## Міністерство освіти і науки України

## Львівський національний університет імені Івана Франка

## Факультет електроніки та комп’ютерних технологій

## 

## 

# **Звіт**

## про виконання лабораторної роботи № 5

## з курсу “Алгоритмізація та програмування”

## «Віконний проект методів розв’язання нелінійних рівнянь»

## 

## 

## 

### Виконав:

### ст. гр. ФЕІ-11

### Стасів Петро

### Перевірив:

### доц. Хвищун І.О.

### Львів-2021

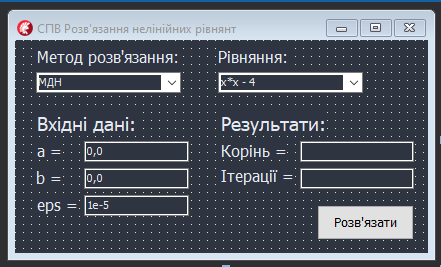
## **Звіт**

**Мета:** у середовищі Delphi написати програму з графічним інтерфейсом яка буде розв’язувати нелінійні рівняння двома способами МДН та методом Ньютона.

### 

### **Виконання лабораторної роботи:**

1. Створення графічного інтерфейсу:



2) Функції які реалізують два методи розв’язання:

**function** BisectionMethod(**const** a : **double**; **const** b : **double**;

**const** eps : **double**; **const** polynomialFunc : PolynomialFunc) : MethodResult;

**begin**

**var** resultV : MethodResult;

**if**(Abs(PolynomialFunc(a)) <= eps) **then**

**begin**

resultV.Root := a;

resultV.Iterations := 1;

BisectionMethod := resultV;

exit;

**end**

**else** **if**(Abs(PolynomialFunc(b)) <= eps) **then**

**begin**

resultV.Root := b;

resultV.Iterations := 1;

BisectionMethod := resultV;

exit;

**end**;

**var** c := 0.0;

**var** iterationsCount := 0;

**var** mA := a;

**var** mB := b;

**while** **True** **do**

**begin**

c := (mA + mB) / 2.0;

**if**(Abs(polynomialFunc(c)) <= eps)

**or** (iterationsCount >= IterationsLimit) **then**

**begin**

resultV.Root := c;

resultV.Iterations := iterationsCount;

BisectionMethod := resultV;

exit;

**end**;

**if**(polynomialFunc(mA) \* polynomialFunc(c) < 0.0) **then**

**begin**

mB := c

**end**

**else**

**begin**

mA := c;

**end**;

iterationsCount := iterationsCount + 1;

**end**;

**end**;

**function** NewtonMethod(**const** a : **double**; **const** b : **double**;

**const** eps : **double**; **const** polynomialFunc : PolynomialFunc) : MethodResult;

**begin**

**var** x := a;

**if** polynomialFunc(x) \* GetFunctionDerivative2(x, eps, polynomialFunc) < 0.0 **then**

**begin**

x := b;

**if** polynomialFunc(x) \* GetFunctionDerivative2(x, eps, polynomialFunc) < 0.0 **then**

Writeln('For the specified interval result is not guaranteed!');

**end**;

**var** iterations := 0;

**while** **True** **do**

**begin**

**if** iterations > IterationsLimit **then**

break;

**var** d := polynomialFunc(x) / GetFunctionDerivative(x, eps, polynomialFunc);

x := x - d;

iterations := iterations + 1;

**if** Abs(d) <= Eps **then**

break;

**end**;

**var** resultV : MethodResult;

resultV.Root := x;

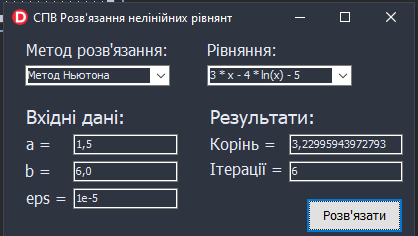
resultV.Iterations := iterations;

NewtonMethod := resultV;

**end**;

**Тестування:**

### 



### **Текст програми:**

**interface**

**uses**

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls;

**const** IterationsLimit = 1024;

**type** MethodResult = **record**

Root : **double**;

Iterations : **integer**;

**end**;

**type** PolynomialFunc = **function**(**const** x : **double**) : **double**;

**type** MethodFunc = **function**(**const** a : **double**; **const** b : **double**;

**const** eps : **double**; **const** polynomialFunc : PolynomialFunc) : MethodResult;

**type**

TForm1 = **class**(TForm)

MethodComboBox: TComboBox;

PolynomialComboBox: TComboBox;

Label1: TLabel;

Label2: TLabel;

Label3: TLabel;

Label4: TLabel;

Label5: TLabel;

AValueEdit: TEdit;

BValueEdit: TEdit;

Label6: TLabel;

EpsValueEdit: TEdit;

Label7: TLabel;

Label8: TLabel;

Label9: TLabel;

RootValueEdit: TEdit;

IterationsValueEdit: TEdit;

SolveButton: TButton;

**procedure** SolveButtonClicked(Sender: **TObject**);

**end**;

**var**

Form1: TForm1;

**implementation**

*{$R \*.dfm}*

**function** Polynomial1(**const** x : **double**) : **double**;

**begin**

Polynomial1 := x \* x - 4.0;

**end**;

**function** Polynomial2(**const** x : **double**) : **double**;

**begin**

Polynomial2 := 3.0 \* x - 4 \* ln(x) - 5.0;

**end**;

**function** GetFunctionDerivative(**const** x : **double**; **const** eps : **double**; **const** polynomialFunc : PolynomialFunc) : **double**;

**begin**

**const** d = Eps / 1000.0;

GetFunctionDerivative := (polynomialFunc(x + d) - polynomialFunc(x)) / d;

**end**;

**function** GetFunctionDerivative2(**const** x : **double**; **const** eps : **double**; **const** polynomialFunc : PolynomialFunc) : **double**;

**begin**

**const** d = Eps / 1000.0;

GetFunctionDerivative2 := (GetFunctionDerivative(x + d, eps, polynomialFunc) - GetFunctionDerivative(x, eps, polynomialFunc)) / d;

**end**;

**function** BisectionMethod(**const** a : **double**; **const** b : **double**;

**const** eps : **double**; **const** polynomialFunc : PolynomialFunc) : MethodResult;

**begin**

**var** resultV : MethodResult;

**if**(Abs(PolynomialFunc(a)) <= eps) **then**

**begin**

resultV.Root := a;

resultV.Iterations := 1;

BisectionMethod := resultV;

exit;

**end**

**else** **if**(Abs(PolynomialFunc(b)) <= eps) **then**

**begin**

resultV.Root := b;

resultV.Iterations := 1;

BisectionMethod := resultV;

exit;

**end**;

**var** c := 0.0;

**var** iterationsCount := 0;

**var** mA := a;

**var** mB := b;

**while** **True** **do**

**begin**

c := (mA + mB) / 2.0;

**if**(Abs(polynomialFunc(c)) <= eps)

**or** (iterationsCount >= IterationsLimit) **then**

**begin**

resultV.Root := c;

resultV.Iterations := iterationsCount;

BisectionMethod := resultV;

exit;

**end**;

**if**(polynomialFunc(mA) \* polynomialFunc(c) < 0.0) **then**

**begin**

mB := c

**end**

**else**

**begin**

mA := c;

**end**;

iterationsCount := iterationsCount + 1;

**end**;

**end**;

**function** NewtonMethod(**const** a : **double**; **const** b : **double**;

**const** eps : **double**; **const** polynomialFunc : PolynomialFunc) : MethodResult;

**begin**

**var** x := a;

**if** polynomialFunc(x) \* GetFunctionDerivative2(x, eps, polynomialFunc) < 0.0 **then**

**begin**

x := b;

**if** polynomialFunc(x) \* GetFunctionDerivative2(x, eps, polynomialFunc) < 0.0 **then**

Writeln('For the specified interval result is not guaranteed!');

**end**;

**var** iterations := 0;

**while** **True** **do**

**begin**

**if** iterations > IterationsLimit **then**

break;

**var** d := polynomialFunc(x) / GetFunctionDerivative(x, eps, polynomialFunc);

x := x - d;

iterations := iterations + 1;

**if** Abs(d) <= Eps **then**

break;

**end**;

**var** resultV : MethodResult;

resultV.Root := x;

resultV.Iterations := iterations;

NewtonMethod := resultV;

**end**;

**procedure** TForm1.SolveButtonClicked(Sender: **TObject**);

**begin**

**var** polynomialFunc : PolynomialFunc;

**case** PolynomialComboBox.ItemIndex **of**

0: **begin**

polynomialFunc := Polynomial1;

**end**;

1 : **begin**

polynomialFunc := Polynomial2;

**end**;

**end**;

**var** methodFunc : MethodFunc;

**case** MethodComboBox.ItemIndex **of**

0: **begin**

methodFunc := BisectionMethod;

**end**;

1: **begin**

methodFunc := NewtonMethod;

**end**;

**end**;

**var** a := StrToFloat(AValueEdit.Text);

**var** b := StrToFloat(BValueEdit.Text);

**var** eps := StrToFloat(EpsValueEdit.Text);

**var** res := methodFunc(a, b, eps, polynomialFunc);

RootValueEdit.Text := FloatToStr(res.Root);

IterationsValueEdit.Text := FloatToStr(res.Iterations);

**end**;

**end**.

**Код лабораторної:** <https://github.com/ptrstasiv/Lab5.git>

**Висновок:** при виконанні даної лабораторної роботи ми ознайомилися з особливостями створення графічного інтерфейсу в середовищі Delphi.