



МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

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

НГТУ



НЭТИ

Кафедра прикладной математики и информатики

Лабораторная работа №1

по дисциплине «Уравнения математической физики»

Решение эллиптических краевых задач методом конечных разностей



ФПМИ

Группа	ПМ-92
Студенты	БЕГИЧЕВ АЛЕКСАНДР ШИШКИН НИКИТА
Преподаватели	ЗАДОРОВЫЙ А.Г. ПЕРСОВА М.Г. ПАТРУШЕВ И. И.
Дата	1.03.2022

Новосибирск

Цель работы

Разработать программу для решения эллиптической краевой задачи методом конечных разностей. Протестировать программу и численно оценить порядок аппроксимации.

Вариант: Область имеет L-образную форму. Предусмотреть учет первых и третьих краевых.

Теоретическая часть

Метод конечных разностей

Метод конечных разностей основан на разложении функции нескольких независимых переменных в окрестности заданной точки в ряд Тейлора:

$$\begin{aligned} u(x_1 + h_1, \dots, x_n + h_n) = & u(x_1, \dots, x_n) + \sum_{j=1}^n h_j \frac{\partial}{\partial x_j} u(x_1, \dots, x_n) + \\ & \frac{1}{2} \left(\sum_{j=1}^n h_j \frac{\partial}{\partial x_j} \right)^2 u(x_1, \dots, x_n) + \dots + \frac{1}{k!} \left(\sum_{j=1}^n h_j \frac{\partial}{\partial x_j} \right)^k u(x_1, \dots, x_n) + \\ & \frac{1}{(k+1)!} \left(\sum_{j=1}^n h_j \frac{\partial}{\partial x_j} \right)^{k+1} u(\xi_1, \dots, \xi_n), \end{aligned} \quad (1)$$

где h_j - произвольные приращения соответствующих аргументов, $\xi_j \in [x_j, x_j + h_j]$, функция $u(x_1, \dots, x_n)$ обладает ограниченными производными до $(k+1)$ -го порядка включительно.

При использовании двух слагаемых при разложении функции в ряд Тейлора (1) производные первого порядка могут быть аппроксимированы следующими конечными разностями первого порядка:

$$\nabla_h^+ u_i = \frac{u_{i+1} - u_i}{h_i}, \quad (2)$$

$$\nabla_h^- u_i = \frac{u_i - u_{i-1}}{h_{i-1}}, \quad (3)$$

$$\bar{\nabla}_h u_i = \frac{u_{i+1} - u_{i-1}}{h_i + h_{i-1}}, \quad (4)$$

где $\nabla_h^+ u_i$ - правая разность, $\nabla_h^- u_i$ - левая разность, $\bar{\nabla}_h u_i$ - двусторонняя разность первого порядка, $u_i = u(x_{i-1})$, $u_i = u(x_i)$, $u_{i+1} = u(x_{i+1})$.

Через конечные разности первого порядка рекуррентно могут быть определены разности второго и более высокого порядка, аппроксимирующие различные производные. На неравномерной сетке производная второго порядка может быть получена следующим образом:

$$\mathbb{V}_h u_i = \frac{2u_{i-1}}{h_{i-1}(h_i + h_{i-1})} - \frac{2u_i}{h_{i-1} h_i} + \frac{2u_{i+1}}{h_i(h_i + h_{i-1})}, \quad (5)$$

с погрешностью порядка $O(h)$.

Если сетка равномерная, то

$$\mathbb{V}_h u_i = \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2}, \quad (6)$$

и погрешность аппроксимации имеет уже второй порядок, если функция обладает ограниченной производной четвертого порядка.

Пусть область Ω двумерная и определена прямоугольная сетка Ω_h как совокупность точек $(x_1, y_1), \dots, (x_n, y_1), (x_1, y_2), \dots, (x_n, y_2), (x_1, y_m), \dots, (x_n, y_m)$. Тогда для двумерного оператора Лапласа

$$\nabla u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

дискретный аналог на неравномерной прямоугольной сетке может быть определен пятиточечным разностным выражением

$$\begin{aligned} \nabla_h u_{i,j} = & \frac{2u_{i-1,j}}{h_{i-1}^x(h_i^x + h_{i-1}^x)} + \frac{2u_{i,j-1}}{h_{j-1}^y(h_j^y + h_{j-1}^y)} + \frac{2u_{i+1,j}}{h_i^x(h_i^x + h_{i-1}^x)} + \\ & \frac{2u_{i,j+1}}{h_j^y(h_j^y + h_{j-1}^y)} - \left(\frac{2}{h_{i-1}^x h_i^x} + \frac{2}{h_{j-1}^y h_j^y} \right) u_{i,j}. \end{aligned} \quad (7)$$

На равномерной сетке пятиточечный разностный оператор Лапласа выглядит следующим образом

$$\nabla_h u_{i,j} = \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h_x^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h_y^2} \quad (8)$$

и имеет второй порядок погрешности.

Учет краевых условий первого рода

Первые краевые условия записываются в виде : $u|_{S_1} = u_g$.

Для узлов, расположенных на границе S_1 , на которых заданы краевые условия первого рода, соответствующие разностные уравнения заменяются соотношениями точно передающими краевые условия, т.е. диагональные элементы матрицы, соответствующие этим узлам заменяются на 1, а соответствующий элемент вектора правой части заменяется на значение u_g функции в этом узле.

Учет краевых условий второго и третьего рода

Вторые и третьи краевые условия записываются в виде:

$$\begin{aligned} \lambda \frac{\partial u}{\partial n} \Big|_{S_2} &= \theta, \\ \lambda \frac{\partial u}{\partial n} \Big|_{S_3} + \beta \left(u|_{S_3} - u_\beta \right) &= 0. \end{aligned}$$

Если расчетная область представляет собой прямоугольник со сторонами, параллельными координатным осям, то направление нормали к границе S_2 и S_3 , на которых заданы краевые условия второго и третьего рода, совпадает с одной из координатных линий, и тогда методы аппроксимации производной по нормали $\frac{\partial u}{\partial n}$ (которая в этом случае будет равна либо $\pm \frac{\partial u}{\partial x}$, либо $\pm \frac{\partial u}{\partial y}$) сводятся к одномерным (2) – (4).

Практическая часть

1. Построить прямоугольную сетку в области в соответствии с заданием. Допускается использовать фиктивные узлы для сохранения регулярной структуры.
2. Выполнить конечноразностную аппроксимацию исходного уравнения в соответствии с заданием. Получить формулы для вычисления компонент матрицы **A** и вектора правой части **b**.
3. Реализовать программу решения двумерной эллиптической задачи методом конечных разностей.
4. Протестировать разработанные программы на полиномах соответствующей степени.
5. Провести исследования порядка аппроксимации реализованных методов для различных задач с неполиномиальными решениями. Сравнить полученный порядок аппроксимации с теоретическим.

Описание программы

Программа состоит из нескольких модулей:

- Класс MFD, в котором происходят основные вычислительные операции.
- Класс DiagMatrix для представления пятидиагональной матрицы.
- Класс Point2D для представления двумерной точки.
- Класс GridFactory, в котором реализован фабричный метод для генерации различных сеток.
- Абстрактный класс Grid и его классы-наследники RegularGrid, IrregularGrid и NestedGrid для представления сеток.
- Структура Boundary для хранения краевых условий.
- Интерфейс ISolver для решения СЛАУ и класс GaussSeidel, который его реализует.
- Интерфейс ITest для тестирования и классы FirstTest, SecondTest, ThirdTest, FourthTest, FifthTest, SixthTest, SeventhTest, EighthTest, которые его реализуют.
- Статичный класс Array1DExtension для методов расширения одномерного массива.
- Подпрограмма на Python для отрисовки сетки.

Тестирование

Первый тест

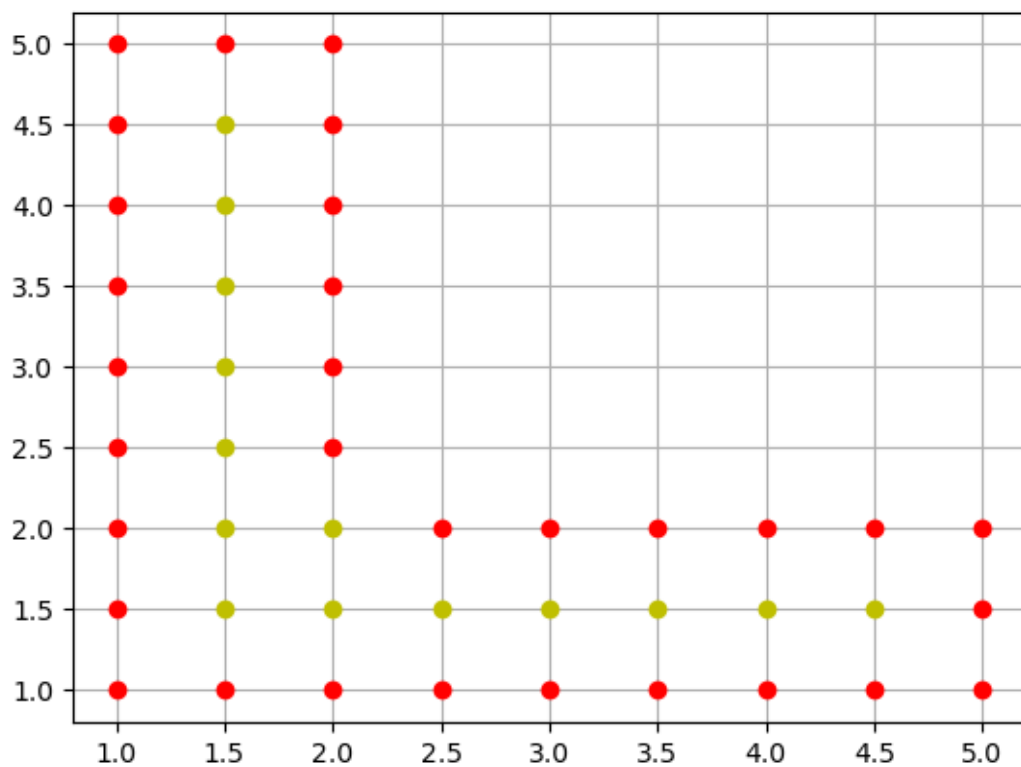
Функция: x

Правая часть: 0,

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

Сетка: равномерная,

Заданы краевые условия 1-го рода на всех границах.



х у	Точное	Численное	Вектор погрешности	Погрешность
1.500 1.500	1.5	1.50000000000000013	1.33E-15	4.02E-16
1.500 2.000	1.5	1.4999999999999998	2.22E-16	
1.500 2.500	1.5	1.50000000000000004	4.44E-16	
1.500 3.000	1.5	1.5	0	
1.500 3.500	1.5	1.5	0	
1.500 4.000	1.5	1.5	0	
1.500 4.500	1.5	1.5	0	
2.000 1.500	2	2	0	
2.000 2.000	2	1.9999999999999998	2.22E-16	
2.500 1.500	2.5	2.50000000000000004	4.44E-16	
3.000 1.500	3	3	0	
3.500 1.500	3.5	3.5	0	
4.000 1.500	4	4	0	
4.500 1.500	4.5	4.5	0	

Второй тест

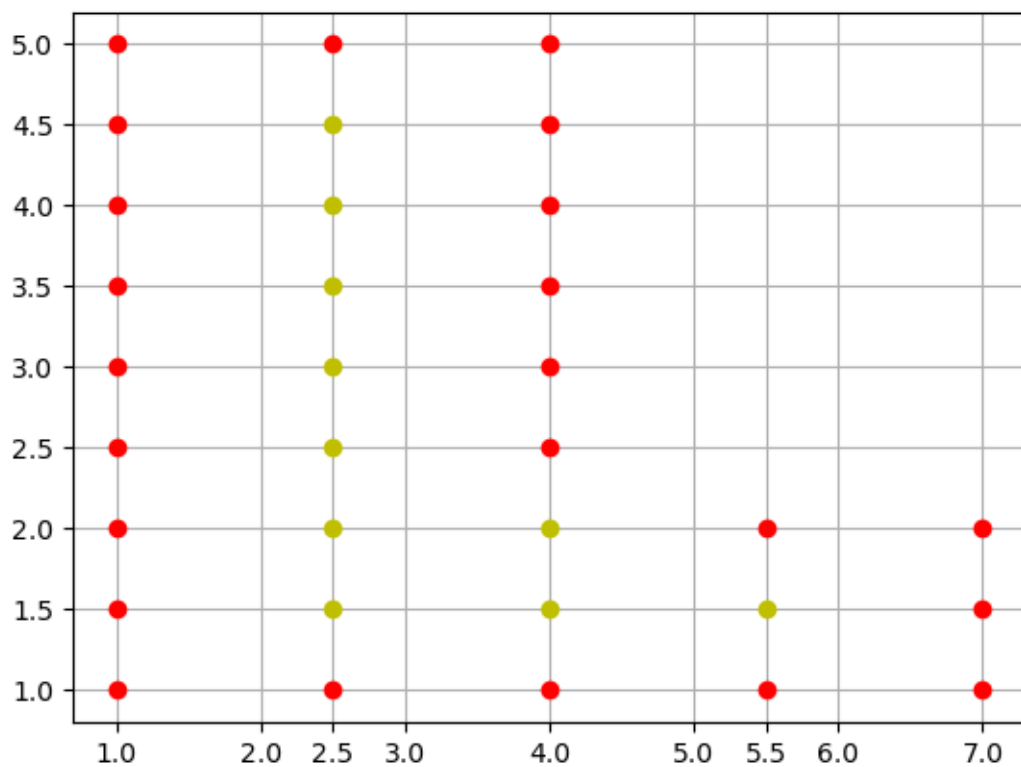
Функция: $x^2 - y$

Правая часть: $-1 + x^2 - y$,

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

Сетка: равномерная,

Заданы краевые условия 1-го рода и на самой верхней границе условие 3-го рода.



х у	Точное	Численное	Вектор погрешности	Погрешность
2.500 1.500	4.75	4.749999979951939	2.00E-8	1.23E-6
2.500 2.000	4.25	4.2499999945634936	5.44E-8	
2.500 2.500	3.75	3.7499998723780643	1.28E-7	
2.500 3.000	3.25	3.2499997069497866	2.93E-7	
2.500 3.500	2.75	2.749999329874131	6.70E-7	
2.500 4.000	2.25	2.2499984688186823	1.53E-6	
2.500 4.500	1.75	1.7499965019100598	3.50E-6	
4.000 1.500	14.5	14.49999999810813	1.89E-9	
4.000 2.000	14	13.999999997086054	2.91E-9	
5.500 1.500	28.75	28.749999999922736	7.73E-11	

Третий тест

Функция: $3x^3 + 2y^3$

Правая часть:

- 1-ая область: $-9x - 6y + 0.5(3x^3 + 2y^3)$,
- 2-ая область: $-36x - 24y + 2(3x^3 + 2y^3)$,

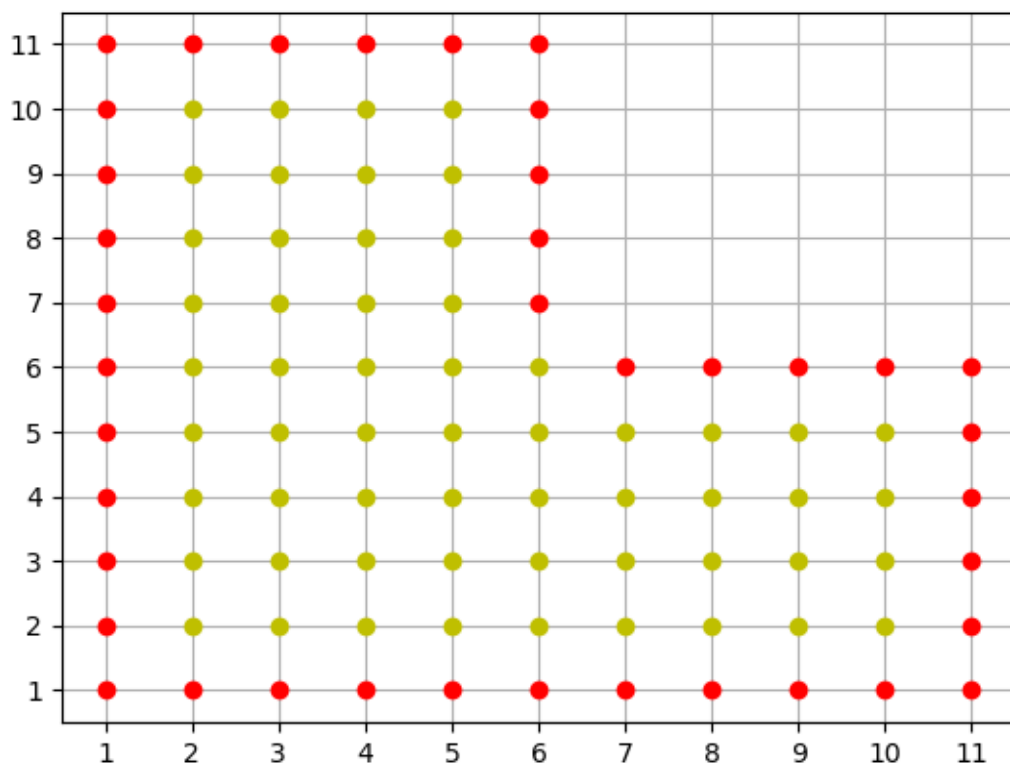
Коэффициенты :

- 1-ая область: $\lambda = 0.5, \gamma = 0.5$,
- 2-ая область: $\lambda = 2, \gamma = 2$.

Сетка:

- равномерная,
- 1-ая область: $[1; 6] \times [1; 11]$,
- 2-ая область: $[6; 11] \times [1; 6]$.

Заданы краевые условия 1-го рода и на нижней границе условие 3-го рода.



х у	Точное	Численное	Вектор погрешности	Погрешность
2.000 2.000	40	40.593814576817486	5.94E-1	1.43E-1
2.000 3.000	78	78.1546987217267	1.55E-1	
2.000 4.000	152	152.0431892181076	4.32E-2	
2.000 5.000	274	274.012790446567	1.28E-2	
2.000 6.000	456	456.0039632598384	3.96E-3	
2.000 7.000	710	710.0012648282988	1.26E-3	
2.000 8.000	1048	1048.000410183163	4.10E-4	
2.000 9.000	1482	1482.0001329319248	1.33E-4	
2.000 10.000	2024	2024.000039430162	3.94E-5	
3.000 2.000	97	97.3591509699836	3.59E-1	
3.000 3.000	135	135.13648981370793	1.36E-1	
3.000 4.000	209	209.04845692224538	4.85E-2	
3.000 5.000	331	331.01679975488815	1.68E-2	
3.000 6.000	513	513.0057610243261	5.76E-3	
3.000 7.000	767	767.0019506984921	1.95E-3	
3.000 8.000	1105	1105.0006531555923	6.53E-4	
3.000 9.000	1539	1539.0002150462992	2.15E-4	
3.000 10.000	2081	2081.0000642188847	6.42E-5	
4.000 2.000	208	208.30525316119508	3.05E-1	
4.000 3.000	246	246.12014245458374	1.20E-1	
4.000 4.000	320	320.0458058245235	4.58E-2	
4.000 5.000	442	442.0169903813024	1.70E-2	
4.000 6.000	624	624.006091408412	6.09E-3	
4.000 7.000	878	878.0020744842436	2.07E-3	
4.000 8.000	1216	1216.000689850007	6.90E-4	
4.000 9.000	1650	1650.0002249250952	2.25E-4	
4.000 10.000	2192	2192.000066617963	6.66E-5	
5.000 2.000	391	391.2916748840099	2.92E-1	
5.000 3.000	429	429.11316347349174	1.13E-1	
5.000 4.000	503	503.04343936448595	4.34E-2	
5.000 5.000	625	625.0162549186881	1.63E-2	
5.000 6.000	807	807.0056311521876	5.63E-3	
5.000 7.000	1061	1061.001640464307	1.64E-3	
5.000 8.000	1399	1399.000496685104	4.97E-4	
5.000 9.000	1833	1833.0001531112066	1.53E-4	
5.000 10.000	2375	2375.000043945834	4.39E-5	
6.000 2.000	664	664.2884784685658	2.88E-1	
6.000 3.000	702	702.110560664379	1.11E-1	
6.000 4.000	776	776.0419726057266	4.20E-2	
6.000 5.000	898	898.0152136954646	1.52E-2	
6.000 6.000	1080	1080.0041689695306	4.17E-3	
7.000 2.000	1045	1045.2889680608653	2.89E-1	
7.000 3.000	1083	1083.1091887741106	1.09E-1	
7.000 4.000	1157	1157.0406493043035	4.06E-2	
7.000 5.000	1279	1279.0136719833777	1.37E-2	
8.000 2.000	1552	1552.28560465262	2.86E-1	
8.000 3.000	1590	1590.1057658410061	1.06E-1	
8.000 4.000	1664	1664.0384131583014	3.84E-2	
8.000 5.000	1786	1786.0124969171204	1.25E-2	
9.000 2.000	2203	2203.272026716583	2.72E-1	
9.000 3.000	2241	2241.0956226199987	9.56E-2	
9.000 4.000	2315	2315.0331537290776	3.32E-2	

х у	Точное	Численное	Вектор погрешности	Погрешность
9.000 5.000	2437	2437.0103994439232	1.04E-2	
10.000 2.000	3016	3016.218878023472	2.19E-1	
10.000 3.000	3054	3054.067166813327	6.72E-2	
10.000 4.000	3128	3128.021333423164	2.13E-2	
10.000 5.000	3250	3250.0063465734174	6.35E-3	

Четвертый тест

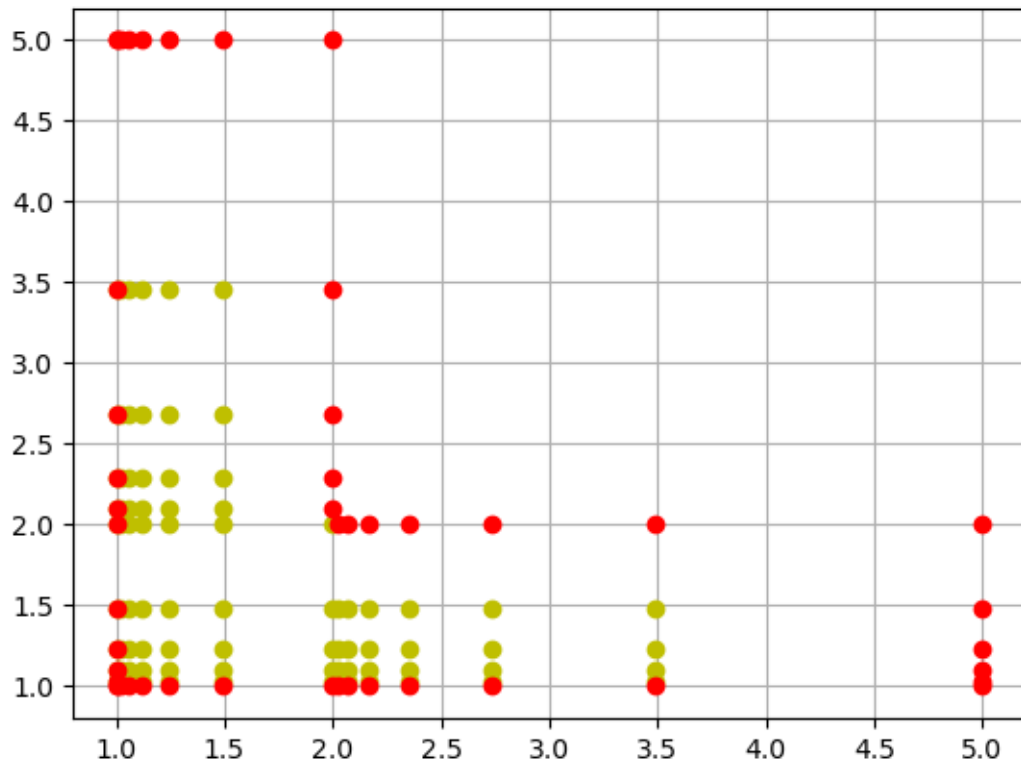
Функция: $x^2 - y$

Правая часть: $-1 + x^2 - y$,

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

Сетка: неравномерная,

Заданы краевые условия 1-го рода на всех границах.

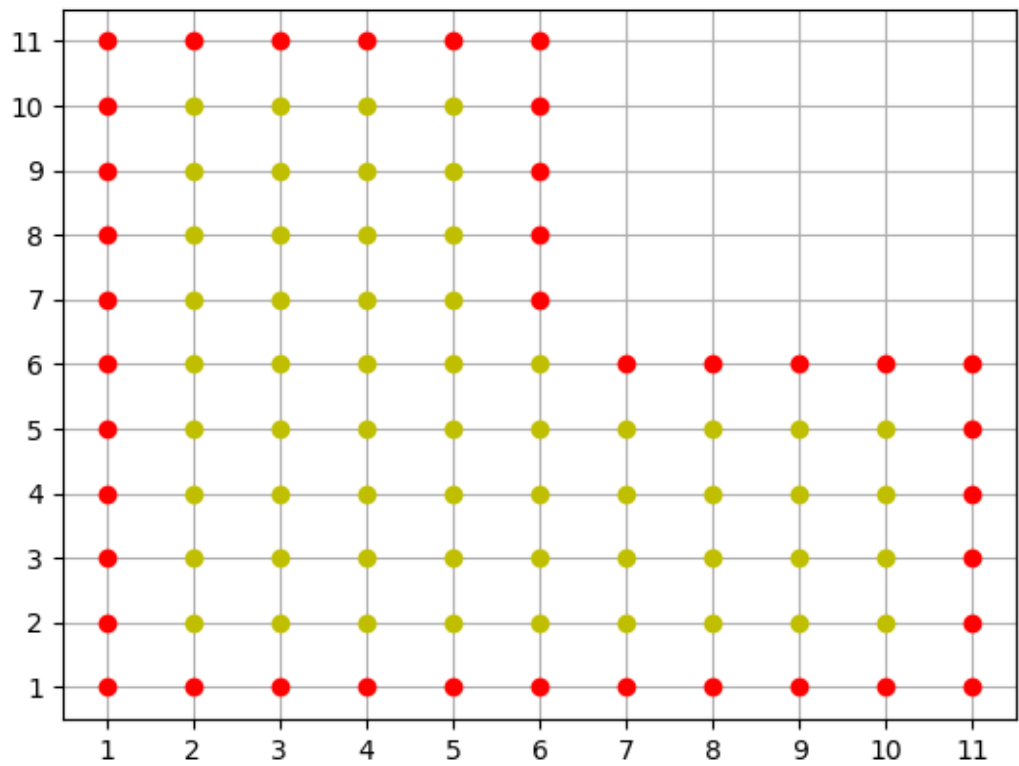


х у	Точное	Численное	Вектор погрешности	Погрешность
1.008 1.032	-0.01644803290	-0.01644803290	8.33E-17	2.92E-15
1.008 1.097	-0.08096416193	-0.08096416193	6.94E-17	
1.008 1.226	-0.20999641999	-0.20999641999	2.78E-17	
1.008 1.484	-0.46806093612	-0.46806093612	0	
1.008 2.000	-0.98418996838	-0.98418996838	0	
1.008 2.097	-1.08096416193	-1.08096416193	2.22E-16	
1.008 2.290	-1.27451254903	-1.27451254903	0	
1.008 2.677	-1.66160932322	-1.66160932322	0	
1.008 3.452	-2.43580287161	-2.43580287161	4.44E-16	
1.024 1.032	0.01554403109	0.01554403109	3.64E-17	
1.024 1.097	-0.04897209794	-0.04897209794	9.02E-17	
1.024 1.226	-0.17800435601	-0.17800435601	1.67E-16	
1.024 1.484	-0.43606887214	-0.43606887214	1.67E-16	
1.024 2.000	-0.95219790440	-0.95219790440	2.22E-16	
1.024 2.097	-1.04897209794	-1.04897209794	4.44E-16	
1.024 2.290	-1.24252048504	-1.24252048504	0	
1.024 2.677	-1.62961725923	-1.62961725923	4.44E-16	
1.024 3.452	-2.40381080762	-2.40381080762	4.44E-16	
1.055 1.032	0.08101616203	0.08101616203	1.39E-16	
1.055 1.097	0.01650003300	0.01650003300	2.29E-16	
1.055 1.226	-0.11253222506	-0.11253222506	3.33E-16	
1.055 1.484	-0.37059674119	-0.37059674119	3.89E-16	
1.055 2.000	-0.88672577345	-0.88672577345	3.33E-16	
1.055 2.097	-0.98349996700	-0.98349996700	4.44E-16	
1.055 2.290	-1.17704835410	-1.17704835410	2.22E-16	
1.055 2.677	-1.56414512829	-1.56414512829	4.44E-16	
1.055 3.452	-2.33833867668	-2.33833867668	4.44E-16	
1.118 1.032	0.21791243582	0.21791243582	1.39E-16	
1.118 1.097	0.15339630679	0.15339630679	2.50E-16	
1.118 1.226	0.02436404873	0.02436404873	3.43E-16	
1.118 1.484	-0.23370046740	-0.23370046740	3.89E-16	
1.118 2.000	-0.74982949966	-0.74982949966	1.11E-16	
1.118 2.097	-0.84660369321	-0.84660369321	5.55E-16	
1.118 2.290	-1.04015208030	-1.04015208030	6.66E-16	
1.118 2.677	-1.42724885450	-1.42724885450	2.22E-16	
1.118 3.452	-2.20144240288	-2.20144240288	0	
1.244 1.032	0.51551303103	0.51551303103	3.33E-16	
1.244 1.097	0.45099690199	0.45099690199	6.11E-16	
1.244 1.226	0.32196464393	0.32196464393	7.22E-16	
1.244 1.484	0.06390012780	0.06390012780	8.05E-16	
1.244 2.000	-0.45222890446	-0.45222890446	1.11E-16	
1.244 2.097	-0.54900309801	-0.54900309801	1.11E-16	
1.244 2.290	-0.74255148510	-0.74255148510	4.44E-16	
1.244 2.677	-1.12964825930	-1.12964825930	2.22E-16	
1.244 3.452	-1.90384180768	-1.90384180768	4.44E-16	
1.496 1.032	1.20594641189	1.20594641189	4.44E-16	
1.496 1.097	1.14143028286	1.14143028286	8.88E-16	
1.496 1.226	1.01239802480	1.01239802480	1.55E-15	
1.496 1.484	0.75433350867	0.75433350867	2.00E-15	
1.496 2.000	0.23820447641	0.23820447641	4.16E-16	
1.496 2.097	0.14143028286	0.14143028286	2.50E-16	
1.496 2.290	-0.05211810424	-0.05211810424	2.08E-17	

х у	Точное	Численное	Вектор погрешности	Погрешность
1.496 2.677	-0.43921487843	-0.43921487843	0	
1.496 3.452	-1.21340842682	-1.21340842682	2.22E-16	
2.000 1.032	2.96774193548	2.96774193548	1.33E-15	
2.000 1.097	2.90322580645	2.90322580645	3.11E-15	
2.000 1.226	2.77419354839	2.77419354839	4.88E-15	
2.000 1.484	2.51612903226	2.51612903226	7.11E-15	
2.000 2.000	2.00000000000	2.00000000000	0	
2.024 1.032	3.06278812558	3.06278812558	8.88E-16	
2.024 1.097	2.99827199654	2.99827199654	4.00E-15	
2.024 1.226	2.86923973848	2.86923973848	5.33E-15	
2.024 1.484	2.61117522235	2.61117522235	7.99E-15	
2.071 1.032	3.25622851246	3.25622851246	1.33E-15	
2.071 1.097	3.19171238342	3.19171238342	4.00E-15	
2.071 1.226	3.06268012536	3.06268012536	6.22E-15	
2.071 1.484	2.80461560923	2.80461560923	7.55E-15	
2.165 1.032	3.65650131300	3.65650131300	2.22E-15	
2.165 1.097	3.59198518397	3.59198518397	4.00E-15	
2.165 1.226	3.46295292591	3.46295292591	5.33E-15	
2.165 1.484	3.20488840978	3.20488840978	6.66E-15	
2.354 1.032	4.51061502123	4.51061502123	8.88E-16	
2.354 1.097	4.44609889220	4.44609889220	2.66E-15	
2.354 1.226	4.31706663413	4.31706663413	4.44E-15	
2.354 1.484	4.05900211800	4.05900211800	5.33E-15	
2.732 1.032	6.43311486623	6.43311486623	8.88E-16	
2.732 1.097	6.36859873720	6.36859873720	2.66E-15	
2.732 1.226	6.23956647913	6.23956647913	3.55E-15	
2.732 1.484	5.98150196300	5.98150196300	4.44E-15	
3.488 1.032	11.13520427041	11.13520427041	3.55E-15	
3.488 1.097	11.07068814138	11.07068814138	8.88E-15	
3.488 1.226	10.94165588331	10.94165588331	8.88E-15	
3.488 1.484	10.68359136718	10.68359136718	5.33E-15	

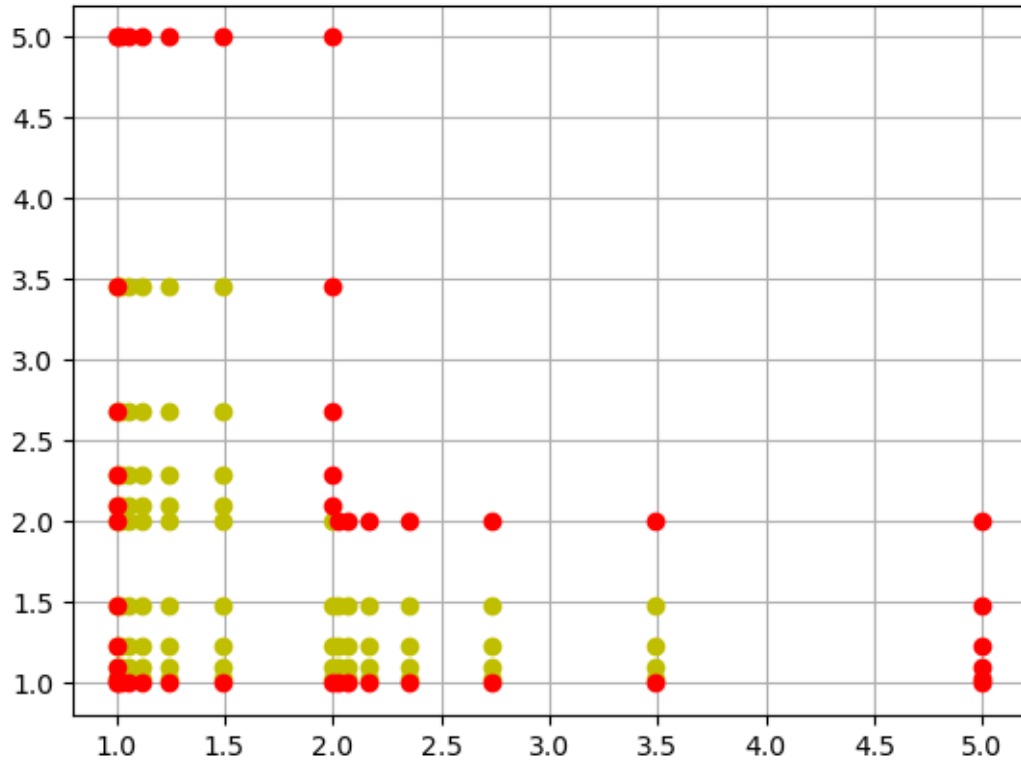
Исследование на определение порядка аппроксимации

Сетка: равномерная,
Коэффициенты: $\lambda = 1, \gamma = 0$,
На границах заданы краевые условия 1-го рода.



Функция	Относительная погрешность	Относительная невязка
$x + y$	4.72E-17	0
$x^2 + 2y^2$	5.02E-17	0
$3x^3 + 4y^3$	5.30E-17	0
x^4	1.02E-4	1.38E-3

Сетка: неравномерная,
 Коэффициенты: $\lambda = 1, \gamma = 0$,
 На границах заданы краевые условия 1-го рода.



Функция	Относительная погрешность	Относительная невязка
$x + y$	5.07E-16	3.89E-13
$x^2 + 2y^2$	2.62E-16	1.97E-13
$3x^3 + 4y^3$	7.08E-4	1.48E-2
x^4	1.85E-3	3.51E-2

Исследование на определение порядка сходимости

Возьмем неполиномиальную и полиномиальную функции и с помощью них оценим порядки сходимости метода.

Функция: $\ln(x + y)$

Правая часть: $\frac{2}{(x + y)^2}$

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

Сетка: равномерная,

Заданы краевые условия 1-го рода на всех границах.

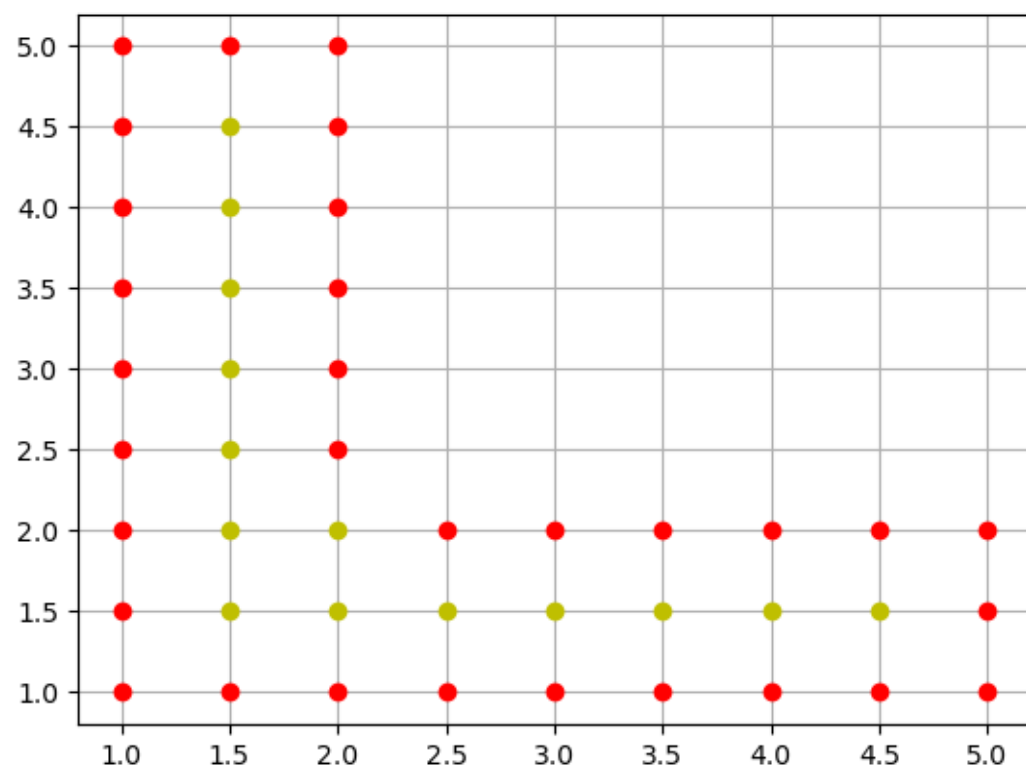
Функция: $4x^4$,

Правая часть: $-48x^2$,

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

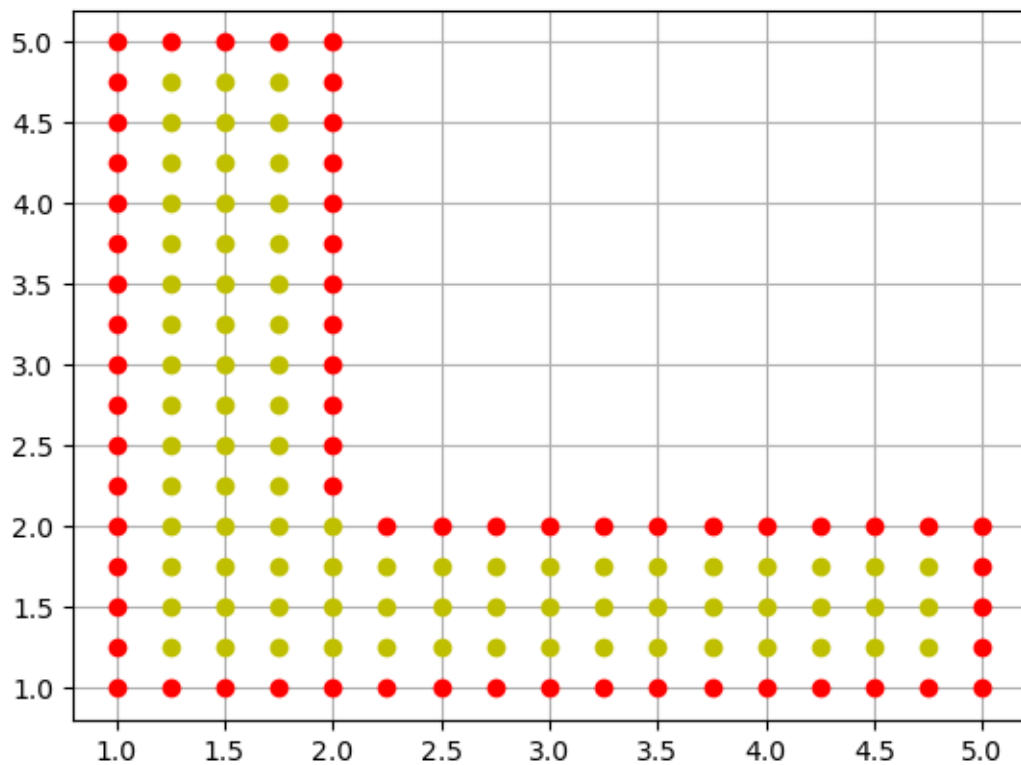
Сетка: равномерная,

Заданы краевые условия 1-го рода на всех границах.



Функция	Погрешность
$\ln(x + y)$	1.65E-4
$4x^4$	2.65E-1

Теперь раздробим сетку:



Функция	Погрешность
$\ln(x + y)$	3.47E-5
$4x^4$	5.30E-2

Далее, определим порядок сходимости метода на неравномерной сетке.

Функция: $3x^3 + 4y^3$,

Правая часть: $-18x - 24y$,

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

Сетка: неравномерная,

Заданы краевые условия 1-го рода на всех границах.

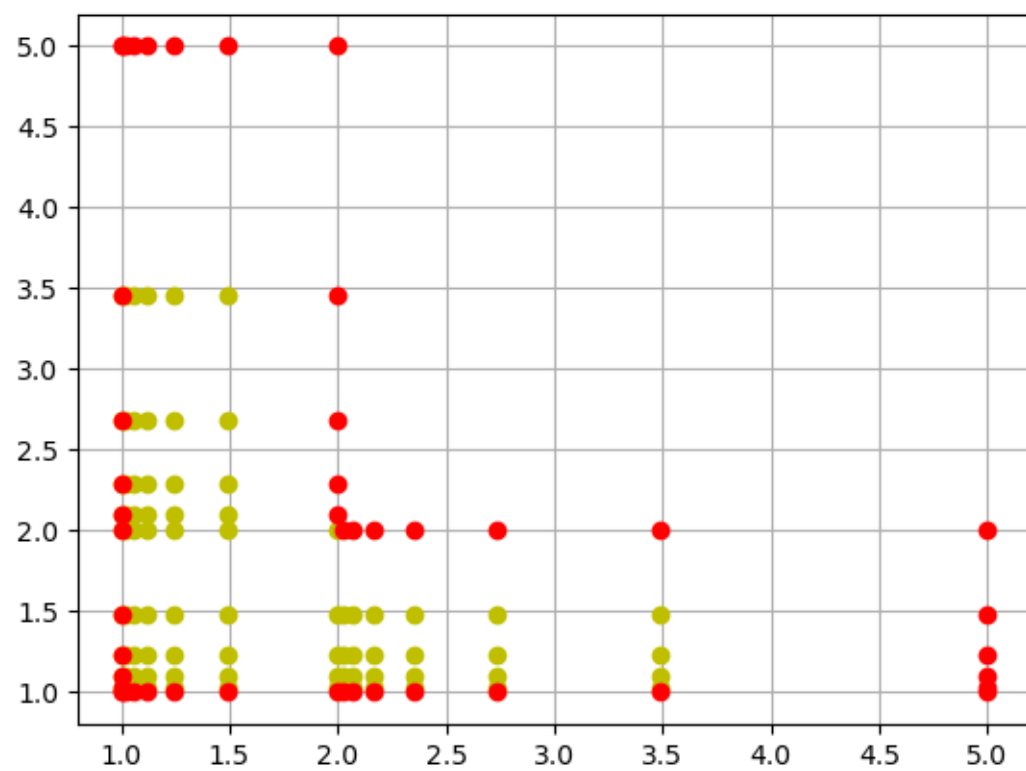
Функция: $\ln(x + y)$,

Правая часть: $\frac{2}{(x + y)^2}$,

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

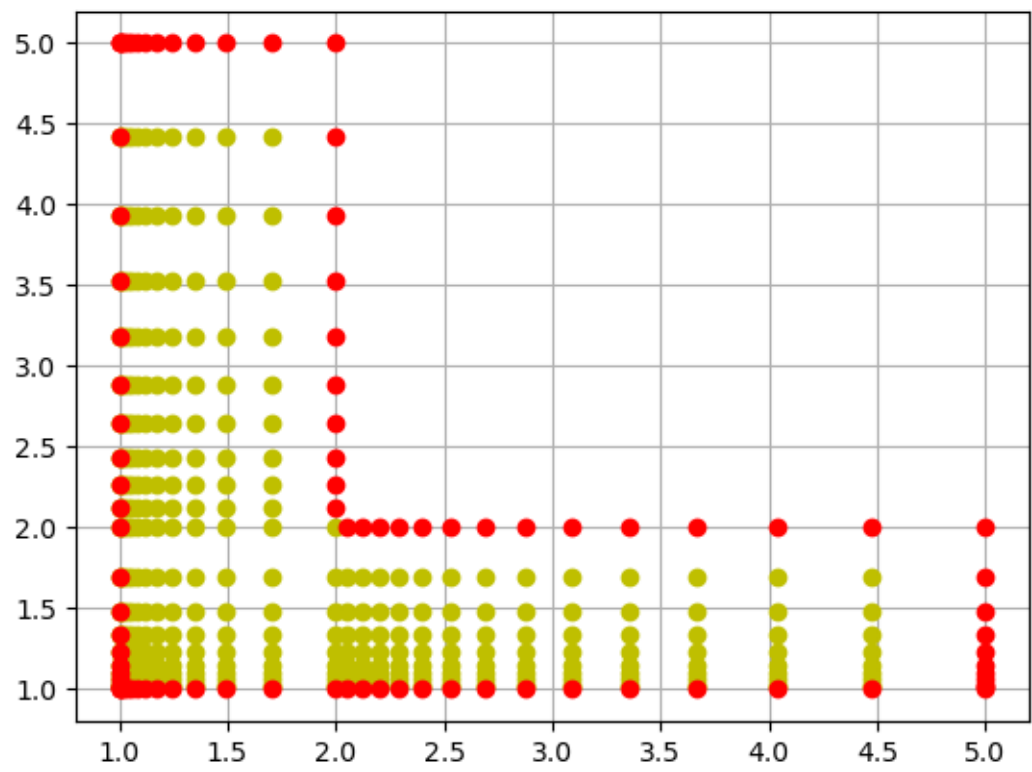
Сетка: неравномерная,

Заданы краевые условия 1-го рода на всех границах.



Функция	Погрешность
$\ln(x+y)$	2.42E-4
$3x^3 + 4y^3$	2.12E-1

Вложенная сетка:



Функция	Погрешность
$\ln(x + y)$	4.82E-5
$3x^3 + 4y^3$	3.89E-2

Проведенные исследования и выводы

Из результатов исследования на определение порядка сходимости на равномерной сетке видно, что при дроблении сетки погрешность упала в $\frac{1.65E-4}{3.47E-5} \approx 4.755$ раз (в случае неполинома) и упала в $\frac{2.65E-1}{5.30E-2} = 5$ раз (в случае полинома), следовательно, порядок $k_1 = \log_2 4.755 \approx 2.2$ и $k_2 = \log_2 5 \approx 2.3$.

Из результатов исследования на определение порядка сходимости на неравномерной сетке видно, что при дроблении погрешность упала в $\frac{2.42-4}{4.82E-5} \approx 5.02$ раз (в случае неполинома) и упала в $\frac{2.12E-1}{3.89E-2} \approx 5.45$ раз (в случае полинома), следовательно, порядок $k_1 = \log_2 5.02 \approx 2.33$ и $k_2 = \log_2 5.45 \approx 2.45$.

Теоретический порядок сходимости на равномерной сетке равен 2, значит, погрешность должна с дроблением сетки падать в 4 раза, а в случае неравномерной – порядок равен 1, это означает, что погрешность должна с дроблением сетки падать в 2 раза. Полученные результаты немного превысили теоретические. Возможно, это связано с заданной сеткой и особенностью задач.

Из результатов исследования на определение порядка аппроксимации видно, что на равномерной сетке погрешность выросла при вычислении полинома четвертого порядка, а в случае неравномерной – при вычислении полинома третьего порядка.

Теоретический порядок аппроксимации равен степени полинома, для которого точно считает метод, в случае равномерной сетки – третий порядок, неравномерной – второй порядок. Полученные результаты совпали с теоретическими.

Тексты основных модулей

Program.cs

```
1 using eMF_1;
2
3 GridFactory gridFactory = new();
4 // MFD mfd = new(gridFactory.CreateGrid(GridType.Irregular,
5   ↪ "grid/grid(irregular).txt"), "boundaries.txt");
6 MFD mfd = new(gridFactory.CreateGrid(GridType.Nested,
7   ↪ "grid/grid(nested).txt"), "boundaries.txt");
8 // MFD mfd = new(gridFactory.CreateGrid(GridType.Regular,
9   ↪ "grid/grid(regular).txt"), "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, 1.22));
19 mfd.Compute();
```

MFD.cs

```
1 namespace eMF_1;
2
3 public enum PointType {
4     Boundary,
5     Internal,
6     Dummy
7 }
8
9 public enum BoundaryType {
10     None,
11     Dirichlet,
12     Neumann,
13     Mixed
14 }
15
16 public enum GridType {
17     Regular,
18     Irregular,
19     Nested
20 }
21
22 public enum NormalType {
23     LeftX,
24     RightX,
25     UpperY,
26     BottomY
27 }
28
29 public class MFD {
30     private Grid _grid;
31     private DiagMatrix _matrix;
32     private ITest _test;
33     private ISolver _solver;
34     private Boundary[] _boundaries;
35     private double[] _q;
36     private double[] _pr;
37     private double _beta;
38     public double[] Weights
39         ⇒ _q;
40
41     public MFD(Grid grid, string boundaryPath) {
42         try {
43             using (var sr = new StreamReader(boundaryPath)) {
44                 _beta = double.Parse(sr.ReadLine());
45                 _boundaries = sr.ReadToEnd().Split("\n")
46                     .Select(str ⇒ Boundary.BoundaryParse(str)).ToArray();
47             }
48
49             _grid = grid;
50
51             } catch (Exception ex) {
52                 Console.WriteLine(ex.Message);
53             }
54         }
55
56     public void SetTest(ITest test)
57         ⇒ _test = test;
58
59     public void SetMethodSolvingSLAE(ISolver solver)
```

```

60         => _solver = solver;
61
62     public void Compute() {
63         try {
64             if (_test is null)
65                 throw new Exception("Set the test!");
66
67             if (_solver is null)
68                 throw new Exception("Set the method solving SLAE!");
69
70             _grid.Build();
71             _grid.AssignBoundaryConditions(_boundaries);
72             Init();
73             BuildMatrix();
74             _q = _solver.Compute(_matrix, _pr);
75
76         } catch (Exception ex) {
77             Console.WriteLine(ex.Message);
78         }
79     }
80
81     private void Init() {
82         _matrix = new(_grid.Points.Count, (_grid.AllLinesX.Count >
83             ↪ _grid.AllLinesX.Count) ?
84             _grid.AllLinesX.Count - 2 : _grid.AllLinesY.Count - 2);
85         _pr = new double[_matrix.Size];
86         _q = new double[_matrix.Size];
87     }
88
89     private void BuildMatrix() {
90         double hx, hy, hix, hiy, hi, h = 1E-12;
91         double lambda, gamma;
92         double us, ubeta;
93         double leftDerivative, rightDerivative;
94         NormalType normalType;
95
96         for (int i = 0; i < _grid.Points.Count; i++) {
97             switch (_grid.Points[i].PointType) {
98                 case PointType.Boundary:
99
100                 switch (_grid.Points[i].BoundaryType) {
101                     case BoundaryType.Dirichlet:
102
103                         _matrix.Diagnostics[i] = 1;
104                         _pr[i] = _test.U(_grid.Points[i]);
105
106                         break;
107
108                     case BoundaryType.Neumann:
109
110                         lambda = _grid.Areas[_grid.Points[i].AreaNumber]
111                             ↪ .Item2;
112
113                         normalType = _grid.Normal(_grid.Points[i]);
114
115                         switch (normalType) {
116                             case NormalType.LeftX:

```

```

116         hi = _grid.AllLinesX[_grid.Points[i].I +
            ↪ 1] -
            ↪ _grid.AllLinesX[_grid.Points[i].I];
117         _matrix.Diags[0][i] = lambda / hi;
118         _matrix.Diags[4][i] = -lambda / hi;
119         _pr[i] = -lambda *
            ↪ RightDerivativeX(_grid.Points[i], h);

120
121         break;
122
123     case NormalType.BottomY:
124
125         hi = _grid.AllLinesY[_grid.Points[i].J +
            ↪ 1] -
            ↪ _grid.AllLinesY[_grid.Points[i].J];
126         _matrix.Diags[0][i] = lambda / hi;
127         _matrix.Diags[3][i] = -lambda / hi;
128         _pr[i] = -lambda *
            ↪ RightDerivativeY(_grid.Points[i], h);

129
130         break;
131
132     case NormalType.RightX:
133
134         hi = _grid.AllLinesX[_grid.Points[i].I]
            ↪ - _grid.AllLinesX[_grid.Points[i].I
            ↪ - 1];
135         _matrix.Diags[0][i] = lambda / hi;
136         _matrix.Diags[2][i + _matrix.Indexes[2]]
            ↪ = -lambda / hi;
137         _pr[i] = lambda *
            ↪ LeftDerivativeX(_grid.Points[i], h);

138
139         break;
140
141     case NormalType.UpperY:
142
143         hi = _grid.AllLinesY[_grid.Points[i].J]
            ↪ - _grid.AllLinesY[_grid.Points[i].J
            ↪ - 1];
144         _matrix.Diags[0][i] = lambda / hi;
145         _matrix.Diags[1][i + _matrix.Indexes[1]]
            ↪ = -lambda / hi;
146         _pr[i] = lambda *
            ↪ LeftDerivativeY(_grid.Points[i], h);

147
148         break;
149
150     default:
151         throw new ArgumentOutOfRangeException(nameof
            ↪ meof(normalType),
152         ↪ $"This type of normal does not exist:
            ↪ {normalType}");
153     }
154
155     break;
156
157     case BoundaryType.Mixed:
158

```

```

159 lambda = _grid.Areas[_grid.Points[i].AreaNumber]
    ↪ .Item2;
160
161 normalType = _grid.Normal(_grid.Points[i]);
162 us = _test.U(_grid.Points[i]);
163
164 switch (normalType) {
165     case NormalType.LeftX:
166
167         hi = _grid.AllLinesX[_grid.Points[i].I +
    ↪ 1] -
    ↪ _grid.AllLinesX[_grid.Points[i].I];
168         rightDerivative =
    ↪ RightDerivativeX(_grid.Points[i], h);
169         ubeta = -lambda * rightDerivative /
    ↪ _beta + us;
170         _matrix.Diags[0][i] = lambda / hi +
    ↪ _beta;
171         _matrix.Diags[4][i] = -lambda / hi;
172         _pr[i] = -lambda * rightDerivative +
    ↪ _beta * (us - ubeta) + _beta * ubeta;
173
174         break;
175
176     case NormalType.BottomY:
177
178         hi = _grid.AllLinesY[_grid.Points[i].J +
    ↪ 1] -
    ↪ _grid.AllLinesY[_grid.Points[i].J];
179         rightDerivative =
    ↪ RightDerivativeY(_grid.Points[i], h);
180         ubeta = -lambda * rightDerivative /
    ↪ _beta + us;
181         _matrix.Diags[0][i] = lambda / hi +
    ↪ _beta;
182         _matrix.Diags[3][i] = -lambda / hi;
183         _pr[i] = -lambda * rightDerivative +
    ↪ _beta * (us - ubeta) + _beta * ubeta;
184
185         break;
186
187     case NormalType.RightX:
188
189         hi = _grid.AllLinesX[_grid.Points[i].I]
    ↪ - _grid.AllLinesX[_grid.Points[i].I
    ↪ - 1];
190         leftDerivative =
    ↪ LeftDerivativeX(_grid.Points[i], h);
191         ubeta = lambda * leftDerivative / _beta
    ↪ + us;
192         _matrix.Diags[0][i] = lambda / hi +
    ↪ _beta;
193         _matrix.Diags[2][i + _matrix.Indexes[2]]
    ↪ = -lambda / hi;
194         _pr[i] = lambda * leftDerivative + _beta
    ↪ * (us - ubeta) + _beta * ubeta;
195
196         break;
197
198     case NormalType.UpperY:

```

```

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

```

```

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

```

DiagMatrix.cs

```

1 namespace eMF_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 }

```

Point2D.cs

```

1 namespace eMF_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; init; }
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]),
24            ↪ (PointType)Enum.Parse(typeof(PointType), data[4]));
25
26        return point;
27    }
28
29    public static Point2D operator +(Point2D point, (double, double) value)
30    ⇒ new(point.X + value.Item1, point.Y + value.Item2, point.I,
31        ↪ point.J, point.PointType);
32
33    public static Point2D operator -(Point2D point, (double, double) value)
34    ⇒ new(point.X - value.Item1, point.Y - value.Item2, point.I,
35        ↪ point.J, point.PointType);
36
37    public override string ToString()
38    ⇒ $"{X} {Y}";
39 }

```

GridFactory.cs

```

1 namespace eMF_1;
2
3 public class GridFactory {
4     public Grid CreateGrid(GridType gridType, string path) {
5         return gridType switch {
6             GridType.Regular ⇒ new RegularGrid(path),
7

```

```

8         GridType.Irregular ⇒ new IrregularGrid(path),
9
10        GridType.Nested ⇒ new NestedGrid(path),
11
12        ⇒ throw new ArgumentOutOfRangeException(nameof(gridType),
13        $"This type of grid does not exist: {gridType}")
14    };
15 }
16 }

```

Grid.cs

```

1 namespace eMF_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,
10     ↪ int)> Areas { get; }
11
12     public abstract void Build();
13
14     public NormalType Normal(Point2D point) {
15         NormalType normalType = default(NormalType);
16
17         if (point.PointType ≠ PointType.Boundary)
18             throw new Exception("To determine the normal to the boundary,
19             ↪ the point needs the boundary!");
20
21         else if (point.X ≥ LinesX[0] && point.X ≤ LinesX[1] && point.Y =
22             ↪ LinesY[2]) {
23             normalType = NormalType.UpperY;
24             return normalType;
25         } else if (point.X ≥ LinesX[1] && point.X ≤ LinesX[2] && point.Y
26             ↪ = LinesY[1]) {
27             normalType = NormalType.UpperY;
28             return normalType;
29         } else if (point.X = LinesX[0] && point.Y ≥ LinesY[0] && point.Y
30             ↪ ≤ LinesY[2]) {
31             normalType = NormalType.LeftX;
32             return normalType;
33         } else if (point.X = LinesX[1] && point.Y ≥ LinesY[1] && point.Y
34             ↪ ≤ LinesY[2]) {
35             normalType = NormalType.RightX;
36             return normalType;
37         } else if (point.X = LinesX[2] && point.Y ≥ LinesY[0] && point.Y
38             ↪ ≤ LinesY[1]) {
39             normalType = NormalType.RightX;
40             return normalType;
41         } else if (point.X ≥ LinesX[0] && point.X ≤ LinesX[2] && point.Y
42             ↪ = LinesY[0]) {
43             normalType = NormalType.BottomY;
44             return normalType;
45         } else
46             return normalType;
47     }
48 }

```

```

41
42 protected PointType PointsTypes(double x, double y) {
43     double eps = 1E-14;
44
45     if ((x > LinesX[0] && x < LinesX[2] && y > LinesY[0] && y <
46         ↪ LinesY[1])
47         || (x > LinesX[0] && x < LinesX[1] && y > LinesY[0] && y < LinesY[2])
48         || (x = LinesX[1] && y = LinesY[1]))
49         return PointType.Internal;
50
51     for (int i = 0; i < AllLinesX.Count; i++)
52         if ((Math.Abs(x - AllLinesX[i]) < eps && Math.Abs(y - LinesY[0])
53             ↪ < eps)
54             || (Math.Abs(x - AllLinesX[i]) < eps && Math.Abs(y - LinesY[1])
55                 ↪ < eps && x ≥ LinesX[1])
56             || (Math.Abs(x - AllLinesX[i]) < eps && Math.Abs(y - LinesY[2])
57                 ↪ < eps && x ≤ LinesX[1]))
58             return PointType.Boundary;
59
60     for (int i = 0; i < AllLinesY.Count; i++)
61         if ((Math.Abs(y - AllLinesY[i]) < eps && Math.Abs(x - LinesX[0])
62             ↪ < eps)
63             || (Math.Abs(y - AllLinesY[i]) < eps && Math.Abs(x - LinesX[1])
64                 ↪ < eps && y ≥ LinesY[1])
65             || (Math.Abs(y - AllLinesY[i]) < eps && Math.Abs(x - LinesX[2])
66                 ↪ < eps && y ≤ LinesY[1]))
67             return PointType.Boundary;
68
69     return PointType.Dummy;
70 }
71
72 protected void SetAreaNumber() {
73     for (int i = 0; i < Points.Count; i++)
74         for (int iArea = 0; iArea < Areas.Length; iArea++) {
75             if (Points[i].X ≥ Areas[iArea].Item4 && Points[i].X ≤
76                 ↪ Areas[iArea].Item5
77             && Points[i].Y ≥ Areas[iArea].Item6 && Points[i].Y ≤
78                 ↪ Areas[iArea].Item7) {
79                 Points[i].AreaNumber = Areas[iArea].Item1;
80             }
81         }
82     }
83
84 public void AssignBoundaryConditions(Boundary[] boundaries) {
85     foreach (var point in Points.Where(point ⇒ point.PointType ==
86         ↪ PointType.Boundary))
87         for (int k = 0; k < boundaries.Length; k++) {
88             if (point.X ≥ LinesX[boundaries[k].X1] && point.X ≤
89                 ↪ LinesX[boundaries[k].X2]
90             && point.Y ≥ LinesY[boundaries[k].Y1] && point.Y ≤
91                 ↪ LinesY[boundaries[k].Y2]) {
92                 point.BoundaryType = boundaries[k].BoundaryType;
93                 break;
94             }
95         }
96     }
97
98 protected void WriteToFilePoints() {
99     using (var sw = new StreamWriter("points/boundaryPoints.txt")) {

```

```

88         Points.ForEach(x => { if (x.PointType == PointType.Boundary)
89             ↪ sw.WriteLine(x); });
90     }
91     using (var sw = new StreamWriter("points/internalPoints.txt")) {
92         Points.ForEach(x => { if (x.PointType == PointType.Internal)
93             ↪ sw.WriteLine(x); });
94     }
95     using (var sw = new StreamWriter("points/dummyPoints.txt")) {
96         Points.ForEach(x => { if (x.PointType == PointType.Dummy)
97             ↪ sw.WriteLine(x); });
98     }
99 }

```

RegularGrid.cs

```

1  namespace eMF_1;
2
3  public class RegularGrid : Grid {
4      private List<double> _allLinesX;
5      private List<double> _allLinesY;
6      private List<Point2D> _points;
7      private (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,
17         ↪ int)> Areas
18         => _areas.ToImmutableArray();
19     public int SplitsX { get; init; }
20     public int SplitsY { get; init; }
21
22     public RegularGrid(string path) {
23         try {
24             using (var sr = new StreamReader(path)) {
25                 LinesX = sr.ReadLine().Split().Select(value =>
26                     ↪ double.Parse(value)).ToImmutableArray();
27                 LinesY = sr.ReadLine().Split().Select(value =>
28                     ↪ double.Parse(value)).ToImmutableArray();
29                 SplitsX = int.Parse(sr.ReadLine());
30                 SplitsY = int.Parse(sr.ReadLine());
31                 _areas = sr.ReadToEnd().Split("\n").Select(row =>
32                     ↪ row.Split())
33                     .Select(value => (int.Parse(value[0]),
34                     ↪ double.Parse(value[1]), double.Parse(value[2]),
35                     ↪ int.Parse(value[3]), int.Parse(value[4]),
36                     ↪ int.Parse(value[5]), int.Parse(value[6]))).ToArray();
37             }
38
39             _allLinesX = new();
40             _allLinesY = new();
41             _points = new();
42         } catch (Exception ex) {

```

```

37         Console.WriteLine(ex.Message);
38     }
39 }
40
41 public override void Build() {
42     double h;
43     double lenght = LinesX.Last() - LinesX.First();
44
45     h = lenght / SplitsX;
46
47     _allLinesX.Add(LinesX.First());
48
49     while (Math.Round(_allLinesX.Last() + h, 1) < LinesX.Last())
50         _allLinesX.Add(_allLinesX.Last() + h);
51
52     _allLinesX = _allLinesX.Union(LinesX).OrderBy(value =>
53         ↪ value).ToList();
54
55     lenght = LinesY.Last() - LinesY.First();
56
57     h = lenght / SplitsY;
58
59     _allLinesY.Add(LinesY.First());
60
61     while (Math.Round(_allLinesY.Last() + h, 1) < LinesY.Last())
62         _allLinesY.Add(_allLinesY.Last() + h);
63
64     _allLinesY = _allLinesY.Union(LinesY).OrderBy(value =>
65         ↪ value).ToList();
66
67     for (int i = 0; i < _allLinesX.Count; i++)
68         for (int j = 0; j < _allLinesY.Count; j++)
69             _points.Add(new(_allLinesX[i], _allLinesY[j], i, j,
70                 PointsTypes(_allLinesX[i], _allLinesY[j])));
71
72     SetAreaNumber();
73     WriteToFilePoints();
74 }

```

IrregularGrid.cs

```

1 namespace eMF_1;
2
3 public class IrregularGrid : Grid {
4     private List<double> _allLinesX;
5     private List<double> _allLinesY;
6     private List<Point2D> _points;
7     private int[] _splitsX;
8     private int[] _splitsY;
9     private double[] _kX;
10    private double[] _kY;
11    private (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 ⇒
    ⇒ double.Parse(value)).ToImmutableArray();
35             LinesY = sr.ReadLine().Split().Select(value ⇒
    ⇒ double.Parse(value)).ToImmutableArray();
36             _splitsX = sr.ReadLine().Split().Select(value ⇒
    ⇒ int.Parse(value)).ToArray();
37             _splitsY = sr.ReadLine().Split().Select(value ⇒
    ⇒ int.Parse(value)).ToArray();
38             _kX = sr.ReadLine().Split().Select(value ⇒
    ⇒ double.Parse(value)).ToArray();
39             _kY = sr.ReadLine().Split().Select(value ⇒
    ⇒ double.Parse(value)).ToArray();
40             _areas = sr.ReadToEnd().Split("\n").Select(row ⇒
    ⇒ row.Split())
41             .Select(value ⇒ (int.Parse(value[0]),
    ⇒ double.Parse(value[1]), double.Parse(value[2]),
42             int.Parse(value[3]), int.Parse(value[4]),
    ⇒ int.Parse(value[5]), int.Parse(value[6]))).ToArray();
43
44         }
45
46         _allLinesX = new();
47         _allLinesY = new();
48         _points = new();
49     } catch (Exception ex) {
50         Console.WriteLine(ex.Message);
51     }
52 }
53
54 public override void Build() {
55     for (int i = 0; i < LinesX.Length - 1; i++) {
56         double h;
57         double sum = 0;
58         double lenght = LinesX[i + 1] - LinesX[i];
59
60         for (int k = 0; k < _splitsX[i]; k++)
61             sum += Math.Pow(_kX[i], k);
62
63         h = lenght / sum;
64
65         _allLinesX.Add(LinesX[i]);
66
67         while (Math.Round(_allLinesX.Last() + h, 1) < LinesX[i + 1]) {

```

```

68         _allLinesX.Add(_allLinesX.Last() + h);
69
70         h *= _kX[i];
71     }
72
73     sum = 0;
74
75 }
76
77 _allLinesX.Add(LinesX.Last());
78
79 for (int i = 0; i < LinesY.Length - 1; i++) {
80     double h;
81     double sum = 0;
82     double lenght = LinesY[i + 1] - LinesY[i];
83
84     for (int k = 0; k < _splitsY[i]; k++)
85         sum += Math.Pow(_kY[i], k);
86
87     h = lenght / sum;
88
89     _allLinesY.Add(LinesY[i]);
90
91     while (Math.Round(_allLinesY.Last() + h, 1) < LinesY[i + 1]) {
92         _allLinesY.Add(_allLinesY.Last() + h);
93
94         h *= _kY[i];
95     }
96
97     sum = 0;
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             PointTypes(_allLinesX[i], _allLinesY[j])));
106
107 SetAreaNumber();
108 WriteToFilePoints();
109 }
110 }

```

NestedGrid.cs

```

1  namespace eMF_1;
2
3  public class NestedGrid : Grid {
4      private List<double> _allLinesX;
5      private List<double> _allLinesY;
6      private List<Point2D> _points;
7      private int[] _splitsX;
8      private int[] _splitsY;
9      private double[] _kX;
10     private double[] _kY;
11     private (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 NestedGrid(string path) {
32     try {
33         using (var sr = new StreamReader(path)) {
34             LinesX = sr.ReadLine().Split().Select(value ⇒
    ⇨ double.Parse(value)).ToImmutableArray();
35             LinesY = sr.ReadLine().Split().Select(value ⇒
    ⇨ double.Parse(value)).ToImmutableArray();
36             _splitsX = sr.ReadLine().Split().Select(value ⇒
    ⇨ int.Parse(value)).ToArray();
37             _splitsY = sr.ReadLine().Split().Select(value ⇒
    ⇨ int.Parse(value)).ToArray();
38             _kX = sr.ReadLine().Split().Select(value ⇒
    ⇨ double.Parse(value)).ToArray();
39             _kY = sr.ReadLine().Split().Select(value ⇒
    ⇨ double.Parse(value)).ToArray();
40             _areas = sr.ReadToEnd().Split("\n").Select(row ⇒
    ⇨ row.Split())
41             .Select(value ⇒ (int.Parse(value[0]),
    ⇨ double.Parse(value[1]), double.Parse(value[2]),
42             int.Parse(value[3]), int.Parse(value[4]),
    ⇨ int.Parse(value[5]), int.Parse(value[6]))).ToArray();
43
44         }
45
46         _allLinesX = new();
47         _allLinesY = new();
48         _points = new();
49     } catch (Exception ex) {
50         Console.WriteLine(ex.Message);
51     }
52 }
53
54 public override void Build() {
55     for (int i = 0; i < LinesX.Length - 1; i++) {
56         double h;
57         double sum = 0;
58         double lenght = LinesX[i + 1] - LinesX[i];
59
60         for (int j = 0; j < _kX.Length; j++)
61             _kX[j] = Math.Sqrt(_kX[j]);
62
63         for (int k = 0; k < _splitsX[i]; k++)
64             sum += Math.Pow(_kX[i], k);

```



```

65         h = lenght / sum;
66
67         _allLinesX.Add(LinesX[i]);
68
69         while (Math.Round(_allLinesX.Last() + h, 1) < LinesX[i + 1]) {
70             _allLinesX.Add(_allLinesX.Last() + h);
71
72             h *= _kX[i];
73         }
74
75         sum = 0;
76
77     }
78
79     _allLinesX.Add(LinesX.Last());
80
81     for (int i = 0; i < LinesY.Length - 1; i++) {
82         double h;
83         double sum = 0;
84         double lenght = LinesY[i + 1] - LinesY[i];
85
86         for (int j = 0; j < _kY.Length; j++)
87             _kY[j] = Math.Sqrt(_kY[j]);
88
89         for (int k = 0; k < _splitsY[i]; k++)
90             sum += Math.Pow(_kY[i], k);
91
92         h = lenght / sum;
93
94         _allLinesY.Add(LinesY[i]);
95
96         while (Math.Round(_allLinesY.Last() + h, 1) < LinesY[i + 1]) {
97             _allLinesY.Add(_allLinesY.Last() + h);
98
99             h *= _kY[i];
100         }
101
102         sum = 0;
103     }
104
105     _allLinesY.Add(LinesY.Last());
106
107     for (int i = 0; i < _allLinesX.Count; i++)
108         for (int j = 0; j < _allLinesY.Count; j++)
109             _points.Add(new(_allLinesX[i], _allLinesY[j], i, j,
110                 PointsTypes(_allLinesX[i], _allLinesY[j])));
111
112     SetAreaNumber();
113     WriteToFilePoints();
114 }
115 }
116 }

```

Boundary.cs

```

1 namespace eMF_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 }

```

ISolver.cs

```

1 namespace eMF_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 }

```

GaussSeidel.cs

```

1 namespace eMF_1;
2
3 public record GaussSeidel(int MaxIters, double Eps, double W) : ISolver {
4     public double[] Compute(DiagMatrix diagMatrix, double[] pr) {
5         double[] qk = new double[diagMatrix.Size];
6         double[] qk1 = new double[diagMatrix.Size];
7         double[] residual = new double[diagMatrix.Size];
8         double prNorm = pr.Norm();
9
10        for (int i = 0; i < MaxIters; i++) {
11            for (int k = 0; k < diagMatrix.Size; k++) {
12                double fstSum = MultLine(diagMatrix, k, qk1, 1);
13                double scdSum = MultLine(diagMatrix, k, qk, 2);
14
15                residual[k] = pr[k] - (fstSum + scdSum);
16                qk1[k] = qk[k] + W * residual[k] / diagMatrix.Diagnostics[k];
17            }
18
19            qk1.Copy(qk);
20            qk1.Fill(0);
21
22            if (residual.Norm() / prNorm < Eps)
23                break;
24        }
25
26        return qk;
27    }
28
29    private double MultLine(DiagMatrix diagMatrix, int i, double[] vector,
        ↪ int method) {
30        double sum = 0;
31
32        if (method == 0 || method == 1) {
33            if (i > 0) {
34                sum += diagMatrix.Diagnostics[i - 1] * vector[i - 1];
35
36                if (i > diagMatrix.ZeroDiags + 1)

```

```

37         sum += diagMatrix.Diagnostics[2][i - diagMatrix.ZeroDiags - 2]
           ↪ * vector[i - diagMatrix.ZeroDiags - 2];
38     }
39 }
40
41 if (method == 0 || method == 2) {
42     sum += diagMatrix.Diagnostics[0][i] * vector[i];
43
44     if (i < diagMatrix.Size - 1) {
45         sum += diagMatrix.Diagnostics[3][i] * vector[i + 1];
46
47         if (i < diagMatrix.Size - diagMatrix.ZeroDiags - 2)
48             sum += diagMatrix.Diagnostics[4][i] * vector[i +
           ↪ diagMatrix.ZeroDiags + 2];
49     }
50 }
51
52 return sum;
53 }
54 }

```

ITest.cs

```

1 namespace eMF_1;
2
3 public interface ITest {
4     public double U(Point2D point);
5     public double F(Point2D point);
6 }

```

FirstTest.cs

```

1 namespace eMF_1;
2
3 public class FirstTest : ITest {
4     public double U(Point2D point)
5         ⇒ point.X;
6
7     public double F(Point2D point)
8         ⇒ 0;
9 }

```

SecondTest.cs

```

1 namespace eMF_1;
2
3 public class SecondTest : ITest {
4     public double U(Point2D point)
5         ⇒ point.X * point.X - point.Y;
6
7     public double F(Point2D point)
8         ⇒ -1 + point.X * point.X - point.Y;
9 }

```

ThirdTest.cs

```

1 namespace eMF_1;
2
3 public class ThirdTest : ITest {
4     public double U(Point2D point)
5         ⇒ 3 * point.X * point.X * point.X + 2 * point.Y * point.Y * point.Y;

```

```

6
7     public double F(Point2D point)
8         ⇒ (point.AreaNumber == 0) ? -9 * point.X - 6 * point.Y + 0.5 *
9           (3 * point.X * point.X * point.X + 2 * point.Y * point.Y * point.Y) :
10        -36 * point.X - 24 * point.Y + 2 * (3 * point.X * point.X * point.X
11        ↪      + 2 * point.Y * point.Y * point.Y);
    }

```

FourthTest.cs

```

1 namespace eMF_1;
2
3 public class FourthTest : ITest {
4     public double U(Point2D point)
5         ⇒ Math.Log(point.X + point.Y);
6
7     public double F(Point2D point)
8         ⇒ 2 / ((point.X + point.Y) * (point.X + point.Y));
9 }

```

FifthTest.cs

```

1 namespace eMF_1;
2
3 public class FifthTest : ITest {
4     public double U(Point2D point)
5         ⇒ 4 * point.X * point.X * point.X * point.X;
6     public double F(Point2D point)
7         ⇒ -48 * point.X * point.X;
8 }

```

SixthTest.cs

```

1 namespace eMF_1;
2
3 public class SixthTest : ITest {
4     public double U(Point2D point)
5         ⇒ 4 * point.X * point.X * point.X * point.X + 2 * point.Y * point.Y
6         ↪      * point.Y * point.Y;
7
8     public double F(Point2D point)
9         ⇒ -48 * point.X * point.X - 24 * point.Y * point.Y;
    }

```

SeventhTest.cs

```
1 namespace eMF_1;
2
3 public class SeventhTest : ITest {
4     public double U(Point2D point)
5         ⇒ Math.Exp(point.X) + point.Y;
6
7     public double F(Point2D point)
8         ⇒ point.Y;
9 }
```

EighthTest.cs

```
1 namespace eMF_1;
2
3 public class EighthTest : ITest {
4     public double U(Point2D point)
5         ⇒ point.X * point.X * point.X + point.Y;
6
7     public double F(Point2D point)
8         ⇒ -6 * point.X;
9 }
```

Array1DExtension.cs

```
1 namespace eMF_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 }
```

graphics.py

```
1 import matplotlib.pyplot as plt
2 from decimal import Decimal as dcm
3
4 xB, xI, xD, yB, yI, yD = [], [], [], [], [], []
5
6 with open("points/boundaryPoints.txt") as file:
7     for line in file:
8         xC, yC = line.split()
9         xB.append(dcm(xC))
10        yB.append(dcm(yC))
```

```
11
12 with open("points/internalPoints.txt") as file:
13     for line in file:
14         xC, yC = line.split()
15         xI.append(dcm(xC))
16         yI.append(dcm(yC))
17
18 with open("points/dummyPoints.txt") as file:
19     for line in file:
20         xC, yC = line.split()
21         xD.append(dcm(xC))
22         yD.append(dcm(yC))
23
24 plt.grid()
25
26 plt.plot(xI, yI, 'o', color='y')
27 plt.plot(xB, yB, 'o', color='r')
28 # plt.plot(xD, yD, 'o', color='b')
29 plt.show()
```