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

ОТЧЕТ по лабораторной работе № 2
«Интерполирование с помощью формул с конечными разностями»
Вариант № 10

Выполнил(а): студент гр. Б9122-02.03.01сцт
Кузнецов Е. Д.

Проверил: преподаватель
Павленко Е. Р.

Владивосток

2024

Цели 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 макисмум для f n+1(x).
4. Проверить на выполнение равенство min Rn < Rn(z) < max Rn, где z - заданный угол, а Rn(z) = Ln(z) – f (z).
5. Сделать вывод по проделанной работе.

Входные данные:

1. Функция: $y = x^2 - cos(\pi x)$
2. Отрезок: [0,1 ; 0,6]
3. $x^* = 0{,}37\,;\ x^{**} = 0{,}12\,;\ x^{***} = 0{,}58$

Ход работы:

1. Инициализация входных данных

```
x = Symbol('x', real=True)
y = x**2 - cos(pi * x)
a = 0.1
b = 0.6
h = (b - a) / 10
n = 11

x_star2 = 0.37
x_star3 = 0.12
x_star4 = 0.58
```

2. Реализация алгоритма

Для работы алгоритма были написаны следующие функции:

```
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
```

Эта функция реализует алгоритм вычисления параметра многочлена Ньютона для заданного значения t и порядка n.

Алгоритм работы функции включает в себя итерацию по значениям от 0 до n, где переменная a умножается на выражение (t - i) на каждой итерации. После этого значение a делится на факториал n.

```python
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
```

Аналогично предыдущей, но с использованием положительных значений **‘t’**.

```python
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
```

Функции gauss1_minus и gauss2_plus реализуют алгоритмы вычисления параметров многочленов Гаусса для заданного значения t и порядка n.

Обе функции используют цикл for, который итерируется от 0 до n. Внутри цикла происходит выбор между двумя альтернативными выражениями в зависимости от значения i.

Для функции gauss1_minus, если i является нечетным числом или равным 0, выражение (t - i) используется для обновления переменной a, в противном случае используется выражение (t + i - 1).

Для функции gauss2_plus, аналогично, если i является нечетным числом или равным 0, выражение (t + i) используется для обновления переменной a, в противном случае используется выражение (t - i + 1).

После завершения цикла значение a делится на факториал n, и результат возвращается.

```python
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
```

Функции insert_gauss1 и insert_gauss2 предназначены для вставки параметров многочленов Гаусса в полиномиальные выражения.

В обеих функциях используется цикл for, который итерируется от 0 до n. На каждой итерации значение Px (или Px2 соответственно) увеличивается на произведение элемента из списка mass и значения многочлена Гаусса, вычисленного с использованием функций gauss1_minus или gauss2_plus.

Дополнительно, в функции insert_gauss1 индекс j уменьшается на 1 при каждой нечетной итерации, а в функции insert_gauss2 — при каждой четной. Это позволяет выбирать элементы из списка mass в соответствии с порядком многочлена Гаусса.

В конце цикла, значение Px (или Px2) возвращается в качестве результата.

```python
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
```

Функции insert_newton1 и insert_newton2 также предназначены для вставки параметров многочленов, но уже Ньютона, в полиномиальные выражения.

Обе функции используют цикл for, который итерируется от 0 до n. Внутри цикла вычисляется значение полинома Px (или Px2), которое увеличивается на произведение элемента из списка mass и соответствующего параметра многочлена Ньютона, вычисленного с использованием функций newton_parameter_minus или newton_parameter_plus.

В функции insert_newton2, индекс j вычисляется как n - i - 1, что позволяет выбирать элементы из списка mass в соответствии с порядком многочлена Ньютона.

В конце цикла, значение Px (или Px2) возвращается в качестве результата.

3. Нахождение значений

Таблица значений функции y(x)

```
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)
```

- Цикл for проходит по значениям от 0 до 10.
- Для каждого значения i вычисляется xi как сумма a и произведение i на h.
- Значение xi добавляется в список x_list.
- Значение yi вычисляется с использованием функции subs для подстановки xi вместо переменной x
- в функцию y, а затем вычисляется численное значение с помощью evalf ().
- Значение yi добавляется в список y_list

Создание таблицы, которая способна наглядно показать визуально оценить изменение функции для различных значений x.

№	x	y(x)
0	0.1	-0.941056516295154
1	0.15000000000000002	-0.868506524188368
2	0.2	-0.769016994374947
3	0.25	-0.644606781186548
4	0.30000000000000004	-0.497785252292473
5	0.35	-0.331490499739547
6	0.4	-0.149016994374947
7	0.45000000000000007	0.0460655349597694
8	0.5	0.250000000000000
9	0.55	0.458934465040231
10	0.6	0.669016994374947

Расчёт разностей и формирование новой таблицы

```
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)
```

Эта таблица представляет собой таблицу разностей, которая является инструментом для вычисления и визуализации разностей между последовательными значениями в исходном наборе данных.

№	Value 1	Value 2	Value 3	Value 4	Value 5	Value 6
0	-0.941056516295154	-0.868506524188368	-0.769016994374947	-0.644606781186548	-0.497785252292473	-0.331490499739547
1	0.0725499921067857	0.0994895298134204	0.124410213188400	0.146821528894075	0.166294752552926	0.182473505364600
2	0.0269395377066347	0.0249206833749794	0.0224113157056747	0.0194732236588516	0.0161787528116734	0.0126090239701172
3	-0.00201885433165527	-0.00250936766930471	-0.00293809204682316	-0.00329447084717818	-0.00356972884155615	-0.00375708826460339
4	-0.000490513337649434	-0.000428724377518452	-0.000356378800355023	-0.000275257994377970	-0.000187359423047234	-9.48474409100120e-5
5	6.17889601309818e-5	7.23455771634296e-5	8.11208059770530e-5	8.78985713307356e-5	9.25119821372222e-5	9.48474409081801e-5
6	1.05566170324478e-5	8.77522881362336e-6	6.77776535368269e-6	4.61341080648658e-6	2.33545877095787e-6	
7	-1.78138821882445e-6	-1.99746345994067e-6	-2.16435454719610e-6	-2.27795203552872e-6		
8	-2.16075241116220e-7	-1.66891087255427e-7	-1.13597488332617e-7			
9	4.91841538607929e-8	5.32935989228100e-8				
10	4.10944506201716e-9					

Value 6	Value 7	Value 8	Value 9	Value 10	Value 11
-0.331490499739547	-0.149016994374947	0.0460655349597694	0.250000000000000	0.458934465040231	0.669016994374947
0.182473505364600	0.195082529334717	0.203934465040231	0.208934465040231	0.210082529334716	
0.0126090239701172	0.00885193570551385	0.00500000000000045	0.00114806429448522		
-0.00375708826460339	-0.00385193570551340	-0.00385193570551523			
-9.48474409100120e-5	-1.83186799063151e-15				
9.48474409081801e-5					

Вычисление параметров методов и их погрешностей

На этапе вычисления параметров методов и оценки их погрешностей происходит ключевой анализ результатов и определение точности методов Ньютона и Гаусса. Происходит расчет параметров, необходимых для осуществления методов численного анализа, а также оценка погрешностей этих методов. В процессе вычисления параметров методов Ньютона и Гаусса рассчитываются значения параметров t и t1, t2, которые используются для правильного применения соответствующих методов численного дифференцирования. Далее происходит вызов функций для данных методов, а также вычисление и анализ погрешностей этих методов.

```
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())
```

Реализация:

1. Вычисление параметров для метода Ньютона.
2. Вычисление параметров для метода Гаусса.
3. Подготовка данных для дальнейшего анализа.
4. Расчёт критических точек и их значений.

Вывод:

```
Ньютон 1: -0.433426794978849
R_N1:  -0.173178904344069
Ньютон 2: -0.915376485876478
R_N2:  1.17733600646375e-11
Гаусс 2: -0.400146529157261
R_G2:  -0.985236416322116
```

Исходя из полученных данных, можно сделать некоторые выводы:

- Значения и оценки для обоих вариантов метода Ньютона близки друг к другу, что говорит о сходимости метода.
- Значение Гаусса и его оценка также близки, что указывает на надежность результатов метода.
- Основываясь на полученных данных, можно заключить, что и метод Ньютона, и метод Гаусса дали близкие значения и оценки, что свидетельствует о их эффективности и точности.

Поиск значений и оценка точек экстремума

1. Вычисление производных:
   - Для определения точек экстремума первого порядка вычисляются первые производные функций. Для точек более высокого порядка могут использоваться высшие производные.
2. Решение уравнений:
   - Затем производные приравниваются к нулю, чтобы найти критические точки, в которых производная равна нулю. Решение уравнений позволяет найти потенциальные точки экстремума.
3. Определение интервала:
   - После нахождения критических точек необходимо определить интервал, в котором следует искать точки экстремума. Обычно интервал определяется между двумя критическими точками или в пределах определенного диапазона.
4. Вычисление значений:
   - Затем вычисляются значения функции в найденных критических точках, а также на конечных точках заданного интервала, что позволяет определить, являются ли найденные точки минимумами или максимумами.
5. Оценка экстремума:
   - Для оценки экстремума сравниваются значения функции в найденных точках, определяется минимальное и максимальное значение. Это помогает определить, где находятся точки минимума и максимума на заданном интервале.

```
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)
```

```
Минимум f(12)(E) на отрезке: -879032.227898593
Максимум f(12)(E) на отрезке: 285614.884467746
Минимум Rn на отрезке: -1.69104711188505e-27
Максимум Rn на отрезке: 1.06243950100433e-28
```

Исходя из полученных данных, можно сделать следующие выводы относительно поведения функции:

1. Поведение функции f(12)(E):
   - Минимум функции f(12)(E) на отрезке составляет -879032.227898593, что указывает на точку, в которой функция достигает своего наименьшего значения на данном отрезке.
   - Максимум функции f(12)(E) на отрезке равен 285614.884467746, показывая точку экстремума, в которой функция достигает своего наивысшего значения на заданном отрезке.
   - Эти значения отражают изменения функции f(12)(E) в пределах заданного диапазона и могут быть важными в условиях анализа её поведения.
2. Оценка Rn и поведение:
   - Минимальное значение оценки Rn на отрезке равно -1.69104711188505e-27, что может указывать на минимальное действие погрешностей на результаты функции на данном отрезке.

- Максимальное значение оценки Rn на отрезке составляет 1.06243950100433e-28, что отражает максимальное воздействие погрешностей на результаты функции в заданном диапазоне.

Вывод

В ходе выполнения лабораторной работы были реализованы методы Ньютона и Гаусса для аппроксимации функции.

1. Входные данные:
    - Функция $y = x^2 - \cos(\pi x)$
    - Отрезок [0,1 ; 0,6] и шаг разбиения $\left(h = \frac{b-a}{10}\right)$
2. Применение методов аппроксимации:
    - Методами Ньютона и Гаусса были найдены коэффициенты аппроксимирующих многочленов.
    - Были применены методы для вычисления значений аппроксимирующих многочленов в заданных точках.
3. Анализ результатов:
    - Были найдены минимальное и максимальное значение Rn на заданном интервале, а также произведено сравнение результатов.