МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Московский авиационный институт

(национальный исследовательский университет)

Факультет: «Информационные технологии и прикладная математика»

Кафедра 806: «Вычислительная математика и программирование»

Курсовой проект

по курсу «Основы информатики»

I семестр

Задание 3. «Вещественный тип. Приближенные вычисления. Табулирование функций.»

|  |  |
| --- | --- |
| Группа: | М8о-107б-18 |
| Студент: | Тояков Артем |
| Преподаватель: | Ридли Александра Николаевна |
| Оценка: |  |
| Дата: |  |

# СОДЕРЖАНИЕ

[ЗАДАНИЕ 3](#_Toc536052729)

[РЕШЕНИЕ 3](#_Toc536052730)

[ОПИСАНИЕ ПЕРЕМЕННЫХ 5](#_Toc536052731)

[ЛИСТИНГ ПРОГРАММНОГО КОДА 6](#_Toc536052732)

[РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ 7](#_Toc536052733)

[ВЫВОД 7](#_Toc536052734)

# ЗАДАНИЕ

Составить программу на языке 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 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 № 15

Функция: cos x

Отрезок: [0.0; 1.0]

Ряд:

# РЕШЕНИЕ

Всё решение сводится к тому, чтобы записать на языке Си две функции и вывести их значения в заданной точке. 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:

-((arg \* arg) / ((2 \* i - 1) \* (2 \* i)))

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. Например, 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;

# ОПИСАНИЕ ПЕРЕМЕННЫХ

|  |  |
| --- | --- |
| **Имя** | **Назначение** |
| numb | Количество отрезков |
| A, B | Постоянные границы интервала |
| arg | Аргументы функции |
| n | Число итераций |
| interval | Шаг аргументов |
| x | Член ряда Тейлора |
| sum\_Teylor | Сумма ряда Тейлора |
| eps | Машинное эпсилон |

# ЛИСТИНГ ПРОГРАММНОГО КОДА

#include <stdio.h>

#include <math.h>

#define A 0.0

#define B 1.0

double epsilon() //определение машинного eps

{

double eps = 1;

while ((1 + eps) != 1) {

eps /= 2;

}

return eps;

}

int main(void)

{

int numb, n = 0, i;

double sum\_Teylor = 1, x = 1, arg, eps = epsilon();

scanf("%d", &numb); //ввод кол-ва делений отрезка

double interval = (B - A) / numb;

printf("Количество итераций\t| Значение x\t| Значение функции\t| Значение по Тейлору \t | \n");

printf("------------------------------------------------------------------------------------------\n");

for (arg = A; arg <= B; arg += interval) {

sum\_Teylor = 1;

x = 1;

for (i = 1; i <= 100; i++) {

x \*= -((arg \* arg) / ((2 \* i - 1) \* (2 \* i)));

if (eps > fabs(x)) {

break;

} else {

sum\_Teylor += x;

n++;

}

}

printf(" n = %d \t\t\t| %.2lf \t\t| %.20lf \t| %.20lf \t | \n", n, arg, cos(arg), sum\_Teylor);

n = 0;

}

return 0;

}

# РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

artoy@artem-trumpeter:~$ gcc -Wall -pedantic -std=c99 kp3.c -lm

artoy@artem-trumpeter:~$ ./a.out

12

Количество итераций | Значение x | Значение функции | Значение по Тейлору |

------------------------------------------------------------------------------------------------------------------------

n = 0 | 0.00 | 1.00000000000000000000 | 1.00000000000000000000 |

n = 4 | 0.08 | 0.99652978670055947230 | 0.99652978670055958332 |

n = 5 | 0.17 | 0.98614323156292504891 | 0.98614323156292504891 |

n = 6 | 0.25 | 0.96891242171064473343 | 0.96891242171064473343 |

n = 6 | 0.33 | 0.94495694631473770020 | 0.94495694631473770020 |

n = 6 | 0.42 | 0.91444306659383034486 | 0.91444306659383034486 |

n = 7 | 0.50 | 0.87758256189037275874 | 0.87758256189037275874 |

n = 7 | 0.58 | 0.83463125983165697974 | 0.83463125983165686872 |

n = 7 | 0.67 | 0.78588726077694803784 | 0.78588726077694792682 |

n = 8 | 0.75 | 0.73168886887382089679 | 0.73168886887382089679 |

n = 8 | 0.83 | 0.67241224408305666493 | 0.67241224408305644289 |

n = 8 | 0.92 | 0.60846879146804511151 | 0.60846879146804511151 |

n = 9 | 1.00 | 0.54030230586813976501 | 0.54030230586813965399 |

# ВЫВОД

После генерации таблицы значений заданной функции можно увидеть, что значения совпадают до 13-14 знака после запятой. Из-за того, что вещественные числа имеют диапазон представления в памяти компьютера, в вычислениях в окрестности границ этого диапазона возникают погрешности. На базе этой программы можно создать множество 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.