## Национальный исследовательский университет «МЭИ» (Московский Энергетический Институт)

**Кафедра математического и компьютерного моделирования**

## Численные методы

## Лабораторная работа №4:

## «Приближение функций»

*Выполнил:* Солонин Е. В. А-14-19

*Преподаватель:* Амосова О. А.

Вариант 52

2021 Москва

**Постановка задачи:**

**Задача 4.1.**  В таблице 4.1 приведены данные о численности населения некоторых крупнейших стран мира по годам с 1950 -2000 г.г. Заполнить последние два столбца таблицы (взять сведения из интернета). На основе этих данных для конкретного варианта построить наилучший многочлен по МНК. Найти численность населения страны в 2019 году и сравнить полученное значение с актуальным значением (взять из интернета).

Решить ту же задачу на основе интерполяционного многочлена. То есть построить интерполяционный многочлен по значениям с 1950-2020 г.г. Вычислить значение для 2019 года и сравнить с актуальными данными. Составить отчет по задаче.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | Страна | 1950 | 1960 | 1970 | 1980 | 1990 | 2000 | 2010 | 2020 | 2019 |
| 4.1.52 | Ангола | 4.5 | 5.2 | 6.2 | 8.1 | 11 | 14.8 | 23,36 | 32,52 | 31,83 |

**Код программы** *(пояснения к коду красным цветом)***:**

import numpy as np  
  
#составление матрицы A сумм для решения системы Ax=b для нахождения коэффициентов МНК

def create\_factor\_mnk(n, data):  
 A = []  
 m = len(x)  
 for i in range(n):  
 row = []  
 for j in range(n):  
 s = 0  
 for k in range(m):  
 s += np.power(data[k], i + j)  
 row.append(s)  
 A.append(np.array(row))  
 return np.array(A)  
  
#составление вектора b сумм для решения системы Ax=b для нахождения коэффициентов МНК

def create\_b\_mnk(n, x, y):  
 b = []  
 m = len(x)  
 for j in range(n):  
 s = 0  
 for k in range(m):  
 s += np.power(x[k], j) \* y[k]  
 b.append(s)  
 return np.array(b)

#аппроксимация функции МНК в конкретной точке(value)   
  
def least\_squares(n, x, y, value):  
 A = create\_factor\_mnk(n+1, x)  
 b = create\_b\_mnk(n+1, x, y)  
 factor = np.linalg.solve(A, b) #коэффициенты многочлена МНК   
 sig = sigma(x, y, factor) #среднее квадратичное отклонение   
 return polynomial(factor, value), sig  
  
#интерполяция функции многочленом Лагранжа в конкретной точке(value)

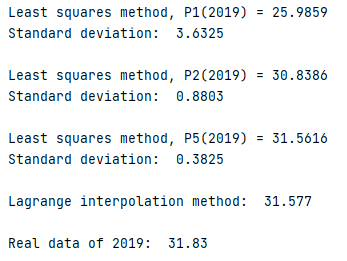
def interpolation\_lagrange(n, x, y, value):  
 L = 0  
 for i in range(n+1):  
 mul = 1  
 for j in range(n+1):  
 if i != j:  
 mul \*= (value - x[j]) / (x[i] - x[j])  
 L += y[i] \* mul  
 return L  
  
#нахождение среднего квадратичного отклонения

def sigma(x, y, factor):  
 s = 0  
 for k in range(len(x)):  
 s += (y[k]-polynomial(factor, x[k]))\*\*2  
 return np.sqrt(1/len(x) \* s)

#нахождние значения многочлена МНК по коэффициентам в конкретной точке(value)   
  
def polynomial(factor, value):  
 s = 0  
 for k in range(len(factor)):  
 s += np.power(value, k) \* factor[k]  
 return s  
  
#вывод значения многочлена МНК степени power

def P(power):  
 value, deviation = least\_squares(power, x, y, 2.019)  
 print(**'Least squares method, P'** + power.\_\_str\_\_() + **'(2019)'**, **'='**, round(value, 4))  
 print(**'Standard deviation: '**, round(deviation, 4))  
 print()  
  
  
x = list(np.arange(1.950, 2.021, 0.01))  
y = list([4.5, 5.2, 6.2, 8.1, 11, 14.8, 23.36, 32.52])  
  
angola\_2019 = 31.83 #реальные данные 2019 года  
interpolation = interpolation\_lagrange(2, x[5:], y[5:], 2.019)  
  
P(1) #многочлен МНК степени 1   
P(2) #многочлен МНК степени 2  
P(5) #многочлен МНК степени 5  
  
print(**'Lagrange interpolation method: '**, round(interpolation, 4))  
print()  
print(**'Real data of 2019: '**, angola\_2019)

**Результаты выполнения:**

****

**Анализ результатов:**

Для коректных вычислений, из-за особенностей хранения чисел в памяти компьютера необходимо представить значения годов в виде e-3, например, 2.019 или 1.950. Это продиктовано тем, что возводя, например, 2019 в 5 степень мы получаем число порядка 10^17, что вызывает переполнение в памяти.

Возьмем 1, 2 и 5 степени многочлена по МНК и выполним апроксимацию для значения 2019 года. Вычисляя среднее квадратичное отклонение для каждой степени, убеждаемся, что наилучший многочлен по МНК – 5ой степени.

Для интерполяции будем использовать многочлен Лагранжа 3-ей степени (выбираем значения 2000, 2010 и 2020 года) для получения более точного приближения.

Таким образом, с помощью многочлена Лагранжа были получены более точные данные, чем с помощью многочлена по МНК для нахождения численности населения Анголы в 2019 году.

**Постановка задачи:**

**Задача 4.2.** Дана функция . Приблизить функцию методом интерполяции, используя многочлен Лагранжа. Степень многочлена N подобрать таким образом, чтобы максимальная величина погрешности на отрезке  не превышала заданной величины Построить графики многочленов и графики погрешностей .Приблизить функцию методом интерполяции, указанным в индивидуальном варианте. Сравнить полученные результаты.

**ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ**

1. Задать исходный отрезок , и  - число отрезков разбиения (для начала  взять произвольно), функцию .

2. Составить таблицу значений функции в  точке отрезка, то есть задать массивы x и y исходных данных.

3. Составить подпрограмму, выполняющую вычисление функции в произвольной точке t отрезка с помощью интерполяционного многочлена Лагранжа L(t) и , указанного в индивидуальном варианте, многочлена P(t).

4. Построить график функции погрешности  и RL(t) и по графику определить максимальную величину полученной погрешности приближения. Если точность не достигнута, то увеличить число узлов интерполяции. Найти значение n=N, при котором точность достигается.

5. На одном чертеже построить графики интерполирующих многочленов и исходной функции .

6. Оформить отчет по задаче.

|  |  |
| --- | --- |
| 4.2.52 | |
|  |  |
| 2, 12, 22, 32, 42, 52 | Многочлен Ньютона с конечными разностями |

**Код программы** *(пояснения к коду красным цветом)***:**

import numpy as np  
import math  
import matplotlib.pyplot as plt  
  
#

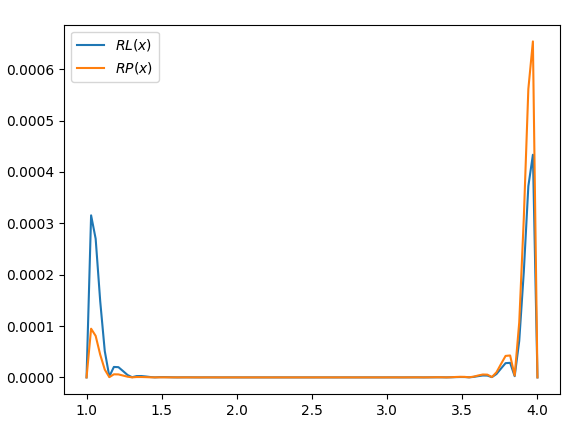
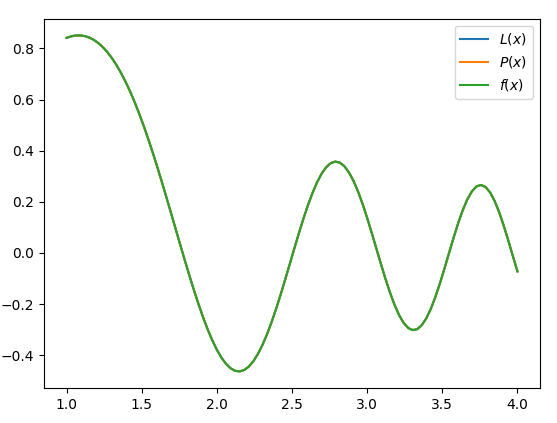
def f(x):  
 return np.sin(np.power(x, 2))/x

#погрешность многочлена Ньютона с конечными разностями  
def r\_p(t):  
 return np.abs(f(t) - Neuton(t))  
  
#погрешность многочлена Лагранжа   
def r\_l(t):  
 return np.abs(f(t) - Lagrange(t))  
  
#интерполяция функции многочленом Лагранжа в конкретной точке(value)

def Lagrange(value):  
 lagrange = 0  
 for i in range(n + 1):  
 mul = 1  
 for j in range(n + 1):  
 if i != j:  
 mul \*= (value - x[j]) / (x[i] - x[j])  
 lagrange += y[i] \* mul  
 return lagrange

#все конечные разности для многочлена Ньютона  
  
def delta\_y():  
 difs = list()  
 difs.append(y)  
 for i in range(n):  
 row = []  
 for j in range(0, len(difs[i])-1):  
 row.append(difs[i][j+1] - difs[i][j])  
 difs.append(list(row))  
 return difs  
  
#интерполяция функции многочленом Ньютона в конкретной точке(value)

def Neuton(value):  
 index, nearest = min(enumerate([a, b]), key=lambda arr: abs(value - arr[1])) #выбор направления интерполирования(вперед/назад) в зависимости от значения  
 q = (value - nearest) / step  
 diffs = delta\_y()  
 P = 0  
 for i in range(n):  
 mul = 1  
 for j in range(i):  
 mul \*= (q-j) if index == 0 else (q+j)  
 if index != 0:  
 index = len(diffs[i])-1  
 mul \*= diffs[i][index] / math.factorial(i)  
 P += mul  
 return P  
  
  
a = 1  
b = 4  
epsilon = 0.001  
n = 20  
step = (b - a) / n  
  
x = list(np.arange(a, b+0.1\*step, step))  
y = list(f(x))  
  
fig = plt.subplots()  
x\_data = np.linspace(a, b, 100)  
y\_data\_l = np.zeros(100)  
y\_data\_p = np.zeros(100)  
for i in range(100):  
 y\_data\_l[i] = r\_l(x\_data[i])   
 y\_data\_p[i] = r\_p(x\_data[i])   
  
plt.plot(x\_data, y\_data\_l, label=**f'$RL(x)$'**) #график погрешности   
plt.plot(x\_data, y\_data\_p, label=**f'$RP(x)$'**) #график погрешности   
plt.legend()  
plt.show()  
  
fig = plt.subplots()  
for i in range(100):  
 y\_data\_l[i] = Lagrange(x\_data[i])  
 y\_data\_p[i] = Neuton(x\_data[i])  
  
plt.plot(x\_data, y\_data\_l, label=**f'$L(x)$'**) #график   
plt.plot(x\_data, y\_data\_p, label=**f'$P(x)$'**) #график   
plt.plot(x\_data, f(x\_data), label=**f'$f(x)$'**) #график   
plt.legend()  
plt.show()

**Результаты выполнения:**

**Анализ результатов:**

Для интерполяции функции методом Ньютона с конечными разностями для конкретного значения будем определять направление интерполирования – вперед или назад, для получения лучшей точности. Так, для нашего отрезка [1; 4]: интерполирование вперед будет применяться на [1; 2.5], назад – [2.5; 4].

Для достижения заданной точности необходимо увеличить число узлов интерполяции до 20. При этом максимальная погрешность для многочлена Лагранжа – примерно 0.0004 < 0.001. Для многочлена Ньютона – 0.0006 < 0.001. Кроме того погрешность минимальна в середине отрезка, и максимальна вблизи границ отрезка.

Графики многочленов и функции совпадают при данной масштабе и числе узлов интерполяции.

**Постановка задачи:**

**Задача 4.3.** Задана функция , определенная на отрезке [-1;1]. Требуется разложить функцию в ряд Тейлора в окрестности нуля с точностью  и произвести экономизацию полученного степенного ряда.

**ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ:**

**1.** Определить функцию , вычисляющую частичную сумму ряда по коэффициентам  .

**2.** Вычислить коэффициенты разложения и определить требуемое количество слагаемых для достижения требуемой точности, построив график.

**3**. Произвести экономизацию степенного ряда до тех пор, пока сохраняется необходимая точность (см. приложение).

**4**. Построить график погрешности каждого этапа экономизации.

|  |  |
| --- | --- |
| N |  |
| 4.3.22  4.3.52 |  |

**Теоретический материал**

Многочлены Чебышёва  могут быть определены с помощью рекуррентного соотношения , , .

Формулы экономизации степенного ряда:

, , ,  , 

, , 

, 

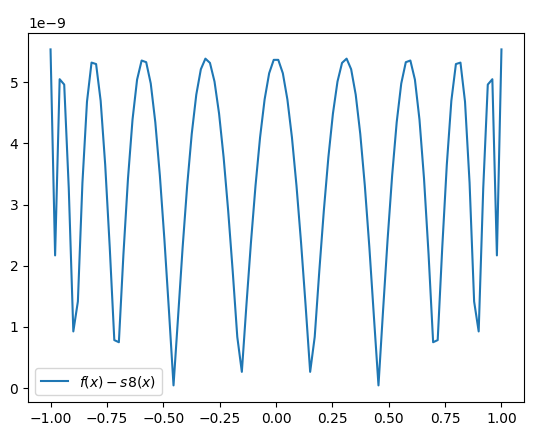
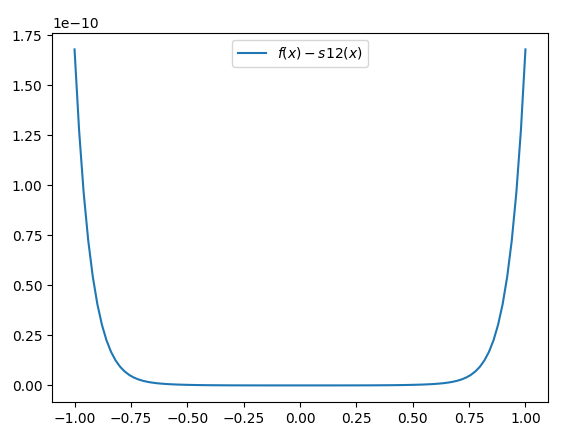
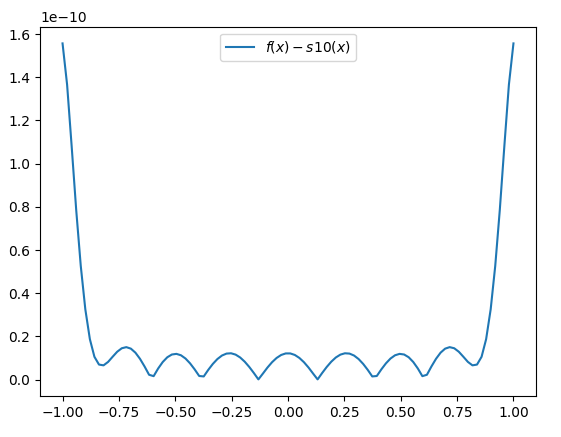
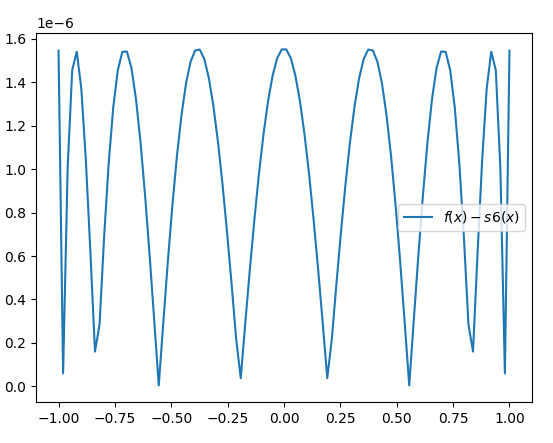
**Код программы:**

import numpy as np  
import math  
import matplotlib.pyplot as plt  
  
  
#

def f(x):  
 return x\*np.sin(x)  
  
#частичная сумма n членов ряда в точке x  
def s(x, n):  
 sum = 0  
 for i in range(n+1):  
 sum += c[i] \* np.power(x, i)  
 return sum  
  
#коэффициенты разложения в ряд Тейлора  
def factor(n):  
 c = [0]  
 for i in range(1, n+1):  
 c.append(np.round(np.sin((i-1)\*np.pi/2) / math.factorial(i-1), 11))  
 return c  
  
#многочлен Чебышева n-ой степени в точке x  
def T(n, x):  
 if n == 0:  
 return 1  
 if n == 1:  
 return x  
 return 2 \* x \* T(n - 1, x) - T(n - 2, x)  
  
#формулы для экономизации ряда  
def eco(x, power):  
 units = {  
 **'1'**: T(1, x),  
 **'2'**: 1 / 2 \* (1 + T(2, x)),  
 **'3'**: 1 / 4 \* (3 \* x + T(3, x)),  
 **'4'**: 1 / 8 \* (8 \* x \*\* 2 - 1 + T(4, x)),  
 **'5'**: 1 / 16 \* (20 \* x \*\* 3 - 5 \* x + T(5, x)),  
 **'6'**: 1 / 32 \* (48 \* x \*\* 4 - 18 \* x \*\* 2 + 1 + T(6, x)),  
 **'7'**: 1 / 64 \* (112 \* x \*\* 5 - 56 \* x \*\* 3 + 7 \* x + T(7, x)),  
 **'8'**: 1 / 128 \* (256 \* x \*\* 6 - 160 \* x \*\* 4 + 32 \* x \*\* 2 - 1 + T(8, x)),  
 **'9'**: 1 / 256 \* (576 \* x \*\* 7 - 432 \* x \*\* 5 + 120 \* x \*\* 3 - 9 \* x + T(9, x)),  
 **'10'**: 1 / 512 \* (1280 \* x \*\* 8 - 1120 \* x \*\* 6 + 400 \* x \*\* 4 - 50 \* x \*\* 2 + 1 + T(10, x)),  
 **'11'**: 1 / 1024 \* (2816 \* x \*\* 9 - 2816 \* x \*\* 7 + 1232 \* x \*\* 5 - 220 \* x \*\* 3 + 11 \* x + T(11, x)),  
 **'12'**: 1 / 2048 \* (6144 \* x \*\* 10 - 6912 \* x \*\* 8 + 3584 \* x \*\* 6 - 840 \* x \*\* 4 + 72 \* x \*\* 2 - 1 + T(12, x))  
 }  
 return units[power.\_\_str\_\_()]  
  
#шаг экономизации  
def eco\_step(n):  
 fig = plt.subplots()  
 for i in range(100):  
 s\_data[i] = s(x[i], n) + s\_data[i] - s(x[i], n + 2) + c[n + 2] \* eco(x[i], n + 2) - c[n + 2] \* T(n + 2, x[i]) / 2 \*\* (n + 1)  
  
 plt.plot(x, np.abs(f(x) - s\_data), label=**f'$f(x)-s**{n}**(x)$'**)  
 plt.legend()  
 plt.show()  
  
  
a = -1  
b = 1  
n = 12  
c = factor(n)  
print(c)  
  
fig = plt.subplots()  
x = np.linspace(a, b, 100)  
s\_data = np.zeros(100)  
for i in range(100):  
 s\_data[i] = s(x[i], n)

#График погрешности   
plt.plot(x, np.abs(f(x) - s\_data), label=**f'$f(x)-s**{n}**(x)$'**)   
plt.legend()  
plt.show()

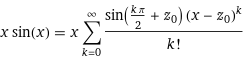
#Три шага экономизации  
eco\_step(10)  
eco\_step(8)  
eco\_step(6)

**Результаты выполнения:**

Коэффициенты ряда Тейлора

**Анализ результатов:**

Для функции разложение в ряд Тейлора в точке 0: x^2 - x^4/6 + x^6/120 - x^8/5040 + x^10/362880 - x^12/39916800 + O(x^13)
(Taylor series)

Коэффициенты могут быть вычислены по формуле:  , при .

Для вычисления с точностью необходимо разложение до n=12.

Заметим, что вес имеют только коэффициенты четных степеней, что позволит нам проводить экономизацию только для четных значений n. Например, при экономизации до n=11 мы получим те же результаты, что и при n=12, т.к. коэффициент при нулевой.

Выполнив первый этап экономизации до n=10, точность немного уменьшается, однако остается в пределах допустимой.

Второй этап экономизации до n=8 - точность уменьшается на порядок, но еще находится в рамках .

На третьем этапе (n=6) точность снижается до .

Таким образом, мы провели два этапа экономизации с n=12 до n=8.