Helper.cs

```
namespace DigitalModels.other;

public struct Receiver      /// Структура приемника
{
    public double x { get; set; }  /// Координата X
    public double y { get; set; }  /// Координата Y
    public double z { get; set; }  /// Координата Z

    public Receiver(double _x, double _y, double _z) {
        this.x = _x; this.y = _y; this.z = _z;
    }

    public Receiver(double[] point) {
        this.x = point[0]; this.y = point[1]; this.z = point[2];
    }

    public void Deconstruct(out double x,
                            out double y,
                            out double z)
    {
        x = this.x;
        y = this.y;
        z = this.z;
    }

    public override string ToString() => $"{x,20} {y,24} {z,26}";
}

public struct Source      /// Структура источника
{
    public double x { get; set; }  /// Координата X
    public double y { get; set; }  /// Координата Y
    public double z { get; set; }  /// Координата Z

    public Source(double _x, double _y, double _z) {
        this.x = _x; this.y = _y; this.z = _z;
    }

    public Source(double[] point) {
        this.x = point[0]; this.y = point[1]; this.z = point[2];
    }

    public void Deconstruct(out double x,
                            out double y,
                            out double z)
    {
        x = this.x;
        y = this.y;
        z = this.z;
    }

    public override string ToString() => $"{x,20} {y,24} {z,26}";
}

public static class Helper
{
    //* Вычисление нормы вектора
    public static double Norma(Vector<double> vec) {
        double res = 0;
        for (int i = 0; i < vec.Length; i++)
            res += vec[i]*vec[i];
        return Sqrt(res);
    }
```

```
    //* Расстояние от точки измерения до электрода
    public static double interval(Receiver R, Source S) {
        return Sqrt(Pow(S.x - R.x, 2) + Pow(S.y - R.y, 2) + Pow(S.z - R.z, 2));
    }

    //* Расчет потенциала
    public static double potential(Source A, Source B, Receiver M, Receiver N, double I) {
        double diff = (1 / interval(M, B) - 1 / interval(M, A)) - (1 / interval(N, B) - 1 / interval(N, A));
        return I / (2.0 * PI * sigma) * diff;
    }

    //* Расчет diff потенциала
    public static double diff_potential(Source A, Source B, Receiver M, Receiver N, double I) {
        double diff = (1 / interval(M, B) - 1 / interval(M, A)) - (1 / interval(N, B) - 1 / interval(N, A));
        return -I / (2.0 * PI * sigma * sigma) * diff;
    }

    //* Расчет суммарного потенциала (1 приемник от 3 источников)
    public static double summPotential(Source[] sources, Receiver receiver, Receiver next_receiver, Vector<double> I) {
        double v1 = potential(sources[0], sources[1], receiver, next_receiver, I[0]);
        double v2 = potential(sources[2], sources[3], receiver, next_receiver, I[1]);
        double v3 = potential(sources[4], sources[5], receiver, next_receiver, I[2]);
        return v1 + v2 + v3;
    }

    //* Окно помощи при запуске (если нет аргументов или по команде)
    public static void ShowHelp() {
        WriteLine("----Команды----                        \n" +
        "-help             - показать справку             \n" +
        "-i                - входной файл                 \n");
    }
}
```

Function.cs

```
namespace DigitalModels;
public static class Function
{
    public static uint           numberFunc;     /// Номер задачи
    public static double         sigma;          /// Значение Sigma
    public static Vector<double> Absolut_I;      /// Истинное решение
    public static Vector<double> omega;          /// Значние omega = 1 / V (потенциал)
    public static Vector<double> I_init;         /// Начальное приближение

    //* Инициализация переменных
    public static void Init(uint numF) {
        numberFunc = numF;

        switch(numberFunc) {
            case 1:
                sigma     = 0.1;
                Absolut_I = new Vector<double>(new []{1.0, 2.0, 3.0});
                omega     = new Vector<double>(3);
                I_init    = new Vector<double>(new []{0.1, 0.1, 0.1});
            break;
        }
    }
}
```

Gauss.cs

```
namespace DigitalModels;
public class Gauss {

    private Matrix mat;           /// Матрица
    private Vector<double> vec;   /// Правая часть
    private int N { get; }        /// Размерность СЛАУ
    private double EPS { get; }   /// Точность

    // ***** Конструктор ***** //
    public Gauss(int N, double eps) {
        this.mat = new Matrix(N);
        this.vec = new Vector<double>(N);
        this.N   = N;
        this.EPS = eps;
    }

    //* Метод решения
    public bool solve(Matrix matrix, Vector<double> vector, Vector<double> res) {
        Matrix.Copy(matrix, mat);
        Vector<double>.Copy(vector, vec);

        for (int i = 0; i < N; i++) {

            // Ищем максимальный элемент
            double MAX = Abs(mat[i,i]);
            int    ind = i;
            for (int s = i + 1; s < N; s++)
                if (Abs(mat[s,i]) > MAX)
                    (MAX, ind) = (Abs(mat[s,i]), s);

            // Проверка на нулевой столбец
            if (MAX <= EPS)
                return false;

            // Меняем строки СЛАУ (т.е. вместе с правой частью)
            for (int k = 0; k < N; k++)
                (mat[i,k], mat[ind,k]) = (mat[ind,k], mat[i,k]);
            (vec[i], vec[ind]) = (vec[ind], vec[i]);

            // Нормируем уравнения
            for (int j = i; j < N; j++) {
                double temp = mat[j,i];
                // Для нуля if пропускается
                if (Abs(temp) > EPS) {
                    for (int p = 0; p < N; p++)
                        mat[j,p] /= temp;
                    vec[j] /= temp;

                    // if на то чтобы не вычитать строку саму из себя
                    if (j != i) {
                        for (int p = 0; p < N; p++)
                            mat[j,p] -= mat[i,p];
                        vec[j] -= vec[i];
                    }
                }
            }
        }

        // Решаем обратным ходом
        for (int i = N - 1; i >= 0; i--) {
            res[i] = vec[i];
            for (int j = 0; j < i; j++)
                vec[j] -= mat[j,i] * res[i];
        }
```

```
        return true;
    }
}
```

Solve.cs

```
namespace DigitalModels;
public class Solve {

    public Receiver[] Receivers   { get; set; }    /// Положение приемников
    public Source[]   Sources     { get; set; }    /// Положение источников
    Gauss gauss;                                   /// Метод решения СЛАУ (ГАУСС)

    // ****** Конструктор ***** //
    public Solve(Data data) {
        (Receivers, Sources) = data;
        gauss                = new Gauss(3, 1e-15);
    }

    //* Основной метод решения
    public void solve() {

        // Считаем сумму потенциалов и находим omega
        Vector<double> V_fact = new Vector<double>(3);
        for (int i = 0; i < V_fact.Length; i++)
            V_fact[i] = summPotential(Sources, Receivers[2*i], Receivers[2*i + 1], Absolut_I);

        Function.omega = new Vector<double>(new[]{1 / V_fact[0], 1 / V_fact[1], 1 / V_fact[2]});

        // Построение матрицы
        Matrix matrix = CreateMatrix();

        // Построение вектора
        Vector<double> vector = CreateVector(V_fact);

        // Решение СЛАУ
        Vector<double> result = new Vector<double>(vector.Length);
        if (!gauss.solve(matrix, vector, result))
            regularization(matrix, vector, result);

        WriteLine(result.ToString());
    }

    //* Регуляризация
    private void regularization(Matrix mat, Vector<double> vec, Vector<double> res) {
        double a_pred = 0, a_next = 1e-15;

        // Находим значение (a) при котором СЛАУ решится
        do {
            for (int i = 0; i < vec.Length; i++)
                mat[i,i] = mat[i,i] - a_pred + a_next;
            a_pred = a_next;
            a_next *= 1.4;
        } while (!gauss.solve(mat, vec, res));

        // Находим наилучшее значение (a)
        double CURRENT_DISCREPANCY = Discrepancy(mat, vec, a_pred);
        a_next = a_pred * 1.4;

        do {
            for (int i = 0; i < vec.Length; i++)
                mat[i,i] = mat[i,i] - a_pred + a_next;
            CURRENT_DISCREPANCY = Discrepancy(mat, vec, a_next);
            a_pred = a_next;
```

```
            a_next *= 1.4;
        } while (CURRENT_DISCREPANCY < 1e-15);
    }

    //* Вычисление невязки
    public double Discrepancy(Matrix mat, Vector<double> vec, double a) {
        // Копируем правую часть и матрицу
        Matrix copyMat = new Matrix(vec.Length);
        Vector<double> copyVec = new Vector<double>(vec.Length);
        Matrix.Copy(mat, copyMat);
        Vector<double>.Copy(vec, copyVec);

        // Прибавить (a)
        for (int i = 0; i < vec.Length; i++)
            copyMat[i,i] += a;

        // Решаем СЛАУ (Вызываем прям из класса с гарантией решаемости, т.к. СЛАУ исправленна)
        Vector<double> res = new Vector<double>(vec.Length);
        gauss.solve(copyMat, copyVec, res);

        return Norma(copyMat * res - copyVec);
    }

    //* Построение матрицы
    public Matrix CreateMatrix() {
        Matrix mat = new Matrix(3);
        for (int i = 0; i < 3; i++)
            for (int j = 0; j < 3; j++)
                for (int k = 0; k < 3; k++) {
                    // Вычислим аналитические производные
                    // Положим I_init = 1
                    double diff_i = diff_potential(Sources[2*i], Sources[2*i + 1], Receivers[2*k], Receivers[2*k + 1], sigma
                    double diff_j = diff_potential(Sources[2*j], Sources[2*j + 1], Receivers[2*k], Receivers[2*k + 1], sigma
                    mat[i,j] += Pow(omega[k], 2) * diff_i * diff_j;
                }
        return mat;
    }

    //* Построение вектора
    public Vector<double> CreateVector(Vector<double> V_fact) {
        Vector<double> vec = new Vector<double>(3);

        for (int i = 0; i < 3; i++)
            for (int j = 0; j < 3; j++) {
                double diff = diff_potential(Sources[2*i], Sources[2*i + 1], Receivers[2*j], Receivers[2*j + 1], sigma);
                double V_current = summPotential(Sources, Receivers[2*j], Receivers[2*j + 1], I_init);
                vec[i] += Pow(omega[j], 2) * diff * (V_current - V_fact[j]);
            }
        return vec;
    }
}
```

Program.cs

```
try {
    // Проверка аргументов
    if (args.Length == 0) throw new ArgumentNullException("Not found arguments!");
    if (args[0] == "-help") {
        ShowHelp(); return;
    }

    // Входные данные
    string json = File.ReadAllText(args[1]);
    Data data = JsonConvert.DeserializeObject<Data>(json)!;
    if (data is null) throw new FileNotFoundException("File uncorrected!");

    // Определение функции
    Function.Init(data.N);

    // Решение обратной задачи
    Solve task = new Solve(data);
    task.solve();
}
catch (FileNotFoundException ex) {
    WriteLine(ex.Message);
}
catch (ArgumentNullException ex) {
    ShowHelp();
    WriteLine(ex.Message);
}
```