Домашнее задание по предмету "Архитектура компьютеров"

Выполнил студент ИУ9-12
Иванов Георгий

Этап 1: разработка работоспособной функции интегрирования

Задача:
Разработать подпрограмму численного интегрирования методом средних прямоугольников.

Начальные условия:
Подынтегральная функция: $f(x) = 1/x$
Интервал интегрирования: $x \in [\pi; 10000]$

Решение задачи

```
#define _USE_MATH_DEFINES
#include <stdio.h>
#include <math.h>
#include <locale.h>

float f (float x) {
   return 1/x;
}

float F (float x) {
   return log (x);
}
float Integral( float Left,float Right, long N, float (*func)(float)) {
   float i, dx;
   float res = 0.0;
   dx = (Right - Left) / N;
   for (i = Left; i < Right; i+=dx)
      res += f(i+(dx/2));
   res *= dx;
   return res;
}

int main()
{
   long n;
   float L = M_PI, R = 10000;
   float V, V0 = F( R ) - F( L );
   setlocale( LC_ALL, "" );
   printf("Steps;Error\n");
   for (n = 1; n < 100; n += 1) {
      V = Integral( L, R, n, f );
      printf( "%ld;=%.15f", n, (V-V0)/V0 );
      printf("\n");
   }
   return 0;
}
```

Более подробно про функцию численного интегрирования. Подается функции для вычисления ее интеграла: три аргумента и сама подынтегральная функция. Первоначально найдем мы шаг dx, который равен (right-left)/N, где right -правая граница интегрирования, left — левая граница интегрирования, N — количество шагов. Наш метод, заключающийся в замене подынтегральной функции на многочлен нулевой степени, то есть константу, на каждом элементарном отрезке. Если рассмотреть график подынтегральной функции, то метод будет заключаться в приближённом вычислении площади под графиком суммированием площадей конечного числа прямоугольников, ширина которых будет определяться расстоянием между соответствующими соседними узлами интегрирования, а высота— значением подынтегральной функции в этих узлах
(рис 1) и (рис 1.б)

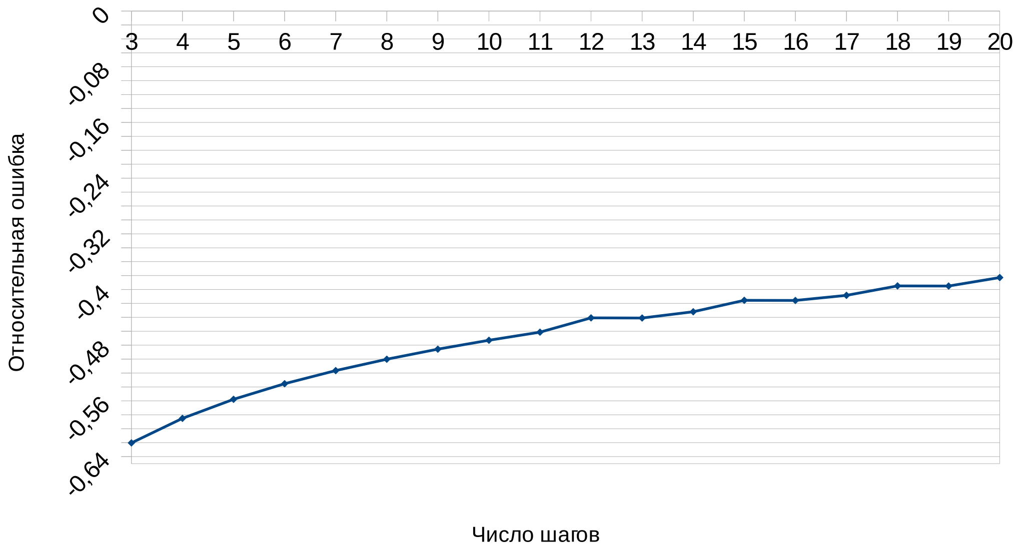

Рисунок 1. График относительной ошибки (при N=3..20 шагов)

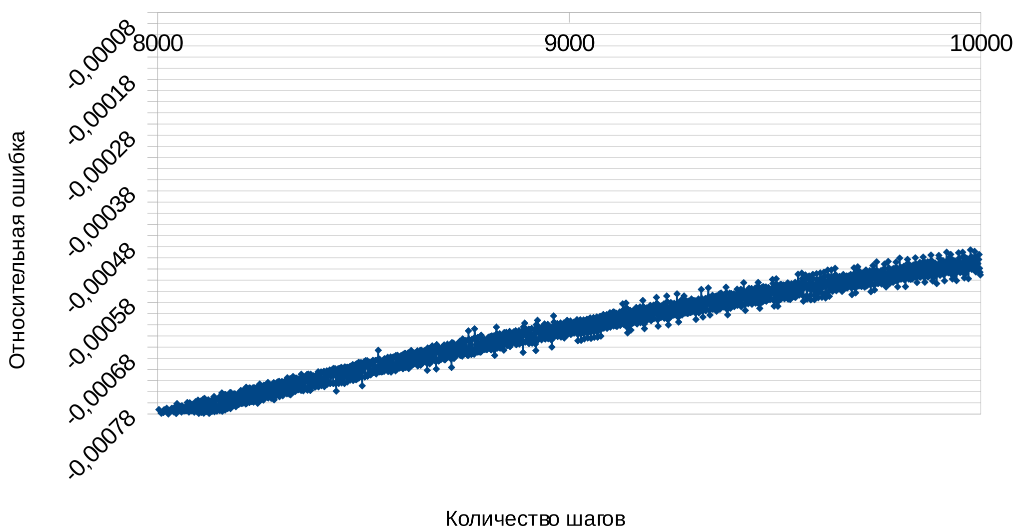

Рисунок 1(b). График относительной ошибки (при N=8000..10000 шагов)

Этап 2. Выдвижение гипотезы

Ошибка возникает при каждой итерации цикла (x+=dx). Если же возникает ошибка при каждой итерации цикла, а значит можно предположить то, что наша функция численного интегрирования, переходит за границы интегрирования.

$$res = (a_1 + a_2)(1 \pm e)$$

Действительно, значение i после всех шагов должно быть равно точно правой границе (то есть 10000), а у нас при выполнении программы значение i равно (при N=101 шагов) равно 10000,008789. Эта ошибка может возникать при том, что погрешность накапливается именно при итеративном прибавление к x dx: (рис. 2)

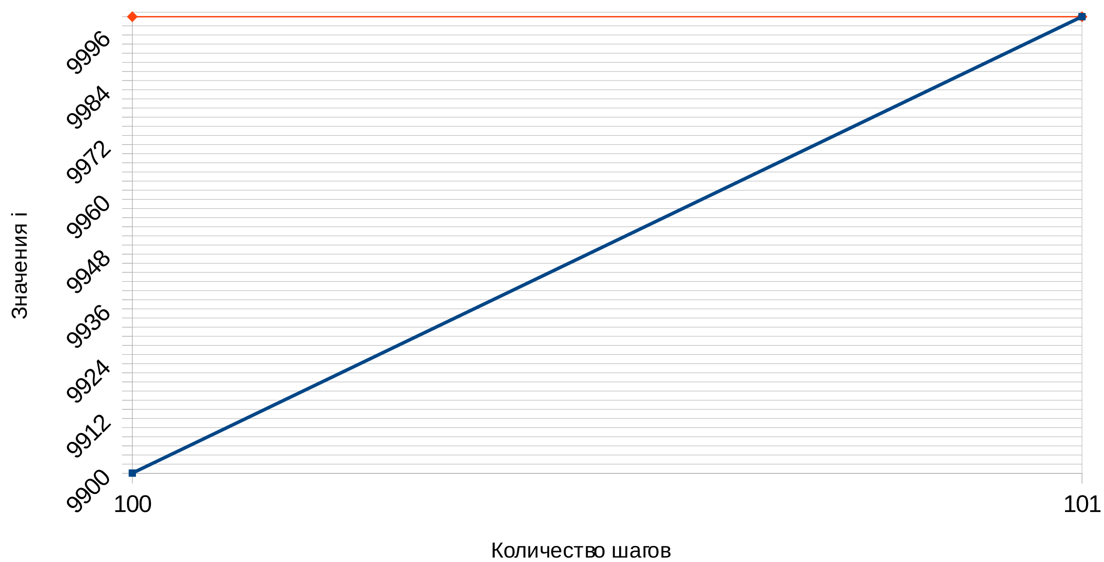

Рисунок 2. Значения i (на последних 2 шагах итераций цикла при N=101)

Действительно,

$$x_0 = L$$
$$x_1 = (L + dx) * (1 \pm e) = L * (1 \pm e) + dx * (1 \pm e)$$
$$x_2 = (L + dx) * (1 \pm e)^2 + dx(1 \pm e) = L * (1 \pm 2 * e) + dx * (2 \pm (1 + 2) * e)$$
$$x_3 = (L + dx) * (1 \pm e)^3 + dx(1 \pm e)^2 + dx(1 \pm e) = L * (1 \pm 3 * e) + dx * (3 \pm 6 * e)$$
$$...$$
$$x_i = (L+dx)*(1\pm e)^i + dx(1\pm e)^{i-1} + ... + dx(1\pm e)^2 + dx(1\pm e) = L*(1\pm i*e) + dx*(i \pm (1+2+3+...+i-1+i)*e)$$

Получается для нашего случая (где $dx = \frac{(R-L)}{N}$)

$$x_i = L * (1 \pm i * e) + \frac{(R-L)}{N} * (i \pm \frac{(i*(i+1))}{2} * e)$$

Для i=N (то есть последнее слагаемое):

$$x_N = L * (1 \pm N * e) + \frac{(R-L)}{N} * (i \pm \frac{(N*(N+1))}{2} * e) = L * (1 \pm N * e) + (R - L) * (1 \pm \frac{(N+1)}{2} * e)$$
$$x_N = L * \pm L * Ne + R - L \pm e(R - L)\frac{(N+1)}{2} = R \pm e * (R\frac{(N+1)}{2} + L\frac{(N-1)}{2})$$

Этап 3: Устранение ошибки при накоплении погрешности

Возможны два случая устранения ошибки: 1) правильная отработка последнего шага
Пример исходного кода:

```
typedef T float;
 T Integral(T Left,T Right, long N, T(*func)(T)) {
   T i, dx, help;
   T res = 0.0;
   dx = (Right - Left) / N;
   for (i = Left; i < Right;i+=dx,N-=1) {
                    if (N==1) {
                                   res *= dx;
                                   help = i+(Right-i)/2;
                                   res += f(help)*(Right-i);
                                   i=Right;
                                   break;
                    }
                    help = i+(dx/2);
                    res += f(help);
       }
   return res;
}
```

Как мы заметим, при последнем шаге у нас происходит в программе то, что последний прямоугольник вычисляется по формуле:

$$square = f(i + \frac{(R-i)}{2}) * (R - i),$$

то есть среднее значение функции в точке $i + (R - i)$, где i — текущее значение (перед последней итерации цикла) , R — правая граница интегрирования. Теперь при запуске программы при любых N шагов наша функция не будет переходить за пределы интегрирования.
Результат выполнения нашей программы: (рис. 3)

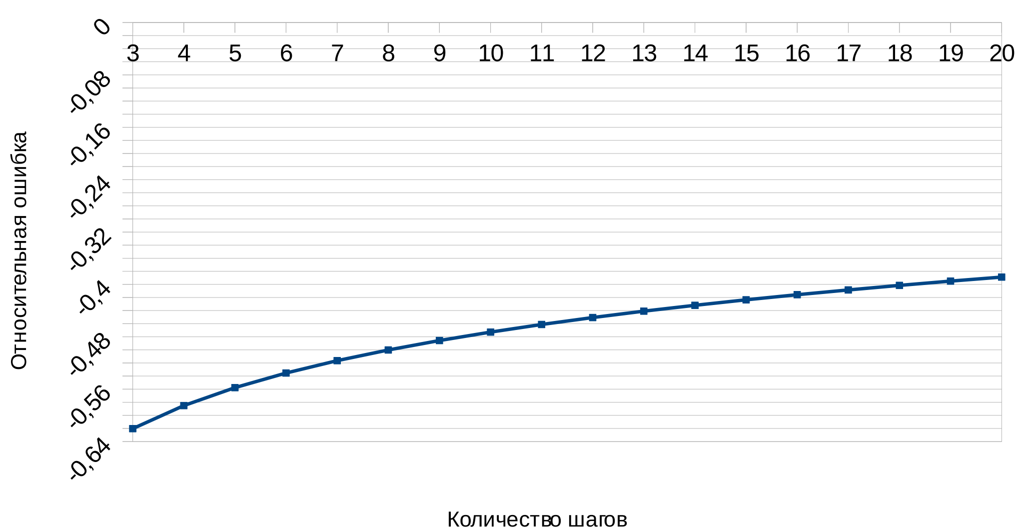

Рисунок 3. График относительной ошибки (при обработке точно последнего шага)

Как можем заметить, то улучшений в программе практически нельзя заметить (если только, график стал более выпуклым, то есть при N=12, N=13 -значения изменились и некоторых других шагах). Но отметим то, что у нас специально(!) обрабатывается последний шаг, то есть при изменении пределов интегрирования на обратные (это значит мы запустим функцию интегрирования в обратном направлении). Пример исходного кода:

```
typedef float T;

T Integral1(T Left,T Right, long N, T(*func)(T)) {
    T i, dx, help;
    T res = 0.0;
    dx = (Right - Left) / N;
    for (i = Right; i > Left;i-=dx,N-=1) {
            if (N==1) {
                    res *= dx;
                    help = i+(i-Left)/2;
                    res += f(help)*(i-Left);
                    i=Left;
                    break;
            }
                    help = i-(dx/2);
            res += f(help);
        }
    return res;
}
int main()
{
    long n;
    T L = 10000, R = M_PI;
    T V, V0 = F(L) - F(R);
    setlocale( LC_ALL, "" );
    printf("Steps;Error\n");
    for (n = 1;n <100; n += 1) {
        V = Integral1( L, R, n, f );
        printf( "%ld;=%.15f", n, (V-V0)/V0 );
        printf("\n");
    }
        return 0;
}
```

Результат выполнения нашей программы:(рис.4)

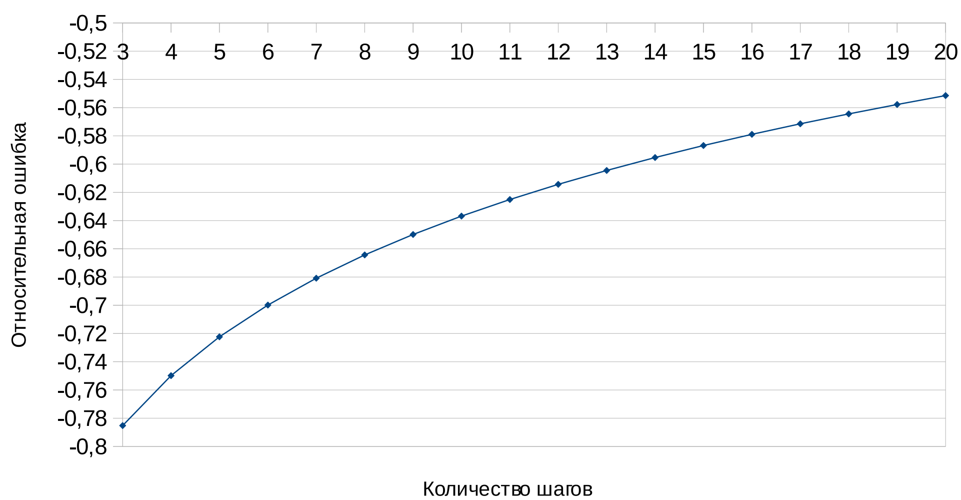

Рисунок 4. График относительной ошибки (выполнение функции интегрирования в обратном направлении, т.е с правой границы интегрирования до левой)

Можно сделать вывод о том, что эта функция не будет универсальной в любом направлении (по условию задания нужно написать «универсальную программу, работающая корректно при любых изменениях в пределах), а значит была допущена ошибка.

2) Рассмотрим такой метод вычисления интеграла, при котором dx вычисляется не до цикла и не равен , а на каждой итерации цикла как

Пример исходного кода:

```
T Integral( T Left,T Right, long N, T (*func)(T)) {
	T x=Left, dx,i, res = 0.0;
	long j;
	for (j = 0; j < N;j++) {
				dx=(Right - x)/(N-j);
				i=x;
				x+=dx;
				res += f((i+x)/2)*dx;
		}
	return res;
}
```

Результат выполнения нашей программы: (рис.5)

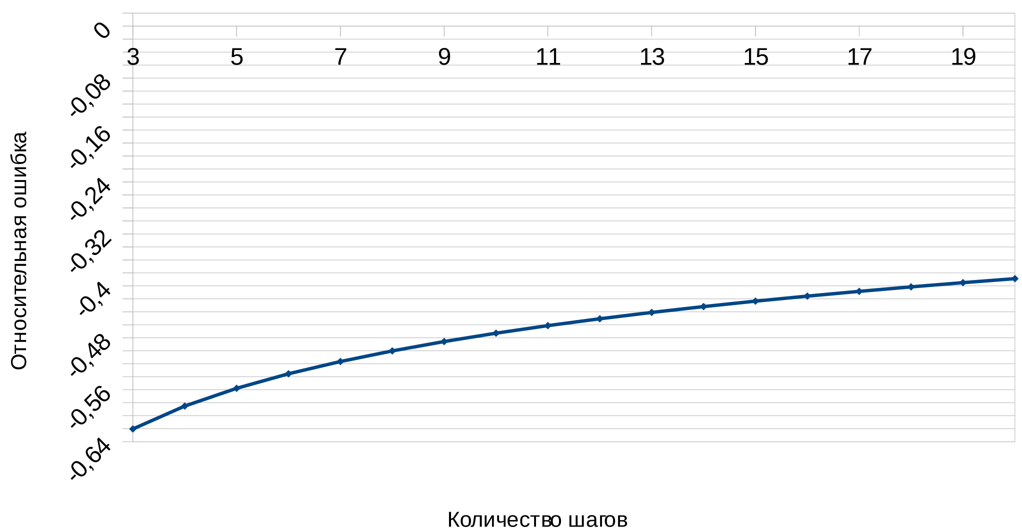

Рисунок 5. График относительной ошибки.

Как можем заметить, то улучшений в программе опять же практически заметить (если только, график стал более выпуклым, то есть при N=12, N=13 -значения изменились и некоторых других шагах относительно первоначальной программы). Но в отличие от прошлых программ, в этой программе происходит вычисление интеграла не происходит за границами интегрирования(так как последняя итерация цикла даст значение dx равным расстоянию от предпоследнего x до Right). То есть при любом запуске программы и любых ввода данных (границ интегрирования) наша функция будет точно вычислять в пределах интегрирования.

Этап 4: Уменьшение относительной ошибки

Воспользуемся методом суммирования Кохена: *если перед нами стоит задача нахождения суммы большого множества чисел, то для достижения данной цели лучше всего использовать метод, известный под названием Kahan Summation. В данном методе коррекция промежуточной суммы производится на протяжении всей работы алгоритма.* Пример исходного кода:

```
T Integral( T Left,T Right, long N, T (*func)(T)) {
        T x=Left, dx, help,i;
        T res = 0.0,res1=0.0,res2=0.0;
                T correction = 0.0;
                dx = (Right-Left)/N;
                res = f((x+dx)/2)*dx;
                x+=dx;
        long j;
        for (j = 1; j < N;j++) {
                                dx=(Right - x)/(N-j);
                                i=x;
                                x+=dx;
                                res1 = f((i+x)/2)*dx-correction;
                                res2 = res+res1;
                                correction = (res2 - res) - res1;
                                res=res2;
                        }
    }
    return res;
}
```

Результат выполнения программы:

Рисунок 5. График относительной ошибки.

Как можем заметить, каждый член вначале корректируется согласно ошибке, накопившаяся на предыдущих этапах. Затем новая сумма вычисляется путем суммирования скорректированного члена и промежуточной суммы. После этого вычисляется поправочный член как разница между изменением в сумме и прибавленного скорректированного члена.

Проверим, этот метод на более маленьком отрезке.

```
T Integral( T Left,T Right, long N, T (*func)(T)) {
        T x=Left, dx, help,i;
        T res = 0.0,res1=0.0,res2=0.0;
                T correction = 0.0;
                dx = (Right-Left)/N;
                res = f((x+dx)/2)*dx;
                x+=dx;
        long j;
        for (j = 1; j < N;j++) {
                                dx=(Right - x)/(N-j);
                                i=x;
                                x+=dx;
                                res1 = f((i+x)/2)*dx-correction;
                                res2 = res+res1;
                                correction = (res2 - res) - res1;
                                res=res2;
                }
    }
    return res;
}


int main()
{
    long n;
    T L = 0.5;
    T R = 0.5+100*1E-6;
    T V, V0 = F( R ) - F( L );
    setlocale( LC_ALL, "" );
    printf("Steps;Error\n");
    for (n = 1; n < 100; n += 1) {
        V = Integral( L, R, n, f );
        printf( "%ld;=%f", n, (V-V0)/V0 );
        printf("\n");
    }
return 0;
}
```

Результат выполнения программы: (рис. 6)

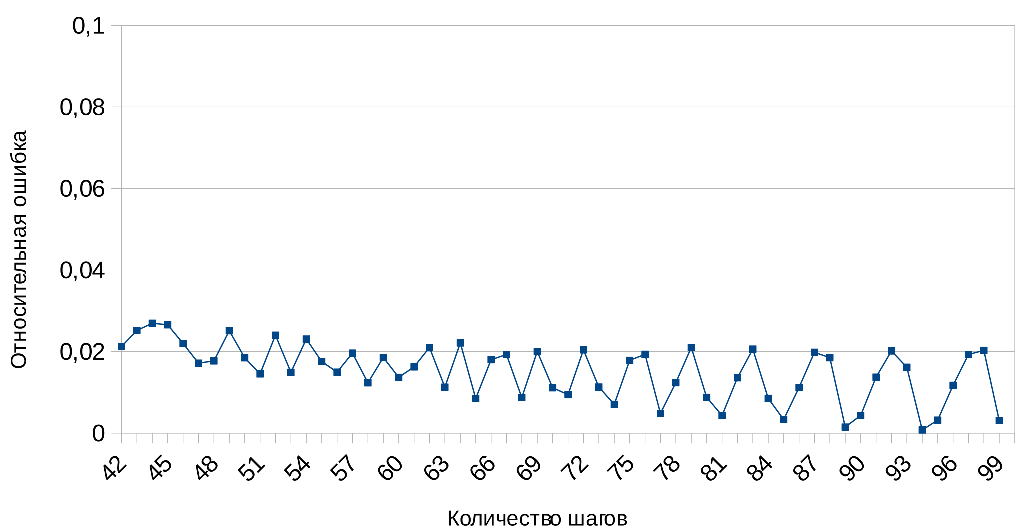

Рисунок 6. График относительной ошибки(при N=42..100)

Можно заметить, при N=47 и далее наблюдаются скачки погрешности. Возможно, это ошибка возникает с вычислением прямоугольника (то есть ширины). Попробуем:

```
T Integral( T Left,T Right, long N, T (*func)(T)) {
       T x=Left, dx, help,i;
       T res = 0.0,res1=0.0,res2=0.0;
              T correction = 0.0;
              dx = (Right-Left)/N;
              res = f((x+dx)/2)*dx;
              x+=dx;
       long j;
       for (j = 1; j < N;j++) {
                            dx=(Right - x)/(N-j);
                            i=x;
                            x+=dx;
                            res1 = f((i+x)/2)*(x-i)-correction;
                            res2 = res+res1;
                            correction = (res2 - res) - res1;
                            res=res2;
                     }
   }
   return res;
}

int main()
{
   long n;
   T L = 0.5;
   T R = 0.5+100*1E-6;
   T V, V0 = F( R ) - F( L );
   setlocale( LC_ALL, "" );
   printf("Steps;Error\n");
   for (n = 1; n < 100; n += 1) {
       V = Integral( L, R, n, f );
       printf( "%ld;=%f", n, (V-V0)/V0 );
       printf("\n");
   }
return 0;
}
```

(изменение в строке $res1 = f((i + x)/2 * (x - i) - correction)$

Результат выполнения нашей программы:
До: (рис. 7)

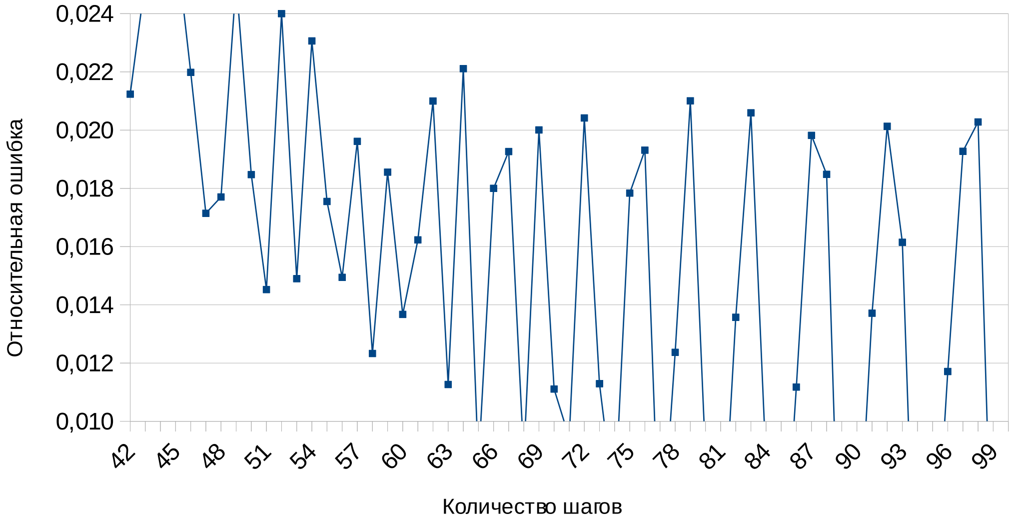

Рисунок 7. График относительной ошибки(при N=42..100)

После:(рис. 8)

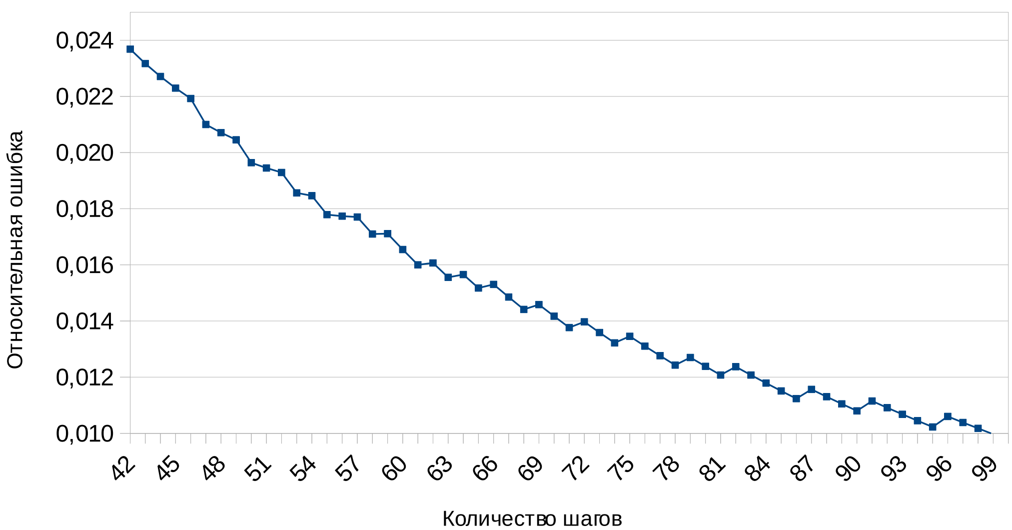

Рисунок 8. График относительной ошибки(при N=42..100)

Здесь можно заметить, что мы вычисляем значение прямоугольника не через высоту $(h = dx)$, а через высоту $(h = x_i - x_{i-1})$. В этом случае площадь вычисляется более точнее.
Уменьшение относительной ошибки (суммирование с компенсацией): Пример исходной программы:

```
void twoSum(float a, float b, float s, float t) {
        s = a + b;
        float a1 = s - b;
        float b1 = s - a;
        float da = a - a1;
        float db = b - b1;
        t = da + db;
}

T Integral( T Left,T Right, long N, T (*func)(T)) {
        T x=Left, dx,i;
        T res = 0.0,res1=0.0,res2=0.0;
        T correction = 0.0;
                T help =0;
        dx = (Right-Left)/N;
        res = f((x+dx)/2)*dx;
        x+=dx;
        long j;
        for (j = 1; j < N;j++) {
                T y;
                dx=(Right - x)/(N-j);
                i=x;
                x+=dx;
                res1 = f((i+x)/2)*(x-i)-correction;
                res2 = res+res1;
                correction = (res2 - res) - res1;
                                twoSum(res,res2,res,help);
                res=res2;
        }
        return res;
}
```

Результат выполнения нашей программы: (рис. 9)

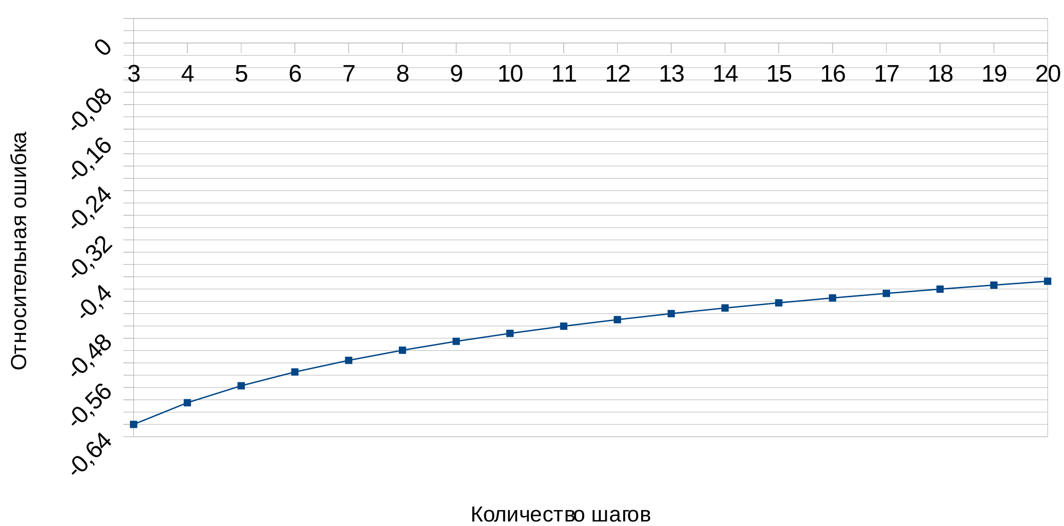

Рисунок 9. График относительной ошибки(при N=3..20)

Можно заметить, для получения погрешности операции суммирования используется операция 2Sum. Она принимает на вход нормальные числа, возвращает числа , такие, что является корректно округленным (округленным в соответствии со стандартом) результатом операции (сложение в смысле операций с плавающей запятой) и выполняется точное равенство.

Этап 5: Погрешность при сложении площадей

Погрешность при сложении площадей в переменной res так же, как и в переменной x при сложении, обусловлена точностью выполнения операции. Т.е. на каждой итерации появляется погрешность e . Пусть мы складываем некоторую сумму a_i, где a_i- площадь прямоугольника i:

$$res = a_1$$
$$res = (a_1 + a_2)(1 \pm e)$$
$$res = (a_1 + a_2) * (1 \pm e)^2 + a_3(1 \pm e)$$
$$res = (a_1 + a_2) * (1 \pm e)^{(N-1)} + a_3(1 \pm e)^{(N-2)} + a_4(1 \pm e)^{(N-3)} + ... + a_N(1 \pm e)$$
$$res = (a_1 + a_2)(1 \pm (N-1)e) + a_3(1 \pm (N-2)e) + ... + a_N(1 \pm e)$$

Значит, большая погрешность у первых аргументов. Однако если первые слагаемые будут достаточно малыми, то влияние этой погрешности будет минимально. То есть нужно проинтегрировать в обратную сторону, чтобы каждый последующий аргумент (прямоугольник) был меньше предыдущего
Пример исходного кода: (результат выполнения (рис. 10))

```
 Integral( T Left,T Right, long N, T (*func)(T)) {
        T x=Right, dx,i;
        T res = 0.0,res1=0.0,res2=0.0;
        T correction = 0.0;
                T help =0;
        dx = (Right-Left)/N;
        res = f(x-dx/2)*dx;
        x-=dx;
        long j;
        for (j = 1; j < N;j++) {
                dx=(x-Left)/(N-j);
                i=x;
                x-=dx;
                res1 = f((i+x)/2)*(i-x)-correction;
                res2 = res+res1;
                correction = (res2 - res) - res1;
                                twoSum(res,res2,res,help);
                res=res2;
        }
        return res;
}
```

Рисунок 10. График относительной ошибки(при N=3..20)

Можно заметить, что ничего не изменилось, хоть мы поменяли границы интегрирования на обратные (то есть поменяли местами и запустили функцию в обратную сторону). А значит мы добились того, что наша программа универсальна, которая работает в обе стороны, независимо от пределов интегрирования (не переходя за их границы).

Этап 6: Проверка

Попробуем заменить нашу функцию на e^x:

```
typedef float T;
T f (T x) {
    return exp(x);
}

T F (T x) {
    return exp(x);
}
int main()
{
    long n;
    T L = 10^(-14);
    T R = 10^(-16);
    T V, V0 = F( R ) - F( L );
    setlocale( LC_ALL, "" );
    printf("Steps;Errors\n");
    for (n = 1; n < 1000; n += 1) {
        V = Integral( L, R, n, f );
        printf( "%ld;=%f", n, (V-V0)/V0 );
        printf("\n");
    }
return 0;
}
```

Результат выполнения нашей программы (рис. 11)

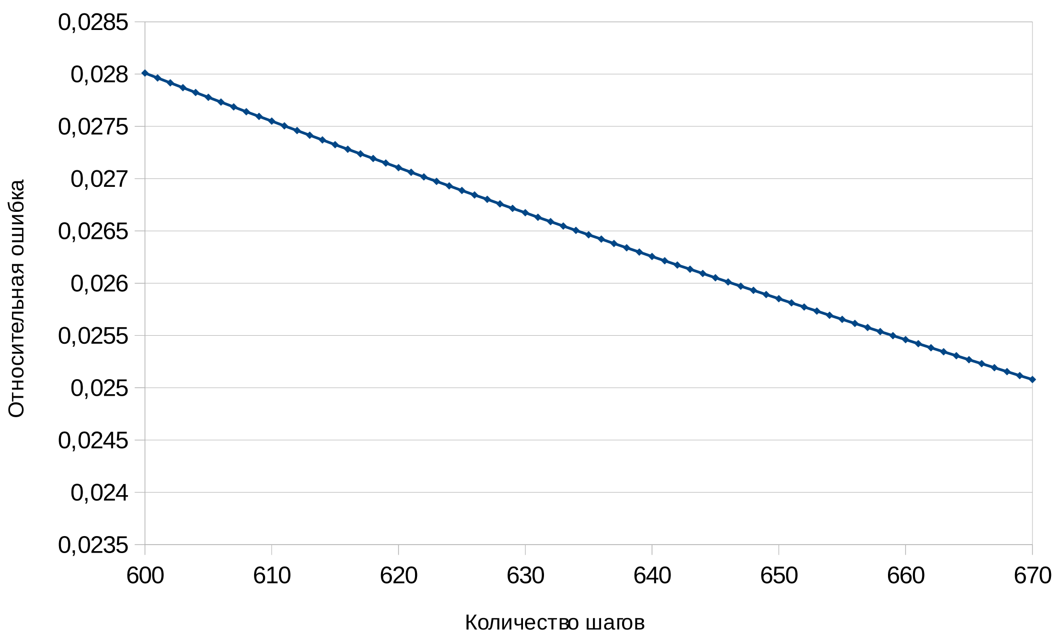

Рисунок 11. График относительной ошибки(при N=600..670)

Оглавление