# 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

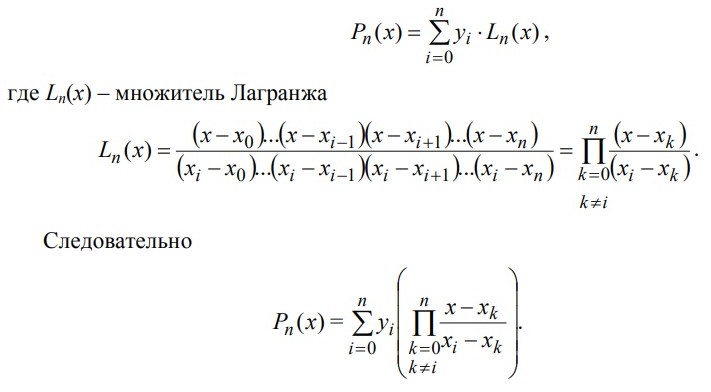
**Цель работы:** исследование методов интерполяции и аппроксимации для равноотстоящих узлов с последующей реализацией на одном из языков программирования

# Основные теоретические положения.

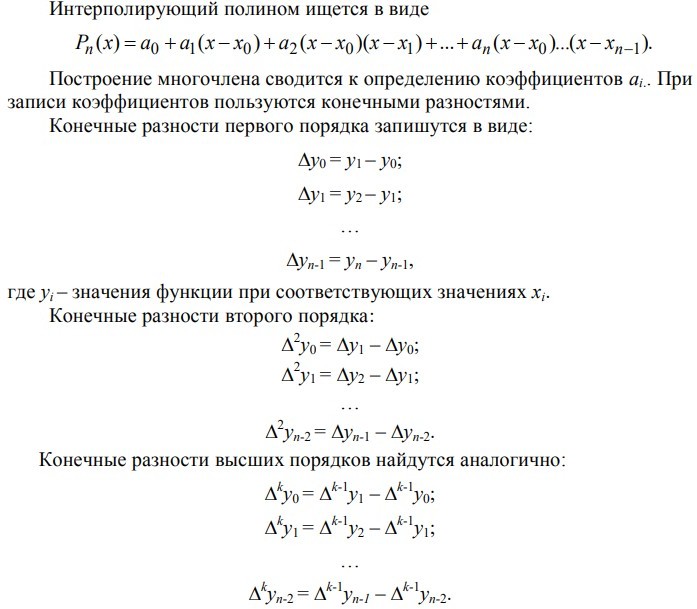
Значения функции *f* ( *xi* )= *yi* заданы в точках *xi*=*x*0 +*nh i* ∈{0 *,…, n*} *,*

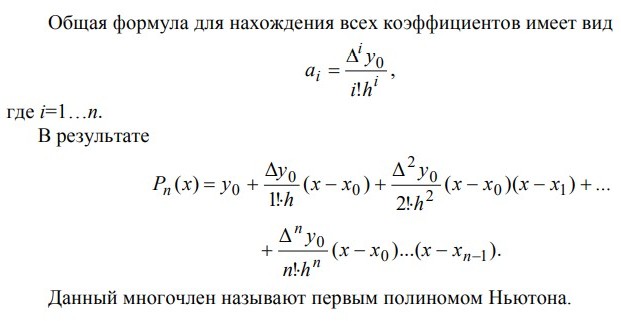
необходимо найти промежуточные значения.

Интерполяционный многочлен Лагранжа.

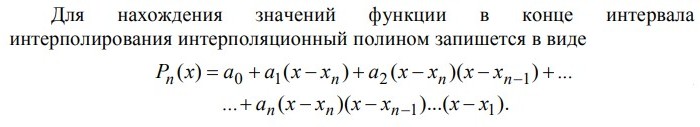


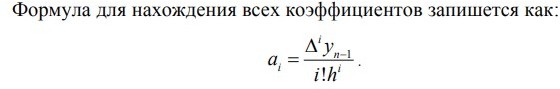
Первый интерполяционный многочлен Ньютона. Точка интерполирования находится в начале таблицы.

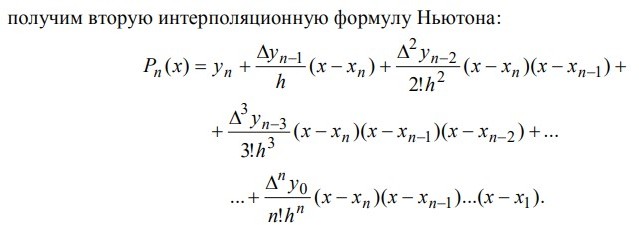




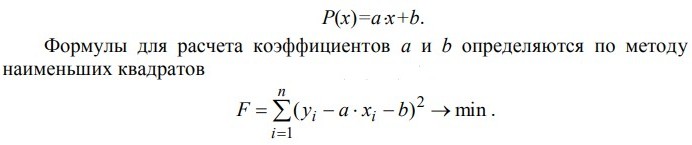
Второй интерполяционный многочлен Ньютона. Точка интерполирования находится в конце таблицы.

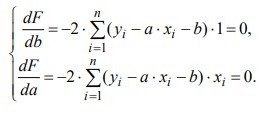


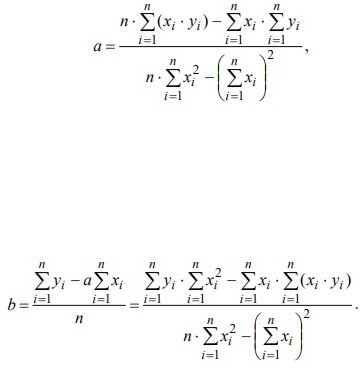




Аппроксимация функции. Необходимо найти эмпирическую формулу, значения которой при *x*=*xi* мало бы отличались от входных данных. Будет использоваться линейная аппроксимация, при которой данные описываются линейной зависимостью







# 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

1. Используя многочлен Лагранжа, найти приближенные значения функции. Подсчитать точные значения, абсолютную погрешность, относительную погрешность, верные и значащие цифры.
2. 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.
3. Составить таблицу конечных разностей до второго порядка. К одному из значений *yi* прибавить погрешность (*α* ), превосходящую допустимую. Показать, как при этом изменятся конечные разности большего порядка.
4. При аппроксимировании функции использовать для нахождения новых значений те же три точки. Рассчитать все, что указано в пункте 4.
5. Составить сводную таблицу по результатам исследования.

# Выполнение работы.

Функция: *y*=sin ( 𝑢∗*x* )

2

|  |  |
| --- | --- |
| **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 PN(x)=c0+c1x+...+cN-1xN- 1+cNxN, где ci — искомые коэффициенты. При помощи таблицы с данными получим СЛУ вида:

c0+c1x0+...+c x N=PN(x0)=y0

N 0

…

c0+c1xN+...+c x N=PN(xN)=yN

N N

или в виде матриц и векторов:

1 *x x*2 ... *xN*

)

*A*=(... 0

... ...

1 *xN x*2 ... *xN*

0

## ...

*N*

0

*, c*=(*c*

...

0

*N*

0

*N*

*N*

*,*... *, c*

)*T , y* =( *y*

*,* ... *, y* )*T*

Система однозначно разрешина в случае если определитель матрицы A не равен нулю. Определитель матрицы А является определителем

Вандермонда: *det* ( *A*)= ∏

0≤ *j*≤*i*≤*N* ∈*?*

( *xi*−*xj*) , значит он не равен нулю если xi

не равно xj при любых неодинаковых j и i. Следовательно коэффициенты ci находятся однозначно т. е. интерполяционный многочлен существует и при этом только один.

1. 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.

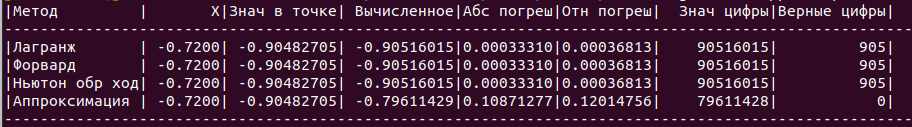


Рисунок 1 — Значения функции в точке -0,72

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.

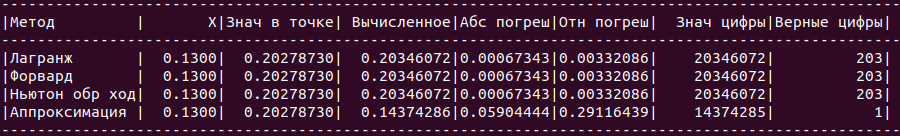
1. Аналогично при помощи многочлена Лагранжа, многочленов Ньютона вперед и назад, линейной аппроксимации найдем приближенное значение функции в первой точке 0,13. Результаты работы программы с выводом точного значения функции в точке, абсолютной и относительнйо погрешностей, значащих и верных цифр представлены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

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.

1. 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.

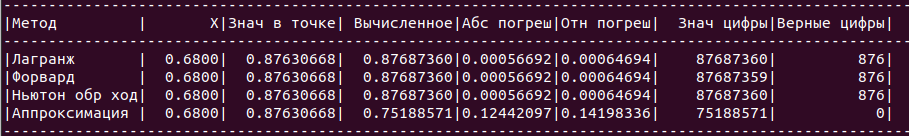


Рисунок 3 — Значения функции в точке 0,68

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.

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

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 yi, на такое же значение будут в изменяться конечные значения delta yi и delta

yi-1, а значит будут изменяться и конечные разности порядков выше, следовательно, чем выше порядок, тем сильнее влияние внисенной погрешности. В таблице 1 представленны возмущения конечных разностей.

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

|  |  |  |  |
| --- | --- | --- | --- |
| X | Y | delta y\* | delta2 y\* |
| x0 | y0 | y1-y0 | y2+alpha-2y1+y0 |
| x1 | y1 | y2+alpha-y1 | y2-2y2-2alpha+y1 |
| x2 | y2 + alpha | y3-y2-alpha | y4-2y3+y2+alpha |
| x3 | y3 | y4-y3 | y5-2y4+y3 |
| x4 | y4 | y5-y4 | - |
| x5 | y5 | - | - |

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

1. 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.

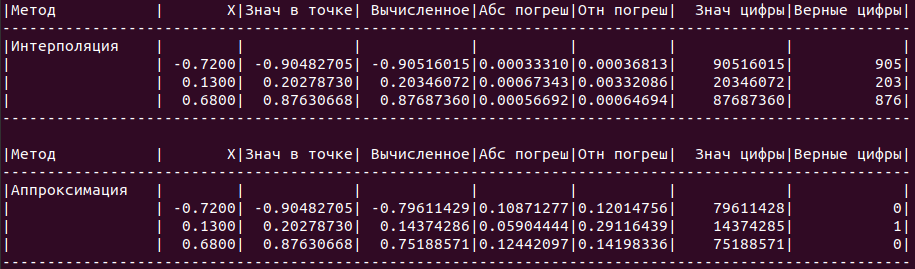


Рисунок 5 — Сравнение интерполяции и аппроксимации.

По рисунку видно, что интерполяция многочленом для всех Х 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.

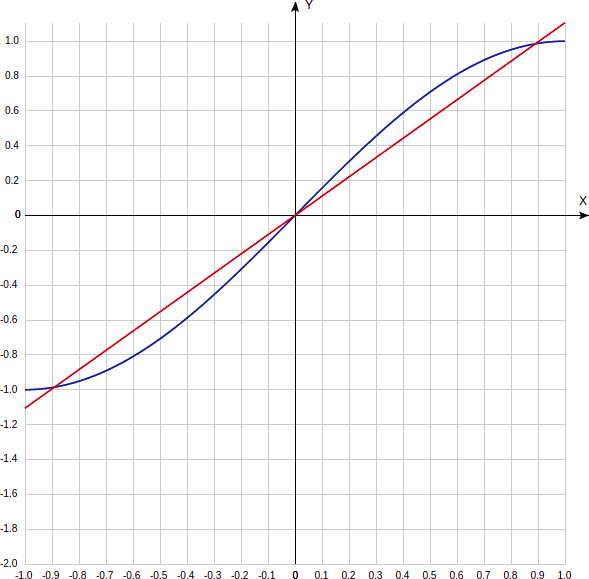


Рисунок 6 — Графики интерполяции (синий), аппроксимации (красный), функции (зеленый).

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

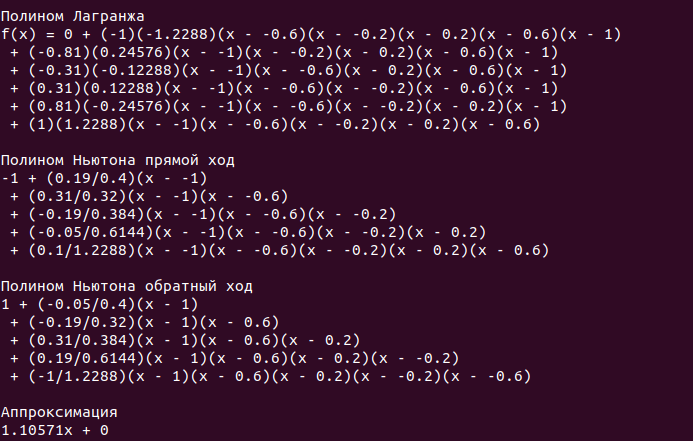


Рисунок 7 — Формулы интерполяционных полиномов, линейной

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

differences);

);

result = newton\_polynom\_forward(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",

"Форвард", x[i], exact, result,

abs\_inaccuracy, rel\_inaccuracy, sc.first, sc.second

);

differences);

result = newton\_polynom\_backward(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",

"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)

{

data, i);

}

double result = 0;

for (int i = 0; i < data.size(); ++i)

result += data[i].second \* lagrange\_multiplier(x, 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)

{

data[k].first);

}

if (k == i)

continue;

multiplier \*= (x - data[k].first) / (data[i].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;

data)

}

std::pair<double, double> approx\_coeffs(const std::vector<xy>&

{

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;

}