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:

```rust
// 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:

```
    Finished dev [unoptimized + debuginfo] target(s) in 0.02s
     Running `target\debug\computational_mathematics.exe`
(0.2345679 * 156780000000000000000) / 12345000000 = 2978983700
(0.2345679 / 12345000000) * 156780000000000000000 = 2978984000
 0.2345679 * 156780000000000000000 / 12345000000 = 2978983717.267449

Process finished with exit code 0
```

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:

```
Исх-е значение: 1e0, результат: 1e0, абс-ая погрешность:
0e0, отн-ая погрешность: 0e0 (%)
Исх-е значение: 2e1, результат: 2.000009e1, абс-ая погрешность:
8.9645386e-5, отн-ая погрешность: 4.4822693e-4 (%)
Исх-е значение: 3e2, результат: 3.0001422e2, абс-ая погрешность:
1.4221191e-2, отн-ая погрешность: 4.740397e-3 (%)
Исх-е значение: 4e3, результат: 4.0001064e3, абс-ая погрешность:
1.0644531e-1, отн-ая погрешность: 2.6611327e-3 (%)
Исх-е значение: 5e6, результат: 4.9994865e6, абс-ая погрешность:
5.135e2, отн-ая погрешность: 1.027e-2 (%)
Исх-е значение: 1.1754944e-38, результат: 1.17548e-38, абс-ая погрешность:
1.43e-43, отн-ая погрешность: 1.2159348e-3 (%)
Исх-е значение: 3.3687953e38, результат: 3.3686973e38, абс-ая погрешность:
9.796404e33, отн-ая погрешность: 2.9079844e-3 (%)

Process finished with exit code 0
```

Код программы:

```
// замена итерации функцией
// версия для одинарной точности с 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:

```
Исх-е значение: 1e0, результат: 1e0, абс-ая погрешность:
0e0, отн-ая погрешность: 0e0 (%)
Исх-е значение: 2e1, результат: 2.0000069e1, абс-ая погрешность:
6.866455e-5, отн-ая погрешность: 3.4332275e-4 (%)
Исх-е значение: 3e2, результат: 3.0000873e2, абс-ая погрешность:
8.728027e-3, отн-ая погрешность: 2.9093425e-3 (%)
Исх-е значение: 4e3, результат: 4.0001143e3, абс-ая погрешность:
1.1425781e-1, отн-ая погрешность: 2.8564453e-3 (%)
Исх-е значение: 5e6, результат: 5.000186e6, абс-ая погрешность:
1.86e2, отн-ая погрешность: 3.72e-3 (%)
Исх-е значение: 1.1754944e-38, результат: 1.175497e-38, абс-ая погрешность:
2.7e-44, отн-ая погрешность: 2.2649765e-4 (%)
Исх-е значение: 3.3687953e38, результат: 3.3687553e38, абс-ая погрешность:
3.9956347e33, отн-ая погрешность: 1.1860722e-3 (%)

Process finished with exit code 0
```

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:

```rust
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:

```
    Finished dev [unoptimized + debuginfo] target(s) in 0.02s
     Running `target\debug\computational_mathematics.exe`
π: 01000000010010010000111111011011
Infinity: 01111111100000000000000000000000
NaN: 01111111110000000000000000000000
Smallest Positive Number: 00000000100000000000000000000000
Largest Positive Number: 01111111011111111111111111111111
Smallest Negative Number: 10000000100000000000000000000000

Process finished with exit code 0
```

Индивидуальное задание

Варивнт 3

	«Прямая» схема со слабой точностью	Улучшенный вариант

3.	numbers = […] # вещ. числа
	res = sum(numbers)
		# алгоритм Кэхэна
		def kahan_sum(numbers):
		  res = .0
		  c = .0
		  for num in numbers:
		  y = num - c
		  t = res + y
		  c = (t - res) - y
		  res = t
		  return res

Код программы:

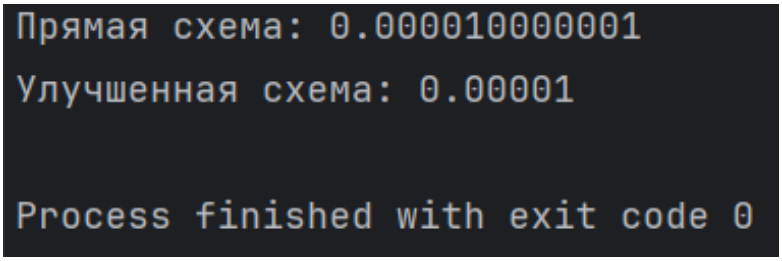
```rust
fn main() {
    // Входные данные
    let numbers: Vec<f32> = vec![
        0.000001, 0.000001, 0.000001, 0.000001, 0.000001,
        0.000001, 0.000001, 0.000001, 0.000001, 0.000001,
    ];

    // 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:



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.