Отчёт по лабораторной работе №2.

Чилеше Абрахам.
Б9122-02.03.01 СЦТ (1)
Вариант-17

Цель работы 1.

- Построить таблицу конечных разностей по значениям табличной функции.
- По соответствующим интерполяционным формулам вычислить значения функции в заданных узлах
- Оценить минимум и макисмум для $f^{n+1}(x)$
- Проверить на выполнение равенство min R_n < $R_n(z)$ < max R_n, где z - заданный угол, а $R_n(z) = L_n(z) - f(z)$
- Сделать вывод по проделанной работе.

Условия:

- функция: $f(x) = 0.5x^2 + \cos(2x)$
- Интервал: [0.6, 1.1]
- x** = 0.62, x*** = 1.07, x**** = 0.83

Ход работы:

- Для этой лабораторной работы я использовал язык программирования c++

1. Библиотеки

```
1   #include <iostream>
2   #include <vector>
3   #include <cmath>
4   #include <iomanip>
5   #include <algorithm>
```

- **<vector>:** Предоставляет реализацию динамического массива, позволяя создавать изменяемые массивы, способные хранить элементы любого типа данных.
- **<cmath>:** Содержит математические функции и константы, такие как тригонометрические функции (*sin, cos, tan*), логарифмические функции (*log, log10*) и константы, такие как pi
- **<iomanip>:** Позволяет управлять форматированием ввода и вывода в C++, включая контроль ширины и точности вывода, установку символов заполнения и управление форматом чисел с плавающей точкой.

<algorithm>: Предоставляет набор функций для выполнения различных операций над последовательностями элементов, включая поиск, сортировку и модификацию. Примеры включают *min(), max()*

2. функция (алгоритмы)

i. **newton_minus_param**: Эта функция вычисляет один из членов многочлена Ньютона с убывающими степенями **(t - i),** где i изменяется от **0** до **n-1**. Затем результат делится на факториал n.

```
double newton_minus_param(double t, int n) {
    double a = 1;
    for (int i = 0; i < n; i++) {
        a *= (t - i);
    }
    a /= tgamma( x: n + 1);
    return a;
}
```

ii. **newton_plus_param:** Похожая на предыдущую функцию, но вычисляет член многочлена Ньютона с возрастающими степенями (t + i).

```
double newton_plus_param(double t, int n) {
    double a = 1;
    for (int i = 0; i < n; i++) {
        a *= (t + i);
    }
    a /= tgamma( x: n + 1);
    return a;
}
```

iii. **gauss1_minus_param:** Эта функция вычисляет один из членов многочлена Гаусса с чередующимися знаками, используя (t - i) для нечетных итераций и (t + i - 1) для четных. Затем результат делится на факториал n.

```
double gauss1_minus_param(double t, int n) {
    double a = 1;
    for (int i = 0; i < n; i++) {
        if (i % 2 == 1 || i == 0) {
            a *= (t - i);
        } else {
            a *= (t + i - 1);
        }
    }
    a /= tgamma( x: n + 1);
    return a;
}
```

iv. **gauss2_plus_param:** Похожая на предыдущую функцию, но использует (t + i) для нечетных итераций и (t - i + 1) для четных.

```
double gauss2_plus_param(double t, int n) {
    double a = 1;
    for (int i = 0; i < n; i++) {
        if (i % 2 == 1 || i == 0) {
            a *= (t + i);
        } else {
            a *= (t - i + 1);
        }
    }
    a /= tgamma( x: n + 1);
    return a;
}
```

v. **insert_gauss1_polynomial:** Строит многочлен Гаусса, суммируя члены, вычисленные с помощью gauss1_minus_param(), и умножая каждый член на соответствующий коэффициент, хранящийся в двумерном векторе mass.

vi. **insert_gauss2_polynomial:** Похожа на предыдущую функцию, но строит многочлен Гаусса с использованием gauss2_plus_param().

```
double insert_gauss1_polynomial(double t, int n, const vector<vector<double>>& mass) {
    double Px = 0;
    int j = 5;
    for (int i = 0; i < n; i++) {
        Px += mass[i][j] * gauss1_minus_param(t, i);
        if (i % 2 != 0) {
            j--;
        }
    }
    return Px;
}

double insert_gauss2_polynomial(double t, int n, const vector<vector<double>>& mass) {
    double Px2 = 0;
    int j = 5;
    for (int i = 0; i < n; i++) {
        Px2 += mass[i][j] * gauss2_plus_param(t, i);
        if (i % 2 == 0) {
            j--;
        }
    }
    return Px2;
}
```

vii. **insert_newton1_polynomial:** Строит многочлен Ньютона, суммируя члены, вычисленные с помощью newton_minus_param(), и умножая каждый член на соответствующий коэффициент, хранящийся в двумерном векторе mass.

viii. **insert_newton2_polynomial**: Похожа на предыдущую функцию, но строит многочлен Ньютона с использованием newton_plus_param().

```
double insert_newton1_polynomial(double t, int n, const vector<vector<double>>& mass) {
    double Px = 0;
    int j = 0;
    for (int i = 0; i < n; i++) {
        Px += mass[i][j] * newton_minus_param(t, i);
    }
    return Px;
}

double insert_newton2_polynomial(double t, int n, const vector<vector<double>>& mass) {
    double Px2 = 0;
    for (int i = 0; i < n; i++) {
        int j = n - i - 1;
        Px2 += mass[i][j] * newton_plus_param(t, i);
    }
    return Px2;
}
```

3. входных данных

- функция: $f(x) = 0.5x^2 + \cos(2x)$
- Интервал: [0.6, 1.1]
- x** = 0.62, x*** = 1.07, x**** = 0.83

```
const double a = 0.6;
const double b = 1.1;
const int n = 11;
const double h = (b - a) / 10;

vector<double> x_list(n);
vector<double> y_list(n);

for (int i = 0; i < n; i++) {
    double xi = a + i * h;
    x_list[i] = xi;
    y_list[i] = 0.5 * pow(xi, y:2) - cos(X:2 * xi);
}


const double x_star2 = 0.62;
const double x_star3 = 1.07;
const double x_star4 = 0.83;
```

4. Нахождение значений

a) Таблица функции y(x)

```
vector<double> x_list(n);
vector<double> y_list(n);

for (int i = 0; i < n; i++) {
    double xi = a + i * h;
    x_list[i] = xi;
    y_list[i] = 0.5 * pow(xi, y:2) - cos(X:2 * xi);
}

cout << setw(n:5) << "No." << setw(n:18) << "x" << setw(n:18) << "y(x)" << endl;
cout<<"-----------------------------------------------"<<endl;
for (int i = 0; i < n; i++) {
    cout << setw(n:5) << i << setw(n:18) << x_list[i] << setw(n:18) << y_list[i] << endl;
}
cout << endl<<endl;
```

Ниже приведена таблица с результатами

```
C:\Users\acnil\CLionProjects\untitled1\cmake
  No.                  x               y(x)
-----------------------------------------------
    0                0.6         -0.182358
    1               0.65        -0.0562488
    2                0.7         0.0750329
    3               0.75          0.210513
    4                0.8            0.3492
    5               0.85          0.490094
    6                0.9          0.632202
    7               0.95           0.77454
    8                  1          0.916147
    9               1.05            1.0561
   10                1.1            1.1935
```

Таблица значений функции y(x)

b) **Расчет разностей и формирование новой таблицы**

к сожалению, поскольку я использовал c++, код получился немного длинным, поэтому я не могу добавить весь фрагмент кода в отчет. полный код вы можете посмотреть в файле c++, который я прикрепил

```
vector<vector<double>> list_diffs = {y_list};

while (list_diffs.back().size() != 1) {
    vector<double> lis;
    for (int i = 0; i < static_cast<int>(list_diffs.back().size()) - 1; i++) {
        lis.push_back(list_diffs.back()[i + 1] - list_diffs.back()[i]);
    }
    list_diffs.push_back(lis);
}

vector<vector<double>> list_to_table = list_diffs;
int max_length = 0;
for (const auto& lst : list_to_table) {
    max_length = max(max_length, static_cast<int>(lst.size()));
}

for (auto& lst : list_to_table) {
    while (lst.size() < static_cast<size_t>(max_length)) {
        lst.push_back(0); // Use 0 or NaN instead of ""
    }
```

Ниже приведена таблица с результатами

No.	Value 1	Value 2	Value 3	Value 4	Value 5	Value 6
0	-0.182358	-0.0562488	0.0750329	0.210513	0.3492	0.490094
1	0.126109	0.131282	0.13548	0.138687	0.140895	0.142108
2	0.00517276	0.00419826	0.00320678	0.00220825	0.00121263	0.000229872
3	-0.000974504	-0.000991473	-0.000998535	-0.00099562	-0.000982757	-0.000960074
4	-1.69684e-05	-7.06194e-06	2.91509e-06	1.2863e-05	2.26824e-05	3.22751e-05
5	9.90647e-06	9.97703e-06	9.9479e-06	9.81938e-06	9.59274e-06	9.27026e-06
6	7.05606e-08	-2.91266e-08	-1.28523e-07	-2.26635e-07	-3.22482e-07	0
7	-9.96872e-08	-9.93962e-08	-9.8112e-08	-9.58475e-08	0	0
8	2.91015e-10	1.28417e-09	2.26444e-09	0	0	0
9	9.93155e-10	9.80272e-10	0	0	0	0
10	-1.28827e-11	0	0	0	0	0

No.	Value 7	Value 8	Value 9	Value 10	Value 11
0	0.632202	0.77454	0.916147	1.0561	1.1935
1	0.142337	0.141607	0.139949	0.137405	0
2	-0.000730202	-0.001658	-0.00254426	0	0
3	-0.000927799	-0.000886254	0	0	0
4	4.15454e-05	0	0	0	0
5	0	0	0	0	0
6	0	0	0	0	0
7	0	0	0	0	0
8	0	0	0	0	0
9	0	0	0	0	0
10	0	0	0	0	0

Вычисление параметров методов и их погрешностей

Нижеприведенный код оценивает производительность методов интерполяции Ньютона и Гаусса, сравнивая их интерполированные значения с точными решениями и анализируя остаточные ошибки в заданном интервале:

```
const double x_star2 = 0.62;
const double x_star3 = 1.07;
const double x_star4 = 0.83;

double t = min(abs(x_list[0] - x_star2), abs(x_list[1] - x_star2)) / h;

cout<<endl;
cout << "Newton 1: " << insert_newton1_polynomial(t, n, list_diffs) << endl;
cout << "R_N1: " << insert_newton1_polynomial(t, n, list_diffs) - (0.5 * pow(x_star2, y: 2) - cos(2 * x_star2)) << endl;

t = -1 * (x_list[n - 1] - x_star3) / h;
cout << "Newton 2: " << insert_newton2_polynomial(t, n, list_diffs) << endl;
cout << "R_N2: " << insert_newton2_polynomial(t, n, list_diffs) - (0.5 * pow(x_star3, y: 2) - cos(2 * x_star3)) << endl;

t = min(abs(x_list[0] - x_star4), abs(x_list[1] - x_star4)) / h;
cout << "Gauss 1: " << insert_gauss1_polynomial(t, n, list_diffs) << endl;
cout << "R_G1: " << insert_gauss1_polynomial(t, n, list_diffs) - (0.5 * pow(x_star4, y: 2) - cos(2 * x_star4)) << endl;

t = -1 * (x_list[n - 1] - x_star4) / h;
cout << "Gauss 2: " << insert_gauss2_polynomial(t, n, list_diffs) << endl;
cout << "R_G2: " << insert_gauss2_polynomial(t, n, list_diffs) - (0.5 * pow(x_star4, y: 2) - cos(2 * x_star4)) << endl;

double R_n = insert_newton1_polynomial(t, n, list_diffs);
double min_Rn = min(R_n, insert_newton1_polynomial(t, n, list_diffs));
double max_Rn = max(R_n, insert_newton1_polynomial(t, n, list_diffs));
cout << "Minimum Rn on interval: " << min_Rn << endl;
cout << "Maximum Rn on interval: " << max_Rn << endl;
```

```
Newton 1: -0.132596
R_N1: 9.64784e-14
Newton 2: 1.11141
R_N2: -5.92859e-14
Gauss 1: 1.00033
R_G1: 0.566797
Gauss 2: -0.231158
R_G2: -0.664693
Minimum Rn on interval: -0.735542
Maximum Rn on interval: -0.735542
```

Вывод:

В процессе выполнения лабораторной работы были реализованы различные численные методы, включая методы Ньютона и методы Гаусса, для аппроксимации функции.

Применение методов аппроксимации:

- С помощью методов Ньютона и Гаусса были вычислены коэффициенты аппроксимирующих многочленов.
- Применили методы для вычисления значений аппроксимирующих многочленов в заданных точках.