МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (ФГБОУ ВО «ВГУ»)

Факультет прикладной математики, информатики и механики

Кафедра прикладной механики и информатики

Отчет по лабораторной работе №1

на тему:

**«Численные исследования метода интерполяции многочленом Лагранжа»**

Выполнил: студент 3 к. 1 гр. ПМИ в.о.

Бедарев Анатолий Анддреевич

Проверил: к.ф-м.н, доц.

Гудович Николай Николаевич

Воронеж – 2017

**Содержание**

|  |  |
| --- | --- |
| Постановка задачи ....................................................................................................... | 3 |
| Указания к выполнению лабораторной работы ....................................................... | 4 |
| Ход выполнения .......................................................................................................... | 4 |
| Выводы по работе ........................................................................................................ | 15 |
| Список литературы ...................................................................................................... | 16 |
| Приложение (листинг) ................................................................................................ | 17 |
|  |  |
|  |  |
|  |  |

**Постановка задачи**

**1**. Составить и отладить программу приближенного нахождения значения функции в заданной точке x\* отрезка [a, b].

*Входные данные*:

- отрезок [a, b];

- функция F(x), по которой производится расчет значений в узлах интерполяции (значения которой приближаются интерполяционным многочленом)

, ; (1)

- степень интерполяционного многочлена N, на первом этапе принять N = 2;

- произвольная точка x\* отрезка [a, b], для которой считается значение интерполяционного многочлена.

В программе предусмотреть вычисление набора узлов интерполяции *x0, x1, …, xn* (считать равноотстоящими друг от друга на отрезке [a, b]; исходя из степени многочлена N = 2, их количество равно 3).

*Выходные данные*:

- значение многочлена в точке x\* (приближенное значение функции, Ln(x\*));

- точное значение функции в точке точке x\*, F(x\*);

- погрешность полученного приближения, *ε*.

**2**. Составить и отладить программу построения графиков исходной функции F(x) и ее интерполяционного многочлена Ln(x), построенного по равноотстоящим узлам интерполяции на отрезке [a, b].

**3**. Провести численный эксперимент для выяснению вопроса о сходимости графика интерполяционного многочлена к графику исходной функции вляинии степени интерполяционного полинома на точность интерполяции.

**Теоретический материал**

Интерполяцией называется такой вид точечной аппроксимации, когда аппроксимирующая функция представляет собой алгебраический многочлен (полином) *ϕ(x)* степени *n*, который в *n+1* точке (узле) *xi (i=0,1,...,n)*, заданных на отрезке [*a,b*], совпадает со значением аппроксимируемой функции *f(x)* в этих узлах.

Для приближенного нахождения значения функции в заданной точке *x\** используется метод Лагранжа: подставив *x\**, узлы интерполяции *x0, x1, …, xn* и значения функции в этих узлах *f(x0), f(x1), …, f(xn)* в формулу:

, (2)

Погрешность интерполяции *rn(x)* зависит от числа узлов n и равна:

, (3)

**Ход выполнения работы**

Для выполнения поставленных задач было разработано несколько программных модулей (см. Приложение) с использованием средств языка технического моделирования MatLab 2017. В качестве дополнительного срадства разработки использовался язык C#, поддержка которого реализована в IDE Misrosoft Visual Studio 2017.

В частности, MatLab использовался для быстрого прототипирования и отладки, вывода графиков и поведения исследований полиномов различной степени.

Средства C# использовались для сравнения точности приближения в зависимости от способов вычисления коэффициентов многочлена Лагранжа. В отличие от MatLab, который предназначен для научных вычислений, C# ближе к традиционным средствам программирования и отражает все достоинства и недостатки современных языков программирования.

**На первом этапе** разработаны функции MatLab (рис.1), находящиеся в следующих файлах:

* *f.m* – модуль, реализующий расчет значения функции F(x) для узла x;
* *interpolate1.m* – основной модуль, реализующий интерполяцию полиномом Лагранжа; на вход получает значения концов отрезка, степень полинома и значение точки x\*, на выходе – значения L(x\*), F(x\*) и погрешности *ε*:

[Lx, Fx, d] = interpolate1(-1, 3, 2, 2.5)

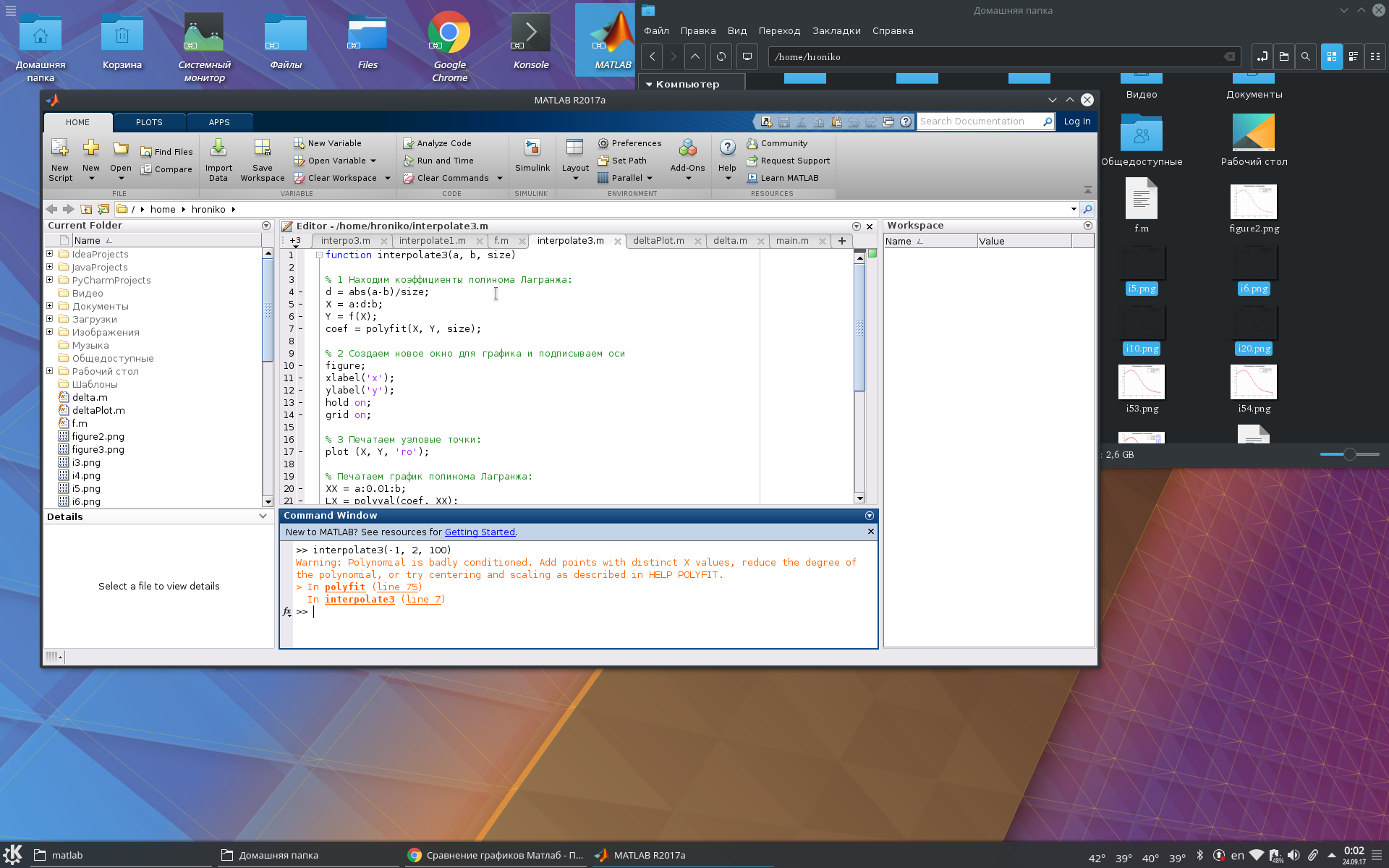


Рисунок 1 – Окно MatLab с открытым в редакторе модулем и консолью

* *interpolate2.m* – вспомогательный модуль, реализующий интерполяцию полиномом Лагранжа по известным узлам; на вход получает массивы узлов X и Y, а также значение точки x\*, на выходе – значения L(x\*), F(x\*) и погрешности *ε*:

[Lx, Fx, d] = interpolate2(X, Y, 2.5)

Вспомогательный модуль вызывается основным модулем после того, как на основе данных об отрезке [a, b] формируются массивы узлов, например:

X = [1; 2; 3];

Y = [-2; 1; 6];

Результатом выполнения интерполяционных модулей выступает вывод в консоль MatLab значений полинома и функции в точке x\* с погрешностью *ε*, а также новое окно с графиками интерполяционного многочлена, нанесенными узловыми точками и точками значений L(x\*), F(x\*) вида (рис. 2):

**На втором этапе** разработан модуль *interpolate3.m*, реализующий построение графиков исходной функции F(x) и ее интерполяционного многочлена Ln(x).

Вызов модуля осуществляется командой следующего вида (пример):

interpolate3(a, b, n),

где a и b – начало и конец отрезка [a, b], n – степень интерполяционного полинома Лагранжа.

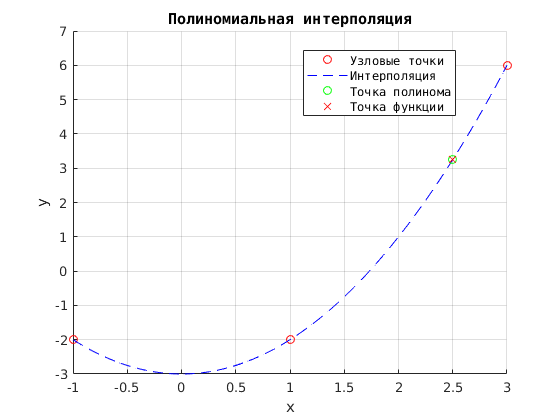


Рисунок 2 – (пример) Результат работы модуля *interpolate1.m*   
График интерполяционного многочлена Лагранжа с нанесенными узлами интерполяции и точками L(x\*), F(x\*)

В результате работы модуля формируется новое окно с нанесенными графиками функции и интерполяционного многочлена (рис. 3).

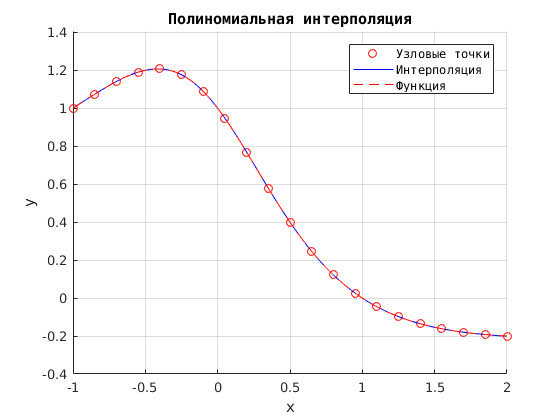


Рисунок 3 – (пример) Результат работы модуля *interpolate3.m*   
Сравнение графиков интерполяционного многочлена Лагранжа и функции F(x)

При построении графиков равноотстоящие узлы интерполяции дополняются промежуточными точками, поскольку узловых точек недостаточно для демонстрации расхождения графиков (в узловых точках они совпадают).

Кроме того, было реализовано решение на C# на основе WinForms и свободной библиотеки ZedGraph с аналогичным функционалом. Основное окно программы с отрисованным графиком приведено на рис. 4.

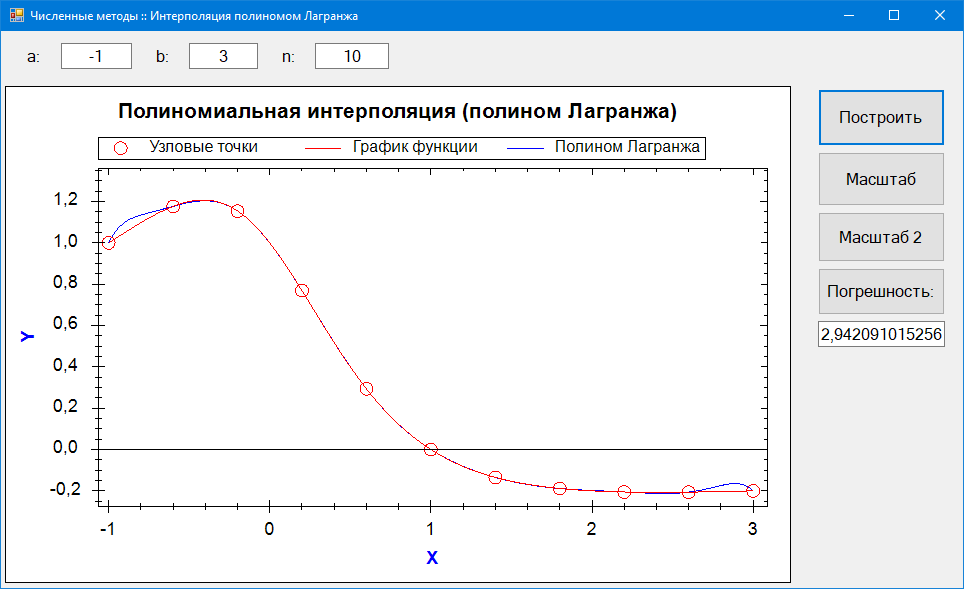


Рисунок 4 – Окно программы на C#: сравнение графиков интерполяционного многочлена Лагранжа и функции F(x)

**На третьем этапе** для выяснения вопроса о сходимости графика интерполяционного многочлена к графику исходной функции и вляинии степени интерполяционного полинома на точность интерполяции выполнен численный эксперимент. В качестве основного инструмента использовался разработанный ранее модуль MatLab *interpolate3.m*.

Изменяя степень полинома Лагранжа от 2 до 100, были получены сравнительные графики (рис. 5, 6). Их анализ показал, что интерполяционный многочлен Лагранжа для нашей функции не обладает сходимостью.

С увеличением степени полинома Лагранжа от 2 до N ≈ 50 (а значит и с увеличением количества узловых точек) график многочлена начинает приближаться к графику функции. Однако уже при N = 60 между графиками четко заметна расходимость, причем она тем выше, чем больше степень полинома. Расходимость полинома Лагранжа для заданной функции F(x) носит локальный характер и локализована на концах отрезка [a, b]. Это особенно отчетливо заметно при больших значениях N (более 70, рис. 6).

|  |  |
| --- | --- |
| а) | б) |
|  |  |
| при N = 2; | при N = 3; |
|  |  |
| в) | г) |
|  |  |
| при N = 5; | при N = 10; |
|  |  |
| д) | е) |
|  |  |
| при N = 20; | при N = 30; |

Рисунок 5 – Сравнение точности приближения

при различных значениях степени полинома Лагранжа

|  |  |
| --- | --- |
| а) | б) |
|  |  |
| при N = 40; | при N = 50; |
|  |  |
| в) | г) |
| C:\Users\KND\AppData\Local\Microsoft\Windows\INetCache\Content.Word\i60.png | C:\Users\KND\AppData\Local\Microsoft\Windows\INetCache\Content.Word\i70.png |
| при N = 60; | при N = 70; |
|  |  |
| д) | е) |
|  |  |
| при N = 80; | при N = 100; |

Рисунок 6 – (продолжение) Сравнение точности приближения

при различных значениях степени полиномом Лагранжа

По рисункам видно, что в диапазоне N между 50 и 60 приближение многочлена к графику останавливается и меняется на расходимость. Чтобы установить, при каком граничном значении N это начинает происходить, были поставлены дополнительные численные эксперименты в обозначенном диапазоне и установлено, что для заданной функции максимальное значение N без расходимости равно 53 (рис. 7).

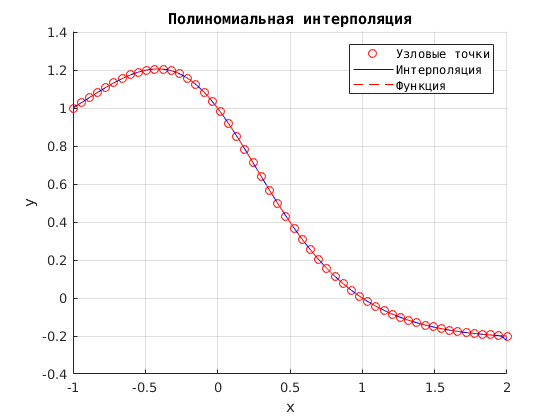


Рисунок 7 – Наилучшая точность приближения

при степени полиномом Лагранжа N = 53   
(максимальное значение N, при котором еще нет роста расхождения)

При N = 54 уже заметны возрастания колебаний полинома Лагранжа вблизи правой границы отрезка [a, b] и, как следствие, рост ошибки (рис. 8).

Данный факт подтверждается графиком зависимости абсолютной погрешности интерполяции от степени полинома Лагранжа (рис. 9), для построения которого использовались дополнительно разработанные модули *delta.m* и *deltaPlot.m*. (см. Приложение).

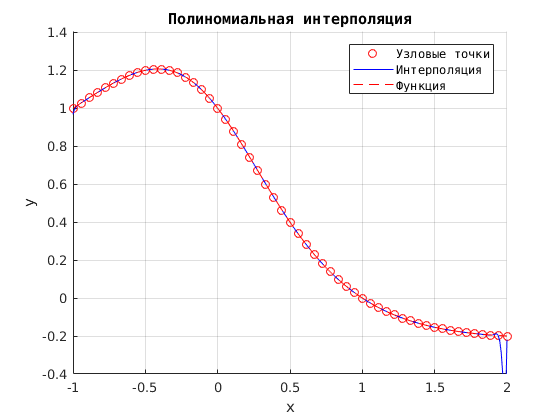


Рисунок 8 – Появление первых колебаний полинома Лагранжа вблизи правой границы диапазона (N = 54) и снижение точности приближения

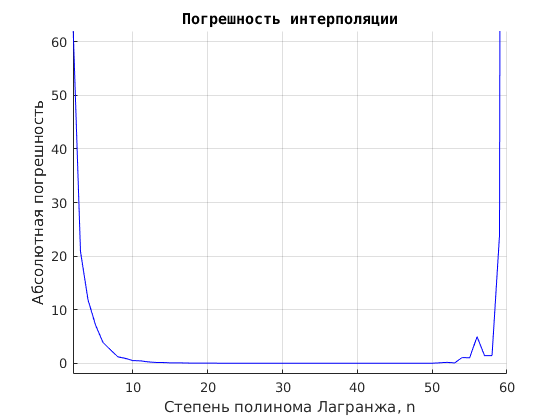


Рисунок 9 – Зависимость абсолютной погрешности интерполяции от степени полинома Лагранжа: заметен скачок погрешности при отметке N = 54

Стоит заметить, что наибольший прирост точности приближения интерполяционного многочлена наблюдается при степени N до значения 10, а затем прирост точности минимален (график погрешности превращается в горизонтальную линию). Это говорит о том, что для интерполяции достаточно использовать полиномы невысоких степеней, что в свою очередь позволит сохранить хороший уровень приближения и вместе с этим существенно снизить вычислительную нагрузку, которая неизбежно возникла бы при использовании интерполяции с более высокой степенью полинома.

**На следующем этапе** проведено сравнение различных способов вычисления значений интерполяционного многочлена в точке. Поскольку есть разные способы вычисления величин, то они могут быть по разному чувствительны к ошибкам округления. Чтобы ответить на вопрос, какой из способов точнее, были разработаны два дополнительных модуля MatLab *polyval1.m* и *polyval2.m*, а также вызывающая их вспомогательная функция *interpolate4diff.m*.

В модуле *polyval1.m* вычисление значения интерполяционного многочлена в точке реализовано таким образом, чтобы в одном цикле выполнялось и умножение, и деление на знаменатель, а в модуле *polyval2.m* эти вычисления разнесены на два цикла, по одному на вычисление числителя и на вычисление знаменателя.

На рис. 10 и 11 приведены результаты работы функции *interpolate4diff.m* для степени интерполяционного многчлена равной 100. Синим цветом выведен график полинома, значения точек которого посчитаны методом модуля *polyval1.m,* а зеленым – модуля *polyval2.m.*

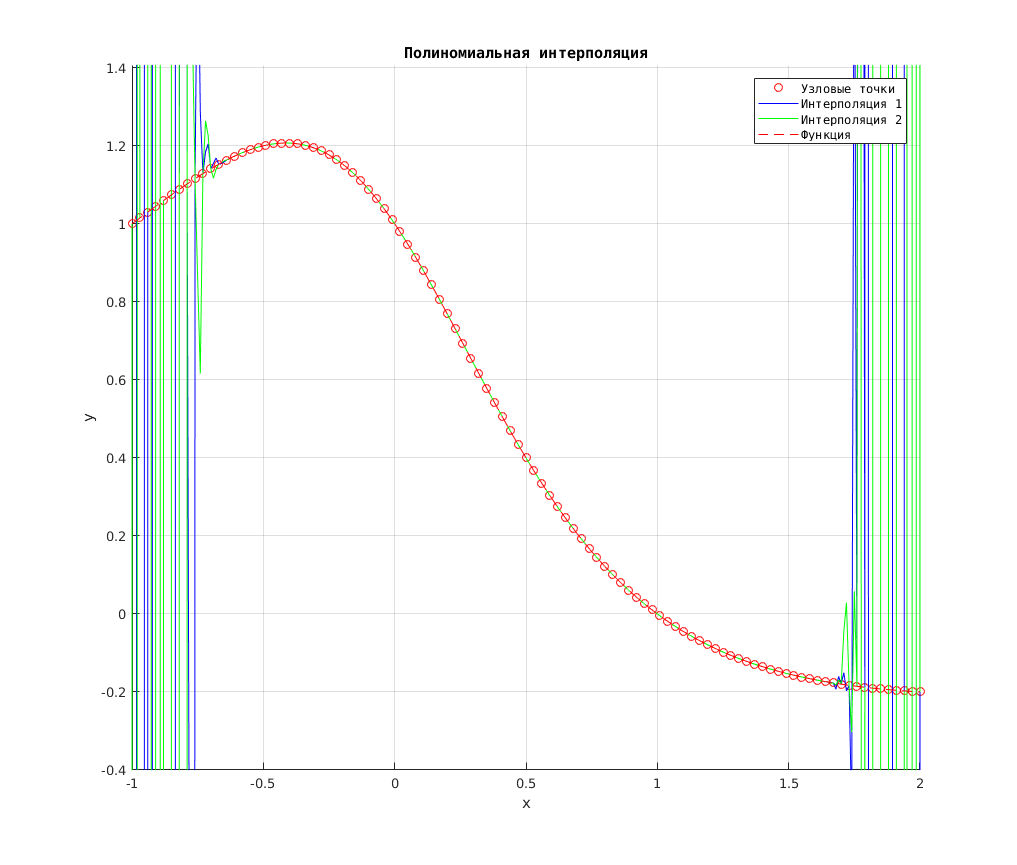


Рисунок 10 – Сравнение чувствительности к ошибкам округления различных способов вычисления значений многочлена Лагранжа (исходный масштаб)

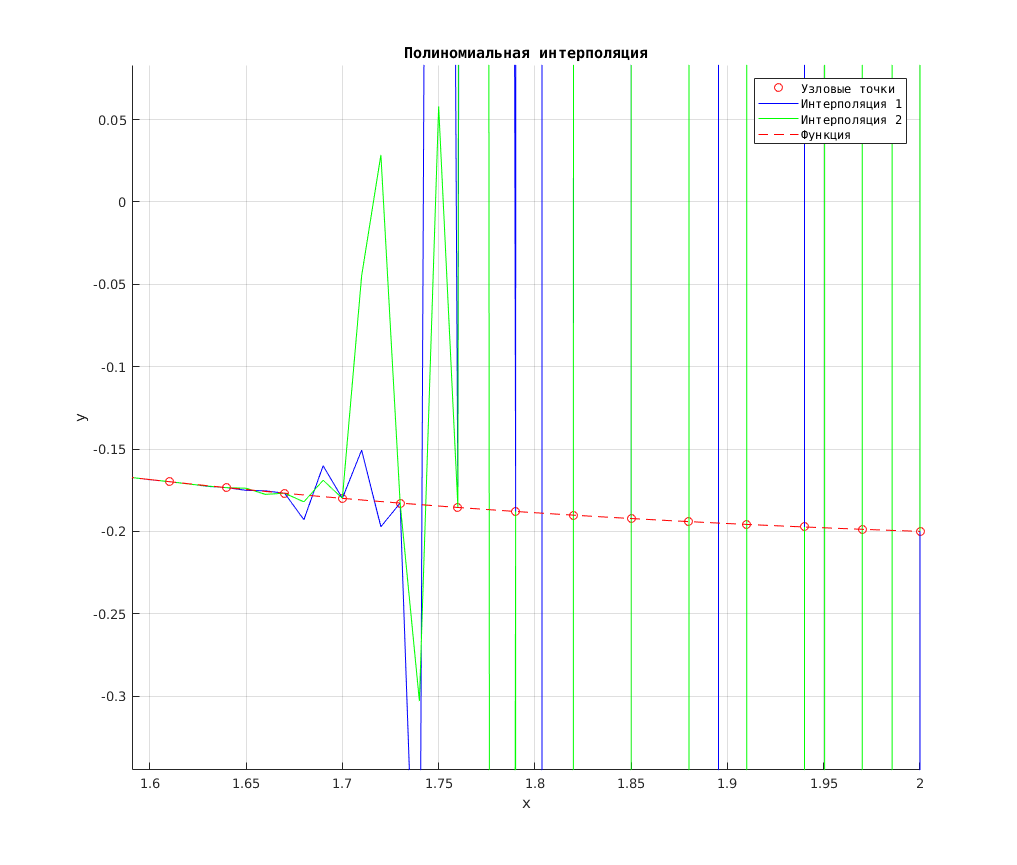
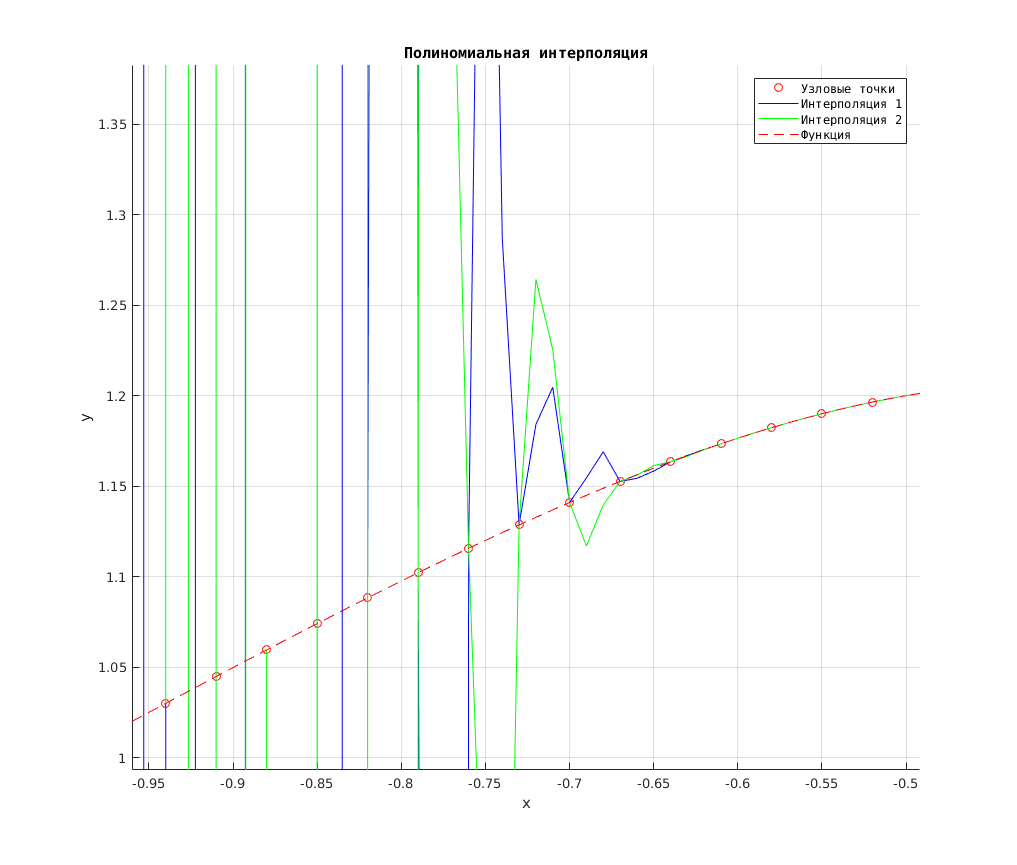


Рисунок 11 – Сравнение чувствительности к ошибкам округления различных способов вычисления значений многочлена Лагранжа (увеличение, показаны области вблизи левой и правой границы отрезка [a, b])

Как видно из графика, модуль *polyval1.m* дает лучшие результаты в плане сходимости (меньшую ошибку округления), чем модуль *polyval2.m.* Таким образом, способ, основанный на совместном выполнении в одном цикле и умножения, и деления на знаменатель дает лучший результат в плане сходимости.

Все алгоритмы, реализованные на MatLab, были перенесены на язык C# и проверены на работоспособность (рис. 12).

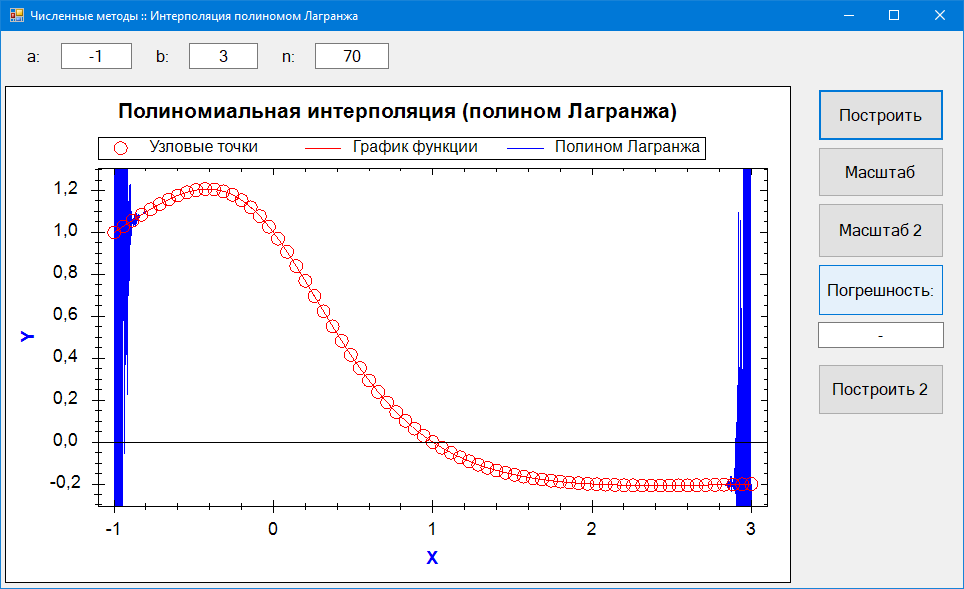


Рисунок 12 – Окно программы на C# с реализованными методами рассчета значений интерполяционного многочлена Лагранжа

В ходе проверки было замечено, что аналогичные алгоритмы на C# дают большую погрешность рассчетов, чем их полные аналоги на MatLab. Вероятно, это вызвано менее эффективной работой с числами двойной точности, реализованной в C#, и, как следствие, вызванной этим большей ошибкой округления. Кроме того при отдельных значениях степени N интерполяционного многочлена погрешность менялась скачкообразно, а не плавно, как это наблюдалось при использовании средств пакета MatLab. Затуднительно однозначно ответить, чем вызван такой результат. Однако как и в случае с реализацией на MatLab, реализация на C# подтвердила выводы о том, что лучшего приближения можно добиться при выполнении в одном цикле и умножения, и деления на знаменатель.

**Выводы по работе**

1. В ходе лабораторной работе был рассмотрен способ интерполяции функций алгебраическими многочленами методом Лагранжа, разработаны демонстрационные модули для MatLab и программа на C# с графическим интерфейсом, с помощью которых можно исследовать интерполяцию функций многочленами.

2. В результате численного эксперимента установлено, что интерполяционный многочлен Лагранжа для заданной функции не обладает сходимостью. По мере увеличения степени многочлена до N = 53 уровень приближения постепенно растет, а затем, с увеличением N, начинают проявляться регулярные осцилляции, которые при больших значениях N достаточно существенны и проявляются на границах интерполяционного отрезка.

3. Наибольшая скорость прироста точности приближения интерполяционного многочлена к функции наблюдается при небольших значениях N (до 10). Поэтому с целью снижения вычислительной нагрузки на ЭВМ целесообразно использовать полиномы невысоких степеней для интерполяции.

4. Помимо осцилляции, на сходимость полинома к интерполируемой функции существенное вляние оказывают ошибки округления. При высоких степениях полинома ошибки наиболее ощутимы и проявляются на графике в виде нерегулярных пилообразных выбросов.

5. Уровень ошибок округления зависит от выбранного способа рассчета значений интерполяционного многочлена. Способ, основанный на совместном вычислении в одном цикле числителя и деления на значенатель, дает меньшую ошибку округления, чем способ с раздельными циклами для числителя и знаменателя.

6. Установлено, что уровень ошибок округления завист также от реализации способа хранения и работы с числами двойной точности в программных средах разработки. Одни и те же алгоритмы могут давать различные результаты, если используется среды разной степени эффективности работы с такими числами.

**Список литературы**

1. *Гудович, А.Н., Гудович Н.Н.* Элементы численных методов. Выпуск 1. Интерполяция алгебраическими многочленами. Многочлен Лагранжа: учебно-методическое пособие для вузов / А.Н. Гудович, Н.Н. Гудович, - Воронеж: Издательско-полиграфический центр Воронежского государственного университета, 2012 г. – 20 с.
2. *Кетков, Ю.Л. и др.* Matlab 7: программирование, численные методы / Ю.Л. Кетков, А.Ю. Кетков, М.М. Шульц, - СпБ.: БХВ-Петербург, 2005 г. – 752 с.

**Приложение (листинг)**

|  |
| --- |
| Код программы на C#, файл Form1.cs |

using System;

using System.Drawing;

using System.Windows.Forms;

using ZedGraph;

using System.Collections.Generic;

using System.Drawing.Drawing2D;

namespace Lab01

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

// Инициализация компонентов:

// Получим панель для рисования

GraphPane pane = zedGraphControl1.GraphPane;

// Изменим тест надписи по оси X

pane.XAxis.Title.Text = "X";

pane.XAxis.Title.FontSpec.FontColor = Color.Blue;

// Изменим текст по оси Y

pane.YAxis.Title.Text = "Y";

pane.YAxis.Title.FontSpec.FontColor = Color.Blue;

// Изменим текст заголовка графика

pane.Title.Text = "Полиномиальная интерполяция (полином Лагранжа)";

}

// Исходная функция, которую аппроксимируем многочленом (можно написать любую)

private double f(double x)

{

return (1 - x) / (1 + x \* x);

}

// Функция вычисления ошибки:

double delta(double y1, double y2)

{

return Math.Abs(y1 - y2);

}

// Основная функция интерполяции

double interpolate2(List<double> X, List<double> Y, double xx)

{

double sum = 0.0; // Сумма-накопитель

for (int i = 0; i < X.Count; i++)

{

double pp = Y[i];

for (int j = 0; j < X.Count; j++)

{

if (i != j)

{

pp \*= (xx - X[j]) / (X[i] - X[j]);

}

}

sum += pp;

}

return sum;

}

// Основная функция интерполяции (альтернативная, раздельно считаются числитель и знаменатель)

double interpolate2alt(List<double> X, List<double> Y, double xx)

{

double sum = 0.0; // Сумма-накопитель

for (int i = 0; i < X.Count; i++)

{

double pp = Y[i];

for (int j = 0; j < X.Count; j++)

{

if (i != j)

{

pp \*= (xx - X[j]); // Считаем числитель

}

}

double pd = 1.0;

for (int j = 0; j < X.Count; j++)

{

if (i != j)

{

pd \*= (X[i] - X[j]); // Считаем знаменатель

}

}

sum += pp / pd;

}

return sum;

}

// Вспомогательная функция интерполяции (+ расчет векторов X и Y)

double interpolate1(double a, double b, int size, double xx)

{

// Передаем в функцию границы отрезка a и b, число разбиений (узлов) этого отрезка size

// и значение xx, для которого ищем y, а также функция f, для которой это все ищется

List<double> X = new List<double>(); // Формируем вектор X (пустой)

List<double> Y = new List<double>(); // Формируем вектор Y (пустой)

double delta = Math.Abs(a - b) / (size - 1); // Находим интервал разбиения

X.Add(a); // Начало отрезка

for (int i = 1; i < size - 1; i++)

{ // Заполняем вектор X

X.Add(X[i - 1] + delta);

}

X.Add(b); // конец отрезка

for (int i = 0; i < size; i++)

{ // Заполняем вектор Y

Y.Add(f(X[i]));

}

// И вызываем основную функцию

return interpolate2(X, Y, xx);

}

// Вспомогательная функция интерполяции (Альтернативная + расчет векторов X и Y)

double interpolate1alt(double a, double b, int size, double xx)

{

// Передаем в функцию границы отрезка a и b, число разбиений (узлов) этого отрезка size

// и значение xx, для которого ищем y, а также функция f, для которой это все ищется

List<double> X = new List<double>(); // Формируем вектор X (пустой)

List<double> Y = new List<double>(); // Формируем вектор Y (пустой)

double delta = Math.Abs(a - b) / (size - 1); // Находим интервал разбиения

X.Add(a); // Начало отрезка

for (int i = 1; i < size - 1; i++)

{ // Заполняем вектор X

X.Add(X[i - 1] + delta);

}

X.Add(b); // конец отрезка

for (int i = 0; i < size; i++)

{ // Заполняем вектор Y

Y.Add(f(X[i]));

}

// И вызываем основную функцию

return interpolate2alt(X, Y, xx);

}

private double f2(double x)

{

if (x == 0)

{

return 1;

}

return Math.Cos(x);

}

private void DrawMashtab()

{

// 1 Читаем значения полей в соотвествующие переменные:

double a = 0.0;

Double.TryParse(txtA.Text, out a);

double b = 0.0;

Double.TryParse(txtB.Text, out b);

int n = 0;

int.TryParse(txtN.Text, out n);

// Получим панель для рисования

GraphPane pane = zedGraphControl1.GraphPane;

// Устанавливаем интересующий нас интервал по оси X

pane.XAxis.Scale.Min = a-0.1; // xmin\_limit;

pane.XAxis.Scale.Max = b+0.1; // xmax\_limit;

// Устанавливаем интересующий нас интервал по оси Y

// Для этого нам надо получить минимум и максимум Y

List<double> X = new List<double>(); // Формируем вектор X (пустой)

List<double> Y = new List<double>(); // Формируем вектор Y (пустой)

double delta = Math.Abs(a - b) / (100); // Находим интервал разбиения

X.Add(a); // Начало отрезка

for (int i = 1; i < 100; i++)

{ // Заполняем вектор X

X.Add(X[i - 1] + delta);

}

X.Add(b); // Начало и конец отрезка

for (int i = 0; i < 100; i++)

{ // Заполняем вектор Y

Y.Add(f(X[i]));

}

Y.Sort();

pane.YAxis.Scale.Min = Y[0] - 0.1; // ymin\_limit;

pane.YAxis.Scale.Max = Y[99] + 0.1; // ymax\_limit;

zedGraphControl1.AxisChange();

// Обновляем график

zedGraphControl1.Invalidate();

}

private void AutoMashtab()

{

GraphPane pane = zedGraphControl1.GraphPane;

// Установим масштаб по умолчанию для оси X

pane.XAxis.Scale.MinAuto = true;

pane.XAxis.Scale.MaxAuto = true;

// Установим масштаб по умолчанию для оси Y

pane.YAxis.Scale.MinAuto = true;

pane.YAxis.Scale.MaxAuto = true;

// Обновим данные об осях

zedGraphControl1.AxisChange();

// Обновляем график

zedGraphControl1.Invalidate();

}

private void btnDraw\_Click(object sender, EventArgs e)

{

// 1 Читаем значения полей в соотвествующие переменные:

double a = 0.0;

Double.TryParse(txtA.Text, out a);

double b = 0.0;

Double.TryParse(txtB.Text, out b);

int n = 0;

int.TryParse(txtN.Text, out n);

// Получим панель для рисования

GraphPane pane = zedGraphControl1.GraphPane;

// Очистим список кривых на тот случай, если до этого сигналы уже были нарисованы

pane.CurveList.Clear();

// 2 Печатаем узловые точки через вспомогательный метод:

plotPoints(a, b, n+1);

// 3 Печатаем график функции на заданном интервале:

plotFunction(a, b, n \* 100);

// 4 Печатаем график полинома на заданном интервале:

plotLagrange(a, b, n \* 100, n + 1);

//DrawGraph();

}

// Вспомогательный метод для печати узловых точек:

private void plotPoints(double a, double b, int size)

{

// Передаем в функцию границы отрезка a и b, число разбиений (узлов) этого отрезка size

// и значение xx, для которого ищем y, а также функция f, для которой это все ищется

List<double> X = new List<double>(); // Формируем вектор X (пустой)

List<double> Y = new List<double>(); // Формируем вектор Y (пустой)

double delta = Math.Abs(a - b) / (size - 1); // Находим интервал разбиения

X.Add(a); // Начало отрезка

for (int i = 1; i < size - 1; i++)

{ // Заполняем вектор X

X.Add(X[i - 1] + delta);

}

X.Add(b); // Начало и конец отрезка

for (int i = 0; i < size; i++)

{ // Заполняем вектор Y

Y.Add(f(X[i]));

}

// Получаем панель для рисования:

GraphPane pane = zedGraphControl1.GraphPane;

// Создадим список точек

PointPairList list = new PointPairList();

for (int i = 0; i < size; i++)

{ // Заполняем список точек:

list.Add(X[i], Y[i]);

}

// Создаем кривую с названием "Узловые точки".

// Обводка кружков будут рисоваться красным цветом (Color.Red),

// Опорные точки - кружки (SymbolType.Circle)

LineItem myCurve = pane.AddCurve("Узловые точки", list, Color.Red, SymbolType.Circle);

// У кривой линия будет невидимой

myCurve.Line.IsVisible = false;

// Размер кружков

myCurve.Symbol.Size = 10;

// Вызываем метод AxisChange (), чтобы обновить данные об осях.

// В противном случае на рисунке будет показана только часть графика,

// которая умещается в интервалы по осям, установленные по умолчанию

zedGraphControl1.AxisChange();

// Обновляем график

zedGraphControl1.Invalidate();

}

// Вспомогательный метод печати графика функции на заданном интервале:

private void plotFunction(double a, double b, int size)

{

// Передаем в функцию границы отрезка a и b, число разбиений (тут уже побольше, чем узлов, раз в 10 побольше) этого отрезка size

// и значение xx, для которого ищем y, а также функция f, для которой это все ищется

List<double> X = new List<double>(); // Формируем вектор X (пустой)

List<double> Y = new List<double>(); // Формируем вектор Y (пустой)

double delta = Math.Abs(a - b) / (size - 1); // Находим интервал разбиения

X.Add(a); // Начало отрезка

for (int i = 1; i < size - 1; i++)

{ // Заполняем вектор X

X.Add(X[i - 1] + delta);

}

X.Add(b); // конец отрезка

for (int i = 0; i < size; i++)

{ // Заполняем вектор Y

Y.Add(f(X[i]));

}

// Получаем панель для рисования:

GraphPane pane = zedGraphControl1.GraphPane;

// Создадим список точек

PointPairList list = new PointPairList();

for (int i = 0; i < size; i++)

{ // Заполняем список точек:

list.Add(X[i], Y[i]);

}

// Создаем кривую с названием "График функции".

LineItem myCurve = pane.AddCurve("График функции", list, Color.Red, SymbolType.None);

// Вызываем метод AxisChange (), чтобы обновить данные об осях.

zedGraphControl1.AxisChange();

// Обновляем график

zedGraphControl1.Invalidate();

}

// Вспомогательный метод печати графика интерполяционного полинома Лагранжа:

private void plotLagrange(double a, double b, int size, int size2)

{

// Передаем в функцию границы отрезка a и b, число разбиений (тут уже побольше, чем узлов, раз в 10 побольше) этого отрезка size

// и значение xx, для которого ищем y, а также функция f, для которой это все ищется

List<double> X = new List<double>(); // Формируем вектор X (пустой)

List<double> Y = new List<double>(); // Формируем вектор Y (пустой)

double delta = Math.Abs(a - b) / (size - 1); // Находим интервал разбиения

X.Add(a); // Начало отрезка

for (int i = 1; i < size - 1; i++)

{ // Заполняем вектор X

X.Add(X[i - 1] + delta);

}

X.Add(b); // конец отрезка

for (int i = 0; i < size; i++)

{ // Заполняем вектор Y

Y.Add(interpolate1(a, b, size2, X[i]));

}

// Получаем панель для рисования:

GraphPane pane = zedGraphControl1.GraphPane;

// Создадим список точек

PointPairList list = new PointPairList();

for (int i = 0; i < size; i++)

{ // Заполняем список точек:

list.Add(X[i], Y[i]);

}

// Создаем кривую с названием "Полином Лагранжа".

LineItem myCurve = pane.AddCurve("Полином Лагранжа", list, Color.Blue, SymbolType.None);

// Вызываем метод AxisChange (), чтобы обновить данные об осях.

zedGraphControl1.AxisChange();

// Обновляем график

zedGraphControl1.Invalidate();

}

// Вспомогательный метод печати графика интерполяционного полинома Лагранжа:

private void plotLagrangeAlt(double a, double b, int size, int size2)

{

// Передаем в функцию границы отрезка a и b, число разбиений (тут уже побольше, чем узлов, раз в 10 побольше) этого отрезка size

// и значение xx, для которого ищем y, а также функция f, для которой это все ищется

List<double> X = new List<double>(); // Формируем вектор X (пустой)

List<double> Y = new List<double>(); // Формируем вектор Y (пустой)

double delta = Math.Abs(a - b) / (size - 1); // Находим интервал разбиения

X.Add(a); // Начало отрезка

//X[0] = a; X[size - 1] = b; // Начало и конец отрезка

for (int i = 1; i < size - 1; i++)

{ // Заполняем вектор X

X.Add(X[i - 1] + delta);

}

X.Add(b); // конец отрезка

for (int i = 0; i < size; i++)

{ // Заполняем вектор Y

Y.Add(interpolate1alt(a, b, size2, X[i]));

}

// Получаем панель для рисования:

GraphPane pane = zedGraphControl1.GraphPane;

// Создадим список точек

PointPairList list = new PointPairList();

for (int i = 0; i < size; i++)

{ // Заполняем список точек:

list.Add(X[i], Y[i]);

}

// Создаем кривую с названием "Полином Лагранжа".

LineItem myCurve = pane.AddCurve("Полином Альтер.", list, Color.Brown, SymbolType.None);

// Вызываем метод AxisChange (), чтобы обновить данные об осях.

zedGraphControl1.AxisChange();

// Обновляем график

zedGraphControl1.Invalidate();

}

private void btnAutoMashtab\_Click(object sender, EventArgs e)

{

AutoMashtab();

}

private void btnMashtab2\_Click(object sender, EventArgs e)

{

DrawMashtab();

}

private void btnDelta\_Click(object sender, EventArgs e)

{

// Вычисление суммарной погрешности для 100 точек между a и b

}

private void btnDelta\_Click\_1(object sender, EventArgs e)

{

// 1 Читаем значения полей в соотвествующие переменные:

double a = 0.0;

Double.TryParse(txtA.Text, out a);

double b = 0.0;

Double.TryParse(txtB.Text, out b);

int n = 0;

int.TryParse(txtN.Text, out n);

int size = 1000;

double sum = 0.0;

// 2

List<double> X = new List<double>(); // Формируем вектор X (пустой)

double delta = Math.Abs(a - b) / (size-1); // Находим интервал разбиения

X.Add(a); // Начало отрезка

for (int i = 1; i < size; i++)

{ // Заполняем вектор X

X.Add(X[i - 1] + delta);

}

X.Add(b); // конец отрезка

for (int i = 0; i < size; i++)

{ // Заполняем вектор Y (

sum =+ Math.Abs(f(X[i])-interpolate1(a, b, n+1, X[i]));

}

// 3

txtDelta.Text = sum.ToString();

}

private void btnDraw2\_Click(object sender, EventArgs e)

{

// 1 Читаем значения полей в соотвествующие переменные:

double a = 0.0;

Double.TryParse(txtA.Text, out a);

double b = 0.0;

Double.TryParse(txtB.Text, out b);

int n = 0;

int.TryParse(txtN.Text, out n);

// Получим панель для рисования

GraphPane pane = zedGraphControl1.GraphPane;

// Очистим список кривых на тот случай, если до этого сигналы уже были нарисованы

pane.CurveList.Clear();

// 2 Печатаем узловые точки через вспомогательный метод:

plotPoints(a, b, n + 1);

// 3 Печатаем график функции на заданном интервале:

plotFunction(a, b, n \* 100);

// 4 Печатаем график полинома на заданном интервале:

plotLagrangeAlt(a, b, n \* 100, n + 1);

}

}

}

|  |
| --- |
| Модуль MatLab **f.m** |

function Y=f(X)

Y = X.^2-3;

end

|  |
| --- |
| Модуль MatLab interpolate1.m |

function [Lx, Fx, delta] = interpolate1(a, b, size, xx)

% Вспомогательная функция интерполяции (+ расчет весторов X и Y)

% Передаем в функцию границы отрезка a и b, степень полинома (Многочлен

% степени 2 = число узлов 3) и значение xx, для которого ищем y

d = abs(a-b)/size; % Расстояние между узлами, size - степень полинома

X = a:d:b; % Формируем вектор X

Y = f(X); % Заполняем вектор Y Иcпользуем функцию, прописанную в файле f.m

coef = polyfit(X, Y, size); % Вычисляем коэффициенты интерполяционного полинома

Lx = polyval(coef, xx); % Вычисляем значение в точке xx по найденному полиному

Fx = f(xx); % Вычисляем значение функции в точке xx

delta = abs(Lx - Fx); % Вычисляем ошибку

% Дальше будем строить графики:

xlabel('x'); ylabel('y'); hold on; grid on; % Подписываем оси, включаем сетку

plot (X, Y, 'ro'); % Печатаем узловые точки

XX = a:0.01:b; % Формируем точки для графика многочлена Лагранжа

LX = polyval(coef, XX); % И соответствующие им Y-ки

plot(XX, LX, 'b--'); % Печатаем график полинома (интерполирующей функции)

plot(xx, Lx, 'go'); % Печатаем точку полинома

plot(xx, Fx, 'rx'); % Печатаем точку функции

% И подписываем оси и легенду:

title('Полиномиальная интерполяция', 'FontName', 'Courier');

h1 = legend('Узловые точки', 'Интерполяция', 'Точка полинома', 'Точка функции');

set(h1, 'FontName', 'Courier');

end

|  |
| --- |
| Модуль MatLab interpolate2.m |

function [Lx, Fx, delta] = interpolate2(X, Y, xx)

% Основная функция интерполяции

% Передаем два вектора узловых точек (X и Y) и интересующую точку xx

coef = polyfit(X, Y, size(X, 1)-1); % Вычисляем коэффициенты интерполяционного полинома

Lx = polyval(coef, xx); % Вычисляем значение в точке xx по найденному полиному

Fx = f(xx); % Вычисляем значение функции в точке xx

delta = abs(Lx - Fx); % Вычисляем ошибку

% Дальше будем строить графики:

xlabel('x'); ylabel('y'); hold on; grid on; % Подписываем оси, включаем сетку

plot (X, Y, 'ro'); % Печатаем узловые точки

XX = X(1):0.01:X(size(X, 1)); % Формируем точки для графика многочлена Лагранжа

LX = polyval(coef, XX); % И соответствующие им Y-ки

plot(XX, LX, 'b--'); % Печатаем график полинома (интерполирующей функции)

plot(xx, Lx, 'go'); % Печатаем точку полинома

plot(xx, Fx, 'rx'); % Печатаем точку функции

% И подписываем оси и легенду:

title('Полиномиальная интерполяция', 'FontName', 'Courier');

h1 = legend('Узловые точки', 'Интерполяция', 'Точка полинома', 'Точка функции');

set(h1, 'FontName', 'Courier');

end

|  |
| --- |
| Модуль MatLab interpolate3.m |

function interpolate3(a, b, size)

% 1 Находим коэффициенты полинома Лагранжа:

d = abs(a-b)/size;

X = a:d:b;

Y = f(X);

coef = polyfit(X, Y, size);

% 2 Создаем новое окно для графика и подписываем оси

figure;

xlabel('x');

ylabel('y');

hold on;

grid on;

% 3 Печатаем узловые точки:

plot (X, Y, 'ro');

% Печатаем график полинома Лагранжа:

XX = a:0.01:b;

LX = polyval(coef, XX);

plot(XX, LX, 'b');

% 4 Печатаем график функции:

X = XX;

Y = f(X);

plot(X, Y, 'r--');

% 5 Подписываем легенду

title('Полиномиальная интерполяция', 'FontName', 'Courier');

h1 = legend('Узловые точки', 'Интерполяция', 'Функция');

set(h1, 'FontName', 'Courier');

% 6 Выставляем более-менее приемлемый масштаб:

axis([a b min(Y)-0.2 max(Y)+0.2])

end

|  |
| --- |
| Модуль MatLab delta.m |

function d = delta(a, b, size)

% Функция вычисления погрешности между функцией и интерполяционным

% многочленом на отрезке для данной степени полинома

% 1 Находим коэффициенты полинома Лагранжа:

d = abs(a-b)/size;

X = a:d:b;

Y = f(X);

coef = polyfit(X, Y, size);

% 2 Считаем значения полинома Лагранжа:

XX = a:0.01:b;

LX = polyval(coef, XX);

% 3 Считаем значения функции:

Y = f(XX);

% 4 Считаем погрешность на отрезке:

Z = Y - LX;

Z = abs(Z);

d = sum(Z);

end

|  |
| --- |
| Модуль MatLab deltaPlot.m |

function deltaPlot(a, b, size1, size2)

X = size1:1:size2;

Y= X \* 0;

for i = 1:1:size(X, 2)

Y(i) = delta(a,b,X(i));

end

% 2 Создаем новое окно для графика и подписываем оси

figure;

xlabel('Степень полинома Лагранжа, n');

ylabel('Абсолютная погрешность');

grid on; hold on;

% 3 Печатаем узловые точки:

plot (X, Y, 'b');

% 5 Подписываем

title('Погрешность интерполяции', 'FontName', 'Courier');

% 6 Выставляем более-менее приемлемый масштаб:

axis([size1 size2 0-2 60+2])

end

|  |
| --- |
| Модуль MatLab polyval1.m |

function Lf = polyval1(X, Y, xx)

% Первый метод вычисления значения интерполяционного многочлена в точке с

% использованием многчлена Лагранжа (в одном цикле и умножение, и деление)

sum = 0.0;

for i = 1:1:size(X, 2)

pp = Y(i);

for j = 1:1:size(X, 2)

if (i ~= j )

pp = pp \* (xx - X(j))/(X(i)-X(j));

end

end

sum = sum + pp;

end

Lf = sum;

end

|  |
| --- |
| Модуль MatLab polyval2.m |

function Lf = polyval2(X, Y, xx)

% Второй метод вычисления значения интерполяционного многочлена в точке с

% использованием многчлена Лагранжа (в одном цикле умножение, а в другом -

% деление)

sum = 0.0;

for i = 1:1:size(X, 2)

pp = Y(i); % Числитель

for j = 1:1:size(X, 2)

if (i ~= j )

pp = pp \* (xx - X(j));

end

end

pd = 1.0; % Значенатель

for j = 1:1:size(X, 2)

if (i ~= j )

pd = pd \* (X(i)-X(j));

end

end

sum = sum + pp/pd;

end

Lf = sum;

end

|  |
| --- |
| Модуль MatLab interpolate4diff.m |

function interpolate4diff(a, b, size)

% 1 Находим коэффициенты полинома Лагранжа:

d = abs(a-b)/size;

X = a:d:b;

Y = f(X);

% 2 Создаем новое окно для графика и подписываем оси

figure;

xlabel('x');

ylabel('y');

hold on;

grid on;

% 3 Печатаем узловые точки:

plot (X, Y, 'ro');

% Печатаем график полинома Лагранжа (первый способ):

XX = a:0.01:b;

LX = XX \* 0;

for i = 1:1:length(XX)

LX(i) = polyval1(X, Y, XX(i));

end

plot(XX, LX, 'b');

% Печатаем график полинома Лагранжа (второй способ):

XX = a:0.01:b;

LX = XX \* 0;

for i = 1:1:length(XX)

LX(i) = polyval2(X, Y, XX(i));

end

plot(XX, LX, 'g');

% 4 Печатаем график функции:

X = XX;

Y = f(X);

plot(X, Y, 'r--');

% 5 Подписываем легенду

title('Полиномиальная интерполяция', 'FontName', 'Courier');

h1 = legend('Узловые точки','Интерполяция 1','Интерполяция 2','Функция');

set(h1, 'FontName', 'Courier');

% 6 Выставляем более-менее приемлемый масштаб:

axis([a b min(Y)-0.2 max(Y)+0.2])

end