непрерывно-дифференцируемой функцией, заданной на $[a, b]$
- кусочное задание должно быть многочленом степени не выше 3 (т.к. кубический сплайн)

Получаем, что на каждом из участков функция задаётся:

$$S_i(x) = a_i + b_i \cdot (x - x_i) + c_i \cdot (x - x_i)^2 + d_i \cdot (x - x_i)^3$$

Далее:

$$S_i(x_i) = a_i$$
$$S_i'(x_i) = b_i$$
$$S_i''(x_i) = 2 \cdot c_i$$
$$S_i'''(x_i) = 6 \cdot d_i$$

Для непрерывности:

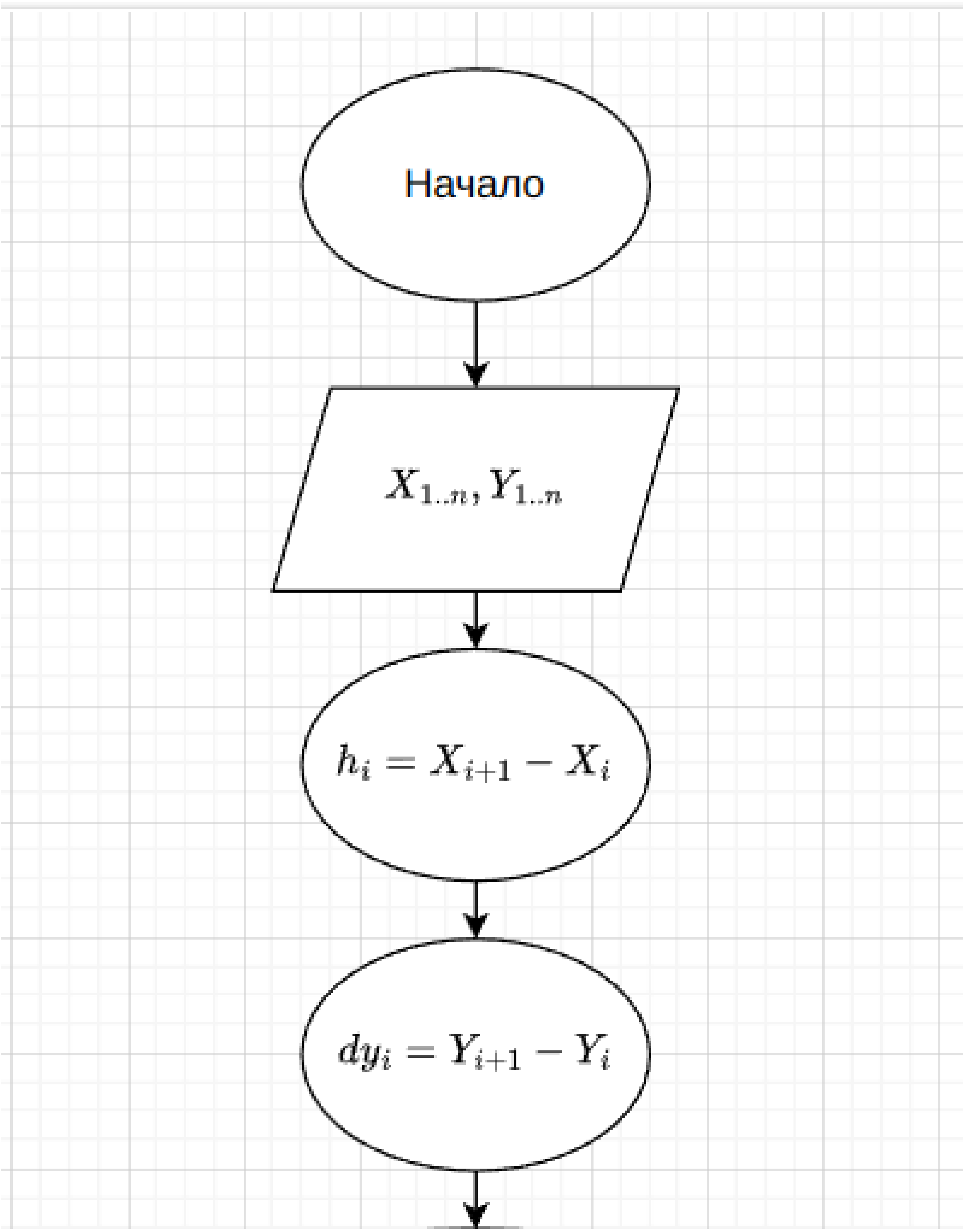
$$S_i(x_i) = S_{i+1}(x_i)$$
$$S_i'(x_i) = S_{i+1}'(x_i)$$
$$S_i''(x_i) = S_{i+1}''(x_i)$$

Кроме того есть дополнительные граничные условия:

$$S''(a) = S''(b) = 0$$

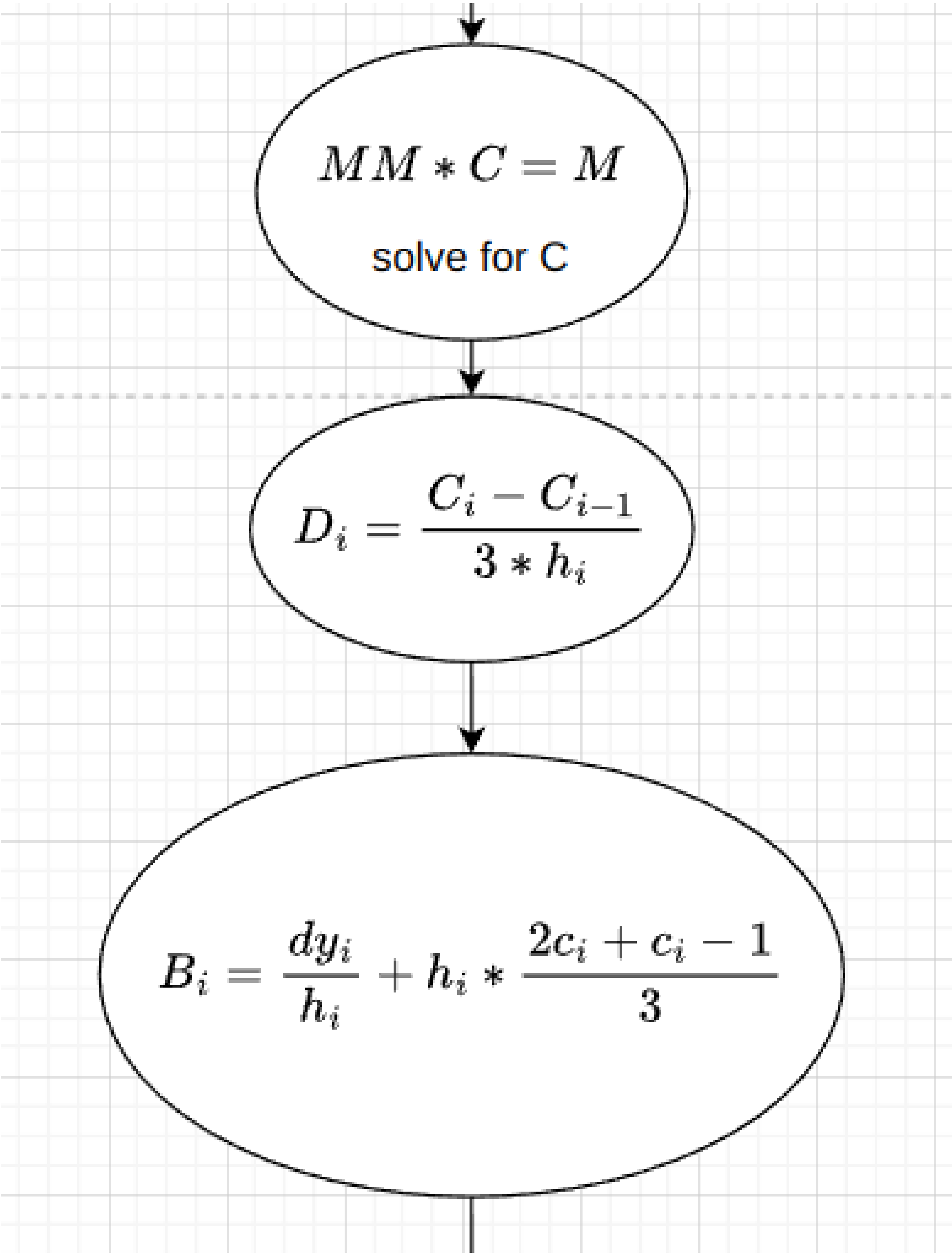
Используя данные условия, можно получить формулы для вычисления коэффициентов a, b, c, d (представлены в блок схеме).

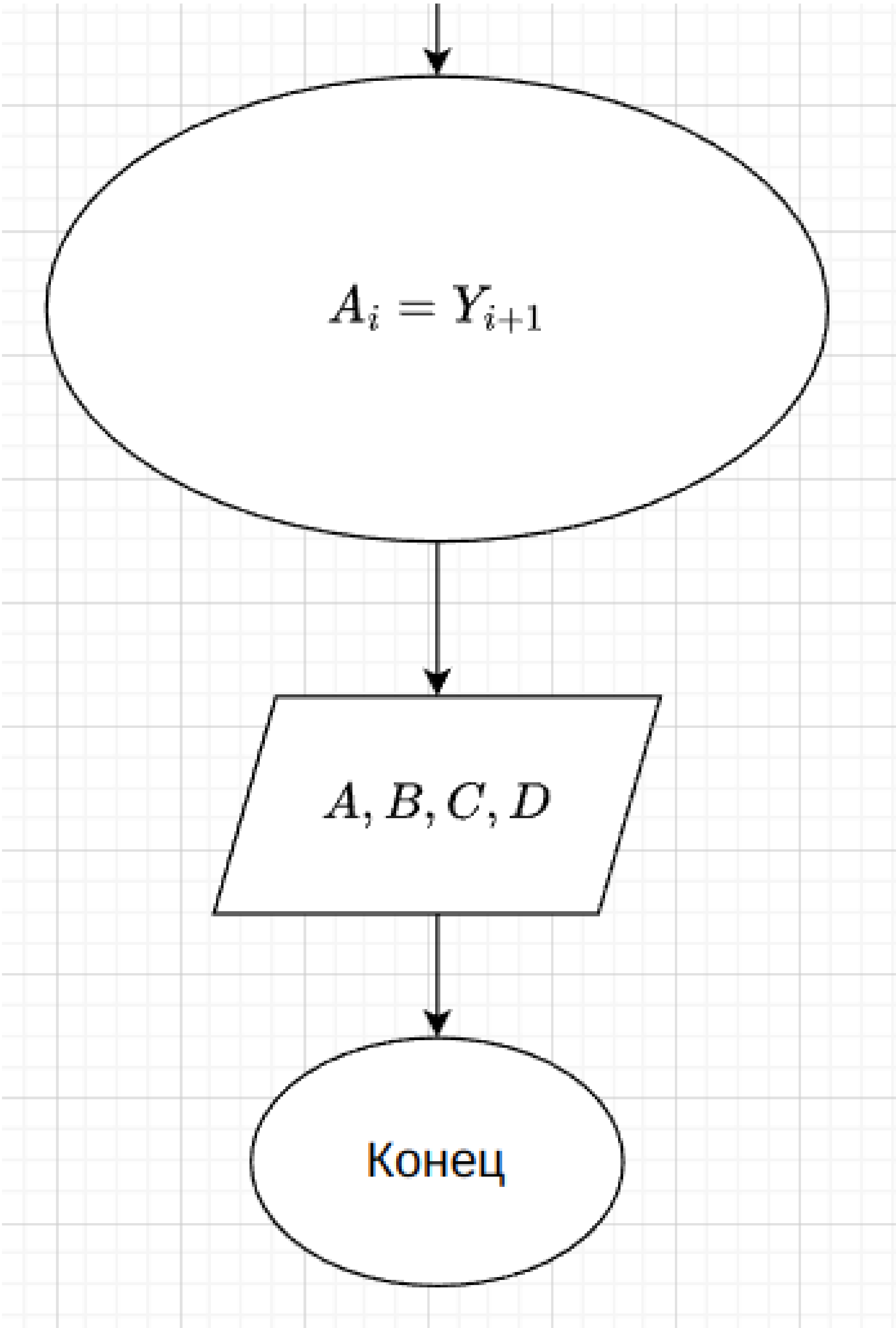
2 Блок-схема метода



$$MM = \begin{pmatrix} 2(h_0 + h_1) & h_1 & 0 & \dots & 0 & 0 \\ h_1 & 2(h_1 + h_2) & h_1 & 0 & \dots & 0 \\ & \dots & & & & \\ 0 & 0 & \dots & 0 & h_{n-1} & 2(h_{n-1} + h_n) \end{pmatrix}$$

$$M = \begin{pmatrix} 3 * (\frac{dy_1}{h_1} - \frac{dy_0}{h_0}) \\ 3 * (\frac{dy_2}{h_2} - \frac{dy_1}{h_1}) \\ \dots \end{pmatrix}$$


$MM * C = M$
solve for C
$D_i = \frac{C_i - C_{i-1}}{3 * h_i}$
$B_i = \frac{dy_i}{h_i} + h_i * \frac{2c_i + c_i - 1}{3}$



$A_i = Y_{i+1}$
A, B, C, D
Конец

3 Исходный код

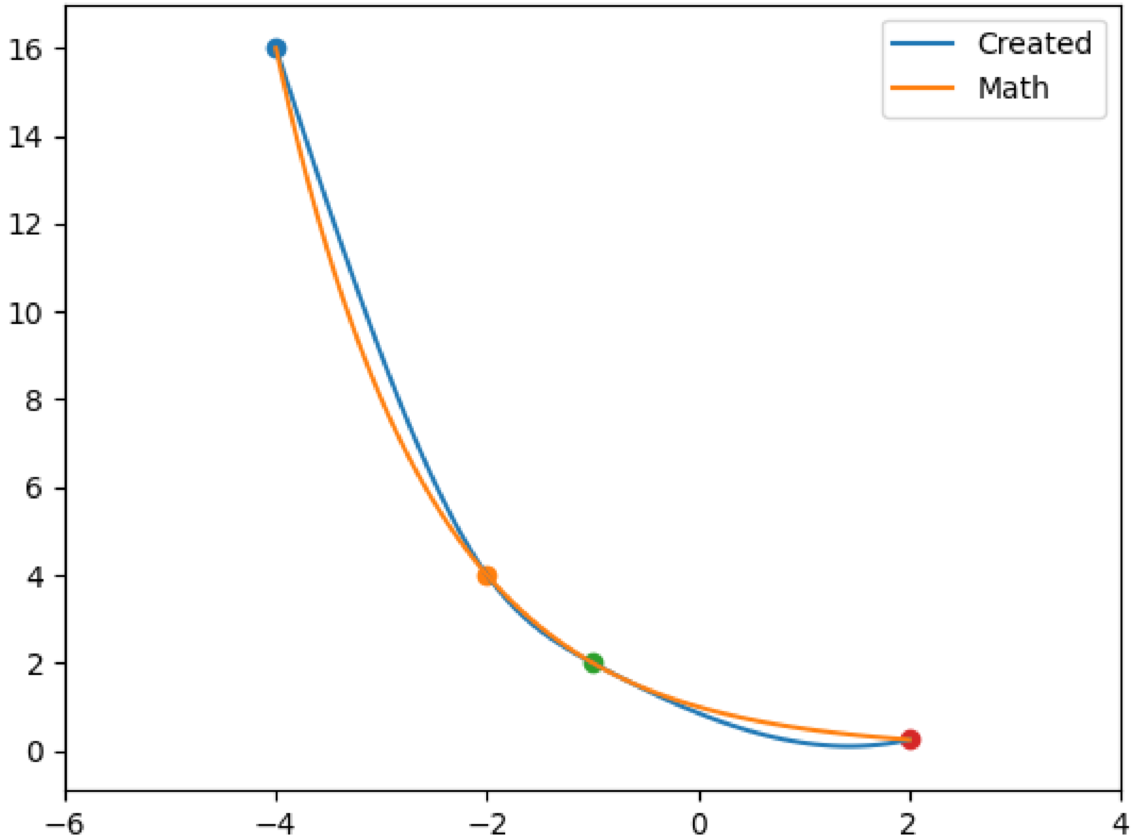
```python
def interpolate_by_spline(x_values: list[float], y_values: list[float], x_point: float) -> float | None:
    h_values = [x_values[i] - x_values[i - 1] for i in range(1, len(x_values))]  # h_i == y_{i + 1}
    dy_values = [y_values[i] - y_values[i - 1] for i in range(1, len(y_values))]

    c_matrix = []

    first_row = [2 * (h_values[0] + h_values[1]), h_values[1]]
    first_row += [0] * (len(h_values) - 2)
    first_row.append(3 * ((dy_values[1]) / h_values[1] - (dy_values[0]) / h_values[0]))

    c_matrix.append(first_row)

    for i in range(1, len(h_values) - 1):
        row = [float(0)] * (i - 1)
        row.append(h_values[i])
        row.append(2 * (h_values[i] + h_values[i + 1]))
        row.append(h_values[i + 1])
        row += [0] * (len(h_values) - 3 - (i - 1))
        row.append(3 * ((dy_values[i + 1]) / h_values[i + 1] - (dy_values[i]) / h_values[i]))

        c_matrix.append(row)
```

```python
conv_n = len(h_values) - 1

last_row = [float(0)] * (conv_n - 1)
last_row.append(h_values[conv_n - 1])
last_row.append(2 * (h_values[conv_n - 1] + h_values[conv_n]))
last_row.append(3 * ((dy_values[conv_n]) / h_values[conv_n] - (dy_values[conv_n - 1]) / h_values[conv_n - 1]))

c_matrix.append(last_row)
c_list = solve_matrix(c_matrix)[0]

d_list = [c_list[0] / (3 * h_values[0])]
for i in range(conv_n):
    d_list.append((c_list[i + 1] - c_list[i]) / (3 * h_values[i + 1]))

b_list = [(dy_values[0] / h_values[0]) + h_values[0] * (2 * c_list[0] / 3)]
for i in range(conv_n):
    b_list.append((dy_values[i + 1] / h_values[i + 1]) + h_values[i + 1] * ((2 * c_list[i + 1] + c_list[i]) / 3))

a_list = y_values[1:]
```

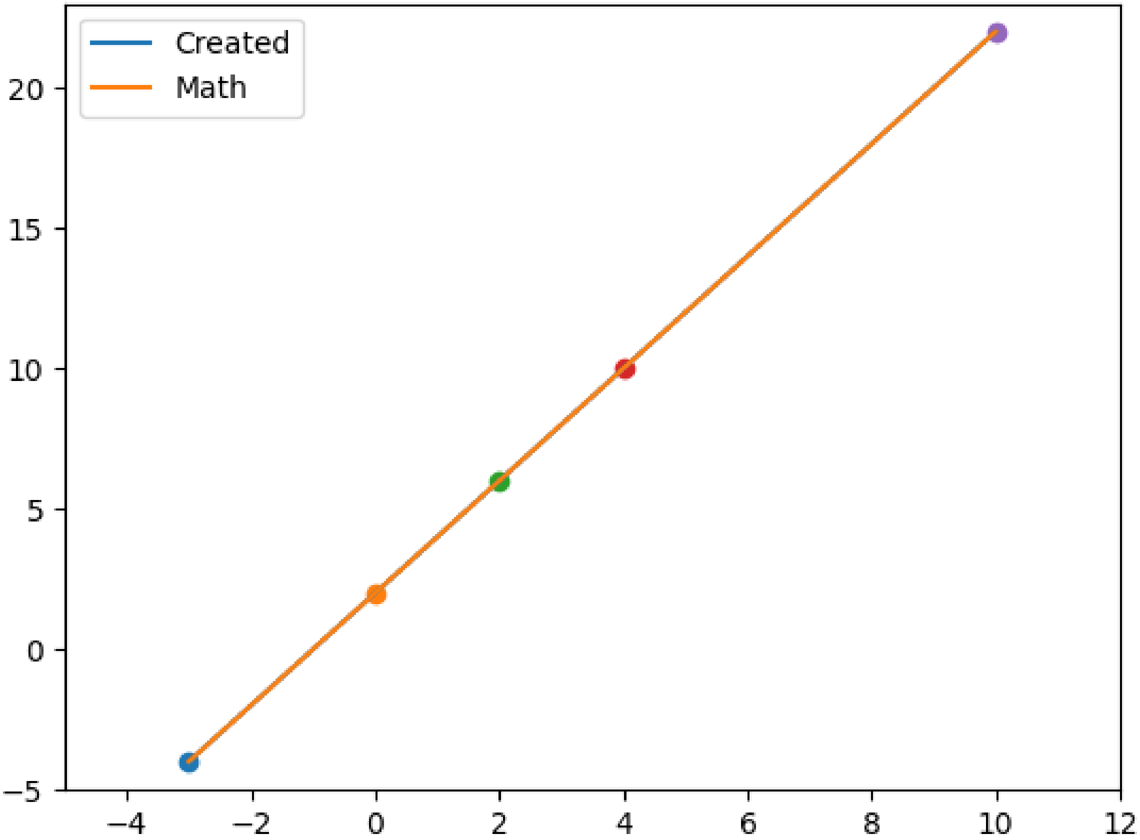
```python
for i in range(1, len(x_values)):

    if x_point == x_values[i]:
        return y_values[i]

    if x_point < x_values[i]:
        need_number = i - 1
        right = x_values[i]
        return a_list[need_number] + \
               b_list[need_number] * (x_point - right) + \
               c_list[need_number] * (x_point - right) * (x_point - right) + \
               d_list[need_number] * (x_point - right) * (x_point - right) * (x_point - right)
```

4 Примеры и результаты работы

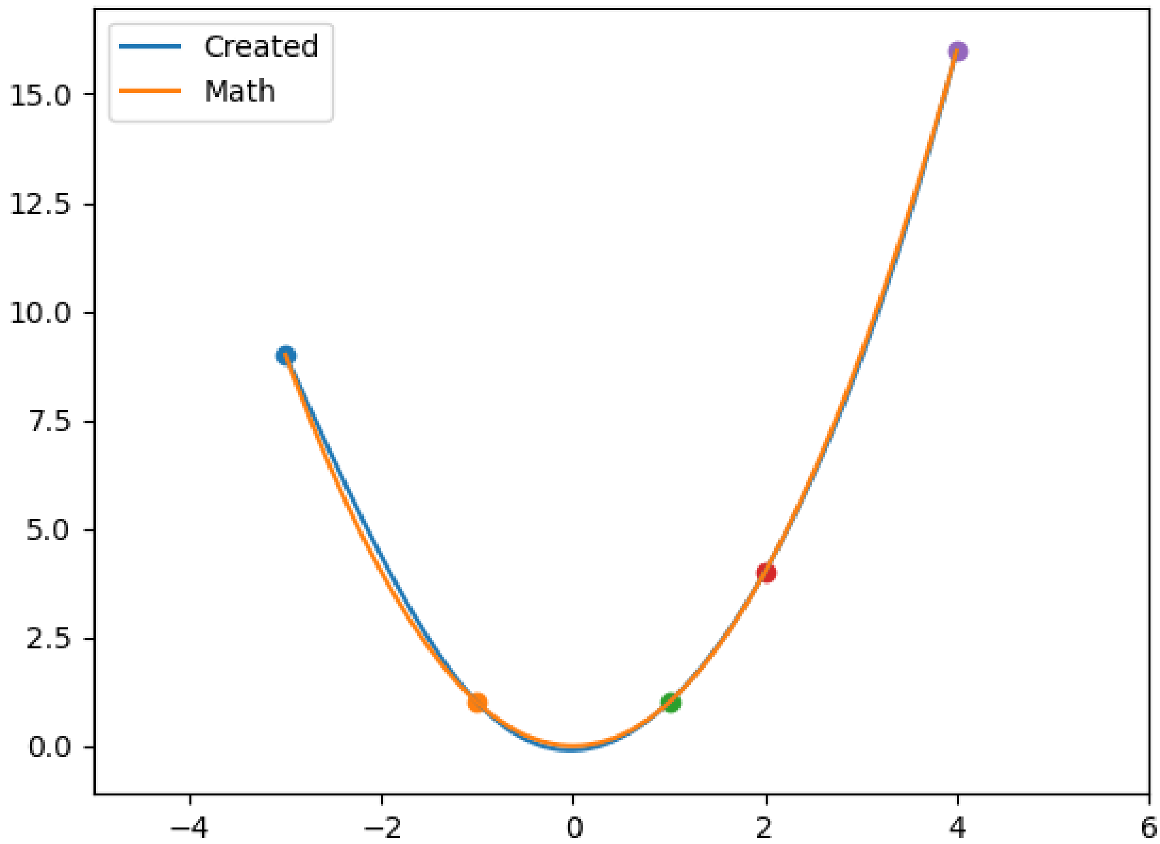
4.1 Пример 1



4.2 Пример 2



4.3 Пример 3



5 Вывод

Судя по примеру 2, кубический сплайн довольно хорошо смог интерполировать линейную функцию, в то время как с параболой и экспонентой есть расхождения (особенно видно в 1 примере)
По сравнению с интерполяцией по Лагранжу, сплайны эффективнее работают, когда имеется больше точек.
Кроме того, возможно интерполяцию по Ньютону более расширяема, по сравнению со сплайнами, потому что в 1 случае построение идёт постепенно (по итерациям), и легко добавить новую точку, а во втором случае может нарушиться граничное условие, поэтому строить нужно заново