МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
УМАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ САДІВНИЦТВА

В. Є. Березовський, Л. Є. Ковальов,
М. О. Медведєва

ЧИСЕЛЬНІ МЕТОДИ З ПРИКЛАДАМИ РЕАЛІЗАЦІЇ МОВОЮ PYTHON

Навчальний посібник

Рекомендовано Вченою радою Уманського НУС
як навчальний посібник для студентів, які навчаються за спеціальністю
122 «Комп'ютерні науки»

2023

УДК 519.6(075.8)
Б48

Рекомендовано Вченою радою Уманського НУС
як навчальний посібник для студентів, які навчаються за
спеціальністю 122 «Комп'ютерні науки»
(протокол № 3 від 22 грудня 2022 року)

Рецензенти:

Євтухов В. М., доктор фізико-математичних наук, професор, завідувач кафедри диференціальних рівнянь, геометрії та топології Одеського національного університету імені І. І. Мечнікова;

Годованюк Т. Л., доктор педагогічних наук, професор, проректор з наукової роботи, професор кафедри вищої математики та методики навчання математики Уманського державного педагогічного університету імені Павла Тичини.

Березовський В. Є.

Б48 Чисельні методи з прикладами реалізації мовою Python / В. Є. Березовський, Л. Є. Ковальов, М. О. Медведєва : навчальний посібник. Умань : ВПЦ «Візаві», 2023. 88 с.

У навчальному посібнику розглянуті основні традиційні питання курсу чисельних методів. Наведено багато прикладів розв'язання задач із застосуванням мови Python. Посібник містить достатню кількість завдань для самостійного розв'язання.

Для студентів, які навчаються за спеціальністю 122 «Комп'ютерні науки».

Посібник може бути використаний студентами, які навчаються за спеціальностями 014.04 «Середня освіта (Математика)», 014.08 «Середня освіта (Фізика)», 014.09 «Середня освіта (Інформатика)», при вивчені дисципліни «Методи обчислень».

УДК 519.6(075.8)

Зміст

Передмова

Широке впровадження математичних методів у різні сфери професійної діяльності людини вимагає створення і використання інструменту математичного моделювання для розв'язання обчислювальних задач. Сучасні успіхи в багатьох галузях науки і техніки неможливі були б без застосування ЕОМ і чисельних методів.

Навчальний посібник орієнтований на практичне закріплення теоретичного курсу з чисельних методів. Для основних задач чисельного аналізу розглядаються питання побудови і практичної реалізації обчислювальних алгоритмів, використання бібліотек чисельного аналізу.

Програмна реалізація розв'язання задач чисельними методами базується на алгоритмічній мові Python, яка останнім часом дуже активно розвивається. Ця високорівнева мова програмування загального призначення підтримується більшістю платформ і поширюється вільно. Приклади кодів, які розглянуті у посібнику, виконані у Python версії 3.9.

При установці Python на комп'ютер обов'язково треба встановити прапорець на «Add Python X.Y to PATH».

Для реалізації алгоритмів чисельних методів мовою Python необхідно буде використовувати деякі зовнішні пакети або модулі цієї мови. Наведемо приклади установки цих пакетів на платформі Windows.

Спочатку необхідно поновити менеджер пакетів pip. У командному рядку Windows вводиться команда:

python –m pip install –U pip

Далі встановлюються необхідні пакети командами:

python –m pip install –U numpy
python –m pip install –U scipy
python –m pip install –U matplotlib
python –m pip install –U lxml
python –m pip install –U sympy

(одночасно з пакетом sympy встановлюється пакет mpmath).

Розділ 1.

Вступ до чисельних методів

1.1. Обчислювальний експеримент

При розв'язанні складних природничо-наукових, інженерних та економічних задач використовують математичне моделювання. *Математична модель* задачі — це наближене подання реальної фізичної системи, об'єкту або процесу. Формулюються основні закони, які керують даним об'єктом дослідження. Ці закони як правило записуються у вигляді формул, систем рівнянь (алгебраїчних, диференціальних, інтегральних). При побудові математичної моделі нехтують факторами, які не впливають суттєво на хід досліджуваного процесу. На даний час побудова та аналіз математичних моделей здійснюється за допомогою коп'ютерних технологій. Такий метод дослідження називають *обчислювальним експериментом*.

Після того як задача сформульована у математичній формі, її необхідно розв'язати. Тільки у небагатьох випадках вдається отримати розв'язок у явному вигляді, аналітично. Іноді твердження, що «розв'язок отриманий» означає, що доведено існування та єдиність розв'язку. Цього недостатньо для практичних застосувань. Саме тут й виникає потреба використання комп'ютерних технологій. Під *чисельним методом* розуміють таку інтерпретацію математичної моделі, яка доступна для реалізації на комп'ютері. У такому разі говорять про перетворення математичної задачі в обчислювальну задачу. При цьому послідовність виконання необхідних арифметичних і логічних операцій визначається алгоритмом її розв'язання. Алгоритм повинен бути рекурсивним і складатися з відносно невеликих блоків, які багаторазово виконуються для різних вхідних даних.

Для того щоб реалізувати чисельний метод, необхідно скласти програму для комп'ютера. Після відлагодження програми проводяться обчислення і аналіз результатів.

Отже, схема обчислювального експерименту така: *математична модель — метод (алгоритм) — програма*.

Предметом даного навчального посібника є один з етапів обчислювального експерименту, а саме етап побудови і дослідження чисельного методу. Тут не обговорюються вихідні задачі та їх математичне подання, але частково розглядається реалізація чисельних методів мовою Pathon.

1.2. Елементи теорії похибок

Процес дослідження вихідного об'єкту методом математичного моделювання та обчислювального експерименту неминуче має наближений характер, тому що на кожному етапі вносяться ті чи інші похибки.

Побудова математичної моделі пов'язана зі спрощенням вихідного об'єкту. Крім того, параметри, які входять до опису задачі, задані наближено. По відношенню до чисельного методу, який реалізує дану математичну модель, вказані похибки є *неусувними*, оскільки вони неминучі у межах даної моделі.

При переході від математичної моделі до чисельного методу виникають похибки, які називаються *похибками методу*. Найбільш типовими похибками методу є *похибка дискретизації* та *похибка округлення*. Побудова чисельного методу для заданої математичної моделі полягає у формулюванні дискретної задачі та розробки обчислювального алгоритму, який дозволяє відшукати розв'язок дискретної задачі. Зрозуміло, що розв'язок дискретної задачі відрізняється від розв'язку вихідної задачі. При введенні, обчисленнях та виведенні даних здійснюються округлення чисел. У процесі роботи алгоритму похибки округлення звичайно накопичуються. Похибку округлення іноді називають *обчислювальною похибкою*.

Отже, слід розрізняти похибки моделі, методу та обчислювальну. У загальному випадку слід намагатися, щоб всі вказані похибки мали один порядок.

Одним з джерел обчислювальних похибок є наближене зображення дійсних чисел в ЕОМ, яке обумовлено скінченністю розрядної сітки.

Незважаючи на те, що вихідні дані зображаються в ЕОМ з високою точністю, накопичення похибок заокруглення у процесі розрахунків може привести до значної кінцевої похибки, а деякі алгоритми можуть виявитись й зовсім непридатними для реальних розрахунків на ЕОМ.

Наближеним значенням деякої величини X називають число a, яке неістотно відрізняється від точного значення цієї величини x.

В ЕОМ найчастіше використовується зображення чисел у формі з плаваючою крапкою:

$$a = Mr^p, \tag{1.2.1}$$

де r — основа системи числення, p — ціле число, яке називають *порядком числа* a, M — *мантиса числа* a, яка у нормалізованій формі відповідає нерівності

$$1 \leqslant |M| < r^1. \tag{1.2.2}$$

Наприклад, для зображення типу числа «float» у мові Python (відповідає типу числа «double» у мовах C, C++, Java) відводиться вісім байт (64 біта):

$$(-1)^s \cdot 1.a_0a_1 \ldots a_{51} \cdot 2^{p-1023}. \tag{1.2.3}$$

Двійкові розряди розподілені так: s — старший 63-й біт, мантиса займає 52 біта від 0-го до 51-го, порядок p — 11 біт від 52-го до 62-го, а зсув -1023 у степені зроблений для того, щоб порядок завжди був невід'ємним.

Можна легко знайти діапазон чисел типу «float». Мінімальне число дорівнює $2^{-1023} \approx 2.225 \cdot 10^{-308}$, а максимальне — $(1 + (1 - 2^{-52})) \cdot 2^{1023} \approx 1.797 \cdot 10^{308}$.

Мінімальне додатне число M_0, яке може бути зображено в ЕОМ з плаваючою точкою, називають *машинним нулем*. Число $M_\infty = M_0^{-1}$ називають *машинною нескінченністю*.

Абсолютною похибкою Δx наближеного значення величини X називають модуль різниці між точним й наближеним значеннями цієї величини:

$$\Delta x = |a - x|. \tag{1.2.4}$$

Відносною похибкою наближеного значення називають відношення абсолютної похибки цього значення до абсолютної величини точного значення:

$$\delta = \frac{|a - x|}{|x|} = \frac{\Delta x}{|x|}. \tag{1.2.5}$$

Точне значення величини x звичайно є невідомим. Тому наведені вирази (1.2.4), (1.2.5) для похибок не можуть бути використані. Ми маємо тільки наближене значення a, і потрібно знайти його *граничну абсолютну похибку* Δa, яка є верхньою оцінкою модуля абсолютної похибки. Тобто $\Delta x \leqslant \Delta a$. Відносну похибку можна оцінити так

$$\delta = \frac{\Delta a}{|a|}. \tag{1.2.6}$$

Відносна точність зображення чисел з плаваючою крапкою визначається кількістю розрядів t, які відводяться для запису мантиси

$$\frac{\Delta a}{|a|} \leqslant 2^{-t}, \tag{1.2.7}$$

де число 2^{-t} називають *машинним епсілоном*.

Значущими цифрами у запису наближеного числа називають: всі ненульові цифри; нулі, які містяться між ненульовими цифрами; нулі, які записані в кінці після округлення.

Перші (зліва направо) n значущі цифри у запису наближеного числа називають вірними, якщо абсолютна похибка числа не перебільшує половини одиниці розряду, що відповідає n-й значущій цифрі.

Наприклад, $x = 1.80352 \pm 0.00007$. Вірними є перші чотири значущі цифри, 5 і 2 не задовольняють визначенню. Якщо $x = 1.80352 \pm 0.00004$, то вірними будуть перші п'ять значущих цифр.

Наступні правила округлення чисел гарантують, що збережені значущу цифри будуть вірними.

Для того, щоб округлити число до n значущих цифр, відкидають всі цифри, які стоять справа від n-ої значущої цифри, або, якщо це потрібно для збереження розрядів, заміняють їх нулями. при цьому:

1) якщо перша відкинута цифра менша 5, то десяткові знаки, які залишаються, зберігаються без зміни;
2) якщо перша відкинута цифра більша 5, то до останньої цифри, що залишається додають одиницю;
3) якщо перша відкинута цифра дорівнює 5 та серед відкинутих цифр є ненульові, то до останньої цифри, що залишається додають одиницю;
4) якщо перша відкинута цифра дорівнює 5 та всі відкинуті цифри є нулями, то остання цифра, що залишається, не змінюється, якщо вона парна, та збільшується на одиницю, якщо вона непарна (правило парної цифри).

Правило парної цифри повинно забезпечити компенсацію знаків похибок.

Приклад 1.1. $3.1415926 \approx 3.142$*;*

$4\,258\,250 \approx 4\,258\,000$*;*

$4\,258\,250 \approx 4\,258\,200$*;*

$2.997925 \cdot 10^8 \approx 2.998 \cdot 10^8$.

Для наближеного числа, яке отримане після округлення, абсолютна похибка Δa приймається рівною половині одиниці останнього розряду числа. Наприклад, для значення $a = 1.843$ абсолютна похибка $\Delta a = 0.0005$.

При зміні форми запису числа кількість значущих цифр не повинна змінюватись. Наприклад, записи $5800 = 5.800 \cdot 10^3$ та $1.320 \cdot 10^2 = 132.0$ рівнозначні, а записи $5800 = 5.8 \cdot 10^3$ та $1.320 \cdot 10^2 = 132$ нерівнозначні.

При виконанні операцій над наближеними числами граничні похибки оцінюються за певними правилами.

При додаванні і відніманні чисел їх абсолютні похибки додаються. При множені або ділені чисел одне на друге їх відносні похибки додаються. При піднесені до степеня наближеного числа його відносна похибка множиться на показник степеня.

Для двох наближених чисел a і b ці правила можна записати у вигляді формул:

$$\Delta(a \pm b) = \Delta a + \Delta b, \qquad \delta(a \cdot b) = \delta a + \delta b,$$
$$\delta(a/b) = \delta a + \delta b, \qquad \delta(a^k) = k\delta a. \tag{1.2.8}$$

При обчисленні функцій, аргументами яких є наближені числа, для оцінки похибок користуються правилом, яке побудоване на обчислені приросту (похибки) функції при заданих приростах (похибках) аргументів.

Розглянемо функцію однієї змінної $y = f(x)$. Нехай a — наближене значення аргументу x, Δa — його абсолютна похибка. Тоді оцінка абсолютної похибки функції виконується за формулою

$$\Delta y = |f'(a)|\Delta a, \tag{1.2.9}$$

де $f'(a)$ — значення похідної функції при аргументі a.

Аналогічний вираз можна записати для функції декількох аргументів. Наприклад, оцінка абсолютної похибки функції $u = f(x, y, z)$, наближені значення аргументів якої відповідно a, b і c, має вигляд

$$\Delta u = |f'_x(a, b, c)|\Delta a + |f'_y(a, b, c)|\Delta b + |f'_z(a, b, c)|\Delta c. \tag{1.2.10}$$

Розглянемо деякі випадки, коли можна уникнути втрати точності за рахунок правильної організації обчислень.

Для відносної похибки різниці двох чисел маємо

$$\delta(a - b) = \frac{\Delta a + \Delta b}{|a - b|}. \tag{1.2.11}$$

При $a \approx b$ ця похибка може бути дуже великою.

Наприклад, нехай $a = 1847$, $b = 1845$. У цьому випадку маємо абсолютні похибки вихідних даних $\Delta a = \Delta b = 0.5$ і відносні похибки $\delta a \approx \delta b = 0.5/1845 \approx 0.0003\,(0.03\%)$. Відносна похибка різниці дорівнює

$$\delta(a-b) = \frac{0.5+0.5}{2} = 0.5\,(50\%).$$

При малих похибках у вихідних даних ми отримали досить неточний результат. При організації обчислювальних алгоритмів слід уникати віднімання близьких чисел; при можливості алгоритм необхідно змінити.

Розглянемо розв'язування квадратного рівняння

$$ax^2 + bx + c = 0.$$

Його корені визначаються формулами

$$x_1 = \frac{-b - \sqrt{D}}{2a}, \qquad x_2 = \frac{-b + \sqrt{D}}{2a}, \qquad D = b^2 - 4ac. \tag{1.2.12}$$

Розглянемо випадок, коли коефіцієнт b значно перевищує за абсолютною величиною інші $b^2 \gg 4ac$. Тоді виникає небезпечність віднімання близьких чисел у чисельнику одного з виразів (1.2.12) внаслідок того, що $\sqrt{D} \approx |b|$.

Існує декілька виходів з цієї ситуації. Найбільш універсальним є використання значення $\operatorname{sign} b$ («знак величини b»):

$$\operatorname{sign} b = \begin{cases} 1, & b \geqslant 0, \\ -1, & b < 0. \end{cases}$$

Тоді один з коренів може бути обчислений за формулою

$$x_1 = -\frac{b + \operatorname{sign} b \cdot \sqrt{D}}{2a}\,. \tag{1.2.13}$$

Вираз для другого кореня можна отримати з представлення квадратного рівняння у вигляді

$$ax^2 + bx + c = a(x - x_1)(x - x_2) = ax^2 - ax(x_1 + x_2) + ax_1x_2.$$

Прирівнюючи вільні члени, отримуємо

$$x_2 = \frac{\mathrm{c}}{ax_1}\,. \tag{1.2.14}$$

Розглянемо ще приклад зниження похибки шляхом покращення алгоритму. Наприклад, при обчислені значення $(a+x)^2$ величина x може виявитись такою, що результатом додавання $a+x$ буде a (при $x \ll a$); у цьому випадку може допомогти формула скороченого множення $(a+x)^2 = a^2 + 2ax + x^2$.

Ще один важливий приклад — використання рядів для обчислення значень функцій. Відомо, що

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

Якщо значення x буде лежати в межах від 0 до $\pi/2$, то ми при обчисленні будемо мати гарну точність. Але якщо x буде приймати великі значення, то похибка обчислень буде великою (наприклад, при $x = 6.8$ відносна похибка буде складати 10%). При збільшені за абсолютною величиною аргументу для заданої точності необхідно використовувати в розкладанні велику кількість доданків, що призводить до значних похибок.

У алгоритмах, які використовують степеневі ряди для обчислення значень функцій, можуть бути застосовані різні способи для запобігання втрати точності. Так, для тригонометричних функцій можна використати формули зведення. При обчисленні експоненти аргумент x можна розбити на суму цілої та дробової частин ($e^x = e^{n+a} = e^n e^a$) та використовувати розкладання у ряд тільки для e^a, а e^n обчислювати множенням.

Практичний інтерес мають оцінки остаточної похибки в залежності від кількості виконаних арифметичних дій n.

Для добутку n дійсних чисел відносна похибка оцінюється приблизно як $n2^{-t}$, де t — кількість розрядів, які відводяться для запису мантиси.

Може виявитись, що на якомусь етапі обчислень в якості проміжного результату буде отриманий або машинний нуль M_0, або машинна нескінченність M_∞.

Для того щоб уникнути такої ситуації, у випадку довільної кількості k співмножників можна запропонувати наступний алгоритм обчислення добутку. Припустимо, що

$$|x_1| \leqslant |x_2| \leqslant \dots \leqslant |x_k|,$$

причому $|x_1| \leqslant 1$, $|x_k| \geqslant 1$. Будемо спочатку проводити множення в порядку $x_1 x_k x_{k-1} \dots$ до тих пір, доки вперше не отримуємо число, яке більше одиниці. Потім отриманий частковий добуток будемо послідовно множити на x_2, x_3 і т. д. до тих пір, доки новий частковий добуток не стане меншим

одиниці. Процес повторюється до тих пір, доки всі співмножники, які залишились, не будуть або тільки більшими одиниці за модулем, або тільки меншими. У подальшому множення проводиться у довільному порядку.

Для суми n дійсних чисел відносна похибка оцінюється наближено як $n^2 2^{-t}$.

При додаванні втрата точності обчислень може з'явитись внаслідок того, що до великого числа додаються малі числа. Цих малих чисел може бути багато, але на результат вони не будуть впливати, оскільки додаються по одному. Тут необхідно дотримуватись правила, у відповідності з яким додавання чисел необхідно проводити по мірі їх зростання. В комп'ютерній арифметиці внаслідок похибки округлення суттєвим є порядок виконання операцій, та відомі з алгебри закони комутативності і дистрибутивності тут не завжди виконуються.

Розділ 2.

Чисельне розв'язання нелінійних рівнянь

2.1. Постановка задачі. Відокремлення коренів

Розглянемо рівняння

$$f(x) = 0, \tag{2.1.1}$$

де $f(x)$ — деяка неперервна функція.

Якщо $f(x)$ — алгебраїчний багаточлен, то рівняння (2.1.1) називають *алгебраїчним*; якщо $f(x)$ містить спеціальні математичні функції (наприклад, тригонометричні, показникові, логарифмічні тощо), то рівняння називають *трансцендентним*.

Значення змінних ξ_j (де $j = 1, 2, 3, \ldots$), за яких виконується рівняння $f(\xi_j) = 0$, називають *нулями* функції $f(x)$ або *розв'язками рівняння* (2.1.1). Рівняння (2.1.1) може мати один, декілько або не мати жодного розв'язків.

Чисельні методи розв'язання нелінійних рівнянь є, як правило, ітераційними методами, які передбачають задання достатньо близьких до шуканого розв'язку початкових даних.

Проміжок $[a, b]$, на якому є один і лише один розв'язок ξ рівняння (2.1.1), називають *проміжком ізоляції*, а процес його знаходження — *відокремленням кореня*.

Якщо функція $f(x)$ заздалегідь відома, то найбільш ефективним способом відокремлення кореня є *графічний метод*. В інших випадках, коли проміжок $[a, b]$ потрібно знайти автоматично (не візуально), то застосовують *табличний* (*аналітичний*) *метод*.

Приклад 2.1. *Відокремити графічним методом корінь рівняння* $2x^3 - 17x + 8 = 0$.

Розв'язання. Реалізуємо графічний метод відокремлення кореня мовою Python. Існує декілько можливостей побудови графіків мовою Python. Скористаємось бібліотекою наукової графіки matplotlib. Вихідний код поданий у лістингу 1, а результат на рис. 2.1, з якого видно, що корені рівняння знаходяться на проміжках $[-4; -3]$, $[0; 1]$ і $[2; 3]$

Лістинг 1. *Реалізація графічного методу відокремлення кореня мовою Python*

```
import numpy as np
import matplotlib.pyplot as plt
y = lambda x: 2*x**3-17*x+8
x = np.linspace(-4,4,100)
plt.plot(x, y(x), color='k', linewidth = 2.5)
plt.plot([-4,4],[0,0], color='k', linewidth = 0.8)
plt.ylabel('f(x)')
plt.xlabel('x')
plt.grid()
plt.savefig('graf_met_vidokr.pdf',bbox_inches='tight')
plt.show()
```

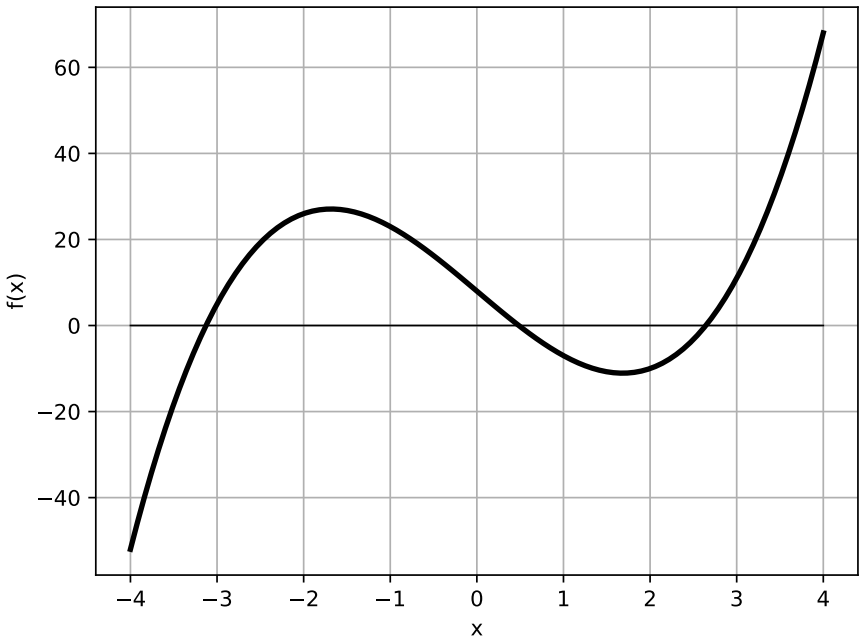

Рис. 2.1. Графік функції $f(x) = 2x^3 - 17x + 8$

При застосуванні в Pyton математичних функцій до елементів списку, кортежу або масиву необхідно попередньо провести їх векторизацію за

допомогою функції Numpy vectorize (приклад на лістингу 1_a та реалізація на рис. 2.2).

Лістинг 1_a. *Реалізація графічного методу відокремлення кореня мовою Python*

```
import math
import numpy as np
import matplotlib.pyplot as plt
def f(x):
    return math.tan(0.84*x+0.5)-x**2
x = np.linspace(-8,8,201)
y = np.vectorize(f)
plt.plot(x, y(x), color='k', linewidth = 2.5)
plt.ylabel('f(x)')
plt.xlabel('x')
plt.grid()
plt.show()
```

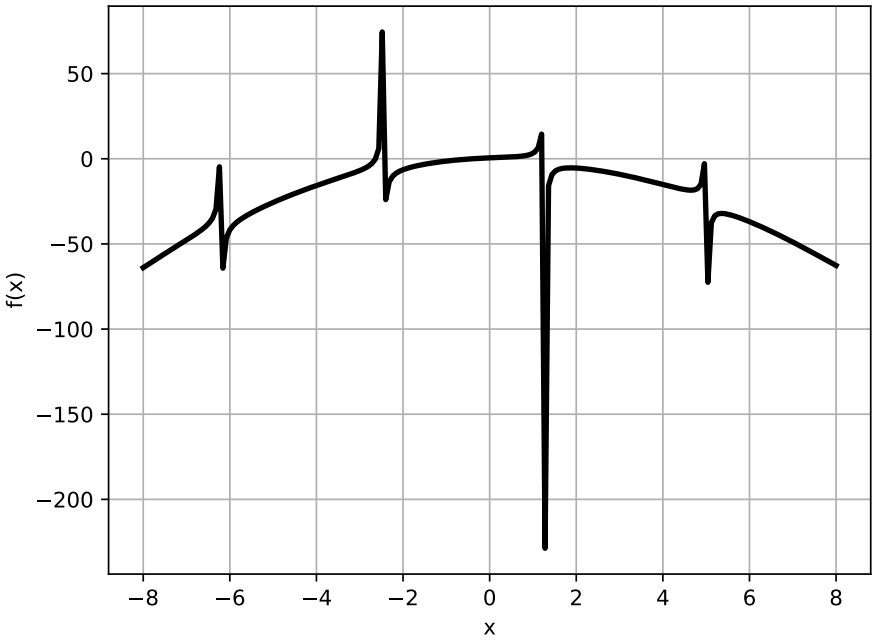

Рис. 2.2. Графік функції $f(x) = \mathrm{tg}(0.84x + 0.5) - x^2$

Для визначання проміжків ізоляції аналітичним методом використовується теорема, відома з курсу математики.

Теорема 2.1. *Якщо функція $f(x)$ є неперервною на відрізку $[a, b]$ і на його кінцях набуває значення протилежних знаків, тобто $f(a) \cdot f(b) <$*

0, *то всередині цього проміжку існує хоча б один розв'язок рівняння* $f(x) = 0$. *Якщо, окрім цього, перша похідна* $f'(x)$ *на цьому проміжку зберігає знак, то цей розв'язок буде єдиним.*

Можна запропонувати наступний алгоритм відокремлення кореня аналітичним методом:

1) визначаємо граничні точки $x = a$ та $x = b$ з області визначення функції $f(x)$;
2) на $[a; b]$ через проміжки достатньо малої довжини h обчислюємо значення $f(x_j)$ $(j = 0, 1, 2, \ldots, n;\ x_0 = a)$;
3) якщо добуток $f(x_j)f(x_{j+1})$ додатній, то можна сподіватися, що на відрізку $[x_j; x_{j+1}]$ коренів рівняння немає;
4) якщо добуток $f(x_j)f(x_{j+1})$ від'ємний і знак похідної $f'(x)$ на кінцях відрізку $[x_j; x_{j+1}]$ не змінюється, то можна сподіватися, що на відрізку корінь один; таким чином, корінь рівняння відокремлений;
5) якщо знак похідної змінюється, то коренів може бути більше одного і відповідний відрізок $[x_j; x_{j+1}]$ розбивається на менші кроки та повторюється описана процедура.

Приклад 2.2. *Відокремити аналітичним методом корінь рівняння* $2x^3 - 17x + 8 = 0$. *Реалізувати алгоритм мовою Python.*

Розв'язання. Задаємо граничні точки значень x та крок табуляції h. Результат зберігаємо у списку pr.

У циклі for для створення списку ми використали функцію arange пакету Numpy, яка на відміну від стандартної функції range, дозволяє працювати з дійсними числами.

Лістинг 2. *Реалізація аналітичного методу відокремлення кореня мовою Python*

```
import numpy as np
def f(x):
    return 2*x**3-17*x+8
a = -4
b = 4
h = 1
pr =[]
for x in np.arange(a,b,h):
    if f(x)*f(x+h) < 0:
        pr.append([x,x+h])
```

```
if len(pr) > 0:
    print('Відрізки ізоляції',pr)
else:
    print('У діапазоні',[a,b],'коренів немає')
```

Результат виконання програми:

```
Відрізки ізоляції [[-4, -3], [0, 1], [2, 3]]
```

Після відокремлення коренів нелінійного рівняння, кожен з них уточнюється із заданою точністю. Для цього використовуються ітераційні методи. *Ітераційний процес* складається з послідовності уточнення початкового наближення x_0. Кожен такий крок називають *ітерацією*. В результаті ітерацій знаходиться послідовність наближених значень кореня $x_1, x_2, \ldots, x_n$. Якщо ці значення із зростанням n наближаються до істинного значення кореня ξ, то кажуть, що ітераційний процес сходиться.

2.2. Метод половинного ділення (метод бісекції)

Метод половинного ділення, який ще називають *методом бісекції*, безпосередньо випливає з аналітичного методу відокремлення кореня.

Нехай для рівняння $f(x) = 0$ знайдений первинний відрізок $[a, b]$ ізоляції кореня. Обчислимо середину відрізку $c = \dfrac{a+b}{2}$. Якщо випадково виявиться, що $f(c) = 0$, то c є кореням рівняння. Якщо ж $f(c) \neq 0$, то далі обираємо ту з половин відрізку $[a, b]$, на кінцях якого функція $f(x)$ має протилежні знаки.

Обраний відрізок позначимо $[a_1, b_1]$, знову поділимо навпіл і виконаємо дії, аналогійні до попередніх. Унаслідок виконання таких кроків матимемо послідовність вкладених відрізків: $[a_1, b_1], [a_2, b_2], \ldots, [a_n, b_n]$. Отже, n-е наближення розв'язку x_n рівняння перебуває на проміжку $[a_n, b_n]$. Оскільки довжина n-го відрізку d_n дорівнює

$$d_n = b_n - a_n = \frac{b-a}{2^n} \to 0$$

та при $n \to \infty$ послідовність $a_1, a_2, \ldots, a_n$ є монотонно неспадною, а послідовність $b_1, b_2, \ldots, b_n$ — монотонно незростаючою. Ці послідовності мають спільну границю:

$$\xi = \lim_{n\to\infty} a_n = \lim_{n\to\infty} b_n. \qquad (2.2.1)$$

Нескладно переконатись, що границя (2.2.1) є розв'язком рівняння $f(x) = 0$.

Якщо процес ділення навпіл зупинити на n-му кроці, то за наближене значення розв'язку рівняння можна обрати значення

$$x_n = \frac{a_n + b_n}{2}. \tag{2.2.2}$$

Абсолютна похибка

$$\varepsilon = |\xi - x_n| \leqslant \frac{b-a}{2^{n+1}}. \tag{2.2.3}$$

Метод половинного ділення доволі повільний, але він завжди збігається. Тобто при його використанні розв'язок отримується завжди, причому із заданою точністю.

Приклад 2.3. *Розв'язати нелінійне рівняння* $e^{2x} + 3x - 4 = 0$ *методом бісекції з точністю* $\varepsilon = 0.001$.

Розв'язання. Розв'язок реалізуємо мовою Python. Опишемо дві функції: $f(x)$, яка відповідає лівій частині заданого рівняння і $bisection(a, b, \varepsilon)$, в якій за допомогою циклу реалізований метод половинного ділення (лістинг 3). Запропонований алгоритм передбачає, що значення $[a, b]$ є результатом відокремлення кореня і не потребують перевірки (відокремлення кореня для заданого рівняння пропонуємо виконати самостійно). Для перевірки умови досягнення заданої точності обчислюється довжина відрізку $[a, b]$ на кожній ітерації та порівнюється з ε.

Лістинг 3. *Реалізація алгоритму методу половинного ділення мовою Python*

```
import math
def f(x):
    return math.exp(2*x) + 3*x - 4
def bisection(a,b,eps):
    counter = 0
    while (b - a > eps):
        c = (a + b) / 2.0
        if (f(b) * f(c) < 0):
            a = c
        else:
            b = c
        print("{:7d} {:7.4f} {:7.4f} {:7.4f} \
```

```
{:7.4f} {:7.4f} {:7.4f}"
                .format(counter,a,b,f(a),f(b),c,f(c)))
            counter += 1
    print("{:}{:7.4f} {:}{:7.4f}"
          .format('корінь =',(a+b)/2,'    f =',f((a+b)/2)))
print("{:8} {:7} {:7} {:7} {:7} {:7} {:7}".format \
      ('counter','a','b','f(a)','f(b)','c','f(c)'))
bisection(0.4,0.6,0.001)
```

Результат виконання кожної ітерації виводиться на друк (при зменшені ε необхідно виводити значення аргументів і функції з більшою кількістю значущих цифр після крапки):

```
counter  a       b       f(a)    f(b)    c       f(c)
      0  0.4000  0.5000 -0.5745  0.2183  0.5000  0.2183
      1  0.4500  0.5000 -0.1904  0.2183  0.4500 -0.1904
      2  0.4500  0.4750 -0.1904  0.0107  0.4750  0.0107
      3  0.4625  0.4750 -0.0906  0.0107  0.4625 -0.0906
      4  0.4688  0.4750 -0.0402  0.0107  0.4688 -0.0402
      5  0.4719  0.4750 -0.0148  0.0107  0.4719 -0.0148
      6  0.4734  0.4750 -0.0020  0.0107  0.4734 -0.0020
      7  0.4734  0.4742 -0.0020  0.0043  0.4742  0.0043
корінь = 0.4738    f = 0.0011
```

2.3. Метод Ньютона (метод дотичних)

Нехай ми знайшли відрізок відокремлення кореня $[a, b]$. Виберемо на ньому початкове наближення x_0. Замінимо функцію $f(x)$ відрізком ряду Тейлора

$$f(x) \approx H_1 = f(x_0) + (x - x_0)f'(x_0)$$

та за наступне наближення x_1 візьмемо корінь рівняння $H_1 = 0$, тобто

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}\,.$$

Взагалі, якщо ітерація x_k відома, то наступне наближення x_{k+1} у *методі Ньютона* визначається за правилом

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}\,, \quad k = 0,\, 1,\, 2,\, \ldots \tag{2.3.1}$$

Метод Ньютона називають також *методом дотичних*, так як нове наближення x_{k+1} є абсцисою точки перетину дотичної, яка проведена у точці $(x_k,\, f(x_k))$ до графіку функції $f(x)$, з віссю Ox (рис. 2.3).

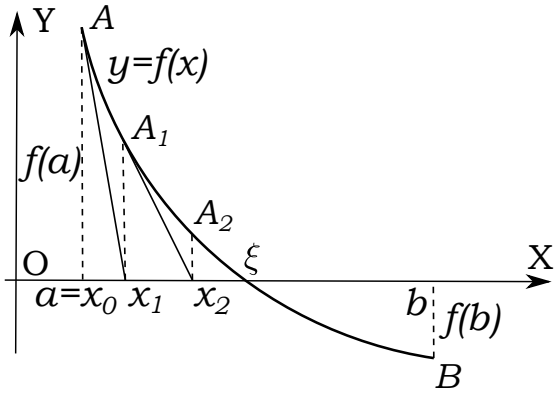

Рис. 2.3. Геометрична інтерпретація розв'язання нелінійних рівнянь методом Ньютона

Метод Ньютона має квадратичну збіжність, тобто на відміну від лінійних задач похибка на наступній ітерації пропорційна квадрату похибки на попередній ітерації.

Однак така швидка збіжність методу Ньютона гарантована лише при дуже гарних, тобто близьких до точного рішення, початкових наближеннях. Якщо початкове наближення вибране невдало, то метод може збігатись повільно, або не збігається взагалі.

За початкове наближення x_0 обирають той кінець відрізка $[a,\, b]$, у якому знак функції співпадає зі знаком другої похідної

$$x_0 = \begin{cases} a, & f(a)f''(a) > 0, \\ b, & f(b)f''(b) > 0. \end{cases} \tag{2.3.2}$$

Крім того при здійснені обчислювального алгоритму можуть виникнути певні проблеми, які будуть розглянуті у прикладі реалізації методу Ньютона мовою Python.

Модифікований метод Ньютона

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_0)}, \quad k = 0,\, 1,\, 2,\, \ldots \tag{2.3.3}$$

застосовують у тому випадку, коли бажають уникнути багаторазових обчислень похідної $f'(x)$. Метод (2.3.3) пред'являє менше вимог до вибору початкового наближення x_0 і дозоляє уникнути можливості ділення на нуль при якійсь ітерації, однак має лише лінійну збіжність.

Приклад 2.4. *Знайти розв'язок нелінійне рівняння* $e^{2x} + 3x - 4 = 0$ *із заданою похибкою* $\varepsilon = 1.0 \cdot 10^{-7}$ *алгоритмічною мовою Python методом Ньютона.*

Розв'язання. Реалізацію метода Ньютона мовою Python можна зробити дуже просто. Але проста реалізація в загальному випадку може привести до деяких похибок, які необхідно буде відшукувати. Основні проблеми простого алгоритму методу Ньютона: функція $f(x)$ викликається у два рази більше ніж це потрібно; при деякій ітерації може виникнути ділення на нуль; при дуже невдалому виборі початкового значення метод буде розбіжним.

У наведеному прикладі реалізації методу Ньютона мовою Python задається максимальна кількість ітерацій для випадку розбіжності алгоритму, обробляється ділення на нуль і забирається зайвий виклик функції $f(x)$.

У вихідному коді крім функції $f(x)$ описується й перша похідна $f'(x)$. Для того, щоб уникнути цілого числа при діленні функції на її похідну, значення похідної описується як тип float.

Лістинг 4. *Реалізація алгоритму методу Ньютона мовою Python*

```
import math
def newton(f, x, df, eps=1.0e-7, N=100):
    f_value = f(x)
    n = 0
    iterats = [(x, f_value)]
    while abs(f_value) > eps and n <= N:
        df_value = float(df(x))
        if abs(df_value) < 1e-14:
            raise ValueError("'f(%g)=%g" % (x, df_value))
        x = x - f_value/df_value
        n += 1
        f_value = f(x)
        iterats.append((x, f_value))
    return x, iterats
def g(x):
    return math.exp(2*x) + 3*x - 4
def dg(x):
    return 2*math.exp(2*x) + 3
x0 = 0.6
x,iterats = newton(g, x0, dg)
```

```
print('root:', x)
for i in range(len(iterats)):
    print('Iteration %3d: f(%g)=%g' % \
          (i, iterats[i][0], iterats[i][1]))
```

В результаті виконання програми виводиться знайдений корінь та інформація по кожній ітерації:

```
root: 0.47368829057502904
Iteration   0: f(0.6)=1.12012
Iteration   1: f(0.483808)=0.0830881
Iteration   2: f(0.473753)=0.000528593
Iteration   3: f(0.473688)=2.16534e-08
```

2.4. Метод хорд

Метод хорд ще називають *методом січних*. Цей метод отримують з методу Ньютона (2.3.1) заміною похідної $f'(x)$ виразом

$$\frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}},$$

який обчислюється за відомими значеннями x_k і x_{k-1}. В результаті отримуємо ітераційний метод

$$x_{k+1} = x_k - \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} f(x_k), \quad k = 0, 1, 2, \ldots \tag{2.4.1}$$

який на відміну від раніше розглянутих методів є *двокроковим*, тобто нове наближення x_{k+1} визначається двома попередніми ітераціями x_k і x_{k-1}. В методі (2.4.1) необхідно задавати два початкових наближення x_0 і x_1.

Геометрична інтерпретація методу січних полягає у наступному. Через точки $(x_{k-1}, f(x_{k-1}))$, $(x_k, f(x_k))$ проводять хорду, абсциса точки перетину якої з віссю Ox і є новим наближенням x_{k+1}. Інакше кажучи, на відрізку $[x_{k-1}, x_k]$ функція $f(x)$ *інтерполюється* багаточленом першого степеня і за наступне наближення x_{k+1} приймається корінь цього багаточлена.

2.5. Метод простої ітерації

Для використання методу простої ітерації вихідне нелінійне рівняння (2.1.1) записується у вигляді

$$x = \varphi(x)$$

та ітерації виконуються за правилом

$$x_{k+1} = \varphi(x_k), \quad k = 0, 1, 2, \ldots,$$

при заданому початковому наближенні x_0.

Для збіжності методу простої ітерації велике значення має вибір функції $\varphi(x)$. Цю функцію можна задавати різними способами, але звичайно вона береться у вигляді

$$\varphi(x) = x + s(x)f(x), \tag{2.5.1}$$

причому функція $s(x)$ не змінює знак на тому відрізку, де відшукується корінь. Метод простої ітерації збіжний при вдалому виборі початкового наближення x_0, якщо на проміжку ізоляції $|\varphi'(x)| < 1$.

Фактично у формі методу простої ітерації (2.5.1) можна записати будь-який однокроковий ітераційний метод.

Зокрема, якщо $s(x) = -s = \text{const}$, то отримуємо *метод релаксації*

$$x_{k+1} = x_k - sf(x_k). \tag{2.5.2}$$

Можна показати, що оптимальним значенням параметра s буде

$$s = \frac{2}{M + m}, \tag{2.5.3}$$

де M та m — відповідно найбільше та найменше значення модуля похідної $|f'(x)|$ на проміжку ізоляції.

Якщо $f'(x)$ має постійний знак на проміжку $[a, b]$, то її значення за абсолютною величиною на кінцях проміжку можна вважати за найбільше та найменше, тобто як M та m.

Слід зауважити, що параметр s має той же знак, що й похідна $f'(x)$ на проміжку $[a, b]$.

Приклад 2.5. *Знайти розв'язок нелінійне рівняння* $e^{2x} + 3x - 4 = 0$ *із заданою похибкою* $\varepsilon = 1.0 \cdot 10^{-7}$ *методом простої ітерації (релаксації).*

Розв'язання. Проміжок ізоляції кореня $[0.4, 0.6]$. Похідна функції $f'(x) = 2e^{2x} + 3$. Похідна на проміжку ізоляції додатна і на його кінцях приймає значення $m = 7.45108$, $M = 9.64023$. Тоді

$$s = \frac{2}{M + m} = 0.117$$

та ітераційна формула

$$x_{k+1} = x_k - 0.117(e^{2x} + 3x - 4).$$

Метод релаксації реалізувати мовою Python не складно. Скористаємось двома змінними xt, xf, які відповідають двом сусіднім наближенням x_{k+1}, x_k. Критерій виходу з ітераційного процесу використаємо наступний: $|x_{k+1} - x_k| < \varepsilon$, початкове наближення $x_0 = 0.4$.

Лістинг 4. *Реалізація алгоритму методу простої ітерації (релаксації) мовою Python*

```
import math
def f(x):
    return math.exp(2*x) + 3*x - 4
def prosti_iter(f,x0,s,eps):
    def fi(x):
        return s*f(x)
    counter = 0
    xf = x0
    xt = xf - fi(xf)
    while (abs(xt-xf) > eps):
        xf = xt
        xt = xf - fi(xf)
        print("%d          %.8f    %.8f" \
          %(counter,xt,f(xt)))
        counter += 1
print('iteration','x','f(x)', sep='        ')
prosti_iter(f,0.4,0.117,1.0e-7)
```

В результаті отримуємо

```
iteration       x        f(x)
0          0.47336821   -0.00261062
1          0.47367365   -0.00011939
2          0.47368762   -0.00000544
3          0.47368826   -0.00000025
4          0.47368829   -0.00000001
```

Порівняння кореня зі значенням отриманим методом Ньютона свідчить про успішне досягнення заданої точності.

Завдання для самостійного розв'язання

Виконати:

а) відокремити корені рівняння графічним методом та уточнити один з них методом бісекції і методом Ньютона з похибкою $\varepsilon = 1.0 \cdot 10^{-5}$;

б) відокремити корені рівняння аналітичним методом та уточнити один з них методом простої ітерації (релаксації) і методом Ньютона з похибкою $\varepsilon = 1.0 \cdot 10^{-5}$.

1. а) $2.2x - 2^x = 0$ б) $x^3 - 0.2x^2 + 0.5x - 1 = 0$
2. а) $x^2 + 4\sin x = 0$ б) $x^3 - 0.1x^2 + 0.4x + 1.2 = 0$
3. а) $5x - 8\ln x = 8$ б) $x^3 - 0.2x^2 + 0.5x - 1.4 = 0$
4. а) $3x - e^x = 0$ б) $x^3 + 2x + 4 = 0$
5. а) $x(x+1)^2 = 1$ б) $x^3 - 3x^2 + 12x - 12 = 0$
6. а) $x = (x+1)^3$ б) $x^3 + 0.2x^2 + 0.5x + 0.8 = 0$
7. а) $x^2 = \sin x$ б) $x^3 + 4x - 6 = 0$
8. а) $x^3 = \sin x$ б) $x^3 + 0.1x^2 + 0.4x - 1.2 = 0$
9. а) $x = \sqrt{\lg(x+2)}$ б) $x^3 + 3x^2 + 6x - 1 = 0$
10. а) $x^2 = \ln(x+1)$ б) $x^3 - 0.1x^2 + 0.4x - 1.5 = 0$
11. а) $2x + \cos x = 0.5$ б) $x^3 - 0.2x^2 + 0.3x - 1.2 = 0$
12. а) $\sin(0.5 + x) = 2x - 0.5$ б) $x^3 + 3x + 1 = 0$
13. а) $2\sin(x - 0.6) = 1.5 - x$ б) $x^3 - 4x^2 + 2 = 0$
14. а) $x + \cos x = 1$ б) $x^3 - 0.1x^2 + 0.3x - 0.6 = 0$
15. а) $\ln x + (x+1)^3 = 0$ б) $x^3 + 2x^2 + 2 = 0$
16. а) $x \cdot 2^x = 1$ б) $x^3 - 3x^2 + 9x - 10 = 0$
17. а) $\sqrt{x+1} = x^{-1}$ б) $x^3 - 2x + 2 = 0$
18. а) $x - \cos x = 0$ б) $x^3 + 3x - 1 = 0$
19. а) $3x + \cos x + 1 = 0$ б) $x^3 + x - 3 = 0$
20. а) $x + \ln x = 0.5$ б) $x^3 + 0.4x^2 + 0.6x - 1.6 = 0$
21. а) $2 - x = \ln x$ б) $x^3 - 0.2x^2 + 0.9x + 1 = 0$
22. а) $(x-1)^2 = 0.5e^x$ б) $x^3 - 0.1x^2 + 0.4x + 2 = 0$
23. а) $(2-x)e^x = 0.5$ б) $x^3 + 3x^2 + 12x + 3 = 0$
24. а) $x - \sin x = 0.25$ б) $x^3 - 3x^2 + 9x - 8 = 0$
25. а) $\sqrt{x} - \cos(0.387x) = 0$ б) $x^3 - 3x^2 + 6x + 3 = 0$
26. а) $\operatorname{tg}(0.4x + 0.4) = x^2$ б) $x^3 - 12x - 10 = 0$
27. а) $3x - \cos x - 1 = 0$ б) $x^3 - 3x^2 + 9x + 2 = 0$
28. а) $\operatorname{ctg} 1.05x - x^2 = 0$ б) $x^3 - 2x + 4 = 0$
29. а) $\operatorname{ctg} x - 0.25x = 0$ б) $x^3 - 3x^2 + 3.5 = 0$
30. а) $\operatorname{tg}(0.44x + 0.3) = x^2$ б) $x^3 - 3x^2 - 24x - 8 = 0$

Розділ 3.

Методи розв'язання систем лінійних алгебраїчних рівнянь

3.1. Основні поняття

Розв'язання систем лінійних алгебраїчних рівнянь (СЛАР) є однією із самих поширених і важливих задач обчислювальної математики. Вирішення багатьох проблем у різних галузях науки й техніки (економічні і технічні задачі, алгоритми математичної фізики і обчислювальної математики, обробка результатів експериментальних досліджень, тощо) зводиться до розв'язання лінійних систем.

У загальному випадку кількість рівнянь може не збігатися з числом невідомих, але ми будемо розглядати тільки ті СЛАР, у яких ця умова виконується. Запишемо систему m лінійних алгебраїчних рівнянь з m невідомими

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1m}x_m = b_1, \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2m}x_m = b_2, \\ \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mm}x_m = b_m. \end{cases} \tag{3.1.1}$$

Сукупність коефіцієнтів цієї системи запишемо у вигляді *квадратної матриці* порядку m:

$$A = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1m} \\ a_{21} & a_{22} & \ldots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mm} \end{pmatrix}. \tag{3.1.2}$$

Систему рівнянь (3.1.1) можна записати у матричному вигляді:

$$AX = B, \tag{3.1.3}$$

де X і B — вектор-стовпець невідомих і вектор-стовпець вільних коефіцієнтів відповідно:

$$X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix}, \qquad B = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}. \tag{3.1.4}$$

При роз'язанні лінійних систем використовуються деякі спеціальні матриці. *Одинична матриця* — квадратна матриця, елементи головної діагоналі якої дорівнюють одиниці, а інші елементи рівні нулю. *Верхня трикутна матриця* — квадратна матриця, в якій всі елементи нижче за головну діагональ дорівнюють нулю. *Нижня трикутна матриця* — квадратна матриця, в якій всі елементи вище за головну діагональ дорівнюють нулю.

Визначником (*детермінантом*) матриці A m-го порядку називають число D ($\det A$), яке дорівнює алгебраїчній сумі $n!$ добутків n-го порядку елементів цієї матриці, в кожен з яких входить по одному і тільки по одному елементу з кожного рядка і кожного стовпця даної матриці:

$$D = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mm} \end{vmatrix} = \sum (-1)^k a_{1\alpha} a_{2\beta} \dots a_{m\omega}\,, \tag{3.1.5}$$

де індекси α, β, …, ω пробігають всі можливі $n!$ перестановок номерів $1, 2, \dots, m$; k — число інверсій, які утворюють другі індекси елементів у даній перестановці.

Для того щоб система (3.1.1) мала єдиний розв'язок, необхідно й достатньо, щоб $\det A \neq 0$. У випадку рівності нулю визначника системи матрицю називають *сингулярною* (*виродженою* або *особливою*). При цьому система або не має розв'язків, або має їх нескінченну множину.

Позначимо через Δ_j *кутовий мінор* порядку j матриці A, тобто мінор, який розміщений на перетині j перших рядків і j перших стовпців матриці A:

$$\Delta_1 = a_{11}, \quad \Delta_2 = \det \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \dots, \quad \Delta_m = \det A. \tag{3.1.6}$$

Чисельні методи розв'язання систем лінійних алгебраїчних рівнянь поділяють на прямі (точні) та ітераційні (наближені). *Прямими методами* називають методи, які в припущенні, що обчислення ведуться точно (без

округлень), приводять за скінчене число арифметичних дій до точних значень ξ_j. Оскільки обчислення на комп'ютері ведуться з округленнями, то розв'язок неминуче міститиме похибки, які накопичуються при великій кількості обчислювальних операцій. До прямих методів відносяться, наприклад, метод Крамера, метод Гауса і його модифікації та ін.

Ітераційні методи — це методи послідовних наближень. В них необхідно задати деякий наближений розв'язок (початкове наближення). Після цього за допомогою деякого алгоритму проводять цикл обчислень (ітерацію). В результаті ітерації знаходять нове наближення. Ітерації проводять до отримання розв'язку з необхідною точністю. До наближених методів відносяться, наприклад, метод Якобі, метод Зейделя та ін.

Алгоритми розв'язання лінійних систем з використанням ітераційних методів звичайно більш складні у порівнянні з прямими методами, але у багатьох випадках вони виявляються кращими. Ітераційні методи вимагають зберігання у пам'яті ЕОМ не всієї матриці системи, а лише деяких векторів з m компонентами. Похибки кінцевих результатів при використанні ітераційних методів не накопичуються, оскільки точність обчислень у кожній ітерації визначається лише результатами попередньої ітерації та практично не залежить від раніше виконаних обчислень.

Ітераційні методи можуть бути використані для уточнення розв'язків, які отримані за допомогою прямих методів. Такі комбіновані алгоритми в багатьох випадках доволі ефективні.

3.2. Прямі методи

Одним із способів розв'язання системи лінійних рівнянь є *правило Крамера*, згідно з яким кожне невідоме подається у вигляді відношення визначників:

$$x_j = \frac{\det(A_j)}{\det(A)}, \qquad j = 1, 2, \ldots, m, \tag{3.2.1}$$

де матриця A_j утворюється з матриці A заміною її j-го стовпця стовпцем вільних коефіцієнтів B.

Але такий спосіб розв'язання системи лінійних рівнянь з m невідомими призводить до обчислення $(m+1)$-го визначника порядку m, що є дуже трудомісткою операцією при великих значеннях m.

У лінійній алгебрі зазвичай використовують спосіб розв'язування системи рівнянь на підставі матричної форми (3.1.3), який ґрунтується на

обчисленні *оберненої матриці* A^{-1}, при умові що $\det(A) \neq 0$:

$$X = \mathrm{A}^{-1}B. \tag{3.2.2}$$

Метод оберненої матриці також потребує значної кількості операцій. Метод Крамера і метод оберненої матриці не застосовуються при $m > 5$.

Найбільш поширеним серед прямих методів розв'язання систем лінійних алгебраїчних рівнянь є *метод Гауса*. Цей метод, заснований на послідовному виключенні невідомих, має безліч модифікацій. Розходження між ними — у порядку виключення невідомих і в способі збереження проміжних результатів. У загальному випадку метод Гауса полягає в приведенні системи (3.1.1) до системи трикутного вигляду, у якій останнє рівняння містить тільки одну змінну (коефіцієнти при інших змінних дорівнюють 0), а всі інші рівняння знизу догори будуть містити на одну змінну більше, ніж попереднє рівняння.

Метод Гауса можна трактувати наступним чином. Спочатку виконується розкладання матриці A у добуток двох трикутних матриць $A = LU$. Потім послідовно розв'язують дві системи рівнянь

$$LY = B, \tag{3.2.3}$$

$$UX = Y \tag{3.2.4}$$

з трикутними матрицями, звідки й знаходиться шуканий вектор X. Розкладання $A = LU$ відповідає прямому ходу методу Гауса, а розв'язання системи (3.2.3)–(3.2.4) — зворотному ходу.

Теоретичне обґрунтування можливості розкладання матриці у добуток двох трикутних матриць містить наступна теорема

Теорема 3.1 (теорема про LU-розкладання)**.** *Нехай всі кутові мінори матриці A відмінні від нуля, $\Delta_j \neq 0$, $j = 1, 2, \ldots, m$. Тоді матрицю A можна представити, причому єдиним способом, у вигляді добутку*

$$A = LU, \tag{3.2.5}$$

де L — нижня трикутна матриця з ненульовими діагональними елементами і U — верхня трикутна матриця з одиничною діагоналлю.

Незважаючи на те, що метод Гауса вимагає набагато менше арифметичних операцій ніж попередні прямі методи, він все рівно може приводити до великих похибок на матрицях навіть розмірностей порядку $m = 50 \div 100$.

Приклад 3.1. *Знайти розв'язок системи рівнянь методами Крамера, оберненої матриці і Гауса*

$$\begin{cases} 15x_1 + 25x_2 + 35x_3 = 12, \\ 9x_1 + 8x_2 + 7x_3 = 13, \\ 9x_1 + 6x_2 + 5x_3 = 7. \end{cases}$$

Розв'язання. При реалізації методу Крамера мовою Python ми скористалися функцією linalg.det(a) пакету NumPy, яка обчислює визначник матриці.

Лістинг 5. *Реалізація методу Крамера мовою Python*

```
import numpy as np
def kram(A,B):
    m = len(A)
    D = np.linalg.det(A)
    r = list()
    for i in range(m):
        Dj = np.copy(A)
        Dj[:,i] = B
        r.append(np.linalg.det(Dj)/D)
    return r
A = [[15.,25.,35.],[9.,8.,7.],[9.,6.,5.]]
B = [12.,13.,7.]
X = kram(A,B)
print('X =',X)
```

Результат:

```
X = [-1.64761904761905, 6.828571428571435,
                                -3.8285714285714296]
```

Лістинг 6. *Реалізація методу оберненої матриці мовою Python*

```
import numpy as np
A = [[15,25,35],[9,8,7],[9,6,5]]
B = [12,13,7]
OA = np.linalg.inv(A)
X = np.matmul(OA,B)
print('X =',X)
```

Результат:

```
X = [-1.64761905  6.82857143 -3.82857143]
```

При реалізації методу оберненої матриці використовувалась функція linalg.inv(a) пакету NumPy, яка визначає обернену матрицю. Крім того для добутку матриць у мові Python не можна використовувати символ «*». Для знаходження добутку матриці на вектор ми скористались функцією matmul(a,b) того ж пакету NumPy.

Реалізувати метод Гауса з використанням LU-розкладання можна декількома способами: скористатись функцією linalg.solve(a, b) пакету NumPy або пакету SciPy; скористатись ланцюжком пакету SciPy linalg.lu_factor() і linalg.lu_solve(), який дає доступ до розкладання; створити власні функції розкладання і розв'язання.

Ми обрали перший варіант і показали як можна перевірити розв'язок за допомогою функції allclose().

Лістинг 7. *Реалізація методу Гауса мовою Python*

```
import numpy as np
A = np.array([[15,25,35],
              [9,8,7],
              [9,6,5]])
B = np.array([12,13,7])
X = np.linalg.solve(A,B)
print('X =',X)
print('Перевірка:',np.allclose(A @ X - B, np.zeros((3,))))
```

Результат:

```
X = [-1.64761905  6.82857143 -3.82857143]
Перевірка: True
```

Наведемо ще приклад розв'язання в Python системи лінійних алгебраїчних рівнянь за допомогою бібліотеки символьної математики SymPy. Якщо всі коефіцієнти і вільні члени системи — цілі числа, то розв'язок в пакеті SymPy буде точним.

Лістинг 8. *Символьне розв'язання системи рівнянь мовою Python*

```
import sympy as smp
x1, x2, x3 = smp.symbols('x1 x2 x3')
slr = [
```

```
    15*x1 + 25*x2 + 35*x3 - 12,
    9*x1 + 8*x2 + 7*x3 - 13,
    9*x1 + 6*x2 + 5*x3 - 7
]
rozv = smp.linsolve(slr, x1, x2, x3)
print('x1, x2, x3:', rozv)
```

Результат:

```
x1, x2, x3: FiniteSet((-173/105, 239/35, -134/35))
```

Зверніть увагу на те, що на початку потрібно об'явити символьні змінні та рівняння записуються у формі, в якій вільні коефіцієнти дорівнюють нулю.

3.3. Ітераційні методи

Розглянемо два приклади ітераційних методів: метод Якобі (або метод простої ітерації) та його модифікацію — метод Зейделя (або метод Гауса–Зейделя).

Перетворимо систему (3.1.1) до вигляду:

$$x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j - \sum_{j=i+1}^{m} a_{ij}x_j\right), \qquad i = 1, 2, \ldots, m, \tag{3.3.1}$$

де припускається, що всі a_{ii} відмінні від нуля.

Наведемо приклад системи з трьох рівнянь:

$$\begin{aligned} x_1 &= \frac{1}{a_{11}}(b_1 - a_{12}x_2 - a_{13}x_3),\\ x_2 &= \frac{1}{a_{22}}(b_2 - a_{21}x_1 - a_{23}x_3),\\ x_3 &= \frac{1}{a_{33}}(b_3 - a_{31}x_1 - a_{32}x_2). \end{aligned}$$

Домовимося, що верхній індекс буде вказувати номер ітерації, наприклад

$$X^k = (x_1^k, x_2^k, \ldots, x_m^k)^{\mathsf{T}},$$

де x_i^k — k-а ітерація i-ої компоненти вектора X.

В *методі Якобі* виходять із запису системи (3.3.1) та ітерації визначаються наступним чином:

$$x_i^{k+1} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^k - \sum_{j=i+1}^{m} a_{ij}x_j^k\right), \tag{3.3.2}$$
$$k = 0, 1, \ldots, k_{max}, \qquad i = 1, 2, \ldots, m.$$

Початкові значення x_i^0, $i = 1, 2, \ldots, m$ задаються довільно. Закінчення ітерацій визначається або заданням максимальної кількості ітерацій k_{max}, або умовою

$$\max_{1 \leqslant i \leqslant m} |x_i^{k+1} - x_i^k| < \varepsilon, \tag{3.3.3}$$

де $\varepsilon > 0$ — задана похибка розв'язку системи рівнянь.

Для збіжності ітераційного процесу достатньо, щоб виконувалася умова домінування діагональних елементів матриці A (модулі діагональних коефіцієнтів для кожного рівняння системи (3.1.1) повинні бути не меншими сум моделей всіх інших коефіцієнтів):

$$|a_{ii}| \geqslant \sum_{i \neq j} |a_{ij}|, \qquad i = 1, 2, \ldots, m. \tag{3.3.4}$$

Вказані умови хоча б для одного рівняння повинні бути строгими. Ці умови є достатніми для збіжності методу, але вони не є необхідними. Тобто для деяких систем ітерації збігаються й при порушені умов (3.3.4).

В методі Якобі при обчислені x_i^{k+1} не використовується інформація, яка отримана в останній момент. *Метод Зейделя* полягає у використовуванні уточнених значень змінних вже на поточному ітераційному кроці. Для підрахунку x_i^{k+1} застосовуються нові значення x_1^{k+1}, x_2^{k+1}, $\ldots$, x_{i-1}^{k+1} і старі значення x_{i+1}^k, x_{i+2}^k, $\ldots$, x_m^k.

Ітераційна формула методу Зейделя має вигляд:

$$x_i^{k+1} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{k+1} - \sum_{j=i+1}^{m} a_{ij}x_j^k\right), \tag{3.3.5}$$
$$k = 0, 1, \ldots, k_{max}, \qquad i = 1, 2, \ldots, m.$$

Зауважимо, що умови завершення процесу розв'язання системи лінійних рівнянь та умови збіжності для методу Зейделя є такі самі, як і для методу Якобі (формули (3.3.3), (3.3.4)), але на практиці швидкість його збігу дещо вища. Тому саме цей метод застосовують найчастіше.

Приклад 3.2. *Знайти розв'язок системи лінійних алгебраїчних рівнянь ітераційними методами*

$$\begin{cases} 15x_1 + 25x_2 + 35x_3 = 12\,, \\ 9x_1 + 8x_2 + 7x_3 = 13\,, \\ 9x_1 + 6x_2 + 5x_3 = 7\,. \end{cases}$$

Розв'язання. Достатня умова збіжності (3.3.4) не виконується:

$$\begin{aligned} |a_{11}| = 15 &< |a_{12}| + |a_{13}| = 60\,, \\ |a_{22}| = 8 &< |a_{21}| + |a_{23}| = 16\,, \\ |a_{33}| = 5 &< |a_{31}| + |a_{32}| = 15\,. \end{aligned}$$

Приведемо систему рівнянь до вигляду, який придатний для застосування методів Якобі і Зейделя. Від першого рівняння віднімемо друге та поставимо результат на місце третього рівняння в системі. Від другого рівняння віднімемо третє та залишимо результат на місці другого рівняння. Третє рівняння помножимо на два та віднімемо від нього друге рівняння. Це буде перше рівняння нової системи. Отже, ми отримали систему рівнянь, для якої виконується достатня умова збіжності:

$$\begin{cases} 9x_1 + 4x_2 + 3x_3 = 1\,, \\ 0x_1 + 2x_2 + 2x_3 = 6\,, \\ 6x_1 + 17x_2 + 28x_3 = -1\,. \end{cases}$$

Методи Якобі і Зейделя розв'язання останньої системи рівнянь реалізуємо мовою Python в одному коді. Ми описали функцію $zeidel(a, b, eps = 1e-9)$, в якій три вхідних параметри: матриця a, вектор вільних членів b і задана похибка eps (за умовчуванням дорівнює $1 \cdot 10^{-9}$). Перша сума $s1$ в коді відповідає методу Якобі, а друга $s1$ — методу Зейделя. Ітераційний процес завершується або при досягненні заданої точності, або при виконанні 1000 ітерацій.

Лістинг 9. *Реалізація методів Якобі і Зейделя розв'язання системи рівнянь мовою Python*

```
import numpy as np
def zeidel(a, b, eps=1e-9):
    m = len(b)
```

```
    x = np.zeros(m)
    d = np.zeros(m)
    iteration = 0
    the_end = False
    while not the_end:
        x_new = np.copy(x)
        for i in range(m):
#            s1 = sum(A[i][j]*x[j] for j in range(i))
            s1 = sum(A[i][j]*x_new[j] for j in range(i))
            s2 = sum(A[i][j]*x[j] for j in range(i+1,m))
            x_new[i] = (b[i] - s1 - s2) / A[i][i]
            d[i] = np.fabs(x_new[i] - x[i])
        p = np.max(d)
        iteration += 1
        x = x_new
        the_end = p <= eps
        if iteration == 1000: the_end = True
    return x,iteration,p
A = np.array([[9,4,3],
              [0,2,2],
              [6,17,28]])
B = np.array([1,6,-1])
x,it,p = zeidel(A, B)
print('iteration:',it)
print('X =',x)
print('Похибка =',p)
```

Результат методу Якобі:

```
iteration: 176
X = [-1.64761905  6.82857143 -3.82857143]
Похибка = 8.987868227450235e-10
```

Результат методу Зейделя:

```
iteration: 32
X = [-1.64761905  6.82857143 -3.82857143]
Похибка = 6.097025107010268e-10
```

Завдання для самостійного розв'язання

Виконати:
розв'язати систему рівнянь будь-яким прямим методом і методом Зейделя з похибкою $1 \cdot 10^{-7}$. При розв'язанні методом Зейделя перевірити систему на збіжність.

1. $\begin{cases} -0.18x_1 + 0.48x_2 - 0.21x_4 = -1.17, \\ 0.93x_1 + 0.08x_2 - 0.11x_3 + 0.18x_4 = 0.51, \\ -0.13x_1 - 0.31x_2 + x_3 + 0.21x_4 = 1.02, \\ -0.08x_1 + 0.33x_3 + 0.72x_4 = 0.28. \end{cases}$

2. $\begin{cases} 0.95x_1 + 0.06x_2 + 0.12x_3 - 0.14x_4 = 2.17, \\ -0.34x_1 - 0.08x_2 + 1.06x_3 - 0.14x_4 = 2.1, \\ -0.04x_1 + 1.12x_2 - 0.08x_3 - 0.11x_4 = -1.4, \\ -0.11x_1 - 0.12x_2 + 1.03x_4 = 0.8. \end{cases}$

3. $\begin{cases} 0.92x_1 + 0.03x_2 + 0.04x_4 - 1.2 = 0, \\ 0.69x_2 - 0.27x_3 + 0.08x_4 + 0.81 = 0, \\ 0.11x_1 - 0.03x_3 + 0.42x_4 + 0.17 = 0, \\ -0.33x_1 + 1.07x_3 - 0.21x_4 - 0.92 = 0. \end{cases}$

4. $\begin{cases} -0.12x_1 + 0.05x_2 + 0.85x_4 = 0.57, \\ -0.14x_2 + 0.66x_2 + 0.18x_3 - 0.24x_4 = 0.89, \\ -0.33x_1 - 0.03x_2 + 0.84x_3 + 0.32x_4 = -1.15, \\ 0.88x_1 + 0.23x_2 - 0.25x_3 + 0.16x_4 = -1.24. \end{cases}$

5. $\begin{cases} 0.25x_1 + 0.22x_2 + 0.14x_3 - x_4 = -1.56, \\ -0.08x_1 + 0.12x_2 + 0.77x_3 - 0.32x_4 = -0.58, \\ -0.12x_1 + x_2 - 0.32x_3 + 0.18x_4 = -0.72, \\ 0.77x_1 + 0.14x_2 - 0.06x_3 + 0.12x_4 = 1.21. \end{cases}$

6. $\begin{cases} 0.16x_1 + 0.24x_2 - x_3 - 0.35x_4 = -1.21, \\ 0.12x_1 - 1.14x_2 + 0.08x_3 + 0.09x_4 = 0.83, \\ 0.23x_1 - 0.08x_2 + 0.05x_3 - 0.75x_4 = -0.65, \\ -0.86x_1 + 0.23x_2 + 0.18x_3 + 0.17x_4 = 1.42. \end{cases}$

7. $$\begin{cases} 0.35x_1 - 0.27x_2 - x_3 - 0.05x_4 = -0.68, \\ -0.76x_1 + 0.21x_2 + 0.06x_3 - 0.34x_4 = -1.42, \\ -0.05x_1 + x_2 - 0.32x_3 - 0.12x_4 = -0.57, \\ 0.12x_1 - 0.43x_2 + 0.04x_3 - 1.21x_4 = 2.14. \end{cases}$$

8. $$\begin{cases} 0.13x_1 - 1.12x_2 + 0.09x_3 - 0.06x_4 = -0.48, \\ -0.83x_1 + 0.27x_2 - 0.13x_3 - 0.11x_4 = 1.42, \\ 0.13x_1 + 0.18x_2 + 0.24x_3 - 0.57x_4 = -0.72, \\ -0.11x_1 - 0.05x_2 + 1.02x_3 - 0.12x_4 = -2.34. \end{cases}$$

9. $$\begin{cases} 0.17x_1 + 0.06x_2 - 1.08x_3 + 0.12x_4 = -1.15, \\ -0.85x_1 + 0.05x_2 - 0.08x_3 + 0.14x_4 = 0.48, \\ -0.32x_1 + 1.13x_2 + 0.12x_3 - 0.11x_4 = 1.24, \\ 0.21x_1 - 0.16x_2 + 0.36x_3 - x_4 = 0.88. \end{cases}$$

10. $$\begin{cases} 0.11x_1 + 0.22x_2 + 0.03x_3 - 0.95x_4 + 0.72 = 0, \\ -x_1 + 0.28x_2 - 0.17x_3 + 0.06x_4 + 0.21 = 0, \\ 0.52x_1 - x_2 + 0.12x_3 + 0.17x_4 - 1.17 = 0, \\ 0.17x_1 - 0.18x_2 - 0.79x_3 - 0.81 = 0. \end{cases}$$

11. $$\begin{cases} -0.07x_1 + 1.38x_2 + 0.05x_3 - 0.41x_4 = 1.8, \\ 0.48x_1 - 0.08x_2 - 0.13x_4 = -0.22, \\ -0.17x_1 - 0.18x_2 + 0.13x_3 + 0.81x_4 = 0.33, \\ -0.04x_1 - 0.42x_2 + 0.89x_3 + 0.07x_4 = -1.3. \end{cases}$$

12. $$\begin{cases} 0.19x_1 - 0.23x_2 + 0.08x_3 - 0.63x_4 = -1.5, \\ -0.99x_1 + 0.02x_2 - 0.62x_3 + 0.08x_4 = 1.3, \\ 0.03x_1 - 0.72x_2 + 0.33x_3 - 0.07x_4 = -1.1, \\ 0.09x_1 + 0.13x_2 - 0.58x_3 + 0.28x_4 = 1.7. \end{cases}$$

13. $$\begin{cases} 3.3x_1 + 2.1x_2 + 2.8x_3 = 0.8, \\ 4.1x_1 + 3.7x_2 + 4.8x_3 = 5.7, \\ 2.7x_1 + 1.8x_2 + 1.1x_3 = 3.2. \end{cases}$$

14. $\begin{cases} 3.2x_1 - 2.5x_2 + 3.7x_3 = 6.5, \\ 0.5x_1 + 0.34x_2 + 1.7x_3 = -0.24, \\ 1.6x_1 + 2.3x_2 - 1.5x_3 = 4.3. \end{cases}$

15. $\begin{cases} 3.2x_1 - 11.5x_2 + 3.8x_3 = 2.8, \\ 0.8x_1 + 1.3x_2 - 6.4x_3 = -6.5, \\ 2.4x_1 + 7.2x_2 - 1.2x_3 = 4.5. \end{cases}$

16. $\begin{cases} 5.4x_1 - 2.4x_2 + 3.8x_3 = 5.5, \\ 2.5x_1 + 6.8x_2 - 1.1x_3 = 4.3, \\ 2.7x_1 - 0.6x_2 + 1.5x_3 = -3.5. \end{cases}$

17. $\begin{cases} 0.9x_1 + 2.7x_2 - 3.8x_3 = 2.4, \\ 2.5x_1 + 5.8x_2 - 0.5x_3 = 3.5, \\ 4.5x_1 - 2.1x_2 + 3.2x_3 = -1.2. \end{cases}$

18. $\begin{cases} 6.3x_1 + 5.2x_2 - 0.6x_3 = 1.5, \\ 3.4x_1 - 2.3x_2 + 3.4x_3 = 2.7, \\ 0.8x_1 + 1.4x_2 + 3.5x_3 = -2.3. \end{cases}$

19. $\begin{cases} 4.1x_1 + 5.2x_2 - 5.8x_3 = 7.0, \\ 3.8x_1 - 3.1x_2 + 4.0x_3 = 5.3, \\ 7.8x_1 + 5.3x_2 - 6.3x_3 = 5.8. \end{cases}$

20. $\begin{cases} 7.1x_1 + 6.8x_2 + 6.1x_3 = 7.0, \\ 5.0x_1 + 4.8x_2 + 5.3x_3 = 6.1, \\ 8.2x_1 + 7.8x_2 + 7.1x_3 = 5.8. \end{cases}$

21. $\begin{cases} 2.8x_1 + 3.8x_2 - 3.2x_3 = 4.5, \\ 2.5x_1 - 2.8x_2 + 3.3x_3 = 7.1, \\ 6.5x_1 - 7.1x_2 + 4.8x_3 = 6.3. \end{cases}$

22. $\begin{cases} 3.8x_1 + 4.1x_2 - 2.3x_3 = 4.8, \\ -2.1x_1 + 3.9x_2 - 5.8x_3 = 3.3, \\ 1.8x_1 + 1.1x_2 - 2.1x_3 = 5.8. \end{cases}$

23. $\begin{cases} 5.4x_1 - 6.2x_2 - 0.5x_3 = 0.52, \\ 3.4x_1 + 2.3x_2 + 0.8x_3 = -0.8, \\ 6.5x_1 - 7.1x_2 + 4.8x_3 = 6.3. \end{cases}$

24. $\begin{cases} 4.5x_1 - 3.5x_2 + 7.4x_3 = 2.5, \\ 3.1x_1 - 0.6x_2 - 2.3x_3 = -1.5, \\ 0.8x_1 + 7.4x_2 - 0.5x_3 = 6.4. \end{cases}$

25. $\begin{cases} 5.6x_1 + 2.7x_2 - 1.7x_3 = 1.9, \\ 3.4x_1 - 3.6x_2 - 6.7x_3 = -2.4, \\ 0.8x_1 + 1.3x_2 + 3.7x_3 = 1.2. \end{cases}$

26. $\begin{cases} 5.4x_1 - 2.3x_2 + 3.4x_3 = -3.5, \\ 4.2x_1 + 1.7x_2 - 2.3x_3 = 2.7, \\ 3.4x_1 + 2.4x_2 + 7.4x_3 = 1.9. \end{cases}$

27. $\begin{cases} 7.6x_1 + 5.8x_2 + 4.7x_3 = 10.1, \\ 3.8x_1 + 4.1x_2 + 2.7x_3 = 9.7, \\ 2.9x_1 + 2.1x_2 + 3.8x_3 = 7.8. \end{cases}$

28. $\begin{cases} 3.6x_1 + 1.8x_2 - 4.7x_3 = 3.8, \\ 2.7x_1 - 3.6x_2 + 1.9x_3 = 0.4, \\ 1.5x_1 + 4.5x_2 + 3.3x_3 = -1.6. \end{cases}$

29. $\begin{cases} 2.7x_1 + 0.9x_2 - 1.5x_3 = 3.5, \\ 4.5x_1 - 2.8x_2 + 6.7x_3 = 2.6, \\ 5.1x_1 + 3.7x_2 - 1.4x_3 = -0.14. \end{cases}$

30. $\begin{cases} 3.8x_1 + 6.7x_2 - 1.2x_3 = 5.2, \\ 6.4x_1 + 1.3x_2 - 2.7x_3 = 3.8, \\ 2.4x_1 - 4.5x_2 + 3.5x_3 = -0.6. \end{cases}$

Розділ 4.

Апроксимація функцій однієї змінної

4.1. Основні поняття

Апроксимація (наближення) функцій є важливим допоміжним апаратом при розв'язанні деяких задач чисельного аналізу: чисельного інтегрування і диференціювання, розв'язання диференціальних рівнянь, розв'язання систем нелінійних рівнянь, задач оптимізації та ін. Задача наближення функцій виникає й при розв'язанні інших практичних і теоретичних задач.

Апроксимацією функції називають наближене представлення складної (яка має громіздке математичне представлення) або заданої у вигляді таблиці функції $f(x)$ більш простою функцією $g(x)$, яка має мінімальні відхилення від вихідної функції у заданій області аргументу x. Функцію $g(x)$ називають *апроксимувальною*.

Якщо наближення будується на заданій дискретній множині $X = \{x_k\}$, де k — натуральне число, то апроксимацію називають *точковою* (або *дискретною*). При побудові наближення на неперервній множині точок, наприклад, на відрізку $[a, b]$, апроксимацію називають *неперервною* (або *інтегральною*).

У подальшому будемо розглядати тільки точкову апроксимацію. Задача про точкову апроксимацію виникає, наприклад, у тому випадку, коли відомі експериментальні дані $y_k = f(x_k)$ деякої величини, яка описується функцією $f(x)$ у точках x_k, $k = 0, 1, \ldots, n$ та необхідно визначити її значення в інших точках. Точкова апроксимація використовується також при згущенні таблиць, коли обчислення значень $f(x)$ є складною операцією.

Існує принципово два різних типи точкової апроксимації функцій: інтерполяція і регресія.

При інтерполяції для даної функції $y = f(x)$ будується функція $g(x)$, яка приймає у заданих точках x_k ті ж значення y_k, що й функція $f(x)$,

тобто

$$g(x_k) = y_k, \qquad k = 0, 1, \ldots, n. \tag{4.1.1}$$

При цьому припускається, що серед значень x_k немає однакових, тобто $x_k \neq x_i$ при $k \neq i$. Точки x_k називають *вузлами інтерполяції*, а функція $g(x)$ — *інтерполяційною функцією*.

Звичайно на практиці для інтерполяційної функції $g(x)$ використовують найпростіші класи функцій. Одним з важливих випадків є інтерполяція функції многочленом

$$g(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_m x^m. \tag{4.1.2}$$

Якщо один й той же многочлен інтерполює функцію $f(x)$ на всьому відрізку $[a, b]$, то говорять про *глобальну інтерполяцію*. При глобальній інтерполяції степінь многочлена на одиницю менша за кількість вузлів $m = n$.

Якщо для окремих частин відрізка $[a, b]$ будують різні многочлени, то такий підхід називають *кусковою* (*локальною*) *інтерполяцією*. При кусковій інтерполяції на практиці стараються підібрати многочлен як можна меншої степені (як правило, $m = 1, 2, 3$).

Інтерполяційні многочлени використовуються для апроксимації функції в проміжних точках (між вузлами) всередині відрізку $[a, b]$. Однак іноді вони використовуються й для наближеного обчислення функції зовні цього відрізку ($x < a$ або $x > b$). Це наближення називають *екстраполяцією*. Однак ним слід користуватися обережно, тому що поблизу кінців відрізка $[a, b]$ інтерполяційний многочлен має коливальний характер зі зростаючою амплітудою і поза відрізком швидко зростає.

Обов'язкове проходження інтерполяційного полінома через табличні точки зумовлює деякі вади цього підходу. Оскільки степінь інтерполяційного многочлена пов'язаний із кількістю вузлів, то за великої кількості вузлів степінь полінома буде високим. Це збільшує час обчислення значень цього полінома та помилки округлення. Крім того, табличні дані можуть бути одержані вимірюванням та містити в собі помилки. Інтерполяційний поліном точно повторюватиме ці помилки. Тому інколи вимагають, щоб графік апроксимувальної функції проходив не через табличні точки, а поряд із ними.

Такий підхід використовують при середньоквадратичному наближенні та рівномірному наближені, які є методами *регресійного аналізу*. Мірою відхилення апроксимувальної функції $g(x)$ від функції $f(x)$ при середньоквадратичному наближенні є сума квадратів різниць значень функцій $g(x)$

і $f(x)$ у вузлових точках, а при рівномірному наближені максимальне значення абсолютної величини різниці значень цих функцій на заданому відрізку.

4.2. Методи глобальної інтерполяції

Нехай на відрізку $[a, b]$ задані вузли інтерполяції x_k, $k = 0, 1, \ldots, n$, в яких відомі значення функції $f(x)$. Задача інтерполяції алгебраїчними многочленами полягає в тому, щоб побудувати многочлен (4.1.2) степеня m, значення якого у вузлах співпадають зі значеннями функції $f(x)$ у цих точках.

Для будь-якої неперервної функції $f(x)$ сформульована задача має єдиний розв'язок. Для відшукання коефіцієнтів $a_0, a_1, \ldots, a_m$ отримуємо систему лінійних рівнянь (4.1.1), яку запишемо у вигляді ($m = n$ для глобальної інтерполяції):

$$a_0 + a_1 x_k + a_2 x_k^2 + \ldots + a_n x_k^n = f(x_k), \qquad k = 0, 1, \ldots, n, \tag{4.2.1}$$

визначник якої (*визначник Вандермонда*) відмінний від нуля, якщо серед точок x_k немає однакових.

Розв'язок системи (4.2.1) можна записати по-різному. Найбільш застосовується запис інтерполяційного многочлена у формі Лагранжа і у формі Ньютона.

Інтерполяційна формула Лагранжа дозволяє представити многочлен (4.2.1) у вигляді лінійної комбінації

$$L_n(x) = \sum_{k=0}^{n} c_k(x) f(x_k) \tag{4.2.2}$$

значень функції $f(x)$ у вузлах інтерполяції.

З умови інтерполяції $L_n(x_i) = f(x_i)$ отримуємо

$$\sum_{k=0}^{n} c_k(x_i) f(x_k) = f(x_k), \qquad i = 0, 1, \ldots, n. \tag{4.2.3}$$

Співвідношення (4.2.3) будуть виконані, якщо на функції $c_k(x)$ накласти умови

$$c_k(x_i) = \begin{cases} 0, & i \neq k, \\ 1, & i = k, \quad i = 0, 1, \ldots, n, \end{cases} \tag{4.2.4}$$

які означають, що кожна з функцій $c_k(x)$, $k = 0, 1, \ldots, n$, має не менше n нулів на відрізку $[a, b]$.

Оскільки $L_n(x)$ — многочлен степеня n, то коефіцієнти $c_k(x)$ будемо шукати також у вигляді многочленів степеня n, а саме у вигляді

$$c_k(x) = \lambda_k(x - x_0)(x - x_1)\ldots(x - x_{k-1})(x - x_{k+1})\ldots(x - x_n).$$

З умови (4.2.4) $c_k(x_k) = 1$ знаходимо

$$\lambda_k = \frac{1}{(x_k - x_0)(x_k - x_1)\ldots(x_k - x_{k-1})(x_k - x_{k+1})\ldots(x_k - x_n)}.$$

Отже, *інтерполяційний многочлен Лагранжа* має вигляд

$$L_n(x) = \sum_{k=0}^{n} \frac{(x - x_0)(x - x_1)\ldots(x - x_{k-1})(x - x_{k+1})\ldots(x - x_n)}{(x_k - x_0)(x_k - x_1)\ldots(x_k - x_{k-1})(x_k - x_{k+1})\ldots(x_k - x_n)} f(x_k). \tag{4.2.5}$$

Приклад 4.1. *Функція задана таблицею. Побудувати інтерполяційний многочлен Лагранжа і знайти значення цієї функції у точці* $x = -3$. *Реалізувати роз'язання мовою програмування Python.*

x_k	2	5	−6	7	4	3	8	9	1	−2
$f(x_k)$	−1	77	−297	249	33	9	389	573	−3	−21

Розв'язання. Для реалізації розв'язання задачі мовою програмування Python скористаємось бібліотеками: NumPy і Matplotlib (модулем pyplot). Для скорочення запису у коді використовуємо їх синоніми np і plt.

Лістинг 10. *Реалізація інтерполяції функції многочленом Лагранжа мовою Python*

```
import numpy as np
import matplotlib.pyplot as plt
def lagrang_in(x,y,q):
    g=0
    for j in range(len(y)):
        d1=1; d2=1
        for i in range(len(x)):
            if i==j:
                d1=d1*1; d2=d2*1
            else:
```

```
                d1=d1*(q-x[i])
                d2=d2*(x[j]-x[i])
        g=g+y[j]*d1/d2
    return g
x=np.array([2,5,-6,7,4,3,8,9,1,-2], dtype=float)
y=np.array([-1,77,-297,249,33,9,389,573,-3,-21], \
           dtype=float)
xnew=np.linspace(np.min(x),np.max(x),100)
ynew=[lagrang_in(x,y,i) for i in xnew]
print('g =',lagrang_in(x,y,-3))
plt.plot(x,y,'o',color = 'k',label='table')
plt.plot(xnew,ynew,color = 'k',label='Lagrang')
plt.grid()
plt.legend()
plt.show()
```

Збережемо вихідні дані в масивах. Обчислення за інтерполяційною формулою Лагранжа реалізовані у функції lagrang_in(x,y,q). Параметрами функції є масиви табличних значень x і y та q — абсциса шуканої точки (в прикладі вона дорівнює -3).

Результат проілюструємо графіками таблично заданої функції і побудованого многочлена. За допомогою функції linspace модуля NumPy формуємо масив xnew зі 100 елементів, які рівномірно розміщені на заданому відрізку абсцис. Значення відповідних ординат генеруємо у списку ynew підставленням у функцію обчислення многочлена Лагранжа абсцис xnew. Будуємо графіки вихідної таблиці даних (окремі точки) та інтерполяційної функції (суцільна лінія) за допомогою функції matplotlib.pyplot.plot. Методи grid(), legend(), show() необхідні для відображення координатної сітки, легенди і вікна з графіком.

Результат виконання коду:

```
g = -51.000000000000256
```

і рис. 4.1.

Інтерполяційна формула Ньютона дозволяє виразити інтерполяційний многочлен $L_n(x)$ через значення $f(x)$ в одному з вузлів і через поділені різниці функції $f(x)$, побудовані за вузлами $x_0, x_1, \ldots, x_n$. Вона є різницевим аналогом формули Тейлора.

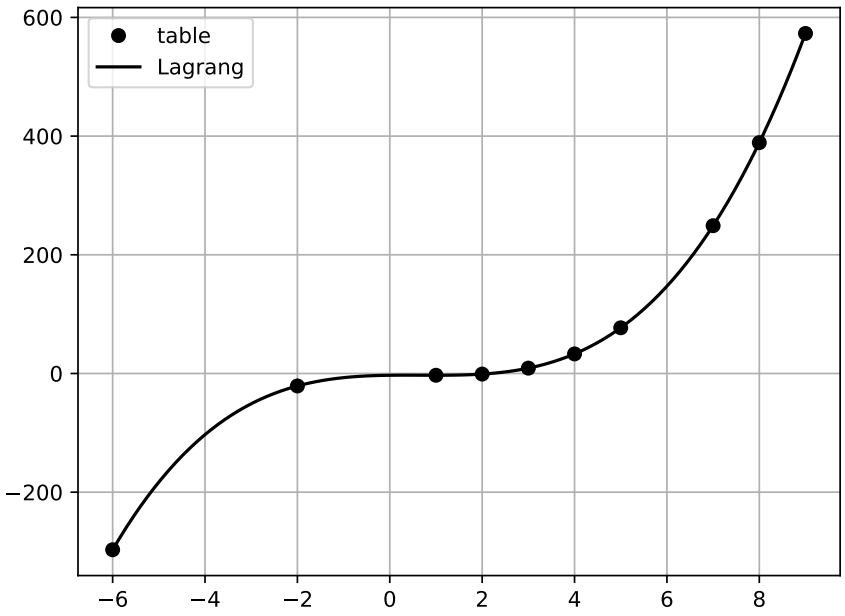

Рис. 4.1. Інтерполяція функції за методом Лагранжа

Нехай у вузлах $x_k \in [a,\, b]$, $k = 0,\, 1,\, \ldots,\, n$ відомі значення функції $f(x)$. Припустимо, що серед точок x_k немає однакових. *Поділеними різницями першого порядку* називають відношення

$$f(x_i,\, x_j) = \frac{f(x_i) - f(x_j)}{x_j - x_i}\,, \quad i,\, j = 0,\, 1,\, \ldots,\, n, \quad i \neq j. \tag{4.2.6}$$

Розглянемо поділені різниці, які складені за сусідніми вузлами. За цими поділеними різницями першого порядку можна побудувати *поділені різниці другого порядку*:

$$\begin{aligned} f(x_0,\, x_1,\, x_2) &= \frac{f(x_1,\, x_2) - f(x_0,\, x_1)}{x_2 - x_0}\,, \\ f(x_1,\, x_2,\, x_3) &= \frac{f(x_2,\, x_3) - f(x_1,\, x_2)}{x_3 - x_1}\,, \ldots, \\ f(x_{n-2},\, x_{n-1},\, x_n) &= \frac{f(x_{n-1},\, x_n) - f(x_{n-2},\, x_{n-1})}{x_n - x_{n-2}}\,. \end{aligned} \tag{4.2.7}$$

Аналогічно визначаються поділені різниці більш високого порядку. Наприклад, якщо відомі різниці $(k-1)$-порядку, то поділена різниця k-порядку визначається як

$$\begin{aligned} &f(x_j,\, x_{j+1},\, \ldots,\, x_{j+k-1},\, x_{j+k}) = \\ &\qquad = \frac{f(x_{j+1},\, x_{j+2}\, \ldots,\, x_{j+k}) - f(x_j,\, x_{j+1}\, \ldots,\, x_{j+k-1})}{x_{j+k} - x_j}\,. \end{aligned} \tag{4.2.8}$$

Інтерполяційним многочленом Ньютона називають многочлен

$$P_n(x) = f(x_0) + (x - x_0)f(x_0, x_1) + (x - x_0)(x - x_1)f(x_0, x_1, x_2) + \ldots$$
$$\ldots + (x - x_0)(x - x_1)\ldots(x - x_{n-1})f(x_0, x_1, \ldots, x_n). \qquad (4.2.9)$$

Потрібно підкреслити, що формули (4.2.5) і (4.2.9) являють собою різний запис одного й того ж інтерполяційного многочлена (4.1.2).

Інтерполяційну формулу Ньютона зручніше застосовувати у тому випадку, коли інтерполюється одна й та ж функція $f(x)$, але кількість вузлів інтерполяції поступово збільшується. Якщо вузли інтерполяції фіксовані та інтерполюється не одна, а декілька функцій, то зручніше користуватися формулою Лагранжа.

Приклад 4.2. *Функція задана таблицею (дивиться приклад 4.1). Побудувати інтерполяційний многочлен Ньютона і знайти значення цієї функції у точці $x = -3$. Реалізувати роз'язання мовою програмування Python.*

Розв'язання. Інтерполяція методом Ньютона реалізована двома функціями: coef(x, y) і in_pol_Newton(c, x, x0). Функція coef(x, y), вхідними параметрами якої є масиви табличних даних, обчислює подільні різниці, тобто визначає коефіцієнти многочлена Ньютона. Функція in_pol_Newton(c, x, x0) будує інтерполяційний многочлен Ньютона і обчислює значення функції у заданій точці. Її вхідними параметрами є масиви коефіцієнтів c і вузлів x та задана абсциса x0. Побудова графіків організована як у прикладі 4.1.

Лістинг 11. *Реалізація інтерполяції функції многочленом Ньютона мовою Python*

```
import numpy as np
import matplotlib.pyplot as plt
def coef(x, y):
    m = len(x)
    c = y.copy()
    i = 0
    for k in range(1,m):
        i += 1
        c[k:m] = (c[k:m] - c[k-1:m-1])/(x[k:m]\
                                        - x[k-i:m-i])
    return c
```

```
def in_pol_Newton(c, x, x0):
    n = len(x) - 1
    y0 = c[n]
    for k in range(1,n+1):
        y0 = c[n-k] + (x0 - x[n-k]) * y0
    return y0
x=np.array([2,5,-6,7,4,3,8,9,1,-2], dtype=float)
y=np.array([-1,77,-297,249,33,9,389,573,-3,-21],\
           dtype=float)
xnew=np.linspace(np.min(x),np.max(x),100)
c = coef(x, y)
ynew=[in_pol_Newton(c, x, i) for i in xnew]
print('g =',in_pol_Newton(c, x, -3))
plt.plot(x,y,'o',color = 'k',label='table')
plt.plot(xnew,ynew,color = 'k',label='Newton')
plt.grid(True)
plt.legend()
plt.show()
```

Результат виконання коду:

```
g = -51.0
```

і рис. 4.2.

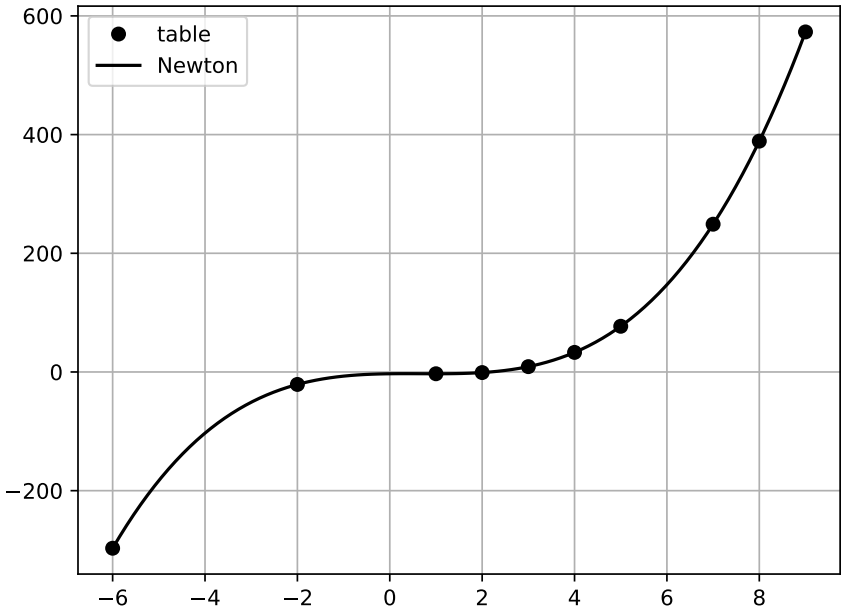

Рис. 4.2. Інтерполяція функції за методом Ньютона

4.3. Методи кускової інтерполяції

Методи глобальної інтерполяції при використанні великої кількості вузлів часто можуть привести до великих похибок. До того ж многочлени високих порядків не зручні у використанні. Тому для зменшення похибок бажано область інтерполяції розбити на декілько інтервалів і на кожному з них використовувати для апроксимації многочлен невисокого порядку, тобто скористатись *кусково-поліноміальною апроксимацією*.

Найпростішим методом кускової інтерполяції є *лінійна інтерполяція*. Вона полягає у тому, що задані точки $(x_k,\, y_k)$, $k = 0,\, 1,\, \ldots,\, n$ з'єднуються прямолінійними відрізками, і графік функції $f(x)$ наближається ламаною лінією з вершинами у даних точках.

Оскільки маємо n інтервалів $(x_{k-1},\, x_k)$, то для кожного з них в якості рівняння інтерполяційного многочлена використовується рівняння прямої, що проходить через дві точки. Для k-го інтервалу $(x_{k-1} \leqslant x \leqslant x_k)$:

$$g_k(x) = a_k x + b_k, \quad a_k = \frac{f(x_k) - f(x_{k-1})}{x_j - x_i}, \quad b_k = f(x_{k-1}) - a_k x_{k-1}. \qquad (4.3.1)$$

При використанні лінійної інтерполяції спочатку необхідно визначити інтервал, у який попадає значення аргументу x, а потім підставити його в формулу (4.3.1) і знайти наближене значення функції в цій точці.

Квадратична інтерполяція для будь-якої точки проводиться по трьох ближніх до неї вузлах. В якості інтерполяційного многочлену на відрізку $[x_{k-1},\, x_{k+1}]$ приймається квадратний тричлен, рівняння якого

$$g_k(x) = a_k x^2 + b_k x + c_k, \qquad (4.3.2)$$

містить три невідомих коефіцієнти a_k, b_k, c_k, для визначення яких необхідні три рівняння. Ці рівняння можна отримати з умови проходження параболи (4.3.2) через три відповідних вузли:

$$\begin{cases} a_k x_{k-1}^2 + b_k x_{k-1} + c_k = f(x_{k-1}), \\ a_k x_k^2 + b_k x_k + c_k = f(x_k), \\ a_k x_{k+1}^2 + b_k x_{k+1} + c_k = f(x_{k+1}). \end{cases}$$

Одним з способів інтерполювання на всьому відрізку $[a,\, b]$ є інтерполювання за допомогою сплайн-функції або просто сплайна. Слово «сплайн» (англійське spline) означає гнучку лінійку, яку використовують для проведення гладких кривих через задані точки площини.

Нехай відрізок $[a, b]$ розбитий на n часткових відрізків $[x_{k-1}, x_k]$, де $k = 1, 2, \ldots, n$, $x_0 = a$, $x_n = b$.

Сплайном називають функцію $s(x)$, яка разом з декількома похідними неперервна на всьому заданому відрізку $[a, b]$, а на кожному частковому відрізку $[x_{k-1}, x_k]$ окремо є деяким алгебраїчним многочленом. Максимальну за всіма частковими відрізками степінь многочленів називають *степеню сплайна*.

На практиці найбільшого поширення отримали *кубічні сплайни* (сплайни третього степеня). Для кубічних сплайнів потрібно, щоб були неперервними сама сплайн-функція та її перша і друга похідні.

На кожному з відрізків $[x_{k-1}, x_k]$ будемо шукати функцію $s(x) = s_k(x)$ у вигляді многочлена третього степеня

$$s_k(x) = a_k + b_k(x - x_k) + \frac{c_k}{2}(x - x_k)^2 + \frac{d_k}{6}(x - x_k)^3, \qquad (4.3.3)$$
$$x_{k-1} \leqslant x \leqslant x_k, \qquad k = 1, 2, \ldots, n,$$

де a_k, b_k, c_k, d_k — коефіцієнти, які потрібно знайти. Коефіцієнти сплайна записані у такому вигляді тому, що

$$a_k = s_k(x_k), \quad b_k = s_k'(x_k), \quad c_k = s_k''(x_k), \quad d_k = s_k'''(x_k).$$

З умов інтерполювання, неперервності сплайна та його першої і другої похідних отримуємо $4n-2$ рівняння для $4n$ невідомих. Два, ще необхідних рівняння, отримуємо при запису граничних умов для $s(x)$. Часто використовуються граничні умови $s''(a) = s''(b) = 0$. З цих умов можна отримати два рівняння $c_0 = c_n = 0$.

Коефіцієнти a_k знаходяться з умов інтерполювання

$$a_k = f(x_k), \qquad k = 1, 2, \ldots, n. \qquad (4.3.4)$$

Введемо позначення: $h_k = x_k - x_{k-1}$. З повної системи рівнянь для визначення коефіцієнтів кубічного сплайну після алгебраїчних перетворень можна отримати систему рівнянь для визначення коефіцієнтів c_k

$$h_k c_{k-1} + 2(h_k + h_{k+1})c_k + h_{k+1}c_{k+1} =$$
$$= 6\left(\frac{f(x_{k+1}) - f(x_k)}{h_{k+1}} - \frac{f(x_k) - f(x_{k-1})}{h_k}\right), \qquad (4.3.5)$$
$$k = 1, 2, \ldots, n-1, \quad c_0 = c_n = 0.$$

За знайденими коефіцієнтами c_k коефіцієнти b_k і d_k визначаються за допомогою формул

$$d_k = \frac{c_k - c_{k-1}}{h_k}, \qquad b_k = \frac{h_k}{2}c_k - \frac{h_k^2}{6}d_k + \frac{f(x_k) - f(x_{k-1})}{h_k}, \qquad (4.3.6)$$
$$k = 1, 2, \ldots, n.$$

Приклад 4.3. *Виконайте інтерполяцію лінійним, квадратичним і кубічним сплайнами даних у 5 вузлах для функції* $f(x) = (2+2(x+0.2)^2)^{-1}$ *на інтервалі* $[-1.2, 1]$. *Реалізувати інтерполювання мовою програмування Python.*

Розв'язання. Для прикладу вибрана невелика кількість вузлів для того, щоб наочно показати різницю інтерполяції різними сплайнами. Слід відмітити, що вже при кількості вузлів $n = 10$ квадратичний і кубічний сплайни дають гарний результат (переконайтесь самостійно).

Лістинг 12. *Реалізація інтерполяції функції сплайнами мовою Python*

```
import scipy
import numpy as np
from scipy import interpolate
import matplotlib.pyplot as plt
def f(x):
    return 1. / (2 + 20*(x+0.2)**2)
xk = np.linspace(-1.2,1,5)
yk = [f(i) for i in xk]
x = np.linspace(-1.2,1,200)
y = [f(i) for i in x]
f1 = scipy.interpolate.interp1d(xk, yk, kind=1)
y1d1 = f1(x)
f2 = scipy.interpolate.interp1d(xk, yk, kind=2)
y1d2 = f2(x)
f3 = scipy.interpolate.interp1d(xk, yk, kind=3)
y1d3 = f3(x)
plt.plot(xk,yk,'o',color='k',label='table')
plt.plot(x,y1d1,color='k',linestyle='—',label='line')
plt.plot(x,y1d2,color='k',linestyle='dotted',\
        label='quadratic')
plt.plot(x,y1d3,color='k',linestyle='-.',label='cubic')
plt.plot(x,y,color='k',linestyle='-',label='function')
```

```
plt.legend()
plt.show()
```

Для реалізації сплайн-інтерполяції ми скористались пакетом interpolate бібліотеки SciPy. Так як бібліотека SciPy створена для роботи з масивами бібліотеки NumPy, останню теж необхідно підключити в програмі.

Клас interp1d в scipy.interpolate — це зручний метод для створення інтерполяційної сплайн-функції для фіксованих точок даних. За допомогою функції interp1d ми створили три функції f1, f2 і f3. Ці функції для заданих масивів даних повертають функцію, виклик якої використовує інтерполяцію для пошуку нових значень точок при різних сплайнах: лінійному, квадратичному і кубічному.

Масив вхідних даних з 5 вузлів побудований за допомогою функції linspace модуля NumPy.

Результат виконання коду: рис. 4.3.

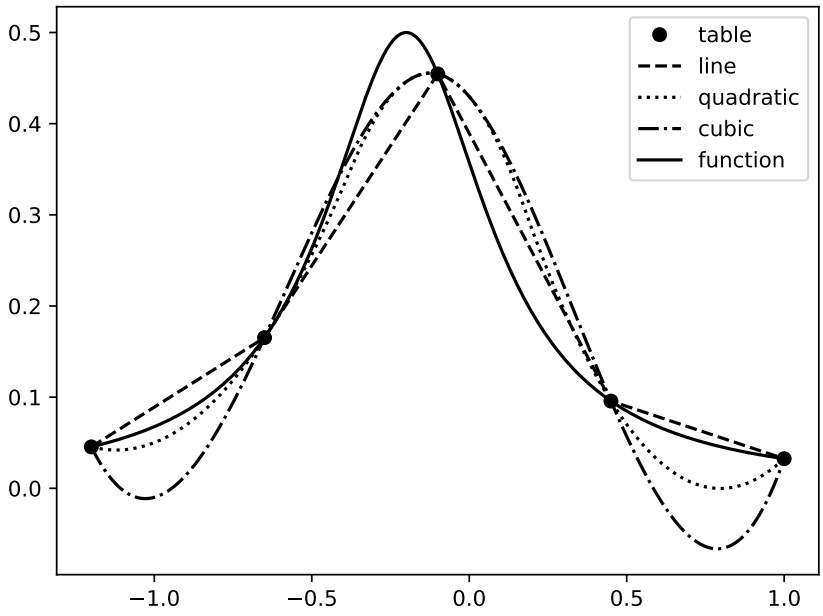

Рис. 4.3. Інтерполяція функції сплайнами

4.4. Метод найменших квадратів

У багатьох випадках (коли вихідні дані містять похибки, повтори або дуже велику кількість точок n) для апроксимації даних використовується *середньоквадратичне наближення* функцій за допомогою многочлена (4.1.2), при умові, що $m \leqslant n$. На практиці намагаються підібрати апроксимувальний многочлен якомога меншої степені (як правило, $m = 1, 2, 3$).

Кількісною оцінкою відхилення многочлена $g(x)$ від заданої функції $f(x)$ на множині точок $(x_k,\, y_k)$ $(k = 0,\, 1,\, \ldots,\, n)$ при середньоквадратичному наближені є величина $S(a_0,\, a_1,\, \ldots,\, a_m)$, яка дорівнює сумі квадратів різниць між значеннями многочлена і функції в даних точках:

$$S(a_0,\, a_1,\, \ldots,\, a_m) = \sum_{k=0}^{n} (a_0 + a_1 x_k + a_2 x_k^2 + \ldots + a_m x_k^m - y_k)^2. \qquad (4.4.1)$$

Коефіцієнти многочлена $a_0,\, a_1,\, \ldots,\, a_m$ знаходять з умови мінімуму функції $S(a_0,\, a_1,\, \ldots,\, a_m)$. У цьому полягає *метод найменших квадратів*.

Оскільки коефіцієнти $a_0,\, a_1,\, \ldots,\, a_m$ виступають у ролі незалежних змінних функції S, то її мінімум знаходять з умови рівності нулю частинних похідних функції S за цими змінними:

$$\frac{\partial S}{\partial a_0} = 0, \qquad \frac{\partial S}{\partial a_1} = 0, \qquad \ldots, \qquad \frac{\partial S}{\partial a_m} = 0. \qquad (4.4.2)$$

Співвідношення (4.4.2) — це система рівнянь для визначення $a_0,\, a_1,\, \ldots,\, a_m$.

Після знаходження частинних похідних систему (4.4.2) можна записати у наступному компактному вигляді:

$$\begin{cases} b_{00} a_0 + b_{01} a_1 + \ldots + b_{0m} a_m = c_0, \\ b_{10} a_0 + b_{11} a_1 + \ldots + b_{1m} a_m = c_1, \\ \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\ b_{m0} a_0 + b_{m1} a_1 + \ldots + b_{mm} a_m = c_m. \end{cases} \qquad (4.4.3)$$

$$b_{ij} = \sum_{k=0}^{n} x_k^{i+j}, \qquad c_i = \sum_{k=0}^{n} x_k^i y_k, \qquad i, j = 0,\, 1,\, \ldots,\, m. \qquad (4.4.4)$$

Приклад 4.4. *Функція задана таблицею. За допомогою методу найменших квадратів для заданої функції отримати емпіричну формулу лінійної регресії. Реалізувати роз'язання мовою програмування Python.*

0	1	2	3	4	5
2.1	2.9	4.15	4.98	5.5	6

Розв'язання. Для лінійного многочлена після розв'язання системи (4.4.3) отримуємо формули для знаходження коефіцієнтів:

$$a_1 = \frac{(n+1) \sum\limits_{k=0}^{n} x_k y_k - \sum\limits_{k=0}^{n} x_k \sum\limits_{k=0}^{n} y_k}{(n+1) \sum\limits_{k=0}^{n} x_k^2 - \left(\sum\limits_{k=0}^{n} x_k \right)^2}, \qquad (4.4.5)$$

$$a_0 = \frac{1}{n+1}\left(\sum_{k=0}^{n} y_k - a_1 \sum_{k=0}^{n} x_k\right). \tag{4.4.6}$$

Код програми, яка обчислює коефіцієнти (4.4.5), (4.4.6) представлений у лістингу 13.

Лістинг 13. *Реалізація методу найменших квадратів для лінійної регресії мовою Python*

```
import matplotlib.pyplot as plt
def line_regres(data_x,data_y):
        k = len(data_x);
        i = 0
        sum_xy = 0
        sum_y = 0
        sum_x = 0
        sum_sqare_x = 0
        while i < k:
            sum_xy += data_x[i]*data_y[i];
            sum_y += data_y[i]
            sum_x += data_x[i]
            sum_sqare_x += data_x[i]*data_x[i]
            i += 1
        a1 = (k*sum_xy - sum_x*sum_y)/ \
                (k*sum_sqare_x - sum_x*sum_x)
        a0 = (sum_y - a1*sum_x)/k
        return [a0,a1]
x = [0, 1, 2, 3, 4, 5]
y = [2.1, 2.9, 4.15, 4.98, 5.5, 6]
a0,a1 = line_regres(x,y)
print('y =',a0,'+(',a1,'* x )')
```

Результат виконання коду:

```
y = 2.262380952380953 +( 0.8037142857142856 * x )
```

В пакеті numpy.polynomial бібліотеки NumPy є функція Polynomial.fit(x, y, deg), яка реалізує метод найменших квадратів для знаходження коефіцієнтів многочлена степені deg для масиву даних (x, y). Скористаємось цією функцією для розв'язання цього ж прикладу.

Слід відмітити, що для представлення коефіцієнтів у звичному вигляді необхідно скористатись процедурою convert().

У лістингу 14 для візуалізації результату реалізації методу найменших квадратів ми скористались бібліотекою Matplotlib.

Лістинг 14. *Використання функції polynomial.polynomial.Polynomial.fit*

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([0,1,2,3,4,5])
y = np.array([2.1, 2.9, 4.15, 4.98, 5.5, 6])
z = np.polynomial.Polynomial.fit(x, y, 1)
z = z.convert()
print('y =',z)
xp = np.linspace(-0.5, 5.5, 100)
plt.plot(x, y, '.',color = 'k',label='table')
plt.plot(xp, z(xp),color = 'k',label='regresiya')
plt.legend()
plt.show()
```

Результат виконання коду:

```
y = 2.262380952380954 + 0.8037142857142858 x**1
```

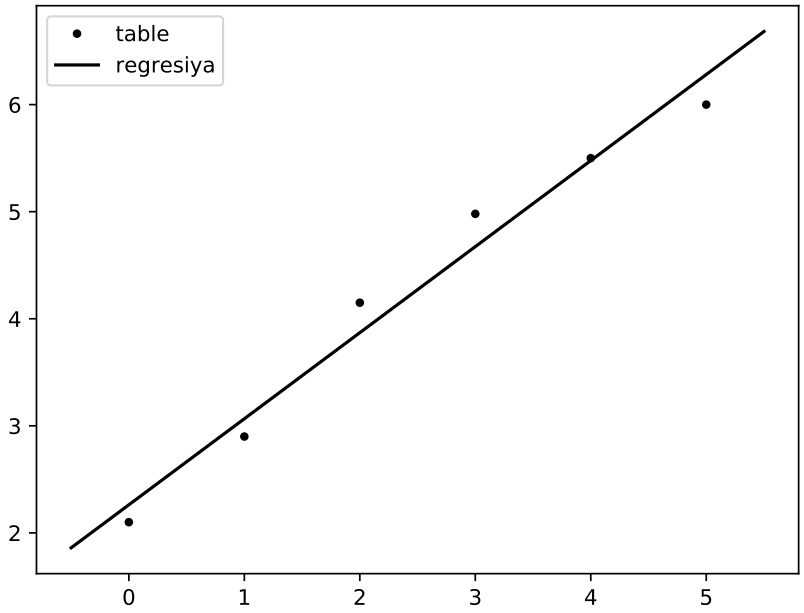

Рис. 4.4. Приклад реалізації поліноміальної регресії для степені deg=1

Завдання для самостійного розв'язання

А. Виконати:
знайти наближене значення функції при даних значеннях аргументу за допомогою: непарний варіант — інтерполяційного многочлена Лагранжа; парний варіант — інтерполяційного многочлена Ньютона.

1. $x = 0.512;\ 0.608;\ 0.736$

0.43	0.48	0.55	0.62	0.70	0.75
1.63597	1.73234	1.87686	2.03345	2.22846	2.35973

2. Таблиця і значення аргументів для визначення знаходяться в попередньому варіанті.
3. $x = 11;\ 15;\ 27$

10	13	17	21	25	28
−514	−460	−382	156	1766	3956

4. Таблиця і значення аргументів для визначення знаходяться в попередньому варіанті.
5. $x = 0.102;\ 0.125;\ 0.203$

0.02	0.08	0.12	0.17	0.23	0.30
1.02316	1.09590	1.14725	1.21483	1.30120	1.40976

6. Таблиця і значення аргументів для визначення знаходяться в попередньому варіанті.
7. $x = 12;\ 14;\ 19$

11	13	15	17	20	22
42.26	35.39	29.88	16.97	6.05	4.21

8. Таблиця і значення аргументів для визначення знаходяться в попередньому варіанті.
9. $x = 0.436;\ 0.482;\ 0.552$

0.35	0.41	0.47	0.51	0.56	0.64
2.73951	2.30080	1.96864	1.78776	1.59502	1.34310

10. Таблиця і значення аргументів для визначення знаходяться в попередньому варіанті.

11. $x = 1.5;\ 3.0;\ 5.0$

1	2	3	7	9	12
-2.20	-6.78	-17.01	-34.77	-45.16	-56.01

12. Таблиця і значення аргументів для визначення знаходяться в попередньому варіанті.
13. $x = 5;\ 8;\ 10$

2	3	6	7	9	11
16.00	21.9	77.08	95.32	114.02	137.64

14. Таблиця і значення аргументів для визначення знаходяться в попередньому варіанті.
15. $x = 0.478;\ 0.537;\ 0.673$

0.41	0.46	0.52	0.60	0.65	0.72
2.57418	2.32513	2.09336	1.86203	1.74926	1.62098

16. Таблиця і значення аргументів для визначення знаходяться в попередньому варіанті.
17. $x = 1.5;\ 2.7;\ 4.0$

0.2	0.7	1.0	3.0	3.7	5.0
0.60	0.72	0.77	0.95	1.14	1.38

18. Таблиця і значення аргументів для визначення знаходяться в попередньому варіанті.
19. $x = 0.715;\ 0.812;\ 0.955$

0.68	0.73	0.80	0.88	0.93	0.99
0.80866	0.89492	1.02964	1.20966	1.34087	1.52368

20. Таблиця і значення аргументів для визначення знаходяться в попередньому варіанті.
21. $x = 0.2;\ 1.0;\ 1.3$

0.1	0.31	0.7	0.9	1.2	1.4
1.38	0.9	0.35	0.23	0.15	0.10

22. Таблиця і значення аргументів для визначення знаходяться в попередньому варіанті.
23. $x = 0.186;\ 0.275;\ 0.332$

0.11	0.15	0.21	0.29	0.35	0.40
9.05421	6.61659	4.69170	3.35106	2.73951	2.36522

24. Таблиця і значення аргументів для визначення знаходяться в попередньому варіанті.
25. $x = 4;\ 8;\ 12$

3.5	5.0	7.2	9.3	11	13.8
112	132	155	176	210	325

26. Таблиця і значення аргументів для визначення знаходяться в попередньому варіанті.
27. $x = 0.074;\ 0.263;\ 0.351$

0.05	0.10	0.17	0.25	0.30	0.36
0.050042	0.100335	0.171657	0.255342	0.309336	0.376403

28. Таблиця і значення аргументів для визначення знаходяться в попередньому варіанті.
29. $x = 2.4;\ 2.9;\ 4.1$

2.3	2.5	2.8	3.2	3.7	4.4
11.0	13.6	15.8	17.1	18.8	21.1

30. Таблиця і значення аргументів для визначення знаходяться в попередньому варіанті.

Б. Виконати:

інтерполяцію кубічним сплайном даних у 5 і 10 вузлах для функції $f(x)$ на заданому інтервалі. Реалізувати інтерполювання мовою програмування Python.

1. (11, 21) $f(x) = x \sin x, \qquad [0,\ 6]$
2. (12, 22) $f(x) = x^2 \sin x, \qquad [-3,\ 3]$
3. (13, 23) $f(x) = x \ln x, \qquad [0.1,\ 6]$
4. (14, 24) $f(x) = \dfrac{1}{1+x}, \qquad [-0.9,\ 1]$

5. (15, 25) $f(x) = x3^x, \quad [0, 6]$
6. (16, 26) $f(x) = x^2 2^x, \quad [-2, 1.5]$
7. (17, 27) $f(x) = \dfrac{x}{1+x^2}, \quad [-3, 3]$
8. (18, 28) $f(x) = x + \sin 2x, \quad [0, 6]$
9. (19, 29) $f(x) = x \cos x, \quad [-6, 6]$
10. (20, 30) $f(x) = x \sin 2x, \quad [0, 6]$

В. Виконати:

результати експерименту подані в таблиці. Методом найменших квадратів виконати апроксимацію многочленами 1-й, 2-й і 3-й степені. Реалізувати метод мовою програмування Python.

1. (11, 21)

0.5	1	3	4.5	6	7.5	9.5	12	16	20	25	30
38	31	20.5	17	15	13.8	12.7	11.4	10	9.1	8.3	7.6

2. (12, 22)

0.5	1	3	4.5	6	7.5	9.5	12	16	20	25	30
10.7	9.4	7.1	6.8	6.1	5.6	5.2	4.6	4.4	4	3.7	3.5

3. (13, 23)

0.5	1	3	4.5	6	7.5	9.5	12	16	20	25	30
7.7	6.8	5	4.4	4.1	3.8	3.5	3.2	2.9	2.7	2.4	2.3

4. (14, 24)

0.5	1	3	4.5	6	7.5	9.5	12	16	20	25	30
11	10.5	7.6	6.6	6.2	5.8	5.2	4.8	4.4	4.1	3.8	3.5

5. (15, 25)

0.5	1	3	4.5	6	7.5	9.5	12	16	20	25	30
11.5	10.5	8.4	7.3	6.7	6.2	5.7	5.3	4.8	4.5	4.3	4.1

6. (16, 26)

0.5	1	3	4.5	6	7.5	9.5	12	16	20	25	30
14	12.2	10.5	9.7	9	8.6	8.1	7.6	7.1	6.7	6.4	6.1

7. (17, 27)

0.5	1	3	4.5	6	7.5	9.5	12	16	20	25	30
21	19	15	13.5	12.5	11.7	10.9	10.2	9.4	8.8	8.2	7.8

8. (18, 28)

0.5	1	3	4.5	6	7.5	9.5	12	16	20	25	30
9.7	8.7	6.8	6	5.7	5.3	4.9	4.6	4.2	3.9	3.6	3.4

9. (19, 29)

450	500	550	600	650	700	750	800	850	900
0.009	0.025	0.068	0.19	0.52	1.43	3.92	10.8	29.7	81.7

10. (20, 30)

450	500	550	600	650	700	750	800	850	900
0.057	0.15	0.39	1.03	2.68	7.02	18.4	48	126	329

Розділ 5.

Чисельні методи розв'язання задач диференціального та інтегрального числення

5.1. Чисельне диференціювання

Задача чисельного диференціювання полягає в знаходженні наближених значень похідних функції $y = f(x)$ в заданих точках у випадках, коли: аналітичний вид функції невідомий; функція задана неявно; аналітичний вираз досить складний; функція задана таблично.

Найбільш прості формули чисельного диференціювання отримуються тоді, коли функція $f(x)$ може бути задана таблицею своїх значень $y_k = f(x_k)$ у рівновіддалених вузлах $x_k = x_0 + kh$, $k = 0, 1, 2, \ldots, N$. Величину h при цьому називають кроком чисельного диференціювання. Для обчислення значення похідної застосовують наближену рівність

$$y' \approx \frac{f(x + \Delta x) - f(x)}{\Delta x} = \frac{\Delta y}{\Delta x}. \tag{5.1.1}$$

Співвідношення (5.1.1) називають *апроксимацією похідної за допомогою відношення скінченних різниць* (скінченні на відміну від нескінченно малих у визначенні похідної).

Значна кількість формул чисельного диференціювання отримана як наслідок інтерполяційних формул. Наведемо кілька найпростіших формул для похідної першого порядку:

формула диференціювання назад (лівих різностей)

$$f'(x_1) \approx \frac{f(x_1) - f(x_1 - h)}{h} = \frac{y_1 - y_0}{h}; \tag{5.1.2}$$

формула диференціювання вперед (правих різностей)

$$f'(x_1) \approx \frac{f(x_1+h)-f(x_1)}{h} = \frac{y_2-y_1}{h}\,; \tag{5.1.3}$$

симетрична формула диференціювання (центральних різностей)

$$f'(x_1) \approx \frac{f(x_1+h)-f(x_1-h)}{2h} = \frac{y_2-y_0}{2h}\,. \tag{5.1.4}$$

Зазначимо, що запис формул через функцію $f(x)$ відповідає постановці задачі диференціювання коли її аналітичний вигляд дуже складний або вона задана параметрично. Це означає, що значення функції $f(x)$ можна розрахувати в потрібних точках x_k з кроком h. При цьому h підбирають залежно від поведінки функції $f(x)$ в околі точки x_1.

Запис формул через y_k відповідає постановці задачі диференціювання, коли функція $f(x)$ задана таблицею. Для крайніх точок x_k можна застосовувати формули диференціювання вперед та назад (відповідно до того, з якого вони боку), а для внутрішніх точок краще застосовувати симетричну формулу диференціювання.

Наведемо ще формулу для похідною першого порядку у випадку чотирьох рівновіддалених вузлів:

$$f'(x_1) \approx \frac{-2y_0-3y_1+6y_2-y_3}{6h}\,. \tag{5.1.5}$$

Похибки, які виникають при чисельному диференціюванні, визначаються відхиленням наближеного значення похідної від її дійсного значення внаслідок використання наближених формул (*похибка апроксимації*) та наближеними значеннями функції у вузлах і похибками округлення при проведені розрахунків на ЕОМ (*похибка округлення*).

Похибка апроксимації зменшується із зменшенням кроку h. Похибка округлення, навпаки, зростає із зменшенням h. Похибка, яка виникає при обчисленні відношення різниць, значно більша похибки значень функції і навіть може необмежено зростати, якщо крок h прямує до нуля. Тому операцію чисельного диференціювання називають некоректною.

Це не означає, що не можна користуватися формулами чисельного диференціювання. Сумарна похибка чисельного диференціювання спадає при зменшені кроку лише до деякого граничного значення, після чого подальше зменшення кроку приводить до її збільшення. Оптимальна точність може бути досягнута за рахунок *регуляризації* процедури чисельного диференціювання. Простим способом регуляризації є такий вибір кроку h, при якому справедлива нерівність $|f(x+h)-f(x)| > \varepsilon$, де ε є деяким досить малим дійсним числом, причому $\varepsilon > 0$.

5.2. Чисельне інтегрування

До обчислення визначених інтегралів зводяться багато наукових і практичних задач: обчислення площі фігур, визначення роботи змінної сили, знаходження пройденого шляху при заданій залежності швидкості від часу, визначення кількості речовини, яка вступила в хімічну реакцію та ін. Розв'язання задач з використанням кратних інтегралів при певних умовах може бути зведено до обчислення визначних інтегралів.

Задача чисельного інтегрування полягає в обчислені визначеного інтегралу у випадках, коли аналітичне визначення неможливе або дуже складне.

Нехай функція $f(x)$ визначена та обмежена на відрізку $[a, b]$ і $a = x_0 < x_1 < \ldots < x_n = b$ — довільне розбиття цього відрізку на n проміжків. Виберемо на кожному відрізку $[x_k, x_{k+1}]$ точку ξ_k. Тоді суму

$$S_n = \sum_{k=0}^{n-1} f(\xi_k)(x_{k+1} - x_k) = \sum_{k=0}^{n-1} f(\xi_k)\Delta x_k \tag{5.2.1}$$

називають *інтегральною сумою* функції $f(x)$ на відрізку $[a, b]$, складеною для даного розбиття і даного вибору точок ξ_k.

Введемо позначення $\lambda_n = \max\limits_{0 \leqslant k \leqslant n-1} \Delta x_k$. Тоді границю інтегральної суми (5.2.1) при $\lambda_n \to 0$, якщо вона існує і скінченна, називають *визначеним інтегралом* від функції $f(x)$ на відрізку $[a, b]$ та позначають $\int\limits_a^b f(x)\,\mathrm{d}x$. Отже,

$$\int_a^b f(x)\,\mathrm{d}x = \lim_{\lambda_n \to 0} S_n. \tag{5.2.2}$$

У випадку $f(x) \geqslant 0$ визначений інтеграл є площею фігури, яка обмежена графіком функції $y = f(x)$ віссю Ox, вертикальними відрізками $x = a$, $x = b$ $(b > a)$ і яку називають *криволінійною трапецією* (рис. 5.1).

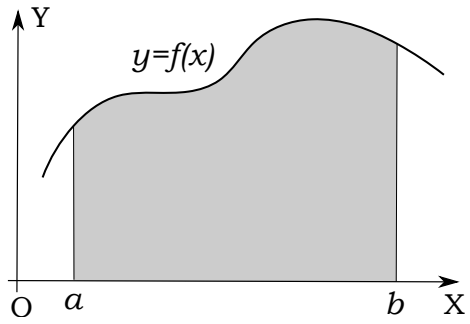

Рис. 5.1. Геометричний зміст визначеного інтегралу

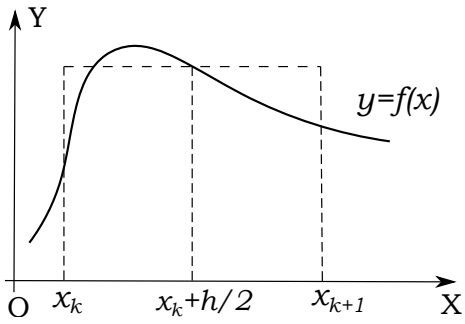

Рис. 5.2. Геометричний зміст формули прямокутників

Простим методом чисельного інтегрування є *метод прямокутників*. Він безпосередньо використовує заміну визначеного інтегралу інтегральною сумою (5.2.1). Частіше використовують розбиття із сталим кроком h. В якості точок ξ_k можна вибрати ліві ($\xi_k = x_k$) або праві ($\xi_k = x_{k+1}$) границі відрізків та отримати відповідні формули лівих і правих прямокутників:

$$\int_a^b f(x)\,\mathrm{d}x \approx h\sum_{k=0}^{n-1} f(x_k), \tag{5.2.3}$$

$$\int_a^b f(x)\,\mathrm{d}x \approx h\sum_{k=0}^{n-1} f(x_k + h). \tag{5.2.4}$$

Більш точною є формула прямокутників, яка використовує значення функції у середніх точках відрізків (рис. 5.2):

$$\int_a^b f(x)\,\mathrm{d}x \approx h\sum_{k=0}^{n-1} f(x_k + \frac{1}{2}h). \tag{5.2.5}$$

Похибку методу середніх прямокутників можна оцінити за формулою:

$$R_{pr} \approx \frac{h^2(b-a)}{24} M_2, \tag{5.2.6}$$

де $M_2 = \max\limits_{a \leqslant x \leqslant b} |f''(x)|$.

Похибка формули середніх прямокутників має другий порядок точності відносно h.

Формула (5.2.1) дозволяє оцінити порядок отриманої помилки до обчислення інтегралу, але для практичного застосування вона малопридатна.

На практиці для оцінки помилок застосовується наступна формула:

$$R \approx \frac{1}{3}|I_{2h} - I_h|, \tag{5.2.7}$$

I_h, I_{2h} — значення інтеграла, обчислені відповідно з кроками h і $2h$.

Приклад 5.1. *Знайти значення визначеного інтегралу*

$$\int_0^1 \sqrt{2x^2 + 1}\, \mathrm{d}x$$

за допомогою методів лівих, правих і середніх прямокутників. Реалізувати роз'язання мовою програмування Python.

Розв'язання. Для знаходження значення визначеного інтегралу були використані формули (5.2.3), (5.2.4), (5.2.5). Похибки оцінювались за формулою (5.2.7). Код програми, яка обчислює інтеграл представлений у лістингу 15.

Лістинг 15. *Реалізація методу прямокутників для обчислення визначеного інтегралу мовою Python*

```
import math
def integral_pryam(f,a,b,n,mt=1/2):
    '''metod pryamokutnykiv
  livi: mt=0, pravi: mt=1, seredni: mt=1/2'''
    m = n//2
    h = (b - a)/m
    s1 = 0
    x = a + mt * h
    for k in range(m):
        s1 = s1 + h * f(x)
        x += h
    h = (b - a)/n
    s2 = 0
    x = a + mt * h
    for k in range(n):
        s2 = s2 + h * f(x)
        x += h
    d = abs(s2 - s1)/3
    return s2,d
def f(x):
```

```
    return math.sqrt(2 * x**2 + 1)
integral,pohybka = integral_pryam(f,0,1,20)
print('s_integral =','%f' % integral,'\tpohybka =',
      '%f' % pohybka)
integral,pohybka = integral_pryam(f,0,1,20,0)
print('l_integral =','%f' % integral,'\tpohybka =',
      '%f' % pohybka)
integral,pohybka = integral_pryam(f,0,1,20,1)
print('p_integral =','%f' % integral,'\tpohybka =',
      '%f' % pohybka)
```

Результат виконання коду:

```
s_integral = 1.271154  pohybka = 0.000120
l_integral = 1.253213  pohybka = 0.005860
p_integral = 1.289816  pohybka = 0.006341
```

Метод трапецій використовує лінійну інтерполяцію, тобто графік функції $y = f(x)$ замінюється на ламану лінію, яка з'єднує точки $(x_k,\, f(x_k))$. У цьому випадку площа всієї фігури (криволінійної трапеції) складається з площ прямолінійних трапецій і формула для обчислення інтегралу має вигляд:

$$\int\limits_a^b f(x)\,\mathrm{d}x \approx h\left(\frac{f(a)+f(b)}{2} + \sum_{k=1}^{n-1} f(x_k)\right). \tag{5.2.8}$$

Похибку методу трапецій можна оцінити за формулою:

$$R_{tr} \approx -\frac{h^2(b-a)}{12} M_2. \tag{5.2.9}$$

Похибка методу трапецій також має другий порядок точності відносно h і практично її можна оцінити за формулою (5.2.7).

Похибка, яку дає формула методу трапецій приблизно вдвічі більша за похибку методу середніх прямокутників і має протилежний знак. Виходячи з цього можна записати уточнену формулу для обчислення визначного інтегралу з використанням значень I_{pr} і I_{tr}, які обчислені за методами середніх прямокутників і трапецій:

$$I \approx \frac{1}{3}(2I_{pr} + I_{tr}). \tag{5.2.10}$$

Приклад 5.2. *Знайти значення визначеного інтегралу*

$$\int_0^1 \sqrt{2x^2+1}\,\mathrm{d}x$$

за допомогою методу трапецій. Реалізувати роз'язання мовою програмування Python.

Розв'язання. Для знаходження значення визначеного інтегралу була використана формул (5.2.8). Похибка оцінювалась за формулою (5.2.7). Код програми, яка обчислює інтеграл представлений у лістингу 16.

Лістинг 16. *Реалізація методу трапецій для обчислення визначеного інтегралу мовою Python*

```
import math
def integral_pryam(f,a,b,n):
    '''metod trapetsiy'''
    m = n//2
    h = (b - a)/m
    s1 = (f(a) + f(b))*h/2
    x = a + h
    for k in range(m-1):
        s1 = s1 + h * f(x)
        x += h
    h = (b - a)/n
    s2 =(f(a) + f(b))*h/2
    x = a + h
    for k in range(n-1):
        s2 = s2 + h * f(x)
        x += h
    d = abs(s2 - s1)/3
    return s2,d
def f(x):
    return math.sqrt(2 * x**2 + 1)
integral,pohybka = integral_pryam(f,0,1,20)
print('integral_trp =','%f' % integral,'pohybka =',
      '%f' % pohybka)
```

Результат виконання коду:

```
integral_trp = 1.271514 pohybka = 0.000241
```

Метод Сімпсона заснований на кусковій інтерполяції підінтегральної функції $f(x)$. Для наближеного обчислення визначеного інтегралу за методом Сімпсона інтервал $[a, b]$ розбивається з кроком h на парну кількість відрізків $2m$. Групуючи вузли трійками на кожному з відрізків $[x_0, x_2]$, $[x_2, x_4]$, …, $[x_{k-1}, x_{k+1}]$, …, $[x_{2m-2}, x_{2m}]$ підінтегральну функцію $f(x)$ замінюють інтерполяційним многочленом другого степеня:

$$f(x) \approx g_k(x) = a_k x^2 + b_k x + c_k, \qquad x_{k-1} \leqslant x \leqslant x_{k+1}. \tag{5.2.11}$$

Отже, площа криволінійної трапеції наближено дорівнює сумі m площин під параболами.

Коефіцієнти квадратних тричленів (5.2.11) можуть бути знайдені з умов рівності многочлена у вузлах x_k відповідним значенням функції $f(x_k)$. В якості $g_k(x)$ можна прийняти інтерполяційний многочлен Лагранжа другого степеня і отримати формулу для наближеного визначення інтегралу, яку називають *формулою Сімпсона*:

$$\begin{aligned}\int_a^b f(x)\,\mathrm{d}x \approx \frac{h}{3}\Big[f(a) + 4\Big(f(x_1) + f(x_3) + \ldots) + f(x_{2m-1})\Big) + \\ +2\Big(f(x_2) + f(x_4) + \cdots + f(x_{2m-2})\Big) + f(b)\Big] = \\ = \frac{h}{3}\Big[f(a) + f(b) + 4\sum_{k=1}^{m} f(x_{2k-1}) + 2\sum_{k=1}^{m-1} f(x_{2k})\Big].\end{aligned} \tag{5.2.12}$$

Похибка формули Сімпсона оцінюється так:

$$R_{smp} \approx \frac{h^4(b-a)}{2880} M_4, \tag{5.2.13}$$

де $M_4 = \max\limits_{a \leqslant x \leqslant b} |f^{\mathrm{IV}}(x)|$.

З формули (5.2.14) видно, що метод Сімпсона суттєво точніший, ніж методи прямокутників і трапецій.

На практиці для оцінки похибки методу Сімпсона застосовується формула:

$$R_{smp} \approx \frac{1}{15} |I_{2h} - I_h|, \tag{5.2.14}$$

I_h, I_{2h} — значення інтеграла, обчислені відповідно з кроками h і $2h$.

Приклад 5.3. *Знайти значення визначеного інтегралу*

$$\int_0^1 \sqrt{2x^2 + 1}\,\mathrm{d}x$$

за допомогою методу Сімпсона. Реалізувати роз'язання мовою програмування Python.

Розв'язання. Формулу (5.2.12) для реалізації алгоритму обчислення інтегралу краще записати у вигляді:

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{h}{3}\Big[f(a)+f(b)+4f(b-h)+ \tag{5.2.15}$$
$$+\sum_{k=0}^{m-2}\Big(4f(a+[2k+1]h)+2f(a+[2k+2]h)\Big)\Big].$$

Алгоритм методу Сімпсона реалізований у вигляді функції simpson(), яка має чотири аргументи: підінтегральна функція, межі інтегрування і похибка (за умовчуванням рівна 0.000001). За безпосереднє обчислення інтегралу в цій функції відповідає функція simp(). Параметр d визначає похибку методу за формулою (5.2.14) і відповідає за завершення ітерацій.

Код програми, яка обчислює інтеграл представлений у лістингу 17.

Лістинг 17. *Реалізація методу Сімпсона для обчислення визначеного інтегралу мовою Python*

```
import math
def simpson(f,a,b,delta=0.000001):
    def simp(f,a,b,n):
        h = (b - a)/n
        m = n//2
        s = f(a) + f(b)+ 4*f(b-h)
        x = a
        for i in range(m-1):
            s = s + 4*f(x+h) + 2*f(x+2*h)
            x = x + 2*h
        s = s*h/3
        return s
    d, n = 1, 1
    while abs(d) > delta:
        d = (simp(f,a,b,n * 2) - simp(f,a,b,n))/15
        n *= 2
    integral = abs(simp(f,a,b,2*n))
    print('Simpson: n        integral          pohybka')
    print('\t%s\t%f\t%f' % (2*n,integral,abs(d)))
```

```
def f(x):
    return math.sqrt(2 * x**2 + 1)
simpson(f,0,1,0.0001)
```

Результат виконання коду:

```
Simpson: n       integral        pohybka
         8       1.271273        0.000038
```

Пропонуємо читачеві самостійно порівняти значення інтегралів, які визначні за методом Сімпсона і за уточненою формулою (5.2.10).

5.3. Чисельне розв'язання задачі Коші для звичайних диференціальних рівнянь

Диференціальними називають рівняння, у яких невідомими є функція, яка входять в рівняння разом зі своїми похідними (частинними похідними).

$$F(x,\, y,\, y',\, y'',\, \ldots,\, y^{\mathrm{N}}) = 0. \tag{5.3.1}$$

Якщо в рівняння входить невідома функція тільки однієї змінної, рівняння називають *звичайним*, якщо декількох — рівнянням в *частинних похідних*.

Порядком диференціального рівняння називають найвищій порядок похідної, яка входить у рівняння.

Розв'язати диференціальне рівняння означає знайти таку функцію $y = g(x)$, підставлення якої в рівняння перетворювало б його у тотожність.

Для того щоб з рівняння N-го порядку отримати розв'язок, необхідно виконати N інтегрувань, що дає N довільних сталих. Розв'язок, який виражає функцію у явному вигляді називають *загальним розв'язком*

$$y = g(x,\, C_1,\, C_2,\, \ldots,\, C_N). \tag{5.3.2}$$

Окремим розв'язком диференціального рівняння називають загальний розв'язок, для якого вказані конкретні значення довільних сталих. Для визначення довільних сталих необхідно задати стільки додаткових умов, скільки сталих, тобто який порядок рівняння. Ці додаткові умови звичайно включають задання значень функції та її похідних в певній точці. Їх називають *початковими умовами*.

Задачу знаходження окремого розв'язку диференціального рівняння при заданих початкових умовах називають *задачею Коші*.

У ряді випадків з диференціального рівняння (5.3.1) можна виразити старшу похідну в явному вигляді та за допомогою замін і підстановок звести до еквівалентної системи N рівнянь першого порядку.

Отже, будемо знаходити розв'язок задачі Коші в наступній постановці:

$$y' = f(x, y), \qquad y(x_0) = y_0. \tag{5.3.3}$$

Методи розв'язання диференціальних рівнянь поділяють на однокрокові і багатокрокові. В однокрокових методах використовується інформація про саму інтегральну криву в попередній точці. До таких методів відносяться метод Ейлера, метод Рунге-Кута. У багатокрокових методах наступну точку інтегральної кривої можна одержати, не роблячи повторних обчислень функції, як в однокрокових методах, у них використовуються прийоми прогнозу і корекції.

Слід також зазначити, що чисельні методи застосовуються тільки до добре обумовлених задач. Диференціальне рівняння називають погано обумовленим, якщо невеликі зміни початкових умов призводять до великої зміни в інтегральній кривій.

Одним з найпростіших методів розв'язання звичайного диференціального рівняння є *метод Ейлера*. Нехай потрібно розв'язати задачу Коші (5.3.3) на відрізку $[a, b]$. Введемо по змінній x рівномірну сітку з кроком h, тобто розглянемо послідовність точок $a = x_0 < x_1 < x_2 < \ldots < x_n = b$. Будемо позначати через $y_k = y(x_k)$ — наближений розв'язок. Відмітимо, що наближений розв'язок є *сітковою функцією*, тобто визначений тільки у точках сітки.

Різницева апроксимація похідної

$$y'_k \approx \frac{\Delta y}{\Delta x} = \frac{y(x_{k+1}) - y(x_k)}{x_{k+1} - x_k} = \frac{y_{k+1} - y_k}{h}. \tag{5.3.4}$$

Так як $y'(x_k) = f(x_k, y_k)$, то отримуємо *формулу Ейлера*:

$$y_{k+1} = y_k + hf(x_k, y_k), \qquad k = 0, 1, 2, \ldots, n-1, \tag{5.3.5}$$

за допомогою якої значення сіткової функції y_{k+1} у будь-якому вузлі x_{k+1} обчислюється за її значенням y_k у попередньому вузлі x_k.

Метод Ейлера має перший порядок точності.

Приклад 5.4. *Розв'язати задачу Коші*

$$y' = y + (1 + x)y^2, \quad y(0) = -1$$

методом Ейлера на відрізку $[0, 1.5]$ *з кроком* $h = 0.1$. *Порівняти наближений розв'язок з точним:*

$$y = -\frac{1}{x + e^{-x}}.$$

Реалізувати роз'язання мовою програмування Python.

Розв'язання. Метод Ейлера для розв'язання задачі Коші реалізований у вигляді функції eiler(f,x0,y0,xn,h), яка має п'ять аргументів: f — права частина диференціального рівняння, x0 — початкова точка відрізку, y0 — значення функції у початковій точці, xn — кінцева точку відрізку, h — крок.

Код програми, яка реалізує метод Ейлера представлений у лістингу 18.

Лістинг 18. *Реалізація методу Ейлера для розв'язання задачі Коші мовою Python*

```
import numpy as np
import matplotlib.pyplot as plt
import math as mt
def eiler(f,x0,y0,xn,h):
    x = []
    y = []
    x.append(x0)
    y.append(y0)
    h = min(h,xn-x0)
    while x0 < xn:
        y0 += h*f(x0,y0)
        x0 += h
        x.append(x0)
        y.append(y0)
    return np.array(x), np.array(y)
def f(x,y):
    return y + (1 + x)*y**2
def f_res(x):
    return -1./(x+mt.exp(-x))
xnew,ynew = eiler(f,0,-1,1.5,0.1)
yres = [f_res(i) for i in xnew]
err = [mt.fabs(f_res(xnew[k])-ynew[k])\
       for k in range(len(xnew))]
for j in range(len(xnew)):
```

```
    print('%.1f %.3f %.4f' % \
          (xnew[j],ynew[j],err[j]))
plt.plot(xnew,ynew,'o',label='eiler',color='k')
xres = np.linspace(np.min(xnew),np.max(xnew),100)
yres2 = [f_res(i) for i in xres]
plt.plot(xres,yres2,lw=2,color='k',label=
         'analytical solution')
plt.grid(True)
plt.legend()
plt.show()
```

Результат виконання коду (x_k, y_k, похибка):

```
0.0 -1.000 0.0000
0.1 -1.000 0.0048
0.2 -0.990 0.0084
0.3 -0.971 0.0106
0.4 -0.946 0.0116
0.5 -0.915 0.0115
0.6 -0.881 0.0106
0.7 -0.845 0.0093
0.8 -0.808 0.0077
0.9 -0.771 0.0060
1.0 -0.735 0.0044
1.1 -0.701 0.0029
1.2 -0.668 0.0016
1.3 -0.636 0.0005
1.4 -0.607 0.0004
1.5 -0.579 0.0011
```

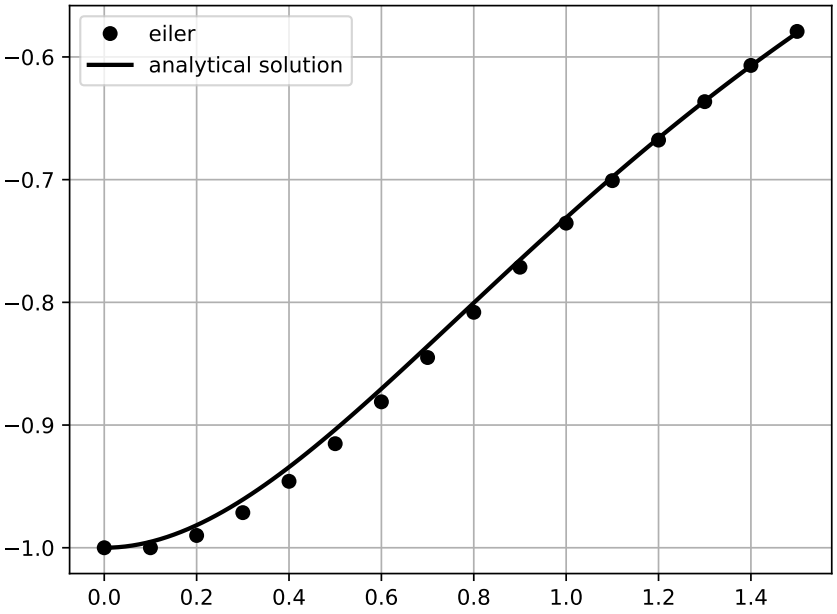

Рис. 5.3. Приклад розв'язання задачі Коші методом Ейлера

Точність методу Ейлера можна істотно підвищити, поліпшивши апроксимацію похідної. Це можна зробити, наприклад, використовуючи середнє значення похідної на початку і кінці інтервалу. У *модифікованому методі Ейлера—Коші* спочатку обчислюється значення функції в наступній точці методом Ейлера $\tilde{y}_{k+1} = y_k + hf(x_k, y_k)$. Це значення використовується потім для обчислення наближеного значення похідної в кінці інтервалу $f(x_{k+1}, \tilde{y}_{k+1})$. Обчисливши середнє між цим значенням похідної і її значенням на початку інтервалу, знайдемо більш точне значення y_{k+1}:

$$y_{k+1} = y_k + \frac{1}{2}h\Big(f(x_k, y_k) + f(x_{k+1}, \tilde{y}_{k+1})\Big), \quad k = 0, 1, 2, \ldots, n-1. \quad (5.3.6)$$

Модифікований метод Ейлера—Коші є методом другого порядку точності $O(h^2)$.

Приклад 5.5. *Розв'язати задачу Коші*

$$y' = y + (1+x)y^2, \quad y(0) = -1$$

методом Ейлера—Коші на відрізку $[0, 1.5]$ *з кроком* $h = 0.1$. *Порівняти наближений розв'язок з точним:*

$$y = -\frac{1}{x + e^{-x}}.$$

Реалізувати роз'язання мовою програмування Python.

Розв'язання. Модифікований метод Ейлера—Коші для розв'язання задачі Коші реалізований у вигляді функції eiler_mod(f,x0,y0,xn,h), яка має п'ять аргументів: f — права частина диференціального рівняння, x0 — початкова точка відрізку, y0 — значення функції у початковій точці, xn — кінцева точку відрізку, h — крок.

Код програми, яка реалізує метод Ейлера представлений у лістингу 19.

Лістинг 19. *Реалізація методу Ейлера—Коші для розв'язання задачі Коші мовою Python*

```
import numpy as np
import matplotlib.pyplot as plt
import math as mt
def eiler_mod(f,x0,y0,xn,h):
    x = []
    y = []
    x.append(x0)
    y.append(y0)
    h = min(h,xn-x0)
    while x0 < xn:
        x1 = x0 + h
        y1 = y0 + h*f(x0,y0)
        y0 += h*(f(x0,y0) + f(x1,y1))/2
        x0 += h
        x.append(x0)
        y.append(y0)
    return np.array(x), np.array(y)
def f(x,y):
    return y + (1 + x)*y**2
def f_res(x):
    return -1./(x+mt.exp(-x))
xnew,ynew = eiler_mod(f,0,-1,1.5,0.1)
yres = [f_res(i) for i in xnew]
err = [mt.fabs(f_res(xnew[k])-ynew[k])\
       for k in range(len(xnew))]
for j in range(len(xnew)):
    print('%.1f %.3f %.4f' % \
          (xnew[j],ynew[j],err[j]))
plt.plot(xnew,ynew,'o',label='eiler_mod',color='k')
```

```
xres = np.linspace(np.min(xnew),np.max(xnew),100)
yres2 = [f_res(i) for i in xres]
plt.plot(xres,yres2,lw=2,color='k',label=
        'analytical solution')
plt.grid(True)
plt.legend()
plt.show()
```

Результат виконання коду (x_k, y_k, похибка):

```
0.0 -1.000 0.0000
0.1 -0.995 0.0002
0.2 -0.981 0.0003
0.3 -0.960 0.0004
0.4 -0.934 0.0004
0.5 -0.903 0.0004
0.6 -0.870 0.0003
0.7 -0.835 0.0002
0.8 -0.800 0.0002
0.9 -0.765 0.0001
1.0 -0.731 0.0000
1.1 -0.698 0.0001
1.2 -0.666 0.0002
1.3 -0.636 0.0002
1.4 -0.608 0.0003
1.5 -0.581 0.0003
```

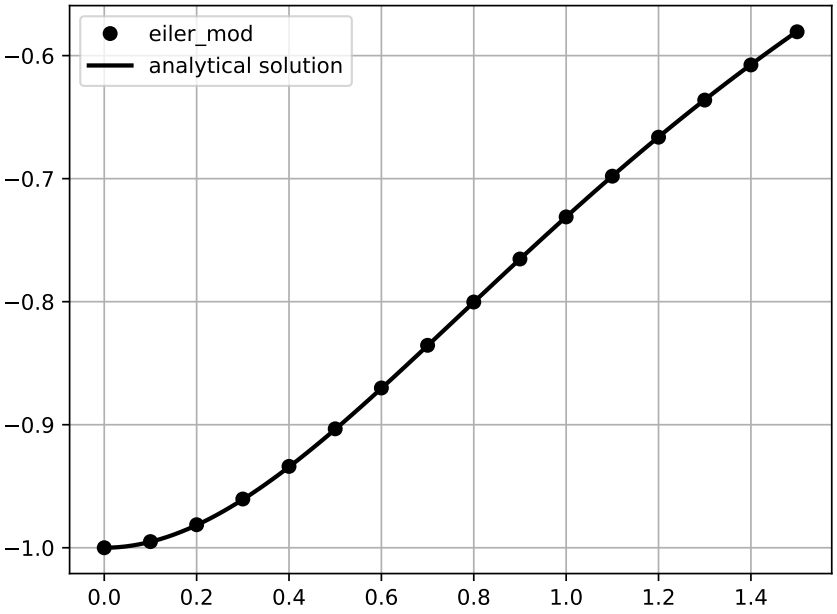

Рис. 5.4. Приклад розв'язання задачі Коші методом Ейлера—Коші

Існують й інші явні однокрокові методи розв'язання задачі Коші для звичайних диференціальних рівнянь. Найбільш поширеним з них є *метод Рунге—Кутта*. На його основі можуть бути побудовані різнецеві схеми різного порядку точності. Метод Ейлера і його модифікований варіант (метод Ейлера—Коші) можуть розглядатись як методи Рунге—Кутта першого і другого порядку.

Найбільш поширеним є метод Рунге—Кутта четвертого порядку, який вважається методом підвищеної точності (четвертого порядку точності). Алгоритм цього методу наступний:

$$\begin{gathered} y_{j+1} = y_j + \frac{1}{6}(K_0 + 2K_1 + 2K_2 + K_3), \quad j = 0,\, 1,\, 2,\, \ldots, \\ K_0 = hf(x_j,\, y_j), \quad K_1 = hf(x_j + h/2,\, y_j + K_0/2), \\ K_2 = hf(x_j + h/2,\, y_j + K_1/2), \quad K_3 = hf(x_j + h,\, y_j + K_2). \end{gathered} \tag{5.3.7}$$

Приклад 5.6. *Розв'язати задачу Коші*

$$y' = y + (1 + x)y^2, \quad y(0) = -1$$

методом Рунге—Кутта четвертого порядку на відрізку $[0,\, 1.5]$ *з кроком* $h = 0.1$. *Порівняти наближений розв'язок з точним:*

$$y = -\frac{1}{x + e^{-x}}\,.$$

Реалізувати роз'язання мовою програмування Python.

Розв'язання. Метод Рунге—Кутта четвертого порядку для розв'язання задачі Коші реалізований у вигляді функції runge_kutta(f,x0,y0,xn,h), яка має п'ять аргументів: f — права частина диференціального рівняння, x0 — початкова точка відрізку, y0 — значення функції у початковій точці, xn — кінцева точку відрізку, h — крок.

Код програми, яка реалізує метод Ейлера представлений у лістингу 20.

Лістинг 20. *Реалізація методу Рунге—Кутта четвертого порядку для розв'язання задачі Коші мовою Python*

```
import numpy as np
def coef(f,x,y,h):
    k0 = h * f(x,y)
    k1 = h * f(x+h/2.,y+k0/2.)
    k2 = h * f(x+h/2.,y+k1/2.)
    k3 = h * f(x+h,y+k2)
    return (k0 + 2.*k1 + 2.*k2 + k3)/6
def runge_kutta(f,x0,y0,xn,h):
    x = []
    y = []
    x.append(x0)
    y.append(y0)
    h = min(h,xn-x0)
    while x0 < xn:
        y0 += coef(f,x0,y0,h)
        x0 += h
        x.append(x0)
        y.append(y0)
    return np.array(x), np.array(y)
import matplotlib.pyplot as plt
import math as mt
def f(x,y):
    return y + (1 + x)*y**2
def f_res(x):
    return -1./(x+mt.exp(-x))
xnew,ynew = runge_kutta(f,0,-1,1.5,0.1)
yres = [f_res(i) for i in xnew]
err = [mt.fabs(f_res(xnew[k])-ynew[k])\
       for k in range(len(xnew))]
```

```
for j in range(len(xnew)):
    print('%.1f %.7f %.7f' % \
          (xnew[j],ynew[j],err[j]))
plt.plot(xnew,ynew,'o',label='runge_kutta',color='k')
plt.plot(xnew,yres,lw=2,color='k',label=
         'analytical solution')
plt.grid(True)
plt.legend()
plt.show()
```

Результат виконання коду (x_k, y_k, похибка):

```
0.0 -1.0000000 0.0000000
0.1 -0.9951856 0.0000003
0.2 -0.9816130 0.0000006
0.3 -0.9607818 0.0000008
0.4 -0.9342991 0.0000009
0.5 -0.9037246 0.0000009
0.6 -0.8704639 0.0000009
0.7 -0.8357107 0.0000007
0.8 -0.8004291 0.0000006
0.9 -0.7653625 0.0000004
1.0 -0.7310583 0.0000003
1.1 -0.6978993 0.0000001
1.2 -0.6661363 0.0000000
1.3 -0.6359173 0.0000001
1.4 -0.6073133 0.0000002
1.5 -0.5803395 0.0000002
```

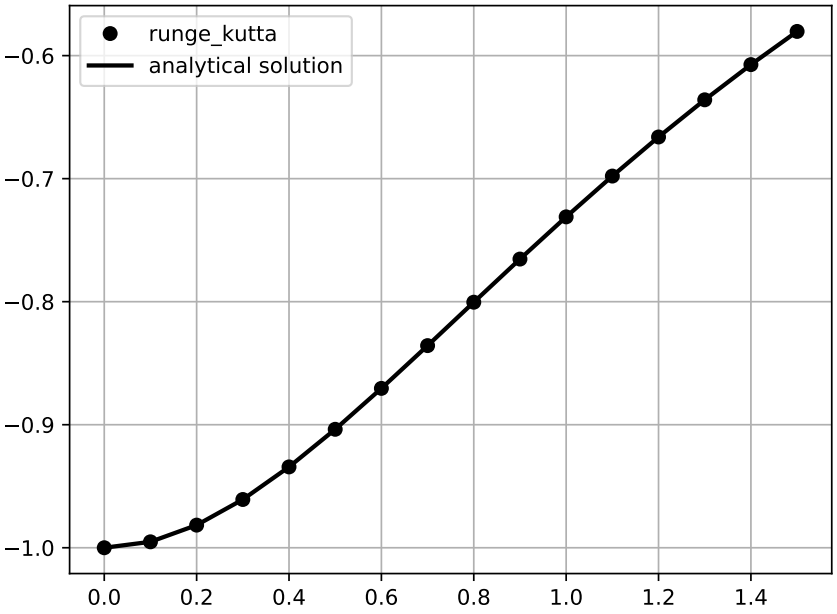

Рис. 5.5. Приклад розв'язання задачі Коші методом Рунге—Кутта четвертого порядку

Для практичної оцінки похибки розв'язання диференціального рівняння проводять з кроками $2h$ і h. За оцінку похибки розв'язку, який отриманий з кроком h, приймають величину:

$$\Delta_h = \frac{1}{2^p - 1} \cdot \max_{0 \leqslant k \leqslant n} |y_k^{2h} - y_{2k}^h|, \tag{5.3.8}$$

де y_k^{2h} — значення сіткової функції у k-й точці, яке обчислене з кроком $2h$; y_{2k}^h — значення сіткової функції у $2k$-й точці, яке обчислене з кроком h (необхідно брати до уваги те, що у цьому випадку точок в два рази більше); p — порядок точності, який для методу Ейлера рівний 1, для методу Ейлера—Коші рівний 2, методу Рунге—Кутта четвертого порядку — 4.

Для досягнення заданої точності обчислення проводять послідовно зменшуючи крок. Процес обчислення припиняється, якщо для відповідного значення кроку h буде виконана умова $\Delta_h \leqslant \varepsilon$, де ε — задана точність.

Завдання для самостійного розв'язання

А. Виконати:

знайти наближене значення визначного інтегралу з точністю 0.00001 від функції $f(x)$ на відрізку $[a, b]$ методами прямокутників, трапецій і Сімпсона. Порівняйте результати. Обчислення реалізуйте мовою Python.

1. $f(x) = \dfrac{\lg(x+2)}{x}, \qquad [1.2,\ 2]$
2. $f(x) = (x+1)\sin x, \qquad [1.6,\ 2.4]$
3. $f(x) = \dfrac{\operatorname{tg}(x^2)}{x^2+1}, \qquad [0.2,\ 1]$
4. $f(x) = \dfrac{\cos x}{1+x}, \qquad [0.6,\ 1.4]$
5. $f(x) = \sqrt{x}\cos(x^2), \qquad [0.4,\ 1.2]$
6. $f(x) = \dfrac{\sin(2x)}{x^2}, \qquad [0.8,\ 1.2]$
7. $f(x) = \dfrac{\lg(x^2+1)}{x}, \qquad [0.8,\ 1.6]$
8. $f(x) = \dfrac{\cos x}{x+2}, \qquad [0.4,\ 1.2]$
9. $f(x) = (2x+0.5)\sin x, \qquad [0.4,\ 1.2]$
10. $f(x) = \dfrac{\operatorname{tg}(x^2+0.5)}{1+2x^2}, \qquad [0.4,\ 0.8]$
11. $f(x) = \dfrac{\sin x}{x+1}, \qquad [0.18,\ 0.98]$
12. $f(x) = \sqrt{x+1}\cos(x^2), \qquad [0.2,\ 1.8]$
13. $f(x) = x^2 \lg x, \qquad [1.4,\ 3]$
14. $f(x) = \dfrac{\lg(x^2+2)}{x+1}, \qquad [1.4,\ 2.2]$
15. $f(x) = \dfrac{\cos(x^2)}{x+1}, \qquad [0.4,\ 1.2]$
16. $f(x) = (x^2+1)\sin(x-0.5), \qquad [0.8,\ 1.6]$
17. $f(x) = x^2\cos x, \qquad [0.6,\ 1.4]$
18. $f(x) = \dfrac{\lg(x^2+3)}{2x}, \qquad [1.2,\ 2]$

19. $f(x) = \dfrac{\lg(x^2 + 0.8)}{x - 1}$, $[2.5,\ 3.3]$

20. $f(x) = \dfrac{\operatorname{tg}(x^2)}{x + 1}$, $[0.5,\ 1.2]$

21. $f(x) = \dfrac{\sin(x^2 - 1)}{2\sqrt{x}}$, $[1.3,\ 2.1]$

22. $f(x) = (x + 1)\cos(x^2)$, $[0.2,\ 1.0]$

23. $f(x) = \dfrac{\sin(x^2 - 0.4)}{x + 2}$, $[0.8,\ 1.2]$

24. $f(x) = \sqrt{x + 1}\lg(x + 3)$, $[0.15,\ 0.63]$

25. $f(x) = \dfrac{\lg(1 + x^2)}{2x - 1}$, $[1.2,\ 2.8]$

26. $f(x) = (\sqrt{x} + 1)\operatorname{tg} 2x$, $[0.6,\ 0.72]$

27. $f(x) = \dfrac{\cos x}{x^2 + 1}$, $[0.8,\ 1.2]$

28. $f(x) = (0.5x + 1)\sin 0.5x$, $[1.2,\ 2.8]$

29. $f(x) = \dfrac{\lg(x^2 + 1)}{x + 1}$, $[0.8,\ 1.6]$

30. $f(x) = 0.5x\lg(0.5x^2)$, $[1.6,\ 3.2]$

Б. Виконати:

розв'язати задачу Коші методами Ейлера, Ейлера—Коші і Рунге—Кутта на відрізку $[a,\ b]$ з кроком $h = 0.1$ при початковій умові $y(x_0) = y_0$. Порівняйте результат з точним значенням $y = y(x)$. Розв'язання реалізувати мовою Python.

1. (11, 21) $y' = e^x - y$ $[0,\ 1]$ $y(0) = 1$ $y = 0.5e^{-x} + 0.5e^x$

2. (12, 22) $y' = \cos x + 4y$ $[0,\ 1]$ $y(0) = 1$
   $y = \dfrac{1}{17}(21e^{4x} + \sin x - 4\cos x)$

3. (13, 23) $y' = \sin x \cos x - y\cos x$ $[0,\ 1]$ $y(0) = 0$
   $y = e^{-\sin x} + \sin x - 1$

4. (14, 24) $y' = \cos^2 x - y\operatorname{tg} x$ $[\pi/4,\ 5\pi/4]$ $y(\pi/4) = 0.5$
   $y = \sin x \cos x$

5. (15, 25) $y' = x^2 + 4x + \dfrac{y}{x+2} + 5 \qquad [-1,\, 0] \qquad y(-1) = 1.5$
   $y = x + 2 + 0.5(x+2)^3 + (x+2)\ln(x+2)$

6. (16, 26) $y' = e^x(x+1)^2 + \dfrac{2y}{x+1} \qquad [0,\, 1] \qquad y(0) = 1$
   $y = e^x(x+1)^2$

7. (17, 27) $y' = \dfrac{y}{\sin x \cos x} - \dfrac{1}{\sin x} - \sin x \qquad [\pi/4,\, 5\pi/4]$
   $y(\pi/4) = 1 + \sqrt{2}/2 \qquad y = \operatorname{tg} x + \cos x$

8. (18, 28) $y' = 2xy - 2x^2 + 1 \qquad [0,\, 1] \qquad y(0) = 2 \qquad y = 2e^{x^2} + x$

9. (19, 29) $y' = \dfrac{3y}{x} + x^3 + x \qquad [1,\, 2] \qquad y(1) = 3 \qquad y = x^4 - x^2 + 2|x|^2$

10. (20, 30) $y' = \dfrac{y}{x} + x\sin x \qquad [\pi/2,\, 3\pi/2] \qquad y(\pi/2) = 1$
    $y = x\Big(\dfrac{2}{\pi} - \cos x\Big)$

Бібліографія

1. Langtangen, Hans Petter. A Primer on Scientific Programming with Python. Texts in Computational Science and Engineering book series (5 ed.). Springer, 2016. 922 p.

2. Алгоритми та методи обчислень [Електронний ресурс] : навч. посіб. / М. А. Новотарський. КПІ ім. Ігоря Сікорського. Електронні текстові дані (1 файл : 4648 Кбайт). Київ : КПІ ім. Ігоря Сікорського, 2019. 407 с.

3. Дослідження операцій. Практичний курс : навч. посіб. / В. Є. Березовський, М .М. Гузій, В. М. Дякон, Л. Є. Ковальов, М. О. Медведєва. Умань : Видавець «Сочінський», 2011. 238 с.

4. Дякон В. М., Ковальов Л. Є. Математичне програмування : навч. посіб. 2-ге вид. К. : Вид-во Європ. ун-ту, 2007. 497 с.

5. Дякон В. М., Ковальов Л. Є. Моделі і методи теорії прийняття рішень : підручник. К. : АНФ ГРУП, 2013. 603 с.

6. Задачин В. М., Конюшенко І. Г. Чисельні методи : навч. посіб. Х. : Вид. ХНЕУ ім. С. Кузнеця, 2014. 180 с.

7. Програмування числових методів мовою Python : підруч. / А. В. Анісімов, А. Ю. Дорошенко, С. Д. Погорілий, Я. Ю. Дорогий ; за ред. А. В. Анісімова. К. : Видавничо-поліграфічний центр «Київський університет», 2014. 640 с.

8. Усов А. В., Шпинковський О. А., Шпинковська М. І. Чисельні методи та їх реалізація у середовищі Scilab : навч. посіб. для студентів вищих навч. закладів. Київ : Освіта України, 2013. 192 с.

9. Шаповаленко В. А., Буката Л. М., Трофименко О. Г. Чисельне обчислення функцій, характеристик матриць і розв'язування нелінійних рів-

нянь та систем рівнянь : навч. посібник. Ч. 1. Одеса : ВЦ ОНАЗ, 2010. 88 с.

10. Шаповаленко В. А., Буката Л. М., Трофименко О. Г. Чисельне обчислення функцій, характеристик матриць. Розв'язок нелінійних рівнянь та систем рівнянь : методичний посібник. Ч. 2. Одеса : ВЦ ОНАЗ, 2010. 70 с.

Предметний покажчик

Навчальне видання

ЧИСЕЛЬНІ МЕТОДИ З ПРИКЛАДАМИ РЕАЛІЗАЦІЇ МОВОЮ PYTHON

Видається в авторській редакції

Підписано до друку 17.01.2023 р. Формат 60х84/16.
Папір офсетний. Ум. друк. арк. 5,12
Тираж 300 прим. Замовлення № 1907

Видавничо-поліграфічний центр «Візаві»
20300, м. Умань, вул. Тищика, 18/19
Свідоцтво суб'єкта видавничої справи
ДК № 2521 від 08.06.2006.
тел. (04744) 4-64-88, (067) 104-64-88
vizavi-print.jimdo.com
e-mail: vizavi008@gmail.com