МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «МЭИ»

Отчёт по лабораторной работе №4
“Приближение функций”
Вариант (2 + 10 = 12)

Выполнил:
студент группы А-13а-19
Башлыков Матвей

Проверил:
Крупин Григорий Владимирович

Задание 4.1

В таблице приведены данные о численности населения по годам 1950-2000 г.г. Заполнить последние два столбца таблицы. На основе этих данных построить наилучший многочлен по МНК. Найти численность населения страны в 2019 году и сравнить полученное значение с актуальным. Решить ту же задачу на основе интерполяционного многочлена. Составить отчёт по задаче.

Вариант N = 12 =>

Страна	Численность населения (в тыс.)							
Филиппины	1950	1960	1970	1980	1990	2000	2010	2020
	21	29	39	48.5	63	83	94	110

```
years = np.array([1950.0, 1960.0, 1970.0, 1980.0, 1990.0, 2000.0, 2010.0, 2020.0])
population = np.array([21.0, 29.0, 39.0, 48.5, 63.0, 83.0, 94.0, 110.0])

year = 2019
population2019 = 108
```

В 2019 году население составило ~108 млн человек.

Теоретический материал:
Метод наименьших квадратов (МНК) - метод, применяемый для аппроксимации точечных значений некоторой функции. Для построенного многочлена должна выполняться система:

$$a_0\varphi_0(x_0) \; + a_1\varphi_1(x_0) \; +\ldots + a_m\varphi_m(x_0) \; \approx y_0$$

$$\ldots$$

$$a_0\varphi_0(x_n) \; + a_1\varphi_1(x_n) \; +\ldots + a_m\varphi_m(x_n) \; \approx y_n$$

Из системы приходим к задаче минимизации квадрата норма вектора невязки. Из формулы для среднеквадратичного отклонения

$$\sigma(\Phi_m, f) \; = \sqrt{\frac{1}{n+1}\sum_{i=0}^{n} (P_m(x_i) \; - f_i)^2} = \sqrt{\frac{1}{n+1}\sum_{i=0}^{n} (\sum_{j=0}^{m} a_j x_i^j \; - f_i)^2}$$

и необходимого условия экстремума выводится система

$$a_0(\sum_{i=0}^{n} x_i^0) \; + a_1(\sum_{i=0}^{n} x_i^1) \; +\ldots + a_m(\sum_{i=0}^{n} x_i^m) \; \approx \sum_{i=0}^{n} x_i^0 y_i$$

$$\ldots$$

$$a_0\left(\sum_{i=0}^{n} x_i^{m}\right) + a_1\left(\sum_{i=0}^{n} x_i^{m+1}\right) + \ldots + a_m\left(\sum_{i=0}^{n} x_i^{2m}\right) = \sum_{i=0}^{n} x_i^{m} y_i$$

При этом метод МНК рекомендуется использовать при достаточно малых размерах системы, поэтому в данной работе исследуются многочлены степени не выше 5.

Интерполяционный многочлен обязан совпадать с исходными значениями в данных точках. В качестве интерполяционного многочлена можно взять многочлен Лагранжа, получаемый по формуле:

$$L_n(x) = \sum_{i=0}^{n} y_i \prod_{k=0,\, k\neq i}^{n} \frac{(x - x_k)}{(x_i - x_k)}$$

Решение:
Напишем функции для вычисления многочлена через МНК (FillLeastSqMatrices) и интерполяционного многочлена (Lagrange)

```python
def ComputeS(years, n, k):
    s = 0
    for i in range(n):
        s += years[i]**k
    return s

def ComputeB(population, years, n, k):
    b = 0
    for i in range(n):
        b += population[i] * years[i]**k
    return b

def FillLeastSqMatrices(A, b, years, population, m, n):
    for k in range(m + 1):
        for i in range(n + 1):
            b[k] += population[i] * years[i]**k
            A[0][k] += years[i]**k
            A[m][k] += years[i]**(k + m)
    #print(A)
    for j in range(m + 1):
        A[j][m - j] = A[0][m]
        for k in range(1, j + 1):
            A[k][j - k] = A[0][j]
        for k in range(1, m - j + 1):
            A[m - k][j + k] = A[m][j]
            #print(A)
    return A, b

def ComputePolynomValue(polynom, year):
    result = 0
    m = len(polynom)
    for i in range(m):
        #print(polynom[i] * years[j]**i)
        result += polynom[i] * year**i
    return result

def StandardDeviation(NewPolynom, years, population, n, m):
    result = 0
    for i in range(n + 1):
        #result += (NewPolynom[j] * years[i]**j - population[i])**2
        result += (ComputePolynomValue(NewPolynom, years[i]) - population[i])**2
    return (result/(n + 1))**0.5
```

```python
def Lagrange(population, years, x0, n):
    res = 0
    max_n = min(population.size, n)
    for i in range(max_n):
        a = 1
        for j in range(max_n):
            if (i != j):
                a = a * (x0 - years[j])/(years[i] - years[j])
        res = res + population[i] * a
    return res
```

Вычислим необходимые значения и сравним с актуальными, а также найдём наименьшее СКО для многочленов МНК не выше 5 степени.

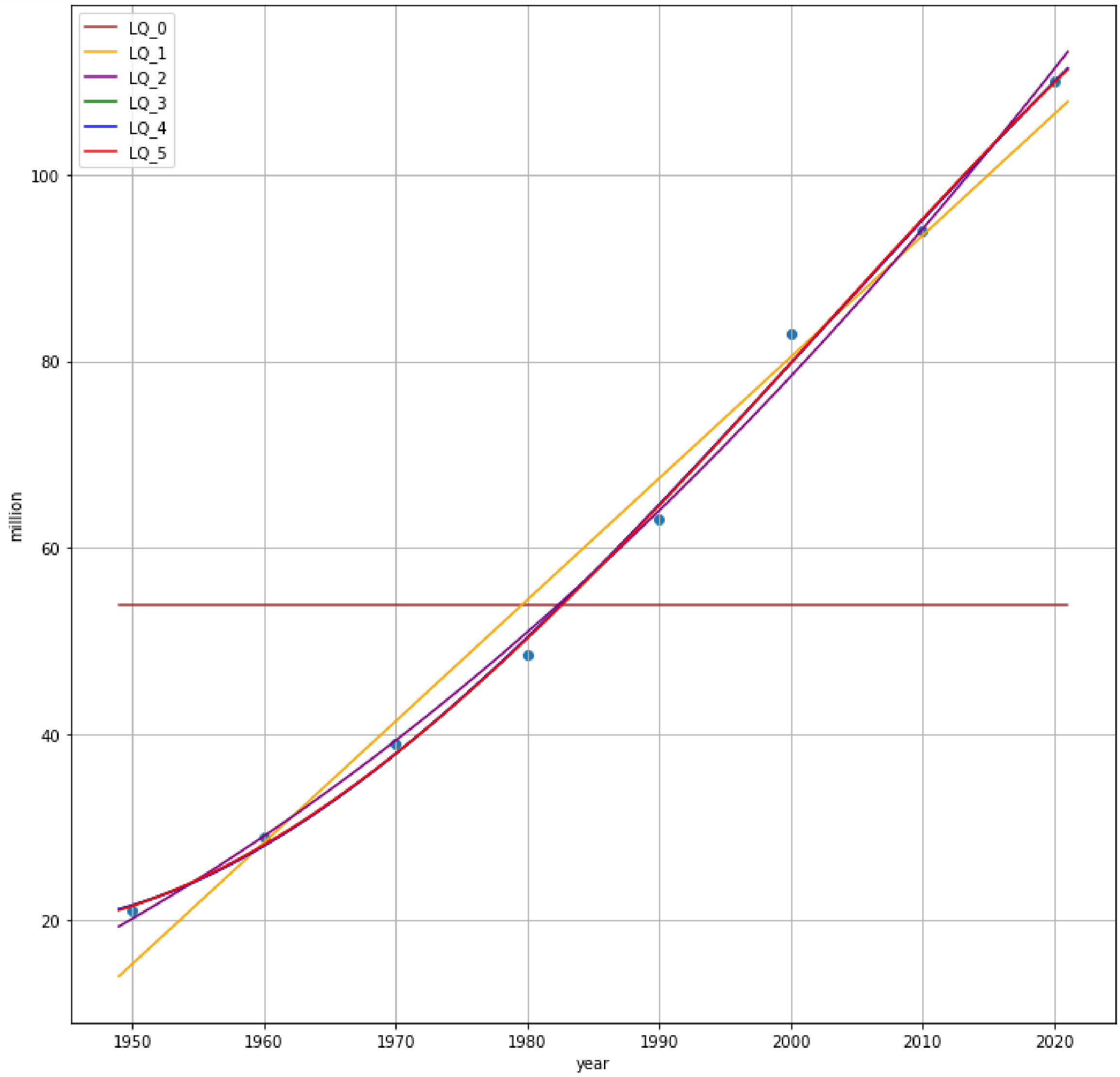
```
a = []
for i in range(6):
    a.append([0] * (i + 1))
s = ComputeS(years, 7, 0)
b = ComputeB(population, years, 7, 0)
a[0][0] = b/s

for i in range(1, 6):
    A = []
    for j in range(i + 1):
        A.append([0]*(i + 1))
    b = np.zeros(i + 1)
    FillLeastSqMatrices(A, b, years, population, i, 7)
    a[i] = np.linalg.solve(A, b)
```

```
for i in range(6):
    print("LQ polynom degree", i, "at 2019.0: ", ComputePolynomValue(a[i], 2019.0))
print("Interpolated polynom at 2019.0:", Lagrange(population, years, 2019.0, population.size))
print("True value: ", population2019)

minD = StandardDeviation(a[0], years, population, 7, 0)
minI = 0
for i in range(1, 6):
    newM = StandardDeviation(a[i], years, population, 7, i)
    if (newM < minD):
        minD = newM
        minI = i
print("The most accurate LQ polynom: degree", minI, "and its Standard Deviation:", minD)
```

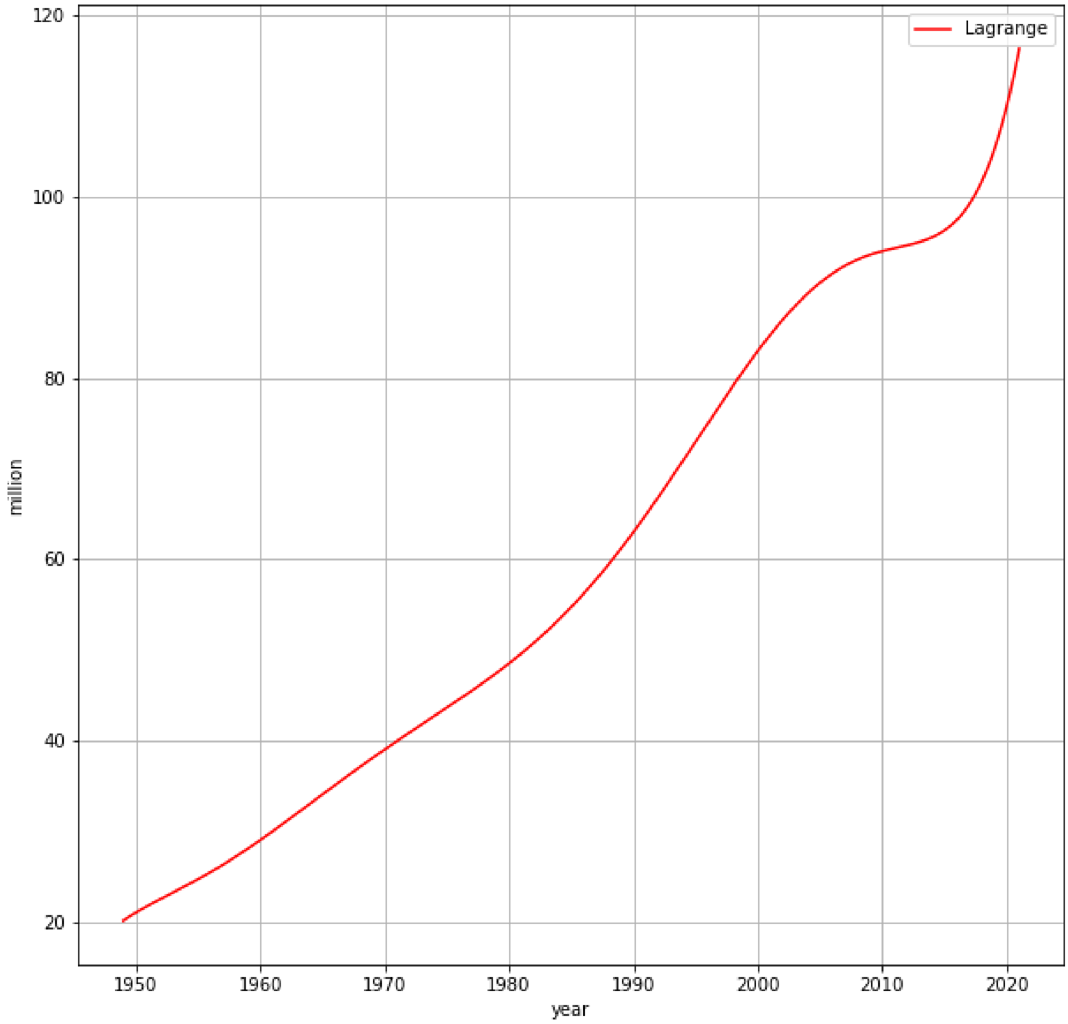
```
LQ polynom degree 0 at 2019.0:  53.92857142857143
LQ polynom degree 1 at 2019.0:  105.23869047619928
LQ polynom degree 2 at 2019.0:  109.61437496366852
LQ polynom degree 3 at 2019.0:  108.55333920801058
LQ polynom degree 4 at 2019.0:  108.56220279727131
LQ polynom degree 5 at 2019.0:  108.53419672558084
Interpolated polynom at 2019.0: 105.221640795625
True value:  108
The most accurate LQ polynom: degree 5 and its Standard Deviation: 1.5624389846352043
```

Как мы видим, многочлен 5 степени метода МНК имеет наименьшую СКО. Кроме того, он ближе всех остальных (в том числе интерполяционного) к актуальному значению численности населения в 2019 году.

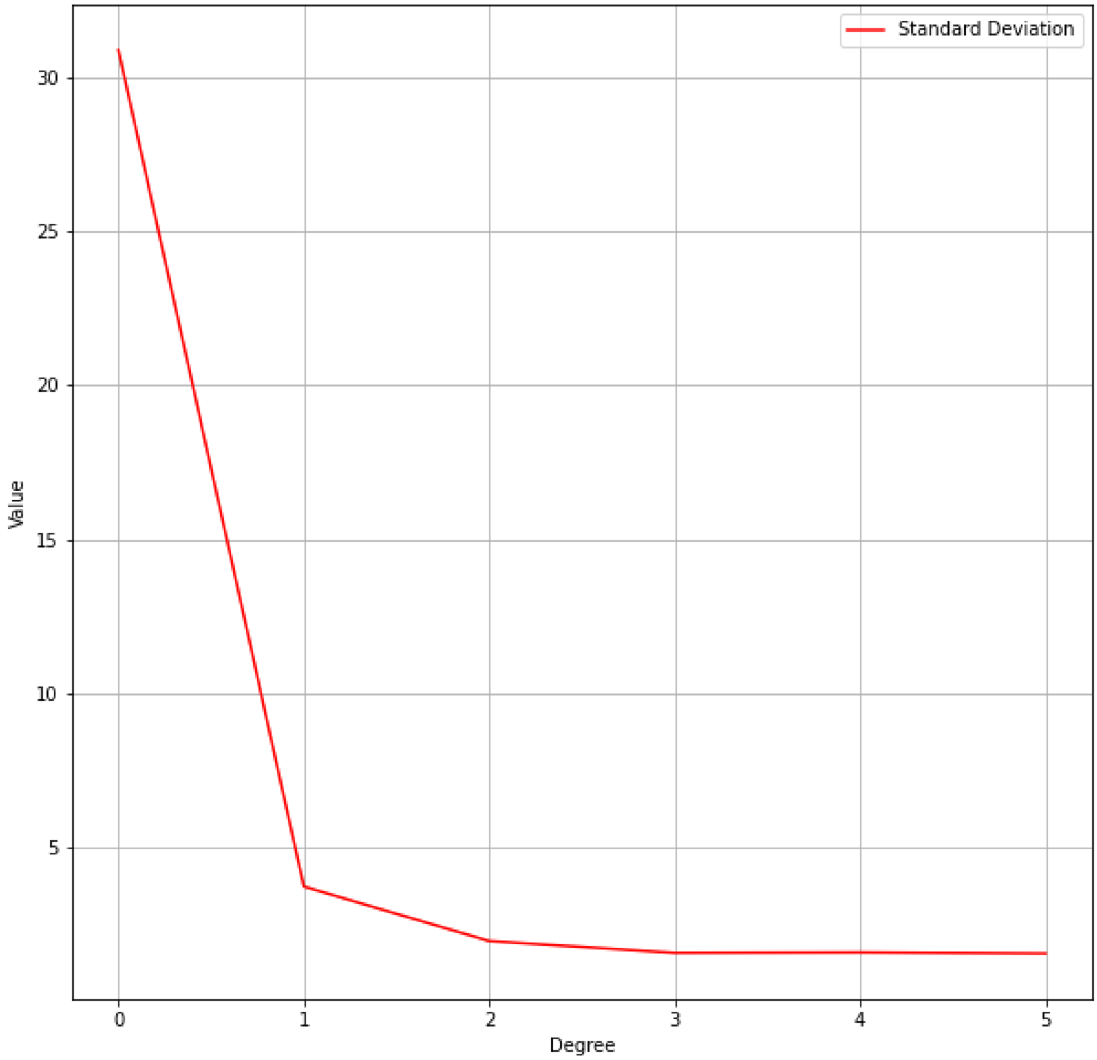
Приложенные графики:
Многочлены МНК



Интерполяционный многочлен:



Изменение СКО в зависимости от степени многочлена:



Как мы видим, многочлен 5 степени МНК достаточно хорошо приближает функцию (получили ~108.5342 при указанных 108) . Однако даже так СКО остаётся велико, что объяснимо плохой обусловленностью матрицы A для системы МНК. Кроме того, в связи с тем, что после многочлена 3 степени СКО фактически “стабилизируется”, более эффективным решением будет выбирать его (получаем значение ~108.5533 в 2019 году)

```
print(np.linalg.cond(A))
```

```
1.6399285129745923e+38
```

Задание 4.2

Имеем $f(x) = x\,sin(x^2)$, [a, b] = [0, $\frac{\pi}{2}$], eps = 0.001

Зададим произвольное начальное число отрезков разбиения, пусть n = 3, имеем 4 точки. Построим таблицу значений:

x	0	$\frac{\pi}{6}$	$\frac{\pi}{3}$	$\frac{\pi}{2}$
y	0	0.14175612	0.93166062	0.98059467

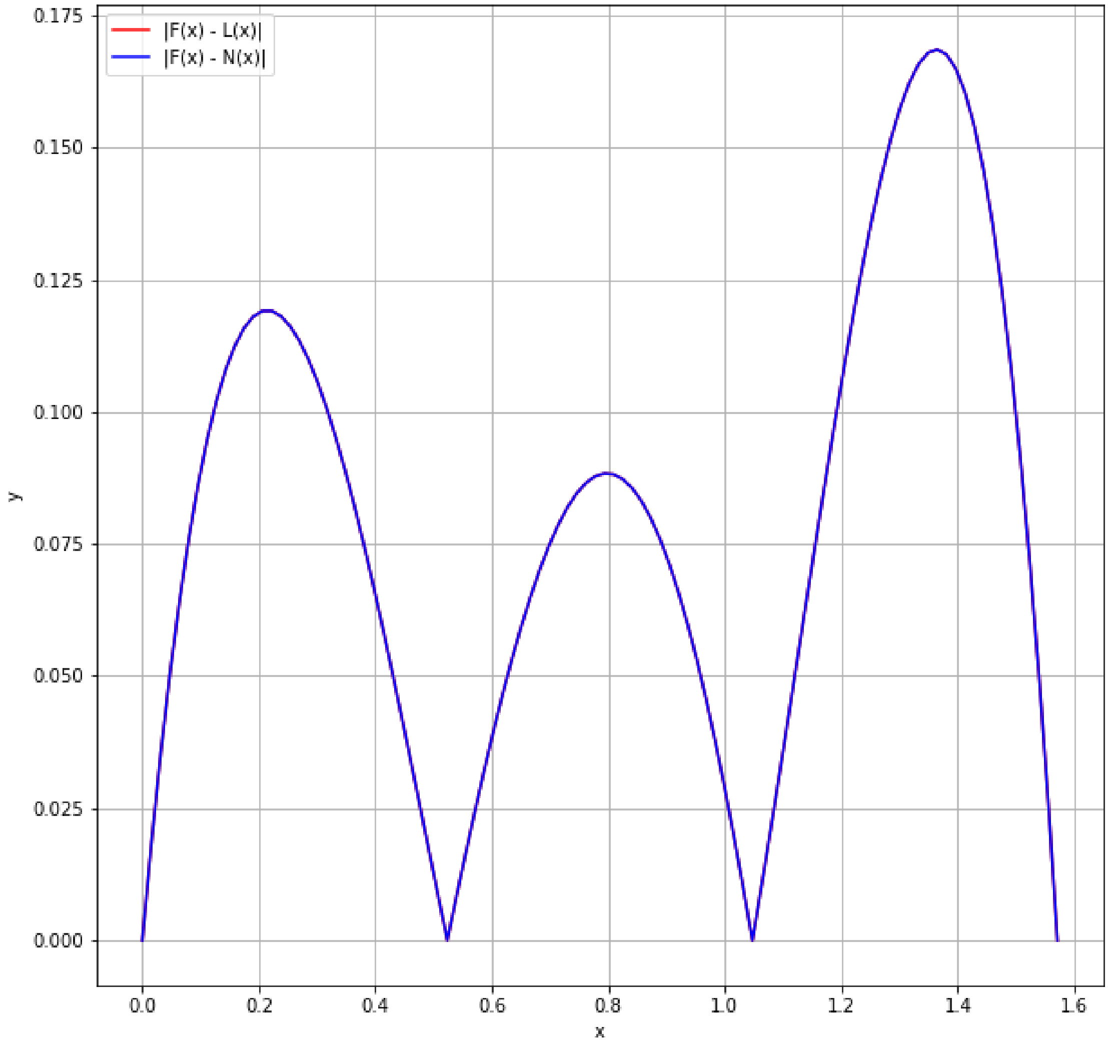
Имеем функции:

```
import math
def F(x):
    return x * np.sin(x*x)

a = 0
b = math.pi/2
eps = 0.001
n_min_Lagrange = 4
n_min_Newton = 4
x = np.linspace(a, b, 4)
y = F(x)
print(x)
print(y)

def Difference(x, y, n):
    n_max = min(y.size - 1, n)
    h = x[1] - x[0]
    saveY = y
    res = np.zeros(n_max)
    newY = np.zeros(saveY.size - 1)
    for i in range(n_max):
        for j in range(saveY.size - 1):
            newY[j] = saveY[j + 1] - saveY[j]
        res[i] = newY[0]
        saveY = newY
        newY = np.zeros(saveY.size - 1)
    return res

def Newton(x0, x, y, res, n):
    n_max = min(x.size - 1, n)
    h = x[1] - x[0]
    a = (x0 - x[0])/h
    result = y[0] + res[0] * a
    for i in range(1, n_max):
        a = a * (x0 - x[i]) / (h * (i + 1))
        result = result + res[i] * a
    return result
```

```
[0.         0.52359878 1.04719755 1.57079633]
[0.         0.14175612 0.93166062 0.98059467]
```

```
def Lagrange(population, years, x0, n):
    res = 0
    max_n = min(population.size, n)
    for i in range(max_n):
        a = 1
        for j in range(max_n):
            if (i != j):
                a = a * (x0 - years[j])/(years[i] - years[j])
        res = res + population[i] * a
    return res
```

Построим графики $RN(t) = |f(t) - N(t)|$ и $RL(t) = |f(t) - L(t)|$:



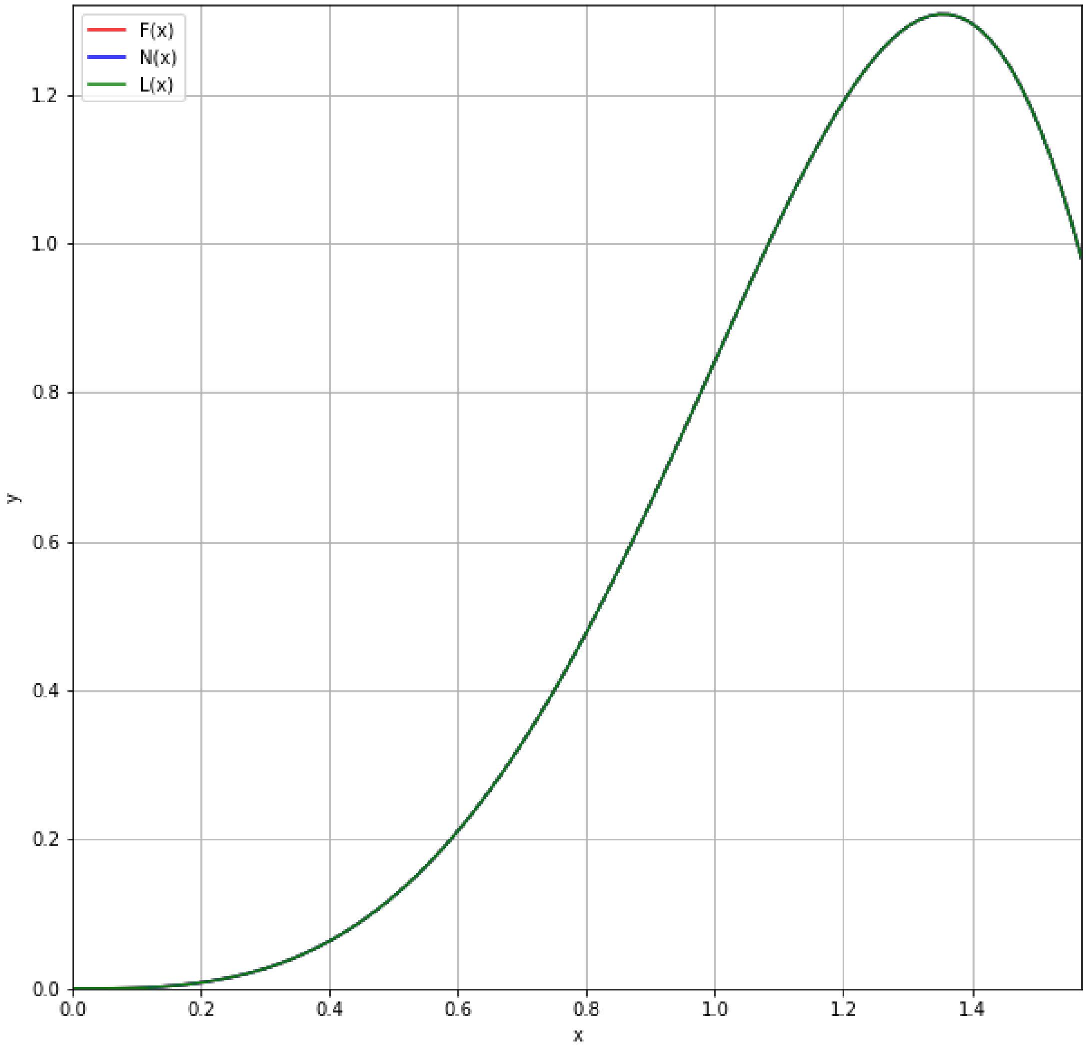
Как мы видим, максимальная погрешность более 0.150, что превышает нашу точность eps = 0.001. Необходимо увеличить число отрезков разбиения. Найдём нужное число:

```python
x_1 = np.linspace(a, b, 100)
flag = True
n = n_min_Lagrange
while flag:
    x_n = np.linspace(a, b, n)
    Lagr = [Lagrange(F(x_n), x_n, x0, n) for x0 in x_1]
    if np.amax(np.abs(Lagr - F(x_1))) < eps:
        n_min_Lagrange = n
        flag = False
    n += 1

n = n_min_Newton
flag = True
while flag:
    x_n = np.linspace(a, b, n)
    Newt = [Newton(x0, x_n, F(x_n), Difference(x_n, F(x_n), n), n) for x0 in x_1]
    if np.amax(np.abs(Newt - F(x_1))) < eps:
        n_min_Newton = n
        flag = False
    n += 1
print(n_min_Lagrange)
print(n_min_Newton)
```

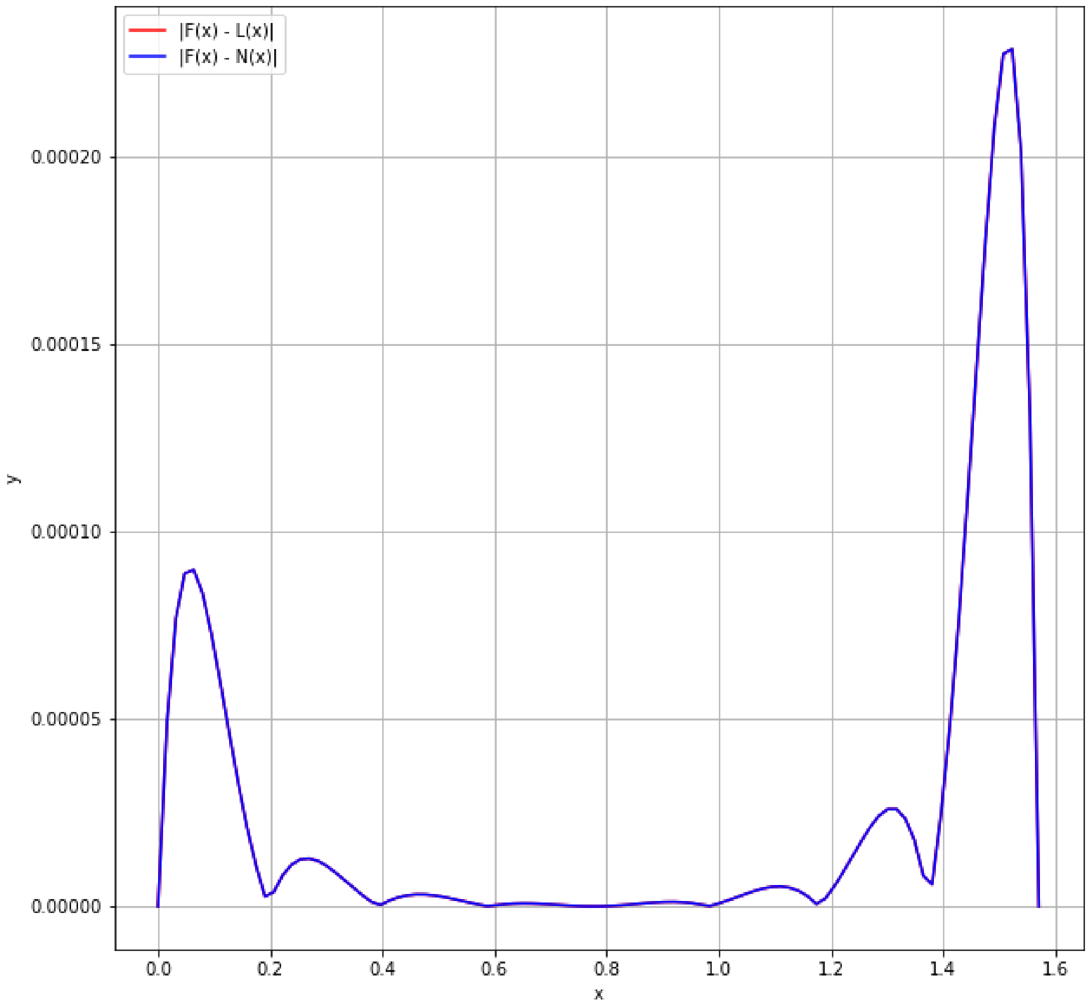
```
9
9
```

То есть подходящим является разбиение на 8 отрезков (9 точек)

Построим на одном чертеже графики интерполирующих многочленов и функции:



Наши графики полностью совпали. Посмотрим ещё раз на графики RN(t) и RL(t):



Погрешность не превышает значения eps = 0.001

Таким образом, нам удалось интерполировать имеющуюся функцию с достаточной точностью, причём оба метода (многочлен Ньютона и многочлен Лагранжа) дали совпадающие значения.