БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники

Кафедра "Электропривод и автоматизация промышленных установок и технологических комплексов"

КУРСОВАЯ РАБОТА

по дисциплине "Информатика"

Тема: "Разработка алгоритма и листинга Паскаль — программы по вычислению сложной функции"

Исполнитель: _____ А. Г. Иванов

(подпись, дата)

студент 2 курса 107613 группы

Руководитель: _____ Д. С. Васильев

(подпись, дата)

Минск 2014

Содержание

Введение

В данной курсовой работе предстоит проанализировать сложную функцию, создать алгоритм программы для вычисления значений сложной функции, вычислить ее значение, вывести результаты в файл в виде таблицы и графика, вычислить ее среднее значение.

Программа написана на свободной реализации языка Паскаль — Free Pascal с помощью Lazarus[1] (среда разработки программного обеспечения на языке Object Pascal для компилятора Free Pascal) и будет адаптирована к работе на операционной системе GNU/Linux.

Первым шагом для разработки программы является создание ее алгоритма, затем происходит программирование ее отдельных блоков, на конечном этапе проходит ее тестирование и анализ результатов.

Бегло просматривая условие задания, видно, что основной упор в программе на владение алгоритмами ветвления, написание функций, а так же работа с массивами данных, владение интегрированной средой разработки IDE(Integrated Development Environment)[2].

В задании требуется использовать формулу Горнера (формула Горнера позволяет вычислить значения полинома в конкретной точке x) при вычислении значения полинома — эта мера требуется для вычисления значений полинома с достаточно большим количеством членов в том случае, если мы знаем зависимость (функцию) по которой изменяются значения коэффициентов при x в соответствующих степенях.

Кроме того, такой метод дает возможность гибко менять количество членов вычисляемого полинома, не внося изменения в программный код.

1. Анализ заданной функции и разработка схемы алгоритма по ее вычислению.

Исходные данные:

$$y_J=\begin{cases} d_3\cdot x^3+d_2\cdot x^2+d_1\cdot x_1+d_0 \quad при \quad x<0 \\ b_5\cdot x^5+b_4\cdot x^4+b_3\cdot x^3+b_2\cdot x^2+b_1\cdot x+b_0 \quad при \quad 0\leqslant x<0.5 \\ ctg(x)+\sqrt[3]{x} \quad при \quad x\geq 0,5 \end{cases},$$

где $j=1,2,3,4\ldots m$; $x=\sin\left(\frac{j}{10}+\frac{\pi}{12}\right)$; $d_i=d_{нач}+\frac{3}{i}$; $b_k=\frac{k+4}{k+1}$; $d_{нач}=0$.

k и i — номера членов соответствующих полиномов;

d_i и b_k — коэффициенты при соответствующих степенях;

m — количество значений функции.

Число значений функции нам задаст параметр j.

Потом будет вычислено число x. Следующим этапом будет сравнение икса по своему значению. По соответствующему x, из условия будет вычисляться своя функция (ветвь).

Число x будет передано в функции, которые будут вычислены в первом и втором случае по формуле Горнера, при этом соответствующие коэффициенты d_i и b_k будут рассчитаны в зависимости от того, какую позицию они занимают. Значение позиции коэффициентов d_0 и b_0 принимаю как первое. Соответственно d_1 и b_1 будут занимать вторую позицию и так далее.

Среднее арифметическое функции будет вычислено и выведено на форму.

Вывод в текстовый файл значений функции реализован в виде таблицы, состоящей из значений аргумента, и соответствующего значения функции.
Программа имеет графический интерфейс GUI (Graphical User Interface) [3].

График функции будет реализован на компоненте TChart, значения x и y будут выводиться в компонент TMemo. Пользователь будет в праве сохранить данные в файл с помощью диалога сохранения файла (компонент TSave).

Данные шаги считаю необходимыми, так как использование готовых компонентов позволит существенно уменьшить программный код, снизит нагрузку на процессор, а так же предоставит возможность сделать программу более качественной, а график наглядным.

2.Программирование отдельных блоков и структур алгоритмов

Ниже можно увидеть алгоритм программы.

Для создания проекта надо запустить среду разработки и нажать Файл → Создать проект → Приложение.

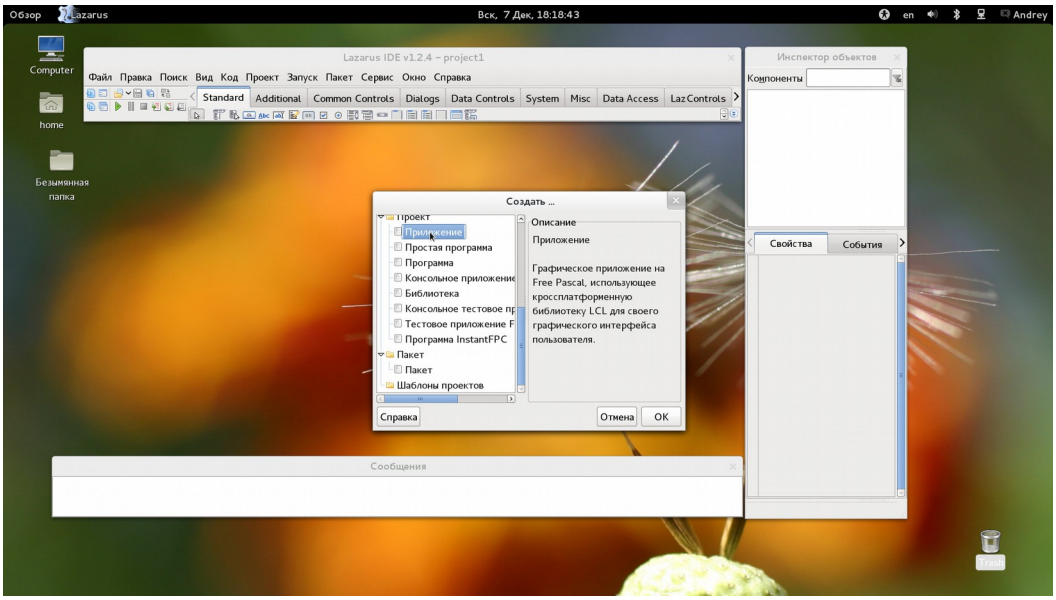

Рисунок 6 — Скриншот примера создания программы.

Будет создано пустое приложение состоящее из главной программы и модуля.

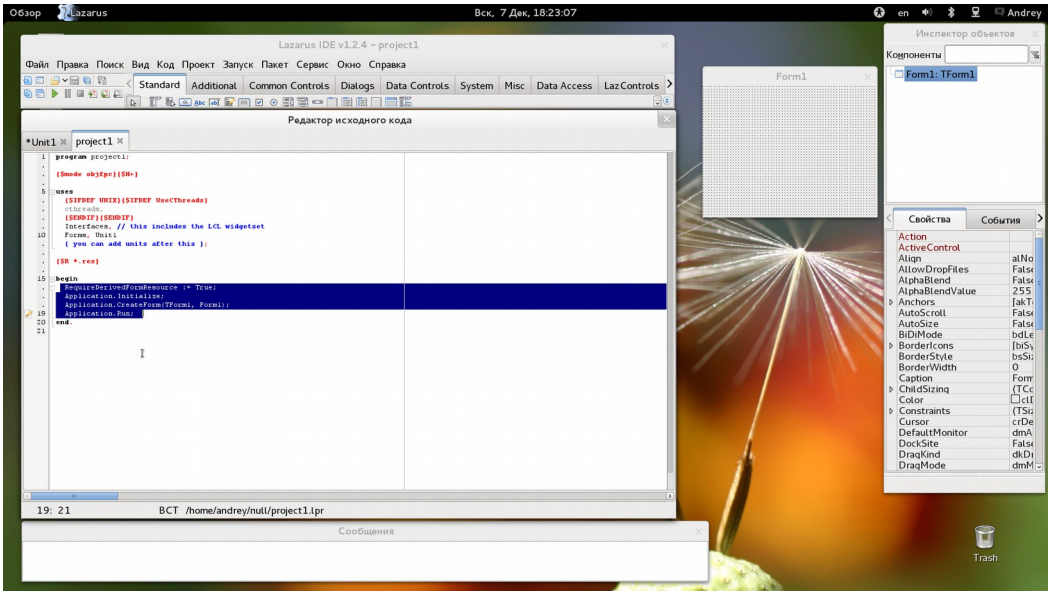

Рисунок 7 — Скриншот пустого проекта программы с графическим интерфейсом.

Интерфейс IDE Lazarus состоит из инспектора обьектов, который позволяет гибко менять свойства компонентов, панели инструментов, где расположены компоненты среды разработки, меню где можно настраивать проект, сборку, отладчик, импорт проектов и других настроек.

Программа состоит из двух частей, основной программы и модуля.

Взглянем на рисунок 1. На нем представлен алгоритм основной

программы. Задача основной программы — создать форму компонента TForm и ее отобразить.

Код автоматически создается средой разработки и его можно увидеть ниже(блоки 2,3,4 рисунок 1):

```
begin
  RequireDerivedFormResource := True;
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Для нас интерес представляет модуль, поэтому взглянем на рисунок 2 — алгоритм модуля Graph. Он состоит из тела модуля, объявленных переменных, двух процедур, и завершения(рис. 2).В блоках 3-4 показываются возможные действия пользователя. Грубо говоря, в зависимость от того на какую кнопку нажмет пользователь, та процедура и выполнится.

У нас есть пустая форма, добавим на нее 2 компонента Tbutton, 3 метки TLabel, компонент графика Tchart, поле ввода Tedit, компонент работы с текстом TMemo, диалог сохранения файла TSave. Для этого в меню среды разработки Lazarus нажмем на нужный компонент и щелкнем на форму.

Таким образом перетаскиваем компоненты, что бы получилось как на рис. 8.

Для того чтобы по нажатию на кнопки у нас выполнялись действия должны обрабатываться события Click.

Щелкаем два раза на первую и вторую кнопки и автоматически среда разработки создаст процедуры, обрабатывающие события OnClick нажатия курсом мыши на обе кнопки.

Так же это можно сделать через Инспектор обьектов в вкладке «События», щелкнув по полю OnClick, будет автоматически добавлена процедура - обработчик события нажатия на клавишу.

На рисунке 8 представлен этап добавления всех компонентов на форму и готовых 2 процедуры нажатия кнопки.

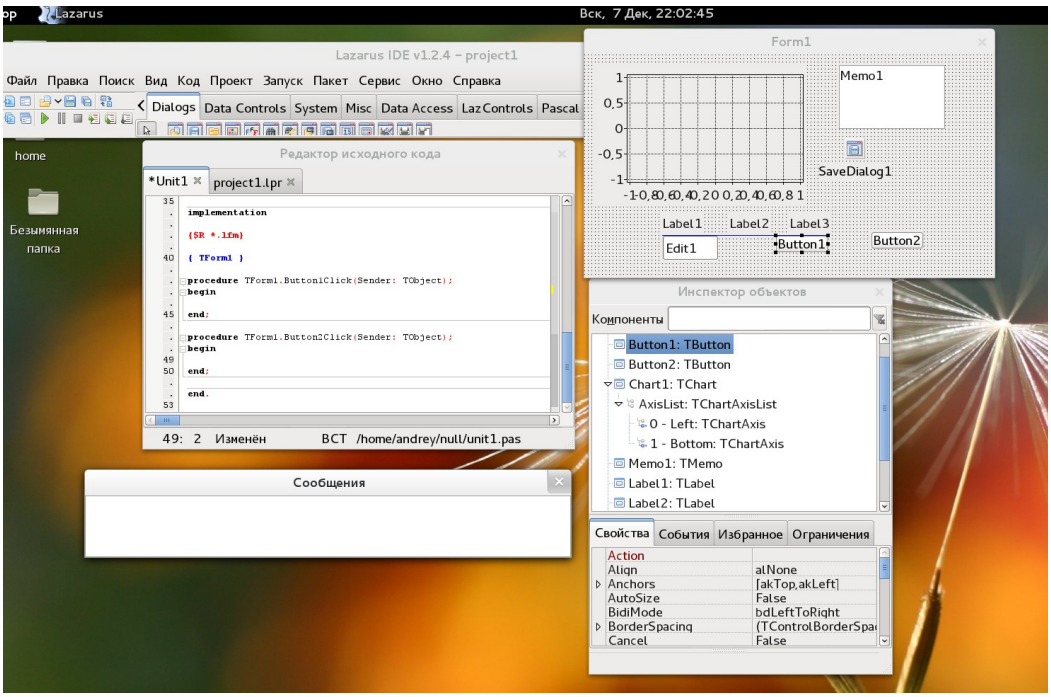

Рисунок 8 — этап создания взаимодействия пользователя и программы.

Добавим на компонент TChart «Серии» или оси графика, для этого два раза щелкнем мышкой на компонент TChart.

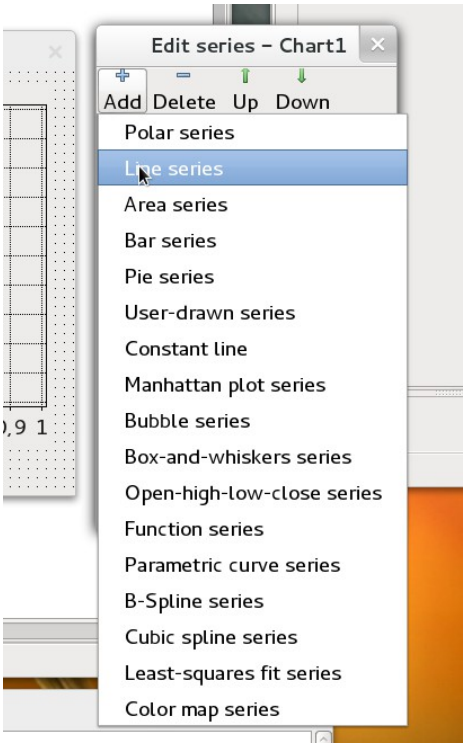

Рисунок 9 — Добавление осей на компонент Tchart.

Переходим к редактору кода.

Для начала необходимо объявить переменные.

Объявление переменных начинаем с специального слова var.

Комментарий в программе рассматривается если перед нам стоит две наклонные черты «//».

```
var
  Form1: TForm1;
  pi:real;                  //Переменная для хранения значения числа Пи.
  x:extended;               //Аргумент сложной функции.
  yj: extended;             //Значение функции вычисляется в цикле.
  m: integer;               //Количество членов
  j: integer;               //Счетчик
  y, ysr: extended;         //Промежуточное значение, среднее арифметическое
  variant:integer;          //Вариант вычисления, ветвь
  st:integer;               //Степень многочлена
  d_i:koef_auto;            //Коэфициент при x в соответствующей степени.
  d_auto:extended;          //Промежуточная переменная
  b_k:koef_auto1;           //Коэфициент при x в соответствующей степени.
  b_auto:extended;          //Промежуточная  переменная.
  ver:integer;              //Вспомогательная переменная.
  Xstr:string[7];           //Строка которая хранит число x
  YStr:string[7];           //Строка которая хранит число y
  Sinput:string[38];        //Строка которая хранит число x
  Input:string[38];         //Строка, хранящая форматированный вывод x и y.
  Separator:String[35]; //Строка разделителя
  srednee:string[5];        //Строка, хранящая среднее арифметическое.
  txtfile:text;             //Файловая переменная
  i:integer;                //Счетчик.
```

Весь код, который будет написан в процедуре нажатия на кнопку будет выполнен. Поэтому, идем в процедуру TForm1.Button1Click.

Пропишем начальные значения, которые нам необходимы(рис. 3, блок 2).

```
 pi:=3.14;
 m:=2;
```

```
j:=1;
yj:=0; // Обнуляем значение функции.
Y:=0;  //Обнуляем промежуточную переменную.
Ysr:=0;  //Обнуляем переменную.
Separator:='------------------------';
```

Для ввода числа(рис.3, блок 3) m в программу у нас будет использоваться текстовое поле ввода TEdit, которой хранит текст строкового типа (В свойстве Tedit.Text), для использования в программе нам нужно эту строку преобразовать в целочисленный тип integer.

Функция которая это производит называется StrtoInt(). В скобках указывается строковая переменная, функция при вызове возвратит целочисленное значение. Для этого впишем следующий код(рис 3, блок 4):

```
m:=StrToInt(Edit1.Text);
```

Если нам надо будет несколько раз строить график за один запуск программы нам надо стирать предыдущий график (рис 3, блок 5).

Данной действие выполняет код Chart1LineSeries1.Clear.

Далее необходимо очистить поле компонента Tmemo от предыдущих результатов, если нам придется несколько раз вычислять функцию.

Данное действие реализуется функцией Form1.Memo1.Clear ;

Выведем первым образом «Шапку» таблицы в поле компонента TMemo.

Ввод текста в компонента осуществляется построчно. Чтобы не создавать строку самостоятельно и не использовать лишний массив для работы со строками используем функцию автоматического создания новой строки при ее вызове — Form1.Memo1.Lines.Append(переменная);.

Вывод шапки таблицы реализуется следующим кодом:

```
Form1.Memo1.Lines.Append(Separator);
SInput:='|X=         | Y=         |';
Form1.Memo1.Lines.Append(SInput);
Form1.Memo1.Lines.Append(Separator);
```

Так как функция у нас будет много раз вычисляться нам нужен оператор

цикла for(рисунок 3 блок 6).

```
for j:=1 to m do begin
                …….
                 end;
```

Значение j будет пробегать от 1 до m.

Первая и вторая ветвь будет вычисляться по формуле Горнера, поэтому необходимо написать функцию (рис.3, блок 11 ветвь 1, блок 11 ветвь 2).

Формула Горнера позволит вычислять сколь угодно огромные полиномы в какой либо точке x.

Формула Горнера имеет вид $y=((a_1\cdot x+a_2)\cdot x+....+a_n)\cdot x+a_{(n+1)}$. (1)

Только при прямом следовании коэффициентов и возрастании счетчика j, коэффициент $a_{(n+1)}$ соответствует самой младшей позиции — позицией без x.

А коэффициент a_1 будет при x^n .

Пример: $a_1\cdot x^n+a_2\cdot x^{(n-1)}+a_3\cdot x^{(n-2)}+....+a_n\cdot x+a_{(n+1)}$ (2)

Так как я хочу чтобы номера коэффициентов соответствовали x с своей степени, я переписал формулу в виде:

$$y=((a_{(n+1)}\cdot x+...+a_n)\cdot x+a_2)\cdot x+a_1 \quad , \qquad (3)$$

от этого ее значение не изменится.

И коэффициенты будут идти от старшей позиции к младшей, при этом полином будет «раскрываться» с x в самой большой степени с коэффициентом с самой старшей позицией, при этом в конце ко всему выражению прибавится коэффициент с самой младшей позицией a_1 .

Порядок следования коэффициентов из массива будет обратным.

Выражение будет иметь тогда вид:

$$a_{(n+1)}\cdot x^n+a_n\cdot x^{(n-1)}+....+a_2\cdot x+a_1 \quad , \qquad (3)$$

Алгоритм функции Gorner1 и Gorner2 представлен на рисунке 5 и 6.

В функцию передается x и st+1 — переменная, определяющая количество коэффициентов.

В блоке 3 рисунка 5 и 6 происходит сравнение счетчика I с количеством

коэффициентов многочлена, так как коэффициентов на один больше чем количество иксов.

Далее они в цикле вычисляются и записываются в массив. Блок 3 и 4 на рисунке 5 и 6.

```
 for i:=1 to st+1 do begin
                d_auto:=d_nach+3/i;
                d_i[i]:=d_auto; end;
```

Аналогично в функции Gorner2:

```
for k:=1 to st+1 do begin
                b_auto:=b_nach+(k+4)/(k*k+1);
                b_k[k]:=b_auto; end;
```

Затем надо воспользоваться формулой Горнера и написать код который будет ей соответствовать:

Для функции Gorner1:

```
 y:=0;
 for i:=st+1 downto 1 do y:=d_i[i]+y*x;
 Gorner_1:=y;
```

Для функции Gorner2:

```
y:=0;
for k:=st+1 downto 1 do y:=b_k[k]+y*x;
Gorner_2:=y;
```

Первая строчка обнуляет значение функции, вторая - цикл.

При этом к коэффициенту старшей степени прибавляется выражение y*x. Сначала y = 0.

Затем цикл повторяется, при этом берется коэффициент на одну позицию ниже, и прибавляется значение предыдущего вычисления.

Таким образом y в конце вычисления имеет значение полинома в точке икс. При этом последний коэффициент с позицией 1 будет прибавлен ко всему выражению.

Третья строчка нужна чтобы присвоить процедуре значение y — результат

вычисления всего полинома.

Когда процедура выполнит действия, она возвратит значение равному y.

У нас теперь есть функции, вычисляющие 2 ветви, 3-ю ветвь можно вычислить не используя функцию, поэтому можем приступить к написанию алгоритма ветвления вычисления сложной функции.

Далее идем в процедуру нажатия на первую кнопку TForm1.Button1Click

На рис.3 блокам 6 - 9 соответствует код:

```
for j:=1 to m do begin
x:=sin((j/10)+pi/12);
if x<0 then { d3·x^3+d2·x^2+d1·x+d0 } begin
                                variant:=1;
                                writeln('Вариант вычисления ', variant);
                                st:=3;
                                yj:=Gorner_1(x, st);
                                 writeln('yj(',x:5:4,')=',yj:0:4);
                                 Chart1LineSeries1.AddXY(x ,yj,'',clBlue);
                                     end
      else              begin
                        if x>=0.5 then { ctg(x)+∛x } begin
                                          variant:=3;
                                          writeln('Вариант вычисления ', variant);
                                          yj:=(1/tan(x))* (Exp((1/3)*Ln(x)));
                                         writeln('yj(',x:5:4,')=',yj:0:4);
                                         Chart1LineSeries1.AddXY(x ,yj,'',clRed);
                                          end
                                else { b5·x^5+b4·x^4+b3·x^3+b2·x2+b1·x+b0 }
                                          begin
                                          variant:=2;
                                          writeln('Вариант вычисления ', variant);
                                          st:=5;
```

```
                                        yj:=Gorner_2(x, st);
                                    Chart1LineSeries1.AddXY(x ,yj,'',clGreen);
                                     writeln('yj(',x:5:4,')=',yj:0:4);
                                      end
                          end;
y:=y+yj;
 XStr:=FloatToStr(x);
YStr:=FloatToStr(yj);
Input:='|'+'X= '+Xstr+' | '+'Y= '+Ystr+'|';
Form1.Memo1.Lines.Append(input);
Form1.Memo1.Lines.Append(separator);
 end;
```

В блоке 8 и 9 идет фильтрация x по ветвям.

В блоке 10 идет присвоение переменной значению ветви.

В блоках 11 идет вычисление сложной функции ветвей 1-3.

В блоке 12 идет передача компоненту Tchart значений x и y., выводятся на стандартный поток вывода с помощью writeln.

Для передачи компоненту Tchart x и y используется функция:

```
Chart1LineSeries1.AddXY(x ,yj,'',clBlue);
```

Сначала передается x, потом y, потом передается цвет точки. Я сделал цвет точек для каждой ветви разный. Так будет нагляднее по какой ветви считается функция.

Далее идет блок 13 рис. 3. В нем выполняется накопление суммы всех значений.

В блоке 14 рис. 3 x и y преобразуются в строку с помощью функции FloatToStr().

Это необходимо, чтобы форматированно вывести на компонент Tmemo таблицы результатов.

В блоке 14 две строки складываются в одну, и передаются в компонент Tmemo.

Как только цикл for выполнится, будет выполняться блок 16.

Код блока 16:

```
ysr:=y/m;
srednee:=FloatToStr(ysr);
label2.caption:=srednee;
```

В нем вычисляется среднее арифметическое, значение переводится в строку и присваивается свойству TLabel2.Caption компонента TLabel2.

Во время работы программы значение среднего арифметического отобразится на форме в виде текстовой надписи.

По умолчанию метки Tlabel 1,2 я сделал не видимыми(visible=false), до того момента пока программно это свойство не будет изменено.

В блоке 17 меняется свойство метки Tlabel 1,2 на видимое.

Этому действию соответствует код:

```
label1.visible:=true;
label2.visible:=true;
```

И так, мы закончили разбор алгоритма главной программы (рис.1) TForm1.Button1Click (рис.3), Gorner_1 (рис. 5), Gorner_2 (рис.6).

Остается рассказать про процедуру TForm1.Button2Click (рис.4) в модуле (рис. 2, блок 4).

При нажатии на кнопку исполняется код, размещенный в ней.

Первым выполняется блок 2, отображается диалог сохранения файла компонент(TSave).

Код: SaveDialog1.Execute;

Далее надо проассоциировать файловую переменную с файлом, путь к которому возвращает компонент SaveDialog1.

Код:AssignFile(TxtFile, SaveDialog1.FileName);

В блоке 4 открываю файл для записи. Код:

```
Rewrite(TxtFile);
```

В блоке 5 расположен цикл for. Обращение к свойству Memo1.Lines.Count возвратит количество строк занятых под таблицу значений.

Вывод построчно из компонента TMemo в файл осуществляется в блоке 6, код:

```
Writeln(TxtFile, Memo1.Lines[i-1]);
```

Полный код процедуры нажатия на кнопку Button2 выглядит следующим образом:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
SaveDialog1.Execute;
AssignFile(TxtFile, SaveDialog1.FileName);
Rewrite(TxtFile);
for i:=1 to Memo1.Lines.Count do
begin
Writeln(TxtFile, Memo1.Lines[i-1]);
end;
closefile(TxtFile);
end;
```

Таким образом я расписал алгоритмы своей програмы.

В этой главе я не упомянул зачем нужны Tlabel1 и Tlabel3.

Они имеют статичную информацию. В инспекторе объектов среды разработки я в поле свойства Caption метки Tlabel1 записал текст «Среднее значение =», в свойство метки Tlabel3 записал «Введите параметр m ниже:» и расположил таким образом, чтобы было наглядно что куда нужно вводить, и какие данные выводятся на форму. В поле компонента TMemo по умолчанию введен текст приветствия и справки с помощью инспектора обьектов.

В текстовом поле Edit1.Text по умолчанию введено число 2, что соответствует минимальному количеству значений для построения прямой линии. Чтобы размер формы нельзя было изменять, выставляю свойству BorderStyle значение как bsSingle. Такими же методами настраивается цвет линии графика, толщины линии и т.д.

3. Главный листинг программы в соотсветствии с разработанным алгоритмом.

Прежде чем дать готовый листинг программы я хочу описать структуру любого проекта, создаваемого в среде разработки Lazarus.

Любой проект в Lazarus—это совокупность файлов, из которых создаётся единый выполняемый файл.

В простейшем случае список файлов проекта имеет вид:

файл описания проекта (.lpi), файл проекта(.lpr), файл ресурсов(.lrs), модуль формы(.lfm), программный модуль(.pas).

Файлы отвечающие за код *.lpr и *.pas. Ниже представляю содержание файла project.lpr:

```
program project1;
{$mode objfpc}{$H+}
uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Interfaces, // this includes the LCL widgetset
  Forms, tachartlazaruspkg, Graph
  { you can add units after this };
{$R *.res}
begin
  RequireDerivedFormResource := True;
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Файл Graph.pas:

```
unit Graph;
{$mode objfpc}{$H+}
interface
uses
  Classes, SysUtils, FileUtil, TAGraph, Forms, Controls, Graphics, Dialogs,
  StdCtrls, crt, CustApp, math, unix, Buttons, Menus, EditBtn,
  CheckLst, Interfaces, ExtCtrls, tachartlazaruspkg, TASeries;
type
  { TForm1 }
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Chart1: TChart;
    Chart1LineSeries1: TLineSeries;
    Edit1: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Memo1: TMemo;
    SaveDialog1: TSaveDialog;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { private declarations }
  public
    { public declarations }
  end;
  type koef_auto1=array[0..25] of extended;
  type  koef_auto=array[0..25] of real;
```

```
var
  Form1: TForm1;
  pi:real;
  x:extended;
  yj: extended;              {аргумент вычисляется в цикле, значение функции}
  m: integer;                {количество членов}
  j: integer;                {счетчик}
  y, ysr: extended;          {промежуточное значение, среднее арифметическое}
  variant:integer;           {вариант вычисления}
  st:integer;                //степень многочлена
  d_i:koef_auto;
  d_auto:extended;
  b_k:koef_auto1;
  b_auto:extended;
  ver:integer;
  XStr:string[7];
  YStr:string[7];
  Sinput:string[38];
  Input:string[38];
  Separator:String[35];
  srednee:string[5];
  txtfile:text;
  i:integer;
implementation
function Gorner_1(x:real; st:integer):real;
 var i: integer;
    y:real;
    d_nach:real;
 begin
 d_i[0]:=0;
```

```
 pi:=3.14;
 d_nach:=0;
 for i:=1 to st+1 do begin
                 d_auto:=d_nach+3/i;
                 d_i[i]:=d_auto;
                 end;
 y:=0;
 for i:=st+1 downto 1 do y:=d_i[i]+y*x;
 Gorner_1:=y;
 end;
function Gorner_2(x:real; st:integer):real;
var k: integer;
    y:extended;
    b_nach:real;
begin
b_k[0]:=0;
b_nach:=0;
for k:=1 to st+1 do begin
                 b_auto:=b_nach+(k+4)/(k*k+1);
                 b_k[k]:=b_auto;
                 end;
y:=0;
for k:=st+1 downto 1 do y:=b_k[k]+y*x;
Gorner_2:=y;
end;
{$R *.lfm}
{ TForm1 }
procedure TForm1.Button1Click(Sender: TObject);
 begin
 pi:=3.14;
```

```
m:=2;//По варианту задания
j:=1;
yj:=0;
y:=0;
ysr:=0;
Chart1LineSeries1.Clear;
Form1.Memo1.Clear;
m:=StrToint(Edit1.Text);
Separator:='------------------------';
Form1.Memo1.Lines.Append(Separator);
SInput:='|X=        | Y=        |';
Form1.Memo1.Lines.Append(SInput);
Form1.Memo1.Lines.Append(Separator);
for j:=1 to m do begin
            x:=sin((j/10)+pi/12);
            if x<0 then { d3·x^3+d2·x^2+d1·x+C0 } begin
                                variant:=1;
                                writeln('Вариант вычисления ', variant);
                                writeln('                              ');
                                st:=3;
                                yj:=Gorner_1(x, st);
                                writeln('yj(',x:5:4,')=',yj:0:4);
                                Chart1LineSeries1.AddXY(x ,yj,'',clBlue);
                                 writeln('Коэфиценты при икс');
                                 for ver:=1 to st+1 do writeln('',d_i[ver]);
                                 writeln(Separator);
                                end
                    else                 begin
                        if x>=0.5 then { ctgx+∛x }   begin
                                            variant:=3;
```

```
                writeln('Вариант вычисления ', variant);
                writeln('                    ');
                yj:=(1/tan(x))+(Exp((1/3)*Ln(x)));
                writeln('yj(',x:5:4,')=',yj:0:4);
                writeln(Separator);
                Chart1LineSeries1.AddXY(x ,yj,'',clRed);
                end
        else { b5·x^5+b4·x^4+b3·x^3+b2·x^2+b1·x+b0 }
                begin
                variant:=2;
                writeln('Вариант вычисления ', variant);
                writeln('                    ');
                 st:=5;
                 yj:=Gorner_2(x, st);
                 writeln('yj(',x:5:4,')=',yj:0:4);
                 writeln('Коэфициенты при икс');
                 for ver:=1 to st do writeln('',b_k[ver]);
                 writeln(Separator);
                 Chart1LineSeries1.AddXY(x ,yj,'',clGreen);
                end
                end;
                y:=y+yj;
                XStr:=FloatToStr(x);
                YStr:=FloatToStr(yj);
                Input:='|'+'X= '+Xstr+' | '+'Y= '+Ystr+'|';
                Form1.Memo1.Lines.Append(input);
                Form1.Memo1.Lines.Append(separator);
        end;
ysr:=y/m;
srednee:=FloatToStr(ysr);
```

```
label2.caption:=srednee;
label1.visible:=true;
label2.visible:=true;
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
SaveDialog1.Execute;
AssignFile(TxtFile, SaveDialog1.FileName);
Rewrite(TxtFile);
for i:=1 to Memo1.Lines.Count do
begin
Writeln(TxtFile, Memo1.Lines[i-1]);
end;
closefile(TxtFile);
end;
end.
```

Остальные файлы - вспомогательные части проекта Lazarus несут некоторые свойства элементов, которые не являются функциональными (не несут исполняемый код) (изображения, настройки формы и т. д.).

Любой код который не вошел в алгоритм, и не описан, однако представлен в полном листинге программы предназначен для проверки результатов работы программы (вывод промежуточных результатов вычисления в стандартный поток вывода (writeln) и является устранимым. Функционал программы от него не зависит.

4. Анализ полученных результатов.

Так как в программу встроен недокументированный мной код, он выводит в стандартный поток вывода результаты работы программы. Увидеть эти данные можно запустив программу из коммандной строки.

В результате работы по первой ветви программа вывела:

$yj(-0.0201)=2.9703$

Коэффициенты при икс:

$d_1=3.00000000000000E+000$
$d_2=1.50000000000000E+000$
$d_3=1.00000000000000E+000$
$d_4=7.50000000000000E-001$

В результате работы по второй ветви программа вывела:

$yj(0.3709)=3.0737$

Коэффициенты при икс:

$b_0=2.5000000000000000E+0000$
$b_1=1.2000000000000000E+0000$
$b_2=6.9999999999999996E-0001$
$b_3=4.7058823529411764E-0001$
$b_4=3.4615384615384615E-0001$
$b_5=2.7027027027027029E-0001$

В результате работы по третьей ветви программа вывела:

$yj(0.9246)=1.7285$

Анализ результатов представляет собой заключительный этап тестирования продукта.

Тестирование включает в себя проверку правильности работы программного продукта, устранение ошибок, выявление багов[7].

В правильности вывода данных в файл и в текстовое поле интерфейса программы я не сомневаюсь, важно проверить сами значения.

Для вычисления результатов и их проверки я буду использовать программный продукт wXMaxima.

Maxima — свободная система компьютерной алгебры, написанная на языке Common Lisp. Maxima имеет широчайший набор средств для проведения

аналитических вычислений, численных вычислений и построения графиков. По набору возможностей система близка к таким коммерческим системам, как Maple и Mathematica.

В то же время она обладает высочайшей степенью переносимости: может работать на всех основных современных операционных системах на компьютерах, начиная от наладонных, и вплоть до самых мощных.

Maxima имеет несколько графических интерфейсов пользователя и надстроек: XMaxima (включен в поставку во многих ОС), wxMaxima (основан на wxWidgets) и других, а также может работать в режиме командной строки (используя псевдографику).

Результаты вычислений приведены ниже:

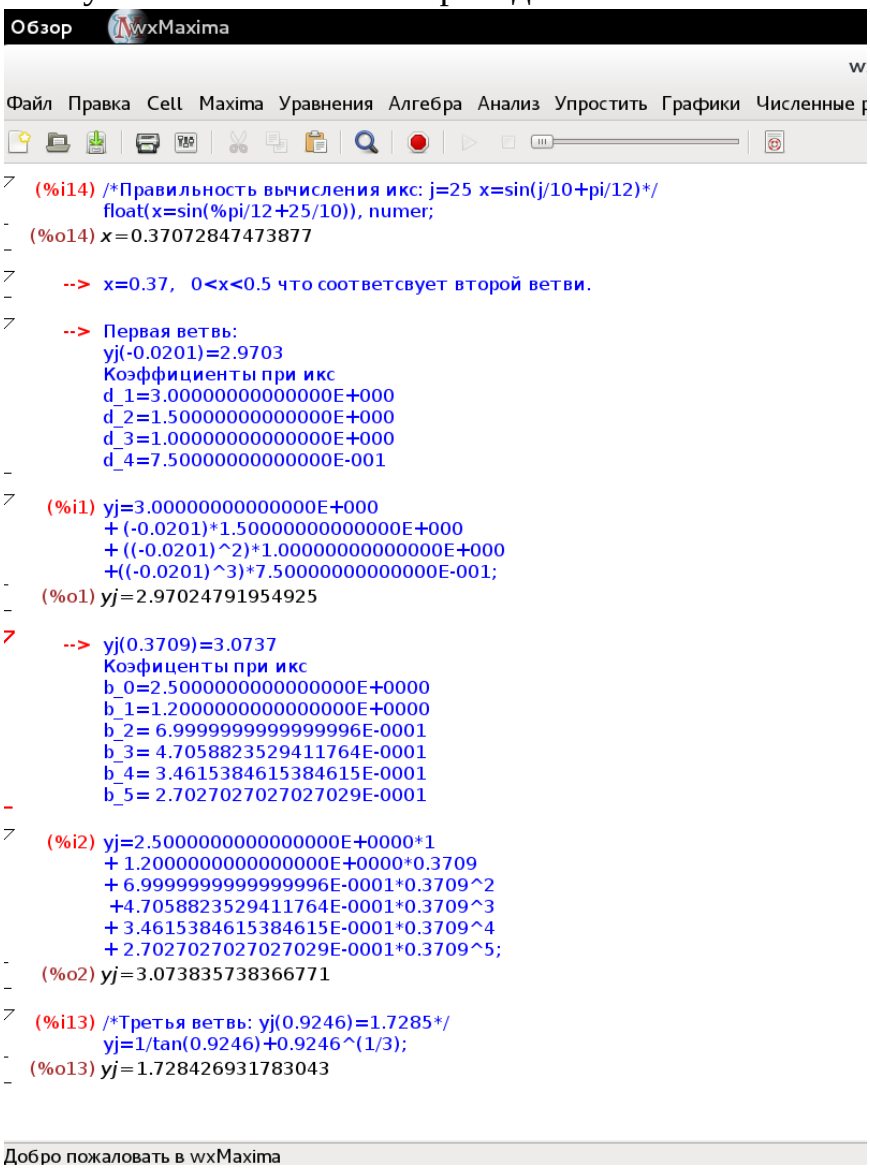

Рисунок 10 — результат проверки вычислений сделанных Паскаль-программой.

Заключение

В результате проведенной работы имеется программа, способная вычислять сложную функцию, выводить ее график, выводить значения прямо на элементы интерфейса, а так же импортировать данные в текстовый файл в любое место по выбору пользователя.

Данный курсовой проект помог мне применить высокоуровневую среду разработки IDE Lazarus, изучить компоненты Tchart, Tmemo, которые я успешно применил в решении поставленной задачи, а так же данная работа помогла применить свои знания компонентов Tedit, Tlabel, Tbutton, TSave программирования программы с графическим интерфейсом для вычисления сложной функции.

Ниже привожу скриншот своей программы:

Рисунок 11 — снимок с экрана окна программы для вычисления сложной функции.

Список использованных источников

1. Сайт, https://www.lazarus.freepascal.org/, официальный сайт программы разработки IDE Lazarus.
2. Сайт, https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8, свободная энциклопедия.
3.Сайт, https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F, свободная энциклопедия.
3. Сайт, http://wiki.freepascal.org/Lazarus_Documentation/ru, официальная документация проекта IDE Lazarus.
4. Сайт, http://wiki.lazarus.freepascal.org/TAChart, документация по использованию компонентов Tchart и др.
5. Самоучитель по программированию на Free Pascal и Lazarus, Алексеев Е.Р., Чеснокова О.В., Кучер Т.В.
6. Free Pascal и Lazarus: Учебник по программированию, Алексеев Е.Р., Чеснокова О. В., Кучер Т. В.
7. Сайт, https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B3, свободная энциклопедия.
8. Основы алгоритмизации и программирования инженерных задач на Паскале, БНТУ, Павлович С.Н, БНТУ 2010.