Национальный исследовательский университет ИТМО
Факультет ПИиКТ

Лабораторная работа №4

по курсу «*Вычислительная математика*»
Вариант «Метод Лагранжа»

Работу выполнил:
Глушков Даниил Григорьевич
Группа P3233

Преподаватель:
Перл О. В.

г. Санкт-Петербург
2022

1 Описание метода

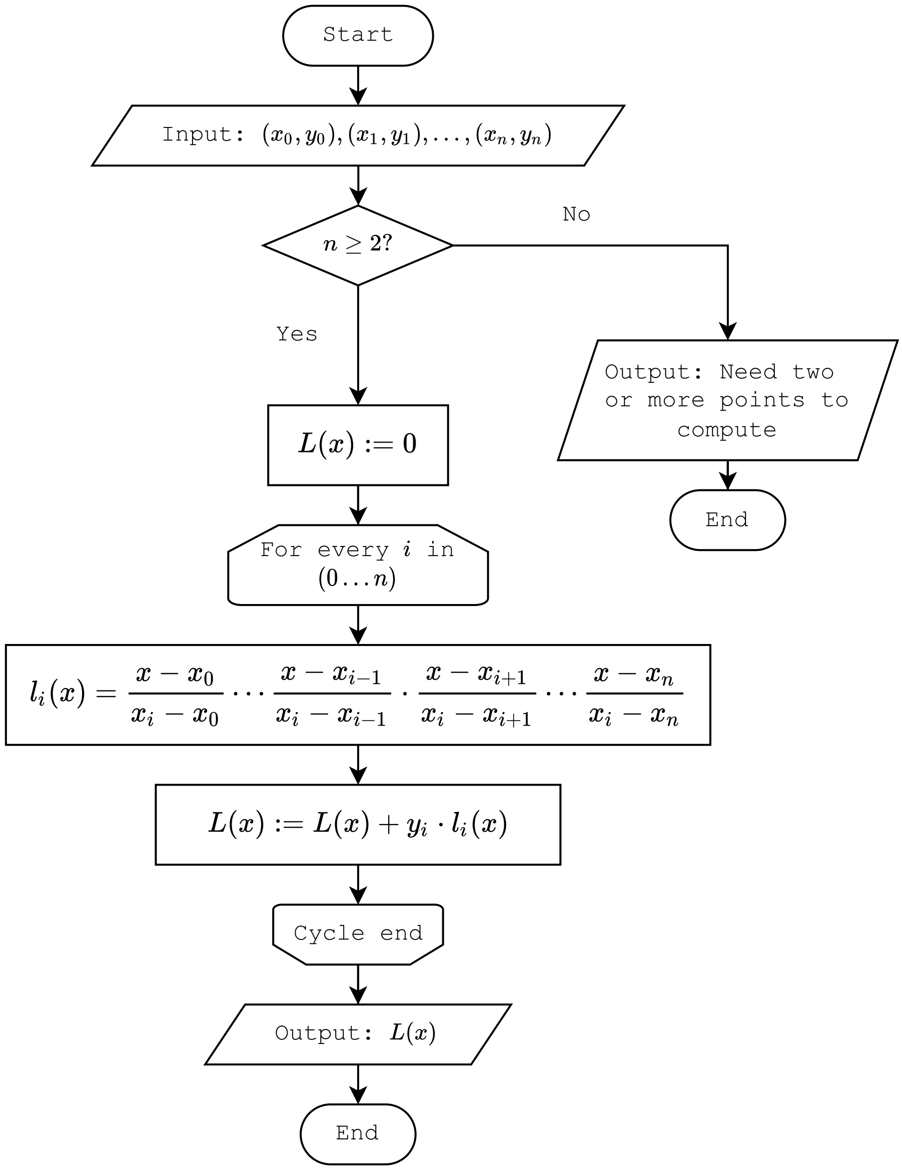
Метод представляет собой интерполяцию точек посредством построения многочлена так, чтобы он принимал нужное значение в заданном наборе точек и при этом имел степень, меньшую или равную количеству точек.

2 Расчетные формулы

$$L(x) := \sum_{j=0}^{k} y_j \ell_j(x)$$

$$\ell_j(x) := \prod_{\substack{0 \le m \le k \\ m \ne j}} \frac{x - x_m}{x_j - x_m}$$

3 Блок-схема



4 Листинг реализации

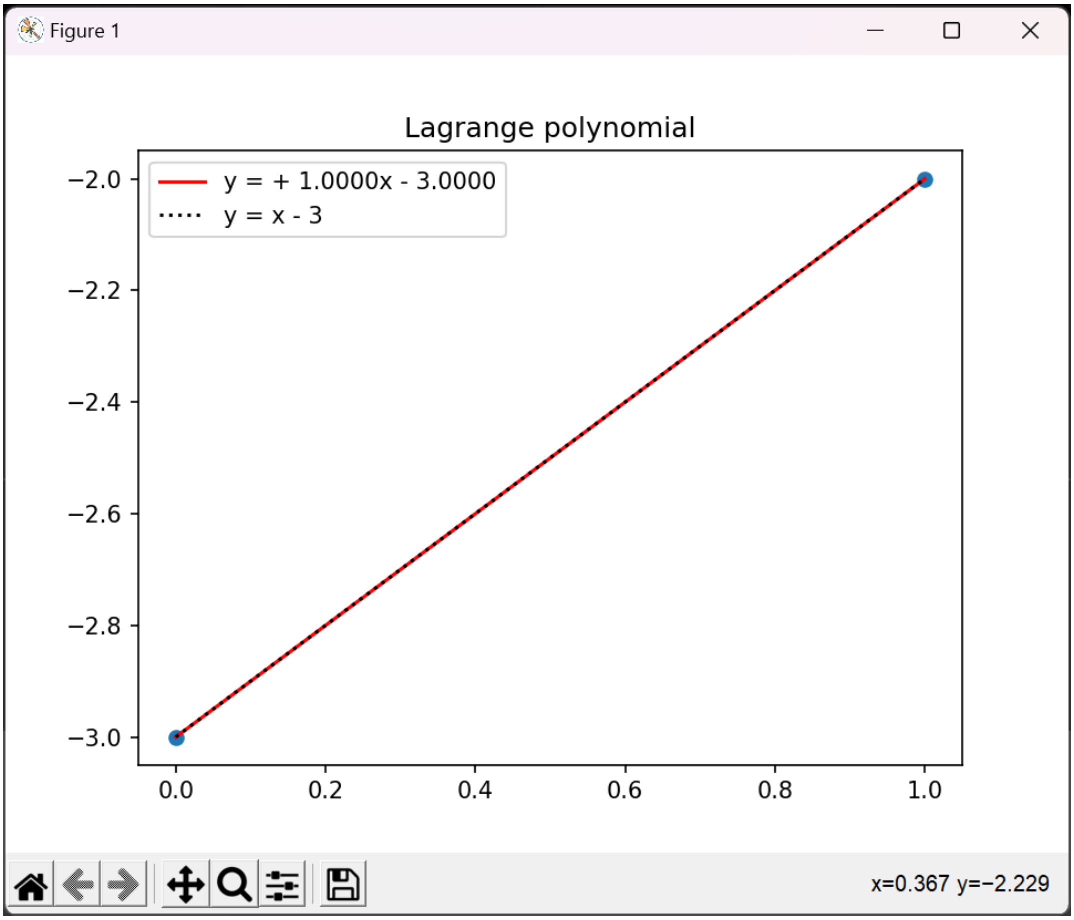
Полный код лабораторной доступен на:
https://github.com/mrcat-pixel/compmath4.git

```
def add_arrs(arr1, arr2):
    return list(map(op.add, arr1, arr2))

def multiply_arr(arr, coef):
    return [a * coef for a in arr]

# multiplies polynomial by ax + b, takes the polynomial coefs and (a, b)
def compute_multi(arr, pair):
    if len(arr) == 0:                               # if empty:
        return [pair[0], pair[1]]                   #   return ax+b

    multi1 = multiply_arr(arr, pair[0]) + [0]       # multiplies poly by ax
    multi2 = [0] + multiply_arr(arr, pair[1])       # multiplies poly by b

    return add_arrs(multi1, multi2)                 # adds two together

# builds the basis polynomial, takes the list of points and i
def generate_basis_poly(lst, i):
    ret = []                                        # coef array
    xi = lst[i][0]
    for j in [x for x in range(len(lst)) if x != i]:
        xj = lst[j][0]
        ret = compute_multi(ret, (1/(xi-xj), -xj/(xi-xj)))
    return ret

# basis polynomial sum, takes the list of points
def generate_polynomial(lst):
    ret = generate_basis_poly(lst, 0)               # gen the polynomial
    ret = multiply_arr(ret, lst[0][1])              # multiply the first basis polynomial
    for i in range(1, len(lst)):

        newarr = generate_basis_poly(lst, i)        # gen the polynomial
        newarr = multiply_arr(newarr, lst[i][1])    # multiply the basis polynomial
        ret = add_arrs(ret, newarr)                 # adds the basis polynomial to the result

    return ret
```

5 Результаты работы программы

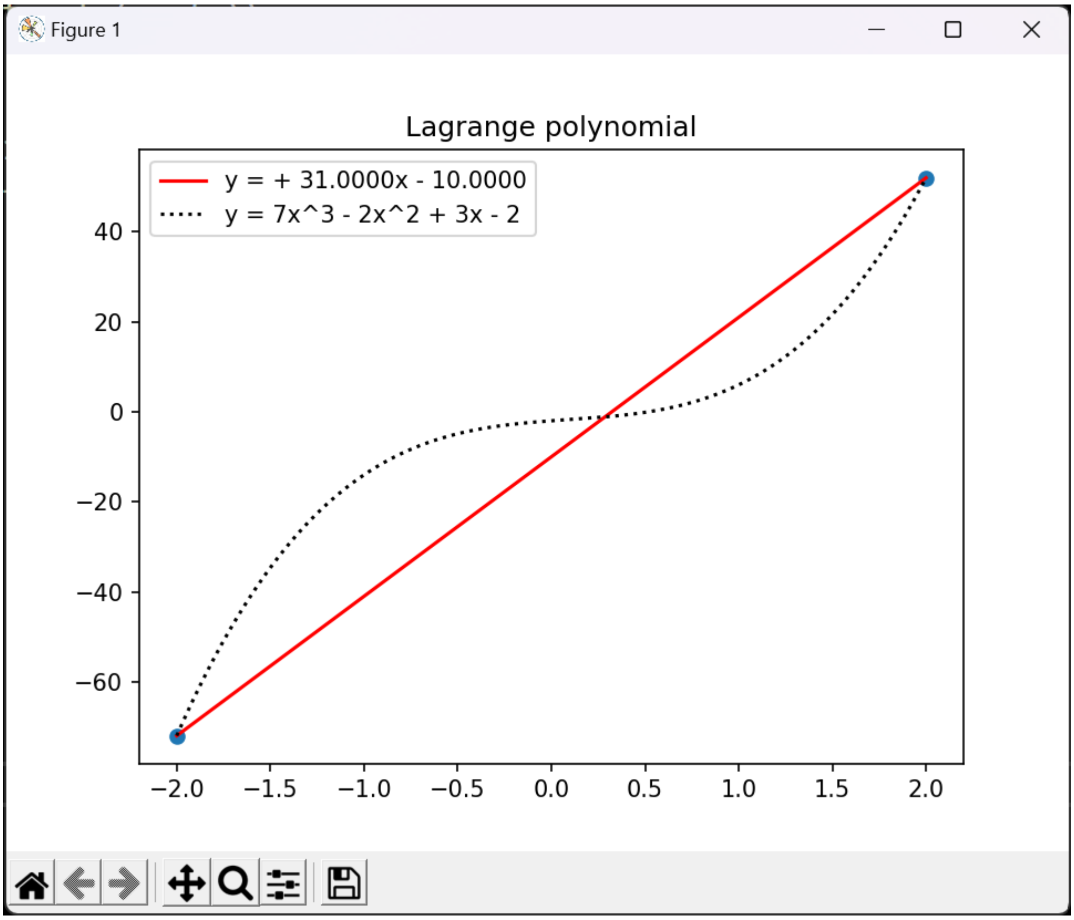
5.1 Пример 1: линейная функция

```
>d
>0 -3
>1 -2
>o 1
Added y = x - 3 to the graph.
>c
--------------------------------
The polynomial formula is:
y = + 1.0000x - 3.0000
--------------------------------
```



5.2 Пример 2: кубическая функция

```
>o 3
Added y = 7x^3 - 2x^2 + 3x - 2 to the graph.
>2 52
>-2 -72
>c
--------------------------------
The polynomial formula is:
y = + 31.0000x - 10.0000
--------------------------------
>0 -2
>c
--------------------------------
The polynomial formula is:
y = - 2.0000x^2 + 31.0000x - 2.0000
--------------------------------
>1 6
>c
--------------------------------
The polynomial formula is:
y = + 7.0000x^3 - 2.0000x^2 + 3.0000x - 2.0000
--------------------------------
```



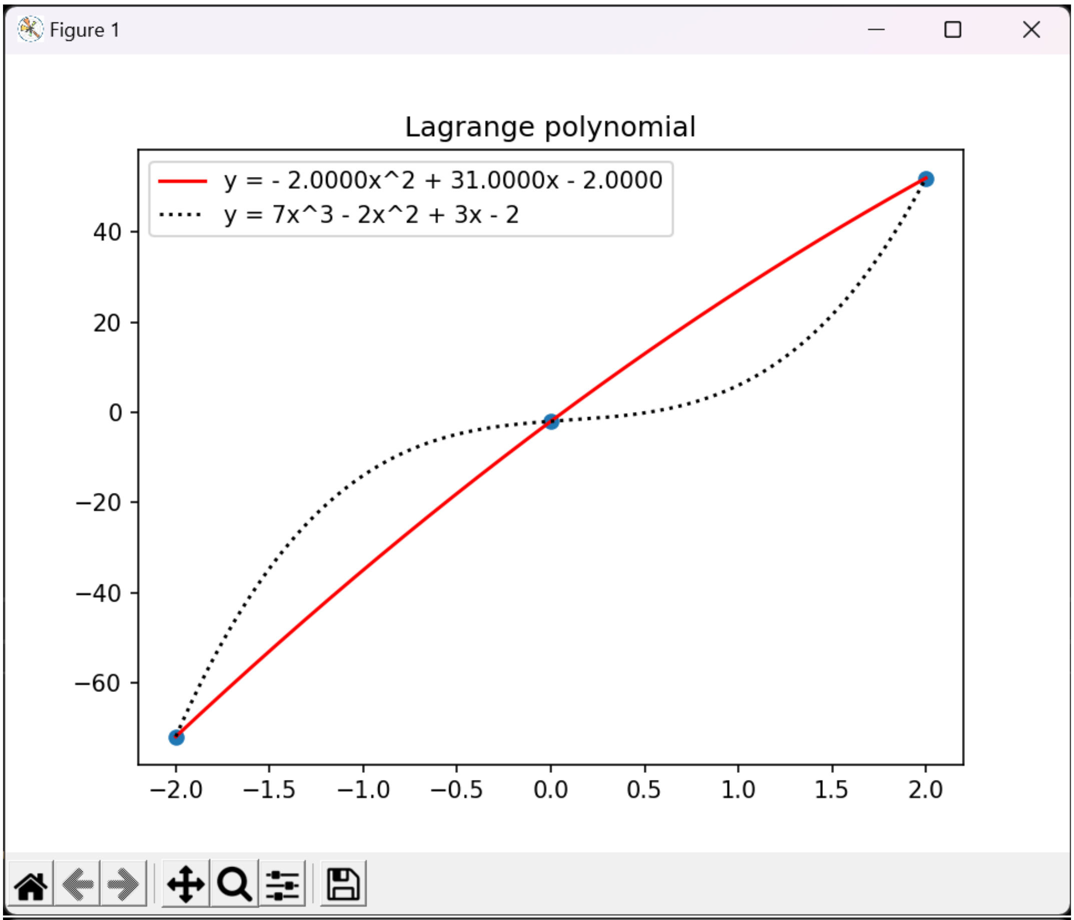


Figure 1
Lagrange polynomial
y = - 2.0000x^2 + 31.0000x - 2.0000
y = 7x^3 - 2x^2 + 3x - 2

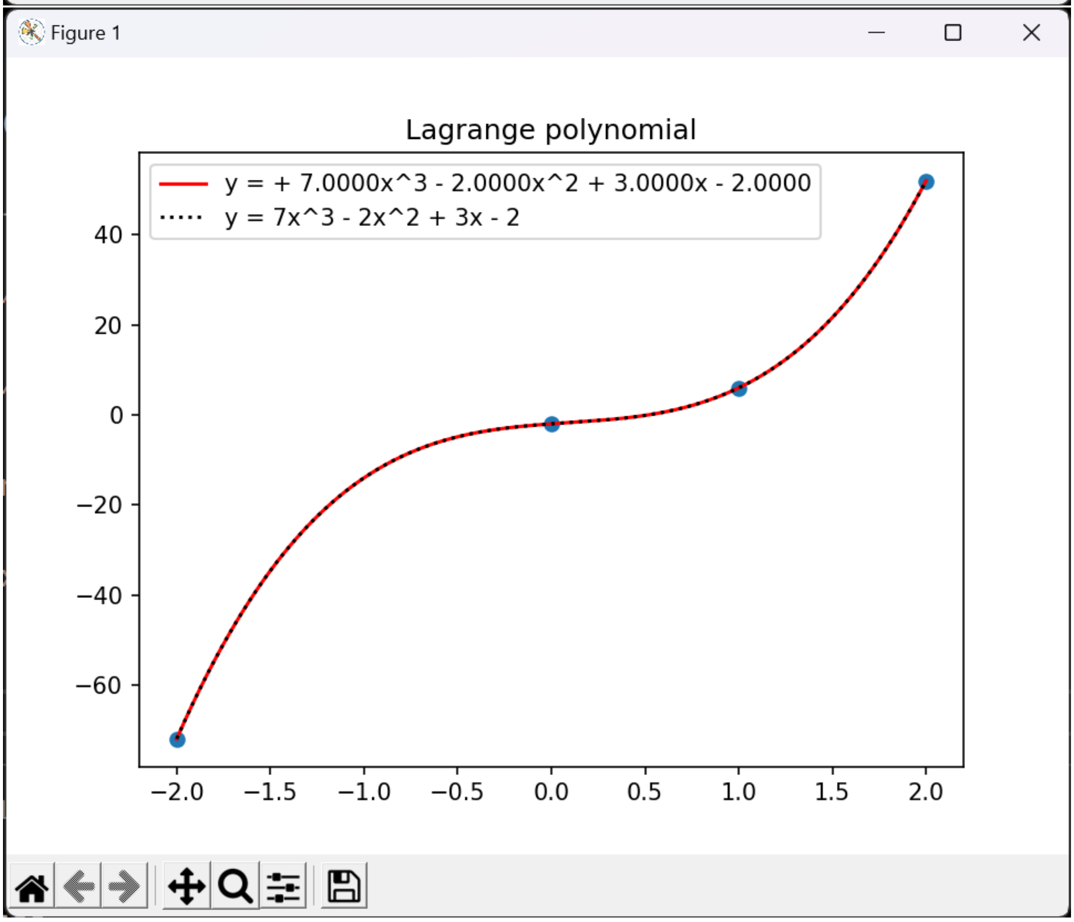


Figure 1
Lagrange polynomial
y = + 7.0000x^3 - 2.0000x^2 + 3.0000x - 2.0000
y = 7x^3 - 2x^2 + 3x - 2

5.3 Пример 3: полином вычислить невозможно

```
>0 1
>0 2
>c
The Lagrange polynomial is impossible to compute for this set of points.
>
```

5.4 Пример 4: синусоида

```
>o 4
Added y = sin(x) to the graph.
>-5 0.9589242747
>5 -0.9589242747
>c
--------------------------------
The polynomial formula is:
y = - 0.1918x + 0.0000
--------------------------------
>1 0.8414709848
>c
--------------------------------
The polynomial formula is:
y = - 0.0431x^2 - 0.1918x + 1.0763
--------------------------------
>-1 -0.8414709848
>c
--------------------------------
The polynomial formula is:
y = - 0.0431x^3 + 0.0000x^2 + 0.8845x + 0.0000
--------------------------------
>-3 -0.1411200081
>3 0.1411200081
>c
--------------------------------
The polynomial formula is:
y = + 0.0035x^5 + 0.0000x^4 - 0.1345x^3 + 0.0000x^2 + 0.9724x + 0.0000
--------------------------------
>4 -0.7568024953
>-4 0.7568024953
>c
--------------------------------
The polynomial formula is:
y = - 0.0001x^7 + 0.0000x^6 + 0.0068x^5 - 0.0000x^4 - 0.1591x^3 + 0.0000x^2 + 0.9938x + 0.0000
--------------------------------
>
```

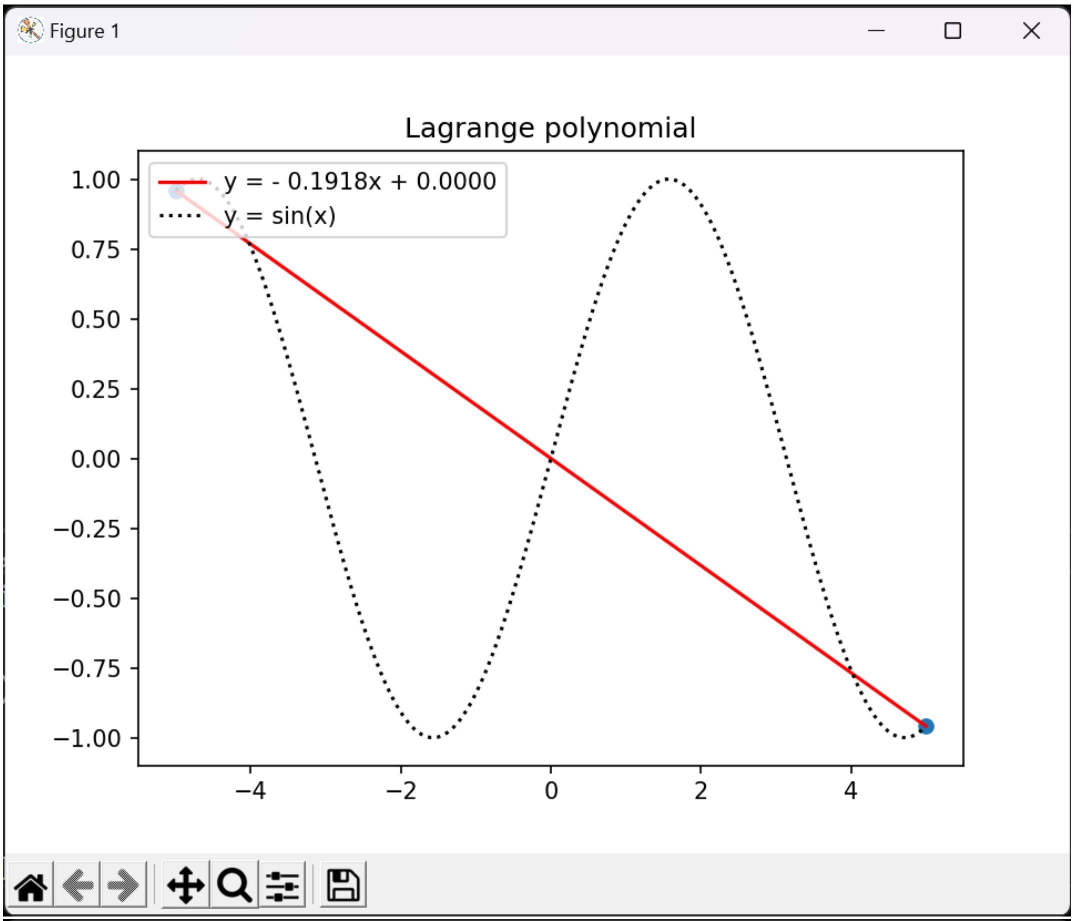


Figure 1
Lagrange polynomial
y = - 0.1918x + 0.0000
y = sin(x)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−4
−2
0
2
4

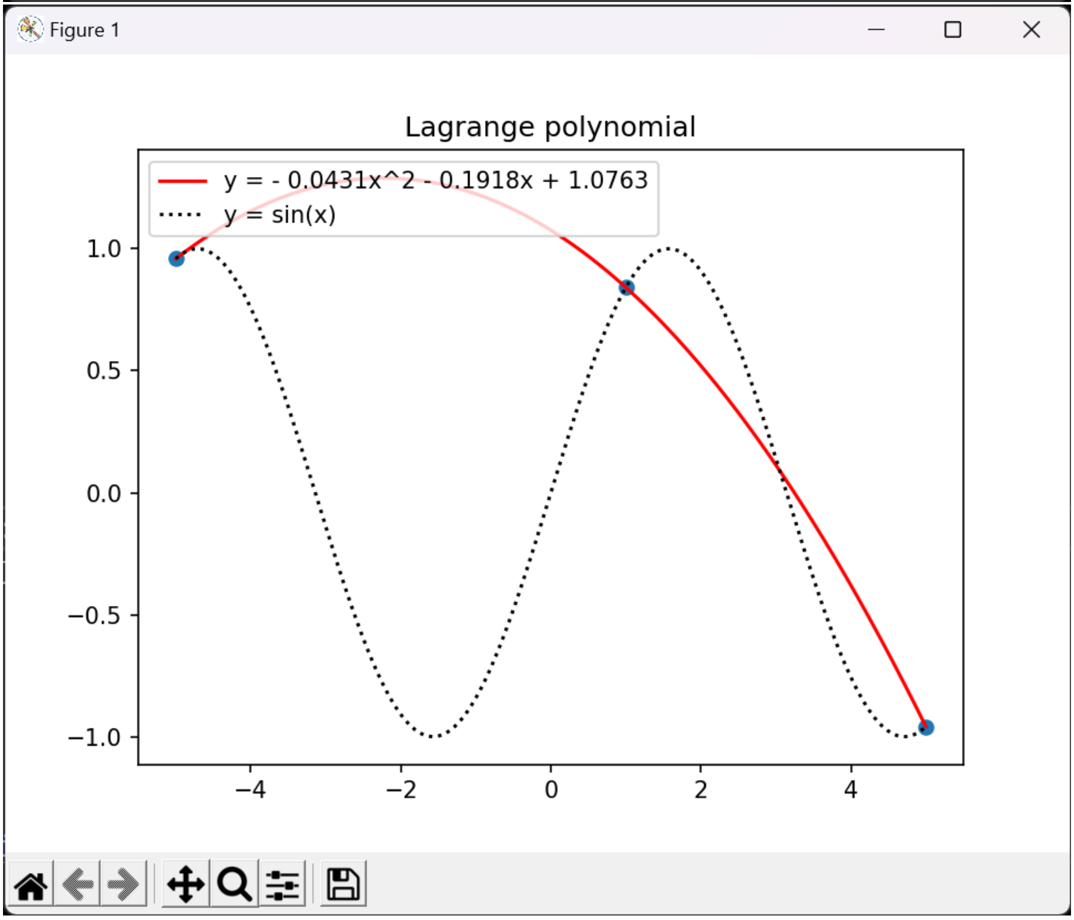


Figure 1
Lagrange polynomial
y = - 0.0431x^2 - 0.1918x + 1.0763
y = sin(x)
1.0
0.5
0.0
−0.5
−1.0
−4
−2
0
2
4

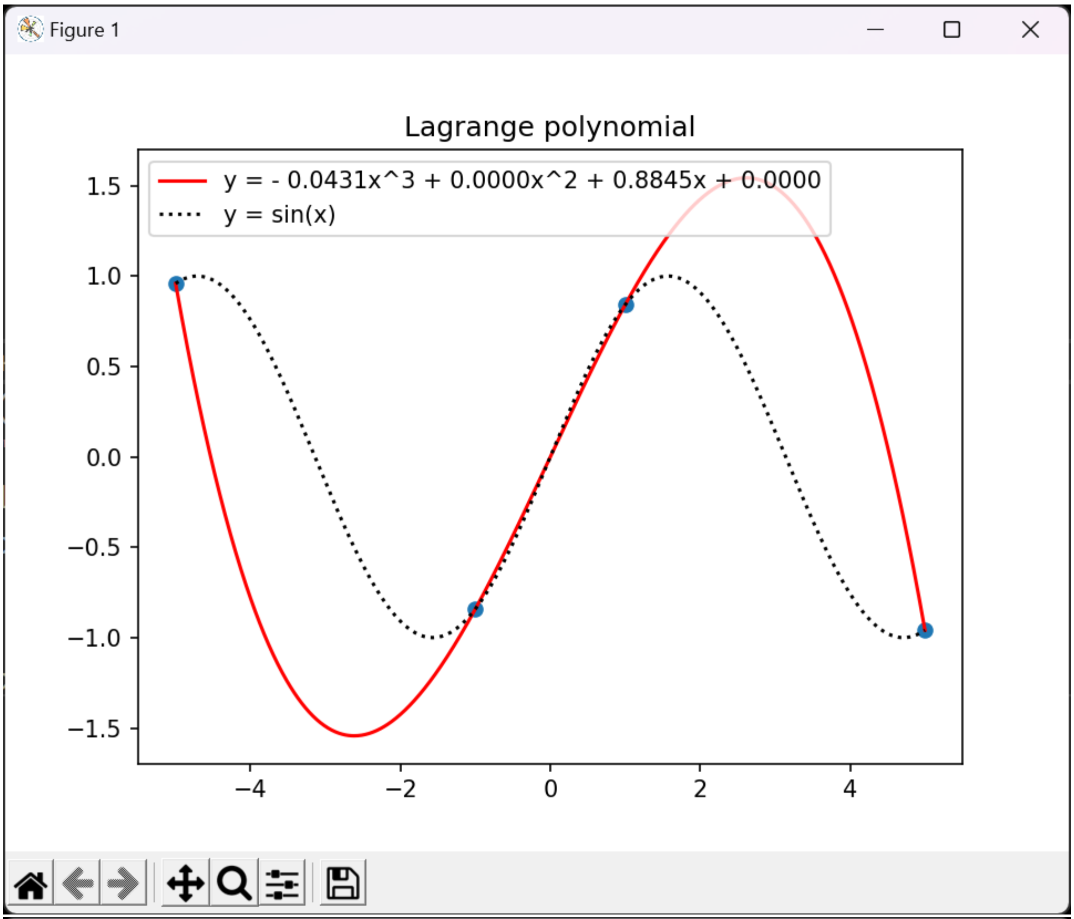


Figure 1
Lagrange polynomial
y = - 0.0431x^3 + 0.0000x^2 + 0.8845x + 0.0000
y = sin(x)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−4
−2
0
2
4

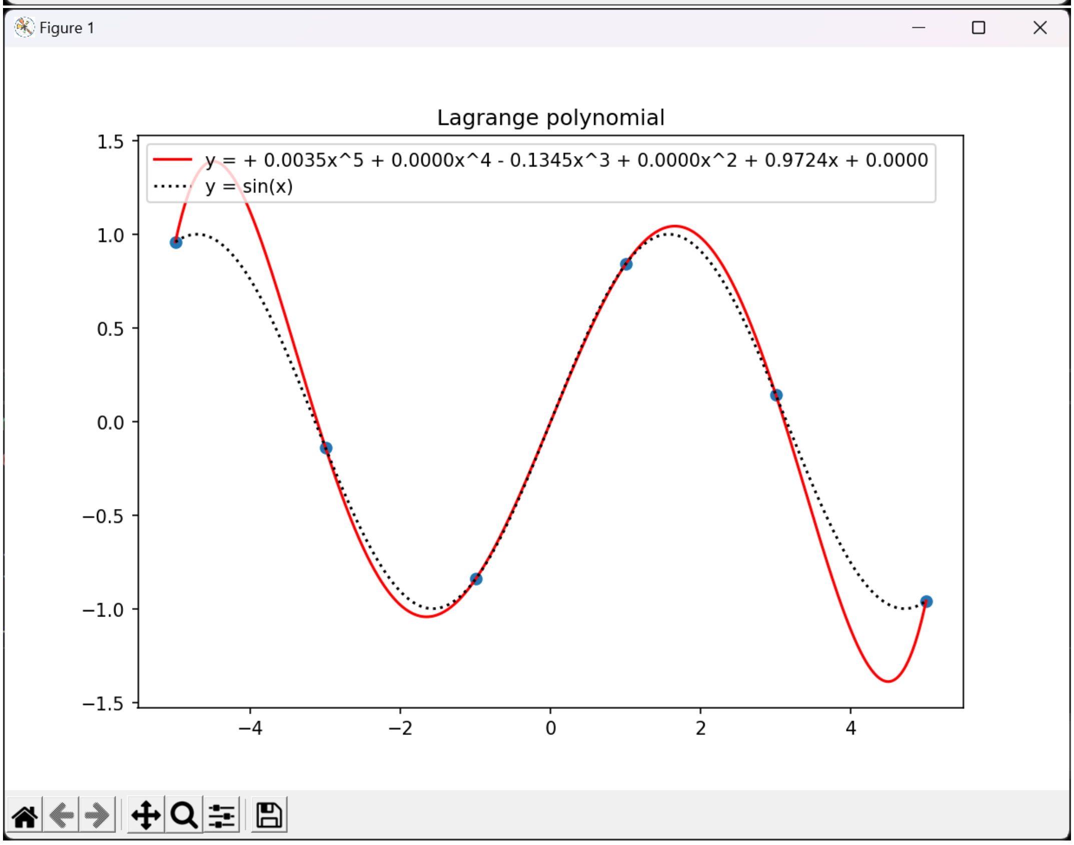
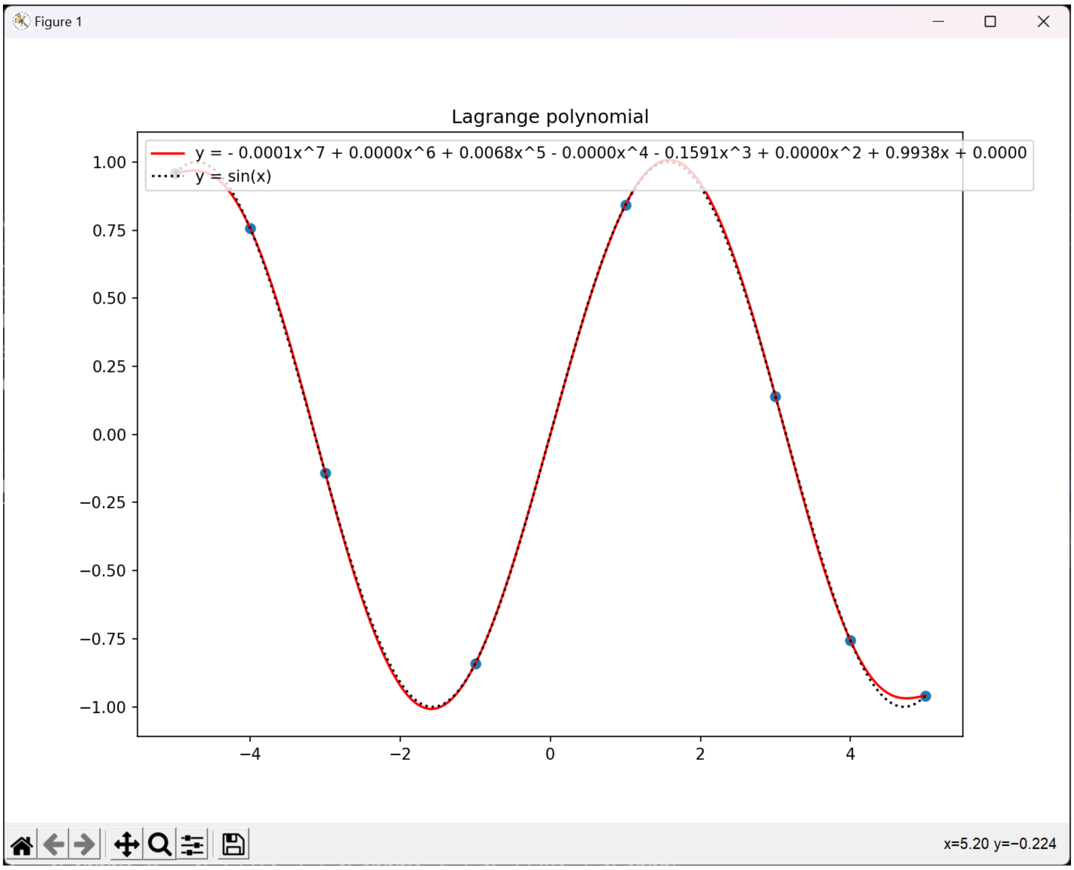
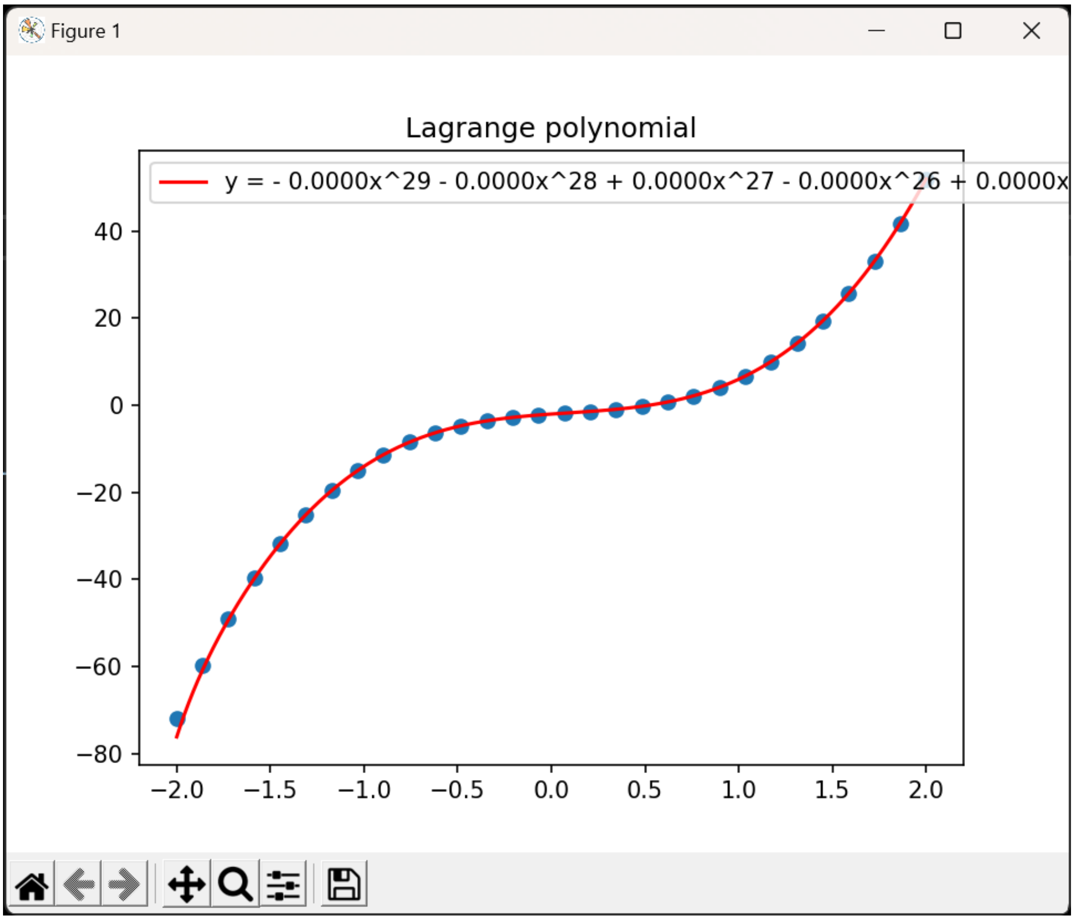


Figure 1
Lagrange polynomial
y = + 0.0035x^5 + 0.0000x^4 - 0.1345x^3 + 0.0000x^2 + 0.9724x + 0.0000
y = sin(x)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−4
−2
0
2
4



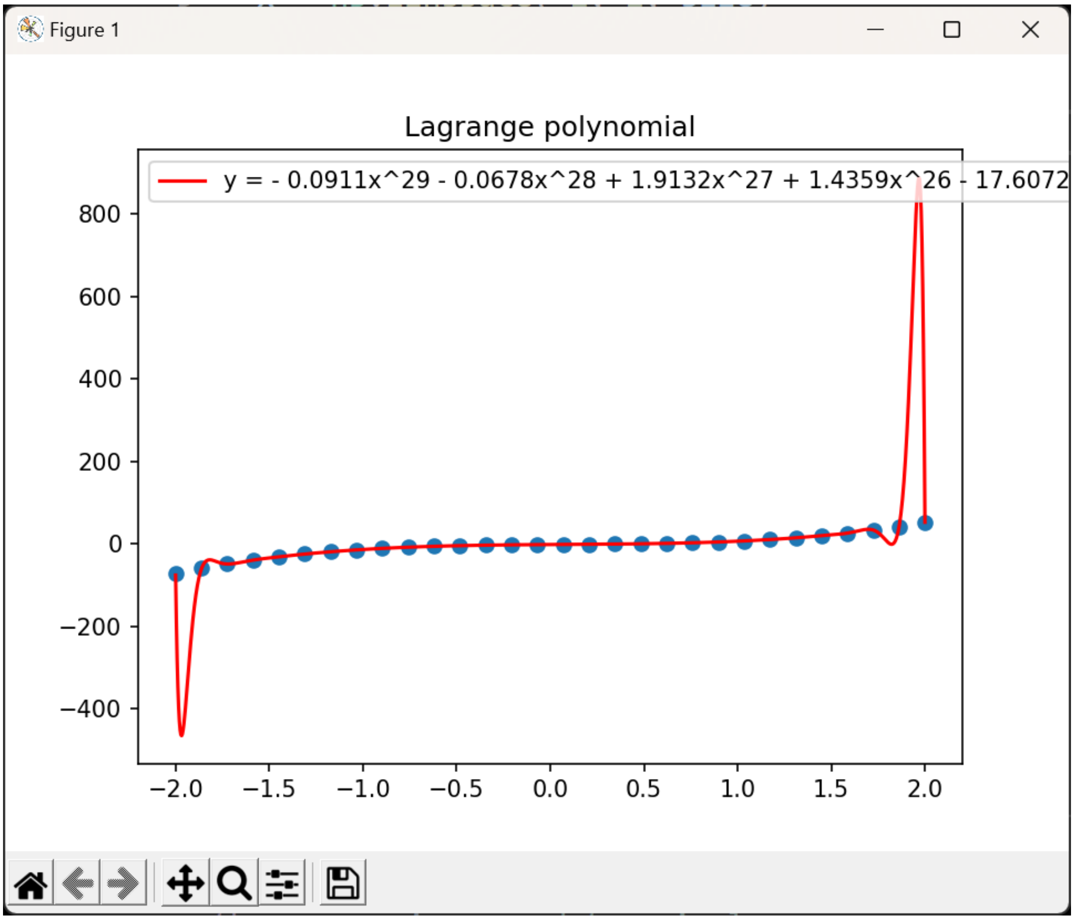
5.5 Пример 5: шум

5.5.1 Без добавления шума



5.5.2 С добавлением шума

(Шум сделан добавлением случайной величины в пределах ± 0.001 *к значению у каждой точки)*



6 Вывод

1. Метод Лагранжа целесообразно использовать, когда нужно восстановить функцию по малому числу точек с малой погрешностью, и нет необходимости добавлять точки после построения полинома.

2. При использовании большого числа точек с увеличением шума по краям интервала начинает очень сильно возрастать ошибка, поэтому в этом случае в зависимости от задачи лучше использовать метод сплайнов или аппроксимацию.

3. При добавлении точек вычисление полинома методом Лагранжа становится неэффективным, и лучше использовать метод Ньютона.