Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

ЛАБОРАТОРНА РОБОТА № 4

з дисципліни «Алгоритми та методи обчислення»

на тему «Розв’язання нелінійних рівнянь на комп’ютері»

ВИКОНАВ:

cтудент ІІ курсу ФІОТ

групи ІО-91

Діденко Владислав Віталійович

Варіант – 9

ПЕРЕВІРИВ:

ст.вик. Порєв В. М.

Київ – 2021

**Мета:** Ознайомлення з методиками та вивчення різних алгоритмів розв’язання нелінійних рівнянь на комп’ютері.

**Завдання:** Закріплення знань студентів при вирішенні практичних завдань з розв’язування нелінійних рівнянь. Оволодіння методами і практичними навичками розв’язування нелінійних рівнянь на комп’ютері. Набуття умінь і навичок при програмуванні та налагодженні програм для розв’язування нелінійних рівнянь на комп'ютері. 

**Текст програми:**

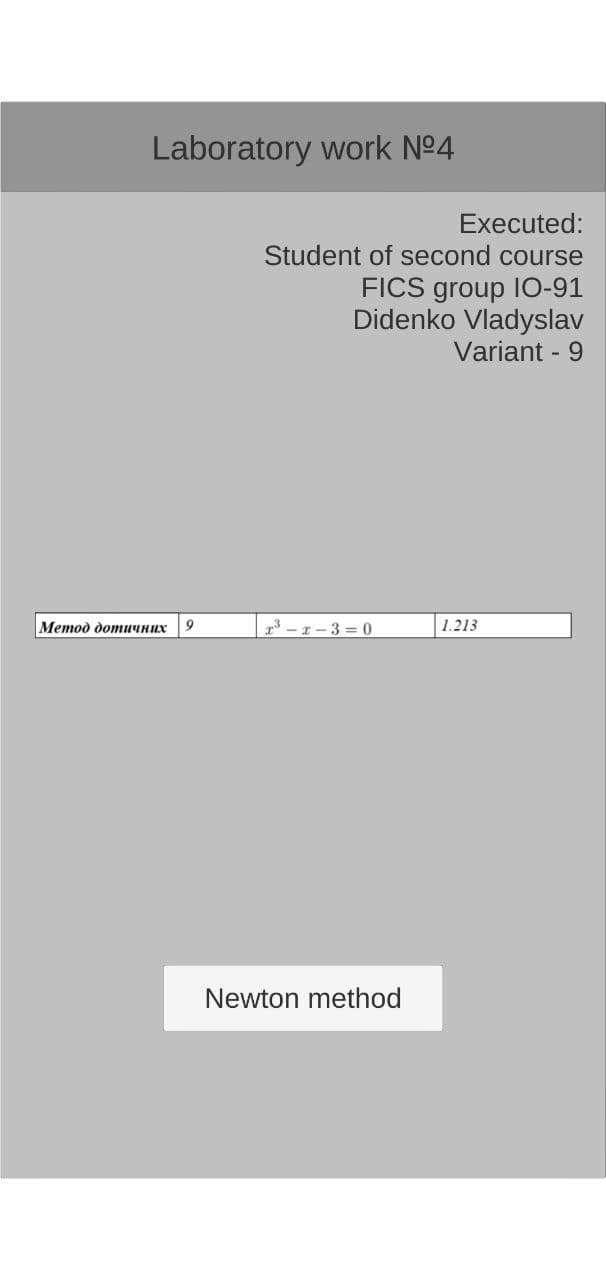
CanvasController.cs

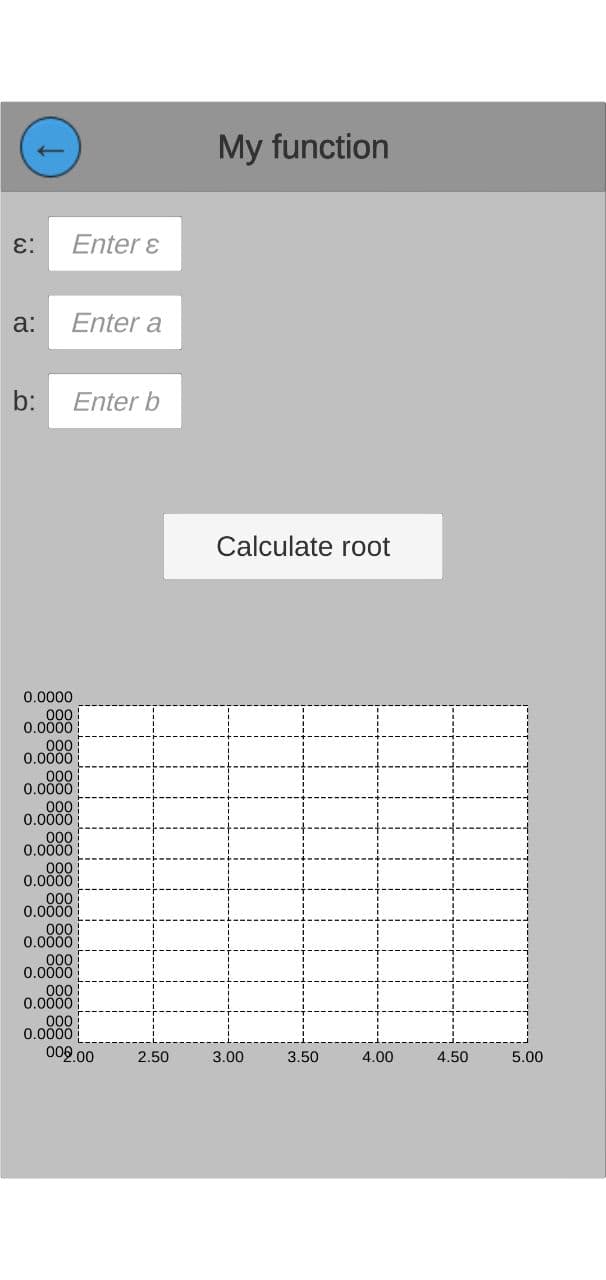
using System;  
using System.Collections;  
using System.Globalization;  
using AwesomeCharts;  
using UnityEngine;  
using UnityEngine.UI;  
  
public class **CanvasController** : MonoBehaviour {  
 public static CanvasController instance { get; private set; }  
 private const float ERROR\_MESSAGE\_TIME = 2f;  
 private const string INPUT\_ERROR\_MESSAGE = "Wrong input";  
  
 public GameObject **info**;  
 public GameObject **newtonMethodGO**;  
  
 public GameObject **error**;  
 public Image **errorImage**;  
 public Text **errorText**;  
 private Coroutine errorCoroutine;  
  
 public LineChart **plot**;  
  
 public InputField **inputLeftBorder**;  
 public InputField **inputRightBorder**;  
 public InputField **inputEpsilon**;  
  
 private float MyFunction(float x) => x \* x \* x - x - 3;  
 private float MyFirstDerivativeFunction(float x) => 3 \* x \* x - 1;  
 private float MySecondDerivativeFunction(float x) => 6 \* x;  
  
  
 private void ShowErrorMessage(string message) {  
 if (errorCoroutine != null) {  
 StopCoroutine(errorCoroutine);  
 }  
  
 errorCoroutine = StartCoroutine(\_ShowErrorMessage(message));  
 }  
  
 private IEnumerator \_ShowErrorMessage(string message) {  
 error.SetActive(true);  
 errorText.text = message;  
 var time = 0f;  
 while (time < ERROR\_MESSAGE\_TIME) {  
 errorImage.color = Color.Lerp(Color.red, Color.clear, time / ERROR\_MESSAGE\_TIME);  
 time += Time.deltaTime;  
 yield return null;  
 }  
  
 error.SetActive(false);  
 errorCoroutine = null;  
 }  
  
 public void RaiseAndShowError(string message) {  
 ShowErrorMessage(message);  
 }  
  
 private void **Awake**() {  
 instance = this;  
 CultureInfo.CurrentCulture = CultureInfo.GetCultureInfo("en-US");  
 }  
  
 private void **Start**() {  
 SetDefault();  
 }  
  
 public void **SetDefault**() {  
 info.SetActive(true);  
 newtonMethodGO.SetActive(false);  
 }  
  
 private void OnClick(GameObject other) {  
 info.SetActive(false);  
 other.SetActive(true);  
 }  
  
 public void **OnNewtonMethod**() {  
 OnClick(newtonMethodGO);  
 }  
  
 private bool CheckInputs(out float leftBorder, out float rightBorder, out float epsilon) {  
 if (!float.TryParse(inputLeftBorder.text, NumberStyles.Float, CultureInfo.InvariantCulture, out leftBorder) ||  
 !float.TryParse(inputRightBorder.text, NumberStyles.Float, CultureInfo.InvariantCulture, out rightBorder) ||  
 !float.TryParse(inputEpsilon.text, NumberStyles.Float, CultureInfo.InvariantCulture, out epsilon)) {  
 (leftBorder, rightBorder, epsilon) = (0, 0, 0);  
 leftBorder = rightBorder = epsilon = 0;  
 return false;  
 }  
  
 return true;  
 }  
  
  
 public void **OnFindRoot**() {  
 ClearPlotChart(plot);  
 if (!CheckInputs(out var leftBorder, out var rightBorder, out var epsilon)) {  
 RaiseAndShowError(INPUT\_ERROR\_MESSAGE);  
 return;  
 }  
  
 DrawFunction(leftBorder, rightBorder);  
 float root;  
 try {  
 root = FindRoot(leftBorder, rightBorder, epsilon);  
 }  
 catch (Exception e) {  
 RaiseAndShowError(e.Message);  
 return;  
 }  
  
 AddEntryToPlot(1, root - root / 1000, MyFunction(root + root / 1000), plot);  
 AddEntryToPlot(1, root + root / 1000, MyFunction(root - root / 1000), plot);  
 RefreshPlotChart();  
 }  
  
 private float FindRoot(float leftBorder, float rightBorder, float epsilon) {  
 NewtonMethod.instance.SetData(MyFunction, MyFirstDerivativeFunction, MySecondDerivativeFunction, leftBorder,  
 rightBorder, epsilon);  
 return NewtonMethod.instance.FindRoot();  
 }  
  
 private void DrawFunction(float leftBorder, float rightBorder, int numOfPoints = 250) {  
 ConfigurePlotChartAxisBorders(plot.XAxis, new Vector2(leftBorder, rightBorder));  
 for (var x = leftBorder; x < rightBorder; x += (rightBorder - leftBorder) / numOfPoints) {  
 AddEntryToPlot(0, x, MyFunction(x), plot);  
 }  
  
 RefreshPlotChart();  
 }  
  
 private void AddEntryToPlot(int dataSet, float x, float y, LineChart plot) {  
 plot.GetChartData().DataSets[dataSet].AddEntry(new LineEntry(x, y));  
 }  
  
 private void AddEntryToPlot(int dataSet, Vector2 entry, LineChart plot) {  
 AddEntryToPlot(dataSet, entry.x, entry.y, plot);  
 }  
  
  
 private void ConfigurePlotChartAxisBorders(AxisBase axisObj, Vector2 axisLimits) {  
 axisObj.MinAxisValue = axisLimits.x;  
 axisObj.MaxAxisValue = axisLimits.y;  
 }  
  
 private void ClearPlotChart(LineChart lineChart) {  
 foreach (var dataSet in lineChart.GetChartData().DataSets) {  
 dataSet.Clear();  
 }  
 }  
  
 private void RefreshPlotChart() {  
 plot.SetDirty();  
 }  
}

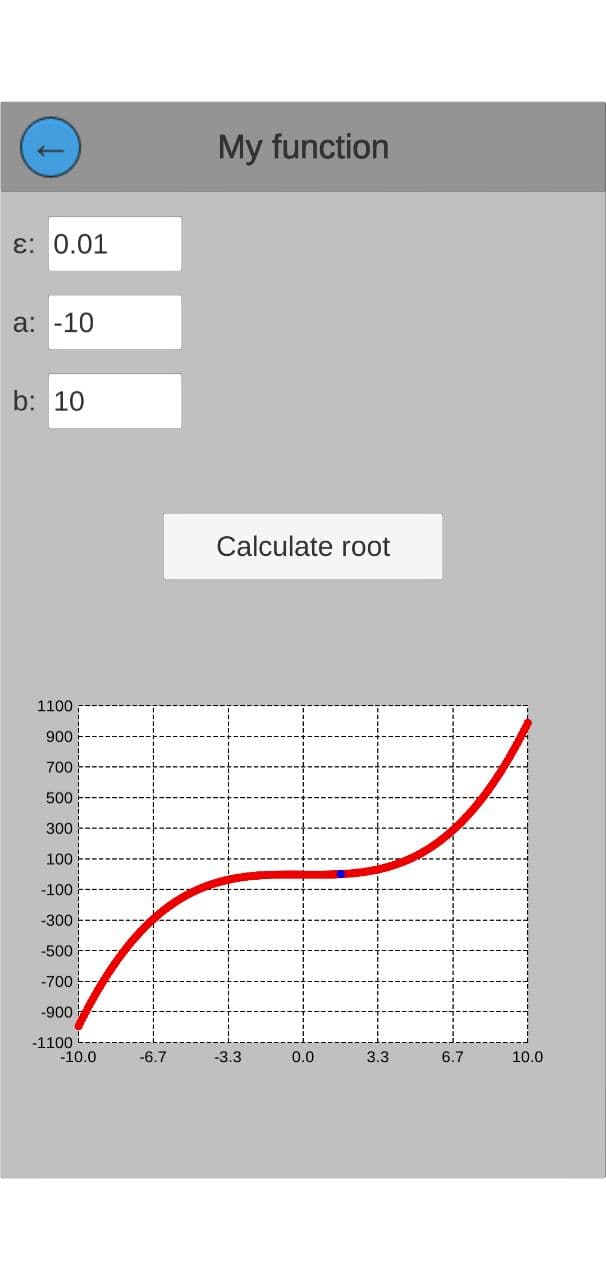
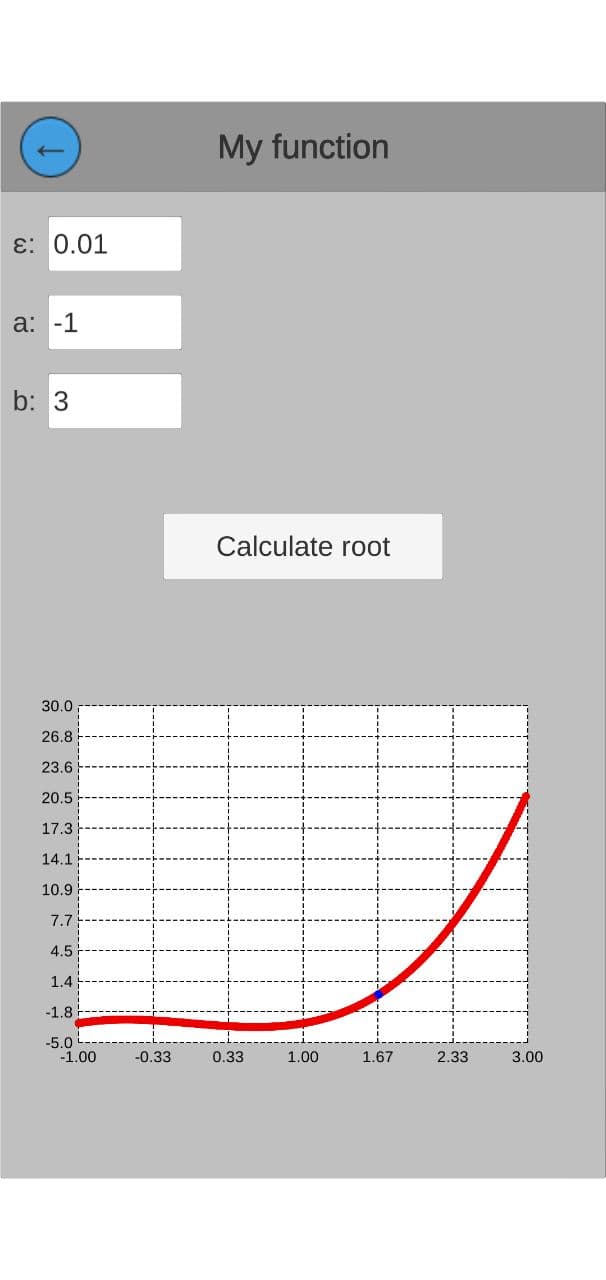
NewtonMethod.cs

using System;  
  
public class NewtonMethod {  
 public static NewtonMethod instance { get; } = new NewtonMethod();  
  
 private const string NO\_ROOT = "There is no root on this interval";  
 private const string WRONG\_BORDERS = "Left border can't be bigger than right border";  
  
 private float leftBorder;  
 private float rightBorder;  
 private float epsilon;  
  
 public delegate float Function(float x);  
  
 public delegate float FirstDerivative(float x);  
  
 public delegate float SecondDerivative(float x);  
  
 private Function fx;  
 private FirstDerivative dfx;  
 private SecondDerivative d2fx;  
  
 public void SetData(Function fx, FirstDerivative dfx, SecondDerivative d2fx, float leftBorder, float rightBorder,  
 float epsilon) {  
 if (leftBorder > rightBorder) {  
 throw new Exception(WRONG\_BORDERS);  
 }  
  
 this.fx = fx;  
 this.dfx = dfx;  
 this.d2fx = d2fx;  
 this.leftBorder = leftBorder;  
 this.rightBorder = rightBorder;  
 this.epsilon = epsilon;  
 }  
  
 public float FindRoot() {  
 return FindRoot(leftBorder, rightBorder);  
 }  
  
 private float FindRoot(float a, float b) {  
 if (fx(a) \* fx(b) > 0) {  
 throw new Exception(NO\_ROOT);  
 }  
  
 if (b - a < epsilon) {  
 return (b + a) / 2f;  
 }  
  
 if (fx(b) \* d2fx(b) < 0) {  
 // (a, b) = Utils.Swap(a, b);  
 b = a;  
 }  
  
 var root = b;  
 do {  
 b = root;   
 root = b - fx(b) / dfx(b);  
 } while (Math.Abs(root - b) > epsilon);  
   
 return root;  
 }  
}  
  
public static class Utils {  
 public static (T b, T a) Swap<T>(T a, T b) {  
 return (b, a);  
 }  
}

**Скріншоти виконання:**

****

****

****

**Висновок:** У ході виконання лабораторної роботи я закріпив знання з базових понять алгоритмів інтерполяції, вивчив основні алгоритми інтерполяції, покращив навички будування графіків за допомогою модуля Awesome Charts. Отримані результати виконання програми є вірними.