МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский Авиационный Институт»
(Национальный Исследовательский Университет)

Институт: №8 «Компьютерные науки и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Курсовая работа по курсу
«Фундаментальная информатика»
I семестр
Задание 3
«Вещественный тип. Приближённые вычисления. Табулирование функций»

Группа	М8О-109Б-22
Студент	Федоров А. А.
Преподаватель	Сысоев М. А.
Оценка	
Дата	

Москва, 2022

Оглавление

Введение

Компьютер способен обрабатывать информацию гораздо быстрее, чем мозг человека, поэтому он применяется во многих сферах нашей жизни, особенно там, где нужны быстрые и точные вычисления а также работа с большими наборами данных. Но память в компьютере ограничена, а значит, ограничен диапазон представляемых чисел. Особенно это касается точности вещественных чисел, которые часто используются в таких точных науках как, например, математика или физика.

Цель курсовой работы – ознакомиться со стандартом IEEE 754, описывающий представление чисел с плавающей запятой в ЭВМ, а также провести сравнение значений заданной функции двумя способами: с помощью встроенных функций языка программирования Си и по формуле Тейлора.

IEEE 754

Данный стандарт разработан ассоциацией IEEE (Institute of Electrical and Electronics Engineers) и используется для представления действительных чисел (чисел с плавающей точкой) в двоичном коде. Наиболее используемый стандарт для вычислений с плавающей точкой, используется многими микропроцессорами и логическими устройствами, а также программными средствами.

Полное название стандарта в ассоциации IEEE:

IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985)

В 2008 года ассоциация IEEE выпустила стандарт IEEE 754-2008, который включил в себя стандарт IEEE 754-1985.

Стандарт IEEE 754-1985 определяет:

- как представлять нормализованные положительные и отрицательные числа с плавающей точкой
- как представлять денормализованные положительные и отрицательные числа с плавающей точкой
- как представлять нулевые числа
- как представлять специальную величину "бесконечность" (Infinity)
- как представлять специальную величину "Не число" (NaN или NaNs)
- четыре режима округления

IEEE 754-1985 определяет четыре формата представления чисел с плавающей запятой:

- с одинарной точностью (single-precision) 32 бита
- с двойной точностью (double-precision) 64 бита
- с одинарной расширенной точностью (single-extended precision) >= 43 бит (редко используемый)
- с двойной расширенной точностью (double-extended precision) >= 79 бит (обычно используют 80 бит)

ОСНОВНЫЕ ПОНЯТИЯ В ПРЕДСТАВЛЕНИИ ЧИСЕЛ С ПЛАВАЮЩЕЙ ЗАПЯТОЙ

Представление чисел в памяти компьютера основывается на *нормализованном экспоненциальном виде*. Например, десятичное число

155,625 имеет следующую запись в нормализованном экспоненциальном виде: $1{,}55625 \cdot 10^{+2} = 1{,}55625 \cdot exp_{10}^{+2}$. Это же число в двоичной системе счисления $10011011{,}101_2$ будет иметь следующий вид: $1{,}55625 \cdot exp_{10}^{+2} = 1{,}0011011101 \cdot exp_2^{+111}$.

Число $1{,}55625 \cdot exp_{10}^{+2}$ состоит из двух частей: мантиссы M=1.55625 и экспоненты exp_{10}=+2. Экспонента представлена основанием системы исчисления (в данном случае 10) и порядком (в данном случае +2). Порядок экспоненты может иметь отрицательное значение, например число $0{,}0155625 = 1{,}55625 \cdot exp_{10}^{-2}$.

Если мантисса $\geq$ 1 и меньше основания системы счисления, то число считается **нормализованным**.

ОПИСАНИЕ ПРЕОБРАЗОВАНИЯ ЧИСЕЛ

Основное применение в технике и программирование получили форматы 32 и 64 бита. Например, в Си используют типы данных float (32 бита) и double (64 бита).

Описание преобразования в 32-х битный формат IEEE 754:

1. Число может быть положительное или отрицательное. Поэтому отводится 1 бит для обозначения знака числа:
   0 - положительное
   1 - отрицательное
   Это самый старший бит в 32-х битной последовательности.
2. Далее пойдут биты экспоненты, для этого выделяют 1 байт (8 бит). Экспонента может быть, как и число, со знаком + или -. Для определения знака экспоненты, чтобы не вводить ещё один бит знака, добавляют смещение к экспоненте в половину байта +127(0111 1111). То есть, если наша экспоната = +7 (+111 в двоичной), то смещенная экспонента = 7+127=134. А если бы наша экспонента была -7, то смещенная экспонента была равна 127-7 =120. Смещенную экспоненту записывают в отведенные 8 бит.
3. Оставшиеся 23 бита отводят для мантиссы. Но у нормализованной двоичной мантиссы первый бит всегда равен 1, так как число лежит в диапазоне 1 <= M < 2. Нет смысла записывать эту единицу, поэтому в отведенные 23 бита записывают остаток от мантиссы.

В следующей таблице представлено число $155{,}625_{10} = 1{,}0011011101 \cdot exp_2^{+111}$ в 32-х битном формате IEEE754:

Таблица 1. Представление числа $155{,}625_{10}$ в 32-х битном формате

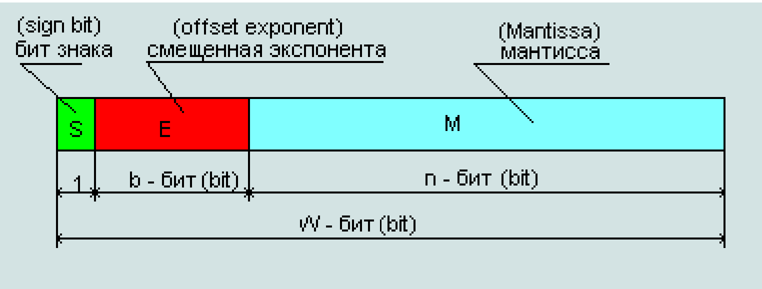
1 бит	8 бит	23 бит	IEEE 754
0	1000 0110	001 1011 1010 0000 0000 0000	431BA000 (hex)
0 (dec)	134 (dec)	1810432 (dec)	
знак числа	смещённая экспонента	остаток от мантиссы	число 155,625 в формате IEEE754

Для остальных форматов точности преобразование аналогичное.

ОБЩЕЕ ПРЕДСТАВЛЕНИЕ НОРМАЛИЗОВАННЫХ ЧИСЕЛ

Формальное представление нормализованных чисел в стандарте IEEE 754 для любого формата точности представлено на следующем рисунке:

Рис. 1 Представление числа в формате IEEE 754



где:

- S - бит знака, если S=0 - положительное число; S=1 - отрицательное число
- E - смещенная экспонента двоичного числа, $exp_2 = E - (2^{(b-1)} - 1)$ - экспонента двоичного нормализованного числа с плавающей точкой, $(2^{(b-1)} - 1)$ - заданное смещение экспоненты
- M - остаток мантиссы двоичного нормализованного числа с плавающей точкой

Формула вычисления десятичных чисел с плавающей точкой из чисел, представленных в стандарте IEEE754:

$$F = (-1)^S \; 2^{(E-2^{(b-1)}+1)} (1+M/2^n)$$ (Формула №1)

Используя формулу №1 вычислим формулы для нахождения десятичных чисел из форматов одинарной (32 бита) и двойной (64 бита) точности IEEE 754:

Рис.2 Формат числа одинарной точности (single-precision) 32 бита

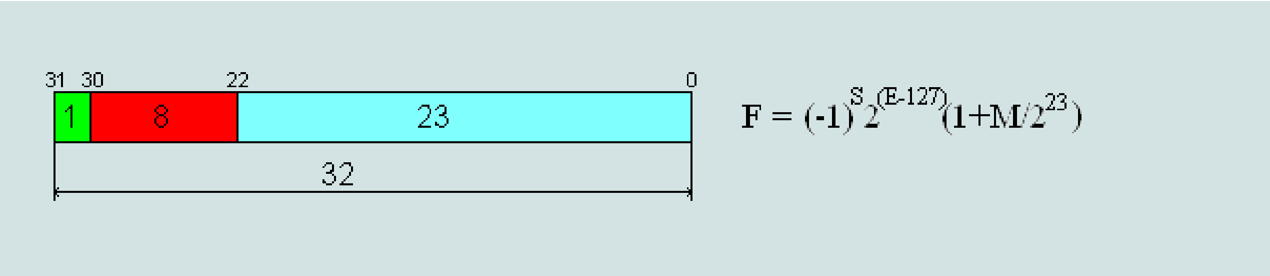
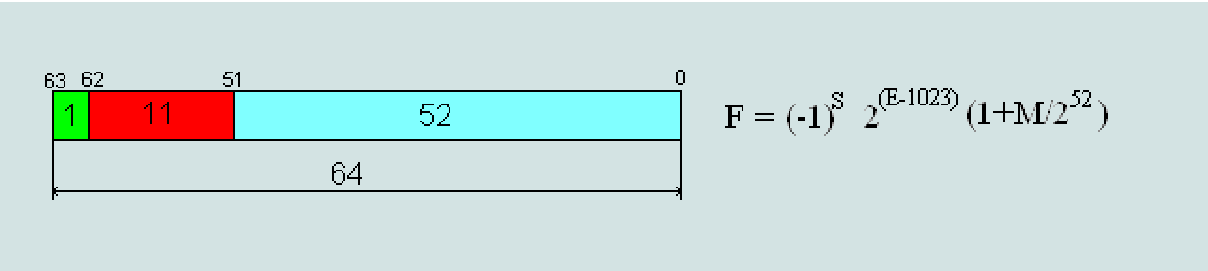


Рис.3 Формат числа двойной точности (double-precision) 64 бит



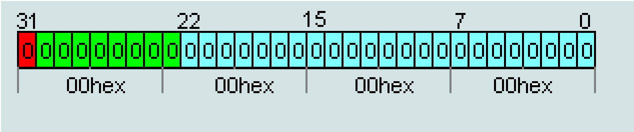
ПРЕДСТАВЛЕНИЕ ДЕНОРМАЛИЗОВАННОГО ЧИСЛА И ДРУГИХ ЧИСЕЛ

Если применить формулу №1 для вычисления минимального и максимального числа одинарной точности представленного в IEEE754, то получим следующие результаты:

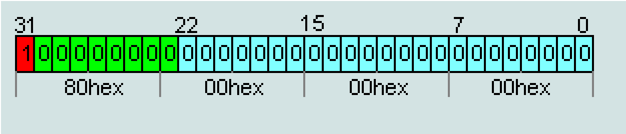
- 00 00 00 00 (hex) = 5,87747175411144e-39 (минимальное положительное число)
- 80 00 00 00 (hex) =-5,87747175411144e-39 (минимальное отрицательное число)
- 7f ff ff ff (hex) = 6,80564693277058e+38 (максимальное положительное число)
- ff ff ff ff (hex) =-6,80564693277058e+38 (максимальное отрицательное число)

Поэтому формула №1 не применяется в следующих случаях:

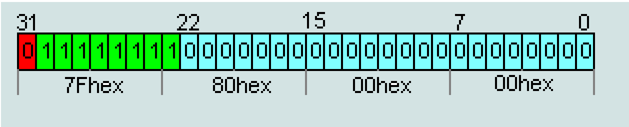
1. Число 00 00 00 00 (hex) считается числом +0



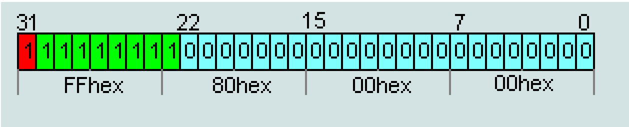
Число 80 00 00 00 (hex) считается числом -0



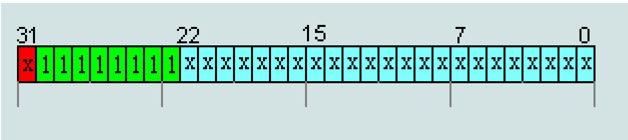
2. Число 7F 80 00 00 (hex) считается числом +∞



число FF 80 00 00 (hex) считается числом -∞

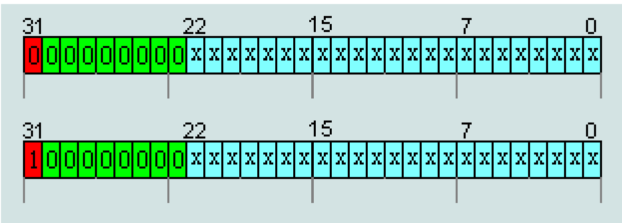


3. Числа FF (1xxx)X XX XX (hex) и 7F (1xxx)X XX XX (hex) не считается числами (not a number, NaN), кроме случая п.2
Число представленное в битах с 0...22 могут быть любым числом кроме 0 (т.е.+∞ и -∞).



NaN может получиться в результате операций с неопределённым результатом (например, 0/0). По определению NaN ≠ NaN.

4. Числа (x000) (0000) (0xxx)X XX XX (hex) считаются денормализованными числами, за исключением чисел п.1 (то есть - 0 и +0)



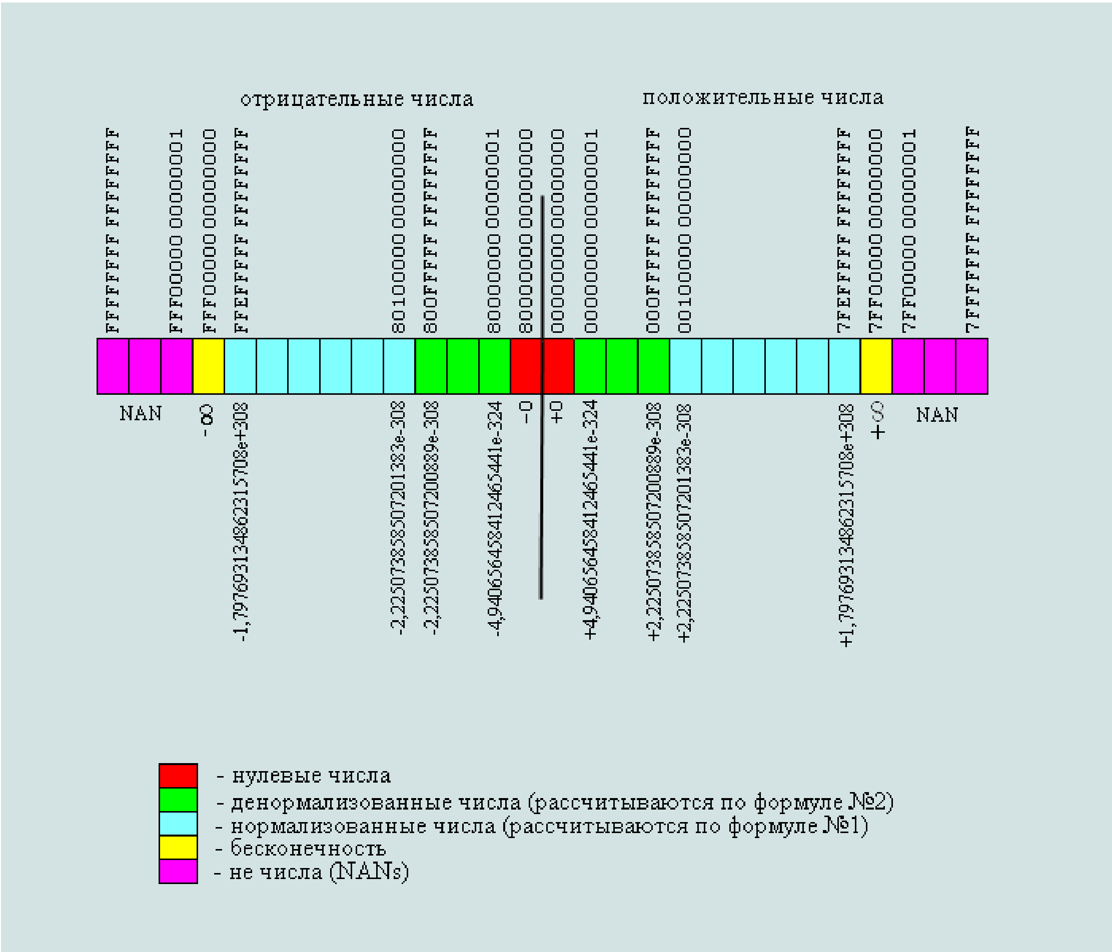
Денормализованные числа – это способ увеличить количество представимых числом значений около нуля для повышения точности вычисления.

Формула расчета денормализованных чисел:

$$F = (-1)^{S}\ 2^{(E-2^{(b-1)}+2)}\ M/2^{n}$$ (Формула №2)

Полный диапазон чисел двойной точности представлен на следующем рисунке:

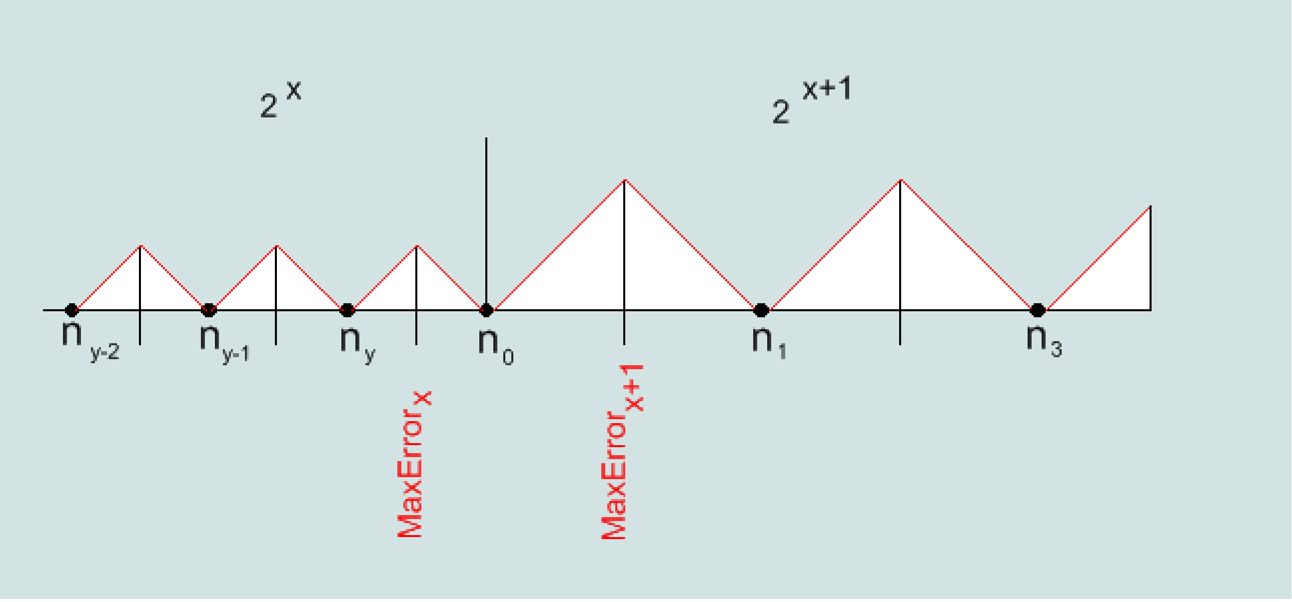
Рис.4 Диапазон чисел формата двойной точности (64 бита) представленных по стандарту IEEE 754



ПОГРЕШНОСТЬ И ОКРУГЛЕНИЕ ЧИСЕЛ

Числа представленные в формате IEEE754 представляют конечное множество, на которое отображается бесконечное множество вещественных чисел. Поэтому исходное число может быть представлено в формате IEEE754 с ошибкой (погрешностью).

Рис.5 Функция ошибки точности представления числа в IEEE754



Абсолютная максимальная ошибка для числа в формате IEEE754 равна в пределе половине шага чисел. Шаг чисел удваивается с увеличением экспоненты двоичного числа на единицу. То есть, чем дальше от нуля, тем шире шаг чисел в формате IEEE754 по числовой оси.

Шаг нормализованных чисел равен $2^{(E-150)}$ (Single) и $2^{(E-1075)}$ (Double). Соответственно предел максимальной абсолютной ошибки будет равен 1/2 шага числа: $2^{(E-151)}$ (Single) и $2^{(E-1076)}$ (Double). Относительная ошибка в % будет равна: $(2^{(E-151)}/F)*100\%$ (Single) и $(2^{(E-1076)}/F)*100\%$ (Double).

Максимальная относительная ошибка нормализованного числа(single):

$$\frac{2^{(E-151)}}{2^{(E-127)}\left(1+\frac{M}{2^{23}}\right)}=\frac{1}{2^{24}+2M}$$

Максимальная относительная ошибка нормализованного числа(double):

$$\frac{2^{(E-1076)}}{2^{(E-1023)}\left(1+\frac{M}{2^{52}}\right)}=\frac{1}{2^{53}+2M}$$

Таблица 2. Максимальная возможная ошибка для чисел Double

IEEE754, hex	число, dec	абсолютная ошибка, dec	относительная, %
00000000 00000001	2^{-1074} ≈4,940656e-324	2^{-1075}≈2,470328e-324	=50
00000000 00000002	2^{-1073} ≈9,881313e-324	2^{-1075}≈2,470328e-324	=25
00000000 00000032	≈2,470328e-322	2^{-1075}≈2,470328e-324	=1
000FFFFF FFFFFFFF	≈2,225073e-308	2^{-1075}≈2,470328e-324	≈1,110223e-14
00100000 00000001	≈2,225074e-308	2^{-1074} ≈4,940656e-324	≈2,220446e-14
2B2BFF2E E48E0530	≈1,0e-100	2^{-385} ≈1,268971e-116	≈1,268971e-14
3FF00000 00000000	=1,0	2^{-52} ≈2,220446e-16	≈2,220446e-14
54B249AD 2594C37D	≈1,0e+100	2^{280} ≈1,942669e+84	≈1,942669e-14
6974E718 D7D7625A	≈1,0e+200	2^{612} ≈1,699641e+184	≈1,699641e-14
7FEFFFFF FFFFFFFF	≈1,79769e+308	2^{971} ≈1,99584e+292	≈1,110223e-14

Стандарт IEEE754 предусматривает четыре способа округления чисел.

Способы округления чисел по стандарту IEEE 754:

1. Округление стремящееся к ближайшему целому.
2. Округление стремящееся к нулю.
3. Округление стремящееся к +∞
4. Округление стремящееся к -∞

Таблица 3. Примеры округления чисел до десятых

исходное число	к ближ. целому	к нулю	к +∞	к -∞
1,33	1,3	1,3	1,4	1,3
-1,33	-1,3	-1,3	-1,3	-1,4
1,37	1,4	1,3	1,4	1,3
-1,37	-1,4	-1,3	-1,3	-1,4
1,35	1,4	1,3	1,4	1,3
-1,35	-1,4	-1,3	-1,3	-1,4

Как происходит округление показано на примерах в таблице 3. При преобразовании чисел необходимо выбрать один из способов округления. По умолчанию это первый способ - округление к ближайшему целому.

Часто в различных устройствах используют второй способ - округление к нулю. При округлении к нулю нужно просто отбросить незначащие разряды числа, поэтому этот способ самый легкий в аппаратной реализации.

МАШИННЫЙ ЭПСИЛОН

Машинный эпсилон – это *минимальное* положительное число, такое, что при прибавлении к нему единицы результат будет отличен от единицы: $1+\varepsilon>1$.

Машинное эпсилон характеризует длину мантиссы, т.е. точность, с которой могут производиться вычисления с плавающей запятой.

Величина машинного эпсилон — это величина относительной погрешности, т.е. погрешность представления чисел с порядком 0 (от 1 до 2) будет равна eps, а с порядком "x" погрешность будет равна $2^x \cdot eps$.

Практическая важность машинного эпсилон связана с тем, что два (отличных от нуля) числа являются одинаковыми с точки зрения машинной арифметики, если их относительная разность по модулю меньше машинного эпсилон.

Ряд Тейлора

Ряд Тейлора – разложение функции в бесконечную сумму степенных функций. Ряд назван в честь английского математика Брука Тейлора.

Пусть функция f(x) бесконечно дифференцируема в некоторой окрестности точки a, тогда ряд

$$f(x) = f(a) + \sum_{k=1}^{\infty} \frac{f^{(k)}(a)}{k!}(x-a)^k$$

называется рядом Тейлора функции f в точке a.

В случае, если a = 0, этот ряд иногда называется **рядом Маклорена**.

Формула Тейлора используется при доказательстве большого числа теорем в дифференциальном исчислении. Говоря нестрого, формула Тейлора показывает поведение функции в окрестности некоторой точки.

Теорема:

- Пусть функция f(x) имеет n+1 производную в некоторой окрестности точки a, U(a, ε);
- Пусть x ∈ U(a, ε);
- Пусть p — произвольное положительное число.

Тогда: ∃ точка ξ ∈ (x, a) при x < a или ξ ∈ (a, x) при x > a:

$$f(x) = f(a) + \sum_{k=1}^{n} \frac{f^{(k)}(a)}{k!}(x-a)^k + \left(\frac{x-a}{x-\xi}\right)^p \frac{(x-\xi)^{n+1}}{n!p} f^{(n+1)}(\xi)$$

Это формула Тейлора с остаточным членом в общей форме (форме **Шлемильха — Роша**).

Различные формы остаточного члена.

В форме Лагранжа:

$$R_{n+1}(x) = \frac{(x-a)^{n+1}}{(n+1)!} f^{(n+1)}[a + \theta(x-a)] \qquad p = n+1$$

В форме Коши:

$$R_{n+1}(x) = \frac{(x-a)^{n+1}(1-\theta)^n}{n!} f^{(n+1)}[a+\theta(x-a)] \qquad p=1$$

Остаточный член в асимптотической форме (форме Пеано):

$$R_{n+1}(x) = o[(x-a)^n]$$

Постановка задачи

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 включая концы отрезка), 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 №13:

ряд	a	b	функция
$x - \frac{x^3}{3!} + \cdots + (-1)^n \frac{x^{2n+1}}{(2n+1)!}$	0.0	1.0	sin(x)

Описание программы

Определим через define параметры:

- NUMBERS_ITERS (максимальное количество итераций при вычислений по Тейлору);
- NUMBER_SEGMENTS (количество точек отрезка [a, b]);
- K_EPSILON (параметр k - экспериментально подбираемый коэффициент, обеспечивающий приемлемую сходимость).

Объявим следующие функции:

- double get_machine_epsilon() – возвращает машинное эпсилон;
- double get_func_value(double x) – возвращает значение заданной функции;
- double get_teilor_value(double x, double eps, int *num_iters) – возвращает значение суммы разложения заданной функции по Тейлору. В num_iters записывает количество пройденных итераций;
- void print_start_table(double a, double b, double eps) – печатает значение отрезка и машинного эпсилона, а также инициализирует таблицу;
- void print_string(int i, double x, double func_value, double tailor_value, int num_iters) – печатает строку в таблице.

```
#include <stdio.h>
#include <math.h>
#define NUMBER_ITERS 100
#define K_EPSILON 100.

double get_machine_epsilon();
double get_func_value(double x);
double get_teilor_value(double x, double eps, int *num_iters);
void print_start_table(double a, double b);
void print_string(int i, double x, double func_value, double teilor_value, int num_iters);
```

Алгоритм функции main:

1) Инициализация констант: eps = ε*k, концы отрезка a и b, step – шаг, на который будет увеличиваться переменная x в пределах данного отрезка.
2) Инициализация переменной x, присваивание ей значения константы a.
3) Инициализация таблицы.
4) Цикл от 0 до NUMBER_SEGMENTS. На каждой итерации вычисляются значения функции и суммы ряда Тейлора от переменной x, происходит вывод значений на экран, увеличивается значение переменной x на step.

```c
int main() {
    int number_segments;
    scanf("%d", &number_segments);
    const double eps = get_machine_epsilon() * K_EPSILON, a = 0, b = 1;
    const double step = (b-a)/(number_segments-1);
    double x = a;
    print_start_table(a, b);
    for (int i = 0; i < number_segments; i++) {
        int num_iters = 0;
        double func_value = get_func_value(x), teilor_value = get_teilor_value(x, eps, &num_iters);
        print_string(i+1, x, func_value, teilor_value, num_iters);
        x += step;
    }
    return 0;
}
```

Реализация остальных функций представлена ниже:

```c

double get_machine_epsilon() {
    double eps = 1.0;
    while (1.0 + eps / 2.0 > 1.0) eps = eps / 2.0;
    return eps;
}
double get_func_value(double x) {
    return sin(x);
}
long double factorial(long double n) {
    long double res = 1;
    for (long double i = 1; i <= n; i++) res *= i;
    return res;
}
double get_teilor_value(double x, double eps, int *num_iters) {
    double sum = get_func_value(0), currentVal = 1;
    int sign = 1, i;
    for (i = 0; i < NUMBER_ITERS && fabs(currentVal) > eps; i++) {
        currentVal = sign * pow(x, 2 * i + 1)/(factorial(2 * i + 1));
        sum += currentVal;

        sign = -sign;
    }
    *num_iters = i;
    return sum;
}
void print_start_table(double a, double b) {
    printf("F(x) = sin(x)\tx on the segment [%.2lf, %.2lf]\nMachine epsilon - %.54lf\nK_EPSILON = %f\n"
           "Max number of iterations - %d\n", a, b, get_machine_epsilon(), K_EPSILON, NUMBER_ITERS);
    printf("_______________________________________________________________________________________________\n");
    printf("|  i  |   x   |         F(x)          |     Taylor series     |  k  |          delta          |\n");
}
void print_string(int i, double x, double func_value, double teilor_value, int num_iters) {
    printf("| %.3d | % .2lf | %.18lf |% .18lf | %.3d |% .18lf |\n", i, x, func_value, teilor_value, num_iters, fabs(func_value-teilor_value));
}
```

В функции get_teilor_value цикл идёт до параметра NUMBER_ITERS или до того момента, пока текущее значение currentVal, вычисляемое на данной итерации, не будет находиться в окрестности eps = ε*k (по условию задачи).

Тесты

```
F(x) = sin(x)   x on the segment [0.00, 1.00]
Machine epsilon - 0.000000000000000222044604925031308084726333618164062500
K_EPSILON = 100.000000
Max number of iterations - 100

-----------------------------------------------------------------------------------------
|  i  |   x   |          F(x)          |     Taylor series      |  k  |         delta          |
| 001 |  0.00 | 0.000000000000000000 | 0.000000000000000000 | 001 | 0.000000000000000000 |
| 002 |  0.11 | 0.110882628509952980 | 0.110882628509952993 | 005 | 0.000000000000000014 |
| 003 |  0.22 | 0.220397743456122258 | 0.220397743456122258 | 006 | 0.000000000000000000 |
| 004 |  0.33 | 0.327194696796152207 | 0.327194696796152262 | 007 | 0.000000000000000056 |
| 005 |  0.44 | 0.429956363528355534 | 0.429956363528355534 | 007 | 0.000000000000000000 |
| 006 |  0.56 | 0.527415385771865530 | 0.527415385771865530 | 008 | 0.000000000000000000 |
| 007 |  0.67 | 0.618369803069737101 | 0.618369803069737101 | 008 | 0.000000000000000000 |
| 008 |  0.78 | 0.701697876146735400 | 0.701697876146735289 | 008 | 0.000000000000000111 |
| 009 |  0.89 | 0.776371921300660572 | 0.776371921300660572 | 009 | 0.000000000000000000 |
| 010 |  1.00 | 0.841470984807896616 | 0.841470984807896616 | 009 | 0.000000000000000000 |
```

```
F(x) = sin(x)   x on the segment [0.00, 1.00]
Machine epsilon - 0.000000000000000222044604925031308084726333618164062500
K_EPSILON = 100.000000
Max number of iterations - 100

-----------------------------------------------------------------------------------------
|  i  |   x   |          F(x)          |     Taylor series      |  k  |         delta          |
| 001 |  0.00 | 0.000000000000000000 | 0.000000000000000000 | 001 | 0.000000000000000000 |
| 002 |  0.05 | 0.052607283338072131 | 0.052607283338072131 | 005 | 0.000000000000000000 |
| 003 |  0.11 | 0.105068873765949117 | 0.105068873765949103 | 005 | 0.000000000000000014 |
| 004 |  0.16 | 0.157239481861750241 | 0.157239481861750241 | 006 | 0.000000000000000000 |
| 005 |  0.21 | 0.208974624062785469 | 0.208974624062785497 | 006 | 0.000000000000000028 |
| 006 |  0.26 | 0.260131022804650114 | 0.260131022804650114 | 006 | 0.000000000000000000 |
| 007 |  0.32 | 0.310567003320374901 | 0.310567003320374901 | 007 | 0.000000000000000000 |
| 008 |  0.37 | 0.360142886000719087 | 0.360142886000719142 | 007 | 0.000000000000000056 |
| 009 |  0.42 | 0.408721373228986162 | 0.408721373228986218 | 007 | 0.000000000000000056 |
| 010 |  0.47 | 0.456167929619045676 | 0.456167929619045731 | 007 | 0.000000000000000056 |
| 011 |  0.53 | 0.502351154603512540 | 0.502351154603512540 | 008 | 0.000000000000000000 |
| 012 |  0.58 | 0.547143146340222986 | 0.547143146340222875 | 008 | 0.000000000000000111 |
| 013 |  0.63 | 0.590419855929186443 | 0.590419855929186443 | 008 | 0.000000000000000000 |
| 014 |  0.68 | 0.632061430959033332 | 0.632061430959033443 | 008 | 0.000000000000000111 |
| 015 |  0.74 | 0.671952547431521330 | 0.671952547431521219 | 008 | 0.000000000000000111 |
| 016 |  0.79 | 0.709982729144857938 | 0.709982729144857827 | 008 | 0.000000000000000111 |
| 017 |  0.84 | 0.746046653651323277 | 0.746046653651323166 | 009 | 0.000000000000000111 |
| 018 |  0.89 | 0.780044443941860566 | 0.780044443941860455 | 009 | 0.000000000000000111 |
| 019 |  0.95 | 0.811881945049831355 | 0.811881945049831355 | 009 | 0.000000000000000000 |
| 020 |  1.00 | 0.841470984807896283 | 0.841470984807896283 | 009 | 0.000000000000000000 |
```

Заключение

По приведённым выше тестам можно заметить, что коэффициент k регулирует точность вычисления по Тейлору. Но при $k \leq 1$ точность начинает зависеть только от машинного эпсилона. Значение по формуле Тейлора будет отличаться от значения встроенной в Си функции в большинстве случаев. Это происходит ввиду погрешности, появляющийся из-за ограниченного диапазона представления вещественных чисел в памяти компьютера.

Формула Тейлора сводит вычисление трансцендентных функций к алгебраическим. Однако этот простой способ не применяется ввиду большой ресурсоёмкости и значительной погрешности.

При работе с вещественными числами стоит обязательно учесть особенности их представления в памяти компьютера.

Список источников

1. IEEE 754 - стандарт двоичной арифметики с плавающей точкой – URL: https://www.softelectro.ru/ieee754.html

2. Что нужно знать про арифметику с плавающей запятой – URL: https://habr.com/ru/post/112953/

3. Ряд Тейлора – URL: https://math.fandom.com/ru/wiki/Ряд_Тейлора