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

Факультет программной инженерии и компьютерной техники

Вычислительная математика

Лабораторная работа №5

Вариант 10

Студент: Крикунов Олег Евгеньевич

P3267

Преподаватель: Машина Екатерина Алексеевна

Оценка: _____

Подпись преподавателя: _____

1. Цели работы

Изучить численные методы интерполяции и реализовать их в программе.

2. Описание метода, расчётные формулы

Многочлен Лагранжа

$$L_n(x) = \sum_{i=0}^{n} y_i \, l_i(x)$$

$$Ln(x) = \sum_{i=0}^{n} y_i \frac{(x-x_0)(x-x_1)\dots(x-x_{i-1})(x-x_{i+1})\dots(x-x_n)}{(x_i-x_0)(x_i-x_1)\dots(x_i-x_{i-1})(x_i-x_{i+1})\dots(x_i-x_n)}$$

$$L_n(x) = \sum_{i=0}^{n} y_i \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} \qquad (8)$$

Полином Лагранжа имеет малую погрешность при небольших значениях n ($n < 20$).

К недостаткам можно отнести то, что с изменением числа узлов приходится все ***вычисления проводить заново***.

Многочлен Ньютона

Интерполяционный многочлен Ньютона с разделенными разностями

При построении интерполяционного полинома в форме Ньютона используется понятие **разделенной разности**.

Разделенные разности применяются для функций, заданных на неравномерной сетке (**неравноотстоящие узлы**).

Разделенные разности (или разностные отношения) **нулевого порядка** совпадают со значениями функции в узлах: $f(x_i) = y_i$.

Определение. Разделенные разности первого порядка называют величины (определяются через разделенные разности нулевого порядка):

$$f(x_0, x_1) = \frac{f(x_1)-f(x_0)}{x_1-x_0}, \quad f(x_1, x_2) = \frac{f(x_2)-f(x_1)}{x_2-x_1} \quad f(x_i, x_{i+1}) = \frac{f(x_{i+1})-f(x_i)}{x_{i+1}-x_i}$$

Разделенные разности второго порядка называют величины (определяются через разделенные разности первого порядка):

$$f(x_0, x_1, x_2) = \frac{f(x_1,x_2)-f(x_0,x_1)}{x_2-x_0}, \quad f(x_1, x_2, x_3) = \frac{f(x_2,x_3)-f(x_1,x_2)}{x_3-x_1}$$

$$f(x_i, x_{i+1}, x_{i+2}) = \frac{f(x_{i+1},x_{i+2})-f(x_i,x_{i+1})}{x_{i+2}-x_i}$$

Интерполяционные формулы Ньютона для равноотстоящих узлов

Введем обозначение: $t = (x - x_0)/h$. Тогда получим формулу Ньютона, которая называется **первой интерполяционной формулой Ньютона для интерполирования вперед:**

$$N_n(x) = y_0 + t\Delta y_0 + \frac{t(t-1)}{2!}\Delta^2 y_0 + \cdots + \frac{t(t-1)\ldots(t-n+1)}{n!}\Delta^n y_0$$

Полученное выражение может аппроксимировать функцию на всем отрезке изменения аргумента $[x_0, x_n]$. Однако более целесообразно (с точки зрения повышения точности расчетов) использовать эту формулу для $x_0 \le x \le x_1$. При этом за x_0 может приниматься любой узел интерполяции x_k. Например, для $x_1 \le x \le x_2$, вместо x_0 надо взять значение x_1. Тогда интерполяционный многочлен Ньютона:

$$N_n(x) = y_i + t\Delta y_i + \frac{t(t-1)}{2!}\Delta^2 y_i + \cdots + \frac{t(t-1)\ldots(t-n+1)}{n!}\Delta^n y_i \quad (*)$$

Интерполяционную формулу (*) обычно используют для вычислений значений функции в точках левой половины отрезка.

Интерполяционные формулы Ньютона для равноотстоящих узлов

Для правой половины отрезка разности вычисляют справа налево: $t = (x - x_n)/h$. Тогда получим формулу Ньютона, которая называется **второй интерполяционной формулой Ньютона для интерполирования назад:**

$$N_n(x) = y_n + t\Delta y_{n-1} + \frac{t(t+1)}{2!}\Delta^2 y_{n-2} + \cdots + \frac{t(t+1)\ldots(t+n-1)}{n!}\Delta^n y_0$$

3. Вычислительная часть

X	y	№ варианта	X_1	X_2
2,10	3,7587	5	2,112	2,205
2,15	4,1861	10	2,355	2,254
2,20	4,9218	15	2,114	2,216
2,25	5,3487	20	2,359	2,259
2,30	5,9275	25	2,128	2,232
2,35	6,4193	30	2,352	2,284
2,40	7,0839	35	2,147	2,247

Таблица конечных разностей:

y	Δ^{2y}	Δ^{3y}	Δ^{4y}	Δ^{5y}
0	1	6	6	0
1	7	12	6	
8	19	18		
27	37			
64				

$X_1 = 2{,}335$

Формула Ньютона

$$f(x) = y_0 + (x - x_0)\Delta y_0 + \frac{(x - x_0)(x - x_1)}{2!}\Delta^2 y_0 + \frac{(x - x_0)(x - x_1)(x - x_2)}{3!}\Delta^3 y_0 + ..$$

$$x_0 = 2.34, \quad y_0 = 6.4796, \quad h = 0.02, \quad \Delta y_0 = 0.0662, \quad \Delta^2 y_0 = 0.0018, \quad \Delta^3 y_0 = 0.0004$$

$$f(2.352) = 6.4796 + (2.352 - 2.34) \cdot 0.0662 + \frac{(2.352 - 2.34)(2.352 - 2.32)}{2!} \cdot 0.0018 + \frac{(2.352 - 2.34)(2.352 - 2.32)(2.352 - 2.34)}{3!} \cdot 0.0004$$

$$P(2.352) = 6.4294 + 0.002 \cdot 0.0038$$

P(2.352) = 6.4294 + 0.0000076 = 6.4294076

Формула Гаусса

$$P(x) = f(x_{-1}) + s\Delta f(x_{-1}) + \frac{s(s+1)}{2!}\Delta^2 f(x_{-1}) + \frac{s(s+1)(s+2)}{3!}\Delta^3 f(x_{-1}) + \cdots$$

$$s = \frac{x - x_0}{h}, x_{-1} = 2.30 \text{ и } h = 0.01$$

$$s = \frac{2.284 - 2.30}{0.01} = -1.6$$

$$P(2.284) = 6.4104 - 1.6 \cdot 0.0038 + \frac{(-1.6)(-0.6)}{2!} =$$

P(2.284) = 6.4104 - 0.00608 = 6.40432

4. Программная часть

Листинг программы:

Полином Лагранжа:

```
#ifndef COMPUTATIONAL_MATHEMATICS_LAGRANGEPOLYNOMIAL_H
#define COMPUTATIONAL_MATHEMATICS_LAGRANGEPOLYNOMIAL_H

#include <vector>

double LagrangeInterpolation(std::vector<double> &x_axis, std::vector<double> &y_axis, double
x) {
    if (x_axis.size() != y_axis.size() || x_axis.empty() || y_axis.empty()) {
        throw std::invalid_argument("Неверный ввод: размеры массивов не равны или один из
массивов равен нулю");
    }

    if (x < x_axis[0] || x > x_axis[x_axis.size() - 1]) {
        throw std::invalid_argument("Точка x выходит за пределы интервала x_axis");
    }
```

```
    double result = 0;

    for (size_t i = 0; i < x_axis.size(); i++) {
        double p = y_axis[i];
        for (size_t j = 0; j < x_axis.size(); j++) {
            if (i != j) {
                if (x_axis[i] == x_axis[j]) {
                    throw std::invalid_argument("Невозможно вычислить интерполяционное
значение: точки x_axis совпадают");
                }
                p *= ((x - x_axis[j]) / (x_axis[i] - x_axis[j]));
            }
        }
        result += p;
    }
    return result;
}

#endif
```

Полином Ньютона с разделенными разностями:

```
#ifndef COMPUTATIONAL_MATHEMATICS_NEWTONPOLYNOMIALDIVIDEDDIFFERENCE_H
#define COMPUTATIONAL_MATHEMATICS_NEWTONPOLYNOMIALDIVIDEDDIFFERENCE_H

double DividedDifference(double x0, double x1, double y0, double y1) {
    return (y1 - y0) / (x1 - x0);
}

double NewtonInterpolationDD(const std::vector<double> &x_axis, const std::vector<double>
&y_axis, double x) {

    if (x_axis.size() != y_axis.size() || x_axis.empty() || y_axis.empty()) {
        throw std::invalid_argument("Неверный ввод: размеры массивов не равны или один из
массивов равен нулю");
    }

    if (x < x_axis[0] || x > x_axis[x_axis.size() - 1]) {
        throw std::invalid_argument("Точка x выходит за пределы интервала x_axis");
    }

    std::vector<std::vector<double>> F(x_axis.size(), std::vector<double>(x_axis.size()));

    for (int i = 0; i < x_axis.size(); i++) {
        F[i][0] = y_axis[i];
    }

    for (int j = 1; j < x_axis.size(); j++) {
        for (int i = 0; i < x_axis.size() - j; i++) {
            F[i][j] = DividedDifference(x_axis[i], x_axis[i + j], F[i][j - 1], F[i + 1][j -
1]);
        }
    }

    double result = F[0][0];
    double term = 1;
    for (int i = 1; i < x_axis.size(); i++) {
        term *= (x - x_axis[i - 1]);
        result += F[0][i] * term;
    }
    return result;
}
```

```
#endif
```

Полином Ньютона с конечными разностями:

```
#ifndef COMPUTATIONAL_MATHEMATICS_NEWTONPOLYNOMIALCD_H
#define COMPUTATIONAL_MATHEMATICS_NEWTONPOLYNOMIALCD_H

double NewtonInterpolationFD(std::vector<double> &x_axis, std::vector<double> &y_axis, double
x) {
    std::vector<std::vector<double>> dy(x_axis.size() - 1, std::vector<double>(x_axis.size() -
1, 0));
    double t, h, mult, sum;
    int factorial;

    h = (x_axis[1] - x_axis[0]);

    for (int i = 0; i < x_axis.size() - 1; i++) {
        dy[i][0] = y_axis[i + 1] - y_axis[i];
    }

    for (int i = 1; i < x_axis.size() - 1; i++) {
        for (int j = 0; j < x_axis.size() - 1 - i; ++j) {
            dy[j][i] = dy[j + 1][i - 1] - dy[j][i - 1];
        }
    }

    mult = 1;
    sum = y_axis[0];
    t = (x - x_axis[0]) / h;
    factorial = 1;
    for (int i = 0; i < x_axis.size() - 1; i++) {
        mult *= (t - i);
        factorial *= i + 1;
        sum += mult * dy[0][i] / (factorial);
    }

    t = (x - x_axis[x_axis.size() - 1]) / h;
    sum = y_axis[x_axis.size() - 1];
    mult = 1;
    factorial = 1;
    for (int i = 0; i < x_axis.size() - 1; i++) {
        mult *= (t + i);
        factorial *= i + 1;
        sum += mult * dy[x_axis.size() - 1 - i - 1][i] / (factorial);
    }

    return sum;
}

#endif
```

Header-файл с функциями:

```
#ifndef COMPUTATIONAL_MATHEMATICS_SOLVINGMETHODS_H
#define COMPUTATIONAL_MATHEMATICS_SOLVINGMETHODS_H

#include <functional>
#include <fstream>
```

```cpp
#include "LagrangePolynomial.h"
#include "NewtonPolynomialFiniteDifference.h"
#include "NewtonPolynomialDividedDifference.h"
#include "Difference.h"

double Interpolate(std::vector<double> &x_axis, std::vector<double> &y_axis, double x,
                   std::function<double(std::vector<double> &, std::vector<double> &,
                                        double)> interpolationFunction) {
    return interpolationFunction(x_axis, y_axis, x);
}

double CustomFunction(double x, size_t choice) {
    switch (choice) {
        case 1:
            return sin(x);
        case 2:
            return cos(x);
        case 3:
            return x * x;
        default:
            return 0.0;
    }
}


#endif
```

Конечные разности:

```cpp
#ifndef COMPUTATIONAL_MATHEMATICS_DIFFERENCE_H
#define COMPUTATIONAL_MATHEMATICS_DIFFERENCE_H

void Difference(const std::vector<double> &y) {
    std::vector<std::vector<double>> dif(y.size(), std::vector<double>(y.size()));

    for (int i = 0; i < y.size(); i++) {
        dif[i][0] = y[i];
    }

    for (size_t i = 1; i < y.size(); i++) {
        for (size_t j = 0; j < y.size() - i; j++) {
            dif[j][i] = dif[j + 1][i - 1] - dif[j][i - 1];
        }
    }

    std::cout << std::endl << "Таблица конечных разностей:" << std::endl;
    for (int i = 0; i < y.size(); i ++) {
        if (i == 0) {std::cout << "y" << " ";}
        else {std::cout << "Δ^" << i + 1 << "y" << " ";}
    }
    std::cout << std::endl;

    for (size_t i = 0; i < y.size(); i++) {
        for (size_t j = 0; j < y.size() - i; j++) {
            std::cout << dif[i][j] << "\t";
        }
        std::cout << "\n";
    }
    std::cout << "\n";
}

#endif
```

5. Примеры и результаты работы программы

```
Какой полином вы хотите использовать?
1)Полином Лагранжа
2)Полином Ньютона с разделенными разностями
3)Полином Ньютона с конечными разностями
1
Как бы вы хотели получить данные?
1)Ввести вручную
2)Считать с файла
3)Получить из предоставленных
2
Введите имя файла: t1
Введите точку, в которой хотите найти значение функции: 2.5

Таблица конечных разностей:
y Δ^2y Δ^3y Δ^4y Δ^5y
0   1   6   6   0
1   7   12  6
8   19  18
27  37
64

y(x) = 15.625
Process finished with exit code 0
```

```
1)Полином Лагранжа
2)Полином Ньютона с разделенными разностями
3)Полином Ньютона с конечными разностями
2
Как бы вы хотели получить данные?
1)Ввести вручную
2)Считать с файла
3)Получить из предоставленных
1
Сколько значений y функции?
5
Введите все значения x и y через пробел:
0 0
1 1
2 8
3 27
4 64
Введите точку, в которой хотите найти значение функции: 2.5

Таблица конечных разностей:
y Δ^2y Δ^3y Δ^4y Δ^5y
0   1   6   6   0
1   7   12  6
8   19  18
27  37
64

y(x) = 15.625
```

```
Какой полином вы хотите использовать?
1)Полином Лагранжа
2)Полином Ньютона с разделенными разностями
3)Полином Ньютона с конечными разностями
3
Как бы вы хотели получить данные?
1)Ввести вручную
2)Считать с файла
3)Получить из предоставленных
3
Выберите функцию для исследования:
1) sin(x)
2) cos(x)
3) x^2
3
Введите начало и конец интервала: 0 5
Введите количество точек на интервале: 6
Введите точку, в которой хотите найти значение функции: 3.7

Таблица конечных разностей:
y Δ^2y Δ^3y Δ^4y Δ^5y Δ^6y
0   1   2   0   0   0
1   3   2   0   0
4   5   2   0
9   7   2
16  9
25

y(x) = 13.69
Process finished with exit code 0
```

6. Вывод:

В ходе выполнения работы я познакомился с численными методами интерполяции. Реализовал в программе полином Лагранжа, полином Ньютона с разделенными/конечными коэффициентами.