МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное
образовательное учреждение высшего образования
“Национальный исследовательский университет ИТМО”

ФАКУЛЬТЕТ БЕЗОПАСНОСТИ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ
Кафедра проектирования и безопасности компьютерных систем

ЛАБОРАТОРНАЯ РАБОТА №7
по дисциплине
"ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА"

по теме:
«ИНТЕРПОЛЯЦИЯ И ПРИБЛИЖЕНИЕ ФУНКЦИЙ»

Выполнил: Нгуен Ле Минь
Группа: N3251
Преподаватель: Гришенцев А.Ю.

УНИВЕРСИТЕТ ИТМО

Санкт-Петербург
2021

Задание 7.1: Интерполяционные формулы Ньютона

1: Задание

Наименование задачи: вторая интерполяционная формула Ньютона.

Вид решения: программа и отчёт.

Реализация решения: язык C или C++.

Разработать алгоритм и написать программу реализующую: первую интерполяционную формулу Ньютона. Отчёт снабдить графиками. Произвести анализ результатов. Предусмотреть возможность выбора размера последовательности преобразования без перекомпиляции программы, размер последовательности N, где $N = 5, 6, 7...$. Оценить вычислительную сложность.

2: Теория

Интерполяционные формулы Ньютона представляют собой формулы вычислительной математики, применяющиеся для полиномиального интерполирования.

Если узлы интерполяции равноотстоящие и упорядочены по величине, так что $x_{i+1} - x_i = h = \text{const}$, то есть $x_i = x_0 + ih$

Интерполяционные полиномы в форме Ньютона удобно использовать, если точка интерполирования находится вблизи начала (прямая формула Ньютона) или конца таблицы (обратная формула Ньютона).

Короткая форма интерполяционной формулы Ньютона :

В случае равноудалённых центров интерполяции, находящихся на единичном расстоянии друг от друга, справедлива формула:

$P_n(x) = \sum_{m=0}^{n} C_x^m \sum_{k=0}^{m} (-1)^{m-k} C_m^k f(k)$

где C_x^m — обобщённые на область действительных чисел биномиальные коэффициенты.

Прямая (или первая) интерполяционная формула Ньютона, применяется для интерполирования вперёд:

$P_n(x) = y_0 + q\Delta y_0 + \frac{q(q-1)}{2!}\Delta^2 y_0 + \ldots + \frac{q(q-1)\ldots(q-n+1)}{n!}\Delta^n y_0$, где $q = \frac{x-x_0}{h}$, $y_i = f_i$, а выражения вида $\Delta^k y_0$ — конечные разности.

Обратная (или вторая) интерполяционная формула Ньютона, применяется для интерполирования назад:

$P_n(x) = y_n + q\Delta y_{n-1} + \frac{q(q+1)}{2!}\Delta^2 y_{n-2} + \ldots + \frac{q(q+1)\ldots(q+n-1)}{n!}\Delta^n y_0$, где $q = \frac{x-x_n}{h}$

3: Код

```
/*
 * Author : Nguyen Le Minh
 * Group : N3251
 * Lab : 7
 */
#include <iostream>
#include <vector>
using namespace std;

int fac(int x) {
    if (x <= 1) return 1;
    int result = 1;
    for (int i = 2; i <= x; i++) {
        result *= i;
    }
    return result;
}

double trlen_k(float q,int k,double delta_k) {
    //k k-ой член в пониноме P(x)
    //delta_k k-ой разность x0
    double result = 1;
    for (int i = k-1; i >=0; i--)
    {
        result *= (q - i);
```

```
    }
    result = result * delta_k / fac(k);
    return result;
}

void solve(){
    int n;
    cout << "Input N:";//размер последовательности
    cin >> n;
    int num;
    cout << "Input number of pairs value(x,y):";
    cin >> num;
    float start;
    float h;//шаг
    cout << "Input start value x0: ";
    cin >> start;
    cout << "Input step value h:";
    cin >> h;
    auto *x = (float*)malloc(sizeof(float)*num);
    auto *y = (double*)malloc(sizeof(double)*num);
    cout << "Input yi:";
    x[0] = start;
    cin >> y[0];

    //вычислим конечные разности

    vector<vector<double>> delta(n); //delta[k][i] k+1-ти разность
    int prev = 0;
    for (int i = 1; i < num; i++)
    {
        x[i] = start + i * h;
        cin >> y[i];
        delta[0].push_back(y[i] - y[prev]);//delta[0][i] первой разность
        prev = i;
    }
    int prev_k;
    for (int k = 1; k < n; k++)
    {
        prev = 0;
        prev_k = k-1;
        for (int i = 1; i < (num - k); i++)
        {
            //delta[k][i] k+1-ти разность
            delta[k].push_back(delta[prev_k][i] - delta[prev_k][prev]);
            prev = i;

        }

    }
    cout << "x y delta_y delta_2y delta_3y" << endl;
    for (int i = 0; i < num-3; i++)
    {
        cout << x[i]<<" " << y[i] << " " << delta[0][i] << " " << delta[1][i] << " " << delta[2][i
        cout << endl;
    }

    // Интерполяционные формулы Ньютона

    float x0,x_new;//x0 наболее близки к x_new по сравнению с заданной xi,x_new>x0
    cout << "Input x_new to find y_new:";
    cin >> x_new;
```

```
    cout << "Input x0 and number of x0 in series:";
    cin >> x0;
    int k;//положение x0 в заданной ряд x
    cin >> k;
    k -= 1;
    float q=(x_new-x0)/h;//число шагов для достижения точки x0
    double Px=y[k];
    for (int i = 1; i <= n; i++)
    {
        cout << trlen_k(q, i, delta[i - 1][k]) << endl;
        Px += trlen_k(q, i, delta[i - 1][k]);
    }
    cout <<"Value Y_new: "<< Px;

    free(x);
    free(y);

}

int main() {
    solve();
    return 0;
}
```

4: Выводы программы

Input N:3
Input number of pairs value(x,y):11
Input start value x0: 1
Input step value h:0.1
Input yi:0.9963
0.1145
0.744
0.1105
0.26936
0.258
0.23669
0.4471
0.9925
0.9986
0.9989
x y delta_y delta_2y delta_3y
1 0.9963 -0.8818 1.5113 -2.7743
1.1 0.1145 0.6295 -1.263 2.05536
1.2 0.744 -0.6335 0.79236 -0.96258
1.3 0.1105 0.15886 -0.17022 0.16027
1.4 0.26936 -0.01136 -0.00995 0.24167
1.5 0.258 -0.02131 0.23172 0.10327
1.6 0.23669 0.21041 0.33499 -0.87429
1.7 0.4471 0.5454 -0.5393 0.5335
Input x_new to find y_new:1.43
Input x0 and number of x0 in series:1.4 5
-0.003408
0.00104475
0.0143794
Value Y_new: 0.281376

Вывод: В ход работе мы написали программу реализующую: первую интерполяционную формулу Ньютона Предусмотреть возможность выбора размера последовательности n На данной пример $n = 3$,ввод 11 пары значение (x, y) ,найти $f(1.43)$ Сложность $O(n * k^2)$

Задание 7.2: Интерполяция Гаусса

1: Задание

Наименование задачи: первая и вторая интерполяционная формула Гаусса.

Вид решения: программа и отчёт.

Реализация решения: язык C или C++.

Разработать алгоритм и написать программу реализующую: интерполяцию на основе первой и второй интерполяционной формулы Гаусса. Произвести сравнение и анализ результатов. Предусмотреть возможность выбора размера последовательности преобразования без перекомпиляции программы, размер последовательности N, где $N = 5, 6, 7\ldots$. Оценить вычислительную сложность.

2: Теория

Интерполяционная формула Гаусса — формула, использующая в качестве узлов интерполяции ближайшие к точке интерполирования x узлы. Если $x = x_0 + th$, то формула :

$$G_{2n+1}(x_0 + th) = f_0 + f^1_{1/2}t + f^2_0\frac{t(t-1)}{2!} + \ldots + f^{2n}_0\frac{t(t^2-1)\ldots[t^2-(n-1)^2](t-n)}{(2n)!}, \qquad (1)$$

написанная по узлам $x_0,\ x_0 + h,\ x_0 - h, \ldots,\ x_0 + nh,\ x_0 - nh$, называется формулой Гаусса для интерполирования вперед, а формула :

$$G_{2n+1}(x_0 + th) = f_0 + f^1_{-1/2}t + f^2_0\frac{t(t+1)}{2!} + \ldots + f^{2n}_0\frac{t(t^2-1)\ldots[t^2-(n-1)^2](t+n)}{(2n)!}, \qquad (2)$$

написанная по узлам $x_0,\ x_0 - h,\ x_0 + h, \ldots,\ x_0 - nh,\ x_0 + nh$, называется формулой Гаусса для интерполирования назад. В формулах (1) и (2) использованы конечные разности, определяемые следующим образом:

$f^1_{i+1/2} = f_{i+1} - f_i,\ f^m_i = f^{m-1}_{i+1/2} - f^{m-1}_{i-1/2}$

3: Код

```
/*
 * Author : Nguyen Le Minh
 * Group : N3251
 * Lab : 7.2
 */
#include<iostream>
#include<vector>
using namespace std;

int fac(int x) {
    if (x <= 1) return 1;
    int result = 1;
    for (int i = 2; i <= x; i++) {
        result *= i;
    }
    return result;
}
//k-ой член первая интерполяционная формула Гаусса
double trlen_k_Gauss_1(float q, int k, vector<vector<double>> delta_k,int index_x0) {
    //k k-ой член в пониноме P(x)
    //delta_k k-ой разность x0
    double result = 1;
    int tmp;
    if (k % 2 == 1) {
        tmp = (k - 1) / 2;//tmp=n-1
        q = q + tmp;
        for (int i = 0; i < k; i++)
        {
            result *= (q - i);
        }
        result = result * delta_k[k-1][index_x0-tmp] / fac(k);
    }
    else
```

```
    {
        tmp = k / 2 - 1;
        q += tmp;
        for (int i = 0; i < k; i++)
        {
            result *= (q - i);
        }
        result = result * delta_k[k - 1][index_x0 - k/2] / fac(k);
    }

    return result;
}
//k-ой член вторая интерполяционная формула Гаусса
double trlen_k_Gauss_2(float q, int k, vector<vector<double>> delta_k, int index_x0) {
    //k k-ой член в пониноме P(x)
    //delta_k k-ой разность x0
    double result = 1;
    int tmp;
    if (k % 2 == 1) {
        tmp = (k - 1) / 2;//tmp=n-1
        q = q + tmp;
        for (int i = 0; i < k; i++)
        {
            result *= (q - i);
        }
        result = result * delta_k[k - 1][index_x0 - tmp-1] / fac(k);
    }
    else
    {
        tmp = k / 2 - 1;
        q += tmp;
        for (int i = 0; i < k; i++)
        {
            result *= (q - i);
        }
        result = result * delta_k[k - 1][index_x0 - k / 2] / fac(k);
    }

    return result;
}

void solver(){
    int n;
    cout << "Input N:";//размер последовательности
    cin >> n;
    int num;
    cout << "Input number of pairs value(x,y):";
    cin >> num;
    float start;
    float h;//шаг
    cout << "Input start value x0: ";
    cin >> start;
    cout << "Input step value h:";
    cin >> h;
    auto *x = (float*)malloc(sizeof(float)*num);
    auto *y = (double*)malloc(sizeof(double)*num);
    cout << "Input yi:";
    x[0] = start;
    cin >> y[0];

    //вычислим конечные разности
```

```
vector<vector<double>> delta(n); //delta[k][i] k+1-ти разность
int prev = 0;
for (int i = 1; i < num; i++)
{
    x[i] = start + i * h;
    cin >> y[i];
    delta[0].push_back(y[i] - y[prev]);//delta[0][i] первой разность
    prev = i;
}
int prev_k;
for (int k = 1; k < n; k++)
{
    prev = 0;
    prev_k = k - 1;
    for (int i = 1; i < (num - k); i++)
    {
        //delta[k][i] k+1-ти разность
        delta[k].push_back(delta[prev_k][i] - delta[prev_k][prev]);
        prev = i;

    }

}
cout << "x y delta_y delta_2y delta_3y" << endl;
for (int i = 0; i < num - 3; i++)
{
    cout << x[i] << " " << y[i] << " " << delta[0][i] << " " << delta[1][i] << " " << delta[2]
    cout << endl;
}

// Интерполяционные формулы Gauss

float x0, x_new;//x0 наболее близки к x_new по сравнению с заданной xi,x_new>x0
cout << "Input x_new to find y_new:";
cin >> x_new;
cout << "Input x0 and number of x0 in series:";
cin >> x0;
int index_x0;//положение x0 в заданной ряд x
cin >> index_x0;
index_x0 -= 1;
float q = (x_new - x0) / h;//число шагов для достижения точки x0
cout << "Value Y_new\n";
double Px = y[index_x0];

// Gauss 1 formula

for (int i = 1; i <= n; i++)
{
    Px += trlen_k_Gauss_1(q, i, delta, index_x0);
}
cout << "By Gauss 1 formula: " << Px<<endl;
// Gauss 2 formula
Px = y[index_x0];
for (int i = 1; i <= n; i++)
{
    Px += trlen_k_Gauss_2(q, i, delta, index_x0);
}
cout << "By Gauss 2 formula: " << Px;
free(x);
free(y);
```

```
}

int main() {
    solver();
    return 0;
}
```

4: Выводы программы

Input N:3
Input number of pairs value(x,y):7
Input start value x0: 0.2
Input step value h:0.05
Input yi:1.552
1.6719
1.7831
1.8847
1.9759
2.0563
2.125
x y delta_y delta_2y delta_3y
0.2 1.552 0.1199 -0.0087 -0.0009
0.25 1.6719 0.1112 -0.0096 -0.0008
0.3 1.7831 0.1016 -0.0104 -0.0004
0.35 1.8847 0.0912 -0.0108 -0.0009
Input x_new to find y_new:0.356
Input x0 and number of x0 in series:0.35 4
Value Y_new
By Gauss 1 formula: 1.8962
By Gauss 2 formula: 1.89746
Проверка :
Приняв шаг $h = 0.05$, построить интерполяционный полином Гаусса для функции $y = -x^3 - x^2 + 3x + 1$, заданной таблицей :

x	0.2	0.25	0.3	0.35	0.4	0.45	0.5
y	1.552	1.6719	1.7831	1.8847	1.9759	2.0563	2.125

полагаем $n = 3$. Приняв $x_0 = 0.35$, $y_0 = 1.8847$, будем иметь:
по первой интерполяционной формуле Гаусса :
$P_n(x) = 1.8847 + 0.0912q - 0.0104\frac{q(q-1)}{2} - 0.0004\frac{(q+1)q(q-1)}{6}$
по второй интерполяционной формуле Гаусса :
$P_n(x) = 1.8847 + 0.1016q - 0.0104\frac{q(q-1)}{2} - 0.0008\frac{(q+1)q(q-1)}{6}$
где $q = \frac{x-0.35}{0.05} = 20(x - 0.35)$
Вычисление значения функций в $x = 0.356$. Программ выводит правильно результат
Мы научились средствами языка C++ реализовывать приближение функции в заданной в равно-отстоящих точках с помощью сплайна третьей степени.

Задание 7.3: Сплайн интерполяция

1: Задание

Наименование задачи: найти приближение функции заданной в равноотстоящих точках, т.е. функции заданной в виде последовательности чисел, с помощью сплайна третьей степени.

Вид решения: программа и отчёт.

Реализация решения: язык C или C++.

Разработать алгоритм и написать программу реализующую: найти приближение функции заданной в равноотстоящих точках, т.е. функции заданной в виде последовательности чисел, с помощью сплайна третьей степени. Произвести анализ результатов. Предусмотреть возможность выбора размера последовательности преобразования без перекомпиляции программы, размер последовательности N, где $N = 5, 6, 7 \ldots$. Оценить вычислительную сложность.

2: Теория

Сплайн – функция, которая вместе с несколькими производными непрерывна на всем заданном отрезке $[a, b]$,а на каждом частичном отрезке $[x_i, x_{i+1}]$ в отдельности является некоторым алгебраическим многочленом.

Степенью сплайна называется максимальная по всем частичным отрезкам степень многочленов, а дефектом сплайна - разность между степенью сплайна и порядком наивысшей непрерывной на $[a, b]$ производной. Например, непрерывная ломанная является сплайном степени 1 с дефектом 1 (так как сама функция – непрерывна, а первая производная уже разрывна).

На практике наиболее часто используются кубические сплайны $S_3(x)$ сплайны третьей степени с непрерывной, по крайней мере, первой производной. При этом величина $m_i = S_3'(x_i)$, называется наклоном сплайна в точке (узле) x_i.

Разобьём отрезок $[a, b]$ на N равных отрезков $[x_i, x_{i+1}]$, где $x_i = a + ih$, $i = 0, 1, ..., N-1$, $x_N = b, h = \frac{b-a}{N}$

Если в узлах $[x_i, x_{i+1}]$ заданы значения f_i, f_{i+1}, которые принимает кубический сплайн, то на частичном отрезке $[x_i, x_{i+1}]$ он принимает вид :

$$S_3(x) = \frac{(x_{i+1}-x)^2(2(x-x_i)+h)}{h^3} f_i + \frac{(x-x_i)^2(2(x_{i+1}-x)+h)}{h^3} f_{i+1} + \frac{(x_{i+1}-x)^2(x-x_i)}{h^2} m_i + \frac{(x-x_i)^2(x-x_{i+1})}{h^2} m_{i+1}. \quad (1)$$

В самом деле, это легко проверить, рассчитав $S_3(x)$ и $S_3'(x)$ в точках x_i, x_{i+1}

Можно доказать, что если многочлен третьей степени принимает в точках x_i, x_{i+1} значения f_i, f_{i+1} и имеет в этих точках производные, соответственно, m_i, m_{i+1}, то он совпадает с многочленом (1).

Таким образом, для того, чтобы задать кубический сплайн на отрезке, необходимо задать значения $f_i, m_i, i = 0, 1, 2, ..., N$ в $N + 1$ в узле x_i.

Кубический сплайн, принимающий в узлах те же значения f_i, что и некоторая функция, называется интерполяционным и служит для аппроксимации функции f на отрезке $[a, b]$ вместе с несколькими производными.

3: Код

```
/*
 * Author : Nguyen Le Minh
 * Group : N3251
 * Lab : 7.3
 */
#include <cmath>
#include <iostream>

float *x, *y, *h, *l, *delta, *lambda, *c, *d, *b;
int N;
char filename[256];
FILE* InFile=nullptr;

void count_num_lines(){
    //count number of lines in input file - number of equations
    int nelf=0;        //non empty line flag
    do{
        nelf = 0;
```

```
        while(fgetc(InFile)!='\n' && !feof(InFile)) nelf=1;
        if(nelf) N++;
    }while(!feof(InFile));
    N--;
}

void readmatrix(){
    int i=0;
    //read matrixes a and b from input file
    for(i=0; i<N+1; i++){
        fscanf(InFile, "%f", &x[i]);
        fscanf(InFile, "%f", &y[i]);
    }
}

void allocmatrix(){
    //allocate memory for matrixes
    x = new float[N+1];
    y = new float[N+1];
    h = new float[N+1];
    l = new float[N+1];
    delta = new float[N+1];
    lambda = new float[N+1];
    c = new float[N+1];
    d = new float[N+1];
    b = new float[N+1];
}

void freematrix(){
    delete [] x;
    delete [] y;
    delete [] h;
    delete [] l;
    delete [] delta;
    delete [] lambda;
    delete [] c;
    delete [] d;
    delete [] b;
}

void printresult(){
    int k=0;
    printf("Таблица результатов:\n");
    printf("A[k]\tB[k]\tC[k]\tD[k]\n");
    for(k=1; k<=N; k++){
        printf("%f\t%f\t%f\t%f\n", y[k], b[k], c[k], d[k]);
    }
}

void solver(){
    int k=0;
    printf("Введите имя файла с точками x y\\n:");
    do{
        printf("\nInput filename: ");
        scanf("%s", filename);
        InFile = fopen(filename, "rt");
    }while(InFile==nullptr);
    count_num_lines();
    rewind(InFile);
    allocmatrix();
    readmatrix();
```

```
    for(k=1; k<=N; k++){
        h[k] = x[k] - x[k-1];
        if(h[k]==0){
            printf("\nError, x[%d]=x[%d]\n", k, k-1);
            exit(1);
        }
        l[k] = (y[k] - y[k-1])/h[k];
    }
    delta[1] = - h[2]/(2*(h[1]+h[2]));
    lambda[1] = 1.5*(l[2] - l[1])/(h[1]+h[2]);
    for(k=3; k<=N; k++){
        delta[k-1] = - h[k]/(2*h[k-1] + 2*h[k] + h[k-1]*delta[k-2]);
        lambda[k-1] = (3*l[k] - 3*l[k-1] - h[k-1]*lambda[k-2]) /
                      (2*h[k-1] + 2*h[k] + h[k-1]*delta[k-2]);
    }
    c[0] = 0;
    c[N] = 0;
    for(k=N; k>=2; k--)
        c[k-1] = delta[k-1]*c[k] + lambda[k-1];
    for(k=1; k<=N; k++){
        d[k] = (c[k] - c[k-1])/(3*h[k]);
        b[k] = l[k] + (2*c[k]*h[k] + h[k]*c[k-1])/3;
    }
    printresult();
    freematrix();
}

int main(){
    solver();
    return 0;
}
```

4: Выводы программы

Введите имя файла с точками x y:
Input filename: test.txt (содержание файла:
1 2
3 4
5 6
7 8
9 10
7 11)
Таблица результатов:
A[k] B[k] C[k] D[k]
4.000000 0.800000 -0.150000 -0.025000
6.000000 1.700000 0.600000 0.125000
8.000000 -1.600000 -2.250000 -0.475000
10.000000 10.7000001 8.400001 1.775000
11.000000 -6.100000 0.000000 1.400000

Мы научились средствами языка C++ реализовывать приближение функции в заданной в равноотстоящих точках с помощью сплайна третьей степени.

Задание 7.4: Фурье интерполяция

1: Задание

Наименование задачи: найти приближение функции заданной в равноотстоящих точках, т.е. функции заданной в виде последовательности чисел, с помощью интерполяции Фурье.

Вид решения: программа и отчёт.

Реализация решения: язык C или C++.

Разработать алгоритм и написать программу реализующую: найти приближение функции заданной в равноотстоящих точках, т.е. функции заданной в виде последовательности чисел, с помощью интерполяции Фурье. Произвести анализ результатов. Предусмотреть возможность выбора размера последовательности преобразования без перекомпиляции программы, размер последовательности N, где $N = 2n$, $n = 1, 2, 3, ...$ Оценить вычислительную сложность.

2: Теория

Для использования математических формул для тригонометрической интерполяцией необходимо ознакомиться с теорией дискретного преобразования Фурье.

В прикладных задачах часто используются различные преобразования Фурье функций непрерывного аргумента, а также представлений функций с помощью сходящихся тригонометрических рядов. Всякую непрерывно дифференцируемую функцию f можно разложить в ряд Фурье:

В прикладных задачах часто используются различные преобразования Фурье функций непрерывного аргумента, а также представлений функций с помощью сходящихся тригонометрических рядов. Всякую непрерывно дифференцируемую функцию f можно разложить в ряд Фурье:

$$f(x)=\sum_{k=-\infty}^{\infty}\alpha_k e^{2\pi ikx}$$

коэффициенты α_k находятся по следующим формулам

$$\alpha_k=\int_0^1 f(x)e^{-2\pi ikx}dx$$

Но как правила функция задана только в некоторых точках или у нас есть возможность узнать её значения только в некотором конечном числе точек. Допустим, $x_j=j/N, j=0,1,\cdots,N-1$.В этом случае аналогом функции непрерывной интерполяции функции будет дискретный вариант:

$$f(x_j)=\sum_{k=0}^{N-1}\alpha_k e^{2\pi ikx_j}, 0\le j<N$$

Разложение имеет место когда функцию можно приблизить тригонометрическим многочленом следующего вида в заданных нам точках

$$S_N(x)=\sum_{k=0}^{N-1}a_k e^{2\pi ikx}$$

Система функций $\phi(x)=2\pi kx, 0\le k<N$ является ортогональной, на множестве точек $x_j=j/N, 0\le j<N$ при том что $(\phi_k,\phi_k)=N$, таким образом разложение имеет место и коэффициенты a_k представляются в виде:

$$a_k=\frac{1}{N}\sum_{l=0}^{N-1}f(x_l)e^{-2\pi ikx_l}, 0\le k<N$$

Далее для удобства записи будем использовать $\omega=e^{2\pi i/N}$

Часто используется следующий вид формул:

$f(x_j)=\sum_{-N/2<k\le N/2}a_k e^{2\pi ikx_j}$, и это соответствует интерполяции тригонометрическим многочленом

$$S_N=\sum_{-N/2<k\le N/2}a_k e^{2\pi ikx},$$

где коэффициенты a_k считаются по тем же формулам.

Если вычисления проводить по вышеприведённым формулам, то на выполнения каждого из преобразований потребуется N^2 арифметических операций (считаем, что $\omega=e^{2\pi i/N}$ уже вычислены). Если N не является простым числом, то количество операций можно значительно сократить, используя быстрое преобразование Фурье.

3: Код

```
/*
 * Author : Nguyen Le Minh
 * Group : N3251
 * Lab : 7.4
 */

#include <iostream>
#include <complex>
#include <cmath>
#include <vector>
using namespace std;

typedef complex<double> base;

#define pi 3.14159265358979323846

void multiply(vector<base> &r,vector<base> &t,vector<base> &ans,int n){
    base change;
    base q = 0;
```

```
    for(int l=0; l<n; l++){
        for(int j=0; j<n; j++){
            change = r[l*n + j]*t[j];
            q = q+change;
        }
        ans[l] = q/(base)n;
        q = 0;
    }
}

void solver(){
    int N;
    cout << "Введите количество элементов (1,2,4,8...): ";
    cin >> N;
    vector<base> a(N*N);
    vector<base> arr(N);
    vector<base> ans(N);
    cout << "Введите вектор комплексный типа (a,b): " << std::endl;
    for (int i = 0; i < N; i++){
        cout << "Введите элемент arr[" << i << "]: ";
        cin >> arr[i];
        for(int j = 0; j < N; j++){
            base tmp(cos(-2*(i)*(j)*pi/N),sin(-2*(i)*(j)*pi/N));
            a[i*N + j] = tmp;
        }
    }
    multiply(a, arr, ans, N);
    for(int i = 0; i < N; i++)
        cout << ans[i] << endl;
}

int main(){
    solver();
    return 0;
}
```

4: Выводы программы

Введите количество элементов (1,2,4,8...): 4
Введите вектор комплексный типа (a,b):
Введите элемент arr[0]: (7,2)
Введите элемент arr[1]: (9,3)
Введите элемент arr[2]: (12,1)
Введите элемент arr[3]: (13,3)
(10.25,2.25)
(-1.25,1.25)
(-0.75,-0.75)
(-1.25,-0.75)

При выполнении этого задания мы научились средствами языка C++ реализовывать приближение функции с помощью интерполяции Фурье.