Отчет по лабораторной работе № 2
Вариант № 3

Винницкая Дина Сергеевна

Группа: Б9122-02-03-01сцт

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 макисмум 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.

Входные данные:

1. **Функция:** $y = x^2 + \ln(x) - 4$
2. **Отрезок** $[1.5, 2.0]$
3. $x^* = 1,52, \quad x^{**} = 1.52, \quad x^{***} = 1,97$

Ход работы:

1 Используемые библиотеки

Для реализации необходимой программы используются следующие библиотеки языка Python:

- **PrettyTable** - библиотека **Python**PrettyTable предоставляет инструменты для создания красиво оформленных таблиц в **Python**. Она позволяет отображать данные в удобочитаемом виде, что упрощает их анализ и визуализацию.
  - **Функциональность:** библиотека **PrettyTable** обеспечивает возможность создания таблиц с различными стилями форматирования, включая плоские колонки, как в **PLAIN _COLUMNS**, что обеспечивает гибкость в представлении данных.
- **sympy** - 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.
  - **Функциональность:** библиотека **sympy** позволяет выполнять различные операции с символами, такие как дифференцирование, интегрирование, решение уравнений и многое другое, что делает ее важным ресурсом для работы с математическими вычислениями в **Python**.

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

2 Инициализация входных данных

Начиная реализацию алгоритма известна непосредственная функция$y = x^2 + \ln(x) - 4$ и отрезок $[1.5, 2.0]$

```
x = Symbol('x', real=True)
y = x**2 - log(x) - 4
a = 1.5
b = 2.0
h = (b - a) / 10
n = 11
x_star2 = 1.52
x_star3 = 1.52
x_star4 = 1.97
```

- В приведенном ниже коде определяется сама функция, границы отрезка, количество узлов для разбиения, x^*, x^{**}, x^{***}.

3 Реализация непосредтвенного алгоритма

Для реализации алгоритма были написаны следующие функции, позволяющие выполнить необходимый пласт работь, удовлетворить условию лабораторной работы и найти искомые значения.

Функция newton_parameter_minus

```
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
```

- Функция **newton_parameter_minus(t: float, n: int)** Эта функция вычисляет параметр для метода Ньютона, используя отрицательные значения параметров.
  Алгоритм
  - Инициализируется переменная **a** равная 1.
  - Далее циклически умножается значение a на **(t - i)** для каждого значения **i** от 0 до n.
  - Затем значение **a** делится на факториал **n**.
  - Возвращается вычисленное значение параметра **a**

Функция newton_parameter_plus

```
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
```

- Функция **newton_parameter_plus(t: float, n: int)** Данная функция вычисляет параметр для метода Ньютона с использованием положительных параметров.
  Алгоритм
  - Инициализируется переменная **a** равная 1.
  - Далее циклически умножается значение a на **(t + i)** для каждого значения **i** от 0 до n.
  - Затем значение **a** делится на факториал **n**.
  - Функция возвращает вычисленное значение параметра **a**

Функция gauss1_minus

```
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
```

- Функция **gauss1_minus(t: float, n: int)** Эта функция рассчитывает параметр для метода Гаусса с отрицательными параметрами.
  Алгоритм
  - Инициализируется переменная **a** равная 1.
  - В цикле вычисляется значение параметра **a**, учитывая условия для умножения на **(t - i) и (t + i - 1)** в зависимости от значения **i**.
  - Функция возвращает вычисленное значение параметра **a**

Функция gauss2_plus

```
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
```

- Функция **gauss2_plus(t: float, n: int)** Эта функция вычисляет параметр для метода Гаусса с положительными параметрами.
  Алгоритм
  - Инициализируется переменная **a** равная 1.
  - В цикле вычисляется значение параметра **a**, учитывая условия для умножения на **(t + i) и (t - i + 1)** в зависимости от значения **i**.
  - Функция возвращает вычисленное значение параметра **a**

Функция insert_gauss1

```
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
```

- Функция **insert_gauss1(t: float, n: int, mass: list)** Данная функция вставляет метод Гаусса с отрицательными параметрами.
  Алгоритм
  - Переменная **Px** инициализируется как 0.
  - Производятся вычисления с использованием метода **gauss1Minus** для вычисления значения **Px**.

– Возвращается результат вычисления **Px**.

Функция insert_gauss2

```python
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
```

- Функция **insert_gauss2(t: float, n: int, mass: list)** Эта функция вставляет метод Гаусса с положительными параметрами.
  Алгоритм
  - Переменная **Px2** инициализируется как 0.
  - Производятся вычисления с использованием метода **gauss2Plus** для вычисления значения **Px2**.
  - Возвращается результат вычисления **Px2**.

Функция insert_newton1

```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
```

- Функция **insert_newton1(t: float, n: int, mass: list)** Данная функция вставляет метод Ньютона с отрицательными параметрами.
  Алгоритм
  - Переменная **Px** инициализируется как 0.
  - Производятся операции с использованием метода **newton_parameterMinus** для вычисления **Px**.
  - Возвращается результат вычисления **Px**.

Функция insert_newton2

```python
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
```

- Функция **insert_newton2(t: float, n: int, mass: list)** Эта функция вставляет метод Ньютона с положительными параметрами.
  Алгоритм
  - Переменная **Px2** инициализируется как 0.
  - Производятся операции с использованием метода **newton_parameterPlus** для вычисления **Px2**.
  - Возвращается результат вычисления **Px2**.

4 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**. Это обеспечивает наглядное представление данных и упрощает анализ поведения функции на определенном диапазоне.

№	x	y(x)
0	1.5	-2.15546510810816
1	1.55	-2.03575493093116
2	1.6	-1.91000362924574
3	1.65	-1.77827528791249
4	1.7	-1.64062825106217
5	1.75	-1.49711578793542
6	1.8	-1.34778666490212
7	1.85	-1.19268563909023
8	1.9	-1.03185388617239
9	1.95	-0.865329372575656
10	2.0	-0.693147180559945

Рис. 1: Таблица значений функции y(x)

Расчет разностей и формирование новой таблицы

```python
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
0	-2.15546510810816	-2.03575493093116	-1.91000362924574 - 1.77827528791249	-1.64062825106217
1	0.119710177177009	0.125751301685420	0.131728341333246	0.137647036850319
2	0.00604112450841043	0.00597703964782581	0.00591869551707314	0.00586542627642905
3	-6.40848605846234e-5	-5.83441307526744e-5	-5.32692406440827e-5	-4.87663631655e-5
4	5.74072983194895e-6	5.07489010859175e-6	4.50287077091716e-6	4.00924189913887e-6

№	Value 5	Value 6	Value 7	Value 8
0	-1.49711578793542	-1.34778666490212	-1.19268563909023	-1.03185388617239
1	0.143512463126748	0.149329123033304	0.155101025811886	0.160831752917838
2	0.00581665990655589	0.00577190277858186	0.00573072710595257	0.00569276067890101
3	-4.47571279740266e-5	-4.11756726292900e-5	-3.79664270515612e-5	-3.50822599298750e-5
4	3.58145534473664e-6	3.20924557772884e-6	2.88416712168615e-6	

№	Value 9	Value 10	Value 11
0	-0.865329372575656		
1	0.166524513596739	0.172182192015710	
2	0.00565767841897113		
3			
4			

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('N 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('N 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('G 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('G 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:

- Вычисляется значение **t** для метода Ньютона 1, которое представляет собой минимальное из двух значений: разницы между нулевым элементом списка **x_list** и **x_star2**, и разницы между первым элементом списка **x_list** и **x_star2**, деленное на **h**.

- Далее при помощи функций `insert_newton1` и `insert_newton2` происходит вычисление метода и его оценка.

2. Вычисление параметров для метода Гаусса:

- Вычисляются значения **t1** и **t2** для метода Гаусса, представляющие собой отношение модуля разности между определенными значениями из **x_list** и **x_star4** к **h**.
- В зависимости от соотношения **t1** и **t2**, вызываются функции `insert_gauss1` или `insert_gauss2` для расчета метода Гаусса и оценки его погрешности.

3. Подготовка данных для дальнейшего анализа:

- Выполняется вычисление произведения $(x - \mathbf{x\_list}[\mathbf{i}])$ для всех элементов **x_list**.

4. Расчет критических точек и их значений:

- Производная **y_der** функции **y** вычисляется по **x** до $(n + 1)$ порядка.
- Проводится поиск критических точек на отрезке между **a** и **b**.
- Значения производной на конечных точках и на критических точках вычисляются и сохраняются в соответствующих словарях.

Вывод

```
Ньютон 1: -2.10831033485776
R_N1:  4.25437463036360e-13
Ньютон 2: -2.10831033485776
R_N2:  4.25881552246210e-13
Гаусс 1: -1.43807949724126
R_G1:  -0.640945954491365
```

Рис. 2: Значение точек экстремума

- Исходя из полученных данных, можно сделать следующие 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

Этапы поиска значений точек экстремума является важным этапом в анализе функций и моделей.

1. Производные функции:

- Для поиска точек экстремума сначала вычисляются производные функций. Это может быть первая производная для определения точек экстремума первого порядка или высшие производные для точек экстремума более высокого порядка.

2. Решение уравнений:

- Затем производные приравниваются к нулю, чтобы найти критические точки, где производная равна нулю. Решение уравнения производной равной нулю позволяет найти потенциальные точки экстремума.

3. Определение интервала:

- После нахождения критических точек необходимо оценить интервал, в котором следует искать точки экстремума, обычно 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 позволяет 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]

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('Min f(12)(E):', minimum)
print('Max f(12)(E):', maximum)

crit_points = solve(R_n, x)
crit_points = [point for point in crit_points if a <= float(point) <= 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('Min Rn:', minimum)
print('Max Rn:', maximum)
```

```
Минимум f(12)(E) на отрезке: 9745.31250000000
Максимум f(12)(E) на отрезке: 307652.613930803
Минимум Rn на отрезке: 0
Максимум Rn на отрезке: 90.7181361218318
```

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 9745.31250000000, что указывает на точку, в

которой функция достигает своего наименьшего значения на данном отрезке.
- Максимум функции f(12)(E) на отрезке равен 307652.613930803, 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 при анализе ее поведения.

2. **Оценка Rn и поведение**:
- Минимальное значение оценки Rn на отрезке равно 0, что может указывать на минимальное воздействие погрешностей на результаты функции на данном отрезке.
- Максимальное значение оценки Rn на отрезке составляет 90.7181361218318, что отражает максимальное воздействие погрешностей на результаты функции в заданном диапазоне.

Проверка равенства

$$R_{11}(x) = \frac{f^{(11+1)}}{(11+1)!} \cdot \prod_{i=1}^{11} \left(x - L[i]\right);$$

$$R_{max}(x) = -\frac{1}{\ln(12)} \cdot \frac{11!}{8{,}04^{12}} \cdot \frac{1}{(11+1)!} = \prod_{i=1}^{11} \left(x - L[i]\right) = -\frac{1}{12\ln(12)} \cdot \frac{1}{8{,}04^{12}} \cdot \prod_{i=1}^{11} \left(x - L[i]\right) = -4{,}59659 \prod_{i=1}^{11} \left(x - L[i]\right)$$

$$R_{min}(x) = -\frac{1}{\ln(12)} \cdot \frac{11!}{1{,}5^{12}} \ \frac{1}{(11+1)!} = \prod_{i=1}^{11} \left(x - L[i]\right) = -\frac{1}{12\ln(12)} \cdot \frac{1}{1{,}5^{12}} \cdot \prod_{i=1}^{11} \left(x - L[i]\right) = -0{,}000258472 \prod_{i=1}^{11} \left(x - L[i]\right)$$

Максимальное значение R_n равно -4.59659, а минимальное значение R_n равно -0.000258472. Исходя из этого, можно сделать вывод, что неравенство $\min R_n < R_n(z) < \max R_n$ выполняется для всех значений $R_n(z)$ в интервале от минимального до максимального значения.

Это означает, что условие $\min R_n < R_n(z) < \max R_n$ соблюдается для всех $R_n(z)$ и подтверждает то, что значения $R_n(z)$ находятся в соответствующем интервале между минимальным и максимальным значениями R_n.

Вывод

В ходе выполнения лабораторной работы были реализованы различные численные методы (методы Ньютона и методы Гаусса) для аппроксимации функции.

1. **Входные данные:**
   - функция $y = x^2 - \log(x) - 4$.
   - Отрезок $[a = 1.5,\ b = 2.0]$ и шаг разбиения $(h = \frac{b-a}{10})$.
2. **Применение методов аппроксимации:**
   - Методами Ньютона и Гаусса были найдены коэффициенты аппроксимирующих многочленов.
   - Были примены методы для вычисления значений аппроксимирующих многочленов в заданных точках.
3. **Анали результатов:**
   - Были найдены минимальное и максимальное значение R_n на заданном интервале.
   - Произведено сравнение результатов и проверка неравенства $\min R_n < R_n(z) < \max R_n$.