Міністерство освіти і науки України
Вінницький національний технічний університет

Б. І. Мокін, В. Б. Мокін, О. Б. Мокін

НАВЧАЛЬНИЙ ПОСІБНИК
для опанування студентами способів розв'язання задач з функціонального аналізу мовою Python

Частина 1

Вінниця
ВНТУ
2022

УДК 517.98
М74

Рекомендовано до друку Вченою радою Вінницького національного технічного університету Міністерства освіти і науки України як навчальний посібник для студентів закладів вищої освіти, що спеціалізуються в галузі інформаційних технологій
(протокол № 7 від «31» березня 2022 р.)

Рецензенти:
В. Я. Данилов, доктор технічних наук, професор (НТУУ «КПІ ім. Сікорського»)
В. І. Клочко, доктор педагогічних наук, професор (ВНТУ)
О. С. Макаренко, доктор фіз.-мат наук, професор (НТУУ «КПІ ім. Сікорського»)

Мокін, Б. І.

М 74 Навчальний посібник для опанування студентами способів розв'язання задач з функціонального аналізу мовою Python. Частина 1 / Б. І. Мокін, В. Б. Мокін, О. Б. Мокін. – Вінниця : ВНТУ, 2022. – 124 с.

ISBN 978-966-641-892-3

В навчальному посібнику викладено способи розв'язання задач з функціонального аналізу, адаптованого до прикладних проблем в галузі інформаційних технологій, у відповідності зі змістом однойменного навчального посібника цих же авторів, а також викладені основи програмування мовою Python і програми реалізації способів розв'язання даного класу задач цією мовою. Частина 1 охоплює задачі з теорії множин, метричних просторів, теорії міри, інтегралів Рімана, Стілтьєса та Лебега, а також задачі з дослідження функціоналів на екстремум.

Навчальний посібник рекомендується для студентів та аспірантів, що спеціалізуються в ІТ-галузі за спеціальностями 124 – «Системний аналіз» та 126 – «Інформаційні системи та технології»

УДК 517.98

ISBN 978-966-641-892-3

ЗМІСТ

ВСТУП

У 2020 році ми опублікували навчальний посібник «Функціональний аналіз, адаптований до прикладних задач в галузі інформаційних технологій» [1], з використанням якого студенти, що навчаються у ВНТУ за спеціальністю 124 – «Системний аналіз», опановують навчальну дисципліну «Функціональний аналіз», яка, згідно з освітньо-професійною програмою цієї спеціальності входить до переліку обов'язкових.

Уже після першого року використання в навчальному процесі нашого навчального посібника з функціонального аналізу ми зрозуміли, що студентам важко засвоювати практичні навички з розв'язання прикладних задач з цієї навчальної дисципліни, користуючись лише навчальним посібником, в якому викладені її теоретичні основи, особливо ж в умовах реалізації навчального процесу в онлайн-режимі, зумовленому карантинними обмеженнями щодо перебування студентів групами в аудиторіях, викликаними світовою пандемією коронавірусної хвороби. А тому в нашому авторському колективі виникла задумка написати з функціонального аналізу ще один навчальний посібник, в якому викласти практичні способи розв'язання задач з функціонального аналізу, характерних для ІТ-галузі, аби цей другий навчальний посібник доповнив перший практикою застосування теоретичних основ функціонального аналізу, викладених у першому навчальному посібнику, при розв'язанні прикладних задач, характерних для ІТ-галузі. Особливо важливим аспектом цього другого навчального посібника, на наш погляд, має стати доведення етапу розв'язання прикладних задач функціонального аналізу до застосування сучасної мови програмування Python, усі дистрибутиви якої викладені в інтернеті для вільного доступу у вигляді програмного середовища Anaconda [2]. Саме ця задумка і реалізована нами у навчальному посібнику, зміст якого подається нижче. І оскільки цей навчальний посібник покликаний доповнювати практикою попередній, присвячений теорії функціонального аналізу, то і розділи його збігаються за назвами і змістом з відповідними розділами свого теоретичного попередника.

Характерною особливістю нашого навчального посібника, зміст якого подається нижче, є те, що ми спочатку демонструємо способи і алгоритми «ручного» розв'язання прикладних задач функціонального аналізу на конкретних простих прикладах, а уже потім показуємо як ці задачі розв'язувати за допомогою мови Python з використанням дистрибутивів і підпрограм, що допускають їх комп'ютерну реалізацію в консолі Spyder.

Такий підхід обумовлений тим, що, як показала практика роботи зі студентами під час вивчення функціонального аналізу, в разі, якщо для розв'язання задачі використати готову программу, написану мовою Python, то студенти, використовуючи цю програму, отримують розв'язок задачі, не розуміючи суті закладених в програму алгоритмів, що, по-перше, надовго не залишається в пам'яті, а по-друге, не сприяє осмисленому застосуванню цих же алгоритмів при розв'язанні інших задач, умови яких в чомусь відрізняються.

Але, оскільки у цьому навчальному посібнику ми використовуватимемо мову програмування Python, то спочатку, з посиланням на популярний [3] та науковий [4] варіанти її викладення, здійснимо екскурс в історію створення цієї мови та в загальних рисах розглянемо основні дистрибутиви, що входять до її структури.

І почнемо зі звернення уваги на те, що розробники мови Python, метою яких було створення безкоштовної, доступної для всіх, мови програмування високого рівня, назвали цю мову не в честь змії і читати цю назву потрібно не «пітон», тобто не в прямому перекладі літер англійської мови, а читати назву мови потрібно «Пайтон», бо назвали розробники її в честь циркового коміка Пайтона, який створив популярне у 70-х роках минулого сторіччя як в Англії, так і у Британії в цілому, комедійне шоу «Летючий цирк Монті Пайтона», яке дуже подобалось розробникам нової мови програмування. Але, щоб задовольнити і тих, кому

подобалось вкладати в слово Python зміст слова «пітон», усі програми, які створювались цією мовою уже в 64-бітовому варіанті, їх розробники розмістили і продовжують розміщувати в програмному середовищі, названому на честь найдовшої у світі змії Anaconda.

Тим нашим читачам, яких цікавить лише використання програм мовою Python для розв'язання прикладних задач, ми радимо детально ознайомитись з навчальним посібником [4], в якому викладено основи мови Python та приклади її застосування для розв'язання наукових задач, а тим, кого цікавить і технологія створення програм мовою Python, варто ознайомитись ще й з навчальним посібником [3], в якому ця технологія у спрощеному вигляді з використанням інтегрованого 32-бітового середовища IDLE викладена так, щоб бути зрозумілою і дітям шкільного віку. Але одразу ж попереджаємо, що в пакет прикладних програм Anaconda, для яких використовується виключно 64-бітове середовище, 32-бітове середовище IDLE не входить, тож IDLE в Anakonda шукати не варто, але його можна викликати за окремим посиланням [5].

Ми згадали про інтегроване середовище IDLE для повноти історичної довідки, але користуватись ним у цьому навчальному посібнику не будемо, адже 32-бітове інтегроване програмне середовище, яким є IDLE, нині вже неактуальне, тому в подальшому на нього посилатись не будемо, а сконцентруємо увагу на 64-бітовому інтегрованому середовищі Anaconda, на яке ви можете зустріти посилання в різних літературних джерелах і як на дистрибутив, і як на пакет прикладних програм (ППП) мовою Python. Встановлювати на своєму комп'ютері дистрибутив Anaconda вигідно ще й тому, що одночасно і автоматично на вашому комп'ютері встановлюється і бібліотека пакетів прикладних програм, які використовуються для наукових досліджень і носять назви: numpy,sympy, scipy, matplotlib.

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 всіляких перетворень виразів, заданих у символьній формі (зокрема і для взяття 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 одно-, дво- і тривимірних графічних зображень результатів розрахунків, виконаних з застосуванням пакетів numpy, sympy та scipy, тобто, це графічний редактор, пристосований до роботи з програмами, написаними мовою Python.

Зміст усіх цих ППП та порядок їх застосування ми будемо розкривати в процесі їх використання для розв'язання задач функціонального аналізу, а у цій вступній частині навчального посібника згадаємо ще лише про те, що реалізовувати ці ППП в разі, якщо ви встановили на своєму комп'ютері 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 виклику в подальшому, 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.

На цьому ми вступну частину нашого навчального посібника закінчуємо і приступаємо до конкретизації його змісту.

Розділ 1 МНОЖИНИ І МЕТРИЧНІ ПРОСТОРИ ТА ЇХ ХАРАКТЕРИСТИКИ (В ПРИКЛАДАХ І ПРОГРАМАХ)

1.1 Множини та їх характеристики

У кінці однойменного підрозділу у базовому навчальному посібнику [1] серед інших стосовно множин сформульовані і такі запитання.

1. Який зміст вкладається в поняття «множина» та які основні характеристики множин і їх приклади ви знаєте?
2. Що собою являють сума, переріз і різниця множин?

Тож із відповідей на ці запитання ми і розпочнемо викладення змісту нашого «Навчального посібника для опанування студентами способів розв'язання задач з функціонального аналізу мовою Python»

Що стосується першого питання, то його розкриття почнемо з повторення означення множини та її характеристик, які дані в роботі [1].

Отже, ***множина*** - це сукупність об'єктів якоїсь природи, які прийнято називати елементами. ***Множина*** вважається заданою, якщо є відомими всі її елементи та правило, згідно з яким ці елементи відносять до цієї ***множини***. ***Множина*** зі скінченною кількістю елементів називається ***скінченною множиною,*** а ***множина*** з нескінченною кількістю елементів називається ***нескінченною множиною***. Прикладом ***скінченної множини*** є ***множина*** автомобілів, зареєстрованих в Україні, а прикладом ***нескінченної множини*** є ***множина*** дійсних чисел на відрізку [0,1] числової осі. ***Множина,*** яка не містить жодного елемента, називається ***пустою.*** Якщо ***множини*** A **і** B складаються з однакових елементів, то вони вважаються ***рівними*** між собою, про що свідчить запис $A = B$. Якщо ж у ***множину*** A входять не всі елементи ***множини*** B , то ***множину*** A називають ***підмножиною множини*** B, про що свідчить запис $A \subset B$. Наприклад, на числовій осі ***множина*** раціональних чисел R є ***підмножиною множини*** дійсних чисел Z, а ***множина*** автомобілів марки «Хонда» є ***підмножиною множини*** автомобілів, зареєстрованих в Україні.

Важливою характеристикою ***множин*** є їх ***еквівалентність***, згідно з якою ***множини*** A, B вважаються ***еквівалентними,*** якщо кожному елементу $a \in A$ за якимось, заздалегідь обумовленим правилом, ставиться у відповідність один-єдиний елемент $b \in B$, а кожному елементу $b \in B$ ставиться у відповідність один-єдиний елемент $a \in A$. Наприклад, еквівалентними є ***множина*** A легкових автомобілів приватної власності, кожний з яких зареєстровано лише на одного власника у певному населеному пункті, і ***множина*** B людей, які є власниками цих автомобілів. Правилом, за яким встановлюється ***еквівалентність*** цих множин, є запис органом реєстрації автомобілів прізвища власника в паспорті автомобіля.

А для того, щоб порівнювати між собою ***нееквівалентні множини***, вводиться поняття їх ***потужності***, яку будемо встановлювати за чимось спільним, що має місце в усіх ***множинах, еквівалентних*** тій, яку ми розглядаємо. Очевидно, що спільним у ***скінченних множинах*** різної природи є лише кількість їх елементів, а тому, якщо ***множина*** A ***має*** n елементів, а ***множина*** B ***має*** m елементів і при цьому $n > m$, то ми констатуємо, що ***множина*** A ***має потужність більшу, ніж множина*** B**.** Але виникає запитання: «А як порівнювати між собою за ***потужністю нескінченні множини***, кожна з яких має нескінченну кількість елементів?».

Досліджуючи на числовій осі різні нескінченні числові послідовності, математики встановили, що з усіх нескінченних послідовностей найшвидше наближається до нескінченності ***натуральний ряд*** N, оскільки кожне його наступне число дорівнює попередньому, збільшеному на одиницю, і при цьому при формуванні цього ряду

пропускаються всі дійсні числа, які містяться на числовій осі у кожній такій одиниці. А тому домовились вважати ***натуральний ряд***, який являє собою нескінченний ряд чисел, ***нескінченною множиною найменшої потужності***, яку домовились позначати символічно малою латинською літерою ***a,*** і всі інші ***нескінченні множини*** порівнювати між собою, виходячи з того, як вони співвідносяться за ***потужністю*** з ***потужністю натурального ряду***, а всі ***нескінченні множини*** з ***потужністю натурального ряду*** називати ***зліченними множинами***, оскільки кожному їх елементу можна приписати індекс, який дорівнює відповідному числу натурального ряду, за рахунок чого кожен їх елемент можна ***полічити.***

І першим фактом, який встановили математики після цієї домовленості стосовно ***потужності натурального ряду***, є те, що ***потужність множини Z дійсних чисел, заданих на відрізку [0,1], є більшою за a .***

Потужність нескінченної множини дійсних чисел на відрізку [0,1] математики ***вирішили назвати потужністю континууму***, а символічно ***позначати малою латинською літерою c,*** отже, справедливою є нерівність **c > a.** Більше того, математики встановили, що ***для потужностей*** **c, a** ***справедливою є рівність*** $c = 2^a$. А оскільки ***множина дійсних чисел Z на відрізку [0, 1]*** числової осі, є сумою ***підмножини R раціональних чисел та підмножини ірраціональних чисел***, заданих на цьому ж відрізку, а кожне раціональне число можна подати у вигляді відношення двох цілих чисел, які є елементами натурального ряду, який є зліченною множиною потужності ***a*** , то ці числа і в чисельнику, і в знаменнику можна полічити, а тому ***підмножина раціональних чисел на відрізку [0, 1] числової осі теж є зліченною множиною потужності a .*** А з цього факту витікає ***два наслідки***, ***перший*** з яких засвідчує, що ***підмножина ірраціональних чисел на вказаному відрізку є нескінченною множиною потужності континууму c ,*** бо лише за рахунок цієї підмножини множина дійсних чисел на вказаному відрізку матиме потужність **c,** а ***другим наслідком є твердження, що в разі додавання до множини потужності континууму будь-якої зліченної підмножини потужність їх суми залишається рівною c*** . Тож на основі цього твердження можна зробити ***важливий висновок, що і вся вісь дійсних чисел є множиною потужності континууму c .***

А тепер у такому ж сконцентрованому варіанті, скориставшись теоретичним матеріалом, наведеним у нашому навчальному посібнику [1], дамо відповідь на друге з наведених вище запитань.

При об'єднанні двох ***множин*** *A* і *B* утворюється нова ***множина*** *M*, яку називають їх ***сумою*** і яка містить у собі всі елементи обох цих ***множин***, причому кожен однаковий елемент обох ***множин*** в їх ***суму*** *M* входить як один елемент, символічно записується це так:

$$M = A \cup B. \tag{1.1}$$

Наприклад, якщо *A* і *B* – це числові ***множини***,

де $$A = \{1,2,3,4,5\},\ B = \{4,5,6,7,8\}, \tag{1.2}$$

то, згідно з (1.1), матимемо

$$M = A \cup B = \{1,2,3,4,5\} \cup \{4,5,6,7,8\} = \{1,2,3,4,5,6,7,8\} \tag{1.3}$$

При перетині двох ***множин*** *A* і *B* утворюється нова ***множина*** *P*, яку називають їх ***перерізом*** і яка містить у собі лише ті елементи обох цих ***множин***, які є однаковими, причому кожен із цих однакових елементів обох ***множин*** в їх ***переріз*** *P* входить як один елемент, символічно записується це так:

$$P = A \cap B. \tag{1.4}$$

Для наведених в умовах попереднього прикладу числових ***множин*** (1.2), згідно з (1.4), матимемо

$$P = A \cap B = \{1,2,3,4,5\} \cap \{4,5,6,7,8\} = \{4,5\}. \tag{1.5}$$

А множина Q, яка складається з елементів ***множини*** A, що не входять у ***множину*** B, називається ***різницею*** цих ***множин*** і позначається як $A - B$ або $A \setminus B$, тобто,

$$Q = A - B = A \setminus B. \tag{1.6}$$

Цілком очевидно, що в загальному випадку

$$\mathrm{A} - \mathrm{B} \neq \mathrm{B} - \mathrm{A}. \tag{1.7}$$

Наприклад, для числових ***множин*** (1.2)

$$A - B = \{1,2,3\}, \tag{1.8}$$

$$B - A = \{6,7,8\}. \tag{1.9}$$

На завершення цього підрозділу розглянемо операцію над множинами, яку, як правило, не розглядають у класичному функціональному аналізі, а тому їй не знайшлося місця і у нашому навчальному посібнику [1], але яка теж може зустрітись в задачах системного аналізу, що використовують операції над множинами. Це ***операція об'єднання множин А та В, яку позначимо R, з вилученням спільних елементів***, тобто операція

$$R = \overline{A} \cup \overline{B}, \tag{1.10}$$

за умови, що

$$A \subset S, \quad B \subset S, \quad A \cup B = S. \tag{1.11}$$

Виходячи з цих умов, для числових ***множин*** (1.2) матимемо

$$R = \overline{A} \cup \overline{B} = \{8,7,6,3,2,1\} \tag{1.12}$$

1.2 Початкові відомості з мови програмування Python, достатні для розв'язання задач, пов'язаних з операціями над множинами

Для того, щоб продемонструвати, як наведені у попередньому підрозділі операції над множинами виконувати ***в програмному середовищі Python***, спочатку, орієнтуючись на наведені в списку використаної літератури джерела [2]–[5], нагадаємо деякі початкові відомості з технології використання цієї мови програмування.

Отже, виходимо з того, що ви вже викликали на свій комп'ютер ***дистрибутив Anaconda*** і вибрали та встановили ***консоль Spyder***. В результаті цих дій ви вже матимете на своєму комп'ютері ***інтерактивну оболонку IPython***, в якій уже можна виконувати певні операції мовою Python і без виклику спеціалізованих ***пакетів*** програм типу: ***numpy, sympy, scipy, matplotlib.***

Після встановлення ***консолі Spyder екран*** вашого комп'ютера буде розділений ***на три частини, в лівій*** з яких ви зможете ***набирати*** потрібні вам програми ***мовою Python*** у вигляді цілісних ***файлів***, які, надавши їм назву і занісши в пам'ять, можна буде викликати з пам'яті за потреби їх використання. ***В правій нижній частині*** екранного ***вікна консолі Spyder*** ви зможете набирати потрібні вам ***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 потребу їх виведення на екран.

В пояснювальному прикладі № 1, поданому нижче, продемонстровано, як набирати команди в командних рядках для виконання операцій ***множення*** з символом «*», ***піднесення до степеня*** з символом «**», ***додавання*** з символом «+», ***віднімання*** 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.*** В разі ж, якщо ви в командному рядку записали щось таке, що не відповідає позначенням чи змісту, прийнятному для мови Python, то в наступному рядку після даного, з'явиться англійською мовою текст роз'яснення суті допущеної вами помилки. Виправивши цю помилку, ви можете продовжити набирання подальших команд вашої програми. Звертаємо увагу на те, що, якщо в командному рядку подається лише одна вихідна величина з конкретизованим її значенням, то після неї не ставляться ніякі інші знаки, а якщо в одному командному рядку подається кілька вихідних величин з конкретизованими значеннями, то між ними ставиться знак «;» (кома з крапкою). Цей же знак ставиться і між символьно записаною операцією в командному рядку і символом вказаного у цьому ж рядку результату цієї операції. ***Звертаємо увагу*** також і на те, що, як і у роботі [4], при формуванні команд у кожній програмі мовою Python, 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.

Пояснювальний приклад № 1

```
In[1]: x=2                          # Внесення змінної x та її значення
In[2]: y=4                          # Внесення змінної y та її значення
In[3]: z=x*y; z                     # Отримання добутку xy змінних
Out[3]: 8
In[4]: z1=x**y; z1                  # Піднесення змінної x до степеня y
Out[4]: 16
In[5]: z2=x**y-10; z2               # Обчислення виразу зі степенем і різницею
Out[5]: 6
In[6]: z3=_*2; z3                   # Перемноження на 2 попереднього числа
Out[6]: 12
In[7]: z4=__*3; z4                  # Перемноження на 3 числа з рядка [5]
Out[7]: 18
In[8]: z5=y**0.5; z5                # Обчислення кореня квадратного з y
Out[8]: 2.0
In[9]: z6=(x+y)*2; z6               # Перемноження на 2 суми x та y
Out[9]: 12
In[10]: z7=_/3; z7                  # Ділення на 3 попереднього числа
Out[10]: 4
```

Кінець пояснювального прикладу № 1.

Оскільки жодна програма не може виконуватись, якщо для запрограмованої задачі не задані вихідні дані, то зупинимось коротко на тому, яким чином задаються ***вихідні дані у програмах мовою Python.***

У цій мові вихідні дані ***можуть задаватись 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 (float), комплексні (complex) або цілі (int) числа, а також логічні змінні (bool) зі змістом «Правда» (True), яка в числовому еквіваленті означає «1», та зі змістом «Неправда» (False), яка в числовому еквіваленті означає «0».***

Стрічки (str) – наголошую, для програм мовою Python саме стрічки, а не рядки – ***утворюються взяттям їх елементів в одинарні (типу апострофа) або подвійні лапки***, як показано в пояснювальному прикладі № 2. Але, якщо ви наберете, наприклад, стрічку "Добрий день!" в рядку під номером [m], то після натискання клавіші Enter в наступному рядку під цим же номером буде надруковано Out[m]: 'Добрий день!' – тобто стрічка буде повторена, але з тим же номером і в одинарних лапках. Якщо ж ви хочете в наступному рядку вивести стрічку «в чистому вигляді», то її треба ***виводити на екран через операцію print( )*** – обидва ці варіанти виведення стрічки наведені в пояснювальному прикладі № 2 як з використанням одинарних, так і подвійних лапок. Відзначимо також, що ***індексування елементів в стрічці починається з нуля***, а за індексом елемента в стрічці, взятим в квадратні дужки, можна викликати його значення; що елементи в стрічці не дозволяється змінювати; що операцією «in» можна перевірити, чи даний елемент є в стрічці; що операцією «+» дві стрічки можна об'єднувати в одну (конкатенація); що за допомогою операції «*» 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; що ***за допомогою функції len( ) можна підрахувати, скільки елементів входить в стрічку*** – усі ці особливості стрічок теж наведені в пояснювальному прикладі № 2.

Пояснювальний приклад № 2

```
In [1]: "Добрий день!"                  # Внесення стрічки у формі ***“.”***
Out[1]: 'Добрий день!'
In [2]: 'Добрий день!'                  # Внесення стрічки у формі ***‘ ‘***
Out[2]: 'Добрий день!'
In [3]: print("Добрий день!")           # Виведення чистої стрічки на екран
Добрий день!
In [4]: print('Добрий день!')          # Виведення чистої стрічки на екран
Добрий день!
In [5]: s1="Добрий день,"               # Внесення стрічки у формі ***s1***
In [6]: s2='шановні студенти!'          # Внесення стрічки  у формі ***s2***
In [7]: s1+s2                           # Формування ***суми*** стрічок ***s1*** та ***s2***
Out[7]: 'Добрий день, шановні студенти!'
In [8]: s3="Ура!"                       # Внесення стрічки у формі ***s3***
In [9]: 4*s3                            # Стрічка з ***чотирьох s3***
Out[9]: 'Ура!Ура!Ура!Ура!'
In [10]: "р" in s3                      # З'ясування ***чи є*** літера «***р***» в ***s3***
Out[10]: True
In [11]: "к" in s2                      # З'ясування ***чи є*** літера «***к***» в ***s3***
Out[11]: False
In [12]: s1[0]                          # Виклик ***із s1*** елемента ***з індексом 0***
Out[12]: 'Д'
In [13]: s1[2]                          # Виклик ***із s1*** елемента ***з індексом 2***
Out[13]: 'б'
In [14]: len(s1)                        # Визначення ***кількості*** членів ***в s1***
Out[14]: 12
```

Кінець пояснювального прикладу № 2.

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=[1,2,7,4]. 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[2]=5 вище наведений список L1 набуває вигляду L1=[1,2,5,4]. ***Конкретизація***, як уже сказаного ***про списки***, так і того, що буде сказано нижче, наведена ***у пояснювальному прикладі № 3***. Коротші списки можуть використовуватись як елементи більш довгих списків, наприклад L2=[[1,2,3], "бик", ['a', 'b', 'c']]. Окремі елементи списку можна вилучати за допомогою операції зрізу «:», зліва від якої в квадратних дужках записується індекс першого елемента, що переходить із базового списку в зрізаний, а з правого боку записується індекс останнього елемента, що уже не переходить із базового списку в зрізаний, наприклад, операція L1[1:3] трансформує наведений вище список L1 в новий список L1=[2,5] під тим же символом. Як і стрічки, за допомогою оператора «+» списки можна об'єднувати, а за допомогою оператора «*» 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. Дуже важливим є те, що ***застосуванням функції list( ) можна будь-яку послідовність***, записану як аргумент цієї функції (в круглих дужках), ***перетворити на список***. Наприклад стрічку s = 'День перший' функцією L=list(s) можна перетворити у список L = ['Д', 'е', 'н', 'ь',' ', 'п', 'е', 'р', 'ш', 'и', 'й']. Підтвердження того, що в два списки входять одні і ті ж елементи, можна отримати за допомогою 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 саме у цьому розділі нашого навчального посібника з функціонального аналізу.

Пояснювальний приклад № 3

```
In [1]: L2=[5,6,7,8]                          # Внесення списку L2
In [2]: L3=[9,10,11,12,13]                    # Внесення списку L3
In [3]: L4=L2+L3; L4                          # Формування списку L4
Out[3]: [5, 6, 7, 8, 9, 10, 11, 12, 13]       … з суми списків L2 та L3
In [4]: L4[2]=1;L4                            # Заміна в L4 елемента з
Out[4]: [5, 6, 1, 8, 9, 10, 11, 12, 13]       ….індексом 2 зі значення 7 на 1
In [5]: L2*3                                  # Повторення списку L2 тричі
Out[5]: [5, 6, 7, 8, 5, 6, 7, 8, 5, 6, 7, 8]
In [6]: L5=L4[2:5];L5                         # Формування з L4 списку з
Out[6]: [1, 8, 9]                               елементів з індексами 2,3,4
In [7]: list('Добрий день!')                  # Формування списку зі стрічки
Out[7]: ['Д', 'о', 'б', 'р', 'и', 'й', ' ', 'д', 'е', 'н', 'ь', '!']
In [8]: 10 in L5                              # Чи є елемент 10 в списку L5?
Out[8]: False
In [9]: 8 in L5                               # Чи є елемент 8 в списку L5?
Out[9]: True
In [10]: len(L4)                              # Скільки елементів в списку L4?
Out[10]: 9
In [11]: len(L5)                              # Скільки елементів в списку L5?
Out[11]: 3
```

```
In [12]: L6=[1,8,9]                    # Внесення списку L6
In [13]: L4==L6                        # Перевірка списків L4 та L6
Out[13]: False                         ….на рівність
In [14]: L5==L6                        # Перевірка списків L5 та L6
Out[14]: True                          ….на рівність
```

Кінець пояснювального прикладу № 3.

Якщо ті ж елементи послідовності, які, бувши оточеними квадратними дужками, задають список L=[1,2,3,4,5,6,7], ***взяти замість квадратних у круглі дужки***, **то отримаємо кортеж** K=(1,2,3,4,5,6,7) із елементів тієї ж послідовності, але жоден із елементів якої тепер уже не можна змінювати. А тому можна стверджувати, що ***стосовно кортежів справедливо усе те, що ми раніше вже визначили для списків, за винятком того, 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[1], отримати елемент 2, але присвоїти нове значення, наприклад 9, елементу кортежу з індексом [1] за допомогою оператора присвоєння K[1]=9, не вдасться, бо програма мовою Python у цьому випадку видасть повідомлення про помилку, як це показано у пояснювальному прикладі № 4, наведеному нижче. Однак, якщо нам у потрібному кортежі якийсь елемент поміняти необхідно, то це усе-таки зробити можна, але попередньо трансформувавши ***кортеж у список за допомогою функції list(K)***

Пояснювальний приклад № 4

```
In [1]: K=(1,2,3,4,5,6,7)              # Внесення кортежу K
In [2]: K[1]                           # Виклик із кортежу K елемента
Out[2]: 2                              ….з індексом 1
In [3]: K[3]                           # Виклик із кортежу K елемента
Out[3]: 4                              ….з індексом 3
In [4]: K1=(8,9,10)                    # Внесення кортежу K1
In [5]: K+K1                           # Отримання суми кортежів K та K1
Out[5]: (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
In [6]: 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
In [7]: L=list(K); L                   # Трансформація кортежу в список L
Out[7]: [1, 2, 3, 4, 5, 6, 7]
In [8]: L[3]=15; L                     # Зміна в списку L значення елемента
Out[8]: [1, 2, 3, 15, 5, 6, 7]         ….з індексом 3 з числа 4 на 15
```

Кінець пояснювального прикладу № 4.

В списках і кортежах доступ до елемента з множини елементів, заданих списком чи кортежем, здійснюється з використанням взятого у квадратні дужки індексу цього елемента, який для першого елемента цієї множини задається нульовим і зростає на одиницю для кожного наступного. Але ***для програм мовою Python можуть використовуватись і множини елементів, доступ до кожного з яких здійснюється з використанням не індексу, а (взятого теж у квадратні дужки) ключа,*** заздалегідь заданого лише для цього елемента. ***Такі множини*** елементів ***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, вираз d1=dict(a=1, b=2,c=3) задає словник d1 зі значеннями елементів 1,2,3 та ключами до цих елементів, заданих літерами a,b,c. А друкує

цей словник за викликом d1 програма у вигляді {'a':1, 'b':2, 'c':3}. Звертаємо увагу, що ***при роздрукуванні виразу для словника використовуються фігурні дужки***, у той час як при роздрукуванні списку використовувалися квадратні дужки, а при роздрукуванні кортежу – дужки круглі. Якщо в аргументних дужках функції dict() не стоїть нічого, то ця функція утворює пустий словник d0, який програма роздруковує як {} і який можна заповнювати поелементно за допомогою операторів присвоювання ключам значень відповідних елементів, наприклад, оператором d['a']=1 в пустий словник вноситься елемент зі значенням 1 та ключем до нього у вигляді 'a', після чого пустий словник {} перетворюється у словник {'a':1}. 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':1, 'b':2}. Цей ***процес, як і інші, що присвячені роботі зі словниками, продемонстровано на пояснювальному прикладі № 5***, поданому нижче. Аргументами функції dict( ) можуть бути також список списків, наприклад d1=dict([['a',1], ['b',2], ['c',3]], або список кортежів, наприклад d1=dict([('a',1),('b',2),('c',3)] – і у обох цих випадках программа роздрукує вам один і той же словник у вигляді {'a':1, 'b':2, 'c':3}. Щоб прочитати елемент словника, наприклад d1, потрібно сформувати оператор виклику у вигляді назви цього словника з поміщеним поряд у квадратних дужках ключем цього елемента – наприклад, якщо в програму в рядок під номером [m] внести вираз d1['b'], то після натискання клавіші Enter отримаєте під цим рядком елемент 2. Оператором «in» у вигляді, наприклад 'a' in d1, внесеному в командний рядок програми, ви можете перевірити чи є елемент з ключем 'a' в словнику d1. Якщо він є, то після натискання клавіші Enter під цим командним рядком ви побачите напис True (правда), а якщо елемента з таким ключем у словнику немає, то ви побачите напис False (неправда). Оператором присвоювання ви можете не лише вносити до словника елементи, яких у ньому ще немає, але можете змінювати ті елементи, для яких цей ключ визначено, нприклад, набравши у командному рядку d1['a']=10; d1['e']=4; d1 та натиснувши клавішу Enter, ви під цим рядком замість словника {'a':1, 'b':2, 'c':3}, побачите словник {'a':10, 'b':2, 'c':3, 'e':4}. А за допомогою оператора del ви можете видаляти елементи зі ловника, наприклад, набравши в командному рядку вираз del d1['c'] і натиснувши клавішу Enter, ви в наступному рядку побачите не той варіант словника d1, який розміщено вище, а словник d1 у варіанті {'a':10, 'b':2, 'e':4}. Функція len( ), як і для словників та кортежів, визначає вам кількість елементів (або, що кількісно одне і те саме, ключів) в словнику. Наприклад, якщо ви у командному рядку наберете len(d1), то після натискання на клавішу Enter ви під цим рядком прочитаєте число 3, оскільки програма буде підраховувати число елементів (ключів) в останньому вашому варіанті словника d1. А тепер ***зупинимось на методі keys( ),*** який впорядковує словник за ключами, через що впорядковані у такий спосіб словники d1 та d2, тобто ці ***словники у варіанті d1.keys( ) та d2.keys( ) можна використовувати для виконання таких операцій на множинах***, як

об'єднання (сума) множин

```
D3=d1.keys( ) | d2.keys( ),
```
(1.13)

різниця множин

```
D4=d1.keys( ) - d2.keys( ),
```
(1.14)

переріз (перетин) множин

```
D5=d1.keys( ) & d2.keys( ),
```
(1.15)

об'єднання множин з вилученням спільних елементів

```
D6=d1.keys( ) ^ d2.keys( ).
```
(1.16)

А якщо вам потрібно отримати список ключів, то це можна здійснити, застосувавши функцію k=list(d.keys( )). Усе, що викладено вище у тексті, присвяченому формуванню та використанню словників, продемонстровано у пояснювальному прикладі № 5, показаному нижче, в якому для концентрації уваги є і спеціально зроблені помилки.

Пояснювальний приклад № 5

```
In [1]: d1=dict(a=1, b=2,c=3); d1          # Безпосереднє формування словника d1
Out[1]: {'a': 1, 'b': 2, 'c': 3}
In [2]: d1=dict([['a',1], ['b',2],['c',3]]); d1   # Формування словника d1 із списків
Out[2]: {'a': 1, 'b': 2, 'c': 3}
In [3]: d1=dict([('a',1), ('b',2),('c',3)]); d1   # Формування словника d1 із кортежів
Out[3]: {'a': 1, 'b': 2, 'c': 3}
In [4]: d1={"a":1,"b":2,"c":3}; d1         # Внесення словника d1
Out[4]: {'a': 1, 'b': 2, 'c': 3}
In [5]: d2={}                              # Формування пустого словника d2
In [6]: d2['a']=1; d2['b']=2;              # Внесення в пустий словник d2
d2['e']=4;d2['l']=5; d2                    ….елементів з їх ключами
Out[6]: {'a': 1, 'b': 2, 'e': 4, 'l': 5}
In [7]: d1['a']                            # Виклик із словника d1 елемента за ключем
Out[7]: 1
In [8]: d2['e'],d2['l']                    # Виклик із словника d2 двох елементів
Out[8]: (4, 5)                             ….за їх ключами
In [9]: 'a' in d1                          # Перевірка наявності ключа ‘a’
Out[9]: True                               … в словнику d1
In [10]: 'e' in d1                         # Перевірка наявності ключа ‘e’
Out[10]: False                             ….в словнику d1
In [11]: 'e' in d2                         # Перевірка наявності ключа ‘e’
Out[11]: True                              ….в словнику d2
In [12]: 'm' in d2                         # Перевірка наявності ключа ‘m’
Out[12]: False                             …. в словнику d2
In [13]: k=list(d1.keys()); k              # Формування списку ключів, присутніх
Out[13]: ['a', 'b', 'c']                   ….у словнику d1
In [14]: d1.keys() | d2.keys()             # Реалізація операції об’єднання множин,
Out[14]: {'a', 'b', 'c', 'e', 'l'}         ….заданих словниками d1 та d2
In [15]: d1.keys() - d2.keys()             # Реалізація операції різниці A-B множин,
Out[15]: {'c'}                             …заданих словниками d1 та d2
In [16]: d2.keys() - d1.keys()             # Реалізація операції різниці B-A множин,
Out[16]: {'e', 'l'}                        …. заданих словниками d2 та d1
In [17]: d1.keys() & d2.keys()             # Реалізація перетину множин A,B,
Out[17]: {'a', 'b'}                        …. заданих словниками d1 та d2
In [18]: d1.keys() ^ d2.keys()             # Реалізація операції об’єднання множин
Out[18]: {'c', 'e', 'l'}                   ….з вилученням однакових елементів
In [19]: len(d1)                           # Визначення потужності множини A,
Out[19]: 3                                 ….заданої словником d1
In [20]: len(d2)                           # Визначення потужності множини B,
Out[20]: 4                                 ….заданої словником d2
In [21]: del d2['b']                       # Видалення із d2 елемента з ключем ‘b’
In [22]: d2                                # Словник d2 після видалення елемента з
Out[22]: {'a': 1, 'e': 4, 'l': 5}          ….ключем ‘b’
```

Кінець пояснювального прикладу № 5

А на завершення цього вступного для програмування мовою Python матеріалу ***розглянемо ще оператори порівняння***, окремі з яких вам знадобляться уже на стадії розв'язання найпростіших задач функціонального аналізу, пов'язаних з множинами.

Отже, ***до операторів порівняння***, які використовуються в булевих операціях формальної логіки, ***відносять: оператор «==»,*** який означає «дорівнює»; ***оператор «!=»,*** який означає «***не дорівнює»; оператор «<»,*** який означає «***менше***»; ***оператор «<=»,*** який означає «***менше або дорівнює***»; ***оператор «>»,*** який означає «***більше***»; ***оператор «>=»,*** який означає «***більше або дорівнює***»; ***оператор «and***», який означає «***і те, і те***»; ***оператор «or***», який означає «***або те, або те***»; ***оператор «not***», який означає «***не те***» А результатом застосування операторів порівняння завжди є або ***булева змінна «True***», яка означає «***Правда***», або ***булева змінна «False***», яка означає «***Неправда»***. В числовому еквіваленті в арифметичних операціях, які мають перевагу перед логічними, ***булевій змінній «True» відповідає число «1», а булевій змінній «False» відповідає число «0»***. 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 легко бачити з пояснювального прикладу № 6,*** розміщеного нижче.

Пояснювальний приклад № 6

```
In [1]: x=10                          # Внесення змінної x та її значення
In [2]: x>12                          # Перевірка умови чи змінна x є більшою порогу
Out[2]: False
In [3]: x<12                          # Перевірка умови чи змінна x є меншою порогу
Out[3]: True
In [4]: 8<x<12                        # Перевірка умови чи змінна x є в інтервалі
Out[4]: True
In [5]: y=15                          # Внесення змінної y та її значення
In [6]: x<y                           # Перевірка умови чи змінна x є меншою y
Out[6]: True
In [7]: (8<x<12)and(y>10)             # Перевірка одночасного виконання двох умов
Out[7]: True
In [8]: (x>10)and(y>10)               # Перевірка одночасного виконання двох умов
Out[8]: False
In [9]: z=not((x>10)and(y>10)); z     # Формування логічної змінної запереченням
Out[9]: True
In [10]: z or (x>10)                  # Перевірка виконання умови АБО
Out[10]: True
In [11]: c=5                          # Внесення константи та її значення
In [12]: c+(8<x<12)                   # Знаходження арифметичної суми константи
Out[12]: 6                            ….та логічної змінної
In [13]: g=(x*(y>10)+(c+              # Визначення результату
(8<x<12))); g                         ….логіко-арифметичної
Out[13]: 16                           ….операції з множенням і додаванням
In [14]: (x*(y>10)+(c+                # Визначення результату
(8<x<12)))**0.5                       ….логіко-арифметичної
Out[14]: 4.0                          ….операції, в якій є ще й піднесення до степеня
```

Кінець пояснювального прикладу № 6

На цьому ми завершимо викладення того мінімуму знань з формування та використання стрічок, списків, кортежів і словників та логічних операцій, які нам потрібні будуть при розв'язанні вже сформульованих задач

1.3 Задачі з операціями над множинами в програмах мовою Python

Покажемо, як, використовуючи матеріал, викладений у двох попередніх підрозділах цього розділу, розв'язувати задачі з операціями над множинами, використовуючи програми мовою Python, а саме: як ***порівнювати множини***, як ***визначати потужність множин***, як ***перевіряти на еквівалентність*** – це буде реалізовувати ***програма 1***, а також як ***знаходити їх суму та об'єднання*** з вилученням спільних елементів, ***як визначати різницю та переріз множин*** – це буде реалізовувати ***програма 2***. В обох програмах після символа «#» записане роз'яснення змісту дії, що виконується програмою в даному командному рядку, що в саму програму не вноситься, а символом «\» (зворотний слеш), який в програму вноситься, здійснюється перенесення частини рядка в рядок наступний.

Програма мовою Python для перевірки множин на рівність, визначення їх потужності та перевірки на еквівалентність

(Програма 1)

```
In [1]: A={1,2,3,4,5}                          # Внесення множини A в оболонку IPython
In [2]: B={4,5,6,7,8}                          # Внесення множини B в оболонку IPython
In [3]: LA=list(A); LA                         # Трансформація множини A в список LA
Out[3]: [1, 2, 3, 4, 5]
In [4]: LB=list(B); LB                         # Трансформація множини B в список LB
Out[4]: [4, 5, 6, 7, 8]
In [5]: LA==LB                                 # Перевірка множин A і B на рівність
Out[5]: False
In [6]: len(LA)                                # Визначення потужності множини A
Out[6]: 5
In [7]: len(LB)                                # Визначення потужності множини B
Out[7]: 5
In [8]: len(LA)==len(LB)                       # Перевірка множин A і B на еквівалентність
Out[8]: True
In [9]: 'Множини A і B не є рівними, але\
 вони мають однакову потужність,\
а тому є еквівалентними'                       # Формування стрічки висновку
Out[9]: 'Множини A і B не є рівними, але\
 вони мають однакову потужність, а тому\
 є еквівалентними'
```

Кінець програми 1.

Програма мовою Python для знаходження суми множин та їх об'єднання з вилученням спільних елементів, а також для визначення різниці та перерізу множин

(Програма 2)

```
In [1]: dLA = {}                               # Формування пустого словника {} множини A
In [2]: dLA['a']=1                             # Внесення в словник dLA елемента з ключем 'a'
In [3]: dLA['b']=2                             # Внесення в словник dLA елемента з ключем 'b'
In [4]: dLA['c']=3                             # Внесення в словник dLA елемента з ключем 'c'
In [5]: dLA['e']=4                             # Внесення в словник dLA елемента з ключем 'e'
In [6]: dLA['h']=5                             # Внесення в словник dLA елемента з ключем 'h'
In [7]: dLA                                    # Формування словника dLA для множини A
Out[7]: {'a': 1, 'b': 2, 'c': 3, 'e': 4, 'h': 5}
```

```
In [8]: dLB={}                          # Формування пустого словника {} множини B
In [9]: dLB['e']=4;dLB['h']=5;\
dLB['p']=6;dLB['q']=7;.dLB['r']=8       # Внесення в словник dLB ключів до усіх елементів
In [10]: dLB                            # Формування словника dLB для множини B
Out[10]: {'e': 4, 'h': 5, 'p': 6, 'q': 7, 'r': 8}
In [11]: dLA.keys() | dLB.keys()        # Операція (1.13) для суми множин (1.1)
Out[11]: {'a', 'b', 'c', 'e', 'h', 'p', 'q', 'r'}
In [12]: dLA.keys() - dLB.keys()        # Операція (1.14) для різниці множин A-B
Out[12]: {'a', 'b', 'c'}
In [13]: dLB.keys() - dLA.keys()        # Операція (1.14) для різниці множин B-A
Out[13]: {'p', 'q', 'r'}
In [14]: dLA.keys() & dLB.keys()       # Операція (1.15) для перерізу множин (1.4)
Out[14]: {'e', 'h'}
In [15]: dLA.keys() ^ dLB.keys()        # Операція (1.16) для суми з вилученням (1.10)
Out[15]: {'a', 'b', 'c', 'p', 'q', 'r'}
```

Кінець програми 2.

Примітка. Результати операцій з множинами, трансформованими у словники, як бачимо в програмі 2, виводяться у вигляді словників, заданих лише ключами елементів, тож, якщо нас цікавить значення якогось конкретного елемента, то його потрібно викликати з результату операції командою виклику елемента, зміст якої викладено ще у пояснювальному прикладі 2, але з викликом не за індексом елемента, як у цьому прикладі, а за ключем.

1.4. Метричні простори та їх характеристики

У кінці однойменного підрозділу у базовому навчальному посібнику [1] серед інших стосовно метричних просторів сформульовані і такі питання:

1. Що собою являє метричний простір?
2. Що таке метрика простору і які умови вона має задовольняти? Наведіть приклади метрик.
3. Як визначається лінійний метричний простір?
4. Що таке норма простору? Наведіть приклади норм.
5. Дайте означення банахового простору.

Тож із відповідей на ці запитання, дотримуючись означення метричного простору та його характеристик, які дані в роботі [1], ми і продовжимо викладення змісту нашого «Навчального посібника для опанування студентами способів розв'язання задач з функціонального аналізу мовою Python»

Отже, множина $\Omega = \{x, y, z, \ldots, u, v, \ldots\}$ елементів якоїсь природи називається ***метричним простором***, якщо кожній упорядкованій парі елементів $x,\ y \in \Omega$ поставлено у відповідність невід'ємне число $\rho\,(x, y)$, яке називають ***метрикою простору*** Ω, якщо це число задовольняє три аксіоми метрики:

1) аксіому тотожності

$$\rho\,(x, y) = 0 \tag{1.17}$$

тоді і лише тоді, коли $x = y$;

2) аксіому симетрії

$$\rho(x, y) = \rho(y, x); \tag{1.18}$$

3) аксіому трикутника

$$\rho(x, y) + \rho(y, z) \geq \rho(x, z). \tag{1.19}$$

Розглядаючи ці аксіоми бачимо, що ***метрика*** $\rho(x, y)$ ***простору*** Ω ***задає відстань між елементами*** x, y ***цього простору.***

Елементи метричного простору називають точками.

Приклади:

1. Для тривимірного, звичного для нас, евклідового простору E_3 відстань між точками $x = \{x_1, x_2, x_3\}$ і $y = \{y_1, y_2, y_3\}$ $(x, y \in E_3)$ визначається формулою

$$\rho(x, y) = \sqrt{\sum_{i=1}^{3} (x_i - y_i)^2}. \tag{1.20}$$

2. Для множини $C[0,1]$ неперервних функцій $x(t)$, $y(t)$, …, заданих на відрізку $t \in [0,1]$, відстань між елементами $x(t)$ і $y(t)$ задається формулою

$$\rho(x, y) = \max_t |x(t) - y(t)|. \tag{1.21}$$

Якщо X – довільний метричний простір, то кажуть, що послідовність

$$\{x_n\} \subset X \tag{1.22}$$

збігається до точки $x_0 \in X$, якщо при $n \to \infty$

$$\rho(x_n, x_0) \to 0, \tag{1.23}$$

або, в іншому записі,

$$\lim_{n\to\infty} x_n = x_0. \tag{1.24}$$

Про послідовність $\{x_n\}$, яка збігається до деякої точки x_0, кажуть, що вона є обмеженою.

Якщо множина M ***містить у собі і всі свої граничні точки, то вона є замкнутою***.

Якщо в метричному просторі X будь-яка послідовність $\{x_n\} \subset X$, що збігається в собі, збігається і до деякої граничної точки x_0, яка є елементом цього ж простору, тобто $x_0 \in X$, то ***цей простір*** X ***називають повним***.

Приклад повного простору: простір $C[0,1]$.

Приклад неповного простору: простір дійсних багаточленів

$$p_n(t) = a_n \cdot t^n + a_{n-1} \cdot t^{n-1} + \cdots + a_1 \cdot t + a_0, \tag{1.25}$$

визначених на відрізку $t \in [0,1]$ з метрикою

$$\rho(p_n, q_m) = \max_t |p_n(t) - q_m(t)|. \tag{1.26}$$

Для підтвердження нагадаємо, що послідовність $\{p_n(t)\}$, де

$$P_n(t) = \sum_{k=0}^{n} \frac{1}{k!} \cdot t^k, \tag{1.27}$$

що належить до класу (1.25), рівномірно збігається в собі, але не має границі у просторі багаточленів, оскільки її границею

$$\lim_{n\to\infty} P_n(t) = e^t \tag{1.28}$$

є трансцендентна функція e^t, яка не є точкою в просторі дійсних багаточленів.

Метричний простір X називається лінійним, якщо на ньому визначені операції додавання і множення на скаляр, котрі задовольняють такі умови:

1) $$x^* + x^{**} = x^{**} + x^*, \ \forall\, x^*, x^{**} \in X; \tag{1.29}$$

2) $$\left(x^* + x^{**}\right) + x^{***} = x^* + \left(x^{**} + x^{***}\right), \ \forall\, x^*, x^{**}, x^{***} \in X; \tag{1.30}$$

3) $$x + 0 = x, \ \forall\, x \in X, 0 \in X, \tag{1.31}$$

де елемент 0 є нулем множини X, а для $\forall\, x^* \in X$ знайдеться протилежний елемент $x^{**} \in X$ такий, що

4) $$x^* + x^{**} = 0, \tag{1.32}$$

5) $$1 \cdot x = x, \ \forall\, x \in X; \tag{1.33}$$

6) $$\alpha \cdot (\beta \cdot x) = (\alpha \cdot \beta) \cdot x, \ \forall\, x \in X \text{ і } \forall\, \alpha, \beta; \tag{1.34}$$

7) $$(\alpha + \beta) \cdot x = \alpha \cdot x + \beta \cdot x, \ \forall\, x \in X \text{ і } \forall\, \alpha, \beta; \tag{1.35}$$

8) $$\alpha \cdot \left(x^* + x^{**}\right) = \alpha \cdot x^* + \alpha \cdot x^{**}, \ \forall\, x^*, x^{**} \in X \text{ і } \forall\, \alpha. \tag{1.36}$$

Лінійний метричний простір X називається нормованим, якщо $\forall\, x \in X$ може бути поставлене у відповідність деяке невід'ємне число $\| x \|$, яке називається нормою і яке задовольняє такі умови:

1) $$\| x \| = 0 \text{ тоді і тільки тоді, коли } x = 0; \tag{1.37}$$

2) $$\| \alpha \cdot x \| = | \alpha | \cdot \| x \|, \ \alpha - \text{скаляр}; \tag{1.38}$$

3) $$\| x^* + x^{**} \| \le \| x^* \| + \| x^{**} \|, \ \forall\, x^*, x^{**} \in X. \tag{1.39}$$

Нескладно переконатись в тому, що норма $\| x \|$ – це відстань від елемента x до нульового елемента множини X.

Приклади норм:

1) для простору $C[0,1]$

$$\| x(t) \| = \max_{t\in[0,1]} | x(t) | \tag{1.40}$$

або

$$\| x(t) \| = \sup_{t\in[0,1]} | x(t) |; \tag{1.41}$$

2) для евклідового простору E_n розмірності n

$$\| x \| = \sqrt{\sum_{i=1}^{n} x_i^2}, \tag{1.42}$$

де $x = \{x_1, x_2, \ldots, x_n\}$, $x \in E_n$.

Зрозуміло, що для будь-якого лінійного нормованого простору X є справедливим співвідношення

$$\| x^* - x^{**} \| = \rho(x^*, x^{**}), \tag{1.43}$$

де $x^*, x^{**} \in X$.

Повний лінійний нормований простір називається банаховим (за прізвищем математика, який досліджував цей простір) і позначається як B-простір.

Зрозуміло, що простори $C[0,1]$ і E_n є банаховими.

Відзначимо, що норма в B-просторі може бути введена по-різному, аби лише відповідала умовам (1.37), (1.38), (1.39).

Наприклад, в просторі неперервних на відрізку $t \in [0,1]$ функцій $x(t)$ норму можна ввести не лише у вигляді (1.40), але й у вигляді

$$\| x \| = \int_0^1 |x(t)| \, dt . \tag{1.44}$$

Такий B-простір називають простором Лебега і позначають $L[0,1]$*, щоб відрізняти від простору* $C[0,1]$ *тих же функцій, але з нормою (1.40).*

Одразу ж відзначимо, що, оскільки в нинішніх ППП numpy, sympy, scipy стандартної операції для обчислення інтегралу від абсолютного значення функції ще не введено, то норму Лебега (1.44) ми обчислюватимемо з використанням виразу

$$\|x\| = \int_a^b |x(t)| dt \approx \sum_{i=1}^{n} |x(t_i)| \Delta_i = \Delta \sum_{i=1}^{n} |x_i| , \tag{1.45}$$

в якому -

$$\Delta_i = t_i - t_{i-1} = const = \Delta, \quad i = 1,2,...,n , \tag{1.46}$$

$$|x(t_i)| = |x_i| \tag{1.47}$$

1.5 Додаткові відомості з мови програмування Python, достатні для розв'язання задач, пов'язаних з метричними просторами

Аналізуючи вирази (1.20), (1.21), (1.25), (1.26), (1.40), (1.41), (1.42), (1.44), ви бачите, що обчислення метрик і норм для різних метричних просторів з використанням програм мовою Python вимагає знань з технології використання цієї мови, додаткових до тих, які уже викладені в підрозділі 1.2. Тож нижче у цьому підрозділі, орієнтуючись на наведені в списку використаної літератури джерела - і у першу чергу на роботи [3], [4], - додамо ці додаткові знання. І ці додаткові знання будуть сконцентрованими у першу чергу на розкритті суті понять: ***змінна, функція, оператор, атрибут, метод, цикл, масив, модуль, пакет***.

Отже, почнемо з поняття ***«змінна»***, ***яка є адресою (ярликом)*** того місця в пам'яті комп'ютера, в якому зберігаються потрібні вам 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 пропуски між буквами, цифрами чи словами, замість яких потрібно для цілісності цього імені використовувати нижнє підкреслення. Наприклад, іменем змінної можуть бути ярлики x, y2, z3, lodka, a2a4, b11c2, bila_corowa, але не можуть бути – 2y, 3z, lod ka, 2aa4, 1b1c2, bila corowa. Ті дані (зокрема і у вигляді виразів), які зберігаються у тому місці пам'яті комп'ютера, якому ви призначили ім'я, прив'язуються до імені змінної символом рівності «=», зліва від якого пишеться ім'я змінної, а справа пишеться число чи вираз, який зберігається у тому місці пам'яті комп'ютера, якому присвоєне це ім'я, наприклад, x=0, y2=10, z3=x*y2, lodka=100. Тож, якщо вам потрібно з пам'яті комп'ютера викликати якісь конкретні дані, які туди вже поміщені під якимось іменем, наприклад, x=0, то ви в потрібний командний рядок [m] програми записуєте In[m]: x, і ваша 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 чіткого усвідомлення, що в мові Python мають на увазі, використовуючи поняття ***«функція»***, процитуємо означення цього поняття, наведене в роботах [3], [4]. Отже, згідно з означенням цього поняття, даним в роботі [3], – цитуємо: «***Функція – команда в мові програмування, яка зазвичай є набором інструкцій для виконання якоїсь дії***». А згідно з означенням цього поняття, даним в роботі [4], – цитуємо мовою оригіналу: «***Функция – это участок кода, который можно вызвать из любого места программы по его имени».*** Порівнюючи ці означення, легко бачити, що вони доповнюють одне одного, характеризуючи це поняття з акцентуванням на різні аспекти, що доповнюють один одного. Як ми уже відзначили в попередньому абзаці, ***змінні – це атрибути***, які є адресами тих даних, що вже 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, фактично, ***в мові Python усі об'єкти є або атрибутами, або методами***. ***В інтерактивній оболонці IPython є низка вбудованих функцій, які будуть виконуватись одразу ж після того, як в командному рядку ви наберете їх ім'я.*** І в першу чергу звертаємо увагу на вбудовану функцію ***print( )***, за допомогою якої інформація, що розміщена в пам'яті комп'ютера під певним іменем, виводиться на екран комп'ютера, якщо ви в аргументні круглі дужки помістите це ім'я. Звертаємо увагу на те, що після запису в командному рядку функції ***print( )***, після неї ніякі розділові знаки не ставляться. Як приклади інших вбудованих функцій наведемо ще декілька: функцію ***abs( )***, яка повертає в програму абсолютне значення (модуль) того числа, що поміщене в аргументних дужках; функцію ***float( )***, яка повертає в програму дійсне значення (десятковий дріб) того числа, що поміщене в аргументні дужки; функцію ***int( )***, яка повертає

в програму цілу частину дробового числа, поміщеного в аргументні дужки; функції ***max( ), min( )***, які повертають в програму максимальний та мінімальний елементи (за якимось критерієм) виразу, що поміщений в аргументні дужки; функцію ***sum( )***, яка реалізує операцію додавання усіх чисел, що поміщені в списку, вкладеному в аргументні дужки, і яка повертає в програму суму цих чисел; функцію ***len( )***, яка підраховує кількість елементів у стрічці, списку чи кортежу, розміщеному у аргументних дужках. Після занесення в командний рядок імен усіх цих вбудованих функцій ніякі розділові знаки після них у цьому командному рядку не ставляться. Але є одна вбудована функція, яка потрібна для створення ***циклу*** і може мати один, два або три числові параметри в аргументних дужках залежно від того, скільки разів цикл має пройти через певний відрізок коду в програмі та через який крок – ця функція носить ім'я ***range(, ,)*** і задає вибраний вами діапазон цілих чисел з першим параметром у вигляді числа, з якого починається відлік кількості циклів 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 число циклів з одиничним ітераційним кроком з початком рахування циклів від нульового значення індексу. Але для того, щоб функція ***range(, ,)*** запрацювала в циклі, командний ***рядок [m]*** з нею має бути записаним так – ***In [m]: for i in range(, ,):.*** Звертаємо увагу на те, що після цієї команди наступний рядок почнеться з чотирьох крапок, а наступна команда, яка є підпорядкованою, має бути після цих крапок ще зсунутою вправо на інтервал зсуву вбудованих функцій. Звертаємо увагу також і на те, що ***після вбудованої функції потрібно для завершення команди натискувати клавішу Enter двічі. Як працюють вбудовані функції показано в пояснювальному прикладі № 7***, наведеному нижче.

Пояснювальний приклад № 7

```
In [1]:  s='puma'                    # Внесення стрічки 'puma' під ім'ям s
In [2]: L=[1,2,3]                    # Внесення списку [1,2,3] під ім'ям L
In [3]: x=10                         # Внесення значення змінної x
In [4]: print(s)                     # Виведення на екран стрічки s
puma
In [5]: print(L)                     # Виведення на екран списку L
[1,2,3]
In [6]: print(x)                     # Виведення на екран значення змінної x
10
In [7]: a=-3; b=4.2                  # Внесення значень чисел a,b
In [8]: abs(a)                       # Визначення модуля числа a
Out[8]: 3
In [9]: float(a)                     # Отримання дійсного значення числа a
Out[9]: -3.0
In [10]: int(b)                      # Отримання цілої частини числа b
Out[10]: 4
In [11]: L=[0,1,8,3,5]               # Внесення списку L
In [12]: max(L)                      # Визначення максимального числа в L
Out[12]: 8
In [13]: min(L)                      # Визначення мінімального числа в L
Out[13]: 0
In [14]: sum(L)                      # Визначення суми чисел в L
Out[14]: 17
In [15]: len(L)                      # Визначення кількості членів в L
Out[15]: 5
```

```
In [16]: range(10,25)                    # Внесення діапазону чисел від 10 до 24
Out[16]: range(10, 25)
In [17]: print(list(range(10,25)))       # Отримання списку чисел від 10 до 24
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,\
 21, 22, 23, 24]
In [18]: range (10,25,2)                 # Внесення попереднього діапазону чисел
Out[18]: range(10, 25, 2)                ….з кроком 2
In [19]: print(list(range(10,25,2)))     # Отримання діапазону чисел від 10 до 24
[10, 12, 14, 16, 18, 20, 22, 24]         ….з кроком 2
In [20]: for i in range(2,5):            # Організація циклу із трьох повторень
             print("Student")                    ….стрічки «Student”

Student
Student
Student
In [21]: for i in range(5):              # Організація циклу з п’яти повторень
             print("Student")            ….стрічки «Student”

Student
Student
Student
Student
Student
In [22}: for i in range(2,5,2):          # Організація циклу із двох повторень
             print("Student")                    ….стрічки «Student”

Student
Student
```

Кінець пояснювального прикладу № 7.

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

Створення функції починається з виклику в командний рядок програми ключового слова ***def***, після якого записується ім’я функції та змінні, які вона використовує з комою між ними (в аргументних дужках). Після цього запису ставиться двокрапка. А у наступному рядку, зсунутому вправо на стандартну для внутрішніх операторів кількість пропусків, після ключового слова ***return*** записується та операція, яку має виконати ця функція. Ну а в наступному рядку вносяться значення змінних, які беруть участь в операції обчислення значення функції. І в цьому ж рядку після коми з крапкою може бути вписаним і ім’я тієї змінної, яка розрахована за основним виразом для заданої функції при заданому значенні аргументних змінних. Якщо перед аргументною змінною, записаною в дужках, вписати символ «*» (зірочка), то це свідчитиме уже не про операцію перемноження, а про те, що ця змінна може використовуватись у функції з довільним числом значень. Але у цьому випадку аргументні змінні, які стоять після змінної з зірочкою, потрібно записувати з використанням не лише їх ключових імен, а й їх значень. Кожна функція може мати у своєму тілі ще й вкладені функції, які мають обчислюватись до обчислення основної функції і відіграють після їх обчислення роль аргументних змінних для основної функції. Для кращого розуміння символів нагадаємо, що ключове слово ***def –*** це скорочення від ***define***, що перекладається як «визначити», а ключове слово ***return*** в перекладі означає «повернути». Приклади викладеного вище у цьому абзаці, наведені у пояснювальному прикладі № 8.

Пояснювальний приклад № 8

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

In [2]: z=5                             # Внесення значення параметра
In [3]: f1(3)                           # Обчислення значення функції
Out[3]: 15                              ….для заданого значення аргументу
In [4]: def f2(x):                      # Визначення функції однієї змінної
            x+=4                        # Внесення значення параметра
            return x                    # Визначення структури функції

In [5]: z=5; f2(5)                      # Обчислення значення функції
Out[5]: 9                               ….для заданого значення аргументу
In [6]: def f4(x):                      # Визначення функції однієї змінної
            return sum(x)               # Визначення структури функції

In [7]: L=[1,2,3,4]                     # Внесення списку значень аргументу
In [8]: f4(L)                           # Обчислення значення функції для
Out[8]: 10                              ….заданого списку значень аргументу
In [9]: L[1]=7                          # Заміна значення аргументу в списку
In [10]: L                              # Виклик зміненого списку значень
Out[10]: [1, 7, 3, 4]                   ….аргументу
In [11]: f4(L)                          # Обчислення значення функції для
Out11]: 15                              ….зміненого списку значень аргументу
In [12]: def f5(x, y):                  # Визначення функції двох змінних
             return x**2*y              # Визначення структури функції

In [13]: x=5; y=4                       # Внесення значень аргументу
In [14]: f5(x, y)                       # Обчислення значення функції для
Out[14]: 100                            ….заданих значень аргументу
In [15]: L1=[5,4]                       # Внесення списку значень аргументу
In [16]: f5(*L1)                        # Обчислення значення функції для
Out[16]: 100                            ….заданого списку значень аргументу
In [17]: def f6(f7, x, y):              # Визначення функції від функції
             return f7(x, y)**2         # Визначення структури функції
                                        ….від функції двох змінних
In [18]: def f7(x, y):                  # Визначення вбудованої функції
             return x+y                 # Визначення структури вбудованої
                                        ….функції двох змінних
In [19]: f6(f7,2,8)                     # Обчислення значення функції від
Out[19]: 100                            ….функції при заданих аргументах
```

Кінець пояснювального прикладу № 8.

Програми, що складені мовою Python, окрім функцій, алгоритм створення яких описано вище, використовують також так звані анонімні функції, які називають ще лямбда-функціями, оскільки містять в собі символ ***lambda.*** Записуються ці анонімні функції у командний рядок ***In[m]*** так: спочатку записується ***ім'я функції***, після якого ставиться знак рівності «=», потім записується символ ***lambda***, потім через пропуск записуються (без дужок) аргументні змінні, між якими ставиться кома, потім ставиться двокрапка «:», а після двокрапки вказується вираз, який має обчислити (з поверненням обчисленого значення в програму) ця функція.

Викликається анонімна функція звичним для виклику функцій способом, тобто вказується її ім'я, і в круглих дужках вказуються конкретні значення аргументних змінних. ***Приклади*** викладеного вище у цьому абзац наведені у ***пояснювальному прикладі № 9***.

Пояснювальний приклад № 9

```
In [1]: f1=lambda: 21                          # Визначення лямбда-функції без аргументів
In [2]: f1( )                                  # Обчислення лямбда-функції без аргументів
Out[2]: 21
In [3]: f2=lambda x, y: x**2+y**3              # Визначення лямбда-функції двох аргументів
In [4]: f2(2,3)                                # Обчислення лямбда-функції двох аргументів
Out[4]: 31                                     ….при конкретизації їх значень
In [5]: f3=lambda x: x**0.5                    # Визначення лямбда-функції одного аргументу
In [6]: f3(4)                                  # Обчислення лямбда-функції одного аргументу
Out[6]: 2.0                                    ….при конкретизації його значення
In [7]: (lambda x, y: x**2+y**3) (2, 3)        # Спосіб визначення лямбда-функції з одночасним
Out[7]: 31                                     ….внесенням значень аргументів та обчисленням
In [8]: def g1(f2,x,y):                        # Визначення функції від лямбда-функції
            f2=lambda x, y: x**2+y**3          # Визначення вбудованої у функцію лямбда-функції
            return f2(1,2)**2                  # Визначення структури функції від лямбда-функції

In [9]: g1(f2,1,2)                             # Обчислення функції від лямбда-функції при
Out[9]: 81                                     ….конкретизації значень аргументів
```

Кінець пояснювального прикладу № 9.

В усіх попередніх прикладах, працюючи з функціями, ми обчислювали їх значення при конкретних значеннях аргументних змінних, а тому отримували кожного разу конкретне число. Але часто нам потрібно знати не одне значення функції при якихось конкретно заданих значеннях аргументних змінних, а цілий ланцюжок значень функції у межах приросту значень аргументних змінних. І при малих приростах аргументних змінних у цьому випадку числові значення функції розміщуватимуться настільки близько одне від одного, що їх графік зливатиметься у суцільну лінію (при одній аргументній змінній) або у суцільну поверхню (при двох аргументних змінних). ***У цьому випадку значення аргументних змінних в програмах мовою Python задаються у вигляді масивів***. І оскільки головним пакетом програм мовою Python, за допомогою якого здійснюються різноманітні числові обчислення, є пакет програм NumPy, який в конкретних програмах записується як numpy, то ***масиви є об'єктами*** саме цього програмного ***пакета numpy***, тобто задавати масиви і здійснювати різноманітні 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 можна лише після того, як ви в командному рядку [m] впишете команду **In [m]: import numpy as np** і натиснете після цього клавішу **Enter.** В роботі [4] 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 дамо словесний портрет перших трьох із цих функцій та продемонструємо їх на пояснювальному прикладі № 10, потім дамо словесний портрет функції ***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 аргументні змінні, але який створює такий же об'єкт, як і ця функція та

продемонструємо їх на пояснювальному прикладі № 11. А потім дамо словесний портрет останніх шести функцій та продемонструємо їх на пояснювальному прикладі № 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 по кожному з його вимірів визначається атрибутом ***shape***, записаним після імені масиву з крапкою між ними, а кількість членів масиву по першому 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 із 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 з 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.

Пояснювальний 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 із списку
In [3]: a1                                  # Виклик операції створення
Out[3]: array([0, 1, 3, 2, 5])              ….одновимірного масиву зі списку
In [4]: print(a1)                           # Візуалізація створеного масиву на екрані
[0 1 3 2 5]                                 …. (Увага! Члени масиву уже без ком)
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 операції створення
Out[7]:                                     ….двовимірного масиву зі списку
array([[3, 4, 5],
       [6, 7, 8]])
In [8]: a2[0,2]                             # Виклик члена масиву з індексом 0 по
Out[8]: 5                                   ….рядку та індексом 2 по стовпцю
In [9]: a2[0][2]                            # Інший варіант виклику члена масиву з
Out[9]: 5                                   ….індексами 0 по рядку та 2 по стовпцю
In [10]: a2.ndim                            # Визначення вимірності масиву
Out[10]: 2
In [11]: a2.shape                           # Визначення кількості рядків та кількості
Out[11]: (2, 3)                             ….стовпців в масиві
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 а3 в
Out[14]: array ([1. , 2.5, 4. , 5.5, 7. ])      ….діапазоні чисел від 1 до 8 через крок 1.5
In [15]: print (a3)                       # Візуалізація одновимірного масиву а3
[1.  2.5 4.  5.5 7. ]
In [16]: a4=np.linspace (1,8,10)          # Створення одновимірного масиву а4
In [17]: a4                               ….із 10 чисел в діапазоні значень від 1 до 8
Out[17]:
array([1.        , 1.77777778, 2.55555556,
3.33333333, 4.11111111, 4.88888889,
5.66666667, 6.44444444, # 7.22222222, 8. ])
In [18]: print(a4)                        # Візуалізація одновимірного масиву а4
[1.         1.77777778 2.55555556 3.33333333
4.11111111 4.88888889 5.66666667 6.44444444
 7.22222222 8. ]
```

Кінець пояснювального прикладу № 10.

А тепер дамо словесний портрет функції ***meshgrid(x,y),*** яка, на відміну від перших трьох, певною мірою є спеціалізованою, та продемонструємо її на пояснювальному прикладі № 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.

Пояснювальний 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 з однаковими
[[4 5 6]                                  ….рядками
 [4 5 6]]
In [6]: print (Y)                         # Виклик на екран масиву Y з однаковими
[[7 7 7]                                  ….стовпцями
 [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 з однаковими
[[-1.  0.  1.]                            ….рядками
 [-1.  0.  1.]
 [-1.  0.  1.]]
```

```
In [11]: print (Y1)                          # Виклик на екран масиву Y1 з однаковими
[[-1. -1. -1.]                               ….стовпцями
 [ 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 з однаковими
[[0 2 4]                                     ….рядками
 [0 2 4]
 [0 2 4]]
In [18]: print(Y2)                           # Виклик на екран масиву Y2 з однаковими
[[ 8  8  8]                                  ….стовпцями
 [10 10 10]
 [12 12 12]]
In [19]: X3,Y3=np.mgrid[0:6:2,8:13:2]        # Масиви X3,Y3 створює оператор mgrid
In [20]: print(X3)                           # Виклик на екран масиву X3 з однаковими
[[0 0 0]                                     ….стовпцями
 [2 2 2]
 [4 4 4]]
In [21]: print(Y3)                           # Виклик на екран масиву Y3 з однаковими
[[ 8 10 12]                                  …..рядками
 [ 8 10 12]
 [ 8 10 12]]
```

Кінець пояснювального прикладу № 11.

А тепер дамо словесний портрет функцій ***zeros( ), ones( ), eye( ), identity( ), diag( ), fromfunction( ),*** які, як і функція ***meshgrid(x,y),*** теж є певною мірою спеціалізованими, та продемонструємо їх на пояснювальному прикладі № 12. Функція ***zeros( ),*** після її виклику з програмного пакета ***numpy*** проставленням скороченого символа ***np.*** перед її іменем, створює ***масив із нулів***, кількість яких визначається числом, проставленим 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 – кількість стовпців з нулями. Функція ***ones( ),*** після її виклику з програмного пакета ***numpy*** проставленням скороченого символа ***np.*** перед її іменем, створює ***масив із одиниць***, кількість яких визначається числом, проставленим 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 з програмного пакета ***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 всі інші елементи є нулями. Функція ***diag( ),*** після виклику якої з програмного пакета ***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 програмного пакета ***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.

Пояснювальний приклад № 12

```
In [1]: a1=np.zeros(4);a1                # Створення одновимірного масиву a1
Out[1]: array([0., 0., 0., 0.])          ….з усіма нулями
In [2]: print(a1)                        # Виклик на екран одновимірного масиву a1
[0. 0. 0. 0.]                            ….з усіма нулями
In [3]: a2=np.zeros([2,4]);a2            # Створення двовимірного масиву a2
Out[3]:                                  ….з усіма нулями
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.]])
In [4]: print(a2)                        # Виклик на екран двовимірного масиву a2
[[0. 0. 0. 0.]                           ….з усіма нулями
 [0. 0. 0. 0.]]
In [5]: b1=np.ones(4);b1                 # Створення одновимірного масиву b1
Out[5]: array([1., 1., 1., 1.])          ….з усіма одиницями
In [6]: print(b1)                        # Виклик на екран одновимірного масиву b1
[1. 1. 1. 1.]                            ….з усіма одиницями
In [7]: b2=np.ones([2,3]);b2             # Створення двовимірного масиву b2
Out[7]:                                  ….з усіма одиницями
array([[1., 1., 1.],
       [1., 1., 1.]])
In [8]: print(b2)                        # Виклик на екран двовимірного масиву b2
[[1. 1. 1.]                              ….з усіма одиницями
 [1. 1. 1.]]
In [9]: d1=np.eye(3);d1                  # Створення одиничного діагонального
Out[9]:                                  ….масиву d1 функцією eye( )
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
In [10]: print(d1)                       # Виклик на екран одиничного діагонального
[[1. 0. 0.]                              ….масиву d1, створеного функцією eye( )
 [0. 1. 0.]
 [0. 0. 1.]]
In [11]: d2=np.identity(3);d2            # Створення одиничного діагонального
Out[11]:                                 ….масиву d2 функцією identity( )
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
In [12]: print(d2)                       # Виклик на екран одиничного діагонального
[[1. 0. 0.]                              ….масиву d2, створеного функцією identity( )
[0. 1. 0.]
 [0. 0. 1.]]
In [13]: d3=np.diag([1,10,100]);d3       # Створення діагонального масиву d3
Out[13]:                                 ….з потрібними членами на діагоналі
array([[  1,   0,   0],
       [  0,  10,   0],
       [  0,   0, 100]])
In [14]: print(d3)                       # Виклик на екран діагонального масиву d3
[[  1   0   0]                           ….з потрібними членами на діагоналі
```

```
 [  0  10   0]
 [  0   0 100]]
In [15]: d4=np.diag([1,3,5],1);d4          # Створення діагонального масиву d4
Out[15]:                                   ….з діагоналлю, зсунутою на 1 вгору
array([[0, 1, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 0, 5],
       [0, 0, 0, 0]])
In [16]: print(d4)                         # Виклик на екран діагонального масиву d4
[[0 1 0 0]                                 ….з діагоналлю, зсунутою на 1 вгору
 [0 0 3 0]
 [0 0 0 5]
 [0 0 0 0]]
In [17]: d5=np.diag([1,3,5],-1);d5         # Створення діагонального масиву d5
Out[17]:                                   ….з діагоналлю, зсунутою на 1 вниз
array([[0, 0, 0, 0],
       [1, 0, 0, 0],
       [0, 3, 0, 0],
       [0, 0, 5, 0]])
In [18]: print(d5)                         # Виклик на екран діагонального масиву d5
[[0 0 0 0]                                 ….з діагоналлю, зсунутою на 1 вниз
 [1 0 0 0]
 [0 3 0 0]
 [0 0 5 0]]
In [19]: def f(i,j):                       # Створення функції f(i,j), змінними якої є
             return i**3+j**3              ….індекси i,j елементів масиву, що формується

In [20]: g1=np.fromfunction(f,(3,3));g1    # Створення квадратного масиву g1
Out[20]:                                   ….з елементів, сформованих функцією f(i,j)
array([[ 0.,  1.,  8.],
       [ 1.,  2.,  9.],
       [ 8.,  9., 16.]])
In [21]: print(g1)                         # Виклик на екран квадратного масиву g1
[[ 0.  1.  8.]                             ….з елементів, сформованих функцією f(i,j)
 [ 1.  2.  9.]
 [ 8.  9. 16.]]
In [22]: g2=np.fromfunction(lambda i,j:\
 i**3+j**3,(3,3))                          # Створення та візуалізація
In [23]: g2                                ….квадратного масиву g2,
Out[23]:                                   ….сформованого з використанням
array([[ 0.,  1.,  8.],                    ….лямбда-функції
       [ 1.,  2.,  9.],
       [ 8.,  9., 16.]])
In [24]: print(g2)                         # Виклик на екран квадратного масиву g2,
[[ 0.  1.  8.]                             ….сформованого з використанням
 [ 1.  2.  9.]                             ….лямбда-функції
 [ 8.  9. 16.]]
```

Кінець пояснювального прикладу № 12.

На пояснювальних прикладах № 9–№ 12 та в поясненнях до них ми уже виклали технологію створення масивів, а тепер, з використанням інформації, викладеної в роботі [4], покажемо, які ***операції*** можна виконувати ***з масивами***. В програмах мовою Python використовується значна кількість цих операцій, але ми зупинимось лише на деяких із них, яких буде достатньо для розв'язання наших задач. 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 їх змінних, заданих масивами, 4) блок операцій з масивами трансформованими в матриці***. І ***щодо першого блоку*** 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 ***aranspose( )*** 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 ***першого блоку*** в програмах мовою Python ***продемонстровано у пояснювальному прикладі № 13.***

Пояснювальний приклад № 13

```
In [1]: import numpy as np                  # Виклик ППП питру 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 і виклик на екран
Out[5]: array([-4, -3, -2, -1])                  ….масиву b1 як різниці a-5
In [6]: b2=a*5;b2                           # Формування і виклик на екран
Out[6]: array([ 5, 10, 15, 20])             ….масиву b2 як добутку a  на 5
In [7]: b3=a/5;b3                           # Формування і виклик на екран
Out[7]: array([0.2, 0.4, 0.6, 0.8])         ….масиву b3 як результат ділення a на 5
In [8]: b4=a**5;b4                          # Формування і виклик на екран
Out[8]: array([   1,   32,  243, 1024],     ….масиву b4 як п'ятого степеню від a
 dtype=int32)
In [9]: b5=a.reshape(2,2);b5                # Перетворення одновимірного масиву a
Out[9]:                                     ….в двовимірний масив b5
array([[1, 2],
       [3, 4]])
```

```
In [10]: a1=np.array([17]);a1                  # Внесення масиву a1 , що містить
Out[10]: array([17])                           ….один елемент в списку
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 і візуалізація
Out[13]:                                       ….двовимірного масиву a3
array([[5, 6, 7],
       [5, 3, 1]])
In [14]: a3.shape=(6,);a3                      # Перетворення двовимірного
Out[14]: array([5, 6, 7, 5, 3, 1])             ….масиву в одновимірний
In [15]: a4=np.arange(16).reshape((4,4))       # Створення масиву a4 розміром (4x4)
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( )                        # Створення масиву із головної
Out[17]: array([ 0,  5, 10, 15])               ….діагоналі масиву a4
In [18]: print(a4.diagonal( ))                 # Виклик на екран масиву, створеного
[ 0  5 10 15]                                  ….із діагоналі масиву a4
In [19]: a5=a4.transpose( );a5                 # Створення масиву a5
Out[19]:                                       ….транспонуванням масиву a4
array([[ 0,  4,  8, 12],
       [ 1,  5,  9, 13],
       [ 2,  6, 10, 14],
       [ 3,  7, 11, 15]])
In [20]: print(a5)                             # Виклик на екран масиву a5
[[ 0  4  8 12]
 [ 1  5  9 13]
 [ 2  6 10 14]
 [ 3  7 11 15]]
In [21]: a5.T                                  # Повернення до масиву a4
Out[21]:                                       ….транспонуванням масиву a5
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
In [22]: a6=np.array([-7,-9,-1,5,10,12]);a6    # Створення та візуалізація
Out[22]: array([-7, -9, -1,  5, 10, 12])       ….масиву a6
In [23]: a6.sum( )                             # Обчислення суми елементів
Out[23]: 10                                    ….масиву a6
In [24]: a6.prod( )                            # Обчислення добутку елементів
Out[24]: -37800                                ….масиву a6
In [25]: a6.min( )                             # Визначення елемента масиву a6
Out[25]: -9                                    ….з найменшим значенням
In [26]: a6.argmin( )                          # Визначення індексу елемента
Out[26]: 1                                     ….масиву a6 з найменшим значенням
```

```
In [27]: a6.max( )                    # Визначення елемента масиву а6
Out[27]: 12                           ….з найбільшим значенням
In [28]: a6.argmax( )                 # Визначення індексу елемента
Out[28]: 5                            ….масиву а6 з найбільшим значенням
In [29]: a6.mean( )                   # Визначення середнього значення
Out[29]: 1.6666666666666667           ….масиву а6
In [30]: a6.std( )                    # Визначення середньоквадратичного
Out[30]: 7.993052538854532            ….відхилення масиву а6
In [31]: a7= np.array([1,5,9]);a7     # Створення та візуалізація
Out[31]: array([1, 5, 9])             ….одновимірного масиву а7
In [32]: b7=a7[:,None];b7             # Перетворення одновимірного масиву а7
Out[32       ]:                       ….у двовимірний масив b7 та його
array([[1],                           ….візуалізація
       [5],
       [9]])
```

Кінець пояснювального прикладу № 13.

Щодо другого блоку, тобто ***блоку 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, 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 мовою Python ***продемонстровано у пояснювальному прикладі № 14.***

Пояснювальний приклад № 14

```
In [1]: import numpy as np            # Виклик ППП питру з символом пр
In [2]: a=np.array([2,4,6,8])         # Внесення масиву-рядка а
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])       ….масивів а та b
In [5]: print(a+b)                    # Виклик на екран суми масивів а та b
[ 4  5  8 12]
In [6]: c=np.array([[1],[2],[3],[4]]) # Внесення масиву-стовпця с
In [7]: c                             # Візуалізація масиву-стовпця с
Out[7]:
array([[1],
       [2],
       [3],
       [4]])
In [8]: a+c                           # Отримання та візуалізація суми
Out[8]:                               ….масиву-рядка а та масиву-стовпця с
```

```
array([[ 3,  5,  7,  9],
       [ 4,  6,  8, 10],
       [ 5,  7,  9, 11],
       [ 6,  8, 10, 12]])
In [9]: print(a+c)                    # Виклик на екран суми
[[ 3  5  7  9]                        ….масиву-рядка a та масиву-стовпця c
 [ 4  6  8 10]
 [ 5  7  9 11]
 [ 6  8 10 12]]
In [10]: a*b                          # Отримання поелементного
Out[10]: array([ 4,  4, 12, 32])      ….добутку масивів a та b
In [11]: print(a*b)                   # Виклик на екран
[ 4  4 12 32]                         ….добутку масивів a та b
In [12]: a/b                          # Реалізація операції поелементного
Out[12]: array([1., 4., 3., 2.])      ….ділення масивів a та b
In [13]: print(a/b)                   # Виклик на екран результату
[1. 4. 3. 2.]                         ….ділення масивів a та b
In [14]: a**b                         # Поелементне піднесення до степеню
Out[14]: array([   4,    4,   36, 4096],   ….масивів a та b
 dtype=int32)
In [15]: print(a**b)                  # Виклик на екран результату
[   4    4   36 4096]                 ….піднесення до степеню масивів a та b
In [16]: a==b                         # Булеве поелементне порівняння
Out[16]: array([ True, False, False, False])   ….масивів a та b за операцією рівність
In [17]: print(a==b)                  # Виклик на екран результату
[ True False False False]             ….булевого порівняння масивів a та b
In [18]: a<b                          # Булеве поелементне порівняння
Out[18]: array([False, False, False, False])   ….масивів a та b за операцією меньше
In [19]: print(a<b)                   # Екранізація результату порівняння
[False False False False]             ….масивів a та b за операцією меньше
In [20]: a>b                          # Булеве поелементне порівняння
Out[20]: array([False,  True,  True,  True])   ….масивів a та b за операцією більше
In [21]: print(a>b)                   # Візуалізація результату порівняння
[False  True  True  True]             ….масивів a та b за операцією більше
```

Кінець пояснювального прикладу №14.

Щодо третього блоку, тобто ***блоку операцій з обчисленням функцій від їх змінних, заданих масивами***, ми розглянемо в межах ППП ***питру*** лише ***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.hstack((a,b))***, яка об'єднує масиви ***a*** та ***b*** по горизонталі; функцію ***np.vstack((a,b))***, яка об'єднує масиви ***a*** та ***b*** по вертикалі; функцію ***np.hsplit(a,k)*** або ***np.hsplit(a,[k,m])***, перша з яких розбиває по горизонталі масив ***a*** на ***k*** підмасивів однакового розміру, а друга розбиває по горизонталі цей же масив, у даному

випадку, на три підмасиви за номерами стовпців ***k*** та ***m***; функцію ***np.vsplit(a,k)*** або ***np.vsplit(a,[k,m])***, перша з яких розбиває по вертикалі масив ***a*** на ***k*** підмасивів однакового розміру, а друга розбиває по вертикалі цей же масив, у даному випадку, на три підмасиви за номерами рядків ***k*** та ***m***; функцію ***np.sort(a)***, яка формує масив ***b*** шляхом сортування числового масиву ***a*** за критерієм зростання значень його членів; функцію ***x.round(k)***, яка 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 мовою Python, окрім функцій ***np.hstack((a,b))***, ***np.vstack((a,b))***, ***np.hsplit(a,k), np.hsplit(a,[k,m])***, ***np.vsplit(a,k), np.vsplit(a,[k,m])***, які при розв'язанні задач функціонального аналізу нами використовуватись не будуть, ***продемонстровано у пояснювальному прикладі № 15.***

Пояснювальний приклад № 15

```
In [1]: import numpy as np                  # Виклик ППП pитру з символом np
In [2]: a1=np.array([2,4,6,8,10,12])        # Внесення одновимірного
                                            ….масиву-списку a1
In [3]: a2=np.array([-3,-1,0,1,3,5])        # Внесення одновимірного масиву-
                                            ….списку a2
In [4]: np.dot(a1,a2)                       # Отримання скалярного добутку
Out[4]: 88                                  ….масивів a1 та a2
In [5]: b1=np.array([[2,4,6],[8,10,12]]);b1 # Внесення та візуалізація
Out[5]:                                     ….двовимірного масиву b1
array([[ 2,  4,  6],
       [ 8, 10, 12]])
In [6]: b2=np.array([[-3,-1],[0,1],[3,5]]);b2 # Внесення та візуалізація
Out[6]:                                     ….двовимірного масиву b2
array([[-3, -1],
       [ 0,  1],
       [ 3,  5]])
In [7]: np.dot(b1,b2)                       # Матричне перемноження масивів
Out[7]:                                     …. b1та b2 та його візуалізація
array([[12, 32],
       [12, 62]])
In [8]:np.matmul(a1,a2)                     # Отримання скалярного добутку
Out[8]: 88                                  ….масивів a1 та a2
In [9]: np.matmul(b1,b2)                    # Матричне перемноження
Out[9]:                                     ….масивів b1 та b2 та його  візуалізація
array([[12, 32],
       [12, 62]])
In [10]: a3=np.array([2+4j,6+8j,10+12j]);a3 # Внесення та візуалізація
Out[10]: array([ 2. +4.j,  6.+8.j, 10.+12.j]) .масиву-списку комплексних чисел a3
In [11]: a4=np.array([-3-1j,0+1j,3+5j]);a4  # Внесення та візуалізація
Out[11]: array([-3.-1.j,  0.+1.j,  3.+5.j]) .масиву-списку комплексних чисел a4
In [12]: np.vdot(a3,a4)                     # Отримання скалярного добутку
Out[12]: (88+30j)                           ….масивів a3 та a4 зі спряженням a3
```

```
In [13]: b4=np.array([[-3-1j], [0+1j],[3+5j]]);b4      # Внесення та візуалізація двовимірного
Out[13]:                                               ….масиву комплексних чисел b4
array([[-3.-1.j],
       [ 0.+1.j],
       [ 3.+5.j]])
In [14]: np.vdot(a3,b4)                                # Отримання скалярного добутку
Out[14]: (88+30j)                                      ….масивів a3 та b4 зі спряженням a3
In [15]: np.dot(a3,b4)                                 # Отримання скалярного добутку
Out[15]: array([-40.+78.j])                            ….масивів a3 та b4 без спряження
In [16]: a5=np.array([3,5]);a5                         # Внесення та візуалізація
Out[16]: array([3, 5])                                 ….одновимірного масиву-списку a5
In [17]: b5=np.array([7,9]);b5                         # Внесення та візуалізація
Out[17]: array([7, 9])                                 ….одновимірного масиву-списку b5
In [18]: np.cross(a5,b5)                               # Векторне перемноження
Out[18]: array(-8)                                     ….масивів a5 та b5 та його  візуалізація
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
Out[21]: array([-8, 16, -8])                           ….масивів a6 та b6 та його візуалізація
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
Out[24]:                                               …. a7 та b7 та його візуалізація
array([[10, 15, 20],
       [20, 30, 40]])
In [25]: np.outer(a6,b6)                               # Зовнішнє перемноження
Out[25]:                                               ….масивів a6 та b6 та його візуалізація
array([[21, 27, 33],
       [35, 45, 55],
       [49, 63, 77]])
In [26]: np.transpose(np.outer(a6,b6))                 # Транспонування результату
Out[26]:                                               ….зовнішнього перемноження
array([[21, 35, 49],                                   ….масивів a6 та b6 та його візуалізація
       [27, 45, 63],
       [33, 55, 77]])
In [27]: np.outer(a6,b6).T                             # Транспонування результату
Out[27]:                                               ….зовнішнього перемноження
array([[21, 35, 49],                                   ….масивів a6 та b6 та його візуалізація
       [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);b8                             # Сортування масиву a8
Out[29]: array([-6, -3, -1,  2,  3,  5,  7,  8])       ….та його візуалізація
In [30]: a9=np.array([5.615,-1.12,8.435,2.10,-1,0])    # Внесення масиву чисел a9
In [31]: b9=a9.round(1);b9                             # Округлення масиву чисел a9 до
Out[31]: array([ 5.6, -1.1,  8.4,  2.1, -1.0])         ….одного знака після коми
                                                       ….та його візуалізація
In [32]: np.piecewise(b8,[b8<0,b8>=0], \               # Абсолютизація(модулізація) масиву
       [lambda b8:-b8,lambda b8: b8])                  ….чисел b8 та її візуалізація
Out[32]: array([ 5.6,  1.1,  8.4,  2.1, 11.2])
```

```
In [33]: np.piecewise(b8,[b8<-1, \                    # Формування із масиву чисел b8
     np.logical_and(b8>=-1,b8<5),b8>=5],\             .нового масиву із чисел, залежних від
[-1,0,1])                                             ….заданих логічних умов, то його
Out[33]: array([ 1., -1.,  1.,  0., -1.])             ….візуалізація
In [34]: np.piecewise(b8,[b8<-1, \
     b8>=5],[lambda t:-t,lambda t:t**2,\              # Формування з масиву чисел b8
        np.logical_and(b8>=-1,b8<5),\                 ….нового масиву з чисел, залежних
lambda t:t**0.5])                                     ….від заданих логічних умов і лямбда-
Out[34]:                                              ….функцій, та його візуалізація
array([ 2.36643191, 1.1 , 2.89827535, 4.41 , 11.2 ])
In [35]: from math import cos                         # Виклик функції cos із модуля math
In [36]: mycos=np.vectorize(cos)                      # Формування операції векторизації
In [37]: x=[-np.pi/2,-np.pi/4,0,np.pi/4,np.pi/2]      # Внесення масиву значень аргументу
In [38]: mycos(x)                                     # Векторизація функції cos
Out[38]:                                              ….та її візуалізація
array([6.12323400e-17, 7.07106781e-01, \
1.00000000e+00, 7.07106781e-01,\
        6.12323400e-17])
In [39]: def f(x):                                    # Формування власної функції f(x)
        return cos(x)*3                               # Визначення тіла власної функції f(x),
                                                      ….пов'язаної з уже визначеною раніше
In [40]: fvec=np.vectorize(f)                         # Формування операції векторизації
In [41]: fvec(x)                                      # Векторизація власної функції f(x)
Out[41]:                                              ….та її візуалізація
array([1.83697020e-16, 2.12132034e+00, \
3.00000000e+00, 2.12132034e+00,\
        1.83697020e-16])
In [42]: l1=lambda x: x**3-2                          # Формування лямбда-функції l1 та її
                                                      ….тіла
In [43]: l1vec=np.vectorize(l1)                       # Векторизація лямбда-функції l1
In [44]: l1vec(x)                                     # Визначення векторизованих значень
Out[44]:                                              ….лямбда-функції l1 для заданих
array([[ -1,   6,  25],                               ….значень аргументу x та їх
       [ 62, 123, 214]])                              ….візуалізація

In [45]: x2=np.linspace(0,6,7);                       # Внесення масиву значень x2
In [46]: x2                                           # Візуалізація масиву значень x2
Out[46]: array([0., 1., 2., 3., 4., 5., 6.])
In [47]: y2=np.linspace(-3,3,7)                       # Внесення масиву значень y2
In [48]: y2                                           # Візуалізація масиву значень y2
Out[48]: array([-3., -2., -1.,  0.,  1.,  2.,  3.])
In [49]: l12=lambda x2,y2: x2**3+y2**2                # Формування лямбда-функції l12
                                                      ….та її тіла
In [50]: l12vec=np.vectorize(l12)                     # Векторизація лямбда-функції l12
In [51]: l12vec(x2,y2)                                # Визначення векторизованих значень
Out[51]:                                              ….лямбда-функції l12 для заданих
array([  9.,   5.,   9.,  27.,  65., 129., 225.])     ….значень аргументів x2, y2
                                                      ….та їх візуалізація
In [52]: a10=np.array([1,2,3,4,5,6,7])                # Внесення одновимірного масиву a10
```

```
In [53]: np.cumsum(a10)                          # Формування масиву із сум
Out[53]:                                         ….попередніх членів масиву a10
array([ 1,  3,  6, 10, 15, 21, 28], dtype=int32) ....та його візуалізація
In [54]: np.diff(a10)                             # Формування масиву із різниць
Out[54]:                                         ….сусідніх членів одновимірного
array([1, 1, 1, 1, 1, 1])                        ….масиву a10 та його візуалізація
In [55]: np.diff([[1,2,3,2,1],[5,1,4,2,3],\      # Формування масиву із різниць
 [1,2,3,4,5]],axis=0)                            ….сусідніх членів двовимірного
Out[55]:                                         ….масиву вздовж осі 0
array([[ 4, -1,  1,  0,  2],                    ….та його візуалізація

In [56]: np.diff([[1,2,3,2,1],[5,1,4,2,3],\      # Формування масиву із різниць
   [1,2,3,4,5]],axis=1)                          ….сусідніх членів двовимірного масиву
Out[56]:                                         …. вздовж осі 1 та його візуалізація
array([[ 1,  1, -1, -1],
       [-4,  3, -2,  1],
       [ 1,  1,  1,  1]])
```

Кінець пояснювального прикладу № 15.

А щодо четвертого блоку, тобто ***блоку операцій з масивами, трансформованими в матриці,*** ми розглянемо***,*** по-перше, ***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 всі операції лінійної алгебри, а по-друге, ***операції,*** що виконуються функціями модуля ***scipy.linalg*** (який скорочено позначається як ***la***) ППП ***scipy,*** а саме: ***операцію*** обчислення визначника ***la.det(a)*** квадратного масиву ***a,*** якого без додаткового перетворення наділено властивостями матриці; ***операцію*** обчислення оберненої матриці ***la.inv(a)*** квадратного масиву ***a,*** якого без додаткового перетворення наділено властивостями матриці; ***операцію*** обчислення власних чисел ***w*** і власних векторів ***v la.eig(a)*** квадратного масиву ***a,*** якого без додаткового перетворення наділено властивостями матриці; ***операцію*** обчислення норми Евкліда ***la.norm(a)*** масиву ***a***. Використання усіх цих функцій і операцій в програмах мовою Python ***продемонстровано у пояснювальному прикладі № 16.***

Пояснювальний 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 '1 2;3 4'
Out[2]:                                          ….в матрицю A1 та її
matrix([[1, 2],                                  ….візуалізація
        [3, 4]])
In [3]: B1=np.mat([[5,6],[7,8]]);B1              # Трансформація списку [[1, 2],[3,4]]
Out[3]:                                          ….в матрицю B1 та її
matrix([[5, 6],                                  ….візуалізація
        [7, 8]])
In [4]: A1+B1                                    # Обчислення суми матриць A1 та B1
Out[4]:                                          ….та її візуалізація
matrix([[ 6,  8],
        [10, 12]])
In [5]: A1*B1                                    # Перемноження матриць A1 та B1
Out[5]:                                          ….та візуалізація результату
```

```
matrix([[19, 22],
        [43, 50]])
In [6]: A1**2                          # Піднесення до другого степеня
Out[6]:                                ….матриці A1 (тобто перемноження її
matrix([[ 7, 10],                      ….саму на себе) та візуалізація результату
        [15, 22]])
In [7]: A1.I                           # Обчислення матриці, оберненої
Out[7]:                                ….до матриці A1, та візуалізація
matrix([[-2. ,  1. ],                  ….результату
        [ 1.5, -0.5]])
In [8]: A1.T                           # Транспонування матриці A1
Out[8]:                                ….та візуалізація результату
matrix([[1, 3],
        [2, 4]])
In [9]: import scipy.linalg as la      # Виклик модуля linalg ППП scipy як la
In [10]: c1=la.det(A1);c1              # Обчислення визначника c1 матриці A1
Out[10]: -2.0                          ….та візуалізація результату
In [11]: A3=la.inv(A1);A3              # Обчислення матриці A3, оберненої
Out[11]:                               ….до матриці A1, та візуалізація
array([[-2. ,  1. ],                   ….результату
       [ 1.5, -0.5]])
In [12]: c=la.norm(A1);c               # Обчислення норми Евкліда c вектора,
Out[12]: 5.477225575051661             ….заданого матрицею A1
In [13]: w,v=la.eig(A1)                # Обчислення власних чисел w та
In [14]: w,v                           ….власних векторів v матриці A1
Out[14]:                               ….та візуалізація результату
(array([-0.37228132+0.j,  5.37228132+0.j]),  # Числові значення власних чисел w
 array([[-0.82456484, -0.41597356],          # Числові значення проєкцій власних
        [ 0.56576746, -0.90937671]]))        ….векторів v
```

Кінець пояснювального прикладу № 16.

1.6 Задачі на визначення характеристик метричних просторів в програмах мовою Python

Покажемо, як, використовуючи матеріал, викладений у двох попередніх підрозділах цього розділу, визначати характеристики метричних просторів, використовуючи програми мовою Python, а саме: як ***визначати норми та метрики банахових просторів***, елементами яких є числа, – це буде реалізовувати ***програма 3***, як ***визначати норми та метрики банахових просторів***, елементами яких є функції, – це буде реалізовувати ***програма 4***, та як ***визначати норми та метрики лебегівських просторів***, елементами яких є функції, – це буде реалізовувати ***програма 5. Нагадуємо,*** що і в цих програмах після символа ***«#»*** записане роз'яснення змісту дії, що виконується програмою в даному командному рядку, яке в саму програму не вноситься, а символом «\» (зворотний слеш), який в програму вноситься, здійснюється перенесення частини рядка в рядок наступний.

Програма мовою Python для визначення норми та метрики банахових просторів, елементами яких є числа

(Програма 3)

```
In [1]: import numpy as np                  # Виклик ППП питру під символом пр
In [2]: a2=np.array([1,2])                  # Внесення масиву проєкцій точки а2
In [3]: a3=np.array([1,2,3])                # Внесення масиву проєкцій точки а3
In [4]: a4=np.array([1,2,3,4])              # Внесення масиву проєкцій точки а4
In [5]: c2=np.array([2,1])                  # Внесення масиву проєкцій точки с2
In [6]: c3=np.array([3,2,1])                # Внесення масиву проєкцій точки с3
In [7]: c4=np.array([4,3,2,1])              # Внесення масиву проєкцій точки с4
In [8]: e2=a2-c2;e2                         # Формування поелементної різниці е2
Out[8]: array([-1,  1])                     ….масивів а2,с2 та її візуалізація
In [9]: e3=a3-c3;e3                         # Формування поелементної різниці е3
Out[9]: array([-2,  0,  2])                 ….масивів а3,с3 та її візуалізація
In [10]: e4=a4-c4;e4                        # Формування поелементної різниці е4
Out[10]: array([-3, -1,  1,  3])            ….масивів а4,с4 та її візуалізація
In [11]: import scipy                       # Виклик ППП scipy
In [12]: import scipy.linalg as la          # Виклик модуля scipy.linalg як la
In [13]: la.norm(a2)                        # Обчислення норми ||а2|| метричного
Out[13]: 2.23606797749979                   ….простору в точці а2 та її візуалізація
In [14]: la.norm(a3)                        # Обчислення норми ||а3|| метричного
Out[14]: 3.7416573867739413                 ….простору в точці а3 та її візуалізація
In [15]: la.norm(a4)                        # Обчислення норми ||а4|| метричного
Out[15]: 5.477225575051661                  ….простору в точці а4 та її візуалізація
In [16]: la.norm(c2)                        # Обчислення норми ||с2|| метричного
Out[16]: 2.23606797749979                   ….простору в точці с2 та її візуалізація
In [17]: la.norm(c3)                        # Обчислення норми ||с3|| метричного
Out[17]: 3.7416573867739413                 ….простору в точці с3 та її візуалізація
In [18]: la.norm(c4)                        # Обчислення норми ||с4|| метричного
Out[18]: 5.477225575051661                  ….простору в точці с4 та її візуалізація
In [19]: m2=la.norm(e2);m2                  # Обчислення метрики m2 простору між
Out[19]: 1.4142135623730951                 ….точками а2,с2 та її візуалізація
In [20]: m3=la.norm(e3);m3                  # Обчислення метрики m3 простору між
Out[20]: 2.8284271247461903                 ….точками а3,с3 та її візуалізація
In 21]: m4=la.norm(e4);m4                   # Обчислення метрики m4 простору між
Out[21]: 4.47213595499958                   ….точками а4,с4 та її візуалізація
```

Кінець програми 3.

Програма мовою Python для визначати норми та метрики банахових просторів С[0,1], елементами яких є функції

(Програма 4)

```
In [1]: import numpy as np                  # Виклик ППП питру під символом пр
In [2]: x=np.linspace(0,1,11)               # Внесення масиву значень аргументу х
In [3]: g1=lambda x: -1+3*x-x**2            # Формування функції g1 лямбда-функцією
In [4]: g1vec=np.vectorize(g1)              # Векторизація сформованої функції g1
In [5]: g11=g1vec(x)                        # Формування значень функції g11 із g1
In [6]: g11                                 # Візуалізація значень функції g11
Out[6]:
array([-1.  , -0.71, -0.44, -0.19,  0.04,  0.25,
        0.44,  0.61,  0.76,  0.89,  1.  ])
```

```
In [7]: g111=np.piecewise(g11,[g11<0,g11>=0],\   # Абсолютизація g111 значень g11
   [lambda g11:-g11,lambda g11: g11])            ….векторизованої функції g1
In [8]: g111                                     # Візуалізація абсолютизованих значень
Out[8]: array([1.  , 0.71, 0.44, 0.19, 0.04, 0.25,   ….g111 векторизованої функції g1
        0.44, 0.61, 0.76, 0.89, 1.  ])
In [9]: ng1=g111.max( );ng1                      # Визначення норми ng1 функції g1
Out[9]: 1.0                                      ….в метричному просторі C[0,1]
In [10]: ig1=g111.argmax( );ig1                  # Визначення індексу ig1 точки, в якій
Out[10]: 0                                       ….абсолютне значення g111 максимальне
In [11]: g2=lambda x: 5*x-6*x**2                 # Формування функції g2 лямбда-функцією
In [12]: g2vec=np.vectorize(g2)                  # Векторизація сформованої функції g2
In [13}: g22=g2vec(x);g22                        # Визначення векторизованої функції g22
Out[13]:                                         ….та її візуалізація
array([ 0.  ,  0.44,  0.76,  0.96,  1.04,  1.  ,
        0.84,  0.56,  0.16,  -0.36, -1.  ])
In [14]: g222=np.piecewise(g22,[g22<0,\          # Абсолютизація g222 значень g22
        g22>=0], [lambda g22:-g22,\               ….векторизованої функції g2
        lambda g22: g22])
In [15]: g222                                    # Візуалізація абсолютизованих значень
Out[15]: array([0.  , 0.44, 0.76, 0.96, 1.04, 1.  ,   ….g222 векторизованої функції g2
        0.84, 0.56, 0.16, 0.36, 1.  ])
In [16]: ng2=g222.max( ); ng2                    # Визначення норми ng2 функції g2
Out[16]: 1.0399999999999998                      ….в метричному просторі C[0,1]
In [17]: ig2=g222.argmax( ); ig2                 # Визначення індексу ig2 точки, в якій
Out[17]: 4                                       ….абсолютне значення g222 максимальне
In [18]: g3=lambda x: -1-2*x+5*x**2              # Формування функції g3 як g1-g2
In [19]: g3vec=np.vectorize(g3)                  # Векторизація сформованої функції g3
In [20]: g33=g3vec(x);g33                        # Визначення векторизованої функції g33
Out[20]:                                         ….та її візуалізація
array([-1.  , -1.15, -1.2 , -1.15, -1.  , -0.75,
        -0.4 ,  0.05,  0.6 ,  1.25,  2.  ])
In 21]: g333=np.piecewise(g33,[g33<0,\           # Абсолютизація g333 значень g33
        g33>=0], [lambda g33:-g33,\              ….векторизованої функції g3
   lambda g33: g33])
In [22]: g333                                    # Візуалізація абсолютизованих значень
Out[22]: array([1.  , 1.15, 1.2 , 1.15, 1.  , 0.75,   ….g333 векторизованої функції g3
        0.4 , 0.05, 0.6 , 1.25, 2.  ])
In [23]: mg3=g333.max( );mg3                     # Обчислення метрики mg3 простору
Out[23]: 2.0                                     ….функцій C[0,1] між точками g1 та g2
In [24]: ig3=g333.argmax( );ig3                  # Визначення індексу ig3 точки, в якій
Out[24]: 10                                      ….абсолютне значення g333 максимальне
```

Кінець програми 4.

Програма мовою Python для визначати норми та метрики просторів Лебега L(0,1), елементами яких є функції

(Програма 5)

```
In [1]: import numpy as np                       # Виклик ППП pumpy під символом np
In [2]: x=np.linspace(0,1,11)                    # Внесення масиву значень аргументу x
In [3]: g1=lambda x: -1+3*x-x**2                 # Формування функції g1 лямбда-функцією
In [4]: g1vec=np.vectorize(g1)                   # Векторизація сформованої функції g1
```

```
In [5]: g11=g1vec(x)                          # Формування значень функції g11 із g1
In [6]: g11                                   # Візуалізація значень функції g11
Out[6]:
array([-1.  , -0.71, -0.44, -0.19,  0.04,  0.25,
        0.44,  0.61,  0.76,  0.89,  1.  ])
In [7]: g111=np.piecewise(g11,[g11<0,g11>=0],\  # Абсолютизація g111 значень g11
          [lambda g11:-g11,lambda g11: g11])    ….векторизованої функції g1
In [8]: g111                                  # Візуалізація абсолютизованих значень
Out[8]: array([1.  , 0.71, 0.44, 0.19, 0.04, 0.25,  ….g111 векторизованої функції g1
                0.44, 0.61, 0.76, 0.89, 1.  ])
In [9]: c1=g111.sum( )                        # Обчислення суми c1 масиву g111
In [10}: nLg1=0.1*c1;nLg1                     # Обчислення норми Лебега nLg1 для
Out[10]: 0.633                                …..функції g1 через суму Дарбу для g111
In [11]: g2=lambda x: 5*x-6*x**2              # Формування функції g2 лямбда-функцією
In [12]: g2vec=np.vectorize(g2)               # Векторизація сформованої функції g2
In [13}: g22=g2vec(x)                         # Формування значень функції g22 із g2
In [14]: g22                                  # Візуалізація значень функції g22
Out[14]:
array([ 0.  ,  0.44,  0.76,  0.96,  1.04,  1.  ,  0.84,
        0.56,  0.16,  -0.36, -1.  ])
In [15]: g222=np.piecewise(g22,[g22<0,\       # Абсолютизація g222 значень g22
         g22>=0],[lambda g22:-g22,\           ….векторизованої функції g2
         lambda g22: g22])
In [16]: g222                                 # Візуалізація абсолютизованих значень
Out[16]: array([0.  , 0.44, 0.76, 0.96, 1.04, 1.  ,  ….g222 векторизованої функції g2
                0.84, 0.56, 0.16, 0.36, 1.  ])
In [17]: c2=g222.sum( )                       # Обчислення суми c2 масиву g222
In [18]: nLg2=0.1*c2;nLg2                     # Обчислення норми Лебега nLg2 для
Out[18]: 0.712                                …..функції g1 через суму Дарбу для g222
In [19]: g3=lambda x: -1-2*x+5*x**2           # Формування функції g3 як g1-g2
In [20]: g3vec=np.vectorize(g3)               # Векторизація сформованої функції g3
In [21]: g33=g3vec(x)                         # Визначення векторизованої функції g33
In [22}: g33                                  # Візуалізація векторизованої функції g33
Out[22]:
array([-1.  , -1.15, -1.2 , -1.15, -1.  , -0.75,
        -0.4 ,  0.05  0.6 ,  1.25,  2.  ])
In [23]: g333=np.piecewise(g33,[g33<0,\       # Абсолютизація g333 значень g33
                g33>=0], [lambda g33:-g33,\   ….векторизованої функції g3
                lambda g33: g33])
In [24]: g333                                 # Візуалізація абсолютизованих значень
Out[24]: array([1.  , 1.15, 1.2 , 1.15, 1.  ,  ….g333 векторизованої функції g3
         0.75, 0.4 , 0.05, 0.6 , 1.25, 2.  ])
In [25]: c3=g333.sum( )                       # Обчислення суми c3 масиву g333
In [26]: mLg3=0.1*c3;mLg3                     # Обчислення метрики mLg3 простору
Out[26]: 1.0550000000000002                   ….функцій L[0,1] між точками g1 та g2
```

Кінець програми 5.

Розділ 2. ГІЛЬБЕРТОВІ ПРОСТОРИ, ЇХ ХАРАКТЕРИСТИКИ ТА АПРОКСИМАЦІЯ ФУНКЦІЙ В НИХ (В ПРИКЛАДАХ І ПРОГРАМАХ)

2.1 Гільбертові простори, їх характеристики та основні апроксимаційні співвідношення

У кінці однойменного підрозділу у базовому навчальному посібнику [1] серед інших стосовно гільбертових просторів сформульовані і такі питання:

1. Дайте означення гільбертового простору. Наведіть приклади гільбертових просторів та співвідношень, за допомогою яких записуються скалярні добутки в цих просторах.
2. Як пов'язані між собою метрика і норма, норма і скалярний добуток в гільбертовому просторі?
3. Що собою являє ортонормована послідовність функцій в гільбертовому просторі?
4. Які ортогональні послідовності на базі степеневих функцій ви знаєте?
5. Як апроксимувати неперервну функцію в гільбертовому просторі?

Тож із відповідей на ці питання, дотримуючись означення гільбертового простору та його характеристик, які дані в роботі [1], ми і продовжимо викладення змісту нашого «Навчального посібника для опанування студентами способів розв'язання задач з функціонального аналізу мовою Python»

Отже, ***банахів простір зі скалярним добутком елементів називають гільбертовим простором (за прізвищем математика, який його вивчав) і позначають як H-простір.***

H-простір може бути скінченновимірним і нескінченновимірним.

Скалярний добуток елементів $f,g\in H$ записують у вигляді (f,g) або $\langle f,g\rangle$.

З означення норми випливає, що для H-простору

$$||f|| = \sqrt{\langle f,f\rangle}, \tag{2.1}$$

а якщо функції комплексні, то в правій частині виразу (2.1) скалярний добуток визначається як добуток прямої функції на спряжену, оскільки лише у цьому випадку скалярний добуток буде дійсним числом, а норма, визначена через скалярний добуток, буде теж дійсним числом, що відображатиме реальну довжину вектора.

Найчастіше H-простір розглядають в двох реалізаціях.

1 Простір l_2 всіх зліченних упорядкованих послідовностей $x\in l_2$

$$x = \{x_1, x_2, \dots, x_n, \dots\}$$

таких, що мають властивість

$$\sum_{i=1}^{\infty} x_i^2 < \infty. \tag{2.2}$$

Для елементів $x^*, x^{**}\in l_2$ справедливими є нижченаведені співвідношення:

$$\wp(x^*, x^{**}) = \sqrt{\sum_{i=1}^{\infty}(x_i^* - x_i^{**})^2}, \tag{2.3}$$

$$||x^*|| = \sqrt{\sum_{i=1}^{\infty}(x_i^*)^2}, \tag{2.4}$$

$$\wp(x^*, x^{**}) = ||x^* - x^{**}||, \tag{2.5}$$

$$\langle x^*, x^{**}\rangle = \sum_{i=1}^{\infty} x_i^* x_i^{**}, \tag{2.6}$$

$$\left||x^*|\right| = \sqrt{\langle x^*, x^*\rangle}. \tag{2.7}$$

З цих співвідношень випливає, що l_2-простір – це узагальнення евклідового E_n-простору при $n \to \infty$. ***l_2-простір називають координатним гільбертовим простором.***

2. Простір $L_2[a, b]$ функцій $f(t)$ з інтегрованим квадратом, тобто, для яких

$$\int_a^b f^2(t)\, dt<\infty.$$

Для функцій $f(t)$,$g(t) \in L_2[a, b]$ справедливими є такі співвідношення:

$$\wp(f, g) = \sqrt{\int_a^b \left(f(t) - g(t)\right)^2 dt}, \tag{2.8}$$

$$\left||f|\right| = \sqrt{\int_a^b \left(f(t)\right)^2 dt}, \tag{2.9}$$

$$\wp(f, g) = \left||f - g|\right|, \tag{2.10}$$

$$\langle f, g\rangle = \int_a^b f(t)g(t)dt \tag{2.11}$$

Окремо випишемо ***дві знамениті нерівності***:
нерівність Коші–Мінковського

$$\left||f + g|\right| \le \left||f|\right| + \left||g|\right|, \tag{2.12}$$

нерівність Буняковського–Шварца

$$|\langle f, g\rangle| \le \left||f|\right| * \left||g|\right|. \tag{2.13}$$

Якщо для функцій $f(t)$, $g(t) \in H[a, b]$ виконується умова

$$\langle f, g\rangle = \int_a^b f(t)g(t)dt=0, \tag{2.14}$$

то вони є ортогональними на $[a, b]$.

Якщо для скінченновимірної чи нескінченної послідовності функцій $\{\varphi_k(t)\ \} \subset H\left[[a, b]\right], t \in [a, b], k = 0,1,2, \ldots$ ***виконується умова***

$$\langle \boldsymbol{\varphi_k}, \boldsymbol{\varphi_m}\rangle = \int_a^b \boldsymbol{\varphi_k(t)\varphi_m(t)dt} = \mathbf{0},\ \boldsymbol{k \neq m}, \tag{2.15}$$

то цю послідовність називають ортогональною, а якщо для цієї ортогональної послідовності виконується ще й умова

$$\langle \boldsymbol{\varphi_k}, \boldsymbol{\varphi_k}\rangle = \int_a^b \boldsymbol{\varphi_k^2(t)dt} = \mathbf{1}, \tag{2.16}$$

то ***цю послідовність називають ортонормованою***.

Послідовність функцій $\{\varphi_k(t)\} \subset H[[a,b]], t \in [a,b], k = 0,1,2,\ldots$ називають ортогональною з вагою $w(t)$, якщо існує така функція $w(t) \in H[a,b]$, яка забезпечує умову

$$\langle \boldsymbol{\varphi_k}, \boldsymbol{\varphi_m} \rangle = \int_a^b \boldsymbol{\varphi_k(t)\varphi_m(t)w(t)dt} = \mathbf{0},\ \boldsymbol{k \neq m}. \tag{2.17}$$

Підмножина ортогональних функцій $\{\varphi_k(t)\} \subset H[[a,b]], t \in [a,b], k = 0,1,2,\ldots$ є повною в *H*-просторі, якщо в ньому не існує відмінної від нуля функції, що була б ортогональною до якоїсь із функцій цієї послідовності.

Послідовність функцій $\{\varphi_k(t)\} \subset H[[a,b]], t \in [a,b], k = 0,1,2,\ldots$ ***називають замкнутою в H-просторі, якщо для*** $\forall\ f(t) \in H[a,b]$ ***і для*** $\forall \varepsilon > 0$ ***можна побудувати таку лінійну комбінацію з функцій*** $\boldsymbol{\varphi_k(t)}$***, взятих з вагою*** $\boldsymbol{\mu_k}$***, щоб виконувалась умова***

$$\left| |f(t) - \mu_0\varphi_0(t) - \mu_1\varphi_1(t) - \mu_2\varphi_2(t) - \cdots | \right| \leq \varepsilon, \tag{2.18}$$

яка свідчить про те, що з похибкою, котра не перевищує ε, функцію $f(t) \in H[a,b]$ на відрізку $[a,b]$ можна подати у вигляді

$$f(t) \approx \sum_{k=0}^{N} \mu_k \varphi_k\ (t), \tag{2.19}$$

де N може бути як кінцевим цілим числом, так і нескінченністю, а

$$\mu_k\ = \int_a^b f(t)\varphi_k(t)dt. \tag{2.20}$$

А далі нагадаємо, що ***під апроксимацією неперервних функцій будемо розуміти процес знаходження у вибраному просторі аналітичного опису функції, заданої елементами якоїсь множини, котра може і не бути підмножиною вибраного простору***. Наприклад, апроксимація багаточленом функції, заданої у вигляді таблиці.

Вагові коефіцієнти $\boldsymbol{\mu_k}$ ***називають коефіцієнтами Фур'є, а їх повну послідовність*** $\{\boldsymbol{\mu_k}\}$ ***називають спектром Фур'є розкладу функції*** $f(t) \in H[a,b]$ ***за ортонормованою системою функцій*** $\{\varphi_k\} \subset H[a,b]$.

Розкриваючи норму виразу (2.18) з використанням співвідношення (2.9) для гільбертового простору у формі $L_2[a,b]$, приходимо до виразу

$$\int_a^b f^2(t)\,dt = \sum_{k=0}^{\infty} \mu_k^2, \tag{2.21}$$

який ***називають рівністю Парсеваля***, корінь квадратний з обох частин якої можна інтерпретувати як довжину вектора $f(t)$ в *H*-просторі, виражену через його проекції $\{\mu_k\}$ на ортогональну систему координат $\{\varphi_k(t)\}$, яка є підмножиною цього ж *H*-простору.

Різними математиками для базових функцій

$$f_k(t) = t^k, \quad k = 0,1,2,\ldots \tag{2.22}$$

отримано різноманітні системи ортонормованих поліномів для різних функцій ваги та інтервалів ортогоналізації. Тому немає потреби кожний раз, коли необхідно апроксимувати функцію $f(t) \in H[a,b]$ за допомогою ортонормованої послідовності, самому будувати цю послідовність. Достатньо вибрати одну з тих, що побудовані іншими, скориставшись довідником з вищої математики чи посібником з математичної теорії обробки результатів експериментів.

Тож для того, щоб здійснити апроксимацію функції $f(t) \in H[a,b]$, $t \in [a,b]$ за допомогою ортонормованої системи поліномів $\{\varphi_k(t)\} \subset H[a,b]$, необхідно, виходячи з

інтервалу ортогоналізації $[a,\ b]$ та зручності вагової функції $w(t)$, вибрати ту чи іншу ортонормовану систему поліномів з довідника та знайти співвідношення для загального члена вибраної системи, розкриваючи яке для k =0,1, 2, 3, …, n отримати таку кількість її членів, якої достатньо для забезпечення заданої точності апроксимації.

Для прикладу, взятого з роботи [6], наведемо формулу для загального члена

$$P_n(t) = \frac{1}{2^n n!}\frac{d^n}{dt^n}(t^2-1)^n, \qquad n = 0,1,2,\dots,N \tag{2.23}$$

та перші 7 членів ортонормованої послідовності для поліномів Лежандра, ваговою функцією для яких є функція $w(t) = 1$, інтервалом ортогоналізації є відрізок ***[-1,1),*** нормувальний множник має вигляд $\frac{2}{2n+1}$, а при апроксимації функції $f(t) \in H[-1,\ 1]$ у вигляді (2.19) вираз (2.20) набуває вигляду

$$\mu_n = \frac{2n+1}{2}\int_{-1}^{1} f(t)P_n(t)dt, \quad n = 0,1,2,\dots,N \tag{2.24}$$

Отже, згідно з виразом (2.23), матимемо

$$\left\{\begin{array}{c} P_0(t) = 1, \\ P_1(t) = t, \\ P_2(t) = \frac{1}{2}(3t^2-1), \\ P_3(t) = \frac{1}{2}(5t^3-3t), \\ P_4(t) = \frac{1}{8}(35t^4-30t^2+3), \\ P_5(t) = \frac{1}{8}(63t^5-70t^3+15t), \\ P_6(t) = \frac{1}{16}(231t^6-315t^4+105t^2-5), \\ P_7(t) = \frac{1}{16}(429t^7-693t^5+315t^3-35t) \end{array}\right\}. \tag{2.25}$$

А для другого прикладу, теж взятого з роботи [6], наведемо формулу для загального члена

$$T_n(t) = \frac{1}{2^n}\left(\left(t+\sqrt{t^2-1}\right)^n + \left(t-\sqrt{t^2-1}\right)^n\right), \qquad n = 1,2,\dots,N \tag{2.26}$$

та перші 7 членів ортонормованої послідовності для поліномів Чебишова 1, ваговою функцією для яких є функція $w(t)=\sqrt{1-t^2}$, інтервалом ортогоналізації є відрізок **[-1,1),** нормувальний множник має вигляд π *при* $k = 0$ *та* $\frac{\pi}{2}$ *при* $k \neq 0$, а при апроксимації функції $f(t) \in H[-1,\ 1]$ у вигляді (2.19) вираз (2.20) набуває вигляду:

$$\mu_n = \frac{1}{\pi}\int_{-1}^{1} f(t)T_n(t)(1-t^2)^{-\frac{1}{2}}\,dt, \quad n = 0 \tag{2.27}$$

$$\mu_n = \frac{2}{\pi}\int_{-1}^{1} f(t)T_n(t)(1-t^2)^{-\frac{1}{2}}\,dt, \quad n = 1,2,\dots,N \tag{2.28}$$

Отже, згідно з виразом (2.26), матимемо

$$\left\{\begin{array}{c} T_0(t)=1, \\ T_1(t)=t, \\ T_2(t)=\frac{1}{2}(2t^2-1), \\ T_3(t)=\frac{1}{4}(4t^3-3t), \\ T_4(t)=\frac{1}{8}(8t^4-8t^2+1), \\ T_5(t)=\frac{1}{16}(16t^5-20t^3+5t), \\ T_6(t)=\frac{1}{32}(32t^6-48t^4+18t^2-1), \\ T_7(t)=\frac{1}{64}(64t^7-112t^5+56t^3-7t) \end{array}\right\}. \quad (2.29)$$

А завершимо ми матеріал цього підрозділу нагадуванням про те, що відома з курсу математичного аналізу апроксимація функції тригонометричним рядом Фур'є теж є прикладом апроксимації з використанням ортонормованої послідовності, що задається періодичними базовими синусними і косинусними функціями, які є ортогональними в діапазоні, заданому їх періодом.

2.2 Додаткові відомості з мови програмування Python, достатні для розв'язання задач, пов'язаних з гільбертовими просторами

Аналізуючи вирази (2.8)–(2.21), ви бачите, що обчислення метрик і норм та вагових коефіцієнтів при апроксимації функції в гільбертовому просторі з використанням програм мовою Python, вимагає знань з технології використання цієї мови, додаткових до тих, які уже викладені в підрозділі 1.2. Тож нижче у цьому підрозділі, орієнтуючись на наведені в списку використаної літератури джерела, і у першу чергу на роботи [3], [4], додамо ці додаткові знання. І ці додаткові знання ми почнемо нарощувати, починаючи з характеристики ***ППП sympy***, який містить в собі функції і методи створення програм мовою Python з використанням символьних обчислень, після виклику якого першою командою має бути команда ***symbols(' '),*** якою змінні та позначені літерами параметри в наших математичних виразах оголошуються символьними, оскільки без цього оголошення ***функції ППП sympy з*** математичними ***виразами***, які ми використовуємо, наприклад, ***в ППП numpy, працювати не будуть***. Із цих символьних змінних і параметрів створюються символьні вирази, які відповідними ***функціями ППП sympy*** можна приводити до зручного для наших подальших перетворень вигляду, наприклад, функцією **sympy.S( ),** ***або просто S( ),*** якщо ППП sympy ми уже викликали, можна будь-яку константу або стрічку ***оголосити символьною***. А функцією ***symbols ('x:n') змінній x*** присвоюються індекси в межах від 0 до *n*-1, перетворюючи її в символьну ***послідовність {x0,x1,x2,…,x(n-1)}.*** Для обчислення значення ***функції f( )*** при конкретному значенні ***змінної x*** використовується ***метод f.subs( ),*** а ***функція factor ( )*** розкладає символьний вираз, що стоїть в аргументних дужках, на символьні множники. ***Функція expand( ),*** здійснюючи усі проміжні операції, розкриває дужки символьного виразу, що стоїть в аргументних дужках, а ***функція collect( )*** збирає коефіцієнти при однакових степенях незалежної змінної. ***Функція cancel( )*** приводить суму дробово-раціональних символьних виразів до спільного знаменника і ділить чисельник та знаменник на спільний множник в разі його наявності, а функція ***apart( )*** розкладає складний символьний вираз на прості дроби. ***Функція sympify(str)*** трансформує ***стрічку*** у символьний ***вираз***, придатний для подальшого використання в ***ППП sympy***, а ***функція var( ),*** реалізуючи

ту ж функцію, що і ***symbols( ),*** дозволяє внесенням додатково в аргументні дужки заданих умов надавати обмеження символьним змінним. ***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*** комплексне число, а універсальна ***функція simplify( )*** завжди реалізує стосовно символьних виразів з комплексними числами ту вказівку, яка стоїть в аргументних дужках. Символьні матриці створюються ***функцією Matrix (***), на які після трансформації розповсюджуються усі операції матричної алгебри, а ***функції integrate( )*** здійснюють інтегрування символьно-заданої функції. Визначений інтеграл теж обчислюється ***функцією integrate( ),*** але з додаванням у аргументних дужках окрім символьної функції та її аргументу ще й границь області інтегрування, при цьому потрібно взяти до уваги, що в ППП ***sympy*** нескінченність позначається подвійною латинською літерою «o» , тобто «oo». ***Викладене вище у цьому абзаці продемонстроване в пояснювальних прикладах 17, 18 та 19.***

Пояснювальний приклад №17

```
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)                             ….змінна якої теж може бути функцією
Out[9]:                                               ….та процес підстановки цієї функції
x**4 + 4*x**2 + 2                                     ….замість змінної і візуалізація результату
In [10]: y=symbols('y:6');y                           # Індексне формування послідовності
Out[10]: (y0, y1, y2, y3, y4, y5)                     …..символьної змінної з нульового індексу
In [11]: y=symbols('y6:11');y                         # Індексне формування послідовності
Out[11]: (y6, y7, y8, y9, y10)                        ….символьної змінної з проміжного індексу
In [12]: str="x**2-4"                                 # Стрічка з математичним виразом
In [13]: expr=sympify(str);expr                       # Формування спецфункції expr
Out[13]:                                              ….перетворенням в неї стрічки
x**2 - 4
In [14]: factor(expr)                                 # Розкладення символьного виразу
Out[14]:                                              ….на множники
(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 в символьному
Out[16]:                                              ....виразі f1
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 дужок в символьному
Out[18]:                                              ….виразі f2 та його спрощення
```

```
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 в символьному
Out[20]:                                        ….виразі f3 та його спрощення
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)                                 ….змінними з обмеженнями
In [22]: expand(log(x/y))                       # Розкриття дужок в символьному
Out[22]:                                        ….виразі log та його спрощення
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 коефіцієнтів, що стоять
Out[24]:                                        ….при однакових степенях змінної x
x**2*(y - 2) + x*(z - 4)
In [25}: cancel(x/a+y/b-z/c)                    # Приведення символьних дробів до
Out[25]:                                        ….спільного знаменника
(-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)) # Приведення символьного виразу до
Out[26]:                                        ….суми найпростіших складових
-1/(5*(x + 5)) + 21/(5*x)
In [27]: M=Matrix([[a,x],[b,y]]);M              # Трансформація символьних списків
Out[27]:                                        ….у матрицю
Matrix([
[a, x],
[b, y]])
In [28]: M[0,1]                                 # Виклик із матриці елемента з 0-індексом
Out[28]:                                        ….рядка та 1-індексом стовпця
 x
```

Кінець пояснювального прикладу № 17.

Пояснювальний приклад № 18 присвячений операціям **з комплексними *числами 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',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]:                                         ….в загальному вигляді
arg(x + I*y)
```

```
In [9]: arg(2+2*I)                                  # Визначення аргументу
Out[9]:                                             ....конкретного числа z
pi/4
In [10]: z1=conjugate(z);z1                         # Формування комплексного числа z1,
Out[10]:                                            ….спряженого з числом z
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
Out[14]:                                            ….вигляді
(x + I*y)/(x - I*y)
In [15]: simplify((2+4*I)/(2-4*I))                  # Частка від ділення конкретних
Out[15]:                                            ….комплексних чисел
-3/5 + 4*I/5
In [16]: var('w')                                   # Оголошення залежної змінної w
Out[16]:                                            ….символьною
w
In [17]: w=z**4;w                                   # Формування функції w
Out[17]:                                            ….комплексної змінної z
(x + I*y)**4
In [18]: w=expand(w);w                              # Розкриття дужок в символьному
Out[18]:                                            ….комплексному виразі та його спрощення
-119 - 120*I
```

Кінець пояснювального прикладу № 18.

Пояснювальний приклад № 19 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
Out[6]:                                             ….символьної функції f1 за x
x**5/5 + x**4/2 + 4*x**3/3 + 3*x**2 + 8*x
In [7]: integrate(f2,x)                             # Однократне інтегрування
Out[7]:                                             ….символьної функції f2 за x
4*x**3*y*z/3 + x**2*(3*y**3*z**2 + y*z)
In [8]: integrate(f2,x,y)                           # Двократне інтегрування
Out[8]:                                             ….символьної функції f2 за x та y
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)                         # Трикратне інтегрування
Out[9]:                                             ….символьної функції f2 за x, y, z
x**2*y**4*z**3/4 + z**2*(x**3*y**2/3 + x**2*y**2/4)
In [10]: f3=integrate(f2,x,y,z);f3                  # Формування символьної функції f3
Out[10]:                                            ….результату інтегрування f2
x**2*y**4*z**3/4 + z**2*(x**3*y**2/3 + x**2*y**2/4)
In [11]: pprint(f3)   ⎛                  ⎞          # Виведення на екран функції f3
 2    4    3          ⎜ 3  2    2  2  ⎟             ....в іншому вигляді
x    y    z     2     ⎜x  y    x  y   ⎟
──────── + z ·⎜───── + ───── ⎟
    4           ⎝  3       4    ⎠
In [12]: integrate(f1,(x,0,1))                      # Обчислення однократного

Out[12]:                                            ….інтеграла від функції f1
391/30                                              …..по x в межах від 0 до 1
In [13]: integrate(f2,(x,0,1),(y,0,2),(z,-1,1))     # Обчислення трикратного
Out[13]:                                            ….інтеграла від функції f2
8                                                   ….по x,y,z в заданих межах
In [14]: integrate(log(x)**3,x)                     # Однократне інтегрування
Out[14]:                                            ….складної логарифмічної
x*log(x)**3 - 3*x*log(x)**2 + 6*x*log(x) - 6*x      ….функції за x
In [15]: integrate(log(x)**3,(x,-1,1))              # Обчислення інтеграла від складної
Out[15]:                                            ….логарифмічної функції за x
-12 + 3*pi**2 - I*pi**3 + 6*I*pi                    ….в межах від -1 до 1
In [16]: integrate(log(x)**3,(x,1,2))               # Обчислення інтеграла від складної
Out[16]:                                            ….логарифмічної функції за x
-6 - 6*log(2)**2 + 2*log(2)**3 + 12*log(2)          ….в межах від 1 до 2
In [17]: integrate(x**3*y**2,(y,0,x**2),(x,0,1))    # Обчислення двократного
Out[17]:                                            ….інтеграла у випадку змінної
1/30                                                ….внутрішньої границі
In [18]: integrate(exp(-2*x**2),(x,0,oo))           # Обчислення однократного
Out[18]:                                            ….інтеграла у випадку границі
sqrt(2)*sqrt(pi)/4                                  ….у вигляді нескінченності
In [19]: integrate(exp(-x**2/4-y**2),(x,0,oo),(y,-oo,oo))  # Обчислення двократного
Out[19]:                                            ….інтеграла в нескінченних
pi                                                  ….границях
```

Кінець пояснювального прикладу № 19.

А далі, використовуючи інформацію, почерпнуту з роботи [4], пояснимо, як з використанням мови Python будувати графіки на координатній площині. Як ми уже відзначали у вступі, ***графічна частина*** пайтонівського програмного середовища ***Anaconda*** зосереджена, в основному, ***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 введення. ***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 при числовому заданні 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 пояснювальних прикладах 20, 21 та 22.***

Пояснювальний приклад № 20 (побудова графіка функції, заданої списком)

```
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
Out[4]: [<matplotlib.lines.Line2D at 0x17818da8400>]
```

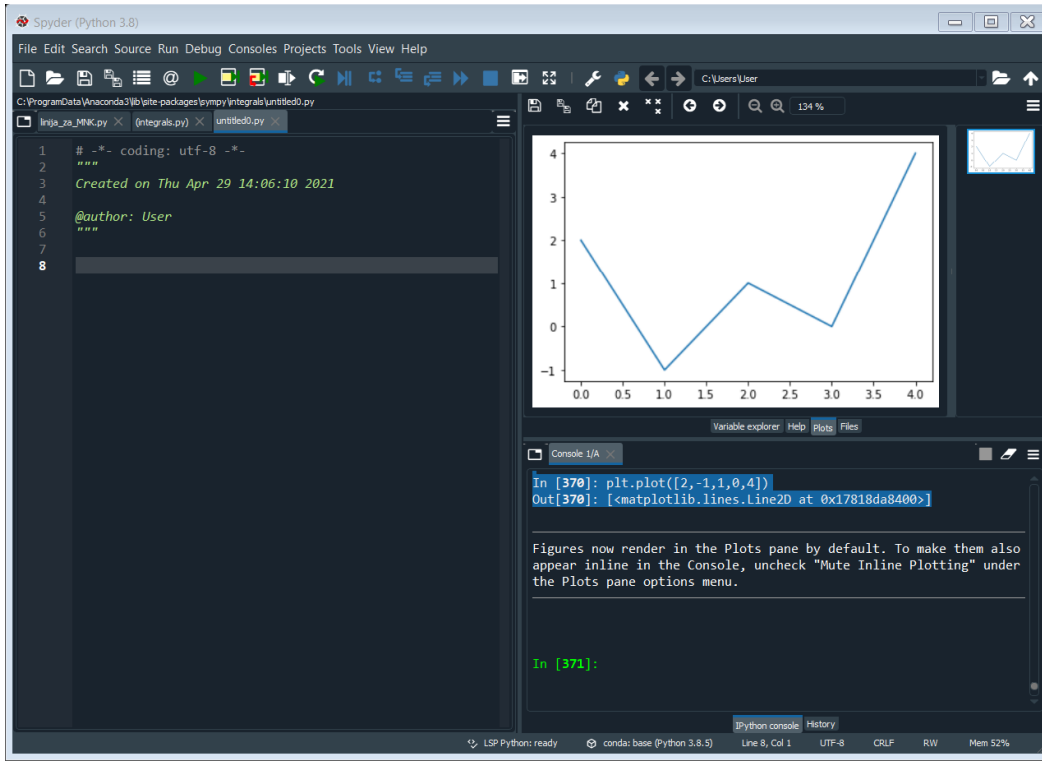

Рисунок 1 – Екран комп'ютера з фігурою1, на якій зображено графік математичної функції, заданої масивом у формі списку

Кінець пояснювального прикладу № 20.

Пояснювальний приклад № 21 (побудова графіка функції заданої циклом):

```
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
Out[376]: [<matplotlib.lines.Line2D
at 0x17818e7e4f0>]
```

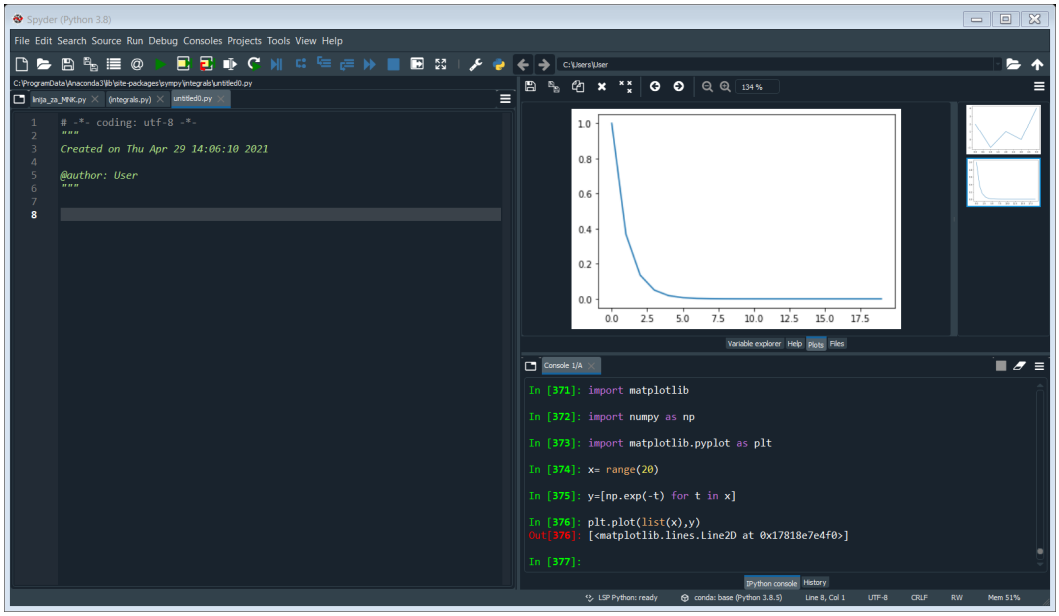

Рисунок 2 – Екран комп'ютера з фігурою 2, на якій зображено графік математичної функції, заданої масивом у формі циклу

Кінець пояснювального прикладу № 21.

Пояснювальний приклад № 22 (побудова на одному рисунку графіків двох функцій)

```
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
Out[6]:
[<matplotlib.lines.Line2D at 0x17818ee1a90>,
```

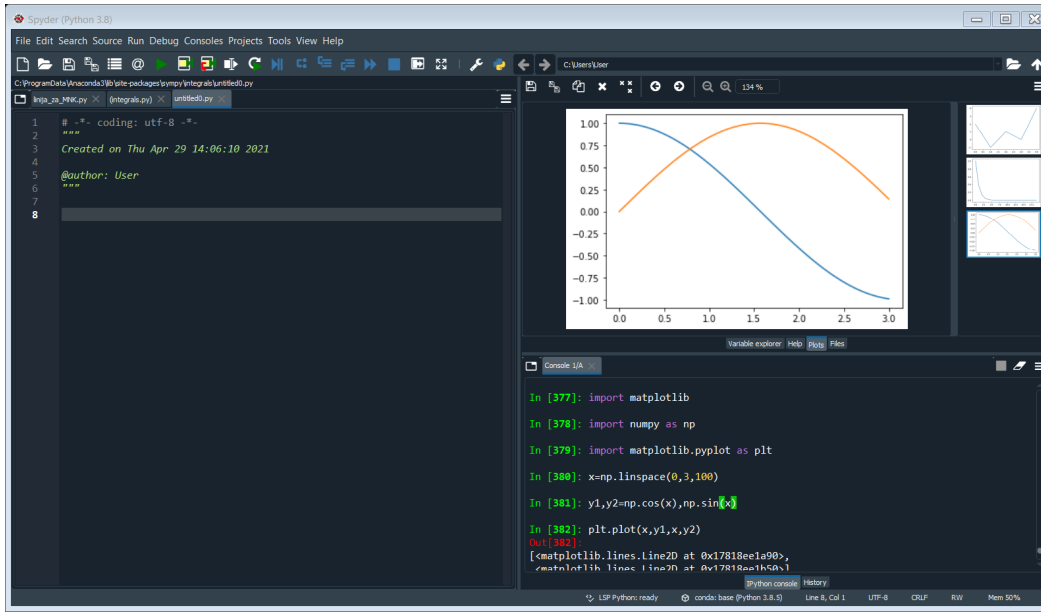

Рисунок 3 – Екран комп’ютера з фігурою 3, на якій зображено графіки двох математичних функцій, заданих в аналітичній формі

Кінець пояснювального прикладу № 22.

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, взявши 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; ***розривної*** – ***«--»***, 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 пояснювальних прикладах 23 та 24.***

Пояснювальний приклад № 23

```
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 та з
Out[8]:                                  ….заданою товщиною ліній
[<matplotlib.lines.Line2D at 0x17819f0b0a0>,
 <matplotlib.lines.Line2D at 0x17819f0b040>,
 <matplotlib.lines.Line2D at 0x17819f0b1f0>]
```

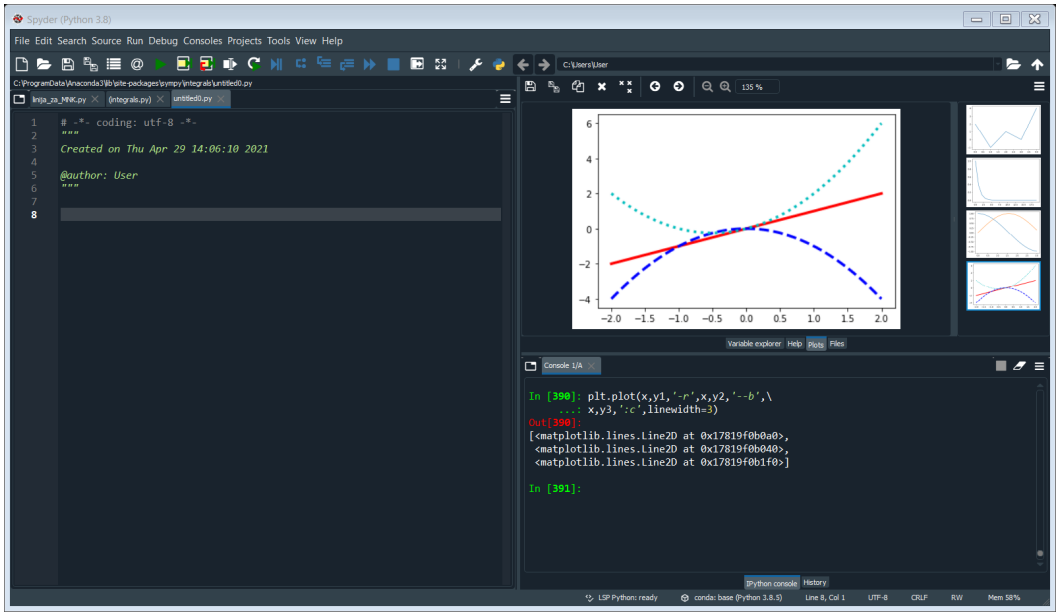

Рисунок 4 – Екран комп’ютера з фігурою 4, на якій лініями в різних кольорах і стилях зображено графіки трьох математичних функцій, заданих в аналітичній формі

Кінець пояснювального прикладу № 23.

Пояснювальний приклад № 24

```
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)                    ….з заданими стилем і кольором та
Out[7]:                                  ….товщиною ліній і маркерами
[<matplotlib.lines.Line2D at 0x1781a1c32e0>,
 <matplotlib.lines.Line2D at 0x1781a1c3340>]
```

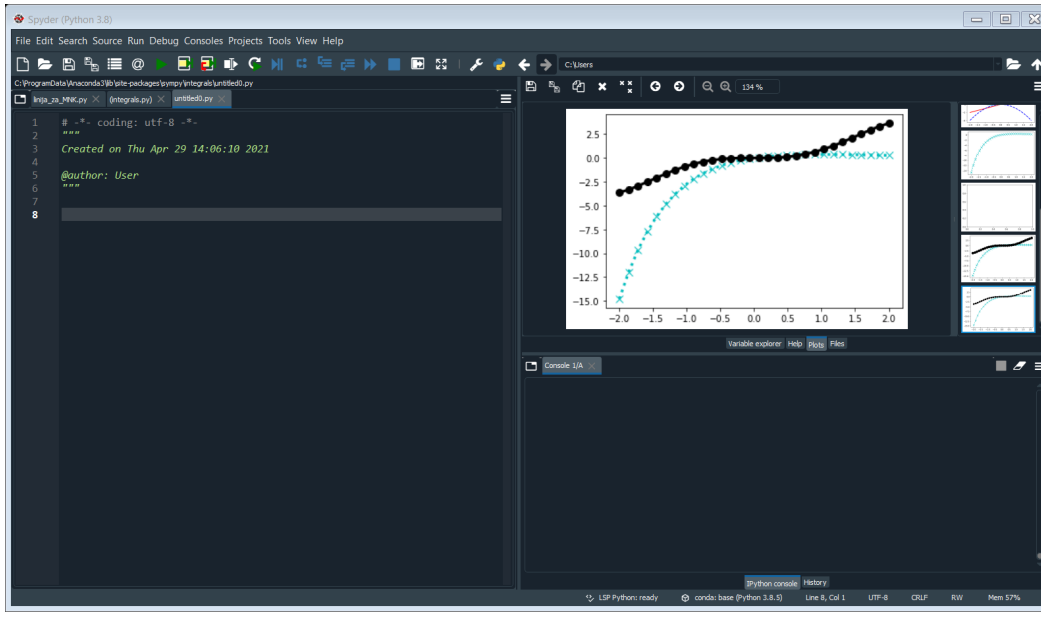

Рисунок 5 – Екран комп’ютера з фігурою 5, на якій лініями в різних кольорах і стилях та з нанесенням маркерів у формі кола та хреста зображено графіки двох математичних функцій, заданих в аналітичній формі

Кінець пояснювального прикладу № 24.

А на завершення першого знайомства з графічними можливостями ППП matplotlib, ***покажемо***, знову ж таки, використовуючи інформацію, викладену в роботі [4], ***як створювати написи на графіках***. Звертаємо увагу на те, що в усіх наведених вище прикладах побудови графіків ми не бачимо ніяких написів ні на графіках, ні на осях координат – це тому, що в програми побудови цих графіків ми не вносили функції, які реалізують написи, і частина яких має бути вписаною в програму до функції plt.plot( ), а частина – після. ***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***. ***Написи 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***, а ***розмістити*** текст ***під кутом*** можна за допомогою аргументу ***rotation. В 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 біля тексту примітки і закінчується в точці поля, якої вона стосується. В координатних дужках цієї функції вказується текст примітки, координати точки, до якої дотягується стрілка, місце розташування примітки та відстань від кінця стрілки до точки поля, куди вона направлена. А оскільки ***ППП matplotlib*** не містить шрифтів з кириличними символами, то при використанні для набору тексту українською мовою ***Windows***, який у варіантах шрифтів Arial, Times New Roman, Tahoma підтримує і кирилицю, необхідно ***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 й українські слова, поставити ***юнікодовий символ «u». Викладене*** вище у цьому абзаці проілюстровано ***в пояснювальному прикладі 25***, основою якого є приклад, наведений з цією ж метою в роботі [4]. 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 ***в пояснювальному прикладі 26***.

Читаючи абзац, наведений вище, бачимо, що при складанні програми, яка буде реалізовувати всі команди створення графіків та написів на них, легко можна припуститись помилок, для виправлення яких потрібно всю програму переписувати спочатку. Тому краще такі ***громіздкі програми*** складати ***у вигляді файлу***, який 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. ***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 в-подальшому ***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, коло 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 пропусків 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 всі 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.

Пояснювальний приклад № 25

Файл «*Прикладнадпису.ру*»

```
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'$\frac{1}{\sigma\sqrt{2\pi}}\,e^{-\frac{(x-\
    mu)^2}{2\sigma^2}}$'                    # Створення стрічки
x=np.linspace(-3,5,40)                      # Внесення масиву значень аргументу x
sigma=1                                     # Внесення значення параметра σ
mu=1                                        # Внесення значення параметра m
y=(1/(sigma*np.sqrt(2*np.pi)))*np.exp(-(x-
    mu)**2/(2*sigma**2))                    # Створення функції
fig=plt.figure(facecolor='white')           # Створення поля (фігури) під рисунок
plt.plot(x,y,'-bo',linewidth=3,markersize=10,
    label=str)                              # Побудова графіка функції
plt.legend(fontsize=18,loc='upper left')    # Команда на створення написів
ax=fig.gca()                                # Прив'язка до поля( фігури) рисунку
plt.title(r'$\mu=1,\sigma=1$')              # Заголовок (з використанням параметрів)
plt.text(0.58,.95,r'Графік функції $\varphi(x)$',  # Створення тексту справа від кривої
    horizontalalignment='left',verticalalignment\
    ='center', transform=ax.transAxes,fontsize=16)
ax.annotate('Максимум',xy=(1,0.4),xytext=(-2,0.25), # Створення примітки з прив'язкою
     arrowprops=dict(facecolor=\
     'green',shrink=0.05))                  ….тексту і кінця стрілки
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'Х-вісь абсцис',{'fontname':'Times \
     New Roman'})                           # Напис під віссю абсцис
plt.ylabel(r'$\varphi(x)$-ордината')        # Напис вздовж ординати
```

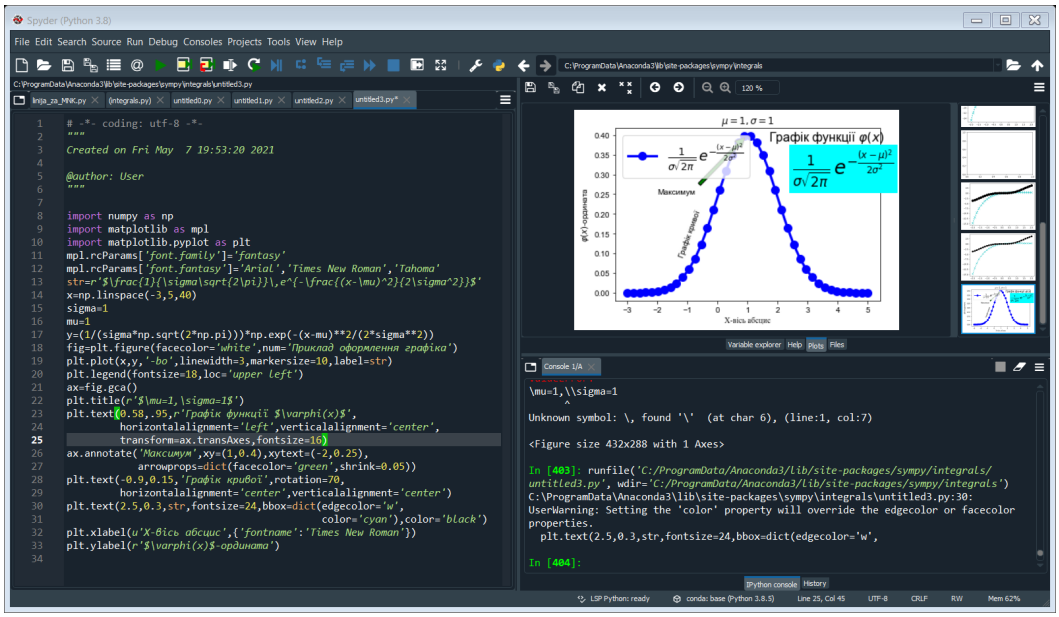

Рисунок 6 – Екран комп'ютера з файлом «***Прикладнадпису»*** та фігурою 6, на якій зображено графік «дзвінницевої» функції, якою задається густина ймовірності нормального закону розподілу випадкової величини, ***з нанесенням надписів,*** визначених програмою, записаною у файлі, на графіку

Кінець пояснювального прикладу № 25.

Пояснювальний приклад №26

Файл *«СуміщенняГрафіків»*

```
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
```

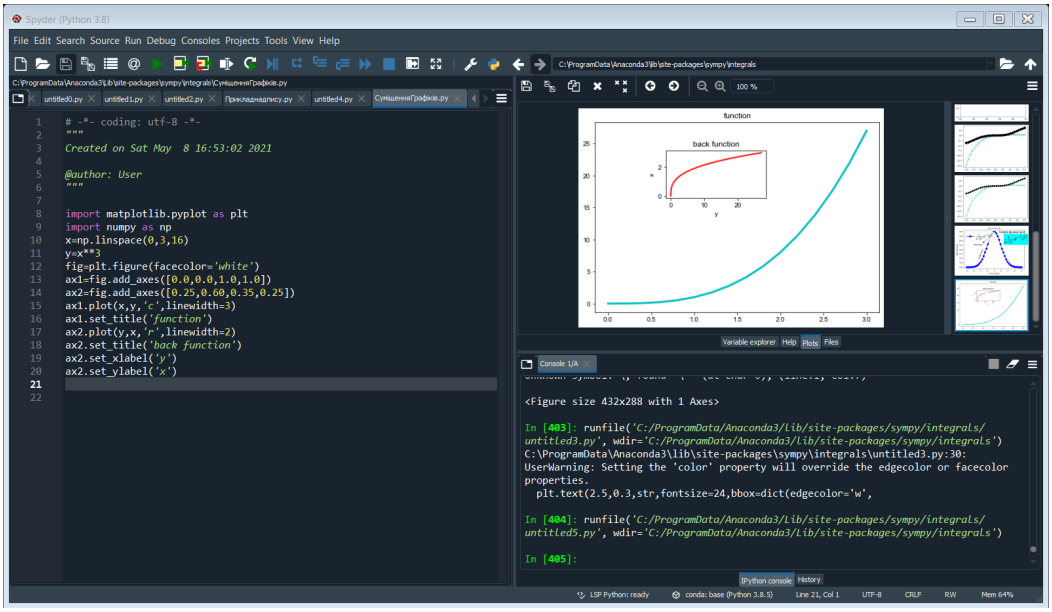

Рисунок 7 – Екран комп'ютера з файлом «***СуміщенняГрафіків***» та фігурою 7, на якій зображено графіки функції та функції, до неї оберненої, з нанесенням надписів на графіках, визначених програмою, записаною у файлі, в якій ***використані методи fig.add_axes( ),*** якими рисунок виконується в прямокутних областях, визначених відносними координатами та відносними розмірами

Кінець пояснювального прикладу №26.

2.3 Задачі з визначення характеристик гільбертових просторів та апроксимації функцій в цих просторах в програмах мовою Python

Почнемо з програмування процесів визначення таких характеристик ***гільбертового простору l_2 всіх зліченних упорядкованих послідовностей $x \in l_2$***, як ***норма, метрика*** та ***скалярний добуток.*** Але оскільки ***гільбертів простір l_2*** одночасно є і банаховим простором, то визначати його ***норму*** і ***метрику*** можна, використовуючи ***програму 3***, створену нами раніше для визначення цих же характеристик банахових числових просторів. А тому до тієї ***програми 3*** ми долучимо ще лише ***програму 6*** для обчислення ***скалярного добутку*** у цьому просторі, використавши для узгодження цих програм між собою ***масиви чисел***, що використані і в ***програмі 3***. Отже,

Програма мовою Python для визначення скалярного добутку у гільбертових просторах l_2, елементами яких є числа

(Програма 6)

```
In [1]: import numpy as np                  # Виклик ППП пипру під символом пр
In [2]: a4=np.array([1,2,3,4])              # Внесення масиву проєкцій точки а4
In [3]: c4=np.array([4,3,2,1])              # Внесення масиву проєкцій точки с4
```

```
In [4]: x4=a4*c4;x4                    # Формування поелементного добутку
Out[4]: array([4, 6, 6, 4])            …..x4 масивів a4 та c4
In [5]: sd=x4.sum( );sd                # Обчислення скалярного добутку
Out[5]: 20                             ….sd в просторі l_2
```

Кінець програми 6.

Програма мовою Python для визначення норми, метрики та скалярного добутку у гільбертових просторах $L_2[0, 1]$, елементами яких є функції дійсної змінної

(Програма 7)

```
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=5*x**3-3*x**2+2*x-4         # Формування символьної функції f1
In [5]: f2=1+2*x+3*x**3                # Формування символьної функції f2
In [6]: f3=f1*f1;f3                    # Формування символьної функції f3,
Out[6]:                                ….яка є квадратом f1, і її візуалізація
(5*x**3 - 3*x**2 + 2*x - 4)**2
In [7]: f4=expand(f3);f4               # Розкриття дужок в символьному
Out[7]:                                ….виразі f3 та позначення його як f4
25*x**6 - 30*x**5 + 29*x**4 - 52*x**3 +\
 28*x**2 - 16*x + 16
In [8]: a=symbols('a')                 # Оголошення a символьним
In [9]: a=integrate(f4,(x,0,1))        # Обчислення однократного
In [10]: a                             …. інтеграла a від функції f4
Out[10]:                               ….за x в межах від 0 до 1
914/105
In [11]: nf1=a**0.5;nf1                # Обчислення норми nf1
Out[11]:                               ….функції f1 та її візуалізація
2.9503833487806
In [12]: f5=f2**2;f5                   # Формування символьної функції f5,
Out[12]:                               ….яка є квадратом f2, і її візуалізація
(3*x**3 + 2*x + 1)**2
In [13]: f6=expand(f5);f6              # Розкриття дужок в символьному
Out[13]:                               ….виразі f5 та позначення його як f6
9*x**6 + 12*x**4 + 6*x**3 + 4*x**2 + 4*x + 1
In [13]: b=symbols('b')                # Оголошення b символьним
In [14]: b=integrate(f6,(x,0,1))       # Обчислення однократного
In [15]: b                             ….інтеграла  b від функції f6
Out[15]:                               ….за x в межах від 0 до 1
1999/210
In [16]: nf2=b**0.5;nf2                # Обчислення норми nf2
Out[16]:                               ….функції f2 та її візуалізація
3.08529538602832
In [17]: f7=f1*f2;f7                   # Формування символьної функції f7,
Out[17]:                               ….яка є добутком f1 та f2, та її
(3*x**3 + 2*x + 1)*(5*x**3 - 3*x**2 + 2*x - 4)  ….візуалізація
In [18]: f8=expand(f7);f8              # Розкриття дужок в символьному
Out[18]:                               ….виразі f7 та позначення його як f8
```

```
15*x**6 - 9*x**5 + 16*x**4 - 13*x**3 + \
x**2 - 6*x - 4
In [19]: sd=integrate(f8,(x,0,1))          # Обчислення скалярного добутку sd
In [20]: sd                                ….функцій f1 та f2 та його
Out[20]:                                   ….візуалізація
-2551/420
In [21]: f9=f1-f2;f9                       # Формування символьної функції f9,
Out[21]:                                   ….яка є різницею f1 та f2, та її
2*x**3 - 3*x**2 – 5                        ….візуалізація
In [22]: f10=f9**2;f10                     # Формування символьної функції f10,
Out[22]:                                   ….яка є квадратом f9, і її візуалізація
(2*x**3 - 3*x**2 - 5)**2
In [23]: f11=expand(f10);f11               # Розкриття дужок в символьному
Out[23]:                                   ….виразі f10 та позначення його як f11
4*x**6 - 12*x**5 + 9*x**4 - 20*x**3 + \
30*x**2 + 25
In [24]: d=symbols('d')                    # Оголошення d символьним
In [25]: d=integrate(f11,(x,0,1))          # Обчислення однократного
In [26]: d                                 ….інтегралу d від функції f11
Out[26]:
1063/35
In [27]: mf1f2=d**0.5;mf1f2                # Обчислення метрики mf1f2
Out[27]:                                   ….простору L2[0, 1] між
5.51102790515785                           ….його точками f1 та f2
```

Кінець програми 7.

Програма мовою Python для визначення норми, метрики та скалярного добутку в гільбертових просторах $L_2[0, 1]$, елементами яких є функції комплексної змінної (Програма 8)

```
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]: f1z=5*z**3-3*z**2+2*z-4            # Формування символьної функції f1z
In [6]: f2z=1+2*z+3*z**3                   # Формування символьної функції f2z
In [7]: f3z=expand(f1z)                    # Розкриття дужок у виразі f1z
In [8]: f4z=collect(f3z,I)                 # Приведення виразу f1z до re( )+I*im( )
In [9]: f5z=conjugate(f4z)                 # Спряження виразу f1z до re( )-I*im( )
In [10]: f6z=f4z*f5z                       # Добуток прямого і спряженого виразів
In [11]: az,bz=symbols(‘az bz’)            # Оголошення az,bz символьними
In [12]: az=integrate(f6z,(x,0,1),(y,0,1));az  # Обчислення подвійного інтеграла від
Out[12]:                                   ….добутку прямого і спряженого виразів
2668/105                                   ….за x та y в межах від 0 до 1
In [13]: nf1z=az**0.5;nf1z                 # Обчислення норми функції
Out[13]:                                   ….комплексної змінної f1z
5.04078603092056                           ….та її візуалізація
In [14]: q3z=expand(f2z)                   # Розкриття дужок у виразі f2z
In [15]: q4z=collect(q3z,I)                # Приведення виразу f2z до re( )+I*im( )
In [16]: q5z=conjugate(q4z)                # Спряження виразу f2z до re( )-I*im( )
```

```
In [17]: q6z=q4z*q5z                                  # Добуток прямого і спряженого виразів
In [18]: bz=integrate(q6z,(x,0,1),(y,0,1));bz         # Обчислення подвійного інтеграла від
Out[18]:                                              ….добутку прямого і спряженого виразів
2171/210                                              ….за x та y в межах від 0 до 1
In [19]: nf2z=bz**0.5;nf2z                            # Обчислення норми функції
Out[19]:                                              ….комплексної змінної f2z
3.21529084813415                                      ….та її візуалізація
In [20]: fq=f4z*q5z                                   # Добуток прямого і спряженого виразів
In [21]: fq1=expand(fq)                               # Розкриття дужок у виразі fq
In [22]: fq2=collect(fq1,I)                           # Приведення виразу fq1 до re( )+I*im( )
In [23]: sdf1f2=integrate(fq2,(x,0,1),(y,0,1))        # Обчислення скалярного добутку
In [24]: sdf1f2                                       ….sdf1f2 функцій f1z та f2z
Out[24]:                                              ….і його візуалізація
547/420 + 223*I/20
In [25]: f12z=f1z-f2z                                 # Формування різниці функцій f1z та f2z
In [26]: f13z=expand(f12z)                            # Розкриття дужок у виразі f12z
In [8]: f14z=collect(f13z,I)                          # Приведення виразу f12z до re( )+I*im( )
In [9]: f15z=conjugate(f14z)                          # Спряження виразу f12z до re( )-I*im( )
In [10]: f16z=f14z*f15z                               # Добуток прямого і спряженого виразів
In [11]: abz=symbols('abz')                           # Оголошення abz символьним
In [12]: abz=integrate(f16z,(x,0,1),(y,0,1));abz      # Обчислення подвійного інтеграла від
Out[12]:                                              ….добутку прямого і спряженого виразів
232/7                                                 ....за x та y в межах від 0 до 1
In [13]: mf12z=abz**0.5;mf12z                         # Обчислення метрики mf12z простору
Out[13]:                                              ….L2[0,1] функцій комплексної змінної
5.75698333703140                                      ….між точками f1z та f2z її візуалізація
```

Кінець програми 8.

Програма мовою Python для апроксимації функцій $f_1(x) = x^3 - 3x^2 + 2x - 4$, $f_2(x) = exp(x) - 2x$ дійсної змінної x, заданих у гільбертовому просторі $L_2[-3, 3]$, зваженими сумами поліномів Лежандра $P_n(t)$, ортонормованих на відрізку $[-1, 1]$

(Програма 9)

```
In [1]: import sympy                                  # Виклик ППП sympy
In [2]: from sympy import *                           # Доступ в sympy до усіх функцій
In [3]: x,t,P=symbols('x t P')                        # Оголошення x,t,P символьними
In [4]: f1x=x**3-3*x**2+2*x-4                         # Формування функції f1 (x), x ∈ [−3,3]
In [5]: f1t=27*t**3-27*t**2+6*t-4                     # Трансформація f1(x) у f1(t), t ∈ [−1,1]
In [6]: P0=1                                                # Формування полінома P0
In [7]: P1=t                                                # Формування полінома P1
In [8]: P2=(3*t**2-1)/2                                     # Формування полінома P2
In [9]: P3=(5*t**3-3*t)/2                                   # Формування полінома P3
In [10]: P4=(35*t**4-30*t**2+3)/8                           # Формування полінома P4
In [11]: P5=(63*t**5-70*t**3+15*t)/8                        # Формування полінома P5
In [12]: P6=(231*t**6-315*t**4+105*t**2-5)/16               # Формування полінома P6
In [13]: P7=(429*t**7-693*t**5+315*t**3-35*t)/16            # Формування полінома P7
In [14]: q0=f1t*P0*1/2                                # Підінтегральний вираз для μ0
In [15]: q1= f1t*P1*3/2                               # Підінтегральний вираз для μ1
In [16]: q2= f1t*P2*5/2                               # Підінтегральний вираз для μ2
In [17]: q3= f1t*P3*7/2                               # Підінтегральний вираз для μ3
In [18]: q4= f1t*P4*9/2                               # Підінтегральний вираз для μ4
```

```
In [19]: q5= f1t*P5*11/2                    # Підінтегральний вираз для μ5
In [20]: q6= f1t*P6*13/2                    # Підінтегральний вираз для μ6
In [21]: q7= f1t*P7*15/2                    # Підінтегральний вираз для μ7
In [22]: mju=symbols('mju')                 # Оголошення mju символьною
In [23]: mju0=integrate(q0,(t,-1,1));mju0   # Обчислення коефіцієнта μ0
Out[23]:
-13
In [24]: mju1=integrate(q1,(t,-1,1));mju1   # Обчислення коефіцієнта μ1
Out[24]:
111/5
In [25]: mju2=integrate(q2,(t,-1,1));mju2   # Обчислення коефіцієнта μ2
Out[25]:
-18
In [26]: mju3=integrate(q3,(t,-1,1));mju3   # Обчислення коефіцієнта μ3
Out[26]:
54/5
In [27}: mju4=integrate(q4,(t,-1,1));mju4   # Обчислення коефіцієнта μ4
Out[27]:
0
In [28]: mju5=integrate(q5,(t,-1,1));mju5   # Обчислення коефіцієнта μ5
Out[28]:
0
In [29]: mju6=integrate(q6,(t,-1,1));mju6   # Обчислення коефіцієнта μ6
Out[29]:
0
In [30]: mju7=integrate(q7,(t,-1,1));mju7   # Обчислення коефіцієнта μ7
Out[30]:
0
In [31]: su=Function('su')(t)               # Оголошення su символьною
In [32]: su01=mju0*P0+mju1*P1               # Апроксимація функції f1t
In [33]: f1t1=expand(su01);f1t1             ….сумою двох
Out[33]:                                    ….поліномів
111*t/5 - 13
In [34]: su02=su01+mju2*P2                  # Апроксимація функції f1t
In [35]: f1t2=expand(su02);f1t2             ….сумою трьох
Out[35]:                                    ….поліномів
-27*t**2 + 111*t/5 - 4
In [36]: su03=su02+mju3*P3                  # Апроксимація функції f1t
In [37]: f1t3=expand(su03);f1t3             ….сумою чотирьох
Out[37]:                                    ….поліномів
27*t**3 - 27*t**2 + 6*t - 4
In [38]: f1t12=f1t-f1t1                     # Обчислення похибки
In [39]: su11=integrate(f1t12*f1t12,(t,-1,1));\   ….апроксимації функції f1t
         su11
Out[39]:                                    ….сумою двох поліномів
28512/175                                   ….через норму різниці між
In [40]: nf1t21=su11**0.5;nf1t21            ….функцією та сумою
Out[40]:                                    ….двох поліномів
12.7642357501620                            # Значення похибки
In [41]: f1t22=f1t-f1t2                     # Обчислення похибки
```

```
In [42]: su12=integrate(f1t22*f1t22,(t,-1,1));su12   ….апроксимації функції f1t
Out[42]:                                             ….сумою трьох поліномів
5832/175                                             ….через норму різниці між
In [43]: nf1t22=su12**0.5;nf1t22                     ….функцією та сумою
Out[43]:                                             ….трьох поліномів
5.77284282530837                                     # Значення похибки
In [44]: f1t32=f1t-f1t3                              # Обчислення похибки
In [45]: su13=integrate(f1t32*f1t32,(t,-1,1));su13   ….апроксимації функції f1t
Out[45]:                                             ….сумою чотирьох поліномів
0                                                    ….через норму різниці між
In [46]: nf1t32=su13**0.5;nf1t32                     ….функцією та сумою
Out[46]:                                             ….чотирьох поліномів
0                                                    # Значення похибки
In [47]: f2x=sin(x)-2*x                              # Формування функції f_2 (x), x ∈ [−3,3]
In [48]: f2t=sin(3*t)-6*t                            # Трансформація f_2(x) у f_2(t), t ∈ [−1,1]
In [49]:  q02=f2t*P0*1/2                                    # Підінтегральний вираз для h_0
In [50]: q12=f2t*P1*3/2                                     # Підінтегральний вираз для h_1
In [51]: q22=f2t*P2*5/2                                     # Підінтегральний вираз для h_2
In [52]: q32=f2t*P3*7/2                                     # Підінтегральний вираз для h_3
In [53]: q42=f2t*P4*9/2                                     # Підінтегральний вираз для h_4
In [54]: q52=f2t*P5*11/2                                    # Підінтегральний вираз для h_5
In [55]: q62=f2t*P6*13/2                                    # Підінтегральний вираз для h_6
In [56]: q72=f2t*P7*15/2                                    # Підінтегральний вираз для h_7
In [57]: h=symbols('h')                                     # Оголошення h символьною
In [58}; h0=integrate(q02,(t,-1,1));h0                      # Обчислення коефіцієнта h_0
Out[58]:
0
In [59]: h1=integrate(q12,(t,-1,1));h1                      # Обчислення коефіцієнта h_1
Out[59]:
-6 + sin(3)/3 - cos(3)
In [60]: import math                                        # Виклик ППП math
In [61]: from math import *                                 # Доступ в math до усіх функцій
In [62]: sin(3)                                             # Обчислення sin(3) в ППП math
Out[62]:
0.1411200080598672
In [63]: cos(3)                                             # Обчислення cos(3) в ППП math
Out[63]:
 -0.9899924966004454
In [64]: h1=-6 + sin(3)/3 - cos(3);h1                       # Дообчислення коефіцієнта h_1
Out[64]:
 -4.963
In [65]: h2=integrate(q22,(t,-1,1));h2                      # Обчислення коефіцієнта h_2
Out[65]:
0
In [66]: h3=integrate(q32,(t,-1,1));h3                      # Обчислення коефіцієнта h_3
Out[66]:
14*cos(3)/9 + 91*sin(3)/27
In [67]: h3=14*cos(3)/9 + 91*sin(3)/27;h3                   # Дообчислення коефіцієнта h_3
Out[67]:
-1.029
```

```
In [68]: h4=integrate(q42,(t,-1,1));h4                    # Обчислення коефіцієнта h4
Out[68]:
0
In [69]: h5=integrate(q52,(t,-1,1));h5                    # Обчислення коефіцієнта h5
Out[69]:
-220*sin(3)/9 - 11*cos(3)/3
In [70]: h5=-220*sin(3)/9 - 11*cos(3)/3;h5                # Дообчислення коефіцієнта h5
Out[70]:
0.18333333333333401
In [71]: h6=integrate(q62,(t,-1,1));h6                    # Обчислення коефіцієнта h6
Out[71]:
0
In [72]: h7=integrate(q72,(t,-1,1));h7                    # Обчислення коефіцієнта h7
Out[72]:
1685*cos(3)/27 + 35455*sin(3)/81
In [73]: h7=1685*cos(3)/27 + 35455*sin(3)/81;h7           # Дообчислення коефіцієнта h7
Out[783]:
-0.06537037037038118
In [74]: f2t13=h1*P1+h3*P3;f2t13                          # Апроксимація функції f2t
Out[74]:                                                  ….сумою двох
-2.5725*t**3 - 3.4195*t                                   ….поліномів
In [75]: f2t15=f2t13+h5*P5;f2t15                          # Апроксимація функції f2t
Out[75]:                                                  ….сумою трьох
1.44375000000001*t**5 - 4.17666666666667*t**3\            ….поліномів
- 3.07575*t
In [76]: f2t17=f2t15+h7*P7;f2t17                          # Апроксимація функції f2t
Out[76]:                                                  ….сумою чотирьох
-1.75274305555585*t**7 + 4.27510416666714*t**5 \          ….поліномів
- 5.46364583333355*t**3 - 2.93275231481479*t
In [77]: f2t13t=f2t-f2t13                                 # Обчислення похибки
In [78]: su21=integrate(f2t13t*f2t13t,(t,-1,1))           ….апроксимації функції f2t
In [79]: nf2t13t=su21**0.5;nf2t13t                        ….сумою двох поліномів
Out[79]:                                                  ….через норму їх різниці
1.51569565986491*(cos(3) + 0.435287289611143*sin(3)**2 -
- 0.145095763203714*sin(3)*cos(3) + 0.662023602244148*sin(3)
+ 0.435287289611143*cos(3)**2 + 0.44374869413813)**0.5
In [80]: 1.51569565986491*(cos(3) + 0.435287289611143*sin(3)**2 -
0.145095763203714*sin(3)*cos(3) + 0.662023602244148*sin(3) +
 0.435287289611143*cos(3)**2 + 0.44374869413813)**0.5
Out[80]:
0.07932847177751723                                       # Значення похибки
In [81]: f2t15t=f2t-f2t15                                 # Обчислення похибки
In [82]: su23=integrate(f2t15t*f2t15t,(t,-1,1))           ….апроксимації функції f2t
In [83]: nf2t15t=su23**0.5;nf2t15t                        ….сумою трьох поліномів
Out[83]:                                                  ….через норму їх різниці
1.77497000496305*(0.80678077682686*cos(3) +
+0.317408069218471*sin(3)**2 - 0.10580268973949*sin(3)*cos(3) +
+ sin(3) + 0.317408069218471*cos(3)**2 + 0.325517774851875)**0.5
In [84]: 1.77497000496305*(0.80678077682686*cos(3) +
+ 0.317408069218471*sin(3)**2 - 0.10580268973949*sin(3)*cos(3) +
```

```
+sin(3) + 0.317408069218471*cos(3)**2 + 0.325517774851875)**0.5
Out[84]:
0.019475579014104023                          # Значення похибки
In [85]: f2t17t=f2t-f2t17                     # Обчислення похибки
In [86]: su25=integrate(f2t17t*f2t17t,(t,-1,1))   ….апроксимації функції f2t
In [87]: nf2t17t=su25**0.5;nf2t17t            ….сумою чотирьох поліномів
Out[87]:                                      ….через норму їх різниці
3.28341711065988*(0.33667821186518*cos(3) +
+ 0.0927572545616921*sin(3)**2 - 0.0309190848538974*sin(3)*cos(3) +
0.0927572545616922*cos(3)**2 + 0.0951800322316149 + sin(3))**0.5
In [88]: 3.28341711065988*(0.33667821186518*cos(3) +
+0.0927572545616921*sin(3)**2 - 0.0309190848538974*sin(3)*cos(3) +
0.0927572545616922*cos(3)**2 + 0.0951800322316149 + sin(3))**0.5
Out[88]:
0.027081143225896158                          # Значення похибки
In [89]: import numpy as np                   # Виклик ППП numpy як np
In [90]: import matplotlib as mpl             # Виклик ППП matplotlib як mpl
In [91]: import matplotlib.pyplot as plt      # Виклик модуля mpl.pyplot як plt
In [92]: mpl.rcParams['font.family']='fantasy'   # Формування бібліотеки шрифтів
In [93]: mpl.rcParams['font.fantasy']='Arial','Times \
         New Roman','Tahoma'
In [94]: expr1=27*x**3-27*x**2+6*x-4          # Формування функції, що
In [95]: expr2=111*x/5 – 13                   ….апроксимується, і її наближень
In [96]: expr3=-27*x**2 + 111*x/5 – 4         ….у ППП sympy у формі expr ,
In [97]: expr4=27*x**3-27*x**2+6*x-4          ….придатній для трансформації
In [98]: y1=lambdify(x,expr1, ‘numpy’)        # Трансформація функції, що
In [99]: y2=lambdify(x,expr2, 'numpy')        ….апроксимується, і її наближень
In [100]: y3=lambdify(x,expr3, 'numpy')       ….з ППП sympy
In [101]: y4=lambdify(x,expr4, 'numpy')       ….у ППП numpy
In [102]: x=np.linspace(-1,1,101)             # Внесення масиву значень x
In [103]: f=y1(x)                             # Формування функції f і її
In [104]: f1=y2(x)                            ….наближень f1,f2,f3 у ППП numpy
In [105]: f2=y3(x)                            ….як функцій незалежної
In [106]: f3=y4(x)                            ….змінної x
In [107]: fig=plt.figure(facecolor='white')   # Формування поля рисунка
In [108]: plt.plot(x,f,'-k',x,f1,'-.g',x,f2,':c',x,f3,'--r',\   # Побудова графіків функції та
                linewidth=3)                  …..її наближень
In [109]: plt.legend(fontsize=18)             # Формування команди на надписи
In [110]: ax=fig.gca()                        # Прив’язка надписів до поля рисунка
In [111]: plt.title(r'Апроксимація  функції \
         поліномами  Лежандра')               # Заголовок
In [112]: plt.text(0.45,.95,r'Графіки наближень до\   # Текст на полі рисунка
          функції', horizontalalignment='center',\
          verticalalignment='center',
          transform=ax.transAxes,fontsize=16)
In [113]: plt.xlabel(u't-вісь абсцис',{'fontname':'Times\
 New Roman'})                                 # Текст на осі абсцис
In [114]: plt.ylabel(r'$f(t)$-ордината')      # Текст на осі ординат
In [115]: plt.grid(True)                      # Нанесення координатної сітки
```

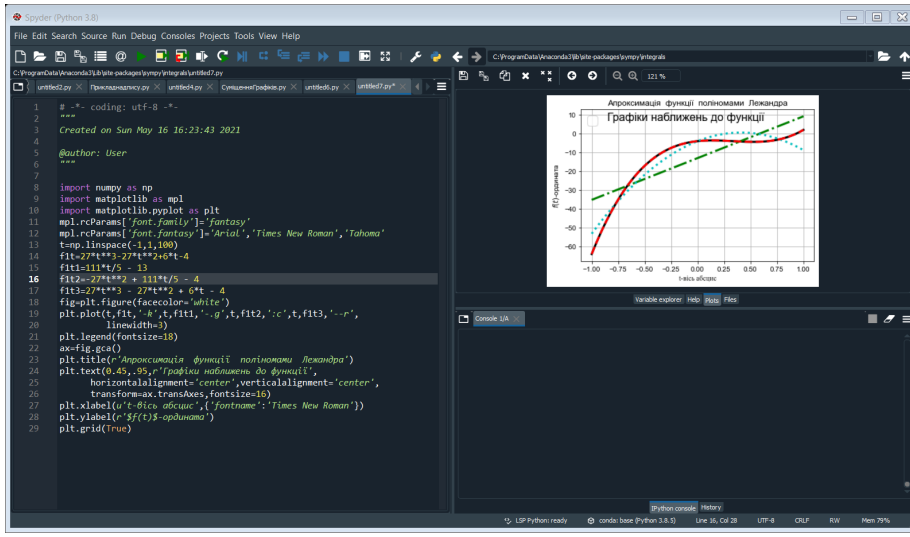

Рисунок 8 – Графік функції $f1t(t) = 27t^3 - 27t^2 + 6t - 4$ та трьох апроксимаційних наближень до неї поліномами Лежандра

Примітка 1. Якщо програму набирати за допомогою командних рядків, то, починаючи з команди In [109], кожна наступна команда, реалізуючись, буде на рисунку знищувати результат виконання попередньої команди, а тому, щоб отримати повний графік з усіма кривими та усіма надписами, потрібно цю програму набирати на лівому полі екрана у вигляді цілісного файлу, після запуску якого отримаємо рисунок у вигляді, який наведено вище.

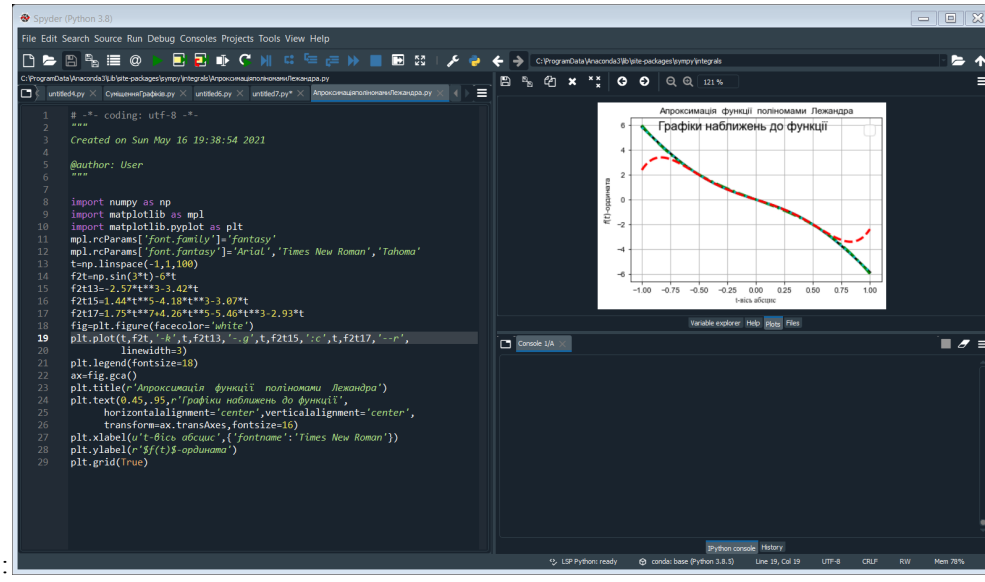

Рисунок 9 – Графік функції $f2t(t) = \sin(3t) - 6t$ та трьох наближень до неї

Примітка 2. Для побудови графіка, наведеного на рис. 9 потрібно в наведеній вище програмі в командах 94–106 замінити функцію $f1t$ та її наближення на функцію $f2t$ та її наближення.

Кінець програми 9.

Розділ 3. ІНТЕГРАЛИ РІМАНА, СТІЛТЬЄСА ТА ЛЕБЕГА (В ПРИКЛАДАХ І ПРОГРАМАХ)

3.1 Інтеграли Рімана, Стілтьєса та Лебега

Після створення Жорданом та Лебегом теорії міри в одновимірному (на відрізку), двовимірному (на площині) та тривимірному (в об'ємі) просторах було поставлено на строгу за математичними мірками основу поняття інтеграла, а також подвійного та потрійного інтегралів. З посиланням на наш навчальний посібник [1], нагадаємо, що , якщо на числовій осі на відрізку $[a,b]$ значень аргументу x задана неперервна і обмежена функція $f(x)$, то, розбивши цей відрізок точками $x_0, x_1, x_2, \dots, x_n$ так, щоб $x_0 = a, x_n = b$, на n сегментів $\Delta_i x$ таких, що

$$\Delta_i x = x_i - x_{i-1}, \quad i = 1,2, \dots, n, \tag{3.1}$$

позначивши ξ_i внутрішню точку сегмента $\Delta_i x$ та визначивши у цій точці значення функції $f(x)$ у вигляді $f(\xi_i)$, отримаємо можливість створити вираз

$$J_1 = \lim_{\Delta_i x \to 0} \sum_{i=1}^{n\to\infty} f(\xi_i)\, \Delta_i x, \tag{3.2}$$

границею якого є число J_1, *цей вираз* називають інтегралом Рімана і позначають

$$J_1 = \int_a^b f(x)dx. \tag{3.3}$$

Узагальнюючи сказане вище на неперервну і обмежену функцію $f(x,y)$ двох змінних x, y, задану на прямокутнику їх значень $x \in [a,b]$, $y \in [c,d]$ на координатній площині (x,y), отримаємо можливість створити вираз

$$J_2 = \lim_{\Delta_i x, \Delta_k y \to 0} \sum_{i=1}^{n\to\infty} \sum_{k=1}^{m\to\infty} f(\xi_i, \eta_k)\, \Delta_k y \Delta_i x, \tag{3.4}$$

границею якого є число J_2, який також є інтегралом Рімана, але двократним, який позначають

$$J_2 = \int_a^b \int_c^d f(x,y)dydx. \tag{3.5}$$

А узагальнюючи сказане вище на неперервну і обмежену функцію $f(x,y,z)$ трьох змінних x, y, z, задану в кубічному об'ємі їх значень $x \in [a,b]$, $y \in [c,d]$, $z \in [e,g]$ у координатному кубі (x,y,z), отримаємо можливість створити вираз

$$J_3 = \lim_{\Delta_i x, \Delta_k y, \Delta_l z \to 0} \sum_{i=1}^{n\to\infty} \sum_{k=1}^{m\to\infty} \sum_{l=1}^{q\to\infty} f(\xi_i, \eta_k, \theta_l)\, \Delta_l z \Delta_k y \Delta_i x, \tag{3.6}$$

границею якого є число J_3, який також є інтегралом Рімана, але трикратним, який позначають

$$J_3 = \int_a^b \int_c^d \int_e^g f(x,y,z)dzdydx. \tag{3.7}$$

А тепер від інтегралів Рімана перейдемо до інтеграла Стілтьєса. Як показано в нашому навчальному посібнику [1], основною відмінністю інтеграла Стілтьєса від інтеграла Рімана є те, що в інтегралі Рімана інтегрування на відрізку $[a,b]$ функції $f(x)$ здійснюється з використанням приростів dx її аргументу на цьому ж відрізку числової осі, а в інтегралі Стілтьєса інтегрування на відрізку $[a,b]$ функції $f(x)$ здійснюється з використанням приростів $dg(x)$ іншої обмеженої функції $g(x)$, заданої на цьому ж відрізку числової осі , а сама

функція $f(x)$, що інтегрується, називається інтегрованою за функцією $g(x)$ на відрізку $[a,b]$ числової осі. Символічно інтеграл Стілтьєса записується так:

$$S = \int_a^b f(x)dg(x), \tag{3.8}$$

а реалізує він, фактично, операцію визначення граничного числа *S* за виразом

$$S = \lim_{\Delta_i x \to 0} \sum_{i=1}^{n\to\infty} f(\xi_i)\left(g(x_i) - g(x_{i-1})\right), \tag{3.9}$$

в якому

$$\Delta_i x = x_i - x_{i-1}, \quad i = 1,2,\dots,n, \qquad \xi_i \in \Delta_i x. \tag{3.10}$$

Цілком очевидно, що наближене значення цього граничного числа можна визначити з виразу

$$S \approx \sum_{i=1}^{n} f(\xi_i)\left(g(x_i) - g(x_{i-1})\right), \tag{3.11}$$

в якому *n* – число, задане в межах розумного вибору, а $\xi_i = x_{i-1}$.

Оскільки алгоритм побудови виразу (3.9) з використанням сум Дарбу-Стілтьєса, запропонований Стілтьєсом, відрізняється від алгоритму побудови інтеграла Рімана з використанням сум Дарбу лише в деталях і викладений в нашому навчальному посібнику [1], то ми на ньому зупинятись не будемо, а відзначимо, що в разі, якщо функція $g(x)$ на відрізку $[a,b]$ є не лише обмеженою, а й неперервною, то, як відомо з курсу математичного аналізу, її диференціал $dg(x)$ можна записати у вигляді

$$dg(x) = g'(x)dx, \tag{3.12}$$

підставляючи який у вираз (3.8), отримаємо інтеграл Стілтьєса

$$S = \int_a^b f(x)g'(x)dx \tag{3.13}$$

у вигляді інтеграла Рімана, але від функції $f(x)g'(x)$. І з цього можна зробити висновок, що інтеграл Стілтьєса як самостійна математична конструкція може бути цікавим лише у випадку, коли або функція *g(x),* або і функція *f(x)* визначені на нульвимірній множині точок відрізка $[a,b]$, тобто є решітчастими функціями, визначеними лише при дискретних значеннях їх аргументів.

Ну і на завершення теоретичної частини цього підрозділу розглянемо інтеграл Лебега, розпочавши цей розгляд наступною цитатою з нашого навчального посібника [1]: «Коли математики побачили, що існують функції, які не інтегруються за Ріманом, вони почали пошук такого узагальнення поняття визначеного інтеграла, за допомогою якого ці функції теж можна було б інтегрувати. І таке узагальнення вдалося здійснити Лебегу, який запропонував приріст координати, за якою здійснюється інтегрування функції $y = f(x)$, заданої на відрізку $[a,b]$, визначати не по осі аргументу *x* , а по функціональній осі *y*, адже у цьому випадку, навіть коли координата *x* задається на нульвимірній множині *E* скінченної чи зліченної кількості точок на осі *x*, координата *y* буде елементом множини дійсних чисел на відрізку [m,M] осі *y*, мірою якого є його довжина, і лівою границею якого є дійсне число *m*, що є мінімальним значенням цієї функції на відрізку $[a,b]$, а правою границею є дійсне число *M*, що є максимальним значенням цієї функції на цьому ж відрізку $[a,b]$». А конструював свій інтеграл Лебег, висунувши умову, щоб вимірна і обмежена нижнім *m* та верхнім *M*

значеннями функція $y = f(x)$, що задана на відрізку $[a,b]$ осі x, була визначеною на множині E з мірою

$$mE(m \le y < M). \tag{3.14}$$

А результатом цієї конструкції Лебегом був отриманий вираз

$$L = \lim_{\Delta_i y \to 0} \sum_{i=1}^{n} y_{i-1} mE_i (y_{i-1} \le y < y_i), \tag{3.15}$$

який мав усі властивості інтеграла при інтегруванні по координаті y, а тому Лебег увів його в математику як нове трактування визначеного інтеграла, який інші математики назвали в його честь інтегралом Лебега, а суму (3.15) назвали інтегральною сумою Лебега на множині E з мірою, визначеною виразом (3.14). Але оскільки міра mE є монотонною функцією координати y, то, позначивши її як $g(y)$, вираз (3.15) можна переписати у вигляді

$$L = \lim_{\Delta_i y \to 0} \sum_{i=1}^{n} y_{i-1} (g(y_i) - g(y_{i-1})), \tag{3.16}$$

який уже має вигляд інтеграла Стілтьєса, а тому може бути записаним і так

$$L = \int_m^M y dg(y). \tag{3.17}$$

А тому і обчисленим наближено інтеграл Лебега може бути за виразом, аналогічним (3.11), тобто за виразом

$$L \approx \sum_{i=1}^{n} y_{i-1} \left(g(y_i) - g(y_{i-1})\right). \tag{3.18}$$

Звертаємо увагу на те, що, якщо функція міри за координатою y на відрізку [m, M] в проекції на відрізок $[a,b]$ осі x є не лише монотонною, а й лінійною, тобто, якщо

$$g(y) = x, \tag{3.19}$$

то вираз (3.17) перетворюється на вираз

$$L = \int_a^b y dx. \tag{3.20}$$

Тобто, у цьому випадку інтегрування за Лебегом на відрізку $[m,M]$осі y і за Ріманом на відрізку $[a,b]$ осі x дають один і той же результат.

І оскільки інтеграли Стілтьєса та Лебега, як і звичні для інженерів та програмістів інтеграли Рімана, теж можна використовувати в практичних задачах системного аналізу з дискретними моделями, про які йшла мова в наступних підрозділах нашого посібника [1], то продемонструємо, як наближено обчислювати інтеграли Стілтьєса та Лебега, але перед цим нагадаємо, що

$$m = \min_{x \in [a,b]} y(x), \tag{3.21}$$

$$M = \max_{x \in [a,b]} y(x). \tag{3.22}$$

А міра функції y(x) на осі y є монотонно зростаючою функцією, граничним значенням якої є довжина відрізка [m,M], тобто

$$mE(m \le y < M) = M\text{-}m. \tag{3.23}$$

3.2 Додаткові відомості з мови програмування Python, достатні для розв'язання задач, пов'язаних з інтегруванням функцій

І викладення цих додаткових знань почнемо з того, як за допомогою мови Python обчислювати визначені інтеграли. Для цього нам, перш за все, знадобиться викликати ППП ***scipy***, бо саме в цьому пакеті у модулі ***scipy.integrate*** є функція ***quad(, ,)***, яка реалізує процес чисельного інтегрування заданої в ППП ***numpy*** функції, в аргументних дужках якої першою координатою задається функція, що інтегрується, другою координатою задається нижня границя інтегрування, а третьою координатою задається верхня границя інтегрування. ***В пояснювальному прикладі № 27*** продемонстровано дію цієї функції ***quad(, ,)*** у випадку, коли границі інтегрування є дійсними числами, наприклад, границями діапазону ***[0,1]*****,** коли тіло функції задається безпосередньо, коли воно задається параметрично, а також у випадку використання лямбда-функції з заданими числовими значеннями параметрів.

Пояснювальний приклад № 27

```
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, 3.773226e-15)            ….в заданих границях [0,1]
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 a3,a2,a1,a0
In [7]: q2=quad(f,0,1,args=(4,3,2,1));q2              # Обчислення інтеграла від функції f(x)
Out[7]: (4.0, 4.440892098500626e-14)                  ….в границях [0,1] з коефіцієнтами 4,3,2,1
In [8]: q3=quad(lambda x: f(x,4,3,2,1),0,1);q3        # Обчислення інтеграла від функції f(x)
Out[8]: (4.0, 4.440892098500626e-14)                  ….з використанням лямбда-функції
```

Кінець пояснювального прикладу № 27.

А далі звертаємо увагу на те, що в разі, якщо нам потрібно обчислити подвійний інтеграл, то знову ж таки нам, перш за все, потрібно буде викликати ППП ***scipy***, бо саме в цьому пакеті у модулі ***scipy.integrate*** є функція ***dblquad(, , , ,)***, яка реалізує процес чисельного інтегрування заданої в ППП ***numpy*** функції ***f(x,y)*** незалежних змінних ***x,y***, в аргументних дужках якої першою координатою задається функція, що інтегрується, другою координатою задається нижня границя інтегрування зовнішнього інтеграла, третьою координатою задається верхня границя інтегрування зовнішнього інтеграла, четвертою координатою задається нижня границя інтегрування внутрішнього інтеграла, а п'ятою координатою задається верхня границя 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 границі внутрішнього інтеграла, навіть якщо вони константи, мають записуватись як функції, оскільки вони дійсно можуть бути заданими у вигляді функцій зовнішньої змінної. ***В пояснювальному прикладі № 28*** продемонстровано дію цієї функції ***dblquad(, , , ,)*** у випадку, коли границі інтегрування є дійсними числами, наприклад, границями діапазону ***[0,1]*** або піддіапазону ***[0,np.inf]***, де через символ ***np.inf*** в ППП ***numpy*** позначається ∞ (нескінченність).

Пояснювальний приклад № 28

```
In [1]: import scipy                                  # Виклик ППП scipy
In [2]: from scipy.integrate import dblquad           # Виклик функції dblquad із
                                                      ….scipy.integrate
```

```
In [3]: import numpy as np                          # Виклик ППП питру під символом пр
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 границя
In [6]: h=lambda x: 1                               # Верхня внутрішня границя
In [7]: J2=dblquad(f,0,1,g,h);J2                    # Обчислення подвійного інтеграла
Out[7]: (0.9888977057628652, 1.528779994947608e-14)
In [8]: g1=lambda x: 0                              # Нижня внутрішня границя
In [9]: h1=lambda x: x                              # Верхня внутрішня границя
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 границя
In [12]: h2=lambda x: x**2                          # Верхня внутрішня границя
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 границя
In [15]: h3=lambda x: np.inf                        # Верхня внутрішня границя
In [16]: J23=dblquad(f,0,1,g3,h3);J23               # Обчислення подвійного інтеграла
Out[16]: (1.1436776435894211, 1.588466384676408e-08)
```

Кінець пояснювального прикладу № 28.

А якщо нам потрібно обчислити потрійний інтеграл, то знову ж таки нам, перш за все, потрібно буде викликати ППП ***scipy***, бо саме в цьому пакеті у модулі ***scipy.integrate*** є функція ***tplquad(, , , , , ,)***, яка реалізує процес чисельного інтегрування заданої в ППП ***питру*** функції ***f(x,y,z)*** незалежних змінних ***x,y,z***, в аргументних дужках якої першою координатою задається функція, що інтегрується, другою координатою задається нижня границя інтегрування зовнішнього інтеграла, третьою координатою задається верхня границя інтегрування зовнішнього інтеграла, четвертою координатою задається нижня границя інтегрування другого внутрішнього інтеграла, п'ятою координатою задається верхня границя інтегрування другого внутрішнього інтеграла, шостою координатою задається нижня границя інтегрування першого внутрішнього інтеграла, а сьомою координатою задається верхня границя інтегрування першого внутрішнього інтеграла, і при цьому в записі функції, що інтегрується, першим має стояти символ внутрішньої змінної, за якою здійснюється перше внутрішнє інтегрування, другим - символ змінної, за якою здійснюється друге внутрішнє інтегрування, а третім – символ, за яким здійснюється зовнішнє інтегрування. Нижня і верхня границі 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, і нижня та верхня границі 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 пояснювальному прикладі № 29*** продемонстровано дію цієї функції ***tplquad(, , , , , ,)*** у випадку, коли границі інтегрування є дійсними числами, наприклад, границями діапазону ***[0,1]*** або піддіапазону ***[0,np.inf]***, де через символ ***np.inf*** в ППП ***питру*** позначається ∞ (нескінченність).

Пояснювальний приклад № 29

```
In [1]: import scipy                                # Виклик ППП scipy
In [2]: from scipy.integrate import tplquad         # Виклик функції tplquad із
                                                    ….scipy.integrate
In [3]: import numpy as np                          # Виклик ППП питру під символом пр
```

```
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 границя
In [6]: h=lambda x: 1                                  # Верхня друга внутрішня границя
In [7]: q=lambda x,y: 0                                # Нижня перша внутрішня границя
In [8]: r=lambda x,y: 1                                # Верхня перша внутрішня границя
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 границя
In [11]: h1=lambda x: x                                # Верхня друга внутрішня границя
In [12]: q1=lambda x,y: 0                              # Нижня перша внутрішня границя
In [13]: r1=lambda x,y: x+y                            # Верхня перша внутрішня границя
In [14]: J31=tplquad(f,0,1,g1,h1,q1,r1)                # Обчислення потрійного інтеграла
In [15]: J31                                           # Візуалізація потрійного інтеграла
Out[15]: (0.5826673475239622, 4.011287899202798e-12)
In [16]: g2=lambda x: 0                                # Нижня друга внутрішня границя
In [17]: h2=lambda x: 1/x                              # Верхня друга внутрішня границя
In [18]: q2=lambda x,y: 0                              # Нижня перша внутрішня границя
In [19]: r2=lambda x,y: 1/(x+y)                        # Верхня перша внутрішня границя
In [20]: J32=tplquad(f,0,np.inf,g2,h2,q2,r2)           # Обчислення потрійного інтеграла
In [21]: J32                                           # Візуалізація потрійного інтеграла
Out[21]: (1.8839075869187127, 1244.097475114733e-12)
```

Кінець пояснювального прикладу № 29.

Як витікає з пояснювальних прикладів № 27, № 28 та № 29, для того, щоб проінтегрувати математичну функцію однієї, двох або трьох незалежних змінних, потрібно лише задати цю математичну функцію та її границі інтегрування і застосувати до неї програмну функцію однократного інтегрування, якщо інтегрується математична функція однієї незалежної змінної, програмну функцію подвійного інтегрування, якщо інтегрується математична функція двох незалежних змінних, або програмну функцію потрійного інтегрування, якщо інтегрується математична функція трьох незалежних змінних. ***Але для наближеного обчислення інтегралів Стілтьєса та Лебега***, для яких в програмному середовищі Anaconda немає програмних функцій інтегрування, нам ***потрібно додати*** до тих знань мови Python, які ми уже засвоїли раніше, ***інформацію про*** ще три важливі програмні функції, а саме: ***інформацію про*** програмну функцію ***numpy.sort( )*** із ППП ***numpy***, яка ***трансформує*** (відсортовує) ***масив***, поміщений в аргументні дужки, таким чином, ***щоб елементи*** цього масиву ***розмістились в порядку зростання*** їх числового значення, ***інформацію про*** програмну ***функцію*** теж із ППП ***numpy,*** яка позначається як ***numpy.diff(, ,)*** і ***визначає кінцеві різниці n-го порядку*** для масиву значень заданої ***математичної функції***, причому в аргументних дужках цієї програмної функції першим задається масив f_i значень математичної функції, другим – порядок ***n*** різниці значень цієї математичної функції, а третім аргументом задається вісь *axis*, вздовж якої визначаються різниці заданої математичної функції. У випадку, коли потрібно знайти лише перші різниці заданої математичної функції, в аргументних дужках програмної функції вказується лише масив значень заданої математичної функції. А щодо ***інформації про*** третю програмну ***функцію*** і теж із ППП ***numpy***, яка нам знадобиться при складанні програми обчислення інтегралів Стілтьєса та Лебега, то вона позначається як ***f1[:-1]*** і ***означає видалення*** зі списку ***f1*** останнього елемента. ***В пояснювальному прикладі № 30***, побудованому з використанням інформації наведеної в роботі [4], проілюстровано дію вищевказаних програмних функцій.

Пояснювальний приклад № 30

```
In [1]: import numpy as np                          # Виклик ППП питру під символом np
In [2]: l1=np.array([1,3,5,7,9])                    # Формування одновимірного масиву l1
In [3]: l11=np.diff(l1); l11                        # Визначення масиву l11 перших різниць
Out[3]: array([2, 2, 2, 2])                         ….масиву l1 та його візуалізація
In [4]: l2=np.array([[1,3,6,7], [5,6,3,1]]); l2     # Формування двовимірного масиву l2
Out[4]:                                             ….та його візуалізація
array([[1, 3, 6, 7],
       [5, 6, 3, 1]])
In [5]: l22=np.diff(l2); l22                        # Визначення масиву l22 перших різниць
Out[5]:                                             ….масиву l2 по рядках та його візуалізація
array([[ 2,  3,  1],
       [ 1, -3, -2]])
In [6]: l23=np.diff(l2, n=2); l23                   # Визначення масиву l23 других різниць
Out[6]:                                             ….масиву l2 по рядках та його візуалізація
array([[ 1, -2],
       [-4,  1]])
In [7]: l24=np.diff(l2, n=2, axis=0); l24           # Визначення масиву l24 других різниць
Out[7]:                                             ….масиву l2 по стовпцях (тобто, вздовж
array([],                                           ….осі Axis=0) та його візуалізація як
shape=(0, 4), dtype=int32)                          ….масиву з нулем елементів в 4 рядках
In [8]: l25=np.diff(l2, n=2, axis=-1); l25          # Підтвердження, що визначення масиву l25
Out[8]:                                             ….других різниць масиву l2 по рядках, який
array([[ 1, -2],                                    ….дорівнює масиву l23, – це визначення їх
       [-4,  1]])                                   ….вздовж осі Axis=-1
In [9]: l26=np.diff(l2, n=1,axis=0); l26            # Визначення масиву l26 перших різниць
Out[9]:                                             ….масиву l2 по стовпцях (тобто, вздовж
array([[ 4,  3, -3, -6]])                           ….осі Axis=0) та його візуалізація
In [10]: x=np.linspace(0,2,11)                      # Формування  масиву значень x
In [11]: def f(x):                                  # Формування власної функції  f(x)
             return x**2                            ….на відрізку [0,2] значень аргументу x

In [12]: fvec=np.vectorize(f)                       # Векторизація власної функції  f(x)
In [13]: f1=fvec(x);f1                              # Візуалізація векторизованої функції f(x)
Out[13]: array([0.  , 0.04, 0.16, 0.36, 0.64,       ….у вигляді списку f1
         1.  , 1.44, 1.96, 2.56, 3.24, 4.  ])
In [14]: f11=np.diff(f1);f11                        # Визначення масиву f11 перших різниць
Out[14]: array([0.04, 0.12, 0.2 , 0.28, 0.36,       ….векторизованої функції f(x)
          0.44, 0.52, 0.6 , 0.68, 0.76])            ….та його візуалізація
In [15]: f2=lambda x: x**2+2*x                      # Формування f(x) через лямбда-функцію
In [16]: f2vec=np.vectorize(f2)                     # Векторизація функції  f(x)
In [17]: f3=f2vec(x);f3                             # Візуалізація векторизованої функції f(x)
Out[17]: array([0.  , 0.44, 0.96, 1.56, 2.24,       ….у вигляді списку f3
          3.  , 3.84, 4.76, 5.76, 6.84, 8.  ])
In [18]: f33=np.diff(f3);f33                        # Визначення масиву f33 перших різниць
Out[18]: array([0.44, 0.52, 0.6 , 0.68, 0.76,       ….векторизованої функції f(x)
          0.84, 0.92, 1.  , 1.08, 1.16])            ….та його візуалізація
In [19]: L1=[-4,2,8,4,0,-1,11,7]                    # Формування масиву у вигляді списку L1
In [20]: L2=np.sort(L1);L2                          # Трансформація масиву L1 в масив L2,
Out[20]: array([-4, -1,  0,  2,  4,  7,  8, 11])    ….відсортований за зростанням значень
```

```
In [21]: L22=L2[:-1]; L22                    # Видалення з масиву L2 останнього
Out[21]: array([-4, -1,  0,  2,  4,  7,  8])  ….елемента і приведення до вигляду L22
```
Кінець пояснювального прикладу №30.

3.3 Задачі з визначення інтегралів Рімана, Стілтьєса та Лебега в програмах мовою Python

Програми мовою Python, за якими можна обчислювати однократні, подвійні та потрійні інтеграли Рімана уже наведені нами в пояснювальних прикладах № 27,№ 28 та № 29, тому на них ми вже у цьому підрозділі акцентувати увагу не будемо, а складемо лише програми для обчислення інтегралів Стілтьєса та Лебега для функцій, визначених на нульвимірних множинах, тобто для решітчастих функцій, які не інтегруються за Ріманом.

Програму мовою Python ***для*** наближеного ***обчислення інтеграла Стілтьєса (3.8***) ми ***будемо складати, використовуючи формулу (3.11***) та враховуючи умову (3.10), а ***програму для*** наближеного ***обчислення інтеграла Лебега (3.17***) ми ***будемо складати, використовуючи формулу (3.18)*** та враховуючи умови (3.10, (3.21)–(3.23).

Програма мовою Python для обчислення інтеграла Стілтьєса від функції $f(x)=x^3+3x^2-2x-4$ за функцією $g(x)=2exp(x)-3x$ дійсної змінної x, заданої на відрізку $[0,2]$ дискретно через проміжок $\Delta_i x=x_i-x_{i-1}=0.1;\quad i=1,2,\dots,20$

(Програма 10)

```
In [1]: import numpy as np                   # Виклик ППП питру під символом пр
In [2]: x=np.linspace(0,2,21)                # Формування масиву значень x
In [3]: def f(x):                            # Формування функції f(x), що
            return x**3+3*x**2-2*x-4         ….інтегрується за Стілтьєсом
                                             ….на відрізку [0, 2] значень аргументу x
In [4]: fvec=np.vectorize(f)                 # Векторизація функції f(x)
In [5]: f1=fvec(x);f1                        # Візуалізація векторизованої функції f(x)
Out[5]:                                      ….у вигляді списку f1
array([-4.   , -4.169, -4.272, -4.303, -4.256,
       -4.125, -3.904, -3.587, -3.168, -2.641,
       -2.   , -1.239, -0.352,  0.667,  1.824,
        3.125 , 4.576,  6.183,  7.952,  9.889, 12.  ])
In [6]: g=lambda x: 2*np.exp(x)-3*x          # Формування функції g(x), за якою
In [7]: gvec=np.vectorize(g)                 ….інтегрується f(x) та її векторизація
In [8]: g1=gvec(x);g1                        # Візуалізація векторизованої функції g(x)
Out[8]:                                      ….у вигляді списку g1
array([2.        , 1.91034184, 1.84280552,
       1.79971762, 1.7836494 , 1.79744254,
       1.8442376 , 1.92750541, 2.05108186,
       2.21920622, 2.43656366, 2.70833205,
       3.04023385, 3.43859334, 3.91039993,
       4.46337814, 5.10606485, 5.84789478,
       6.69929493, 7.67178888, 8.7781122 ])
In [9]: g11=np.diff(g1);g11                  # Визначення масиву g11 перших різниць
Out[9]:                                      ….векторизованої функції g(x)
```

```
        array([-0.08965816, -0.06753632,          ….та його візуалізація
        -0.0430879, -0.01606822,  0.01379315,
         0.04679506, 0.08326781,  0.12357644,
         0.16812437,  0.21735743, 0.27176839,
         0.3319018 ,  0.39835949,  0.4718066 ,
         0.55297821, 0.64268671,  0.74182993,
         0.85140015,  0.97249396,  1.10632331])
In [10]: f11=f1[:-1];f11                  # Видалення з масиву f1 останнього
Out[10]:                                  ….елемента і приведення його до вигляду
array([-4.  , -4.169, -4.272, -4.303, -4.256, -4.125, ….f11 для узгодження з масивом g11
        -3.904, -3.587, -3.168, -2.641, -2.  , -1.239,
        -0.352,  0.667,  1.824,  3.125, 4.576,  6.183,
         7.952,  9.889])
In [11]: s1=f11*g11;s1                    # Поелементне перемноження
Out[11]:                                  ….масивів f11 та g11 та
array([ 0.35863266,  0.28155892,  0.18407151,  ….візуалізація результату
        0.06914155, -0.05870363, -0.19302962,
        -0.32507755, -0.4432687 , -0.53261799,
        -0.57404098, -0.54353678, -0.41122633,
        -0.14022254,  0.314695  ,  1.00863225,
         2.00839596,  3.39461378,  5.2642071 ,
         7.73327194, 10.94043125])
In [12]: S=s1.sum( );S                    # Обчислення інтеграла Стілтьєса
Out[12]: 28.33592779370339                ….від дискретної функції f(x) за функцією)
                                          …. g(x на відрізку [0,2] значень аргументу x
```

Кінець програми 10

Програма мовою Python для обчислення інтеграла Лебега від функції $f(x) = e^{-x}sin2x$ дійсної змінної x, заданої на відрізку $[0,3]$ дискретно в точках через проміжок $\Delta_i x = x_i - x_{i-1} = 0.15; \quad i = 1,2,\dots,20$

(Програма 11)

```
In [1]: import numpy as np                # Виклик ППП питру під символом пр
In [2]: x=np.linspace(0,3,21)             # Формування масиву значень x
In [3]: def f(x):                         # Формування функції f(x), що
            return np.exp(-x)*np.sin(3*x)  ….інтегрується за Лебегом
                                           ….на відрізку [0,3] значень аргументу x
In [4]: fvec=np.vectorize(f)              # Векторизація функції f(x)
In [5]: f1=fvec(x);f1                     # Візуалізація векторизованої функції f(x)
Out[5]:                                   ….у вигляді списку f1
array([ 0.        ,  0.3743783 ,  0.58030285,
        0.62214868,  0.53445891, 0.36753575,
        0.17375969, -0.00294201, -0.1332846 ,
        -0.20441749, -0.21811645, -0.1866539 ,
        -0.12773711, -0.05972154,  0.00205897,
        0.0474343 ,  0.07199992,  0.07646286,
        0.06518194,  0.0443898 , 0.02051817])
In [6]: M=max(f1);M                       # Визначення максимуму M функції f(x)
Out[6]: 0.6221486811452028                ….на відрізку [0,3] значень аргументу x
In [7]: m=min(f1);m                       # Визначення мінімуму m функції f(x)
Out[7]: -0.21811645170452654              ….на відрізку [0,3] значень аргументу x
```

```
In [8]: mEf=M-m;mEf                          # Визначення міри M-m функції f(x)
Out[8]: 0.8402651328497294                   ….на відрізку [0,3] значень аргументу x
In [9]: f11=np.sort(f1);f11                  # Трансформація масиву f1 в масив f11,
Out[9]:                                      ….відсортований за зростанням значень
array([-0.21811645, -0.20441749, -0.1866539 ,
       -0.1332846 , -0.12773711, -0.05972154,
       -0.00294201,  0.        ,  0.00205897,
        0.02051817, 0.0443898 ,  0.0474343 ,
       0.06518194,  0.07199992,  0.07646286,
       0.17375969,  0.36753575,  0.3743783 ,
       0.53445891,  0.58030285, 0.62214868])
In [10]: g=np.diff(f11);g                    # Визначення масиву g перших різниць
Out[10]:                                     ….векторизованої та відсортованої функції f(x)
array([0.01369896, 0.0177636 , 0.0533693 ,
       0.00554749, 0.06801557, 0.05677952,
       0.00294201, 0.00205897, 0.0184592 ,
       0.02387163, 0.0030445 , 0.01774765,
       0.00681798, 0.00446294, 0.09729683,
       0.19377606, 0.00684255, 0.16008061,
       0.04584394, 0.04184583])
In [11]: f111=f11[:-1];f111                  # Видалення з масиву f11 останнього
Out[11]:                                     ….елемента і приведення його до вигляду
array([-0.21811645, -0.20441749, -0.1866539 ,  …. f111 для узгодження з масивом g
       -0.1332846 , -0.12773711,-0.05972154,
       -0.00294201,  0.        ,  0.00205897,
        0.02051817, 0.0443898 ,  0.0474343 ,
        0.06518194,  0.07199992,  0.07646286,
        0.17375969,  0.36753575,  0.3743783 ,
        0.53445891,  0.58030285])
In [12]: l1=f111*g;l1                        # Поелементне перемноження
Out[12]:                                     ….масивів f111 та g та
array([-2.98796838e-03, -3.63118992e-03,     ….візуалізація результату
       -9.96158766e-03, -7.39394368e-04,
       -8.68811299e-03, -3.39096030e-03,
       -8.65544179e-06,  0.00000000e+00,
       3.80069447e-05,  4.89802141e-04,
        1.35144637e-04,  8.41847084e-04,
        4.44409318e-04,  3.21331035e-04,
        7.43959408e-03,  3.36704688e-02,
        2.51488210e-03,  5.99307062e-02,
        2.45017000e-02,  2.42832565e-02])
In [13]: L=np.sum(l1);L                      # Обчислення інтеграла Лебега від
Out[13]: 0.1252032798312037                  ….дискретної функції f(x) на відрізку [0,3]
                                             ….значень аргументу x
```

Кінець програми 11

Завершимо цей розділ ми зауваженням, що наведені вище програми 10 та 11, за допомогою яких обчислюються інтеграли Стілтьєса та Лебега, вперше були опублікованими у нашій роботі [6].

Розділ 4 ФУНКЦІОНАЛИ ТА МЕТОДИ ПОШУКУ ЇХ БЕЗУМОВНИХ ЕКСТРЕМУМІВ (В ПРИКЛАДАХ І ПРОГРАМАХ)

4.1 Функції і функціонали, що використовуються в прикладних задачах системного аналізу, та методи пошуку їх безумовних екстремумів

У кінці однойменного підрозділу у базовому навчальному посібнику [1] серед інших стосовно функцій та функціоналів сформульовані і такі питання:

1. Розкрийте поняття «функція» та способи її задавання.
2. Дайте означення неперервної, гладкої та кусково-гладкої функцій.
3. Розкрийте поняття «функціонал» і наведіть приклади його задавання.
4. Як знайти координату точки екстремуму для функції однієї незалежної змінної?
5. Як розрізняти, де в точці екстремуму має місце мінімум функції однієї незалежної змінної, а де – максимум?
6. Які умови існування екстремуму функції однієї незалежної змінної є необхідними, а які – достатніми?
7. Як визначити відсутність екстремуму в точці, для якої необхідні умови існування екстремуму функції однієї незалежної змінної виконуються?
8. Що собою являють локальний та глобальний екстремуми функції?
9. Як знайти координати точки екстремуму для функції двох незалежних змінних?
10. Як розпізнати, де в точці екстремуму має місце мінімум функції двох незалежних змінних, а де – максимум?
11. Які умови існування екстремуму функції двох незалежних змінних є необхідними, а які – достатніми?
12. Як визначити відсутність екстремуму в точці, для якої необхідні умови існування екстремуму функції двох незалежних змінних виконуються?
13. Як змістовно відрізнити абсолютний та відносний екстремуми функціонала?
14. Як визначити відстань нульового і першого порядку між двома функціями?
15. Коли на екстремалі досягається сильний відносний екстремум функціонала? Коли досягається слабкий відносний екстремум?
16. Для чого потрібне рівняння Ейлера?
17. Як виглядає рівняння Ейлера у формі нелінійного диференціального рівняння другого порядку?
18. Як виглядає рівняння Ейлера у канонічній формі?
19. Якими є достатні умови існування екстремуму функціонала в найпростішій задачі варіаційного числення? Як розрізнити максимум і мінімум функціонала?
20. Що собою являють умови трансверсальності? Для чого вони потрібні?
21. Що собою являють необхідні умови існування екстремуму функціонала, який пов’язує декілька невідомих функцій та їх перших похідних?
22. Якими є достатні умови існування екстремуму функціонала, що пов’язує декілька невідомих функцій та їх перших похідних?
23. Як виглядає рівняння Ейлера–Пуассона? Дайте його інтерпретацію.
24. Якими є достатні умови існування екстремуму функціонала, що пов’язує екстремаль з її старшими похідними? Як розрізнити максимум та мінімум функціонала в цьому випадку?

Тож із відповідей на ці запитання, дотримуючись означень, які дані в роботі [1], ми і продовжимо викладення змісту нашого «Навчального посібника для опанування студентами способів розв’язання задач з функціонального аналізу мовою Python».

Отже, ***функція*** – це ***закон***, за яким одній ***числовій множині*** ставиться у відповідність інша ***числова множина***. Умовно функцію найчастіше записують так

$$y = f(x), x \in X, y \in Y, \tag{4.1}$$

або так

$$y = y(x), x \in X, y \in Y, \tag{4.2}$$

при цьому множину X називають областю задавання функції, а множину Y – областю значень цієї функції, а $\in$ – символ належності елемента до множини.

Якщо кожному ***числу*** $x \in X$ функція f задає у відповідність лише ***одне число*** $y \in Y$, то така ***функція*** називається ***однозначною***, а якщо кожному ***числу*** $x \in X$ функція f задає у відповідність два або ***більше чисел*** $y \in Y$, то така ***функція*** називається ***багатозначною***. Задавати функцію можна у вигляді таблиці, графіком або однією чи декількома формулами. ***Функція,*** графік якої не має розривів, належить до класу ***неперервних***, а неперервна функція, графік якої не містить зломів, а тому має неперервну першу похідну, належить до класу ***гладких***. Неперервна функція, графік якої має зломи, а тому її похідна – розриви 1-го роду, належить до класу ***кусково-гладких***.

Графічна інтерпретація усіх вищевикладених понять наведена у нашому базовому навчальному посібнику [1], тому у цьому навчальному посібнику ми їх повторювати не будемо.

А далі нагадаємо, що ***функціонал*** – це ***закон***, за яким ***множині функцій*** $Y(X)$ ставиться у відповідність ***множина чисел*** J (графічну інтерпретацію теж дивись в роботі [1]). При цьому множину функцій $Y(X)$ називають областю задання функціонала, а множину чисел J – областю його значень. Умовно функціонал найчастіше записують так

$$J_y = J(y(x), x), \quad x \in X, \quad y(x) \in Y(X), \quad J_y \in J. \tag{4.3}$$

Прикладами функціоналів можуть бути визначені інтеграли:

$$J_y = \int_a^b y(x)dx, \tag{4.4}$$

$$J_y^f = \int_a^b f(x,y)dx, \tag{4.5}$$

$$J_y^F = \int_a^b F(x,y,y')dx, \tag{4.6}$$

в яких $f(x, y)$ – математичний вираз, який є конструкцією з незалежної змінної x та її функції $y(x)$, а $F(x, y, y')$ – математичний вираз, який є конструкцією з незалежної змінної x, її функції $y(x)$ та першої похідної $y'(x)$ від цієї функції; при цьому відрізок $[a, b]$ є областю задавання функції $y(x)$, тобто $x \in [a, b]$.

А далі нагадаємо, що завершальним етапом системного аналізу складного об'єкта є його оптимізація, яка полягає у пошуку таких параметрів цього об'єкта, за яких його вихідна координата характеризуються найкращим чином за показниками, вибраними як міра цієї «найкращості». Саме ці ***показники***, внутрішніми складовими яких є функції, що характеризують ***процеси*** в складних об'єктах, якими характеризується ***«найкращість»*** цих процесів, як правило, є ***функціоналами***, серед яких в загальній методології функціонального

аналізу в його застосунку у формі варіаційного числення найпоширенішими є уже наведені вище функціонали (4.4)-(4.6), а також:
1) функціонал

$$J^F_{y^{(n)}} = \int_a^b f\left(x, y, y', y'', \dots, y^{(n)}\right)dx, \tag{4.7}$$

який пов'язує між собою не лише функцію $y(x)$ та її похідну $y'(x)$, як у випадку (4.6), але і старші похідні $y''(x)$,…, $y^{(n)}(x)$;

2) функціонал

$$J^F_{yz} = \int_a^b F(x, y, z, y', z')dx, \tag{4..8}$$

який пов'язує між собою функції

$$y(x), z(x), \tag{4.9}$$

що задають поверхню у звичному тривимірному просторі $\{x, y, z\}$, та їх похідні

$$y'(x), z'(x) \tag{4.10}$$

у цьому ж просторі;

3) функціонал

$$J^F_{y_1' y_n'} = \int_a^b F(x, y_1, y_2, \dots, y_n, y_1', y_2', \dots, y_n')dx, \tag{4.11}$$

який пов'язує множину функцій

$$y_1, y_2, \dots, y_n, \tag{4.12}$$

що задає поверхню в *n*-вимірному просторі, та множину

$$y_1', y_2', \dots, y_n' \tag{4.13}$$

перших похідних цих функцій у цьому ж просторі.

Дослідженням умов, за яких ці функціонали набувають екстремальних значень, виконується в рамках розділу функціонального аналізу, який носить назву варіаційного числення, ***а функції, на яких ці функціонали набувають екстремальних значень, називаються екстремалями***. Але, перш ніж розпочати дослідження умов, за яких функціонали набувають екстремальних значень, нагадаємо за яких умов екстремуму набувають функції, оскільки це дасть змогу завдяки певним аналогіям між цими умовами, легше зрозуміти матеріал, присвячений умовам набуття екстремуму функціоналами.

А далі розглянемо дві функції $f_1(x)$,$f_2(x)$, графіки яких в області $x \in (-\infty, \infty)$ наведені в роботі [1] на рис. 4.1.

Нагадаємо, що для побудови графіка першої похідної від функції, заданої графічно, потрібно в кожній точці графіка функції визначити тангенс кута нахилу до осі x дотичної до цього графіка, проведеної в даній точці. Аналогічно, з використанням графіка першої похідної, будується графік другої похідної цієї функції.

З цих графіків видно, що, як у випадку мінімуму функції $f_1(x)$ в точці x_{01}, так і у випадку максимуму функції $f_2(x)$ в точці x_{02}, перші похідні $f_1'(x)$, $f_2'(x)$ в цих точках дорівнюють нулю. Тож цілком зрозумілим є той факт, що значення x_{01}, x_{02} потрібно знаходити шляхом розв'язання рівнянь

$$f_1'(x) = 0, \;\; f_2'(x) = 0\,, \tag{4.14}$$

і як розпізнати, де матимемо максимум, а де – мінімум, теж легко бачити з графіків, наведених в роботі [1] на рис. 4.1. Як бачимо, в точці x_{01} маємо

$$f_1''(x_{01}) > 0, \tag{4.15}$$

і у цій точці має місце мінімум функції $f_1(x)$, а в точці x_{02} маємо

$$f_2''(x_{02}) < 0, \tag{4.16}$$

і в точці має місце максимум функції $f_2(x)$.

Тому можна стверджувати, що ***рівняння (4.14) задають необхідні умови існування екстремуму функції однієї змінної, а нерівності (4.15), (4.16) – достатні***. Із рівнянь (4.14) витікає, що

$$f_1(x_{01}) = \min_{x\in[a,b]} f_1(x), \tag{4.17}$$

$$f_2(x_{02}) = \max_{x\in[a,b]} f_2(x). \tag{4.18}$$

Виникає запитання: «А чи завжди при виконанні умов (4.14) має місце екстремум функції однієї змінної?». Виявляється, що не завжди. Це легко бачити з графіків функції $f_3(x)$ та її першої похідної $f_3'(x)$, наведених на рис. 4.2 в роботі [1]. З цих графіків видно, що в точці x_{03}, незважаючи на виконання необхідних умов екстремуму, тобто, незважаючи на те, що

$$f_3'(x)=0, \tag{4.19}$$

маємо і не максимум, і не мінімум функції, а її перегин.

Отже, умовою того, що в точці x_{03} маємо перегин функції, як видно з графіка другої похідної $f_3''(x)$, наведеного теж на рис. 4.2 в роботі [1], є виконання рівності

$$f_3''(x_{03})=0. \tag{4.20}$$

А далі розглянемо графік функції $f_4(x)$, наведений на рис. 4.3 в роботі [1]. І з цього графіка видно, що значення цієї функції в точці x_{01} є меншим за її ж значення в точці x_{03} , а значення цієї ж функції в точці x_{02} є більшим за її значення в точці x_{04}.

В зв'язку з цим ***максимуми та мінімуми функції поділяють на глобальні та локальні***. Зрозуміло, що в точці глобального мінімуму функція має найменше значення в області її задавання, а в точці глобального максимуму функція має найбільше значення в цій області.

Тож, досліджуючи функцію на екстремум, потрібно не забувати дослідити її в усіх кореневих точках, отримуваних з рівняння необхідних умов екстремуму, інакше є небезпека зупинитись на локальному екстремумі і пропустити глобальний.

А тепер розглянемо функцію $f_{12}(x_1, x_2)$ двох незалежних змінних x_1 та x_2, де $x_1 \in [a, b]$, $x_2 \in [c, d]$, графік якої зображено на рис. 4.4. в роботі [1].

Зрозуміло, що такою функцією можна описати поверхню гори, яка піднялася над площиною землі, обмеженою прямокутником зі сторонами «ab» та «cd».

На цьому ж рисунку показані графіки функцій однієї незалежної змінної $f_1(x_1, e)$ та $f_2(g, x_2)$, які є лініями перерізу поверхні гори з площинами, паралельними координатним площинам, за умови, що ці площини проходять через точку з координатами (g, e) на площині землі. Як бачимо, в функціях $f_1(x_1, e)$, $f_2(g, x_2)$ одна зі змінних перетворилась на параметр, який після того, як ми його задали чисельно, перетворюється на коефіцієнт. Після цього функція двох незалежних змінних стає функцією лише однієї незалежної змінної.

Задамося питанням: «Як знайти найвищу точку гори?».

З математичної точки зору це означає, що потрібно знайти максимум функції $f_{12}(x_1, x_2)$. Зрозуміло, що для того, щоб знайти цей максимум, потрібно спочатку знайти його координати на площині x_1, x_2 (на рис. 4.4 в роботі [1] – це точка (g, e)).

Оскільки ми вже знаємо, що для того, щоб знайти координату точки максимуму для функції однієї незалежної змінної, потрібно взяти похідну від цієї функції, прирівняти цю похідну до нуля і знайти корінь (чи корені) отриманого рівняння, то з викладеного вище сам собою напрошується ***спосіб пошуку координат точки максимуму функції $f_{12}(x_1, x_2)$ двох незалежних змінних x_1 та x_2 – потрібно взяти перші частинні похідні від цієї функції за кожною з незалежних змінних, вважаючи другу незалежну змінну в цьому процесі параметром, прирівняти отримані перші частинні похідні нулю і спільно розв'язати отриману систему двох рівнянь з двома невідомими***. Цей розв'язок і дасть нам чисельне значення координат точки максимуму функції.

Формально цю умову можна записати так:

$$\begin{cases} \frac{\partial f_{12}(x_1,x_2)}{\partial x_1} = 0, \\ \frac{\partial f_{12}(x_1,x_2)}{\partial x_2} = 0, \end{cases} \tag{4.21}$$

$$\Downarrow$$

$$\{x_1 = g,\ \ x_2 = e,$$

$$\Downarrow$$

$$f_{12}(g,e) = \max_{\substack{x_1\in[a,b] \\ x_2\in[c,d]}} f_{12}(x_1,x_2), \tag{4.22}$$

Але цілком зрозуміло, що функцією $f_{12}(x_1, x_2)$ може описуватись не лише поверхня гори над площиною землі, але і поверхня провалля під площиною землі, тож із системи рівнянь (4.21) незрозуміло, що саме ми знайшли, розв'язавши їх, координати точки максимуму, які забезпечують рівність (4.22), чи точки мінімуму.

$$f_{12}(g,e) = \min_{\substack{x_1\in[a,b] \\ x_2\in[c,d]}} f_{12}(x_1,x_2)\,. \tag{4.23}$$

Узагальнюючи викладки, які ми наводили щодо характеру екстремуму функції однієї незалежної змінної, на функцію двох незалежних змінних, можна стверджувати: для того, ***щоб розв'язок системи рівнянь (4.21) задавав координати точки максимуму функції $f_{12}(x_1, x_2)$,*** тобто забезпечував виконання умови (4.22) ***достатньо виконання системи нерівностей відносно других частинних похідних цієї функції такого характеру***

$$\begin{cases} \frac{\partial^2 f_{12}(x_1=g,\ \ x_2=e)}{\partial x_1^2} \\ \frac{\partial^2 f_{12}(x_1=g,\ \ x_2=e)}{\partial x_2^2} \end{cases} < 0, \tag{4.24}$$

а для того, щоб цей розв'язок задавав координати ***точки мінімуму*** функції $f_{12}(x_1, x_2)$, достатньо виконання системи нерівностей відносно других частинних похідних цієї функції ***такого характеру***

$$\begin{cases} \frac{\partial^2 f_{12}(x_1=g,\ \ x_2=e)}{\partial x_1^2} \\ \frac{\partial^2 f_{12}(x_1=g,\ \ x_2=e)}{\partial x_2^2} \end{cases} > 0. \tag{4.25}$$

Якщо в умовах (4.24) або (4.25) одна з других частинних похідних дорівнює нулю, то точка (g, e) лежить на прямій, по якій відбувається перегин поверхні, що описується функцією $f_{12}(x_1, x_2)$.

Якщо в умовах (4.24) або (4.25) обидві другі частинні похідні дорівнюють нулю, то точка (g, e) лежить на плоскій ділянці поверхні, яка має місце на якійсь ділянці гори.

Якщо ж в умовах (4.24) або (4.25) одна друга частинна похідна більша за нуль, а інша – менша за нуль, то точка (g, e) ***є сідловою*** – це нижня точка перевалу між двома горами або верхня точка перевалу між двома проваллями.

Завершуючи виклад матеріалу, присвяченого дослідженням функцій багатьох змінних на екстремум, відзначимо, що якщо функція має більше двох незалежних змінних, то необхідні умови існування екстремуму відрізнятимуться від (4.21) лише більшою кількістю аналогічних рівнянь. Але достатні умови вже не можуть обмежуватись виконанням лише нерівностей (4.24), (4.25). Необхідно, щоб виконувались ще й деякі більш складні умови.

А тепер перейдемо до такого важливого розділу функціонального аналізу, яким є дослідження функціоналів на екстремум. І почнемо з зауваження , що с***укупність методів пошуку екстремумів функціоналів різних типів, із числа наведених нами вище, становить суть варіаційного числення***, для розуміння основ якого розглянемо функціонал (4.6).

Але, перш ніж визначати умови, за яких функціонал (4.6) набуватиме екстремального значення, уточнимо, що ми розумітимемо під поняттями абсолютний та відносний екстремуми функціонала, та що ми будемо розуміти під поняттям слабкий відносний екстремум функціонала.

За аналогією з тим, як для функції ми визначали глобальний і локальний екстремуми, для функціонала визначається абсолютний і відносний екстремуми, перший з яких задає найбільше (чи найменше) значення функціонала на всій множині функцій, на якій цей функціонал задано, а другий задає найбільше (чи найменше) значення функціонала на підмножині близьких функцій, які є лише частиною від усієї множини функцій, на яких задано цей функціонал.

В свою чергу серед відносних екстремумів розрізняють сильний і слабкий.

Для уніфікації підходів домовились вважати, що на екстремалі $y = f_e(x)$ досягається ***сильний відносний мінімум*** функціонала, якщо його з***начення*** на цій кривій у заданому діапазоні $[a, b]$ значень аргументу x ***є меншим***, ніж на усіх інших кривих $y = f_i(x), i = 1,2, \dots, n$ даного класу функцій, відстань нульового порядку

$$\Delta_0 = \max_{[a,b]} |f_e(x) - f_i(x)|, \ \ i = 1,2, \dots, n \qquad (4.26)$$

до яких мала. Очевидно, що ***сильний відносний максимум*** функціонала буде досягатись на екстремалі з цього ж класу кривих, якщо його ***значення*** буде в заданому діапазоні значень аргументу ***найбільшим***.

Якщо ж відносний мінімум (чи максимум) функціонала досягається на екстремалі, відстань першого порядку

$$\Delta_1 = \max_{[a,b]} |f_e'(x) - f_i'(x)|, \ \ i = 1,2, \dots, n \qquad (4.27)$$

від якої до усіх інших кривих даного класу функцій є малою, то на цій екстремалі має місце слабкий відносний мінімум (чи максимум).

Зрозуміло, що ***абсолютний екстремум є в той же час і відносним, а сильний відносний екстремум є в той же час і слабким.***

А тому, якщо якась умова має виконуватись відносно слабкого відносного екстремуму, то вона має бути справедливою і для сильного відносного екстремуму, і для абсолютного.

Розібравшись з наведеними вище поняттями, визначимо, які умови має задовольняти функція $y = f(x)$, щоб на ній мав місце слабкий відносний мінімум функціонала (4.6). І в роботі [1] показано, що ця функція має бути розв'язком рівняння Ейлера

$$F_y - \frac{d}{dx} F_y' = 0, \qquad (4.28)$$

в якому

$$F_y = \frac{\partial F(x,y,y')}{\partial y}, \quad F_{y'} = \frac{\partial F(x,y,y')}{\partial y'}. \tag{4.29}$$

Одразу ж зауважимо, що у варіаційному численні використовуються і такі позначення:

$$F_x = \frac{\partial F(x,y,y')}{\partial x}, \quad F_{y'y} = \frac{\partial^2 F(x,y,y')}{\partial y' \partial y}, \quad F_{y'y'} = \frac{\partial^2 F(x,y,y')}{\partial (y')^2} \tag{4.30}$$

А ***функція*** $\boldsymbol{y = y(x)}$***,*** яка є розв'язком рівняння Ейлера (4.28) ***називається екстремаллю функціонала.***

На використанні рівняння Ейлера в різних інтерпретаціях власне і базується варіаційне числення.

Розписавши рівняння Ейлера з використанням формули повної похідної від функції трьох змінних при диференціюванні складової $F_{y'}$, отримаємо

$$F_y - \left(\frac{\partial F_{y'}(x,y,y')}{\partial x} \frac{dx}{dx} + \frac{\partial F_{y'}(x,y,y')}{\partial y} \frac{dy}{dx} + \frac{\partial F_{y'}(x,y,y')}{\partial y'} \frac{\partial y'}{dx} \right) = 0, \tag{4.31}$$

або

$$F_y - F_{y'x} - F_{y'y} y' - F_{y'y'} y'' = 0 \tag{4.32}$$

З виразу (4.32) бачимо, що ***рівняння Ейлера є нелінійним диференціальним рівнянням другого порядку,*** для якого не існує єдиного способу розв'язання, оскільки в загальному випадку коефіцієнти і при похідних, і при вільному члені не є константами і навіть не функціями лише незалежної змінної x, а залежать і від функціональної змінної y, і від її похідної y'.

Оскільки для отримання розв'язку диференціального рівняння 2-го порядку його потрібно двічі інтегрувати, то в загальному вигляді цей розв'язок $\boldsymbol{y(x, C_1, C_2)}$ ***містить дві сталі інтегрування*** $\boldsymbol{C_1, C_2}$, для конкретизації яких до умов задачі оптимізації функціоналу (4.6) на відрізку $[a, b]$ використовуються граничні умови, що мають вигляд

$$\begin{cases} y(a, C_1, C_2) = y_a, \\ y(b, C_1, C_2) = y_b, \end{cases} \tag{4.33}$$

В деяких задачах дослідження функціоналів на екстремум рівняння Ейлера, яке є нелінійним диференціальним рівнянням другого порядку, зручно подавати у вигляді системи двох диференціальних рівнянь першого порядку, для отримання якої вводяться дві нові змінні p, H, пов'язані з функціоналом F, що досліджується співвідношеннями

$$F_{y'} = p, \tag{4.34}$$

$$F - y' F_{y'} = F - y' p = H, \tag{4.35}$$

використовуючи які, рівняння Ейлера у вигляді (4.28) легко приводиться до вигляду

$$\begin{cases} \frac{\partial H}{\partial p} = -\frac{dy}{dx}, \\ \frac{\partial H}{\partial y} = \frac{dp}{dx} \end{cases} \tag{4.36}$$

Систему рівнянь (4.36) прийнято називати канонічною формою подання рівняння Ейлера (4.28), а задачу пошуку екстремалі $\boldsymbol{y(x)}$***, яка доставляє екстремум функціоналу (4.6) і є кривою, защемленою на кінцях, в класичному варіаційному численні прийнято називати найпростішою.***

Наявність розв'язку рівняння Ейлера для екстремалі $y(x)$ є умовою, необхідною для того, ***щоб*** на ній досягався ***мінімум чи максимум*** функціонала (4.16). Але, як і у випадку екстремуму функції, необхідні умови екстремуму функціонала обов'язково потрібно доповнити умовами достатніми, за допомогою яких розпізнаються як ті функції, на яких максимум чи мінімум функціонала досягається, так і ті, на яких, незважаючи на виконання необхідних умов, функціонал екстремуму не досягає.

В роботі [1] ці достатні умови екстремуму функціонала (4.6) виведені строго, а тут ми наведемо лише кінцевий результат цього виведення. Отже, ***на екстремалі $y(x)$ в межах відрізка $[a, b]$ досягається мінімум функціонала (4.6), якщо*** для всіх $x \in [a, b]$ маємо

$$F_{y'y'} > 0, \tag{4.37}$$

і ***максимум, якщо***

$$F_{y'y'} < 0. \tag{4.38}$$

Умови (4.37), (4.38) є достатніми умовами досягнення на екстремалі $y(x)$ ***в межах*** $x \in [a, b]$ ***екстремуму функціонала (4.6). Ці умови часто називають умовами Лежандра***, за іменем математика, який їх вивів.

Всі викладки, що наведені вище, стосувалися задач пошуку екстремуму функціоналів в класі функцій, графіки яких защемлені на кінцях. Але ***існує*** досить ***багато задач*** оптимізації, ***в яких екстремалі*** необхідно шукати ***в класі функцій, кінці графіків яких є рухомими.*** Математично така задача формулюється так: серед кривих, кінці яких можуть рухатись по двох інших кривих – позначимо їх $\phi(x)$ і $\psi(x)$ (рисунок 4.7 в роботі [1]), – знайти криву $y(x)$, що доставляє мінімум функціоналу (4.6). В роботі [1] строго доведено, що екстремаль $y(x, C_1, C_2)$ крім того, що вона має бути розв'язком рівняння Ейлера (4.28), має ще й задовольняти умови

$$\begin{cases} \left(F - (y' - \phi')F_{y'}\right) \underset{x \to x_0}{\Longrightarrow} 0 \\ \left(F - (y' - \psi')F_{y'}\right) \underset{x \to x_1}{\Longrightarrow} 0 \end{cases} \tag{4.39}$$

або

$$\begin{cases} (H + \phi' p) \underset{x \to x_0}{\Longrightarrow} 0 \\ (H + \psi' p) \underset{x \to x_1}{\Longrightarrow} 0, \end{cases} \tag{4.40}$$

де x_0, x_1 – координати точок, в яких знаходиться екстремаль $y(x, C_1, C_2)$ на початку руху її кінців по кривих $\phi(x)$ і $\psi(x)$. ***Умови, що задаються виразами (4.39) або (4.40), називаються умовами трансверсальності***. Вони дозволяють знайти точки перетину екстремалі $y(x, C_1, C_2)$, яка є розв'язком рівняння Ейлера, з лініями $\phi(x)$ і $\psi(x)$, котрі задають закони зміни рухомих кінців цієї екстремалі. Оскільки рівняння Ейлера є диференціальним рівнянням другого порядку, то його загальний розв'язок, як ми уже відзначали, який задає множину екстремалей, містить дві довільні сталі C_1, C_2. Саме для конкретизації екстремалі ці сталі визначаються у цьому випадку не з системи рівнянь (4.33), а з системи рівнянь (4.39) або (4.40), кількість яких має дорівнювати кількості шуканих сталих.

Якщо ж екстремум функціонала досягається на кривій з точкою злому x_0, то двох рівнянь для визначення сталих уже буде недостатньо, оскільки їх значення C_1, C_2 для екстремалі на ділянці до точки злому і після цієї точки – C_3, C_4 – будуть різними, адже різними є рівняння, які задають екстремаль на цих ділянках. Зрозуміло, що для визначення чотирьох сталих у розв'язку рівняння Ейлера для екстремалей $y(x, C_1, C_2, C_3, C_4)$ з однією точкою злому x_0 на відрізку $[a, b]$ потрібно мати чотири рівняння. Два з них будуть задаватись граничними умовами (4.33), тобто значеннями екстремалі в точках $x = a$ і $x = b$, а два інших потрібно визначити в точці злому x_0, при наближенні до неї по екстремалі з лівого

боку, тобто з боку $(x_0 - 0)$, та з правого боку, тобто з боку $(x_0 + 0)$. В роботі [1] здійснене строге виведення цих рівнянь, які ми тут наведемо в їх в кінцевому вигляді, а саме:

$$\begin{cases} H(x_0 - 0) = H(x_0 + 0), \\ p(x_0 - 0) = p(x_0 + 0) \end{cases} \tag{4.41}$$

Умови (4.41) називають умовами Вейєрштрасса–Ердмана. Вони і задають ті додаткові два рівняння, які потрібно мати для однозначного визначення екстремалі $y(x, C_1, C_2, C_3, C_4)$ ***, отриманої шляхом розв'язання рівняння Ейлера зліва і справа від точки злому*** $\boldsymbol{x_0} \in [\boldsymbol{a}, \boldsymbol{b}]$ **.** Введенням цих умов ми розширили клас функцій, на яких може досягатись екстремум функціонала (4.6). І якщо раніше ми шукали екстремалі лише в класі гладких функцій, графіками яких є плавні криві, то тепер ми зможемо розв'язувати задачу мінімізації функціонала в більш широкому класі – класі кусково-гладких функцій, графіками яких є неперервні криві, що містять точки злому між плавними складовими графіка.

А тепер перейдемо до дослідження на екстремум функціоналів, що пов'язують між собою декілька невідомих функцій та їх перших похідних**,** тобто розглянемо задачу пошуку екстремуму функціонала (4.11), що зв'язує між собою множину функцій (4.12), яка задає поверхню в n-вимірному просторі, та множину їх перших похідних (4.13).

Алгоритм методу пошуку екстремалей функціонала (4.11) обґрунтований і детально проаналізований в роботі [1], тому тут ми лише наведемо його кінцевий результат. Отже, **множина функцій**

$$\{ y_1(x, C_1, C_{11}), y_2(x, C_2, C_{22}), \dots, y_n(x, C_n, C_{nn}) \}, \tag{4.42}$$

що містить $\boldsymbol{2n}$ ***сталих інтегрування*** $\boldsymbol{C_i, C_{ii}}$, $\boldsymbol{i = 1, 2, \dots, n}$ ***доставлятиме екстремум функціоналу (4.11) і буде множиною екстремалей цього функціонала, якщо вона є розв'язком системи рівнянь Ейлера, що мають вигляд***

$$\begin{cases} F_{y_1} - \frac{d}{dx} F_{y_1'} = 0, \\ F_{y_2} - \frac{d}{dx} F_{y_2'} = 0, \\ \dots\dots\dots\dots\dots\dots\dots, \\ F_{y_n} - \frac{d}{dx} F_{y_n'} = 0 \end{cases} \tag{4.43}$$

Цілком очевидно, що числові значення сталих інтегрування в рівняннях екстремалей у цьому випадку, як і у випадку функціонала (4.6), потрібно знаходити з системи рівнянь, яких тепер уже буде $2n$ і які синтезуватимуться з використанням граничних умов для кожної екстремалі та матимуть вигляд

$$\begin{cases} y_1(a, C_1, C_{11}) = y_{1a}, \\ y_1(b, C_1, C_{11}) = y_{1b}, \\ \dots\dots\dots\dots\dots\dots\dots, \\ y_n(a, C_n, C_{nn}) = y_{na}, \\ y_n(b, C_n, C_{nn}) = y_n b \end{cases} \tag{4.44}$$

Як і у випадку функціонала (4.6), тим, що ***множина функцій (4.42) є розв'язком системи (4.43), задається лише необхідна умова існування екстремуму функціонала*** (4.11). А для перевірки достатніх умов існування екстремуму функціонала (4.11) на множині екстремалей (4.42) потрібно, як і у випадку функціонала (4.6), переконатись, що виконуються ***умови Лежандра***, які для однієї екстремалі мали вигляд (4.37) для мінімуму і (4.38) для максимуму, а для системи екстремалей (4.42) ***для мінімуму*** матимуть вигляд

$$\begin{cases} F_{y_1'y_1'} > 0, \\ \begin{vmatrix} F_{y_1'y_1'} & F_{y_1'y_2'} \\ F_{y_2'y_1'} & F_{y_2'y_2'} \end{vmatrix} > 0, \\ \dots\dots\dots\dots\dots\dots\dots\dots, \\ \begin{vmatrix} F_{y_1'y_1'} & F_{y_1'y_2'} \dots . F_{y_1'y_n'} \\ F_{y_2'y_1'} & F_{y_2'y_2'} \dots\dots F_{y_2'y_n'} \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots \\ F_{y_n'y_1'} & F_{y_n'y_2'} \dots\dots F_{y_n'y_n'} \end{vmatrix} > 0 \end{cases} \tag{4.45}$$

Цілком очевидно, що ***достатні умови існування максимуму*** функціонала (4.11) на множині екстремалей (4.42***) за Лежандром*** будуть мати ***вигляд, аналогічний (4.45), але знаки*** нерівностей в них будуть ***протилежними***.

А тепер перейдемо до пошуку екстремумів функціоналів, що залежать від старших похідних невідомої функції, тобто, перейдемо до дослідження на екстремум функціоналів, що мають вигляд (4.7), які пов'язують між собою не лише функцію $y(x)$та її похідну $y'(x)$, як у функціоналі (4.6), але і старші похідні $y''(x), \dots, y^{(n)}(x)$. І почнемо з зауваження, що ще Ейлер в першій половині вісімнадцятого сторіччя евристично довів, що ***функція $\boldsymbol{y(x)}$ буде екстремаллю функціонала (4.7) у тому випадку, якщо вона є розв'язком рівняння***

$$F_y - \frac{d}{dx}F_{y'} + \frac{d^2}{dx^2}F_{y''} - \dots + (-1)^n \frac{d^n}{dx^n}F_{y^{(n)}} = 0 \tag{4.46}$$

А строге доведення того, що екстремаллю функціонала (4.7) є функція $y(x, C_1, C_2, \dots, C_{2n})$, яка є розв'язком рівняння (4.46), здійснив Пуассон. Ось чому це рівняння й увійшло у варіаційне числення з подвійною назвою « рівняння Ейлера–Пуассона».

Оскільки ***екстремаль $\boldsymbol{y(x, C_1, C_2, \dots, C_{2n})}$, у цьому випадку містить $\boldsymbol{2n}$ сталих інтегрування $\boldsymbol{C_i, i = 1, 2, \dots, 2n}$, кількість яких дорівнює подвоєному порядку $\boldsymbol{k}$ старшої похідної у функціоналі (4.7)***, то і рівнянь для їх визначення потрібно синтезувати теж $2n$, а тому і граничних умов потрібно задати стільки ж, для чого на границях відрізка $[a, b]$ задається не лише значення екстремалі, а її похідних до $(k-1)$-ої включно. Наприклад, якщо порядок старшої похідної у функціоналі (4.7) дорівнює двом, то сталих інтегрування потрібно визначати чотири, розв'язуючи систему чотирьох рівнянь, складених з використанням граничних умов, які матимуть вигляд:

$$\begin{cases} y(a, C_1, C_1, C_3, C_4) = y_a, \\ y'(a, C_1, C_1, C_3, C_4) = y_a', \\ \\ y(b, C_1, C_2, C_3, C_4) = y_b, \\ y'(b, C_1, C_2, C_3, C_4) = y_b' \end{cases} \tag{4.47}$$

Дуже ***простими для цієї задачі виявились умови Лежандра***, за якими мінімум функціонала (4.7) на екстремалі $y(x)$ відрізняють від максимуму. І суть їх полягає у тому, що для того, щоб на екстремалі $y(x)$ досягався ***мінімум*** функціонала (4.7), достатньо ***виконання умови***

$$F_{y^{(n)}y^{(n)}} > 0, \tag{4.48}$$

а для ***максимуму*** –

$$F_{y^{(n)}y^{(n)}} < 0. \tag{4.49}$$

Отже, ми дали відповіді на всі поставлені на початку цього підрозділу запитання, а тому маємо уже цілком достатньо інформації для програмування цих відповідей мовою Python.

4.2 Додаткові відомості з мови програмування Python, достатні для розв'язання задач, пов'язаних з дослідженнями функцій та функціоналів на безумовний екстремум

Аналізуючи характер формул, викладених у попередньому підрозділі 4.1, бачимо, що для їх реалізації в програмах мовою Python нам знадобиться знання низки програмних функцій цієї мови програмування, про які ще не йшла мова в підрозділах, присвячених викладенню додаткових відомостей з технології програмування мовою Python, оскільки ні про диференціювання функцій та розв'язання алгебраїчних рівнянь, які потрібні для дослідження на екстремум функцій, ні про розв'язання диференціальних рівнянь, які потрібні для дослідження на екстремум функціоналів, мова раніше ще не йшла. Тому до того, що ми уже знаємо стосовно технології програмування мовою Python, додамо нижче, використовуючи як базу роботу [4], і ці відомості.

Отже, ***реалізацію*** програмних функцій ***диференціювання*** нам потрібно ***починати*** з ***виклику ППП sympy***, про використання програмних функцій якого нами вже наведена достатньо повна інформація в підрозділі 2.2. ***Однократне диференціювання*** за однією незалежною змінною ***здійснюється*** програмною функцією ***diff(,)***, в аргументних дужках якої першим стоїть вираз, що диференціюється, а другим змінна, за якою здійснюється диференціювання, а ***n-кратне диференціювання*** здійснюється ***цією*** ж програмною ***функцією***, але в аргументних дужках додається ***третім аргументом число***, яким конкретизується кількість ***диференціювань.*** В разі необхідності визначення ***змішаних частинних похідних*** від виразу, що містить ***дві незалежні змінні***, ***в*** аргументних ***дужках*** цієї ж програмної ***функції*** ми зобов'язані записати окрім виразу, що диференціюється, ***на другому і третьому місцях обидві незалежні змінні***, за якими здійснюється диференціювання, а якщо потрібно визначити ***змішану частинну похідну з порядком, вищим першого***, програмна ***функція*** уже матиме ***5 аргументів***, першим із яких виступатиме вираз, що диференціюється, ***другим – перша*** незалежна ***змінна***, ***третім – число*** диференціювань ***за першою*** змінною, ***четвертим – друга*** незалежна ***змінна***, а ***п'ятим – число*** диференціювань ***за другою*** змінною. 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***, за якою 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 за нею***.

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, що вище сказане у цьому абзаці, проілюстроване в ***пояснювальному прикладі № 31.***

Пояснювальний приклад № 31

```
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 від виразу,
Out[5]:                                         ….що стоїть в дужках та візуалізація
(9*x**2 - 12*x + 2)*exp(-3*x)                   ….результату
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 за x та 2-го порядку
12                                              ….за y від функції f(x,y) і її візуалізація
In 8]: diff(f,x,y)                              # Визначення змішаної частинної
Out[8]:                                         ….похідної 1-го порядку за x та y від
6*x**2*y                                        …. функції f(x,y) і її візуалізація
In [9]: f.diff(x,y)                             # Визначення змішаної частинної
Out[9]:                                         ….похідної 1-го порядку за 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 за x та 2-го порядку
6*x*y*(x**2*y**3*(3*x**2*y**3 + 2) +            …. за y від функції f1 і її візуалізація
+ 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 за x та 2-го порядку
6*x*y*(x**2*y**3*(3*x**2*y**3 + 2) +            …. за y від функції f1 методом f2.doit( )
+ 6*x**2*y**3 + 2)*exp(x**2*y**3)               ….і її візуалізація
```

Кінець пояснювального прикладу № 31.

Оскільки при дослідженні функції на екстремум окрім диференціювання ***потрібно*** ще й ***розв'язувати алгебраїчні рівняння***, що утворюють прирівнюванням до нуля похідної від заданої функції, 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; і записується функція ***Eq( )*** першою в аргументних дужках функції ***solveset ( ),*** а другою в цих дужках записується змінна, стосовно якої складається алгебраїчне рівняння і корені з якої з цього рівняння визначаються та записуються як результат у наступному рядку у вигляді множини чисел. Варто 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 до нуля (без запису нуля), а другим в цих дужках записується змінна, стосовно якої складається алгебраїчне рівняння і корені з якої 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 й 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 до нуля, або список виразів, що задають систему рівнянь, а другим в цих дужках записується або змінна, стосовно якої складається алгебраїчне рівняння і корені з якої з цього рівняння визначаються та записуються як результат у наступному рядку, або записуються ті змінні, стосовно яких складається система алгебраїчних рівнянь, корені з яких з 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 перший аргумент, слідом за яким записуються змінні, стосовно яких складені ці лінійні рівняння. Усе, що сказано вище стосовно розв’язання алгебраїчних рівнянь, проілюстроване в ***пояснювальному прикладі № 32.***

Пояснювальний приклад № 32

```
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( , ) та візуалізація
FiniteSet(3, -3/2 - 3*sqrt(3)*I/2, -3/2 +     …. результату розв’язання рівняння
+ 3*sqrt(3)*I/2)
In [7]: solveset((expr1-expr2),x)             # Розв’язання рівняння з використанням
Out[7]:                                       ….функції expr1-expr2  та візуалізація
FiniteSet(3, -3/2 - 3*sqrt(3)*I/2, -3/2 +     …. результату розв’язання рівняння
+ 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 функції expr
Out[9]:                                       ….та їх візуалізація у вигляді словника
 {-2: 1, -3: 1, 1: 2}                         ….з вказанням значення та кратності
In [10]: real_roots(x**3-27,x)                # Визначення дійсних коренів
Out[10]:                                      ….рівняння x**3 - 27 = 0
[3]                                           ….та їх візуалізація
In [11]: solve([x**2-y-5,x+2*y**2-4],x,y)     # Розв’язання системи рівнянь,
Out[174]:                                     ….заданих списком, та
[(2, -1)]                                     ….візуалізація результату
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))                             ….та візуалізація результату
```

Кінець пояснювального прикладу № 32.

Дослідження функціоналів на екстремум передбачають етап визначення екстремалей шляхом розв’язання рівнянь Ейлера, Ейлера–Пуассона та Ейлера–Лагранжа, які відносять до класу нелінійних диференціальних рівнянь. А тому ***потрібно знати*** як створювати ***програми мовою Python для розв’язання цих рівнянь. В*** програмних ***пакетах sympy та scipy є кілька програмних функцій***, які розв’язують диференціальні рівняння та системи

диференціальних рівнянь. В пакеті ***sympy*** – це програмна функція ***dsolve( )***, яка використовується для символьного розв`язання звичайних диференціальних рівнянь, а в пакеті ***scipy*** – це програмні функції ***ode( )*** та ***odeint( )***, які входять до структури модуля ***scipy.integrate.*** Оскільки програмні функції пакета ***sympy*** виводять результат розв'язання диференціальних рівнянь у вигляді аналітичних виразів, то, на перший погляд, їх використання є пріоритетним. Але, оскільки цим програмним функціям під силу не кожне диференціальне рівняння, то у більшості практичних задач, в яких потрібно розв'язувати диференціальні рівняння, доцільніше застосовувати для їх розв'язання програмні функції пакета ***scipy***, за допомогою яких розв'язок отримується в чисельній формі, але при цьому в обов'язковому порядку доведеться викликати і пакет ***numpy***, оскільки при чисельних розрахунках ***потрібно задавати масиви*** значень незалежної змінної, ***які є об'єктами*** саме цього пакета ***мовою Python.*** Як правило, ***розв'язки*** диференціальних рівнянь ***характеризують розвиток у часі процесів***, які ними описуються, тож корисним є графічне відображення цих розв'язків. І у цьому випадку не обійтись без виклику ще й пакета ***matplotlib*** для реалізації ***підпрограми побудови графіків*** цих розв'язків. Ми теж будемо притримуватись цього підходу, використовуючи ті відомості стосовно побудови графіків, які були викладені в пояснювальних прикладах раніше. ***Звертаємо увагу*** на те, що ***при символьному розв'язанні*** диференціальних рівнянь програмними функціями пакета ***sympy*** оголошувати ***символьними*** потрібно не лише імена незалежних змінних, ***але 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. В аргументні дужки програмної функції ***dsolve(,)*** окремою опцією ***hint*** 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'.*** Усе, що сказано вище у цьому абзаці стосовно розв'язання диференціальних рівнянь з використанням програмних функцій пакета ***sympy***, проілюстровано в ***пояснювальному прикладі № 33.***

Пояснювальний приклад № 33

```
In [1]: import sympy                                  # Виклик ППП sympy
In [2]: from sympy import *                           # Виклик із sympy усіх функцій
In [4]: x=symbols('x')                                # Оголошення символьною змінної x
In [5]: f=Function('f')                               # Оголошення символьною  функції f
In [6]: dsolve(Eq(f(x).diff(x,2)+                     # Розв'язання диференціального
+3*f(x).diff(x)+2*f(x),2*x),f(x))                     ….рівняння ДР, заданого програмною
Out[6]:                                               ….функцією Eq(ДР), відносно функції
Eq(f(x), C1*exp(-2*x) + C2*exp(-x) +                  …. f(x )і візуалізація результату y
+ x - 3/2)                                            …. вигляді програмної функції Eq(f(x))
In [7]: t=symbols('t')                                # Оголошення символьною змінної t
In [8]: y=Function('y')                               # Оголошення символьною  функції y
In [9]: eq=Eq(y(t).diff(t,2)+2*y(t).diff(t),sin(3*t)) # Формування функції Eq(ДР), як eq
In [10]: rez=dsolve(eq,y(t));rez                      # Розв'язання диференціального
Out[10]:                                              ….рівняння ДР, заданого функцією eq,
Eq(y(t), C1 + C2*exp(-2*t) - sin(3*t)/13 –            …. відносно функції y(t) і у вигляді
-2*cos(3*t)/39)                                       …. Eq(y(t),) візуалізація результату
In [11]: rez=dsolve(eq,hint='best');rez               # Перевірка результату розв'язання
```

Out[11]: …***ДР на «найкращість»***

Eq(y(t), C1 + C2*exp(-2*t) - sin(3*t)/13 –
-2*cos(3*t)/39)

In [12]: from IPython.display import * # ***Виклик*** із ***IPython.display*** усіх функцій

In [13]: init_printing(use_latex=True) # ***Запуск*** режиму ***«красивого»*** друку

In [14]: var('t C1 C2') # ***Оголошення символьними t,C1,C2***

Out[14]: (t, C1, C2)

In [15]: u=Function('u')(t) # ***Оголошення символьною*** функції ***u(t)***

In [16]: de=Eq(u.diff(t,2)+3*u.diff(t)+2*u,2*t) # ***Запис ДР*** у формі ***Eq(ДР)***

In [17]: display(de) # ***«Красива» візуалізація ДР***

$$2u(t) + 3\frac{d}{dt}u(t) + \frac{d^2}{dt^2}u(t) = 2t$$

In [18]: des=dsolve(de,u) # ***Розв'язання ДР***

In [19]: display(des) # ***«Красива» візуалізація*** розв'язку ***ДР***

$$u(t) = C_1e^{-2t} + C_2e^{-t} + t - \frac{3}{2}$$

In [20]: eq1=des.rhs.subs(t,0);eq1 # ***Формування «красивої»*** правої частини

Out[20]: ….функції ***u(t)*** у формі ***u(0)***

$$C_1 + C_2 - \frac{3}{2}$$

In [21]: eq2=des.rhs.diff(t).subs(t,0);eq2 # ***Формування «красивої»*** правої частини

Out[21]: ….похідної $u'(t)$ у формі $u'(0)$

$$-2C_1 - C_2 + 1$$

In [22]: seq=solve([eq1,eq2-1],C1,C2);seq # ***Розв'язання*** системи рівнянь ***eq1,eq2*** для

Out[22]: ….початкових умов $u(0) = 0,\ u'(0) = 1$

$$\left\{C_1 : -\frac{3}{2}, C_2 : 3\right\}$$

In [23]: rez=des.rhs.subs([(C1,seq[C1]), # ***Підстановка*** значень констант $C1, C2$

(C2,seq[C2])]) ….з отриманого словника ***в розв'язок ДР des***

In [24]: display(rez) # ***«Красивий» друк*** правої частини ***розв'язку***

$$t - \frac{3}{2} + 3e^{-t} - \frac{3}{2}e^{-2t}$$

In [25]: t=symbols('t') # ***Оголошення символьною*** змінної ***t***

In [26}: x,y,z=symbols('x y z', cls = Function) # ***Оголошення символьними*** функцій ***x,y,z***

In [27]: eq1=Eq(x(t).diff(t),-x(t)+y(t)+z(t)) # ***Формування 1-го рівняння*** системи ДР

In [28]: eq2=Eq(y(t).diff(t),-x(t)+y(t)-z(t)) # ***Формування 2-го рівняння*** системи ДР

In [29]: eq3=Eq(z(t).diff(t), x(t)-y(t)+z(t)) # ***Формування 3-го рівняння*** системи ДР

In [30]: rez=dsolve((eq1,eq2,eq3)) # ***Розв'язання*** системи ***ДР***

In [31]: display(rez[0]) # ***«Красивий» друк*** складової ***x(t)*** розв'язку

$$x(t) = -3C_1e^{-t} + C_2$$

In [32]: display(rez[1]) # ***«Красивий» друк*** складової ***y(t)*** розв'язку

$$y(t) = -C_1e^{-t} + C_2 - C_3e^{2t}$$

In [33]: display(rez[2]) # ***«Красивий» друк*** складової ***z(t)*** розв'язку

$$z(t) = C_1e^{-t} + C_3e^{2t}$$

In [34]: eq4=rez[0].rhs.subs(t,0);eq4 # ***Формування «красивої»*** правої частини

Out[34]: ….функції ***x(t)*** у формі ***x(0)***

$$-3C_1 + C_2$$

In [35]: eq5=rez[1].rhs.subs(t,0);eq5 # ***Формування «красивої»*** правої частини

Out[35]: ….функції ***y(t)*** у формі ***y(0)***

$$-C_1 + C_2 - C_3$$

In [36]: eq6=rez[2].rhs.subs(t,0);eq6 # ***Формування «красивої»*** правої частини
Out[36]: ….функції ***z(t)*** у формі ***z(0)***

$$C_1 + C_3$$

In [37]: var('C1 C2 C3') # ***Оголошення*** символьними ***C1,C2,C3***
Out[37]:

$$(C1, C2, C3)$$

In [38]: seq1=solve([eq4+1,eq5-1,eq6], # ***Розв'язання*** системи рівнянь ***eq4,eq5,eq6***
C1,C2,C3);seq1 ….для початкових умов: ***x(0)=-1, y(0)=1,***
Out[38]: ,,,,***z(0)=0*** та ***візуалізація*** розв'язку ***словником***

$$\left\{C1: \frac{2}{3}, C2: 1, C3: \frac{1}{3}\right\}$$

In [39]: rez1=rez[0].rhs.subs([(C1,seq1[C1]), # ***Підстановка*** значень констант $C1, C2$
(C2,seq1[C2])]);rez1 ….із словника ***в розв'язок rez[0] системи ДР***
Out[39]: ….і ***«красивий» друк*** його правої частини

$$1 - 2e^{-t}$$

In [40]: rez2=rez[1].rhs.subs([(C1,seq1[C1]), # ***Підстановка*** значень констант $C1, C2, C3$
(C2,seq1[C2]),(C3,seq1[C3])]);rez2 ….із словника ***в розв'язок rez[1] системи ДР***
Out[40]: ….і ***«красивий» друк*** його правої частини

$$1 - \frac{1}{3}e^{2t} - \frac{2}{3}e^{-t}$$

In [41]: rez3=rez[2].rhs.subs([(C1,seq1[C1]), # ***Підстановка*** значень констант $C1, C3$
(C3,seq1[C3])]);rez3 ….із словника ***в розв'язок rez[2] системи ДР***
Out[41]: ….і ***«красивий» друк*** його правої частини

$$\frac{1}{3}e^{2t} + \frac{2}{3}e^{-t}$$

Кінець пояснювального прикладу № 33.

Що ж до застосовування для розв'язання диференціальних рівнянь програмних функцій пакета ***scipy***, за допомогою яких розв'язок отримується в числовій формі, то, 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 приведення 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 є функцією правих частин системи ***n диференціальних рівнянь 1-го порядку func(y,t)***, де $y = [y_1, y_2, \dots y_n]$, другий аргумент – список початкових значень $y(t_0) = [y_1(t_0), y_2(t_0), \dots, y_n(t_0)]$, а третій аргумент – це масив значень незалежної змінної t, в які ми хочемо визначити значення розв'язку $y(t)$. Що ж стосується програмної функції ***ode( )***, то вона розв'язує і ті диференціальні рівняння, з якими не може впоратись більш проста за інтерфейсом програмна функція ***odeint(, ,).*** Усе, що сказано вище у цьому абзаці стосовно розв'язання диференціальних рівнянь з використанням програмних функцій пакета ***scipy***, проілюстроване в ***пояснювальному прикладі №34.***

Пояснювальний приклад № 34

```
In [1]: import numpy as np                          # Виклик ППП numpy як np
In [2]: from scipy.integrate import odeint          # Виклик з модуля функції odeint
In [3]: import matplotlib.pyplot as plt             # Виклик модуля mpl.pyplot як plt
In [4]: def dydt(y,t):                              # Формування правої частини
            return np.exp(-2*t)*np.sin(3*t)         ….диференціального рівняння

In [5]: t=np.linspace(0.0,4.0,41)                   # Формування масиву для  t  в питру
In [6]: y0=0.2                                      # Внесення початкової умови
In [7]: y=odeint(dydt,y0,t)                         # Розв'язання диф. рівняння
In [8]: y=np.array(y).flatten( )                    # Трансформація розв'язку в масив
In [9]: fig=plt.figure(facecolor='white')           # Формування поля рисунку
<Figure size 432x288 with 0 Axes>

In [10]: plt.plot(t, y,'-r',linewidth=3)            # Побудова графіка функції
```

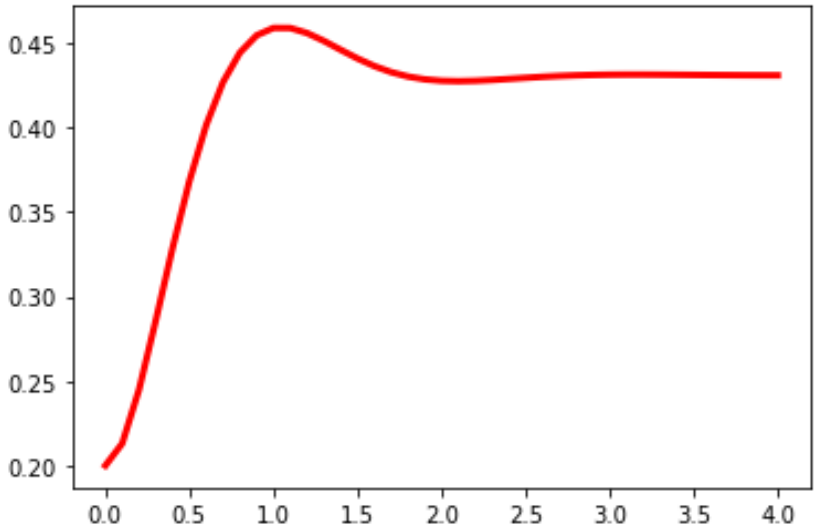

Рисунок 10 – Графік розв'язку диференціального рівняння $\frac{dy}{dt} = e^{-2t}\sin(3t)$ на відрізку значень $t \in [0,4]$ з початковими умовами $y(0) = 0,2$

```
In [11]: def f(y,t):                                # Формування правих частин
             y1,y2 = y                              ….системи диференціальних
             return [y2-y1,y2+y1]                   …. рівнянь

In [12]: t=np.linspace(0.0,4.0,41)                  # Формування масиву для  t  в питру
In [13]: y0=[1,1]                                   # Внесення початкових умов
In [14]: w=odeint(f,y0,t)                           # Розв'язання системи диф. рівнянь
```

```
In [15]: y1=w[:,0]                                  # Виокремлення розв'язку першого ДР
In [16]: y2=w[:,1]                                  # Виокремлення розв'язку другого ДР
In [17]: fig=plt.figure(facecolor='white')          # Формування поля рисунка
<Figure size 432x288 with 0 Axes>
In [18]: plt.plot(t,y1,'-r',t,y2,'-g',linewidth=2)  # Графіки розв'язків
Out[18]:
[<matplotlib.lines.Line2D at 0x2b3e8b29070>,
 <matplotlib.lines.Line2D at 0x2b3e8b29130>]
```

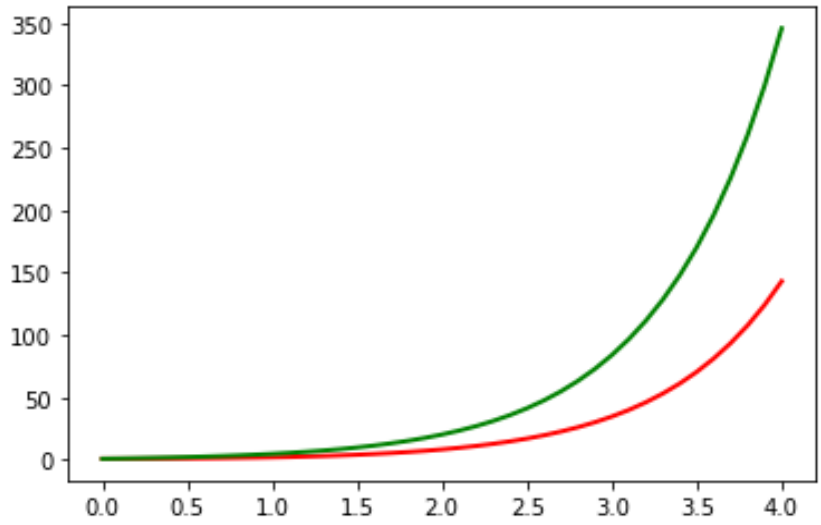

Рисунок 11 – Графік розв'язку системи диференціальних рівнянь $\frac{dy_1}{dt} = y_2 - y_1$, $\frac{dy_2}{dt} = y_2 + y_1$ на відрізку значень $t \in [0,4]$ з початковими умовами $y(0) = [1,1]$

```
In [19]: def f(y,t):                                # Формування правих частин
             y1,y2=y                                ….системи ДР 1-го порядку після
             return [y2,-t*y2-y1**0.5+np.sin(2*t)]  ….трансформації ДР 2-го порядку
                                                    ….у цю систему
In [20]: t=np.linspace(0,2,21)                      # Формування масиву для  t  в питру
In [21]: y0=[0,0.5]                                 # Внесення початкових умов
In [22]: w=odeint(f,y0,t)                           # Розв'язання системи диф. рівнянь
In [23]: y1=w[:,0]                                  # Виокремлення розв'язку ДР
In [24]: fig=plt.figure(facecolor='white')          # Формування поля рисунку
<Figure size 432x288 with 0 Axes>
In [25]: plt.plot(t,y1,'-r',linewidth=3)            # Графік розв'язку
Out[25]: [<matplotlib.lines.Line2D at 0x2b3e8b8adc0>]
```

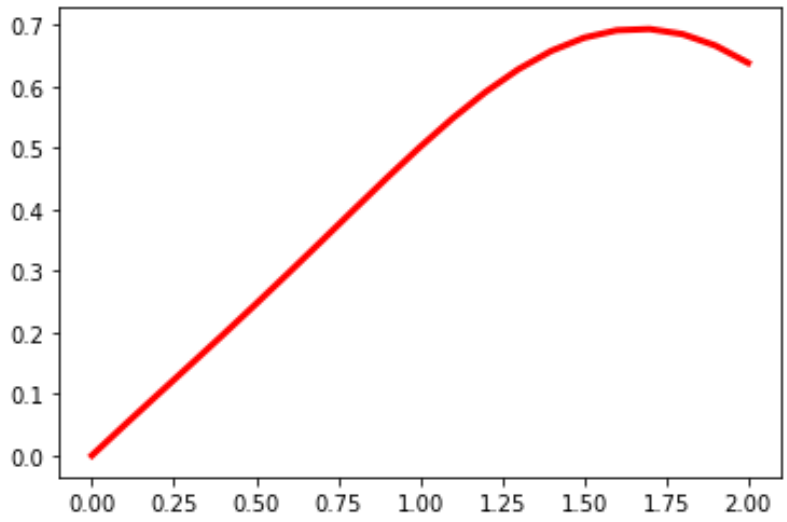

Рисунок 12 – Графік розв'язку диференціального рівняння $\frac{d^2y}{dt^2} + t\frac{dy}{dt} + \sqrt{y} = \sin(2t)$ на відрізку значень $t \in [0,2]$ з початковими умовами $y(0) = 0,\ y'(0) = 0{,}5$

Кінець пояснювального прикладу № 34.

4.3 Задачі дослідження функцій та функціоналів на безумовний екстремум в програмах мовою Python

Програму мовою Python ***для*** дослідження на ***екстремум*** функції $\boldsymbol{f(x)}$ однієї незалежної змінної $\boldsymbol{x}$ ми ***будемо складати, використовуючи формули (4.14)–(4.20)***, причому ми впишемо в одну програму і випадок, коли функція має максимум, і випадок, коли функція має мінімум.

Програма мовою Python для дослідження на екстремум функції $\boldsymbol{f_1(x) = -\frac{1}{3}x^3 + x - 1}$ дійсної змінної $\boldsymbol{x}$ на відрізку [-2,2]

(Програма 12)

```
In [1]: import sympy                    # Виклик ППП sympy
In [2]: from sympy import *             # Виклик із sympy усіх функцій
In [3]: x=symbols('x')                  # Оголошення символьною змінної x
In [4]: f1=-x**3/3+x-1                  # Формування функції f1
In [5]: f11=diff(f1,x);f11              # Диференціювання функції f1
Out[5]:                                 ….і візуалізація її першої похідної f11
                                            1 − x**2
In [6]: sol1=solve(f11,x);sol1          # Розв'язання рівняння f11 = 0
Out[6]:                                 ….і візуалізація списку його коренів
                                            [−1,1]
```

```
In [7]: f12=diff(f11,x);f12                 # Диференціювання похідної f11 від f1
Out[7]:                                     ….і візуалізація другої похідної f12
                                                −2x
In [8]: f12.subs(x,sol1[1])                 # Обчислення другої похідної при x=1
Out[8]:                                     ….і візуалізація результату
−2                                          ….та з'ясування, що маємо максимум
In [9]: f1.subs(x,sol1[1])                  # Обчислення максимуму функції f1
Out[9]:                                     ….та візуалізація результату
-1/3
In [10}: f12.subs(x,sol1[0])                # Обчислення другої похідної при x=-1
Out[10]:                                    ….і візуалізація результату
2                                           ….та з'ясування, що маємо мінімум
In [11]: f1.subs(x,sol1[0])                 # Обчислення мінімуму функції f1
Out[11]:                                    ….та візуалізація результату
-5/3
```

Кінець програми 12

Програма мовою Python для дослідження на екстремум функції $f_2(x,y) = x^3 + y^3 + 2x^2 + 2y^2 + xy + x + y + 5$ дійсних змінних x, y , заданих у квадраті $x \in [-2, 2], y \in$[-2,2]. (Програма 13)

```
In [1]: import sympy                        # Виклик ППП sympy
In [2]: from sympy import *                 # Виклик із sympy усіх функцій
In [3]: import numpy                        # Виклик ППП numpy
In [4]: from numpy import *                 # Виклик із numpy усіх функцій
In [5]: x,y=symbols('x y')                  # Оголошення символьними змінних x,y
In [6]: f2=x**3+y**3+2*x**2+                # Формування функції f2
+2*y**2+x*y+x+y+5
In [7]: f21x=diff(f2,x);f21x                # Диференціювання функції f2 за x
Out[7]:                                     ….і візуалізація її першої частинної
3*x**2 + 4*x + y + 1                        ….похідної f21x за x
In [8]: f21y=diff(f2,y);f21y                # Диференціювання функції f2 за y
Out[8]:                                     ….і візуалізація її першої частинної
x + 3*y**2 + 4*y + 1                        ….похідної f21y за y
In [9]: sol2=solve([f21x,f21y],x,y);sol2    # Розв'язання системи рівнянь
Out[9]:                                     ….f21x = 0, f21y = 0
[(-1, 0),                                   ….і візуалізація списку його коренів
 (0, -1),
 (-(-3/2 + sqrt(13)/2)*(1/6 + sqrt(13)/6),
 -5/6 + sqrt(13)/6),
 (-(1/6 - sqrt(13)/6)*(-sqrt(13)/2 - 3/2),
 -5/6 - sqrt(13)/6)]
In [10]: f22x=diff(f21x,x);f22x             # Диференціювання частинної похідної
Out[10]:                                    …. f21x за x і візуалізація другої
6*x + 4                                     …. частинної похідної f22x від f2 за x
In [11]: f22y=diff(f21y,y);f22y             # Диференціювання частинної похідної
Out[11]:                                    …. f21y за y і візуалізація другої
6*y + 4                                     …. частинної похідної f22y від f2 за y
In [12]: f22x.subs(x,sol2[0][0])            # Обчислення другої похідної f22x
Out[12]:                                    ….в екстремальній точці sol2[0]
-2                                          ….і візуалізація результату
```

```
In [13]: f22y.subs(y,sol2[0][1])
Out[13]:
4
In [14]: f22x.subs(x,sol2[1][0])
Out[14]:
4
In [15]: f22y.subs(y,sol2[1][1])
Out[15]:
-2
In [16]: f22x.subs(x,sol2[2][0])
Out[16]:
-6*(-3/2 + sqrt(13)/2)*(1/6 +
+ sqrt(13)/6) + 4
In [17]: -6*(-3/2 + 13**0.5/2)*(1/6 +
+ 13**0.5/6) + 4
Out[17]:
2.6055512754639896
In [18]: f22y.subs(y,sol2[2][1])
Out[18]:
-1 + sqrt(13)
In [19]: -1+13**0.5
Out[19]:
2.605551275463989
In [20]: f22x.subs(x,sol2[3][0])
Out[20]:
-6*(1/6 - sqrt(13)/6)*(-sqrt(13)/2 -
- 3/2) + 4
In [21]: -6*(1/6 - 13**0.5/6)*
(-13**0.5/2 - 3/2) + 4
Out[21]:
-4.605551275463991
In [22]: f22y.subs(y,sol2[3][1])
Out[22]:
-sqrt(13) – 1
In [23]: -13**0.5 – 1
Out[23]:
-4.605551275463991
In [24]: f2.subs([(x,sol2[0][0]),
(y,sol2[0][1])])
Out[24]:
5
In [25]: f2.subs([(x,sol2[1][0]),
(y,sol2[1][1])])
Out[25]:
5
In [26]: f2.subs([(x,sol2[2][0]),
(y,sol2[2][1])])
Out[26]:
-(-3/2 + sqrt(13)/2)*(1/6 + sqrt(13)/6) +
+ (-5/6 + sqrt(13)/6)**3 - (-3/2 +
```

 # *Обчислення* другої похідної *f22y*
….в екстремальній точці ***sol2[0]***
….і ***візуалізація результату***
 # *Обчислення* другої похідної *f22x*
….в екстремальній точці ***sol2[1]***
….і ***візуалізація результату***
 # *Обчислення* другої похідної *f22y*
….в екстремальній точці ***sol2[1]***
….і ***візуалізація результату***
 # *Обчислення* другої похідної *f22x*
….в екстремальній точці ***sol2[2]***
….і ***візуалізація результату***

 # *Числове обчислення результату*
….попередньої операції після
….подання його у вигляді,
….придатному для обчислень у ***питру***
 # *Обчислення* другої похідної *f22y*
….в екстремальній точці ***sol2[2]***
….і ***візуалізація результату***
 # *Числове обчислення результату*
….попередньої операції у ***питру***

 # *Обчислення* другої похідної *f22x*
….в екстремальній точці ***sol2[3]***
….і ***візуалізація результату***

 # *Числове обчислення результату*
….попередньої операції у ***питру***
….і ***візуалізація результату***

 # *Обчислення* другої похідної *f22y*
….в екстремальній точці ***sol2[3]***
….і ***візуалізація результату***
 # *Числове обчислення результату*
….попередньої операції у ***питру***
….і ***візуалізація результату***

 # *Обчислення* значення функції *f2*
….***у сідловій*** точці ***sol2[0]***
….і ***візуалізація*** результату

 # *Обчислення* значення функції *f2*
….***у сідловій*** точці ***sol2[1]***
….і ***візуалізація*** результату

 # *Обчислення* значення функції *f2*
….***у*** точці ***мінімуму sol2[2]***
….і ***візуалізація*** результату

```
+ sqrt(13)/2)**3*(1/6 + sqrt(13)/6)**3 –
- (-3/2 + sqrt(13)/2)*(-5/6 +
+ sqrt(13)/6)*(1/6 + sqrt(13)/6) +
+2*(-5/6 + sqrt(13)/6)**2 + 2*(-3/2 +
+ sqrt(13)/2)**2*(1/6 + sqrt(13)/6)**2 +
+ sqrt(13)/6 + 25/6
In [27]:  -(-3/2 + 13**0.5/2)*(1/6 +                # Числове обчислення результату
+13**0.5/6) + (-5/6 + 13**0.5/6)**3                  ….попередньої операції після
- (-3/2 + 13**0.5/2)**3*(1/6 + 13**0.5/6)**3-       ….подання його у вигляді,
 - (-3/2 + 13**0.5/2)*(-5/6 +13**0.5/6)*(1/6 +      ….придатному для обчислень у питру,
+ 13**0.5/6) + 2*(-5/6 + 13**0.5/6)**2 +
+ 2*(-3/2 + 13**0.5/2)**2*(1/6 + 13**0.5/6)**2 +
+ 13**0.5/6 + 25/6
Out[27]:                                             ….і візуалізація результату
4.780145063314225
In [28]: f2.subs([(x,sol2[3][0]),(y,sol2[3][1])])    # Обчислення значення функції f2
Out[28]:                                             ….у точці максимуму sol2[3]
(-5/6 - sqrt(13)/6)**3 - (1/6 - sqrt(13)/6)**3*      ….і візуалізація результату
(-sqrt(13)/2 - 3/2)**3 - (1/6 - sqrt(13)/6)*
(-sqrt(13)/2 - 3/2) - sqrt(13)/6 - (-5/6 –
- sqrt(13)/6)*(1/6 - sqrt(13)/6)*(-sqrt(13)/2 –
- 3/2) + 2*(-5/6 - sqrt(13)/6)**2 + 2*(1/6 –
- sqrt(13)/6)**2*(-sqrt(13)/2 - 3/2)**2 + 25/6
In [29]: (-5/6 - 13**0.5/6)**3 - (1/6 –              # Числове обчислення результату
-13**0.5/6)**3*(-13**0.5/2 - 3/2)**3 –               ….попередньої операції після
-(1/6 - 13**0.5/6)*(-13**0.5/2 –                     ….подання його у вигляді,
- 3/2) - 13**0.5/6 - (-5/6 - 13**0.5/6)*(1/6 –       ….придатному для обчислень у питру,
-13**0.5/6)*(-13**0.5/2 - 3/2) + 2*(-5/6 –
13**0.5/6)**2 + 2*(1/6 - 13**0.5/6)**2*(-13**0.5/2 –
-3/2)**2 + 25/6
Out[29]:                                             ….і візуалізація результату
 6.516151232982072
```

Кінець програми 13.

Програма мовою Python для дослідження на безумовний екстремум функціонала

$$J_1 = \int_a^b F_1(t, y, y')dt$$

у випадку, коли $a = 0$, $b = 1$, $F_1(t, y, y') = t^2 + y^2 + ty + (y')^2$, а екстремаль $y(t)$ починається в точці$(y(0) = 0,\ y'(0) = 1)$

(Програма 14)

```
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)   # Виклик функції «красивого» друку
In [5]: t=symbols('t')                  # Оголошення символьною змінної t
In [6]: y=Function('y')(t)              # Оголошення символьною функції y(t)
In [7]: z=Function('z')(t)              # Оголошення символьною функції z(t)
```

```
In 8]: z=y.diff(t)                    # Оголошення функції z(t) похідною від y(t)
In [9]: u=Function('u')(t)            # Оголошення символьним виразу u(t)
In [10]: u=t**2+y**2+t*y+z**2         # Формування ядра функціоналу як u(t)
In [11]: de=Eq(u.diff(y)-u.diff(z,t),0)  # Формування рівняння Ейлера для u(t)
In [12]: display(de)                  # Виклик на екран рівняння Ейлера
```

$$t + 2y(t) - 2\frac{d^2}{dt^2}y(t) = 0$$

```
In [12]: des=dsolve(de)               # Розв'язання рівняння Ейлера
In [13]: display(des)                 # Виклик на екран розв'язку рівняння Ейлера
```

$$y(t) = C_1 e^{-t} + C_2 e^{t} - \frac{t}{2}$$

```
In [14]: eq1=des.rhs.subs(t,0);eq1    # Формування «красивої» правої частини
Out[14]:                              ….функції y(t) у формі y(0)
```

$$C_1 + C_2$$

```
In [15]: eq2=des.rhs.diff(t).subs(t,0);eq2  # Формування «красивої» правої частини
Out[15]:                              ….похідної y'(t) у формі y'(0)
```

$$-C_1 + C_2 - \frac{1}{2}$$

```
In [16]: seq=solve([eq1,eq2-1],C1,C2);seq  # Розв'язання системи рівнянь eq1,eq2 для
Out[16]:                              ….початкових умов y(0) = 0, y'(0) = 1
```

$$\left\{C_1: -\frac{3}{4}, C_2: \frac{3}{4}\right\}$$

```
In [17]: rez=des.rhs.subs([(C1,seq[C1]),  # Підстановка значень констант C1, C2
(C2,seq[C2])]);rez                    ….з отриманого словника в розв'язок des
Out[17]:                              ….рівняння Ейлера і його «красивий» друк
```

$$-\frac{t}{2} + \frac{3}{4}e^{t} - \frac{3}{4}e^{-t}$$

```
In [18]: F=Lambda(t,rez)              # Формування символьної ламбда-функції F
In [19]: display(Latex('$y(t)='+      # Виклик на екран розв'язку рівняння
+str(latex(F(t)))+'$'))               …. Ейлера у «красивому» форматі
```

$$y(t) = -\frac{t}{2} + \frac{3e^{t}}{4} - \frac{3e^{-t}}{4}$$

```
In [20]: import numpy as np           # Виклик ППП pitpy як np
In [21]: import matplotlib.pyplot as plt  # Виклик модуля matplotlib.pyplot як plt
In [22}: x=symbols('x')               # Оголошення символьною змінної x
In [23]: expr=-x/2+3*exp(x)/4-        # Формування розв'язку рівняння Ейлера
-3*exp(-x)/4                          ….у вигляді, придатному для трансформації
In [24]: f=lambdify(x,expr,"numpy")   # Трансформація розв'язку Ейлера в pitpy
In [25]: x=np.linspace(0,1,21)        # Формування масиву значень x в pitpy
In [26]: f=f(x)                       # Формування масиву значень f(x) в pitpy
In [27]: fig=plt.figure(facecolor='white')  # Формування поля рисунка для графіка
In [28]: plt.plot(x,f(x),'-r',linewidth=3)  # Побудова графіка екстремалі Ейлера
Out[28]: [<matplotlib.lines.Line2D    …. у вигляді червоної суцільної лінії
at 0x1d0a7047970>]
```

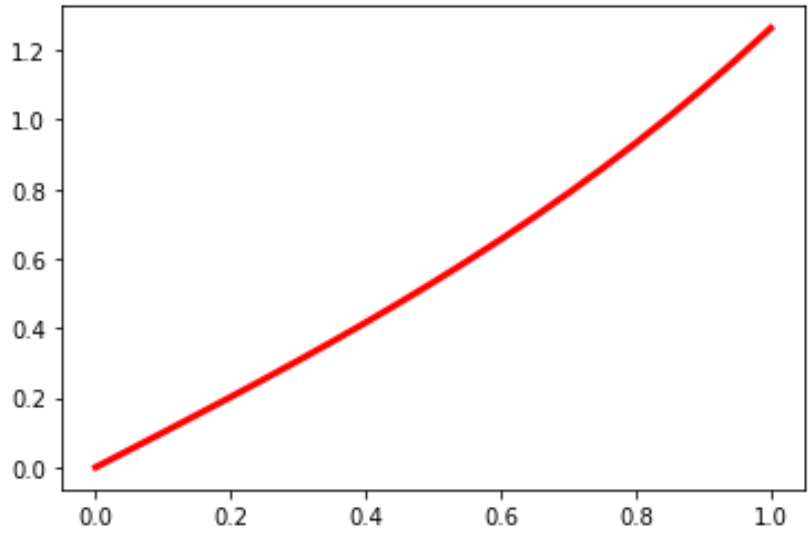

Рисунок 13 – Графік екстремалі функціонала $J_1 = \int_a^b F_1(t, y, y')dt$ у випадку, коли $a = 0,\ b = 1,\ F_1(t, y, y') = t^2 + y^2 + ty + (y')^2$, а екстремаль $y(t)$ починається в точці $(y(0) = 0,\ \ y'(0) = 1)$

Кінець програми 14.

Програма мовою Python для дослідження на безумовний екстремум функціонала

$$\boldsymbol{J_1 = \int_a^b F_1(t, y, y', y'')dt}$$

у випадку, коли $a = 0,\ b = 1,\ F_1(t, y, y', y'') = t^2 + y^2 + (y')^2 + (y'')^2 + ty + ty' + ty'' + y'y''$, а екстремаль $y(t)$ починається в точці $(y(0) = 0,\ \ y'(0) = 1,\ \ y''(0) = 0,\ \ y'''(0) = -1)$

(Програма 15)

```
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)   # Виклик функції «красивого» друку
In [5]: t=symbols('t')                  # Оголошення символьною змінної t
In [6]: y=Function('y')(t)              # Оголошення символьною функції y(t)
In [7]: z=Function('z')(t)              # Оголошення символьною функції z(t)
In [8]: w=Function('w')(t)              # Оголошення символьною функції w(t)
In [9]: z=y.diff(t)                     # Оголошення  функції z(t) похідною від y(t)
In [10]: w=z.diff(t)                    # Оголошення  функції w(t) похідною від z(t)
In [11]: u=Function('u')(t)             # Оголошення символьним виразу u(t)
In [12]: u=t**2+y**2+z**2+w**2+         # Формування ядра функціонала як u(t)
         +t*y+t*z+t*w+z*w
In [13]: de1=Eq(u.diff(y)-u.diff(z,t)+  # Формування рівняння Ейлера для u(t)
         +u.diff(w,1,t,2),0)
In [14}: display(de1)                   # Виклик на екран рівняння Ейлера
```

$$t + 2y(t) - 2\frac{d^2}{dt^2}y(t) + 2\frac{d^4}{dt^4}y(t) - 1 = 0$$

In [15]: des1=dsolve(de1) # ***Розв'язання*** рівняння Ейлера
In [16]: display(des1) # ***Виклик на екран розв'язку*** рівняння Ейлера

$$y(t) = -\frac{t}{2} + \left(C_1 sin\left(\frac{t}{2}\right) + C_2 cos\left(\frac{t}{2}\right)\right)e^{\frac{-\sqrt{3}t}{2}} + \left(C_3 sin\left(\frac{t}{2}\right) + C_4 cos\left(\frac{t}{2}\right)\right)e^{\frac{\sqrt{3}t}{2}} + \frac{1}{2}$$

In [17]: eq11=des1.rhs.subs(t,0);eq11 # ***Формування «красивої»*** правої частини
Out[17]: ….функції ***y(t)*** у формі ***y(0)***

$$C_2 + C_4 + \frac{1}{2}$$

In [18]: eq12=des1.rhs.diff(t).subs(t,0) # ***Формування «красивої»*** правої частини
In [19]: eq12 ….похідної $y'(t)$ у формі $y'(0)$
Out[19]:

$$\frac{1}{2}C_1 - \frac{\sqrt{3}}{2}C_2 + \frac{1}{2}C_3 + \frac{\sqrt{3}}{2}C_4 - \frac{1}{2}$$

In [20]: eq13=des1.rhs.diff(t,t).subs(t,0) # ***Формування «красивої»*** правої частини
In [21]: eq13 ….похідної $y''(t)$ у формі $y''(0)$
Out[21]:

$$-\frac{\sqrt{3}}{2}C_1 + \frac{1}{2}C_2 + \frac{\sqrt{3}}{2}C_3 + \frac{1}{2}C_4$$

In [22]: eq14=des1.rhs.diff(t,3).subs(t,0) # ***Формування «красивої»*** правої частини
In [23]: eq14 ….похідної $y'''(t)$ у формі $y'''(0)$
Out[23]:

$$C_! + C_3$$

In [24]: var('C1 C2 C3 C4') # ***Оголошення символьними*** C_1, C_2, C_3, C_4

$$(C_1\ C_2\ C_3\ C_4)$$

In [25]: seq1=solve([eq11,eq12-1,eq13,\
eq14+1],C1,C2,C3,C4) # ***Розв'язання*** системи рівнянь ***eq11,eq12,*** ….***eq13,eq14*** для початкових умов $y(0) = 0,$
In [26]: seq1 ….$y'(0) = 1,\ y''(0) = 0,\ y'''(0) = -1$
Out [26]: ….і ***візуалізація*** словника ***результату***

$$\left\{C_1 : -\frac{1}{2} - \frac{\sqrt{3}}{12}, C_2 : -\frac{1}{4} - \frac{2\sqrt{3}}{3}, C_3 : -\frac{1}{2} + \frac{\sqrt{3}}{12}, C_4 : -\frac{1}{4} + \frac{2\sqrt{3}}{3}\right\}$$

In [27]: rez1=des1.rhs.subs([(C1,
seq1[C1]),(C2,seq1[C2]),(C3,
seq1[C3]),(C4,seq1[C4])]); rez1 # ***Підстановка*** значень констант $C1, C2$,C3,C4
….з отриманого словника ***в розв'язок des***1
….рівняння Ейлера ***і його «красивий» друк***

$$-\frac{t}{2} + \left((-\frac{1}{2} - \frac{\sqrt{3}}{12})sin\left(\frac{t}{2}\right) + (-\frac{1}{4} - \frac{2\sqrt{3}}{3})cos\left(\frac{t}{2}\right)\right)e^{\frac{-\sqrt{3}t}{2}} +$$
$$+\left((-\frac{1}{2} + \frac{\sqrt{3}}{12})sin\left(\frac{t}{2}\right) + (-\frac{1}{4} + \frac{2\sqrt{3}}{3})cos\left(\frac{t}{2}\right)\right)e^{\frac{\sqrt{3}t}{2}} + \frac{1}{2}$$

In [28]: F1=Lambda(t,rez1) # ***Формування*** символьної ламбда-функції ***F1***
In [29]: display(Latex('$y(t)='+\
+str(latex(F1(t)))+'$')) # ***Виклик на екран розв'язку*** рівняння …. Ейлера у ***«красивому» форматі***

$$y(t) = -\frac{t}{2} + \left((-\frac{1}{2} - \frac{\sqrt{3}}{12}) sin\left(\frac{t}{2}\right) + (-\frac{1}{4} - \frac{2\sqrt{3}}{3}) cos\left(\frac{t}{2}\right) \right) e^{\frac{-\sqrt{3}t}{2}} +$$
$$+ \left((-\frac{1}{2} + \frac{\sqrt{3}}{12}) sin\left(\frac{t}{2}\right) + (-\frac{1}{4} + \frac{2\sqrt{3}}{3}) cos\left(\frac{t}{2}\right) \right) e^{\frac{\sqrt{3}t}{2}} + \frac{1}{2}$$

```
In [30]: import numpy as np                    # Виклик ППП питру як пр
In [31]: x=np.linspace(0,1,21)                # Формування масиву значень x
In [32]: def y(x):                            # Формування правої частини
             return -x/2+((-1/2-              ….розв'язку, який є екстремаллю
             -3**0.5/12)*np.sin(x/2)+         ….функціонала
         +(-1/4-2*3**0.5/3)*np.cos(x/2))*
         np.exp(-x*3**0.5/2)+
         +((-1/2+3**0.5/12)*np.sin(x/2)+
         +(-1/4+2*3**0.5/3)*
         np.cos(x/2))*np.exp(x*3**0.5/2)+
         +1/2

In [33]: fig=plt.figure(facecolor='white')    # Формування поля рисунка для графіка
In [34]: plt.plot(x,y(x),'-r',linewidth=3)    # Побудова графіка екстремалі
Out[34]: [<matplotlib.lines.Line2D
at 0x1d0a7247250>]
```

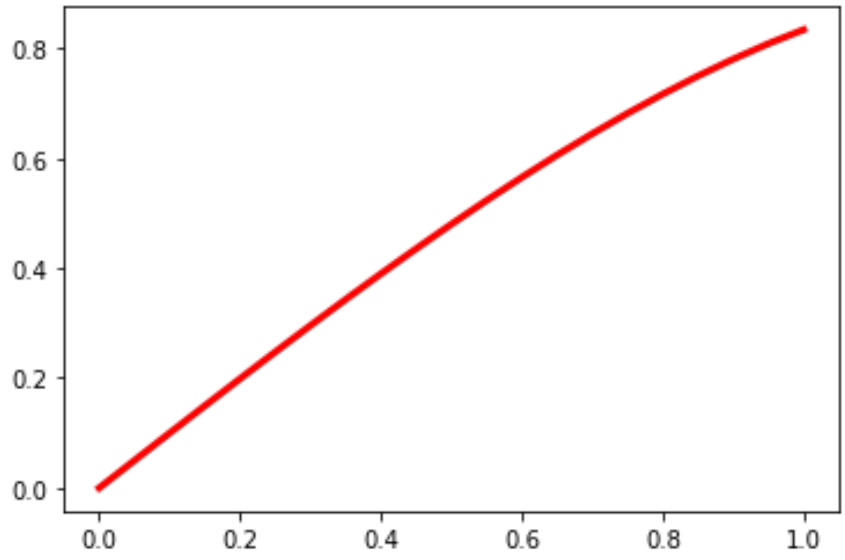

Рисунок 14 – Графік екстремалі функціонала $J_1 = \int_a^b F_1(t, y, y', y'')dt$ **у випадку, коли** $\boldsymbol{a = 0,\ b = 1,\ F_1(t, y, y', y'') = t^2 + y^2 + (y')^2 + (y'')^2 + ty + ty' + ty'' + y'y''}$**, а екстремаль** $\boldsymbol{y(t)}$ **починається в точці**$\boldsymbol{(y(0) = 0,\ y'(0) = 1,\ y''(0) = 0,\ y'''(0) = -1)}$

Кінець програми 15.

Програма мовою Python для дослідження на безумовний екстремум функціонала

$$J_1 = \int_a^b F_1(t, y_1, y_2, y_3, y_1', y_2', y_3')dt$$

у випадку, коли $a = 0$, $b = 1$, $F_1(t, y_1, y_2, y_3, y_1', y_2', y_3') = t^2 + y_1^2 + y_2^2 + y_3^2 + 3y_1y_2 + (y_1')^2 + (y_2')^2 + (y_3')^2 + 5y_2y_3$, **а екстремалі** $y_1(t), y_2(t), y_3(t)$ **починаються в точках:** $(y_1(0) = 0,\ y_1'(0) = 1), (y_2(0) = 0, y_2'(0) = 2), (y_3(0) = 0, y_3'(0) = -1)$

(Програма 16)

```
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)   # Виклик функції «красивого» друку
In [5]: t=symbols('t')                  # Оголошення символьною змінної t
In [6]: y1=Function('y1')(t)            # Оголошення символьною функції y1(t)
In [7]: y2=Function('y2')(t)            # Оголошення символьною функції y2(t)
In [8]: y3=Function('y3')(t)            # Оголошення символьною функції y3(t)
In [9]: z1=Function('z1')(t)            # Оголошення символьною функції z1(t)
In [10]: z1=y1.diff(t)                  # Оголошення  функції z1(t) похідною від y1(t)
In [11]: z2=Function('z2')(t)           # Оголошення символьною функції z2(t)
In [12]: z2=y2.diff(t)                  # Оголошення  функції z2(t) похідною від y2(t)
In [13]: z3=Function('z3')(t)           # Оголошення символьною функції z3(t)
In [14]: z3=y3.diff(t)                  # Оголошення  функції z3(t) похідною від y3(t)
In [15]: u=Function('u')(t)             # Оголошення символьним виразу u(t)
In [16]: u=t**2+y1**2+y2**2+y3**2+\     # Формування ядра функціонала як u(t)
+3*y1*y2+z1**2+z2**2+z3**2+5*y2*y3
In [17]: de11=Eq(u.diff(y1)-u.diff(z1,t),0)  # Формування рівняння Ейлера для y1(t)
In [18]: de12=Eq(u.diff(y2)-u.diff(z2,t),0)  # Формування рівняння Ейлера для y2(t)
In [19]: de13=Eq(u.diff(y3)-u.diff(z3,t),0)  # Формування рівняння Ейлера для y3(t)
In [20]: display(de11,de12,de13)        # Виклик на екран системи рівнянь Ейлера
```

$$2y_1(t) + 3y_2(t) - 2\frac{d^2}{dt^2}y_1(t) = 0,$$

$$3y_1(t) + 2y_2(t) - 2\frac{d^2}{dt^2}y_2(t) + 5y_3(t) = 0,$$

$$-2\frac{d^2}{dt^2}y_3(t) + 2y_3(t) + 5y_2(t) = 0$$

```
In [21]: eq11=Eq(y1.diff(t)-z1,0)       # Трансформація системи трьох
In [22]: eq12=Eq(y2.diff(t)-z2,0)       ….диференціальних рівнянь 2-го порядку
In [23]: eq13=Eq(y3.diff(t)-z3,0)       ….de11,de12,de13 в систему шести
In [24]: eq14=Eq(y1+3*y2/2-z1.diff(t),0)  ….диференціальних рівнянь 1-го порядку
In [25]: eq15=Eq(3*y1/2+y2+5*y3/2-\     ….eq11,eq12,eq13,eq14,eq15,eq16
        -z2.diff(t),0)
In [26]: eq16=Eq(-z3.diff(t)+5*y2/2+y3,0)
In [27]: des16=dsolve(eq11,eq12,eq13,\  # Спроба розв'язання системи шести ДР
        eq14,eq15,eq16);des16           ….1-го порядку засобами ППП sympy
Out [27]: ValueError: dsolve() and      # Програмне середовище вказує, що
 classify_ode() only work with functions  ….sympy може розв'язувати лише ДР
of one variable, not True               ….відносно функцій однієї змінної
In [28]: import numpy as np             # Виклик ППП питру
```

```
In [29]: from scipy.integrate                # Виклик із модуля scipy.integrate
         import odeint                       ….функції odeint, яка розв'язує ДР
In [30]: import matplotlib.pyplot as plt     # Виклик matplotlib.pyplot як plt
In [31}: def f(y,t):                         # Формування списку правих частин
                y1,y2,y3,z1,z2,z3=y          ….системи лиференціальних рівнянь,
                return [z1,z2,z3,y1+3*y2/2,  ….трансформованих до 1-го порядку
                        3*y1/2+y2+5*y3/2,    ….у вигляді функції, змінними в якій є
                        5*y2/2+y3]           ….змінні функціонала, що досліджується,
….                                           ….та їх похідні
In [32]:y0=[0,0,0,1,2,-1]                    # Внесення початкових умов
In [33]: t=np.linspace(0,1,21)               # Формування масиву значень змінної t
In [34]: [y1,y2,y3,z1,z2,z3]=odeint(f,y0,t,\ # Числове розв'язання системи ДР
         full_output=False).T                ….з транспонуванням стовпців у рядки
In [35]: fig=plt.figure(facecolor='white')   # Формування поля рисунка
In [37]: plt.plot(t,y1,'-r',t,y2,'-g',t,y3,'-c',  # Побудова графіків розв'язків ДР,
         linewidth=3)                        ….які є екстремалями функціонала
```

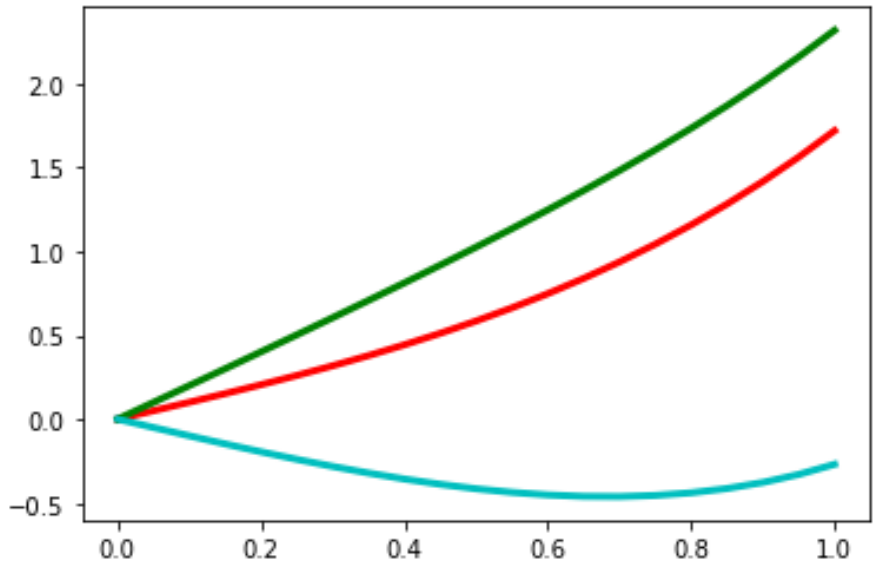

Рисунок 15 – Графік екстремалей функціонала $J_1 = \int_a^b F_1(t, y_1, y_2, y_3, y_1', y_2', y_3')dt$ у випадку, коли $a = 0$, $b = 1$, $F_1(t, y_1, y_2, y_3, y_1', y_2', y_3') = t^2 + y_1^2 + y_2^2 + y_3^2 + 3y_1y_2 + (y_1')^2 + (y_2')^2 + (y_3')^2 + 5y_2y_3$, а екстремалі $y_1(t), y_2(t), y_3(t)$ починаються в точках: $(y_1(0) = 0,\ y_1'(0) = 1), (y_2(0) = 0, y_2'(0) = 2), (y_3(0) = 0, y_3'(0) = -1)$

Кінець програми 16.

Розділ 5 ВАРІАЦІЙНІ МЕТОДИ ДОСЛІДЖЕННЯ ФУНКЦІОНАЛІВ НА УМОВНИЙ ЕКСТРЕМУМ (В ПРИКЛАДАХ І ПРОГРАМАХ)

5.1 Метод невизначених множників Лагранжа та його ізоперіметрична інтерпретація

У кінці однойменного підрозділу у базовому навчальному посібнику [1] серед інших стосовно варіаційних методів дослідження функціоналів на умовний екстремум сформульовані і такі запитання:

1. В чому відмінність пошуку умовного екстремуму від безумовного?
2. Сформулюйте задачу Лагранжа і алгоритм її розв'язання з використанням невизначених множників.
3. Запишіть алгоритм Лагранжа для функціонала, що залежить від кількох функцій та їх похідних, в умовах обмежень, визначених одним рівнянням, а також системою рівнянь.
4. Як формулюється ізоперіметрична задача оптимізації і чому вона називається ізоперіметричною?
5. Побудуйте алгоритм розв'язання ізоперіметричної задачі оптимізації.
6. Як відрізнити мінімум функціонала від максимуму в задачі на умовний екстремум?
7. Що собою являє екстремаль функціонала з обмеженнями у вигляді нерівностей?
8. Як скласти рівняння для визначення точок припасування функцій і сталих інтегрування в задачі пошуку екстремуму функціонала з обмеженнями у вигляді нерівностей? Скільки потрібно таких рівнянь і чому?

Тож із відповідей на ці запитання ми і розпочнемо викладення змісту другої частини нашого «Навчального посібника для опанування студентами способів розв'язання задач з функціонального аналізу мовою Python».

Що стосується першого питання, то його розкриття почнемо з повторення означень з цього питання, які дані в роботі [1].

Нехай дано функціонал

$$J = \int_a^b F(x, y, z, y', z')dx \tag{5.1}$$

і необхідно знайти такі функції

$$y(x), z(x), \tag{5.2}$$

які доставляють екстремум функціоналу (5.1) за умови, що

$$\varphi(x, y, z) = 0, \tag{5.3}$$

тобто за умови, що всі точки кривої (5.2) лежать на поверхні (5.3).

Ці необхідні умови існування екстремуму функціонала (5.1) за наявності обмеження (5.3) визначив Лагранж, який синтезував рівняння

$$\begin{cases} F_y - \frac{d}{dx}F_{y'} + \lambda\varphi_y = 0, \\ F_z - \frac{d}{dx}F_{z'} + \lambda\varphi_z = 0, \end{cases} \tag{5.4}$$

і показав, що для того, щоб функції (5.2) доставляли екстремум функціоналу (5.1) за наявності обмежень (5.3), необхідно, щоб вони були розв'язком системи рівнянь (5.4), в яких параметр λ Лагранж, який ввів його вперше, назвав невизначеним множником, а тому у варіаційне числення, яке є невід'ємною частиною функціонального аналізу, метод пошуку екстремалей, на яких функціонали набувають умовного екстремуму, ввійшов під назвою методу

невизначених множників Лагранжа. Але два рівняння системи (5.4) не дозволяють однозначно знайти дві невідомі функції та невідомий множник

$$y(x),\ z\ (x),\lambda. \tag{5.5}$$

а тому їх потрібно доповнити третім рівнянням, за яке взяти рівняння обмеження (5.3). І тоді система із трьох рівнянь (5.3),(5.4) відносно двох невідомих функцій та невідомого множника (5.5) матиме однозначний розв'язок, який і буде розв'язком задачі пошуку умовного екстремуму функціонала (5.1).

Узагальнюючи застосований підхід, Лагранж запропонував для задач з пошуку екстремуму функціоналів за наявності обмежень, тобто задач на умовний екстремум функціонала (5.1), конструювати функцію

$$L(x, y, z, y', z') = F(x, y, z, y', z') + \lambda\varphi(x, y, z), \tag{5.6}$$

названу послідовниками Лагранжа функцією його імені, для якої в точці екстремуму (x_0, y_0, z_0) рівняння (5.3) перетворюється у тотожність, а тому

$$L(x_0, y_0, z_0, y_0', z_0') = F(x_0, y_0, z_0, y_0', z_0'), \tag{5.7}$$

що дозволяє задачу на умовний екстремум функціоналу (5.1) з обмеженнями (5.3) звести до задачі на безумовний екстремум функціонала

$$J^L = \int_a^b L(x, y, z, y', z')dx, \tag{5.8}$$

екстремалі якого знаходяться з рівнянь Ейлера, записаних відносно функції Лагранжа, тобто у вигляді:

$$\begin{cases} L_y - \frac{d}{dx}L_{y'} = 0, \\ L_z - \frac{d}{dx}L_{z'} = 0, \end{cases} \tag{5.9}$$

доповнених третім рівнянням у вигляді (5.3). Легко бачити, що, підставляючи у рівняння (5.9) функцію Лагранжа (5.6), ми отримаємо рівняння (5.4), з яких знаходяться екстремалі функціонала (5.1) за наявності обмеження (5.3), що є підтвердженням правомірності Лагранжевої трансформації задачі на умовний екстремум функціонала за наявності обмежень в задачу на безумовний екстремум цього ж функціонала, але відносно його функції Лагранжа, в структуру якої внесені і обмеження.

Узагальнюючи викладене вище, можна стверджувати, що в разі, якщо функціонал має вигляд

$$J^F = \int_a^b F(x, y_1, y_2, \dots, y_n, y_1', y_2', \dots, y_n')dx, \tag{5.10}$$

то за наявності обмеження як у вигляді

$$\varphi(x, y_1, y_2, \dots, y_n) = 0, \tag{5.11}$$

так і у вигляді

$$\varphi_j(x, y_1, y_2, \dots, y_n) = C_j, \quad j \in [1, m],\ m < n, \quad C_j = const_j, \tag{5.12}$$

його екстремалі можна знайти, трансформуючи його у функціонал класу (5.8), у якому функція Лагранжа при обмеженнях (5.11) матиме вигляд

$$L(x, y_1, y_2, \dots, y_1', y_2', \dots, y_n') = F(x, y_1, y_2, \dots, y_n, y_1', y_2', \dots, y_n') + \lambda\ \varphi(x, y_1, y_2, \dots, y_n), \tag{5.13}$$

а при обмеженнях (5.12) матиме вигляд

$$L(x, y_1, y_2, \dots, y_1', y_2', \dots, y_n') = F(x, y_1, y_2, \dots, y_n, y_1', y_2', \dots, y_n') + \sum_{j=1}^{m} \lambda_j\ \varphi_j(x, y_1, y_2, \dots, y_n). \tag{5.14}$$

При цьому і для обмежень (5.11,) і для обмежень (5.12) система рівнянь Ейлера-Лагранжа матиме вигляд

$$\begin{cases} L_{y_1} - \frac{d}{dx} L_{y_1'} = 0, \\ L_{y_2} - \frac{d}{dx} L_{y_2'} = 0, \\ \dots\dots\dots\dots\dots\dots\dots, \\ L_{y_n} - \frac{d}{dx} L_{y_n'} = 0, \end{cases} \tag{5.15}$$

але для отримання шляхом її розв'язання ***n*** екстремалей

$$y_1(x), y_2(x), \dots, y_n(x) \tag{5.16}$$

функціонала (5.10) в разі обмежень (5.11) необхідно цю систему рівнянь (5.15) розв'язувати сумісно з рівнянням (5.11), оскільки окрім ***n*** екстремалей (5.16) визначати потрібно і невизначений множник Лагранжа λ, а для отримання екстремалей (5.16) цього функціонала (5.10) в разі обмежень (5.12), необхідно систему рівнянь (5.15) розв'язувати сумісно з системою рівнянь (5.12), оскільки у цьому випадку окрім ***n*** екстремалей (5.16) визначати потрібно і ***m*** невизначених множників Лагранжа λ_j, $j = [1, m]$. А тому за наявності обмеження у вигляді (5.11) система (5.15) набуває вигляду

$$\begin{cases} L_{y_1} + \lambda\, \varphi_{y_1}(x, y_1, y_2, \dots, y_n) - \frac{d}{dx} L_{y_1'} = 0, \\ L_{y_2} + \lambda\, \varphi_{y_2}(x, y_1, y_2, \dots, y_n) - \frac{d}{dx} L_{y_2'} = 0, \\ \dots\dots\dots\dots\dots\dots\dots, \\ L_{y_n} + \lambda\, \varphi_{y_n}(x, y_1, y_2, \dots, y_n) - \frac{d}{dx} L_{y_n'} = 0, \end{cases} \tag{5.17}$$

а за наявності обмежень у вигляді (5.12) система (5.15) набуває вигляду

$$\begin{cases} L_{y_1} + \lambda_1\, \varphi_{y_1}(x, y_1, y_2, \dots, y_n) - \frac{d}{dx} L_{y_1'} = 0, \\ L_{y_2} + \lambda_2\, \varphi_{y_2}(x, y_1, y_2, \dots, y_n) - \frac{d}{dx} L_{y_2'} = 0, \\ \dots\dots\dots\dots\dots\dots\dots \\ L_{y_m} + \lambda_m\, \varphi_{y_m}(x, y_1, y_2, \dots, y_n) - \frac{d}{dx} L_{y_m'} = 0 \\ \dots\dots\dots\dots\dots\dots\dots, \\ L_{y_{m+1}} - \frac{d}{dx} L_{y_{m+1}'} = 0, \\ \dots\dots\dots\dots\dots\dots\dots, \\ L_{y_n} - \frac{d}{dx} L_{y_n'} = 0 \end{cases} \tag{5.18}$$

В разі ж якщо задача пошуку екстремалей функціонала належить до класу ізоперіметричних, тобто таких, в яких пошук екстремалі функціонала

$$J^F = \int_a^b F(x, y, y')dx \tag{5.19}$$

здійснюється в умовах дії обмеження теж у вигляді функціонала

$$J_0^K = \int_a^b K(x, y, y')dx, \tag{5.20}$$

яка геометрично інтерпретується як пошук найбільшої площі плоскої фігури, обмеженої кривою заданої довжини, то функцію Лагранжа потрібно задавати у вигляді

$$L(x, y, \psi, \psi') = F(x, y, y') + \lambda\, (\psi'(x) - K(x, y, y')), \tag{5.21}$$

де

$$\psi(x) = \int_a^x K(x, y, y')dx, \tag{5.22}$$

для якої система рівнянь Ейлера-Лагранжа (5.15) набуває вигляду

$$\begin{cases} L_y - \frac{d}{dx}L_y = 0, \\ L_\psi - \frac{d}{dx}L_{\psi'} = 0, \end{cases} \tag{5.23}$$

а її загальний розв'язок набуває вигляду

$$y(x, C_1, C_2, C_3), \tag{5.24}$$

три сталі інтегрування C_1, C_2, C_3 в якому знаходять з рівнянь граничних умов

$$\begin{cases} y(a, C_1, C_2, C_3) = y_a, \\ y(b, C_1, C_2, C_3) = y_b \end{cases} \tag{5.25}$$

та сумісно розгянутого разом з ними рівняння

$$Q(C_1, C_2, C_3) = J_0^K, \tag{5.26}$$

яке отримуємо після підстановки загального розв'язку (5.24) у функціонал обмеження (5.20).

Як ми уже звертали увагу в роботах [1], [2] завершувати розв'язання задачі з дослідження функціонала на екстремум потрібно перевіркою умов Лежандра, згідно з якими визначається, чого ж ми досягли на знайденій його екстремалі – максимуму цього функціонала чи мінімуму.

І оскільки синтезом функції Лагранжа для функціонала, який досліджується на екстремум за наявності обмежень, та рівнянь Ейлера–Лагранжа для нього, ми трансформуємо задачу дослідження цього функціонала на умовний екстремум в задачу його дослідження на безумовний екстремум, то наведені в нашій першій частині цього навчального посібника умови Лежандра будуть справедливими і у цьому випадку після заміни в них підінтегральної функції на функцію Лагранжа, тобто для системи екстремалей (5.16) ***для мінімуму*** умови Лежандра матимуть вигляд

$$\begin{cases} L_{y'_1 y'_1} > 0, \\ \begin{vmatrix} L_{y'_1 y'_1} & L_{y'_1 y'_2} \\ L_{y'_2 y'_1} & L_{y'_2 y'_2} \end{vmatrix} > 0, \\ \dots\dots\dots\dots\dots\dots\dots\dots, \\ \begin{vmatrix} L_{y'_1 y'_1} & L_{y'_1 y'_2} & \dots & L_{y'_1 y'_n} \\ L_{y'_2 y'_1} & L_{y'_2 y'_2} & \dots & L_{y'_2 y'_n} \\ \dots & \dots & \dots & \dots \\ L_{y'_n y'_1} & L_{y'_n y'_2} & \dots & L_{y'_n y'_n} \end{vmatrix} > 0 \end{cases} \tag{5.27}$$

А ***достатні умови існування максимуму*** функціонала (5.10) на множині екстремалей (5.16***) за Лежандром*** будуть мати ***вигляд, аналогічний (5.27), але знаки*** нерівностей в них будуть ***протилежними***.

Що ж до функціонала (5.19) в ізопериметричній задачі, то з викладених вище причин для досягнення ним на екстремалі (5.24) мінімуму, необхідно, щоб для всіх $x \in [a, b]$ виконувалась нерівність

$$F_{y'y'} > 0, \tag{5.28}$$

а для досягнення ***максимуму,*** необхідно, щоб для всіх $x \in [a, b]$ виконувалась нерівність

$$F_{y'y'} < 0. \tag{5.29}$$

5.2 Додаткові відомості з мови програмування Python, достатні для розв'язання задач, пов'язаних з застосуванням варіаційних методів

Аналізуючи характер формул, викладених у попередньому підрозділі 5.1, бачимо, що для їх реалізації в програмах мовою Python нам уже не потрібні нові знання функцій цієї мови програмування, додаткові до тих, про які уже йшла мова в підрозділах, присвячених викладенню додаткових відомостей з технології програмування мовою Python у першій частині цього навчального посібника, оскільки після синтезу функції Лагранжа для функціонала, який досліджується на екстремум за наявності обмежень, задача на умовний екстремум трансформується в задачу на безумовний екстремум, для усіх варіантів якої нами уже складені програми мовою Python у першій частині цього навчального посібника. Тому у цьому підрозділі для поновлення в оперативній пам'яті нашого головного мозку ми лише нагадаємо ті основні функції технології програмування мовою Python, які уже фігурували в розділі, присвяченому дослідженню функціоналів на безумовний екстремум у першій частині цього навчального посібника, не вдаючись до їх деталізації та не формуючи для їх роз'яснення нових пояснювальних прикладів, адже за необхідності для поновлення в своїй пам'яті цієї деталізації та роз'яснення її суті за допомогою пояснювальних прикладів кожен бажаючий може звернутись до четвертого розділу першої частини цього навчального посібника.

Отже, ***реалізацію*** програмних функцій ***диференціювання*** нам потрібно ***починати*** з ***виклику ППП sympy***. ***Однократне диференціювання*** за однією незалежною змінною ***здійснюється*** програмною функцією ***diff(,)***, в аргументних дужках якої першим стоїть вираз, що диференціюється, а другим змінна, за якою здійснюється диференціювання, а ***n-кратне диференціювання*** здійснюється ***цією*** ж програмною ***функцією***, але в аргументних дужках додається ***третім аргументом число***, яким конкретизується кількість ***диференціювань.*** При визначенні ***змішаних частинних похідних*** від виразу, що містить ***дві незалежні змінні***, ***в*** аргументних ***дужках*** цієї ж програмної ***функції*** окрім виразу, що диференціюється, ***на другому і третьому місцях записуються обидві незалежні змінні***, за якими здійснюється диференціювання, а якщо необхідно визначити ***змішану частинну похідну з порядком, вищим першого***, програмна ***функція*** уже матиме ***5 аргументів***, першим із яких виступатиме вираз, що диференціюється, ***другим – перша*** незалежна ***змінна***, ***третім – число*** диференціювань ***за першою*** змінною, ***четвертим – друга*** незалежна ***змінна***, а ***п'ятим – число*** диференціювань ***за другою*** змінною. В разі, ***якщо*** програмна функція ***diff(,)*** виявляється ***неспроможною*** здійснити диференціювання, потрібно застосовувати ***метод диференціювання***, який має ту ж символіку, але запис методу має вигляд ***f.diff(,),*** а в аргументних дужках вписується ***на першому місці*** незалежна ***змінна*** за якою 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 за нею***. ***Якщо ж 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( ).*** Дослідження функціоналів на безумовний екстремум з використанням функції Лагранжа передбачають етап визначення екстремалей шляхом розв'язання рівнянь Ейлера–Лагранжа, які відносять до класу нелінійних диференціальних рівнянь***. В*** програмних ***пакетах sympy та scipy є кілька програмних функцій***, які розв'язують диференціальні рівняння та системи диференціальних рівнянь. В пакеті ***sympy –*** це програмна функція ***dsolve( )***, яка використовується для символьного розв`язання звичайних диференціальних рівнянь, а в пакеті ***scipy –*** це програмні функції ***ode( )*** та ***odeint( )***, які входять до структури модуля ***scipy.integrate.*** Оскільки програмні функції пакета ***sympy*** виводять результат розв'язання диференціальних рівнянь у вигляді аналітичних виразів, то, на перший погляд, їх використання є пріоритетним. Але, оскільки цим програмним функціям під силу не кожне

диференціальне рівняння, то у більшості практичних задач, в яких потрібно розв'язувати диференціальні рівняння, доцільніше застосовувати для їх розв'язання програмні функції пакета ***scipy***, за допомогою яких розв'язок отримується в числовій формі, але при цьому в обов'язковому порядку доведеться викликати і пакет ***numpy,*** оскільки при числових розрахунках ***необхідно задавати масиви*** значень незалежної змінної, ***які є об'єктами*** саме цього пакета ***мовою Python.*** Як правило, ***розв'язки*** диференціальних рівнянь ***характеризують розвиток у часі процесів***, які ними описуються, тож корисним є графічне відображення цих розв'язків – і у цьому випадку не обійтись без виклику ще й пакета ***matplotlib*** для реалізації ***підпрограми побудови графіків*** цих розв'язків. ***Звертаємо увагу*** на те, що ***при символьному розв'язанні*** диференціальних рівнянь програмними функціями пакета ***sympy*** оголошувати ***символьними*** потрібно не лише імена незалежних змінних, ***але і 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. Що ж до застосовування для розв'язання диференціальних рівнянь програмних функцій пакета ***scipy***, за допомогою яких розв'язок отримується в числовій формі, то, 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 приведення 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 є функцією правих частин системи ***n диференціальних рівнянь 1-го порядку func(y,t)***, де $y = [y_1, y_2, \dots, y_n]$, другий аргумент – список початкових значень $y(t_0) = [y_1(t_0), y_2(t_0), \dots, y_n(t_0)]$, а третій аргумент – це масив значень незалежної змінної t, в які ми хочемо визначити значення розв'язку $y(t)$. Що ж стосується програмної функції ***ode( )***, то вона розв'язує і ті диференціальні рівняння, з якими не може впоратись більш проста за інтерфейсом програмна функція ***odeint(, ,).***

5.3 Задачі дослідження функціоналів на умовний екстремум в програмах мовою Python

Програма мовою Python для дослідження на умовний екстремум функціонала

$$J_1 = \int_a^b F_1(t, y, y')dt$$

у випадку, коли $a = 0$, $b = 1$, $F_1(t, y, y') = y^2 + yy' + (y')^2$, а екстремаль $y(t)$ починається в точці($y(0) = 0$, $y'(0) = 1$) та задовольняє обмеження $y = 1$

(Програма 17)

```
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)   # Виклик функції «красивого» друку
In [5]: t=symbols('t')                  # Оголошення символьною змінної t
In [6]: λ =symbols ('λ')                # Оголошення символьним множника λ
```

In [7]: y=Function('y')(t) # ***Оголошення символьною*** функції ***y(t)***
In [8]: z=Function('z')(t) # ***Оголошення символьною*** функції ***z(t)***
In [9]: z=y.diff(t) # ***Оголошення*** функції ***z(t)*** похідною від ***y(t)***
In [10]: u=Function('u')(t) # ***Оголошення символьним*** виразу ***u(t)***
In [11]: u=y**2+y*z+z**2+$\lambda(y-1)$ # ***Формування*** ядра функціонала як ***u(t)***
In [12]: de=Eq(u.diff(y)-u.diff(z,t),0) # ***Формування*** рівняння Ейлера–Лагранжа для ***u(t)***
In [13]: display(de) # ***Виклик на екран*** рівняння Ейлера–Лагранжа

$$\lambda + 2y(t) - 2\frac{d^2}{dt^2}y(t) = 0$$

In [14]: des=dsolve(de) # ***Розв'язання*** рівняння Ейлера
In [15]: display(des) # ***Виклик на екран розв'язку*** рівняння Ейлера

$$y(t) = C_1e^{-t} + C_2e^{t} - \frac{\lambda}{2}$$

In [16]: eq1=des.rhs.subs(t,0);eq1 # ***Формування «красивої»*** правої частини
Out[16]: ….функції ***y(t)*** у формі ***y(0)***

$$C_1 + C_2 - \frac{\lambda}{2}$$

In [17]: eq2=des.rhs.diff(t).subs(t,0);eq2 # ***Формування «красивої»*** правої частини
Out[17]: ….похідної $y'(t)$ у формі $y'(0)$

$$-C_1 + C_2$$

In [18]: seq=solve([eq1,eq2-1],C1,C2);seq # ***Розв'язання*** системи рівнянь ***eq1,eq2*** для
Out[18]: ….початкових умов $y(0) = 0,\ y'(0) = 1$

$$\left\{C_1: \frac{\lambda}{4} - \frac{1}{2}, C_2: \frac{\lambda}{4} + \frac{1}{2}\right\}$$

In [19]: rez=des.rhs.subs([(C1,seq[C1]), # ***Підстановка*** значень констант $C1, C2$
(C2,seq[C2])]);rez ….з отриманого словника ***в розв'язок des***
Out[19]: ….рівняння Ейлера ***і його «красивий» друк***

$$-\frac{\lambda}{2} + \left(\frac{\lambda}{4} - \frac{1}{2}\right)e^{-t} + (\frac{\lambda}{4} + \frac{1}{2})e^{t}$$

In [20]: eq3=rez.subs(t,1);eq3 # ***Формування «красивої»*** правої частини
Out[20]: ….функції ***y(t)*** у формі ***y(1)***

$$-\frac{\lambda}{2} + \left(\frac{\lambda}{4} - \frac{1}{2}\right)e^{-1} + (\frac{\lambda}{4} + \frac{1}{2})e^{1}$$

In [21]: expr1=-λ/2+(λ/4-1/2)/exp(1)+ # ***Формування*** лівої частини рівняння ***y(1)=1***
+(λ/4+1/2)* exp(1)
In [22]: expr2=1 # ***Формування*** правої частини рівняння ***y(1)=1***
In [23]: solveset(Eq(expr1,expr2),λ) # ***Розв'язання*** рівняння ***y(1)=1*** відносно λ
Out [23]:

$$\frac{-4.0(-1.0e - 0.5 + 0.5e^2)}{(-1.0 + 1.0e)^2}$$

In [24]: import numpy as np # ***Виклик*** ППП ***питру*** як ***пр***
In [25]: λ=-4*(-np.exp(1)-0.5+ # ***Формування*** виразу та обчислення
+0.5*(np.exp(1))**2)/(-1+\ ….значення множника λ
np.exp(1))**2;λ ….в ППП ***питру***
Out [25]:

-0,6423909789760494

```
In [26]: λ=np.array(-0.6423909789760494) # Трансформація числа в масив
In [27]: λ=λ.round(3);λ                 # Скорочення кількості знаків після коми
Out[27]: -0.642                         ….до трьох
In [28]: d={}                           # Формування  пустого словника d
In [29]: d['λ']=-0.642                  # Внесення в словник значення множника λ
In [30]: d                              # Виклик створеного словника
Out [30]:
```

$$\{'\lambda': -0.642\}$$

```
In [31]: y=y.subs({'λ':-0.642});y       # Підстановка значення множника λ в
                                        ….розв'язок y(t)
Out [31]:
```

$$0.321 - 0.6605e^{-t} + 0.3395e^{t}$$

```
In [32]: F=Lambda(t,y)                  # Формування символьної ламбда-функції F
In [33]: display(Latex('$y(t)='+\       # Виклик на екран розв'язку рівняння Ейлера–
         +str(latex(F(t)))+'$'))        …. Лагранжа у «красивому» форматі
```

$$y(t) = 0.321 - 0.6605e^{-t} + 0.3395e^{t}$$

```
In [34]: import matplotlib              # Виклик ППП matplotlib
In [35]: import matplotlib.pyplot as plt # Виклик модуля matplotlib.pyplot як plt
In [36]: x=symbols('x')                 # Оголошення символьною змінної x
In [37]: expr=0.321-0.6605*exp(-x) +\   # Формування розв'язку рівняння Ейлера–
         +0.3395*exp(x)                 ….Лагранжа, придатного для трансформації
In [38]: f=lambdify(x, expr,'numpy')    # Трансформація розв'язку в питру
In [39]: x=np.linspace(0,1,21)          # Формування масиву значень x  в питру
In [40]: fig= plt.figure (facecolor='white') # Формування поля рисунка для графіка
In [41]: plt.plot(x,f(x),'-r',linewidth=3)   # Побудова графіка екстремалі
```

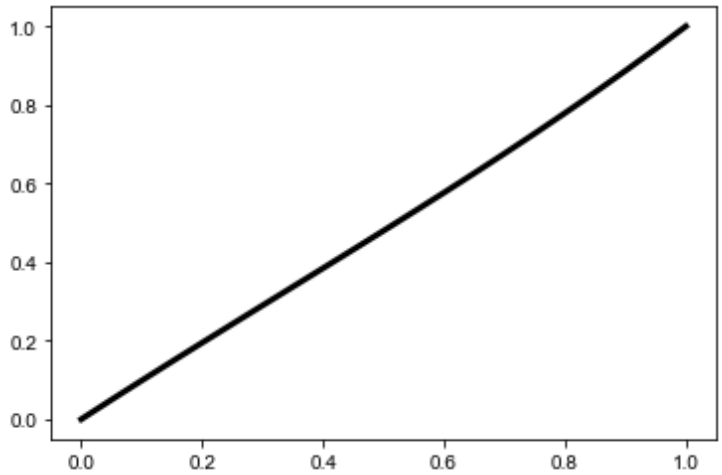

Рисунок 16 – Графік екстремалі функціонала $J_1 = \int_a^b F_1(t, y, y')dt$ у випадку, коли $a = 0,\ b = 1,\ F_1(t, y, y') = y^2 + yy' + (y')^2$, а екстремаль $y(t)$ починається в точці $(y(0) = 0,\ \ y'(0) = 1)$ та задовольняє обмеження $y = 1$

Кінець програми 17.

Примітка. Уважно проаналізувавши усі співвідношення, наведені в попередньому підрозділі, бачимо, що функція Лагранжа для функціонала з кількома змінними та кількома обмеженнями буде відрізнятись від функції Лагранжа для функціонала з однією змінною і одним обмеженням лише кількістю складових, які, якщо домножити кожну на свій невизначений множник Лагранжа, роблять функцію Лагранжа залежною одночасно від усіх цих обмежень та усіх невизначених множників Лагранжа біля кожного з цих обмежень. А тому програма мовою Python для дослідження функціонала, що залежить від кількох змінних та кількох обмежень, на умовний екстремум буде відрізнятись від наведеної вище програми для дослідження функціонала, що залежить від однієї змінної лише тим, що доведеться розв'язувати не одне алгебраїчне рівняння, складене відносно одного множника Лагранжа шляхом підстановки розв'язку рівняння Ейлера–Лагранжа, в якому множник Лагранжа виступає в ролі невідомого параметра, в рівняння обмеження, а доведеться розв'язувати систему алгебраїчних рівнянь, кожне з яких міститиме в собі усі невідомі множники Лагранжа після підстановки в кожне з них усіх розв'язків системи рівнянь Ейлера–Лагранжа, в кожному з яких один із множників Лагранжа виступає в ролі невідомого параметра. Але оскільки програма, що реалізуватиме це узагальнення, не несе принципових відмінностей від тієї програми, що нами наведена вище, то ми не бачимо сенсу розміщувати у цьому підрозділі ще й програму реалізації мовою Python процесу дослідження функціонала, що залежить від кількох змінних та кількох обмежень, справедливо сподіваючись, що кожен, хто працюватиме з цим навчальним посібником, з задоволенням напише цю узагальнену програму самостійно, що послужить йому критерієм того, що він гарно засвоїв викладений вище матеріал. Тож далі ми одразу перейдемо до складання програми мовою Python, яка реалізуватиме процес дослідження на умовний екстремум функціонала, для якого за обмеження виступає також функціонал, тобто складемо програму для розв'язання ізоперіметричної задачі.

Програма мовою Python для дослідження на умовний екстремум функціонала

$$J_1 = \int_a^b F_1(t, y, y')dt$$

у випадку, коли $a = 0$, $b = 1$, $F_1(t, y, y') = y + y^2 + y' + (y')^2$, **а екстремаль** $y(t)$ **починається в точці**$(y(0) = 0,\ y'(0) = 1)$ **та задовольняє обмеження** $J_2 = \int_a^b (y + y')dt$, **де** $J_2 = 0.5$

(Програма 18)

```
In [1]: import sympy                    # Виклик ППП sympy
In [2]: from sympy import*              # Виклик із ППП sympy усіх функцій
In [3]: from IPython.display import*    # Виклик із IPython.display усіх функцій
In [4]: init_printing(use_latex=True)   # Виклик функції «красивого» друку
In [5]: t=symbols('t')                  # Оголошення символьною змінної t
In [6]: y=Function('y')(t)              # Оголошення символьною функції y(t)
In [7]: z=Function('z')(t)              # Оголошення символьною функції z(t)
In [8]: z=y.diff(t)                     # Оголошення функції z(t) похідною від y(t)
In [9] y1= Function('y1')(t)            # Оголошення символьною функції y1(t)
In [10]: z1=Function('z1')(t)           # Оголошення символьною функції z1(t)
In [11]: z1=y1.diff(t)                  # Оголошення  функції z1(t) похідною від y1(t)
In [12]: C3=symbols('C3')               # Оголошення символьною константи C3
In [13]: u=Function('u')(t)             # Оголошення символьним виразу u(t)
In [14]: u=y+y**2+z+z**2+               # Формування функції Лагранжа як u(t)
+C3*(z1-y-z)
```

In [15]: de=Eq(u.diff(y)-u.diff(z,t),0) # ***Формування*** рівняння Ейлера–Лагранжа для ***u(t)***
In [16]: des=dsolve(de) # ***Розв'язання*** рівняння Ейлера
In [17]: display(des) # ***Виклик на екран розв'язку*** рівняння Ейлера

$$y(t) = C_1 e^{-t} + C_2 e^{t} + \frac{C_3}{2} - \frac{1}{2}$$

In [18]: eq1=des.rhs.subs(t,0);eq1 # ***Формування «красивої»*** правої частини
Out[18]: ….функції ***y(t)*** у формі ***y(0)***

$$C_1 + C_2 + \frac{C_3}{2} - \frac{1}{2}$$

In [18]: eq2=des.rhs.diff(t).subs(t,0);eq2 # ***Формування «красивої»*** правої частини
Out[18]: ….похідної $y'(t)$ у формі $y'(0)$

$$-C_1 + C_2$$

In [19]: var('C1 C2') # ***Оголошення символьними*** констант ***C1,C2***

Out[19]

(C1,C2)

In [20]: seq=solve([eq1,eq2-1],C1,C2);seq # ***Розв'язання*** системи рівнянь ***eq1,eq2-1*** для
Out[20]: ….початкових умов $y(0) = 0,\ y'(0) = 1$

$$\left\{C_1: -\frac{C_3}{4} - \frac{1}{4}, C_2: -\frac{C_3}{4} + \frac{3}{4}\right\}$$

In [21]: rez=des.rhs.subs([(C1,seq[C1]),\ # ***Підстановка*** значень констант $C1, C2$
(C2,seq[C2])]);rez ….із отриманого словника ***в розв'язок des***
Out[21]: ….рівняння Ейлера–Лагранжа ***і його друкування***

$$\frac{C_3}{2} + \left(-\frac{C_3}{4} - \frac{1}{4}\right)e^{-t} + \left(-\frac{C_3}{4} + \frac{3}{4}\right)e^{t} - \frac{1}{2}$$

In [22]: inte1=integrate(rez+\ # ***Обчислення інтеграла*** від функції
+rez.diff(t),(t,0,1));inte1 ….$y(t) + y'(t)$ в межах від 0 до 1
Out[22]:

$$C_3 + \frac{e(3-C_3)}{2} - 2$$

In [23]: solveset(Eq(inte1,0.5),C3) # ***Обчислення множника*** C_3 із рівняння
Out [23]: ….для обмеження

$$\frac{2.5(-1.0 + 0.6e)}{-1.0 + 0.5\ e}$$

In [24]: import numpy as np # ***Виклик*** ППП ***питру*** як ***пр***
In [25]: C3=2.5*(-1.0+\ # ***Формування*** виразу та обчислення
+0.6*(np.exp(1))/(-1.0+0.5*np.exp(1)); ….значення множника C_3 в ППП ***питру***
Out [25]:

4,39221119

In [26]: C3=np.array(4.39221119) # ***Трансформація числа в масив***
In [27]: C3=C3.round(3);C3 # ***Скорочення кількості знаків*** після коми
Out[27]: 4.392 ….до трьох
In [28]: d={} # ***Формування*** пустого словника ***d***

In [29]: d['C3']=4.392 # ***Внесення*** в словник значення множника C_3
In [30]: d # ***Виклик*** створеного словника
Out [30]:

{'C3': 4.392}

In [31]: y=rez.subs({'C3':4.392});y # ***Підстановка*** значення множника C_3 в ….розв'язок ***y(t)***
Out [31]:

$$1.696 - 1.348e^{-t} - 0.348e^{t}$$

In [32]: F=Lambda(t,y) # ***Формування*** символьної ламбда-функції ***F***
In [33]: display(Latex('$y(t)='+\ # ***Виклик на екран розв'язку*** рівняння Ейлера–
+str(latex(F(t)))+'$')) …. Лагранжа у ***«красивому» форматі***

$$y(t) = 1.696 - 1.348e^{-t} - 0.348e^{t}$$

In [34]: import matplotlib # ***Виклик*** ППП matplotlib
In [35]: import matplotlib.pyplot as plt # ***Виклик модуля*** matplotlib.pyplot як ***plt***
In [36}: x=symbols('x') # ***Оголошення символьною*** змінної ***x***
In [37]: expr=1.696-1.348*exp(-x) - \ # ***Формування розв'язку*** рівняння Ейлера–
-0.348*exp(x) ….Лагранжа, придатного для трансформації
In [38]: f=lambdify(x, expr,"numpy") # ***Трансформація*** розв'язку в ***питру***
In [39]: x=np.linspace(0,1,21) # ***Формування масиву*** значень ***x*** в ***питру***
In [40]: fig= plt.figure (facecolor='white') # ***Формування поля*** рисунка для графіка
In [41]: plt.plot(x,f(x),'-r',linewidth=3) # ***Побудова графіка*** екстремалі

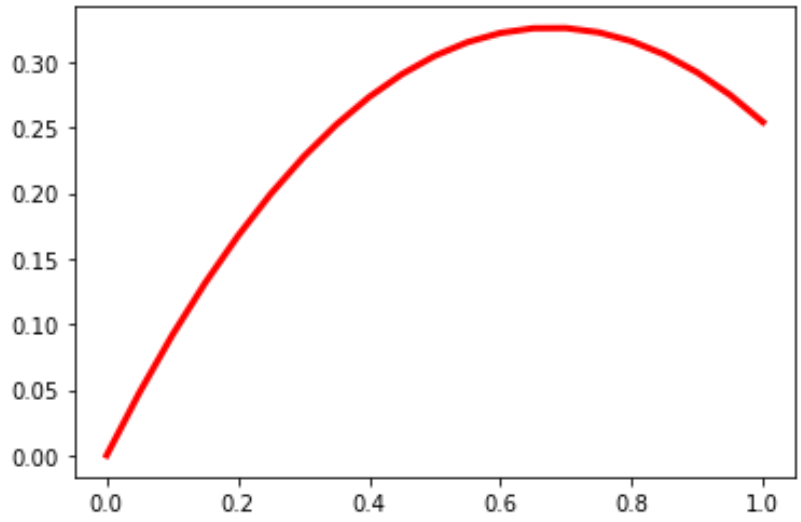

Рисунок 17 – Графік екстремалі функціонала $\boldsymbol{J_1 = \int_a^b F_1(t, y, y')dt}$ у випадку, коли $\boldsymbol{a = 0,\ b = 1,\ F_1(t, y, y') = y + y^2 + y' + (y')^2}$, а екстремаль $y(t)$ починається в точці $(y(0) = 0,\ \ y'(0) = 1)$ та задовольняє обмеження $\boldsymbol{J_2 = \int_a^b (y + y')dt}$, де $\boldsymbol{J_2 = 0.5}$

Кінець програми 18.

5.4 Особливості дослідження функціоналів на умовний екстремум в гільбертових просторах

У кінці однойменного підрозділу у базовому навчальному посібнику [1] серед інших стосовно варіаційних методів дослідження функціоналів на умовний екстремум сформульовані і такі запитання:

1. В чому суть прямих методів пошуку екстремумів функціоналів? Суть методу Рітца.
2. В якому вигляді відшуковується екстремаль функціонала в методі Рітца? Які ортонормовані послідовності використовуються в цьому методі?
3. Розкрийте суть етапів розв'язання задачі пошуку екстремуму функціонала методом Рітца.

Тож із відповідей на ці запитання ми і продовжимо викладення змісту другої частини нашого «Навчального посібника для опанування студентами способів розв'язання задач з функціонального аналізу мовою Python» в частині дослідження функціоналів на умовний екстремум в гільбертових просторах

Нехай дано функціонал

$$J_y^F = \int_a^b F(x, y, y')dx, \tag{5.30}$$

і потрібно знайти екстремаль $y(x)$, яка мінімізує цей функціонал за умови, що на границях області $[a, b]$ маємо

$$\begin{bmatrix} y(a) = y_a, \\ y(b) = y_b \end{bmatrix} \tag{5.31}$$

і мають місце обмеження вигляду

$$g(x, y, y') = 0 \tag{5.32}$$

або

$$\int_a^b K(x, y, y')dx = J_K \tag{5.33}$$

Будемо шукати екстремаль $y(x)$ у вигляді

$$y(x) = \sum_{i=1}^n C_i \varphi_i(x), \tag{5.34}$$

де $\varphi_i(x)$ – відомі ортонормовані поліноми з числа тих, що розглянуті в другому розділі, наприклад, поліноми Лагерра, Лежандра чи Чебишова, які ми свідомо вибрали та детально розглянули у підрозділі 2.3, виходячи зі зручності їх використання для розв'язання нашої задачі.

Як ми уже вказували у нашій роботі [1], вибором виразу (5.34) ми фактично зводимо задачу пошуку екстремалей функціонала (5.30) в умовах дії обмежень (5.32) або (5.33) до задачі визначення коефіцієнтів $C_i,\ i = 1,2, \dots n$ в кілька етапів.

На першому етапі розв'язання задачі, підставляючи вираз (5.34) у рівняння граничних умов (5.31), ми скорочуємо на два кількість невідомих коефіцієнтів $C_i,\ i = 1,2, \dots n$, які нам потрібно знайти, щоб однозначно визначити екстремаль (5.34), котра доставляє мінімум функціоналу (5.30). Це нам вдається зробити на першому етапі завдяки тому, що за допомогою двох рівнянь (5.31) можемо два коефіцієнти, наприклад C_1, C_2, виразити через інші коефіцієнти C_i, для яких $i = 3,4, \dots, n,$ тобто, визначимо ці коефіцієнти як алгебраїчні залежності

$$\begin{cases} C_1 = f_1(y_a, y_b, C_3, C_4, \dots, C_n), \\ C_2 = f_2(y_a, y_b, C_3, C_4, \dots, C_n) \end{cases} \tag{5.35}$$

В результаті цих дій і підстановки виразів (5.35) у вираз (5.34) майбутня екстремаль (5.34) набуде вигляду

$$y(x) = y_1(x, C_3, C_4, \dots, C_n) \tag{5.36}$$

На другому етапі розв'язання задачі, підставляючи вираз (5.36) у вираз(5.32) або (5.33), ми отримаємо рівняння, за допомогою якого можна вилучити ще один коефіцієнт C_i, наприклад C_3, виразивши його через коефіцієнти C_i, для яких $i = 4,5, \dots, n$, тобто, визначимо цей коефіцієнт як алгебраїчну залежність

$$C_3 = f_3(C_4, C_5, \dots, C_n). \tag{5.37}$$

Тож після завершення цього етапу майбутню екстремаль (5.34) уже матимемо у вигляді

$$y(x) = y_2(x, C_4, C_5, \dots, C_n)\ . \tag{5.38}$$

На третьому етапі розв'язання задачі, підставляючи вираз (5.38) у функціонал (5.30), тобто, обраховуючи функцію $F(x, y, y')$ і беручи інтеграл, ми отримаємо функцію

$$y_3(C_4, C_5, \dots, C_n), \tag{5.39}$$

яка вже не містить в собі змінної x і буде функцією виключно лише коефіцієнтів $C_4, C_5, \dots, C_n$, а тому для пошуку оптимальних значень цих коефіцієнтів потрібно використовувати стандартну методику пошуку екстремуму функції кількох змінних.

Тож на четвертому етапі розв'язання задачі згідно з цією стандартною методикою, складемо стільки алгебраїчних рівнянь відносно невідомих коефіцієнтів $C_4, C_5, \dots, C_n$, скільки їх є у виразі (5.39), для чого візьмемо частинні похідні від цього виразу за кожним із невідомих коефіцієнтів і прирівняємо кожну з цих частинних похідних до нуля. Цим ми створимо систему $n-3$ рівнянь з $n-3$ невідомими $C_4, C_5, \dots, C_n$

$$\frac{\partial y_3(*)}{\partial C_i} = 0, \quad i = 4,5, \dots, n \tag{5.40}$$

На п'ятому етапі розв'язання задачі розв'яжемо отриману систему алгебраїчних рівнянь (5.40) відносно $C_4, C_5, \dots, C_n$.

На шостому етапі розв'язання задачі знайдені числові значення коефіцієнтів $C_4, C_5, \dots, C_n$ підставляємо у вираз (5.37) і визначаємо числове значення коефіцієнта C_3, який, у свою чергу, разом із числовими значеннями коефіцієнтів $C_4, C_5, \dots, C_n$ підставляємо у вирази (5.35) та визначаємо числові значення коефіцієнтів C_1, C_2.

На сьомому етапі розв'язання задачі знайдені числові значення усіх коефіцієнтів $C_1, C_2, C_3, C_4, C_5, \dots, C_n$ підставляємо у вираз (5.34).

Це і буде екстремаль $y(x)$, яка в умовах дії обмежень (5.32) чи (5.33) та граничних умов (5.31) доставляє екстремум функціоналу (5.30).

5.5 Задачі дослідження функціоналів на умовний екстремум в гільбертових просторах в програмах мовою Python

Програма мовою Python для дослідження на умовний екстремум функціонала $J_1 = \int_a^b F_1(t, y, y')dt$ методом Рітца з використанням перших 5 ортонормованих поліномів Лежандра у випадку, коли $a = -1$, $b = 1$, $F_1(t, y, y') = y + y^2 + y' + (y')^2$, а екстремаль $y(t)$ починається в точці($y(0) = 0$, $y'(0) = 1$) та задовольняє обмеження $J_2 = \int_a^b F_2(t, y, y')dt$, де $F_2(t, y, y') = y + y'$, $J_2 = 1$

(Програма 19)

```
In [1]: import sympy                    # Виклик ППП sympy
In [2]: from sympy import*              # Виклик із ППП sympy усіх функцій
```

```
In [3]: from IPython.display import*          # Виклик із IPython.display усіх функцій
In [4]: init_printing(use_latex=True)        # Виклик функції «красивого» друку
In [5]: t=symbols ('t')                      # Оголошення символьною змінної t
In [6]: P0=Function('P0')(t)                 # Оголошення символьним полінома P0(t)
In [7]: P1=Function('P1')(t)                 # Оголошення символьним полінома P1(t)
In [8]: P2=Function('P2')(t)                 # Оголошення символьним полінома P2(t)
In [9]: P3=Function('P3')(t)                 # Оголошення символьним полінома P3(t)
In [10]: P4=Function('P4')(t)                # Оголошення символьним полінома P4(t)
In [11]: P0=1                                # Формування полінома P0(t)
In [12]: P1=t                                # Формування полінома P1(t)
In [13]: P2=(3*t**2-1)/2                     # Формування полінома P2(t)
In [14]: P3=(5*t**3-3*t)/2                   # Формування полінома P3(t)
In [15]: P4=(35*t**4-30*t**2+3)/8            # Формування полінома P4(t)
In [16]: var('C0 C1 C2 C3 C4')               # Оголошення символьними
Out[16]:                                     .... констант C0,C1,C2,C3,C4
```

(C0,C1,C2,C3,C4)

```
In [17]: y=Function('y') (t)                 # Оголошення символьною функції y(t)
In [18]: z=Function('z') (t)                 # Оголошення символьною функції z(t)
In [19]: y=C0*P0+C1*P1+C2*P2+C3*P3+C4*P4     # Формування функції y(t)
In [20]: display(y)                          # Виклик на екран функції y(t)
```

$$C_0 + C_1 t + C_2\left(\frac{3t^2}{2} - \frac{1}{2}\right) + C_3\left(\frac{5t^3}{2} - \frac{3t}{2}\right) + C_4\left(\frac{35t^4}{8} - \frac{15t^2}{4} + \frac{3}{8}\right)$$

```
In [21]: z=y.diff(t)                         # Формування похідної від функції y(t)
In [22]: display(z)                          # Виклик на екран похідної z(t)
```

$$C_1 + 3C_2 t + C_3\left(\frac{15t^2}{2} - \frac{3}{2}\right) + C_4\left(\frac{35t^3}{2} - \frac{15t}{2}\right)$$

```
In [23]: eq1=y.subs(t,0);eq1                 # Формування правої частини
Out[23]:                                     ….функції y(t) у формі y(0)
```

$$C_0 - \frac{C_2}{2} + 3C_4/8$$

```
In [24]: eq2=z.subs(t,0);eq2                 # Формування правої частини
Out[24]:                                     ….похідної y′(t) у формі y′(0)
```

$$C_1 - 3C_3/2$$

```
In [25]: seq=solve ([eq1,eq2-1],C0,C1);seq   # Розв'язання системи рівнянь eq1,eq2-1 для
Out[25]:                                     ….початкових умов y(0) = 0,  y′(0) = 1
```

$$\left\{C_0 : \frac{C_2}{2} - \frac{3C_4}{8}, C_1 : \frac{3C_3}{2} + 1\right\}$$

```
In [26]: rez=y.subs([(C0,seq[C0]),(C1,seq[C1])])   # Підстановка значень C0, C1 у функцію y(t)
In [27]: rez1=z.subs([(C0,seq[C0]),(C1,seq[C1])])  # Підстановка значень C0, C1 в похідну y′(t
In [28]: u=Function('u')(t)                  # Оголошення символьним виразу u(t)
In [29]: u=rez+rez1                          # Формування виразу F2(t, y, y′)
In [30]: eq3=integrate(u,(t,-1,1));eq3       # Інтегрування виразу F2(t, y, y′)
```

Out[30]:

$$C_2 + 5C_3 - \frac{3C_4}{4} + 2$$

In [31]: seq1=solve([eq3-1],C2);seq1 # ***Розв'язання*** відносно C_2 рівняння
Out[31]: ….обмеження, яке отримуємо після

$$\{C2\text{:-}5C_3 + \frac{3C_4}{4} - 1\}$$

….інтегрування виразу $F_2(t, y, y')$

In [32]: rez2=rez.subs([(C2,seq1[C2])]) # ***Підстановка*** значення $C2$ у функцію ***y(t)***
In [33]: rez3=rez1.subs([(C2,seq1[C2])]) # ***Підстановка*** значення $C2$ у похідну $\boldsymbol{y'(t)}$
In [34]: u1=Function ('u1')(t) # ***Оголошення символьним*** виразу ***u1(t)***
In [35]: u1=rez2+rez2**2+rez3+rez3**2 # ***Формування*** виразу $F_1(t, y, y')$
In [36]: w=symbols('w') # ***Оголошення символьною*** змінної ***w***
In [37]: w=integrate(u1,(t,-1,1));w # ***Інтегрування*** виразу $F_2(t, y, y')$
Out[37]:

$$\frac{2755C_3^2}{14} + 81C_3 - 108C_3C_4 + \frac{1477C_4^2}{45} - \frac{108C_4}{5} + \frac{317}{30}$$

In [38]: eq4=diff(w,C3);eq4 # ***Визначення*** частинної похідної від ***w*** за C_3
Out[38]:

$$\frac{2755C_3}{7} - 108C_4 + 81$$

In [39]: eq5=diff(w,C4);eq5 # ***Визначення*** частинної похідної від ***w*** за C_4
Out[39]:

$$-108C_3 + \frac{2954C_4}{45} - \frac{108}{5}$$

In [40]: seq2=solve([eq4,eq5],C3,C4);seq2 # ***Розв'язання*** системи рівнянь ***eq4,eq5***
Out[40]: ….відносно C_3 та C_4

$$\left\{C_3\text{:} -\frac{67149}{318865}, C_4\text{:} -\frac{7776}{446411}\right\}$$

In [41]: rez4=rez2.subs([(C3,seq2[C3]),\ # ***Підстановка*** значень $C3, C4$ у функцію ***y(t)***
(C4,seq2[C4])]);rez4

$$-\frac{4860t^4}{63773} - \frac{67149t^3}{127546} + \frac{7980t^2}{63773} + t$$

In [42]: F=Lambda(t,rez4) # ***Формування*** символьної ламбда-функції ***F***
In [43]: display(Latex('$rez4(t)='+\ # ***Виклик на екран екстремалі rez4(t)***
+str(latex(F(t)))+'$')) ….у ***«красивому» форматі***

$$rez4(t) = -\frac{4860t^4}{63773} - \frac{67149t^3}{127546} + \frac{7980t^2}{63773} + t$$

In [44]: expr=-4860*t**4/63773- # ***Формування екстремалі*** у вигляді,
-67149*t**3/127546+7980*t**2/63773+t …, придатному для трансформації
In [45]: expr1=expr.evalf(3);expr1 #***Збереження*** **в** коефіцієнтах екстремалі
$-0.0762t^4 - 0.526t^3 + 0.125t^2 + t$ ….трьох значущих цифр
In [46]: from sympy.plotting import plot # ***Виклик*** із ***sympy.plotting*** модуля ***plot***
In [47]: exstremal=plot(expr1,(t,-1,1)) # ***Побудова графіка*** екстремалі

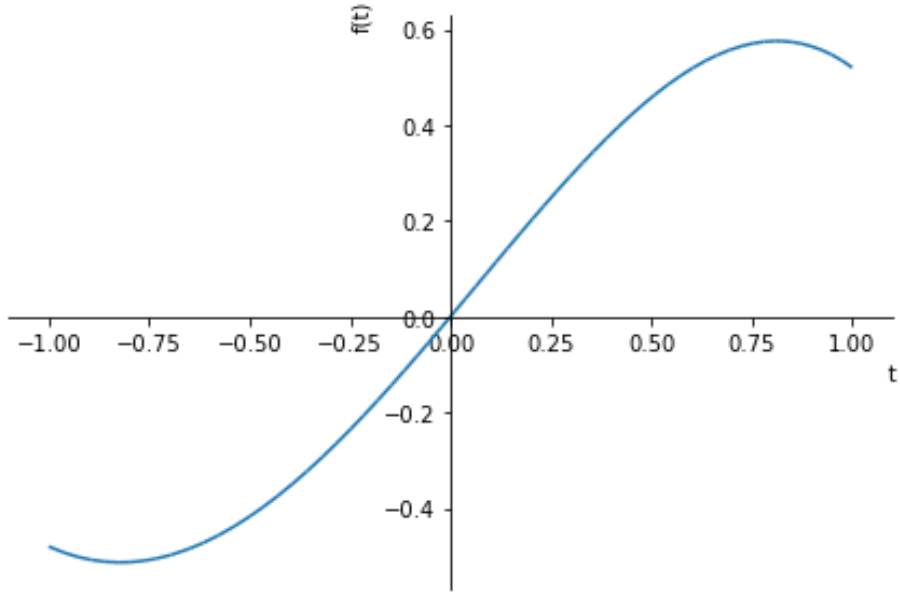

Рисунок 18 – Графік екстремалі функціонала $\boldsymbol{J_1 = \int_a^b F_1(t, y, y')dt}$ у випадку, коли $\boldsymbol{a = -1}$, $\boldsymbol{b = 1}$, $\boldsymbol{F_1(t, y, y') = y + y^2 + y' + (y')^2}$, а екстремаль $y(t)$ починається в точці ($y(0) = 0$, $y'(0) = 1$) та задовольняє обмеження $\boldsymbol{J_2 = \int_a^b F_2(t, y, y')dt}$, **де** $\boldsymbol{F_2(t, y, y') = y + y'}$, $\boldsymbol{J_2 = 1}$

Кінець програми 19.

***Перше підсумкове зауваження*:** в усіх програмах мовою Python, що подані вище, задля того, щоб довести їх до стадії отримання конкретних розв'язків і побудови конкретних графіків, використані задані авторами функції і функціонали, але це не завадить нашим читачам використовувати створені нами програми в процесі своїх досліджень, оскільки з наших пояснень щодо кожної команди, кожної програми кожному досліднику легко зрозуміти, як вбудовувати свою функцію чи функціонал в запропоновану нами програму.

Друге підсумкове зауваження*:** в створених програмах ми продемонстрували, як будувати графіки розв'язків з використанням окремого графічного редактора **matplotlib** та з використанням графічного редактора, вбудованого в ППП **sympy.** Але можна побудувати потрібний графік ще й за допомогою графічного редактора, вбудованого в ППП **mpmath**, трансформувавши в нього символьну функцію ***y(t) та використавши процедуру:

```
import mpmath as mp
f=lambdify(t,y(t),'mpmath')
mp.plot([f(t)],[-1,1])
```

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. 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.

2. Python. [Електронний ресурс]. Режим доступу : https://www.python.org//downloads//.

3. Бріггс Джейсон Р. Python для дітей (веселий вступ до програмування). / перекладачка з англійської Олександра Гордійчук. Львів : Видавництво старого Лева, 2019. 400 с.

4. Доля П. Г. Введение в научный Python. Харків : ХНУ ім. Каразіна, 2016. 265 с.

5. https://www.python.org/getit

6. Мокін Б. І., Мокін О. Б., Шалагай Д. О. Про один із підходів наближеного обчислення інтегралів Стілтьєса і Лебега на мові Python в задачах системного аналізу з дискретними моделями. *Вісник Вінницького політехнічного інституту*, 2021. № 3. С. 61–68.

Навчальне видання

Мокін Борис Івович
Мокін Віталій Борисович
Мокін Олександр Борисович

НАВЧАЛЬНИЙ ПОСІБНИК
для опанування студентами способів розв'язання задач з функціонального аналізу мовою Python

Частина 1

Редактор В. Дружиніна

Оригінал-макет підготовлено Б. Мокіним

Підписано до друку 15.06.2022. Формат 29,7×42¼. Папір офсетний.
Гарнітура Times New Roman. Ум. друк. арк. 14,78.
Наклад 35 пр. Зам. № 2022-057.

Видавець та виготовлювач –
Вінницький національний технічний університет,
редакційно-видавничий відділ.
21021, м. Вінниця, Хмельницьке шосе, 95,
ВНТУ, ГНК, к. 114.
Тел. (0432) 65-18-06.
press.vntu.edu.ua; *email*: irvc.vntu@gmail.com.
Свідоцтво суб'єкта видавничої справи
серія ДК № 3516 від 01.07.2009 р.