МОСКОВСКИЙ ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Физтех-школа аэрокосмических технологий

Лабораторная работа № 2

Особенности работы чисел с плавающей точкой

Автор:
Леонид Ефремов
Б03-403

Долгопрудный 2024

Содержание

В данной лабораторной работе рассматриваестся тип данных вещественных чисел в C++

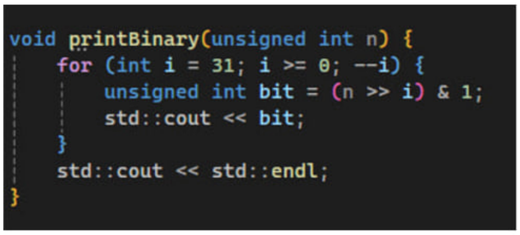
Целью лабораторной работы является определение свойств чисел с плавающей точкой, поиск ситуаций с их нерпедсказуемой работой. Определение условий корректной работы.

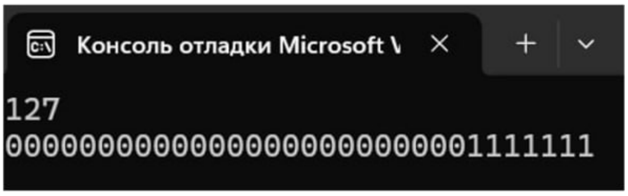
0.1 Эксперимент

С посощью C++ и Python изучаем работу C++. Управляющий код в отчёте.

1 unsigned int -> binary

Выведем на экран вид хранения беззнакового целого числа:





2 float -> binary

Выведем на экран вид хранения числа с плавающей точкой:

```cpp
union FloatUnion {
    float f;
    unsigned int n;
};

void printFloatInBinary(float number) {
    FloatUnion u;
    u.f = number;
    printBinary(u.n);
}
```

```
Консоль отладки Microsoft \
5.5
01000000101100000000000000000000
```

Расшифруем запись с экрана: Первый бит 0 означает что число положительное. Затем идут 8 бит экспоненты: 10000001.
Затем идет мантисса 01100000000000000000000. Таким образом число будет иметь вид 101.1. Целая часть при переводе действительно дает 5, а дробная часть 0.5

3 Переполнение мантиссы

Напишем код, который будет сохранять во float числа вида 10^n, где n будет возрастать:

```cpp
int main() {
    float number=10;
    int n = 11;
    float res = 1;
    for (int i = 0; i < n; i++) {
        res *= number;
        std::cout <<std::fixed<< "Curr: " << res << std::endl;
        std::cout << "Bin: "; printFloatInBinary(res);
        std::cout << '\n';
    }
```

```
Curr: 1000000000.000000
Bin: 01001110011011100110101100101000

Curr: 10000000000.000000
Bin: 01010000000101010000001011111001

Curr: 99999997952.000000
Bin: 01010001101110100100001110110111
```

Запустив программу заметим, что начиная с 11 степени, сохраненное значение вовсе не является степенью 10:

Числа оказываются достаточно близкими к истинному значению, но чем больше степень – тем больше разница. Это связано с дискретностью типа float.

4 БезРиконечный Морти(цикл)

```cpp
int main() {
    float number=10;
    int n=0;
    for (float i = 16767300; i < 20000000; i++) {

        std::cout <<std::fixed<< "Curr: " << i << std::endl;
        std::cout << "Int: "<<n;
        std::cout << '\n';

        if (n == 16777310) { break; }
            n++;
    }
```

```
Curr: 16777216.000000
Int: 172192
Curr: 16777216.000000
Int: 172193
Curr: 16777216.000000
Int: 172194
```

Выполнение кода прерывается на 16777310, так как начиная с 16777216 итератор цикла i перестает меняться и цикл становится бесконечным:

5 Расчет числа π(Пи)

Будем находить число π с помощью алгоритмов Эйлера, Лейбница, Виета и Валлиса. Построим графики зависимости найденного числа от итераций для каждого алгоритма.

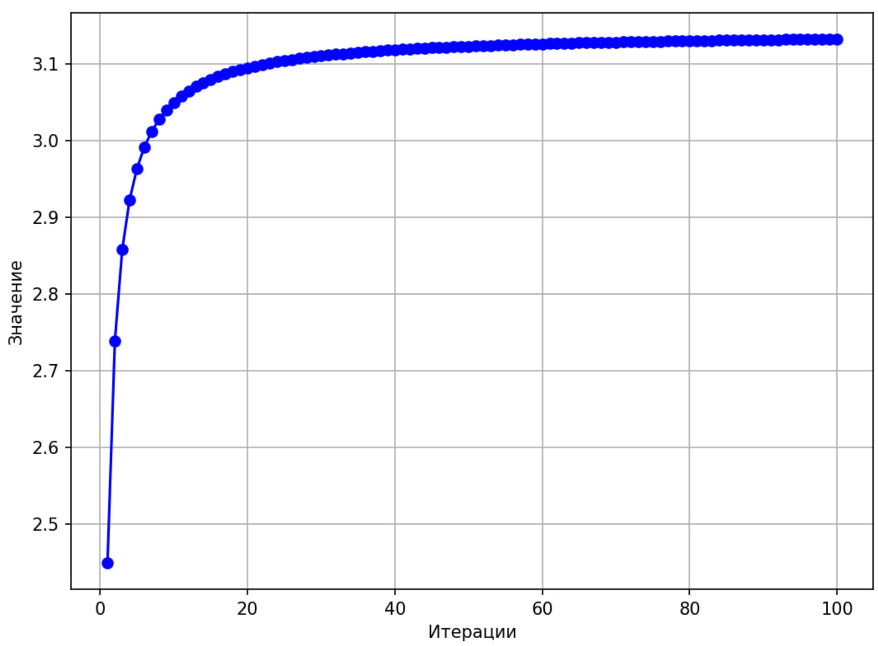
```cpp
// Алгоритм Эйлера
void euler_pi(int iterations, const std::string& file_name) {
    float result = 0;
    std::ofstream file(file_name);
    if (file.is_open()) {
        for (int i = 1; i <= iterations; ++i) {
            result += 1 / (pow(i, 2));
            float pi_approximation = sqrt(result * 6);
            file << i << " " << pi_approximation << std::endl;
        }
        file.close();
    }
}
```

```cpp
// Алгоритм Лейбница
void leibniz_pi(int iterations, const std::string& file_name) {
    float result = 0;
    int sign = 1;
    std::ofstream file(file_name);
    if (file.is_open()) {
        for (int i = 1; i <= iterations * 2; i += 2) {
            result += sign * (1.0 / i);
            sign *= -1;
            float pi_approximation = result * 4;
            file << i << " " << pi_approximation << std::endl;
        }
        file.close();
    }
}
```

```cpp
// Алгоритм Виета
void viete_pi(int iterations, const std::string& file_name) {
    float result = 2;
    std::ofstream file(file_name);
    if (file.is_open()) {
        for (int i = 1; i <= iterations; ++i) {
            result = 2 + sqrt(result);
            float pi_approximation = 2 * 2 / result;
            file << i << " " << pi_approximation << std::endl;
        }
        file.close();
    }
}
```

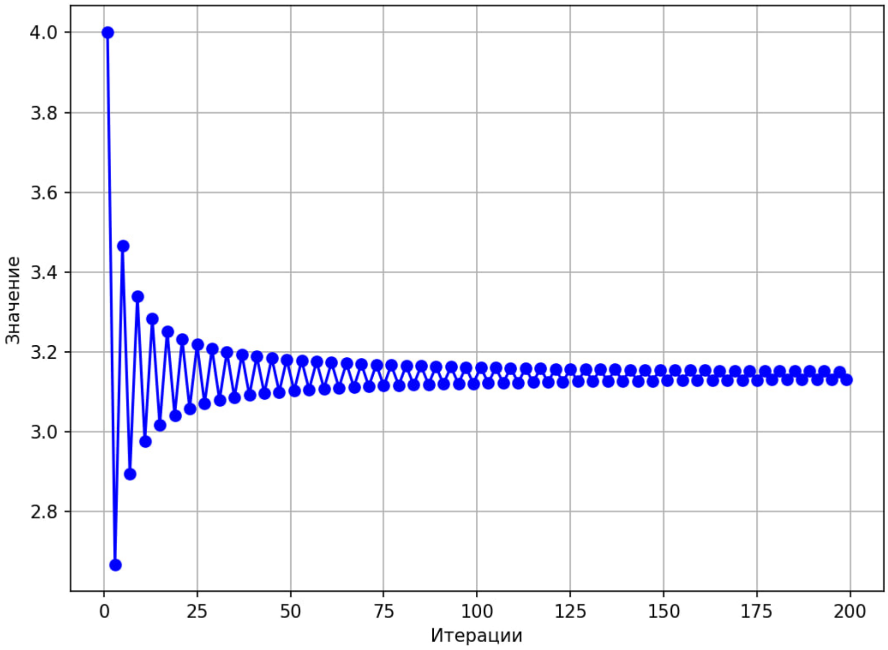
```cpp
// Алгоритм Валлиса
void wallis_pi(int iterations, const std::string& file_name) {
    float result = 1;
    std::ofstream file(file_name);
    if (file.is_open()) {
        for (int i = 1; i <= iterations; ++i) {
            result *= (pow(2 * i, 2)) / ((2 * i - 1) * (2 * i + 1));
            float pi_approximation = result * 2;
            file << i << " " << pi_approximation << std::endl;
        }
        file.close();
    }
}
```

5.1 Алгоритм Эйлера

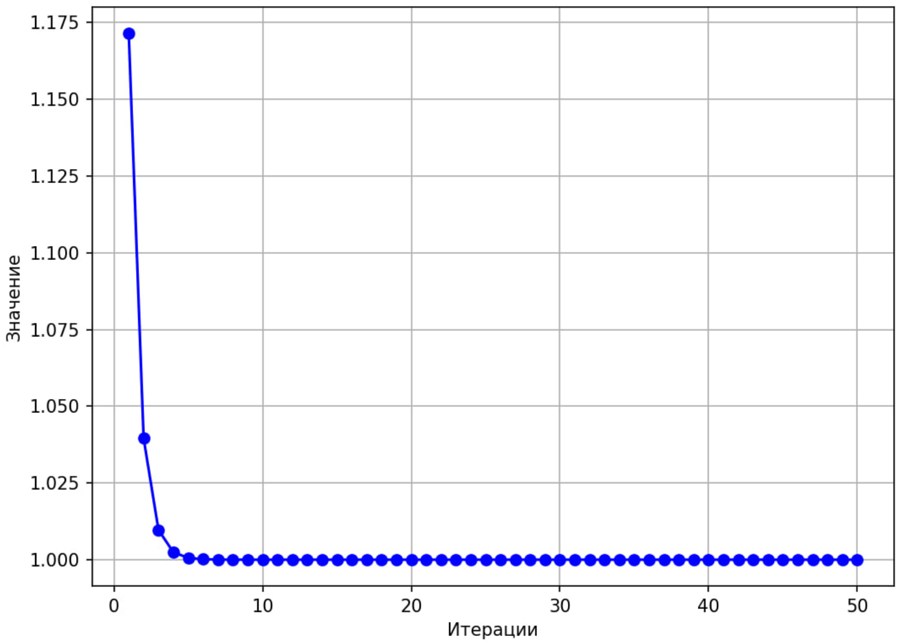


Алгоритм за малое количество итераций достигает значения 3.14139342308044433359375 и больше не меняется, это связано с тем, что прибавляемые числа в алгоритме уже не могут "перескочить"до следующего значения float.

5.2 Алгоритм Лейбница

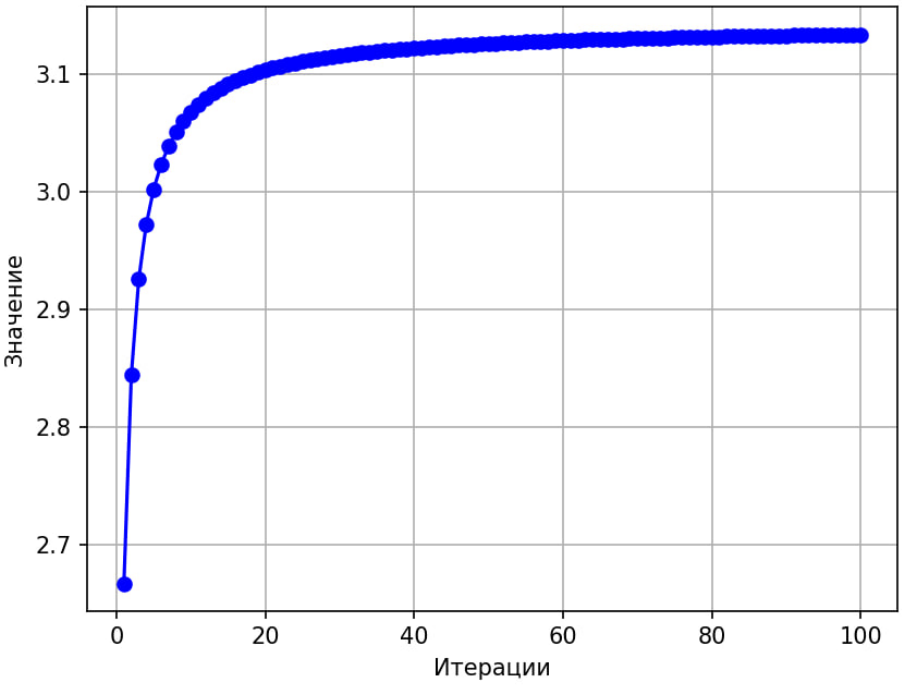


5.3 Алгоритм Виета



Этот алгоритм очень быстро ломается.

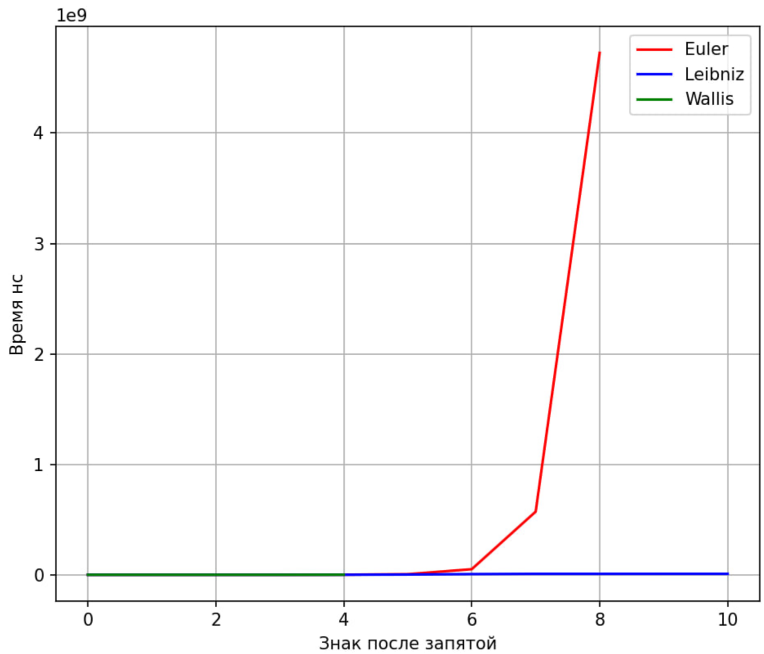
5.4 Алгоритм Валлиса

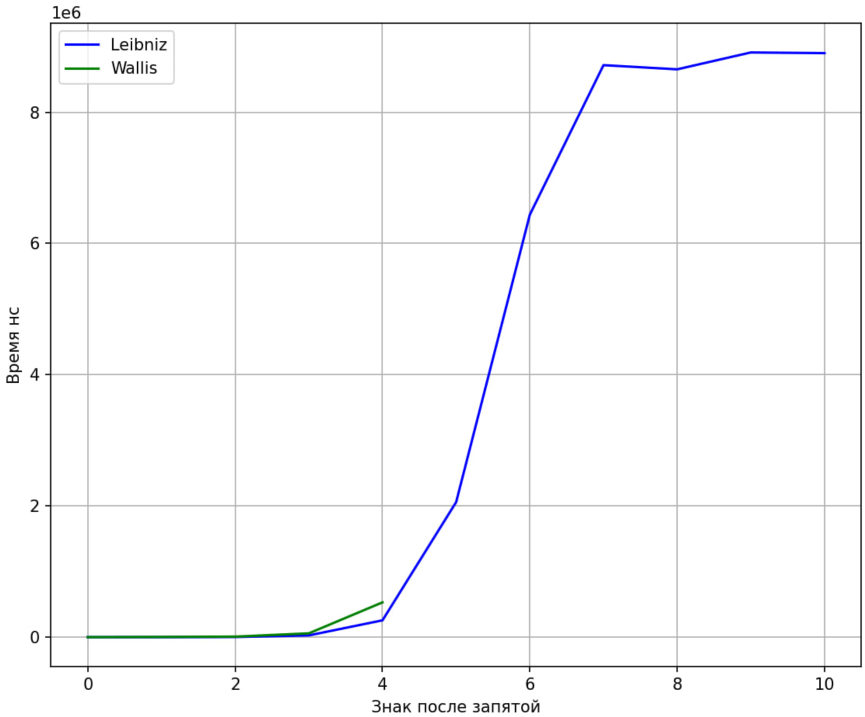


Алгоритм за малое количество итераций достигает значения почим истиного значения и больше не меняется, это связано с тем, что прибавляемые числа в алгоритме уже не могут "перескочить"до следующего значения float. График напоминает алгоритм Эйлера.

6 Время расчета числа π(Пи)

Для каждого алгоритма выше рассчитаем то, за какое среднее время и количество итераций они достигают до каждого знака π.
Затем, объединим данные и построим графики:





Алгоритм Эйлера и Валлиса оказался неспособен вычислить больше 8 знаков π даже на типе double и поэтому его график обрывается.
Самым быстрым и точным алгоритмом оказался алгоритм Лейбница. Он также не "ломается"на больших итерациях, а просто застывает.

7 Вывод

В ходе выполнения лабораторной работы было произведено определение свойств чисел с плавающей точкой, найдены ситуации с их нерпедсказуемой работой, расшифрована их запись в памяти. Произведена оценка эффективности алгоритмов поиска числа Пи на числах с плавающей точкой.