Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота №1

з дисципліни
«Алгоритми і структури даних»

Виконав:

студент групи ІМ-43
Балалаєв Максим Юрійович
номер варіанту: 3

Перевірив:

Сергієнко А. М.

Київ 2025

Завдання

Дане натуральне число *n*. Знайти суму перших *n* членів ряду чисел, заданого рекурентною формулою. Розв'язати задачу трьома способами:

1) у програмі використати рекурсивну функцію, яка виконує обчислення і членів ряду, і суми на рекурсивному спуску;

2) у програмі використати рекурсивну функцію, яка виконує обчислення і членів ряду, і суми на рекурсивному поверненні;

3) у програмі використати рекурсивну функцію, яка виконує обчислення членів ряду на рекурсивному спуску, а обчислення суми на рекурсивному поверненні.

Варіант № 3

$$F_1 = 1; \quad F_i = -F_{i-1} \cdot x \cdot (2i-3)/(2i-2), \quad i > 1;$$
$$\sum_{i=1}^{n} F_i = 1/\sqrt{1+x}, \quad |x| < 1.$$

Текст програм

Завдання 1 - обчислення членів ряду й суми на рекурсивному спуску

```
#include <stdio.h>

double sum_descending(const unsigned n, const double x, const int i,
const double prev_F, double sum)
{
    double F;
    if (i > n)
    {
        return sum;
    }
    else if (i == 1)
    {
        F = 1;
    }
    else
    {
        F = -prev_F * x * (2 * i - 3) / (2 * i - 2);
    }
    sum += F;

    return sum_descending(n, x, i + 1, F, sum);
}

double sum_wrapper(const unsigned n, const double x)
{
```

```
    return sum_descending(n, x, 1, 1, 0);
}

int main()
{
    double x;
    printf("Enter x: ");
    scanf("%lf", &x);

    int n;
    printf("Enter n: ");
    scanf("%i", &n);

    if (n < 1 || x > 1 || x < -1)
    {
        printf("Incorrect input\n");
        return 1;
    }

    double result = sum_wrapper(n, x);
    printf("Result: %lf\n", result);

    return 0;
}
```

Завдання 2 - обчислення членів ряду й суми на рекурсивному поверненні

```
#include <stdio.h>

typedef struct
{
    double sum;
    double last_F;
} Result;

Result sum_ascending(const unsigned n, const double x)
{
    if (n == 1)
        return (Result){.sum = 1, .last_F = 1};
    else
    {
        Result prev = sum_ascending(n - 1, x);
        double F = -prev.last_F * x * (2 * n - 3) / (2 * n - 2);

        return (Result){.sum = prev.sum + F, .last_F = F};
    }
}

int main()
{
    double x;
    printf("Enter x: ");
```

```
    scanf("%lf", &x);

    int n;
    printf("Enter n: ");
    scanf("%i", &n);

    if (n < 1 || x > 1 || x < -1)
    {
        printf("Incorrect input\n");
        return 1;
    }

    double result = sum_ascending(n, x).sum;
    printf("Result: %lf\n", result);

    return 0;
}
```

Завдання 3 - обчислення членів ряду на рекурсивному спуску, а суми - на поверненні

```
#include <stdio.h>

double sum_mixed(const unsigned n, const double x, const int i, const
double prev_F)
{
    if (i > n)
        return 1;

    double F = -prev_F * x * (2 * i - 3) / (2 * i - 2);
    return F + sum_mixed(n, x, i + 1, F);
}

double sum_wrapper(const unsigned n, const double x)
{
    return sum_mixed(n, x, 2, 1);
}

int main()
{
    double x;
    printf("Enter x: ");
    scanf("%lf", &x);

    int n;
    printf("Enter n: ");
    scanf("%i", &n);

    if (n < 1 || x > 1 || x < -1)
    {
        printf("Incorrect input\n");
        return 1;
```

```
    }

    double result = sum_wrapper(n, x);
    printf("Result: %lf\n", result);

    return 0;
}
```

Програма для перевірки – обчислення за допомогою циклу for

```
#include <stdio.h>

double sum_loop(const unsigned n, const double x)
{
    double sum = 1, F = 1;
    for (int i = 2; i <= n; i++)
    {
        F *= -x * (2 * i - 3) / (2 * i - 2);
        sum += F;
    }

    return sum;
}

int main()
{
    double x;
    printf("Enter x: ");
    scanf("%lf", &x);

    int n;
    printf("Enter n: ");
    scanf("%i", &n);

    if (n < 1 || x > 1 || x < -1)
    {
        printf("Incorrect input\n");
        return 1;
    }

    double result = sum_loop(n, x);
    printf("Result: %lf\n", result);

    return 0;
}
```

Тестування програм

```
C:\Users\Flagrate\Desktop\КПІ\Лаба АСД\summer-2025\1>"1 (descent).exe"
Enter x: 0.5
Enter n: 5
Result: 0.821777

C:\Users\Flagrate\Desktop\КПІ\Лаба АСД\summer-2025\1>"2 (ascent).exe"
Enter x: 0.5
Enter n: 5
Result: 0.821777

C:\Users\Flagrate\Desktop\КПІ\Лаба АСД\summer-2025\1>"3 (mixed).exe"
Enter x: 0.5
Enter n: 5
Result: 0.821777

C:\Users\Flagrate\Desktop\КПІ\Лаба АСД\summer-2025\1>loop_calc.exe
Enter x: 0.5
Enter n: 5
Result: 0.821777
```

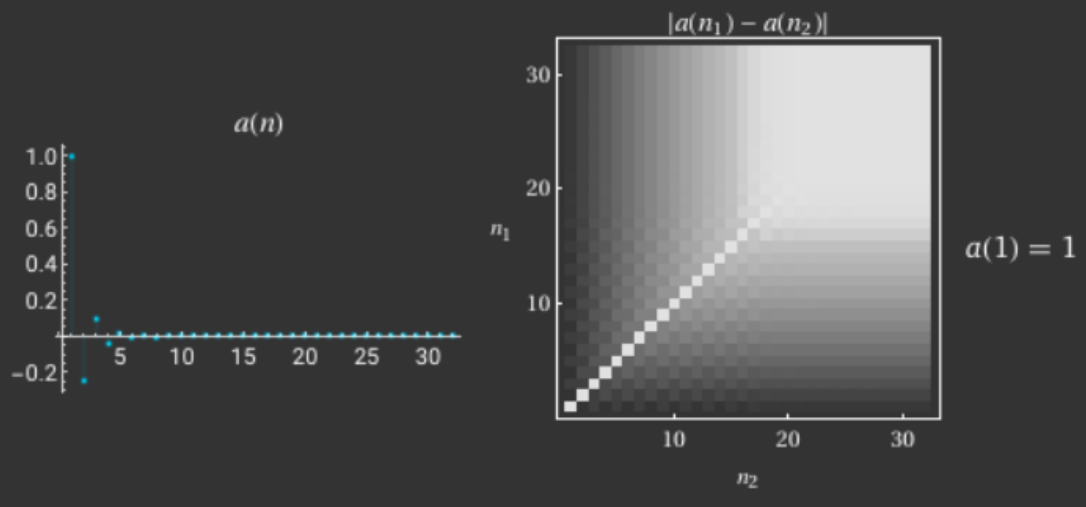
Також із метою тестування було виконано розрахунок значення рекурентної формули для **n = 5** за допомогою онлайн-калькулятора *Wolfram Alpha*

Wolfram Alpha підтримує послідовності:

Input

$$a(n) = -a(n-1)\left(0.5 \times \frac{2n-3}{2n-2}\right)$$

Value plot and recurrence plot



Values

n	1	2	3	4	5
$a(n)$	1	−0.25	0.09375	−0.0390625	0.0170898

Input interpretation

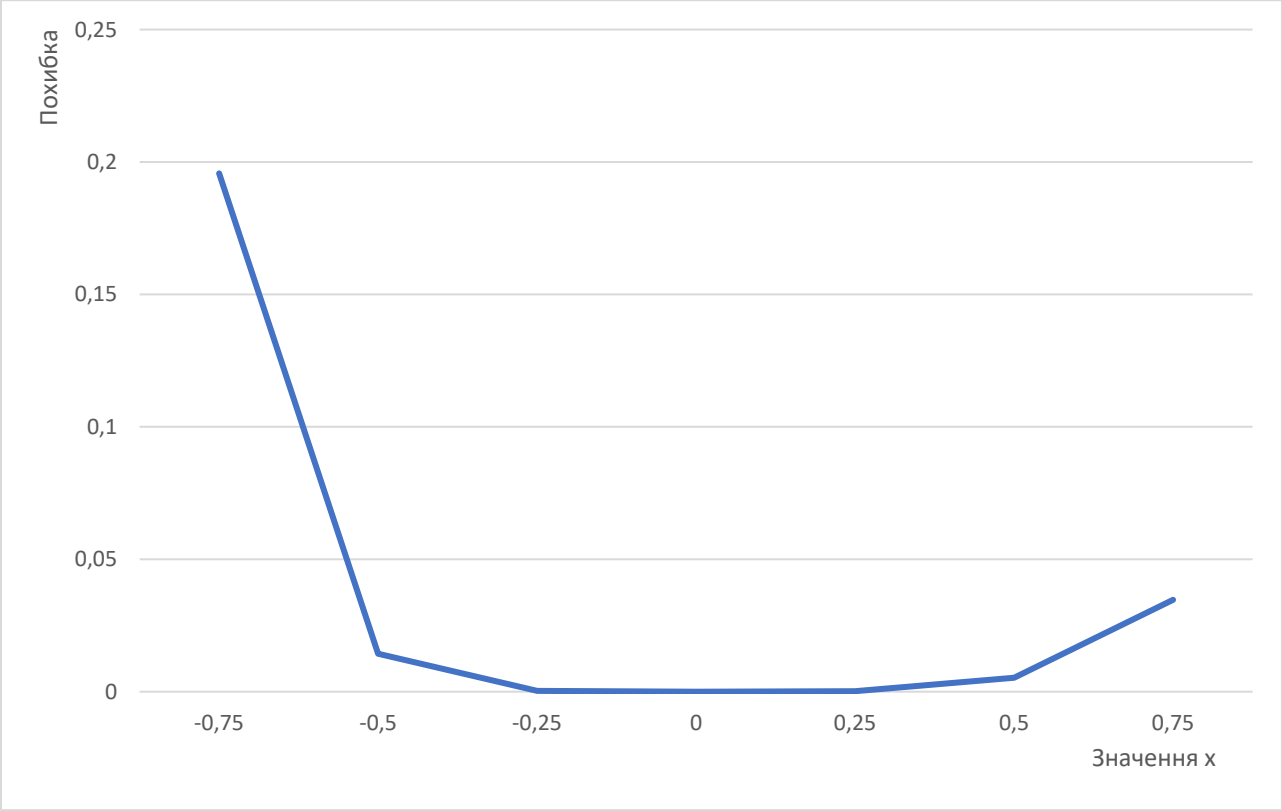
$1 - 0.25 + 0.09375 - 0.0390625 + 0.0170898$

Result

0.8217773

Графік похибки обчислення функції

n = 5



Висновки

У цій лабораторній роботі виконано наближене обчислення значення математичної функції із застосуванням рекурентної формули мовою програмування С. Це було зроблено 3 шляхами:

1) обчислення і членів ряду, і суми виконується на рекурсивному спуску;

2) обчислення і членів ряду, і суми виконується на рекурсивному поверненні;

3) обчислення членів ряду виконується на рекурсивному спуску, а обчислення суми на рекурсивному поверненні.

Для перевірки результатів роботи було також створено циклічний алгоритм, який розраховує ту ж формулу, і використано онлайн-калькулятор Wolfram Alpha. Всі результати обчислень збіглись.

Було проведено аналіз похибки обчислення функції та побудовано графік залежності значення похибки від значення *x*. Похибка є нерівноцінною для різних значень x, оскільки для обчислюваної функції – $\frac{1}{\sqrt{1+x}}$ – y=0 є асимптотою, тобто значення функції нескінченно прямує до 0 зі зростанням аргумента.

Рекурсія – це метод програмування, який дає змогу функції багаторазово викликати себе доти, доки не буде виконано умову завершення (базовий випадок, або сигналізуюча функція). Вона дає змогу розбивати складні проблеми на підзадачі, а потім розв’язувати їх за допомогою однієї й тієї самої техніки. Це дає змогу уникнути використання складних структур керування, як-от цикли, і замість цього застосовувати чистий модульний підхід.

Але у рекурсії є й недоліки. Погано прописана сигналізуюча функція призведе до нескінченної рекурсія - тобто, функція викликатиме сама себе без упину, використовуючи ресурси машини, доки не буде зупинена переповненням стеку.