**Московский авиационный институт**

(национальный исследовательский университет)

Факультет радиоэлектроники

Кафедра 403



**Разработка алгоритмов и программ решения**

**алгебраических задач численными методами**

Расчётно-графическая работа

по дисциплине «Информатика»

|  |  |
| --- | --- |
|  | Выполнил:  *студент группы М4О-103Б-17*  Парамонов Н. М. |
|  | Принял:  *преподаватель кафедры 403*  Кошелькова Л. В. |

Москва

2018 г.

**Оглавление**

[1. Условие задачи 3](#_Toc512806841)

[2. Анализ задания 3](#_Toc512806842)

[3. Теоретические сведения 4](#_Toc512806843)

[4. Схемы алгоритмов 5](#_Toc512806844)

[5. Описание C++ Builder-программы 12](#_Toc512806845)

[6. Текст программы 13](#_Toc512806846)

[7. Пример выполнения программы 17](#_Toc512806847)

[8. Консольное приложение 18](#_Toc512806848)

[9. Набор тестов 20](#_Toc512806849)

[10. Выводы 24](#_Toc512806850)

[11. Список использованной литературы 25](#_Toc512806851)

# **Условие задачи**

Вариант №79

Разработать схему алгоритма, составить C++ Builder-проект вычисления таблицы значений функции:

Аргумент X принимает N значений от Xn с шагом Dx, а параметр А принимает значения от An до Аk с шагом Da. Параметр B принимает значение, численно равное корню нелинейного уравнения:

вычисленного на интервале и с заданной погрешностью .

# **Анализ задания**

Входные данные:

1. Xn – начальное значение аргумента, тип – с плавающей точкой;
2. Dx – шаг изменения аргумента, тип – с плавающей точкой;
3. N – число значений аргумента, тип – целый;
4. An – начальное значение параметра A, тип – с плавающей точкой;
5. Ak – конечное значение параметра A, тип – с плавающей точкой;
6. Da – шаг изменения параметра A, тип – с плавающей точкой;
7. C, D – интервал изоляции, тип – с плавающей точкой;
8. Eps – погрешность вычисления корня нелинейного уравнения, тип – с плавающей точкой;
9. Km - максимальное количество итераций, тип – целый;

Выходные данные:

1. Mx – массив (одномерный) значений аргумента X, тип – с плавающей точкой;
2. My – массив (двумерный) значений функции Y, тип – с плавающей точкой;
3. Ma – массив (одномерный) значений параметра A, тип – с плавающей точкой;
4. B – численное значение корня нелинейного уравнения, тип – с плавающей точкой;
5. Zt – Погрешность вычисления корня по невязке, тип – с плавающей точкой
6. Er – массив (двумерный) признака ошибки при вычислении функции, тип – целый;
7. Equat() – признак ошибки при вычислении корня нелинейного уравнения, тип – bool.

В алгоритме выполняются следующие функции:

1. Ввод исходных данных;
2. Вычисление корня нелинейного уравнения;
3. Вычисление таблицы значений функции;
4. Проверка значения подкоренного выражения и формирование признака ошибки, если оно имеет отрицательный знак;
5. Вывод результатов вычислений.

# **Теоретические сведения**

***Метод дихотомии***

Метод половинного деления, или иначе метод дихотомии. Метод дихотомии получил свое название от древнегреческого слова διχοτομία, что в переводе означает деление надвое. Его мы используем довольно часто. Допустим, играя в игру "Угадай число", где один игрок загадывает число от 1 до 100, а другой пытается его отгадать, руководствуясь подсказками "больше" или "меньше". Логично предположить, что первым числом будет названо 50, а вторым, в случае если оно меньше - 25, если больше - 75. Таким образом, на каждом этапе неопределенность неизвестного x3 x1 x2 X Y y=f(x) уменьшается в 2 раза. Т.е. даже самый невезучий в мире человек отгадает загаданное число в данном диапазоне за 7 предположений вместо 100 случайных утверждений.

Алгоритм метода половинного деления основан на теореме Больцано - Коши о промежуточных значениях непрерывной функции и следствии из неё.

Теорема Больцано - Коши: если непрерывная функция принимает два значения, то она принимает любое значение между ними.

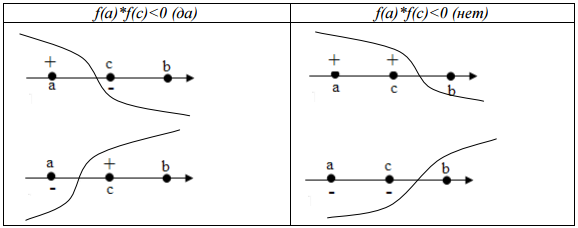
Следствие (теорема о нуле непрерывной функции): если непрерывная функция принимает на концах отрезка положительное и отрицательное значения, то существует точка, в которой она равна 0.

Алгоритм:

1. Задать отрезок [a,b] и погрешность ε.

2. Вычислить c=(a+b)/2

3. Определить интервал дальнейшего поиска: если f(a) и f(c) имеют разные знаки, т.е. f(a)\*f(c)<0, то b=c, в противном случае a=c.



4. Если длина нового отрезка |b-a|<=ε, то вычислить значение корня c=(a+b)/2 и остановиться, в противном случае перейти к шагу 2.

# **Схемы алгоритмов**

В соответствии с принципами структурного программирования каждый функционально законченный фрагмент программы оформ­лен в виде подпрограммы. В результате программа включает глав­ную программу и набор подпрограмм, предназначенных соответ­ственно для табулирования функции (Tab), вычисления корня нелинейного уравнения (Equat), вывода результатов выполнения программы (RezOut).

Схема алгоритма главной программы представлена на рис. 1, а таблица обозначения переменных главной программы – в табл. 1.

Главная программа начинается с ввода значений входных данных.

*Таблица 1*

|  |  |  |
| --- | --- | --- |
| Обозначение в задании | Обозначение в алгоритме | Наименование |
| Xn | xn | Начальное значение аргумента, тип – с плавающей точкой |
| Dx | dx | Шаг изменения аргумента, тип – с плавающей точкой |
| N | n | Число значений аргумента, тип – целый |
| An | an | Начальное значение параметра A, тип – с плавающей точкой |
| Ak | ak | Конечное значение параметра A, тип – с плавающей точкой |
| Da | da | Шаг изменения параметра A, тип – с плавающей точкой |
| C, D | c, d | Интервал изоляции, тип – с плавающей точкой |
|  | eps | Погрешность вычисления корня нелинейного уравнения, тип – с плавающей точкой |
|  | km | Максимальное количество итераций, тип – целый |
|  | Mx | Массив значений аргумента X, тип – с плавающей точкой |
|  | My | Массив значений функции Y, тип – с плавающей точкой |
|  | Ma | Массив значений параметра A, тип – с плавающей точкой |
| B | b | Параметр функции, тип – с плавающей точкой |
| Zt | zt | Погрешность вычисления корня по невязке, тип – с плавающей точкой |
|  | Er | Массив признака ошибки при вычислении функции, тип – целый |
|  | Equat() | Признак ошибки при вычислении определённого интеграла, тип – целый |
| X | x | Аргумент, тип – с плавающей точкой |
| Y | y | Функция, тип – с плавающей точкой |
|  | m | Количество значений параметра A, тип – целый |
|  | i, j | Счётчики числа повторений циклов, тип – целый |

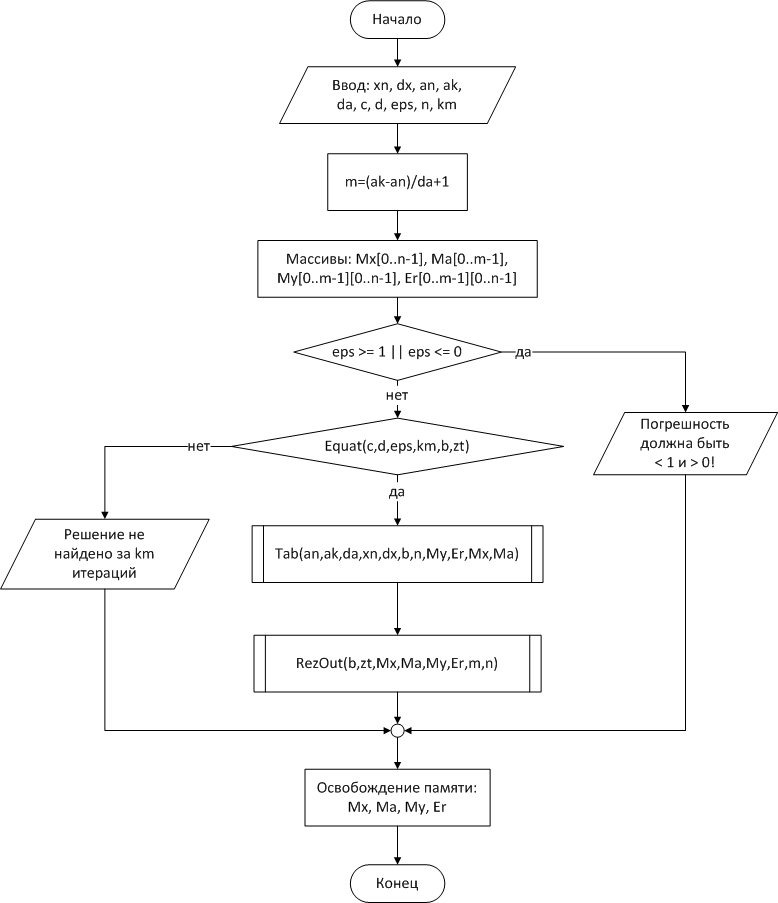


Рис. 1. Схема алгоритма главной программы

Вычисление значения корня уравнения производится путём обращения к ПФ **Equat**, возвращающей также признак ошибки в случае, если значение корня уравнения не найдено за предельно допустимое число итераций Km. При этом выводится диагностическое сообщение «Решение н найдено за Km итераций», иначе происходит табулирование функции (ПП **Tab**) и вывод результатов выполнения программы (ПП **RezOut**). Значение m определяет количество значений параметра A.

Схемы алгоритмов подпрограмм, используемых в данной программе, с указанием их назначения и списков формальных параметров приведены на рис. 2 – 5.

Подпрограмма-функция **F** (рис. 2) предназначена для вычисления значения функции уравнения, представляет собой один оператор присваивания и используется в ПП вычисления значения корня уравнения Equat.

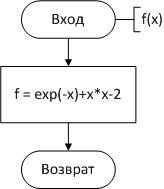


Рис. 2. Схема алгоритма подпрограммы-функции F

Подпрограмма-функция F предназначена для вычисления значения функции F.

Список формальных параметров: X.

Входные параметры:

x – аргумент функции, тип – с плавающей точкой.

Подпрограмма-процедура вычисления корня уравнения **Equat** (рис. 3) реализуется циклом итерационного типа, который завершается при условии fabs(f(z)) <= eps (проверка по невязке). Во избежание «зацикливания» программы при неудачном выборе начального приближения (или ошибках в данных) в алгоритме предусмотрено задание предельно допустимого количества повторений цикла Km и выполнение соответствующего арифметического цикла с проверкой окончания i <= Km.

Подпрограмма-процедура табулирования **Tab** (рис. 4), выполненная в виде двойного цикла, определяет функциональную зависимость вида y = f(a,x) при различных значениях параметров, поэтому внутренний цикл должен быть связан с изменением аргумента X, а внешний – с изменением параметра A. Во внутреннем цикле имеются две развилки: одна из них обусловлена тем, что функция задаётся разными формулами на разных участках изменения аргумента (проверка условия X < 1), вторая – проверяет знак подкоренного выражения и при условии A\*X+B2 > 0 вычисляет значение функции, в противном случае – формирует признак ошибки Er[I][J] = 1.

Подпрограмма-процедура **RezOut** (рис. 5) выводит результаты выполнения программы. По структуре она построена аналогично подпрограмме Tab: представляет собой двойной цикл арифметического типа, но в отличие от ПП Tab она получает количество значений параметра A в качестве входных данных.

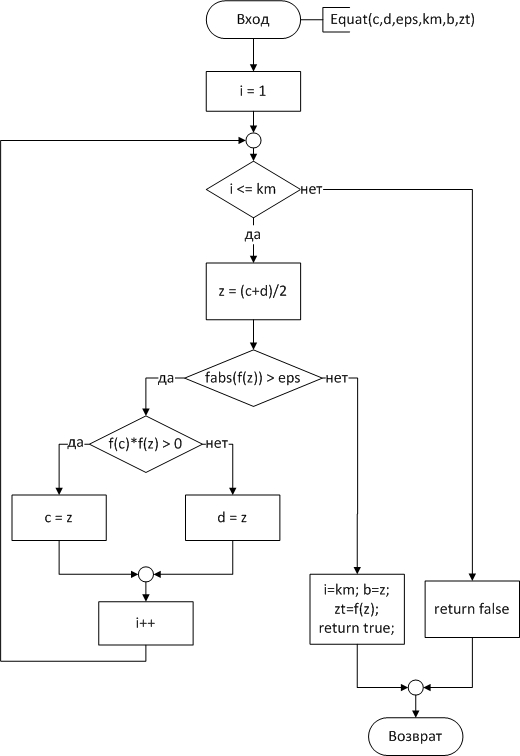


Рис. 3. Схема алгоритма подпрограммы-функции Equat

Подпрограмма-функция Equat предназначена для вычисления значения корня нелинейного уравнения с заданной погрешностью методом половинного деления.  
Список формальных параметров: c, d, eps, km, b, zt.

Входные параметры:

c, d – интервал изоляции, тип – с плавающей точкой;

eps – погрешность вычисления корня уравнения, тип – с плавающей точкой;

km – предельно допустимое количество итераций, тип – целый.

Выходные параметры:

b – значение корня нелинейного уравнения, тип – с плавающей точкой;

zt – погрешность вычисления корня по невязке, тип – с плавающей точкой;

Equat() – признак ошибки, тип – bool.

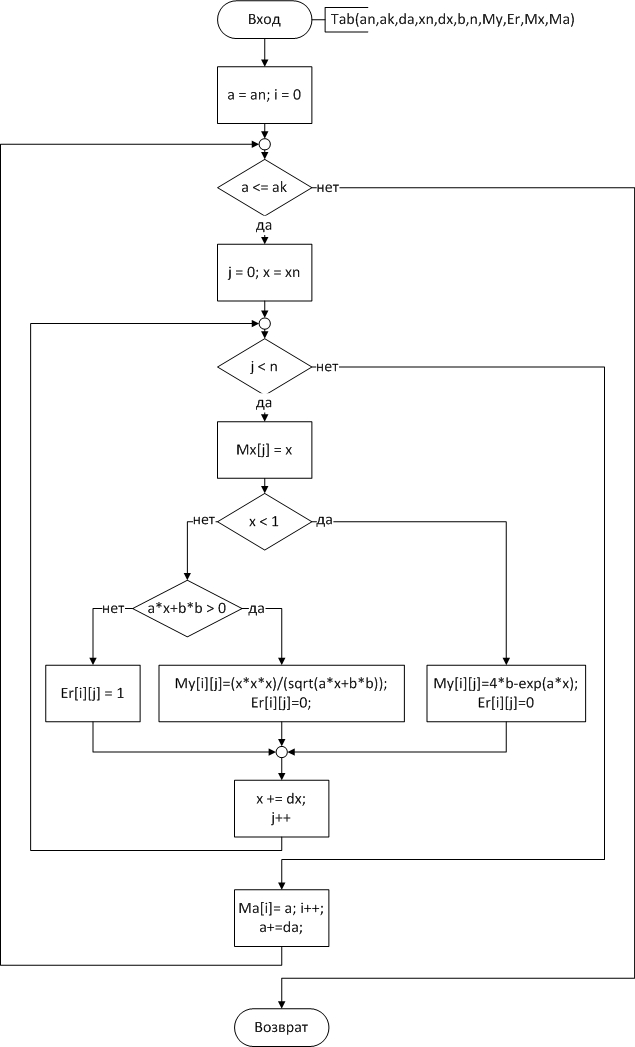


Рис. 4. Схема алгоритма подпрограммы-процедуры Tab

Подпрограмма-процедура Tab предназначена для вычисления таблицы значений функции Y.  
Список формальных параметров: an, ak, da, xn, dx, n, b, er, Mx, My, Ma.

Входные параметры:

an – начальное значение параметра A, тип – с плавающей точкой;

ak – конечное значение параметра A, тип – с плавающей точкой;

da – шаг изменения параметра A, тип – с плавающей точкой;

xn – начальное значение аргумента, тип – с плавающей точкой;

dx – шаг изменения аргумента, тип – с плавающей точкой;

n – количество значений аргумента, тип – целый;

b – параметр функции, тип – с плавающей точкой.

Выходные параметры:

Er – массив признака ошибки, тип – целый;

Mx – массив значений аргумента X, тип – с плавающей точкой;

My – массив значений функции Y, тип – с плавающей точкой;

Ma – массив значений параметра A, тип – с плавающей точкой;

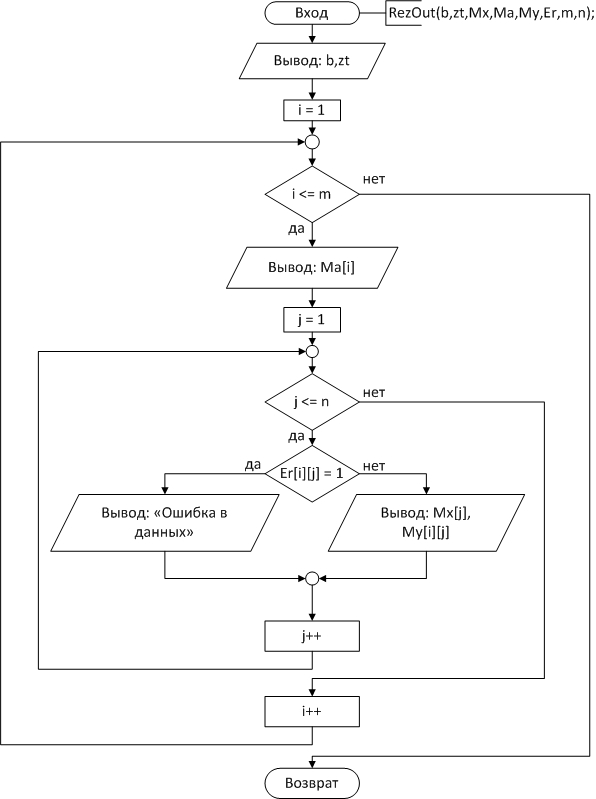


Рис. 5. Схема алгоритма подпрограммы-процедуры RezOut

Подпрограмма-процедура RezOut предназначена для вывода результатов выполнения программы на внешние носители информации.  
Список формальных параметров: Mx, My, Ma, Er, n, m.

Входные параметры:

Mx – массив значений аргумента X, тип – с плавающей точкой;

My – массив значений функции Y, тип – с плавающей точкой;

Ma – массив значений параметра A, тип – с плавающей точкой;

Er – массив признака ошибки, тип – целый;

n – количество значений аргумента, тип – целый;

m – количество значений параметра A, тип - целый.

# **Описание C++ Builder-программы**

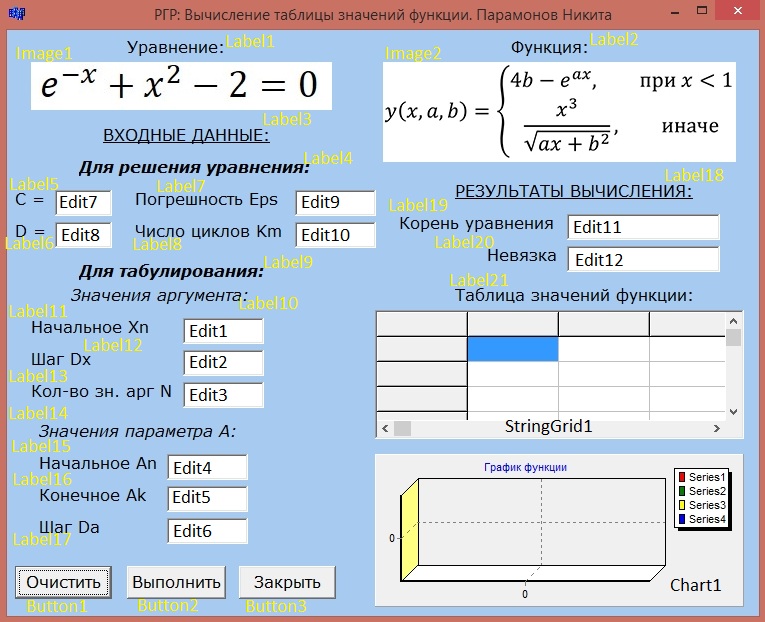


Рис. 6. Форма C++ Builder-приложения (структура проекта)

Разработка приложения в визуальной среде программирования C++ Builder включает два этапа:

* разработка интерфейса приложения;
* определение функциональности приложения, т.е. написание кода.

Интерфейс определяет способ взаимодействия пользователя и приложения, т.е. внешний вид формы при выполнении приложения и то, каким образом пользователь управляет приложением.

Разработка интерфейса состоит в создании главного окна, т.е. в расположении на форме необходимых компонентов редактирования, отображения и управления. Внешний вид формы для задачи табулирования функции представлен на рисунке 6. На форме расположены следующие визуальные компоненты: Label, Edit, Button, StringGrid, Chart, Image.

Функциональность приложения определяется процедурами, которые выполняются при возникновении определенных событий. Структура C++ Builder-проекта соответствует рассмотренным в предыдущем разделе схемам алгоритмов, но дополнительно включает процедуры или функции преобразования данных символьного типа в арифметические при вводе и обратного преобразования арифметических данных в строковые – при выводе. Текст модуля формы представлен в следующем пункте.

Обработчик кнопки «Выполнить» по событию OnClick реализует процедуры (Equat, Tab, RezOut), необходимые для выполнения задачи. Обработчик включает в себя: функции преобразования входных данных типа String, полученных из компонентов Edit, в числа с плавающей точкой типа double или целые числа типа int; вызов процедур Equat, Tab, RezOut; заполнение таблицы StringGrid – вывод данных; функции преобразования выходных данных типа double (значения корня уравнения и невязки) в данные типа String для вывода в компоненты Edit; вывод сообщений об ошибках, если они присутствуют; вывод графика с использованием компонента Chart.

Обработчик кнопки «Очистить» по событию OnClick включает в себя: очистку компонентов Edit, используемых для получения входных (выходных – для значения корня уравнения и невязки) данных, очистку серий компонента Chart, очистку компонента StringGrid в цикле.

Обработчик кнопки «Закрыть» по событию OnClick включает в себя метод Close, обеспечивающий закрытие приложения.

# **Текст программы**

Код модуля UnitRGR.cpp:

//---------------------------------------------------------------------------

#include <vcl.h>

#pragma hdrstop

#include <math.h>

#include "UnitRGR.h"

#pragma package(smart\_init)

#pragma resource "\*.dfm"

TForm1 \*Form1;

//---------------------------------------------------------------------------

\_\_fastcall TForm1::TForm1(TComponent\* Owner)

: TForm(Owner){

}

//---------------------------------------------------------------------------

// Кнопка "Закрыть"

void \_\_fastcall TForm1::Button3Click(TObject \*Sender){

Close();

}

//---------------------------------------------------------------------------

// Кнопка "Очистить"

void \_\_fastcall TForm1::Button1Click(TObject \*Sender){

Edit1->Clear();Edit2->Clear();Edit3->Clear();Edit4->Clear();Edit5->Clear();Edit6->Clear();Edit7->Clear();Edit8->Clear();Edit9->Clear();Edit10->Clear();Edit11->Clear();Edit12->Clear();

for (int i=0; i<=StringGrid1->ColCount; i++)

for (int j=0; j<=StringGrid1->RowCount; j++)

StringGrid1->Cells[i][j]="";

for(int i=0;i<=Form1->Chart1->SeriesList->CountActive()-1;i++) Form1->Chart1->Series[i]->Clear();

}

//---------------------------------------------------------------------------

// Прототип нелинейное уравнение

double f(double);

// Прототип нахождение корня нелин. ур-я

bool Equat(double,double,double,int,double&,double&);

// Прототип табулирование функции

void Tab(double,double,double,double,double,double,int,double\*\*,double\*\*,double\*,double\*);

// Прототип вывод результатов

void RezOut(double,double,double\*,double\*,double\*\*,double\*\*,int,int);

//---------------------------------------------------------------------------

//Кнопка "Выполнить"

void \_\_fastcall TForm1::Button2Click(TObject \*Sender){

// Ввод исходных данных

double

xn=StrToFloat(Edit1->Text),

dx=StrToFloat(Edit2->Text),

an=StrToFloat(Edit4->Text),

ak=StrToFloat(Edit5->Text),

da=StrToFloat(Edit6->Text),

c=StrToFloat(Edit7->Text),

d=StrToFloat(Edit8->Text),

eps=StrToFloat(Edit9->Text),

b,zt;

int

n=StrToInt(Edit3->Text),

km=StrToInt(Edit10->Text),

m=(int)((ak-an)/da+1); // Кол-во зн-й параметра А

// Объявление динамических массивов

double\*Mx=new double[n];

double\*Ma=new double[m];

double \*\*My= new double\* [m];

for(int i=0;i<m;i++) My[i]=new double[n];

double \*\*Er= new double\* [m];

for(int i=0;i<m;i++) Er[i]=new double[n];

// Проверка на наличие ошибок и вызов подпрограмм

if (eps >= 1 || eps <= 0) ShowMessage("Погрешность должна быть < 1 и > 0!");

else{

if (Equat(c,d,eps,km,b,zt)){

Tab(an,ak,da,xn,dx,b,n,My,Er,Mx,Ma);

RezOut(b,zt,Mx,Ma,My,Er,m,n);

} else ShowMessage("Решение не найдено за "+IntToStr(km)+" итераций");

}

// Освобождение памяти

delete [] Mx;

delete [] Ma;

for(int i=0;i<m;i++) delete[]My[i];

delete [] My;

for(int i=0;i<m;i++) delete[]Er[i];

delete [] Er;

}

//---------------------------------------------------------------------------

// Нелинейное уравнение

double f(double x){

return (exp(-x)+x\*x-2);

}

//---------------------------------------------------------------------------

// Нахождение корня нелинейного уравнения методом деления отрезка пополам

bool Equat(double c,double d,double eps,int km,double&b,double&zt){

int i=1;

double z;

while(i<=km){

z=(c+d)/2;

if(fabs(f(z)) > eps)

if(f(c)\*f(z) > 0) c=z; else d=z;

else{

i=km;

b=z;

zt=f(z); // Погрешность вычисления корня по невязке

return true;

}

i++;

}

return false;

}

//---------------------------------------------------------------------------

// Табулирование функции

void Tab(double an,double ak,double da,double xn,double dx,double b,int n,double\*\*My,double\*\*Er,double\*Mx,double\*Ma){

double a=an,x;

int i=0;

while (a<=ak){

x=xn;

for (int j=0;j<n;j++){

Mx[j]=x;

if (x<1){

My[i][j]=4\*b-exp(a\*x);

Er[i][j]=0;

}

else

if (a\*x+b\*b>0){

My[i][j]=(x\*x\*x)/(sqrt(a\*x+b\*b));

Er[i][j]=0;

}

else Er[i][j]=1;

x+=dx;

}

Ma[i]= a;

i++;

a+=da;

}

}

//---------------------------------------------------------------------------

// Вывод результатов

void RezOut(double b,double zt,double\*Mx,double\*Ma,double\*\*My,double\*\*Er,int m,int n){

Form1->Edit11->Text=FloatToStrF(b,ffGeneral,10,6);

Form1->Edit12->Text=FloatToStrF(zt,ffGeneral,10,6);

Form1->StringGrid1->RowCount=n+1;

Form1->StringGrid1->ColCount=m+1;

Form1->StringGrid1->Cells[0][0]="X/A";

for(int i=0;i<m;i++){

Form1->StringGrid1->Cells[i+1][0]="A["+IntToStr(i)+"]="+FloatToStr(Ma[i]);

Form1->Chart1->Series[i]->Title="A["+IntToStr(i)+"]";

Form1->Chart1->Series[i]->Clear();

for (int j=0;j<n;j++){

Form1->StringGrid1->Cells[0][j+1]="X["+IntToStr(j)+"]="+FloatToStr(Mx[j]);

if(Er[i][j]==1)

Form1->StringGrid1->Cells[i+1][j+1]="Err";

else

Form1->StringGrid1->Cells[i+1][j+1]=FloatToStrF(My[i][j],ffGeneral,6,5);

Form1->Chart1->Series[i]->AddXY(Mx[j],My[i][j]);

}

}

}

//---------------------------------------------------------------------------

Файл UnitRGR.h:

//---------------------------------------------------------------------------

#ifndef UnitRGRH

#define UnitRGRH

//---------------------------------------------------------------------------

#include <Classes.hpp>

#include <Controls.hpp>

#include <StdCtrls.hpp>

#include <Forms.hpp>

#include <ExtCtrls.hpp>

#include <Grids.hpp>

#include <jpeg.hpp>

#include <Chart.hpp>

#include <TeEngine.hpp>

#include <TeeProcs.hpp>

#include <Series.hpp>

//---------------------------------------------------------------------------

class TForm1 : public TForm

{

\_\_published: // IDE-managed Components

TLabel \*Label2;

TImage \*Image1;

TImage \*Image2;

TLabel \*Label3;

TLabel \*Label4;

TLabel \*Label5;

TEdit \*Edit1;

TEdit \*Edit2;

TEdit \*Edit3;

TLabel \*Label1;

TLabel \*Label6;

TLabel \*Label7;

TLabel \*Label8;

TLabel \*Label9;

TLabel \*Label10;

TLabel \*Label11;

TLabel \*Label12;

TEdit \*Edit4;

TEdit \*Edit5;

TEdit \*Edit6;

TLabel \*Label13;

TStringGrid \*StringGrid1;

TButton \*Button1;

TButton \*Button2;

TButton \*Button3;

TLabel \*Label14;

TLabel \*Label15;

TLabel \*Label16;

TLabel \*Label17;

TEdit \*Edit7;

TEdit \*Edit8;

TLabel \*Label18;

TLabel \*Label19;

TEdit \*Edit9;

TEdit \*Edit10;

TEdit \*Edit11;

TEdit \*Edit12;

TLabel \*Label20;

TLabel \*Label21;

TChart \*Chart1;

TLineSeries \*Series1;

TLineSeries \*Series2;

TLineSeries \*Series3;

TLineSeries \*Series4;

TLineSeries \*Series5;

TLineSeries \*Series6;

TLineSeries \*Series7;

void \_\_fastcall Button3Click(TObject \*Sender);

void \_\_fastcall Button1Click(TObject \*Sender);

void \_\_fastcall Button2Click(TObject \*Sender);

private: // User declarations

public: // User declarations

\_\_fastcall TForm1(TComponent\* Owner);

};

//---------------------------------------------------------------------------

extern PACKAGE TForm1 \*Form1;

//---------------------------------------------------------------------------

#endif

# **Пример выполнения программы:**

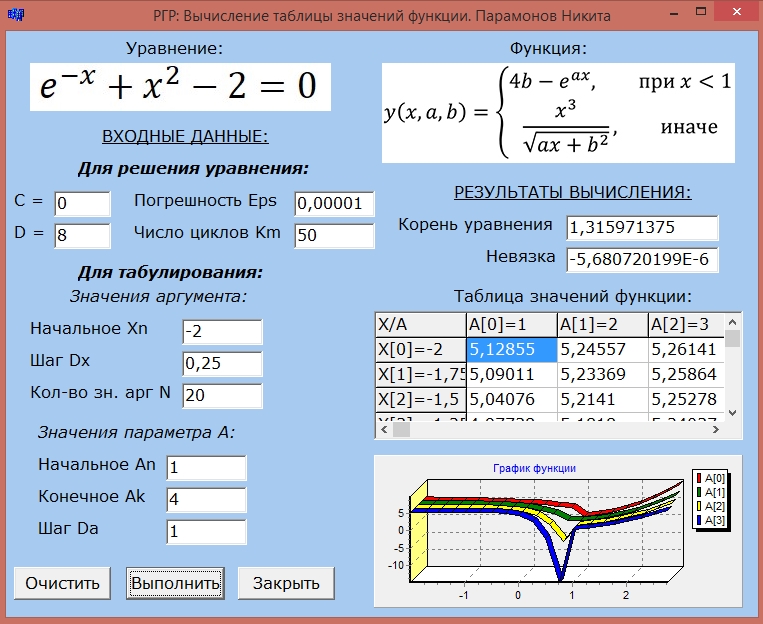


Рис. 7. Форма C++Builder-приложения с результатами выполнения программы

# **Консольное приложение**

Текст программы:

//---------------------------------------------------------------------------

#pragma hdrstop

#include <math.h>

#include <iostream>

#include <windows.h>

#pragma argsused

using namespace std;

//---------------------------------------------------------------------------

// Прототип нелинейное уравнение

double f(double);

// Прототип нахождение корня нелин. ур-я

bool Equat(double,double,double,int,double&,double&);

// Прототип табулирование функции

void Tab(double,double,double,double,double,double,int,double\*\*,double\*\*,double\*,double\*);

// Прототип вывод результатов

void RezOut(double,double,double\*,double\*,double\*\*,double\*\*,int,int);

//---------------------------------------------------------------------------

int main(int argc, char\* argv[]){

SetConsoleCP(1251);SetConsoleOutputCP(1251); // Русский язык в консоли

double xn,dx,an,ak,da,c,d,eps,b,zt;

int n,km,m;

cout << "Введите xn,dx,an,ak,da,c,d,eps,n,km: " << endl;

scanf("%lf %lf %lf %lf %lf %lf %lf %lf %i %i", &xn,&dx,&an,&ak,&da,&c,&d,&eps,&n,&km);

system("cls");

printf("\t\tВХОДНЫЕ ДАННЫЕ\nXn=%.2lf Dx=%.2lf An=%.2lf Ak=%.2lf Da=%.2lf C=%.2lf D=%.2lf Eps=%lf N=%i Km=%i\n\n\n",xn,dx,an,ak,da,c,d,eps,n,km);

m=(int)((ak-an)/da+1);

double\*Mx=new double[n];

double\*Ma=new double[m];

double \*\*My= new double\* [m];

for(int i=0;i<m;i++) My[i]=new double[n];

double \*\*Er= new double\* [m];

for(int i=0;i<m;i++) Er[i]=new double[n];

if (eps >= 1 || eps <= 0) cout << "\n\a\t\tПогрешность должна быть < 1 и > 0!\n";

else{

if (Equat(c,d,eps,km,b,zt)){

Tab(an,ak,da,xn,dx,b,n,My,Er,Mx,Ma);

RezOut(b,zt,Mx,Ma,My,Er,m,n);

} else cout << "\n\a\t\tРешение не найдено за "<<km<<" итераций\n";

}

delete [] Mx;

delete [] Ma;

for(int i=0;i<m;i++) delete[]My[i];

delete [] My;

for(int i=0;i<m;i++) delete[]Er[i];

delete [] Er;

system("pause");return 0;

}

//---------------------------------------------------------------------------

// Нелинейное уравнение

double f(double x){

return (exp(-x)+x\*x-2);

}

//---------------------------------------------------------------------------

// Нахождение корня нелинейного уравнения методом деления отрезка пополам

bool Equat(double c,double d,double eps,int km,double&b,double&zt){

int i=1;

double z;

while(i<=km){

z=(c+d)/2;

if(fabs(f(z)) > eps)

if(f(c)\*f(z) > 0) c=z; else d=z;

else{

i=km;

b=z;

zt=f(z); // Погрешность вычисления корня по невязке

return true;

}

i++;

}

return false;

}

//---------------------------------------------------------------------------

// Табулирование функции

void Tab(double an,double ak,double da,double xn,double dx,double b,int n,double\*\*My,double\*\*Er,double\*Mx,double\*Ma){

double a=an,x;

int i=0;

while (a<=ak){

x=xn;

for (int j=0;j<n;j++){

Mx[j]=x;

if (x<1){

My[i][j]=4\*b-exp(a\*x);

Er[i][j]=0;

}

else

if (a\*x+b\*b>0){

My[i][j]=(x\*x\*x)/(sqrt(a\*x+b\*b));

Er[i][j]=0;

}

else Er[i][j]=1;

x+=dx;

}

Ma[i]= a;

i++;

a+=da;

}

}

//---------------------------------------------------------------------------

// Вывод результатов

void RezOut(double b,double zt,double\*Mx,double\*Ma,double\*\*My,double\*\*Er,int m,int n){

cout << "\t\tРЕЗУЛЬТАТЫ ВЫЧИСЛЕНИЯ\nКорень уранения: " << b << "; Невязка: " << zt << endl;

cout<<"\nТаблица значений функции:\n"<<endl;

printf("%-15s", "X/A");

for(int i=0;i<m;i++) printf("%2s%i%2s%-10.3lf","A[",i,"]=",Ma[i]);

cout << endl;

for(int i=0;i<n;i++){

printf("%2s%02i%2s%-9.2lf","X[",i,"]=",Mx[i]);

for (int j=0;j<m;j++){

if(Er[j][i]==1)

printf("%-15s", "Err");

else

printf("%-15.6lf", My[j][i]);

}

cout << endl;

}

}

//---------------------------------------------------------------------------

Пример выполнения программы:

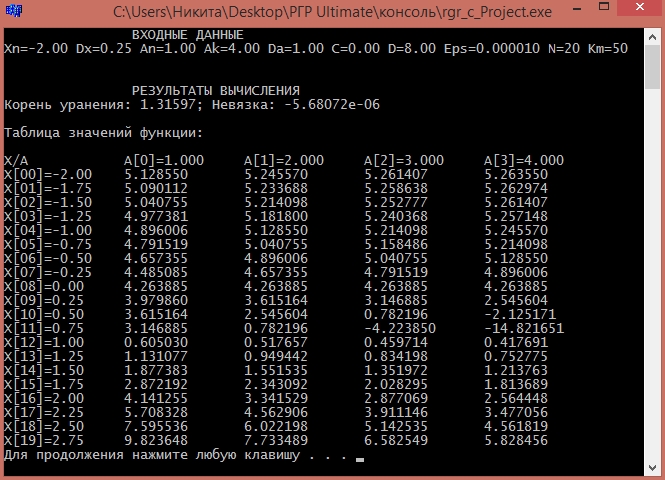


Рис. 8. Результаты выполнения консольного приложения

# **Набор тестов**

Для проверки правильности алгоритмов составим тесты для всех возможных путей вычислений и выполним контрольные просчеты пользуясь независимыми от C++ Builder-среды вычислительными средствами (Калькулятор, WolframAlpha и др.). В данном случае воспользуемся системой «WolframAlpha».

***Тест 1.*** *Проверка правильности вычисления корня нелинейного уравнения.*

Пусть входные данные имеют следующие значения: C=0; D=8; Eps=0,00001; Km=50. Результат вычисления C++ Builder-приложения показан на рис. 7 и равен 1,315971375. Результат вычисления в WolframAlpha показан на рис. 9 и равен на выбранном отрезке 1,31597, а значит при погрешности Eps=0,00001 результат верен.

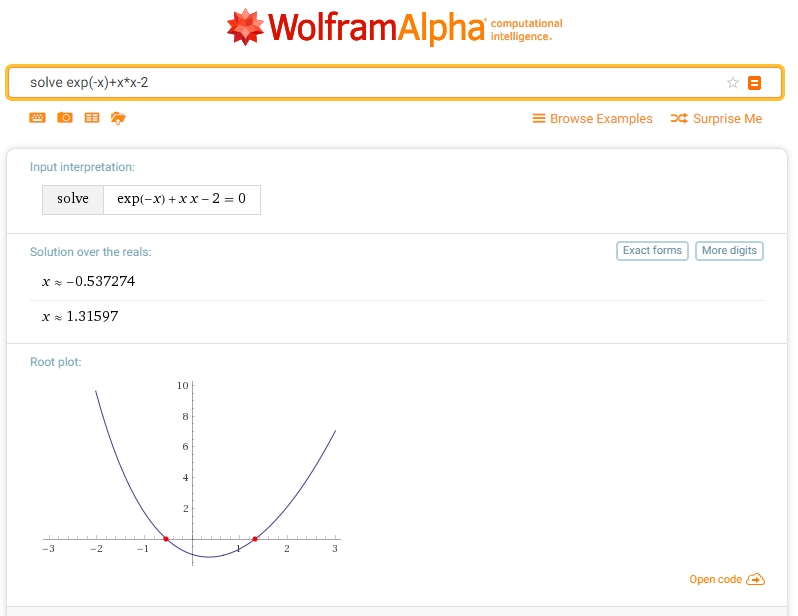


Рис. 9. Результат вычисления корня нелинейного уравнения в WolframAlpha

***Тест 2.*** *Проверка ветви, вычисляющей значение функции при X < 1.*

Пусть входные данные имеют следующие значения: C=0; D=8; Eps=0,00001; Km=50; X=-1,75; A=2. Результат вычисления C++ Builder-приложения показан на рис. 11 и равен -5,23369. Результат вычисления в WolframAlpha показан на рис. 10 и равен 5,23369. Следовательно, значение функции при таких входных параметрах вычислено правильно.

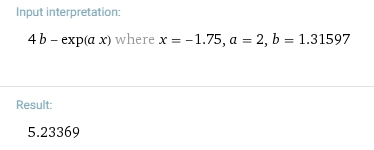


Рис. 10. Результат вычисления функции при X=-1,75 и A=2 в WolframAlpha

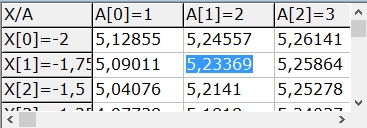


Рис. 11. Результат вычисления функции при X=-1,75 и A=2 в приложении

***Тест 3.*** *Проверка ветви, вычисляющей значение функции при X >= 1.*

Пусть входные данные имеют следующие значения: C=0; D=8; Eps=0,00001; Km=50; X=2,5; A=1. Результат вычисления C++ Builder-приложения равен 7,59554 (рис. 13). Результат вычисления в WolframAlpha показан на рис. 12 и равен 7,59554. Следовательно, значение функции при таких входных параметрах вычислено правильно.

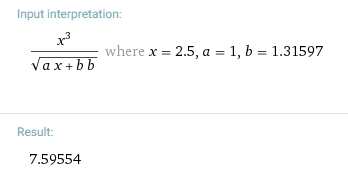


Рис. 12. Результат вычисления функции при X=2,5 и A=1 в WolframAlpha

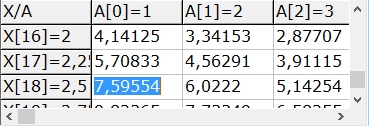


Рис. 13. Результат вычисления функции при X=2,5 и A=1 в приложении

**При вводе неверных значений C++ Builder-программа выводит сообщение об ошибке.**

При недостаточном числе итераций Km выводится сообщение об ошибке (рис. 14):

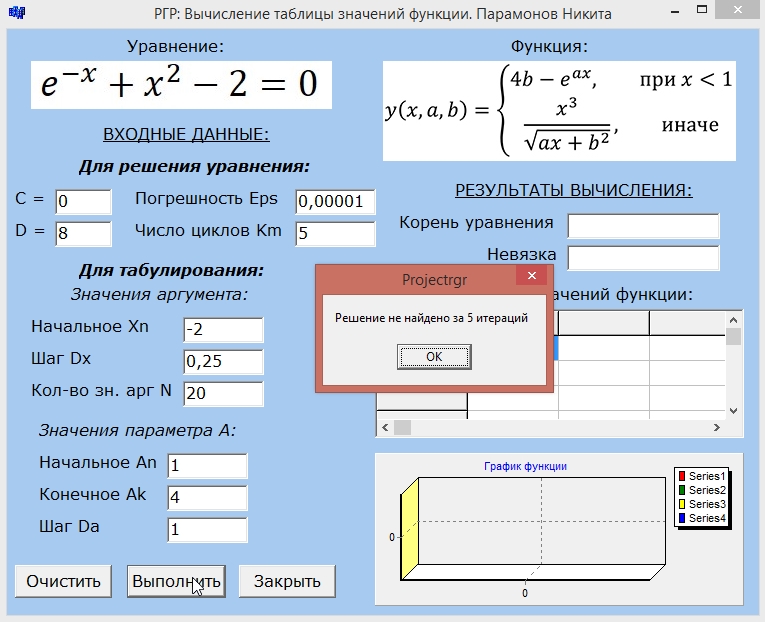


Рис. 14. Сообщение об ошибке при недостаточном числе итераций

Сообщение об ошибке также выводится при погрешности Eps большей или равной 1, или меньшей 0 (рис. 15):

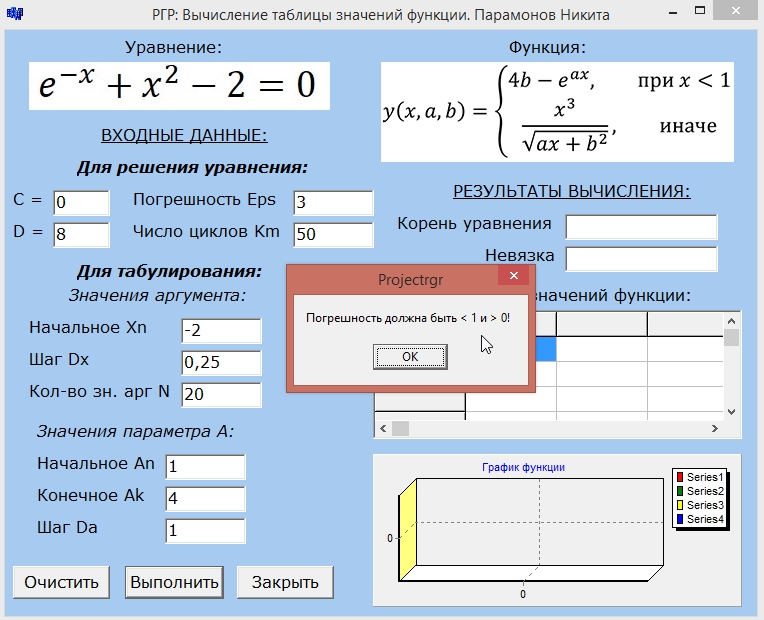


Рис. 15. Сообщение об ошибке при неправильной погрешности Eps

Если при определённых значениях аргумента X и параметра A подкоренное выражение становится меньше 0, программа выводит строку «Err» в таблице значений функции (рис. 16), сообщающую о наличии ошибки в вычислениях (выход за область определения функции):

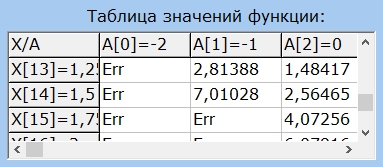


Рис. 16. Ошибка в вычислениях, если подкоренное выражение меньше 0

# **10. Выводы**

Анализ распечатки результатов выполнения программы показывает, что полученные значения функции приближенно совпадают с результатами, полученными для контрольных тестовых примеров с помощью WolframAlpha, что подтверждает работоспособность программы. Следовательно, программа правильно вычисляет заданную функцию по всем ветвям алгоритма и может быть использована для других значений аргументов и параметров функции.

# **11. Список использованной литературы**

* Учебное пособие «Решение алгебраических задач численными методами в среде Delphi» Авторы: Л.В. Кошелькова, А.И. Заковряшин
* Кошелькова Л.В. Программирование в C++BUILDER. Методические указания к лабораторным работам.- МАИ, 2018
* Архангельский А.Я. Программирование в C++ Builder. 7-e изд. — М.: Бином-Пресс, 2010.
* http://cppstudio.com/ - Основы программирования на языках Си и C++ для начинающих