Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовая работа**

**по курсу “Вычислительные системы”**

**I семестр**

**Задание 3**

**«Вещественный тип. Приближенные вычисления. Табулирование функций»**

|  |  |
| --- | --- |
| **Студент:** | Аксенов А. Е. |
| **Группа:** | 8О-108Б, №9 |
| **Преподаватель:** | Поповкин А. В. |
|  |  |
| **Оценка:** |  |
| **Дата:** |  |

# Содержание

1. Задание……………………………………………………………………….…....3
2. Решение……………………………….……………………………………….......3
3. Способы вычисления машинного эпсилон…………..……………………….…4
4. Краткий алгоритм программы…………….……………………………………..5
5. Код программы……….…….……………………………………………………..5
6. Результат работы программы….…………………………………………………6
7. Заключение………………………………………………………………………12
8. Список литературы……………………………………………………………...12

## **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 (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 1

Функция:

Отрезок: [-1,0 ; 1,0]

Ряд:

**Решение**

Всё решение заключается в том, 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 (цикл while). Ряд Тейлора для функции 𝑓(𝑥) в окрестности точки 𝑎 выглядит следующим образом:

Главный вопрос заключается именно в точности вычислений. Дело в том, что точность каких-либо алгоритмов в ЭВМ ограничена. Отсюда появляется такое понятие, как «машинное эпсилон». От него зависит то, насколько точно можно посчитать значение функции по ряду Тейлора.

Машинное эпсилон — это наибольшая 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 + ε ≠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 math.h и использовать константу 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 Epsilon(void)

{

double epsilon = 1.0;

while ((1.0 + (epsilon / 2.0)) > 1.0) {

epsilon /= 2.0;

}

return epsilon;

}

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.

**Краткий алгоритм программы**

1. Создаем функции, которые высчитывают машинное эпсилон, и функцию, находящую модуль числа (хотя, есть fabs).
2. Считаем эпсилон, выводим основные данные (шаг, эпсилон…)
3. Выводим названия столбцов таблицы
4. Делаем цикл, и для каждой точки считаем значение функции по формуле (в одну строчку), и с помощью цикла while() – высчитываем по формуле Тейлора.
5. В конце каждого такого цикла выводим значения и количество итераций.

**Код программы**

#include<stdio.h>

#include<math.h>

double Epsilon(void) {

double epsilon = 1.0;

while ((1.0 + (epsilon / 2.0)) > 1.0) {

epsilon /= 2.0;

}

return epsilon;

}

double fabs(double a) {

if (a<0) {

a \*= -1;

}

return a;

}

int main() {

double n,k;

scanf("%lf",&n);

scanf("%lf",&k);

printf("\n");

double eps;

eps = Epsilon();

printf("Epsilon = %.16lf \n \n", eps);

printf("Step = %.5lf \n \n", (2/n));

printf("Point X \t Taylor \t Function \t Iterations \n");

double x,taylor,function,iters;

for (double i=0 ; i<n+1 ; ++i) {

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

iters = 1;

taylor = 0;

function = (x/(9+x\*x));

while ((fabs(function-taylor) >= (k\*eps)) && (iters < 100)) {

iters += 1;

taylor -= pow(-1,iters-1)\*(pow(x,(iters-1)\*2-1)/pow(9,iters-1));

}

printf("%0.5lf \t %0.10lf \t %0.10lf \t %0.0lf \n",x,taylor,function,iters);

}

return 0;

}

**Результат работы программы**

fallfire@vb:~/kursach3$ ./kurs

200 10

Epsilon = 0.0000000000000002

Step = 0.01000

Point X Taylor Function Iterations

-1.00000 -0.1000000000 -0.1000000000 16

-0.99000 -0.0991974028 -0.0991974028 16

-0.98000 -0.0983896229 -0.0983896229 16

-0.97000 -0.0975766782 -0.0975766782 15

-0.96000 -0.0967585873 -0.0967585873 15

-0.95000 -0.0959353699 -0.0959353699 15

-0.94000 -0.0951070460 -0.0951070460 15

-0.93000 -0.0942736368 -0.0942736368 15

-0.92000 -0.0934351641 -0.0934351641 15

-0.91000 -0.0925916505 -0.0925916505 15

-0.90000 -0.0917431193 -0.0917431193 15

-0.89000 -0.0908895947 -0.0908895947 14

-0.88000 -0.0900311017 -0.0900311017 14

-0.87000 -0.0891676660 -0.0891676660 14

-0.86000 -0.0882993141 -0.0882993141 14

-0.85000 -0.0874260735 -0.0874260735 14

-0.84000 -0.0865479723 -0.0865479723 14

-0.83000 -0.0856650394 -0.0856650394 14

-0.82000 -0.0847773045 -0.0847773045 14

-0.81000 -0.0838847982 -0.0838847982 13

-0.80000 -0.0829875519 -0.0829875519 13

-0.79000 -0.0820855976 -0.0820855976 13

-0.78000 -0.0811789684 -0.0811789684 13

-0.77000 -0.0802676980 -0.0802676980 13

-0.76000 -0.0793518209 -0.0793518209 13

-0.75000 -0.0784313725 -0.0784313725 13

-0.74000 -0.0775063890 -0.0775063890 13

-0.73000 -0.0765769073 -0.0765769073 13

-0.72000 -0.0756429652 -0.0756429652 12

-0.71000 -0.0747046012 -0.0747046012 12

-0.70000 -0.0737618546 -0.0737618546 12

-0.69000 -0.0728147656 -0.0728147656 12

-0.68000 -0.0718633750 -0.0718633750 12

-0.67000 -0.0709077247 -0.0709077247 12

-0.66000 -0.0699478571 -0.0699478571 12

-0.65000 -0.0689838153 -0.0689838153 12

-0.64000 -0.0680156436 -0.0680156436 12

-0.63000 -0.0670433866 -0.0670433866 11

-0.62000 -0.0660670901 -0.0660670901 11

-0.61000 -0.0650868002 -0.0650868002 11

-0.60000 -0.0641025641 -0.0641025641 11

-0.59000 -0.0631144297 -0.0631144297 11

-0.58000 -0.0621224455 -0.0621224455 11

-0.57000 -0.0611266609 -0.0611266609 11

-0.56000 -0.0601271259 -0.0601271259 11

-0.55000 -0.0591238914 -0.0591238914 11

-0.54000 -0.0581170089 -0.0581170089 11

-0.53000 -0.0571065306 -0.0571065306 10

-0.52000 -0.0560925095 -0.0560925095 10

-0.51000 -0.0550749992 -0.0550749992 10

-0.50000 -0.0540540541 -0.0540540541 10

-0.49000 -0.0530297291 -0.0530297291 10

-0.48000 -0.0520020801 -0.0520020801 10

-0.47000 -0.0509711633 -0.0509711633 10

-0.46000 -0.0499370359 -0.0499370359 10

-0.45000 -0.0488997555 -0.0488997555 10

-0.44000 -0.0478593804 -0.0478593804 9

-0.43000 -0.0468159697 -0.0468159697 9

-0.42000 -0.0457695828 -0.0457695828 9

-0.41000 -0.0447202801 -0.0447202801 9

-0.40000 -0.0436681223 -0.0436681223 9

-0.39000 -0.0426131707 -0.0426131707 9

-0.38000 -0.0415554875 -0.0415554875 9

-0.37000 -0.0404951351 -0.0404951351 9

-0.36000 -0.0394321767 -0.0394321767 9

-0.35000 -0.0383666758 -0.0383666758 9

-0.34000 -0.0372986967 -0.0372986967 8

-0.33000 -0.0362283042 -0.0362283042 8

-0.32000 -0.0351555634 -0.0351555634 8

-0.31000 -0.0340805400 -0.0340805400 8

-0.30000 -0.0330033003 -0.0330033003 8

-0.29000 -0.0319239110 -0.0319239110 8

-0.28000 -0.0308424392 -0.0308424392 8

-0.27000 -0.0297589525 -0.0297589525 8

-0.26000 -0.0286735189 -0.0286735189 8

-0.25000 -0.0275862069 -0.0275862069 8

-0.24000 -0.0264970853 -0.0264970853 7

-0.23000 -0.0254062234 -0.0254062234 7

-0.22000 -0.0243136908 -0.0243136908 7

-0.21000 -0.0232195575 -0.0232195575 7

-0.20000 -0.0221238938 -0.0221238938 7

-0.19000 -0.0210267704 -0.0210267704 7

-0.18000 -0.0199282583 -0.0199282583 7

-0.17000 -0.0188284287 -0.0188284287 7

-0.16000 -0.0177273533 -0.0177273533 7

-0.15000 -0.0166251039 -0.0166251039 6

-0.14000 -0.0155217526 -0.0155217526 6

-0.13000 -0.0144173718 -0.0144173718 6

-0.12000 -0.0133120341 -0.0133120341 6

-0.11000 -0.0122058122 -0.0122058122 6

-0.10000 -0.0110987791 -0.0110987791 6

-0.09000 -0.0099910081 -0.0099910081 6

-0.08000 -0.0088825724 -0.0088825724 6

-0.07000 -0.0077735455 -0.0077735455 5

-0.06000 -0.0066640011 -0.0066640011 5

-0.05000 -0.0055540128 -0.0055540128 5

-0.04000 -0.0044436545 -0.0044436545 5

-0.03000 -0.0033330000 -0.0033330000 5

-0.02000 -0.0022221235 -0.0022221235 4

-0.01000 -0.0011110988 -0.0011110988 4

0.00000 0.0000000000 0.0000000000 1

0.01000 0.0011110988 0.0011110988 4

0.02000 0.0022221235 0.0022221235 4

0.03000 0.0033330000 0.0033330000 5

0.04000 0.0044436545 0.0044436545 5

0.05000 0.0055540128 0.0055540128 5

0.06000 0.0066640011 0.0066640011 5

0.07000 0.0077735455 0.0077735455 5

0.08000 0.0088825724 0.0088825724 6

0.09000 0.0099910081 0.0099910081 6

0.10000 0.0110987791 0.0110987791 6

0.11000 0.0122058122 0.0122058122 6

0.12000 0.0133120341 0.0133120341 6

0.13000 0.0144173718 0.0144173718 6

0.14000 0.0155217526 0.0155217526 6

0.15000 0.0166251039 0.0166251039 6

0.16000 0.0177273533 0.0177273533 7

0.17000 0.0188284287 0.0188284287 7

0.18000 0.0199282583 0.0199282583 7

0.19000 0.0210267704 0.0210267704 7

0.20000 0.0221238938 0.0221238938 7

0.21000 0.0232195575 0.0232195575 7

0.22000 0.0243136908 0.0243136908 7

0.23000 0.0254062234 0.0254062234 7

0.24000 0.0264970853 0.0264970853 7

0.25000 0.0275862069 0.0275862069 8

0.26000 0.0286735189 0.0286735189 8

0.27000 0.0297589525 0.0297589525 8

0.28000 0.0308424392 0.0308424392 8

0.29000 0.0319239110 0.0319239110 8

0.30000 0.0330033003 0.0330033003 8

0.31000 0.0340805400 0.0340805400 8

0.32000 0.0351555634 0.0351555634 8

0.33000 0.0362283042 0.0362283042 8

0.34000 0.0372986967 0.0372986967 8

0.35000 0.0383666758 0.0383666758 9

0.36000 0.0394321767 0.0394321767 9

0.37000 0.0404951351 0.0404951351 9

0.38000 0.0415554875 0.0415554875 9

0.39000 0.0426131707 0.0426131707 9

0.40000 0.0436681223 0.0436681223 9

0.41000 0.0447202801 0.0447202801 9

0.42000 0.0457695828 0.0457695828 9

0.43000 0.0468159697 0.0468159697 9

0.44000 0.0478593804 0.0478593804 9

0.45000 0.0488997555 0.0488997555 10

0.46000 0.0499370359 0.0499370359 10

0.47000 0.0509711633 0.0509711633 10

0.48000 0.0520020801 0.0520020801 10

0.49000 0.0530297291 0.0530297291 10

0.50000 0.0540540541 0.0540540541 10

0.51000 0.0550749992 0.0550749992 10

0.52000 0.0560925095 0.0560925095 10

0.53000 0.0571065306 0.0571065306 10

0.54000 0.0581170089 0.0581170089 11

0.55000 0.0591238914 0.0591238914 11

0.56000 0.0601271259 0.0601271259 11

0.57000 0.0611266609 0.0611266609 11

0.58000 0.0621224455 0.0621224455 11

0.59000 0.0631144297 0.0631144297 11

0.60000 0.0641025641 0.0641025641 11

0.61000 0.0650868002 0.0650868002 11

0.62000 0.0660670901 0.0660670901 11

0.63000 0.0670433866 0.0670433866 11

0.64000 0.0680156436 0.0680156436 12

0.65000 0.0689838153 0.0689838153 12

0.66000 0.0699478571 0.0699478571 12

0.67000 0.0709077247 0.0709077247 12

0.68000 0.0718633750 0.0718633750 12

0.69000 0.0728147656 0.0728147656 12

0.70000 0.0737618546 0.0737618546 12

0.71000 0.0747046012 0.0747046012 12

0.72000 0.0756429652 0.0756429652 12

0.73000 0.0765769073 0.0765769073 13

0.74000 0.0775063890 0.0775063890 13

0.75000 0.0784313725 0.0784313725 13

0.76000 0.0793518209 0.0793518209 13

0.77000 0.0802676980 0.0802676980 13

0.78000 0.0811789684 0.0811789684 13

0.79000 0.0820855976 0.0820855976 13

0.80000 0.0829875519 0.0829875519 13

0.81000 0.0838847982 0.0838847982 13

0.82000 0.0847773045 0.0847773045 14

0.83000 0.0856650394 0.0856650394 14

0.84000 0.0865479723 0.0865479723 14

0.85000 0.0874260735 0.0874260735 14

0.86000 0.0882993141 0.0882993141 14

0.87000 0.0891676660 0.0891676660 14

0.88000 0.0900311017 0.0900311017 14

0.89000 0.0908895947 0.0908895947 14

0.90000 0.0917431193 0.0917431193 15

0.91000 0.0925916505 0.0925916505 15

0.92000 0.0934351641 0.0934351641 15

0.93000 0.0942736368 0.0942736368 15

0.94000 0.0951070460 0.0951070460 15

0.95000 0.0959353699 0.0959353699 15

0.96000 0.0967585873 0.0967585873 15

0.97000 0.0975766782 0.0975766782 15

0.98000 0.0983896229 0.0983896229 16

0.99000 0.0991974028 0.0991974028 16

1.00000 0.1000000000 0.1000000000 16

**Заключение**

После успешного выполнения программы, в таблице значений заданной функции видно, что значения практически совпадают (но примерно на 14-15 знаке уже различаются). Так как, 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 основе данной программы можно составить множество аналогичных программ, выполняющих те же действиями, но с другими функциями, однако они не имеют прикладного применения, ведь вычисление значения функции по ряду Тейлора требует много времени, а это неэффективно.

**Список литературы**

1. Ряд Тейлора. [Электронный ресурс].

URL: <https://www.calc.ru/Rvad-Teylora-Ryady-Maklorena.html>

1. Керниган, Б. Язык программирования Си: учебное пособие/ Б. Керниган, Д. Ритчи – Москва: Вильямс, 2016.
2. Подбельский, В. В. Программирование на языке Си. / В. В. Подбельский, С. С. Фомин. 2-е дополненное издание. Учебное издание. (Москва: «Финансы и статистика», 2014)