

Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»
Кафедра информатики

Отчет по лабораторной работе №6
Интерполяционные многочлены

Выполнил:
студент группы 953501
Кременевский В.С

Руководитель:
доцент
Анисимов В.Я.

Минск 2021

Содержание

- 1. Цель работы**
- 2. Теоретические сведения**
- 3. Тестовые примеры**
- 4. Решение задания**
- 5. Выводы**
- 6. Программная реализация**

1.

Цель работы

- 1) Изучить интерполяцию функций с помощью интерполяционных многочленов Лагранжа и Ньютона
- 2) Реализовать программно интерполяцию Лагранжа и Ньютона
- 3) Проверить работу интерполяции на тестовых примерах
- 4) Выполнить задание по варианту
- 5) Оценить погрешность

2.

Теоретические сведения

Из математического анализа известно, что в окрестности точки x_0 любую n раз непрерывно дифференцируемую функцию можно аппроксимировать (приблизить) ее многочленом Тейлора:

$$P_n(x) = \sum_{k=0}^n \frac{f^{(k)}(x_0)(x - x_0)^k}{k!},$$

причем

$$\begin{aligned} f(x_0) &= P_n(x_0), \\ f'(x_0) &= P'_n(x_0), \\ \dots &\dots \\ f^{(n)}(x_0) &= P_n^{(n)}(x_0). \end{aligned}$$

Очевидно, такая аппроксимация во многих отношениях является очень хорошей, но она имеет локальный характер, т.е. хорошо аппроксимирует функцию только вблизи точки x_0 . Это главный недостаток аппроксимации с помощью многочлена Тейлора.

Если речь идет об аппроксимации функции на отрезке, применяются другие методы.

Возможен и другой подход, когда в качестве аппроксимирующей функции берут многочлен или другую достаточно простую функцию, значения которых совпадают со значениями исходной функции в заданных заранее точках, так называемых узлах. Такого рода приближение функций имеет свое собственное название - интерполяция.

7.1. Интерполяционный многочлен

Пусть $f(x)$ – функция, непрерывная на отрезке $[a, b]$.

Выберем на этом отрезке точки, называемые *узлами интерполяции*:

Ставится задача найти многочлен $P_n(x)$ такой, что

$$P_n(x_k) = y_k, \quad \forall k = 0, 1, \dots, n. \quad (7.1)$$

Такой многочлен $P_n(x)$ называется *интерполяционным многочленом*, а задача его нахождения – *задачей интерполяции*.

Покажем, что задача интерполяции имеет решение, причем единственное.

$$\text{Пусть } P_n(x) = \sum_{k=0}^n a_k x^{n-k}.$$

Тогда для определения коэффициентов многочлена из условия (7.1) получаем систему:

$$\begin{cases} a_0 x_0^n + a_1 x_0^{n-1} + \dots + a_n = y_0 \\ a_0 x_1^n + a_1 x_1^{n-1} + \dots + a_n = y_1 \\ \dots \\ a_0 x_n^n + a_1 x_n^{n-1} + \dots + a_n = y_n. \end{cases}$$

Ее определитель Δ с точностью до знака совпадает с так называемым определителем Вандермонда.

$$W(x_0, \dots, x_n) = \begin{vmatrix} 1 & 1 & \dots & 1 \\ x_0 & x_1 & \dots & x_n \\ x_0^2 & x_1^2 & \dots & x_n^2 \\ \dots & \dots & \dots & \dots \\ x_0^n & x_1^n & \dots & x_n^n \end{vmatrix} = \prod_{i < j} (x_j - x_i) \neq 0.$$

Поскольку все x_i различны, определитель Δ отличен от нуля, и, следовательно, система имеет единственное решение. Отсюда вытекает существование и единственность интерполяционного многочлена.

Погрешность интерполяции.

Обозначим

$$R_n(x) = f(x) - P_n(x) \text{ и будем искать ее оценку.}$$

Пусть $f(x) \in C^{n+1}[a, b]$. Положим $R_n(x) = \omega(x)r(x)$,

где $\omega(x) = (x - x_0)(x - x_1) \cdot \dots \cdot (x - x_n)$.

Зафиксируем произвольную точку x , отличную от узлов интерполяции x_i , $i = 0, n$, и построим вспомогательную функцию:

$$F(t) = P_n(t) + \omega(t)r(x) - f(t), \quad a \leq t \leq b. \quad (7.2)$$

Очевидно, $F(x) = 0$ и, кроме того $F(x_k) = 0$, $k = \overline{0, n}$.

Таким образом, функция $F(t)$ имеет по крайней мере $(n+2)$ нуля на отрезке $[a, b]$. Применим теорему Ролля, по которой между каждой парой нулей функции находится по крайней мере один нуль производной этой функции.

Тогда производная $F'(t)$ имеет по крайней мере $(n+1)$ нулей на данном интервале (a, b) . Продолжая рассуждение, получим в итоге, что $F^{(n)}(t)$ имеет, по крайней мере, два нуля, а $F^{(n+1)}(t)$ – один нуль в некоторой точке ξ на (a, b) .

Продифференцируем равенство (7.2) $(n+1)$ раз и подставим $t = \xi$.
Получим

$$F^{(n+1)}(\xi) = (n+1)! \cdot r(x) - f^{(n+1)}(\xi) = 0.$$

$$\text{Откуда } r(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}.$$

Тогда

$$R_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \omega(x),$$

где $\xi \in [a, b]$ (очевидно формула напоминает остаток формулы Тейлора в форме Лагранжа). В итоге имеем оценку погрешности интерполяции:

$$|R_n(x)| \leq \frac{M_{n+1}}{(n+1)!} |\omega(x)|, \quad \text{где } M_{n+1} = \max_{a \leq x \leq b} |f^{(n+1)}(x)|.$$

Интерполяционный многочлен Лагранжа

Пусть даны узлы на отрезке $[a, b]$, $a \leq x_0 < x_1 < \dots < x_n \leq b$, и значения функции $F(x)$ в узлах

$$f(x_i) = y_i, \quad i = \overline{0, n}.$$

$$\text{Пусть } \omega(x) = (x - x_0)(x - x_1) \cdots (x - x_n),$$

$$\omega_j(x) = (x - x_0) \cdots (x - x_{j-1})(x - x_{j+1}) \cdots (x - x_n),$$

$$\text{т. е. } \omega_j(x) = \frac{\omega(x)}{x - x_j}.$$

$$\text{Положим } l_j(x) = \frac{\omega_j(x)}{\omega_j(x_j)},$$

$$\text{т. е. } l_j(x) = \frac{(x - x_0) \cdots (x - x_{j-1})(x - x_{j+1}) \cdots (x - x_n)}{(x_j - x_0) \cdots (x_j - x_{j-1})(x_j - x_{j+1}) \cdots (x_j - x_n)}.$$

$$\text{Очевидно } l_j(x_i) = \begin{cases} 0, & \text{при } i \neq j \\ 1, & \text{при } i = j. \end{cases}$$

$$\text{Построим многочлен } L_n(x) = \sum_{j=0}^n l_j(x) y_j.$$

Легко видеть, что $L_n(x_i) = l_i(x_i) y_i = 1 \cdot y_i = y_i$, $i = \overline{0, n}$, т.е. это интерполяционный многочлен. Его называют интерполяционным многочленом Лагранжа.

Интерполяционный многочлен Ньютона

Пусть x_0, x_1, \dots, x_n – набор узлов интерполяции,

y_0, y_1, \dots, y_n – значения функции $f(x)$ в узлах.

Величину $\Delta y_k = y_{k+1} - y_k$ называют конечной разностью первого порядка в k -м узле.

Аналогично определяются конечные разности высших порядков.

$$\Delta^2 y_k = \Delta y_{k+1} - \Delta y_k = y_{k+2} - y_{k+1} - (y_{k+1} - y_k) = y_{k+2} - 2y_{k+1} + y_k$$

$$\Delta^i y_k = \Delta^{i-1} y_{k+1} - \Delta^{i-1} y_k = \sum_{i=0}^n (-1)^{n-i} C_n^i y_{k+i} \quad \Delta^i y_k = \Delta^{i-1} y_{k+1} - \Delta^{i-1} y_k = \sum_{i=0}^n (-1)^{n-i} C_n^i y_{k+i} .$$

Конечные разности обычно считают по схеме:

x_i	y_i	Δy_i	$\Delta^2 y_i$	$\Delta^3 y_i$
x_0	y_0	$\Delta y_0 = y_1 - y_0$		
x_1	y_1	$\Delta y_1 = y_2 - y_1$	$\Delta^2 y_0 = \Delta y_1 - \Delta y_0$	
x_2	y_2	$\Delta y_2 = y_3 - y_2$	$\Delta^2 y_1 = \Delta y_2 - \Delta y_1$	
x_3	y_3			$\Delta^3 y_0 = \Delta^2 y_1 - \Delta^2 y_0$

Разделенной разностью первого порядка называется выражение

$$f_1(x_k, x_{k+1}) = \frac{y_{k+1} - y_k}{x_{k+1} - x_k} = \frac{\Delta y_k}{\Delta x_k} .$$

Разделенной разностью второго порядка называется выражение

$$f_2(x_k, x_{k+1}, x_{k+2}) = \frac{f_1(x_{k+1}, x_{k+2}) - f_1(x_k, x_{k+1})}{x_{k+2} - x_k} \text{ и т. д.}$$

Пусть x – любая точка отрезка, не совпадающая с узлами. Тогда

$$f_1(x, x_0) = \frac{y_0 - f(x)}{x_0 - x},$$

откуда $f(x) = y_0 + f_1(x, x_0)(x - x_0)$. (7.3)

$$\text{Далее } f_2(x, x_0, x_1) = \frac{f_1(x_0, x_1) - f_1(x, x_0)}{x_1 - x},$$

откуда $f_1(x, x_0) = f_1(x_0, x_1) + f_2(x, x_0, x_1)(x - x_1)$.

Подставляя в (7.3), получаем

$$f(x) = y_0 + f_1(x_0, x_1)(x - x_0) + f_2(x, x_0, x_1)(x - x_0)(x - x_1). \quad (7.4)$$

$$\text{Далее } f_3(x, x_0, x_1, x_2) = \frac{f_2(x_0, x_1, x_2) - f_2(x, x_0, x_1)}{x_2 - x},$$

откуда $f_2(x, x_0, x_1) = f_2(x_0, x_1, x_2) + f_3(x, x_0, x_1, x_2)(x - x_2)$.

Подставляя в (4), имеем:

$$f(x) = y_0 + f_1(x_0, x_1)(x - x_0) + f_2(x, x_0, x_1)(x - x_0)(x - x_1) + \\ + f_3(x, x_0, x_1, x_2)(x - x_0)(x - x_1)(x - x_2). \quad (7.5)$$

Продолжая процесс, получим:

$$f(x) = N_n(x) + f_{n+1}(x, x_0, \dots, x_n)(x - x_0) \dots (x - x_n),$$

где $N_n(x) = y_0 + f_1(x_0, x_1)(x - x_0) + \dots + f_n(x_0, \dots, x_n)(x - x_0) \dots (x - x_{n-1})$.

$$\text{Очевидно, при } x = x_i, \quad \forall i = \overline{0, n}, \quad f(x_i) = N_n(x_i), \quad i = \overline{0, n},$$

т. е. $N_n(x)$ – интерполяционный многочлен. Его называют интерполяционным многочленом Ньютона.

Достоинство интерполяционного многочлена Ньютона: он удобен при расширении интерполяции и добавлении узлов.

Недостаток: в какой-то степени он сложнее в подсчете конечных разностей по сравнению с многочленом Лагранжа.

Интерполяционный многочлен Ньютона - Грегори

Рассмотрим случай задачи интерполяции с равноотстоящими узлами, т. е. пусть

$$h = x_{i+1} - x_i, \text{ для всех } i = \overline{0, n}.$$

Будем искать интерполяционный многочлен Ньютона в форме

$$N(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \dots + a_n(x - x_0) \cdots (x - x_{n-1}),$$

где коэффициенты многочлена не определены.

Используем условие

$$N(x_i) = y_i, \quad i = \overline{0, n}.$$

Получим:

$$N(x_0) = y_0 = a_0$$

$$N(x_1) = y_1 = a_0 + a_1 h$$

$$N(x_2) = y_2 = a_0 + 2ha_1 + 2h^2a_2$$

..... .

Откуда $a_0 = y_0$,

$$a_1 = \frac{y_1 - y_0}{h} = \frac{\Delta y_0}{h},$$

$$y_2 = y_0 + 2h \frac{y_1 - y_0}{h} + 2h^2 a_2,$$

$$a_2 = \frac{y_2 - y_0 - 2(y_1 - y_0)}{2h^2} = \frac{y_2 - 2y_1 + y_0}{2h^2} = \frac{\Delta^2 y_0}{2h^2}.$$

Продолжая, можем по индукции получить формулу

$$a_k = \frac{\Delta^k y_0}{k! h^k}, \quad k = 1, \dots, n.$$

В итоге получаем интерполяционный многочлен Ньютона - Грегори:

$$N(x) = y_0 + \frac{\Delta y_0}{h} (x - x_0) + \frac{\Delta^2 y_0}{2! h^2} (x - x_0)(x - x_1) + \dots + \frac{\Delta^n y_0}{n! h^n} (x - x_0)(x - x_1) \dots (x - x_{n-1}).$$

3.

Тестовые примеры

Тестовый пример 1. Построить интерполяционные многочлены в форме Лагранжа и Ньютона. Найти значение в точке $x = 3$

```
x_1 = [2, 4, 5]
y_1 = [1, 15, 28]
point_1 = 3
```

```
Data:
x = [2, 4, 5]
y = [1, 15, 28]
-----
Lagrange interpolation polynomial:
L(x) = (5/3 - x/3)*(2 - x/2) + 15*(5 - x)*(x/2 - 1) + 28*(x/3 - 2/3)*(x - 4)
Interpolation point: x = 3
Lagrange poly result on x = 3
L(3) = 6.0000
-----
Newton interpolation polynomial:
N(x) = 7.0*x + 2.0*(x - 4)*(x - 2) - 13.0
Interpolation point: x = 3
Newton poly result on x = 3
N(3) = 6.0000
```

Lagrange Polynomial

```
: lagrange
: 
$$\left(\frac{5}{3} - \frac{x}{3}\right)\left(2 - \frac{x}{2}\right) + 15(5 - x)\left(\frac{x}{2} - 1\right) + 28\left(\frac{x}{3} - \frac{2}{3}\right)(x - 4)$$

: simplify(lagrange)
: 
$$2x^2 - 5x + 3$$

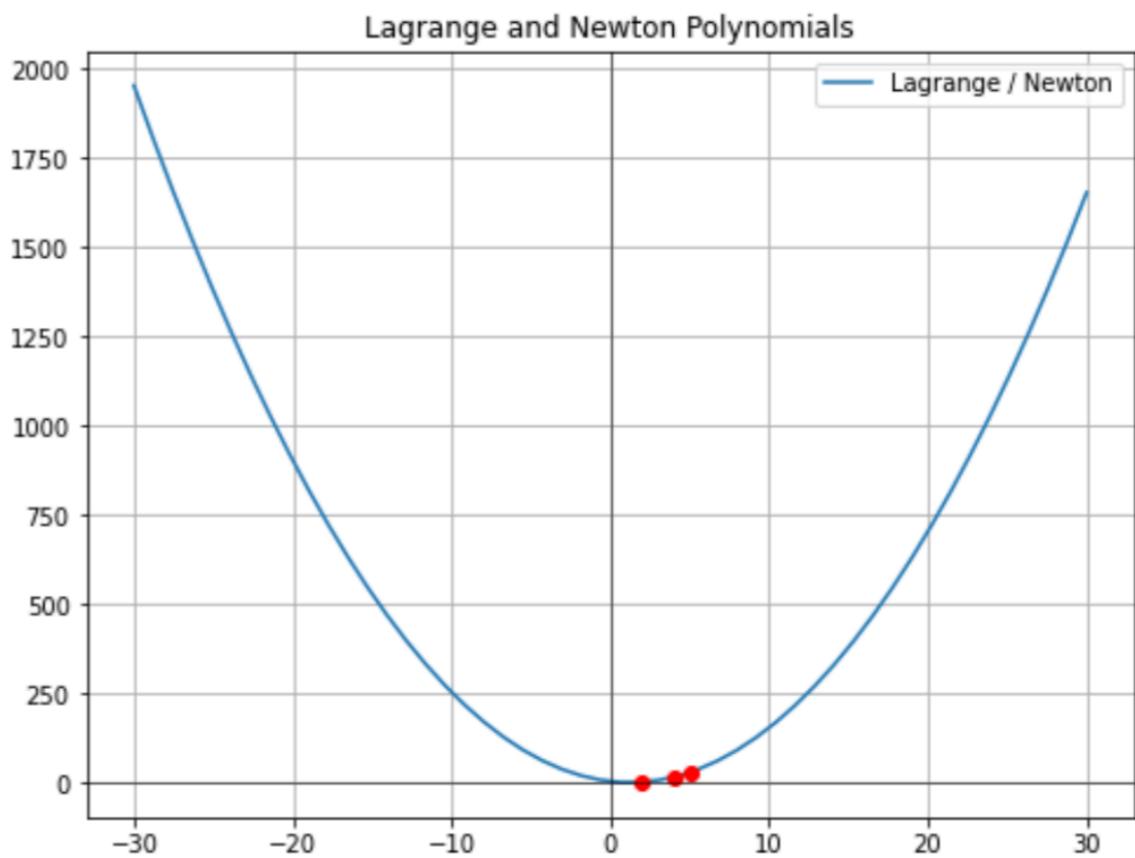
```

Newton Polynomial ¶

```
: newton
: 
$$7.0x + 2.0(x - 4)(x - 2) - 13.0$$

: simplify(newton)
: 
$$2.0x^2 - 5.0x + 3.0$$

```



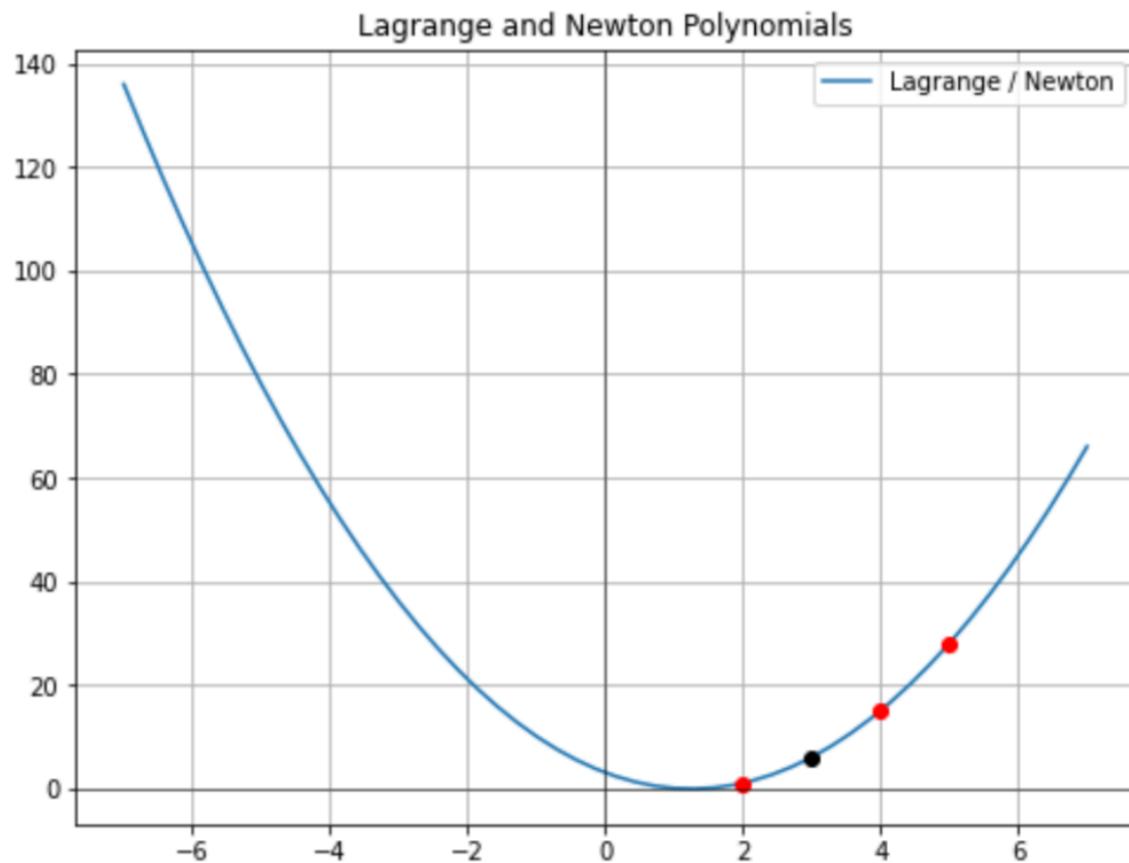
Значение в точке

Point!

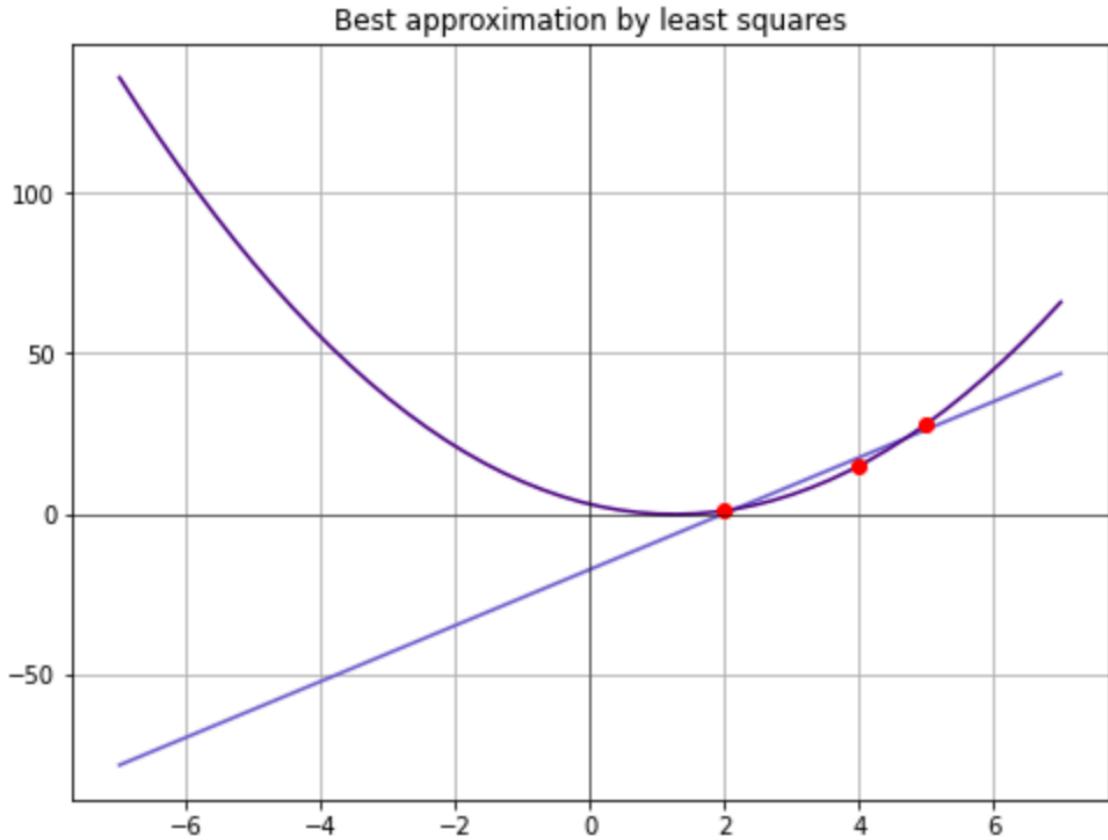
$$x = 3$$

$$L(3) = 6.0$$

$$N(3) = 6.0$$



Графики функций наилучшего приближения:



```
1 numpy.polynomial.polynomial.Polynomial(all_best[0])
```

$$x \mapsto -17.285714285714302 + 8.714285714285719 x$$

```
1 numpy.polynomial.polynomial.Polynomial(all_best[1])
```

$$x \mapsto 3.00000000000028 - 5.000000000000191 x + 2.000000000000028 x^2$$

Тестовый пример 2. Построить интерполяционные многочлены в форме Лагранжа и Ньютона. Найти значение в точке $x = 7$

```
x_2 = [5, 6, 9, 11]
y_2 = [12, 13, 14, 16]
point_2 = 7
```

```
Data:
x = [5, 6, 9, 11]
y = [12, 13, 14, 16]
-----
Lagrange interpolation polynomial:
L(x) = 12*(11/6 - x/6)*(9/4 - x/4)*(6 - x) + 13*(11/5 - x/5)*(3 - x/3)*(x - 5) + 14*(11/2 - x/2)*(x/4 - 5/4)*(x/3 - 2) + 16*(x/6 - 5/6)*(x/5 - 6/5)*(x/2 - 9/2)
Interpolation point: x = 7
Lagrange poly result on x = 7
L(7) = 13.4667
-----
Newton interpolation polynomial:
N(x) = 1.0*x + 0.05*(x - 9)*(x - 6)*(x - 5) - 0.1666666666666667*(x - 6)*(x - 5) + 7.0
Interpolation point: x = 7
Newton poly result on x = 7
N(7) = 13.4667
```

Lagrange Polynomial

```
: lagrange
: 
$$12 \left( \frac{11}{6} - \frac{x}{6} \right) \left( \frac{9}{4} - \frac{x}{4} \right) (6 - x) + 13 \left( \frac{11}{5} - \frac{x}{5} \right) \left( 3 - \frac{x}{3} \right) (x - 5) + 14 \left( \frac{11}{2} - \frac{x}{2} \right) \left( \frac{x}{4} - \frac{5}{4} \right) \left( \frac{x}{3} - 2 \right) + 16 \left( \frac{x}{6} - \frac{5}{6} \right) \left( \frac{x}{5} - \frac{6}{5} \right) \left( \frac{x}{2} - \frac{9}{2} \right)$$

: simplify(lagrange)
: 
$$\frac{x^3}{20} - \frac{7x^2}{6} + \frac{557x}{60} - \frac{23}{2}$$

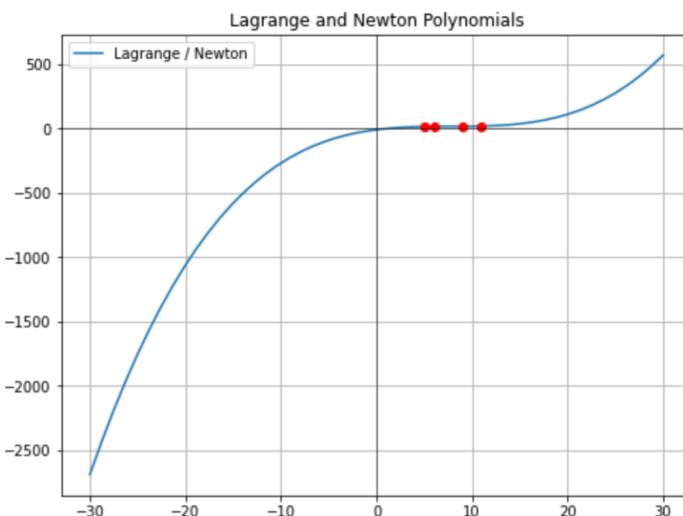
```

Newton Polynomial

```
: newton
: 
$$1.0x + 0.05(x - 9)(x - 6)(x - 5) - 0.1666666666666667(x - 6)(x - 5) + 7.0$$

: simplify(newton)
: 
$$0.05x^3 - 1.1666666666667x^2 + 9.2833333333333x - 11.5$$

```



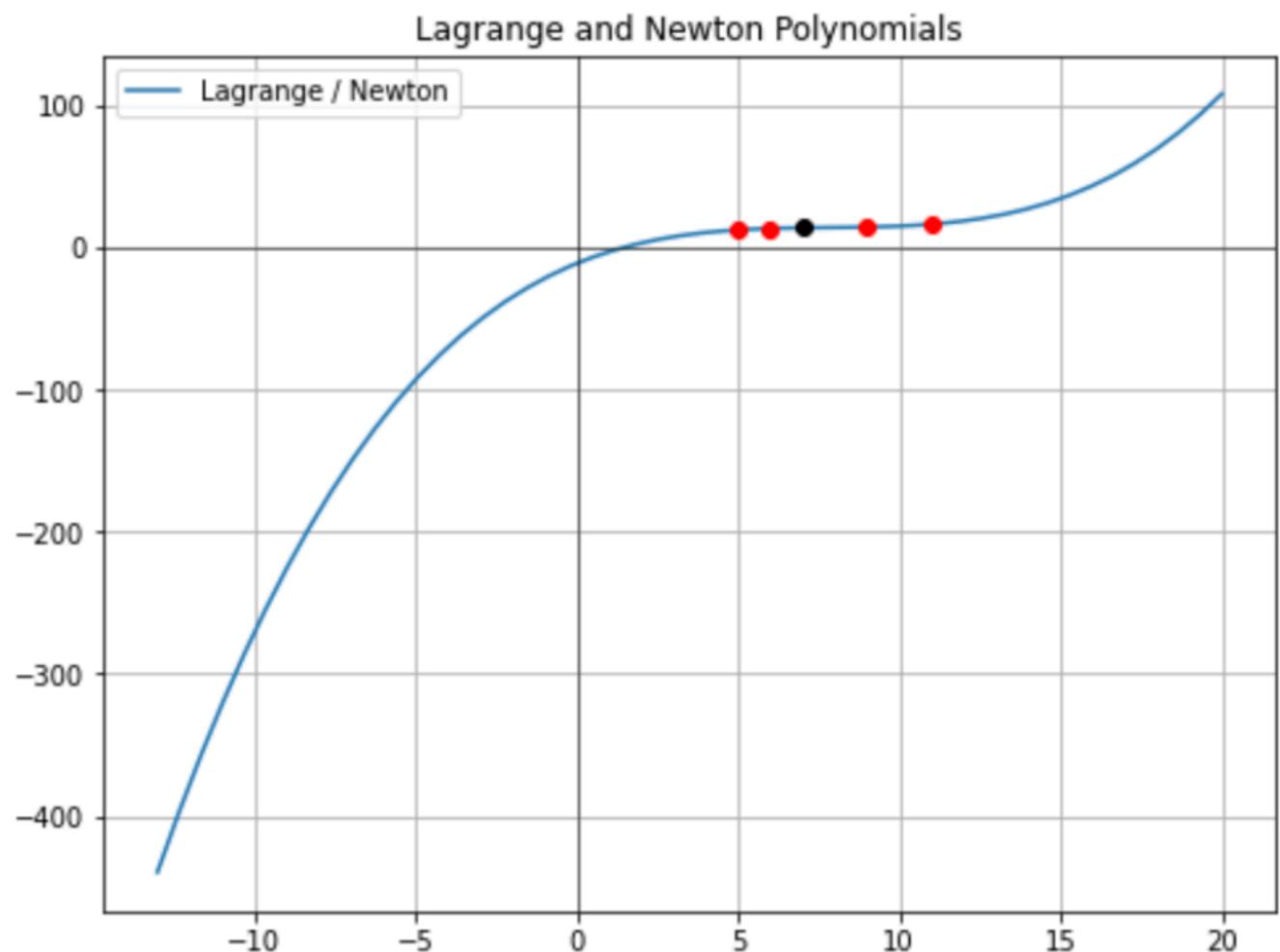
Найдем значение в точке

Point!

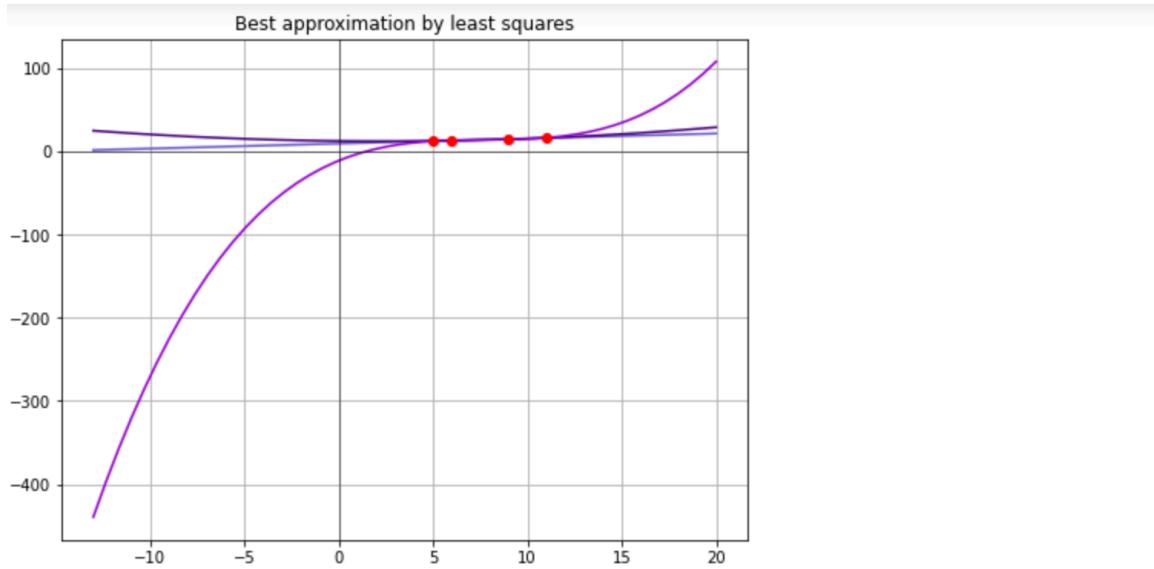
$x = 7$

$L(7) = 13.46666666666669$

$N(7) = 13.46666666666665$



Графики функций наилучшего приближения:



```
1 numpy.polynomial.polynomial.Polynomial(all_best[0])
```

```
:> 9.065934065934071 + 0.6043956043956038 x
```

```
1 numpy.polynomial.polynomial.Polynomial(all_best[1])
```

```
:> 12.169491525423739 - 0.2514124293785342 x + 0.053672316384180976 x2
```

```
1 numpy.polynomial.polynomial.Polynomial(all_best[2])
```

```
:> -11.500000000000001963 + 9.28333333334225 x - 1.166666666666794 x2 + 0.0500000000000000571 x3
```

Тестовый пример 3. Построить интерполяционные многочлены в форме Лагранжа и Ньютона. Найти значение в точке $x = 1$

```
x_3 = [-5, -1, 0, 2]
y_3 = [-2, 6, 1, 3]
point_3 = 1
```

```

Data:
x = [-5, -1, 0, 2]
y = [-2, 6, 1, 3]
-----
Lagrange interpolation polynomial:
L(x) = 2*x*(2/7 - x/7)*(-x/4 - 1/4)/5 - 6*x*(2/3 - x/3)*(x/4 + 5/4) + 3*x*(x/7 + 5/7)*(x/3 + 1/3)/2 + (1 - x/2)*(x/5 + 1)*(x + 1)
Interpolation point: x = 1
Lagrange poly result on x = 1
L(1) = -0.9714

```

```

Newton interpolation polynomial:
N(x) = 0.485714285714286*x*(x + 1)*(x + 5) + 2.0*x - 1.4*(x + 1)*(x + 5) + 8.0
Interpolation point: x = 1
Newton poly result on x = 1
N(1) = -0.9714

```

Lagrange Polynomial

```
lagrange
```

$$\frac{2x \left(\frac{2}{7} - \frac{x}{7}\right) \left(-\frac{x}{4} - \frac{1}{4}\right)}{5} - 6x \left(\frac{2}{3} - \frac{x}{3}\right) \left(\frac{x}{4} + \frac{5}{4}\right) + \frac{3x \left(\frac{x}{7} + \frac{5}{7}\right) \left(\frac{x}{3} + \frac{1}{3}\right)}{2} + \left(1 - \frac{x}{2}\right) \left(\frac{x}{5} + 1\right) (x + 1)$$

```
simplify(lagrange)
```

$$\frac{17x^3}{35} + \frac{53x^2}{35} - \frac{139x}{35} + 1$$

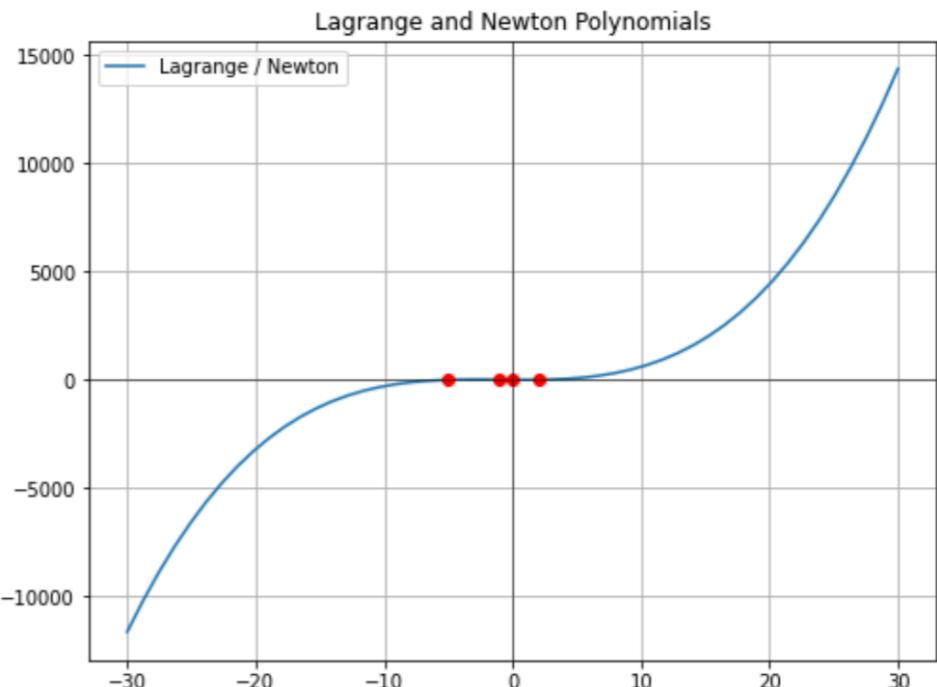
Newton Polynomial

```
newton
```

$$0.485714285714286x(x + 1)(x + 5) + 2.0x - 1.4(x + 1)(x + 5) + 8.0$$

```
simplify(newton)
```

$$0.485714285714286x^3 + 1.51428571428571x^2 - 3.97142857142857x + 1.0$$



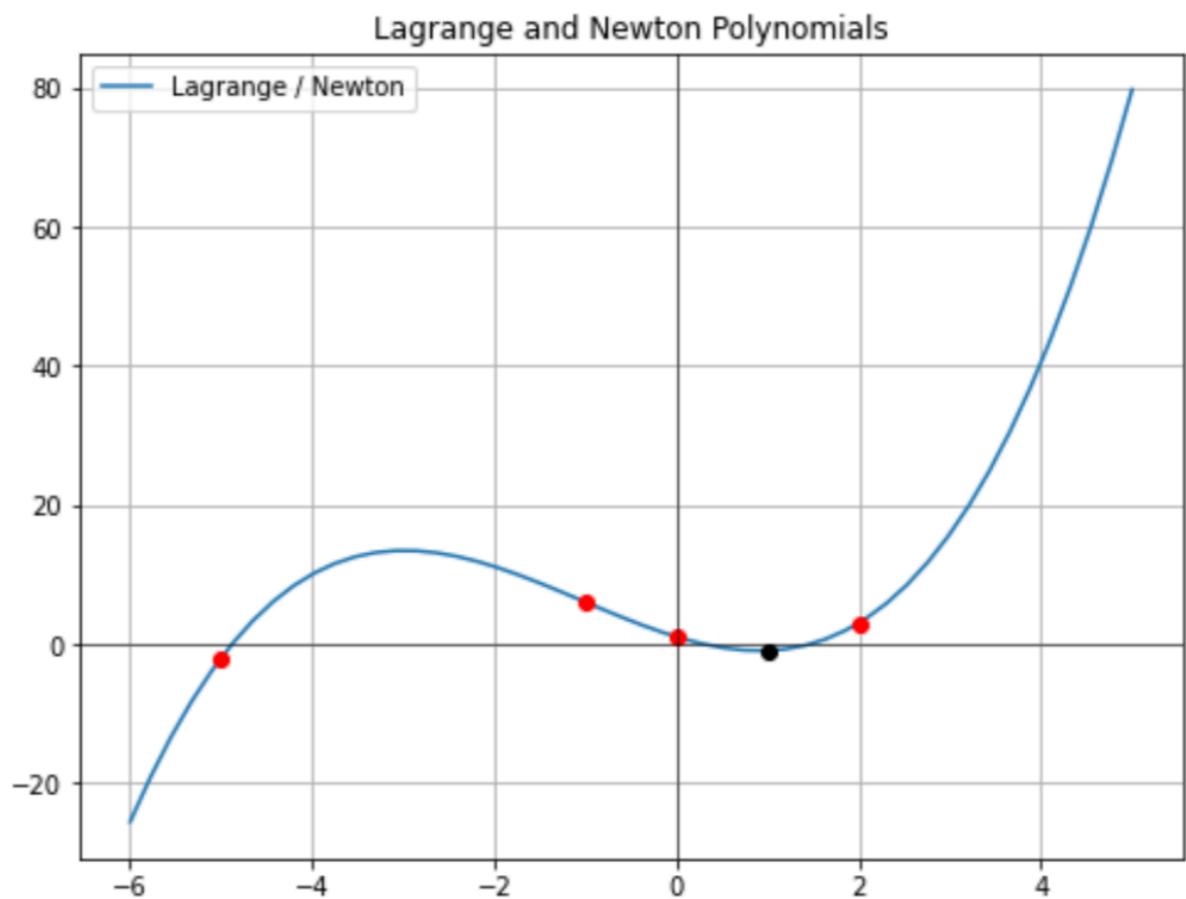
Найдем значение в точке

Point!

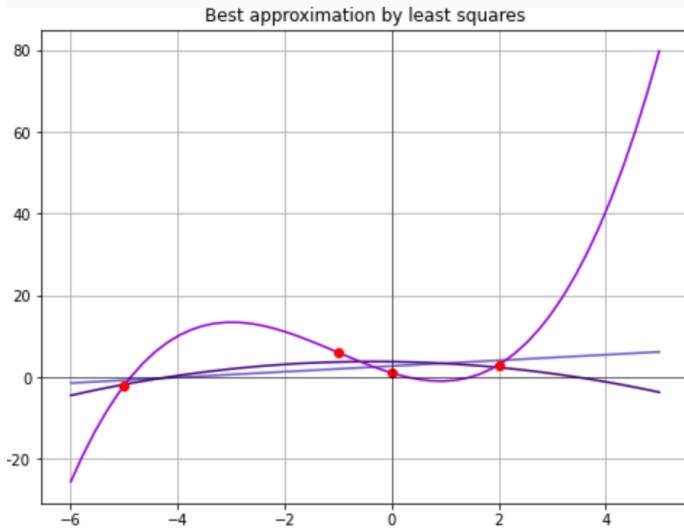
$x = 1$

$L(1) = -0.9714285714285713$

$N(1) = -0.9714285714285644$



Многочлены наилучшего приближения



```
1 numpy.polynomial.polynomial.Polynomial(all_best[0])
↪ 2.6923076923076925 + 0.6923076923076923 x
```

```
1 numpy.polynomial.polynomial.Polynomial(all_best[1])
↪ 3.7656552614590058 - 0.19044544867656568 x - 0.2608134280180762 x2
```

```
1 numpy.polynomial.polynomial.Polynomial(all_best[2])
↪ 0.999999999999345 - 3.9714285714286253 x + 1.5142857142857467 x2 + 0.4857142857142938 x3
```

Тестовый пример 4. Построить интерполяционные многочлены в форме Лагранжа и Ньютона. Найти значение в точке $x = 2.5$

```
x_4 = [0, 1, 2, 3]
y_4 = [5, 1, 7, 29]
point_4 = 2.5
```

```

Data:
x = [0, 1, 2, 3]
y = [5, 1, 7, 29]
-----
Lagrange interpolation polynomial:
L(x) = x*(3/2 - x/2)*(2 - x) + 7*x*(3 - x)*(x - 1)/2 + 29*x*(x/2 - 1/2)*(x - 2)/3 + 5*(1 - x)*(1 - x/2)*(1 - x/3)
Interpolation point: x = 2.5
Lagrange poly result on x = 2.5
L(2.5) = 15.6250
-----
Newton interpolation polynomial:
N(x) = 1.0*x*(x - 2)*(x - 1) + 5.0*x*(x - 1) - 4.0*x + 5
Interpolation point: x = 2.5
Newton poly result on x = 2.5
N(2.5) = 15.6250

```

Lagrange Polynomial

```

|: lagrange
|:  $x \left(\frac{3}{2} - \frac{x}{2}\right) (2 - x) + \frac{7x (3 - x) (x - 1)}{2} + \frac{29x \left(\frac{x}{2} - \frac{1}{2}\right) (x - 2)}{3} + 5 (1 - x) \left(1 - \frac{x}{2}\right) \left(1 - \frac{x}{3}\right)$ 
|: simplify(lagrange)
|:  $x^3 + 2x^2 - 7x + 5$ 

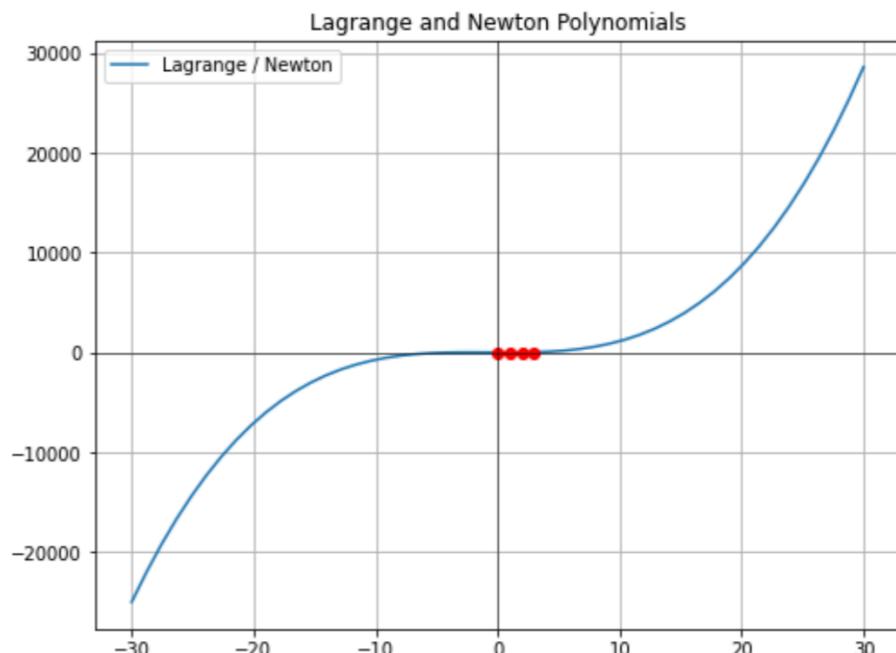
```

Newton Polynomial

```

|: newton
|:  $1.0x (x - 2) (x - 1) + 5.0x (x - 1) - 4.0x + 5$ 
|: simplify(newton)
|:  $1.0x^3 + 2.0x^2 - 7.0x + 5.0$ 

```



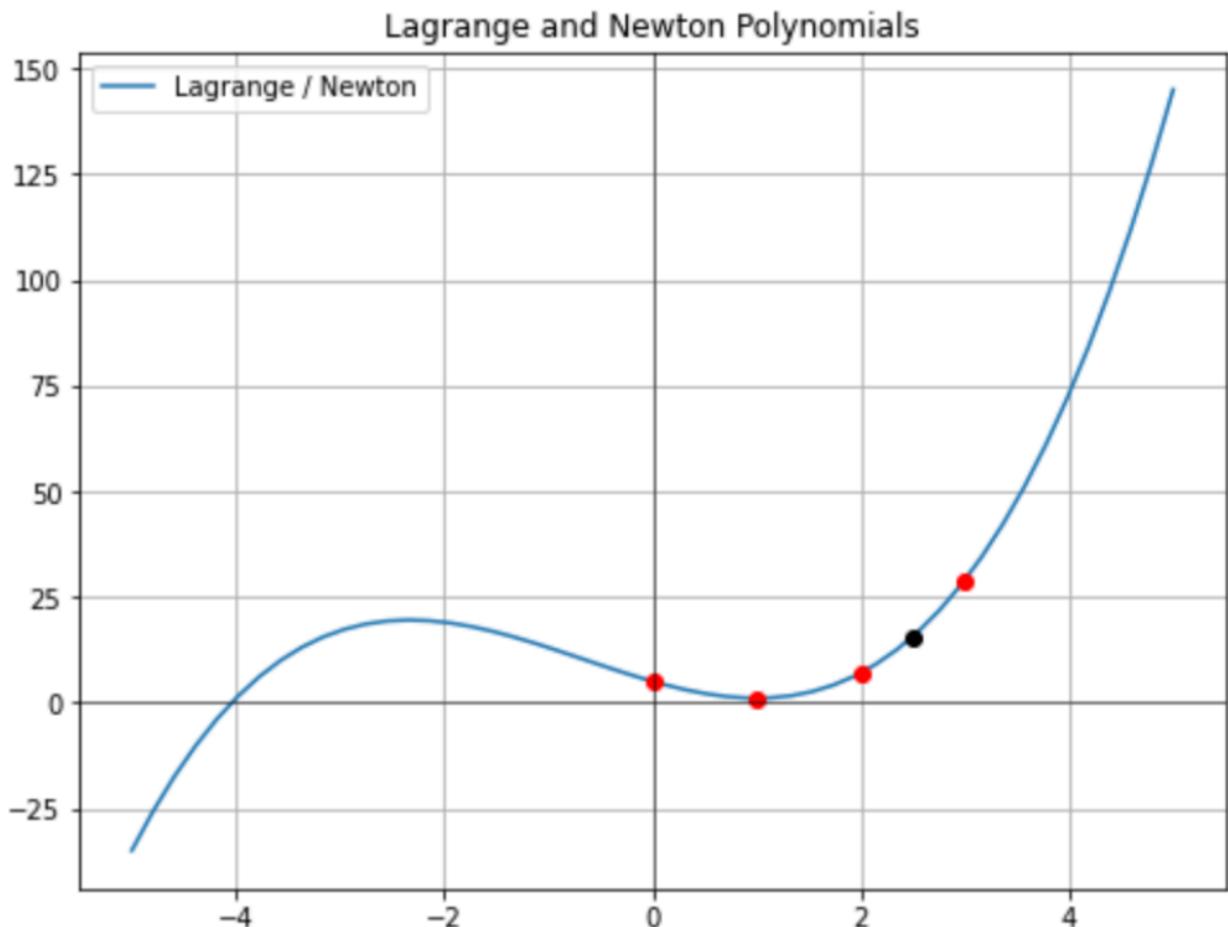
Найдем значение в точке

Point!

$x = 2.5$

$L(2.5) = 15.625$

$N(2.5) = 15.625$



Найдем многочлены наилучшего приближения

```
numpy.polynomial.polynomial.Polynomial(all_best[0])
```

```
→ -1.2000000000000053 + 7.8000000000000025 x
```

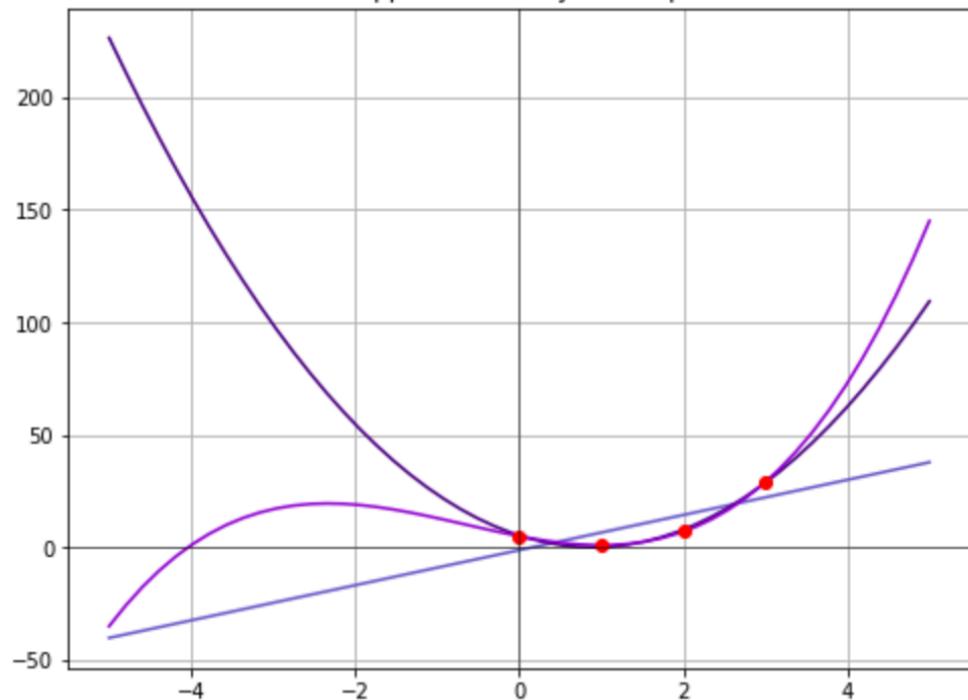
```
numpy.polynomial.polynomial.Polynomial(all_best[1])
```

```
→ 5.300000000000003 - 11.69999999999985 x + 6.49999999999994 x2
```

```
numpy.polynomial.polynomial.Polynomial(all_best[2])
```

```
→ 0.999999999999345 - 3.9714285714286253 x + 1.5142857142857467 x2 + 0.4857142857142938 x3
```

Best approximation by least squares



Тестовый пример 5. Построить интерполяционные многочлены в форме Лагранжа и Ньютона. Найти значение в точке $x = 0.8$

```
x_5 = [0.1, 0.5, 0.7, 1.2, 1.5]
y_5 = [1.2, 2.7, 3.8, 4.7, 6]
point_5 = 0.8
```

```

Data:
x = [0.1, 0.5, 0.7, 1.2, 1.5]
y = [1.2, 2.7, 3.8, 4.7, 6]
-----
Lagrange interpolation polynomial:
L(x) = 1.2*(1.07142857142857 - 0.714285714285714*x)*(1.09090909090909 - 0.909090909090909*x)*(1.166666666666667 - 1.66666666666667*x)*(1.25 - 2.5*x) + 2.7*(1.5 - 1.0*x)*(1.71428571428571 - 1.42857142857143*x)*(3.5 - 5.0*x)*(2.5*x - 0.25) + 3.8*(1.875 - 1.25*x)*(2.4 - 2.0*x)*(1.66666666666667*x - 0.166666666666667)*(5.0*x - 2.5) + 4.7*(5.0 - 3.3333333333*x)*(0.909090909090909*x - 0.0909090909090909)*(1.42857142857143*x - 0.714285714285714)*(2.0*x - 1.4) + 6*(0.714285714285714*x - 0.0714285714285714)*(1.0*x - 0.5)*(1.25*x - 0.875)*(3.333333333333*x - 4.0)
Interpolation point: x = 0.8
Lagrange poly result on x = 0.8
L(0.8) = 4.1855
-----
Newton interpolation polynomial:
N(x) = 3.75*x + 11.3636363636364*(x - 1.2)*(x - 0.7)*(x - 0.5)*(x - 0.1) - 7.45670995670995*(x - 0.7)*(x - 0.5)*(x - 0.1) + 2.91666666666666*(x - 0.5)*(x - 0.1) + 0.825
Interpolation point: x = 0.8
Newton poly result on x = 0.8
N(0.8) = 4.1855

```

Lagrange Polynomial

```
lagrange
```

$$1.2(1.07142857142857 - 0.714285714285714x)(1.090909090909 - 0.909090909090909x)(1.166666666666667 - 1.66666666666667x)(1.25 - 2.5x) + 2.7(1.5 - 1.0x)(1.71428571428571 - 1.42857142857143x)(3.5 - 5.0x)(2.5x - 0.25) + 3.8(1.875 - 1.25x)(2.4 - 2.0x)(1.66666666666667x - 0.166666666666667)(5.0x - 2.5) + 4.7(5.0 - 3.33333333333x)(0.909090909090909x - 0.0909090909090909)(1.42857142857143x - 0.714285714285714)(2.0x - 1.4) + 6(0.714285714285714x - 0.0714285714285714)(1.0x - 0.5)(1.25x - 0.875)(3.333333333333x - 4.0)$$

```
simplify(lagrange)
```

$$11.36363636364x^4 - 35.8658008658009x^3 + 35.6785714285714x^2 - 8.31147186147186x + 1.70909090909091$$

Newton Polynomial

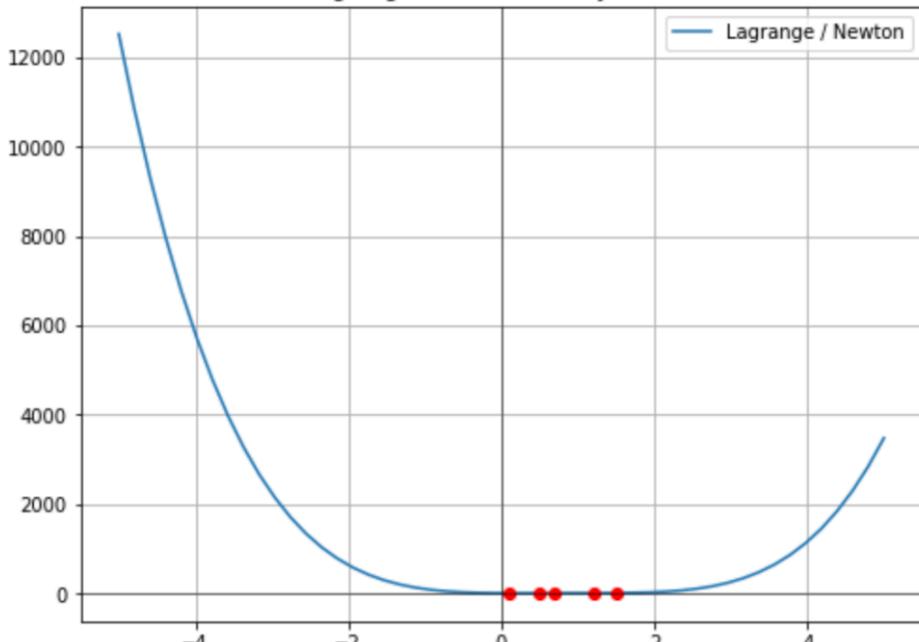
```
newton
```

$$3.75x + 11.3636363636364(x - 1.2)(x - 0.7)(x - 0.5)(x - 0.1) - 7.45670995670995(x - 0.7)(x - 0.5)(x - 0.1) + 2.91666666666666(x - 0.5)(x - 0.1) + 0.825$$

```
simplify(newton)
```

$$11.36363636364x^4 - 35.8658008658009x^3 + 35.6785714285714x^2 - 8.31147186147185x + 1.70909090909091$$

Lagrange and Newton Polynomials



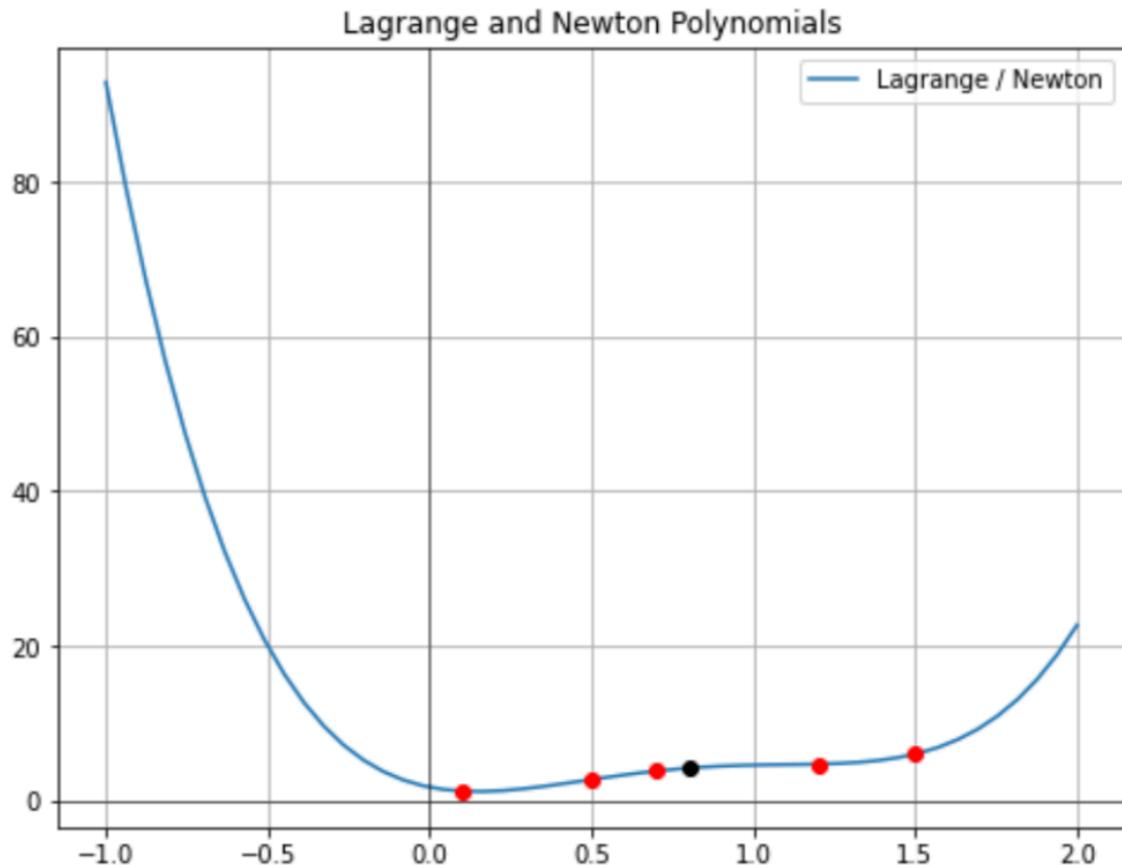
Найдем значение в точке

Point!

$x = 0.8$

$L(0.8) = 4.185454545454566$

$N(0.8) = 4.1854545454544$



Тестовый пример 6. Построить интерполяционные многочлены в форме Лагранжа и Ньютона. Найти значение в точке $x = 0.25$

```
x_6 = [3/2, -1/3, 0, 1/6, 0.5]
y_6 = [1, -np.sqrt(3)/2, 0, 0.5, 1]
point_6 = 0.25
```

Lagrange Polynomial

```
: lagrange
: 2.59807621135332x(0.33333333333333 - 2.0x)(0.6 - 1.2x)(0.8181818181818 - 0.545454545454546x)
+ 3.0x(1.125 - 0.75x)(1.5 - 3.0x)(2.0x + 0.66666666666667) + 2.0x(1.5 - 1.0x)(1.2x + 0.4)(3.0x - 0.5)
+ 0.66666666666667x(0.5454545454546x + 0.1818181818182)(0.75x - 0.125)(1.0x - 0.5)

: simplify(lagrange)
: x(3.17160932331929x3 - 5.62182020052513x2 - 0.397423233070764x + 3.20771550125175)
```

Newton Polynomial

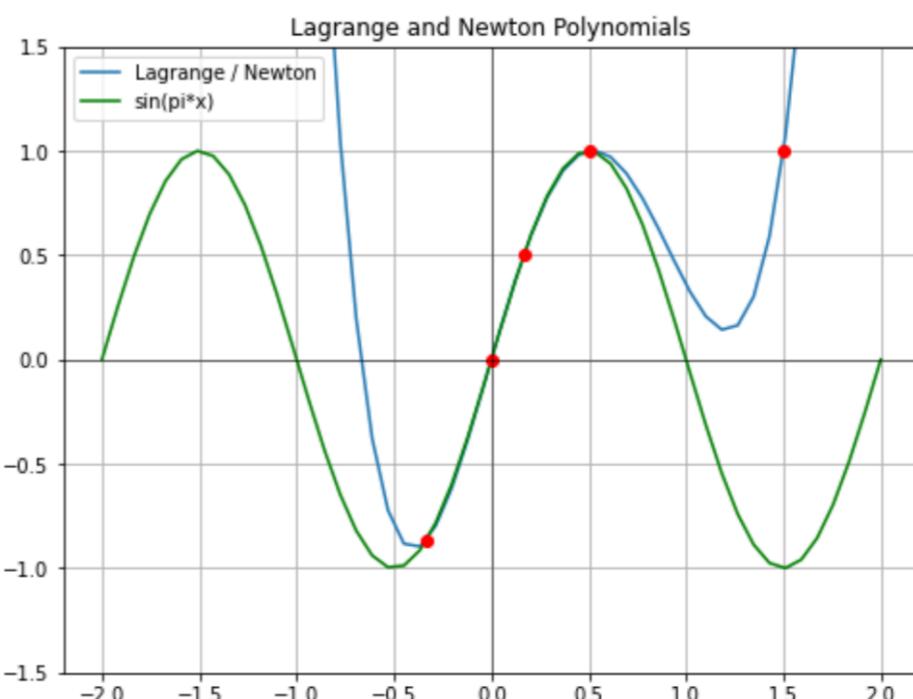
```
: newton
: 3.1716093233193x(x - 1.5)(x - 0.16666666666667)(x + 0.33333333333333) - 1.39300776943275x(x - 1.5)(x + 0.33333333333333)
+ 1.01783203842788x - 1.05349611528363(x - 1.5)(x + 0.33333333333333) - 0.526748057641814

: simplify(newton)
: 3.1716093233193x4 - 5.62182020052514x3 - 0.397423233070763x2 + 3.20771550125176x - 1.11022302462516 · 10-16
```

Data:
 $x = [1.5, -0.33333333333333, 0, 0.1666666666666666, 0.5]$
 $y = [1, -0.8660254037844386, 0, 0.5, 1]$

Lagrange interpolation polynomial:
 $L(x) = 2.59807621135332*x*(0.33333333333333 - 2.0*x)*(0.6 - 1.2*x)*(0.8181818181818 - 0.545454545454546*x) + 3.0*x*(1.125 - 0.75*x)*(1.5 - 3.0*x)*(2.0*x + 0.66666666666667) + 2.0*x*(1.5 - 1.0*x)*(1.2*x + 0.4)*(3.0*x - 0.5) + 0.66666666666667*x*(0.5454545454546*x + 0.1818181818182)*(0.75*x - 0.125)*(1.0*x - 0.5)$
Interpolation point: $x = 0.25$
Lagrange poly result on $x = 0.25$
 $L(0.25) = 0.7016$

Newton interpolation polynomial:
 $N(x) = 3.1716093233193*x*(x - 1.5)*(x - 0.16666666666667)*(x + 0.33333333333333) - 1.39300776943275*x*(x - 1.5)*(x + 0.33333333333333) + 1.01783203842788*x - 1.05349611528363*(x - 1.5)*(x + 0.33333333333333) - 0.526748057641814$
Interpolation point: $x = 0.25$
Newton poly result on $x = 0.25$
 $N(0.25) = 0.7016$



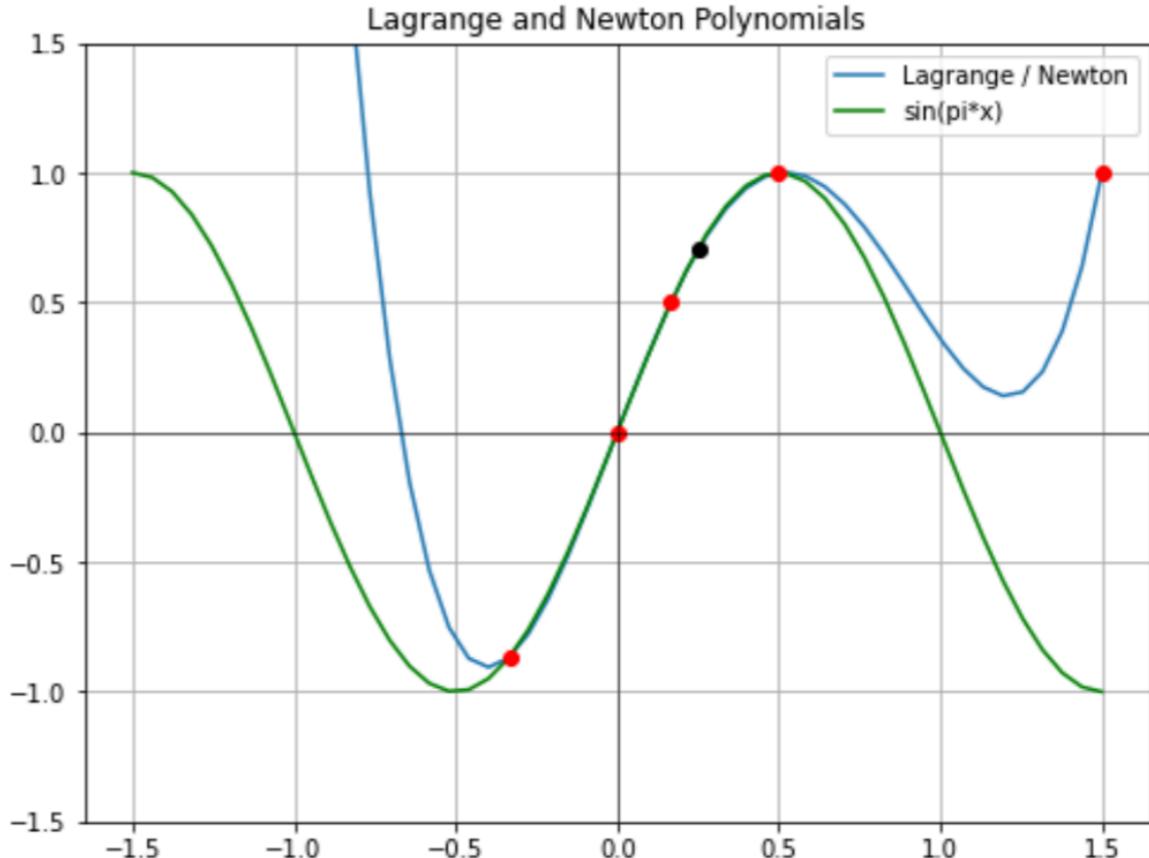
Найдем значение в точке

Point!

$x = 0.25$

$L(0.25) = 0.7016380815320267$

$N(0.25) = 0.7016380815320298$



Error = 0.03436384974342177

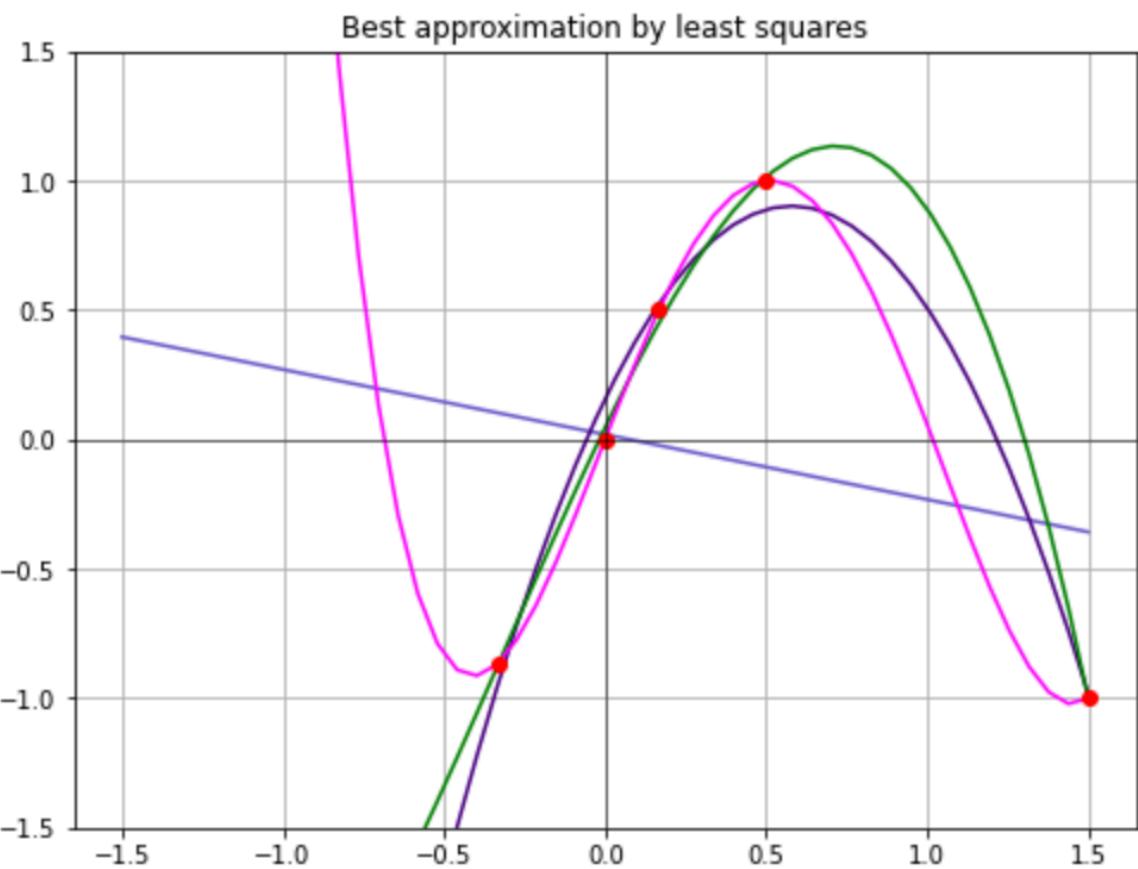
Найдем многочлены наилучшего приближения:

```
numpy.polynomial.polynomial.Polynomial(all_best[0])
→ -0.0037428594428794426 + 0.9014666691436137 x

numpy.polynomial.polynomial.Polynomial(all_best[1])
→ 0.07216992603554165 + 2.4660813316759604 x - 1.2308302011921126 x2

numpy.polynomial.polynomial.Polynomial(all_best[2])
→ 0.04592725808139893 + 2.5789679105470213 x - 0.8375979879025193 x2 - 0.8978683116388972 x3

numpy.polynomial.polynomial.Polynomial(all_best[3])
→ 9.887630585144005e-14 + 3.1925639860989103 x - 0.32166565731428876 x2 - 5.4400020186971 x3 + 2.626154777859041 x4
```



4.

**Решение задания
Вариант 10**

Построить интерполяционные многочлены в форме Лагранжа и Ньютона. Оценить погрешность. Вычислить значение функции в точке интерполяционного многочлена и многочлена наилучшего приближения. Сравнить значения

$$y_i = p_i + (-1)^k m$$

```
x = [0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
y = [1.8  2.21 2.59 2.93 3.26 3.56 3.84 4.1  4.35 4.59 4.81]
```

Data:

```
x = [0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
y = [1.8  2.21 2.59 2.93 3.26 3.56 3.84 4.1   4.35 4.59 4.81]
```

Lagrange Polynomial

```
: lagrange
```

$$\begin{aligned}
& 22.1x(1.1111111111111 - 1.1111111111111x)(1.125 - 1.25x)(1.14285714285714 - 1.42857142857143x)(1.16666666666667 \\
& - 1.6666666666667x)(1.2 - 2.0x)(1.25 - 2.5x)(1.3333333333333 - 3.3333333333333x)(1.5 - 5.0x)(2.0 - 10.0x) \\
& + 12.95x(1.25 - 1.25x)(1.28571428571429 - 1.42857142857143x)(1.3333333333333 - 1.6666666666667x)(1.4 - 2.0x)(1.5 \\
& - 2.5x)(1.6666666666667 - 3.3333333333333x)(2.0 - 5.0x)(3.0 - 10.0x)(10.0x - 1.0) \\
& + 9.7666666666667x(1.42857142857143 - 1.42857142857143x)(1.5 - 1.6666666666667x)(1.6 - 2.0x)(1.75 - 2.5x)(2.0 \\
& - 3.333333333333x)(2.5 - 5.0x)(4.0 - 10.0x)(5.0x - 0.5)(10.0x - 2.0) \\
& + 8.15x(1.6666666666667 - 1.6666666666667x)(1.8 - 2.0x)(2.0 - 2.5x)(2.3333333333333 - 3.3333333333333x)(3.0 - 5.0x)(5.0 \\
& - 10.0x)(3.333333333333x - 0.3333333333333)(5.0x - 1.0)(10.0x - 3.0) \\
& + 7.12x(2.0 - 2.0x)(2.25 - 2.5x)(2.6666666666667 - 3.333333333333x)(3.5 - 5.0x)(6.0 - 10.0x)(2.5x - 0.25)(3.333333333333x \\
& - 0.6666666666667)(5.0x - 1.5)(10.0x - 4.0) \\
& + 6.4x(2.5 - 2.5x)(3.0 - 3.333333333333x)(4.0 - 5.0x)(7.0 - 10.0x)(2.0x - 0.2)(2.5x - 0.5)(3.333333333333x - 1.0)(5.0x \\
& - 2.0)(10.0x - 5.0) \\
& + 5.85714285714286x(3.333333333333 - 3.333333333333x)(4.5 - 5.0x)(7.99999999999999 \\
& - 9.99999999999999x)(1.6666666666667x - 0.16666666666667)(2.0x - 0.4)(2.5x - 0.75)(3.333333333333x \\
& - 1.333333333333)(5.0x - 2.5)(10.0x - 6.0) \\
& + 5.4375x(5.0 - 5.0x)(9.0 - 10.0x)(1.42857142857143x - 0.142857142857143)(1.6666666666667x - 0.3333333333333)(2.0x \\
& - 0.6)(2.5x - 1.0)(3.333333333333x - 1.6666666666667)(5.0x - 3.0)(9.9999999999999x - 6.9999999999999) \\
& + 5.1x(10.0 - 10.0x)(1.25x - 0.125)(1.42857142857143x - 0.285714285714286)(1.6666666666667x - 0.5)(2.0x - 0.8)(2.5x \\
& - 1.25)(3.333333333333x - 2.0)(5.0x - 3.5)(10.0x - 8.0) \\
& + 4.81x(1.111111111111x - 0.1111111111111)(1.25x - 0.25)(1.42857142857143x - 0.428571428571429)(1.6666666666667x \\
& - 0.6666666666667)(2.0x - 1.0)(2.5x - 1.5)(3.333333333333x - 2.333333333333)(5.0x - 4.0)(10.0x - 9.0) \\
& + 1.8(1.0 - 10.0x)(1.0 - 5.0x)(1.0 - 3.333333333333x)(1.0 - 2.5x)(1.0 - 2.0x)(1.0 - 1.6666666666667x)(1.0 \\
& - 1.42857142857143x)(1.0 - 1.25x)(1.0 - 1.111111111111x)(1.0 - 1.0x)
\end{aligned}$$

```
simplify(lagrange)
```

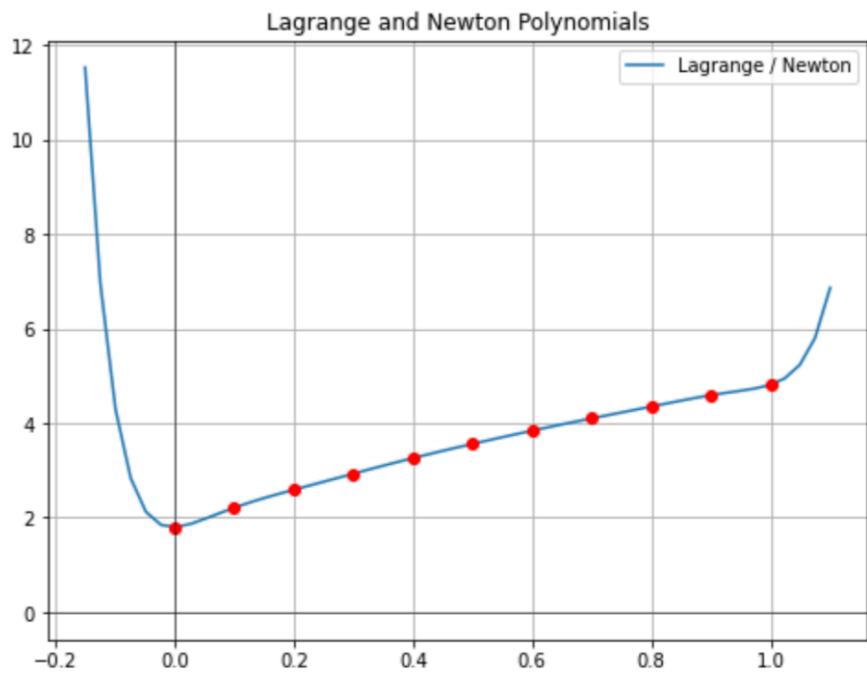
$$\begin{aligned}
& 3279.32098765438x^{10} - 16823.7433862444x^9 + 37136.2433862388x^8 - 46113.5912698247x^7 + 35322.1643518358x^6 \\
& - 17192.6215277705x^5 + 5268.12692901166x^4 - 966.267030423158x^3 + 92.7493452380804x^2 + 0.62821428571533x + 1.8
\end{aligned}$$
Newton Polynomial

```
newton
```

$$\begin{aligned}
& 3279.32098765415x(x - 0.9)(x - 0.8)(x - 0.7)(x - 0.6)(x - 0.5)(x - 0.4)(x - 0.3)(x - 0.2)(x - 0.1) \\
& - 2066.79894179886x(x - 0.8)(x - 0.7)(x - 0.6)(x - 0.5)(x - 0.4)(x - 0.3)(x - 0.2)(x - 0.1) \\
& + 1165.67460317457x(x - 0.7)(x - 0.6)(x - 0.5)(x - 0.4)(x - 0.3)(x - 0.2)(x - 0.1) \\
& - 575.396825396813x(x - 0.6)(x - 0.5)(x - 0.4)(x - 0.3)(x - 0.2)(x - 0.1) \\
& + 236.11111111108x(x - 0.5)(x - 0.4)(x - 0.3)(x - 0.2)(x - 0.1) - 74.999999999997x(x - 0.4)(x - 0.3)(x - 0.2)(x - 0.1) \\
& + 16.666666666666x(x - 0.3)(x - 0.2)(x - 0.1) - 1.6666666666666x(x - 0.2)(x - 0.1) - 1.5x(x - 0.1) + 4.1x + 1.8
\end{aligned}$$

```
simplify(newton)
```

$$\begin{aligned}
& 3279.32098765415x^{10} - 16823.7433862425x^9 + 37136.2433862416x^8 - 46113.5912698391x^7 + 35322.1643518503x^6 \\
& - 17192.621527777x^5 + 5268.12692901213x^4 - 966.267030423243x^3 + 92.7493452380918x^2 + 0.62821428571441x + 1.8
\end{aligned}$$



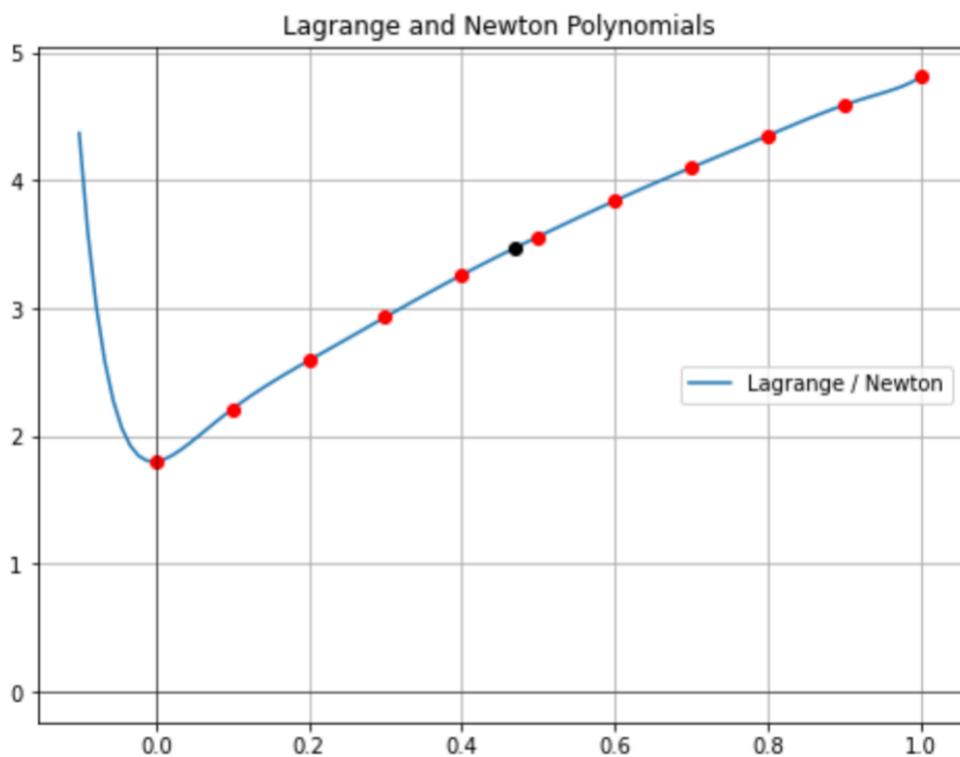
Найдем приближение в точке

Point!

$x = 0.47$

$L(0.47) = 3.47265207970881$

$N(0.47) = 3.4726520797088036$



Многочлены меньшей степени наилучшего приближения

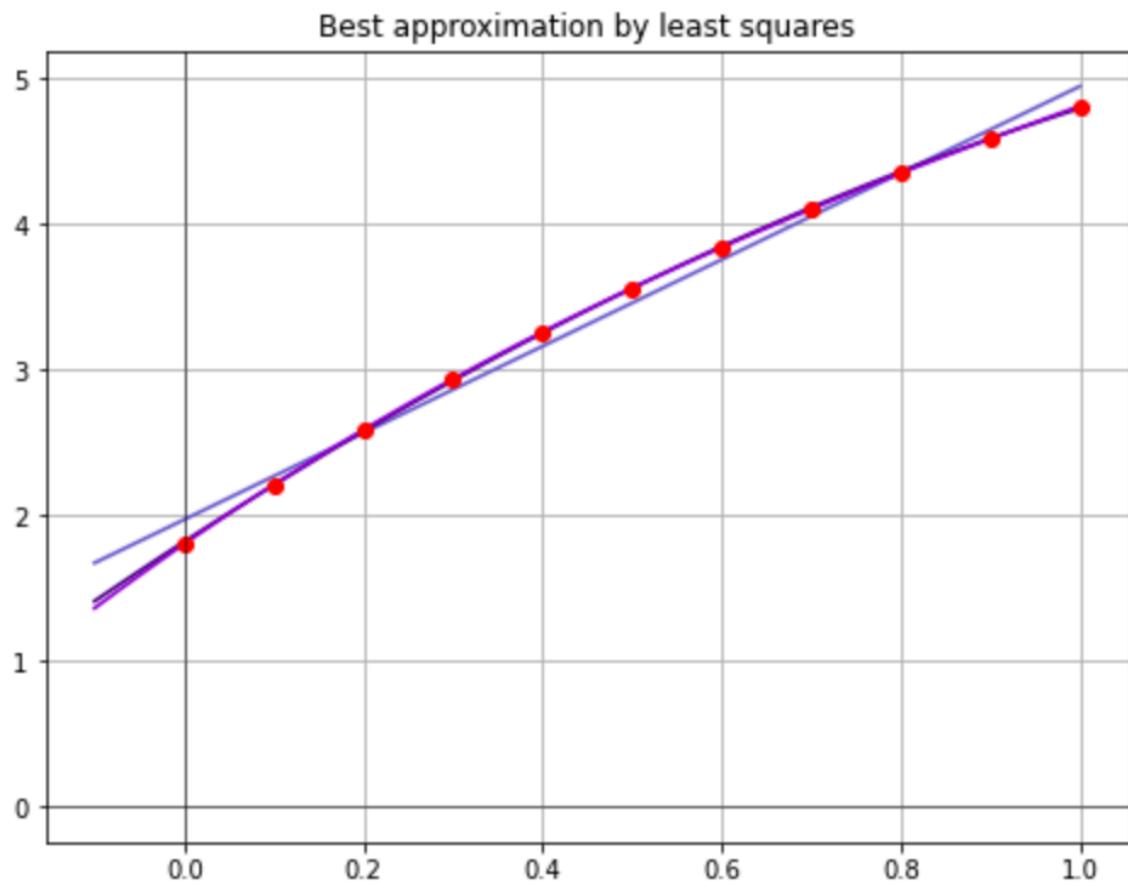
```
1 numpy.polynomial.polynomial.Polynomial(all_best[0])
```

$$x \mapsto 1.96863636363657 + 2.9790909090909063 x$$

Polynomial of smallest degree with best approximation

```
1 numpy.polynomial.polynomial.Polynomial(all_best[1])
```

$$x \mapsto 1.8170629370629405 + 3.989580419580408 x - 1.0104895104895015 x^2$$



5.

Выводы

В ходе выполнения лабораторной работы был изучен такой прием, как интерполяция, который может обеспечить нахождение неизвестных промежуточных значений некоторой функции, по имеющемуся дискретному набору её известных значений. Также такой прием, позволяет по значениям, полученными опытным путем, строить функцию, на которую с высокой точностью могли бы попадать другие получаемые значения.

Также был изучен интерполяционный многочлен Лагранжа, который довольно просто находится, но не предоставляет гибкости, а также интерполяционный многочлен Ньютона, который более сложно находится, но благодаря нему можно добавлять новые полученные точки и на их основе достраивать, буквально немного изменять уже до этого полученный многочлен Ньютона, в результате чего будет получен новый многочлен, не пересчитывая полностью старые коэффициенты, что довольно круто может играть при необходимости добавлять большое количество функций, при условии, что очень важна производительности и время работы алгоритма.

Lagrange interpolation

```

def lagrange_l(x_ls: list, x_j):
    x = symbols('x')
    lg = 1
    n = len(x_ls)
    for i in range(n):
        if x_j == x_ls[i]:
            continue
        lg *= (x - x_ls[i]) / (x_j - x_ls[i])
    return lg

def lagrange_poly(x, y):
    if len(x) != len(y):
        raise ValueError("x and y are not same length")

    n = len(x)
    # x = symbols('x')

    L = 0
    for i in range(n):
        L += y[i] * lagrange_l(x, x[i])
    return L

```

Newton Interpolation

```

def y_diff_table(y: list):
    f_diff = np.zeros(shape=(len(y) - 1, len(y) - 1))
    n = f_diff.shape[0]

    for i in range(n):
        f_diff[0, i] = (y[i+1] - y[i])

    for i in range(1, n):
        for j in range(n-i):
            f_diff[i, j] = f_diff[i-1, j+1] - f_diff[i - 1, j]

    return f_diff

def get_a(x: list, y: list):
    a_table = np.zeros(shape=(len(y) - 1, len(y) - 1))
    n = a_table.shape[0]

    for i in range(n):
        a_table[0, i] = (y[i + 1] - y[i]) / (x[i+1] - x[i])

    for i in range(1, n):
        for j in range(n - i):
            a_table[i, j] = (a_table[i - 1, j + 1] - a_table[i - 1, j]) / (x[i+1+j] - x[j])

    return a_table

```

```

def newton_poly(x_ls: list, y_ls: list):
    if len(x_ls) != len(y_ls):
        raise TypeError("x and y must be same size")

    n = len(x_ls)

    x = symbols('x')
    N = 0
    a_table = get_a(x_ls, y_ls)
    for i in range(n):
        if i == 0:
            N += y_ls[0]
            continue
        x_poly = 1
        for j in range(i):
            x_poly *= (x-x_ls[j])

        N += a_table[i-1, 0] * x_poly

    return N

```

Least squares method

1	def sum_of_pow(x, pow):
2	n = x.shape[0]
3	sum = 0
4	for i in range(n):
5	sum += np.power(x[i], pow)
6	return sum
7	
8	
9	def sum_x_y_pows(x, y, pow):
10	n = x.shape[0]
11	sum = 0
12	
13	for i in range(n):
14	sum += y[i] * np.power(x[i], pow)
15	return sum
16	

```
def min_square(x, y, m):
    if y.shape[0] < m:
        raise Exception("Low size")

    if x.shape[0] != y.shape[0]:
        raise Exception("size error")
    n = x.shape[0]

    A = np.zeros(shape=(m+1, m+1))
    for i in range(m+1):
        for j in range(m+1):
            A[i, j] = sum_of_pow(x, i+j)

    b = np.zeros(shape=m + 1)
    for i in range(m+1):
        b[i] = sum_x_y_pows(x, y, i)

    a_vec = np.linalg.solve(A, b)
    return a_vec

def get_best_approx_poly(x, y):
    all_polys = list()
    for i in range(1, x.shape[0]):
        all_polys.append(min_square(x, y, i))
    return all_polys
```