МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

“Московский Авиационный Институт”

(Национальный Исследовательский Университет)

Факультет: “Информационные технологии и прикладная математика”

Кафедра: “Вычислительная математика и программирование”

**Курсовой проект**

**по курсу «Основы информатики»**

**I семестр**

**Задание 3**

**“Вещественный тип. Приближенные вычисления. Табулирование функций.”**

Группа: М80 – 107Б-18

Студент: Гамов Павел Антонович

Преподаватель: Ридли Александра Николаевна

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2018.

**Содержание**

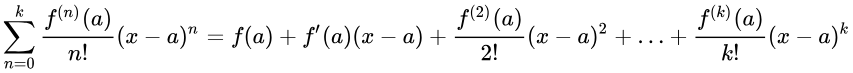
1. Введение
2. Подготовка
3. Ряд Тейлора
4. Числа с плавающей запятой
5. Машинный эпсилон
6. Вычисление и код
7. Заключение
8. Список использованной литературы

**Введение**

Современные компьютеры, ввиду своей быстроты и надежности, часто используются для выполнения различных вычислений, которые порой невозможно сделать вручную. Данный факт позволяет нынешним ученым и инженерам достигать невероятных точностей в расчетах, начиная от постройки дома до вычисления траекторий полета небесных тел. В данной работе нам необходимо познакомиться с таким понятием как машинный ноль и машинный эпсилон, разобраться в архитектуре машинного счисления и представления числа, а также применить эти знания для вычисления значений функции используя ряды Тейлора.

**Ряд Тейлора**

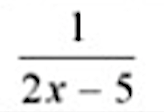
Ряд Тейлора – разложение функции в бесконечную сумму степенных функций.

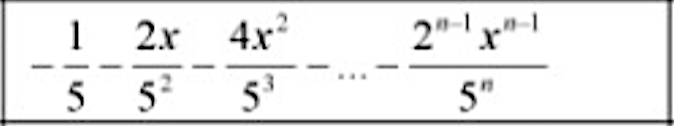


Данный метод позволяет разложить любую функцию в окрестности точки ‘a’ и имеющую в ней ‘k’ производных в сумму степенных слагаемых.

**Подготовка к реализации**

Для работы мне была предоставлена данная функция



Разложив ее в ряд Тейлора относительно точки 0 (ряд Маклорена) мы получаем данный ряд простых степенных функций. Общая формула, зависящая от параметра n, представлена справа от ряда. 

Таким образом найдя сумму бесконечного ряда слагаемых, где вместо х стоит необходимый нам аргумент, мы можем получить точное значение функции в данной точке. Тут мы сталкиваемся с проблемой: наш компьютер не способен найти бесконечную сумму слагаемых, нам надо выбрать, при каких n считать дальше уже нет смысла. Для этого стоит рассмотреть поподробнее архитектуру устройства чисел с плавающей запятой.

**Числа с плавающей запятой**

Компьютерная арифметика требует точного представления числа, с которым работает компьютер, ведь под каждое число отводится строго определённое количество памяти, чем оно больше, тем больше диапазон используемых чисел. Одним из таких видов можно считать числа со статической запятой. Представим себе умножение числа 1,2 на 1,2, результатом будет число 1,44, но компьютер не сможет записать последнюю четверку, ведь число данного порядка не может быть записано в память, отведенную ему, поэтому он выдаст ответ 1,4, что отличается от правды. Числа с плавающей запятой позволяют расширить диапазон представления чисел и увеличить точность счисления. Для этого в компьютерной записи данных чисел используется три компонента: бит под знак числа, биты под порядок и биты под мантиссу.

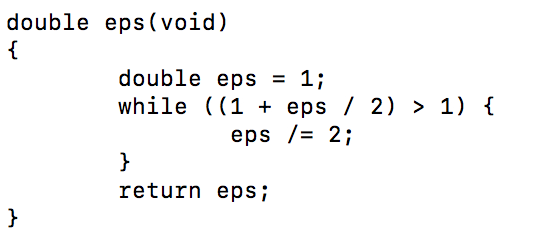


Рассмотрим 32 битное представление числа.

Десятичное число приводится к нормальному путем сдвига порядка пока число не больше нуля и не меньше 1. Отсюда получаем 1.01е в степени 2. Таким образом мантисса числа равна 01, а порядок 2 + 127 в двоичной. Прибавление числа 127 необходимо для избежание попадания порядка в отрицательную область. Данная запись числа позволяет работать с очень маленькими числами с точностью до 23 знака после запятой. Но что будет если мы выйдем за пределы? В какой-то момент станут бессмысленны операции над числами, находящимися в сильно различных степенях, так как при нормализации они будут теряться, уходя в ноль. Следовательно, если мы работаем с числами до 16 знака после запятой, прибавления чисел из -100 степени нам вообще не надо, поговорим про машинный эпсилон.

**Машинный эпсилон.**

Число, значение меньше которого при прибавлении к константе не будет изменять ее значения. Данная функция на языке Си позволяет найти эпсилон для точности вычисления 16 порядка, который чаще всего реализуется в числах двойной точности как double.

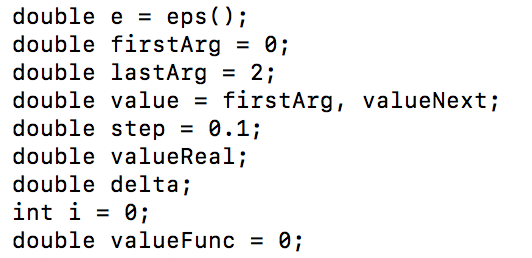


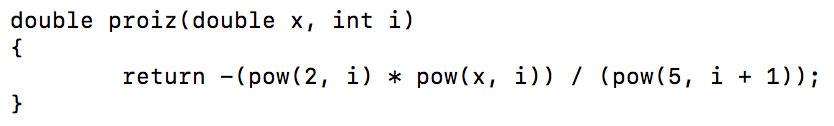
Вернемся к нашему заданию, чем дальше мы суммируем Ряд Тейлора, тем ближе мы приближаемся к верному значению, но тем больше времени и ресурсов мы тратим. Найдя машинный эпсилон для вычисления минимальной разницы чисел интересующего нас порядка, мы можем ограничить количество итераций, прекращая приближения значения, когда новое слагаемое меньше него.

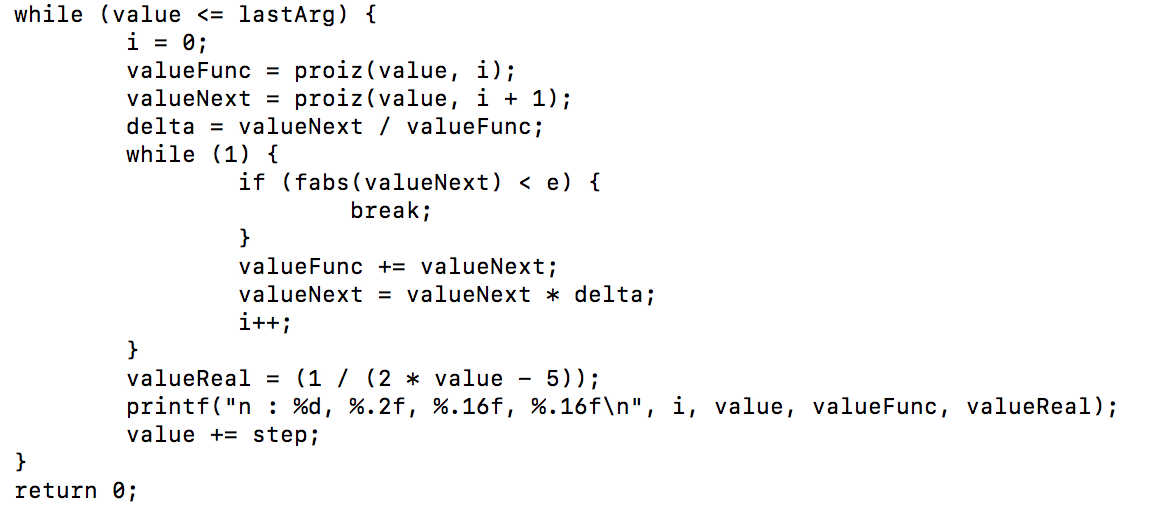
**Вычисления ряда Тейлора**



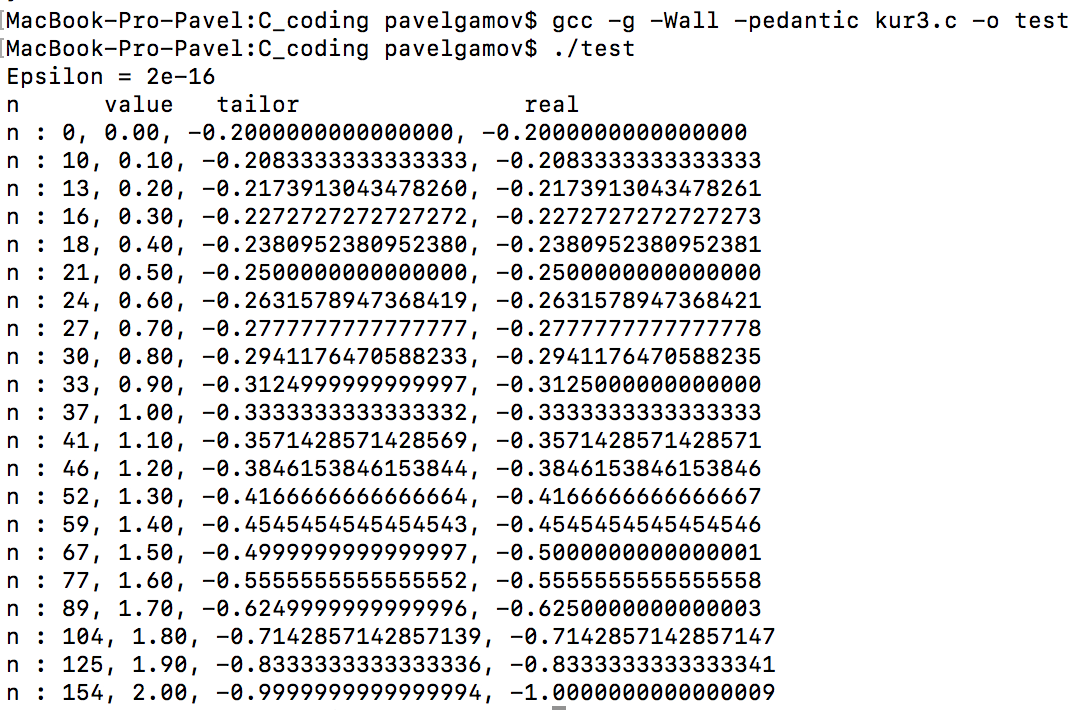
Присоединим math.h ради pow и fabs.







valueFunc – значение функции в точке, valueNext – вычисление следующего члена для нахождения delta, на значение которого мы будем умножать valueNext, для избежания постоянного пересчёта функции при новом значения параметра i. Внешний цикл while используется для того, чтобы найти значения в точках от 0 до 2 с шагом 0,1. Внутренний while для приближения значения пока не будет выполняться условие “valueNext < e”, где е – ранее посчитанный эпсилон. Далее идет вывод полученных данных в консоль, с последующим изменением аргумента value или выходом из цикла.



n – количество итераций, требуемых для достижения допустимой погрешности. tailor – то, что мы получили, value – наш икс, real – реальное значение функции.

**Заключение**

В данной работе мы познакомились с архитектурой чисел с плавающей запятой, применили алгоритмы для нахождения значения функции в определенной точке посредством разложения ее в ряд Тейлора определенной точности вычисления используя машинный эпсилон.

**Список использованной литературы:**

Ряд Тейлора –

https://ru.wikipedia.org/wiki/Ряд\_Тейлора

Машинный эпсилон –

http://www.cyberforum.ru/python/thread1320767.html

https://ru.wikipedia.org/wiki/Машинный\_ноль

Архитектура плавающих чисел -

https://ru.wikipedia.org/wiki/Число\_с\_плавающей\_запятой

https://www.youtube.com/watch?v=e7Wukn56-O4