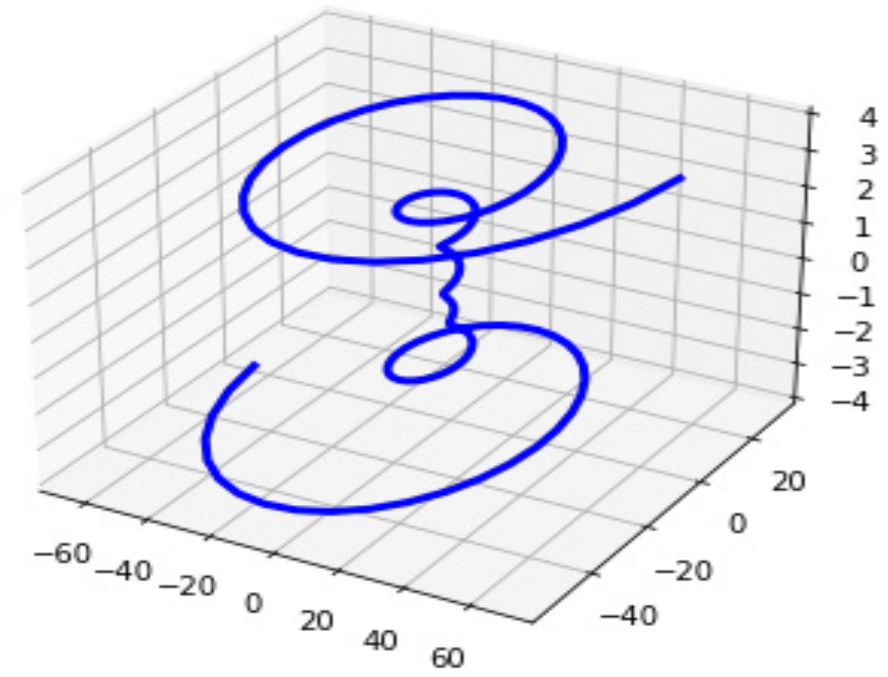
Б. І. Мокін, В. Б. Мокін, О. Б. Мокін

МЕТОДИ ТА ЗАСОБИ КОМП'ЮТЕРНИХ ОБЧИСЛЕНЬ



Міністерство освіти і науки України
Вінницький національний технічний університет

Б. І. Мокін, В. Б. Мокін, О. Б. Мокін

МЕТОДИ ТА ЗАСОБИ КОМП'ЮТЕРНИХ ОБЧИСЛЕНЬ

Навчальний посібник

Вінниця
ВНТУ
2024

УДК 517.98
М74

Рекомендовано до видання Вченою радою Вінницького національного технічного університету Міністерства освіти і науки України як навчальний посібник для студентів і аспірантів закладів вищої освіти, що спеціалізуються в галузі інформаційних технологій, та науково-педагогічних працівників, які викладають навчальну дисципліну «Методи та засоби комп'ютерних обчислень» (протокол № 4 від 31.10.2024 р.)

Рецензенти:
В. Я. Данилов, д.т.н., професор (НТУУ «КПІ» імені І. Сікорського)
В. М. Михалевич, д.т.н., професор (ВНТУ)
О. Н. Романюк, д.т.н., професор (ВНТУ)

Мокін, Б. І.

М74 Методи та засоби комп'ютерних обчислень : навчальний посібник [Електронний ресурс] / Б. І. Мокін, В. Б. Мокін, О. Б. Мокін – Вінниця : ВНТУ, 2024. – (PDF, 155 с.)

ISBN 978-617-8163-29-7 (PDF)

В навчальному посібнику викладено методи реалізації комп'ютерних обчислень з використанням програм, створених мовою Python, та рекомендації викладачам, на що потрібно звертати увагу студентам і аспірантам при вивченні ними кожного з розділів.

Навчальний посібник рекомендується для студентів і аспірантів спеціальності 124 «Системний аналіз» та для науково-педагогічних працівників, що викладають навчальну дисципліну «Методи та засоби комп'ютерних обчислень»

УДК 517.98

ISBN 978-617-8163-29-7 (PDF)

ЗМІСТ

ВСТУП

На перший погляд виглядає нераціональним наше рішення створити власний навчальний посібник з методів та засобів комп'ютерних обчислень (МЗКО) у той час, як аналогічний навчальний посібник в двох частинах [1, 2] з назвою, що суттєво корелюється з МЗКО, уже створено колективом авторів на чолі з професором Квєтним ще 10 років тому. Більш того, цим колективом у минулому році створено підручник [3] з назвою, що відрізняється від МЗКО лише одним словом. І у цих навчальних посібниках наведена велика кількість способів розв'язання обчислювальних задач, а у цьому підручнику наведені приклади програм комп'ютерної реалізації цих способів з використанням стандартних пакетів їх комп'ютерної реалізації у кількох програмних середовищах.

Але ми звернули увагу на те, що МЗКО як навчальна дисципліна вивчається студентами закладів вищої освіти (ЗВО), що спеціалізуються в галузі інформаційних технологій (ІТ), на першому курсі, коли у них ще немає ґрунтовних знань з основ програмування, оскільки навчальні дисципліни, пов'язані з вивченням мов програмування та їх застосуванням у вигляді пакетів прикладних програм, згідно з навчальними планами спеціальностей ІТ-галузі, заплановані до вивчення уже після засвоєння МЗКО. Тож, на наш погляд, корисним для студентів першого курсу буде і навчальний посібник з МЗКО, в якому акцентуватиметься увага не лише на обчислювальних методах і готових програмах їх реалізації кількома мовами, а й приділятиметься достатньо уваги вивченню основ програмування якоюсь однією конкретною мовою, програми якою будуть використовуватись в прикладах розв'язання обчислювальних задач в процесі засвоєння навчальної дисципліни МЗКО. Але, звичайно ж, використання частини годин, відведених навчальними планами ІТ-спеціальностей під МЗКО на засвоєння основ програмування якоюсь однією конкретною мовою, приведе до скорочення матеріалу, присвяченого обчислювальним методам з залишенням в їх обоймі не всього їхнього різноманіття, а лише одного-двох з цього різноманіття, найпоширеніших у практиці їх комп'ютерної реалізації. Саме ця ідея, що покладена в основу нашого посібника з МЗКО, і становить першу його відмінність від навчальних посібників та підручників з МЗКО, створених іншими авторами.

При виборі мови програмування для нашого посібника з МЗКО ми виходили з нижчевикладеного. Як загальновідомо, при використанні пакетів широко розповсюджених в практиці програм, наприклад, Matcad чи Matlab, необхідно купувати ліцензії на використання цих програмних продуктів, що є обтяжливим для малозабезпечених матеріально та фінансово студентів першого курсу, які навіть при забезпеченні доступу до цих програмних продуктів в ліцензованому варіанті у комп'ютерних центрах ЗВО при виконанні домашніх завдань та домашній підготовці до виконання

лабораторних робіт змушені будуть встановлювати ці програмні продукти на своїх домашніх ноутбуках чи у своїх гаджетах. Тому перевагу при виборі мови програмування для МЗКО ми надали мові програмування **Python**, яка створена ентузіастами розробки пакетів прикладних програм цією мовою з їх застосуванням на безкоштовній основі. Тобто, щоб використовувати ці пакети не потрібно купувати ліцензії на їх використання, а достатньо лише звернутись до програмного середовища **Anaconda** [4], в бібліотеці якого розміщені як усі вже створені мовою **Python** пакети прикладних програм, так і **Python-програми**, що створюються усіма бажаючими приєднатись до числа ентузіастів їх створення. І, звичайно ж, при орієнтації на використання мови **Python** не лишнім буде вказати і на те, що в роботах [5], [6] викладені ази програмування мовою **Python,** які, за необхідності, можна доповнювати і звертанням до навчального посібника харківського професора Долі [7].

А другою відмінністю нашого посібника з МЗКО від створених іншими авторами є наявність методичних вказівок для викладачів навчальної дисципліни МЗКО, в яких акцентується увага на особливостях вивчення того чи іншого розділу посібника та на порадах, які розділи їм варто рекомендувати студентам для розширення їхнього бачення способів розв'язання поставлених обчислювальних задач, що викладені як в посібниках, створених іншими авторами, так і в посібниках та підручниках [8–12], створених нами, які є дотичими до МЗКО, а також викладені в монографії професорів Бокса і Дженкінса [13]. Водночас особливої уваги заслуговуватимуть методичні рекомендації викладачам навчальної дисципліни МЗКО стосовно використання матеріалу, викладеного з застосуванням мови **Python** в наших навчальних посібниках [14, 15] з функціонального аналізу.

Ну і нарешті третьою, завершальною, відмінністю нашого посібника з МЗКО від створених іншими авторами є послідовність вивчення МЗКО, яка в нашому посібнику реалізована структурою у вигляді трьох розділів, в першому із яких ми даємо основи програмування мовою **Python**; у другому розділі – способи здійснення обчислень в задачах прикладної математики; а в третьому – Python-програми реалізації викладених способів здійснення обчислень.

Розділ 1 ОСНОВИ ПРОГРАМУВАННЯ МОВОЮ PYTHON

1.1 Загальна характеристика пакетів прикладних програм, створених мовою Python

Почнемо викладення матеріалу з того, що нами уже встановлене на нашому комп'ютері 64-бітове інтегроване середовище **Anaconda,** на яке ви можете зустріти посилання в різних літературних джерелах і як на бібліотеку, і як на дистрибутив, і як на пакет прикладних програм (ППП), створених мовою **Python**.

Потрібно відзначити, що встановлювати на своєму комп'ютері дистрибутив **Anaconda** вигідно ще й тому, що одночасно на вашому комп'ютері встановлюється і бібліотека пакетів прикладних програм, які використовуються для реалізації як різнопланових комп'ютерних обчислень, так і для реалізації складних алгоритмів наукових досліджень, і носять назви – **numpy, sympy, scipy, matplotlib,** суть яких розшифровується нижче. Отже:

ППП **numpy** (або **NumPy** – числовий Пайтон) – це пакет програм, який використовується для числових розрахунків;

ППП **sympy** (або **SymPy** – символьний Пайтон) – це пакет програм, який використовується для різних перетворень виразів, заданих у символьній формі (зокрема і для знаходження похідних та невизначених інтегралів від складних функцій);

ППП **scipy** (або **SciPy** – науковий Пайтон) – це пакет програм, який зручно застосовувати при обробленні результатів наукових досліджень у сукупності з пакетами **sympy** та **numpy**;

ППП **matplotlib** – це пакет програм для реалізації одно-, дво- і тривимірних графічних зображень результатів розрахунків, виконаних з застосуванням пакетів **numpy, sympy** та **scipy**.

Реалізовувати ці ППП в разі, якщо на комп'ютері встановлено дистрибутив **Anaconda**, можна за допомогою інтерпретаторів (або, що одне і те ж, консолей) **IPython, IPython Notebook**, який ще називають **Jupyter Notebook** та **Spyder**, головне вікно останнього з яких на екрані розділене на три частини, причому ліва частина призначена для набору і редагування програм у вигляді файлів, які можна або виконувати, або відправляти в пам'ять з можливістю надалі їх виклику, права нижня частина призначена для набору програм за допомогою командного рядка, їх тестування та виведення результатів їх дії і результатів тестування на екран, а права верхня частина призначена для виведення на екран рисунків, що супроводжують обчислення.

Ми в нашому навчальному посібнику будемо використовувати саме консоль **Spyder**, назва якої є абревіатурою від **Scientific PYthon Development EnviRonment**, що перекладається як «Наукового Пайтону середовище розвитку», і яка є найбільш узагальненою, оскільки інтегрує в собі основні риси інших двох інтерпретаторів, згаданих вище.

Методичні рекомендації до підрозділу 1.1. Розпочинаючи викладення матеріалу цього підрозділу викладачеві МЗКО обов'язково потрібно згадати про те, що:

1) засновники мови **Python**, метою яких було створення безкоштовної, доступної для всіх, мови програмування високого рівня, назвали цю мову не на честь змії і читати цю назву потрібно не «пітон», а читати назву мови потрібно «**Пайтон»**, бо засновники цієї мови назвали її на честь циркового коміка Пайтона, який створив популярне у 70-х роках минулого сторіччя у Великобританії комедійне шоу «**Летючий цирк Монті Пайтона**», яке дуже подобалось засновникам нової мови програмування і яке, до речі, є популярним і понині. Але, щоб задовольнити і тих, кому подобалось читати слово **Python** як «**пітон**», усі програми, які створювались цією мовою **в 64-бітовому варіанті**, їх розробники розмістили і продовжують розміщувати в програмному середовищі, названому на честь найдовшої у світі змії **Anaconda**;

2) історію створення мови **Python** вперше розповів Джейсон Р. Бріггс у своїй книжці, яка у перекладі з англійської, виконаному Олександрою Гордійчук, опублікована у Львові у видавництві старого лева у 2019 році, як: **Бріггс Джейсон Р. Python для дітей** (веселий вступ до програмування), і яка не ввійшла у прикінцевий список літератури, використаної при написанні цього навчального посібника, лише тому, що окрім посилання на історію створення мови **Python** з неї більше нічого у цьому нашому посібнику з МЗКО не використовується;

3) причиною такого ігнорування книжки Джейсона Р. Бріггса під час написання нашого навчального посібника є те, що мова **Python** у ній викладена ще в **32-бітовому середовищі IDLE**, з використанням якого нині уже ніхто **Python-програми** не створює.

1.2 Командні рядки, змінні, атрибути, функції і методи в Python-програмах

Нагадаємо, що після встановлення консолі **Spyder** екран вашого комп'ютера буде розділений на три частини, в лівій з яких ми зможемо набирати потрібні нам програми у вигляді файлів, які можна буде заносити в пам'ять і викликати з пам'яті у разі потреби їх використання. В правій нижній частині екранного вікна консолі **Spyder** ми зможемо набирати потрібні нам програми в інтерактивній оболонці **IPython,** використовуючи командні рядки, кожен із яких починається символом **In[m]**, де m – номер командного рядка, а перехід від поточного рядка до наступного здійснюється натискуванням клавіші **Enter**. А в правій верхній частині цього екранного вікна будуть відображатись рисунки і графіки функцій, якими супроводжуватиметься виконання заданих вами програм, якщо ви програмно вказуватимете на їх виведення на екран.

Перш ніж почати створення потрібних нам програм в інтерактивній оболонці **IPython** розкриємо суть понять: **змінна**, **атрибут**, **функція**, **метод в Python-середовищі Anaconda.**

Отже, почнемо з поняття «**змінна**», що **є адресою** (ярликом) того місця в пам'яті комп'ютера, в якому зберігаються потрібні нам дані і для конкретизації якої використовується ім'я, що має починатись з літери якогось алфавіту і може містити в собі цифри і букви та цілі слова, але не може містити в собі проміжки між буквами, цифрами чи словами, замість яких потрібно використовувати нижнє підкреслення. Наприклад, іменем змінної можуть бути ярлики – x, y3, z2, line, x2y4, a12b3, zelena_trava, але не можуть бути – 3y, 2z, li ne, 2xy4, 12ab3, zelena trava. Ті дані (зокрема і у вигляді виразів), які зберігаються у тому місці пам'яті комп'ютера, якому ми призначили ім'я, прив'язуються до імені змінної символом рівності «=», зліва від якого пишеться ім'я змінної, а справа – число чи вираз, який зберігається у тому місці пам'яті комп'ютера, якому присвоєне це ім'я, наприклад, x = 1, y3 = 5, z2 = 2x, line = 10. Тож, якщо нам потрібно з пам'яті комп'ютера викликати якісь конкретні дані, які туди вже поміщені під якимось іменем, наприклад **x = 1**, то ми в потрібний командний рядок **[m]** програми записуємо, **In[m]: x**, і наша програма зможе після цього, виконуючи команди, що записані в наступних рядках. використовувати і ці дані. **Усі змінні в** мові **Python називають** узагальнено **атрибутами.**

А для усвідомлення, що в мові Python мають на увазі, використовуючи поняття «**функція**», ми розкриємо суть цього поняття, спираючись на його означення в роботі [7], згідно з яким **під функцією розуміють сукупність кодів програми, яка реалізує якусь дію і яку можна викликати з програми, використавши присвоєне їй ім'я**. Отже, **змінні – це атрибути**, які є адресами тих даних, що уже поміщені в пам'ять комп'ютера в місця, яким присвоєні ці адреси, а **функції – це методи**, за допомогою яких, з використанням певної сукупності кодів програми, обчислюються нові дані, які після цього теж можуть заноситись в пам'ять комп'ютера в інші місця, які теж матимуть свої адреси, визначені новими іменами, або ж використовуватись як нові дані при виконанні подальших команд цієї програми. Тож **в мові Python усі об'єкти є або атрибутами, або методами.**

Методичні рекомендації до підрозділу 1.2. Завершуючи викладення матеріалу цього підрозділу викладачеві МЗКО обов'язково потрібно ще раз звернути увагу студентів на те, що:

1) в математиці змінна – це дійсне (float), ціле (int) чи комлексне (complex) число $\boldsymbol{x}$, задане в певному діапазоні значень $x \in [a, b]$, а в програмуванні – це адреса комірки машинної пам'яті, в якій зберігається якесь число x; і що в математиці функція – це закон, за яким кожному числу з однієї множини чисел $x \in [a, b]$ ставиться у відповідність число y з іншої множини чисел $y \in [c, d]$, а в програмуванні – це сукупність кодів програми, яка реалізує якусь дію і яку можна викликати з програми, використавши присвоєне їй ім'я;

2) в математиці функція задається або формулою $y = f(x)$, або графіком на площині з координатними осями x, y, або таблицею з рядками

значень x та зі стовпцями значень y, а в програмуванні функція задається алгоритмом, який реалізує метод оброблення певної сукупності кодів програми.

1.3 Операції в Python-середовищі Anaconda, які можна виконувати і без виклику пакетів прикладних програм numpy, sympy, scipy, matplotlib

Викладення матеріалу цього підрозділу почнемо з констатації факту, що вами вже встановлено на комп'ютер як **дистрибутив Anaconda**, так і **консоль Spyder**. В результаті цього ви вже маєте на своєму комп'ютері **інтерактивну оболонку IPython**, в якій уже можна виконувати деякі операції і без виклику прикладних пакетів програм **numpy, sympy, scipy, matplotlib**.

1.3.1 Уведення числових початкових даних та арифметичні і логічні операції з ними

Щоб виконати якусь операцію за алгоритмом **Python-програми**, потрібно командою, що передує команді, згідно з якою виконується ця операція, ввести в програму початкові дані, котрі братимуть участь у виконанні цієї операції. **Спочатку розглянемо лише випадок, коли початкові дані є числовими з множини цілих (int), дійсних (float) чи комплексних (complex) чисел**. Наприклад, якщо вам потрібно скласти два цілих числа **5** і **7**, ви маєте, задавшись адресами в пам'яті комп'ютера цих чисел у вигляді змінних ***x, y*** (чи позначених якимись іншими символами), об'єднати ці символи з заданими числами знаком **«=»** (дорівнює) та ввести результат цього об'єднання в програму, розмістивши кожного з них або в окремий командний рядок, або розмістивши їх в одному командному рядку, але розділивши їх знаком **«;»** (кома з крапкою). Кінець командного рядка введення початкових даних потрібно залишити відкритим, тобто, в кінці цього рядка не має стояти ні крапка, ні кома, ні якийсь інший розділовий знак.

Аналогічно вчиняєте ви і тоді, коли вам потрібно ввести в програму дійсні, як додатні, так і від'ємні, числа, наприклад **x1=3,2** і **y1= -4,805**; і тоді, коли вам треба ввести в програму комплексні числа, наприклад, **z1=-2,2+4,33j** і **z2=7,55-2,805j.**

І одразу ж зауважимо, що вище записані дійсні та комплексні числа так, як прийнято в підручниках з математики, тобто, відділивши дробову частину цих чисел від цілої частини комою. Але **в Python-програмах дробову частину від цілої в дійсних і комплексних числах потрібно відділяти крапкою.**

Отже, ввести початкові дані в тіло програми потрібно так, як показано в прикладі № 1.

У цьому ж прикладі № 1 продемонстровано і виконання арифметичних операцій **додавання** з символом «+», **віднімання** з символом «-»,

множення з символом «*», піднесення **до степеня** з символом «**», **ділення** з символом «/», використання результату **попередньої операції** з символом «_» (одинарне підкреслення), використання результату операції, **попередньої попередній** з символом «__» (подвійне підкреслення).

Звертаємо увагу на те, що після командного рядка під номером **[m],** в якому програмується безпосереднє виконання якоїсь операції, під символом **«Out[m]»** з'являється рядок, в якому наводиться результат виконання цієї операції. Але, якщо в командному рядку ми результату цієї операції присвоїмо ім'я, позначивши його якимось символом із числа дозволених, то результат цієї операції з'явиться в наступному рядку під символом **«Out[m]»** лише у тому випадку, коли ми, поставивши у командному рядку, в якому визначена операція з присвоєнням її результату якогось символа, кому з крапкою, після неї запишемо символ, присвоєний результату. В разі ж, якщо ми цього не зробимо, то для того, щоб викликати на екран цей результат, в наступний командний рядок треба вписати символ, присвоєний цьому результату в попередньому командному рядку.

А для кращого розуміння суті кожної команди ми у кожному командному рядку, в якому записуватиметься одна команда, справа від цієї команди **після** умовного символа «#» розміщуватимемо **пояснення дії**, яка реалізується даною командою.

Приклад № 1

```
In [1]: x=5                                  # Внесення змінної x та її значення
In [2]: y=7                                  # Внесення змінної y та її значення
In {3]: x1=3.2; y1= -4.805                   # Внесення змінних x1 та y1 і їх значень
In [4]: z1= -2.2+4.33j; z2=7.55-2.805j       # Внесення змінних z1 та z2 і їх значень
In [5]: x*y                                  # Обчислення добутку змінних x,y
Out [5]: 35
In [6]: x1y1=x1*y1                           # Обчислення добутку змінних x1,y1
In [7]: x1y1                                 #Виклик на екран добутку змінних x1,y1
Out [7]: -15.376
In [8]: x + y                                # Обчислення суми змінних x,y
Out [8]: 12
In [9]: x-y                                  # Обчислення різниці змінних x,y
Out [9]: -2
In [10]: z1+z2                               # Обчислення суми змінних z1,z2
Out [10]: (5.35+1.525j)
In [11]: z1-z2                               # Обчислення різниці змінних z1,z2
Out [11]: (-9.75+7.135j)
In [12]: z1z2=z1*z2                          # Обчислення добутку змінних z1,z2
In [13]: z1z2                                # Виклик на екран добутку змінних z1,z2
Out [13]: (-4.464349999999998+
          +38.8625j)
In [14]: z12=z1/z2; z12                      # Ділення z1 на z2 з викликом результату
Out [14]: (-0.4432775902461096+
          +0.40882203435227327j)
```

```
In [15]: x**0.5                          # Добування кореня квадратного з x
Out [15]: 2.23606797749979
In [16]: z2**0.5                         # Добування кореня квадратного з z2
Ou t[16]: (2.793226096026343-
        -0.5021075816222695j)
In [17]: _/2                             # Ділення попереднього результату на 2
Out [17]: (1.3966130480131715-
        -0.25105379081113477j)
In [18]: __*4                            # Множення результату операції № 16 на 4
Out [18]: (11.172904384105372-
        -2.008430326489078j)
In [19]: _**0.5                          # Добування кореня з результату № 18
Out [19]: (3.355956668583318-
        -0.2992336500186273j)
In [20]: __*2                            # Множення результату операції № 18 на 2
Out [20]: (22.345808768210745-
        -4.016860652978156j)
In [21]: z11=z1**3; z11                  # Піднесення z1 до кубу з викликом
Out [21]: (113.09474000000002-             результату на екран
        -18.311136999999988j)
```

А тепер розглянемо логічні операції з булевими змінними **(bool),** в яких використовуються оператори порівняння: оператор «**дорівнює**» з символом «==», оператор «**не дорівнює**» з символом «**!=**», оператор «**менше**» з символом «**<**», оператор «**менше або дорівнює**» з символом «**<=**», оператор «**більше**» з символом «**>**», оператор «**більше або дорівнює**» з символом «**>=**», оператор «**і те і те**» з символом «**and**», оператор «**або те або те**» з символом «**or**», оператор «**не те**» з символом «**not**».

Результатом застосування операторів порівняння є або булева змінна «**Правда**» з символом «**True**», або булева змінна «**Неправда**» з символом «**False**».

Потрібно звернути увагу на те, що в арифметичних операціях, які мають перевагу над логічними, **булева змінна «True» чисельно** прирівнюється до **одиниці**, а **булева змінна «False» чисельно** прирівнюється до **нуля**.

Потрібно також пам'ятати, що застосуванням функції булевої логіки «**bool» будь-яке число**, за винятком нуля, набуває значення «**True**», а **нуль** застосуванням цієї функції набуває **значення «False**».

Як працюють оператори порівняння і булеві змінні в логічних операціях показано в прикладі № 2.

Приклад № 2

```
In [1]: x=11                             # Внесення змінної x та її значення
In [2]: x>13                             # Перевірка умови x > 13
Out [2]: False
In [3]: x<13                             # Перевірка умови x <13
Out [3]: True
```

```
In [4]: 7<x<14                          # Перевірка умови 7<x<14
Out [4]: True
In [5]: y=16                            # Внесення змінної y та її значення
In [6]: x<y                             # Перевірка умови x < y
Out [6]: True
In [7]: (7<x<13) and (y>13)             # Перевірка  виконання обох умов
Out [7]: True
In [8]: (x>13) and (y>13)               # Перевірка виконання обох умов
Out [8]: False
In [9]: z=not ((x>13) and (y>13)); z    # Операція логічного заперечення
Out [9]: True
In [10]: z or (x>13)                    # Перевірка виконання умови «або те, або те»
Out[10]: True
In [11]: c=13                           # Внесення константи та її значення
In [12]: c+(7<x<13)                     # Логіко-арифметична операція
Out [12]: 14
In [13]: g=(x*(y>13) + (c+(7<x<13))); g # Логіко-арифметична операція
Out [13]: 25
In [14]: (x*(y>13) + (c+(7<x<13)))**0.5 # Логіко-арифметична операція
Out [14]: 5.0
```

Підсумуємо викладене у цьому підрозділі зауваженням, що **початкові дані в Python-програми** можуть вноситись **не лише** у вигляді дійсних **(float**), комплексних (**complex**) або цілих (**int)** чисел, **а також** логічних змінних **(bool**) зі змістом «**Правда» (True**), яка в числовому еквіваленті означає **«1»,** та зі змістом **«Неправда» (False**), яка в числовому еквіваленті означає «**0**», **але** вони можуть вноситись і у вигляді стрічок **(str),** списків **(list),** кортежів **(tuple**) та словників (**dict**), елементами яких можуть бути літери певного алфавіту або слова, складені з цих літер, дійсні, комплексні або цілі числа, а також логічні змінні.

1.3.2 Стрічки та операції з ними

Стрічки (str) – це функції, що утворюються взяттям їх елементів в одинарні (типу апострофа) або подвійні лапки.

Але, перш ніж наводити приклади стрічок і операцій з ними, **згадаємо** про **дві функції, вбудовані в інтерактивну оболонку IPython**, а тому з ними можна працювати і **без виклику** пакетів прикладних програм **numpy, sympy, scipy, matplotlib.**

Перша з них – це функція **print( ),** використанням якої результат будь-якої операції можна вивести на екран, якщо символ цього результату помістити в аргументні дужки цієї функції.

А **друга** з них – це функція **len( ),** використанням якої можна підрахувати кількість елементів у тому об’єкті, символ якого поміщено в аргументні дужки цієї функції.

А тепер знову **повернемось до стрічок**. І почнемо з того, що, якщо ми наберемо, наприклад, стрічку “Гарна погода” в рядку під номером [k], то

після натискання клавіші Enter в наступному рядку під цим же номером буде надруковано Out[k]: 'Гарна погода'. Тобто стрічка буде повторена з тим же номером, але в одинарних лапках. Якщо ж ми бажаємо в наступному рядку вивести стрічку «в чистому вигляді», тобто, без лапок, то її треба виводити на екран функцією print( ) – обидва ці **варіанти виведення на екран стрічки наведені у прикладі № 3,** як з використанням одинарних, так і подвійних лапок.

Індексування елементів в стрічці починається з нуля, а за індексом елемента в стрічці, взятому в квадратні дужки, можна викликати його значення.

Елементи в стрічці не дозволяється змінювати.

Операцією «**in**» можна перевірити, **чи є** потрібний елемент в стрічці,

Операцією «+» **дві** стрічки можна **об'єднувати в одну**, тобто, здійснити їх конкатенацію.

Операцією «*» стрічку можна **повторити** стільки разів, на скільки вказує число, яке стоїть перед символом цієї операції, утворивши у такий спосіб стрічку, в стільки ж разів довшу.

За допомогою функції **len( )** можна підрахувати, скільки елементів входить в стрічку – усі ці операції та **особливості стрічок теж наведені в прикладі № 3.**

Приклад № 3

```
In [1]: "Дощить,"                    # Внесення стрічки у формі "."
Out [1]: 'Дощить,'
In [2]: 'Дощить,'                    # Внесення стрічки у формі ' '
Out [2]: 'Дощить,'
In [3]: print ("Дощить,")            # Виведення чистої стрічки на екран
Дощить,
In [4]: print ('Дощить,')            # Виведення чистої стрічки на екран
Дощить,
In [5]: s1="Дощить,"                 # Внесення стрічки у формі s1
In [6]: s2='але тепло!'              # Внесення стрічки  у формі s2
In [7]: s1+s2                        # Формування суми стрічок s1 та s2
Out [7]: 'Дощить, але тепло!'
In [8]: s3="Слава!"                  # Внесення стрічки у формі s3
In [9]: 3*s3                         # Стрічка з трьох s3
Out [9]: 'Слава! Слава! Слава!'
In [10]: "о" in s1                   # З'ясування чи є літера «о» в s1
Out [10]: True
In [11]: "к" in s2                   # З'ясування чи є літера «к» в s3
Out [11]: False
In [12]: s2[1]                       # Виклик із s2 елемента з індексом 1
Out [12]: 'л'
In [13]: s1[3]                       # Виклик із s1 елемента з індексом 3
Out [13]: 'и'
In [14]: len(s2)                     # Визначення кількості членів в s2
Out [14]: 10
```

1.3.3 Списки та операції з ними

Списками (list) для програм мовою Python є взяті в квадратні дужки з відділенням один від одного комами елементи будь-якої природи, наприклад L=[1, ‘день’, 7, “автомобіль”, ‘щука’], які для наукового варіанта застосування мови **Python**, як правило, є числами, наприклад **L1=[5,20,17,45].**

Значення елементів списку можна змінювати за допомогою оператора присвоювання у вигляді назви списку з розміщеним поряд у квадратних дужках індексом елемента, який замінюється тим, що стоїть після знака «=», при цьому **індексація елементів списку, як і елементів стрічок, починається з нуля,** наприклад під дією оператора **присвоювання L1[3]=7** вищенаведений список L1 набуває вигляду L1=[5,20,17,7].

Конкретизація, як уже сказаного про списки, так і того, що буде сказано нижче, наведена у прикладі № 4.

Коротші списки можуть використовуватись як елементи більш довгих списків, наприклад L2=[[11,20,31], "корова", ['a', 'd', 'e']].

Окремі **елементи списку можна вилучати** за допомогою операції зрізу, символ якої **«:»** поміщається в квадратні дужки, а **зліва** від цього символа записується **індекс першого елемента**, **що переходить** із базового списку в зрізаний, а **з правого боку** записується **індекс останнього елемента**, **що вже не переходить** із базового списку в зрізаний, наприклад операція L1[0:2] трансформує наведений вище список L1 в новий список L1=[5,20], який позначається тим же символом.

Списки, як і стрічки, за допомогою оператора **«+»** можна **об’єднувати**,

А використанням оператора «*» список можна **повторювати** стільки разів, формуючи у такий спосіб більш довгий список, на скільки вказує число, що стоїть справа від цього оператора (нагадаємо, що при повторенні стрічок число повторів потрібно записувати зліва від оператора).

Як і для стрічок, за допомогою оператора «**in**» можна **перевірити чи входить** елемент, яким ми цікавимось, до списку, а за допомогою функції **len( )** можна **підрахувати** число елементів списку.

Варто вказати і на те, що застосуванням функції **list( )** можна будь-яку послідовність, записану як аргумент цієї функції (в круглих дужках), **перетворити** на список. Наприклад стрічку s= 'День народження' функцією L=list(s) можна перетворити у список
L=['Д', 'е', 'н', 'ь',' ', 'н', 'а', 'р', 'о', 'д', 'ж', 'е', 'н', 'н', 'я'].

А будь-який список, наприклад L1=[1,2,3,4], застосуванням функції **s=str(L1)** можна **трансформувати у стрічку**, яка для нашого прикладу матиме вигляд s='[1, 2, 3, 4]'.

Підтвердити, що в два списки входять одні і ті ж елементи, можна використанням оператора **«==»** (подвійний знак рівності).

Списки мають і ще низку цікавих характеристик, але ми зупинились у цьому підрозділі лише на тих із них, які нами будуть використовуватись надалі.

Приклад № 4

```
In [1]: L1=[5,6,6,5]                                  # Внесення в програму списку L1
In [2]: L2=[7,10,1,12,11]                             # Внесення в програму списку L2
In [3]: L3=L1+L2                                      # Формування списку L3
In [4]: L3                                            # Виклик на екран списку L3
Out [4]: [5, 6, 6, 5, 7, 10, 1, 12, 11]
In [5]: L3[1]=2                                       # Заміна в L3 елемента з індексом 1
In [6]: L3                                            # Виклик на екран списку L3
Out [6]: [5, 2, 6, 5, 7, 10, 1, 12, 11]
In [7]: L4=L1*3                                       # Повторення списку L1 тричі
In [8]: L4                                            # Виклик на екран списку L4
Out [8]: [5, 6, 6, 5, 5, 6, 6, 5, 5, 6, 6, 5]
In [9]: L5=L4[2:5]                                    # Формування зі списку L4 списку L5
In [10]: L5                                           # Виклик на екран списку L5
Out [10]: [6, 5, 5]
In [11]: list('Добрий день!')                         # Формування списку зі стрічки
Out [11]: ['Д', 'о', 'б', 'р', 'и', 'й', ' ', 'д', 'е', 'н', 'ь', '!']
In [12]: 7 in L5                                      # Чи є елемент 7 в списку L5?
Out[12]: False
In [13]: 6 in L5                                      # Чи є елемент 6 в списку L5?
Out[13]: True
In [14]: len (L4)                                     # Скільки елементів в списку L4?
Out [14]: 12
In [15]: len(L5)                                      # Скільки елементів в списку L5?
Out[15]: 3
In [16]: L6 = [2,10,11]                               # Внесення списку L6
In [17]: L4==L6                                       # Перевірка чи є рівними списки L4, L6
Out[17]: False
```

1.3.4 Кортежі та операції з ними

Якщо ті ж **елементи** послідовності, які задають список, наприклад список L=[1,2,3,4,5,6,7, 8], **замість квадратних взяти у круглі дужки**, то отримаємо **кортеж,** який для наведеного прикладу буде мати вигляд K=(1,2,3,4,5,6,7, 8) і який окрім форми запису відрізнятиметься від списку за властивостями лише тим, що **жоден із елементів кортежу не може бути заміненим іншим елементом**, як це мало місце при роботі зі списками. Тобто, елемент кортежу за індексом викликати можна, наприклад, набравши K[2], отримати елемент 3, але присвоїти нове значення, наприклад 15, елементу кортежу з індексом [2] за допомогою оператора присвоєння K[2]=15, не вдасться, бо програма, що написана мовою **Python**, у цьому випадку повідомить про помилку, як це показано у прикладі № 5, наведеному нижче.

Однак, якщо нам потрібно **у кортежі** якийсь елемент **поміняти**, то це зробити можна, якщо **попередньо**, скориставшись функцію **L=list (K), трансформувати кортеж К у список L**, а після внесення потрібних змін елементів у списку потім **знову повернутись до кортежу**, скориставшись функцією **K=tuple(L).**

Приклад № 5

```
In [1]: K= (1,2,3,4,5,6,7,8)                    # Внесення кортежу K
In [2]: K [2]                                   # Виклик із K елемента з індексом 2
Out [2]: 3
In [3]: K1= (9,10,11)                           # Внесення кортежу K1
In [4]: K2= K+K1; K2                            # Об'єднання кортежів K та K1
Out [4]: (1, 2, 3, 4, 5, 6, 7, 8, 9, 10,11)
In [5]: K [3]=15; K                             # Спроба зміни елемента з індексом 3
Traceback (most recent call last):
  File "<ipython-input-12-77c115e834f4>",
  line 1, in <module> K[3]=15; K
TypeError: 'tuple' object does not support
  item assignment                               # Помилка з поясненням її суті
In [6]: L= list (K); L                          # Трансформація кортежу K в список L
Out [6]: [1, 2, 3, 4, 5, 6, 7, 8]
In [7]: L [3]=15; L                             # Заміна в L елемента з індексом 3
Out[7]: [1, 2, 3, 15, 5, 6, 7, 8]
In [8]: K= tuple (L); K                         # Трансформація списку L в кортеж K
Out [8]: (1, 2, 3, 15, 5, 6, 7, 8)
```

1.3.5 Словники та операції з ними

Як уже вказано вище, в стрічках, списках і в кортежах доступ до елементів здійснюється з використанням взятих у квадратні дужки їх індексів. Але в програмах, створених мовою **Python**, можуть використовуватись і множини елементів, доступ до яких здійснюється з використанням не індексів, а ключів, заданих для кожного з цих елементів. **Множини елементів з ключами до них називають словниками.**

Створюються словники за допомогою функції **dict( )**, в аргументній частині (в круглих дужках) якої задаються і значення елементів, і ключі для доступу до кожного з цих елементів. Наприклад, виразом **d=dict(a=5, b=2,c=1)** задається словник d зі значеннями елементів 5, 2, 1 та ключами до цих елементів, заданих літерами a, b, c. А друкує цей словник за викликом d програма у вигляді **{'a':5, 'b':2, 'c':1}**.

Звертаємо увагу на те, що, якщо при роздрукуванні списку використовуються квадратні дужки, а при роздрукуванні кортежу використовуються дужки круглі, то **при роздрукуванні словника використовуються дужки фігурні.**

Якщо в аргументних дужках функції **d=dict()** не стоїть нічого, то ця функція утворює **пустий словник**, який програма роздруковує як {} і який

можна заповнювати поелементно за допомогою операторів присвоювання ключам значень відповідних елементів, наприклад оператором **d['a']=5** в пустий словник вноситься елемент зі значенням 5 та ключем до нього у вигляді ‘a’, після чого пустий словник {} перетворюється у словник **{'a':5}.** А вносячи на наступному кроці за допомогою оператора присвоювання **d['b']=2** в словник, який до першого внесення був пустим, елемент зі значенням 2 та ключем до нього у вигляді ‘b’, ми словник, що був на старті пустим, перетворюємо у словник **{'a':5, 'b':2}.** Цей процес продемонстровано у прикладі № 6, поданому нижче.

Аргументами функції d=dict( ) можуть бути також список списків, наприклад **d1=dict([['a',5], ['b',2], ['c',1]])**, або список кортежів, наприклад **d1=dict([('a',5),('b',2),('c',1)])** – і у обох цих випадках програма роздрукує вам один і той же словник у вигляді **{'a':5, 'b':2, 'c':1}.**

Щоб прочитати елемент словника, потрібно сформувати оператор виклику у вигляді символа, використаного як ім’я цього словника, з поміщеним поряд у квадратних дужках ключем цього елемента, Наприклад, якщо в програму в рядок під номером [m] внести вираз d['b'], то після натискання на клавішу Enter отримаємо під цим рядком елемент 2.

Оператором **«in»** у вигляді, наприклад **'a' in d**, внесеному в командний рядок програми, ми отримуємо можливість перевірити **чи є** елемент з ключем **'a'** в словнику **d**. Якщо він **є**, то після натискання клавіші **Enter** під цим командним рядком ми побачимо **True** (правда), а якщо елемента з таким ключем у словнику **немає**, то ми побачимо **False** (неправда).

Оператором **присвоювання** ми можемо **не лише вносити** до словника елементи, яких у ньому ще немає, **але** можемо **й змінювати** ті елементи, для яких цей ключ уже задано. Наприклад, набравши у командному рядку **d['a']=7; d['e']=4; d** та натиснувши клавішу **Enter**, ми під цим рядком **замість** словника **{'a':5, 'b':2, 'c':1},** побачите словник **{'a':7, 'b':2, 'c':1, 'e':4}.**

А оператором **del** ми можемо **видаляти** елементи зі словника. Наприклад, набравши в командному рядку вираз **del d['c']** і натиснувши клавішу **Enter,** ми в наступному рядку побачимо не той варіант словника **d**, який розміщено вище, а словник d у варіанті **{'a':7, 'b':2, 'e':4}.**

Функція **len( )** визначає нам **кількість** елементів (або, що кількісно одне і те ж, ключів) в словнику. Наприклад, якщо ми у командному рядку наберемо len(d), то після натискання на клавішу Enter ми під цим рядком прочитаєте число 3, оскільки програма буде підраховувати число елементів (ключів) в останньому нашому варіанті словника d.

А ще ми звернемо увагу на **метод keys( ),** який **впорядковує** словник за ключами. Адже впорядковані застосуванням цього методу словники d1 та d2, тобто, ці словники у варіанті **d1.keys( )** та **d2.keys( )** можна **використовувати** для виконання операцій на таких множинах, як **об’єднання** множин **D1=d1.keys( ) | d2.keys( )**, **різниця** множин **D2=d1.keys( ) - d2.keys( ), перетин** множин **D3=d1.keys( ) & d2.keys( )** та **об’єднання** множин **з вилученням** спільних елементів **D6=d1.keys( ) ^ d2.keys( ).**

Якщо ж нам потрібно отримати **список ключів**, то це можна здійснити, застосувавши функцію **k=list(d.keys( )).**

Усе, що викладено вище, продемонстровано у прикладі № 6.

Приклад № 6

```
In [1]: d1=dict(a=1, b=5,c=2); d1                    # Формування словника d1 функцією
Out [1]: {'a': 1, 'b': 5, 'c': 2}
In [2]: d1=dict([['a',1], ['b',5],['c',2]]); d1      # Формування словника d1 із списків
Out [2]: {'a': 1, 'b': 5, 'c': 2}
In [3]: d1=dict([('a',1),('b',5),('c',2)]); d1       # Формування словника d1 із кортежів
Out [3]: {'a': 1, 'b': 5, 'c': 2}
In [4]: d1={"a":1,"b":5,"c":2}; d1                   # Безпосереднє внесення словника d1
Out [4]: {'a': 1, 'b': 5, 'c': 2}
In [5]: d2= {}                                       # Внесення пустого словника d2
In [6]: d2['a']=1; d2['b']=5; d2['e']=4;d2['l']=5    # Внесення елементів в пустий словник
In [7]: d2                                           # Виклик заповненого словника d2
Out [7]: {'a': 1, 'b': 5, 'e': 4, 'l': 5}
In [8]: d1['b']                                      # Виклик із словника d1 елемента за ключем
Out [8]: 5
In [9]: d2['e']; d2['l']                             # Виклик із словника d2 двох елементів
Out [9]: (4, 5)
In [10]: 'a' in d1                                   # Перевірка наявності ключа 'a' в d1
Out [10]: True
In [11]: 'e' in d1                                   # Перевірка наявності ключа 'e' в d1
Out [11]: False
In [12]: 'e' in d2                                   # Перевірка наявності ключа 'e' в d2
Out [12]: True
In [13}: 'm' in d2                                   # Перевірка наявності ключа 'm' в d2
Out [13]: False
In [14]: k=list(d1.keys()); k                        # Формування списку ключів у словнику d1
Out [14]: ['a', 'b', 'c']
In [15]: d1.keys() | d2.keys()                       # Об'єднання множин з використанням
                                                       словників d1, d2

Out [15]: {'a', 'b', 'c', 'e', 'l'}
In [16]: d1.keys() - d2.keys()                       # Різниця множин за словниками d1, d2
Out [16]: {'c'}
In [17]: d2.keys() - d1.keys()                       # Різниця множин за словниками d2, d1
Out [17]: {'e', 'l'}
In [18]: d1.keys() & d2.keys()                       # Перетин множин з використанням
                                                       словників d1, d2

Out [18]: {'a', 'b'}
In [19]: d1.keys() ^ d2.keys()                       # Об'єднання множин з вилученням
                                                       спільних елементів

Out [19]: {'c', 'e', 'l'}
In [20]: len(d1)                                     # Визначення числа ключів в d1
Out[20]: 3
In [21]: len(d2)                                     # Визначення числа ключів в d2
Out[21]: 4
```

```
In [22]: del d2['b']                    # Видалення з d2 елемента з ключем 'b'
In [23]: d2                             # Словник d2 після видалення ключа 'b'
Out[23]: {'a': 1, 'e': 4, 'l': 5}
```

1.3.6 Вбудовані, власні і анонімні функції та способи їх формування

В інтерактивній оболонці **IPython є низка вбудованих функцій**, які після набирання їх імені в командному рядку виконуються одразу ж після розміщення в їх круглі аргументні дужки відповідного атрибуту і **натискання** на клавішу **Enter.**

З двома такими функціями:

– функцією **print( )**, за допомогою якої інформація, що розміщена в пам'яті комп'ютера під певним іменем, виводиться на його екран, якщо в аргументні круглі дужки ми помістимо це ім'я,

– а також з функцією **len( ),** яка підраховує кількість елементів у стрічці, списку чи кортежі, розміщених з використанням їх символьного імені у круглих аргументних дужках, -

ми уже познайомились у пункті 1.3.2.

Але **в інтерактивній оболонці IPython є** ціла **низка** інших вбудованих **функцій**, серед яких широко використовуються **такі** з них, як:

- функція **abs( )**, яка визначає абсолютне значення (модуль) того числа, що поміщене в аргументних дужках;
- функція **float( )**, яка визначає дійсне значення (десятковий дріб) того числа, що поміщене в аргументні дужки;
- функція **int( ),** яка визначає цілу частину дробового числа, поміщеного в аргументні дужки;
- функції **max( ), min( ),** які визначають максимальний та мінімальний елементи (за якимось критерієм) атрибуту, що поміщений в аргументні дужки;
- функція **sum( ),** яка визначає суму усіх чисел, що поміщені в списку, вкладеному в аргументні дужки;
- функція **range(, ,),** яка задає вибраний нами діапазон цілих чисел з першим параметром у вигляді числа, з якого починається відлік у вибраному діапазоні, з другим параметром, яким задається права межа діапазону, яка буде на одиницю меншою значення цього параметра, та третім параметром, яким задається крок, через який здійснюється наступна ітерація, і якщо третій параметр не вказується, то ітераційний крок дорівнює одиниці, а якщо вказано лише один параметр, то він задає послідовність цілих чисел у вибраному діапазоні з одиничним ітераційним кроком з початком рахування від нульового значення індексу.

Потрібно вказати, що після занесення в командний рядок імені кожної з усіх цих вбудованих функцій ніякі розділові знаки після них у цьому командному рядку не ставляться, і що **реалізуються вони при однократному натисканні клавіші Enter.**

Але, якщо вбудована функція **range(, ,)** використовується **для створення циклу**, **то** командний рядок [m] з нею має бути записаним так – In [m]: for i in range(, ,):, а в наступному рядку, скориставшись вбудованою функцією **print( ),** ми маємо вказати, стосовно якого атрибута здійснюватиметься цикл. А для **завершення реалізації** цієї команди потрібно **натискувати клавішу Enter двічі**.

Усе, що висловлено вище стосовно вбудованих функцій, продемонстровано у прикладі № 7.

Приклад № 7

```
In [1]: a=-7; b=5.2                          # Внесення значень чисел a, b
In [2]: z=3+4j; z1=4+5j                      # Внесення комплексів z, z1
In [3]: abs (a)                              # Визначення модуля числа a
Out [3]: 7
In [4]: abs(z)                               # Визначення модуля комплексу z
Out [4]: 5.0
In [5]: abs(z1)                              # Визначення модуля комплексу z1
Out [5]: 6.4031242374328485
In [6]: float(a)                             # Отримання дійсного значення числа a
Out [6]: -7.0
In [7]: int(b)                               # Отримання цілої частини числа b
Out [7]: 5
In [8]: L= [0,1,8,3,5,12, -2]                # Внесення списку L
In [9]: L1 = ['kat', 'dog', 'elephant']      # Внесення списку L1
In [10]: max (L)                             # Визначення максимального числа в L
Out [10]: 12
In [11]: min (L)                             # Визначення мінімального числа в L
Out[11]: -2
In [12]: sum (L)                             # Визначення суми чисел в L
Out[12]: 27
In [13]: min ([abs(z),abs(z1)])              # Визначення мінімуму модуля в [z,z1]
Out [13]: 5.0
In [14]: max([abs(z),abs(z1)])               # Визначення максимуму модуля в [z,z1]
Out [14]: 6.4031242374328485
In [15]: max(L1)                             # Визначення максимуму в списку L1
Out [15]: 'elephant'
In [16]: min(L1)                             # Визначення мінімуму в списку L1
Out [16]: 'cat'
In [17]: range (10,25)                       # Внесення діапазону чисел від 10 до 24
Out [17]: range (10, 25)
In [18]: list (range (10,15))                # Створення списку чисел в діапазоні
Out [18]: [10, 11, 12, 13, 14]
In [19]: range (10,25,2)                     # Внесення діапазону чисел з кроком 2
Out[19]: range (10, 25, 2)
In [20]: list (range (10,25,2))              # Створення списку чисел в діапазоні
Out [20]: [10, 12, 14, 16, 18, 20, 22, 24]
```

```
In [21]: for i in range (2,5):                    # Цикл із трьох повторень str “Student”
             print("Student")

Student
Student
Student
In [22}: for i in range(2,5,2):                   # Цикл із двох повторень str “Student”
             print("Student")

Student
Student
In [23]: for i in range (3):                      # Цикл із трьох повторень str “tanc”
             print ('tanc')

tanc
tanc
tanc
In [24]: for i in range (3):                      # Цикл із трьох повторень числа 5
             print(5)

5
5
5
In [25]: for i in range(3):                       # Цикл із трьох повторень списку L
             print(L)

[0,1,8,3,5,12,-2]
[0,1,8,3,5,12,-2]
[0,1,8,3,5,12,-2]
```

Примітка. Як ми вже відзначили раніше, функції **max( ) та min( ) визначають максимум та мінімум за якимось критерієм**, яким для цих функцій, внесених командами **In [10], In [11],** є місце розміщення їх атрибутів на усій числовій осі від -∞ до ∞; для функцій, внесених командами **In [13], In [14]**, є місце розміщення їх атрибутів на додатній половині числової осі від 0 до ∞; а для функцій, внесених командами **In [15], In [16],** є місце розміщення перших літер в іменах їх атрибутів в латинському алфавіті.

Вище **ми розглянули кілька функцій, уже вбудованих** в інтерактивну **оболонку IPython**, а тепер зупинимось на тому, як самому створювати функції у цій оболонці.

А самостійне створення функції потрібно починати з виклику в командний рядок програми символьного слова **def** (це скорочення від **define**, що в перекладі з англійської мови означає «визначити»), після якого через проміжок записується ім’я функції, в аргументних дужках якої записуються символи змінних, які вона використовуватиме, з комою між ними. **Після** цього запису ставиться **двокрапка**.

А **у другому рядку** цієї ж команди, зсунутому вправо на стандартну для внутрішніх операторів кількість проміжків, після символьного слова **return** (що в перекладі з англійської мови означає **«повернути»**) записується сукупність тих операцій, що їх має реалізувати ця функція.

Ну а в наступний командний рядок (**після командного рядка з def**) вносяться **значення змінних**, які беруть участь в операціях обчислення значення функції

Якщо перед символьною конструкцією з кількох змінних записаною в аргументних дужках функції, вписати **символ «*»** (зірочка), то це свідчитиме уже не про операцію перемноження, а про те, що ця символьна конструкція може використовуватись у функції **з довільним числом значень змінних**, внесених у цю символьну конструкцію. Але у цьому випадку аргументні змінні, що внесені у символьну конструкцію, які стоять після символьної конструкції з зірочкою, потрібно записувати з використанням не лише їх імен, а й їх значень.

Кожна функція може мати у своєму тілі ще й вкладені функції, які мають обчислюватись до обчислення основної функції і які після їх обчислення набувають статусу аргументних змінних для основної функції.

Усе, що висловлено вище у матеріалі, присвяченому самостійному створенню функцій, продемонстровано у прикладі № 8.

Приклад № 8

```
In [1]: def f1(x):                          # Визначення функції однієї змінної
            return x**2*z                   # Визначення структури функції

In [2]: z=3                                 # Внесення значення параметра
In [3]: f1(2)                               # Обчислення значення функції
Out [3]: 12
In [4]: def f2(x):                          # Визначення функції однієї змінної
            x+=4                            # Внесення значення параметра
            return x                        # Визначення структури функції

In [5]: f2(3)                               # Обчислення значення функції
Out [5]: 7
In [6]: def f3(x):                          # Визначення функції однієї змінної
            return sum(x)                   # Визначення структури функції

In [7]: L=[1,2,3,4,5]                       # Внесення списку L значень аргументу
In [8]: f3(L)                               # Обчислення значення функції для L
Out [8]: 15
L[1]=5                                      # Заміна значення аргументу в списку
In [9]: L                                   # Виклик зміненого списку L
Out [9]: [1, 5, 3, 4,5]
In [10]: f3(L)                              # Обчислення функції для нового списку L
Out [10]: 18
```

```
In [11]: def f4(x, y):                    # Визначення функції двох змінних
             return x**2*y                # Визначення структури функції

In [12]: x=4; y=6                         # Внесення значень аргументу
In [13]: f4(x, y)                         # Обчислення значення функції
Out[13]: 96
In [14]: L1=[4,6]                         # Внесення списку L1 значень аргументу
In [15]: f4(*L1)                          # Обчислення значення функції для L1
Out [15]: 96
In [16]: def f5(f6, x, y):                # Визначення функції від функції
             return f6(x, y)**2           # Структура залежності функції від функції

In [17]: def f6(x, y):                    # Визначення вбудованої функції
             return x**2+y                # Структури вбудованої функції

In [18]: f5(f6,3,2)                       # Обчислення функції від функції
Out[19]: 121
```

В інтерактивній оболонці IPython, окрім вбудованих і самостійно створюваних функцій, алгоритм створення яких описано вище, **використовують** також **анонімні функції**, які називають ще й лямбда-функціями, оскільки задаються з використанням слова **lambda.**

Записуються анонімні функції у командний рядок **In[m]** так: спочатку записується **ім'я функції**, після якого ставиться знак рівності «=», потім записується слово **lambda,** потім через проміжок записуються (без дужок) **аргументні змінні**, між якими ставиться кома, потім ставиться двокрапка **«:»**, а після двокрапки вказується **вираз**, який має обчислити ця функція.

Викликається анонімна функція звичним для виклику функцій способом, тобто вказується її ім'я і в круглих дужках вказуються конкретні значення аргументних змінних.

Усе, викладене вище стосовно анонімних функцій, продемонстровано у прикладі № 9.

Приклад № 9

```
In [1]: f1=lambda: 11                     # Формування лямбда-функції без аргументів
In [2]: f1( )                             # Обчислення лямбда-функції без аргументів
Out [2]: 11
In [3]: f2= lambda x, y: x**3+y**2        # Формування лямбда-функції двох аргументів
In [4]: f2(2,2)                           # Обчислення лямбда-функції двох аргументів
Out [4]: 12
In [5]: f3= lambda x: 2*x**0.5            # Формування лямбда-функції одного аргументу
In [6]: f3(16)                            # Обчислення лямбда-функції одного аргументу
Out [6]: 8.0
In [7]: (lambda x, y: x**3+y**2) (2, 2)   # Спосіб формування і обчислення лямбда-
                                              функції
Out[7]: 12
```

```
In [8]: def g(f2,x,y):                          # Формування функції від лямбда-функції
            f2= lambda x, y: x**3+y**2          # Формування вбудованої лямбда-функції
            return f2(x,y)**2                   # Визначення структури функції від
                                                    лямбда- функції
In [9]: g(f2,2,2)                               # Обчислення функції від лямбда-функції
Out[9]: 144
```

Методичні рекомендації до підрозділу 1.3. Завершуючи викладення матеріалу цього підрозділу викладачеві МЗКО обов'язково потрібно ще раз звернути увагу студентів на те, що:

1) вносити початкові числові дані в програму можна як окремим командним рядком для кожного числа, так і одним командним рядком для всієї множини початкових чисел, розділяючи їх одне від одного знаками «;» (кома з крапкою);

2) в кінці запису в командному рядку ніяких розділових знаків ставити не потрібно;

3) в десяткових дробах, які використовуються в програмах, дробову частину потрібно відділяти від цілої частини не комою, як ми звикли ще зі школи, а крапкою;

4) рядки слів, літер чи чисел треба називати стрічками, якщо вони поміщені в одинарні чи подвійні лапки, або якщо до них застосовується вбудована функція str( );

5) об'єднувати короткі стрічки, яким присвоєні символьні імена, в розлогий вираз потрібно, застосовуючи символ «+» між цими іменами, тобто здійснюючи їх конкатенацію;

6) для виклику стрічки, якій присвоєно ім'я s, без лапок на екран комп'ютера потрібно застосовувати вбудовану функцію print( ), в аргументні дужки якої поміщувати ім'я стрічки s;

7) будь-яку множину елементів, якій присвоєне якесь символьне ім'я, можна перетворити у список, якщо застосувати до неї функцію list( ), в аргументні дужки якої помістити ім'я цієї множини;

8) будь-яка множина елементів після її поміщення в квадратні дужки з проставленими між елементами комами є списком;

9) будь-яку множину елементів, якій присвоєне якесь символьне ім'я, можна перетворити у кортеж, якщо застосувати до неї функцію tuple( ), в аргументні дужки якої помістити ім'я цієї множини;

10) будь-яка множина елементів після її поміщення в круглі дужки з проставленими між елементами комами є кортежем;

11) кортеж окрім форми запису відрізняється від списку за властивостями лише тим, що жоден із елементів кортежу не може бути заміненим іншим елементом;

12) у випадку, коли потрібно змінити якийсь елемент кортежу, його спочатку потрібно за допомогою функції list( ) перетворити у список, в якому дозволяється змінювати елементи, а потім – після внесення необхідних змін

у списку – повернутись до кортежу, застосувавши функцію tuple( ) до зміненого списку;

13) тіло словника поміщається у фігурні дужки, в яких символ, який використовується як ключ до числового значення змінної, обов'язково береться в лапки, а між ключем та цим числовим значенням ставиться знак «:» (двокрапка);

14) для того, щоб можна було використати словники, які відтворюють множини, для реалізації операцій об'єднання, різниці чи перетину цих множин, потрібно спочатку здійснити впорядкування їх ключів методом keys( );

15) створюючи власну функцію, треба не забувати в кінці першого командного рядка поставити знак «:» (двокрапку), початок другого рядка змістити вправо на чотири проміжки від початку першого рядка, а після набрання тіла функції натиснути клавішу Enter двічі;

16) при створенні власної функції від функції тих же змінних доцільно для внутрішньої функції використовувати анонімну функцію lambda();

17) розміщення пояснення дії, яка реалізується командою програми, справа від команди після умовного символа «#» запозичено з роботи [7];

18) після внесення пояснення до команди, щоб продовжити набирання наступних команд програми, потрібно курсор в командному рядку змістити вліво через текст пояснення до місця закінчення основного тексту кожного командного рядка.

1.4 Обчислення з використанням ППП numpy

1.4.1 Масиви як форма внесення початкових даних в ППП numpy

В усіх попередніх підрозділах ми обчислювали значення вбудованих чи створених самостійно функцій при конкретних значеннях аргументів, отримуючи кожного разу конкретне число. Але, як правило, в прикладних задачах нам потрібно знати не одне значення функції при конкретному значенні аргументу, а множину значень цієї функції у межах приросту значень її аргументу. І при малих приростах аргументу числові значення функції розміщуватимуться настільки близько одне від одного, що їх графік зливатиметься у суцільну лінію.

У цьому випадку значення аргументів функцій в програмах, створених мовою Python, задаються у вигляді масивів. І оскільки головним пакетом програм, створених мовою Python, за допомогою якого здійснюються чисельні обчислення, є пакет програм ***numpy***, то ***масиви є об'єктами*** саме цього програмного ***пакета.***

А тому створювати масиви і здійснювати операції з ними потрібно після виклику з використанням символьного слова ***import*** програмного пакета ***numpy***, для позначення якого можна використовувати скорочення ***np*** після його закріплення за допомогою символьного слова ***as*** і яке потрібно прописувати перед іменем масиву, об'єднуючи їх крапкою.

Отже, використовувати масиви ми можемо лише після того, як в командний рядок [m] впишемо команду

In [m]: import numpy as np

і натиснемо після цього клавішу **Enter.**

У роботі [7] наведено багато інформації стосовно способів створення масивів, способів їх перетворення та виконання операцій з ними, але ми візьмемо з цієї роботи лише те, що потрібно буде нам для розв'язання прикладних задач в галузі інформаційних технологій.

Отже, створювати масиви в програмному пакеті ***numpy*** ми будемо за допомогою функцій

array( ), arange( ), linspace( ), meshgrid( ), zeros( ), ones( ), eye( ), identity( ), diag( ), fromfunction( ).

Спочатку ми охарактеризуємо перші три з наведених функцій та продемонструємо їх дію прикладом № 10, потім охарактеризуємо функцію ***meshgrid( ),*** але після з'ясування суті цієї функції ми одразу ж розкриємо ще й суть оператора (не функції) ***np.mgrid[, ,]***, який є у цьому ж пакеті, в якому іншим способом задаються незалежні змінні, але який створює такий же об'єкт, як і ця функція, та продемонструємо їх прикладом № 11.

А потім розкриємо суть останніх шести функцій та продемонструємо їх дію прикладом № 12.

Отже, функція ***array( ),*** після її виклику з програмного пакета ***numpy*** проставленням скороченого символа ***np.*** перед її іменем, створює масив із будь-якої послідовності чисел, записаних в аргументних дужках у вигляді списку, при цьому якщо список буде одновимірним, то і створений масив теж буде одновимірним, а якщо список буде двовимірним, тобто таким, в якому елементи теж являють собою списки, то і масив отримаємо двовимірний. Доступ до елементів створеного масиву здійснюється за тією ж схемою, що і доступ до елементів списку, тобто з використанням індексів елементів, взятих у квадратні дужки після імені масиву, нумерація яких, як і в списках, починається з нуля. Кількість членів масиву визначається атрибутом ***size***, записаним після імені масиву з крапкою між ними; розмірність масиву визначається атрибутом ***ndim***, записаним після імені масиву з крапкою між ними; кількість членів масиву за кожним з його вимірів визначається атрибутом ***shape***, записаним після імені масиву з крапкою між ними, а кількість членів масиву за першим з вимірів визначає уже згадувана нами раніше функція ***len( )*** з іменем масиву в координатних дужках.

Функція ***arange( ),*** після її виклику з програмного пакета ***numpy*** проставленням скороченого символа ***np.*** перед її іменем, створює одновимірний масив із чисел, які послідовно з наростанням йдуть одне за одним, і перше з яких записується в аргументних дужках першим, останнє з

яких має значення на крок менше від вказаного в аргументних дужках другого числа, а третім в аргументних дужках записується число, яким визначається довжина кроку між числами цієї послідовності.

Функція ***linspace( ),*** після її виклику з програмного пакета ***numpy*** проставленням скороченого символа ***np.*** перед її іменем, створює одновимірний масив із чисел, які послідовно з рівномірним наростанням йдуть одне за одним, і перше з яких записується в аргументних дужках першим, останнє із яких записується в аргументних дужках другим числом, а третім в аргументних дужках записується число, яким визначається скільки кроків потрібно зробити для формування цієї послідовності від її початкового значення до кінцевого.

Усе, що викладене вище стосовно структури та виклику функцій ***array( ), arange( ), linspace( )***, продемонстровано **прикладом № 10.**

Приклад № 10

```
In [1]: import numpy as np                    # Виклик пакета numpy під іменем np
In [2]: a1=np.array([0,1,3,2,5])              # Створення масиву зі списку
In [3]: a1                                    # Виклик створеного масиву
Out[3]: array([0, 1, 3, 2, 5])
In [4]: print(a1)                             # Візуалізація створеного масиву
[0 1 3 2 5]
In [5]: L=[[3,4,5],[6,7,8]];L                 # Формування списку з членами-списками
Out[5]: [[3, 4, 5], [6, 7, 8]]
In [6]: a2=np.array(L)                        # Створення масиву зі списку
In [7]: a2                                    # Виклик створеного масиву
Out[7]:
array([[3, 4, 5],
       [6, 7, 8]])
In [8]: a2[0,2]                               # Виклик члена масиву за індексами
Out[8]: 5
In [9]: a2[0][2]                              # Інший варіант виклику члена масиву
Out[9]: 5
In [10]: a2.ndim                              # Визначення розмірності масиву
Out[10]: 2
In [11]: a2.shape                             # Визначення кількості рядків і стовпців
Out[11]: (2, 3)
In [12]: a2.size                              # Визначення кількості членів в масиві
Out[12]: 6
In [13]: len(a2)                              # Визначення кількості рядків в масиві
Out[13]: 2
In [14]: a3=np.arange (1,8,1.5);a3            # Створення одновимірного масиву a3
Out[14]: array ([1. , 2.5, 4. , 5.5, 7. ])
In [15]: print (a3)                           # Візуалізація одновимірного масиву a3
[1.  2.5  4.  5.5  7. ]
In [16]: a4=np.linspace (1,8,10)              # Створення одновимірного масиву a4
```

А тепер охарактеризуємо функцію ***meshgrid(x,y),*** яка є, на відміну від перших трьох, певною мірою спеціалізованою, та продемонструємо її дію прикладом № 11.

Отже, функція ***meshgrid(x,y),*** після її виклику з програмного пакета ***numpy*** проставленням скороченого символа ***np.*** перед її іменем, створює двовимірний масив ***X, Y,*** в якому масив ***X*** має однакові рядки, а масив ***Y*** має однакові стовпці будь-якої послідовності чисел, записаних в аргументних дужках для змінних ***x, y*** у вигляді списків, при цьому рядки масиву ***X*** формуються зі списку, виписаного для змінної ***x,*** з кількістю рядків, що дорівнює кількості членів у змінній ***y,*** а стовпці масиву ***Y*** формуються зі списку, виписаного для змінної ***y,*** з кількістю стовпців, що дорівнює кількості членів у змінній ***x***.

А оператор ***np.mgrid[, ,]***, який є у цьому ж пакеті, але в якому аргументні змінні задаються не в круглих, а в квадратних дужках і тим же способом для кожної змінної, як і у функції ***arange( )***, однак з двокрапками замість ком між складовими для кожної змінної та з комою між діапазонами для кожної з них, створює такий же об'єкт, як і функція ***meshgrid(x,y)***, але у транспонованому вигляді.

Усе, що викладене вище стосовно структури та виклику функції ***meshgrid(x,y)*** та оператора ***np.mgrid[, ,]***, продемонстровано **прикладом № 11.**

Приклад № 11

```
In [1]: import numpy as np                  # Виклик ППП numpy під іменем np
In [2]: x=np.array ([4,5,6])                # Внесення масиву x
In [3]: y=np.array ([7,8])                  # Внесення масиву y
In [4]: X,Y=np.meshgrid (x,y)               # Отримання масивів X,Y
In [5]: print (X)                           # Виклик на екран масиву X
[[4 5 6]
 [4 5 6]]
In [6]: print (Y)                           # Виклик на екран масиву Y
[[7 7 7]
 [8 8 8]]
In [7]: x1=np.linspace (-1,1,3)             # Внесення масиву x1
In [8]: y1=np.linspace (-1,1,3)             # Внесення масиву y1
In [9]: X1,Y1=np.meshgrid (x1,y1)           # Отримання масивів X1,Y1
In [10]: print (X1)                         # Виклик на екран масиву X1
[[-1.  0.  1.]
 [-1.  0.  1.]
 [-1.  0.  1.]]
In [11]: print (Y1)                         # Виклик на екран масиву Y1
[[-1. -1. -1.]
 [ 0.  0.  0.]
 [ 1.  1.  1.]]
In [12]: x2=np.arange (0,6,2)               # Внесення масиву x2
In [13]: y2=np.arange (8,13,2)              # Внесення масиву y2
In [14]: print (x2)                         # Виклик на екран масиву x2
```

```
[0 2 4]
In [15]: print (y2)                          # Виклик на екран масиву y2
[ 8 10 12]
In [16]: X2,Y2=np.meshgrid(x2,y2)            # Отримання масивів X2,Y2
In [17]: print(X2)                           # Виклик на екран масиву X2
[[0 2 4]
 [0 2 4]
 [0 2 4]]
In [18]: print(Y2)                           # Виклик на екран масиву Y2
[[ 8  8  8]
 [10 10 10]
 [12 12 12]]
In [19]: X3,Y3=np.mgrid[0:6:2,8:13:2]       # Масиви X3,Y3 створює  mgrid
In [20]: print(X3)                           # Виклик на екран масиву X3
[[0 0 0]
 [2 2 2]
 [4 4 4]]
In [21]: print(Y3)                           # Виклик на екран масиву Y3
[[ 8 10 12]
 [ 8 10 12]
 [ 8 10 12]]
```

А далі покажемо, які масиви створюють функції

zeros( ), ones( ), eye( ), identity( ), diag( ), fromfunction(.)

Функція ***zeros( ),*** після її виклику з ППП ***numpy*** проставленням скороченого символа ***np.*** перед її іменем, створює ***масив із нулів***, кількість яких визначається числом, проставлених в аргументних дужках, а в разі проставлення в аргументних дужках цієї функції списку з двох чисел перше з них визначатиме кількість рядків з нулями, а друге визначатиме кількість стовпців з нулями.

Функція ***ones( ),*** після її виклику з ППП ***numpy*** проставленням скороченого символа ***np.*** перед її іменем, створює ***масив із одиниць***, кількість яких визначається числом, проставлених в аргументних дужках, а в разі проставлення в аргументних дужках цієї функції списку з двох чисел перше з них визначатиме кількість рядків з одиницями, а друге – кількість стовпців з одиницями.

Функції ***eye( ), identity( ),*** після виклику кожної з них із ППП ***numpy*** проставленням скороченого символа ***np.*** перед іменем кожної, створюють ***квадратний одиничний масив***, розмірність якого визначається числом, проставленим в аргументних дужках, елементами головної діагоналі якого є одиниці, а усі інші елементи є нулями.

Функція ***diag( ),*** після виклику якої з програмного ППП ***numpy*** проставленням скороченого символа ***np.*** перед її іменем, створює ***квадратний масив***, елементами головної діагоналі якого є елементи списку, вписаного в аргументні дужки, а розмірність визначається кількістю елементів цього списку.

Функція ***fromfunction( ),*** після виклику якої з ППП ***numpy*** проставленням скороченого символа ***np.*** перед її іменем, створює ***прямокутний масив***, елементами якого є значення ***лямбда-функції***, вписаної як ***перший аргумент*** в аргументні дужки, а розмірність визначається ***кортежом із двох чисел***, вписаним як ***другий аргумент*** в аргументні дужки.

Усе, що викладене стосовно функцій ***zeros( ), ones( ), eye( ), identity( ), diag( ), fromfunction(.)***, продемонстровано **прикладом № 12.**

Приклад № 12

```
In [1]: import numpy as np                  # Виклик ППП numpy під іменем np
In [2]: a1=np.zeros(4);a1                   # Створення одновимірного масиву нулів
Out[2]: array([0., 0., 0., 0.])
In [3]: a2=np.zeros([2,4]);a2               # Створення двовимірного масиву нулів
Out[3]:
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.]])
In [4]: b1=np.ones(4);b1                    # Створення одновимірного масиву одиниць
Out[4]: array([1., 1., 1., 1.])
In [5]: b2=np.ones([2,3]);b2                # Створення двовимірного масиву одиниць
Out[5]:
array([[1., 1., 1.],
       [1., 1., 1.]])
In [6]: d1=np.eye(3);d1                     # Створення одиничної діагоналі
Out[6]:
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
In [7]: d2=np.identity(3);d2                # Створення одиничної діагоналі
Out[7]:
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
In [8]: d3=np.diag([1,10,100]);d3           # Створення діагонального масиву
Out[8]:
array([[  1,   0,   0],
       [  0,  10,   0],
       [  0,   0, 100]])
In [9]: d4=np.diag([1,3,5],1);d4            # Створення масиву з зсунутою діагоналлю
Out[9]:
array([[0, 1, 0, 0]
       [0, 0, 3, 0],
       [0, 0, 0, 5],
       [0, 0, 0, 0]])
In [10]: d5=np.diag([1,3,5],-1);d5          # Створення масиву з зсунутою діагоналлю
Out[10]:
array([[0, 0, 0, 0],
       [1, 0, 0, 0],
```

```
      [0, 3, 0, 0],
      [0, 0, 5, 0]])
In [11]: def f(i,j):                          # Створення індексної функції f(i,j)
             return i**3+j**3

In [12]: g1=np.fromfunction(f,(3,3));g1       # Створення масиву, сформованого функцією f(i,j)
Out[12]:
array([[ 0.,  1.,  8.],
       [ 1.,  2.,  9.],
       [ 8.,  9., 16.]])
In [13]: g2=np.fromfunction(lambda i,j:\      # Створення масиву лямбда-функцією
         i**3+j**3,(3,3))
In [14]: g2                                   # Виклик на екран масиву лямбда-функції
Out[14]:
array([[ 0.,  1.,  8.],
       [ 1.,  2.,  9.],
       [ 8.,  9., 16.]])
```

На прикладах № 9–№ 12 та в поясненнях до них ми уже виклали технологію створення масивів, а тепер, з використанням інформації, викладеної в роботі [7], покажемо, які операції можна виконувати з масивами. В програмах, створених мовою Python, використовується значна кількість цих операцій, але ми зупинимось лише на деяких із них, яких буде достатньо для досягнення наших цілей. І розглянемо ми ці операції в рамках чотирьох блоків, а саме: 1) блок операцій з елементами одного масиву, 2) блок операцій з елементами двох масивів, 3) блок операцій з обчисленням функцій від змінних, заданих масивами, 4) блок операцій з масивами, трансформованими в матриці.

1.4.2 Операції з елементами одного масиву

У цьому пункті ми розглянемо лише ***операції***:

- ***додавання до*** масиву ***a*** якогось числа ***c,*** тобто ***операцію a+c***, в результаті якої до кожного елемента цього масиву додається це число;
- ***множення*** масиву ***a*** на якесь число ***c,*** тобто ***операцію a*c***, в результаті якої кожен елемент цього масиву множиться на це число;
- ***ділення*** масиву ***a*** на якесь число ***c,*** тобто ***операцію a/c***, в результаті якої кожен елемент цього масиву ділиться на це число;
- ***піднесення*** масиву ***a*** до степеня ***c,*** тобто ***операцію a**c***, в результаті якої кожен елемент цього масиву підноситься до степеня, заданого цим числом;
- ***перетворення одновимірного*** масиву ***a*** в ***багатовимірний*** квадратний масив ***b*** розмірності ***(c, c)***, елементами якого є елементи масиву ***a***, якщо їх кількість ділиться на ***c***, тобто ***операцію a.reshape(c, c)***;
- ***перетворення одноелементного*** масиву ***a*** в ***скаляр b***, тобто ***операцію*** з використанням функції ***np.asscalar(a)***;

- ***перетворення двовимірного*** масиву ***b*** в однойменний ***одновимірний*** з тих же елементів, тобто ***операцію b.shape***;

- ***формування одновимірного масиву b*** з діагональних членів ***двовимірного масиву a,*** тобто ***операцію a.diagonal( )***;

- ***транспонування*** масиву ***a***, тобто ***операцію*** заміни рядків масиву стовпцями з однаковими індексами ***a.transpose( )*** та її двійника ***a.T***;

- ***обчислення суми*** та ***добутку*** усіх членів масиву ***a*** виконанням операцій ***a.sum( ), a.prod( )***;

- ***визначення елемента*** масиву ***a*** з ***найменшим*** числовим значенням та його ***індексу*** виконанням операцій ***a.min( )*** та ***a.argmin( )***;

- ***визначення елемента*** масиву ***a*** з ***найбільшим*** числовим значенням та його ***індексу*** виконанням операцій ***a.max( )*** та ***a.argmax( )***;

- ***обчислення середнього значення*** масиву ***a*** та ***середньоквадратичного відхилення*** від нього виконанням операцій ***a.mean( )*** та ***a.std( )***;

- ***перетворення одновимірного*** масиву ***a*** у вигляді рядка у ***двовимірний*** масив ***b*** у вигляді стовпця виконанням операції ***a[:,None]***.

Використання усіх цих операцій в програмах, створених мовою Python, ***продемонстровано у прикладі № 13.***

Приклад № 13

```
In [1]: import numpy as np                    # Виклик ППП pитру з символом пр
In [2]: a=np.array([1,2,3,4])                 # Внесення масиву a
In [3]: b=a+5                                 # Формування масиву b як суми a+5
In [4]:b                                      # Виклик на екран масиву b
Out[4]: array([6, 7, 8, 9])
In [5]: b1=a-5;b1                             # Формування масиву b1 як різниці a-5
Out[5]: array([-4, -3, -2, -1])
In [6]: b2=a*5;b2                             # Формування масиву b2 як добутку a  на 5
Out[6]: array([ 5, 10, 15, 20])
In [7]: b3=a/5;b3                             # Формування результату  ділення a на 5
Out[7]: array([0.2, 0.4, 0.6, 0.8])
In [8]: b4=a**5;b4                            # Масив як  результат піднесення до степеня
Out[8]: array([   1,   32,  243, 1024], dtype=int32)
In [9]: b5=a.reshape(2,2);b5                  # Формування  двовимірного масиву
Out[9]:
array([[1, 2],
       [3, 4]])
In [10]: a1=np.array([17]);a1                 # Внесення масиву a1 з одним елементом
Out[10]: array([17])
In [11]: b6=np.asscalar(a1)                   # Перетворення масиву a1 в скаляр b6
In [12]: b6                                   # Виклик на екран скаляра b6
Out[12]: 17
In [13]: a3=np.array([[5,6,7],[5,3,1]]);a3    # Створення двовимірного масиву a3
Out[13]:
array([[5, 6, 7],
       [5, 3, 1]])
In [14]: a3.shape=(6,);a3                     # Перетворення масиву a3 в одновимірний
```

```
Out[14]: array([5, 6, 7, 5, 3, 1])
In [15]: a4=np.arange(16).reshape((4,4))      # Створення масиву a4 розміром (4×4)
In [16]: a4                                   # Виклик на екран масиву a4
Out[16]:
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
In [17]: a4.diagonal( )                       # Одновимірна діагоналізація масиву a4
Out[17]: array([ 0,  5, 10, 15])
In [18]: a5=a4.transpose( );a5                # Транспонування масиву a4
Out[18]:
array([[ 0,  4,  8, 12],
       [ 1,  5,  9, 13],
       [ 2,  6, 10, 14],
       [ 3,  7, 11, 15]])
In [19]: a5.T                                 # Повернення до масиву a4
Out[19]:
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
In [20]: a6=np.array([-7,-9,-1,5,10,12]);a6   # Внесення масиву a6
Out[20]: array([-7, -9, -1,  5, 10, 12])
In [21]: a6.sum( )                            # Сума елементів масиву a6
Out[21]: 10
In [22]: a6.prod( )                           # Добуток елементів масиву a6
Out[22]: -37800
In [23]: a6.min( )                            # Найменший елемент масиву a6
Out[23]: -9
In [24]: a6.argmin( )                         # Індекс найменшого елемента в a6
Out[24]: 1
In [25]: a6.max( )                            # Найбільший елемент масиву a6
Out[25]: 12
In [26]: a6.argmax( )                         # Індекс найбільшого елемента в a6
Out[26]: 5
In [27]: a6.mean( )                           # Усереднення масиву a6
Out[27]: 1.6666666666666667
In [28]: a6.std( )                            # Середньоквадратичне відхилення a6
Out[28]: 7.993052538854532
In [29]: a7= np.array([1,5,9]);a7             # Внесення одновимірного масиву a7
Out[29]: array([1, 5, 9])
In [30]: b7=a7[:,None];b7                     # Перетворення a7 у двовимірний масив b7
Out[30]:
array([[1],
       [5],
       [9]])
```

1.4.3 Операції з елементами двох масивів

Із операцій з елементами двох масивів ми розглянемо лише операції:

- ***додавання масивів a*** та ***b*** однакової розмірності**,** тобто ***операцію a+b***, в результаті якої до кожного елемента одного масиву додається елемент з тим же індексом другого масиву (зокрема масивів, один із яких задається рядком, а другий стовпцем);

- ***множення масивів a*** та ***b*** однакової розмірності**,** тобто ***операцію a*b***, в результаті якої кожен елемент одного масиву множиться на елемент з тим же індексом другого масиву;

- ***ділення масивів a*** та ***b*** однакової розмірності**,** тобто ***операцію a/b***, в результаті якої кожен елемент одного масиву ділиться на елемент з тим же індексом другого масиву;

- ***піднесення до степеня масивів a*** та ***b*** однакової розмірності**,** тобто ***операцію a**b***, в результаті якої кожен елемент одного масиву підноситься до степеня, що дорівнює елементові з тим же індексом другого масиву;

- ***порівняння масивів a*** та ***b*** однакової розмірності ***з використанням операторів булевої логіки***, наприклад ***(a < b),*** тобто ***операцію булевої логіки,*** в результаті якої кожен елемент одного масиву порівнюється з елементом з тим же індексом другого масиву і формується логічний масив такої ж розмірності, елементами якого є лише символи логіки ***True*** або ***False***

Використання усіх цих операцій в програмах, створених мовою Python, продемонстровано у ***прикладі № 14.***

Приклад № 14

```
In [1]: import numpy as np                    # Виклик ППП pитру з символом np
In [2]: a=np.array([2,4,6,8])                 # Внесення масиву-рядка a
In [3]: b=np.array([2,1,2,4])                 # Внесення масиву-рядка b
In [4]: a+b                                   # Отримання поелементної суми
Out[4]: array([ 4,  5,  8, 12])
In [5]: c=np.array([[1],[2],[3],[4]]); c      # Внесення масиву-стовпця c
Out[5]:
array([[1],
       [2],
       [3],
       [4]])
In [6]: a+c                                   # Отримання суми масивів a та c
Out[6]:
array([[ 3,  5,  7,  9],
       [ 4,  6,  8, 10],
       [ 5,  7,  9, 11],
       [ 6,  8, 10, 12]])
In [7]: a*b                                   # Отримання поелементного добутку
Out[7]: array([ 4,  4, 12, 32])
In [8]: a/b                                   # Реалізація поелементного ділення
```

```
Out[8]: array([1., 4., 3., 2.])
In [9]: a**b                                  # Поелементне піднесення до степеня
Out[9]: array([   4,    4,   36, 4096], dtype=int32)
In [10]: a==b                                 # Булеве поелементне порівняння
Out[10]: array([ True, False, False, False])
In [11]: a<b                                  # Булеве поелементне порівняння
Out[11]: array([False, False, False, False])
In [12]: a>b                                  # Булеве поелементне порівняння
Out[12]: array([False,  True,  True,  True])
In [13]: print(a>b)                           # Візуалізація результату порівняння
[False  True  True  True]
```

1.4.4 Обчисленням функцій від змінних, заданих масивами

Із операцій з обчисленням функцій від змінних, заданих масивами, ми розглянемо в межах ППП numpy лише такі функції:

- функцію ***np.dot( )*** та її двійника у останніх версіях цього пакета функцію ***np.matmul( ),*** які для одновимірних масивів з однаковою кількістю елементів визначають їх скалярний добуток, а для двовимірних масивів здійснюють їх матричне перемноження;
- функцію ***np.vdot( )***, яка визначає скалярний добуток багатовимірних масивів, попередньо трансформуючи їх до одновимірного вигляду та реалізуючи операцію спряження, якщо елементами масиву є комплексні числа;
- функцію ***np.cross( )***, яка реалізує операцію векторного перемноження одновимірних масивів;
- функцію ***np.outer( )***, яка реалізує операцію зовнішнього перемноження одновимірних масивів;
- функцію ***np.transpose(a)***, яка, як і метод ***a.transpose( )***, здійснює транспонування масиву ***a***;
- функцію ***np.sort(a)***, яка формує масив ***b*** шляхом сортування числового масиву ***a*** за критерієм зростання значень його членів;
- метод ***x.round(k)***, який залишає в дробових числах масиву ***x*** стільки знаків після коми, скільки їх вказано в аргументних дужках, тобто ***k***;
- функцію ***np.piecewise(x,[ ],[ ]),*** якою, задаючи в аргументних дужках відповідні умови, можна формувати новий масив, членами якого будуть лише абсолютні значення чисел, які є елементами масиву ***x***;
- функцію ***np.vectorize(f)***, яка здійснює векторизацію, тобто отримання значень у вигляді списку (чи кортежу) скалярної функції ***f(x)***;
- функцію ***np.cumsum(a)***, яка формує масив, кожен член якого дорівнює сумі усіх попередніх членів породного масиву ***a***;
- функцію ***np.diff(a,n,axis)***, яка обчислює кінцеві різниці ***n-го*** порядку відносно членів масиву ***a*** вздовж осі ***axis***.

Використання усіх цих функцій в програмах, створених мовою Python, продемонстровано у ***прикладі № 15.***

Приклад № 15

```
In [1]: import numpy as np                          # Виклик ППП питру з символом пр
In [2]: a1=np.array([2,4,6,8,10,12])                # Внесення одновимірного масиву a1
In [3]: a2=np.array([-3,-1,0,1,3,5])                # Внесення одновимірного масиву a2
In [4]: np.dot(a1,a2)                               # Скалярний добуток масивів a1 та a2
Out[4]: 88
In [5]: b1=np.array([[2,4,6],[8,10,12]]);b1         # Внесення двовимірного масиву b1
Out[5]:
array([[ 2,  4,  6],
       [ 8, 10, 12]])
In [6]: b2=np.array([[-3,-1],[0,1],[3,5]]);b2       # Внесення двовимірного масиву b2
Out[6]:
array([[-3, -1],
       [ 0,  1],
       [ 3,  5]])
In [7]: np.dot(b1,b2)                               # Матричне перемноження масивів b1, b2
Out[7]:
array([[12, 32],
       [12, 62]])
In [8]:np.matmul(a1,a2)                             # Скалярний добуток масивів a1 та a2
Out[8]: 88
In [9]: np.matmul(b1,b2)                            # Матричне перемноження масивів b1, b2
Out[9]:
array([[12, 32],
       [12, 62]])
In [10]: a3=np.array([2+4j,6+8j,10+12j]);a3         # Внесення масиву комплексних чисел a3
Out[10]: array([ 2. +4.j,  6.+8.j, 10.+12.j])
In [11]: a4=np.array([-3-1j,0+1j,3+5j]);a4          # Внесення масиву комплексних чисел a4
Out[11]: array([-3.-1.j,  0.+1.j,  3.+5.j])
In [12]: np.vdot(a3,a4)                             #Скалярний добуток a3, a4 (спряження a3)
Out[12]: (88+30j)
In [13]: b4=np.array([[-3-1j], [0+1j],[3+5j]]);b4   # Внесення  двовимірного масиву b4
Out[13]:
array([[-3.-1.j],
       [ 0.+1.j],
       [ 3.+5.j]])
In [14]: np.vdot(a3,b4)                             #Скалярний добуток a3,b4 (спряження a3)
Out[14]: (88+30j)
In [15]: np.dot(a3,b4)                              # Скалярний добуток a3,b4 (без спряження)
Out[15]: array([-40.+78.j])
In [16]: a5=np.array([3,5]);a5                      # Внесення одновимірного масиву a5
Out[16]: array([3, 5])
In [17]: b5=np.array([7,9]);b5                      # Внесення одновимірного масиву b5
Out[17]: array([7, 9])
In [18]: np.cross(a5,b5)                            # Векторне перемноження масивів a5, b5
Out[18]: array(-8)
In [19]: a6=np.array([3,5,7])                       # Внесення одновимірного масиву a6
In [20]: b6=np.array([7,9,11])                      # Внесення одновимірного масиву b6
```

```
In [21]: np.cross(a6,b6)                    # Векторне перемноження масивів a6, b6
Out[21]: array([-8, 16, -8])
In [22]: a7=np.array([5,10])                # Внесення двоелементного масиву a7
In [23]: b7=np.array([2,3,4])               # Внесення триелементного масиву b7
In [24]: np.outer(a7,b7)                    # Зовнішнє перемноження масивів a7, b7
Out[24]:
array([[10, 15, 20],
       [20, 30, 40]])
In [25]: np.outer(a6,b6)                    # Зовнішнє перемноження масивів a6, b6
Out[25]:
array([[21, 27, 33],
       [35, 45, 55],
       [49, 63, 77]])
In [26]: np.transpose(np.outer(a6,b6))      # Транспонування попереднього результату
Out[26]:
array([[21, 35, 49],
       [27, 45, 63],
       [33, 55, 77]])
In [27]: np.outer(a6,b6).T                  # Транспонування результату інакше
Out[27]:
array([[21, 35, 49],
       [27, 45, 63],
       [33, 55, 77]])
In [28]: a8=np.array([8,3,-1,-6,5,2,-3,7])  # Внесення одновимірного масиву a8
In [29]: b8=np.sort(a8)                     # Сортування масиву a8
In [30]: b8                                 # Виведення на екран відсортованого
                                              масиву b8
Out[30]: array([-6, -3, -1,  2,  3,  5,  7,  8])
In [31]: a9=np.array([5.615,-1.12,8.435,2.10,-1,0])  # Внесення масиву чисел a9
In [32]: b9=a9.round(1)                     # Укорочення чисел в масиві a9
                                              з залишенням лише одного знаку після коми
In [33]: b9                                 # Виведення на екран укороченого масиву b9
Out[33]: array([ 5.6,  -1.1,   8.4,   2.1, -1.0])
In [34]: np.piecewise(b8,[b8<0,b8>=0], \    # Модулізація масиву чисел b8
   [lambda b8:-b8,lambda b8: b8])
Out[34]: array([ 6,3,1,2,3,5,7,8])
In [35]: def f(x):                          # Формування власної функції f(x)
             return x*3-2                   # Визначення тіла власної функції f(x)

In [36]: x=np.arange(1,7)                   # Внесення масиву значень аргументу x
In [37]: fvec=np.vectorize(f)               # Формування операції векторизації
In [38]: fvec(x)                            # Векторизація власної функції f(x)
Out[38]:
array([ -1,   6,  25,  62, 123, 214])

In [39]: l1=lambda x: x**3-2                # Формування лямбда-функції l1 та її тіла
In [40]: l1vec=np.vectorize(l1)             # Формування операції векторизації
```

```
In [41]: l1vec(x)                          # Векторизація лямбда-функції l1
Out[41]:
array([ -1,   6,  25,  62, 123, 214])

In [42]: x2=np.linspace(0,6,7); x2         # Внесення масиву значень аргументу x2
Out[42]: array([0., 1., 2., 3., 4., 5., 6.])
In [43]: y2=np.linspace(-3,3,7); y2        # Внесення масиву значень аргументу y2
Out[43]: array([-3., -2., -1.,  0.,  1.,  2.,  3.])
In [44]: l12=lambda x2,y2: x2**3+y2**2     # Формування лямбда-функції l12 та її тіла
In [45]: l12vec=np.vectorize(l12)          # Формування операції векторизації
In [46]: l12vec(x2,y2)                     # Векторизація лямбда-функції l12
Out[46]:
array([  9.,   5.,   9.,  27.,  65., 129., 225.])

In [47]: a10=np.array([1,2,3,4,5,6,7])     # Внесення одновимірного масиву a10
In [48]: np.cumsum(a10)                    # Масив із сум попередніх членів a10
Out[48]:
array([ 1,  3,  6, 10, 15, 21, 28], dtype=int32)
In [49]: np.diff(a10)                      #  Масив із різниць сусідніх членів a10
Out[49]:
 array([1, 1, 1, 1, 1, 1])
In [50]: np.diff([[1,2,3,2,1],[5,1,4,2,3],\   # Масив із різниць сусідніх
[1,2,3,4,5]],axis=0)                          членів двовимірного масиву вздовж осі 0
Out[50]:
array([[ 4, -1, 1, 0, 2],
        [-4, 1,-1, 2, 2]])
In [51]: np.diff([[1,2,3,2,1],[5,1,4,2,3],\   # Масив із різниць сусідніх
   [1,2,3,4,5]],axis=1)                       членів двовимірного масиву вздовж осі 1
Out[51]:
array([[ 1,  1, -1, -1],
      [-4,  3, -2,  1],
      [ 1,  1,  1,  1]])
```

1.4.5 Трансформація масивів у матриці

Із операцій з масивами, трансформованими в матриці, ми розглянемо ***лише операції,*** що виконуються в межах ППП ***питру*** з об'єктами класу ***matrix*** (скорочено ***mat*)**, для внесення в який масив ***a*** трансформується в матрицю ***A*** того ж розміру, на яку розповсюджуються усі операції лінійної алгебри.

Ці операції в програмах, створених мовою Python, продемонстровано у ***прикладі № 16.***

Приклад № 16

```
In [1]: import numpy as np                 # Виклик ППП питру з символом np
In [2]: A1=np.mat('1 2;3 4');A1            # Трансформація стрічки в матрицю A1
Out[2]:
```

```
matrix([[1, 2],
        [3, 4]])
In [3]: B1=np.mat([[5,6],[7,8]]);B1          # Трансформація списку .в матрицю B1
Out[3]:
matrix([[5, 6],
        [7, 8]])
In [4]: A1+B1                                # Обчислення суми матриць A1 та B1
Out[4]:
matrix([[ 6,  8],
        [10, 12]])
In [5]: A1*B1                                # Перемноження матриць A1 та B1
Out[5]:
matrix([[19, 22],
        [43, 50]])
In [6]: A1**2                                # Піднесення до степеня .матриці A1
Out[6]:
matrix([[ 7, 10],
        [15, 22]])
In [7]: A1.I                                 # Обчислення матриці, оберненої
Out[7]:                                        до матриці A1
matrix([[-2. ,  1. ],
        [ 1.5, -0.5]])
In [8]: A1.T                                 # Транспонування матриці A1
Out[8]:
matrix([[1, 3],
        [2, 4]])
```

1.4.6 Цикли

При розв'язуванні різного роду рівнянь, особливо нелінійного характеру, виникає потреба в застосуванні алгоритмів наближених обчислень за одними і тими ж формулами, що приводить нас до реалізації циклічних алгоритмів обчислень, які цикл за циклом наближають нас до отримання розв'язку рівняння з заданою точністю.

Циклічні алгоритми обчислень реалізуються з використанням операторів управління обчислювальним процесом, про які мова піде нижче.

Одразу ж відзначимо, що, пояснюючи суть алгоритмів циклічних обчислень, що реалізуються з використанням операторів управління обчислювальним процесом, ми будемо спиратись на матеріали, викладені в роботі [7].

І почнемо пояснювати суть алгоритмів циклічних обчислень, що реалізуються з використанням операторів управління обчислювальним процесом, ми з нагадування про те, що всі команди програми виконуються послідовно в порядку їх запису, але цей порядок може бути примусово змінений за допомогою внесення в програму операторів циклу та умовних інструкцій.

У мові Python є лише два оператори циклу:

1) оператор ***for,*** що в перекладі означає ***для;***
2) оператор ***while,*** що в перекладі означає ***доки***.

For-цикл використовують, коли відомою є його довжина.

While-цикл використовують, коли невідомо, якої довжини він має бути.

Характерною особливістю оператора циклу ***for*** є те, що він працює або з функцією ***range(k),*** що в перекладі означає ***діапазон цілих чисел від 0 до k-1,*** або з функцією ***list( ),*** що в перекладі означає ***список атрибутів, поміщених в аргументні дужки.***

Функція range(k) розміщується в одному рядку з командою, що містить в собі оператор ***for*** , але після нього.

Функція list( ) задається в рядку, що передує рядку з командою, яка містить в собі оператор ***for***.

For-цикл повторюватиметься стільки разів, скільки чисел знаходиться в діапазоні чи скільки атрибутів списку поміщено в аргументні дужки.

Варіанти застосування оператора циклу ***for*** наведені у ***прикладі № 17***.

Приклад № 17

```
In [1]: import numpy                          # Виклик ППП numpy
In [2]: from numpy import*                    # Доступ до усіх функцій ППП numpy
In [3]: for i in range(4):                    # Для i з діапазону від 0 до 3
            print(i**2)                         друкувати i**2 в стовпчик

0
1
4
9
In [4]: for i in range(4):                    # Для i з діапазону від 0 до 3
            print(i**2,end=" ")                 друкувати i**2 в рядок

0 1 4 9
In [5]: for x in range(3):                    # Для x з діапазону від 0 до 2
            print('Слава Україні!')             друкувати Слава Україні! в стовпчик

Слава Україні!
Слава Україні!
Слава Україні!
In [6]: for x in range(3):                    # Для x з діапазону від 0 до 2
            print('Слава Україні!',end=" ")     друкувати Слава Україні! в рядок

Слава Україні! Слава Україні! Слава Україні!
In [7]: L=[2,3,4]                             # Внесення списку L
In [8]: for x in L:                           # Для x зі списку L
            print(2*x+x**3)                     друкувати 2x + x**3 в стовпчик

12
33
72
```

```
In [9]: for x in L:                          # Для x зі списку L
            print(2*x+x**3,end=" ")            друкувати 2x + x^3 в рядок

12 33 72
```

Для оператора циклу ***while*** характерним є те, що в рядку з командою, що містить в собі оператор циклу ***while,*** обов'язково поміщується умова, виконання якої дозволяє продовжувати бути в циклі змінній, початкове значення якої задається в рядку, що передує рядку з командою, яка містить в собі оператор циклу ***while.***

Варіанти з оператором циклу ***while*** наведені у ***прикладі № 18***.

Приклад № 18

```
In [1]: import numpy                         # Виклик ППП numpy
In [2]: from numpy import*                   # Доступ до усіх функцій ППП numpy
In [3]: i=0                                  # Внесення початкових даних
In [4]: while i<4:                           # Формування циклу «доки i<4
            i=i+2                              до значення i додавати 2
            print(i**3)                        і друкувати значення i^3 в стовпчик»

8
64
In [5]: i=0
In [6]: while i<4:                           # Формування циклу «доки i<4
            i=i+2                              до значення i додавати 2
            print(i**3,end=" ")                і друкувати значення i^3 в рядок»

8 64
In [7]: i=3                                  # Внесення початкових даних
In [8]: q=5                                    у вигляді параметрів i та q
In [9]: while i<6 and q<8:                   # Формування циклу «доки i<6 і q<8
            i=i+2                              до значень i, q додавати 2 і друкувати
            q=q+2                              значення пар (i, q) в стовпчик»
            print(i,q)

5 7
6 8
7 9
In [10]: i=4;q=7                             # Внесення початкових пар i,q
In [11]: while i<6 or q<10:                  # Формування циклу «доки i<6 або q<10
             i=i+1                             до значень i, q додавати одиницю
             q=q+1                             і друкувати значення пар (i, q) в стовпчик»
             print(i,q)

5 8
6 9
7 10
```

```
In [12]: i=4;q=7                          # Внесення початкових пар i, q
In [13]: while i<6 or q<10:               # Формування циклу «доки i<6 або q<10
             i=i+1                          до значень i, q додавати одиницю
             q=q+1                          і друкувати значення пар (i, q) в рядок»
             print((i,q),end=" ")

(5, 8) (6, 9) (7, 10)
In [14]: L=[5,4,3,1,2,3,4]                # Внесення списку L
In [15]: i=6                              # Внесення початкового індексу
In [16]: while i:                         # Формування циклу «доки i>0 друкувати
             print(L[i],end=" ")            список в оберненому порядку»,
             i-=1                           на що вказує умова i-=1»,
4 3 2 1 3 4                                 але при цьому втрачається перший член
                                              списку, внесеного на початку
```

А далі викладемо суть «умовних інструкцій»:

1) інструкція ***if***, яка містить в собі ще й приховане ***then*** і є інструкцією ***if-then***, що означає ***якщо_, то_***, та вимагає виконання команди, яка йде другою, якщо умова, що задана першою командою, виконується;

2) інструкція ***if-else***, яка містить в собі ще й приховане ***then*** і є інструкцією ***if-then-else***, що означає ***якщо, то, інакше***, та вимагає виконання команди, яка йде другою, якщо умова, що задана першою командою, виконується, і вимагає виконання команди, яка йде третьою, якщо умова, задана першою командою, не виконується;

3) інструкція ***if-elif-else***, яка містить в собі ще й двічі приховане ***then*** і є інструкцією ***if-then-elif-then-else***, що означає ***якщо_, то_, або якщо_, то_, інакше_***, та вимагає виконання команди, яка йде другою, якщо умова, що задана першою командою, виконується, вимагає виконання команди, яка йде четвертою, якщо умова, що задана першою командою, не виконується, але виконується умова, що задана третьою командою, і вимагає виконання команди, яка йде п’ятою, якщо не виконуються умови, що задані першою та третьою командами.

Як працюють ці ***умовні інструкції*** продемонстровано у ***прикладі № 19***.

Приклад № 19

```
In [1]: import numpy                      # Виклик ППП numpy
In [2]: from numpy import*                # Доступ до усіх функцій numpy
In [2]: i=0                               # Внесення початкового індексу
In [3]: s1='ліфт працює'                  # Внесення першої стрічки умов
In [4]: s2='ліфт не працює'               # Внесення другої стрічки умов
In [5]: s3='йду пішки'                    # Внесення умови на виході
In [6]: for поверх in range(10):          # Формування циклу для поверху і із 9
             if поверх <5 and s1:           якщо поверх <5 і ліфт
                поверх=поверх+1             працює, оглядаючи поверхи,
                print('поверх %s=%s' %(поверх, s3))   приймаю рішення, що на:
        …….
```

```
поверх 1=йду пішки
поверх 2=йду пішки
поверх 3=йду пішки
поверх 4=йду пішки
поверх 5=йду пішки
In [7]: for поверх in range(10):                    # Формування циклу для поверху і з 9
            if поверх <5 and s1:                      якщо поверх <5 і ліфт
               поверх=поверх+1                        працює, оглядаючи поверхи,
               print('поверх %s=%s' %(поверх, s3))    приймаю попереднє рішення,
            elif поверх >5 and s2:                    або, якщо поверх >5 і ліфт
               поверх=поверх+1                        не працює, оглядаючи поверхи,
               print('поверх %s=%s' %(поверх, s3))    приймаю рішення, що на:

поверх 1=йду пішки
поверх 2=йду пішки
поверх 3=йду пішки
поверх 4=йду пішки
поверх 5=йду пішки
поверх 6=йду пішки
поверх 7=йду пішки
поверх 8=йду пішки
поверх 9 =йду пішки
```

А на завершення матеріалу, присвяченого циклам та умовним інструкціям, звертаємо увагу на те, що в тілі кожного з циклів (як з оператором ***for,*** так і з оператором ***while***) можуть міститись як умовні інструкції, так і оператори ***break*** та ***continue.***

Оператор ***break*** завершує цикл достроково і передає право виконуватись команді, яка розміщена одразу після завершальної команди циклу.

Оператор ***continue*** дозволяє виконувати наступну ітерацію циклу до завершення попередньої.

Як працюють оператори ***break*** та ***continue*** продемонстровано у ***прикладі № 20.***

Приклад № 20

```
In [1]: import numpy                          # Виклик ППП numpy
In [2]: from numpy import*                    # Доступ до усіх функцій numpy
In [2]: s1='втомився'                         # Внесення першої стрічки умов
In [3]: s2='пішов дощ'                        # Внесення другої стрічки умов
In [4]: відстань=0                            # Внесення початку відліку
In [5]: while відстань < 15:                  # Для циклу «доки відстань <15»
            print(відстань, end=" ")            друкувати її значення в рядок,
            if відстань==5 and s1:              а, якщо відстань дорівнює 5 і
               break                            «втомився», то завершити цикл
            elif відстань==10 and s2:           або, якщо відстань 10 і «пішов дощ»
               break                            то теж завершити цикл,
```

```
        else:                                  інакше до змінної «відстань»
            відстань=відстань+1                додавати одиницю»

0 1 2 3 4 5                                    # Результат руху за таких умов
In [6]: while відстань < 15:                   # Формування циклу «доки відстань
            print(відстань, end=" ")             <15, друкувати її значення в рядок
            if відстань==10 and s1:              а, якщо відстань 10 і «втомився»,
                break                            то завершити цикл достроково,
            elif відстань==5 and s2:             або, якщо відстань 5 і «пішов дощ»,
                break                            то теж завершити цикл,
            else:                                інакше
                відстань=відстань+1              до змінної «відстань» додавати 1

5                                              # Результат руху за таких умов
In [7]: while відстань < 15:                   # Для циклу «доки відстань <15
            print(відстань, end=" ")             друкувати її значення в рядок,
            if відстань==10 and s1:              а, якщо відстань 10 і «втомився»,
                break                            то завершити цикл достроково,
            elif відстань==5 and s2:             або, якщо відстань 5 і «пішов дощ»,
                break                            то теж завершити цикл,
            else:                                інакше
                відстань=відстань+1              до змінної «відстань» додавати 1

5 6 7 8 9 10                                   # Результат руху за таких умов
In [8]: for відстань in range(15):             # Цикл «для відстані із відрізка [0,14],
            if 5 < відстань < 10:                якщо відстань більша 5 або менша 10,
                continue                         перейти до наступної операції в циклі,
            print(відстань,end=" ")              і друкувати її значення в рядок

0 1 2 3 4 5 10 11 12 13 14                     # Результат руху за таких умов
```

1.4.7 Генерація випадкових чисел

У процесі комп'ютерного моделювання об'єктів з випадковими змінами їх параметрів чи стохастичними процесами в них для реалізації цих випадковостей потрібно «підмішувати» до їх моделей імпульси випадкового характеру, для генерації яких ***в Python-програмах*** використовують ***модуль random,*** для виклику функцій з якого, як правило, застосовується символ ***rnd***, що присвоюється цьому модулю після його виклику.

І якщо цей модуль викликається з ***ППП numpy***, то спочатку потрібно викликати сам ППП numpy, а потім викликати цей модуль, використовуючи ім'я ***numpy.random***, або ***np.random***, якщо пакету ***numpy*** уже присвоєно символ ***np.***

В модулі ***numpy.random*** в ППП ***numpy*** нас цікавитимуть лише функції, які генерують масиви випадкових чисел. І почнемо ми їх характеристику з функції, яка у варіанті:

- ***np.random.sample ( )*** генерує ***одне раціональне число*** з ***відрізку [0,1]***;

- ***np.random.sample (k)*** та у варіанті ***np.random.random (k)***, а також у варіанті ***np.random.rand (k)*** генерує ***масив із k*** раціональних випадкових чисел з ***відрізку [0,1]***;

- ***np.random.sample ((k,m))*** та у варіанті ***np.random.rand((k,m))*** генерує ***двовимірний масив*** раціональних випадкових чисел з ***відрізку [0,1]***, який містить ***k рядків*** та ***m стовпців***;

- ***np.random.randint (p,q,k)*** генерує масив із ***k*** цілих випадкових чисел з діапазону (***p,q)***;

- ***np.random.randint (p,q,(k,m))*** генерує двовимірний масив цілих випадкових чисел із діапазону (***p,q)***, який містить ***k*** рядків та ***m*** стовпців;

- ***np.random.uniform (a,b,k)*** генерує одновимірний масив із ***k*** випадкових раціональних чисел, заданих на відрізку ***[a,b]***;

- ***np.random.uniform (a,b,(k,m))*** генерує двовимірний масив раціональних випадкових чисел із відрізка ***[a,b]***, який містить ***k*** рядків та ***m*** стовпців.

Як генерує випадкові числа кожна з вищезгаданих функцій продемонстровано ***у прикладі № 21***.

Приклад № 21

```
In [1]: import numpy as np                  # Виклик ППП numpy як np
In [2]: np.random.sample( )                 # Генерація одного числа з діапазону [0,1]
Out[2]: 0.4203889601376536
In [3]: np.random.sample(2)                 # Генерація двох чисел із діапазону [0,1]
Out[3]: array([0.28426121, 0.74714427]),
In [4]: np.random.sample((3,2))             # Генерація двовимірного
Out[4]:                                       масиву раціональних
array([[0.87792665, 0.50704843],              випадкових чисел із діапазону [0,1]
       [0.93988959, 0.07967668],              з трьома рядками та
       [0.10346116, 0.43782785]])             двома стовпцями
In [5]: np.random.random(2)                 #Генерація двох чисел із діапазону [0,1]
Out[5]: array([0.10687142, 0.61193353])
In [6]: np.random.rand(2)                   #Генерація двох чисел із діапазону [0,1]
Out[6]: array([0.45619613, 0.78303445]),
In [7]: np.random.rand(3,2)                 # Генерація двовимірного
Out[7]:                                       масиву раціональних
array([[0.89741184, 0.82107899],              випадкових чисел із діапазону [0,1]
       [0.62675848, 0.87184645],              з трьома рядками та
       [0.72502582, 0.19836619]])             двома стовпцями
In [8]: np.random.randint (3,21,3)          # Генерація масиву з трьох
Out[8]: array([13, 8, 11])                    цілих чисел із діапазону (3,21)
In [9]: np.random.randint (3,21,(3,2))      # Генерація двовимірного
Out[9]:                                       масиву із випадкових
array([[ 8,  8],                              цілих чисел із діапазону
       [ 7, 17],                              (3,21) з трьома рядками
       [ 4, 14]])                             та двома стовпцями
In [10]: np.random.uniform(3,21,3)          # Генерація масиву з трьох
```

```
Out[10]: array([ 3.63378103, 17.82438909,
 5.26676606])
In [11]: np.random.uniform(3,21,(2,3))
Out[11]:
array(
[[ 4.14709428,  4.38411862, 11.56849929],
[19.87322246, 18.28741987, 17.48519693]])
```

раціональних випадкових чисел із діапазону (3,21)
Генерація двовимірного масиву з випадкових раціональних чисел із діапазону (3,21) з двома рядками і трьома стовпцями

1.4.8 Файли

В усіх, наведених вище 21 прикладах, у процесі створення Python-програм ми використовували командні рядки, оскільки, реалізуючи програми, створені у такий спосіб, ми мали можливість одразу пояснювати і суть кожної програмної команди, записаної в командному рядку, і безпосередньо після кожної команди виводити на екран результат її виконання, що сприяє кращому розумінню для чого була використана кожна конкретна команда, і чому використана саме вона. Додатковою зручністю використання командних рядків у випадку створення програми було ще й те, що в разі виникнення помилки при формуванні конкретної команди, одразу ж після цього командного рядка на екрані висвітлювалось пояснення, в чому ж полягає суть допущеної помилки, що підвищувало навчальний ефект, бо виправлення усвідомлених помилок дозволяє студентам краще сприймати викладений матеріал.

Але ***громіздкі програми*** доцільніше складати ***у вигляді файлу***, який у разі використання консолі ***Spyder*** записується у лівій частині екранного вікна комп'ютера – команда за командою, але без їх нумерації, і в який можна вносити виправлення помилок в конкретному рядку, не переписуючи усю програму спочатку та не псуючи її вигляд на екрані комп'ютера.

Про наявність помилок у якомусь рядку програми у цьому випадку ***сигналізуватиме червона мітка***, яка з'являтиметься з лівого боку цього рядка і яка світитиме своїм червоним кольором до тих пір, поки помилка не буде виправлена.

А в разі, ***якщо*** набрана ***команда*** написана правильно, але далі ***не використовується***, то з лівого боку цього рядка засвітиться ***жовта мітка***, яка теж світитиме своїм жовтим кольором до тих пір, поки в програмі не з'явиться інша команда, яка буде з цією командою взаємодіяти.

Щоб почати створювати файл, потрібно у верхньому рядку-меню, що розміщений над лівим екранним полем найвище, ***натиснути іконку File (Файл***), після чого відкриється вертикально розміщене меню, в якому ***вибрати іконку New file (Новий файл***), після натиснення на яку ліве екранне поле очиститься від того файлу, який там було створено раніше, і на ньому можна буде набирати новий файл, формуючи у ньому команду за командою потрібну програму, виправляючи в процесі набору допущені помилки.

Для перевірки правильності роботи створеної у файлі програми потрібно у тому ж верхньому рядку-меню, в якому ми знаходили іконку

File, ***вибрати 5-ту іконку Run (Дія)***, натиснувши на яку, побачимо знову ж таки вертикальне меню, в якому виберемо ***і натиснемо ту ж таки іконку Run, біля якої розміщено зелений трикутник***. Наша програма запрацює і якщо вона реалізує лише розрахунки, то їх результат ми побачимо в правому нижньому екранному вікні, а якщо програма і графік створює, то його ми побачимо у правому верхньому вікні.

Переконавшись, що програма працює правильно, ми знову ***натискуємо на іконку File*** у верхньому рядку-меню, а потім у вертикальному меню, що відкриється, виберемо і ***натиснемо іконку Save as.*** Після цього нам відкриється файловий провідник, в якому ми ***дамо ім'я нашому файлу***, пам'ятаючи, що в імені файлу не має бути проміжків між символами, ***та натиснемо на іконку «Зберегти».***

Під вибраним нами ім'ям наш файл зберігатиметься в пам'яті комп'ютера, а його текст на екрані можна стерти, аби підготувати екранне поле для набрання нового файлу. В разі, якщо ми ***побажаємо знову викликати*** створений і занесений в пам'ять комп'ютера файл, то у вертикальному меню під іконкою ***File*** верхнього рядка-меню виберемо і ***натиснемо іконку Open (Відкрити).*** Після цього на екрані з'явиться файловий провідник і нам буде запропоновано у його командному полі ввести ім'я потрібного нам файлу. Після внесення імені файлу усі його командні рядки одразу ж з'являться в лівому екранному вікні комп'ютера, і ми отримаємо можливість або використовувати знову програму, записану у цьому файлі, або ввести в неї потрібні зміни.

Прикладів з поясненням викладеного вище у цьому випадку ми наводити не будемо, оскільки кожну з програм, наведену в уже викладених вище 21 прикладах, можна переписати у вигляді файлу в лівому вікні консолі ***Spyder***, «опустивши» номери командних рядків. А як іменувати створений файл, записувати його в пам'ять комп'ютера та викликати з пам'яті на екран ми вище вже описали.

Методичні рекомендації до підрозділу 1.4. Завершуючи викладення матеріалу цього підрозділу викладачеві МЗКО обов'язково потрібно ще раз звернути увагу студентів на те, що:

1) масиви даних – це атрибути саме ***ППП питру***, тож якщо потрібно здійснювати комп'ютерні обчислення з непоодинокими вхідними даними, виклик ***ППП питру*** є обов'язковим;

2) алгебраїчні операції з масивами, ще не трансформованими в матриці, суттєво відрізняються від алгебраїчних операцій з матрицями;

3) в ***ППП питру*** як власна функція, так і ламбда-функція у процесі задання значень її аргументу у вигляді масиву буде обчислювати значення функції лише при одному значенні аргументу, а щоб обчислити її значення при всіх значеннях аргументу, заданих масивом, необхідно здійснити її векторизацію;

4) в разі, коли ми хочемо бачити результат на кожному кроці, то програму потрібно створювати з використанням командного рядка, а в разі, коли ми хочемо мати можливість вносити правки в уже створену програму, то програму потрібно створювати у вигляді файлу;

5) лише при створенні програми у вигляді файлу на рисунках зберігаються усі надписи і позначення, а при створенні програми з використанням командного рядка кожна наступна команда стирає з екрана результат виконання попередньої;

6) ***for-цикл*** потрібно використовувати, коли відоме число циклів, а коли число циклів невідоме, то – ***while-цикл***;

7) для генерації випадкових імпульсів з одиничного діапазону і з довільного діапазону потрібно використовувати різні оператори.

1.5 Побудова графіків

У цьому підрозділі ми пояснимо, як, реалізуючи ***Python-програми***, будувати ***графіки і діаграми*** на координатній площині, та як будувати ***поверхні*** в тривимірному просторі. Для цього ми використаємо ***ППП matplotlib.***

Викладений у цьому підрозділі матеріал нами створено з використанням більш розлогого його викладення у роботі [7]. Тож в разі, якщо викладеного нами матеріалу комусь із читачів буде недостатньо для реалізації їх графічних задач, їм потрібно звертатись до базової і для нас роботи [7], в якій вони знайдуть додаткову інформацію стосовно графічних зображень, що доповнюватиме ту, яка викладена нами.

1.5.1 Побудова графіків на площині з використанням ППП matplotlib

Як ми уже відзначали у вступній частині, ***графічна реалізація Python-програм*** зосереджена, переважно, ***у ППП matplotlib***, тому розпочинати процес побудови графіків потрібно з виклику саме цього програмного пакета.

Але, оскільки ***ППП matplotlib працює з масивами даних, які є об'єктами ППП numpy***, то обов'язково після виклику ***ППП matplotlib*** потрібно викликати і ***ППП numpy***, задаючи йому скорочення у вигляді символа ***np***.

Основним модулем в ППП matplotlib, в якому зосереджені функції реалізації ***графіки***, є ***matplotlib.pyplot***, який у скороченому вигляді позначають символом ***plt***.

Математична ***функція***, ***графік*** якої ***будується програмною функцією plot(),*** до символьного позначення якої перед нею через крапку потрібно додавати скорочений символ ***plt*** модуля ***matplotlib.pyplot***, ***задається*** або ***списком*** числових значень цієї математичної функції в аргументних дужках програмної функції, або ***символами*** попередньо заданих формулою математичної ***функції та масиву її аргументу у формі range( ), arange( ) або linspace( ).***

А команда ***plt.show( )*** викликатиме графік на екран у його правому верхньому вікні, який в деяких варіантах модуля може проєктуватись і без її застосування.

Натисненням на піктограму ***«дискетка»***, яка розміщена першою у лінійці меню над графіком, побудований ***графік*** буде занесено ***у пам'ять*** комп'ютера ***як об'єкт Figure*** з поточним номером.

Потрібно пам'ятати, що у разі числового задання математичної функції у вигляді списку її значень завжди мається на увазі, що аргумент цієї математичної функції приймає цілочислові значення, що починаються з нуля.

Викладене вище проілюстровано в прикладах № 22, № 23 та № 24.

Приклад № 22 (Python-програма побудови графіка функції, заданої списком) (результат – на рис. 1.1)

```
In [1]: import matplotlib                    # Виклик ППП matplotlib
In [2]: import numpy as np                   # Виклик ППП numpy як np
In [3]: import matplotlib.pyplot as plt      # Виклик модуля matplotlib.pyplot як plt
In [4]: plt.plot([2,-1,1,0,4])               # Побудова графіка для функції-списку
```

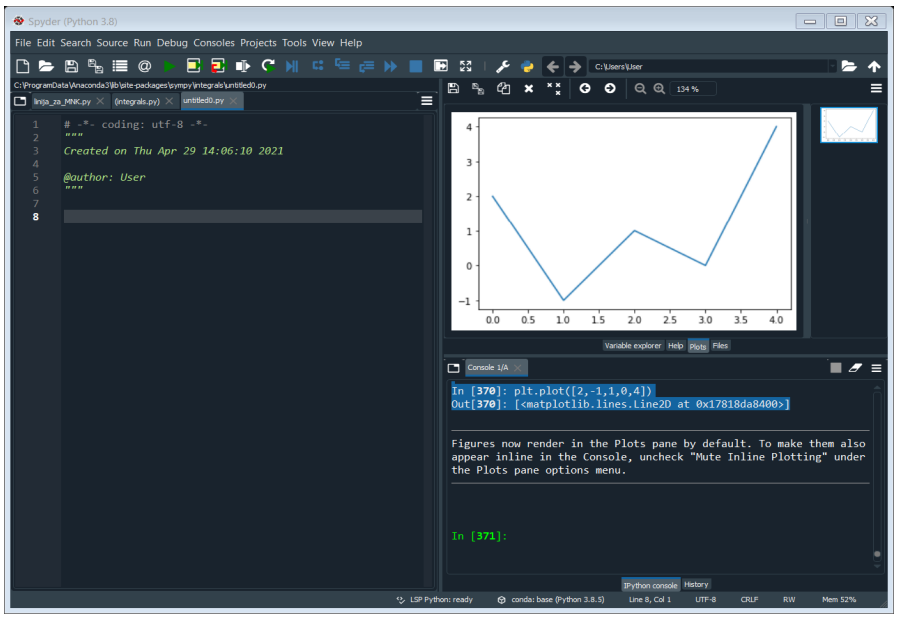


Рисунок 1.1 – Екран комп'ютера з фігурою 1, на якій зображено графік математичної функції, заданої масивом у формі списку

Приклад № 23 (Python-програма побудови графіка функції, заданої циклом) (результат – на рис. 1.2)

```
In [1]: import matplotlib                    # Виклик ППП matplotlib
In [2]: import numpy as np                   # Виклик ППП numpy як np
In [3]: import matplotlib.pyplot as plt      # Виклик модуля matplotlib.pyplot як plt
In [4]: x= range(20)                         # Внесення масиву значень аргументу x
In [5]: y=[np.exp(-t) for t in x]            # Обчислення в циклі значень функції y
In [6]: plt.plot(list(x),y)                  # Побудова графіка для функції з циклу
```

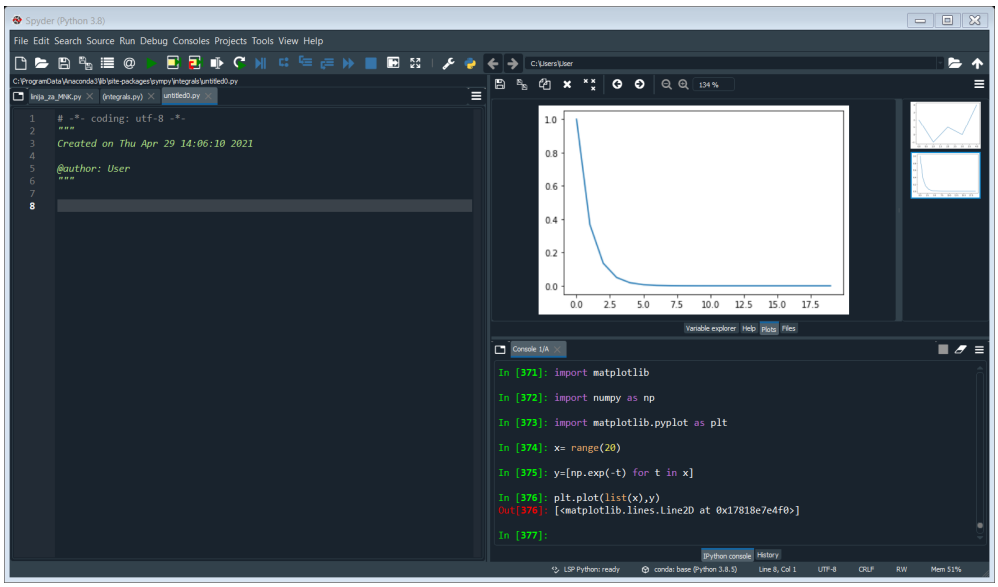


Рисунок 1.2 – Екран комп'ютера з фігурою 2, на якій зображено графік математичної функції, заданої масивом у формі циклу

Приклад № 24 (Python-програма побудови суміщених графіків двох функцій) (результат – на рис. 1.3):

```
In [1]: import matplotlib                    # Виклик ППП matplotlib
In [2]: import numpy as np                   # Виклик ППП numpy як np
In [3]: import matplotlib.pyplot as plt      # Виклик модуля matplotlib.pyplot як plt
In [4]: x=np.linspace(0,3,100)               # Внесення масиву значень аргументу x
In [5]: y1,y2=np.cos(x),np.sin(x)            # Обчислення значень функцій y1,y2
In [6]: plt.plot(x,y1,x,y2)                  # Побудова графіків функцій y1,y2
```

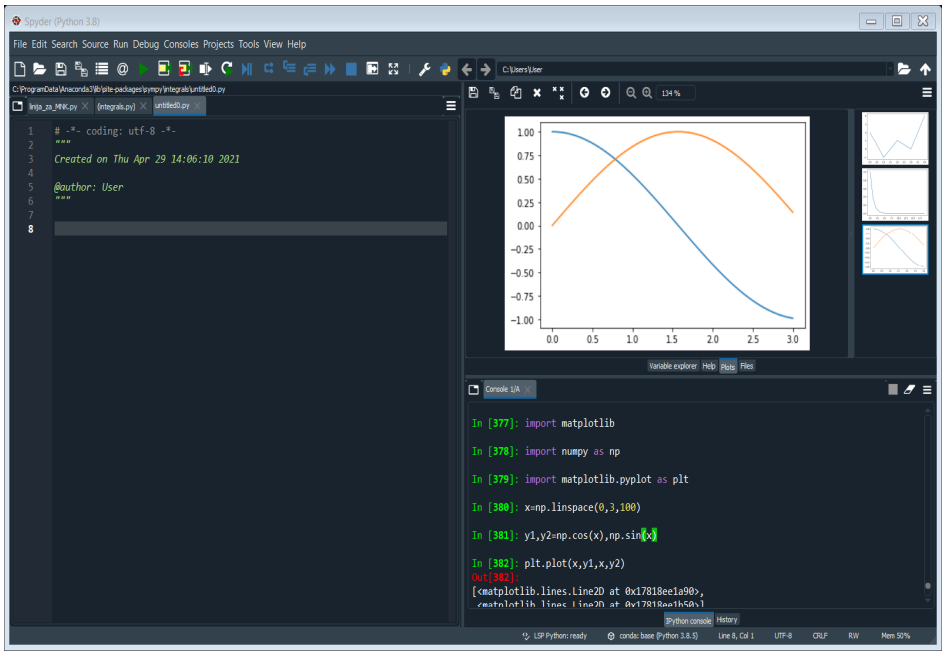


Рисунок 1.3 – Екран комп'ютера з фігурою 3, на якій суміщені графіки двох математичних функцій, заданих в аналітичній формі

В аргументних дужках програмної функції ***plot( )*** окрім ***символа аргументу*** та ***символа*** математичної ***функції*** можна задавати ***стиль і колір лінії*** графіка, а також ***форму маркерів*** в окремих точках на графіку, помістивши їх ***символи в одинарні лапки***.

Наприклад '***:rh***' свідчитиме, що ***лінія пунктирна*** бо вказано її символ у вигляді двокрапки ***«:»***, що ***вона червоного кольору***, бо вказано ***її символ*** у вигляді першої літери англійської назви червоного кольору ***«r»*** і що ***маркерні точки*** мають форму шестикутника, бо вказано ***їх символ*** у вигляді латинської літери ***«h».***

Інші кольори теж символізуються першими літерами їх англійських найменувань, але можуть використовуватись і повні англійські назви кольорів, якщо колір не об'єднується у спільну конструкцію зі стилем лінії та маркерами.

Що ж до ***стилю ліній***, то їх символами є: ***суцільної*** – «-», тобто, мінус; ***розривно***ї – ***«--»***, тобто, два мінуси; ***штрих-пунктирної*** – ***«-.»***, тобто, мінус-крапка.

Що ж до ***маркерів***, то вони символізуються так: ***кружок*** – ***«o»***, ***квадрат*** – ***«s»***, ***ромб*** – ***«d»***, ***хрест*** – ***«x»***, ***плюс*** – ***«+»***, ***п'ятикутник*** – ***«p»***, ***зірка*** – ***«*»***.

Крім того, в аргументних дужках програмної функції ***plot( )*** після символів аргументу і математичної функції та опцій стилю і кольору лінії графіка та форми маркерів ***потрібно задавати*** опціями ***linewidth*** та ***markersize*** товщину лінії та зовнішній розмір маркера у визначених стандартах, наприклад, linewidth=3, markersize=10.

Якщо ж ми не бажаємо, щоб точки, визначені маркерами, з'єднувались лініями, то в аргументні дужки вносимо також апострофну складову ***linestyle=' '***

Викладене у цьому абзаці ***проілюстровано в прикладах № 25 та № 26***.

Приклад № 25 (Python-програма побудови графіків зі стилем і кольором) (результат – на рис. 1.4)

```
In [1]: import matplotlib                    # Виклик ППП matplotlib
In [2]: import numpy as np                   # Виклик ППП numpy як np
In [3]: import matplotlib.pyplot as plt      # Виклик модуля matplotlib.pyplot як plt
In [4]: x=np.linspace(-2,2,100)              # Внесення масиву значень аргументу x
In [5]: y1=x                                 # Обчислення значень функції y1
In [6]: y2=-x**2                             # Обчислення значень функції y2
In [7]: y3=x+x**2                            # Обчислення значень функції y3
In [8]: plt.plot(x,y1,'-r',x,y2,'--b',\      # Побудова графіків функцій y1,y2,y3
x,y3,':c',linewidth=3)                       ….з заданими стилем і кольором
```

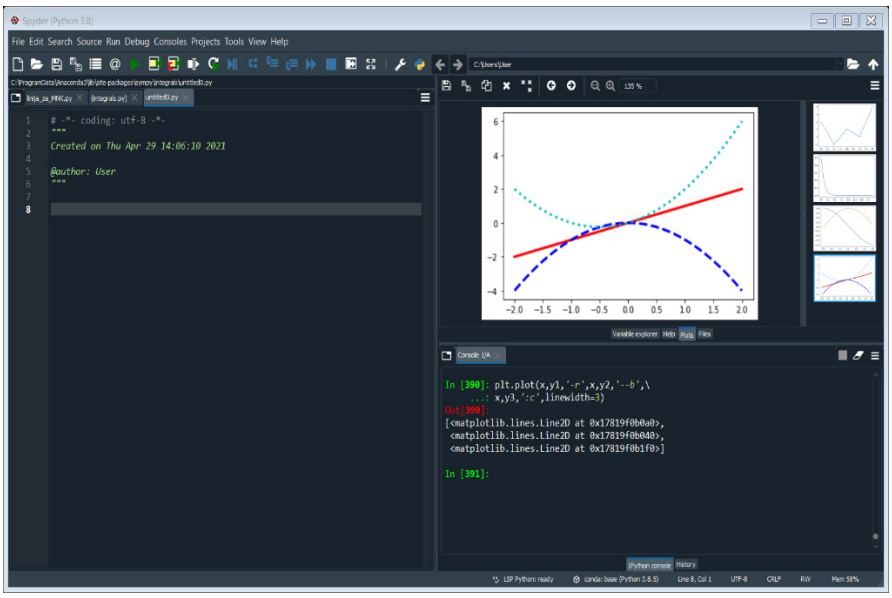


Рисунок 1.4 – Екран комп'ютера з фігурою 4, на якій лініями в різних кольорах і стилях зображено графіки трьох математичних функцій, заданих в аналітичній формі

Приклад № 26 (Python-програма побудови графіків в кольорах і з маркерами) (результат – на рис. 1.5)

```
In [1]: import matplotlib                          # Виклик ППП matplotlib
In [2]: import numpy as np                         # Виклик ППП numpy як np
In [3]: import matplotlib.pyplot as plt            # Виклик модуля matplotlib.pyplot як plt
In [4]: x=np.linspace(-2,2,30)                     # Внесення масиву значень аргументу x
In [5]: y1=x*np.exp(-x)                            # Обчислення значень функції y1
In [6]: y2=x**2*np.sin(x)                          # Обчислення значень функції y2
In [7]: plt.plot(x,y1,':cx',x,y2,'-ko',linewidth=3,\   # Побудова графіків функцій y1,y2
   markersize=8)                                   ...зі стилем, кольором та маркерами
```

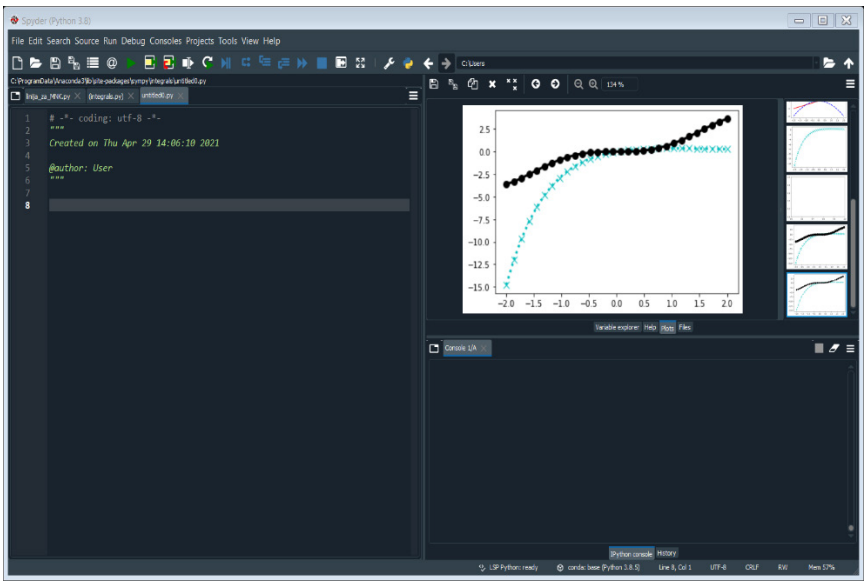


Рисунок 1.5 – Екран комп'ютера з фігурою 5, на якій лініями в різних кольорах і стилях та з нанесенням маркерів у формі кола та хреста зображено графіки двох математичних функцій, заданих в аналітичній формі

1.5.2 Побудова графіків на площині з нанесенням надписів у графічному середовищі matplotlib

У цьому пункті ми ***покажемо***, знову ж таки, використовуючи інформацію, викладену в роботі [7], ***як створювати надписи на графіках***.

В усіх наведених вище прикладах побудови графіків, ми не побачили ніяких надписів ні на графіках, ні на осях координат; це тому, що в програми побудови цих графіків ми не вносили функцій, які реалізують надписи, і частина яких має бути вписаною в програму до функції plt.plot( ), а частина – після неї.

Обов'язковою у цьому випадку ***стає*** команда ***plt.figure (facecolor ='white'),*** яка задає ту частину площини ***Axes*** стандартизованих розмірів з білим фоном, на якій буде розміщено ***рисунок***.

Надписи на осях Axis формуються командами ***plt.xlabel ( )*** та ***plt.ylabel ( )***.

Заголовок графіка формується командою ***plt.title ( ),*** в аргументних дужках якої окрім тексту заголовка у вигляді стрічки вписується ще й розмір літер.

Текстові пояснення на графіках формуються командою ***plt.legend ( ),*** в аргументних дужках якої вписується розмір літер та ***після якої*** в програмі розміщується команда ***plt.text ( )*** з самим текстом і координатами його прив'язки до поля Axes з використанням складової ***transform=ax.transAxes,*** яка переводить іменовані координати у безрозмірні відносні в межах одиниці.

Взяти в кольорову ***рамку текст*** можна аргументом ***bbox***.

Розмістити текст ***під кутом*** можна за допомогою аргументу ***rotation.***

У разі, якщо в тексті використовуються ***математичні формули***, їх потрібно з обох боків оточити ***знаками долара $.***

Для використання функцій ***latex(***) автоматичної кодової ***розмітки*** текстів ***TeX*** її потрібно викликати ***з ППП sympy***, який позначається в аргументних дужках функцією ***S( ).***

Оскільки ***TeX використовує у своїй структурі символ «\» (слеш***), то ***перед стрічкою*** аргументного виразу потрібно вписати ***літеру «r»,*** щоб програма не сприймала його як спецсимвол перенесення тексту на наступний рядок.

Окрім внесення в поле графіка тексту командою ***plt.text ( )*** в це поле можна вносити командою ***plt.annotate( )*** короткі текстові ***примітки зі стрілкою***, що починається коло тексту примітки і закінчується в точці поля, якої вона стосується. В координатних дужках цієї функції вказується текст примітки, координати точки, до якої направлена стрілка, місце розташування примітки та відстань від кінця стрілки до точки поля, куди вона направлена.

А оскільки ***ППП matplotlib*** не містить шрифтів кирилицею, то при використанні для набору тексту українською мовою у ***Windows***, який у варіантах шрифтів Arial, Times New Roman, Tahoma підтримує і кирилицю, потрібно ***перед набиранням тексту сформувати*** бібліотеку (***fantasy) з***

цих ***шрифтів*** командами: ***mpl.rsParams['font.family']='fantasy'; mpl.rsParams['font.fantasy']='Arial', 'Times New Roman', 'Tahoma'***, а перед стрічкою, в якій є і українські слова, поставити ***юнікодовий символ «и».***

Викладене вище у цьому абзаці проілюстровано у формі файлу ***в прикладі № 27***, запозиченому нами з роботи [7].

А те, як ***на одному об'єктному полі Axes*** розмістити ***два графічних об'єкти ax1, ax2***, що містять два графіки з окремими координатними осями, проілюстровано у формі файлу ***в прикладі № 28***, теж запозиченому нами з роботи [7].

Приклад № 27 (Python-програма побудови графіків на площині з нанесенням надписів) (результат – на рис. 1.6)

```
import numpy as np                          # Виклик ППП numpy як np
import matplotlib as mpl                    # Виклик ППП matplotlib як mpl
import matplotlib.pyplot as plt             # Виклик модуля matplotlib.pyplot як plt
mpl.rcParams['font.family']='fantasy'       # Створення бібліотеки шрифтів
mpl.rcParams['font.fantasy']='Arial','Times New\
      Roman','Tahoma'
str=r'$\frac{1}{\sigma\sqrt{2\pi}}\,e^{-\frac{(x-\  # Створення стрічки
      mu)^2}{2\sigma^2}}$'
x=np.linspace(-3,5,40)                      # Внесення масиву значень аргументу x
sigma=1                                     # Внесення значення параметра sigma
mu=1                                        # Внесення значення параметра ти
y=(1/(sigma*np.sqrt(2*np.pi)))*np.exp(-(x-  # Створення функції
      mu)**2/(2*sigma**2))
fig=plt.figure(facecolor='white')           # Створення поля (фігури) під рисунок
plt.plot(x,y,'-bo',linewidth=3,markersize=10, # Побудова графіка функції
      label=str)
plt.legend(fontsize=18,loc='upper left')    # Команда на створення написів
ax=fig.gca()                                # Прив'язка до поля ( фігури) рисунка
plt.title(r'$\mu=1,\sigma=1$')              # Заголовок (з використанням параметрів)
plt.text(0.58,.95,r'Графік функції $\varphi(x)$', # Створення тексту
     horizontalalignment='left',verticalalignment\
      ='center', transform=ax.transAxes,\
      fontsize=16)
ax.annotate('Максимум',xy=(1,0.4),xytext=\  # Створення примітки з прив'язкою
      (-2,0.25), arrowprops=dict(facecolor=\
      'green',shrink=0.05))
plt.text(-0.9,0.15,'Графік кривої',rotation=70, # Створення тексту під кутом 70°
     horizontalalignment=\
      'center',verticalalignment='center')
plt.text(2.5,0.3,str,fontsize=24,bbox=\
      dict(edgecolor='w',\                  # Взяття стрічки
       color='cyan'),color='black')           в кольорову рамку
```

```
plt.xlabel(u'X-вісь абсцис',{'fontname':'Times \
        New Roman'})                    # Надпис під віссю абсцис
plt.ylabel(r'$\varphi(x)$-ордината')    # Надпис вздовж ординати
```

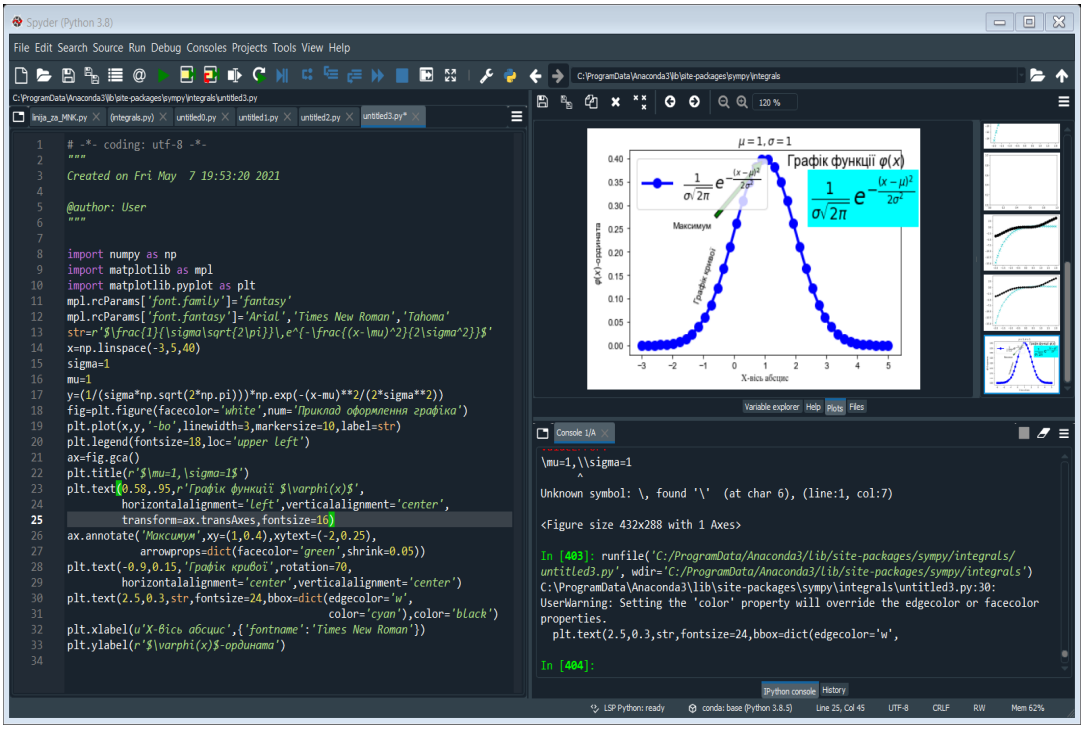


Рисунок 1.6 – Екран комп'ютера з файлом прикладу № 27 та фігурою 6, на якій зображено графік «дзвінницевої» функції, якою задається густина ймовірності нормального закону розподілу випадкової величини, ***з нанесенням надписів*** на графіку, визначених програмою, записаною у файлі

Приклад № 28 (Python-програма суміщення двох графіків на одному рисунку з нанесенням надписів) (результат – на рис. 1.7)

```
import matplotlib.pyplot as plt          # Виклик модуля matplotlib.pyplot як plt
import numpy as np                       # Виклик ППП numpy як np
x=np.linspace(0,3,16)                    # Внесення масиву значень аргументу x
y=x**3                                   # Обчислення функції y
fig=plt.figure(facecolor='white')        # Створення поля (фігури) під рисунок
ax1=fig.add_axes([0.0,0.0,1.0,1.0])      # Створення поля під графік y=y(x)
ax2=fig.add_axes([0.25,0.60,0.35,0.25])  # Створення поля під графік x=x(y)
ax1.plot(x,y,'c',linewidth=3)            # Побудова графіка функції y=y(x)
ax1.set_title('function')                # Заголовок над графіком функції y=y(x)
ax2.plot(y,x,'r',linewidth=2)            # Побудова графіка функції x=x(y)
ax2.set_title('back function')           # Заголовок над графіком функції x=x(y)
ax2.set_xlabel('y')                      # Заміна надпису на внутрішній осі з y на x
ax2.set_ylabel('x')                      # Заміна надпису на внутрішній осі з x на y
```

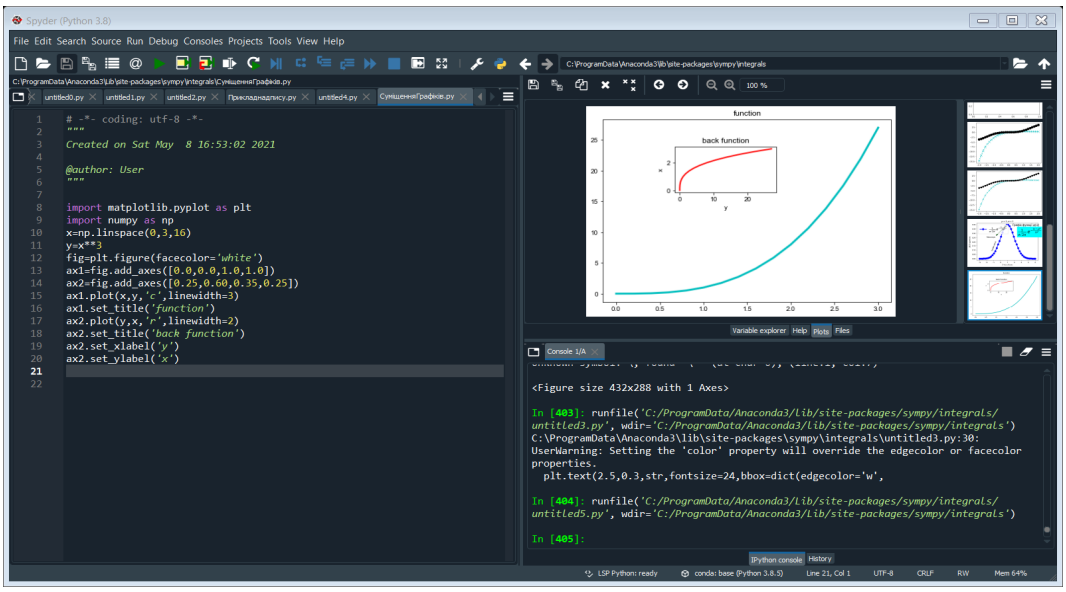


Рисунок 1.7 – Екран комп'ютера з файлом прикладу № 28 та фігурою 7, на якій зображено графіки функції та функції, до неї оберненої, з нанесенням надписів на графіках, визначених програмою, записаною у файлі, в якій ***використані методи fig.add_axes( ),*** якими рисунок виконується в прямокутних областях, визначених відносними координатами та відносними розмірами

1.5.3 Побудова стовпцевих та кругових діаграм на площині з використанням ППП matplotlib

Оформляючи звіти про результати наукових досліджень та ***створення текстів*** для публікацій в наукових журналах у вигляді статей ***для візуальної демонстрації*** співвідношень між елементами різних масивів, отриманих в процесі досліджень, часто ***використовують стовпцеві та кругові діаграми.***

Стовпцева діаграма являє собою сукупність прямокутників однакової, але меншої одиниці ширини, кожен із яких прив'язаний до послідовно пронумерованих одиничних відрізків осі абсцис (наприклад місяців року чи певної послідовності років) та кожен із яких має висоту, яка задається відповідним числовим значенням елемента масиву, стовпцева діаграма для якого створюється.

Стовпцева діаграма створюється функцією ***matplotlib.pyplot.bar (locs,vals,width,color***), в якій ***locs*** – абсциса лівого краю кожного стовпця, прив'язана до одиничного відрізка, на якому цей стовпець встановлюється, ***vals*** – сукупність числових значень масиву, стовпцева діаграма якого створюється, ***width*** – ширина кожного стовпця, ***color*** – колір, в який зафарбовуються усі стовпці.

Файл Python-програми, якою реалізується функція ***matplotlib.pyplot.bar (locs,vals,width,color)***, наведений ***у прикладі № 29***.

Приклад № 29 (Python-програма побудови стовпцевої діаграми для одного масиву даних) (результат – на рис. 1.8)

```
import matplotlib                          # Виклик ППП matplotlib
import matplotlib.pyplot as plt            # Виклик модуля matplotlib.pyplot як plt
import numpy as np                         # Виклик ППП numpy як np
locs=np.arange(7)                          # Прив'язка основ стовпців до відрізків осі
data=[6,1,5,9,4,8,3]                       # Масив для діаграми у вигляді списка
plt.bar(locs,data,width=0.75,              # Побудова стовпцевої діаграми
        color='green')
```

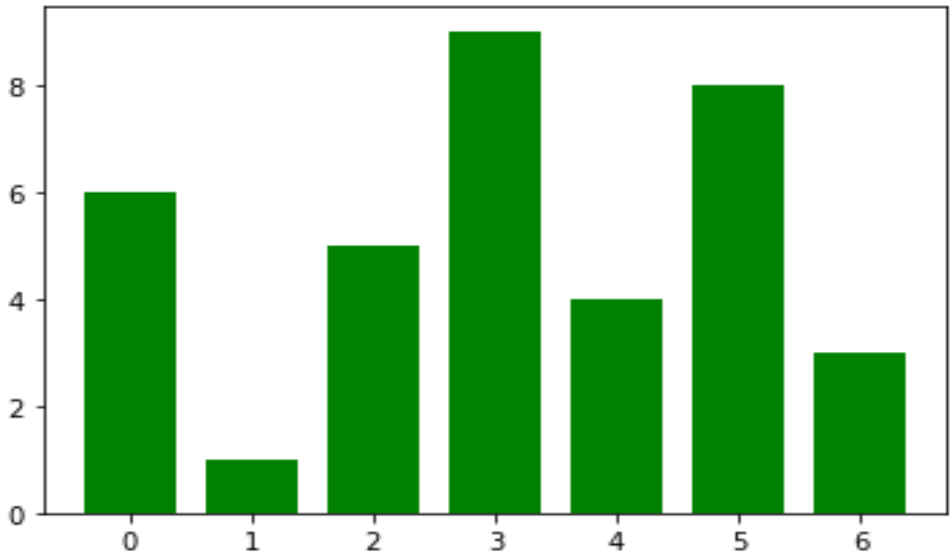


Рисунок 1.8 – Стовпцева діаграма для одного масиву до прикладу № 29

А на рисунку 1.9 зображена ***стовпцева діаграма***, на якій відображена картина порівняння між собою одразу трьох масивів з використанням тієї ж функції, використаної тричі в одному файлі, наведеному у ***прикладі № 30.***

Приклад № 30 (Python-програма побудови стовпцевої діаграми для трьох масивів даних) (результат – на рис. 1.9)

```
import matplotlib                          # Виклик ППП matplotlib
import matplotlib.pyplot as plt            # Виклик модуля matplotlib.pyplot як plt
import numpy as np                         # Виклик ППП numpy як np
fig=plt.figure(facecolor='white')          # Створення білого поля рисунка
n=7                                        # Визначення кількості стовпців
data1=[1,2,3,4,5,6,7]                      # Перший масив у вигляді списка
data2=[8,7,6,5,4,3,2]                      # Другий масив у вигляді списка
data3=[4,4,4,4,4,4,4]                      # Третій масив у вигляді списка
locs=np.arange(1,n+1)                      # Прив'язка основ стовпців до відрізків осі
wid=0.31                                   # Визначення ширини стовпців
```

```
plt.bar(locs,data1,width=wid,                  # Побудова стовпців першого масиву
        color='green')
plt.bar(locs+wid,data2,width=wid,              # Побудова стовпців другого масиву
        color='blue')
plt.bar(locs+2*wid,data3,width=wid,            # Побудова стовпців третього масиву
        color='red')
fig.gca( ).grid(True)                          # Об'єднання усіх стовпців на діаграмі
```

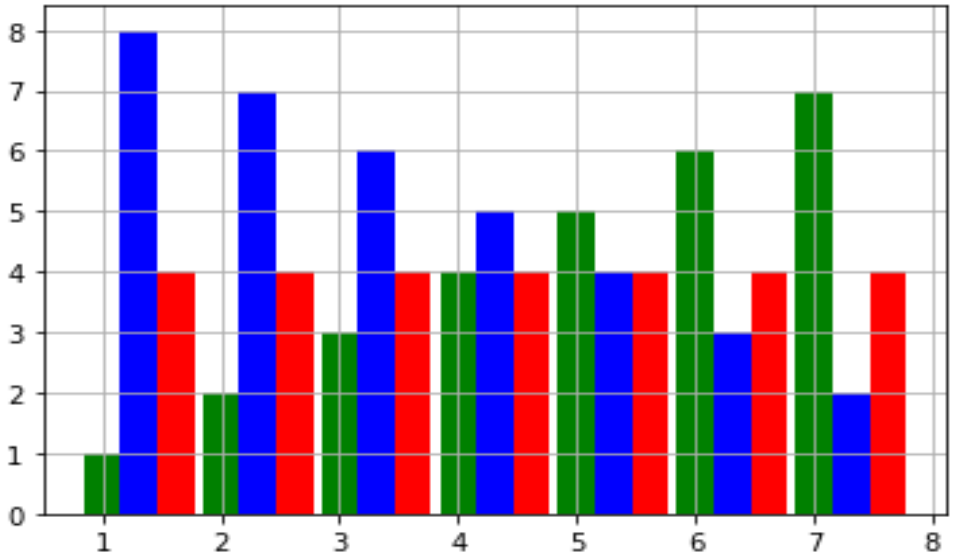


Рисунок 1.9 – Стовпцева діаграма для трьох масивів до прикладу № 30

У випадку, коли потрібно графічно відобразити співвідношення між різними елементами масиву, приведеними до одного відрізку часу (наприклад співвідношення між різними видами продукції, які виробляє протягом року одне і те ж підприємство) ***кругова діаграма*** створюється функцією ***matplotlib.pyplot.pie (vals, labels***), в якій ***vals*** – сукупність числових значень масиву, кругова діаграма якого створюється, а ***labels*** – сукупність кольорових міток, якими покриваються сектори, площа кожного з яких дорівнює числовому значенню елемента масиву, відображеному площею, якою оцінюється його вклад у загальну суму, відображену повною площею круга.

Файл Python-програми, якою реалізується функція ***matplotlib.pyplot.pie (vals, labels)***, наведений ***у прикладі № 31***.

Приклад № 31 (Python-програма побудови найпростішої кругової діаграми) (результат – на рис. 1.10)

```
import matplotlib                              # Виклик ППП matplotlib
import matplotlib.pyplot as plt                # Виклик модуля matplotlib.pyplot як plt
import numpy as np                             # Виклик ППП numpy як np
fig=plt.figure(facecolor='white')              # Створення білого поля рисунка
data=[7,3,5,9,1]                               # Масив для діаграми у вигляді списка
```

```
lbls=['blue','brown','green','red','purple']   # Кольорове покриття секторів діаграми
plt.pie(data, labels=lbls)                     # Побудова кругової діаграми
fig.gca().axis('image')                        # Прив'язка діаграми до поля рисунка
```

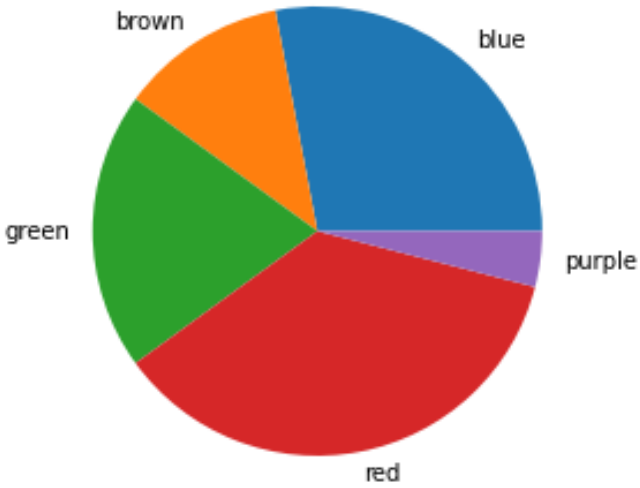


Рисунок 1.10 – Найпростіша кругова діаграма до прикладу № 31

А у випадку, коли потрібно не лише графічно відобразити співвідношення між різними елементами масиву, приведеними до одного відрізку часу, а й вказати ці співвідношення у відсотках та виокремити елемент, якому надається пріоритет, ***кругова діаграма*** створюється функцією ***matplotlib.pyplot.pie (vals, labels, explode, autopct, shadow***), в якій додаються: опція ***explode,*** якою визначаються радіальні зсуви кожного сектора, опція ***autopct,*** якою визначаються процентні частки кожного елемента масиву, та опція ***shadow,*** яка надає дозвіл на одночасне занесення на діаграму усього того, що поміщено в аргументних дужках.

Файл Python-програми, якою реалізується функція ***matplotlib.pyplot.pie (vals, labels, explode, autopct, shadow),*** наведений ***у прикладі № 32.***

Приклад № 32 (Python-програма побудови кругової діаграми з відсотковими визначеннями та домінантним сектором) (результат – на рис. 1.11)

```
import matplotlib                              # Виклик ППП matplotlib
import matplotlib.pyplot as plt                # Виклик модуля matplotlib.pyplot як plt
import numpy as np                             # Виклик ППП numpy як np
fig=plt.figure(facecolor='white')              # Створення білого поля рисунка
data=[7,3,5,9,1]                               # Масив для діаграми у вигляді списка
epd=[0,0,0.25,0,0]                             # Масив радіальних зсувів секторів діаграми
lbls=['blue','brown','green','red',            # Кольорове покриття секторів діаграми
      'purple']
```

```
plt.pie(data, labels=lbls, explode=epd,          # Побудова кругової діаграми з визначенням
      autopct='%2.2f%%',shadow=                    кольорів секторів, їх радіальних зсувів та
        True)                                      відсоткових співвідношень з 2-ма знаками
fig.gca().axis('image')                          # Прив'язка діаграми до поля рисунка
```

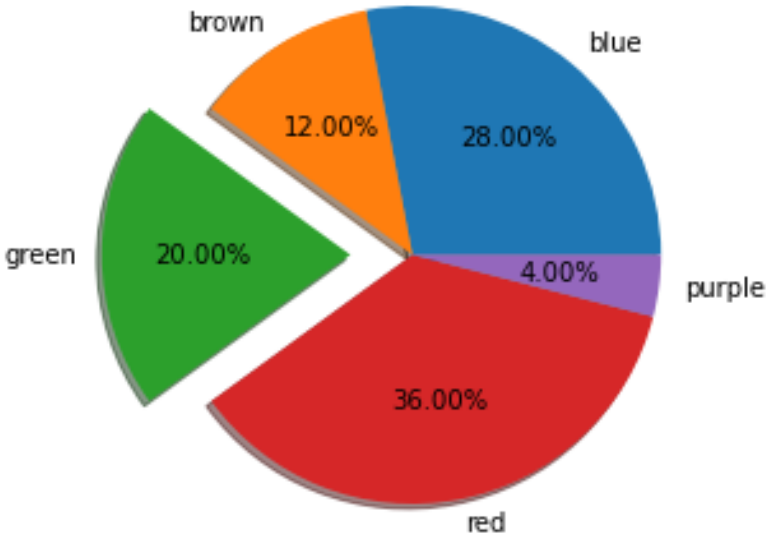


Рисунок 1.11 – Кругова діаграма з деталізацією параметрів до прикладу № 32

1.5.4 Побудова графіків ліній і поверхонь у тривимірному просторі з використанням ППП matplotlib

Бібліотека ***ППП matplotlib*** містить в собі велику кількість програм, що використовують функції побудови графіків у тривимірному просторі, значна кількість їх наведена і в роботі [7]. Ми ж у нашому навчальному посібнику ставимо за мету лише ознайомлення з кількома такими функціями, аби викладенням побудови графіків з їх використанням продемонструвати можливості ***ППП matplotlib*** та показати як створювати програми для побудови цих графіків і цим самим прокласти дорогу бажаючим до відомих бібліотек.

Отже, починати створювати власну програму побудови графіків у тривимірному просторі потрібно з виклику пакета ***matplotlib (***з присвоєнням йому короткого символа ***mpl),*** оскільки в ньому зосереджені усі модулі і функції побудови графіків.

Потім потрібно викликати модуль ***matplotlib.pyplot (***з присвоєнням йому короткого символа ***plt)***, оскільки з його використанням ми будемо створювати поле рисунка.

Потім із модуля ***mpl_toolkits.mplot3d*** потрібно викликати його внутрішній модуль ***Axes3D,*** в якому зосереджені функції реалізації тривимірної графіки.

А оскільки масиви значень вхідних даних і аргументів функцій, що використовуватимуться при побудові графіків, є атрибутами ***ППП numpy***, то обов'язково потрібно викликати і ***ППП numpy***, присвоївши йому короткий символ ***np.***

А далі уже потрібно викликати ту функцію з модуля ***Axes3D***, з використанням якої ми будемо будувати графік лінії чи поверхні в тривимірному просторі.

Варіанти програми побудови графіка лінії в тривимірному просторі наведено в ***прикладі № 33***, варіанти побудови графіка поверхні в тривимірному просторі наведено в ***прикладі № 34***.

Приклад № 33 (Python-програма побудови графіка лінії в тривимірному просторі – варіант 1) (результат – на рис. 1.12)

```
import matplotlib as mpl                    # Виклик ППП matplotlib як mpl
import matplotlib.pyplot as plt             # Виклик модуля matplotlib.pyplot як plt
from mpl_toolkits.mplot3d import Axes3D     # Виклик внутрішнього модуля Axes3D
import numpy as np                          # Виклик ППП numpy як np
fig=plt.figure()                            # Створення поля рисунка
ax=Axes3D(fig)                              # Прив'язка модуля Axes3D до поля
u=np.linspace(-6*np.pi,6*np.pi,200)         # Внесення масиву значень змінної u
r=u**3/100+2                                # Перший варіант
x=r*np.cos(u)                                 рівняння лінії
y=r*np.sin(u)                                 в параметричній
z=u/(1.5*np.pi)                               формі
ax.plot(x,y,z,'blue',linewidth=3)           # Побудова графіка синім кольором
```

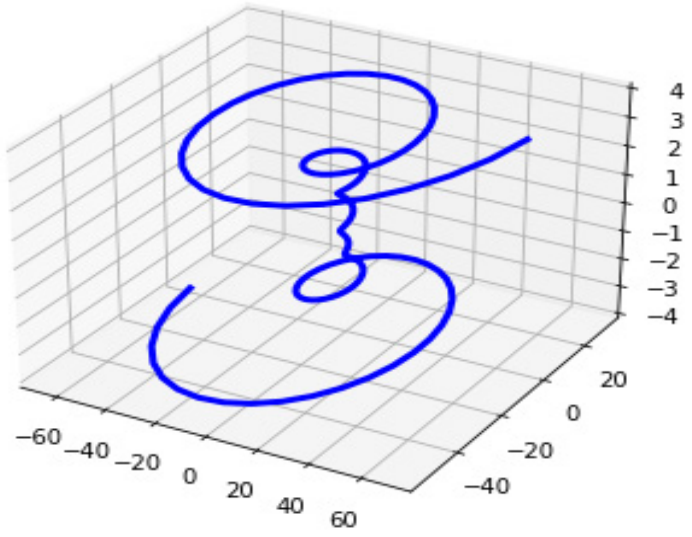


Рисунок 1.12 – Графік лінії в тривимірному просторі – варіант 1

Приклад № 33 (Python-програма побудови графіка лінії в тривимірному просторі – варіант 2) (результат – на рис. 1.13)

```
import matplotlib as mpl                          # Виклик ППП matplotlib як mpl
import matplotlib.pyplot as plt                   # Виклик модуля matplotlib.pyplot як plt
from mpl_toolkits.mplot3d import Axes3D           # Виклик внутрішнього модуля Axes3D
import numpy as np                                # Виклик ППП numpy як np
fig=plt.figure()                                  # Створення поля рисунка
ax=Axes3D(fig)                                    # Прив'язка модуля Axes3D до поля
u=np.linspace(-6*np.pi,6*np.pi,200)               # Внесення масиву значень змінної u
r=u**2/100                                        # Другий варіант
x=r*np.cos(u)                                       рівняння лінії
y=r*np.sin(u)                                       в параметричній
z=u/(1.5*np.pi)                                     формі
ax.plot(x,y,z,'green',linewidth=3)                # Побудова графіка зеленим кольором
```

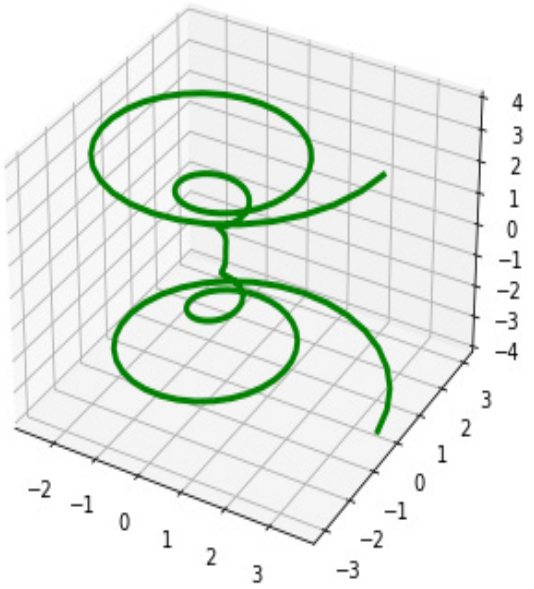


Рисунок 1.13 – Графік лінії в тривимірному просторі – варіант 2

Приклад № 33 (Python-програма побудови графіка лінії у тривимірному просторі – варіант 3) (результат – на рис. 1.14)

```
import matplotlib as mpl                          # Виклик ППП matplotlib як mpl
import matplotlib.pyplot as plt                   # Виклик модуля matplotlib.pyplot як plt
from mpl_toolkits.mplot3d import Axes3D           # Виклик внутрішнього модуля Axes3D
import numpy as np                                # Виклик ППП numpy як np
fig=plt.figure()                                  # Створення поля рисунка
ax=Axes3D(fig)                                    # Прив'язка модуля Axes3D до поля
u=np.linspace(-6*np.pi,6*np.pi,200)               # Внесення масиву значень змінної u
r=u**2/100+2                                      # Третій варіант
x=r*np.cos(u)                                       рівняння лінії
y=r*np.sin(u)                                       в параметричній
z=u/(1.5*np.pi)                                     формі
ax.plot(x,y,z,'red',linewidth=3)                  # Побудова графіка червоним кольором
```

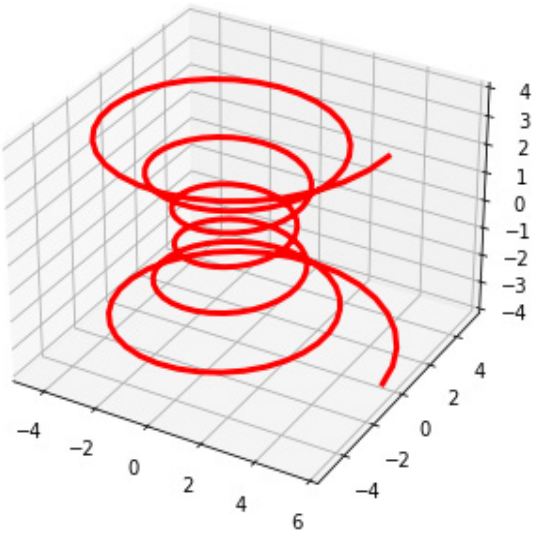


Рисунок 1.14 – Графік лінії в тривимірному просторі – варіант 3

Приклад № 34 (Python-програма побудови графіка поверхні у тривимірному просторі – варіант 1) (результат – на рис. 1.15)

```
import matplotlib as mpl                    # Виклик ППП matplotlib як mpl
import matplotlib.pyplot as plt             # Виклик модуля matplotlib.pyplot як plt
from mpl_toolkits.mplot3d import Axes3D     # Виклик внутрішнього модуля Axes3D
import numpy as np                          # Виклик ППП numpy як np
fig=plt.figure()                            # Створення поля рисунка
ax=Axes3D(fig)                              # Прив'язка модуля Axes3D до поля
u=np.linspace(-6*np.pi,6*np.pi,100)         # Внесення масиву значень змінної u
x,y=np.meshgrid(u,u)                        # Створення допоміжних векторів
r=np.sqrt(x**2+y**2)                        # Перший варіант рівнянь для
z=np.sin(r)/r**2                              опису поверхні
ax.plot_surface(x,y,z,rstride=1,cstride=1,  # Побудова поверхні за координатами
        linewidth=2)                          вузлів з одиничним кроком
```

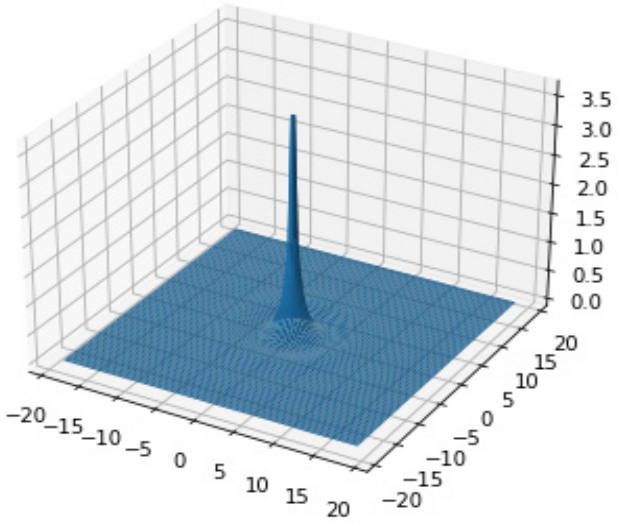


Рисунок 1.15 – Графік поверхні в тривимірному просторі – варіант 1

Приклад № 34 (Python-програма побудови графіка поверхні у тривимірному просторі – варіант 2) (результат – на рис. 1.16)

```
import matplotlib as mpl                    # Виклик ППП matplotlib як mpl
import matplotlib.pyplot as plt             # Виклик модуля matplotlib.pyplot як plt
from mpl_toolkits.mplot3d import Axes3D     # Виклик внутрішнього модуля Axes3D
import numpy as np                          # Виклик ППП numpy як np
fig=plt.figure()                            # Створення поля рисунка
ax=Axes3D(fig)                              # Прив'язка модуля Axes3D до поля
u=np.linspace(-6*np.pi,6*np.pi,100)         # Внесення масиву значень змінної u
x,y=np.meshgrid(u,u)                        # Створення допоміжних векторів
r=np.sqrt(x**2+y**2)                        # Другий варіант рівнянь для
z=np.sin(r)/r                                 опису поверхні
ax.plot_surface(x,y,z,rstride=1,cstride=1,  # Побудова поверхні за координатами
        linewidth=2)                          вузлів з одиничним кроком
```

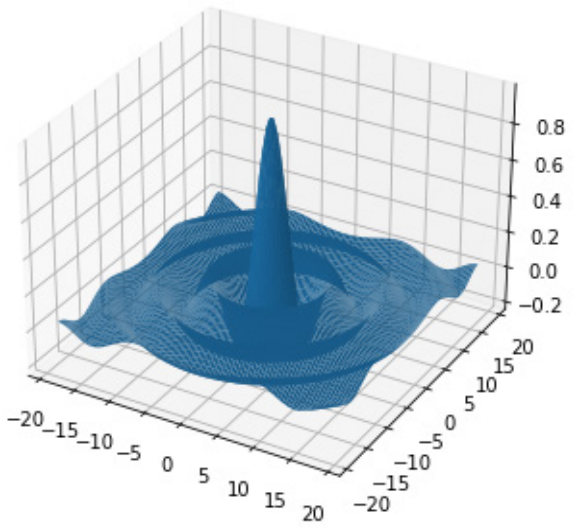


Рисунок 1.16 – Графік поверхні в тривимірному просторі – варіант 2

1.5.5 Побудова стовпцевих діаграм у тривимірному просторі з використанням ППП matplotlib

У поданні результатів досліджень, присвячених вивченню біологічного різноманіття на певних площах чи урожайності певних рослин із числа тих, що споживають люди чи худоба, корисними бувають графічні побудови у вигляді стовпцевих діаграм у тривимірному просторі.

Для їх побудови в ***ППП matplotlib*** використовується функція ***Axes3D.bar(xs, ys, zs, zdir=" ", color, alpha, width),*** в якій ***xs*** – масив абсцис лівих нижніх кутів стовпців діаграми, ***ys*** – масив, елементи якого визначають висоту стовпців, ***zs*** – масив аплікат лівих нижніх кутів стовпців діаграми, опція ***zdir=" "*** визначає до якої осі будуть ортогональними стовпці діаграми, опція ***color*** визначає в який колір будуть зафарбовані стовпці

діаграми, опція ***alpha*** визначає ступінь прозорості кольору, в який зафарбовані стовпці, опція ***width*** визначає ширину кожного стовпця.

Як будує стовпцеву діаграму в тривимірному просторі функція ***Axes3D.bar(xs, ys, zs, zdir=" ", color, alpha, width)*** продемонстровано в ***прикладах № 35*** та ***№ 36***.

Приклад № 35 (Python-програма побудови стовпцевої діаграми з двома рядами стовпців у тривимірному просторі – варіант 1) (результат – на рис. 1.17)

```
import matplotlib as mpl                                  # Виклик ППП matplotlib як mpl
import matplotlib.pyplot as plt                           # Виклик модуля matplotlib.pyplot як plt
from mpl_toolkits.mplot3d import Axes3D                   # Виклик внутрішнього модуля Axes3D
import numpy as np                                        # Виклик ППП numpy як np
fig=plt.figure(facecolor='white')                         # Створення поля рисунка (білого кольору)
ax=fig.add_subplot(111,projection='3d')                   # Прив'язка модуля Axes3D до поля
xs1=np.arange(10)                                         # Масив абсцис лівих нижніх кутів стовпців
ys1=np.random.rand(10)                                    # Масив висот стовпців
zs1=np.zeros(len(xs1))                                    # Масив аплікат лівих нижніх кутів стовпців
cs1=['c']*len(xs1)                                        # Визначення кольору стовпців
ax.bar(xs1,ys1,zs1,zdir='y',color=cs1,width=0.5)          # Побудова одного ряду стовпцевої діаграми
xs2=np.arange(10)                                         # Масив абсцис лівих нижніх кутів стовпців
ys2=np.random.rand(10)                                    # Масив висот стовпців
zs2=np.ones(len(xs2))                                     # Масив аплікат лівих нижніх кутів стовпців
cs2=['g']*len(xs2)                                        # Визначення зеленого кольору стовпців
ax.bar(xs2,ys2,zs2,zdir='y',color=cs2,alpha=0.6,          # Побудова стовпцевої діаграми з
       width=0.5)                                           двома рядами стовпців
```

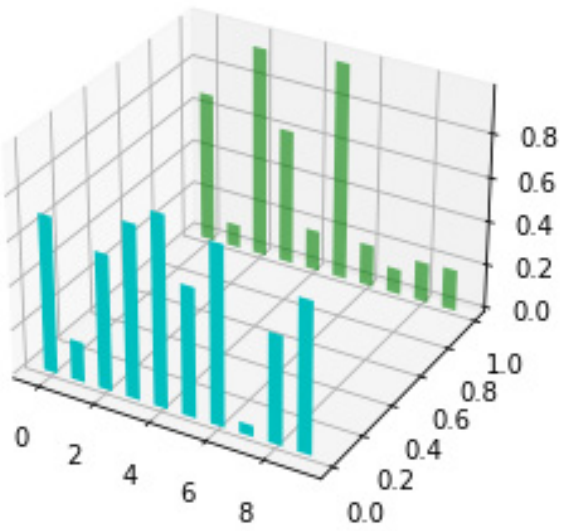


Рисунок 1.17 – Графік стовпцевої діаграми з двома рядами стовпців у тривимірному просторі – варіант 1

Приклад № 36 (Python-програма побудови діагональної стовпцевої діаграми у тривимірному просторі – варіант 2) (результат – на рис. 1.18)

```
import matplotlib as mpl                          # Виклик ППП matplotlib як mpl
import matplotlib.pyplot as plt                   # Виклик модуля matplotlib.pyplot як plt
from mpl_toolkits.mplot3d import Axes3D           # Виклик внутрішнього модуля Axes3D
import numpy as np                                # Виклик ППП numpy як np
fig=plt.figure(facecolor='white')                 # Створення поля рисунка (білого кольору)
ax=fig.add_subplot(111,projection='3d')           # Прив'язка модуля Axes3D до поля
xs1=np.arange(15)                                 # Масив абсцис лівих нижніх кутів стовпців
ys1=np.random.rand(15)                            # Масив висот стовпців
zs1=np.linspace(0.0,1.5,15)                       # Масив аплікат лівих нижніх кутів стовпців
cs1=['r']*len(xs1)                                # Визначення червоного кольору стовпців
ax.bar(xs1,ys1,zs1,zdir='y',color=cs1,alpha=0.6,  # Побудова стовпцевої діаграми з
       width=0.5)                                 #  діагональним розміщенням стовпців
```

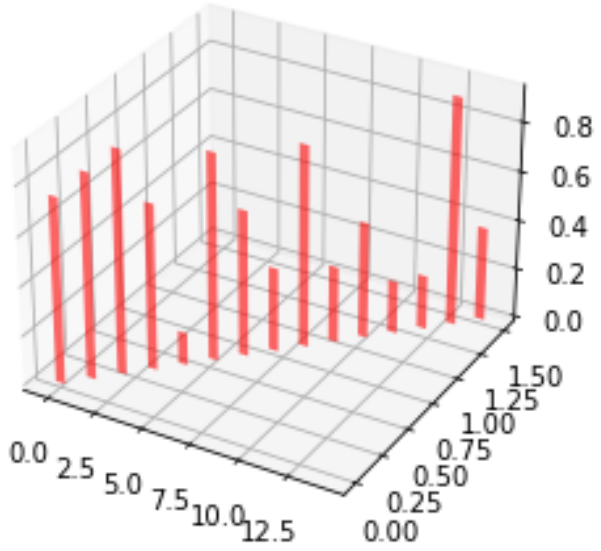


Рисунок 1.18 – Графік діагональної стовпцевої діаграми у тривимірному просторі – варіант 2

Методичні рекомендації до підрозділу 1.5. Завершуючи викладення матеріалу цього підрозділу викладачеві МЗКО обов'язково потрібно ще раз звернути увагу студентів на те, що:

1) в разі, якщо на графіку потрібно нанести кілька надписів, програму реалізації цього графіка потрібно здійснювати у вигляді файлу, бо в разі створення програми з використанням командних рядків, кожен із наступних командних рядків стиратиме з екрана результат, отриманий виконанням попередньої команди;

2) ***ППП matplotlib*** може реалізовувати побудову графіків лише у сукупності з ***ППП numpy***;

3) в разі, якщо на круговій діаграмі потрібно вказати відсоткові співвідношення секторів, то потрібно бути дуже уважним при створенні опції ***autopct='%2.2f%%'*** в аргументних дужках функції ***matplotlib.pyplot.pie( )***;

4) графіки ліній на площині та поверхонь в просторі можна будувати, використовуючи не лише ***ППП matplotlib*** та модуль ***sympy.plotting*** із ***ППП sympy***, про який мова піде в підрозділі, присвяченому використанню ***ППП sympy***, але і за допомогою модуля ***mpmath*** з ***ППП math,*** сумісного з ***Python-програмами,*** вивчення якого ми залишаємо читачам цього навчального посібника для самостійного освоєння

Розділ 2 МЕТОДИ ОБЧИСЛЕНЬ

2.1 Інтегрування функцій

Із усіх інтегралів, що використовуються у вищій математиці, ми **зупинимось** лише на формуванні та обчисленні **інтеграла Рімана** як такого, що використовується найчастіше в прикладних задачах, пов'язаних з інтегруванням функцій. Його формування викладемо в нижченаведеній послідовності.

Нехай на числовій осі на відрізку **[*a*,*b*]** значень аргументу $\boldsymbol{x}$ задана неперервна і обмежена функція ***f*(*x*)**. Розіб'ємо цей відрізок точками $\boldsymbol{x_0, x_1, x_2, \dots, x_n}$ на $\boldsymbol{n}$ сегментів

$$\Delta_i x = x_{i+1} - x_i,\ i = 0,1,\dots,n-1 \tag{2.1}$$

так, щоб $x_0 = a,\ x_n = b$.

Нехай $\boldsymbol{\xi_i}$ є внутрішньою точкою сегмента $\boldsymbol{\Delta_i x}$, тобто, $\boldsymbol{\xi_i \in \Delta_i x}$. Оскільки функція ***f*(*x*)** є неперервною і обмеженою, то на кожному сегменті $\boldsymbol{\Delta_i x}$ у якихось його точках вона набуватиме максимального $\boldsymbol{M_i}$ та мінімального $\boldsymbol{m_i}$ числового значення, тобто справедливим буде вираз

$$m_i \le f(\xi_i) \le M_i. \tag{2.2}$$

Домножимо члени нерівності (2.2) на $\boldsymbol{\Delta_i x}$ і підсумуємо ці добутки, результатом буде вираз

$$\sum_{i=0}^{n-1} m_i \Delta_i x \le \sum_{i=0}^{n-1} f(\xi_i)\Delta_i x \ \le \ \sum_{i=0}^{n-1} M_i \Delta_i x, \tag{2.3}$$

в якому праву і ліву частини нерівності називають, відповідно, верхньою та нижньою сумами Дарбу. Очевидно, що при збільшенні кількості сегментів $\boldsymbol{\Delta_i x}$ в сумах Дарбу кожен з них наближатиметься до нуля, верхній індекс *n* наближатиметься до нескінченності, а самі ці суми наближатимуться зверху і знизу до спільної межі **у вигляді числа**

$$J = \lim_{\Delta_i x \to 0} \sum_{i=0}^{n-1} f(\xi_i)\Delta_i x, \tag{2.4}$$

яке називають інтегралом Рімана і позначають

$$J = \int_a^b f(x)dx. \tag{2.5}$$

Виходячи з того, що кожна складова в сумах Дарбу задає площу прямокутника з основою $\boldsymbol{\Delta_i x}$ та висотою, відповідно, $\boldsymbol{M_i}$ та $\boldsymbol{m_i}$, інтеграл Рімана є виразом, за допомогою якого визначається площа плоскої фігури, обмеженої знизу відрізком **[*a*,*b*]** числової осі, а зверху – графіком функції ***f*(*x*)** в межах від ***f*(*a*)** до ***f*(*b*).**

Ця геометрична інтерпретація дозволяє вираз (2.5) називати визначеним інтегралом.

Узагальнюючи вираз (2.5) на функцію $\boldsymbol{f(x,y)}$ двох незалежних змінних $\boldsymbol{x,y}$, заданих на відрізках ***[a,b], [c,d]***, отримаємо вираз для двократного інтеграла

$$J_1 = \int_a^b \int_c^d f(x,y)dydx, \tag{2.6}$$

яким визначається об'єм фігури, обмеженої поверхнею $\boldsymbol{f(x,y)}$ над прямокутником зі сторонами ***[a,b], [c,d]*** на координатній площині x, y.

Якщо ж у виразі (2.5) ми не будемо вказувати межі, то після інтегрування отримаємо не число ***J*****,** а функцію ***F(x)*****,** до якої потрібно додати сталу інтегрування ***C***, тобто у цьому випадку замість виразу (2.5) матимемо вираз

$$\int f(x)dx = F(x) + C, \tag{2.7}$$

який називають невизначеним інтегралом від функції ***f(x).***

Зв'язок між виразами (2.5) та (2.7) визначається формулою Ньютона-Лейбніца, згідно з якою

$$\int_a^b f(x)dx = F(b)-F(a). \tag{2.8}$$

А в будь-якому підручнику з математичного аналізу можна ознайомитись з процесом отримання невизначених інтегралів ***F(x)*** для найбільш вживаних неперервних функцій *f(x).* Наприклад, для степеневої функції вираз (2.7) набуває вигляду

$$\int x^n dx = \frac{1}{n+1}x^{n+1} + C. \tag{2.9}$$

Методичні рекомендації до підрозділу 2.1. Завершуючи викладення матеріалу цього підрозділу викладачеві МЗКО обов'язково потрібно звернути увагу студентів на те, що:

1) в будь-якому з довідників з вищої математики вони можуть знайти таблиці невизначених інтегралів до найбільш вживаних неперервних функцій, використовуючи які та формулу Ньютона-Лейбніца, легко обчислити визначені інтеграли від цих функцій в заданих межах;

2) є автори наукових статей і навіть підручників з нематематичних дисциплін, які, записуючи формули для невизначених інтегралів, опускають сталу інтегрування *C*, але, використовуючи ці формули, потрібно пам'ятати, що сталі інтегрування в них є, і лише у формулі Ньютона-Лейбніца після підстановки меж інтегрування ці сталі відсутні, бо взаємно знищуються знаком мінус між ними.

2.2 Диференціювання функцій

Почнемо з фізики. Якщо автомобіль за відрізок часу $\boldsymbol{\Delta t = t_2 - t_1}$ подолає шлях $\boldsymbol{\Delta y = y(t_2) - y(t_1)}$, то, посилаючись ще на шкільний курс фізики, можна стверджувати, що цей шлях автомобіль долає з середньою швидкістю

$$v_s = \frac{\Delta y}{\Delta t}. \tag{2.10}$$

Якщо ж зменшувати відрізок часу до нуля, то отримаємо **вираз**

$$v(t) = \lim_{\Delta t \to 0} \frac{\Delta y}{\Delta t} = y'(t), \tag{2.11}$$

яким визначається швидкість автомобіля в даний момент часу та **який визначає похідну $\boldsymbol{y'(t)}$ функції $\boldsymbol{y(t)}$**.

Із виразу (2.11) витікає, що для будь-якої неперервної функції $y(x)$, заданої на відрізку [a,b] значень її аргументу *x,* її похідна визначатиметься виразом

$$y'(x) = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x} \tag{2.12}$$

і характеризуватиме швидкість зміни цієї функції у кожній точці на цьому відрізку.

В будь-якому підручнику з математичного аналізу можна ознайомитись з процесом отримання похідних $\boldsymbol{f'(x)}$ для найбільш вживаних неперервних функцій ***f(x).*** Наприклад, для степеневої функції

$$f(x) = x^n, \tag{2.13}$$

виконуючи граничний перехід (2.12), отримаємо

$$f'(x) = nx^{n-1}. \tag{2.14}$$

Цілком очевидно, що для константи $\boldsymbol{C}$, заданої на відрізку ***[a, b],*** приріст ΔC дорівнюватиме нулю, а тому, згідно з виразом (2.12), справедливим є вираз

$$C' = 0. \tag{2.15}$$

Знову ж таки в будь-якому підручнику з математичного аналізу доводиться, що похідні функцій

$$f\big(y(x)\big), \quad f_1(x) = g_1(x) \cdot h_1(x), \quad f_2(x) = \frac{g_2(x)}{h_2(x)} \tag{2.16}$$

матимуть відповідно вигляд:

$$f'\big(y(x)\big) = f'(y) \cdot y'(x), \tag{2.17}$$

$$f_1'(x) = g_1'(x) \cdot h_1(x) + g_1(x) \cdot h_1'(x), \tag{2.18}$$

$$f_2'(x) = \frac{g_2'(x) \cdot h_2(x) - g_2(x) \cdot h_2'(x)}{\left(h_2(x)\right)^2} \tag{2.19}$$

Оскільки похідна $\boldsymbol{y'(x)}$ неперервної функції $\boldsymbol{y(x)}$, заданої на відрізку ***[a, b]*** значень її аргументу $\boldsymbol{x}$, залишається функцією аргументу $\boldsymbol{x}$, то до неї теж можна застосувати граничний перехід (2.12), в результаті чого ми отримаємо другу похідну $\boldsymbol{y''(x)}$ неперервної функції $\boldsymbol{y(x)}$, заданої на відрізку ***[a, b]*** значень її аргументу $\boldsymbol{x}$, тобто, справедливим буде вираз

$$y''(x) = \left(y'(x)\right)' = \lim_{\Delta x \to 0} \frac{\Delta y'(x)}{\Delta x}. \tag{2.20}$$

Узагальнюючи цю процедуру, для визначення $\boldsymbol{n}$-ої похідної $\boldsymbol{y^{(n)}(x)}$ матимемо вираз

$$y^{(n)}(x) = \left(y^{(n-1)}(x)\right)' = \lim_{\Delta x \to 0} \frac{\Delta y^{(n-1)}(x)}{\Delta x}. \tag{2.21}$$

А частинні похідні функції двох змінних $\boldsymbol{f(x, y)}$ матимуть вигляд:

$$f_x'(x, y) = \frac{\partial f}{\partial x}, \qquad f_y'(x, y) = \frac{\partial f}{\partial y}. \tag{2.22}$$

Методичні рекомендації до підрозділу 2.2. Завершуючи викладення матеріалу цього підрозділу викладачеві МЗКО обов'язково потрібно звернути увагу студентів на те, що:

1) для визначення похідної $f'(x)$ довільної функції $f(x)$ однієї змінної не потрібно кожен раз здійснювати граничний перехід за виразом (2.11), а достатньо скористатись довідником з вищої математики, в якому уже визначені і наведені похідні найбільш вживаних в математичних перетвореннях функцій, або звернутись до чат-бота, наприклад Chat-GPT, з проханням надати вам вираз для похідної функції, яка вам потрібна;

2) у разі визначення частинної похідної функції від двох змінних за однією з них друга змінна при диференціюванні цієї функції розглядається як константа;

3) у разі визначення частинних похідних функції від двох змінних за однією змінною частинна похідна може мати один порядок, а за іншою змінною – інший;

4) оскільки диференціювання функції є операцією, оберненою до інтегрування, то похідна невизначеного інтеграла дорівнює підінтегральній функції;

5) похідна визначеного інтеграла зі змінною верхньою межею, який фактично є функцією від підінтегральної функції, теж дорівнює підінтегральній функції.

2.3 Апроксимація функцій рядами

Отримавши з експериментального дослідження якогось об'єкта кількісні дані, ми, як правило, намагаємось створити функцію, якою можна було б відтворити не лише ці експериментальні дані, а й отримати їх наступні значення і без продовження експерименту.

Якщо ми від цієї функції вимагаємо, щоб її **графік** проходив через **кожну** експериментально отриману **точку**, **то** має місце **інтерполяція**.

Якщо ж ми від цієї функції вимагаємо, щоб її **графік** проходив так, щоб **найкоротша відстань** від нього **до** кожної з експериментальних **точок** була **мінімальною** за якимось критерієм, **то** має місце **апроксимація.**

Оскільки при вимірюваннях експериментально отриманих даних завжди мають місце похибки, викликані як похибками вимірювальних засобів, так і похибками від завад, то немає сенсу пов'язувати ці експериментальні дані між собою інтерполяційними співвідношеннями, а є сенс конструювати для їх відтворення функції, застосовуючи методологію апроксимації.

І як функціональні конструкції при апроксимації експериментальних даних широко використовуються поліноміальні відрізки степеневих рядів та відрізки тригонометричних рядів Фур'є.

2.3.1 Апроксимація функцій степеневими рядами

Нагадаємо, що, як відомо з курсу математичного аналізу, будь-яку неперервну функцію $\boldsymbol{f(x)}$ на заданому відрізку значень $[\boldsymbol{a, b}]$ її аргументу $\boldsymbol{x}$ можна відтворити степеневим рядом Тейлора в околі точки $\boldsymbol{x = x_0}$ у вигляді

$$f(x) = f(x_0) + \frac{1}{1!}f'(x_0)(x - x_0) + \frac{1}{2!}f''(x_0)(x - x_0)^2 + \frac{1}{3!}f'''(x_0)(x - x_0)^3 + \ldots, \quad (2.23)$$

або степеневим рядом Маклорена в околі точки $\boldsymbol{x = 0}$ у вигляді

$$f(x) = f(0) + \frac{1}{1!}f'(0)x + \frac{1}{2!}f''(0)x^2 + \frac{1}{3!}f'''(0)x^3 + \ldots, \quad (2.24)$$

використовуючи з яких лише перших два члени, отримаємо рівняння прямої у вигляді полінома першого степеня

$$f(x) = c_0 + c_1 x, \quad (2.25)$$

а використовуючи з яких лише перших три члени, отримаємо рівняння параболи у вигляді полінома другого степеня

$$f(x) = c_0 + c_1 x + c_2 x^2. \quad (2.26)$$

Очевидно, що рівнянням (2.25) можна апроксимувати множину експериментальних даних, які на координатній площині $\boldsymbol{xy}$ в межах

відрізка $\boldsymbol{x \in [a, b]}$ розміщуються у вузькому еліпсі, повздовжня вісь якого чисельно набагато перевищує поперечну, а рівнянням (2.26) можна апроксимувати множину експериментальних даних, які на координатній площині $\boldsymbol{xy}$ в межах відрізка $\boldsymbol{x \in [a, b]}$ розміщуються в арковій конструкції з прямуванням до мінімуму чи до максимуму та з недалеко віддаленими одна від одної нижньою та верхньою дугами.

Цілком очевидно, що в разі, коли отримані нами експериментальні дані на координатній площині $\boldsymbol{xy}$ в межах відрізка $\boldsymbol{x \in [a, b]}$ відображають не один екстремум, а два, то для їх апроксимації потрібно використати поліном четвертого порядку, який матимемо, якщо зі степеневих рядів (2.23), (2.24) використаємо перших п'ять членів, в результаті чого цей поліном матиме вигляд

$$f(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + c_4 x^4. \tag{2.27}$$

Тож, як бачимо, збільшення кількості екстремумів в конструкціях на координатній площині $\boldsymbol{xy}$, що заповнюються експериментальними даними, вимагатиме від нас використання для їх апроксимації поліномів з порядком, вдвічі більшим від кількості екстремумів.

Зауважимо, що в разі, якщо апроксимація експериментальних даних здійснюється залежно від моментів часу $\boldsymbol{t_i,\ i = 0, 1, 2, \dots, m}$, в які здійснені вимірювання, то у виразах (2.23)–(2.27) потрібно замість незалежної змінної $\boldsymbol{x}$ підставити незалежну змінну ***t.***

А якщо у виразах (2.25), (2.26), (2.27) при апроксимації ними експериментальних даних $\boldsymbol{y_i = f(x_i),\ i = 0, 1, 2, \dots n - 1}$ ми бажаємо отримати оптимальні числові значення параметрів $\boldsymbol{c_k,\ k = 0, 1, 2, 3, 4,}$ то потрібно для їх визначення використати метод найменших квадратів (**МНК**) з критерієм оптимальності у вигляді

$$\Sigma = \sum_{i=0}^{n-1} \left(y_i - f(x_i)\right)^2, \tag{2.28}$$

який, наприклад, апроксимуючи експериментальні дані функцією (2.26), матиме вигляд

$$\Sigma = \sum_{i=0}^{n-1} \left(y_i - c_0 - c_1 x_i - c_2 x_i^2\right)^2. \tag{2.29}$$

Мінімізація цього критерію у вигляді (2.29) зводиться до сумісного розв'язання системи трьох рівнянь

$$\begin{cases} \dfrac{\partial \Sigma}{\partial c_0} = 0, \\ \dfrac{\partial \Sigma}{\partial c_1} = 0, \\ \dfrac{\partial \Sigma}{\partial c_2} = 0, \end{cases} \tag{2.30}$$

які диференціюванням виразу (2.29) та зосередженням членів з параметрами $\boldsymbol{c_0, c_1, c_2}$ у лівій частині легко трансформуються до вигляду

$$\begin{cases} c_0 + c_1\sum_{i=0}^{n-1} x_i + c_2\sum_{i=0}^{n-1} x_i^2 = \sum_{i=0}^{n-1} y_i, \\ c_0\sum_{i=0}^{n-1} x_i + c_1\sum_{i=0}^{n-1} x_i^2 + c_2\sum_{i=0}^{n-1} x_i^3 = \sum_{i=0}^{n-1} x_i y_i, \\ c_0\sum_{i=0}^{n-1} x_i^2 + c_1\sum_{i=0}^{n-1} x_i^3 + c_2\sum_{i=0}^{n-1} x_i^4 = \sum_{i=0}^{n-1} x_i^2 y_i. \end{cases} \quad (2.31)$$

Розв'язуючи систему рівнянь (2.31) отримаємо числові значення параметрів $\boldsymbol{c_0, c_1, c_2}$ полінома (2.26), який буде оптимально відображати ту множину експериментальних даних, з використанням яких ми побудували систему рівнянь (2.31).

2.3.2 Апроксимація функцій рядами Фур'є

Якщо ж у множині експериментальних даних, нанесених на координатну площину ***yt***, проглядається періодична складова з періодом $\boldsymbol{T}$, то для апроксимації цієї множини функцією $\boldsymbol{y = f(t)}$ доцільно використовувати відрізок ряду Фур'є у добре відомому з будь-якого підручника з математичного аналізу вигляді

$$f(t) = \frac{p_0}{2} + \sum_{i=1}^{n} p_i \cos(i\omega_1 t) + \sum_{i=1}^{n} q_i \sin(i\omega_1 t), \quad (2.32)$$

в якому $\boldsymbol{\omega_1 = \frac{2\pi}{T}}$ – частота першої гармоніки, а коефіцієнти $\boldsymbol{p_i,\ i = 0, 1, 2, \dots, n}$ та $\boldsymbol{q_i,\ i = 1, 2, \dots, n}$ обчислюються за виразами:

$$p_0 = \frac{1}{T}\int_0^T f(t)dt, \quad (2.33)$$

$$p_i = \frac{2}{T}\int_0^T f(t)\cos(i\omega_1 t)\,dt, \qquad q_i = \frac{2}{T}\int_0^T f(t)\sin(i\omega_1 t)\,dt. \quad (2.34)$$

Завершимо ми цей матеріал, присвячений апроксимації експериментальних даних відрізками ряду Фур'є, двома зауваженнями:

1) оскільки в результаті експерименту ми маємо значення функції $\boldsymbol{y = f(t)}$ лише в окремих точках $\boldsymbol{y_i = f(t_i),\ i = 0, 1, 2, \dots, n,}$ то інтеграли у виразах (2.33), (2.34) потрібно обчислювати одним із методів наближень;

2) оскільки при наближеному обчисленні інтегралів (2.33), (2.34) потрібно буде замість $\boldsymbol{dt}$ використовувати $\boldsymbol{\Delta_i t = t_{i+1} - t_i}$, $\boldsymbol{i = 0, 1, 2, \dots, n-1}$, то вимірювання експериментальних даних потрібно здійснювати через рівні проміжки часу, щоб мати $\boldsymbol{\Delta_i = Const}$;

3) якщо статистична сукупність експериментальних даних має аркоподібну конструкцію, що починається в околі початку координат, у першому квадранті і має симетричну, стосовно початку координат, аркоподібну конструкцію в третьому квадранті, то відрізок ряду Фур'є (2.32) для її апроксимації стає коротшим і в нього ввійдуть лише синусоїдальні гармоніки

$$f(t) = \sum_{i=1}^{n} q_i \sin(i\omega_1 t). \quad (2.35)$$

А якщо статистична сукупність експериментальних даних має аркоподібну конструкцію, яка є симетричною стосовно функціональної осі з максимумом на цій осі, то відрізок ряду Фур'є (2.32) для її апроксимації теж стає коротшим і в нього ввійдуть лише косинусоїдальні гармоніки та складова, що «піднімає» цю сукупність над віссю абсцис

$$f(t) = \frac{p_0}{2} + \sum_{i=1}^{n} p_i \cos(i\omega_1 t). \quad (2.36)$$

Методичні рекомендації до підрозділу 2.3. Завершуючи викладення матеріалу цього підрозділу викладачеві МЗКО обов'язково потрібно звернути увагу студентів на те, що:

1) згідно з теоремою Вейєрштрасса тренд будь-якої сукупності експериментальних точок можна апроксимувати степеневим поліномом з достатньою для розв'язання практичних задач точністю, компенсуючи наростання складності поля точок підвищенням степеня апроксимувального полінома;

2) в разі, якщо зі збільшенням абсцис $\boldsymbol{x}$ ординати $\boldsymbol{y}$ точок поля експериментальних даних статистично зменшуються, прямуючи асимптотично до осі $\boldsymbol{x}$, то для апроксимації тренду цього поля точок потрібно замість полінома, заданого, наприклад виразом (2.26), використовувати трансцендентний вираз

$$f_1(x) = (c_0 + c_1 x + c_2 x^2) e^{-\alpha x}, \quad (2.38)$$

замість критерію оптимальності (2.29) використовувати критерій

$$\Sigma_1 = \sum_{i=0}^{n-1} \left(y_i - (c_0 + c_1 x_i + c_2 x_i^2) e^{-\alpha x_i}\right)^2, \quad (2.39)$$

а замість системи трьох рівнянь (2.30) для визначення оптимальних значень параметрів $\boldsymbol{c_0, c_1, c_2}$ конструювати систему чотирьох рівнянь

$$\begin{cases} \dfrac{\partial \sum_1}{\partial c_0} = 0, \\ \dfrac{\partial \sum_1}{\partial c_1} = 0, \\ \dfrac{\partial \sum_1}{\partial c_2} = 0, \\ \dfrac{\partial \sum_1}{\partial \alpha} = 0 \end{cases} \quad (2.40)$$

для визначення оптимальних значень параметрів $\boldsymbol{c_0, c_1, c_2, \alpha}$;

3) в разі, якщо зі статистичним збільшенням абсцис $\boldsymbol{x}$ ординати $\boldsymbol{y}$ точок поля експериментальних даних, починаючи з області біля початку системи координат, статистично зменшуються, прямуючи до осі $\boldsymbol{x}$, але не круто, і чітко проглядається згасаюча періодична складова, то для апроксимації тренду цього поля точок доцільно використовувати трансцендентний вираз

$$f_2(x) = Y_0 e^{-\alpha x} \sin(\omega x), \tag{2.41}$$

а в разі, якщо зі статистичним збільшенням абсцис $\boldsymbol{x}$ ординати $\boldsymbol{y}$ точок поля експериментальних даних, стартуючи з функціональної осі, мають статистичну тенденцію круто зменшуватись, прямуючи до осі $\boldsymbol{x}$, і чітко проглядається згасаюча періодична складова, то для апроксимації тренду цього поля точок доцільно використовувати трансцендентний вираз

$$f_2(x) = Y_0 e^{-\alpha x} \cos(\omega x), \tag{2.42}$$

при цьому критерієм оптимальності при використанні для апроксимації тренду поля експериментальних точок функції (2.41) буде вираз

$$\Sigma_2 = \sum_{i=0}^{n-1} (y_i - Y_0 e^{-\alpha x_i} \sin(\omega x_i))^2, \tag{2.43}$$

а при використанні для апроксимації тренду поля експериментальних точок функції (2.42) критерієм оптимальності буде вираз

$$\Sigma_3 = \sum_{i=0}^{n-1} (y_i - Y_0 e^{-\alpha x_i} co\, s(\omega x_i))^2, \tag{2.44}$$

і, як у випадку використання критерію (2.43), так і у випадку використання критерію (2.44), для визначення оптимальних числових значень параметрів $\boldsymbol{Y_0, \alpha, \omega}$ потрібно брати від цих критеріальних виразів частинні похідні за цими параметрами, прирівнювати ці частинні похідні нулю і розв'язувати систему трьох рівнянь з трьома невідомими, якими є ці параметри;

4) при апроксимації тренду поля експериментальних точок відрізком ряду Фур'є, обчислюючи коефіцієнти Фур'є з використанням інтегралів (2.33), (2.34), ми, водночас, відфільтровуємо з експериментальних даних усі випадкові завади з нульовим середнім значенням, тому що інтегрування – це математична оболонка процедури фільтрації даних від завад цього класу;

5) апроксимуючи тренд поля експериментальних точок відрізками рядів, ми по каналу зв'язку уже можемо передавати не масиви експериментальних даних, які можуть бути аж надто великими, а лише коефіцієнти цих відрізків рядів, з використанням яких сторона, що приймає, може сама генерувати ці дані; ця операція називається ***стисканням даних***.

2.4 Розв’язання алгебраїчних рівнянь та їх систем

2.4.1 Розв’язання алгебраїчних рівнянь

Як відомо ще зі шкільних підручників з алгебри, до класу алгебраїчних належать рівняння формату

$$a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \cdots + a_1 x + a_0 = 0, \qquad (2.45)$$

число коренів яких дорівнює їх порядку ***n.*** І якщо це число є непарним, то серед коренів обов’язково матиме місце хоча б один корінь, який є дійсним числом; а якщо це число є парним, то серед коренів алгебраїчного рівняння (2.45) дійсних може і не бути, а всі вони можуть бути парами комплексно-спряжених чисел. Для алгебраїчного рівняння першого порядку

$$a_1 x + a_0 = 0, \qquad (2.46)$$

корінь x^* знаходиться за виразом

$$x^* = -\frac{a_0}{a_1}, \qquad (2.47)$$

для алгебраїчного рівняння другого порядку

$$a_2 x^2 + a_1 x + a_0 = 0, \qquad (2.48)$$

пара коренів $x_1^*,\ x_2^*$ знаходиться за виразом

$$x_{1(2)}^* = \frac{-a_1 \pm \sqrt{a_1^2 - 4a_0 a_2}}{2a_2}, \qquad (2.49)$$

а для алгебраїчного рівняння третього порядку

$$a_3 x^3 + a_2 x^2 + a_1 x + a_0 = 0 \qquad (2.50)$$

і вище готових формул для прямого обчислення їх коренів, якщо не брати до уваги екзотичні формули Кардано, що справедливі лише для спеціально сконструйованих рівнянь третього порядку, не існує; їх розв’язують методами послідовних наближень, найбільш простим із яких є метод приведення рівняння ***n***-го порядку (2.45) до такого формату, в якому у лівій частині знаходитиметься лише змінна ***x*** у першому степені, а усі інші члени виразу (2.45) перейдуть у праву частину, тобто, до формату

$$x_{(i+1)} = -\frac{1}{a_1}\left(a_0 + a_2 x_{(i)}^2 + a_3 x_{(i)}^3 + \cdots + a_n x_{(i)}^n\right), \qquad (2.51)$$

в якому ***i = 0, 1, 2, …, m*** – крок наближення, останнім із яких є той, для якого виконується нерівність

$$\left|x^*_{(m+1)} - x^*_{(m)}\right| < \varepsilon, \tag{2.52}$$

де ε – вибраний рівень похибки обчислення кореня x^* алгебраїчного рівняння (2.45), а коренем визнається число $x^*_{(m+1)}$.

2.4.2 Розв'язання систем алгебраїчних рівнянь

В процесі розв'язання значної кількості прикладних задач виникає потреба в розв'язанні системи ***n*** лінійних рівнянь формату

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}\,x_3 + \cdots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n = b_2, \\ \cdots\ \cdots\ \cdots\ \cdots \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nn}x_n = b_n, \end{cases} \tag{2.53}$$

яка у матричній формі запису має вигляд

$$\boldsymbol{A} \cdot \boldsymbol{X} = \boldsymbol{B}, \tag{2.54}$$

де квадратна матриця ***A*** та матриці-стовпці ***X, B*** – це взяті в квадратні дужки сукупності чисел та символів, які для системи лінійних рівнянь (2.53) записуються так:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \ldots a_{1n} \\ a_{21} & a_{22} & a_{32} \ldots a_{2n} \\ & \ldots & \\ a_{n1} & a_{n2} & a_{n3} \ldots a_{nn} \end{bmatrix} (n \times n),\ X = \begin{bmatrix} x_1 \\ x_2 \\ \ldots \\ x_n \end{bmatrix} (n \times 1),\ B = \begin{bmatrix} b_1 \\ b_2 \\ \ldots \\ b_n \end{bmatrix} (n \times 1). \tag{2.55}$$

Потрібно взяти до уваги, що в круглих дужках справа від матриці вказується її розмірність, причому перше число вказує на кількість рядків, а друге число – на кількість стовпців.

Матриця A^{-1}, яка після перемноження її на матрицю A породжує одиничну матрицю I, називається матрицею, оберненою до матриці A.

А одинична матриця – це матриця, яка має формат

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 \ldots 0 \\ 0 & 1 & 0 & 0 \ldots 0 \\ 0 & 0 & 1 & 0 \ldots 0 \\ & & \ldots & \\ 0 & 0 & 0 & 0 \ldots 1 \end{bmatrix} (n \times n). \tag{2.56}$$

Результатом перемноження одиничної матриці на матрицю-стовпець залишається та ж матриця-стовпець.

Отже, якщо ми створимо матрицю $\boldsymbol{A^{-1}}$ і помножимо на цю матрицю зліва обидві частини матричного рівняння (2.54), то отримаємо матричний вираз

$$\boldsymbol{X = A^{-1} \cdot B}, \tag{2.57}$$

який є розв'язком матричного рівняння (2.54).

Як витікає з виразів (2.53), (2.54), (2.55) у разі перемноження квадратної матриці $\boldsymbol{A}$ на матрицю-стовпець $\boldsymbol{X}$ з тією ж кількістю рядків отримаємо матрицю-стовпець $\boldsymbol{B}$, перший член якої дорівнює сумі добутків членів першого рядка матриці $\boldsymbol{A}$ на відповідні члени матриці-стовпця $\boldsymbol{X}$, другий член якої дорівнює сумі добутків членів другого рядка матриці $\boldsymbol{A}$ на відповідні члени матриці-стовпця $\boldsymbol{X}$ – і так аж до останнього члена матриці-стовпця $\boldsymbol{B}$.

А обернена матриця $\boldsymbol{A^{-1}}$ створюється за таким алгоритмом: кожен член матриці $\boldsymbol{A}$ заміняється його алгебраїчним доповненням, що є визначником матриці на порядок меншої розмірності, яка створюється викреслюванням рядка і стовпця матриці $\boldsymbol{A}$, на перетині яких знаходиться цей елемент, потім цей визначник множиться на $(-1)^{i+j}$, де $\boldsymbol{i}$ – це номер рядка, а $\boldsymbol{j}$ – номер стовпця, на перетині яких розміщується елемент матриці $\boldsymbol{A}$, який заміщується його алгебраїчним доповненням; після цього обчислюється визначник $|\boldsymbol{A}|$ матриці $\boldsymbol{A}$, на який ділиться кожне створене алгебраїчне доповнення. Нижче цей алгоритм демонструється на прикладі матриці $\boldsymbol{A}$ розмірності 3 на 3.

$$A = \begin{bmatrix} 2 & 4 & 1 \\ 5 & 2 & 6 \\ 1 & 3 & 1 \end{bmatrix} \tag{2.58}$$

$$|A| = 2 \cdot 2 \cdot 1 + 4 \cdot 6 \cdot 1 + 5 \cdot 3 \cdot 1 - 1 \cdot 2 \cdot 1 - 2 \cdot 6 \cdot 3 - 5 \cdot 4 \cdot 1 = -15$$

$$A^{-1} = \frac{1}{-15}\begin{bmatrix} \begin{vmatrix} 2 & 6 \\ 3 & 1 \end{vmatrix} & -\begin{vmatrix} 5 & 6 \\ 1 & 1 \end{vmatrix} & \begin{vmatrix} 5 & 2 \\ 1 & 3 \end{vmatrix} \\ -\begin{vmatrix} 4 & 1 \\ 3 & 1 \end{vmatrix} & \begin{vmatrix} 2 & 1 \\ 1 & 1 \end{vmatrix} & -\begin{vmatrix} 2 & 4 \\ 1 & 3 \end{vmatrix} \\ \begin{vmatrix} 4 & 1 \\ 2 & 6 \end{vmatrix} & -\begin{vmatrix} 2 & 1 \\ 5 & 6 \end{vmatrix} & \begin{vmatrix} 2 & 4 \\ 5 & 2 \end{vmatrix} \end{bmatrix} = \begin{bmatrix} \frac{16}{15} & -\frac{1}{15} & -\frac{13}{15} \\ \frac{1}{15} & -\frac{1}{15} & \frac{2}{15} \\ -\frac{22}{15} & \frac{7}{15} & \frac{16}{15} \end{bmatrix}. \tag{2.59}$$

Методичні рекомендації до підрозділу 2.4. Завершуючи викладення матеріалу цього підрозділу викладачеві МЗКО обов'язково потрібно звернути увагу студентів на те, що:

1) перед початком реалізації алгоритму послідовних наближень до кореня алгебраїчного рівняння потрібно спочатку, задавшись кількома значеннями незалежної змінної, визначити кілька значень функції та зупинити цей процес після першої ж зміни її знака, і те значення незалежної змінної, при реалізації якого знак функції змінився, взяти за початкове для алгоритму послідовних наближень до кореня;

2) після обчислення першого кореня p_1 алгебраїчного рівняння, функцію, прирівнюванням якої до нуля це рівняння створюється, потрібно

розділити на двочлен (x - p_1) і стосовно частки, що залишилась від ділення, для обчислення другого кореня p_2 цього алгебраїчного рівняння виконати дії, описані в попередньому пункті;

3) в разі, якщо після підстановки послідовності значень незалежної змінної в алгебраїчне рівняння знак його функціональної частини не змінюється в області значень цієї змінної, допустимих за умовами постановки задачі, то це рівняння дійсних коренів у цій області не має;

4) множити одна на одну можна лише матриці, число стовпців у першої з яких дорівнює числу рядків у другій, і що матриця, яку отримаємо в результаті перемноження, матиме число рядків першої матриці-множника та число стовпців другої матриці-множника, тобто від множення матриці з розмірністю $(n\times 1)$ на матрицю з розмірністю $(1\times n)$ отримаємо матрицю з розмірністю $(n \times n)$, а від множення матриці з розмірністю $(1\times n)$ на матрицю з розмірністю $(n\times 1)$ отримаємо матрицю з розмірністю (1×1).

2.5 Розв'язання диференціальних рівнянь та їх систем

Почнемо викладення матеріалу цього підрозділу з того, що нагадаємо деякі означення, відомі з підручників з диференціальних рівнянь.

Диференціальним називається рівняння, в лівій частині якого маємо математичну конструкцію $\boldsymbol{f(t)}$, що містить в собі деяку функцію $\boldsymbol{y(t)}$ незалежної змінної $\boldsymbol{t}$ та її похідних $\boldsymbol{y'(t)}$, $\boldsymbol{y''(t)}, \boldsymbol{y'''(t)}, \dots, \boldsymbol{y^{(n)}(t)}$, а в правій частині маємо математичну конструкцію $\boldsymbol{g(t)}$, що містить в собі деяку функцію $\boldsymbol{x(t)}$ незалежної змінної $\boldsymbol{t}$ та її похідних $\boldsymbol{x'(t)}$, $\boldsymbol{x''(t)}$, $\boldsymbol{x'''(t)}$, ... , $\boldsymbol{x^{(m)}(t)}$.

Порядок $\boldsymbol{k = 1, 2, \dots, n}$ диференціального рівняння визначається порядком старшої похідної $\boldsymbol{y^{(n)}(t)}$ в його лівій частині.

Аналітичний розв'язок мають лише ті диференціальні рівняння, порядок старшої похідної в лівій його частині є не меншим порядку старшої похідної у правій його частині, тобто, за умови $\boldsymbol{m \le n}$.

Якщо математична конструкція $\boldsymbol{f(t)}$ в лівій частині диференціального рівняння являє собою зважену відповідними коефіцієнтами суму функції $\boldsymbol{y(t)}$ та її похідних, тобто, коли це диференціальне рівняння має вигляд

$$a_n \frac{d^n y}{dt^n} + a_{n-1} \frac{d^{n-1} y}{dt^{n-1}} + \dots + a_1 \frac{dy}{dt} + a_0 y = g(t), \qquad (2.60)$$

то воно називається лінійним диференціальним рівнянням ***n-го*** порядку.

Для того, щоб знайти загальний розв'язок лінійного диференціального рівняння (2.60), потрібно задати числові значення ваговим коефіцієнтам $a_0, a_1, \dots, a_n$ в його лівій частині і задати функцію $\boldsymbol{g(t)}$ в його правій частині, яка може мати вигляд

$$g(t) = b_m \frac{d^m x}{dt^m} + b_{m-1} \frac{d^{m-1} x}{dt^{m-1}} + b_1 \frac{dx}{dt} + b_0 x \qquad (2.61)$$

та яка завжди може бути визначеною за виразом (2.61), якщо відомими є вагові коефіцієнти $b_0, b_1, \dots, b_m$ і задана функція $\boldsymbol{x(t)}$, ще до початку розв'язання диференціального рівняння.

Але, щоб виділити з загального поля розв'язків лінійного диференціального рівняння (2.60), яке графічно являтиме собою сукупність траєкторій на площині ***ty,*** єдину траєкторію $\boldsymbol{y(t)}$ необхідно задати координати точки, з якої ця траєкторія стартуватиме. Ці координати, які називають початковими умовами, для лінійного диференціального рівняння 1-го порядку задаються виразом

$$y(0) = y_0, \qquad (2.62)$$

для лінійного диференціального рівняння 2-го порядку задаються виразами:

$$\begin{cases} y(0) = y_0, \\ y'(0) = y_0', \end{cases} \qquad (2.63)$$

а для лінійного диференціального рівняння ***n-го*** порядку задаються виразами:

$$\begin{cases} y(0) = y_0, \\ y'(0) = y_0', \\ \dots \\ y^{(n-1)}(0) = y_0^{(n-1)}. \end{cases} \qquad (2.64)$$

Цілком очевидно, що в правих частинах початкових умов (2.62)–(2.64) вписані числа, якими задаються координати точки початку траєкторії конкретного розв'язку лінійного диференціального рівняння у відповідному до його порядку просторі.

В разі, якщо лінійне диференціальне рівняння (2.60) є математичною моделлю динамічного об'єкта, то функція $\boldsymbol{x(t)}$, що задається у правій частині цього рівняння, описуватиме сигнал, який надходить із зовнішнього середовища на вхід цього динамічного об'єкта, а функція $\boldsymbol{y(t)}$, яка є розв'язком диференціального рівняння, описуватиме реакцію цього об'єкта на сигнал, що надійшов на його вхід.

Якщо в лінійному диференціальному рівнянні (2.60) праву частину прирівняти нулю, то отримаємо лінійне диференціальне рівняння у форматі

$$a_n \frac{d^n y}{dt^n} + a_{n-1} \frac{d^{n-1} y}{dt^{n-1}} + \dots + a_1 \frac{dy}{dt} + a_0 y = 0, \qquad (2.65)$$

яке називають однорідним лінійним диференціальним рівнянням ***n-го*** порядку, для якого визначають характеристичне рівняння, що є алгебраїчним і має вигляд

$$a_n p^n + a_{n-1} p^{n-1} + a_{n-2} p^{n-2} + \cdots + a_1 p + a_0 = 0. \qquad (2.66)$$

Зіставлюючи вирази (2.65) та (2.66), бачимо, що характеристичне рівняння (2.66) формується з однорідного лінійного диференціального рівняння (2.65) заміною похідних порядку ***k,*** де ***k = 0, 1, 2, …, n,*** на відповідні степені змінної ***p.*** Від числових значень коренів характеристичного рівняння (2.66), як буде показано нижче на прикладах, залежить форма розв'язку $y_{\text{п}}(t)$ однорідного лінійного диференціального рівняння (2.65).

Якщо ж початкові умови (2.64) є нульовими, тобто, в правих частинах кожного з виразів у (2.64) стоять нулі, то матимемо

$$y_{\text{п}}(t) = 0. \qquad (2.67)$$

У цьому випадку розв'язок $y_x(t)$ лінійного диференціального рівняння (2.60) визначатиметься лише функцією $x(t)$, яка формує за виразом (2.61) його праву частину $g(t)$.

В загальному випадку, коли мають місце і ненульові початкові умови і ненульовою є функція $x(t)$, розв'язком лінійного диференціального рівняння (2.60) буде функція

$$y(t) = y_{\text{п}}(t) + y_x(t). \qquad (2.68)$$

А завершимо ми цей вступ в теорію диференціальних рівнянь зауваженням, що в разі, якщо математична конструкція $\boldsymbol{f(t)}$, що містить в собі деяку функцію $\boldsymbol{y(t)}$ незалежної змінної $\boldsymbol{t}$ та її похідних $\boldsymbol{y'(t)}$, $\boldsymbol{y''(t)}$, $\boldsymbol{y'''(t)}$, … , $\boldsymbol{y^{(n)}(t)}$, в лівій частині диференціального рівняння не є зваженою відповідними коефіцієнтами сумою функції $\boldsymbol{y(t)}$ та її похідних, а в ній є і члени, що являють собою степені від якихось із цих складових або їх добутками, наприклад,

$$a_n \frac{d^n y}{dt^n} + a_{n-1} \frac{d^{n-1} y}{dt^{n-1}} + \cdots + a_1 \left(\frac{dy}{dt}\right)^2 + a_0 y = g(t), \qquad (2.69)$$

або

$$a_n \frac{d^n y}{dt^n} + a_{n-1} \frac{d^{n-1} y}{dt^{n-1}} + \cdots + a_1 \frac{dy}{dt} y + a_0 y = g(t), \qquad (2.70)$$

або

$$a_n \frac{d^n y}{dt^n} + a_{n-1} \frac{d^{n-1} y}{dt^{n-1}} + \cdots + a_1 \frac{dy}{dt} + a_0 y^2 = g(t), \qquad (2.71)$$

то таке диференціальне рівняння уже входитиме в клас нелінійних диференціальних рівнянь, розв'язок для якого уже не можна записати у вигляді (2.68), оскільки для відповідного йому однорідного рівняння не можна скласти характеристичне рівняння (2.66), а тому не можна для нього

виокремити складову розв'язку $y_{\text{п}}(t)$, яка формується коренями характеристичного рівняння.

Для нелінійних диференціальних рівнянь не існують узагальнені методи розв'язання, а кожне з них розв'язується індивідуально і, як правило, методом послідовних наближень після його трансформації у систему диференціальних рівнянь першого порядку, про що мова піде далі.

2.5.1 Розв'язання диференціальних рівнянь

Нехай задане диференціальне рівняння 1-го порядку у вигляді

$$a_1 \frac{dy}{dt} + a_0 y = b_1 \frac{dx}{dt} + b_0 x \tag{2.72}$$

з початковою умовою

$$y(0) = y_0. \tag{2.73}$$

Потрібно знайти розв'язок диференціального рівняння (2.72) за умови, щоб його траєкторія починалась в точці, заданій умовою (2,73).

Розв'язок шукатимемо у вигляді (2.68), вважаючи, що

$$x(t) = t^2. \tag{2.74}$$

Отже, спочатку знайдемо складову $y_{\text{п}}(t)$ розв'язку (2.68). Для цього, прирівнявши нулю праву частину рівняння (2.72), знайдемо розв'язок відповідного рівнянню (2.72) однорідного лінійного диференціального рівняння

$$a_1 \frac{dy}{dt} + a_0 y = 0. \tag{2.75}$$

Трансформуємо рівняння (2.75) до вигляду

$$a_1 \frac{dy}{dt} = -a_0 y. \tag{2.76}$$

Обидві частини рівняння (2.76) помножимо на ***dt*** і розділимо на ***y*** та на $\boldsymbol{a_1}$. В результаті цих дій отримаємо рівняння

$$\frac{dy}{y} = -\frac{a_0}{a_1} dt. \tag{2.77}$$

Проінтегруємо обидві частини рівняння (2.77). Оскільки інтеграли табличні, то результат інтегрування можемо записати одразу. Цей результат матиме вигляд

$$\ln|y| = -\frac{a_0}{a_1} t + C, \tag{2.78}$$

де $\boldsymbol{C}$ – поки що невідома нам стала.

Уведемо позначення:

$$p_1 = -\frac{a_0}{a_1}, \tag{2.79}$$

$$C = \ln|C_1|. \tag{2.80}$$

Підставляючи вирази (2.79) та (2.80) у рівняння (2.78), отримаємо його у вигляді

$$\ln\left|\frac{y}{C_1}\right| = p_1 t. \tag{2.81}$$

А з рівняння (2.81) уже нескладно отримати вираз

$$y = C_1 e^{p_1 t}, \tag{2.82}$$

який є масштабованим значеннями константи $\boldsymbol{C_1}$ полем експоненціальних розв'язків однорідного лінійного диференціального рівняння (2,75).

А далі, скориставшись розв'язком (2.82) однорідного лінійного диференціального рівняння (2.75) та методом варіації сталої, знайдемо загальний розв'язок ***y(t)*** лінійного диференціального рівняння (2.72).

Щоб визначити структуру цього розв'язку спочатку у праву частину рівняння (2.72), яка має вигляд

$$g(t) = b_1 \frac{dx}{dt} + b_0 x, \tag{2.83}$$

підставимо вираз (2.74). В результаті цієї підстановки матимемо

$$g(t) = 2b_1 t + b_0 t^2. \tag{2.84}$$

З урахуванням виразу (2.84) рівняння (2.72) набуває вигляду

$$a_1 \frac{dy}{dt} + a_0 y = 2b_1 t + b_0 t^2. \tag{2.85}$$

Будемо шукати загальний розв'язок рівняння (2.85) у вигляді (2.82), але вважаючи сталу C_1 не константою, а функцією $C_1(t)$, тобто у вигляді структури

$$y = C_1(t) e^{p_1 t}, \tag{2.86}$$

в якій прояв однорідності лівої частини рівняння (2.85) уже забезпечений наявністю експоненти, а для прояву неоднорідності правої частини рівняння (2.85) ми підставимо вираз (2.86) у це рівняння, аби реалізувати цей прояв через функцію $C_1(t)$.

Отже, підставляючи вираз (2.86) у рівняння (2.85) та пам'ятаючи чому дорівнює похідна від добутку двох функцій, отримаємо

$$a_1 \left(\frac{dC_1(t)}{dt} e^{p_1 t} + p_1 C_1(t) e^{p_1 t}\right) + a_0 C_1(t) e^{p_1 t} = 2b_1 t + b_0 t^2. \tag{2.87}$$

А розкриваючи дужки та беручи до уваги вираз (2.79) з виразу (2.87) отримаємо

$$\frac{dC_1(t)}{dt} = \left(\frac{2b_1}{a_1} t + \frac{b_0}{a_1} t^2 \right) e^{-p_1 t}. \quad (2.88)$$

Інтегруючи вираз (2.88), матимемо

$$C_1(t) = \frac{2b_1}{a_1} \int t\, e^{-p_1 t} dt + \frac{b_0}{a_1} \int t^2\, e^{-p_1 t} dt + C_1^*. \quad (2.89)$$

А беручи інтеграли частинами з виразу (2.89) матимемо

$$C_1(t) = -\frac{2b_1}{a_1 p_1} \left(t + \frac{1}{p_1} \right) e^{-p_1 t} - \frac{b_0}{a_1 p_1} \left(t^2 + \frac{2}{p_1} t + \frac{2}{p_1^2} \right) e^{-p_1 t} + C_1^*. \quad (2.90)$$

Після підстановки виразів (2.90) та (2.79) у вираз (2.86) і спрощення отримаємо загальний розв'язок лінійного диференціального рівняння (2.85) у вигляді

$$y(t) = C_1^* e^{-\frac{a_0}{a_1} t} + \left(\frac{2a_0^2 b_0}{a_1^2 a_0} - \frac{2a_1 b_1}{a_0^2} \right) + \left(\frac{2b_1}{a_0} - \frac{2a_1 b_0}{a_0^2} \right) t + \frac{b_0}{a_0} t^2. \quad (2.91)$$

Легко бачити, що у виразі (2.91) перший член правої частини являє собою складову, зумовлену впливом початкових умов на однорідне рівняння, а інші три члени характеризують складову, зумовлену неоднорідністю та структурою правої частини неоднорідного диференціального рівняння.

Конкретизуємо параметри рівняння, яке ми розв'язуємо. Нехай

$$a_0 = 1;\ \ a_1 = 2;\ \ b_0 = 2;\ \ b_1 = 1. \quad (2.92)$$

Підставивши параметри з виразів (2.92) у вираз (2.91), отримаємо загальний розв'язок лінійного диференціального рівняння (2.85) у вигляді

$$y(t) = C_1^* e^{-\frac{t}{2}} - 3 - 6t + 2t^2. \quad (2.93)$$

А конкретизуючи початкові умови (2.73), наприклад, нехай

$$y(0) = 2, \quad (2.94)$$

та підставляючи ці початкові умови (2.94) у вираз (2.93), отримаємо рівняння

$$2 = C_1^* - 3, \quad (2.95)$$

з якого витікає, що

$$C_1^* = 5. \quad (2.96)$$

Підставляючи вираз (2.96) у (2.93), отримаємо траєкторію розв'язку лінійного диференціального рівняння (2.85), яка починається з точки, заданої виразом (2.94).

Ми так детально розписали алгоритм розв'язання лінійного диференціального рівняння (2.85) задля того, щоб показати, наскільки процес розв'язання навіть диференціального рівняння 1-го порядку є затратним в часі.

Але у будь-якому підручнику з диференціальних рівнянь можна знайти інформацію про те, що у такий же спосіб можна розв'язувати і лінійне диференціальне рівняння 2-го порядку

$$a_2\frac{d^2y}{dt^2}+a_1\frac{dy}{dt}+a_0y=g(t), \tag{2.97}$$

але у цьому випадку ми матимемо розв'язок однорідного рівняння, яке отримаємо прирівненням до нуля лівої частини виразу (2.97), у вигляді

$$y_n(t)=C_1e^{p_1t}+C_2e^{p_2t}, \tag{2.98}$$

де p_1, p_2 – корені характеристичного рівняння

$$a_2p^2+a_1p+a_0=0, \tag{2.99}$$

і вже дві сталі C_1, C_2 потрібно буде оголошувати функціями $C_1(t)$, $C_2(t)$, що приведе до необхідності для їх конкретизації розв'язувати вже систему двох диференціальних рівнянь складених стосовно них з урахуванням правої частини рівняння (2.97).

Але, знову ж таки, в будь-якому підручнику з диференціальних рівнянь можна знайти інформацію про те, що для цих функцій $C_1(t), C_2(t)$, які після їх розкриття визначатимуть загальний розв'язок диференціального рівняння (2.97) у вигляді

$$y(t)=C_1(t)e^{p_1t}+C_2(t)e^{p_2t}, \tag{2.100}$$

застосуванням методу варіації сталих уже отримана система диференціальних рівнянь стосовно їх похідних $C_1'(t), C_2'(t)$, яка має такий вигляд:

$$\begin{cases} C_1'(t)e^{p_1t}+C_2'(t)e^{p_2t}=0, \\ C_1'(t)p_1e^{p_1t}+C_2'(t)p_2e^{p_2t}=g(t). \end{cases} \tag{2.101}$$

Тож, отримавши з першого рівняння цієї системи (2.101) вираз

$$C_1'(t)e^{p_1t}=-C_2'(t)e^{p_2t} \tag{2.102}$$

та підставивши його в друге рівняння системи (2.101), отримаємо диференціальне рівняння

$$-C_2'(t)p_1e^{p_2t}+C_2'(t)p_2e^{p_2t}=g(t), \tag{2.103}$$

яке легко приводиться до вигляду

$$C_2'(t) = \frac{1}{p_2 - p_1} g(t) e^{-p_2 t}, \qquad (2.104)$$

та інтегруючи обидві частини якого знайдемо, що

$$C_2(t) = \frac{1}{p_2 - p_1} \int g(t) e^{-p_2 t} dt \; + C_2^*. \qquad (2.105)$$

Якщо задати функцію ***g(t)*** у вигляді (2.84), то отримаємо вираз, подібний до виразу (2.89) з тією лише різницею, що поміняються індекси, а на місці a_1 фігуруватиме $(p_2 - p_1)$. Отже матимемо вираз

$$C_2(t) = \frac{2b_1}{p_2 - p_1} \int t \; e^{-p_2 t} dt + \frac{b_0}{p_2 - p_1} \int t^2 \; e^{-p_2 t} dt + C_2^*. \qquad (2.106)$$

Тож, використовуючи результати інтегрування з виразу (2.90), знайдемо, що

$$C_2(t) = -\frac{2b_1}{(p_2 - p_1)p_2} \left(t + \frac{1}{p_2} \right) e^{-p_2 t} - \frac{b_0}{(p_2 - p_1)p_2} \left(t^2 + \frac{2}{p_2} t + \frac{2}{p_2^2} \right) e^{-p_2 t} + C_2^*. \qquad (2.107)$$

А підставляючи вираз (2.104) у вираз (2.102), отримаємо

$$C_1'(t) = \frac{1}{p_1 - p_2} g(t) e^{-p_1 t}. \qquad (2.108)$$

Тобто, отримаємо диференціальне рівняння подібне до (2.104), що дає нам право його розв'язок записати у вигляді (2.107), лише змінивши індекси. Отже, матимемо

$$C_1(t) = -\frac{2b_1}{(p_1 - p_2)p_1} \left(t + \frac{1}{p_1} \right) e^{-p_1 t} - \frac{b_0}{(p_1 - p_2)p_1} \left(t^2 + \frac{2}{p_1} t + \frac{2}{p_1^2} \right) e^{-p_1 t} + C_1^*. \qquad (2.109)$$

А тепер знайдемо складові розв'язку нашого диференціального рівняння

$$C_1(t) e^{p_1 t}, \;\; C_2(t) e^{p_2 t}. \qquad (2.110)$$

Отже, матимемо:

$$C_1(t) e^{p_1 t} = -\frac{2b_1}{(p_1 - p_2)p_1^2} (p_1 t + 1) - \frac{b_0}{(p_1 - p_2)p_1^3} (p_1^2 t^2 + 2p_1 t + 2) + C_1^* e^{p_1 t}, \qquad (2.111)$$

$$C_2(t) e^{p_2 t} = -\frac{2b_1}{(p_2 - p_1)p_2^2} (p_2 t + 1) - \frac{b_0}{(p_2 - p_1)p_2^3} (p_2^2 t^2 + 2p_2 t + 2) + C_2^* e^{p_2 t}. \qquad (2.112)$$

Конкретизуємо вирази (2.111), (2.112).

Нехай диференціальне рівняння (2.97) має такі значення параметрів:

$$a_2 = 1; a_1 = 6; a_0 = 8; b_1 = 1; b_0 = 2. \qquad (2.113)$$

При таких значеннях параметрів характеристичне рівняння (2.99) набуде вигляду

$$p^2 + 6p + 8 = 0. \qquad (2.114)$$

Це рівняння має корені:

$$p_1 = -2;\ \ p_2 = -4. \tag{2.115}$$

Підставляючи значення параметрів із виразу (2.113) та коренів із виразу (2.115) у вирази (2.111) та (2.112), отримаємо:

$$C_1(t)e^{-2t} = -\frac{2}{8}(-2t+1) + \frac{2}{16}(4t^2 - 4t + 2) + C_1^* e^{-2t}, \tag{2.116}$$

$$C_2(t)e^{-4t} = \frac{2}{32}(-4t+1) - \frac{2}{128}(16t^2 - 8t + 2) + C_2^* e^{-4t}. \tag{2.117}$$

А підставляючи вирази (2.116) та (2.117) у вираз (2.100) та спрощуючи його, отримаємо загальний розв'язок нашого диференціального рівняння у вигляді

$$y(t) = C_1^* e^{-2t} + C_2^* e^{-4t} + \frac{1}{4}t^2 - \frac{1}{8}t + \frac{1}{32}. \tag{2.118}$$

Диференціюючи вираз (2.118), знайдемо, що

$$y'(t) = -2C_1^* e^{-2t} - 4C_2^* e^{-4t} + \frac{1}{2}t - \frac{1}{8}. \tag{2.119}$$

Задамо початкові умови. Нехай:

$$\begin{cases} y(0) = 0, \\ y'(0) = 1. \end{cases} \tag{2.120}$$

Із виразів (2.118), (2.119), (2.120) отримуємо систему двох рівнянь

$$\begin{cases} C_1^* + C_2^* + \frac{1}{32} = 0, \\ -2C_1^* - 4C_2^* - \frac{1}{8} = 1, \end{cases} \tag{2.121}$$

яку приведемо до вигляду

$$\begin{cases} C_1^* + C_2^* = -\frac{1}{32}, \\ -C_1^* - 2C_2^* = \frac{9}{16}. \end{cases} \tag{2.122}$$

Розв'язуючи систему (2.122), знайдемо, що

$$C_1^* = \frac{16}{32}, \qquad C_2^* = -\frac{17}{32}. \tag{2.123}$$

Підставляючи вирази (2.123) у вираз (2.118), визначаємо траєкторію розв'язку нашого диференціального рівняння з початком у точці, заданої початковими умовами (2.120), у вигляді

$$y(t) = \frac{16}{32}e^{-2t} - \frac{17}{32}e^{-4t} + \frac{1}{4}t^2 - \frac{1}{8}t + \frac{1}{32}. \tag{2.124}$$

Як бачимо, для розв'язання неоднорідного лінійного диференціального рівняння 2-го порядку методом варіації сталих ми витратили чимало часу на різні перетворення та підстановки, тож ще більше втрат часу та клопоту матимемо, якщо у такий же спосіб вручну будемо розв'язувати неоднорідні лінійні диференціальні рівняння з порядком, вищим 2-го.

Але, якщо права частина диференціального рівняння (2.97) є поліномом за степенями незалежної змінної ***t***, то краще застосовувати більш простий метод розв'язання цього рівняння та йому подібних, в якому використовується розкладання розв'язку ***y(t)*** у степеневий ряд

$$y(t) = y(0) + \frac{y'(0)}{1!}t + \frac{y''(0)}{2!}t^2 + \cdots, \tag{2.125}$$

який після першого диференціювання стає степеневим рядом

$$y'(t) = y'(0) + y''(0)t + \frac{y'''(0)}{2!}t^2 + \cdots, \tag{2.126}$$

а після другого диференціювання стає степеневим рядом

$$y''(t) = y''(0) + y'''(0)t + \frac{y^{(4)}(0)}{2!}t^2 + \cdots. \tag{2.127}$$

Підставляючи вирази (2.125), (2.126), (2.127) та (2.84) в рівняння (2.97) та здійснюючи перенесення усіх його членів в ліву частину, отримаємо

$$\begin{gathered} a_2\left(y''(0) + y'''(0)t + \frac{y^{(4)}(0)}{2!}t^2 + \cdots\right) + \\ +a_1\left(y'(0) + y''(0)t + \frac{y'''(0)}{2!}t^2 + \cdots\right) + \\ +a_0(y(0) + \frac{y'(0)}{1!}t + \frac{y''(0)}{2!}t^2 + \cdots) - 2b_1t - b_0t^2 = 0. \end{gathered} \tag{2.128}$$

Вираз (2.128) можна переписати і так:

$$\begin{gathered} a_2y''(0) + a_1y'(0) + a_0y(0) + (a_2y'''(0) + a_1y''(0) + \\ +a_0y'(0) - 2b_1)t + (a_2\frac{y^{(4)}(0)}{2} + a_1\frac{y'''(0)}{2} + a_0\frac{y''(0)}{2} - \ b_0)t^2 + \cdots = 0. \end{gathered} \tag{2.129}$$

Вираз (2.129) є тотожністю, яка може виконуватись лише за умови, що коефіцієнти при всіх степенях змінної ***t*** дорівнюють нулю. Отже, справедливою є система рівнянь:

$$\begin{cases} a_2y''(0) + a_1y'(0) + a_0y(0) = 0, \\ a_2y'''(0) + a_1y''(0) + a_0y'(0) - 2b_1 = 0, \\ a_2\dfrac{y^{(4)}(0)}{2} + a_1\dfrac{y'''(0)}{2} + a_0\dfrac{y''(0)}{2} - b_0 = 0. \end{cases} \tag{2.130}$$

Оскільки диференціальне рівняння (2.97), яке ми розв'язуємо, має другий порядок, то для нього заданими є початкові умови у вигляді (2.63), підставляючи які в перше рівняння системи (2.130), знайдемо, що

$$y''(0) = -\frac{1}{a_2}(a_1 y_0' + a_0 y_0) = y_0''. \tag{2.131}$$

Підставляючи числові значення з виразів (2.63) та (2.131) в друге рівняння системи (2.130) знайдемо, що

$$y'''(0) = -\frac{1}{a_2}(a_1 y_0'' + a_0 y_0' - 2b_1) = y_0'''. \tag{2.132}$$

А підставляючи числові значення із виразів (2.132) та (2.131) в третє рівняння системи (2.130), знайдемо, що

$$y^{(4)}(0) = -\frac{1}{a_2}(a_1 y_0''' + a_0 y_0'' - 2b_0) = y_0^{(4)}. \tag{2.133}$$

Ну і нарешті, підставляючи задані початковими умовами (2.63) та отримані у виразах (2.131), (2.132), (2.133) числові значення у вираз (2.125), отримаємо розв'язок диференціального рівняння (2.97) у вигляді

$$y(t) = y_0 + \frac{y_0'}{1!}t + \frac{y_0''}{2!}t^2 + \frac{y_0'''}{3!}t^3 + \frac{y_0^{(4)}}{4!}t^4. \tag{2.134}$$

Якщо відрізок степеневого ряду (2.134), яким визначається розв'язок ***y(t)***, за рівнем похибки розв'язання нас задовольняє, то отриманням виразу (2.134) ми задачу розв'язання диференціального рівняння (2.97) завершуємо.

А якщо похибка розв'язання нас не влаштовує, що може мати місце при розв'язанні диференціального рівняння з порядком, вищим двох, то до відрізка степеневого ряду (2.134) додаємо нові члени з вищими степенями, коефіцієнти в яких теж обчислюємо за продемонстрованим вище алгоритмом,

Але набагато простіше і точніше процес розв'язання диференціального рівняння високого порядку реалізувати, якщо попередньо трансформувати це рівняння, наприклад, ***n-го*** порядку в ***n*** диференціальних рівнянь 1-го порядку – про це і піде мова в наступному пункті.

2.5.2 Розв'язання систем диференціальних рівнянь

Спочатку на конкретному прикладі системи двох неоднорідних диференціальних рівнянь 1-го порядку, які містять у своїй структурі дві невідомі функції та їх перші похідні та мають вигляд

$$\begin{cases} \frac{dy}{dt} + 2y + 4z = 2t, \\ \frac{dz}{dt} + y - z = 3t^2, \end{cases} \tag{2.135}$$

покажемо, як можна розв'язувати системи ***n*** лінійних диференціальних рівнянь 1-го порядку, які містять у своїй структурі ***n*** невідомих функцій та їх перших похідних, методом редукції цієї системи до одного лінійного диференціального рівняння ***n-***го порядку, яке містить у своїй структурі лише одну невідому функцію та її похідні до ***n-***ї включно.

На першому етапі розв'язання системи диференціальних рівнянь (2.135) перепишемо перше рівняння цієї системи так, щоб зліва мати лише функцію ***z(t)***, тобто, запишемо його у вигляді

$$z = \frac{1}{4}\left(2t - 2y - \frac{dy}{dt}\right). \tag{2.136}$$

На другому етапі розв'язання системи диференціальних рівнянь (2.135) продиференціюємо перше рівняння цієї системи, в результаті чого отримаємо диференціальне рівняння, що матиме вигляд

$$\frac{d^2y}{dt^2} + 2\frac{dy}{dt} + 4\frac{dz}{dt} = 2. \tag{2.137}$$

На третьому етапі розв'язання системи диференціальних рівнянь (2.135) підставимо в друге рівняння цієї системи замість функції ***z(t)*** її еквівалент у вигляді правої частини виразу (2.136), в результаті чого отримаємо диференціальне рівняння

$$\frac{dz}{dt} + y - \frac{1}{4}\left(2t - 2y - \frac{dy}{dt}\right) = 3t^2, \tag{2.138}$$

яке легко зводиться до вигляду

$$\frac{dz}{dt} = 3t^2 + \frac{1}{2}t - \frac{3}{2}y - \frac{1}{4}\frac{dy}{dt}. \tag{2.139}$$

На четвертому етапі розв'язання системи диференціальних рівнянь (2.135) підставимо вираз (2.139) в рівняння (2.137), в результаті чого отримаємо диференціальне рівняння

$$\frac{d^2y}{dt^2} + 2\frac{dy}{dt} + 4\left(3t^2 + \frac{1}{2}t - \frac{3}{2}y - \frac{1}{4}\frac{dy}{dt}\right) = 2, \tag{2.140}$$

яке легко зводиться до вигляду

$$\frac{d^2y}{dt^2} + \frac{dy}{dt} - 6y = 2 - 2t - 12t^2. \tag{2.141}$$

Послідовність дій, яку треба виконати для розв'язання неоднорідного лінійного диференціального рівняння 2-го порядку методом варіації сталих, ми навели в попередньому пункті, тож повторювати її ще й тут немає сенсу.

А для закріплення викладеного вище алгоритму наведемо ще й чотири етапи зведення системи диференціальних рівнянь (2.135) до одного диференціального рівняння 2-го порядку стосовно функції ***z(t)***.

Отже на першому етапі розв'язання перепишемо друге рівняння системи (2.135) так, щоб зліва мати лише функцію ***y(t)***, тобто, запишемо його у вигляді

$$y = 3t^2 + z - \frac{dz}{dt} \tag{2.142}$$

На другому етапі розв'язання продиференціюємо друге рівняння системи (2.135), в результаті чого отримаємо

$$\frac{d^2z}{dt^2} + \frac{dy}{dt} - \frac{dz}{dt} = 6t. \tag{2.143}$$

На третьому етапі розв'язання підставимо в перше рівняння системи (2.135) замість функції ***y(t)*** її еквівалент у вигляді правої частини виразу (2.142), в результаті чого отримаємо

$$\frac{dy}{dt} + 2\left(3t^2 + z - \frac{dz}{dt}\right) + 4z = 2t, \tag{2.144}$$

або

$$\frac{dy}{dt} = 2t - 6t^2 - 6z + 2\frac{dz}{dt}. \tag{2.145}$$

На четвертому етапі розв'язання системи (2.135) підставимо вираз (2.145) в рівняння (2.143), в результаті чого отримаємо диференціальне рівняння

$$\frac{d^2z}{dt^2} + 2t - 6t^2 - 6z + 2\frac{dz}{dt} - \frac{dz}{dt} = 6t, \tag{2.146}$$

яке легко зводиться до вигляду

$$\frac{d^2z}{dt^2} + \frac{dz}{dt} - 6z = 4t + 6t^2. \tag{2.147}$$

Послідовність дій, яку треба виконати для розв'язання і цього неоднорідного лінійного диференціального рівняння 2-го порядку методом варіації сталих, ми навели в попередньому пункті, тож повторювати її ще й тут теж немає сенсу.

Важливі зауваження:

1. Порівнюючи ліву частину диференціальних рівнянь (2.141) та (2.147), бачимо, що вони мають однакову структуру, тож ці рівняння при їх написанні в однорідному варіанті матимуть одне й те ж характеристичне рівняння

$$p^2 + p - 6 = 0, \tag{2.148}$$

яке має корені

$$p_1 = 2; \quad p_2 = -3, \tag{2.149}$$

а тому розв’язками однорідного рівняння, виписаного як для одного з них, так і для другого, буде вираз з однаковою структурою, уже вказаною нами у (2.98).

2. У разі, якщо в систему лінійних диференціальних рівнянь, яку потрібно сумісно розв’язати, входить три і більше таких рівнянь, то трудомісткість процесу трансформації цієї системи до одного рівняння більш високого порядку суттєво зростає, а тому цей процес потрібно алгоритмізувати і передати для реалізації комп’ютеру.

А далі розглянемо задачу протилежного характеру. Тобто, нехай задано неоднорідне лінійне диференціальне рівняння 3-го порядку

$$a_3\frac{d^3y}{dt^3}+a_2\frac{d^2y}{dt^2}+a_1\frac{dy}{dt}+a_0y=g(t) \tag{2.150}$$

з початковими умовами:

$$\begin{cases} y(t_0)=y_0, \\ y'(t_0)=y_0', \\ y''(t_0)=y_0''. \end{cases} \tag{2.151}$$

та з правою частиною, заданою функцією $g(t)$, і потрібно трансформувати його у систему трьох диференціальних рівнянь 1-го порядку так, щоб її, починаючи з моменту часу t_0, можна було розв’язувати методом послідовних наближень.

Позначимо:

$$y=y_1, \tag{2.152}$$

$$\frac{dy}{dt}=\frac{dy_1}{dt}=y_2, \tag{2.153}$$

$$\frac{d^2y}{dt^2}=\frac{dy_2}{dt}=y_3, \tag{2.154}$$

$$\frac{d^3y}{dt^3}=\frac{d^2y_2}{dt^2}=\frac{dy_3}{dt}. \tag{2.155}$$

З урахуванням уведених позначень диференціальне рівняння 3-го порядку (2.150) можна трансформувати у систему трьох диференціальних рівнянь 1-го порядку

$$\begin{cases} \frac{dy_1}{dt}=y_2, \\ \frac{dy_2}{dt}=y_3, \\ \frac{dy_3}{dt}=\frac{1}{a_3}g(t)-\frac{a_0}{a_3}y_1-\frac{a_1}{a_3}y_2-\frac{a_2}{a_3}y_3, \end{cases} \tag{2.156}$$

а початкові умови набудуть вигляду:

$$\begin{cases} y_1(t_0) = y_0, \\ y_2(t_0) = y_0^{'}, \\ y_3(t_0) = y_0^{''}. \end{cases} \quad (2.157)$$

Розіб'ємо відрізок $[\boldsymbol{t_0, t_n}]$ значень незалежної змінної $\boldsymbol{t}$, на якому ми бажаємо визначити розв'язок $\boldsymbol{y(t)}$ диференціального рівняння (2.150), точками $\boldsymbol{t_i, i = 0, 1, 2, \dots, n-1}$ на $\boldsymbol{n}$ малих та рівних між собою на рівні $\boldsymbol{\delta}$ відрізків

$$\Delta_i t = t_{i+1} - t_i = \delta. \quad (2.158)$$

Проінтегруємо чисельно усі три диференціальні рівняння системи (2.156) з обчисленням інтегралів за методом прямокутників, в результаті чого отримаємо алгоритм їх наближеного розв'язку з кінця в початок у вигляді:

$$\begin{cases} y_3(t_{i+1}) = \left(1 - \frac{a_2}{a_3}\delta\right) y_3(t_i) + \frac{\delta}{a_3} g(t_i) - \frac{a_0\delta}{a_3} y_1(t_i) - \frac{a_1\delta}{a_3} y_2(t_i), \\ y_2(t_{i+1}) = y_2(t_i) + y_3(t_i)\delta, \\ y_1(t_{i+1}) = y_1(t_i) + y_2(t_i)\delta, \\ i = 0, 1, 2, \dots, n. \end{cases} \quad (2.159)$$

Підставляючи в систему рівнянь (2.159) $\boldsymbol{i = 0}$, числове значення правої частини $g(t_0)$ рівняння (2.150) та початкові умови (2.157), знайдемо числові значення $y_3(t_1),\ y_2(t_1),\ y_1(t_1)$, з використанням яких та $g(t_1)$ при $\boldsymbol{i = 1}$ з системи рівнянь (2.159) знайдемо числові значення $y_3(t_2),\ y_2(t_2),\ y_1(t_2)$, і так продовжуватимемо аж до отримання числових значень $y_3(t_n), y_2(t_n),\ y_1(t_n)$.

Сукупність числових значень

$$y_1(t_0), y_1(t_1), y_1(t_2), \dots, y_1(t_n) \quad (2.160)$$

і являтиме собою в позначенні (2.152) розв'язок неоднорідного диференціального рівняння (2.150), обчислений методом послідовних наближень.

Цілком очевидно, що чим меншим ми візьмемо відрізок $\boldsymbol{\delta}$, тим точніше розв'язок диференціального рівняння (2.150) у вигляді (2.160) відображатиме розв'язок цього рівняння у вигляді $\boldsymbol{y(t)}$ для усіх значень $t \in [t_0, t_n]$.

Методичні рекомендації до підрозділу 2.5. Завершуючи викладення матеріалу цього підрозділу викладачеві МЗКО обов'язково потрібно звернути увагу студентів на те, що:

1) метод варіації сталих розв'язку однорідного диференціального рівняння розповсюджується лише на неоднорідні лінійні диференціальні

рівняння, а розв'язання з його використанням нелінійних диференціальних рівняння неможливо в принципі;

2) загальних методів розв'язання нелінійних диференціальних рівнянь, які приводили б до отримання розв'язків в аналітичному вигляді, не існує, а тому нелінійні диференціальні рівняння, як правило, розв'язуються наближеними методами, що приводять до отримання розв'язків у вигляді числових масивів;

3) нелінійні диференціальні рівняння високих порядків спочатку трансформуються у системи диференціальних рівнянь першого порядку, а потім уже розв'язуються наближеними методами.

2.6 Перетворення за Лапласом як метод розв'язання диференціальних рівнянь на комплексній площині

2.6.1 Пряме перетворення за Лапласом як метод трансформації диференціальних рівнянь в алгебраїчні

З підручників з вищої математики, теоретичних основ електротехніки та теорії автоматичного керування, відомо, що за допомогою перетворення за Лапласом

$$L\{f(t)\} = F(p) = \int_0^\infty f(t)e^{-pt}dt \qquad (2.161)$$

кожній неперервній функції f часу t, заданій на множині дійсних чисел, яка задовольняє умову $f(t)=0$ при $t<0$, умови Діріхле і називається оригіналом, можна поставити у відповідність функцію F комплексної змінної $p=\sigma+j\omega$, яка називається зображенням оригіналу на комплексній площині. Ця відповідність записується у такий спосіб:

$$f(t) \Leftrightarrow F(p). \qquad (2.162)$$

Наприклад, експоненті $e^{-\alpha t}$ при $t \geq 0$ на комплексній площині відповідає зображення

$$F(p) = \int_0^\infty e^{-\alpha t}\cdot e^{-pt}dt = \int_0^\infty e^{-(p+\alpha)t}dt = \frac{1}{-(p+\alpha)}e^{-(p+\alpha)t}\Big|_0^\infty = \frac{1}{p+\alpha}. \qquad (2.163)$$

Головною перевагою аналізу в області зображень $F(p)$, порівняно з аналізом в області оригіналів $f(t)$, є те, що за нульових початкових умов

операції диференціювання $\frac{d}{dt}$ оригіналу $f(t)$ у часовому просторі відповідає операція перемноження на комплексну змінну p його зображення $F(p)$ на комплексній площині, оскільки

$$L\left\{\frac{df}{dt}\right\} = \int_0^\infty \frac{df}{dt} e^{-pt} dt = \left(f(t)e^{-pt}\right)\Big|_0^\infty - \int_0^\infty f(t)\left(-pe^{-pt}\right)dt = \\ = p\int_0^\infty f(t)e^{-pt}dt = pF(p). \tag{2.164}$$

Узагальнюючи, матимемо

$$\frac{d^n f}{dt^n} = f^{(n)}(t) \Leftrightarrow p^n \cdot F(p). \tag{2.165}$$

А операції інтегрування оригіналу $f(t)$ у часовому просторі відповідає операція ділення на комплексну змінну p його зображення $F(p)$ на комплексній площині, оскільки

$$L\left\{\int_0^t f(\tau)d\tau\right\} = \int_0^\infty \left(\int_0^t f(\tau)d\tau\right)e^{-pt}dt = \left(\frac{1}{-p}e^{-pt}\int_0^t f(\tau)d\tau\right)\Big|_0^\infty - \int_0^\infty \left(\frac{1}{-p}e^{-pt}\right)f(t)dt = \\ = \frac{1}{p}\int_0^\infty f(t)e^{-pt}dt = \frac{1}{p}F(p) \tag{2.166}$$

Узагальнюючи, матимемо

$$\underbrace{\int_0^t \ldots \int_0^t}_{n} f(\tau_1, \tau_2, \ldots, \tau_n)\, d\tau_1\, d\tau_2 \ldots d\tau_n \Leftrightarrow \frac{F(p)}{p^n}. \tag{2.167}$$

Завдяки властивостям, вказаним вище, диференціальним та інтегральним рівнянням, записаним у часовому просторі, відповідають алгебраїчні рівняння на комплексній площині, наприклад, диференціальному рівнянню в області оригіналів $x(t)$, $y(t)$

$$a_2\frac{d^2y(t)}{dt^2} + a_1\frac{dy(t)}{dt} + a_0y(t) = b_1\frac{dx(t)}{dt} + b_0x(t) \tag{2.168}$$

на комплексній площині відповідає алгебраїчне рівняння

$$a_2p^2Y(p) + a_1pY(p) + a_0Y(p) = b_1pX(p) + b_0X(p) \tag{2.169}$$

відносно зображень $X(p)$ та $Y(p)$. Його розв'язком є функція

$$Y(p)=\frac{b_1 p+b_0}{a_2 p^2+a_1 p+a_0}X(p)=\frac{C(p)}{D(p)}. \tag{2.170}$$

А щоб ця функція стала розв'язком диференціального рівняння (2.168) у часовій області, треба здійснити стосовно неї обернене перетворення за Лапласом, про яке піде мова в наступному пункті.

2.6.2 Обернене перетворення за Лапласом як метод трансформації функцій з комплексної області в часову

Крім прямого перетворення за Лапласом, яке визначається виразом (2.161), існує і обернене перетворення за Лапласом

$$L^{-1}\{F(p)\}=\frac{1}{2\pi j}\int_{\sigma-j\infty}^{\sigma+j\infty}F(p)e^{pt}dp=\frac{1}{2\pi j}\oint F(p)e^{pt}dp=f(t), \tag{2.171}$$

згідно з яким за відомим зображенням $F(p)$ можна знайти оригінал $f(t)$. Це обернене перетворення при розв'язанні практичних задач реалізується застосуванням формул розкладання, одна з яких – для некратних полюсів p_i зображення

$$Y(p)=\frac{C(p)}{D(p)}, \tag{2.172}$$

де $C(p), D(p)$ – багаточлени за степенями p порядків m та n, відповідно, а полюси p_i – це корені рівняння

$$D(p)=0, \tag{2.173}$$

має вигляд

$$y(t)=\sum_{i=1}^{n}\frac{C(p_i)}{D'(p_i)}e^{p_i t}, \tag{2.174}$$

де

$$D'(p_i)=\left.\frac{dD}{dp}\right|_{p=p_i}. \tag{2.175}$$

А у випадку, якщо серед коренів рівняння (2.173) *n*-го порядку є кратний корінь, наприклад p_1 кратності k, тобто, коли рівняння (2.173) набуває вигляду

$$(p-p_1)^k D_{n-k}(p)=0, \tag{2.176}$$

то замість формули розкладання у вигляді (2.174) потрібно застосовувати формулу розкладання у вигляді

$$y(t)=\frac{1}{(k-1)!}\frac{d^{(k-1)}}{dp^{(k-1)}}\left[\frac{C(p)(p-p_1)^k e^{pt}}{D(p)}\right]_{p=p_1}+\sum_{i=2}^{n-1}\frac{C(p_i)}{D'(p_i)}e^{p_i t}\,, \qquad (2.177)$$

або, що одне і те ж, у вигляді

$$y(t)=\frac{1}{(k-1)!}\frac{d^{(k-1)}}{dp^{(k-1)}}\left[\frac{C(p)e^{pt}}{D_{n-k}(p)}\right]_{p=p_1}+\sum_{i=2}^{n-1}\frac{C(p_i)}{D'(p_i)}e^{p_i t}\,. \qquad 2.178)$$

Методичні рекомендації до підрозділу 2.6. Завершуючи викладення матеріалу цього підрозділу викладачеві МЗКО обов'язково потрібно звернути увагу студентів на те, що:

1) перетворювати за Лапласом можна лише лінійні диференціальні рівняння;

2) формули розкладання виведені шляхом застосування теореми лишків, відомої у теорії функцій комплексної змінної, до обчислення інтеграла, за яким здійснюється обернене перетворення за Лапласом;

3) перетворення за Лапласом доцільно застосовувати лише при розв'язанні лінійних диференціальних рівнянь з нульовими початковими умовами, оскільки за наявності ненульових початкових умов трудомісткість процесів розв'язання лінійного диференціального рівняння застосуванням перетворення за Лапласом стає вищою трудомісткості розв'язання цього рівняння безпосередньо в часовій області.

2.7 Авторегресійні моделі часових рядів

2.7.1 Авторегресійна модель стаціонарного часового ряду

Під час прогнозування розвитку дискретних процесів, що мають випадковий характер, широке застосування знаходять моделі часових рядів, які являють собою послідовності значень випадкового процесу z_t, взятих через однакові проміжки часу ***t***.

Як і будь-який інший випадковий процес, часовий ряд z_t може бути стаціонарним, для якого характерною є рівновага його значень z_t в околі середнього значення m_z, яке є константою, або нестаціонарним, для якого ковзне середнє значення $m_z(t)$ процесу є функцією часу t.

Введемо кілька корисних операторів, які будемо використовувати в моделях часових рядів.

Оператор B зсуву назад на одну одиницю часу

$$z_{t-1} = Bz_t\,, \tag{2.179}$$

узагальнення якого приводить до виразу

$$z_{t-m} = B^m z_t\,. \tag{2.180}$$

Різницевий оператор ∇ із зсувом назад на одну одиницю часу

$$\nabla z_t = z_t - z_{t-1}\,. \tag{2.181}$$

для якого справедливим є вираз

$$\nabla z_t = z_t - z_{t-1} = z_t - Bz_t = \left(1-B\right)z_t\,. \tag{2.182}$$

а тому

$$\nabla = 1 - B\,. \tag{2.183}$$

Оператор суми S

$$Sz_t = z_t + z_{t-1} + z_{t-2} + \ \ldots\ = \sum_{j=0}^{\infty} z_{t-j}\,. \tag{2.184}$$

для якого справедливим є вираз

$$Sz_t = \left(1 + B + B^2 + \ \ldots\right)z_t = \frac{1}{1-B}z_t = \left(1-B\right)^{-1} z_t\,. \tag{2.185}$$

або

$$S\,z_t = \nabla^{-1} z_t\,, \tag{2.186}$$

а тому

$$S = \nabla^{-1}\,. \tag{2.187}$$

оператор суми є оберненим різницевому оператору зі зсувом назад.

Далі нагадаємо, що послідовність некорельованих і розподілених нормально випадкових імпульсів a_t з нульовим середнім значенням і дисперсією

$$\sigma_a^2 = const \tag{2.188}$$

називають дискретним білим шумом.

Здійснимо операцію центрування часового ряду z_t, віднявши від кожного значення z_t середнє значення μ часового ряду, яке знаходиться з виразу

$$\mu = \frac{1}{N}\sum_{t=1}^{N} z_t\,. \tag{2.189}$$

Отже для центрованого часового ряду матимемо

$$\tilde{z}_t = z_t - \mu\,. \tag{2.190}$$

Популярна в теорії прогнозування часових рядів модель авторегресії – це модель, яка встановлює ще не виміряне значення якоїсь координати процесу у даний момент часу на основі своїх попередніх значень. Кількість врахованих попередніх значень визначає порядок авторегресії.

Для центрованого часового ряду $\tilde{z}_t$ модель авторегресії порядку p (скорочено: АР(p)) записується у вигляді

$$\tilde{z}_t = \phi_1\tilde{z}_{t-1} + \phi_2\tilde{z}_{t-2} + \ \ldots + \phi_p\tilde{z}_{t-p} + a_t\,, \tag{2.191}$$

де a_t – імпульс білого шуму.

Підставляючи вираз (2.180) у (2.191), отримаємо

$$\tilde{z}_t - \phi_1 B\tilde{z}_t - \phi_2 B^2\tilde{z}_t - \ldots - \phi_p B^p\tilde{z}_t = a_t\,,$$

або

$$\Phi(B)\tilde{z}_t = a_t\,, \tag{2.192}$$

де $\Phi(B)$ – оператор авторегресії порядку p, який має вигляд

$$\Phi(B) = 1 - \phi_1 B - \phi_2 B^2 - \ldots - \phi_p B^p\ . \tag{2.193}$$

У процесі розв'язання задачі ідентифікації моделі часового ряду z_t на основі оператора авторегресії порядку p потрібно визначити $p+2$ невідомих, якими є коефіцієнти $\phi_1, \phi_2, \ldots, \phi_p$ оператора $\Phi(B)$, середнє значення μ процесу z_t та дисперсія σ_a^2 білого шуму a_t, яка є головним параметром в програмі генерації імпульсів білого шуму для формування моделі авторегресії.

Про те, як розв'язати цю задачу, мова піде після введення понять автоковаріації та автокореляції часового ряду.

Автоковаріацією γ_k часового ряду z_t з затримкою k називають вираз

$$\gamma_k = \operatorname{cov}\{z_t, z_{t+k}\} = E\{(z_t - \mu)\cdot(z_{t+k} - \mu)\}\,, \tag{2.194}$$

в якому E – символ обчислення математичного очікування від виразу, що стоїть у фігурних дужках.

Очевидно, що

$$\gamma_0 = E\left\{(z_t - \mu)^2\right\} = \sigma_z^2 \tag{2.195}$$

– дисперсія часового ряду z_t.

Для отримання статистичної оцінки γ_k^* автоковаріації γ_k, визначеної виразом (2.194), використовують вираз

$$\gamma_k^* = \frac{1}{N-k}\sum_{t=1}^{N-k}\left(z_t-\mu\right)\cdot\left(z_{t+k}-\mu\right). \tag{2.196}$$

Автоковаріація γ_k характеризує ступінь лінійного зв'язку між значеннями часового ряду z_t та z_{t+k}. Для неї є справедливими співвідношення

$$\begin{cases}\left|\gamma_k\right| \le \gamma_0, \\ \gamma_k = \gamma_{-k}.\end{cases} \tag{2.197}$$

Автокореляцією ρ_k часового ряду z_t із затримкою k називають вираз

$$\rho_k = \frac{\gamma_k}{\gamma_0} = \frac{E\left\{\left(z_t-\mu\right)\cdot\left(z_{t+k}-\mu\right)\right\}}{E\left\{\left(z_t-\mu\right)^2\right\}}. \tag{2.198}$$

Для довільної автокореляції ρ_k справедливими є такі співвідношення:

$$\begin{cases}\left|\rho_k\right| \le \rho_0, \\ \rho_0 = 1, \\ \rho_k = \rho_{-k}.\end{cases} \tag{2.199}$$

Вся сукупність γ_k задає автоковаріаційну функцію часового ряду z_t. Вона належить до класу дискретних функцій. Аналогічно, сукупність всіх значень ρ_k задає дискретну автокореляційну функцію часового ряду z_t.

А тепер повернемось до сформульованої уже вище задачі визначення коефіцієнтів оператора авторегресії, тобто отримаємо відповідь на запитання: «А як же визначити коефіцієнти авторегресії моделі часового ряду у формі АР$\left(p\right)$?».

Відповідь на це запитання дали Юл та Уокер, вивівши свої знамениті рівняння, які мають вигляд

$$\gamma_k = \phi_1\gamma_{k-1} + \phi_2\gamma_{k-2} + \ \ldots + \phi_p\gamma_{k-p} \tag{2.200}$$

для всіх k від 1 до p та

$$\gamma_0 = \phi_1\gamma_1 + \phi_2\gamma_2 + \ \ldots + \phi_p\gamma_p + \sigma_a^2 \tag{2.201}$$

при $k = 0$.

Із виразу (2.201), маємо

$$\sigma_a^2 = \sigma_z^2 - \phi_1\gamma_1 - \phi_2\gamma_2 - \ \ldots - \phi_p\gamma_p\,. \tag{2.202}$$

Тож, якщо для реалізації стаціонарного часового ряду z_t довжиною N вже обчислені оціночні середнє значення μ і дисперсія σ_z^2 за формулами, наведеними вище, і знайдені якимсь чином коефіцієнти $\phi_1, \phi_2, \ldots, \phi_p$ та обчислені за виразом (2.196) автоковаріації $\gamma_1, \gamma_2, \ldots, \gamma_p$, то дисперсія σ_a^2 білого шуму обчислюється за виразом (2.202).

Перш ніж навести алгоритм отримання числових значень коефіцієнтів $\phi_1, \phi_2, \ldots, \phi_p$, розділимо всі рівняння системи (2.200) на γ_0.

У результаті цього отримаємо систему рівнянь вже не відносно автоковаріацій γ_k, $k = \overline{1, p}$, а відносно автокореляцій ρ_k, $k = \overline{1, p}$, тобто систему

$$\begin{cases} \phi_1\rho_0 + \phi_2\rho_1 + \phi_3\rho_2 + \ldots + \phi_p\rho_{p-1} = \rho_1, \\ \phi_1\rho_1 + \phi_2\rho_0 + \phi_3\rho_1 + \ldots + \phi_p\rho_{p-2} = \rho_2, \\ \phi_1\rho_2 + \phi_2\rho_1 + \phi_3\rho_0 + \ldots + \phi_p\rho_{p-3} = \rho_3, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \phi_1\rho_{p-1} + \phi_2\rho_{p-2} + \phi_3\rho_{p-3} + \ldots + \phi_p\rho_0 = \rho_p, \end{cases} \tag{2.203}$$

або, у матричній формі,

$$M \cdot \Phi = \rho\,, \tag{2.204}$$

де матриці $M\left(p \times p\right)$, $\Phi\left(p \times 1\right)$, $\rho\left(p \times 1\right)$ мають вигляд:

$$M = \begin{bmatrix} 1 & \rho_1 & \rho_2 & \cdots & \rho_{p-1} \\ \rho_1 & 1 & \rho_1 & \cdots & \rho_{p-2} \\ \rho_2 & \rho_1 & 1 & \cdots & \rho_{p-3} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \rho_{p-1} & \rho_{p-2} & \rho_{p-3} & \cdots & 1 \end{bmatrix}, \tag{2.205}$$

$$\Phi = \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_p \end{bmatrix}, \qquad \rho = \begin{bmatrix} \rho_1 \\ \rho_2 \\ \rho_3 \\ \vdots \\ \rho_p \end{bmatrix}. \tag{2.206}$$

У рівняннях (2.203) невідомими є значення коефіцієнтів $\phi_1, \phi_2, \ldots, \phi_p$.

Для розв'язання системи (2.203), або, що одне і те ж, системи (2.204), спочатку визначаємо матрицю M^{-1}, яка є оберненою до матриці M. Потім множимо матричне рівняння (2.204) зліва на M^{-1}. У результаті цього отримуємо

$$M^{-1}M \cdot \Phi = M^{-1}\rho, \tag{2.207}$$

або

$$I \cdot \Phi = M^{-1}\rho, \tag{2.208}$$

І, остаточно,

$$\Phi = M^{-1}\rho. \tag{2.209}$$

Саме рівняння у вигляді (2.203), (2.204) носять назву рівнянь Юла – Уокера. Їх розв'язок у вигляді (2.209) дозволяє за попередньо обрахованими автокореляціями ρ_k, $k = \overline{1, p}$ часового ряду z_t визначити вектор $\Phi(p \times 1)$ коефіцієнтів $\phi_1, \phi_2, \ldots, \phi_p$ оператора авторегресії у моделі АР(p).

В моделі часового ряду z_t на основі авторегресії порядку p у формуванні поточного значення ряду бере участь лише один поточний імпульс білого шуму a_t. Якщо ж від цього імпульсу a_t відняти зважену суму q попередніх значень білого шуму, то отримаємо модель авторегресії, яка більш адекватно віддзеркалюватиме властивості часового ряду z_t, оскільки крім авторегресії ця модель буде враховувати ще й ковзне середнє процесу.

Така модель часового ряду z_t носить назву моделі авторегресії - ковзного середнього порядку (p, q) (скорочено: модель АРКС(p, q)) і має вигляд

$$\begin{aligned} \tilde{z}_t = \phi_1\tilde{z}_{t-1} + \phi_2\tilde{z}_{t-2} + \ldots + \phi_p\tilde{z}_{t-p} + \\ a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - \ldots - \theta_q a_{t-q} \end{aligned} \tag{2.210}$$

Перенісши всі члени з $\tilde{z}_{t-i}$, $i = \overline{1, p}$ у ліву частину рівняння (2.210), отримаємо рівняння

$$\Phi(B)\tilde{z}_t = \Theta(B)a_t, \tag{2.211}$$

в якому оператор $\Phi(B)$ визначається виразом (2.193), а оператор $\Theta(B)$ має вигляд

$$\Theta((B) = 1 - \theta_1 B - \theta_2 B^2 - \cdots - \theta_q B^q. \tag{2.212}$$

Рівняння (2.210) та (2.211) є основними формами моделі часового ряду z_t на основі АРКС(p, q).

Для ідентифікації цієї моделі потрібно визначити $p+q+2$ невідомих, якими є коефіцієнти ϕ_i, $i=\overline{1, p}$ оператора $\Phi(B)$, коефіцієнти θ_j, $j=\overline{1, q}$ оператора $\Theta(B)$, середнє значення μ процесу z_t та дисперсія σ_a^2 білого шуму a_t.

2.7.2 Авторегресійна модель нестаціонарного часового ряду

Всі моделі часових рядів, що побудовані вище, базувались на умові стаціонарності цих рядів. Але у повсякденному житті постійно стикаємося і з нестаціонарними випадковими процесами. Наприклад, це процеси пуску або гальмування будь-якого технологічного обладнання, яке реалізує технологічний процес стохастичного характеру.

Покажемо, що такі нестаціонарні випадкові процеси, які у процесі їх дискретизації перетворюються на часові ряди, можна досить адекватно описувати за допомогою моделі, в якій закладені оператори авторегресії – проінтегрованого ковзного середнього.

Для їх синтезу припустимо, що у моделі АРКС(p, q), поданій виразом (2.211), оператор $\Phi(B)$ має d кратних коренів, що дорівнюють одиниці.

У цьому випадку, згідно з теоремою Вієта, оператор $\Phi(B)$ можна записати у вигляді

$$\Phi(B)=(1-B)^d \cdot \left(1-\phi_1^* B-\phi_2^* B^2- \ \ldots-\phi_l^* B^l\right), \tag{2.213}$$

де $d+l=p$. (2.214)

Позначимо

$$\Phi^*(B)=1-\phi_1^* B-\phi_2^* B^2- \ \ldots-\phi_l^* B^l. \tag{2.215}$$

З урахуванням виразів (2.213) та (2.215) рівняння (2.211) можна переписати так

$$\Phi^*(B)\cdot(1-B)^d \tilde{z}_t=\Theta(B) a_t. \tag{2.216}$$

Оскільки, згідно з виразом (2.183),

$$(1-B)^d=\nabla^d, \tag{2.217}$$

тобто, $(1-B)^d$ є різницевим оператором із зсувом назад порядку d, то рівняння

$$(1-B)^d z_t=\nabla^d z_t \tag{2.218}$$

задає нову змінну w_t, яка пов'язана з z_t співвідношенням

$$w_t = \nabla^d z_t. \tag{2.219}$$

Підставляючи вираз (2.219) у рівняння (2.216), отримуємо

$$\Phi^*(B) w_t = \Theta(B) a_t. \tag{2.220}$$

А це вже модель АРКС(l, q) відносно w_t, яку можна переписати і так

$$\begin{aligned} w_t = \phi_1^* w_{t-1} + \phi_2^* w_{t-2} + \ \ldots + \phi_l^* w_{t-l} + \\ + a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - \ \ldots - \theta_q a_{t-q}. \end{aligned} \tag{2.221}$$

Рівняння (2.219), (2.220) задають модель нестаціонарного часового ряду z_t у вигляді авторегресії – проінтегрованого ковзного середнього порядку (l, q, d). Скорочено: модель АРПКС(l, q, d).

Звертаємо увагу на те, що вже перша різниця ∇z_t значень будь-якого нестаціонарного часового ряду z_t має менший ступінь нестаціонарності, ніж сам часовий ряд z_t. Ще менший ступінь нестаціонарності матиме друга різниця $\nabla^2 z_t$, яка є різницею перших різниць $\nabla(\nabla z_t)$ цього часового ряду z_t.

Підвищуючи порядок d різниці $\nabla^d z_t$, рано чи пізно дійдемо до такого її значення w_t, яке вже являтиме собою стаціонарний часовий ряд відносно w_t, модель для якого можна подавати у вигляді (2.220).

Дамо пояснення чому у назві моделі АРПКС(l, q, d) має місце слово «проінтегроване» стосовно ковзного середнього.

Нагадаємо, що оберненим оператором для ∇, згідно з виразом (2.187) є оператор суми S. Тому, отримавши w_t з рівняння (2.220), для переходу до часового ряду z_t потрібно координату w_t підсумувати d разів, оскільки, помножуючи рівняння (2.219) зліва на ∇^{-d}, маємо

$$\nabla^{-d} w_t = \nabla^{-d} \nabla^d z_t, \tag{2.222}$$

або

$$\nabla^{-d} w_t = I \cdot z_t, \tag{2.223}$$

звідки, з урахуванням (2.187), матимемо

$$z_t = \nabla^{-d} w_t = S^d w_t. \tag{2.224}$$

Методичні рекомендації до підрозділу 2.7. Завершуючи викладення матеріалу цього підрозділу викладачеві МЗКО обов'язково потрібно звернути увагу студентів на те, що:

1) найскладнішим завданням у процесі використання моделі АРПКС (l, q, d) є визначення числового значення параметра інтегрування d, або, в інших термінах, визначення кількості різниць, які потрібно послідовно взяти від нестаціонарного часового ряду z_t, щоб перетворити його у стаціонарний ряд відносно якоїсь різниці цього ряду. У практичній площині це завдання розв'язується визначенням середнього значення чергової різниці; і ту різницю, для якої середнє значення на відрізку значень часового ряду, який ми маємо, дорівнюватиме нулю, ми і сприймаємо уже як стаціонарний випадковий процес, для якого можна синтезувати авторегресійну математичну модель;

2) при обчисленні автоковаріацій їх значення визначаються з прийнятним рівнем похибки лише за умови, що довжина часового ряду, для якого визначаються автоковаріації, перевищує в 10 разів найбільший зсув реалізації цього часового ряду при обчисленні автоковаріацій.

Розділ 3 PYTHON-ПРОГРАМИ ЯК ЗАСОБИ КОМП'ЮТЕРНИХ ОБЧИСЛЕНЬ

3.1 Обчислення з використанням ППП sympy

У цьому підрозділі, орієнтуючись на наведені в списку використаної літератури джерела і, у першу чергу, на роботу [7], дамо характеристику ***ППП sympy***, який містить в собі функції і методи створення програм мовою ***Python*** для здійснення символьних обчислень.

І першою командою, яку ми маємо виконати після виклику ***ППП sympy***, повинна бути команда ***symbols(' '),*** якою змінні та позначені літерами параметри в наших обчислювальних виразах необхідно оголосити символьними.

Із цих символьних змінних і параметрів створюються символьні вирази, які відповідними ***функціями ППП sympy***, частину яких ми наводимо нижче, можна приводити до зручного для здійснення обчислень вигляду. Отже:

- функція **sympy.S( )** ***або просто S( )*** будь-яку константу або стрічку оголошує ***символьною***;
- функція ***symbols ('x:n'***) змінній ***x*** присвоює індекси в межах від 0 до $n-1$, в результаті чого формується символьна ***послідовність {x0, x1, x2, …, x(n-1)}***;
- метод ***f.subs( )*** обчислює значення ***функції f( )*** при конкретному значенні змінної ***x***;
- функція ***factor ( )*** розкладає символьний вираз, що стоїть в аргументних дужках, на символьні множники;
- функція ***expand( ),*** здійснюючи всі проміжні операції, розкриває дужки символьного виразу, що стоїть в аргументних дужках;
- функція ***collect( )*** збирає коефіцієнти при однакових степенях незалежної змінної;
- функція ***cancel( )*** зводить суму дробово-раціональних символьних виразів до спільного знаменника та ділить чисельник і знаменник на спільний множник в разі його наявності;
- функція ***apart( )*** розкладає складний символьний вираз на прості дроби;
- функція ***sympify(str)*** трансформує ***стрічку*** у символьний ***вираз***, придатний для подальшого використання в ***ППП sympy***;
- функція ***var( ),*** реалізуючи ту ж функцію, що і ***symbols( ),*** дозволяє внесенням додатково в аргументні дужки заданих умов надавати обмеження символьним змінним;
- метод ***f.evalf( )*** залишає в числовому результаті обчислення функції ***f*** стільки знаків, скільки вказано числом, поміщеним в аргументні дужки;
- функції ***re(z), im(z), Abs(z), arg(z)*** обчислюють ***дійсну*** та ***уявну*** частини комплексного числа ***z*** і його ***модуль*** та ***аргумент***;

- функція ***conjugate (z)*** формує ***спряжене*** до ***z*** комплексне число;

- функція ***simplify( )*** завжди реалізує стосовно символьних виразів з комплексними числами ту вказівку, яка стоїть в аргументних дужках;

- функція ***Matrix (***) створює символьні матриці, на які розповсюджуються усі операції матричної алгебри.

Викладене вище у цьому підрозділі продемонстровано ***в прикладах № 37 та № 38.***

Приклад № 37

```
In [1]: import sympy                          # Виклик ППП sympy
In [2]: from sympy import *                   # Доступ в sympy до усіх функцій
In [3]: x,y,z,a,b,c=symbols('x y z a b c')    # Оголошення x,y,z,a,b,c символьними
In [4]: f=a**3*x+3*a**2*x**2+a*x**3+x**4      # Формування функції f(x) з параметром a
In [5]: f.subs(a,2)                           # Внесення в f(x) значення параметра a
Out[5]:                                       # Візуалізація f(x) після підстановки
x**4 + 2*x**3 + 12*x**2 + 8*x                 в неї значення параметра a
In [6]: f.subs(x,1)                           # Параметризація функції f(a,x)
Out[6]:                                       # Візуалізація f(a,x) після підстановки
a**3 + 3*a**2 + a + 1                           в неї значення змінної x
In [7]: f.subs([(a,2),(x,1)])                 # Обчислення функції f(a,x) після
Out[7]:                                         підстановки в неї значень a та x
23
In [8]: expr=x**2+4*x+2                       # Формування спецфункції expr
In [9]: expr.subs(x,x**2)                     # Підстановка функції замість змінної
Out[9]:
x**4 + 4*x**2 + 2
In [10]: y=symbols('y:6');y                   # Індексне формування послідовності
Out[10]: (y0, y1, y2, y3, y4, y5)
In [11]: y=symbols('y6:11');y                 # Індексне формування послідовності
Out[11]: (y6, y7, y8, y9, y10)
In [12]: str="x**2-4"                         # Математичний вираз стрічкою
In [13]: expr=sympify(str);expr               # Спецфункція expr для стрічки
Out[13]:
x**2 – 4
In [14]: factor(expr)                         # Розкладення виразу на множники
Out[14]:
(x - 2)*(x + 2)
In [15]: f1=(2*a+x)**2*(a+2*x)**2             # Формування символьної функції f1
In [16]: expand(f1)                           # Розкриття дужок у виразі f1
Out[16]:
4*a**4 + 20*a**3*x + 33*a**2*x**2 +\
         20*a*x**3 + 4*x**4
In [17]: f2=(2*a+x)**2-(a+2*x)**2             # Формування символьної функції f2
In [18]: expand(f2)                           # Розкриття у виразі f2 та його спрощення
Out[18]:
3*a**2 - 3*x**2
```

```
In [19]: f3=((2*a+x)**2-(a+2*x)**2)**2          # Формування символьної функції f3
In [20]: expand(f3)                             # Розкриття дужок у виразі f3
Out[20]:
9*a**4 - 18*a**2*x**2 + 9*x**4
In [21]: var('x,y',positive=True)               # Оголошення x, y символьними
Out[21]: (x, y)
In [22]: expand(log(x/y))                       # Розкриття дужок у виразі log
Out[22]:
log(x) - log(y)
In [23]: expr= x**2*y+x*z-2*x**2-4*x            # Формування спецфункції expr
In [24]: collect(expr,x)                        # Збирання членів при степенях x
Out[24]:
x**2*(y - 2) + x*(z - 4)
In [25}: cancel(x/a+y/b-z/c)                    # Зведення символьних дробів
Out[25]:
(-a*b*z + a*c*y + b*c*x)/(a*b*c)
In [26]: apart((4*x**3+21*x**2)/(x**4+5*x**3))  # Зведення виразу до складових
Out[26]:
-1/(5*(x + 5)) + 21/(5*x)
In [27]: M=Matrix([[a,x],[b,y]]);M              # Трансформація списків у матрицю
Out[27]:
Matrix([
[a, x],
[b, y]])
In [28]: M[0,1]                                 # Виклик із матриці елемента за його
                                                  індексами
Out[28]:
x
```

Приклад № 38, присвячений операціям з **комплексними *числами в*** ППП ***sympy***

```
In [1]: import sympy                            # Виклик ППП sympy
In [2]: from sympy import *                     # Доступ в sympy до усіх функцій
In [3]: x,y,z=symbols('x y z',real=True)        # Оголошення x,y,z символьними з умовою
In [4]: z=x+y*I                                 # Формування комплексного числа z
In [5]: re(z)                                   # Визначення дійсної частини числа z
Out[5]:
x
In [6]: im(z)                                   # Визначення уявної частини числа z
Out[6]:
y
In [7]: Abs(z)                                  # Визначення модуля числа z
Out[7]:
sqrt(x**2 + y**2)
In [8]: arg(z)                                  # Визначення аргументу числа z
```

```
Out[8]:
arg(x + I*y)
In [9]: arg(2+2*I)                    # Конкретизація аргументу числа z
Out[9]:
pi/4
In [10]: z1=conjugate(z);z1           # Комплексне число z1, спряжене з z
Out[10]:
x - I*y
In [11]: z+z1                         # Сума комплексно-спряжених чисел
Out[11]:
2*x
In [12]: z-z1                         # Різниця комплексно-спряжених чисел
Out[12]:
2*I*y
In [13]: simplify(z*z1)               # Добуток комплексно-спряжених чисел
Out[13]:
x**2 + y**2
In [14]: simplify(z/z1)               #Ділення комплексних чисел в загальному вигляді
Out[14]:
(x + I*y)/(x - I*y)
In [15]: simplify((2+4*I)/(2-4*I))    # Частка ділення конкретних комплексних чисел
Out[15]:
-3/5 + 4*I/5
In [16]: var('w')                     # Оголошення змінної w символьною
Out[16]:
w
In [17]: w=z**4;w                     # Формування функції w змінної z
Out[17]:
(x + I*y)**4
In [18]: w=expand(w);w                # Упорядкування комплексного виразу в
                                        алгебраїчній формі
Out[18]:
-119 - 120*I
```

3.1.1 Інтегрування функцій в рамках програм *ППП sympy*

Функція ***integrate( )*** здійснює інтегрування символьної функції, а при додаванні в аргументних дужках окрім символьної функції та її аргументу ще й меж області інтегрування ця функція обчислює визначений інтеграл (при цьому потрібно взяти до уваги, що в ***ППП sympy*** нескінченність позначається подвійною латинською літерою **«o»** , тобто ***«oo»)***.

Приклад № 39, присвячений операціям **інтегрування** в рамках ***ППП sympy***

```
In [1]: import sympy                  # Виклик ППП sympy
In [2]: from sympy import *           # Доступ в sympy до усіх функцій
In [3]: x,y,z=symbols('x y z')        # Оголошення x,y,z символьними
```

```
In [4]: f1=x**4+2*x**3+4*x**2+6*x+8          # Формування символьної функції f1
In [5]: f2=2*x*y*z+4*x**2*y*z+6*x*y**3*z**2  # Формування символьної функції f2
In [6]: integrate(f1,x)                      # Визначення інтеграла від f1 по x
Out[6]:
x**5/5 + x**4/2 + 4*x**3/3 + 3*x**2 + 8*x
In [7]: integrate(f2,x)                      # Однократне інтегрування f2 по x
Out[7]:
4*x**3*y*z/3 + x**2*(3*y**3*z**2 + y*z)
In [8]: integrate(f2,x,y)                    # Інтегрування f2 по x та y
Out[8]:
3*x**2*y**4*z**2/4 + y**2*(2*x**3*z/3 + x**2*z/2)
In [9]: integrate(f2,x,y,z)                  # Інтегрування f2 по x, y, z
Out[9]:
x**2*y**4*z**3/4 + z**2*(x**3*y**2/3 + x**2*y**2/4)
In [10]: integrate(f1,(x,0,1))               # Обчислення однократного
Out[10]:                                       інтеграла від функції f1
391/30                                         по x в межах від 0 до 1
In [11]: integrate(f2,(x,0,1),(y,0,2),(z,-1,1))  # Обчислення трикратного
Out[11]:                                       інтеграла від функції f2
8                                              по x,y,z в заданих межах
In [12]: integrate(log(x)**3,x)              # Однократне інтегрування
Out[12]:                                       складної логарифмічної
x*log(x)**3 - 3*x*log(x)**2 + 6*x*log(x) - 6*x  функції по x
In [13]: integrate(log(x)**3,(x,-1,1))       # Обчислення інтеграла від складної
Out[13]:                                       логарифмічної функції по x
-12 + 3*pi**2 - I*pi**3 + 6*I*pi               в межах від -1 до 1
In [14]: integrate(x**3*y**2,(y,0,x**2),(x,0,1))  # Обчислення двократного
Out[14]:                                       інтеграла у випадку змінної
1/30                                           внутрішньої межі
In [15]: integrate(exp(-2*x**2),(x,0,oo))    # Обчислення однократного
Out[15]:                                       інтеграла у випадку межі
sqrt(2)*sqrt(pi)/4                             у вигляді нескінченності
In [16]: integrate(exp(-x**2/4-y**2),(x,0,oo),\  # Обчислення двократного
   (y,-oo,oo))                                 інтеграла в нескінченних межах
Out[16]:
pi
```

3.1.2 Диференціювання функцій в рамках програм *ППП sympy*

Реалізацію програмних функцій ***диференціювання*** нам потрібно ***починати*** з ***виклику ППП sympy*** усіх функцій з цього пакета та символізації потрібних змінних.

Однократне диференціювання по одній незалежній змінній ***здійснюється*** програмною функцією ***diff(,)***, в аргументних дужках якої першим стоїть вираз, що диференціюється, а другим змінна, по якій здійснюється диференціювання;

n-кратне диференціювання здійснюється ***цією*** ж програмною ***функцією diff(, ,)***, але в аргументних дужках додається ***третім аргументом число***, яким конкретизується кількість ***диференціювань.***

В разі потреби визначення ***змішаних частинних похідних*** від виразу, що містить ***дві незалежні змінні***, ***в*** аргументних ***дужках*** цієї ж програмної ***функції diff(, ,)*** ми маємо записати окрім виразу, що диференціюється, ***на другому і третьому місцях обидві незалежні змінні***, по яких здійснюється диференціювання, а якщо потрібно визначити ***змішану частинну похідну з порядком, вищим першого***, програмна ***функція diff( )*** уже матиме ***5 аргументів***, першим із яких виступатиме вираз, що диференціюється, ***другим*** – ***перша*** незалежна ***змінна***, ***третім – число*** диференціювань ***по першій*** змінній, ***четвертим*** – ***друга*** незалежна ***змінна***, а ***п'ятим – число*** диференціювань ***по другій*** змінній.

В разі, ***якщо*** програмна функція ***diff( )***, виявляється ***неспроможною*** здійснити диференціювання через непідйомну для неї складність виразу, що диференціюється, застосовують ***метод диференціювання***, який має ту ж символіку, але перед цим символом через крапку записується символ ***функції f***, що диференціюється, тобто запис методу має вигляд ***f.diff(,),*** а в аргументних дужках уже не вписується першим аргументом вираз, що диференціюється, а вписується ***на першому місці*** незалежна ***змінна,*** по якій здійснюється диференціювання, а ***другим аргументом*** вказується ***число диференціювань***, якщо воно більше одиниці. ***А за наявності*** ще й ***другої*** незалежної ***змінної*** у функції ***f*** в аргументних дужках ***на третьому місці*** записується ця ***друга змінна***, а ***на четвертому*** місці – ***кількість диференціювань по ній***.

У разі, ***якщо і метод диференціювання*** виявляється ***неспроможним*** продиференціювати задану функцію ***f***, він видає у відповідь оператор ***Derivative,*** після чого потрібно сформувати програмну функцію ***g=Derivative(f, , , ,)*** і застосувати до неї безаргументний метод ***g.doit( ).***

Усе, викладене вище, проілюстроване в ***прикладі № 40.***

Приклад № 40, присвячений операціям **диференціювання** в рамках ***ППП sympy***

```
In [1]: import sympy                          # Виклик ППП sympy
In [2]: from sympy import *                   # Виклик із sympy усіх функцій
In [3]: x,y=symbols('x y')                    # Оголошення x, y символьними
In [4]: diff(sin(2*x)*cos(3*x))               # Диференціюваання виразу в дужках
Out[4]:                                       # Результат диференціювання
-3*sin(2*x)*sin(3*x) + 2*cos(2*x)*cos(3*x)
In [5]: diff(x**2*exp(-3*x),x,2)              # Визначення другої похідної
Out[5]:
(9*x**2 - 12*x + 2)*exp(-3*x)
In [6]: f=x**3*y**2                           # Формування функції f(x,y) двох змінних
```

```
In [7]: diff(f,x,3,y,2)                    # Визначення змішаної частинної
Out[7]:                                      похідної 3-го порядку по x та 2-го
12                                           порядку по y від функції f(x,y)
In 8]: diff(f,x,y)                         # Визначення змішаної частинної
Out[8]:                                      похідної 1-го порядку по x та y від
6*x**2*y                                     функції f(x,y)
In [9]: f.diff(x,y)                        # Визначення змішаної частинної
Out[9]:                                      похідної 1-го порядку по x та y від
6*x**2*y                                     функції f(x,y) методом f.diff( )
In [10]: f1=exp(x**2*y**3)                 # Формування функції f1 двох змінних
In [11]: diff(f1,x,y,2)                    # Визначення змішаної частинної
Out[11]:                                     похідної 1-го порядку по x та 2-го
6*x*y*(x**2*y**3*(3*x**2*y**3 + 2) +         порядку по y від функції f1
+ 6*x**2*y**3 + 2)*exp(x**2*y**3)

In [12]: f2=Derivative(f1,x,y,2)           # Підготовка f2 через Derivative від f1
In [13]: f2.doit( )                        # Визначення змішаної частинної
Out[13]:                                     похідної 1-го порядку по x та 2-го
6*x*y*(x**2*y**3*(3*x**2*y**3 + 2) +         порядку по y від функції f1
+ 6*x**2*y**3 + 2)*exp(x**2*y**3)            методом f2.doit( )
```

3.1.3 Апроксимація функцій рядами в рамках програм *ППП sympy*

В пункті 2.3.1 ми показали, як можна апроксимувати неперервні функції степеневими рядами, а в пункті 2.3.2 ми показали, як можна апроксимувати неперервні функції рядами Фур'є.

Для апроксимації неперервних функцій ***f(x)*** степеневими рядами в ***ППП sympy*** використовується програмна функція ***sympy.series(,[, ,])***, в аргументних дужках якої вказується ім'я функції, що апроксимується, та список із незалежної змінної ***x,*** координати x_0 точки, в околі якої здійснюється апроксимація, та кількості ***n*** членів степеневого ряду, що утримується. Якщо x_0 та ***n*** в аргументних дужках відсутні, то це означає, що x_0 приймає значення 0, а ***n*** приймає значення 6.

Для апроксимації неперервних функцій ***f(x)*** степеневими рядами в ***ППП sympy*** використовується також програмний метод ***sympy.f(x).series(,[,])***, в аргументних дужках якого вказується ім'я незалежної змінної ***x*** та список із координати x_0 точки, в околі якої здійснюється апроксимація, та кількості ***n*** членів степеневого ряду, що утримується. І у цьому випадку, якщо x_0 та ***n*** в аргументних дужках відсутні, то це означає, що x_0 приймає значення 0, а ***n*** приймає значення 6.

Цілком очевидно, що функції ***f(x),g(x)***, апроксимовані степеневими рядами, можна диференціювати, інтегрувати і додавати, але важливо пам'ятати, що з використанням програмної функції ***sympy.simplify( )*** їх можна перемножати та кожну з них підносити до степеня.

Усе, що викладено вище у цьому пункті, продемонстровано ***в прикладі № 41.***

Приклад № 41, присвячений апроксимації функцій степеневими рядами та операціям з ними в рамках ***ППП sympy***

```
In [1]: import sympy                          # Виклик ППП sympy
In [2]: from sympy import *                   # Виклик із sympy усіх функцій
In [3]: x=symbols('x')                        # Оголошення змінної x символьною
In [4]: f=Function ('f')(x)                   # Оголошення функції f(x) символьною
In [5]: g=Function ('g')(x)                   # Оголошення функції g(x) символьною
In [6]: f=exp(x)*sin(x)                       # Формування функції f(x)
In [7]: g=exp(-x)*cos(x)                      # Формування функції g(x)
In [8]: f1=series(f, x,1,5)                   # Апроксимація функції f(x) в околі точки
                                                x0=1 степ. рядом з 5 членами і залишком
In [9]: g1=g.series(x,1,5)                    # Апроксимація функції g(x) в околі точки
                                                x0=1 степ. рядом з 5 членами і залишком
In [10]: f2=f1.removeO( )                     # Обрізання від ряду f1 його залишку
In [11]: g2=g1.removeO()                      # Обрізання від ряду g1 його залишку
In [12]: f3=f2.expand( )                      # Розкриття дужок в f2 та упорядкування
In [13]: g3=g2.expand( )                      # Розкриття дужок в g2 та упорядкування
In [14]: f4=f3.evalf(3);f4                    # Залишення в коефіцієнтах ряду f3 лише
                                                по три значущі цифри

Out[14]:
-0.381*x**4 + 1.25*x**3 + 1.52*x - 0.108
In [15]: g4=g3.evalf(3);g4                    # Залишення в коефіцієнтах ряду g3 лише
                                                по три значущі цифри

Out[15]:
-0.0331*x**4 + 0.0956*x**3 + 0.221*x**2\
 - 1.11*x + 1.02
In [16]: f5=f4.diff(x);f5                     #  Диференціювання ряду f4
Out[16]:
-1.52*x**3 + 3.76*x**2 + 1.52
In [17]: g5=g4.diff(x,2);g5                   # Двократне диференціювання ряду g4
Out[17]:
-0.397*x**2 + 0.573*x + 0.443
In [18]: f6=integrate(f4,x);f6                # Інтегрування ряду f4
Out[18]:
-0.0762*x**5 + 0.313*x**4 + 0.762*x**2\
 - 0.108*x
In [19]: g6=integrate(g4,x);g6                # Інтегрування ряду g4
Out[19]:
-0.00662*x**5 + 0.0239*x**4 + 0.0738*x**3\
 - 0.553*x**2 + 1.02*x
In [20]: f4+g4                                # Обчислення суми обрізаних рядів f4 та g4
Out[20]:
```

```
-0.414*x**4 + 1.35*x**3 + 0.221*x**2 + \
0.419*x + 0.912
In [21]: f7=f4.series(x,0,5);f7          # Трансформація ряду f4 в окіл точки x0=0
Out[21]:
-0.381*x**4 + 1.25*x**3 + 1.52*x - 0.108
In [22]: g7=g4.series(x,0,5);g7          # Трансформація ряду g4 в окіл точки x0=0
Out[22]:
-0.0331*x**4 + 0.0956*x**3 +\
 0.221*x**2 - 1.11*x + 1.02
In [23]: f7+g7                           # Обчислення суми обрізаних рядів f4 та g4
                                           після їх зміщення в окіл точки x0=0
Out[23]:
-0.414*x**4 + 1.35*x**3 + 0.221*x**2 +
0.419*x + 0.912
In [24]: f8=f4.series(x,3,5)             # Трансформація ряду f4 в окіл точки x0=3
In [25]: f9=f8.expand(); f9              # Розкриття дужок в f8  та упорядкування
Out[25]:
-0.381*x**4 + 1.25*x**3 + 0.00391*x**2 \
+ 1.52*x - 0.0938
In [26]: g8=g4.series(x,3,5)             # Трансформація ряду g4 в окіл точки x0=3
In [27]: g9=g8.expand(); g9              # Розкриття дужок в g8 та упорядкування
Out[27]:
-0.0331*x**4 + 0.0956*x**3 +\
 0.222*x**2 - 1.11*x + 1.02
In [28]: f9+g9                           # Обчислення суми обрізаних рядів f4 та g4
                                           після їх зміщення в окіл точки x0=3
Out[28]:
-0.414*x**4 + 1.35*x**3 + 0.226*x**2 +\
 0.417*x + 0.928
In [29]: simplify(f4**2)                 # Піднесення ряду f4 до квадрата
In [30]: simplify(f4**2).expand()        # Розкриття дужок в simplify(f4**2)
Out[30]:                                   та його упорядкування
0.145*x**8 - 0.954*x**7 + 1.57*x**6 - \
1.16*x**5 + 3.9*x**4 - 0.271*x**3 + \
2.32*x**2 - 0.33*x + 0.0117
In [31]: simplify(f4*g4)                 # Перемноження обрізаних рядів f4 та g4
In [32]: f4g4=(simplify(f4*g4)).expand();f4g4  # Розкриття дужок в simplify(f4*g4)
Out[34]:                                   та його упорядкування
0.0126*x**8 - 0.0779*x**7 + 0.0352*x**6 +\
 0.648*x**5 - 1.62*x**4 + 1.6*x**3 - \
1.71*x**2 + 1.68*x - 0.111
```

А у наступному ***прикладі № 42*** ми продемонструємо, як можна поєднати апроксимацію неперервних функцій степеневими рядами з апроксимацією рядами Фур'є, скориставшись ортонормованими поліномами, базовими функціями для яких є степеневі поліноми, а коефіцієнти в яких визначаються за тими ж формулами, що і в рядах Фур'є.

І це поєднання ми продемонструємо на прикладі створення **Python-програми № 42 для** апроксимації **функції** $f_2(t) = 5t^7 + 2t^2 - 6t$ дійсної змінної t, заданої на відрізку $t \in [-1, 1]$, зваженими сумами **поліномів Лежандра** $P_n(t)$, ортонормованими **на відрізку** $[-1, 1]$, яку подамо у вигляді файлу.

Але, перш ніж створювати цю програму, нагадаємо, що ортогональними на заданому відрізку значень незалежної змінної називаються поліноми, визначені інтеграли від добутку яких дорівнюють нулю, а нормованими на заданому відрізку значень незалежної змінної називаються поліноми, визначені інтеграли від квадратів кожного з яких дорівнюють одиниці.

Приклад № 42 (Файл *Python-програми* апроксимації функцій методом поєднання степеневих рядів з рядами Фур'є) (результат – на рис. 3.1)

```
import sympy                                  # Виклик ППП sympy
from sympy import*                            # Виклик із sympy усіх функцій
t,P=symbols('t P')                            # Оголошення t,P символьними
f2t=5*t**7+2*t**2-6*t                         # Формування функції f2t
P0=1                                          # 0-ий поліном Лежандра
P1=t                                          # 1-ий поліном Лежандра
P2=(3*t**2-1)/2                               # 2-ий поліном Лежандра
P3=(5*t**3-3*t)/2                             # 3-ій поліном Лежандра
P4=(35*t**4-30*t**2+3)/8                      # 4-ий поліном Лежандра
P5=(63*t**5-70*t**3+15*t)/8                   # 5-ий поліном Лежандра
P6=(231*t**6-315*t**4+105*t**2-5)/16          # 6-ий поліном Лежандра
P7=(429*t**7-693*t**5+315*t**3-35*t)/16       # 7-ий поліном Лежандра
q=symbols('q')                                # Оголошення q символьним
q02=f2t*P0*1/2                                # 0-ий підінтегральний вираз
q12=f2t*P1*3/2                                # 1-ий підінтегральний вираз
q22=f2t*P2*5/2                                # 2-ий підінтегральний вираз
q32=f2t*P3*7/2                                # 3-ій підінтегральний вираз
q42=f2t*P4*9/2                                # 4-ий підінтегральний вираз
q52=f2t*P5*11/2                               # 5-ий підінтегральний вираз
q62=f2t*P6*13/2                               # 6-ий підінтегральний вираз
q72=f2t*P7*15/2                               # 7-ий підінтегральний вираз
h=symbols('h')                                # Оголошення h символьним
h0=integrate(q02,(t,-1,1))                    # Обчислення 0-го коефіцієнта Фур'є h0
h1=integrate(q12,(t,-1,1))                    # Обчислення 1-го коефіцієнта Фур'є h1
h2=integrate(q22,(t,-1,1))                    # Обчислення 2-го коефіцієнта Фур'є h2
h3=integrate(q32,(t,-1,1))                    # Обчислення 3-го коефіцієнта Фур'є h3
h4=integrate(q42,(t,-1,1))                    # Обчислення 4-го коефіцієнта Фур'є h4
h5=integrate(q52,(t,-1,1))                    # Обчислення 5-го коефіцієнта Фур'є h5
h6=integrate(q62,(t,-1,1))                    # Обчислення 6-го коефіцієнта Фур'є h6
h7=integrate(q72,(t,-1,1))                    # Обчислення 7-го коефіцієнта Фур'є h7
f2t12=h1*P1+h2*P2                             # Формування 1-го наближення до f2t
f2t13=f2t12+h3*P3                             # Формування 2-го наближення до f2t
```

```
f2t14=f2t13+h4*P4                       # Формування 3-го наближення до f2t
f2t15=f2t14+h5*P5                       # Формування 4-го наближення до f2t
su=symbols('su')                        # Оголошення su символьним
f2t12t=f2t-f2t12                        # Обчислення
su21=integrate(f2t12t*f2t12t,(t,-1,1))   похибки 1-го
nf2t12t=su21**0.5                        наближення
print (nf2t12t)                          та його друкування
f2t13t=f2t-f2t13                        # Обчислення
su22=integrate(f2t13t*f2t13t,(t,-1,1))   похибки 2-го
nf2t13t=su22**0.5                        наближення
print(nf2t13t)                           та його друкування
f2t14t=f2t-f2t14                        # Обчислення
su23=integrate(f2t14t*f2t14t,(t,-1,1))   похибки 3-го
nf2t14t=su23**0.5                        наближення
print(nf2t14t)                           та його друкування
f2t15t=f2t-f2t15                        # Обчислення
su24=integrate(f2t15t*f2t15t,(t,-1,1))   похибки 4-го
nf2t15t=su24**0.5                        наближення
print(nf2t15t)                           та його друкування

import numpy as np                      # Виклик ППП numpy як np
import matplotlib as mpl                # Виклик ППП matplotlib як mpl
import matplotlib.pyplot as plt         # Виклик matplotlib.pyplot як plt
mpl.rcParams['font.family']='fantasy'   # Формування
mpl.rcParams['font.fantasy']='Arial',\     бібліотеки
        'Times New Roman','Tahoma'         шрифтів
x=symbols('x')                          # Оголошення x символьним
expr1= f2t.subs(t,x)                    # Функція для транспортування f2t
expr2= f2t12.subs(t,x)                  # Функція для транспортування f2t12
expr3= f2t13.subs(t,x)                  # Функція для транспортування f2t13
expr4= f2t14.subs(t,x)                  # Функція для транспортування f2t14
expr5= f2t15.subs(t,x)                  # Функція для транспортування f2t15
y1=lambdify(x,expr1, 'numpy')           # Транспортування в numpy expr1
y2=lambdify(x,expr2, 'numpy')           # Транспортування в numpy expr2
y3=lambdify(x,expr3, 'numpy')           # Транспортування в numpy expr3
y4=lambdify(x,expr4, 'numpy')           # Транспортування в numpy expr4
y5=lambdify(x,expr5, 'numpy')           # Транспортування в numpy expr5
x=np.linspace(-1,1,100)                 # Внесення масиву значень змінної x

y1vec=np.vectorize(y1)                  # Векторизація 1-ї перенесеної функції
f=y1vec(x)

y2vec=np.vectorize(y2)                  # Векторизація 2-ї перенесеної функції
f1=y2vec(x)

y3vec=np.vectorize(y3)                  # Векторизація 3-ї перенесеної функції
f2=y3vec(x)
```

```
y4vec=np.vectorize(y4)                  # Векторизація 4-ї перенесеної функції
f3=y4vec(x)

y5vec=np.vectorize(y5)                  # Векторизація 5-ї перенесеної функції
f4=y5vec(x)

fig=plt.figure(facecolor='white')       # Створення поля рисунка
plt.plot(x,f,'-k',x,f1,'-.g',x,f2,':c',x,f3,'--r',   # Побудова графіків функції і її наближень
        x, f4,'-b', linewidth=3)

plt.legend(fontsize=18)                 # Команда на створення надписів
ax=fig.gca()                            # Прив'язка до поля рисунка
plt.title(r'Апроксимація  функції       # Заголовок
        поліномами  Лежандра')
plt.text(0.45,.95,r'Графіки наближень
        до функції" ,                   # Текст на рисунку за координатами
        horizontalalignment='center',
      verticalalignment='center',
      transform=ax.transAxes,fontsize=16)
plt.xlabel(u'x-вісь абсцис',{'fontname':  # Текст на рисунку під віссю абсцис
        'Times New Roman'})
plt.ylabel(r'$f(x)$-ордината')          # Текст на рисунку на осі орлинат
plt.grid(True)                          # Нанесення координатної сітки
```

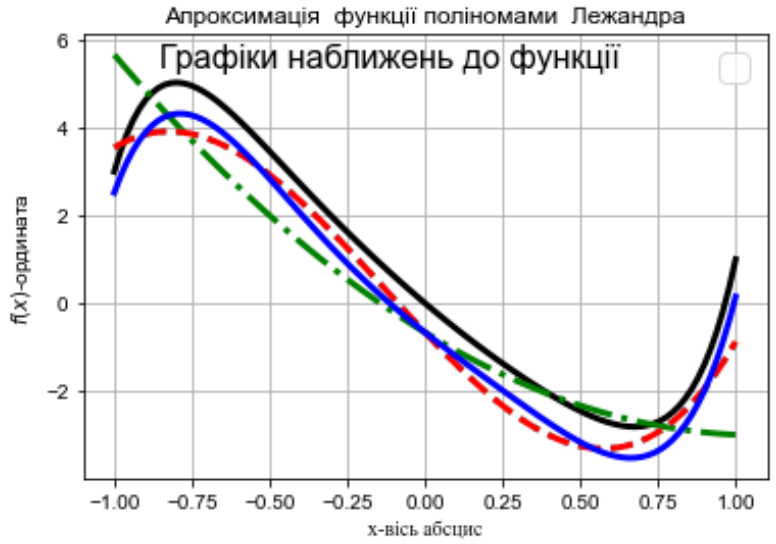


Рисунок 3.1 – Графік функції (чорний колір) та її 1-го (зелений колір), 2-го та 3-го (червоний колір) і 4-го (синій колір) наближень

Похибки наближень
1.53960071783900
1.04153121249699
1.04153121249699
0.945264797229951

3.1.4 Розв'язання алгебраїчних рівнянь програмами *ППП sympy*

Оскільки в процесі проведення досліджень часто виникають ситуації, коли ***потрібно розв'язувати алгебраїчні рівняння***, то ***потрібно знати*** як створювати ***програми мовою Python для розв'язання цих рівнянь.***

В програмному ***пакеті sympy є кілька програмних функцій, які дозволяють визначати корені алгебраїчних рівнянь***, записаних в символьній формі – ***це***, по-перше, програмна ***функція solveset ( ),*** яка розв'язує алгебраїчні рівняння, записані в символьній формі з використанням функції ***Eq(expr1,expr2),*** в якій функція ***expr1*** віддзеркалює ту частину рівняння, яка містить незалежну змінну, а функція ***expr2*** містить ту частину рівняння, яка не містить незалежної змінної і може бути зокрема і нулем. Записується функція ***Eq( )*** першою в аргументних дужках функції ***solveset ( ),*** а другою в цих дужках записується змінна, стосовно якої складається алгебраїчне рівняння і корені по якій з цього рівняння визначаються та записуються як результат у наступному рядку у вигляді множини чисел.

Доцільно зауважити, що замість функції ***Eq(expr1,expr2))*** першою в аргументних дужках функції ***solveset ( )*** може застосовуватись і функція ***expr1-expr2.***

Другою програмною функцією, яка дозволяє визначати корені алгебраїчного рівняння, записаного в символьній формі, ***в*** програмному ***пакеті sympy є функція roots( ),*** першим в аргументних дужках якої стоїть вираз, що прирівнюється нулю (без запису нуля), а другим в цих дужках записується змінна, стосовно якої складається алгебраїчне рівняння і корені по якій з цього рівняння визначаються та записуються як результат у наступному рядку у вигляді словника з зазначенням коренів та їх кратності.

Ця програмна функція має модифікацію у вигляді real_roots( ), яка визначає лише дійсні корені виразу, розміщеного в аргументних дужках, які записуються у вигляді списку.

А третьою програмною функцією, яка дозволяє визначати корені алгебраїчного рівняння, записаного ***в символьній формі, в програмному пакеті sympy є функція solve( )***, яка відрізняється від перших двох, по-перше тим, що може розв'язувати не лише одне алгебраїчне рівняння, складене відносно однієї змінної, а і системи алгебраїчних рівнянь, а по-друге, дозволяє знаходити корені цих рівнянь не лише у вигляді списку чисел, а й у вигляді аналітичних виразів, якщо вони можуть бути отримані в принципі. І першим в аргументних дужках цієї програмної функції стоїть вираз, що прирівнюється нулю або список виразів, що задають систему рівнянь, а другим в цих дужках записується або змінна, стосовно якої складається алгебраїчне рівняння і корені по якій з цього рівняння визначаються та записуються як результат у наступному рядку, або записуються ті змінні, стосовно яких складається система алгебраїчних рівнянь, корені по яких із цієї системи рівнянь визначаються та записуються як результат у наступному рядку.

Ця програмна функція має модифікацію у вигляді linsolve( ), яка визначає лише корені системи лінійних рівнянь, розміщеної в аргументних дужках у вигляді списку як першого аргументу, слідом за яким записуються змінні, стосовно яких складені ці лінійні рівняння.

Усе, що висловлене вище стосовно розв'язання алгебраїчних рівнянь, проілюстроване в ***прикладі № 43.***

Приклад № 43, присвячений розв'язанню алгебраїчних рівнянь в рамках ***ППП sympy***

```
In [1]: import sympy                          # Виклик ППП sympy
In [2]: from sympy import *                   # Виклик із sympy усіх функцій
In [3]: x,y=symbols('x y')                    # Оголошення x,y символьними
In[4]: expr1=x**3                             # Формування лівої частини рівняння
In [5]: expr2=27                              # Формування правої частини рівняння
In [6]: solveset(Eq(expr1,expr2),x)           # Розв'язання рівняння з використанням
Out[6]:                                         функції Eq( , )
FiniteSet(3, -3/2 - 3*sqrt(3)*I/2, -3/2 +
+ 3*sqrt(3)*I/2)
In [7]: solveset((expr1-expr2),x)             # Розв'язання рівняння з використанням
Out[7]:                                         функції expr1-expr2
FiniteSet(3, -3/2 - 3*sqrt(3)*I/2, -3/2 +
+ 3*sqrt(3)*I/2)
In [8]: expr=x**4+3*x**3-3*x**2-7*x+6         # Формування функції expr
In [9]: roots(expr)                           # Визначення коренів рівняння з
Out[9]:                                         функцією expr у лівій частині
 {-2: 1, -3: 1, 1: 2}                           у вигляді словника
In [10]: real_roots(x**3-27,x)                # Визначення дійсних коренів
Out[10]:                                        рівняння x^3 − 27 = 0
[3]
In [11]: solve([x**2-y-5,x+2*y**2-4],x,y)     # Розв'язання системи рівнянь,
Out[174]:                                       заданих списком
[(2, -1)]
In [12]: linsolve([2*x-3*y,3*x+y-11],x,y)     # Розв'язання системи лінійних
Out[12]:                                        рівнянь, заданих списком
FiniteSet((3, 2))
```

3.1.5 Розв'язання диференціальних рівнянь в рамках програм *ППП sympy*

Оскільки в процесі проведення досліджень часто виникають ситуації, коли ***потрібно розв'язувати диференціальні рівняння***, то ***потрібно знати*** як створювати ***програми мовою Python для розв'язання цих рівнянь.***

В ***ППП sympy є програмна функція, яка дозволяє розв'язувати диференціальні рівняння та їх системи***, записані в символьній формі – ***це***

функція ***dsolve( )***, яка використовується для символьного розв`язання звичайних диференціальних рівнянь.

Звертаємо увагу на те, що ***при символьному розв'язанні*** диференціальних рівнянь програмною функцією пакета ***sympy*** оголошувати ***символьними*** потрібно не лише імена незалежних змінних, ***а й імена функцій*** від цих змінних.

При використанні програмної функції ***dsolve(,)*** в її аргументних дужках першим аргументом потрібно вписувати програмну функцію ***Eq( )***, якою задається диференціальне рівняння, що розв'язується, а другим аргументом потрібно вписувати ім'я функції, яку ми отримаємо в результаті розв'язання цього диференціального рівняння.

Якщо в аргументні дужки програмної функції ***dsolve(,)*** окремою опцією вписати функцію ***hint,*** то нею можна викликати або розв'язки диференціального рівняння усіма доступними методами, яких в інструкції ***classify_ode( )*** є аж 8, присвоївши їй значення '***all***', або найкращий розв'язок, присвоївши їй значення '***best***.

В першому розділі ми розповіли про те, як будувати графіки на площині і в просторі, використовуючи ***ППП matplotlib*** в поєднанні з ***ППП numpy***. Але графіки можна будувати, і не покидаючи ***ППП sympy***, а скориставшись його модулем ***sympy.plotting*** та ***імпортуючи*** з нього функцію ***plot( )*** для побудови графіка функції, заданої у звичайний спосіб, та ***імпортуючи*** з нього функцію ***plot_parametric( )*** для побудови графіка функції, заданої параметричними рівняннями.

Усе, що висловлене вище стосовно розв'язання диференціальних рівнянь з використанням функції ***dsolve(,)***, проілюстроване в ***прикладах № 44 та № 45.***

Приклад № 44, присвячений розв'язанню диференціального рівняння в рамках ***ППП sympy*** (результат – на рис. 3.2)

```
In [1]: import sympy                              # Виклик ППП sympy
In [2]: from sympy import *                       # Виклик із sympy усіх функцій
In [3]: from IPython.display import *             # Виклик із IPython.display усіх функцій
In [4]: init_printing(use_latex=True)             # Запуск режиму «красивого» друку
In [5]: t=symbols('t')                            # Оголошення символьною змінної t
In [6]: u=Function('u')(t)                        # Оголошення символьною функції u(t)
In [7]: de=Eq(u.diff(t,2)+3*u.diff(t)+2*u,2*t)    # Запис ДР у формі Eq(ДР)
In [8]: display(de)                               # «Красива» візуалізація ДР
```

$$2u(t) + 3\frac{d}{dt}u(t) + \frac{d^2}{dt^2}u(t) = 2t$$

```
In [9]: des=dsolve(de,u)                          # Розв'язання ДР
In [10]: display(des)                             # «Красива» візуалізація розв'язку ДР
```

$$u(t) = C_1 e^{-2t} + C_2 e^{-t} + t - \frac{3}{2}$$

```
In [11]: var('C1 C2')                             # Оголошення символьними C1,C2
Out[11]:
```

(C1, C2)

In [12]: eq1=des.rhs.subs(t,0);eq1 # ***Формування «красивої»*** правої частини функції ***u(t)*** у формі ***u(0)***

Out[12]:

$$C_1 + C_2 - \frac{3}{2}$$

In [13]: eq2=des.rhs.diff(t).subs(t,0);eq2 # ***Формування «красивої»*** правої частини похідної $u'(t)$ у формі $u'(0)$

Out[13]:

$$-2C_1 - C_2 + 1$$

In [14]: seq=solve([eq1,eq2-1],C1,C2);seq # ***Розв'язання*** системи рівнянь ***eq1,eq2*** для початкових умов $u(0) = 0,\ u'(0) = 1$

Out[14]:

$$\left\{C_1 : -\frac{3}{2}, C_2 : 3\right\}$$

In [15]: rez=des.rhs.subs([(C1,seq[C1]), (C2,seq[C2])]) # ***Підстановка*** значень констант $C1, C2$ із отриманого словника ***в розв'язок des***

In [16]: display(rez) # ***«Красивий» друк розв'язку***

$$t - \frac{3}{2} + 3e^{-t} - \frac{3}{2}e^{-2t}$$

In [17]: from sympy.plotting import plot # ***Виклик*** із ***sympy.plotting*** функції ***plot***

In [18]: plot(rez,(t,0,1)) # ***Побудова*** графіка ***розв'язку ДР***

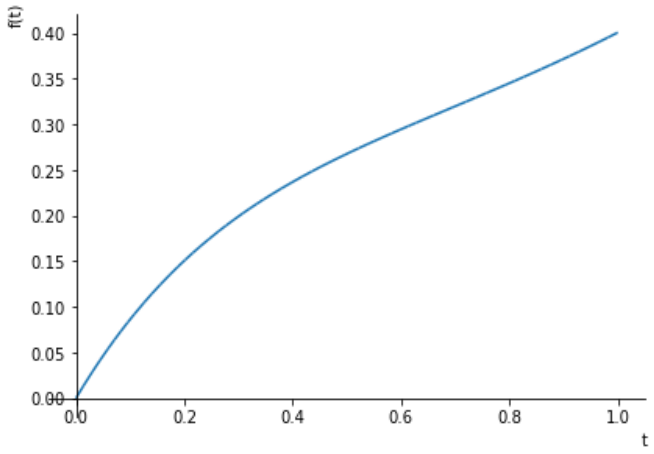


Рисунок 3.2 – Графік розв'язку диференціального рівняння (приклад № 44)

Приклад № 45, присвячений розв'язанню системи диференціальних рівнянь в рамках ***ППП sympy*** (результат – на рис. 3.3)

In [1]: import sympy # ***Виклик*** ППП ***sympy***

In [2]: from sympy import* # ***Виклик*** із ***sympy*** усіх функцій

In [3]: from IPython.display import* # ***Виклик*** з ***IPython.display*** всіх функцій

In [4]: init_printing(use_latex=True) # ***Запуск*** режиму ***«красивого»*** друку

In [5]: t=symbols('t') # ***Оголошення символьною*** змінної ***t***

In [6}: x,y,z=symbols('x y z', cls = Function) # ***Оголошення символьними*** функцій

In [7]: eq1=Eq(x(t).diff(t),-x(t)+y(t)+z(t)) # ***Формування 1-го ДР*** системи

In [8]: eq2=Eq(y(t).diff(t),-x(t)+y(t)-z(t)) # ***Формування 2-го ДР*** системи

In [9]: eq3=Eq(z(t).diff(t), x(t)-y(t)+z(t)) # ***Формування 3-го ДР*** системи

In [10]: rez=dsolve((eq1,eq2,eq3)) # ***Розв'язання*** системи ***ДР***

In [11]: display(rez[0]) # ***«Красивий» друк*** складової ***x(t)***

$$x(t) = -3C_1e^{-t} + C_2$$

In [12]: display(rez[1]) # ***«Красивий» друк*** складової ***y(t)***

$$y(t) = -C_1e^{-t} + C_2 - C_3e^{2t}$$

In [13]: display(rez[2]) # ***«Красивий» друк*** складової ***z(t)***

$$z(t) = C_1e^{-t} + C_3e^{2t}$$

In [14]: eq4=rez[0].rhs.subs(t,0);eq4 # ***Формування «красивої»*** правої частини функції ***x(t)*** у формі ***x(0)***

Out[14]:

$$-3C_1 + C_2$$

In [15]: eq5=rez[1].rhs.subs(t,0);eq5 # ***Формування «красивої»*** правої частини функції ***y(t)*** у формі ***y(0)***

Out[15]:

$$-C_1 + C_2 - C_3$$

In [16]: eq6=rez[2].rhs.subs(t,0);eq6 # ***Формування «красивої»*** правої частини функції ***z(t)*** у формі ***z(0)***

Out[16]:

$$C_1 + C_3$$

In [17]: var('C1 C2 C3') # ***Оголошення*** символьними ***C1,C2,C3***

Out[17]:

$$(C1, C2, C3)$$

In [18]: seq1=solve([eq4+1,eq5-1,eq6], C1,C2,C3);seq # ***Розв'язання*** системи рівнянь ***eq4,eq5,eq6*** для початкових умов: ***x(0)=-1, y(0)=1, z(0)=0***

Out[18]:

$$\left\{C1: \frac{2}{3}, C2: 1, C3: \frac{1}{3}\right\}$$

In [19]: rez1=rez[0].rhs.subs([(C1,seq1[C1]), (C2,seq1[C2])]);rez1 # ***Підстановка*** значень констант $C1, C2$ із словника ***в розв'язок rez[0]*** і його ***«красивий» друк***

Out[19]:

$$1 - 2e^{-t}$$

In [20]: rez2=rez[1].rhs.subs([(C1,seq1[C1]), (C2,seq1[C2]),(C3,seq1[C3])]); rez2 # ***Підстановка*** значень констант $C1, C2, C3$ в розв'язок $\boldsymbol{rez}[\mathbf{1}]$ і його ***«красивий» друк***

Out[20]:

$$1 - \frac{1}{3}e^{2t} - \frac{2}{3}e^{-t}$$

In [21]: rez3=rez[2].rhs.subs([(C1,seq1[C1]), (C3,seq1[C3])]);rez3 # ***Підстановка*** значень констант $C1, C3$ із словника ***в розв'язок rez[2]*** і його ***«красивий» друк***

Out[21]:

$$\frac{1}{3}e^{2t} + \frac{2}{3}e^{-t}$$

In [22]: from sympy.plotting import plot # ***Виклик*** із ***sympy.plotting*** функції ***plot***

In [23]: gr1=plot(rez1,(t,0,1),show= False, line_color='k') # ***Побудова*** графіка ***розв'язку x(t)*** без виведення на екран

In [24]: gr2=plot(rez2,(t,0,1),show= False, line_color='g') # ***Побудова*** графіка ***розв'язку y(t)*** без виведення на екран

```
In [25]: gr3=plot(rez3,(t,0,1),show=
         False, line_color='r')
In [26]: gr1.extend(gr2)
In [27]: gr1.extend(gr3)
In [28]: gr1.show()
```

Побудова графіка ***розв'язку z(t)*** без виведення на екран
Приєднання графіка ***y(t)*** до ***x(t)***
Приєднання графіка ***z(t)*** до ***x(t)***
Побудова графіків ***x(t), y(t), z(t*** на одному рисунку

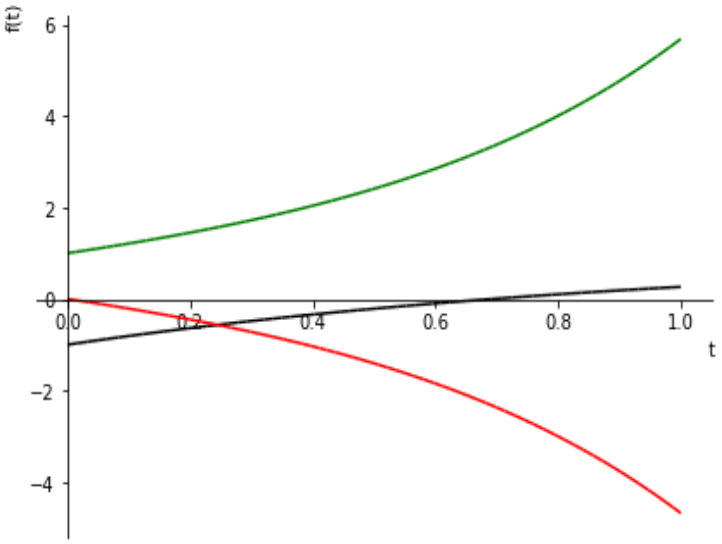


Рисунок 3.3 – Графіки розв'язків системи диференціальних рівнянь для прикладу № 45

3.1.6 Побудова графіків і поверхонь засобами ППП sympy

Як ми уже відзначили у попередньому пункті, будувати графіки на площині і в просторі можна, не лише використовуючи ***ППП matplotlib*** в поєднанні з ***ППП numpy***, а їх можна будувати, і не покидаючи ***ППП sympy***, скориставшись його модулем ***sympy.plotting*** та ***імпортуючи*** з нього функцію ***plot( )*** для побудови графіка функції, заданої у звичайний спосіб, та ***імпортуючи*** з нього різні варіанти функції ***plot_parametric( )*** для побудови графіків функцій ***y(x)***, заданих параметричними рівняннями ***x(t),y(t)***.

В прикладах № 44 та № 45 ми уже продемонстрували, які графіки будує функція ***plot( )*** з модуля ***sympy.plotting.***

Тож в наступних кількох прикладах ми продемонструємо додатково, які лінії і поверхні будують такі функції з модуля ***sympy.plotting,*** як:

- функція ***plot_implicit( )*** з опціями ***f(x,y)<1,g(x,y)<1;***
- функція ***plot_parametric( )*** з опціями ***x(t),y(t),*** (t, t_{min}, t_{max});
- функція ***plot3d( )*** з опціями ***f(x,y),g(x,y),*** (x, x_{min}, x_{max}), (y, y_{min}, y_{max});
- функція ***plot3d_parametric_line( )*** з опціями ***f(t),g(t),*** (t, t_{min}, t_{max});
- функція ***plot3d_parametric_surface( )*** з опціями ***f(x,y),g(x,y),*** (x, x_{min}, x_{max}), (y, y_{min}, y_{max}).

У тій же послідовності ми розглянемо приклади графіків, що реалізується наведеними вище функціями. Отже:

Приклад № 46, присвячений графічній реалізації булевої операції ***And*** в рамках ***ППП sympy*** (результат – на рис. 3.4)

```
In [1]: import sympy                          # Виклик ППП sympy
In [2]: from sympy import*                    # Виклик із sympy усіх функцій
In [3]: from sympy.plotting import\           # Виклик функції plot_implicit()
          plot_implicit
In [4]: x,y=symbols('x y')                    # Оголошення символьними змінних x,y
In [5]: f=Function('f')(x,y)                  # Оголошення символьною функції f
In [6]: g=Function('g')(x,y)                  # Оголошення символьною функції g
In [7]: f=x**2+y**2                           # Формування функції f
In [8]: g=(x-0.5)**2+(y-0.5)**2               # Формування функції g
In [9]: plot_implicit(And(f<1,g<1),           # Графічна реалізація булевої операції And
          (x,-1,1),(y,-1,1))
```

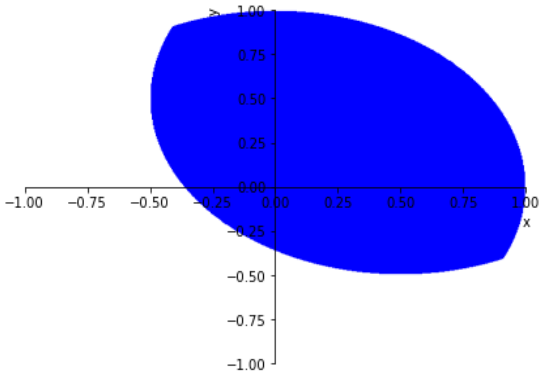


Рисунок 3.4 – Графічна реалізація булевої операції ***And***

Приклад № 47, присвячений графічній реалізації булевої операції ***Or*** в рамках ***ППП sympy*** (результат – на рис. 3.5)

```
In [1]: import sympy                          # Виклик ППП sympy
In [2]: from sympy import*                    # Виклик із sympy усіх функцій
In [3]: from sympy.plotting import\           # Виклик функції plot_implicit()
          plot_implicit
In [4]: x,y=symbols('x y')                    # Оголошення символьними змінних x,y
In [5]: f=Function('f')(x,y)                  # Оголошення символьною функції f
In [6]: g=Function('g')(x,y)                  # Оголошення символьною функції g
In [7]: f=x**2+y**2                           # Формування функції f
In [8]: g=(x-0.5)**2+(y-0.5)**2               # Формування функції g
In [9]: plot_implicit(Or(f<1,g<1),            # Графічна реалізація булевої операції Or
          (x,-1,1),(y,-1,1))
```

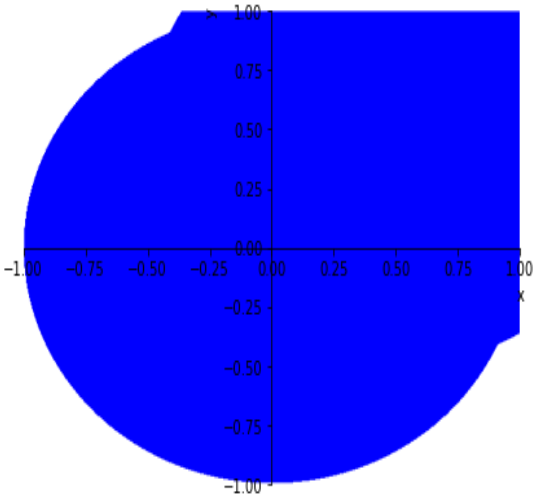


Рисунок 3.5 – Графічна реалізація булевої операції ***Or***

Приклад № 48, присвячений графічній реалізації булевої операції ***And(f<1,Not(g<1***) в рамках ***ППП sympy*** (результат – на рис. 3.6)

```
In [1]: import sympy                          # Виклик ППП sympy
In [2]: from sympy import*                    # Виклик із sympy усіх функцій
In [3]: from sympy.plotting import\           # Виклик функції plot_implicit()
            plot_implicit
In [4]: x,y=symbols('x y')                    # Оголошення символьними змінних x,y
In [5]: f=Function('f')(x,y)                  # Оголошення символьною функції f
In [6]: g=Function('g')(x,y)                  # Оголошення символьною функції g
In [7]: f=x**2+y**2                           # Формування функції f
In [8]: g=(x-0.5)**2+(y-0.5)**2               # Формування функції g
In [9]: plot_implicit(And(f<1,Not(g<1)),      # Графічна реалізація булевої операції
(x,-1,1),(y,-1,1))                              And(f<1,Not(g<1)
```

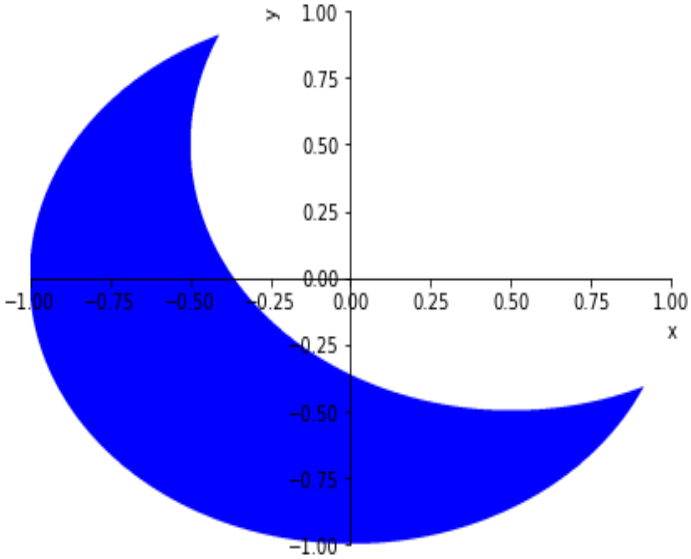


Рисунок 3.6 – Графічна реалізація булевої операції ***And(f<1,Not(g<1)***

Приклад № 49, присвячений побудові графіка лінії на площині функцією ***plot_parametric( )*** в рамках ***ППП sympy*** (результат – на рис. 3.7)

```
In [1]: import sympy                          # Виклик ППП sympy
In [2]: from sympy import*                    # Виклик із sympy усіх функцій
In [3]: from sympy.plotting import\           # Виклик функції plot_parametric()
        plot_parametric
In [4]: t=symbols('t')                        # Оголошення символьною змінної t
In [5]: plot_parametric(t*sin(t),(t-1)*       # Побудова графіка лінії на площині
        cos(t),(t,0,20),line_color='g')         функцією plot_parametric()
```

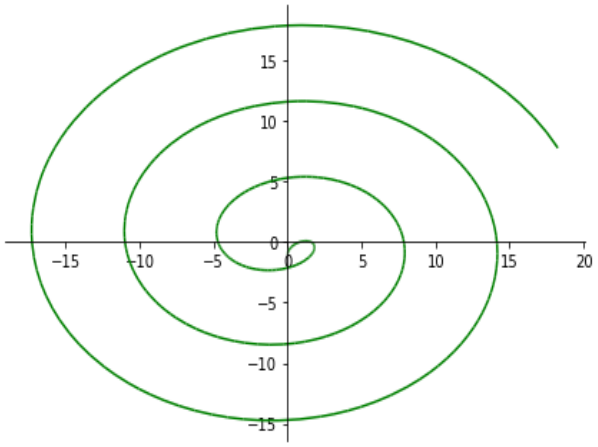


Рисунок 3.7 – Графік лінії, побудований на площині функцією ***plot_parametric( )***

Приклад № 50, присвячений побудові поверхні над площиною функцією ***plot3d( )*** в рамках ***ППП sympy*** (результат – на рис. 3.8)

```
In [1]: import sympy                          # Виклик ППП sympy
In [2]: from sympy import*                    # Виклик із sympy усіх функцій
In [3]: from sympy.plotting import\           # Виклик функції plot3d()
          plot3d
In [4]: x,y=symbols('x y')                    # Оголошення символьними змінних x,y
In [5]: f=Function('f ')(x,y)                 # Оголошення символьною функції f
In [6]: f=exp(-Abs(x+y))                      # Формування функції f
In [7]: plot3d((f),(x,-2,2),(y,-2,2))         # Побудова поверхні над площиною
                                                функцією plot3d()
```

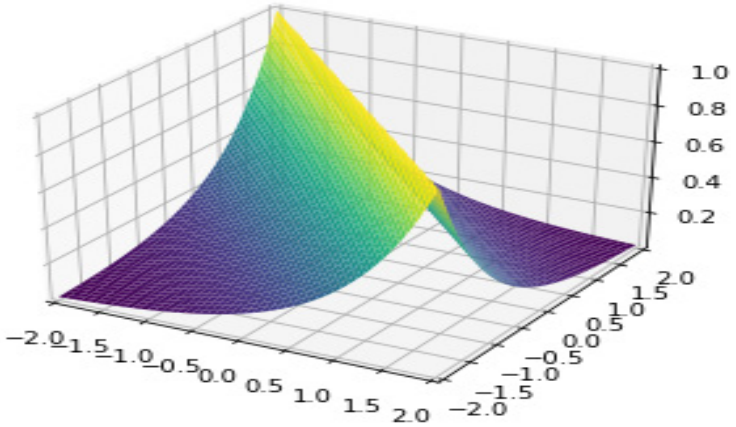


Рисунок 3.8 – Графік поверхні, побудованої над площиною функцією ***plot3d( )***

Приклад № 51, присвячений побудові лінії в тривимірному просторі функцією ***plot3d_parametric_line( )*** в рамках ***ППП sympy*** (результат – на рис. 3.9)

```
In [1]: import sympy                          # Виклик ППП sympy
In [2]: from sympy import*                    # Виклик із sympy усіх функцій
In [3]: from sympy.plotting import\           # Виклик функції
        plot3d_parametric_line                  plot3d_parametric_line()
In [4]: t=symbols('t')                        # Оголошення символьною змінної t
In [5]: plot3d_parametric_line(cos(t),        # Побудова графіка лінії у просторі функцією
        sin(t),t,(t,0,30))                      plot3d_parametric_line()
```

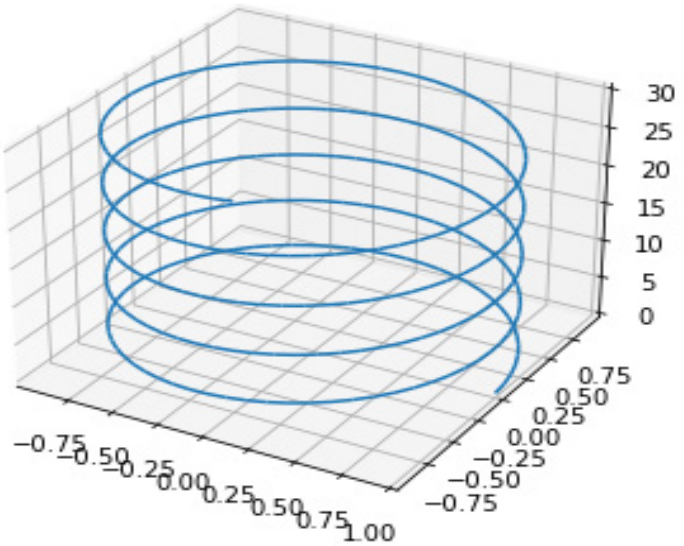


Рисунок 3.9 – Графік лінії, побудованої у просторі функцією ***plot3d_parametric_line( )***

Приклад № 52, присвячений побудові поверхні в тривимірному просторі функцією ***plot3d_parametric_surface( )*** в рамках ***ППП sympy*** (результат – на рис. 3.10)

```
In [1]: import sympy                          # Виклик ППП sympy
In [2]: from sympy import*                    # Виклик із sympy усіх функцій
In [3]: from sympy.plotting import\           # Виклик функції
        plot3d_parametric_surface               plot3d_parametric_surface()
In [4]: x,y=symbols('x y')                    # Оголошення символьними змінних x,y
In [5]: plot3d_parametric_surface(cos(x)*\    # Побудова поверхні у просторі функцією
        y**2,sin(x)*y**2,x,\                    plot3d_parametric_surface()
        (x,0,6.28),(y,-1,1))
```

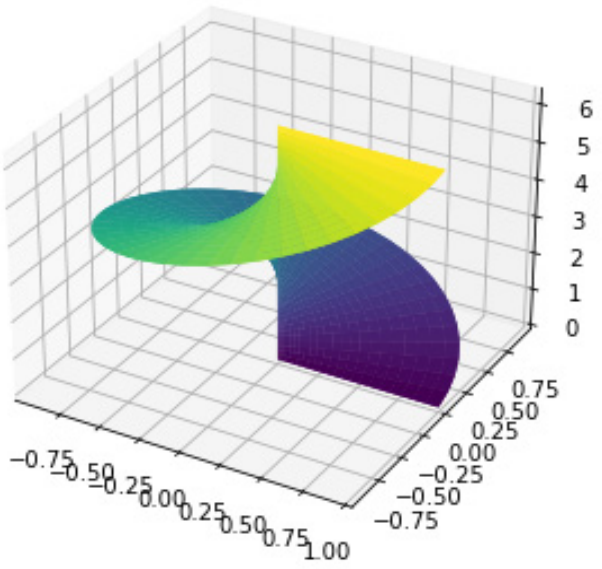


Рисунок 3.10 – Графік поверхні, побудованої у просторі функцією ***plot3d_parametric_surface( )***

Методичні рекомендації до підрозділу 3.1. Завершуючи викладення матеріалу цього підрозділу викладачеві МЗКО обов'язково потрібно ще раз звернути увагу студентів на те, що:

1) починати програмувати з використанням програмних функцій ***ППП sympy*** потрібно з оголошення символьними незалежних змінних математичних функцій;

2) оголошувати символьними атрибути в ***ППП sympy*** можна як програмною функцією symbols( ), так і програмною функцією var( );

3) оголошувати математичну функцію символьною в ***ППП sympy***, використовуючи програмну функцію Function( ), необхідно лише у випадку, коли ця математична функція в подальших розрахунках використовується як атрибут;

4) оголошуючи програмною функцією Function( ) математичну функцію символьною, потрібно в додаткових дужках з правого боку від програмної функції вказувати математичною функцією яких змінних вона є;

5) бажаючи зменшити кількість знаків в результаті обчислень значення функції ***f*** в ***ППП sympy,*** потрібно використовувати програмний метод ***f.evalf( )***, в аргументних дужках якого вказується кількість знаків, яку ми хочемо зберегти, який не тотожний програмному методу ***f.round( )*** в ***ППП numpy,*** в аргументних дужках якого вказується кількість знаків дробу після коми;

6) бажаючи транспортувати функцію ***expr***, що залежить від змінної ***x,*** з ***ППП sympy*** в ***ППП numpy***, потрібно застосовувати програмну функцію ***sympy.lambdify(x,expr,"numpy")***.

3.2 Обчислення з використанням ППП scipy

3.2.1 Розв'язання диференціальних рівнянь в рамках програм ППП scipy

За потреби розв'язувати диференціальні рівняння цілком природним є бажання спочатку спробувати їх розв'язати, використовуючи програмну функцію ***dsolve( )*** з ***ППП sympy***, оскільки вона виводить результат розв'язання диференціальних рівнянь у вигляді аналітичних виразів, Але, оскільки ця програмна функція спроможна розв'язати не кожне диференціальне рівняння, то у значній кількості практичних задач, в яких потрібно розв'язувати диференціальні рівняння, доцільніше застосовувати для їх розв'язання програмні функції ***ode( )*** та ***odeint( )***, які входять до структури модуля ***scipy.integrate*** пакета ***scipy***, за допомогою яких розв'язок отримується в числовій формі,

Але, оскільки при чисельних розрахунках ***потрібно задавати масиви*** значень незалежної змінної, ***які є об'єктами*** пакета ***numpy***, то, працюючи з пакетом ***scipy***, в обов'язковому порядку доведеться викликати і пакет ***numpy.***

А в силу того, що ***розв'язки*** диференціальних рівнянь ***характеризують розвиток у часі процесів***, які ними описуються, а також в силу того, що розв'язки, які отримані чисельними методами, не мають аналітичного опису, а являють собою масиви чисел, виникає потреба графічного відображення цих розв'язків, для реалізації якого потрібно викликати ще й пакет ***matplotlib.***

Отже, як уже було підкреслено вище, в програмному ***пакеті scipy є дві програмні функції***, які розв'язують диференціальні рівняння та системи диференціальних рівнянь, – це програмні функції ***ode( )*** та ***odeint( )***, які входять до структури модуля ***scipy.integrate,*** за допомогою яких розв'язок отримується в числовій формі.

У разі їх застосування, перш за все, потрібно звернути увагу на те, що програмна функція ***odeint(, ,)*** розв'язує ***диференціальне рівняння n-го порядку***, що має вигляд $y^{(n)} = f\left(t, y, y', y'', \dots, y^{(n-1)}\right)$ з початковими умовами $y(t_0) = y_0;\ y'(t_0) = y_0';\ y''(t_0) = y_0'';\ \dots;\ y^{(n-1)}(t_0) = y_0^{(n-1)}$ лише після зведення його до системи ***n диференціальних рівнянь 1-го порядку,*** що мають вигляд $y_i' = f_i(y_1, y_2, \dots, y_n, t);\ i = 1,2, \dots, n$ з початковими умовами $y_1(t_0) = y_0;\ y_2(t_0) = y_0';\ \dots;\ y_n(t_0) = y_0^{(n-1)}$, для чого потрібно здійснити трансформацію ***диференціального рівняння n-го порядку*** в ***n-вимірний простір змінних стану*** $y_i;\ i = 1,2, \dots, n.$

А в аргументні дужки програмної функції ***odeint(, ,)*** потрібно вписати три аргументи, перший з яких ***func(y,t)***, де $y = [y_1,\ y_2, \dots,\ y_n]$ є функцією правих частин системи ***n диференціальних рівнянь 1-го порядку***, другий аргумент – ***список*** початкових значень $\boldsymbol{y(t_0) = [y_1(t_0), y_2(t_0), \dots, y_n(t_0)]}$, а третій аргумент – ***це масив*** значень незалежної змінної $\boldsymbol{t}$, в які ми хочемо визначити значення розв'язку $\boldsymbol{y(t)}$.

Що ж стосується програмної функції ***ode( )***, то вона розв'язує і ті диференціальні рівняння, з якими не може впоратись більш проста програмна функція ***odeint(, ,).***

Усе, що висловлено вище стосовно розв'язання диференціальних рівнянь з використанням програмних функцій пакета ***scipy***, проілюстроване в ***прикладах № 53, № 54, № 55.***

Приклад № 53, присвячений знаходженню числового розв'язку диференціального рівняння 1-го порядку з використанням програмної функції ***odeint(, ,)*** (результат – на рис. 3.11)

```
In [1]: import numpy as np                          # Виклик ППП numpy як np
In [2]: import matplotlib                           # Виклик matplotlib
In [3]: import matplotlib.pyplot as plt             # Виклик matplotlib. pyplot як plt
In [4]: import scipy                                # Виклик ППП scipy
In [5]: from scipy.integrate import odeint          # Виклик  функції odeint
In [6]: def dydt(y,t):                              # Формування диференціального рівняння
            return np.exp(-t)*np.sin(t)**2

In [7]: t=np.linspace(0.0,3.0,31)                   # Формування масиву для  t  в ППП питру
In [8]: y0=0.0                                      # Внесення початкової умови
In [9]: y=odeint(dydt,y0,t)                         # Розв'язання диференціального рівняння
In [10]: y=np.array(y).flatten( )                   # Трансформація розв'язку в масив
In [11]: fig=plt.figure(facecolor='white')          # Формування поля рисунка
In [12]: plt.plot(t,y,'-g',linewidth=4)             # Побудова графіка розв'язку
```

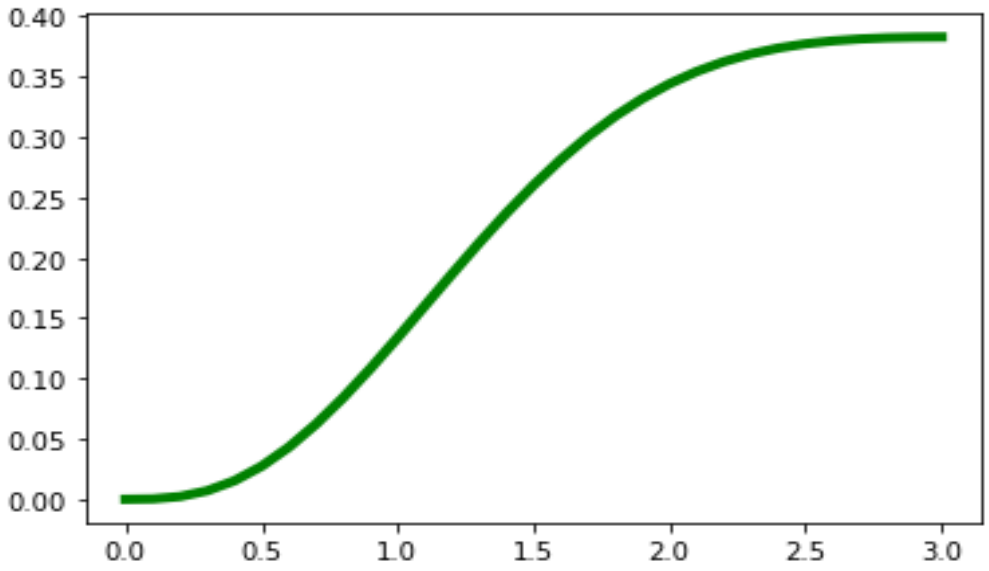


Рисунок 3.11 – Графік розв'язку диференціального рівняння з прикладу № 53

Приклад № 54, присвячений знаходженню числового розв'язку системи диференціальних рівнянь 1-го порядку програмною функцією ***odeint(, ,)*** (результат – на рис. 3.12).

```
In [1]: import numpy as np                    # Виклик ППП numpy як np
In [2]: import matplotlib                     # Виклик ППП matplotlib
In [3]: import matplotlib.pyplot as plt       # Виклик matplotlib. pyplot як plt
In [4]: import scipy                          # Виклик ППП scipy
In [5]: from scipy.integrate import odeint    # Виклик  функції odeint
In [6]: def dydt(y,t):                        # Формування системи диференціальних
            y1,y2=y                             рівнянь 1-го порядку
            return [y2-2*y1,y2+2*y1]

In [7]: t=np.linspace(0.0,3.0,31)             # Формування масиву для  t  в ППП питру
In [8]: y0=[0.0,0.0]                          # Внесення початкової умови
In [9]: w=odeint(dydt,y0,t)                   # Розв'язання системи диференціальних
                                                рівнянь
In [10]: y1=np.array(w[:,0]).flatten( )       # Трансформація першого розв'язку в
                                                масив
In [11]: y2=np.array(w[:,1]).flatten( )       # Трансформація другого розв'язку в масив
In [12]: fig=plt.figure(facecolor='white')    # Формування поля рисунка
In [12]: plt.plot(t,y1,'-g',t,y2,'-.b',       # Побудова графіків розв'язків
             linewidth=4)
```

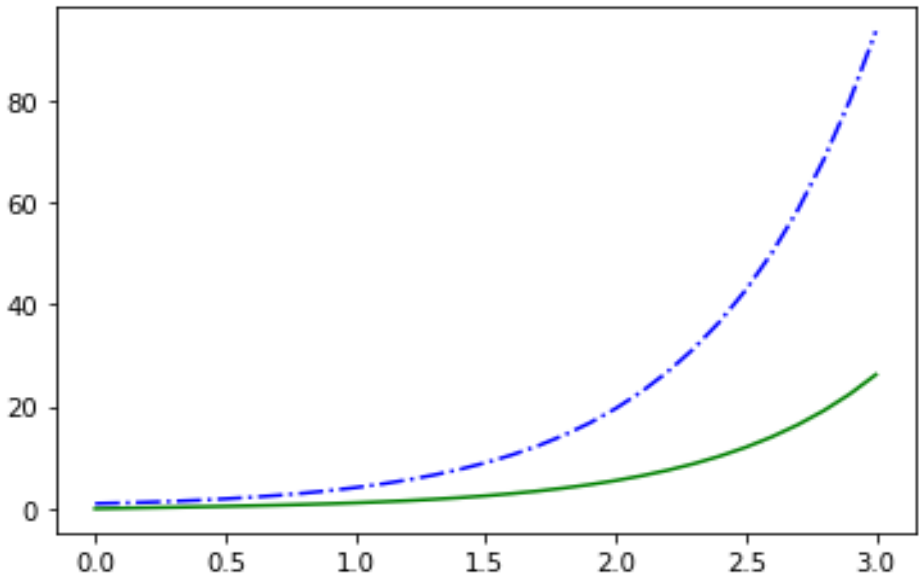


Рисунок 3.12 – Графіки розв'язків системи диференціальних рівнянь з прикладу № 54

Приклад № 55, присвячений знаходженню числового розв'язку диференціального рівняння 2-го порядку програмною функцією ***odeint( )*** (результат – на рис. 3.13)

```
In [1]: import numpy as np                  # Виклик ППП numpy як np
In [2]: import matplotlib                   # Виклик ППП matplotlib
In [3]: import matplotlib.pyplot as plt     # Виклик matplotlib. pyplot як plt
In [4]: import scipy                        # Виклик ППП scipy
In [5]: from scipy.integrate import odeint  # Виклик  функції odeint
In [6]: def f(y,t):                         # Формування системи диференціальних
            y1,y2=y                           рівнянь 1-го порядку після трансформації
            return [y2,-t*y2-y1**0.5+         диференціального рівняння 2-го порядку
                2*np.cos(3*t)]                у цю систему
....
In [7]: t=np.linspace(0.0,3.0,31)           # Формування масиву для t в питру
In [8]: y0=[0.0,1.5]                        # Внесення початкових умов
In [9]: w=odeint(f,y0,t)                    # Розв'язання системи диференціальних
                                              рівнянь 1-го порядку
In [10]: y1=np.array(w[:,0]).flatten( )     # Трансформація розв'язку в масив
In [11]: fig=plt.figure(facecolor='white')  # Формування поля рисунка
In [12]: plt.plot(t,y1,'-k',linewidth=4)    # Графік розв'язку
```

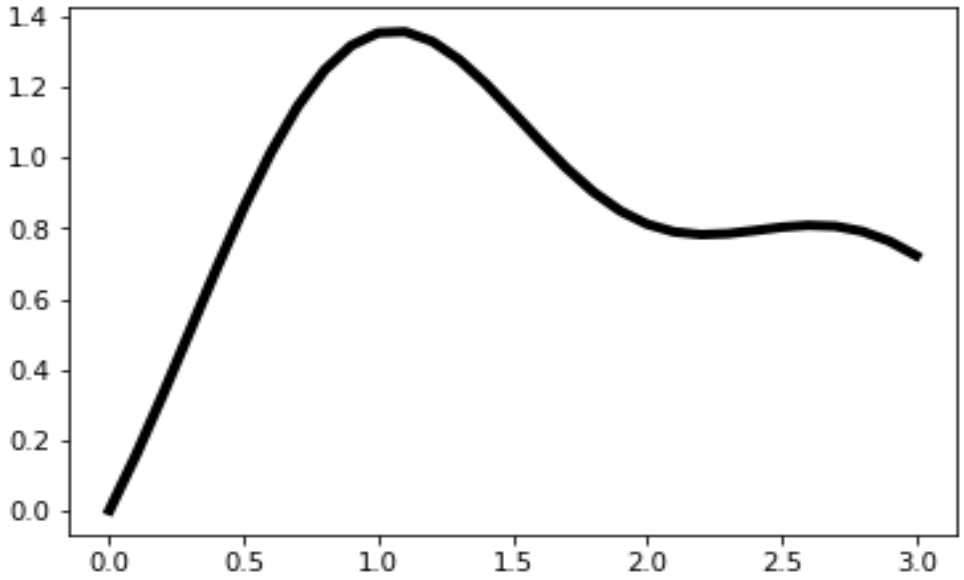


Рисунок 3.13 – Графік розв'язку диференціального рівняння 2-го порядку з прикладу № 55

3.2.2 Чисельне інтегрування функцій в рамках програм ППП scipy

Чисельне інтегрування функцій в рамках ***Python-програм*** ППП ***scipy*** при однократному інтегруванні реалізується програмною функцією ***quad(, ,)***, при подвійному інтегруванні – програмною функцією ***dblquad(, , , ,)***, при потрійному інтегруванні – програмною функцією ***tplquad(, , , , , ,),*** які потрібно імпортувати ***з*** модуля ***scipy.integrate.***

Функція ***quad(, ,)***, яка реалізує процес однократного чисельного інтегрування заданої в ППП ***numpy*** функції ***f(x)*** незалежної змінної ***x***, в аргументних дужках першою координатою має функцію, що інтегрується, другою координатою має нижню межу інтегрування, а третьою координатою має верхню межу інтегрування.

Функція ***dblquad(, , , ,)***, яка реалізує процес подвійного чисельного інтегрування заданої в ППП ***numpy*** функції ***f(x,y)*** двох незалежних змінних ***x,y***, в аргументних дужках першою координатою має функцію, що інтегрується, другою координатою – нижню межу інтегрування зовнішнього інтеграла, третьою координатою – верхню межу інтегрування зовнішнього інтеграла, четвертою координатою – нижню межу інтегрування внутрішнього інтеграла, а п'ятою координатою – верхню межу інтегрування внутрішнього інтеграла; і при цьому в записі функції, що інтегрується, першим має стояти символ внутрішньої змінної, а другим – символ зовнішньої змінної, а нижня і верхня межі внутрішнього інтеграла, навіть якщо вони константи, мають записуватись як функції, оскільки вони дійсно можуть бути заданими у вигляді функцій зовнішньої змінної.

Функція ***tplquad(, , , , , ,)***, яка реалізує процес потрійного чисельного інтегрування заданої в ППП ***numpy*** функції ***f(x,y,z)*** трьох незалежних змінних ***x,y,z***, в аргументних дужках першою координатою має функцію, що інтегрується, другою координатою – нижню межу інтегрування зовнішнього інтеграла, третьою координатою – верхню межу інтегрування зовнішнього інтеграла, четвертою координатою – нижню межу інтегрування другого внутрішнього інтеграла, п'ятою координатою – верхню межу інтегрування другого внутрішнього інтеграла, шостою координатою – нижню межу інтегрування першого внутрішнього інтеграла, а сьомою координатою – верхню межу інтегрування першого внутрішнього інтеграла; і при цьому в записі функції, що інтегрується, першим має стояти символ внутрішньої змінної, по якій здійснюється перше внутрішнє інтегрування, другим – символ змінної, по якій здійснюється друге внутрішнє інтегрування, а третім – символ, по якому здійснюється зовнішнє інтегрування, а нижня і верхня межі першого внутрішнього інтеграла, навіть якщо вони константи, мають записуватись як функції, оскільки вони дійсно можуть бути заданими у вигляді функцій двох змінних, які не беруть участь в першому внутрішньому інтегруванні, а нижня та верхня межі другого внутрішнього інтеграла, навіть якщо вони константи, теж мають записуватись як функції, оскільки вони дійсно можуть бути заданими у вигляді функцій тієї змінної, яка не бере участь у другому внутрішньому інтегруванні, тобто у вигляді функцій зовнішньої змінної.

В прикладі № 56 продемонстровано дію програмної функції ***quad(, ,)*** у випадку, коли межі інтегрування є дійсними числами з діапазону ***[0,1]*****,** коли тіло функції задається безпосередньо, та коли воно задається параметрично з використанням лямбда-функції з заданими числовими значеннями параметрів.

В прикладі № 57 продемонстровано дію програмної функції ***dblquad(, , , ,)*** у випадку, коли межі інтегрування є дійсними числами з діапазону ***[0,1]*** та піддіапазону ***[0,np.inf]***, де через символ ***np.inf*** в ППП ***numpy*** позначається ∞ (нескінченність).

В прикладі № 58 продемонстровано дію програмної функції ***tplquad(, , , , , ,)*** у випадку, коли межі інтегрування є дійсними числами з діапазону ***[0,1]*** та піддіапазону ***[0,np.inf]***, де через символ ***np.inf*** в ППП ***numpy*** позначається ∞ (нескінченність).

Приклад № 56, присвячений чисельному інтегруванню функції ***f(x)*** програмною функцією ***quad(, ,).***

```
In [1]: import scipy                          # Виклик ППП scipy
In [2]: from scipy.integrate import quad      # Виклик функції quad модуля
                                                  scipy.integrate
In [3]: import numpy as np                    # Виклик ППП numpy під символом np
In [4]: def f(x):                             # Формування власної функції f(x)
            return np.exp(-x)**2*np.cos(x)**3 # Визначення тіла власної функції f(x)
....
```

```
In [5]: q1=quad(f,0,1);q1                    # Обчислення інтеграла від функції f(x)
Out[5]: (0.3398620054810545,                   в заданих межах [0,1]
         3.773226e-15)
In [6]: def f(x,a3,a2,a1,a0):                # Формування власної функції f(x)
            return a3*x**3+a2*x**2+a1*x+a0   # Визначення тіла власної функції f(x)
….                                             з невизначеними коефіцієнтами
In [7]: q3=quad(lambda x: f(x,4,3,2,1),0,1);q3  # Обчислення інтеграла функції f(x)
Out[7]: (4.0, 4.440892098500626e-14)            з використанням лямбда-функції
                                                (друге число в результаті вказує на
                                                значення похибки інтегрування)
```

Приклад № 57, присвячений чисельному інтегруванню функції ***f(x,y)*** програмною функцією ***dblquad(, , , ,).***

```
In [1]: import scipy                         # Виклик ППП scipy
In [2]: from scipy.integrate import dblquad  # Виклик функції dblquad
In [3]: import numpy as np                   # Виклик ППП numpy під символом np
In [4]: f=lambda y,x: np.exp(-y+x)*np.cos(-y+x)  # Формування власної функції f(y,x)
In [5]: g=lambda x: 0                        # Нижня внутрішня межа
In [6]: h=lambda x: 1                        # Верхня внутрішня межа
In [7]: J2=dblquad(f,0,1,g,h);J2             # Обчислення подвійного інтеграла
Out[7]: (0.9888977057628652,                   (друге число в результаті вказує на
         1.528779994947608e-14)                значення похибки інтегрування)
In [8]: g1=lambda x: 0                       # Нижня внутрішня межа
In [9]: h1=lambda x: x                       # Верхня внутрішня межа
In [10]: J21=dblquad(f,0,1,g1,h1);J21        # Обчислення подвійного інтеграла
Out[10]: (0.6436776435894213,
          1.526371853528196e-14)
In [11]: g2=lambda x: 0                      # Нижня внутрішня межа
In [12]: h2=lambda x: x**2                   # Верхня внутрішня межа
In [13]: J22=dblquad(f,0,1,g2,h2);J22        # Обчислення подвійного інтеграла
Out[13]: (0.46048907194512184,
          1.5239637234674068e-14)
In [14]: g3=lambda x: 0                      # Нижня внутрішня межа
In [15]: h3=lambda x: np.inf                 # Верхня внутрішня межа
In [16]: J23=dblquad(f,0,1,g3,h3);J23        # Обчислення подвійного інтеграла
Out[16]: (1.1436776435894211,
          1.588466384676408e-08)
```

Приклад № 58, присвячений чисельному інтегруванню функції ***f(x,y,z)*** програмною функцією ***tplquad(, , , , , ,).***

```
In [1]: import scipy                         # Виклик ППП scipy
In [2]: from scipy.integrate import tplquad  # Виклик функції tplquad
In [3]: import numpy as np                   # Виклик ППП numpy під символом np
In [4]: def f(z,y,x):                        # Формування власної функції f(z,y,x)
            return 1/(x**2+x*y+y**2+y*z+z**2)
```

```
In [5]: g=lambda x: 0                                  # Нижня друга внутрішня межа
In [6]: h=lambda x: 1                                  # Верхня друга внутрішня межа
In [7]: q=lambda y,x: 0                                # Нижня перша внутрішня межа
In [8]: r=lambda y,x: 1                                # Верхня перша внутрішня межа
In [9]: J3=tplquad(f,0,1,g,h,q,r);J3                   # Обчислення потрійного інтеграла
Out[9]: (5969.807421550039,
         667632.1076135307e-12)
In [10]: g1=lambda x: 0                                # Нижня друга внутрішня межа
In [11]: h1=lambda x: x                                # Верхня друга внутрішня межа
In [12]: q1=lambda y,x: 0                              # Нижня перша внутрішня межа
In [13]: r1=lambda y,x: y+x                            # Верхня перша внутрішня межа
In [14]: J31=tplquad(f,0,1,g1,h1,q1,r1); J31           # Обчислення потрійного інтеграла
Out[14]: (0.5826673475239622,
          4.011287899202798e-12)
In [15]: g2=lambda x: 0                                # Нижня друга внутрішня межа
In [16]: h2=lambda x: 1/x                              # Верхня друга внутрішня межа
In [17]: q2=lambda y,x: 0                              # Нижня перша внутрішня межа
In [18]: r2=lambda y,x: 1/(y+x)                        # Верхня перша внутрішня межа
In [19]: J32=tplquad(f,0,np.inf,g2,h2,q2,r2); \
          J32                                          # Обчислення потрійного інтеграла
Out[19]: (1.8839075869187127,
          1244.097475114733e-11)
```

3.2.3 Обчислення довжини ліній та площі поверхонь тіл в рамках програм ППП scipy

Для демонстрації практичного застосування програмних функцій ***scipy.integrate.quad( ), scipy.integrate.dblquad( )*** покажемо як, використовуючи їх, можна розв'язувати задачі, в яких потрібно обчислювати довжини ліній та площі поверхонь тіл.

Наприклад, саме з використанням програмної функції ***scipy.integrate.quad( )*** та рівняння ланцюгової лінії можна обчислити довжину проводів лінії електропередачі та тролейних проводів тролейбусних і трамвайних ліній, а також ліній електроживлення електропотягів між двома точками підвішування цих ліній на сусідніх опорах, без обчислення яких неможливо визначити параметр їх максимального провисання, достатній для запобігання обривів при низьких температурах навколишнього середовища; а з використанням програмної функції ***scipy.integrate.dblquad( )*** та рівняння бічної поверхні можна обчислити площу сферичної чи параболічної антени радіолокатора чи радіотелескопа, без уміння обчислювати яку неможливо розрахувати потрібну потужність радіовипромінювання цих антен.

В прикладах № 59, № 60 ми покажемо, як в рамках ***ППП scipy*** обчислювати довжину лінії та площу бічної поверхні тіла, використовуючи програмні функції: ***scipy.integrate.quad( ), scipy.integrate.dblquad( )***

Приклад № 59, файл ***Python-програми*** обчислення довжини ланцюгової лінії, закріпленої в двох точках, та побудови її графіка (результат – на рис. 3.14)

```
import scipy                                  # Виклик ППП scipy
from scipy.integrate import quad              # Виклик програмної функції quad
import sympy                                  # Виклик ППП sympy
from sympy import*                            # Виклик із sympy усіх функцій
import matplotlib                             # Виклик ППП matplotlib
import matplotlib.pyplot as plt               # Виклик модуля matplotlib.pyplot як plt
import numpy as np                            # Виклик ППП numpy під символом np
x=symbols('x')                                # Оголошення символьною змінної x
f=Function('f')(x)                            # Оголошення символьною функції f(x)
f=cosh(x)                                     # Формування функції f(x)
g=Function('g')(x)                            # Оголошення символьною функції g(x)
g=(1+f.diff(x)**2)**0.5                       # Формування функції g(x)
g1= lambdify(x,g,"numpy")                     # Трансформація функції g(x) в ППП numpy
L=quad(g1,0,2);L                              # Обчислення довжини лінії L в заданих межах
print(L)                                      # Виведення на екран результату обчислення L
fig=plt.figure(facecolor='white')             # Створення поля рисунка білого кольору
ax=plt.axes(xlim=(-1,1),ylim=(0,2))           # Визначення ширини та висоти поля рисунка
f1=lambdify(x,f,"numpy")                      # Трансформація функції f(x) в ППП numpy
x=np.linspace(-1,1,41)                        # Внесення масиву значень змінної x
f1vec=np.vectorize(f1)                        # Векторизація трансформованої функції f(x)
f2=f1vec(x)                                   # Обчислення значень векторизованої функції
ax.plot(x,f2,lw=4)                            # Побудова графіка векторизованої функції
ax.grid(True)                                 # Нанесення координатної сітки на графік
```

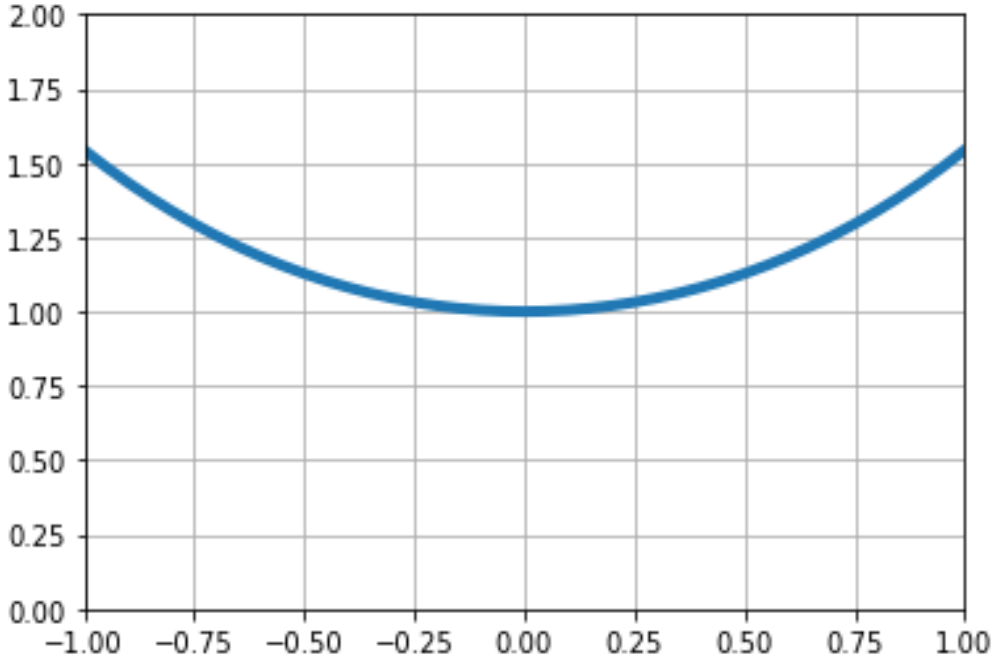


Рисунок 3.14 – Графік ланцюгової лінії, закріпленої в двох точках, що має довжину (3.6268604078470186, 4.0266239318931457e-14)

Приклад № 60, файл ***Python-програми*** обчислення площі антенного параболоїда та побудови його графіка (результат – на рис. 3.15)

```
import scipy                                  # Виклик ППП scipy
from scipy.integrate import dblquad           # Виклик програмної функції dblquad
import sympy                                  # Виклик ППП sympy
from sympy import*                            # Виклик із sympy усіх функцій
import matplotlib as mpl                      # Виклик ППП matplotlib як mpl
from mpl_toolkits.mplot3d import Axes3D       # Виклик внутрішнього модуля Axes3D
import mpl.pyplot as plt                      # Виклик модуля matplotlib.pyplot як plt
import numpy as np                            # Виклик ППП numpy під символом np
x,y=symbols('x y')                            # Оголошення символьними змінних x,y
f=Function('f')(x,y)                          # Оголошення символьною функції f(x,y)
f=x**2+y**2                                   # Формування функції f(x,y)
f1=Function('f1')(x,y)                        # Оголошення символьною функції f1(x,y)
f1=(1+f.diff(x)**2+f.diff(y)**2)**0.5         # Формування функції f1(x,y)
f2= lambdify((y,x),f1,"numpy")                # Трансформація функції f1(x,y) в ППП numpy
g=lambda x: -x**2                             # Нижня внутрішня межа
h=lambda x: x**2                              # Верхня внутрішня межа
s=2*dblquad(f2,-2,2,g,h)[0]                   # Обчислення площі поверхні в заданих межах
print(s)                                      # Виведення на екран результату обчислення s
fig=plt.figure(facecolor='white')             # Створення поля рисунка білого кольору
ax=Axes3D(fig)                                # Прив'язка графопобудовувача до поля рисунка
f3=lambdify((y,x),f,"numpy")                  # Трансформація функції f(x,y) в ППП numpy
u=np.linspace(-4.,4.,100)                     # Внесення масиву значень змінної u
x,y=np.meshgrid(u,u)                          # Прив'язка координат x,y до масиву змінної u
r=x**2+y**2                                   # Формування рівняння зрізів поверхні параболоїда
z=r                                           # Формування координати z, що задає висоту зрізів
ax.plot_surface(x,y,z,rstride=1,cstride=1,    # Побудова поверхні  за координатами
        linewidth=2)                          #    вузлів з одиничним кроком
ax.grid(True)                                 # Нанесення координатної сітки на графік
```

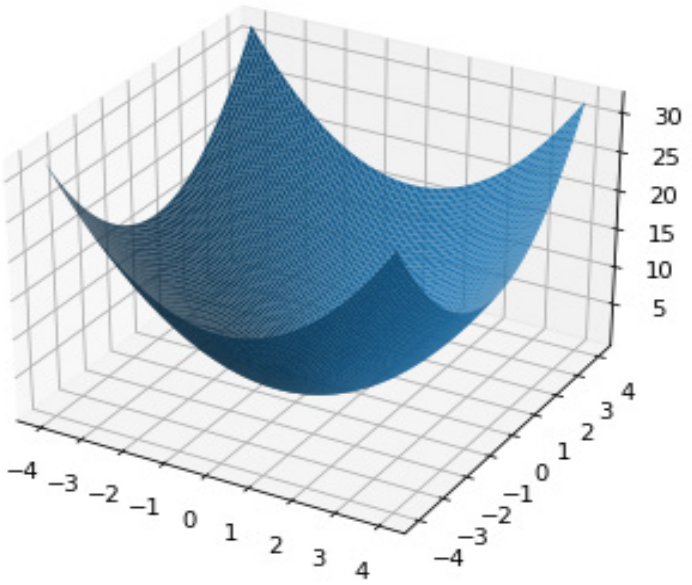


Рисунок 3.15 – Графік поверхні параболоїда, що має площу 88.8601363560463 3

3.3 Генерація випадкових чисел засобами пакету random

У першому розділі цього нашого навчального посібника, присвяченого вивченню основ програмування мовою ***Python***, в пункті 1.4.7 ми вже розглядали тему генерації випадкових чисел програмними функціями модуля ***numpy.random*** в ППП ***numpy***, без використання яких, як ми надалі переконаємось, неможливо синтезувати авторегресійні оператори для моделювання часових рядів, оскільки саме завдяки «зрізанню» цими числами детермінованої послідовності δ-функцій, що генерується в моменти часу, кратні періоду дискретизації часового ряду, ми і отримуємо можливість генерувати послідовність імпульсів з амплітудою, що визначається значенням згенерованого у цей же момент часу випадкового числа. При викладенні цього матеріалу ми використовували відомості, почерпнуті з присвячених програмуванню мовою ***Python*** базових робіт [5], [7], [15]. На ці ж базові джерела інформації ми посилатимемось і при викладенні матеріалу, який подаємо нижче.

І почнемо викладення потрібного нам надалі матеріалу ми з того, що акцентуємо увагу на наявності в ***Python*** окрім модуля ***numpy.random,*** що входить до складу ППП ***numpy,*** ще одного модуля ***random***, який входить в усі три дистрибутиви мови ***Python*** та викликається при використанні будь-якого з них саме за цим іменем з присвоєнням йому для подальшої активації його програмних функцій символа ***rnd***. Цей модуль містить в собі більшість програмних функцій, які дублюють однойменні програмні функції модуля ***numpy.random***, але його перевага в тому, що, користуючись ним, ми можемо генерувати випадкові числа, не покидаючи того пакета програм, в якому працювали до появи потреби генерації випадкових чисел, що спрощує конкретні ***Python-програми***, в яких така потреба виникає.

Отже, далі, виходячи з наших цілей, розглянемо в модулі ***random*** генерацію випадкових чисел найвживанішими програмними функціями:

- ***rnd.randrange*** *(start,stop,step)*, яка вибирає випадкове ціле число ***y*** з діапазону [start, stop) випадкових цілих чисел, що відрізняються між собою не менше ніж на *step*;
- ***rnd.randint*** *(start,stop)*, яка вибирає випадкове ціле число ***y*** з діапазону [start, stop] випадкових цілих чисел;
- ***rnd.uniform*** *(start,stop)*, яка вибирає випадкове раціональне число ***y*** з інтервалу (start, stop);
- ***rnd.random*** *( )*, яка вибирає випадкове раціональне число ***y*** з одиничного відрізка [0,1];
- ***rnd.choice*** *($\{y_i\}$)*, яка вибирає випадкове число $\boldsymbol{y_i}$ з послідовності $\{\boldsymbol{y_i}\}$**,** заданої у формі списку, кортежу або масиву;
- ***rnd.sample*** *($\{y_i\}, k$)*, яка створює список довжиною ***k*** випадково вибраних елементів із послідовності $\{y_i\}$;
- ***rnd.seed*** *( )*, яка запускає генератор випадкових чисел, що реалізує одну з наведених вище функцій, з прив'язкою до системного часу.

Як генерують випадкові числа ці програмні функції продемонстровано і покомандно пояснено в прикладі № 61.

Приклад № 61, генерація випадкових чисел програмними функціями модуля ***random***

```
In [1]: import random as rnd                    # Виклик модуля random як rnd
In [2]: rnd.seed( )                             # Виклик із модуля random функції seed ()
In [3]: rnd.randrange (3,30,4)                  # Генерація цілого випадкового числа
Out[3]: 3                                         з заданого діапазону з заданим кроком
                                                  з використанням граничних значень
In [4]: rnd.randrange (3,30,4)                  # Генерація ще одного цілого випадкового
 Out[4]: 23                                     числа з того ж діапазону з тим же кроком

In [5]: rnd.randint (3,30)                      # Генерація цілого випадкового числа
Out[5]: 17                                        з заданого діапазону з одиничним кроком
                                                  з використанням граничних значень
In [6]: rnd.randint(3,30)                       # Генерація ще одного цілого випадкового
 Out[6]: 5                                        числа з того ж діапазону з тим же кроком

In [7]: rnd.uniform (3,30)                      # Генерація раціонального випадкового числа
Out[7]: 9.19975053747536                          з заданого діапазону без використання
                                                  граничних значень
In [8]: rnd.uniform (3,30)                      # Генерація ще одного раціонального
Out[8]: 13.09875589535126                         випадкового числа з того ж діапазону

In [9]: rnd.random ()                           # Генерація раціонального випадкового числа
Out[9]: 0.841950248480422                         з одиничного відрізка з використанням
                                                  граничних значень
In [10]: rnd.random ()                          # Генерація ще одного раціонального
Out[10]: 0.5516994841556719                       випадкового числа з одиничного діапазону


In [11]: rnd.choice ([1,3,5,7,9,11,13,15])      # Випадковий вибір елемента з заданої
                                                  списком послідовності з використанням
                                                  граничних значень
Out[11]: 3
In [12]: rnd.choice ([1,3,5,7,9,11,13,15])      # Ще один варіант випадкового вибору
Out[12]: 1                                        елемента з заданої списком послідовності

In [13]: rnd.sample ([1,3,5,7,9,11,13,15],3)    # Формування списку із «трьох» елементів,
Out[13]: [3, 5, 9]                                відібраних випадково з заданого списку,
                                                  з використанням граничних значень
In [14]: rnd.sample ([1,3,5,7,9,11,13,15],3)    # Ще один варіант випадкового вибору трьох
Out[14]: [3, 5, 13]                               елементів з заданого списку
```

3.4 Python-програми для аналізу процесів в динамічних системах та для прогнозування часових рядів

Ми завершили написання нашого навчального посібника в задуманому варіанті, і в статусі завершального етапу комплексного використання викладених у ньому знань наведемо дві Python-програми, створені в процесі написання нами іншого навчального посібника [15], присвяченого розв'язанню задач функціонального аналізу.

Перша з цих Python-програм створена для аналізу процесу в динамічній системі з використанням математичних моделей, наведених у другому розділі цього навчального посібника, а друга – для прогнозування часових рядів з використанням математичних моделей, теж наведених у другому розділі.

3.4.1 Python-програма аналізу процесу в динамічній системі з використанням перетворення за Лапласом

Задачу аналізу сформулюємо так:
нехай математична модель динамічної системи у формі передаточної функції має вигляд

$$W(p) = \frac{2p+4}{p^2+7p+12}.$$

На вхід цієї системи подається сигнал

$$x(t) = t.$$

Потрібно визначити реакцію $y(t)$ цієї динамічної системи на даний вхідний сигнал.

Розв'язуватимемо цю задачу з використанням наведеної нижче ***Python-програми*** (результат – на рис. 3.16)***:***

```
In [1]: import sympy                          # Виклик ППП sympy
In [2]: from sympy import *                   # Виклик усіх фукцій sympy
In [3]: t = symbols ('t')                     # Оголошення символьною змінної t
In [4]: p = symbols ('p')                     # Оголошення символьною змінної p
In [5]: x = Function ('x')(t)                 # Символізація функції x(t)
In [6]: y = Function ('y')(t)                 # Символізація функції y(t)
In [7]: C = Function ('C')(p)                 # Символізація функції C(p)
In [8]: D = Function ('D')(p)                 # Символізація функції D(p)
In [9]: W = Function ('W')(p)                 # Символізація функції W(p)
In [10]: X = Function ('X')(p)                # Символізація функції X(p)
In [11]: Y = Function ('Y')(p)                # Символізація функції Y(p)
In [12]: D1 = Function ('D1')(p)              # Символізація функції D1(p)
In [13]: Y1 = Function ('Y1')(p,t)            # Символізація функції Y1(p,t)
In [14]: Y2 = Function ('Y2')(p,t)            # Символізація функції Y2(p,t)
In [15]: C = 2*p+4                            # Формування функції C(p)
```

```
In [16]: D = p**2+7*p+12                    # Формування функції D(p)
In [17]: W = C/D, W                         # Формування функції W(p)
Out[17]:
(2*p + 4)/(p**2 + 7*p + 12)
In [18]: x = t                              # Внесення функції x(t)
In [19]: x1 = x*exp(-p*t)                   # Формування ядра x1 інтеграла Лапласа
In [20]: X1 = Function ('X1')(p)            # Символізація функції X1(p)
In [21]: X1 = integrate (x1,(t,0,oo));X1    # Інтегрування функції x1
Out[21]:
Piecewise((p**(-2), Abs(arg(p)) < pi/2),    # Інтеграл як складена функція-кортеж
(Integral(t*exp(-p*t), (t, 0, oo)), True))
In [22]: X11=(p**(-2), Abs(arg(p)) < pi/2)  # Виокремлення кортежа зі складовою
                                              L{x(t)} з тіла складеної функції
In [23]: X2=Function ('X2')(p)              # Символізація функції X2(p)
In [24]: X2 = X11[0][0], X2                 # Обчислення оператора L{x(t)} = X2(p)
Out[24]:
p**(-2)
In [25]: Y = W*X2, Y                        # Визначення зображення Y(p) за Лапласом
Out[25]:
(2*p + 4)/(p**2*(p**2 + 7*p + 12))
In [26]: C1 = Function ('C1')(p)            # Символізація функції C1(p)
In [27]: expr = Y                           # Створення виразу expr
In [28]: C1, D1 = fraction (expr)           # Фракціонування виразу expr
In [29]: D2 = Function ('D2')(p)            # Символізація функції D2(p)
In [30]: D2 = D1.diff(p)                    # D2(p) як похідна від D1(p)
In [31]: roots (Eq(D1,0), p)                # Визначення полюсів Y(p)
Out[31]: {-3: 1, -4: 1, 0: 2}                 з рівняння D1(p)=0
In [32]: d0 ={ }                            # Створення пустого словника d0
In [33]: d0["a"]=-3                         # Внесення в d0 полюсів Y(p)
In [34]: d0["b"]=-4                           з ключами “a”,”b”,”c”
In [35]: d0["c"]=0
In [36]: d0                                 # Виклик на екран заповненого словника
Out[36]: {'a': -3, 'b': -4, 'c': 0}
In [37]: p1,p2,p3 = symbols('p1 p2 p3')     # Оголошення символьними p1, p2, p3
In [38]: p1 = d0['a'], p1                   # Формування полюса p1 зображення Y(p)
Out[38]: -3
In [39]: p2 = d0['b'], p2                   # Формування полюса p2 зображення Y(p)
Out[39]: -4
In [40]: p3 = d0['c'], p3                   # Формування полюса p3 зображення Y(p)
Out[40]: 0
In [41]: Y1 = C1*exp(p*t)/D2                # Формування складової функції y(t),
                                              зумовленої простим полюсом Y(p)
In [42]: Y2 = diff(C1*(p-p3)**2*            # Формування складової функції y(t),
                exp(p*t)/D1,p)                зумовленої кратним полюсом Y(p)
In [43]: y = Y1.subs(p,p1)+Y1.subs(p,p2)+   # Визначення y(t) як L^{-1}{Y(p)}
         Y2.subs(p,p3), y                     застосуванням теореми розкладання
Out[43]:
t/3 - 1/36 - 2*exp(-3*t)/9 + exp(-4*t)/4
```

```
In [44]: p11 = plot(x,(t,0,2),show=\          # Побудова графіка x(t) без екранізації
         False, line_color = 'c')
In [45]: p22 = plot(y,(t,0,2),show=\          # Побудова графіка y(t) без екранізації
         False, line_color = 'r')
In [46]: p11.extend(p22)                      # Поєднання на одному рисунку обох графіків
In [47]: p11.show( )                          # Виведення на екран рисунка з графіками
```

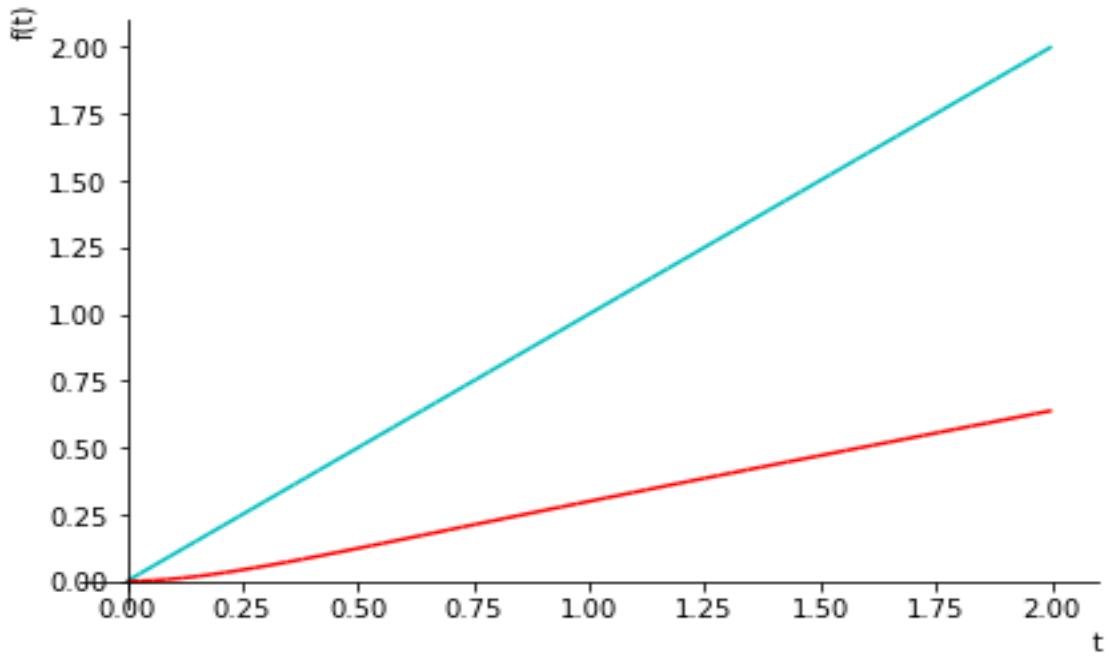


Рисунок 3.16 – Графіки вхідного сигналу ***x(t) = t***, що діє на динамічну систему з заданою передаточною функцією ***W(p)***, та її вихідного сигналу ***y(t)*** на відрізку часу $t \in [0,2]$

3.4.2 Python-програма синтезу авторегресійної моделі часового ряду для прогнозування його наступних значень

Задачу прогнозування часового ряду сформулюємо так:
нехай нами зафіксовані *N* послідовних значень стаціонарного часового ряду

y_t={5.,8.,3.,4.,6.,3.,2.,7.,5.,4.,3.,6.,4.,5.,3.,8.,6.,4.,3.,5.,4.,2.,7.,4.,5.,3.,6.,3.,4.,5.},

заданих в точках $t \in [1, N]$ при *N*=30.

Потрібно **з використанням моделі АР(3) зі структурою**

$$y_t = b + m_t,$$

$$b = \frac{1}{N}\sum_{i=1}^{N} y_i,$$

$$m_t = g_1 m_{t-1} + g_2 m_{t-2} + g_3 m_{t-3} + a_t$$

та алгоритму Юла-Уокера визначення її параметрів здійснити прогнозування наступних 10 його значень.

Розв'язуватимемо цю задачу з використанням наведеної нижче ***Python-програми*** (результат – на рис. 3.17).

```
In [1]: import numpy as np                  # Виклик ППП numpy як np
In [2]: L=[5.,8.,3.,4.,6.,3.,2.,7.,5.,4.,3.,  # Внесення списку з N
        6.,4.,5.,3.,8.,6.,4.,3.,5.,4.,2.,7.,4.,   значень часового ряду
        5.,3.,6.,3.,4.,5.]
In [3]: N=30                                # Фіксація N=30
In [4]: def fun (x):                        # Формування функції, яка
            return np.sum(x)                   визначає суму членів числового масиву

In [5]: fun(L)                              # Використання сформованої функції
Out[5]: 137.0                                  для визначення суми членів списку L
In [6]: b=_/N;b                             # Обчислення параметра b часового ряду
Out[6]: 4.566666666666666
In [7]: b=b.round(3);b                      # Залишення в b лише трьох знаків після коми
Out[7]: 4.567
In [8]: L1=L-b                              # Центрування часового ряду
In [9]: def fun (x):                        # Формування функції для скалярного добутку
            return np.dot(x,x)

In [10]: fun(L1,L1)                         # Визначення скалярного добутку L1 з L1
Out[10]:
77.36667
In [11]: q0=_/N; q0                         # Обчислення дисперсії q0 часового ряду
Out[11]:
 2.5788889999999998
In [12]: q0=q0.round(3); q0                 # Залишення в q0  трьох знаків після коми
Out[12]:
 2.579
In [13]: L2=L1[:-1]                         # Вилучення зі списку L1 останнього члена
In [14]: L5=L1[1:]                          # Вилучення зі списку L1 першого члена
In [15]: def fun (x,y):                     # Формування функції для визначення
             return np.dot(x,y)                скалярного добутку масиву x з масивом y

In [16]: fun(L2, L5)                        # Обчислення скалярного добутку L2 з L5
Out[16]:
 -22.820818999999993
In [17]: q1=_/(N-1); q1                     # Обчислення автоковаріації q1 часового ряду
Out[17]:
 -0.786924793103448
In [18]: q1=q1.round(3);q1                  # Залишення в q1  трьох знаків після коми
Out[18]:
 -0.787
In [19]: L3=L2[:-1]                         # Вилучення зі списку L2 останнього члена
In [20]: L6=L5[1:]                          # Вилучення зі списку L5 першого члена
In [21]: fun(L3,L6)                         # Обчислення скалярного добутку L3 з L6
```

```
Out[21]:
-14.874308000000001
In [22]: q2=_/(N-2);q2                  # Обчислення автоковаріації q2 часового ряду
Out[22]:
 -0.5312252857142857
In [23]: q2=q2.round(3);q2              # Залишення в q2  трьох знаків після коми
Out[23]:
-0.531
In [24]: L4=L3[:-1]                     # Вилучення зі списку L3 останнього члена
In [25]: L7=L6[1:]                      # Вилучення зі списку L6 першого члена
In [26]: fun(L4,L7)                     # Обчислення скалярного добутку L4 з L7
Out[26]:
2.6702030000000008
In [27]: q3=_/(N-3)                     # Обчислення автоковаріації q3 часового ряду
Out[27]:
0.09889640740740743
In [28]: q3=q3.round(3); q3             # Залишення в q3  трьох знаків після коми
Out[28]:
0.099
In [29]: r0=q0/q0;r0                    # Обчислення автокореляції r0
Out[29]:
 1.0
In [30]: r1=q1/q0;r1                    # Обчислення автокореляції r1
Out[30]:
 -0.30515703761147733
In [31]: r1=r1.round(3);r1              # Залишення в r1  трьох знаків після коми
Out[31]:
 -0.305
In [32]: r2=q2/q0;r2                    # Обчислення автокореляції r2
Out[32]:
 -0.2058937572702598
In [33]: r2=r2.round(3);r2              # Залишення в r2 трьох знаків після коми
Out[33]:
 -0.206
In [34]: r3=q3/q0;r3                    # Обчислення автокореляції r3
Out[34]:
 0.038386971694455214
In [35]: r3=r3.round(3);r3              # Залишення в r3  трьох знаків після коми
Out[35]:
 0.038
In [36]: L9=[r0,r1,r2,r3];L9            # Формування списку L9 автокореляцій
Out[36]:
 [1.0, -0.305, -0.206, 0.038]
In [37]: import sympy                   # Виклик ППП sympy
In [38]: from sympy import*             # Виклик усіх функцій з sympy
In [39]: r,r0,r1,r2,r3 =
         symbols('r r0 r1 r2 r3')       # Символізація автокореляцій
```

```
In [40]: M = symbols('M')                    # Символізація матриці Юла-Уокера
In [41]: M = Matrix([[r0,r1,r2],[r1,r0,r1],  # Форматування матриці Юла-Уокера
         [r2,r1,r0]]);M                        в загальному вигляді для АР(3)
Out[41]:
Matrix([
[r0, r1, r2],
[r1, r0, r1],
[r2, r1, r0]])
In [42]: g,g1,g2,g3=symbols('g g1 g2 g3')    # Символізація параметрів авторегресії
In [43]: g = Matrix([g1,g2,g3]);g            # Форматування матриці-стовпця g
Out[43]:                                       в загальному вигляді
Matrix([
[g1],
[g2],
[g3]])
In [44]: M =M.subs([(r0,1),(r1,-0.305),      # Ідентифікація матриці Юла-Уокера
         (r2,-0.206)]);M
Out[44]:
Matrix([
[     1, -0.305, -0.206],
[-0.305,      1, -0.305],
[-0.206, -0.305,      1]])
In [45]: r=Matrix([r1,r2,r3]); r             # Форматування матриці автокореляцій r
Out[45]:                                       в загальному вигляді
Matrix([
[r1],
[r2],
[r3]])
In [46]: r=r.subs([(r1,-0.305),(r2,-0.206),  # Ідентифікація матриці автокореляцій r
(r3,0.038)]); r
Out[46]:
Matrix([
[-0.305],
[-0.206],
[ 0.038]])
In [47]: B=simplify(M.inv())                 # Обчислення оберненої матриці
In [48]: g=B*r; g                            # Розв'язання системи рівнянь Юла-Уокера
In [49]: g=g.evalf(3); g                     # Скорочення до трьох знаків
Out[49]:                                       в числових значеннях параметрів
Matrix([                                       авторегресії АР(3)
[-0.465],
[-0.403],
[-0.181]])
In [50]: g1=g[0,0]                           # Ідентифікація параметра g1
In [51]: g2=g[1,0]                           # Ідентифікація параметра g2
In [52]: g3=g[2,0]                           # Ідентифікація параметра g3
In [53]: a = symbols ('a')                   # Символізація імпульсу a білого шуму
```

```
In [54]: ska = symbols ('ska')                    # Символізація дисперсії ska білого шуму
In [55]: ska = q0-g1*q1-g2*q2-g3*q3; ska          # Обчислення дисперсії ska за Юлом-Уокером
Out [55]: 2.01681854248047
In [56]: skv = symbols ('skv')                    # Символізація СКВ skv білого шуму
In [57]: skv = ska**(0.5); skv                    # Обчислення СКВ skv білого шуму
In [58]: skv = skv.evalf(3); skv                  # Скорочення до трьох знаків
Out[58]:                                            в числовому значенні skv
1.42
In [59]: a11,a22 = symbols ('a11 a22')            # Символізація меж діапазону генерації
In [60]: a11 = -3*skv; a11                        # Обчислення нижньої a11 межі
Out[60]:
-4.26
In [61]: a22 = 3*skv; a22                         # Обчислення верхньої a22 межі
Out[61]:
4.26
In [62]: import random as rnd                     # Виклик модуля генерації
                                                  імпульсів білого шуму
In [63]: m = symbols('m:10')                      # Індексна символізація параметра m
In [64]: l = symbols ('l:10')                     # Індексна символізація параметра l
In [65]: m=list(m);m                              # Створення списку для параметра m
Out[65]:
 [m0, m1, m2, m3, m4, m5, m6, m7, m8, m9]
In [66]: l=list(l);l                              # Створення списку для параметра l
Out[66]:
 [l0, l1, l2, l3, l4, l5, l6, l7, l8, l9]
In [67]: d = symbols ('d:10')                     # Індексна символізація параметра d
In [68]: d = list (d); d                          # Створення списку для параметра d
Out[68]:
[d0,d1,d2,d3,d4,d5,d6,d7,d8,d9]
In [69]: d[0] = rnd.uniform (-4.26,4.26); d[0]    # Генерація випадкового імпульсу d[0]
Out[69]:                                            для першої ітерації прогнозу
3.7878324717658174
In [70]: m[0]=g1*L1[30]+g2*L1[29]+\               # Обчислення N+1-го члена списку L1
g3*L1[28]+ d[0]; m[0]                               з використанням моделі авторегресії
Out[70]:
4.09812879744941
In [71]: L1 = np.append(L1,[m[0]])                # Додавання в кінці L1  члена m[0]
Out[71]:
In [72]: l[0]=b+m[0]; l[0]                        # Обчислення N+1-го члена списку L
Out[72]:
8.66512879744941
In [73]: L = np.append(L,[l[0]])                  # Додавання в кінці L  члена l[0]
In [74]: d[1] = rnd.uniform (-4.26,4.26); d[1]    # Генерація випадкового імпульсу d[1]
Out[74]:                                            для другої ітерації прогнозу
 -2.587298897242724
In [75]: m[1]= g1*L1[31]+g2*L1[30]+\              # Обчислення N+2-го члена списку L1
g3*L1[29]+ d[1]; m[1]                               з використанням моделі авторегресії
```

```
Out[75]:
-2.08074576204602
In [76]: L1 = np.append(L1,[m[1]])          # Додавання в кінці L1 члена m[1]
In [77]: l[1]=b+m[1]; l[1]                  # Обчислення N+2 -го члена списку L
Out[77]:
2.48625423795398
In [78]: L=np.append(L,[l[1]])              # Додавання в кінці L члена l[1]
In [79]: d[2] = rnd.uniform (-4.26,4.26); d[2]  # Генерація випадкового імпульсу d[2]
Out[79]:                                        для третьої ітерації прогнозу
 0.8071389548290568
In [80]: m[2]= g1*L1[32]+g2*L1[31]+\        # Обчислення N+3 -го члена списку L1
g3*L1[30]+ d[2]; m[2]                           з використанням моделі авторегресії
Out[80]:
0.0451337059883733
In [81]: L1 = np.append(L1,[m[2]])          # Додавання в кінці L1 члена m[2]
In [82]: l[2]=b+m[2]; l[2]                  # Обчислення N+3 -го члена списку L
Out[82]:
4.61213370598837
In [83]: L=np.append(L,[l[2]])              # Додавання в кінці L члена l[2]
In [84]: d[3] = rnd.uniform (-4.26,4.26); d[3]  # Генерація випадкового імпульсу d[3]
Out[84]:                                        для четвертої ітерації прогнозу
3.5146713080878706
In [85]: m[3]= g1*L1[33]+g2*L1[32]+\        # Обчислення N+4-го члена списку L1
g3*L1[31]+ d[3]; m[3]                           з використанням моделі авторегресії
Out[85]:
3.59161472387185
In [86]: L1 = np.append(L1,[m[3]])          # Додавання в кінці L1 члена m[3]
In [87]: l[3]=b+m[3]; l[3]                  # Обчислення N+4-го члена списку L
Out[87]:
8.15861472387185
In [88]: L=np.append(L,[l[3]])              # Додавання в кінці L члена l[3]
In [89]: d[4] = rnd.uniform (-4.26,4.26); d[4]  # Генерація випадкового імпульсу d[4]
Out[89]:                                        для п'ятої ітерації прогнозу
3.332153915638866
In [90]: m[4]= g1*L1[34]+g2*L1[33]+\        # Обчислення N+5-го члена списку L1
g3*L1[32]+ d[4]; m[4]                           з використанням моделі авторегресії
Out[90]:
0.178801469969288
In [91]: L1 = np.append(L1,[m[4]])          # Додавання в кінці L1 члена m[4]
In [92]: l[4]=b+m[4]; l[4]                  # Обчислення N+5-го члена списку L
Out[92]:
4.74580146996929
In [93]: L=np.append(L,[l[4]])              # Додавання в кінці L нового члена l[4]
In [94]: d[5] = rnd.uniform (-4.26,4.26); d[5]  # Генерація випадкового імпульсу d[5]
Out[94]:                                        для шостої ітерації прогнозу
2.867785147080328
```

```
In [95]: m[5]= g1*L1[35]+g2*L1[34]+\          # Обчислення N+6-го члена списку L1
g3*L1[33]+ d[5]; m[5]                          з використанням моделі авторегресії
Out[95]:
1.32898394299138
In [96]: L1 = np.append(L1,[m[5]])            # Додавання в кінці L1 члена m[5]
In [97]: l[5]=b+m[5]; l[5]                    # Обчислення N+6-го члена списку L
Out[97]:
5.89598394299138
In [98]: L=np.append(L,[l[5]])                # Додавання в кінці L члена l[5]
In [99]: d[6] = rnd.uniform (-4.26,4.26); d[6] # Генерація випадкового імпульсу d[6]
Out[99]:                                       для сьомої ітерації прогнозу
1.69270588124252
In [100]: m[6]= g1*L1[36]+g2*L1[35]+\         # Обчислення N+7-го члена списку L1
g3*L1[34]+ d[6]; m[6]                          з використанням моделі авторегресії
Out[100]:
-0.471995713797095
In [101]: L1 = np.append(L1,[m[6]])           # Додавання в кінці L1 члена m[6]
In [102]: l[6]=b+m[6]; l[6]                   # Обчислення N+7-го члена списку L
Out[102]:
4.09500428620290
In [103]: L=np.append(L,[l[6]])               # Додавання в кінці L члена l[6]
In [104]: d[7] = rnd.uniform (-4.26,4.26); d[7] # Генерація випадкового імпульсу d[7]
Out[104]:                                      для восьмої ітерації прогнозу
0.10021632009046666
In [105]: m[7]= g1*L1[37]+g2*L1[36]+\         # Обчислення N+8-го члена списку L1
g3*L1[35]+ d[7]; m[7]                          з використанням моделі авторегресії
Out[105]:
-0.248149939266434
In [106]: L1 = np.append(L1,[m[7]])           # Додавання в кінці L1 члена m[7]
In [107]: l[7]=b+m[7]; l[7]                   # Обчислення N+8 -го члена списку L
Out[107]:
4.31885006073357
In [108]: L=np.append(L,[l[7]])               # Додавання в кінці L нового члена l[7]
In [109]: d[8] = rnd.uniform (-4.26,4.26); d[8] # Генерація випадкового імпульсу d[8]
Out[109]:                                      для дев'ятої ітерації прогнозу
-2.232660689069062
In [110]: m[8]= g1*L1[38]+g2*L1[37]+\         # Обчислення N+9-го члена списку L1
g3*L1[36]+ d[8]; m[8]                          з використанням моделі авторегресії
Out[110]:
-2.16719334714070
In [111]: L1 = np.append(L1,[m[8]])           # Додавання в кінці L1 члена m[8]
In [112]: l[8]=b+m[8]; l[8]                   # Обчислення N+9-го члена списку L
Out[112]:
2.39980665285930
In [113]: L=np.append(L,[l[8]])               # Додавання в кінці L члена l[8]
In [114]: d[9] = rnd.uniform (-4.26,4.26); d[9] # Генерація випадкового імпульсу d[9]
```

```
Out[114]:                                          для десятої ітерації прогнозу
1.544009122634895
In [115]: m[9]= g1*L1[39]+g2*L1[38]+\              # Обчислення N+10-го члена списку L1
g3*L1[37]+ d[9]; m[9]                                з використанням моделі авторегресії
Out[115]:
2.73738643318719
In [116]: L1 = np.append(L1,[m[9]])                # Додавання в кінці L1  члена m[9]
In [117]: l[9]=b+m[9]; l[9]                        # Обчислення N+10-го члена списку L
Out[117]:
7.30438643318719
In [118]: L=np.append(L,[l[9]])                    # Додавання в кінці L  члена l[9]
In [119]: import matplotlib                        # Виклик ППП matplotlib
In [120]: import matplotlib.pyplot as plt          # Виклик модуля matplotlib.pyplot
In [121]: plt.plot(L1)                             # Побудова графіка центрованого
                                                     часового ряду
```

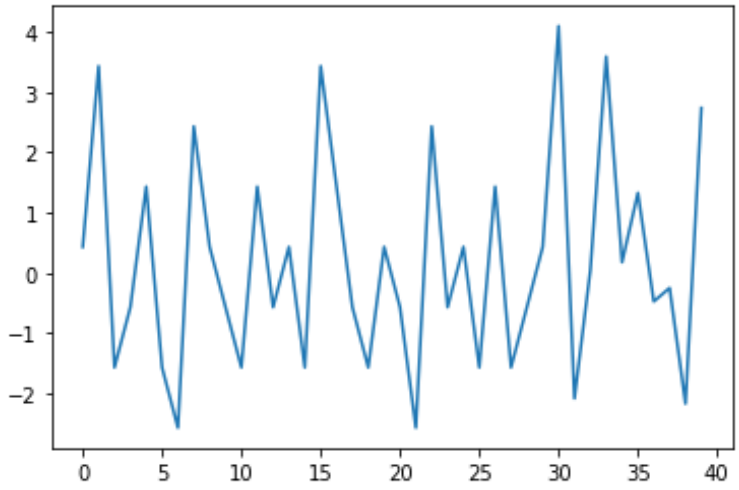


Рисунок 3.17 – Графік часового ряду m_t, який в діапазоні $t \in [1,30]$ заповнений експериментально отриманими значеннями, а за межами цього діапазону заповнений прогнозними значеннями, отриманими з використанням авторегресійної моделі, ідентифікованої з використанням експериментально отриманих значень

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Комп’ютерне моделювання систем та процесів. Методи обчислень. Част. 1 : навчальний посібник / Р. Н. Квєтний та ін. Вінниця : ВНТУ, 2013. 191 с.
2. Комп’ютерне моделювання систем та процесів. Методи обчислень. Част. 2 : навчальний посібник / Р. Н. Квєтний та ін. Вінниця : ВНТУ, 2013. 235 с.
3. Методи та алгоритми комп’ютерних обчислень. Теорія і практика : підручник / Р. Н. Квєтний та ін. Вінниця : ВНТУ, 2023. 280 с.
4. IDE Anaconda. Режим доступу: https://anaconda.com
5. Python. [Електронний ресурс]. Режим доступу: https://www.python.org//downloads/
6. Python. Керівництво [Електронний ресурс]. https://www.python.org/getit/.
7. Доля П. Г. Уведення в науковий Python. Харків : ХНУ ім. Каразіна, 2016. 265 с.
8. Мокін Б. І., Мокін О. Б., Мокін В. Б. Методологія та організація наукових досліджень : підручник. Вінниця : ВНТУ, 2023. 230 с.
9. Мокін Б. І., Мокін, О. Б., Мокін В. Б. Практикум для самостійної роботи студентів з навчальної дисципліни «Методологія та організація наукових досліджень». Частина 1: від постановки задачі до синтезу та ідентифікації математичної моделі. Вінниця : ВНТУ, 2018. 179 с. - Режим доступу: http://www.mokin.com.ua/pedagogical/posibn/6546.html
10. Мокін Б. І., Мокін О. Б. Теорія автоматичного керування, методологія та практика оптимізації : навчальний посібник. Вінниця : ВНТУ, 2013. 210 с.
11. Мокін Б. І., Мокін В. Б., Мокін О. Б. Математичні методи ідентифікації динамічних систем : навчальний посібник. Вінниця : ВНТУ, 2010. 260 с.
12. Мокін Б. І., Мокін В. Б, Мокін О. Б. Функціональний аналіз, адаптований до прикладних задач в галузі інформаційних технологій : навчальний посібник. Вінниця : ВНТУ, 2020. 192 с.
13. Box George E. P., Jenkins Gwilym M. TIME SERIES ANALYSIS. Forecasting and control. HOLDEN-DAY: San Francisco, Cambridge, London, Amsterdam, 1970. 532 p.
14. Мокін Б. І., Мокін В. Б., Мокін О. Б. Навчальний посібник для опанування студентами способів розв’язання задач з функціонального аналізу мовою Python. Частина 1. Вінниця : ВНТУ, 2022. 124 с.
15. Мокін Б. І., Мокін В. Б., Мокін О. Б. Навчальний посібник для опанування студентами способів розв’язання задач з функціонального аналізу мовою Python. Частина 2. Вінниця : ВНТУ, 2023. 144 с.

Електронне навчальне видання

Борис Іванович Мокін
Віталій Борисович Мокін
Олександр Борисович Мокін

МЕТОДИ ТА ЗАСОБИ КОМП'ЮТЕРНИХ ОБЧИСЛЕНЬ

Навчальний посібник

Рукопис оформив *Б. Мокін*

Редактор *В. Дружиніна*

Файл підготовано у *Редакційно-видавничому відділі ВНТУ*

Підписано до видання 24.12.2024 р.
Гарнітура Times New Roman.
Зам. № Р2024-202.

Видавець та виготовлювач
Вінницький національний технічний університет,
Редакційно-видавничий відділ.
ВНТУ, ГНК, к. 114.
Хмельницьке шосе, 95, м. Вінниця, 21021.
press.vntu.edu.ua;
email: rvv.vntu@gmail.com.
Свідоцтво суб'єкта видавничої справи
серія ДК № 3516 від 01.07.2009 р.