Лабораторная работа 1

**Приближение функций**

**Задание 1.**

1) Написать программу, которая для заданных в варианте функций строит

интерполяционный многочлен Ньютона по равномерной сетке узлов.

2) С помощью написанной программы для каждой из функций построить

интерполяционные многочлены степени

n = 2, 4, 8,16

. Вывести аналитическое

представление многочлена 2-й степени (в форме Ньютона).

3) Для каждой из функций построить 4 графика для сравнения интерполируемой

функции и интерполяционного многочлена (см. пример ниже). Если построение графиков

в вашем языке программирования слишком трудоемко, то можно воспользоваться

сторонними программами. Например: в своей программе сделать таблицу значений

аргумента и соответствующих значений функции, сохранить ее в файл, затем этот файл

импортировать в программу для построения графиков (например, Excel).

**Задание 2.**

1) Написать программу, которая для заданных в варианте функций строит

интерполяционный многочлен Ньютона по чебышевской сетке узлов.

2) С помощью написанной программы для каждой из функций построить

интерполяционные многочлены степени

n = 2, 4, 8,16

. Вывести аналитическое

представление многочлена 2-й степени (в форме Ньютона).

3) Для каждой из функций построить 4 графика для сравнения интерполируемой

функции и интерполяционного многочлена.

**Задание 3.**

1) Написать программу, которая для заданных в варианте функций строит

интерполяционный кубический сплайн по равномерной сетке узлов. В качестве

дополнительных условий использовать значения вторых производных на границах

отрезка. Для решения СЛАУ использовать любой подходящий метод, реализованный в

прошлом семестре, или реализовать подходящий метод заново.

2) С помощью написанной программы для каждой из функций построить

интерполяционные кубические сплайны по n + 1 узлам:

n = 2, 4, 8,16.

3) Для каждой из функций построить 4 графика для сравнения интерполируемой

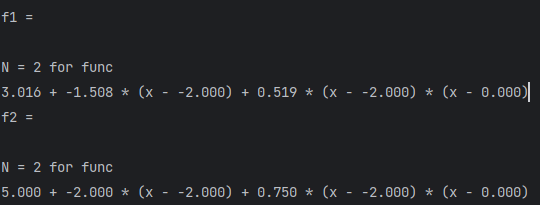
функции и интерполяционного кубического сплайна.

Задание 1

Листинг программы:

import numpy as np  
import matplotlib.pyplot as plt  
  
  
# Заданные функции  
def f1(x):  
 return pow(x, 2) \* np.cos(3 \* x - 1)  
  
  
def f2(x):  
 return np.abs(x \* np.abs(x) - 1)  
  
# Вычисление разделённых разностей  
def divided\_diff(x, y):  
 n = len(x)  
 coef = np.zeros([n, n])  
 coef[:, 0] = y  
  
 for j in range(1, n):  
 for i in range(n - j):  
 coef[i][j] = (coef[i + 1][j - 1] - coef[i][j - 1]) / (x[i + j] - x[i])  
  
 return coef[0]  
  
  
# Построение интерполяционного многочлена Ньютона  
def newton(x, y):  
 coef = divided\_diff(x, y)  
 n = len(x)  
 t = np.linspace(min(x), max(x), 1000)  
 result = np.zeros\_like(t)  
  
 for i in range(n):  
 term = coef[i]  
 for j in range(i):  
 term \*= (t - x[j])  
 result += term  
  
 return t, result  
  
  
def startProgr(min, max, degree):  
 valuesX = np.linspace(min, max, degree)  
 valuesF1 = f1(valuesX)  
 valuesF2 = f2(valuesX)  
 return valuesX, valuesF1, valuesF2  
  
  
def printRes(function, newton, number):  
 print(function + ": ")  
 print(newton.evalf(n=number))  
  
  
def create\_polynomial(x, y):  
 coef = divided\_diff(x, y)  
 if len(x) == 3:  
 print("\nN = 2 for func")  
 print(f"{coef[0]:.3f} + {coef[1]:.3f} \* (x - {x[0]:.3f}) + {coef[2]:.3f} \* (x - {x[0]:.3f}) \* (x - {x[1]:.3f})")  
 return lambda point: sum(coef[i] \* np.prod(point - x[:i]) for i in range(len(x)))  
  
  
xmin, xmax = -2, 2  
degrees = [2, 4, 8, 16]  
  
for degree in degrees:  
 print(f"Степень интерполяции n = {degree}:")  
  
 # Узлы интерполяции  
 valuesX, valuesF1, valuesF2 = startProgr(xmin, xmax, degree+1)  
  
 # Построение интерполяционного многочлена Ньютона для f1(x)  
 t\_f1, newtonF1 = newton(valuesX, valuesF1)  
 t\_f2, newtonF2 = newton(valuesX, valuesF2)  
  
 if degree == 2:  
 print('f1 = ')  
 create\_polynomial(valuesX, valuesF1)  
 print('f2 = ')  
 create\_polynomial(valuesX, valuesF2)  
  
  
 # Построение графиков для многочленов каждой степени  
 plt.figure(figsize=(10, 6))  
  
 # График для f1(x)  
 plt.subplot(2, 2, 1)  
 plt.plot(t\_f1, f1(t\_f1), label='f1(x)')  
 plt.plot(t\_f1, newtonF1, label='Интерп. мн-н Ньютона для f1(x)')  
 plt.title('f1(x)')  
 plt.xlabel('x')  
 plt.ylabel('y')  
 plt.legend()  
 plt.grid(True)  
  
 # График для f2(x)  
 plt.subplot(2, 2, 2)  
 plt.plot(t\_f2, f2(t\_f2), label='f2(x)')  
 plt.plot(t\_f2, newtonF2,label='Интерп. мн-н Ньютона для f2(x)')  
 plt.title('f2(x)')  
 plt.xlabel('x')  
 plt.ylabel('y')  
 plt.legend()  
 plt.grid(True)  
  
 plt.tight\_layout()  
 plt.show()

Многочлены для n = 2:

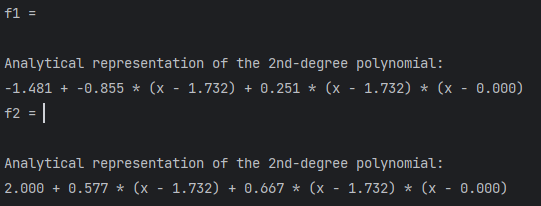


Задание 2

Листинг программы:

import numpy as np  
import matplotlib.pyplot as plt  
  
  
# Заданные функции  
def f1(x):  
 return pow(x, 2) \* np.cos(3 \* x - 1)  
  
  
def f2(x):  
 return np.abs(x \* np.abs(x) - 1)  
  
  
def chebyshev(min, max, degree):  
 nodes = np.zeros(degree)  
 for i in range(degree):  
 nodes[i] = 0.5 \* (xmin + xmax) + 0.5 \* (xmax - xmin) \* np.cos((2 \* i + 1) \* np.pi / (2 \* degree))  
 return nodes  
  
# Вычисление разделённых разностей  
def divided\_diff(x, y):  
 n = len(x)  
 coef = np.zeros([n, n])  
 coef[:, 0] = y  
  
 for j in range(1, n):  
 for i in range(n - j):  
 coef[i][j] = (coef[i + 1][j - 1] - coef[i][j - 1]) / (x[i + j] - x[i])  
  
 return coef[0]  
  
  
# Построение интерполяционного многочлена Ньютона  
def newton(x, y):  
 coef = divided\_diff(x, y)  
 n = len(x)  
 t = chebyshev(min(x), max(x), 1000)  
 result = np.zeros\_like(t)  
  
 for i in range(n):  
 term = coef[i]  
 for j in range(i):  
 term \*= (t - x[j])  
 result += term  
  
 return t, result  
  
  
def startProgr(min, max, degree):  
 valuesX = chebyshev(min, max, degree)  
 valuesF1 = [f1(value) for value in valuesX]  
 valuesF2 = [f2(value) for value in valuesX]  
 return valuesX, valuesF1, valuesF2  
  
  
def printRes(function, newton, number):  
 print(function + ": ")  
 print(newton.evalf(n=number))  
  
  
def create\_polynomial(x, y):  
 coef = divided\_diff(x, y)  
 if len(x) == 3:  
 print("\nAnalytical representation of the 2nd-degree polynomial:")  
 print(f"{coef[0]:.3f} + {coef[1]:.3f} \* (x - {x[0]:.3f}) + {coef[2]:.3f} \* (x - {x[0]:.3f}) \* (x - {x[1]:.3f})")  
 return lambda point: sum(coef[i] \* np.prod(point - x[:i]) for i in range(len(x)))  
  
  
  
xmin, xmax = -2, 2  
degrees = [2, 4, 8, 16]  
  
for degree in degrees:  
 print(f"Степень интерполяции n = {degree}:")  
  
 # Узлы интерполяции  
 valuesX, valuesF1, valuesF2 = startProgr(xmin, xmax, degree+1)  
  
 # Построение интерполяционного многочлена Ньютона для f1(x)  
 t\_f1, newtonF1 = newton(valuesX, valuesF1)  
 t\_f2, newtonF2 = newton(valuesX, valuesF2)  
  
 if degree == 2:  
 print('f1 = ')  
 create\_polynomial(valuesX, valuesF1)  
 print('f2 = ')  
 create\_polynomial(valuesX, valuesF2)  
  
 # Построение графиков для многочленов каждой степени  
 plt.figure(figsize=(10, 6))  
  
 # График для f1(x)  
 plt.subplot(2, 2, 1)  
 plt.plot(t\_f1, f1(t\_f1), label='f1(x)')  
 plt.plot(t\_f1, newtonF1, label='Интерп. мн-н Ньютона для f1(x)')  
 plt.title('f1(x)')  
 plt.xlabel('x')  
 plt.ylabel('y')  
 plt.legend()  
 plt.grid(True)  
  
 # График для f2(x)  
 plt.subplot(2, 2, 2)  
 plt.plot(t\_f2, f2(t\_f2), label='f2(x)')  
 plt.plot(t\_f2, newtonF2,label='Интерп. мн-н Ньютона для f2(x)')  
 plt.title('f2(x)')  
 plt.xlabel('x')  
 plt.ylabel('y')  
 plt.legend()  
 plt.grid(True)  
  
 plt.tight\_layout()  
 plt.show()

Многочлены для n = 2:



Задание 3

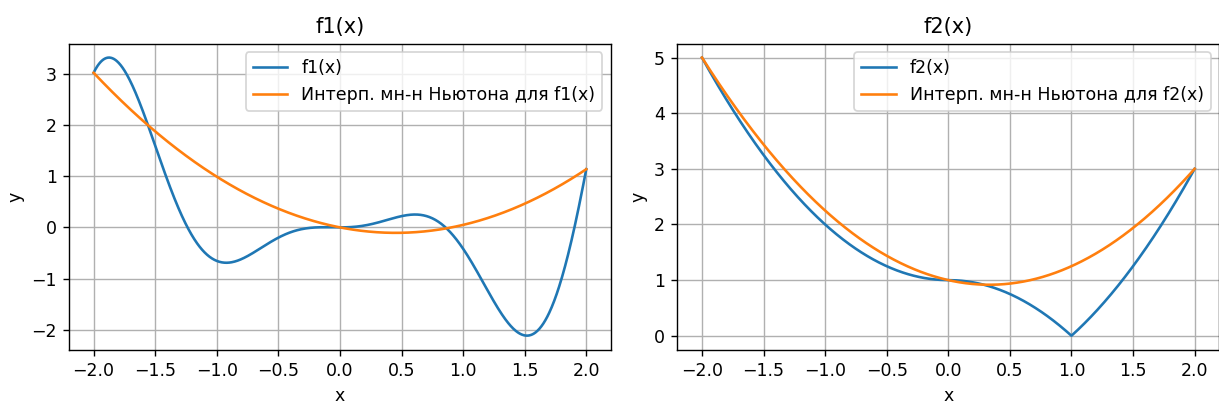
Листинг программы :

import numpy as np  
from scipy.interpolate import CubicSpline  
import matplotlib.pyplot as plt  
  
  
# Заданные функции  
def f1(x):  
 return pow(x, 2) \* np.cos(3 \* x - 1)  
  
  
def f2(x):  
 return np.abs(x \* np.abs(x) - 1)  
  
  
# Функция для построения интерполяционного кубического сплайна  
def cubic\_spline\_interpolation(x, y, d2y\_start, d2y\_end):  
 cs = CubicSpline(x, y, bc\_type=((2, d2y\_start), (2, d2y\_end)))  
 return cs  
  
  
def startProgr(min, max, degree):  
 valuesX = np.linspace(min, max, degree)  
 valuesF1 = [f1(value) for value in valuesX]  
 valuesF2 = [f2(value) for value in valuesX]  
 return valuesX, valuesF1, valuesF2  
  
# Заданные узлы для интерполяции  
xmin, xmax = -2, 2  
degrees = [2, 4, 8, 16]  
  
# Для каждой функции  
  
for degree in degrees:  
  
 valuesX, valuesF1, valuesF2 = startProgr(xmin, xmax, degree+1)  
 # Значения вторых производных на границах  
  
 d2y\_f1\_start = f1(valuesX[1]) - 2 \* f1(valuesX[0]) + f1(valuesX[1])  
 d2y\_f1\_end = f1(valuesX[-2]) - 2 \* f1(valuesX[-1]) + f1(valuesX[-2])  
  
 d2y\_f2\_start = f2(valuesX[1]) - 2 \* f2(valuesX[0]) + f2(valuesX[1])  
 d2y\_f2\_end = f2(valuesX[-2]) - 2 \* f2(valuesX[-1]) + f2(valuesX[-2])  
  
 # Построение интерполяционного кубического сплайна  
 cs\_f1 = cubic\_spline\_interpolation(valuesX, valuesF1, d2y\_f1\_start, d2y\_f1\_end)  
 cs\_f2 = cubic\_spline\_interpolation(valuesX, valuesF2, d2y\_f2\_start, d2y\_f2\_end)  
  
 # Оценка значений сплайна на более широком диапазоне для построения графика  
 lineValuesX = np.linspace(xmin, xmax, 500)  
 y\_interp\_f1 = cs\_f1(lineValuesX)  
 y\_interp\_f2 = cs\_f2(lineValuesX)  
  
 # Построение графиков  
 # График для f1(x)  
 plt.subplot(2, 2, 1)  
 lineValuesX = np.linspace(xmin, xmax, 500)  
 plt.plot(lineValuesX, [f1(x) for x in lineValuesX], label='f1(x)')  
 plt.plot(lineValuesX, y\_interp\_f1, label='Интерп. мн-н Ньютона для f1(x)')  
 plt.title('f1(x)')  
 plt.xlabel('x')  
 plt.ylabel('y')  
 plt.legend()  
 plt.grid(True)  
  
 # График для f2(x)  
 plt.subplot(2, 2, 2)  
 plt.plot(lineValuesX, [f2(x) for x in lineValuesX], label='f2(x)')  
 plt.plot(lineValuesX, y\_interp\_f2, label='Интерп. мн-н Ньютона для f2(x)')  
 plt.title('f2(x)')  
 plt.xlabel('x')  
 plt.ylabel('y')  
 plt.legend()  
 plt.grid(True)  
  
 plt.tight\_layout()  
 plt.show()

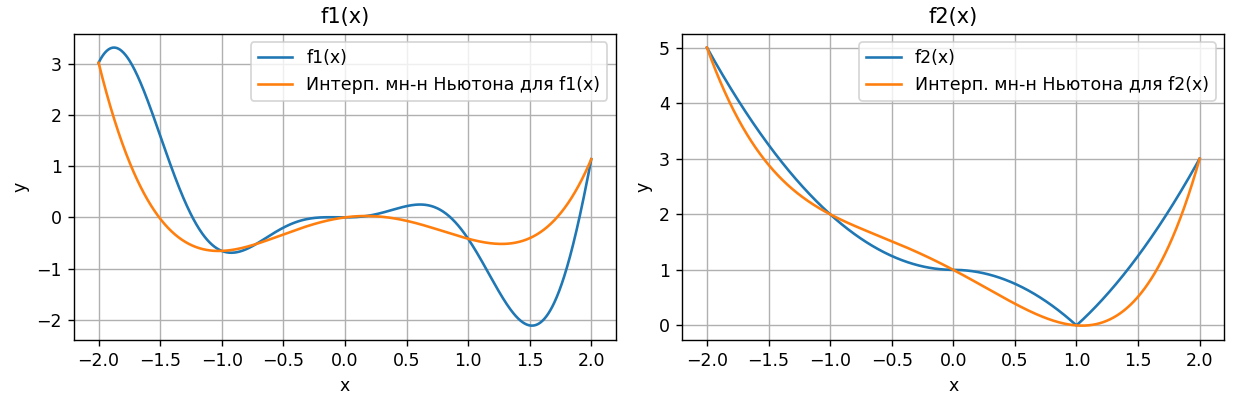
Графики

Задание 1

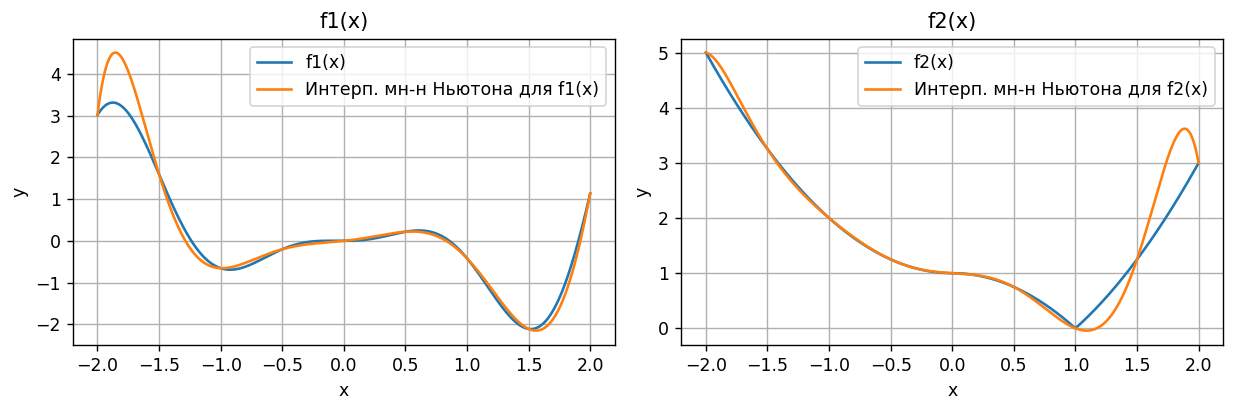
N = 2



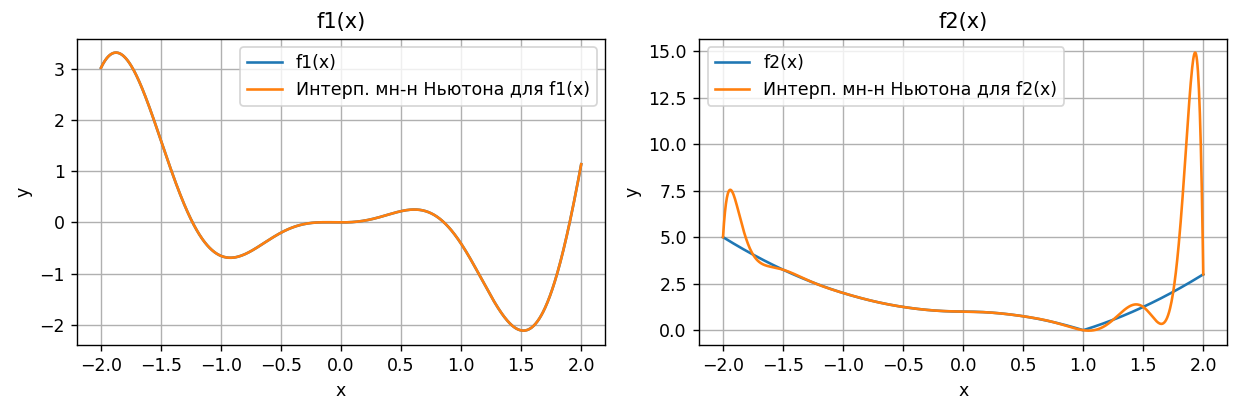
N=4



N=8

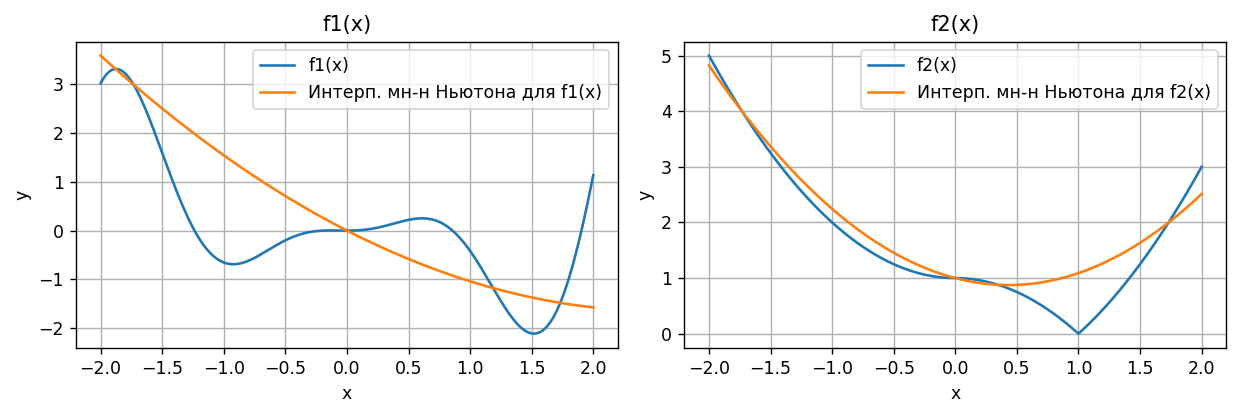


N=16

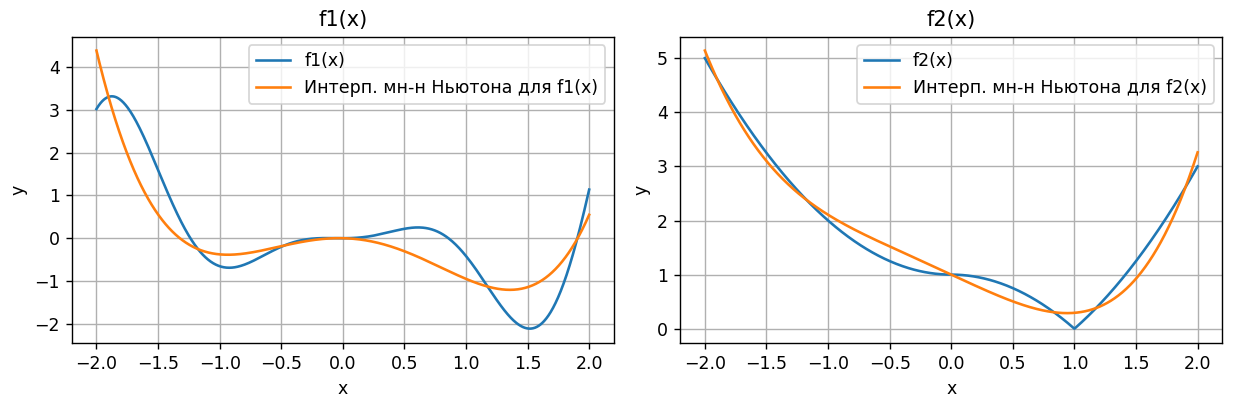


Задание 2

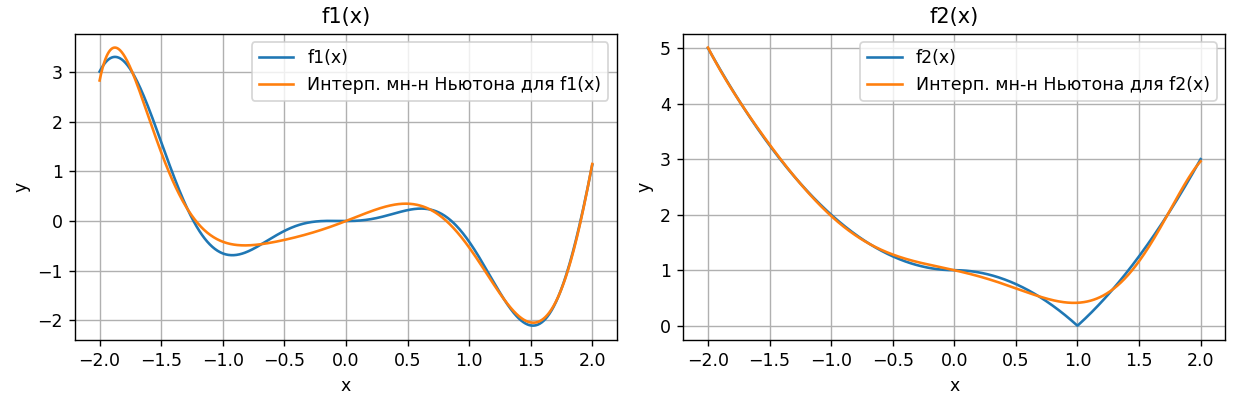
N = 2



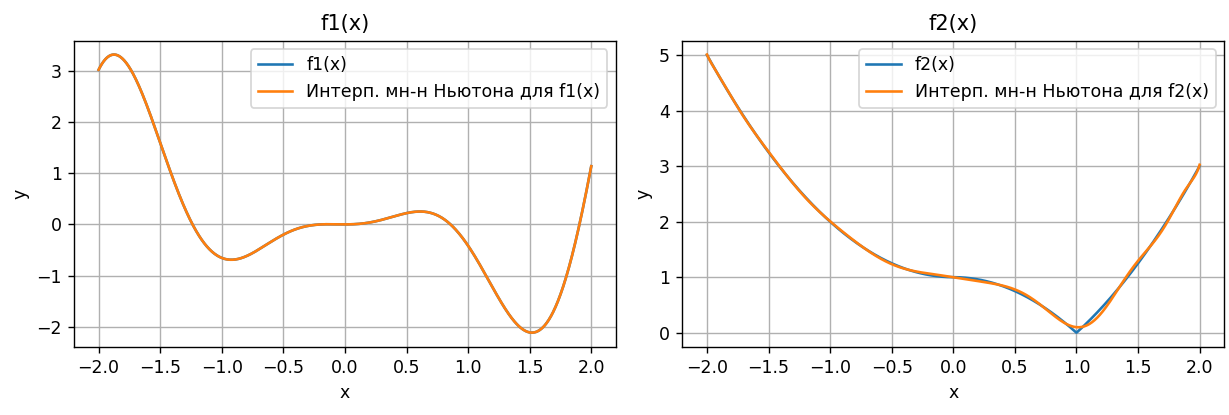
N = 4



N = 8

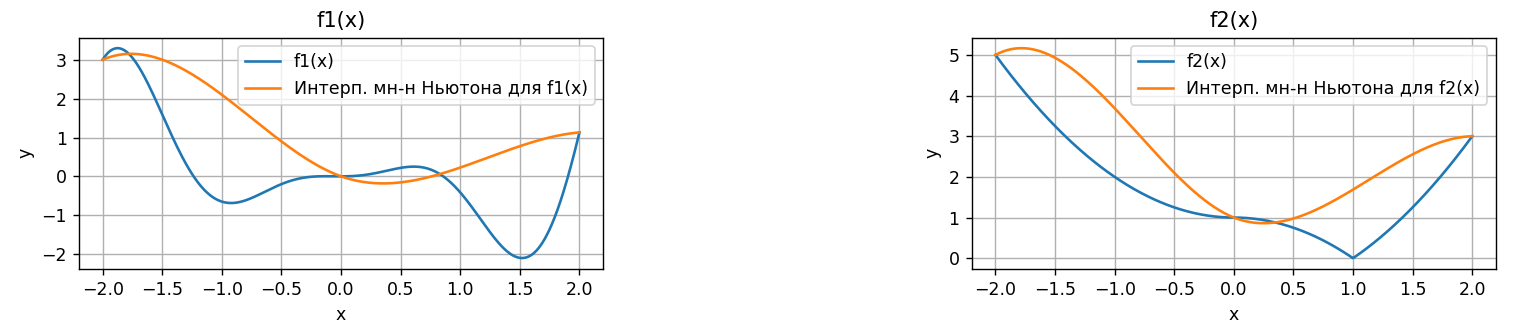


N = 16

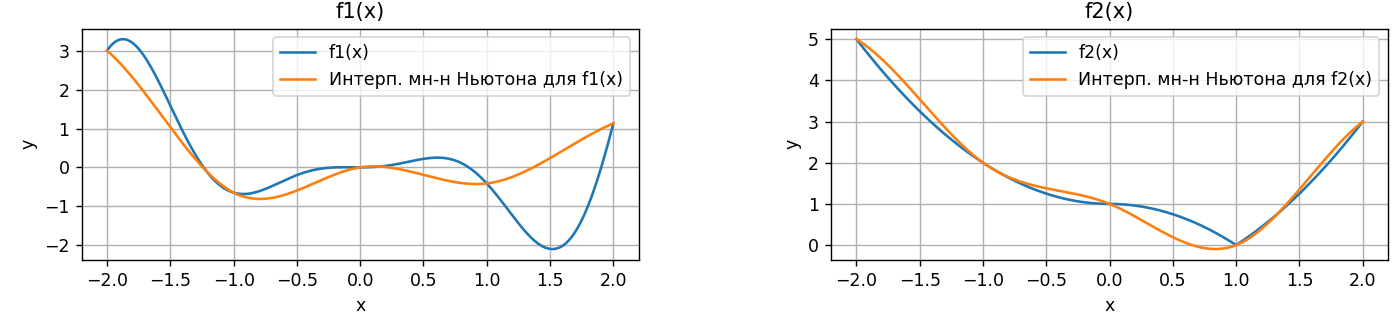


Задание 3

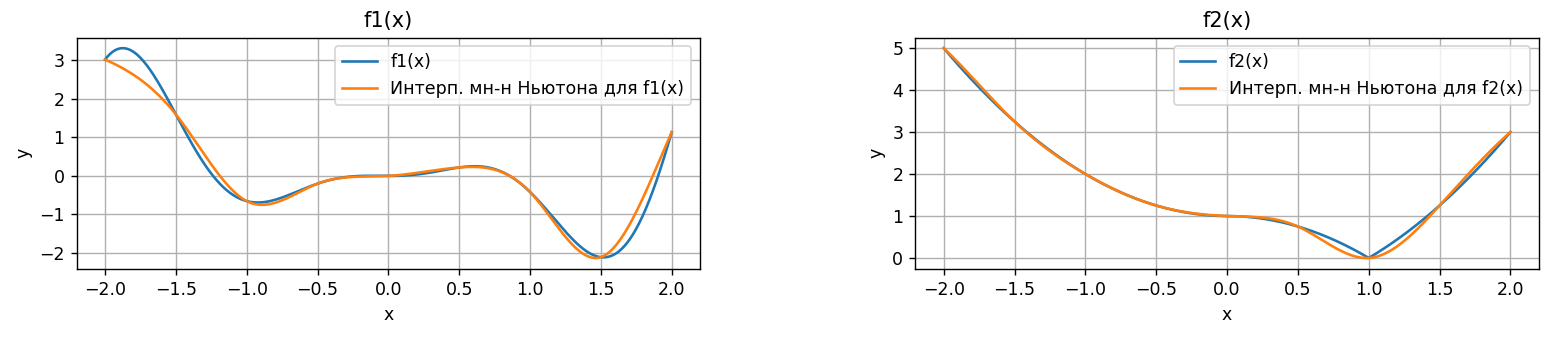
N = 2



N = 4



N = 8



N = 16

