

МОСКОВСКИЙ ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ  
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Физтех-школа аэрокосмических технологий

Лабораторная работа  
Хранение переменных. C++

Работу выполнил  
Лохматов Арсений Игоревич  
Б03-303

Долгопрудный 2023



# Задача 1

**Цель:** С помощью union и функции из предыдущего пункта вывести на экран представление float в памяти компьютера. Проверьте, что IEEE 754-2008 (стандартный стандарт) действительно работает.

**Ход работы:** Функцию для перевода числа в двоичное представление возьмём из предыдущей задачи. Union объединяет в себе два поля, которые начинаются с одной ячейки памяти. Мы вводим float f, тогда как вызываем функцию от поля unsigned int u, и выводим результат на экран.

```
#include <iostream>
using namespace std;

void Binary(unsigned int n) {
    int x[32] = {0};
    for (int i = 0; i < 32; i++) {
        x[i] = (n - ((n >> 1) << 1));
        n = n >> 1;
    }
    for (int i = 31; i > 0; i--) {
        cout << x[i];
    }
    cout << '\n';
}

union fu{
    float f;
    unsigned int u;
};

int main() {
    fu a;
    cin >> a.f;
    Binary(a.u);
    return 0;
}
```

Результаты программы приведены на фото. Действительно, IEEE 754-2008 (стандартный стандарт) работает правильно.

```
PS C:\Users\teorema\Projects\lab-work-float> .\task1
165.834
0100001100100101110101011000000
```

```
PS C:\Users\teorema\Projects\lab-work-float> .\task1
-674.345
1100010000101000100101100001010
```

Рис. 2: Результаты работы программы для задания 1.

## Задача 2

**Цель:** Воспроизвести переполнение мантиссы. Проще и нагляднее всего – в цикле выводить десятичное и двоичное представление степеней десятки (10.0, 100.0, 1000.0, ... ).

**Ход работы:** Основную часть программы возьмём из предыдущей задачи. Но теперь будем поочерёдно выводить числа и его двоичное представление, каждое число увеличивая в 10 раз. Посмотрим, что выведет программа.

```
#include <iostream>

using namespace std;

void Binary(unsigned int n) {
    int x[32] = {0};
    for (int i = 0; i < 32; i++) {
        x[i] = (n - ((n >> 1) << 1));
        n = n >> 1;
    }
    for (int i = 31; i > 0; i--) {
        cout << x[i];
    }
    cout << '\n';
}

union fu{
    float f;
    unsigned int u;
};

int main() {
    float big = 1.0;
    cout << fixed;
    cout.precision(2);
    fu a;
    for (int i = 0; i < 100; i++) {
        a.f = big;
        cout << a.f << '\n';
        Binary(a.u);
        big *= 10.0;
    }
    return 0;
}
```

Результаты программы приведены ниже. Видим, что, начиная с определённого достаточно большого вещественного числа, возникает переполнение мантиссы.

```
1.00
00111111100000000000000000000000
10.00
01000001001000000000000000000000
100.00
01000010110010000000000000000000
```

```

1000.00
01000100011110100000000000000000
10000.00
01000110000111000100000000000000
100000.00
01000111110000110101000000000000
1000000.00
01001001011101000010010000000000
10000000.00
0100101100011000100101101000000
100000000.00
0100110010111110101111000010000
1000000000.00
0100111001101110011010110010100
10000000000.00
0101000000010101000000101111100
99999997952.00
0101000110111010010000111011011
999999995904.00
0101001101101000110101001010010
9999999827968.00
0101010100010001100001001110011
100000000376832.00
0101011010110101111001100010000
999999986991104.00
0101100001100011010111111010100
10000000272564224.00
0101101000001110000110111100101
99999998430674944.00
0101101110110001101000101011110

...
1000000040918478759629753193752166400.00
0111101101000000100101111100111
10000000567641112624826207124609564672.00
0111110011110000101111011100001
100000006944061726476491472742798852096.00
0111111010010110011101101001101
inf
01111111100000000000000000000000
inf
01111111100000000000000000000000
inf
01111111100000000000000000000000
inf
01111111100000000000000000000000

```

## Задача 3

**Цель:** Воспроизвести бесконечный цикл, возникающий из-за дискретности и неравномерности диапазона значений float. Тут воспользуемся тем фактом, что float'ы при увеличении своего значения начинают стоять все дальше и дальше друг от друга. Например, если вы пойдете с шагом 1 от нуля до большого числа, то рано или поздно дойдете до того места, где расстояние между соседними float'ами будет больше единицы. Тогда при прибавлении единицы к переменной цикла эта переменная вообще не изменится – мы как бы не смогли перепрыгнуть на следующее число, наш шаг потерялся в погрешности. И все, застряли в бесконечном цикле, который сами создали.

**Ход работы:** Напишем программу, которая создаёт этот бесконечный цикл. Но затем немного модифицируем её так, чтобы она вывела одно число - тот самый float, начиная с которого результат не будет изменяться. Для этого на каждом шаге будем сравнивать полученное число с числом на единицу больше.

```
#include <iostream>
using namespace std;

int main() {
    for (float i = 0; i <= 1000000000; i++) {
        if (i+1 == i) {
            cout << i << "\n";
            break;
        }
    }
    return 0;
}
```

Результаты программы приведен ниже. Программа, начиная с 1.67772e+07 впадает в бесконечный цикл, постоянно выводя данное число, всё время пытаясь его увеличить на единицу, но безрезультатно.

```
PS C:\Users\teorema\Projects\lab-work-float> .\task3
1.67772e+07
```

Рис. 3: Результаты работы программы для задания 3.

## Задача 4

**Цель:** Науглнить 4-5 итерационных формул для числа пи. Посчитать зависимость пи, вычисленного по этим формулам, от числа итераций. Построить графики для разных формул (если что, в конце файла есть небольшой мануал). Лучше использовать float - на double будут те же проблемы, что и у float, но лучше замаскированы и менее наглядны.

- Сходится монотонно с одной стороны:

– Формула Валлиса:

$$\frac{\pi}{2} = \frac{2 \cdot 2}{1 \cdot 3} \cdot \frac{4 \cdot 4}{3 \cdot 5} \cdot \frac{6 \cdot 6}{5 \cdot 7} \cdot \frac{8 \cdot 8}{7 \cdot 9} \cdot \dots;$$

Функция, которая записывает в файл номер операции и соответствующее ей значение числа  $\pi$ , приведена ниже. По этим точкам построим график, чтобы наглядно увидеть зависимость значения на данной итерации от числа итераций.

```
void rad1 () {  
    ofstream f("1.csv", ios::out);  
    float pi=1, t=2;  
    for (int i=1; i< 10001; i+=10) {  
        pi *= (t*t)/((t-1)*(t+1));  
        t += 2;  
        f << i<<'\t'<<pi*2 << endl;  
    }  
    cout << '\n';  
};
```

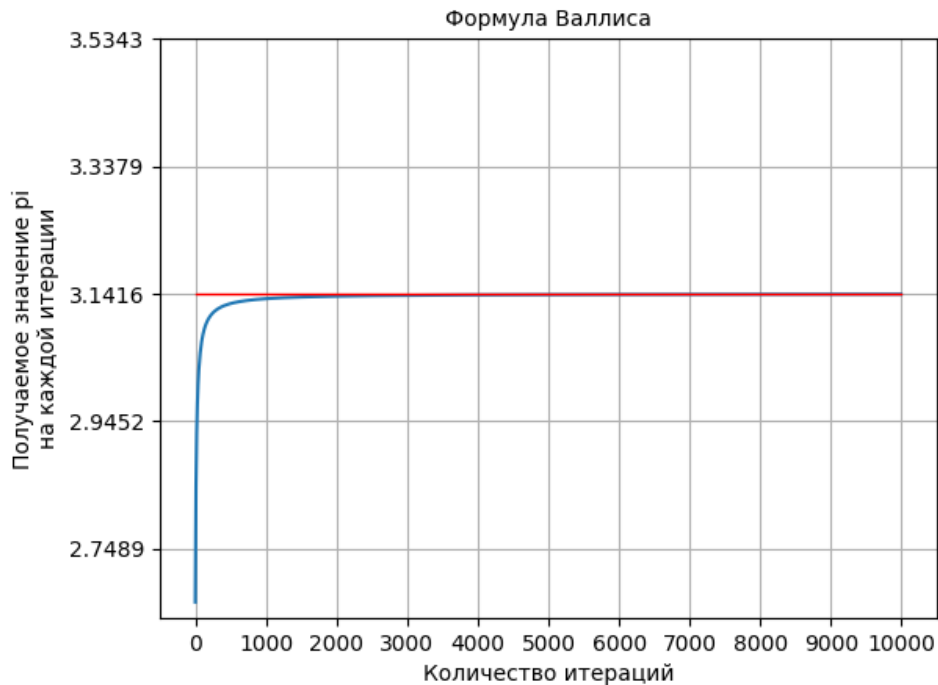


Рис. 4: Результат для формула Валлиса.

– Формула Франсуа Виета:

$$2/\pi = \frac{\sqrt{2}}{2} \cdot \frac{\sqrt{2+\sqrt{2}}}{2} \cdot \frac{\sqrt{2+\sqrt{2+\sqrt{2}}}}{2} \cdot \dots;$$

Функция, которая записывает в файл номер операции и соответствующее ей значение числа  $\pi$ , приведена ниже. По этим точкам построим график, чтобы наглядно увидеть зависимость значения на данной итерации от числа итераций.

```
void rad3 () {  
    ofstream f("3.csv", ios::out);  
    float a = 0.0, pi = 1;  
    for (int i = 0; i < 101; i++) {  
        a = sqrt(2 + a);  
        pi = pi * a/2;  
        f << i << '\t' << 2/pi << endl;  
    }  
    cout << '\n';  
};
```

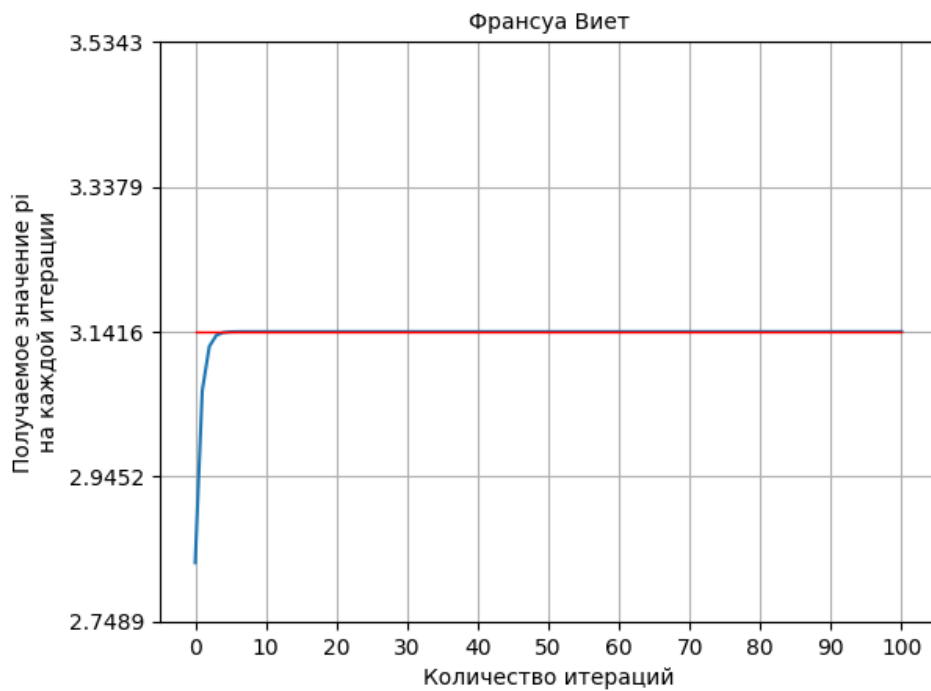


Рис. 5: Результат для Франсуа Виета.



- Колеблется вокруг точного значения, но потом сходится:

– Ряд Лейбница:

$$\frac{\pi}{4} = \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots;$$

Функция, которая записывает в файл номер операции и соответствующее ей значение числа  $\pi$ , приведена ниже. По этим точкам построим график, чтобы наглядно увидеть зависимость значения на данной итерации от числа итераций.

```
void rad2 () {
    ofstream f("2.csv", ios::out);
    float pi=0, t=1;
    for (int i=1; i < 101; i+=1) {
        int k = -1;
        for (int j = 0; j < i; j++) {
            k = k * (-1);
        }
        pi += k / t;
        f << i << '\t' << pi*4 << endl;
        t += 2;
    }
    cout << '\n';
};
```

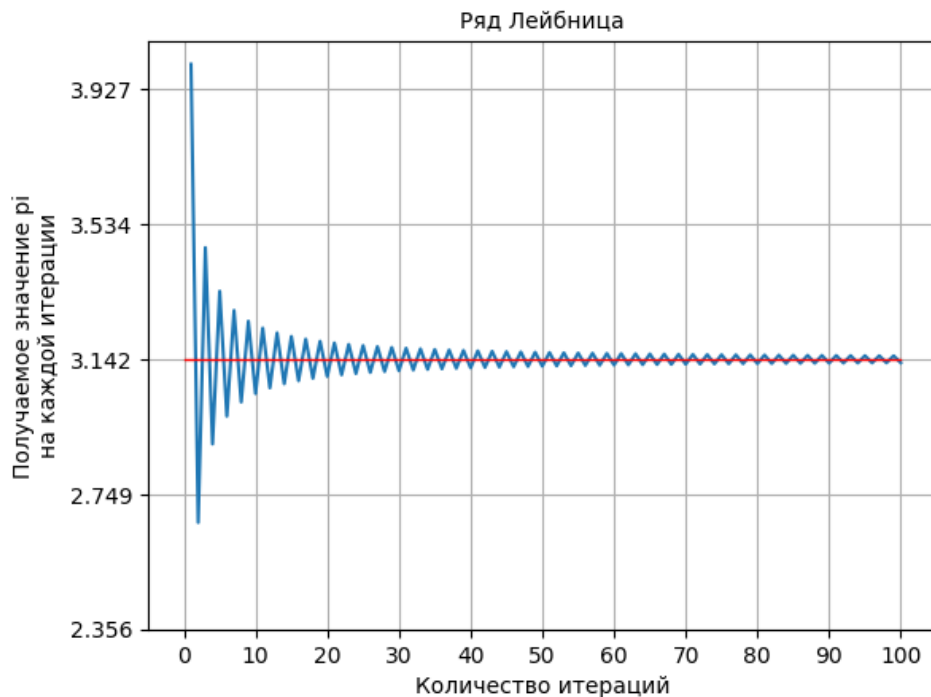


Рис. 6: Результат для ряда Лейбница.

– Преобразование Мадхава:

$$\pi = 2 \cdot \sqrt{3} \cdot \sum_{i=0}^{\infty} \frac{(-1)^i}{3^i \cdot (2 \cdot i + 1)}.$$

Функция, которая записывает в файл номер операции и соответствующее ей значение числа  $\pi$ , приведена ниже. По этим точкам построим график, чтобы наглядно увидеть зависимость значения на данной итерации от числа итераций.

```
void rad4 () {  
    ofstream f("4.csv", ios::out);  
    float pi = 0;  
    for (int i = 0; i < 101; i++) {  
        pi += pow(-1, i)/(pow(3, i)*(2*i+1));  
        f << i << '\t' << 2*sqrt(3)*pi << endl;  
    }  
    cout << '\n';  
};
```

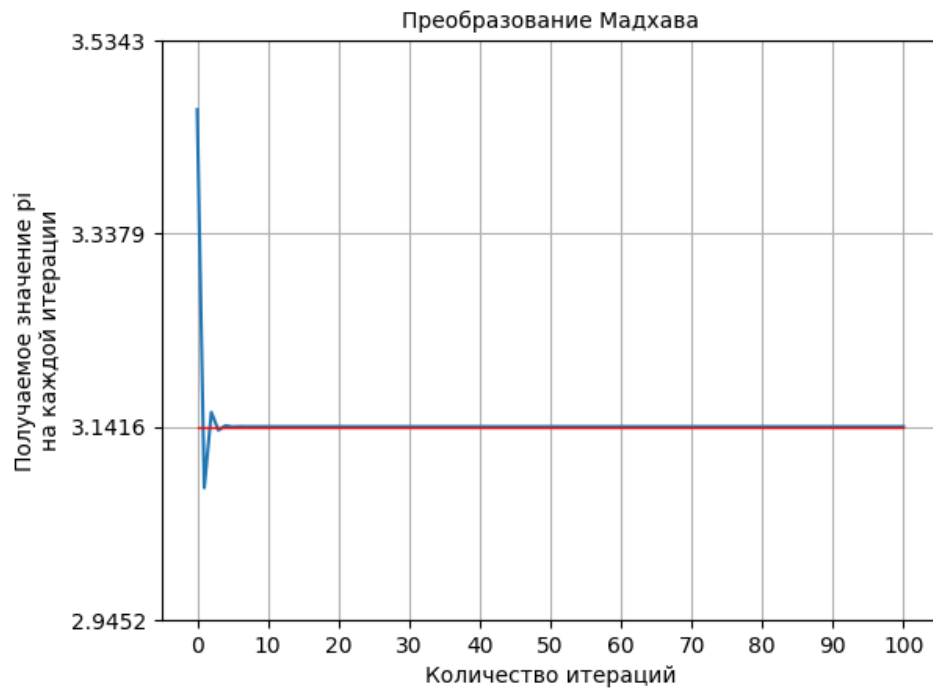


Рис. 7: Результат для преобразования Мадхава.

## Задача 5

**Цель:** Посмотрим на эффективность тех формул, которые мы нашли. Для каждой из них построим график среднего времени достижения каждого десятичного знака числа  $\pi$ .

**Ход работы:** Напишем программу, которая на каждой итерации сравнивает знаки после запятой числа, которое получается на каждой итерации наших функций из предыдущего задания, со знаками числа  $\pi$ . Программа выводит соответствующий знак и время, за которое он был получен. Например, для ряда Лейбница функция имеет вид (для других функций преобразования будут те же - на каждой итерации сравниваем определённый знак после запятой):

```
double get_time() {
    return std::chrono::duration_cast<std::chrono::microseconds>
        (std::chrono::steady_clock::now().time_since_epoch()).count()/1e3;
};

void rad2 () {
    ofstream f("1-1.csv", ios::out);
    const double PI = 3.14159265359;
    int l = 1;
    float pi=0, t=1, time_0 = get_time();
    for (int i=1; i < 100000000; i+=1) {
        int k = -1;
        for (int j = 0; j < i; j++) {
            k = k * (-1);
        }
        pi += k / t;
        for (int j = 1; j < 11; j++) {
            if (round(4*pi*pow(10, j))/pow(10, j) == round(PI*pow(10, j))/pow(10, j)) {
                cout << i << " " << j << " " << abs(get_time()-time_0) << " ms" << '\n';
                f << j << '\t' << abs(get_time()-time_0) << endl;
                l ++;
                break;
            }
        }
        t += 2;
    }
    cout << '\n';
};
```

Время работы компьютера зависит от многого, не только от числа итераций. Для приближённого значения времени проведёт серию экспериментов (например, 10) и возьмём среднее значение по времени. В таком случае оценка будет более точной.

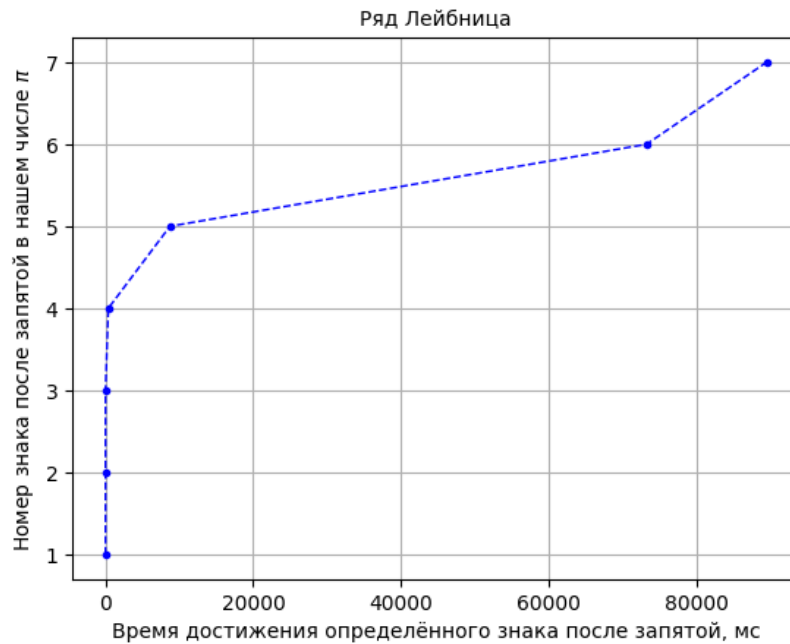


Рис. 8: Эффективность ряда Лейбница.

Видим, что по данной формуле начиная с 4 знака время достаточно большое. Построим график для первых трёх знаков - оценим их скорость достижения:

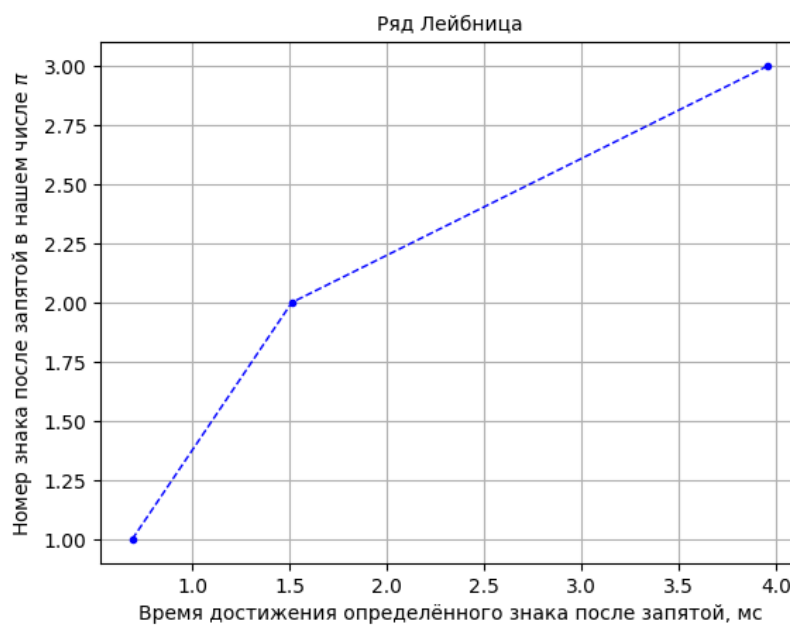


Рис. 9: Эффективность ряда Лейбница для первых знаков.

Для формулы Валлиса всё немного сложнее. Как мы видели, это сходящаяся последовательность (Рис 4.), а значит получаемое число не достигает  $\pi$ , то есть знаки после запятой могут не достигаться.

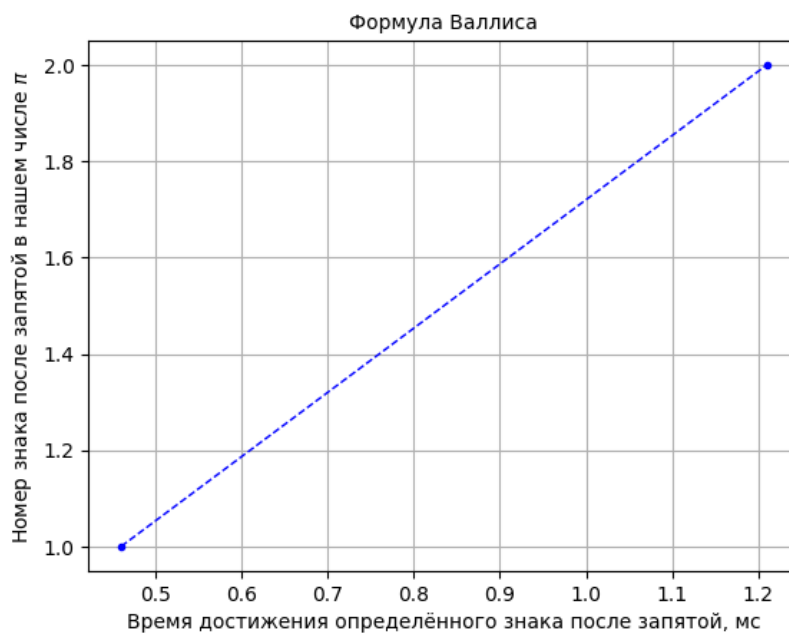


Рис. 10: Эффективность формулы Валлиса.

Однако формула Франсуа Виета, график которой (Рис. 5) тоже сходится к числу  $\pi$ , достигает знаки после запятой, в отличие от формулы Валлиса:

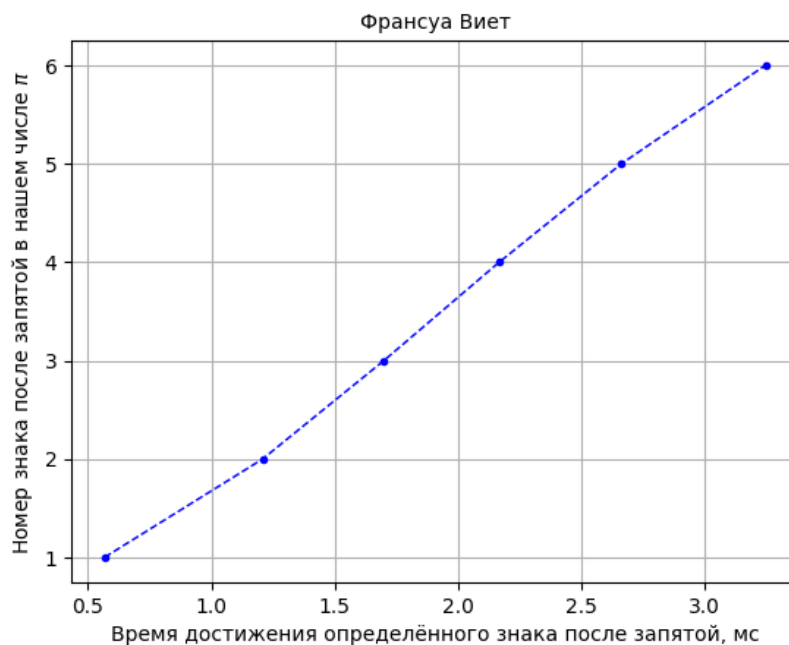


Рис. 11: Эффективность ряда Франсуа Виета.

Ну и для преобразования Мадхава график скорости имеет вид:

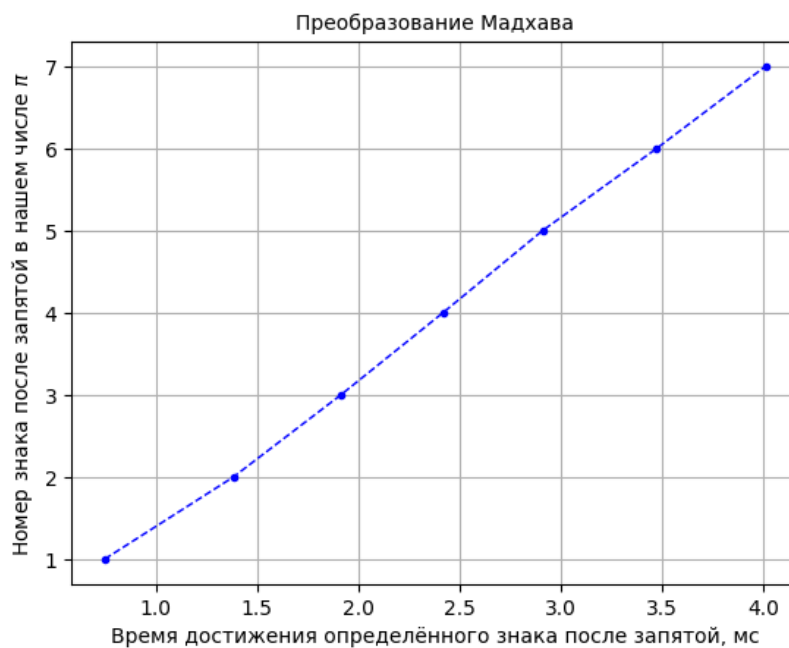


Рис. 12: Эффективность преобразования Мадхава.

Сравним эффективность этих формул, сделав наложение графиков. Так мы сможем наглядно оценить, какая формула более эффективна при достижении того или иного знака.

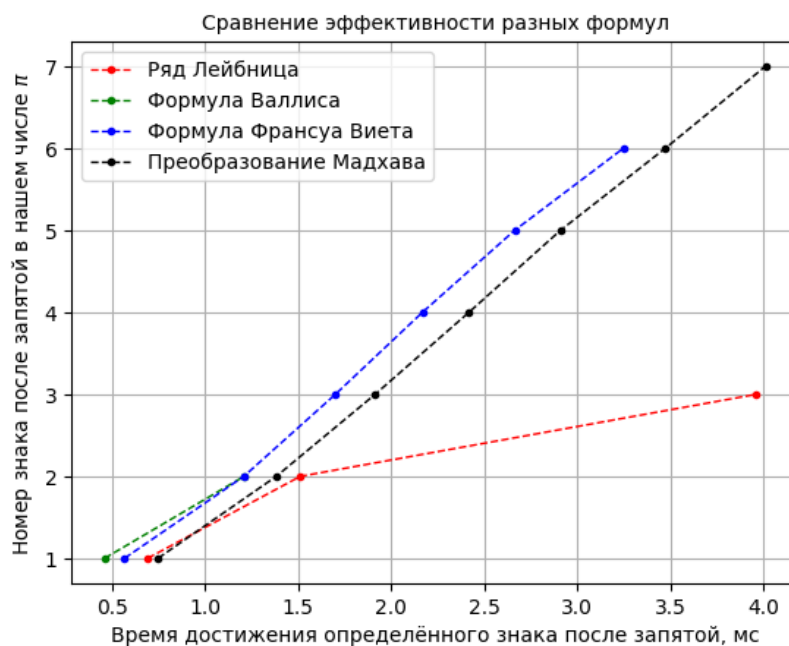


Рис. 13: Сравнение эффективности формул.

## Вывод

- Научились писать программы для перевода чисел в двоичное представление для записи в памяти компьютера, а так же посмотрели, как происходит переполнение мантиссы.
- Доказали, что float'ы расположены на разном расстоянии друг от друга - с помощью этого факта создали бесконечный цикл.
- Экспериментировали с числом  $\pi$ , проверяли итерационные формулы получения этого числа, получили зависимость числа от количества итераций для каждой формулы.
- Оценили эффективность каждой формулы. В некоторых случаях (как для формулы Валлиса) оценка получилась плохая. Это может быть связано как с самой формулой (не достигаются знаки после запятой), так и с техническими требованиями ПК.
- Сравнили эффективность разных формул.