**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 на равных частей, находящихся в рекомендованной области хорошей точности формулы Тейлора. Вычисления по формуле Тейлора проводить по экономной в сложностном смысле схеме с точностью , где - машинное эпсилон, аппаратно реализованного вещественного типа для данной ЭВМ, а – экспериментально подбираемый коэффициент, обеспечивающий приемлемую сходимость. Число итераций должно ограничиваться сверху числом порядка 100. Программа должна сама определять машинное и обеспечивать корректные размеры генерируемой таблицы.

Вариант 4

Функция:

Отрезок:

Ряд:

**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 в окрестности точки выглядит следующим образом:

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 . 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. Делим 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 = 1.0010101010101010101010101010101010101010101010101011 \*   
   4/3 = 1.0101010101010101010101010101010101010101010101010101 \*   
   При вычитании все биты, кроме последнего, зануляются:  
   7/3 – 4/3 = 1.0000000000000000000000000000000000000000000000000001 \*   
   Получается, 7/3 – 4/3 = 1 +   
    = 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 подсчет значений степенной функции. Можно заметить, что каждый последующий член ряда может быть получен быстрее и с меньшими затратами при учете имеющихся значений предыдущих элементов. Для данной функции рекуррентная формула имеет вид

**Листинг программного кода**

#include <stdio.h>

#include <math.h>

#include <float.h>

double function(double arg)

{

return log(2 + arg);

}

double function\_taylor(double arg)

{

double res = log(2) + (arg / 2), prev = (arg / 2), pres = 0.0;

int iter = 1;

for (iter; fabs(prev) > DBL\_EPSILON && iter <= 100; ++iter) {

pres = prev \* (((-1) \* arg \* iter) / ((iter + 1) \* 2));

res += pres;

prev = pres;

}

printf("|n = %d |", iter);

return res;

}

int main(void)

{

#define A -1.0

#define B 1.0

double d = 0.0;

scanf("%lf", &d);

d = (B - A) / (d - 1);

printf("| Итерации || 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**

raccoonrocket@raccoonrocket-VirtualBox:~$ gcc kp3.c

raccoonrocket@raccoonrocket-VirtualBox:~$ ./a.out

15

| Итерации || Значение x | | Значение функции | | Зн-ие по ф-ле Тейлора |

|n = 47 ||-1.00| |0.00000000000000000000| |0.00000000000000012327|

|n = 39 ||-0.86| |0.13353139262452257130| |0.13353139262452259906|

|n = 32 ||-0.71| |0.25131442828090599928| |0.25131442828090594377|

|n = 27 ||-0.57| |0.35667494393873222513| |0.35667494393873228065|

|n = 22 ||-0.43| |0.45198512374305710448| |0.45198512374305710448|

|n = 18 ||-0.29| |0.53899650073268678963| |0.53899650073268701167|

|n = 13 ||-0.14| |0.61903920840622339572| |0.61903920840622328470|

|n = 1 ||-0.00| |0.69314718055994517520| |0.69314718055994517520|

|n = 13 ||0.14| |0.76214005204689672102| |0.76214005204689660999|

|n = 18 ||0.29| |0.82667857318446791304| |0.82667857318446780202|

|n = 22 ||0.43| |0.88730319500090271134| |0.88730319500090237828|

|n = 27 ||0.57| |0.94446160884085128551| |0.94446160884085150755|

|n = 32 ||0.71| |0.99852883011112703038| |0.99852883011112680833|

|n = 39 ||0.86| |1.04982212449867762238| |1.04982212449867806647|

|n = 47 ||1.00| |1.09861228866810956006| |1.09861228866811000415| **Выводы**

После генерации таблицы значений заданной функции можно увидеть, что значения совпадают до 13 знака включительно 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.

На базе этой программы можно создать много аналогичных программ, выполняющих те же действия с другими функциями, но они не имеют прикладного применения, так как вычисление значения функции по ряду Тейлора требует много процессорного времени, что неэффективно в перспективе глобального применения.