**Цель работы** – Изучение метода группового учета аргументов

1. Задание на лабораторную работу

Определить связь между входными и выходными параметрами задачи методом группового учета аргументов.

Вариант задания выбрать в соответствии с номером по журналу.

Для решения задачи использовать полиномиальный алгоритм МГУА

В процессе решения задачи предусмотреть:

1. Ввод исходной таблицы данных (чтение данных из файла в формате .xls)

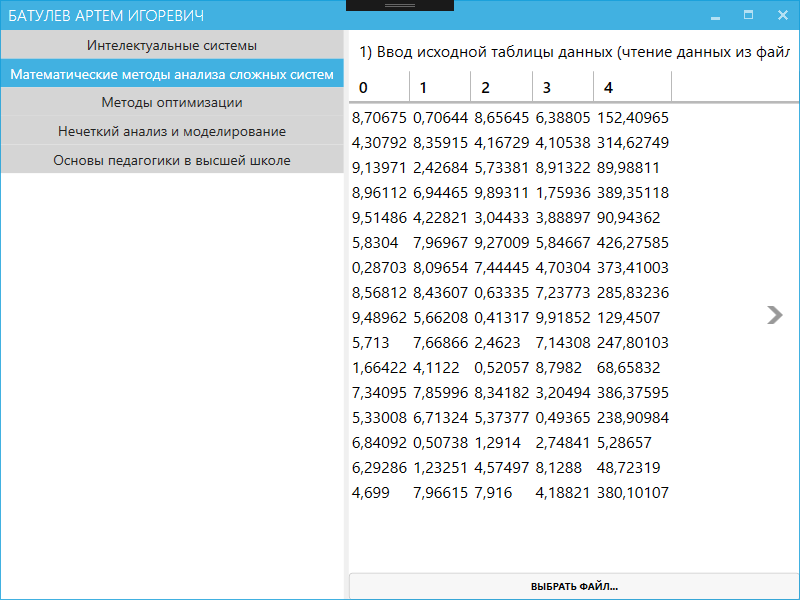
Формат таблицы данных

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |

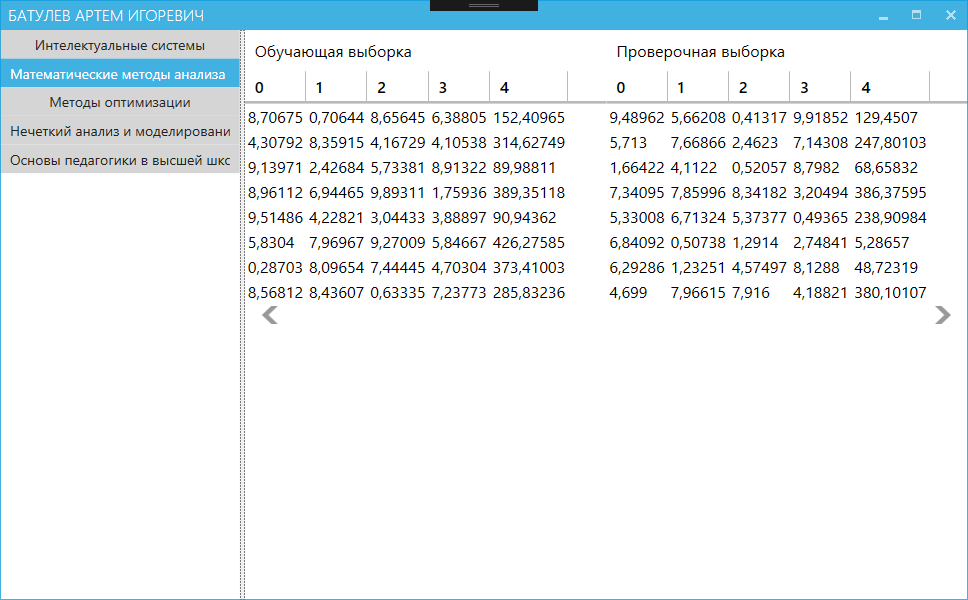
1. Решение задачи аппроксимации, вычисление коэффициентов опорного полинома, вычисление критерия качества
2. Отображение критерия качества на каждом ряде селекции (в текстовом или графическом виде)
3. Остановку процесса при достижении минимума критерия качества
4. Отображение итоговой функции, вычисление ошибки аппроксимации

2. Алгоритм решения задачи

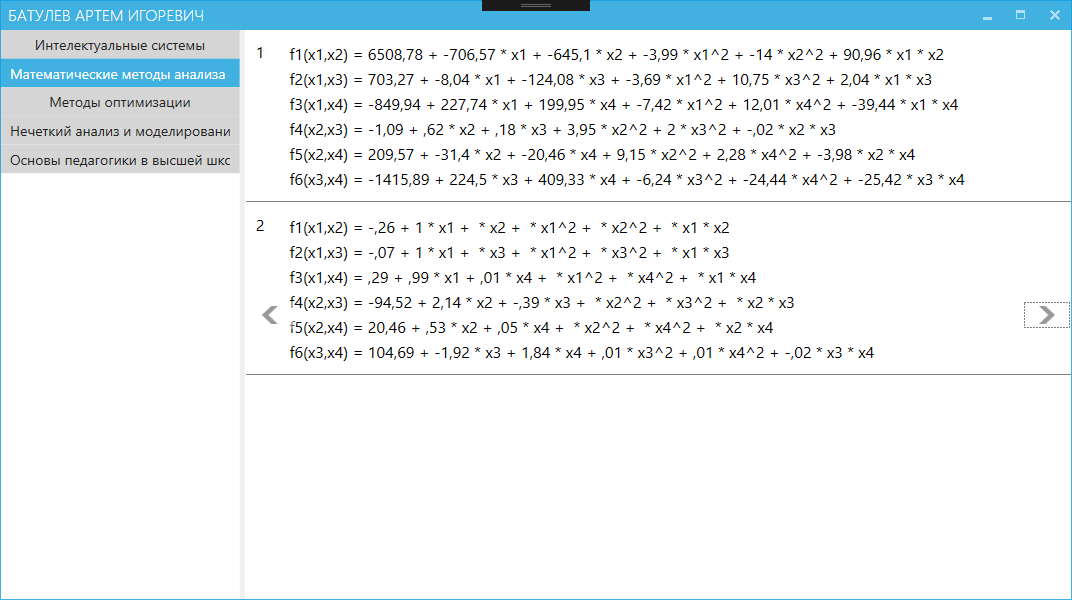
1. Ввод исходной таблицы данных (чтение данных из файла в формате .xls)

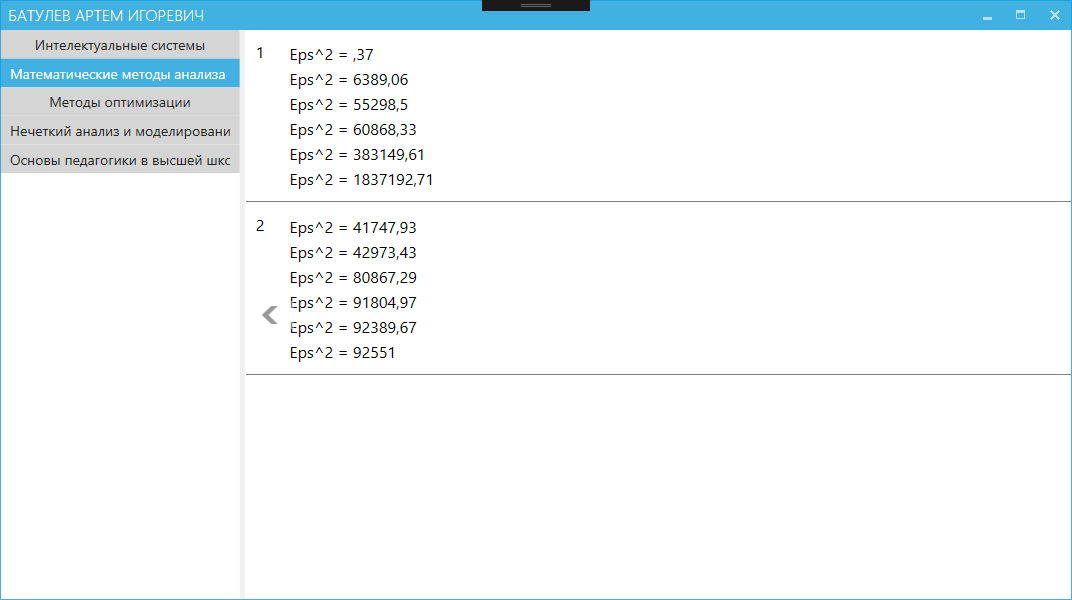


1. Разделяем выборку на обучающую и проверочную.



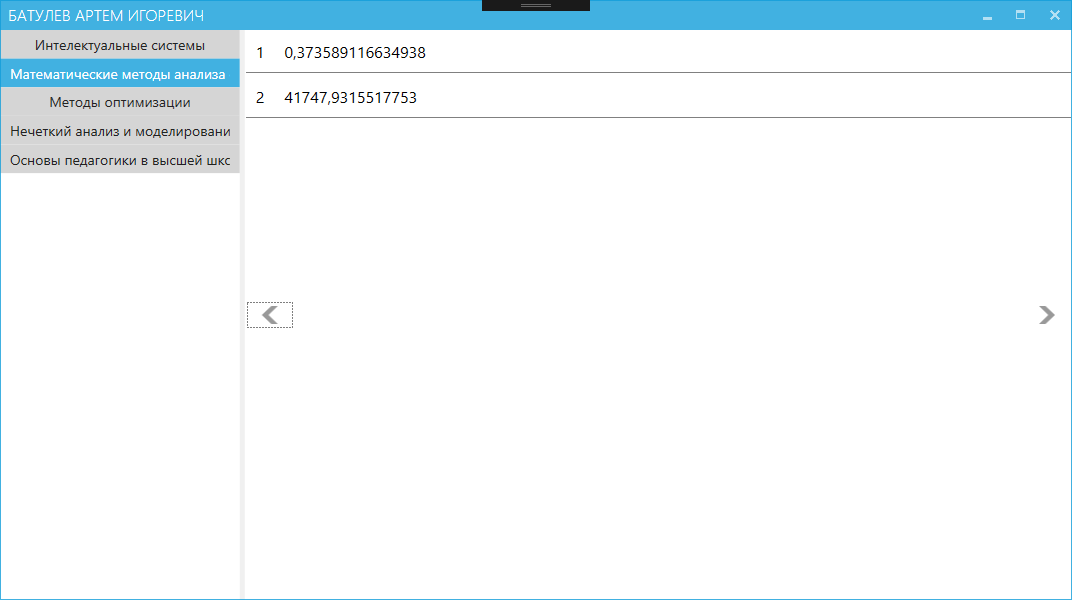
1. На обучающей выборке вычисляем коэффициенты регрессии
2. На проверочной выборке отбираем лучшие модели





1. Проверяем критерий ɛ2→min





Минимальное значение Eps^2 = 0.373589

1. Лучшие модели используются для расчета новых аргументов
2. Операция повторяется с п.2 до п.6 до тех пор пока не выполнится условие



Условие остановки выполнено. Лучшая модель на итерации 1.

Листинг алгоритма

private void MGUA()

{

// Критерий минимума на текущем шаге.

double epsilonSquaredCurrent = 0;

// Критерий минимума на предыдущем шаге.

double epsilonSquaredPrev = double.MaxValue;

// Текущая итерация.

int iteration = 1;

int[] sortingFuncsNumbers = new int[6];

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

// Проверяются величины критерия точности для наилучших моделей

// текущего и предыдущего ряда селекции.

while (true)

{

if (iteration != 1)

{

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

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

{

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

for (int j = 0; j < 4; j++)

{

buf[i][j] = MathixHelper.getNewValue(sortingFuncsNumbers[j], coefArr, \_dataTraining[i]);

}

buf[i][4] = \_dataTraining[i][4];

\_dataTraining[i] = buf[i];

}

}

#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

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

{

var x1 = \_dataTraining[i][0];

var x2 = \_dataTraining[i][1];

var x3 = \_dataTraining[i][2];

var x4 = \_dataTraining[i][3];

var y = \_dataTraining[i][4];

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

matr1[i][0] = x1;

matr1[i][1] = x2;

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

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

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

matr1[i][5] = y;

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

matr2[i][0] = x1;

matr2[i][1] = x3;

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

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

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

matr2[i][5] = y;

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

matr3[i][0] = x1;

matr3[i][1] = x4;

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

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

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

matr3[i][5] = y;

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

matr4[i][0] = x2;

matr4[i][1] = x3;

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

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

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

matr4[i][5] = y;

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

matr5[i][0] = x2;

matr5[i][1] = x4;

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

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

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

matr5[i][5] = y;

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

matr6[i][0] = x3;

matr6[i][1] = x4;

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

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

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

matr6[i][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[] coef1 = MathixHelper.Solve(design1);

double[] coef2 = MathixHelper.Solve(design2);

double[] coef3 = MathixHelper.Solve(design3);

double[] coef4 = MathixHelper.Solve(design4);

double[] coef5 = MathixHelper.Solve(design5);

double[] coef6 = MathixHelper.Solve(design6);

coefArr[0] = coef1;

coefArr[1] = coef2;

coefArr[2] = coef3;

coefArr[3] = coef4;

coefArr[4] = coef5;

coefArr[5] = coef6;

if (iteration != 1)

{

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

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

{

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

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

{

buf[i][j] = MathixHelper.getNewValue(sortingFuncsNumbers[j], coefArr, \_dataChecking[i]);

}

buf[i][\_columnIndexOutputVariable] = \_dataChecking[i][\_columnIndexOutputVariable];

\_dataChecking[i] = buf[i];

}

}

double epsSquare1 = 0;

double epsSquare2 = 0;

double epsSquare3 = 0;

double epsSquare4 = 0;

double epsSquare5 = 0;

double epsSquare6 = 0;

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

{

var x1 = \_dataChecking[i][0];

var x2 = \_dataChecking[i][1];

var x3 = \_dataChecking[i][2];

var x4 = \_dataChecking[i][3];

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

epsSquare1 += Math.Pow(y - MathixHelper.solveFunction(coef1, x1, x2), 2);

epsSquare2 += Math.Pow(y - MathixHelper.solveFunction(coef2, x1, x3), 2);

epsSquare3 += Math.Pow(y - MathixHelper.solveFunction(coef3, x1, x4), 2);

epsSquare4 += Math.Pow(y - MathixHelper.solveFunction(coef4, x2, x3), 2);

epsSquare5 += Math.Pow(y - MathixHelper.solveFunction(coef5, x2, x4), 2);

epsSquare6 += Math.Pow(y - MathixHelper.solveFunction(coef6, x3, x4), 2);

}

double[] epsArray = new double[\_coefficientArrayCount];

epsArray[0] = (double)epsSquare1 / \_checkingRowsCount;

epsArray[1] = (double)epsSquare2 / \_checkingRowsCount;

epsArray[2] = (double)epsSquare3 / \_checkingRowsCount;

epsArray[3] = (double)epsSquare4 / \_checkingRowsCount;

epsArray[4] = (double)epsSquare5 / \_checkingRowsCount;

epsArray[5] = (double)epsSquare6 / \_checkingRowsCount;

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

{

sortingFuncsNumbers[i] = i;

}

SortArray(epsArray, sortingFuncsNumbers);

epsilonSquaredPrev = epsilonSquaredCurrent;

epsilonSquaredCurrent = epsArray[0];

FillResult(iteration, epsilonSquaredCurrent, coefArr, epsArray);

if (epsilonSquaredCurrent > epsilonSquaredPrev && iteration > 1)

{

break;

}

iteration++;

}

}