Московский Авиационный институт

(Национальный исследовательский университет)

Курсовой проект

по курсам

“Архитектура компьютера”, “Программные и аппаратные средства информатики”

1 семестр

Задание 3

Студент: Белоносов К.А.

Группа: М8О-103Б-21

Руководитель: Севастьянов В. С.

Оценка:

Дата:

Подпись:

Москва, 2021

Содержание

Введение

Составить программу на Си, которая печатает таблицу значений элементарной функции, вычисленной двумя способами: по формуле Тейлора и с помощью встроенных функций языка программирования. В качестве аргументов таблицы взять точки разбиения отрезка [a, b] на n равных частей (n + 1 точка, включая концы отрезка), находящихся в рекомендованной области хорошей точности формулы Тейлора. Вычисления по формуле Тейлора проводить по экономной в сложностном смысле схеме с точностью ε*k, где ε – машинное эпсилон аппаратно реализованного вещественного типа для данной ЭВМ, а k – экспериментально подбираемый коэффициент, обеспечивающий приемлемую сходимость. Число итераций должно ограничиваться сверху числом порядка 100. Программа должна сама определять машинное ε и обеспечивать корректные размеры генерируемой таблицы.

Вариант

Вариант №3

3	$x-\frac{5}{2}x^2+\ldots+\frac{(-1)^{n+1}\cdot 2^n-1}{n}x^n$	–0.2	0.3	$\ln(1+x-2x^2)$

Программное обеспечение

Операционная система семейства: linux, наименование: ubuntu, версия 20.04.3 LTS

Интерпретатор команд: bash версия 5.0.17(1)

Система программирования VS Code, редактор текстов emacs версия 27.1

Описание работы, алгоритм

Программа в цикле вычисляет значение функции с помощью ряда Тейлора в точках заданного отрезка. Вычисление происходит до тех пор, пока число итераций меньше 100 или значение очередного слагаемого станет меньше машинного эпсилона, умноженного на константу. Также вычисляется значение функции “обычным” методом.

По ходу вычислений печатается таблица. В первом столбце – текущий x, во второй – значение функции по ряду Тейлора, в третьей – значение самой функции, а в четвертой – число пройденных итераций ряда Тейлора.

Описание переменных

Переменная	**Тип**	**Назначение**
Epsillon	double	Машинный эпсилон
Taylor		Текущее значение ф-ии по Тейлору
a		Левая граница отрезка
b		Правая граница отрезка
x		Текущий аргумент ф-ии
function		Значение ф-ии
Buffer		Промежуточное хранение вычислений
n	int	Число разбиений отрезка
Counter		Счетчик

Проверка программы

Входные данные – число n. В тестах n = 10. Также число k – в тестах 10 и 1

Выходные данные:

```
Машинное эпсилон для типа double в системе VS Code = 2.220446e-16
Число n:
10
Число k:
10
Таблица значений ряда Тейлора и стандартной функции для f(x) = (((-1)^(n + 1) * 2^n - 1)*x^n)/n
```

x	Тейлор	f(x)	число итераций
-0.20	-0.32850407442261625456936260	-0.32850407442261669865857240	35
-0.15	-0.21691300452529474473983840	-0.21691300452529518882904820	27
-0.10	-0.12783337150988477448620980	-0.12783337150988488550851230	21
-0.05	-0.05657034959618328223696080	-0.05657034959618333080921810	15
0.00	0.00000000298023221934062840	0.00000000298023227873755980	3
0.05	0.04401688883905046506539450	0.04401688883905064547663600	15
0.10	0.07696104444749735096209520	0.07696104444749746198439770	21
0.15	0.09984533766675858512673610	0.09984533766675871002682640	27
0.20	0.11332868690355582530493450	0.11332868690355611673847850	35
0.25	0.11778303565638278960214790	0.11778303565638326144693340	45
0.30	0.11332868317826609583232060	0.11332868317826561010974730	60

```
Машинное эпсилон для типа double в системе VS Code = 2.220446e-16
Число n:
10
Число k:
1
Таблица значений ряда Тейлора и стандартной функции для f(x) = (((-1)^(n + 1) * 2^n - 1)*x^n)/n
```

x	Тейлор	f(x)	число итераций
-0.20	-0.32850407442261669865857240	-0.32850407442261669865857240	37
-0.15	-0.21691300452529510558902300	-0.21691300452529518882904820	29
-0.10	-0.12783337150988488550851230	-0.12783337150988488550851230	22
-0.05	-0.05657034959618335165162590	-0.05657034959618333080921810	16
0.00	0.00000000298023221934062840	0.00000000298023227873755980	3
0.05	0.04401688883905053445433350	0.04401688883905064547663600	16
0.10	0.07696104444749744810660990	0.07696104444749746198439770	22
0.15	0.09984533766675877941576540	0.09984533766675871002682640	29
0.20	0.11332868690355603347175160	0.11332868690355611673847850	37
0.25	0.11778303565638327532472120	0.11778303565638326144693340	48
0.30	0.11332868317826565174311070	0.11332868317826561010974730	64

Вывод

В ходе работы я написал и протестировал программу на языке Си, которая оценивает точность вычисления функции с помощью ряда Тейлора. Основываясь на результатах, можно сделать вывод, что ряд Тейлора дает точные результаты, но существуют более быстрые способы представления трансцендентных функций.

Исходный код

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

int main(void) {
    double Epsillon = 1.0;
    double Taylor;
    int n;
    int k = 1000;
    double a = -0.2f;
    double b = 0.3f;
    double x;
    int Counter;

    while (1.0 + (Epsillon / 2.0) > 1.0) {
    Epsillon /= 2.0;
    }

    printf("Машинное эпсилон для типа double в системе VS Code = %e\n",
Epsillon);
    printf("Число n:\n");
    scanf("%d", &n);
    printf("Число k:\n");
    scanf("%d", &k);
    printf("Таблица значений ряда Тейлора и стандартной функции для f(x) =
(((-1)^(n + 1) * 2^n - 1)*x^n)/n\n");

printf("_____________________________________________________________________
_____________________\n");
    printf("|   x   |           Тейлор             |             f(x)
| число итераций |\n");

printf("_____________________________________________________________________
_____________________\n");

    x = -0.2f;
    for (int i = 0; i <= n; i++) {
        double function;
        function = log(1 + x - 2 * pow(x,2));
        Taylor = 0;
        Counter = 1;
        double Buffer = 1;
        while (fabs(Buffer) > Epsillon * k && Counter < 100) {
            Buffer = ((pow(-1, Counter + 1) * pow(2, Counter) - 1) *
pow(x, Counter)) / Counter;
            Taylor += Buffer;
```

```
            Counter += 1;
        }
        if (x < 0) {
            if (Taylor < 0) {
                if (Counter >= 10) {
                    printf("| %.2f | %.25f | %.25f |       %d       |\n",
x, Taylor, function, Counter);
                } else {
                   printf("| %.2f | %.25f | %.25f |        %d       |\n",
x, Taylor, function, Counter);
                }
            } else {
                if (Counter >= 10) {
                    if (function > 0) {
                        printf("| %.2f |  %.25f |  %.25f |       %d
|\n", x, Taylor, function, Counter);
                    } else {
                        printf("| %.2f |  %.25f | %.25f |       %d
|\n", x, Taylor, function, Counter);
                    }
                } else {
                    if (function > 0) {
                        printf("| %.2f |  %.25f |  %.25f |        %d
|\n", x, Taylor, function, Counter);
                    } else {
                        printf("| %.2f |  %.25f | %.25f |        %d
|\n", x, Taylor, function, Counter);
                    }
                }
            }
        } else {
            if (Taylor < 0) {
                if (Counter >= 10) {
                    if (function > 0) {
                            printf("|  %.2f | %.25f |  %.25f |       %d
|\n", x, Taylor, function, Counter);
                        } else {
                            printf("|  %.2f | %.25f | %.25f |       %d
|\n", x, Taylor, function, Counter);
                        }
                } else {
                    if (function > 0) {
                            printf("|  %.2f | %.25f |  %.25f |        %d
|\n", x, Taylor, function, Counter);
                        } else {
                            printf("|  %.2f | %.25f | %.25f |        %d
|\n", x, Taylor, function, Counter);
                        }
                }
            } else {
                if (Counter >= 10) {
                    if (function > 0) {
                            printf("|  %.2f |  %.25f |  %.25f |       %d
|\n", x, Taylor, function, Counter);
                        } else {
                            printf("|  %.2f |  %.25f | %.25f |       %d
|\n", x, Taylor, function, Counter);
                        }
                } else {
                    if (function > 0) {
```

```
                            printf("|  %.2f |  %.25f |  %.25f |        %d
|\n", x, Taylor, function, Counter);
                        } else {
                            printf("|  %.2f |  %.25f | %.25f |        %d
|\n", x, Taylor, function, Counter);
                        }
                    }
                }
            }

printf("_____________________________________________________________________________
_____________________\n");
        x += (b - a) / n;
    }
    return 0;
}
```