ФГАОУ ВО «МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Лабораторная работа №3

Организация циклов

Вариант № 28

По дисциплине:

Основы программирования

Выполнил студент 1-го курса группы 243-323

Онищенко А. А.

Проверил

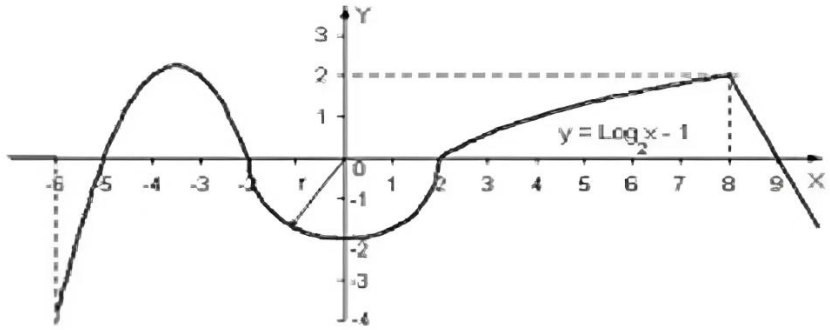
_____Никишина И. Н.

Москва, 2025

Задание 1

Постановка задачи

Вычислить и вывести на экран в виде таблицы значения функции, заданной графически, на интервале от *Хнач* до *Хкон* с шагом *dx*. Интервал и шаг задать таким образом, чтобы проверить все ветви программы. Таблицу снабдить заголовком и шапкой.



Теоретическая часть

Для решения задачи используется обновленная программа из первого задания второй лабораторной работы. В нее добавляется оператор цикла, проверяющий условие при котором программа продолжит выполняться по кругу.

Описание программы

Программа написана на алгоритмическом языке Python 3.6, реализована в среде ОС Windows 10 и состоит из частей, отвечающих за ввод данных, вычисление и представление данных на экране монитора.

Описание алгоритма

1. Ввести значения переменных *Xbeg, Xend, Dx.*
2. Присвоить текущему значению *Xt* начальное значение *Xbeg*.
3. Вычислить значение функции и вывести в виде строки таблицы.
4. Вычислить новое значение *Xt*.
5. Повторить с пункта 3, если *Xt* < *Xend.*
6. Завершить программу.

Описание входных и выходных данных

Входные данные поступают от пользователя с клавиатуры, выходные выводятся на экран. Данные имеют тип float.

Листинг программы

```
from math import *

print("Enter Xbeg, Xend, Dx")

xb = float(input("Xbeg = "))
xe = float(input("Xend = "))
dx = float(input("Dx = "))

print(f"Xbeg = {xb:7.2f}  Xend = {xe:7.2f}  Dx = {dx:7.2f}\n")

xt = xb

print("+---------+---------+")
print("|    X    |    Y    |")
print("+---------+---------+")

while xt < xe:
    if -6 <= xt <= -2:
        y = (-8/9) * ((xt + 3.5) ** 2) + 2
    elif -2 <= xt < 2:
        y = -sqrt(4 - xt ** 2)
    elif 2 <= xt < 8:
        y = log(xt, 2) - 1
    else:
        y = -2 * xt + 18
    print(f"| {xt:7.2f} | {y:7.2f} |")
    xt += dx
print("+---------+---------+")
```

Результаты и тестовые кейсы

```
Xbeg = -6

Xend = 10

Dx = 0.5

Xbeg =   -6.00  Xend =   10.00  Dx =    0.50


+---------+---------+

|    X    |    Y    |

+---------+---------+

|   -6.00 |   -3.56 |

|   -5.50 |   -1.56 |

|   -5.00 |    0.00 |

|   -4.50 |    1.11 |

|   -4.00 |    1.78 |

|   -3.50 |    2.00 |

|   -3.00 |    1.78 |

|   -2.50 |    1.11 |

|   -2.00 |    0.00 |

|   -1.50 |   -1.32 |
```

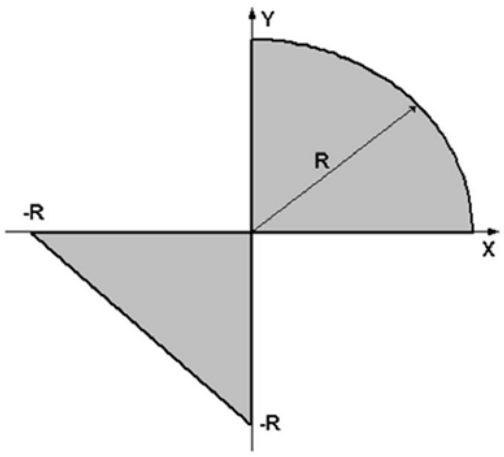
```
|   -1.00 |   -1.73 |
|   -0.50 |   -1.94 |
|    0.00 |   -2.00 |
|    0.50 |   -1.94 |
|    1.00 |   -1.73 |
|    1.50 |   -1.32 |
|    2.00 |    0.00 |
|    2.50 |    0.32 |
|    3.00 |    0.58 |
|    3.50 |    0.81 |
|    4.00 |    1.00 |
|    4.50 |    1.17 |
|    5.00 |    1.32 |
|    5.50 |    1.46 |
|    6.00 |    1.58 |
|    6.50 |    1.70 |
|    7.00 |    1.81 |
|    7.50 |    1.91 |
|    8.00 |    2.00 |
|    8.50 |    1.00 |
|    9.00 |    0.00 |
|    9.50 |   -1.00 |
+---------+---------+
```

Задание 2

Постановка задачи

Для 10 выстрелов, координаты которых задаются генератором случайных чисел, вывести текстовые сообщения о попадании в мишень.

28)



Теоретическая часть

Для решения задачи используется обновленная программа из второго задания второй лабораторной работы. В нее добавляется оператор цикла с параметром для прохождения по значениям точек заданных с помощью функции *uniform* модуля `random`, в качестве параметров минимального и максимального значения *uniform* принимается R для X и для Y.

Описание программы

Программа написана на алгоритмическом языке Python 3.6, реализована в среде ОС Windows 10 и состоит из частей, отвечающих за ввод данных, вычисление и представление данных на экране монитора.

Описание алгоритма

1. Ввести значение радиуса R в качестве ограничения.
2. Задать количество прохождений цикла и переменную счетчик.
3. В каждом цикле генерировать X и Y .
4. Проверить на принадлежность к области.
5. Вывести результат проверки принадлежности точки.
6. Вернуться к пункту 3.
7. Завершить программу когда счетчик вышел за пределы.

Описание входных и выходных данных

Входные данные поступают с клавиатуры, а выходные выводятся на монитор для просмотра. Входные и выходные данные имеют тип float.

Листинг программы

```
from math import *
from random import *

r = float(input("Enter R: "))
flag = 0

print("+---------+---------+---------+")
print("|    X    |    Y    |  Result |")
print("+---------+---------+---------+")

for n in range(10):
    x = uniform(-r, r)
    y = uniform(-r, r)
    if x >= -r and x <= 0 and y >= -x - r and y <= 0 \
        or x >= 0 and x <= r and y <= sqrt(r ** 2 - x ** 2) and y >= 0:
        flag = 1
        result = "Yes"
    else:
        flag = 0
        result = "No"

    print(f"| {x:7.2f} | {y:7.2f} | {result:^7} |")

print("+---------+---------+---------+")
```

Результаты и тестовые кейсы

```
Enter R: 5
```

X	Y	Result
0.10	3.48	Yes
1.26	-2.94	No
-0.19	-2.32	Yes
-4.48	-4.02	No
-2.06	2.25	No
-0.06	-0.05	Yes
0.35	-2.88	No
-1.70	3.14	No
-1.84	-3.54	No
0.16	1.22	Yes

```
Enter R: 7
```

X	Y	Result
-0.60	-1.14	Yes
-3.11	2.79	No
6.07	1.91	Yes
-6.41	-5.99	No
-1.63	-3.36	Yes
-4.39	-1.15	Yes
0.84	-1.07	No
4.58	-0.66	No
0.24	4.06	Yes
-5.51	2.01	No

Задание 3

Постановка задачи

Вычислить и вывести на экран в виде таблицы значения функции, заданной с помощью ряда Тейлора, на интервале от *Xbeg* до *Xend* с шагом *Dx* с точностью ε.

Таблицу снабдить заголовком и шапкой. Каждая строка таблицы должны содержать значение аргумента, значение функции и количество просуммированных членов ряда.

$$(1-x)^{-4} = 1 + \frac{1}{1*2*3}(2*3*4*x + 3*4*5*x^2 + 4*5*6*x^3 + 5*6*7*x^4 + \cdots)$$
$$|x| \le 1$$

Теоретическая часть

Ряд Тейлора - это разложение функции в бесконечный ряд в окрестности некоторой точки. В данном случае используется разложение функции $(1-x)^{-4}$.

Область сходимости ряда определяется условием |*x*|<1. При |*x*|≥1 ряд расходится, что проверяется в коде с помощью функции fabs().

Формула разложения функции $(1-x)^{-4}$ в ряд Тейлора:

$$(1-x)^{-4} = 1 + 4x + 10x^2 + 20x^3 + 35x^4 + \cdots$$

Рекуррентная формула для вычисления следующего члена ряда:

an+1=*an*·*x*·*nn*+3, где *a*1=4*x*

Коэффициенты в разложении представляют собой числа, которые можно получить по формуле:

Cn=*n*!3!(*n*+3)!

Точность вычислений определяется параметром ϵ (эпсилон). Вычисления продолжаются до тех пор, пока модуль очередного члена ряда не станет меньше заданной точности.

Проверка корректности входных данных осуществляется путем контроля условия |*x*|<1. При нарушении этого условия вычисление невозможно, так как ряд расходится.

NaN (Not a Number) используется в коде для обозначения результатов, которые не могут быть вычислены (например, при |*x*|≥1).

Абсолютная величина (модуль) числа используется для:

- Проверки условия сходимости ряда
- Определения точности вычислений
- Корректной работы с отрицательными значениями

Метод вычисления включает:

1. Инициализацию первого члена ряда
2. Вычисление последующих членов по рекуррентной формуле
3. Суммирование членов ряда
4. Подсчет количества членов

5. Проверку условия окончания вычислений

Сравнение результатов производится между:

- Приближенным значением (сумма ряда)
- Точным значением функции $(1-x)^{-4}$

Такой подход позволяет оценить точность вычислений и количество членов ряда, необходимых для достижения заданной точности.

Описание программы

Программа написана на алгоритмическом языке Python 3.6, реализована в среде ОС Windows 10 и состоит из частей, отвечающих за ввод данных, вычисление и представление данных на экране монитора.

Описание алгоритма

1. Импорт модуля

- Подключается модуль math для использования функции fabs() (абсолютное значение)

2. Функция taylor_series()

- Принимает два параметра: x (значение аргумента) и epsilon (точность вычислений)
- Проверяет условие сходимости: если |x| >= 1, возвращает NaN и 0
- Инициализирует начальные значения:
    - result = 1.0 (первый член ряда)
    - n = 1 (номер текущего члена)
    - term = 4*x (второй член ряда)
    - total_terms = 2 (счетчик членов ряда)
- Выполняет основной цикл:
    - Увеличивает номер члена n
    - Вычисляет следующий член по формуле: term *= x * (n + 3) / n
    - Добавляет член к результату: result += term
    - Увеличивает счетчик членов: total_terms += 1
    - Проверяет условие выхода: если |term| < epsilon, завершает цикл
- Возвращает результат и количество членов ряда

3. Функция exact_value()

- Вычисляет точное значение функции $(1-x)^{-4}$
- Проверяет условие |x| < 1
- При нарушении условия возвращает NaN

4. Основной алгоритм программы

- Вывод приветствия и информации о функции
- Ввод параметров:
    - xb (начальное значение x)
    - xe (конечное значение x)
    - dx (шаг)
    - epsilon (точность)

- Проверка корректности входных данных:
    - Если |xb| >= 1 или |xe| >= 1, программа завершается с ошибкой

5. Вывод результатов

- Формируется шапка таблицы
- Организуется цикл по x:
    - Вычисляется приближенное значение через taylor_series()
    - Вычисляется точное значение через exact_value()
    - Проверяется корректность результатов
    - Формируется строка таблицы
    - Значение x увеличивается на dx
- Выводится нижняя граница таблицы

6. Особенности реализации

- Используется round(x + dx, 10) для учета погрешности округления
- При выводе используется форматирование строк:
    - Ширина полей: 13 символов для x, 13.6f
    - Точность: 13.9f для значений функции
- Проверка на NaN выполняется через exact == exact

7. Последовательность выполнения

1. Инициализация
2. Ввод данных
3. Проверка корректности
4. Формирование таблицы
5. Вычисление значений
6. Вывод результатов

8. Условия завершения

- Программа завершается:
    - При ошибке ввода (|x| >= 1)
    - После вывода всех значений в заданном диапазоне

Описание входных и выходных данных

Входные данные поступают от пользователя с клавиатуры, выходные выводятся на экран. Данные имеют тип float.

Листинг программы

```
from math import fabs

def taylor_series(x, epsilon):
    """Вычисление (1-x)^(-4) с помощью ряда Тейлора"""
    if fabs(x) >= 1:
        return float('nan'), 0

    result = 1.0  # Первый член ряда (1)
    n = 1  # Номер текущего члена
    term = 4 * x  # Второй член ряда (4x)
    total_terms = 2  # Уже учли два члена

    if fabs(term) < epsilon:
        return result, 1
```

```
    result += term

    while True:
        n += 1
        # Рекуррентная формула для следующего члена:
        term *= x * (n + 3) / n
        result += term
        total_terms += 1

        if fabs(term) < epsilon:
            break

    return result, total_terms


def exact_value(x):
    """Точное значение функции (1-x)^(-4)"""
    return (1 - x) ** (-4) if fabs(x) < 1 else float('nan')


# Ввод параметров
print("Вычисление функции (1-x)^(-4) с помощью ряда Тейлора")
print("Ряд: (1-x)^(-4) = 1 + 4x + 10x² + 20x³ + 35x⁴ + ...")
xb = float(input("Введите Xbeg (|x| < 1): "))
xe = float(input("Введите Xend (|x| < 1): "))
dx = float(input("Введите шаг Dx: "))
epsilon = float(input("Введите точность ε: "))

# Проверка корректности ввода
if fabs(xb) >= 1 or fabs(xe) >= 1:
    print("Ошибка: |x| должен быть < 1")
    exit()

# Вывод шапки таблицы
print("\n{:^80}".format("Таблица значений функции (1-x)^(-4)"))
print("+---------------+---------------+---------------+---------------+")
print("|       X       |  Приближенное |     Точное    | Членов ряда   |")
print("+---------------+---------------+---------------+---------------+")

# Вычисление и вывод значений
x = xb
while x <= xe + 1e-10:  # Учет погрешности округления
    approx, terms = taylor_series(x, epsilon)
    exact = exact_value(x)

    if exact == exact:  # Проверка на NaN
        print(f"| {x:13.6f} | {approx:13.9f} | {exact:13.9f} | {terms:13d} |")
    else:
        print(f"| {x:13.6f} | {'NaN':^13} | {'NaN':^13} | {'NaN':^13} |")

    x = round(x + dx, 10)

print("+---------------+---------------+---------------+---------------+")
```

Результаты работы программы

```
Вычисление функции (1-x)^(-4) с помощью ряда Тейлора
Ряд: (1-x)^(-4) = 1 + 4x + 10x² + 20x³ + 35x⁴ + ...
Введите Xbeg (|x| < 1): -0.9
Введите Xend (|x| < 1): 0.9
Введите шаг Dx: 0.1
Введите точность ε: 0.0001

                    Таблица значений функции (1-x)^(-4)
+---------------+---------------+---------------+---------------+
|       X       |  Приближенное |    Точное     | Членов ряда   |
+---------------+---------------+---------------+---------------+
|     -0.900000 |   0.076686508 |   0.076733604 |           226 |
|     -0.800000 |   0.095217123 |   0.095259869 |            96 |
|     -0.700000 |   0.119690770 |   0.119730367 |            56 |
|     -0.600000 |   0.152624996 |   0.152587891 |            37 |
|     -0.500000 |   0.197495878 |   0.197530864 |            26 |
|     -0.400000 |   0.260337180 |   0.260308205 |            19 |
|     -0.300000 |   0.350103900 |   0.350127797 |            14 |
|     -0.200000 |   0.482259046 |   0.482253086 |            11 |
|     -0.100000 |   0.683024000 |   0.683013455 |             7 |
|      0.000000 |   1.000000000 |   1.000000000 |             1 |
|      0.100000 |   1.524144000 |   1.524157903 |             7 |
|      0.200000 |   2.441396326 |   2.441406250 |            11 |
|      0.300000 |   4.164880614 |   4.164931279 |            14 |
|      0.400000 |   7.715971314 |   7.716049383 |            19 |
|      0.500000 |  15.999878109 |  16.000000000 |            26 |
|      0.600000 |  39.062327594 |  39.062500000 |            37 |
|      0.700000 | 123.456530327 | 123.456790123 |            56 |
|      0.800000 | 624.999557628 | 625.000000000 |            96 |
|      0.900000 | 9999.998983678 | 10000.000000000 |           226 |
+---------------+---------------+---------------+---------------+

Process finished with exit code 0
```

Список используемой литературы

1. Н.А. Прохоренок, В.А. Дронов, Python 3 и PyQt 5. Разработка приложений: СПб.: БХВ-Петербург, 2017

2. В.П. Рядченко, Методическое пособие по выполнению лабораторных работ.