**Министерство образования и науки Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**Псковский государственный университет**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Контрольная работа**

по дисциплине

**«Вычислительная математика»**

Выполнил студент

группы 1022-03

Ковалевский Р.А.

Проверил: Трофимов В.М.

Дата проверки:\_\_\_\_\_\_\_\_

**Псков**

**2018**

Оглавление

[ЛАБОРАТОРНАЯ РАБОТА № 1 3](#_Toc516532229)

[ЛАБОРАТОРНАЯ РАБОТА № 2 10](#_Toc516532230)

[ЛАБОРАТОРНАЯ РАБОТА № 3 15](#_Toc516532231)

[ЛАБОРАТОРНАЯ РАБОТА № 4 20](#_Toc516532233)

[ЛАБОРАТОРНАЯ РАБОТА №8 25](#_Toc516532234)

# ЛАБОРАТОРНАЯ РАБОТА № 1

**Метод половинного деления**

Для применения этого метода необходимо, чтобы функция на отрезке [a, b] содержала только один корень. При нахождении этого корня отрезок [a, b] делится пополам, то есть выбирается начальное приближение x = (a+b)/2, затем выбирается тот из отрезков [a, x] или [x, b], на концах которого функция f (x) имеет противоположенные знаки. Новый отрезок снова делится пополам и так далее, до тех пор, пока длина отрезка, на концах которого функция имеет противоположенные знаки, не будет меньше заданного числа ε. Блок-схема метода приведена на рис. 2.



Рис. 2. Блок-схема алгоритма метода половинного деления.

Варианты заданий приведены в табл. 1. Для всех заданий принять точность вычислений ε = 10-5.

Таблица 1. Варианты заданий к лабораторной работе №1

|  |  |  |  |
| --- | --- | --- | --- |
| № вар. | Уравнение | Отрезок, содержа-щий корень | Метод численного решения |
| 15 |  | [1;2] | половинного деления |

**Блок схема уравнения**



**Текст программы С + +**

**#include <iostream>**

**#include <cmath>**

**#include <windows.h>**

**using namespace std;**

**double f(double x)**

**{**

**double f;**

**return f=cos(x)-exp(-x\*x/2)+x-1;**

**}**

**int main()**

**{**

**SetConsoleCP(1251);**

**SetConsoleOutputCP(1251);**

**double a, b,x;**

**const double e = 0.00001;**

**a = 1;**

**b = 2;**

**x=0;**

**cout<<"a = "<<a<<endl;**

**cout<<"b = "<<b<<endl;**

**cout<<"e = "<<e<<endl;**

**cout<<"x = "<<x<<endl;**

**while (b - a > e)**

**{**

**if(f(a) \* f(x) < 0)**

**{**

**b = x;**

**}**

**else**

**{**

**a = x;**

**}**

**x= (a + b) / 2;**

**}**

**cout<<"x="<< x<<endl;**

**cout<<"f(x)="<< f(x)<<endl;**

**system("pause");**

**return 0;**

**Текст программы на PascalABC**

**program Laboratornay1;**

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

**begin**

**f:=cos(x)-exp(-x\*x/2)+x-1;**

**end;**

**var**

**a, b, e, c, x: real;**

**begin**

**a:=1;**

**b:=2;**

**write ('e=');**

**read(e);**

**c:=(a+b)/2;**

**while abs(b-a)>e do**

**begin**

**if f(a)\*f(c)<0 then**

**b:=c**

**else**

**a:=c;**

**c:=(a+b)/2;**

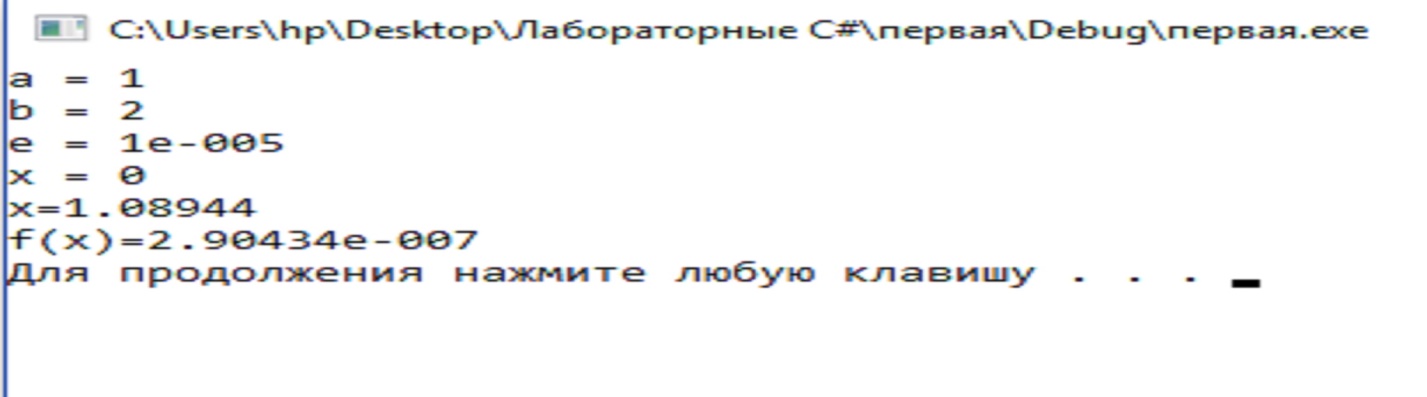
**end;**

**x:=(a+b)/2;**

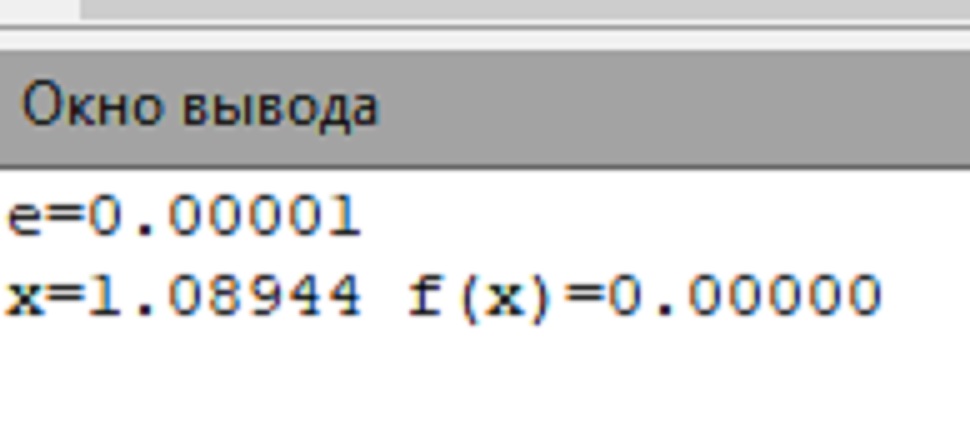
**writeln ('x=',x:5:5,' f(x)=',f(x):5:5);**

**end.**

**Результат С ++**

****

**Результат PascalABC**

****

**Проверка на MatCad**

**М е т о д п о л о в и н н о г о д е л е н и я**











**Вывод:**

Изучив язык высокого уровня С++ и Pascal я находил точки методом половинного деления в уравнение. Результате выполненной работы я определил, что если уменьшается точность погрешности увеличивается деления пополам.

# ЛАБОРАТОРНАЯ РАБОТА № 2

**Отделение и уточнение корней нелинейных уравнений**

Данная работа аналогична работе №1 с той разницей, что здесь задан ориентировочный участок функции, где могу быть один или не-сколько корней. Поэтому необходимо сначала отделить и определить до-статочно малый участок, где есть корень, методом проб, а затем уточнить значение корня быстросходящимся методом. Так как корней на задан-ном участке может быть несколько, например, как показано на рис. 5, то после уточнения корня необходимо проверять наличие корней и далее до конца участка.

При выполнении задания для метода проб отрезок предлагается делить на 20 частей. Из-за сложности анализа функций на сходимость для уточнения корней использовать метод половинного деления с задан-ной точностью.



Рис. 5. Отделение корней нелинейного уравнения.

Блок-схема алгоритма приведена на рис. 6, варианты заданий в таблице 2.

Таблица 2. Варианты заданий к лабораторной работе № 2.

|  |  |  |  |
| --- | --- | --- | --- |
| № вар | Уравнение | Отрезок | Заданная точность ε |
| 15 |  | [0; 2] | 10-3 |



Рис. 6. **Блок-схема алгоритма отделения и уточнения корней уравнения.**

**Текст программы Pascal**

uses crt;

function f(x:real):real;

begin

f:=x\*x-5\*sin(5\*x)+exp(-x);

end;

procedure koren(a,b,eps:real);

var c:real;

begin

repeat

c:=(f(b)\*a-f(a)\*b)/(f(b)-f(a));

if f(a)\*f(c) > 0 then a := c

else b := c;

until abs((f(b)\*a-f(a)\*b)/(f(b)-f(a))-c)<eps;

writeLn(c:10:7);

end;

var a,b,x,h,eps:real;

k:integer;

begin

clrscr;

a:=0;

b:=2;

h:=0.5;

eps:=0.001;

k:=0;

x:=a;

while x<b+h do

begin

if f(x)\*f(x+h)<0 then

begin

inc (k);

write(k,'-i корень в интервале [',x:4:1,';',x+h:4:1,']=');

koren(x,x+h,eps);

end;

x:=x+h;

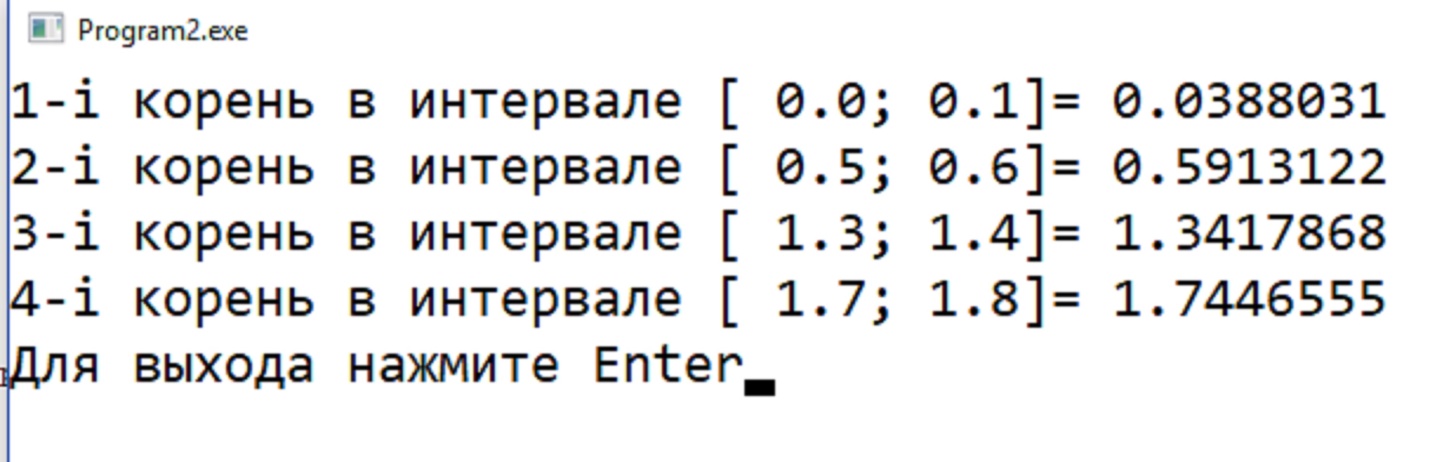
end;

write('Для выхода нажмите Enter');

readln

end.

**Результат Pascal**



**Проверка на MatCad**











**Вывод :**

В результате проверки был разработан алгоритм, который делил отрезок на 20 частей. На этих участках проверялось наличие корня и если корень существовал, то методом проб уточнялся до заданной точности.

# ЛАБОРАТОРНАЯ РАБОТА № 3

Вычисление определенных интегралов с заданным шагом интегрирования

Численные методы нахождения значений определенного интеграла используются в тех случаях, когда неизвестна первообраз-ная для функции f (x) или ее вычисление сопряжено с трудностями. Большинство методов основано на замене подынтегральной функции аппроксимирующей функцией более простого вида и последующим ин-тегрированием отрезков этой более простой функции.

1. Метод прямоугольников.

Этот метод основан на аппроксимации кусочно-постоянными функциями. Рабочая формула метода: 

(3.1)

Здесь n-число отрезков разбиения.

В случае, если:

Δ=0 – получаем рабочую формулу левых прямоугольников;

Δ=h/2 – формулу центральных прямоугольников;

Δ=h – формулу правых прямоугольников.

2. Метод трапеций.

В этом случае аппроксимация производится кусочно-линейной функцией в соответствии с рис. 7. Рабочая формула метода: 

(3.2)

Где n – число отрезков разбиения.



Рис. 7. Замена значения интеграла суммой площадей трапеций.

3. Метод Симпсона.

В этом случае подынтегральная функция аппроксимируется от-резками парабол. Рабочая формула метода:



Здесь n - число отрезков разбиения (обязательно четное).

Формулу можно привести к виду



Блок-схема алгоритма примера приведена на рис. 8, варианты за-даний в табл. 3. Вычислить приведенный в варианте интеграл методами центральных прямоугольников, трапеций и Симпсона при одинаковом заданном количестве отрезков разбиения.

**Таб. 3. Варианты задания к лабораторной работе № 3.**

|  |  |  |
| --- | --- | --- |
| № вар. | Интеграл | Количество частей разбиения |
| 15 |  | 240 |

**Блок-схема вычисления интеграла методом Симпсона**.



**Текст программы Pascal**

program Laboratornay;

uses crt;

var

x,a,b,h,s:real;

n:integer;

function Y(t:real):real;

begin

Y:=power(2,3\*x);

end;

begin

clrscr;

write('Отрезок интегрирования [a,b] : ');

read(a,b);

write('На сколько частей нужно разделить отрезок? n=');

read(n);

h:=(b-a)/n;

s:=0; x:=a+h;

while x<b do

begin

s:=s+4\*Y(x);

x:=x+h;

s:=s+2\*Y(x);

x:=x+h;

end;

s:=h/3\*(s+Y(a)-Y(b));

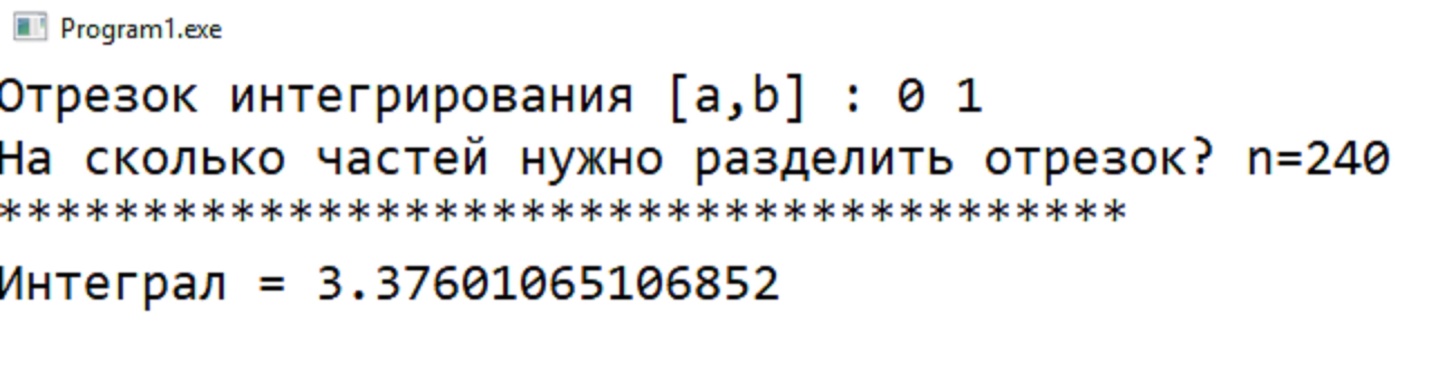
writeln('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

writeln('Интеграл = ',s);

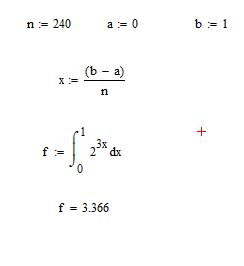
readkey;

end.

**Результат Pascal**

****

**Проверка на MatCad**

****

**Вывод :**

В ходе выполнения лабораторной работы были изучены и разобраны основы языка Pascal. Вычислен интеграл методом Симпсона.

# ЛАБОРАТОРНАЯ РАБОТА № 4

**Вычисление определенных интегралов с заданной точностью**

Обычно при вычислении интегралов заранее известна точность, с которой необходимо произвести расчет. Количество же участков разбие-ния зависит от свойств подынтегральной функции и выбирается намно-го больше, чем это необходимо для обеспечения заданной точности.

Чтобы не делать лишних вычислений (а с ростом вычислений уве-личивается и аппаратная ошибка, связанная с конечной длиной числа, представленного в ЭВМ), используют алгоритм с удвоением отрезков разбиения и сравнением интегралов, вычисленных с n отрезками разби-ения и 2n.

Алгоритм метода

1. Вычисление интеграла с двумя отрезками разбиения.

2. Увеличение отрезков разбиения в 2 раза.

3. Вычисление интеграла с новым количеством отрезков.

4. Сравнение предыдущего и вычисленного значения интеграла. Если разность значений больше заданной точности, то перейти на шаг 2, иначе конец алгоритма.

На рис. 9 представлена схема вычисления интеграла по приведенному выше алгоритму. Очевидно, что значения f (x0), и f (xn) постоянны для интеграла с любым количеством отрезков разбиения, поэтому их можно вычислить один раз в начале программы. Значения промежуточных результатов (х11 для двух отрезков разбиения) будут использоваться как четные слагаемые для следующего значения интеграла (х22 для четырех отрезков разбиения). Таким образом, остается вычислить только сумму для нечетных значений функции.

**Метод прямоугольников.**

Этот метод основан на аппроксимации кусочно-постоянными функциями. Рабочая формула метода:



Здесь n-число отрезков разбиения.

В случае, если:

Δ=0 – получаем рабочую формулу левых прямоугольников;

Δ=h/2 – формулу центральных прямоугольников;

Δ=h – формулу правых прямоугольников.

**Таб. 3. Варианты задания к лабораторной работе № 4.**

|  |  |  |
| --- | --- | --- |
| № вар. | Интеграл | Количество частей разбиения |
| 15 |  | 240 |

**Блок-схема алгоритма вычисления интеграла с заданной точностью.**



**Текст программы Pascal**

**program** Laboratornay4;

**const**

min = 0;

**const**

max= 1;

**const**

e = 240;

**var**

x, s1, h, s, a, n: real;

i: integer;

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

**begin**

result := power(2,3\*x);

**end**;

**begin**

s := 0;

s1 := 0;

h := 0;

a := 0;

n := 240;

**repeat**

**begin**

s1 := s;

h := (max - min) / n;

a := min;

s := 0;

x := a;

i := 0;

**while**(i < n - 1)

**do**

**begin**

inc(i);

s := s + abs(f(x));

x := x + h;

**end**;

s := h \* s;

n := n \* 2;

**end**;

**until** (abs(s1 - s) < e);

writeln( '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

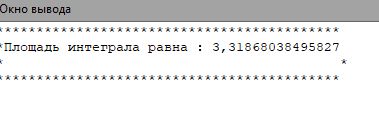
writeln( '\*Площадь интеграла равна : ' + s.toString());

writeln( '\* \*');

writeln( '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

**end**.

**Результат Pascal :**

****

**Вывод :**

В ходе выполнения лабораторной работы были изучены и разобраны основы языка Pascal. Вычислен интеграл методом прямоугольников с заданной точностью.

# ЛАБОРАТОРНАЯ РАБОТА №8

**Интерполяция сеточных функций полиномами.**

Простейшая задача интерполирования заключается в следующем. На отрезке [*a, b*] задано *n*+1 точек *x*0, *x*1, …, *xn*, которые называются узлами интерполяции, и значения некоторой функции *f*(*x*) в этих точках:

*f*(*x*0) = *y*0, *f*(*x*1) = *y*1, …, *f*(*xn*) = *yn*, (8.1)

Требуется построить функцию *F*(*x*) (интерполирующая функция), принадлежащую известному классу и принимающую в узлах интерполяции те же значения, что и *f*(*x*). При использовании интерполяционной формулы Лагранжа функция *F*(*x*) является полиномом *Ln*(*x*) степени не выше *n*, который выражается следующей формулой:

 (8.2)

или

 (8.3)

Таким образом, для решения данной задачи необходимо организовать тройной циклический процесс, два цикла которого предназначены для вычисления одной точки полинома *Ln*(*x*).

Используя интерполяционную формулу Лагранжа, вывести в виде таблицы значения функции *f*(*x*) для точек *x* с заданным шагом, если известны значения функции *f*(*x*i) в узлах *x*i.

Блок-схема алгоритма для данного примера приведена на рис. 11, варианты заданий в табл. 6.

**Табл 6. Варианты заданий к лабораторной работе № 8.**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| №  вар |  | Значения *i* | | | | | | | | Отрезок интерполирования | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | *xнач* | *Xкон* | *шаг* |
| 15 | *xi*  *yi* | 0,3  7,2 | 0,9  5,8 | 1,1  4,4 | 1,8  0,2 | 2,1  2 | 2,8  3,4 | 3,4  -1,2 | 3,9  -3,7 | 0,6 | 3,4 | 0,2 |

**Блок-схема алгоритма интерполирования функции**



**Текст программы Pascal**

**uses** crt;

**var**

F, L, xn, x0, y0, xk, sh: real;

i, j, n, d: integer;

x: **array**[1..8] **of** real;

y: **array**[1..8] **of** real;

**begin**

clrscr;

xn := 0.6;xk := 3.4;sh := 0.2;

n := 8;d := 1;

x[1] := 0.3;y[1] :=7.2;

x[2] := 0.9;y[2] := 5.8;

x[3] := 1.1;y[3] := 4.4;

x[4] := 1.8;y[4] :=0.2;

x[5] := 2.1;y[5] := 2;

x[6] := 2.8;y[6] := 3.4;

x[7] :=3.4;y[7] := -1.2;

x[8] := 3.9;y[8] :=-3.7;

writeln('-----------------------------');

writeln('| d | x | y |');

writeln('-----------------------------');

**while** xn <= xk **do**

**begin**

y0 := 0;

x0 := xn;

F := 0;

**for** j := 1 **to** n **do**

**begin**

L := 1;

**for** i := 1 **to** n **do**

**begin**

**if** i <> j **then**

**begin**

L := L \* (x0 - x[i]) / (x[j] - x[i]);

**end**;

**end**;

y0 := y0 + y[j] \* L;

**end**;

writeln('| ', d:2, ' | ', x0:3:2, ' | ', y0:8:3, ' |');

xn := xn + sh;

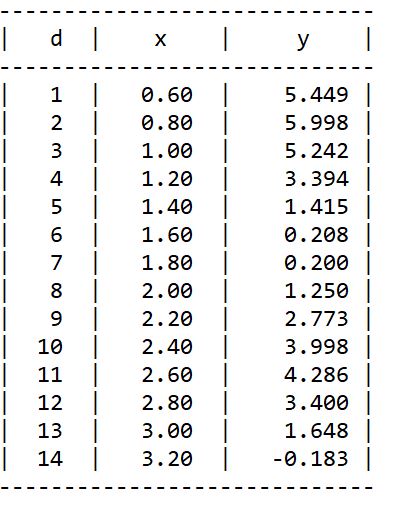
d := d + 1;

**end**;

writeln('-----------------------------');

readln; end;

**Результат Pascal :**



**Вывод :**

После проведенных работы в Pascal по лабораторной работе были вычислены значения функции f (x) для точек x с заданным шагом и выведены в виде таблицы.