Московский авиационный институт

(Национальный исследовательский университет)

Факультет прикладной математики и физики

Курсовой проект

по курсу

«Фундаментальная информатика»

I семестр

Задание 3

Студентка: Соломатина С.

Группа: М8О-113Б-21

Руководители: Довженко А.

Оценка:

Дата:

Москва

2021г.

Задание

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, находящихся в рекомендованной области хорошей точности формулы Тейлора. Вычисления по формуле Тейлора проводить по экономной в сложностном смысле схеме с точностью $\varepsilon * k$, 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 8

Функция: $\frac{1}{2x-5}$

Отрезок: $[0{,}0; 2{,}0]$

Ряд: $-\frac{1}{5} - \frac{2x}{5^2} - \frac{4x^2}{5^3} - \ldots - \frac{2^{n-1}x^{n-1}}{5^n}$

Решение

Всё решение сводится к тому, чтобы записать на языке Си две функции и вывести их значения на заданном отрезке. Функция, реализующая вычисление с помощью ряда Тейлора, представляет собой итерационный процесс, в ходе которого последовательно вычисляется сумма членов ряда. Ряд Тейлора для функции $f(x)$ в окрестности точки a выглядит следующим образом:

$$f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n + \cdots$$

Основной вопрос заключается в точности вычислений. Дело в том, что точность каких-либо алгоритмов в ЭВМ ограничена. Отсюда и возникает понятие

«машинного эпсилон». От машинного эпсилон зависит, насколько точно можно посчитать значение функции по ряду Тейлора.

Машинное эпсилон — это максимальная относительная погрешность для конкретной процедуры округления, это такое числовое значение, меньше которого невозможно задавать относительную точность для любого алгоритма, возвращающего вещественные числа. Его можно представить как $1.0 + \varepsilon \neq 1.0$. Чем меньше его значение, тем выше точность вычисления. Практическая важность машинного эпсилон связана с тем, что два числа являются одинаковыми с точки зрения машинной арифметики, если их относительная разность по модулю меньше эпсилон. Необходимо сказать об округлении чисел: правило округления в стандарте IEEE754 говорит о том, что результат любой арифметической операции должен быть таким, как если бы он был выполнен над точными значениями и округлен до ближайшего числа, представимого в этом формате. Округление до ближайшего в стандарте сделано не так как мы привыкли. Математически показано, что если 0,5 округлять до 1 (в большую сторону), то существует набор операций, при которых ошибка округления будет возрастать до бесконечности. Поэтому в IEEE754 применяется правило округления до четного.

Способы вычисления машинного эпсилон:

1. Подключить библиотеку limits.h и использовать константу DBL_EPSILON
2. Делить 1.0 пополам пока не получится так, что мы не можем отличить одно от другого. Если так случилось, значит, разница на предыдущем шаге и есть машинное эпсилон.

   ```
   double eps = 1.0;
   while (1.0 + (eps / 2.0) > 1.0) {
   eps /= 2.0;
   }
   ```
3. Нужно найти число, у которого мантисса «сдвинута» на единицу левее другого числа. Эти числа 7/3 и 4/3:

   7/3 = 10.010101010101…

4/3 = 1.0101010101010…

Запишем числа в формате IEEE-734:

7/3 = $1.00101010101010101010101010101010101010101010101010101011 * 2^1$

4/3 = $1.01010101010101010101010101010101010101010101010101010101 * 2^0$

При вычитании все биты, кроме последнего, зануляются:

7/3 – 4/3 = $1.00000000000000000000000000000000000000000000000000000001 * 2^0$

Получается, 7/3 – 4/3 = 1 + ε

ε = 7/3 – 4/3 – 1

double eps = 7.0 / 3.0 – 4.0 / 3.0 – 1.0;

Выполнение итерации в функции ряда Тейлора включает вычисление значения очередного члена ряда, для получения которого необходим подсчёт значений степенной функции. Можно заметить, что каждый последующий член ряда может быть получен быстрее и с меньшими затратами при учёте имеющихся значений предыдущих элементов. Для данной функции рекуррентная формула имеет вид $\frac{2x}{5}$.

Листинг программного кода

```
#include <stdio.h>

 #define epsilon (7.0 / 3.0 - 4.0 / 3.0 - 1.0)
 #define A 0.0
 #define B 2.0

 double function(double arg)
 {
        return 1 / (2 * arg - 5);
 }

 double function_taylor(double arg)
 {
        double res = - (1.0 / 5), prev = - (1.0 / 5), pres;
        int iter = 0;
```

```
	for (; prev < -epsilon && iter <= 100; ++iter) {
		pres = prev * 2 * arg / 5;
		res += pres;
		prev = pres;
	}
	iter > 100 ? printf("|n = %3d\t|\t", iter - 1) : printf("|n = %3d\t|\t", iter);
	return res;
}

int main(void)
{
	double d = 0.0;
	scanf("%lf", &d);
	d = 1.0 / (d * 2.0);
	printf("|Итерации |\t|Значение x|\t|Значение функции |\t|Зн-ие по ф-ле Тейлора | \n");
	for (double arg = A; arg <= B; arg += d) {
		printf("|%.2lf |\t|%.20lf|\t|%.20lf| \n", arg, function(arg), function_taylor(arg));
	}
	return 0;
}
```

Результат работы программы

ssolomvi@ssolomvi:~$ gcc kp3.c -lm -std=c99 ssolomvi@ssolomvi:~$ ./a.out

8

|Итерации		|Значение x| |Значение функции	|Зн-ие по ф-ле Тейлора
	n = 1		0.00		-0.20000000000000001110		-0.20000000000000001110
	n = 10		0.06		-0.20512820512820512109		-0.20512820512820512109
	n = 12		0.12		-0.21052631578947367252		-0.21052631578947375579
	n = 14		0.19		-0.21621621621621622822		-0.21621621621621620046
	n = 15		0.25		-0.22222222222222220989		-0.22222222222222220989
	n = 17		0.31		-0.22857142857142856429		-0.22857142857142856429
	n = 19		0.38		-0.23529411764705882026		-0.23529411764705884802
	n = 20		0.44		-0.24242424242424243097		-0.24242424242424243097
	n = 22		0.50		-0.25000000000000000000		-0.25000000000000011102
	n = 24		0.56		-0.25806451612903225090		-0.25806451612903230641
	n = 25		0.62		-0.26666666666666666297		-0.26666666666666660745

n = 27		0.69		-0.27586206896551723755		-0.27586206896551723755
n = 29		0.75		-0.28571428571428569843		-0.28571428571428564291
n = 31		0.81		-0.29629629629629627985		-0.29629629629629627985
n = 33		0.88		-0.30769230769230770939		-0.30769230769230754285
n = 36		0.94		-0.32000000000000000666		-0.32000000000000011768
n = 38		1.00		-0.33333333333333331483		-0.33333333333333342585
n = 41		1.06		-0.34782608695652172948		-0.34782608695652167397
n = 44		1.12		-0.36363636363636364646		-0.36363636363636359095
n = 47		1.19		-0.38095238095238093123		-0.38095238095238082021
n = 50		1.25		-0.40000000000000002220		-0.39999999999999985567
n = 54		1.31		-0.42105263157894734505		-0.42105263157894706749
n = 58		1.38		-0.44444444444444441977		-0.44444444444444441977
n = 63		1.44		-0.47058823529411764053		-0.47058823529411741848
n = 68		1.50		-0.50000000000000000000		-0.49999999999999988898
n = 74		1.56		-0.53333333333333332593		-0.53333333333333310389
n = 80		1.62		-0.57142857142857139685		-0.57142857142857117481
n = 88		1.69		-0.61538461538461541878		-0.61538461538461486366
n = 97		1.75		-0.66666666666666662966		-0.66666666666666629659
n = 100		1.81		-0.72727272727272729291		-0.72727272727272285202
n = 100		1.88		-0.80000000000000004441		-0.79999999999985549337
n = 100		1.94		-0.88888888888888883955		-0.88888888888434225422
n = 100		2.00		-1.00000000000000000000		-0.99999999986962939680

Выводы

После генерации таблицы значений заданной функции можно увидеть, что значения совпадают до 14 знака после запятой. Из-за того, что существует понятие ограниченности разрядной сетки, вещественные числа имеют диапазон представления в памяти компьютера, что неизбежно приводит к тому, что в вычислениях в окрестности границ этого диапазона возникают погрешности.

Вычисление значения функции по ряду Тейлора требует много процессорного времени, что неэффективно в перспективе глобального применения.