Helper.cs

```
namespace FDM;
public struct Matrix /// Структура матрицы
{
    public int N;
    public int maxIter;
    public double EPS;
    public int shift1, dshift, shift2;
    public double[] di, du1, du2, dl1, dl2, pr, x, absolut_x;
}

public static class Helper
{
    //* Вычисление нормы вектора
    public static double Norm(double[] array) {
        double norm = 0;
        for (int i = 0; i < array.Count(); i++)
            norm += array[i] * array[i];
        return Sqrt(norm);
    }
}
```

Data.cs

```
namespace FDM;
public class Data
{
    //* Данные задачи
    public int         CountX { get; set; }   /// Количество точек на оси X
    public int         CountY { get; set; }   /// Количество точек на оси Y
    public double[]    X      { get; set; }   /// Значения X-ов
    public double[]    Y      { get; set; }   /// Значения Y-ов
    public int         GX     { get; set; }   /// К-во точек на нижней границе области "Г"
    public int         GY     { get; set; }   /// К-во точек на правой границе области "Г"

    public void Deconstruct(out int      countX,
                            out int      countY,
                            out double[] x,
                            out double[] y,
                            out int      gx,
                            out int      gy)
    {
        countX = CountX;
        countY = CountY;
        x = X;
        y = Y;
        gx = GX;
        gy = GY;
    }
}
```

Function.cs

```
namespace FDM;
public static class Function
{
    public static uint   NumberFunc;     /// Номер задачи
    public static double lambda;         /// Лямбда
    public static double gamma;          /// Гамма

    //* Инициализации лямбды и гаммы
    public static void Init() {
        switch(NumberFunc)
        {
            case 1: /// easy
            lambda = 2; gamma = 3;
            break;

            case 2: /// sec_kraev
            lambda = 2; gamma = 3;
            break;

            case 3: /// polynom_2
            lambda = 2; gamma = 3;
            break;

            case 4: /// polynom_3
            lambda = 2; gamma = 3;
            break;

            case 5: /// polynom_4
            lambda = 2; gamma = 3;
            break;

            case 6: /// not_polynom
            lambda = 1; gamma = 1;
            break;
        }
    }

    //* Функция u(x,y)
    public static double Absolut(double x, double y, uint side = 0)
    {
        switch(NumberFunc)
        {
            case 1: /// easy
            return 2*x + 4*y;

            case 2: /// sec_kraev
            return side switch
            {
                3 => 4,
                6 => 8,
                _ => 2*x + 4*y
            };

            case 3: /// polynom_2
            return 2*Pow(x, 2) + 4*Pow(y, 2);

            case 4: /// polynom_3
            return 2*Pow(x, 3) + 4*Pow(y, 3);

            case 5: /// polynom_4
            return 2*Pow(x, 4) + 4*Pow(y, 4);

            case 6: /// not_polynom
            return Sin(x + y);
        }
        return 0;
    }

    //* Функция f(x,y)
    public static double Func(double x, double y)
    {
        switch(NumberFunc)
        {
            case 1: /// easy
            return 6*x + 12*y;

            case 2: /// sec_kraev
            return 6*x + 12*y;

            case 3: /// polynom_2
            return 6*Pow(x, 2) + 12*Pow(y, 2) - 24;

            case 4: /// polynom_3
            return -24*x - 48*y + 6*Pow(x, 3) + 12*Pow(y, 3);

            case 5: /// polynom_4
            return -48*Pow(x, 2) - 96*Pow(y, 2) + 6*Pow(x, 4) + 12*Pow(y, 4);

            case 6: /// not_polynom
            return 3*Sin(x + y);
        }
        return 0;
    }

    //* Функция проверки имеется ли на стороне области второе краевое
    public static bool IsSecondKraev(uint side)
    {
        switch(NumberFunc)
        {
            case 1: /// easy
            return side switch
            {
                _ => false,
            };

            case 2: /// sec_kraev
            return side switch
            {
                3 => true,
                6 => true,
                _ => false
            };

            case 3: /// polynom_2
            return side switch
            {
                _ => false,
            };

            case 4: /// polynom_3
            return side switch
            {
                _ => false,
            };

            case 5: /// polynom_4
            return side switch
            {
                _ => false,
            };

            case 6: /// not_polynom
            return side switch
            {
                _ => false,
            };
        }
        return false;
    }
}
```

Seidel.cs

```
namespace FDM;
public class Seidel
{
    private Matrix matrix; /// Матрица
    private double omega;  /// Параметр релаксации

    public Seidel(Matrix matrix, int iter, double eps, double omega = 1) {
        this.matrix         = matrix;
        this.omega          = omega;
        this.matrix.maxIter = iter;
        this.matrix.EPS     = eps;
    }

    //* Решение СЛАУ
    public void solve(bool flag = false) {
        double sum, Nev = 0, norm_f;
        int Iter = 0;
        norm_f = Norm(matrix.pr);

        do {
            Nev = 0;
            for (int i = 0; i < matrix.N; i++) {
                sum = matrix.di[i] * matrix.x[i];

                if (i < matrix.N - 1)
                    sum += matrix.du1[i] * matrix.x[i + 1];
                if (i >= 2)
                    sum += matrix.dl1[i - 1] * matrix.x[i - 1];

                if (i < matrix.dshift)
                    sum += matrix.du2[i] * matrix.x[matrix.shift1 + i];
                else if (i < matrix.N - matrix.shift2)
                    sum += matrix.du2[i] * matrix.x[matrix.shift2 + i];

                if (i >= matrix.shift1 + matrix.dshift)
                    sum += matrix.dl2[i - matrix.shift1] * matrix.x[i - matrix.shift2];
                else if (i >= matrix.shift1)
                    sum += matrix.dl2[i - matrix.shift1] * matrix.x[i - matrix.shift1];

                Nev += (matrix.pr[i] - sum) * (matrix.pr[i] - sum);
                matrix.x[i] += omega / matrix.di[i] * (matrix.pr[i] - sum);
            }

            Nev = Sqrt(Nev) / norm_f; /// Относительная невязка
            Iter++;
            if (flag)
                WriteLine($"Iter: {Iter, -10} Nev: {Nev.ToString("E3")}");
        } while (Nev > matrix.EPS &&
                    Iter <= matrix.maxIter);
    }
}
```

Solve.cs

```
namespace FDM;
public class Solve
{
    public int         CountX { get; set; }   /// Количество точек на оси X
    public int         CountY { get; set; }   /// Количество точек на оси Y
    public double[]    X      { get; set; }   /// Значения X-ов
    public double[]    Y      { get; set; }   /// Значения Y-ов
    public int         GX     { get; set; }   /// К-во точек на нижней границе области "Г"
    public int         GY     { get; set; }   /// К-во точек на правой границе области "Г"
    private Matrix matrix;   /// 5-диагональная матрица

    public Solve(Data data, uint Num) {
        (CountX, CountY, X, Y, GX, GY) = data;
        Function.NumberFunc = Num;
        Function.Init();
    }

    //* Решение задачи
    public void solve() {
        memory();                                        //? Выделение памяти
        completion();                                    //? Заполнение матрицы
        var task = new Seidel(matrix, 10000, 1e-14);     //? Создание метода Гаусса-Зейделя
        task.solve(true);                                //? Решение СЛАУ
        writeTable();                                    //? Записб таблички с решением
    }

    //* Заполнение матрицы (область "Г") снизу->вверх
    private void completion() {
        int id = 0;                //: индекс узла
        double hx1, hx2, hy1, hy2; //: h-ки приращения аргументов

        // Нижняя линия области "Г"
        matrix.pr[0] = Absolut(X[0], Y[0]);       // Левый нижний угол
        hy1 = Abs(Y[0] - Y[1]);
        for (int i = 1; i < GX - 1; i++) {
            matrix.pr[i] = Absolut(X[i], Y[0], 1);
            if (IsSecondKraev(1)) {
                matrix.di[i]  = -lambda / hy1;
                matrix.du2[i] =  lambda / hy1;
            }
        }
        id = GX - 1;
        matrix.pr[id] = Absolut(X[GX - 1], Y[0]);  // Правый нижний угол
        id++;

        // До линии между шапкой и ножкой
        for (int i = 1; i < GY - 1; i++, id++) {
            hy1 = Abs(Y[i] - Y[i - 1]);
            hy2 = Abs(Y[i + 1] - Y[i]);
            matrix.pr[id] = Absolut(X[0], Y[i], 2);
            if (IsSecondKraev(2)) {
                hx1 = Abs(X[0] - X[1]);
                matrix.di [id] = -lambda / hx1;
                matrix.du1[id] =  lambda / hx1;
            }
            id++;

            for (int j = 1; j < GX - 1; j++, id++) {
                hx1 = Abs(X[j] - X[j - 1]);
                hx2 = Abs(X[j + 1] - X[j]);
                matrix.pr [id]                 = Func(X[j], Y[i]);
                matrix.dl1[id - 1]             = -2*lambda / (hx1 * (hx2 + hx1));
                matrix.dl2[id - matrix.shift1] = -2*lambda / (hy1 * (hy2 + hy1));
                matrix.du1[id]                 = -2*lambda / (hx2 * (hx2 + hx1));
                matrix.du2[id]                 = -2*lambda / (hy2 * (hy2 + hy1));
                matrix.di [id]                 = lambda * (2/(hx1*hx2) + 2/(hy1*hy2)) + gamma;
            }

            matrix.pr[id] = Absolut(X[GX - 1], Y[i], 3);
            if (IsSecondKraev(3)) {
                hx1 = Abs(X[GX - 1] - X[GX - 2]);
                matrix.di[id] = lambda / hx1;
                matrix.dl1[id - 1] = -lambda / hx1;
            }
        }

        matrix.dshift = id;

        // Между шляпкой и ножкой
        hy1 = Abs(Y[GY - 1] - Y[GY - 2]);
        hy2 = Abs(Y[GY] - Y[GY - 1]);
        matrix.pr[id] = Absolut(X[0], Y[GY - 1], 2);
        if (IsSecondKraev(2)) {
            hx1 = Abs(X[0] - X[1]);
            matrix.di[id] = -lambda / hx1;
            matrix.du1[id] = lambda / hx1;
        }
        id++;

        for (int i = 1; i < GX; i++, id++) {
            hx1 = Abs(X[i] - X[i - 1]);
            hx2 = Abs(X[i + 1] - X[i]);
            matrix.pr[id]                  = Func(X[i], Y[GY - 1]);
            matrix.dl1[id - 1]             = -2*lambda / (hx1 * (hx2 + hx1));
            matrix.dl2[id - matrix.shift1] = -2*lambda / (hy1 * (hy2 + hy1));
            matrix.du1[id]                 = -2*lambda / (hx2 * (hx2 + hx1));
            matrix.du2[id]                 = -2*lambda / (hy2 * (hy2 + hy1));
            matrix.di [id]                 = lambda * (2/(hx1*hx2) + 2/(hy1*hy2)) + gamma;
        }

        for (int i = GX; i < CountX - 1; i++, id++) {
            matrix.pr[id] = Absolut(X[i], Y[GY - 1], 4);
            if (IsSecondKraev(4)) {
                matrix.di[id] = -lambda / hy2;
                matrix.du2[id] = lambda / hy2;
            }
        }
        matrix.pr[id] = Absolut(X[CountX - 1], Y[GY - 1]);
        id++;

        // Шляпка
        for (int i = GY; i < CountY - 1; i++) {
            hy1 = Abs(Y[i] - Y[i - 1]);
            hy2 = Abs(Y[i + 1] - Y[i]);
            matrix.pr[id] = Absolut(X[0], Y[i], 2);
            if (IsSecondKraev(2)) {
                hx1 = Abs(X[0] - X[1]);
                matrix.di[id] = -lambda / hx1;
                matrix.du1[id] = lambda / hx1;
            }
            id++;

            for (int j = 1; j < CountX - 1; j++, id++) {
                hx1 = Abs(X[j] - X[j - 1]);
                hx2 = Abs(X[j + 1] - X[j]);
                matrix.pr[id]                  = Func(X[j], Y[i]);
                matrix.dl1[id - 1]             = -2*lambda / (hx1 * (hx2 + hx1));
                matrix.dl2[id - matrix.shift1] = -2*lambda / (hy1 * (hy2 + hy1));
                matrix.du1[id]                 = -2*lambda / (hx2 * (hx2 + hx1));
                matrix.du2[id]                 = -2*lambda / (hy2 * (hy2 + hy1));
                matrix.di [id]                 = lambda * (2/(hx1*hx2) + 2/(hy1*hy2)) + gamma;
            }
            matrix.pr[id] = Absolut(X[CountX - 1], Y[i], 5);
            if (IsSecondKraev(5)) {
                hx1 = Abs(X[CountX - 1] - X[CountX - 2]);
                matrix.di[id] = lambda / hx1;
                matrix.dl1[id - 1] = -lambda / hx1;
            }
            id++;
        }

        // Верхушка шляпки
        matrix.pr[id] = Absolut(X[0], Y[CountY - 1]);
        id++;
        hy1 = Abs(Y[CountY - 1] - Y[CountY - 2]);
        for (int i = 1; i < CountX - 1; i++, id++) {
            matrix.pr[id] = Absolut(X[i], Y[CountY - 1], 6);
            if (IsSecondKraev(6)) {
                matrix.di[id] = lambda / hy1;
                matrix.dl2[id - matrix.shift1] = -lambda / hy1;
            }
        }
        matrix.pr[id] = Absolut(X[CountX - 1], Y[CountY - 1]);
    }

    //* Выделяем память под матрицу
    private void memory() {
        matrix.N = GX * (CountY - GY) + CountX * GY;  // Размерность матрицы
        matrix.shift1    = GX;
        matrix.shift2    = CountX;
        matrix.di        = new double[matrix.N];
        matrix.du1       = new double[matrix.N];
        matrix.du2       = new double[matrix.N];
        matrix.dl1       = new double[matrix.N];
        matrix.dl2       = new double[matrix.N];
        matrix.pr        = new double[matrix.N];
        matrix.x         = new double[matrix.N];
        matrix.absolut_x = new double[matrix.N];
        Array.Fill(matrix.di, 1); // Заполнение диагонали единицами
    }

    //* Заполнение и запись таблички с решением
    private void writeTable() {
        StringBuilder table = new StringBuilder();
        string margin = String.Join("", Enumerable.Repeat("-", 16));

        table.Append(String.Join("", Enumerable.Repeat("-", 86)) + "\n");
        table.Append($"|X{" ",-14} | Y{" ",-13} | U{" ",-12}  | U`{" ",-12} | |U`- U| {" ",-7}|\n");
        table.Append($"|" + margin + "|" + margin + "|" + margin + "|" + margin + "|" + margin + "|\n");

        int id      = 0;
        for (int i = 0; i < CountY - GY; i++) {
            for (int j = 0; j < GX; j++, id++) {
                double absolut = Absolut(X[j], Y[i]);
                table.Append($"|{String.Format("{0,16}", X[j])}" +
                                $"|{String.Format("{0,16}", Y[i])}" +
                                $"|{String.Format("{0,16}", matrix.x[id].ToString("E6"))}" +
                                $"|{String.Format("{0,16}", absolut.ToString("E6"))}" +
                                $"|{String.Format("{0,16}", Abs(absolut - matrix.x[id]).ToString("E6"))}|\n");
            }
        }

        for (int i = GY - 1; i < CountY; i++) {
            for (int j = 0; j < CountX; j++, id++) {
                double absolut = Absolut(X[j], Y[i]);
                table.Append($"|{String.Format("{0,16}", X[j])}" +
                                $"|{String.Format("{0,16}", Y[i])}" +
                                $"|{String.Format("{0,16}", matrix.x[id].ToString("E6"))}" +
                                $"|{String.Format("{0,16}", absolut.ToString("E6"))}" +
                                $"|{String.Format("{0,16}", Abs(absolut - matrix.x[id]).ToString("E6"))}|\n");
            }
        }
        table.Append(String.Join("", Enumerable.Repeat("-", 86)) + "\n");
        File.WriteAllText("test/tables/table.txt", table.ToString());
    }

}
```

Program.cs

```
try
{
    string json = File.ReadAllText(@"test\easy.json");         //: Простейший тест (1)
    //string json = File.ReadAllText(@"test\sec_kraev.json");     //: Тест на вторые краевые (2)
    //string json = File.ReadAllText(@"test\polynom_2.json");     //: Тест на полином второй степени (3)
    //string json = File.ReadAllText(@"test\polynom_3.json");     //: Тест на полином третьей степени (4)
    //string json = File.ReadAllText(@"test\polynom_4.json");     //: Тест на полином четвертой степени (5)
    //string json = File.ReadAllText(@"test\not_polynomh1.json"); //: Тест на не полиноминальной функции (h1 - шаг) (6)
    //string json = File.ReadAllText(@"test\not_polynomh2.json"); //: Тест на не полиноминальной функции (h1/2 - шаг) (6)
    //string json = File.ReadAllText(@"test\not_polynomh3.json"); //: Тест на не полиноминальной функции (h1/4 - шаг) (6)
    //string json = File.ReadAllText(@"test\uneven.json");        //: Тест на неравномерной сетке (1)

    Data data = JsonConvert.DeserializeObject<Data>(json)!;

    if (data is null) throw new FileNotFoundException("File uncorrected!");

    Solve task = new Solve(data, 1);
    task.solve();
}
catch (FileNotFoundException ex)
{
    WriteLine(ex.Message);
}
```