Требуется решить задачу линейного программирования для целевой функции вида: 

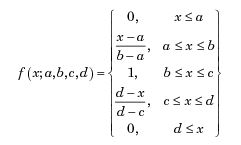
При ограничениях:

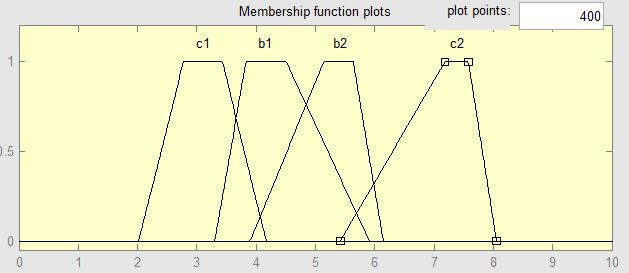




Где коэффициенты  являются нечеткими переменными и выбираются в соответствии ­­─ целые числа.

Вариант 3

Trapmf 



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Переменная |  |  |  |  |
| Параметры | [2 2.8 3.5 4.2] | [5.44 7.2 7.6 8.07] | [3.3 3.83 4.51 5.911] | [3.901 5.141 5.641 6.141] |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Переменная |  |  |  |  |
| Параметры | 2 | 1 | 1 | 2 |

Тестирование программы.

На рисунке 1 изображена результирующая таблица со всеми коэффициентами, их значениями и результатом функции.

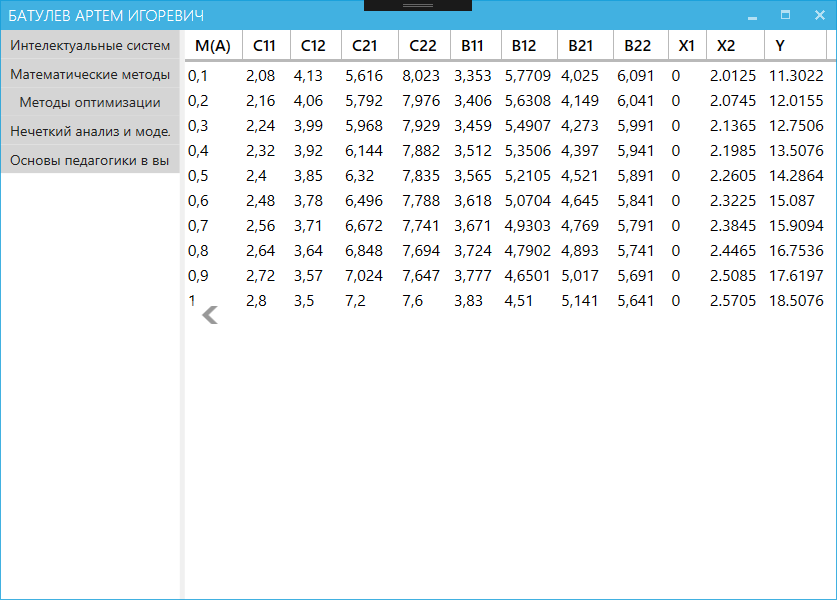


Рис 1.

На рисунке 2 изображены результаты выполнения программы.

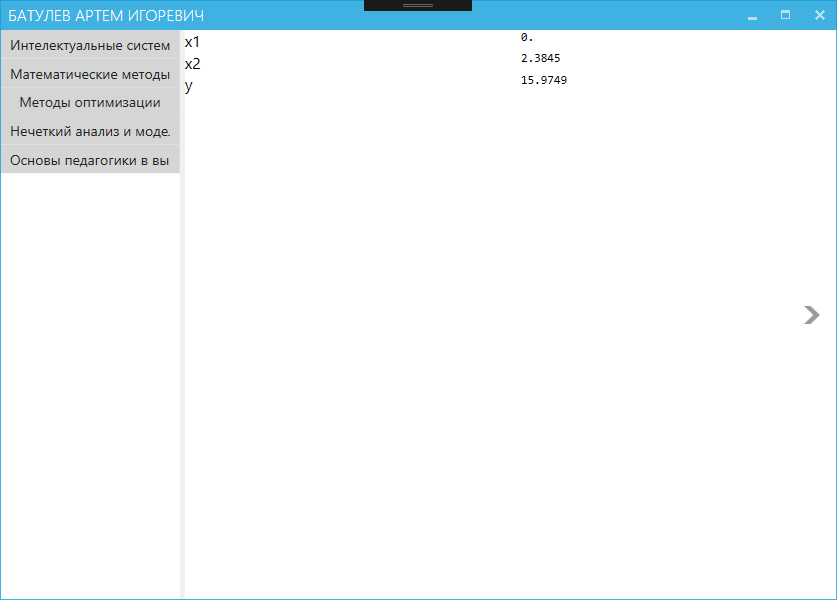


Рис 2.

Листинг программы приведен в приложении А.

Приложение А

Листинг программы

namespace OptimizationMethods.ViewModels.Laba3

{

public class Lab3ViewModel : BaseJob

{

private readonly CultureInfo \_culture = CultureInfo.InvariantCulture;

private string \_x1;

private string \_x2;

private string \_y;

private DataView \_result;

public Lab3ViewModel(MathKernel mathKernel)

: base(mathKernel)

{

Go();

}

public DataView Result

{

get { return \_result; }

set { SetProperty(ref \_result, value); }

}

private void Go()

{

var dataTable = new DataTable();

dataTable.Columns.Add(new DataColumn("M(a)"));

dataTable.Columns.Add(new DataColumn("C11"));

dataTable.Columns.Add(new DataColumn("C12"));

dataTable.Columns.Add(new DataColumn("C21"));

dataTable.Columns.Add(new DataColumn("C22"));

dataTable.Columns.Add(new DataColumn("b11"));

dataTable.Columns.Add(new DataColumn("b12"));

dataTable.Columns.Add(new DataColumn("b21"));

dataTable.Columns.Add(new DataColumn("b22"));

dataTable.Columns.Add(new DataColumn("x1"));

dataTable.Columns.Add(new DataColumn("x2"));

dataTable.Columns.Add(new DataColumn("y"));

// Задаем уравнения нечетких переменных.

// c1.

Compute("a1=2");

Compute("b1=2.8");

Compute("c1=3.5");

Compute("d1=4.2");

Compute("fc1=If[ x <= a1, 0, If[ a1 <= x <= b1, ((x - a1)/(b1 - a1)), If[ b1 <= x <= c1, 1, If[ c1 <= x <= d1, ((d1 - x)/(d1 - c1)), If[ d1 <= x, 0]]]]]");

// c2.

Compute("a2=5.44");

Compute("b2=7.2");

Compute("c2=7.6");

Compute("d2=8.07");

Compute("fc2=If[ x <= a2, 0, If[ a2 <= x <= b2, ((x - a2)/(b2 - a2)), If[ b2 <= x <= c2, 1, If[ c2 <= x <= d2, ((d2 - x)/(d2 - c2)), If[ d2 <= x, 0]]]]]");

// b1.

Compute("a3=3.3");

Compute("b3=3.83");

Compute("c3=4.51");

Compute("d3=5.911");

Compute("fb1=If[ x <= a3, 0, If[ a3 <= x <= b3, ((x - a3)/(b3 - a3)), If[ b3 <= x <= c3, 1, If[ c3 <= x <= d3, ((d3 - x)/(d3 - c3)), If[ d3 <= x, 0]]]]]");

// b2.

Compute("a4=3.901");

Compute("b4=5.141");

Compute("c4=5.641");

Compute("d4=6.141");

Compute("fb2=If[ x <= a4, 0, If[ a4 <= x <= b4, ((x - a4)/(b4 - a4)), If[ b4 <= x <= c4, 1, If[ c4 <= x <= d4, ((d4 - x)/(d4 - c4)), If[ d4 <= x, 0]]]]]");

// Для каждого уровня достоверности.

for (int i = 0; i < 10; i++)

{

var row = dataTable.NewRow();

dataTable.Rows.Add(row);

var a = 0.1 \* (i + 1);

string aS = a.ToString(CultureInfo.InvariantCulture);

Compute($"y={aS}");

row[0] = a;

//Решить уравнение при определенном уровне

Compute("r = x/.Solve[ fc1 == y, {x} ]");

var c11 = double.Parse(a == 1 ? Compute("Reduce[fc1 == y, {x}][[2]][[1]]") : Compute("r[[1]]"), \_culture);

var c12 = double.Parse(a == 1 ? Compute("Reduce[fc1 == y, {x}][[2]][[5]]") : Compute("r[[2]]"), \_culture);

if (c11 > c12)

{

double b = c12;

c12 = c11;

c11 = b;

}

row[1] = c11;

row[2] = c12;

Compute("r = x/.Solve[ fc2 == y, {x} ]");

var c21 = double.Parse(a == 1 ? Compute("Reduce[fc2 == y, {x}][[1]][[1]]") : Compute("r[[1]]"), \_culture);

var c22 = double.Parse(a == 1 ? Compute("Reduce[fc2 == y, {x}][[1]][[5]]") : Compute("r[[2]]"), \_culture);

if (c21 > c22)

{

double b = c22;

c22 = c21;

c21 = b;

}

row[3] = c21;

row[4] = c22;

Compute("r = x/.Solve[ fb1 == y, {x} ]");

var b1 = double.Parse(a == 1 ? Compute("Reduce[fb1 == y, {x}][[2]][[1]]") : Compute("r[[1]]"), \_culture);

row[5] = b1;

row[6] = double.Parse(a == 1 ? Compute("Reduce[fb1 == y, {x}][[2]][[5]]") : Compute("r[[2]]"), \_culture);

Compute("r = x/.Solve[fb2==y,x]");

var b2 = double.Parse(a == 1 ? Compute("Reduce[fb2 == y, {x}][[2]][[1]]") : Compute("r[[1]]"), \_culture);

row[7] = b2;

row[8] = double.Parse(a == 1 ? Compute("Reduce[fb2 == y, {x}][[2]][[5]]") : Compute("r[[2]]"), \_culture);

CalculateLinearProgramming(c11, c12, c21, c22, b1, b2, row);

Compute($"Total[{row[9]}] / 5.5");

}

Defuzzification(dataTable);

Result = dataTable.DefaultView;

}

/// <summary>

/// Решить интервальную задачу (лаба 2) для полученных значениях.

/// </summary>

/// <param name="C11"></param>

/// <param name="C12"></param>

/// <param name="C21"></param>

/// <param name="C22"></param>

/// <param name="b1"></param>

/// <param name="b2"></param>

/// <param name="iteration"></param>

private void CalculateLinearProgramming(double C11, double C12, double C21, double C22, double b1, double b2, DataRow row)

{

double[,] C = new double[5, 2];

double[] z = new double[5];

double[,] x = new double[2, 5];

bool[] right = new bool[5];

C[0, 0] = C11;

C[0, 1] = C21;

C[1, 0] = C12;

C[1, 1] = C22;

double min1 = C[0, 0];

double min2 = C[0, 1];

double max1 = C[1, 0];

double max2 = C[1, 1];

C[2, 0] = min1;

C[2, 1] = max2;

C[3, 0] = max1;

C[3, 1] = min2;

C[4, 0] = (min1 + max1) / 2;

C[4, 1] = (min2 + max2) / 2;

for (int i = 0; i < 5; i++)

{

Compute($"solve = Maximize[ {C[i, 0].ToString(\_culture)} \* x1 + {C[i, 1].ToString(\_culture)} \* x2, {{ 2 \* x1 + 1 \* x2 <= {b1.ToString(\_culture)}, 1 \* x1 + 2 \* x2 <= {b2.ToString(\_culture)}, x1>=0,x2>=0 }}, {{x1, x2}} ]");

Func<string, string> f = (t) => t.Last() == '.' ? t.Remove(t.Length - 1) : t;

x[0, i] = double.Parse(f(Compute("N[First[{x1, x2} /.Last[solve]]]")), \_culture);

x[1, i] = double.Parse(f(Compute("N[Last[{x1, x2} /.Last[solve]]]")), \_culture);

z[i] = double.Parse(f(Compute("N[First[solve]]")), \_culture);

}

for (int i = 0; i < 5; i++)

for (int j = 0; j < 5; j++)

if ((x[0, i] == x[0, j]) && (x[1, i] == x[1, j]) && (i != j))

right[i] = true;

int n = 0;

for (int i = 0; i < 5; i++)

if (right[i])

n++;

double[,] solve = new double[n, 3];

n = 0;

for (int i = 0; i < 5; i++)

if (right[i])

{

solve[n, 0] = z[i];

solve[n, 1] = x[0, i];

solve[n, 2] = x[1, i];

n++;

}

if (n != 0)

{

double max = solve[0, 0];

n = 0;

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

if (max < solve[i, 0])

{

max = solve[i, 0];

n = i;

}

row[9] = solve[n, 1].ToString(\_culture);

row[10] = solve[n, 2].ToString(\_culture);

row[11] = max.ToString(\_culture);

}

else

{

Compute("solve=Minimize[Abs[" + z[0].ToString(\_culture) + "-(" + C[0, 0].ToString(\_culture) + "\*x1+" + C[0, 1].ToString(\_culture) + "\*x2)]+" +

"Abs[" + z[1].ToString(\_culture) + "-(" + C[1, 0].ToString(\_culture) + "\*x1+" + C[1, 1].ToString(\_culture) + "\*x2)]+" +

"Abs[" + z[2].ToString(\_culture) + "-(" + C[2, 0].ToString(\_culture) + "\*x1+" + C[2, 1].ToString(\_culture) + "\*x2)]+" +

"Abs[" + z[3].ToString(\_culture) + "-(" + C[3, 0].ToString(\_culture) + "\*x1+" + C[3, 1].ToString(\_culture) + "\*x2)]+" +

"Abs[" + z[4].ToString(\_culture) + "-(" + C[4, 0].ToString(\_culture) + "\*x1+" + C[4, 1].ToString(\_culture) + "\*x2)]," +

"{" + "2\*x1+" + "1\*x2<=" + b1.ToString(\_culture) + ","

+ "1\*x1+" + "2\*x2<=" + b2.ToString(\_culture) + ",x1>=0,x2>=0},{x1,x2}]");

row[11] = double.Parse(Compute("N[First[solve]]"), \_culture);

row[9] = double.Parse(Compute("First[{x1, x2} /.Last[solve]]"), \_culture);

row[10] = double.Parse(Compute("Last[{x1, x2} /.Last[solve]]"), \_culture);

}

}

/// <summary>

/// Дефазификация решений.

/// </summary>

private void Defuzzification(DataTable dataView)

{

X1 = Compute("Total[" +

"{0.1 \* " + dataView.Rows[0][9] + ", " +

"0.2 \* " + dataView.Rows[1][9] + ", " +

"0.3 \* " + dataView.Rows[2][9] + ", " +

"0.4 \* " + dataView.Rows[3][9] + ", " +

"0.5 \* " + dataView.Rows[4][9] + ", " +

"0.6 \* " + dataView.Rows[5][9] + ", " +

"0.7 \* " + dataView.Rows[6][9] + ", " +

"0.8 \* " + dataView.Rows[7][9] + ", " +

"0.9 \* " + dataView.Rows[8][9] + ", " +

dataView.Rows[9][9] + "}] / 5.5");

X2 = Compute("Total[{" +

"0.1 \* " + dataView.Rows[0][10] + ", " +

"0.2 \* " + dataView.Rows[1][10] + ", " +

"0.3 \* " + dataView.Rows[2][10] + ", " +

"0.4 \* " + dataView.Rows[3][10] + ", " +

"0.5 \* " + dataView.Rows[4][10] + ", " +

"0.6 \* " + dataView.Rows[5][10] + ", " +

"0.7 \* " + dataView.Rows[6][10] + ", " +

"0.8 \* " + dataView.Rows[7][10] + ", " +

"0.9 \* " + dataView.Rows[8][10] + ", " +

dataView.Rows[9][10] + "}] / 5.5");

Y = Compute("Total[{" +

"0.1 \* " + dataView.Rows[0][11] + ", " +

"0.2 \* " + dataView.Rows[1][11] + ", " +

"0.3 \* " + dataView.Rows[2][11] + ", " +

"0.4 \* " + dataView.Rows[3][11] + ", " +

"0.5 \* " + dataView.Rows[4][11] + ", " +

"0.6 \* " + dataView.Rows[5][11] + ", " +

"0.7 \* " + dataView.Rows[6][11] + ", " +

"0.8 \* " + dataView.Rows[7][11] + ", " +

"0.9 \* " + dataView.Rows[8][11] + ", " +

dataView.Rows[9][11] + "}] / 5.5");

}

public string X1

{

get { return \_x1; }

set { SetProperty(ref \_x1, value); }

}

public string X2

{

get { return \_x2; }

set { SetProperty(ref \_x2, value); }

}

public string Y

{

get { return \_y; }

set { SetProperty(ref \_y, value); }

}

}

}