



МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Новосибирский государственный технический университет»



**НГТУ
НЭТИ** | **Факультет прикладной
математики и информатики**

Кафедра прикладной математики

Лабораторная работа №1
по дисциплине «Уравнения математической физики»

**Решение эллиптических краевых задач методом конечных
разностей.
Дополнительное задание**

Студенты БЕГИЧЕВ АЛЕКСАНДР

ШИШКИН НИКИТА

Группа ПМ-92

Преподаватели ЗАДОРОВЫЙ А. Г.

ПАТРУШЕВ И. И.

ПЕРСОВА М. Г.

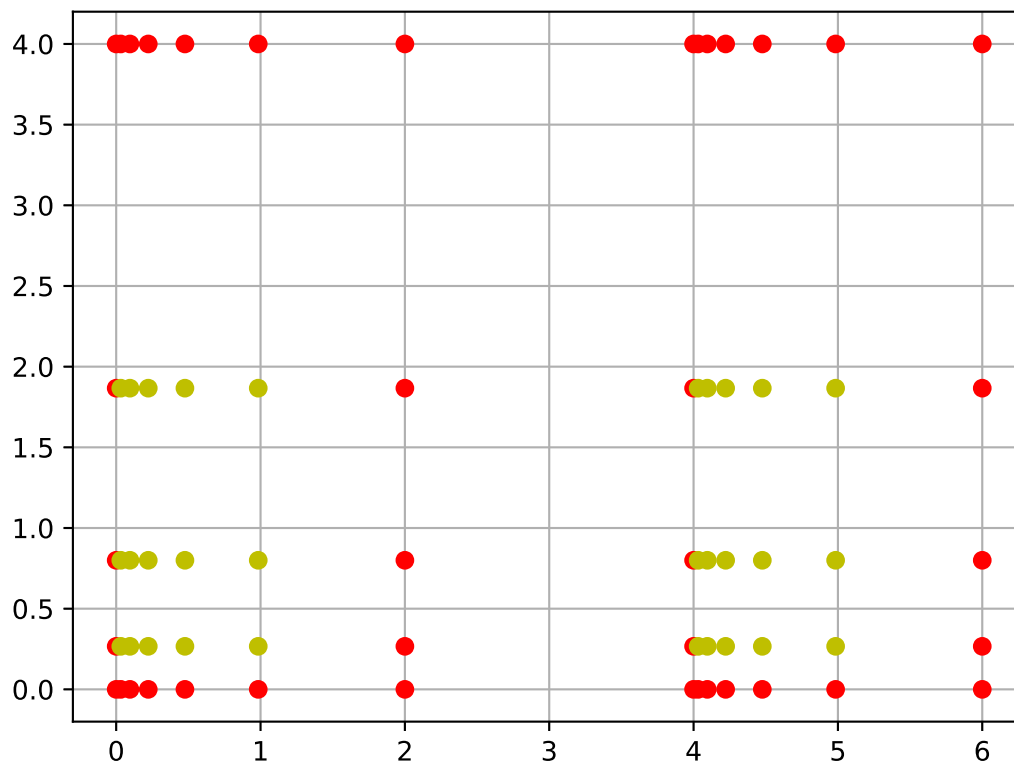
Новосибирск, 2022

Цель работы

Разработать программу для решения эллиптической краевой задачи методом конечных разностей, и при этом сетка может иметь любую прямоугольную форму.

Входные данные для сетки

Рассмотрим пример входных файлов для данной сетки:



```
You, now | 1 author (You)
1 0 2 4 6
2 0 4
3 6 6 6
4 4
5 2 2 2
6 2
7 0 1 0 0 1 0 1
8 1 1 0 2 3 0 1
```

Рис. 1: Входной файл для сетки

```
You, 1 second ago | 1 author (You)
1 2 0
2 1 0 1 0 0
3 1 0 1 1 1
4 1 2 3 0 0
5 1 2 3 1 1
6 1 0 0 0 1
7 1 1 1 0 1
8 1 2 2 0 1
9 1 3 3 0 1
```

Рис. 2: Входной файл для краевых

Входной файл для сетки

- * x -линии
- * y -линии
- * количество разбиений каждого интервала по x
- * количество разбиений каждого интервала по y
- * коэффициент разрядки для каждого интервала по x
- * коэффициент разрядки для каждого интервала по y

Далее задаются подобласти в виде:

- * номер подобласти
- * значение λ
- * значение γ
- * индекс в массиве x -линий (начало по x)
- * индекс в массиве x -линий (конец по x)
- * индекс в массиве y -линий (начало по y)
- * индекс в массиве y -линий (конец по y)

В программе есть возможность задать вложенную сетку, для этого во входной файл добавляется параметр, отвечающий за количество вложенности сетки. Также, если в программе выбрана равномерная сетка, тогда коэффициент разрядки во входном файле не задается.

Входной файл для краевых условий

- * значение β

Далее задаются краевые условия в виде:

- * тип краевого условия
- * индекс в массиве x -линий (начало по x)
- * индекс в массиве x -линий (конец по x)
- * индекс в массиве y -линий (начало по y)
- * индекс в массиве y -линий (конец по y)

Описание программы находится в основном отчете и не требует приведения изменений, так как архитектура программы осталась прежней.

Тестирование

Первый тест

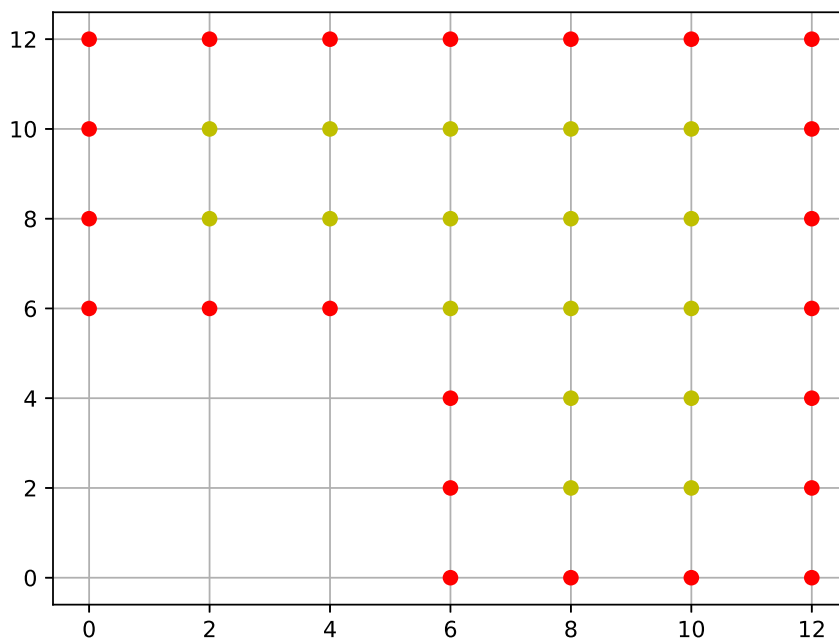
Функция: $x^3 + y$

Правая часть: $-6x$

Коэффициенты $\lambda = 1, \gamma = 0$

Сетка: равномерная

Заданы краевые условия 1-го рода и на левой верхней и правой границах краевое условие 2-го рода.



x_i	y_i	Точное	Численное	Вектор погрешности	Погрешность
2	8	16	15.99999999999405	$5.95 \cdot 10^{-12}$	$1.13 \cdot 10^{-14}$
2	10	18	17.99999999999534	$4.66 \cdot 10^{-12}$	—
4	8	72	71.99999999999245	$7.55 \cdot 10^{-12}$	—
4	10	74	73.99999999999461	$5.39 \cdot 10^{-12}$	—
6	6	222	221.99999999999843	$1.16 \cdot 10^{-11}$	—
6	8	224	223.9999999999987	$1.3 \cdot 10^{-11}$	—
6	10	226	225.99999999999255	$7.45 \cdot 10^{-12}$	—
8	2	514	513.9999999999987	$1.13 \cdot 10^{-11}$	—
8	4	516	515.9999999999982	$1.8 \cdot 10^{-11}$	—
8	6	518	517.9999999999784	$2.16 \cdot 10^{-11}$	—
8	8	520	519.9999999999982	$1.8 \cdot 10^{-11}$	—
8	10	522	521.9999999999907	$9.32 \cdot 10^{-12}$	—
10	2	1,002	1,001.99999999999842	$1.58 \cdot 10^{-11}$	—
10	4	1,004	1,003.9999999999758	$2.42 \cdot 10^{-11}$	—
10	6	1,006	1,005.999999999974	$2.6 \cdot 10^{-11}$	—
10	8	1,008	1,007.9999999999799	$2.01 \cdot 10^{-11}$	—
10	10	1,010	1,009.9999999999899	$1.01 \cdot 10^{-11}$	—

Второй тест

Функция: $4x^2 + y$

Правая часть: $-4 + 0.5 \cdot (4x^2 + y)$

Коэффициенты $\lambda = 0.5, \gamma = 0.5$

Сетка: неравномерная

Заданы краевые условия 1-го рода и на самых левой и правой границах краевые условия 2-го и 3-го рода.



Относительная погрешность	Относительная невязка
$1.52 \cdot 10^{-14}$	$5.81 \cdot 10^{-16}$

Третий тест

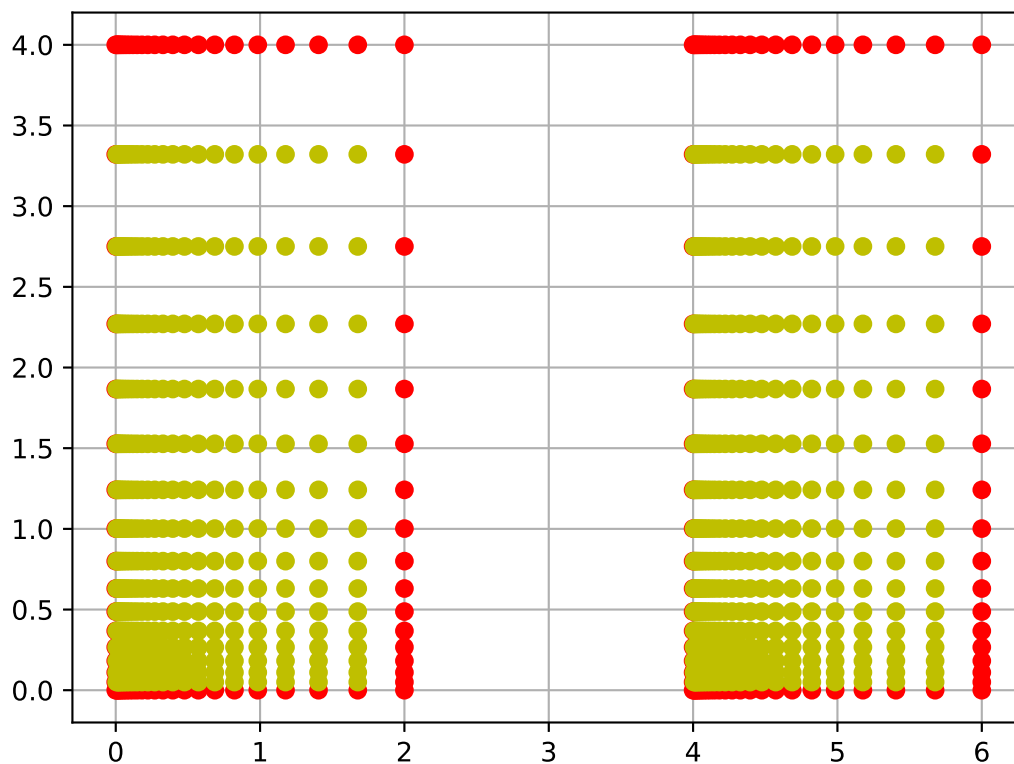
Функция: $x^2 - y$

Правая часть: $-1 + x^2 - y$

Коэффициенты $\lambda = 0.5, \gamma = 1$

Сетка: неравномерная

Заданы краевые условия 1-го рода на всех границах.



Относительная погрешность	Относительная невязка
$4.3 \cdot 10^{-16}$	$6.35 \cdot 10^{-13}$

Тексты основных модулей

Program.cs

```
1 using eMP_1;
2
3 GridFactory gridFactory = new();
4 MFD mfd = new(gridFactory.CreateGrid(GridType.Irregular,
5     ↪ "InputParameters/grid(irregular).txt"), "InputParameters/boundaries.txt");
6 // MFD mfd = new(gridFactory.CreateGrid(GridType.Nested,
7     ↪ "InputParameters/grid(nested).txt"), "InputParameters/boundaries.txt");
8 // MFD mfd = new(gridFactory.CreateGrid(GridType.Regular,
9     ↪ "InputParameters/grid(regular).txt"), "InputParameters/boundaries.txt");
10 // mfd.SetTest(new FirstTest()); //  $x$ ,  $\lambda = 1$ ,  $\gamma = 0$ 
11 // mfd.SetTest(new SecondTest()); //  $x^2 - y$ ,  $\lambda = 0.5$ ,  $\gamma = 1$ 
12 // mfd.SetTest(new ThirdTest()); //  $3x^3 + 2y^3$ , 1 area:  $\lambda = \gamma = 0.5$ , 2 area:  $\lambda = \gamma = 2$ 
13 // mfd.SetTest(new FourthTest()); //  $\ln(x + y)$ ,  $\lambda = 1$ ,  $\gamma = 1$ 
14 // mfd.SetTest(new FifthTest()); //  $4x^4$ ,  $\lambda = 1$ ,  $\gamma = 0$ 
15 // mfd.SetTest(new SixthTest()); //  $4x^4 + 2y^4$ ,  $\lambda = 1$ ,  $\gamma = 0$ 
16 // mfd.SetTest(new SeventhTest()); //  $e^x + y$ ,  $\lambda = 1$ ,  $\gamma = 1$ 
17 // mfd.SetTest(new EighthTest()); //  $x^3 + y$ ,  $\lambda = 1$ ,  $\gamma = 0$ 
18 mfd.SetMethodSolvingSLAE(new GaussSeidel(1000, 1E-14, 1.17));
19 mfd.Compute();
```

MFD.cs

```
1 namespace eMP_1;
2
3 public enum PointType {
4     None,
5     Boundary,
6     Internal,
7     Dummy
8 }
9
10 public enum BoundaryType {
11     None,
12     Dirichlet,
13     Neumann,
14     Mixed
15 }
16
17 public enum GridType {
18     Regular,
19     Irregular,
20     Nested
21 }
22
23 public enum NormalType {
24     None,
25     LeftX,
26     RightX,
27     UpperY,
28     BottomY
29 }
30
31 public class MFD {
32     private readonly Grid _grid;
33     private DiagMatrix _matrix;
34     private ITest _test;
```

```

35 private ISolver _solver;
36 private readonly Boundary[] _boundaries;
37 private double[] _q;
38 private double[] _pr;
39 private readonly double _beta;
40 public double[] Weights
41     => _q;
42
43 public MFD(Grid grid, string boundaryPath) {
44     try {
45         using (var sr = new StreamReader(boundaryPath)) {
46             _beta = double.Parse(sr.ReadLine());
47             _boundaries = sr.ReadToEnd().Split("\n")
48                 .Select(str => Boundary.BoundaryParse(str)).ToArray();
49         }
50
51         _grid = grid;
52
53     } catch (Exception ex) {
54         Console.WriteLine(ex.Message);
55     }
56 }
57
58 public void SetTest(ITest test)
59     => _test = test;
60
61 public void SetMethodSolvingSLAE(ISolver solver)
62     => _solver = solver;
63
64 public void Compute() {
65     try {
66         if (_test is null)
67             throw new Exception("Set the test!");
68
69         if (_solver is null)
70             throw new Exception("Set the method solving SLAE!");
71
72         _grid.Build();
73         _grid.AssignBoundaryConditions(_boundaries);
74         Init();
75         BuildMatrix();
76         _q = _solver.Compute(_matrix, _pr);
77
78     } catch (Exception ex) {
79         Console.WriteLine(ex.Message);
80     }
81 }
82
83 private void Init() {
84     _matrix = new(_grid.Points.Count, (_grid.AllLinesX.Count >
85         ↪ _grid.AllLinesX.Count) ?
86     _grid.AllLinesX.Count - 2 : _grid.AllLinesY.Count - 2);
87     _pr = new double[_matrix.Size];
88     _q = new double[_matrix.Size];
89 }
90
91 private void BuildMatrix() {
92     double hx, hy, hix, hiy, hi;
93     double lambda, gamma;
94     double us, ubeta;

```



```

94     double leftDerivative, rightDerivative;
95     NormalType normalType;
96
97     for (int i = 0; i < _grid.Points.Count; i++) {
98         switch (_grid.Points[i].PointType) {
99             case PointType.Boundary:
100
101                 switch (_grid.Points[i].BoundaryType) {
102                     case BoundaryType.Dirichlet:
103
104                         _matrix.Diagnostics[i] = 1;
105                         _pr[i] = _test.U(_grid.Points[i]);
106
107                         break;
108
109                     case BoundaryType.Neumann:
110
111                         lambda = _grid.Areas[_grid.Points[i].AreaNumber].Item2;
112
113                         normalType = _grid.Normal(_grid.Points[i]);
114
115                         switch (normalType) {
116                             case NormalType.LeftX:
117
118                                 hi = _grid.AllLinesX[_grid.Points[i].I + 1] -
119                                     ↪ _grid.AllLinesX[_grid.Points[i].I];
120                                 _matrix.Diagnostics[i] = lambda / hi;
121                                 _matrix.Diagnostics[4][i] = -lambda / hi;
122                                 _pr[i] = -lambda *
123                                     ↪ RightDerivativeX(_grid.Points[i], hi);
124
125                                 break;
126
127                             case NormalType.BottomY:
128
129                                 hi = _grid.AllLinesY[_grid.Points[i].J + 1] -
130                                     ↪ _grid.AllLinesY[_grid.Points[i].J];
131                                 _matrix.Diagnostics[i] = lambda / hi;
132                                 _matrix.Diagnostics[3][i] = -lambda / hi;
133                                 _pr[i] = -lambda *
134                                     ↪ RightDerivativeY(_grid.Points[i], hi);
135
136                                 break;
137
138                             case NormalType.RightX:
139
140                                 hi = _grid.AllLinesX[_grid.Points[i].I] -
141                                     ↪ _grid.AllLinesX[_grid.Points[i].I - 1];
142                                 _matrix.Diagnostics[i] = lambda / hi;
143                                 _matrix.Diagnostics[2][i + _matrix.Indexes[2]] =
144                                     ↪ -lambda / hi;
145                                 _pr[i] = lambda *
146                                     ↪ LeftDerivativeX(_grid.Points[i], hi);
147
148                                 break;
149
150                             case NormalType.UpperY:
151
152                                 hi = _grid.AllLinesY[_grid.Points[i].J] -
153                                     ↪ _grid.AllLinesY[_grid.Points[i].J - 1];

```

```

146         _matrix.Diagnostics[0][i] = lambda / hi;
147         _matrix.Diagnostics[1][i + _matrix.Indexes[1]] =
148             ↪ -lambda / hi;
149         _pr[i] = lambda *
150             ↪ LeftDerivativeY(_grid.Points[i], hi);
151
152         break;
153
154     default:
155         throw new ArgumentOutOfRangeException(nameof(normalType),
156             ↪ alType),
157         $"This type of normal does not exist:
158             ↪ {normalType}");
159     }
160
161     break;
162
163     case BoundaryType.Mixed:
164
165         lambda = _grid.Areas[_grid.Points[i].AreaNumber].Item2;
166
167         normalType = _grid.Normal(_grid.Points[i]);
168         us = _test.U(_grid.Points[i]);
169
170         switch (normalType) {
171             case NormalType.LeftX:
172
173                 hi = _grid.AllLinesX[_grid.Points[i].I + 1] -
174                     ↪ _grid.AllLinesX[_grid.Points[i].I];
175                 rightDerivative =
176                     ↪ RightDerivativeX(_grid.Points[i], hi);
177                 ubeta = -lambda * rightDerivative / _beta + us;
178                 _matrix.Diagnostics[0][i] = lambda / hi + _beta;
179                 _matrix.Diagnostics[4][i] = -lambda / hi;
180                 _pr[i] = -lambda * rightDerivative + _beta * (us
181                     ↪ - ubeta) + _beta * ubeta;
182
183                 break;
184
185             case NormalType.BottomY:
186
187                 hi = _grid.AllLinesY[_grid.Points[i].J + 1] -
188                     ↪ _grid.AllLinesY[_grid.Points[i].J];
189                 rightDerivative =
190                     ↪ RightDerivativeY(_grid.Points[i], hi);
191                 ubeta = -lambda * rightDerivative / _beta + us;
192                 _matrix.Diagnostics[0][i] = lambda / hi + _beta;
193                 _matrix.Diagnostics[3][i] = -lambda / hi;
194                 _pr[i] = -lambda * rightDerivative + _beta * (us
195                     ↪ - ubeta) + _beta * ubeta;
196
197                 break;
198
199             case NormalType.RightX:
200
201                 hi = _grid.AllLinesX[_grid.Points[i].I] -
202                     ↪ _grid.AllLinesX[_grid.Points[i].I - 1];
203                 leftDerivative = LeftDerivativeX(_grid.Points[i],
204                     ↪ hi);
205                 ubeta = lambda * leftDerivative / _beta + us;

```

```

194         _matrix.Diags[0][i] = lambda / hi + _beta;
195         _matrix.Diags[2][i + _matrix.Indexes[2]] =
196             ↪ -lambda / hi;
197         _pr[i] = lambda * leftDerivative + _beta * (us -
198             ↪ ubeta) + _beta * ubeta;
199
200         break;
201
202     case NormalType.UpperY:
203
204         hi = _grid.AllLinesY[_grid.Points[i].J] -
205             ↪ _grid.AllLinesY[_grid.Points[i].J - 1];
206         leftDerivative = LeftDerivativeY(_grid.Points[i],
207             ↪ hi);
208         ubeta = lambda * leftDerivative / _beta + us;
209         _matrix.Diags[0][i] = lambda / hi + _beta;
210         _matrix.Diags[1][i + _matrix.Indexes[1]] =
211             ↪ -lambda / hi;
212         _pr[i] = lambda * leftDerivative + _beta * (us -
213             ↪ ubeta) + _beta * ubeta;
214
215         break;
216
217     default:
218         throw new ArgumentOutOfRangeException(nameof(normalType),
219             ↪ alType),
220         $"This type of normal does not exist:
221             ↪ {normalType}");
222     }
223
224     break;
225
226     default:
227         throw new
228             ↪ ArgumentOutOfRangeException(nameof(BoundaryType),
229             ↪ $"This type of boundary does not exist:
230             ↪ {_grid.Points[i].BoundaryType}");
231     }
232
233     break;
234
235     case PointType.Internal:
236
237         hx = _grid.AllLinesX[_grid.Points[i].I + 1] -
238             ↪ _grid.AllLinesX[_grid.Points[i].I];
239         hy = _grid.AllLinesY[_grid.Points[i].J + 1] -
240             ↪ _grid.AllLinesY[_grid.Points[i].J];
241
242         (lambda, gamma) =
243         (_grid.Areas[_grid.Points[i].AreaNumber].Item2,
244         ↪ _grid.Areas[_grid.Points[i].AreaNumber].Item3);
245
246         _pr[i] = _test.F(_grid.Points[i]);
247
248         if (_grid is RegularGrid) {
249             _matrix.Diags[0][i] = lambda * (2.0 / (hx * hx) + 2.0 / (hy *
250                 ↪ hy)) + gamma;
251             _matrix.Diags[3][i] = -lambda / (hy * hy);
252             _matrix.Diags[4][i] = -lambda / (hx * hx);
253         }
254     }

```

```

241         _matrix.Diagnostics[1][i + _matrix.Indexes[1]] = -lambda / (hy *
242             ↪ hy);
243         _matrix.Diagnostics[2][i + _matrix.Indexes[2]] = -lambda / (hx *
244             ↪ hx);
245     } else {
246         hix = _grid.AllLinesX[_grid.Points[i].I] -
247             ↪ _grid.AllLinesX[_grid.Points[i].I - 1];
248         hiy = _grid.AllLinesY[_grid.Points[i].J] -
249             ↪ _grid.AllLinesY[_grid.Points[i].J - 1];
250
251         _matrix.Diagnostics[0][i] = lambda * (2.0 / (hix * hx) + 2.0 / (hiy
252             ↪ * hy)) + gamma;
253         _matrix.Diagnostics[2][i + _matrix.Indexes[2]] = -lambda * 2.0 /
254             ↪ (hix * (hx + hix));
255         _matrix.Diagnostics[1][i + _matrix.Indexes[1]] = -lambda * 2.0 /
256             ↪ (hiy * (hy + hiy));
257         _matrix.Diagnostics[4][i] = -lambda * 2.0 / (hx * (hx + hix));
258         _matrix.Diagnostics[3][i] = -lambda * 2.0 / (hy * (hy + hiy));
259     }
260
261     break;
262
263     case PointType.Dummy:
264
265         _matrix.Diagnostics[0][i] = 1;
266         _pr[i] = 0;
267
268         break;
269
270     default:
271         throw new ArgumentOutOfRangeException(nameof(PointType),
272             ↪ $"This type of point does not exist:
273             ↪ {_grid.Points[i].PointType}");
274 }
275 }
276
277 private double LeftDerivativeX(Point2D point, double h)
278     => (_test.U(point) - _test.U(point - (h, 0))) / h;
279
280 private double LeftDerivativeY(Point2D point, double h)
281     => (_test.U(point) - _test.U(point - (0, h))) / h;
282
283 private double RightDerivativeX(Point2D point, double h)
284     => (_test.U(point + (h, 0)) - _test.U(point)) / h;
285
286 private double RightDerivativeY(Point2D point, double h)
287     => (_test.U(point + (0, h)) - _test.U(point)) / h;
288 }

```

GridFactory.cs

```

1  namespace eMP_1;
2
3  public interface IFactory {
4      public Grid CreateGrid(GridType gridType, string path);
5  }
6
7  public class GridFactory : IFactory {
8      public Grid CreateGrid(GridType gridType, string path) {

```

```

9         return gridType switch {
10             GridType.Regular => new RegularGrid(path),
11
12             GridType.Irregular => new IrregularGrid(path),
13
14             GridType.Nested => new NestedGrid(path),
15
16             _ => throw new ArgumentOutOfRangeException(nameof(gridType),
17                 $"This type of grid does not exist: {gridType}")
18         };
19     }
20 }

```

Grid.cs

```

1 namespace eMP_1;
2
3 public abstract class Grid {
4     public abstract ImmutableArray<double> LinesX { get; init; }
5     public abstract ImmutableArray<double> LinesY { get; init; }
6     public abstract ImmutableList<double> AllLinesX { get; }
7     public abstract ImmutableList<double> AllLinesY { get; }
8     public abstract ImmutableList<Point2D> Points { get; }
9     public abstract ImmutableArray<(int, double, double, int, int, int, int)> Areas {
10         ↪ get; }
11
12     public abstract void Build();
13
14     public NormalType Normal(Point2D point) {
15         if (point.PointType != PointType.Boundary) {
16             throw new Exception("To determine the normal to the boundary, the point
17                 ↪ needs the boundary!");
18         }
19
20         var normalType = NormalType.None;
21
22         foreach (var area in Areas) {
23             normalType = point switch {
24                 _ when point.X == LinesX[area.Item4] => NormalType.LeftX,
25                 _ when point.Y == LinesY[area.Item6] => NormalType.BottomY,
26                 _ when point.X == LinesX[area.Item5] => NormalType.RightX,
27                 _ when point.Y == LinesY[area.Item7] => NormalType.UpperY,
28
29                 _ => NormalType.None,
30             };
31
32             if (normalType != NormalType.None) {
33                 break;
34             }
35         }
36
37         return normalType;
38     }
39
40     protected PointType PointTypesWithoutInternalCheck(double x, double y) {
41         foreach (var area in Areas) {
42             if (x >= LinesX[area.Item4] && x <= LinesX[area.Item5] && y >=
43                 ↪ LinesY[area.Item6] && y <= LinesY[area.Item7]) {
44                 return PointType.Boundary;
45             }
46         }
47     }
48 }

```

```

43     }
44
45     return PointType.Dummy;
46 }
47
48 protected void InternalCheck() {
49     foreach (var point in Points.Where(point => point.PointType !=
50         ↳ PointType.Dummy)) {
51         if (Points.Any(1stPoint => 1stPoint.I == point.I - 1 && 1stPoint.J ==
52             ↳ point.J && 1stPoint.PointType != PointType.Dummy) &&
53             Points.Any(1stPoint => 1stPoint.I == point.I + 1 && 1stPoint.J ==
54                 ↳ point.J && 1stPoint.PointType != PointType.Dummy) &&
55             Points.Any(1stPoint => 1stPoint.J == point.J - 1 && 1stPoint.I ==
56                 ↳ point.I && 1stPoint.PointType != PointType.Dummy) &&
57             Points.Any(1stPoint => 1stPoint.J == point.J + 1 && 1stPoint.I ==
58                 ↳ point.I && 1stPoint.PointType != PointType.Dummy)) {
59                 point.PointType = PointType.Internal;
60         }
61     }
62 }
63
64 protected void SetAreaNumber() {
65     for (int i = 0; i < Points.Count; i++) {
66         for (int iArea = 0; iArea < Areas.Length; iArea++) {
67             if (Points[i].X >= Areas[iArea].Item4 && Points[i].X <=
68                 ↳ Areas[iArea].Item5 &&
69                 Points[i].Y >= Areas[iArea].Item6 && Points[i].Y <=
70                     ↳ Areas[iArea].Item7) {
71                 Points[i].AreaNumber = Areas[iArea].Item1;
72             }
73         }
74     }
75 }
76
77 public void AssignBoundaryConditions(Boundary[] boundaries) {
78     foreach (var point in Points.Where(point => point.PointType ==
79         ↳ PointType.Boundary)) {
80         for (int k = 0; k < boundaries.Length; k++) {
81             if (point.X >= LinesX[boundaries[k].X1] && point.X <=
82                 ↳ LinesX[boundaries[k].X2]
83             && point.Y >= LinesY[boundaries[k].Y1] && point.Y <=
84                 ↳ LinesY[boundaries[k].Y2]) {
85                 point.BoundaryType = boundaries[k].BoundaryType;
86                 break;
87             }
88         }
89     }
90 }
91
92 protected void WriteToFilePoints() {
93     using (var sw = new StreamWriter("points/boundaryPoints.txt")) {
94         Points.ForEach(x => { if (x.PointType == PointType.Boundary)
95             ↳ sw.WriteLine(x); });
96     }
97
98     using (var sw = new StreamWriter("points/internalPoints.txt")) {
99         Points.ForEach(x => { if (x.PointType == PointType.Internal)
100             ↳ sw.WriteLine(x); });
101     }
102 }

```

```

91         using (var sw = new StreamWriter("points/dummyPoints.txt")) {
92             Points.ForEach(x => { if (x.PointType == PointType.Dummy)
93                 ↪ sw.WriteLine(x); });
94         }
95     }

```

RegularGrid.cs

```

1  namespace eMP_1;
2
3  public class RegularGrid : Grid {
4      private List<double> _allLinesX;
5      private List<double> _allLinesY;
6      private readonly List<Point2D> _points;
7      private readonly (int, double, double, int, int, int, int)[] _areas;
8      public override ImmutableArray<double> LinesX { get; init; }
9      public override ImmutableArray<double> LinesY { get; init; }
10     public override ImmutableList<double> AllLinesX
11         => _allLinesX.ToImmutableList();
12     public override ImmutableList<double> AllLinesY
13         => _allLinesY.ToImmutableList();
14     public override ImmutableList<Point2D> Points
15         => _points.ToImmutableList();
16     public override ImmutableArray<(int, double, double, int, int, int, int)> Areas
17         => _areas.ToImmutableArray();
18     public int SplitsX { get; init; }
19     public int SplitsY { get; init; }
20
21     public RegularGrid(string path) {
22         try {
23             using (var sr = new StreamReader(path)) {
24                 LinesX = sr.ReadLine().Split().Select(value =>
25                     ↪ double.Parse(value)).ToImmutableArray();
26                 LinesY = sr.ReadLine().Split().Select(value =>
27                     ↪ double.Parse(value)).ToImmutableArray();
28                 SplitsX = int.Parse(sr.ReadLine());
29                 SplitsY = int.Parse(sr.ReadLine());
30                 _areas = sr.ReadToEnd().Split("\n").Select(row => row.Split())
31                     .Select(value => (int.Parse(value[0]), double.Parse(value[1]),
32                     ↪ double.Parse(value[2]),
33                     int.Parse(value[3]), int.Parse(value[4]), int.Parse(value[5]),
34                     ↪ int.Parse(value[6]))).ToArray();
35             }
36
37             _allLinesX = new();
38             _allLinesY = new();
39             _points = new();
40         } catch (Exception ex) {
41             Console.WriteLine(ex.Message);
42         }
43     }
44
45     public override void Build() {
46         double h;
47         double lenght = LinesX.Last() - LinesX.First();
48
49         h = lenght / SplitsX;
50
51         _allLinesX.Add(LinesX.First());

```

```

48     while (Math.Round(_allLinesX.Last() + h, 1) < LinesX.Last())
49         _allLinesX.Add(_allLinesX.Last() + h);
50
51     _allLinesX = _allLinesX.Union(LinesX).OrderBy(value => value).ToList();
52
53     lenght = LinesY.Last() - LinesY.First();
54
55     h = lenght / SplitsY;
56
57     _allLinesY.Add(LinesY.First());
58
59     while (Math.Round(_allLinesY.Last() + h, 1) < LinesY.Last())
60         _allLinesY.Add(_allLinesY.Last() + h);
61
62     _allLinesY = _allLinesY.Union(LinesY).OrderBy(value => value).ToList();
63
64     for (int i = 0; i < _allLinesX.Count; i++)
65         for (int j = 0; j < _allLinesY.Count; j++)
66             _points.Add(new(_allLinesX[i], _allLinesY[j], i, j,
67                 PointTypesWithoutInternalCheck(_allLinesX[i], _allLinesY[j])));
68
69     InternalCheck();
70     SetAreaNumber();
71     WriteToFilePoints();
72 }
73 }
74

```

IrregularGrid.cs

```

1  namespace EMP_1;
2
3  public class IrregularGrid : Grid {
4      private readonly List<double> _allLinesX;
5      private readonly List<double> _allLinesY;
6      private readonly List<Point2D> _points;
7      private readonly int[] _splitsX;
8      private readonly int[] _splitsY;
9      private readonly double[] _kX;
10     private readonly double[] _kY;
11     private readonly (int, double, double, int, int, int, int)[] _areas;
12     public override ImmutableArray<double> LinesX { get; init; }
13     public override ImmutableArray<double> LinesY { get; init; }
14     public override ImmutableList<Point2D> Points
15         => _points.ToImmutableList();
16     public override ImmutableList<double> AllLinesX
17         => _allLinesX.ToImmutableList();
18     public override ImmutableList<double> AllLinesY
19         => _allLinesY.ToImmutableList();
20     public override ImmutableArray<(int, double, double, int, int, int, int)> Areas
21         => _areas.ToImmutableArray();
22     public ImmutableArray<int> SplitsX
23         => _splitsX.ToImmutableArray();
24     public ImmutableArray<int> SplitsY
25         => _splitsY.ToImmutableArray();
26     public ImmutableArray<double> KX
27         => _kX.ToImmutableArray();
28     public ImmutableArray<double> KY
29         => _kY.ToImmutableArray();
30

```



```

31 public IrregularGrid(string path) {
32     try {
33         using (var sr = new StreamReader(path)) {
34             LinesX = sr.ReadLine().Split().Select(value =>
35                 ↪ double.Parse(value)).ToImmutableArray();
36             LinesY = sr.ReadLine().Split().Select(value =>
37                 ↪ double.Parse(value)).ToImmutableArray();
38             _splitsX = sr.ReadLine().Split().Select(value =>
39                 ↪ int.Parse(value)).ToArray();
40             _splitsY = sr.ReadLine().Split().Select(value =>
41                 ↪ int.Parse(value)).ToArray();
42             _kX = sr.ReadLine().Split().Select(value =>
43                 ↪ double.Parse(value)).ToArray();
44             _kY = sr.ReadLine().Split().Select(value =>
45                 ↪ double.Parse(value)).ToArray();
46             _areas = sr.ReadToEnd().Split("\n").Select(row => row.Split())
47                 .Select(value => (int.Parse(value[0]), double.Parse(value[1]),
48                 ↪ double.Parse(value[2]),
49                 ↪ int.Parse(value[3]), int.Parse(value[4]), int.Parse(value[5]),
50                 ↪ int.Parse(value[6]))).ToArray();
51         }
52         _allLinesX = new();
53         _allLinesY = new();
54         _points = new();
55     } catch (Exception ex) {
56         Console.WriteLine(ex.Message);
57     }
58 }
59
60 public override void Build() {
61     for (int i = 0; i < LinesX.Length - 1; i++) {
62         double h;
63         double sum = 0;
64         double lenght = LinesX[i + 1] - LinesX[i];
65
66         for (int k = 0; k < _splitsX[i]; k++)
67             sum += Math.Pow(_kX[i], k);
68
69         h = lenght / sum;
70
71         _allLinesX.Add(LinesX[i]);
72
73         while (Math.Round(_allLinesX.Last() + h, 1) < LinesX[i + 1]) {
74             _allLinesX.Add(_allLinesX.Last() + h);
75
76             h *= _kX[i];
77         }
78     }
79
80     _allLinesX.Add(LinesX.Last());
81
82     for (int i = 0; i < LinesY.Length - 1; i++) {
83         double h;
84         double sum = 0;
85         double lenght = LinesY[i + 1] - LinesY[i];
86
87         for (int k = 0; k < _splitsY[i]; k++)

```

```

83         sum += Math.Pow(_kY[i], k);
84
85         h = lenght / sum;
86
87         _allLinesY.Add(LinesY[i]);
88
89         while (Math.Round(_allLinesY.Last() + h, 1) < LinesY[i + 1]) {
90             _allLinesY.Add(_allLinesY.Last() + h);
91
92             h *= _kY[i];
93         }
94     }
95
96     _allLinesY.Add(LinesY.Last());
97
98     for (int i = 0; i < _allLinesX.Count; i++)
99         for (int j = 0; j < _allLinesY.Count; j++)
100             _points.Add(new(_allLinesX[i], _allLinesY[j], i, j,
101                 PointTypesWithoutInternalCheck(_allLinesX[i], _allLinesY[j])));
102
103     InternalCheck();
104     SetAreaNumber();
105     WriteToFilePoints();
106 }
107 }

```

NestedGrid.cs

```

1  namespace eMP_1;
2
3  public class NestedGrid : Grid {
4      private readonly List<double> _allLinesX;
5      private readonly List<double> _allLinesY;
6      private readonly List<Point2D> _points;
7      private readonly int[] _splitsX;
8      private readonly int[] _splitsY;
9      private readonly double[] _kX;
10     private readonly double[] _kY;
11     private readonly (int, double, double, int, int, int, int)[] _areas;
12     public override ImmutableArray<double> LinesX { get; init; }
13     public override ImmutableArray<double> LinesY { get; init; }
14     public override ImmutableList<Point2D> Points
15         => _points.ToImmutableList();
16     public override ImmutableList<double> AllLinesX
17         => _allLinesX.ToImmutableList();
18     public override ImmutableList<double> AllLinesY
19         => _allLinesY.ToImmutableList();
20     public override ImmutableArray<(int, double, double, int, int, int, int)> Areas
21         => _areas.ToImmutableArray();
22     public ImmutableArray<int> SplitsX => _splitsX.ToImmutableArray();
23     public ImmutableArray<int> SplitsY => _splitsY.ToImmutableArray();
24     public ImmutableArray<double> KX => _kX.ToImmutableArray();
25     public ImmutableArray<double> KY => _kY.ToImmutableArray();
26     public int NumberNesting { get; init; }
27
28     public NestedGrid(string path) {
29         try {
30             using (var sr = new StreamReader(path)) {
31                 LinesX = sr.ReadLine().Split().Select(value =>
32                     double.Parse(value)).ToImmutableArray();

```

```

31         LinesY = sr.ReadLine().Split().Select(value =>
32             ↪ double.Parse(value)).ToImmutableArray();
33         _splitsX = sr.ReadLine().Split().Select(value =>
34             ↪ int.Parse(value)).ToArray();
35         _splitsY = sr.ReadLine().Split().Select(value =>
36             ↪ int.Parse(value)).ToArray();
37         _kX = sr.ReadLine().Split().Select(value =>
38             ↪ double.Parse(value)).ToArray();
39         _kY = sr.ReadLine().Split().Select(value =>
40             ↪ double.Parse(value)).ToArray();
41         NumberNesting = int.Parse(sr.ReadLine());
42         _areas = sr.ReadToEnd().Split("\n").Select(row => row.Split())
43         .Select(value => (int.Parse(value[0]), double.Parse(value[1]),
44             ↪ double.Parse(value[2]),
45             int.Parse(value[3]), int.Parse(value[4]), int.Parse(value[5]),
46             ↪ int.Parse(value[6]))).ToArray();
47     }
48
49     _allLinesX = new();
50     _allLinesY = new();
51     _points = new();
52 } catch (Exception ex) {
53     Console.WriteLine(ex.Message);
54 }
55
56 public override void Build() {
57     for (int i = 0; i < NumberNesting; i++)
58         for (int j = 0; j < _kX.Length; j++)
59             _kX[j] = Math.Sqrt(_kX[j]);
60
61     for (int i = 0; i < LinesX.Length - 1; i++) {
62         double h;
63         double sum = 0;
64         double lenght = LinesX[i + 1] - LinesX[i];
65
66         for (int k = 0; k < _splitsX[i]; k++)
67             sum += Math.Pow(_kX[i], k);
68
69         h = lenght / sum;
70
71         _allLinesX.Add(LinesX[i]);
72
73         while (Math.Round(_allLinesX.Last() + h, 1) < LinesX[i + 1]) {
74             _allLinesX.Add(_allLinesX.Last() + h);
75
76             h *= _kX[i];
77         }
78     }
79
80     _allLinesX.Add(LinesX.Last());
81
82     for (int i = 0; i < NumberNesting; i++)
83         for (int j = 0; j < _kY.Length; j++)
84             _kY[j] = Math.Sqrt(_kY[j]);
85
86     for (int i = 0; i < LinesY.Length - 1; i++) {
87         double h;
88         double sum = 0;

```

```

84         double lenght = LinesY[i + 1] - LinesY[i];
85
86         for (int k = 0; k < _splitsY[i]; k++)
87             sum += Math.Pow(_kY[i], k);
88
89         h = lenght / sum;
90
91         _allLinesY.Add(LinesY[i]);
92
93         while (Math.Round(_allLinesY.Last() + h, 1) < LinesY[i + 1]) {
94             _allLinesY.Add(_allLinesY.Last() + h);
95
96             h *= _kY[i];
97         }
98     }
99
100     _allLinesY.Add(LinesY.Last());
101
102     for (int i = 0; i < _allLinesX.Count; i++)
103         for (int j = 0; j < _allLinesY.Count; j++)
104             _points.Add(new(_allLinesX[i], _allLinesY[j], i, j,
105                 PointTypesWithoutInternalCheck(_allLinesX[i], _allLinesY[j])));
106
107     InternalCheck();
108     SetAreaNumber();
109     WriteToFilePoints();
110 }
111 }

```

DiagMatrix.cs

```

1  namespace eMP_1;
2
3  public class DiagMatrix {
4      public double[][] Diags { get; set; }
5      public int[] Indexes { get; init; }
6      public int Size { get; init; }
7      public int ZeroDiags { get; init; }
8
9      public DiagMatrix(int countPoints, int zeroDiags) {
10         Size = countPoints;
11         ZeroDiags = zeroDiags;
12         Diags = new double[5][];
13         Diags[0] = new double[countPoints];
14         Diags[1] = new double[countPoints - 1];
15         Diags[2] = new double[countPoints - zeroDiags - 2];
16         Diags[3] = new double[countPoints - 1];
17         Diags[4] = new double[countPoints - zeroDiags - 2];
18         Indexes = new int[] { 0, -1, -2 - zeroDiags, 1, 2 + zeroDiags };
19     }
20 }

```

Solver.cs

```

1  namespace eMP_1;
2
3  public interface ISolver {
4      public int MaxIters { get; init; }
5      public double Eps { get; init; }

```

```

6     public double W { get; init; }
7
8     public double[] Compute(DiagMatrix diagMatrix, double[] pr);
9 }
10
11 public record GaussSeidel(int MaxIters, double Eps, double W) : ISolver {
12     public double[] Compute(DiagMatrix diagMatrix, double[] pr) {
13         double[] qk = new double[diagMatrix.Size];
14         double[] qk1 = new double[diagMatrix.Size];
15         double[] residual = new double[diagMatrix.Size];
16         double prNorm = pr.Norm();
17
18         for (int i = 0; i < MaxIters; i++) {
19             for (int k = 0; k < diagMatrix.Size; k++) {
20                 double fstSum = MultLine(diagMatrix, k, qk1, 1);
21                 double scdSum = MultLine(diagMatrix, k, qk, 2);
22
23                 residual[k] = pr[k] - (fstSum + scdSum);
24                 qk1[k] = qk[k] + W * residual[k] / diagMatrix.Diags[0][k];
25             }
26
27             qk1.Copy(qk);
28             qk1.Fill(0);
29
30             if (residual.Norm() / prNorm < Eps)
31                 break;
32         }
33
34         return qk;
35     }
36
37     private static double MultLine(DiagMatrix diagMatrix, int i, double[] vector, int
↵ method) {
38         double sum = 0;
39
40         if (method == 0 || method == 1) {
41             if (i > 0) {
42                 sum += diagMatrix.Diags[1][i - 1] * vector[i - 1];
43
44                 if (i > diagMatrix.ZeroDiags + 1)
45                     sum += diagMatrix.Diags[2][i - diagMatrix.ZeroDiags - 2] *
↵ vector[i - diagMatrix.ZeroDiags - 2];
46             }
47         }
48
49         if (method == 0 || method == 2) {
50             sum += diagMatrix.Diags[0][i] * vector[i];
51
52             if (i < diagMatrix.Size - 1) {
53                 sum += diagMatrix.Diags[3][i] * vector[i + 1];
54
55                 if (i < diagMatrix.Size - diagMatrix.ZeroDiags - 2)
56                     sum += diagMatrix.Diags[4][i] * vector[i + diagMatrix.ZeroDiags +
↵ 2];
57             }
58         }
59
60         return sum;
61     }
62 }

```

Boundary.cs

```
1 namespace eMP_1;
2
3 public readonly record struct Boundary(BoundaryType BoundaryType, int X1, int X2, int
   ↪ Y1, int Y2) {
4     public static Boundary BoundaryParse(string boundaryStr) {
5         var data = boundaryStr.Split();
6         Boundary boundary = new((BoundaryType)Enum.Parse(typeof(BoundaryType),
   ↪ data[0]),
7         int.Parse(data[1]), int.Parse(data[2]), int.Parse(data[3]),
   ↪ int.Parse(data[4]));
8
9         return boundary;
10    }
11 }
```

Point2D.cs

```
1 namespace eMP_1;
2
3 public class Point2D {
4     public double X { get; init; }
5     public double Y { get; init; }
6     public int I { get; init; }
7     public int J { get; init; }
8     public PointType PointType { get; set; }
9     public BoundaryType BoundaryType { get; set; } = BoundaryType.None;
10    public int AreaNumber { get; set; }
11
12    public Point2D(double x, double y, int i, int j, PointType pointType) {
13        X = x;
14        Y = y;
15        I = i;
16        J = j;
17        PointType = pointType;
18    }
19
20    public static Point2D Parse(string pointStr) {
21        var data = pointStr.Split();
22        Point2D point = new(double.Parse(data[0]), double.Parse(data[1]),
23        int.Parse(data[2]), int.Parse(data[3]),
   ↪ (PointType)Enum.Parse(typeof(PointType), data[4]));
24
25        return point;
26    }
27
28    public static Point2D operator +(Point2D point, (double, double) value)
   => new(point.X + value.Item1, point.Y + value.Item2, point.I, point.J,
   ↪ point.PointType);
29
30
31    public static Point2D operator -(Point2D point, (double, double) value)
   => new(point.X - value.Item1, point.Y - value.Item2, point.I, point.J,
   ↪ point.PointType);
32
33
34    public override string ToString()
35        => $"{X} {Y}";
36 }
```

Tests.cs

```
1 namespace eMP_1;
2
3 public interface ITest {
4     public double U(Point2D point);
5     public double F(Point2D point);
6 }
7
8 public class FirstTest : ITest {
9     public double U(Point2D point)
10         => point.X;
11
12     public double F(Point2D point)
13         => 0;
14 }
15
16 public class SecondTest : ITest {
17     public double U(Point2D point)
18         => point.X * point.X - point.Y;
19
20     public double F(Point2D point)
21         => -1 + point.X * point.X - point.Y;
22 }
23
24 public class ThirdTest : ITest {
25     public double U(Point2D point)
26         => 3 * point.X * point.X * point.X + 2 * point.Y * point.Y * point.Y;
27
28     public double F(Point2D point)
29         => (point.AreaNumber == 0) ? -9 * point.X - 6 * point.Y + 0.5 *
30         (3 * point.X * point.X * point.X + 2 * point.Y * point.Y * point.Y) :
31         -36 * point.X - 24 * point.Y + 2 * (3 * point.X * point.X * point.X + 2 *
32         point.Y * point.Y * point.Y);
33 }
34
35 public class FourthTest : ITest {
36     public double U(Point2D point)
37         => Math.Log(point.X + point.Y);
38
39     public double F(Point2D point)
40         => 2 / ((point.X + point.Y) * (point.X + point.Y));
41 }
42
43 public class FifthTest : ITest {
44     public double U(Point2D point)
45         => 4 * point.X * point.X * point.X * point.X;
46     public double F(Point2D point)
47         => -48 * point.X * point.X;
48 }
49
50 public class SixthTest : ITest {
51     public double U(Point2D point)
52         => 4 * point.X * point.X * point.X * point.X + 2 * point.Y * point.Y *
53         point.Y * point.Y;
54
55     public double F(Point2D point)
56         => -48 * point.X * point.X - 24 * point.Y * point.Y;
```

```

57 public class SeventhTest : ITest {
58     public double U(Point2D point)
59         => Math.Exp(point.X) + point.Y;
60
61     public double F(Point2D point)
62         => point.Y;
63 }
64
65 public class EighthTest : ITest {
66     public double U(Point2D point)
67         => point.X * point.X * point.X + point.Y;
68
69     public double F(Point2D point)
70         => -6 * point.X;
71 }

```

Array1DExtension.cs

```

1  namespace eMP_1;
2
3  public static class Array1DExtension {
4      public static double Norm(this double[] array) {
5          double result = 0;
6
7          for (int i = 0; i < array.Length; i++)
8              result += array[i] * array[i];
9
10         return Math.Sqrt(result);
11     }
12
13     public static void Fill(this double[] array, double value) {
14         for (int i = 0; i < array.Length; i++)
15             array[i] = value;
16     }
17
18     public static void Copy(this double[] source, double[] destination) {
19         for (int i = 0; i < source.Length; i++)
20             destination[i] = source[i];
21     }
22 }

```