МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФГБОУ ВО АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Институт цифровых технологий, электроники и физики

Кафедра вычислительной техники и электроники (ВТиЭ)

Лабораторная работа № 004

Решение задач численными методами с использованием циклов

Выполнил студент 595 гр.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. В. Лаптев

Проверил: к.т.н,, доцент каф. ВТиЭ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ю.Г. Скурыдин

Лабораторная работа защищена

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020 г.

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Барнаул 2020

**ЗАДАНИЕ №1.**

**1. Формулировка задачи**

Создать программу для вычисления функции с точностью до 0,0001. Считать, что требуемая точность достигнута, если вычислена сумма нескольких первых слагаемых, и очередное слагаемое оказывается по модулю меньше, чем 0,0001. В таком случае данное слагаемое, а также все последующие слагаемые при расчёте суммы учитываться не должны. Стандартную функцию для вычисления синуса не использовать.

**2. Постановка задачи**

На вход программы поступает вещественное значение переменной x. Программа определяет сумму элементов сходящегося числового ряда с заданной точностью. На выход программы поступает вещественное число, которое является значением функции .

**3. Математическая модель**

С клавиатуры вводится вещественное число x. И первоначальное значение функции становится равным этому числу. Затем организуется цикл с проверкой условия соответствия значения функции заданной степени точности, внутри которого организуется еще один цикл для вычисления факториала и происходит вычисление значения функции с постоянным увеличением точности за счёт прибавления к полученному значению функции нового значения последнего члена и образования обновлённого значения функции. Данную математическую модель можно рассмотреть на примере. С клавиатуры вводится значение 0.5 (значение х). Переменной у присваивается значение 0.5. Затем проверяется условие (0.5 0.0001), которое является необходимым для входа в цикл. Если это условие не выполняется, то происходит выход из цикла и округление полученного значения у до заданной степени точности и вывод соответствующего значения на экран. Иначе происходит вычисление факториала, необходимого порядка, с помощью которого вычисляется член ряда нужного порядка (в данном случае вычисляется 3 и член ряда равен: -0.53/3 = -0.0208). Затем это значение прибавляется к полученному значению у и образуется новое значение у (у = 0.5 - 0.53/3 = 0.4792). Т.к. полученное значение члена ряда по модулю больше, чем 0.0001, то вычисляется следующее значение члена (в данном случае вычисляется 5 и член ряда равен: 0.55/5 = 0.0002). Затем это значение прибавляется к полученному значению у и образуется новое значение у (у = 0.4792 + 0.55/5 = 0,4794). И таким образом цикл работает до тех пор, пока заданная степень точности не будет больше, чем новое значение и не произойдёт выход из цикла.

**4. Описание алгоритма**

Начало

4.1 Ввод переменной х

4.2 Присвоение переменной у значения переменной х, задания начального значения суммы (sum = 1), счётчика знакочередования (i = -1), степени точности ( = 0.0001), начальной степени переменной х (n = 1) и факториала (fac = 1, k)

Начало цикла

4.3 Проверка условия . Если условие не выполняется, то переход к пункту 4.11

4.4 Присвоение факториалу и числу нового начального значения в начале каждой итерации цикла

Начало цикла

4.5 Проверка условия (k <= n). Если условие не выполняется, то осуществляется выход из цикла и переход к пункту 4.8

4.6 k++

4.7 Вычисление факториала числа k и переход к пункту 4.5

Окончание работы внутреннего цикла

4.8 Вычисление очередного члена сходящегося числового ряда

4.9 Вычисление общего значения функции

4.10 Смена знака, для вычисления следующего члена числового ряда и переход к пункту 4.3

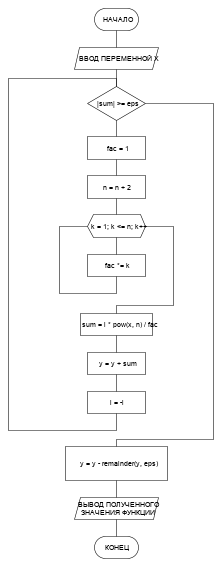
Окончание работы основного цикла

4.11 Процесс округления полученной суммы до степени точности, равной

4.12 Вывод полученной суммы на экран

Конец

**5. Опорный граф (блок-схема) алгоритма**



**6. Проект программы с определением замкнутых программных единиц и необходимых структур данных**

В данном случае программа должна состоять из трех частей – блок ввода, блок вывода и блок преобразований, который включает в себя два цикла и выполнение математических преобразований.

**7. Текст программы:**

**на языке С++**

#include <iostream>;

#include <math.h>;

#include <cmath>;

using namespace std;

int main()

{

int i = -1, k;

double x, y, eps = 0.0001, sum = 1, n = 1, fac = 1;

setlocale(0, "");

cout << "Введите значение x: ";

cin >> x; // ВВод переменной х

y = x;

while (abs(sum) >= eps) // Цикл для вычисления значения функции

{

fac = 1;

n = n + 2;

for (k = 1; k <= n; k++) // Цикл для вычисления значения факториала

fac \*= k;

sum = i \* pow(x, n) / fac; // Вычисление очередного члена последовательности

y = y + sum; // Вычисление значения функции

i = -i;

}

cout << "y = " << y - remainder(y, eps); // Округление значения и вывод его на экран

return 0;

}

**на языке Pascal**

**Program** lab\_4\_1;

**Var** x, y, sum, eps, k, fac:real;

i, n, j, m:integer;

**Begin**

Writeln ('Введите значение x: ');

Readln (x); // Ввод переменной х

sum := x;

eps := 0.0001;

n := 3;

i := -1;

y := x;

fac := 1;

**for** j := 1 **to** 6 **do** // Цикл, который отвечает за определение количества знаков после запятой в данном числе эпсилон, чтобы в дальнейшем,

**begin** // чтобы в дальнейшем округлить полученную сумму членов последовательности до необходимой точности

k := round(eps, j); // Определение количества знаков после запятой в числе k, путём добавления к числу 1 знака после запятой в каждой итерации цикла

**if** (k = eps) **then** // Условие, при котором будет найдено точное количество знаков после запятой

**break**; // (т.е. когда число знаков после запятой в числе k будет равно количеству знаков после запятой в числе эпсилон)и осуществится выход из цикла

**end**;

**while** abs(sum) >= eps **do** // Цикл для вычисления значения функции

**begin**

fac := 1;

**for** m := 1 **to** n **do** // Цикл для вычисления факториала

fac := fac \* m;

sum := i \* power(x, n) / fac; // Вычисление очередного члена последовательности

y := y + sum; // Вычисление общего значения функции

n := n + 2;

i := -i;

**end**;

Writeln ('y = ', round(y, j)); // Округление результата и вывод ответа

**End**.

**8. Проверка работоспособности (тестирование) программы**

Для проверки работоспособности программы введём число 3.14 на консоль и на выходе (после округления) должно быть выведено число 0.0016.

**9. Сравнительный анализ и оценка эффективности работы программ на разных языках программирования**

В данной задаче, программы написанные на разных языках программирования несколько отличны по структуре друг от друга. Например, в программе на С++ округление значения происходит уже в конце работы программы, а в Паскале нужен дополнительный цикл, для определения количества знаков после запятой, для округления полученной далее суммы, который прописывается ранее. Благодаря чему программа написанная на языке С++ будет выполняться быстрее и гораздо эффективнее, чем на Паскале.

**ЗАДАНИЕ №2.**

**1. Формулировка задачи**

По формуле Симпсона и методом прямоугольников вычислить приближённое значение интеграла . Точность не превышает 0.001. Сравнить методы вычислений по количеству итераций.

**2. Постановка задачи**

На вход программы поступает целое положительное число n, которым задаётся шаг разбиения данного интеграла. Далее, программа вычисляет приближённое значение интеграла с заданной точностью по методу Симпсона и методом прямоугольников. На выход программы вещественными положительными числами поступает значение интеграла, с учётом степени точности, затем подсчитывается количество итераций для каждого метода и выводится текстовое сообщение о том, какой метод короче.

**3. Математическая модель**

Математическую модель данной задачи можно рассмотреть на следующем примере. На вход программы поступает число n = 20, с помощью которого определяется шаг интегрирования (h). Далее осуществляется проверка условия соответствия погрешности для входа в основной цикл и, с помощью внутреннего цикла, вычисляется значение интеграла методом прямоугольников. По формуле: sum1 += h \* f1(a + h \* (y - 0.5)), вычисляется у-е значение суммы, которое прибавляется к полученному ранее значению (т.е. sum1 = 0, и после преобразований sum1 = 0 + 0.00499791693 = 0.00499791693), происходит увеличение счётчика цикла у на 1. Затем также находятся остальные значения параметра sum1, пока не будет достигнута необходимая степень точности. В данном случае: новое значение sum1 = 0.00499791693 + 0.0149438132 = 0.0199417302 и т.д. После того как найдено конечное значение sum1, оно присваивается переменной integ1 и округляется до степени точности, eps. В данном примере: 0.31102… округляется до 3 знаков после запятой: 0.311. И на выходе программы полученное число выводится в качестве ответа и также выводится количество итераций, которое потребовалось для решения интеграла (число у). Похожим образом организуется работа циклов для нахождения интеграла по методу Симпсона. Отличие заключается лишь в наличие ещё одного цикла, который к полученному значению суммы, каждую итерацию, прибавляет ещё одно значение sum, которое вычисляется отдельно для четной и отдельно для нечетной итерации этого цикла. Эти значения также суммируются и по формуле: sum \*= h / 3, определяется конечное значение интеграла, которое присваивается переменной integ и округляется до степени точности eps (в данном случае integ = 0.318) и выводится количество итераций. Такая погрешность в вычислении является допустимой, т.к. интеграл вычисляется лишь приближённо.

**4. Описание алгоритма**

Начало

4.1 Создание функции подинтегрального выражения для метода прямоугольников и для метода Симпсона

4.2 Инициализация переменных a = 0, b = 1, integ = o, integ1 = 0, n = 20, h, eps = 0.001, sum1 = 0, sum = 0, x, y

4.3 Вычисление числа разбиений интеграла (шага интегрирования)

Начало цикла (метод прямоугольников)

4.4 Проверка условия sum1 <= eps. Если условие не выполняется, то переход к пункту 4.8

Начало внутреннего цикла

4.5 Проверка условия y <= n. Если условие не выполняется, то переход к пункту 4.8

4.6 у++

4.7 Вычисление нового значения sum1 для у-ой итерации цикла и её суммирование с предыдущим значением переменной

Окончание работы внутреннего цикла

Окончание работы цикла

4.8 Присвоение переменной integ1 значения sum1

4.9 Округление интеграла до заданной степени точности и вывод значения на экран

4.10 Вывод количества итераций на экран

Начало цикла (метод Симпсона)

4.11 Проверка условия sum <= eps. Если условие не выполняется, то переход к пункту 4.21

Начало внутреннего цикла

4.12 Проверка условия x <= n / 4. Если условие не выполняется, то переход к пункту 4.15

4.13 х++

4.14 Вычисление нового значения sum для х-ой итерации цикла и её суммирование с предыдущим значением переменной

Окончание работы внутреннего цикла

Начало 2 внутреннего цикла

4.15 Проверка условия х <= n. Если условие не выполняется, то переход к пункту 4.20

4.16 х++

4.17 Проверка условия x % 2 == 0. Если условие не выполняется, то переход к пункту 4.19

4.18 Вычисление нового значения суммы для чётного шага и переход к пункту 4.20

4.19 Вычисление нового значения суммы для нечётного шага

Окончание работы 2 внутреннего цикла

4.20 Вычисление конечного значения общей суммы интеграла

Окончание работы цикла

4.21 Присвоение значению integ значения sum

4.22 Округление до степени точности и вывод значения интеграла на экран

4.23 Вывод на экран количества итераций

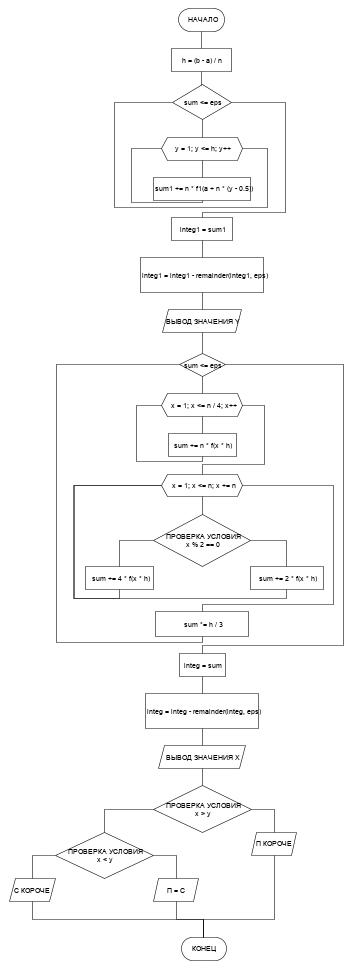
4.24 Проверка условия х > у. Если условие выполняется: Метод Симпсона короче метода прямоугольников и завершение работы программы

4.25 Проверка условия х < у. Если условие выполняется: Метод прямоугольников короче метода Симпсона и завершение работы программы

4.26 х = у. Вывод ответа: Метод Симпсона равен методу прямоугольников

Конец

**5. Опорный граф (блок-схема) алгоритма**

****

**6. Проект программы с определением замкнутых программных единиц и необходимых структур данных**

В данном случае программа должна состоять из четырёх частей – блок ввода, блок вывода и блок преобразований, включающий цикл, в котором присутствует проверка условия на вхождение и который содержит в себе ещё один цикл для нахождения площади фигуры под графиком интеграла (криволинейной трапеции или прямоугольника, в зависимости от случая), на каждый шаг разбиения (который присутствует отдельно для метода Симпсона и отдельно для метода прямоугольников). В качестве структурных данных могут быть введены переменные для хранения исходного числа и остальных необходимых переменных.

**7. Текст программы:**

**на языке С++**

#include <iostream>;

#include <math.h>;

#include <cmath>;

using namespace std;

double f(double x)

{

return (sin(x) / (2 \* sqrt(x)));

}

double f1(double y)

{

return (sin(y) / (2 \* sqrt(y)));

}

int main()

{

int x, y;

double a = 0, b = 1, integ = 0, n = 20, h, sum = 0, eps = 0.001, sum1 = 0, integ1 = 0;

h = (b - a) / n;

// Метод прямоугольников

while (sum1 <= eps)

{

for (y = 1; y <= n; y++)

sum1 += h \* f1(h \* (y - 0.5));

}

integ1 = sum1;

cout << "Metodom pryamoygolnikov: " << integ1 - remainder(integ1, eps) << endl; // Функция отвечает за округление полученного выражения до заданной степени точности, отнимая от полученного значения модуль остатка значений, которые находятся в скобках

cout << "Kol-vo iteracy: " << y << endl;

cout << "" << endl;

// Метод Симпсона

while (sum <= eps)

{

for (x = 1; x <= n / 4; x++)

sum += n \* f(x \* h);

for (x = 1; x <= n; x += n)

{

if (x % 2 == 0)

sum += 2 \* f(x \* h);

else

sum += 4 \* f(x \* h);

}

sum \*= h / 3;

}

integ = sum;

cout << "Metodom Simpsona: " << integ - remainder(integ, eps) << endl;

cout << "Kol-vo iteracy: " << x << endl;

cout << "" << endl;

if (x > y)

cout << "Metod Simpsona dlinnee metoda pryamoygolnikov" << endl;

else if (x < y)

cout << "Metod pryamoygolnikov dlinnee metoda Simpsona" << endl;

else

cout << "Metod Simpsona raven metodu pryamoygolnikov" << endl;

return 0;

}

**на языке Pascal**

**Program** lab\_4\_2;

**Var** integ1, integ, eps, k, a, b, sum, sum1, h:real;

j, x, y, n:integer;

**function** f(x: real): real;

**begin**

f := (sin(x) / (2 \* sqrt(x)));

**end**;

**function** f1(y: real): real;

**begin**

f1 := (sin(y) / (2 \* sqrt(y)));

**end**;

**Begin**

sum := 0;

eps := 0.001;

sum1 := 0;

integ1 := 0;

a := 0;

b := 1;

integ := 0;

n := 20;

**for** j := 1 **to** 3 **do** // Цикл, который отвечает за определение количества знаков после запятой в данном числе эпсилон, чтобы в дальнейшем,

**begin** // чтобы в дальнейшем округлить полученную сумму членов последовательности до необходимой точности

k := round(eps, j); // Определение количества знаков после запятой в числе k, путём добавления к числу 1 знака после запятой в каждой итерации цикла

**if** (k = eps) **then** // Условие, при котором будет найдено точное количество знаков после запятой

**break**; // (т.е. когда число знаков после запятой в числе k будет равно количеству знаков после запятой в числе эпсилон)и осуществится выход из цикла

**end**;

h := (b - a) / n;

// Метод прямоугольников

**while** sum1 <= eps **do**

**begin**

y := 1;

**while** y <= n **do**

**begin**

sum1 := sum1 + h \* f1(h \* (y - 0.5));

y := y + 1;

**end**;

**end**;

integ1 := sum1;

Writeln ('Metodom pryamoygolnikov: ', round(integ1, j));

Writeln ('Kol-vo iteracy: ', y);

Writeln ('');

// Метод Симпсона

**while** sum <= eps **do**

**begin**

x := 1;

**while** x <= n / 4 **do**

**begin**

sum := sum + n \* f(x \* h);

x := x + 1;

**end**;

x := 1;

**while** x < n **do**

**begin**

**if** (x **mod** 2 = 0) **then**

sum := sum + 2 \* f(x \* h)

**else**

sum := sum + 4 \* f(x \* h);

x := x + n;

**end**;

sum := sum \* h / 3;

**end**;

integ := sum;

Writeln ('Metodom Simpsona: ', round(integ, j));

Writeln ('Kol-vo iteracy: ', x);

Writeln ('');

**if** (x < y) **then**

Writeln ('Metod Simpsona koroche metoda pryamoygolnikov')

**else if** (x > y) **then**

Writeln ('Metod pryamoygolnikov koroche metoda Simpsona')

**else**

Writeln ('Metod Simpsona raven metodu pryamoygolnikov');

**End**.

**8. Проверка работоспособности (тестирование) программы**

Для проверки работоспособности по запросу (после запуска программы) введем число: n = 0,1. Тогда результатом выполнения программы будет число: 0.311, 21 итераций (метод прямоугольников), 0.318, 21 итераций (метод Симпсона); Метод Симпсона равен методу прямоугольников.

**9. Сравнительный анализ и оценка эффективности работы программ на разных языках программирования**

В данной задаче, программы написанные на разных языках программирования несколько отличны по структуре друг от друга. Например, в программе на С++ округление значения происходит уже в конце работы программы, а в Паскале нужен дополнительный цикл, для определения количества знаков после запятой, для округления полученной далее суммы, который прописывается ранее, а также цикл для нахождения интеграла по методу Симпсона в С++ работает эффективнее, чем тот же цикл на Паскале. Благодаря чему программа написанная на языке С++ будет выполняться быстрее и гораздо эффективнее, чем на Паскале.

Контрольные вопросы:

Почему в методе Симпсона исходный отрезок разбивается только на четное число мелких отрезков?

Ответ: потому что метод Симпсона использует параболы при вычислении интеграла, а параболы содержат минимум 3 точки которые можно разделить максимум на 2 интервала. 2 это четное число, соответственно разбиения могут быть только черными.

Что называется приближением? Почему первое приближение в большинстве случаев не позволяет получить значение интеграла с высокой точностью? При каких условиях возможно получение точного решения уже в первом приближении?

Ответ: приближение - это вычисление определенного интеграла с использованием какого-либо из его пределов интегрирования путем нахождения площади фигуры под графиком вблизи этого предела.

Потому что первое приближение вычисляет значение интеграла вблизи первого предела интегрирования, а это не всегда точный результат, так как с увеличением интервала интегрирования возрастает и погрешность вычисления. Поэтому для достижения большей точности необходимо вычислить интеграл вблизи 2 предела интегрирования (во втором приближении).

Я думаю, что это возможно в случае, если искомый интеграл либо имеет слишком маленький интервал интегрирования, то есть если пределы интегрирования отличаются лишь в тысячные или сотые доли, либо когда необходимо найти интеграл от несложной функции, где понятно поведение графика функции на плоскости.

Какой процесс называется итерационным?

Ответ: любой процесс, в котором необходимо повторение действий (цикл) с постоянным увеличением точности искомого результатам проверкой условия на выход из цикла

Что нужно изменить в программах, если степень точности будет повышена до 0,0001?

Ответ: шаг разбиения надо увеличить, можно увеличить количество приближений и само условие входа в цикл.