public partial class MMACSMainViewModel : BindableBase

{

/// <summary>

/// a0,ai,aj,aij,aii,ajj

/// </summary>

private const int \_coefficientArrayCount = 6;

private int iteration;

private int \_rowCount;

private int \_columnCount;

private int \_trainingRowsCount;

private int \_checkingRowsCount;

private int \_columnIndexOutputVariable;

private double[][] \_originalTable;

private double[][] \_dataTraining;

private double[][] \_dataChecking;

private void SelectFile()

{

var openFileDialog = new OpenFileDialog();

var showDialog = openFileDialog.ShowDialog();

if (showDialog != null && showDialog.Value)

{

var fileName = openFileDialog.FileName;

if (fileName.Last() != 'x')

{

fileName = Convert(fileName);

}

var newFile = new FileInfo(fileName);

var pck = new ExcelPackage(newFile);

var firstWorksheet = pck.Workbook.Worksheets.First();

\_rowCount = firstWorksheet.Dimension.End.Row;

\_columnCount = firstWorksheet.Dimension.End.Column;

\_originalTable = new double[\_rowCount][];

for (var rowNum = 0; rowNum < \_rowCount; rowNum++)

{

\_originalTable[rowNum] = new double[\_columnCount];

var wsRow = firstWorksheet.Cells[rowNum + 1, 1, rowNum + 1, \_columnCount];

foreach (var cell in wsRow)

{

double r;

double.TryParse(cell.Text, out r);

\_originalTable[rowNum][cell.Start.Column - 1] = r;

}

}

OriginalMatrix = Fill(\_originalTable);

OnPropertyChanged(nameof(OriginalMatrix));

ConfigTrainingData();

ConfigCheckingData();

Selection.Clear();

OnPropertyChanged(nameof(Selection));

iteration = 0;

Mgua(new MguaParams

{

Sample = \_originalTable

});

}

}

public string Convert(string fileName)

{

var app = new Microsoft.Office.Interop.Excel.Application();

var wb = app.Workbooks.Open(fileName);

var newName = fileName + "x";

if (File.Exists(newName))

{

return newName;

}

wb.SaveAs(Filename: newName, FileFormat: Microsoft.Office.Interop.Excel.XlFileFormat.xlOpenXMLWorkbook);

wb.Close();

app.Quit();

return newName;

}

private void ConfigTrainingData()

{

\_trainingRowsCount = (int)Math.Ceiling((double)\_rowCount / 2);

\_dataTraining = new double[\_trainingRowsCount][];

for (int i = 0; i < \_trainingRowsCount; ++i)

{

\_dataTraining[i] = new double[\_columnCount];

for (int j = 0; j < \_columnCount; j++)

{

\_dataTraining[i][j] = \_originalTable[i][j];

}

}

TrainingData = Fill(\_dataTraining);

OnPropertyChanged(nameof(TrainingData));

}

private void ConfigCheckingData()

{

\_checkingRowsCount = \_rowCount - \_trainingRowsCount;

\_dataChecking = new double[\_checkingRowsCount][];

for (int i = \_trainingRowsCount; i < \_rowCount; ++i)

{

var currentCheckingRow = i - \_trainingRowsCount;

\_dataChecking[currentCheckingRow] = new double[\_columnCount];

for (int j = 0; j < \_columnCount; j++)

{

\_dataChecking[currentCheckingRow][j] = \_originalTable[i][j];

}

}

CheckingData = Fill(\_dataChecking);

OnPropertyChanged(nameof(CheckingData));

}

private DataView Fill(double[][] source)

{

var rowCount = source.Length;

var columnCount = (rowCount > 0 ? source[0].Length : 0);

var tbl = new DataTable();

for (int c = 0; c < columnCount; c++)

{

tbl.Columns.Add(new DataColumn(c.ToString()));

}

for (int r = 0; r < rowCount; r++)

{

var row = tbl.NewRow();

for (int c = 0; c < columnCount; c++)

{

row[c] = source[r][c];

}

tbl.Rows.Add(row);

}

return tbl.DefaultView;

}

/// <summary>

/// Многорядный полиномиальный алгоритм МГУА.

/// Опорная функция - полином вида

/// y = a0 + a1x1 + a2x2 + a3x1^2 + a4x2^2 + a5x1x2

/// </summary>

/// <param name="mguaParams">Исходные данные или данные предыдущего ряда селекции.</param>

private void Mgua(MguaParams mguaParams)

{

// 1. Выборка делится на обучающую и проверочную.

// Nвыб = Nобуч + Nпров

var trainingCheckingSample = DivideTrainingCheckingSample(mguaParams.Sample);

var parX = GetAllParX(mguaParams.Sample);

// 2. На обучающей выборке вычисляются коэффициенты регрессии.

// a0-a5

var regressionCoefficients = EvaluateRegressionCoefficients(trainingCheckingSample.TrainingSample);

// 3. На проверочной выборке отбираются лучшие модели.

var bestModels = GetBestModels(trainingCheckingSample.CheckingSample, regressionCoefficients, parX);

// 4. Проверяется критерий E^2 -> min.

double epsilonSquaredMin = bestModels.First().Value;

FillResult(iteration++, epsilonSquaredMin, regressionCoefficients, bestModels.Values.ToArray(), bestModels.Keys.ToArray());

// 6. Операция повторяется до тех пор, пока не выполнится условие.

if (mguaParams.EpsilonSquaredMin.HasValue && mguaParams.EpsilonSquaredMin < epsilonSquaredMin)

{

return;

}

// 5. Лучшие модели используются для расчета новых аргументов.

// Расчет переменных для следующего шага селекции.

mguaParams.Sample = GetNewVariables(mguaParams.Sample, regressionCoefficients, parX, bestModels.Keys.ToArray());

// Фиксация E^2s min для сравнения на следующем ряде селекции.

// (s - номер селекции)

mguaParams.EpsilonSquaredMin = epsilonSquaredMin;

Mgua(mguaParams);

}

private Row[] GetAllParX(double[][] sample)

{

var rowCount = sample.Length;

var result = new List<Row>();

for (int rowNumber = 0; rowNumber < rowCount; rowNumber++)

{

var x1 = sample[rowNumber][0];

var x2 = sample[rowNumber][1];

var x3 = sample[rowNumber][2];

var x4 = sample[rowNumber][3];

result.Add(new Row

{

Pars = new[]

{

new FunctionPar

{

Name = "x1x2",

X1 = x1,

X2 = x2

},

new FunctionPar

{

Name = "x1x3",

X1 = x1,

X2 = x3

},

new FunctionPar

{

Name = "x1x4",

X1 = x1,

X2 = x4

},

new FunctionPar

{

Name = "x2x3",

X1 = x2,

X2 = x3

},

new FunctionPar

{

Name = "x2x4",

X1 = x2,

X2 = x4

},

new FunctionPar

{

Name = "x3x4",

X1 = x3,

X2 = x4

},

}

});

}

return result.ToArray();

}

private double[][] GetNewVariables(double[][] mguaParamsSample, double[][] coefArr, Row[] allRowFunctions, string[] bestModelsSortedFunctionName)

{

var rowCount = mguaParamsSample.Length;

var result = new double[rowCount][];

for (int rowNumber = 0; rowNumber < rowCount; rowNumber++)

{

var colCount = mguaParamsSample[rowNumber].Length;

result[rowNumber] = new double[colCount];

for (int colNumber = 0; colNumber < colCount - 1; colNumber++)

{

var functionName = bestModelsSortedFunctionName[colNumber];

result[rowNumber][colNumber] = GetVariable(coefArr[colNumber], allRowFunctions[rowNumber].Pars

.First(x => x.Name == functionName)

.X1, allRowFunctions[rowNumber].Pars

.First(x => x.Name == functionName)

.X2);

}

result[rowNumber][colCount - 1] = mguaParamsSample[rowNumber][colCount - 1];

}

return result;

}

private double GetVariable(double[] coefArr, double x1, double x2)

{

var a0 = coefArr[0];

var a1 = coefArr[1];

var a2 = coefArr[2];

var a3 = coefArr[3];

var a4 = coefArr[4];

var a5 = coefArr[5];

return a0 + a1 \* x1 + a2 \* x2 + a3 \* x1 \* x1 + a4 \* x2 \* x2 + a5 \* x1 \* x2;

}

/// <summary>

/// Делим выборку на обучающую и проверочную.

/// </summary>

/// <returns></returns>

private TrainingCheckingSample DivideTrainingCheckingSample(double[][] sample)

{

var training = new List<double[]>();

var checking = new List<double[]>();

for (int i = 0; i < sample.Length; i++)

{

// 2,4,6 ... проверочная

if (i % 2 == 0)

{

training.Add(sample[i]);

}

// обучающая

else

{

checking.Add(sample[i]);

}

}

return new TrainingCheckingSample

{

TrainingSample = training.ToArray(),

CheckingSample = checking.ToArray()

};

}

/// <summary>

/// Вычисляум коэффициенты регрессии.

/// </summary>

/// <param name="trainingSample"></param>

/// <returns>Массив коэффициентов для строк.</returns>

private double[][] EvaluateRegressionCoefficients(double[][] trainingSample)

{

var trainingRowsCount = trainingSample.Length;

#region Создаем и инициализируем матрицы

double[][] matr1 = new double[trainingRowsCount][];

double[][] matr2 = new double[trainingRowsCount][];

double[][] matr3 = new double[trainingRowsCount][];

double[][] matr4 = new double[trainingRowsCount][];

double[][] matr5 = new double[trainingRowsCount][];

double[][] matr6 = new double[trainingRowsCount][];

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

{

matr1[i] = new double[\_coefficientArrayCount];

matr2[i] = new double[\_coefficientArrayCount];

matr3[i] = new double[\_coefficientArrayCount];

matr4[i] = new double[\_coefficientArrayCount];

matr5[i] = new double[\_coefficientArrayCount];

matr6[i] = new double[\_coefficientArrayCount];

}

#endregion Создаем и инициализируем матрицы

// Требуется создать 4\*(4-1)/2 = 6 функций с попарным сочетанием входных переменных

// функции вида y = a0 + a1 \*x1 + a2\* x2 + a3\*x1\*x1 + a4\*x2\*x2 + a5\*x1\*x2

// для решения имеющимся набором библиотечных функций зададим матрицы, в которых приведем эти уравнения к

// линейному виду, то есть заранее рассчитаем перемножения иксов.

for (int rowNumber = 0; rowNumber < trainingRowsCount; ++rowNumber)

{

var x1 = trainingSample[rowNumber][0];

var x2 = trainingSample[rowNumber][1];

var x3 = trainingSample[rowNumber][2];

var x4 = trainingSample[rowNumber][3];

var y = trainingSample[rowNumber][4];

// fi(x1,x2)

// x1 x2 x1^2 x2^2 x1\*x2 y

matr1[rowNumber][0] = x1;

matr1[rowNumber][1] = x2;

matr1[rowNumber][2] = x1 \* x1;

matr1[rowNumber][3] = x2 \* x2;

matr1[rowNumber][4] = x1 \* x2;

matr1[rowNumber][5] = y;

// fi(x1,x3)

// x1 x3 x1^2 x3^2 x1\*x3 y

matr2[rowNumber][0] = x1;

matr2[rowNumber][1] = x3;

matr2[rowNumber][2] = x1 \* x1;

matr2[rowNumber][3] = x3 \* x3;

matr2[rowNumber][4] = x1 \* x3;

matr2[rowNumber][5] = y;

// fi(x1,x4)

// x1 x4 x1^2 x4^2 x1\*x4 y

matr3[rowNumber][0] = x1;

matr3[rowNumber][1] = x4;

matr3[rowNumber][2] = x1 \* x1;

matr3[rowNumber][3] = x4 \* x4;

matr3[rowNumber][4] = x1 \* x4;

matr3[rowNumber][5] = y;

// fi(x2,x3)

// x2 x3 x2^2 x3^2 x2\*x3 y

matr4[rowNumber][0] = x2;

matr4[rowNumber][1] = x3;

matr4[rowNumber][2] = x2 \* x2;

matr4[rowNumber][3] = x3 \* x3;

matr4[rowNumber][4] = x2 \* x3;

matr4[rowNumber][5] = y;

// fi(x2,x4)

//x2 x4 x2^2 x4^2 x2\*x4 y

matr5[rowNumber][0] = x2;

matr5[rowNumber][1] = x4;

matr5[rowNumber][2] = x2 \* x2;

matr5[rowNumber][3] = x4 \* x4;

matr5[rowNumber][4] = x2 \* x4;

matr5[rowNumber][5] = y;

// fi(x3,x4)

// x3 x4 x3^2 x4^2 x3\*x4 y

matr6[rowNumber][0] = x3;

matr6[rowNumber][1] = x4;

matr6[rowNumber][2] = x3 \* x3;

matr6[rowNumber][3] = x4 \* x4;

matr6[rowNumber][4] = x3 \* x4;

matr6[rowNumber][5] = y;

}

double[][] design1 = MathixHelper.Design(matr1);

double[][] design2 = MathixHelper.Design(matr2);

double[][] design3 = MathixHelper.Design(matr3);

double[][] design4 = MathixHelper.Design(matr4);

double[][] design5 = MathixHelper.Design(matr5);

double[][] design6 = MathixHelper.Design(matr6);

double[][] coefArr = new double[6][];

// a0

coefArr[0] = MathixHelper.Solve(design1);

// a1

coefArr[1] = MathixHelper.Solve(design2);

// a2

coefArr[2] = MathixHelper.Solve(design3);

// a3

coefArr[3] = MathixHelper.Solve(design4);

// a4

coefArr[4] = MathixHelper.Solve(design5);

// a5

coefArr[5] = MathixHelper.Solve(design6);

return coefArr;

}

/// <summary>

/// Отбираем лучшие модели.

/// </summary>

/// <param name="checkingSample"></param>

/// <param name="regressionCoefficients"></param>

/// <returns></returns>

private Dictionary<string, double> GetBestModels(double[][] checkingSample, double[][] regressionCoefficients, Row[] allRows)

{

var checkingRowsCount = checkingSample.Length;

double epsSquareAcc1 = 0;

double epsSquareAcc2 = 0;

double epsSquareAcc3 = 0;

double epsSquareAcc4 = 0;

double epsSquareAcc5 = 0;

double epsSquareAcc6 = 0;

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

{

var x1 = checkingSample[i][0];

var x2 = checkingSample[i][1];

var x3 = checkingSample[i][2];

var x4 = checkingSample[i][3];

var y = checkingSample[i][\_columnIndexOutputVariable];

epsSquareAcc1 += Math.Pow(y - MathixHelper.solveFunction(regressionCoefficients[0], x1, x2), 2);

epsSquareAcc2 += Math.Pow(y - MathixHelper.solveFunction(regressionCoefficients[1], x1, x3), 2);

epsSquareAcc3 += Math.Pow(y - MathixHelper.solveFunction(regressionCoefficients[2], x1, x4), 2);

epsSquareAcc4 += Math.Pow(y - MathixHelper.solveFunction(regressionCoefficients[3], x2, x3), 2);

epsSquareAcc5 += Math.Pow(y - MathixHelper.solveFunction(regressionCoefficients[4], x2, x4), 2);

epsSquareAcc6 += Math.Pow(y - MathixHelper.solveFunction(regressionCoefficients[5], x3, x4), 2);

}

// пара имя функции и значение E^2

var result = new Dictionary<string, double>();

result["x1x2"] = (double)epsSquareAcc1 / checkingRowsCount;

result["x1x3"] = (double)epsSquareAcc2 / checkingRowsCount;

result["x1x4"] = (double)epsSquareAcc3 / checkingRowsCount;

result["x2x3"] = (double)epsSquareAcc4 / checkingRowsCount;

result["x2x4"] = (double)epsSquareAcc5 / checkingRowsCount;

result["x3x4"] = (double)epsSquareAcc6 / checkingRowsCount;

return result.OrderBy(x => x.Value).Take(4).ToDictionary(x => x.Key, x => x.Value);

}

private double[][] Split(double[][] checkingSample, double[][] trainingSample)

{

var len = checkingSample.Length + trainingSample.Length;

var ci = 0;

var ti = 0;

var result = new List<double[]>();

for (int rowNumber = 0; rowNumber < len; rowNumber++)

{

if (rowNumber % 2 == 0)

{

result.Add(trainingSample[ti]);

ti++;

}

else

{

result.Add(checkingSample[ci]);

ci++;

}

}

return result.ToArray();

}

private void FillResult(int iteration, double epsilonSquaredCurrent, double[][] coefArr, double[] epsArray, string[] bestFunc)

{

var sel = new ResultViewModel

{

Step = iteration.ToString(),

Value = epsilonSquaredCurrent.ToString(),

};

var d = new Dictionary<string, string>();

d["x1x2"] =

$"f1(x1,x2) = {coefArr[0][0]:#.##} + {coefArr[0][1]:#.##} \* x1 + {coefArr[0][2]:#.##} \* x2 + {coefArr[0][3]:#.##} \* x1^2 + {coefArr[0][4]:#.##} \* x2^2 + {coefArr[0][5]:#.##} \* x1 \* x2";

d["x1x3"] =

$"f2(x1,x3) = {coefArr[1][0]:#.##} + {coefArr[1][1]:#.##} \* x1 + {coefArr[1][2]:#.##} \* x3 + {coefArr[1][3]:#.##} \* x1^2 + {coefArr[1][4]:#.##} \* x3^2 + {coefArr[1][5]:#.##} \* x1 \* x3";

d["x1x4"] =

$"f3(x1,x4) = {coefArr[2][0]:#.##} + {coefArr[2][1]:#.##} \* x1 + {coefArr[2][2]:#.##} \* x4 + {coefArr[2][3]:#.##} \* x1^2 + {coefArr[2][4]:#.##} \* x4^2 + {coefArr[2][5]:#.##} \* x1 \* x4";

d["x2x3"] =

$"f4(x2,x3) = {coefArr[3][0]:#.##} + {coefArr[3][1]:#.##} \* x2 + {coefArr[3][2]:#.##} \* x3 + {coefArr[3][3]:#.##} \* x2^2 + {coefArr[3][4]:#.##} \* x3^2 + {coefArr[3][5]:#.##} \* x2 \* x3";

d["x2x4"] =

$"f5(x2,x4) = {coefArr[4][0]:#.##} + {coefArr[4][1]:#.##} \* x2 + {coefArr[4][2]:#.##} \* x4 + {coefArr[4][3]:#.##} \* x2^2 + {coefArr[4][4]:#.##} \* x4^2 + {coefArr[4][5]:#.##} \* x2 \* x4";

d["x3x4"] =

$"f6(x3,x4) = {coefArr[5][0]:#.##} + {coefArr[5][1]:#.##} \* x3 + {coefArr[5][2]:#.##} \* x4 + {coefArr[5][3]:#.##} \* x3^2 + {coefArr[5][4]:#.##} \* x4^2 + {coefArr[5][5]:#.##} \* x3 \* x4";

foreach (var s in bestFunc)

{

sel.Functions.Add(d[s]);

}

sel.Eps.Add($"Eps^2 = {epsArray[0]:#.##}");

sel.Eps.Add($"Eps^2 = {epsArray[1]:#.##}");

sel.Eps.Add($"Eps^2 = {epsArray[2]:#.##}");

sel.Eps.Add($"Eps^2 = {epsArray[3]:#.##}");

//sel.Eps.Add($"Eps^2 = {epsArray[4]:#.##}");

//sel.Eps.Add($"Eps^2 = {epsArray[5]:#.##}");

Selection.Add(sel);