НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет Программной инженерии и компьютерных технологий

Направление: Нейротехнологии и программная инженерия

Дисциплина: Вычислительная математика

Лабораторная работа № 1

“Интерполяционный полином Ньютона”

Выполнил студент

Рязанов Демид Витальевич

Группа Р3221

Преподаватель: Перл Ольга Вячеславовна

г. Санкт-Петербург

2024

Содержание

Описание метода

Интерполяционный многочлен Ньютона – формула, которая используется для полиномиального интерполирования.

Полином в форме Ньютона может быть представлен в более компактном виде (по схеме Горнера), которая получается путем последовательного вынесения за скобки множителей:

$$P_n(x)=f[x_0]+(x-x_0)(f[x_1,x_0]+(x-x_1)(f[x_2,x_1,x_0]+(x-x_2)(f[x_3,x_2,x_1,x_0]+...)))$$

Алгоритм:

1) Сначала ищутся **f** по формуле

$$f[x_i]=y_i,\ для\ \ i=0..n$$

$$f[x_k,x_{k-1},...,x_1,x_0]=\frac{f[x_k,x_{k-1},...,x_2,x_1]-f[x_{k-1},x_{k-2},...,x_1,x_0]}{x_k-x_0}$$

для этого используем матрицу **matrix**

Пример для n = 5

i \ j	0	1	2	3	4
0	$f[x_0]=y_0$	$f[x_1,x_0]$	$f[x_2,x_1,x_0]$	$f[x_3,x_2,x_1,x_0]$	$f[x_4,x_3,x_2,x_1]$
1	$f[x_1]=y_1$	$f[x_2,x_1]$	$f[x_3,x_2,x_1]$	$f[x_4,x_3,x_2,x_1]$	0
2	$f[x_2]=y_2$	$f[x_3,x_2]$	$f[x_4,x_3,x_2]$	0	0
3	$f[x_3]=y_3$	$f[x_4,x_3]$	0	0	0
4	$f[x_4]=y_4$	0	0	0	0

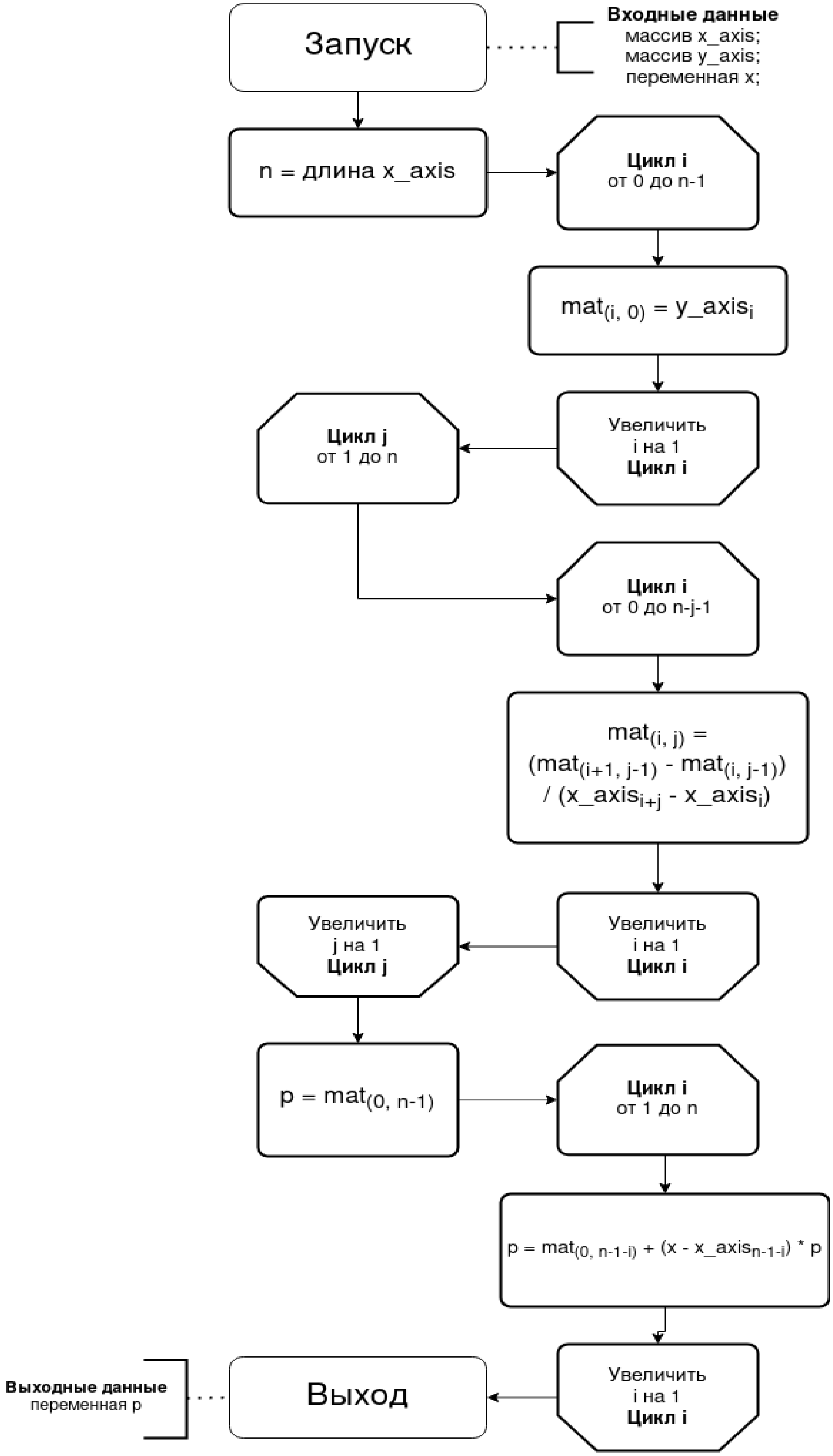
Найдем **f**, по формуле

$$mas[i][j]=\frac{mas[i+1][j-1]-mas[i][j-1]}{x_{i+j}-x_i}\ ,\ для\ \ j=1..n\ ,\ i=0..(n-j)$$

2) Потом с помощью **f** вычисляется полином при указанном значении **x** по схеме Горнера накапливая ответ в переменной **polynom**

Изначально **polynom** = **matrix[0][n-1]**

Блок-схема


Запуск
Входные данные
массив x_axis;
массив y_axis;
переменная x;
n = длина x_axis
Цикл i
от 0 до n-1
mat(i, 0) = y_axis_i
Увеличить
i на 1
Цикл i
Цикл j
от 1 до n
Цикл i
от 0 до n-j-1
mat(i, j) =
(mat(i+1, j-1) - mat(i, j-1))
/ (x_axis_i+j - x_axis_i)
Увеличить
i на 1
Цикл i
Увеличить
j на 1
Цикл j
p = mat(0, n-1)
Цикл i
от 1 до n
p = mat(0, n-1-i) + (x - x_axis_n-1-i) * p
Увеличить
i на 1
Цикл i
Выход
Выходные данные
переменная p

Примечание: Для удобства на блок-схеме переименованы переменные: polynom → p, matrix → mat

Исходный код

```
def interpolate_by_newton(x_axis, y_axis, x
    n = len(x_axis)
    matrix = [[0.0] * n for i in range(n)]
    for i in range(n):
        matrix[i][0] = y_axis[i]
    for j in range(1, n):
        for i in range(n-j):
            matrix[i][j] = (matrix[i+1][j-1] - matrix[i][j-1]) / (x_axis[i+j] - x_axis[i])
    n = n - 1
    polynom = matrix[0][n]
    for i in range(1, n + 1):
        polynom = matrix[0][n - i] + (x - x_axis[n - i]) * polynom
    return polynom
```

Примеры работы

	Пример 1	**Пример 2**	**Пример 3**	**Пример 4**	**Пример 5**
x_axis	1 2 3	1 5 4 2	1 0.3 0.4 0.2 14 3.3 23	3 4 7	123 41 134
y_axis	4 5 6	3 4 6 2	2 4 12 0 43 2 2	3 4 7	352 23 134
x	5	7	4	1	5
Вывод	8	-25.5	2894.1922625892967	1	-1209.9491452685788

Вывод

Точность работы алгоритма зависит от количества входных данных: чем больше узлов, тем точнее полином приближен к реальной функции.

Отличие от полинома Лагранжа заключается в способе задания. Полином Лагранжа использует базисные полиномы, которые вычисляются для каждой точки набора, а затем объединяются в один полином с помощью коэффициентов Лагранжа. Полином Ньютона использует конечные разности, которые вычисляются для каждой точки набора и затем комбинируются в полином. Также при добавлении нового узла полином Ньютона не надо считать заново, добавляется лишь одно новое слагаемое.

Сложность алгоритма $O(n^2)$