Лабораторная работа №4

Тужилкина Н.Г., БПМ-18-2, вариант 24.

# Интерполяционные многочлены Лагранжа и Ньютона

Пусть на множестве задана сетка , определяемая точкой , а на сетке задана сеточная функция

Сеточная функция может задаваться совокупностью пар: .

Требуется найти функцию , принимающую в точках те же значения, что и функция , т. е.

Точки ‒ узлы интерполяции, а искомая функция ‒ интерполирующая.

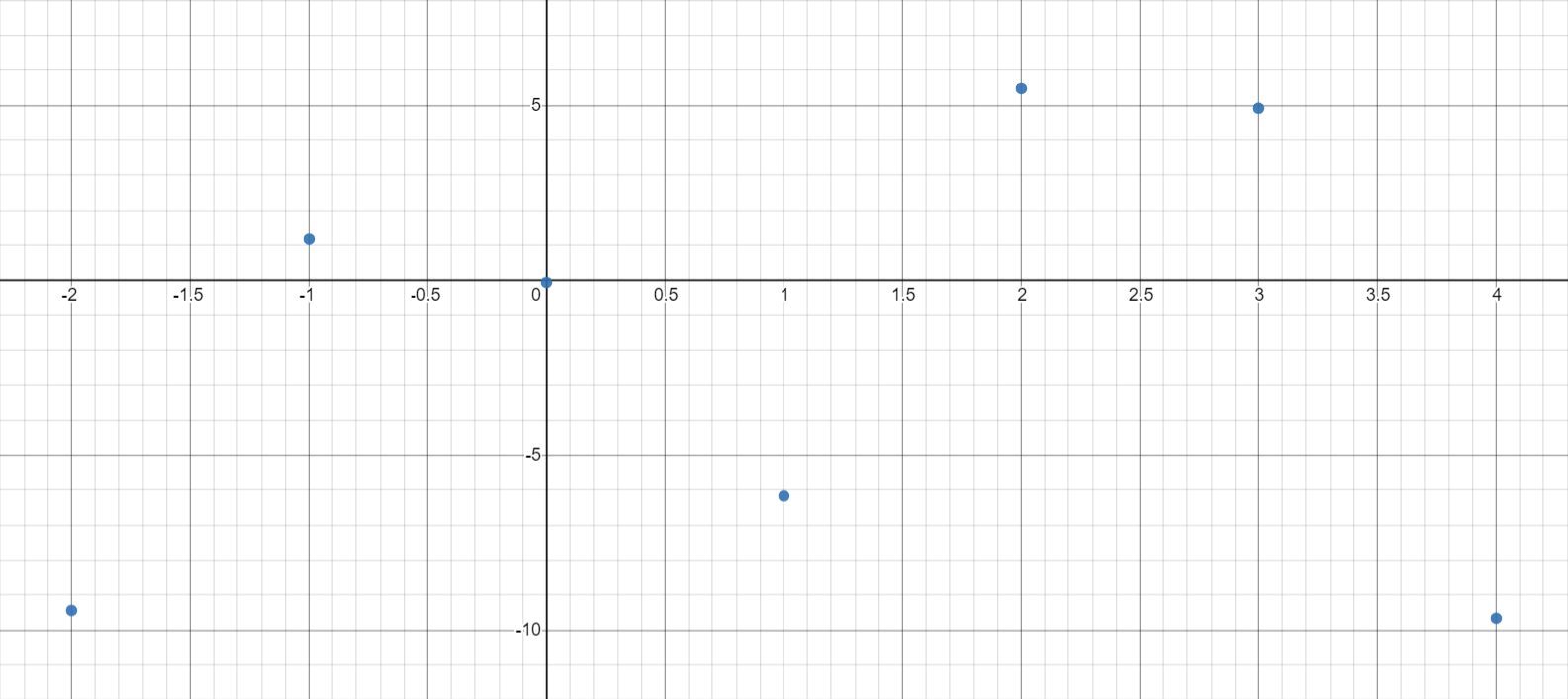
В такой общей постановке задача может иметь бесконечное множество решений.

Эта задача становится однозначной, если вместо произвольной функции искать полином степени не выше , удовлетворяющий условиям (2).

Решением этой задачи являются многочлен Лагранжа и многочлен Ньютона, различие между которыми состоит в форме их записи.

Требуется построить для заданных точек интерполяционные многочлены Лагранжа и Ньютона и найти их значения в точках и .

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | -2 | -1 | 0 | 1 | 2 | 3 | 4 |
|  | -9.44 | 1.77 | -0.06 | -6.17 | 5.48 | 4.92 | -9.66 |



## Цель работы:

1. Изучить методы функциональной интерполяции:
   1. Многочлен Лагранжа
   2. Многочлен Ньютона
2. Написать и отладить программу построения интерполяционных многочленов Лагранжа и Ньютона.
3. Найти значения многочленов в точках и .

## Многочлен Лагранжа

### Изложение метода

Интерполяционный многочлен Лагранжа записывается в виде

где ‒ базисные многочлены Лагранжа:

### Текст программы

void Lagrange(const vector<point>& data) {

size\_t n(data.size() - 1);

cout << "Сеточная функция:\n";

for (size\_t i(0); i <= n; ++i) {

cout << data[i];

(i != n) ? (cout << ", ") : (cout << "\n\n\n");

}

vector<double> denum;

cout << "Интерполяционный многочлен Лагранжа:\nL\_" << n + 1 << "(x) = ";

for (size\_t j(0); j <= n; ++j) {

denum.push\_back(1);

for (size\_t i(0); i <= n; ++i) {

if (j == i) continue;

if (data[i].x > 0) cout << "(x - " << data[i].x << ") \* ";

else if (data[i].x == 0) cout << "x \* ";

else cout << "(x + " << abs(data[i].x) << ") \* ";

denum.back() \*= data[i].x - data[j].x;

}

(data[j].y > 0) ?

(cout << data[j].y) :

(cout << "(-" << abs(data[j].y) << ")");

(denum.back() > 0) ?

(cout << " / " << denum.back()) :

(cout << " / (" << denum.back() << ")");

if (j != n) cout << " + ";

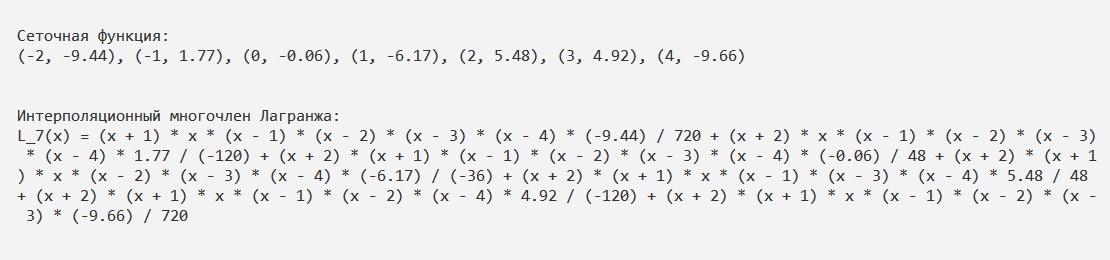
}

cout << "\n";

}

### Представление результатов

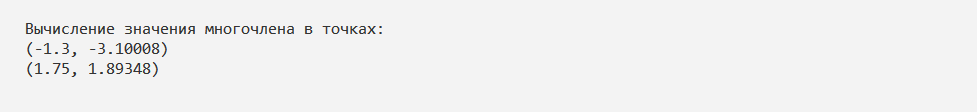
Результат работы программы:

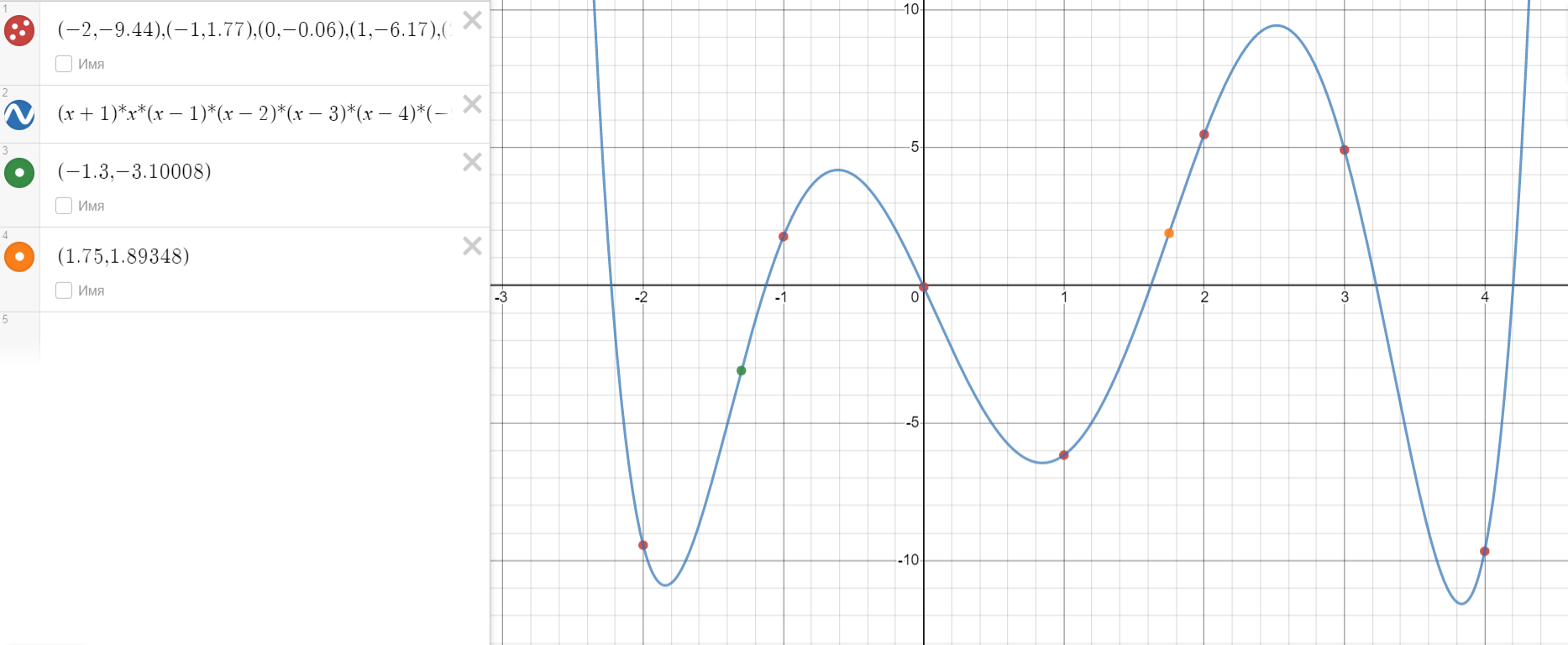


Сеточная функция:

Интерполяционный многочлен Лагранжа:

Подсчет значений многочлена в точках и :





## Многочлен Ньютона

### Изложение метода

Пусть функция  задана с произвольным шагом, и точки таблицы значений пронумерованы в произвольном порядке.

Разделенные разности нулевого порядка совпадают со значениями функции в узлах. Разделенные разности первого порядка определяются через разделенные разности нулевого порядка:

Разделенные разности второго порядка определяются через разделенные разности первого порядка:

Разделенные разности -го порядка определяются через разделенные разности порядка :

Используя понятие разделенной разности интерполяционный многочлен Ньютона можно записать в следующем виде:

### Текст программы

void Newton(const vector<point>& data) {

size\_t n(data.size() - 1);

cout << "Интерполяционный многочлен Ньютона:\nP\_" << n + 1 << "(x) = ";

cout << data[0].y << " + ";

vector<double> delta;

// Разделенные разности 0-го порядка совпадают со значениями функции в узлах

for (const auto& it : data)

delta.push\_back(it.y);

for (int m(1); m <= n; ++m) {

// Разделенные разности k-го порядка определяются через разделенные разности порядка k-1

vector<double> delta\_new;

for (int i(0); i <= n - m; ++i) {

delta\_new.push\_back((delta[i + 1] - delta[i]) / (data[i + m].x - data[i].x));

}

delta = delta\_new;

(delta[0] > 0) ?

(cout << delta[0] << " \* ") :

(cout << "(-" << abs(delta[0]) << ") \* ");

for (int i(0); i <= m - 1; ++i) {

if (data[i].x > 0) cout << "(x - " << data[i].x << ")";

else if (data[i].x == 0) cout << "x";

else cout << "(x + " << abs(data[i].x) << ")";

if (i != m - 1) cout << " \* ";

}

if (m != n) cout << " + ";

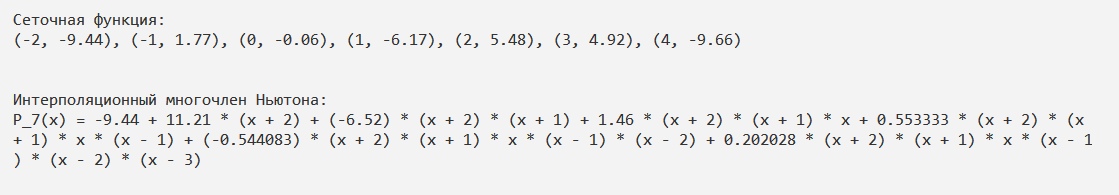
}

cout << "\n";

}

### Представление результатов

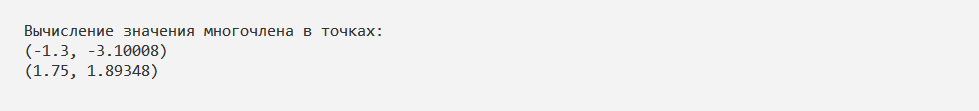
Результат работы программы:



Сеточная функция:

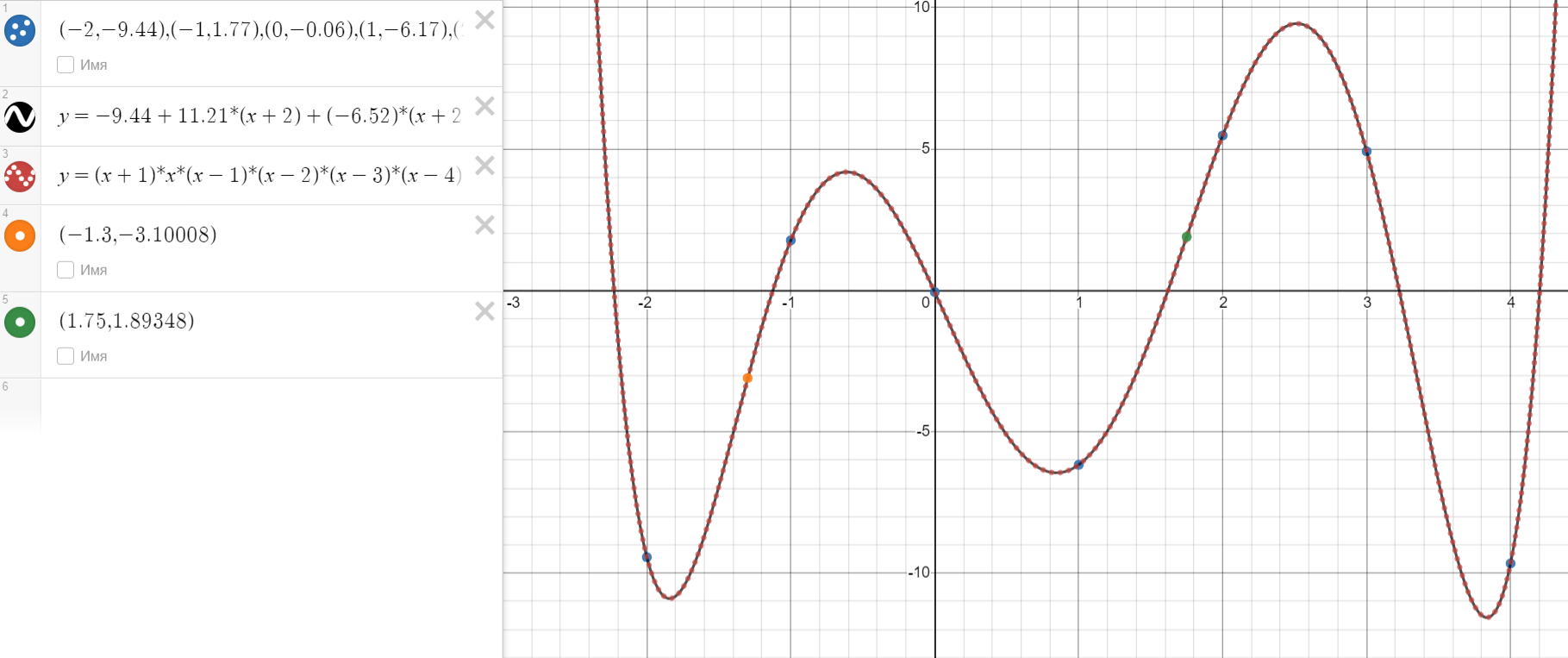
Интерполяционный многочлен Лагранжа:

Подсчет значений многочлена в точках и :



## Вывод

При введении дополнительных узлов интерполяции все коэффициенты многочлена Лагранжа необходимо пересчитывать заново. От этого недостатка свободны многочлены Ньютона. Степень многочленов Ньютона можно последовательно повышать путем добавления очередных слагаемых, имеющих более высокую степень.



Значения многочленов в точках и совпадают: и соответственно.