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)

ИИТУС
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
Дмитриев Андрей
Проверил:
Четвертухин В.Р.

Белгород, 2024 г.

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

Цель работы обуславливает постановку и решение следующих **задач**:

1) Рассмотреть источники погрешности в решении численных задач и способы их оценки.

2) Изучить особенности работы с машинными числами как с результатом дискретной

проекции вещественных чисел на конкретную архитектуру компьютера.

3) Выяснить условия обеспечения вычислительной устойчивости решения численных задач.

4) Выполнить индивидуальное задание, закрепляющее на практике полученные знания (номер задания соответствует номеру студента по журналу; если этот номер больше, чем максимальное число заданий, тогда вариант задания вычисляется по формуле: номер по журналу % максимальный номер задания, где % - остаток от деления). На основании представленных вычислительных схем («прямой» и улучшенной) подобрать такие входные данные, что в первом случае схема демонстрировала бы заметную потерю в точности, а вторая на тех же входных данных — улучшала бы результат. В ходе выполнения данного задания следует использовать любой нескриптовой язык программирования, поддерживающий работу с машинными числами одинарной точности. Скриптовой язык Python, использованный для описания вычислительной схемы в задании, не использовать.

5) Отразить в отчете все полученные результаты. Сделать выводы.

Ход выполнения лабораторной работы

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

Код:

```rust
pub fn run() {
    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:

```
(0.2345679 * 156780000000000000000) / 12345000000 = 2978983700
(0.2345679 / 12345000000) * 156780000000000000000 = 2978984000
 0.2345679 * 156780000000000000000 / 12345000000 = 2978983717.267449
```

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

Код:

```rust
pub fn run1() {
    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:

```
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 (%)
```

```rust
pub fn run2() {
    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:

```
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 (%)
```

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:

```rust
pub fn run() {
    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:

```
π: 01000000010010010000111111011011
Infinity: 01111111100000000000000000000000
NaN: 01111111110000000000000000000000
Smallest Positive Number: 00000000100000000000000000000000
Largest Positive Number: 01111111011111111111111111111111
Smallest Negative Number: 10000000100000000000000000000000
```

Вывод:

В задании 1, от порядка вычесления зависит точность. Это происходит вследствие ограниченной мантиссы, чем мантисса больше, тем больше точность (пример 3). В примере 1 сначала выполняется умножение с числом многобольше единицы и числом меньше единицы, что даёт точнее

записать промежуточный результат. Во 2-ом примере первым действием отсекается часть мантисы и далее идёт большая погрешность.

В задании 2, вариант неитерационного алгоритма оказался точнее из-за того, что происходило, как можно меньшее количество действий, но на вход для первого вызова функции powf подавалось число имеющее сильную погрешность. Так для числа 4000 итерационный алгоритм оказался точнее.

Вариант 2:

Подобрать такие входные данные, что в первом случае схема демонстрировала бы заметную потерю в точности, а вторая на тех же входных данных — улучшала бы результат.

Коды реализаций:

```rust
pub fn pow_simpl(x: f32, n: i32) -> f32 {
    let mut res: f32 = 1.;
    for _ in 0..n {
        res *= x;
    }

    return res;
}
```

```rust
pub fn pow_cool(x: f32, n: i32) -> f32 {
    let mut log_res: f32 = 0.;
    for _ in 0..n {
        log_res += x.ln();
    }
    return log_res.exp();
}
```

Таблица результатов:

x^y:	3^7	3^12	3^20
Real res:	2187	531441	3486784401
Simpl alg:	2187	531441	3486784500
Cool alg:	2187.0002	531440.1	3486765800

x^y:	727^2	727^4	727^7
Real res:	528529	279342903841	107334880228778779303
Simpl alg:	528529	279342900000	107334880000000000000
Cool alg:	528528.8	279342700000	107335370000000000000

x^y:	3.124^7	3.124^12	3.124^20
Real res:	2903.8700	864036.92	7838275140.77095882
Simpl alg:	2903.8706	864037.2	7838279000
Cool alg:	2903.8716	864037.6	7838283300

x^y:	4.12463^7	4.12463^12	4.12463^18
Real res:	20309.3008	24244875.3	119379548577.785
Simpl alg:	20309.303	24244878	119379550000
Cool alg:	20309.291	24244822	119379720000

x^y:	0.12^4	0.12^7	0.12^12
Real res:	0.00020736	0.00000035831808	0.000000000008916100448256
Simpl alg:	0.00020735998	0.000000358318	0.000000000008916098
Cool alg:	0.00020735996	0.00000035831746	0.000000000008916062

Вывод:

Улучшенный алгоритм не дал прироста точности, даже наоборот представлял менее точные результаты. Это происходит из-за потери дробной части при вычислении логарифма.

Вывод: Изучили особенности организации вычислительных процессов, связанные с погрешностями, приближенным характером вычислений на компьютерах современного типа, вычислительной устойчивостью. На практике увидели, как подходы к вычислению вещественных значений влияют на точность результата. В ходе исследования по индивидуальному заданию «улучшенный» алгоритм показал во многих тестах результат хуже, чем при использовании примитивного алгоритма.

GitHub: https://github.com/AnDreV133