МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский государственный электротехнический университет
„ЛЭТИ“ им. В.И. Ульянова (Ленина)»
(СПбГЭТУ)

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ИНФОРМАТИКИ

Кафедра вычислительной техники

Отчёт о лабораторной работе № 2

по дисциплине
«Программирование»

на тему
«Вычисление суммы ряда»

Выполнили:

студенты группы 0321 _____ (Климанов М.)

_____ (Русских В.)

_____ (Феллер В.)

Проверил:

Преподаватель каф. ВТ, к.т.н., доцент _____ (Миронов С.Э.)

Санкт-Петербург, 2020

СОДЕРЖАНИЕ

1 Цель работы

Целью этой лабораторной работы является изучение математических и циклических операторов языка C и получение практических навыков создания программ для эффективного вычисления суммы ряда.

2 Задание

Дан функциональный числовой ряд

$$S(x) = \sum_{m=1}^{\infty} (-1)^{m+1} \cdot \frac{2^{m-1}}{x^m} = \frac{1}{x} - \frac{2}{x^2} + \frac{4}{x^3} - \frac{8}{x^4} + \dots \tag{1}$$

Необходимо определить его область сходимости и написать программу на языке C, которая эффективно вычисляет его сумму в заданной точке с заданной допустимой погрешностью $\pm\varepsilon$. Иными словами нужно вычислить такую частичную сумму $S_n(x) = \sum_{m=1}^{n} (-1)^{m+1} \cdot \frac{2^{m-1}}{x^m}$, что

$$|S(x) - S_n(x)| < \varepsilon \tag{2}$$

3 Ход работы

В ходе выполнения лабораторной работы сначала была написана наивная реализация алгоритма вычисления суммы ряда (1). После этого последовательно были найдены две возможности вычислить сумму этого ряда более эффективно. Исходный код всех трёх реализаций приведён в приложении А.

3.1 Сходимость ряда

Область сходимости ряда (1) можно найти с помощью радикального признака сходимости Коши [1, с. 298]:

$$\lim_{n\to\infty}\sqrt[n]{\frac{2^{n-1}}{|x^n|}} = \lim_{n\to\infty}\frac{2}{\sqrt[n]{2}\cdot|x|} = \frac{2}{|x|} < 1 \implies x \in (-\infty;-2)\cup(2;+\infty) \qquad (3)$$

Таким образом, ряд (1) сходится абсолютно на области $(-\infty;-2)\cup(2;+\infty)$. Когда программе на вход подаётся значение x, не входящее в эту область, она должна завершаться и сообщать об ошибке.

3.2 Наивный подход

Известно, что остаток знакочередующегося ряда не превышает по модулю первого отброшенного слагаемого [1, с. 330]. Этот факт позволяет использовать следующий критерий остановки алгоритма: *алгоритм должен прекратить суммирование, когда последнее вычисленное слагаемое по модулю меньше* ε.

Наивная реализация приведена в листинге А.1. Из этого листинга видно, что на каждой итерации рабочего цикла программы используется 1 операция деления, 2 операции умножения, 1 операция сложения, операция сравнения, условный оператор и вызов функции `fabs()`. В следующей версии программы количество операций в цикле значительно сократится.

3.3 Схема Горнера

Чтобы исключить операцию сравнения и условный оператор на каждой итерации цикла, можно заранее вычислить такое n, что $|S(x) - S_n(x)| < \varepsilon$. Тогда количество итераций рабочего цикла будет известно на момент его начала и лишние сравнения не понадобятся.

Сначала нужно найти такое n, что $|a_{n+1}| < \varepsilon$, преобразовав это неравенство:

$$|a_{n+1}| = \left|\frac{2^n}{x^{n+1}}\right| = \frac{2^n}{|x|^{n+1}} < \varepsilon \implies n \geq \frac{\ln 2\varepsilon}{\ln\frac{2}{|x|}} \qquad (4)$$

Теперь, если $|S(x) - S_n(x)| < |a_{n+1}|$ согласно оценке остатка знакочередующегося ряда, и $|a_{n+1}| < \varepsilon$, то $|S(x) - S_n(x)| < \varepsilon$ при любом найденном в (4) n, что и требовалось получить.

Если $x < -2$, то $S(x) = -\frac{1}{|x|} - \frac{2}{|x|^2} - \frac{4}{|x|^3} - \ldots$ То есть, ряд (1) перестаёт быть знакочередующимся. Это значит, что оценка n в (4) в таком случае некорректна и необходимо вычислять n по-другому.

Чтобы получить оценку n для случая $x < -2$, можно вычислить остаток ряда

$$R(x) = \sum_{m=1}^{\infty} \left(\frac{2}{|x|}\right)^m \tag{5}$$

Остаток ряда (5) будет больше остатка ряда (1) при $x < -2$, т.к. $\left(\frac{2}{|x|}\right)^m > \frac{2^{m-1}}{|x|^m}$ для всех m. Остаток ряда (5) можно вычислить точно как остаток геометрического ряда:

$$R(x) - R_n(x) = \frac{\frac{2}{|x|}\left(1 - \left(\frac{2}{|x|}\right)^n\right)}{1 - \frac{2}{|x|}} - \frac{\frac{2}{|x|}}{1 - \frac{2}{|x|}} = \frac{2^{n+1}}{|x|^n(|x| - 2)} \tag{6}$$

Новую оценку n можно получить аналогично (4), преобразовав неравенство:

$$\frac{2^{n+1}}{|x|^n(|x| - 2)} < \varepsilon \implies n > \frac{\ln\frac{\varepsilon}{2} + \ln(|x| - 2)}{\ln\frac{2}{|x|}} \tag{7}$$

$S_n(x)$ можно вычислять ещё более эффективно, преобразовав его по схеме Горнера [2, с. 538] следующим образом:

$$\begin{aligned} S_n(x) &= \frac{(-1)^2}{2} \cdot \frac{2}{x} + \frac{(-1)^3}{2} \cdot \left(\frac{2}{x}\right)^2 + \ldots + \frac{(-1)^{n+1}}{2} \cdot \left(\frac{2}{x}\right)^n = \\ &= \left(\left(\ldots \left(\frac{2}{x} \cdot \frac{(-1)^{n+1}}{2} + \frac{(-1)^n}{2} \right) \frac{2}{x} + \ldots + \frac{(-1)^3}{2} \right) \frac{2}{x} + \frac{(-1)^2}{2} \right) \frac{2}{x} \end{aligned} \tag{8}$$

Реализация вычисления суммы ряда с разложением по схеме Горнера приведена в листинге А.2. Здесь на каждой итерации рабочего цикла используется только 1 умножение и 2 сложения.

3.4 Нахождение функции, в которую раскладывается ряд

Поскольку ряд (1) сходится абсолютно, его слагаемые можно произвольно переставлять, и сумма ряда при этом не изменится. Его можно представить как разность двух геометрических рядов и получить очень простую элементарную функцию, в которую раскладывается ряд:

$$\begin{aligned} S(x) &= \sum_{m=1}^{\infty} (-1)^{m+1} \cdot \frac{2^{m-1}}{x^m} = \frac{1}{2} \left(\sum_{m=1}^{\infty} \left(\frac{2}{x} \right)^{2m-1} - \sum_{m=1}^{\infty} \left(\frac{2}{x} \right)^{2m} \right) \\ &= \frac{1}{x+2} \end{aligned} \tag{9}$$

Реализация вычисления суммы ряда с использованием функции $\frac{1}{x+2}$ приведена в листинге А.3. Здесь сумма ряда вычисляется с точностью, которую могут дать числа с плавающей точкой двойной точности (обычно около 15 десятичных цифр после запятой).

4 Тестирование программ

В этом разделе будет проведена проверка корректности работы созданных программ путём запуска тест-кейсов для каждого варианта реализации. В начале будет протестирована самая точная реализация, которая будет использована как эталон для обнаружения недостатков в других реализациях.

4.1 Тестирование абсолютно точной реализации

В таблице 1 приведены входные значения x и соответствующие им выходные значения $S(x)$, произведённые программой из листинга А.3.

Таблица 1 — Результаты тестов программы А.3

x	$S(x)$
-10	$-0{,}125$
-3	-1
$-2{,}001$	-1000
$2{,}01$	$0{,}249377$
4	$0{,}166667$
8	$0{,}1$

4.2 Тестирование наивной реализации

В таблице 2 приведены входные значения x и ε, а также соответствующие им выходные значения n и $S_n(x)$, произведённые программой из листинга А.1. С целью оценки погрешности в ней также приведены остатки частичной суммы ряда. Серым цветом в этой таблице выделены тест-кейсы, в которых остаток превысил заданное значение ε.

Как уже было замечено, оценка остатка с помощью последнего отброшенного члена ряда некорректна для отрицательных x. Именно поэтому в строке 3 реальный остаток суммы ряда превысил заданный ε.

Чем ближе x к точкам -2 и 2 при фиксированном ε, тем медленнее убывают слагаемые в ряде и тем больше членов ряда необходимо вычислить, чтобы получить суму ряда с заданной точностью. Например, в строке 4 $n = 1024$. Это значит, что программе необходимо вычислить 1024-й член ряда: $\frac{2^{1023}}{x^{1024}}$. При этом максимально допустимое значение показателя в числах с плавающей точкой двойной точности равно 1023. Как следствие, происходит переполнение числа с плавающей точкой и расчёты завершаются значительно раньше, чем требуется для достижения нужной точности.

Таблица 2 — Результаты тестов программы А.1

x	ε	n	$S_n(x)$	$\lvert S(x) - S_n(x)\rvert$
-10	0,1	2	$-0{,}12$	0,005
-10	0,01	3	$-0{,}124$	0,001
-3	0,0001	22	$-0{,}999866$	0,000134
$-2{,}001$	0,001	1024	$-400{,}327837$	599,672163
2,01	0,01	785	0,254348	0,004971
4	0,0001	13	0,166687	0,00002
8	0,00001	8	0,099998	0,000002

4.3 Тестирование реализации по схеме Горнера

В таблице 3 приведены тест-кейсы для программы А.2. Из сравнения таблиц 2 и 3 видно, что недостатки, характерные для наивной реализации, исправлены в реализации по схеме Горнера.

Таблица 3 — Результаты тестов программы А.2

x	ε	n	$S_n(x)$	$\lvert S(x) - S_n(x)\rvert$
-10	0,1	1	$-0{,}1$	0,025
-10	0,01	3	$-0{,}124$	0,001
-3	0,0001	25	$-0{,}999960$	0,00004
$-2{,}001$	0,001	29025	$-999{,}999500$	0,0005
2,01	0,01	785	0,254348	0,004971
4	0,0001	13	0,166687	0,00002
8	0,00001	8	0,099998	0,000002

5 Выводы

В ходе лабораторной работы изучены математические и циклические операторов языка C, написаны три программы, вычисляющие один и тот же ряд с разной степенью эффективности. Изучены способы оптимизации вычисления многочленов (схемы Горнера и Эстрина). Использованные техники оптимизации вычислений можно использовать для более широкого класса задач (вычисление других рядов, сумм, многочленов).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Фихтенгольц Г.М. Курс дифференциального и интегрального исчисления. В 3 т. Т. II / Пред. и прим. А.А. Флоринского. : 8-е изд. — М. : ФИЗМАТЛИТ, 2003. ISBN 5-9221-0157-9.

[2] Кнут, Дональд Эрвин. Искусство программирования, т. 2. Получисленные алгоритмы, 3-е изд. : Пер. с англ. — М. : ООО «И.Д. Вильямс», 2017. ISBN 978-5-8459-0081-4 (рус.).

ПРИЛОЖЕНИЕ А

Исходный код программ

Листинг А.1 — Наивное вычисление суммы ряда

```c
#include <stdio.h>
#include <math.h>


int main() {
    int MAX_SERIES_N = 10000; // Максимальный номер члена ряда.
    int i;
    double eps;               // epsilon -- такое число, что остаток ряда не должен его превысить.
    double x, initial_x, series_sum = 0, series_term, numerator = 1;

    printf("S = (1 / x) - (2 / x^2) + (4 / x^3) - (8 / x^4) + ...\n");
    printf("x = ");
    scanf("%lf", &x);

    if (x >= -2.0 && x <= 2.0) {
        // Ряд расходится, программа завершается.
        printf("\nРяд расходится в заданной точке x.\n");
        return 0;
    }
    initial_x = x;

    printf("epsilon = ");
    scanf("%lf", &eps);

    if (eps <= 0) {
        // Некорректный epsion
        printf("\nEpsilon -- это требуемый остаток ряда. Он должен быть положительным.\n");
        return 0;
    }

    printf("\n...Вычисления...\n");

    for (i = 1; i <= MAX_SERIES_N; ++i) {
        series_term = numerator / x;
        series_sum += series_term;
        numerator *= -2;                    // Минус обеспечивает чередование знаков.
        x *= initial_x;

        if (fabs(series_term) < eps) {
            printf("Член ряда меньше epsilon. Остановка вычислений...\n");
            ++i;
            break;
        }
    }

    printf("Номер последнего вычисленного элемента ряда: %d\n", i - 1);
    printf("Частичная сумма ряда S_%d = %lf\n", i - 1, series_sum);

    return 0;
}
```

Листинг А.2 — Вычисление суммы ряда по схеме Горнера

```c
#include <stdio.h>
#include <math.h>


int main() {
    int MAX_SERIES_N = 1000000; // Максимальный номер члена ряда.
    int i, n;
    double eps;                 // epsilon -- такое число, что остаток ряда не должен его превысить.
    double x, a, b;

    printf("S = (1 / x) - (2 / x^2) + (4 / x^3) - (8 / x^4) + ...\n");
    printf("x = ");
    scanf("%lf", &x);

    if (x >= -2.0 && x <= 2.0) {
        // Ряд расходится, программа завершается.
        printf("\nРяд расходится в заданной точке x.\n");
        return 0;
    }

    printf("epsilon = ");
    scanf("%lf", &eps);

    if (eps <= 0) {
        // Некорректный epsion
        printf("\nEpsilon -- это требуемый остаток ряда. Он должен быть положительным.\n");
        return 0;
    }

    printf("\n...Вычисления...\n");

    if (x > 2) {
        n = (int) (log(2.0 * eps) / log(2.0 / fabs(x))) + 1;
    }
    else {
        // При отрицательных x ряд больше не знакочередующийся, а
        // значит и оценка остатка совсем другая.
        n = (int) ((log(eps / 2.0) + log(fabs(x) - 2.0)) / log(2.0 / fabs(x))) + 1;
    }

    n = (n > 0) ? n : 1;
    n = (n > MAX_SERIES_N) ? MAX_SERIES_N : n;
    x = 2 / x;
    printf("Для обеспечения требуемой точности нужно n = %d членов ряда.\n", n);

    a = (n % 2 == 1) ? 0.5 : -0.5;
    b = a;
    for (i = 0; i < n - 1; ++i) {
        a = -a;
        b = a + b * x;
    }
    b = b * x;

    printf("Частичная сумма ряда S_%d = %lf\n", n, b);

    return 0;
}
```

Листинг А.3 — Вычисление суммы ряда в замкнутой форме

```c
#include <stdio.h>


int main() {
    double x;

    printf("S = (1 / x) - (2 / x^2) + (4 / x^3) - (8 / x^4) + ...\n");
    printf("x = ");
    scanf("%lf", &x);

    if (x >= -2.0 && x <= 2.0) {
        // Ряд расходится, программа завершается.
        printf("\nРяд расходится в заданной точке x.\n");
    }
    else {
        printf("Сумма ряда S = %lf\n", 1 / (x + 2));
    }

    return 0;
}
```