Міністерство освіти і науки України
Львівський національний університет імені Івана Франка

ОСНОВИ ОБЧИСЛЮВАЛЬНОЇ МАТЕМАТИКИ

Микола Михайлович Бокало
Андрій Романович Глова
Анатолій Омелянович Музичук
Марія Богданівна Смичок

Текст лекцій

Львів – 2022

Вступ

Обчислювальна математика — розділ математики, що включає коло питань, пов'язаних із виконанням наближених обчислень. У більш вузькому розумінні обчислювальна математика — теорія чисельних методів розв'язування типових математичних задач, яка має широке коло прикладних використань для проведення наукових та інженерних розрахунків. На її основі в останні десятиліття розвинулися нові області обчислювальних наук, як наприклад, обчислювальна хімія, обчислювальна біологія тощо. До задач обчислювальної математики належать: розв'язування систем лінійних рівнянь; пошук власних значень і векторів матриці; розв'язування нелінійних рівнянь; розв'язування систем нелінійних алгебраїчних рівнянь; розв'язування задач для диференціальних рівнянь та систем; розв'язування інтегральних рівнянь; задачі інтерполяції та інші. Основним об'єктом вивчення обчислювальної математики є чисельні методи розв'язування різноманітних математичних задач та алгоритмізація цих методів. В цьому курсі вивчаємо деякі з найбільш популярних чисельних методів.

Зміст

1 Теорія похибок наближень **7**
1.1. Поняття похибки. Абсолютна та відносна похибки. Причини виникнення похибок. . . . 7
1.2 Пряма задача теорії похибок. Похибки арифметичних операцій 9
1.3 Обернена задача теорії похибок 13

2 Чисельне розв'язування систем лінійних алгебраїчних рівнянь **15**
2.1 Прямі методи розв'язування коректних СЛАР 15
2.1.1. Метод Гаусса 16
2.1.2. Метод прогонки 22
2.2 Ітераційні методи розв'язування коректних СЛАР 25
2.2.1. Довідкова інформація 25
2.2.2. Загальна схема застосування методу простих ітерацій 27
2.2.3. Застосування методу простих ітерацій в явній формі. Метод Якобі 28
2.2.4. Застосування методу простої ітерації в неявній формі. Метод Зейделя 34
2.3 Знаходження псевдорозв'язків несумісних СЛАР 38
2.4 Знаходження власних значень і власних векторів матриць 42
2.5 Лабораторний практикум з розв'язування систем лінійних алгебраїчних рівнянь 50
2.5.1 Застосування методу Гаусса 51
2.5.2 Застосування методу прогонки для роз'язування СЛАР з тридіагональною матрицею 55
2.5.3 Застосування методу простої ітерації 58
2.5.4 Застосування методу Якобі 63
2.5.5 Застосування методу Зейделя 67
2.5.6 Знаходження псевдорозв'язків несумісних СЛАР 73
2.5.7 Знаходження власних значень і власних векторів матриць 76

3 Чисельне розв'язування нелінійних рівнянь та їх систем **81**
3.1 Розв'язування нелінійних рівнянь 81
3.1.1. Методи локалізації коренів 81
3.1.2 Метод ділення навпіл (дихотомії або бісекції) 86
3.1.3. Метод послідовних наближень (простої ітерації) з використанням стискуючих відображень 87
3.1.4. Метод Ньютона (дотичних) 88
3.1.5 Метод хорд (лінійної інтерполяції, пропорційних частин, січних) 91
3.2 Розв'язування систем нелінійних рівнянь 92
3.2.1. Локалізація розв'язків систем нелінійних рівнянь 92
3.2.2 Метод простої ітерації (послідовних наближень) 95
3.2.3 Метод Ньютона 96
3.3 Лабораторний практикум з розв'язування нелінійних рівнянь 99
3.3.1 Локалізація розв'язків рівняння графічним методом 99
3.3.2 Застосування методу бісекції 104
3.3.3 Застосування методу простої ітерації 108
3.3.4 Застосування методу Ньютона 110
3.3.5 Застосування методу хорд 114
3.4 Лабораторний практикум з розв'язування систем нелінійних рівнянь 117
3.4.1 Застосування методу простої ітерації 117

3.4.2 Застосування методу Ньютона . . . 127

4 Наближення функцій 139

4.1 Інтерполяція функцій многочленами . . . 139
4.1.1 Інтерполяційний многочлен Лагранжа . . . 140
4.1.2 Інтерполяційний многочлен Ньютона . . . 142
4.2Інтерполяція сплайнами . . . 145
4.3 Середньо квадратичне наближення . . . 148
4.4 Лабораторний практикум з наближення функцій . . . 151
4.4.1 Застосування інтерполяційного полінома Лагранжа . . . 151
4.4.2 Інтерполяційний поліном Ньютона . . . 159
4.4.3 Інтерполяція кубічними сплайнами . . . 168

5 Чисельне диференціювання та інтегрування 179

5.1 Чисельне диференціювання . . . 179
5.2 Чисельне інтегрування . . . 180
5.2.1 Загальна квадратурна формула інтерполяційного типу . . . 180
5.2.2 Квадратури Ньютона-Котеса: квадратурні формули прямокутників, трапецій і парабол (Сімпсона) . . . 181
5.3 Лабораторний практикум з чисельного інтегрування . . . 188

6 Чисельне розв'язування задач для диференціальних рівнянь та їх систем 193

6.1 Чисельне розв'язування задачі Коші для звичайних диференціальних рівнянь першого порядку. . . . 193
6.1.1. Формулювання задачі Коші для звичайного диференціального рівняння, розв'язаного стосовно похідної, та коректність цієї задачі . . . 193
6.1.2 Метод Ейлера. . . . 197
6.1.3. Методи Рунге-Кутта . . . 198
6.2 Чисельне розв'язування задачі Коші для нормальних систем звичайних диференціальних рівнянь . . . 201
6.2.1. Постановка і коректність задачі Коші для нормальних систем звичайних диференціальних рівнянь . . . 201
6.2.2. Метод Ейлера розв'язування задачі Коші для НС . . . 203
6.3 Чисельне розв'язування задачі Коші для звичайних диференціальних рівнянь вищих порядків . . . 204
6.3.1. Постановка і коректність задачі Коші для звичайних диференціальних рівнянь вищих порядків . . . 204
6.3.2. Метод Ейлера розв'язування задачі Коші для рівнянь вищих порядків . . . 204
6.4 Чисельне розв'язування крайових задач для звичайних диференціальних рівнянь . . . 206
6.4.1. Постановка і коректність крайових задач для звичайних диференціальних рівнянь 206
6.4.2. Класичний метод (метод варіації сталих) розв'язування крайових задач для ЗДР 207
6.4.3 Метод сіток (різницевий метод) . . . 210
6.5 Чисельне розв'язування крайових задач для еліптичних рівнянь. Метод сіток . . . 213
6.5.1. Постановка задачі Діріхле для рівняння Пуассона . . . 213
6.5.2 Різницева схема . . . 214
6.6 Чисельне розв'язування мішаних задач для параболічних рівнянь. Метод сіток . . . 216
6.6.1. Постановка мішаної задачі для рівняння теплопровідності . . . 216
6.6.2. Різницева схема. . . . 216
6.7Чисельне розв'язування мішаних задач для гіперболічних рівнянь. Метод сіток . . . 218
6.7.1. Постановка мішаної задачі для рівняння коливання струни . . . 218
6.7.2. Різницева схема. . . . 218
6.8 Лабораторний практикум з чисельного розв'язування задач для диференціальних рівнянь та їх систем . . . 222
6.8.1 Застосування методу Ейлера до задачі Коші для звичайних диференціальних рівнянь першого порядку . . . 222
6.8.2 Метод Рунге-Кутта . . . 230
6.8.3 Метод Ейлера і Рунге-Кутта . . . 238
6.8.4 Розв'язування крайових задач для ЗДР методом скінчених різниць (сіток) . . . 252

6.8.5 Чисельне розв'язування крайових задач для еліптичних рівнянь методом скінчених різниць . . . 262

7 Чисельне розв'язування інтегральних рівнянь 271

7.1 Основи теорії інтегральних рівнянь Фредгольма другого роду . . . 271

7.2 Методи розв'язування лінійних інтегральних рівнянь Фредгольма другого роду . . . 272

7.2.1. Метод послідовних наближень . . . 272

7.2.2. Метод механічних квадратур . . . 274

7.2.3. Метод заміни виродженим ядром . . . 276

7.3 Лабораторний практикум з чисельного розв'язування інтегральних рівнянь . . . 280

7.3.1 Метод механічних квадратур . . . 280

Розділ 1

Теорія похибок наближень

1.1 . Поняття похибки. Абсолютна та відносна похибки. Причини виникнення похибок.

Нехай a — деяке число, яке називають точним, а x — інше дійсне число, яке називають наближенням a. Тоді величину $x - a$ називають *похибкою наближення* числа a числом x. На практиці користуються поняттями *абсолютної похибки*

$$\Delta x := |x - a|, \tag{1.1.1}$$

та *відносної похибки*

$$\delta x := \frac{\Delta x}{|a|}, \tag{1.1.2}$$

якщо $a \neq 0$.

Зауваження 1.1.1. На практиці досить часто відносну похибку обчислюють за формулою

$$\delta x := \frac{\Delta x}{|x|}, \quad x \neq 0, \tag{1.1.3}$$

замість формули (1.1.2), оскільки точне число a рідко буває відомим.

Під *граничною абсолютною похибкою* та *граничною відносною похибкою* розуміють, відповідно, які-небудь числа Δ_x та δ_x такі, що

$$\Delta x \leqslant \Delta_x, \qquad \delta x \leqslant \delta_x. \tag{1.1.4}$$

Приклад 1.1.1. Знайти Δx, δx й вказати які-небудь значення Δ_x, δ_x, якщо

$$a = 2,43; \quad x = 2,4.$$

Розв'язання. Згідно з означенням маємо $\Delta x = |2,4 - 2,43| = 0,03$ — абсолютна похибка, $\delta x = \frac{0,03}{2,43} = 0,012345679...$ — відносна похибка. Граничними абсолютною та відносною похибками можуть бути, наприклад, $\Delta_x = 0,1$, $\delta_x = 0,02$. □

Похибки, які виникають при чисельному розв'язуванні різноманітних реальних задач із використанням математичного апарату, можна поділити на п'ять груп.

1. Похибки задач. Ці похибки пов'язані зі самою постановкою математичної задачі. Під час розв'язування різних реальних задач, що описують ті чи інші процеси або явища, спочатку складають математичну модель задачі, тобто описують процес чи явище за допомогою певних математичних співвідношень. Зрозуміло, що при побудові математичної моделі враховують лише найбільш важливі параметри або приймають певні припущення. Безумовно, що це спрощує задачу, але й породжує низку похибок, які називаються *похибками задачі.*

2. Похибки методу. Часто математичну задачу важко або зовсім неможливо розв'язати в точній постановці. Тоді цю задачу замінюють близькою за результатами наближеною задачею. В результаті такої заміни виникають похибки, котрі називаються *похибками методу.*

3. Залишкові похибки. Ці похибки пов'язані з наявністю нескінченних процесів у математичному аналізі (наприклад, для обчислення $\sin x$ використовують його розклад у ряд Тейлора: $\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - ...$). Оскільки нескінченний процес взагалі кажучи, не може бути завершений за скінчене число кроків, то при використанні такого процесу під час розв'язування задачі його обривають на певному кроці. Таке обривання і породжує так звану *залишкову похибку.*

4. Початкові похибки. Ці похибки виникають через наявність у математичних формулах числових параметрів, значення яких можуть бути визначені лише наближено. Такими величинами, наприклад, є всі фізичні сталі.

5. Похибки заокруглення. При позиційному зображенні чисел на кількість розрядів, що використовуються, існує обмеження. Це обмеження вимагає заокруглювати числа (початкові дані та результати виконання арифметичних операцій) при розв'язуванні задачі. Внаслідок таких заокруглень і виникають похибки, які називаються *похибками заокруглення.*

Нагадаємо *правило заокруглення* дійсних чисел, записаних у вигляді десяткового дробу. Нехай

$$a = c_m \dots c_k \dots c_0, c_{-1} \dots c_{-n} c_{-n-1} \dots$$

— позиційне зображення числа десятковим дробом. Тут $m, k, n \in \mathbb{N} \bigcup \{0\}$, $k \leq m$, $c_i \in \{0, 1, \dots, 9\}$, при $i \in \{\dots, -n-1, -n, \dots, -1, 0, 1, \dots, k, \dots, m\}$, $c_m \neq 0$, якщо $m > 0$. Припустимо, що нам потрібно заокруглити десятковий дріб a до розряду, в якому стоїть цифра c_{-n}. Тоді заокругленням a буде

$$x = c_m \dots c_k \dots c_0, c_{-1} \dots c_{-n}, \quad \text{якщо} \quad c_{-n-1} < 5,$$

і

$$x = c_m \dots c_k \dots c_0, c_{-1} \dots c_{-n} + 0, \underbrace{0 \dots 0}_{n-1} 1, \quad \text{якщо} \quad c_{-n-1} \geqslant 5,$$

тобто при заокругленні просто відкидаємо всі цифри, які стоять правіше від цифри c_{-n}, якщо $c_{-n-1} < 5$, і відкидаємо всі цифри, які стоять правіше від цифри c_{-n}, та одночасно збільшуємо цифру c_{-n} на 1, причому, якщо $c_{-n} = 9$, то замість c_{-n} пишемо 0, а цифру, яка стоїть ліворуч від c_{-n}, збільшуємо на 1.

Відмітимо, що при заокругленні числа a до числа x за граничну абсолютну похибку можна взяти число $\Delta_x = 0, \underbrace{0 \dots 0}_{n} 5$, а за граничну відносну похибку — число $\delta_x = 0, \underbrace{0 \dots 0}_{n} 5 / x$.

Приклад 1.1.2. Заокруглити десятковий дріб $a = 2,475$ до десятих і знайти абсолютну та відносну похибки та вказати граничні абсолютну та відносну похибки.

Розв'язування. За правилом заокруглення десяткових дробів маємо $x = 2,5$. Тоді

$$\Delta x = |x - a| = |2,5 - 2,475| = 0,025; \quad \delta x = \frac{0,025}{2,5} = 0,01.$$

Очевидно, що можна взяти $\Delta_x = 0,03$ і $\delta_x = 0,01$. □

Тепер припустимо, що нам потрібно заокруглити даний десятковий дріб до розряду, в якому стоїть цифра c_k. Тоді

$$x = c_m \dots c_k \underbrace{0 \dots 0}_{k}, 0, \quad \text{якщо} \quad c_{k-1} < 5,$$

і

$$x = c_m \dots c_k 0 \dots 0, 0 + 0, 10 \dots 0, 0, \quad \text{якщо} \quad c_k \geq 5,$$

тобто всі цифри, які стоять правіше від c_k, замінюємо нулями, причому цифр в розрядах, що є правіше розряду десятих, не пишемо, і одночасно, якщо $c_{k-1} \geqslant 5$, то цифру c_k збільшуємо на 1, коли $c_k \leqslant 8$, а коли $c_k = 9$, то замість c_k пишемо 0, а цифру c_{k+1} збільшуємо на 1.

Приклад 1.1.3. Заокруглити число $a = 297,21$ до десятків і знайти Δx та δx та вказати Δ_x і δ_x.

Розв'язування. Згідно з правилами заокруглення маємо $x = 300,0$. Тоді

$$\Delta x = |300,0 - 297,21| = 2,79; \quad \delta x = \frac{2,79}{297,21} = 0,0093873019... .$$

Можна взяти $\Delta_x = 3$ і $\delta_x = 0,01$.

Похибка розв'язування задачі, що виникає через неточність початкової інформації, називається **неусувною похибкою**. Вона включає похибку задачі і початкову похибку.

Основні завдання теорії похибок такі:
1. Оцінка точності результату розв'язування задачі залежно від різних видів похибок. Ця задача називається **прямою задачею** теорії похибок.
2. Визначення, з якою точністю треба взяти початкові дані, щоб неусувна похибка результату розв'язування задачі була менша за задану величину. Ця задача називається **оберненою задачею** теорії похибок.

1.2. Пряма задача теорії похибок. Похибки арифметичних операцій

Нехай D — опукла область в просторі $\mathbb{R}^n$, $u = f(z_1, ..., z_n)$, $(z_1, ..., z_n) \in D$, — задана функція.

Припустимо, що $a_1, ..., a_n$ — точні числа такі, що $(a_1, ..., a_n) \in D$, $x_1, ..., x_n$ — відповідно їх наближення такі, що $(x_1, \ldots, x_n) \in D$, а $\Delta_{x_1}, ..., \Delta_{x_n}$ — граничні абсолютні похибки наближень. Зауважимо, що

$$|x_i - a_i| \leqslant \Delta_{x_i} \quad \Leftrightarrow \quad -\Delta_{x_i} \leqslant x_i - a_i \leqslant \Delta_{x_i} \quad \Leftrightarrow \quad x_i - \Delta_{x_i} \leqslant a_i \leqslant x_i + \Delta_{x_i}, \quad i = \overline{1,n}.$$

Отже, впорядкований набір точних чисел $(a_1, ..., a_n)$ належить множині

$$\Pi = \left\{(z_1, ..., z_n) \in \mathbb{R}^n \;\middle|\; x_i - \Delta_{x_i} \leqslant z_i \leqslant x_i + \Delta_{x_i}, \quad i = \overline{1,n}\right\},$$

яку називають *областю невизначеності* точних чисел. Припустимо, що $f \in C^1(\Pi)$.

Нехай $b := f(a_1, ..., a_n)$ — точне значення функції f , а $y := f(x_1, ..., x_n)$ — його наближення. Знайдемо формули для знаходження граничних абсолютної та відносної похибок наближення числа b числом y. При цьому вважатимемо, що нам відомі тільки (!!!) наближення точних чисел і їх граничні абсолютні похибки.

На підставі теореми Лагранжа про скінчений приріст отримаємо

$$\Delta y := |y - b| = |f(x_1, \ldots, x_n) - f(a_1, \ldots, a_n)| = \left|\sum_{i=1}^{n} \frac{\partial f(z_1^*, \ldots, z_n^*)}{\partial z_i}(x_i - a_i)\right| \leqslant$$

$$\leqslant \sum_{i=1}^{n} \left|\frac{\partial f(z_1^*, \cdots, z_n^*)}{\partial z_i}\right| \Delta x_i, \tag{1.2.1}$$

де $\Delta x_i := |x_i - a_i|$, $i = \overline{1,n}$, а $(z_1^*, \cdots, z_n^*)$ — деяка точка з множини Π, що лежить на відрізку, який з'єднує точки $(x_1, \ldots, x_n)$ й $(a_1, \ldots, a_n)$.

Оскільки значення Δ_{x_i}, $i = \overline{1,n}$, є досить малими, а функції $\partial f(z_1, \ldots, z_n)/\partial z_i$, $(z_1, ..., z_n) \in \Pi$, $i = \overline{1,n}$, є неперервними, то можна вважати, що значення $\partial f(z_1^*, \ldots, z_n^*)/\partial z_i$ дуже мало відрізняється від значення $\partial f(x_1, \ldots, x_n)/\partial z_i$ для кожного $i \in \{1, .., n\}$. Тому з (1.2.1)

отримуємо ***формулу для обчислення граничної абсолютної похибки наближення значення функції***:

$$\Delta_y = \sum_{i=1}^{n} \left| \frac{\partial f\left(x_1, ..., x_n\right)}{\partial z_i} \right| \Delta_{x_i}. \tag{1.2.2}$$

Тепер знайдемо формулу для обчислення граничної відносної похибки наближеного значення функції. Для цього поділимо рівність (1.2.2) на $|y| = |f\left(x_1, \ldots, x_n\right)|$ і врахуємо, що $\Delta_{x_i} = |x_i|\delta_{x_i}$. У результаті отримаємо

$$\delta_y := \frac{\Delta_y}{|y|} = \sum_{i=1}^{n} \left| \frac{1}{f(x_1, ..., x_n)} \frac{\partial f(x_1, \ldots, x_n)}{\partial z_i} \right| \Delta_{x_i} =$$

$$= \sum_{i=}^{n} \left| \frac{\partial \ln |f(x_1, \ldots, x_n)|}{\partial z_i} \right| \Delta_{x_i} = \sum_{i=}^{n} \left| \frac{\partial \ln |f(x_1, \ldots, x_n)|}{\partial z_i} \cdot x_i \right| \delta_{x_i}.$$

Звідси маємо ***формулу для знаходження граничної відносної похибки наближення значення функції***:

$$\delta_y = \sum_{i=}^{n} \left| \frac{\partial \ln |f(x_1, \ldots, x_n)|}{\partial z_i} \right| \Delta_{x_i} \equiv \sum_{i=1}^{n} \left| \frac{\partial \ln |f(x_1, .., x_n)|}{\partial z_i} x_i \right| \delta_{x_i}. \tag{1.2.3}$$

Із формул (1.2.2) і (1.2.3) отримаємо похибки арифметичних операцій над дійсними числами.

1°. *Похибки операції додавання чисел*

Нехай $a_i > 0$, $i = \overline{1, n}$, — точні дійсні числа, а $x_i > 0$, $i = \overline{1, n}$, — їх наближення, де $n \in \mathbb{N}$ — довільне. Позначимо

$$b := a_1 + \ldots + a_n, \quad y := x_1 + \ldots + x_n,$$

тобто b — точне значення суми додатних чисел $a_1, \ldots, a_n$, а y — її наближене значення. Запишемо формули для знаходження граничних абсолютної та відносної похибок наближення числа b числом y через похибки наближень чисел $a_1, \ldots, a_n$ відповідно числами $x_1, \ldots, x_n$.

Розглянемо функцію

$$f(z_1, \ldots, z_n) = z_1 + \ldots + z_n, \quad (z_1, \ldots, z_n) \in \mathbb{R}^n.$$

Легко знайти

$$\frac{\partial f(z_1, \ldots, z_n)}{\partial z_i} = 1, \quad i = \overline{1, n},$$

$$\frac{\partial \ln |f(z_1, \ldots, z_n)|}{\partial z_i} = \frac{\partial \ln |z_1 + \ldots + z_n)|}{\partial z_i} = \frac{1}{z_1 + \ldots + z_n}, \quad i = \overline{1, n},$$

для всіх $(z_1, \ldots, z_n) \in \mathbb{R}^n$.

Звідси та формул (1.2.2) і (1.2.3) маємо ***формулу для обчислення граничної абсолютної похибки операції додавання чисел***:

$$\Delta_y = \sum_{i=1}^{n} \Delta_{x_i}, \tag{1.2.4}$$

$$\delta_y = \sum_{i=1}^{n} \frac{x_i}{x_1 + \ldots + x_n} \delta_{x_i} \leqslant \max_{1 \leqslant i \leqslant n} \delta_{x_i}.$$

Тому можна використовувати таку ***формулу для обчислення граничної відносної похибки операції додавання чисел***:

$$\delta_y = \max_{1 \leqslant i \leqslant n} \delta_{x_i}. \tag{1.2.5}$$

Зауваження 1.2.1. Легко переконатися, що формули (1.2.4) і (1.2.5) для знаходження похибок наближення суми чисел можна знайти безпосередньо, не використовуючи формул (1.2.2) і (1.2.3). Справді, маємо

$$\Delta y = |y - b| = |(x_1 + \ldots + x_n) - (a_1 + \ldots + a_n)| =$$
$$= |(x_1 - a_1) + \ldots + (x_n - a_n)| \leqslant |x_1 - a_1| + \ldots + |x_n - a_n| = \Delta x_1 + \ldots + \Delta x_n.$$

Звідси отримуємо формулу (1.2.4). З цієї формули врахувавши, що $\Delta x_i = x_i \delta_i$, $i = \overline{1, n}$, здобуваємо

$$\delta y = \frac{\Delta y}{|y|} = \frac{1}{x_1 + \ldots + x_n} \sum_{i=1}^{n} \Delta_{x_i} = \sum_{i=1}^{n} \frac{x_i}{x_1 + \ldots + x_n} \delta_{x_i} \leq \max_{i \leqslant 1 \leqslant n} \delta_{x_i}.$$

Звідси отримуємо формулу (1.2.5).

2°. *Похибки операції віднімання чисел*

Нехай $a_1 > 0$, $a_2 > 0$ — точні числа, а $x_1 > 0$, $x_2 > 0$ — їх наближення. Позначимо

$$b := a_1 - a_2, \quad y := x_1 - x_2,$$

тобто b — точне значення різниці чисел a_1 і a_2, а y — її наближення.

Знайдемо формули для знаходження Δ_y і δ_y. Для цього розглянемо функцію

$$f(z_1, z_2) := z_1 - z_2, \quad (z_1, z_2) \in \mathbb{R}^2.$$

Легко знайти

$$\frac{\partial f(z_1, z_2)}{\partial z_1} = \frac{\partial (z_1 - z_2)}{\partial z_1} = 1, \quad \frac{\partial f(z_1, z_2)}{\partial z_2} = \frac{\partial (z_1 - z_2)}{\partial z_2} = -1, \quad (z_1, z_2) \in \mathbb{R}^2,$$
$$\frac{\partial \ln |f(z_1, z_2)|}{\partial z_1} = \frac{\partial \ln |z_1 - z_2|}{\partial z_1} = \frac{1}{z_1 - z_2},$$
$$\frac{\partial \ln |f(z_1, z_2)|}{\partial z_2} = \frac{\partial \ln |z_1 - z_2|}{\partial z_2} = -\frac{1}{z_1 - z_2},$$
$$(z_1, z_2) \in \mathbb{R}^2, \ z_1 \neq z_2.$$

Отже, на підставі формул (1.2.2) і (1.2.3) маємо ***формули для обчислення граничних абсолютної та відносної похибок операції віднімання чисел***:

$$\Delta_y = \Delta_{x_1} + \Delta_{x_2}, \tag{1.2.6}$$

$$\delta_y = \frac{x_1 \delta_{x_1} + x_2 \delta_{x_2}}{|x_1 - x_2|}. \tag{1.2.7}$$

Звідси можна зробити висновок, що при відніманні близьких за значенням чисел відносна похибка результату може бути дуже великою. Тому при застосуванні чисельних методів бажано уникати віднімання близьких за значенням чисел. Наприклад, при знаходженні похибки віднімання чисел $\sqrt{1001} - \sqrt{1000}$ можна попередньо помножити й поділити даний вираз на $\sqrt{1001} + \sqrt{1000}$:

$$\sqrt{1001} - \sqrt{1000} = \frac{1001 - 1000}{\sqrt{1001} + \sqrt{1000}} = \frac{1}{\sqrt{1001} + \sqrt{1000}}.$$

Як далі покажемо, відносна похибка числа $\frac{1}{\sqrt{1001}+\sqrt{1000}}$ суттєво менша ніж числа $\sqrt{1001} - \sqrt{1000}$.

Зауваження 1.2.2. Відмітимо, що (1.2.6) і (1.2.7) можна отримати безпосередньо не використовуючи формул (1.2.2) і (1.2.3).

3°. *Похибки операції множення чисел*

Спочатку зауважимо, що коли a — точне число, то x — його наближення, то $-x$ є наближенням $-a$ і

$$\Delta(-x) = \Delta x, \qquad \delta(-x) = \delta x.$$

Справді, маємо

$$\Delta(-x) = |(-x) - (-a)| = |-x + a| = |x - a| = \Delta x,$$
$$\delta(-x) = \frac{\Delta(-x)}{|-x|} = \frac{\Delta x}{|x|} = \delta x.$$

Звідси випливає, що при розгляді похибок результатів множення чи ділення дійсних чисел можна вважати, що ці числа є додатними. Отже, нехай $a_i > 0$, $i = \overline{1,n}$, — точні числа, а $x_i > 0$, $i = \overline{1,n}$, — відповідні їх наближення. Позначимо

$$b = a_1 \cdot \ldots \cdot a_n, \quad y = x_1 \cdot \ldots \cdot x_n,$$

тобто b — точне значення добутку чисел $a_1, \ldots, a_n$, а y — його наближення.

Розглянемо функцію

$$f(z_1, \ldots, z_n) = z_1 \cdot \ldots \cdot z_n, \quad (z_1, \ldots, z_n) \in \mathbb{R}^n.$$

Легко знайти

$$\frac{\partial f(z_1, \ldots, z_n)}{\partial z_i} = z_1 \cdot \ldots \cdot z_{i-1} \cdot z_{i+1} \cdot \ldots \cdot z_n \equiv \prod_{j=1, j\neq i}^{n} z_j,$$
$$\frac{\partial \ln|f(z_1, \ldots, z_n)|}{\partial z_i} = \frac{\partial \ln|z_1 \cdot \ldots \cdot z_n|}{\partial z_i} = \frac{\partial}{\partial z_i}\left(\ln|z_1| + \ldots + \ln|z_i| + \ldots + \ln|z_n|\right) = \frac{1}{z_i},$$
$$i = \overline{1,n}, \quad (z_1, \ldots, z_n) \in \mathbb{R}^n.$$

За формулою (1.2.2) маємо ***формули для обчислення граничних абсолютної та відносної похибок операції множення чисел***:

$$\Delta_y = \sum_{i=1}^{n} \left(\prod_{j=1, j\neq i}^{n} x_j \right) \Delta_{x_i} \equiv \sum_{i=1}^{n} x_1 \cdot \ldots \cdot x_{i-1} \cdot \Delta_{x_i} \cdot x_{i+1} \cdot \ldots \cdot x_n, \tag{1.2.8}$$

$$\delta_y = \sum_{i=1}^{n} \delta_{x_i}. \tag{1.2.9}$$

4°. *Похибки операції ділення чисел*

Нехай $a_1 > 0$, $a_2 > 0$ — точні числа, а $x_1 > 0$, $x_2 > 0$ — їх наближення. Позначимо

$$b := \frac{a_1}{a_2}, \quad y := \frac{x_1}{x_2} \quad —$$

відповідно, точне значення частки чисел a_1, a_2 і його наближення. Знайдемо Δ_y і δ_y. Для цього введемо в розгляд функцію

$$f(z_1, z_2) = \frac{z_1}{z_2}, \quad (z_1, z_2) \in \mathbb{R}^2, \quad z_2 \neq 0.$$

Легко знайти

$$\frac{\partial f(z_1, z_2)}{\partial z_1} = \frac{\partial}{\partial z_1}\left(\frac{z_1}{z_2}\right) = \frac{1}{z_2},$$
$$\frac{\partial f(z_1, z_2)}{\partial z_2} = \frac{\partial}{\partial z_2}\left(\frac{z_1}{z_2}\right) = -\frac{z_1}{z_2^2},$$
$$\frac{\partial \ln|f(z_1, z_2)|}{\partial z_1} = \frac{\partial}{\partial z_1}\left(\ln|z_1| - \ln|z_2|\right) = \frac{1}{z_1},$$
$$\frac{\partial \ln|f(z_1, z_2)|}{\partial z_2} = \frac{\partial}{\partial z_2}\left(\ln|z_1| - \ln|z_2|\right) = -\frac{1}{z_2}.$$

За формулами (1.2.2) і (1.2.3) знаходимо ***формули для обчислення граничних абсолютної та відносної похибок операції ділення чисел***:

$$\Delta_y = \frac{x_2\Delta_{x_1} + x_1\Delta_{x_2}}{x_2^2}, \tag{1.2.10}$$

$$\delta_y = \delta_{x_1} + \delta_{x_2}. \tag{1.2.11}$$

Зауваження 1.2.3. Формули (1.2.8) – (1.2.11) можна отримати безпосередньо, але легше це робити, використовуючи формули (1.2.2) і (1.2.3).

1.3. Обернена задача теорії похибок

Нехай $n \in \mathbb{N}$, D — область в $\mathbb{R}^n$, $f(z_1, \ldots, z_n)$, $(z_1, \ldots, z_n) \in D$, — деяка неперервна функція. Припустимо, що $a_1, \ldots, a_n$ — точні числа такі, що $(a_1, \ldots, a_n) \in D$. Нехай $b := f(a_1, \ldots, a_n)$ — точне значення функції f в точці $(a_1, \ldots, a_n)$. *Обернена задача теорії похибок* полягає у знаходженні граничних абсолютних та відносних похибок наближень $(x_1, \ldots, x_n) \in D$ таких, що гранична похибка наближення значення функції $y = f(x_1, \ldots, x_n)$ є Δ_y.

Нехай $\Pi := \{(z_1, \ldots, z_n) \,|\, c_i \le z_i \le d_i\}$ — область невизначеності точного значення $(a_1, \ldots, a_n)$, тобто множина, якій належить a і звідки беремо наближення $(x_1, \ldots, x_n)$. Тут $-\infty < c_i < d_i < +\infty$, $i = \overline{1, n}$. Припустимо, що $\Delta_{x_1} = \ldots = \Delta_{x_n} =: \Delta_x$ і покладемо

$$M := \max_{(z_1,\ldots,z_n)\in\Pi} \sum_{i=1}^{n} \left| \frac{\partial f(z_1, \ldots, z_n)}{\partial z_i} \right|.$$

Тоді згідно з формулою (1.2.2) можна вважати, що

$$\Delta_y = M\Delta_x. \tag{1.3.1}$$

Припускаючи, що нам задана гранична абсолютна похибка Δ_y наближення значення y заданої функції, з формули (1.3.1) отримаємо ***формулу знаходження граничних абсолютних похибок наближень аргументів функції, коли задана гранична абсолютна похибка значення цієї функції***:

$$\Delta_x = M^{-1}\Delta_y. \tag{1.3.2}$$

Аналогічно шукаємо граничні відносні похибки аргументів функції, коли задана гранична відносна похибка значення цієї функції. Нехай

$$\delta_{x_1} = \ldots = \delta_{x_n} =: \delta_x, \quad N := \max_{(z_1,\ldots,z_n)\in\Pi} \sum_{i=1}^{n} \left| \frac{\partial \ln |f(z_1, \ldots, z_n)|}{\partial z_i} z_i \right|.$$

Тоді на підставі формули (1.2.3) можна вважати, що

$$\delta_y = N\delta_x$$

звідси отримаємо ***формулу для знаходження граничних відносних похибок аргументів функції, коли задана гранична відносна похибка наближення значення цієї функції***:

$$\delta_x = N^{-1}\delta_y. \tag{1.3.3}$$

Приклад 1.3.1. Написати формули знаходження граничних похибок наближень чисел, коли задані похибки їх суми.

Розв'язування. Нехай $f(z_1, \ldots, z_n) = z_1 + \ldots + z_n$.

Маємо

$$\frac{\partial f(z_1, \ldots, z_n)}{\partial z_i} = 1, \qquad i = \overline{1, n},$$

$$\frac{\partial \ln |f(z_1, \ldots, z_n)|}{\partial z_i} = \frac{\partial \ln |z_1 + \ldots + z_n|}{\partial z_i} = \frac{1}{z_1 + \ldots + z_n}, \qquad i = \overline{1, n}.$$

Далі припускаємо, що $z_i > 0, \quad i = \overline{1, n}$. Отож, маємо $M = n, \quad N = 1$. Звідси отримаємо

$$\Delta_x = n^{-1} \Delta_y, \qquad \delta_x = \delta_y.$$

□

Вправи для самостійної роботи

1. Заокруглити числа і знайти абсолютну та відносну похибки заокруглення та вказати граничні абсолютну та відносні похибки:

а) 2,71 (до десятих); б) 3758 (до тисяч); в) 5,756 (до сотих).

2. Розв'язати пряму задачу теорії похибок стосовно заданих функцій, наближень значень аргументів та їх граничних абсолютних похибок:

а) $f(z) := z^3, \quad x = 3; \quad \Delta_x = 0,3;$

б) $f(z_1, z_2) := 2z_1^3 + z_2^2; \quad x_1 = 3; \quad x_2 = 1; \quad \Delta x_1 = 0,3; \quad \Delta x_2 = 0,4;$

в) $f(z_1, z_2, z_3) := 3z_1 + \ln z_2 + 1/z_3; \quad x_1 = 3; \quad x_2 = 1; \quad x_3 = 2; \quad \Delta x_1 = 0,3; \quad \Delta x_2 = 0,1; \quad \Delta x_3 = 0,2.$

3. Розв'язати обернену задачу теорії похибок стосовно заданих функцій і областей невизначеностей точних чисел:

а) $f(z) := 2 \ln z, \quad \Pi := [1, 4];$

б) $f(z_1, z_2) := \sin z_1 + 3z^2, \quad \Pi := [0; \pi] \times [0; 3];$

в) $f(z_1, z_2, z_3) := 2z_1^2 + 3z_2 z_3, \quad \Pi := \{(z_1, z_2, z_3) \,\big|\, 1 \leqslant z_1 \leqslant 2, \quad 2 \leqslant z_2 \leqslant 4, \quad 1 \leqslant z_3 \leqslant 5\}.$

Розділ 2

Чисельне розв'язування систем лінійних алгебраїчних рівнянь

Системою лінійних алгебраїчних рівнянь (коротко, ***СЛАР***) називають систему рівнянь

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n, \end{cases} \tag{2.0.1}$$

або, коротше,

$$\sum_{j=1}^{n} a_{ij}x_j = b_j, \quad i = \overline{1,n},$$

де $n \geqslant 2$ – задане натуральне число, a_{ij}, b_i, $i, j = \overline{1,n}$, – задані числа, які називають, відповідно, *коефіцієнтами* та *вільними членами* СЛАР, x_i, $i = \overline{1,n}$, – шукані числові величини.

Якщо ввести позначення

$$A := \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{pmatrix}, \quad b := \begin{pmatrix} b_1 \\ b_2 \\ \ldots \\ b_n \end{pmatrix}, \quad x := \begin{pmatrix} x_1 \\ x_2 \\ \ldots \\ x_n \end{pmatrix} \equiv (x_1, x_2 \ldots, x_n)^\top,$$

то систему (2.0.1) можна коротко записати у вигляді

$$Ax = b. \tag{2.0.2}$$

Далі в цьому розділі розглядаємо випадок, коли матриця A є невиродженою, тобто $\det A \neq 0$ (визначник матриці A відмінний від нуля). Тоді, як відомо, система (2.0.2) має і тільки один розв'язок і її називають коректною. Наша мета в цьому розділі — описати методи знаходження розв'язків коректних СЛАР.

2.1. Прямі методи розв'язування коректних СЛАР

З лінійної алгебри відомо, що для розв'язування коректної системи (2.0.1) використовують методи Крамера, оберненої матриці та Гаусса. Методи Крамера та оберненої матриці для досить великих значень n використовувати нераціонально, бо це приводить до дуже громіздких обчислень та нагромадження похибок, а тому, як правило, використовують метод Гаусса та різні його модифікації.

2.1.1 . Метод Гаусса

Ідея методу Гаусса полягає в тому, щоб систему лінійних алгебраїчних рівнянь за допомогою еквівалентних перетворень звести до системи з трикутною матрицею, оскільки розв'язання такої системи не вимагає жодних зусиль. Процес зведення системи до трикутного вигляду називається *прямим ходом*. А відшукання розв'язку зведеної системи — *зворотним ходом*. Прямий хід складається із $(n-1)$-го кроку, на кожному з яких із частини рівнянь системи вилучається одне з невідомих.

Припустимо, що в системі (2.0.1) коефіцієнт $a_{11} \neq 0$ (якщо $a_{11} = 0$, то перше рівняння і рівняння, в якому коефіцієнт при x_1 не дорівнює нулеві, переставляємо місцями). Тоді на першому кроці вилучаємо x_1 із усіх рівнянь, крім першого. Щоб вилучити x_1 з i-го, $i \in \{2, 3, \ldots, n\}$, рівняння системи (2.0.1), треба перше рівняння помножити на число

$$m_i^{(1)} = \frac{a_{i1}}{a_{11}}$$

і відняти від i-го рівняння. Отримуємо систему

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n = b_1, \\ \qquad a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + \ldots + a_{2n}^{(1)}x_n = b_2^{(1)}, \\ \qquad a_{32}^{(1)}x_2 + a_{33}^{(1)}x_3 + \ldots + a_{3n}^{(1)}x_n = b_3^{(1)}, \\ \qquad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\ \qquad a_{n2}^{(1)}x_2 + a_{n3}^{(1)}x_3 + \ldots + a_{nn}^{(1)}x_n = b_n^{(1)}, \end{cases} \tag{2.1.1}$$

де

$$a_{ij}^{(1)} := a_{ij} - m_i^{(1)}a_{1j}, \quad b_i^{(1)} := b_i - m_i^{(1)}b_1, \quad i, j \in \{2, \ldots, n\}.$$

На другому кроці, якщо $a_{22}^{(1)} \neq 0$ (у випадку $a_{22}^{(1)} = 0$ переставляємо рівняння), вилучаємо x_2 з усіх рівнянь, крім першого і другого. Щоб вилучити x_2 з i-го, $i \in \{3, 4, \ldots, n\}$, рівняння системи (2.1.1), треба друге рівняння помножити на число

$$m_i^{(2)} = \frac{a_{i2}^{(1)}}{a_{22}^{(1)}}$$

і відняти від i-го рівняння. Одержимо систему

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n = b_1, \\ \qquad a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + \ldots + a_{2n}^{(1)}x_n = b_2^{(1)}, \\ \qquad\qquad a_{33}^{(2)}x_3 + \ldots + a_{3n}^{(2)}x_n = b_3^{(2)}, \\ \qquad\qquad \ldots \quad \ldots \quad \ldots \quad \ldots \\ \qquad\qquad a_{n3}^{(2)}x_3 + \ldots + a_{nn}^{(2)}x_n = b_n^{(2)}, \end{cases} \tag{2.1.2}$$

де

$$a_{ij}^{(2)} := a_{ij}^{(1)} - m_i^{(2)}a_{2j}^{(1)}, \quad b_i^{(2)} := b_i^{(1)} - m_i^{(2)}b_2^{(1)}, \quad i, j = 3, \ldots, n.$$

І т.д. Після виконання $k-$го кроку маємо $a_{i,j}^{(k)} = 0$ при $i \geqslant k,\ j < k$.

Якщо $k < n-1$, то на наступному $(k+1)-$му кроці елементи матриці та вектора правої частини перераховують за формулами

$$a_{i,j}^{(k+1)} = a_{i,j}^{(k)} - m_i^{(k+1)}\, a_{k,j}^{(k)}, \quad i, j = k+1, \ldots, n, \tag{2.1.3}$$

$$b_i^{(k+1)} = b_i^{(k)} - m_i^{(k+1)}\, b_k^{(k)}, \quad i = k+1, \ldots, n, \tag{2.1.4}$$

де

$$m_i^{(k+1)} = \frac{a_{i,k}^{(k)}}{a_{k,k}^{(k)}}, \quad i = k+1, \ldots, n.$$

Через $n-1$ кроків отримаємо систему

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n = b_1, \\ \qquad a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + \ldots + a_{2n}^{(1)}x_n = b_2^{(1)}, \\ \qquad\qquad a_{33}^{(2)}x_3 + \ldots + a_{3n}^{(2)}x_n = b_3^{(2)}, \\ \qquad\qquad \ldots \quad \ldots \quad \ldots \quad \ldots \\ \qquad\qquad\qquad a_{nn}^{(n-1)}x_n = b_n^{(n-1)}, \end{cases} \tag{2.1.5}$$

матриця якої є трикутною.

Звівши систему (2.0.1) до трикутного вигляду, виконаємо зворотний хід. З останнього рівняння системи (2.1.5) визначимо x_n; підставивши x_n у передостаннє рівняння, визначимо x_{n-1} і т.д.; підставивши знайдені значення $x_n, x_{n-1}, \ldots, x_2$ у перше рівняння, визначимо x_1.

На практиці зручно записувати реалізацію прямого ходу як відповідні елементарні перетворення розширеної матриці заданої системи, тобто матриці

$$A|b = \left(\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \ldots & a_{1n} & b_1 \\ a_{21} & a_{22} & a_{23} & \ldots & a_{2n} & b_2 \\ \ldots & \ldots & \ldots & \ldots & & \ldots \\ a_{n1} & a_{n2} & a_{n3} & \ldots & a_{nn} & b_n \end{array}\right). \tag{2.1.6}$$

Зазначимо, що якщо у процесі виконання прямого ходу на якомусь кроці в новій системі з'явилось рівняння, в якому всі коефіцієнти лівої частини дорівнюють нулеві, а права частина не дорівнює нулеві, то, очевидно, система не має розв'язків, тобто є несумісною. Якщо ж і права частина дорівнює нулеві, то таке рівняння можна вилучити із системи, не порушуючи еквівалентності систем. Якщо на k-му кроці ($k \in \{1, \ldots, n-1\}$) при вилученні x_k вилучається хоча б одна з невідомих x_{k+s} ($s \in \{1, \ldots, n-k\}$) з останніх $n-k$ рівнянь, то це свідчить про те, що система рівнянь має безліч розв'язків. Однак у випадку коректної системи рівнянь прямий хід завжди приводить до системи, матриця котрої має трикутний вигляд.

Зауваження 2.1.1. Зворотний хід в системі Гаусса можна робити аналогічно як прямий хід. Пояснимо це на розширеній матриці $A|b$ даної СЛАР.

Отож, припустимо, що ми зробили прямий хід методу Гаусса:

$$A|b = \left(\begin{array}{cccccc|c} a_{11} & a_{12} & \ldots & a_{1n-2} & a_{1n-2} & a_{1n} & b_1 \\ a_{21} & a_{22} & \ldots & a_{2n-2} & a_{2n-2} & a_{2n} & b_2 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ a_{n-21} & a_{n-22} & \ldots & a_{n-2n-2} & a_{n-2n-1} & a_{n-2n} & b_{n-2} \\ a_{n-11} & a_{n-12} & \ldots & a_{n-1n-2} & a_{n-1n-1} & a_{n-1n} & b_{n-1} \\ a_{n1} & a_{n2} & \ldots & a_{nn-2} & a_{nn-1} & a_{nn} & b_n \end{array}\right) \sim$$

$$\sim \left(\begin{array}{cccccc|c} a_{11}^{(0)} & a_{12}^{(0)} & \ldots & a_{1n-2}^{(0)} & a_{1n-2}^{(0)} & a_{1n}^{(0)} & b_1^{(0)} \\ 0 & a_{22}^{(1)} & \ldots & a_{2n-2}^{(1)} & a_{2n-1}^{(1)} & a_{2n}^{(1)} & b_2^{(1)} \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & a_{n-2n-2}^{(n-3)} & a_{n-2n-1}^{(n-3)} & a_{n-2n}^{(n-3)} & b_{n-2}^{(n-3)} \\ 0 & 0 & \ldots & 0 & a_{n-1n-1}^{(n-2)} & a_{n-1n}^{(n-2)} & b_{n-1}^{(n-2)} \\ 0 & 0 & \ldots & 0 & 0 & a_{nn}^{(n-1)} & b_n^{(n-1)} \end{array}\right),$$

де $a_{1j}^{(0)} := a_{1j},\ j = \overline{1,n},\quad b_1^{(0)} := b_1$.

Тепер n-ий рядок отриманої матриці ділимо на $a_{nn}^{(n-1)}$ і фіксуємо. У результаті n-ий рядок матиме вигляд $(0 \quad 0 \quad 0 \quad \ldots \quad 0 \quad 0 \quad 1 \quad | \quad b_n^{(n)})$. Далі робимо такі елементарні перетворення отриманої матриці. Для кожного $i \in \{1, \ldots, n-1\}$ множимо n-ий рядок, тобто

$$(0 \quad 0 \quad 0 \quad \ldots \quad 0 \quad 0 \quad 1 \quad | \quad b_n^{(n)}),$$

на число $a_{kn}^{(k-1)}$ (це елемент, що стоїть на перетині k-го рядка і n-го стопчика) і віднімаємо від k-го рядка. У результаті отримаємо матрицю

$$\left(\begin{array}{cccccc|c} a_{11}^{(0)} & a_{12}^{(0)} & \dots & a_{1n-2}^{(0)} & a_{1n-1}^{(0)} & 0 & b_1^{(1)} \\ 0 & a_{22}^{1} & \dots & a_{2n-2}^{1} & a_{2n-1}^{1} & 0 & b_2^{(2)} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & a_{n-2n-2}^{(n-3)} & a_{n-2n-1}^{(n-3)} & 0 & b_{n-2}^{(n-2)} \\ 0 & 0 & \dots & 0 & a_{n-1n-1}^{(n-2)} & 0 & b_{n-1}^{(n-1)} \\ 0 & 0 & \dots & 0 & 0 & 1 & b_n^{(n)} \end{array}\right).$$

Далі проводимо елементарні перетворення отриманої матриці цілком аналогічні до проведених вище, але працюємо тільки з елементами, які стоять на перетині перших $n-1$ рядків і перших $n-1$ стовпчиків основної матриці, а також з $n-1$ елементами стовпчика вільних членів. У результаті отримаємо матрицю

$$\left(\begin{array}{cccccc|c} a_{11}^{(0)} & a_{12}^{(0)} & \dots & a_{n-2}^{(0)} & 0 & 0 & b_1^{(2)} \\ 0 & a_{22}^{(1)} & \dots & a_{2\,n-2}^{(1)} & 0 & 0 & b_2^{(3)} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & a_{n-2\,n-2}^{(n-3)} & 0 & 0 & b_{n-2}^{(n-1)} \\ 0 & 0 & \dots & 0 & 1 & 0 & b_{n-1}^{(n)} \\ 0 & 0 & \dots & 0 & 0 & 1 & b_n^{(n)} \end{array}\right).$$

Продовжуючи діяти аналогічно до того, як ми робили, приходимо до матриці

$$\left(\begin{array}{cccccc|c} 1 & 0 & \dots & 0 & 0 & 0 & b_1^{(n)} \\ 0 & 1 & \dots & 0 & 0 & 0 & b_2^{(n)} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & 0 & 0 & b_{n-2}^{(n)} \\ 0 & 0 & \dots & 0 & 1 & 0 & b_{n-1}^{(n)} \\ 0 & 0 & \dots & 0 & 0 & 1 & b_n^{(n)} \end{array}\right),$$

тобто отримали СЛАР

$$\left\{\begin{array}{rcl} x_1 & = & b_1^{(n)} \\ x_2 & = & b_2^{(n)} \\ \dots & & \\ x_{n-2} & = & b_{n-2}^{(n)} \\ x_{n-1} & = & b_{n-1}^{(n)} \\ x_n & = & b_n^{(n)} \end{array}\right.,$$

розв'язком якої, очевидно, є вектор $(b_1^{(n)}, \dots, b_n^{(n)})^\top$.

Зауваження 2.1.2. Відмітимо, що звівши нашу систему до системи (2.1.5), визначник матриці A обчислюємо за формулою

$$\det := a_{11} \cdot a_{22}^{(1)} \cdot \ldots \cdot a_{nn}^{(n-1)}. \tag{2.1.7}$$

Зауваження 2.1.3. За допомогою методу Гаусса можна знаходити **обернену матрицю**. Нагадаємо її означення. Нехай $A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \in \mathbb{R}^{n\times n}$ — деяка матриця. Матрицю $X = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \dots & \dots & \dots & \\ x_{n1} & x_{n2} & \dots & x_{nn} \end{pmatrix} \in \mathbb{R}^{n\times n}$ таку, що

$$XA = AX = I,$$

де $I = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \\ 0 & 0 & \dots & 1 \end{pmatrix}$ — одинична матриця, тобто матриця, елементами якої є

$$\delta_{ij} = \begin{cases} 1, & i = j, \\ 0, & i \neq j, \end{cases}$$

— символ Кронекерра, називають оберненою матрицею до матриці A і позначають через A^{-1}. Обернена до A матриця A^{-1} існує, якщо $\det A \neq 0$, тобто A є невиродженою матрицею. Обернену матрицю A^{-1} можна обчислити за відомою формулою

$$A^{-1} = \frac{1}{\det A} \begin{pmatrix} A_{11} & A_{21} & \dots & A_{n1} \\ A_{12} & A_{22} & \dots & A_{n2} \\ \dots & \dots & \dots & \\ A_{1n} & A_{2n} & \dots & A_{nn} \end{pmatrix}, \tag{2.1.8}$$

де

$$A_{ij} = (-1)^{i+j} M_{ij}, \quad i, j = \overline{1, n},$$

а для кожних $i, j \in \{1, \dots, n\}$ під M_{ij} розуміють визначник матриці, отриманої з матриці A викреслюванням i-го рядка та j-го стовпчика. Але з обчислювальної точки зору при великих значеннях n формула (2.1.8) неефективна за рахунок великого накопичення похибок обчислень. Тому, як правило, використовують інші способи знаходження A^{-1}. Один з них полягає у використанні для цих цілей методу Гаусса. Суть цього підходу така.

Позначимо

$$x^j := \begin{pmatrix} x_{1j} \\ x_{2j} \\ \dots \\ x_{nj} \end{pmatrix}, \quad j = \overline{1, n}, \qquad X = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \dots & \dots & \dots & \\ x_{n1} & x_{n2} & \dots & x_{nn} \end{pmatrix} =: (x^1, x^2, \dots, x^n),$$

$$e^j := \begin{pmatrix} 0 \\ 0 \\ \dots \\ 1 \\ \dots \\ 0 \end{pmatrix}, \quad j = \overline{1, n}, \qquad I = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \\ 0 & 0 & \dots & 1 \end{pmatrix} =: (e^1, e^2, \dots, e^n),$$

тобто e^j — n- вимірний вектор, j-ий елемент якого дорівнює 1, а решта - 0, $j = \overline{1, n}$, а отже I — одинична матриця. Тоді згідно з означенням оберненої матриці маємо

$$AX = I \quad \Longleftrightarrow \quad A(x^1, x^2, \dots, x^n) = (e^1, e^2, \dots, e^n),$$

тобто

$$Ax^j = e^j, \quad j = \overline{1, n}. \tag{2.1.9}$$

Отже, для знаходження оберненої до A матриці X треба розв'язати сукупність n СЛАР вигляду (2.1.9). Цю сукупність можна розв'язувати одночасно, використовуючи метод Гаусса, якщо записати "розширену"матрицю:

$$\left(\begin{array}{cccc|cccc} a_{11} & a_{12} & \dots & a_{1n} & 1 & 0 & \dots & 0 \\ a_{21} & a_{22} & \dots & a_{2n} & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} & 0 & 0 & \dots & 1 \end{array}\right).$$

Далі робимо елементарні перетворення цієї матриці згідно з правилами прямого ходу методу Гаусса. Наприклад, якщо $a_{11} \neq 0$, то множимо 1-ий рядок "розширеної"матриці на $\frac{a_{i1}}{a_{11}}$, де i набуває значень з множини $\{2, \dots, n\}$, і віднімаємо від i-го рядка. Зробивши прямий хід методу Гаусса, отримаємо

$$\left(\begin{array}{cccc|cccc} a_{11}^0 & a_{12}^0 & \dots & a_{1n}^0 & 1 & 0 & \dots & 0 \\ 0 & a_{22}^1 & \dots & a_{2n}^1 & b_{21} & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & a_{nn}^{(n-1)} & b_{n1} & b_{n2} & \dots & 1 \end{array}\right).$$

Тоді робимо зворотній хід методу Гаусса. У результаті отримаємо:

$$\left(\begin{array}{cccc|cccc} 1 & 0 & \dots & 0 & x_{11} & x_{12} & \dots & x_{1n} \\ 0 & 1 & \dots & 0 & x_{21} & x_{22} & \dots & x_{2n} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & x_{n1} & x_{n2} & \dots & x_{nn} \end{array}\right),$$

звідси маємо обернену матрицю X.

Приклад 2.1.1. Методом Гаусса розв'язати СЛАР

$$\begin{cases} 2x_1 + 2x_2 + 3x_3 &= 1, \\ x_1 + 3x_2 + 2x_3 &= -8, \\ 2x_1 + x_2 + 2x_3 &= 3. \end{cases}$$

► Запишемо розширену матрицю даної системи:

$$A|b = \left(\begin{array}{ccc|c} 2 & 2 & 3 & 1 \\ 1 & 3 & 2 & -8 \\ 2 & 1 & 2 & 3 \end{array}\right). \tag{2.1.10}$$

На першому кроці знайдемо

$$\begin{aligned} & m_2^{(1)} = \frac{a_{21}}{a_{11}} = \frac{1}{2}, \quad m_3^{(1)} = \frac{a_{31}}{a_{11}} = 1, \\ & a_{22}^{(1)} = a_{22} - m_2^{(1)} a_{12} = 2, \quad a_{23}^{(1)} = a_{23} - m_2^{(1)} a_{13} = \frac{1}{2}, \\ & a_{32}^{(1)} = a_{32} - m_3^{(1)} a_{12} = -1, \quad a_{33}^{(1)} = a_{33} - m_3^{(1)} a_{13} = -1, \\ & b_2^{(1)} = b_2 - m_2^{(1)} b_1 = -\frac{17}{2}, \quad b_3^{(1)} = b_3 - m_3^{(1)} b_1 = 2. \end{aligned}$$

У результаті одержимо матрицю

$$\left(\begin{array}{ccc|c} 2 & 2 & 3 & 1 \\ 0 & 2 & \frac{1}{2} & -\frac{17}{2} \\ 0 & -1 & -1 & 3 \end{array}\right).$$

На другому кроці знайдемо

$$m_3^{(2)} = \frac{a_{32}^{(1)}}{a_{22}^{(1)}} = -\frac{1}{2}, \quad a_{33}^{(2)} = a_{33}^{(1)} - m_3^{(2)} a_{23}^{(1)} = -\frac{3}{4}, \quad b_3^{(2)} = b_3^{(1)} - m_3^{(2)} b_2^{(1)} = -\frac{9}{4}.$$

У результаті одержимо матрицю

$$\left(\begin{array}{ccc|c} 2 & 2 & 3 & 1 \\ 0 & 2 & \frac{1}{2} & -\frac{17}{2} \\ 0 & 0 & \frac{3}{4} & -\frac{9}{4} \end{array}\right),$$

а отже, еквівалентна даній з трикутною основною матрицею система має вигляд

$$\begin{cases} 2x_1 + 2x_2 + 3x_3 &= 1, \\ 2x_2 + \frac{1}{2}x_3 &= -\frac{17}{2}, \\ -\frac{3}{4}x_3 &= -\frac{9}{4}. \end{cases}$$

Використовуючи обернений хід методу Гаусса, знайдемо розв'язок даної системи:

$$x_3 = 3; \quad x_2 = -5; \quad x_1 = 1.$$

Обернений хід методу Гаусса можна зробити і так:

$$\left(\begin{array}{ccc|c} 2 & 2 & 3 & 1 \\ 0 & 2 & \frac{1}{2} & -\frac{17}{2} \\ 0 & 0 & -\frac{3}{4} & -\frac{9}{4} \end{array}\right) \sim \left(\begin{array}{ccc|c} 2 & 2 & 3 & 1 \\ 0 & 2 & \frac{1}{2} & -\frac{17}{2} \\ 0 & 0 & 1 & 3 \end{array}\right) \sim$$
$$\sim \left(\begin{array}{ccc|c} 2 & 2 & 0 & -8 \\ 0 & 2 & 0 & -10 \\ 0 & 0 & 1 & 3 \end{array}\right) \sim \left(\begin{array}{ccc|c} 2 & 0 & 0 & 8 \\ 0 & 1 & 0 & -5 \\ 0 & 0 & 1 & 3 \end{array}\right) \sim$$
$$\sim \left(\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -5 \\ 0 & 0 & 1 & 3 \end{array}\right),$$

звідки

$$\begin{cases} x_1 - 1 \\ x_2 = -5 \\ x_3 = 3, \end{cases}$$

◄

Приклад 2.1.2. Методом Гаусса обчисліть визначник матриці

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 4 \end{pmatrix}.$$

► Оскільки $\det A = a_{11} a_{12}^{(1)} a_{33}^{(2)}$, то проведемо обчислення за схемою

$$m_2^{(1)} = \frac{a_{21}}{a_{11}} = 2, \quad m_3^{(1)} = \frac{a_{31}}{a_{11}} = 3, \quad a_{22}^{(1)} = a_{22} - m_2^{(1)} a_{12} = -1,$$
$$a_{23}^{(1)} = a_{23} - m_2^{(1)} a_{13} = -2, \quad a_{32}^{(1)} = a_{32} - m_3^{(1)} a_{12} = -2,$$
$$a_{33}^{(1)} = a_{33} - m_3^{(1)} a_{13} = -5, \quad m_3^{(2)} = \frac{a_{32}^{(1)}}{a_{22}^{(1)}} = 2,$$
$$a_{33}^{(2)} = a_{33}^{(1)} - m_3^{(1)} a_{23}^{(1)} = -1.$$

Отож, дана матриця A еквівалентна матриці

$$\widetilde{A} = \begin{pmatrix} 1 & 2 & 3 \\ 0 & -1 & -2 \\ 0 & 0 & -1 \end{pmatrix},$$

причому $\det A = \det \widetilde{A} = 1 \cdot (-1) \cdot (-1) = 1$.

◄

Приклад 2.1.3. Використовуючи метод Гаусса обчисліть обернену матрицю A^{-1} до матриці

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 1 & 2 \\ 2 & 2 & 3 \end{pmatrix}.$$

► Використаємо спосіб

$$A|I \sim \quad I|A^{-1}.$$

Маємо

$$A|I = \left(\begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 1 & 1 & 2 & 0 & 1 & 0 \\ 2 & 2 & 3 & 0 & 0 & 1 \end{array}\right) \sim \left(\begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 0 & -1 & -1 & -1 & 1 & 0 \\ 0 & -2 & -3 & -2 & 0 & 1 \end{array}\right) \sim$$

$$\sim \left(\begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 0 & -1 & -1 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & -2 & 1 \end{array}\right) \sim \left(\begin{array}{ccc|ccc} 1 & 2 & 0 & 1 & -6 & 3 \\ 0 & 1 & 0 & 1 & -3 & 1 \\ 0 & 0 & 1 & 0 & 2 & -1 \end{array}\right) \sim$$

$$\sim \left(\begin{array}{ccc|ccc} 1 & 0 & 0 & -1 & 0 & 1 \\ 0 & 1 & 0 & 1 & -3 & 1 \\ 0 & 0 & 1 & 0 & 2 & -1 \end{array}\right) = \quad I|A^{-1}.$$

Отож, маємо

$$A^{-1} = \begin{pmatrix} -1 & 0 & 1 \\ 1 & -3 & 1 \\ 0 & 2 & -1 \end{pmatrix}.$$

Зробимо перевірку:

$$AA^{-1} = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 1 & 2 \\ 2 & 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} -1 & 0 & 1 \\ 1 & -3 & 1 \\ 0 & 2 & -1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

◀

2.1.2 . Метод прогонки

Найвідоміший різновид методу Гаусса — *метод прогонки*, який застосовують до систем з тридіагональними матрицями. Такі системи часто зустрічаються при розв'язанні крайових задач для звичайних диференціальних рівнянь і рівнянь із частинними похідними другого порядку методом сіток. Ці системи можуть бути записані у вигляді:

$$\left\{ \begin{array}{rcl} c_0 x_0 + b_0 x_1 & = & g_0, \\ a_i x_{i-1} + c_i x_i + b_i x_{i+1} & = & g_i, \quad i = \overline{1, N-1}, \\ a_n x_{N-1} + c_N x_N & = & g_N, \end{array} \right. \tag{2.1.11}$$

або

$$Hx = g,$$

де N – натуральне число, $x = (x_0, x_1, \ldots, x_N)^\top$ – вектор невідомих, $g = (g_0, g_1, \ldots, g_N)^\top$ – вектор вільних членів, а

$$H = \begin{pmatrix} c_0 & b_0 & 0 & \ldots & 0 & \ldots & 0 & 0 & 0\ldots & 0 & 0 & 0 \\ a_1 & c_1 & b_1 & \ldots & 0 & \ldots & 0 & 0 & 0\ldots & 0 & 0 & 0 \\ \ldots & \ldots & \ldots & & \ldots & & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & 0\ldots & & a_i & c_i & b_i\ldots & 0 & 0 & 0 \\ \ldots & \ldots & \ldots & & \ldots & & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & 0\ldots & & 0 & 0 & 0\ldots & a_{N-1} & c_{N-1} & b_{N-1} \\ 0 & 0 & 0 & \ldots & 0\ldots & & 0 & 0 & 0\ldots & 0 & a_N & c_N \end{pmatrix}$$

– тридіагональна квадратна матриця розмірності $N+1$.

Унаслідок прямого ходу в методі Гаусса із системи (2.1.11) одержимо систему з дводіагональною верхньотрикутною матрицею. Тому формули зворотнього ходу мають вигляд:

$$x_N = \beta_N, \quad x_i = \alpha_i x_{i+1} + \beta_i, \quad i = N-1, N-2, ..., 0, \tag{2.1.12}$$

де α_i, $i = \overline{0, N-1}$, β_i, $i = \overline{0, N}$, — деякі числа.

Для визначення значень α_i, β_i підставимо вираз $x_{i-1} = \alpha_{i-1}x_i + \beta_{i-1}$ у i-те, $i \in \{1, \dots, N\}$, рівняння системи (2.1.11). У результаті отримаємо систему

$$c_0x_0 + b_0x_1 = g_0, \quad (a_i\alpha_{i-1} + c_i)x_i + b_ix_{i+1} = g_i - a_i\beta_{i-1}, \quad i = \overline{1, N-1},$$

$$(a_N\alpha_{N-1} + c_N)x_N = g_N - a_N\beta_{N-1}. \tag{2.1.13}$$

Рівняння (2.1.13) розв'яжемо відносно $x_0, x_1, \dots, x_N$ і в результаті одержимо

$$x_0 = -\frac{b_0}{c_0}x_1 + \frac{g_0}{c_0}, \quad x_i = -\frac{b_i}{a_i\alpha_{i-1} + c_i}x_{i+1} + \frac{g_i - a_i\beta_{i-1}}{a_i\alpha_{i-1} + c_i}, \quad i = \overline{1, N-1},$$

$$x_N = \frac{g_N - a_N\beta_{N-1}}{a_N\alpha_{N-1} + c_N}. \tag{2.1.14}$$

Порівнюючи рівність (2.1.12) і (2.1.14), отримаємо

$$\begin{cases} \alpha_0 := -\dfrac{b_0}{c_0}, \quad \beta_0 := \dfrac{g_0}{c_0}, \\[2ex] \alpha_i = -\dfrac{b_i}{a_i\alpha_{i-1} + c_i}, \quad \beta_i = \dfrac{g_i - a_i\beta_{i-1}}{a_i\alpha_{i-1} + c_i}, \quad i = \overline{1, N-1}, \\[2ex] \beta_N := \dfrac{g_N - a_N\beta_{N-1}}{a_N\alpha_{N-1} + c_N}. \end{cases} \tag{2.1.15}$$

Отже, **алгоритм методу прогонки** має вигляд:

1 крок. Обчислити прогоночні коефіцієнти за формулами (2.1.15);

2 крок. Знайти розв'язок системи (2.1.11) за формулами (2.1.12).

Зауваження 2.1.4. *Формули прогонки* можна застосувати, якщо знаменники у виразах (2.1.15) не перетворюються в нуль.

Приклад 2.1.4. Методом прогонки розв'язати СЛАР:

$$\begin{cases} x_0 + 3x_1 = 4 \\ x_0 + 2x_1 - x_2 = 2 \\ x_1 + 4x_2 + x_3 = 6 \\ x_2 + 4x_3 = 5. \end{cases} \iff \begin{pmatrix} 1 & 3 & 0 & 0 \\ 1 & 2 & -1 & 0 \\ 0 & 1 & 4 & 1 \\ 0 & 0 & 1 & 4 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 4 \\ 2 \\ 6 \\ 5 \end{pmatrix}.$$

Розв'язування. Маємо

$$c_0 = 1; \quad b_0 = 3; \quad g_0 = 4;$$
$$a_1 = 1; \quad c_1 = 2; \quad b_1 = -1; \quad g_1 = 2;$$
$$a_2 = 1; \quad c_2 = 4; \quad b_2 = 1; \quad g_2 = 6;$$
$$a_3 = 1; \quad c_3 = 4; \quad g_3 = 5.$$

Тоді на підставі формул (2.1.15) знаходимо

$$\alpha_1 = -\frac{b_0}{c_0} = -\frac{3}{1} = -3; \quad \beta_1 = \frac{g_0}{c_0} = \frac{4}{1} = 4;$$
$$\alpha_2 = -\frac{b_1}{a_1\alpha_1 + c_1} = -\frac{-1}{1\cdot(-3) + 2} = -\frac{1}{1} = -1; \quad \beta_2 = \frac{g_1 - a_1\beta_1}{a_1\alpha_1 + c_1} = \frac{2 - 1\cdot 4}{1\cdot(-3) + 2} = -\frac{-2}{-1} = 2;$$
$$\alpha_3 = -\frac{b_2}{a_2\alpha_2 + c_2} = -\frac{-1}{1\cdot(-1) + 4} = -\frac{1}{3}; \quad \beta_3 = \frac{g_2 - a_2\beta_2}{a_2\alpha_2 + c_2} = \frac{6 - 1\cdot 2}{3} = \frac{4}{3};$$
$$\beta_4 = \frac{g_3 - a_3\beta_3}{a_3\alpha_3 + c_3} = \frac{5 - \frac{4}{3}}{1\cdot(-\frac{1}{3}) + 4} = \frac{11}{3} : \frac{11}{3} = 1.$$

Тоді на підставі формул (2.1.12) знаходимо

$$\begin{aligned} x_3 &= \beta_4 = 1; \\ x_2 &= \alpha_3 x_3 + \beta_3 = -\frac{1}{3} \cdot 1 + \frac{4}{3} = 1; \\ x_1 &= \alpha_2 x_2 + \beta_2 = (-1) \cdot 1 + 2 = 1; \\ x_0 &= \alpha_1 x_1 + \beta_1 = -3 \cdot 1 + 4 = 1. \end{aligned}$$

Отже, $(1; 1; 1; 1)$ — розв'язок даної системи. □

Вправи для самостійної роботи

I. Методом Гаусса розв'яжіть СЛАР $Ax = b$ та знайдіть $\det A$ і A^{-1}, якщо

$$A = \begin{pmatrix} 0 & 3 & 1 \\ 7 & -13 & -2 \\ 1 & 2 & 4 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 2 \\ 4 \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}.$$

II. Методом прогонки розв'яжіть СЛАР

$$\begin{cases} 2x_0 + 5x_1 &= 7 \\ x_0 - 3x_1 + 2x_2 &= 0 \\ x_1 + 6x_2 - x_3 &= 6 \\ x_2 + 4x_3 &= 5. \end{cases}$$

2.2. Ітераційні методи розв'язування коректних СЛАР

2.2.1. Довідкова інформація

Під $\mathbb{R}^n$, де $n \geqslant 2$, розумітимемо лінійний простір. складений з впорядкованих наборів $n-$ок дійсних чисел $x := \begin{pmatrix} x_1 \\ \dots \\ x_n \end{pmatrix} \equiv (x_1, \dots, x_n)^\top$, наділений лінійними операціями:
1) $x + y := (x_1 + y_1, ..., x_n + y_n)^\top$,
2) $\lambda x := (\lambda x_1, ..., \lambda x_n)^\top$
для будь-яких $x = (x_1, ..., x_n)^\top$, $y = (y_1, ..., y_n)^\top$, $\lambda \in \mathbb{R}$.

На просторі $\mathbb{R}^n$ вводять норму одним із таких трьох способів:

$$\|x\|_1 := \max_{1 \leqslant i \leqslant n} |x_i|, \quad \|x\|_2 := \sum_{i=1}^n |x_i|, \quad \|x\|_3 := (\sum_{i=1}^n |x_i|^2)^{1/2}.$$

Далі, якщо не матиме принципового значення, яка з норм на $\mathbb{R}^n$ розглядається, писатимемо $\|x\|$ замість $\|x\|_1$, $\|x\|_2$ чи $\|x\|_3$.

Як відомо, коли маємо норму $\|\cdot\|$ на лінійному просторі, то відстань між його елементами x та y виражається через $\|x - y\|$. Тому кажуть, що послідовність $\{x^k = (x_1^k, ..., x_n^k)^\top\}_{k=1}^\infty$ елементів $\mathbb{R}^n$ *збігається* до елемента $x^* = (x_1^*, ..., x_n^*)^\top$, якщо $\|x^k - x^*\| \to 0$ при $k \to \infty$. Тоді елемент x^* називають *границею* послідовності $\{x^k\}_{k=1}^\infty$, а саму послідовність – *збіжною*.

Позначимо через $\mathbb{R}^{n \times n}$ лінійний простір, складений з квадратичних матриць розміру n з дійсними елементами, тобто елементами цього простору є квадратичні таблиці дійсних чисел, які мають n рядків та n стовпців:

$$A = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{n1} & \dots & a_{nn} \end{pmatrix}.$$

Добутком матриці $A \in \mathbb{R}^{n \times n}$ на вектор $x \in \mathbb{R}^n$ називають вектор $Ax \in \mathbb{R}^n$, визначений за правилом:

$$Ax = \begin{pmatrix} a_{11}x_1 + & \dots & +a_{1n}x_n \\ \dots & \dots & \dots \\ a_{n1}x_1 + & \dots & +a_{nn}x_n \end{pmatrix} \equiv \begin{pmatrix} \sum_{j=1}^n a_{1j}x_j \\ \dots \\ \sum_{j=1}^n a_{nj}x_j \end{pmatrix} \equiv$$
$$\equiv \left(\sum_{j=1}^n a_{1j}x_j, \quad \dots, \quad \sum_{j=1}^n a_{nj}x_j\right)^\top.$$

На лінійному просторі $\mathbb{R}^{n \times n}$ задають норми $\|\cdot\|_l$, $\quad l = 1, 2, 3$, так, щоби виконувалися нерівності

$$\|Ax\|_l \leqslant \|A\|_l \|x\|_l, \quad l = 1, 2, 3.$$

Розглянемо способи визначення $\|A\|_l$, $l = 1, 2, 3$. Маємо

$$\|Ax\|_1 = \max_{1 \leqslant i \leqslant n} |\sum_{j=1}^n a_{ij}x_j| \leqslant \max_{1 \leqslant i \leqslant n} \sum_{j=1}^n |a_{ij}|\,|x_j| \leqslant$$
$$\leqslant \Big(\max_{1 \leqslant i \leqslant n} \sum_{j=1}^n |a_{ij}|\Big)\Big(\max_{1 \leqslant j \leqslant n} |x_j|\Big) = \|A\|_1 \|x\|_1,$$

а отже,

$$\|A\|_1 := \max_{1 \leqslant i \leqslant n} \sum_{j=1}^n |a_{ij}|. \tag{2.2.1}$$

Тепер розглянемо

$$\|Ax\|_2 = \sum_{i=1}^n \Big|\sum_{j=1}^n a_{ij}x_j\Big| \leqslant \sum_{i=1}^n \sum_{j=1}^n |a_{ij}||x_j| =$$
$$= \sum_{j=1}^n \sum_{i=1}^n |a_{ij}||x_j| \leqslant \Big(\max_{1 \leqslant j \leqslant n} \sum_{i=1}^n |a_{ij}|\Big)\Big(\sum_{j=1}^n |x_j|\Big) = \|A\|_2 \|x\|_2,$$

тобто

$$\|A\|_2 := \max_{1\leqslant j\leqslant n} \sum_{i=1}^{n} |a_{ij}|. \tag{2.2.2}$$

Маємо, використовуючи нерівність Коші-Буняковського:

$$\|Ax\|_3^2 = \sum_{i=1}^{n} \Big(\sum_{j=1}^{n} a_{ij}x_j \Big)^2 \leqslant \Big(\sum_{i,j=1}^{n} |a_{ij}|^2 \Big) \Big(\sum_{j=1}^{n} |x_j|^2 \Big) = \|A\|_3^2 \|x\|_3^2,$$

тобто

$$\|A\|_3 := \Big(\sum_{i,j=1}^{n} |a_{ij}|^2 \Big)^{1/2}. \tag{2.2.3}$$

Розглянемо *матричні ряди*. Нехай $\{a_k\}_{k=0}^{\infty} \subset \mathbb{R}$, $\{A_k\}_{k=0}^{\infty} \subset \mathbb{R}^{n\times n}$ — послідовності, відповідно, дійсних чисел і матриць. Розглянемо матричний ряд

$$a_0A_0 + a_1A_1 + \ldots + a_kA_k + \ldots \ . \tag{2.2.4}$$

Частинними сумами цього ряду є матриці

$$S_k := a_0A_0 + a_1A_1 + \ldots + a_kA_k, \quad k \in \mathbb{N}_0 := \mathbb{N} \cup \{0\}. \tag{2.2.5}$$

Ряд (2.2.4) називають *збіжним*, якщо існує матриця $S \in \mathbb{R}^{n\times n}$ така, що

$$\|S - S_k\| \to 0 \quad \text{при} \quad k \to \infty,$$

де $\|\cdot\|$ — норма в $\mathbb{R}^{n\times n}$. Матрицю S називають *сумою* ряду (2.2.4).

Далі на просторі $\mathbb{R}^{n\times n}$ будемо розглядати тільки норми $\|\cdot\|_1$ і $\|\cdot\|_2$. Для них, зокрема, виконується нерівність

$$\|AB\|_l \leq \|A\|_l \|B\|_l, \quad l = 1, 2. \tag{2.2.6}$$

Нехай $B = \begin{pmatrix} b_{11} & b_{12} & \ldots & b_{1n} \\ b_{21} & b_{22} & \ldots & b_{2n} \\ \ldots & \ldots & \ldots & \\ b_{n1} & b_{n2} & \ldots & b_{nn} \end{pmatrix} \in \mathbb{R}^{n\times n}$. Розглянемо степеневий матричний ряд

$$I + B + B^2 + \ldots + B^k + \ldots \tag{2.2.7}$$

і встановимо умови його збіжності. Для цього нам потрібне рівняння

$$\det(B - \lambda I) = 0 \quad \Leftrightarrow \quad \begin{vmatrix} b_{11} - \lambda & b_{12} & \ldots & b_{1n} \\ b_{21} & b_{22} - \lambda & \ldots & b_{2n} \\ \ldots & \ldots & \ldots & \\ b_{n1} & b_{n2} & \ldots & b_{nn} - \lambda \end{vmatrix} = 0 \quad \Leftrightarrow$$

$$\Leftrightarrow \quad \lambda^n + a_1\lambda^{n-1} + \ldots + a_n = 0 \tag{2.2.8}$$

в полі комплексних чисел.

Нехай $\lambda_1, \ldots, \lambda_m$ $(1 \leq m \leq n)$ — всі корені рівняння (2.2.8). Як відомо, їх називають *власними значеннями* матриці B.

Теорема 2.2.1. *Ряд (2.2.7) збіжний тоді і лише тоді, коли*

$$|\lambda_j| < 1, \quad j = 1, \ldots, m, \tag{2.2.9}$$

і його сумою є матриця $(I - B)^{-1}$, обернена до $I - B$.

Доведення. Доведення цієї теореми можна знайти в [**?**]. □

Наслідок 2.2.1. *Якщо $\|B\| < 1$, то ряд (2.2.7) збіжний і його сумою є матриця $(I - B)^{-1}$ (обернена до $I - B$).*

Доведення. З умови даного твердження випливає виконання умови теореми 2.2.2, а звідси і висновок нашого твердження. Але легко можна довести дане твердження безпосередньо, що ми тут і зробимо.

Неважко переконатися, що нормований лінійний простір $\mathbb{R}^{n\times n}$ є повним, оскільки збіжність послідовності матриць в $\mathbb{R}^{n\times n}$ є еквівалентною збіжності n^2 числових послідовностей. Тому, нам досить показати, що послідовність частинних сум

$$S_k = I + B + B^2 + \ldots + B^k, \quad k \in \mathbb{N}_0 \tag{2.2.10}$$

є фундаментальною. Використовуючи (2.2.6), маємо для будь-яких $k, l \in \mathbb{N}$

$$\begin{aligned} &\|S_{k+l} - S_k\| = \|B^{k+1} + \ldots + B^{k+l}\| \leqslant \|B\|^{k+1} + \ldots + \|B\|^{k+l} = \\ &= \|B\|^{k+1}\left(1 + \ldots + \|B\|^{l-1}\right) \leqslant \|B\|^{k+1}\left(1 + \ldots + \|B\|^{l-1} + \ldots\right) = \\ &= \frac{\|B\|^{k+1}}{1 - \|B\|}. \end{aligned} \tag{2.2.11}$$

Тут ми використали формулу суми нескінченно спадної геометричної прогресії

$$1 + q + \ldots + q^{l-1} + \ldots = \frac{1}{1-q}, \quad |q| < 1.$$

З (2.2.11) і того, що $\|B\| < 1$, а отже, $\|B\|^{k+1} \to 0$ при $k \to \infty$, випливає, що для будь якого числа $\varepsilon > 0$ знайдеться число $k_0 \in \mathbb{N}$ таке, що

$$\|S_{k+l} - S_k\| < \varepsilon \quad \forall k \geqslant k_0, \ \forall l \geqslant 1.$$

Це означає, що послідовність (2.2.10) є фундаментальною, а отже, збіжною.

Те, що сумою ряду (2.2.7) є матриця $(I - B)^{-1}$ легко випливає з означення оберненої матриці і очевидної рівності

$$(I - B)(I + B + B^2 + \ldots + B^k + \ldots) = I.$$

Доведення завершене. □

2.2.2. Загальна схема застосування методу простих ітерацій

Розглянемо СЛАР

$$Ax = b, \tag{2.2.12}$$

де $n \geq 2$ — довільне натуральне число,

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \ldots \\ x_n \end{pmatrix} \in \mathbb{R}^n, \quad A = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{pmatrix} \in \mathbb{R}^{n\times n}, \quad b = \begin{pmatrix} b_1 \\ b_2 \\ \ldots \\ b_n \end{pmatrix} \in \mathbb{R}^n,$$

причому $\det A \neq 0$.

Перетворимо систему (2.2.12) так. Нехай $H \in \mathbb{R}^{n\times n}$, $\det H \neq 0$. Помножимо систему (2.2.12) на H зліва:

$$HAx = Hb. \tag{2.2.13}$$

Зокрема, множенню на H системи рівнянь (2.2.12) може відповідати перестановці місцями рівнянь цієї системи. Тепер виберемо довільно матриці $M, L \in \mathbb{R}^{n\times n}$, $\det M \neq 0$, такі, що

$$HA = M + L. \tag{2.2.14}$$

Тоді з (2.2.13) на підставі (2.2.14), поклавши $d := Hb$, отримаємо еквівалентну системі (2.2.12) систему

$$Mx = -Lx + d. \tag{2.2.15}$$

Побудуємо послідовність векторів з $\mathbb{R}^n$ — наближень розв'язку системи (2.2.12):

$$x^0, x^1, \ldots, x^k, \ldots, \tag{2.2.16}$$

за правилом

$$x^0 \ \text{— початкове наближення}, \tag{2.2.17}$$

$$Mx^{k+1} = -Lx^k + d, \quad k \in \mathbb{N}_0 := \mathbb{N} \cup 0. \tag{2.2.18}$$

Відмітимо, що початкове наближення x^0 вибирають довільно, але частіше всього беруть $x^0 := d$, де d — стовпчик вільних членів.

Твердження 2.2.1. *Якщо послідовність* (2.2.16) *є збіжною до* x^* *в* $\mathbb{R}^n$, *то* x^* *— розв'язок системи* (2.2.12).

Доведення. Маємо $\|x^k - x^*\| \to 0$ при $k \to \infty$, де $\|\cdot\|$ — яка-небудь із норм $\|\cdot\|_l$, $l = 1, 2, 3$. Тоді $\|Lx^k - Lx^*\| = \|L(x^k - x^*)\| \le \|L\|\|x^k - x^*\| \to 0$ при $k \to \infty$. Аналогічно доводимо, що $\|Mx^{k+1} - Mx^*\| \to 0$ при $k \to \infty$. Врахувавши це, переходимо до границі в (2.2.17) при $k \to \infty$. У результаті отримаємо рівність

$$Mx^* = -Lx^* + d,$$

звідки

$$(M + L)x^* = d \quad \Leftrightarrow \quad H^{-1} \times \Big| HAx^* = Hb \quad \Leftrightarrow \quad Ax^* = b,$$

що і треба було довести. □

Ітераційний процес (2.2.17) називають *методом простих ітерацій в явній формі*, якщо $M = I$, де I — одинична матриця в просторі $\mathbb{R}^{n\times n}$, і *методом простих ітерацій в неявній формі* в іншому випадку.

2.2.3. Застосування методу простих ітерацій в явній формі. Метод Якобі

Розглянемо детальніше випадок $M = I$, де

$$I = \begin{pmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \ldots & \ldots & \ldots & \\ 0 & 0 & \ldots & 1 \end{pmatrix} \quad \text{— одинична матриця.}$$

Тоді поклавши $B := -L$, систему (2.2.15) запишемо у вигляді

$$x = Bx + d. \tag{2.2.19}$$

де B — квадратна матриця розміру n, $d := (d_1, \ldots, d_n)^\top$ — стовпчик вільних членів.

Цю систему розв'язуємо методом *послідовних наближень*, а саме, будуємо послідовність

$$x^0, x^1, \ldots, x^k, \ldots \tag{2.2.20}$$

наближень розв'язку системи (2.2.19) за правилом:

$$x^0 \in \mathbb{R}^n \ \text{— початкове наближення}, \tag{2.2.21}$$

$$x^{k+1} = Bx^k + d, \quad k \in \mathbb{N}_0 := \mathbb{N} \cup \{0\}, \tag{2.2.22}$$

де використано позначення $x^k = (x_1^k, x_2^k, \ldots, x_n^k)^\top$, – k-та ітерація вектора $x = (x_1, \ldots, x_n)^\top$.

В скалярній формі цей процес можна записати у формі

$$x_i^0, \ i = \overline{1, n}, \ \text{– початкові наближення}, \tag{2.2.23}$$

$$x_i^{k+1} = \sum_{j=1}^{n} b_{ij}\, x_j^k + d_j, \quad i = \overline{1, n}, \quad k \in \mathbb{N}_0, \tag{2.2.24}$$

де x_i^k – k-та ітерація i-ої компоненти вектора $x = (x_1, \ldots, x_n)^\top$, $i = \overline{1,n}$.

Початкові наближення x_i^0, $i = \overline{1,n}$, задають довільно, але переважно за початкові наближення беруть елементи стовпчика вільних членів d.

Встановимо необхідну і достатню умови збіжності ітераційного процесу (2.2.21), (2.2.22).

Теорема 2.2.2. *Ітераційний процес* (2.2.21), (2.2.22) *збіжний тоді і тільки тоді, коли*

$$|\lambda_j| < 1, \quad j = \overline{1,m}, \tag{2.2.25}$$

де $\lambda_1, \ldots \lambda_m$ *— корені рівняння* (2.2.8).

Доведення. Доведення цієї теореми див. [**?**]. □

На практиці зручно використовувати достатню умову збіжності розглядуваного ітераційного процесу, яка дана в наступному твердженні.

Теорема 2.2.3. *Якщо*

$$\|B\|_l < 1 \tag{2.2.26}$$

для деякого $l \in \{1, 2\}$, *то послідовність* (2.2.20) *збіжна до розв'язку* x^* *системи* (2.2.19) *(за нормою* $\|\cdot\|_l$*) і виконується оцінка*

$$\|x^* - x^k\|_l \le \frac{\|B\|_l^k}{1 - \|B\|_l} \|x^1 - x^0\|_l, \quad k \in \mathbb{N}, \tag{2.2.27}$$

причому, якщо $x^0 = d$, *то*

$$\|x^* - x^k\|_l \le \frac{\|B\|_l^{k+1}}{1 - \|B\|_l} \|d\|, \quad k \in \mathbb{N}. \tag{2.2.28}$$

Крім того,

$$x^* = (I + B + B^2 + \ldots + B^k + \ldots)d \tag{2.2.29}$$

і наближення розв'язку системи (2.2.19) *можна шукати за формулою*

$$x^k := (I + B + B^2 + \ldots + B^k)d, \quad k \in \mathbb{N}_0. \tag{2.2.30}$$

Доведення. Згідно з процесом (2.2.21), (2.2.22) маємо

$$\begin{aligned}
&x^1 := Bx_0 + d, \quad x^2 := Bx_1 + d = B\left(Bx_0 + d\right) + d = B^2x^0 + Bd + d,\\
&x^3 := Bx_2 + d = B\left(B^2x_0 + Bd + d\right) + d = B^3x^0 + B^2d + Bd + d, \ldots,\\
&x^k := B^kx_0 + B^{k-1}d + \ldots + Bd + d = B^kx_0 + \left(I + B + B^2 + \ldots + B^{k-1}\right)d,\\
&k \in \mathbb{N}.
\end{aligned} \tag{2.2.31}$$

Оскільки, $\|B\|_l < 1$, то $\|B^kx^0\|_l \le \|B\|_l^k\|x^0\|_l \to 0$ при $k \to \infty$. Врахувавши це і перейшовши до границі при $k \to \infty$ в (2.2.31), отримаємо (2.2.29). Звідси випливає, що послідовність (2.2.30) збігається до x^* і правильна оцінка (2.2.28).

Покажемо правильність оцінки (2.2.27). Маємо з (2.2.10)

$$\|x^{j+1} - x^j\|_l = \|B\left(x^j - x^{j-1}\right)\|_l \le \|B\|_l\|\left(x^j - x^{j-1}\right)\|_l \le \ldots \le \|B\|_l^j\|\left(x^1 - x^0\right)\|_l.$$

Тоді для довільних $k, s \in \mathbb{N}$ маємо

$$\begin{aligned}
&\|x^{k+s} - x^k\|_l = \|x^{k+s} - x^{k+s-1} + x^{k+s-1} - \ldots + x^{k+1} - x^k\|_l \le\\
&\le \|x^{k+s} - x^{k+s-1}\|_l + \ldots + \|x^{k+1} - x^k\|_l \le\\
&\le \left(\|B\|_l^{k+s-1} + \cdots + \|B\|_l^k\right)\|x^1 - x^0\|_l \leqslant \frac{\|B\|_l^k}{1 - \|B\|_l}\|x^1 - x^0\|_l.
\end{aligned}$$

Перейшовши в цій нерівності до границі при $s \to \infty$, отримаємо (2.2.27). Доведення завершене. □

Зауваження 2.2.1. Оцінки (2.2.27), (2.2.27) використовують для знаходження кількості ітерацій, при якій досягають потрібної точності $\varepsilon > 0$. Справді, якщо $k \in \mathbb{N}$ є розв'язком нерівності

$$\frac{\|B\|_l^k}{1 - \|B\|_l} \|x^1 - x^0\|_l < \varepsilon, \tag{2.2.32}$$

то згідно з (2.2.27) матимемо

$$\|x^* - x^k\|_l < \varepsilon, \tag{2.2.33}$$

причому, якщо $x^0 = d$ і $k \in \mathbb{N}$ є розв'язком нерівності

$$\frac{\|B\|_l^{k+1}}{1 - \|B\|_l} \|d\| < \varepsilon, \tag{2.2.34}$$

то

$$\|x^* - x^k\|_l < \varepsilon. \tag{2.2.35}$$

Зауваження 2.2.2. Умову (2.2.33) можна деталізувати так:

$$\|B\|_1 := \max_{1 \leqslant i \leqslant n} \sum_{j=1}^{n} |b_{ij}| < 1\,,$$

або

$$\|B\|_2 := \max_{1 \leqslant j \leqslant n} \sum_{i=1}^{n} |b_{ij}| < 1\,.$$

Розглянемо важливий випадок методу простої ітерації в явній формі — так званий ***метод Якобі***. Припустимо, що $a_{ii} \neq 0$, $i = \overline{1, n}$ в СЛАР (2.2.12). Нехай

$$D = \begin{pmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \dots & \dots & \dots & \\ 0 & 0 & \dots & a_{nn} \end{pmatrix} \quad -$$

діагональна матриця, головна діагональ якої збігається з головною діагоналлю матриці A. Покладемо

$$H := D^{-1} = \begin{pmatrix} \dfrac{1}{a_{11}} & 0 & \dots & 0 \\ 0 & \dfrac{1}{a_{22}} & \dots & 0 \\ \dots & \dots & \dots & \\ 0 & 0 & \dots & \dfrac{1}{a_{nn}} \end{pmatrix}, \tag{2.2.36}$$

де D^{-1} — обернена до D матриця. Тоді

$$HA = D^{-1}A = \begin{pmatrix} 1 & \dfrac{a_{12}}{a_{11}} & \dots & \dfrac{a_{1n}}{a_{11}} \\ \dfrac{a_{21}}{a_{22}} & 1 & \dots & \dfrac{a_{2n}}{a_{22}} \\ \dots & \dots & \dots & \dots \\ \dfrac{a_{n1}}{a_{nn}} & \dfrac{a_{n2}}{a_{nn}} & \dots & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{pmatrix} +$$

$$+\begin{pmatrix} 0 & \frac{a_{12}}{a_{11}} & \dots & \frac{a_{1n}}{a_{11}} \\ \frac{a_{21}}{a_{22}} & 0 & \dots & \frac{a_{2n}}{a_{22}} \\ \dots & \dots & \dots & \dots \\ \frac{a_{n1}}{a_{nn}} & \frac{a_{n2}}{a_{nn}} & \dots & 0 \end{pmatrix} \equiv M + L,$$

де

$$M := \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{pmatrix}, \quad L := \begin{pmatrix} 0 & \frac{a_{12}}{a_{11}} & \dots & \frac{a_{1n}}{a_{11}} \\ \frac{a_{21}}{a_{22}} & 0 & \dots & \frac{a_{2n}}{a_{22}} \\ \dots & \dots & \dots & \dots \\ \frac{a_{n1}}{a_{nn}} & \frac{a_{n2}}{a_{nn}} & \dots & 0 \end{pmatrix}. \tag{2.2.37}$$

Тоді (2.2.17) матиме вигляд

$$x_i = -\sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j - \sum_{j=i+1}^{N} \frac{a_{ij}}{a_{ii}} x_j + \frac{b_i}{a_{ii}}, \quad i = \overline{1,n}. \tag{2.2.38}$$

Вважаємо, що значення суми дорівнює нулю, якщо верхня границя сумування менша за нижню.

В *методі Якобі* виходять з запису системи у вигляді (2.2.38), причому ітерації визначають так:

$$x_i^0 \ \text{— довільно вибране,}$$

$$x_i^{k+1} = -\sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^k - \sum_{j=i+1}^{n} \frac{a_{ij}}{a_{ii}} x_j^k + \frac{b_i}{a_{ii}}, \quad i = \overline{1,n}, \quad k \in \mathbb{N}_0 := \mathbb{N} \cup \{0\}. \tag{2.2.39}$$

Виникає питання про умови збіжності ітераційного процесу Якобі. Відповідь на нього в таких двох твердженнях.

Теорема 2.2.4. *Ітераційний процес Якобі* (2.2.39) *збіжний тоді й тільки тоді, коли*

$$|\lambda_j| < 1, \quad j = 1, \dots, m,$$

де $\lambda_j, \quad j = 1, \dots, m,$ *– всеможливі корені алгебраїчного рівняння*

$$\begin{vmatrix} \lambda a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & \lambda a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \\ a_{n1} & a_{n2} & \dots & \lambda a_{nn} \end{vmatrix} = 0 \tag{2.2.40}$$

в полі $\mathbb{C}$.

Доведення. Ітераційний процес Якобі має вигляд (2.2.21), (2.2.22) з $B = -L$. На підставі теореми 2.2.2 маємо, що процес Якобі є збіжний тоді й лише тоді, коли $|\lambda_j| < 1, \quad j = 1, \dots, m$, де $\lambda_j, \quad j = 1, \dots, m$, — корені рівняння

$$\det(-L - \lambda I) = 0. \tag{2.2.41}$$

Але це рівняння рівносильне рівнянню

$$\det(L + \lambda I) = 0.$$

На підставі рівності

$$\det(C \cdot D) = \det C \cdot \det D,$$

маємо

$$\det(L+\lambda I) = \det(D^{-1}D(L+\lambda I)) = \det D^{-1}\cdot\det(DL+\lambda D).$$

Звідси отримаємо, що рівняння (2.2.40) рівносильне рівнянню

$$\det(DL+\lambda D) = 0, \tag{2.2.42}$$

тобто рівнянню (2.2.40). Доведення завершене.

□

Наслідок 2.2.2. *Якщо виконується одна з двох умов:*

1)

$$\max_{1\leqslant i\leqslant n}\sum_{j=1,j\neq i}^{n}|a_{ij}/a_{ii}| < 1,$$

2)

$$\max_{1\leqslant j\leqslant n}\sum_{i=1,i\neq j}^{n}|a_{ij}/a_{ii}| < 1,$$

то ітераційний процес Якобі збігається.

Доведення. Це твердження є безпосереднім наслідком теореми 2.2.3 (див. зауваження 1). □

Приклад 2.2.1. Методом Якобі знайдіть перші три ітераційних наближення (враховуючи початкове) розв'язку СЛАР

$$\begin{cases} 10x_1 + x_2 + 2x_3 &= 18 \\ x_1 + 5x_2 - x_3 &= 8 \\ x_1 - 2x_2 + 10x_3 &= 27 \end{cases} \tag{2.2.43}$$

а також

1) перевірте умови збіжності ітераційного процесу,

2) знайдіть відхилення третьої ітерації від точного розв'язку,

3) визначте значення $k \in \mathbb{N}$, при якому k-те наближення у відповідній нормі відрізняється від точного розв'язку не більше від заданого числа $\varepsilon > 0$.

Розв'язування. Поділимо перше рівняння на коефіцієнт біля x_1, тобто на 10, друге рівняння — на коефіцієнт біля x_2, тобто на 5, а третє рівняння — на коефіцієнт біля x_3, тобто на 10. У результаті отримаємо

$$\begin{cases} x_1 + 0,1x_2 + 0,2x_3 &= 1,8 \\ 0,2x_1 + x_2 - 0,2x_3 &= 1,6 \\ 0,1x_1 - 0,2x_2 + x_3 &= 2,7\,. \end{cases} \tag{2.2.44}$$

Для побудови ітераційного процесу Якобі в системі (2.2.44) зробимо такі перетворення: в першому рівнянні перенесемо другий і третій члени лівої частини в праву частину, в другому рівнянні перенесемо перший і третій члени лівої частини в праву частину і в третьому члені перенесемо перший і другий члени лівої частини в праву частину:

$$\begin{cases} x_1 = -0,1x_2 - 0,2x_3 + 1,8 \\ x_2 = -0,2x_1 + 0,2x_3 + 1,6 \\ x_3 = -0,1x_1 + 0,2x_2 + 2,7 \end{cases}. \tag{2.2.45}$$

Ввівши позначення

$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \quad B = \begin{pmatrix} 0 & -0,1 & -0,2 \\ -0,2 & 0 & 0,2 \\ -0,1 & 0,2 & 0 \end{pmatrix}, \quad d = \begin{pmatrix} 1,8 \\ 1,6 \\ 2,7 \end{pmatrix},$$

систему (2.2.45) можна записати у вигляді

$$x = Bx + d. \tag{2.2.46}$$

Ітераційний *процес Якобі* будується так: задаємо довільно початкове наближення $x^0 \in \mathbb{R}^3$ і шукаємо наближення за формулою

$$x^{k+1} = Bx^k + d, \quad k = 0, 1, 2, \cdots \quad .$$

В скалярній формі цей процес запишемо так: нехай $(x_1^0, x_2^0, x_3^0)^\top$ — довільно задані початкові наближення розв'язку даної системи. Тоді наступні наближення отримують за формулами:

$$\begin{cases} x_1^{k+1} = -0,1x_2^k - 0,2x_3^k + 1,8 \\ x_2^{k+1} = -0,2x_1^k + 0,2x_3^k + 1,6 \\ x_3^{k+1} = -0,1x_1^k + 0,2x_2^k + 2,7 \end{cases} . \tag{2.2.47}$$

Перевіримо чи даний процес є збіжним. Маємо

$$\|B\|_1 := \max\{|-0,1| + |-0,2|; |-0,2| + |0,2|; |-0,1| + |0,2|\} = \\ = \max\{0,3; 0,4; 0,3\} = 0,4 < 1.$$

Також бачимо, що

$$\|B\|_2 := \max\{|-0,2| + |-0,1|; |-0,1| + |0,2|; |-0,2| + |0,2|\} = 0,4 < 1.$$

Отже, на підставі теореми 2.2.3 процес Якобі збіжний і

$$\|x^* - x^k\|_l \leqslant \frac{0,4^k}{1-0,4}\|x^1 - x^0\|_l, \quad k \in \mathbb{N}, \quad l = 1, 2. \tag{2.2.48}$$

Візьмемо такі початкові наближення: $x_1^0 := 1$; $x_2^0 := 2$; $x_3^0 := 1$, тобто $x^0 = (1; 2; 1)^\top$. Тоді з (2.2.47) маємо

$$x_1^1 := -0,1x_2^0 - 0,2x_3^0 + 1,8 = -0,1 \cdot 2 - 0,2 \cdot 1 + 1,8 = 1,4\,;$$
$$x_2^1 := -0,2x_1^0 + 0,2x_3^0 + 1,6 = -0,2 \cdot 1 + 0,2 \cdot 1 + 1,6 = 1,6\,;$$
$$x_3^1 := -0,1x_1^0 + 0,2x_2^0 + 2,7 = -0,1 \cdot 1 + 0,2 \cdot 2 + 2,7 = 3\,.$$

Отже, $x_1 = (1,4; 1,6; 3)^\top$ — перше наближення. Тоді знову використаємо (2.2.47) при $k = 1$:

$$x_1^2 := -0,1x_2^1 - 0,2x_3^1 + 1,8 = -0,1 \cdot 1,4 - 0,2 \cdot 3 + 1,8 = 1,06\,;$$
$$x_2^2 := -0,2x_1^1 + 0,2x_3^1 + 1,6 = -0,2 \cdot 1,4 + 0,2 \cdot 3 + 1,6 = 1,92\,;$$
$$x_3^2 := -0,1x_1^1 + 0,2x_2^1 + 2,7 = -0,1 \cdot 1,4 + 0,2 \cdot 3 + 2,7 = 2,88\,.$$

Отже, $x^2 := (1,64; 1,92; 2,88)^\top$ — друге наближення.

Знайдемо відхилення другого наближення $x^2 = (x_1^2; x_2^2; x_3^2)^\top$ від точного розв'язку $x^* = (x_1^*; x_2^*; x_3^*)^\top$. Для цього скористаємося оцінкою (2.2.48) при $l = 1$. Маємо

$$\|x^1 - x^0\|_1 := \max\{|x_1^1 - x_1^0|; |x_2^1 - x_2^0|; |x_3^1 - x_3^0|\} = \max\{0,4;\ 0,4;\ 2\} = 2; \\ 0,4^2 = 0,16. \tag{2.2.49}$$

Тоді на підставі (2.2.48), (2.2.49) маємо

$$\|x^* - x^2\|_1 \leq \frac{0,16}{0,6} \cdot 2 = \frac{8}{15}.$$

На підставі (2.2.48) маємо оцінки відхилень наближень розв'язки від точного розв'язку

$$\|x^* - x^k\|_1 \leqslant \frac{0,4^k}{0,3}, \quad k \in \mathbb{N}. \tag{2.2.50}$$

Визначимо значення $k \in \mathbb{N}$, при якому маємо

$$\|x^* - x^k\|_1 < \varepsilon. \tag{2.2.51}$$

З (2.2.50), (2.2.51) випливає, потрібне значення $k \in \mathbb{N}$ знаходимо як розв'язок нерівності

$$\frac{0,4^k}{0,3} < \varepsilon. \tag{2.2.52}$$

Звідси знаходимо

$$k > \log_{0,4}(0,3\,\varepsilon).$$

Отож, якщо $\varepsilon \in (0;1)$, то $\|x^* - x^k\|_1 < \varepsilon$ при $k := [\log_{0,4}(0,3\,\varepsilon)] + 1$. □

2.2.4. Застосування методу простої ітерації в неявній формі. Метод Зейделя

Вияснимо умови збіжності ітераційного процесу (2.2.17), (2.2.18) у випадку $M \neq I$, тобто методу простої ітерації в неявній формі.

Теорема 2.2.5. *Ітераційний процес* (2.2.17), (2.2.18) *збіжний тоді й лише тоді, коли*

$$\max_{1 \leqslant j \leqslant m} |\lambda_j| < 1,$$

де $\lambda_1, \ldots, \lambda_m$ *— всеможливі корені рівняння*

$$\det(L + \lambda M) = 0 \tag{2.2.53}$$

у полі комплексних чисел $\mathbb{C}$.

Доведення. Ітераційний процес (2.2.17), (2.2.18) можна записати у вигляді:

$$x^{k+1} = -M^{-1}Lx^k + M^{-1}d.$$

Тепер використаємо теорему 2.2.2 з $B = -M^{-1}L$. Звідси отримаємо, що процес Зейделя збіжний тоді і лише тоді, коли $|\lambda_j| < 1,\ j = 1, \ldots, m$, де $\lambda_j,\ j = 1, \ldots, m$, — всеможливі корені рівняння

$$\det\left(-M^{-1}L - \lambda I\right) = 0. \tag{2.2.54}$$

Використовуючи властивість

$$\det(C \cdot D) = \det C \cdot \det D,$$

маємо

$$\det\left(-M^{-1}L - \lambda I\right) = \det\left(-M^{-1}(L + \lambda M)\right) = \det(-M^{-1}) \cdot \det(L + \lambda M).$$

Звідси отримаємо, що рівняння (2.2.40) рівносильне рівнянню (2.2.60). □

Зауваження 2.2.3. Як видно з доведення теореми 2.2.3 достатньою умовою збіжності ітераційного процесу (2.2.17), (2.2.18) є умова

$$\|M^{-1}L\| < 1. \tag{2.2.55}$$

Тепер розглянемо випадок ітераційного процесу в неявній формі — так званий ***метод Зейделя***. Для цього помножимо векторне рівняння (2.2.12) на введену вище (див. (2.2.36)) матрицю H:

$$HAx = Hb,$$

і запишемо

$$HA = \begin{pmatrix} 1 & \frac{a_{12}}{a_{11}} & \dots & \frac{a_{1n}}{a_{11}} \\ \frac{a_{21}}{a_{22}} & 1 & \dots & \frac{a_{2n}}{a_{22}} \\ \dots & \dots & \dots & \dots \\ \frac{a_{n1}}{a_{nn}} & \frac{a_{n2}}{a_{nn}} & \dots & 1 \end{pmatrix} =$$

$$= \begin{pmatrix} 1 & 0 & \dots & 0 \\ \frac{a_{21}}{a_{22}} & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ \frac{a_{n1}}{a_{nn}} & \frac{a_{n2}}{a_{nn}} & \dots & 1 \end{pmatrix} + \begin{pmatrix} 0 & \frac{a_{12}}{a_{11}} & \dots & \frac{a_{1n}}{a_{11}} \\ 0 & 0 & \dots & \frac{a_{2n}}{a_{22}} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 \end{pmatrix} \equiv M + L,$$

де

$$M := \begin{pmatrix} 1 & 0 & \dots & 0 \\ \frac{a_{21}}{a_{22}} & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ \frac{a_{n1}}{a_{nn}} & \frac{a_{n2}}{a_{nn}} & \dots & 1 \end{pmatrix}, \quad L := \begin{pmatrix} 0 & \frac{a_{12}}{a_{11}} & \dots & \frac{a_{1n}}{a_{11}} \\ 0 & 0 & \dots & \frac{a_{2n}}{a_{22}} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 \end{pmatrix}. \tag{2.2.56}$$

Тоді ітераційний процес *Зейделя* можна записати у вигляді:

$$x_i^{k+1} + \sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^{k+1} = -\sum_{j=i+1}^{n} \frac{a_{ij}}{a_{ii}} x_j^k + \frac{b_i}{a_{ii}}, \quad i = \overline{1,n}, \quad k \in \mathbb{N}_0. \tag{2.2.57}$$

Щоб зрозуміти, як знаходять звідси значення x_i^{k+1}, $i = \overline{1,n}$, запишемо детальніше перші два рівняння системи (2.2.57):

$$x_1^{k+1} = -\sum_{j=2}^{n} \frac{a_{1j}}{a_{11}} x_j^k + \frac{b_1}{a_{11}}, \tag{2.2.58}$$

$$x_2^{k+1} = -\frac{a_{21}}{a_{22}} x_1^{k+1} - \sum_{j=3}^{n} \frac{a_{2j}}{a_{22}} x_j^k + \frac{b_2}{a_{22}}. \tag{2.2.59}$$

Перша компонента x_1^{k+1} вектора x^{k+1} знаходиться з рівняння (2.2.58) явно, для її обчислення потрібно знати вектор x^k і значення b_1. При знаходженні x_2^{k+1} з рівняння (2.2.59) використовуються тільки що знайдене значення x_1^{k+1} і відомі значення x_j^k, $j = 3, 4, \dots, n$, з попередньої ітерації. Отже, компоненти вектора x^{k+1} знаходять з рівняння (2.2.57) послідовно, починаючи з $i = 1$.

Наслідок 2.2.3. *Ітераційний процес Зейделя збіжний тоді й лише тоді, коли*

$$|\lambda_j| < 1, \quad j = 1, \dots, m,$$

де λ_j, $j = 1, \dots, m$, *— всеможливі корені рівняння*

$$\begin{vmatrix} \lambda a_{11} & a_{12} & \dots & a_{1n} \\ \lambda a_{21} & \lambda a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \\ \lambda a_{n1} & \lambda a_{n2} & \dots & \lambda a_{nn} \end{vmatrix} = 0 \tag{2.2.60}$$

у полі комплексних чисел $\mathbb{C}$.

Приклад 2.2.2. Методом Зейделя розв'язати СЛАР

$$\begin{cases} 10x_1 + x_2 + 2x_3 &= 18 \\ x_1 + 5x_2 - x_3 &= 8 \\ x_1 - 2x_2 + 10x_3 &= 27\,. \end{cases} \tag{2.2.61}$$

Доведення. Для знаходження наближення розв'язку СЛАР *методом Зейделя* перетворимо систему (2.2.44) так: в першому рівнянні другий і третій члени лівої частини перенесемо в праву частину, в другому рівнянні третій член лівої частини перенесемо в праву частину, а третє рівняння залишимо без змін. У результаті отримаємо СЛАР

$$\begin{cases} x_1 = -0,1x_2 - 0,2x_3 + 1,8 \\ 0,2x_1 + x_2 = 0,2x_3 + 1,6 \\ 0,1x_1 - 0,2x_2 + x_3 = 2,7\,. \end{cases} \tag{2.2.62}$$

Поклавши

$$M := \begin{pmatrix} 1 & 0 & 0 \\ 0,2 & 1 & 0 \\ 0,1 & -0,2 & 1 \end{pmatrix}, \quad L := \begin{pmatrix} 0 & 0,1 & 0,2 \\ 0 & 0 & -0,2 \\ 0 & 0 & 0 \end{pmatrix}.$$

Систему (2.2.62) можемо записати у вигляді

$$Mx = -Lx + d. \tag{2.2.63}$$

На підставі (2.2.63) ітераційний процес Зейделя будуємо так: початкове наближення $x^0 = (x_1^0, x_2^0, x_3^0)^\top$ задаємо довільно, а решту шукаємо за ітераційною формулою

$$Mx^{k+1} = -Lx^k + d, \quad k = 0, 1, 2, \cdots, \tag{2.2.64}$$

де $x^k = (x_1^k, x_2^k, x_3^k)^\top$ — k-та ітерація. В скалярній формі процес (2.2.64) має вигляд

$$\begin{cases} x_1^{k+1} &:= -0,1x_2^k - 0,2x_3^k + 1,8 \\ 0,2x_1^{k+1} + x_2^{k+1} &:= 0,2x_3^k + 1,6 \\ 0,1x_1^{k+1} - 0,2x_2^{k+1} + x_3^{k+1} &:= 2,7 \end{cases} \iff$$
$$\iff \begin{cases} x_1^{k+1} &:= -0,1x_2^k - 0,2x_3^k + 1,8 \\ x_2^{k+1} &:= -0,2x_1^{k+1} + 0,2x_3^k + 1,6 \\ x_3^{k+1} &:= -0,1x_1^{k+1} + 0,2x_2^{k+1} + 2,7 \end{cases}. \tag{2.2.65}$$

Нехай

$$x_1^0 := 1;\ x_2^0 := 2;\ x_3^0 := 1; \quad \text{тобто} \quad x^0 = (1, 2, 1)^\top.$$

Тоді з (2.2.65) при $k = 0$ маємо

$$\begin{aligned} x_1^1 &:= -0,1x_2^0 - 0,2x_3^0 + 1,8 = -0,1 \cdot 2 - 0,2 \cdot 1 + 1,8 = 1,4\,; \\ x_2^1 &:= -0,2x_1^1 + 0,2x_3^0 + 1,6 = -0,2 \cdot 1,4 + 0,2 \cdot 1 + 1,6 = 1,52\,; \\ x_3^1 &:= -0,1x_1^1 + 0,2x_2^1 + 2,7 = -0,1 \cdot 1,4 + 0,2 \cdot 1,52 + 2,7 = 2,864\,. \end{aligned}$$

Отже, $x^1 = (x_1^1, x_2^1, x_3^1)^\top = (1,4; 1,52; 2,864)^\top$ — перше наближення. Аналогічно знаходимо друге наближення $x^2 = (x_1^2, x_2^2, x_3^2)^\top$ і т.д. □

Вправи для самостійної роботи

I. Методом простих ітерацій знайдіть перші три ітераційних наближення (враховуючи початкове) розв'язку СЛАР

$$\begin{cases} x_1 &= 0,1x_1 - 0,2x_2 + 2, \\ x_2 &= 0,2x_1 - 0,3x_2 + 0,1x_3 + 1, \\ x_3 &= 0,4x_2 + 0,4x_3 - 1, \end{cases}$$

а також

1) перевірте умови збіжності ітераційного процесу,

2) знайдіть відхилення третьої ітерації від точного розв'язку,

3) визначте значення $k \in \mathbb{N}$, при якому k-те наближення у відповідній нормі відрізняється від точного розв'язку не більше від заданого числа $\varepsilon > 0$.

II. Методами Якобі та Зейделя знайдіть перші три ітераційних наближення (враховуючи початкове) розв'язку СЛАР

$$\begin{cases} 4x_1 + x_2 - x_3 &= 4, \\ x_1 - 8x_2 + x_3 &= -6, \\ x_2 + 10x_3 &= 12. \end{cases}$$

2.3. Знаходження псевдорозв'язків несумісних СЛАР

Нехай n, m — довільні фіксовані натуральні числа, причому $n \geq 2$. Розглянемо СЛАР

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_m, \end{cases} \tag{2.3.1}$$

де $a_{ij}, b_j, i = \overline{1,m}, j = \overline{1,n}$, — задані дійсні числа.

Позначимо

$$x := \begin{pmatrix} x_1 \\ x_2 \\ \ldots \\ x_n \end{pmatrix}, \quad A := \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{pmatrix}, \quad b := \begin{pmatrix} b_1 \\ b_2 \\ \ldots \\ b_m \end{pmatrix},$$

$$A|b = \left(\begin{array}{cccc|c} a_{11} & a_{12} & \ldots & a_{1n} & b_1 \\ a_{21} & a_{22} & \ldots & a_{2n} & b_2 \\ \ldots & \ldots & \ldots & & \ldots \\ a_{m1} & a_{m2} & \ldots & a_{mn} & b_m \end{array}\right).$$

Тут $x \in \mathbb{R}^n$ — стовпчик невідомих величин, $A \in \mathbb{R}^{m\times n}$ — основна матриця, $b \in \mathbb{R}^m$ — стовпчик вільних членів, $A|b \in \mathbb{R}^{m\times(n+1)}$ — розширена матриця системи (2.3.1). Систему (2.3.1) можна записати у вигляді

$$Ax = b.$$

Під розв'язком системи (2.3.1) розуміють впорядкований набір $x^* = (x_1^*, \ldots, x_n^*)^\top$ дійсних чисел такий, що $Ax^* = b$.

На питання про існування та єдиність розв'язку системи (2.3.1) дає відповідь теорема Кронеккера-Капеллі. Сформулюємо цю теорему, але перед цим нагадаємо відомий факт: якщо r — кількість лінійно незалежних рядків матриці H розміру $p\times q$ (p — кількість рядків, q — кількість стовпчиків), то кількість лінійно незалежних стовпчиків та найвищий з порядків ненульових мінорів матриці H теж дорівнюють r, причому $r \leqslant \min\{p, q\}$. Число r називають рангом матриці H й, як правило, позначають через $\operatorname{rang} H$.

Теорема 2.3.1 (теорема Кронекерра-Капеллі)**.** *Правильні такі твердження:*

1) *Якщо $\operatorname{rang} A = \operatorname{rang} A|b = n$, то система (2.3.1) має і тільки один розв'язок.*

2) *Якщо $\operatorname{rang} A = \operatorname{rang} A|b < n$, то система (2.3.1) має безліч розв'язків.*

3) *Якщо $\operatorname{rang} A \neq \operatorname{rang} A|b$ (тоді $\operatorname{rang} A < \operatorname{rang} A|b$), то система (2.3.1) не має розв'язків.*

Якщо система (2.3.1) має розв'язки (один або безліч), тобто виконуються умови тверджень 1 та 2 теореми 1, то кажуть що ця система є *сумісною*, а в протилежному випадку, тобто коли розв'язків немає, що рівносильно виконанню умови твердження 3, — *несумісною*. У випадку коли для системи (2.3.1) виконана умова твердження 1, її елементарними перетвореннями можна звести до еквівалентної системи, складеної з n рівнянь і визначник основної матриці нової системи буде відмінний від нуля. Такі системи ми розглядали в двох попередніх підрозділах. Коли ж виконуються умова твердження 2, то систему (2.3.1) можна розв'язувати методом Гаусса або якоюсь модифікацією цього методу, врахувавши, що система (2.3.1) елементарними перетвореннями зводиться до системи, в якої кількість рівнянь менша за кількість невідомих і деякі змінні є вільними.

Тут ми розглянемо випадок, коли для системи (2.3.1) виконується умова твердження 3. В цьому випадку вводиться поняття *псевдорозв'язку* системи (2.3.1).

Означення 2.3.1. *Псевдорозв'язком несумісної системи (2.3.1) (коли вона не має розв'язків) називають вектор $x^* \in \mathbb{R}^n$ такий, що*

$$\|Ax^* - b\| = \inf_{x\in\mathbb{R}^n} \|Ax - b\|,$$

де $\|\cdot\|$ — одна із норм в просторі $\mathbb{R}^m$.

Зауваження 2.3.1. Легко бачити, що коли система має розв'язки, то вони і тільки вони будуть її псевдорозв'язками. Це, зокрема, означає, що шукати псевдорозв'язки можна для будь-якої СЛАР і вони будуть розв'язками, коли ця система є сумісною.

Для визначення псевдорозв'язку, як правило, беруть норму $\|\cdot\|_3$ й означення псевдорозв'язку x^* базується на умові

$$\|Ax^* - b\|_3^2 = \inf_{x\in\mathbb{R}^n} \|Ax - b\|_3^2\,, \tag{2.3.2}$$

де

$$\|y\|_3 = \Big(\sum_{i=1}^m |y_i|^2\Big)^{1/2}, \quad y \in \mathbb{R}^m.$$

Теорема 2.3.2. *Псевдорозв'яки системи* (2.3.1) *існують та збігаються з розв'язками системи*

$$A^\top Ax = A^\top b. \tag{2.3.3}$$

Доведення. Введемо позначення

$$\begin{aligned} V(x) = \|Ax^* - b\|_3^2 = \sum_{i=1}^m (a_{i1}x_1 + \ldots + a_{ik}x_k + \ldots + a_{in}x_n - b_i)^2 \equiv \\ \equiv \sum_{i=1}^m \Big(\sum_{j=1}^n a_{ij}x_j - b_i\Big)^2, \quad x = (x_1, \ldots, x_n)^\top \in \mathbb{R}^n. \end{aligned} \tag{2.3.4}$$

Зі сказаного випливає, що якщо x^* псевдорозв'язок системи (2.3.1), то

$$V(x^*) = \inf_{x\in\mathbb{R}^n} V(x).$$

З необхідної умови локального мінімуму функції багатьох змінних випливає, що коли x^* — псевдорозв'язок, то

$$\frac{\partial V(x^*)}{\partial x_k} = 0, \quad k = \overline{1,n},$$

тобто x^* є розв'язком системи

$$\frac{\partial V(x)}{\partial x_k} = 0, \quad k = \overline{1,n}. \tag{2.3.5}$$

Деталізуємо систему (2.3.5). Як легко бачити (див. (2.3.4)), маємо

$$\begin{aligned} \frac{\partial V(x)}{\partial x_k} &= \sum_{i=1}^m 2\,(a_{i1}x_1 + \ldots + a_{ik}x_k + \ldots + a_{in} - b_i)\cdot a_{ik} = \\ &= 2\left[\left(\sum_{i=1}^m a_{ik}a_{i1}\right)x_1 + \ldots + \left(\sum_{i=1}^m a_{ik}a_{il}\right)x_l + \ldots + \right. \\ &\left. + \left(\sum_{i=1}^m a_{ik}a_{in}\right)x_n - \sum_{i=1}^m a_{ik}b_i\right] \equiv 2\,[c_{k1}x_1 + \ldots + c_{kl}x_l + \ldots + c_nx_n - d_k]\,, \end{aligned} \tag{2.3.6}$$

де

$$c_{kl} := \sum_{i=1}^m a_{ik}a_{il}, \quad d_k := \sum_{i=1}^m a_{ik}b_i, \quad k,l = \overline{1,n}. \tag{2.3.7}$$

Нагадаємо, що транспонована до A матриця $A^\top$ має вигляд

$$A^\top := \begin{pmatrix} a_{11} & a_{21} & \ldots & a_{m1} \\ a_{12} & a_{22} & \ldots & a_{m2} \\ \ldots & \ldots & \ldots & \\ a_{1n} & a_{2n} & \ldots & a_{mn} \end{pmatrix}.$$

Звідси і формули (2.3.7) отримуємо, що $c_{k,l}$, $k,l = \overline{1,n}$, є елементами матриці $A^\top A$, а d_k, $k = \overline{1,n}$, — елементи вектора $A^\top b$. Отже, систему (2.3.5), враховуючи (2.3.6), можна записати у вигляді системи (2.3.3). Ця система містить n рівнянь з n невідомими. Очевидно, що коли $\det A^\top A \neq 0$, то

система (2.3.3) має і тільки один розв'язок. Покажемо, що у випадку $\det A^\top A = 0$ система має безліч розв'язків для довільного $b \in \mathbb{R}^n$. Для цього використаємо альтернативу Фредгольма. Оскільки, матриця $A^\top A$ симетрична, бо $(A^\top A)^\top = A^\top (A^\top)^\top = A^\top A$, і дійсна, то вона самоспряжена, а тому система (2.3.3) буде сумісною, якщо

$$\left(A^\top b, x\right) = 0$$

для кожного розв'язку x системи $A^\top Ax = 0$, де $(\cdot,\cdot)$ — скалярний добуток в $\mathbb{R}^n$, тобто

$$(x, y) = \sum_{i=1}^{n} x_i y_i, \quad x, y \in \mathbb{R}^n.$$

Маємо

$$A^\top Ax = 0 \quad \Rightarrow \quad (A^\top Ax, x) = 0 \quad \Rightarrow \quad (Ax, Ax) = 0 \quad \Rightarrow \quad Ax = 0.$$

Звідси випливає

$$\left(A^\top b, x\right) = (b, Ax) = 0.$$

Отже, на підставі альтернативи Фредгольма отримуємо, що система (2.3.3) має розв'язки, тобто є сумісною. Розв'язки цієї системи є псевдорозв'язками системи (2.3.1). Покажемо це. А для цього введемо такі позначення:

$$\nabla V(x) := \begin{pmatrix} \frac{\partial V(x)}{\partial x_1} \\ \dots \\ \frac{\partial V(x)}{\partial x_n} \end{pmatrix}, \quad \nabla^2 V(x) := \begin{pmatrix} \frac{\partial^2 V(x)}{\partial x_1^2} & \frac{\partial^2 V(x)}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 V(x)}{\partial x_1 dx_n} \\ \dots & \dots & \dots & \\ \frac{\partial^2 V(x)}{\partial x_n \partial x_1} & \frac{\partial^2 V(x)}{\partial x_n \partial x_2} & \dots & \frac{\partial^2 V(x)}{\partial x_n^2} \end{pmatrix}.$$

За відомою формулою Тейлора маємо

$$V(x) = V(x^*) + (\nabla V(x^*), x - x^*) + \frac{1}{2}\left(\nabla^2 V(y)(x - x^*), x - x^*\right), \tag{2.3.8}$$

де $y \in \mathbb{R}^n$ — деякий елемент з $\mathbb{R}^n$. Оскільки, x^* — розв'язок системи (2.3.5), то $\nabla V(x^*) = 0$. З (2.3.6) маємо

$$\frac{\partial^2 V(x)}{\partial x_l \partial x_k} = 2c_{kl} \quad \Leftrightarrow \quad \nabla^2 V(y) = 2A^\top A.$$

Звідси та з (2.3.8) отримаємо

$$V(x) = V(x^*) + \left(A^\top A(x - x^*), x - x^*\right). \tag{2.3.9}$$

Але $(A^\top Az, z) = (Az, Az) \geqslant 0$ для будь-якого $z \in \mathbb{R}^n$, тобто

$$(A^\top (x - x^*), x - x^*) \geqslant 0 \quad \forall x \in \mathbb{R}^n.$$

Звідси та з (2.3.9) маємо $V(x) \geqslant V(x^*)$ для кожного $x \in \mathbb{R}^n$, тобто x^* — псевдорозв'язок системи (2.3.1). □

Зауваження 2.3.2. Відмітимо, що коли A — квадратна невироджена матриця, то система (2.3.3) рівносильна системі (2.3.1) і, очевидно, має і тільки один розв'язок, який, по суті, є єдиним псевдорозв'язком. Справді, оскільки матриця A є невиродженою, то невиродженою є матриця $A^\top$, а отже, помноживши систему (2.3.3) зліва на $(A^\top)^{-1}$, одержимо (2.3.1).

Приклад 2.3.1. Переконатися, що задана СЛАР є несумісною і знайти її псевдорозв'язки:

$$\begin{cases} x_1 + x_2 &= 2, \\ 2x_1 - x_2 &= 1, \\ 4x_1 + x_2 &= 6. \end{cases}$$

Розв'язування. Маємо

$$A = \begin{pmatrix} 1 & 1 \\ 2 & -1 \\ 4 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} 2 \\ 1 \\ 6 \end{pmatrix}, \quad A|b = \left(\begin{array}{cc|c} 1 & 1 & 2 \\ 2 & -1 & 1 \\ 4 & 1 & 6 \end{array}\right).$$

Зведемо матрицю $A|b$ до східчастого вигляду:

$$\left(\begin{array}{cc|c} 1 & 1 & 2 \\ 2 & -1 & 1 \\ 4 & 1 & 6 \end{array}\right) \sim \left(\begin{array}{cc|c} 1 & 1 & 2 \\ 0 & -3 & -3 \\ 0 & -3 & -2 \end{array}\right) \sim \left(\begin{array}{cc|c} 1 & 1 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{array}\right).$$

Звідси випливає, що $\operatorname{rang} A = 2$ і $\operatorname{rang} A|b = 3$, тобто дана система несумісна. Знайдемо її псевдорозв'язки. Для цього зауважимо, що

$$A^\top = \begin{pmatrix} 1 & 2 & 4 \\ 1 & -1 & 1 \end{pmatrix}, \quad A^\top A = \begin{pmatrix} 1 & 2 & 4 \\ 1 & -1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 2 & -1 \\ 4 & 1 \end{pmatrix} = \begin{pmatrix} 19 & 3 \\ 3 & 3 \end{pmatrix},$$

$$A^\top b = \begin{pmatrix} 1 & 2 & 4 \\ 1 & -1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 \\ 1 \\ 6 \end{pmatrix} = \begin{pmatrix} 28 \\ 7 \end{pmatrix}.$$

Отже, система для знаходження псевдорозв'язків (див. (2.3.3)) має вигляд

$$\begin{pmatrix} 19 & 3 \\ 3 & 3 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 28 \\ 7 \end{pmatrix}.$$

Маємо

$$\left(\begin{array}{cc|c} 19 & 3 & 28 \\ 3 & 3 & 7 \end{array}\right) \sim \left(\begin{array}{cc|c} 3 & 3 & 7 \\ 0 & 48 & 49 \end{array}\right) \quad \Longrightarrow \quad \left\{ \begin{array}{rcl} 3x_1 + 3x_2 & = & 7, \\ 48x_2 & = & 49, \end{array} \right.$$

Звідси $x_1^* = \frac{21}{16}$, $x_2^* = \frac{49}{48}$ — псевдорозв'язок заданої системи.

Приклад 2.3.2. Переконатися, що задана СЛАР є несумісною і знайти її псевдорозв'язки:

$$\left\{ \begin{array}{rcl} x_1 + x_2 & = & 2, \\ 2x_1 + 2x_2 & = & 5. \end{array} \right.$$

Розв'язування. Маємо

$$A = \begin{pmatrix} 1 & 1 \\ 2 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 2 \\ 5 \end{pmatrix}, \quad A|b = \left(\begin{array}{cc|c} 1 & 1 & 2 \\ 2 & 2 & 5 \end{array}\right).$$

Легко переконатися, що

$$\left(\begin{array}{cc|c} 1 & 1 & 2 \\ 2 & 2 & 5 \end{array}\right) \sim \left(\begin{array}{cc|c} 1 & 1 & 2 \\ 0 & 0 & 1 \end{array}\right),$$

тобто $\operatorname{rang} A = 1$ і $\operatorname{rang} A|b = 2$, тобто дана система несумісна. Знайдемо її псевдорозв'язки. Маємо

$$A^\top = \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}, \quad A^\top A = \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 2 & 2 \end{pmatrix} = \begin{pmatrix} 5 & 5 \\ 5 & 5 \end{pmatrix},$$

$$A^\top b = \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} 2 \\ 5 \end{pmatrix} = \begin{pmatrix} 12 \\ 12 \end{pmatrix}.$$

Отже, маємо систему для знаходження псевдорозв'язків (див. (2.3.3))

$$\begin{pmatrix} 5 & 5 \\ 5 & 5 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 12 \\ 12 \end{pmatrix} \Longrightarrow \left\{ \begin{array}{rcl} 5x_1 + 5x_2 & = & 12, \\ 5x_1 + 5x_2 & = & 12 \end{array} \right. .$$

Звідси отримуємо, що псевдорозв'язками даної системи є всі вектори $\begin{pmatrix} x_1^* \\ x_2^* \end{pmatrix}$, для компонент яких виконується рівність

$$x_1^* + x_2^* = 2,4.$$

Вправи для самостійної роботи

Переконатися, що задана СЛАР є несумісною і знайти її псевдорозв'язки:

1) $\left\{ \begin{array}{rcl} x_1 + 2x_2 & = & 2 \\ 2x_1 + 4x_2 & = & 5 \end{array} \right.$; 2) $\left\{ \begin{array}{rcl} 3x_1 + x_2 & = & 2 \\ x_1 + 4x_2 & = & 5 \\ 4x_1 + 5x_2 & = & 5 \end{array} \right.$; 3) $\left\{ \begin{array}{rcl} 4x_1 + x_2 & = & 2 \\ x_1 + x_2 & = & 3 \\ 5x_1 + 2x_2 & = & 6 \end{array} \right.$;

4) $\left\{ \begin{array}{rcl} x_1 + 2x_2 + x_3 & = & 3 \\ 2x_1 + 4x_2 + 2x_3 & = & 5 \end{array} \right.$.

2.4. Знаходження власних значень і власних векторів матриць

Нагадаємо деякі факти з ***теорії матриць***. Нехай $A \in \mathbb{C}^{n\times n}$ — яка-небудь квадратна матриця розміру n, складена з комплексних чисел. Число λ називають *власним значенням* матриці A, якщо існує *ненульовий вектор* $v \in \mathbb{C}^n$ такий, що виконується рівність

$$Av = \lambda v. \tag{2.4.1}$$

Згаданий вектор v називається *власним вектором*, який відповідає власному значенню λ. Відшукання власних значень і власних векторів матриці A зводиться (як випливає з (2.4.1)) до знаходження таких значень λ, за яких система лінійних алгебраїчних рівнянь

$$\big(A - \lambda I\big)v = 0 \tag{2.4.2}$$

має ненульові розв'язки (тут і далі I — одинична матриця, 0 — нульовий вектор). Це означає, що власні значення матриці A є розв'язками рівняння

$$\det\big(A - \lambda I\big) = 0,$$

тобто

$$\begin{vmatrix} a_{11} - \lambda & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} - \lambda \end{vmatrix} = 0. \tag{2.4.3}$$

Рівняння (2.4.3) можна записати у вигляді

$$\lambda^n + a_1\lambda^{n-1} + \cdots + a_n = 0,$$

яке згідно з основною теоремою алгебри можна переписати так:

$$(\lambda - \lambda_1)^{k_1} \cdots (\lambda - \lambda_m)^{k_m} = 0, \tag{2.4.4}$$

де $1 \leqslant m \leqslant n$, $\lambda_1, \ldots, \lambda_m$ — різні (загалом, комплексні) числа, $k_1 \geqslant 1, \ldots, k_m \geqslant 1$ — натуральні, $k_1 + \cdots + k_m = n$. Числа $\lambda_1, \ldots, \lambda_m$ є власними значеннями матриці A і тільки вони. Значення $k_1, \ldots, k_m$ називають *алгебраїчними кратностями* відповідних власних значень.

Для кожного $j \in \{1, \ldots, m\}$ власному значенню λ_j відповідає власний підпростір $B(\lambda_j)$, який складається з власних векторів, відповідних цьому власному значенню, та нульового вектора. Іншими словами, власний підпростір $B(\lambda_j)$ — це лінійний підпростір простору $\mathbb{C}^n$, складений з розв'язків лінійної однорідної системи рівнянь

$$\big(A - \lambda_j I\big)v = 0. \tag{2.4.5}$$

Розмірність цього власного підпростору $B(\lambda_j)$ позначимо через l_j. Отже, якщо $v^1, \ldots, v^{l_j}$ — яка-небудь система лінійно незалежних власних векторів, що відповідають власному значенню λ_j (лінійно незалежних розв'язків системи (2.4.5)), то будь-який власний вектор, що відповідає власному значенню λ_j, можна записати у вигляді

$$v = C_1 v^1 + \ldots + C_{l_j} v^{l_j}, \tag{2.4.6}$$

де $C_1, \ldots, C_{l_j}$ — деякі сталі. Тому векторну функцію v від змінних $C_1, \ldots, C_{l_j} \in \mathbb{C}$ (їх називають довільними сталими) вигляду (2.4.6), трактують як *загальний власний вектор*.

Значення l_j називається *геометричною кратністю* власного значення λ_j і, як відомо, $1 \leqslant l_j \leqslant k_j$. Фактично, число l_j — це максимальна кількість лінійно незалежних власних векторів, які відповідають власному значенню λ_j. Значення l_j можна визначити як *кількість вільних змінних* у системі (2.4.2).

Приклад 2.4.1. Знайти власні значення та власні вектори матриці

$$A = \begin{pmatrix} 2 & -1 & 1 \\ 1 & 2 & -1 \\ 1 & -1 & 2 \end{pmatrix}.$$

Розв'язування. Знайдемо власні значення матриці A, а це є корені рівняння

$$\det(A - \lambda I) = 0.$$

Оскільки

$$\det(A - \lambda I) \equiv \begin{vmatrix} 2-\lambda & -1 & 1 \\ 1 & 2-\lambda & -1 \\ 1 & -1 & 2-\lambda \end{vmatrix} = -(\lambda-1)(\lambda-2)(\lambda-3),$$

то числа $\lambda_1 = 1$, $\lambda_2 = 2$, $\lambda_3 = 3$ є власними значеннями матриці A, алгебраїчні кратності, а отже і геометричні (бо вони більші або рівні 1 і менші або рівні відповідних алгебраїчних кратностей) яких дорівнюють 1.

Запишемо системи лінійних алгебраїчних рівнянь для знаходження власних векторів

$$\begin{pmatrix} 2-\lambda_k & -1 & 1 \\ 1 & 2-\lambda_k & -1 \\ 1 & -1 & 2-\lambda_k \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \quad k = 1, 2, 3. \tag{2.4.7}$$

Системи (2.4.7) зручно розв'язувати, зводячи їхні основні матриці до східчастого вигляду.

Для $k = 1$ маємо

$$(A - I) \equiv \begin{pmatrix} 1 & -1 & 1 \\ 1 & 1 & -1 \\ 1 & -1 & 1 \end{pmatrix} \sim \begin{pmatrix} 1 & -1 & 1 \\ 0 & 2 & -2 \\ 0 & 0 & 0 \end{pmatrix} \sim \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{pmatrix}.$$

Отже, перша з систем (2.4.7) (при $k = 1$) рівносильна системі

$$v_1 = 0, \qquad v_2 - v_3 = 0.$$

Звідси, прийнявши $v_3 = C_1$, де C_1 — довільна стала, отримаємо

$$v = \begin{pmatrix} 0 \\ C_1 \\ C_1 \end{pmatrix} \equiv C_1 \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix},$$

— загальний власний вектор відповідний власному значенню λ_1.

У випадку $k = 2$ аналогічно з (2.4.7) одержуємо

$$(A - 2I) \equiv \begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ 1 & -1 & 0 \end{pmatrix} \sim \begin{pmatrix} 1 & 0 & -1 \\ 0 & -1 & 1 \\ 0 & -1 & 1 \end{pmatrix} \sim \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{pmatrix}.$$

Звідси $v_1 = v_3$, $v_2 = v_3$. Прийнявши $v_3 = C_2$, де C_2 — довільна стала, отримаємо $v_1 = C_2$, $v_2 = C_2$, а отже,

$$v = \begin{pmatrix} C_2 \\ C_2 \\ C_2 \end{pmatrix} \equiv C_2 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

— загальний власний вектор відповідний власному значенню λ_2.

Аналогічно зведемо основну матрицю системи (2.4.7) до східчастого вигляду у випадку $k = 3$:

$$(A - 3I) \equiv \begin{pmatrix} -1 & -1 & 1 \\ 1 & -1 & -1 \\ 1 & -1 & -1 \end{pmatrix} \sim \begin{pmatrix} 1 & -1 & -1 \\ 0 & -2 & 0 \\ 0 & 0 & 0 \end{pmatrix} \sim \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

Отже, маємо

$$v_1 = v_3, \quad v_2 = 0.$$

Міркуючи так само, як у попередньому випадку, доходимо висновку, що можна прийняти $v_3 = C_3$, де C_3 — довільна стала. Тоді $v_1 = C_3$, $v_2 = 0$. Отож, отримаємо

$$v = \begin{pmatrix} C_3 \\ 0 \\ C_3 \end{pmatrix} \equiv C_3 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

— загальний власний вектор відповідний власному значенню λ_3. □

Приклад 2.4.2. Знайти власні значення та власні вектори матриці

$$A = \begin{pmatrix} 2 & -1 & 2 \\ 1 & 0 & 2 \\ -2 & 1 & -1 \end{pmatrix}.$$

Розв'язування. Знайдемо власні значення матриці A. Оскільки

$$\det(A - \lambda I) = \begin{vmatrix} 2-\lambda & -1 & 2 \\ 1 & -\lambda & 2 \\ -2 & 1 & -1-\lambda \end{vmatrix} =$$
$$= (2-\lambda)(\lambda+1)\lambda + 4 + 2 - 4\lambda - 2(2-\lambda) - (1+\lambda) =$$

$$= \lambda(2 + 2\lambda - \lambda - \lambda^2) - 3\lambda + 1 = -\lambda^3 + \lambda^2 - 1 + 1 =$$

$$= -\lambda^2(\lambda - 1) - (\lambda - 1) = -(\lambda-1)(\lambda^2+1) =$$
$$= -(\lambda-1)(\lambda-i)(\lambda+i),$$

то власними значеннями матриці A є числа $\lambda_1 = 1$, $\lambda_2 = i$, $\lambda_3 = -i$.

Знайдемо загальні власні вектори для власних значень λ_1, λ_2, λ_3. Для цього розглянемо лінійні алгебричні системи

$$(A - \lambda_k I)v = 0 \tag{2.4.8}$$

для $k = 1, 2, 3$.

Як і в попередньому прикладі, системи (2.4.8) будемо розв'язувати, зводячи їхні основні матриці до східчастого вигляду.

У випадку $k = 1$ одержуємо

$$(A - I) \equiv \begin{pmatrix} 1 & -1 & 2 \\ 1 & -1 & 2 \\ 2 & 1 & -2 \end{pmatrix} \sim \begin{pmatrix} 1 & -1 & 2 \\ 0 & 3 & -6 \\ 0 & 0 & 0 \end{pmatrix} \sim \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -2 \\ 0 & 0 & 0 \end{pmatrix}.$$

Отже, для знаходження власних векторів маємо систему рівнянь

$$v_1 = 0, \quad v_2 - 2v_3 = 0\,.$$

Приймемо $v_3 = C_1$, де C_1 — довільна стала. Тоді $v_2 = 2C_1$, $v_1 = 0$. Отож,

$$v = \begin{pmatrix} 0 \\ 2C_1 \\ C_1 \end{pmatrix} \equiv C_1 \begin{pmatrix} 0 \\ 2 \\ 1 \end{pmatrix}$$

— загальний власний вектор для власного значення λ_1.

Аналогічно розглянемо випадок $k = 2$:

$$(A - iI) \equiv \begin{pmatrix} 2-i & -1 & 2 \\ 1 & -i & 2 \\ -2 & 1 & -1-i \end{pmatrix} \sim \begin{pmatrix} 1 & -i & 2 \\ 5 & -2-i & 4+2i \\ 0 & 1-2i & 3-i \end{pmatrix} \sim$$

$$\sim \begin{pmatrix} 1 & -i & 2 \\ 0 & -2+4i & -6+2i \\ 0 & 1-2i & 3-i \end{pmatrix} \sim \begin{pmatrix} 1 & -i & 2 \\ 0 & 1-2i & 3-i \\ 0 & 0 & 0 \end{pmatrix} \sim$$

$$\sim \begin{pmatrix} 1 & -i & 2 \\ 0 & 5 & 5+5i \\ 0 & 0 & 0 \end{pmatrix} \sim \begin{pmatrix} 1 & -i & 2 \\ 0 & 1 & 1+i \\ 0 & 0 & 0 \end{pmatrix} \sim \begin{pmatrix} 1 & 0 & 1+i \\ 0 & 1 & 1+i \\ 0 & 0 & 0 \end{pmatrix}.$$

Тут ми зробили такі елементарні перетворення матриці $(A - iI)$:

1) переставили місцями перший і другий рядки;
2) помножили другий рядок на $(2 + i)$;
3) до третього рядка додали перший, помножений на 2;
4) до другого рядка додали перший, помножений на (-5);
5) другий рядок поділили на (-2);
6) від третього рядка відняли другий;
7) домножили другий рядок на $(1 + 2i)$;
8) поділили другий рядок на 5;
9) до першого рядка додали другий, помножений на i.

Отже, друга з систем (2.4.8) $(k = 2)$ еквівалентна системі

$$v_1 + (1 + i)v_3 = 0, \quad v_2 + (1 + i)v_3 = 0.$$

Приймемо $v_3 = C_2$, де C_2 — довільне. Отож,

$$v^2 = \begin{pmatrix} -(1+i)\,C_2 \\ -(1+i)\,C_2 \\ C_2 \end{pmatrix} = C_2 \begin{pmatrix} -1-i \\ -1-i \\ 1 \end{pmatrix}$$

— загальний власний вектор для власного значення λ_2.

Для знаходження загального власного вектора для власного значення $\lambda_3 = -i$, зауважимо таке. Оскільки матриця A дійсна, то комплексно спряженою до матриці $(A - \lambda I)$ буде матриця $(A - \overline{\lambda} I)$. Отож, якщо v^2 — загальний власний вектор відповідний власному значенню $\lambda_2 = i$, тобто

$$(A - iI)v^2 = 0, \tag{2.4.9}$$

то

$$(A + iI)\overline{v^2} = 0,$$

а отже,

$$v^3 = C_3 \begin{pmatrix} -1+i \\ -1+i \\ 1 \end{pmatrix}, \quad C_3 - \text{ довільна стала},$$

— загальний власний вектор відповідний власному значенню $\lambda_3 = -i$. □

Приклад 2.4.3. Знайти власні значення та власні вектори матриці

$$A := \begin{pmatrix} -2 & 1 & -2 \\ 1 & -2 & 2 \\ 3 & -3 & 5 \end{pmatrix}.$$

Розв'язування. Знайдемо власні значення матриці A. Для цього зауважимо, що

$$\det(A - \lambda I) = \begin{vmatrix} -2-\lambda & 1 & -2 \\ 1 & -2-\lambda & 2 \\ 3 & -3 & 5-\lambda \end{vmatrix} = -(\lambda+1)^2(\lambda-3).$$

Отже, власними значеннями матриці A є числа $\lambda_1 = 3$ (алгебраїчна кратність $k_1 = 1$), $\lambda_2 = -1$ (алгебраїчна кратність $k_2 = 2$).

Знайдемо загальні власні вектори для власних значень λ_1, λ_2, тобто загальні розв'язки систем

$$(A - \lambda_k I)v = 0, \qquad k = 1, 2. \tag{2.4.10}$$

Розглянемо випадок $k = 1$. Маємо

$$(A - 3I) \equiv \begin{pmatrix} -5 & 1 & -2 \\ 1 & -5 & 2 \\ 3 & -3 & 2 \end{pmatrix} \sim \begin{pmatrix} 1 & -5 & 2 \\ 0 & -24 & 8 \\ 0 & 12 & -4 \end{pmatrix} \sim$$

$$\sim \begin{pmatrix} 1 & -5 & 2 \\ 0 & 3 & -1 \\ 0 & 0 & 0 \end{pmatrix} \sim \begin{pmatrix} 1 & 1 & 0 \\ 0 & 3 & -1 \\ 0 & 0 & 0 \end{pmatrix}.$$

Отож, система (2.4.10) при $k = 1$ еквівалентна системі

$$\begin{cases} v_1 = -v_2, \\ v_3 = 3v_2. \end{cases}$$

Оскільки тут є одна вільна змінна, то геометрична кратність власного значення λ_1 дорівнює 1. Тому приймемо $v_2 = C_1$, де C_1 — довільна стала. Отже, $v_1 = -C_1$, $v_3 = 3C_1$ і загальний власний вектор (для λ_1) такий:

$$v^1 = \begin{pmatrix} -C_1 \\ C_1 \\ 3C_1 \end{pmatrix} \equiv \begin{pmatrix} -C_1 \\ C_1 \\ 3C_1 \end{pmatrix}.$$

У випадку $k = 2$ з (2.4.10) отримаємо

$$(A + I) \equiv \begin{pmatrix} -1 & 1 & -2 \\ 1 & -1 & 2 \\ 3 & -3 & 6 \end{pmatrix} \sim \begin{pmatrix} 1 & -1 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix},$$

тобто друга з систем (2.4.10) еквівалентна системі

$$v_1 = v_2 - 2v_3.$$

Звідси випливає, що задана система має дві вільні змінні, тобто геометрична кратність власного значення λ_2 дорівнює 2. Тому можемо прийняти $v_2 = C_2$, $v_3 = C_3$, де C_2, C_3 — довільні сталі. Тоді $v_1 = C_2 - 2C_3$. Отож,

$$v^2 = \begin{pmatrix} C_2 - 2C_3 \\ C_2 \\ C_3 \end{pmatrix} \equiv C_2 \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} + C_3 \begin{pmatrix} -2 \\ 0 \\ 1 \end{pmatrix}$$

— загальний власний вектор для власного значення λ_2. □

Приклад 2.4.4. Знайти власні значення та власні вектори матриці

$$A := \begin{pmatrix} 2 & 1 & 0 \\ 0 & 2 & 4 \\ 1 & 0 & -1 \end{pmatrix}.$$

Розв'язування. Знайдемо власні значення матриці A. Маємо

$$\det(A - \lambda I) = \begin{vmatrix} 2-\lambda & 1 & 0 \\ 0 & 2-\lambda & 4 \\ 1 & 0 & -1-\lambda \end{vmatrix} =$$

$$= -(2-\lambda)^2(1+\lambda) + 4 = -(4 - 4\lambda + \lambda^2)(1+\lambda) + 4 =$$

$$= -4 + 4\lambda - \lambda^2 - 4\lambda + 4\lambda^2 - \lambda^3 + 4 = -\lambda^3 + 3\lambda^2 = -\lambda^2(\lambda - 3).$$

Отже, власними значеннями матриці A є числа $\lambda_1 = 3$ (алгебрична кратність $k_1 = 1$), $\lambda_2 = 0$ (алгебрична кратність $k_2 = 2$).

Знайдемо загальні власні вектори для власних значень λ_1 і λ_2, тобто загальні розв'язки лінійних алгебричних систем

$$(A - \lambda_k I)v = 0, \qquad k = 1, 2. \tag{2.4.11}$$

Спочатку розглянемо випадок $k = 1$. Маємо

$$(A - 3I) \equiv \begin{pmatrix} -1 & 1 & -4 \\ 0 & -1 & 4 \\ 1 & 0 & -4 \end{pmatrix} \sim \begin{pmatrix} 1 & -1 & 0 \\ 0 & 1 & -4 \\ 0 & 1 & -4 \end{pmatrix} \sim$$

$$\sim \begin{pmatrix} 1 & -1 & 0 \\ 0 & 1 & -4 \\ 0 & 0 & 0 \end{pmatrix} \sim \begin{pmatrix} 1 & 0 & -4 \\ 0 & 1 & -4 \\ 0 & 0 & 0 \end{pmatrix},$$

звідки $l_1 = 1$ і

$$\begin{cases} v_1 = 4v_3, \\ v_2 = 4v_3. \end{cases}$$

Приймемо $v_3 = C_1$. Тоді $v_1 = 4C_1$, $v_2 = 4C_1$. Отже,

$$v^1 = \begin{pmatrix} 4C_1 \\ 4C_1 \\ C_1 \end{pmatrix} \equiv C_1 \begin{pmatrix} 4 \\ 4 \\ 1 \end{pmatrix}$$

— загальний власний вектор відповідний власному значенню λ_1.

Розглянемо випадок $k = 2$. Зважаючи, що

$$(A - 0I) \equiv \begin{pmatrix} 2 & 1 & 0 \\ 0 & 2 & 4 \\ 1 & 0 & -1 \end{pmatrix} \sim \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & 2 \\ 0 & 0 & 0 \end{pmatrix},$$

приходимо висновку, що друга з систем (2.4.11) ($k = 2$) еквівалентна системі

$$\begin{cases} v_1 = v_3, \\ v_2 = -2v_3. \end{cases} \tag{2.4.12}$$

В цій системі одна вільна змінна, наприклад, v_3. Це означає, що геометрична кратність l_2 власного значення λ_2 дорівнює 1 ($l_2 = 1$). Приймемо $v_3 = {}_2$, де C_2 — довільна стала. Тоді

$$v^1 = \begin{pmatrix} C_2 \\ -2C_2 \\ C_2 \end{pmatrix} \equiv C_2 \begin{pmatrix} 1 \\ -2 \\ 1 \end{pmatrix}$$

— загальний власний вектор для власного значення λ_2. □

Приклад 2.4.5. Знайти власні значення та власні вектори матриці

$$A := \begin{pmatrix} 2 & -1 & -1 \\ 2 & -1 & -2 \\ -1 & 1 & 2 \end{pmatrix}.$$

Розв'язування. Знайдемо власні значення матриці A. Оскільки

$$\det(A - \lambda I) = \begin{vmatrix} 2-\lambda & -1 & -1 \\ 2 & -1-\lambda & -2 \\ -1 & 1 & 2-\lambda \end{vmatrix} =$$

$$= -\lambda^3 + 3\lambda^2 - 3\lambda + 1 = -(\lambda - 1)^3,$$

то характеристичне рівняння

$$\det(A - \lambda I) = 0$$

має один корінь $\lambda_1 = 1$ кратності 3. Отже, число $\lambda_1 = 1$ є власним значенням матриці A алгебричної кратності $k_1 = 3$.

Знайдемо загальний власний вектор, який відповідає власному значенню λ_1 із системи

$$(A - \lambda_1 I)v = 0. \tag{2.4.13}$$

Маємо

$$(A - I) \equiv \begin{pmatrix} 1 & -1 & -1 \\ 2 & -2 & -2 \\ -1 & 1 & 1 \end{pmatrix} \sim \begin{pmatrix} 1 & -1 & -1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix},$$

тобто система (2.4.13) еквівалентна системі

$$v_1 = v_2 + v_3$$

.

Вважаючи v_2 і v_3 вільними змінними, отримаємо

$$v^1 = \begin{pmatrix} v_2 + v_3 \\ v_2 \\ v_3 \end{pmatrix} \qquad \text{— з. в. в. для в. з. } \lambda_1. \tag{2.4.14}$$

З. в. в. визначається двома параметрами, а це означає, що геометрична кратність в. з. λ_1 дорівнює $l_1 = 2$. Загальний власний вектор матиме вигляд

$$v^1 = \begin{pmatrix} C_1 + C_2 \\ C_1 \\ C_2 \end{pmatrix} \equiv C_1 \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} + C_2 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \qquad \text{— з. в. в. для в. з. } \lambda_1. \tag{2.4.15}$$

□

Приклад 2.4.6. Знайти власні значення та власні вектори матриці

$$A := \begin{pmatrix} 4 & -1 & 0 \\ 3 & 1 & -1 \\ 1 & 0 & 1 \end{pmatrix}.$$

Розв'язування. Знайдемо власні значення матриці A. Маємо

$$\det(A - \lambda I) = \begin{vmatrix} 4-\lambda & -1 & 0 \\ 3 & 1-\lambda & -1 \\ 1 & 0 & 1-\lambda \end{vmatrix} =$$
$$= (4-\lambda)(1-\lambda)^2 + 1 + 3(1-\lambda) =$$
$$= -\lambda^3 + 6\lambda^2 - 12\lambda + 8 = -(\lambda - 2)^3.$$

Отже, число $\lambda_1 = 2$ є власним значенням матриці A алгебричної кратності $k_1 = 3$. Знайдемо загальний власний вектор для власного значення $\lambda_1 = 2$. Оскільки

$$(A - 2I) \equiv \begin{pmatrix} 2 & -1 & 0 \\ 3 & -1 & -1 \\ 1 & 0 & -1 \end{pmatrix} \sim \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -2 \\ 0 & 1 & -2 \end{pmatrix} \sim \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -2 \\ 0 & 0 & 0 \end{pmatrix},$$

то система

$$(A - \lambda_1 I)v = 0$$

еквівалентна системі

$$v_1 = v_3, \qquad v_2 = 2v_3.$$

Оскільки в цій системі є одна вільна змінна, то геометрична кратність в. з. λ_1 дорівнює $l_1 = 1$. Загальний власний вектор маємо у вигляді

$$v^1 = \begin{pmatrix} C_1 \\ 2C_1 \\ C_1 \end{pmatrix} \equiv C_1 \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix},$$

де C_1, C_2, C_3 — довільні сталі. □

Вправи для самостійної роботи

Знайти власні значення та власні вектори матриці

1) $A = \begin{pmatrix} 1 & -1 & 1 \\ 1 & 1 & -1 \\ 2 & -1 & 0 \end{pmatrix}$ $(\lambda_1 = 1; \lambda_2 = 2; \lambda_3 = -1)$;

2) A$=\begin{pmatrix} 1 & -1 & -1 \\ 1 & 1 & 0 \\ 3 & 0 & 1 \end{pmatrix}$ $(\lambda_1 = 1; \lambda_2 = 3)$;

3) A$=\begin{pmatrix} 1 & -1 & -1 \\ 1 & 1 & -1 \\ 0 & -1 & 2 \end{pmatrix}$ $(\lambda_1 = 1; \lambda_2 = 2)$;

4) A$=\begin{pmatrix} 4 & -1 & 0 \\ 3 & 1 & -1 \\ 1 & 0 & 1 \end{pmatrix}$ $(\lambda_1 = 2)$;

5) A$=\begin{pmatrix} 2 & -1 & -1 \\ 2 & -1 & -2 \\ -1 & 1 & 2 \end{pmatrix}$ $(\lambda_1 = 1)$.

2.5. Лабораторний практикум з розв'язування систем лінійних алгебраїчних рівнянь

У цьому підрозділі розглянемо деякі аспекти описаних в розділі 2 чисельних методів розв'язування СЛАР, які пов'язані із застосування цих методів на практиці. Зокрема розглянемо підходи до простої програмної реалізації згаданих методів, а також продемонструємо використання розроблених програм та деяких Python-бібліотек для проведення обчислювальних експериментів для дослідження чисельних розв'язків СЛАР.

Наведемо для зручності подальшої роботи визначення основних понять, які стосуються СЛАР, а також розрахункові формули чисельних методів. Причому формули подаватимемо у вигляді, орієнтованому на відповідну програмну реалізацію.

Отож, нехай маємо СЛАР

$$Ax = b \tag{2.5.1}$$

з добре обумовленою квадратною матрицею і вектором вільних членів

$$A := \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \in \mathbb{R}^{n\times n}, \quad b := \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \in \mathbb{R}^n,\ n \in \mathbb{N}, \tag{2.5.2}$$

$$x := \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \equiv (x_1, x_2 \dots, x_n)^\top \in \mathbb{R}^n \text{ – вектор невідомих величин.}$$

Нагадаємо означення норм у просторах матриць та векторів, які будуть потрібними далі при використанні ітераційних методів:

$$\|A\|_1 := \max_{1\le j\le n} \sum_{i=1}^{n} |a_{ij}|, \quad \|A\|_\infty := \max_{1\le i\le n} \sum_{j=1}^{n} |a_{ij}|. \tag{2.5.3}$$

$$\|b\|_2 := \sqrt{\sum_{i=1}^{n} |b_i^2|}, \quad \|b\|_\infty := \max_{1\le i\le n} |b_i|. \tag{2.5.4}$$

Ноутбуки, які будуть потрібні нам для чисельного розв'язування СЛАР, можна завантажити з репозиторію. Кожен ноутбук крім відповідного програмного коду також містить основні визначення та використані розрахункові формули. Крім того, там наведено деякі поясненя щодо використання програмного коду. Зокрема, стосовно налаштування середовища для виконання коду інтерпретатором. Оскільки чисельні методи реалізовано з використанням бібліотеки *NumPy*, то в даному випадку підготовка середовища зводиться до виконання такої комірки ноутбука:

[1]:
```
import numpy as np
```

Перед виконанням обчислень необхідно виконати комірки, в яких визначено функції, які реалізують конкретний чисельний метод і задають дані конкретних СЛАР, а також деякі допоміжні функції.

Допоміжні функції

[2]:
```
def norm_1(a):
    """обчислення норми_1 матриці a"""
    m=0
    for j in range(a.shape[1]):
        s=0
        for i in range(a.shape[0]):
```

```
            s+=abs(a[i][j])
        if s>m:
            m=s
    return m

def norm_2(a):
    """обчислення норми_2 матриці a"""
    m=0
    for i in range(a.shape[0]):
        s=0
        for j in range(a.shape[1]):
            s+=abs(a[i][j])
        if s>m:
            m=s
    return m


def norm_3(a):
    """обчислення евклідової норми вектора a"""
    return np.sqrt(np.sum(a**2))
```

Зазначимо, що усі розроблені ноутбуки є незалежними один від одного, хоч і можуть містити програмний код, який виконує те саме завдання для різних методів чи задач. В матеріалах лабораторного практикуму будемо уникати таких повторень.

2.5.1 Застосування методу Гаусса

Метод Гаусса полягає в покроковому зведенні системи (2.5.1) до трикутного вигляду (прямий хід):

$$A := \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ 0 & a_{22}^{(1)} & \dots & a_{2n}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{nn}^{(n-1)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(1)} \\ \vdots \\ b_n^{(n-1)} \end{pmatrix}, \tag{2.5.5}$$

після чого елементи розв'язку обчислюють зворотною підстановкою (обернений хід).

Тут верхній індекс елементів матриці та вектора вказує номер кроку, на якому вони були обчислені.

Після виконання $k-$го кроку маємо $a_{i,j}^{(k)} = 0$ при $i \geq k,\ j < k$.

На наступному $(k+1)-$му кроці елементи матриці та вектора правої частини перераховують за формулами

$$a_{i,j}^{(k+1)} = a_{i,j}^{(k)} - m_{i,k}\, a_{k,j}^{(k)},\ i,j = k+1,\dots,n, \tag{2.5.6}$$

$$b_i^{(k+1)} = b_i^{(k)} - m_{i,k}\, b_k^{(k)},\ i = k+1,\dots,n, \tag{2.5.7}$$

де $m_{i,k} = \frac{a_{i,k}^{(k)}}{a_{k,k}^{(k)}},\ i = k+1,\dots,n.$

Вважаємо, що на кожному кроці $a_{k,k}^{(k)} \neq 0$, якщо ж ні, то переставляємо рядки матриці, щоб ця умова виконувалася.

Пояснення до використання програмного коду

- Підготувати обчислювальне середовище і потрібні функції :
  1. виконати комірку для підготовки середовища

2. виконати комірку, де визначені функції `Gaussian_elimination`, `backward_substitution` і `GEM_solver`

- Обчислити шуканий розв'язок заданої СЛАР :
  1. виконати комірки, де визначені функції `set_matrix` і `set_vector`
  2. виконати комірку з викликом функції `GEM_solver` з аргументами, які визначені в попередньому пункті. Результатом цієї функції є вектор шуканого розв'язку.

Програмна реалізація методу

`Gaussian_elimination` **- функція, яка реалізує прямий хід методу Гаусса**

[2]:
```
def Gaussian_elimination(a,b):
    """ зведення добре обумовленої квадратної матриці a до верхньої трикутної
        та перетворення вектора b вільних членів
        методом Гаусса
    """
    n=b.size
    for k in range(1,n):
        for i in range(k,n):
            m=a[i,k-1]/a[k-1,k-1]
            for j in range(k,n):
                a[i,j]-= m * a[k-1,j]
            b[i]-= m * b[k-1]
```

`backward_substitution`**- функція, яка реалізує зворотній хід методу Гаусса**

[3]:
```
def backward_substitution(a,b):
    """ Розв'язування СЛАР з трикутною матрицею a
        і вектором b вільних членів.
        Розв'язок буде збережено у векторі b
    """
    n=b.size
    b[n-1]/=a[n-1,n-1]
    for k in range(1,n):
        for j in range(n-k,n):
            b[n-k-1]-=a[n-k-1,j]*b[j]
        b[n-k-1]/=a[n-k-1,n-k-1]
```

`GEM_solver`**- функція, яка реалізує розв'язування СЛАР методом Гаусса**

[4]:
```
def GEM_solver(n, set_matrix, set_vector):
    """ Розв'язування СЛАР методом Гаусса
    """
    a=set_matrix(n)
    b=set_vector(n)
    Gaussian_elimination(a,b)
    backward_substitution(a,b)
    return b
```

`set_matrix` **і** `set_vector` **– функції для задання конкретних СЛАР:**

- `set_matrix` – задає і повертає як результат матрицю СЛАР
- `set_vector` – задає і повертає як результат вектор вільних членів

У наступному прикладі як коефіцієнти СЛАР, так і її вільні члени задані безпосередньо у визначенні відповідних структур даних – масивів бібліотеки $NumPy$. На практиці можна задавати такі дані різними способами. В одному з наступних прикладів вони будуть обчислені за формулами. При заданні СЛАР важливою є лише одна вимога – це мають бути масиви $NumPy$.

Зазначимо, що треба виконувати відповідну комірку з визначеннм зазначених функцій кожного разу, коли матрицю або вектор змінюють.

Обчислювальні експерименти
Продемонструємо на прикладах розв'язування СЛАР методом Гаусса.

Приклад 1. (приклад 2.1) Обчислити методом Гаусса розв'язок СЛАР

$$\begin{cases} 2x_1 + 2x_2 + 3x_3 &= 1, \\ x_1 + 3x_2 + 2x_3 &= -8, \\ 2x_1 + x_2 + 2x_3 &= 3. \end{cases}$$

Спочатку визначимо функції, які задаватимуть матрицю і вектор вільних членів СЛАР :

[5]:
```
def set_matrix(n):
    """ функція для задання матриці конкретної СЛАР (по рядках)"""
    return np.array([[2, 2, 3],[1,3,2],[2,1,2]],dtype=float )
```

[6]:
```
def set_vector(n):
    """ функція для задання вектора вільних членів конкретної СЛАР"""
    return np.array([1, -8, 3],dtype=float)
```

Знайдемо чисельний розв'язок СЛАР:

[7]:
```
n=3
x = GEM_solver(n, set_matrix, set_vector)
print(f'Чисельний розв\'язок СЛАР при n={n}\n  x={x}')
```

```
Чисельний розв'язок СЛАР при n=3
  x=[ 1. -5.  3.]
```

Метод Гаусса є представником так званих прямих методів. Відомо, що вони є чутливими до похибок, які виникають в процесі обчислень. Продемонструємо це на наступному прикладі.

Приклад 2. Нехай елементи матриці і вектор вільних членів СЛАР задані формулами $a_{i,j} = (i+j+1)^{-1}$, $i,j = \overline{0,n-1}$, $n \in \mathbb{N}$, та $b_i = \sum_{j=0}^{n-1}(i+j+1)^{-1}$, $i = \overline{0,n-1}$, $n \in \mathbb{N}$, відповідно. Обчислити методом Гаусса розв'язок СЛАР (2.5.1) при різних значеннях n.

Легко бачити, що при довільному n розв'язком такої системи є вектор $x = (1,1,\ldots,1)^\top$, складений з одиничок. Зазначимо, що так задану матрицю називають матрицею Гільберта. Відомо, що при зростанні розміру n задана у такий спосіб СЛАР стає погано обумовленою і похибки обчислень суттєво впливатимуть на її чисельний розв'язок. Переконаємося в цьому на практиці.

Спочатку визначимо функції, які задаватимуть таку СЛАР:

[8]:
```
def set_matrix(n):
    """ задання матриці системи Гільберта"""
    a = np.empty((n,n),dtype=float)
    for i in range(n):
        for j in range(n):
            a[i,j] = 1/(i+j+1)
    return a
```

[9]:
```
def set_vector(n):
    """ задання вектора вільних членів системи Гільберта"""
    b = np.zeros(n ,dtype=float)
    for i in range(n):
        for j in range(n):
            b[i] += 1/(i+j+1)
    return b
```

Для невеликих значень n можемо переглянути, який вигляд мають матриця і вектор вільних членів, згенеровані цими функціями:

[10]:
```
set_matrix(5)
```

[10]:
```
array([[1.        , 0.5       , 0.33333333, 0.25      , 0.2       ],
       [0.5       , 0.33333333, 0.25      , 0.2       , 0.16666667],
       [0.33333333, 0.25      , 0.2       , 0.16666667, 0.14285714],
       [0.25      , 0.2       , 0.16666667, 0.14285714, 0.125     ],
       [0.2       , 0.16666667, 0.14285714, 0.125     , 0.11111111]])
```

[11]:
```
set_vector(5)
```

[11]:
```
array([2.28333333, 1.45      , 1.09285714, 0.88452381, 0.74563492])
```

Знайдемо тепер чисельний розв'язок методом Гаусса при $n = 5$:

[12]:
```
n=5
x = GEM_solver(n, set_matrix, set_vector)
print(f'Чисельний розв\'язок СЛАР при n={n}\n  x={x}')
```

```
Чисельний розв'язок СЛАР при n=5
  x=[1. 1. 1. 1. 1.]
```

Як бачимо, він повністю співпадає з точним розв'язком цієї системи.

Знайдемо тепер чисельний розв'язок СЛАР більшого розміру:

[13]:
```
n=10
x = GEM_solver(n, set_matrix, set_vector)
print(f'Чисельний розв\'язок СЛАР при n={n}\n  x={x}')
```

```
Чисельний розв'язок СЛАР при n=10
  x=[1.         1.00000011 0.99999774 1.00002048 0.99990264 1.00026691
 0.99956309 1.0004214  0.99977914 1.0000485 ]
```

[14]:
```
n=12
x = GEM_solver(n, set_matrix, set_vector)
print(f'Чисельний розв\'язок СЛАР при n={n}\n  x={x}')
```

```
Чисельний розв'язок СЛАР при n=12
  x=[0.99999998 1.00000272 0.99991614 1.00112492 0.99185897 1.03538471
 0.90231481 1.17541948 0.79576776 1.14866257 0.93852547 1.01102248]
```

[15]:
```
n=15
x = GEM_solver(n, set_matrix, set_vector)
print(f'Чисельний розв\'язок СЛАР при n={n}\n  x={x}')
```

```
Чисельний розв'язок СЛАР при n=15
  x=[ 0.99999997  1.00000242  0.99999492  0.99864026  1.02566811  0.78193349
  2.06684093 -2.2790367   7.53239313 -7.35504757  7.38066706 -1.12904142
  0.42574875  1.73328423  0.81795234]
```

Як бачимо, вже при $n \geq 12$ чисельний розв'язок суттєво відрізняється від точного, тобто метод Гаусса стає непридатним для таких СЛАР. Далі покажемо, що за допомогою спеціальних підходів, зокрема ітераційними методами можна досягнути точніші розв'язки розглянутиз СЛАР.

2.5.2 Застосування методу прогонки для роз'язування СЛАР з тридіагональною матрицею

Нехай маємо СЛАР

$$Hx = g, \tag{2.5.8}$$

де $H \in \mathbb{R}^{(n+1)\times(n+1)}$ — задана квадратна тридіагональна матриця

$$H = \begin{pmatrix} c_0 & b_0 & 0 & & \dots & 0 \\ a_1 & c_1 & b_1 & 0 & \dots & \vdots \\ 0 & a_2 & c_2 & b_2 & \dots & \vdots \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ \vdots & & & a_{n-1} & c_{n-1} & b_{n-1} \\ 0 & \dots & \dots & 0 & a_n & c_n \end{pmatrix},$$

n — натуральне число, $g \in \mathbb{R}^{n+1}$ — заданий вектор вільних членів і $x \in \mathbb{R}^{n+1}$ — невідомий вектор.

Шляхом зведення цієї системи до трикутного вигляду згідно методу Гауса можна показати, що, компоненти вектора x можна послідовно обчислити (починаючи з кінця) за формулами

$$x_n = \beta_n, \quad x_i = \alpha_i \cdot x_{i+1} + \beta_i, \quad i = \overline{n-1, 0}, \tag{2.5.9}$$

де

$$\alpha_0 := -\frac{b_0}{c_0}, \quad \beta_0 := \frac{g_0}{c_0}, \quad \alpha_i := -\frac{b_i}{a_i \cdot \alpha_{i-1} + c_i}, \quad \beta_i := \frac{g_i - a_i \cdot \beta_{i-1}}{a_i \cdot \alpha_{i-1} + c_i}, \quad i = \overline{1, n}. \tag{2.5.10}$$

Вважаємо, що матриця H є такою, що $c_0 \neq 0$ і $a_i \cdot \alpha_{i-1} + c_i \neq 0,\ i = \overline{1, n}$.

Пояснення до використання програмного коду

- Підготувати обчислювальне середовище і потрібні функції :
  1. виконати комірку для підготовки середовища
  2. виконати комірку, де визначена функція `TDMA_solver`

- Обчислити шуканий розв'язок заданої СЛАР :
  1. виконати комірки, де визначені функції `set_matrix_diagonals` і `set_vector`
  2. виконати комірку з викликом функції `TDMA_solver` з аргументами, які визначені в попередньому пункті. Результатом цієї функції є вектор шуканого розв'язку.

Програмна реалізація методу

TDMA_solver - функція, яка реалізує метод прогонки для розв'язування СЛАР

[2]:
```
def TDMA_solver(n,set_matrix_diagonals,set_vector):
    """ метод прогонки для розв'язування СЛАР
        з 3-діагональною матрицею
        вектор c-головна діагональ
        вектори a і b - нижня і верхня діагоналі, паралельні головній
        вектор g - вільні члени
      """
    c,a,b = set_matrix_diagonals(n)
    g = set_vector(n)

    alpha = np.empty(n+1, dtype=float )
    beta  = np.empty(n+1, dtype=float )
```

```
    if c[0] !=0 :
        alpha[0] =-b[0]/c[0]
        beta [0] = g[0]/c[0]
    else:
        raise Exception('c[0]==0')

    for i in range(1,n+1):
        w=a[i]*alpha[i-1]+c[i]
        if w != 0 :
            alpha[i] =-b[i]/w
            beta[i]  = (g[i] - a[i]*beta[i-1])/w
        else:
            raise Exception('w==0')

    x = np.empty(n+1, dtype=float )
    x[n] = beta[n]
    for i in range(n-1,-1,-1):
        x[i] = alpha[i]*x[i+1] + beta[i]
    return x
```

`set_matrix_diagonals` і `set_vector` – функції для задання діагоналей матриці H та вектора g конкретних СЛАР:

- `set_matrix_diagonals` – задає і повертає як результат три діагоналі матриці H;
- `set_vector` – задає і повертає як результат вектор вільних членів g.

Обчислювальні експерименти
Продемонструємо на прикладах розв'язування СЛАР методом прогонки.

Приклад 1. (приклад 2.3) Обчислити методом прогонки розв'язок СЛАР

$$\begin{cases} x_0 + 3x_1 & = 4 \\ x_0 + 2x_1 - \ x_2 & = 2 \\ x_1 + 4x_2 + \ x_3 & = 6 \\ x_2 + 4x_3 & = 5. \end{cases}$$

Спочатку визначимо функції, які задаватимуть матрицю і вектор вільних членів СЛАР :

[3]:
```
def set_matrix_diagonals(n):
    """ функція задає 3 діагоналі матриці СЛАР """
    c=np.array([1, 2, 4, 4], dtype=float)
    a=np.array([0, 1, 1, 1], dtype=float)
    b=np.array([3,-1, 1, 0], dtype=float)
    return c,a,b
```

[4]:
```
def set_vector(n):
    """ функція задає вектор вільних членів СЛАР"""
    g = np.array([4, 2, 6, 5], dtype=float)
    return g
```

Тепер можемо викликати функцію `TDMA_solver` з відповідними аргументами:

[5]:
```
n=3
x = TDMA_solver(n, set_matrix_diagonals, set_vector)
print(f'Чисельний розв\'язок СЛАР при n={n}\n  x={x}')
```

```
Чисельний розв'язок СЛАР при n=3
  x=[1. 1. 1. 1.]
```

Приклад 2. (приклад 2.1.2.II) Обчислити методом прогонки розв'язок СЛАР

$$\begin{cases} 2x_0 + 5x_1 & = 7 \\ x_0 - 3x_1 + 2x_2 & = 0 \\ x_1 + 6x_2 - x_3 & = 6 \\ x_2 + 4x_3 & = 5. \end{cases}$$

Отримання чисельного розв'язку цієї СЛАР повністю повторює розв'язування попереднього прикладу:

[6]:
```
def set_matrix_diagonals(n):
    """ функція задає 3 діагоналі матриці СЛАР """
    c=np.array([2, -3, 6, 4], dtype=float)
    a=np.array([0, 1, 1, 1], dtype=float)
    b=np.array([5, 2, -1, 0], dtype=float)
    return c,a,b
```

[7]:
```
def set_vector(n):
    """ функція задає вектор вільних членів СЛАР"""
    g = np.array([7, 0, 6, 5], dtype=float)
    return g
```

[8]:
```
n=3
x = TDMA_solver(n,set_matrix_diagonals,set_vector)
print(f'Чисельний розв\'язок СЛАР при n={n}\n  x={x}')
```

```
Чисельний розв'язок СЛАР при n=3
  x=[1. 1. 1. 1.]
```

Зауважимо, використаний підхід до програмної реалізації методу прогонки дає змогу просто інтегрувати розроблений програмний код в інші ноутбуки. Прикладом цього є використання функції `TDMA_solver` в ноутбуці, де розглянуто чисельне розв'язування крайових задач методом скінчених різниць. Оскільки застосування цього методу до звичайних диференціальних рівнянь приводить до тридіагональної матриці, то особливістю програмного коду функції `set_matrix_diagonals` у цьому випадку є обчислення діагоналей за відповідними формулами.

2.5.3 Застосування методу простої ітерації

Нехай СЛАР має вигляд

$$x = Bx + d, \tag{2.5.11}$$

де $B \in \mathbb{R}^{n\times n}$, $d \in \mathbb{R}^n$ — задані квадратна матриця і вектор вільних членів відповідно, $x \in \mathbb{R}^n$ — невідомий вектор.

Вважаємо, що матриця B задовольняє умову

$$\|B\|_1 < 1 \quad \text{чи } \|B^\top\|_1 < 1. \tag{2.5.12}$$

Згідно методу простої ітерації чисельний розв'язок шукаємо шляхом побудови послідовності

$$x^0, x^1, \ldots, x^k, \ldots \tag{2.5.13}$$

за правилом:

$$x^{k+1} = Bx^k + d, \quad k \in \mathbb{N}_0 := \mathbb{N} \cup \{0\}, \tag{2.5.14}$$

де $x^0 \in \mathbb{R}^n$— початкове наближення, $x^k = (x_1^k, x_2^k, \ldots, x_n^k)^\top$ — наближення розв'язку на k-тій ітерації.

Відомо (**Теорема 2.2.3**), що виконання нерівності (2.5.12) є достатньою умовою для збіжності ітерацій до шуканого розв'язку СЛАР.

Пояснення до використання програмного коду

- Підготувати обчислювальне середовище і потрібні функції :
  1. виконати комірку для підготовки середовища
  2. виконати комірку, де визначені допоміжні функції `norm_1`,`norm_2` і `norm_3`
  3. виконати комірку, де визначені функції `simple_iteration` і `simple_iteration_solver`
- Обчислити шуканий розв'язок заданої СЛАР :
  1. виконати комірки, де визначені функції `set_matrix` і `set_vector`
  2. виконати комірку, де визначена функція `set_x0`
  3. виконати комірку із заданням точності `eps` і максимальної кількості ітерацій `max_iter`
  4. виконати комірку з викликом функції `simple_iteration_solver`

Програмна реалізація методу

`simple_iteration` **- функція, яка розв'язує СЛАР** (2.5.11) **методом простої ітерації**

[3]:
```
def simple_iteration(b,d,x0,eps,max_iter):
    """ послідовно обчислює наближення розв'язку СЛАР
      методом простої ітерації, поки евклідова норма
      різниці двох послідовних наближень не стане меншою заданої точності eps
      x0 -- початкове наближення """
    x_prev=x0.copy()
    k=1
    x_new=np.matmul(b,x0)+d
    while norm_3(x_new-x_prev) > eps and k < max_iter:
        k+=1
        x_prev=x_new
        x_new=np.matmul(b,x_prev)+d
    return k, x_new
```

`simple_iteration_solver` – **функція, яка при виконанні достатньої умови збіжності** (2.5.12) **організовує обчислення чисельного розв'язку СЛАР** (2.5.11) **методом простої ітерації.**

[4]:
```
def simple_iteration_solver(set_matrix, set_vector,set_x0, eps, max_iter):
    """ Розв'язування СЛАР методом простих ітерацій """
    b=set_matrix()
    k=0
    if norm_1(b)<1 or norm_2(b)<1 :
        d=set_vector()
        x0=set_x0()
        k, x=simple_iteration(b,d,x0,eps,max_iter)
        return k, x
    else:
        return k, b
```

Алгоритм роботи функції `simple_iteration_solver` є таким:

- На початку формується матриця СЛАР (2.5.11)
- Якщо виконується достатня умова збіжності (2.5.12), то буде задано вектор вільних членів СЛАР і початкове наближення розв'язку. Після цього шляхом ітерацій обчислюється чисельний розв'язок, який функція повертає як елемент кортежу. У цьому випадку перший елемент цього кортежу матиме значення кількості виконаних ітерацій.
- Якщо умова (2.5.12) не виконується, то першим елементом кортежу є число 0.

`set_matrix`, `set_vector` і `set_x0` – функції для задання конкретних СЛАР і початкового наближення :

- `set_matrix` – задає і повертає як результат матрицю СЛАР
- `set_vector` – задає і повертає як результат вектор вільних членів
- `set_x0` – задає і повертає як результат вектор початкового наближення

Треба виконувати відповідну комірку з визначеннм цих функцій кожного разу, коли матрицю або вектори змінюють.

Обчислювальні експерименти

Продемонструємо на прикладах застосування методу простої ітерації до розв'язування СЛАР.

Приклад 1. (приклад 2.4) Обчислити методом простої ітерації розв'язок СЛАР

$$\begin{cases} x_1 + 0.1x_2 + 0.2x_3 &= 1.8 \\ 0.2x_1 + x_2 - 0.2x_3 &= 1.6 \\ 0.1x_1 - 0.2x_2 + x_3 &= 2.7 \end{cases} \tag{2.5.15}$$

Легко переконатися, що вектор $x = (1, 2, 3)^\top$ є точним розв'язком цієї системи, збережемо це значення:

[5]:
```
x=np.array([[1], [2], [3]])
```

Задану СЛАР приведемо до вигляду (2.5.11), де матриця B є такою:

$$B = \begin{pmatrix} 0 & -0,1 & -0,2 \\ -0,2 & 0 & 0,2 \\ -0,1 & 0,2 & 0 \end{pmatrix}.$$

Підготуємо функцію, яка задаватиме цю матрицю:

[6]:
```
def set_matrix():
    """ функція для задання матриці конкретної СЛАР"""
    return np.array([[0, -0.1, -0.2],[-0.2,0,0.2],[-0.1,0.2,0]])
```

Можемо переконатися, що матриця B задовольняє нерівність (2.5.12):

[32]:
```
print(f"Норми матриці B: {norm_1(set_matrix())}, {norm_2(set_matrix())}")
```

```
Норми матриці B: 0.4, 0.4
```

Отож, ітераційний процес буде збіжний.

Визначимо функції, які повертатимуть вектор d і вектор початкового наближення x^0, беручи $x^0 = d$:

[7]:
```
def set_vector():
    """ функція для задання вектора вільних членів конкретної СЛАР"""
    return np.array([[1.8], [1.6], [2.7]])
```

[8]:
```
def set_x0():
    """ функція для задання вектора початкового наближення розв'язку"""
    return set_vector()
```

Задамо тепер значення параметрів допустимої похибки чисельного розв'язку eps

[9]:
```
eps=0.001
```

та максимального числа ітерацій max_iter

[10]:
```
max_iter=100
```

Тепер знаходимо чисельний розв'язок

[11]:
```
k, xk=simple_iteration_solver(set_matrix, set_vector, set_x0, eps, max_iter)

if k > 0 and k < max_iter:
    print(f"Чисельний розв'язок системи \n x={xk} \n обчислено за {k} ітерацій")
    print(f"похибка чисельного розв'язку {norm_3(xk-x)}.")
elif  k == max_iter :
    print(f"Чисельний розв'язок системи не обчислено за {k} ітерацій")
else:
    print("Матриця СЛАР  не задовольняє достатню умову збіжності ітерацій")
```

```
Чисельний розв'язок системи
 x=[[1.0001988 ]
 [1.99975656]
 [2.99979648]]
 обчислено за 7 ітерацій
похибка чисельного розв'язку 0.00037443939963607115.
```

Бачимо, що чисельний розв'язок x^7 близький до точного за евклідовою нормою. Поекспериментуємо з розв'язуванням, змінюючи значення параметра eps :

[30]:
```
eps=0.00001
k, xk=simple_iteration_solver(set_matrix, set_vector, set_x0, eps, max_iter)

if k > 0 and k < max_iter:
    print(f"Чисельний розв'язок системи \n x={xk} \n обчислено за {k} ітерацій")
    print(f"похибка чисельного розв'язку {norm_3(xk-x)}.")
elif  k == max_iter :
    print(f"Чисельний розв'язок системи не обчислено за {k} ітерацій")
else:
    print("Матриця СЛАР  не задовольняє достатню умову збіжності ітерацій")
```

```
Чисельний розв'язок системи
 x=[[0.99999847]
 [2.00000188]
 [3.00000159]]
 обчислено за 13 ітерацій
похибка чисельного розв'язку 3.063853949051739e-06.
```

[21]:
```
eps=0.000001
k, xk=simple_iteration_solver(set_matrix, set_vector, set_x0, eps, max_iter)

if k > 0 and k < max_iter:
    print(f"Чисельний розв'язок системи \n x={xk} \n обчислено за {k} ітерацій")
    print(f"похибка чисельного розв'язку {norm_3(xk-x)}.")
elif  k == max_iter :
    print(f"Чисельний розв'язок системи не обчислено за {k} ітерацій")
else:
    print("Матриця СЛАР не задовольняє достатню умову збіжності ітерацій")
```

```
Чисельний розв'язок системи
 x=[[1.00000009]
 [1.99999989]
 [2.99999991]]
 обчислено за 14 ітерацій
похибка чисельного розв'язку 0.0.
```

Розглянемо тепер чисельні розв'язки СЛАР при різних початкових наближеннях.

[33]:
```
def set_x0():
    """ функція для задання вектора початкового наближення розв'язку"""
    return np.array([[10 ], [10], [11]])
```

[34]:
```
eps=0.000001
k, xk=simple_iteration_solver(set_matrix, set_vector, set_x0, eps, max_iter)

if k > 0 and k < max_iter:
    print(f"Чисельний розв'язок системи \n x={xk} \n обчислено за {k} ітерацій")
    print(f"похибка чисельного розв'язку {norm_3(xk-x)}.")
elif  k == max_iter :
    print(f"Чисельний розв'язок системи не обчислено за {k} ітерацій")
else:
    print("Матриця СЛАР не задовольняє достатню умову збіжності ітерацій")
```

```
Чисельний розв'язок системи
 x=[[0.99999983]
 [2.00000021]
 [3.00000018]]
 обчислено за 15 ітерацій
похибка чисельного розв'язку 4.850158884211414e-07.
```

[35]:
```
def set_x0():
    """ функція для задання вектора початкового наближення розв'язку"""
    return np.array([[0 ], [1], [0]])
```

[36]:
```
eps=0.000001
k, xk=simple_iteration_solver(set_matrix, set_vector, set_x0, eps, max_iter)

if k > 0 and k < max_iter:
    print(f"Чисельний розв'язок системи \n x={xk} \n обчислено за {k} ітерацій")
    print(f"похибка чисельного розв'язку {norm_3(xk-x)}.")
elif  k == max_iter :
    print(f"Чисельний розв'язок системи не обчислено за {k} ітерацій")
else:
    print("Матриця СЛАР  не задовольняє достатню умову збіжності ітерацій")
```

```
Чисельний розв'язок системи
 x=[[1.00000021]
```

```
 [1.99999975]
 [2.99999979]]
 обчислено за 14 ітерацій
похибка чисельного розв'язку 2.2485513311112248e-07.
```

[39]:
```python
def set_x0():
    """ функція для задання вектора початкового наближення розв'язку"""
    return np.array([[-1 ], [-100], [11]])
```

[40]:
```python
eps=0.000001
k, xk=simple_iteration_solver(set_matrix, set_vector, set_x0, eps, max_iter)

if k > 0 and k < max_iter:
    print(f"Чисельний розв'язок системи \n x={xk} \n обчислено за {k} ітерацій")
    print(f"похибка чисельного розв'язку {norm_3(xk-x)}.")
elif  k == max_iter :
    print(f"Чисельний розв'язок системи не обчислено за {k} ітерацій")
else:
    print("Матриця СЛАР  не задовольняє достатню умову збіжності ітерацій")
```

```
Чисельний розв'язок системи
 x=[[1.00000019]
 [1.99999977]
 [2.9999998 ]]
 обчислено за 17 ітерацій
похибка чисельного розв'язку 1.95478385736245e-07.
```

Як бачимо, для даної СЛАР збіжність методу при різних початкових наближеннях практично не відрізняється.

2.5.4 Застосування методу Якобі

Нехай маємо СЛАР (2.5.1), для якої $\det A \neq 0$, а також $a_{ii} \neq 0,\ i \in \overline{1,n}$.

Позначимо

$$B := \begin{pmatrix} 0 & -\frac{a_{12}}{a_{11}} & \dots & -\frac{a_{1n}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & 0 & \dots & -\frac{a_{2n}}{a_{22}} \\ \vdots & \vdots & \ddots & \vdots \\ -\frac{a_{n1}}{a_{nn}} & -\frac{a_{n2}}{a_{nn}} & \dots & 0 \end{pmatrix}, \quad d = \begin{pmatrix} \frac{b_1}{a_{11}} \\ \frac{b_2}{a_{22}} \\ \vdots \\ \frac{b_n}{a_{nn}} \end{pmatrix}. \tag{2.5.16}$$

Метод *Якобі* полягає у заміні (2.5.1) еквівалентною СЛАР

$$x = Bx + d, \tag{2.5.17}$$

для якої будують ітераційний процес обчислення послідовності

$$x^0, x^1, \dots, x^k, \dots$$

наближень розв'язку системи за правилом:

$$x^{k+1} = B\,x^k + d, \quad k \in \mathbb{N}_0. \tag{2.5.18}$$

де $x^0 \in \mathbb{R}^n$— початкове наближення, $x^k = (x_1^k, x_2^k, \dots, x_n^k)^\top$ — наближення розв'язку на k-тій ітерації.

Якщо отримана матриця B задовольняє умову

$$\|B\|_1 < 1 \quad \text{чи } \|B^\top\|_1 < 1 \tag{2.5.19}$$

то ітераційний процес Якобі збігається.

Пояснення до використання програмного коду

- Підготувати обчислювальне середовище і потрібні функції :
  1. виконати комірку для підготовки середовища
  2. виконати комірку, де визначені допоміжні функції `norm_1`,`norm_2` і `norm_3`
  3. виконати комірку, де визначена функція `simple_iteration`
  4. виконати комірку, де визначені функції `Jacobi_modification` і `Jacobi_solver`
- Обчислити шуканий розв'язок заданої СЛАР :
  1. виконати комірки, де визначені функції `set_matrix` і `set_vector`
  2. виконати комірку, де визначена функція `set_x0`
  3. виконати комірку із заданням точності `eps` і максимальної кількості ітерацій `max_iter`
  4. Виконати комірку з викликом функції `Jacobi_solver`

Програмна реалізація методу

`Jacobi_solver` – **функція, яка організовує обчислення чисельного розв'язку СЛАР** (2.5.1) **методом Якобі з точністю** `eps` **при виконанні достатньої умови збіжності** (2.5.19).

[3]:
```
def Jacobi_solver(set_matrix, set_vector,set_x0, eps, max_iter):
    """ керує етапами чисельного розв'язування СЛАР  методом Якобі
      """
    b=set_matrix()
    d=set_vector()
    Jacobi_modification(b, d)
    k=0
    if norm_1(b)<1 or norm_2(b)<1 :
```

```
        x0=set_x0()
        k, x=simple_iteration(b,d,x0,eps,max_iter)
        return k, x
    else:
        return k, d
```

`Jacobi_modification` – **функція, яка перетворює матрицю і вектор вільних членів СЛАР** (2.5.1) **за формулою** (2.5.16)

[4]:
```
def Jacobi_modification(a, b):
    """ модифікація матриці a і вектора вільних членів b СЛАР згідно формули␣
↪(2.5.19)"""
    for i in range(a.shape[0]):
        b[i] /=a [i,i]
        a[i,:] /= -a[i,i]
        a[i,i] = 0
```

`simple_iteration` – **функція, яка розв'язує СЛАР** (2.5.17) **методом простої ітерації**

[5]:
```
def simple_iteration(b,d,x0,eps,max_iter):
    """ послідовно обчислює наближення розв'язку СЛАР
      методом простої ітерації, поки евклідова норма
      різниці двох послідовних наближень не стане меншою заданої точності eps
      x0 -- початкове наближення """
    x_prev=x0.copy()
    k=1
    x_new=np.matmul(b,x0)+d
    while norm_3(x_new-x_prev) > eps and k < max_iter:
        k+=1
        x_prev=x_new
        x_new=np.matmul(b,x_prev)+d
    return k, x_new
```

`set_matrix`, `set_vector` **і** `set_x0` – **функції для задання конкретних СЛАР та вектора початкового наближення у такий самий спосіб, як і у випадку простої ітерації.**

Загальне керування процесом чисельного розв'язування СЛАР відбувається у функції `Jacobi_solver`:

- На початку формується матриця і вектор вільних членів конкретної СЛАР (2.5.1), які далі модифікуються функцією `Jacobi_modification` за формулою (2.5.16).
- Якщо виконується достатня умова збіжності (2.5.19), то буде задано початкове наближення розв'язку. Після цього, враховуючи обмеження на кількість ітерацій, обчислюється чисельний розв'язок за допомогою функції `simple_iteration`, розглянутої у попередньому підрозділі. Якщо чисельний розв'язок знайдений, то функція `simple_iteration` повертає його як елемент кортежу. У цьому випадку перший елемент цього кортежу матиме значення кількості виконаних ітерацій.
- Якщо умова (2.5.19) не виконується, то першим елементом кортежу є число 0.

Обчислювальні експерименти

Продемонструємо на прикладах застосування ітераційного методу Якобі до розв'язування СЛАР.

Приклад 1. (приклад 2.4) Обчислити методом Якобі розв'язок СЛАР

$$\begin{cases} 10x_1 + \ \ x_2 + \ \ 2x_3 & = 18 \\ \ \ x_1 + 5x_2 - \ \ \ x_3 & = 8 \\ \ \ x_1 - 2x_2 + 10x_3 & = 27 \end{cases} \tag{2.5.20}$$

Легко переконатися, що вектор $x = (1, 2, 3)^\top$ є точним розв'язком цієї системи, збережемо це значення для подальшої оцінки чисельного розв'язку:

[6]:
```
x=np.array([[1], [2], [3]])
```

Підготуємо функції, які задаватимуть матрицю СЛАР та вектори вільних членів d і початкового наближення x^0:

[7]:
```
def set_matrix():
    """ функція для задання матриці конкретної СЛАР"""
    matrix=np.array([[10, 1, 2],[1,5,-1],[1,-2,10]],dtype=float )
    return matrix
```

[8]:
```
def set_vector():
    """ функція для задання вектора вільних членів конкретної СЛАР"""
    vector=np.array([[18], [8], [27]],dtype=float)
    return vector
```

[9]:
```
def set_x0():
    """ функція для задання вектора початкового наближення розв'язку"""
    return np.array([[1], [2], [1]],dtype=float)
```

Оскільки ми не робили оцінки необхідної кількості ітерацій, то наперед обмежимо їх достатньо великим числом:

[10]:
```
max_iter=100
```

Знайдемо чисельний розв'язок СЛАР і його абсолютну похибку, попередньо задавши значення параметра eps:

[11]:
```
eps=0.001
k, xk = Jacobi_solver(set_matrix, set_vector, set_x0, eps, max_iter)

if k > 0 and k < max_iter:
    print(f"Чисельний розв'язок системи \n x={xk} \n обчислено за {k} ітерацій")
    print(f"похибка чисельного розв'язку {norm_3(xk-x)}.")
elif  k == max_iter :
    print(f"Чисельний розв'язок системи не обчислено за {k} ітерацій")
else:
    print("Матриця СЛАР не задовольняє достатню умову збіжності ітерацій")
```

```
Чисельний розв'язок системи
 x=[[1.00010128]
 [1.99987424]
 [2.99989456]]
 обчислено за 8 ітерацій
похибка чисельного розв'язку 0.00019284918874576268.
```

Можемо досягти більшої точності чисельного розв'язку, модифікуючи значення параметра eps:

[12]:
```
eps=0.00001
k, xk = Jacobi_solver(set_matrix, set_vector, set_x0, eps, max_iter)

if k > 0 and k < max_iter:
    print(f"Чисельний розв'язок системи \n x={xk} \n обчислено за {k} ітерацій")
```

```
    print(f"похибка чисельного розв'язку {norm_3(xk-x)}.")
elif  k == max_iter :
    print(f"Чисельний розв'язок системи не обчислено за {k} ітерацій")
else:
    print("Матриця СЛАР не задовольняє достатню умову збіжності ітерацій")
```

```
Чисельний розв'язок системи
 x=[[1.00000123]
 [1.99999848]
 [2.99999872]]
 обчислено за 12 ітерацій
похибка чисельного розв'язку 2.337751989001623e-06.
```

[13]:
```
eps=0.0000001
k, xk = Jacobi_solver(set_matrix, set_vector, set_x0, eps, max_iter)

if k > 0 and k < max_iter:
    print(f"Чисельний розв'язок системи \n x={xk} \n обчислено за {k} ітерацій")
    print(f"похибка чисельного розв'язку {norm_3(xk-x)}.")
elif  k == max_iter :
    print(f"Чисельний розв'язок системи не обчислено за {k} ітерацій")
else:
    print("Матриця СЛАР не задовольняє достатню умову збіжності ітерацій")
```

```
Чисельний розв'язок системи
 x=[[1.00000001]
 [1.99999998]
 [2.99999998]]
 обчислено за 16 ітерацій
похибка чисельного розв'язку 2.8353437253144928e-08.
```

2.5.5 Застосування методу Зейделя

Нехай маємо СЛАР (2.5.1) і, як і раніше, вважаємо, що $\det A \neq 0$ і $a_{ii} \neq 0,\ i \in \overline{1,n}$.

Метод *Зейделя* полягає в обчисленні послідовності

$$x^0, x^1, \ldots, x^k, \ldots$$

наближень розв'язку системи за правилом:

$$x_i^{k+1} = -\sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^{k+1} - \sum_{j=i+1}^{n} \frac{a_{ij}}{a_{ii}} x_j^k + d_i, \quad i = \overline{1,n}, \quad k \in \mathbb{N}_0. \tag{2.5.21}$$

де $x^k = (x_1^k, x_2^k, \ldots, x_n^k)^\top$ – k-та ітерація вектора наближень, $x^0 \in \mathbb{R}^n$ – початкове наближення.

Пояснення до використання програмного коду

- Підготувати обчислювальне середовище і потрібні функції :
  1. виконати комірку для підготовки середовища
  2. виконати комірку, де визначені допоміжні функції `norm_1`,`norm_2` і `norm_3`
  3. виконати комірки, де визначені функції `Seidel_solver`, `Jacobi_modification` і `Seidel_iteration`
- Обчислити шуканий розв'язок заданої СЛАР :
  1. виконати комірки, де визначені функції `set_matrix` і `set_vector`
  2. виконати комірку, де визначена функція `set_x0`
  3. виконати комірку із заданням точності `eps` і максимальної кількості ітерацій `max_iter`
  4. виконати комірку з викликом функції `Seidel_solver` з відповідними значеннями її аргументів.

Програмна реалізація методу

`Seidel_solver` – **функція, яка організовує обчислення наближеного розв'язку СЛАР методом Зейделя з точністю** `eps`

[2]:
```
def Seidel_solver(set_matrix, set_vector,set_x0, eps, max_iter):
    """ керує етапами чисельного розв'язування СЛАР  методом Зейделя """
    b=set_matrix()
    d=set_vector()
    Jacobi_modification(b, d)
    k=0
    if norm_1(b)<1 or norm_2(b)<1 :
        x0=set_x0()
        k, x = Seidel_iteration(b,d,x0,eps,max_iter)
        return k, x
    else:
        return k, d
```

`Jacobi_modification` – **функція, яка перетворює матрицю і вектор вільних членів СЛАР** (2.5.1) **за формулою** (2.5.16)

[3]:
```
def Jacobi_modification(a, b):
    """ модифікація матриці і вектора вільних членів СЛАР """
    for i in range(a.shape[0]):
        b[i] /= a[i,i]
        a[i,:] /= -a[i,i]
        a[i,i]=0
```

`Seidel_iteration` – функція, яка реалізує метод Зейделя для розв'язування СЛАР

[4]:
```
def Seidel_iteration(b,d,x0,eps,iter_print):
    """ послідовно обчислює наближення розв'язку СЛАР
      за методом Зейделя, поки евклідова норма
      різниці двох послідовних наближень не стане меншою заданої точності␣
↪eps
      """
    n=b.shape[0]
    x_prev=x0.copy()

    k=1
    x_n = np.empty(n)
    x_n[0]=np.matmul(b[0,1:],x_prev[1:]).sum() +d[0]
    for i in range(1,n):
        x_n[i]= np.matmul(b[i,:i],x_n[:i]).sum() + np.matmul(b[i,i+1:
↪],x_prev[i+1:]).sum() +d[i]

    while norm_3(x_n - x_prev)  > eps and k < max_iter:
        x_prev=x_n.copy()

        k+=1
        x_n[0]=np.matmul(b[0,1:],x_prev[1:]).sum() +d[0]
        for i in range(1,n):
            x_n[i]= np.matmul(b[i,:i],x_n[:i]).sum() + np.matmul(b[i,i+1:
↪],x_prev[i+1:]).sum() +d[i]

    return k, x_n
```

`set_matrix`, `set_vector` і `set_x0` – як і в методі простої ітерації, це функції для задання конкретної СЛАР та початкового наближення її розв'язку

Обчислювальні експерименти

Продемонструємо на прикладах застосування ітераційного методу Зейделя до розв'язування СЛАР.

Приклад 1. (приклад 2.4) Обчислити методом Зейделя розв'язок СЛАР

$$\begin{cases} 10x_1 + \ \ x_2 + \ \ 2x_3 & = 18 \\ \ \ x_1 + 5x_2 - \ \ \ x_3 & = 8 \\ \ \ x_1 - 2x_2 + 10x_3 & = 27 \end{cases} \tag{2.5.22}$$

Легко переконатися, що вектор $x = (1, 2, 3)^\top$ є точним розв'язком цієї системи, збережемо це значення:

[6]:
```
x=np.array([1, 2, 3])
```

Підготуємо функції, які задаватимуть матрицю СЛАР та вектори вільних членів d і початкового наближення x^0:

[7]:
```
def set_matrix():
    """ функція для задання матриці СЛАР"""
    matrix=np.array([[10, 1, 2],[1,5,-1],[1,-2,10]],dtype=float )
    return matrix
```

[8]:
```
def set_vector():
    """ функція для задання вектора вільних членів СЛАР"""
    vector=np.array([[18], [8], [27]],dtype=float)
    return vector
```

[9]:
```
def set_x0():
    """ функція для задання вектора початкового наближення розв'язку"""
    return np.array([[1], [2], [1]],dtype=float)
```

Оскільки ми не робили оцінки необхідної кількості ітерацій, то наперед обмежимо їх достатньо великим числом:

[10]:
```
max_iter=100
```

Знайдемо чисельний розв'язок СЛАР, попередньо задавши значення параметра *eps*:

[11]:
```
eps=0.001
k, xk = Seidel_solver(set_matrix, set_vector, set_x0, eps, max_iter)

if k > 0 and k < max_iter:
    print(f"Чисельний розв'язок системи \n x={xk} \n обчислено за {k}␣
 ↪ітерацій")
    print(f"похибка чисельного розв'язку {norm_3(xk-x)}.")
elif  k == max_iter :
    print(f"Чисельний розв'язок системи не обчислено за {k} ітерацій")
else:
    print("Матриця СЛАР  не задовольняє достатню умову збіжності ітерацій")
```

```
Чисельний розв'язок системи
 x=[1.00008551 1.99994699 2.99998085]
 обчислено за 5 ітерацій
похибка чисельного розв'язку 0.00010241602653679628.
```

Можемо досягти більшої точності чисельного розв'язку, модифікуючи значення параметра *eps*:

[12]:
```
eps=0.00001
k, xk = Seidel_solver(set_matrix, set_vector, set_x0, eps, max_iter)

if k > 0 and k < max_iter:
    print(f"Чисельний розв'язок системи \n x={xk} \n обчислено за {k}␣
 ↪ітерацій")
    print(f"похибка чисельного розв'язку {norm_3(xk-x)}.")
elif  k == max_iter :
    print(f"Чисельний розв'язок системи не обчислено за {k} ітерацій")
else:
    print("Матриця СЛАР не задовольняє достатню умову збіжності ітерацій")
```

```
Чисельний розв'язок системи
 x=[1.00000097 1.9999994  2.99999978]
 обчислено за 7 ітерацій
похибка чисельного розв'язку 1.1670628701770979e-06.
```

[13]:
```
eps=0.0000001
k, xk = Seidel_solver(set_matrix, set_vector, set_x0, eps, max_iter)

if k > 0 and k < max_iter:
    print(f"Чисельний розв'язок системи \n x={xk} \n обчислено за {k}␣
 ↪ітерацій")
```

```
    print(f"похибка чисельного розв'язку {norm_3(xk-x)}.")
elif  k == max_iter :
    print(f"Чисельний розв'язок системи не обчислено за {k} ітерацій")
else:
    print("Матриця СЛАР  не задовольняє достатню умову збіжності ітерацій")
```

```
Чисельний розв'язок системи
 x=[1. 2. 3.]
 обчислено за 10 ітерацій
похибка чисельного розв'язку 1.4192169363730446e-09.
```

Якщо порівняти отримані чисельні розв'язки даної СЛАР з розв'язками методу Якобі, то можна побачити, що метод Зейделя досягає потрібну точність за меншу кількість ітерацій.

Приклад 2. (приклад 2 у підрозділі 2.5.1) Обчислити методом Зейделя чисельні розв'язки СЛАР з матрицею Гільберта різних розмірів `n`.

Відразу зазначимо, що достатння умова збіжності ітераційного процесцу (див. **Теорема 2.2.3**) для заданої матриці не виконується. Тому для виконання ітераційного процесу змінимо обмеження на норму матриці у функції `Seidel_solver`, а також для цієї функції додамо новий аргумент `n`, щоб було зручно виконувати обчислення з матрицями різних розмірів. Результат такої модифікації назвемо `Seidel_solver_Hilbert`:

[14]:
```
def Seidel_solver_Hilbert(n,set_matrix, set_vector,set_x0, eps, max_iter):
    """ Розв'язування СЛАР методом простих ітерацій """
    b=set_matrix(n)
    d=set_vector(n)
    Jacobi_modification(b, d)
    k=0
    if norm_1(b)<20 or norm_2(b)<20 :
        x0=set_x0(n)
        k, x = Seidel_iteration(b,d,x0,eps,max_iter)
        return k, x
    else:
        return k, d
```

Беручи за початкове наближення вектор правої частини, знайдемо чисельні розв'язки при різних розмірах системи:

[15]:
```
eps=0.0001
max_iter=1000
n=5
k, xk = Seidel_solver_Hilbert(n,set_matrix, set_vector, set_x0, eps,␣
 ↪max_iter)

if k > 0 and k < max_iter:
    print(f"Чисельний розв'язок системи \n x={xk} \n обчислено за {k}␣
 ↪ітерацій")
    #print(f"похибка чисельного розв'язку {norm_3(xk-x)}.")
elif  k == max_iter :
    print(f"Чисельний розв'язок системи не обчислено за {k} ітерацій")
else:
    print("Матриця СЛАР не задовольняє достатню умову збіжності ітерацій")
```

```
Чисельний розв'язок системи
 x=[1.00172981 0.98670063 1.0124375  1.02442054 0.97337122]
 обчислено за 602 ітерацій
```

[16]:
```
n=10
k, xk = Seidel_solver_Hilbert(n,set_matrix, set_vector, set_x0, eps,␣
↪max_iter)

if k > 0 and k < max_iter:
    print(f"Чисельний розв'язок системи \n x={xk} \n обчислено за {k}␣
↪ітерацій")
    #print(f"похибка чисельного розв'язку {norm_3(xk-x)}.")
elif  k == max_iter :
    print(f"Чисельний розв'язок системи не обчислено за {k} ітерацій")
else:
    print("Матриця СЛАР не задовольняє достатню умову збіжності ітерацій")
```

```
Чисельний розв'язок системи
 x=[0.99752956 1.02308501 0.96858383 0.97552268 1.00732935 1.0278794
 1.0300369  1.01645641 0.99168809 0.95978807]
 обчислено за 850 ітерацій
```

[17]:
```
n=12
k, xk = Seidel_solver_Hilbert(n,set_matrix, set_vector, set_x0, eps,␣
↪max_iter)

if k > 0 and k < max_iter:
    print(f"Чисельний розв'язок системи \n x={xk} \n обчислено за {k}␣
↪ітерацій")
    #print(f"похибка чисельного розв'язку {norm_3(xk-x)}.")
elif  k == max_iter :
    print(f"Чисельний розв'язок системи не обчислено за {k} ітерацій")
else:
    print("Матриця СЛАР не задовольняє достатню умову збіжності ітерацій")
```

```
Чисельний розв'язок системи
 x=[0.99865585 1.00656166 1.01250716 0.96765149 0.97843442 1.00333258
 1.02182158 1.02829329 1.02316179 1.00869615 0.98738837 0.96140406]
 обчислено за 913 ітерацій
```

[18]:
```
n=15
k, xk = Seidel_solver_Hilbert(n,set_matrix, set_vector, set_x0, eps,␣
↪max_iter)

if k > 0 and k < max_iter:
    print(f"Чисельний розв'язок системи \n x={xk} \n обчислено за {k}␣
↪ітерацій")
    #print(f"похибка чисельного розв'язку {norm_3(xk-x)}.")
elif  k == max_iter :
    print(f"Чисельний розв'язок системи не обчислено за {k} ітерацій")
else:
    print("Матриця СЛАР не задовольняє достатню умову збіжності ітерацій")
```

```
Чисельний розв'язок системи
 x=[1.00024743 0.98645069 1.0549502  0.97495445 0.95783066 0.97419367
 0.99810721 1.01765648 1.02884157 1.03127559 1.02599859 1.01446903
 0.99813178 0.9782552  0.95588898]
 обчислено за 767 ітерацій
```

Як бачимо з результатів обчислень, чисельне розв'язування заданої СЛАР ітераційними методами дає змогу отримати точніші чисельні розв'язки, ніж прямим методом Гаусса. Разом з тим, оскільки матриця такої системи не задовольняє достатньої умові збіжності, швидкість збіжності є малою, причому обраний критерій зупинки ітераційного процесу

не гарантує отримання чисельного розв'язку із заданою точністю.

2.5.6 Знаходження псевдорозв'язків несумісних СЛАР

Нехай n, m – довільні фіксовані натуральні числа, причому $n \geq 2$. Розглянемо СЛАР

$$A\,x = b, \tag{2.5.23}$$

де $x \in \mathbb{R}^n$ – невідомий вектор, $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$ - задані матриця і вектор вільних членів відповідно:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}, \quad b := \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}. \tag{2.5.24}$$

Вважаємо, що система (2.5.23) несумісна, тобто не має розв'язків.

Означення. Псевдорозв'язком *несумісної* системи (2.5.23) називають вектор $x^* \in \mathbb{R}^n$ такий, що

$$\|Ax^* - b\| = \inf_{x\in\mathbb{R}^n} \|Ax - b\|, \tag{2.5.25}$$

де $\|\cdot\|$ — одна із норм в просторі $\mathbb{R}^m$.

Відомо, що псевдорозв'яки системи (2.5.23) існують та збігаються з розв'язками системи

$$A^\top Ax = A^\top b. \tag{2.5.26}$$

Для знаходження псевдорозв'язків використаємо функцію `linalg.solve` з бібліотеки `SciPy` і метод `dot` множення багатовимірних масивів з бібліотеки `NumPy`.

Пояснення до використання програмного коду

- Підготувати обчислювальне середовище і потрібні функції :
  1. виконати комірку для підготовки середовища (з імпортом `NumPy` і `SciPy.linalg`)
  2. виконати комірку, в якій визначена функція `linalg_solver`
- Обчислити шуканий псевдорозв'язок заданої СЛАР (2.5.23):
  1. виконати комірки, в яких визначені функції `set_matrix` і `set_vector`
  2. виконати комірку з викликом функції `linalg_solver`

Програмна реалізація методу

Підготовка середовища

[1]:
```
import numpy as np
from scipy import linalg
```

`linalg_solver` – функція, яка організовує обчислення чисельного розв'язку СЛАР (2.5.23)

- На початку формується матриця і вектор вільних членів СЛАР (2.5.23)
- Далі обчислюється транспонована матриця і формується СЛАР (2.5.26)
- Після цього функція `linalg.solve` обчислює чисельний розв'язок СЛАР (2.5.26)

[2]:
```
def linalg_solver(set_matrix, set_vector):
    """
      a.T -- значенням виразу є транспонована матриця для матриці a
      a.dot(b) -- значенням виразу є добуток матриць   a і b
    """
    a=set_matrix()
```

```
    b=set_vector()
    aT=a.T
    aTa=aT.dot(a)
    aTb=aT.dot(b)
    return linalg.solve(aTa, aTb)
```

set_matrix і set_vector – функції, як і раніше задають матрицю та векторів вільних членів конкретної СЛАР.

Обчислювальні експерименти
Продемонструємо обчислення псевдорозв'язків.

Приклад 1. (Приклад 2.6) Обчислити псевдорозв'язки СЛАР

$$\begin{cases} x_1 + x_2 &= 2, \\ 2x_1 - x_2 &= 1, \\ 4x_1 + x_2 &= 6. \end{cases}$$

Підготуємо функції, які задаватимуть матрицю і вектор вільних членів СЛАР (2.5.23):

[3]:
```
def set_matrix():
    """ функція для задання матриці СЛАР (2.5.23)"""
    matrix=np.array([[1, 1],[2,-1],[4,1]],dtype=float )
    return matrix
```

[4]:
```
def set_vector():
    """ функція для задання вектора вільних членів СЛАР(2.5.23)"""
    vector=np.array([[2], [1], [6]],dtype=float)
    return vector
```

[5]:
```
x=linalg_solver(set_matrix, set_vector)
print(f"Псевдорозв'язок СЛАР \n x={x}")
```

```
Псевдорозв'язок СЛАР
 x=[[1.16666667]
 [1.16666667]]
```

Приклад 2. (вправа 2) Обчислити псевдорозв'язки СЛАР

$$\begin{cases} 3x_1 + x_2 &= 2, \\ x_1 + 4x_2 &= 5, \\ 4x_1 + 5x_2 &= 5. \end{cases}$$

Підготуємо функції, які задаватимуть матрицю і вектор вільних членів СЛАР (2.5.23):

[6]:
```
def set_matrix():
    """ функція для задання матриці СЛАР"""
    matrix=np.array([[3, 1],[1,4],[4,5]],dtype=float )
    return matrix
```

[7]:
```
def set_vector():
    """ функція для задання вектора вільних членів СЛАР"""
    vector=np.array([[2], [5], [5]],dtype=float)
    return vector
```

[8]:
```
x=linalg_solver(set_matrix, set_vector)
print(f"Псевдорозв'язок СЛАР \n x={x}")
```

```
Псевдорозв'язок СЛАР [[0.09090909]
 [1.06060606]]
```

Приклад 3. (вправа 4) Обчислити псевдорозв'язки СЛАР

$$\begin{cases} x_1 + 2x_2 + x_3 &= 3 \\ 2x_1 + 4x_2 + 2x_3 &= 5 \end{cases}$$

У цьому випадку матриця отриманої СЛАР (2.5.26) виявилася сингулярною, тому використання `scipy.linalg` зумовить помилку під час виконання цієї функції. Щоб пересвідчитися в цьому, підготуємо функції, які задаватимуть матрицю і вектор вільних членів заданої СЛАР і підставимо їх у функцію `linalg_solver`:

[9]:
```
def set_matrix():
    """ функція для задання матриці СЛАР"""
    matrix=np.array([[1, 2, 1],[2,4,2]],dtype=float )
    return matrix
```

[10]:
```
def set_vector():
    """ функція для задання вектора вільних членів СЛАР"""
    vector=np.array([[3], [5]],dtype=float)
    return vector
```

[11]:
```
x=linalg_solver(set_matrix, set_vector)
print(f"Псевдорозв'язок СЛАР \n x={x}")
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
C:\Users\AF9AD~1.MUZ\AppData\Local\Temp/ipykernel_24244/4098447305.py in <modul >
----> 1 x=linalg_solver(set_matrix, set_vector)
      2 print('Псевдорозв\'язок СЛАР', x)

C:\Users\AF9AD~1.MUZ\AppData\Local\Temp/ipykernel_24244/980201514.py in␣
 ↪linalg_solver(set_matrix, set_vector)
      9     aTa=aT.dot(a)
     10     aTb=aT.dot(b)
---> 11     return linalg.solve(aTa, aTb)

~\anaconda3\lib\site-packages\scipy\linalg\basic.py in solve(a, b, sym_pos,␣
 ↪lower, overwrite_a, overwrite_b, debug, check_finite, assume_a, transposed)
    217                                                 (a1, b1))
    218         lu, ipvt, info = getrf(a1, overwrite_a=overwrite_a)
--> 219         _solve_check(n, info)
    220         x, info = getrs(lu, ipvt, b1,
    221                         trans=trans, overwrite_b=overwrite_b)

~\anaconda3\lib\site-packages\scipy\linalg\basic.py in _solve_check(n, info,␣
 ↪lamch, rcond)
     27                          '.'.format(-info))
     28     elif 0 < info:
---> 29         raise LinAlgError('Matrix is singular.')
     30
     31     if lamch is None:

LinAlgError: Matrix is singular.
```

2.5.7 Знаходження власних значень і власних векторів матриць

Нехай $A \in \mathbb{R}^n(\mathbb{C})$ – яка-небудь матриця. Число λ називається *власним значенням* матриці A, якщо існує ненульовий вектор $v \in \mathbb{C}^n$ такий, що виконується рівність

$$Av = \lambda v. \tag{2.5.27}$$

Тоді вектор v називається *власним вектором*, який відповідає власному значенню λ.

Для знаходження власних значень і власних векторів матриць можна використати бібліотечну функцію `eig`, яка належить модулю `linalg`, що є частиною бібліотеки `scipy`.

Першим аргументом функції `eig` є матриця. Якщо решту аргументів прийняти за замовчуванням, то результатом виконання цієї функції є два `numpy.ndarray` масиви: - вектор, елементами якого є власні числа, - матриця, стовпцями якої є власні вектори, впорядковані за порядком відповідних власних значень у попередньому векторі.

Пояснення до використання програмного коду

- Підготувати обчислювальне середовище і потрібні функції :
  1. виконати комірку для підготовки середовища, зокрема імпортувати модуль `linalg`
  2. виконати комірку, де **визначена** функція `eigenvalue_problem_solver`
- Обчислити власні числа і власні вектори заданої СЛАР :
  1. виконати комірку, де визначена функція `set_matrix`
  2. Виконати комірку з викликом функції `eigenvalue_problem_solver`

Програмна реалізація методів

Підготовка середовища

[1]:
```
import numpy as np
from scipy import linalg
```

`eigenvalue_problem_solver` – **функція, яка визначає процес обчислення власних значень та векторів матриці за допомогою** `linalg.eig`

[2]:
```
def eigenvalue_problem_solver(n, matrix):
    """ функція для задання конкретної матриці  """
    eig_values, eig_vectors = linalg.eig(matrix(n))
    return eig_values, eig_vectors.T
```

`residual_calc` – **функція для обчислення нев'язки** $residual := Av - \lambda v$ **після пістановки власних значень та відповідних власних векторів матриці у рівняння** (2.5.27)

[3]:
```
def residual_calc(matrix,eig_values,eig_vector):
    for i in range(matrix.shape[0]):
        residual = linalg.norm(matrix.dot(eig_vector[i,:]) -␣
 ↪eig_values[i]*eig_vector[i,:])
        print(f"i={i}, residual={residual}")
```

Обчислювальні експерименти

Продемонструємо на прикладах застосування функції `linalg.eig` для обчислення власних значень та векторів квадратних матриць.

Приклад 1. (example 5.3) Обчислити власні значення та вектори матриці

$$A = \begin{pmatrix} 15 & -2 & 2 \\ 1 & 10 & -3 \\ -2 & 1 & 0 \end{pmatrix}.$$

Визначимо функцію для задання матриці

[4]:
```
def set_matrix(n):
    """ функція для задання конкретної матриці  """
    return np.array([[15,  - 2,  2], [ 1,  10 , -3], [ -2, 1,␣
 ↪0]],dtype=float)
```

[5]:
```
lmbds, vs = eigenvalue_problem_solver(3,set_matrix)
vs
```

[5]:
```
array([[-0.08811726,  0.30873868,  0.94705637],
       [-0.94359219, -0.31169403,  0.11171665],
       [ 0.39292879,  0.91947889,  0.01286632]])
```

[6]:
```
print(f'Власні числа: \n{lmbds} ')
```

```
Власні числа:
[ 0.51208483+0.j 14.10255576+0.j 10.38535941+0.j]
```

[7]:
```
print(f'Власні вектори: \n{vs} ')
```

```
Власні вектори:
[[-0.08811726  0.30873868  0.94705637]
 [-0.94359219 -0.31169403  0.11171665]
 [ 0.39292879  0.91947889  0.01286632]]
```

Переконаємося, що для кожного з отриманих власних значень і відповідних власних векторів нев'язка буде малою:

[8]:
```
residual_calc(set_matrix(3),lmbds,vs)
```

```
i=0, residual=1.5207579772826452e-15
i=1, residual=2.5219419200973697e-15
i=2, residual=3.5658083926151344e-15
```

Приклад 2. Обчислити власні значення та вектори матриці Гільберта різних розмірів (див.приклади розв'язування методом Гаусса).

Визначимо функцію, яка генеруватиме матрицю Гільберта:

[9]:
```
def H_matrix(n):
    """ функція для задання матриці Гільберта"""
    a = np.empty((n,n),dtype=float)
    for i in range(n):
        for j in range(n):
            a[i,j] = 1/(i+j+1)
    return a
```

Обчислимо власні значення та вектори матриці Гільберта для різних значень `n`.

[10]:
```
n=3
Hvals, Hvects = eigenvalue_problem_solver(n,H_matrix)
```

[11]:
```
print(f'Власні числа: \n{Hvals} ')
```

```
Власні числа:
[1.40831893+0.j 0.12232707+0.j 0.00268734+0.j]
```

[12]:
```
print(f'Власні вектори: \n{Hvects} ')
```

```
Власні вектори:
[[ 0.82704493  0.4598639   0.32329844]
 [ 0.54744843 -0.52829024 -0.64900666]
 [ 0.12765933 -0.71374689  0.68867153]]
```

[13]:
```
n=5
Hvals, Hvects = eigenvalue_problem_solver(n,H_matrix)
```

[14]:
```
print(f'Власні числа: \n{Hvals} ')
```

```
Власні числа:
[1.56705069e+00+0.j 2.08534219e-01+0.j 1.14074916e-02+0.j
 3.05898040e-04+0.j 3.28792877e-06+0.j]
```

[15]:
```
print(f'Власні вектори: \n{Hvects} ')
```

```
Власні вектори:
[[-0.76785474 -0.44579106 -0.32157829 -0.25343894 -0.20982264]
 [-0.60187148  0.27591342  0.42487662  0.44390304  0.42901335]
 [-0.21421362  0.72410213  0.12045328 -0.30957397 -0.56519341]
 [-0.04716181  0.43266733 -0.66735044 -0.23302452  0.55759995]
 [ 0.00617386 -0.11669275  0.50616366 -0.76719119  0.37624555]]
```

[16]:
```
n=10
Hvals, Hvects = eigenvalue_problem_solver(n,H_matrix)
```

[17]:
```
print(f'Власні числа: \n{Hvals} ')
```

```
Власні числа:
[1.75191967e+00+0.j 3.42929548e-01+0.j 3.57418163e-02+0.j
 2.53089077e-03+0.j 1.28749614e-04+0.j 4.72968929e-06+0.j
 1.22896774e-07+0.j 2.14743882e-09+0.j 2.26674554e-11+0.j
 1.09322786e-13+0.j]
```

[18]:
```
print(f'Власні вектори: \n{Hvects} ')
```

```
Власні вектори:
[[ 6.99514891e-01  4.25998913e-01  3.16976988e-01  2.55523006e-01
   2.15277840e-01  1.86578238e-01  1.64946526e-01  1.47992143e-01
   1.34310446e-01  1.23016713e-01]
 [ 6.37640842e-01 -7.04371051e-02 -2.33556605e-01 -2.82108835e-01
  -2.93657165e-01 -2.90981823e-01 -2.82497991e-01 -2.71744237e-01
  -2.60323765e-01 -2.48988410e-01]
 [-3.03404206e-01  5.99560151e-01  3.78571505e-01  1.44313674e-01
  -2.74969528e-02 -1.46916578e-01 -2.29138818e-01 -2.85631577e-01
  -3.24264002e-01 -3.50361227e-01]
 [ 1.05515795e-01 -5.76271783e-01  2.01933680e-01  4.12043529e-01
   3.51322220e-01  1.95432922e-01  1.93990025e-02 -1.47212403e-01
  -2.94208926e-01 -4.19514164e-01]
 [-2.93126764e-02  3.22879147e-01 -5.83114115e-01 -1.69552768e-01
   2.29493333e-01  3.74836695e-01  3.09362024e-01  1.12143643e-01
  -1.54211321e-01 -4.48204956e-01]
 [-6.67484774e-03  1.27505803e-01 -4.97391459e-01  4.10971927e-01
   3.63294163e-01 -4.87344006e-02 -3.43217612e-01 -3.52928835e-01
  -7.57666942e-02  4.30053664e-01]
```

```
 [ 1.24745611e-03 -3.76884259e-02  2.55155899e-01 -5.66480585e-01
   2.25122887e-01  4.26343115e-01  2.94090306e-02 -3.71927497e-01
  -3.26500431e-01  3.66506465e-01]
 [ 1.87818384e-04 -8.42526387e-03  8.88775676e-02 -3.56877238e-01
   5.70640477e-01 -1.34051444e-01 -4.46875715e-01  6.15397744e-02
   4.94713512e-01 -2.69891725e-01]
 [ 2.16832927e-05 -1.37869181e-03  2.12016008e-02 -1.33416320e-01
   4.04674430e-01 -5.80924741e-01  2.13963522e-01  4.06588652e-01
  -4.93025119e-01  1.62312164e-01]
 [-1.67403199e-06  1.45567590e-04 -3.11587894e-03  2.84375525e-02
  -1.36071415e-01  3.75033634e-01 -6.16652433e-01  5.97017611e-01
  -3.13921546e-01  6.91298093e-02]]
```

Можна переконатися, що для кожного з отриманих власних значень і відповідних власних векторів нев'язка буде малою. Наприклад, нев'язка для останнього результату буде такою:

[19]:
```
residual_calc(H_matrix(10),Hvals,Hvects)
```

```
i=0, residual=1.0448048107080504e-15
i=1, residual=5.114581455958995e-16
i=2, residual=2.262086438359439e-16
i=3, residual=1.8628763940619207e-16
i=4, residual=1.0494865757486728e-16
i=5, residual=3.245010027011731e-16
i=6, residual=1.162316126507303e-16
i=7, residual=4.8236658328790986e-17
i=8, residual=5.4402650311295894e-17
i=9, residual=3.055098308349397e-17
```

На завершення ще розглянемо число обумовленості розглянутих матриць, яке обчислюють за формулою $K(A) := \frac{|\lambda_{max}|}{|\lambda_{min}|}$, де λ_{max} і λ_{min} – максимальне і мінімальне за абсолютною величиною власні значення матриці A. Матимемо $K(A_3) \approx 5 \times 10^2$, $K(A_5) \approx 10^5$ і $K(A_{10}) \approx 10^{13}$ при розмірах $n = 3$, $n = 5$ і $n = 10$ відповідно, що означає погану обумовленість матриць Гільберта вже при невеликих розмірах. Відомо, що чисельне розв'язування таких систем є проблематичним.

Отож, приходимо до висновку, що за допомогою бібліотечної функції `linalg.eig` можна обчислювати власні значення і власні вектори матриць з великою точністю. При чисельному розв'язуванні СЛАР таким шляхом можна дослідити обумовленість відповідних матриць.

Розділ 3

Чисельне розв'язування нелінійних рівнянь та їх систем

3.1. Розв'язування нелінійних рівнянь

Нехай задано рівняння

$$f(x) = 0, \tag{3.1.1}$$

де $f(x)$, $x \in \langle c, d\rangle$, – неперервна функція, $-\infty \leqslant c < d \leqslant +\infty$.

Тут і далі символ "$\langle$" означає відкриваючу круглу "(" або квадратну "[" дужку, а символ "$\rangle$" означає закриваючу круглу ")" або квадратну "]" дужку.

Під *коренем* або, іншими словами, *точним розв'язком* рівняння (3.1.1) розуміють число $x_* \in \langle c, d\rangle$, яке при підстановці його в дане рівняння перетворює рівняння в правильну рівність, тобто $f(x_*) = 0$. Ми будемо шукати наближення кореня даного рівняння з деякою заданою точністю $\varepsilon > 0$ чисельними методами.

Чисельне розв'язування рівняння (3.1.1) складається з двох етапів:

1-ий етап: відокремлення (локалізація) коренів, тобто пошук числових проміжків, на яких є і тільки один корінь цього рівняння;

2-ий етап: знаходження наближень коренів з наперед заданою точністю на кожному з виділених проміжків.

Наближені значення коренів рівняння (3.1.1) уточнюють різними ітераційними методами. Розглянемо деякі з них. Але спочатку наведемо методи локалізації коренів даного рівняння.

3.1.1. Методи локалізації коренів

3.1.1.1 Табуляція функції.

Для відокремлення коренів корисні такі відомі з математичного аналізу твердження.

Теорема 3.1.1. *Якщо функція $f(x)$, $x \in [a, b]$, є неперервною і набуває значень різних знаків на кінцях відрізка $[a, b]$, тобто $f(a)f(b) < 0$, то на інтервалі (a, b) є принаймі один корінь рівняння (3.1.1).*

Теорема 3.1.2. *Якщо функція $f(x)$, $x \in [a, b]$, є неперервною і має похідну на інтервалі (a, b), яка зберігає там постійний знак, тобто або $f'(x) < 0$, $x \in (a, b)$, або $f'(x) > 0$, $x \in (a, b)$, то рівняння (3.1.1) на відрізку $[a, b]$ не має коренів, якщо $f(a)f(b) > 0$, а якщо $f(a)f(b) < 0$, то має корінь і тільки один.*

В цьому пункті припускаємо, що $f \in C^1(\langle c, d\rangle)$, тобто функція f має неперервну на $\langle c, d\rangle$ похідну, та існують скінчені або нескінчені границі $\lim\limits_{x\to c+0} f(x) = y_* \neq 0$, $\lim\limits_{x\to d+0} f(x) = y^* \neq 0$. Також вважаємо, що похідна f' має не більше ніж скінченну кількість нулів, які можемо легко визначити.

Проміжки локалізації коренів рівняння (15) визначаємо *методом табуляції* функції f. Алгоритм цього методу такий.

1. Знаходимо похідну f' функції f і розв'язуємо рівняння

$$f'(x) = 0. \tag{3.1.2}$$

Можливі два випадки:
а) рівняння (3.1.2) коренів не має; б) рівняння (3.1.2) має хоча б один корінь.

У випадку а) функція f є монотонною, а отже, якщо $y_* y^* > 0$, то рівняння (3.1.1) коренів немає, а якщо $y_* y^* < 0$, то має і тільки один (див. теорему 3.1.2). Припустимо, що $y_* y^* < 0$. Візьмемо значення $z_0, z_1 \in (c, d)$, $z_0 < z_1$, такі, що $f(z_0) \cdot y_* > 0$ й $f(z_1) \cdot y^* > 0$. Іншими словами, якщо $y_* > 0$, то $f(z_0) > 0$, а якщо $y_* < 0$, то $f(z_0) < 0$. І так само вибираємо z_1. Отже, на відрізку $[z_0, z_1]$ рівняння (3.1.1) має і тільки один розв'язок.

Розглянемо випадок б). Нехай $z_1, ..., z_m$ — корені рівняння (3.1.2) $(m \in \mathbb{N})$. Для зручності вважатимемо, що

$$c < z_1 < .. < z_m < d.$$

Знайдемо

$$y_i = f(z_i), \quad i = \overline{1, m}.$$

Якщо $c \in \langle c, d\rangle$, то покладемо $z_0 := c$ і $y_0 = f(z_0)$, а якщо $c \notin \langle c, d\rangle$, то візьмемо яке-небудь число $z_0 \in (c, z_1)$ таке, що $f(z_0) \cdot y_* > 0$ і покладемо $y_0 = f(z_0)$. Аналогічно, якщо $d \in \langle c, d\rangle$, то покладемо $z_{m+1} := d$ і $y_{m+1} := f(z_{m+1})$, а якщо $d \notin \langle c, d\rangle$, то візьмемо яке-небудь $z_{m+1} \in (z_m, d)$ таке, що $f(z_{m+1}) \cdot y^* > 0$ і покладемо $y_{m+1} := f(z_{m+1})$.

Тепер розглянемо відрізки $[z_0, z_1], ..., [z_m, z_{m+1}]$. Нехай $[z_i, z_{i+1}]$ — один з них. Дивимося на знак добутку $y_i y_{i+1}$. Якщо $y_i y_{i+1} > 0$, то згідно з твердженням теореми 3.1.2 рівняння (3.1.1) на відрізку $[z_i, z_{i+1}]$ не має розв'язків, а якщо $y_i y_{i+1} < 0$, то на відрізку $[z_i, z_{i+1}]$ рівняння (3.1.1) має і тільки один корінь, а отже, це є проміжок локалізації коренів цього рівняння.

Тепер відмітимо, що для ефективного застосування чисельних методів знаходження коренів рівняння (3.1.1), як правило, бажано, щоб відрізок, на якому шукаємо корінь, мав довжину рівну або меншу за 1. Для цього можна застосувати такий прийом.

Нехай $[z_0, z_1]$ — відрізок, на якому рівняння (3.1.1) має корінь і тільки один. Візьмемо яке-небудь розбиття цього відрізку:

$$z_0 =: \widetilde{x}_0 < \widetilde{x}_1 < ... < \widetilde{x}_l := z_1,$$

де $l \in N$ — якесь число. Знаходимо

$$\widetilde{y}_i = f(\widetilde{x}_i), \quad i = \overline{0, l}.$$

Далі знаходимо значення $s \in 0, ..., l-1$ таке, що числа $\widetilde{y}_0, ..., \widetilde{y}_s$ — одного знаку, а число $\widetilde{y}_{s+1}$ — протилежного. Тоді корінь рівняння (3.1.1) на відрізку $[z_0, z_1]$ належить відрізку $[x_s, x_{s+1}]$, який має наперед визначену довжину.

Звичайно, якщо ми потрапимо у ситуацію, коли для якогось значення $i \in \{0, ..., l\}$ маємо $\widetilde{y}_i = 0$, шо це означатиме, що $\widetilde{x}_i$ є корінь рівняння (3.1.1) на відрізку $[z_0, z_1]$ і потреби у продовженні процесу розв'язування рівняння (3.1.1) на відрізку $[z_0, z_1]$ не буде.

Приклад 3.1.1. Локалізувати корені рівняння

$$x^3 - 12x = 0.$$

Розв'язування. Маємо $f(x) = x^3 - 12x, \quad x \in (-\infty, +\infty)$. Тоді

$$y_* := \lim_{x\to-\infty} f(x) = -\infty; \quad y^* := \lim_{x\to+\infty} f(x) = +\infty.$$

Знайдемо

$$f'(x) = 3x^2 - 12 = 3(x^2 - 4) = 3(x-2)(x+2).$$

Отже,

$$z_1 = -2; \quad z_2 = 2.$$

Вибираємо $z_0 \in (-\infty, -2)$ таке, щоби було $f(z_0) < 0$, бо $y_* = -\infty$. Легко побачити, що $f(-4) = -64+48 = -16 < 0$, а тому можемо взяти $z_0 = -4$. Тепер виберемо $z_3 \in (2; +\infty)$ таке, щоби було $f(z_3) > 0$, бо $y^* = +\infty$. Оскільки $f(4) = 64 - 48 = 16 > 0$, що можемо взяти $z_3 = 4$.

Отже, маємо

$$y_0 := f(-4) = -16; \quad y_1 := f(-2) = 16; \quad y_2 := f(2) = -16; \quad y_3 := f(4) = 16.$$

Отримані числові значення запишемо в таблицю:

x	-4	-2	2	4
y	-16	16	-16	16

Отже, на проміжках $[-4; -2]$, $[-2; 2]$, $[2; 4]$ знаходиться по одному з коренів даного рівняння.

Звузимо відрізки, на яких знаходяться корені даного рівняння. Розглянемо відрізок $[-4; -2]$ і візьмемо

$$\widetilde{x}_0 = -4; \quad \widetilde{x}_1 = -3; \quad \widetilde{x}_2 = -2.$$

Маємо

$$\widetilde{y}_0 := f(-4) = -16, \quad \widetilde{y}_1 := f(-3) = 9, \quad \widetilde{y}_2 := f(-2) = 16.$$

Запишемо ці дані в таблицю:

x	-4	-3	-2
y	-16	9	16

.

Звідси бачимо, що корінь даного рівняння з відрізку $[-4; -2]$ знаходиться на відрізку $[-4; -3]$.

Тепер розглянемо відрізок $[-2; 2]$. Візьмемо

$$\widetilde{x}_0 = -2; \quad \widetilde{x}_1 = -1; \quad \widetilde{x}_2 = 0; \quad \widetilde{x}_3 = 1; \quad \widetilde{x}_4 = 2$$

і знайдемо

$$\widetilde{y}_0 := f(-2) = 16; \quad \widetilde{y}_1 := f(-1) = 11; \quad \widetilde{y}_2 := f(0) = 0;$$

$$\widetilde{y}_3 := f(1) = -11; \quad \widetilde{y}_4 := f(2) = -16.$$

Звідси отримуємо, що коренем на відрізку $[-2; 2]$ є число 0. В силу непарності функції f корінь даного рівняння з відрізку $[2; 4]$ знаходиться на відрізку $[3; 4]$.

Легко знайти корені даного рівняння:

$$x_1 = -\sqrt{12} \approx -3,5; \quad x_2 = 0; \quad x_3 = \sqrt{12} \approx 3,5.$$

Бачимо, що $x_1 \in [-4; -3]$, $x_2 \in \{0\}$, $x_3 \in [3; 4]$, тобто ми добре визначили проміжки локалізації коренів даного рівняння. $\square$

3.1.1.2 Графічний спосіб.

Універсальним методом відокремлення коренів є побудова графіка функції $y = f(x)$ за допомогою комп'ютера, тобто графічне відокремлення. Якщо графік функції f досить складно побудувати, то можна записати рівняння (3.1.1) у вигляді рівносильного йому рівняння

$$\varphi(x) = \psi(x), \tag{3.1.3}$$

а тоді побудувати графіки функцій $\varphi(x)$, $\psi(x)$, $x \in \langle c, d\rangle$. Абсциси точок перетину цих графіків є розв'язками рівняння (3.1.3), а отже, і рівняння (3.1.1)

3.1.1.3 Аналітичний метод відділення коренів.

Розглянемо рівняння (3.1.1), коли воно є алгебраїчним, тобто має вигляд

$$f(x) := a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 = 0, \tag{3.1.4}$$

де $a_j \in \mathbb{R}$, $j = \overline{0, n}$.

Далі всюди будемо вважати, що $a_n > 0$, оскільки в протилежному випадку дане рівняння множенням на -1 зводимо до еквівалентного з вказаною умовою.

Спочатку вияснимо способи знаходження меж для додатних і від'ємних коренів рівняння (4.2.1). При цьому треба пам'ятати таке: якщо вказують межі для дійсних коренів рівняння, то це зовсім не означає, що в цих межах насправді існують корені.

Зауважимо, що для відшукання меж додатних і від'ємних коренів рівняння (4.2.1) досить вміти шукати верхню межу додатних коренів цього рівняння. Цей висновок базується на такому твердженні.

Твердження 3.1.1. *Якщо N_0 — верхня межа додатних коренів рівняння (4.2.1), а N_1, N_2, N_3 — відповідно верхні межі додатних коренів рівнянь*

$$x^n f(\frac{1}{x}) = 0, \quad f(-x) = 0, \quad x^n f(-\frac{1}{x}) = 0,$$

то додатні корені рівняння (4.2.1) є на відрізку $[\frac{1}{N_1}, N_0]$, а від'ємні — на відрізку $[-N_2, -\frac{1}{N_3}]$.

Доведення. Справді, якщо α — додатний корінь рівняння (4.2.1), то $\frac{1}{\alpha}$ — додатний корінь рівняння $x^n f(\frac{1}{x}) = 0$ і $\frac{1}{\alpha} \leqslant N_1$. Звідси маємо $\alpha \geqslant \frac{1}{N_1}$. Якщо β — від'ємний корінь рівняння (4.2.1), то $-\beta$ — додатний корінь рівняння $f(-x) = 0$ і $-\beta \leqslant N_2$. Звідси $\beta \geqslant -N_2$. Якщо β — від'ємний корінь рівняння (4.2.1), то $-\frac{1}{\beta}$ — додатний корінь рівняння $x^n f(-\frac{1}{x}) = 0$ і $-\frac{1}{\beta} \leqslant N_3$. Звідси $\beta \leqslant -\frac{1}{N_3}$. □

Формула Маклорена для відшукання верхньої межі додатних коренів. Нехай в рівнянні (4.2.1) маємо $a_n > 0$ і a_{n-k} $(k \geqslant 1)$ — перший із від'ємних коефіцієнтів рівняння (4.2.1), рахуючи зліва направо (якби рівняння не мало від'ємних коефіцієнтів, то воно не мало б і додатних коренів). Приймемо, що

$$b = \max_{a_i < 0} |a_i|.$$

Теорема 3.1.3. *Усі дійсні корені рівняння (4.2.1) задовольняють нерівність*

$$x \leqslant 1 + \sqrt[k]{\frac{b}{a_n}}. \tag{3.1.5}$$

Приклад 3.1.2. Знайти верхню межу коренів рівняння

$$2x^4 - 11x^3 + 16x^2 - x - 6 = 0.$$

Розв'язування. Оскільки $n = 4$, $a_n = a_4 = 2$, $a_{n-k} = a_3 = -11$, то $k = 1$. Маємо $b = \max\{11, 1, 6\} = 11$. Тому всі корені рівняння задовольняють нерівність $x \leqslant 6,5$. Зауважимо, що це досить неточна межа, оскільки коренями цього рівняння є числа $-\frac{1}{2}$; 1, 2, 3. □

Приклад 3.1.3. Відшукати верхню межу коренів рівняння

$$x^4 + 3x^3 - 37x^2 + 33x + 72 = 0.$$

Розв'язування. Оскільки $a_n = a_4 = 1$, $a_{n-k} = a_2 = -37$, то $k = 2$, $b = 37$. Тому всі корені рівняння задовольняють нерівність $x \leqslant 1 + \sqrt{37}$. Зауважимо, що коренями рівняння є числа -8; -1, 3, 3. □

Метод Ньютона для відшукання верхньої межі коренів. Цей метод базується на такому твердженні.

Теорема 3.1.4. *Якщо для деякого $c \in \mathbb{R}$ виконуються умови*

$$f(c) > 0, \quad f'(c) > 0, \quad f''(c) > 0, \quad \ldots, \quad f^{(n)}(c) > 0,$$

то число c є верхньою межею коренів рівняння (4.2.1).

Доведення. На основі формули Тейлора отримаємо

$$f(x) = f(c) + \frac{f'(c)}{1!}(x-c) + \frac{f''(c)}{2!}(x-c)^2 + \cdots + \frac{f^{(n)}(c)}{n!}(x-c)^n.$$

Звідси видно, що для всіх $x \geqslant c$ маємо $f(x) > 0$, тобто серед чисел більших або рівних c не може бути коренів рівняння (4.2.1). А це означає, що число c є верхньою межею коренів рівняння (4.2.1). Теорема доведена. □

Відшукати значення c можна, базуючись на таких міркуваннях. Оскільки похідна $f^{(n)}(x) = a_n \cdot n!$, $x \in (-\infty, +\infty)$, є додатною сталою, бо $a_n > 0$, то функція $f^{(n-1)}$ є зростаючою. Це означає, що існує число c_1 таке, що $f^{(n-1)}(x) > 0$ при $x \geqslant c_1$. Звідси випливає, що функція $f^{(n-2)}$ на промені $[c_1, +\infty)$ є зростаючою. Тому існує число $c_2 \geqslant c_1$ таке, що $f^{(n-2)}(x) > 0$ при $x \geqslant c_2$. Міркуючи так і далі, ми нарешті знайдемо число $c_n \geqslant c_{n-1}$, що при $x \geqslant c_n$ виконуватиметься нерівність $f(x) > 0$. Отже, $c = c_n$.

Приклад 3.1.4. Знайти за допомогою методу Ньютона верхню межу коренів рівняння

$$f(x) := x^4 + 3x^3 - 37x^2 + 33x + 72 = 0.$$

Розв'язування. Відшукаємо спочатку всі похідні від f до четвертого порядку включно. Одержимо

$$f'(x) = 4x^3 + 9x^2 - 74x + 33, \quad f''(x) = 12x^2 + 18x - 74,$$
$$f^{(3)}(x) = 24x + 18, \quad f^{(4)}(x) = 24 > 0.$$

Маємо $f^{(3)}(x) > 0$ при $x > -\frac{3}{4}$; $f''(x) > 0$, наприклад, при $x > 2$; $f'(x) > 0$ при $x > 3$. Тому за верхню межу коренів рівняння можна взяти будь-яке число $c > 3$. □

Спосіб Штурма відокремлення коренів. Припустимо, що рівняння (4.2.1) з дійсними коефіцієнтами не має кратних коренів (оскільки інакше ми могли б многочлен $f(x)$ розділити на найбільший спільний дільник його і його похідної). Складемо таку систему функцій:

$$f(x),\ f'(x),\ R_1(x),\ R_2(x),\ \ldots,\ R_{m-1}(x),\ R_m(x), \quad x \in \mathbb{R}, \tag{3.1.6}$$

де через $R_1(x)$ позначено остачу від ділення $f(x)$ на $f'(x)$, взяту з протилежним знаком; через $R_2(x)$ — остачу від ділення $f'(x)$ на $R_1(x)$, взяту також з протилежним знаком; і так доти, доки не дійдемо до $R_m(x) = const$. Ця система функцій, а також будь-яка

інша, отримана з цієї множенням функцій, що входять у систему, на сталі додатні числа, є однією з т. зв. систем Штурма. Нехай

$$f(x),\ f_0(x),\ f_1(x)\ ,\ldots,\ f_m(x), \quad x \in \mathbb{R},$$

– система функцій, яка збігається з системою (3.1.6) з точністю хіба що до додатних сталих множників. Обчислимо значення цих функцій у двох точках a і b:

$$f(a),\ f_0(a),\ f_1(a),\ \ldots,\ f_m(a); \tag{3.1.7}$$

$$f(b),\ f_0(b),\ f_1(b),\ \ldots,\ f_m(b). \tag{3.1.8}$$

Позначимо кількість перемін знаків у (3.1.7) через $W(a)$, а у (3.1.8) — через $W(b)$.

Теорема 3.1.5. *Якщо рівняння (4.2.1) не має кратних коренів і дійсні числа a та b, $a < b$, не є коренями цього рівняння, то $W(a) \geqslant W(b)$ і різниця $W(a) - W(b)$ дорівнює кількості дійсних коренів даного рівняння, розміщених між a та b.*

Доведення. Доведення теореми є в курсі з вищої алгебри. □

Приклад 3.1.5. Користуючись теоремою 3.1.5, відокремити дійсні корені рівняння

$$f(x) := x^3 + 3x^2 - 1 = 0.$$

Розв'язання. Знайдемо спочатку межі дійсних коренів цього рівняння. Очевидно, що $f(x) > 0$ при $x \geqslant 1$, оскільки тоді $x^3 - 1 \geqslant 0$ і $3x^2 > 0$. Тому число 1 можна прийняти за верхню межу коренів рівняння. Для відшукання нижньої межі розглянемо рівняння

$$f(-x) \equiv -x^3 + 3x^2 - 1 = 0.$$

При $x \geqslant 3$ маємо $f(-x) < 0$, бо тоді $-x^3 + 3x^2 = -x^2(x-3) \leqslant 0$. Тому число 3 є верхньою межею коренів рівняння $f(-x) = 0$. А це означає, що число -3 є нижньою межею коренів заданого рівняння. Отже, всі дійсні корені рівняння лежать в проміжку $[-3; 1]$.

Складемо систему Штурма:

$$f(x) = x^3 + 3x^2 - 1, \quad f_0(x) = x^2 + 2x, \quad f_1(x) = 2x + 1, \quad f_2(x) = 1.$$

Таблиця зміни знаків у ряді Штурма така:

x	$f(x)$	$f_0(x)$	$f_1(x)$	$f_2(x)$	$W(x)$
-3	$-$	$+$	$-$	$+$	3
-2	$+$	$+$	$-$	$+$	2
-1	$+$	$-$	$-$	$+$	2
0	$-$	$+$	$+$	$+$	1
1	$+$	$+$	$+$	$+$	0

Отже, дійсні корені рівняння (3.1.1) лежать у проміжках: $[-3; -2], [-1; 0], [0; 1]$. □

3.1.2. Метод ділення навпіл (дихотомії або бісекції)

Нехай на відрізку $[a, b]$ рівняння (3.1.1) має і тільки один корінь x_*, причому $f(a)f(b) < 0$. Припустимо, що нам потрібно знайти наближення цього кореня з точністю $\varepsilon > 0$.

Обчислення будемо виконувати за такою схемою. Позначимо $x_0 = a$, $x_1 = b$. Очевидно, що корінь рівняння (3.1.1) належить відрізку $[x_0, x_1]$. Тоді покладемо $x_2 := (x_0 + x_1)/2$ і

за x_3 виберемо те зі значень x_0 чи x_1, для якого $f(x_2)f(x_3) < 0$. Отже, x_* лежить між x_2 і x_3. Далі покладемо $x_4 := (x_2+x_3)/2$ та обчислюємо $f(x_4)$. Тоді за x_5 беремо те зі значень x_2 чи x_3, для якого $f(x_4)f(x_5) < 0$. Тоді x_* лежить між x_4 і x_5. Цей процес продовжуємо доти, доки довжина відрізка, що з'єднує точки x_{2k} та x_{2k+1} і містить корінь x_*, не стане меншою за ε, тобто $|x_{2k+1} - x_{2k}| < \varepsilon$. Середина відрізка, що з'єднує точки x_{2k} та x_{2k+1}, тобто $x_{2k+2} := (x_{2k} + x_{2k+1})/2$, дає значення наближення кореня із заданою точністю ε.

Зауваження 3.1.1. Даний ітераційний процес, очевидно, збігається зі швидкістю геометричної прогресії зі знаменником 1/2, тобто

$$|x_* - x_{2k+2}| \leqslant 2^{-k}|x_1 - x_0| = 2^{-k}|b - a|, \quad k \in \mathbb{N}. \tag{3.1.9}$$

Звідси випливає, що значення $k \in \mathbb{N}$, при якому $|x_* - x_{2k+2}| < \varepsilon$ отримуємо з нерівності

$$2^{-k}|b - a| < \varepsilon. \tag{3.1.10}$$

Отож, для знаходження наближення кореня рівняння (3.1.1) з точністю ε спочатку визначаємо якомога менше число k, при якому виконується нерівність (3.1.10), а тоді обчислюємо шукане наближення x_{2k+2} кореня x_*.

Основний недолік цього методу — повільна збіжність.

3.1.3 . Метод послідовних наближень (простої ітерації) з використанням стискуючих відображень

Нехай рівняння (3.1.1) на відрізку $[a, b]$ має і тільки один корінь x_*. Запишемо це рівняння у вигляді

$$x = \varphi(x), \tag{3.1.11}$$

де

$$\varphi(x) := x + \rho(x)f(x), \quad x \in [a, b],$$

а $\rho : [a, b] \to \mathbb{R}$ – довільна неперервна функція, яка не має коренів на $[a, b]$. Зокрема, може бути $\rho(x) = 1$, $x \in [a, b]$.

Метод простої ітерації або, іншими словами, *метод послідовних наближень* визначається так:

• задаємо початкове значення $x_0 \in [a, b]$ і

• знаходимо послідовні наближення $x_1, x_2, x_3, \ldots$ розв'язку рівняння (3.1.11), використовуючи формулу

$$x_{n+1} = \varphi(x_n), \quad n = 0, 1, 2, \ldots . \tag{3.1.12}$$

Означення 3.1.1. *Кажуть, що функція $\varphi : [a, b] \to \mathbb{R}$ задовольняє умову Ліпшиця зі сталою $q > 0$, якщо*

$$|\varphi(x'') - \varphi(x')| \leqslant q|x'' - x'|, \quad x',\ x'' \in [a, b], \tag{3.1.13}$$

Якщо ж додатково маємо, що $q \in (0, 1)$, то функцію φ називають стискуючим відображенням.

Зауваження 3.1.2. Якщо функція φ має похідну на $[a, b]$, то умова Ліпшиця виконується, коли

$$|\varphi'(x)| \leqslant q \quad \forall x \in [a, b],$$

бо тоді згідно з формулою скінчених приростів маємо

$$|\varphi(x'') - \varphi(x')| = |\varphi'(\xi)||x'' - x'| \leqslant q|x'' - x'|,$$

де $\xi = x' + \theta(x'' - x'),\ 0 < \theta < 1$.

Правильні такі твердження.

Теорема 3.1.6 (наслідок з принципу стискуючих відображень)**.** *Нехай функція* $\varphi : [a,b] \to \mathbb{R}$ *задовольняє умову Ліпшиця зі сталою* $q \in (0,1)$ *і* $\varphi([a,b]) \subset [a,b]$ *(тобто* $\varphi(x) \in [a,b]$ *для кожного* $x \in [a,b]$*). Тоді рівняння* (3.1.11) *має на відрізку* $[a,b]$ *єдиний корінь* x_* *і він є границею послідовності* $\{x_n\}$*, що визначається формулою* (3.1.12)*, де* x_0 *вибирається довільно з відрізка* $[a,b]$*.*

Теорема 3.1.7. *Нехай функція* $\varphi : [a,b] \to \mathbb{R}$ *задовольняє умову Ліпшиця зі сталою* $q \in (0,1)$ *та* x_0 *і* $d > 0$ *такі, що* $[x_0 - d, x_0 + d] \subset [a,b]$ *і*

$$|\varphi(x_0) - x_0| \leqslant (1-q)d. \tag{3.1.14}$$

Тоді рівняння (3.1.11) *має на відрізку* $[x_0 - d, x_0 + d]$ *єдиний корінь* x_* *і він є границею послідовності* $\{x_n\}$*, що визначається формулою* (3.1.12)*.*

Доведення. Покажемо, що відображення φ переводить множину $[x_0 - d, x_0 + d]$ в себе. Справді, якщо $x \in [x_0 - d, x_0 + d]$, тобто $|x - x_0| \leqslant d$, то

$$\begin{aligned}&|\varphi(x) - x_0| = |\varphi(x) - \varphi(x_0) + \varphi(x_0) - x_0| \leqslant |\varphi(x) - \varphi(x_0)| + |\varphi(x_0) - x_0| \\ &\leqslant q|x - x_0| + (1-q)d \leqslant qd + (1-q)d = d.\end{aligned}$$

Крім того, $\varphi : [x_0 - d, x_0 + d] \to [x_0 - d, x_0 + d]$ — стискуюче відображення в силу умови Ліпшиця зі сталою $q \in (0,1)$.

Отже, на підставі принципу стискаючих відображень на відрізку $[x_0 - d, x_0 + d]$ існує єдиний розв'язок рівняння (3.1.11) і він є границею послідовності $\{x_n\}$, що визначається формулою (3.1.12). □

Зауваження 3.1.3. Умова (3.1.14) є підказкою для правильного вибору початкового наближення кореня рівняння (3.1.11).

При використанні методу послідовних наближень для похибки $x_{n+1} - x_*$ маємо оцінку

$$|x_{n+1} - x_*| = |\varphi(x_n) - \varphi(x_*)| \leqslant q|x_n - x_*| \leqslant \cdots \leqslant q^{n+1}|x_0 - x_*| \leqslant q^{n+1}|b - a|.$$

Тому кажуть, що метод послідовних наближень збігається зі швидкістю геометричної прогресії зі знаменником q.

3.1.4 . Метод Ньютона (дотичних)

Розглянемо рівняння (3.1.1) на відрізку $[a,b] \subset \langle c,d \rangle$ при умові, що на цьому відрізку існує і тільки один корінь x_* цього рівняння, причому $f(a) \cdot f(b) < 0$.

Вважаємо, що $f \in C^1([a,b])$, $f'(x) \neq 0$ для всіх $x \in [a,b]$, і будуємо послідовність

$$x_0, x_1, \ldots, x_n, \ldots \tag{3.1.15}$$

наближень кореня x_* таким чином. Значення $x_0 \in [a,b]$ виберемо довільно, опираючись, можливо, на якусь додаткову інформацію. Далі, якщо $x_n \in [a,b]$ — відомий член послідовності (3.1.15), то за наступний член x_{n+1} беремо абсцису точки перетину дотичної до графіка функції f в точці $(x_n, f(x_n))$. Для того, щоби цей метод був коректним, необхідне виконання умови: $x_n \in [a,b]$ для кожного $n \in \mathbb{N}$.

Запишемо формулу для знаходження x_{n+1}. Рівняння дотичної до графіка f в точці $(x_n, f(x_n))$ має вигляд

$$y = f(x_n) + f'(x_n)(x - x_n), \quad x \in \mathbb{R}.$$

Поклавши тут $y = 0$, отримаємо

$$0 = f(x_n) + f'(x_n)(x_{n+1} - x_n),$$

звідки

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n = 0, 1, 2 \ldots . \tag{3.1.16}$$

Покажемо, що коли $\{x_n\} \subset [a,b]$ та $x_n \to \widehat{x}$ при $n \to \infty$, то $\widehat{x} = x_*$ — корінь рівняння (3.1.1). Справді, перейшовши в (3.1.16) до границі при $n \to \infty$, отримаємо

$$\widehat{x} = \widehat{x} - \frac{f(\widehat{x})}{f'(\widehat{x})},$$

звідки

$$f(\widehat{x}) = 0,$$

тобто $\widehat{x} \in [a,b]$ — корінь рівняння (3.1.1). Оскільки, на відрізку $[a,b]$ дане рівняння має тільки один корінь x_*, то $\widehat{x} = x_*$.

Цей метод називають *методом Ньютона* або, іншими словами, *методом дотичних.*

Записавши рівняння (3.1.1) у вигляді (3.1.11), де

$$\varphi(x) = x - \frac{f(x)}{f'(x)},$$

помічаємо, що метод Ньютона є методом простої ітерації розв'язування рівняння (3.1.11).

Розглянемо теорему, яка конкретно вказує на вибір початкового наближення для одного класу функцій f.

Теорема 3.1.8. *Нехай $f(a)f(b) < 0$, функції f', f'' неперервні і відмінні від нуля на $[a,b]$ або, що те саме, зберігають знак на $[a,b]$. Тоді рівняння (3.1.1) має і тільки один корінь x_* та, якщо початкове $x_0 \in [a,b]$ задовольняє умову*

$$f(x_0)f''(x_0) > 0, \tag{3.1.17}$$

то послідовність $\{x_n\}$, отримана методом Ньютона, збігається до кореня x_*.

Крім того, для довільного $n \in \mathbb{N}$ маємо оцінку відхилення x_n від x_*:

$$|x_n - x^*| \leqslant \frac{M_2}{2m_1}|x_n - x_{n-1}|^2, \tag{3.1.18}$$

де

$$M_2 := \max_{x\in[a,b]} |f''(x)|, \quad m_1 := \min_{x\in[a,b]} |f'(x)|.$$

Доведення. Спочатку розглянемо випадок

$$f(a) < 0, \quad f(b) > 0, \quad f'(x) > 0, \quad f''(x) > 0, \quad x \in [a,b].$$

Тоді точка $x_0 \in [a,b]$, яка задовольняє умову (3.1.17), міститься, очевидно, справа від x^*, тобто $x_0 > x_*$, бо $f(x_0) > 0$.

Покажемо, що $x_n \in [x_*, x_{n-1}] \subset [a,b]$ для будь-якого $n \in \mathbb{N}$. Для цього застосуємо метод математичної індукції.

Оскільки $f(x_0)/f'(x_0) > 0$ і

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)},$$

то $x_1 < x_0$. Використовуючи формулу Тейлора, одержимо

$$0 = f(x_*) = f(x_0) + f'(x_0)(x_* - x_0) + \frac{1}{2}f''(\xi)(x_* - x_0)^2, \quad \xi \in (x_*, x_0),$$

звідки

$$f(x_0) = -f'(x_0)(x_* - x_0) - \frac{1}{2}f''(\xi)(x_* - x_0)^2,$$

а тому

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = x_0 + x_* - x_0 + \frac{1}{2}\frac{f''(\xi)}{f'(x_0)}(x^* - x_0)^2 > x_*,$$

тобто $x_1 \in [x_*, x_0] \subset [a,b]$.

Тепер припустимо, що $x_n \in [x_*, x_{n-1}] \subset [a, b]$ для деякого $n \in \mathbb{N}$, і доведемо, що $x_{n+1} \in [x_*, x_n]$.

Оскільки за припущенням $x_n \in [x_*, x_{n-1}] \subset [a, b]$, то $f(x_n) > 0$, а тому

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} < x_n.$$

За формулою Тейлора маємо

$$0 = f(x_*) = f(x_n) + f'(x_n)(x_* - x_n) + \frac{1}{2} f''(\xi)(x_* - x_n)^2, \quad \xi \in (x_*, x_n),$$

звідки

$$f(x_n) = -f'(x_n)(x_* - x_n) - \frac{1}{2} f''(\xi)(x_* - x_n)^2,$$

а отже,

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n + x_* - x_n + \frac{1}{2}\frac{f''(\xi)}{f'(x_n)}(x_* - x_n)^2 > x_*.$$

Отже, $x_{n+1} \in [x^*, x_n]$ що і треба було довести. А це означає, що послідовність $\{x_n\}$ монотонно спадає та обмежена знизу, тобто існує границя $\lim\limits_{n\to\infty} x_n = \widehat{x}$. Перейшовши до границі в (3.1.16), переконуємося, що $\widehat{x} = x_*$.

Аналогічно розглядаються інші можливі випадки розміщення знаків $f(a)$, $f(b)$, f', f''.

Оцінимо похибку $|x_n - x^*|$. За формулою Лагранжа маємо

$$f(x_n) - f(x^*) = (x_n - x_*) f'(\xi)$$

або

$$x_n - x_* = \frac{f(x_n)}{f'(\xi)},$$

звідки

$$|x_n - x_*| \leqslant \frac{|f(x_n)|}{m_1}. \tag{3.1.19}$$

За формулою Тейлора отримаємо

$$f(x_n) = f(x_{n-1}) + (x_n - x_{n-1}) f'(x_{n-1}) + \frac{1}{2} f''(\eta)(x_n - x_{n-1})^2,$$

де

$$\eta = x_{n-1} + \theta(x_n - x_{n-1}), \quad \theta \in (0, 1) \ \text{— деяке число.}$$

Оскільки з (3.1.16) маємо

$$f(x_{n-1}) + (x_n - x_{n-1}) f'(x_{n-1}) = 0,$$

то

$$|f(x_n)| \leqslant \frac{1}{2} M_2 |x_n - x_{n-1}|^2.$$

Звідси та з нерівності (3.1.19) випливає оцінка (3.1.18). □

Зауваження 3.1.4. Оцінка (3.1.18) є апостеріорною, а тому зручною для практичного застосування і свідчить про високу швидкість збіжності методу Ньютона. Недоліками методу є те, що на кожній ітерації потрібно обчислювати значення функції та її похідної, а також складність вибору початкового наближення.

3.1.5. Метод хорд (лінійної інтерполяції, пропорційних частин, січних)

Розглянемо рівняння (3.1.1) на відрізку $[a, b] \subset \langle c, d\rangle$, який є проміжком локалізації кореня цього рівняння, тобто на $[a, b]$ існує і тільки один корінь x_* цього рівняння і $f(a) \cdot f(b) < 0$.

Будуємо послідовність

$$x_0, x_1, x_2, \ldots, x_n, \ldots \tag{3.1.20}$$

наближень кореня $x_* \in [a, b]$ таким чином. Значення $x_0, x_1 \in [a, b]$ є початковим і ми задаємо їх довільно. Далі, якщо відомі значення x_{n-1}, x_n, то наближення x_{n+1} беремо абсцису точки перетину з віссю абсцис прямої, що проходить через точки $(x_{n-1}, f(x_{n-1}))$ та $(x_n, f(x_n))$.

Запишемо формулу для знаходження x_{n+1}. Спочатку введемо позначення: $y_{n-1} := f(x_{n-1})$, $y_n := f(x_n)$. Тоді запишемо рівняння прямої, що проходить через точки (x_{n-1}, y_{n-1}) і (x_n, y_n):

$$\frac{x - x_n}{x_{n-1} - x_n} = \frac{y - y_n}{y_{n-1} - y_n}.$$

Звідси, поклавши $y = 0$, $x = x_{n+1}$, одержуємо

$$\frac{x_{n+1} - x_n}{x_{n-1} - x_n} = \frac{-y_n}{y_{n-1} - y_n},$$

тобто

$$x_{n+1} - x_n = -\frac{(x_n - x_{n-1}) \cdot y_n}{y_n - y_{n-1}}.$$

Отож, маємо таку формулу для знаходження x_{n+1}:

$$x_{n+1} = x_n - \frac{(x_n - x_{n-1}) \cdot f(x_n)}{f(x_n) - f(x_{n-1})}, \quad n = 1, 2, 3, \ldots \quad . \tag{3.1.21}$$

Цей метод називають або *методом хорд*, або *методом січних*, або *методом лінійної інтерполяції*, або *методом пропорційних частин*.

Відмітимо, що метод січних, на відміну від попередніх методів, є двокроковим, тобто нове наближення x_{n+1} визначається через дві попередні ітерації x_n і x_{n-1}, а тому необхідно задавати два початкових наближення x_0 і x_1.

Зауважимо, що коли члени послідовності $\{x_n\}_{n=0}^{\infty}$ наближень кореня рівняння (3.1.1), знайдена методом січних, є збіжною до $\widehat{x}$, то $\widehat{x}$ – корінь цього рівняння, а точніше $\widehat{x} = x_*$. Справді, перейдемо в (3.1.21) до границі при $n \to \infty$. У результаті, врахувавши, що

$$f'(\widehat{x}) = \lim_{n\to\infty} \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}},$$

отримаємо

$$\widehat{x} = \widehat{x} - \frac{f(\widehat{x})}{f'(\widehat{x})} \quad \Rightarrow \quad f(\widehat{x}) = 0,$$

а це значить, що $\widehat{x}$ — корінь рівняння (3.1.1). Оскільки це рівняння має єдиний корінь x_* на відрізку $[a, b]$, то $\widehat{x} = x_*$.

3.2. Розв'язування систем нелінійних рівнянь

3.2.1. Локалізація розв'язків систем нелінійних рівнянь

Нехай $n \geqslant 1$ — довільне натуральне число. Нагадаємо, що

$$\mathbb{R}^n = \left\{ x = \begin{pmatrix} x_1 \\ \dots \\ x_n \end{pmatrix} \equiv (x_1, \dots, x_n)^\top \,\middle|\, x_1, \dots, x_n \in \mathbb{R} \right\}$$

— лінійний простір з однією з норм

$$\|x\|_1 := \max\{|x_1|, \dots, |x_n|\}, \quad \|x\|_2 = |x_1| + \ldots + |x_n|, \quad \|x\|_3 = \sqrt{x_1^2 + \ldots + x_n^2}.$$

Далі, якщо для нас не має принципового значення, яку з норм використовувати, будемо писати $\|\cdot\|$ замість $\|\cdot\|_l,\ l \in \{1, 2, 3\}$.

Розглянемо нелінійну систему рівнянь

$$\begin{cases} f_1(x_1, \dots, x_n) = 0 \\ \dots\dots\dots\dots\dots\dots \\ f_n(x_1, \dots, x_n) = 0 \end{cases} \quad \Leftrightarrow \quad f(x) = 0, \tag{3.2.1}$$

де $f_1, \dots, f_n$ — визначені на деякій множині $D \subset \mathbb{R}^n$ функції, а

$$f(x) := \begin{pmatrix} f_1(x_1, \dots, x_n) \\ \dots \\ f_n(x_1, \dots, x_n) \end{pmatrix}, \ x = \begin{pmatrix} x_1 \\ \dots \\ x_n \end{pmatrix} \in D, \quad \text{— векторна функція.}$$

Встановимо деякі критерії існування та єдиності розв'язку системи (3.2.1).

Теорема 3.2.1 (лема Браудера). *Нехай $f : \mathbb{R}^n \to \mathbb{R}^n$ – неперервне відображення таке, що для деякого числа $R > 0$ маємо нерівність*

$$(f(x), x)_{\mathbb{R}^n} \geqslant 0 \quad \forall x \in \mathbb{R}^n, \quad \|x\| = R.$$

Тоді система рівнянь

$$f(x) = 0 \tag{3.2.2}$$

має розв'язок x^* такий, що $\|x^*\| \leqslant R$.

Відображення $f : \mathbb{R}^n \to \mathbb{R}^n$ називають *строго монотонним*, якщо

$$(f(x^1) - f(x^2), x^1 - x^2)_{\mathbb{R}^n} > 0 \tag{3.2.3}$$

для будь-яких $x^1, x^2 \in \mathbb{R}^n$, $x^1 \neq x^2$.

Теорема 3.2.2. *Якщо відображення $f : \mathbb{R}^n \to \mathbb{R}^n$ є строго монотонним, то система рівнянь (3.2.2) не може мати більше одного розв'язку.*

Доведення. Доведемо методом від супротивного. Припустимо, що наше твердження неправильне. Тоді нехай x^* і x^{**} різні розв'язки системи (3.2.2), тобто $f(x^*) = 0$ і $f(x^{**}) = 0$. Тоді на підставі (3.2.3) маємо

$$0 = (f(x^*) - f(x^{**}), x^* - x^{**})_{\mathbb{R}^n} > 0.$$

Отримана суперечність доводить наше твердження. □

Приклад 3.2.1. Нехай

$$f(x) := \begin{pmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \end{pmatrix}, \quad x := \begin{pmatrix} x_1 \\ x_2 \end{pmatrix},$$

де

$$f_1(x_1, x_2) = g_{11}x_1 + g_{12}x_1^5 + g_{13}x_2^2 + g_{14}, \quad (x_1, x_2)^\top \in \mathbb{R}^2, \tag{3.2.4}$$

$$f_2(x_1,x_2) = g_{21}x_1^3 + g_{22}x_2 + g_{23}x_2^3 + g_{24}, \quad (x_1,x_2)^\top \in \mathbb{R}^2, \tag{3.2.5}$$

$g_{ij},\ i=1,2,\ j=1,2,3,4,$ — сталі.

Знайти умови на значення сталих $g_{ij},\ i=1,2,\ j=1,2,3,4,$ при яких система рівнянь

$$f(x) = 0 \tag{3.2.6}$$

має розв'язок, і локалізувати його.

Розв'язування. Використаємо теорему 3.2.1. Для цього, застосовуючи нерівність Коші

$$ab \leqslant \varepsilon a^2 + (4\varepsilon)^{-1}b^2, \tag{3.2.7}$$

зробимо оцінку

$$\begin{gathered}(f(x),x)_{\mathbb{R}^2} = f_1(x_1,x_2)x_1 + f_2(x_1,x_2)x_2 = \\ = g_{11}x_1^2 + g_{12}x_1^6 + g_{13}x_2^2x_1 + g_{14}x_1 + g_{21}x_1^3x_2 + g_{22}x_2^2 + g_{23}x_2^4 + g_{24}x_2 \geqslant \\ \geqslant g_{11}x_1^2 + g_{12}x_1^6 - |g_{13}|(\varepsilon_1x_1^2 + (4\varepsilon_1)^{-1}x_2^4) - \varepsilon_2x_1^2 - (4\varepsilon_2)^{-1}g_{14}^2 - |g_{21}|(\varepsilon_3x_2^2 + (4\varepsilon_3)^{-1}x_1^6) + \\ + g_{22}x_2^2 + g_{23}x_2^4 - \varepsilon_4x_2^2 - (4\varepsilon_4)^{-1}g_{24}^2 = \\ = (g_{11} - \varepsilon_1 \mid g_{13} \mid - \varepsilon_2)x_1^2 + (g_{22} - \varepsilon_3 \mid g_{21} \mid - \varepsilon_4)x_2^2 + \\ + (g_{12} - (4\varepsilon_3)^{-1} \mid g_{21} \mid)x_1^6 + (g_{23} - (4\varepsilon_1)^{-1}|g_{13}|)x_2^4 - \\ - (4\varepsilon_2)^{-1}g_{14}^2 - (4\varepsilon_4)^{-1}g_{24}^2.\end{gathered} \tag{3.2.8}$$

Нехай $g_{ij},\ i=\overline{1,2},\ j=\overline{1,4},$ задовольняють умову: *існують додатні сталі* $\varepsilon_1,\varepsilon_2,\varepsilon_3,\varepsilon_4$ *такі, що*

$$\widetilde{g}_{11} := g_{11} - \varepsilon_1 \mid g_{12} \mid - \varepsilon_2 > 0, \qquad \widetilde{g}_{22} := g_{22} - \varepsilon_3 \mid g_{21} \mid - \varepsilon_4 > 0, \tag{3.2.9}$$

$$g_{12} - (4\varepsilon_3)^{-1}|g_{21}|) \geqslant 0, \qquad g_{23} - (4\varepsilon_1)^{-1}|g_{13}|) \geqslant 0. \tag{3.2.10}$$

Тоді з (3.2.8) маємо

$$\begin{gathered}(f(x),x)_{\mathbb{R}^2} \geqslant \widetilde{g}_{11}x_1^2 + \widetilde{g}_{22}x_2^2 - \widetilde{g}_{33} \geqslant \min\{\widetilde{g}_{11},\widetilde{g}_{22}\}\big(x_1^2 + x_2^2\big) - \widetilde{g}_{33} = \\ = \min\{\widetilde{g}_{11},\widetilde{g}_{22}\}\|x\|_3^2 - \widetilde{g}_{33},\end{gathered} \tag{3.2.11}$$

де

$$\widetilde{g}_{33} := (4\varepsilon_2)^{-1}g_{14}^2 + (4\varepsilon_4)^{-1}g_{24}^2.$$

Нехай $R > 0$ — таке, що

$$\min\{\widetilde{g}_{11},\widetilde{g}_{22}\} \cdot R^2 - \widetilde{g}_{33} \geqslant 0.$$

Тоді з (3.2.11) випливає, що

$$(f(x),x)_{\mathbb{R}^2} \geqslant 0 \quad \text{при} \quad \|x\| = R. \tag{3.2.12}$$

Отже, при виконанні умов (3.2.9) і (3.2.10) нелінійна система рівнянь

$$\begin{cases} f_1(x_1,x_2) = 0, \\ f_2(x_1,x_2) = 0 \end{cases}$$

де f_1, f_2 задані в (3.2.4), (3.2.5), має і тільки один розв'язок в $\mathbb{R}^2$ й цей розв'язок $x^* = (x_1^*, x_2^*)^\top$ задовольняє оцінку $\|x^*\|_3 \leqslant R$, де число $R > 0$ вказано вище. □

Приклад 3.2.2. Знайти умови на значення сталих $g_{ij},\ i=1,2,\ j=1,2,3,4,$ при яких функція f є строго монотонною, а отже, система рівнянь (3.2.6) має не більше одного розв'язку.

Доведення. Нехай $x^1, x^2 \in \mathbb{R}^2$ — довільні фіксовані точки. Позначимо

$$h(t) := f(x^2 + t(x^1 - x^2)), \quad t \in [0, 1].$$

За теоремою Лагранжа про скінчені різниці маємо

$$f(x^1) - f(x^2) = h(1) - h(0) = h'(\theta) = \nabla f(x^2 + \theta(x^1 - x^2))(x^1 - x^2), \tag{3.2.13}$$

де

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f_1(x_1,x_2)}{\partial x_1} & \frac{\partial f_1(x_1,x_2)}{\partial x_2} \\ \frac{\partial f_2(x_1,x_2)}{\partial x_1} & \frac{\partial f_2(x_1,x_2)}{\partial x_2} \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in \mathbb{R}^2.$$

В нашому випадку маємо

$$\nabla f(x) = \begin{pmatrix} g_{11} + 5g_{12}x_1^4 & 2g_{13}x_2 \\ 3g_{21}x_1^2 & g_{22} + 3g_{23}x_2^2 \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in \mathbb{R}^2.$$

Знайдемо умови на значення $g_{ij},\ i = \overline{1,2},\ j = \overline{1,4}$, при яких $\nabla f(x)$ – додатно визначена матриця при всіх $x \in \mathbb{R}^n$. Припустимо, що

$$g_{11} > 0, \quad g_{12} > 0, \quad g_{22} > 0, \quad g_{23} > 0. \tag{3.2.14}$$

Тоді $g_{11} + 5g_{12}x_1^4 > 0$ при всіх $x = (x_1, x_2)^\top \in \mathbb{R}^2$. Далі треба, щоби виконувалася нерівність

$$6g_{13}g_{21}x_1^2x_2 - (g_{11} + 5g_{12}x_1^4)(g_{22} + 3g_{23}x_2^2) < 0. \tag{3.2.15}$$

при всіх $x = (x_1, x_2)^\top \in \mathbb{R}^2$.

За нерівністю Коші маємо

$$6g_{13}g_{21}x_1^2x_2 \leqslant 6 \mid g_{13} \mid\mid g_{21} \mid (\varepsilon_5 x_1^4 + (4\varepsilon_5)^{-1}x_2^2). \tag{3.2.16}$$

Звідки маємо

$$6g_{13}g_{21}x_1^2x_2 - (g_{11} + 5g_{12}x_1^4)(g_{22} + 3g_{23}x_2^2) \leqslant$$

$$\leqslant 6\varepsilon_5|g_{13}||g_{21}|x_1^4 + 6(4\varepsilon_5)^{-1}|g_{13}||g_{21}|x_2^2 - g_{11}g_{22} - 5g_{12}g_{22}x_1^4 -$$

$$-3g_{11}g_{23}x_2^2 - 15g_{12}g_{23}x_1^4x_2^2 = -(5g_{12}g_{22} - 6\varepsilon_5|g_{13}||g_{21}|)x_1^4 -$$

$$-(3g_{11}g_{23} - 6(4\varepsilon_5)^{-1}|g_{13}||g_{21}|)x_2^2 - g_{11}g_{22} - 15g_{12}g_{23}x_1^4x_2^2 < 0,$$

якщо

$$\begin{cases} 5g_{12}g_{22} - 6\varepsilon_5|g_{13}||g_{21}| \geqslant 0 \\ 3g_{11}g_{23} - 6(4\varepsilon_5)^{-1}|g_{13}||g_{21}| \geqslant 0 \end{cases} \tag{3.2.17}$$

для деякого $\varepsilon_5 > 0$.

Зі сказаного випливає, що при виконанні умов (3.2.14), (3.2.17) матриця $\nabla f(x)$ є додатно визначеною для будь-якої точки $x \in \mathbb{R}^2$. Отже, маємо

$$(\nabla f(x)z, z)_{\mathbb{R}^2} > 0, \quad x, z \in \mathbb{R}^2.$$

Звідси та (3.2.13), поклавши $z = x^1 - x^2$, отримаємо, що

$$(f(x^1) - f(x^2), x^1 - x^2)_{\mathbb{R}^2} > 0$$

для будь-яких $x^1, x^2 \in \mathbb{R}^n$, $x^1 \neq x^2$, тобто наше відображення $f : \mathbb{R}^2 \to \mathbb{R}^2$ є строго монотонним при виконанні умов (3.2.14), (3.2.17). □

3.2.2. Метод простої ітерації (послідовних наближень)

Запишемо (3.2.1) у вигляді

$$\begin{cases} x_1 = g_1(x_1, \dots, x_n) \\ \dots\dots\dots\dots\dots\dots \\ x_n = g_n(x_1, \dots, x_n) \end{cases} \quad \Leftrightarrow \quad x := g(x). \tag{3.2.18}$$

Побудуємо послідовність

$$x^0 = \begin{pmatrix} x_1^0 \\ \dots \\ x_n^0 \end{pmatrix}, \; x^1 = \begin{pmatrix} x_1^1 \\ \dots \\ x_n^1 \end{pmatrix}, \; \dots, \; x^k = \begin{pmatrix} x_1^k \\ \dots \\ x_n^k \end{pmatrix}, \; x^{k+1} = \begin{pmatrix} x_1^{k+1} \\ \dots \\ x_n^{k+1} \end{pmatrix}, \dots, \tag{3.2.19}$$

за правилом:
$x^0 \in D$ — початкове наближення, яке довільно вибирають, а решту членів знаходять за рекурентним співвідношенням

$$x^{k+1} := g(x^k) \quad \Leftrightarrow \quad \begin{cases} x_1^{k+1} := g_1(x_1^k, \dots, x_n^k) \\ \dots\dots\dots\dots\dots\dots\dots \\ x_n^{k+1} := g_n(x_1^k, \dots, x_n^k) \end{cases}, \quad k \in \mathbb{N} \cup \{0\}.$$

Цей процес називають *методом простих ітерацій.*

Теорема 3.2.3. *Нехай D – замкнена множина в $\mathbb{R}^n$ і відображення $g = \begin{pmatrix} g_1 \\ \dots \\ g_n \end{pmatrix}$ переводить множину D саму в себе, причому для якого-небудь $l \in \{1, 2, 3\}$ маємо*

$$\|g(x) - g(y)\|_l \leqslant q\|x - y\|_l \quad \text{для будь-яких } x, y \in D, \tag{3.2.20}$$

де $q \in (0, 1)$ — деяке число.

Тоді система (3.2.18) має і тільки один розв'язок на множині D, причому він є границею послідовності (3.2.19).

Приклад 3.2.3. Записати формули для знаходження послідовності наближень розв'язку заданої системи рівнянь

$$\begin{cases} 4x_1 - \sin x_2 + 1 &= 0, \\ \cos x_1 - 2x_2 + 3 &= 0 \end{cases}$$

методом простої ітерації та довести її збіжність.

Розв'язування. Дану систему запишемо у вигляді

$$\begin{cases} x_1 &= \dfrac{1}{4}\sin x_2 - \dfrac{1}{4}, \\ x_2 &= \dfrac{1}{2}\cos x_1 + \dfrac{3}{2}. \end{cases}$$

Тут

$$\begin{cases} g_1(x_1, x_2) = \dfrac{1}{4}\sin x_2 - \dfrac{1}{4}, \\ g_2(x_1, x_2) = \dfrac{1}{2}\cos x_1 + \dfrac{3}{2}. \end{cases}$$

Маємо

$$-1 \leqslant \sin x_2 \leqslant 1 \quad \left| \times \frac{1}{4} \right| - \frac{1}{4} \quad \Rightarrow \quad -\frac{1}{2} \leqslant \frac{1}{4}\sin x_2 - \frac{1}{4} \leqslant 0,$$

$$-1 \leqslant \cos x_1 \leqslant 1 \quad \left| \times \frac{1}{2} \right| + \frac{3}{2} \quad \Rightarrow \quad 1 \leqslant \frac{1}{2}\cos x_1 + \frac{3}{2} \leqslant 2.$$

Звідси випливають нерівності

$$-\frac{1}{2} \leqslant g_1(x_1,x_2) \leqslant 0, \quad 1 \leqslant g_2(x_1,x_2) \leqslant 2.$$

Покладемо

$$D := \left\{(x_1, x_2)^\top \in \mathbb{R}^2 \quad \Big| -\frac{1}{2} \leqslant x_1 \leqslant 0,\ 1 \leqslant x_2 \leqslant 2\right\}.$$

Отже, відображення

$$D \ni x := \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \mapsto \begin{pmatrix} g_1(x_1, x_2) \\ g_2(x_1, x_2) \end{pmatrix} =: g(x)$$

переводить множину D в себе.

Перевіримо виконання умови

$$\|g(x) - g(y)\|_2 \leqslant q\|x - y\|_2, \quad x, y \in D.$$

Використовуючи формулу Лагранжа про скінчений приріст, маємо

$$g(x) - g(y) = \begin{pmatrix} g_1(x_1, x_2) - g_1(y_1, y_2) \\ g_2(x_1, x_2) - g_2(y_1, y_2 \end{pmatrix} = \begin{pmatrix} \frac{1}{4}(\sin x_2 - \sin y_2) \\ \frac{1}{2}(\cos x_1 - \cos y_1) \end{pmatrix} =$$

$$= \begin{pmatrix} \frac{1}{4}(\cos \xi_2)(x_2 - y_2) \\ \frac{1}{2}(-\sin \xi_1)(x_1 - y_1) \end{pmatrix},$$

де ξ_1, ξ_2 — деякі дійсні числа.

Звідси отримуємо

$$\|g(x) - g(y)\|_2 = |g_1(x_1, x_2) - g_1(y_1, y_2)| + |g_2(x_1, x_2) - g(y_1, y_2)| \leqslant$$

$$\leqslant \frac{1}{4}|x_2 - y_2| + \frac{1}{2}|x_1 - y_1| \leqslant \frac{1}{2}(|x_1 - y_1| + |x_2 - y_2|) = \frac{1}{2}\|x - y\|_2.$$

Отже, виконуються всі умови теореми 3.2.3, які є достатніми для збіжності простого ітераційного процесу:

$$\begin{cases} x_1^{k+1} = \frac{1}{4}\sin x_2^k - \frac{1}{4}, \\ x_2^{k+1} = \frac{1}{2}\cos x_1^k + \frac{3}{2}, \end{cases} \qquad k = 0, 1, 2, 3 \dots,$$

де x_1^0, x_2^0 – довільно вибрані числа за умов $\begin{cases} -\dfrac{1}{2} \leqslant x_1^0 \leqslant 0, \\ 1 \leqslant x_2^0 \leqslant 2. \end{cases}$ □

3.2.3. Метод Ньютона

Розглянимо систему нелінійних рівнянь

$$f(x) = 0, \tag{3.2.21}$$

де

$$f(x) = \begin{pmatrix} f_1(x_1, \dots, x_n) \\ \dots\dots\dots \\ f_n(x_1, \dots, x_n) \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ \dots \\ x_n \end{pmatrix} \equiv (x_1, \dots, x_n)^\top \in D,$$

D – область в $\mathbb{R}^n$, тобто (3.2.21) еквівалентно системі

$$\begin{cases} f_1(x_1, \dots, x_n) = 0, \\ \dots\dots\dots \\ f_n(x_1 \dots, x_n) = 0. \end{cases} \tag{3.2.22}$$

Припускаємо, що $f_i \in C^1(D)$, $i = \overline{1,n}$, і матриця Якобі

$$\nabla f(x) := \begin{pmatrix} \dfrac{\partial f_1(x_1,\dots,x_n)}{\partial x_1} & \dots & \dfrac{\partial f_1(x_1,\dots,x_n)}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_n(x_1,\dots,x_n)}{\partial x_1} & \dots & \dfrac{\partial f_n(x_1,\dots,x_n)}{\partial x_n} \end{pmatrix}$$

невироджена в кожній точці $x = (x_1,\dots,x_n)^\top \in D$.

За формулою Тейлора маємо

$$f(x'') = f(x') + \nabla f(x')(x'' - x') + o(\|x'' - x'\|), \qquad x', x'' \in D, \tag{3.2.23}$$

де $o(t)$ – функція така, що $\dfrac{o(t)}{t} \to 0$ при $t \to 0$. Звідси випливає, що якщо послідовність

$$x^0, x^{1,}, ..., x^k, ...$$

збігається до x^* і

$$f(x^k) + \nabla f(x^k)(x^{k+1} - x^k) = 0, \quad k \in \mathbb{N} \cup \{0\}, \tag{3.2.24}$$

то x^* – розв'язок системи (3.2.21). Справді, за формулою (3.2.23) маємо

$$f(x^{k+1}) = f(x^k) + \nabla f(x^k)(x^{k+1} - x^k) + o(\|x^{k+1} - x^k\|), \quad k \in \mathbb{N}. \tag{3.2.25}$$

Якщо в (3.2.25) врахувати (3.2.24), то отримаємо $f(x^{k+1}) = o(\|x^{k+1} - x^k\|)$. Перейшовши до границі при $k \to \infty$, то отримаємо $f(x^*) = 0$. Звідки можемо робити висновок, що ітераційний процес знаходження розв'язку (3.2.21), тобто послідовності $x^0, x^{1,}, ..., x^k, ...$, що збігається до розв'язку x^* системи (3.2.21), який заданий так:

$$f(x^k) + \nabla f(x^k)(x^{k+1} - x^k) = 0, \tag{3.2.26}$$

звідки

$$x^{k+1} = x^k - [\nabla f(x^k)]^{-1} f(x^k), \qquad k = 0, 1, 2, ... \tag{3.2.27}$$

Ввівши позначення $z^k := x^{k+1} - x^k$ співвідношення (3.2.26) запишемо у вигляді таких двох співвідношень:

$$\nabla f(x^k) \cdot z^k = -f(x^k), \tag{3.2.28}$$

$$x^{k+1} = x^k + z^k, \qquad k = 1, 2, ... \ . \tag{3.2.29}$$

Отож, ***метод Ньютона*** полягає у знаходженні x^{k+1} в два етапи:

1) *розв'язуємо систему лінійних алгебраїчних рівнянь* (3.2.28) *стосовно* z^k*;*

2) *знаходимо* x^{k+1} *за формулою* (3.2.29).

На практиці також використовують модифікацію методу Ньютона, яка полягає у заміні співвідношення (3.2.28) на співвідношення

$$\nabla f(x^0) z^k = -f(x^k).$$

Приклад 3.2.4. Записати рекурентні співвідношення (3.2.28) для знаходження наближень розв'язку системи рівнянь

$$\begin{cases} x_1^2 - 2x_2 + 3 = 0 \\ 2x_1 + 3x_2^2 - 1 = 0 \end{cases} . \tag{3.2.30}$$

Розв'язування. Приймемо

$$f_1(x_1,x_2) := x_1^2 - 2x_2 + 3, \quad f_2(x_1,x_2) := 2x_1 + 3x_2^2 - 1, \quad (x_1,x_2)^\top \in \mathbb{R}^2.$$

Тоді для векторної функції $f(x) := \begin{pmatrix} f_1(x_1,x_2) \\ f_1(x_1,x_2) \end{pmatrix}$ маємо маємо

$$\nabla f(x) := \begin{pmatrix} \frac{\partial f_1(x_1,x_2)}{\partial x_1} & \frac{\partial f_1(x_1,x_2)}{\partial x_2} \\ \frac{\partial f_2(x_1,x_2)}{\partial x_1} & \frac{\partial f_2(x_1,x_2)}{\partial x_2} \end{pmatrix} = \begin{pmatrix} 2x_1 & -2 \\ 2 & 6x_2 \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in \mathbb{R}^2.$$

Оскільки

$$x^k = \begin{pmatrix} x_1^k \\ x_2^k \end{pmatrix}, \quad z^k = \begin{pmatrix} z_1^k \\ z_2^k \end{pmatrix}, \quad \nabla f(x^k) := \begin{pmatrix} 2x_1^k & -2 \\ 2 & 6x_2^k \end{pmatrix},$$

$$f(x^k) := \begin{pmatrix} (x_1^k)^2 - 2x_2^k + 3 \\ 2x_1^k + 3(x_2^k)^2 - 1 \end{pmatrix},$$

то співвідношення (3.2.28) для системи (3.2.30) запишеться у вигляді

$$\begin{pmatrix} 2x_1^k & -2 \\ 2 & 6x_2^k \end{pmatrix} \begin{pmatrix} z_1^k \\ z_2^k \end{pmatrix} = \begin{pmatrix} -(x_1^k)^2 + 2x_2^k - 3 \\ -2x_1^k - 3(x_2^k)^2 + 1 \end{pmatrix}, \quad k = 0, 1, 2, \ldots .$$

□

Вправи для самостійної роботи

1. Методом табуляції функції локалізувати корені рівняння

$$e^x - x - 2 = 0$$

і знайти перші три наближення (не рахуючи початкового) найбільшого з коренів методами бісекції та Ньютона (заокруглення проводити з точністю $0,01$).

2. Графічним методом локалізувати корені рівняння

$$\sin x - 2x - 1 = 0$$

і знайти перші три наближення (не рахуючи початкового) найменшого з коренів методами простої ітерації та січних (заокруглення проводити з точністю $0,01$).

3. Використовуючи
а) формулу Маклорена
і
б) метод Ньютона
знайти верхні та нижні межі для від'ємних і додатних коренів рівняння

$$8x^3 - 12x^2 - 2x + 3 = 0.$$

в) Методом Штурма локалізувати всі корені даного рівняння.

4. Записати формули для знаходження послідовності наближень розв'язку заданої системи рівнянь

$$\begin{cases} 8x_1 - \cos(x_2 - \pi/3) + 3 &= 0 \\ \sin(x_1 + \pi/4) - 4x_2 + 3 &= 0 \end{cases}$$

методом простої ітерації та довести її збіжність.

5. Записати формули для знаходження послідовності наближень розв'язку заданої системи рівнянь

$$\begin{cases} x_1^2 - 2e^{x_2} + 3 = 0 \\ \ln x_1 + 3x_2^2 - 1 = 0 \end{cases}$$

методом Ньютона.

3.3. Лабораторний практикум з розв'язування нелінійних рівнянь

Розглянемо застосування на практиці чисельних методів, які були розглянуті раніше в цьому розділі, до чисельного розв'язування рівняння

$$f(x) = 0, \tag{3.3.1}$$

де $f(x)$, $x \in \langle c, d \rangle$, – неперервна функція, $-\infty \leqslant c < d \leqslant +\infty$. Зазначимо, що приклади програмної реалізації відповідних методів, які будуть наведені далі, можна завантажити у вигляді готових до використання ноутбуків за **посиланням**.

3.3.1 Локалізація розв'язків рівняння графічним методом

Побудову графіка функції `f` виконуватимемо за допомогою функції `plot_graphics`, яка реалізована на основі бібліотеки `matplotlib`. Найпростіше це робити в інтерактивному режимі, який встановлють командою `%matplotlib widget`. У цьому випадку після виконання функції `plot_graphics`, яка побудує потрібний графік на графічній панелі, треба виконати такі дії:

1. активізувати режим `Zoom to rectangle` за допомогою іконки меню на графічній панелі (яка містить зображення прямокутника);
2. на графічній панелі визначити мишкою якомога меншу прямокутну область, яка містить точку перетину графіка функції з віссю абсцис.

Виконання пункту 2 автоматично приведе до відображення визначеної прямокутної області на всю графічну панель, що дає змогу локалізувати нуль функції на відрізку `[a, b]` меншої довжини. Виконуючи цей пункт кілька разів, можна досягти локалізації нуля на відрізку потрібної довжини. Зазначимо, що таке масштабування графіка відбуватиметься на тій самій графічній панелі.

Якщо режим `widget` не встановлено, то функцію `plot_graphics` виконують кілька разів, послідовно уточнюючи координати відрізку `[a, b]`, поки не локалізують розв'язок з потрібною точністю.

Пояснення до використання програмного коду

- Підготувати потрібні функції :
  1. виконати комірку, де виконується підготовка середовища (в JupyterLab розкоментувати рядок з `%matplotlib widget`, щоб мати змогу масштабувати графік при локалізації у п.4);
  2. виконати комірку з визначенням функції `plot_graphics`;
  3. виконати комірку з визначенням функції `f` (цю комірку виконувати кожного разу після внесення змін у визначення функції `f`).
- Локалізувати потрібний корінь рівняння :
  1. виконати комірку, в якій задається відрізок;
  2. виконати комірку з викликом функції `plot_graphics` для побудови графіка;
  3. уточнити (звузити) відрізок `[a, b]`, щоб на ньому знаходився лише один корінь рівняння;
  4. якщо в середовищі доступний інтерактивний режим (наприклад, в JupyterLab), то активізувати `Zoom to rectangle` (вибрати в меню на графічній панелі іконку з прямокутником) і виділити мишкою прямокутну область з потрібним фрагментом графіка;
  5. пункти 1 - 3 можна повторити для точнішої локалізації потрібного розв'язку.

Програмна реалізація графічного методу

Підготовка середовища

[1]:
```
%matplotlib widget
import numpy as np
import matplotlib.pyplot as plt
```

`plot_graphics` – **функція для побудови графіка функції** `f` **на відрізку** `[a,b]` **за значеннями в** `n` **точках**

[3]:
```
def plot_graphics(f, a, b, n):
    """фуункція для побудови графіка функції f
       на відрізку [a,b] за значеннями в n точках
    """
    xarr = np.linspace(a, b, n)
    y=f(xarr)
    fig = plt.figure()
    ax = fig.gca()
    ax.plot(xarr,y)
    ax.axhline(color="grey", ls="--", zorder=-1)
    ax.axvline(color="grey", ls="--", zorder=-1)
    ax.set_xlim(a,b)
    ax.set_xlabel('x')
    ax.set_ylabel('f(x)')
    plt.show()
```

Обчислювальні експерименти

Приклад 1. Локалізувати другий додатній корінь рівняння

$$sin(x^2 - 2x) = 0, \ x \in \mathbb{R}. \tag{3.3.2}$$

Виконаємо комірку, в якій визначено функцію f:

[3]:
```
def f(x):
    """функція лівої частини рівняння (3.3.2)"""
    return np.sin(x*x-2*x)
```

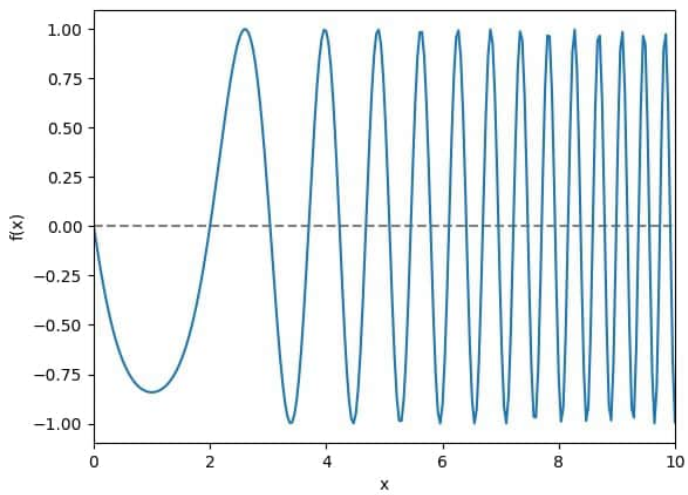
Тепер задамо початкові координати відрізка $[a, b]$

[4]:
```
a=0
b=10
```

і викличемо функцію `plot_graphics` :

[5]:
```
plot_graphics(f, a, b, 256)
```

Результатом її виконання буде поява графічної панелі з графіком функції



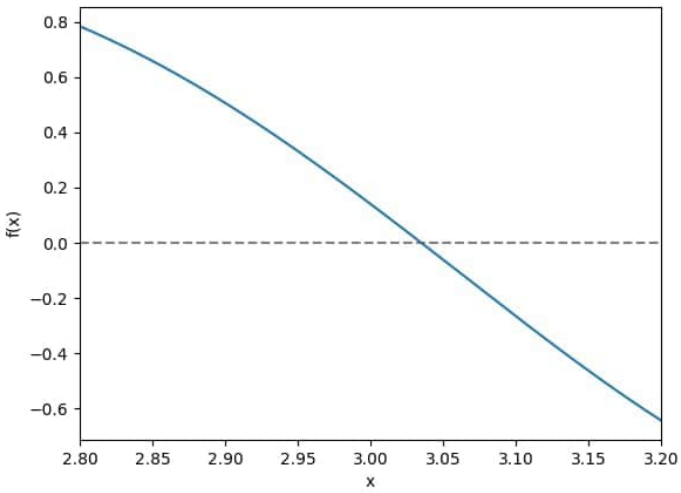
В інтерактивному режимі на графічній панелі використовуємо в меню іконку `Zoom to rectangle` і кілька разів звужуємо відрізок, на якому автоматично перемальовуватиметься графік. Причому буде використовуватися та сама графічна панель.

Без використання автоматичного масштабування можна кілька разів явно (використовуючи нові комірки) задати нові значення кінців відрізка і повторно будувати графік:

1-ий крок:

[6]:
```
a=2.8
b=3.2
```

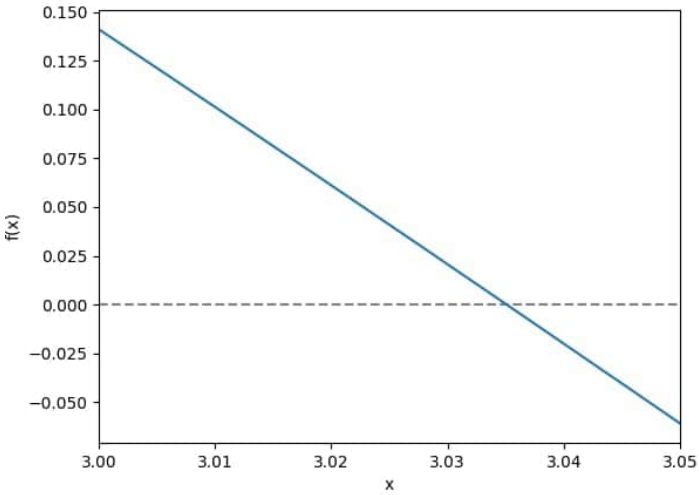
[7]:
```
plot_graphics(f, a, b, 256)
```



2-ий крок:

[8]:
```
a=3.0
b=3.05
```

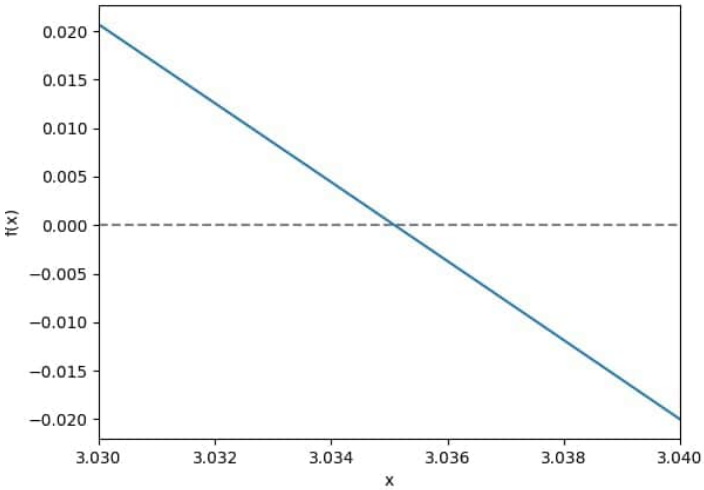
[9]:
```
plot_graphics(f, a, b, 256)
```



3-ій крок:

[10]:
```
a=3.03
b=3.04
```

[11]:
```
plot_graphics(f, a, b, 256)
```



Зауважимо, що на кожному кроці побудова графіка виконувалася з нової комірки на новій графічній панелі. Такий прийом може виявитися зручним для порівняння графіків, отриманих з різним масштабом.

Висновок. Другий додатній корінь рівняння знаходиться на відрізку $[3.034, 3.036]$.

Приклад 2. Локалізувати другий додатній корінь рівняння

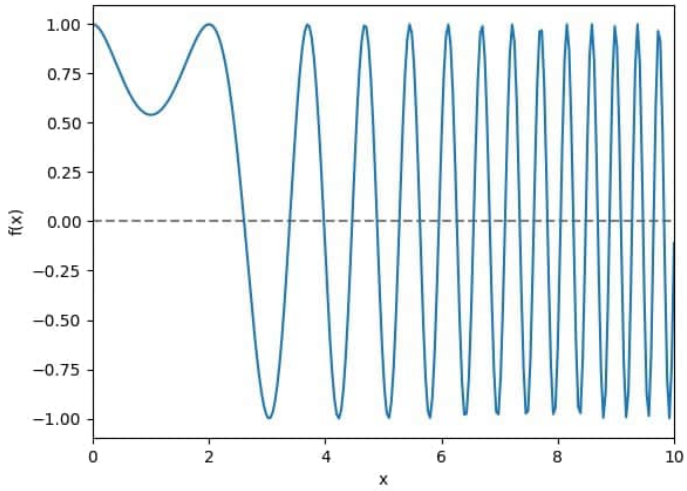
$$cos(x^2 - 2x) = 0,\ x \in \mathbb{R}. \tag{3.3.3}$$

Відмінність від попереднього прикладу є лише у визначенні функції лівої частини рівняння (3.3.3). Зазначимо, що для зручності для неї використано інший ідентифікатор "fcs". За умови, що активовано інтерактивний режим, подібно попередньому прикладу визначимо згадану функцію:

[12]:
```
def fcs(x):
    """функція лівої частини рівняння (3.3.3)"""
    return np.cos(x*x-2*x)
```

і побудуємо графік

[13]:
```
a=0
b=10
plot_graphics(fcs, a, b, 256)
```



Далі за допомогою `Zoom to rectangle` доб'ємося зображення у потрібному масштабі тієї області, де графік перетинає вісь абсцис.

Висновок. Другий додатній корінь рівняння знаходиться на відрізку [3.389, 3.390].

Дослідження графіків доцільно завершувати наступною командою, яка деактивує утворені раніше графічні панелі:

[14]:
```
plt.close('all')
```

3.3.2 Застосування методу бісекції

Нехай рівняння (3.3.1) має і тільки один розв'язок на відрізку $[a, b]$, причому

$$f(a)f(b) < 0. \tag{3.3.4}$$

Алгоритм методу бісекції, розглянутий у підрозділі 3.1.2, реалізовано функцією `bisection`. Зауважимо, що запропонована реалізація не передбачає зберігання послідовності відрізків $[x_{2k}, x_{2k+1}]$, $k \in \mathbb{N}$, про які йдеться у підрозділі 3.1.2. Натомість на кожному кроці оновлюються координати відрізка $[a, b]$, на якому буде локалізований розв'язок рівняння. З цією метою виконується перевірка, чи чергова точка поділу відрізка не перетворює в нуль ліву частину рівняння (3.3.1), а також визначається, на якому з відрізків, отриманих після поділу, виконується умова (3.3.4).

Ноутбук з усім програмним кодом, необхідним для розв'язування рівняння (3.3.1), можна завантажити з репозиторію за **посиланням**. Цей ноутбук також містить функцію `plot_graphics`, потрібну для графічної локалізації розв'язків.

Пояснення до використання програмного коду

- Підготувати середовище і потрібні функції:
  1. виконати комірку для підготовки середовища
  2. виконати комірки, де **визначені** функції `bisection` і `plot_graphics`
- Обчислити чисельні розв'язки конкретного рівняння:
  1. виконати комірку, де **визначена** функція `f` лівої частини рівняння;
  2. локалізовати потрібний розв'язок (як це продемонстровано в підрозділі 3.3.1) і задати координати відповідного відрізка;
  3. задати точність `eps` чисельного розв'язку;
  4. виконати комірку, де є **виклик** функції `bisection`.
  5. Для знаходження іншого розв'язку треба спочатку виконати підпункт 2, щоб локалізувати його, а потім виконати підпункт 4.

Програмна реалізація методу

`bisection` – **функція для знаходження локалізованого чисельного розв'язку рівняння методом бісекції.**

[2]:
```
def bisection(f,a,b,eps):
    """ знаходження методом бісекції чисельного розв'язку рівняння (1),
        де f -- неперервна на відрізку [a,b] функція,
        на цьому відрізку рівняння має і тільки один розв'язок,
        eps -- задана точність
    """
    k=0
    ba=np.abs(b-a)
    if ba < eps:
        return (a+b)/2, k+1
    while ba > eps:
        fa=f(a)
        k+=1
        x=(a+b)/2
        fx=f(x)
        if fx==0 :
            return x, k
        if fa*fx < 0:
            b=x
        else:
            a=x
        ba=np.abs(b-a)
    return (a+b)/2, k+1
```

Обчислювальні експерименти

Приклад 1. Обчислити методом бісекції другий додатній розв'язок рівняння (3.3.2) з точністю $eps = 10^{-5}$.

Легко переконатися, що першими додатніми розв'язками є $x_1 = 2.0$, а також $x_2 = 1+\sqrt{1+\pi}$ і $x_3 = 1+\sqrt{1+2\pi}$. Виконуючи обчислення, отримаємо $x_2 = 3.035090330572526$ і $x_3 = 3.698737724785346$ з точністю 10^{-15}. Далі називатимемо їх аналітичними розв'язками і збережемо обчислені значення для подальшого аналізу похибок чисельних розв'язків:

[3]:
```
x_1=2.0
x_2=3.035090330572526
x_3=3.698737724785346
```

Нагадаємо, що для визначення функції правої частини зазначеного рівняння треба виконати комірку

[4]:
```
def f(x):
    """функція лівої частини рівняння (3.3.2)"""
    return np.sin(x*x-2*x)
```

Також виконаємо комірку, де встановлюється значення координат відрізка, на якому локалізовано шуканий розв'язок (у підрозділі 3.3.1 вже було встановлено, що це відрізок $[3.034, 3.036]$):

[5]:
```
a=3.034
b=3.036
```

Виконання наступної комірки зумовить виклик функції `bisection`, яка виконає обчислення шуканого чисельного розв'язку:

[6]:
```
eps=0.00001
x=bisection(f, a, b, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps}, кількість ітерацій␣
↪k={k}.")
```

```
Розв'язок рівняння x=3.03508984375 з точністю eps=1e-05, кількість ітерацій k=9.
```

[7]:
```
print(f"Відносна похибка чисельного розв'язку delta={np.abs(x_2-x)/x_2}")
```

```
Відносна похибка чисельного розв'язку delta=1.6039803540187618e-07
```

Зазначимо, що згідно теорії, викладеної в підрозділі 3.1.2, чисельний розв'язок рівняння (3.3.1), отриманий методом бісекції, збігається до точного розв'язку. Уявлення про характер цієї збіжності можна отримати, якщо задавати менші значення параметра `eps`. Для обчислення чисельного розв'язку з іншою точністю достатньо змінити ініціалізатор змінної *eps* у попередній комірці і знову виконати цю комірку. Або можна попереньо згадану комірку скопіювати (відзначити збоку курсором згадану комірку та послідовно натиснувши клавіші з літерами *c* і *v*), а потім вже з нею виконати зазначені дії:

[8]:
```
eps=0.0000001
x=bisection(f, a, b, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps}, кількість ітерацій␣
↪k={k}.")
```

```
Розв'язок рівняння x=3.0350903015136717 з точністю eps=1e-07, кількість ітерацій␣
↪k=16.
```

[9]:
```
print(f"Відносна похибка чисельного розв'язку delta={np.abs(x_2-x)/x_2}")
```

```
Відносна похибка чисельного розв'язку delta=9.57429640268604e-09
```

[10]:
```
eps = 0.000000001
x,k = bisection(f, a, b, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps}, кількість ітерацій␣
↪k={k}.")
```

```
Розв'язок рівняння x=3.035090330600738 з точністю eps=1e-09, кількість ітерацій
k=22.
```

[11]:
```
print(f"Відносна похибка чисельного розв'язку delta={np.abs(x_2-x)/x_2}")
```

```
Відносна похибка чисельного розв'язку delta=9.295307964700006e-12
```

Приклад 2. Обчислити методом бісекції другий додатній розв'язок рівняння (3.3.3) з точністю $eps = 10^{-5}$.

Легко переконатися, що $x_2 = 1 + np.sqrt(1 + 3 * np.pi/2)$ і $x_3 = 1 + np.sqrt(1 + 5 * np.pi/2)$. Після обчислень матимемо $x_2 = 3.390060455382811$ і $x_3 = 3.975564086685831$ з точністю 10^{-15}. Збережемо ці значення:

[12]:
```
x_2=3.390060455382811
x_3=3.975564086685831
```

Якщо продовжуємо обчислення у тому ж ноутбуці, то спершу виконаємо комірку для визначення функції, що задає праву частину рівняння (3.3.3):

[13]:
```
def fcs(x):
    """функція лівої частини рівняння (3.3.3)"""
    return np.cos(x*x-2*x)
```

Далі задаємо потрібний інтервал (див. Приклад 2 у підрозділі 3.3.1)

[14]:
```
a=3.389
b=3.390
```

і обчислюємо наближення шуканого розв'язку:

[15]:
```
eps=0.00001
x, k=bisection(fcs, a, b, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps}, кількість ітерацій␣
↪k={k}.")
```

```
Розв'язок рівняння x=3.3899960937499998 з точністю eps=1e-05, кількість ітерацій␣
↪k=8.
```

[16]:
```
print(f"Відносна похибка чисельного розв'язку delta={np.abs(x_2-x)/x_2}")
```

```
Відносна похибка чисельного розв'язку delta=1.8985393817718306e-05
```

Можемо, як і в попередньому прикладі, поекспериментувати зі значенням параметра eps:

[17]:
```
eps=0.0000001
x=bisection(fcs, a, b, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps}, кількість ітерацій␣
↪k={k}.")
```

```
Розв'язок рівняння x=3.389999969482422 з точністю eps=1e-07, кількість ітерацій␣
↪k=15.
```

[18]:
```
print(f"Відносна похибка чисельного розв'язку delta={np.abs(x_2-x)/x_2}")
```

```
Відносна похибка чисельного розв'язку delta=1.7842130305719424e-05
```

[19]:
```
eps = 0.000000001
x,k = bisection(f, a, b, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps}, кількість ітерацій␣
↪k={k}.")
```

```
Розв'язок рівняння x=3.3899999995231633 з точністю eps=1e-09, кількість ітерацій
k=21.
```

[20]:
```
print(f"Відносна похибка чисельного розв'язку delta={np.abs(x_2-x)/x_2}")
```

```
Відносна похибка чисельного розв'язку delta=1.7833268888100083e-05
```

Приклад 3. Нехай у рівнянні (3.1.1) $f(x) = x - 1$. Обчислити методом бісекції розв'язок цього рівняння з точністю $eps = 10^{-5}$.

Метою цього прикладу є продемонструвати особливість реалізації методу бісекції, а саме потребу в перевірці, чи на якомусь кроці точка поділу відрізка навпіл не співпаде з розв'язком рівняння. Виконаємо послідовно наступні комірки:

[21]:
```
def flnr(x):
    """функція лівої частини рівняння (1)"""
    return x-1
```

Далі задаємо відрізок, де локалізований (досить приблизно) розв'язок

[22]:
```
a=0
b=4
```

і обчислюємо чисельний розв'язок:

[23]:
```
eps=0.00001
x=bisection(flnr, a, b, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps}, кількість ітерацій␣
↪k={k}.")
```

```
Розв'язок рівняння x=1.0 з точністю eps=1e-05, кількість ітерацій k=2.
```

Як бачимо, за рахунок зазначеної вище перевірки чисельний розв'язок буде отримано вже на другій ітерації, хоч при цьому довжина отриманого після поділу відрізка рівна 1. Легко переконатися безпосереднім обчисленням, що при відсутності такої перевірки матимемо неправильний результат.

Висновок. Оскільки швидкість збіжності методу бісекції є невисокою, доцільно попередньо локалізувати шуканий розв'язок рівняння на якомога короткому відрізку.

3.3.3 Застосування методу простої ітерації

Нехай рівняння (3.3.1) має і тільки один розв'язок x_* на відрізку $[a, b]$.

Запишемо це рівняння у вигляді

$$x = \varphi(x), \tag{3.3.5}$$

де

$$\varphi(x) := x + \rho(x)f(x), \quad x \in [a, b], \tag{3.3.6}$$

а $\rho : [a, b] \to \mathbb{R}$ – довільна неперервна функція, яка не має нулів на $[a, b]$. Зокрема, може бути $\rho(x) = 1$, $x \in [a, b]$.

Метод простої ітерації визначається так:

- задаємо початкове значення $x_0 \in [a, b]$ і
- знаходимо послідовні наближення $x_1, x_2, x_3, \ldots$ розв'язку рівняння (3.3.5) за формулою

$$x_{n+1} = \varphi(x_n), \quad n = 0, 1, 2, \ldots . \tag{3.3.7}$$

Нехай функція $\varphi : [a, b] \to \mathbb{R}$ є стискуючим відображенням, тобто задовольняє умову Ліпшиця

$$|\varphi(x'') - \varphi(x')| \leqslant q|x'' - x'|, \quad x',\ x'' \in [a, b], \tag{3.3.8}$$

зі сталою $q \in (0, 1)$, та x_0 і $d > 0$ такі, що $[x_0 - d, x_0 + d] \subset [a, b]$ і

$$|\varphi(x_0) - x_0| \leqslant (1 - q)d. \tag{3.3.9}$$

Тоді за теоремою 3.1.7 рівняння (3.3.5) має на відрізку $[x_0 - d, x_0 + d]$ єдиний розв'язок x_* і він є границею послідовності (3.3.7).

Для похибки $x_n - x_*$ маємо оцінку

$$|x_n - x_*| \leqslant q^n|b - a|. \tag{3.3.10}$$

тобто метод простої ітерації збігається зі швидкістю геометричної прогресії із знаменником q.

Пояснення до використання програмного коду

- Підготувати середовище і потрібні функції:
  1. виконати комірку для підготовки середовища
  2. виконати комірки, де **визначені** функції `simple_iteration` і `plot_graphics`
- Обчислити чисельні розв'язки конкретного рівняння:
  1. виконати комірки, де **визначені** функції `f` і `rho`
  2. локалізовати потрібний розв'язок (як це продемонстровано в підрозділі 3.3.1) і задати координати відповідного відрізка;
  3. задати точність `eps` чисельного розв'язку і початкове наближення `x0`;
  4. виконати комірку, де є **виклик** функції `simple_iteration`.
  5. Для знаходження іншого розв'язку треба спочатку виконати підпункт 2, щоб локалізувати його, а потім виконати підпункт 4.

Програмна реалізація методу

`simple_iteration` – **функція, яка реалізує метод простої ітерації**

[3]:
```
def simple_iteration(f,rho,a,b, x0, eps):
    """ знаходження методом простої ітерації чисельного розв'язку рівняння
        де f і rho-- неперервні функції на відрізку [a, b],
        x0 -- початкове наближення
```

```
        eps -- задана точність
    """

    x_prev=x0
    k=1
    x_new = x_prev + rho(x_prev)*f(x_prev)

    while np.abs(x_new-x_prev) > eps:
        k+=1
        x_prev = x_new
        x_new = x_prev + rho(x_prev)*f(x_prev)
    return x_new,k
```

Обчислювальні експерименти
Знаходження чисельних розв'язків методом простої ітерації продемонструємо на прикладах.

Приклад 1. Нехай у рівнянні (3.3.1) $f(x) = sin(x^2 - 2x)$. Обчислити методом простої ітерації другий додатній розв'язок цього рівняння з точністю $eps = 10^{-5}$.

У попередньому підрозділі було знайдено значення перших додатніх розв'язків $x_1 = 2.0$, $x_2 = 3.035090330572526$ і $x_3 = 3.698737724785346$ з точністю 10^{-15}. Як і раніше, використаємо ці значення для подальшого аналізу похибок чисельних розв'язків.

Задамо функції `f` і `rho`, які фігурують у виразі (3.3.6):

[5]:
```
def f(x):
    """функція лівої частини рівняння """
    return np.sin(x*x-2*x)
```

[6]:
```
def rho(x):
    """функція \rho в рівнянні (3.3.6)"""
    return 1
```

Вважаючи, що шуканий корінь знаходиться на відрізку $[3.034, 3.036]$ (це вже було встановлено у підрозділі 3.3.1), задамо значення координат цього відрізка:

[15]:
```
a=3.034
b=3.036
```

Тепер задамо бажану точність розв'язків і знайдемо чисельні розв'язки при різних початкових наближеннях:

[16]:
```
eps=0.0001
x0=3.034
x, k = simple_iteration(f, rho, a, b,x0, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps} за k={k} ітерацій")
```

```
Розв'язок рівняння x=3.0351201307008653 з точністю eps=0.0001 за k=254302
ітерацій
```

[17]:
```
x0=3.035
x, k = simple_iteration(f, rho, a, b,x0, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps} за k={k} ітерацій")
```

```
Розв'язок рівняння x=3.035142461207598 з точністю eps=0.0001 за k=65073 ітерацій
```

[19]:
```
x0=3.036
x, k = simple_iteration(f, rho, a, b,x0, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps} за k={k} ітерацій")
```

```
Розв'язок рівняння x=3.035074467154927 з точністю eps=0.0001 за k=190492
ітерацій
```

Отримані результати підтверджують очікувану залежність кількості ітерацій від початкового наближення.

Вимоги до точності чисельного розв'язку заданого рівняння можна посилити:

[18]:
```
eps=0.000001
x0=3.035
x, k = simple_iteration(f, rho, a, b,x0, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps} за k={k} ітерацій")
```

```
Розв'язок рівняння x=3.035089587774611 з точністю eps=1e-06 за k=1710633
ітерацій
```

Очевидно, що такий результат був досягнутий за більшу кількість ітерацій.

3.3.4 Застосування методу Ньютона

Нехай рівняння (3.3.1) має і тільки один розв'язок x_* на відрізку $[a, b]$ і $f(a) \cdot f(b) < 0$.

Вважаємо, що $f \in C^1([a, b])$, $f'(x) \neq 0$ для всіх $x \in [a, b]$.

Метод Ньютона визначається так:

- задаємо початкове значення $x_0 \in [a, b]$ і
- знаходимо послідовні наближення $x_1, x_2, x_3, \ldots$ розв'язку рівняння (3.3.1) за формулою

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n = 0, 1, 2, \ldots . \tag{3.3.11}$$

Якщо $f(a)f(b) < 0$, функції f', f'' неперервні і відмінні від нуля на $[a, b]$, то за теоремою 3.1.8 рівняння (3.3.1) має і тільки один розв'язок x_* та, якщо початкове $x_0 \in [a, b]$ задовольняє умову

$$f(x_0)f''(x_0) > 0, \tag{3.3.12}$$

то послідовність $\{x_n\}$, отримана методом Ньютона, збігається до розв'язку x_*.

Крім того, для довільного $n \in \mathbb{N}$ маємо оцінку відхилення x_n від x_*:

$$|x_n - x_*| \leqslant \frac{M_2}{2m_1}|x_n - x_{n-1}|^2, \tag{3.3.13}$$

де

$$M_2 := \max_{x \in [a,b]} |f''(x)|, \quad m_1 := \min_{x \in [a,b]} |f'(x)|. \tag{3.3.14}$$

Пояснення до використання програмного коду

- Підготувати середовище і потрібні функції:
  1. виконати комірку для підготовки середовища
  2. виконати комірки, де **визначені** функції `Newton_iteration` і `plot_graphics`
- Обчислити чисельні розв'язки конкретного рівняння:
  1. виконати комірки, де **визначені** функції `f` і `f_deriv`
  2. локалізовати потрібний розв'язок (як це продемонстровано в підрозділі 3.3.1) і задати координати відповідного відрізка;
  3. задати точність `eps` чисельного розв'язку і початкове наближення `x0`;
  4. виконати комірку, де є **виклик** функції `Newton_iteration`.
  5. Для знаходження іншого розв'язку треба спочатку виконати підпункт 2, щоб локалізувати його, а потім виконати підпункт 4.

Програмна реалізація методу

`Newton_iteration` – **функція, яка реалізує метод Ньютона**

[2]:
```
def Newton_iteration(f,f_deriv, a, b, x0, eps):
    """ знаходження методом Ньютона чисельного розв'язку рівняння ,
        де f -- неперервна функція на відрізку [a, b],
        f_deriv -- похідна функція на відрізку [a, b]
        x0 -- початкове наближення
        eps -- задана точність
    """
    x_prev=x0
    k=1
    x_new = x_prev - f(x_prev)/f_deriv(x_prev)
    if np.abs(x_new-x_prev)<eps:
        return x_new,k
    while np.abs(x_new-x_prev) > eps:
        k+=1
        x_prev = x_new
        x_new = x_prev - f(x_prev)/f_deriv(x_prev)
    return x_new,k
```

Обчислювальні експерименти
Знаходження чисельних розв'язків методом Ньютона продемонструємо на тих самих прикладах, які розв'язували іншими методами.

Приклад 1. Нехай у рівнянні (3.1.1) $f(x) = sin(x^2 - 2x)$. Обчислити методом Ньютона другий додатній розв'язок цього рівняння з точністю $eps = 10^{-5}$.

Задамо `f` і `f_deriv` – функції лівої частини рівняння (3.3.1) та її похідної:

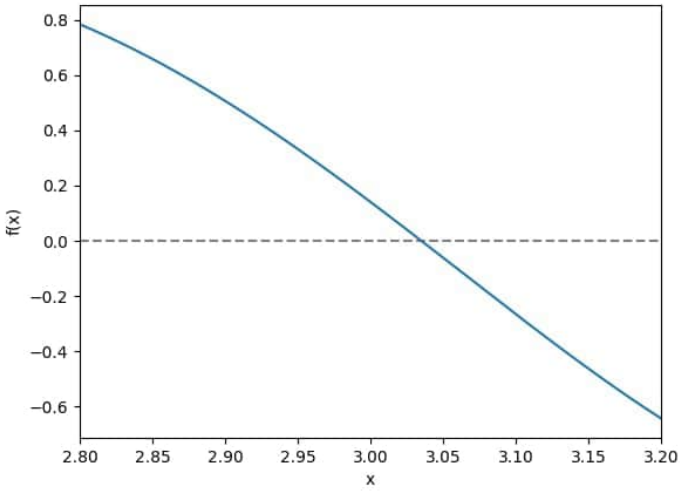
[5]:
```
def f(x):
    """функція лівої частини рівняння """
    return np.sin(x*x-2*x)

def f_deriv(x):
    """похідна функції лівої частини рівняння """
    return (2*x - 2)*np.cos(x*x-2*x)
```

У попередніх випадках був визначений досить короткий відрізок локалізації шуканого кореня і вибору початкового наближення, оскільки від цього в значній мірі залежала кількість ітерацій, необхідних для досягнення заданої точності розв'язку. Покажемо, що метод Ньютона менш чутливий до цього фактору.

Намалюємо графік заданої функції на такому відрізку:

[8]:
```
a=2.8
b=3.2
```

[9]:
```
plot_graphics(f, a, b, 256)
```



На основі графіка візьмемо $x_0 = 3$ за початкове наближення. Виконуючи наступні комірки, можна переконатися, що навіть при такому грубому початковому наближенні абсолютна та відноста похибки чисельного розв'язку швидко заникають при збільшенні кількості ітерацій:

[25]:
```
x0=3.0
eps=0.001
x, k = Newton_iteration(f, f_deriv, a, b,x0, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps} за k={k} ітерацій")
```

```
Розв'язок рівняння x=3.0350904029782013 з точністю eps=0.001 за k=2 ітерацій
```

[26]:
```
print(f"Відносна похибка чисельного розв'язку delta={np.abs(x_2-x)/x_2}")
```

```
Відносна похибка чисельного розв'язку delta=2.3856184609700732e-08
```

[27]:
```
eps=0.00001
x, k = Newton_iteration(f, f_deriv, a, b,x0, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps} за k={k} ітерацій")
```

```
Розв'язок рівняння x=3.0350903305725274 з точністю eps=1e-05 за k=3 ітерацій
```

[28]:
```
print(f"Відносна похибка чисельного розв'язку delta={np.abs(x_2-x)/x_2}")
```

```
Відносна похибка чисельного розв'язку delta=4.389548528853422e-16
```

[30]:
```
eps = 0.00000000001
x, k = Newton_iteration(f, f_deriv, a, b,x0, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps} за k={k} ітерацій")
```

```
Розв'язок рівняння x=3.035090330572526 з точністю eps=1e-11 за k=4 ітерацій
```

Як бачимо, вже за чотири ітерації усі 16 цифр отриманого чисельного розв'язку ті самі, що і цифри обчисленого раніше з точністю 10^{-15} значення аналітичного розв'язку. Виконання наступних комірок демонструє, що при точнішому початковому наближенні чисельний розв'язок співпадатиме з аналітичним уже на другій ітерації:

[31]:
```
eps=0.00001
x0=3.035
x, k = Newton_iteration(f, f_deriv, a, b,x0, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps} за k={k} ітерацій")
```

```
Розв'язок рівняння x=3.035090330572526 з точністю eps=1e-05 за k=2 ітерацій
```

[32]:
```
eps=0.00000000001
x0=3.035
x, k = Newton_iteration(f, f_deriv, a, b,x0, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps} за k={k} ітерацій")
```

```
Розв'язок рівняння x=3.035090330572526 з точністю eps=1e-11 за k=3 ітерацій
```

[33]:
```
eps=0.0000000000001
x0=3.035
x, k = Newton_iteration(f, f_deriv, a, b,x0, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps} за k={k} ітерацій")
```

```
Розв'язок рівняння x=3.035090330572526 з точністю eps=1e-13 за k=3 ітерацій
```

Разом з тим, через грубе задання початкового наближення чисельний розв'язок за методом Ньютона може співпасти з іншими аналітичними розв'язками, наприклад, при заданих у наступних комірках наближеннях матимемо співпадіння з x_1 та x_3:

[51]:
```
eps=0.0000001
x0=2.5
x, k = Newton_iteration(f, f_deriv, a, b,x0, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps} за k={k} ітерацій")
```

```
Розв'язок рівняння x=2.0 з точністю eps=1e-07 за k=8 ітерацій
```

[52]:
```
eps=0.0000001
x0=2.7
x, k = Newton_iteration(f, f_deriv, a, b,x0, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps} за k={k} ітерацій")
```

```
Розв'язок рівняння x=3.698737724785346 з точністю eps=1e-07 за k=5 ітерацій
```

Приклад 2. Нехай у рівнянні (3.1.1) $f(x) = cos(x^2 - 2x)$. Обчислити методом Ньютона другий додатній розв'язок цього рівняння з точністю $eps = 10^{-5}$.

Визначимо функції лівої частини рівняння і її похідної:

[53]:
```
def fcs(x):
    """функція лівої частини рівняння (1)"""
    return np.cos(x*x-2*x)

def fcs_deriv(x):
    """похідна функції лівої частини рівняння (1)"""
    return -(2*x - 2)*np.sin(x*x-2*x)
```

Задамо уже відомий з попередніх підрозділів відрізок, де знаходиться шуканий розв'язок:

[54]:
```
a=3.38
b=3.39
```

Послідовне виконання наступних комірок даватиме чисельний розв'язок заданого рівняння при відповідних значеннях параметра eps:

[61]:
```
x0=3.38
eps=0.001
x, k = Newton_iteration(fcs, fcs_deriv, a, b,x0, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps} за k={k} ітерацій")
```

```
Розв'язок рівняння x=3.3900604555586753 з точністю eps=0.001 за k=2 ітерацій
```

[62]:
```
print(f"Відносна похибка чисельного розв'язку delta={np.abs(x_2-x)/x_2}")
```

```
Відносна похибка чисельного розв'язку delta=5.1876423561323855e-11
```

[56]:
```
eps=0.00001
x, k = Newton_iteration(fcs, fcs_deriv, a, b,x0, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps} за k={k} ітерацій")
```

```
Розв'язок рівняння x=3.390060455382811 з точністю eps=1e-05 за k=3 ітерацій
```

[63]:
```
eps = 0.000001
x, k = Newton_iteration(fcs, fcs_deriv, a, b,x0, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps} за k={k} ітерацій")
```

```
Розв'язок рівняння x=3.390060455382811 з точністю eps=1e-06 за k=3 ітерацій
```

Як бачимо, при заданому початковому наближенні вже за три ітерації чисельний розв'язок співпалдає з аналітичним. Зазначимо, грубше початкове наближення вимагає більшу кількість ітерацій, або ж можемо отримати наближення іншого розв'язку:

[67]:
```
eps = 0.00000001
x0 = 3.2
x, k = Newton_iteration(fcs, fcs_deriv, a, b,x0, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps} за k={k} ітерацій")
```

```
Розв'язок рівняння x=3.3900604553828106 з точністю eps=1e-08 за k=5 ітерацій
```

[72]:
```
eps = 0.00000001
x0 = 3.1
x, k = Newton_iteration(fcs, fcs_deriv, a, b,x0, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps} за k={k} ітерацій")
```

```
Розв'язок рівняння x=3.975564086685831 з точністю eps=1e-08 за k=4 ітерацій
```

[14]:
```
plt.close('all')
```

Отож, метод Ньютона дає змогу ефективно отримувати чисельні розв'язки нелінійних рівнянь у тих випадках, коли похідна функції лівої частини рівняння не перетворюється в нуль поблизу шуканого розв'язку.

3.3.5 Застосування методу хорд

Нехай рівняння (3.3.1) має і тільки один розв'язок x_* на відрізку $[a, b]$ і $f(a) \cdot f(b) < 0$.

Ітераційний метод січних визначається так:

- задаємо початкові значення $x_0, x_1 \in [a, b]$
- знаходимо послідовні наближення $x_2, x_3, \ldots$ розв'язку рівняння (3.3.1) за формулою

$$x_{n+1} = x_n - \frac{(x_n - x_{n-1}) \cdot f(x_n)}{f(x_n) - f(x_{n-1})}, \quad n = 1, 2, 3, \ldots \quad . \tag{3.3.15}$$

Пояснення до використання програмного коду

- Підготувати середовище і потрібні функції:
  1. виконати комірку для підготовки середовища
  2. виконати комірки, де **визначені** функції `secant_iteration` і `plot_graphics`
- Обчислити чисельні розв'язки конкретного рівняння:
  1. виконати комірку, де **визначена** функція `f`
  2. локалізовати потрібний розв'язок (як це продемонстровано в підрозділі 3.3.1) і задати координати відповідного відрізка;
  3. задати точність `eps` чисельного розв'язку та початкові наближення `x0` і `x1`;
  4. виконати комірку, де є **виклик** функції `secant_iteration`.

5. Для знаходження іншого розв'язку треба спочатку виконати підпункт 2, щоб локалізувати його, а потім виконати підпункт 4.

Програмна реалізація методу

`secant_iteration` – **функція, яка реалізує метод січних**

[2]:
```
def secant_iteration(f,a, b, x0,x1, eps):
    """ знаходження методом січних наближеного кореня рівняння,
        де f -- неперервна функція на відрізку [a, b],
        x0,x1 -- початкові наближення
        eps -- задана точність
    """
    x_pprev=x0
    x_prev=x1
    k=2
    x_new = x_prev - f(x_prev)*(x_prev-x_pprev)/(f(x_prev)-f(x_pprev))
    if np.abs(x_new-x_prev)<eps:
        return x_new,k
    while np.abs(x_new-x_prev) > eps:
        k+=1
        x_pprev=x_prev
        x_prev = x_new
        x_new = x_prev - f(x_prev)*(x_prev-x_pprev)/(f(x_prev)-f(x_pprev))
    return x_new,k
```

Обчислювальні експерименти
Знаходження чисельних розв'язків методом січних продемонструємо на тих самих, що і раніше, прикладах.

Приклад 1. Нехай у рівнянні (3.1.1) $f(x) = sin(x^2 - 2x)$. Обчислити методом січних другий додатній розв'язок цього рівняння з точністю $eps = 10^{-5}$.

Визначимо функцію для задання лівої частини рівняння:

[5]:
```
def f(x):
    """функція лівої частини рівняння """
    return np.sin(x*x-2*x)
```

Задамо відомий уже з попередніх підрозділів відрізок, де знаходиться шуканий корінь, а також досить грубі початкові наближення на цьому відрізку:

[8]:
```
a=2.8
b=3.2
x0=3
x1=b
```

Отримаємо чисельні розв'язки рівняння при різних значеннях параметра eps:

[10]:
```
eps=0.001
x, k = secant_iteration(f, a, b,x0, x1, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps} за k={k} ітерацій")
```

```
Розв'язок рівняння x=3.0350502181580414 з точністю eps=0.001 за k=3 ітерацій
```

[11]:
```
print(f"Відносна похибка чисельного розв'язку delta={np.abs(x_2-x)/x_2}")
```

```
Відносна похибка чисельного розв'язку delta=1.321621767912365e-05
```

[12]:
```
eps=0.00001
x, k = secant_iteration(f, a, b,x0, x1, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps} за k={k} ітерацій")
```

```
Розв'язок рівняння x=3.035090330572613 з точністю eps=1e-05 за k=5 ітерацій
```

[13]:
```
print(f"Відносна похибка чисельного розв'язку delta={np.abs(x_2-x)/x_2}")
```

```
Відносна похибка чисельного розв'язку delta=2.867838372184236e-14
```

[14]:
```
eps=0.00000000001
x, k = secant_iteration(f, a, b,x0, x1, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps} за k={k} ітерацій")
```

```
Розв'язок рівняння x=3.035090330572526 з точністю eps=1e-11 за k=6 ітерацій
```

Як бачимо, навіть при грубих початкових наближеннях абсолютна та відноста похибки чисельних розв'язків швидко заникають при незначному збільшенні кількості ітерацій і вже за шість ітерації абсолютна похибка чисельного розв'язку не перевищує 10^{-15}.

Виконання наступних комірок демонструє, що при точніших початкових наближеннях чисельний розв'язок співпадатиме з аналітичним уже на п'ятій ітерації:

[15]:
```
eps=0.00001
x0=3.035
x1=3.05
x, k = secant_iteration(f, a, b,x0, x1, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps} за k={k} ітерацій")
```

```
Розв'язок рівняння x=3.035090329740257 з точністю eps=1e-05 за k=3 ітерацій
```

[16]:
```
print(f"Відносна похибка чисельного розв'язку delta={np.abs(x_2-x)/x_2}")
```

```
Відносна похибка чисельного розв'язку delta=2.7421553555232615e-10
```

[17]:
```
eps=0.00000000001
x0=3.035
x1=3.05
x, k = secant_iteration(f, a, b,x0, x1, eps)
print(f"Розв'язок рівняння x={x} з точністю eps={eps} за k={k} ітерацій")
```

```
Розв'язок рівняння x=3.035090330572526 з точністю eps=1e-11 за k=5 ітерацій
```

3.4. Лабораторний практикум з розв'язування систем нелінійних рівнянь

Розглянемо нелінійну систему рівнянь

$$\begin{cases} f_1(x_1,\dots,x_n)=0 \\ \quad\vdots \\ f_n(x_1,\dots,x_n)=0 \end{cases} \quad\Leftrightarrow\quad f(x)=0, \tag{3.4.1}$$

де $f_1,\dots,f_n$ — визначені на деякій множині $D\subset\mathbb{R}^n$ функції, а

$$f(x):=\begin{pmatrix} f_1(x_1,\dots,x_n) \\ \vdots \\ f_n(x_1,\dots,x_n)\end{pmatrix},\ x=\begin{pmatrix} x_1 \\ \vdots \\ x_n\end{pmatrix}\in D, \quad \text{— векторна функція.}$$

У попередньому підрозділі при розгляді часткового випадку системи (3.4.1) при $n=1$, тобто при чисельному розв'язуванні нелінійних рівнянь, було продемонстровано залежність ефективності ітераційних методів від точності локалізації розв'язків та їхніх початкових наближеннь, а також важливу роль графічного підходу у цьому відношенні.

Графічна візуалізація також має сенс при дослідженні поведінки функцій f_i, визначених в $\mathbb{R}^2$. Використовуючи для цього різні способи подання графіків, можна дати відповідь стосовно існування розв'язків системи, а також наочно побачити лінії перетину відповідних поверхонь і підібрати добре початкове наближення для чисельного розв'язку. В Jupyter-ноутбуках такі графічні побудови зручно виконувати на інтерактивних графічних панелях з використанням потужних графічних бібліотек, які дають змогу користувачеві переглядати зображення під різними кутами зору, динамічно змінюючи їх за допомогою мишки.

Далі у цьому підрозді розглянемо застосування методів простих ітерацій та Ньютона для розв'язування систем, що складаються з двох нелінійних рівнянь, заданих на прямокутнику

$$D:=[a,b]\times[c,d], \tag{3.4.2}$$

де a,b,c,d – деякі числа.

3.4.1 Застосування методу простої ітерації

Запишемо систему рівнянь (3.4.1) при $n=2$ у вигляді

$$\begin{cases} x_1=g_1(x_1,x_2) \\ x_2=g_2(x_1,x_2)\end{cases} \quad\Leftrightarrow\quad x=g(x),\ \ g(x):=\begin{pmatrix} g_1(x_1,x_2) \\ g_2(x_1,x_2)\end{pmatrix},\ x\in D. \tag{3.4.3}$$

Алгоритм методу простої ітерації є таким:

- задаємо початкове наближення розв'язку $x^0\in D$;
- обчислюємо послідовні наближення $x^1,x^2,x^3,\dots$ розв'язку за рекурентним співвідношенням

$$x^{k+1}:=g(x^k) \quad\Leftrightarrow\quad \begin{cases} x_1^{k+1}:=g_1(x_1^k,x_2^k) \\ x_2^{k+1}:=g_2(x_1^k,x_2^k)\end{cases}, \quad k\in\mathbb{N}\cup\{0\}.$$

Ітераційні кроки будемо виконувати до тих пір, поки не досягнемо виконання однієї з умов $\|x^{k+1}-x^k\|\le\varepsilon$ чи $k>kmax$. Тут $\|\cdot\|$ – позначення евклідової норми, $\varepsilon>0$ і $kmax\in\mathbb{N}$ – деякі числа. Число ε будемо називати точністю чисельного розв'язку. Параметр $kmax$ відіграватиме роль обмеження на кількість ітерацій, а саме ітераційний процес буде зупинено, якщо при $k=kmax$ не буде досягнуто заданої точності.

Для вибору початкового наближення будумемо аналізувати тривимірні графіки функцій f_1 і f_2, використовуючи функції бібліотеки *matplotlib*. Для зручності, візуалізацію заданих функцій виконуватимемо окремою функцією, реалізація якої передбачає побудову графіків функцій f_1 і f_2 як на окремих інтерактивних графічних панелях, так і на спільній панелі, що створює добрі умови для дослідження поведінки згаданих функцій.

Пояснення до використання програмного коду

- Підготувати середовище і потрібні функції :
  1. виконати комірку для підготовки середовища
  2. виконати комірки, де **визначені** функції `simple_iteration` і "D3_plotter"
- Обчислити чисельний розв'язок конкретної заданої системи
  1. виконати комірки, в яких **визначені** функції `f` і `g`
  2. виконати комірку з **викликом** функції `D3_plotter` для побудови тривимірних графіків функції `f` і, аналізуючи лінії їхніх перетинів на площині $x_3 = 0$, задати початкове наближення `x0`
  3. задати точність `eps` чисельного розв'язку
  4. виконати комірку з **викликом** функції `simple_iteration`

Програмна реалізація методу

Підготовка середовища

[1]:
```
%matplotlib widget
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits import mplot3d
```

Допоміжна функція

[2]:
```
def norm_3(a):
    """обчислення евклідової норми вектора a"""
    return np.sqrt(np.sum(a**2))
```

`simple_iteration` – функція, яка реалізує метод простої ітерації

[3]:
```
def simple_iteration(g, x0, eps, kmax):
    """ знаходження методом простої ітерації наближеного розв'язку системи␣
 ↪рівнянь (3.4.1),
        де g -- неперервна векторна функція,
        x0 -- початкове наближення
        eps -- задана точність
    """
    x_prev=x0.copy()
    k=1
    x_new =g(x_prev)
    while norm_3(x_new-x_prev) > eps and k<kmax:
        k+=1
        x_prev = x_new
        x_new = g(x_prev)
    return k, x_new
```

`D3_plotter` – функція для побудови тривимірних графіків функції `f`

[4]:
```
def D3_plotter(f, D, N0, N1, plotting=True):
    """
    """
    x0=np.linspace(D[0,0], D[0,1], N0+1)
```

```
    x1=np.linspace(D[1,0], D[1,1], N1+1)
    print(f"x0={x0}")
    print(f"x1={x1}")

    f0 = np.empty((N0+1,N1+1), dtype=float)
    f1 = np.empty((N0+1,N1+1), dtype=float)
    f2 = np.empty((N0+1,N1+1), dtype=float)

    for j in range(N1+1):
        for i in range(N0+1):
            f0[j,i],f1[j,i] = f(x0[i],x1[j])
            f2[j,i] = 0
    #print(f"f0={f0}")
    #print(f"f1={f1}")

    if plotting :
        X1, X0 = np.meshgrid(x1, x0)
        # set up a figure twice as wide as it is tall
        fig = plt.figure(figsize=plt.figaspect(0.5))
        # =============
        # First subplot
        # =============
        # set up the axes for the first plot
        ax = fig.add_subplot(1, 2, 1, projection='3d')
        ax.set_xlabel('x1')
        ax.set_ylabel('x0')
        ax.set_zlabel('f');
        ax.set_title(f"Графіки функції f0 і f2=0");
        surf = ax.plot_surface(X1, X0, f0, rstride=1, cstride=1,␣
↪cmap='viridis', linewidth=0, antialiased=False)
        #surf = ax.plot_surface(X1, X0, f1, rstride=1, cstride=1,␣
↪cmap='viridis', linewidth=0, antialiased=False)
        ax.plot_surface(X1, X0, f2, rstride=1, cstride=1, cmap='viridis',␣
↪linewidth=0, antialiased=False)
        fig.colorbar(surf, shrink=0.5, aspect=10)
        # ==============
        # Second subplot
        # ==============
        # set up the axes for the second plot
        ax = fig.add_subplot(1, 2, 2, projection='3d')
        ax.set_xlabel('x1')
        ax.set_ylabel('x0')
        ax.set_zlabel('f');
        ax.set_title(f"Графіки функцій f0 і f2=0");
        surf = ax.plot_wireframe(X1, X0, f0, rstride=2, cstride=2)
        #surf = ax.plot_wireframe(X1, X0, f1, rstride=2, cstride=2)
        ax.plot_wireframe(X1, X0, f2, rstride=2, cstride=2)

        fig1 = plt.figure(figsize=plt.figaspect(0.5))
        # =============
        # First subplot
        # =============
        # set up the axes for the first plot
        ax1 = fig1.add_subplot(1, 2, 1, projection='3d')
        ax1.set_xlabel('x1')
        ax1.set_ylabel('x0')
        ax1.set_zlabel('f');
```

```
        ax1.set_title(f"Графіки функцій f1 і f2=0");
        #surf = ax1.plot_surface(X1, X0, f0, rstride=1, cstride=1,␣
↪cmap='viridis', linewidth=0, antialiased=False)
        surf1 = ax1.plot_surface(X1, X0, f1, rstride=1, cstride=1,␣
↪cmap='viridis', linewidth=0, antialiased=False)
        surf2 = ax1.plot_surface(X1, X0, f2, rstride=1, cstride=1,␣
↪cmap='viridis', linewidth=0, antialiased=False)
        fig1.colorbar(surf1, shrink=0.5, aspect=10)
        # ==============
        # Second subplot
        # ==============
        # set up the axes for the second plot
        ax1 = fig1.add_subplot(1, 2, 2, projection='3d')
        ax1.set_xlabel('x1')
        ax1.set_ylabel('x0')
        ax1.set_zlabel('f');
        ax1.set_title(f"Графіки функцій f1 і f2=0");
        #surf = ax1.plot_wireframe(X1, X0, f0, rstride=2, cstride=2)
        ax1.plot_wireframe(X1, X0, f1, rstride=2, cstride=2)
        ax1.plot_wireframe(X1, X0, f2, rstride=2, cstride=2)

        fig2 = plt.figure(figsize=plt.figaspect(0.5))
        # =============
        # First subplot
        # =============
        # set up the axes for the first plot
        ax2 = fig2.add_subplot(1, 2, 1, projection='3d')
        ax2.set_xlabel('x1')
        ax2.set_ylabel('x0')
        ax2.set_zlabel('f');
        ax2.set_title(f"Графіки функцій f0,f1 і f2=0");
        surf0 = ax2.plot_surface(X1, X0, f0, rstride=1, cstride=1,␣
↪cmap='viridis', linewidth=0, antialiased=False)
        surf1 = ax2.plot_surface(X1, X0, f1, rstride=1, cstride=1,␣
↪cmap='viridis', linewidth=0, antialiased=False)
        surf2 = ax2.plot_surface(X1, X0, f2, rstride=1, cstride=1,␣
↪cmap='viridis', linewidth=0, antialiased=False)
        fig2.colorbar(surf0, shrink=0.5, aspect=10)
        # ==============
        # Second subplot
        # ==============
        # set up the axes for the second plot
        ax2 = fig2.add_subplot(1, 2, 2, projection='3d')
        ax2.set_xlabel('x1')
        ax2.set_ylabel('x0')
        ax2.set_zlabel('f');
        ax2.set_title(f"Графіки функцій f0,f1 і f2=0");
        surf0 = ax2.plot_wireframe(X1, X0, f0, rstride=2, cstride=2)
        surf1 = ax2.plot_wireframe(X1, X0, f1, rstride=2, cstride=2)
        surf2 = ax2.plot_wireframe(X1, X0, f2, rstride=2, cstride=2)
```

`domain` – **функція, яка запаковує в масив координати лівої частини системи рівнянь** (3.4.1) **і правої частини системи рівнянь** (3.4.3) **відповідно**

[5]:
```
def domain(a,b,c,d):
    return np.array([[a,b],[c,d]])
```

f і g – векторні функції лівої частини системи рівнянь (3.4.1) **і правої частини системи рівнянь** (3.4.3) **відповідно**

Обчислювальні експерименти
Продемонструємо застосування методу простої ітерації до розв'язування систем двох нелінійних рівнянь.

Приклад 1. (приклад 3.8) Обчислити методом простої ітерації розв'язок системи

$$\begin{cases} 4x_1 - \sin x_2 + 1 &= 0, \\ \cos x_1 - 2x_2 + 3 &= 0 \end{cases}, \quad (x_1, x_2)^\top \in D.$$

Запишемо систему у вигляді (3.4.3)

$$\begin{cases} x_1 &= 0.25 \sin x_2 - 0.25, \\ x_2 &= 0.5 \cos x_1 + 1.5. \end{cases}$$

У розділі 3.2.2, де було обґрунтовано метод ітерацій для заданої системи, визначено область

$$D := \left\{(x_1, x_2)^\top \in \mathbb{R}^2 \quad \Big| -\frac{1}{2} \leqslant x_1 \leqslant 0,\ 1 \leqslant x_2 \leqslant 2\right\},$$

яку функції g_0 і g_2 переводять саму в себе. Будемо задавати початкове наближення $x0$ всередині цієї області.

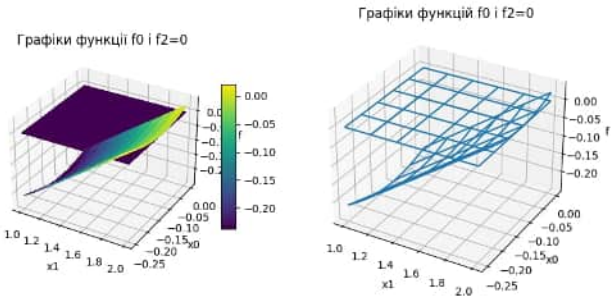
Визначимо векторні функції f і g:

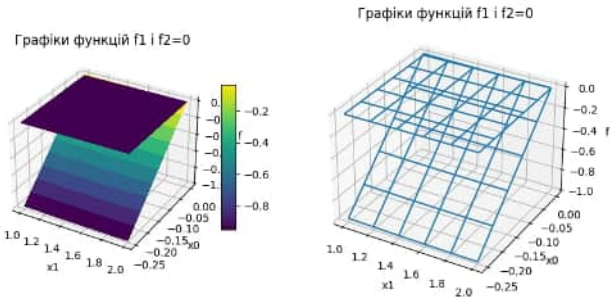
[6]:
```
def f(x0,x1):
    """функції лівої частини системи рівнянь (3.4.1)"""
    f0 = x0 - (np.sin(x1)-1)/4
    f1 = x1 - (np.cos(x0)+3)/2
    return f0,f1
def g(x):
    """функції правої частини системи рівнянь (3.4.3)"""
    g0 = (np.sin(x[1])-1)/4
    g1 = (np.cos(x[0])+3)/2
    return np.array([g0,g1])
```

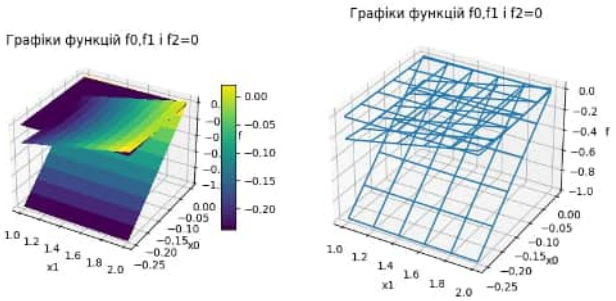
Задамо область D і побудуємо графіки функцій лівої частини системи рівнянь– спочатку кожну з них окремо, щоб бачити лінії їхнього перетину площини $x_3 = 0$, а потім разом на одній графічній панелі:

[7]:
```
D = domain(-0.25,0,1,2)
N0=10
N1=10
D3_plotter(f, D, N0, N1)
```

```
x0=[-0.25  -0.225 -0.2   -0.175 -0.15  -0.125 -0.1   -0.075 -0.05  -0.025
  0.   ]
x1=[1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2. ]
```







Зауважимо, що отримані графіки інтерактивні, їх можна обертати за допомогою мишки. Аналізуючи отримані зображення, можемо вибрати таке початкове наближення $x0$:

[8]:
```
x0=np.array([-0.05 , 1.8])
```

Тепер знайдемо чисельний розв'язок при різних значеннях параметра eps:

[9]:
```
eps=0.001
kmax=1000
k, xk = simple_iteration(g, x0, eps,kmax)
print(f"Чисельний розв'язок системи рівнянь x={xk} з точністю eps={eps},␣
↪обчислений за k={k} ітерацій")
```

```
Чисельний розв'язок системи рівнянь x=[-0.02267453  1.99987219] з точністю
eps=0.001, обчислений за k=3 ітерацій
```

[10]:
```
eps=0.0000001
k, xk = simple_iteration(g, x0, eps,kmax)
print(f"Чисельний розв'язок системи рівнянь x={xk} з точністю eps={eps},␣
↪обчислений за k={k} ітерацій")
```

```
Чисельний розв'язок системи рівнянь x=[-0.02266229  1.99987161] з точністю
eps=1e-07, обчислений за k=6 ітерацій
```

[11]:
```
eps=0.000000001
k, xk = simple_iteration(g, x0, eps,kmax)
print(f"Чисельний розв'язок системи рівнянь x={xk} з точністю eps={eps},␣
↪обчислений за k={k} ітерацій")
```

```
Чисельний розв'язок системи рівнянь x=[-0.02266229  1.99987161] з точністю
eps=1e-09, обчислений за k=7 ітерацій
```

Можна переконатися, що при заданні інших початкових наближень з області D ітераційний процес також збігатиметься:

[12]:
```
x0=np.array([0, 1.4])
```

[13]:
```
eps=0.001
k, xk = simple_iteration(g, x0, eps,kmax)
print(f"Чисельний розв'язок системи рівнянь x={xk} з точністю eps={eps},␣
↪обчислений за k={k} ітерацій")
```

```
Чисельний розв'язок системи рівнянь x=[-0.0226753   1.99987146] з точністю
eps=0.001, обчислений за k=3 ітерацій
```

[14]:
```
eps=0.0000001
k, xk = simple_iteration(g, x0, eps,kmax)
print(f"Чисельний розв'язок системи рівнянь x={xk} з точністю eps={eps},␣
↪обчислений за k={k} ітерацій")
```

```
Чисельний розв'язок системи рівнянь x=[-0.02266229  1.99987161] з точністю
eps=1e-07, обчислений за k=6 ітерацій
```

Приклад 2. (приклад 3.9м) Обчислити методом простої ітерації розв'язок системи

$$\begin{cases} x_1^2 - 2x_2 + 3 = 0, \\ 2x_1 + 3x_2^2 - 8 = 0. \end{cases}$$

Запишемо задану систему у придатному для ітерацій вигляді:

$$\begin{cases} x_1 = -1.5x_2^2 + 4, \\ x_2 = 0.5x_1^2 + 1.5. \end{cases}$$

Визначимо відповідні векторні функції для цих двох систем:

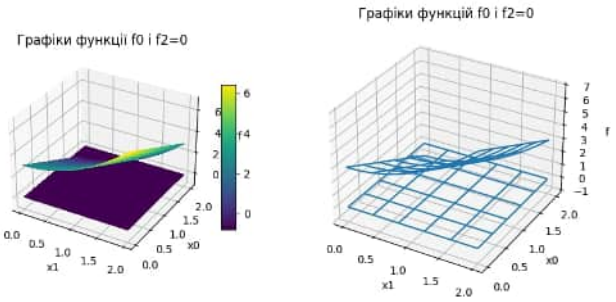
[15]:
```
def f2(x0,x1):
    """функції лівої частини системи рівнянь (3.4.1)"""
    f0 = x0**2 - 2*x1 + 3
    f1 = 2*x0 + 3*x1**2 - 8
    return f0,f1
def g2(x):
    """функції правої частини системи рівнянь (3.4.3)"""
    g0 = 3*x[0]  +  3*x[1]**2 - 8
    g1 =   x[0]**2 -x[1] + 3
```

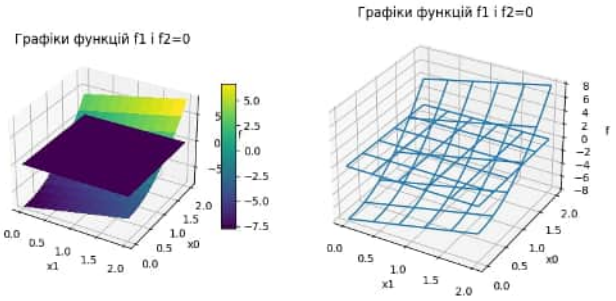
```
    return np.array([g0,g1])
```

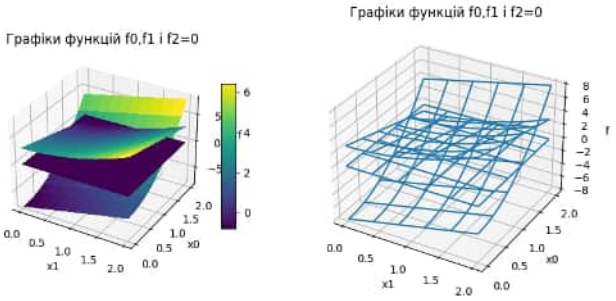
Побудуємо графіки функцій лівої частини заданої системи рівнянь:

[16]:
```
D2 = domain( 0,2,0,2)
N0=10
N1=10
D3_plotter(f2, D2, N0, N1)
```

```
x0=[0.  0.2 0.4 0.6 0.8 1.  1.2 1.4 1.6 1.8 2. ]
x1=[0.  0.2 0.4 0.6 0.8 1.  1.2 1.4 1.6 1.8 2. ]
```







На основі отриманих графіків можемо зробити висновок про існування розв'язків системи і після задання значень початкового наближення `x0`, точності `eps` і обмеження на

кількість ітерацій kmax зробити спробу отримати чисельний розв'язок:

[17]:
```
x0 = np.array([0.3, 1.5])#0.34762568 1.56042181
eps = 0.001
kmax = 1000
k, xk = simple_iteration(g2, x0, eps,kmax)
print(f"Чисельний розв'язок системи рівнянь x={xk} з точністю eps={eps},␣
↪обчислений за k={k} ітерацій")
```

```
Чисельний розв'язок системи рівнянь x=[inf nan] з точністю eps=0.001, обчислений
за k=13 ітерацій
```

```
C:\Users\a.hlova\AppData\Local\Temp\ipykernel_32728\494702554.py:3:
RuntimeWarning: overflow encountered in square
  return np.sqrt(np.sum(a**2))
C:\Users\a.hlova\AppData\Local\Temp\ipykernel_32728\2354711323.py:8:
RuntimeWarning: overflow encountered in scalar power
  g0 = 3*x[0]  +  3*x[1]**2 - 8
C:\Users\a.hlova\AppData\Local\Temp\ipykernel_32728\2354711323.py:9:
RuntimeWarning: overflow encountered in scalar power
  g1 =   x[0]**2 -x[1] + 3
C:\Users\a.hlova\AppData\Local\Temp\ipykernel_32728\2354711323.py:9:
RuntimeWarning: invalid value encountered in scalar subtract
  g1 =   x[0]**2 -x[1] + 3
C:\Users\a.hlova\AppData\Local\Temp\ipykernel_32728\4185661350.py:10:
RuntimeWarning: invalid value encountered in subtract
  while norm_3(x_new-x_prev) > eps and k<kmax:
```

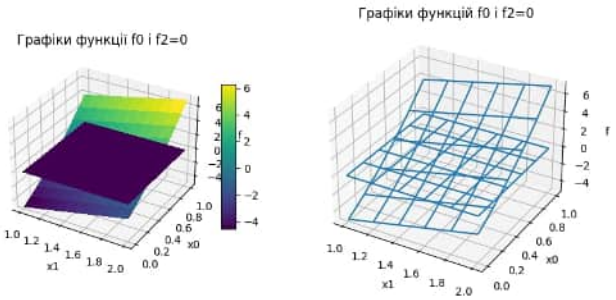
Отримані повідомлення про переповнення проміжних значень вже при невеликій кількості виконаних ітерацій свідчать про розбіжність методу. Можемо за допомогою графічних побудов переконатися, що функція g2 не виконує стрискаючого відображення. З цією метою дещо змінимо реалізацію функції g2 (назвавши її gg2), щоб вона задовольняла інтерфейс функції D3_plotter:

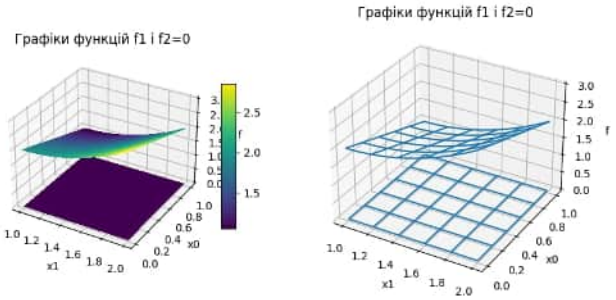
[18]:
```
def gg2(x0,x1):
    """функції правої частини системи рівнянь (3.4.3)"""
    g0 = 3*x0  +  3*x1**2 - 8
    g1 =   x0**2 -x1 + 3
    return g0,g1
```

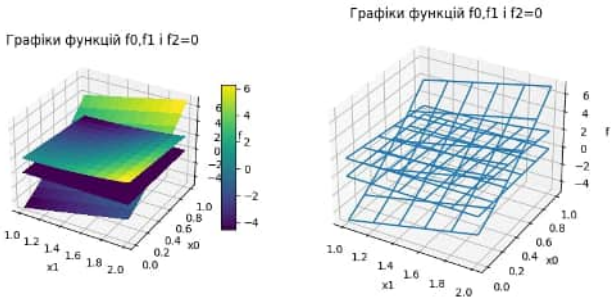
Побудуємо графіки цієї функції:

[20]:
```
D2 = domain( 0,1,1,2)
N0=10
N1=10
D3_plotter(gg2, D2, N0, N1)
```

```
x0=[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
x1=[1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2. ]
```







На основі шкали значень робимо висновок, що відображення дійсно не є стискаючим, очевидно через квадратичний характер обох компонент векторної функції g2.

Отож побудований нами ітераційний процес не дає змоги отримати чисельний розв'язок методом ітерацій. Разом з тим зазначимо, що його вдається отримати методом Ньютона.

[40]:
```
plt.close('all')
```

3.4.2 Застосування методу Ньютона

Розглянемо деякі аспекти знаходження методом Ньютона чисельних розв'язків для систем нелінійних рівнянь (3.4.1) при $n = 2$. Отож стосовно системи

$$\begin{cases} f_1(x_1, x_2) = 0 \\ f_2(x_1, x_2) = 0 \end{cases} \quad \Leftrightarrow \quad f(x) = 0, \quad f(x) := \begin{pmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \end{pmatrix}, \; x := \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in D, \tag{3.4.4}$$

припускаємо, що $f_i \in C^1(D)$, $i = \overline{1, 2}$, і матриця Якобі

$$\nabla f(x) := \begin{pmatrix} \dfrac{\partial f_1(x_1, x_2)}{\partial x_1} & \dfrac{\partial f_1(x_1, x_2)}{\partial x_2} \\ \dfrac{\partial f_2(x_1, x_2)}{\partial x_1} & \dfrac{\partial f_2(x_1, x_2)}{\partial x_2} \end{pmatrix} \tag{3.4.5}$$

невироджена в кожній точці $x \in D$.

Алгоритм методу Ньютона є таким:

- задаємо початкове наближення розв'язку $x^0 \in D$;
- знаходимо послідовні наближення $x^1, x^2, x^3, \ldots$ розв'язку за формулою

$$x^{k+1} = x^k - [\nabla f(x^k)]^{-1} f(x^k), \qquad k = 0, 1, 2, \ldots. \tag{3.4.6}$$

до тих пір, поки не досягнемо виконання однієї з умов $\|x^{k+1} - x^k\| \le \varepsilon$ чи $k > kmax$. Тут $[\nabla f(x^k)]^{-1}$ – обернена до матриці Якобі, $\|\cdot\|$ – позначення евклідової норми, $\varepsilon > 0$ і $kmax \in \mathbb{N}$ – деякі числа.

Число ε будемо називати точністю чисельного розв'язку. Параметр $kmax$ відіграватиме роль обмеження на кількість ітерацій, а саме ітераційний процес буде зупинено, якщо при $k = kmax$ не буде досягнуто заданої точності.

У програмній реалізації методу обернену до матриці Якобі будемо знаходити бібліотечною функцією *linalg.inv*. Як і при застосуванні методу простої ітерації, досліджувати поведінку заданих функцій будемо за допомогою графічної візуалізації.

Пояснення до використання програмного коду

- Підготувати середовище і потрібні функції :
  1. виконати комірку для підготовки середовища
  2. виконати комірки, в яких **визначені** функції `Newton_iteration` і `D3_plotter`
- Обчислити чисельний розв'язок конкретної заданої системи
  1. виконати комірку, де **визначені** функції `f` і `inverse_Jacobian_matrix`
  2. виконати комірку з **викликом** функції `D3_plotter` для побудови тривимірних графіків функції `f` і, аналізуючи лінії їхніх перетинів на площині $x_3 = 0$, задати початкове наближення `x0`
  3. задати точність `eps` чисельного розв'язку
  4. виконати комірку з **викликом** функції `Newton_iteration`

Програмна реалізація методу

Підготовка середовища

[1]:
```
%matplotlib widget
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits import mplot3d
from scipy import linalg
```

`Newton_iteration` – функція, яка реалізує метод Ньютона

[2]:
```
def Newton_iteration(f,invJ, x0, eps, kmax):
    """ знаходження методом Ньютона наближеного кореня рівняння (3.4.4),
        де f -- неперервна функція на відрізку [a, b],
        f_deriv -- похідна функція на відрізку [a, b]
        x0 -- початкове наближення
        eps -- задана точність
    """
    x_prev=x0.copy()
    k=1

    x_new = x_prev - invJ(x_prev).dot(f(x_prev))

    while norm_3(x_new-x_prev) > eps and k<kmax:
        k+=1
        x_prev = x_new
        x_new = x_prev - invJ(x_prev).dot(f(x_prev))
    return k, x_new
```

`D3_plotter` – **функція для побудови тривимірних графіків функції** `f`

Допоміжні функції

[4]:
```
def norm_3(a):
    """обчислення евклідової норми вектора a"""
    return np.sqrt(np.sum(a**2))
```

`domain` – **функція, яка запаковує в масив координати лівої частини системи рівнянь** (3.4.4) **і правої частини системи рівнянь** (3.4.6) **відповідно**

[5]:
```
def domain(a,b,c,d):
    return np.array([[a,b],[c,d]])
```

`f` **і** `inverse_Jacobian_matrix` – **векторна функція лівої частини рівняння** (3.4.4) **та обернена її матриці Якобі**

Обчислювальні експерименти

Продемонструємо застосування методу Ньютона до розв'язування систем двох нелінійних рівнянь.

Приклад 1. (приклад 3.8) Обчислити методом Ньютона розв'язок системи

$$\begin{cases} 4x_1 - \sin x_2 + 1 &= 0, \\ \cos x_1 - 2x_2 + 3 &= 0, \end{cases} \qquad (x_1, x_2)^\top \in D.$$

Нагадаємо, що у розділі 3 було обґрунтовано застосування іншого чисельного методу – ітерацій – до заданої системи в області

$$D := \left\{(x_1, x_2)^\top \in \mathbb{R}^2 \quad \Big| -\frac{1}{2} \leqslant x_1 \leqslant 0,\ 1 \leqslant x_2 \leqslant 2\right\},$$

де цим методом і було знайдено чисельний розв'язок системи. Тому далі також будемо розглядати цю систему в `D` і задавати початкове наближення `x0` всередині цієї області.

Визначимо векторну функцію, яка задає ліву частину системи (3.4.4), і функцію, яка повертає обернену матрицю для якобіана лівої частини:

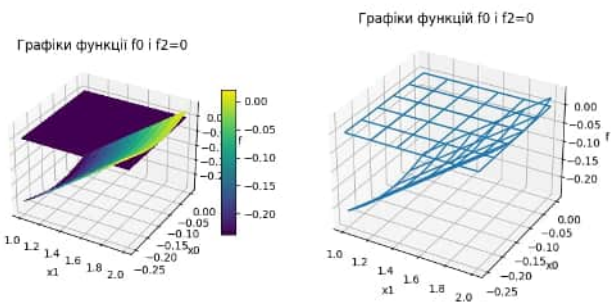
[6]:
```
def f(x):
    """функції лівої частини системи рівнянь (3.4.4)"""
    f0 = 4*x[0] - np.sin(x[1]) + 1
    f1 = np.cos(x[0]) - 2*x[1] + 3
    return np.array([f0,f1])

def inverse_Jacobian_matrix(x):
    """обернена матриця якобіана лівої частини рівняння (3.4.5)"""
    df00 = 4
    df01 = - np.cos(x[1])
    df10 = - np.sin(x[0])
    df11 =-2
    invJ = linalg.inv(np.array([[df00, df01],[df10, df11]]))
    return invJ
```

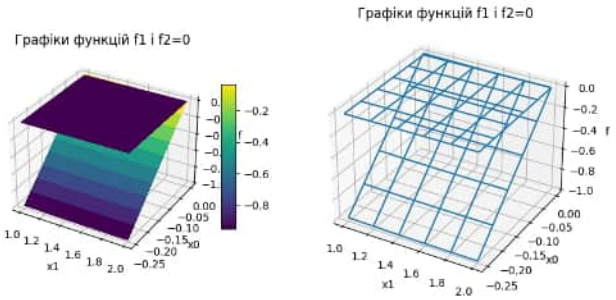
Побудуємо області D графіки функцій лівої частини системи рівнянь– спочатку кожну з них окремо, щоб бачити лінії їхнього перетину площини $x_3 = 0$, а потім разом на одній графічній панелі. Щоб мати змогу скористатися функцією D3_plotter, спочатку дещо змінимо реалізацію функції f (назвавши її ff), щоб вона задовольняла інтерфейс функції D3_plotter:

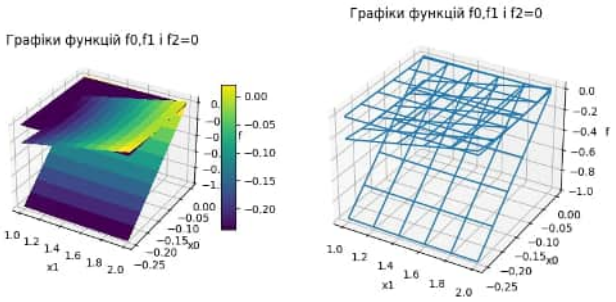
[7]:
```
def ff(x0,x1):
    """функції лівої частини системи рівнянь (3.4.4)"""
    f0 = x0 - (np.sin(x1)-1)/4
    f1 = x1 - (np.cos(x0)+3)/2
    return f0,f1
```

[8]:
```
D = domain(-0.25,0,1,2)
N0=10
N1=10
D3_plotter(ff, D, N0, N1)
```

```
x0=[-0.25  -0.225 -0.2   -0.175 -0.15  -0.125 -0.1   -0.075 -0.05  -0.025
  0.   ]
x1=[1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2. ]
```







Зауважимо, що отримані графіки інтерактивні, їх можна обертати за допомогою мишки для динамічної зміни кута зору.

На основі отриманих графіків можемо зробити висновок про існування розв'язків системи. Після задання значень початкового наближення `x0`(такого ж, як і при застосуванні методу простої ітерації) та обмеження на кількість ітерацій `kmax` знайдемо чисельний розв'язок при різних значеннях параметра `eps`:

[9]:
```
x0=np.array([-0.5 , 2])
kmax = 100
```

[10]:
```
eps=0.001
k, xk = Newton_iteration(f, inverse_Jacobian_matrix, x0, eps,kmax)
print(f"Чисельний розв'язок системи рівнянь x={xk} з точністю eps={eps} за␣
↪k={k} ітерацій")
```

```
Чисельний розв'язок системи рівнянь x=[-0.02266229  1.99987163] з точністю
eps=0.001 за k=3 ітерацій
```

[11]:
```
eps=0.0000001
k, xk = Newton_iteration(f, inverse_Jacobian_matrix, x0, eps,kmax)
print(f"Чисельний розв'язок системи рівнянь x={xk} з точністю eps={eps} за␣
↪k={k} ітерацій")
```

```
Чисельний розв'язок системи рівнянь x=[-0.02266229  1.99987161] з точністю
eps=1e-07 за k=4 ітерацій
```

[12]:
```
eps=0.000000001
k, xk = Newton_iteration(f, inverse_Jacobian_matrix, x0, eps,kmax)
```

```
print(f"Чисельний розв'язок системи рівнянь x={xk} з точністю eps={eps} за␣
↪k={k} ітерацій")
```

```
Чисельний розв'язок системи рівнянь x=[-0.02266229  1.99987161] з точністю
eps=1e-09 за k=5 ітерацій
```

Можна переконатися, що при заданні інших початкових наближень з області D ітераційний процес також збігатиметься:

[13]:
```
x0=np.array([0, 1])
kmax = 100
```

[14]:
```
eps=0.001
k, xk = Newton_iteration(f, inverse_Jacobian_matrix, x0, eps,kmax)
print(f"Чисельний розв'язок системи рівнянь x={xk} з точністю eps={eps} за␣
↪k={k} ітерацій")
```

```
Чисельний розв'язок системи рівнянь x=[-0.02266229  1.99987161] з точністю
eps=0.001 за k=4 ітерацій
```

[15]:
```
eps=0.0000001
k, xk = Newton_iteration(f, inverse_Jacobian_matrix, x0, eps,kmax)
print(f"Чисельний розв'язок системи рівнянь x={xk} з точністю eps={eps} за␣
↪k={k} ітерацій")
```

```
Чисельний розв'язок системи рівнянь x=[-0.02266229  1.99987161] з точністю
eps=1e-07 за k=5 ітерацій
```

Порівнюючи з методом простої ітерації бачимо, що для цього прикладу задана точність досягається за меншу кількість ітерацій.

Приклад 2. (приклад 3.9м) Обчислити методом Ньютона розв'язок системи

$$\begin{cases} x_1^2 - 2x_2 + 3 = 0, \\ 2x_1 + 3x_2^2 - 8 = 0. \end{cases}$$

Визначимо векторну функцію, яка задає ліву частину системи, і функцію, яка повертає обернену матрицю якобіана лівої частини:

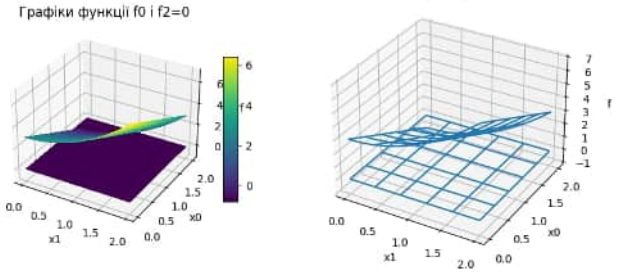
[16]:
```
def f(x):
    """функції лівої частини системи рівнянь (3.4.4)"""
    f0 = x[0]**2 - 2*x[1] + 3
    f1 = 2*x[0] + 3*x[1]**2 - 8
    return np.array([f0,f1])

def inverse_Jacobian_matrix(x):
    """обернена матриця якобіана лівої частини рівняння (3.4.5)"""
    df00 = 2*x[0]
    df01 = -2
    df10 = 2
    df11 = 6*x[1]
    invJ = linalg.inv(np.array([[df00, df01],[df10, df11]]))
    return invJ
```

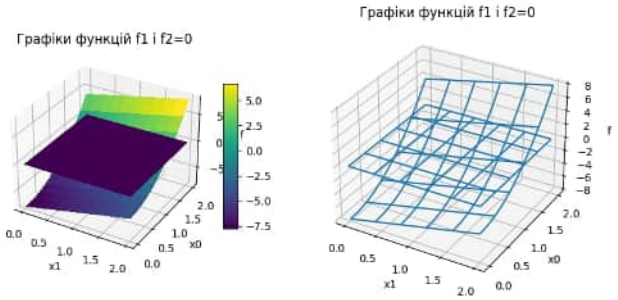
Як і в попередньому прикладі, дещо змінимо реалізацію функції `f` і побудуємо її графіки:

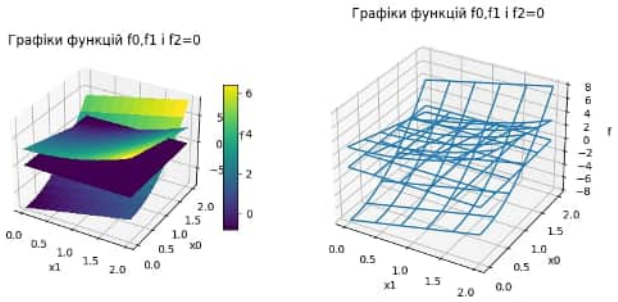
[17]:
```
def ff2(x0,x1):
    """функції лівої частини системи рівнянь (3.4.4)"""
    f0 = x0**2 - 2*x1 + 3
    f1 = 2*x0 + 3*x1**2 - 8
    return f0,f1
```

[18]:
```
D2 = domain( 0,2,0,2)
N0=10
N1=10
D3_plotter(ff2, D2, N0, N1)
```

```
x0=[0.  0.2 0.4 0.6 0.8 1.  1.2 1.4 1.6 1.8 2. ]
x1=[0.  0.2 0.4 0.6 0.8 1.  1.2 1.4 1.6 1.8 2. ]
```







На основі отриманих графіків можемо зробити висновок про існування розв'язків системи і після задання значень початкового наближення `x0` та обмеження на кількість ітерацій `kmax` знайдемо чисельний розв'язок при різних значеннях параметра `eps`:

[19]:
```
kmax=100
x0=np.array([1.01, 1.99])
eps=0.001
k, xk = Newton_iteration(f, inverse_Jacobian_matrix, x0, eps,kmax)
print(f"Чисельний розв'язок системи рівнянь x={xk} з точністю eps={eps} за␣
↪k={k} ітерацій")
```

```
Чисельний розв'язок системи рівнянь x=[0.34762643 1.56042166] з точністю
eps=0.001 за k=4 ітерацій
```

[20]:
```
eps=0.00001
k, xk = Newton_iteration(f, inverse_Jacobian_matrix, x0, eps,kmax)
print(f"Чисельний розв'язок системи рівнянь x={xk} з точністю eps={eps} за␣
↪k={k} ітерацій")
```

```
Чисельний розв'язок системи рівнянь x=[0.34762568 1.56042181] з точністю
eps=1e-05 за k=5 ітерацій
```

[21]:
```
eps=0.0000001
k, xk = Newton_iteration(f, inverse_Jacobian_matrix, x0, eps,kmax)
print(f"Чисельний розв'язок системи рівнянь x={xk} з точністю eps={eps} за␣
↪k={k} ітерацій")
```

```
Чисельний розв'язок системи рівнянь x=[0.34762568 1.56042181] з точністю
eps=1e-07 за k=6 ітерацій
```

Порахуємо чисельний розв'язок ще при іншому початковому наближенні:

[22]:
```
eps=0.001
kmax=100
x0=np.array([0,1])
k, xk = Newton_iteration(f, inverse_Jacobian_matrix, x0, eps,kmax)
print(f"Чисельний розв'язок системи рівнянь x={xk} з точністю eps={eps} за␣
↪k={k} ітерацій")
```

```
Чисельний розв'язок системи рівнянь x=[0.34762582 1.56042178] з точністю
eps=0.001 за k=5 ітерацій
```

Як бачимо, і в цьому випадку метод Ньютона демонструє свою ефективність.

Приклад 1. (example 7.1) Обчислити методом Ньютона розв'язок системи

$$\begin{cases} e^{x_1^2+x_2^2} = 1, \\ e^{x_1^2-x_2^2} = 1. \end{cases}$$

Легко бачити, що розв'язком заданої системи є вектор $x = (0,0)^\top$, будемо використовувати його для аналізу чисельних розв'язків.

Перепишемо систему у вигляді (3.4.4):

$$\begin{cases} e^{x_1^2+x_2^2} - 1 = 0, \\ e^{x_1^2-x_2^2} - 1 = 0. \end{cases}$$

Як і в попередньому прикладі, визначимо потрібні функції:

[23]:
```
def f3(x):
    """функції лівої частини системи рівнянь (3.4.4)"""
    f0 = np.exp(x[0]**2 + x[1]**2) - 1
    f1 = np.exp(x[0]**2 - x[1]**2) - 1
    return np.array([f0,f1])
```

```
def inverse_Jacobian_matrix(x):
    """обернена матриця якобіана лівої частини рівняння (3.4.5)"""
    df00 = np.exp(x[0]**2 + x[1]**2)*2*x[0]
    df01 = np.exp(x[0]**2 + x[1]**2)*2*x[1]
    df10 = np.exp(x[0]**2 - x[1]**2)*2*x[0]
    df11 =-np.exp(x[0]**2 - x[1]**2)*2*x[1]
    invJ = linalg.inv(np.array([[df00, df01],[df10, df11]]))
    return invJ
```

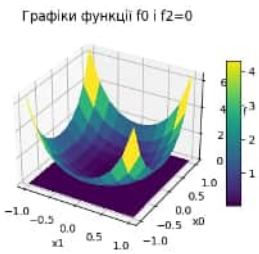
Візуалізуємо поведінку заданих функцій за допомогою фукції `D3_plotter`. Для узгодження з інтерфейсом цієї функції модифікуємо реалізацію функції `f3`:

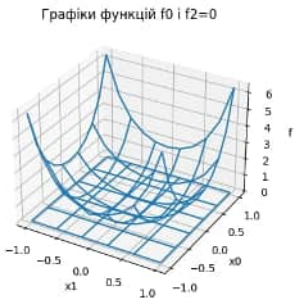
[24]:
```
def ff3(x0, x1):
    """функції лівої частини системи рівнянь (3.4.4)"""
    f0 = np.exp(x0**2 + x1**2) - 1
    f1 = np.exp(x0**2 - x1**2) - 1
    return f0,f1
```

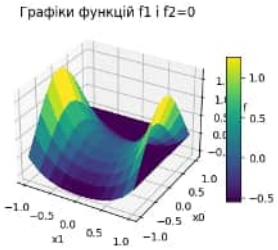
Тепер знайдемо чисельний розв'язок при різних значеннях параметра *eps*, попередньо задавши обмеження кількості операцій та початкове наближення:

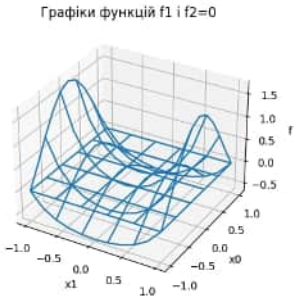
[25]:
```
D3 = domain( -1,1,-1,1)
N0=10
N1=10
D3_plotter(ff3, D3, N0, N1)
```

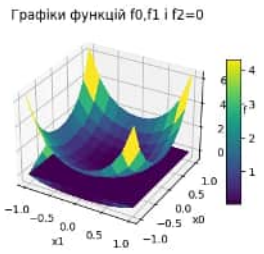
```
x0=[-1.  -0.8 -0.6 -0.4 -0.2  0.   0.2  0.4  0.6  0.8  1. ]
x1=[-1.  -0.8 -0.6 -0.4 -0.2  0.   0.2  0.4  0.6  0.8  1. ]
```

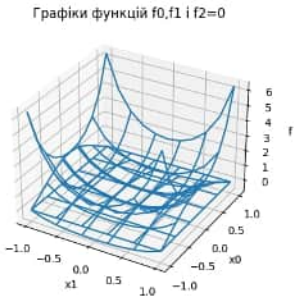












З отриманих графіків очевидно, що окрім вектора $x = (0,0)^\top$ система немає більше розв'язків. Тому далі дослідимо, як чисельні розв'язки збігатимуться до цього вектора.

Задамо обмеження на кількість ітерацій та початкове наближення і обчислимо чисельний розв'язок при кількох значеннях параметра eps:

[26]:
```
kmax=100
x0=np.array([0.1, 0.1])
```

[27]:
```
eps=0.001
k, xk = Newton_iteration(f3, inverse_Jacobian_matrix, x0, eps,kmax)
print(f"Чисельний розв'язок системи рівнянь x={xk} з точністю eps={eps} за␣
↪k={k} ітерацій")
```

```
Чисельний розв'язок системи рівнянь x=[0.00039585 0.00039585] з точністю
eps=0.001 за k=8 ітерацій
```

[28]:
```
eps=0.00001
k, xk = Newton_iteration(f3, inverse_Jacobian_matrix, x0, eps,kmax)
print(f"Чисельний розв'язок системи рівнянь x={xk} з точністю eps={eps} за␣
↪k={k} ітерацій")
```

```
Чисельний розв'язок системи рівнянь x=[6.18511692e-06 6.18511692e-06] з точністю
eps=1e-05 за k=14 ітерацій
```

[29]:
```
eps=0.0000001
k, xk = Newton_iteration(f3, inverse_Jacobian_matrix, x0, eps,kmax)
print(f"Чисельний розв'язок системи рівнянь x={xk} з точністю eps={eps} за␣
↪k={k} ітерацій")
```

```
Чисельний розв'язок системи рівнянь x=[4.82473385e-08 4.82472950e-08] з точністю
eps=1e-07 за k=21 ітерацій
```

Як бачимо, чисельний розв'язок заданої системи рівнянь досить швидко збігається до точного розв'язку.

Якщо за початкове наближення взяти вектор правої частини, то для досягнення такої ж точності, як в попередньому випадку, треба виконати більше ітерацій:

[30]:
```
x0=np.array([1, 1])
```

[31]:
```
eps=0.001
k, xk = Newton_iteration(f3, inverse_Jacobian_matrix, x0, eps,kmax)
print(f"Чисельний розв'язок системи рівнянь x={xk} з точністю eps={eps} за␣
↪k={k} ітерацій")
```

```
Чисельний розв'язок системи рівнянь x=[0.00040068 0.00040068] з точністю
eps=0.001 за k=13 ітерацій
```

[32]:
```
eps=0.00001
k, xk = Newton_iteration(f3, inverse_Jacobian_matrix, x0, eps,kmax)
print(f"Чисельний розв'язок системи рівнянь x={xk} з точністю eps={eps} за␣
↪k={k} ітерацій")
```

```
Чисельний розв'язок системи рівнянь x=[6.26059772e-06 6.26059770e-06] з точністю
eps=1e-05 за k=19 ітерацій
```

Зазначимо, що при грубшому початковому наближенні також матимемо збіжний процес, але, що слід було очікувати, кількість ітерацій значно зростає:

[33]:
```
kmax=200
x0=np.array([5, 5])
```

[34]:
```
eps=0.001
k, xk = Newton_iteration(f3, inverse_Jacobian_matrix, x0, eps,kmax)
print(f"Чисельний розв'язок системи рівнянь x={xk} з точністю eps={eps} за␣
↪k={k} ітерацій")
```

```
Чисельний розв'язок системи рівнянь x=[0.00036503 0.00036503] з точністю
eps=0.001 за k=62 ітерацій
```

Разом з тим нагадаємо про обчислення оберненої матриці Якобі для заданої функції `f` на кожному кроці методу Ньютона. Необхідною умовою такої операції є невиродженість матриці Якобі в точках області, через які проходитимуть ітерації. З вигляду якобіана очевидно, що виродження буде при попаданні аргумента на одну з осей координат. Зокрема з наступним початковим наближенням отримаємо повідомлення про зупинку обчислень через виродженість матриці Якобі:

[35]:
```
x0=np.array([0.1, 0])
```

[36]:
```
eps=0.001
k, xk = Newton_iteration(f3, inverse_Jacobian_matrix, x0, eps,kmax)
print(f"Чисельний розв'язок системи рівнянь x={xk} з точністю eps={eps} за␣
↪k={k} ітерацій")
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
Cell In[36], line 2
      1 eps=0.001
----> 2 k, xk = Newton_iteration(f3, inverse_Jacobian_matrix, x0, eps,kmax)
      3 print(f"Чисельний розв'язок системи рівнянь x={xk} з точністю eps={eps}
 ↪за k={k} ітерацій")

Cell In[2], line 11, in Newton_iteration(f, invJ, x0, eps, kmax)
      8 x_prev=x0.copy()
      9 k=1
---> 11 x_new = x_prev - invJ(x_prev).dot(f(x_prev))
     13 while norm_3(x_new-x_prev) > eps and k<kmax:
     14     k+=1

Cell In[23], line 13, in inverse_Jacobian_matrix(x)
     11 df10 = np.exp(x[0]**2 - x[1]**2)*2*x[0]
     12 df11 =-np.exp(x[0]**2 - x[1]**2)*2*x[1]
---> 13 invJ = linalg.inv(np.array([[df00, df01],[df10, df11]]))
     14 return invJ
```

```
File␣
↪~\AppData\Local\Programs\Python\Python311\Lib\site-packages\scipy\linalg\_basic.
↪py:975, in inv(a, overwrite_a, check_finite)
    973     inv_a, info = getri(lu, piv, lwork=lwork, overwrite_lu=1)
    974 if info > 0:
--> 975     raise LinAlgError("singular matrix")
    976 if info < 0:
    977     raise ValueError('illegal value in %d-th argument of internal '
    978                      'getrf|getri' % -info)

LinAlgError: singular matrix
```

Зазначимо, що тут була роглянута найпростіша реалізація методу Ньютона. Існують різні варіанти цього методу, які дають змогу ефективно шукати чисельні розв'язки для різних випадків нелінійних рівнянь.

Вправи для самостійної роботи

1. Локалізувати на відрізках довжиною, яка не більша 0.001, по 5 найбільших від'ємних і найменших невід'ємних коренів рівняння (3.3.2).
2. *Оснастити функцію `bisection` апаратом винятків для контролю вхідних даних.
3. **Написати функцію, яка для заданої функції `f` отримує список відрізків, на яких локалізовано по одному нулі цієї функції, а результатами цієї функції є нулі, обчислені методом бісекції із заданою точністю ε, а також числа ітерацій, за які відповідні нулі були обчислені. Для знаходження нулів використати функцію `bisection`. Продемонструвати на даних, отриманих у вправі 1.

Розділ 4

Наближення функцій

Найпростіша задача наближення функції полягає у такому. В скінченній кількості точок $x_0, x_1, ..., x_n$ відомі значення $y_0, y_1, ..., y_n$ функції

$$y = f(x), \quad x \in [a, b],$$

й необхідно знайти її значення при інших значеннях x. Інколи з деяких додаткових міркувань відомо, що дану функцію доцільно наближати функцією із деякої сім'ї

$$y = g(x; a_0, a_1, ..., a_n), \ x \in [a, b], \quad (a_0, a_1, \ldots, a_n) \in \Omega \subset \mathbb{R}^n,$$

тобто шукати значення $a_0^*, a_1^*, \ldots, a_n^*$ параметрів $a_0, a_1, \ldots, a_n$ такі, що

$$f(x) \approx g\left(x; a_0^*, a_1^*, \ldots, a_n^*\right), \quad x \in [a, b]. \tag{4.0.1}$$

Якщо ми знайшли відповідні значення параметрів $a_0, a_1, \ldots, a_n$, то

$$f(x) = g(x; a_0^*, a_1^*, \ldots, a_n^*) + R_n(x), \quad x \in [a, b],$$

де

$$R_n(x) := f(x) - g(x; a_0^*, a_1^*, \ldots, a_n^*), \quad x \in [a, b],$$

— залишковий член наближення функції.

4.1. Інтерполяція функцій многочленами

Якщо значення параметрів $a_0, a_1, ..., a_n$ у формулі (4.0.1) визначають з умов

$$y_i = g(x_i; a_0, a_1, ..., a_n), \quad i = \overline{0, n},$$

де $x_i \in [a, b], \ y_i = f(x_i), \ i = \overline{0, n}$, — задані, то такий спосіб наближення називають *інтерполяцією* або *інтерполюванням*, а $x_i \in [a, b], \ i = \overline{0, n}$, — вузлами інтерполяції.

Нехай z_1 — найменше з вузлів інтерполяції, а z_2 — найбільше з них. Якщо точка $x \in [a, b]$, в якій обчислюється значення $f(x)$ лежить зовні$[z_1, z_2]$, то разом з терміном інтерполяція використовується термін *екстраполяція*.

Серед способів інтерполювання найбільш поширений випадок лінійного інтерполювання, коли наближення шукають у вигляді

$$g(x, a_0, ..., a_n) = \sum_{j=0}^{n} a_j \varphi_j(x), \quad x \in [a, b],$$

де $\{\varphi_i(x), \ x \in [a, b]\}_{i=\overline{0,n}}$ — задана система лінійно незалежних функцій з простору $C[a, b]$, а значення коефіцієнтів a_i визначаються з умов:

$$\sum_{i=0}^{n} a_j \varphi_j(x_i) = y_i, \quad i = \overline{0, n}. \tag{4.1.1}$$

Метод розв'язування задачі, при якому коефіцієнти a_i визначають безпосереднім розв'язування системи (4.1.1), називають *методом неозначених коефіцієнтів.*

Найбільш часто використовується інтерполяція многочленами. Однак, це не єдино можливий тип інтерполяції. Інколи зручно наближати функцію тригонометричними функціями, а також раціональними функціями.

4.1.1. Інтерполяційний многочлен Лагранжа

Найчастіше на практиці використовується лінійне інтерполювання, коли $\varphi_j(x) = x^j$, $x \in [a, b]$, $j = \overline{0, n}$, тобто інтерполювання многочленами:

$$L_n(x) = \sum_{j=0}^{n} a_j x^j. \tag{4.1.2}$$

Тоді система рівнянь (4.1.1) має вигляд

$$\sum_{i=0}^{n} a_i {x_j}^i = y_j, \quad j = \overline{0, n}. \tag{4.1.3}$$

Оскільки визначник цієї системи є визначником Вандермонда

$$\Delta = \begin{vmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_n & x_n^2 & \dots & x_n^n \end{vmatrix},$$

а всі x_i, $j = \overline{0, n}$, — різні, то $\Delta \neq 0$. Отже, система (4.1.3) має і тільки один розв'язок. Отож, ми довели існування та єдиність інтерполяційного многочлена (4.1.2).

Безпосереднє знаходження коефіцієнтів многочлена L_n (див. (4.1.2)), розв'язуючи систему (4.1.3), вже при порівняно невеликих n (наприклад, для $n = 20$) призводить до великої обчислювальної похибки. Тому будемо шукати явне зображення інтерполяційного многочлена, не розв'язуючи систему (4.1.3).

Задача інтерполювання буде розв'язана, якщо побудувати многочлени $\Phi_i(x)$, $i = \overline{0, n}$, степеня не вище n такі, що

$$\Phi_i(x_j) = \begin{cases} 0, & i \neq j, \\ 1, & i = j, \end{cases} \qquad i, j = \overline{0, n}.$$

Тоді многочлен

$$L_n(x) = \sum_{i=0}^{n} y_i \Phi_i(x) \tag{4.1.4}$$

буде шуканим інтерполяційним многочленом. Справді, маємо

$$L_n(x_i) = \sum_{j=0}^{n} y_j \Phi_j(x_i) = y_i, \quad i = \overline{0, n}.$$

Крім того, L_n — многочлен степеня n.

Многочлени Φ_i, $i = \overline{0, n}$, будемо шукати у вигляді:

$$\Phi_i(x) = C_i(x - x_0)(x - x_1) \dots (x - x_{i-1})(x - x_{i+1}) \dots (x - x_n),$$

де C_i – неозначені коефіцієнти, які знайдемо з умови $\Phi_i(x_i) = 1$. Тоді

$$C_i = [(x_i - x_0)(x_i - x_1) \dots (x_i - x_{i-1})(x_i - x_{i+1}) \dots (x_i - x_n)]^{-1}.$$

Отже,

$$\Phi_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j} \equiv \frac{(x - x_0) \dots (x - x_{i-1})(x - x_{i+1}) \dots (x - x_n)}{(x_i - x_0) \dots (x_i - x_{i-1})(x_i - x_{i+1}) \dots (x_i - x_n)}.$$

Інтерполяційний многочлен, записаний у вигляді

$$\begin{aligned} L_n(x) &= \sum_{i=0}^{n} y_i \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j} \equiv \\ &\equiv \sum_{i=0}^{n} y_i \frac{(x-x_0)\dots(x-x_{i-1})(x-x_{i+1})\dots(x-x_n)}{(x_i-x_0)\dots(x_i-x_{i-1})(x_i-x_{i+1})\dots(x_i-x_n)} \end{aligned} \tag{4.1.5}$$

називають інтерполяційним многочленом у формі Лагранжа або, коротше, *інтерполяційним многочленом Лагранжа.*

Існують інші форми запису цього ж інтерполяційного многочлена, наприклад, інтерполяційна формула Ньютона, яку ми будемо розглядати далі.

Проведемо ***дослідження похибки***, яка виникає при заміні функції інтерполяційним многочленом. Нехай функція f визначена в $n+1$ вузлі інтерполяції $x_i \in [a,b], i = \overline{0,n}$, а L_n — інтерполяційний многочлен Лагранжа. Залишковим членом інтерполяційного многочлена (похибкою інтерполювання) називають

$$R_n(x) = f(x) - L_n(x).$$

Очевидно, що у вузлах інтерполяції цей залишковий член дорівнює нулю.

Припустимо, що $f \in C^{n+1}[a,b]$. Введемо допоміжну функцію

$$\varphi(x) = f(x) - L_n(x) - K\omega_{n+1}(x), \quad x \in [a,b], \tag{4.1.6}$$

де

$$\omega_{n+1}(x) := \prod_{j=0}^{n}(x - x_j) \equiv (x-x_0)\cdot\ldots\cdot(x-x_n),$$

а K — стала.

Зауважимо, що $\varphi \in C^{n+1}[a,b]$ та $\varphi(x_i) = 0,\ i = \overline{0,n}$. Нехай $x_* \in [a,b]$ — точка, в якій оцінюється похибка. Виберемо сталу K за умови, що $\varphi(x_*) = 0$. Для цього достатньо покласти

$$K := \frac{f(x_*) - L_n(x_*)}{\omega_{n+1}(x_*)}.$$

При такому виборі K функція φ приймає значення нуль в $n+2$ точках $x_0, ..., x_n, x_*$. На основі теореми Ролля її похідна φ' має не менше $n+1$ коренів. Міркуючи аналогічно, приходимо до висновку, що φ'' має не менше n коренів. Послідовно застосовуючи теорему Ролля до похідних вищого порядку функції φ, одержимо, що функція $\varphi^{(n+1)}$ має принаймні один корінь, тобто існує $\xi \in [a,b]$ таке, що $\varphi^{(n+1)}(\xi) = 0$. Оскільки

$$\varphi^{(n+1)}(x) = f^{(n+1)}(x) - K(n+1)!,$$

то з умови $\varphi^{(n+1)}(\xi) = 0$ будемо мати

$$K = \frac{f^{(n+1)}(\xi)}{(n+1)!}.$$

Звідси та з (4.1.6) і того, що $\varphi(x_*) = 0$, випливає рівність

$$f(x_*) - L_n(x_*) - \frac{f^{(n+1)}(\xi)}{(n+1)!}\omega_{n+1}(x_*) = 0, \quad \xi = \xi(x_*) \in [a,b], \tag{4.1.7}$$

звідки

$$R_n(x) := f(x) - L_n(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}\omega_{n+1}(x), \quad x \in [a,b]. \tag{4.1.8}$$

Покладаючи

$$M_{n+1} = \sup_{x\in[a,b]} |f^{(n+1)}(x)|,$$

отримаємо *оцінку залишкового члена*

$$|R_n(x)| \leqslant \frac{M_{n+1}}{(n+1)!}|\omega_{n+1}(x)|, \quad x \in [a,b]. \tag{4.1.9}$$

Приклад 4.1.1. Побудувати інтерполяційний многочлен Лагранжа для функції, заданої на відрізку $[0;7]$ таблицею

i	0	1	2	3
x_i	0	2	3	5
y_i	2	4	6	8

.

Доведення. Тут $n = 3$. За формулою (4.1.5) маємо

$$L_3(x) := y_0\frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)} + y_1\frac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)} +$$
$$+y_2\frac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)} + y_3\frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)} =$$
$$= 2\frac{(x-2)(x-3)(x-5)}{(0-2)(0-3)(0-5)} + 4\frac{(x-0)(x-3)(x-5)}{(2-0)(2-3)(2-5)} +$$
$$+6\frac{(x-0)(x-2)(x-5)}{(3-0)(3-2)(3-5)} + 8\frac{(x-0)(x-2)(x-3)}{(5-0)(5-2)(5-3)}.$$

□

4.1.2. Інтерполяційний многочлен Ньютона

Нехай $f \in C[a,b]$ і $x_0, x_1, \ldots, x_n \in [a,b]$ — система вузлів інтерполювання ($x_i \neq x_j$ при $i \neq j$, $i,j = 0,1,\ldots,n$). Введемо поняття ***розділених різниць***:

$$\begin{cases} f(x_0;x_1) := \dfrac{f(x_1)-f(x_0)}{x_1-x_0}, \\[2ex] f(x_1;x_2) := \dfrac{f(x_2)-f(x_1)}{x_2-x_1}, \\[2ex] \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\[2ex] f(x_{n-1};x_n) := \dfrac{f(x_n)-f(x_{n-1})}{x_n-x_{n-1}} \end{cases} \tag{4.1.10}$$

— *розділені різниці першого порядку*,

$$\begin{cases} f(x_0;x_1;x_2) := \dfrac{f(x_1;x_2)-f(x_0;x_1)}{x_2-x_0}, \\[2ex] f(x_1;x_2;x_3) := \dfrac{f(x_2;x_3)-f(x_1;x_2)}{x_3-x_1}, \\[2ex] \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\[2ex] f(x_{n-2};x_{n-1};x_n) := \dfrac{f(x_{n-1};x_n)-f(x_{n-2};x_{n-1})}{x_n-x_{n-2}} \end{cases} \tag{4.1.11}$$

— *розділені різниці другого порядку* і т. д. Якщо ми вже визначили розділені різниці k-го порядку $f(x_i;x_{i+1};\ldots;x_{i+k})$, $i = 0,1,\ldots,n-k$, то розділені різниці $(k+1)$-го порядку обчислюють за допомогою формул

$$f(x_{i-1};x_i;\ldots;x_{i+k}) := \frac{f(x_i;x_{i+1};\ldots;x_{i+k}) - f(x_{i-1};x_i;\ldots;x_{i+k-1})}{x_{i+k}-x_{i-1}}, \quad i = 1,2,\ldots,n-k. \tag{4.1.12}$$

Відомо [**?**], що розділені різниці можна обчислювати за формулою

$$f(x_0; x_1; \ldots; x_k) = \sum_{i=0}^{k} \frac{y_i}{(x_i - x_0)\ldots(x_i - x_{i-1})(x_i - x_{i+1})\ldots(x_i - x_k)}, \qquad k = 1, \ldots, n. \tag{4.1.13}$$

Тепер розглянемо інтерполяційний многочлен Лагранжа. Він незручний тим, що зі збільшенням кількості вузлів змінюються всі доданки у формулі. Зручнішою для практичного використання була б формула такого вигляду:

$$L_n(x) = A_0 + (x - x_0)A_1 + (x - x_0)(x - x_1)A_2 + \ldots + (x - x_0)(x - x_1)\ldots(x - x_{n-1})A_n,$$

де A_i, $i = 0, 1, \ldots, n$, — числові коефіцієнти. Тоді збільшення кількості вузлів приводило б до збільшення кількості доданків, а попередньо обчислені доданки залишилися би без зміни.

Подамо $L_n(x)$ у такому вигляді:

$$L_n(x) = L_0(x) + (L_1(x) - L_0(x)) + (L_2(x) - L_1(x)) + \ldots + (L_n(x) - L_{n-1}(x)),$$

де $L_k(x)$, $x \in \mathbb{R}$, —інтерполяційний многочлен Лагранжа, побудований за вузлами $x_0, \ldots, x_k$, $k \in \{0, 1, \ldots, n\}$, зокрема, $L_0(x) := y_0$.

Розглянемо різницю

$$Q_k(x) = L_k(x) - L_{k-1}(x), \quad k = 1, 2, \ldots, n, \quad x \in \mathbb{R}.$$

Очевидно, Q_k є многочленом степеня k. Він набуває нульового значення у точках $x_0, x_1, \ldots x_{k-1}$, оскільки

$$L_k(x_i) = y_i, \quad i = 0, 1, \ldots, k, \qquad L_{k-1}(x_i) = y_i, \quad i = 0, 1, \ldots, k-1.$$

Тому

$$Q_k(x) = A_k(x - x_0)(x - x_1)\ldots(x - x_{k-1}),$$

де A_k — деяка стала.

Щоб знайти значення A_k, покладемо $x = x_k$. Одержимо

$$y_k - L_{k-1}(x_k) = A_k(x_k - x_0)(x_k - x_1)\ldots(x_k - x_{k-1}),$$

або

$$y_k - \sum_{i=0}^{k-1} y_i \frac{(x_k - x_0)\ldots(x_k - x_{i-1})(x_k - x_{i+1})\ldots(x_k - x_{k-1})}{(x_i - x_0)\ldots(x_i - x_{i-1})(x_i - x_{i+1})\ldots(x_i - x_{k-1})} = \\ = A_k(x_k - x_0)(x_k - x_1)\ldots(x_k - x_{k-1}).$$

Звідси

$$A_k = \frac{y_k}{(x_k - x_0)(x_k - x_1)\ldots(x_k - x_{k-1})} + \\ + \sum_{i=0}^{k-1} \frac{y_i}{(x_i - x_0)\ldots(x_i - x_{i-1})(x_i - x_{i+1})\ldots(x_i - x_{k-1})(x_i - x_k)} = \\ = \sum_{i=0}^{k} \frac{y_i}{(x_i - x_0)\ldots(x_i - x_{i-1})(x_i - x_{i+1})\ldots(x_i - x_k)} = f(x_0; x_1; \ldots; x_k)$$

— розділена різниця k-го порядку.

Отже,

$$L_n(x) = f(x_0) + (x - x_0)f(x_0; x_1) + (x - x_0)(x - x_1)f(x_0; x_1; x_2) + \\ + \ldots + (x - x_0)(x - x_1)\ldots(x - x_{n-1})f(x_0; x_1; \ldots; x_n). \tag{4.1.14}$$

Отриманий многочлен $L_n(x)$ називається ***інтерполяційним многочленом Ньютона для нерівновіддалених вузлів інтерполювання***. Він зручніший для обчислень, ніж інтерполяційний многочлен Лагранжа, бо зі збільшенням кількості вузлів не потрібно повторювати всю роботу наново, як при обчисленнях за формулою Лагранжа.

Отож, маємо

$$f(x) = L_n(x) + R_{n+1}(x),$$

де L_n визначено в (4.1.14), а $R_{n+1}(x)$ — залишковий член. Він такий же, як і в формулі Лагранжа:

$$R_{n+1}(x) = \frac{f^{n+1}(\xi(x))\omega_{n+1}(x)}{(n+1)!}.$$

Однак його можна записати в іншій формі. Для цього розглянемо розділену різницю $(n+1)$-го порядку

$$\begin{aligned} f(x; x_0; \ldots; x_n) = \frac{f(x)}{(x-x_0)(x-x_1)\ldots(x-x_n)} + \\ + \frac{y_0}{(x_0-x)(x_0-x_1)\ldots(x_0-x_n)} + \ldots + \\ + \frac{y_n}{(x_n-x)(x_n-x_0)\ldots(x_n-x_{n-1})}. \end{aligned}$$

Знайдемо із цього співвідношення $f(x)$:

$$\begin{aligned} f(x) = y_0 \frac{(x-x_1)(x-x_2)\ldots(x-x_n)}{(x_0-x_1)(x_0-x_2)\ldots(x_0-x_n)} + \ldots + \\ + y_n \frac{(x-x_0)(x-x_1)\ldots(x-x_{n-1})}{(x_n-x_0)(x_n-x_1)\ldots(x_n-x_{n-1})} + \\ + (x-x_0)(x-x_1)\ldots(x-x_n) f(x; x_0; \ldots; x_n). \end{aligned}$$

Тому

$$f(x) = L_n(x) + (x-x_0)(x-x_1)\ldots(x-x_n) f(x; x_0; \ldots; x_n).$$

Отже,

$$R_{n+1}(x) = (x-x_0)(x-x_1)\ldots(x-x_n) f(x; x_0; \ldots; x_n),$$

або

$$R_{n+1}(x) = \omega_{n+1}(x) f(x; x_0; \ldots; x_n).$$

Зокрема, якщо функція f має похідну порядку $n+1$, то одержимо

$$f(x; x_0; x_1; \ldots; x_n) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!},$$

де $\xi = \xi(x)$ — точка, яка належить найменшому проміжкові, що містить усі точки $x_0, x_1, \ldots, x_n, x$.

Формулу (4.1.14) також називають **інтерполяційним многочленом Ньютона для інтерполювання вперед у разі нерівновіддалених вузлів інтерполювання**. Вона, як звичайно, застосовується для наближення функції поблизу початкового вузла x_0.

Якщо вузли інтерполювання вибрати в порядку $x_n, x_{n-1}, \ldots, x_0$, то аналогічно можна отримати формулу

$$\begin{aligned} L_n(x) = f(x_n) + (x-x_n) f(x_{n-1}; x_n) + \\ + (x-x_n)(x-x_{n-1}) f(x_{n-2}; x_{n-1}; x_n) + \ldots + \\ + (x-x_n)(x-x_{n-1})\ldots(x-x_1) f(x_0; x_1; \ldots x_n), \end{aligned}$$

яке має назву ***інтерполяційним многочленом Ньютона для інтерполювання назад у разі нерівновіддалених вузлів інтерполювання***. Вона, як звичайно, використовується для наближення функції поблизу кінцевого вузла x_n.

Одержані формули справедливі для будь-якої системи вузлів $x_0, x_1, \ldots, x_n$, такої, що $x_i \neq x_j$ при $i \neq j$ та $x_i \in [a, b]$ для $i, j = 0, 1, \ldots n$. Вузли інтерполювання, котрі найближче лежать до точки x, більше впливають на інтерполяційний многочлен, ніж вузли, які лежать далі. Тому доцільно за x_0 і x_1 взяти найближчі до x вузли інтерполювання і здійснити спочатку лінійну інтерполяцію за цими вузлами. Пізніше поступово використовувати інші вузли так, щоб вони, якщо це можливо, розміщувались відносно x симетрично. Отримані при цьому поправки будуть звичайно незначними.

Приклад 4.1.2. Побудувати інтерполяційний многочлен Ньютона для функції, заданої на відрізку $[0; 7]$ таблицею

i	0	1	2	3
x_i	0	2	3	5
y_i	2	4	6	8

.

Використовуючи цей многочлен, знайти наближення розглядуваної функції при $x = 1$.

Доведення. Тут $n = 3$. За формулами (4.1.10), (4.1.11), (4.1.12) знаходимо розділені різниці

$$\begin{cases} f(x_0; x_1) := \dfrac{f(x_1) - f(x_0)}{x_1 - x_0} = \dfrac{4-2}{2-0} = 1; \\[2ex] f(x_1; x_2) := \dfrac{f(x_2) - f(x_1)}{x_2 - x_1} = \dfrac{6-4}{3-2} = 2; \\[2ex] f(x_2; x_3) := \dfrac{f(x_3) - f(x_2)}{x_3 - x_2} = \dfrac{8-6}{5-3} = 1; \\[2ex] f(x_0; x_1; x_2) := \dfrac{f(x_1; x_2) - f(x_0; x_1)}{x_2 - x_0} = \dfrac{2-1}{3-0} = \dfrac{1}{3}; \\[2ex] f(x_1; x_2; x_3) := \dfrac{f(x_2; x_3) - f(x_1; x_2)}{x_3 - x_1} = \dfrac{1-2}{5-2} = -\dfrac{1}{3}; \\[2ex] f(x_0; x_1; x_2; x_3) := \dfrac{f(x_0; x_1; x_2) - f(x_1; x_2; x_3)}{x_3 - x_0} = \dfrac{-1/3 - 1/3}{5} = -\dfrac{2}{15}. \end{cases}$$

Отже, згідно з формулою (4.1.14) інтерполяційний многочлен Ньютона має вигляд

$$L_3(x) = 2 + 1 \cdot (x - 0) + \frac{1}{3} \cdot (x - 0)(x - 2) - \frac{2}{15} \cdot (x - 0)(x - 2)(x - 3).$$

Звідси легко знаходимо $L_3(1) = 2 + 1 - \dfrac{1}{3} - \dfrac{2}{15} = \dfrac{36}{15}$. □

4.2. Інтерполяція сплайнами

Інтерполяція многочленом Лагранжа або Ньютона на всьому відрізку $[a, b]$ з використанням великої кількості вузлів інтерполяції часто призводить до поганого наближення, що пояснюється сильним нагромадженням похибок в процесі обчислення. Крім того, через розбіжність процесу інтерполяції збільшення кількості вузлів не призводить до підвищення точності. Для того, щоб запобігти великим похибкам, весь відрізок $[a, b]$ розбивають на окремі відрізки та на кожному з них наближають функцію f многочленом невисокого степеня (так звана кусково-поліноміальна інтерполяція).

Одним зі способів інтерполювання на всьому відрізку є інтерполяція за допомогою сплайнів. *Сплайном (сплайн-функцією)* називають кусково-поліноміальну функцію, визначену на відрізку $[a, b]$ і таку, що має на цьому відрізку деяку кількість неперервних похідних. Найпоширеніші в інженерних розрахунках сплайни складені з многочленів третього степеня, які називають кубічними сплайнами.

Нехай на відрізку $[a, b]$ задано сітку

$$\omega = \{x_0, x_1, \ldots, x_n \,|\, a =: x_0 < x_1 < \ldots < x_n := b\},$$

у вузлах якої задано значення $y_0, y_1, \ldots, y_n$ функції $f \in C[a, b]$.

Задача кусково-кубічної інтерполяції полягає у знаходженні функції $g : [a, b] \to \mathbb{R}$ (кубічного сплайну) такої, що

- функція g є двічі неперервно диференційовною на $[a, b]$, тобто

$$g \in C^2[a, b]; \tag{4.2.1}$$

- для кожного $i \in \{1, \ldots, n\}$ звуження функції g на відрізок $[x_{i-1}, x_i]$ є кубічним многочленом, тобто має вигляд

$$g(x) = a_0^{(i)} + a_1^{(i)} x + a_2^{(i)} x^2 + a_3^{(i)} x^3 \equiv \sum_{k=0}^{3} a_k^{(i)} x^k; \tag{4.2.2}$$

- у вузлах сітки ω виконуються рівності

$$g(x_i) = y_i, \quad i = \overline{0, n}; \tag{4.2.3}$$

- її друга похідна g'' задовольняє крайові умови

$$g''(a) = g''(b) = 0. \tag{4.2.4}$$

Поставлена задача має єдиний розв'язок і його вигляд такий

$$g(x) := m_{i-1} \frac{(x_i - x)^3}{6h_i} + m_i \frac{(x - x_{i-1})^3}{6h_i} + \left(y_{i-1} - \frac{m_{i-1} h_i^2}{6}\right) \frac{x_i - x}{h_i} + \\ + \left(y_i - \frac{m_i h_i^2}{6}\right) \frac{x - x_{i-1}}{h_i}, \quad x \in [x_{i-1}, x_i], \quad i = \overline{1, n}, \tag{4.2.5}$$

де $h_i := x_i - x_{i-1},\ i = \overline{1, n}$, а сталі $m_i,\ i = \overline{1, n}$, знаходять із системи рівнянь

$$\begin{cases} m_0 = 0, \\ \dfrac{h_i}{6} m_{i-1} + \dfrac{h_i + h_{i+1}}{3} m_i + \dfrac{h_{i+1}}{6} m_{i+1} = \\ = \dfrac{y_{i-1}}{h_i} - \left(\dfrac{1}{h_i} + \dfrac{1}{h_{i+1}}\right) y_i + \dfrac{y_{i+1}}{h_{i+1}}, \quad i = \overline{1, n-1}, \\ m_n = 0. \end{cases} \tag{4.2.6}$$

Приклад 4.2.1. Інтерполювати кубічним сплайном функцію, задану на відрізку $[0; 9]$ таблицею

i	0	1	2	3
x_i	0	3	6	9
y_i	12	6	9	3

.

Доведення. В цьому прикладі $n = 3$. За формулою $h_i := x_i - x_{i-1},\ i = 1, 2, 3$, знаходимо

$$h_1 = h_2 = h_3 = 3.$$

Система (4.2.6) в нашому випадку має вигляд

$$\begin{cases} m_0 = 0, \\ \dfrac{h_1}{6}m_0 + \dfrac{h_1+h_2}{3}m_1 + \dfrac{h_2}{6}m_2 = \dfrac{y_0}{h_1} - \left(\dfrac{1}{h_1} + \dfrac{1}{h_2}\right)y_1 + \dfrac{y_2}{h_2}, \\ \dfrac{h_2}{6}m_1 + \dfrac{h_2+h_3}{3}m_2 + \dfrac{h_3}{6}m_3 = \dfrac{y_1}{h_2} - \left(\dfrac{1}{h_2} + \dfrac{1}{h_3}\right)y_2 + \dfrac{y_3}{h_3}, \\ m_3 = 0. \end{cases}$$

Звідси, підставивши конкретні значення $h_1, h_2, h_3,\ y_0, y_1, y_2, y_3$, отримаємо

$$\begin{cases} m_0 = 0 \\ \dfrac{3}{6}m_0 + \dfrac{3+3}{3}m_1 + \dfrac{3}{6}m_2 = \dfrac{12}{3} - \left(\dfrac{1}{3} + \dfrac{1}{3}\right)6 + \dfrac{9}{3} \\ \dfrac{3}{6}m_1 + \dfrac{3+3}{3}m_2 + \dfrac{3}{6}m_3 = \dfrac{6}{3} - \left(\dfrac{1}{3} + \dfrac{1}{3}\right)9 + \dfrac{3}{3} \\ m_3 = 0. \end{cases}$$

Після спрощення здобуваємо

$$\begin{cases} m_0 = 0 \\ 2m_1 + \dfrac{1}{2}m_2 = 3 \\ \dfrac{1}{2}m_1 + 2m_2 = -3 \\ m_3 = 0 \end{cases} \qquad \Longrightarrow \begin{cases} m_0 = 0 \\ m_1 = 2 \\ m_2 = -2 \\ m_3 = 0. \end{cases}$$

Отже, шуканий кубічний сплайн має вигляд

$$g(x) \qquad := \qquad \begin{cases} m_0\dfrac{(x_1-x)^3}{6h_1} + m_1\dfrac{(x-x_0)^3}{6h_1} + \left(y_0 - \dfrac{m_0h_1^2}{6}\right)\dfrac{x_1-x}{h_1} + \\ +\left(y_1 - \dfrac{m_1h_1^2}{6}\right)\dfrac{x-x_0}{h_1}, \quad \text{якщо} \quad x \in [x_0, x_1], \\ m_1\dfrac{(x_2-x)^3}{6h_2} + m_2\dfrac{(x-x_1)^3}{6h_2} + \left(y_1 - \dfrac{m_1h_2^2}{6}\right)\dfrac{x_2-x}{h_2} + \\ +\left(y_2 - \dfrac{m_2h_2^2}{6}\right)\dfrac{x-x_1}{h_2}, \quad \text{якщо} \quad x \in [x_1, x_2], \\ m_2\dfrac{(x_3-x)^3}{6h_3} + m_3\dfrac{(x-x_2)^3}{6h_3} + \left(y_2 - \dfrac{m_2h_3^2}{6}\right)\dfrac{x_3-x}{h_3} + \\ +\left(y_3 - \dfrac{m_3h_3^2}{6}\right)\dfrac{x-x_2}{h_3}, \quad \text{якщо} \quad x \in [x_2, x_3], \end{cases}$$

тобто

$$g(x) := \begin{cases} 2\dfrac{x^3}{6\cdot 3} + 12\dfrac{3-x}{3} + \left(6 - \dfrac{2\cdot 3^2}{6}\right)\dfrac{x}{3}, & \text{якщо } x \in [0;3], \\ 2\dfrac{(6-x)^3}{6\cdot 3} - 2\dfrac{(x-3)^3}{6\cdot 3} + \left(6 - \dfrac{2\cdot 3^2}{6}\right)\dfrac{6-x}{3} + \\ +\left(9 + \dfrac{2\cdot 3^2}{6}\right)\dfrac{x-3}{3}, & \text{якщо } x \in [3;6], \\ -2\dfrac{(9-x)^3}{6\cdot 3} + \left(9 + \dfrac{2\cdot 3^2}{6}\right)\dfrac{9-x}{3} + 3\dfrac{x-6}{3}, & \text{якщо } x \in [6;9]. \end{cases}$$

□

4.3. Середньо квадратичне наближення

Нехай деяка величина описується функцією $y = f(x)$, $x \in [a, b]$, явний вигляд якої нам не відомий, але ми маємо результати вимірювань $y_1, y_2, \ldots, y_n$ цієї величини при значеннях $x_1, x_2, ..., x_n \in [a, b]$ незалежної змінної x, тобто маємо таблицю значень розглядуваної величини

x	x_1	x_2	$\ldots$	x_n
y	y_1	y_2	$\ldots$	y_n

.

За даними таблиці треба вибрати із сім'ї залежних від параметрів $a_1, ..., a_m$ аналітичних функцій

$$y = g(x; a_1, ..., a_m), \quad x \in [a, b], \quad (a_1, \ldots, a_m) \in \Omega, \tag{4.3.1}$$

де $m \in \mathbb{N}$, $m < n$, Ω — область в просторі $\mathbb{R}^m$, ту, яка "досить добре" наближує функцію f на всьому відрізку $[a, b]$. Вигляд сім'ї функцій (4.3.1), як правило, відомий на основі додаткових міркувань.

Якщо система рівнянь

$$\begin{cases} g(x_1; a_1, ..., a_m) = y_1 \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ g(x_n; a_1, ..., a_m) = y_n \end{cases} \tag{4.3.2}$$

має єдиний розв'язок, то він може бути знайдений з певних m рівнянь системи (4.3.2). Однак, у загальному випадку значення y_i, x_i, $i = \overline{1, n}$, є наближеними та точний вигляд залежності (4.3.1) невідомий і через це система (4.3.2) переважно є несумісною. Тому визначимо параметри $a_1, ..., a_m$ так, щоб у деякому розумінні всі рівняння системи (4.3.2) задовольнялись з найменшою похибкою, а точніше, щоб шукані значення параметрів $a_1, ..., a_m$ мінімізували функцію

$$S(a_1, ..., a_m) = \sum_{i=1}^{n} \left[g(x_i; a_1, ..., a_m) - y_i\right]^2.$$

Такий метод розв'язання системи (4.3.2) називається *методом найменших квадратів*.

Якщо функція $S(a_1, ..., a_m)$ досягає абсолютного мінімуму в області зміни параметрів $a_1, ..., a_m$, то, розв'язуючи систему

$$\frac{\partial S}{\partial a_k} = 2\sum_{i=1}^{n}[g(x_i; a_1, ..., a_m) - y_i]\frac{\partial g(x_i; a_1, ..., a_m)}{\partial a_k} = 0, \quad k = \overline{1, m},$$

знаходимо точки, в яких може бути екстремум. Вибравши той розв'язок, який належить області зміни параметрів $a_1, ..., a_m$ і в якому функція $S(a_1, ..., a_m)$ має абсолютний мінімум, знаходимо потрібні значення $a_1, ..., a_m$.

Якщо $g(x, a_1, ..., a_m)$ лінійно залежать від параметрів $a_1, ..., a_m$, тобто

$$g(x; a_1, ..., a_m) = \sum_{j=1}^{m} a_j g_j(x), \tag{4.3.3}$$

то система (4.3.2) набуде вигляду

$$\sum_{j=1}^{m} g_j(x_i) a_j = y_i, \quad i = \overline{1, n}. \tag{4.3.4}$$

Якщо ввести позначення

$$b_{ij} := g_j(x_i), \quad i = \overline{1, n}, j = \overline{1, m}, \qquad B := \begin{pmatrix} b_{11} & \dots & b_{1m} \\ \dots & & \\ b_{n1} & \dots & b_{nm} \end{pmatrix},$$

то система (4.3.4) матиме вигляд

$$\sum_{j=1}^{m} b_{ij} a_j = y_i, \quad i = \overline{1, n}, \quad \Longleftrightarrow \quad B\widehat{a} = \widehat{y}, \tag{4.3.5}$$

де $\widehat{a} := (a_1, \dots, a_m)^\top$, $\widehat{y} := (y_1, \dots, y_m)^\top$.

Метод найменших квадратів розв'язування системи (4.3.3) полягає у тому, щоб визначити невідомі, які мінімізують суму квадратів нев'язок, тобто суму вигляду

$$S(a_1, ..., a_m) = \sum_{i=1}^{n} \left[\sum_{j=1}^{m} b_{ij} a_j - y_i \right]^2.$$

Як випливає з підрозділу **??** (див. теорему 2.3.2), шукані значення параметрів $a_1, \dots, a_n$ знаходимо із системи рівнянь

$$\boxed{B^\top B \widehat{a} = B^\top \widehat{y}.} \tag{4.3.6}$$

Приклад 4.3.1. Методом найменших квадратів для функції, заданої на відрізку $[0; 4]$ таблицею

i	0	1	2
x_i	1	2	3
y_i	2	3	5

,

побудуйте лінійний і квадратичний многочлени, що є середньо квадратичними наближеннями даної функції.

Доведення. Будемо шукати лінійний многочлен, що є середньо квадратичними наближеннями даної функції, у вигляді

$$g(x; a_1, a_2) = a_1 + a_2 x, \quad x \in [0; 4], \tag{4.3.7}$$

де значення a_1, a_2 шукаємо за умов

$$\begin{cases} g(1; a_1, a_2) & = a_1 + a_2 = 2, \\ g(2; a_1, a_2) & = a_1 + 2a_2 = 3, \\ g(3; a_1, a_2) & = a_1 + 3a_2 = 5. \end{cases}$$

Отже, маємо

$$B = \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{pmatrix}, \quad \widehat{y} = \begin{pmatrix} 2 \\ 3 \\ 5 \end{pmatrix}.$$

Тепер зауважимо, що

$$B^\top = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix}, \quad B^\top B = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{pmatrix} = \begin{pmatrix} 3 & 6 \\ 6 & 14 \end{pmatrix},$$

$$A^\top \widehat{y} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} 2 \\ 3 \\ 5 \end{pmatrix} = \begin{pmatrix} 10 \\ 23 \end{pmatrix}.$$

Отже, система для знаходження коефіцієнтів a_1, a_2 многочлена (4.3.8) (див. (4.3.6)) має вигляд

$$\begin{pmatrix} 3 & 6 \\ 6 & 14 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} 10 \\ 23 \end{pmatrix}.$$

Звідси отримуємо

$$\left(\begin{array}{cc|c} 3 & 6 & 10 \\ 6 & 14 & 23 \end{array}\right) \sim \left(\begin{array}{cc|c} 3 & 6 & 10 \\ 0 & 2 & 3 \end{array}\right) \sim \left(\begin{array}{cc|c} 6 & 0 & 1 \\ 0 & 2 & 3 \end{array}\right) \sim \left(\begin{array}{cc|c} 1 & 0 & \frac{1}{6} \\ 0 & 1 & \frac{3}{2} \end{array}\right) \Longrightarrow \begin{cases} a_1 = & \frac{1}{6}, \\ a_2 = & \frac{3}{2}. \end{cases}$$

Отже, шуканий многочлен (4.3.8) має вигляд

$$g(x; a_1, a_2) = \frac{1}{6} + \frac{3}{2}x, \quad x \in [0; 4].$$

Тепер шукаємо квадратичний многочлен, що є середньо квадратичними наближеннями даної функції, у вигляді

$$g(x; a_1, a_2, a_3) = a_1 x^2 + a_2 x + a_2, \quad x \in [0; 4], \tag{4.3.8}$$

де значення a_1, a_2, a_3 шукаємо за умов

$$\begin{cases} g(1; a_1, a_2, a_3) & = a_1 + a_2 + a_3 = 2, \\ g(2; a_1, a_2, a_3) & = 4a_1 + 2a_2 + a_3 = 3, \\ g(3; a_1, a_2, a_3) & = 9a_1 + 3a_2 + a_3 = 5. \end{cases}$$

Долі робимо так само, у лінійному випадку. □

Вправи для самостійної роботи

1. Побудувати інтерполяційні многочлени Лагранжа і Ньютона для функції, заданої на відрізку $[1; 8]$ таблицею

i	0	1	2	3
x_i	1	3	4	7
y_i	3	2	6	9

.

2. Інтерполювати кубічним сплайном функцію, задану на відрізку $[0; 12]$ таблицею

i	0	1	2	3
x_i	0	6	9	12
y_i	3	9	6	12

.

3. Методом найменших квадратів для функції, заданої на відрізку $[0; 4]$ таблицею

i	0	1	2
x_i	1	4	5
y_i	3	-2	1

,

побудуйте лінійний і квадратичний многочлени, що є середньо квадратичними наближеннями даної функції.

4.4. Лабораторний практикум з наближення функцій

Нехай маємо впорядковану множину вузлів інтерполювання $\{x_0, x_1, ..., x_n\}$ і відомі значення $\{y_0 = f(x_0), y_1 = f(x_1), ..., y_n = f(x_n)\}$ деякої функції $y = f(x), \quad x \in [a, b]$. Нагадаємо, що задача інтерполювання полягає у наближенні значень функції f на відрізку $[a, b]$ поза вузлами інтерполювання.

У цьому практикумі розглянемо особливості практичного застосування різних методів інтерполювання. Оскільки на практиці не завжди вдається скористатися оцінкою залишкового члена (4.1.9), то для оцінки якості інтерполювання використовуватимемо спеціальні техніки, зокрема графічну візуалізацію засобами бібліотеки `matplotlib`.

4.4.1 Застосування інтерполяційного полінома Лагранжа

Інтерполяційний поліном Лагранжа має вигляд

$$L_n(x) = \sum_{i=0}^{n} f(x_i)\Phi_i(x), \tag{4.4.1}$$

де

$$\Phi_i(x) = \frac{(x - x_0)\dots(x - x_{i-1})(x - x_{i+1})\dots(x - x_n)}{(x_i - x_0)\dots(x_i - x_{i-1})(x_i - x_{i+1})\dots(x_i - x_n)}, \; i = \overline{0, n}. \tag{4.4.2}$$

Пояснення до використання програмного коду

- Підготувати середовище і потрібні функції :
  1. виконати комірку для підготовки середовища
  2. виконати комірку, в якій **визначені** функції `Lagrange_interpolation` і `L_interpolator`
- Обчислити наближені значення конкретної функції :
  1. виконати комірку, де **визначені** функції, які задають вузли інтерполювання і значення конкретної функції у цих вузлах
  2. виконати комірку, в якій задають координати відрізка `[a,b]` і викликають функції для задання даних інтерполювання
  3. якщо треба обчислити:
     - одне наближене значення функції в деякій точці, то виконати комірку з викликом функції `Lagrange_interpolation` з відповідними значеннями аргументів;
     - кілька наближених значень функції в рівновіддалених точках на `[a,b]`, то викликати функцію `L_interpolator` з відповідними значеннями аргументів; у цьому випадку за замовчуванням (при `prnt=True`) будуть побудовані графіки полінома і функції, яку інтерполюють (при `fr=True`), а також буде обчислено найбільше відхилення знайденого полінома від цієї фукції.

Програмна реалізація методу

Підготовка середовища

[1]:
```
#%matplotlib inline
%matplotlib widget
import matplotlib.pyplot as plt
import numpy as np
```

`Lagrange_interpolation` – **функція для обчислення інтерполяційного полінома Лагранжа**

[2]:
```
def Lagrange_interpolation(xv,x,f):
    """
    обчислення полінома в точці xv
       xv - точка, в якій наближують функцію
```

```
        x  - масив вузлів інтерполювання
        f  - масив значень функції у вузлах
    """
    n=x.size
    fxv=0
    for i in range(n):
        lagr=1.
        for j in range(n):
            if (i==j) :
                continue
            lagr*=(xv-x[j])/(x[i]-x[j])
        fxv+=f[i]*lagr
    return fxv
```

L_interpolator – функція, яка реалізує процес інтерполювання

[95]:
```
def L_interpolator(xv,fv,a,b,n,ng,prnt=True,fr=False):
    """наближення на відрізку [a,b] функції f поліномом Лагранжа n-го␣
↪степеня з рівновіддаленими вузлами інтерполювання
        ng - кількість точок, у яких обчислюють значення полінома
    """
    x = xv(a,b,n)
    fx = fv(a,b,n)

    xg=np.linspace(a,b,ng+1)


    pv=np.empty(ng+1)
    i=0
    for xv in xg:
        pv[i]=Lagrange_interpolation(xv,x,fx)
        i+=1

    if prnt== False:
        return pv

    fig = plt.figure(figsize=(8, 5))
    ax = fig.gca()
    ax.axhline(color="grey", ls="--", zorder=-1)
    ax.axvline(color="grey", ls="--", zorder=-1)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    plt.plot(xg, pv, label = '$L_n(x)$')
    ax.legend()
    plt.scatter(x, fx, marker='o')

    if fr==True:
        fg=fv(a,b,ng)
        ax.legend()
        plt.plot(xg, fg,'--', label = '$f(x)$')
        ax.legend()
        eps=np.max(np.abs(pv-fg))
        print(f"eps={eps}")

    return pv
```

Обчислювальні експерименти

Продемонструємо використання інтерполяційного полінома Лагранжа для наближення функцій.

Приклад 1. Проінтерполювати функцію f, задану на відрізку $[a, b] = [0, 5]$ таблицею

i	0	1	2	3
x_i	0	2	3	5
y_i	2	4	6	8

.

Визначимо функції, які повертають вузли інтерполювання та відповідні значення функції. З метою встановлення універсального інтерфейсу для таких функцій, зокрема для забезпечення потрібними даними також при інтерполюванні на n рівновіддалених точках на відрізку $[a, b]$, передбачимо у цих функціях відповідні аргументи:

[4]:
```
def xv1(a=0,b=5,n=3):
    """ встановлення вузлів інтерполювання"""
    return np.array([0,2,3,5], dtype='float32')
```

[5]:
```
def fv1(a=0,b=5,n=3):
    """ задання значень функції у вузлах інтерполювання"""
    return np.array([2,4,6,8], dtype='float32')
```

Задамо також кінці відрізка та кількість вузлів інтерполювання:

[6]:
```
a = 0
b = 5
n = 3
x = xv1(a,b,n)
fx = fv1(a,b,n)
```

Обчислюємо значення інтерполяційного полінома Лагранжа `n`-го степеня в точці `xi`

[7]:
```
xi=1.5
Lagrange_interpolation(xi,x,fx)
```

[7]:
```
3.0999999999999996
```

[8]:
```
xi=2.5
Lagrange_interpolation(xi,x,fx)
```

[8]:
```
5.0
```

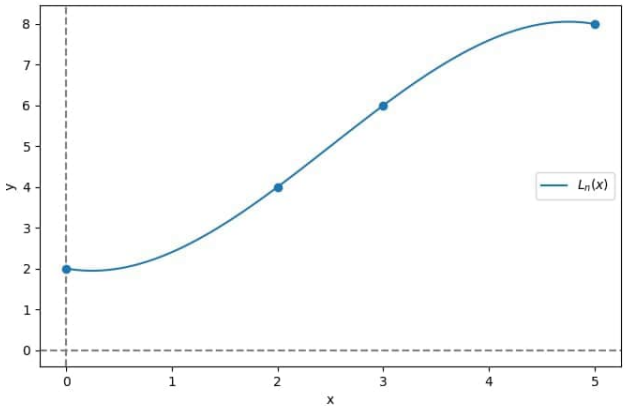
Якщо ж треба обчислити значення інтерполяційного полінома Лагранжа `n`-го степеня в `ng` **рівновіддалених точках** на `[a,b]`, то використовуємо функцію `L_interpolator`, яка визначає необхідні для інтерполювання кроки :

[9]:
```
ng=60
n=3
L_interpolator(xv1,fv1,a,b,n, ng)
```

```
Canvas(toolbar=Toolbar(toolitems=[('Home', 'Reset original view', 'home',␣
↪'home'), ('Back', 'Back to previous ...
```

[9]:
```
array([2.        , 1.9679784 , 1.94938272, 1.94375   , 1.95061728,
       1.9695216 , 2.        , 2.04158951, 2.09382716, 2.15625   ,
       2.22839506, 2.30979938, 2.4       , 2.49853395, 2.60493827,
       2.71875   , 2.83950617, 2.96674383, 3.1       , 3.23881173,
       3.38271605, 3.53125   , 3.68395062, 3.84035494, 4.        ,
       4.16242284, 4.32716049, 4.49375   , 4.6617284 , 4.83063272,
       5.        , 5.16936728, 5.3382716 , 5.50625   , 5.67283951,
       5.83757716, 6.        , 6.15964506, 6.31604938, 6.46875   ,
```

```
6.61728395, 6.76118827, 6.9       , 7.03325617, 7.16049383,
7.28125   , 7.39506173, 7.50146605, 7.6       , 7.69020062,
7.77160494, 7.84375   , 7.90617284, 7.95841049, 8.        ,
8.0304784 , 8.04938272, 8.05625   , 8.05061728, 8.0320216 ,
8.        ])
```



Зазначимо, що функція `L_interpolator` при значеннях аргументів `prnt=True` виводить на графічну панель графік інтерполяційного полінома, а якщо відома формула інтерпольованої функції (як це буде продемонстровано в наступному прикладі), то при `fr=True` будується також графік цієї функції і виводиться максимум відхилення інтерполяційного полінома від заданої функції.

Приклад 2. Проінтерполювати функцію $f(x) = sin(x),\ x \in [0, 2\pi]$ за значеннями у рівновіддалених вузлах $\{x_0, x_1, ..., x_n,\ n \in [0, 2\pi]\}$.

У цьому випадку функції для задання даних інтерполювання можна реалізувати так:

[10]:
```
def xv2(a,b,n):
    x=np.linspace(a,b,n+1)
    return x
```

[11]:
```
def fv2(a,b,n):
    x=np.linspace(a,b,n+1)
    f=np.sin(x)#+1
    return f
```

Після задання координат відрізка і порядку інтерполяційного полінома виконаємо комірку з викликом функції `L_interpolator`, яка реалізує весь алгоритм інтерполювання від задання вхідних даних до побудови відповідних графіків та обчислення похибки інтерполюванні за нормою $\|\cdot\|_\infty$:

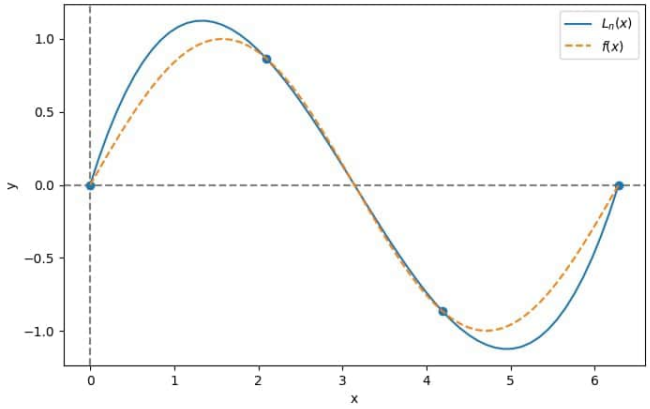
[12]:
```
a=0
b=2*np.pi
n=3
```

[13]:
```
ng=60
pL = L_interpolator(xv2,fv2,a,b,n, ng, fr=True)
```

```
eps=0.2545937399527187
```

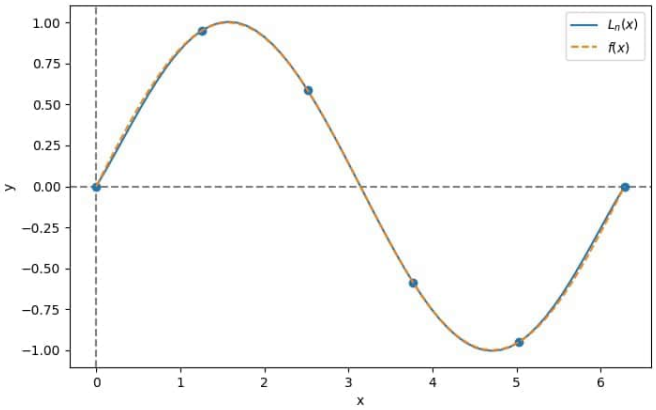
Зазначимо, що параметр `ng` визначає кількість рівновіддалених точок, в яких буде обчислено значення інтерполяційного полінома.

Виконаємо інтерполювання на послідовності значень порядку інтерполяційного полінома:



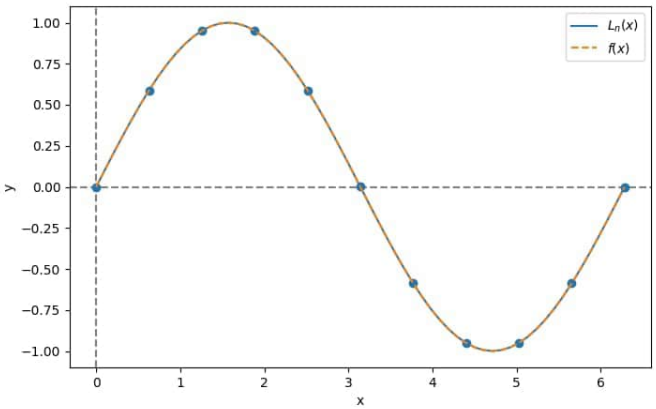
[14]:
```
n = 5
pL = L_interpolator(xv2,fv2,a,b,n, ng, fr=True)
```

```
eps=0.02664282649990979
```



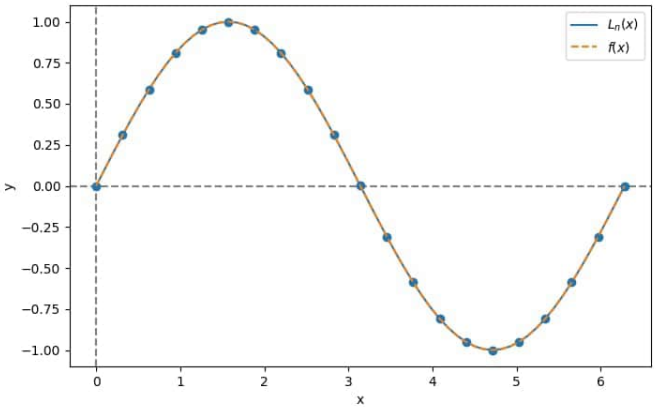
[15]:
```
n = 10
pL = L_interpolator(xv2,fv2,a,b,n, ng, fr=True)
```

```
eps=5.08951732331453e-05
```



[16]:
```
n = 20
pL = L_interpolator(xv2,fv2,a,b,n, ng, fr=True)
```

```
eps=2.823019595865617e-13
```



Як бачимо, відхилення значень полінома від значень заданої функції швидко зменшується в нормі $\|\cdot\|_\infty$ із зростанням кількості вузлів. Як відомо (див.розділ 4.2), при інтерполюванні розглянутими поліномами у випадку рівновіддалених вузлів так буде не для довільної функції. Це демонструє наступний приклад.

Приклад 3.(example 8.1)Проінтерполювати функцію $f(x) = (1+x^2)^{-1}$, $x \in [-5, 5]$ на **рівновіддалених вузлах**.

Підготуємо за аналогією до попереднього прикладу потрібні функції та виконаємо інтерполювання, збільшуючи з деяким кроком значення порядку інтерполяційного полінома:

[17]:
```
def xv3(a,b,n):
    x=np.linspace(a,b,n+1)
    return x
```

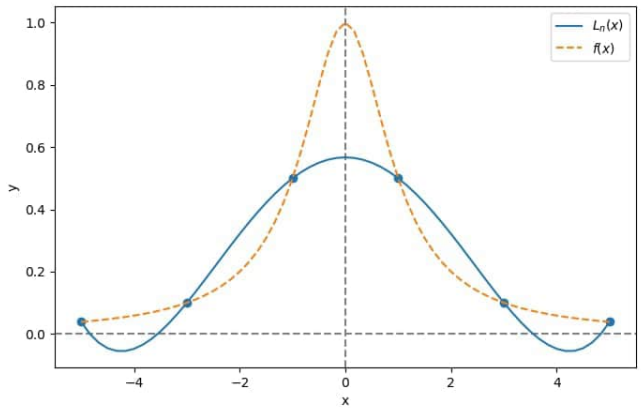
[18]:
```
def fv3(a,b,n):
    x=np.linspace(a,b,n+1)
    f=1/(1+x*x)
```

```
    return f
```

[19]:
```
a=-5
b=5
ng=60
```

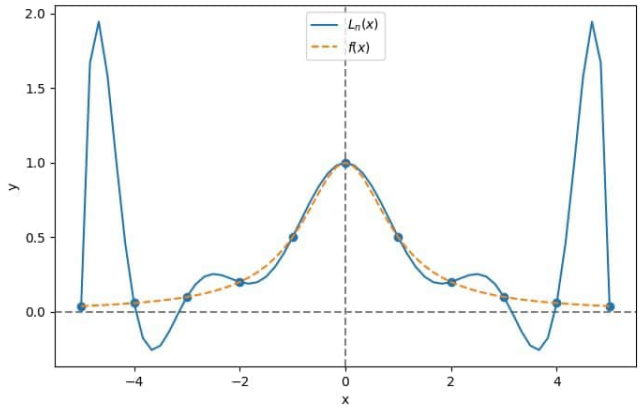
[20]:
```
n = 5
pL = L_interpolator(xv3,fv3,a,b,n, ng, fr=True)
```

eps=0.4326923076923077



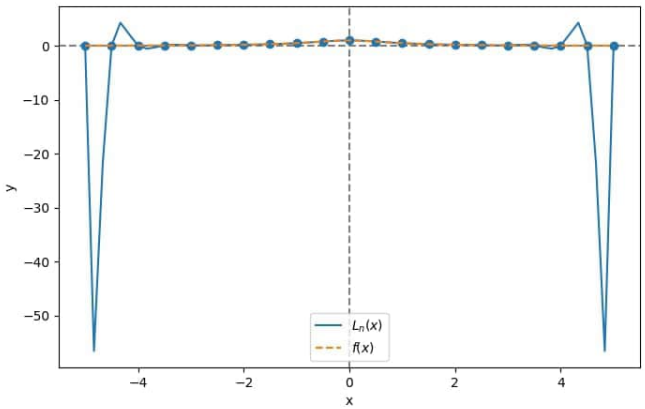
[21]:
```
n = 10
pL = L_interpolator(xv3,fv3,a,b,n, ng, fr=True)
```

eps=1.9007638550703463



[22]:
```
n = 20
pL = L_interpolator(xv3,fv3,a,b,n, ng, fr=True)
```

eps=56.63118692857735



Як бачимо, для заданої функції відхилення значень інтерполяційного полінома зростає поблизу кінців відрізка. Отже обраний спосіб інтерполювання на підходить для даної функції.

[23]:
```
plt.close('all')
```

4.4.2 Інтерполяційний поліном Ньютона

Інтерполяційний поліном Ньютона $L_n(x)$ є такою формою подання інтерполяційного полінома, яка забезпечує його обчислення через відповідне значення $L_{n-1}(x)$:

$$L_n(x) = L_{n-1}(x) + (x - x_0)(x - x_1)\dots(x - x_{n-1})f(x_0; x_1; \dots; x_n),\ n \in \mathbb{N}, \tag{4.4.3}$$

де $f(x_0; x_1; \dots; x_n)$ – розділена різниця n-го порядку.

Розділені різниці обчислюють рекурентно за формулою

$$f(x_{i-1}; x_i; \dots; x_{i+k}) := \frac{f(x_i; x_{i+1}; \dots; x_{i+k}) - f(x_{i-1}; x_i; \dots; x_{i+k-1})}{x_{i+k} - x_{i-1}},\ i = \overline{1, n-k},\ k = \overline{1, n}. \tag{4.4.4}$$

Під час обчислення розділені різниці можна зберігати у такій структурі з рядками змінної довжини

$$\begin{cases} f(x_0) \\ f(x_1) \quad f(x_0; x_1) \\ f(x_2) \quad f(x_1; x_2) \quad f(x_0; x_1; x_2) \\ \dots \quad \dots \\ f(x_n) \quad f(x_{n-1}; x_n) \quad f(x_{n-2}; x_{n-1}; x_n) \quad \dots \quad f(x_0; x_1; \dots; x_n), \end{cases} \tag{4.4.5}$$

У явному вигляді подання (4.4.3) можна переписати так

$L_n(x) = f(x_0) + (x - x_0)f(x_0; x_1) + (x - x_0)(x - x_1)f(x_0; x_1; x_2) + \dots + (x - x_0)(x - x_1)\dots(x - x_{n-1})f(x_0; x_1; \dots; x_n)$.

Пояснення до використання програмного коду

- Підготувати середовище і потрібні функції :
  1. виконати комірку для підготовки середовища
  2. виконати комірки, де **визначені** функції `divided_differences`,`Newton_interpolation` і `N_interpolator`
- Обчислити наближені значення конкретної функції:
  1. виконати комірку, де **визначені** функції, які задають вузли інтерполювання і значення конкретної функції у цих вузлах
  2. виконати комірку, в якій задають координати відрізка `[a,b]` і викликають функції для задання даних інтерполювання і обчислення розділених різниць
  3. якщо треба обчислити:
     - одне наближене значення функції в деякій точці, то виконати комірку з викликом функції `Newton_interpolation` з відповідними значеннями аргументів
     - кілька наближених значень функції в рівновіддалених точках на `[a,b]`, то викликати функцію `N_interpolator` з відповідними значеннями аргументів; у цьому випадку за замовчуванням (при `prnt=True`) будуть побудовані графіки полінома і функції, яку інтерполюють (при `fr=True`), а також буде обчислено найбільше відхилення знайденого полінома від цієї фукції.

Програмна реалізація методу

Підготовка середовища

`divided_differences` – **функція для обчислення розділених різниць функції `f` на масиві точок `xs`**

[2]:
```
def divided_differences(xs,f):
    """обчислення розділених різниць функції f на масиві точок xs """
```

```
    ddm=np.empty(xs.size)
    ddm[0]=f[0]

    dd=list()
    dd.append(f)
    n=xs.size-1

    for k in range(1,n+1):
        fn=np.empty(n-k+1)
        for i in range(n-k+1):
            fn[i]=(dd[k-1][i+1]-dd[k-1][i])/(xs[i+k]-xs[i])
        ddm[k]=fn[0]
        dd.append(fn)
    return ddm
```

`Newton_interpolation` – **функція для обчислення інтерполяційного полінома Ньютона**

[3]:
```
def Newton_interpolation(x,k, xs, ddm):
    """обчислення значення інтерполяційного полінома
       x - точка, в якій обчислюють значення полінома
       k - степінь полінома
       xs - масив вузлів
       ddm - розділені різниці
    """
    pk=ddm[0]
    w=1
    for i in range(1,k+1):
        w*=x-xs[i-1]
        pk+=w*ddm[i]
    return pk
```

`N_interpolator` – **функція, яка реалізує процес інтерполювання**

[30]:
```
def N_interpolator(xv,fv,a,b,n,m,ng,prnt=True,fr=False):
    """наближення заданої таблично функції на відрізку [a,b] поліномом␣
 ↪Ньютона m-го степеня
       xv - функція, яка встановлює вузли інтерполювання
       fv - функція, яка задає табличні значення функції у вузлах␣
 ↪інтерполювання
       ng - кількість точок, у яких обчислюють значення полінома
    """
    x=xv(a,b,n)

    if m>x.size-1:
        print(f" недостатня кількість інтерполяційних вузлів")
        return

    f=fv(a,b,n)
    ddm=divided_differences(x,f)

    xg=np.linspace(a,b,ng+1)

    pm=np.empty(ng+1)
    i=0
    for xi in xg:
        pm[i]=Newton_interpolation(xi,m, x, ddm)
```

```
        i+=1

    if prnt== False:
        return pm

    plt.close('all')

    fig = plt.figure(figsize=(8, 5))
    ax = fig.gca()
    ax.axhline(color="grey", ls="--", zorder=-1)
    ax.axvline(color="grey", ls="--", zorder=-1)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    plt.plot(xg, pm, label = '$L_n(x)$')
    ax.legend()

    plt.scatter(x, f, marker='o')
    if fr==True:
        fg=fv(a,b,ng)
        plt.plot(xg, fg,'--', label = '$f(x)$')
        ax.legend()
        eps=np.max(np.abs(pm-fg))
        print(f"eps={eps}")

    return pm
```

Обчислювальні експерименти

Продемонструємо використання інтерполяційного полінома Ньютона для наближення функцій. Для порівняння з інтерполюванням поліномом Лагранжа розглянемо ті ж самі приклади, що і в попередньому розділі.

Приклад 1. Проінтерполювати функцію f, задану на відрізку $[a, b] = [0, 5]$ таблицею

i	0	1	2	3
x_i	0	2	3	5
y_i	2	4	6	8

Визначимо функції, які повертають вузли інтерполювання та відповідні значення функції:

[5]:
```
def xv1(a,b,n):
    """ встановлення вузлів інтерполювання"""
    return np.array([0,2,3,5], dtype='float32')
```

[6]:
```
def fv1(a,b,n):
    """ задання значень функції у вузлах інтерполювання"""
    return np.array([2,4,6,8], dtype='float32')
```

Задамо також кінці відрізка та кількість вузлів інтерполювання:

[7]:
```
a=0
b=5
n=3
```

Якщо треба обчислити наближені значення функції **в окремих точках**, то спочатку обчислимо за відомими значеннями функцій розділені різниці:

[8]:
```
x1=xv1(a,b,n)
ddm = divided_differences(x1,fv1(a,b,n))
```

і використаємо їх у функції `Newton_interpolation` для побудови інтерполяційного полінома Ньютона `m`-го порядку (`m <= n`) та обчислення його значення в точці `xi`:

[9]:
```
xi=1.5
m=3
Newton_interpolation(xi,m, x1, ddm)
```

[9]: 3.1

Щоб знайти значення інтерполяційного полінома в іншій точці, задаємо її координати і повторюємо виклик `Newton_interpolation` :

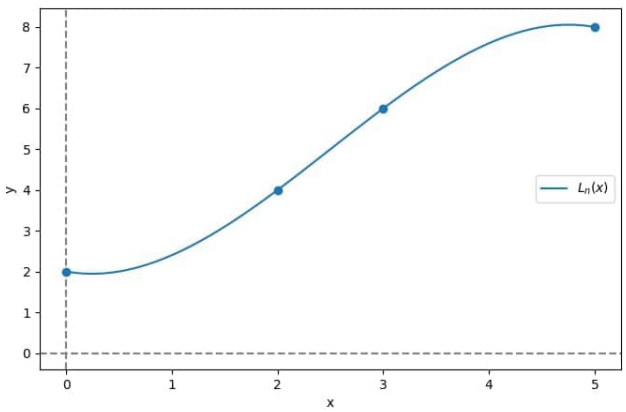
[10]:
```
xi=2.5
Newton_interpolation(xi,m, x1, ddm)
```

[10]: 5.0

Якщо ж треба обчислити значення інтерполяційного полінома Ньютона `m`-го степеня в `ng` **рівновіддалених точках** на `[a,b]`, то використовуємо функцію `N_interpolator`, яка визначає необхідні для інтерполювання кроки :

[11]:
```
m=3
ng=60
N_interpolator(xv1,fv1,a,b,n,m,ng,prnt=True)
```

[11]:
```
array([2.        , 1.9679784 , 1.94938272, 1.94375   , 1.95061728,
       1.9695216 , 2.        , 2.04158951, 2.09382716, 2.15625   ,
       2.22839506, 2.30979938, 2.4       , 2.49853395, 2.60493827,
       2.71875   , 2.83950617, 2.96674383, 3.1       , 3.23881173,
       3.38271605, 3.53125   , 3.68395062, 3.84035494, 4.        ,
       4.16242284, 4.32716049, 4.49375   , 4.6617284 , 4.83063272,
       5.        , 5.16936728, 5.3382716 , 5.50625   , 5.67283951,
       5.83757716, 6.        , 6.15964506, 6.31604938, 6.46875   ,
       6.61728395, 6.76118827, 6.9       , 7.03325617, 7.16049383,
       7.28125   , 7.39506173, 7.50146605, 7.6       , 7.69020062,
       7.77160494, 7.84375   , 7.90617284, 7.95841049, 8.        ,
       8.0304784 , 8.04938272, 8.05625   , 8.05061728, 8.0320216 ,
       8.        ])
```



Зазначимо, що функція `N_interpolator` при значеннях аргументів `prnt=True` виводить на графічну панель графік інтерполяційного полінома, а якщо відома формула інтерпо-

льованої функції (як це буде продемонстровано в наступному прикладі), то при `fr=True` будується графік цієї функції і виводиться максимум відхилення інтерполяційного полінома від заданої функції.

Приклад 2. Проінтерполювати функцію $f(x) = sin(x),\ x \in [0, 2\pi]$ за значеннями у рівновіддалених вузлах $\{x_0, x_1, ..., x_n,\ n \in [0, 2\pi]\}$.

Визначимо функції, які окремо повертають множину рівновіддалених вузлів інтерполювання та відповідні значення функції:

[12]:
```
def xv2(a,b,n):
    x=np.linspace(a,b,n+1)
    return x
```

[13]:
```
def fv2(a,b,n):
    x=np.linspace(a,b,n+1)
    f=np.sin(x)#+1
    return f
```

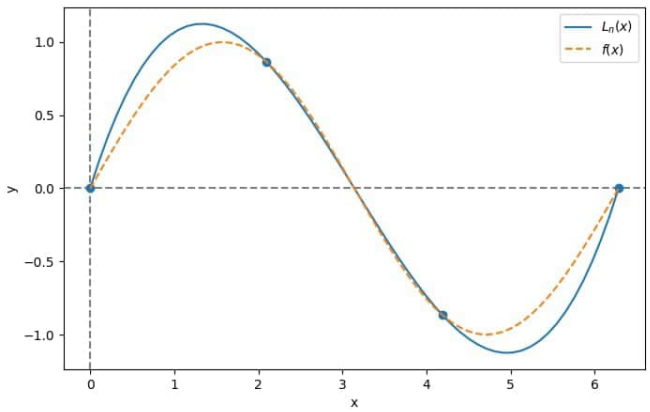
Визначимо координати відрізка інтерполювання та кількість вузлів:

[14]:
```
a=0
b=2*np.pi
n=3
```

Тепер обчислимо значення інтерполяційного полінома Ньютона `m`-го степеня в `ng` точках відрізка:

[15]:
```
m=3
ng=60
pN = N_interpolator(xv2,fv2,a,b,n,m,ng,fr=True)
```

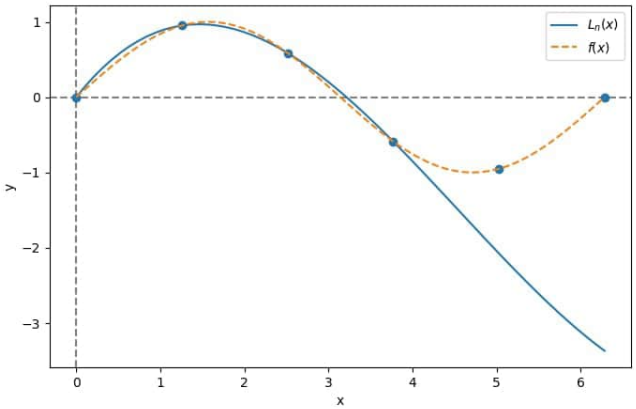
```
eps=0.2545937399527193
```



Розглянемо тепер інтерполяцію поліномами вищого порядку. Зазначимо, що обчислення інтерполяційних поліномів за формулою (4.4.3) не надає суттєвих перевах для інтерполювання на усьому заданому відрізку. Оскільки поліноми меншого порядку використовуватимуть меншу кількість вузлів інтерполювання, то в на інтервалі поза використаними вузлами спостерігатиметься значне відхилення значень полінома від значень заданої функції, що демострує такий наприклад:

[16]:
```
n = 5
m = 3
pN = N_interpolator(xv2,fv2,a,b,n,m,ng,fr=True)
```

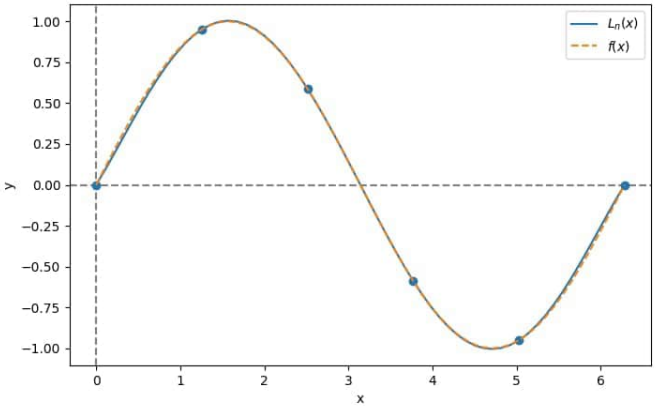
```
eps=3.3677098243468913
```



Тому далі, збільшуючи кількість вузлів, задаватимемо такий самий поряток полінома:

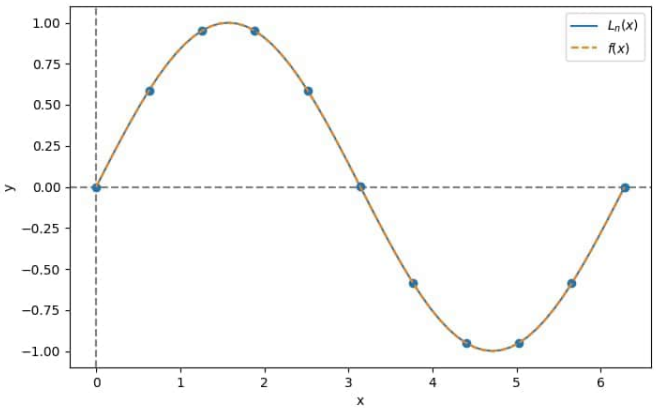
[17]:
```
n = 5
m = 5
pN = N_interpolator(xv2,fv2,a,b,n,m,ng,fr=True)
```

```
eps=0.02664282649990979
```



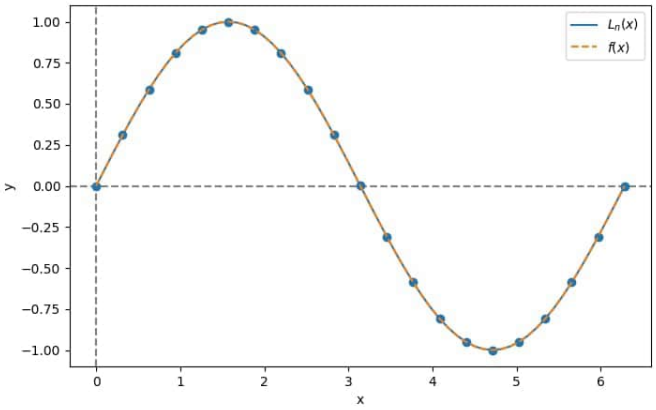
[18]:
```
n = 10
m = 10
pN = N_interpolator(xv2,fv2,a,b,n,m,ng,fr=True)
```

```
eps=5.0895173235671054e-05
```



[19]:
```
n = 20
m = 20
pN = N_interpolator(xv2,fv2,a,b,n,m,ng,fr=True)
```

```
eps=1.3457290837237679e-13
```



Як бачимо, відхилення значень полінома від значень заданої функції швидко зменшується в нормі $\|\cdot\|_\infty$ із зростанням кількості вузлів. Однак така картина спостерігатиметься не для довільної функції. Наступний приклад демонструє (як і у випадку інтерполювання поліномами Лагранжа) значні похибки при інтерполюванні поліномом Ньютона у випадку рівновіддалених вузлів.

Приклад 3.(example 8.1)Проінтерполювати функцію $f(x) = (1+x^2)^{-1},\ x \in [-5,5]$ на **рівновіддалених вузлах**.

Виконаємо аналогічні кроки, що і в попередньому прикладі і розглянемо інтерполювання при зростанні кількості рівновіддалених вузлів:

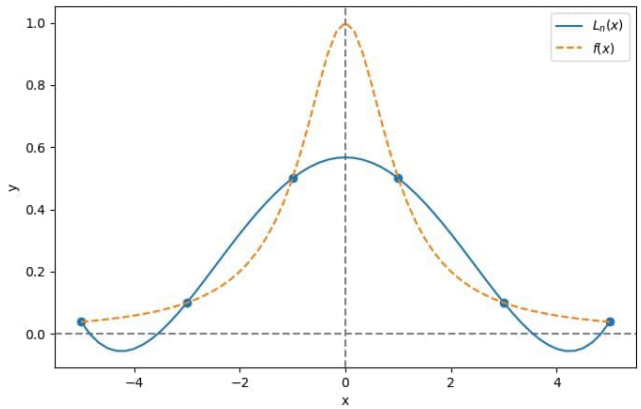
[20]:
```
def xv3(a,b,n):
    x=np.linspace(a,b,n+1)
    return x
```

[21]:
```
def fv3(a,b,n):
    x=np.linspace(a,b,n+1)
    f=1/(1+x*x)
    return f
```

[22]:
```
a=-5
b=5
```

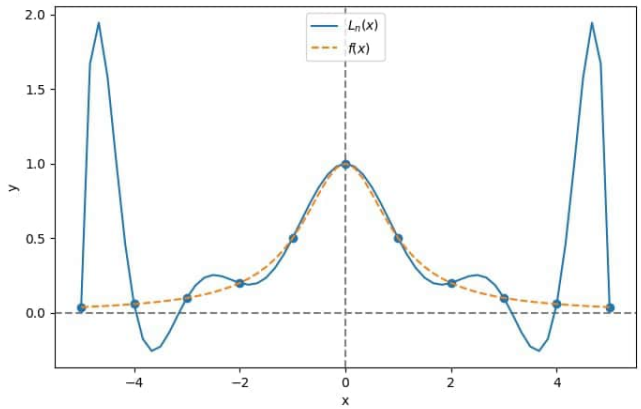
[23]:
```
n = 5
m = 5
pN = N_interpolator(xv3,fv3,a,b,n,m,ng,fr=True)
```

```
eps=0.4326923076923076
```



[24]:
```
n = 10
m = 10
pN = N_interpolator(xv3,fv3,a,b,n,m,ng,fr=True)
```

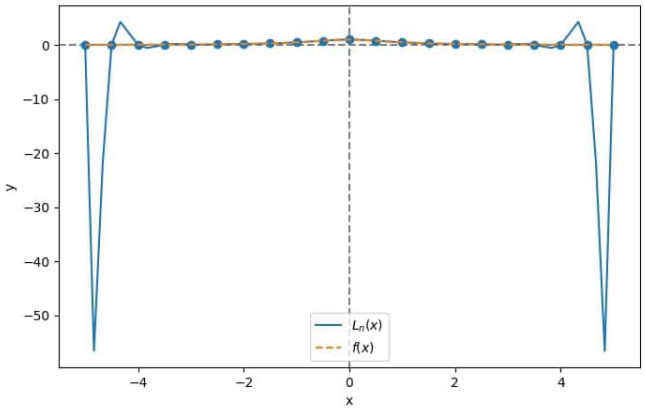
```
eps=1.9007638550703447
```



[25]:
```
n = 20
m = 20
```

```
pN = N_interpolator(xv3,fv3,a,b,n,m,ng,fr=True)
```

```
eps=56.63118692857639
```



Зазначимо, що отримані похибки інтерполяції відрізняються від випадку інтерполювання поліномами Лагранжа лише 1-2 цифрами. Як і раніше, для заданої функції відхилення значень інтерполяційного полінома зростає поблизу кінців відрізка.

[ ]:
```
plt.close('all')
```

4.4.3 Інтерполяція кубічними сплайнами

Нехай на відрізку $[a,b]$ задано сітку $\{x_0, x_1, \ldots, x_n \,|\, a = x_0 < x_1 < \ldots < x_n = b\}$, у вузлах якої задано значення $\{y_0, y_1, \ldots, y_n\}$ функції $f \in C[a,b]$.

Інтерполяційний кубічний (нормальний) сплайн має вигляд

$$g(x) := m_{i-1}\frac{(x_i - x)^3}{6h_i} + m_i\frac{(x - x_{i-1})^3}{6h_i} + \left[y_{i-1} - \frac{m_{i-1}h_i^2}{6}\right]\frac{x_i - x}{h_i} + \\ + \left[y_i - \frac{m_i h_i^2}{6}\right]\frac{x - x_{i-1}}{h_i}, \quad x \in [x_{i-1}, x_i], \quad i = \overline{1,n}, \quad (4.4.6)$$

де $h_i := x_i - x_{i-1},\ i = \overline{1,n},\ m_0 = 0,\ m_n = 0$, а сталі $m_i,\ i = \overline{1,n-1}$, знаходять із системи рівнянь

$$\frac{h_i}{6}m_{i-1} + \frac{h_i + h_{i+1}}{3}m_i + \frac{h_{i+1}}{6}m_{i+1} = \frac{y_{i-1}}{h_i} - \left[\frac{1}{h_i} + \frac{1}{h_{i+1}}\right]y_i + \frac{y_{i+1}}{h_{i+1}}, \quad i = \overline{1,n-1}.$$

Якщо домножити кожне i-е рівняння цієї системи на вираз $\alpha_i := 6/(h_i + h_{i+1})$, то отримаємо СЛАР

$$Hm = Dy.$$

Тут A і D – тридіагональні матриці розміром $(n-1)\times(n-1)$ і $(n-1)\times(n+1)$ відповідно, елементи яких обчислюють за формулами

$$a_{i,i} = 2,\ a_{i+1,i} = \frac{h_{i+1}}{(h_{i+1} + h_{i+2})},\ a_{i,i+1} = \frac{h_{i+1}}{(h_i + h_{i+1})},$$

$$d_{i,i} = \frac{6}{h_i(h_i + h_{i+1}))},\ d_{i,i+1} = -\frac{6}{h_i h_{i+1}},\ d_{i,i+2} = \frac{6}{h_{i+1}(h_i + h_{i+1})}, i = \overline{1,n-1}.$$

Пояснення до використання програмного коду

- Підготувати середовище і потрібні функції :
  1. виконати комірку для підготовки середовища
  2. виконати комірки, в яких **визначені** функції `spline3_interpolation`, `spline3_interpolator`, `calculate_m`, `TDMA_solver`, `set_matrix_diagonals`, `set_vector` і `hv`
- Обчислити наближені значення конкретної функції :
  1. виконати комірку, де **визначені** функції, які задають вузли інтерполювання і значення конкретної функції у цих вузлах
  2. виконати комірку, в якій задають координати відрізка `[a,b]` і викликають функції для задання даних інтерполювання і обчислення коефіцієнтів сплайна
  3. якщо треба обчислити:
     - одне наближене значення функції в деякій точці, то виконати комірку з викликом функції `spline3_interpolation` з відповідними значеннями аргументів;
     - кілька наближених значень функції в рівновіддалених точках на `[a,b]`, то викликати функцію `spline3_interpolator` з відповідними значеннями аргументів; у цьому випадку за замовчуванням (при `prnt=True`) будуть побудовані графіки сплайна і функції, яку інтерполюють (при `fr=True`), а також буде обчислено найбільше відхилення знайденого сплайна від цієї фукції.

Програмна реалізація методу

[1]:
```
#%matplotlib inline
%matplotlib widget
import matplotlib.pyplot as plt
import numpy as np
```

spline3_interpolation – функція для обчислення інтерполяційного кубічного сплайна

[2]:
```
def spline3_interpolation(xp,x,y,h,m):
    n = x.size -1
    if xp < x[0] or xp > x[n]:
        raise Exception('точка поза відрізком')
    i=1
    while (i <= n) and (xp > x[i]):
        i+=1

    g = m[i-1] * (x[i]-xp)**3 /(6*h[i-1])
    g += m[i] * (xp-x[i-1])**3 /(6*h[i-1]) + (y[i-1]-m[i-1]*h[i-1]**2/6) *␣
↪(x[i]-xp)/h[i-1]
    g += (y[i]-m[i]*h[i-1]**2/6) * (xp -x[i-1])/h[i-1]

    return g
```

spline3_interpolator – функція, яка реалізує процес інтерполювання кубічним сплайном

[3]:
```
def spline3_interpolator(xv,fv,a,b,n,ng,prnt=True,fr=False):
    """наближення на відрізку [a,b] функції f кубічними сплайнами з␣
↪рівновіддаленими вузлами інтерполювання
        ng - кількість точок, у яких обчислюють значення полінома
    """
    x = xv(a,b,n)
    y = fv(a,b,n)
    h = hv(x)
    m=calculate_m(x,y,h)

    xg=np.linspace(a,b,ng+1)


    pv=np.empty(ng+1)
    i=0
    for xp in xg:
        pv[i]=spline3_interpolation(xp,x,y,h,m)
        i+=1

    if prnt== False:
        return pv

    fig = plt.figure(figsize=(8, 5))
    ax = fig.gca()
    ax.axhline(color="grey", ls="--", zorder=-1)
    ax.axvline(color="grey", ls="--", zorder=-1)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    plt.plot(xg, pv, label = '$g(x)$')
    ax.legend()
    plt.scatter(x, y, marker='o')

    if fr==True:
        fg=fv(a,b,ng)
        plt.plot(xg, fg,'--', label = '$f(x)$')
```

```
        ax.legend()
        eps=np.max(np.abs(pv-fg))
        print(f"eps={eps}")

    return pv
```

calculate_m – **функція, яка реалізує обчислення коефіцієнтів сплайна**

[4]:
```
def calculate_m(x,y,h):
    nel=x.size
    n=nel-1
    n2=x.size-3
    m2=TDMA_solver(n2,set_matrix_diagonals,set_vector,h,y)
    m=np.empty(x.size, dtype='float32')
    m[0]=0
    m[n]=0
    for i in range(1,n):
        m[i]=m2[i-1]
    return m
```

TDMA_solver – **функція, яка реалізує метод прогонки для розв'язування СЛАР**

[5]:
```
def TDMA_solver(n,set_matrix_diagonals,set_vector,h,y):
    """ метод прогонки для розв'язування СЛАР
        з 3-діагональною матрицею
        n+1 - кількість невідомих; n- максимальне значення індексу
        вектор c-головна діагональ
        вектори a і b - нижня і верхня діагоналі, паралельні головній
        вектор g - вільні члени
      """
    c,a,b = set_matrix_diagonals(n,h)
    g = set_vector(n,h,y)

    alpha = np.empty(n+1, dtype=float )
    beta  = np.empty(n+1, dtype=float )

    if c[0] !=0 :
        alpha[0] =-b[0]/c[0]
        beta [0] = g[0]/c[0]
    else:
        raise Exception('c[0]==0')

    for i in range(1,n+1):
        w=a[i]*alpha[i-1]+c[i]
        if w != 0 :
            alpha[i] =-b[i]/w
            beta[i]  = (g[i] - a[i]*beta[i-1])/w
        else:
            raise Exception('w==0')

    x = np.empty(n+1, dtype=float )
    x[n] = beta[n]
    for i in range(n-1,-1,-1):
        x[i] = alpha[i]*x[i+1] + beta[i]
    return x
```

`set_matrix_diagonals` – **функція, яка обчислює діагоналі матриці СЛАР**

[6]:
```
def set_matrix_diagonals(n,h):
    """ функція задає 3 діагоналі матриці СЛАР
    n - останнє значення індексу
    """
    #n=h.size-1
    c=np.full(n+1,2)

    a=np.empty(n+1)
    a[0]=0
    b=np.empty(n+1)
    b[n]=0
    for i in range(1,n+1):
        a[i]=h[i] /(h[i-1]+h[i])
        b[i-1]=h[i-1] /(h[i-1]+h[i])
    return c,a,b
```

`set_vector` – **функція, яка обчислює вектор вільних членів СЛАР**

[7]:
```
def set_vector(n,h,y):
    d = np.empty(n+1, dtype='float32')
    for i in range(1,n+2):
        d[i-1] =6/(h[i-1]+h[i]) * ( (y[i+1]-y[i])/h[i] - (y[i]-y[i-1])/
 ↪h[i-1])
    return d
```

`hv` – **функція, яка обчислює відрізки між вузлами**

[8]:
```
def hv(x):
    """ визначення відрізків між вузлами"""
    n=x.size
    h=np.empty(n-1)
    for i in range(n-1):
        h[i]=x[i+1]-x[i]
    return h
```

Обчислювальні експерименти

Продемонструємо використання кубічного сплайна для інтерполяції функцій.

Приклад 1. Проінтерполювати функцію f, задану на відрізку $[a, b] = [0, 9]$ таблицею

i	0	1	2	3
x_i	0	3	6	9
y_i	12	6	9	3

.

Визначимо функції, які повертають вузли інтерполювання та відповідні значення функції:

[9]:
```
def xv1(a,b,n):
    """ встановлення вузлів інтерполювання"""
    return np.array([0,3,6,9], dtype='float32')
```

[10]:
```
def yv1(a,b,n):
    """ задання значень функції у вузлах інтерполювання"""
    return np.array([12,6,9,3], dtype='float32')
```

Задамо дані інтерполяційної задачі і обчислимо коефіцієнти сплайна:

[11]:
```
a=0
b=9
n=3

x = xv1(a,b,n)
y = yv1(a,b,n)

h=hv(xv1(a,b,n))

m=calculate_m(x,y,h)
```

Тепер можна обчислювати значення сплайна в довільній точці відрізку $[a, b]$:

[12]:
```
xp = 1
spline3_interpolation(xp,x,y,h,m)
```

[12]: 9.11111111111111

[13]:
```
xp = 4
spline3_interpolation(xp,x,y,h,m)
```

[13]: 6.77777777777778

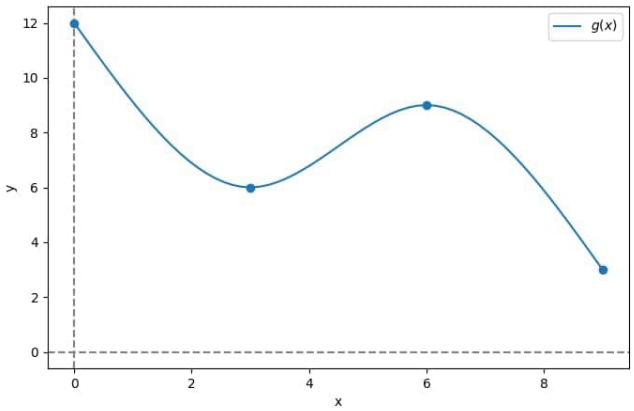
[14]:
```
xp = 7
spline3_interpolation(xp,x,y,h,m)
```

[14]: 8.11111111111111

Якщо ж треба обчислити значення сплайна у `ng` **рівновіддалених точках** на `[a,b]`, то використовуємо функцію `spline3_interpolator`, яка визначає необхідні для інтерполювання кроки:

[15]:
```
a=0
b=9
n=3
```

[16]:
```
ng=60
ps = spline3_interpolator(xv1,yv1,a,b,n, ng)
```



Приклад 2. Проінтерполювати функцію $f(x) = sin(x)$, $x \in [0, 2\pi]$ за значеннями у рівновіддалених вузлах $\{x_0, x_1, ..., x_n,\ n \in [0, 2\pi]\}$.

Визначимо функції, які повертають множину рівновіддалених вузлів інтерполювання та відповідні значення функції:

[17]:
```
def xv2(a,b,n):
    x=np.linspace(a,b,n+1)
    return x
```

[18]:
```
def yv2(a,b,n):
    x=np.linspace(a,b,n+1)
    f=np.sin(x)#+1
    return f
```

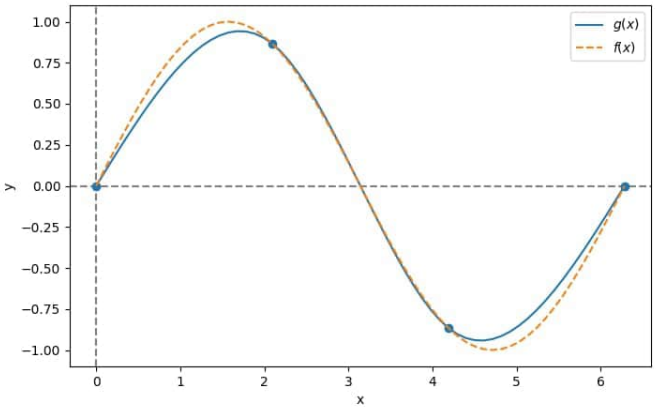
Визначимо координати відрізка інтерполювання:

[19]:
```
a=0
b=2*np.pi
```

і виконаємо інтерполювання, обчислюючи при цьому значення сплайна в ng точках відрізка для різних значень параметра n:

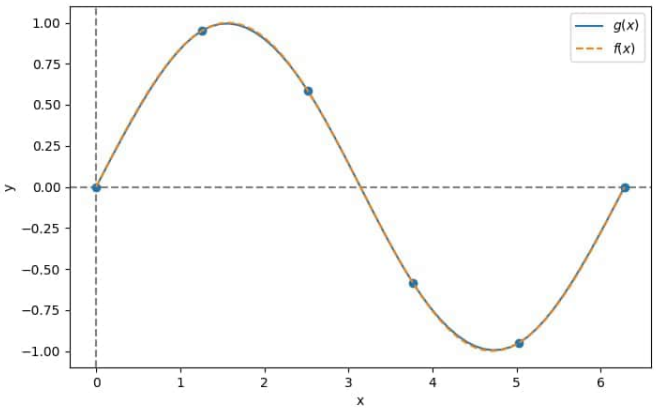
[20]:
```
ng=60
n=3
ps = spline3_interpolator(xv2,yv2,a,b,n, ng, fr=True)
```

```
eps=0.10851070178281641
```



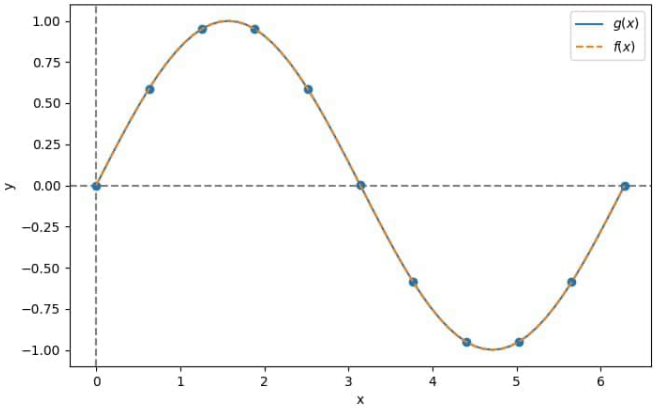
[21]:
```
n=5
ps = spline3_interpolator(xv2,yv2,a,b,n, ng, fr=True)
```

```
eps=0.008946309828320342
```



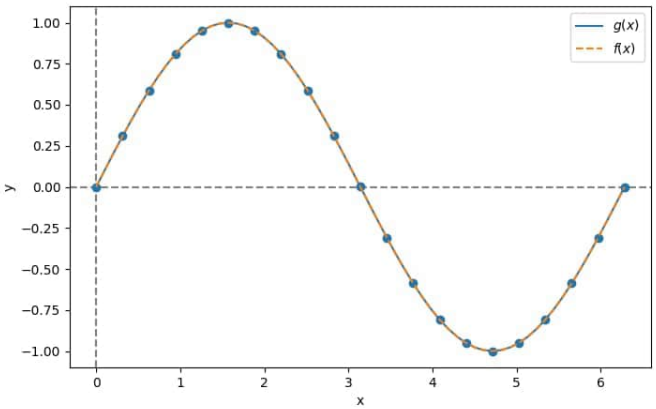
[22]:
```
n=10
ps = spline3_interpolator(xv2,yv2,a,b,n, ng, fr=True)
```

```
eps=0.0004472593032810446
```



[23]:
```
n=20
ps = spline3_interpolator(xv2,yv2,a,b,n, ng, fr=True)
```

```
eps=2.0516548986826422e-05
```



Відзначимо швидке зменшення в нормі $\|\cdot\|_\infty$ відхилення сплайна від заданої функції із зростанням кількості частин, на які розділено відрізок $[a,b]$. Разом з тим, для заданої функції інтерполювання поліномами Лагранжа і Ньютона виявилося точнішим при співмірній кількості вузлів інтерполювання. В той же час, інтерполювання сплайнами має перевагу над згаданими поліномами, яку продемонструємо на прикладі.

Приклад 3.(example 8.1)Проінтерполювати функцію $f(x) = (1+x^2)^{-1},\ x \in [-5,5]$ на **рівновіддалених вузлах**.

Нагадаємо, що інтерполяція заданої функції поліномами Лагранжа і Ньютона на рівновіддалених вузлах була незадовільною поблизу кінців відрізка. Покажемо, що інтерполяція кубічним сплайном не має такого недоліку.

Виконаємо аналогічні кроки, що і в попередньому прикладі і розглянемо інтерполювання при зростанні кількості рівновіддалених вузлів:

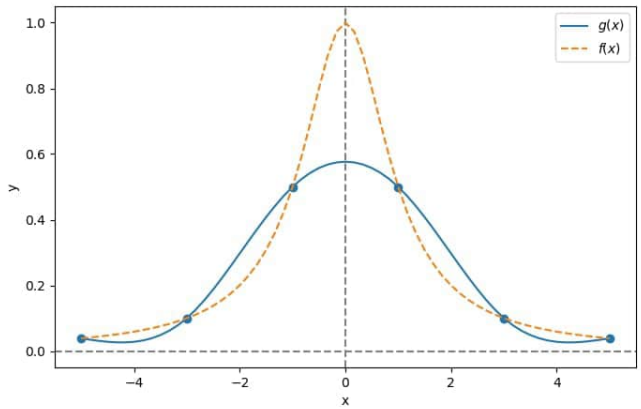
[24]:
```
def xv3(a,b,n):
    x=np.linspace(a,b,n+1)
    return x
```

[25]:
```
def yv3(a,b,n):
    x=np.linspace(a,b,n+1)
    f=1/(1+x*x)
    return f
```

[26]:
```
a=-5
b=5
ng=60
```

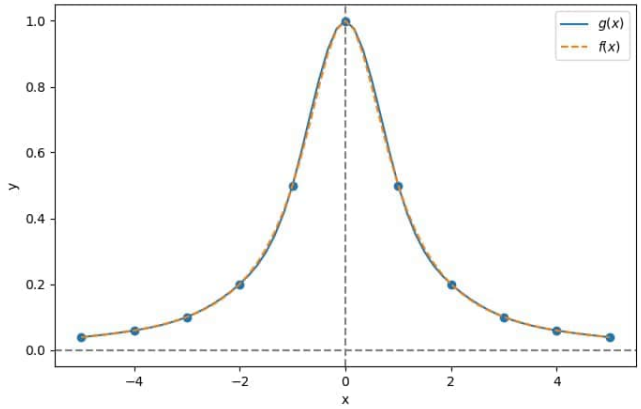
[27]:
```
n = 5
ps = spline3_interpolator(xv3,yv3,a,b,n, ng, fr=True)
```

```
eps=0.4234817773103714
```



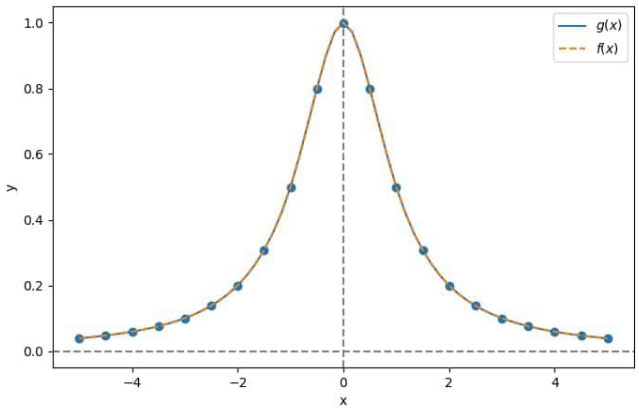
[28]:
```
n = 10
ps = spline3_interpolator(xv3,yv3,a,b,n, ng, fr=True)
```

```
eps=0.02091373773495131
```



[29]:
```
n = 20
ps = spline3_interpolator(xv3,yv3,a,b,n, ng, fr=True)
```

```
eps=0.0028346638974414695
```



Як бачимо, для заданої функції відхилення значень сплайна зменшується при збільшенні кількості вузлів інтерполювання. Звідси може зробити висновок, що для задачі інтерполювання може виявитися успішним підхід, коли весь відрізок задання функції розділяють на частини, на кожній з яких інтерполюють поліномами невисокого порядку.

[30]:
```
plt.close('all')
```

Розділ 5

Чисельне диференціювання та інтегрування

5.1. Чисельне диференціювання

Задача чисельного диференціювання формулюється так: за заданими значеннями функції $y = f(x)$, $x \in [a, b]$, в точках $x_i \in [a, b]$, $i = \overline{0, n}$, і заданими $x \in [a, b]$ та $k \in \mathbb{N}$ знайти наближення значення $f^{(k)}(x)$ і оцінити похибку.

Найпростіші формули чисельного диференціювання дістають за допомогою диференціювання інтерполяційних многочленів. Зобразимо функцію f через інтерполяційний многочлен Ньютона

$$f(x) = L_n(x) + R_n(x), \tag{5.1.1}$$

де

$$\begin{aligned} L_n(x) &:= f(x_0) + (x - x_0)f(x_0; x_1) + \ldots + \\ &+ (x - x_0)(x - x_1)\ldots(x - x_{n-1})f(x_0; x_1; \ldots; x_n), \\ R_n(x) &:= (x - x_0)(x - x_1)\ldots(x - x_n)f(x; x_0; \ldots; x_n) \equiv \\ &\equiv \omega_{n+1}(x)f(x; x_0; \ldots; x_n), \end{aligned}$$

де

$$\omega_{n+1}(x) := (x - x_0)\ldots(x - x_n), \quad x \in [a, b].$$

З (5.1.1) одержуємо таке співвідношення для похідної k-го порядку

$$f^{(k)}(x) = L_n^{(k)}(x) + R_n^{(k)}(x),$$

де похідна від залишкового члена за допомогою формули Лейбніца зображається так

$$R_n^{(k)}(x) = \sum_{i=0}^{k} C_k^i f^{(i)}(x; x_0; \ldots; x_n)\omega_{n+1}^{(k-i)}(x), \tag{5.1.2}$$

де $C_k^i = \dfrac{k!}{i!(k-i)!}$, $i = \overline{0, k}$.

За означенням розділеної різниці з кратними вузлами маємо

$$f^{(q)}(x; x_0; \ldots; x_n) = f(x; \ldots; x; x_0; \ldots; x_n)q!, \quad q \in \mathbb{N}.$$

Отже, співвідношення (5.1.2) можна записати у вигляді

$$R_n^{(k)}(x) = \sum_{i=0}^{k} C_k^i i! f(x; \ldots; x; x_0; \ldots; x_n)\omega_{n+1}^{(k-i)}(x). \tag{5.1.3}$$

Виражаючи розділену різницю через похідну, дістаємо оцінку

$$|R_n^{(k)}(x)| \le \sum_{i=0}^{k} \frac{k!}{(k-i)!(n+i+1)!} \max_{\xi\in[y_1,y_2]} |f^{(n+i+1)}(\xi)| \cdot |\omega_{n+1}^{(k-i)}(x)|,$$
$$y_1 = \min\{x, x_0, \ldots, x_n\}, \quad y_1 = \max\{x, x_0, \ldots, x_n\}.$$

5.2. Чисельне інтегрування

Якщо функція $f(x)$, $x \in [a,b]$, є неперервною і відома її первісна $F(x)$, $x \in [a,b]$, то для обчислення визначеного інтеграла $\int_a^b f(x)\,dx$ можна використати формулу Ньютона-Лейбніца

$$\int_a^b f(x)dx = F(b) - F(a).$$

Однак ця формула мало придатна для практики, бо клас функцій, для яких первісна виражається через елементарні функції, є дуже вузьким. Крім цього, на практиці підінтегральна функція часто задається табличною і тоді саме поняття первісної втрачає зміст. Тому важливого значення набувають чисельні методи обчислення визначених інтегралів.

Розглянемо кілька методів, які дають змогу знаходити наближення значення інтеграла $\int_a^b f(x)\,dx$ для функцій f із широких класів. Суть цих методів полягає в обчисленні значення інтеграла на основі значень підінтегральної функції в скінченій кількості точок відрізку $[a,b]$. А точніше, будемо розглядати методи, в яких

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n} A_i^{(n)} f(x_i), \tag{5.2.1}$$

де $a \le x_0 < x_1 < \ldots < x_n \le b$, а $A_i^{(n)}$, $i = \overline{0,n}$, — сталі.

Формулу (5.2.1) називають ***квадратурною формулою***, точки x_i, $i = \overline{0,n}$, — ***вузлами*** або ***абсцисами квадратурної формули***, а числа $A_i^{(n)}$, $i = \overline{0,n}$, — ***коефіцієнтами квадратурної формули***.

Величину

$$\widehat{R}_n(f) := \int_a^b f(x)dx - \sum_{k=0}^{n} A_i^{(n)} f(x_i)$$

називають ***залишковим членом квадратурної формули***. При цьому питання оцінки $\widehat{R}_n(f)$ має сенс лише у тому разі, якщо функція f задана аналітично.

Ввівши поняття залишкового члена, квадратурну формулу запишемо у вигляді:

$$\int_a^b f(x)\,dx = \sum_{k=0}^{n} A_k^{(n)} f(x_k) + \widehat{R}_n(f). \tag{5.2.2}$$

5.2.1. Загальна квадратурна формула інтерполяційного типу

Для побудови квадратурних формул найчастіше використовують інтерполяційний многочлен Лагранжа. У результаті отримують так звані квадратурні формули інтерполяційного типу.

Розглянемо загальну схему побудови квадратурної формули (5.2.2), використовуючи інтерполяційний многочлен Лагранжа. Для цього запишемо функцію f у вигляді

$$f(x) = L_n(x) + R_n(x), \quad x \in [a,b], \tag{5.2.3}$$

де L_n — інтерполяційний многочлен Лагранжа, побудований для функції f за вузлами інтерполювання $x_0, x_1, ..., x_n$:

$$L_n(x) = \sum_{i=0}^{n} f(x_i) \frac{(x-x_0)\dots(x-x_{i-1})(x-x_{i+1})\dots(x-x_n)}{(x_i-x_0)\dots(x_i-x_{i-1})(x_i-x_{i+1})\dots(x_i-x_n)},$$

а R_n — його залишковий член, причому, якщо $f \in C^{n+1}[a,b]$, то

$$R_n(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}\omega_{n+1}(x), \quad x \in [a,b],$$

де $\omega_{n+1}(x) := (x-x_0)\dots(x-x_n)$, $\xi(x) \in [a,b]$.

Проінтегрувавши рівність (5.2.3) у межах від a до b, одержимо

$$\int_a^b f(x)\,dx = \sum_{i=0}^{n} f(x_i) \int_a^b \frac{(x-x_0)\dots(x-x_{i-1})(x-x_{i+1})\dots(x-x_n)}{(x_i-x_0)\dots(x_i-x_{i-1})(x_i-x_{i+1})\dots(x_i-x_n)}\,dx+$$

$$+\widehat{R}_n(f),$$

де у випадку $f \in C^{n+1}[a,b]$ маємо

$$\widehat{R}_n(f) = \frac{1}{(n+1)!}\int_a^b f^{(n+1)}(\xi(x))\omega_{n+1}(x)\,dx.$$

Позначивши

$$A_i^{(n)} = \int_a^b \frac{(x-x_0)\dots(x-x_{i-1})(x-x_{i+1})\dots(x-x_n)}{(x_i-x_0)\dots(x_i-x_{i-1})(x_i-x_{i+1})\dots(x_i-x_n)}\,dx, \quad i = 0,1,...,n, \tag{5.2.4}$$

отримаємо формулу (5.2.2).

Отже, ми одержали квадратурну формулу (5.2.2), в якій коефіцієнти $A_i^{(n)}$, $i = \overline{0,n}$, обчислюються за формулою (5.2.4). Таку квадратурну формулу називають ***квадратурною формулою інтерполяційного типу***.

Слід зазначити, що коефіцієнти $A_i^{(n)}$, $i = \overline{0,n}$, при заданому розподілі вузлів інтерполювання не залежать від вибору функції f. Крім цього, якщо f є алгебраїчним многочленом степеня не вище n, то залишковий член квадратурної формули $\widehat{R}_n(f) = 0$. Справді, в цьому випадку отримуємо, що многочлен $L_n - f$ степеня не вище n набуває в $(n+1)$-й точці $x_0, x_1, ..., x_n$ значення нуль. А це можливо тільки тоді, коли $L_n - f \equiv 0$, тобто $L_n \equiv f$.

5.2.2. Квадратури Ньютона-Котеса: квадратурні формули прямокутників, трапецій і парабол (Сімпсона)

Розглянемо квадратурні формули інтерполяційного типу у випадку, якщо підінтегральна функція замінюється інтерполяційним многочленом Лагранжа, побудованим за рівновіддаленими вузлами інтерполювання. Такі квадратурні формули називають ***формулами Ньютона-Котеса***.

На практиці найчастіше використовуються формули Ньютона-Котеса, побудованими за допомогою інтерполяційних многочленів Лагранжа степенів $n = 0, 1, 2$. Ці формули відповідно мають назви: ***формула прямокутників, формула трапецій, формула парабол (Сімпсона)***. Розглянемо докладніше ці часткові випадки формули Ньютона-Котеса. Зважаючи на їхню практичну важливість, виведемо ці формули незалежно від загальних міркувань.

5.2.2.1 Формула прямокутників

При $n = 0$ використовується лише один вузол інтерполювання $x_0 = (a+b)/2$. Функція f на проміжку $[a,b]$ замінюється інтерполяційним многочленом нульового степеня

$$L_0(x) = f\left(\frac{a+b}{2}\right).$$

Тому

$$\int_a^b f(x)\,dx = \int_a^b L_0(x)\,dx + \widehat{R}_0(f),$$

звідки

$$\int_a^b f(x)\,dx = (b-a)f\left(\frac{a+b}{2}\right) + \widehat{R}_0(f). \tag{5.2.5}$$

Одержану квадратурна формула (5.2.5) називають ***малою квадратурною формулою прямокутників***.

Якщо $f \in C^2[a,b]$, то для залишкового члена $\widehat{R}_0(f)$ можна отримати простий вираз. Справді, зобразимо функцію f в околі точки $\frac{a+b}{2}$ за формулою Тейлора. Матимемо

$$f(x) = f\left(\frac{a+b}{2}\right) + f'\left(\frac{a+b}{2}\right)\left(x - \frac{a+b}{2}\right) + \frac{f''(\xi)}{2!}\left(x - \frac{a+b}{2}\right)^2, \tag{5.2.6}$$

де $\xi = \xi(x) \in (a,b)$.

Інтегруючи рівність (5.2.6), одержуємо

$$\int_a^b f(x)\,dx = (b-a)f\left(\frac{a+b}{2}\right) + f'\left(\frac{a+b}{2}\right)\int_a^b \left(x - \frac{a+b}{2}\right)dx +$$

$$+\frac{1}{2}\int_a^b \left(x - \frac{a+b}{2}\right)^2 f''(\xi(x))\,dx.$$

Звідси, позаяк

$$\int_a^b \left(x - \frac{a+b}{2}\right)dx = 0,$$

отримаємо

$$\int_a^b f(x)dx = (b-a)f\left(\frac{a+b}{2}\right) + \frac{1}{2}\int_a^b \left(x - \frac{a+b}{2}\right)^2 f''(\xi(x))\,dx.$$

Тому

$$\widehat{R}_0(f) = \frac{1}{2}\int_a^b \left(x - \frac{a+b}{2}\right)^2 f''(\xi(x))\,dx.$$

Оскільки f'' — неперервна функція на $[a,b]$, а $\left(x - \frac{a+b}{2}\right)^2$ не змінює знаку на $[a,b]$, то на основі теореми про середнє значення визначеного інтеграла існує $\theta \in (a,b)$ таке, що

$$\widehat{R}_0(f) = \frac{1}{2}f''(\theta)\int_a^b \left(x - \frac{a+b}{2}\right)^2 dx,$$

звідки

$$\widehat{R}_0(f) = \frac{(b-a)^3}{24}f''(\theta).$$

Тоді формула середніх прямокутників матиме вигляд

$$\int\limits_a^b f(x)\,dx = (b-a)f\left(\frac{a+b}{2}\right) + \frac{(b-a)^3}{24}f''(\theta). \tag{5.2.7}$$

Зауваження. Якщо за x_0 взяти точку a або b, то одержимо відповідно такі квадратурні формули

$$\int\limits_a^b f(x)dx = (b-a)f(a) + \bar{R}_0(f),$$

$$\int\limits_a^b f(x)dx = (b-a)f(b) + \widetilde{R}_0(f).$$

Першу з них називають ***квадратурною формулою лівих прямокутників***, а другу — ***квадратурною формулою правих прямокутників***. Ці формули теж є формулами Ньютона-Котеса, але мають гірший порядок точності.

Для підвищення точності обчислення визначеного інтеграла відрізок $[a,b]$, як правило, розбивають на певну кількість рівних частин та застосовують формулу прямокутників для кожної частини.

Нехай проміжок $[a,b]$ розбивається на m рівних відрізків точками

$$z_0 = a, \quad z_k = z_0 + kh, \quad k = \overline{1,m}, \quad \text{де} \;\; h := \frac{b-a}{m}.$$

Тоді

$$\int\limits_a^b f(x)dx = \int\limits_{z_0}^{z_1} f(x)\,dx + \int\limits_{z_1}^{z_2} f(x)\,dx + ... + \int\limits_{z_{m-1}}^{z_m} f(x)\,dx =$$

$$= (z_1 - z_0)f\left(\frac{z_0+z_1}{2}\right) + (z_2 - z_1)f\left(\frac{z_1+z_2}{2}\right) + ... + (z_m - z_{m-1})\left(\frac{z_{m-1}+z_m}{2}\right) +$$

$$+\frac{(z_1-z_0)^3}{24}f''(\theta_1) + \frac{(z_2-z_1)^3}{24}f''(\theta_2) + ... + \frac{(z_m-z_{m-1})^3}{24}f''(\theta_m),$$

де $\theta_i \in (z_{i-1}, z_i),\ i = \overline{1,m}$.

Звідси

$$\int_a^b f(x)\,dx = \frac{b-a}{m}\left(f\left(a+\frac{h}{2}\right) + f\left(a+\frac{3h}{2}\right) + \ldots + f\left(a+\frac{2m-1}{2}h\right)\right) +$$

$$+\frac{(b-a)^3}{24m^3}\left(f''(\theta_1) + f''(\theta_2) + \ldots + f''(\theta_m)\right).$$

Нехай $\eta \in (a,b)$ таке, що

$$f''(\eta) = \frac{f''(\theta_1) + f''(\theta_2) + \ldots + f''(\theta_m)}{m},$$

то остаточно отримуємо

$$\int_a^b f(x)\,dx = \frac{b-a}{m}\left(f\left(a+\frac{h}{2}\right) + f\left(a+\frac{3h}{2}\right) + \ldots \; + f\left(a+\frac{2m-1}{2}h\right)\right) + \frac{(b-a)^3}{24m^2}f''(\eta). \tag{5.2.8}$$

Формулу (5.2.8) називають ***великою квадратурною формулою прямокутників***.

5.2.2.2 Формула трапецій

При $n = 1$ функція f замінюється інтерполяційним многочленом першого степеня L_1, побудованим за значеннями функцій f у точках $x_0 = a$ і $x_1 = b$, тобто графік функції f на проміжку $[a, b]$ замінюється відрізком, що сполучає точки $(a, f(a))$ і $(b, f(b))$. Очевидно, що коли $f(x) > 0,\ x \in (a, b)$, то отримаємо трапецію, утворену вказаним відрізком та відповідними відрізками прямих $x = a,\ x = b,\ y = 0$.

Позаяк

$$L_1(x) = f(a)\frac{x-b}{a-b} + f(b)\frac{x-a}{b-a},$$

то з формули

$$\int_a^b f(x)\,dx = \int_a^b L_1(x)\,dx + \widehat{R}_1(f)$$

маємо

$$\int_a^b f(x)\,dx = \frac{f(a)}{a-b}\int_a^b (x-b)\,dx + \frac{f(b)}{b-a}\int_a^b (x-a)\,dx + \widehat{R}_1(f).$$

Звідси одержуємо

$$\int_a^b f(x)\,dx = \frac{1}{2}(b-a)(f(a)+f(b)) + \widehat{R}_1(f). \tag{5.2.9}$$

Формулу (5.2.9) називають ***малою квадратурною формулою трапецій***.

Знайдемо залишковий член $\widehat{R}_1(f)$ формули (5.2.9). Нехай $f \in C^2[a, b]$. Оскільки

$$R_1(x) = \frac{f''(\xi(x))}{2!}(x-a)(x-b), \quad \xi \in (a, b),$$

то

$$\widehat{R}_1(f) = \frac{1}{2}\int_a^b f''(\xi(x))(x-a)(x-b)\,dx.$$

Оскільки функція f'' є неперервною на проміжку $[a, b]$, а квадратний тричлен $(x-a)(x-b)$ не змінює знака на $[a, b]$, а точніше, $(x-a)(x-b) < 0$ при $x \in (a, b)$, то на підставі теореми про середнє значення визначеного інтеграла отримуємо

$$\widehat{R}_1(f) = \frac{f''(\eta)}{2}\int_a^b (x-a)(x-b)\,dx = \frac{f''(\eta)}{2}\int_a^b \left(x^2 - (a+b)x + ab\right)\,dx =$$

$$= -\frac{(b-a)^3}{12}f''(\eta).$$

де $\eta \in (a, b)$. Отож, малу формулу трапецій можна записати у вигляді

$$\int_a^b f(x)\,dx = \frac{1}{2}(b-a)(f(a)+f(b)) - \frac{(b-a)^3}{12}f''(\eta), \tag{5.2.10}$$

де η — деяке число з відрізка $[a, b]$.

Тепер аналогічно, як у попередньому випадку, розіб'ємо відрізок $[a, b]$ на m ($m \in \mathbb{N}$) рівних частин точками

$$z_0 = a, \quad z_k = z_0 + kh, \quad k = \overline{1, m}, \quad \text{де} \quad h := \frac{b-a}{m},$$

і застосуємо малу формулу трапеції (5.2.10) до кожного з одержаних відрізків $[z_{k-1}, z_k]$, $k = \overline{1, m}$. У результаті отримаємо

$$\int_a^b f(x)\,dx = \int_{z_0}^{z_1} f(x)\,dx + \int_{z_1}^{z_2} f(x)\,dx + \ldots + \int_{z_{m-1}}^{z_m} f(x)\,dx =$$

$$= \frac{z_1 - z_0}{2}\left(f(z_0) + f(z_1)\right) + \frac{z_2 - z_1}{2}\left(f(z_1) + f(z_2)\right) + \ldots + \frac{z_m - z_{m-1}}{2}\left(f(z_{m-1}) + f(z_m)\right) -$$

$$-\frac{(z_1-z_0)^3}{12}f''(\eta_1)-\frac{(z_2-z_1)^3}{12}f''(\eta_2)-\ldots-\frac{(z_m-z_{m-1})^3}{12}f''(\eta_m)=$$

$$=\frac{b-a}{2m}\Big(f(z_0)+2f(z_1)+2f(z_2)+\ldots+2f(z_{m-1})+f(z_m)\Big)-\frac{(b-a)^3}{12m^3}\Big(f''(\eta_1)+f''(\eta_2)+\ldots+f''(\eta_m)\Big),$$

де $\eta_i\in(z_{i-1},z_i)$. Позаяк існує $\eta\in(a,b)$ таке, що

$$f''(\eta)=\frac{f''(\eta_1)+f''(\eta_2)+\ldots+f''(\eta_m)}{m},$$

то остаточно одержуємо

$$\int_a^b f(x)\,dx=\frac{b-a}{2m}(f(a)+2f(a+h)+2f(a+2h)+\ldots+2f(a+(m-1)h)+f(b))-$$
$$-\frac{(b-a)^3}{12m^2}f''(\eta). \tag{5.2.11}$$

Формула (5.2.11) називається ***великою квадратурною формулою трапецій***.

5.2.2.3 Формула парабол (Сімпсона)

При $n=2$ функція f замінюється інтерполяційним многочленом другого степеня L_2, побудованим за значеннями функції f у точках $x_0=a, x_1=\frac{a+b}{2}, x_2=b$. Графік функції f на проміжку $[a,b]$ замінюється параболою, проведеною через точки $(a,f(a))$, $\left(\frac{a+b}{2},f\left(\frac{a+b}{2}\right)\right)$, $(b,f(b))$.

Оскільки

$$L_2(x)=f(a)\frac{\left(x-\frac{a+b}{2}\right)(x-b)}{\left(a-\frac{a+b}{2}\right)(a-b)}+f\left(\frac{a+b}{2}\right)\frac{(x-a)(x-b)}{\left(\frac{a+b}{2}-a\right)\left(\frac{a+b}{2}-b\right)}+$$
$$+f(b)\frac{(x-a)\left(x-\frac{a+b}{2}\right)}{(b-a)\left(b-\frac{a+b}{2}\right)}, \tag{5.2.12}$$

то з формули

$$\int_a^b f(x)\,dx=\int_a^b L_2(x)\,dx+\widehat{R}_2(f) \tag{5.2.13}$$

маємо

$$\int_a^b f(x)\,dx=\frac{2f(a)}{(b-a)^2}\int_a^b\left(x-\frac{a+b}{2}\right)(x-b)\,dx-\frac{4f\left(\frac{a+b}{2}\right)}{(b-a)^2}\int_a^b(x-a)(x-b)\,dx+$$
$$+\frac{2f(b)}{(b-a)^2}\int_a^b(x-a)\left(x-\frac{a+b}{2}\right)dx+\widehat{R}_2(f). \tag{5.2.14}$$

Обчисливши інтеграли, що містяться в правій частині рівності, одержимо

$$\int_a^b\left(x-\frac{a+b}{2}\right)(x-b)\,dx=\frac{1}{12}(b-a)^3,$$

$$\int_a^b(x-a)(x-b)\,dx=-\frac{1}{6}(b-a)^3,$$

$$\int_a^b(x-a)\left(x-\frac{a+b}{2}\right)dx=\frac{1}{12}(b-a)^3.$$

Тоді

$$\int_a^b f(x)\,dx=\frac{b-a}{6}\left(f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right)+\widehat{R}_2(f). \tag{5.2.15}$$

Формула (5.2.15) називають ***малою квадратурною формулою парабол***.

Відомо [**?**], що у випадку $f \in C^4[a,b]$ залишковий член малої квадратурної формули парабол має вигляд

$$\widehat{R}_2(f) = -\frac{(b-a)^5}{2880} f^{(4)}(\eta),$$

де $\eta \in [a,b]$ — деяке число.

Отож, у випадку, коли $f \in C^4[a,b]$, малу квадратурну формулу парабол можна записати у вигляді

$$\int_a^b f(x)\,dx = \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right) - \frac{(b-a)^5 f^{(4)}(\eta)}{2880}. \tag{5.2.16}$$

Тепер розіб'ємо відрізок $[a,b]$ на $2m$ ($m \in \mathbb{N}$) рівних частин точками

$$z_0 = a, \quad z_k = z_0 + kh, \quad k = \overline{1,2m}, \quad \text{де} \quad h := \frac{b-a}{2m},$$

і застосуємо малу формулу парабол (5.2.16) до кожного з проміжків $[z_{2j-2}, z_{2j}]$, $j = \overline{1,m}$. У результаті отримаємо

$$\int_a^b f(x)\,dx = \int_{z_0}^{z_2} f(x)\,dx + \int_{z_2}^{z_4} f(x)\,dx + \ldots + \int_{z_{2m-2}}^{z_{2m}} f(x)\,dx =$$

$$= \frac{z_2 - z_0}{6}\left(f(z_0) + 4f(z_1) + f(z_2)\right) + \frac{z_4 - z_2}{6}\left(f(z_2) + 4f(z_3) + f(z_4)\right) +$$

$$+ \ldots + \frac{z_{2m} - z_{2m-2}}{6}\left(f(z_{2m-2}) + 4f(z_{2m-1}) + f(z_{2m})\right) -$$

$$-\frac{(z_2 - z_0)^5}{2880} f^{(4)}(\eta_1) - \frac{(z_4 - z_2)^5}{2880} f^{(4)}(\eta_2) - \ldots - \frac{(z_{2m} - z_{2m-2})^5}{2880} f^{(4)}(\eta_m) =$$

$$= \frac{b-a}{6m}\Big(f(z_0) + 4f(z_1) + 2f(z_2) + 4f(z_3) + 2f(z_4) + \ldots +$$

$$+2f(z_{2m-2}) + 4f(z_{2m-1}) + f(z_{2m})\Big) - \frac{(b-a)^5}{2880m^5}\Big(f^{(4)}(\eta_1) + f^{(4)}(\eta_2) + \ldots + f^{(4)}(\eta_m)\Big),$$

де $\eta_j \in (z_{2j-2}, z_{2j})$, $j = \overline{1,m}$.

Позаяк існує число $\eta \in (a,b)$ таке, що

$$f^{(4)}(\eta) = \frac{f^{(4)}(\eta_1) + f^{(4)}(\eta_2) + \ldots + f^{(4)}(\eta_m)}{m},$$

то остаточно одержуємо ***велику квадратурну формулу парабол***

$$\int_a^b f(x)\,dx = \frac{b-a}{6m}\big(f(z_0) + 4f(z_1) + 2f(z_2) + +4f(z_3) + 2f(z_4) + \ldots + 2f(z_{2m-2}) +$$

$$+4f(z_{2m-1}) + f(z_{2m})\big) - \frac{(b-a)^5}{2880m^4} f^{(4)}(\eta). \tag{5.2.17}$$

Вправи для самостійної роботи

1. Використовуючи велику квадратурну формулу середніх прямокутників, знайти наближення значення визначеного інтеграла

$$\int_0^{10} x^4\,dx, \quad m = 5.$$

Визначити значення m, при якому абсолютна похибка буде меншою за $0,001$.

2. Використовуючи велику квадратурну формулу трапецій, знайти наближення значення визначеного інтеграла

$$\int_{2}^{7} \cos 2x \, dx, \quad m = 5.$$

Визначити значення m, при якому абсолютна похибка буде меншою за $0,001$.

3. Використовуючи велику квадратурну формулу парабол, знайти наближення значення визначеного інтеграла

$$\int_{-2}^{6} e^{-2x} \, dx, \quad m = 4.$$

Визначити значення m, при якому абсолютна похибка буде меншою за $0,0001$.

5.3. Лабораторний практикум з чисельного інтегрування

Нехай проміжок інтегрування розбито на m відрізків довжиною $h = \frac{b-a}{m}$ точками $x_k = a + kh, \quad k = \overline{0, m}$. Тоді великі квадратурні формули для обчислення значення інтеграла

$$I := \int_a^b f(x)\,dx$$

мають вигляд:

- формула прямокутників

$$I_{0,m} = h \sum_{i=0}^{m-1} f(x_i), \tag{5.3.1}$$

- формула трапецій

$$I_{1,m} = h \left[\frac{1}{2}(f(x_0) + f(x_m)) + \sum_{i=1}^{m-1} f(x_i) \right] \tag{5.3.2}$$

- формула парабол

$$I_{2,m} = \frac{h}{6} \left[f(x_0) + 4 \sum_{i=0}^{m-1} f(x_{2i+1}) + 2 \sum_{i=1}^{m-1} f(x_{2i}) + f(x_{2m}) \right]. \tag{5.3.3}$$

Для оцінки результатів чисельного інтегрування розглядають абсолютну $e_{n,m} := |I - I_{n,m}|$ і відносну $d_{n,m} := \frac{e_{n,m}}{|I|} \times 100\%$ похибки чисельного інтегрування, а також відношення $r_{n,m} = \frac{e_{n,m}}{e_{n,2m}}$, яке характеризує швидкість збіжності відповідної квадратурної формули.

Пояснення до використання програмного коду

- Підготувати середовище і потрібні функції :
  1. виконати комірку для підготовки середовища
  2. виконати комірки, в яких **визначені** функції `rectangle_formula`, `trapezoidal_formula`, `simpson_formula`
- Обчислити наближені значення конкретної функції :
  1. виконати комірку, де **визначена** функція, яка задає підінтегральну функцію `f`
  2. виконати комірку з викликом потрібної функції `rectangle_formula`, `trapezoidal_formula` чи `simpson_formula`, задаючи перед виконанням відповідні аргументи цієї функції.

Програмна реалізація методів

Підготовка середовища

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib widget
```

`rectangle_formula` – функція для чисельного інтегрування за великою формулою середніх прямокутників

[2]:
```
def rectangle_formula(f,a,b,m):
    """Чисельне інтегрування за великою формулою середніх прямокутників
    """
    h=(b-a)/m
```

```
    x=np.linspace(a+h/2, b-h/2, m) #обчислення усіх вузлів квадратурної␣
↪формули
    y=f(x)                          # обчислення функції в усіх вузлах
    return h*y.sum()                # обчислення наближеного значення␣
↪інтеграла
```

`trapezoidal_formula` – **функція для чисельного інтегрування за великою формулою трапецій**

[3]:
```
def trapezoidal_formula(f,a,b,m):
    """ Чисельне інтегрування за великою формулою трапецій
    """
    h=(b-a)/m
    x=np.linspace(a, b, m+1)
    y=f(x)
    return h*( 0.5*(y[0]+y[m]) + y[1:m].sum() )
```

`simpson_formula` – **функція для чисельного інтегрування за великою формулою парабол**

[4]:
```
def simpson_formula(f,a,b,m):
    """  обчислення наближеного значення інтеграла
        за великою формулою парабол
      """
    h=(b-a)/m
    x=np.linspace(a, b, 2*m+1)
    y=f(x)
    return h/6*(y[0]+4*y[1::2].sum()+2*y[2:2*m-1:2].sum()+y[2*m])
```

`f` – **функція, яка задає обчислення конкретної підінтегральної функції**

Обчислювальні експерименти

Продемонструємо застосування різних квадратурних формул для чисельного інтегрування неперервних функцій.

Приклад 1. Обчислити інтеграл від $f(x) = x$ на відрізку $[a, b] = [0, 10]$ різними квадратурними формулами.

Визначимо функцію `f1`, яка задає обчислення підінтегральної функції f:

[5]:
```
def f1(x):
    return x
```

Тепер можемо викликати функції для чисельного інтегрування:

[6]:
```
rectangle_formula(f1,0,10,1)
```

[6]: 50.0

[7]:
```
trapezoidal_formula(f1,0,10,10)
```

[7]: 50.0

[8]:
```
simpson_formula(f1,0,10,10)
```

[8]: 50.0

Оскільки підінтегральна функція у прикладі 1 є сталою, то усі розглянуті квадратурні формули є точними для цього випадку.

Приклад 2. Обчислити інтеграл від $f(x) = x\,e^{-x}\,cos(2x)$ на відрізку $[a, b] = [0, 2\pi]$ різними квадратурними формулами та дослідити збіжність чисельних результатів.

Значення інтеграла з точністю до 10^{-16} можна обчислити так:

[9]:
```
iex = 0.12*(np.exp(-2*np.pi)-1)-0.4*np.pi*np.exp(-2*np.pi)
print(f"Точне значення інтеграла  {iex}")
```

```
Точне значення інтеграла  -0.12212260461896841
```

Обчислення підінтегральної функції можна задати так:

[10]:
```
def f2(x):
    return x*np.exp(-x)*np.cos(2*x)
```

Тепер виконаємо чисельне інтегрування різними квадратурними формулами при різних значеннях параметра m:

[11]:
```
m=20
print(f"Чисельні значення інтеграла при m={m}")
```

```
Чисельні значення інтеграла при m=20
```

[12]:
```
ri = rectangle_formula(f2,0,2*np.pi,m)
print(f"ri = {ri}, відносна похибка {np.abs(iex-ri)/np.abs(iex)*100}%")
```

```
ri = -0.11786295252589699, відносна похибка 3.4880128100459853%
```

[13]:
```
ti = trapezoidal_formula(f2,0,2*np.pi,m)
print(f"ti = {ti}, відносна похибка {np.abs(iex-ti)/np.abs(iex)*100}%")
```

```
ti = -0.13055053965359473, відносна похибка 6.901208061293891%
```

[14]:
```
si = simpson_formula(f2,0,2*np.pi,m)
print(f"si = {si}, відносна похибка {np.abs(iex-si)/np.abs(iex)*100}%")
```

```
si = -0.12209214823512955, відносна похибка 0.02493918626603797%
```

[15]:
```
m=100
print(f"Чисельні значення інтеграла при m={m}")
```

```
Чисельні значення інтеграла при m=100
```

[16]:
```
ri = rectangle_formula(f2,0,2*np.pi,m)
print(f"ri = {ri}, відносна похибка {np.abs(iex-ri)/np.abs(iex)*100}%")
```

```
ri = -0.12195631578862703, відносна похибка 0.13616547965072726%
```

[17]:
```
ti = trapezoidal_formula(f2,0,2*np.pi,m)
print(f"ti = {ti}, відносна похибка {np.abs(iex-ti)/np.abs(iex)*100}%")
```

```
ti = -0.12245503440935065, відносна похибка 0.27220987582065304%
```

[18]:
```
si = simpson_formula(f2,0,2*np.pi,m)
print(f"si = {si}, відносна похибка {np.abs(iex-si)/np.abs(iex)*100}%")
```

```
si = -0.12212255532886822, відносна похибка 4.036116028610757e-05%
```

Отримані результати демонструють зменшення відносної похибки квадратурних формул при збільшенні значення параметра m. Щоб отримати повнішу картину цього процесу виконаємо серію обчислень абсолютної похибки, подвоюючи на кожному кроці параметр m.

Побудуємо таку множину значень $m = 2^k,\ k = \overline{2,14}$:

[19]:
```
mk=[2**k for k in range(2,15)]
mk
```

[19]:
```
[4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384]
```

Обчислимо абсолютні похибки $e_{n,m}$ та відношення $r_{n,m}$ для розглянутих квадратурних формул та збережемо їх у таблиці:

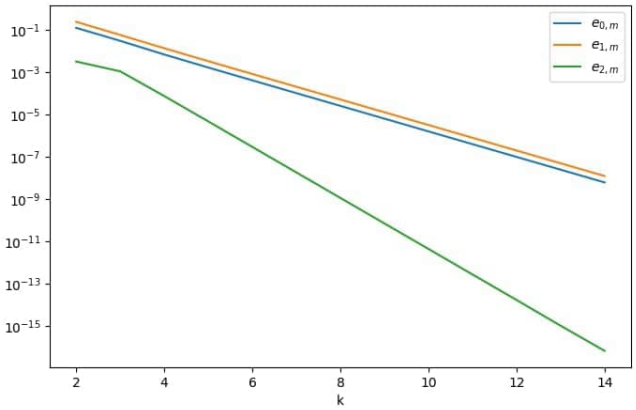
[20]:
```
e_0=[ np.abs( iex - rectangle_formula(f2,0,2*np.pi,m)) for m in mk]
e_1=[ np.abs( iex - trapezoidal_formula(f2,0,2*np.pi,m)) for m in mk]
e_2=[ np.abs( iex - simpson_formula(f2,0,2*np.pi,m)) for m in mk]

r_0=[0,]
r_1=[0,]
r_2=[0,]
for i in range(1,len(e_0)):
    r_0.append((e_0[i-1]/e_0[i]))
    r_1.append((e_1[i-1]/e_1[i]))
    r_2.append((e_2[i-1]/e_2[i]))

df=pd.DataFrame({'k':range(2,15),'m':mk,'e_0_m':e_0,'r_0_m':r_0,'e_1_m':
 ↪e_1,'r_1_m':r_1,'e_2_m':e_2,'r_2_m':r_2}).set_index('k')
df
```

[20]:
```
         m         e_0_m     r_0_m         e_1_m     r_1_m         e_2_m  \
k
2        4  1.221226e-01  0.000000  2.348282e-01  0.000000  3.138990e-03
3        8  2.980433e-02  4.097479  5.635282e-02  4.167107  1.085280e-03
4       16  6.747837e-03  4.416871  1.327424e-02  4.245275  7.381014e-05
5       32  1.638624e-03  4.117989  3.263203e-03  4.067857  4.681869e-06
6       64  4.065851e-04  4.030213  8.122894e-04  4.017291  2.936021e-07
7      128  1.014536e-04  4.007596  2.028522e-04  4.004342  1.836519e-08
8      256  2.535135e-05  4.001902  5.069927e-05  4.001087  1.148058e-09
9      512  6.337085e-06  4.000476  1.267396e-05  4.000272  7.175727e-11
10    1024  1.584224e-06  4.000119  3.168435e-06  4.000068  4.484899e-12
11    2048  3.960531e-07  4.000030  7.921054e-07  4.000017  2.802758e-13
12    4096  9.901310e-08  4.000007  1.980261e-07  4.000004  1.749989e-14
13    8192  2.475326e-08  4.000002  4.950652e-08  4.000001  1.068590e-15
14   16384  6.188315e-09  4.000000  1.237663e-08  4.000000  6.938894e-17

         r_2_m
k
2     0.000000
3     2.892332
4    14.703676
5    15.765100
6    15.946308
7    15.986883
8    15.996740
9    15.999188
10   15.999752
11   16.001733
12   16.015860
13   16.376623
14   15.400000
```

Побудуємо графіки отриманих похибок. Оскільки їх значення швидко заникають, то

будуємо їх у логарифмішній шкалі, беручи на осі абсцис степінь двійки при обчисленні значення параметра m:

[46]:
```
fig = plt.figure(figsize=(8, 5))
df.e_0_m.plot(logy=True, label = '$e_{0,m}$')
df.e_1_m.plot(logy=True, label = '$e_{1,m}$')
df.e_2_m.plot(logy=True, label = '$e_{2,m}$')
ax = fig.gca()
ax.legend();
```



Як бачимо, отримані за допомогою квадратурних формул чисельні результати швидко збігаються до точного значення інтеграла від заданої функції, що відповідає теоретичним оцінкам (5.2.8), (5.2.11) і (5.2.17). Зокрема для великих квадратурних формул прямокутників і трапецій маємо швидкості збіжності, які дорівнюють 4, що означає зменшення у 4 рази абсолютних похибок при подвоєнні розбиття проміжку інтегрування. Для формули парабол швидкість збіжності близька до 16.

[ ]:
```
plt.close('all')
```

Розділ 6

Чисельне розв'язування задач для диференціальних рівнянь та їх систем

6.1. Чисельне розв'язування задачі Коші для звичайних диференціальних рівнянь першого порядку.

6.1.1. Формулювання задачі Коші для звичайного диференціального рівняння, розв'язаного стосовно похідної, та коректність цієї задачі

Розглянемо розв'язане стосовно похідної звичайне диференціальне рівняння першого порядку:

$$u' = f(x, u), \tag{6.1.1}$$

де x – незалежна змінна, $u = u(x)$ – шукана функція, u' – похідна u, $f : D \to \mathbb{R}$ – задана функція, D – область в $\mathbb{R}^2$.

Нагадаємо, що *розв'язком* рівняння (6.1.1) називають функцію

$$u = u(x), \quad x \in \langle a, b\rangle,$$

яка задовольняє умови
1) $u \in C^1\big(\langle a, b\rangle\big)$, тобто вона є неперервно-диференційовною;
2) $(x, u(x)) \in D \quad \forall x \in \langle a, b\rangle$, тобто її графік лежить в області визначення

функції f;
3) $u'(x) = f(x, u(x)) \quad \forall x \in \langle a, b\rangle$, тобто вона задовольняє рівняння (6.1.1).

Задача Коші для рівняння (6.1.1) полягає у знаходженні розв'язку рівняння (6.1.1), який задовольняє умову

$$u(x_0) = u_0, \tag{6.1.2}$$

де (x_0, u_0) — задана точка області D.

Умова (6.1.2) називається *початковою умовою*, а пара чисел (x_0, u_0) — *початковими даними*. Під *вхідними даними* задачі Коші для рівняння (6.1.1) розуміємо функцію f та початкові дані (x_0, u_0).

З геометричної точки зору задача Коші для рівняння (6.1.1) полягає у знаходженні інтегральної лінії цього рівняння, яка проходить через задану точку (x_0, u_0).

Далі сформульовану задачу будемо коротко називати задачею (6.1.1), (6.1.2).

При формулюванні будь-якої задачі виникає питання про її *коректність*. Стосовно задачі (6.1.1), (6.1.2) це означає виконання таких трьох умов: 1) *існування її розв'язку,* 2) *його єдиність* та 3) *неперервну залежність розв'язку від вхідних даних.*

Принагідно зауважимо таке. Нехай $u = \varphi(x)$, $x \in \langle a, b\rangle$, – розв'язок задачі (6.1.1), (6.1.2), а $\langle c, d\rangle \subset \langle a, b\rangle$ — який-небудь проміжок такий, що $x_0 \in \langle c, d\rangle$ і $\langle c, d\rangle \neq \langle a, b\rangle$. Тоді функція $u = \widehat{\varphi}(x)$, $x \in \langle c, d\rangle$, коли $\widehat{\varphi}(x) = \varphi(x)$, $x \in \langle c, d\rangle$, теж є розв'язком задачі (6.1.1), (6.1.2) і він відрізняється від попереднього областю визначення. Відмітимо, що функцію $\widehat{\varphi}$ називають *звуженням* функції φ на проміжок $\langle c, d\rangle$, а функцію φ *продовженням* функції $\widehat{\varphi}$ на проміжок $\langle a, b\rangle$. Отож, якщо задача Коші для ЗДР має хоча б один розв'язок, то вона має безліч розв'язків (в цьому випадку будемо казати, що задача має розв'язки). Але при наявності у цієї задачі безлічі розв'язків можлива одна із двох ситуацій: 1) будь-які два розв'язки даної задачі збігаються на спільній частині їх областей визначення; 2) існує хоча б два розв'язки даної задачі, які в деяких точках спільної частини їх областей визначення мають різні значення.

Приклад 6.1.1. Розглянемо задачу Коші

$$u' = u, \qquad u(0) = u_0 \in \mathbb{R}.$$

Припустимо, що $u = u(x)$, $x \in \langle a, b\rangle$, — який-небудь розв'язок цієї задачі. Тоді для кожного $x \in \langle a, b\rangle$ маємо рівності

$$u'(x) - u(x) = 0 \Big| \times e^{-x} \quad \Leftrightarrow \quad u'(x)e^{-x} - u(x)e^{-x} = 0 \quad \Leftrightarrow$$

$$\big(u(x)e^{-x}\big)' = 0 \quad \Leftrightarrow \quad u(x)e^{-x} = C \quad \Leftrightarrow \quad u(x) = Ce^{x},$$

де C — стала. Знайдемо її значення, використавши початкову умову. Оскільки $u(0) = u_0$, то $u_0 = C$, звідки $u(x) = u_0e^x$, $x \in \langle a, b\rangle$. Легко бачити, що даний розв'язок є звуженням непродовжуваного розв'язку $u = u_0e^x$, $x \in \mathbb{R}$, якщо $\langle a, b\rangle \neq \mathbb{R}$, і збігається з цим непродовжуваним розв'язком в протилежному випадку. Отож, маємо першу ситуацію.

Приклад 6.1.2. Розглянемо задачу Коші

$$u' = 3\sqrt[3]{u^2}, \qquad u(0) = 0$$

Легко переконатися, що ця задача має розв'язки

$$u = 0,\ x \in \mathbb{R}, \quad \text{і} \quad u = x^3,\ x \in \mathbb{R},$$

які не збігатися ніде, за винятком точки 0, хоча умова теореми 6.2.2 виконана. Отож, маємо другу ситуацію.

В першій ситуації кажуть, що розв'язок задачі *єдиний*, а в другій – нема єдиності розв'язку задачі. Далі ми покажемо, що в першій ситуації (а для її наявності потрібні певні умови на f) існує і тільки одна функція (до речі, вона визначна на інтервалі числової осі), яка є розв'язком задачі (6.1.1), (6.1.2) та є продовженням будь-якого іншого розв'язку цієї задачі. Такий розв'язок називають *непродовжуваним.*

При дослідженні питання про коректність задачі (6.1.1), (6.1.2) дуже важливу роль відіграє інтегральне рівняння

$$u = u_0 + \int\limits_{x_0}^{x} f(s, u)\, ds. \tag{6.1.3}$$

Розв'язком рівняння (6.1.3) називають функцію $u = u(x)$, $x \in \langle c, d\rangle$ ($x_0 \in \langle c, d\rangle$), яка задовольняє умови:
1) $u \in C\big(\langle c, d\rangle\big)$;
2) $(x, u(x)) \in D \quad \forall x \in \langle c, d\rangle$;
3) $u(x) = u_0 + \int\limits_{x_0}^{x} f(s,\, u(s))\, ds \quad \forall x \in \langle c, d\rangle$.

Теорема 6.1.1. *Нехай функція f є неперервною. Тоді будь-який розв'язок задачі (6.1.1), (6.1.2) є розв'язком рівняння (6.1.3) і, навпаки, довільний розв'язок рівняння (6.1.3) є розв'язком задачі (6.1.1), (6.1.2).*

Зауважимо, що умова неперервності функції f є природною при дослідженні рівняння (6.1.1), оскільки його розв'язками ми називаємо неперервно диференційовні функції, що задовольняють рівняння. Далі всюди в цьому параграфі будемо вважати, що f є неперервною. Тому, як випливає з теореми 6.2.1, для встановлення умов коректності задачі (6.1.1), (6.1.2) нам досить знайти умови на f, при яких рівняння (6.1.3) має розв'язок, він єдиний і неперервно залежить від f та (x_0, u_0).

Дослідимо питання існування розв'язку задачі Коші для рівняння (6.1.1) з початковою умовою (6.1.2).

Теорема 6.1.2 (Пеано)**.** *Якщо функція f є неперервною, то задача (6.1.1), (6.1.2) має розв'язки.*

Умова теореми Пеано не гарантує однозначності задачі Коші. Це підтверджує приклад 6.1.2, оскільки рівняння з цього прикладу задовольняє умови цієї теореми.

Означення 6.1.1. *Кажуть, що функція $f(x, u)$, $(x, u) \in D$, задовольняє умову Ліпшіца за другим аргументом (змінною u) на D (глобально), якщо існує стала $L \geqslant 0$ така, що для будь-яких $(x, u_1), (x, u_2) \in D$ виконується нерівність*

$$|f(x, u_1) - f(x, u_2)| \leqslant L|u_1 - u_2|.$$

Найменша зі сталих типу L називається сталою Ліпшіца.

Позначимо через $\partial_x := \frac{\partial}{\partial x}$, $\partial_u := \frac{\partial}{\partial u}$ — диференціювання за змінними, відповідно, x та u.

Нехай функція $f(x, u)$, $(x, u) \in D$, є неперервною і має обмежену частинну похідну $\partial_u f(x, u)$, $(x, u) \in D$. Тоді функція f задовольняє умову Ліпшіца за другим аргументом глобально.

Означення 6.1.2. *Кажуть, що функція $f(x, u)$, $(x, u) \in D$, задовольняє умову Ліпшіца за другим аргументом (тобто за змінною u) на D локально, якщо для будь-якого компакту $F \subset D$ звуження цієї функції на F задовольняє умову Ліпшіца (на F глобально).*

Нехай D — область в $\mathbb{R}^2$ і функція $f(x, u)$, $(x, u) \in D$, разом з частинною похідною $\partial_u f(x, u)$, $(x, u) \in D$, є неперервними на D. Тоді функція f задовольняє умову Ліпшіца за другим аргументом локально.

Відмітимо, що слова *«будь-які два розв'язки $u = h_1(x)$, $x \in \langle a_1, b_1\rangle$, і $x = h_2(x)$, $x \in \langle a_2, b_2\rangle$, задачі (6.1.1),(6.1.2) збігаються на спільній частині їх областей визначення»* означають, що $h_1(x) = h_2(x)$ $\forall x \in \rangle a_1, b_1\rangle \cap \rangle a_2, b_2\rangle$.

Теорема 6.1.3 (Пікар)**.** *Нехай функція f є неперервною і задовольняє умову Ліпшіца за другим аргументом локально. Тоді задача (6.1.1), (6.1.2) має розв'язки й будь-які два з них збігаються на спільній частині їх областей визначення.*

Теорема 6.1.4 (про існування непродовжуваного розв'язку)**.** *Нехай функція f є неперервною і задовольняє умову Ліпшіца за другим аргументом локально. Тоді задача (6.1.1), (6.1.2) має розв'язок, який визначений на інтервалі числової осі та є продовженням будь-якого іншого розв'язку цієї задачі.*

Очевидно, що розв'язок, про який йде мова в теоремі 6.1.4, тільки один. Цей розв'язок називають *непродовжуваним*. Його область визначення, як показує наведений далі приклад, визначається вхідними даними задачі.

Приклад 6.1.3. Розв'язати задачу Коші

$$u' = u^2, \quad u(0) = u_0 > 0.$$

Доведення. Знайдемо її розв'язок. Маємо

$$u' = u^2 \;\Leftrightarrow\; \frac{du}{dx} = u^2 \Big| \times \frac{dx}{u^2} \;\Leftrightarrow\; \frac{du}{u^2} = dx, \quad u = 0 \;\Leftrightarrow\; -\frac{1}{u} = x - C, \quad u = 0 \;\Leftrightarrow$$

$$u = \frac{1}{C - x}, \quad u = 0 \quad -$$

повна сім'я розв'язків даного рівняння. Знайдемо значення C з даної початкової умови, підставивши вираз $u = 1/(C - x)$ в цю умову:

$$u_0 = 1/C \quad \Rightarrow \quad C = 1/u_0.$$

Отож, розв'язком даної задачі Коші є функція

$$u = 1/\left(1/u_0 - x\right), \quad x \in (-\infty, 1/u_0),$$

і будь-який інший розв'язок цієї задачі є звуженням цього розв'язку. Це, зокрема, означає, що нема розв'язків даної задачі Коші, які були б визначеними при $x > 1/u_0$. □

Звідси, зокрема, випливає, що не можна гарантувати існування розв'язку задачі (6.1.1), (6.1.2), визначеного на наперед заданому проміжку. Тому важливо вміти визначити проміжки, на яких визначені розв'язки задачі Коші для ЗДР, використовуючи тільки вхідні дані, бо знайти аналітично розв'язки рівняння в загальному випадку неможливо.

Наведемо *спосіб знаходження проміжків, на яких визначені розв'язки задачі Коші для ЗДР*, використовуючи тільки вхідні дані.

Нехай числа $p > 0$ і $q > 0$ такі, що прямокутник

$$P := \{(x, u) \,|\, x_0 \leqslant x \leqslant x_0 + p, \;\; u_0 - q \leqslant u \leqslant u_0 + q\}$$

лежить в області D. Покладемо

$$a := x_0, \quad m := \max_{(x,u)\in P} |f(x, u)|, \quad b := a + \min\Big\{p, \frac{q}{m}\Big\}.$$

Відомо, що задача (6.1.1), (6.1.2) має розв'язок, визначений на відрізку $[a, b]$, який називають *відрізком Пеано* для задачі (6.1.1), (6.1.2). Цей розв'язок отримаємо як границю рівномірно збіжної послідовності $\{\overset{k}{u}\}_{k=1}^{\infty}$ функцій з простору $C[a, b]$, графіки яких є так званими *ламаними Ейлера*.

Теорема 6.1.5. *Нехай $f \in C^{r-1}(D)$, де r — яке-небудь натуральне число. Тоді будь-який розв'язок задачі (6.1.1), (6.1.2) є r-раз неперервно диференційовною функцією, причому, якщо $u = u(x)$, $x \in \langle a, b\rangle$, — який-небудь розв'язок цієї задачі, то*

$$\begin{cases} u'(x) = g_1(x, u(x)), & x \in \langle a, b\rangle, \\ u''(x) = g_2(x, u(x)), & x \in \langle a, b\rangle, \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ u^{(r)}(x) = g_r(x, u(x)), & x \in \langle a, b\rangle, \end{cases} \tag{6.1.4}$$

де для кожного $k \in \{1, \dots, r\}$ функція $g_k(x, u)$, $(x, u) \in D$, визначається функціями

$$f(x, u), \;\; \partial_x f(x, u), \;\; \partial_u f(x, u), \;\; \dots, \;\; \partial_x^i \partial_u^j f(x, u), \quad i + j \leq k - 1, \;\; (x, u) \in D,$$

причому $g_1(x, u) := f(x, u)$, $(x, u) \in D$.

Доведення. Нехай $u = u(x)$, $x \in \langle a, b\rangle$, — її розв'язок. Тоді маємо рівність

$$u'(x) = f(x, u(x)) =: g_1(x, u(x)), \quad x \in \langle a, b\rangle. \tag{6.1.5}$$

Припустимо, що $f \in C^1(D)$. Тоді рівність (6.1.5) можна продиференціювати:

$$\begin{aligned} u''(x) = (f(x, u(x)))' = \partial_x f(x, u(x)) + \partial_u f(x, u(x))u'(x) = \\ = \partial_x f(x, u(x)) + \partial_u f(x, u(x))f(x, u(x)) =: g_2(x, u(x)), \quad x \in \langle a, b\rangle, \end{aligned} \tag{6.1.6}$$

Аналогічно, якщо $f \in C^2(D)$, то рівність (6.1.6) можна продиференціювати:

$$u'''(x) = \big(\partial_x f(x,u(x)) + \partial_u f(x,u(x))f(x,u(x))\big)' = \partial_x^2 f(x,u(x)) +$$

$$+\partial_u\partial_x f(x,u(x))u'(x) + \partial_x\partial_u f(x,u(x))f(x,u(x)) + \partial_u^2 f(x,u(x))f(x,h(x))u'(x) +$$

$$+\partial_u f(x,u(x))\partial_x f(x,u(x)) + \partial_u f(x,u(x))\partial_u f(x,u(x))u'(x) =$$

$$= \partial_x^2 f(x,u(x)) + 2\partial_x\partial_u f(x,u(x))f(x,u(x)) + \partial_u^2 f(x,u(x))(f(x,u(x)))^2 +$$

$$+\partial_x f(x,u(x))\partial_u f(x,u(x)) + (\partial_u f(x,u(x)))^2 f(x,u(x))) =: g_3(x,u(x)), \quad x \in \langle a,b\rangle.$$

І т. д. □

6.1.2. Метод Ейлера.

Розглянемо задачу (6.1.1), (6.1.2), припускаючи, що $f \in C^{r-1}(D)$, де $r \in \mathbb{N}$. Нехай $u = u(x)$, $x \in [a,b]$, — розв'язок цієї задачі. Тоді згідно з теоремою 6.1.5 цей розв'язок належить простору $C^r[a,b]$ і за формулою Тейлора для довільної точки $x_* \in [a,b]$ та будь-якого числа $h \in \mathbb{R}$ такого, що $x_* + h \in [a,b]$, маємо рівність

$$u(x_*+h) = u(x_*) + \frac{1}{1!}u'(x_*)h + \ldots + \frac{1}{r!}u^{(r)}(x_*)h^r + o(h^r), \tag{6.1.7}$$

де $o(h^r) = \dfrac{1}{r!}\left[u^{(r)}(x_*+\theta h) - u^{(r)}(x_*)\right]h^r$, $\theta \in (0,1)$ — деяке число.

Врахувавши (6.1.4), можемо записати

$$u(x_*+h) = u(x_*) + \frac{1}{1!}g_1(x_*,u(x_*))h + \ldots + \frac{1}{r!}g_r(x_*,h(x_*))h^r + o(h^r). \tag{6.1.8}$$

На підставі формули (6.1.8) пропонується такий спосіб побудови наближень розв'язку задачі (6.1.1), (6.1.2) на відрізку $[a,b]$, коли $x_0 = a$:

1. Задаємо рівномірне розбиття відрізка $[a,b]$ точками $x_0, x_1, \ldots, x_n$, де $n \in \mathbb{N}$ і

$$x_i := a + ih, \quad i = \overline{0,n}, \quad h := \frac{b-a}{n}.$$

2. Значення u_0 беремо з умови (6.1.2), а наближення u_i, $i = \overline{1,n}$, відповідно, значень $u(x_i)$, $i = \overline{1,n}$, розв'язку даної задачі знаходимо за рекурентною формулою (див. (6.1.8)):

$$u_{i+1} = u_i + \frac{1}{1!}g_1(x_i,u_i)\cdot h + \ldots + \frac{1}{r!}g_r(x_i,u_i)\cdot h^r, \quad i = \overline{0,n-1}. \tag{6.1.9}$$

Коли $r = 1$, то

$$u_{i+1} = u_i + f(x_i,u_i)\cdot h, \quad i = \overline{0,n-1}. \tag{6.1.10}$$

Знаходження наближень розв'язку задачі (6.1.1), (6.1.2) за формулою (6.1.10) називають ***методом Ейлера***.

Завдання для самостійної роботи

Методом Ейлера розв'язати задачу Коші для ЗДР:

$$u' = u^3 + x^2, \qquad x \in [0\,;3],$$

$$u(0) = 2,$$

взявши $n = 3$.

6.1.3. Методи Рунге-Кутта

Нехай задача (6.1.1), (6.1.2) має єдиний розв'язок, визначений на відрізку $[a, b]$, де $a = x_0$, і ми шукаємо $u_1, \ldots, u_n$ — наближення значень цього розв'язку в точках $x_1, \ldots, x_n$, де $n \in \mathbb{N}$ — деяке число, а

$$x_i := a + ih, \quad i = \overline{0, n}, \qquad h := (b - a)/n.$$

Метод Рунге-Кутта полягає у знаходженні наближень u_i, $i = \overline{1, n}$, (u_0 беремо з початкової умови (6.1.2)) за правилом

$$u_{i+1} = u_i + p_{r,1}k_{i,1}(h) + \ldots + p_{r,r}k_{i,r}(h), \tag{6.1.11}$$

де r — деяке натуральне число,

$$k_{i,1}(h) := f(x_i, u_i) \cdot h,$$

$$k_{i,j}(h) := f\big(x_i + \alpha_j h, u_i + \beta_{j,1}k_{i,1}(h) + \ldots + \beta_{j,j-1}k_{i,j-1}(h)\big) \cdot h, \quad j = \overline{2, r},$$

а коефіцієнти $p_{r,k}, \alpha_j, \beta_{j,l}$, $k = \overline{1, r}, j = \overline{2, r}, l = \overline{1, j-1}$, знаходять із відповідної системи рівнянь.

Розглянемо ***часткові випадки методу Рунге-Кутта***.

1. Нехай $r = 1$. Для знаходження наближень u_i, $i = \overline{0, n}$, розв'язку задачі маємо формули

$$u_{i+1} = u_i + f(x_i, u_i) \cdot h, \quad i = 0, 1, \ldots, n-1. \tag{6.1.12}$$

Формула (6.1.12) виражає ***метод Рунге-Кутта 1 порядку***. Цей метод збігається з методом Ейлера.

2. Нехай $r = 2$. Для знаходження наближень розв'язку задачі (6.1.1), (6.1.2) в точках $x_1, x_2, \ldots, x_n$ маємо формули:

$$u_{i+1} = u_i + p_{2,1}k_{i,1}(h) + p_{2,2}k_{i,2}(h), \tag{6.1.13}$$

де

$$\begin{cases} k_{i,1}(h) := f\left(x_i, u_i\right) \cdot h, \\ k_{i,2}(h) := f\left(x_i + \alpha_2 h, u_i + \beta_{2,1}k_{i,1}(h)\right)) \cdot h, \end{cases} \qquad i = \overline{0, n-1}, \tag{6.1.14}$$

а коефіцієнти $p_{2,1}, p_{2,2}, \alpha_2, \beta_{2,1}$ є розв'язками системи нелінійних рівнянь:

$$\begin{cases} 1 - p_{2,1} - p_{2,2} = 0, \\ 1 - p_{2,2}\alpha_2 = 0. \\ 1 - 2p_{2,2}\beta_{2,1} = 0. \end{cases} \tag{6.1.15}$$

Формули (6.1.13), (6.1.14) виражають ***метод Рунге-Кутта другого порядку***.

Система рівнянь (6.1.15) має безліч розв'язків. Тому один із шуканих коефіцієнтів можна вибрати довільно, а решта визначити через його значення, наприклад,

$$p_{22} = t, \quad p_{21} = 1 - t, \quad \alpha_2 = \frac{1}{2t}, \quad \beta_{21} = \frac{1}{2t},$$

де $t \in \mathbb{R}$ — довільне.

А) Якщо $t = 1$, то $p_{2,2} = 1$, $p_{2,1} = 0$, $\alpha_2 = \beta_{2,1} = \frac{1}{2}$ і формули методу Рунге-Кутта другого порядку матимуть вигляд

$$u_{i+1} = u_i + k_{i,2}(h),$$

де

$$k_{i,1}(h) := f(x_i, u_i)) \cdot h, \quad k_{i,2}(h) := f\left(x_i + \frac{1}{2}h, u_i + \frac{1}{2}k_{i,1}(h)\right) \cdot h, \quad i = \overline{0, n-1}.$$

Б) Якщо $t = 1/2$, то $p_{2,2} = p_{2,1} = 1/2$, $\alpha_2 = \beta_{2,1} = 1$ і формули методу Рунге-Кутта другого порядку набудуть вигляду

$$u_{i+1} = u_i + \frac{1}{2}k_{i,1}(h) + \frac{1}{2}k_{i,2}(h),$$

де

$$k_{i,1}(h) := f(x_i, u_i) \cdot h, \quad k_{i,2}(h) := f(x_i + h, u_i + k_{i,1}(h)) \cdot h, \quad i = \overline{0, n-1}.$$

3. Нехай $r = 3$. Тоді можна вказати три варіанти ***формули методу Рунге-Кутта третього порядку***:

А)

$$u_{i+1} = u_i + \frac{1}{6}k_{i,1}(h) + \frac{2}{3}k_{i,2}(h) + \frac{1}{6}k_{i,3}(h),$$

де

$$k_{i,1}(h) := f(x_i, u_i) \cdot h, \quad k_{i,2}(h) := f\Big(x_i + \frac{1}{2}h, u_i + \frac{1}{2}k_{i,1}(h)\Big) \cdot h,$$

$$k_{i,3}(h) := f\big(x_i + h, u_i - k_{i,1}(h) + 2k_{i,2}(h)\big) \cdot h, \quad i = \overline{0, n-1};$$

Б)

$$u_{i+1} = u_i + \frac{1}{4}k_{i,1}(h) + \frac{3}{4}k_{i,3}(h),$$

де

$$k_{i,1}(h) := f(x_i, u_i) \cdot h, \quad k_{i,2}(h) := f\Big(x_i + \frac{1}{3}h, u_i + \frac{1}{3}k_{i,1}(h)\Big) \cdot h,$$

$$k_{i,3}(h) := f\Big(x_i + \frac{2}{3}h, u_i + \frac{2}{3}k_{i,2}(h)\Big) \cdot h, \quad i = \overline{0, n-1};$$

В)

$$u_{i+1} = u_i + \frac{1}{9}(2k_{i,1}(h) + 3k_{i,2}(h) + 4k_{i,3}(h)),$$

де

$$k_{i,1}(h) := f(x_i, u_i) \cdot h, \quad k_{i,2}(h) := f\left(x_i + \frac{1}{2}h, u_i + \frac{1}{2}k_{i,1}(h)\right) \cdot h,$$

$$k_{i,3}(h) := f\left(x_i + \frac{3}{4}h, u_i + \frac{3}{4}k_{i,2}(h)\right) \cdot h, \quad i = \overline{0, n-1}.$$

4. На практиці найчастіше застосовують ***формули методу Рунге-Кутта четвертого порядку***, який одержуть при $r = 4$. У цьому випадку можна використати такі розрахункові формули:

А)

$$u_{i+1} = u_i + \frac{1}{6}k_{i,1}(h) + \frac{1}{3}k_{i,2}(h) + \frac{1}{3}k_{i,3}(h) + \frac{1}{6}k_{i,4}(h),$$

де

$$k_{i,1}(h) := f(x_i, u_i) \cdot h, \quad k_{i,2}(h) := f\left(x_i + \frac{1}{2}h, u_i + \frac{1}{2}k_{i,1}(h)\right) \cdot h,$$

$$k_{i,3}(h) := f(x_i + \frac{1}{2}h, u_i + \frac{1}{2}k_{i,2}(h)) \cdot h, \quad k_{i,4}(h) := f(x_i + h, u_i + k_{i,3}(h)) \cdot h,$$

$i = \overline{0, n-1}$;

Б)

$$u_{i+1} = u_i + \frac{1}{8}(k_{i,1}(h) + 3k_{i,2}(h) + 3k_{i,3}(h) + k_{i,4}(h)),$$

де

$$k_{i,1}(h) := f(x_i, u_i) \cdot h, \quad k_{i,2}(h) := f\left(x_i + \frac{1}{3}h, u_i + \frac{1}{3}k_{i,1}(h)\right) \cdot h,$$

$$k_{i,3}(h) := f(x_i + \frac{2}{3}h, u_i - \frac{1}{3}k_{i,1}(h) + k_{i,2}(h)) \cdot h,$$

$$k_{i,4}(h) := f(x_i + h, y_i + k_{i,1}(h) - k_{i,2}(h) + k_{i,3}(h)) \cdot h,$$

$i = \overline{0, n-1}$;

В)

$$u_{i+1} = u_i + \frac{1}{6}(k_{i,1}(h) + 4k_{i,3}(h) + k_{i,4}(h)),$$

де

$$k_{i,1}(h) = f(x_i, u_i) \cdot h, \quad k_{i,2}(h) := f\left(x_i + \frac{1}{4}h, u_i + \frac{1}{4}k_{i,1}(h)\right) \cdot h,$$

$$k_{i,3}(h) := f(x_i + \frac{1}{2}h, u_i + \frac{1}{2}k_{i,2}(h)) \cdot h,$$

$$k_{i,4}(h) = f(x_i + h, u_i + k_{i,1}(h) - 2k_{i,2}(h) + 2k_{i,3}(h)) \cdot h, \quad i = \overline{0, n-1}.$$

Отже, для $r \leqslant 4$ маємо методи Рунге-Кутта $r-$го порядку. Їх використання пов'язане з обчисленням правої частини f рівняння (6.1.1) у r точках. При $r = 5$ отримуємо метод Рунге-Кутта четвертого порядку. Тому в цьому випадку розрахункові формули не мають практичного застосування. Метод Рунге-Кутта п'ятого порядку одержують при $r = 6$, а шостого — при $r = 8$. Очевидно, зі збільшенням значення r обсяг обчислень у методі Рунге-Кутта зростає. Тому до методу Рунге-Кутта вищих порядків (понад 4) вдаються переважно тоді, коли обчислення проводять з великим кроком.

Завдання для самостійної роботи

Методом Рунге-Кутта другого і третього порядків (по одному з варіантів А), ...) розв'язати задачу Коші для ЗДР:

$$u' = u + x, \qquad x \in [0\,;3],$$

$$u(0) = 1,$$

взявши $n = 3$.

6.2. Чисельне розв'язування задачі Коші для нормальних систем звичайних диференціальних рівнянь

6.2.1. Постановка і коректність задачі Коші для нормальних систем звичайних диференціальних рівнянь

Описані у попередньому параграфі чисельні методи можна легко перенести на задачу Коші для систем звичайних диференціальних рівнянь першого порядку або, точніше, *нормальних систем звичайних диференціальних рівнянь* (НС). Щоб скоротити записи, обмежимося системою двох диференціальних рівнянь першого порядку.

Розглянемо нормальну систему звичайних диференціальних рівнянь:

$$\begin{cases} u' = f(x,u,v) \\ v' = g(x,u,v) \end{cases}, \tag{6.2.1}$$

де $f(x,u,v)$, $g(x,u,v)$, $(x,u,v) \in G$, — задані функції, G — область в $\mathbb{R}^3$.

Під розв'язком НС (6.2.1) розуміємо пару неперервно диференційовних функцій $u = u(x)$, $v = v(x)$, $x \in \langle a,b \rangle$, які при підставлянні їх в рівняння системи перетворюють ці рівняння в тотожності.

Задача Коші для НС (6.2.1) полягає у знаходженні розв'язку цієї системи рівнянь який задовольняє початкові умови

$$\begin{cases} u(x_0) = u_0 \\ v(x_0) = v_0 \end{cases}. \tag{6.2.2}$$

Далі сформульовану задачу будемо коротко називати задачею (6.2.1), (6.2.2).

При постановці будь-якої задачі виникає питання про її *коректність*. Стосовно задачі (6.2.1), (6.2.2) це означає виконання таких трьох умов: 1) *існування її розв'язку,* 2) *його єдиність* та 3) *неперервну залежність розв'язку від вхідних даних.*

Принагідно зауважимо таке. На підставі тих самих аргументів, які були наведені вище стосовно одного рівняння першого порядку, робимо висновок, що коли задача (6.2.1), (6.2.2) має хоча б один розв'язок, то вона має безліч розв'язків (в цьому випадку будемо казати, що задача має розв'язки). Але при наявності у цієї задачі безлічі розв'язків можлива одна із двох ситуацій: 1) будь-які два розв'язки даної задачі збігаються на спільній частині їх областей визначення; 2) існує хоча б два розв'язки даної задачі, які в деяких точках спільної частини їх областей визначення мають різні значення. В першій ситуації кажуть, що розв'язок задачі *єдиний*, а в другій – нема єдиності розв'язку задачі. Далі ми покажемо, що в першій ситуації (а для її наявності потрібні певні умови на f і g) існує і тільки одна пара функцій (до речі, вона визначна на інтервалі числової осі), яка є розв'язком задачі (6.2.1), (6.2.2) та є продовженням будь-якого іншого розв'язку цієї задачі. Такий розв'язок називають *непродовжуваним.*

При дослідженні питання про коректність задачі (6.2.1), (6.2.2) дуже важливу роль відіграє система інтегральних рівнянь

$$\begin{cases} u = u_0 + \int\limits_{x_0}^{x} f(s,u,v)\,ds \\ v = u_0 + \int\limits_{x_0}^{x} g(s,u,v)\,ds \end{cases}. \tag{6.2.3}$$

Розв'язком системи інтегральних рівнянь (6.2.3) називають пару неперервних функцій $u = u(x)$, $v = v(x)$, $x \in \langle c,d \rangle$ ($x_0 \in \langle c,d \rangle$), які при підставлянні їх в рівняння перетворюють рівняння в рівності.

Теорема 6.2.1. *Нехай функції f і g є неперервними. Тоді будь-який розв'язок задачі (6.2.1), (6.2.2) є розв'язком системи інтегральних рівнянь (6.2.3) і, навпаки, довільний розв'язок рівняння (6.2.3) є розв'язком задачі (6.2.1), (6.2.2).*

Зауважимо, що умова неперервності функцій f і g є природною при дослідженні системи рівнянь (6.2.1), оскільки його розв'язками ми називаємо пари неперервно диференційовних функцій, що задовольняють рівняння цієї системи. Далі всюди в цьому параграфі будемо вважати, що f і g є неперервними. Тому, як випливає з теореми 6.2.1, для встановлення умов коректності задачі (6.2.1), (6.2.2) нам досить знайти умови на f і g, при яких система (6.2.5) має розв'язок, він єдиний і неперервно залежить від f, g та (x_0, u_0, v_0).

Дослідимо питання існування розв'язку задачі Коші (6.2.1), (6.2.2).

Теорема 6.2.2 (Пеано)**.** *Якщо функції f і g є неперервними, то задача (6.2.1), (6.2.2) має розв'язки.*

Умова теореми Пеано не гарантує однозначності задачі Коші. Це підтверджують відповідні приклади.

Означення 6.2.1. *Кажуть, що функція $f(x, u, v)$, $(x, u, v) \in G$, задовольняє умову Ліпшиця за всіма аргументами, крім першого (тобто за змінними u, v) на G (глобально), якщо існує стала $L \geqslant 0$ така, що для будь-яких $(x, u_1, v_1), (x, u_2, v_2) \in G$ виконується нерівність*

$$|f(x, u_1, v_1) - f(x, u_2, v_2)| \leqslant L\left(|u_1 - u_2| + |v_1 - v_2|\right).$$

Означення 6.2.2. *Кажуть, що функція $f(x, u, v)$, $(x, u, v) \in G$, задовольняє умову Ліпшиця за всіма аргументами, крім першого (тобто за змінними u, v), на G локально, якщо для будь-якого компакту $K \subset G$ звуження цієї функції на K задовольняє умову Ліпшиця (глобально).*

Позначимо через $\partial_x := \frac{\partial}{\partial x}$, $\partial_u := \frac{\partial}{\partial u}$ та $\partial_v := \frac{\partial}{\partial v}$ — диференціювання за змінними, відповідно, x, u та v.

Твердження 6.2.1. *Нехай функції f, $\partial_u f$, $\partial_v f$ є неперервними на G. Тоді функція f задовольняє умову Ліпшиця за всіма аргументами, крім першого (тобто за змінними u, v), на G локально.*

Теорема 6.2.3 (Пікар)**.** *Нехай функції f і g є неперервними та задовольняють умову Ліпшиця за всіма аргументами, крім першого (тобто за змінними u, v), на G локально. Тоді задача (6.2.1), (6.2.2) має розв'язки й будь-які два з них збігаються на спільній частині їх областей визначення.*

Теорема 6.2.4 (про існування непродовжуваного розв'язку)**.** *Нехай функції f і g є неперервними та задовольняють умову Ліпшиця за всіма аргументами, крім першого (тобто за змінними u, v) на G локально. Тоді задача (6.2.1), (6.2.2) має розв'язок, який визначений на інтервалі числової осі та є продовженням будь-якого іншого розв'язку цієї задачі.*

Очевидно, що розв'язок, про який йде мова в теоремі 6.2.4, тільки один. Цей розв'язок називають непродовжуваним. Його область визначення, як показують приклади, визначається вхідними даними. Тому не можна гарантувати існування розв'язку задачі (6.2.1), (6.2.2), визначеному на наперед заданому проміжку.

Наведемо *спосіб знаходження проміжків, на яких визначені розв'язки задачі Коші для НС*, використовуючи тільки вхідні дані.

Нехай числа $r > 0$, $p > 0$ та $q > 0$ такі, що паралелепіпед

$$\Pi := \{(x, u, v) \,|\, x_0 \leqslant x \leqslant x_0 + r, \;\; u_0 - p \leqslant u \leqslant u_0 + p, \;\; v_0 - q \leqslant v \leqslant v_0 + q\}$$

лежить в області G. Покладемо

$$a := x_0, \quad b := a + \min\Big\{r, \frac{p}{\max\limits_{\Pi} |f|}, \frac{q}{\max\limits_{\Pi} |g|}\Big\}.$$

Відомо, що задача (6.2.1), (6.2.2) має розв'язок, визначений на відрізку $[a, b]$, який називають *відрізком Пеано* для задачі (6.2.1), (6.2.2).

Для знаходження наближення розв'язку задачі (6.2.1), (6.2.2) можна використати такі ж чисельні методи, які використовують у випадку одного рівняння, тобто методи Ейлера, Рунге-Кутта та інші.

6.2.2. Метод Ейлера розв'язування задачі Коші для НС

Нехай відомо, що задача (6.2.1), (6.2.2) має розв'язок, визначений на відрізку $[a, b]$, де $a = x_0$. Опишемо метод Ейлера знаходження наближення цього розв'язку в точках $x_1, \ldots, x_n \in [a, b]$, визначених за правилом

$$x_i = x_0 + ih, \ i = \overline{1, n}, \quad \text{де } n \in \mathbb{N}, \ h := \frac{b-a}{n}. \tag{6.2.4}$$

Якщо $u = u(x)$, $v = v(x)$, $x \in [a, b]$, — шуканий розв'язок задачі (6.2.1), (6.2.2), то, підставляючи його в систему (6.2.1), отримуємо

$$\begin{cases} u'(x) = f\big(x, u(x), v(x)\big), \\ v'(x) = g\big(x, u(x), v(x)\big), \quad x \in [a, b]. \end{cases} \tag{6.2.5}$$

Для кожного $i \in \{0, 1, \ldots, n-1\}$ проінтегруємо тотожності (6.2.5) на проміжку $[x_i, x_{i+1}]$:

$$\begin{cases} u(x_{i+1}) = u(x_i) + \int\limits_{x_i}^{x_{i+1}} f\big(x, u(x), v(x)\big)\, dx, \\ v(x_{i+1}) = v(x_i) + \int\limits_{x_i}^{x_{i+1}} g\big(x, u(x), v(x)\big)\, dx. \end{cases} \tag{6.2.6}$$

Якщо крок інтегрування h досить малий, то внаслідок неперервності функцій f та g можемо вважати, що для кожного $i \in \{0, 1, \ldots, n-1\}$ функції $f(\cdot, u(\cdot), v(\cdot))$ і $g(\cdot, u(\cdot), v(\cdot))$ є сталими на проміжку $[x_i, x_{i+1}]$, тобто

$$f\big(x, y(x), z(x)\big) = f\big(x_i, y(x_i), z(x_i)\big),$$

$$g\big(x, y(x), z(x)\big) = g\big(x_i, y(x_i), z(x_i)\big), \ x \in [x_i, x_{i+1}].$$

Тоді з формул (6.2.5) дістанемо

$$\begin{cases} u_{i+1} = u_i + f\left(x_i, u_i, v_i\right) \cdot h, \\ v_{i+1} = v_i + g\left(x_i, u_i, v_i\right) \cdot h, \quad i = \overline{0, n-1}. \end{cases} \tag{6.2.7}$$

де $u_i \approx u(x_i)$, $\quad v_i \approx v(x_i)$, $i = \overline{1, n}$, — наближення значень розв'язку задачі (6.2.1), (6.2.2) в точках $x_1, \ldots, x_n \in [a, b]$.

Формули (6.2.7) виражають ***метод Ейлера*** чисельного розв'язування задачі Коші (6.2.1), (6.2.2). Збіжність цього методу можна обґрунтувати так само, як це було зроблено у випадку задачі Коші для диференціального рівняння першого порядку.

Завдання для самостійної роботи

Методом Ейлера розв'язати задачу Коші для ЗДР другого порядку:

$$u' = 2u - v + x, \quad v' = 3xu + v, \qquad x \in [0\,; 3],$$

$$u(0) = 1, \quad v(0) = 5,$$

взявши $n = 3$.

6.3. Чисельне розв'язування задачі Коші для звичайних диференціальних рівнянь вищих порядків

6.3.1. Постановка і коректність задачі Коші для звичайних диференціальних рівнянь вищих порядків

Описане вище чисельне розв'язування задачі Коші для системи звичайних диференціальних рівнянь першого порядку можна легко перенести на випадок задачі Коші для звичайних диференціальних рівнянь вищих порядків. Щоб скоротити записи, обмежимося рівняннями другого порядку.

Розглянемо ***задачу Коші*** для звичайного диференціального рівняння другого порядку

$$u'' = f(x, u, u'), \tag{6.3.1}$$

де $f(x, u, v)$, $(x, u, v) \in G$, — задана функція, G — область в $\mathbb{R}^3$.

Під розв'язком рівняння (6.3.1) розуміємо функцію $u = u(x)$, $x \in \langle a, b\rangle$, яка належить простору $C^2(\langle a, b\rangle)$ і при підставлянні її в рівняння перетворює його в тотожність на проміжку $\langle a, b\rangle$.

Задача Коші для рівняння (6.3.1) полягає у знаходженні його розв'язку, який задовольняє початкові умови

$$u(x_0) = u_0, \quad u'(x_0) = v_0, \tag{6.3.2}$$

де $(x_0, u_0, v_0) \in G$ — довільно фіксована точка.

Нехай $u = u(x)$, $x \in \langle a, b\rangle$, — розв'язок задачі (6.3.1), (6.3.2). Позначимо

$$v(x) := u'(x), \quad x \in \langle a, b\rangle. \tag{6.3.3}$$

Тоді пара функцій $u = u(x)$, $v = v(x))$, $x \in \langle a, b\rangle$, є розв'язком задачі Коші для системи рівнянь

$$\begin{cases} u' = v \\ v' = f(x, u, v) \end{cases}, \tag{6.3.4}$$

який задовольняє початкові умови

$$\begin{cases} u(x_0) = u_0 \\ v(x_0) = v_0 \end{cases}. \tag{6.3.5}$$

Легко переконатися, що коли пара функцій $u = u(x)$, $v = v(x)$, $x \in \langle a, b\rangle$, — розв'язок задачі (6.3.4), (6.3.5), то функція $u = u(x)$, $x \in \langle a, b\rangle$, є розв'язком задачі (6.3.1), (6.3.2). Звідси можна зробити висновок, що для чисельного розв'язування задачі (6.3.1), (6.3.2) достатньо звести її до задачі (6.3.4), (6.3.5), а далі чисельно розв'язувати цю задачу.

6.3.2. Метод Ейлера розв'язування задачі Коші для рівнянь вищих порядків

Тепер розглянемо метод Ейлера знаходження чисельних наближень розв'язку задачі Коші для рівнянь вищих порядків. Для спрощення викладення і більшої ясності розглянемо це на прикладі задачі (6.3.1), (6.3.2).

Нехай $u = u(x)$, $x \in \langle a, b\rangle$, — розв'язок задачі (6.3.1), (6.3.2). Введемо позначення (6.3.3). Тоді пара функцій $u = u(x)$, $v = v(x))$, $x \in \langle a, b\rangle$, є розв'язком задачі Коші (6.3.4), (6.3.5), а отже, можна використати метод Ейлера розв'язування цієї задачі (6.2.1), (6.2.2), тобто шукати в точках $x_1, \ldots, x_n \in [a, b]$, визначених в (6.2.4), наближення $u_i \approx u(x_i)$, $v_i \approx v(x_i)$, $i = \overline{1, n}$, значень розв'язку задачі (6.3.4), (6.3.5) за правилом

$$\begin{cases} u_{i+1} = u_i + v_i \cdot h, \\ v_{i+1} = v_i + f(x_i, u_i, v_i) \cdot h, \quad i = \overline{0, n-1}. \end{cases} \tag{6.3.6}$$

Зі сказаного вище випливає, що u_i, $i = \overline{0, n}$, і є наближення розв'язку задачі (6.3.1), (6.3.2).

Завдання для самостійної роботи

Методом Ейлера розв'язати задачу Коші для ЗДР другого порядку:

$$u'' = 2xu - u', \quad x \in [0\,;4],$$

$$u(0) = 1, \quad u'(0) = 2,$$

взявши $n = 4$.

6.4. Чисельне розв'язування крайових задач для звичайних диференціальних рівнянь

6.4.1. Постановка і коректність крайових задач для звичайних диференціальних рівнянь

Розглянемо рівняння

$$a_0(x)u'' + a_1(x)u' + a_2(x)u = g(x), \quad x \in (a,b), \tag{6.4.1}$$

де $a_0, a_1, a_2, g \in C[a,b],\ a_0(x) \neq 0\ \forall x \in [a,b],\ -\infty < a < b < +\infty$ — задані, а $u : [a,b] \to \mathbb{R}$ — невідома функція.

Розв'язком рівняння (6.4.1) називають функцію $u = h(x),\ x \in [a,b]$, яка належить простору $C^2[a,b]$ та задовольняє це рівняння поточково.

Якщо зробити в рівнянні (6.4.1) заміну змінних

$$x = z + a, \quad z \in [0,l],\ x \in [a,b], \quad \text{де} \ \ l := b - a,$$

то отримуємо рівняння

$$\widetilde{a}_0(z)\widetilde{u}'' + \widetilde{a}_1(z)\widetilde{u}' + \widetilde{a}_2(z)\widetilde{u} = \widetilde{g}(z), \quad z \in [0,l],$$

де

$$\widetilde{a}_0(z) := a_0(z+a), \quad \widetilde{a}_1(z) := a_1(z+a), \quad \widetilde{a}_2(z) := a_2(z+a), \quad \widetilde{g}(z) := g(z+a),$$
$$\widetilde{u}(z) := u(z+a), \quad z \in [0,l],$$

тобто, по суті, нічого не міняється, але область визначення рівняння є інтервал $(0,l)$, що з точки зору зручності викладання матеріалу і огляду алгоритмів розв'язування задач для рівняння (1) є зручнішим.Тому далі вважаємо, що $a = 0, b = l > 0$, тобто рівняння (6.4.1) задано на інтервалі $(0,l)$.

Тепер зауважимо, що коли в рівнянні (6.4.1) зробимо заміну функції u на нову функцію $\widehat{u}$ за правилом:

$$u(x) = \widehat{u}(x) \exp\left\{-\frac{1}{2}\int_0^x \frac{a_1(s)}{a_0(s)}\,ds\right\}, \quad x \in [0,l], \tag{6.4.2}$$

то отримаємо рівняння, яке після заміни позначення $\widehat{u}$ на u має вигляд

$$u'' + q(x)u = f(x), \quad x \in (0,l), \tag{6.4.3}$$

де $q, f \in C[0,l]$ — деякі відомі функції.

Оскільки між розв'язками рівнянь (6.4.1) та (6.4.3) існує, як було показано, взаємно однозначне відображення, то далі будемо розглядати рівняння (6.4.3) замість (6.4.1).

Крайова задача для звичайного диференціального рівняння (6.4.3) полягає у знаходженні розв'язку цього рівняння, який задовольняє такі крайові умови:

$$\begin{cases} \alpha_0 u'(0) + \beta_0 u(0) = \gamma_0, \\ \alpha_1 u'(l) + \beta_1 u(l) = \gamma_1, \end{cases} \tag{6.4.4}$$

де $\alpha_0, \beta_0, \gamma_0, \alpha_1, \beta_1, \gamma_1$ — сталі, причому $|\alpha_0| + |\beta_0| > 0$, $|\alpha_1| + |\beta_1| > 0$.

Далі крайову задачу для рівняння (6.4.3) з крайовими умовами (6.4.4) коротко називатимемо задачею (6.4.3), (6.4.4). Якщо в рівнянні (6.4.3) маємо $f = 0$, тобто це рівняння має вигляд

$$u'' + q(x)u = 0, \quad x \in (0,l), \tag{6.4.5}$$

то його називають *однорідним*, а в протилежному випадку — *неоднорідним*. Якщо в крайових умовах (6.4.4) маємо $\gamma_0 = 0$ і $\gamma_1 = 0$, тобто ці умови мають вигляд:

$$\begin{cases} \alpha_0 u'(0) + \beta_0 u(0) = 0, \\ \alpha_1 u'(l) + \beta_1 u(l) = 0, \end{cases} \tag{6.4.6}$$

то їх називаємо *однорідними*, а в іншому випадку — *неоднорідними*.

З теорії диференціальних рівнянь відоме таке твердження.

Теорема 6.4.1. *Задача (6.4.3), (6.4.4) має і тільки один розв'язок тоді й лише тоді коли відповідна цій задачі однорідна задача (6.4.5),(6.4.6) має тільки нульовий розв'язок.*

Наслідок 6.4.1. *Якщо $q(x) \leqslant 0$ для всіх $x \in [0, l]$, $\alpha_0\beta_0 \leqslant 0$, $\alpha_1\beta_1 \geqslant 0$ та у випадку $q(x) = 0$ для всіх $x \in [0, l]$ маємо або $\beta_0 \neq 0$, або $\beta_1 \neq 0$, то задача (6.4.3), (6.4.4) має і тільки один розв'язок.*

Доведення. Покажемо, що задача (6.4.5), (6.4.6) має тільки нульовий розв'язок. Для цього використаємо метод доведення від супротивного. Припустимо, що u — ненульовий розв'язок задачі (6.4.5), (6.4.6). Підставимо його в рівняння і помножимо отриману рівність для кожного $x \in [0, l]$ на $u(x)$ та проінтегруємо по проміжку $[0, l]$:

$$\int_0^l [u''u + qu^2]dx = 0. \tag{6.4.7}$$

Використавши формулу інтегрування частинами, отримуємо

$$\int_0^l u''udx = u'u\big|_0^l - \int_0^l u'^2\,dx = u'(l)u(l) - u'(0)u(0) - \int_0^l u'^2dx. \tag{6.4.8}$$

Отож, з (6.4.7) і (6.4.8) маємо

$$u'(l)u(l) - u'(0)u(0) - \int_0^l [u'^2 - qu^2]\,dx = 0. \tag{6.4.9}$$

Якщо $\alpha_0\beta_0 = 0$, тобто $u'(0) = 0$ або $u(0) = 0$, то $u'(0)u(0) = 0$. Аналогічно, якщо $\alpha_1\beta_1 = 0$, то $u'(l)u(l) = 0$. Тепер нехай $\alpha_0\beta_0 < 0$, тоді з першої з умов (6.4.6) маємо $u'(0) = -\frac{\beta_0}{\alpha_0}u(0)$, а отже, $u'(0)u(0) = -\dfrac{\beta_0}{\alpha_0}u^2(0) \geqslant 0$. Аналогічно показуємо, що $u'(l)u(l) = -\frac{\beta_1}{\alpha_1}u^2(l) \leqslant 0$ у випадку $\alpha_1\beta_1 > 0$. Отже, маємо

$$u'(l)u(l) - u'(0)u(0) \leqslant 0. \tag{6.4.10}$$

Тоді з (6.4.9) на підставі (6.4.10) одержимо

$$\int_0^l [u'^2 - qu^2]\,dx \leqslant 0. \tag{6.4.11}$$

Ясно, що $u'^2 - qu^2 \geqslant 0$ (нагадаємо, що $q \leqslant 0$), а тому з (6.4.11) отримуємо, що $u'^2 - qu^2 = 0$ на $[a, b]$, звідки випливає, що $u'^2 = 0$ і $qu^2 = 0$ на $[a, b]$, тобто $u(x) = C$, $x \in [a, b]$, і $Cq(x) = 0$, $x \in [a, b]$, де C — деяка стала. Звідси безпосередньо отримуємо, що у випадку $q \neq 0$ будемо мати рівність $C = 0$, тобто $u = 0$. Якщо ж $q = 0$, то з крайових умов (6.4.6) при умові, що або $\beta_0 \neq 0$, або $\beta_1 \neq 0$, також випливає, що $C = 0$, тобто $u = 0$. Отримана суперечність доводить, що задача (6.4.5), (6.4.6) має тільки нульовий розв'язок, а це на підставі теореми 6.4.1 означає, що наше твердження правильне. □

Далі всюди припускаємо, що виконуються умови теореми 6.4.1, тобто задача (6.4.3), (6.4.4) має і тільки один розв'язок та нас цікавить його знаходження в числовій формі.

6.4.2. Класичний метод (метод варіації сталих) розв'язування крайових задач для ЗДР

Як відомо з теорії диференціальних рівнянь, будь-який розв'язок рівняння (6.4.3) може бути поданий у вигляді:

$$u(x) = C_1v_1(x) + C_2v_2(x) + w(x), \quad x \in [0, l], \tag{6.4.12}$$

де v_1, v_2 – лінійно незалежні розв'язки рівняння (6.4.5), а w – розв'язок (частковий) рівняння (6.4.3).

Функції v_1 та v_2 знаходимо як розв'язки задачі Коші для рівняння (6.4.5), відповідно, з початковими умовами

$$u(0) = h_{11}, \quad u'(0) = h_{21}, \tag{6.4.13}$$

та

$$u(0) = h_{12}, \quad u'(0) = h_{22}, \tag{6.4.14}$$

де $h_{kl},\ k,l=1,2,$ — дійсні числа такі, що

$$\begin{vmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{vmatrix} \neq 0.$$

Функцію w знаходимо, як розв'язок задачі Коші для рівняння (6.4.3) з початковими умовами:

$$u(0)=h_{13}, \quad u'(0)=h_{23}, \tag{6.4.15}$$

де h_{13}, h_{23} — які-небудь числа.

На практиці, як правило, вибирають

$$h_{11}=1, \quad h_{12}=0, \quad h_{21}=0, \quad h_{22}=1, \quad h_{13}=0, \quad h_{23}=0.$$

Якщо знайшли функції v_1, v_2 і w, то підставляємо вираз (6.4.12) повного загального розв'язку рівняння (6.4.3) у крайові умови (6.4.4):

$$\alpha_0\big(C_1v_1'(0)+C_2v_2'(0)+w'(0)\big)+\beta_0\big(C_1v_1(0)+C_2v(0)+w(0)\big)=\gamma_0,$$

$$\alpha_1\big(C_1v_1'(l)+C_2v_2'(l)+w'(l)\big)+\beta_1\big(C_1v(l)+C_2v(l)+w(l)\big)=\gamma_1,$$

Звідси отримаємо систему лінійних алгебраїчних рівнянь для знаходження значень C_1, C_2:

$$\begin{cases} \big(\alpha_0v_1'(0)+\beta_0v_1(0)\big)C_1+\big(\alpha_0v_2'(0)+\beta_0v_2(0)\big)C_2=\gamma_0-\big(\alpha_0w'(0)+\beta_0w(0)\big), \\ \big(\alpha_1v_1'(l)+\beta_1v_1(l)\big)C_1+\big(\alpha_1v_2'(l)+\beta_1v_2(l)\big)C_2=\gamma_1-\big(\alpha_1w'(l)+\beta_1w(l)\big). \end{cases} \tag{6.4.16}$$

Якщо ввести позначення

$$a_{11}:=\alpha_0v_1'(0)+\beta_0v_1(0), \quad a_{12}:=\alpha_0v_2'(0)+\beta_0v_2(0),$$

$$a_{21}:=\alpha_1v_1'(l)+\beta_1v_1(l), \quad a_{22}:=\alpha_1v_2'(l)+\beta_1v_2(l),$$

$$b_1:=\gamma_0-\big(\alpha_0w'(0)+\beta_0w(0)\big), \quad b_2:=\gamma_1-\big(\alpha_1w'(l)+\beta_1w(l)\big),$$

то система (6.4.16) матиме вигляд

$$\begin{cases} a_{11}C_1+a_{12}C_2 &= b_1, \\ a_{21}C_1+a_{22}C_2 &= b_2, \end{cases} \tag{6.4.17}$$

розв'язки якої використовуючи методом Крамера, записати у вигляді

$$C_1=\frac{\Delta_1}{\Delta}, \quad C_2=\frac{\Delta_2}{\Delta},$$

де

$$\Delta:=\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \quad \Delta_1:=\begin{pmatrix} b_1 & a_{12} \\ b_2 & a_{22}, \end{pmatrix}, \quad \Delta_2:=\begin{pmatrix} a_{11} & b_1 \\ a_{21} & b_2, \end{pmatrix}. \tag{6.4.18}$$

Приклад 6.4.1. Розв'язати крайову задачу

$$u''-4u=8x, \quad x\in(0\,;1), \tag{6.4.19}$$

$$u(0)=0, \quad u'(1)=2. \tag{6.4.20}$$

Розв'язування. Спочатку знайдемо повний загальний розв'язок рівняння (6.4.19). Для цього потрібно знайти два лінійно незалежні розв'язки v_1 і v_2 відповідно рівнянню (6.4.19) лінійного однорідного рівняння:

$$u''-4u=0, \quad x\in(0\,;1). \tag{6.4.21}$$

Це рівняння зі сталими коефіцієнтами, а тому, шукаємо його розв'язки у вигляді функції $u=e^{\lambda x}$, де λ – стала, значення якої шукаємо, за умови, що при підставленні цієї функції в рівняння (6.4.21), отримаємо тотожність:

$$(\lambda^2-4)e^{\lambda x}=0 \quad \Leftrightarrow \quad \lambda^2-4=0 \quad \Leftrightarrow \quad \lambda_{1,2}=\pm 2.$$

Отже маємо

$$v_1(x) = e^{2x}, \quad v_2(x) = e^{-2x}, \quad x \in [0\,;1], \tag{6.4.22}$$

Справді, ці функції, які ми показали, є розв'язками рівняння (6.4.21) і

$$\begin{vmatrix} v_1(0) & v_2(0) \\ v_1'(0) & v_1'(0) \end{vmatrix} = \begin{vmatrix} 1 & 1 \\ 2 & -2 \end{vmatrix} = -4 \neq 0,$$

тобто вони є лінійно незалежними.

Знайдемо функцію w, як частковий розв'язок рівняння (6.4.19). Оскільки вільний член цього рівняння є многочленом першого порядку, то його показник (як квазімногочлена) дорівнює нулю, який не є характеристичним числом, і проект часткового розв'язку рівняння (6.4.19) можна записати у вигляді

$$u = ax + b, \quad x \in [0\,;1].$$

де a, b – неозначені коефіцієнти деякого многочлена першого порядку. Підставивши цей проєкт розв'язку у рівняння (6.4.19), знаходимо $a = -2, b = 0$, тобто

$$w(x) = -2x, \quad x \in [0\,;1]. \tag{6.4.23}$$

Отже, будь-який розв'язок рівняння (6.4.19) можна записати (див. (6.4.12), (6.4.22), (6.4.23)) у вигляді

$$u(x) = C_1 e^{2x} + C_2 e^{-2x} - 2x, \quad x \in [0\,;1], \tag{6.4.24}$$

де C_1, C_2 — довільні сталі.

Підставимо цей вираз у крайові умови (6.4.20), попередньо знайшовши похідну $u'(x)$:

$$u'(x) = 2C_1 e^{2x} - 2C_2 e^{-2x} - 2, \quad x \in [0\,;1],$$

де C_1, C_2 – довільні сталі.

У результаті отримаємо систем рівнянь на знаходження значень C_1 і C_2:

$$\begin{cases} C_1 + C_2 = 0, \\ 2e^2 C_1 - 2e^{-2} C_2 - 2 = 2. \end{cases}$$

Розв'яжемо її. З першого рівняння маємо $C_2 = -C_1$, а тоді з другого рівняння

$$2(e^2 + e^{-2})C_1 = 4 \quad \Rightarrow \quad C_1 = 2/(e^2 + e^{-2}) \quad \text{і} \quad C_2 = -2/(e^2 + e^{-2}).$$

Звідси та з (6.4.24) маємо розв'язок задачі (6.4.19), (6.4.20):

$$u = 2e^{2x}/(e^2 + e^{-2}) - 2e^{-2x}/(e^2 + e^{-2}) - 2x, \quad x \in [0\,;1].$$

□

Задачі для самостійної роботи

Розв'язати крайові задачі:

1. $u'' - 9u = 3e^{2x}, \quad x \in [0\,;2],$

$u'(0) = 1, \quad u'(2) = 0;$

2. $u'' - 16u = 5x + 3, \quad x \in [0\,;3],$

$u'(0) - u(0) = 0, \quad u'(3) = 3.$

6.4.3. Метод сіток (різницевий метод)

Ми вже розглянули метод розв'язування крайових задач, який базується на зведенні крайової задачі до задачі Коші. Але його не тільки не можна застосувати для розв'язування всіх типів крайових задач для лінійних диференціальних рівнянь, а й зовсім – для нелінійних диференціальних рівнянь. Тому розглянемо більш загальний метод — метод сіток. Суть цього методу полягає у зведенні крайової задачі до розв'язування певної системи лінійних алгебраїчних рівнянь. Розглянемо цей метод на прикладі лінійних диференціальних рівнянь другого порядку.

Нехай треба знайти визначений на відрізку $[0, l]$ розв'язок диференціального рівняння

$$u'' + q(x)u = f(x), \tag{6.4.25}$$

який задовольняє крайові умови

$$\begin{cases} \alpha_0 u'(0) + \beta_0 u(0) = \gamma_0, \\ \alpha_1 u'(l) + \beta_1 u(l) = \gamma_1, \end{cases} \tag{6.4.26}$$

де $q, f \in C([0, l])$, $\alpha_0, \beta_0, \gamma_0, \alpha_1, \beta_1, \gamma_1 \in \mathbb{R}$, причому $|\alpha_0| + |\beta_0| > 0, \quad |\alpha_1| + |\beta_1| > 0$. Вважаємо, що ця крайова задача має і тільки один розв'язок.

Суть методу сіток така. На відрізку $[0, l]$ вибирають систему точок:

$$x_i = ih, \quad i = \overline{0, N}, \quad \text{де} \;\; h := \frac{l}{N}$$

(очевидно, що $0 = x_0 < x_1 < \ldots < x_N = l$). Сукупність точок x_i, $i = \overline{0, N}$, називають *сіткою*, будь-яку з точок сітки називають її *вузлом*, а значення $u(x_i)$, $i = \overline{0, N}$, у вузлах сітки називають *сітковими значеннями* функції u. В методі сіток відшукують наближення сіткових значень u_i, $i = \overline{0, N}$, розв'язку u у вузлах x_i, $i = \overline{0, N}$. В основу цього методу покладемо заміну похідних першого і другого порядку від функції u у вузлах сітки різницевими співвідношеннями. У результаті для обчислення значень u_i, $i = \overline{0, N}$, отримують деяку систему рівнянь

$$\sum_{i=0}^{N} c_{ij} u_i = d_j, \quad j = \overline{0, N}, \tag{6.4.27}$$

де c_{ij}, d_j, $i = \overline{0, N}, j = \overline{0, N}$, — деякі числа.

Система (6.4.27) повинна добре апроксимувати диференціальне рівняння (6.4.25) у точках x_i, $i = \overline{1, N-1}$, і крайові умови (6.4.26), відповідно, в точках x_0 і x_N. Якщо система рівнянь має єдиний розв'язок, то його вважають наближенням значень розв'язку крайової задачі у точках $x_0, x_1, \ldots, x_N$, тобто $u_0 \approx u(x_0), u_1 \approx u(x_1), \ldots, u_N \approx u(x_N)$.

Диференціальне рівняння (6.4.25) розглядатимемо лише у внутрішніх точках сітки, тобто при $x = x_i$, $i = \overline{1, N-1}$, а крайові умови (6.4.26) – відповідно при $x_0 = 0$ і $x_N = l$. Підставивши в диференціальне рівняння (6.4.25) $x = x_i$, одержимо

$$u''(x_i) + q(x_i)u(x_i) = f(x_i), \quad i = \overline{1, N-1}. \tag{6.4.28}$$

Тепер для кожного $i \in \{1, \ldots, N-1\}$ виразимо $u'(x_i)$ і $u''(x_i)$ через значення функції u у точках x_{i-1}, x_i, x_{i+1}, тобто через $u(x_{i-1}), u(x_i), u(x_{i+1})$.

Припустимо, що функція u на проміжку $[0, l]$ має неперервні похідні до четвертого порядку включно. Зобразимо функцію u в околі точки x_i за формулою Тейлора

$$u(x) = u(x_i) + \frac{u'(x_i)}{1!}(x - x_i) + \frac{u''(x_i)}{2!}(x - x_i)^2 + \frac{u'''(x_i)}{3!}(x - x_i)^3 + \frac{u^{(4)}(\xi)}{4!}(x - x_i)^4,$$

де ξ – деяка точка між x і x_i.

З цієї рівності при, відповідно, $x = x_{i+1}$ і $x = x_{i-1}$ матимемо

$$u(x_{i+1}) = u(x_i) + \frac{u'(x_i)}{1!}\, h + \frac{u''(x_i)}{2!}\, h^2 + \frac{u'''(x_i)}{3!}\, h^3 + \frac{u^{(4)}(\xi_1)}{4!}\, h^4, \quad \xi_1 \in (x_i, x_{i+1}), \tag{6.4.29}$$

$$u(x_{i-1}) = u(x_i) - \frac{u'(x_i)}{1!}\,h + \frac{u''(x_i)}{2!}\,h^2 - \frac{u'''(x_i)}{3!}\,h^3 + \frac{u^{(4)}(\xi_2)}{4!}\,h^4, \quad \xi_2 \in (x_{i-1}, x_i). \tag{6.4.30}$$

Звідси одержуємо такі вирази для $u'(x_i)$:

$$\frac{u(x_{i+1}) - u(x_i)}{h} = u'(x_i) + O(h), \tag{6.4.31}$$

$$\frac{u(x_i) - u(x_{i-1})}{h} = u'(x_i) + O(h), \tag{6.4.32}$$

$$\frac{u(x_{i+1}) - u(x_{i-1})}{2h} = u'(x_i) + O(h^2). \tag{6.4.33}$$

Додавши формули (6.4.29) і (6.4.30), отримаємо вираз для $u''(x_i)$:

$$\frac{u(x_{i+1}) - 2u(x_i) + u(x_{i-1})}{h^2} = u''(x_i) + O(h^2). \tag{6.4.34}$$

Зазначимо, що це не єдині можливі вирази для $u'(x_i)$ і $u''(x_i)$. Використовуючи значення функції u в інших точках, крім зазначених, можна знайти точніші заміни, проте вони будуть і складнішими.

Підставивши у (6.4.28) замість $u'(x_i)$ і $u''(x_i)$ їхні вирази із формул (6.4.33) і (6.4.34), одержимо

$$\frac{u(x_{i+1}) - 2u(x_i) + u(x_{i-1})}{h^2} + q(x_i)u(x_i) = f(x_i) + O(h^2),$$

або, помноживши обидві частини рівності на h^2, матимемо

$$u(x_{i+1}) + \Big(h^2 q(x_i) - 2\Big)u(x_i) + u(x_{i-1}) = h^2 f(x_i) + O(h^4).$$

Звідси одержуємо таку *систему алгебраїчних рівнянь*, яка апроксимує диференціальне рівняння (6.4.25) у внутрішніх точках відрізку $[0, l]$:

$$u_{i-1} + \Big(h^2 q_i - 2\Big)u_i + u_{i+1} = h^2 f_i, \quad i = \overline{1, N-1}, \tag{6.4.35}$$

де $q_i := q(x_i)$, $f_i := f(x_i)$, $i = \overline{1, N-1}$.

Застосувавши формули (6.4.31), (6.4.32), перепишемо крайові умови (6.4.26) так:

$$\alpha_0 \frac{u(x_1) - u(x_0)}{h} + \beta_0 u(x_0) = \gamma_0 + O(h);$$

$$\alpha_1 \frac{u(x_N) - u(x_{N-1})}{h} + \beta_1 u(x_N) = \gamma_1 + O(h),$$

або

$$\alpha_0 u(x_1) + (\beta_0 h - \alpha_0)u(x_0) = h\gamma_0 + O(h^2);$$
$$(\alpha_1 + h\beta_1)u(x_N) - \alpha_1 u(x_{N-1}) = h\gamma_1 + O(h^2).$$

Звідси отримуємо такі рівняння для апроксимації крайових умов:

$$\begin{cases} (-\alpha_0 + \beta_0 h)u_0 + \alpha_0 u_1 = h\gamma_0, \\ -\alpha_1 u_{N-1} + (\alpha_1 + h\beta_1)u_N = h\gamma_1. \end{cases} \tag{6.4.36}$$

Як бачимо, рівняння (6.4.35) і (6.4.36) разом утворюють систему лінійних алгебраїчних рівнянь для визначення наближень сіткових значень $u_0, u_1, \ldots, u_N$:

$$\begin{cases} (-\alpha_0 + \beta_0 h)u_0 + \alpha_0 u_1 = h\gamma_0, \\ u_{i-1} + \Big(h^2 q_i - 2\Big)u_i + u_{i+1} = h^2 f_i, \quad i = \overline{1, N-1}, \\ -\alpha_1 u_{N-1} + (\alpha_1 + h\beta_1)u_N = h\gamma_1. \end{cases} \tag{6.4.37}$$

де $q_i := q(x_i)$, $f_i := f(x_i)$, $i = \overline{1, N-1}$.

Матриця системи (6.4.37) є тридіагональною і має такий вигляд:

$$D = \begin{pmatrix} -\alpha_0 + \beta_0 h & \alpha_0 & 0 & \dots & 0 & 0 & 0 \\ 1 & h^2 q_1 - 2 & 1 & \dots & 0 & 0 & 0 \\ 0 & 1 & h^2 q_2 - 2 & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 & h^2 q_{N-1} - 2 & 1 \\ 0 & 0 & 0 & \dots & 0 & -\alpha_1 & \alpha_1 + h\beta_1 \end{pmatrix}.$$

Отже, система рівнянь (6.4.35) замінює диференціальне рівняння (6.4.25) у внутрішніх точках проміжку $[a, b]$ з похибкою, яка має четвертий порядок щодо h, а рівняння (6.4.36) замінюють відповідні крайові умови (6.4.26) з похибкою другого порядку щодо h. Це означає, що диференціальне рівняння апроксимується з меншою похибкою, ніж крайові умови. Таке явище буває небажаним під час розв'язування крайової задачі. Порядок щодо h апроксимації крайових умов можна збільшити на одиницю, якщо для заміни похідних $u'(0)$ і $u'(l)$ використати точніші вирази, а саме:

$$\frac{-u(x_2) + 4u(x_1) - 3u(x_0)}{2h} = u'(x_0) + O(h^2).$$

і

$$\frac{3u(x_N) - 4u(x_{N-1}) + u(x_{N-2})}{2h} = u'(x_N) + O(h^2).$$

Ці вирази легко одержати використовуючи формулу Тейлора. Справді, за формулою Тейлора:

$$u(x_0 + h) = u(x_0) + \frac{u'(x_0)}{1!}\, h + \frac{u''(x_0)}{2!}\, h^2 + \frac{u'''(x_0)}{3!}\, h^3 + \frac{u^{(4)}(\eta_1)}{4!}\, h^4, \quad \eta_1 \in (x_0, x_1)$$

$$u(x_0 + 2h) = u(x_0) + \frac{u'(x_0)}{1!}\, 2h + \frac{u''(x_0)}{2!}\, (2h)^2 + \frac{u'''(x_0)}{3!}\, (2h)^3 + \frac{u^{(4)}(\eta_2)}{4!}\, (2h)^4,$$

$\eta_2 \in (x_0, x_2)$.
Звідси

$$4u(x_1) - u(x_2) - 3u(x_0) = 2h\, u'(x_0) + O(h^3),$$

або

$$\frac{-u(x_2) + 4u(x_1) - 3u(x_0)}{2h} = u'(x_0) + O(h^2).$$

Аналогічно можна отримати і друге співвідношення.

Приклад 6.4.2. Розв'язати методом сіток задачу

$$u'' - (x+1)u = x^2, \quad x \in (0, 3), \tag{6.4.38}$$

$$u(0) = 0, \quad u(3) = 0, \tag{6.4.39}$$

взявши $h = 1$.

Розв'язування. Приймемо

$$q(x) = -(x+1), \quad f(x) = x^2, \quad x \in (0, 3);$$

$$x_0 = 0, \quad x_1 = 1, \quad x_2 = 2, \quad x_3 = 3.$$

Тоді

$$u_0 := u(0) = 0, \quad u_1 \approx u(1), \quad u_2 \approx u(2), \quad u_3 := u(3) = 0;$$

$$q_1 := q(1) = -(1+1) = -2, \quad q_2 := q(2) = -(2+1) = -3;$$

$$f_1 := f(1) = 1^2 = 1, \quad f_2 := f(2) = 2^2 = 4.$$

Тоді згідно з формулами (109) і (110) отримаємо систему рівнянь

$$\begin{cases} k=0: & u_0=0, \\ k=1: & u_0+(1^2\cdot(-2)-2)u_1+u_2=1^2\cdot 1, \\ k=2: & u_1+(1^2\cdot(-3)-2)u_2+u_3=1^2\cdot 4, \\ k=3: & u_3=0, \end{cases} \tag{6.4.40}$$

Звідси маємо

$$\begin{cases} u_0=0, \\ -4u_1+u_2=1, \\ u_1-5u_2=4, \\ u_3=0, \end{cases} \tag{6.4.41}$$

Звідси знаходимо наближення розв'язку задачі (6.4.38), (6.4.39):

$$u_0=0, \quad u_1=-\frac{9}{19}, \quad u_2=-\frac{17}{19}, \quad u_3=0.$$

Відповідь:

x_i	0	1	2	3
u_i	0	$-9/19$	$-17/19$	0

□

Вправи для самостійної роботи

Розв'язати крайові задачі методом сіток:

0. $u''+u=1$ на відрізку $[0,\pi/2]$, $u(0)=0$, $u'(\pi/2)=1$.

Відомо, що функція $u(x)=1-sin(x)-cos(x)=1-\sqrt{2}sin(x+\pi/4)$ є розв'язком такої задачі.

1. $u''-(x+2)u=x^3+1, \quad x\in(0\,;4)$,

$u(0)=0$, $u(4)=0$;

2. $u''-x^2u=2x+1, \quad x\in(0\,;3)$,

$u'(0)=0$, $u(3)=0$.

6.5. Чисельне розв'язування крайових задач для еліптичних рівнянь. Метод сіток

6.5.1. Постановка задачі Діріхле для рівняння Пуассона

Нехай $a>0$, $b>0$ – деякі числа. Позначимо

$$D:=(0,a)\times(0,b)\equiv\{(x,y)\mid 0<x<a,\ 0<y<b\},$$
$$\overline{D}:=[0,a]\times[0,b]\equiv\{(x,y)\mid 0\leqslant x\leqslant a,\ 0\leqslant y\leqslant b\}.$$

Задача Діріхле для рівняння Пуассона полягає у знаходженні функції $u\in C^2(D)\cap C(\overline{D})$, яка задовольняє рівняння

$$\Delta u=f(x,y), \quad (x,y)\in D, \tag{6.5.1}$$

і крайові умови

$$u|_{x=0}=\varphi_1(y), \quad u|_{x=a}=\varphi_2(y), \quad y\in[0,b], \tag{6.5.2}$$
$$u|_{y=0}=\psi_1(x), \quad u|_{y=b}=\psi_2(x), \quad x\in(0,a), \tag{6.5.3}$$

де

• $f\in C(\overline{D})$, $\varphi_k\in C([0,b])$, $\psi_k\in C([0,a])$, $k=1,2$, — задані функції, причому виконуються умови узгодження:

$$\varphi_1(0)=\psi_1(0), \quad \varphi_1(b)=\psi_2(0), \quad \varphi_2(0)=\psi_1(a), \quad \varphi_2(b)=\psi_2(a),$$

• $\Delta u(x,y):=u_{xx}(x,y)+u_{yy}(x,y), \quad (x,y)\in D$, — дія оператора Лапласа на функцію u.

6.5.2 Різницева схема

Нехай N, M – які-небудь натуральні числа. Покладемо

$$h := \frac{a}{N}, \quad \tau := \frac{b}{M}$$

і визначимо

$$x_i := ih, \ i = \overline{0, N}, \quad y_i := j\tau, \ j = \overline{0, M}.$$

Нехай

$$u_{i,j} \approx u(x_i, y_j), \ i = \overline{0, N}, \ j = \overline{0, M}, \quad —$$

наближення функції u, а

$$f_{i,j} := f(x_i, y_j), \quad i = \overline{1, N-1}, \ j = \overline{1, M-1},$$

$$\varphi_{k,j} := \varphi_k(y_j), \quad k = \overline{1, 2}, \ j = \overline{0, M},$$

$$\psi_{k,i} := \psi_k(x_i), \quad k = \overline{1, 2}, \ i = \overline{1, N-1} \quad —$$

сіткові значення вхідних даних.

Використаємо такі апроксимації похідних функцій u:

$$u_{xx}(x_i, y_j) \approx \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2}, \quad u_{yy}(x_i, y_j) \approx \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{\tau^2},$$

$$i = \overline{1, N-1}, \ j = \overline{1, M-1}.$$

У результаті отримаємо таку різницеву схему для чисельного розв'язування задачі (6.5.1)-(6.5.3):

$$\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{\tau^2} = f_{i,j} \quad \Leftrightarrow$$

$$\frac{u_{i-1,j} + u_{i+1,j}}{h^2} - \left(\frac{2}{h^2} + \frac{2}{\tau^2}\right) u_{i,j} + \frac{u_{i,j-1} + u_{i,j+1}}{\tau^2} = f_{i,j}, \tag{6.5.4}$$

$i = \overline{1, N-1}, \ j = \overline{1, M-1}$,

$$u_{0,j} = \varphi_{1,j}, \quad u_{N,j} = \varphi_{2,j}, \quad j = \overline{0, M}, \tag{6.5.5}$$

$$u_{i,0} = \psi_{1,i}, \quad u_{i,M} = \psi_{2,i}, \quad i = \overline{1, N-1}. \tag{6.5.6}$$

Співвідношення (6.5.4) — (6.5.6) утворюють систему лінійних алгебраїчних рівнянь (СЛАР) відносно шуканих значень наближень розв'язку даної задачі. Цю систему розв'язують одним із раніше розглянутих методів розв'язування СЛАР.

Приклад 6.5.1. Розв'язати методом сіток крайову задачу для рівняння Пуассона:

$$\Delta u = x + y, \quad (x, y) \in (0, 3) \times (0, 2), \tag{6.5.7}$$

$$u|_{x=0} = y, \quad u|_{x=3} = 2y, \quad y \in [0, 2], \tag{6.5.8}$$

$$u|_{y=0} = 0, \quad u|_{y=2} = \frac{2}{3}x + 2, \quad x \in (0, 3). \tag{6.5.9}$$

Розв'язування. Маємо

$$f(x, y) := x + y, \quad (x, y) \in (0, 3) \times (0, 2), \quad \varphi_1(y) := y, \quad \varphi_2(y) := 2y, \quad y \in [0, 2],$$

$$\psi_1(x) := 0, \quad \psi_2(x) := \frac{2}{3}x + 2, \quad x \in (0, 3).$$

Візьмемо $N = 3$, $M = 2$. Тоді $h = \tau = 1$,

$$x_i = ih, \quad i = \overline{0, 3} \quad \Leftrightarrow \quad x_0 = 0, \ x_1 = 1, \ x_2 = 2, \ x_3 = 3,$$

$$y_j = j\tau, \ j = \overline{0, 2} \quad \Leftrightarrow \quad y_0 = 0, \ y_1 = 1, \ y_2 = 2.$$

Отож, маємо

$$f_{1,1} := f(x_1, y_1) = 1 \cdot 1 + 1 \cdot 1 = 2; \quad f_{2,1} := f(x_2, y_1) = 2 \cdot 1 + 1 \cdot 1 = 3;$$

$$\varphi_{1,0} = 0; \quad \varphi_{1,1} = 1; \quad \varphi_{1,2} = 2; \quad \varphi_{2,0} = 0; \quad \varphi_{2,1} = 2; \quad \varphi_{2,2} = 4; \tag{6.5.10}$$

$$\psi_{1,1} = 0; \quad \psi_{1,2} = 0; \quad \psi_{2,1} = \frac{2}{3} \cdot 1 + 2 = \frac{8}{3}; \quad \psi_{2,2} = \frac{2}{3} \cdot 2 + 2 = \frac{10}{3}. \tag{6.5.11}$$

Отже, для знаходження наближень розв'язку нашої задачі маємо (див. (6.5.4) — (6.5.6)) систему лінійних алгебраїчних рівнянь

$$\begin{cases} u_{0,0} = 0; \quad u_{0,1} = 1; \quad u_{0,2} = 2; \quad u_{3,0} = 0; \quad u_{3,1} = 2; \quad u_{3,2} = 4; \\ u_{1,0} = 0; \quad u_{2,0} = 0; \quad u_{1,2} = \dfrac{8}{3}; \quad u_{2,2} = \dfrac{10}{3}; \\ (i,j) = (1,1): \quad \dfrac{u_{0,1} + u_{2,1}}{1^2} - \left(\dfrac{2}{1^2} + \dfrac{2}{1^2}\right) u_{1,1} + \dfrac{u_{1,0} + u_{1,2}}{1^2} = 2; \\ (i,j) = (2,1): \quad \dfrac{u_{1,1} + u_{3,1}}{1^2} - \left(\dfrac{2}{1^2} + \dfrac{2}{1^2}\right) u_{2,1} + \dfrac{u_{2,0} + u_{2,2}}{1^2} = 3. \end{cases} \tag{6.5.12}$$

Звідси, зокрема, для знаходження $u_{1,1}, u_{2,1}$ матимемо СЛАР

$$\begin{cases} 1 + u_{2,1} - 4u_{1,1} + 0 + \dfrac{8}{3} = 2 \\ u_{1,1} + 2 - 4u_{2,1} + 0 + \dfrac{10}{3} = 3 \end{cases} \Leftrightarrow \begin{cases} -4u_{1,1} + u_{2,1} = -\dfrac{5}{3} \\ u_{1,1} - 4u_{2,1} = -\dfrac{7}{3} \end{cases}.$$

Розв'язками цієї системи є $u_{1,1} = 3/5$, $u_{2,1} = 11/15$, що разом з відомими значеннями $u_{i,j}$ із системи (6.5.12), отримаємо чисельний розв'язок нашої задачі:

$i \diagdown j$	0	1	2
0	0	1	2
1	0	3/5	8/3
2	0	11/15	10/3
3	0	2	4

.

□

Вправи для самостійної роботи

Методом сіток розв'язати задачі:

1. $\Delta u = 2x - y, \quad (x, y) \in (0\,;2) \times (0\,;3),$

$u|_{x=0} = 0, \; u|_{x=2} = 1, \quad x \in [0\,;3],$

$u|_{y=0} = \frac{1}{2}x, \; u|_{y=3} = \frac{1}{2}x, \quad y \in (0\,;2);$

2. $\Delta u = 2xy, \quad (x, y) \in (0\,;4) \times (0\,;2),$

$u|_{x=0} = 2, \; u|_{x=4} = -2y + 6, \quad y \in [0, 2],$

$u|_{y=0} = x + 2, \; u|_{y=2} = 2, \quad x \in (0, 4).$

6.6. Чисельне розв'язування мішаних задач для параболічних рівнянь. Метод сіток

6.6.1. Постановка мішаної задачі для рівняння теплопровідності

Нехай $l > 0$, $T > 0$ – деякі числа. Позначимо $Q := (0, l) \times (0, T]$, $\overline{Q} = [0, l] \times [0, T]$.

Розглянемо *мішану задачу першого роду для рівняння теплопровідності*: знайти функцію $u \in C^{2,1}(Q) \cap C(\overline{Q})$, яка задовольняє рівняння:

$$u_t - a^2 u_{xx} = f(x, t), \quad (x, t) \in Q, \tag{6.6.1}$$

крайові умови:

$$u|_{x=0} = 0, \quad u|_{x=l} = 0, \quad t \in (0, T], \tag{6.6.2}$$

та початкову умову

$$u|_{t=0} = \varphi(x), \quad x \in [0, l], \tag{6.6.3}$$

де
- $a > 0$ – деяка стала,
- $f \in C(\overline{Q})$, $\varphi \in C([0, l])$ — задані функції, причому виконується умова узгодження: $\varphi(0) = \varphi(l) = 0$.

6.6.2. Різницева схема.

Опишемо метод сіток чисельного розв'язування задачі (6.6.1) — (6.6.3).

Нехай

$$N, M \ \text{ – які-небудь фіксовані натуральні числа}, \quad h := \frac{l}{N}, \quad \tau := \frac{T}{M}.$$

Приймемо

$$x_i := ih, \quad i = \overline{0, N}, \quad t_j := j\tau, \quad j = \overline{0, M}.$$

Нехай

$$u_{i,j} \approx u(x_i, t_j), \quad i = \overline{0, N}, \quad j = \overline{0, M}, \quad —$$

наближення розв'язку даної задачі. Знайдемо

$$f_{i,j} = f(x_i, t_j), \quad i = \overline{1, N-1}, \ j = \overline{0, M-1}; \quad \varphi_i := \varphi(x_i), \quad i = \overline{0, N},$$

і використаємо такі апроксимації похідних функції u:

$$u_{xx}(x_i, t_j) \approx \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2}, \quad i = \overline{1, N-1}, \ j = \overline{0, M-1},$$

$$u_t(x_i, t_j) \approx \frac{u_{i,j+1} - u_{i,j}}{\tau}, \quad i = \overline{1, N-1}, \ j = \overline{0, M-1}.$$

У результаті отримаємо різницеву схему для знаходження чисельного розв'язку задачі (6.6.1) – (6.6.3):

$$\frac{u_{i,j+1} - u_{i,j}}{\tau} - a^2 \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2} = f_{i,j}, \quad i = \overline{1, N-1}, \ j = \overline{0, M-1}, \tag{6.6.4}$$

$$u_{0,j} = 0, \quad u_{N,j} = 0, \quad j = \overline{1, M}, \tag{6.6.5}$$

$$u_{i,0} = \varphi_i, \quad i = \overline{0, N}. \tag{6.6.6}$$

Рівняння (6.6.4) можна переписати у вигляді

$$u_{i,j+1} = a^2 \frac{\tau}{h^2}(u_{i-1,j} + u_{i+1,j}) + \big(1 - a^2 \frac{\tau}{h^2}\big) u_{i,j} + f_{i,j}, \quad i = \overline{1, N-1}, \quad j = \overline{0, M-1}. \tag{6.6.7}$$

!!!напевно в (6.6.7) біля $u_{i,j}$ пропала 2, а біля $f_{i,j}$ τ. Тоді треба поправити формули в розрахунках у прикладі 0.1.7 Різницеву схему (6.6.5), (6.6.6), (6.6.7) називають *явною*.

Обчислення за нею проводять в такому порядку: спочатку за формулами (6.6.5) і (6.6.6) знаходимо $u_{i,0}$, $i = \overline{0, N}$, $u_{0,j}$, $u_{N,j}$, $\quad j = \overline{1, M}$. Далі за формулою (6.6.7) при $j = 0$ знаходимо $u_{i,1}$, $i = \overline{1, N-1}$, а потім за цією ж формулою при $j = 1$ знаходимо $u_{i,2}$, $i = \overline{1, N-1}$, і так продовжуємо включно до $j = M - 1$. На цьому кроці матимемо $u_{i,M}$, $i = \overline{1, N-1}$, а отже, процес розв'язування буде завершено.

Приклад 6.6.1. Методом сіток розв'язати мішану задачу для рівняння теплопровідності

$$u_t - u_{xx} = x^2 + t, \ (x,t) \in (0,3) \times (0,3], \tag{6.6.8}$$

$$u|_{x=0} = 0, \ u|_{x=3} = 0, \quad t \in (0,3], \tag{6.6.9}$$

$$u|_{t=0} = x(3-x), \quad x \in [0,3]. \tag{6.6.10}$$

Розв'язування. Маємо

$$f(x,t) := x^2 + t, \ (x,t) \in (0,3) \times (0,3], \quad \varphi(x) := x(3-x), \quad x \in [0,3].$$

Нехай $N = 3$, $M = 3$. Тоді

$$h = \frac{3}{3} = 1, \ \tau = \frac{3}{3} = 1,$$

а отже,

$$x_i = i, \quad i = \overline{0,3}, \quad \Leftrightarrow \quad x_0 = 0; \ \ x_1 = 1; \ \ x_2 = 2; \ \ x_3 = 3;$$

$$t_j = j, \quad j = \overline{0,3}, \quad \Leftrightarrow \quad t_0 = 0; \ \ t_1 = 1; \ \ t_2 = 2; \ \ t_3 = 3.$$

Нехай

$$u_{i,j} \approx u(i,j), \quad i = \overline{0,3}, \ j = \overline{0,3}, \quad -$$

наближення розв'язку нашої задачі.

Знаходимо

$$f_{ij} := x_i^2 + t_j = i^2 + j, \quad i = \overline{1,2}, \ j = \overline{0,2}, \quad \Leftrightarrow$$

$$\Leftrightarrow \quad f_{1,0} = 1^2 + 0 = 1; \ \ f_{2,0} = 4; \ \ f_{1,1} = 2; \ \ f_{2,1} = 5; \ \ f_{1,2} = 3; \ \ f_{2,2} = 6.$$

$$\varphi_i := x_i(3 - x_i) = i(3-i), \quad i = \overline{0,3}, \quad \Leftrightarrow \quad \varphi_0 = 0; \ \ \varphi_1 = 2; \ \ \varphi_2 = 2; \ \ \varphi_3 = 0.$$

Отже, на підставі (6.6.5), (6.6.6), (6.6.7) отримуємо систему рівнянь для знаходження $u_{i,j}$, $i = \overline{0,3}$, $j = \overline{0,3}$:

$$\begin{cases} u_{0,1} = 0; \ \ u_{0,2} = 0; \ \ u_{0,3} = 0; \ \ u_{3,1} = 0; \ \ u_{3,2} = 0; \ \ u_{3,3} = 0; \\ \\ u_{0,0} = 0; \ \ u_{1,0} = 2; \ \ u_{2,0} = 2; \ \ u_{3,0} = 0; \\ \\ u_{i,j+1} = 4(u_{i-1,j} + u_{i+1,j}) - 3u_{i,j} + f_{i,j}, \quad i = \overline{1,2}, \ j = \overline{0,2}, \end{cases}$$

Звідси, зокрема, знаходимо

$$\begin{cases} j = 0, \ i = 1: \ u_{1,1} = 4(u_{0,0} + u_{2,0}) - 3u_{1,0} + f_{1,0} \\ j = 0, \ i = 2: \ u_{2,1} = 4(u_{1,0} + u_{3,0}) - 3u_{2,0} + f_{2,0} \end{cases} \quad \Leftrightarrow$$

$$\Leftrightarrow \quad \begin{cases} u_{1,1} = 4(0+2) - 3 \cdot 2 + 1 = 3 \\ u_{2,1} = 4(2+0) - 3 \cdot 2 + 4 = 6 \end{cases},$$

$$\begin{cases} j = 1, \ i = 1: \ u_{1,2} = 4(u_{0,1} + u_{2,1}) - 3u_{1,1} + f_{1,1} \\ j = 1, \ i = 2: \ u_{2,2} = 4(u_{1,1} + u_{3,1}) - 3u_{2,1} + f_{2,1} \end{cases} \quad \Leftrightarrow$$

$$\Leftrightarrow \quad \begin{cases} u_{1,2} = 4(0+6) - 3 \cdot 3 + 2 = 17 \\ u_{2,2} = 4(3+0) - 3 \cdot 6 + 5 = -1 \end{cases},$$

$$\begin{cases} j = 2, \ i = 1: \ u_{1,3} = 4(u_{0,2} + u_{2,2}) - 3u_{1,2} + f_{1,2} \\ j = 2, \ i = 2: \ u_{2,3} = 4(u_{1,2} + u_{3,2}) - 3u_{2,2} + f_{2,2} \end{cases} \quad \Leftrightarrow$$

$$\Leftrightarrow \quad \begin{cases} u_{1,3} = 4(0-1) - 3 \cdot 17 + 3 = -52 \\ u_{2,3} = 4(18+0) - 3 \cdot (-1) + 6 = 81 \end{cases}.$$

Отож, знайдене наближення розв'язку даної задачі може бути задане таблицею

$i \diagdown j$	0	1	2	3
0	0	0	0	0
1	2	3	17	-52
2	2	6	-1	77
3	0	0	0	0

.

□

Вправи для самостійної роботи:

Методом сіток розв'язати мішані задачі для рівняння теплопровідності:

1. $u_t - u_{xx} = 2x - t, \quad (x,t) \in (0\,;3) \times (0\,;4],$

$u|_{x=0} = 0, \quad u|_{x=3} = 0, \quad t \in (0\,;4],$

$u|_{t=0} = 3x(3-x), \quad x \in [0\,;3].$

2. $u_t - 9u_{xx} = xt, \quad (x,t) \in (0\,;4) \times (0\,;2],$

$u|_{x=0} = 0, \quad u|_{x=4} = 0, \quad t \in (0\,;4],$

$u|_{t=0} = \sin \pi x, \quad x \in [0\,;4].$

6.7. Чисельне розв'язування мішаних задач для гіперболічних рівнянь. Метод сіток

6.7.1. Постановка мішаної задачі для рівняння коливання струни

Нехай $l > 0$, $T > 0$ – деякі числа. Позначимо $Q := (0,l) \times (0,T]$, $\overline{Q} = [0,l] \times [0,T]$.

Розглянемо *мішану задачу для рівняння коливання струни*: знайти функцію $u \in C^2(Q) \cap C^1(\overline{Q})$, яка задовольняє рівняння:

$$u_{tt} - a^2 u_{xx} = f(x,t), \quad (x,t) \in Q, \tag{6.7.1}$$

крайові умови:

$$u|_{x=0} = 0, \quad u|_{x=l} = 0, \quad t \in (0,T], \tag{6.7.2}$$

та початкову умову

$$u|_{t=0} = \varphi(x), \quad u_t|_{t=0} = \psi(x), \quad x \in [0,l], \tag{6.7.3}$$

де
• $a > 0$ – деяка стала,
• $f \in C(\overline{Q})$, φ, $\psi \in C([0,l])$ — задані функції, причому виконується умова узгодження: $\varphi(0) = \varphi(l) = 0$, $\psi(0) = \psi(l) = 0$.

6.7.2. Різницева схема.

Опишемо метод сіток чисельного розв'язування задачі (6.7.1) — (6.7.3). Нехай

$$N, M \ \text{– деякі натуральні числа}, \quad h := \frac{l}{N}, \quad \tau := \frac{T}{M}.$$

Приймемо

$$x_i := ih, \quad i = \overline{0,N}, \quad t_j := j\tau, \quad j = \overline{0,M}.$$

Нехай

$$u_{i,j} \approx u(x_i, t_j), \quad i = \overline{0,N}, \quad j = \overline{0,M}. \quad —$$

наближення розв'язку нашої задачі.

Використаємо такі апроксимації похідних функції u:

$$u_{xx}(x_i,t_j) \approx \frac{u_{i-1,j}-2u_{i,j}+u_{i+1,j}}{h^2}, \quad i=\overline{1,N-1},\ j=\overline{1,M-1},$$

$$u_t(x_i,t_j) \approx \frac{u_{i,j+1}-u_{i,j}}{\tau}, \quad i=\overline{1,N-1},\ j=\overline{0,M-1},$$

$$u_{tt}(x_i,t_j) \approx \frac{u_{i,j-1}-2u_{i,j}+u_{i,j+1}}{\tau^2}, \quad i=\overline{1,N-1},\ j=\overline{1,M-1}.$$

Покладемо

$$f_{i,j} := f(x_i,t_j), \quad i=\overline{1,N-1},\ j=\overline{0,M-1};$$
$$\varphi_i := \varphi(x_i), \quad \psi_i := \psi(x_i), \quad i=\overline{0,N}.$$

У результаті отримаємо різницеву схему для знаходження чисельного розв'язку задачі (6.7.1) – (6.7.3):

$$\frac{u_{i,j-1}-2u_{i,j}+u_{i,j+1}}{\tau^2} - a^2\frac{u_{i-1,j}-2u_{i,j}+u_{i+1,j}}{h^2} = f_{i,j}, \tag{6.7.4}$$
$$i=\overline{1,N-1},\ j=\overline{1,M-1},$$
$$u_{0,j}=0, \quad u_{N,j}=0, \quad j=\overline{2,M}, \tag{6.7.5}$$
$$u_{i,0}=\varphi_i, \quad \frac{u_{i,1}-u_{i,0}}{\tau}=\psi_i, \quad i=\overline{0,N}. \tag{6.7.6}$$

З (6.7.4) — (6.7.6) маємо

$$u_{i,j+1} = pu_{i+1,j} + 2(1-p)u_{i,j} + pu_{i-1,j} - u_{i,j-1} + \tau^2 f_{i,j}, \tag{6.7.7}$$
$$i=\overline{1,N-1}, \quad j=\overline{1,M-1},$$
$$u_{0,j}=0, \quad u_{N,j}=0, \quad j=\overline{2,M}, \tag{6.7.8}$$
$$u_{i,0}=\varphi_i, \quad u_{i,1}=\varphi_i+\tau\psi_i, \quad i=\overline{0,N}, \tag{6.7.9}$$

де $p := \dfrac{\tau^2}{h^2}a^2$.

Обчислення за нею проводять в такому порядку: спочатку беремо $j=0$ і на підставі першої з рівностей (6.7.9) по черзі знаходимо $u_{i,0}$, $i=\overline{0,N}$. Далі беремо $j=1$ і на підставі другої з рівностей (6.7.9) по черзі знаходимо $u_{i,1}$, $i=\overline{0,N}$. Далі, викорстовуючи рівняння (6.7.7) і (6.7.8), знаходимо $u_{i,j}$, $i=\overline{0,N}$, $j=\overline{2,M}$.

Приклад 6.7.1. Методом сіток розв'язати мішану задачу для рівняння коливання струни:

$$u_{tt}-u_{xx}=x+t, \quad (x,t)\in(0,4)\times(0,3], \tag{6.7.10}$$
$$u|_{x=0}=0, \quad u|_{x=4}=0, \quad t\in(0,4], \tag{6.7.11}$$
$$u|_{t=0}=x(4-x), \quad u_t|_{t=0}=\sin\frac{\pi}{2}x, \quad x\in[0,4]. \tag{6.7.12}$$

Розв'язування. Маємо

$$f(x,t) := x+t, \quad (x,t)\in(0,4)\times(0,3],$$
$$\varphi(x) := x(4-x), \quad \psi(x) := \sin\frac{\pi}{2}x, \quad x\in[0,4].$$

Нехай $N=4$, $M=3$. Тоді

$$h := \frac{4}{4}=1; \quad \tau := \frac{3}{3}=1; \quad p:=1.$$
$$x_i=i,\ i=\overline{0,4}, \quad \Leftrightarrow \quad x_0=0;\ x_1=1;\ x_2=2;\ x_3=3;\ x_4=4;$$
$$t_j=j,\ j=\overline{0,3}, \quad \Leftrightarrow \quad t_0=0;\ t_1=1;\ t_2=2;\ t_3=3.$$

Для знаходження наближень

$$u_{i,j} \approx u(i,j), \quad i = \overline{0,4},\ j = \overline{0,3},$$

використаємо систему рівнянь (6.7.7), (6.7.8), (6.7.9). Але спочатку визначимо

$$f_{i,j} := x_i + t_j, \quad i = \overline{1,3},\ j = \overline{1,2}, \quad \Leftrightarrow$$

$$\Leftrightarrow \quad f_{1,1} := 2; \quad f_{1,2} := 3; \quad f_{2,1} := 3; \quad f_{2,2} := 4; \quad f_{3,1} := 4; \quad f_{3,2} := 5;$$

$$\varphi_i := x_i(4 - x_i), \quad i = \overline{0,4}, \quad \Leftrightarrow$$

$$\Leftrightarrow \varphi_0 := 0; \quad \varphi_1 := 3; \quad \varphi_2 := 4; \quad \varphi_3 := 3; \quad \varphi_4 := 0;$$

$$\psi_i := \sin\frac{\pi}{2}x_i, \quad i = \overline{0,4}, \quad \Leftrightarrow$$

$$\Leftrightarrow \psi_0 := 0; \quad \psi_1 := 1; \quad \psi_2 := 0; \quad \psi_3 := -1; \quad \psi_4 := 0.$$

Згідно з (6.7.7), (6.7.8), (6.7.9) маємо

$$\begin{cases} u_{i,0} = \varphi_i, \quad i = \overline{0,4}, \quad \Leftrightarrow \quad u_{0,0} = 0;\ u_{1,0} = 3;,\ u_{2,0} = 4;\ u_{3,0} = 3;\ u_{4,0} = 0; \\ u_{i,1} = \varphi_i + \tau\psi_i, \quad i = \overline{0,4}, \quad \Leftrightarrow \quad u_{0,1} = 0;\ u_{1,1} = 4;\ u_{2,1} = 4; u_{3,1} = 2;\ u_{4,1} = 0; \\ u_{0,2} = 0;\ u_{0,3} = 0;\ u_{4,2} = 0;\ u_{4,3} = 0; \\ u_{i,j+1} = u_{i+1,j} + u_{i-1,j} - u_{i,j-1} + f_{i,j}, \quad i = \overline{1,3}, \quad j = \overline{1,2}. \end{cases}$$

Звідси, зокрема, знаходимо:

$$\begin{cases} j = 1,\ i = 1:\ u_{1,2} = u_{2,1} + u_{0,1} - u_{1,0} + f_{1,1} \\ j = 1,\ i = 2:\ u_{2,2} = u_{3,1} + u_{1,1} - u_{2,0} + f_{2,1} \\ j = 1,\ i = 3:\ u_{3,2} = u_{4,1} + u_{2,1} - u_{3,0} + f_{3,1} \end{cases} \Leftrightarrow \begin{cases} u_{1,2} = 4 + 0 - 3 + 2 = 3 \\ u_{2,2} = 2 + 4 - 4 + 3 = 5 \\ u_{3,2} = 0 + 4 - 3 + 4 = 5 \end{cases},$$

$$\begin{cases} j = 2,\ i = 1:\ u_{1,3} = u_{2,2} + u_{0,2} - u_{1,1} + f_{1,2} \\ j = 2,\ i = 2:\ u_{2,3} = u_{3,2} + u_{1,2} - u_{2,1} + f_{2,2} \\ j = 2,\ i = 3:\ u_{3,3} = u_{4,2} + u_{2,2} - u_{3,1} + f_{3,2} \end{cases} \Leftrightarrow \begin{cases} u_{1,3} = 5 + 0 - 4 + 3 = 4 \\ u_{2,3} = 5 + 3 - 4 + 4 = 8 \\ u_{3,3} = 0 + 5 - 2 + 5 = 8 \end{cases}.$$

Знайдене наближення розв'язку даної задачі таке:

$i \diagdown j$	0	1	2	3
0	0	0	0	0
1	3	4	3	4
2	4	4	5	8
3	3	2	5	8
4	0	0	0	0

.

□

Вправи для самостійної роботи:

Методом сіток розв'язати мішані задачі для рівняння коливання струни:

1. $u_{tt} - 4u_{xx} = 2x - t, \quad (x,t) \in (0\,;3) \times (0\,;4],$

$u|_{x=0} = 0, \quad u|_{x=3} = 0, \quad t \in (0\,;4],$

$u|_{t=0} = 2x(3-x), \quad u_t|_{t=0} = 3\sin \pi x, \quad x \in [0\,;3].$

2. $u_{tt} - 9u_{xx} = xt, \quad (x,t) \in (0\,;4) \times (0\,;2],$

$u|_{x=0} = 0, \quad u|_{x=4} = 0, \quad t \in (0\,;4],$

$u|_{t=0} = x^2(4-x), \quad u_t|_{t=0} = \cos \dfrac{\pi(x+1)}{2}, \quad x \in [0\,;4].$

6.8. Лабораторний практикум з чисельного розв'язування задач для диференціальних рівнянь та їх систем

6.8.1 Застосування методу Ейлера до задачі Коші для звичайних диференціальних рівнянь першого порядку

Розв'язок задачі Коші (6.1.1),(6.1.2) розглядаємо на відрізку $[a, b]$, коли $x_0 = a$.

Задаємо рівномірне розбиття відрізка $[a, b]$ точками $x_0, x_1, \ldots, x_n$, де $n \in \mathbb{N}$ і $x_i := a + ih, \quad i = \overline{0, n}, \quad h := \frac{b-a}{n}$.

Чисельним розв'язком задачі (6.1.1),(6.1.2) вважаємо наближення u_i, $i = \overline{1, n}$, відповідно, значень $u(x_i)$, $i = \overline{1, n}$, розв'язку цієї задачі, який знаходимо за рекурентною формулою

$$u_{i+1} = u_i + h\, f(x_i, u_i), \quad i = \overline{0, n-1}, \tag{6.8.1}$$

беручи значення u_0 з початкової умови (6.1.2).

Пояснення до використання програмного коду

- Підготувати середовище і потрібні функції :
  1. виконати комірку для підготовки середовища
  2. виконати комірку, де **визначена** функція `Euler_method`
- Обчислити чисельний розв'язок конкретної задачі Коші
  1. виконати комірку, де **визначена** функція `f`
  2. виконати комірку з викликом функції `Euler_method`, перед виконанням задати відповідні аргументи цієї функції.
- Щоб переконатися, що чисельний розв'язок достатньо точний, можна виконати кілька послідовних викликів функції `Euler_method`, збільшуючи кількість вузлів сітки.

Програмна реалізація методів

Підготовка середовища

[1]:
```
%matplotlib widget
import matplotlib.pyplot as plt

import numpy as np
import pandas as pd
```

`Euler_method` – функція, яка реалізує явний метод Ейлера для знаходження чисельного розв'язку задачі Коші

[2]:
```
def Euler_method(f,u0,a,b,n):
    """ явний метод Ейлера
    """
    h=(b-a)/n
    x=np.linspace(a, b, n+1)

    u=np.empty(n+1)
    u[0]=u0
    for i in range(n):
        u[i+1] = u[i] + h*f(x[i],u[i])

    return u
```

`f` – функція, яка задає обчислення правої частини конкретного рівняння

Обчислювальні експерименти
Продемонструємо застосування методу Ейлера для знаходження чисельного розв'язку задачі Коші.

Приклад 1. За допомогою методу Ейлера знайти чисельний розв'язок задачі Коші (6.1.1),(6.1.2), коли $f(x,u) = u^2 + 2x - x^4, \quad [a,b] = [0,\ 1], \quad u(0) = 0.$

Очевидно, що функція $u(x) = x^2$ є точним розв'язком задачі Коші у цьому випадку.

Визначимо функцію правої частини рівняння (6.1.1):

[3]:
```
def f2(x, u):
    return u**2+2*x-x**4
```

Задамо координати відрізка і початкові дані:

[4]:
```
a=0
b=1
u0=0
```

Знайдемо чисельні розв'язки задачі на послідовності сіток, подвоюючи на кожному кроці кількість вузлів і зберігаючи відповідні масиви для подальшого аналізу:

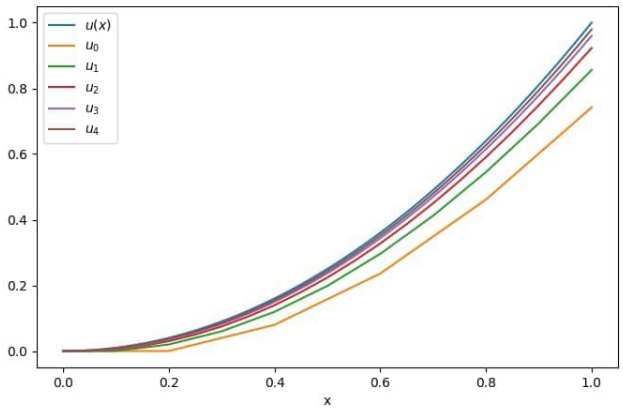
[5]:
```
n_start = 5

n = n_start
x0=np.linspace(a, b, n+1)
u_0=Euler_method(f2,u0,a,b,n)

n*=2
x1=np.linspace(a, b, n+1)
u_1=Euler_method(f2,u0,a,b,n)

n*=2
x2=np.linspace(a, b, n+1)
u_2=Euler_method(f2,u0,a,b,n)

n*=2
x3=np.linspace(a, b, n+1)
u_3=Euler_method(f2,u0,a,b,n)

n*=2
x4=np.linspace(a, b, n+1)
u_4=Euler_method(f2,u0,a,b,n)
```

Порахуємо також значення точного розв'язку задачі на рівномірній сітці:

[6]:
```
x=np.linspace(a, b, 256)
ux=x**2
```

Побудуємо графіки отриманих чисельних і точного розв'язків задачі:

[61]:
```
fig = plt.figure(figsize=(8, 5))
plt.plot(x, ux, label='$u(x)$')
plt.plot(x0, u_0, label='$u_0$')
plt.plot(x1, u_1, label='$u_1$')
plt.plot(x2, u_2, label='$u_2$')
plt.plot(x3, u_3, label='$u_3$')
plt.plot(x4, u_4, label='$u_4$')
ax = fig.gca()
ax.legend()
ax.set_xlabel('x')
```



Зауважимо, що графічні побудови за вибраного режиму виконуються з автоматичною лінійною інтертерполяцією чисельних розв'язків, які насправді обчислені лише у вузлах відповідної сітки. Нагадаємо, що отримані зобаження можна масштабувати, використовуючи режим 'Zoom to rectangle'.

Як бачимо, графіки чисельних розв'язків візуально близькі до графіка точного розв'язку. Для детальнішого аналізу занесемо в таблицю значення усіх розв'язків на спільній множині вузлів `x_0`:

[8]:
```
u = x0**2

df=pd.DataFrame({'x_0':x0[1::],'u_0':u_0[1::],'u_1':u_1[2::2],'u_2':u_2[4::
↪4],'u_3':u_3[8::8],'u_4':u_4[16::16],'u':u[1::]})
df
```

[8]:

	x_0	u_0	u_1	u_2	u_3	u_4	u
0	0.2	0.000000	0.019990	0.029982	0.034986	0.037492	0.04
1	0.4	0.079680	0.119418	0.139550	0.149727	0.154851	0.16
2	0.6	0.235830	0.295966	0.327335	0.343481	0.351690	0.36
3	0.8	0.461033	0.544711	0.590442	0.614666	0.627183	0.64
4	1.0	0.741623	0.855895	0.922810	0.959860	0.979503	1.00

На множині `x_0` маємо такі значення абсолютних похибок отриманих чисельних розв'язків:

[9]:
```
df1=pd.DataFrame()
df1['e_0']=np.abs(df['u']-df['u_0'])
df1['e_1']=np.abs(df['u']-df['u_1'])
df1['e_2']=np.abs(df['u']-df['u_2'])
df1['e_3']=np.abs(df['u']-df['u_3'])
df1['e_4']=np.abs(df['u']-df['u_4'])
df1
```

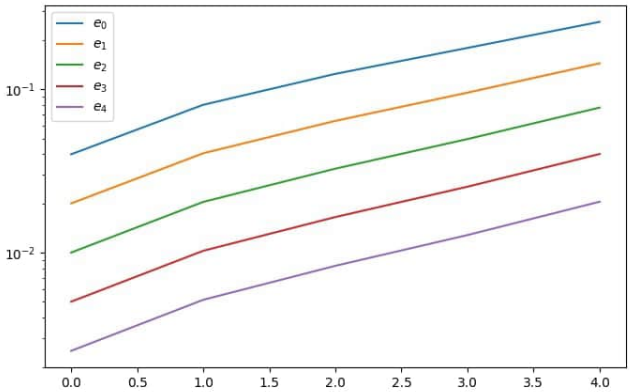
[9]:

	e_0	e_1	e_2	e_3	e_4
0	0.040000	0.020010	0.010018	0.005014	0.002508
1	0.080320	0.040582	0.020450	0.010273	0.005149
2	0.124170	0.064034	0.032665	0.016519	0.008310
3	0.178967	0.095289	0.049558	0.025334	0.012817
4	0.258377	0.144105	0.077190	0.040140	0.020497

Побудуємо графіки абсолютних похибок у логарифмічній шкалі:

[64]:
```
fig = plt.figure(figsize=(8, 5))
df1.e_0.plot(logy=True, label = '$e_0$')
```

```
df1.e_1.plot(logy=True, label = '$e_1$')
df1.e_2.plot(logy=True, label = '$e_2$')
df1.e_3.plot(logy=True, label = '$e_3$')
df1.e_4.plot(logy=True, label = '$e_4$')
ax = fig.gca()
ax.legend()
```



Як бачимо, при згущенні сітки похибки чисельних розв'язків на кожному кроці зменшуються. Зазначимо, що на фіксованій сітці вони дещо відрізняються між собою, зростаючи у вузлах поблизу правого кінця відрізку.

Обчислимо для кожного чисельного розв'язку його абсолютну похибку за нормою $\|\cdot\|_\infty$

[12]:
```
ne_0 = max(np.abs(df1['e_0']))
ne_1 = max(np.abs(df1['e_1']))
ne_2 = max(np.abs(df1['e_2']))
ne_3 = max(np.abs(df1['e_3']))
ne_4 = max(np.abs(df1['e_4']))
print(f" ne_0 = {ne_0:1.2e}\n ne_1 = {ne_1:1.2e}\n ne_2 = {ne_2:1.2e}\n␣
↪ne_3 = {ne_3:1.2e}\n ne_4 = {ne_4:1.2e}")
```

```
 ne_0 = 2.58e-01
 ne_1 = 1.44e-01
 ne_2 = 7.72e-02
 ne_3 = 4.01e-02
 ne_4 = 2.05e-02
```

Оцінимо швидкість збіжності чисельних розв'язків до точного, а саме у скільки разів зменшуватиметься абсолютна похибка при подвоєнні кількості вузлів сітки:

[13]:
```
df2=pd.DataFrame()
df2['r_0'] = df1['e_0'] / df1['e_1']
df2['r_1'] = df1['e_1'] / df1['e_2']
df2['r_2'] = df1['e_2'] / df1['e_3']
df2['r_3'] = df1['e_3'] / df1['e_4']

df2
```

[13]:

	r_0	r_1	r_2	r_3
0	1.999000	1.997379	1.998048	1.998849
1	1.979222	1.984420	1.990733	1.994970

```
2  1.939134  1.960322  1.977406  1.987941
3  1.878148  1.922780  1.956151  1.976559
4  1.792979  1.866874  1.923033  1.958333
```

Як бачимо, швидкість наближається до числа 2, що узгоджується з теоретичними оцінками похибок методу Ейлера.

[14]:
```
plt.close('all')
```

Приклад 2. $f(x,u) = -u\,cosx + e^{-sinx}, \quad [a,b] = [0,\,10], \quad u(0) = 1.$

Відомо [(1.69 Самойленко)] , що функція $u(x) = (x+1)e^{-sinx}$ є точним розв'язком задачі Коші у цьому випадку.

Задамо функцію правої частини диференціального рівняння і решту даних задачі Коші:

[15]:
```
def f3(x, u):
    return -u*np.cos(x) + np.exp(-np.sin(x))
```

[16]:
```
a=0
b=10
u0=1
```

Оскільки на цей раз розглядаємо задачу на більшому відрізку і поведінка розв'язку є складнішою, то відразу почнемо шукати чисельні розв'язки при більшому значенні параметра `n_start`, який задає початкову кількість вузлів сітки. Далі повторимо усі кроки, якими отримали і досліджували чисельні розв'язки попередньої задачі:

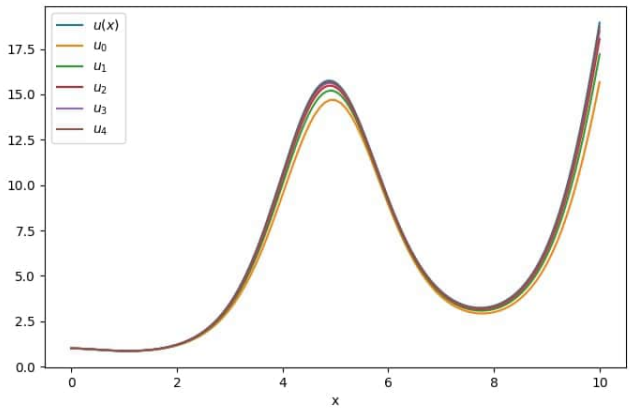
[70]:
```
n_start = 100

n = n_start
x0=np.linspace(a, b, n+1)
u_0=Euler_method(f3,u0,a,b,n)

n*=2
x1=np.linspace(a, b, n+1)
u_1=Euler_method(f3,u0,a,b,n)

n*=2
x2=np.linspace(a, b, n+1)
u_2=Euler_method(f3,u0,a,b,n)

n*=2
x3=np.linspace(a, b, n+1)
u_3=Euler_method(f3,u0,a,b,n)

n*=2
x4=np.linspace(a, b, n+1)
u_4=Euler_method(f3,u0,a,b,n)

x=np.linspace(a, b, 256)
ux=(x+1)*np.exp(-np.sin(x))

fig = plt.figure(figsize=(8, 5))
plt.plot(x, ux, label='$u(x)$')
plt.plot(x0, u_0, label='$u_0$')
plt.plot(x1, u_1, label='$u_1$')
plt.plot(x2, u_2, label='$u_2$')
plt.plot(x3, u_3, label='$u_3$')
```

```
plt.plot(x4, u_4, label='$u_4$')
ax = fig.gca()
ax.legend();
ax.set_xlabel('x')
u=(x0+1)*np.exp(-np.sin(x0))

df=pd.DataFrame({'x_0':x0[1::],'u_0':u_0[1::],'u_1':u_1[2::2],'u_2':u_2[4::
 ↪4],'u_3':u_3[8::8],'u_4':u_4[16::16],'u':u[1::]})
df
```

[18]:

	x_0	u_0	u_1	u_2	u_3	u_4	u
0	0.1	1.000000	0.997625	0.996528	0.996001	0.995742	0.995487
1	0.2	0.990998	0.987197	0.985446	0.984604	0.984192	0.983785
2	0.3	0.975856	0.971386	0.969332	0.968346	0.967864	0.967388
3	0.4	0.957043	0.952486	0.950399	0.949401	0.948913	0.948431
4	0.5	0.936639	0.932425	0.930508	0.929594	0.929148	0.928708
..	...	...	...	...	...	...	...
95	9.6	10.498641	11.492380	12.037450	12.323347	12.469819	12.618712
96	9.7	11.651473	12.770818	13.385431	13.707976	13.873266	14.041319
97	9.8	12.903998	14.157057	14.845542	15.206970	15.392217	15.580580
98	9.9	14.248884	15.641425	16.406735	16.808539	17.014490	17.223913
99	10.0	15.673900	17.208556	18.051831	18.494522	18.721419	18.952131

[100 rows x 7 columns]



Оскільки розглядаємо задачу на довшому відрізку і, як видно з графіку, поведінка її розв'язку є складнішою, то похибка отриманих чисельних розв'язків є більшою у цьому випадку:

[19]:

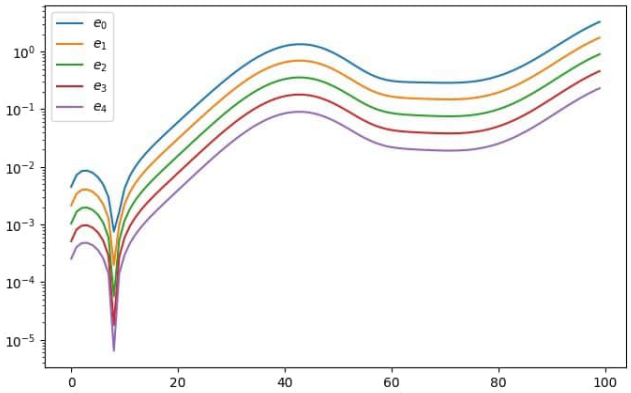
```
df1=pd.DataFrame()
df1['e_0']=np.abs(df['u']-df['u_0'])
df1['e_1']=np.abs(df['u']-df['u_1'])
df1['e_2']=np.abs(df['u']-df['u_2'])
df1['e_3']=np.abs(df['u']-df['u_3'])
df1['e_4']=np.abs(df['u']-df['u_4'])
df1
```

[19]:

	e_0	e_1	e_2	e_3	e_4
0	0.004513	0.002138	0.001041	0.000514	0.000255
1	0.007213	0.003412	0.001661	0.000819	0.000407
2	0.008468	0.003998	0.001944	0.000959	0.000476

```
3   0.008612  0.004055  0.001968  0.000970  0.000481
4   0.007931  0.003716  0.001799  0.000885  0.000439
..       ...       ...       ...       ...       ...
95  2.120071  1.126332  0.581262  0.295365  0.148893
96  2.389846  1.270501  0.655888  0.333344  0.168053
97  2.676582  1.423523  0.735038  0.373610  0.188363
98  2.975028  1.582488  0.817177  0.415374  0.209423
99  3.278231  1.743575  0.900300  0.457609  0.230713

[100 rows x 5 columns]
```

[72]:
```
fig = plt.figure(figsize=(8, 5))
df1.e_0.plot(logy=True, label = '$e_0$')
df1.e_1.plot(logy=True, label = '$e_1$')
df1.e_2.plot(logy=True, label = '$e_2$')
df1.e_3.plot(logy=True, label = '$e_3$')
df1.e_4.plot(logy=True, label = '$e_4$')
ax = fig.gca()
ax.legend();
```



Обчислимо для кожного чисельного розв'язку його абсолютну похибку за нормою $\|\cdot\|_\infty$

[21]:
```
ne_0 = max(np.abs(df1['e_0']))
ne_1 = max(np.abs(df1['e_1']))
ne_2 = max(np.abs(df1['e_2']))
ne_3 = max(np.abs(df1['e_3']))
ne_4 = max(np.abs(df1['e_4']))
print(f" ne_0={ne_0:1.2e}\n ne_1={ne_1:1.2e}\n ne_2={ne_2:1.2e}\n␣
↪ne_3={ne_3:1.2e}\n ne_4={ne_4:1.2e}")
```

```
 ne_0=3.28e+00
 ne_1=1.74e+00
 ne_2=9.00e-01
 ne_3=4.58e-01
 ne_4=2.31e-01
```

Знайдемо тепер значення швидкості збіжності чисельних розв'язків:

[22]:
```
df2=pd.DataFrame()
df2['r_0'] = df1['e_0'] / df1['e_1']
df2['r_1'] = df1['e_1'] / df1['e_2']
```

```
df2['r_2'] = df1['e_2'] / df1['e_3']
df2['r_3'] = df1['e_3'] / df1['e_4']

df2
```

[22]:

	r_0	r_1	r_2	r_3
0	2.110890	2.053274	2.026133	2.012945
1	2.113901	2.054808	2.026907	2.013334
2	2.117906	2.056875	2.027957	2.013863
3	2.123949	2.060006	2.029549	2.014665
4	2.134008	2.065217	2.032199	2.016001
..	...	...	...	...
95	1.882280	1.937735	1.967946	1.983733
96	1.881026	1.937069	1.967603	1.983558
97	1.880252	1.936665	1.967396	1.983454
98	1.879969	1.936529	1.967329	1.983421
99	1.880178	1.936659	1.967402	1.983459

[100 rows x 4 columns]

Як бачимо, отримані значення швидкості наближаються до числа 2, як і передбачають результати теоретичних досліджень. Зазначимо, що це повільна збіжність, тому при обчисленні розв'язку на сітках з більшою кількістю вузлів (при більшому значенні параметра `n_start`) наступить такий момент, коли похибка почне зростати через похибку арифметичних операцій. Така властивість методу Ейлера обмежує його застосування і оправдовує його використання хіба для отримання певних уявлень про розв'язок задачі Коші.

[ ]:
```
plt.close('all')
```

6.8.2 Метод Рунге-Кутта

Нехай задача (6.1.1),(6.1.2) має єдиний розв'язок, визначений на відрізку $[a, b]$, де $a = x_0$, і ми шукаємо $u_1, \ldots, u_n$ – наближення значень цього розв'язку в точках $x_1, \ldots, x_n$, де $n \in \mathbb{N}$ – деяке число, а

$$x_i := a + ih, \quad i = \overline{0, n}, \qquad h := (b - a)/n.$$

Чисельним розв'язком задачі (6.1.1),(6.1.2) вважаємо наближення u_i, $i = \overline{1, n}$, (u_0 беремо з початкової умови (6.1.2)) за правилом

$$u_{i+1} = u_i + \sum_{j=1}^{r} p_{r,j} k_{i,j}(h), \tag{6.8.2}$$

де r – деяке натуральне число,

$$k_{i,1}(h) := f(x_i, u_i) \cdot h,$$

$$k_{i,j}(h) := f\big(x_i + \alpha_j h, u_i + \sum_{l=1}^{j-1} \beta_{j,l} k_{i,l}(h)\big) \cdot h, \quad j = \overline{2, r},$$

а коефіцієнти $p_{r,k}, \alpha_j, \beta_{j,l}$, $k = \overline{1, r}, j = \overline{2, r}, l = \overline{1, j-1}$, знаходять із відповідної системи рівнянь.

На практиці часто застосовують метод Рунге-Кутта четвертого порядку, який одержуть при $r = 4$. У цьому випадку можна використати такі розрахункові формули:

$$u_{i+1} = u_i + \frac{1}{6} k_{i,1}(h) + \frac{1}{3} k_{i,2}(h) + \frac{1}{3} k_{i,3}(h) + \frac{1}{6} k_{i,4}(h),$$

де

$$k_{i,1}(h) := f(x_i, u_i) \cdot h, \quad k_{i,2}(h) := f\left(x_i + \frac{1}{2} h, u_i + \frac{1}{2} k_{i,1}(h)\right) \cdot h,$$

$$k_{i,3}(h) := f(x_i + \frac{1}{2} h, u_i + \frac{1}{2} k_{i,2}(h)) \cdot h, \quad k_{i,4}(h) := f(x_i + h, u_i + k_{i,3}(h)) \cdot h, \quad i = \overline{0, n-1}.$$

Пояснення до використання програмного коду

- Підготувати середовище і потрібні функції :
  1. виконати комірку для підготовки середовища
  2. виконати комірку з функцією `RK4_method`
- Обчислити чисельний розв'язок конкретної задачі Коші
  1. виконати комірку, де **визначена** функція `f`
  2. виконати комірку з викликом функції `RK4_method`, перед виконанням задати відповідні аргументи цієї функції.
- Щоб переконатися, що чисельний розв'язок достатньо точний, можна виконати кілька послідовних викликів функції `RK4_method`, збільшуючи кількість вузлів сітки.

Програмна реалізація методів

Підготовка середовища

[1]:
```
%matplotlib widget
import matplotlib.pyplot as plt

import numpy as np
import pandas as pd
```

`RK4_method` – функція, яка реалізує явний метод Ейлера для знаходження чисельного розв'язку задачі Коші

[2]:
```
def RK4_method(f,u0,a,b,n):
    """ метод Рунге-Кутта четвертого порядку
    """
    h=(b-a)/n
    x=np.linspace(a, b, n+1)


    u=np.empty(n+1)
    u[0]=u0
    for i in range(n):
        k1 = f(x[i], u[i])
        k2 = f(x[i] + h/2, u[i] + h/2*k1)
        k3 = f(x[i] + h/2, u[i] + h/2*k2)
        k4 = f(x[i+1], u[i] + h*k3)

        u[i+1] =u [i] + h/6*(k1 + 2*k2 + 2*k3 + k4)

    return u
```

f – функція, яка задає обчислення правої частини конкретного рівняння

Обчислювальні експерименти
Продемонструємо застосування методу Рунге-Кутта для знаходження чисельного розв'язку задачі Коші.

Приклад 1. За допомогою методу Рунге-Кутта 4-го порядку знайти чисельний розв'язок задачі Коші (6.1.1),(6.1.2), коли $f(x,u) = u^2 + 2x - x^4, \quad [a,b] = [0,\,1], \quad u(0) = 0.$

Очевидно, що функція $u(x) = x^2$ є точним розв'язком задачі Коші у цьому випадку.

Визначимо функцію правої частини рівняння (6.1.1):

[3]:
```
def f2(x, u):
    return u**2+2*x-x**4
```

Задамо координати відрізка і початкові дані:

[4]:
```
a=0
b=1
u0=0
```

Знайдемо чисельні розв'язки задачі на послідовності сіток, подвоюючи на кожному кроці кількість вузлів і зберігаючи відповідні масиви для подальшого аналізу:

[5]:
```
n_start = 5

n = n_start
x0=np.linspace(a, b, n+1)
u_0=RK4_method(f2,u0,a,b,n)

n*=2
x1=np.linspace(a, b, n+1)
u_1=RK4_method(f2,u0,a,b,n)

n*=2
x2=np.linspace(a, b, n+1)
u_2=RK4_method(f2,u0,a,b,n)
```

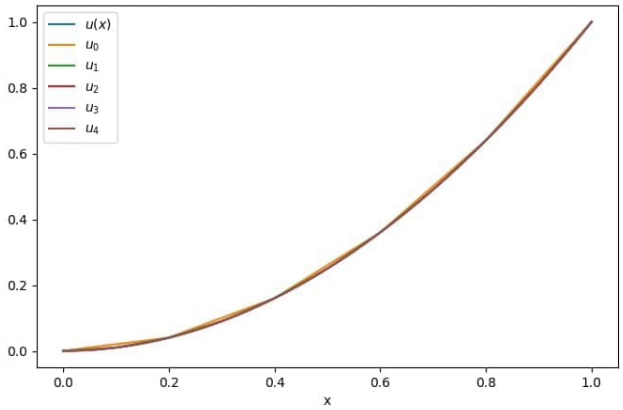
```
n*=2
x3=np.linspace(a, b, n+1)
u_3=RK4_method(f2,u0,a,b,n)

n*=2
x4=np.linspace(a, b, n+1)
u_4=RK4_method(f2,u0,a,b,n)
```

Порахуємо також значення точного розв'язку задачі на рівномірній сітці:

[6]:
```
x=np.linspace(a, b, 256)
ux=x**2
```

Побудуємо графіки отриманих чисельних і точного розв'язків задачі:

[37]:
```
fig = plt.figure(figsize=(8, 5))
plt.plot(x, ux, label='$u(x)$')
plt.plot(x0, u_0, label='$u_0$')
plt.plot(x1, u_1, label='$u_1$')
plt.plot(x2, u_2, label='$u_2$')
plt.plot(x3, u_3, label='$u_3$')
plt.plot(x4, u_4, label='$u_4$')
ax = fig.gca()
ax.legend()
ax.set_xlabel('x');
```



Зауважимо, що графічні побудови за вибраного режиму виконуються з автоматичною лінійною інтертерполяцією чисельних розв'язків, які насправді обчислені лише у вузлах відповідної сітки. Нагадаємо, що отримані зобаження можна масштабувати, використовуючи режим 'Zoom to rectangle'.

Як бачимо, графіки чисельних розв'язків візуально близькі до графіка точного розв'язку. Для детальнішого аналізу занесемо в таблицю значення усіх розв'язків на спільній множині вузлів `x_0`:

[8]:
```
u = x0**2

df=pd.DataFrame({'x_0':x0[1::],'u_0':u_0[1::],'u_1':u_1[2::2],'u_2':u_2[4::
↪4],'u_3':u_3[8::8],'u_4':u_4[16::16],'u':u[1::]})
df
```

[8]:

	x_0	u_0	u_1	u_2	u_3	u_4	u
0	0.2	0.040013	0.040001	0.040000	0.04	0.04	0.04

```
1  0.4  0.160029  0.160002  0.160000  0.16  0.16  0.16
2  0.6  0.360052  0.360003  0.360000  0.36  0.36  0.36
3  0.8  0.640088  0.640006  0.640000  0.64  0.64  0.64
4  1.0  1.000142  1.000009  1.000001  1.00  1.00  1.00
```

На множині вузлів `x_0` маємо такі значення абсолютних похибок отриманих чисельних розв'язків:

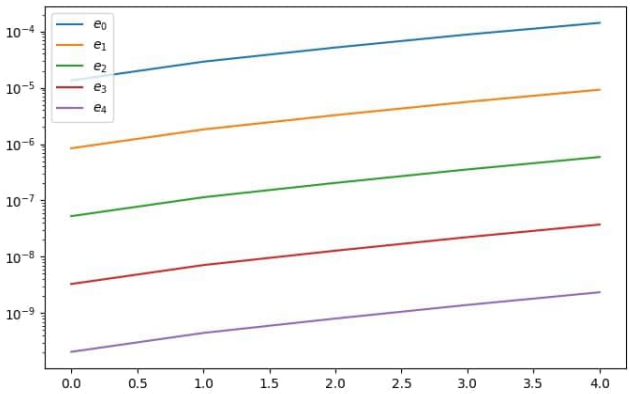
[9]:
```
df1=pd.DataFrame()
df1['e_0']=np.abs(df['u']-df['u_0'])
df1['e_1']=np.abs(df['u']-df['u_1'])
df1['e_2']=np.abs(df['u']-df['u_2'])
df1['e_3']=np.abs(df['u']-df['u_3'])
df1['e_4']=np.abs(df['u']-df['u_4'])
df1
```

[9]:

	e_0	e_1	e_2	e_3	e_4
0	0.000013	8.427528e-07	5.269722e-08	3.294049e-09	2.058864e-10
1	0.000029	1.821414e-06	1.139999e-07	7.128789e-09	4.456444e-10
2	0.000052	3.265084e-06	2.049552e-07	1.283629e-08	8.030726e-10
3	0.000088	5.623451e-06	3.554703e-07	2.234675e-08	1.400790e-09
4	0.000142	9.237892e-06	5.906353e-07	3.735716e-08	2.349089e-09

Побудуємо графіки абсолютних похибок у логарифмічній шкалі:

[40]:
```
fig = plt.figure(figsize=(8, 5))
df1.e_0.plot(logy=True, label = '$e_0$')
df1.e_1.plot(logy=True, label = '$e_1$')
df1.e_2.plot(logy=True, label = '$e_2$')
df1.e_3.plot(logy=True, label = '$e_3$')
df1.e_4.plot(logy=True, label = '$e_4$')
ax = fig.gca()
ax.legend();
```



Як бачимо, при згущенні сітки похибки чисельних розв'язків на кожному кроці зменшуються. Зазначимо, що на фіксованій сітці вони дещо відрізняються між собою, поволі зростаючи у вузлах поблизу правого кінця відрізку.

Обчислимо для кожного чисельного розв'язку його абсолютну похибку за нормою $\|\cdot\|_\infty$

[11]:
```
ne_0 = max(np.abs(df1['e_0']))
ne_1 = max(np.abs(df1['e_1']))
```

```
ne_2 = max(np.abs(df1['e_2']))
ne_3 = max(np.abs(df1['e_3']))
ne_4 = max(np.abs(df1['e_4']))
print(f" ne_0={ne_0:1.2e}\n ne_1={ne_1:1.2e}\n ne_2={ne_2:1.2e}\n␣
↪ne_3={ne_3:1.2e}\n ne_4={ne_4:1.2e}")
```

```
ne_0=1.42e-04
ne_1=9.24e-06
ne_2=5.91e-07
ne_3=3.74e-08
ne_4=2.35e-09
```

Оцінимо швидкість збіжності чисельних розв'язків до точного, а саме у скільки разів зменшуватметься абсолютна похибка при подвоєнні кількості вузлів сітки:

[12]:
```
df2=pd.DataFrame()
df2['r_0'] = df1['e_0'] / df1['e_1']
df2['r_1'] = df1['e_1'] / df1['e_2']
df2['r_2'] = df1['e_2'] / df1['e_3']
df2['r_3'] = df1['e_3'] / df1['e_4']

df2
```

[12]:

	r_0	r_1	r_2	r_3
0	15.979276	15.992358	15.997706	15.999346
1	15.940022	15.977328	15.991484	15.996585
2	15.859714	15.930726	15.966850	15.983975
3	15.671779	15.819746	15.907025	15.952964
4	15.361403	15.640602	15.810499	15.902829

Як бачимо, швидкість наближається до числа 16, що узгоджується з теоретичними оцінками похибок методів Рунге-Кутта.

[13]:
```
plt.close('all')
```

Приклад 2. $f(x,u) = -u\,cosx + e^{-sinx}, \quad [a,b] = [0,\,10], \quad u(0) = 1.$

Відомо [(1.69 Самойленко)] , що функція $u(x) = (x+1)e^{-sinx}$ є точним розв'язком задачі Коші у цьому випадку.

Задамо функцію правої частини диференціального рівняння і решту даних задачі Коші:

[14]:
```
def f3(x, u):
    return -u*np.cos(x) + np.exp(-np.sin(x))
```

[15]:
```
a=0
b=10
u0=1
```

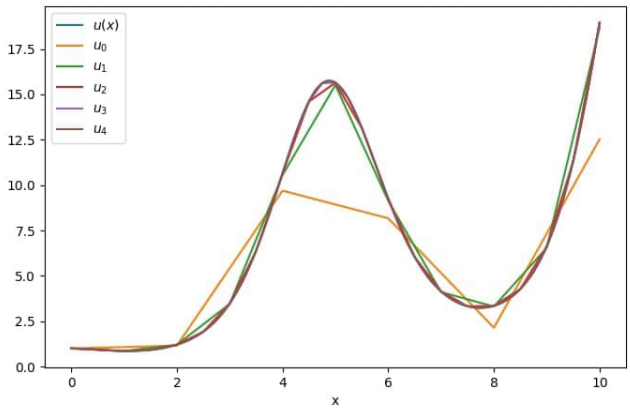
Далі повторимо усі кроки, якими отримали і досліджували чисельні розв'язки попередньої задачі:

[46]:
```
n_start = 5

n = n_start
x0=np.linspace(a, b, n+1)
u_0=RK4_method(f3,u0,a,b,n)

n*=2
x1=np.linspace(a, b, n+1)
u_1=RK4_method(f3,u0,a,b,n)
```

```
n*=2
x2=np.linspace(a, b, n+1)
u_2=RK4_method(f3,u0,a,b,n)

n*=2
x3=np.linspace(a, b, n+1)
u_3=RK4_method(f3,u0,a,b,n)

n*=2
x4=np.linspace(a, b, n+1)
u_4=RK4_method(f3,u0,a,b,n)

x=np.linspace(a, b, 256)
ux=(x+1)*np.exp(-np.sin(x))

fig = plt.figure(figsize=(8, 5))
plt.plot(x, ux, label='$u(x)$')
plt.plot(x0, u_0, label='$u_0$')
plt.plot(x1, u_1, label='$u_1$')
plt.plot(x2, u_2, label='$u_2$')
plt.plot(x3, u_3, label='$u_3$')
plt.plot(x4, u_4, label='$u_4$')
ax = fig.gca()
ax.legend()
ax.set_xlabel('x')

u=(x0+1)*np.exp(-np.sin(x0))

df=pd.DataFrame({'x_0':x0[1::],'u_0':u_0[1::],'u_1':u_1[2::2],'u_2':u_2[4::
 ↪4],'u_3':u_3[8::8],'u_4':u_4[16::16],'u':u[1::]})
df
```

[16]:

	x_0	u_0	u_1	u_2	u_3	u_4	u
0	2.0	1.152763	1.204188	1.208120	1.208405	1.208420	1.208421
1	4.0	9.693075	10.556816	10.649402	10.656734	10.657217	10.657250
2	6.0	8.174354	9.133145	9.250316	9.256140	9.256475	9.256497
3	8.0	2.143113	3.311437	3.344964	3.346271	3.346333	3.346337
4	10.0	12.526120	18.664765	18.936153	18.951184	18.952073	18.952131



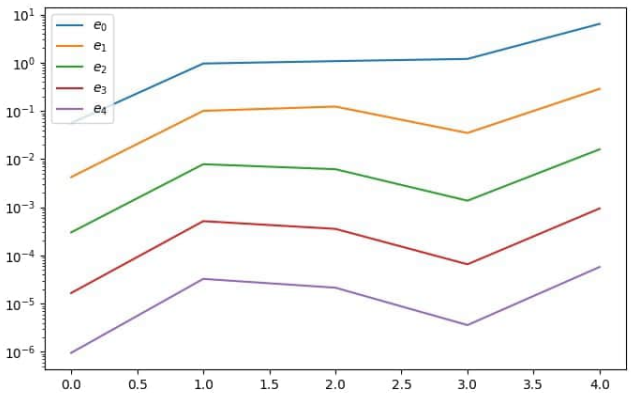
Оскільки розглядаємо задачу на довшому відрізку і, як видно з графіку, поведінка її розв'язку є складнішою, то похибка отриманих чисельних розв'язків є більшою у цьому випадку:

[17]:
```
df1=pd.DataFrame()
df1['e_0']=np.abs(df['u']-df['u_0'])
df1['e_1']=np.abs(df['u']-df['u_1'])
df1['e_2']=np.abs(df['u']-df['u_2'])
df1['e_3']=np.abs(df['u']-df['u_3'])
df1['e_4']=np.abs(df['u']-df['u_4'])
df1
```

[17]:

	e_0	e_1	e_2	e_3	e_4
0	0.055658	0.004234	0.000302	0.000017	9.525676e-07
1	0.964175	0.100434	0.007848	0.000516	3.266958e-05
2	1.082142	0.123352	0.006180	0.000357	2.144942e-05
3	1.203224	0.034900	0.001373	0.000066	3.593800e-06
4	6.426011	0.287366	0.015978	0.000947	5.766687e-05

[48]:
```
fig = plt.figure(figsize=(8, 5))
df1.e_0.plot(logy=True, label='$e_0$')
df1.e_1.plot(logy=True, label='$e_1$')
df1.e_2.plot(logy=True, label='$e_2$')
df1.e_3.plot(logy=True, label='$e_3$')
df1.e_4.plot(logy=True, label='$e_4$')
ax = fig.gca()
ax.legend();
```



Обчислимо для кожного чисельного розв'язку його абсолютну похибку за нормою $\|\cdot\|_\infty$

[20]:
```
ne_0 = max(np.abs(df1['e_0']))
ne_1 = max(np.abs(df1['e_1']))
ne_2 = max(np.abs(df1['e_2']))
ne_3 = max(np.abs(df1['e_3']))
ne_4 = max(np.abs(df1['e_4']))
print(f" ne_0={ne_0:1.2e}\n ne_1={ne_1:1.2e}\n ne_2={ne_2:1.2e}\n␣
 ↪ne_3={ne_3:1.2e}\n ne_4={ne_4:1.2e}")
```

```
ne_0=6.43e+00
ne_1=2.87e-01
ne_2=1.60e-02
```

```
ne_3=9.47e-04
ne_4=5.77e-05
```

Знайдемо тепер значення швидкості збіжності чисельних розв'язків:

[21]:
```
df2=pd.DataFrame()
df2['r_0'] = df1['e_0'] / df1['e_1']
df2['r_1'] = df1['e_1'] / df1['e_2']
df2['r_2'] = df1['e_2'] / df1['e_3']
df2['r_3'] = df1['e_3'] / df1['e_4']

df2
```

[21]:

	r_0	r_1	r_2	r_3
0	13.146823	14.034844	18.138471	17.458484
1	9.600087	12.797077	15.218977	15.784874
2	8.772826	19.958474	17.323763	16.632586
3	34.476730	25.416805	20.917407	18.265754
4	22.361753	17.985473	16.873061	16.420769

Як бачимо, отримані значення наближуються до 16, як і передбачають результати теоретичних досліджень. Якщо повторити усі кроки з обчислення розв'язку, але на сітках з більшою кількістю вузлів (при більшому значенні параметра `n_start`), то можна пересвідчитися у подальшому зменшенні похибок чисельних розв'язків.

[ ]:
```
plt.close('all')
```

6.8.3 Метод Ейлера і Рунге-Кутта

Чисельний розв'язок задачі Коші знаходять **методом Ейлера** за рекурентними формулами

$$\begin{cases} u_{i+1} = u_i + h\, f(x_i, u_i, v_i), \\ v_{i+1} = v_i + h\, g(x_i, u_i, v_i), \quad i = \overline{0, n-1}. \end{cases} \tag{6.8.3}$$

Значення u_0 і v_0 беруть з початкових умов.

Чисельний розв'язок задачі Коші знаходять **методом Рунге-Кутта** за рекурентними формулами

$$\begin{cases} u_{i+1} = u_i + \frac{1}{6}k_{i,1}(h) + \frac{1}{3}k_{i,2}(h) + \frac{1}{3}k_{i,3}(h) + \frac{1}{6}k_{i,4}(h), \\ v_{i+1} = v_i + \frac{1}{6}m_{i,1}(h) + \frac{1}{3}m_{i,2}(h) + \frac{1}{3}m_{i,3}(h) + \frac{1}{6}m_{i,4}(h), \quad i = \overline{0, n-1}, \end{cases} \tag{6.8.4}$$

де

$$k_{i,1}(h) := f(x_i,\ u_i,\ v_i) \cdot h, \quad m_{i,1}(h) := g(x_i,\ u_i,\ v_i) \cdot h,$$

$$k_{i,2}(h) := f\left(x_i + \frac{1}{2}h,\ u_i + \frac{1}{2}k_{i,1}(h),\ v_i + \frac{1}{2}m_{i,1}(h)\right) \cdot h,$$

$$m_{i,2}(h) := g\left(x_i + \frac{1}{2}h,\ u_i + \frac{1}{2}k_{i,1}(h),\ v_i + \frac{1}{2}m_{i,1}(h)\right) \cdot h,$$

$$k_{i,3}(h) := f(x_i + \frac{1}{2}h,\ u_i + \frac{1}{2}k_{i,2}(h),\ v_i + \frac{1}{2}m_{i,2}(h)) \cdot h,$$

$$m_{i,3}(h) := g(x_i + \frac{1}{2}h,\ u_i + \frac{1}{2}k_{i,2}(h),\ v_i + \frac{1}{2}m_{i,2}(h)) \cdot h,$$

$$k_{i,4}(h) := f(x_i + h,\ u_i + k_{i,3}(h),\ v_i + m_{i,3}(h)) \cdot h,$$

$$m_{i,4}(h) := g(x_i + h,\ u_i + k_{i,3}(h),\ v_i + m_{i,3}(h)) \cdot h.$$

Значення u_0 і v_0 беруть з початкових умов.

Пояснення до використання програмного коду

- Підготувати середовище і потрібні функції :
  1. виконати комірку для підготовки середовища
  2. виконати комірку з функцією `Euler_NS` чи `RK4_NS`
- Обчислити чисельний розв'язок конкретної задачі Коші
  1. виконати комірку, де **визначені** функції `f` і `g`
  2. виконати комірку з викликом функції `Euler_NS` чи `RK4__NS`, перед виконанням задати відповідні аргументи цієї функції.
- Щоб переконатися, що чисельний розв'язок достатньо точний, можна виконати кілька послідовних викликів функції `Euler_NS` чи `RK4__NS`, збільшуючи кількість вузлів сітки.

Програмна реалізація методів

Підготовка середовища

[1]:
```
%matplotlib widget
import matplotlib.pyplot as plt

import numpy as np
```

```
import pandas as pd
```

`Euler_NS` – **функція, яка реалізує явний метод Ейлера для знаходження чисельного розв'язку задачі Коші для системи двох ЗДР 1-го порядку**

[2]:
```
def Euler_NS(f,g,u0,v0,a,b,n, alpha, beta, gamma, delta):
    """ явний метод Ейлера для задачі Коші
        для системи двох ЗДР 1-го порядку
    """
    h=(b-a)/n
    x=np.linspace(a, b, n+1)

    u=np.empty(n+1)
    v=np.empty(n+1)
    u[0]=u0
    v[0]=v0

    for i in range(n):
        u[i+1]=u[i] + h*f(u[i], v[i], alpha, beta)
        v[i+1]=v[i] + h*g(u[i], v[i], gamma, delta)

    return u, v
```

`RK4_NS` – **функція, яка реалізує метод Рунге-Кутта для знаходження чисельного розв'язку задачі Коші для системи двох ЗДР 1-го порядку**

[3]:
```
def RK4_NS(f,g,u0,v0,a,b,n, alpha, beta, gamma, delta):
    """ метод Рунге-Кутта четвертого порядку
        для задачі Коші для системи двох ЗДР 1-го порядку
    """
    h=(b-a)/n
    x=np.linspace(a, b, n+1)

    u=np.empty(n+1)
    v=np.empty(n+1)
    u[0]=u0
    v[0]=v0

    for i in range(n):
        k1 = f(u[i], v[i], alpha, beta)
        m1 = g(u[i], v[i], gamma, delta)
        k2 = f(u[i] + h/2*k1, v[i] + h/2*m1, alpha, beta)
        m2 = g(u[i] + h/2*k1, v[i] + h/2*m1, gamma, delta)
        k3 = f(u[i] + h/2*k2, v[i] + h/2*m2, alpha, beta)
        m3 = g(u[i] + h/2*k2, v[i] + h/2*m2, gamma, delta)
        k4 = f(u[i] + h*k3, v[i] + h*m3, alpha, beta)
        m4 = g(u[i] + h*k3, v[i] + h*m3, gamma, delta)

        u[i+1]=u[i] + h/6*(k1 + 2*k2 + 2*k3 + k4)
        v[i+1]=v[i] + h/6*(m1 + 2*m2 + 2*m3 + m4)

    return u, v
```

`f, g` – **функції, які задають обчислення правих частин рівнянь конкретної системи**

Обчислювальні експерименти
Продемонструємо застосування методів Ейлера та Рунге-Кутта для знаходження чисельного розв'язку задачі Коші для системи двох ЗДР 1-го порядку.

Приклад 1. За допомогою методів Ейлера та Рунге-Кутта знайти чисельні розв'язки задачі Коші (6.1.1),(6.1.2), коли

$f(x,u,v) = \alpha u + \beta uv, \quad g(x,u,v) = \gamma v + \delta uv, \quad x \in [a,b],$

$u(0) = u0, \quad v(0) = v0, \quad \alpha, \beta, \gamma, \delta, u0, v0$– задані числа.

Система рівнянь (6.1.1) відома під назвою рівнянь Лотки-Волmтерра і описує динаміку популяції двох видів – жертви і хижака, детальніше див., наприклад, розділ 2.1 [Ортега].

Визначимо функції правих частин рівняннь (6.1.1):

[4]:
```
def f(u, v, alpha, beta ):
    return alpha*u + beta*u*v
```

[5]:
```
def g(u, v, gamma, delta ):
    return gamma*v + delta*u*v
```

Задамо значення параметрів задачі і початкові дані:

[6]:
```
a=0
b=12
u0=80
v0=30
alpha = 0.25
beta = - 0.01
gamma = -1
delta = 0.01
```

1) Застосування методу Ейлера

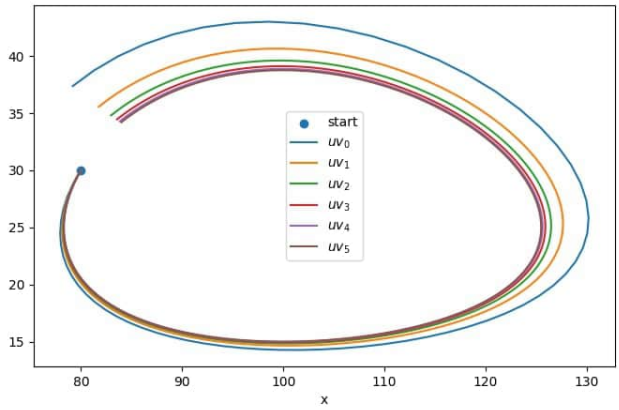
Знайдемо методом Ейлера чисельні розв'язки задачі на послідовності сіток, подвоюючи на кожному кроці кількість вузлів і зберігаючи відповідні масиви для подальшого аналізу:

[7]:
```
n_start = 64

n = n_start
x0=np.linspace(a, b, n+1)
u_0, v_0 = Euler_NS(f, g, u0,v0, a,b,n, alpha, beta, gamma, delta)

n *= 2
x1=np.linspace(a, b, n+1)
u_1, v_1 = Euler_NS(f, g, u0,v0, a,b,n, alpha, beta, gamma, delta)

n *= 2
x2=np.linspace(a, b, n+1)
u_2, v_2 = Euler_NS(f, g, u0,v0, a,b,n, alpha, beta, gamma, delta)

n *= 2
x3=np.linspace(a, b, n+1)
u_3, v_3 = Euler_NS(f, g, u0,v0, a,b,n, alpha, beta, gamma, delta)

n *= 2
x4=np.linspace(a, b, n+1)
u_4, v_4 = Euler_NS(f, g, u0,v0, a,b,n, alpha, beta, gamma, delta)
```

```
n *= 2
x5=np.linspace(a, b, n+1)
u_5, v_5 = Euler_NS(f, g, u0,v0, a,b,n, alpha, beta, gamma, delta)
```

Побудуємо графіки отриманих чисельних розв'язків задачі:

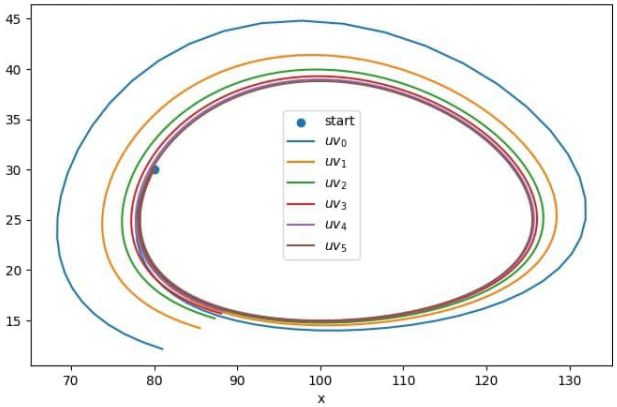
[48]:
```
fig = plt.figure(figsize=(8, 5))
plt.scatter(u0,v0, marker='o', label='start')
plt.plot( u_0,v_0, label='$uv_0$')
plt.plot( u_1,v_1, label='$uv_1$')
plt.plot( u_2,v_2, label='$uv_2$')
plt.plot( u_3,v_3, label='$uv_3$')
plt.plot( u_4,v_4, label='$uv_4$')
plt.plot( u_5,v_5, label='$uv_5$')
ax = fig.gca()
ax.legend()
ax.set_xlabel('x');
```



Як бачимо, усі графіки починаються в точці $start(u0, v0)$. При збільшенні значення параметра b вони матимуть вигляд еліпсоподібних спіралей:

[49]:
```
b=16

n_start = 64

n = n_start
x0=np.linspace(a, b, n+1)
u_0, v_0 = Euler_NS(f, g, u0,v0, a,b,n, alpha, beta, gamma, delta)

n *= 2
x1=np.linspace(a, b, n+1)
u_1, v_1 = Euler_NS(f, g, u0,v0, a,b,n, alpha, beta, gamma, delta)

n *= 2
x2=np.linspace(a, b, n+1)
u_2, v_2 = Euler_NS(f, g, u0,v0, a,b,n, alpha, beta, gamma, delta)

n *= 2
x3=np.linspace(a, b, n+1)
u_3, v_3 = Euler_NS(f, g, u0,v0, a,b,n, alpha, beta, gamma, delta)
```

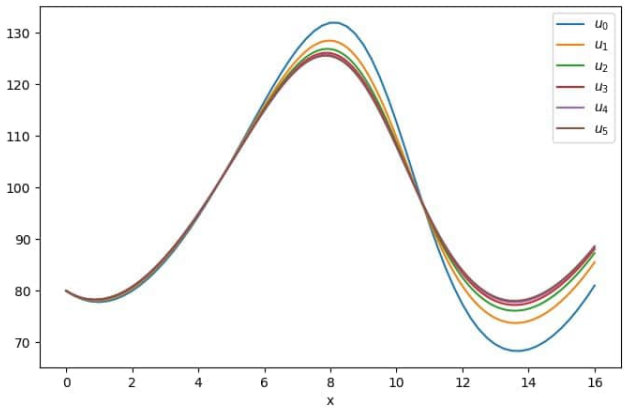
```
n *= 2
x4=np.linspace(a, b, n+1)
u_4, v_4 = Euler_NS(f, g, u0,v0, a,b,n, alpha, beta, gamma, delta)

n *= 2
x5=np.linspace(a, b, n+1)
u_5, v_5 = Euler_NS(f, g, u0,v0, a,b,n, alpha, beta, gamma, delta)


fig = plt.figure(figsize=(8, 5))
plt.scatter(u0,v0, marker='o', label='start')
plt.plot( u_0,v_0, label='$uv_0$')
plt.plot( u_1,v_1, label='$uv_1$')
plt.plot( u_2,v_2, label='$uv_2$')
plt.plot( u_3,v_3, label='$uv_3$')
plt.plot( u_4,v_4, label='$uv_4$')
plt.plot( u_5,v_5, label='$uv_5$')
ax = fig.gca()
ax.legend()
ax.set_xlabel('x');
```



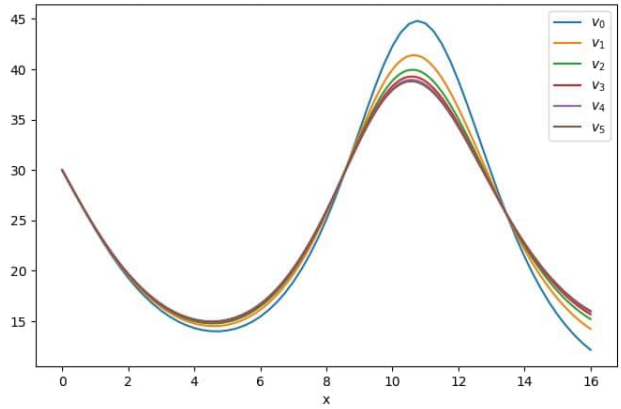
Зауважимо, що при збільшенні вузлів сітки графіки чисельних розв'язків наближаються до замкнутої еліпсоподібної фігури, яка характеризує популяції взаємопов'язаних видів при заданих значеннях параметрів моделі.

Побудуємо графіки чисельних розв'язків як функцій змінної x:

[50]:
```
fig = plt.figure(figsize=(8, 5))

plt.plot(x0, u_0, label='$u_0$')
plt.plot(x1, u_1, label='$u_1$')
plt.plot(x2, u_2, label='$u_2$')
plt.plot(x3, u_3, label='$u_3$')
plt.plot(x4, u_4, label='$u_4$')
plt.plot(x5, u_5, label='$u_5$')
ax = fig.gca()
ax.legend()
ax.set_xlabel('x');
```



[51]:
```
fig = plt.figure(figsize=(8, 5))

plt.plot(x0, v_0, label='$v_0$')
plt.plot(x1, v_1, label='$v_1$')
plt.plot(x2, v_2, label='$v_2$')
plt.plot(x3, v_3, label='$v_3$')
plt.plot(x4, v_4, label='$v_4$')
plt.plot(x5, v_5, label='$v_5$')
ax = fig.gca()
ax.legend()
ax.set_xlabel('x');
```



Співставляючи оримані графіки бачимо, що зростання популяції жертв (функція `u`) приводить з часом до зростання кількості хижаків (функція `v`), які за деякий час знову зменшать популяцію жертв, що в свою чергу негативно вплине на кількість хижаків.

Для подальшого аналізу занесемо в таблиці значення усіх розв'язків на спільній множині вузлів `x_0`. Далі при виведенні таблиць будемо виводити лише 5 перших рядків (щоб отримати усі рядки таблиці треба забрати виклик методу `head(5)`):

[12]:
```
dfue=pd.DataFrame({'x_0':x0[1::],'u_0':u_0[1::],'u_1':u_1[2::2],'u_2':
 ↪u_2[4::4],'u_3':u_3[8::8],'u_4':u_4[16::16],'u_5':u_5[32::32]})
```

```
dfue.head(5)
```

[12]:

	x_0	u_0	u_1	u_2	u_3	u_4	u_5
0	0.25	79.000000	79.077656	79.116139	79.135249	79.144765	79.149513
1	0.50	78.308750	78.460503	78.535028	78.571880	78.590195	78.599324
2	0.75	77.916472	78.136485	78.243684	78.296499	78.322702	78.335751
3	1.00	77.811404	78.092334	78.228254	78.294999	78.328060	78.344511
4	1.25	77.980763	78.314338	78.474686	78.553184	78.592008	78.611313

[13]:

```
dfve=pd.DataFrame({'x_0':x0[1::],'v_0':v_0[1::],'v_1':v_1[2::2],'v_2':
 ↪v_2[4::4],'v_3':v_3[8::8],'v_4':v_4[16::16],'v_5':v_5[32::32]})
dfve.head(5)
```

[13]:

	x_0	v_0	v_1	v_2	v_3	v_4	v_5
0	0.25	28.500000	28.500469	28.502218	28.503450	28.504152	28.504524
1	0.50	27.003750	27.017216	27.026498	27.031729	27.034485	27.035898
2	0.75	25.539387	25.575457	25.596594	25.607866	25.613670	25.616613
3	1.00	24.129388	24.194888	24.230869	24.249579	24.259104	24.263907
4	1.25	22.790895	22.890210	22.942890	22.969886	22.983536	22.990398

2) Застосування методу Рунге-Кутта

Розглянемо чисельне розв'язування тієї самої задачі Коші, але тепер застосуємо метод Рунге-Кутта. Після задання тих самих значень параметрів задачі і початкових даних знайдемо чисельні розв'язки задачі на послідовності сіток, подвоюючи на кожному кроці кількість вузлів:

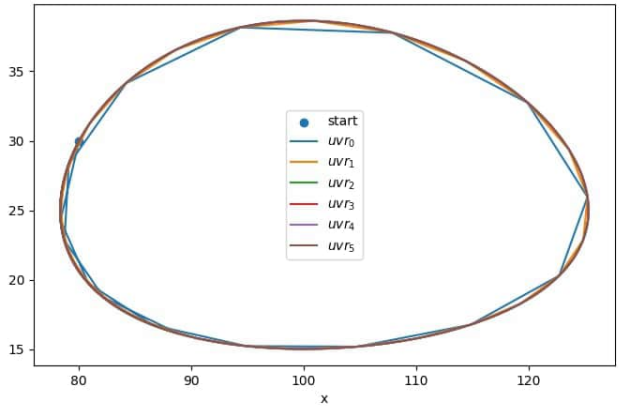
[14]:

```
a=0
b=16
u0=80
v0=30
alpha = 0.25
beta = - 0.01
gamma = -1
delta = 0.01

n_start = 16

n = n_start
xr0=np.linspace(a, b, n+1)
ur_0, vr_0 = RK4_NS(f, g, u0,v0, a,b,n, alpha, beta, gamma, delta)

n *= 2
x1=np.linspace(a, b, n+1)
ur_1, vr_1 = RK4_NS(f, g, u0,v0, a,b,n, alpha, beta, gamma, delta)

n *= 2
x2=np.linspace(a, b, n+1)
ur_2, vr_2 = RK4_NS(f, g, u0,v0, a,b,n, alpha, beta, gamma, delta)

n *= 2
x3=np.linspace(a, b, n+1)
ur_3, vr_3 = RK4_NS(f, g, u0,v0, a,b,n, alpha, beta, gamma, delta)

n *= 2
x4=np.linspace(a, b, n+1)
ur_4, vr_4 = RK4_NS(f, g, u0,v0, a,b,n, alpha, beta, gamma, delta)

n *= 2
```

```
x5=np.linspace(a, b, n+1)
ur_5, vr_5 = RK4_NS(f, g, u0,v0, a,b,n, alpha, beta, gamma, delta)
```

Побудуємо графіки отриманих розв'язків:

[55]:
```
fig = plt.figure(figsize=(8, 5))

plt.scatter(u0,v0, marker='o', label='start')
plt.plot( ur_0,vr_0, label='$uvr_0$')
plt.plot( ur_1,vr_1, label='$uvr_1$')
plt.plot( ur_2,vr_2, label='$uvr_2$')
plt.plot( ur_3,vr_3, label='$uvr_3$')
plt.plot( ur_4,vr_4, label='$uvr_4$')
plt.plot( ur_5,vr_5, label='$uvr_5$')
ax = fig.gca()
ax.legend()
ax.set_xlabel('x');
```



Зауважимо, що як і слід було очікувати, метод Рунге-Кутта демонструє швидшу збіжність, ніж метод Ейлера. Тому графіки отриманих розв'язків починають візуально співпадати уже на сітках з невеликою кількістю вузлів. Для того, щоб їх розрізнити, можна використати режим масштабування.

Для подальшого аналізу занесемо в таблиці значення усіх розв'язків на спільній множині вузлів x_0:

[16]:
```
dfur=pd.DataFrame({'x_0':xr0[1::],'ur_0':ur_0[1::],'ur_1':ur_1[2::
↪2],'ur_2':ur_2[4::4],'ur_3':ur_3[8::8],'ur_4':ur_4[16::16],'ur_5':
↪ur_5[32::32]})
dfur.head(5)
```

[16]:
```
   x_0        ur_0        ur_1        ur_2        ur_3        ur_4        ␣
↪ur_5
0  1.0   78.357869   78.360676   78.360894   78.360909   78.360910   78.
↪360910
1  2.0   80.840126   80.847055   80.847536   80.847567   80.847569   80.
↪847569
2  3.0   86.611094   86.621690   86.622387   86.622431   86.622434   86.
↪622434
```

```
3  4.0   94.902810   94.916288   94.917139   94.917192   94.917195   94.
↪917196
4  5.0  104.794859  104.810173  104.811099  104.811156  104.811159  104.
↪811159
```

[17]:
```
dfvr=pd.DataFrame({'x_0':xr0[1::],'vr_0':vr_0[1::],'vr_1':vr_1[2::
↪2],'vr_2':vr_2[4::4],'vr_3':vr_3[8::8],'vr_4':vr_4[16::16],'vr_5':
↪vr_5[32::32]})
dfvr.head(5)
```

[17]:

	x_0	vr_0	vr_1	vr_2	vr_3	vr_4	vr_5
0	1.0	24.269596	24.268784	24.268739	24.268737	24.268737	24.268737
1	2.0	19.731535	19.730446	19.730403	19.730401	19.730401	19.730401
2	3.0	16.728963	16.728113	16.728100	16.728100	16.728100	16.728100
3	4.0	15.226495	15.226261	15.226296	15.226300	15.226300	15.226300
4	5.0	15.191791	15.192664	15.192778	15.192787	15.192787	15.192787

Як бачимо, в останній та передостанній колонках обох таблиць відповідні значення можуть відрізнятися лише останньою цифрою. Оскільки точний розв'язок задачі Коші невідомий, то далі обчислюватимемо значення абсолютних похибок чисельних розв'язків та швидкості збіжності до `ur_5` і `vr_5` відповідно на спільній множині вузлів `x_0`:

[18]:
```
dfur1=pd.DataFrame()
dfur1['eur_0']=np.abs(dfur['ur_5']-dfur['ur_0'])
dfur1['eur_1']=np.abs(dfur['ur_5']-dfur['ur_1'])
dfur1['eur_2']=np.abs(dfur['ur_5']-dfur['ur_2'])
dfur1['eur_3']=np.abs(dfur['ur_5']-dfur['ur_3'])

dfur1.head(5)
```

[18]:

	eur_0	eur_1	eur_2	eur_3
0	0.003041	0.000234	0.000016	0.000001
1	0.007444	0.000514	0.000033	0.000002
2	0.011340	0.000744	0.000047	0.000003
3	0.014386	0.000907	0.000056	0.000004
4	0.016301	0.000986	0.000060	0.000004

[19]:
```
dfvr1=pd.DataFrame()
dfvr1['evr_0']=np.abs(dfvr['vr_5']-dfvr['vr_0'])
dfvr1['evr_1']=np.abs(dfvr['vr_5']-dfvr['vr_1'])
dfvr1['evr_2']=np.abs(dfvr['vr_5']-dfvr['vr_2'])
dfvr1['evr_3']=np.abs(dfvr['vr_5']-dfvr['vr_3'])

dfvr1.head(5)
```

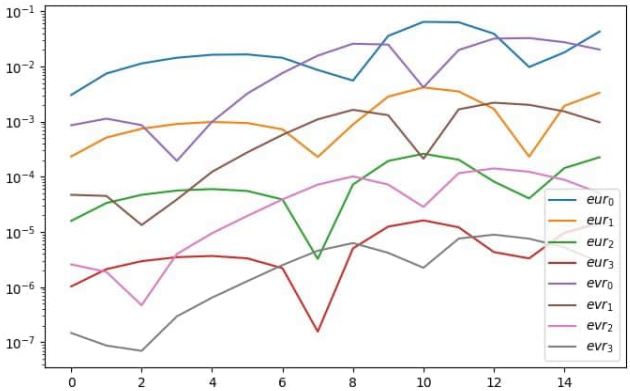
[19]:

	evr_0	evr_1	evr_2	evr_3
0	0.000860	0.000047	2.589884e-06	1.479729e-07
1	0.001134	0.000045	1.889312e-06	8.750560e-08
2	0.000863	0.000013	4.721431e-07	7.016251e-08
3	0.000195	0.000039	3.984675e-06	2.955270e-07
4	0.000996	0.000123	9.508043e-06	6.462838e-07

Побудуємо графіки абсолютних похибок у логарифмічній шкалі:

[60]:
```
fig = plt.figure(figsize=(8, 5))

dfur1.eur_0.plot(logy=True, label='$eur_0$')
dfur1.eur_1.plot(logy=True, label='$eur_1$')
```

```
dfur1.eur_2.plot(logy=True, label='$eur_2$')
dfur1.eur_3.plot(logy=True, label='$eur_3$')

dfvr1.evr_0.plot(logy=True, label='$evr_0$')
dfvr1.evr_1.plot(logy=True, label='$evr_1$')
dfvr1.evr_2.plot(logy=True, label='$evr_2$')
dfvr1.evr_3.plot(logy=True, label='$evr_3$')

ax = fig.gca()
ax.legend();
```



Обчислимо для кожного чисельного розв'язку його абсолютну похибку за нормою $\|\cdot\|_\infty$

[21]:
```
neur_0 = max(np.abs(dfur1['eur_0']))
neur_1 = max(np.abs(dfur1['eur_1']))
neur_2 = max(np.abs(dfur1['eur_2']))
neur_3 = max(np.abs(dfur1['eur_3']))

nevr_0 = max(np.abs(dfvr1['evr_0']))
nevr_1 = max(np.abs(dfvr1['evr_1']))
nevr_2 = max(np.abs(dfvr1['evr_2']))
nevr_3 = max(np.abs(dfvr1['evr_3']))

print(f" neur_0={neur_0:1.2e},  nevr_0={nevr_0:1.2e}")
print(f" neur_1={neur_1:1.2e},  nevr_1={nevr_1:1.2e}")
print(f" neur_2={neur_2:1.2e},  nevr_2={nevr_2:1.2e}")
print(f" neur_3={neur_3:1.2e},  nevr_3={nevr_3:1.2e}")
```

```
 neur_0=6.43e-02,  nevr_0=3.27e-02
 neur_1=4.17e-03,  nevr_1=2.20e-03
 neur_2=2.62e-04,  nevr_2=1.42e-04
 neur_3=1.62e-05,  nevr_3=8.94e-06
```

Оцінимо швидкість збіжності чисельних розв'язків до значень `ur_5` і `vr_5`, а саме у скільки разів зменшуватметься абсолютна похибка при подвоєнні кількості вузлів сітки:

[22]:
```
dfur2=pd.DataFrame()
dfur2['rur_0'] = dfur1['eur_0'] / dfur1['eur_1']
dfur2['rur_1'] = dfur1['eur_1'] / dfur1['eur_2']
dfur2['rur_2'] = dfur1['eur_2'] / dfur1['eur_3']
```

```
dfur2
```

[22]:

	rur_0	rur_1	rur_2
0	13.017090	14.682301	15.450107
1	14.470429	15.370804	15.780677
2	15.240025	15.754353	15.969734
3	15.856276	16.071234	16.129092
4	16.524370	16.433398	16.317183
5	17.516002	17.024969	16.641528
6	19.799359	18.640998	17.615547
7	37.760466	70.247195	20.826680
8	6.241677	12.232101	14.383725
9	12.586289	14.670839	15.472717
10	15.416978	15.957177	16.093198
11	17.917106	17.277577	16.781742
12	22.982935	20.824289	18.969776
13	42.054732	5.721357	12.196141
14	9.335953	13.402174	14.902320
15	12.823169	14.745786	15.504031

[23]:

```
dfvr2=pd.DataFrame()
dfvr2['rvr_0'] = dfvr1['evr_0'] / dfvr1['evr_1']
dfvr2['rvr_1'] = dfvr1['evr_1'] / dfvr1['evr_2']
dfvr2['rvr_2'] = dfvr1['evr_2'] / dfvr1['evr_3']

dfvr2
```

[23]:

	rvr_0	rvr_1	rvr_2
0	18.212808	18.222573	17.502414
1	25.164813	23.848759	21.590755
2	64.385699	28.391763	6.729280
3	5.019630	9.732501	13.483285
4	8.088470	12.953927	14.711869
5	11.634806	14.178397	15.227392
6	13.191806	14.783989	15.494060
7	14.270031	15.284982	15.736454
8	15.763894	16.081268	16.148067
9	19.056782	18.087787	17.265084
10	19.698983	7.465637	12.671119
11	11.874208	14.346201	15.317090
12	14.511231	15.522094	15.879978
13	16.193285	16.354420	16.298228
14	17.955295	17.317782	16.807865
15	20.802629	19.142593	17.866246

Як бачимо, отримані значення швидкості збіжності близькі до числа 16, характерного для методу Рунге-Кутта 4-го порядку.

3)Порівняння результатів, отриманих методами Ейлера і Рунге-Кутта

Вважаючи чисельні розв'язки, отримані методом Рунге-Кутта, достатньо точними, їх можна використати для оцінки результатів, які дає метод Ейлера.

Для зручності обчислень, додамо до таблиць результатів методу Ейлера відповідні значення розв'язків `ur_5` і `vr_5`:

[24]:

```
dfue['ur_5']=ur_5[8::8]
dfue.head(5)
```

[24]:
```
    x_0        u_0        u_1        u_2        u_3        u_4        u_5 ␣
↪\
0  0.25  79.000000  79.077656  79.116139  79.135249  79.144765  79.149513
1  0.50  78.308750  78.460503  78.535028  78.571880  78.590195  78.599324
2  0.75  77.916472  78.136485  78.243684  78.296499  78.322702  78.335751
3  1.00  77.811404  78.092334  78.228254  78.294999  78.328060  78.344511
4  1.25  77.980763  78.314338  78.474686  78.553184  78.592008  78.611313

        ur_5
0  79.154254
1  78.608434
2  78.348765
3  78.360910
4  78.630546
```

[25]:
```
dfve['vr_5']=vr_5[8::8]
dfve.head(5)
```

[25]:
```
    x_0        v_0        v_1        v_2        v_3        v_4        v_5 ␣
↪\
0  0.25  28.500000  28.500469  28.502218  28.503450  28.504152  28.504524
1  0.50  27.003750  27.017216  27.026498  27.031729  27.034485  27.035898
2  0.75  25.539387  25.575457  25.596594  25.607866  25.613670  25.616613
3  1.00  24.129388  24.194888  24.230869  24.249579  24.259104  24.263907
4  1.25  22.790895  22.890210  22.942890  22.969886  22.983536  22.990398

        vr_5
0  28.504910
1  27.037333
2  25.619582
3  24.268737
4  22.997284
```

Далі виконаємо обчислення відхилень колонок модифікованих таблиць від `ur_5` і `vr_5` відповідно:

[26]:
```
dfue1=pd.DataFrame()
dfue1['eue_0']=np.abs(dfue['ur_5']-dfue['u_0'])
dfue1['eue_1']=np.abs(dfue['ur_5']-dfue['u_1'])
dfue1['eue_2']=np.abs(dfue['ur_5']-dfue['u_2'])
dfue1['eue_3']=np.abs(dfue['ur_5']-dfue['u_3'])
dfue1['eue_4']=np.abs(dfue['ur_5']-dfue['u_4'])
dfue1['eue_5']=np.abs(dfue['ur_5']-dfue['u_5'])

dfue1.head(5)
```

[26]:
```
      eue_0     eue_1     eue_2     eue_3     eue_4     eue_5
0  0.154254  0.076598  0.038115  0.019006  0.009489  0.004741
1  0.299684  0.147931  0.073406  0.036554  0.018239  0.009110
2  0.432292  0.212279  0.105081  0.052266  0.026063  0.013014
3  0.549506  0.268576  0.132656  0.065911  0.032850  0.016398
4  0.649783  0.316208  0.155860  0.077362  0.038538  0.019233
```

[27]:
```
dfve1=pd.DataFrame()
dfve1['eve_0']=np.abs(dfve['vr_5']-dfve['v_0'])
dfve1['eve_1']=np.abs(dfve['vr_5']-dfve['v_1'])
dfve1['eve_2']=np.abs(dfve['vr_5']-dfve['v_2'])
dfve1['eve_3']=np.abs(dfve['vr_5']-dfve['v_3'])
dfve1['eve_4']=np.abs(dfve['vr_5']-dfve['v_4'])
```

```
dfve1['eve_5']=np.abs(dfve['vr_5']-dfve['v_5'])

dfve1.head(5)
```

[27]:

	eve_0	eve_1	eve_2	eve_3	eve_4	eve_5
0	0.004910	0.004441	0.002692	0.001460	0.000758	0.000386
1	0.033583	0.020117	0.010835	0.005605	0.002848	0.001436
2	0.080195	0.044125	0.022989	0.011716	0.005912	0.002969
3	0.139349	0.073848	0.037868	0.019157	0.009633	0.004830
4	0.206389	0.107073	0.054394	0.027398	0.013747	0.006886

Можемо оцінити отримані відхилення за нормою $\|\cdot\|_\infty$:

[28]:
```
neue_0 = max(np.abs(dfue1['eue_0']))
neue_1 = max(np.abs(dfue1['eue_1']))
neue_2 = max(np.abs(dfue1['eue_2']))
neue_3 = max(np.abs(dfue1['eue_3']))
neue_4 = max(np.abs(dfue1['eue_4']))
neue_5 = max(np.abs(dfue1['eue_5']))

neve_0 = max(np.abs(dfve1['eve_0']))
neve_1 = max(np.abs(dfve1['eve_1']))
neve_2 = max(np.abs(dfve1['eve_2']))
neve_3 = max(np.abs(dfve1['eve_3']))
neve_4 = max(np.abs(dfve1['eve_4']))
neve_5 = max(np.abs(dfve1['eve_5']))

print(f" neue_0={neue_0:1.2e},  neve_0={neve_0:1.2e}")
print(f" neue_1={neue_1:1.2e},  neve_1={neve_1:1.2e}")
print(f" neue_2={neue_2:1.2e},  neve_2={neve_2:1.2e}")
print(f" neue_3={neue_3:1.2e},  neve_3={neve_3:1.2e}")
print(f" neue_4={neue_4:1.2e},  neve_4={neve_4:1.2e}")
print(f" neue_5={neue_5:1.2e},  neve_5={neve_5:1.2e}")
```

```
 neue_0=1.01e+01,  neve_0=6.39e+00
 neue_1=4.58e+00,  neve_1=2.83e+00
 neue_2=2.18e+00,  neve_2=1.33e+00
 neue_3=1.07e+00,  neve_3=6.48e-01
 neue_4=5.27e-01,  neve_4=3.20e-01
 neue_5=2.62e-01,  neve_5=1.59e-01
```

Як бачимо, відхилення для обох невідомих функцій приблизно однакові.

Оцінимо швидкість збіжності методу Ейлера:

[29]:
```
dfue2=pd.DataFrame()
dfue2['rue_0'] = dfue1['eue_0'] / dfue1['eue_1']
dfue2['rue_1'] = dfue1['eue_1'] / dfue1['eue_2']
dfue2['rue_2'] = dfue1['eue_2'] / dfue1['eue_3']
dfue2['rue_3'] = dfue1['eue_3'] / dfue1['eue_4']
dfue2['rue_4'] = dfue1['eue_4'] / dfue1['eue_5']

dfue2.head(5)
```

[29]:

	rue_0	rue_1	rue_2	rue_3	rue_4
0	2.013814	2.009658	2.005468	2.002887	2.001481
1	2.025842	2.015235	2.008148	2.004200	2.002131
2	2.036432	2.020150	2.010512	2.005359	2.002705
3	2.045998	2.024608	2.012662	2.006415	2.003228

```
4  2.054920  2.028797  2.014691  2.007414  2.003723
```

[30]:
```
dfve2=pd.DataFrame()
dfve2['rve_0'] = dfve1['eve_0'] / dfve1['eve_1']
dfve2['rve_1'] = dfve1['eve_1'] / dfve1['eve_2']
dfve2['rve_2'] = dfve1['eve_2'] / dfve1['eve_3']
dfve2['rve_3'] = dfve1['eve_3'] / dfve1['eve_4']
dfve2['rve_3'] = dfve1['eve_4'] / dfve1['eve_5']

dfve2.head(5)
```

[30]:

	rve_0	rve_1	rve_2	rve_3
0	1.105544	1.649991	1.843320	1.963804
1	1.669404	1.856640	1.933195	1.984154
2	1.817445	1.919421	1.962182	1.990988
3	1.886957	1.950179	1.976657	1.994447
4	1.927546	1.968474	1.985341	1.996535

Як і в попередньому випадку, отримані значення узгоджуються з теоретичними результатами. Як і очікувалося, значення швидкості близькі до числа 2.

6.8.4 Розв'язування крайових задач для ЗДР методом скінчених різниць (сіток)

Розглянемо крайову задачу

$$u'' + q(x)u = f(x), \quad x \in (0, l), \tag{6.8.5}$$

$$\alpha_0 u'(0) + \beta_0 u(0) = \gamma_0, \tag{6.8.6}$$

$$\alpha_1 u'(l) + \beta_1 u(l) = \gamma_1, \tag{6.8.7}$$

де $q, f \in C([0, l])$, а $\alpha_0, \beta_0, \gamma_0, \alpha_1, \beta_1, \gamma_1$ – сталі, причому
$|\alpha_0| + |\beta_0| > 0, \quad |\alpha_1| + |\beta_1| > 0.$

Нехай ця задача має і тільки один розв'язок $C^2[0, l]$.

Побудуємо сітку з $N + 1,\ N \in \mathbb{N}$, точок $x_i = ih, \quad i = \overline{0, N}$, де $h := \dfrac{l}{N}$.

У методі скінчених різниць чисельним розв'язком крайової задачі (6.8.5)-(6.8.7) є наближення сіткових значень u_i, $i = \overline{0, N}$, розв'язку u у вузлах x_i, $i = \overline{0, N}$.

Після заміни похідних першого і другого порядку від функції u у вузлах сітки різницевими співвідношеннями матимемо СЛАР

$$\sum_{i=0}^{N} c_{ij} u_i = d_j, \quad j = \overline{0, N}, \tag{6.8.8}$$

з квадратною тридіагональною матрицею

$$C = \begin{pmatrix} -\alpha_0 + \beta_0 h & \alpha_0 & 0 & \dots & 0 & 0 & 0 \\ 1 & h^2 q_1 - 2 & 1 & \dots & 0 & 0 & 0 \\ 0 & 1 & h^2 q_2 - 2 & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 & h^2 q_{N-1} - 2 & 1 \\ 0 & 0 & 0 & \dots & 0 & -\alpha_1 & \alpha_1 + h\beta_1 \end{pmatrix}$$

і вектором вільних членів d, де $d_0 = h\gamma_0,\ d_N = h\gamma_1,\ d_i = h^2 f(x_i),\ i = \overline{1, N-1}$.

Беручи до уваги діагональну структуру матриці, розв'язок СЛАР (6.8.8) знаходимо методом прогонки. Отриманий вектор і буде шуканим чисельним розв'язком крайової задачі (6.8.5)-(6.8.7).

Пояснення до використання програмного коду

- Підготувати потрібні функції :
  1. виконати комірку для підготовки середовища
  2. виконати комірки, де **визначені** функції `FDA_solver`,`set_matrix_diagonals_and_vector` і `tridiagonal_matrix_algorithm`
- Обчислити чисельний розв'язок конкретної крайової задачі :
  1. виконати комірку, де **визначені** функції `q` і `f`, які задають конкретну крайову задачу
  2. виконати комірку, в якій задано усі параметри крайової задачі
  3. виконати комірку з викликом функції `FDA_solver`, перед виконанням задати відповідні аргументи цієї функції.
- Задання параметра сітки і обчислення матриці та вектора вільних членів СЛАР :
  1. виконати комірку, в якій задано параметр N
  2. виконати комірку з викликом функції set_matrix_diagonals_and_vector
- Знайти чисельний розв'язок крайової задачі:
  1. Виконати комірку з викликом функції tridiagonal_matrix_algorithm

Програмна реалізація методів

Підготовка середовища

[1]:
```
%matplotlib widget
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

`FDA_solver` – **функція, яка отримує вхідні параметри чисельного методу, формує масиви значень розв'язку на межі області, задає матрицю і вектор вільних членів СЛАР, знаходить значення чисельного розв'язку у внутрішніх вузлах сітки і повертає вектор, який містить значення розв'язку у всіх точках сітки. За замовчуванням (при `plotting=True`) будуються графіки чисельного розв'язку задачі.**

[2]:
```
def FDA_solver(f,q, alpha0,alpha1,beta0,beta1, gamma0, gamma1,l,N):
    """ Розв'язування крайових задач (6.8.5)-(6.8.7)
        методом скінчених різниць
    """
    c,a,b,d,x =␣
 ↪set_matrix_diagonals_and_vector(f,q,alpha0,alpha1,beta0,beta1, gamma0,␣
 ↪gamma1,l,N)
    u = tridiagonal_matrix_algorithm(a,b,c,d)
    return u,x
```

`set_matrix_diagonals_and_vector` – **функція для задання діагоналей матриці C і вектора d**

[3]:
```
def set_matrix_diagonals_and_vector(f,q,alpha0,alpha1,beta0,beta1, gamma0,␣
 ↪gamma1,l,N):
    """ функція задає 3 діагоналі матриці і вектор вільних членів СЛАР """

    h=l/N
    h2=h*h
    x=np.linspace(0, l, N+1)

    c = np.empty(N+1, dtype=float )
    for i in range(1,N):
        c[i] = h2*q(x[i])-2
    c[0] = -alpha0+h*beta0
    c[N] =  alpha1+h*beta1

    a=np.ones(N+1, dtype=float)
    a[0] = 0
    a[N] = -alpha1

    b=np.ones(N+1, dtype=float)
    b[0] = alpha0
    b[N] = 0

    d = np.empty(N+1, dtype=float )
    for i in range(1,N):
        d[i] = h2*f(x[i])
    d[0]=h*gamma0
    d[N]=h*gamma1
```

```
    return c,a,b,d,x
```

`tridiagonal_matrix_algorithm` – **функція, яка реалізує метод прогонки для розв'язування СЛАР**

[4]:
```
def tridiagonal_matrix_algorithm(a,b,c,g):
    """ метод прогонки для розв'язування СЛАР
        з 3-діагональною матрицею
        вектор c-головна діагональ
        вектори a і b - нижня і верхня діагоналі, паралельні головній
        вектор g - вільні члени
      """
    n1=c.size

    alpha = np.empty(n1, dtype=float )
    beta  = np.empty(n1, dtype=float )

    if c[0] !=0 :
        alpha[0] =-b[0]/c[0]
        beta [0] = g[0]/c[0]
    else:
        raise Exception('c[0]==0')

    for i in range(1,n1):
        w=a[i]*alpha[i-1]+c[i]
        if w != 0 :
            alpha[i] =-b[i]/w
            beta[i]  = (g[i] - a[i]*beta[i-1])/w
        else:
            raise Exception('w==0')

    x = np.empty(n1, dtype=float )
    n = n1-1
    x[n] = beta[n]
    for i in range(n-1,-1,-1):
        x[i] = alpha[i]*x[i+1] + beta[i]
    return x
```

`f` **і** `q` – **функції, які задають конкретне рівняння**

Обчислювальні експерименти
Продемонструємо застосування методу скінчених різниць для знаходження чисельного розв'язку задачі (6.8.5)-(6.8.7).

Приклад 1. Отримати методом скінчених різниць чисельний розв'язок крайової задачі

$u'' + u = 1$ на відрізку $[0, \pi/2]$, $u(0) = 0$, $u'(\pi/2) = 1$.

Легко бачити, що функція $u(x) = 1 - cos(x)$ є розв'язком такої задачі.

Отож, задамо відомі функції, що фігурують в рівнянні:

[5]:
```
def f1(x):
    return 1
def q1(x):
    return 1
```

А також значення усіх параметрів задачі:

[6]:
```
l=np.pi/2

alpha0=0
beta0=1
gamma0=0

alpha1=1
beta1=0
gamma1=1
```

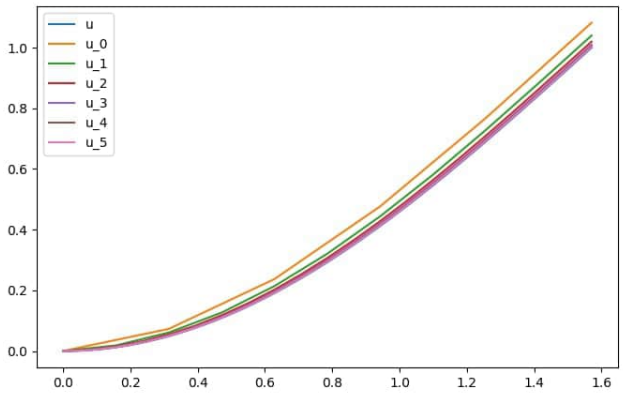
Для фіксування початкового розміру сітки будемо використовувати параметр `N_start`. Задамо його значення і спочатку обчислимо на досить густій сітці значення аналітичного (точного) розв'язку, а потім знайдемо чисельні розв'язки задачі на послідовності сіток, подвоюючи кожного разу кількість вузлів:

[7]:
```
N_start=5
x=np.linspace(0, l, N_start*64+1)
u = 1 - np.cos(x)
```

[8]:
```
N = N_start
u_0,x0 = FDA_solver(f1,q1, alpha0,alpha1,beta0,beta1, gamma0, gamma1,l,N)

N *= 2
u_1,x1 = FDA_solver(f1,q1, alpha0,alpha1,beta0,beta1, gamma0, gamma1,l,N)

N *= 2
u_2,x2 = FDA_solver(f1,q1, alpha0,alpha1,beta0,beta1, gamma0, gamma1,l,N)

N *= 2
u_3,x3 = FDA_solver(f1,q1, alpha0,alpha1,beta0,beta1, gamma0, gamma1,l,N)

N *= 2
u_4,x4 = FDA_solver(f1,q1, alpha0,alpha1,beta0,beta1, gamma0, gamma1,l,N)

N *= 2
u_5,x5 = FDA_solver(f1,q1, alpha0,alpha1,beta0,beta1, gamma0, gamma1,l,N)
```

Побудуємо графіки отриманих розв'язків:

[9]:
```
fig = plt.figure(figsize=(8, 5))
plt.plot(x, u, label='u')
plt.plot(x0, u_0, label='u_0')
plt.plot(x1, u_1, label='u_1')

plt.plot(x2, u_2, label='u_2')
plt.plot(x3, u_3, label='u_3')
plt.plot(x4, u_4, label='u_4')
plt.plot(x5, u_5, label='u_5')
ax = fig.gca()
ax.legend();
```



Зазначимо, що бібліотечна функція `plot` при побудові графіків виконує лінійну інтерполяцію по значеннях, заданих у вузлах сітки. Нагадаємо також, що в режимі `Zoom to rectangle` можна маштабувати зображення на графічній панелі.

Отож, як видно з отриманих графіків, чисельні розв'язки швидко збігаються до точного розв'язку.

Для точнішого аналізу збережемо отримані дані в таблиці, розглядаючи чисельні розв'язки лише на множині вузлів, яка є перетином побудованої послідовності сіток:

[10]:
```
df=pd.DataFrame({'x':x0[1:x0.size-1:],'u_0':u_0[1:u_0.size-1:],'u_1':u_1[2:
↪u_1.size-2:2],'u_2':u_2[4:u_2.size-4:4],'u_3':u_3[8:u_3.size-8:8],'u_4':
↪u_4[16:u_4.size-16:16],'u_5':u_5[32:u_5.size-32:32],'u':u[64:u.size-64:
↪64]})
df
```

[10]:
```
          x       u_0       u_1       u_2       u_3       u_4       u_5  \
0  0.314159  0.072848  0.060959  0.054977  0.051968  0.050458  0.049701
1  0.628319  0.237203  0.214025  0.202507  0.196749  0.193867  0.192425
2  0.942478  0.476843  0.444186  0.428141  0.420167  0.416188  0.414201
3  1.256637  0.768116  0.728865  0.709781  0.700350  0.695659  0.693319

          u
0  0.048943
1  0.190983
2  0.412215
3  0.690983
```

Обчислимо абсолютні похибки отриманих розв'язків:

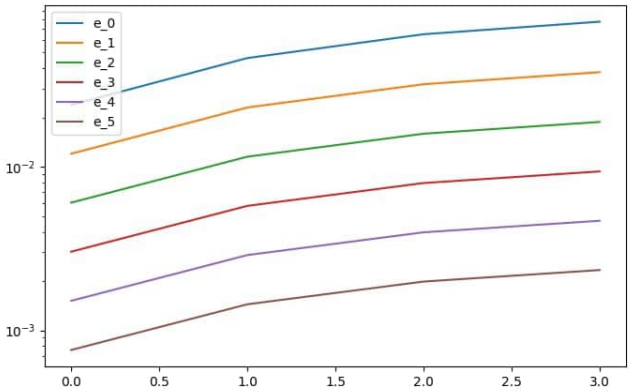
[11]:
```
df1=pd.DataFrame()
df1['e_0']=np.abs(df['u']-df['u_0'])
df1['e_1']=np.abs(df['u']-df['u_1'])
df1['e_2']=np.abs(df['u']-df['u_2'])
df1['e_3']=np.abs(df['u']-df['u_3'])
df1['e_4']=np.abs(df['u']-df['u_4'])
df1['e_5']=np.abs(df['u']-df['u_5'])
df1
```

[11]:
```
        e_0       e_1       e_2       e_3       e_4       e_5
0  0.023905  0.012015  0.006034  0.003025  0.001515  0.000758
1  0.046220  0.023042  0.011524  0.005766  0.002884  0.001442
```

```
2  0.064628  0.031971  0.015927  0.007952  0.003973  0.001986
3  0.077133  0.037882  0.018798  0.009367  0.004676  0.002336
```

Побудуємо графіки похибок в логарифмічній шкалі:

[12]:
```
fig = plt.figure(figsize=(8, 5))
df1.e_0.plot(logy=True)
df1.e_1.plot(logy=True)
df1.e_2.plot(logy=True)
df1.e_3.plot(logy=True)
df1.e_4.plot(logy=True)
df1.e_5.plot(logy=True)
ax = fig.gca()
ax.legend();
```



Обчислимо для кожного чисельного розв'язку його абсолютну похибку `ne_*` за нормою $\|\cdot\|_\infty$:

[13]:
```
ne_0 = max(np.abs(df1['e_0']))
ne_1 = max(np.abs(df1['e_1']))
ne_2 = max(np.abs(df1['e_2']))
ne_3 = max(np.abs(df1['e_3']))
ne_4 = max(np.abs(df1['e_4']))
ne_5 = max(np.abs(df1['e_5']))
print(f" ne_0={ne_0:1.2e}\n ne_1={ne_1:1.2e}\n ne_2={ne_2:1.2e}\n␣
 ↪ne_3={ne_3:1.2e}\n ne_4={ne_4:1.2e}\n ne_5={ne_5:1.2e}")
```

```
 ne_0=7.71e-02
 ne_1=3.79e-02
 ne_2=1.88e-02
 ne_3=9.37e-03
 ne_4=4.68e-03
 ne_5=2.34e-03
```

Як бачимо, подвоєння кількості вузлів сітки зумовлює зменшення вдвічі зазначеної похибки. Якщо порахуємо так звану швидкість збіжності, то отримаємо дуже близьке до 2 значення:

[14]:
```
df2=pd.DataFrame()
df2['r_0'] = df1['e_0'] / df1['e_1']
df2['r_1'] = df1['e_1'] / df1['e_2']
df2['r_2'] = df1['e_2'] / df1['e_3']
df2['r_3'] = df1['e_3'] / df1['e_4']
```

```
df2['r_4'] = df1['e_4'] / df1['e_5']

df2
```

[14]:

	r_0	r_1	r_2	r_3	r_4
0	1.989555	1.991292	1.994750	1.997147	1.998516
1	2.005872	1.999455	1.998844	1.999198	1.999543
2	2.021432	2.007431	2.002890	2.001237	2.000566
3	2.036130	2.015192	2.006883	2.003263	2.001587

Приклад 2. (Приклад 6.5) Отримати методом скінчених різниць чисельний розв'язок крайової задачі

$u'' - (x+1)u = x^2$ на відрізку $[0,3]$, $u(0) = 0$, $u(3) = 0$.

Виконуємо ті самі кроки, що і в попередньому прикладі. Відмінність лише у тому, що точний розв'язок такої задачі невідомий.

Отож, визначимо потрібні функції та задамо значення параметрів задачі:

[15]:
```
def f2(x):
    return x*x

def q2(x):
    return -(x+1)

l=3

alpha0=0
beta0=1
gamma0=0

alpha1=0
beta1=1
gamma1=0
```

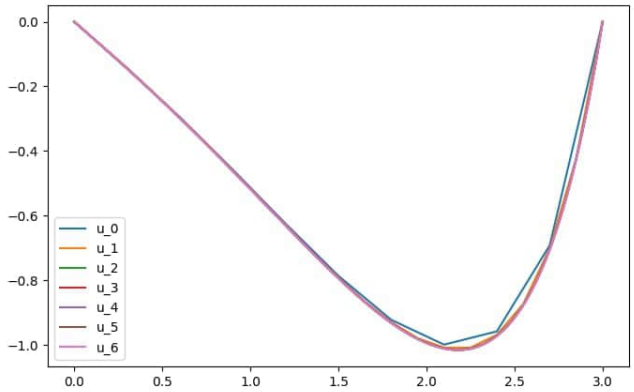
Знаходимо чисельні розв'язки на послідовності сіток, у яких на кожному кроці подвоюється кількість вузлів:

[16]:
```
N_start=10
N = N_start
u_0,x0 = FDA_solver(f2,q2, alpha0,alpha1,beta0,beta1, gamma0, gamma1,l,N)
N *= 2
u_1,x1 = FDA_solver(f2,q2, alpha0,alpha1,beta0,beta1, gamma0, gamma1,l,N)
N *= 2
u_2,x2 = FDA_solver(f2,q2, alpha0,alpha1,beta0,beta1, gamma0, gamma1,l,N)
N *= 2
u_3,x3 = FDA_solver(f2,q2, alpha0,alpha1,beta0,beta1, gamma0, gamma1,l,N)
N *= 2
u_4,x4 = FDA_solver(f2,q2, alpha0,alpha1,beta0,beta1, gamma0, gamma1,l,N)
N *= 2
u_5,x5 = FDA_solver(f2,q2, alpha0,alpha1,beta0,beta1, gamma0, gamma1,l,N)
N *= 2
u_6,x6 = FDA_solver(f2,q2, alpha0,alpha1,beta0,beta1, gamma0, gamma1,l,N)
```

Побудуємо графіки отриманих розв'язків:

[17]:
```
fig = plt.figure(figsize=(8, 5))

plt.plot(x0, u_0, label='u_0')
```

```
plt.plot(x1, u_1, label='u_1')
plt.plot(x2, u_2, label='u_2')
plt.plot(x3, u_3, label='u_3')
plt.plot(x4, u_4, label='u_4')
plt.plot(x5, u_5, label='u_5')
plt.plot(x6, u_6, label='u_6')
ax = fig.gca()
ax.legend();
```



Візуально спостерігаємо збіжність чисельних розв'язків до деякої функції. Для кількісного аналізу збережемо дані в таблиці:

[18]:
```
df=pd.DataFrame({'x':x0[1:x0.size-1:],'u_0':u_0[1:u_0.size-1:],'u_1':u_1[2:
↪u_1.size-2:2],'u_2':u_2[4:u_2.size-4:4],'u_3':u_3[8:u_3.size-8:8],'u_4':
↪u_4[16:u_4.size-16:16],'u_5':u_5[32:u_5.size-32:32],'u_6':u_6[64:u_6.
↪size-64:64]})
df
```

[18]:
```
     x       u_0       u_1       u_2       u_3       u_4       u_5       ␣
↪u_6
0  0.3 -0.143329 -0.144571 -0.144885 -0.144964 -0.144983 -0.144988 -0.
↪144989
1  0.6 -0.295327 -0.297509 -0.298062 -0.298200 -0.298235 -0.298243 -0.
↪298245
2  0.9 -0.457452 -0.460504 -0.461278 -0.461472 -0.461521 -0.461533 -0.
↪461536
3  1.2 -0.624901 -0.628958 -0.629989 -0.630248 -0.630312 -0.630329 -0.
↪630333
4  1.5 -0.786481 -0.791835 -0.793200 -0.793543 -0.793629 -0.793651 -0.
↪793656
5  1.8 -0.922520 -0.929538 -0.931334 -0.931786 -0.931899 -0.931928 -0.
↪931935
6  2.1 -0.999433 -1.008353 -1.010646 -1.011223 -1.011368 -1.011404 -1.
↪011413
7  2.4 -0.958288 -0.968669 -0.971350 -0.972026 -0.972196 -0.972238 -0.
↪972249
8  2.7 -0.691980 -0.701283 -0.703699 -0.704309 -0.704462 -0.704500 -0.
↪704509
```

Оскільки точний розв'язок невідомий, то дослідимо, як швидко чисельні розв'язки збігаються до

u_6:

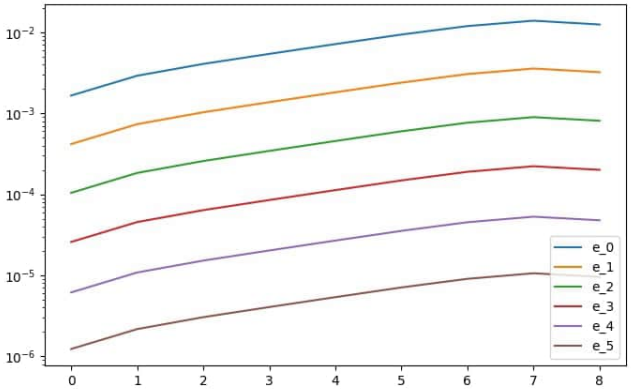
[19]:
```
df1=pd.DataFrame()
df1['e_0']=np.abs(df['u_6']-df['u_0'])
df1['e_1']=np.abs(df['u_6']-df['u_1'])
df1['e_2']=np.abs(df['u_6']-df['u_2'])
df1['e_3']=np.abs(df['u_6']-df['u_3'])
df1['e_4']=np.abs(df['u_6']-df['u_4'])
df1['e_5']=np.abs(df['u_6']-df['u_5'])
df1
```

[19]:

	e_0	e_1	e_2	e_3	e_4	e_5
0	0.001661	0.000418	0.000104	0.000026	0.000006	0.000001
1	0.002919	0.000736	0.000184	0.000045	0.000011	0.000002
2	0.004084	0.001032	0.000258	0.000064	0.000015	0.000003
3	0.005431	0.001375	0.000344	0.000085	0.000020	0.000004
4	0.007174	0.001821	0.000456	0.000113	0.000027	0.000005
5	0.009415	0.002397	0.000600	0.000149	0.000035	0.000007
6	0.011980	0.003060	0.000767	0.000190	0.000045	0.000009
7	0.013960	0.003580	0.000898	0.000222	0.000053	0.000011
8	0.012530	0.003226	0.000811	0.000201	0.000048	0.000010

Графіки (у логарифмічній шкалі) демонструють однаковий порядок отриманих похибок на усьому відрізку 1:

[20]:
```
fig = plt.figure(figsize=(8, 5))
df1.e_0.plot(logy=True)
df1.e_1.plot(logy=True)
df1.e_2.plot(logy=True)
df1.e_3.plot(logy=True)
df1.e_4.plot(logy=True)
df1.e_5.plot(logy=True)
ax = fig.gca()
ax.legend();
```



Обчислимо для кожного чисельного розв'язку його абсолютну похибку за нормою $\|\cdot\|_\infty$:

[21]:
```
ne_0 = max(np.abs(df1['e_0']))
ne_1 = max(np.abs(df1['e_1']))
ne_2 = max(np.abs(df1['e_2']))
ne_3 = max(np.abs(df1['e_3']))
```

```
ne_4 = max(np.abs(df1['e_4']))
ne_5 = max(np.abs(df1['e_5']))
print(f" ne_0={ne_0:1.2e}\n ne_1={ne_1:1.2e}\n ne_2={ne_2:1.2e}\n␣
↪ne_3={ne_3:1.2e}\n ne_4={ne_4:1.2e}\n ne_5={ne_5:1.2e}")
```

```
 ne_0=1.40e-02
 ne_1=3.58e-03
 ne_2=8.98e-04
 ne_3=2.22e-04
 ne_4=5.30e-05
 ne_5=1.06e-05
```

Якщо обчислити швидкість збіжності (до `u_6`), то матимемо число, близьке до 4, яке має тенденцію до збільшення при збільшенні вузлів сітки:

[22]:
```
df2=pd.DataFrame()
df2['r_0'] = df1['e_0'] / df1['e_1']
df2['r_1'] = df1['e_1'] / df1['e_2']
df2['r_2'] = df1['e_2'] / df1['e_3']
df2['r_3'] = df1['e_3'] / df1['e_4']
df2['r_4'] = df1['e_4'] / df1['e_5']

df2
```

[22]:

	r_0	r_1	r_2	r_3	r_4
0	3.973235	4.004190	4.045699	4.199501	4.999852
1	3.965843	4.002246	4.045203	4.199372	4.999813
2	3.958273	4.000242	4.044690	4.199239	4.999773
3	3.949734	3.997981	4.044112	4.199088	4.999729
4	3.939694	3.995326	4.043433	4.198911	4.999676
5	3.927997	3.992234	4.042642	4.198706	4.999615
6	3.914706	3.988711	4.041740	4.198471	4.999545
7	3.899928	3.984778	4.040733	4.198208	4.999466
8	3.883733	3.980444	4.039621	4.197919	4.999380

[23]:
```
plt.close('all')
```

6.8.5 Чисельне розв'язування крайових задач для еліптичних рівнянь методом скінчених різниць

6.8.5.1 Постановка задачі Діріхле для рівняння Пуассона

Нехай $D := (0,a) \times (0,b) \equiv \{(x,y) \mid 0 < x < a,\ 0 < y < b\}$, де $a > 0$ і $b > 0$ – деякі числа.

Розглянемо задачу Діріхле для рівняння Пуассона. Треба знайти функцію $u \in C^2(D) \cap C(\overline{D})$, яка задовольняє рівняння

$$u_{xx}(x,y) + u_{yy}(x,y) = f(x,y), \quad (x,y) \in D, \tag{6.8.9}$$

і крайові умови

$$u|_{x=0} = \varphi_1(y), \quad u|_{x=a} = \varphi_2(y), \quad y \in [0,b], \tag{6.8.10}$$

$$u|_{y=0} = \psi_1(x), \quad u|_{y=b} = \psi_2(x), \quad x \in (0,a), \tag{6.8.11}$$

де $f \in C(\overline{D})$, $\varphi_k \in C([0,b])$, $\psi_k \in C([0,a])$, $k = 1,2$, – задані функції, причому виконуються умови узгодження:

$$\varphi_1(0) = \psi_1(0), \quad \varphi_1(b) = \psi_2(0), \quad \varphi_2(0) = \psi_1(a), \quad \varphi_2(b) = \psi_2(a).$$

6.8.5.2 Різницева схема

Нехай N_x, N_y – які-небудь натуральні числа і

$$h_x := \frac{a}{N_x}, \quad h_y := \frac{b}{N_y}.$$

Побудуємо сітку

$$\{(x_i, y_j) \mid x_i := ih_x,\ y_j := jh_y, \quad i = \overline{0, N_x},\ j = \overline{0, N_y}\},$$

на якій розглянемо наближення функції u

$$u_{i,j} \approx u(x_i, y_j),\ i = \overline{0, N_x},\ j = \overline{0, N_y},$$

і сіткові значення вхідних даних:

$$f_{i,j} := f(x_i, y_j), \quad \varphi_{k,j} := \varphi_k(y_j), \quad \psi_{k,i} := \psi_k(x_i), \quad k = \overline{1,2}, \quad i = \overline{1, N_x - 1},\ j = \overline{1, N_y - 1}.$$

Позначимо $N := (N_x - 1) \times (N_y - 1)$, $\alpha := h_x^{-2},\ \beta := h_y^{-2}$.

Сіткові значення впорядкуємо у вигляді векторів $v, g \in \mathbb{R}^N$ так, що

$$v_k = u_{i,j} \text{ при } k = (i-1)(N_y - 1) + j - 1, \quad i = \overline{1, N_x - 1},\ j = \overline{1, N_y - 1}, \quad k = \overline{0, N - 1},$$

$$g_k = \begin{cases} f_{1,1} - \beta\psi_{1,1} - \alpha\varphi_{1,1}, & k = 0, \\ f_{1,N_y-1} - \beta\psi_{2,1} - \alpha\varphi_{1,N_y-1}, & k = N_y - 2, \\ f_{N_x-1,1} - \beta\psi_{1,N_x-1} - \alpha\varphi_{2,1}, & k = (N_x - 2)(N_y - 1), \\ f_{N_x-1,N_y-1} - \beta\psi_{2,N_x-1} - \alpha\varphi_{2,N_y-1}, & k = N - 1, \\ f_{1,j} - \alpha\varphi_{1,j}, \quad j = \overline{2, N_y - 2}, & k = j - 1, \\ f_{N_x-1,j} - \alpha\varphi_{2,j}, \quad j = \overline{2, N_y - 2}, & k = (N_x - 2)(N_y - 1) + j - 1, \\ f_{i,1} - \beta\psi_{1,i}, \quad i = \overline{2, N_x - 2}, & k = (i-1)(N_y - 1), \\ f_{i,N_y-1} - \beta\psi_{2,i}, \quad i = \overline{2, N_x - 2}, & k = i(N_y - 1) - 1, \\ f_{i,j}, \quad i = \overline{2, N_x - 2},\ j = \overline{2, N_y - 2}, & k = (i-1)(N_y - 1) + j - 1. \end{cases}$$

Якщо у внутрішніх вузлах сітки використати такі апроксимації:

$$u_{xx}(x_i, y_j) \approx \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h_x^2}, \quad u_{yy}(x_i, y_j) \approx \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h_y^2}, i = \overline{1, N_x - 1},\ j = \overline{1, N_y - 1},$$

то різницеву схему для задачі (6.8.9)-(6.8.11) можна звести до такої СЛАР відносно шуканих наближень розв'язку:

$$Cv = g, \tag{6.8.12}$$

з матрицею $C \in \mathbb{R}^{N\times N}$, яка має блочну структуру

$$C = \begin{pmatrix} B & A & 0 & 0 & \dots & 0 & 0 & 0 \\ A & B & A & 0 & \dots & 0 & 0 & 0 \\ 0 & A & B & A & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \\ 0 & 0 & 0 & 0 & \dots & A & B & A \\ 0 & 0 & 0 & 0 & \dots & 0 & A & B \end{pmatrix}$$

де блоки $A, B \in \mathbb{R}^{(N_y-1)\times(N_y-1)}$ є діагональною і тридіагональною матрицями відповідно, які мають вираз

$$A = \begin{pmatrix} \alpha & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & \alpha & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & \alpha & 0 & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \\ 0 & 0 & 0 & 0 & \dots & 0 & \alpha & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & \alpha \end{pmatrix}$$

$$B = \begin{pmatrix} -2(\alpha+\beta) & \beta & 0 & 0 & \dots & 0 & 0 & 0 \\ \beta & -2(\alpha+\beta) & \beta & 0 & \dots & 0 & 0 & 0 \\ 0 & \beta & -2(\alpha+\beta) & \beta & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \\ 0 & 0 & 0 & 0 & \dots & \beta & -2(\alpha+\beta) & \beta \\ 0 & 0 & 0 & 0 & \dots & 0 & \beta & -2(\alpha+\beta) \end{pmatrix}$$

Співвідношення (6.8.12) утворюють СЛАР відносно шуканих значень наближень розв'язку даної задачі.

Пояснення до використання програмного коду

- Підготувати потрібні функції :
  1. виконати комірку для підготовки середовища
  2. виконати комірки, де **визначені** функції `FDA_E_solver`,`set_matrix` і `set_vector`
- Обчислити чисельний розв'язок конкретної крайової задачі :
  1. виконати комірку, де **визначені** функції `phi1`,`phi2`, `psi1`, `psi2` і `f`, які задають конкретну крайову задачу
  2. виконати комірку, в якій задано усі параметри крайової задачі
  3. виконати комірку з викликом функції `FDA_E_solver`, перед виконанням задати відповідні аргументи цієї функції.
- Щоб переконатися, що чисельний розв'язок достатньо точний, можна виконати кілька послідовних викликів функції `FDA_E_solver`, збільшуючи кількість вузлів сітки.

Програмна реалізація методів

Підготовка середовища

[1]:
```
%matplotlib widget
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy import linalg
from mpl_toolkits import mplot3d
```

`FDA_E_solver` – функція, яка отримує вхідні параметри чисельного методу, формує масиви значень розв'язку на межі області, задає матрицю вектор вільних членів СЛАР, знаходить значення чисельного розв'язку у внутрішніх вузлах сітки і повертає матрицю, яка містить значення розв'язку у всіх точках сітки. За замовчуванням (при `plotting=True`) будуються 3D-графіки чисельного розв'язку задачі.

[2]:
```
def FDA_E_solver(f, phi1, phi2, psi1, psi2, a, b, Nx, Ny, plotting=True):
    """ Розв'язування крайової задачі (6.8.9) - (6.8.11)
        методом скінчених різниць
    """
    hx = a/Nx
    hy = b/Ny
    alpha = 1/(hx*hx)
    beta  = 1/(hy*hy)

    N = (Nx-1)*(Ny-1)

    c = set_matrix(alpha,beta,Nx, Ny)

    x=np.linspace(0, a, Nx+1)
    y=np.linspace(0, b, Ny+1)

    ps1 = np.empty(Nx+1, dtype=float)
    ps2 = np.empty(Nx+1, dtype=float)
    for i in range(Nx+1):
        ps1[i] = psi1(x[i])
        ps2[i] = psi2(x[i])

    ph1 = np.empty(Ny+1, dtype=float)
    ph2 = np.empty(Ny+1, dtype=float)
    for j in range(Ny+1):
        ph1[j] = phi1(y[j])
        ph2[j] = phi2(y[j])

    g = set_vector(f, ph1, ph2, ps1, ps2, x, y, alpha,beta, Nx, Ny)

    u = linalg.solve(c, g)

    uij=u.reshape(((Nx-1),(Ny-1)))

    ue = np.empty((Nx+1,Ny+1), dtype=float)

    ue[0,:] = ph1
    ue[Nx,:]= ph2
    ue[:,Ny]= ps2
    ue[:,0] = ps1
    for i in range(1,Nx):
        for j in range(1,Ny):
            ue[i,j] = uij[i-1,j-1]

    if plotting :
        Y, X = np.meshgrid(y, x)
        # set up a figure twice as wide as it is tall
        fig = plt.figure(figsize=plt.figaspect(0.5))
        # =============
        # First subplot
        # =============
```

```
        # set up the axes for the first plot
        ax = fig.add_subplot(1, 2, 1, projection='3d')
        ax.set_xlabel('y')
        ax.set_ylabel('x')
        ax.set_zlabel('u');
        ax.set_title(f"Розв'язок задачі Nx={Nx}, Ny={Ny}");
        surf = ax.plot_surface(Y, X, ue, rstride=1, cstride=1,␣
↪cmap='viridis', linewidth=0, antialiased=False)
        fig.colorbar(surf, shrink=0.5, aspect=10)
        # ==============
        # Second subplot
        # ==============
        # set up the axes for the second plot
        ax = fig.add_subplot(1, 2, 2, projection='3d')
        ax.set_xlabel('y')
        ax.set_ylabel('x')
        ax.set_zlabel('u');
        ax.set_title(f"Розв'язок задачі Nx={Nx}, Ny={Ny}");
        surf = ax.plot_wireframe(Y, X, ue, rstride=5, cstride=5)
    return ue
```

set_matrix – функція, яка формує матрицю C

[3]:
```
def set_matrix(alpha,beta,Nx, Ny):
    """ функція задає матрицю СЛАР """

    N = (Nx-1)*(Ny-1)
    c=np.zeros((N,N), dtype=float)
    c[0,0] = -2*(alpha+beta)
    c[0,1] = beta
    c[N-1,N-1] = -2*(alpha+beta)
    c[N-1,N-2] = beta
    for i in range(1,N-1):
        c[i,i-1] = beta
        c[i,i]   = -2*(alpha+beta)
        c[i,i+1] = beta
    nstop = (Ny-1)*(Nx-2)
    for k in range(Ny-2, nstop, Ny-1):
        c[k,k+1] = 0
        c[k+1,k] = 0


    for k in range(nstop):
        c[k,k+Ny-1] = alpha
        c[k+Ny-1,k] = alpha

    return c
```

set_vector – функція, яка формує вектор вільних членів СЛАР

[4]:
```
def set_vector(f, ph1, ph2, ps1, ps2, x, y, alpha,beta, Nx, Ny):
    """ функція задає вектор вільних членів СЛАР """

    N = (Nx-1)*(Ny-1)
    g=np.empty(N, dtype=float)

    k=0
```

```
    for i in range(1,Nx):
        for j in range(1,Ny):
            g[k] = f(x[i],y[j])
            k += 1

    if Nx==3 and Ny==2:
        g[0]-= beta*(ps1[1]+ps2[1])+alpha*ph1[1]
        g[1]-= beta*(ps1[2]+ps2[2])+alpha*ph2[1]
        print(f"g={g}")
        return g

    if Nx < 4 or Ny < 4:
        raise Exception('invalid grid')

    k=0
    g[k] -= beta*ps1[1]+alpha*ph1[1]
    k=Ny-2
    g[k] -= beta*ps2[1]+alpha*ph1[Ny-1]
    k=(Nx-2)*(Ny-1)
    g[k] -= beta*ps1[Nx-1]+alpha*ph2[1]
    k=N-1
    g[k] -= beta*ps2[Nx-1]+alpha*ph2[Ny-1]

    for j in range(2,Ny-2):
        k=j-1
        g[k] -= alpha*ph1[j]
        k=(Nx-2)*(Ny-1)+j-1
        g[k] -= alpha*ph2[j]
    for i in range(2,Nx-2):
        k=(i-1)*(Ny-1)
        g[k] -= beta*ps1[i]
        k=i*(Ny-1)-1
        g[k] -= beta*ps2[i]

    return g
```

`phi1`,`phi2`, `psi1`, `psi2` і `f` – функції, які задають конкретну крайову задачу

Обчислювальні експерименти

Продемонструємо застосування методу скінчених різниць до розв'язування крайової задачі Діріхле для рівняння Пуасона.

Приклад 1. (приклад 1.6) Знайти чисельний розв'язок крайової задачі

$$\Delta u = x + y, \quad (x, y) \in (0, 3) \times (0, 2), \tag{6.8.13}$$

$$u|_{x=0} = y, \quad u|_{x=3} = 2y, \quad y \in [0, 2], \tag{6.8.14}$$

$$u|_{y=0} = 0, \quad u|_{y=2} = \frac{2}{3}x + 2, \quad x \in (0, 3), \tag{6.8.15}$$

застосовуючи метод скінчених різниць на послідовності сіток, в яких подвоюється кількість вузлів по кожній змінній.

Визначимо функції, які задають праву частину рівняння (6.8.9) і крайові умови на сторонах прямокутника:

[5]:
```
def f(x,y):
    return x+y
def phi1(y):
```

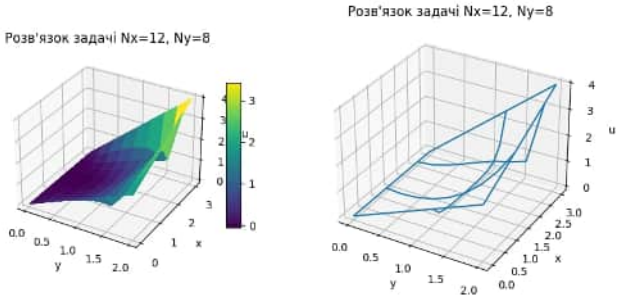
```
    return y
def phi2(y):
    return 2*y
def psi1(x):
    return 0
def psi2(x):
    return 2*x/3+2
```

Задамо сторони прямокутника і початкові значення параметрів сітки:

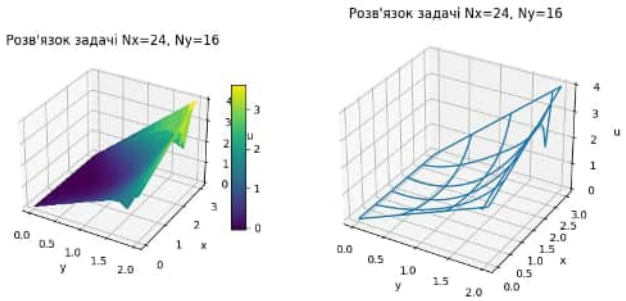
[6]:
```
a=3
b=2
Nx_start = 12
Ny_start = 8
```

Знайдемо чисельні розв'язки задачі на послідовності сіток, подвоюючи на кожному кроці кількість вузлів по кожній змінній і зберігаючи відповідні масиви результатів; для подальшого аналізу також зберігатимемо значення чисельного розв'язку у точках початкової сітки:

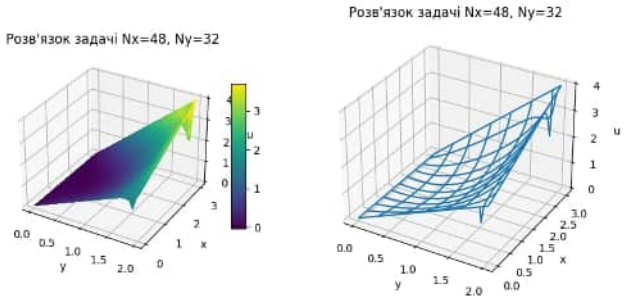
[7]:
```
Nx=Nx_start
Ny=Ny_start
u_0 = FDA_E_solver(f, phi1, phi2, psi1, psi2, a, b, Nx, Ny)
u0=u_0[1:Nx:1,1:Ny:1].reshape((Nx-1)*(Ny-1))
```



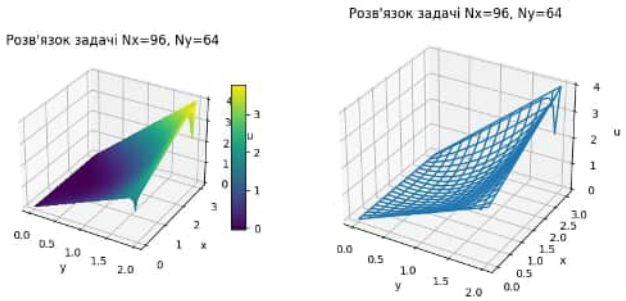
[8]:
```
Nx *= 2
Ny *= 2
u_1 = FDA_E_solver(f, phi1, phi2, psi1, psi2, a, b, Nx, Ny)
u1=u_1[2:Nx-1:2,2:Ny-1:2].reshape((Nx_start-1)*(Ny_start-1))
```



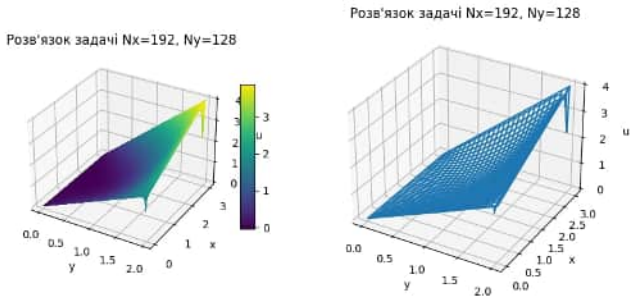
[9]:
```
Nx *= 2
Ny *= 2
u_2 = FDA_E_solver(f, phi1, phi2, psi1, psi2, a, b, Nx, Ny)
u2=u_2[4:Nx-3:4,4:Ny-3:4].reshape((Nx_start-1)*(Ny_start-1))
```



[10]:
```
Nx *= 2
Ny *= 2
u_3 = FDA_E_solver(f, phi1, phi2, psi1, psi2, a, b, Nx, Ny)
u3=u_3[8:Nx-7:8,8:Ny-7:8].reshape((Nx_start-1)*(Ny_start-1))
```



[11]:
```
Nx *= 2
Ny *= 2
u_4 = FDA_E_solver(f, phi1, phi2, psi1, psi2, a, b, Nx, Ny)
u4=u_4[16:Nx-15:16,16:Ny-15:16].reshape((Nx_start-1)*(Ny_start-1))
```



Як бачимо, графіки чисельних розв'язків візуально подібні між собою. Для детальнішого аналізу занесемо в таблицю значення усіх розв'язків на спільній для усіх сіток множині вузлів:

[12]:
```
df=pd.DataFrame({'u_0':u0, 'u_1':u1,'u_2':u2, 'u_3':u3,'u_4':u4})
df
```

[12]:

	u_0	u_1	u_2	u_3	u_4
0	0.164497	0.171949	0.173837	0.174327	0.174453
1	0.344323	0.361248	0.365401	0.366471	0.366745
2	0.536998	0.569520	0.577063	0.578975	0.579464
3	0.731952	0.797518	0.811631	0.815112	0.815995
4	0.889774	1.041444	1.072022	1.079294	1.081118
..	...	...	...	...	...
72	0.861878	0.949424	0.972746	0.978782	0.980323
73	1.177350	1.354034	1.401200	1.413340	1.416432
74	1.374996	1.757882	1.863289	1.890393	1.897288
75	1.141355	2.065495	2.340931	2.413504	2.432058
76	2.370952	1.972072	2.763810	2.983314	3.040900

[77 rows x 5 columns]

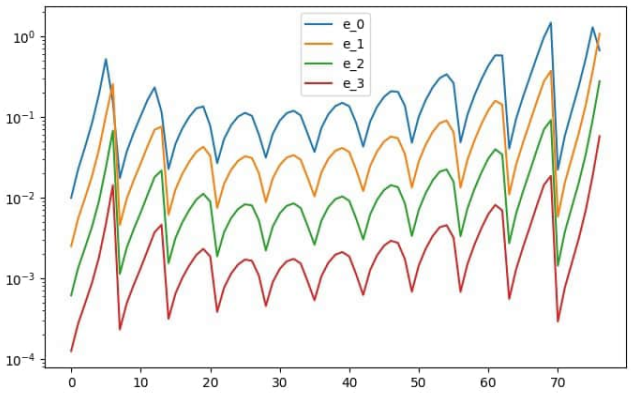
[13]:
```
df1=pd.DataFrame()
df1['e_0']=np.abs(df['u_4']-df['u_0'])
df1['e_1']=np.abs(df['u_4']-df['u_1'])
df1['e_2']=np.abs(df['u_4']-df['u_2'])
df1['e_3']=np.abs(df['u_4']-df['u_3'])

df1
```

[13]:

	e_0	e_1	e_2	e_3
0	0.009956	0.002504	0.000616	0.000126
1	0.022423	0.005497	0.001344	0.000274
2	0.042465	0.009944	0.002401	0.000488
3	0.084043	0.018477	0.004363	0.000883
4	0.191344	0.039674	0.009096	0.001823
..	...	...	...	...
72	0.118445	0.030899	0.007576	0.001541
73	0.239082	0.062398	0.015232	0.003093
74	0.522292	0.139406	0.033998	0.006894
75	1.290703	0.366563	0.091127	0.018554
76	0.669948	1.068827	0.277090	0.057586

[77 rows x 4 columns]

[14]:
```
fig = plt.figure(figsize=(8, 5))
df1.e_0.plot(logy=True)
df1.e_1.plot(logy=True)
df1.e_2.plot(logy=True)
df1.e_3.plot(logy=True)

ax = fig.gca()
ax.legend();
```



[15]:
```
ne_0 = max(np.abs(df1['e_0']))
ne_1 = max(np.abs(df1['e_1']))
ne_2 = max(np.abs(df1['e_2']))
ne_3 = max(np.abs(df1['e_3']))

print(f" ne_0={ne_0:1.2e}\n ne_1={ne_1:1.2e}\n ne_2={ne_2:1.2e}\n␣
↪ne_3={ne_3:1.2e}")
```

```
 ne_0=1.47e+00
 ne_1=1.07e+00
 ne_2=2.77e-01
 ne_3=5.76e-02
```

[16]:
```
df2=pd.DataFrame()
df2['r_0'] = df1['e_0'] / df1['e_1']
df2['r_1'] = df1['e_1'] / df1['e_2']
df2['r_2'] = df1['e_2'] / df1['e_3']

df2
```

[16]:

	r_0	r_1	r_2
0	3.976692	4.064306	4.895639
1	4.079051	4.089734	4.902357
2	4.270485	4.141921	4.916664
3	4.548628	4.234533	4.943848
4	4.822896	4.361724	4.988298
..	...	...	...
72	3.833333	4.078289	4.917374
73	3.831570	4.096498	4.925325
74	3.746554	4.100383	4.931254
75	3.521095	4.022558	4.911399
76	0.626807	3.857329	4.811776

[77 rows x 3 columns]

[17]:
```
plt.close('all')
```

Розділ 7

Чисельне розв'язування інтегральних рівнянь

7.1. Основи теорії інтегральних рівнянь Фредгольма другого роду

Нехай $n \in \mathbb{N}$ — деяке натуральне число, $G \subset \mathbb{R}^n$ — замикання обмеженої області або замкнена кусково-гладка обмежена поверхня розмірності $m \in \{1, \ldots, n-1\}$. Припустимо, що

$$K(x,y) = \frac{h(x,y)}{|x-y|^\gamma}, \quad x, y \in G, \ x \neq y, \tag{7.1.1}$$

де $h \in C(G \times G)$ — задана дійсна функція, γ — дійсне число таке, що $\gamma < n$, якщо G — замикання обмеженої області, і $\gamma < m$, якщо G — поверхня розмірності m.

Функцію

$$K^*(x,y) := K(y,x), \quad x, y \in G, \quad x \neq y, \tag{7.1.2}$$

називають *спряженою* до функції K.

Лінійним інтегральним рівнянням Фредгольма другого роду називають рівняння

$$u - \lambda \int\limits_G K(x,y) u \, d\mu(y) = f(x), \quad x \in G, \tag{7.1.3}$$

де $\lambda \in \mathbb{R}$ — параметр, $f \in C(G)$ — задана функція, $u \in C(G)$ — невідома функція, $d\mu(\cdot)$ — елемент міри на множині G.

Функцію K називають *ядром* інтегрального рівняння (7.1.3).

Лінійне інтегральне рівняння Фредгольма другого роду

$$w(x) - \lambda \int\limits_G K^*(x,y) w(y) \, d\mu(y) = g(x), \quad x \in G, \tag{7.1.4}$$

де $g \in C(G)$ – задана функція, $w \in C(G)$ – невідома функція, називають ***спряженим*** до рівняння ((7.1.2)).

Якщо в рівнянні (7.1.3) маємо $f(x) = 0$, $x \in G$, тобто воно має вигляд

$$u - \lambda \int\limits_G K(x,y) u \, d\mu(y) = 0, \quad x \in G, \tag{7.1.5}$$

та це інтегральне рівняння називають *однорідним*, а в іншому випадку — *неоднорідним*. Якщо в рівнянні (7.1.3) $f \neq 0$ і функція K та параметр λ в рівнянні (7.1.5) такі ж, як в рівнянні (7.1.3),

то кажуть, що рівняння (7.1.5) є *відповідним* рівнянню (7.1.3) однорідним рівнянням. Розглядаємо також відповідне рівнянню (7.1.4) однорідне рівняння

$$w(x) - \lambda \int\limits_G K^*(x,y)w(y)\,d\mu(y) = 0, \quad x \in G. \tag{7.1.6}$$

Теорема 7.1.1 (альтернатива Фредгольма)**.** *Нехай $\lambda \in \mathbb{R}$ — яке-небудь число. Правильні такі твердження.*

1°. *Рівняння (7.1.5) і (7.1.6) (спряжені однорідні інтегральні рівняння) мають скінченну та однакову максимальну кількість лінійно незалежних розв'язків, зокрема, якщо одне з них має тільки тривіальний (нульовий) розв'язок, то друге теж має тільки тривіальний розв'язок.*

2°. *Якщо рівняння (7.1.5) (відповідно, рівняння (7.1.6)) має тільки тривіальний розв'язок, то рівняння (7.1.3) (відповідно, рівняння (7.1.4)) має і тільки один розв'язок для будь-якої функції $f \in C(G)$ (відповідно, $g \in C(G)$).*

3°. *Якщо рівняння (7.1.5) має нетривіальні розв'язки та m — максимальна кількість лінійно незалежних розв'язків цього рівняння, то рівняння (7.1.3) має розв'язки тоді і тільки тоді, коли виконуються рівності*

$$\int\limits_G f(x)w_i(x)\,d\mu(x) = 0, \quad i = \overline{1,m}, \tag{7.1.7}$$

де $w_1, ..., w_m$ — лінійно незалежні розв'язки рівняння (7.1.6), причому будь-який розв'язок рівняння (7.1.3) має вигляд

$$v(x) = \sum_{i=1}^{m} C_i v_i(x) + v^*(x), \quad x \in G, \tag{7.1.8}$$

де v^* — який-небудь фіксований розв'язок цього рівняння, $v_1, ..., v_m$ — лінійно незалежні розв'язки рівняння (7.1.5), а $C_1, ..., C_m$ — довільні сталі.

Зауваження 7.1.1. Очевидно, що якщо рівняння (7.1.6) має нетривіальні розв'язки і m – максимальна кількість лінійно незалежних розв'язків цього рівняння, то рівняння (7.1.4) має розв'язки тоді і тільки тоді, коли виконується умова

$$\int\limits_G g(x)v_i(x)\,d\mu(y) = 0, \quad i = \overline{1,m},$$

де а $v_1, ..., v_m$ – лінійно незалежні розв'язки рівняння (7.1.5).

Значення λ^* параметра λ, при якому рівняння (7.1.5) має нетривіальні розв'язки, називають *характеристичним числом* ядер K та K^*, а число m — максимальна кількість лінійно незалежних розв'язків рівняння (7.1.5) (відповідно, (7.1.6)) — *кратністю* характеристичного числа λ^*.

Теорема 7.1.2. *Ядра K та K^* мають не більше ніж зліченну кількість характеристичних чисел і множини цих чисел не мають скінчених точок згущення.*

7.2. Методи розв'язування лінійних інтегральних рівнянь Фредгольма другого роду

7.2.1. Метод послідовних наближень

Нехай

$$-\infty < a < b < +\infty, \quad K \in C([a,b] \times [a,b]), \quad f \in C([a,b]), \quad \lambda \in \mathbb{R}.$$

Розглянемо лінійне одновимірне інтегральне рівняння Фредгольма другого роду

$$u - \lambda \int_a^b K(x,y)u\,dy = f(x), \quad x \in [a,b]. \tag{7.2.1}$$

Під *розв'язком* рівняння (7.2.1) розуміють функцію $u = u(x)$, $x \in [a, b]$, з простору $C([a, b])$, яка при підставлянні в рівняння (7.2.1) перетворює його в тотожність:

$$u(x) - \lambda \int_a^b K(x, y)u(y)\, dy = f(x), \quad x \in [a, b]. \tag{7.2.2}$$

Перепишемо рівність (7.2.2) у вигляді

$$u(x) = \lambda \int_a^b K(x, y)u(y)\, dy + f(x), \quad x \in [a, b].$$

Метод послідовних наближень знаходження розв'язків (7.2.1) полягає у побудові функційної послідовності

$$u_0(\cdot), u_1(\cdot), \ldots, u_k(\cdot), \ldots \tag{7.2.3}$$

за правилом

$$\begin{cases} u_0(x) = f(x), \quad x \in [a, b], \\ \\ u_{k+1}(x) = \lambda \int\limits_a^b K(x, y)u_k(y)\, dy + f(x), \quad x \in [a, b], \quad k \in \mathbb{N}_0 := \mathbb{N} \cup \{0\}. \end{cases} \tag{7.2.4}$$

Твердження 7.2.1. *Якщо*

$$|\lambda| < 1/ \max_{x \in [a,b]} \int_a^b |K(x, y)|\, dy, \tag{7.2.5}$$

то послідовність (7.2.3) *рівномірно збігається до розв'язку рівняння* (7.2.1).

Переконатися, що послідовність послідовних наближень розв'язку даного інтегрального рівняння рівномірно збігається і знайти перші три члени цієї послідовності:

$$u - \frac{1}{3} \int_0^1 (x + y)u\, dy = x, \quad x \in [0, 1].$$

Розв'язування. В даному рівнянні

$$\lambda = \frac{1}{3}, \quad K(x, y) := x + y, \quad (x, y) \in [0; 1].$$

Маємо

$$\max_{x \in [0;1]} \int_0^1 |x + y|\, dy = \max_{x \in [0;1]} (xy + y^2/2)\Big|_{y=0}^{y=1} = \max_{x \in [0;1]} (x + 1/2) = 3/2.$$

Оскільки $1/\max_{x \in [0;1]} \int_0^1 |x + y|\, dy = 2/3$, то звідси та умови (7.2.5) легко випливає, що послідовність послідовних наближень розв'язку даного інтегрального рівняння рівномірно збігається до розв'язку.

Тепер запишемо дане рівняння у вигляді

$$u = \frac{1}{3} \int_0^1 (x + y)u dy + x.$$

Приймаємо

$$u_0(x) := x, \quad x \in [0, 1].$$

і згідно з (7.2.4) знаходимо

$$u_{k+1}(x) := \frac{1}{3} \int_0^1 (x + y)u_k(y)\, dy + x, \quad x \in [0, 1], \quad k \in \mathbb{N} \cup \{0\}.$$

Отож, отримуємо

$$u_1(x) := \frac{1}{3} \int_0^1 (x + y)u_0(y)\, dy + x = \frac{1}{3} \int_0^1 (x + y)y\, dy + x =$$
$$= \frac{1}{3} \int_0^1 (xy + y^2)dy + x = \frac{1}{3} \left(x\frac{y^2}{2} + \frac{y^3}{3} \right) \Big|_{y=0}^{y=1} + x = \frac{1}{3} \left(\frac{7}{2}x + \frac{1}{3} \right), \quad x \in [0; 1].$$

$$u_2(x) = \frac{1}{3}\int_0^1 (x+y)u_1(y)\,dy + x = \frac{1}{9}\int_0^1 (x+y)\left(\frac{7}{2}y + \frac{1}{3}\right)\,dy + x =$$
$$= \frac{1}{9}\int_0^1 \left[\frac{7}{2}xy + \frac{1}{3}x + \frac{7}{2}y^2 + \frac{1}{3}y\right]\,dy + x = \frac{1}{9}\left[\frac{7}{4}xy^2 + \frac{1}{3}xy + \frac{7}{6}y^3 + \frac{1}{6}y^2\right]\Big|_{y=0}^{y=1} + x =$$
$$= \frac{1}{9}\left[\frac{7}{4}x + \frac{1}{3}x + \frac{8}{6}\right] + x = \frac{133}{108}x + \frac{4}{27}, \quad x \in [0;1].$$

Вправи для самостійної роботи

Переконатися, що послідовність послідовних наближень розв'язку даного інтегрального рівняння рівномірно збігається і знайти перші три члени цієї послідовності:

1. $u - \frac{1}{6}\int\limits_0^2 xyu\,dy = x + 1, \quad x \in [0;2].$

2. $u - \frac{1}{10}\int\limits_0^3 (x+2y)u\,dy = x^2, \quad x \in [0;3].$

7.2.2. Метод механічних квадратур

Припустимо, що $u = u(x)$, $x \in [a,b]$, — розв'язок рівняння (7.2.1). Нехай $n \in \mathbb{N}$ — довільне фіксоване число,

$$x_0 := a, \quad x_i := a + ih, \quad i = \overline{0,n}, \quad \text{де} \;\; h := \frac{b-a}{n}.$$

Далі використовуємо формулу чисельного інтегрування

$$\int_a^b g(y)\,dy = \sum_{j=0}^{n} A_j^{(n)} g(x_j) + R_n(g), \quad g \in C[a,b], \tag{7.2.6}$$

де $A_0^{(n)}, A_1^{(n)}, \ldots, A_n^{(n)}$ — числа, які не залежать від g, $R_n(g)$ — залишковий член.

На підставі (7.2.6) маємо

$$\int_a^b K(x,y)u(y)\,dy = \sum_{j=0}^{n} A_j^{(n)} K(x,x_j)u(x_j) + R_n(Ku)(x), \quad x \in [a,b]. \tag{7.2.7}$$

Тому рівність (7.2.2) можна записати у вигляді

$$u(x) - \lambda\left[\sum_{j=0}^{n} A_j^{(n)} K(x,x_j)u(x_j) + R_n(Ku)(x)\right] = f(x), \quad x \in [a,b].$$

Звідси, покладаючи почергово $x = x_i$, $i = \overline{0,n}$, і нехтуючи залишковим членом $R_n(Ku)(\cdot)$, вважаючи його достатньо малим, отримаємо для наближень $u_i \approx u(x_i)$, $i = \overline{0,n}$, таку систему лінійних алгебраїчних рівнянь

$$u_i - \lambda\sum_{j=0}^{n} A_j^{(n)} K_{ij} u_j = f_i, \quad i = \overline{0,n}, \tag{7.2.8}$$

де

$$K_{ij} := K(x_i,x_j), \quad f_i := f(x_i), \quad i,j = \overline{0,n}.$$

Методом механічних квадратур розв'язки рівняння:

$$u - \int_0^2 (x+y)u dy = x^2, \quad x \in [0;2]. \tag{7.2.9}$$

Розв'язування. Візьмемо $n = 2$. Тоді $x_0 := 0, x_1 := 1, x_2 := 2$. Використаємо чисельне інтегрування методом трапецій:

$$\int_0^2 g(y)dy \approx \frac{1}{2}g(0) + g(1) + \frac{1}{2}g(2).$$

Отже,

$$A_0^{(2)} = \frac{1}{2}; \quad A_1^{(2)} = 1; \quad A_2^{(2)} = \frac{1}{2}.$$

Маємо $K(x, y) = x + y$, $\lambda = 1$. Тоді

$$K_{i,j} := K(x_i, x_j) = x_i + x_j = i + j, \quad i, j = \overline{0, 2},$$

тобто

$$\begin{aligned} &K_{0,0} := 0 + 0 = 0; \quad K_{0,1} := 0 + 1 = 1; \quad K_{0,2} := 0 + 2 = 2;\\ &K_{1,0} := 1 + 0 = 1; \quad K_{1,1} := 1 + 1 = 2; \quad K_{1,2} := 1 + 2 = 3;\\ &K_{2,0} := 2 + 0 = 2; \quad K_{2,1} := 2 + 1 = 3; \quad K_{2,2} := 2 + 2 = 4. \end{aligned}$$

Отже, для знаходження u_0, u_1, u_2 маємо систему лінійних алгебраїчних рівнянь

$$u_i - \sum_{j=0}^{2} A_j^{(2)} K_{ij} u_j = f_i, \quad i = 0, 1, 2.$$

де

$$f_0 := x_0^2 = 0; \quad f_1 := x_1^2 = 1^2 = 1; \quad f_2 := x_2^2 = 2^2 = 4.$$

Деталізуємо цю СЛАР:

$$\begin{cases} u_0 - (A_0^{(2)} K_{00} u_0 + A_1^{(2)} K_{01} u_1 + A_2^{(2)} K_{02} u_2) = f_0 \\ u_1 - (A_0^{(2)} K_{10} u_0 + A_1^{(2)} K_{11} u_1 + A_2^{(2)} K_{12} u_2) = f_1 \\ u_2 - (A_0^{(2)} K_{20} u_0 + A_1^{(2)} K_{21} u_1 + A_2^{(2)} K_{22} u_2) = f_2 \end{cases},$$

а точніше

$$\begin{cases} u_0 - (\frac{1}{2} \cdot 0 \cdot u_0 + 1 \cdot 1 \cdot u_1 + \frac{1}{2} \cdot 2 \cdot u_2) = 0 \\ u_1 - (\frac{1}{2} \cdot 1 \cdot u_0 + 1 \cdot 2 \cdot u_1 + \frac{1}{2} \cdot 3 \cdot u_2) = 1 \\ u_2 - (\frac{1}{2} \cdot 2 \cdot u_0 + 1 \cdot 3 \cdot u_1 + \frac{1}{2} \cdot 4 \cdot u_2) = 4 \end{cases} \Leftrightarrow$$

$$\Leftrightarrow \begin{cases} u_0 - u_1 - u_2 = 0 \\ -\frac{1}{2}u_0 - u_1 - \frac{3}{2}u_2 = 1 \\ -u_0 - 3u_1 - u_2 = 4 \end{cases} \Leftrightarrow \begin{cases} u_0 - u_1 - u_2 = 0 \\ u_0 + 2u_1 + u_2 = -2 \\ u_0 + 3u_1 + u_2 = -4 \end{cases}.$$

Розв'яжемо отриману СЛАР методом Гаусса:

$$\left(\begin{array}{ccc|c} 1 & -1 & -1 & 0 \\ 1 & 2 & 3 & -2 \\ 1 & 3 & 1 & -4 \end{array}\right) \sim \left(\begin{array}{ccc|c} 1 & -1 & -1 & 0 \\ 0 & 3 & 4 & -2 \\ 0 & 4 & 2 & -4 \end{array}\right) \sim \left(\begin{array}{ccc|c} 1 & -1 & -1 & 0 \\ 0 & 1 & 1/2 & -1 \\ 0 & 0 & 5/2 & 1 \end{array}\right).$$

Отже, маємо

$$\begin{cases} u_0 - u_1 - u_2 = 0 \\ u_1 + \frac{1}{2}u_2 = -1 \\ \frac{5}{2}u_2 = 1 \end{cases} \Leftrightarrow \begin{cases} u_2 = \frac{2}{5} \\ u_1 = -\frac{6}{5} \\ u_0 = -\frac{4}{5} \end{cases}.$$

Відповідь:

i	0	1	2
u_i	$-0,8$	$-1,2$	$0,4$

.

Вправи для самостійної роботи

Методом механічних квадратур знайти наближення розв'язків заданих рівнянь:

1. $u - \frac{1}{6}\int\limits_0^2 xy^2 u\,dy = x+1, \quad x \in [0;2].$

2. $u - \frac{1}{10}\int\limits_0^4 (x+2y)u\,dy = x^2, \quad x \in [0;4].$

7.2.3. Метод заміни виродженим ядром

Нехай ядро K рівняння (7.2.1) є виродженим, тобто

$$K(x,y) = \sum_{i=1}^{m} A_i(x)B_i(y), \quad x,y \in [a,b], \tag{7.2.10}$$

де $m \in \mathbb{N}$, а $\{A_i\}_{i=1}^m$, $\{B_i\}_{i=1}^m$ — лінійно незалежні на $[a,b]$ функції. Тоді для розв'язку $u = u(x)$, $x \in [a,b]$, рівняння (7.2.1) маємо рівність

$$u(x) - \lambda \sum_{i=1}^{m} A_i(x) \int_a^b B_i(y)u(y)dy = f(x), \quad x \in [a,b]. \tag{7.2.11}$$

Якщо ввести позначення

$$C_i := \int_a^b B_i(y)u(y)dy, \quad i = \overline{1,m},$$

то отримаємо зображення розв'язку

$$u(x) = \lambda \sum_{i=1}^{m} C_i A_i(x) + f(x), \quad x \in [a,b], \tag{7.2.12}$$

де $C_1, \ldots, C_m$ — сталі, які мають такі значення, що функція u, визначена формулою (7.2.12), задовольняє рівність (7.2.11), тобто

$$\lambda \sum_{i=1}^{m} C_i A_i(x) + f(x) - \lambda \sum_{i=1}^{m} A_i(x) \int_a^b B_i(y) \left[\lambda \sum_{j=1}^{m} C_j A_j(y) + f(y)\right] dy = f(x), \quad x \in [a,b].$$

Звідси після скорочення (враховуємо, що $\lambda \neq 0$) і спрощення отримуємо

$$\sum_{i=1}^{m} \left[C_i - \lambda \sum_{j=1}^{m} \left(\int_a^b B_i(y)A_j(y)\,dy\right) C_j - \int_a^b B_i(y)f(y)dy\right] A_i(x) = 0, \quad x \in [a,b].$$

Звідси, врахувавши лінійну незалежність системи функцій $\{A_i\}_{i=1}^m$, отримаємо систему лінійних алгебраїчних рівнянь

$$C_i - \lambda \sum_{j=1}^{m} d_{ij} C_j = g_i, \quad i = \overline{1,m}, \tag{7.2.13}$$

де

$$d_{ij} := \int_a^b B_i(y)A_j(y)\,dy, \quad g_i := \int_a^b B_i(y)f(y)\,dy, \quad i,j = \overline{1,m}.$$

Систему (7.2.13) можна переписати у вигляді

$$\sum_{j=1}^{m} (\delta_{ij} - \lambda d_{ij}) C_j = g_i, \quad i = \overline{1,m}, \tag{7.2.14}$$

де $\delta_{ij} = \begin{cases} 1, & \text{якщо } i = j \\ 0, & \text{якщо } i \neq j \end{cases}$ — символ Кронеккера.

Якщо ввести позначення:

$$D(\lambda) := \{\delta_{ij} - \lambda d_{ij}\}_{i,j=\overline{1,m}} \quad \text{— основна матриця системи (7.2.14),}$$

то можемо зробити такий висновок: *коли $\lambda \in \mathbb{R}$ таке, що $\det D(\lambda) \neq 0$, то рівняння (7.2.1) має і тільки один розв'язок, причому він має вигляд (7.2.12), де $C_1, \ldots, C_m$ — єдиний розв'язок СЛАР (7.2.14).*

Якщо ядро K рівняння (7.2.1) є невиродженим, то можна наблизити його виродженим ядром і знайти наближення розв'язку рівняння (7.2.1) розв'язком рівняння з відповідним виродженим ядром. Є кілька способів апроксимувати дане ядро $K(x, y)$, $x, y \in [a, b]$, виродженим ядром $\sum_{i=1}^{m} A_i(x)B_i(y)$, $x, y \in [a, b]$:

1. Замінити ядро K його головною частиною розвинення цього ядра за формулою Тейлора або по одній зі змінних x чи y, або по обох. Наприклад, використати наближення вигляду

$$K(x, y) \approx K(x_*, y_*) + \sum_{0<i+j\leqslant m} \frac{\partial^{i+j} K(x_*, y_*)}{\partial x^i \partial y^j}(x - x_*)^i (y - y_*)^j, \quad (x, y) \in [a, b] \times [a, b], \tag{7.2.15}$$

де $m \in \mathbb{N}$, $(x_*, y_*) \in [a, b] \times [a, b]$.

2. Використати відомі ортогональні розвинення функцій.

3. Використати інтерполювання функції $K(x, y)$, $x, y \in [a, b]$, за одним або двома аргументами. Наприклад, використати наближення вигляду

$$K(x, y) \approx \sum_{i=1}^{m} \frac{(x - x_1)\ldots(x - x_{i-1})(x - x_{i+1})\ldots(x - x_m)}{(x_i - x_1)\ldots(x_i - x_{i-1})(x_i - x_{i+1})\ldots(x_i - x_m)} K(x_i, y), \quad x \in [a, b].$$

де $x_1, ..., x_m$ — вузли інтерполювання.

Розв'язати рівняння з виродженим ядром:

$$u - \int_0^1 xyudy = x + 1, \quad x \in [0, 1]. \tag{7.2.16}$$

Розв'язування. Нехай $u = u(x)$, $x \in [0, 1]$, — розв'язок даного рівняння. Тоді маємо рівність

$$u(x) - x \int_0^1 yu(y)dy = x + 1, \quad x \in [0, 1]. \tag{7.2.17}$$

Покладемо $C := \int_0^1 yu(y)dy$ і з (7.2.17) отримаємо зображення розв'язку рівняння (7.2.16)

$$u(x) = Cx + x + 1, \quad x \in [0, 1], \tag{7.2.18}$$

де значення сталої C таке, що функція u, визначена в (7.2.18), задовольняє рівність (7.2.17), тобто

$$Cx + x + 1 - x \int_0^1 y(Cy + y + 1)dy = x + 1, \quad x \in [0, 1],$$

звідси маємо

$$[C - \int_0^1 (Cy^2 + y^2 + y)dy]x = 0, \quad x \in [0, 1] \quad \Leftrightarrow \quad C - \left(C\frac{y^3}{3} + \frac{y^3}{3} + \frac{y^2}{2}\right)\Big|_0^1 = 0$$
$$\Leftrightarrow \quad C - \left(\frac{1}{3}C + \frac{1}{3} + \frac{1}{2}\right) = 0 \quad \Leftrightarrow \quad \frac{2}{3}C = \frac{5}{6} \quad \Leftrightarrow \quad C = \frac{5}{4}.$$

Підставивши знайдене значення C в (7.2.18), отримаємо

$$u = \frac{5}{4}x + x + 1 \equiv \frac{9}{4}x + 1, \quad x \in [0;1].$$

Методом заміни виродженим ядром розв'язати інтегральне рівняння:

$$u - \int_0^2 \ln(x+y+1) \cdot u\, dy = x^2, \quad x \in [0;2]. \tag{7.2.19}$$

Розв'язування. Маємо $K(x,y) = \ln(x+y+1)$, $x \in [0;2]$. Згідно з формулою Тейлора можна вважати, що

$$K(x,y) \approx K(0,0) + \frac{\partial K(0,0)}{\partial x}x + \frac{\partial K(0,0)}{\partial y}y, \quad x,y \in [0;2]. \tag{7.2.20}$$

Оскільки

$$\frac{\partial \ln(x+y+1)}{\partial x} = \frac{\partial \ln(x+y+1)}{\partial y} = \frac{1}{x+y+1},$$

то на підставі (7.2.20) матимемо

$$\ln(x+y+1) \approx x+y.$$

Звідси випливає, що розв'язок рівняння

$$u - \int_0^2 (x+y)u\, dy = x^2, \quad x \in [0;2]. \tag{7.2.21}$$

є наближенням розв'язку рівняння (7.2.19).

Нехай $u = u(x)$, $x \in [0;2]$, — розв'язок рівняння (7.2.21), тобто

$$u(x) - x\int_0^2 u(y)\, dy - \int_0^2 yu(y)\, dy = x^2, \quad x \in [0;2]. \tag{7.2.22}$$

Покладемо

$$C_1 := \int_0^2 u(y)\, dy, \quad C_2 = \int_0^2 yu(y)\, dy.$$

Тоді з (7.2.22) отримаємо

$$u(x) = C_1x + C_2 + x^2, \quad x \in [0;2]. \tag{7.2.23}$$

де C_1, C_2 — сталі такі, що функція, зображена в (7.2.23) задовольняє рівність (7.2.22), тобто

$$C_1x + C_2 + x^2 - x\int_0^2 (C_1y + C_2 + y^2)\, dy - \int_0^2 y(C_1y + C_2 + y^2)\, dy = x^2, \quad x \in [0;2].$$

Звідси випливає рівність

$$C_1x + C_2 + x^2 - x\Big(C_1\frac{y^2}{2} + C_2y + \frac{y^3}{3}\Big)\Big|_{y=0}^{y=2} - \Big(C_1\frac{y^3}{3} + C_2\frac{y^2}{2} + \frac{y^4}{4}\Big)\Big|_{y=0}^{y=2} = 0, \quad x \in [0;2].$$

яка рівносильна рівності

$$C_1x + C_2 + x^2 - x\Big(2C_1 + 2C_2 + \frac{8}{3}\Big) - \Big(\frac{8}{3}C_1 + 2C_2 + 4\Big) = 0 \quad \Leftrightarrow$$

$$\Leftrightarrow \quad -\Big(C_1 + 2C_2 + \frac{8}{3}\Big)x - \Big(\frac{8}{3}C_1 + C_2 + 4\Big) = 0, \quad x \in [0;1], \quad \Leftrightarrow$$

$$\begin{cases} C_1 + 2C_2 = -\frac{8}{3} \\ \frac{8}{3}C_1 + C_2 = -4 \end{cases},$$

$$\begin{cases} C_1 = -\frac{15}{13} \\ \\ C_2 = -\frac{28}{39} \end{cases}. \tag{7.2.24}$$

Підставивши значення C_1 і C_2 з (7.2.24) у (7.2.23) отримаємо розв'язок рівняння (7.2.21), який є наближенням розв'язку рівняння (7.2.19):

$$u = -\frac{15}{13}x - \frac{28}{39} + x^2, \quad x \in [0; 2].$$

Вправи для самостійної роботи

Методом заміни виродженим ядром знайти наближення розв'язків рівнянь:

1. $u - \int\limits_0^2 \ln(3x + 2y + 1) \cdot u\, dy = x + 1, \quad x \in [0; 2],$

2. $u - \int\limits_0^4 (e^{xy} + 2x + y - 1) \cdot u\, dy = 2x, \quad x \in [0; 4],$

використавши наближення ядра за формулою Тейлора:

$$K(x, y) \approx K(0, 0) + \frac{\partial K(0, 0)}{\partial x} x + \frac{\partial K(0, 0)}{\partial y} y, \quad x, y \in [a, b].$$

7.3. Лабораторний практикум з чисельного розв'язування інтегральних рівнянь

7.3.1 Метод механічних квадратур

Використаємо в (7.2.1) формулу чисельного інтегрування

$$\int_a^b g(y)\,dy \approx \sum_{j=0}^{n} A_j^{(n)} g(x_j) \tag{7.3.1}$$

де $n \in \mathbb{N}$ – довільне фіксоване число, $A_0^{(n)}, A_1^{(n)}, \ldots, A_n^{(n)}$ і $x_0, x_1, \ldots, x_n$ – відомі коефіцієнти і вузли квадратурної формули. Отримаємо:

$$u(x) - \lambda \left[\sum_{j=0}^{n} A_j^{(n)} K(x, x_j) u(x_j) \right] \approx f(x), \quad x \in [a, b] \tag{7.3.2}$$

Наближений розв'язок рівняння (7.2.1) розглядаємо в точках $x_0, x_1, \ldots, x_n$, а саме шукаємо наближення u_i, $i = \overline{0, n}$, значень розв'язку $u(x_i)$, $i = \overline{0, n}$.

Тоді, покладаючи почергово $x = x_i$, $i = \overline{0, n}$, з (7.3.2) отримаємо систему лінійних алгебричних рівнянь

$$u_i - \lambda \sum_{j=0}^{n} A_j^{(n)} K_{ij} u_j = f_i, \quad i = \overline{0, n}, \tag{7.3.3}$$

Далі для обчислень використовуватимемо квадратурні формули, вузли яких утворюють рівномірне розбиття проміжку інтегрування

$x_i := a + ih,\ i = \overline{0, n},\ h := \frac{b-a}{n}$.

Похибку чисельного розв'язку в точці x_i, $i = \overline{1, n}$, позначимо $\varepsilon_i := |u(x_i) - u_i|$.

Обчислювальні експерименти

Будемо використовувати велику квадратурну формулу трапецій.

Знаходження чисельних розв'язків рівняння (7.2.1) продемонструємо на таких прикладах:

Приклад 1. $K(x, y) = x + y, \quad f(x) = x^2, \quad [a, b] = [0,\ 2]$

Приклад 2. $K(x, y) = xy, \quad f(x) = x + 1, \quad [a, b] = [0,\ 1]$.

Можна перевірити, що функція $u(x) = 2.25x + 1$ є точним розв'язком рівняння (7.2.1) у цьому випадку.

Пояснення до використання програмного коду

- Підготувати потрібні функції :
  1. виконати комірку з імпортом бібліотек numpy, pandas і matplotlib
  2. виконати комірку з функцією `A`, яка задає коефіцієнти великої квадратурної формули трапецій
  3. виконати комірку з функцією `ie_MQuadrature`
  4. запрограмувати і виконати комірку з функцією `f`
  5. запрограмувати і виконати комірку з функцією `K`
- Для отримання наближення розв'язку треба виконати комірку з викликом функції `ie_MQuadrature` з відповідними аргументами цієї функції.

- Щоб переконатися, що наближення достатньо точне, можна виконати кілька послідовних викликів, збільшуючи значення параметра n.

[1]:
```
# Імпорт бібліотек
%matplotlib widget
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

[2]:
```
# Коефіцієнти великої квадратурної формули трапецій
def A(n):
    w=np.ones(n+1, dtype=float)
    w[0]=0.5
    w[n]=0.5
    return w
```

[3]:
```
def ie_MQuadrature(K,lmbd,a,b,n,A,f):
    """  метод механічних квадратур
         з рівномірним розбиттям проміжку інтегрування
    """
    x=np.linspace(a, b, n+1)
    c=f(x)
    h=(b-a)/n
    w=lmbd*h*A(n)

    B=np.zeros((n+1,n+1),dtype=float)
    for i in range(n+1):
        B[i,:]=-w*K(x[i],x)
        B[i,i]+=1

    u=np.linalg.solve(B, c)

    return u
```

1) Чисельне розв'язування прикладу 1

[4]:
```
def K1(xi,x):
    return x+xi
```

[5]:
```
def f1(x):
    return x**2
```

[6]:
```
lmbd=1
a=0
b=2
n=2
```

[7]:
```
u_0=ie_MQuadrature(K1,lmbd,a,b,n,A,f1)
```

[8]:
```
x_0=np.linspace(a, b, n+1)
df=pd.DataFrame({'x':x_0,'u':u_0})
df
```

[8]:

	x	u
0	0.0	-0.8
1	1.0	-1.2

```
2  2.0  0.4
```

2) Чисельне розв'язування прикладу 2

[9]:
```
def K2(xi,x):
    return x*xi
```

[10]:
```
def f2(x):
    return x+1
```

[11]:
```
lmbd=1
a=0
b=1
n=2
```

[12]:
```
u_0=ie_MQuadrature(K2,lmbd,a,b,n,A,f2)
```

[13]:
```
x_0=np.linspace(a, b, n+1)
df=pd.DataFrame({'x':x_0,'u':u_0})
df
```

[13]:
```
     x    u
0  0.0  1.0
1  0.5  2.2
2  1.0  3.4
```

Щоб продемонструвати збіжність чисельного розв'язку до аналітичного, розглянемо його на послідовності розбиттів, кожного разу вдвічі зменшуючи крок h. В програмі результат кожного `i`-того експерименту зберігаємо в масиві `u_i`.

[14]:
```
u_0=ie_MQuadrature(K2,lmbd,a,b,n,A,f2)

u_1=ie_MQuadrature(K2,lmbd,a,b,2*n,A,f2)

u_2=ie_MQuadrature(K2,lmbd,a,b,4*n,A,f2)

u_3=ie_MQuadrature(K2,lmbd,a,b,8*n,A,f2)

u_4=ie_MQuadrature(K2,lmbd,a,b,16*n,A,f2)
```

[15]:
```
x_0=np.linspace(a, b, n+1)
x=np.linspace(a, b, 64*n+1)
u=2.25*x+1
```

[16]:
```
df=pd.DataFrame({'x_0':x_0[::],'u_0':u_0[::],'u_1':u_1[::2],'u_2':u_2[::
 ↪4],'u_3':u_3[::8],'u_4':u_4[::16],'u':u[::64]})
df
```

[16]:
```
   x_0  u_0       u_1       u_2       u_3       u_4      u
0  0.0  1.0  1.000000  1.000000  1.000000  1.000000  1.000
1  0.5  2.2  2.142857  2.129412  2.126100  2.125275  2.125
2  1.0  3.4  3.285714  3.258824  3.252199  3.250549  3.250
```

Зауважимо, що використаний спосіб побудови табличок вимагає, щоб у кожній колонці була однакова кількість чисел.

Щоб отримати табличку з довшими колонками, опустимо колонку `u_0`:

[17]:
```
x_1=np.linspace(a, b, 2*n+1)
df1=pd.DataFrame({'x_1':x_1[::],'u_1':u_1[::],'u_2':u_2[::2],'u_3':u_3[::
↪4],'u_4':u_4[::8],'u':u[::32]})
df1
```

[17]:

	x_1	u_1	u_2	u_3	u_4	u
0	0.00	1.000000	1.000000	1.000000	1.000000	1.0000
1	0.25	1.571429	1.564706	1.563050	1.562637	1.5625
2	0.50	2.142857	2.129412	2.126100	2.125275	2.1250
3	0.75	2.714286	2.694118	2.689150	2.687912	2.6875
4	1.00	3.285714	3.258824	3.252199	3.250549	3.2500

Оскільки відомо точний розв'язок рівняння, то знайдемо похибку чисельного розв'язку на кожному розбитті та порядок збіжності.

[18]:
```
err=pd.DataFrame({'x_0':x_0})
err['e_0']=np.abs(df['u_0'] -df['u'])

err['e_1']=np.abs(df['u_1'] -df['u'])
err['r_1']=np.log2(err['e_0'] /err['e_1'])

err['e_2']=np.abs(df['u_2'] -df['u'])
err['r_2']=np.log2(err['e_1'] /err['e_2'])

err['e_3']=np.abs(df['u_3'] -df['u'])
err['r_3']=np.log2(err['e_2'] /err['e_3'])

err['e_4']=np.abs(df['u_4'] -df['u'])
err['r_4']=np.log2(err['e_3'] /err['e_4'])
err
```

[18]:

	x_0	e_0	e_1	r_1	e_2	r_2	e_3	r_3
0	0.0	0.000	0.000000	NaN	0.000000	NaN	0.000000	NaN
1	0.5	0.075	0.017857	2.070389	0.004412	2.017074	0.001100	2.004237
2	1.0	0.150	0.035714	2.070389	0.008824	2.017074	0.002199	2.004237

	e_4	r_4
0	0.000000	NaN
1	0.000275	2.001057
2	0.000549	2.001057

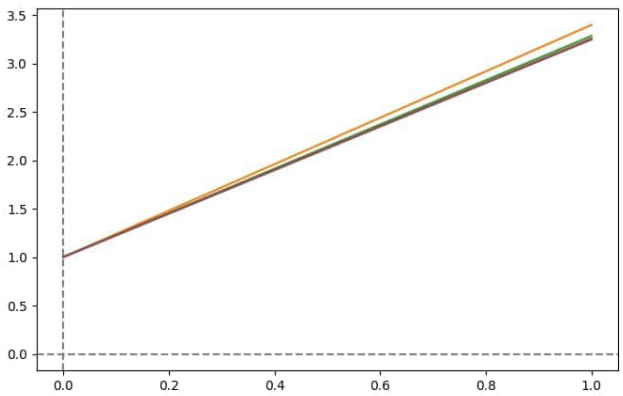
[19]:
```
err=pd.DataFrame({'x_1':x_1})

err['e_1']=np.abs(df['u_1'] -df['u'])
#err['r_1']=np.log2(err['e_0'] /err['e_1'])

err['e_2']=np.abs(df['u_2'] -df['u'])
err['r_2']=np.log2(err['e_1'] /err['e_2'])

err['e_3']=np.abs(df['u_3'] -df['u'])
err['r_3']=np.log2(err['e_2'] /err['e_3'])

err['e_4']=np.abs(df['u_4'] -df['u'])
err['r_4']=np.log2(err['e_3'] /err['e_4'])
err
```

[19]:
```
    x_1       e_1       e_2       r_2       e_3       r_3       e_4
↪r_4
0  0.00  0.000000  0.000000       NaN  0.000000       NaN  0.000000
↪NaN
1  0.25  0.017857  0.004412  2.017074  0.001100  2.004237  0.000275  2.
↪001057
2  0.50  0.035714  0.008824  2.017074  0.002199  2.004237  0.000549  2.
↪001057
3  0.75       NaN       NaN       NaN       NaN       NaN       NaN
↪NaN
4  1.00       NaN       NaN       NaN       NaN       NaN       NaN
↪NaN
```

[20]:
```
x_1=np.linspace(a, b, 2*n+1)

x_2=np.linspace(a, b, 4*n+1)

x_3=np.linspace(a, b, 8*n+1)

x_4=np.linspace(a, b, 16*n+1)
```

[21]:
```
plt.close('all')
```

[22]:
```
fig = plt.figure(figsize=(8, 5))
ax = fig.gca()
ax.axhline(color="grey", ls="--", zorder=-1)
ax.axvline(color="grey", ls="--", zorder=-1)

plt.plot(x, u)

plt.plot(x_0, u_0)

plt.plot(x_1, u_1)

plt.plot(x_2, u_2)

plt.plot(x_3, u_3)

plt.plot(x_4, u_4)
```



[23]:
```
plt.close('all')
```