МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»



Кафедра СМАРТ технологий

Лабораторная работа №1

«Использование графических возможностей C# для визуализации данных стохастических процессов»

По дисциплине «Технологии визуализации данных систем управления»

Группа 201-325

Студент Холодилов И.В.

Дата 11.05.2023

Преподаватель Идиатулов Т.Т.

2023

# Цель работы

Разработать приложение по генерации стохастических данных с заданным профилем распределения и визуализации распределения случайных величин.

# Задачи

* Подготовить приложение на языке C# для статистической обработки и визуализации собранных наборов данных

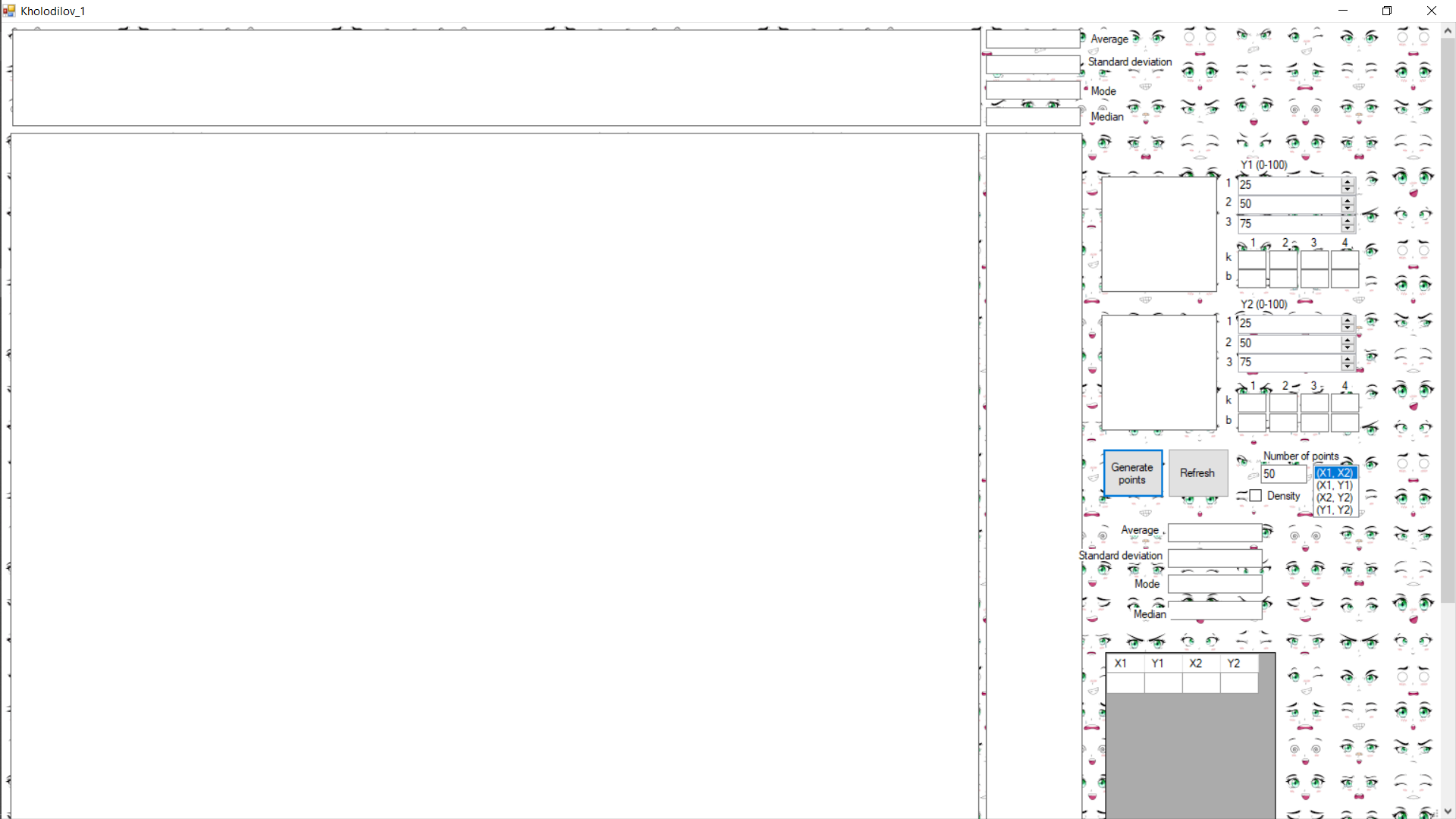


Рисунок 1 - Form

* Реализовать генерацию заданного (через текстовое поле) количества случайных точек (X1, X2), где X1 и X2 – равномерно распределенные случайные величины на диапазоне [0 ÷ 1]. Подготовить функционал для настройки профиля преобразования (пересчета) двух случайных величин (Y1, Y2) из равномерно распределенных случайных величин (X1, X2). Реализовать отрисовку наборов данных в виде облака точек, с возможностью выбора пар параметров, используемых как координаты точек.

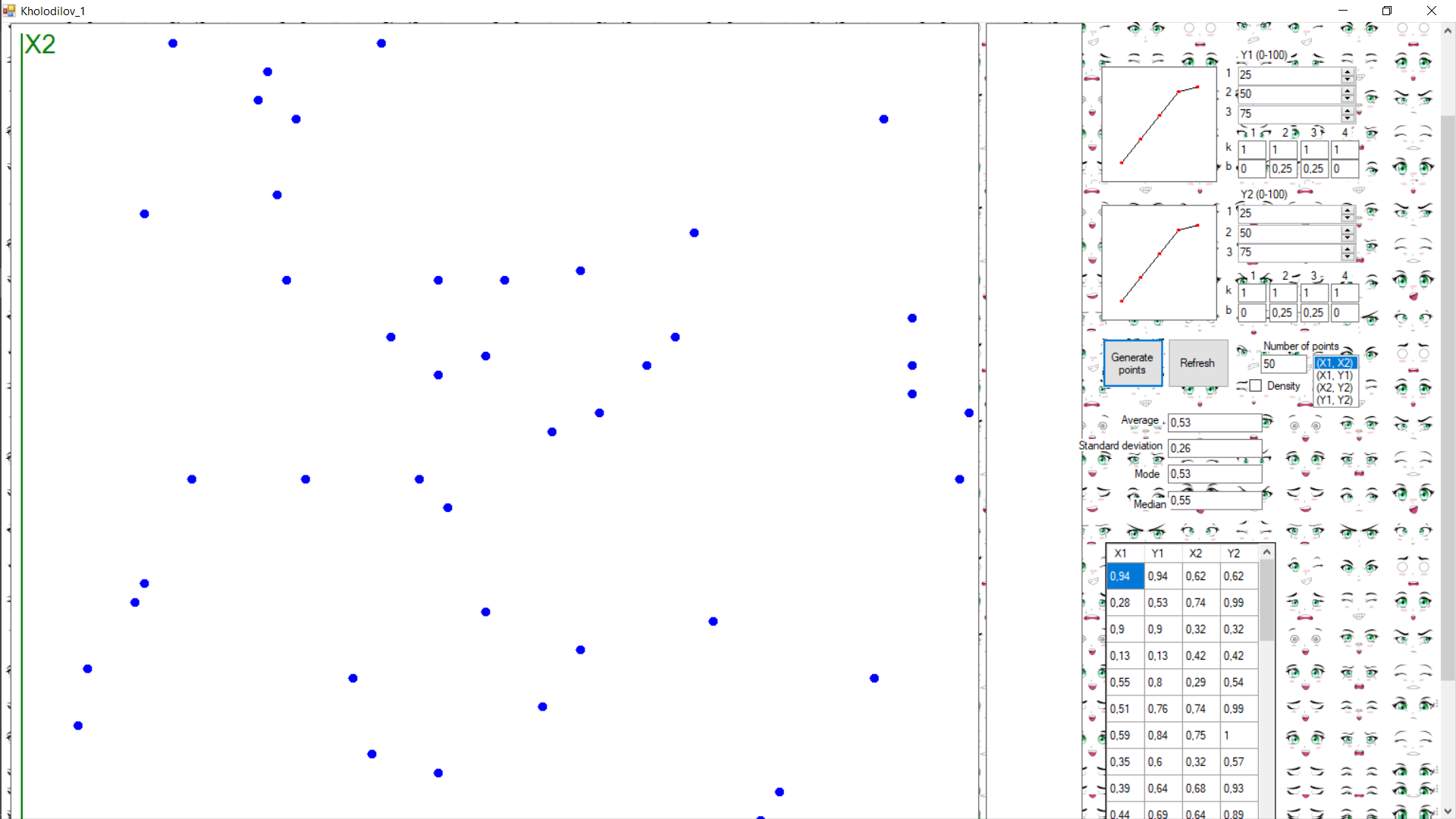


Рисунок 2 - Генерация и отображение данных в виде изображения с соответствующими осями

* Реализовать отображение профиля преобразования как кусочно-линейных функций (по пяти точкам – первая и последняя привязаны к границам диапазона генерации равномерно-распределенных случайных величин).

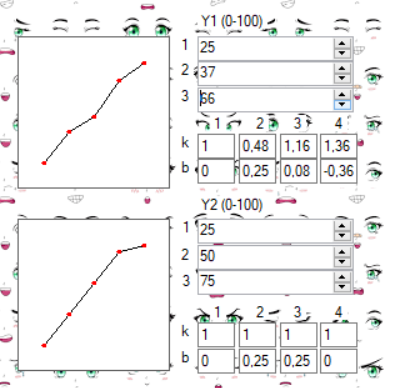


Рисунок 3 - Генерация вторых значений данных с помощию numericupdown

* Должна существовать возможность отобразить точки (X1, X2), (X1, Y1), (X2, Y2), (Y1, Y2).



Рисунок 4 - Выбос зависимости осей

* Реализовать расчёт плотности распределения случайных точек и выполнить фоновую окраску области отрисовки случайных точек. При подсчете плотности разделить диапазон отображения по каждой оси на 10 интервалов.

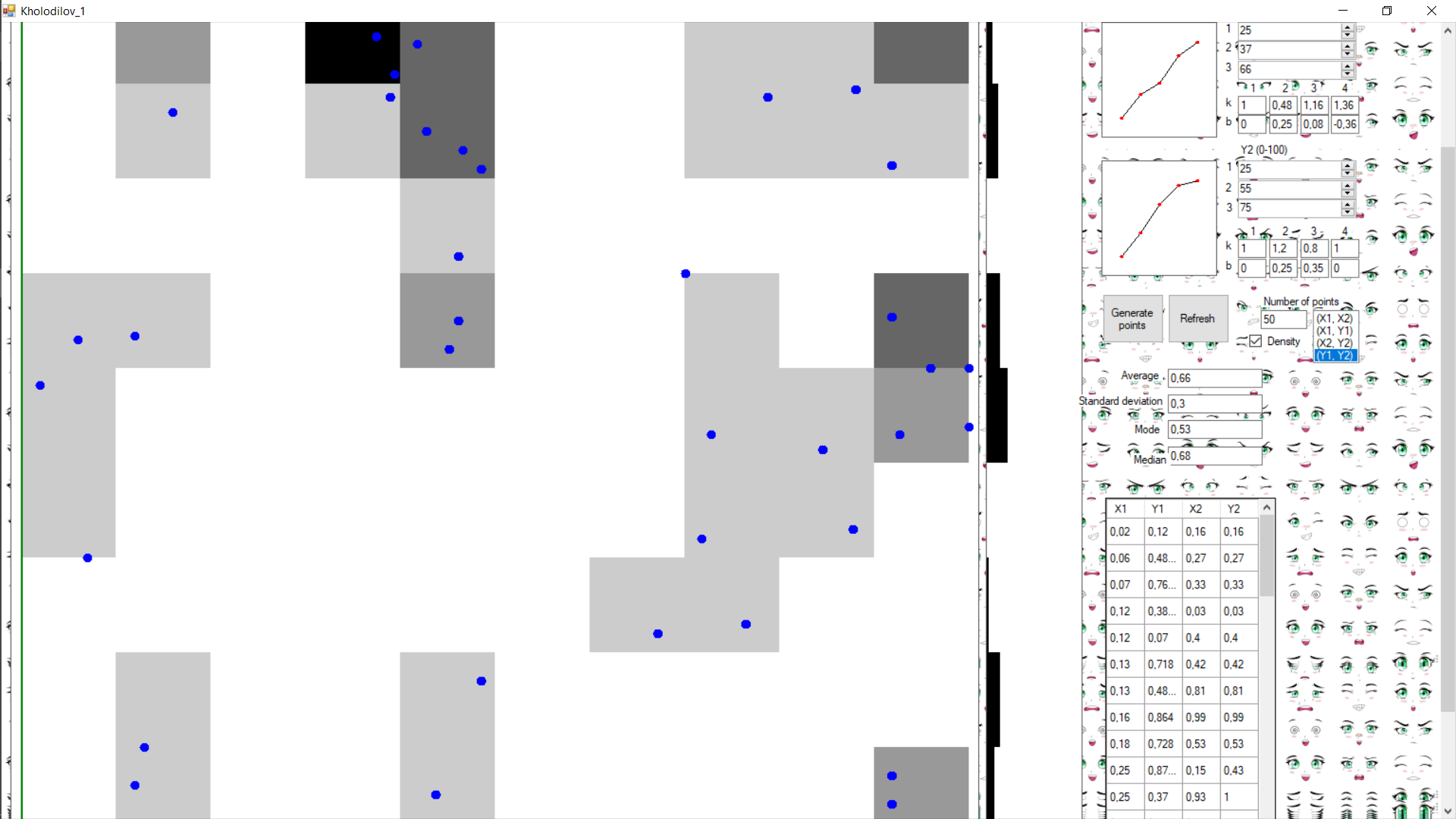


Рисунок 5 - Создание гистограммы по выбранным данным

* Добавлен в приложении расчёт статистических данных (описательной статистики) полученного распределения (среднее, средне-квадратическое отклонение, мода, медиана), а также отображения гистограммы распределения каждой из случайных величин. Расположить оси гистограмм вдоль соответствующих осей на диаграмме облака точек.

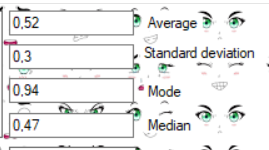


Рисунок 6 - Подсчет всех мат. функций

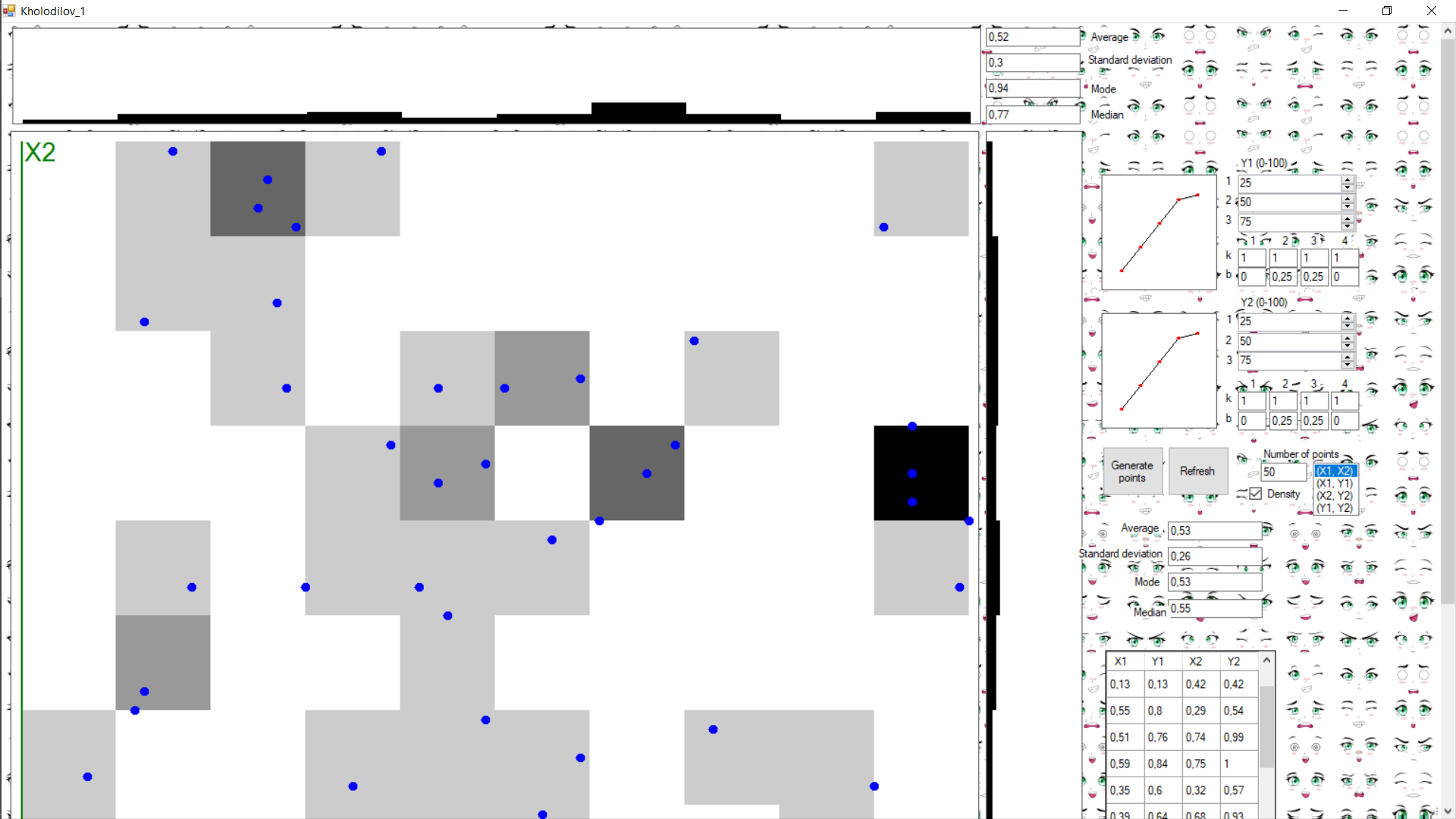


Рисунок 7 - Итоговый вид проекта

**Вывод**

В ходе создания программы было написанно ПО для визуализации собранных наборов пространственных данных методом проекции на плоскость.

# Приложение A

Листинг A-1 – программный код:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Drawing;  using System.Windows.Forms;  //using System.Windows.Media;  namespace Lab\_1  {  public partial class Kholodilov\_1 : Form  {  Random random = new Random();  List<PointF> points\_x = new List<PointF>();  List<PointF> points\_y = new List<PointF>();  bool start\_work = false;  bool is\_y1\_param\_change = false;  bool is\_y2\_param\_change = false;  bool is\_axes\_change = true;  bool is\_density\_change = false;  int[,] density = new int[10, 10];  int[] density\_up = new int[10];  int[] density\_right = new int[10];  float summ\_up = 0;  float summ\_right = 0;  public Kholodilov\_1()  {  InitializeComponent();  timer1.Enabled = true;  dataGridView1.RowHeadersVisible = false;  dataGridView1.ColumnCount = 4;  int size\_grid = 40;  dataGridView1.Columns[0].Width = size\_grid;  dataGridView1.Columns[1].Width = size\_grid;  dataGridView1.Columns[2].Width = size\_grid;  dataGridView1.Columns[3].Width = size\_grid;  dataGridView1.Width = size\_grid \* 4 + 20;  listBox1.SelectedIndex = 0;  X1Y1\_box.Image = new Bitmap(120, 120);  X2Y2\_box.Image = new Bitmap(120, 120);  Main\_box.Image = new Bitmap(1020, 1020);  Up\_box.Image = new Bitmap(1020, 100);  Right\_box.Image = new Bitmap(100, 1020);  }  private void Form1\_Load\_1(object sender, EventArgs e)  {  }  public float check\_max\_min\_point (float d)  {  if (d <= 0) return 0.0f;  else if (d >= 1) return 1.0f;  else return d;  }  private void Form1\_FormClosed\_1(object sender, FormClosedEventArgs e)  {  timer1.Stop();  }  public int check\_id\_density (int d)  {  if (d < 0) return 0;  else if (d > 9) return 9;  else return d;  }  public void draw\_main\_img ()  {  density = new int[10, 10] { { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } };  density\_up = new int[10] { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };  density\_right = new int[10] { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };  summ\_right = 0;  summ\_up = 0;  Graphics graphics = Graphics.FromImage(Main\_box.Image);  Graphics graphics\_up = Graphics.FromImage(Up\_box.Image);  Graphics graphics\_righ = Graphics.FromImage(Right\_box.Image);  graphics.FillRectangle(Brushes.White, new Rectangle(0, 0, Main\_box.Width, Main\_box.Height));  graphics\_up.FillRectangle(Brushes.White, new Rectangle(0, 0, Up\_box.Width, Up\_box.Height));  graphics\_righ.FillRectangle(Brushes.White, new Rectangle(0, 0, Right\_box.Width, Right\_box.Height));  if (is\_density.Checked)  {  int max = 0;  for (int i = 0; i < points\_x.Count; i++)  {  switch (listBox1.SelectedIndex)  {  case 1:  density[check\_id\_density((int)Math.Ceiling(points\_x[i].X \* 10 - 1)), check\_id\_density((int)Math.Ceiling(points\_y[i].X \* 10 - 1))]++;  if (density[check\_id\_density((int)Math.Ceiling(points\_x[i].X \* 10 - 1)), check\_id\_density((int)Math.Ceiling(points\_y[i].X \* 10 - 1))] > max) max = density[check\_id\_density((int)Math.Ceiling(points\_x[i].X \* 10 - 1)), check\_id\_density((int)Math.Ceiling(points\_y[i].X \* 10 - 1))];  break;  case 2:  density[check\_id\_density((int)Math.Ceiling(points\_x[i].Y \* 10 - 1)), check\_id\_density((int)Math.Ceiling(points\_y[i].Y \* 10 - 1))]++;  if (density[check\_id\_density((int)Math.Ceiling(points\_x[i].Y \* 10 - 1)), check\_id\_density((int)Math.Ceiling(points\_y[i].Y \* 10 - 1))] > max) max = density[check\_id\_density((int)Math.Ceiling(points\_x[i].Y \* 10 - 1)), check\_id\_density((int)Math.Ceiling(points\_y[i].Y \* 10 - 1))];  break;  case 3:  density[check\_id\_density((int)Math.Ceiling(points\_y[i].X \* 10 - 1)), check\_id\_density((int)Math.Ceiling(points\_y[i].Y \* 10 - 1))]++;  if (density[check\_id\_density((int)Math.Ceiling(points\_y[i].X \* 10 - 1)), check\_id\_density((int)Math.Ceiling(points\_y[i].Y \* 10 - 1))] > max) max = density[check\_id\_density((int)Math.Ceiling(points\_y[i].X \* 10 - 1)), check\_id\_density((int)Math.Ceiling(points\_y[i].Y \* 10 - 1))];  break;  default:  density[check\_id\_density((int)Math.Ceiling(points\_x[i].X \* 10 - 1)), check\_id\_density((int)Math.Ceiling(points\_x[i].Y \* 10 - 1))]++;  if (density[check\_id\_density((int)Math.Ceiling(points\_x[i].X \* 10 - 1)), check\_id\_density((int)Math.Ceiling(points\_x[i].Y \* 10 - 1))] > max) max = density[check\_id\_density((int)Math.Ceiling(points\_x[i].X \* 10 - 1)), check\_id\_density((int)Math.Ceiling(points\_x[i].Y \* 10 - 1))];  break;  }  }  for (int i = 0; i < 10; i++)  {  for (int j = 0; j < 10; j++)  {  System.Drawing.Brush brush = new SolidBrush(System.Drawing.Color.FromArgb((int)(density[i,j]\*255/max), 0, 0, 0));  graphics.FillRectangle(brush, 10 + 100 \* i, 10 + 900 - (100 \* j), 100, 100);  density\_up[j] += density[i, j];  density\_right[i] += density[i, j];  }  }  for (int i = 0; i < 10; i++)  {  graphics\_up.FillRectangle(Brushes.Black, 10 + 100 \* i, 100 - (int)((density\_up[i] \* 100) / Int32.Parse(num\_point.Text)), 100, (int)((density\_up[i] \* 100) / Int32.Parse(num\_point.Text)));  graphics\_righ.FillRectangle(Brushes.Black, 0, 10 + 100 \* i, (int)((density\_right[i] \* 100) / Int32.Parse(num\_point.Text)), 100);  //graphics\_up.FillRectangle(Brushes.Black, 10 + 100 \* i, 100 - ((density\_up[i] \* 100) / Int32.Parse(num\_point.Text)), 100, 10);  //MessageBox.Show((100 - (density\_up[i] / Int32.Parse(num\_point.Text) \* 100)).ToString());  }  }  Font drawFont = new Font("Arial", 20);  StringFormat drawFormat = new StringFormat();  SolidBrush drawBrush = new SolidBrush(Color.Black);  graphics.DrawString(listBox1.Items[listBox1.SelectedIndex].ToString().Substring(5, 2), drawFont, Brushes.Green, 10, 5, drawFormat);  graphics.DrawString(listBox1.Items[listBox1.SelectedIndex].ToString().Substring(1, 2), drawFont, Brushes.Red, Main\_box.Width - 47, Main\_box.Height - 40, drawFormat);  graphics.FillRectangle(Brushes.Green, 10, 10, 2, Main\_box.Height - 20);  graphics.FillRectangle(Brushes.Red, 10, Main\_box.Height - 10, Main\_box.Width - 20, 2);  for (int i = 0; i < points\_x.Count; i++)  {  switch (listBox1.SelectedIndex)  {  case 1:  graphics.FillEllipse(Brushes.Blue, 10 + points\_x[i].X \* 1000 - 5, 10 + 1000 - points\_y[i].X \* 1000 - 5, 5 + 5, 5 + 5);  summ\_right += points\_y[i].X;  summ\_up += points\_x[i].X;  break;  case 2:  graphics.FillEllipse(Brushes.Blue, 10 + points\_x[i].Y \* 1000 - 5, 10 + 1000 - points\_y[i].Y \* 1000 - 5, 5 + 5, 5 + 5);  summ\_right += points\_y[i].Y;  summ\_up += points\_x[i].Y;  break;  case 3:  graphics.FillEllipse(Brushes.Blue, 10 + points\_y[i].X \* 1000 - 5, 10 + 1000 - points\_y[i].Y \* 1000 - 5, 5 + 5, 5 + 5);  summ\_right += points\_y[i].Y;  summ\_up += points\_y[i].X;  break;  default:  graphics.FillEllipse(Brushes.Blue, 10 + points\_x[i].X \* 1000 - 5, 10 + 1000 - points\_x[i].Y \* 1000 - 5, 5 + 5, 5 + 5);  summ\_right += points\_x[i].Y;  summ\_up += points\_x[i].X;  break;  }  }  if (is\_y1\_param\_change || is\_y2\_param\_change || is\_axes\_change)  {  float ave\_right = ((float)(summ\_right / points\_x.Count));  float ave\_up = ((float)(summ\_up / points\_x.Count));  double st\_deviation\_up = 0;  double st\_deviation\_right = 0;  for (int i = 0; i < points\_x.Count; i++)  {  switch (listBox1.SelectedIndex)  {  case 1:  //summ\_right += points\_y[i].X;  //summ\_up += points\_x[i].X;  st\_deviation\_up += Math.Pow(points\_x[i].X - ave\_up, 2);  st\_deviation\_right += Math.Pow(points\_y[i].X - ave\_right, 2);  break;  case 2:  //summ\_right += points\_y[i].Y;  //summ\_up += points\_x[i].Y;  st\_deviation\_up += Math.Pow(points\_x[i].Y - ave\_up, 2);  st\_deviation\_right += Math.Pow(points\_y[i].Y - ave\_right, 2);  break;  case 3:  //summ\_right += points\_y[i].Y;  //summ\_up += points\_y[i].X;  st\_deviation\_up += Math.Pow(points\_y[i].X - ave\_up, 2);  st\_deviation\_right += Math.Pow(points\_y[i].Y - ave\_right, 2);  break;  default:  //summ\_right += points\_x[i].Y;  //summ\_up += points\_x[i].X;  st\_deviation\_up += Math.Pow(points\_x[i].X - ave\_up, 2);  st\_deviation\_right += Math.Pow(points\_x[i].Y - ave\_right, 2);  break;  }  }  average\_right.Text = Math.Round(ave\_right, 2).ToString();  average\_up.Text = Math.Round(ave\_up, 2).ToString();  standard\_deviation\_right.Text = Math.Round(Math.Sqrt((float)(st\_deviation\_right / points\_x.Count)), 2).ToString();  standard\_deviation\_up.Text = Math.Round(Math.Sqrt((float)(st\_deviation\_up / points\_x.Count)), 2).ToString();    // Mode  int m\_right = 0;  int m\_up = 0;  float m\_data\_right = -1;  float m\_data\_up = -1;  float m\_last\_right = -1;  float m\_last\_up = -1;  switch (listBox1.SelectedIndex)  {  case 1:  //summ\_right += points\_y[i].X;  //summ\_up += points\_x[i].X;  points\_y = Sort\_list(points\_y, true);  points\_x = Sort\_list(points\_x, true);  for (int i = 0; i < points\_x.Count; i++)  {  if (points\_y[i].X == m\_last\_right)  {  m\_right++;  }  else  {  if (m\_data\_right < m\_right)  {  mode\_right.Text = m\_last\_right.ToString();  m\_data\_right = m\_right;  }  m\_right = 0;  }  m\_last\_right = points\_y[i].X;  if (points\_x[i].X == m\_last\_up)  {  m\_up++;  }  else  {  if (m\_data\_up < m\_up)  {  mode\_up.Text = m\_last\_up.ToString();  m\_data\_up = m\_up;  }  m\_up = 0;  }  m\_last\_up = points\_x[i].X;  }  break;  case 2:  //summ\_right += points\_y[i].Y;  //summ\_up += points\_x[i].Y;  points\_y = Sort\_list(points\_y, false);  points\_x = Sort\_list(points\_x, false);  for (int i = 0; i < points\_x.Count; i++)  {  if (points\_y[i].Y == m\_last\_right)  {  m\_right++;  }  else  {  if (m\_data\_right < m\_right)  {  mode\_right.Text = m\_last\_right.ToString();  m\_data\_right = m\_right;  }  m\_right = 0;  }  m\_last\_right = points\_y[i].Y;  if (points\_x[i].Y == m\_last\_up)  {  m\_up++;  }  else  {  if (m\_data\_up < m\_up)  {  mode\_up.Text = m\_last\_up.ToString();  m\_data\_up = m\_up;  }  m\_up = 0;  }  m\_last\_up = points\_x[i].Y;  }  break;  case 3:  //summ\_right += points\_y[i].Y;  //summ\_up += points\_y[i].X;  points\_y = Sort\_list(points\_y, false);  for (int i = 0; i < points\_x.Count; i++)  {  if (points\_y[i].Y == m\_last\_right)  {  m\_right++;  }  else  {  if (m\_data\_right < m\_right)  {  mode\_right.Text = m\_last\_right.ToString();  m\_data\_right = m\_right;  }  m\_right = 0;  }  m\_last\_right = points\_y[i].Y;  }  points\_y = Sort\_list(points\_y, true);  for (int i = 0; i < points\_x.Count; i++)  {  if (points\_y[i].X == m\_last\_up)  {  m\_up++;  }  else  {  if (m\_data\_up < m\_up)  {  mode\_up.Text = m\_last\_up.ToString();  m\_data\_up = m\_up;  }  m\_up = 0;  }  m\_last\_up = points\_y[i].X;  }  break;  default:  //summ\_right += points\_x[i].Y;  //summ\_up += points\_x[i].X;  points\_x = Sort\_list(points\_x, false);  for (int i = 0; i < points\_x.Count; i++)  {  if (points\_x[i].Y == m\_last\_right)  {  m\_right++;  }  else  {  if (m\_data\_right < m\_right)  {  mode\_right.Text = m\_last\_right.ToString();  m\_data\_right = m\_right;  }  m\_right = 0;  }  m\_last\_right = points\_x[i].Y;  }  points\_x = Sort\_list(points\_x, true);  for (int i = 0; i < points\_x.Count; i++)  {  if (points\_x[i].X == m\_last\_up)  {  m\_up++;  }  else  {  if (m\_data\_up < m\_up)  {  mode\_up.Text = m\_last\_up.ToString();  m\_data\_up = m\_up;  }  m\_up = 0;  }  m\_last\_up = points\_x[i].X;  }  break;  }  }  // Median  switch (listBox1.SelectedIndex)  {  case 1:  //summ\_right += points\_y[i].X;  //summ\_up += points\_x[i].X;  if (points\_x.Count % 2 == 0)  {  median\_right.Text = ((points\_y[(int)Math.Ceiling((double)points\_x.Count / 2)].X + points\_y[(int)Math.Ceiling((double)points\_x.Count / 2) - 1].X) / 2).ToString();  median\_up.Text = ((points\_x[(int)Math.Ceiling((double)points\_x.Count / 2)].X + points\_x[(int)Math.Ceiling((double)points\_x.Count / 2) - 1].X) / 2 ).ToString();  }  else  {  median\_right.Text = points\_y[ (int)Math.Ceiling((double)points\_x.Count / 2) ].X.ToString();  median\_up.Text = points\_x[(int)Math.Ceiling((double)points\_x.Count / 2)].X.ToString();  }  break;  case 2:  //summ\_right += points\_y[i].Y;  //summ\_up += points\_x[i].Y;  if (points\_x.Count % 2 == 0)  {  median\_right.Text = ((points\_y[(int)Math.Ceiling((double)points\_x.Count / 2)].Y + points\_y[(int)Math.Ceiling((double)points\_x.Count / 2) - 1].Y) / 2).ToString();  median\_up.Text = ((points\_x[(int)Math.Ceiling((double)points\_x.Count / 2)].Y + points\_x[(int)Math.Ceiling((double)points\_x.Count / 2) - 1].Y) / 2).ToString();  }  else  {  median\_right.Text = points\_y[(int)Math.Ceiling((double)points\_x.Count / 2)].Y.ToString();  median\_up.Text = points\_x[(int)Math.Ceiling((double)points\_x.Count / 2)].Y.ToString();  }  break;  case 3:  //summ\_right += points\_y[i].Y;  //summ\_up += points\_y[i].X;  if (points\_x.Count % 2 == 0)  {  median\_up.Text = ((points\_y[(int)Math.Ceiling((double)points\_x.Count / 2)].X + points\_y[(int)Math.Ceiling((double)points\_x.Count / 2) - 1].X) / 2).ToString();  points\_y = Sort\_list(points\_y, false);  median\_right.Text = ((points\_y[(int)Math.Ceiling((double)points\_x.Count / 2)].Y + points\_y[(int)Math.Ceiling((double)points\_x.Count / 2) - 1].Y) / 2).ToString();  }  else  {  median\_up.Text = points\_y[(int)Math.Ceiling((double)points\_x.Count / 2)].X.ToString();  points\_y = Sort\_list(points\_y, false);  median\_right.Text = points\_y[(int)Math.Ceiling((double)points\_x.Count / 2)].Y.ToString();  }  break;  default:  //summ\_right += points\_x[i].Y;  //summ\_up += points\_x[i].X;  if (points\_x.Count % 2 == 0)  {  median\_up.Text = ((points\_x[(int)Math.Ceiling((double)points\_x.Count / 2)].X + points\_x[(int)Math.Ceiling((double)points\_x.Count / 2) - 1].X) / 2).ToString();  points\_x = Sort\_list(points\_x, false);  median\_right.Text = ((points\_x[(int)Math.Ceiling((double)points\_x.Count / 2)].Y + points\_x[(int)Math.Ceiling((double)points\_x.Count / 2) - 1].Y) / 2).ToString();  }  else  {  median\_up.Text = points\_x[(int)Math.Ceiling((double)points\_x.Count / 2)].X.ToString();  points\_x = Sort\_list(points\_x, false);  median\_right.Text = points\_x[(int)Math.Ceiling((double)points\_x.Count / 2)].Y.ToString();  }  break;  }  median\_right.Text = Math.Round(float.Parse(median\_right.Text), 2).ToString();  median\_up.Text = Math.Round(float.Parse(median\_up.Text), 2).ToString();  Main\_box.Refresh();  Up\_box.Refresh();  Right\_box.Refresh();  }  private void timer1\_Tick(object sender, EventArgs e)  {  if (start\_work && (is\_y1\_param\_change || is\_y2\_param\_change))  {  if (is\_y1\_param\_change) generate\_y\_points(2);  else generate\_y\_points(1);  view\_datagreed();  draw\_main\_img();  is\_y1\_param\_change = false;  is\_y2\_param\_change = false;  }  else if (start\_work && (is\_axes\_change || is\_density\_change))  {  draw\_main\_img();  is\_axes\_change = false;  is\_density\_change = false;  }  }  public List<PointF> Sort\_list(List<PointF> list, bool is\_fist = true)  {  float[,] massive\_data = new float[2, list.Count];  for (int i = 0; i < list.Count; i++)  {  massive\_data[0, i] = list[i].X;  massive\_data[1, i] = list[i].Y;  }  int n = list.Count;  for (int i = 0; i < n - 1; i++)  for (int j = 0; j < n - i - 1; j++)  if (is\_fist)  {  if (massive\_data[0, j] > massive\_data[0, j + 1])  {  var tempVar = massive\_data[0, j];  massive\_data[0, j] = massive\_data[0, j + 1];  massive\_data[0, j + 1] = tempVar;  tempVar = massive\_data[1, j];  massive\_data[1, j] = massive\_data[1, j + 1];  massive\_data[1, j + 1] = tempVar;  }  }  else  {  if (massive\_data[1, j] > massive\_data[1, j + 1])  {  var tempVar = massive\_data[1, j];  massive\_data[1, j] = massive\_data[1, j + 1];  massive\_data[1, j + 1] = tempVar;  tempVar = massive\_data[0, j];  massive\_data[0, j] = massive\_data[0, j + 1];  massive\_data[0, j + 1] = tempVar;  }  }  list.Clear();  for (int i = 0; i < Int32.Parse(num\_point.Text); i++)  {  list.Add(new PointF(massive\_data[0, i], massive\_data[1, i]));  }  return list;  }  private void Start\_but\_Click(object sender, EventArgs e)  {  if (Int32.Parse(num\_point.Text) > 0)  {  for (int i = 0; i < Int32.Parse(num\_point.Text); i++)  {  PointF generate = new Point();  generate.X = (float)Math.Round(random.NextDouble(), 2);  generate.Y = (float)Math.Round(random.NextDouble(), 2);  points\_x.Add(generate);  points\_y.Add(generate);  }  start\_work = true;  //Sort\_x();  generate\_y\_points();  view\_datagreed();  draw\_main\_img();  timer1.Start();  }  }  public void generate\_y\_points(int case\_y = 0)  {  double k1\_y1\_p = 0.0f, k2\_y1\_p = 0.0f, k3\_y1\_p = 0.0f, k4\_y1\_p = 0.0f;  double b2\_y1\_p = 0.0f, b3\_y1\_p = 0.0f, b4\_y1\_p = 0.0f;  double k1\_y2\_p = 0.0f, k2\_y2\_p = 0.0f, k3\_y2\_p = 0.0f, k4\_y2\_p = 0.0f;  double b2\_y2\_p = 0.0f, b3\_y2\_p = 0.0f, b4\_y2\_p = 0.0f;  if (case\_y == 2 || case\_y == 0)  {  // For X1  // generate param k and b for x < 0.25  k1\_y1\_p = (((double)(y1\_1\_param.Value) / 100)) / (0.25 - 0);  k1\_y1.Text = k1\_y1\_p.ToString();  b1\_y1.Text = "0";  // generate param k and b for x >= 0.25 && x < 0.5  k2\_y1\_p = ((((double)(y1\_2\_param.Value) / 100) - ((double)(y1\_1\_param.Value)) / 100)) / (0.5 - 0.25);  b2\_y1\_p = ((double)(y1\_2\_param.Value) / 100) - k2\_y1\_p \* 0.25;  k2\_y1.Text = k2\_y1\_p.ToString();  b2\_y1.Text = b2\_y1\_p.ToString();  // generate param k and b for x >= 0.5 && x < 0.75  k3\_y1\_p = ((((double)(y1\_3\_param.Value) / 100) - ((double)(y1\_2\_param.Value)) / 100)) / (0.75 - 0.5);  b3\_y1\_p = ((double)(y1\_3\_param.Value) / 100) - k3\_y1\_p \* 0.5;  k3\_y1.Text = k3\_y1\_p.ToString();  b3\_y1.Text = b3\_y1\_p.ToString();  // generate param k and b for x >= 0.75  k4\_y1\_p = (1 - ((double)(y1\_3\_param.Value)) / 100) / (1 - 0.75);  b4\_y1\_p = ((double)(y1\_3\_param.Value) / 100) - k4\_y1\_p \* 0.75;  k4\_y1.Text = k4\_y1\_p.ToString();  b4\_y1.Text = b4\_y1\_p.ToString();    Graphics graphics = Graphics.FromImage(X1Y1\_box.Image);  Pen pen = new Pen(Color.Black);  graphics.FillRectangle(Brushes.White, new Rectangle(0, 0, X1Y1\_box.Width, X1Y1\_box.Height));  graphics.DrawLine(pen, 20, 100, 40, 100 - (int)y1\_1\_param.Value);  graphics.DrawLine(pen, 40, 100 - (int)y1\_1\_param.Value, 60, 100 - (int)y1\_2\_param.Value);  graphics.DrawLine(pen, 60, 100 - (int)y1\_2\_param.Value, 80, 100 - (int)y1\_3\_param.Value);  graphics.DrawLine(pen, 80, 100 - (int)y1\_3\_param.Value, 100, 20);  graphics.FillEllipse(Brushes.Red, 20 - 2, 100 - 2, 2 + 2, 2 + 2);  graphics.FillEllipse(Brushes.Red, 40 - 2, 100 - (int)y1\_1\_param.Value - 2, 2 + 2, 2 + 2);  graphics.FillEllipse(Brushes.Red, 60 - 2, 100 - (int)y1\_2\_param.Value - 2, 2 + 2, 2 + 2);  graphics.FillEllipse(Brushes.Red, 80 - 2, 100 - (int)y1\_3\_param.Value - 2, 2 + 2, 2 + 2);  graphics.FillEllipse(Brushes.Red, 100 - 2, 20 - 2, 2 + 2, 2 + 2);  X1Y1\_box.Refresh();  }  if (case\_y == 1 || case\_y == 0)  {  // For X2 and Y2  // generate param k and b for x < 0.25  k1\_y2\_p = (((double)(y2\_1\_param.Value) / 100)) / (0.25 - 0);  k1\_y2.Text = k1\_y2\_p.ToString();  b1\_y2.Text = "0";  // generate param k and b for x >= 0.25 && x < 0.5  k2\_y2\_p = ((((double)(y2\_2\_param.Value) / 100) - ((double)(y2\_1\_param.Value)) / 100)) / (0.5 - 0.25);  b2\_y2\_p = ((double)(y2\_2\_param.Value) / 100) - k2\_y2\_p \* 0.25;  k2\_y2.Text = k2\_y2\_p.ToString();  b2\_y2.Text = b2\_y2\_p.ToString();  // generate param k and b for x >= 0.5 && x < 0.75  k3\_y2\_p = ((((double)(y2\_3\_param.Value) / 100) - ((double)(y2\_2\_param.Value)) / 100)) / (0.75 - 0.5);  b3\_y2\_p = ((double)(y2\_3\_param.Value) / 100) - k3\_y2\_p \* 0.5;  k3\_y2.Text = k3\_y2\_p.ToString();  b3\_y2.Text = b3\_y2\_p.ToString();  // generate param k and b for x >= 0.75  k4\_y2\_p = (1 - ((double)(y2\_3\_param.Value)) / 100) / (1 - 0.75);  b4\_y2\_p = ((double)(y2\_3\_param.Value) / 100) - k4\_y2\_p \* 0.75;  k4\_y2.Text = k4\_y2\_p.ToString();  b4\_y2.Text = b4\_y2\_p.ToString();  Graphics graphics = Graphics.FromImage(X2Y2\_box.Image);  Pen pen = new Pen(Color.Black);  graphics.FillRectangle(Brushes.White, new Rectangle(0, 0, X2Y2\_box.Width, X2Y2\_box.Height));  graphics.DrawLine(pen, 20, 100, 40, 100 - (int)y2\_1\_param.Value);  graphics.DrawLine(pen, 40, 100 - (int)y2\_1\_param.Value, 60, 100 - (int)y2\_2\_param.Value);  graphics.DrawLine(pen, 60, 100 - (int)y2\_2\_param.Value, 80, 100 - (int)y2\_3\_param.Value);  graphics.DrawLine(pen, 80, 100 - (int)y2\_3\_param.Value, 100, 20);  graphics.FillEllipse(Brushes.Red, 20 - 2, 100 - 2, 2 + 2, 2 + 2);  graphics.FillEllipse(Brushes.Red, 40 - 2, 100 - (int)y2\_1\_param.Value - 2, 2 + 2, 2 + 2);  graphics.FillEllipse(Brushes.Red, 60 - 2, 100 - (int)y2\_2\_param.Value - 2, 2 + 2, 2 + 2);  graphics.FillEllipse(Brushes.Red, 80 - 2, 100 - (int)y2\_3\_param.Value - 2, 2 + 2, 2 + 2);  graphics.FillEllipse(Brushes.Red, 100 - 2, 20 - 2, 2 + 2, 2 + 2);  X2Y2\_box.Refresh();  }  for (int i = 0; i < Int32.Parse(num\_point.Text); i++)  {  if (points\_x[i].X < 0.25)  {  if (case\_y == 1) points\_y[i] = new PointF(points\_y[i].X, check\_max\_min\_point((float)(points\_x[i].Y \* k1\_y2\_p)));  else if (case\_y == 2) points\_y[i] = new PointF(check\_max\_min\_point((float)(points\_x[i].X \* k1\_y1\_p)), points\_y[i].Y);  else points\_y[i] = new PointF(check\_max\_min\_point((float)(points\_x[i].X \* k1\_y1\_p)), check\_max\_min\_point((float)(points\_x[i].Y \* k1\_y2\_p)));  }  else if (points\_x[i].X >= 0.25 && points\_x[i].X < 0.5)  {  if (case\_y == 1) points\_y[i] = new PointF(points\_y[i].X, check\_max\_min\_point((float)(points\_x[i].Y \* k2\_y2\_p + b2\_y2\_p)));  else if (case\_y == 2) points\_y[i] = new PointF(check\_max\_min\_point((float)(points\_x[i].X \* k2\_y1\_p + b2\_y1\_p)), points\_y[i].Y);  else points\_y[i] = new PointF(check\_max\_min\_point((float)(points\_x[i].X \* k2\_y1\_p + b2\_y1\_p)), check\_max\_min\_point((float)(points\_x[i].Y \* k2\_y2\_p + b2\_y2\_p)));  }  else if (points\_x[i].X >= 0.5 && points\_x[i].X < 0.75)  {  if (case\_y == 1) points\_y[i] = new PointF(points\_y[i].X, check\_max\_min\_point((float)(points\_x[i].Y \* k3\_y2\_p + b3\_y2\_p)));  else if (case\_y == 2) points\_y[i] = new PointF(check\_max\_min\_point((float)(points\_x[i].X \* k3\_y1\_p + b3\_y1\_p)), points\_y[i].Y);  else points\_y[i] = new PointF(check\_max\_min\_point((float)(points\_x[i].X \* k3\_y1\_p + b3\_y1\_p)), check\_max\_min\_point((float)(points\_x[i].Y \* k3\_y2\_p + b3\_y2\_p)));  }  else  {  if (case\_y == 1) points\_y[i] = new PointF(points\_y[i].X, check\_max\_min\_point((float)(points\_x[i].Y \* k4\_y2\_p + b4\_y2\_p)));  else if (case\_y == 2) points\_y[i] = new PointF(check\_max\_min\_point((float)(points\_x[i].X \* k4\_y1\_p + b4\_y1\_p)), points\_y[i].Y);  else points\_y[i] = new PointF(check\_max\_min\_point((float)(points\_x[i].X \* k4\_y1\_p + b4\_y1\_p)), check\_max\_min\_point((float)(points\_x[i].Y \* k4\_y2\_p + b4\_y2\_p)));  }  }  }  public void view\_datagreed()  {  if (Int32.Parse(num\_point.Text) > 0)  {  dataGridView1.RowCount = 1;  for (int i = 0; i < Int32.Parse(num\_point.Text); i++)  {  dataGridView1.RowCount += 1;  dataGridView1.Rows[i].Cells[0].Value = points\_x[i].X;  dataGridView1.Rows[i].Cells[1].Value = points\_y[i].X;  dataGridView1.Rows[i].Cells[2].Value = points\_x[i].Y;  dataGridView1.Rows[i].Cells[3].Value = points\_y[i].Y;  }  }  }  private void Send\_Click(object sender, EventArgs e)  {  points\_x.Clear();  points\_y.Clear();  dataGridView1.Rows.Clear();  dataGridView1.Refresh();  k4\_y2.Clear(); k3\_y2.Clear(); k2\_y2.Clear(); k1\_y2.Clear(); k4\_y1.Clear(); k3\_y1.Clear(); k2\_y1.Clear(); k1\_y1.Clear();  b4\_y1.Clear(); b3\_y1.Clear(); b2\_y1.Clear(); b1\_y1.Clear(); b4\_y2.Clear(); b3\_y2.Clear(); b2\_y2.Clear(); b1\_y2.Clear();  Graphics graphics = Graphics.FromImage(X1Y1\_box.Image);  graphics.FillRectangle(Brushes.White, new Rectangle(0, 0, X1Y1\_box.Width, X1Y1\_box.Height));  X1Y1\_box.Refresh();  graphics = Graphics.FromImage(X2Y2\_box.Image);  graphics.FillRectangle(Brushes.White, new Rectangle(0, 0, X2Y2\_box.Width, X2Y2\_box.Height));  X2Y2\_box.Refresh();  graphics = Graphics.FromImage(Main\_box.Image);  graphics.FillRectangle(Brushes.White, new Rectangle(0, 0, Main\_box.Width, Main\_box.Height));  Main\_box.Refresh();  graphics = Graphics.FromImage(Up\_box.Image);  graphics.FillRectangle(Brushes.White, new Rectangle(0, 0, Up\_box.Width, Up\_box.Height));  Up\_box.Refresh();  graphics = Graphics.FromImage(Right\_box.Image);  graphics.FillRectangle(Brushes.White, new Rectangle(0, 0, Right\_box.Width, Right\_box.Height));  Right\_box.Refresh();  }  private void y1\_1\_param\_ValueChanged\_1(object sender, EventArgs e)  {  is\_y1\_param\_change = true;  }  private void y2\_1\_param\_ValueChanged(object sender, EventArgs e)  {  is\_y2\_param\_change = true;  }  private void listBox1\_Click(object sender, EventArgs e)  {  is\_axes\_change = true;  }  private void is\_density\_CheckedChanged(object sender, EventArgs e)  {  is\_density\_change = true;  }  }  } |