МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования

«Дальневосточный федеральный университет» (ДВФУ)

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

Департамент математического и компьютерного моделирования

ОТЧЁТ по лабораторной работе № 2

«Интерполирование функции с помощью интерполяционных формул с конечными разностями»

Вариант № 8

Выполнила: студент гр. Б9122-02.03.01сцт
Ф.И.О.
Ильяхова Алиса Алексеевна

Проверил: преподаватель
Ф.И.О.
Павленко Елизавета Робертовна

Владивосток

2024

Цель работы:

1. Построить таблицу конечных разностей по значениям табличной функции.

2. По соответствующим интерполяционным формулам вычислить значения функции в заданных узлах.

3. Оценить минимум и максимум для $f^{n+1}(x)$.

4. Проверить на выполнение равенство min R_n < $R_n(z)$ < max R_n, где z - заданный угол, а $R_n(z) = L_n(z) - f(z)$.

5. Сделать вывод по проделанной работе.

Основное:

1.1. Данные и их инициализация в коде:

$y = x^2 – \sin(x)$ на промежутке [0.5, 1.0]

$x^* = 0.77$; $x^{***} = 0.97$

$x^{**} = 0.52$; $x^{****} = 0.73$

```
x = Symbol('x', real=True)
y = x**2 - sin(x)
a = 0.5
b = 1.0
h = (b - a) / 10
n = 11

x_star2 = 0.52
x_star3 = 0.97
x_star4 = 0.73
```

1.2. Метод Ньютона:

```
def newton_parameter_minus(t: float, n: int):
    a = 1

    for i in range(n):
        a = a * (t - i)

    a = a / factorial(n)
    return a

def newton_parameter_plus(t: float, n: int):
```

```
    a = 1
    for i in range(n):
        a = a * (t + i)

    a = a / factorial(n)

    return a
```

Алгоритм таков: инициализируется переменная **a** равная **1**. Далее циклически умножается значение **a** на **(t + i)** или **(t – i)** для каждого значения **i**. Затем значение **a** делится на факториал **n**. Функция возвращает вычисленное значение параметра.

```
def insert_newton1(t: float, n: int, mass: list):
    Px = 0
    j = 0
    for i in range(n):
        Px += mass[i][j] * newton_parameter_minus(t, i)

    return Px

def insert_newton2(t: float, n: int, mass: list):
    Px2 = 0
    for i in range(0, n):
        j = n - i - 1
        Px2 += mass[i][j] * newton_parameter_plus(t, i)

    return Px2
```

Алгоритм таков: переменная **Px** или **Px2** инициализируется как 0. Производятся операции с использованием метода newton_parameterMinus или newton_patameterPlus для вычисления **Px** или **Px2**. Возвращается результат вычисления **Px** или **Px2**.

1.3. Метод Гаусса:

```
def gauss1_minus(t: float, n: int):
    a = 1

    for i in range(n):
        if i % 2 == 1 or i == 0:
            a = a * (t - i)
        else:
            a = a * (t + i - 1)

    a = a / factorial(n)
```

```
    return a

def gauss2_plus(t: float, n: int):
    a = 1

    for i in range(n):
        if i % 2 == 1 or i == 0:
            a = a * (t + i)
        else:
            a = a * (t - i + 1)

    a = a / factorial(n)
    return a
```

Алгоритм таков: инициализируется переменная **a** равная **1**. В цикле вычисляется значение параметра **a**, учитывая условия для умножения на **(t - i)** и **(t + i - 1)** или **(t + i)** и **(t - i + 1)** в зависимости от значения **i**. Функция возвращает вычисленное значение параметра **a**.

```
def insert_gauss1(t: float, n: int, mass: list):
    Px = 0
    j = 5
    for i in range(n):
        Px += mass[i][j] * gauss1_minus(t, i)
        if i % 2 != 0:
            j -= 1
    return Px

def insert_gauss2(t: float, n: int, mass: list):
    Px2 = 0
    j = 5
    for i in range(n):
        Px2 += mass[i][j] * gauss2_plus(t, i)
        if i % 2 == 0:
            j -= 1
    return Px2
```

Алгоритм таков: переменная **Px** или **Px2** инициализируется как 0. Производятся вычисления с использованием метода gauss1Minus или gauss2Plus для вычисления значения **Px** или **Px2**. Возвращается результат вычисления **Px** или **Px2**.

1.4. Таблица значений функции y(x)

№	x	y(x)
0	0.5	-0.229425538604203
1	0.55	-0.220187228930659
2	0.6	-0.204642473395035
3	0.65	-0.182686405736040
4	0.7	-0.154217687237691
5	0.75	-0.119138760023334
6	0.8	-0.0773560908995227
7	0.8500000000000001	-0.0287804051402927
8	0.9	0.0266730903725166
9	0.95	0.0890844952106262
10	1.0	0.158529015192103

```
x_list = []
y_list = []
for i in range(0, 11):
    xi = a + i * h
    x_list.append(xi)
    yi = y.subs(x, xi).evalf()
    y_list.append(yi)

table.add_column("№", [i for i in range(0, 11)])
table.add_column("x", x_list)
table.add_column("y(x)", y_list)
print(table)
```

1.5. Расчёт разностей

№	Value 1	Value 2	Value 3	Value 4
0	-0.229425538604203	-0.220187228930659	-0.204642473395035	-0.182686405736040
1	0.00923830967354383	0.0155447555356238	0.0219560676589959	0.0284687184983484
2	0.00630644586207996	0.00641131212337209	0.00651265083935254	0.00661020871600854
3	0.000104866261292136	0.000101338715980448	9.75578766559959e-5	9.35331934459604e-5
4	-3.52754531168786e-6	-3.78083932445206e-6	-4.02468321003546e-6	-4.25846748192127e-6
5	-2.53294012764194e-7	-2.43843885583406e-7	-2.33784271885806e-7	-2.23140321331528e-7
6	9.45012718078786e-9	1.00596136975994e-8	1.06439505542788e-8	1.12016847464425e-8
7	6.09486516811586e-10	5.84336856679357e-10	5.57734192163650e-10	5.29738142240888e-10
8	-2.51496601322287e-11	-2.66026645157069e-11	-2.79960499227627e-11	
9	-1.45300438347817e-12	-1.39338540705580e-12		
10	5.96189764223709e-14			

Value 5	Value 6	Value 7	Value 8	Value 9
-0.154217687237691	-0.119138760023334	-0.0773560908995227	-0.0287804051402927	0.0266730903725166
0.0350789272143570	0.0417826691238115	0.0485756857592300	0.0554534955128093	0.0624114048381096
0.00670374190945450	0.00679301663541854	0.00687780975357932	0.00695790932530027	0.00703311514336769
8.92747259640392e-5	8.47931181607864e-5	8.00995717209485e-5	7.52058180674142e-5	
-4.48160780325280e-6	-4.69354643983788e-6	-4.89375365353428e-6		
-2.11938636585085e-7	-2.00207213696402e-7			
1.17314228886833e-8				

Value 10	Value 11
0.0890844952106262	0.158529015192103
0.0694445199814773	

```
list_diffs = [y_list.copy()]

while len(list_diffs[-1]) != 1:
    lis = []
    for i in range(0, len(list_diffs[-1]) - 1):
        lis.append(list_diffs[-1][i + 1] - list_diffs[-1][i])
    list_diffs.append(lis)

list_to_table = list_diffs.copy()
max_length = len(max(list_to_table, key=len))

for lst in list_to_table:
    while len(lst) < max_length:
        lst.append("")

table.field_names = ["№", "Value 1", "Value 2", "Value 3", "Value 4", "Value
5", "Value 6", "Value 7", "Value 8",
                     "Value 9", "Value 10", "Value 11"]
for i in range(0, len(list_to_table)):
    table.add_row([f"{i}"] + list_to_table[i])

table.set_style(PLAIN_COLUMNS)
print(table)
```

1.6. Экстремумы

```
Ньютон 1: -0.226480137843736
R_N1:  3.88578058618805e-16
Ньютон 2: 0.116014286661550
R_N2:  -2.77555756156289e-17
Гаусс 2: -0.133969691494705
R_G2:  -5.64910071221281e-8
```

```
Минимум f(12)(E) на отрезке: -0.841470984807897
Максимум f(12)(E) на отрезке: -0.479425538604203
Минимум Rn на отрезке: -5.14229701640716e-34
Максимум Rn на отрезке: 0
```

```
w = 1
for i in range(11):
    w = w * (x - x_list[i])

y_der = diff(y, x, n + 1)
R_n = y_der * w / factorial(n + 1)

crit_points = solve(y_der, x)
crit_points = [point for point in crit_points if a <= float(point) <= b]
endpoints = [a, b]
values_at_endpoints = {endpoint: y_der.subs(x, endpoint).evalf() for endpoint
in endpoints}
values_at_critical_points = {cp: y_der.subs(x, cp).evalf() for cp in
crit_points}
extremum_values = list(values_at_endpoints.values()) +
list(values_at_critical_points.values())
minimum = min(extremum_values)
maximum = max(extremum_values)
print('Минимум f(12)(E) на отрезке:', minimum)
print('Максимум f(12)(E) на отрезке:', maximum)

crit_points = solve(R_n, x)
crit_points = [point for point in crit_points if a <= float(point) <= b]
endpoints = [a, b]
values_at_endpoints = {endpoint: R_n.subs(x, endpoint).evalf() for endpoint in
endpoints}
values_at_critical_points = {cp: R_n.subs(x, cp).evalf() for cp in crit_points}
extremum_values = list(values_at_endpoints.values()) +
list(values_at_critical_points.values())
minimum = min(extremum_values)
maximum = max(extremum_values)
print('Минимум Rn на отрезке:', minimum)
print('Максимум Rn на отрезке:', maximum)
```

Используемые библиотеки:

В ходе работы мне потребовалось использовать следующие библиотеки: PrettyTable, sympy

Библиотека **PrettyTable** предоставляет инструменты для создания красиво оформленных таблиц в Python. Она позволяет отображать данные в удобочитаемом виде, что упрощает их анализ и визуализацию. Библиотека **sympy** представляет собой мощный символьный математический пакет для Python. Она способна обрабатывать символьные выражения, уравнения, и действия, что делает ее полезным инструментом в области научных вычислений, анализа данных и математического моделирования.

Вывод:

В ходе выполнения лабораторной работы были реализованы различные численные методы(методы Ньютона и методы Гаусса) для аппроксимации функции. Данными методами были найдены коэффициенты аппроксимирующих многочленов. Были применены методы для вычисления значений аппроксимирующих многочленов в заданных точках. Были найдены минимальное и максимальное значение R_n на заданном интервале. Произведено сравнение результатов и проверка неравенства $\min R_n < R_n(z) < \max R_n$.

Полный код:

```
from prettytable import PrettyTable, PLAIN_COLUMNS
from sympy import *

def newton_parameter_minus(t: float, n: int):
    a = 1

    for i in range(n):
        a = a * (t - i)

    a = a / factorial(n)
    return a

def newton_parameter_plus(t: float, n: int):
    a = 1
    for i in range(n):
        a = a * (t + i)

    a = a / factorial(n)

    return a

def gauss1_minus(t: float, n: int):
```

```
    a = 1

    for i in range(n):
        if i % 2 == 1 or i == 0:
            a = a * (t - i)
        else:
            a = a * (t + i - 1)

    a = a / factorial(n)
    return a

def gauss2_plus(t: float, n: int):
    a = 1

    for i in range(n):
        if i % 2 == 1 or i == 0:
            a = a * (t + i)
        else:
            a = a * (t - i + 1)

    a = a / factorial(n)
    return a

def insert_gauss1(t: float, n: int, mass: list):
    Px = 0
    j = 5
    for i in range(n):
        Px += mass[i][j] * gauss1_minus(t, i)
        if i % 2 != 0:
            j -= 1
    return Px


def insert_gauss2(t: float, n: int, mass: list):
    Px2 = 0
    j = 5
    for i in range(n):
        Px2 += mass[i][j] * gauss2_plus(t, i)
        if i % 2 == 0:
            j -= 1
    return Px2
```

```
def insert_newton1(t: float, n: int, mass: list):
    Px = 0
    j = 0
    for i in range(n):
        Px += mass[i][j] * newton_parameter_minus(t, i)

    return Px


def insert_newton2(t: float, n: int, mass: list):
    Px2 = 0
    for i in range(0, n):
        j = n - i - 1
        Px2 += mass[i][j] * newton_parameter_plus(t, i)

    return Px2


table = PrettyTable()

x = Symbol('x', real=True)
y = x**2 - sin(x)
a = 0.5
b = 1.0
h = (b - a) / 10
n = 11

x_star2 = 0.52
x_star3 = 0.97
x_star4 = 0.73

x_list = []
y_list = []
for i in range(0, 11):
    xi = a + i * h
    x_list.append(xi)
    yi = y.subs(x, xi).evalf()
    y_list.append(yi)
```

```
table.add_column("№", [i for i in range(0, 11)])
table.add_column("x", x_list)
table.add_column("y(x)", y_list)
print(table)

table.clear()

list_diffs = [y_list.copy()]

while len(list_diffs[-1]) != 1:
    lis = []
    for i in range(0, len(list_diffs[-1]) - 1):
        lis.append(list_diffs[-1][i + 1] - list_diffs[-1][i])
    list_diffs.append(lis)

list_to_table = list_diffs.copy()
max_length = len(max(list_to_table, key=len))

for lst in list_to_table:
    while len(lst) < max_length:
        lst.append("")

table.field_names = ["№", "Value 1", "Value 2", "Value 3", "Value
4", "Value 5", "Value 6", "Value 7", "Value 8",
                     "Value 9", "Value 10", "Value 11"]
for i in range(0, len(list_to_table)):
    table.add_row([f"{i}"] + list_to_table[i])

table.set_style(PLAIN_COLUMNS)
print(table)

t = min(abs(x_list[0] - x_star2), abs(x_list[1] - x_star2)) / h

print('Ньютон 1:', insert_newton1(t, 11, list_diffs))
print("R_N1: ", insert_newton1(t, 11, list_diffs) - y.subs(x,
x_star2).evalf())

t = -1 * (x_list[-1] - x_star3) / h

print('Ньютон 2:', insert_newton2(t, 11, list_diffs))
print("R_N2: ", insert_newton2(t, 11, list_diffs) - y.subs(x,
```

```
x_star3).evalf())

i = 0
for i in range(n - 1):
    if (x_list[i] < x_star4) and (x_list[i + 1] > x_star4):
        break

t1 = abs(x_list[i] - x_star4) / h
t2 = abs(x_list[i + 1] - x_star4) / h

if t1 < t2:
    print('Гаусс 1:', insert_gauss1(t1, 11, list_diffs))
    print("R_G1: ", insert_gauss1(t1, 11, list_diffs) - y.subs(x,
x_star4).evalf())
else:
    t2 = -1 * t2
    print('Гаусс 2:', insert_gauss2(t2, 11, list_diffs))
    print("R_G2: ", insert_gauss2(t2, 11, list_diffs) - y.subs(x,
x_star4).evalf())

w = 1
for i in range(11):
    w = w * (x - x_list[i])

y_der = diff(y, x, n + 1)
R_n = y_der * w / factorial(n + 1)

crit_points = solve(y_der, x)
crit_points = [point for point in crit_points if a <= float(point)
<= b]
endpoints = [a, b]
values_at_endpoints = {endpoint: y_der.subs(x, endpoint).evalf()
for endpoint in endpoints}
values_at_critical_points = {cp: y_der.subs(x, cp).evalf() for cp
in crit_points}
extremum_values = list(values_at_endpoints.values()) +
list(values_at_critical_points.values())
minimum = min(extremum_values)
maximum = max(extremum_values)
print('Минимум f(12)(E) на отрезке:', minimum)
print('Максимум f(12)(E) на отрезке:', maximum)
```

```
crit_points = solve(R_n, x)
crit_points = [point for point in crit_points if a <= float(point)
<= b]
endpoints = [a, b]
values_at_endpoints = {endpoint: R_n.subs(x, endpoint).evalf() for
endpoint in endpoints}
values_at_critical_points = {cp: R_n.subs(x, cp).evalf() for cp in
crit_points}
extremum_values = list(values_at_endpoints.values()) +
list(values_at_critical_points.values())
minimum = min(extremum_values)
maximum = max(extremum_values)
print('Минимум Rn на отрезке:', minimum)
print('Максимум Rn на отрезке:', maximum)
```