Университет ИТМО
МФ КТиУ, Ф ПИиКТ

Лабораторная работа №3.2
Дисциплина «Вычислительная математика»

Приближение функций

Вариант:
Интерполирование методом Ньютона

Выполнил:
Студент группы P3212
Куприянов Артур Алексеевич

Преподаватель:
Перл Ольга Вячеславовна

г. Санкт-Петербург
2020 г.

Теория

Интерполирование методом Ньютона сводится к нахождению раздельных разностей и их умножением на разницу между произвольной точки и заданного значения X.

$$Nn(x) = f(x_0) + f(x_0,\ x_1)\cdot(x - x_0) + f(x_0,\ x_1, x_2)\cdot(x - x_0)\cdot(x - x_1) + \cdots + f(x_0,\ x_1, \ldots, x_n)\cdot(x - x_0)\cdot(x - x_1)\ldots(x - x_{n-1}) \quad (13)$$

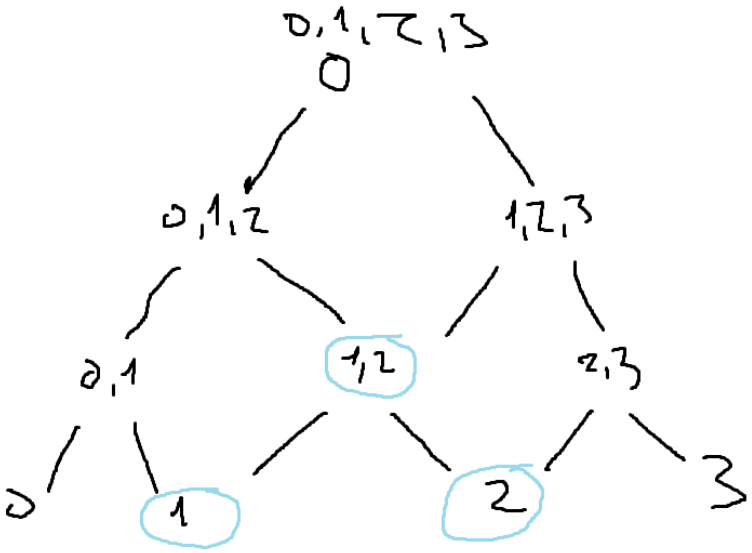
При этом

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

Реализация

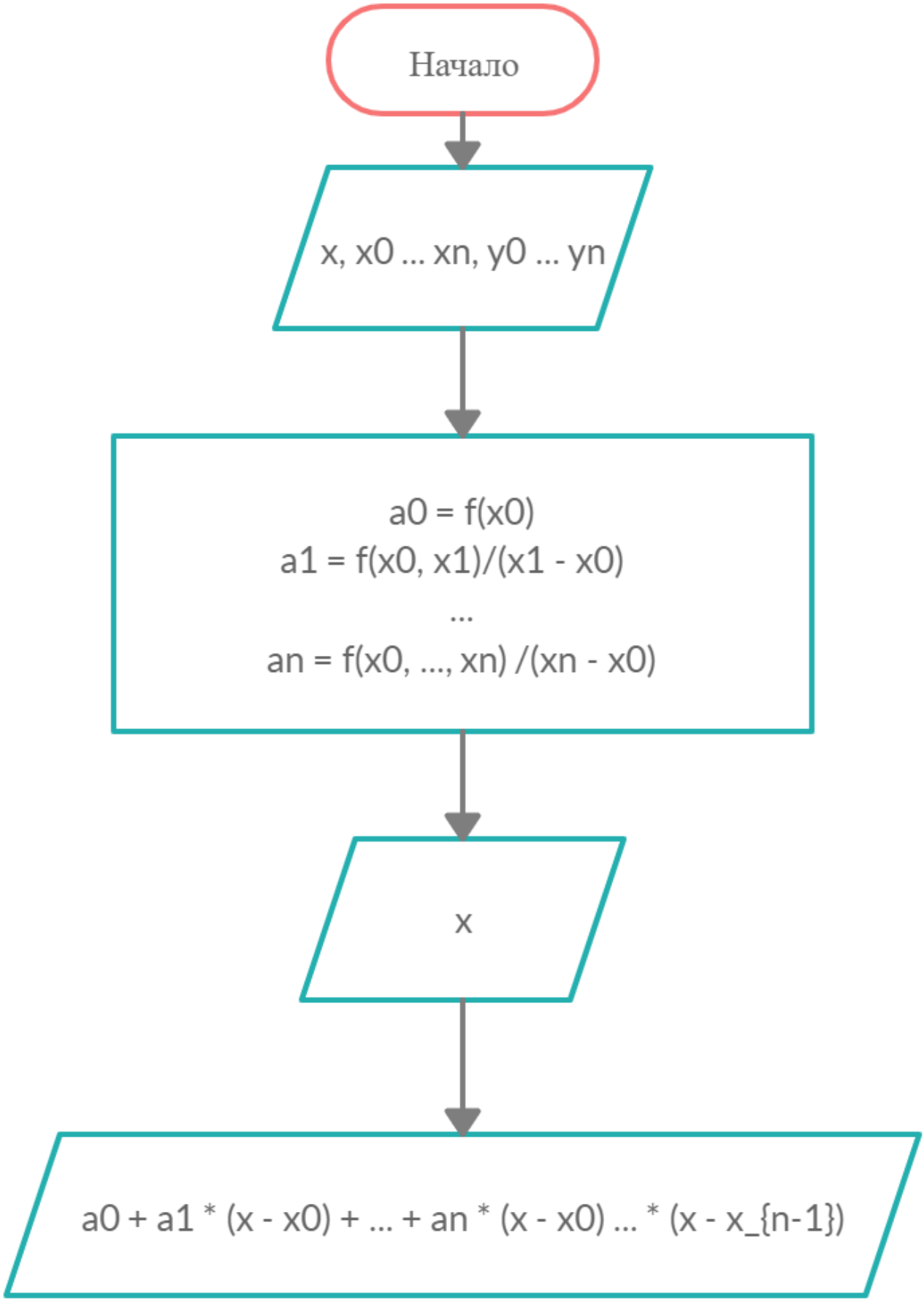
Для программной реализации можно сделать рекурсивный вызов с мемоизацией, так как при вычислении нескольких значений должны использовать уже вычисленные раздельные разности.

Это можно представить в виде графа:



Чтобы сэкономить вычислительные ресурсы мы можем использовать двумерный массив с размером количества исходных точек, которые будут расположены на побочной диагонали матрицы

Блок-схема


Начало
x, x0 ... xn, y0 ... yn
a0 = f(x0)
a1 = f(x0, x1)/(x1 - x0)
...
an = f(x0, ..., xn) /(xn - x0)
x
a0 + a1 * (x - x0) + ... + an * (x - x0) ... * (x - x_{n-1})

```java
@Override
public ExtendedFunction interpolate(final List<Dot> dots) {
    List<Dot> points = List.copyOf(dots);

    double[][] dividedDifferences = calculateDividedDifferences( points);

    return new ExtendedFunction(x -> {
        double sum = 0;
        double coef = 1;
        for (int i = dividedDifferences.length - 1; i >= 0; i--) {
            sum += coef * dividedDifferences[i][0];
            coef *= x - points.get(i).getX();
        }

        return sum;
    });
}

private double[][] calculateDividedDifferences(List<Dot> points) {
    double[][] cache = createEmptyMatrix(points.size());
    calcDividedDifference(0, 0, cache, points);
    return cache;
}

private double calcDividedDifference(int i, int j, double[][] matrix, List<Dot> points){
    if (!Double.isNaN(matrix[i][j])) {
        return matrix[i][j];
    }

    if (j == matrix.length - 1 - i) {
        matrix[i][j] = points.get(i).getY();
        return points.get(i).getY();
    }


    matrix[i][j] = ((calcDividedDifference(i, j + 1, matrix, points) - calcDividedDifference(i + 1, j, matrix, points))
                / (points.get(i).getX() - points.get(matrix.length - j - 1).getX()) );
    return matrix[i][j];
}


private double[][] createEmptyMatrix(int size){
    double[][] matrix = new double[size][size];

    for (int i = 0; i < matrix.length; i++) {
        for (int j = 0; j < matrix.length; j++) {
            matrix[i][j] = Double.NaN;
        }
    }

    return matrix;
}
```

Вывод

Метод интерполяции Ньютона позволяет сделать такую функцию, которая проходит через исходные точки, благодаря особенности функции, в которой координаты исходных точек превращаются в константы.

Определенно, плюсом этого метода является то, что можно создать функцию, которая будет зависеть только входных данных, а сама функция будет построена только из исходных точек. Например, в методе Лагранжа необходимо перерасчитывать коэффициенты. Тем не менее, метод Ньютона применяется в тех случаях, когда узлы равноудалены друг от друга

Также метод Ньютона не так хорошо себя проявляет себя в тригонометрических и периодических функциях, как метод кубических сплайнов, т.к. другой метод является кусочно-полиноминальной.

Такой интерполяцией можно предсказать какие-либо значение в области заданных исходных функций.