МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №6

по дисциплине «Вычислительная математика»

Тема: Интерполяционные и аппроксимационные формулы для равноотстоящих узлов

Студент гр. 0304 _____ Алексеев Р.В.

Преподаватель _____ Попова Е.В.

Санкт-Петербург

2021

Вариант 1

Цель работы: исследование методов интерполяции и аппроксимации для равноотстоящих узлов с последующей реализацией на одном из языков программирования

Основные теоретические положения.

Значения функции $f(x_i)=y_i$ заданы в точках $x_i=x_0+nh \quad i \in \{0,\dots,n\}$, необходимо найти промежуточные значения.

Интерполяционный многочлен Лагранжа.

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

где $L_n(x)$ – множитель Лагранжа

$$L_n(x)=\frac{(x-x_0)\dots(x-x_{i-1})(x-x_{i+1})\dots(x-x_n)}{(x_i-x_0)\dots(x_i-x_{i-1})(x_i-x_{i+1})\dots(x_i-x_n)}=\prod_{\substack{k=0\\k\neq i}}^{n}\frac{(x-x_k)}{(x_i-x_k)}.$$

Следовательно

$$P_n(x)=\sum_{i=0}^{n} y_i\left(\prod_{\substack{k=0\\k\neq i}}^{n}\frac{x-x_k}{x_i-x_k}\right).$$

Первый интерполяционный многочлен Ньютона. Точка интерполирования находится в начале таблицы.

Интерполирующий полином ищется в виде

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

Построение многочлена сводится к определению коэффициентов a_i.. При записи коэффициентов пользуются конечными разностями.

Конечные разности первого порядка запишутся в виде:

$$\Delta y_0 = y_1 - y_0;$$

$$\Delta y_1 = y_2 - y_1;$$

$$\dots$$

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

где y_i – значения функции при соответствующих значениях x_i.

Конечные разности второго порядка:

$$\Delta^2 y_0 = \Delta y_1 - \Delta y_0;$$

$$\Delta^2 y_1 = \Delta y_2 - \Delta y_1;$$

$$\dots$$

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

Конечные разности высших порядков найдутся аналогично:

$$\Delta^k y_0 = \Delta^{k-1} y_1 - \Delta^{k-1} y_0;$$

$$\Delta^k y_1 = \Delta^{k-1} y_2 - \Delta^{k-1} y_1;$$

$$\dots$$

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

Общая формула для нахождения всех коэффициентов имеет вид

$$a_i = \frac{\Delta^i y_0}{i! h^i},$$

где i=1…n.

В результате

$$P_n(x) = y_0 + \frac{\Delta y_0}{1! \cdot h}(x - x_0) + \frac{\Delta^2 y_0}{2! \cdot h^2}(x - x_0)(x - x_1) + ...$$

$$+ \frac{\Delta^n y_0}{n! \cdot h^n}(x - x_0)...(x - x_{n-1}).$$

Данный многочлен называют первым полиномом Ньютона.

Второй интерполяционный многочлен Ньютона. Точка интерполирования находится в конце таблицы.

Для нахождения значений функции в конце интервала интерполирования интерполяционный полином запишется в виде

$$P_n(x) = a_0 + a_1(x - x_n) + a_2(x - x_n)(x - x_{n-1}) + \ldots$$
$$\ldots + a_n(x - x_n)(x - x_{n-1})\ldots(x - x_1).$$

Формула для нахождения всех коэффициентов запишется как:

$$a_i = \frac{\Delta^i y_{n-1}}{i! h^i}.$$

получим вторую интерполяционную формулу Ньютона:

$$P_n(x) = y_n + \frac{\Delta y_{n-1}}{h}(x - x_n) + \frac{\Delta^2 y_{n-2}}{2! h^2}(x - x_n)(x - x_{n-1}) +$$
$$+ \frac{\Delta^3 y_{n-3}}{3! h^3}(x - x_n)(x - x_{n-1})(x - x_{n-2}) + \ldots$$
$$\ldots + \frac{\Delta^n y_0}{n! h^n}(x - x_n)(x - x_{n-1})\ldots(x - x_1).$$

Аппроксимация функции. Необходимо найти эмпирическую формулу, значения которой при $x = x_i$ мало бы отличались от входных данных. Будет использоваться линейная аппроксимация, при которой данные описываются линейной зависимостью

$$P(x) = a \cdot x + b.$$

Формулы для расчета коэффициентов a и b определяются по методу наименьших квадратов

$$F = \sum_{i=1}^{n} (y_i - a \cdot x_i - b)^2 \to \min.$$

$$\begin{cases} \dfrac{dF}{db} = -2 \cdot \sum\limits_{i=1}^{n} (y_i - a \cdot x_i - b) \cdot 1 = 0, \\ \dfrac{dF}{da} = -2 \cdot \sum\limits_{i=1}^{n} (y_i - a \cdot x_i - b) \cdot x_i = 0. \end{cases}$$

$$a = \frac{n \cdot \sum_{i=1}^{n}(x_i \cdot y_i) - \sum_{i=1}^{n} x_i \cdot \sum_{i=1}^{n} y_i}{n \cdot \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2},$$

$$b = \frac{\sum_{i=1}^{n} y_i - a\sum_{i=1}^{n} x_i}{n} = \frac{\sum_{i=1}^{n} y_i \cdot \sum_{i=1}^{n} x_i^2 - \sum_{i=1}^{n} x_i \cdot \sum_{i=1}^{n}(x_i \cdot y_i)}{n \cdot \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}.$$

Порядок выполнения работы.

1. Выбрать три точки, не входящие таблицу данных в начале, конце, по середине таблицы.

2. Составить необходимые подпрограммы-функции.

3. Составить головную программу, содержащую обращение к соответствующим подпрограммам и осуществляющую печать результатов

4. Используя многочлен Лагранжа, найти приближенные значения функции. Подсчитать точные значения, абсолютную погрешность, относительную погрешность, верные и значащие цифры.

5. Используя многочлены Ньютона найти все, что перечислено в предыдущем пункте, дополнительно – в процентном соотношении точку приемлемого использования первого многочлена по сравнению со вторым многочленом при работе с точкой в начале таблицы и наоборот.

6. Составить таблицу конечных разностей до второго порядка. К одному из значений y_i прибавить погрешность (α), превосходящую допустимую. Показать, как при этом изменятся конечные разности большего порядка.

7. При аппроксимировании функции использовать для нахождения новых значений те же три точки. Рассчитать все, что указано в пункте 4.

8. Составить сводную таблицу по результатам исследования.

Выполнение работы.

Функция: $y = \sin\left(\frac{\pi * x}{2}\right)$

x	**y**
-1	-1
-0,6	-0,81
-0,2	-0,31
0,2	0,31
0,6	0,81
1	1

Точки для вычислений: -0,72, 0,13, 0,68

1. Теорема:

   Для табличной функции существует единственный интерполяционный многочлен, степень которого не выше N, где N — количество данных в таблице.

   Доказательство:

   Интерполяционный многочлен будет иметь вид $P_N(x)=c_0+c_1x+...+c_{N-1}x^{N-1}+c_Nx^N$, где c_i — искомые коэффициенты. При помощи таблицы с данными получим СЛУ вида:

$c_0+c_1x_0+...+c_Nx_0^N=P_N(x_0)=y_0$

…

$c_0+c_1x_N+...+c_Nx_N^N=P_N(x_N)=y_N$

или в виде матриц и векторов:

$$A=\begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^N \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_N & x_N^2 & \dots & x_N^N \end{pmatrix},\ c=(c_0,\dots,c_N)^T,\ y=(y_0,\dots,y_N)^T$$

Система однозначно разрешина в случае если определитель матрицы A не равен нулю. Определитель матрицы A является определителем Вандермонда: $det(A)=\prod_{0\le j\le i\le N<?}(x_i-x_j)$, значит он не равен нулю если x_i не равно x_j при любых неодинаковых j и i. Следовательно коэффициенты c_i находятся однозначно т. е. интерполяционный многочлен существует и при этом только один.

2. При помощи многочлена Лагранжа, многочленов Ньютона вперед и назад, линейной аппроксимации найдем приближенное значение функции в первой точке -0,72. Результаты работы программы с выводом точного значения функции в точке, абсолютной и относительнйо погрешностей, значащих и верных цифр представлены на рис. 1.

Метод	X	Знач в точке	Вычисленное	Абс погреш	Отн погреш	Знач цифры	Верные цифры
Лагранж	-0.7200	-0.90482705	-0.90516015	0.00033310	0.00036813	90516015	905
Форвард	-0.7200	-0.90482705	-0.90516015	0.00033310	0.00036813	90516015	905
Ньютон обр ход	-0.7200	-0.90482705	-0.90516015	0.00033310	0.00036813	90516015	905
Аппроксимация	-0.7200	-0.90482705	-0.79611429	0.10871277	0.12014756	79611428	0

Рисунок 1 — Значения функции в точке -0,72

По таблице результатов видно, что все методы дают одинаковый результат в точке, но точность результатов, полученного интерполяционными многочленами, выше точности результата, полученного методом линейной аппроксимации. Также из-за погрешности вычисления в значения в точке количество верных чисел различно: для интерполяционных многочленов — 3, для метода

линейной аппроксимации — 0. Относительная погрешность для метода линейной аппроксимации выше относительной погрешности для интерполяиционных многочленов, 12,015% и 0,037% соответсвенно.

3. Аналогично при помощи многочлена Лагранжа, многочленов Ньютона вперед и назад, линейной аппроксимации найдем приближенное значение функции в первой точке 0,13. Результаты работы программы с выводом точного значения функции в точке, абсолютной и относительнйо погрешностей, значащих и верных цифр представлены на рис. 2.

Метод	X	Знач в точке	Вычисленное	Абс погреш	Отн погреш	Знач цифры	Верные цифры
Лагранж	0.1300	0.20278730	0.20346072	0.00067343	0.00332086	20346072	203
Форвард	0.1300	0.20278730	0.20346072	0.00067343	0.00332086	20346072	203
Ньютон обр ход	0.1300	0.20278730	0.20346072	0.00067343	0.00332086	20346072	203
Аппроксимация	0.1300	0.20278730	0.14374286	0.05904444	0.29116439	14374285	1

Рисунок 2 — Значения функции в точке 0,13

По таблице результатов видно, что точность полученного результата аналогична предыдущему случаю. Также видно, что из-за погрешности вычисления в значения в точке количество верных чисел различно: для интерполяционных многочленов — 3, для метода линейной аппроксимации — 1. Относительная погрешность для метода линейной аппроксимации выше относительной погрешности для интерполяиционных многочленов, 29,12% и 0,33% соответсвенно.

4. Аналогично первым двум случаям при помощи многочлена Лагранжа, многочленов Ньютона вперед и назад, линейной аппроксимации найдем приближенное значение функции в первой точке 0,68. Результаты работы программы с выводом точного значения функции в точке, абсолютной и относительнйо погрешностей, значащих и верных цифр представлены на рис. 3.

Метод	X	Знач в точке	Вычисленное	Абс погреш	Отн погреш	Знач цифры	Верные цифры
Лагранж	0.6800	0.87630668	0.87687360	0.00056692	0.00064694	87687360	876
Форвард	0.6800	0.87630668	0.87687360	0.00056692	0.00064694	87687359	876
Ньютон обр ход	0.6800	0.87630668	0.87687360	0.00056692	0.00064694	87687360	876
Аппроксимация	0.6800	0.87630668	0.75188571	0.12442097	0.14198336	75188571	0

Рисунок 3 — Значения функции в точке 0,68

По таблице результатов видно, что точность полученного результата аналогична предыдущим случаям. Также видно, что из-за погрешности вычисления в значения в точке количество верных чисел различно: для интерполяционных многочленов — 3, для метода линейной аппроксимации — 0. Относительная погрешность для метода линейной аппроксимации выше относительной погрешности для интерполяиционных многочленов, 14,198% и 0,065% соответсвенно.

5. Сравним таблицы конечных результатов с и без внесения погрешности в одно из значений. Таблицы представлены на рис. 4.

Конечная разность без погрешности

X	Y	Δy	Δ^2y
-1	-1	0.19	0.31
-0.6	-0.81	0.5	0.12
-0.2	-0.31	0.62	-0.12
0.2	0.31	0.5	-0.31
0.6	0.81	0.19	-
1	1	-	-

Конечная разность с погрешностью

X	Y	Δy	Δ^2y
-1	-1	0.19	0.155
-0.6	-0.81	0.345	0.43
-0.2	-0.465	0.775	-0.275
0.2	0.31	0.5	-0.31
0.6	0.81	0.19	-
1	1	-	-

Рисунок 4 — Таблицы конечных результатов с и без внесения погрошености в одно из значений.

На рисунке видно, что при внисении погрешности в одно из значений y_i, на такое же значение будут в изменяться конечные значения delta y_i и delta

y_{i-1}, а значит будут изменяться и конечные разности порядков выше, следовательно, чем выше порядок, тем сильнее влияние внесенной погрешности. В таблице 1 представленны возмущения конечных разностей.

Таблица 1. Возмущения конечных разностей до 2-ого порядка.

X	Y	delta y*	$delta^2$ y*
x_0	y_0	y_1-y_0	y_2+alpha-2y_1+y_0
x_1	y_1	y_2+alpha-y_1	y_2-2y_2-2alpha+y_1
x_2	y_2 + alpha	y_3-y_2-alpha	y_4-2y_3+y_2+alpha
x_3	y_3	y_4-y_3	y_5-2y_4+y_3
x_4	y_4	y_5-y_4	-
x_5	y_5	-	-

По таблице видно, что чем больше порядок конечной разности, тем больше возмущение разностей. Возмущение может достигать величины 2^{n-1}alpha, где n — порядок.

6. Составим таблицы результатов вычислений значений функции в точках -0,72, 0,13, 0,68 и сравним результаты при интерполяции и аппроксимации. Результаты представлены на рис. 5.

Метод	X	Знач в точке	Вычисленное	Абс погреш	Отн погреш	Знач цифры	Верные цифры
Интерполяция							
	-0.7200	-0.90482705	-0.90516015	0.00033310	0.00036813	90516015	905
	0.1300	0.20278730	0.20346072	0.00067343	0.00332086	20346072	203
	0.6800	0.87630668	0.87687360	0.00056692	0.00064694	87687360	876

Метод	X	Знач в точке	Вычисленное	Абс погреш	Отн погреш	Знач цифры	Верные цифры
Аппроксимация							
	-0.7200	-0.90482705	-0.79611429	0.10871277	0.12014756	79611428	0
	0.1300	0.20278730	0.14374286	0.05904444	0.29116439	14374285	1
	0.6800	0.87630668	0.75188571	0.12442097	0.14198336	75188571	0

Рисунок 5 — Сравнение интерполяции и аппроксимации.

По рисунку видно, что интерполяция многочленом для всех X дает значениее точнее чем линейная аппроксимация. Количество верных чисел также больше у метода интерполяции — 3 для всех точек, чем у методя аппроксимации — 0 для -0,72 и 0,68 и 1 для 0,13. Относительная погрешность при интерполяции меньше для всех точек (0,037% для -0,72,

0,332% для 0,13, 0,065% для 0,68) по сравнению с методом линейной аппроксимации (12,015% для -0,72, 29,116% для 0,13, 14,198% для 0,68). Построим графики интерполяции (синий), аппроксимации (красный), функции (зеленый). Графики представленны на рис. 6.

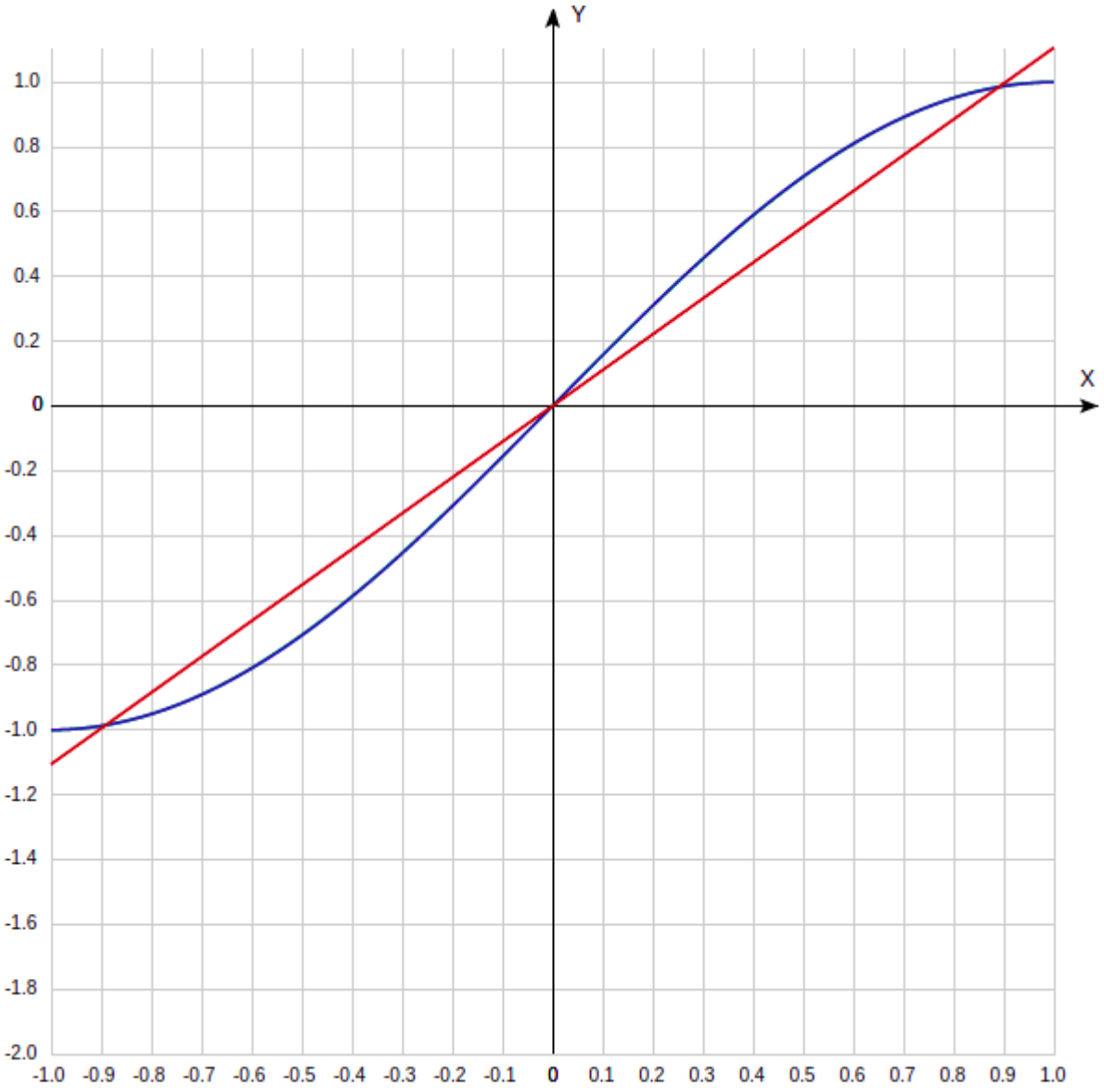

Рисунок 6 — Графики интерполяции (синий), аппроксимации (красный), функции (зеленый).

7. Найдем формулы интерполяционных многочленов Лангранжа и Ньютона назад и вперед, линейной аппроксимации. Формулы представленны на рис. 7.

```
Полином Лагранжа
f(x) = 0 + (-1)(-1.2288)(x - -0.6)(x - -0.2)(x - 0.2)(x - 0.6)(x - 1)
 + (-0.81)(0.24576)(x - -1)(x - -0.2)(x - 0.2)(x - 0.6)(x - 1)
 + (-0.31)(-0.12288)(x - -1)(x - -0.6)(x - 0.2)(x - 0.6)(x - 1)
 + (0.31)(0.12288)(x - -1)(x - -0.6)(x - -0.2)(x - 0.6)(x - 1)
 + (0.81)(-0.24576)(x - -1)(x - -0.6)(x - -0.2)(x - 0.2)(x - 1)
 + (1)(1.2288)(x - -1)(x - -0.6)(x - -0.2)(x - 0.2)(x - 0.6)

Полином Ньютона прямой ход
-1 + (0.19/0.4)(x - -1)
 + (0.31/0.32)(x - -1)(x - -0.6)
 + (-0.19/0.384)(x - -1)(x - -0.6)(x - -0.2)
 + (-0.05/0.6144)(x - -1)(x - -0.6)(x - -0.2)(x - 0.2)
 + (0.1/1.2288)(x - -1)(x - -0.6)(x - -0.2)(x - 0.2)(x - 0.6)

Полином Ньютона обратный ход
1 + (-0.05/0.4)(x - 1)
 + (-0.19/0.32)(x - 1)(x - 0.6)
 + (0.31/0.384)(x - 1)(x - 0.6)(x - 0.2)
 + (0.19/0.6144)(x - 1)(x - 0.6)(x - 0.2)(x - -0.2)
 + (-1/1.2288)(x - 1)(x - 0.6)(x - 0.2)(x - -0.2)(x - -0.6)

Аппроксимация
1.10571x + 0
```

Рисунок 7 — Формулы интерполяционных полиномов, линейной аппроксимации.

Разработанный программный код см. в приложении А.

Выводы.

Были исследованны методы интерполяции и аппроксимации функции для случая равноотстоящих узлов. Была создана программа, находящая значение функции в точке, используя линейную аппроксимацию и интерпоялиционные многочлены Лагранжа и Ньютона вперед и назад. Было доказано что для таблично-заданной функции значения, полученные при помощи интерполяционных многочленов, совпадают, при этом эти значчения точнее тех, которые получены при помощи линейной аппроксимации.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <vector>
#include <cstdio>
#include <cmath>

using xy = std::pair<double, double>;

double f(double x)
{
	return std::sin((M_PI * x) / 2.0);
}

double lagrange_polynom(double x, const std::vector<xy>& data);
double newton_polynom_forward(double x, const std::vector<xy>& data, double** differences);
double newton_polynom_backward(double x, const std::vector<xy>& data, double** differences);
double approximation(double x, std::pair<double, double> approx_coeffs);

std::pair<double, double> approx_coeffs(const std::vector<xy>& data);
double lagrange_multiplier(double x, const std::vector<xy>& data, int i);
double** calc_finite_differences(const std::vector<xy>& data);
void calc_n_differences(double** differences, int order, int count);

void print_lagrange_polynom(const std::vector<xy>& data);
void print_newton_polynom_forward(const std::vector<xy>& data, double** differences);
void print_newton_polynom_backward(const std::vector<xy>& data, double** differences);
void print_approximation(std::pair<double, double> approx_coeffs);
std::pair<int64_t, int64_t> sc_numbers(double calculated, double exact);

int main()
{
	std::vector<xy> data;
	data.push_back({-1.0, -1.00});
	data.push_back({-0.6, -0.81});
	data.push_back({-0.2, -0.31});
	data.push_back({0.2, 0.31});
	data.push_back({0.6, 0.81});
	data.push_back({1.0, 1.00});

	double** differences = calc_finite_differences(data);
```

```
          std::pair<double,     double>     approximation_coeffs     =
approx_coeffs(data);

          std::vector<double> x({-0.72, 0.13, 0.68});
          double exact;
          double result;
          double abs_inaccuracy;
          double rel_inaccuracy;
          std::pair<int64_t, int64_t> sc;

          for (size_t i = 0; i < x.size(); ++i)
          {
               exact = f(x[i]);
               std::printf("|%-19s|%8s|%12s|%23s|%10s|%10s|%21s|
%12s|\n", "Метод", "X", "Знач в точке", "Вычисленное", "Абс погреш",
"Отн погреш", "Знач цифры", "Верные цифры");


std::printf("-----------------------------------------------------------
-----------------------------------------\n");

               result = lagrange_polynom(x[i], data);
               abs_inaccuracy = std::fabs(exact - result);
               rel_inaccuracy = std::fabs(abs_inaccuracy / exact);
               sc = sc_numbers(result, exact);

               std::printf("|%-21s|%8.4f|%12.8lf|%12.8lf|%10.8lf|
%10.8lf|%12ld|%12ld|\n",
                    "Лагранж",
                    x[i],
                    exact,
                    result,
                    abs_inaccuracy,
                    rel_inaccuracy,
                    sc.first,
                    sc.second
               );

               result     =     newton_polynom_forward(x[i],     data,
differences);
               abs_inaccuracy = std::fabs(exact - result);
               rel_inaccuracy = std::fabs(abs_inaccuracy / exact);
               sc = sc_numbers(result, exact);

               std::printf("|%-21s|%8.4f|%12.8lf|%12.8lf|%10.8lf|
%10.8lf|%12ld|%12ld|\n",
                    "Форвард",
                    x[i],
                    exact,
                    result,
                    abs_inaccuracy,
                    rel_inaccuracy,
                    sc.first,
                    sc.second
               );
```

```
                result    =    newton_polynom_backward(x[i],    data,
differences);
                abs_inaccuracy = std::fabs(exact - result);
                rel_inaccuracy = std::fabs(abs_inaccuracy / exact);
                sc = sc_numbers(result, exact);

                std::printf("|%-16s|%8.4f|%12.8lf|%12.8lf|%10.8lf|
%10.8lf|%12ld|%12ld|\n",
                      "Ньютон обр ход",
                      x[i],
                      exact,
                      result,
                      abs_inaccuracy,
                      rel_inaccuracy,
                      sc.first,
                      sc.second
                );

                result = approximation(x[i], approximation_coeffs);
                abs_inaccuracy = std::fabs(exact - result);
                rel_inaccuracy = std::fabs(abs_inaccuracy / exact);
                sc = sc_numbers(result, exact);

                std::printf("|%-27s|%8.4f|%12.8lf|%12.8lf|%10.8lf|
%10.8lf|%12ld|%12ld|\n",
                      "Аппроксимация",
                      x[i],
                      exact,
                      result,
                      abs_inaccuracy,
                      rel_inaccuracy,
                      sc.first,
                      sc.second
                );


std::printf("----------------------------------------------------------
-----------------------------------------\n");
          }

          /* pt. 6 */
          std::vector<xy> data_br;
          data_br.push_back({-1.0, -1.00});
          data_br.push_back({-0.6, -0.81});
          data_br.push_back({-0.2, -0.31 * 1.5});
          data_br.push_back({0.2, 0.31});
          data_br.push_back({0.6, 0.81});
          data_br.push_back({1.0, 1.00});
          double** differences_br = calc_finite_differences(data_br);
          std::printf("\nКонечная разность без погрешности\n");
          std::printf("|%8s|%8s|%9s|%9s|\n",    "X",    "Y*",    "Δy*",
"Δ^2y*");
          std::printf("-------------------------------------\n");
          for (int i = 0; i < data.size(); ++i)
          {
                if (data.size() - i > 2)
```

```
                    std::printf("|%8.5g|%8.5g|%8.5g|%8.5g|\n",
data.at(i).first, differences[0][i], differences[1][i], differences[2]
[i]);
               else if (data.size() - i == 1)
                    std::printf("|%8.5g|%8.5g|%8s|%8s|\n",
data.at(i).first, differences[0][i], "-", "-");
               else
                    std::printf("|%8.5g|%8.5g|%8.5g|%8s|\n",
data.at(i).first, differences[0][i], differences[1][i], "-");
          }
          std::printf("-------------------------------------\n");
          std::printf("\nКонечная разность с погрешностью\n");
          std::printf("|%8s|%8s|%9s|%9s|\n",    "X",    "Y*",    "Δy*",
"Δ^2y*");
          std::printf("-------------------------------------\n");
          for (int i = 0; i < data_br.size(); ++i)
          {
               if (data_br.size() - i > 2)
                    std::printf("|%8.5g|%8.5g|%8.5g|%8.5g|\n",
data_br.at(i).first,    differences_br[0][i],    differences_br[1][i],
differences_br[2][i]);
               else if (data_br.size() - i == 1)
                    std::printf("|%8.5g|%8.5g|%8s|%8s|\n",
data_br.at(i).first, differences_br[0][i], "-", "-");
               else
                    std::printf("|%8.5g|%8.5g|%8.5g|%8s|\n",
data_br.at(i).first, differences_br[0][i], differences_br[1][i], "-");
          }
          std::printf("-------------------------------------\n");


          std::printf("\n|%-21s|%8s|%12s|%23s|%10s|%10s|%21s|%12s|\n",
"Метод",  "X",  "Знач  в  точке",  "Вычисленное",  "Абс  погреш",  "Отн
погреш", "Знач цифры", "Верные цифры");

std::printf("-------------------------------------------------------
-----------------------------------------\n");
          std::printf("|%-28s|%8s|%12s|%12s|%10s|%10s|%12s|%12s|\n",
"Интерполяция", "", "", "", "", "", "", "");
          for (int i = 0; i < x.size(); ++i)
          {
               exact = f(x[i]);
               result = lagrange_polynom(x[i], data);
               abs_inaccuracy = std::fabs(exact - result);
               rel_inaccuracy = std::fabs(abs_inaccuracy / exact);
               sc = sc_numbers(result, exact);

               std::printf("|%-16s|%8.4f|%12.8lf|%12.8lf|%10.8lf|
%10.8lf|%12ld|%12ld|\n",
                    "",
                    x[i],
                    exact,
                    result,
                    abs_inaccuracy,
                    rel_inaccuracy,
                    sc.first,
```

```
                    sc.second
               );
          }

std::printf("-----------------------------------------------------------
-----------------------------------------\n");

          std::printf("\n|%-21s|%8s|%12s|%23s|%10s|%10s|%21s|%12s|\n",
"Метод", "X", "Знач в точке", "Вычисленное", "Абс погреш", "Отн
погреш", "Знач цифры", "Верные цифры");

std::printf("-----------------------------------------------------------
-----------------------------------------\n");
          std::printf("|%-29s|%8s|%12s|%12s|%10s|%10s|%12s|%12s|\n",
"Аппроксимация", "", "", "", "", "", "", "");
          for (int i = 0; i < x.size(); ++i)
          {
               exact = f(x[i]);
               result = approximation(x[i], approximation_coeffs);
               abs_inaccuracy = std::fabs(exact - result);
               rel_inaccuracy = std::fabs(abs_inaccuracy / exact);
               sc = sc_numbers(result, exact);

               std::printf("|%-16s|%8.4f|%12.8lf|%12.8lf|%10.8lf|
%10.8lf|%12ld|%12ld|\n",
                    "",
                    x[i],
                    exact,
                    result,
                    abs_inaccuracy,
                    rel_inaccuracy,
                    sc.first,
                    sc.second
               );
          }

std::printf("-----------------------------------------------------------
-----------------------------------------\n");

          std::cout << std::endl;
          print_lagrange_polynom(data);
          print_newton_polynom_forward(data, differences);
          print_newton_polynom_backward(data, differences);
          print_approximation(approximation_coeffs);

          for (int i = 0; i < data.size(); ++i)
          {
               delete [] differences_br[i];
               delete [] differences[i];
          }
          delete [] differences_br;
          delete [] differences;

          return 0;
     }
```

```
	double lagrange_polynom(double x, const std::vector<xy>& data)
	{
		double result = 0;
		for (int i = 0; i < data.size(); ++i)
			result += data[i].second * lagrange_multiplier(x,
data, i);
		return result;
	}

	double lagrange_multiplier(double x, const std::vector<xy>& data,
int i)
	{
		double multiplier = 1;
		for (int k = 0; k < data.size(); ++k)
		{
			if (k == i)
				continue;
			multiplier *= (x - data[k].first) / (data[i].first -
data[k].first);
		}
		return multiplier;
	}

	double** calc_finite_differences(const std::vector<xy>& data)
	{
		double** differences = new double*[data.size()];
		for (int i = 0; i < data.size(); ++i)
			differences[i] = new double[data.size()];

		for (int i = 0; i < data.size(); ++i)
			differences[0][i] = data[i].second;

		for (int i = 1; i < data.size(); ++i)
			calc_n_differences(differences, i, data.size() - i);

		return differences;
	}

	void calc_n_differences(double** differences, int order, int
count)
	{
		for (int i = 0; i < count; ++i)
			differences[order][i] = differences[order - 1][i + 1]
- differences[order - 1][i];
	}

	double newton_polynom_forward(double x, const std::vector<xy>&
data, double** differences)
	{
		double result = data.at(0).second;
		double denom = 1;
		double multiplier = 1;
		double h = data.at(1).first - data.at(0).first;
		for (int i = 1; i < data.size(); ++i)
		{
			denom *= i * h;
```

```
            multiplier *= x - data.at(i - 1).first;
            result += differences[i][0] * multiplier / denom;
        }
        return result;
    }

    double newton_polynom_backward(double x, const std::vector<xy>&
data, double** differences)
    {
        double result = data.at(data.size()-1).second;

        double denom = 1;
        double multiplier = 1;
        double h = data.at(1).first - data.at(0).first;
        for (int i = data.size() - 2; i >= 0; --i)
        {
            denom *= (data.size() - i - 1) * h;
            multiplier *= x - data.at(i + 1).first;
            result += differences[data.size() - i - 1][i] *
multiplier / denom;
        }
        return result;
    }

    std::pair<double, double> approx_coeffs(const std::vector<xy>&
data)
    {
        double x_sum = 0, x2_sum = 0, y_sum = 0, xy_sum = 0;
        int n = data.size();
        for (const xy& element: data)
        {
            x_sum += element.first;
            y_sum += element.second;
            xy_sum += element.first * element.second;
            x2_sum += element.first * element.first;
        }
        double a = (n * xy_sum - x_sum * y_sum) / (n * x2_sum -
x_sum * x_sum);
        double b = (y_sum - a * x_sum) / n;
        return {a, b};
    }

    double approximation(double x, std::pair<double, double>
approx_coeffs)
    {
        return approx_coeffs.first * x + approx_coeffs.second;
    }

    std::pair<int64_t, int64_t> sc_numbers(double calculated, double
exact)
    {
        int64_t significant, correct;
        int64_t signs = 8, correct_cnt = 0;
        double inaccuracy = std::fabs(exact - calculated);
        int64_t nums_cnt;
        for (int64_t i = 0; i < signs; ++i)
```

```
            {
                  if (correct_cnt == 0 && inaccuracy < 1)
                        inaccuracy *= 10;
                  else if (correct_cnt == 0)
                        correct_cnt = i;
                  calculated *= 10;
                  exact *= 10;
            }
            significant = static_cast<int64_t>(std::fabs(calculated));
            correct = static_cast<int64_t>(std::fabs(exact));
            nums_cnt = std::to_string(significant).size();
            correct = significant / std::pow(10, 1 + nums_cnt -
correct_cnt);
            return {significant, correct};
      }


      void print_lagrange_polynom(const std::vector<xy>& data)
      {
            double result = 0;
            std::cout << "Полином Лагранжа" << std::endl;
            std::cout << "f(x) = 0";
            for (int i = 0; i < data.size(); ++i)
            {
                  std::cout << " + (" << data[i].second << ")" << "";
                  double denom = 1;
                  for (int k = 0; k < data.size(); ++k)
                  {
                        if (k == i)
                              continue;
                        denom *= (data[i].first - data[k].first);
                  }
                  std::cout << "(" << denom << ")";
                  for (int k = 0; k < data.size(); ++k)
                  {
                        if (k == i)
                              continue;
                        std::cout << "(x - " << data[k].first << ")";
                        denom *= (data[i].first - data[k].first);
                  }
                  std::cout << std::endl;
            }
            std::cout << std::endl;
      }

      void print_newton_polynom_forward(const std::vector<xy>& data,
double** differences)
      {
            double denom = 1;
            double h = data.at(1).first - data.at(0).first;
            std::cout << "Полином Ньютона прямой ход" << std::endl;
            std::cout << data.at(0).second;
            for (int i = 1; i < data.size(); ++i)
            {
                  std::cout << " + ";
                  denom *= i * h;
```

```
                std::cout << "(" << differences[i][0] << "/" << denom
<< ")";
                for (int k = 1; k <= i; ++k)
                     std::cout << "(x - " << data.at(k - 1).first <<
")";
                std::cout << std::endl;
           }
           std::cout << std::endl;
     }

     void  print_newton_polynom_backward(const  std::vector<xy>&  data,
double** differences)
     {
           double denom = 1;
           double h = data.at(1).first - data.at(0).first;
           std::cout << "Полином Ньютона обратный ход" << std::endl;
           std::cout << data.at(data.size()-1).second;
           for (int i = data.size() - 2; i >= 0; --i)
           {
                std::cout << " + ";
                denom *= (data.size() - i - 1) * h;
                std::cout << "(" << differences[i][0] << "/" << denom
<< ")";
                for (int k = data.size() - 2; k >= i; --k)
                     std::cout << "(x - " << data.at(k + 1).first <<
")";
                std::cout << std::endl;
           }
           std::cout << std::endl;
     }

     void print_approximation(std::pair<double, double> approx_coeffs)
     {
           std::cout << "Аппроксимация" << std::endl;
           std::cout   <<   approx_coeffs.first   <<   "x   +   "   <<
approx_coeffs.second << std::endl;
           std::cout << std::endl;
     }
```