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 № 3

з дисципліни «Алгоритми та методи обчислення»

на тему «Інтерполяція функцій»

ВИКОНАВ:

Студент ІІ курсу ФІОТ

групи ІО-91

Діденко Владислав Віталійович

Варіант – 9

ПЕРЕВІРИВ:

ст.вик. Порєв В. М.

Київ – 2021

**Мета:** Ознайомлення з інтерполяційними формулами Лагранжа, Ньютона, рекурентним співвідношенням Ейткена, методами оцінки похибки інтерполяції.

**Завдання:** Закріплення, поглиблення і розширення знань студентів при вирішенні практичних обчислювальних завдань. Оволодіння обчислювальними методами і практичними методами оцінки похибки обчислень. Придбання умінь і навичок при програмуванні та налагодженні обчислювальних завдань на комп'ютері.



**Текст програми:**

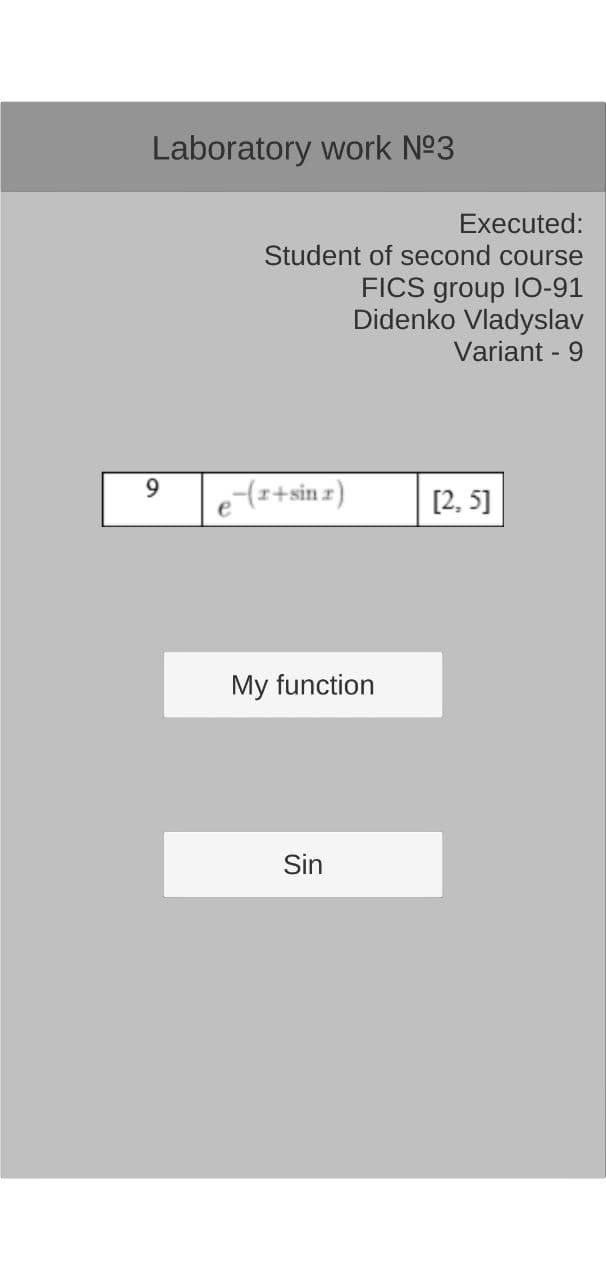
CanvasController.cs

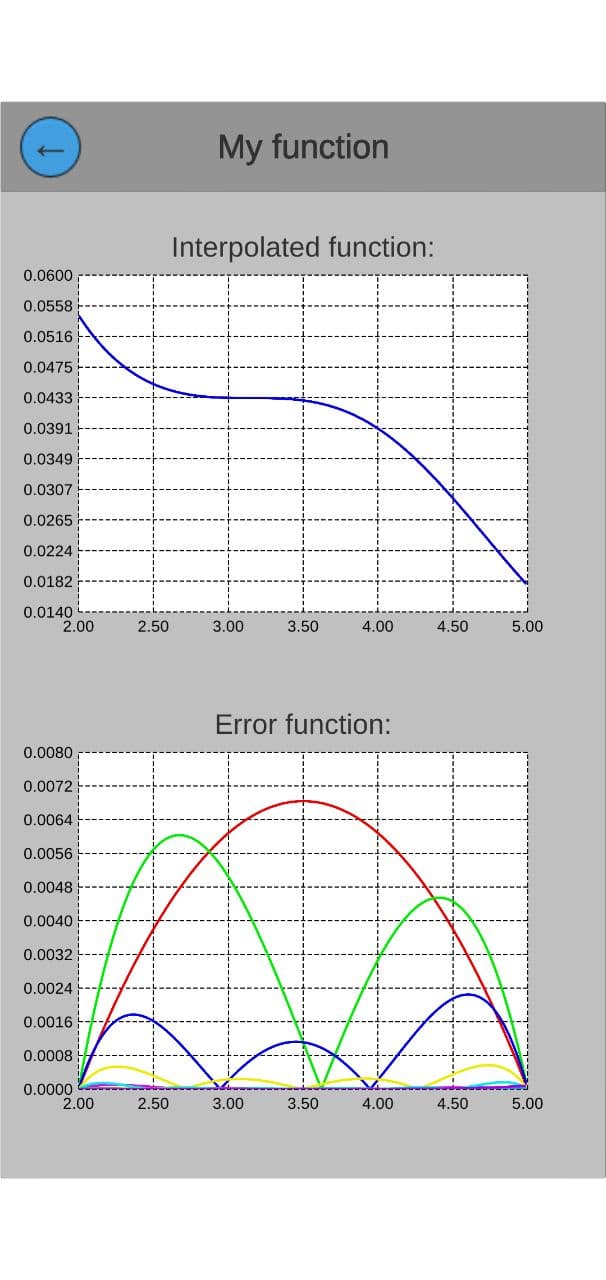
using System.Collections;  
using AwesomeCharts;  
using UnityEngine;  
using UnityEngine.UI;  
  
public class CanvasController : MonoBehaviour {  
 public static CanvasController instance;  
 public static Interpolation interpolation;  
 private const float ERROR\_MESSAGE\_TIME = 2f;  
 private const string INPUT\_ERROR\_MESSAGE = "Wrong input";  
 public GameObject info;  
 public GameObject myFunc;  
 public GameObject sin;  
 public GameObject error;  
 public Image errorImage;  
 public Text errorText;  
 private Coroutine errorCoroutine;  
  
 public LineChart myFuncPlot;  
 public LineChart myFuncErrorPlot;  
 public LineChart sinPlot;  
 public LineChart sinErrorPlot;  
 public float leftBorder;  
 public float rightBorder;  
  
  
 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;  
 interpolation = new Interpolation(this);  
 }  
  
 private void Start() {  
 SetDefault();  
 ConfigurePlotChartAxisBorders(myFuncPlot.XAxis, new Vector2(leftBorder, rightBorder));  
 ConfigurePlotChartAxisBorders(myFuncErrorPlot.XAxis, new Vector2(leftBorder, rightBorder));   
 ConfigurePlotChartAxisBorders(sinPlot.XAxis, new Vector2(leftBorder, rightBorder));  
 ConfigurePlotChartAxisBorders(sinErrorPlot.XAxis, new Vector2(leftBorder, rightBorder));   
 }  
  
 public void SetDefault() {  
 info.SetActive(true);  
 myFunc.SetActive(false);  
 sin.SetActive(false);  
 }  
  
 private void OnClick(GameObject other) {  
 info.SetActive(false);  
 other.SetActive(true);  
 }  
  
 public void OnMyFunc() {  
 OnClick(myFunc);  
 MyFunction();  
 }  
  
 public void OnSin() {  
 OnClick(sin);  
 SinFunction();  
 }  
  
 private void SinFunction() {  
 ClearPlotChart(sinPlot);  
 ClearPlotChart(sinErrorPlot);  
   
 interpolation.SinFunctionInterpolation(11, leftBorder, rightBorder);  
 interpolation.SinFunctionError(11, leftBorder, rightBorder);  
  
 RefreshAllPlotCharts();  
 }  
  
 private void MyFunction() {  
 ClearPlotChart(myFuncPlot);  
 ClearPlotChart(myFuncErrorPlot);  
   
 interpolation.MyFunctionInterpolation(11, leftBorder, rightBorder);  
 interpolation.MyFunctionError(11, leftBorder, rightBorder);  
  
 RefreshAllPlotCharts();  
 }  
  
 public void AddEntryToMainPlot(int dataSet, Vector2 entry, LineChart plot) {  
 plot.GetChartData().DataSets[dataSet].AddEntry(new LineEntry(entry.x, entry.y));   
 }  
  
 public void AddEntryToErrorPlot(int dataSet, Vector2 entry, LineChart plot) {  
 plot.GetChartData().DataSets[dataSet].AddEntry(new LineEntry(entry.x, entry.y));  
 }  
  
 private void ConfigurePlotChartAxisBorders(AxisBase axisObj, Vector2 axis) {  
 axisObj.MinAxisValue = axis.x;  
 axisObj.MaxAxisValue = axis.y;  
 }  
  
 private void ClearPlotChart(LineChart lineChart) {  
 foreach (var dataSet in lineChart.GetChartData().DataSets) {  
 dataSet.Clear();  
 }  
 }  
  
 private void RefreshAllPlotCharts() {  
 myFuncPlot.SetDirty();  
 myFuncErrorPlot.SetDirty();  
 sinPlot.SetDirty();  
 sinErrorPlot.SetDirty();  
 }  
}

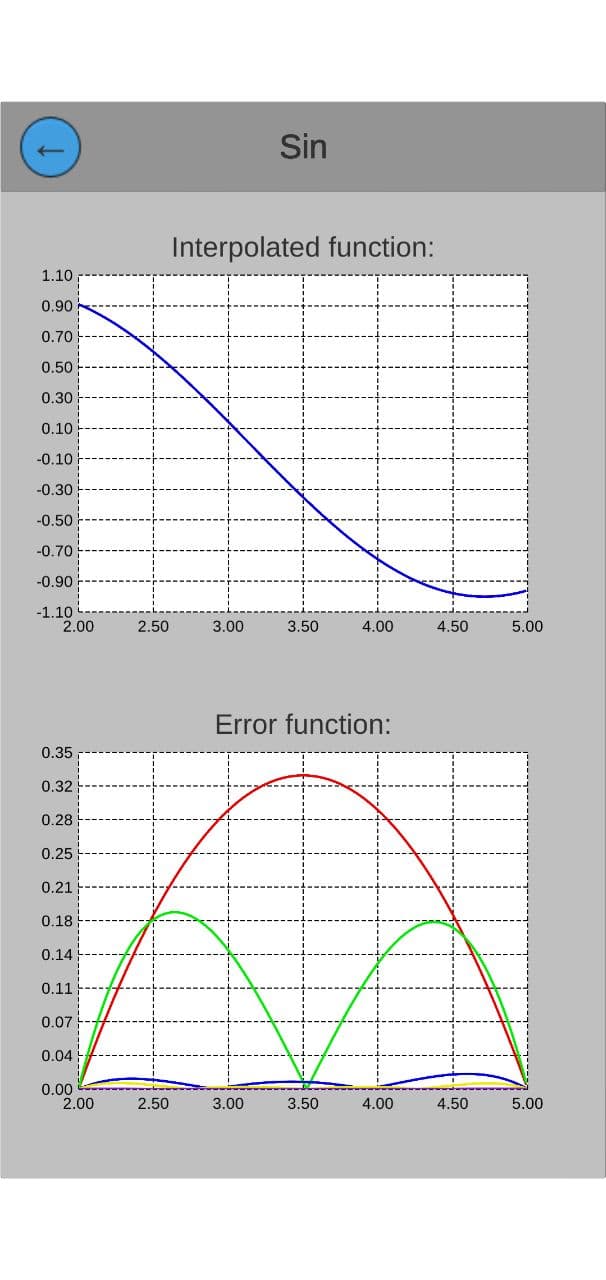
Interpolation.cs

using UnityEngine;  
  
public class Interpolation {  
 private CanvasController ui;  
  
 public Interpolation(CanvasController ui) {  
 this.ui = ui;  
 }  
  
 public void SinFunctionInterpolation(int degree, float leftBorder, float rightBorder) {  
 float[] xArray = ValuesInInterval(leftBorder, rightBorder, degree);  
 float[] yArray = new float[xArray.Length];  
 for (int i = 0; i < xArray.Length; i++) {  
 yArray[i] = Mathf.Sin(xArray[i]);  
 }  
  
 float xValue = leftBorder;  
 float step = 0.01f;  
 while (xValue < rightBorder) {  
 float interpolatedValue = Interpolate(xArray, yArray, xValue);  
 float analyticValue = Mathf.Sin(xValue);  
 ui.AddEntryToMainPlot(0, new Vector2(xValue, interpolatedValue), CanvasController.instance.sinPlot);  
 ui.AddEntryToMainPlot(1, new Vector2(xValue, analyticValue), CanvasController.instance.sinPlot);  
 xValue += step;  
 }  
 }  
  
 public void MyFunctionInterpolation(int degree, float leftBorder, float rightBorder) {  
 float[] xArray = ValuesInInterval(leftBorder, rightBorder, degree);  
 float[] yArray = new float[xArray.Length];  
 for (int i = 0; i < xArray.Length; i++) {  
 yArray[i] = 3 \* Mathf.Pow(Mathf.Cos(xArray[i]), 2) - Mathf.Sqrt(xArray[i]);  
 }  
  
 float xValue = leftBorder;  
 float step = 0.01f;  
 while (xValue < rightBorder) {  
 float interpolatedValue = Interpolate(xArray, yArray, xValue);  
 float analyticValue = 3 \* Mathf.Pow(Mathf.Cos(xValue), 2) - Mathf.Sqrt(xValue);  
 ui.AddEntryToMainPlot(0, new Vector2(xValue, interpolatedValue), CanvasController.instance.myFuncPlot);  
 ui.AddEntryToMainPlot(1, new Vector2(xValue, analyticValue), CanvasController.instance.myFuncPlot);  
 xValue += step;  
 }  
 }  
  
 public void SinFunctionError(int maxDegree, float leftBorder, float rightBorder) {  
 float[][] xArrays = new float[maxDegree + 1][];  
 float[][] yArrays = new float[maxDegree + 1][];  
  
 for (int i = 0; i < maxDegree + 1; i++) {  
 xArrays[i] = ValuesInInterval(leftBorder, rightBorder, i + 2);  
 yArrays[i] = new float[xArrays[i].Length];  
 for (int j = 0; j < xArrays[i].Length; j++) {  
 yArrays[i][j] = Mathf.Sin(xArrays[i][j]);  
 }  
 }  
  
 for (int i = 0; i < maxDegree - 1; i++) {  
 float xValue = leftBorder;  
 float step = 0.01f;  
 while (xValue < rightBorder) {  
 float yValue = Mathf.Abs(Interpolate(xArrays[i], yArrays[i], xValue) -  
 Interpolate(xArrays[i + 1], yArrays[i + 1], xValue));  
 ui.AddEntryToErrorPlot(i, new Vector2(xValue, yValue), CanvasController.instance.sinErrorPlot);  
 xValue += step;  
 }  
 }  
 }  
  
 public void MyFunctionError(int maxDegree, float leftBorder, float rightBorder) {  
 float[][] xArrays = new float[maxDegree + 1][];  
 float[][] yArrays = new float[maxDegree + 1][];  
  
 for (int i = 0; i < maxDegree + 1; i++) {  
 xArrays[i] = ValuesInInterval(leftBorder, rightBorder, i + 2);  
 yArrays[i] = new float[xArrays[i].Length];  
 for (int j = 0; j < xArrays[i].Length; j++) {  
 yArrays[i][j] = Mathf.Exp(-(xArrays[i][j] + Mathf.Sin(xArrays[i][j])));  
 }  
 }  
  
 for (int i = 0; i < maxDegree - 1; i++) {  
 float xValue = leftBorder;  
 float step = 0.01f;  
 while (xValue < rightBorder) {  
 float yValue = Mathf.Abs(Interpolate(xArrays[i], yArrays[i], xValue) -  
 Interpolate(xArrays[i + 1], yArrays[i + 1], xValue));  
 ui.AddEntryToErrorPlot(i, new Vector2(xValue, yValue), CanvasController.instance.myFuncErrorPlot);  
 xValue += step;  
 }  
 }  
 }  
  
 public float Interpolate(float[] xArray, float[] yArray, float xValue) {  
 float functionValue = 0;  
  
 for (int i = 0; i < xArray.Length; i++) {  
 functionValue += DifferenceOfX(i, xValue, xArray) \* DividedDifference(0, i, xArray, yArray);  
 }  
  
 return functionValue;  
 }  
  
 private float DividedDifference(int leftBorderIndex, int rightBorderIndex, float[] xArray, float[] yArray) {  
 float dividedDifference = 0;  
  
 for (int j = leftBorderIndex; j <= rightBorderIndex; j++) {  
 float denominator = 1;  
 for (int i = leftBorderIndex; i <= rightBorderIndex; i++) {  
 if (i == j) {  
 continue;  
 }  
  
 denominator \*= (xArray[j] - xArray[i]);  
 }  
  
 dividedDifference += yArray[j] / denominator;  
 }  
  
 return dividedDifference;  
 }  
  
 private float DifferenceOfX(int amountOfX, float xValue, float[] xArray) {  
 float differenceOfX = 1;  
  
 for (int i = 0; i < amountOfX; i++) {  
 differenceOfX \*= xValue - xArray[i];  
 }  
  
 return differenceOfX;  
 }  
  
 private float[] ValuesInInterval(float leftBorder, float rightBorder, int amountOfValues) {  
 float step = Mathf.Abs(rightBorder - leftBorder) / (amountOfValues - 1);  
 float[] values = new float[amountOfValues];  
  
 for (int i = 0; i < amountOfValues; i++) {  
 values[i] = leftBorder + step \* i;  
 }  
  
 return values;  
 }  
}

**Скріншоти виконання:**

****

****

****

**Розмитість оцінки похибки в точці x = 3 (функція за варіантом):**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 2 | -0.0061 | -0.00115 | 0.81157 |
| 3 | 0.00499 | 0.004958 | 0.006495 |
| 4 | -0.000239 | -3.214 \* 10^(-5) | 0.86446 |
| 5 | 0.000235 | 0.0002067 | 0.12 |
| 6 | -2.9635 \* 10^(-5) | -2.82116 \* 10^(-5) | 0.048 |
| 7 | 4.2692 \* 10^(-5) | 1.4231 \* 10^(-6) | 0.6667 |
| 8 | -2.41 \* 10^(-6) | -2.846 \* 10^(-6) | -0.1808 |
| 9 | -4.0978 \* 10^(-7) | -4.3586 \* 10^(-7) | -0.06364 |
| 10 | -3.35276 \* 10^(-8) | -2.6077 \* 10^(-8) | 0.22222 |
| 11 | 1.11758 \* 10^(-8) | 7.45058\* 10^(-9) | 0.33333 |

**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.