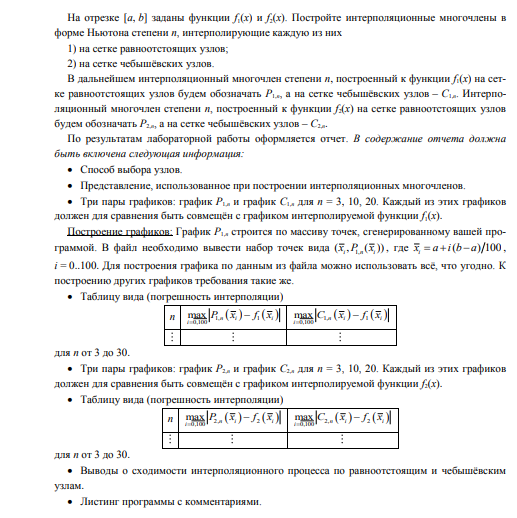
Отчет по дисциплине: «Численные методы»

Лабораторная работа №1

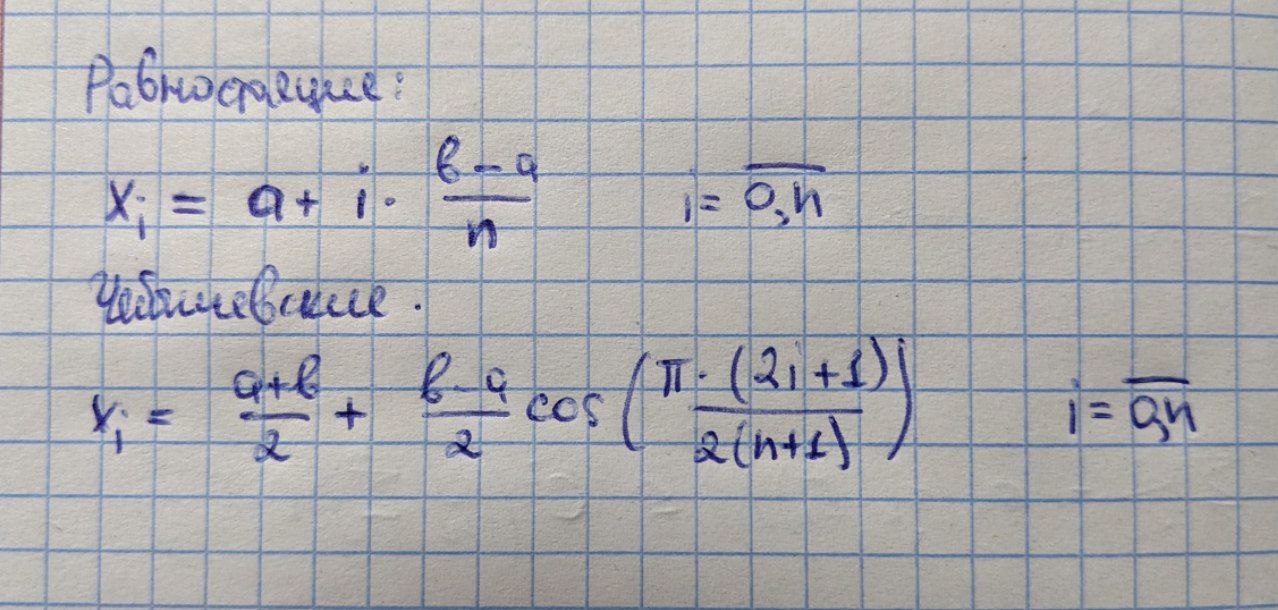
## «Интерполяция алгебраическими многочленами»

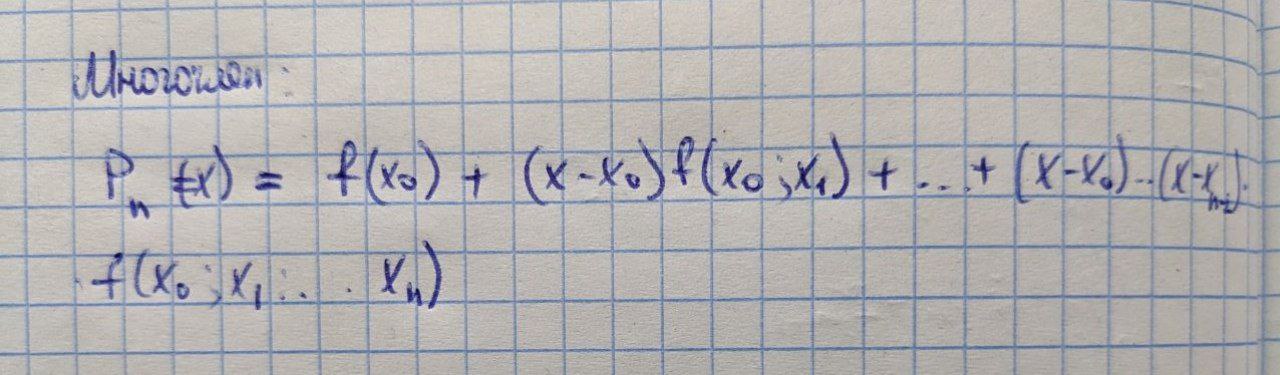
Подготовил студент 3 курса 4 группы

Кондратович Артём



Использованные формулы:





Листинг программы:

using lab\_1\_.HelpClasses;

using OxyPlot;

using OxyPlot.Legends;

using OxyPlot.Series;

using System.Drawing;

using System.Text;

namespace lab\_1\_

{

public partial class Form1 : Form

{

private readonly Function \_functions;

public Form1()

{

InitializeComponent();

\_functions = new Function(new MyRange(-3, 3));

}

public LineSeries CalculatePn(int n, int type)

{

var name = type switch

{

1 => $"P1",

2 => $"P2",

\_ => throw new NotSupportedException()

};

var result = CalculatePolynomial(type, n, 1);

var table = GetErrorTable(type, name, 1);

var stringBuilder = new StringBuilder();

foreach (var line in table)

{

stringBuilder.AppendLine(line);

}

Pn.Text = stringBuilder.ToString();

var line1 = new LineSeries()

{

Title = name,

Color = OxyColors.Blue,

StrokeThickness = 1,

MarkerSize = 2,

MarkerType = MarkerType.Circle

};

foreach (var item in result.First)

{

line1.Points.Add(new DataPoint(item.First, item.Second));

}

return line1;

}

public LineSeries CalculateCn(int n, int type)

{

var name = type switch

{

1 => $"C1",

2 => $"C2",

\_ => throw new NotSupportedException()

};

var result = CalculatePolynomial(type, n, 2);

var table = GetErrorTable(type, name, 2);

var stringBuilder = new StringBuilder();

foreach (var line in table)

{

stringBuilder.AppendLine(line);

}

Cn.Text = stringBuilder.ToString();

var line1 = new LineSeries()

{

Title = name,

Color = OxyColors.Red,

StrokeThickness = 1,

MarkerSize = 2,

MarkerType = MarkerType.Circle

};

foreach (var item in result.First)

{

line1.Points.Add(new DataPoint(item.First, item.Second));

}

return line1;

}

private void PrintReport(int n, int type)

{

var pn = CalculatePn(n, type);

var cn = CalculateCn(n, type);

var function = type switch

{

1 => \_functions.F1,

2 => \_functions.F2,

\_ => throw new NotSupportedException()

};

var myModel = new PlotModel { Title = "" };

myModel.Legends.Add(new Legend()

{

LegendTitle = "",

LegendPosition = LegendPosition.RightTop

});

myModel.Series.Add(new FunctionSeries(function, \_functions.Range.A, \_functions.Range.B, 0.001));

myModel.Series.Add(pn);

myModel.Series.Add(cn);

plotView1.Model = myModel;

}

private void Button1\_Click(object sender, EventArgs e)

{

if (f1.Checked)

{

if (int.TryParse(textBox1.Text, out var n))

{

PrintReport(n, 1);

}

else

{

textBox1.Text = "Wrong number!";

}

}

if (f2.Checked)

{

if (int.TryParse(textBox1.Text, out var n))

{

PrintReport(n, 2);

}

else

{

textBox1.Text = "Wrong number!";

}

}

}

private void F1\_CheckedChanged(object sender, EventArgs e)

{

if (f1.Checked)

{

f2.Checked = false;

}

else

{

f2.Checked = true;

}

}

private void F2\_CheckedChanged(object sender, EventArgs e)

{

if (f2.Checked)

{

f1.Checked = false;

}

else

{

f1.Checked = true;

}

}

private string[] GetErrorTable(int type, string polynomial, int knotsType)

{

var result = new List<string>();

for (var i = 3; i <= 30; i++)

{

var pair = CalculatePolynomial(type, i, knotsType);

var max = 0.0;

for (var j = 0; j < pair.First.Count; j++)

{

max = Math.Max(Math.Abs(pair.First[j].Second - pair.Second[j]), max);

}

result.Add($"n = {i}: max|{polynomial} - f{type}| = {max}");

}

return [.. result];

}

private Pair<List<Pair<double, double>>, List<double>> CalculatePolynomial(int type, int n, int knotsType)

{

var knots = knotsType switch

{

1 => \_functions.Range.GetEquidistantKnots(n),

2 => \_functions.Range.GetChebyshevKnots(n),

\_ => throw new NotSupportedException()

};

var function = type switch

{

1 => \_functions.F1,

2 => \_functions.F2,

\_ => throw new NotSupportedException()

};

var matrix = new double[n + 1, n + 1];

for (var i = 0; i < n + 1; i++)

{

matrix[0, i] = function(knots[i]);

}

for (var i = 1; i < n + 1; i++)

{

for (var j = 0; j < n + 1 - i; j++)

{

matrix[i, j] = (matrix[i - 1, j + 1] - matrix[i - 1, j]) / (knots[i + j] - knots[j]);

}

}

var polynomialValues = new List<Pair<double, double>>();

var points = \_functions.Range.GetPoints();

var accurateValues = new List<double>();

foreach (var point in points)

{

var sum = 0.0;

for (var i = 0; i < n + 1; i++)

{

var temp = matrix[i, 0];

for (var j = 0; j < i; j++)

{

temp \*= (point - knots[j]);

}

sum += temp;

}

polynomialValues.Add(new Pair<double, double>(point, sum));

accurateValues.Add(function(point));

}

return new Pair<List<Pair<double, double>>, List<double>>(polynomialValues, accurateValues);

}

}

}

Результаты продемонстрированы во время выполнения программы