НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет ПИиКТ

Отчёт по лабораторной работе на тему:

Интерполяция и аппроксимация

Метод Ньютона

Выполнил
студент

Агнистова Алина Юрьевна

Группа № P3225

Преподаватель: Перл Ольга Вячеславовна

г. Санкт-Петербург

2024

Задание

Написать программу для построения интерполяционного полинома Ньютона. Входные данные: набор точек (x, y) и координата x, для которой необходимо найти значение интерполяционного полинома.

Описание метода и расчётные формулы

Интерполяционный полином Ньютона – полином степени n (n – количество узлов интерполяции) в виде:

$$P_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_n(x - x_0) \dots (x - x_{n-1})$$

Построение многочлена сводится к определению коэффициентов a_i – разделённые разности. Разделённая разность – обобщение понятия производной для дискретного набора точек. Определяется по формуле:

$$a_i = \frac{\Delta^i y_0}{i!\, h^i}, \text{где } i = 1 \dots n, \Delta^i y_0 - \text{конечная разность порядка } i, h - \text{шаг интерполяции}$$

Шаг интерполяции высчитывается по формуле: $h = x_{i+1} - x_i$

Конечная разность первого порядка рассчитывается по формуле:

$$\Delta y_{n-1} = y_n - y_{n-1}$$

Конечная разность высшего порядка рассчитывается по формуле:

$$\Delta^k y_{n-2} = \Delta^{k-1} y_{n-1} - \Delta^{k-1} y_{n-2}$$

В результате полином примет вид:

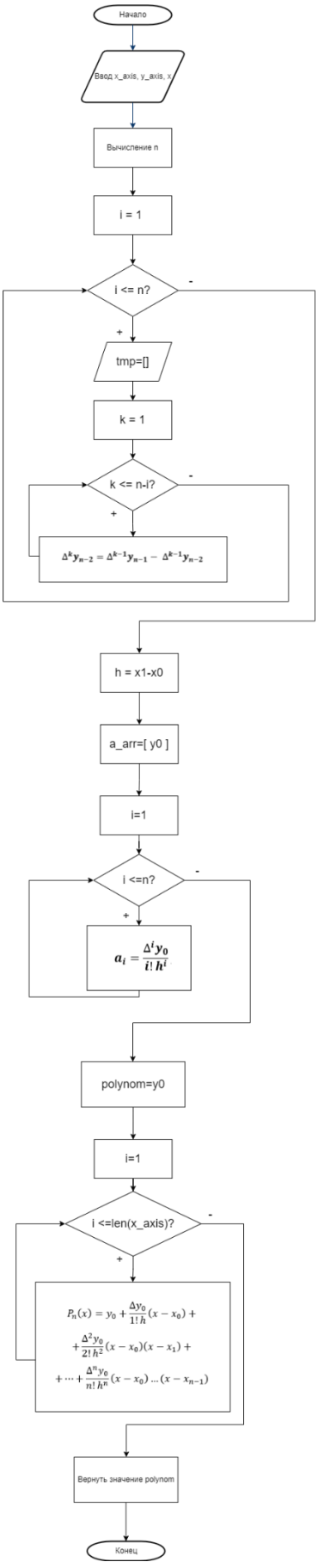
$$P_n(x) = y_0 + \frac{\Delta y_0}{1!\, h}(x - x_0) +$$

$$+ \frac{\Delta^2 y_0}{2!\, h^2}(x - x_0)(x - x_1) +$$

$$+ \cdots + \frac{\Delta^n y_0}{n!\, h^n}(x - x_0) \dots (x - x_{n-1})$$

Интерполяционный метод Ньютона позволяет найти промежуточные значения величины по имеющемуся дискретному набору известных значений. Во многих задачах является оптимальным выбором между точностью и вычислительной эффективностью по сравнению с другими методами (например, многочлен Лагранжа эквивалентен по точности, но вычислительная эффективность ниже).

Блок-схема реализованного метода



Код метода

```
1 usage
def interpolate_by_newton(x_axis, y_axis, x):
    n = len(y_axis)
    #Вычисление разностей
    diff = [y_axis.copy()]
    for i in range(1, n):
        tmp = []
        for k in range(n - i):
            dif = diff[i - 1][k + 1] - diff[i - 1][k]
            tmp.append(dif)
        diff.append(tmp)
    #Вычисление коэффициентов
    h = x_axis[1] - x_axis[0]
    a_arr = [diff[0][0]]
    for i in range(1, n):
        a_arr.append(diff[i][0] / math.factorial(i) * h**i)
    #Вычисление полинома
    polynom = a_arr[0]
    for i in range(1, len(x_axis)):
        tmp = a_arr[i]
        for k in range(i):
            tmp *= (x - x_axis[k])
        polynom += tmp
    return polynom
```

```
if __name__ == '__main__':
    x_axis = list(map(float, input().rstrip().split()))
    y_axis = list(map(float, input().rstrip().split()))
    if (len(x_axis) != len(y_axis)):
        print("Количество входных данных X и Y должно быть одинаково")
        exit()
    x = int(input().strip())
    result = interpolate_by_newton(x_axis, y_axis, x)
    print (str(result) + '\n')
```

Примеры и результаты работы программы

Пример 1

Выбранная функция: $f(x) = x^2$

Предполагаемый результат: 6.25

Входные данные:

x-axis	y-axis	x
1	1	2.5
2	4	
3	9	
4	16	

Выходные данные: 6.25

```
1 2 3 4
1 4 9 16
2.5
6.25


Process finished with exit code 0
```

Пример 2

Выбранная функция: $f(x) = x^2$

Предполагаемый результат: 20.25

Входные данные:

x-axis	y-axis	x
1	1	4.5
2	4	
3	9	
4	16	

Выходные данные: 20.25

```
1 2 3 4
1 4 9 16
4.5
20.25


Process finished with exit code 0
```

Пример 3

Выбранная функция: $f(x) = x^2$

Предполагаемый результат: 6.25 (без учёта выброса в данных)

Входные данные:

x-axis	y-axis	x
1	1	2.5
2	4	
3	9	
4	100	

Выходные данные: 1

```
1 2 3 4
1 4 9 100
2.5
1.0


Process finished with exit code 0
```

Пример 4

Выбранная функция: $f(x) = x^3$

Предполагаемый результат: 64

Входные данные:

x-axis	y-axis	x
1	3	4
3	27	
5	125	
7	343	

Выходные данные: 407

```
1 3 5 7
3 27 125 343
4
407.0


Process finished with exit code 0
```

Пример 5

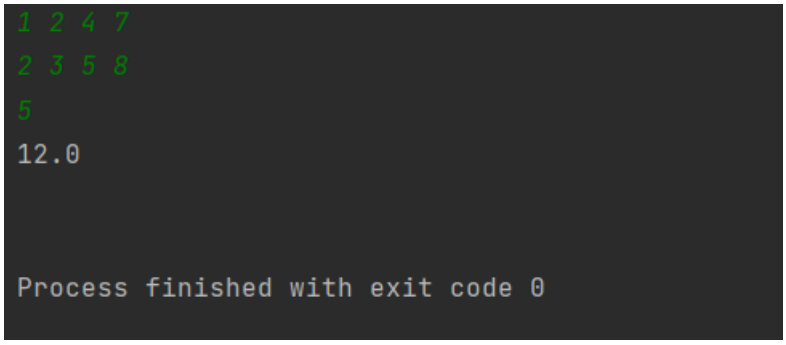
Выбранная функция: $f(x) = x + 1$

Предполагаемый результат: 6

Входные данные:

x-axis	y-axis	x
1	2	5
2	3	
4	5	
7	8	

Выходные данные: 12



Проанализировав результаты запуска реализованного метода, можно сделать вывод, что код некорректно работает при случае неравномерного распределения (примеры 4 и 5), что вызвано отличием математического вычисления коэффициентов для равномерного и неравномерного распределения, что я не учла. Также код чувствителен к выбросам в данных (пример 3), но это связано с чувствительностью самого метода.

Для оценки метода приведена таблица сравнения метода Ньютона с методом Лагранжа и методом кубических сплайнов:

	Метод Ньютона	Метод Лагранжа	Кубические сплайны
Применимость	Интерполяция в случае последовательного добавления точек	Интерполяция с небольшим числом точек	Интерполяция, требующая гладкой кривой
Вычислительная сложность	Средняя (легко добавлять новые точки)	Высокая (неэффективно при большом количестве точек)	Высокая (требуется решение системы)
Точность, гладкость	Зависит от распределения точек; возможны осцилляции на концах интервала	Высокая точность внутри интервала, возможны осцилляции на концах интервала	Высокая гладкость и точность
Сложность добавления новых точек	Можно добавлять новые точки без полного пересчёта	Полный пересчёт для каждой новой точки	Пересчёт коэффициентов при добавлении точки

Таким образом, как говорилось выше, метод Ньютона является хорошим выбором в случаях, когда нужно последовательно добавлять точки, при этом вычислительная эффективность допускает работу с большим количеством точек (не требуется полный пересчёт, как в случае с методом Лагранжа), но точность зависит от распределения точек (точнее всего – метод кубических сплайнов). Метод Ньютона – золотая середина, если выбирать по вычислительной сложности и точности среди анализируемых методов.

Чаще всего метод Ньютона применяется в случае последовательного добавления точек, так как добавление новых точек не составляет особой сложности. Метод сохраняет свою эффективность при увеличении объема входных данных, так как не производится повторных вычислений, однако может стать численно нестабильным при большом числе точек. Стоит отметить, что требуется различность точек по x, так как в случае совпадения двух или более точек возникает деление на ноль.

Временная алгоритмическая сложность складывается из двух основных действий: вычисление разделённых разностей и построение полинома. При вычислении разделённых разностей нужно итеративно применять формулу к каждому набору точек, что приводит к количеству операций $\frac{n(n-1)}{2}$, значит сложность составляет $O(n^2)$. Построение полинома требует вычисление произведений для каждого слагаемого, причём

для n −го слагаемого необходимо n умножений, тогда количество операций $\frac{n(n-1)}{2} \rightarrow O(n^2)$. Получаем, что общая алгоритмическая сложность является квадратичной.

Что касается численных ошибок численного метода, то необходимо упомянуть, что для полиномов высокой степени могут возникать осцилляции и метод неустойчив к выбросам данных.

Вывод

В результате выполнения лабораторной работы была реализована интерполяция методом Ньютона на языке программирования Python. Программа разделяется на три основных аспекта: вычисление разделённых разностей, вычисление коэффициентов, построение полинома и вычисление значения в точке. Алгоритмическая сложность метода составила $O(n^2)$.

Метод Ньютона – метод интерполяции, позволяющий построить приближенную функцию по дискретному набору точек. Он характеризуется средними показателями вычислительной эффективности и точности (относительно других методов интерполяции), но чувствителен к выбросам в данных. Метод применим в случаях, когда точность не является ключевым требованием, но часто требуется добавление новых точек. В случаях, когда важна гладкость и точность лучше выбрать метод кубических сплайнов, а при небольшом количестве точек – метод Лагранжа рациональнее, так как сложность математических вычислений ниже.