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	ЗАДОРОЖНЫЙ А.Г.
	ПЕРСОВА М.Г.
	ПАТРУШЕВ И. И.
Дата	21.02.2022

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:

$$u(x_1 + h_1, \ldots, x_n + h_n) = u(x_1, \ldots, x_n) + \sum_{j=1}^{n} h_j \frac{\partial}{\partial x_j} u(x_1, \ldots, x_n) +$$

$$\frac{1}{2}\left(\sum_{j=1}^{n} h_j \frac{\partial}{\partial x_j}\right)^2 u(x_1, \ldots, x_n) + \cdots + \frac{1}{k!}\left(\sum_{j=1}^{n} h_j \frac{\partial}{\partial x_j}\right)^k u(x_1, \ldots, x_n) + \tag{1}$$

$$\frac{1}{(k+1)!}\left(\sum_{j=1}^{n} h_j \frac{\partial}{\partial x_j}\right)^{k+1} u(\xi_1, \ldots, \xi_n),$$

где h_j - произвольные приращения соответствующих аргументов, $\xi_j \in [x_j, x_j + h_j]$, функция $u(x_1, \ldots, x_n)$ 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:

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

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

$$\overline{\nabla}_h u_i = \frac{u_{i+1} - u_{i-1}}{h_i + h_{i-1}}, \tag{4}$$

где $\nabla_h^+ u_i$ – правая разность, $\nabla_h^- u_i$ – левая разность, $\overline{\nabla}_h u_i$ – 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:

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

с погрешностью порядка $O(h)$.

Если сетка равномерная, то

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

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 $(x_1, y_1), \ldots, (x_n, y_1), (x_1, y_2), \ldots, (x_n, y_2), (x_1, y_m), \ldots, (x_n, y_m)$.
Тогда для двумерного оператора Лапласа

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

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

$$\mathbb{V}_h u_{i,j} = \frac{2u_{i-1,j}}{h^x_{i-1}(h^x_i + h^x_{i-1}} + \frac{2u_{i,j-1}}{h^y_{j-1}(h^y_j + h^y_{j-1})} + \frac{2u_{i+1,j}}{h^x_i(h^x_i + h^x_{i-1}} + \frac{2u_{i,j+1}}{h^y_j(h^y_j + h^y_{j-1})} - \left(\frac{2}{h^x_{i-1}h^x_i} + \frac{2}{h^y_{j-1}h^y_j}\right) u_{i,j}. \tag{7}$$

На равномерной сетке пятиточечный разностный оператор Лапласа выглядит следующим образом

$$\mathbb{V}_h u_{i,j} = \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h_x^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h_y^2} \tag{8}$$

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.ею 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:

$$\lambda \left.\frac{\partial u}{\partial n}\right|_{S_2} = \theta,$$

$$\lambda \left.\frac{\partial u}{\partial n}\right|_{S_3} + \beta\left(u|_{S_3} - u_\beta\right) = 0.$$

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 $\dfrac{\partial u}{\partial n}$ (которая в этом случае будет равна либо $\pm\dfrac{\partial u}{\partial x}$, либо $\pm\dfrac{\partial u}{\partial y}$) сводятся к одномерным (2) – (4).

Практическая часть

1. Построить прямоугольную сетку в обасти 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 $\mathbf{A}$ и вектора правой части $\mathbf{b}$.
3. Реализовать программу решения двумерной эллиптической задачи методом конечных разностей.
4. Протестировать разработанные программы на полиномах соответствующей степени.
5. Провести исследования порядка аппроксимации реализованного методов для различных задач с неполиномиальными решениями. Сравнить полученный порядок аппроксимации с теоретическим.

Описание программы

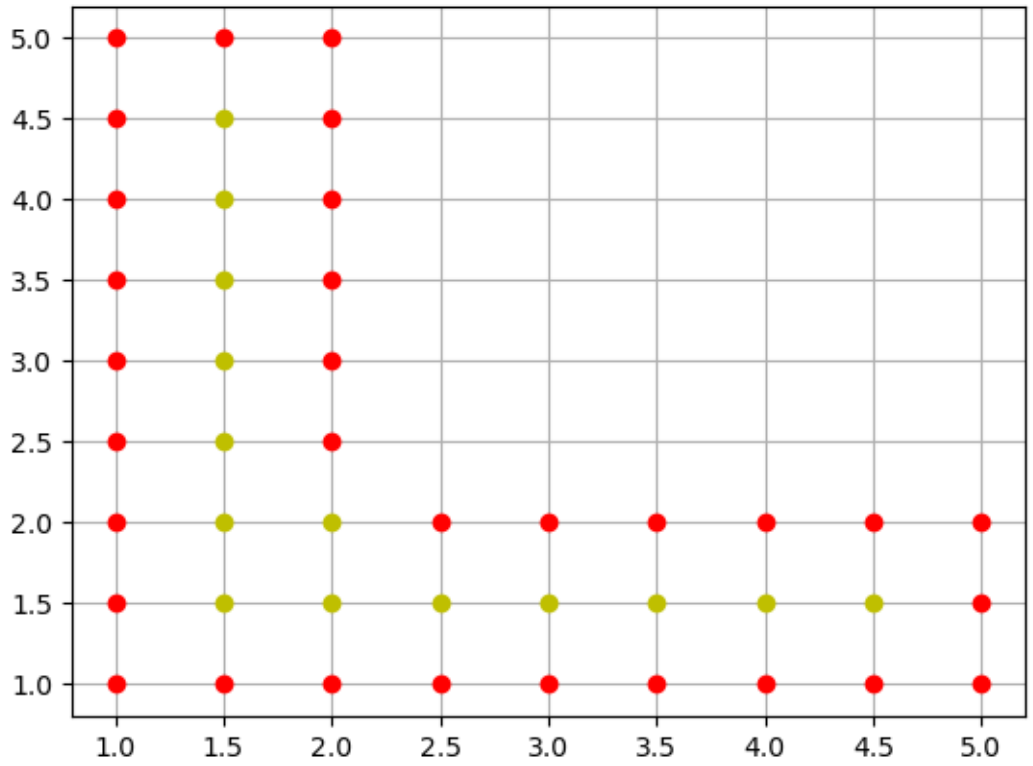
Программа состоит из нескольких модулей:

- Класс MFD, в котором происходят основны 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

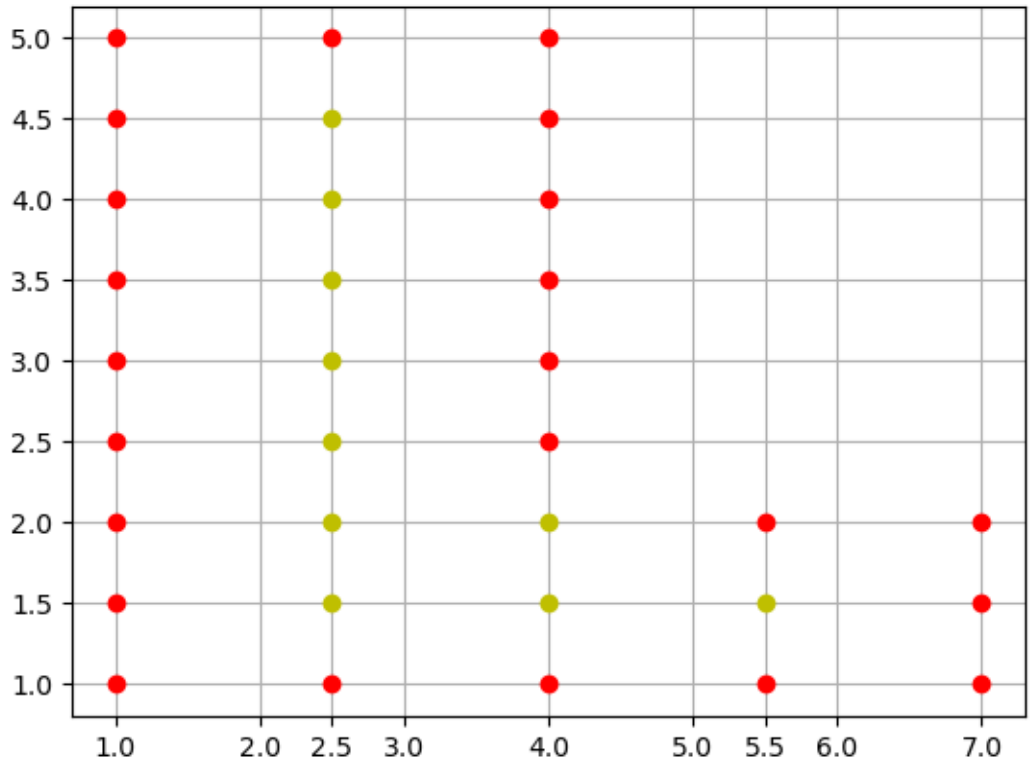
Функция: x
Правая часть: 0,
Коэффициенты : $\lambda = 1$, $\gamma = 0$,
Сетка: равномерная,
Заданы краевые условия 1-го рода на всех границах.



x y	Точное	Численное	Вектор погрешности	Погрешность
1.500 1.500	1.5	1.5000000000000013	1.33E-15	4.02E-16
1.500 2.000	1.5	1.4999999999999998	2.22E-16	
1.500 2.500	1.5	1.5000000000000004	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	2.5000000000000004	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.



x y	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	4.249999945634936	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: $-36x - 24y + 2\,(3x^3 + 2y^3)$,

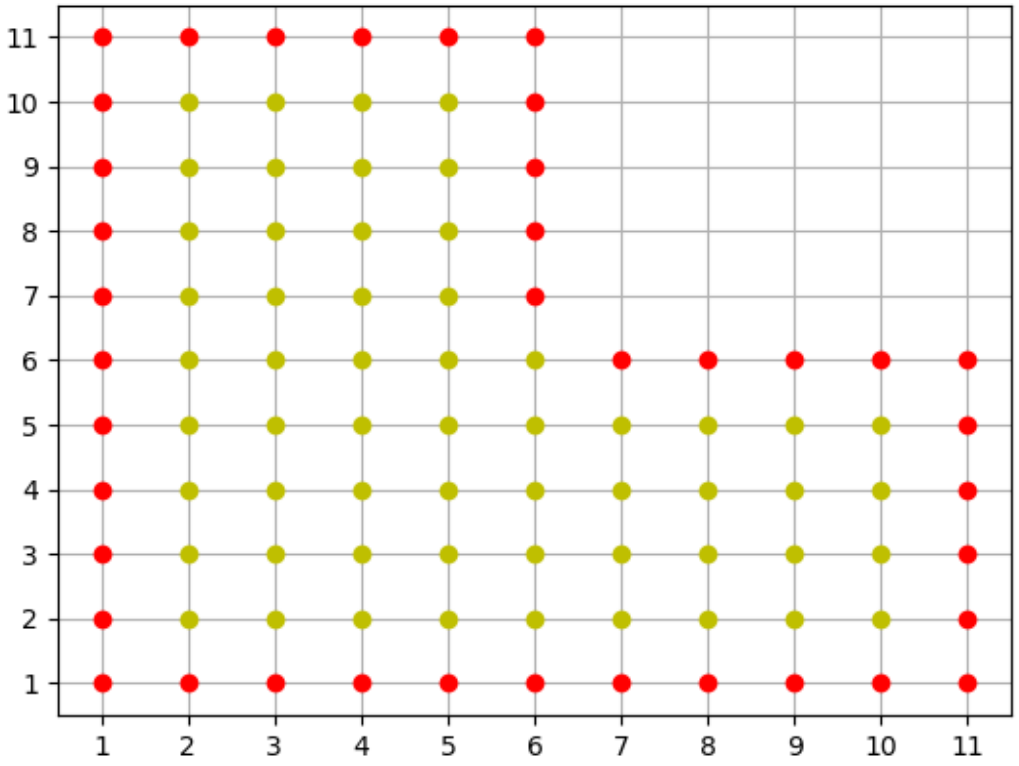
Коэффициенты :

- 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.

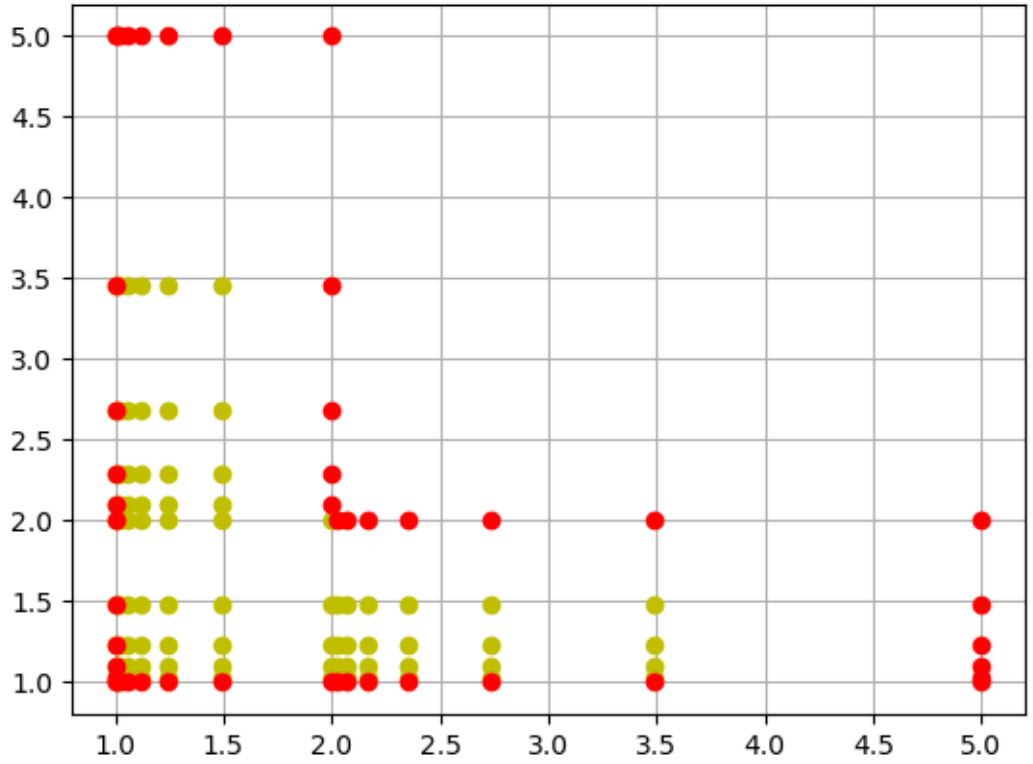


x y	Точное	Численное	Вектор погрешности	Погрешность
2.000 2.000	40	40.593814576817486	5.94E-1	1.43E-1
2.000 3.000	78	78.1546987217267	1.55E-1	
2.000 4.000	152	152.0431892181076	4.32E-2	
2.000 5.000	274	274.012790446567	1.28E-2	
2.000 6.000	456	456.0039632598384	3.96E-3	
2.000 7.000	710	710.0012648282988	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	

x y	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.

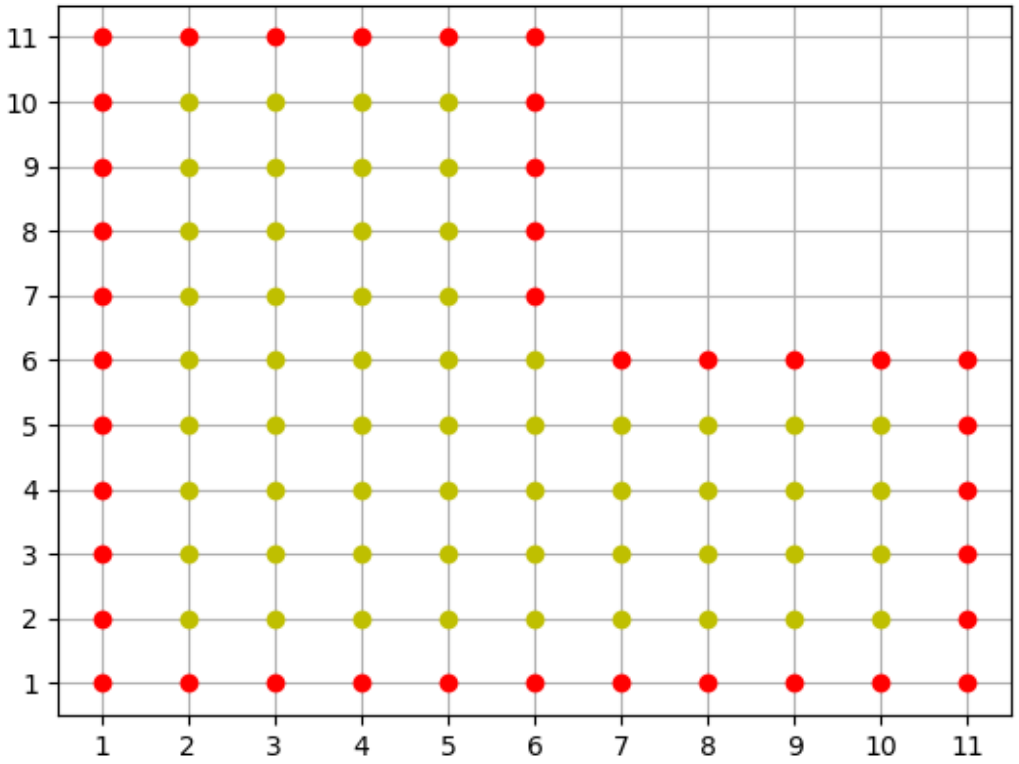


x y	Точное	Численное	Вектор погрешности	Погрешность
1.008 1.032	-0.01644803290	-0.01644803290	8.33E-17	2.92E-15
1.008 1.097	-0.08096416193	-0.08096416193	6.94E-17	
1.008 1.226	-0.20999641999	-0.20999641999	2.78E-17	
1.008 1.484	-0.46806093612	-0.46806093612	0	
1.008 2.000	-0.98418996838	-0.98418996838	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	

x y	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	

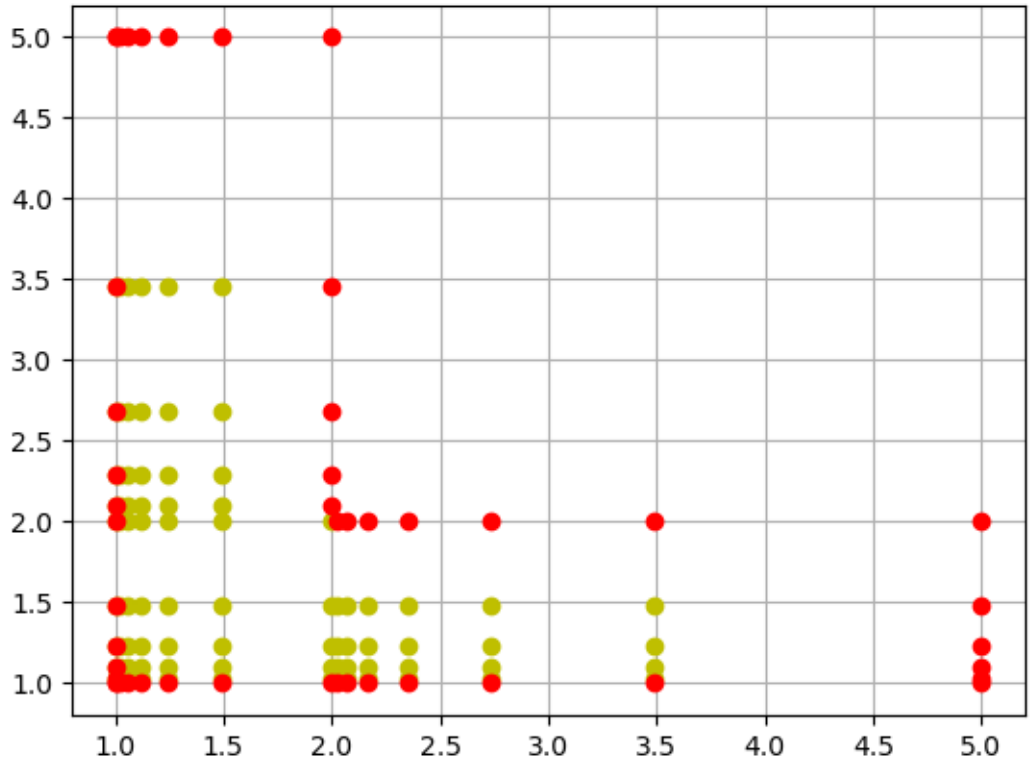
Исследование на определение порядка аппроксимации

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.

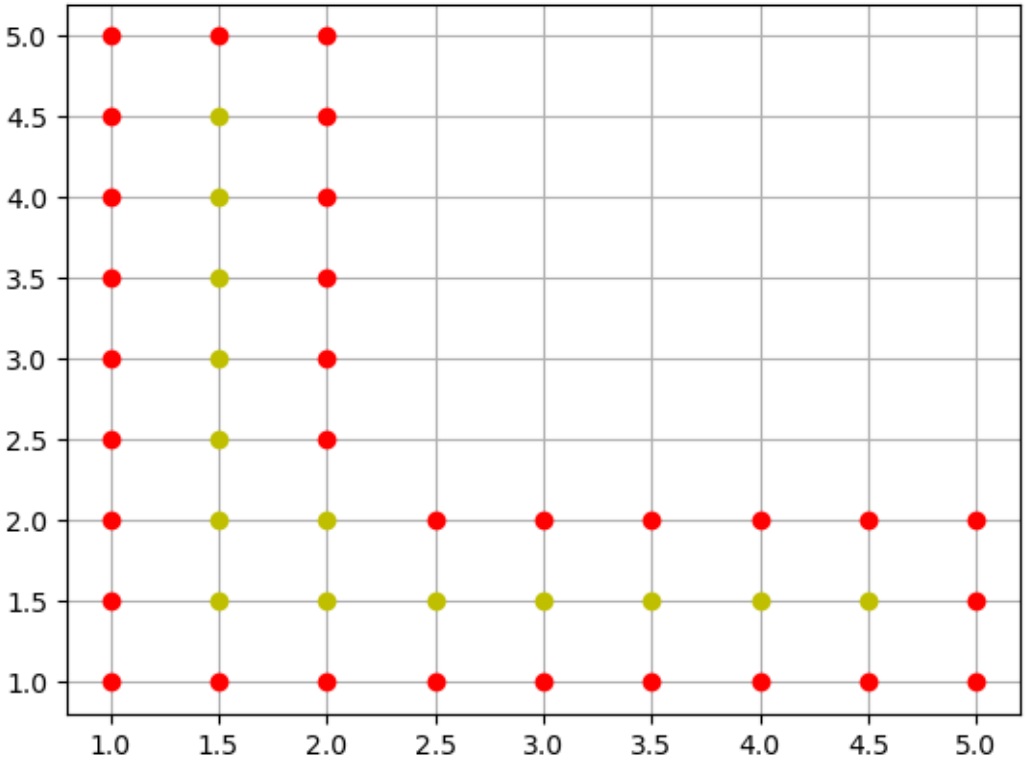


Функция	Относительная погрешность	Относительная невязка
$x + y$	5.07E-16	3.89E-13
$x^2 + 2y^2$	2.62E-16	1.97E-13
$3x^3 + 4y^3$	7.08E-4	1.48E-2
x^4	1.85E-3	3.51E-2

Исследование на определение порядка сходимости

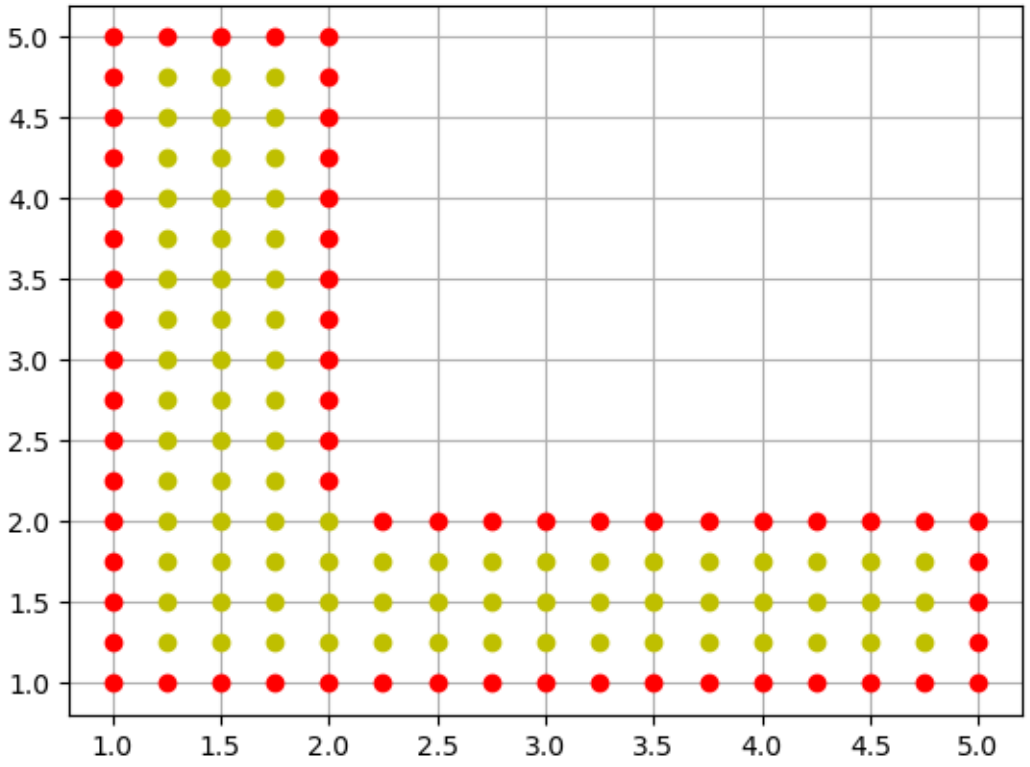
Возьмем неполиномиальную функцию и с помощью нее оценим порядок сходимости метода.
Функция: $\ln(x + y)$
Правая часть: $\dfrac{2}{(x+y)^2}$
Коэффициенты: $\lambda = 1, \gamma = 0$,
Сетка: равномерная,
Заданы краевые условия 1-го рода на всех границах.

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:

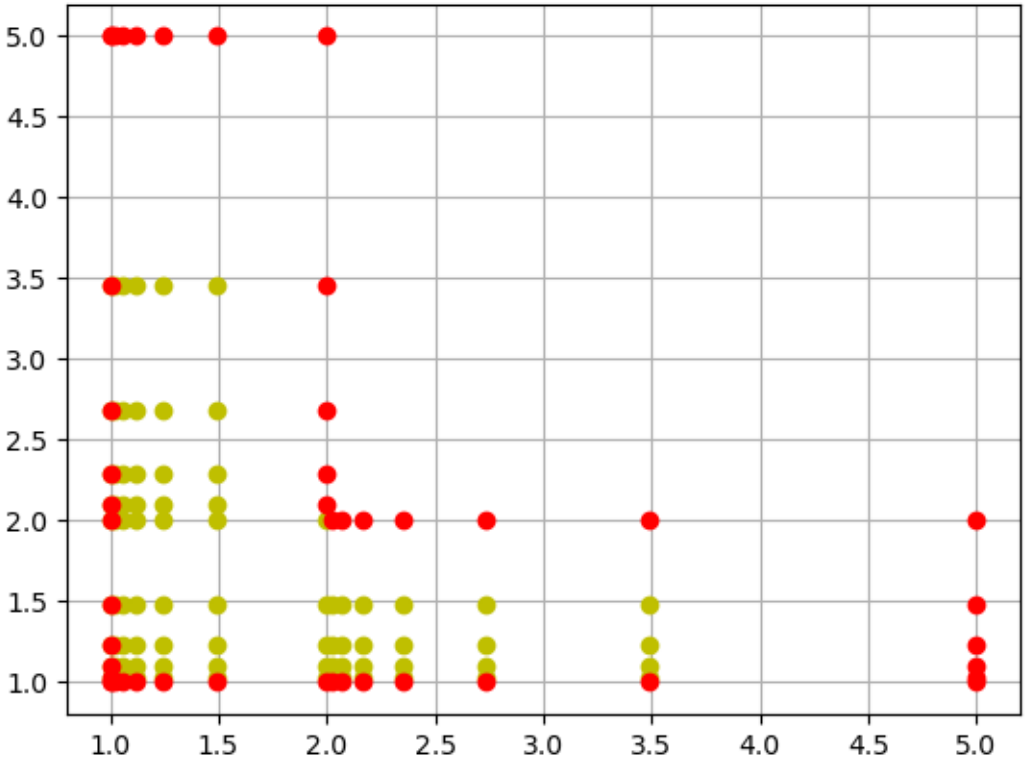


Функция	Погрешность
$\ln(x+y)$	3.47E-5
$4x^4$	5.30E-2

Далее, определим порядок сходимости метода на неравномерной сетке.

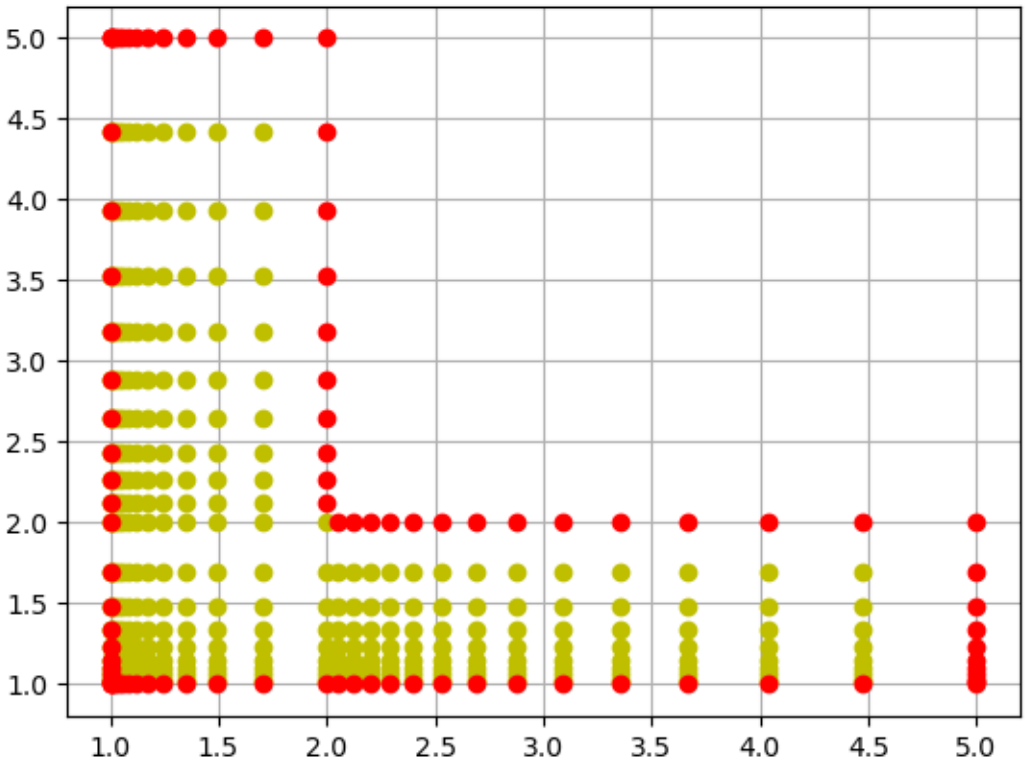
Функция: $3x^3 + 4y^3$,
Правая часть: $\frac{2}{(x+y)^2}$,
Коэффициенты $\lambda = 1, \gamma = 0$,
Сетка: неравномерная,
Заданы краевые условия 1-го рода на всех границах.

Функция: $\ln(x+y)$,
Правая часть: $-18x - 24y$,
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 $\frac{\text{1.65E-4}}{\text{3.47E-5}} \approx 4.755$ раз (в случае неполинома) и упала в $\frac{\text{2.65E-1}}{\text{5.30E-2}} = 5$ 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 $\frac{\text{2.42-4}}{\text{4.82E-5}} \approx 5.02$ раз (в случае неполинома) и упала в $\frac{\text{2.12E-1}}{\text{3.89E-2}} \approx 5.45$ 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 теоритические. 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

```
using eMF_1;

GridFactory gridFactory = new();
// MFD mfd = new(gridFactory.CreateGrid(GridType.Irregular,
 ↪  "grid/grid(irregular).txt"), "boundaries.txt");
MFD mfd = new(gridFactory.CreateGrid(GridType.Nested,
 ↪  "grid/grid(nested).txt"), "boundaries.txt");
// MFD mfd = new(gridFactory.CreateGrid(GridType.Regular,
 ↪  "grid/grid(regular).txt"), "boundaries.txt");
// mfd.SetTest(new FirstTest());  x, λ = 1, γ = 0
// mfd.SetTest(new SecondTest()); x^2 − y, λ = 0.5, γ = 1
// mfd.SetTest(new ThirdTest());  3x^3 + 2y^3, 1 area: λ = γ = 0.5, 2 area : λ = γ = 2
// mfd.SetTest(new FourthTest()); ln(x + y), λ = 1, γ = 1
// mfd.SetTest(new FifthTest());  4x^4, λ = 1, γ = 0
// mfd.SetTest(new SixthTest());  4x^4 + 2y^4, λ = 1, γ = 0
// mfd.SetTest(new SeventhTest());e^x + y, λ = 1, γ = 1
// mfd.SetTest(new EighthTest()); x^3 + y, λ = 1, γ = 0
mfd.SetMethodSolvingSLAE(new GaussSeidel(1000, 1E-14, 1.22));
mfd.Compute();
```

MFD.cs

```
namespace eMF_1;

public enum PointType
{
    Boundary,
    Internal,
    Dummy
}

public enum BoundaryType
{
    None,
    Dirichlet,
    Neumann,
    Mixed
}

public enum GridType
{
    Regular,
    Irregular,
    Nested
}

public enum NormalType
{
    LeftX,
    RightX,
    UpperY,
    BottomY
}

public class MFD
{
    private Grid _grid;
    private DiagMatrix _matrix;
    private ITest _test;
    private ISolver _solver;
    private Boundary[] _boundaries;
    private double[] _q;
    private double[] _pr;
    private double _beta;
    public double[] Weights
        => _q;

    public MFD(Grid grid, string boundaryPath)
    {
        try
        {
            using (var sr = new StreamReader(boundaryPath))
            {
                _beta = double.Parse(sr.ReadLine());
                _boundaries = sr.ReadToEnd().Split("\n")
                .Select(str => Boundary.BoundaryParse(str)).ToArray();
            }

            _grid = grid;
        }
        catch (Exception ex)
```

```
60          {
61              Console.WriteLine(ex.Message);
62          }
63      }
64
65      public void SetTest(ITest test)
66          => _test = test;
67
68      public void SetMethodSolvingSLAE(ISolver solver)
69          => _solver = solver;
70
71      public void Compute()
72      {
73          try
74          {
75              if (_test is null)
76                  throw new Exception("Set the test!");
77
78              if (_solver is null)
79                  throw new Exception("Set the method solving SLAE!");
80
81              _grid.Build();
82              _grid.AssignBoundaryConditions(_boundaries);
83              Init();
84              BuildMatrix();
85              _q = _solver.Compute(_matrix, _pr);
86          }
87          catch (Exception ex)
88          {
89              Console.WriteLine(ex.Message);
90          }
91      }
92
93      private void Init()
94      {
95          _matrix = new(_grid.Points.Count, (_grid.AllLinesX.Count >
            ↪  _grid.AllLinesX.Count) ?
96          _grid.AllLinesX.Count - 2 : _grid.AllLinesY.Count - 2);
97          _pr = new double[_matrix.Size];
98          _q = new double[_matrix.Size];
99      }
100
101     private void BuildMatrix()
102     {
103         double hx, hy, hix, hiy, hi, h = 1E-12;
104         double lambda, gamma;
105         double us, ubeta;
106         double leftDerivative, rightDerivative;
107         NormalType normalType;
108
109         for (int i = 0; i < _grid.Points.Count; i++)
110         {
111             switch (_grid.Points[i].PointType)
112             {
113                 case PointType.Boundary:
114
115                     switch (_grid.Points[i].BoundaryType)
116                     {
117                         case BoundaryType.Dirichlet:
118
```

```
                    _matrix.Diags[0][i] = 1;
                    _pr[i] = _test.U(_grid.Points[i]);

                    break;

                case BoundaryType.Neumann:

                    lambda = _grid.Areas[_grid.Points[i].AreaNumber]
                    ↪  .Item2;

                    normalType = _grid.Normal(_grid.Points[i]);

                    switch (normalType)
                    {
                        case NormalType.LeftX:

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

                            break;

                        case NormalType.BottomY:

                            hi = _grid.AllLinesY[_grid.Points[i].J +
                            ↪  1] -
                            ↪  _grid.AllLinesY[_grid.Points[i].J];
                            _matrix.Diags[0][i] = lambda / hi;
                            _matrix.Diags[3][i] = -lambda / hi;
                            _pr[i] = -lambda *
                            ↪  RightDerivativeY(_grid.Points[i], h);

                            break;

                        case NormalType.RightX:

                            hi = _grid.AllLinesX[_grid.Points[i].I]
                            ↪  - _grid.AllLinesX[_grid.Points[i].I
                            ↪  - 1];
                            _matrix.Diags[0][i] = lambda / hi;
                            _matrix.Diags[2][i + _matrix.Indexes[2]]
                            ↪  = -lambda / hi;
                            _pr[i] = lambda *
                            ↪  LeftDerivativeX(_grid.Points[i], h);

                            break;

                        case NormalType.UpperY:

                            hi = _grid.AllLinesY[_grid.Points[i].J]
                            ↪  - _grid.AllLinesY[_grid.Points[i].J
                            ↪  - 1];
                            _matrix.Diags[0][i] = lambda / hi;
                            _matrix.Diags[1][i + _matrix.Indexes[1]]
                            ↪  = -lambda / hi;
```

```
                        _pr[i] = lambda *
                        ↪  LeftDerivativeY(_grid.Points[i], h);

                        break;

                    default:
                        throw new ArgumentOutOfRangeException(na
                        ↪  meof(normalType),
                        $"This type of normal does not exist:
                        ↪  {normalType}");
                }

                break;

            case BoundaryType.Mixed:

                lambda = _grid.Areas[_grid.Points[i].AreaNumber]
                ↪  .Item2;

                normalType = _grid.Normal(_grid.Points[i]);
                us = _test.U(_grid.Points[i]);

                switch (normalType)
                {
                    case NormalType.LeftX:

                        hi = _grid.AllLinesX[_grid.Points[i].I +
                        ↪  1] -
                        ↪  _grid.AllLinesX[_grid.Points[i].I];
                        rightDerivative =
                        ↪  RightDerivativeX(_grid.Points[i], h);
                        ubeta = -lambda * rightDerivative /
                        ↪  _beta + us;
                        _matrix.Diags[0][i] = lambda / hi +
                        ↪  _beta;
                        _matrix.Diags[4][i] = -lambda / hi;
                        _pr[i] = -lambda * rightDerivative +
                        ↪  _beta * (us - ubeta) + _beta * ubeta;

                        break;

                    case NormalType.BottomY:

                        hi = _grid.AllLinesY[_grid.Points[i].J +
                        ↪  1] -
                        ↪  _grid.AllLinesY[_grid.Points[i].J];
                        rightDerivative =
                        ↪  RightDerivativeY(_grid.Points[i], h);
                        ubeta = -lambda * rightDerivative /
                        ↪  _beta + us;
                        _matrix.Diags[0][i] = lambda / hi +
                        ↪  _beta;
                        _matrix.Diags[3][i] = -lambda / hi;
                        _pr[i] = -lambda * rightDerivative +
                        ↪  _beta * (us - ubeta) + _beta * ubeta;

                        break;

                    case NormalType.RightX:
```

```
                                hi = _grid.AllLinesX[_grid.Points[i].I]
                                ↪  - _grid.AllLinesX[_grid.Points[i].I
                                ↪  - 1];
                                leftDerivative =
                                ↪  LeftDerivativeX(_grid.Points[i], h);
                                ubeta = lambda * leftDerivative / _beta
                                ↪  + us;
                                _matrix.Diags[0][i] = lambda / hi +
                                ↪  _beta;
                                _matrix.Diags[2][i + _matrix.Indexes[2]]
                                ↪  = -lambda / hi;
                                _pr[i] = lambda * leftDerivative + _beta
                                ↪  * (us - ubeta) + _beta * ubeta;

                                break;

                            case NormalType.UpperY:

                                hi = _grid.AllLinesY[_grid.Points[i].J]
                                ↪  - _grid.AllLinesY[_grid.Points[i].J
                                ↪  - 1];
                                leftDerivative =
                                ↪  LeftDerivativeY(_grid.Points[i], h);
                                ubeta = lambda * leftDerivative / _beta
                                ↪  + us;
                                _matrix.Diags[0][i] = lambda / hi +
                                ↪  _beta;
                                _matrix.Diags[1][i + _matrix.Indexes[1]]
                                ↪  = -lambda / hi;
                                _pr[i] = lambda * leftDerivative + _beta
                                ↪  * (us - ubeta) + _beta * ubeta;

                                break;

                            default:
                                throw new ArgumentOutOfRangeException(na
                                ↪  meof(normalType),
                                $"This type of normal does not exist:
                                ↪  {normalType}");
                        }

                        break;


                    default:
                        throw new ArgumentOutOfRangeException(nameof(Bou
                        ↪  ndaryType),
                        $"This type of boundary does not exist:
                        ↪  {_grid.Points[i].BoundaryType}");
                }

                break;

            case PointType.Internal:

                hx = _grid.AllLinesX[_grid.Points[i].I + 1] -
                ↪  _grid.AllLinesX[_grid.Points[i].I];
                hy = _grid.AllLinesY[_grid.Points[i].J + 1] -
                ↪  _grid.AllLinesY[_grid.Points[i].J];

```

```
                    (lambda, gamma) =
                    (_grid.Areas[_grid.Points[i].AreaNumber].Item2,
                    _grid.Areas[_grid.Points[i].AreaNumber].Item3);

                    _pr[i] = _test.F(_grid.Points[i]);

                    if (_grid is RegularGrid)
                    {
                        _matrix.Diags[0][i] = lambda * (2.0 / (hx * hx) +
                        ↪  2.0 / (hy * hy)) + gamma;
                        _matrix.Diags[3][i] = -lambda / (hy * hy);
                        _matrix.Diags[4][i] = -lambda / (hx * hx);
                        _matrix.Diags[1][i + _matrix.Indexes[1]] = -lambda /
                        ↪  (hy * hy);
                        _matrix.Diags[2][i + _matrix.Indexes[2]] = -lambda /
                        ↪  (hx * hx);
                    }
                    else
                    {
                        hix = _grid.AllLinesX[_grid.Points[i].I] -
                        ↪  _grid.AllLinesX[_grid.Points[i].I - 1];
                        hiy = _grid.AllLinesY[_grid.Points[i].J] -
                        ↪  _grid.AllLinesY[_grid.Points[i].J - 1];

                        _matrix.Diags[0][i] = lambda * (2.0 / (hix * hx) +
                        ↪  2.0 / (hiy * hy)) + gamma;
                        _matrix.Diags[2][i + _matrix.Indexes[2]] = -lambda *
                        ↪  2.0 / (hix * (hx + hix));
                        _matrix.Diags[1][i + _matrix.Indexes[1]] = -lambda *
                        ↪  2.0 / (hiy * (hy + hiy));
                        _matrix.Diags[4][i] = -lambda * 2.0 / (hx * (hx +
                        ↪  hix));
                        _matrix.Diags[3][i] = -lambda * 2.0 / (hy * (hy +
                        ↪  hiy));
                    }

                    break;

                case PointType.Dummy:

                    _matrix.Diags[0][i] = 1;
                    _pr[i] = 0;

                    break;

                default:
                    throw new ArgumentOutOfRangeException(nameof(PointType),
                    $"This type of point does not exist:
                    ↪  {_grid.Points[i].PointType}");
            }
        }
    }

    private double LeftDerivativeX(Point2D point, double h)
        => (_test.U(point) - _test.U(point - (h, 0))) / h;

    private double LeftDerivativeY(Point2D point, double h)
        => (_test.U(point) - _test.U(point - (0, h))) / h;

```

```
296     private double RightDerivativeX(Point2D point, double h)
297         => (_test.U(point + (h, 0)) - _test.U(point)) / h;
298
299     private double RightDerivativeY(Point2D point, double h)
300         => (_test.U(point + (0, h)) - _test.U(point)) / h;
301 }
```

DiagMatrix.cs

```
1  namespace eMF_1;
2
3  public class DiagMatrix
4  {
5      public double[][] Diags { get; set; }
6      public int[] Indexes { get; init; }
7      public int Size { get; init; }
8      public int ZeroDiags { get; init; }
9
10     public DiagMatrix(int countPoints, int zeroDiags)
11     {
12         Size = countPoints;
13         ZeroDiags = zeroDiags;
14         Diags = new double[5][];
15         Diags[0] = new double[countPoints];
16         Diags[1] = new double[countPoints - 1];
17         Diags[2] = new double[countPoints - zeroDiags - 2];
18         Diags[3] = new double[countPoints - 1];
19         Diags[4] = new double[countPoints - zeroDiags - 2];
20         Indexes = new int[] { 0, -1, -2 - zeroDiags, 1, 2 + zeroDiags };
21     }
22 }
```

Point2D.cs

```
1  namespace eMF_1;
2
3  public class Point2D
4  {
5      public double X { get; init; }
6      public double Y { get; init; }
7      public int I { get; init; }
8      public int J { get; init; }
9      public PointType PointType { get; init; }
10     public BoundaryType BoundaryType { get; set; } = BoundaryType.None;
11     public int AreaNumber { get; set; }
12
13     public Point2D(double x, double y, int i, int j, PointType pointType)
14     {
15         X = x;
16         Y = y;
17         I = i;
18         J = j;
19         PointType = pointType;
20     }
21
22     public static Point2D Parse(string pointStr)
23     {
24         var data = pointStr.Split();
25         Point2D point = new(double.Parse(data[0]), double.Parse(data[1]),
26         int.Parse(data[2]), int.Parse(data[3]),
            ↪  (PointType)Enum.Parse(typeof(PointType), data[4]));
```

```
        return point;
    }

    public static Point2D operator +(Point2D point, (double, double) value)
        => new(point.X + value.Item1, point.Y + value.Item2, point.I,
        ↪  point.J, point.PointType);

    public static Point2D operator -(Point2D point, (double, double) value)
        => new(point.X - value.Item1, point.Y - value.Item2, point.I,
        ↪  point.J, point.PointType);

    public override string ToString()
        => $"{X} {Y}";
}
```

GridFactory.cs

```
namespace eMF_1;

public class GridFactory
{
    public Grid CreateGrid(GridType gridType, string path)
    {
        return gridType switch
        {
            GridType.Regular => new RegularGrid(path),

            GridType.Irregular => new IrregularGrid(path),

            GridType.Nested => new NestedGrid(path),

            _ => throw new ArgumentOutOfRangeException(nameof(gridType),
            $"This type of grid does not exist: {gridType}")
        };
    }
}
```

Grid.cs

```
namespace eMF_1;

public abstract class Grid
{
    public abstract double[] LinesX { get; init; }
    public abstract double[] LinesY { get; init; }
    public abstract List<double> AllLinesX { get; }
    public abstract List<double> AllLinesY { get; }
    public abstract List<Point2D> Points { get; }
    public abstract (int, double, double, int, int, int, int)[] Areas { get;
    ↪  init; }

    public abstract void Build();

    public NormalType Normal(Point2D point)
    {
        NormalType normalType = default(NormalType);

        if (point.PointType ≠ PointType.Boundary)
            throw new Exception("To determine the normal to the boundary,
            ↪  the point needs the boundary!");
```

```
20
21          else if (point.X ⩾ LinesX[0] && point.X ⩽ LinesX[1] && point.Y ==
            ↪  LinesY[2])
22          {
23              normalType = NormalType.UpperY;
24              return normalType;
25          }
26          else if (point.X ⩾ LinesX[1] && point.X ⩽ LinesX[2] && point.Y ==
            ↪  LinesY[1])
27          {
28              normalType = NormalType.UpperY;
29              return normalType;
30          }
31          else if (point.X == LinesX[0] && point.Y ⩾ LinesY[0] && point.Y ⩽
            ↪  LinesY[2])
32          {
33              normalType = NormalType.LeftX;
34              return normalType;
35          }
36          else if (point.X == LinesX[1] && point.Y ⩾ LinesY[1] && point.Y ⩽
            ↪  LinesY[2])
37          {
38              normalType = NormalType.RightX;
39              return normalType;
40          }
41          else if (point.X == LinesX[2] && point.Y ⩾ LinesY[0] && point.Y ⩽
            ↪  LinesY[1])
42          {
43              normalType = NormalType.RightX;
44              return normalType;
45          }
46          else if (point.X ⩾ LinesX[0] && point.X ⩽ LinesX[2] && point.Y ==
            ↪  LinesY[0])
47          {
48              normalType = NormalType.BottomY;
49              return normalType;
50          }
51          else
52              return normalType;
53
54      }
55
56      protected PointType PointsTypes(double x, double y)
57      {
58          double eps = 1E-14;
59
60          if ((x > LinesX[0] && x < LinesX[2] && y > LinesY[0] && y <
            ↪  LinesY[1])
61          || (x > LinesX[0] && x < LinesX[1] && y > LinesY[0] && y < LinesY[2])
62          || (x == LinesX[1] && y == LinesY[1]))
63              return PointType.Internal;
64
65          for (int i = 0; i < AllLinesX.Count; i++)
66              if ((Math.Abs(x - AllLinesX[i]) < eps && Math.Abs(y - LinesY[0])
                ↪  < eps)
67              || (Math.Abs(x - AllLinesX[i]) < eps && Math.Abs(y - LinesY[1])
                ↪  < eps && x ⩾ LinesX[1])
68              || (Math.Abs(x - AllLinesX[i]) < eps && Math.Abs(y - LinesY[2])
                ↪  < eps && x ⩽ LinesX[1]))
```

```
69                  return PointType.Boundary;
70
71              for (int i = 0; i < AllLinesY.Count; i++)
72                  if ((Math.Abs(y - AllLinesY[i]) < eps && Math.Abs(x - LinesX[0])
                    ↪  < eps)
73                  || (Math.Abs(y - AllLinesY[i]) < eps && Math.Abs(x - LinesX[1])
                    ↪  < eps && y ⩾ LinesY[1])
74                  || (Math.Abs(y - AllLinesY[i]) < eps && Math.Abs(x - LinesX[2])
                    ↪  < eps && y ⩽ LinesY[1]))
75                  return PointType.Boundary;
76
77              return PointType.Dummy;
78          }
79
80          protected void SetAreaNumber()
81          {
82              for (int i = 0; i < Points.Count; i++)
83                  for (int iArea = 0; iArea < Areas.Length; iArea++)
84                  {
85                      if (Points[i].X ⩾ Areas[iArea].Item4 && Points[i].X ⩽
                        ↪  Areas[iArea].Item5
86                      && Points[i].Y ⩾ Areas[iArea].Item6 && Points[i].Y ⩽
                        ↪  Areas[iArea].Item7)
87                      {
88                          Points[i].AreaNumber = Areas[iArea].Item1;
89                      }
90                  }
91          }
92
93          public void AssignBoundaryConditions(Boundary[] boundaries)
94          {
95              foreach (var point in Points.Where(point ⇒ point.PointType ==
                ↪  PointType.Boundary))
96                  for (int k = 0; k < boundaries.Length; k++)
97                  {
98                      if (point.X ⩾ LinesX[boundaries[k].X1] && point.X ⩽
                        ↪  LinesX[boundaries[k].X2]
99                      && point.Y ⩾ LinesY[boundaries[k].Y1] && point.Y ⩽
                        ↪  LinesY[boundaries[k].Y2])
100                     {
101                         point.BoundaryType = boundaries[k].BoundaryType;
102                         break;
103                     }
104                 }
105         }
106
107         protected void WriteToFilePoints()
108         {
109             using (var sw = new StreamWriter("points/boundaryPoints.txt"))
110             {
111                 Points.ForEach(x ⇒ { if (x.PointType == PointType.Boundary)
                    ↪  sw.WriteLine(x); });
112             }
113
114             using (var sw = new StreamWriter("points/internalPoints.txt"))
115             {
116                 Points.ForEach(x ⇒ { if (x.PointType == PointType.Internal)
                    ↪  sw.WriteLine(x); });
117             }
```

```
118
119         using (var sw = new StreamWriter("points/dummyPoints.txt"))
120         {
121             Points.ForEach(x => { if (x.PointType == PointType.Dummy)
                ↪  sw.WriteLine(x); });
122         }
123     }
124 }
```

RegularGrid.cs

```
1  namespace eMF_1;
2
3  public class RegularGrid : Grid
4  {
5      private List<double> _allLinesX;
6      private List<double> _allLinesY;
7      private List<Point2D> _points;
8      public override double[] LinesX { get; init; }
9      public override double[] LinesY { get; init; }
10     public override List<double> AllLinesX
11         => _allLinesX;
12     public override List<double> AllLinesY
13         => _allLinesY;
14     public override List<Point2D> Points
15         => _points;
16     public override (int, double, double, int, int, int, int)[] Areas { get;
       ↪  init; }
17     public int SplitsX { get; init; }
18     public int SplitsY { get; init; }
19
20     public RegularGrid(string path)
21     {
22         try
23         {
24             using (var sr = new StreamReader(path))
25             {
26                 LinesX = sr.ReadLine().Split().Select(value =>
                    ↪  double.Parse(value)).ToArray();
27                 LinesY = sr.ReadLine().Split().Select(value =>
                    ↪  double.Parse(value)).ToArray();
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]), double.Parse(value[2]),
32                 int.Parse(value[3]), int.Parse(value[4]),
                    ↪  int.Parse(value[5]), int.Parse(value[6]))).ToArray();
33             }
34
35             _allLinesX = new();
36             _allLinesY = new();
37             _points = new();
38         }
39         catch (Exception ex)
40         {
41             Console.WriteLine(ex.Message);
42         }
43     }
44
```

```
    public override void Build()
    {
        double h;
        double lenght = LinesX.Last() - LinesX.First();

        h = lenght / SplitsX;

        _allLinesX.Add(LinesX.First());

        while (Math.Round(_allLinesX.Last() + h, 1) < LinesX.Last())
            _allLinesX.Add(_allLinesX.Last() + h);

        _allLinesX = _allLinesX.Union(LinesX).OrderBy(value ⇒
        ↪  value).ToList();

        lenght = LinesY.Last() - LinesY.First();

        h = lenght / SplitsY;

        _allLinesY.Add(LinesY.First());

        while (Math.Round(_allLinesY.Last() + h, 1) < LinesY.Last())
            _allLinesY.Add(_allLinesY.Last() + h);

        _allLinesY = _allLinesY.Union(LinesY).OrderBy(value ⇒
        ↪  value).ToList();

        for (int i = 0; i < _allLinesX.Count; i++)
            for (int j = 0; j < _allLinesY.Count; j++)
                _points.Add(new(_allLinesX[i], _allLinesY[j], i, j,
                PointsTypes(_allLinesX[i], _allLinesY[j])));

        SetAreaNumber();
        WriteToFilePoints();
    }
}
```

IrregularGrid.cs

```
namespace eMF_1;

public class IrregularGrid : Grid
{
    private List<double> _allLinesX;
    private List<double> _allLinesY;
    private List<Point2D> _points;
    public override double[] LinesX { get; init; }
    public override double[] LinesY { get; init; }
    public override List<Point2D> Points
     ⇒ _points;
    public override List<double> AllLinesX
        ⇒ _allLinesX;
    public override List<double> AllLinesY
        ⇒ _allLinesY;
    public override (int, double, double, int, int, int, int)[] Areas { get;
    ↪  init; }
    public int[] SplitsX { get; init; }
    public int[] SplitsY { get; init; }
    public double[] KX { get; init; }
    public double[] KY { get; init; }
```

```
21
22      public IrregularGrid(string path)
23      {
24          try
25          {
26              using (var sr = new StreamReader(path))
27              {
28                  LinesX = sr.ReadLine().Split().Select(value ⇒
                    ↪  double.Parse(value)).ToArray();
29                  LinesY = sr.ReadLine().Split().Select(value ⇒
                    ↪  double.Parse(value)).ToArray();
30                  SplitsX = sr.ReadLine().Split().Select(value ⇒
                    ↪  int.Parse(value)).ToArray();
31                  SplitsY = sr.ReadLine().Split().Select(value ⇒
                    ↪  int.Parse(value)).ToArray();
32                  KX = sr.ReadLine().Split().Select(value ⇒
                    ↪  double.Parse(value)).ToArray();
33                  KY = sr.ReadLine().Split().Select(value ⇒
                    ↪  double.Parse(value)).ToArray();
34                  Areas = sr.ReadToEnd().Split("\n").Select(row ⇒ row.Split())
35                  .Select(value ⇒ (int.Parse(value[0]),
                    ↪  double.Parse(value[1]), double.Parse(value[2]),
36                  int.Parse(value[3]), int.Parse(value[4]),
                    ↪  int.Parse(value[5]), int.Parse(value[6]))).ToArray();
37
38              }
39
40              _allLinesX = new();
41              _allLinesY = new();
42              _points = new();
43          }
44          catch (Exception ex)
45          {
46              Console.WriteLine(ex.Message);
47          }
48      }
49
50      public override void Build()
51      {
52          for (int i = 0; i < LinesX.Length - 1; i++)
53          {
54              double h;
55              double sum = 0;
56              double lenght = LinesX[i + 1] - LinesX[i];
57
58              for (int k = 0; k < SplitsX[i]; k++)
59                  sum += Math.Pow(KX[i], k);
60
61              h = lenght / sum;
62
63              _allLinesX.Add(LinesX[i]);
64
65              while (Math.Round(_allLinesX.Last() + h, 1) < LinesX[i + 1])
66              {
67                  _allLinesX.Add(_allLinesX.Last() + h);
68
69                  h *= KX[i];
70              }
71
```

```
            sum = 0;

        }

        _allLinesX.Add(LinesX.Last());

        for (int i = 0; i < LinesY.Length - 1; i++)
        {
            double h;
            double sum = 0;
            double lenght = LinesY[i + 1] - LinesY[i];

            for (int k = 0; k < SplitsY[i]; k++)
                sum += Math.Pow(KY[i], k);

            h = lenght / sum;

            _allLinesY.Add(LinesY[i]);

            while (Math.Round(_allLinesY.Last() + h, 1) < LinesY[i + 1])
            {
                _allLinesY.Add(_allLinesY.Last() + h);

                h *= KY[i];
            }

            sum = 0;
        }

        _allLinesY.Add(LinesY.Last());

        for (int i = 0; i < _allLinesX.Count; i++)
            for (int j = 0; j < _allLinesY.Count; j++)
                _points.Add(new(_allLinesX[i], _allLinesY[j], i, j,
                PointsTypes(_allLinesX[i], _allLinesY[j])));

        SetAreaNumber();
        WriteToFilePoints();
    }
}
```

NestedGrid.cs

```
namespace eMF_1;

public class NestedGrid : Grid
{
    private List<double> _allLinesX;
    private List<double> _allLinesY;
    private List<Point2D> _points;
    public override double[] LinesX { get; init; }
    public override double[] LinesY { get; init; }
    public override List<Point2D> Points
     => _points;
    public override List<double> AllLinesX
        => _allLinesX;
    public override List<double> AllLinesY
        => _allLinesY;
    public override (int, double, double, int, int, int, int)[] Areas { get;
    ↪  init; }
    public int[] SplitsX { get; init; }
```

```
    public int[] SplitsY { get; init; }
    public double[] KX { get; init; }
    public double[] KY { get; init; }

    public NestedGrid(string path)
    {
        try
        {
            using (var sr = new StreamReader(path))
            {
                LinesX = sr.ReadLine().Split().Select(value ⇒
                ↪  double.Parse(value)).ToArray();
                LinesY = sr.ReadLine().Split().Select(value ⇒
                ↪  double.Parse(value)).ToArray();
                SplitsX = sr.ReadLine().Split().Select(value ⇒
                ↪  int.Parse(value)).ToArray();
                SplitsY = sr.ReadLine().Split().Select(value ⇒
                ↪  int.Parse(value)).ToArray();
                KX = sr.ReadLine().Split().Select(value ⇒
                ↪  double.Parse(value)).ToArray();
                KY = sr.ReadLine().Split().Select(value ⇒
                ↪  double.Parse(value)).ToArray();
                Areas = sr.ReadToEnd().Split("\n").Select(row ⇒ row.Split())
                .Select(value ⇒ (int.Parse(value[0]),
                ↪  double.Parse(value[1]), double.Parse(value[2]),
                int.Parse(value[3]), int.Parse(value[4]),
                ↪  int.Parse(value[5]), int.Parse(value[6]))).ToArray();

            }

            _allLinesX = new();
            _allLinesY = new();
            _points = new();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
    }

    public override void Build()
    {
        for (int i = 0; i < LinesX.Length - 1; i++)
        {
            double h;
            double sum = 0;
            double lenght = LinesX[i + 1] - LinesX[i];

            for (int j = 0; j < KX.Length; j++)
                KX[j] = Math.Sqrt(KX[j]);

            for (int k = 0; k < SplitsX[i]; k++)
                sum += Math.Pow(KX[i], k);

            h = lenght / sum;

            _allLinesX.Add(LinesX[i]);

            while (Math.Round(_allLinesX.Last() + h, 1) < LinesX[i + 1])
```

```
            {
                _allLinesX.Add(_allLinesX.Last() + h);

                h *= KX[i];
            }

            sum = 0;

        }

        _allLinesX.Add(LinesX.Last());

        for (int i = 0; i < LinesY.Length - 1; i++)
        {
            double h;
            double sum = 0;
            double lenght = LinesY[i + 1] - LinesY[i];

            for (int j = 0; j < KY.Length; j++)
                KY[j] = Math.Sqrt(KY[j]);

            for (int k = 0; k < SplitsY[i]; k++)
                sum += Math.Pow(KY[i], k);

            h = lenght / sum;

            _allLinesY.Add(LinesY[i]);

            while (Math.Round(_allLinesY.Last() + h, 1) < LinesY[i + 1])
            {
                _allLinesY.Add(_allLinesY.Last() + h);

                h *= KY[i];
            }

            sum = 0;
        }

        _allLinesY.Add(LinesY.Last());

        for (int i = 0; i < _allLinesX.Count; i++)
            for (int j = 0; j < _allLinesY.Count; j++)
                _points.Add(new(_allLinesX[i], _allLinesY[j], i, j,
                PointsTypes(_allLinesX[i], _allLinesY[j])));

        SetAreaNumber();
        WriteToFilePoints();
    }
}
```

Boundary.cs

```
namespace eMF_1;

public readonly record struct Boundary(BoundaryType BoundaryType, int X1,
↪  int X2, int Y1, int Y2)
{
    public static Boundary BoundaryParse(string boundaryStr)
    {
        var data = boundaryStr.Split();
```

```
        Boundary boundary =
        ↪  new((BoundaryType)Enum.Parse(typeof(BoundaryType), data[0]),
        int.Parse(data[1]), int.Parse(data[2]), int.Parse(data[3]),
        ↪  int.Parse(data[4]));

        return boundary;
    }
}
```

ISolver.cs

```
namespace eMF_1;

public interface ISolver
{
    public int MaxIters { get; init; }
    public double Eps { get; init; }
    public double W { get; init; }

    public double[] Compute(DiagMatrix diagMatrix, double[] pr);
}
```

GaussSeidel.cs

```
namespace eMF_1;

public record GaussSeidel(int MaxIters, double Eps, double W) : ISolver
{
    public double[] Compute(DiagMatrix diagMatrix, double[] pr)
    {
        double[] qk = new double[diagMatrix.Size];
        double[] qk1 = new double[diagMatrix.Size];
        double[] residual = new double[diagMatrix.Size];
        double prNorm = pr.Norm();

        for (int i = 0; i < MaxIters; i++)
        {
            for (int k = 0; k < diagMatrix.Size; k++)
            {
                double fstSum = MultLine(diagMatrix, k, qk1, 1);
                double scdSum = MultLine(diagMatrix, k, qk, 2);

                residual[k] = pr[k] - (fstSum + scdSum);
                qk1[k] = qk[k] + W * residual[k] / diagMatrix.Diags[0][k];
            }

            qk1.Copy(qk);
            qk1.Fill(0);

            if (residual.Norm() / prNorm < Eps)
                break;
        }

        return qk;
    }

    private double MultLine(DiagMatrix diagMatrix, int i, double[] vector,
    ↪  int method)
    {
        double sum = 0;

```

```
        if (method == 0 || method == 1)
        {
            if (i > 0)
            {
                sum += diagMatrix.Diags[1][i - 1] * vector[i - 1];

                if (i > diagMatrix.ZeroDiags + 1)
                    sum += diagMatrix.Diags[2][i - diagMatrix.ZeroDiags - 2]
                    ↪  * vector[i - diagMatrix.ZeroDiags - 2];
            }
        }

        if (method == 0 || method == 2)
        {
            sum += diagMatrix.Diags[0][i] * vector[i];

            if (i < diagMatrix.Size - 1)
            {
                sum += diagMatrix.Diags[3][i] * vector[i + 1];

                if (i < diagMatrix.Size - diagMatrix.ZeroDiags - 2)
                    sum += diagMatrix.Diags[4][i] * vector[i +
                    ↪  diagMatrix.ZeroDiags + 2];
            }
        }

        return sum;
    }
}
```

ITest.cs

```
namespace eMF_1;

public interface ITest
{
    public double U(Point2D point);
    public double F(Point2D point);
}
```

FirstTest.cs

```
namespace eMF_1;

public class FirstTest : ITest
{
    public double U(Point2D point)
        ⇒ point.X;

    public double F(Point2D point)
        ⇒ 0;
}
```

SecondTest.cs

```
namespace eMF_1;

public class SecondTest : ITest
{
    public double U(Point2D point)
        ⇒ point.X * point.X - point.Y;
```

```
    public double F(Point2D point)
        => -1 + point.X * point.X - point.Y;
}
```

ThirdTest.cs

```
namespace eMF_1;

public class ThirdTest : ITest
{
    public double U(Point2D point)
        => 3 * point.X * point.X * point.X + 2 * point.Y * point.Y * point.Y;

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

FourthTest.cs

```
namespace eMF_1;

public class FourthTest : ITest
{
    public double U(Point2D point)
        => Math.Log(point.X + point.Y);

    public double F(Point2D point)
        => 2 / ((point.X + point.Y) * (point.X + point.Y));
}
```

FifthTest.cs

```
namespace eMF_1;

public class FifthTest : ITest
{
    public double U(Point2D point)
        => 4 * point.X * point.X * point.X * point.X;
    public double F(Point2D point)
        => -48 * point.X * point.X;
}
```

SixthTest.cs

```
namespace eMF_1;

public class SixthTest : ITest
{
    public double U(Point2D point)
        => 3 * point.X * point.X * point.X * point.X + point.Y;

    public double F(Point2D point)
        => -36 * point.X * point.X;
}
```

SeventhTest.cs

```cs
namespace eMF_1;

public class SeventhTest : ITest
{
    public double U(Point2D point)
        => Math.Exp(point.X) + point.Y;

    public double F(Point2D point)
        => point.Y;
}
```

EighthTest.cs

```cs
namespace eMF_1;

public class EighthTest : ITest
{
    public double U(Point2D point)
        => point.X * point.X * point.X + point.Y;

    public double F(Point2D point)
        => -6 * point.X;
}
```

Array1DExtension.cs

```cs
namespace eMF_1;

public static class Array1DExtension
{
    public static double Norm(this double[] array)
    {
        double result = 0;

        for (int i = 0; i < array.Length; i++)
            result += array[i] * array[i];

        return Math.Sqrt(result);
    }

    public static void Fill(this double[] array, double value)
    {
        for (int i = 0; i < array.Length; i++)
            array[i] = value;
    }

    public static void Copy(this double[] source, double[] destination)
    {
        for (int i = 0; i < source.Length; i++)
            destination[i] = source[i];
    }
}
```

graphics.py

```py
import matplotlib.pyplot as plt
from decimal import Decimal as dcm

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()
```