Численные методы, ЛР-4

6.1.8

Задание

Задача 6.1. Функция *y=f(x)* задана таблицей значений $y_0, y_1, \dots y_n$ в точках $x_0, x_1, \dots x_n$. Используя метод наименьших квадратов (МНК), найти многочлен $Pm(x) = a_0 + a_1 x + \dots + a_m x^m$ наилучшего среднеквадратичного приближения оптимальной степени *m=m**. За оптимальное значение *m** принять ту степень многочлена, начиная с которой величина $\sigma_m = \sqrt{\frac{1}{n-m}\sum_{k=0}^{n}(P_m(x_k) - y_k)^2}$ стабилизируется или начинает возрастать.

ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ:

1. Задать векторы *x* и *y* исходных данных.
2. Написать программу-функцию **mnk**, найти с ее помощью многочлены *Pm*, *m*=0,1,2,..., по методу наименьших квадратов. Вычислить соответствующие им значения σ_m.
3. Построить гистограмму зависимости σ_m от *m,* на основании которой выбрать оптимальную степень *m** многочлена наилучшего среднеквадратичного приближения.
4. На одном чертеже построить графики многочленов *Pm*, *m*=0,1,2,..., *m**, и точечный график исходной функции.

Вариант

6.1.8	
0	1.019
0.3	1.4889
0.6	2.2079
0.9	3.0548
1.2	3.8648
1.5	4.2161
1.8	5.1180
2.1	5.7661
2.4	6.6720
2.7	7.1960
3	7.8551

Решение

```
# задаем точки по условию
x = np.array([0.0, 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.1, 2.4, 2.7,
y = np.array([1.019, 1.4889, 2.2079, 3.0548, 3.8648, 4.2161, 5.:
n = len(x)
```

```
# считаем сигмы для определения лучшего m
sigmas = []  # массив с разными сигмами
functions = []  # массив функция для вычисления полиномов
coeffs = []  # коэффициенты для полиновом
for m in range(n):
    f, coeff = mnk(x, y, m)
    sigma = np.sqrt((1 / (n - m)) * np.sum((f(x) - y) ** 2))
    sigmas.append(sigma)
    functions.append(f)
    coeffs.append(coeff)
```

Функция МНК:

```
import numpy as np

def mnk(x: np.array, y: np.array, higher_degree=1):
    m = higher_degree + 1

    b = np.empty(m)  # b
    g = np.empty((m, m))  # Γ

    # заполняем b = (P^T)*y
    for j in range(m):
        b[j] = np.sum(y * x ** j)

    # заполняем Γ = (P^T)*P
    for j in range(m):
        for k in range(m):
```

```
            g[j][k] = sum(x ** (k + j))

    best_coeffs = np.linalg.solve(g, b)

    def f(px):
        py = 0
        for i in range(m):
            py += best_coeffs[i] * px ** i
        return py

    return f, best_coeffs
```

Результат

График сигм от m

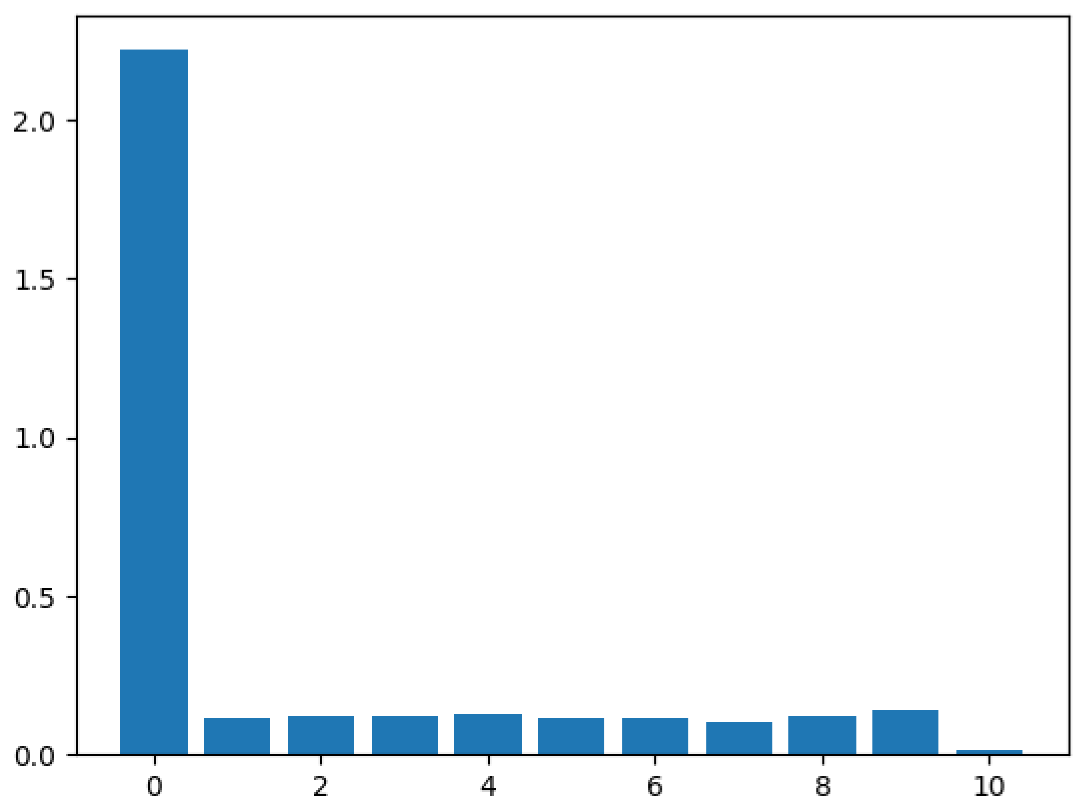
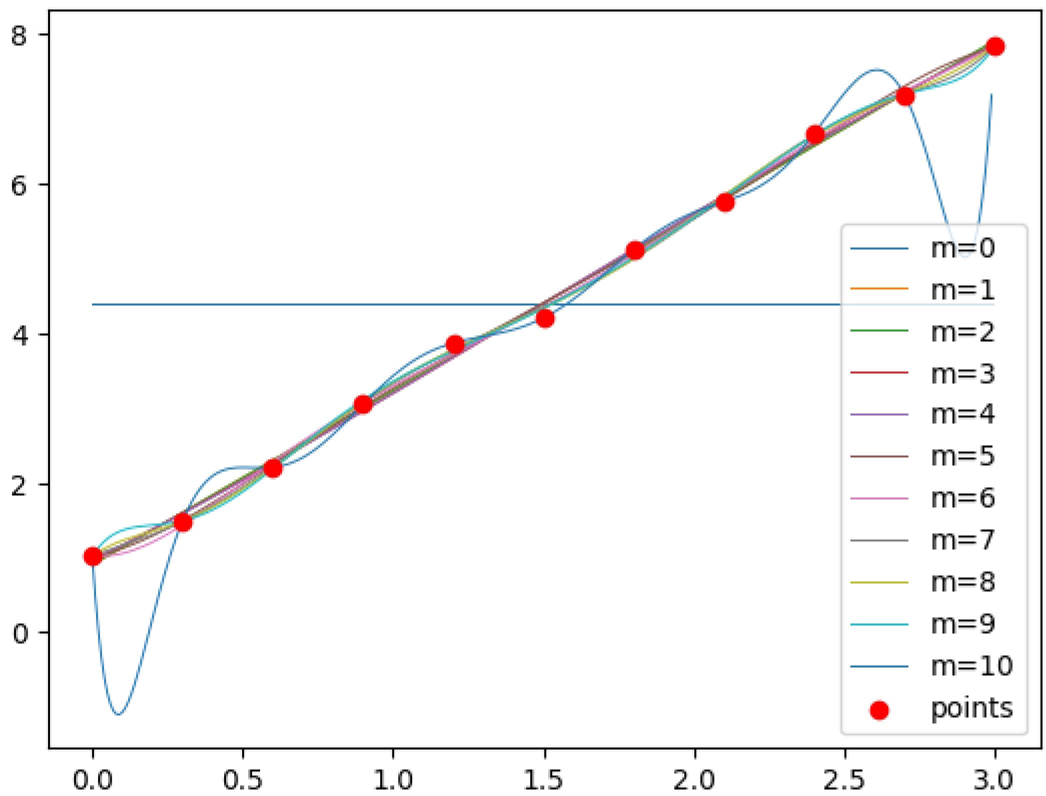


График полученной функции при разных m



6.2.2

Задание

Задача 6.2. В таблице приведены результаты наблюдений за перемещением x материальной точки по оси Ox в моменты времени $t \in [t_0, T]$. Известно, что движение является равномерным и описывается линейной зависимостью $x(t)=vt+b$. Используя метод наименьших квадратов, определить скорость v и спрогнозировать положение точки в момент времени $t=2T$. На одном чертеже построить график движения точки и точечный график исходных наблюдений.

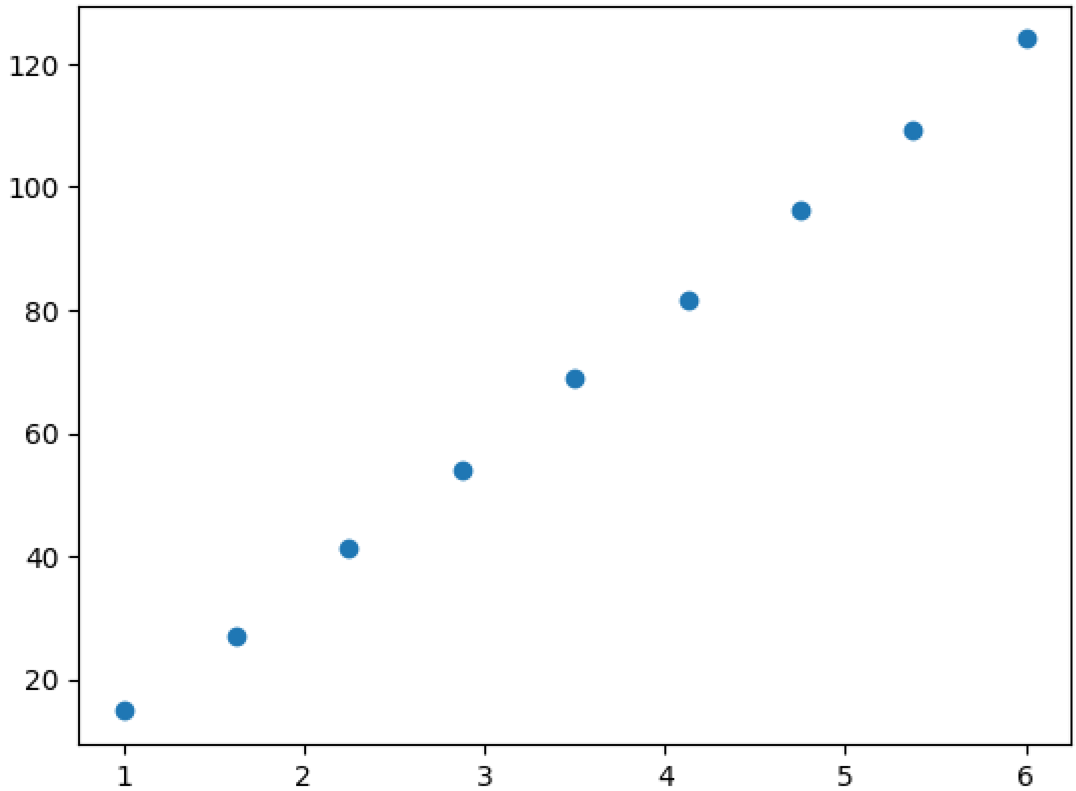
Вариант

6.2.2	t	1	1.625	2.25	2.88	3.5	4.13	4.75	5.375	6
	x	14.86	27.15	41.19	54	69.03	81.6	96.11	109.4	124.03

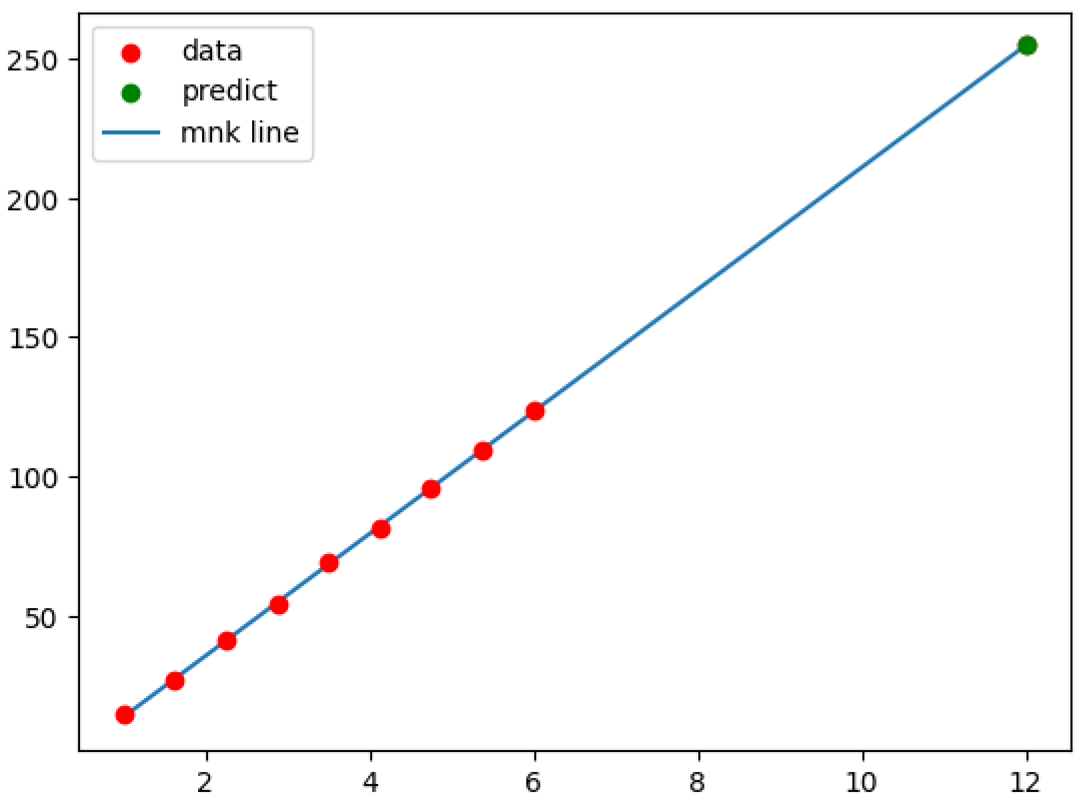
Решение

```
t = [1.0, 1.625, 2.25, 2.88, 3.5, 4.13, 4.75, 5.375, 6]
x = [14.86, 27.15, 41.19, 54, 69.03, 81.6, 96.11, 109.4, 124.03]
plt.scatter(t, x)

f, coeffs = mnk(np.array(t), np.array(x), higher_degree=1)
b, v = coeffs

T2 = 2 * t[-1]
t.append(T2)
x.append(f(T2))

plt.scatter(t, x, c='r', label='data', zorder=10)
plt.scatter(T2, f(T2), c='g', label='predict', zorder=10)
plt.plot(t, f(np.array(t)), label='mnk line')
plt.legend()
```

Результат



120
100
80
60
40
20
1
2
3
4
5
6



data
predict
mnk line
250
200
150
100
50
2
4
6
8
10
12

6.7.4

Задание

Задача 6.7. Дана кусочно-гладкая функция *y=f(x)*. Сравнить качество приближения функции кусочно-линейной и глобальной интерполяциями.

ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ:

1. Вычислить значения функции $y_i = f(x_i)$ в произвольных точках x_i, *i=0,1,..., k-1*, отрезка [*a, b*], по которым будет осуществляться интерполяция функции.
2. Составить программу-функцию, вычисляющую значение интерполяционного многочлена 1-ой степени по точкам (x_i, y_i) и (x_{i+1}, y_{i+1}) в произвольной точке отрезка $[x_i, x_{i+1}]$. С ее помощью вычислить приближенные значения функции *f(x)* при кусочно-линейной интерполяции в *3k* точках исходного отрезка [*a,b*].
3. Написать функцию **inter**, возвращающую значение интерполяционного многочлена в форме Ньютона (с разделенными разностями). Вычислить приближенные значения функции *f(x)* в тех же *3k* точках отрезка при глобальной интерполяции, используя написанную функцию **inter**. На одном чертеже построить графики интерполирующих функций, график исходной функции *f(x)*, а также отметить точки (x_i, y_i), *i=0,1,...,k-1*, по которым осуществлялась интерполяция.
4. Вычислить практическую величину погрешностей Δ_j, *j=0,1,...,3k-1*, приближения функции *f(x)* в *3k* точках для кусочно-линейной и глобальной интерполяций. На одном чертеже построить графики погрешностей. Сравнить качество приближения.

Вариант

6.7.4	
$\lvert x-3\rvert \cdot (x^2+1)$	$[0,4]$

Теория

Как посчитать разделенные разности

1. Таблица разделенных разностей. Пусть функция f задана на таблице $x_0, x_1, \ldots, x_n$ значений аргумента с произвольным (не обязательно постоянным) шагом, причем точки таблицы занумерованы в произвольном (не обязательно возрастающем) порядке. Величины

$$f(x_i; x_{i+1}) = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}$$

принято называть *разделенными разностями первого порядка* функции f. *Разделенные разности второго порядка* определяются формулой

$$f(x_i; x_{i+1}; x_{i+2}) = \frac{f(x_{i+1}; x_{i+2}) - f(x_i; x_{i+1})}{x_{i+2} - x_i}.$$

Аналогично определяются разделенные разности третьего и более высоких порядков. Общее определение *разделенной разности порядка* $k \geqslant 2$ таково:

$$f(x_i, x_{i+1}, \ldots, x_{i+k}) = \frac{f(x_{i+1}; \ldots; x_{i+k}) - f(x_i; \ldots; x_{i+k-1})}{x_{i+k} - x_i}.$$

Интерполяционный многочлен в форме Нтютона с разделенными разностями

1. Интерполяционный многочлен Ньютона с разделенными разностями. Используя разделенные разности, интерполяционный многочлен можно записать в следующем виде:

$$P_n(x) = f(x_0) + f(x_0; x_1)(x - x_0) + f(x_0; x_1; x_2)(x - x_0)(x - x_1) + \ldots$$
$$\ldots + f(x_0; x_1; \ldots, x_n)(x-x_0)(x-x_1)\ldots(x-x_{n-1}) = \sum_{k=0}^{n} f(x_0; x_1; \ldots, x_k)\omega_k(x). \quad (11.52)$$

Здесь $\omega_0(x) \equiv 1$, $\omega_k(x) = (x - x_0)(x - x_1)\ldots(x - x_{k-1})$. Записанный в таком виде интерполяционный многочлен называют *интерполяционным многочленом Ньютона с разделенными разностями.*

Решение

Кусочно-линейная

```
def get_linear_interp(x1, x2, y1, y2, xp):
    """
    Функция для вычисления значения интерполяционного многочлена
    """
    if x1 <= xp <= x2:
        yp = y1 + ((y2 - y1) / (x2 - x1)) * (xp - x1)
```

```
        return yp


def get_linear_interp_all(x_list, y_list):
    """
    Функция, которая вычисляет приближенные значения функции при
    """
    k = 3
    x_list_linear = []
    y_list_linear = []
    for i in range(len(x_list) - 1):
        d = x_list[i + 1] - x_list[i]
        delta = d / k
        for j in range(k):
            x = x_list[i] + j * delta
            y = get_linear_interp(x_list[i], x_list[i + 1], y_li

            x_list_linear.append(x)
            y_list_linear.append(y)

    x_list_linear.append(x_list[-1])
    y_list_linear.append(y_list[-1])
    return x_list_linear, y_list_linear
```

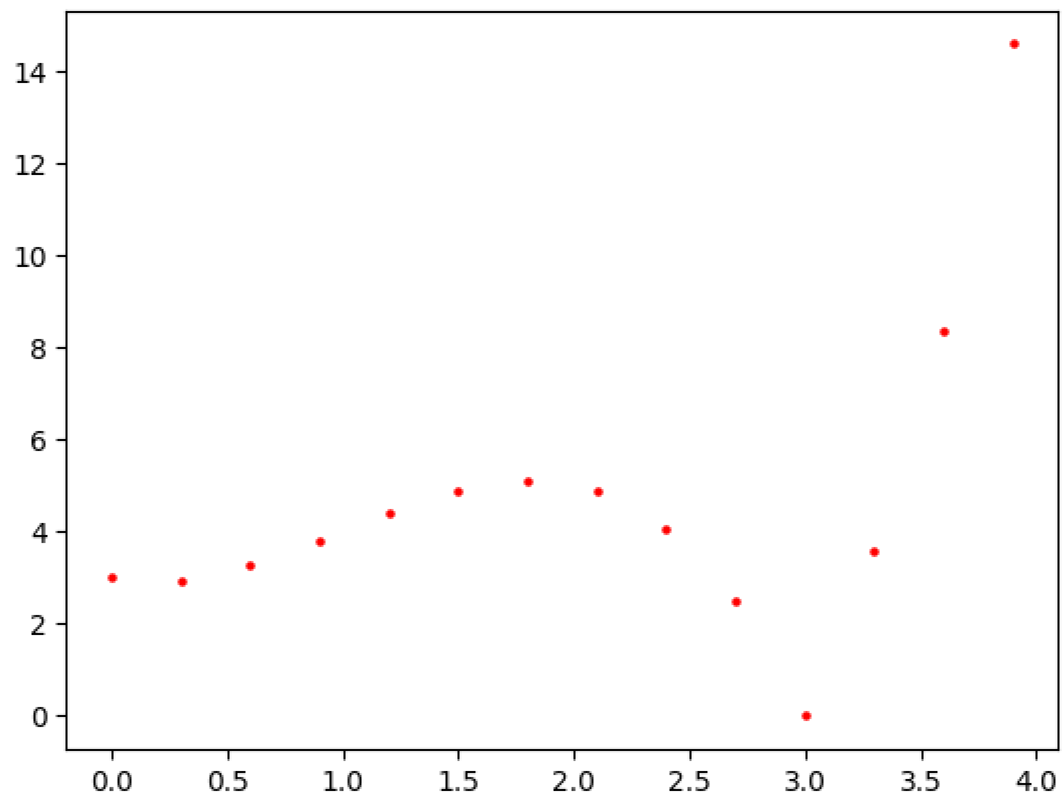
В форме Ньютона

```
from functools import cache


@cache
def calc_separated_differences_new(f, i1, i2):
    """
    Посчитать разделенные суммы от i1 до i2 включительно
    """
    k = i2 - i1
    if k == 0:
```

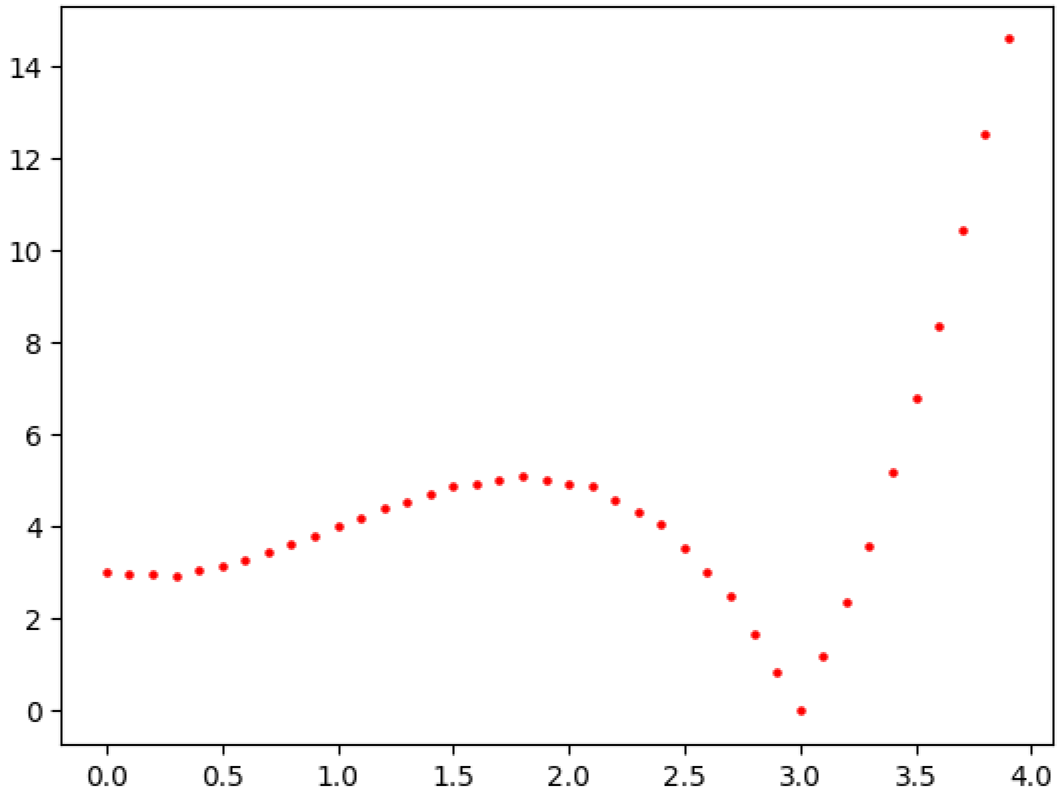
```
        assert False
    if k == 1:
        return (f(x_list[i1 + 1]) - f(x_list[i1])) / (x_list[i1
    if k >= 2:
        f1 = calc_separated_differences_new(f, i1 + 1, i2)
        f2 = calc_separated_differences_new(f, i1, i2 - 1)
        return (f1 - f2) / (x_list[i2] - x_list[i1])



def inter(f, x_list, x):
    """
    Функция для вычисления интерполяционного многочлена в форме
    """
    p = f(x_list[0])
    for k in range(1, len(x_list)):
        w = 1
        for j in range(k):
            w *= x - x_list[j]
        sep = calc_separated_differences_new(f, 0, k)
        p += sep * w
    return p
```

Результаты

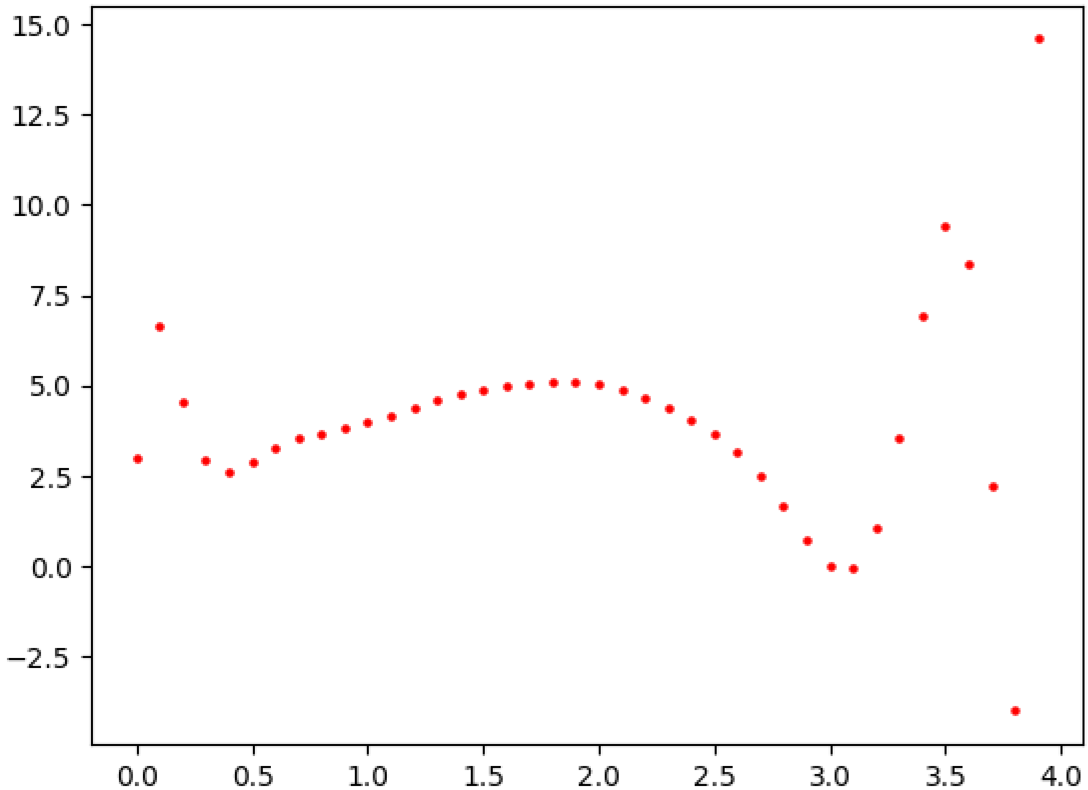
Изачальные k точек

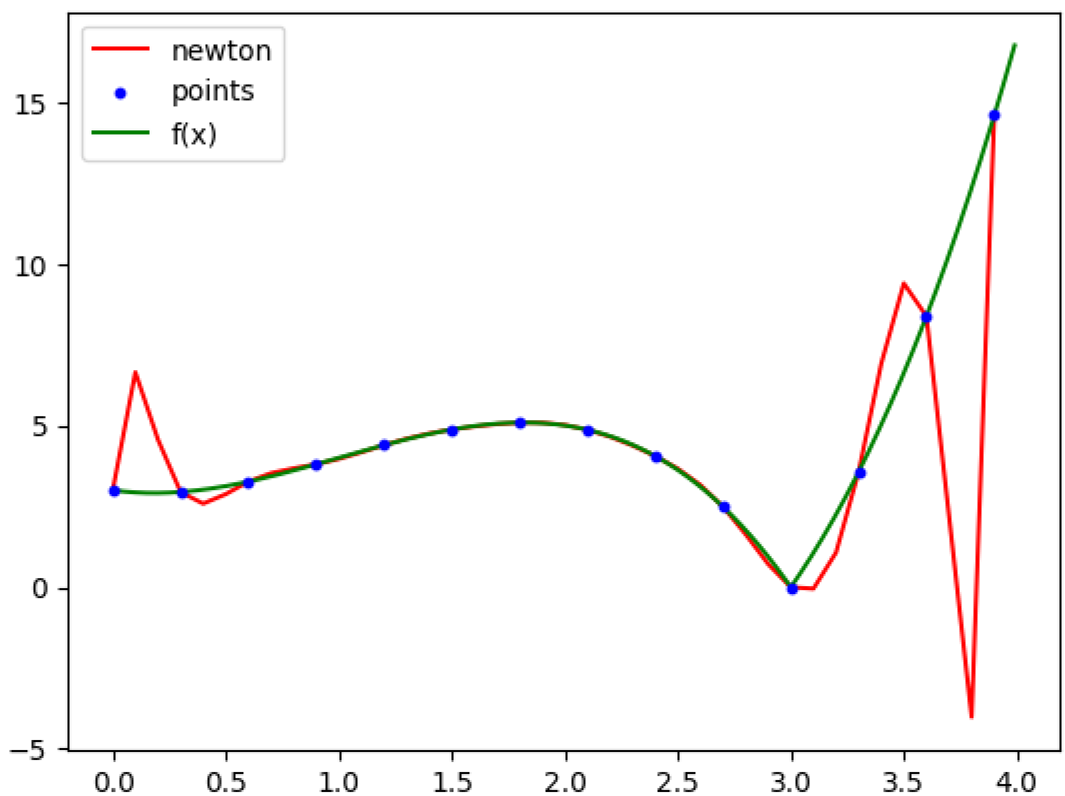


3k точек при кусочно-линейной интерполяции

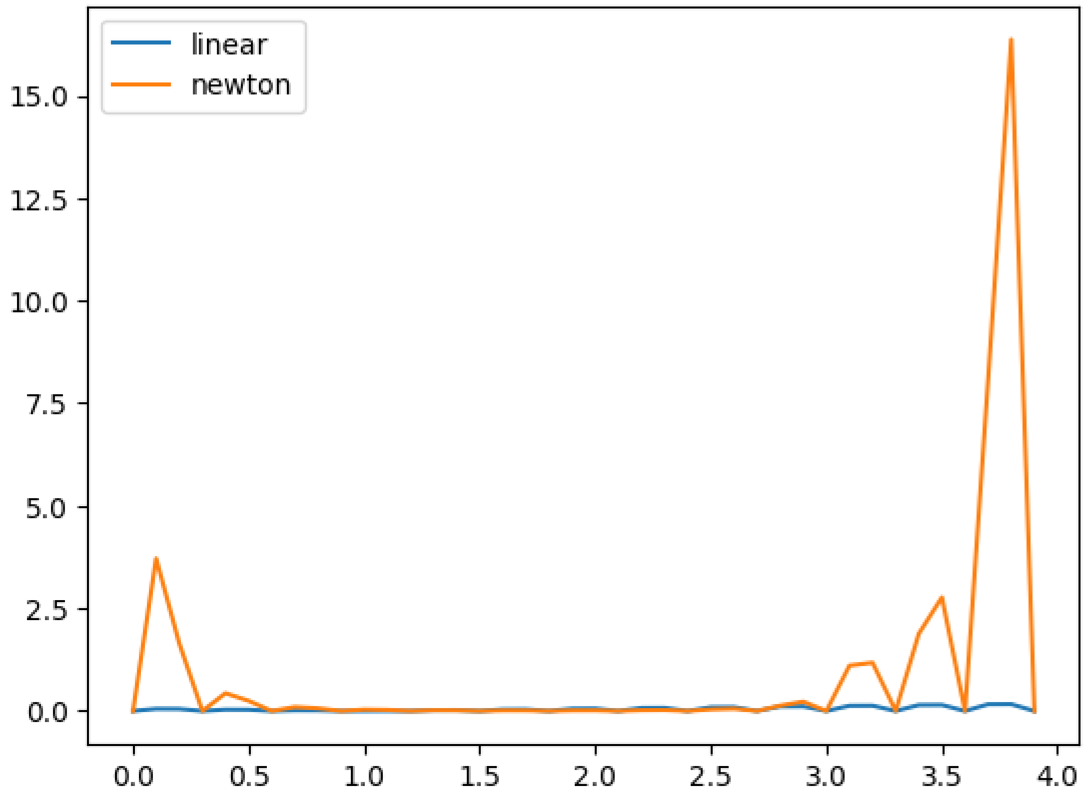


3k точек через многочлен в форме Ньютона


15.0
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


newton
points
f(x)
15
10
5
0
−5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

Сравнение ошибок



Видно, что интерполяционный многочлен в форме Нтютона хуже справляется на границах

6.8.4

Задание

Задача 6.8. Дана функция $y=f(x)$. Приблизить $f(x)$ методом глобальной интерполяции при равномерном и чебышевском распределениях узлов интерполяции. Сравнить качество приближения.

ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ:

1. Составить программу-функцию построения интерполяционного многочлена при произвольном распределении узлов (количество узлов - любое).
2. Используя составленную программу, вычислить приближенные значения функции $f(x)$в $3k$ точках исходного отрезка $[a, b]$ по k узлам интерполяции, распределенным равномерно на отрезке. На одном чертеже построить графики интерполяционного многочлена и исходной функции.
3. Используя составленную программу, вычислить приближенные значения функции $f(x)$ в тех же $3k$ точках исходного отрезка по k узлам интерполяции, имеющим чебышевское распределение. На одном чертеже построить графики интерполяционного многочлена и исходной функции.
4. Сравнить качество приближения функции $f(x)$ при разном распределении узлов.
5. Выполнить п. 2-4, строя интерполяционный многочлен по $2k$ узлам интерполяции.
6. Сравнить результаты при разном числе узлов.

Вариант

6.8.4	
$e^{\cos(3x)}$	$[0,\pi]$

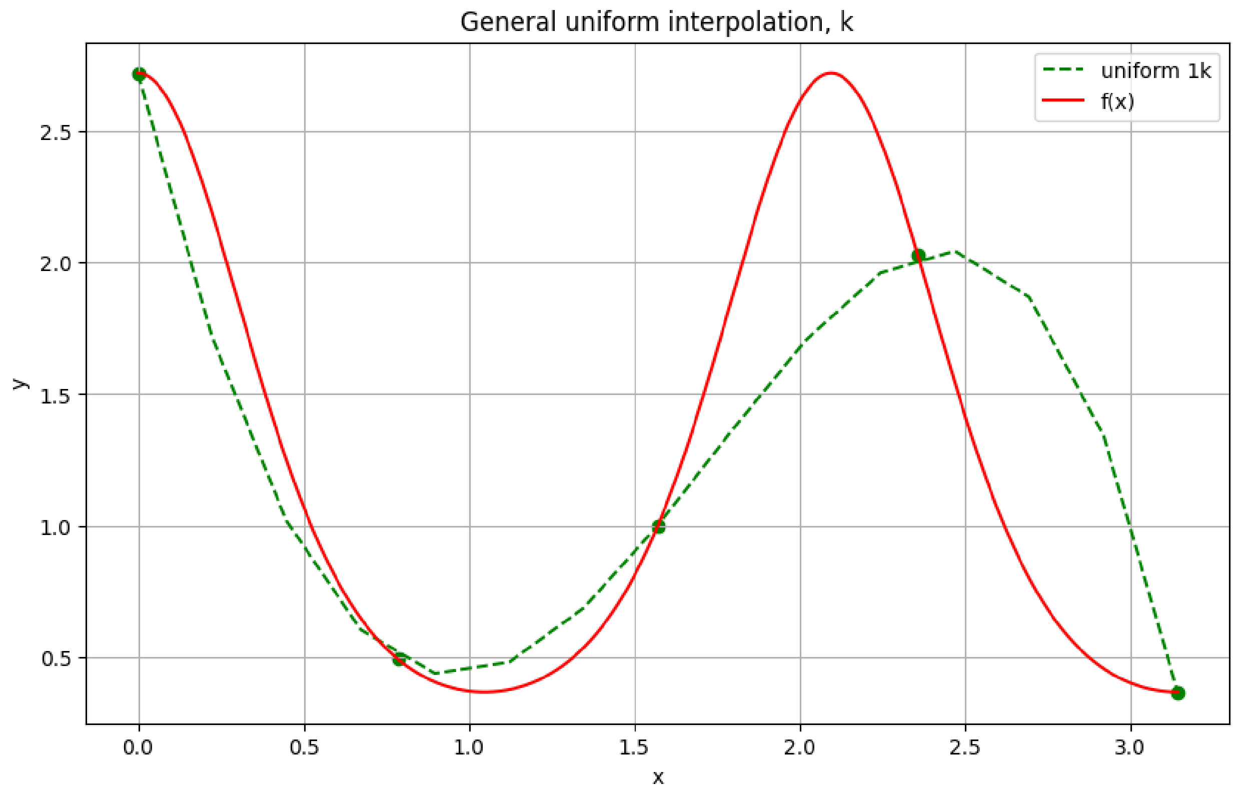
Решение

Метод глобальной интерполяции: интерполяционный многочлен в форме Нтютона с разделенными разностями

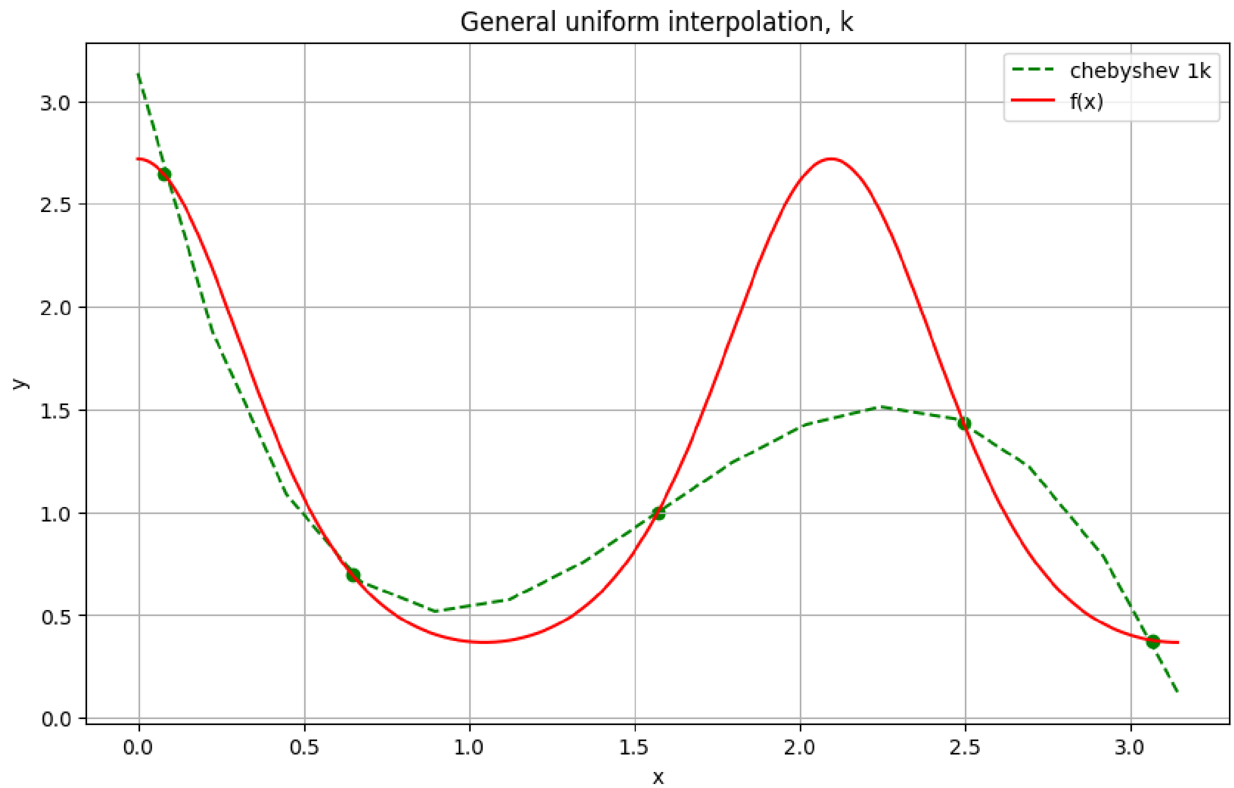
```
x_list_uniform = np.linspace(x_min, x_max, 1 * k)
x_list_chebyshev = (0.5 * (x_min + x_max) +
                    0.5 * (x_max - x_min) * np.cos(
                    (2 * np.arange(1, multiplier * k + 1) - 1) /
```

Результаты (k = 5)

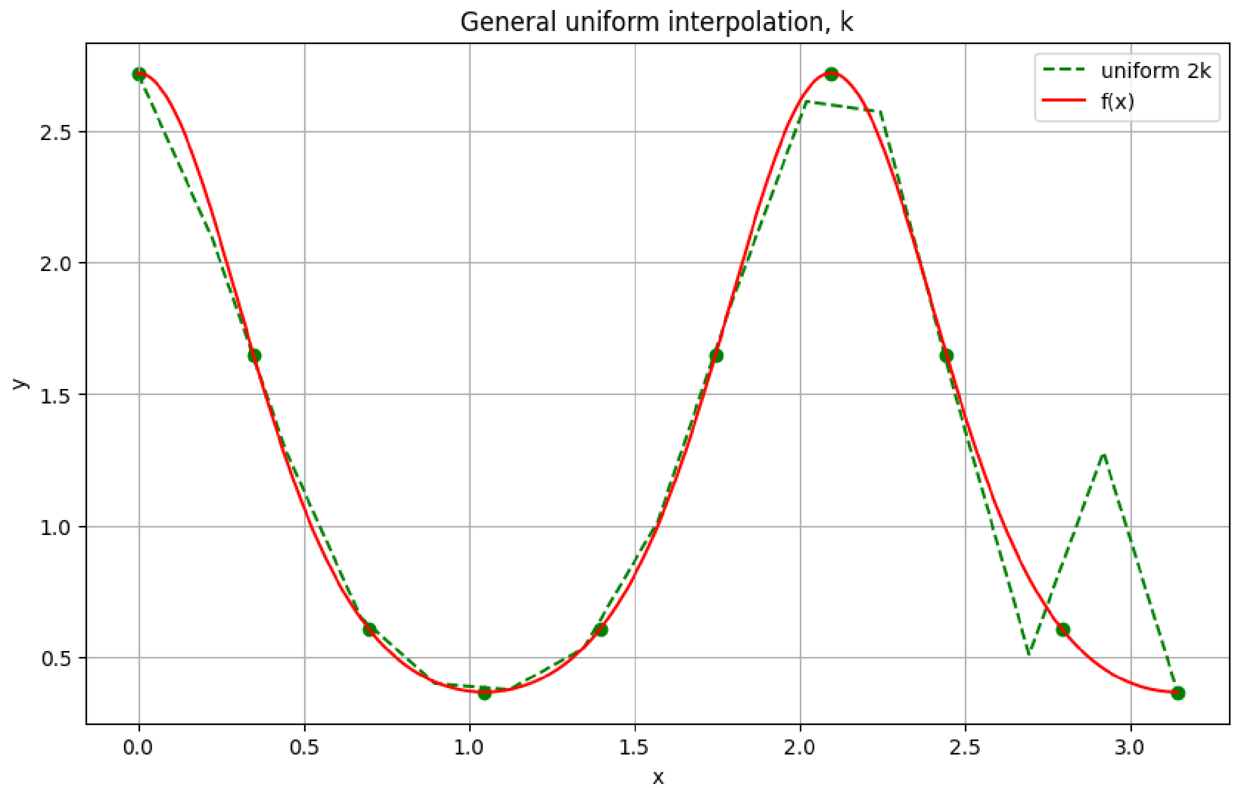
uniform при разбиении на k точек



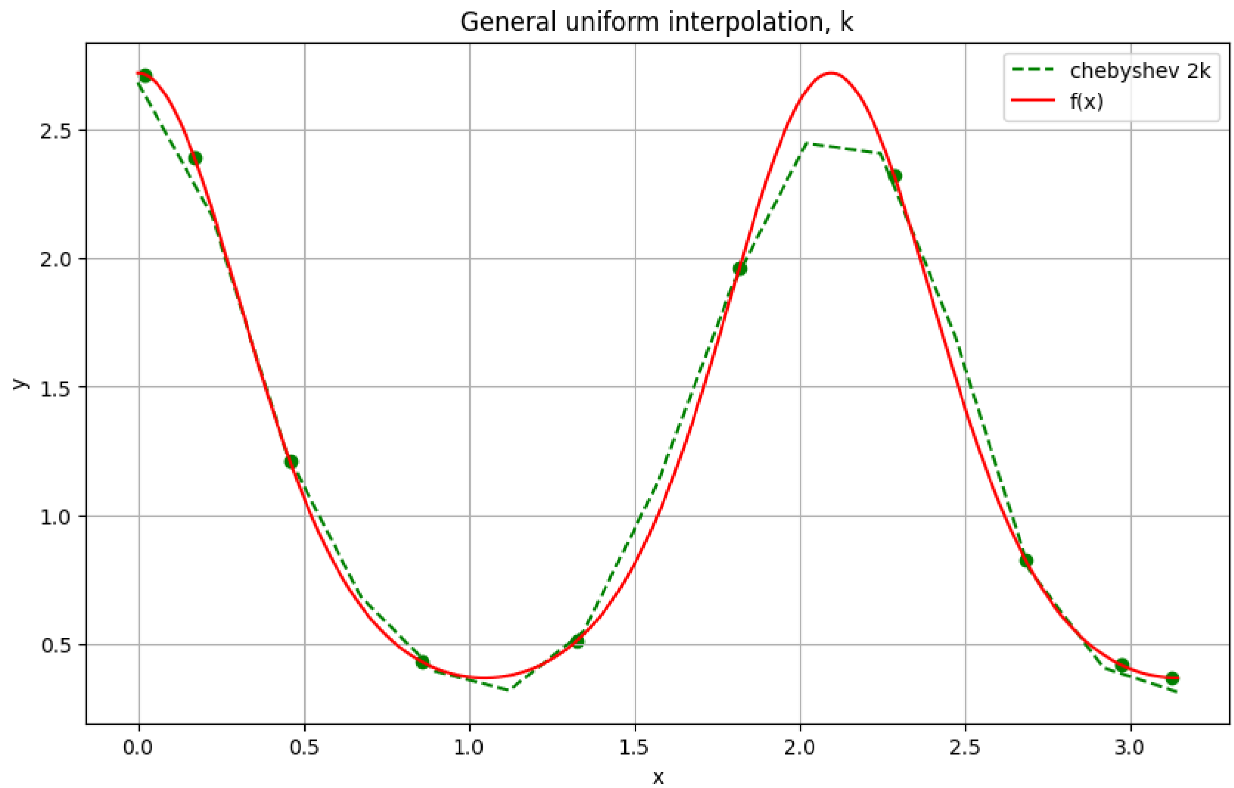
chebyshev при разбиении на k точек



uniform при разбиении на 2k точек



chebyshev при разбиении на 2k точек



Сравнение ошибок для разбиения на k точек

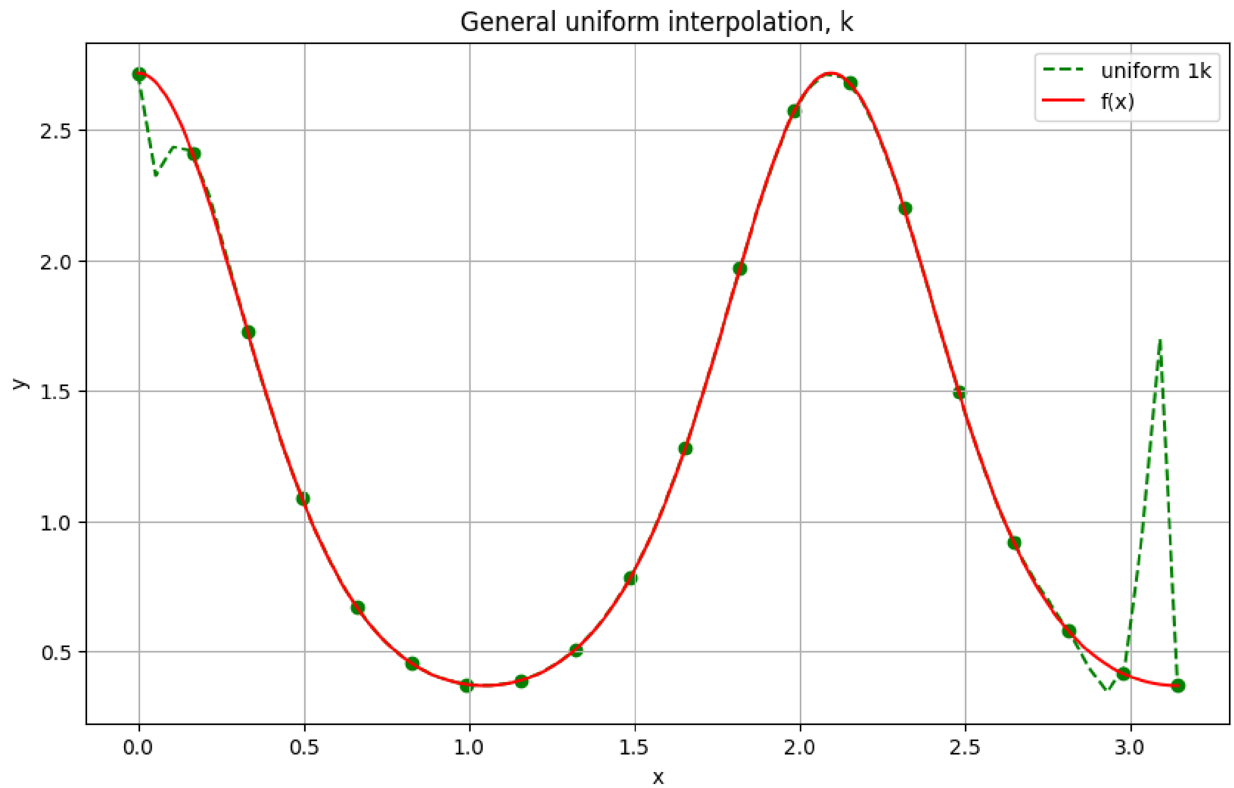
```
MSE для uniform: 0.5087654248724588
MSE для chebyshev: 0.4852486399402095
```

Сравнение ошибок для разбиения на 2k точек

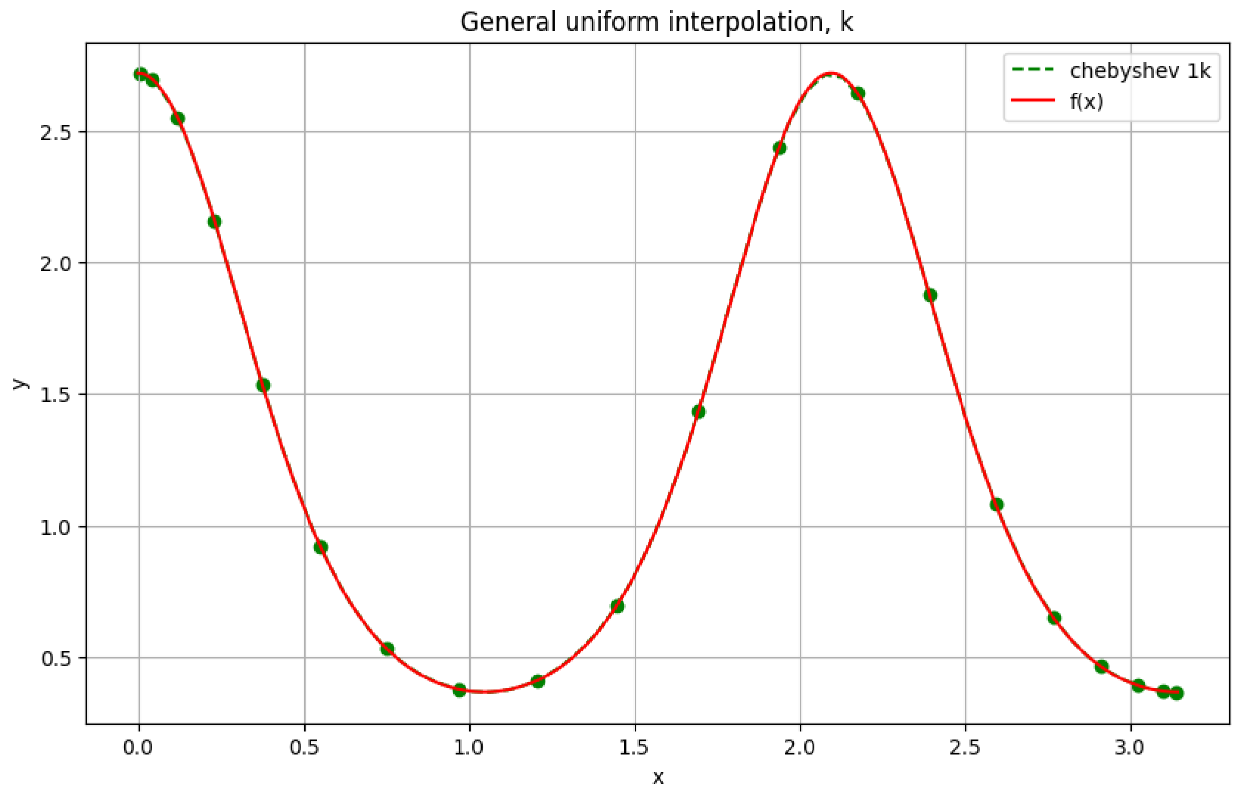
```
MSE для равномерных узлов: 0.22867517450515623
MSE для чебышевских узлов: 0.0811109676715146
```

Результаты (k = 20)

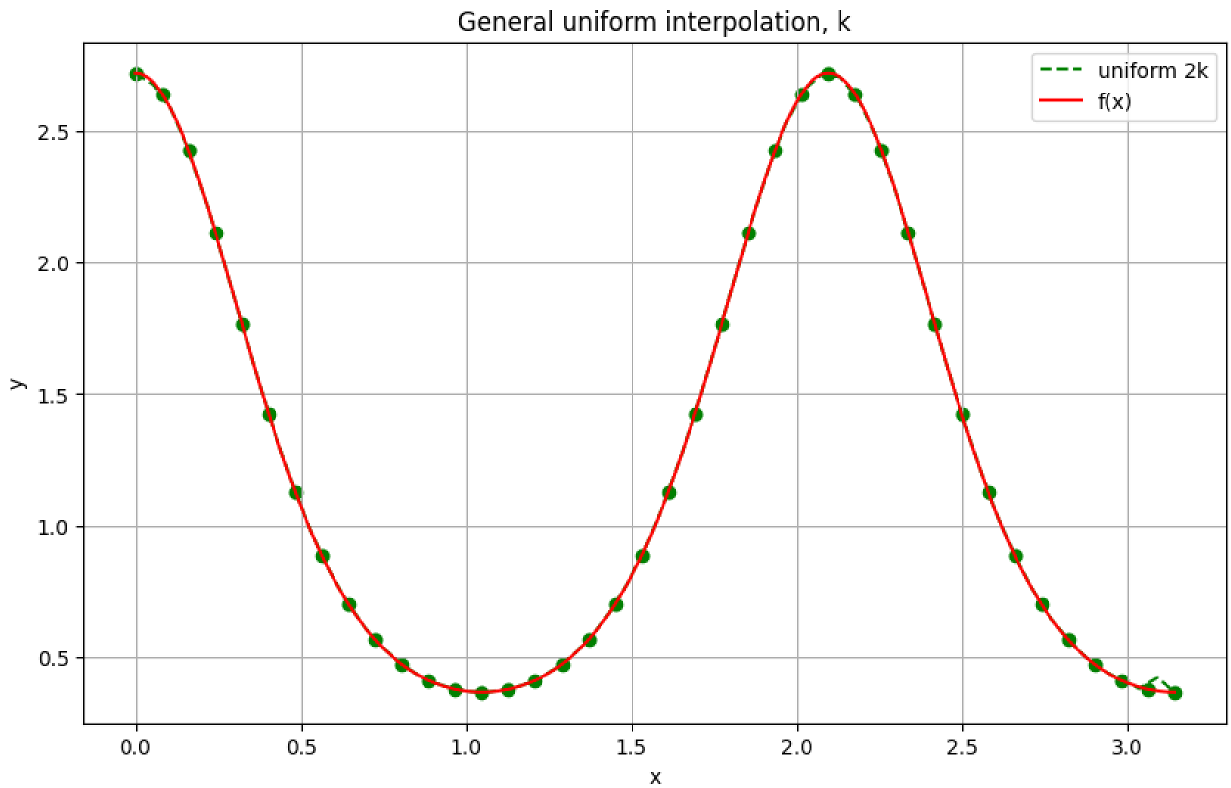
uniform при разбиении на k точек



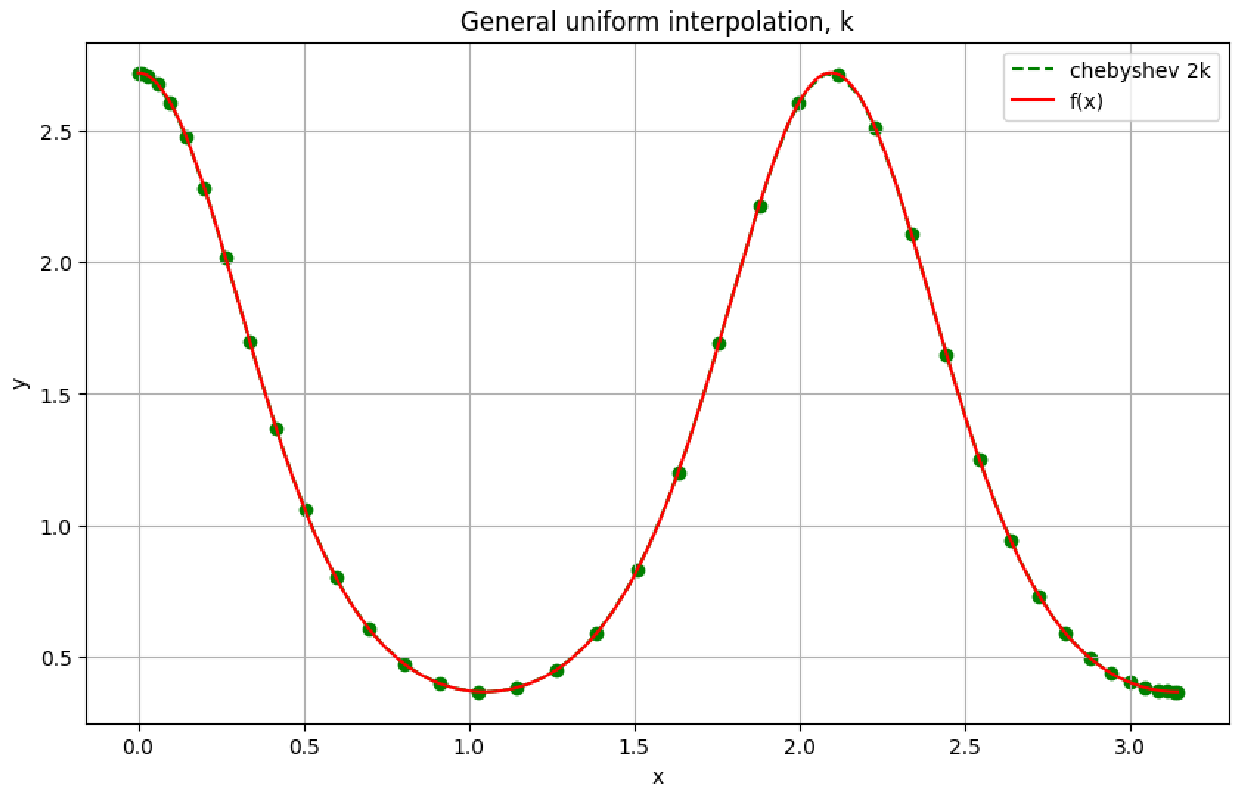
chebyshev при разбиении на k точек



uniform при разбиении на 2k точек



chebyshev при разбиении на 2k точек



Сравнение ошибок для разбиения на k точек

```
MSE для uniform: 0.19358735251255246
MSE для chebyshev: 0.0010877779077564 14
```

Сравнение ошибок для разбиения на 2k точек

```
MSE для uniform: 0.006674563627358505
MSE для chebyshev: 7.215279583334551e-08
```