Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего профессионального образования
"Национальный исследовательский университет
"Высшая школа экономики"

Московский институт электроники и математики Национального исследовательского университета "Высшая школа экономики"

Департамент прикладной математики

ОТЧЕТ
По лабораторной работе №3
на тему
«ПРИБЛИЖЕНИЕ ФУНКЦИЙ»

ФИО студента	Номер группы	Дата	Вариант
Ткаченко Никита Андреевич	БПМ211	18.05.2024	6.1.21, 6.2.4, 6.6.11, 6.8.2

Москва – 2024г.

6.1. Решение систем нелинейных уравнений

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.

6.1.21	
0	-2.815
0.25	-2.18
0.5	-0.225
0.75	1.722
1	3.492
1.25	3.31
1.5	2.945
1.75	1.449
2	0.334
2.25	-1.906
2.5	-3.430
2.75	-2.983
3	0.087

Нам дана таблица пар y = f(x), по которой при помощи метода наименьший квадратов требуется найти коэффициенты многочлена наилучшего приближения к заданной функции.

Аппроксимирующие функции выглядят как Pm(x) = a0*f0(x) + a1*f1(x) + … + am*fm(x), в нашем случае функции f1 — fm — это степенные функции 1, x, x^2, x^3, … , x^m, поэтому получаем многочлен вида: Pm(x) = a0 + a1x + a2x^2 + … + am*x^m.

Такой многочлен называется интерполяционным, если Pm(x1) = y1, что можно представить в виде СЛАУ, или P*a = y (где a это вектор коэффициентов).

$$\begin{aligned}
\varphi_0(x_0)a_0 + \varphi_1(x_0)a_1 + \dots + \varphi_m(x_0)a_m &= y_0,\\
\varphi_0(x_1)a_0 + \varphi_1(x_1)a_1 + \dots + \varphi_m(x_1)a_m &= y_1,\\
\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\\
\varphi_0(x_n)a_0 + \varphi_1(x_n)a_1 + \dots + \varphi_m(x_n)a_m &= y_n,
\end{aligned}$$

Если мы рассмотрим матрицу Грамма G = P*P:

$$\Gamma = P^*P = \begin{bmatrix} (\varphi_0, \varphi_0) & (\varphi_1, \varphi_0) & \dots & (\varphi_m, \varphi_0) \\ (\varphi_0, \varphi_1) & (\varphi_1, \varphi_1) & \dots & (\varphi_m, \varphi_1) \\ \dots & \dots & \dots & \dots \\ (\varphi_0, \varphi_m) & (\varphi_1, \varphi_m) & \dots & (\varphi_m, \varphi_m) \end{bmatrix}.$$

, где P* = Transpose(P) в нашем случае, и систему Ga = P*y, то решение этой системы относительно коэффициентов a и будет набором нужных нам коэффициентов.

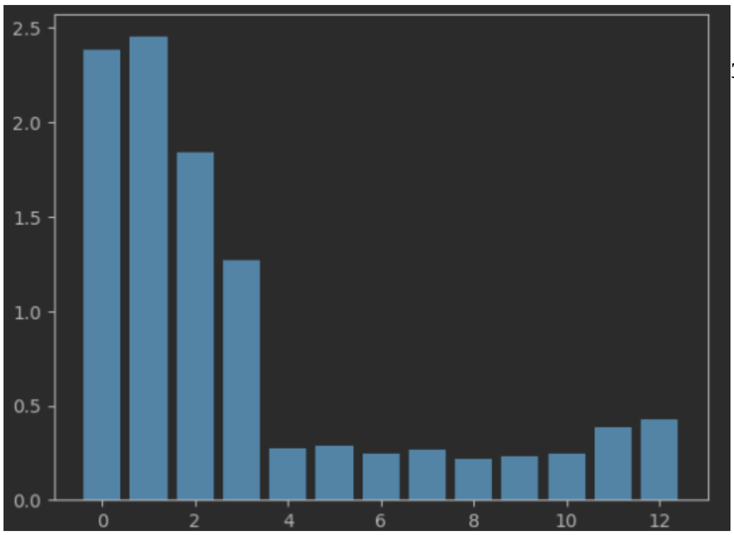
(коэффициенты вектора правой части можно вычислить по формуле $b_j = (y, \varphi_j) = \sum_{l=0}^{n} y_l \overline{\varphi_j(x_l)}$,
)

Реализуем это в коде:

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

    b = np.empty(m)  # b
    gramm = np.empty((m, m))  # Г

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

    # Г = (P^T)*P
    for j in range(m):
        for k in range(m):
            gramm[j][k] = sum(x ** (k + j))

    best_coefficients = np.linalg.solve(gramm, b)

    def approximation(point_x):
        point_y = 0
        for i in range(m):
            point_y += best_coefficients[i] * point_x ** i
        return point_y

    return approximation, best_coefficients
```

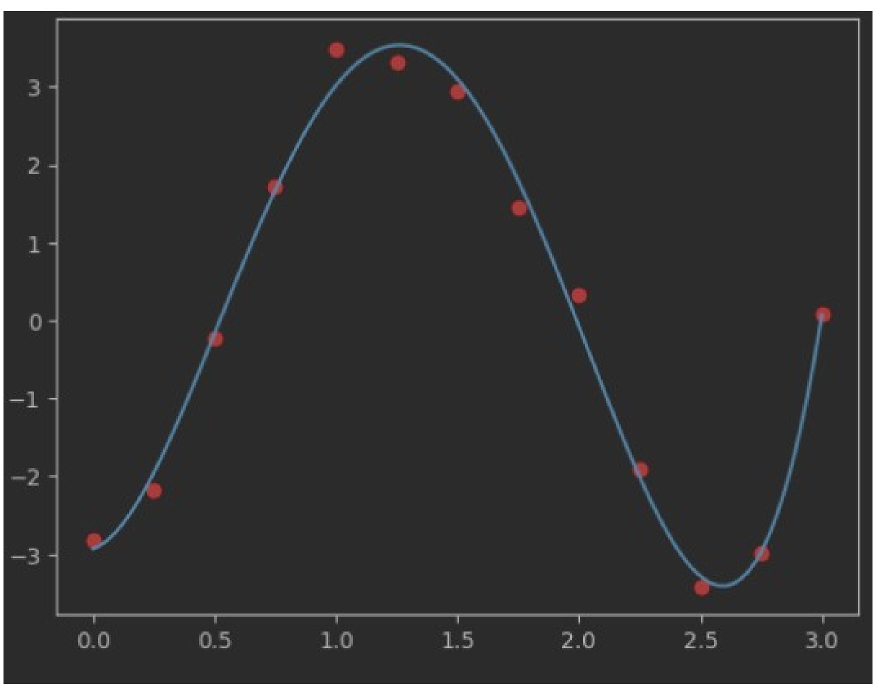
Теперь найдем оптимальную степень интерполяционного многочлена, пройдясь по всем возможным и графически ее определя:

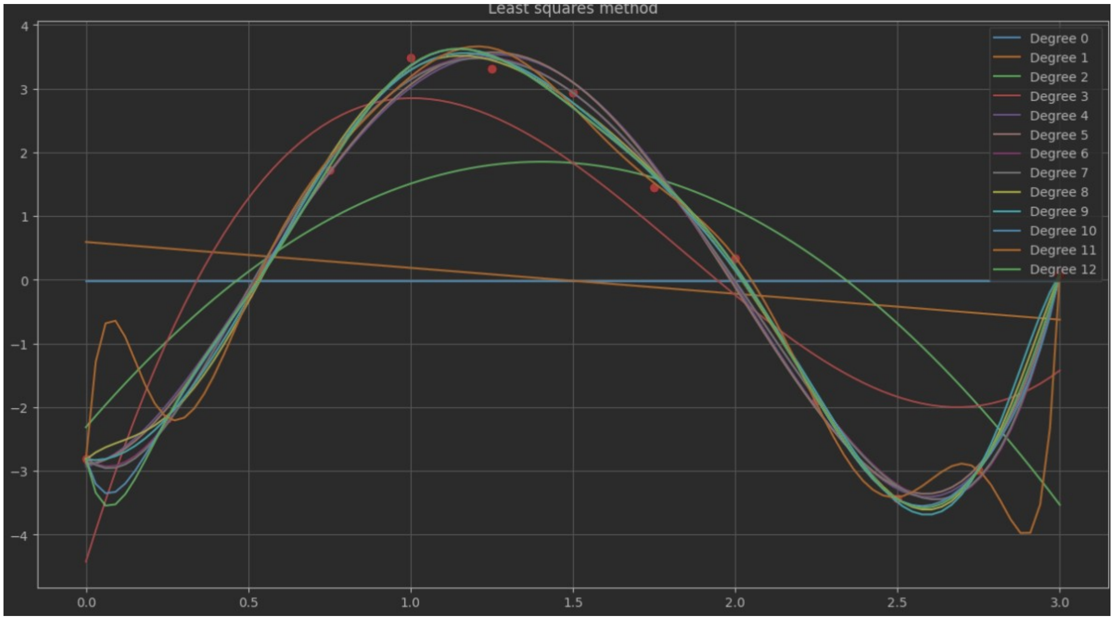
```
rmse_array = []
functions = []
coefficients = []
for m in range(len(x)):
    function, coefficient = lsm(x, y, m)
    rmse = np.sqrt((1 / (len(x) - m)) * np.sum((function(x) - y) ** 2))
    rmse_array.append(rmse)
    functions.append(function)
    coefficients.append(coefficient)
```



Видно, что график стабилизиуретса на значениях m >= 4, поэтому выберем m = 4.

Наконец построим получившийся многочлен, отобразив исходные точки на этом же графике:





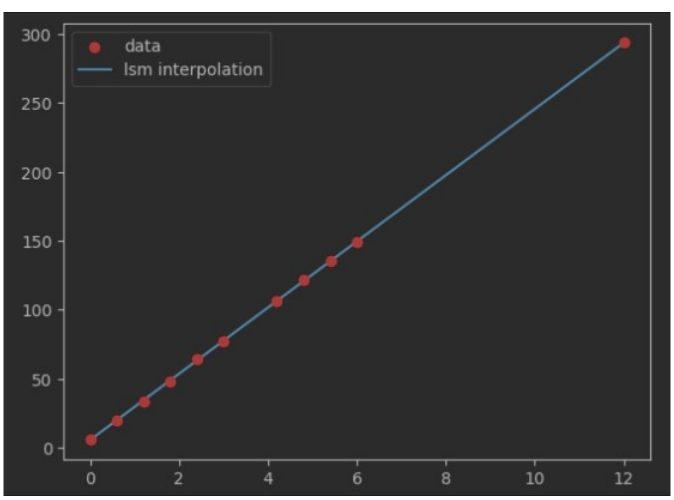
6.2 Перемещение материальной точки

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.

6.2.4	t	0	0.6	1.2	1.8	2.4	3	4.2	4.8	5.4	6
	x	6.449	19.97	33.91	48.2	64.15	76.9	106.2	122.2	135.6	149

Воспользуемся только что написанным методом с первой степенью многочлена (мы знаем что зависимость линейная), и найдем скорость — второй коэффициент (у члена x):

```
approximation, best_coefficients = lsm(np.array(t),x,1)
print(best_coefficients)
t.append(t[-1]*2)
x.append(approximation(t[-1]))
```



6.6 Многочлены Лагранжа

Задача 6.6. Дана функция $y=f(x)$. Приблизить $f(x)$ на отрезке $[a,b]$ интерполяционными многочленами Лагранжа 1, 2, 3 степеней. На одном чертеже построить графики приближающих многочленов и функции $f(x)$. Для многочлена 3 степени сравнить качество приближения при различном выборе узлов интерполяции.

6.6.11	
$x^2\cos(x)$	$\left[\frac{\pi}{2}, \pi\right]$

Многочлен Лагранжа - интерполяционный многочлен следующего вида:

$$L_n(x) = \sum_{j=0}^{n} y_j l_{nj}(x), \text{ , где } l_{nj}(x) = \prod_{\substack{k=1 \\ k \neq j}}^{n} \frac{x - x_k}{x_j - x_k} = \frac{(x-x_0)(x-x_1)\ldots(x-x_{j-1})(x-x_{j+1})\ldots(x-x_n)}{(x_j-x_0)(x_j-x_1)\ldots(x_j-x_{j-1})(x_j-x_{j+1})\ldots(x_j-x_n)}.$$

Пример такого многочлена первой и второй степени:

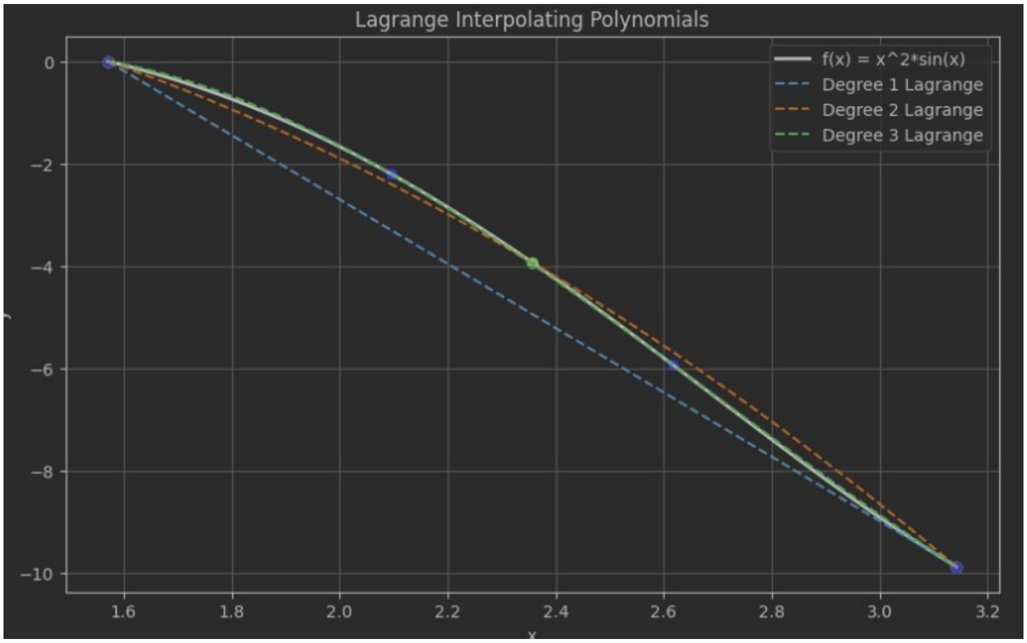
$$L_1(x) = y_0 \frac{x - x_1}{x_0 - x_1} + y_1 \frac{x - x_0}{x_1 - x_0},$$

$$L_2(x) = y_0 \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} + y_1 \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} + y_2 \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}.$$

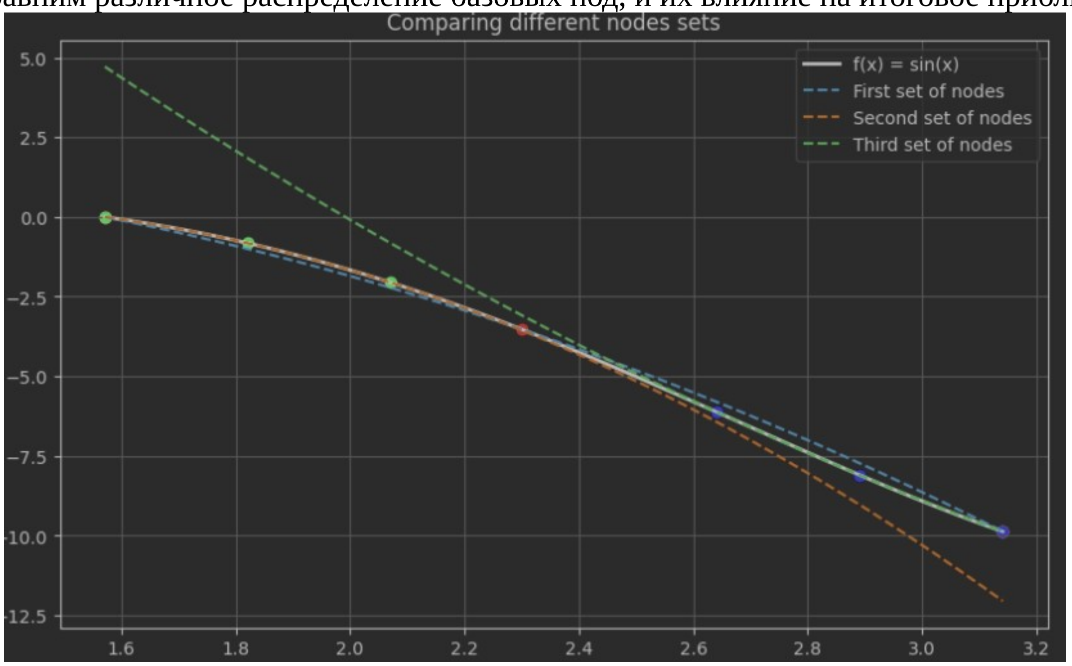
```
def lagrange_interpolating_polynomial(x_nodes, x):
    y_nodes = f(x_nodes)
    def basis_polynomial(i):
        p = [(x - x_nodes[j]) / (x_nodes[i] - x_nodes[j]) for j in range(len(x_nodes)) if j != i]
        return np.prod(p, axis=0)

    P = np.sum(y_nodes[i] * basis_polynomial(i) for i in range(len(x_nodes)))
    return P
```

Составим три многочлена трех степеней по выбранным нодам (количество нод = степени + 1, потому что многочлен первой степени это прямая (на двух точках)), и получим уже интерполированные значения для каждого из них:

```
a, b = np.pi/2, np.pi
nodes_1 = np.linspace(a, b, 2)
nodes_2 = np.linspace(a, b, 3)
nodes_3 = np.linspace(a, b, 4)

x_vals = np.linspace(a, b, 500)
y_1 = lagrange_interpolating_polynomial(nodes_1, x_vals)
y_2 = lagrange_interpolating_polynomial(nodes_2, x_vals)
y_3 = lagrange_interpolating_polynomial(nodes_3, x_vals)
```



Теперь сравним различное распределение базовых нод, и их влияние на итоговое приближение:



Видно, что наилучнее приближение дает равномерное распределение трех нод по выбранному отрезку, а не их группировка в начале/конце отрезка.

6.8 Равномерное и Чебышевское распределеное узлов интерполяции

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.

6.8.2	
$e^x \sin(5x)$	[1.5,3.5]

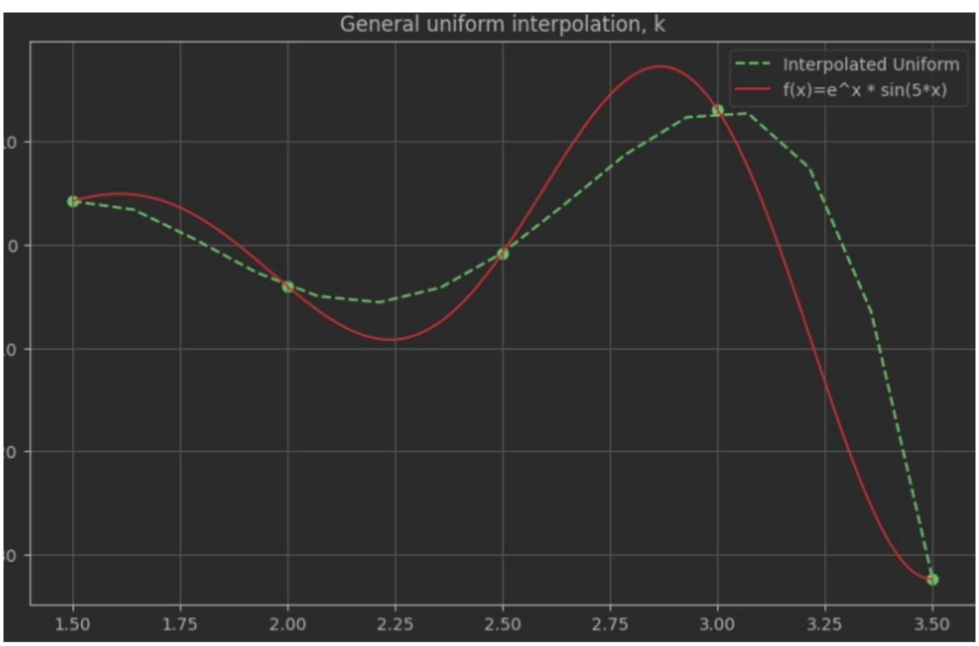
Метод глобальной интерполяции отличается от иного тем, что итогом является один многочлен, приближающий весь интервал интерполяции сразу, а не набор таких многочленов. Метод Лагранжа является глобальным методом, поэтому я воспользуюсь им.

Зададим распределение точек по нормальному закону и по распределению Чебышева, для k=5:

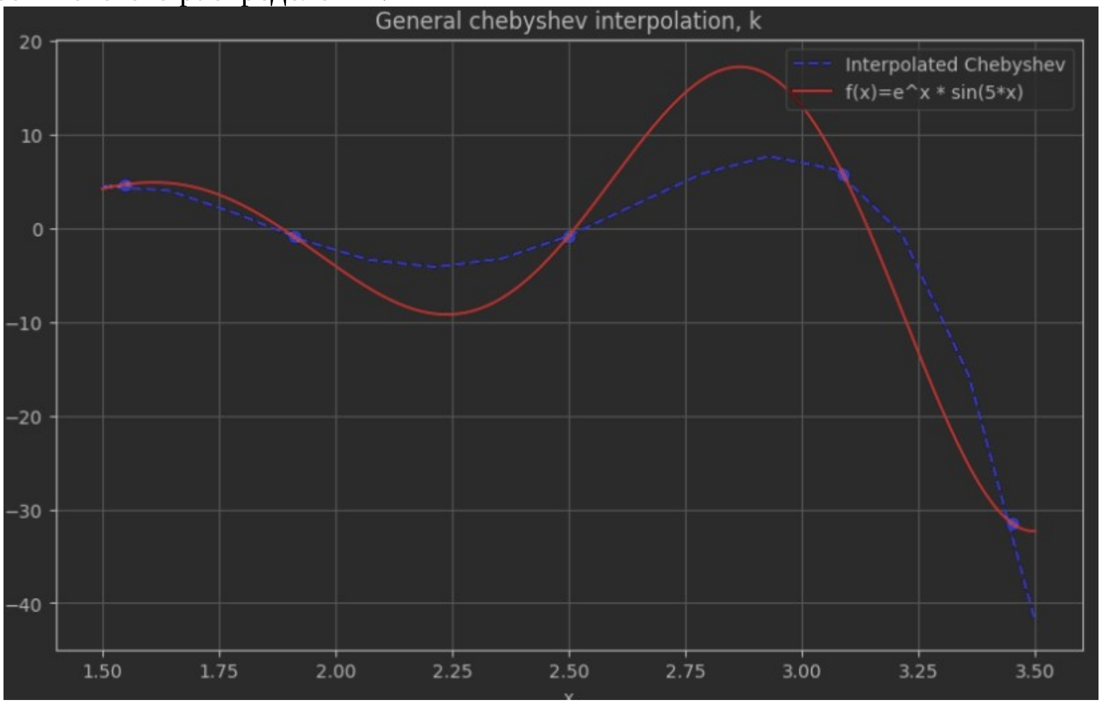
Распределение Чебышева выглядит так: $$x_k = \frac{1}{2}(a+b) + \frac{1}{2}(b-a)\cos\left(\frac{2k-1}{2n}\pi\right), \quad k = 1, \ldots, n.$$

```
a, b = 1.5, 3.5
k = 5
mult = 1
x_uniform = np.linspace(a, b, mult*k)
x_chebyshev = 0.5 * (a + b) + 0.5 * (b - a) * np.cos((2 * np.arange(1, mult*k + 1) - 1) / (2 * mult*k) * np.pi)

x_dense = np.linspace(a, b, 3 * k)
y_true = f(x_dense)
```

Построим многочлен по k узлам и найдем интерполированные значения по 3k узлам для равномерного распределения:



И для Чебышевского распределения:



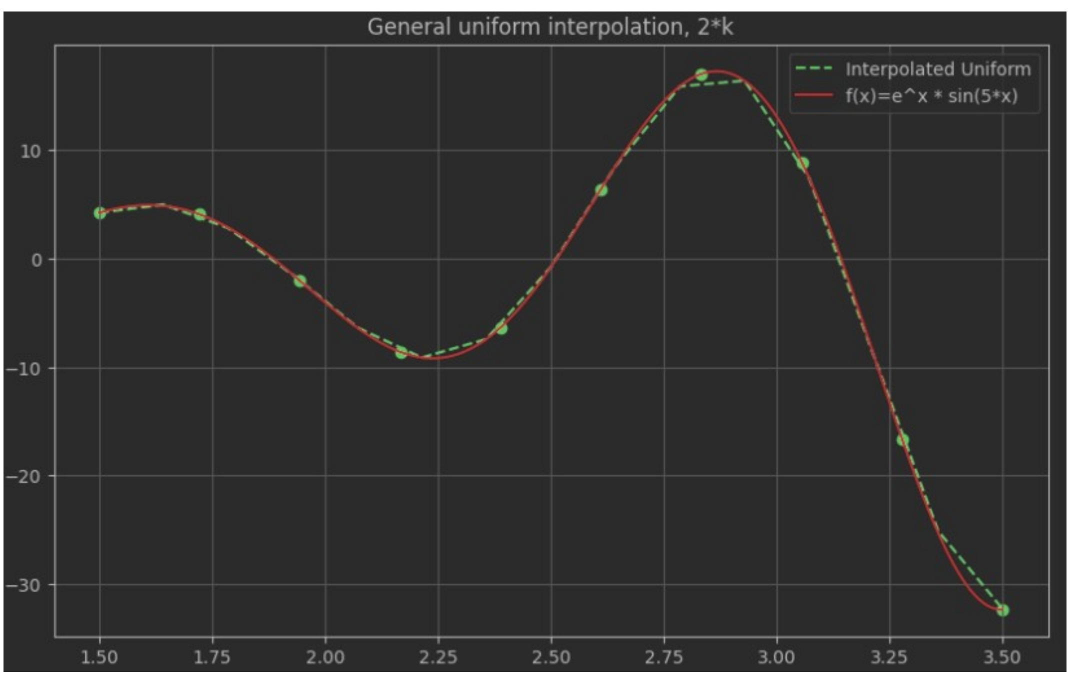
Видно, что точки интерполяции - их распределение - различаются между графиками.

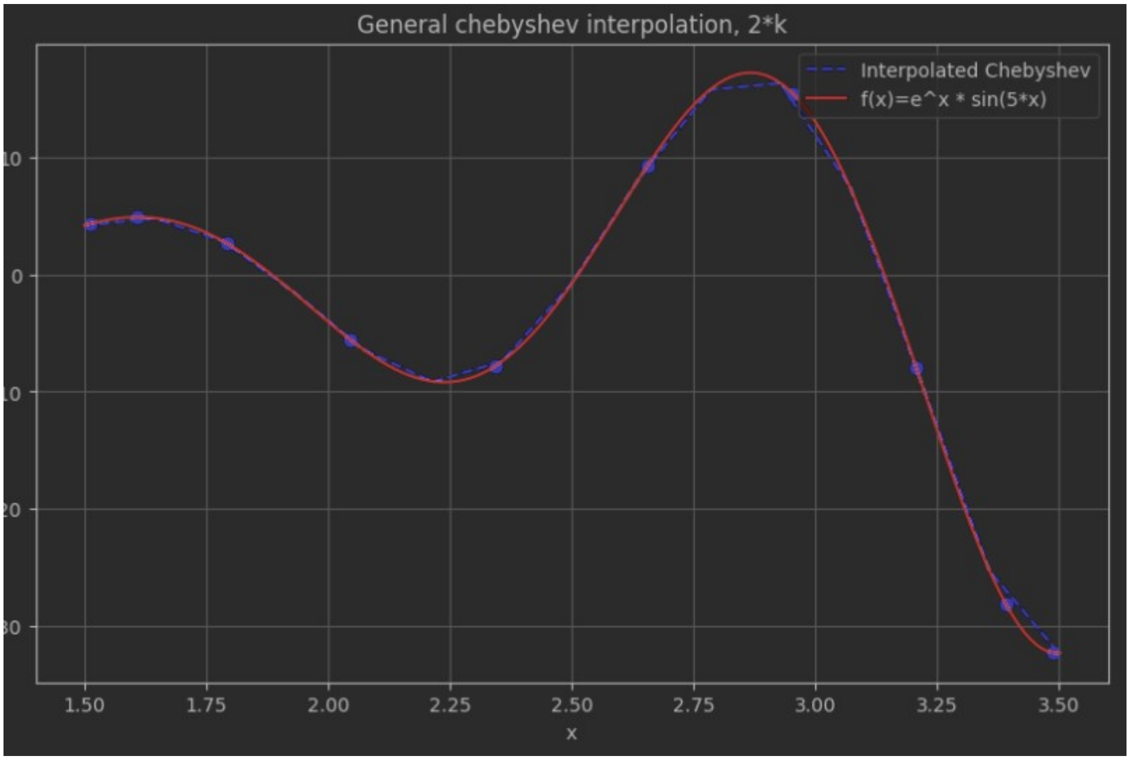
Теперь сравним качество приближения для разного распределения узлов, посчитав MSE между исходной функцией и многочленом для каждого из графиков:

```
MSE для равномерных узлов: 51.22783241181496
MSE для чебышевских узлов: 34.663443726876814
```

Выходит, Чебышевское распределение приближает чуть лучше для такого значения k.

Проделаем все тоже самое для 2k исходных узлов:





Если сравнить результаты для разного набора узлов:

```
MSE для равномерных узлов: 51.22783241181496
MSE для чебышевских узлов: 34.663443726876814
```

```
MSE для равномерных узлов: 0.0035609297968879823
MSE для чебышевских узлов: 0.0010366882097171295
```

То видно, что в среднем интерполяция получается лучшей для Чебышевского распределения, а ее точность многократно увеличилась лишь при удвоении числа узлов интерполяции.