Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

Лабораторная работа №4

По вычислительной математике

Вариант 3

Выполнил:
Студент группы P3216
Векшин Арсений Иванович
Преподаватель:
Малышева Татьяна Алексеевна

г. Санкт-Петербург

2024 г.

Оглавление

Цель работы

Найти функцию, являющуюся наилучшим приближением заданной табличной функции по методу наименьших квадратов.

Исходные данные:

Пользователь вводит таблично заданную функцию.

Программная реализация задачи:

1. Исходные данные вводятся в файл стандартного ввода
2. Сформировать и вывести таблицу значений функции, значений аппроксимации и ее отклонений, коэффициент корреляции и меру отклонения (только линейная аппроксимация).

Вычислительная реализация задачи

1. Сформировать таблицу табулирования заданной функции на указанном интервале (см. табл. 1)
2. Построить линейное и квадратичное приближения по 11 точкам заданного интервала;
3. Найти среднеквадратичные отклонения для каждой аппроксимирующей функции. Ответы дать с тремя знаками после запятой;
4. Выбрать наилучшее приближение;
5. Построить графики заданной функции, а также полученные линейное и квадратичное приближения;
6. **Подробные вычисления привести в отчете.**

Вычислительная реализация

Функция: $y = \frac{4x}{x^4+3}$ на исследуемом интервале $x \in [-2, 0]$ $h = 0.2$

I	0	1	2	3	4	5	6	7	8	9	10
X	-2.0	-1.8	-1.6	-1.4	-1.2	-1.0	-0.8	-0.6	-0.4	-0.2	0
Y	-0.421	-0.533	-0.67	-0.819	-0.946	-1	-0.939	-0.767	-0.529	-0.267	0

Линейная аппроксимация: N = 11

Вычисляем суммы:

SX = -11

SY = -6.891

SXX = 15.4

SXY = 7.632

Получаем СЛАУ:

$$\begin{cases} 15.4a - 11b = 7.632 \\ -11a - 11b = -6.891 \end{cases}$$

Решение:
a = 0.55
b = 0.076

Квадратичная аппроксимация:
N = 11
Вычисляем суммы:
SX = -11.0
SXX = 15.4
SXXX = -24.2
SXXXX = 40.5328
SY = -6.891
SXY = 7.632
SXXY = -9.97
Получаем СЛАУ:

$$\begin{cases} -11a - 11b + 15.4c = -6.891 \\ -11a + 15.4b - 24.2c = 7.632 \\ 15.4a - 24.2b + 40.5328c = -9.97 \end{cases}$$

Решение: a = -0.0041; b = 1.7572; c = 0.8048
Таблица аппроксимации:

I	0	1	2	3	4	5	6	7	8	9	10
X	-2.0	-1.8	-1.6	-1.4	-1.2	-1.0	-0.8	-0.6	-0.4	-0.2	0
Y	-0.421	-0.533	-0.67	-0.819	-0.946	-1	-0.939	-0.767	-0.529	-0.267	0
P1	−1.024	−0.914	−0.804	−0.694	−0.584	−0.474	-0.364	-0.254	-0.144	-0.034	0.076
P2	−0.299	−0.56	−0.755	−0.887	−0.954	−0.957	−0.895	−0.769	−0.578	−0.323	−0.004
E1	-0.603	-0.381	-0.134	0.125	0.362	0.483	0.531	0.515	0.385	0.233	0.076
E2	0.122	-0.027	-0.085	-0.058	-0.008	0.043	0.044	-0.002	-0.049	-0.056	-0.004

https://www.desmos.com/calculator/al7ttz4ldg?lang=ru

Исходный код программы

Линейный

```
def aproximate(points):
    SX, SXX, SY, SXY = 0, 0, 0, 0
    for p in points:
        SX += p[0]
        SXX += p[0] ** 2
        SY += p[1]
        SXY += p[0] * p[1]

    a = (SXY * len(points) - SX*SY) / (SXX * len(points) - SX*SX)
    b = (SXX * SY - SX * SXY) / (SXX * len(points) - SX*SX)
    return Function([a, b], FunctionType.linear)
```

Полином

```
def aproximate(points, degree):
    if(degree<2 or degree>8):
        print("АХТУНГ: степень апроксимации некоректна")
        sys.exit(-1)

    A = np.zeros((degree+1, degree+1))
    B = []

    A[0][0] = len(points)
    for i in range(1, degree*2+1):
        _val = 0
        for p in points:
            _val += p[0]**i
        fill_rev_diag(list(A), _val, i)
        # fill_rev_diag(A, _val, i)

    for i in range(degree+1):
        _val = 0
        for p in points: _val += (p[0] ** i) * p[1]
        B.append(_val)

    _koofs = solve_slau(A, B)

    return Function(_koofs, FunctionType.polynomial)
```

Логарифмический

```
def aproximate(points):
    _points = copy.deepcopy(points)

    for i in range(len(_points)):
        _points[i][0] = m.log(points[i][0], m.e)

    koofs = linear.aproximate(_points).get_koofs()
    # koofs[0] = m.exp(koofs[0])
    return Function(koofs, FunctionType.logariphm)
```

Степенной

```
def aproximate(points):
    _points = copy.deepcopy(points)
    for i in range(len(_points)):
        _points[i][0] = m.log(_points[i][0], m.e)
        _points[i][1] = m.log(_points[i][1], m.e)

    koofs = linear.aproximate(_points).get_koofs()
    koofs.reverse()
    koofs[0] = m.exp(koofs[0])

    return Function(koofs, FunctionType.power)
```

Экспоненциальный

```
def aproximate(points):
    _points = copy.deepcopy(points)
    for i in range(len(_points)):
        _points[i][1] = m.log(_points[i][1], m.e)

    koofs = linear.aproximate(_points).get_koofs()
    koofs.reverse()
    koofs[0] = m.exp(koofs[0])

    return Function(koofs, FunctionType.exponent)
```

Полный код программы:
https://github.com/ArsenyVekshin/ITMO/tree/master/CompMath/lab4

Вывод

В результате работы программы были написаны и наглядно визуализированы различные виды аппроксимации.