МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В. Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**Лабораторная работа № 0**

по дисциплине: Вычислительная математика

тема: «Погрешности. Приближенные вычисления. Вычислительная устойчивость.»

Выполнил: ст. группы ПВ-223

Игнатьев Артур Олегович

Проверил:

асс. Четвертухин Виктор Романович

Белгород 2024г.

**Лабораторная работа №0**

**«Погрешности. Приближенные вычисления. Вычислительная устойчивость.»**

**Цель работы:** Изучить особенности организации вычислительных процессов, связанные с погрешностями, приближенным характером вычислений на компьютерах современного типа, вычислительной устойчивостью.

**Ход выполнения лабораторной работы:**

1) Запустить и проинтерпретировать результаты работы разных вычислительных схем для простого арифметического выражения на языке Rust.

Код программы:

// демонстрация чувствительности результата вычисления к последовательности  
// арифметических операций  
fn main() {  
 let num1: f32 = 0.23456789;  
 let num2: f32 = 1.5678e+20;  
 let num3: f32 = 1.2345e+10;  
 let result1 = (num1 \* num2) / num3;  
 let result2 = (num1 / num3) \* num2;  
 let result3: f64 = num1 as f64 \* num2 as f64 / num3 as f64;  
 println!("({} \* {}) / {} = {}", num1, num2, num3, result1);  
 println!("({} / {}) \* {} = {}", num1, num3, num2, result2);  
 println!(" {} \* {} / {} = {}", num1, num2, num3, result3);  
}

Результат выполнения:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

2) Запустить и проинтерпретировать результаты работы разных вычислительных схем для интерационного и неитерационного вычисления на языке Rust.

Код программы:

// демонстрация накопления погрешности для итерационного процесса  
// версия для одинарной точности  
fn main() {  
 let numbers = [1.0f32, 20., 300., 4000., 5e6, f32::*MIN\_POSITIVE*,  
 f32::*MAX*\*0.99]; // вектор с числами одинарной точности  
 let iterations = 10; // число итераций  
 for &number in &numbers {  
 let mut result = number;  
 for \_ in 0..iterations {  
 result = result.sqrt(); // послед. извлечение квадратного корня  
 }  
 for \_ in 0..iterations {  
 result = result \* result; // послед. возведение числа в квадрат  
 }  
 let error = (number - result).abs();  
 println!("Исх-е значение: {:e}, результат: {:e}, абс-ая погрешность:  
{:e}, отн-ая погрешность: {:e} (%)", number, result, error, error \* 100. /  
 number);  
 }  
}

Результат выполнения:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Код программы:

// замена итерации функцией  
// версия для одинарной точности c powf  
fn main() {  
 let numbers = [1.0f32, 20., 300., 4000., 5e6, f32::*MIN\_POSITIVE*,  
 f32::*MAX* \* 0.99];  
 let iterations = 10;  
 for &number in &numbers {  
// извлекаем корень  
 let intermediate = number.powf(1.0f32 / (1 << iterations) as f32);  
// восстанавливаем значение  
 let result = intermediate.powf((1 << iterations) as f32);  
 let error = (number - result).abs();  
 println!("Исх-е значение: {:e}, результат: {:e}, абс-ая погрешность:  
{:e}, отн-ая погрешность: {:e} (%)", number, result, error, error \* 100. /  
 number);  
 }  
}

Результат выполнения:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

3) С помощью программы на языке Rust вывести на экран двоичное представление машинных чисел одинарной точности стандарта IEEE 754 для записи: числа π, бесконечности, нечисла (NaN), наименьшего положительного числа, наибольшего положительного числа, наименьшего отрицательного числа. Сформулировать обоснование полученных результатов в пунктах 1 и 2, опираясь на двоичное представление машинных чисел.

Код программы:

fn main() {  
 let pi = std::f32::consts::*PI*;  
 let infinity = std::f32::*INFINITY*;  
 let nan = std::f32::*NAN*;  
 let smallest\_positive = std::f32::*MIN\_POSITIVE*;  
 let largest\_positive = std::f32::*MAX*;  
 let smallest\_negative = -std::f32::*MIN\_POSITIVE*;  
 println!("π: {}", float\_to\_binary\_string(pi));  
 println!("Infinity: {}", float\_to\_binary\_string(infinity));  
 println!("NaN: {}", float\_to\_binary\_string(nan));  
 println!("Smallest Positive Number: {}",  
 float\_to\_binary\_string(smallest\_positive));  
 println!("Largest Positive Number: {}",  
 float\_to\_binary\_string(largest\_positive));  
 println!("Smallest Negative Number: {}",  
 float\_to\_binary\_string(smallest\_negative));  
}  
fn float\_to\_binary\_string(num: f32) -> String {  
 let bits = num.to\_bits();  
 format!("{:032b}", bits)  
}

Результат выполнения:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

**Индивидуальное задание**

**Варивнт 3**



Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Код программы:

fn main() {  
 // Входные данные  
 let numbers: Vec<f32> = vec  
 0.000001, 0.000001, 0.000001, 0.000001, 0.000001,  
 0.000001, 0.000001, 0.000001, 0.000001, 0.000001,  
 ];  
  
 // Схема с потерей точности  
 let mut res: f32 = 0.0;  
 for &num in &numbers {  
 res += num;  
 }  
 println!("Прямая схема: {}", res);  
  
 // Улучшенная схема  
 let kahan\_res = kahan\_sum(&numbers);  
 println!("Улучшенная схема: {}", kahan\_res);  
}  
  
fn kahan\_sum(numbers: &Vec<f32>) -> f32 {  
 let mut res: f32 = 0.0;  
 let mut c: f32 = 0.0;  
 for &num in numbers {  
 let y: f32 = num - c;  
 let t: f32 = res + y;  
 c = (t - res) - y;  
 res = t;  
 }  
 res  
}

Результат выполнения:

![Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

В этом примере используются очень маленькие числа (0.000001), которые могут потерять точность из-за ошибок округления в прямой схеме, но которые учитываются более эффективно в улучшенной схеме Кэхэна.

**Вывод:** в ходе выполнения лабораторной работы были изучены особенности организации вычислительных процессов, связанные с погрешностями, приближенным характером вычислений на компьютерах современного типа, вычислительной устойчивостью.