Отчет о выполнении лабораторной работы по информатике

Хранение переменных

Студент: Копытова Виктория Сергеевна
Группа: Б03-304

1 Аннотация

Цель работы: сформировать понимание принципов работы на компьютере с действительными числами.

2 Ход работы

2.1 Вывод unsigned int в двоичной системе счисления

```
#include <iostream>
using namespace std;
void  intToBinary(unsigned int num)
{
    int i=0;
    for(int i=0;i<32;i++)
    {
        if (num == (num<<1)>>1)
            cout<<0;
        else
            cout<<1;
        num = num<<1;
    }
    cout<<'\n';
}
int main()
{
    int n;
    cin>>n;
    intToBinary(n);
    return 0;
}
```

Программа смещает число на 1 бит влево, затем на 1 бит вправо и проверяет, меняется ли число. Если меняется, на экран выводится 1, иначе – 0. После этого число снова смещается на 1 бит влево, чтобы отбросить выведенную цифру и цикл повторяется ещё 31 раз. Например для числа $5_{10} = 101_2$:

1	0	1

Смещение на 1 бит вправо:

0	1	0

Смещение на 1 бит влево:

0	0	1

$$101_2 = 5_{10}, \quad 001_2 = 1_{10}, \quad 5 \neq 1 \Rightarrow \text{на экран выводится } 1.$$

Результатом выполнения программы является не 101, а 00000000000000000000000000000101, но это позволяет использовать её для преобразования больших чисел.

2.2 Вывод float в двоичной системе счисления

```cpp
#include <iostream>
using namespace std;
union floatNumber
{
    float floatPart;
    unsigned int intPart;
};
void  intToBinary(unsigned int num)
{
    int i=0;
    for(int i=0;i<32;i++)
    {
        if (num == (num<<1)>>1)
            cout<<0;
        else
            cout<<1;
        num = num<<1;
    }
}
int main()
{
    floatNumber a;
    cin>>a.floatPart;
    intToBinary(a.intPart);
    return 0;
}
```

Для преобразования числа с плавающей точкой в двоичную систему счисления воспользуемся функцией intToBinary из предыдущего пункта, а само число запишем в тип union. Для числа 137.375 получим:

$$137.375_{10} = 01000011000010010110000000000000_2$$

2.3 Переполнение мантиссы

Напишем программу, которая выводит на экран число 10^{i}, где $i \in \mathbb{Z}, i \geqslant 0$, и его двоичную запись.

```cpp
#include <iostream>
using namespace std;
unsigned int power(unsigned int n,unsigned int base)
{
    if (base==0)
        return 1;
    int k=n;
    for (int i=0;i<base-1;i++)
        n*=k;
    return n;
}
void  intToBinary(unsigned int num)
{
    int i=0;
    for(int i=0;i<32;i++)
    {
        if (num == (num<<1)>>1)
            cout<<0;
        else
            cout<<1;
        num = num<<1;
    }
    cout<<'\n';
}
int main()
{
    cout << fixed;
    cout.precision(2);
    for (int i = 0; i<50; i++)
    {
        cout<<power(10,i)<<endl;
        intToBinary(power(10,i));
        cout<<'\n';
    }
    return 0;
}
```

Запустив программу, мы увидим, что числа большие чем 10^8 представляют собой не степень десяти. Это происходит потому, что большие числа не влезают полностью в мантиссу и часть двоичного разложения этого числа обрезается.

Результат выполнения программы для $i \geqslant 8$:

```
100000000
00000101111101011110000100000000
```

```
1000000000
00111011100110101100101000000000

1410065408
01010100000010111110010000000000

1215752192
01001000011101101110100000000000

3567587328
11010100101001010001000000000000
```

2.4 Бесконечный цикл

Рассмотрим программу:

```cpp
#include <iostream>
using namespace std;
int main()
{
    cout << fixed;
    cout.precision(2);
    for(float i=1;i<16777217;i+=1)
    {
        cout<<i<<endl;
    }
    return 0;
}
```

Мы видим, что цикл конечный, но при запуске программы через какое-то время получим такой результат:

```
16777209.00
16777210.00
16777211.00
16777212.00
16777213.00
16777214.00
16777215.00
16777216.00
16777216.00
16777216.00
16777216.00
16777216.00
16777216.00
```

```
16777216.00
16777216.00
16777216.00
```

Цикл получается бесконечным. Это связано с тем, что переменные типа float при увеличении своего значения начинают стоять все дальше и дальше друг от друга (из-за дискретности диапазона) и рано или поздно расстояние между ними становится больше единицы. То есть при прибавлении единицы к большому числу мы получаем то же самое число.

2.5 Расчёт числа π

Для расчёта числа π восполльзуемся несколькими итерационными формулами.

1. Формула Лейбница.

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots$$

```
#include <iostream>
using namespace std;
float pi(long long int n)
{
    float sum = 0.0;
    int sign = 1;
    for (long long int i = 0; i < n; ++i)
    {
        sum += sign/(2.0*i+1.0);
        sign *= -1;
    }
    return 4.0*sum;
}
int main()
{
    cout << fixed;
    cout.precision(7);
    for(long long int i=0;i<10000;i++)
    {
        cout<<pi(i)<<end;
    }
    return 0;
}
```

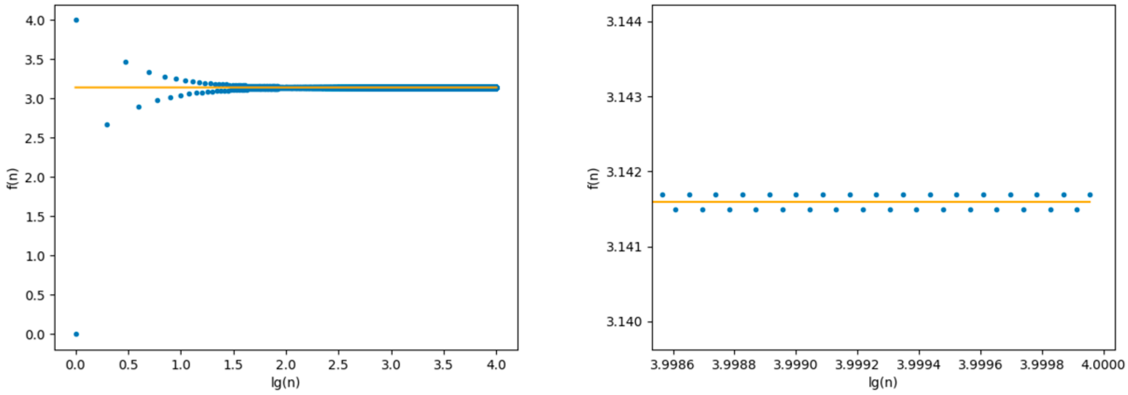

Рис. 1: Зависимость результата расчёта от числа итераций для формулы Лейбница (здесь и далее n – число итераций, оранжевая линия – табличное значение числа pi для сравнения)

Мы видим, что расчитанное значение числа π флуктуирует около истинного значения. Попробуем увеличить количество итераций:

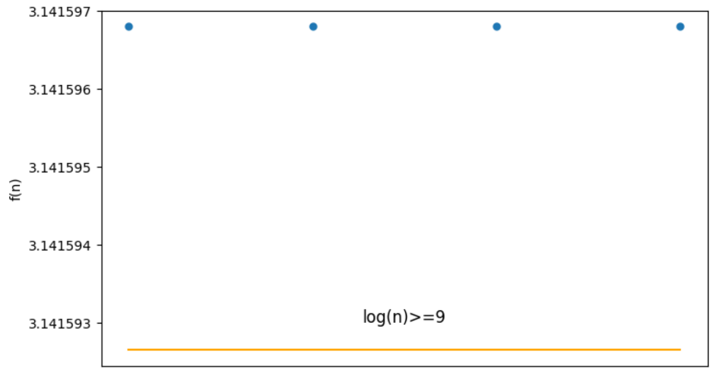

Рис. 2: Формула Лейбница при большом количестве итераций

Из-за возникающих погрешностей расчитанные значения не сходятся к истинному.

2. Площадь под графиком единичной окружности. Расчитаем площадь под графиком $x^2 + y^2 = 1$ с помощью правила трапеции:

$$\int_a^b f(x)dx = \frac{b-a}{2n}(f(x_1)+2f(x_2)+2f(x_3)+2f(x_3)+...+2f(x_n)+2f(x_n+1)).$$

```
#include <iostream>
#include <math.h>
```

```
using namespace std;
float f(float x)
{
    return sqrt(1-pow(x,2));
}
int main()
{
    cout<<fixed;
    cout.precision(7);
    float integration;
    integration = f(0) + f(1);
    for (long long int n=1; n<=10000; n++)
    {
        for (int i=1; i<n; i++)
        {
            integration += 2*f(static_cast<float>(i)/n);
        }
        integration = integration/(2*n);
        cout<<integration*4<<endl;
    }
    return 0;
}
```

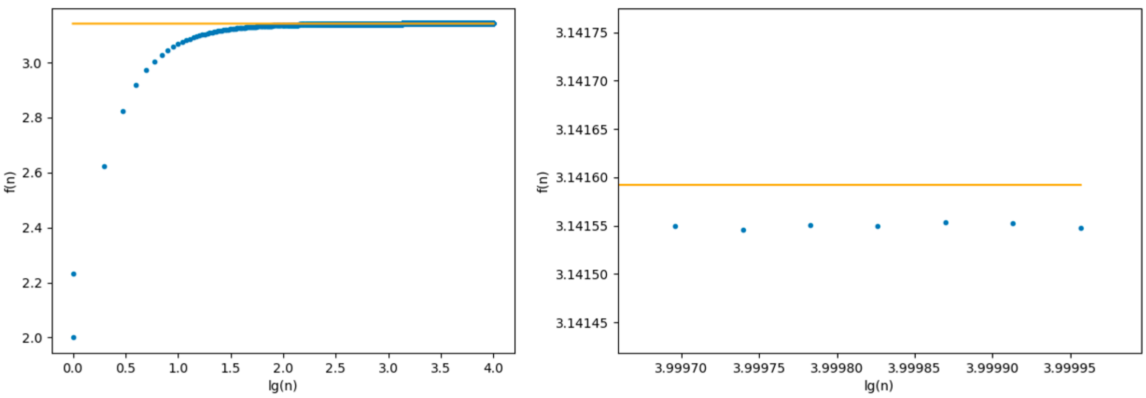

Рис. 3: Расчёт площади под графиком единичной окружности

Увеличим количество итераций:

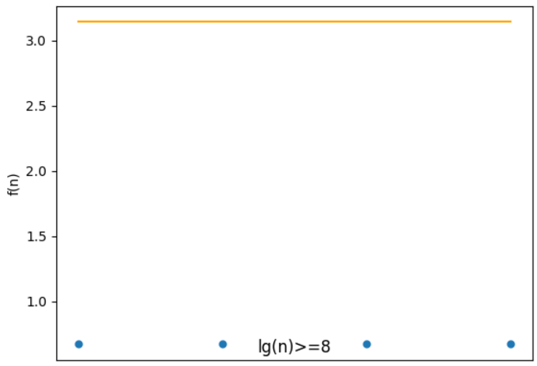

Рис. 4: Формула площади при большом количестве итераций

Полученные значения сильно отличаются от истинных.

3. Ряд обратных квадратов

$$\frac{\pi^2}{6} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2} + \dots$$

```
#include <iostream>
#include <math.h>
using namespace std;
float piCalc(float n)
{
    float pi, ans, ans1, a = 0;
    for (float i=1;i<n;i++)
    {
        a=i*i;
        ans+=1/a;
    }
    ans1=6*ans;
    pi=sqrt(ans1);
    return pi;
}
int main()
{
    cout<<fixed;
    cout.precision(7);
    for(float i=0.0;i<10000.0;i++)
    {
        cout<<piCalc(i)<<endl;
    }
    return 0;
}
```

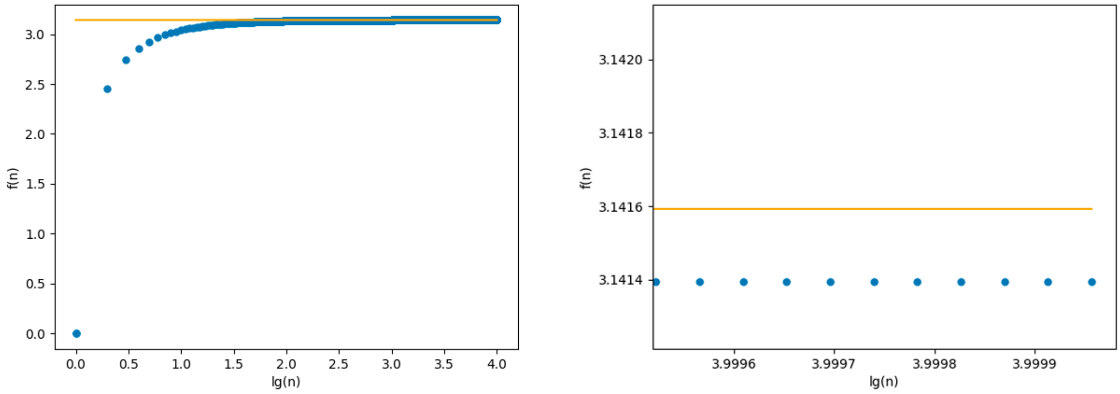

Рис. 5: Ряд обратных квадратов

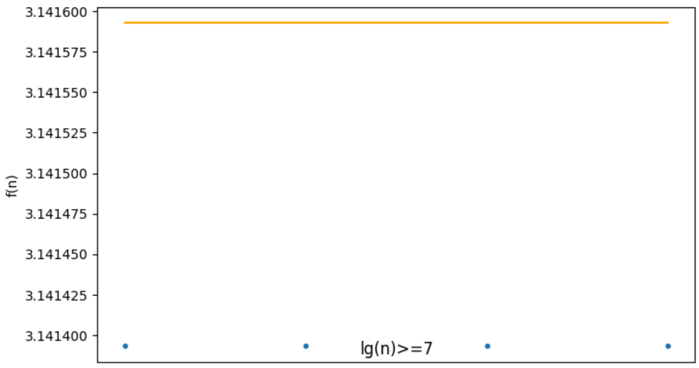

Рис. 6: Ряд обратных квадратов для большого количества итераций

4. Формула Мэчина и ряд Тэйлора

```
#include <iostream>
#include <math.h>
using namespace std;
float xCalc(float x, long unsigned int n)
{
    float result =0.0;
    int sign = 1;
    long unsigned int k=1;
    for (long unsigned int i = 1; i<=n;i++)
    {
        result+=sign*pow(x,k)/k;
        k+=2;
        sign*=(-1);
```

```
    }
    return result;
}
int main()
{
    cout<<fixed;
    cout.precision(7);
    for(long unsigned int i=0;i<10000;i++)
    {
        cout<<4.0*(4.0*xCalc(0.2,i)-xCalc(0.0041841,i))<<endl;
    }
    return 0;
}
```

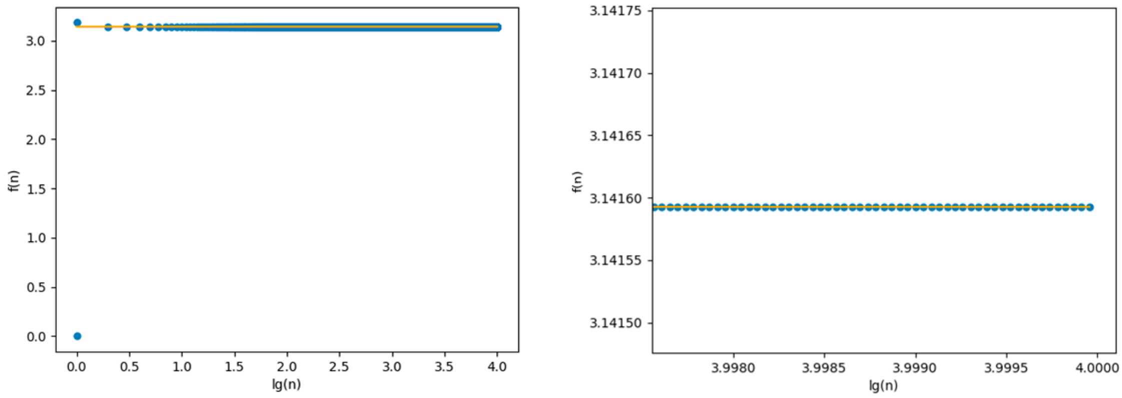

Рис. 7: Формула Мэчина

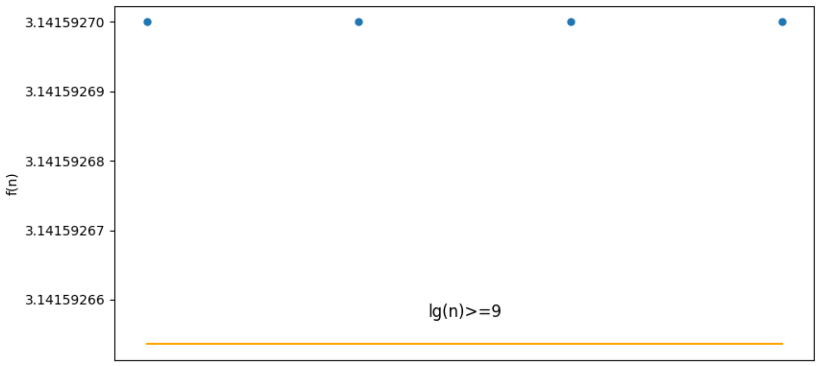

Рис. 8: Формула Мэчина при большом количестве итераций

Формула Мэчина сходится к нужному значению быстрее и ближе всех ранее рассмотренных формул.

2.6 Время десятичных знаков числа π

Расчитаем сколько нужно времени, чтобы правильно вычислить десятичные знаки числа π по приведённым выше формулам. Пример кода для формулы Лейбница:

```
#include <iostream>
#include <chrono>
using namespace std;
float pi(long long int n) {
    float sum = 0.0;
    int sign = 1;
    for (long long int i = 0; i < n; ++i)
    {
        sum += sign/(2.0*i+1.0);
        sign *= -1;
    }
    return 4.0*sum;
}
int main()
{
    cout<<fixed;
    cout.precision(10);
    auto start = chrono::steady_clock::now();
    for(long long int i=0;i<100000000;i++)
    {

        float p=pi(i);
        if (p>=3.14 && p<3.15)
        {
            break;
        }
    }
    auto end = chrono::steady_clock::now();
    auto elapsed = chrono::duration_cast<chrono::microseconds>(end - start);
    double time = (double)(elapsed.count());
    cout<<time;
}
```

Проведём 10 измерений для каждого десятичного знака и расчитаем средние занчения, данные занесём в таблицу 1. (Процессор Intel Core i7 12700H 2,3 ГГц; ноутбук подключён к питанию)

	Время, мкс			
Достигаемое значение	Формула Лейбница	Площадь единичной окружности	Ряд обратных квадратов	Фомула Мэчина
3,1		8	2810	2
3,14	45	1180	62343	2
3,141	9954	7351	160780	2,4
3,1415	363567	237264	значение не достигается	2,5
3,14159	81724660	24342907	значение не достигается	2,5
3,141592	198065308	28501394	значение не достигается	3
3,1415926	значение не достигается	значение не достигается	значение не достигается	значение не достигается

Таблица 1: Время вычисления числа π

Мы видим, что формула Мэчина работает быстрее всех, но ни один из методов не даёт результата с точностью выше 6 знаков после запятой. Именно такая точность характерна для типа float.

3 Вывод

При использовании компьютера для вычислений с действительными числами нужно учитывать дискретность диапазона и ограниченность выделяемой на число памяти. Например, при работе с большими числами типа float появляются ошибки.

Также должа быть учтена максимальная точность каждого типа переменных, чтобы погрешность была минимальна.