**Министерство образования Республики Беларусь**

**Белорусский государственный университет**

Факультет прикладной математики и информатики

Лабораторная работа №2

По курсу «Численные методы»

**Приближение функций**

Вариант №9

Работу выполнил:

студент 3 курса 7 группы

**Шатерник Артём**

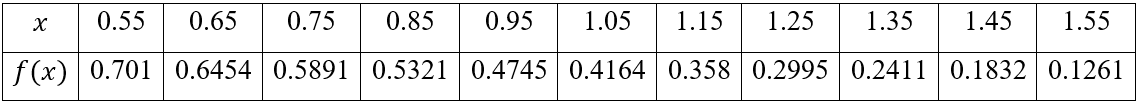
Преподаватель:

**Будник А. М**.

**Минск 2024**

1. **Постановка задачи.**

На отрезке восстановить значения функции

В точках по узлам:

Используя метод наименьших квадратов, многочлен Лагранжа, многочлен Ньютона, многочлен Чебышева и интерполирование в конце таблицы.

1. **Алгоритм решения.**
2. **Метод наименьших квадратов.**

Будем строить полином степени :

При табличном задании функции коэффициенты находим из системы уравнений:

Решать полученную систему будем обычным методом Гаусса.

Оценка погрешности:

Истинное значение погрешности будем считать в этом и дальнейших методах по формуле:

где точное значение в точке, приближённое значение.

Теоретическую погрешность для МНК найдём по формуле:

1. **Многочлен Лагранжа.**

Строим по формуле:

Оценка погрешности по формуле:

где

и

1. **Многочлен Ньютона.**

Строим по формуле:

где

разделённые разности порядка.

Оценка погрешности аналогична оценке погрешности для метода Лагранжа, так как оба этих метода строят один и тот же многочлен.

Также оценку можно провести по формуле:

1. **Многочлены Чебышева.**

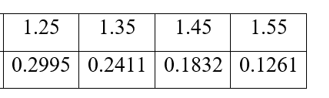
Для минимизации остатка интерполирования воспользуемся чебышёвской сеткой узлов, для перехода к которой используем формулу:

Далее строим многочлен Ньютона на новой сетке.

Оценки погрешности:

1. **Интерполирование в конце таблицы.**

Для интерполирования в конце таблицы будем строить многочлен степени, для чего будем использовать 4 последних узла:



Находить приближённое значение будем только для точки , так как только она лежит между этими узлами.

Сделаем замену вида:

учитывая, что и .

Далее будем строить многочлен по формуле:

Оценка погрешности:

где

1. **Листинг программы.**

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

**# Метод Гаусса  
def gaussFunc(matrix):  
 copy\_matrix = np.copy(matrix)  
 for nrow, row in enumerate(copy\_matrix):  
 divider = row[nrow]  
 row /= divider  
 for lower\_row in copy\_matrix[nrow+1:]:  
 factor = lower\_row[nrow]  
 lower\_row -= factor \* row  
 return copy\_matrix  
  
def gauss\_reverse(matrix):  
 n\_row=matrix.shape[0]  
 x= [None] \* n\_row  
 for i in range(n\_row-1, -1,-1):  
 x[i]=matrix[i,-1]-np.dot(matrix[i, i+1:n\_row], x[i+1:])  
 return np.array(x)  
  
def gauss\_method(matrix):  
 return gauss\_reverse(gaussFunc(matrix))**

**# Отрезок  
my\_N = 9  
a = 0.1 + 0.05 \* my\_N  
b = 1 + a  
print(f"a: {a} \nb: {b}")**

**# Узлы  
n = 10  
h = 1 / 10  
nodes = [round(a + i \* h, 3) for i in range(0, n + 1)]  
print("Узлы:", nodes)**

**# Значения в узлах  
f\_nodes = [a \* math.e\*\*(-x) + (1 - a) \* math.cos(x) for x in nodes]  
print("Значения в узлах:\n", f\_nodes)  
for x in f\_nodes:  
 print(round(x, 4), end=', ')**

**# Точки для восстановления  
x\_find = [nodes[0] + 2 \* h / 3, nodes[int(n / 2)] + 0.5 \* h, nodes[n] - h / 3]  
print("Точки для восстановления:\n", x\_find)**

**# Метод наименьших квадратов (МНК)  
m = int(n / 2)  
print("Степень полинома:", m)  
  
def MNK(nodes, funcs, m):  
 n = len(nodes)  
 matrix = np.zeros((m + 1, m + 2))  
 for k in range(0, m + 1):  
 for i in range(0, m + 1):  
 for j in range(0, n):  
 matrix[k][i] += nodes[j]\*\*(i + k)  
 for i in range(0, n):  
 matrix[k][m + 1] += funcs[i] \* nodes[i]\*\*k  
 return gauss\_method(matrix)  
  
  
coeff = MNK(nodes, f\_nodes, m)  
print("Коэффициенты многочлена:\n", coeff)**

**def func(x, a):  
 return a \* math.e\*\*(-x) + (1 - a) \* math.cos(x)  
  
def polynom(x, coeffs):  
 res = 0  
 for i in range(len(coeffs)):  
 res += x\*\*i \* coeffs[i]  
 return res**

**# Значения в точках восстановления  
y\_find\_MNK = [polynom(i, coeff) for i in x\_find]  
print("Восстановленные значения:", y\_find\_MNK)**

**# Истинная погрешность  
r\_MNK = [abs(func(i, a) - polynom(i, coeff)) for i in x\_find]  
print("Истинная погрешность:", r\_MNK)**

**# Теоретическая погрешность  
delta\_f\_MNK = 0  
for i in range(len(nodes)):  
 delta\_f\_MNK += (f\_nodes[i] - polynom(nodes[i], coeff))\*\*2  
delta\_f\_MNK = math.sqrt(delta\_f\_MNK)  
print("Погрешность:", delta\_f\_MNK)**

**# Многочлен Лагранжа  
def lagrange\_interp(nodes, funcs, find\_x):  
 results = []  
 n = len(nodes)  
 for x in find\_x:  
 pol = 0  
 for i in range(n):  
 base = 1  
 for j in range(n):  
 if i != j:  
 base \*= (x - nodes[j]) / (nodes[i] - nodes[j])  
 pol += base \* funcs[i]  
 results.append(pol)  
 return results  
  
y\_find\_lagrange = lagrange\_interp(nodes, f\_nodes, x\_find)  
print("Восстановленные значения:", y\_find\_lagrange)**

**# Истинная погрешность  
y\_real = [func(x, a) for x in x\_find]  
r\_lagrange = [abs(y\_real[i] - y\_find\_lagrange[i]) for i in range(len(x\_find))]  
print("Истинная погрешность:", r\_lagrange)**

**# Теоретическая погрешность  
# Производная 11 порядка  
def func\_der(x, a):  
 return abs(- a \* math.e\*\*(-x) + (1 - a) \* math.sin(x))  
  
# Находим максимум модуля функции  
x = np.arange(a, b + 0.001, 0.01)  
y = [func\_der(i, a) for i in x]  
M = max(y)  
plt.plot(x, y, color="c")  
plt.plot(b, M, "go", color="m")  
plt.axhline(func\_der(b, a), linestyle="--", alpha=0.5)  
plt.axvline(b, linestyle="--", alpha=0.5)  
plt.xticks([b] + list(np.arange(a, 1.0, 0.25)))  
plt.yticks([M] + list(np.arange(0.1, 0.80, 0.25)))  
plt.grid(axis="y")  
print("M:", M)**

**w = [1 for i in range(len(x\_find))]  
for i in range(len(x\_find)):  
 for j in range(n + 1):  
 w[i] \*= (x\_find[i] - nodes[j])  
print(w)**

**# Погрешности для точек  
delta\_r\_lagrange = []  
fact = math.factorial(n + 1)  
for i in range(len(w)):  
 delta\_r\_lagrange.append(abs(M \* w[i]) / fact)  
print("Оценки погрешностей для точек восстановления:", delta\_r\_lagrange)  
for i in range(len(w)):  
 print(f"r\_{i} <= {delta\_r\_lagrange[i]}")**

**# Метод Ньютона  
def get\_sep\_diff\_table(nodes, funcs):  
 n = len(nodes)  
 sep\_diff\_table = [  
 nodes,  
 funcs  
 ]  
 # Столбцы  
 for i in range(1, n):  
 column = []  
 # Строки  
 for j in range(n - i):  
 column.append((sep\_diff\_table[i][j + 1] - sep\_diff\_table[i][j]) / (sep\_diff\_table[0][j + i] - sep\_diff\_table[0][j]))  
 sep\_diff\_table.append(column)  
 return sep\_diff\_table  
  
def newton\_interp(nodes, funcs, find\_x):  
 n = len(nodes)  
 sep\_diff\_table = get\_sep\_diff\_table(nodes, funcs)  
 results = []  
 for x in find\_x:  
 find\_y = funcs[0]  
 prev = 1  
 for i in range(1, n):  
 prev \*= (x - nodes[i - 1])  
 find\_y += prev \* sep\_diff\_table[i + 1][0]  
 results.append(find\_y)  
 return results  
  
y\_find\_newton = newton\_interp(nodes, f\_nodes, x\_find)  
print("Восстановленные значения:", y\_find\_newton)**

**# Истинная погрешность  
r\_newton = [abs(y\_real[i] - y\_find\_newton[i]) for i in range(len(x\_find))]  
print("Истинная погрешность:", r\_newton)**

**# Теоретическая погрешность как у метода Лагранжа  
delta\_r\_newton = delta\_r\_lagrange  
for i in range(len(w)):  
 print(f"r\_{i} <= {delta\_r\_newton[i]}")**

**# Многочлен Чебышева  
# Пересчитываем узлы  
nodes\_cheb = []  
for i in range(n + 1):  
 nodes\_cheb.append((a + b) / 2 + math.cos((2 \* i + 1) \* math.pi / (2 \* (n + 1))) \* (b - a) / 2)  
print("Новые узлы:", nodes\_cheb)**

**# Значения в узлах  
f\_nodes\_cheb = []  
for i in range(n + 1):  
 f\_nodes\_cheb.append(func(nodes\_cheb[i], a))  
print("Новые значения в узлах:", f\_nodes\_cheb)**

**# Воспользуемся многочленом Ньютона по новой сетке узлов  
y\_find\_cheb = newton\_interp(nodes\_cheb, f\_nodes\_cheb, x\_find)  
print("Восстановленные значения:", y\_find\_cheb)**

**# Истинная погрешность  
r\_cheb = [abs(y\_real[i] - y\_find\_cheb[i]) for i in range(len(x\_find))]  
print("Истинная погрешность:", r\_cheb)**

**# Теоретическая погрешность общая  
delta\_r\_cheb\_gen = (M \* pow(b - a, n + 1)) / (math.factorial(n + 1) \* pow(2, 2 \* n + 1))  
print("Оценка для погрешностей:", delta\_r\_cheb\_gen)**

**# Теоретическая погрешность для каждой точки  
# Ищем w на чебышевской сетке  
w\_cheb = [1 for i in range(len(x\_find))]  
for i in range(len(x\_find)):  
 for j in range(n + 1):  
 w\_cheb[i] \*= (x\_find[i] - nodes\_cheb[j])  
print("w:", w\_cheb)  
# Погрешности  
delta\_r\_cheb = []  
fact = math.factorial(n + 1)  
for i in range(len(w)):  
 delta\_r\_cheb.append(abs(M \* w\_cheb[i]) / fact)  
print("Оценки погрешностей для точек восстановления:", delta\_r\_cheb)  
for i in range(len(w)):  
 print(f"r\_{i} <= {delta\_r\_cheb[i]}")**

**# Интерполяция в конце таблицы  
# Берём 4 узла  
m = 4  
nodes\_end = nodes[len(nodes) - m:]  
f\_nodes\_end = f\_nodes[len(f\_nodes) - m:]  
print("Узлы:", nodes\_end)  
print("Значения в них:", f\_nodes\_end)**

**# Таблица конечных разностей  
def get\_finite\_diff\_table(nodes, funcs):  
 n = len(nodes)  
 finite\_diff\_table = [  
 nodes,  
 funcs  
 ]  
 # Столбцы  
 for i in range(1, n):  
 column = []  
 # Строки  
 for j in range(n - i):  
 column.append((finite\_diff\_table[i][j + 1] - finite\_diff\_table[i][j]))  
 finite\_diff\_table.append(column)  
 return finite\_diff\_table  
  
finite\_table = get\_finite\_diff\_table(nodes\_end, f\_nodes\_end)  
print("Таблица конечных разностей\n", finite\_table)**

**# Замена на t  
t\_find = []  
for node in x\_find:  
 t\_find.append((node - nodes\_end[::-1][0]) / h)  
print(t\_find)**

**# Интерполирование  
def table\_end\_interp(nodes, funcs, find\_x):**

**n = len(nodes)**

**fin\_tab = get\_finite\_diff\_table(nodes, funcs)**

**results = []**

**for iter, x in enumerate(find\_x):**

**find\_y = funcs[n - 1]**

**prev = 1**

**fact = 1**

**for i in range(1, n):**

**prev \*= x + i - 1**

**find\_y += prev \* fin\_tab[i + 1][n - i - 1] / fact**

**fact \*= i + 1**

**results.append(find\_y)**

**return results**

**y\_find\_end = table\_end\_interp(nodes\_end, f\_nodes\_end, t\_find)  
print("В конце таблицы:", y\_find\_end)**

**# Истинная погрешность  
r\_end = [abs(y\_real[i] - y\_find\_end[i]) for i in range(len(x\_find))]  
print("Истинная погрешность:", r\_end)**

**# Теоретическая погрешность  
# Производная 4 порядка совпадет с самой функцией  
a, b = 1.25, 1.55**

**x = np.arange(a, b + 0.001, 0.01)**

**y = [func(i, 0.55) for i in x]**

**M = max(y)  
k = m - 1  
w\_end = []  
for t in t\_find:  
 w\_temp = t  
 for i in range(k):  
 w\_temp \*= t + i + 1  
 w\_end.append(w\_temp)  
print(w\_end)  
  
delta\_r\_end = []  
for i in range(len(t\_find)):  
 delta\_r\_end.append(pow(h, k + 1) \* w\_end[i] \* M / math.factorial(k + 1))  
  
print(delta\_r\_end)**

1. **Результат и его анализ.**
2. **Метод наименьших квадратов.**

**Результат.**

Получаем следующие коэффициенты многочлена:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1.00005981 | -0.55040221 | 0.05109348 | -0.09318502 | 0.04274368 | -0.00484003 |

Восстановленные значения:

[0.6639721063254683, 0.38719733942043694, 0.1450396192228639].

Истинная погрешность:

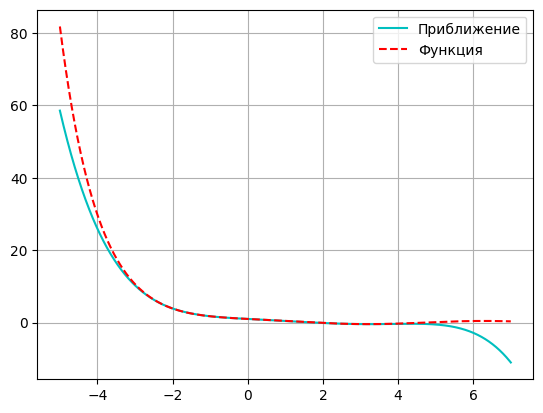
[3.329822140241134e-08, 1.1255016585387523e-08, 1.3148866689904892e-08].

Теоретическая погрешность: 4.5770075957360186e-08.

**Анализ.**

Истинные значения погрешности меньше её теоретической оценки, как и должно быть. Получили погрешности так как строили полином всего 5 степени, точность можно повысить, увеличив степень полинома.

Можно сравнить графики приближённой и реальной функции:



По графику видно, что по мере отдаления от отрезка [a, b] точность начинает заметно падать, в окрестности же [a, b] графики практически неотличимы.

1. **Многочлен Лагранжа.**

**Результат.**

Восстановленные значения:

[0.6639721396236813, 0.38719735067545374, 0.14503963237175216].

Истинная погрешность:

[8.43769498715119e-15, 2.220446049250313e-16, 2.1566082253343666e-14].

.

Теоретические погрешности в точках восстановления:

[1.6058213967572225e-14, 4.0034251063674126e-16, 3.4226332364342484e-14].

**Анализ.**

Истинные значения погрешности меньше её теоретической оценки, для первой точки на порядок ниже. Сравнение с МНК не является корректным, так в данном случае строится полином 10 степени.

1. **Многочлен Ньютона.**

**Результат.**

Восстановленные значения:

[0.6639721396236812, 0.38719735067545374, 0.14503963237175233].

Истинная погрешность:

[8.548717289613705e-15, 2.220446049250313e-16, 2.173261570703744e-14].

Теоретическая погрешность та же, что у многочлена Лагранжа.

**Анализ.**

Опять видно, что истинные значения погрешности ниже, чем теоретические оценки. Если сравнить с многочленом Лагранжа можно заметить, что получили погрешности того же порядка, что неудивительно, так как фактически мы строим один и тот же многочлен разными способами.

Также можно заметить, что в средней точке их значения полностью совпадают, на боковых точках многочлен Ньютона дал слегка худший результат, возможно это связанно с накоплением погрешностей при построении таблицы разделённых разностей.

1. **Многочлен Чебышева.**

**Результат.**

Новые узлы:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1.5449 | 1.5049 | 1.4279 | 1.3203 | 1.1909 | 1.050 | 0.9091 | 0.7797 | 0.6721 | 0.5952 | 0.5550 |

Восстановленные значения (методом Ньютона):

[0.6639721396236916, 0.38719735067545563, 0.14503963237173215].

Истинная погрешность:

[1.887379141862766e-15, 2.1094237467877974e-15, 1.5543122344752192e-15].

Теоретическая погрешность:

Общая оценка.

Оценка для каждой точки :

[3.418272532648346e-15, 3.5502616862258365e-15, 2.4811419281768608e-15].

**Анализ.**

Оценки в точках получились того же порядка, как и общая оценка. Сами значения этих оценок получились более точными.

Истинные значения погрешностей ниже, чем их оценки.

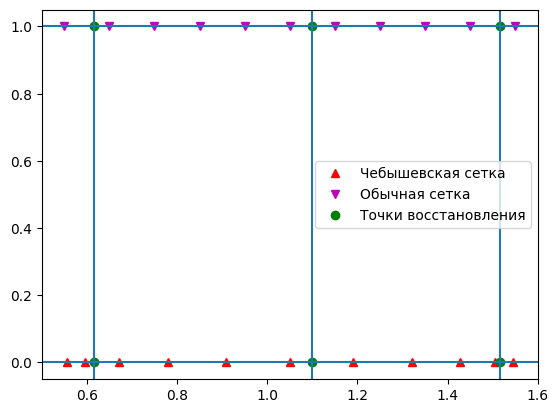
Если сравнивать с многочленом Ньютона можно заметить следующее:

* на границах точность повысилась

В левой точке порядок не изменился, в правой же точность лучше на порядок

* в средней точке точность понизилась на один порядок

Если схематически изобразить две стеки узлов, то получим следующее:



Можно заметить, что сетка перестала быть равномерной и средняя точка сдвинулась к одному из узлов, а количество узлов в её окрестности не увеличилось, это может быть причиной ухудшения точности в ней.

1. **Интерполяция в конце таблицы.**

**Результат.**

Рассматриваем только последнюю точку восстановления.

После замены на она станет равна: -0.3333.

Восстановленное значение:

[0.14504045000348767].

Истинная погрешность:

[8.176317570773861e-07]

Теоретическая погрешность:

[-2.884602595210065e-06].

**Анализ.**

Истинная погрешность ниже, чем теоретическая, как и должно быть. Получили точность порядка , так как строили полином всего 3 степени, так что сравнение с другими методами не совсем корректно. Точность можно повысить, если взять на том же отрезке более мелкое разбиение и строить по нему полином более высокой степени.