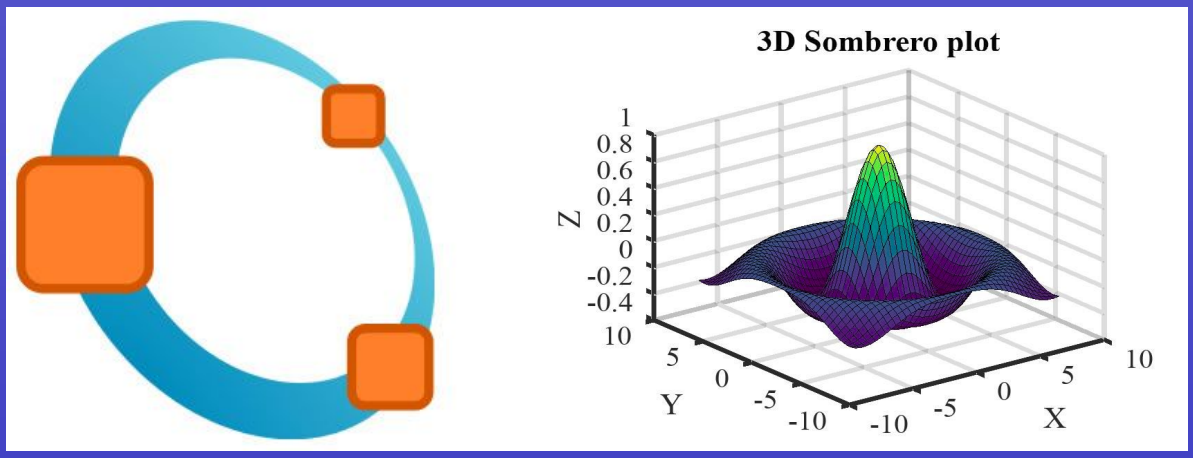
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«Харківський політехнічний інститут»

Г. В. Безпрозванних, Є. С. Москвітін



ПРИКЛАДНЕ ПРОГРАМУВАННЯ В GNU Octave

для студентів спеціальності 141 «Електроенергетика, електротехніка та електромеханіка»

НАВЧАЛЬНИЙ ПОСІБНИК

Харків 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Г. В. Безпрозванних, Є. С. Москвітін

ПРИКЛАДНЕ ПРОГРАМУВАННЯ В GNU Octave

Навчальний посібник
для студентів спеціальності
141 «Електроенергетика, електротехніка
та електромеханіка»

Рекомендовано Вченою радою НТУ «ХПІ»

Харків
НТУ «ХПІ»
2025

УДК 621-03
Б83

Рецензенти:

В.Є. Плюгін, д-р техн. наук, проф., завідувач кафедри систем електропостачання та електроспоживання, Харківський національний університет міського господарства імені О.М.Бекетова;

О.О. Мірошник, д-р техн. наук, проф., завідувач кафедри електропостачання та енергетичного менеджменту, Державний біотехнологічний університет

Рекомендовано Вченою радою НТУ «ХПІ» як навчальний посібник для студентів спеціальності 141 «Електроенергетика, електротехніка та електромеханіка» протокол №10 від 26 грудня 2024 року

Безпрозванних Г. В.

Б83 Прикладне програмування в GNU Octave: навчальний посібник для студентів спеціальності 141 «Електроенергетика, електротехніка та електромеханіка»./Г.В. Безпрозванних, Є.С. Москвітін. – Харків : НТУ «ХПІ», 2025. – 201с.

ISBN 978-617-05-0525-5

Навчальний посібник орієнтовано на вивчення безкоштовного обчислювального середовища GNU Octave для студентів спеціалізації «Електроізоляційна, кабельна та оптоволоконна техніка». Надано базові елементи програмування GNU Octave. Основний акцент зроблено на команди та функції прикладної мови програмування з графічною побудовою математичних функцій. Докладно описані типи даних, способи їх обробки та графічні можливості подання результатів. Для кращого розуміння роботи програм наводяться лістинги виконання команд.

Посібник призначено для студентів спеціальності 141 «Електроенергетика, електротехніка та електромеханіка», а також буде корисним аспірантам, викладачам та інженерно-технічним працівникам в галузі електроенергетики, електротехніки та електромеханіки.

Іл. 67. Табл. 8. Бібліогр. 27 назв.

УДК 621-03

ISBN 978-617-05-0525-5

ВСТУП

Використання програмного забезпечення у навчальному процесі є центральним стрижнем при підготовці бакалаврів та магістрів. Розумним рішенням є реалізація програмних додатків, призначених для моделювання різноманітних електроізоляційних та кабельних систем з використанням класичних мов програмування (Pascal, C, C++, Java, Python). Але у цьому випадку необхідні спеціальні знання. В останні роки використання інструментів обчислення, що містять інтерпретовані мови програмування високого рівня, було знайдено як вирішення цієї проблеми: Matlab, SciLab, Spyder IDE для мови Python і GNU Octave [1—5]. Найбільшу популярність у фахівців набув програмний пакет Matlab, який дозволяє вирішувати найрізноманітніші інженерні завдання. Для нього розроблені і продовжують розроблятися розширення Toolbox. Також у Matlab для користувача доступна сучасна мова програмування. Але з Matlab на законних підставах можуть працювати користувачі, які мають відповідну ліцензію (для студентів ліцензію зазвичай купує навчальний заклад). Як бути тим, хто може користуватися цим програмним продуктом лише в кількох комп'ютерних класах протягом обмеженого часу, а в ліцензії хмарний доступ до Matlab не передбачено?

Саме для таких користувачів розроблено вільно розповсюджуваний програмний продукт Octave.

Octave – потужна математична мова інтерпретувального типу. Інтерпретатор Octave входить до складу багатьох Linux-дистрибутивів (Debian, Ubuntu, Mandriva, ALT Linux), є версія і для ОС Windows. Система GNU Octave© пройшла довгий шлях розвитку від вузькоспеціалізованого модулю матричного програмного

забезпечення для великих комп'ютерів до універсальної інтегрованої системи комп'ютерної математики, орієнтованої на персональні комп'ютери. Математичний пакет Octave розроблявся для ОС Linux і тому саме в ОС Linux користувач отримає можливість повноцінно працювати з Octave і використовувати все можливості пакета [1—3].

Майже всі думають, що ім'я Octave має щось спільне з музикою. Насправді це ім'я одного з авторів супутнього програмного забезпечення для підручника з проєктування хімічних реакторів на рівні бакалаврату професора Октаве Левеншпіля.

Octave є програмним забезпеченням з відкритим кодом для числових обчислень, та засноване на мові програмування високого рівня зі значним інструментарієм для вирішення загальних числових задач лінійної алгебри, знаходження коренів нелінійних рівнянь, інтегрування звичайних функцій, маніпулювання многочленами, інтегрування звичайних диференціальних та диференціально-алгебраїчних рівнянь.

Octave підтримує безліч типів даних, включаючи скаляри, вектори, матриці, комплексні числа, рядки, структури та комірки. Має вбудовані функції для різних математичних операцій, таких як транспонування, звернення, власні значення, факторизація і т. п. Дозволяє створювати власні функції та змінні, а також використовувати умовні оператори, цикли, рекурсію та інші елементи програмування. Забезпечує зручний інтерфейс командного рядку для вирішення лінійних і нелінійних задач чисельним методом, легко розширюється та налаштовується за допомогою визначених користувачем функцій, написаних на власній мові Octave, або з використанням динамічно завантажених модулів, написаних на C++, C, Fortran, або іншій мові.

NASA використовує Octave для розробки систем стикування космічних апаратів. Jaguar аналізує дані, які отримують від їх болідів Formula 1. Шеффілдський Університет використовує Octave для розробки програмного забезпечення для розпізнавання ракових клітин.

Важливо, що синтаксис команд Octave ідентичний синтаксису команд Matlab [1—5]. Мова програмування Octave ідентична мові програмування Matlab. Тому все, що розроблено та виконувалося в середовищі Octave, може бути з мінімальними витратами перенесено в Matlab. Перенесення розробки з середовища Matlab в Octave не завжди відбувається просто: деякі функції Matlab в Octave не реалізовані. Ще більшою мірою це стосується пакетів розширення: вони є і в Octave, але не ідентичні пакетам Matlab. Тим не менш, на початковій стадії освоєння різниця між Matlab та Octave проявляється тільки в інтерфейсі користувача. Тому все, що дізнаємося про роботу з Octave, стане в нагоді при роботі з Matlab. Необхідно наголосити, що Octave також може використовуватися як пакетно-орієнтована мова.

У будь-якому разі це альтернативне програмне середовище надає можливості студентам виконувати роботу на своїх власних комп'ютерах, не купуючи обмежену у можливостях "студентську версію" комерційного Matlab.

Посібник призначено для студентів спеціальності 141 – Електроенергетика, електротехніка та електромеханіка при вивченні навчальних дисциплін «Прикладне програмування в електроізоляційній, кабельній та оптоволоконній техніці», «Фізика діелектриків», «Теорія електромагнітних полів в електроізоляційній, кабельній та оптоволоконній техніці”, «Математичне моделювання в електроізоляційній, кабельній та оптоволоконній техніці», “Основи оптоволоконної техніки: кабелі зв’язку”, “Кабельна техніка” тощо.

Розділ 1. Принципи та особливості роботи з інтерпретатором Octave

Octave, як мова програмування, що інтерпретується, дозволяє виконувати обчислення в інтерактивному режимі або у вигляді скриптів. В інтерактивному режимі користувач може вводити команди у командному рядку Octave та отримувати негайний результат. У пакетному режимі користувач може записати скрипт у текстовому файлі з розширенням .m і запустити його за допомогою команди **octave ім'я_файла.m**. Скрипт може містити послідовність команд, функцій, коментарів та інших елементів Octave.

Octave — крос-платформовий пакет, процедури його встановлення різних операційних систем (ОС) описано на ресурсі https://octave.org/. Працювати з пакетом можна або з командного рядка інтерпретатора мови або через інтегроване середовище розробки (IDE) з графічного інтерфейсу. Як правило, команда **octave** запускає роботу у командному рядку, команда **octave --gui** — у режимі IDE. Команда **octave <ім'я_файлу>** запустить до виконання файл (скрипт), що містить конструкції мови **Octave**, з результатами їх виконання та завершенням роботи.

Довідкові відомості щодо елементів Octave можна отримати за допомогою команди **help**, а опис її варіацій за командою **help help**. Наприклад команда **help --list** виведе список усіх операторів, ключових слів, вбудованих та завантажуваних функцій, доступних у поточному сеансі **Octave**. Команда **help <ім'я>** виведе інформацію про елемент, що має це ім'я. Команда **doc** виведе графічне вікно, що забезпечує пошук та перегляд **Octave** документації.

У вікнах IDE Octave з графічним інтерфейсом реалізовані: інтерпретатор Octave мови, що працює в режимі командного рядка;

область змінних, де перелічені створені у сеансі роботи змінні та їх типи; журнал виконаних команд; редактор коду з підсвічуванням синтаксису та вбудованим відладчиком; документація із можливістю пошуку; редактор змінних; вбудований диспетчер файлів. IDE доцільно використовувати під час налагодження програм.

Починаючи з версії 4.0.0 Octave має GUI (графічний інтерфейс користувача) (рис. 1.1) [1—2]. Використання GUI має кілька переваг:

— вбудований редактор перевіряє синтаксис коду Octave;

— у редакторі можна встановлювати точки розриву та переходити через свій код рядок за рядком;

— можна від'єднати деякі вікна від основної рами. Найчастіше використовують редактор в окремому вікні;

— можна переміщатися по дереву каталогів за допомогою верхнього рядка GUI;

— графіка, згенерована Octave, завжди відображатиметься в окремих вікнах.

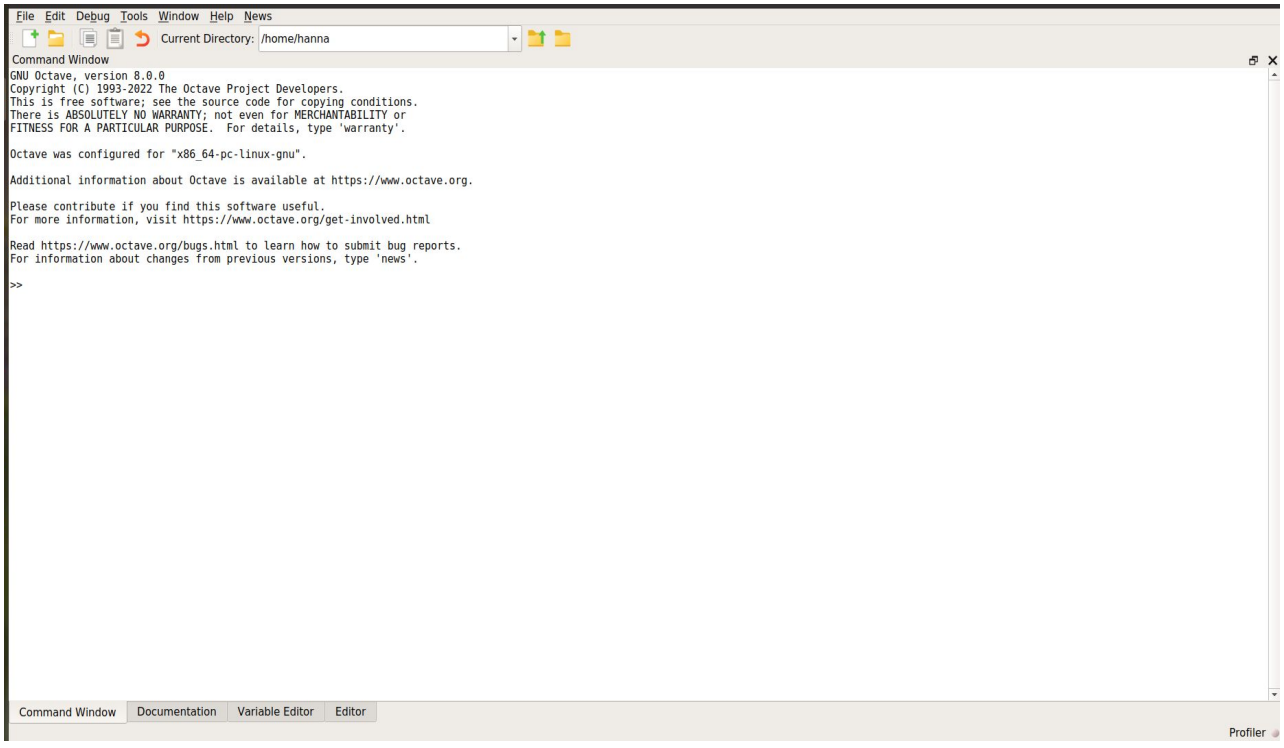


Рисунок 1.1 — Типовий скріншот Octave GUI

В межах однієї віконної рамки можна виконувати команди та спостерігати за їх результатами у вікні команд (**Command Window**); редагувати сегменти коду та запускати безпосередньо з вікна редактора (**Editor**) одним натисканням клавіші (F5); зібрати інформацію про всі змінні в поточній робочій області у вікні “Робоча область” (**Workspace**); працювати з вбудованим Файловим браузером (File Browser); прочитати стандартну документацію Octave у вікні Документація (**Documentation**); читати та змінювати поточний каталог у верхньому рядку GUI. Починаючи з версії 4.4.0, Octave має редактор змінних (**Variable Editor**).

1.1. Octave в ролі супер калькулятора

Система Octave сконструйована таким чином, що будь-які (іноді дуже складні) розрахунки можуть виконуватися *в режимі прямих розрахунків,* тобто без підготовки програми. Це перетворює Octave на надзвичайно потужний калькулятор, який дозволяє виконувати не тільки звичайні розрахунки, притаманні для калькуляторів (наприклад, арифметичні операції і обчислення елементарних функцій), а також операції з векторами і матрицями, комплексними числами, рядами і многочленами. Можна встановити і відобразити практично миттєво графіки різних функцій — від простої синусоїди до складної тривимірної фігури.

Робота з системою в режимі прямих обчислень носить діалогічний характер і відбувається за правилом «задати питання, отримати відповідь». Користувач на клавіатурі задає вираз, редагує його (за необхідності) в команді рядок і завершує вхід натисканням клавіші ENTER. На рис. 1.2 показані найпростіші обчислення.

Основною конструкцією, з якою працюють Matlab та GNU Octave, є матриця [1—4, 6—8]. Це не означає, що не можна

виконувати операції із звичайними (або комплексними) числами, просто треба розуміти, що такій кількості відповідає матриця розмірності |X|. Важливо, що **число розглядається в Octave як матриця з одного рядка та одного стовпця**. Octave заснований на тій же концепції, що і Matlab (Матрична лабораторія).

Навіть з таких простих прикладів можна зробити деякі повчальні висновки:

— для вказівки введення вихідних даних використовується символ **>>**;

— дані вводяться за допомогою найпростішого лінійного редактора;

— заблокувати вихід результату розрахунку деякого виразу після нього повинен бути встановлений символ **;** (крапка з комою);

— якщо не задано змінну для значення результату обчислень, тоді Octave призначає таку змінну з назвою **ans**;

— ознака присвоєння є звичайним знаком рівності для математиків **=**, а не комбінованим **:=**, як і в багатьох інших мовах програмування і математичних системах;

— результат обчислення відображаються у вихідних рядках (без підпису **>>**);

— вбудовані функції (наприклад, sin) записуються малими літерами і задаються їх аргументи в *круглих дужках*;

— діалог відбувається в стилі «задав запитання – отримав відповідь».

Наступний приклад, наведений на рис. 1.3, ілюструє застосування системи Octave для виконання простих векторних операцій. У цьому прикладі задається чотири-елементний вектор $\vec{V}$ зі значеннями елементів 1, 2, 3 і 4. Далі (зосередитеся на цьому!) функції синуса і

експонента обчислюються аргументом у вигляді *вектора*, а не скаляра.

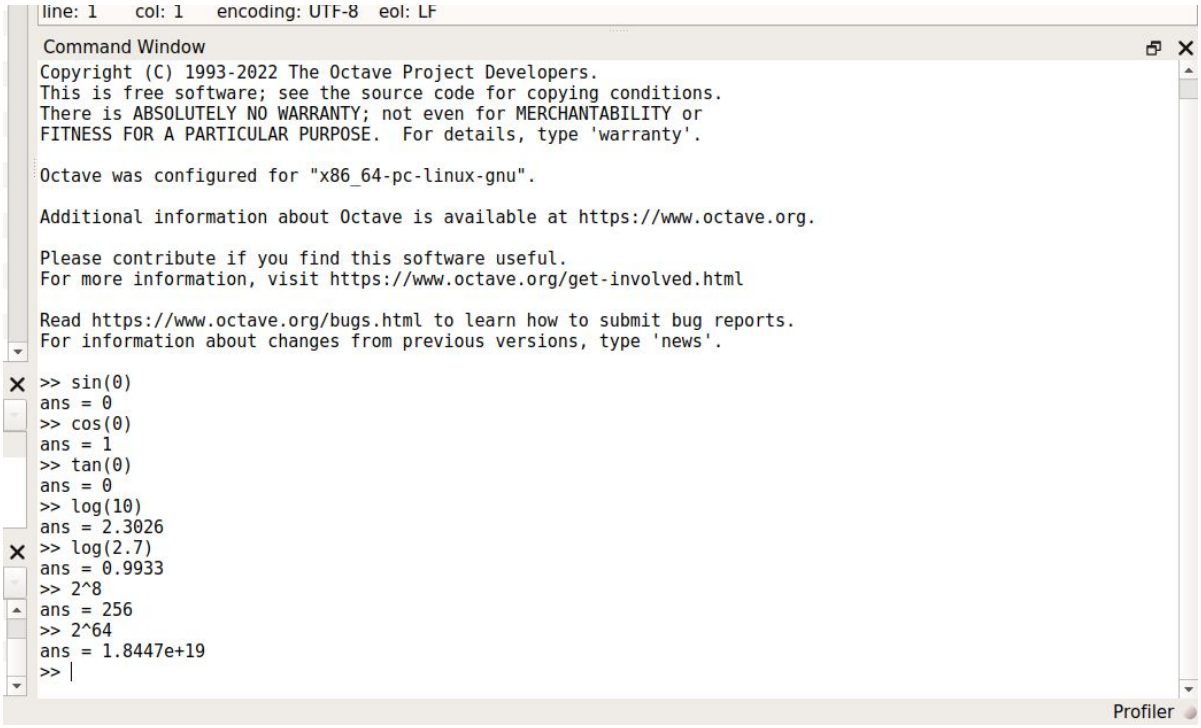


Рисунок 1.2 — Приклад простих обчислень в Octave

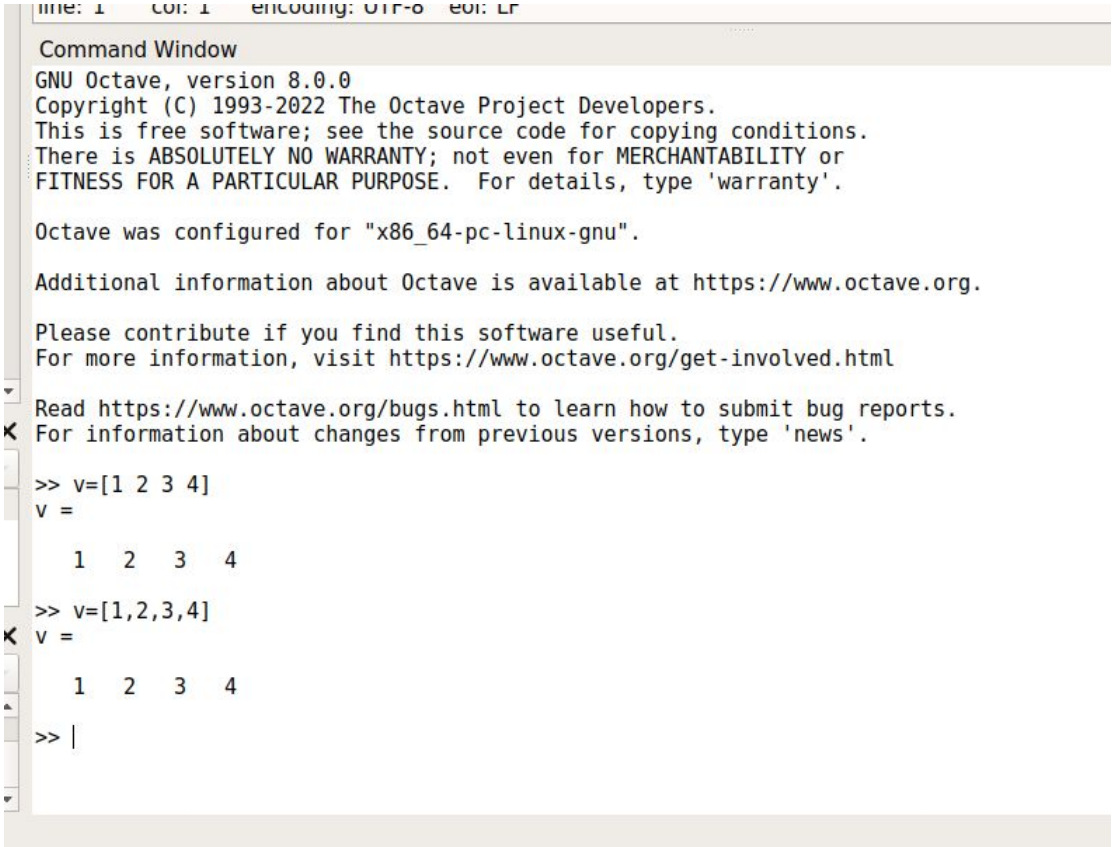


Рисунок 1.3 — Приклад простих векторних операцій в Octave

Два записи для вектора $\vec{V}$ =[l 2 3 4] і $\vec{V}$ =[1,2,3,4] ідентичні.

Таким чином, вектори визначаються переліком їх елементів, розділених просторами або комами. Список взято в квадратні дужки. Для виділення *n*-го елемента вектора $\vec{V}$ використовується вираз **V**(n), який визначає відповідні індексовані змінні.

У більшості математичних систем обчислення sin($\vec{V}$) і exp($\vec{V}$), де $\vec{V}$ — **вектор**, буде супроводжуватися помилкою, оскільки функції sin і exp повинні мати аргумент як **скалярна величина**.

Примітка: [**Скаляр** — величина, кожне значення якої може бути виражене лише одним числом. Це: довжина, ширина, маса, площа, об'єм, густина, температура та ін.]

Типовий **вектор** виражається двома чи більшою кількістю чисел (своїми координатами). І навіть для одномірного вектора лише одного числа мало — тому, що у вектора є ще напрямок. І точка застосування, якщо вектор не вільний. Векторами характеризують силові фізичні поля, швидкість та багато інших величин.]

Octave — матрична система, тож **вектор** — своєрідна матриця розміром 1×*n*. Результат розрахунків — вектор того ж розміру, що і аргумент V, але повернуті елементи вектора будуть синусами або експонентами елементів вектора $\vec{V}$.

Інший приклад (рис. 1.4) показує найпростіші матричні операції. Для матриці M розміром 2x2 обчислюється матриця MX=sin(M).

Матриця задається як ряд векторів, що представляють її рядки у квадратних дужках. Для поділу елементів векторів використовуються пробіли (або кома), а для відділення одного вектора від іншого — крапка з комою. Щоб вибрати один елемент матриці M, використовуємо вираз виду M(j,i), де M — ім'я матриці, j — номер рядка, а i — номер стовпця.

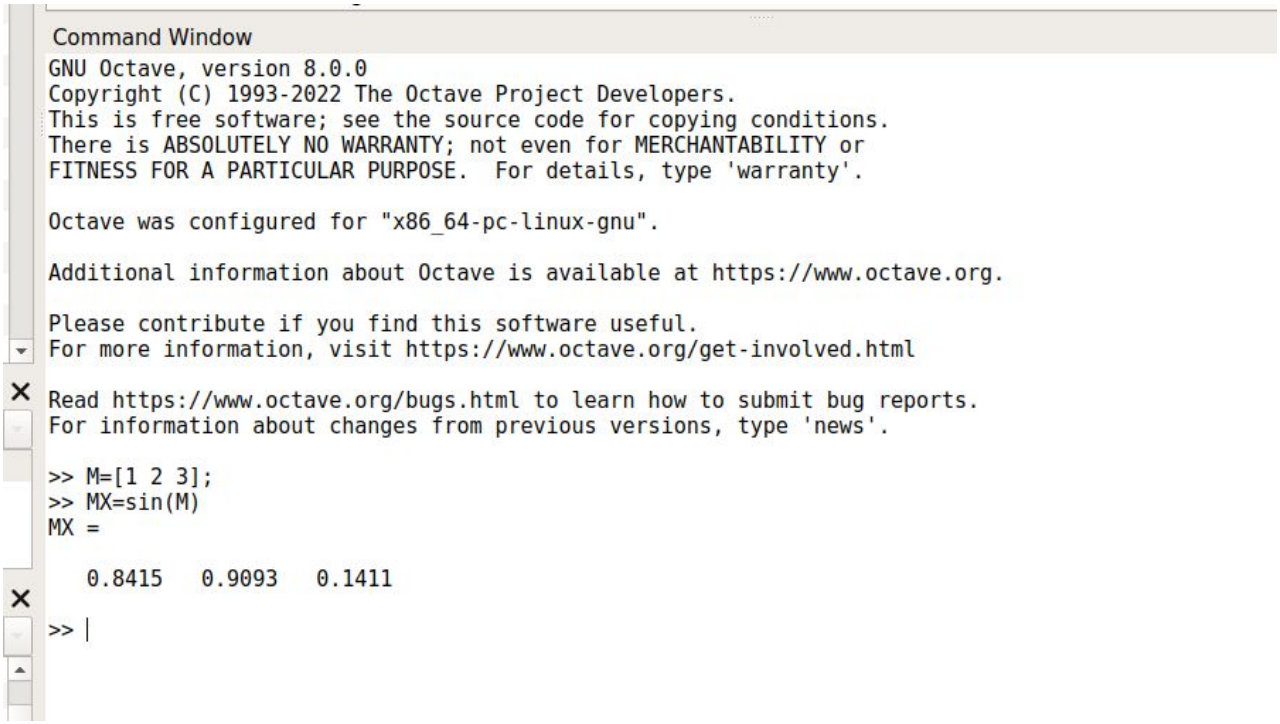


Рисунок 1.4 — Найпростіші операції з матрицею

Як видно з цих прикладів, введення початкових виразів для обчислень в системі Octave здійснюється в самому звичайному текстовому форматі. У тому ж форматі відображаються результати розрахунків, крім графічних розрахунків. Нижче наведено приклади запису розрахунків рис. 1.3 і рис. 1.4. [Для кращого сприйняття початок текстової частини виділено жирним шрифтом.]

Для початку виберіть >Довідка Octave> в меню >Довідка.

```
>> 2+3
ans=
5
>> sin(1)
ans=
0.8415
>> type sin
sin is a built-in function.
>> help sin
'sin' is a built-in function from the file libinterp/corefcn/mappers.cc
 -- sin (X)
    Compute the sine for each element of X in radians.
    See also: asin, sind, sinh.
```

```
Additional help for built-in functions and operators is
available in the online version of the manual.  Use the command
'doc <topic>' to search the manual index.
Help and information about Octave is also available on the WWW
at https://www.octave.org and via the help@octave.org
mailing list.
>> V=[l 2 3 4]
V =
1 2 3 4
>> sin(V)
ans =
0.8415      0.9093      0.1411     -0.7568
>> 3*V
ans =
3     6     9     12
>> V^2
??? Error using ==> ^
Matrix must be square.
>> V.^2
ans=
1 4 9 16
>> V+2
ans =
3    4    5    6
>>
```

Примітка:

Зверніть увагу на форму відповіді при виконанні простих операцій без вказівки змінної, якій вона приписується результат. У таких випадках Octave сама присвоює змінну ans, якій вона призначається результат і значення якого потім виводиться на екран.

Порівняйте ці записи із записами у фактичних сесіях (рис. 1.3 та 1.4): вони практично ідентичні. За винятком текстових версій прикладів для збереження місця **в цьому посібнику, прибрані проміжки між рядками**. Робимо показ презентації сесій як прямі копії екрану тільки тоді, коли вона пов'язана зі специфікою проведених розрахунків, наприклад, у разі виведення графіків або демонстрацією елементів інтерфейсу. В інших випадках

використовується вид сесії безпосередньо в тексті цього посібника в текстовому форматі, представленому вище – основний для командного режиму роботи з системою Octave. В цьому випадку вхідні рядки будуть позначено вхідним маркером **>>** на початку. Заради компактності порожні рядки будуть вилучені.

1.2. Дійсні і комплексні числа

Число — це найпростіший об'єкт Octave, який представляє кількісні дані. Числа можна вважати константами, назви яких збігаються з їх значеннями. Цифри використовуються в загальноприйнятому їх уявленні. Вони можуть бути цілими, дробовими, з фіксованою комою та з рухомою комою. Є можливість представлення чисел у форматі із зазначенням мантиси і порядка числа.

Нижче наведені приклади подання чисел:

```
0
2
-3
2.301 0.00001 123.45бе-24
-234.456e10
```

В мантисі чисел ціла частина відокремлена від дробової не комою, а крапкою, як це прийнято в більшості мов програмування. Для поділу порядку номера використовується символ E. Знак плюс для цифр не відзначається, а знак мінус числа *називається унарним мінусом*. Пробіли між символами в цифрах неприпустимі.

Числа можуть бути *комплексними: z* =Re(*x*)+Im(*x*)***i*. Такі числа містять фактичні дійсні Re(*z*) та уявні Im(*z*) частини. Уявна частина має множник *i* або *j*, що означає квадратний корінь з -1:

```
3i
2j
2+3i
-3.141i
-123.456+2.7e-3i
```

Функція **real(z)** повертає дійсну частину Re(z) комплексного числа. Функція **imag(z)** є уявною частиною Im(z) комплексного числа.

Для отримання модуля комплексного числа необхідно використовувати функцію **abs(z)**, а для визначення фази — **angle(z**). Нижче наведені найпростіші приклади роботи з комплексними числами:

```
>> i
ans=
0 +1.0000i
>> j
ans =
0 + 1.0000i
>> z=2+3i
z =
2.0000 + 3.0000i
>> abs(z)
ans =
3.6056
>> real(z)
ans=
2
>> imag(z)
ans =
3
>> angle(z)
ans =
0.9828
```

В Octave не прийнято ділити числа на цілі числа і дробові, короткі і довгі, і т.п. як це поширено в більшості мов програмування, хоча і вказувати числа в таких формах можливо.

У загальному випадку операції над числами виконуються в прийнятому форматі та зчитуються як формат *подвійної точності.*

Такий формат задовольняє переважну більшість вимог до числових розрахунків, але абсолютно непридатні для символічних обчислень з довільною (абсолютною) точністю. Символьні обчислення Octave може виконувати за допомогою спеціального символічного пакету розширень Symbolic Math Toolbox.

Константа — це заздалегідь визначене числове або символьне значення, представлене унікальною назвою. Числа (наприклад, 1, -2 і 1, 23) є безіменними *числовими константами.*

Інші види констант в Octave називаються *системними змінними*, тому що, з однієї сторони, вони встановлюються системою при завантаженні, а з іншої — можуть бути заміщені. Основні системні змінні, що використовуються в системі Octave, перераховані нижче:

i або j — уявна одиниця ($\sqrt{-1}$);

pi — число π= 3.141592653589793...;

eps — похибка операцій з номерами з рухомою комою (2^{-52});

realmin — найменше число з рухомою комою (2^{-1022});

realmax — найбільше число з рухомою комою (2^{1023});

inf — машинне значення нескінченності;

ans — змінна, яка зберігає результат останньої операції і зазвичай викликає його появу на екрані дисплея;

NaN — вказівка на нечисловий характер даних (Not-a-Number).

Приклади використання системних змінних:

```
>>
2*pi
ans =6.2832
 >> eps
ans =2.2204e-016
>> real min
ans=2.2251e-308
>> realmax
ans=1.7977e+308
>> 1/0
```

Warning: Divide by zero,
ans=Inf
>> 0/0
Warning: Divide by zero,
ans =NaN

Як зазначено, системні змінні можна *перевизначити.* Можна встановити системну змінну eps на інше значення, наприклад eps=0.0001. Однак важливим є те, що їх значення за замовчуванням встановлюються відразу після завантаження системи. Тому, на відміну від звичайних змінних, системні змінні ніколи не можуть бути невизначеними.

Символічна константа — це ланцюжок символів, укладених в апострофи, наприклад:

'Hello my friend!'

'Вітаю'

'2+3'

Якщо в апострофах поміщено математичний вираз, то він *не обчислюється* і розглядається просто як ланцюжок символів. Тому '2+3' не поверне число 5. Однак, за допомогою спеціальних функцій перетворення символьні вирази можна перетворити в обчислювані вирази. Відповідні функції перетворення розглянемо далі.

1.3. Знищення визначень змінних

Змінні займають певне розташування в пам'яті комп'ютера, яке називається *робочою областю* (Workspace). Для очищення робочого місця користуються функцією **clear** в різних формах, наприклад:

clear — знищення визначень усіх змінних;

clear x — знищення визначення змінної *x*;

clear a, b, c — руйнування визначень декількох змінних.

Знищена (стерта в робочому просторі) змінна стає невизначеною. Не можна використовувати невизначені змінні, і такі спроби будуть супроводжуватися повідомленнями про помилки. Ось приклади налаштування та знищення змінних:

```
>> x=2*pi
x =
6.2832
>> V=[l 2 3 4 5]
V =
1   2   3   4   5
>> MAT
??? Undefined function or variable 'MAT'.
>> MAT=[1 2 3 4; 5 6 7 8]
MAT=
1   2   3   4
5   6   7   8
>> clear V
>> V
??? Undefined function or variable 'V'.
>> clear
>> x
??? Undefined function or variable 'x'.
>> M
??? Undefined function or variable 'M'.
```

Зверніть увагу, що спочатку змінна V вибірково стирається, а потім всі інші змінні стираються за допомогою команди clear без параметрів.

1.4. Оператори і функції

Оператор — це спеціальне позначення для певної операції над даними, ***операндами***. Наприклад, найпростішими арифметичними операторами є позначка суми + , віднімання –, множення * і ділення /. Оператори використовуються спільно з операндами. Наприклад, у виразі 2 + 3 знак + є оператором додавання, а цифри 2 і 3 — операндами.

Слід зазначити, що більшість операторів відносяться до матричних операцій, що може викликати серйозні непорозуміння. Наприклад, оператори множення * і ділення / обчислюють добуток і частку ділення двох багатовимірних масивів, векторів або матриць. Існує ряд спеціальних операторів, наприклад, **** — оператор означає ділення *справа наліво,* а оператори **.*** і **./** середнє *поелементне* множення і *поелементне* ділення масивів відповідно.

Наведені нижче приклади ілюструють приклад векторних операцій:

```
>> V1=[2    4    6    8]
V1=
2 4 6 8
>> V2=[1    2    3    4]
V2 =
1 2 3 4
 >> V1/V2
ans =
2
>> V1.*V2
ans=
2   8   18   32
 >> V1./V2
ans =
2   2   2   2
```

Повний список операторів можна отримати за допомогою команди >> **help ops**. Поступово будуть розглянуто всі оператори системи Octave з обговоренням особливостей їх застосування. Частина повного списку операторів, що містять арифметичні оператори, наведено нижче.

Функції — це об'єкти, які мають унікальні імена, які виконують певні перетворення своїх аргументів і при цьому повертають результати цих перетворень. *Повернення результату* є відмінною рисою функцій. При цьому результат обчислення функції з одним

вихідним параметром підставляється на місце її виклику, що дає можливість використовувати функції в математичних виразах, таких як функція sin в 2*sin(pi/2).

>> help ops
Operators and special characters.
Arithmetic operators.

Plus	- Plus	+
Up! us	- Unary plus	+
Minus	- Minus	-
Umlnus	- Unary minus	-
Mtimes	- Matrix multiply	*
times	- Array multiply	*
mpower	- Matrix power	^
poWer	- Array power	.^
mldlvlde	- Backslash or left matrix divide	\
mrdlvlde	- Slash or right matrix divide	/
Idi-vide	- Left array divide	.\
rdlvlde	- Right array divide	./
kron	- Kronecker tensor product	kron

Багато функцій допускають ряд форм запису, які відрізняються переліком параметрів. Якщо функція повертає кілька значень, вона записується як [Yl, Y2....]=func(Xl, X2...), де Yl, Y2,... — перелік *вихідних параметрів* і X1, X2.... — список *вхідних аргументів* (параметрів). Список елементарних функцій можна знайти, запустивши команду **help elfun**, а зі списком спеціальних функцій, за допомогою команди

help specfun. Функції можуть бути *вбудованими* (внутрішніми) і *зовнішніми,* або *m-функціями.* Отже, вбудовані є найпоширенішими елементарними функціями, наприклад, **sin(x)** та **exp(y)**, тоді як функція **sinh(x)** є зовнішньою функцією. Зовнішні функції містять свої визначення в *m*-файлах. Налаштування таких функцій за допомогою спеціального редактора *m*-файлів буде розглянуто у розділі 3. Вбудовані функції зберігаються в скомпільованому ядрі системи Octave, тому виконуються вони вкрай швидко.

1.5. Застосування оператора : (двокрапка)

Дуже часто доводиться формувати впорядковані числові послідовності. Такі послідовності потрібні для створення векторів або значень абсциси при побудові графіків. Для цього в Octave використовується оператор **:** (двокрапка):

Початкове_значення**:**Крок**:**Кінцеве_значення

Ця конструкція породжує зростальну послідовність чисел, яка починається з початкового значення, йде з заданим кроком і закінчується кінцевим значенням. Якщо крок не вказано, установлюється значення 1. Якщо вказано кінцеве значення менше початкового значення, відображається повідомлення про помилку. Приклади застосування оператора **:** наведені нижче:

```
>> 1:5
ans =
1   2   3   4   5
>> i=0:2:10
i = 0   2   4   6   8   10
>> j=10:-2:2
j =
10   8   6   4   2
>> V=0:pi/2:2*pi;
>> V
V =
0   1.5708   3.1416   4.7124   6.2832
```

```
>> X= l:-.2:0
X=
1.0000   0.8000   0.6000   0.4000   0.20000
>> 5:2
ans=
Empty matrix:1-by-0
```

Як зазначалося, приналежність Octave до матричних систем вносить корективі в призначення операторів і призводить до інцидентів, якщо вони використовуються невміло. Розглянемо наступний типовий приклад:

```
>> x=0:5
x=
0 1 2 3 4 5
>> cos(x)
ans =
1.0000   0.5403   -0.4161   -0.9900   -0.6536   0.2837
>> sin(x)/x
ans = -0.0862
```

Розрахунок масиву косинусів виконано правильно. Обчислення масиву значень функції *sin*(*x*)/*x* дає несподіваний, на перший погляд, ефект – замість масиву з шістьма елементами розраховується єдине значення!

Причина «парадоксу» полягає в тому, що оператор / обчислює відношення *двох матриць, векторів* або багатовимірних *масивів.* Якщо вони мають однакову розмірність, то в результаті вийде одне число, яке в даному випадку і є тим, що дала система. Для отримання вектора значень sin(x)/x, необхідно використовувати спеціальний *поелементний* оператор ділення масивів — **./.** В такому випадку отримано масив чисел:

```
>> sin(x)./x
Warning: Divide by zero.
ans =
NaN   0.8415   0.4546   0.0470   -0.1892   -0.1918
```

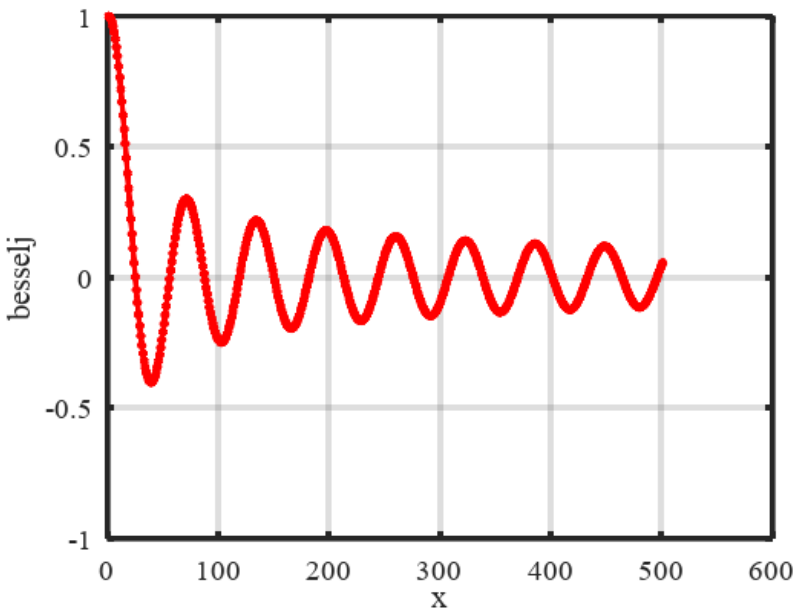
Однак і тут не обійшлося без особливостей. Так, при x=0 значення sin(x)/x надає тип невизначеності, що ліквідується 0/0=1. Втім, як і будь-яка числова система, Octave класифікує спробу поділу на 0 як помилку і відображає попередження. А замість очікуваного числового значення виводиться символьна константа **NaN**, що означає, що невизначеність 0/0 — це все ж не визначене число.

Вирази з оператором **:** можна використовувати як аргументи функції для отримання декількох значень. Наприклад, у наведеному нижче прикладі обчислюються функція Бесселя порядку від 0 до 5 зі значенням аргументу 0.5:

```
>> besselj(0:1:5,1/2)
ans =
0.9385   0.2423   0.0306   0.0026   0.0002   0.0000
```

А в наведеному нижче прикладі обчислюється шість значень функції Бесселя нульового порядку (рис.1.5) для значень аргументів від 0 до 5 з кроком 1:

```
>> besselj(0,0:1:5)
ans =
1.0000   0.7652   0.2239 -0.2601   -0.3971   -0.1776
```



Графік функції Бесселя нульового порядку для значень аргументу від 0 до 50 з кроком 0,1 представлено на рис. 1.5: besselj(0,0:0.1:50).

Рисунок 1.5 — Функція Бесселя нульового порядку

Таким чином, оператор **:** єзручним засобом для визначення правильної послідовності чисел та широко використовується при роботі з інструментами побудови діаграм. Можливості даного оператора наведено у практичних реалізаціях програмування.

1.6. Звітування про помилки та виправлення помилок

Діагностика помилок важлива при взаємодії з системою Octave. Навряд чи знайдеться користувач, який запам'ятовує точне написання тисяч операторів і функцій, що входять в систему Octave і в пакети додатків. Тому ніхто не застрахований від помилкового написання математичних виразів або команд. Octave діагностує введені команди і вирази і генерує відповідні повідомлення про помилки або попередження. Приклад повідомлення про помилку (розділити на 0) було щойно наведено.

Розглянемо ряд інших прикладів. Введемо, наприклад, помилковий вислів >> sqr(2) і натиснемо клавішу Enter. Система повідомить про помилку:

error: 'sqr' undefined near line 1, column 1

Це повідомлення вказує на те, що змінна або функція не визначена, і вказує, яка з них — sqr. В цьому випадку, звичайно, можна просто набрати правильний вираз. Однак у випадку з громіздким виразом краще використовувати редактор. Для цього достатньо натиснути клавішу вгору, щоб продивитись попередні рядки. В результаті в рядку введення з'являється вираз >> sqr(2) з курсором в кінці його. У Octave 7 тепер можна натиснути клавішу табуляції, система введе підказку, проаналізувавши вже введені символи. Якщо є кілька варіантів, доведеться натиснути клавішу табуляції ще раз. З трьох операторів, запропонованих системою,

обираємо sqrt. Тепер за допомогою клавіші вгору знову необхідно виділити потрібний рядок і, використовуючи ліву клавішу, встановити курсор після букви r. Натиснувши клавішу вгору, а потім клавішу ENTER, вираз прийме подібний вигляд:

```
>> sqrt(2)
ans= 1.4142
```

У разі наявності одного тільки варіанта для введених символів, після натискання клавіші табуляції система закінчила вхід без подачі рядків. Розрахунки дають очікуваний результат — значення квадратного кореня з двох.

У системі Octave зовнішні визначення використовуються так само, як і вбудовані функції та оператори. Ніяких додаткових вказівок на їх використання робити не слід. Потрібно лише переконатися, що використовувані визначення дійсно існують у вигляді файлів з розширенням .m. Однак при введенні назви визначення, що не існує, система відреагує на це повідомленням про помилку:

```
>> hsin(1)
error: 'hsin' undefined near line 1, column 1
>> sinh(1)
ans= 1.1752
```

У цьому прикладі спеціально не позначено назву зовнішньої функції для обчислювання гіперболічного синусу. Система припустила, що функція або змінна на ім'я hsin не визначається як внутрішня або m-функція. Функція, названа sinh, є частиною функцій системи Octave і задається як М-функція. Тим часом в останньому прикладі системі не надано вказівок на те, що саме зовнішню функцію слід шукати! І цей розрахунок є таким же простим, як обчислення вбудованої функції, подібної до sin. Звичайно, швидкість розрахунків за зовнішніми визначеннями несуттєво нижче, ніж на вбудованих функціях або операторах.

При цьому розрахунки відбуваються наступним чином: спочатку система швидко визначає, чи є введене слово серед службових слів системи. Якщо так, то необхідні обчислення виконуються відразу, якщо немає – то система шукає на диску *m*-файл з відповідним ім'ям. Якщо файлу не існує, відображається повідомлення про помилку, і обчислення зупиняються. Якщо файл знайдено, він завантажується з жорсткого диска в пам'ять машини і виконується. Даний алгоритм схожий з мовами програмування LOGO і FORT, використовуваними в мовах програмування, розроблених і адаптованих під завдання користувача.

Іноді при виведенні результатів розрахунків спостерігається скорочення **NaN** (зі слів Not a Number — не номер). Воно позначає **невизначеність**, таку як 0/0 або Inf/Inf, де Inf — системна змінна зі значенням нескінченності машини. Також можуть з'являтися різні попередження про помилки. наприклад, у разі ділення цільового число на 0, з'являється попередження типу «Warning: Devide by Zero.» («Увага: Ділення на нуль»). Діапазон чисел, представлених в системі, лежить від 10^{-308} до 10^{+308}.

В Octave потрібно розрізняти *попередження* про помилку та *повідомлення* про помилку. *Попередження* (зазвичай після слова, Warning) не припиняє розрахунки, а лише попереджає користувача про те, що діагностична помилка може вплинути на хід розрахунків. *Повідомлення про помилку* (після error) зупиняє обчислення.

1.7. Формати чисел

За замовчуванням Octave видає числові результати в *нормалізованій формі* з чотирма цифрами після десяткової точки і однією до неї. Взагалі така форма уявлення завжди влаштовує. При роботі з числовими даними можна задавати різні *формати* уявлення

чисел. Однак у будь-якому випадку всі обчислення проводяться з граничною, так званою *подвійною,* точністю. Для встановлення формату представлення чисел використовується команда **format name**, де name – ім'я формату. Для числових даних name може бути наступним повідомленням: short – коротке подання у фіксованому форматі (5 знаків), short e — коротке подання в експоненційному форматі (5 знаків мантиси та 3 знаки порядку), long — довге подання у фіксованому форматі (15 знаків), long e — довге подання в експоненційному форматі (15 знаків мантиси та 3 знаки порядку), hex — подання чисел у шістнадцятковій формі; bank — подання для грошових одиниць (табл.1.1).

Для ілюстрації різних форматів розглянемо вектор, що містить два елементи-числа: x=[4/3 1.2345e-6]. У різних форматах їх уявлення матимуть вигляд, представлений у табл.1.1.

Таблиця 1.1 — Формати чисел в Octave

format short	1.3333	0.0000
format short e	1.3333E+000	1.2345e-006
format long	1.333333333333338	0.000001234500000
format long e	1.333333333333338E+000	1.23450000000000e-006
format bank	1.33	0.00
format rat	4/3	9/7290401
format loose	1.33	1.2345e-006
format compact	1.3333	1.2345e-006

Завдання формату позначається тільки на *формі виведення* чисел. Обчислення все одно відбуваються у форматі подвійної точності, а введення чисел можливе у будь-якому зручному для користувача вигляді.

1.8. Формування векторів та матриць. Особливості завдання векторів та матриць

Описані вище прості правила обчислень поширюються і на значно складніші обчислення, які (при використанні звичайних мов програмування типу Бейсик чи Паскаль) вимагають складання спеціальних програм. Octave — система, спеціально призначена для проведення складних обчислень із векторами, матрицями та масивами [9-11]. При цьому вона за умовчанням передбачає, що кожна задана змінна — це вектор, матриця або масив. Усе визначається конкретним значенням змінної. Наприклад, якщо поставлено X=1, то це означає, що **X** — це вектор з єдиним елементом, що має значення 1. Якщо треба задати вектор з трьох елементів, їх значення слід перерахувати в квадратних дужках, розділяючи пробілами. Так, наприклад, присвоєння

```
>> V=[1 2 3]
V=
1   2   3
```

задає вектор **V**, що має три елементи зі значеннями 1, 2 та 3. Після введення вектора система виводить його на екран дисплея.

Завдання матриці вимагає вказівки кількох рядків. Для розмежування рядків використовується знак **;** (крапка з комою). Цей же знак у кінці введення запобігає виведенню матриці або вектора (і взагалі результату будь-якої операції) на екрані дисплея. Так, введення

```
>> M=[1 2 3; 4 5 6; 7 8 9];
```
задає квадратну матрицю, яку можна вивести:
```
>> M
M =
1     2     3
4     5     6
7     8     9
```

Можливе введення елементів матриць та векторів у вигляді арифметичних виразів, що містять будь-які доступні системі функції, наприклад:

```
>> V=[2+2/(3+4) exp(5) sqrt(10)];
>> V
V =
2.2857     148.4132     3.1623
```

Для вказівки окремого елемента вектора або матриці використовуються вирази V1) або M(i,j). Наприклад, якщо задати

```
>> M(2,2)
ans= 5
```

то результат дорівнюватиме 5. Якщо потрібно присвоювати елементу M(i,j) [У тексті програм Octave краще не використовувати і та j як індекси, оскільки і та j — позначення квадратного кореня з √–1. Можна використати I і J.] нове значення *x*, слід використовувати вираз

```
M(ij)=x
```

Наприклад, якщо елементу M(2,2) треба присвоювати значення 10, слід записати

```
>> M(2,2)=10
```

Вираз M(i) з одним індексом дає доступ до елементів матриці, розгорнутим в один стовпець. Така матриця утворюється із вихідної, якщо поспіль виписати її стовпці.

Наступний приклад пояснює такий доступ до елементів матриці M:

```
>> M=[1 2 3; 4 5 6; 7 8 9]
M =
1     2     3
4     5     6
7     8     9
>> M(2)
ans =4
>> M(8)
ans =6
>> M(9)
ans =9
>> M(5)=100;
>> M
M =
1     2     3
4     100   6
7     8     9
```

Можливе завдання векторів та матриць з комплексними елементами, наприклад:

```
>> i=sqrt(-1);
>> CM=[1 2; 3 4] + i*[5 6; 7 8]
```

або

```
>> CM=[1+5*i 2+6*i; 3+7*i 4+8*i]
```

Це створює матрицю:

```
CM=
1.0000 + 5.0000i     2.0000 + 6.0000i
3.0000 + 7.0000i     4.0000 + 8.0000i
```

Поряд з операціями над окремими елементами матриць і векторів система дозволяє проводити операції множення, поділу та піднесення до степеня відразу над усіма елементами, тобто над масивами. Для цього перед знаком операції ставиться крапка. Наприклад, оператор * означає множення для векторів або матриць, а оператор **.*** — поелементне множення всіх елементів масиву. Так, якщо M — матриця, то M.*2 дасть матрицю, всі елементи якої

помножені на скаляр — число 2. Втім, для множення матриці, на скаляр обидва вирази — M*2 і M.*2 — виявляються еквівалентними.

Є також ряд спеціальних функцій завдання векторів і матриць. Наприклад, функція **magic(n)** задає магічну матрицю розміру n×n, у якої сума всіх стовпців, всіх рядків і навіть діагоналей дорівнює одному й тому ж числу [5-7]:

```
>> M=magic(4)
M =
16   2    3   13
5    11   10  8
9    7    6   12
4    14   15  1
>> sum(M)
ans=
34    34    34    34
>> sum(M')
ans=
34    34    34    34
>> sum(diag(M))
ans=34
>> M(1,2)+M(2,2)+M(3,2)+M(4,2)
ans= 34
```

Вже сама собою можливість створення такої матриці за допомогою простої функції **magic** зацікавить любителів математики. Векторних та матричних функцій у системі безліч. Нагадаємо, що для стирання змінних робочої області пам'яті служить команда **clea**r.

1.9. Об'єднання матриць

Описаний спосіб завдання матриць дозволяє виконати операцію **конкатенації** — об'єднання малих матриць у велику [6-7,9]. Наприклад, створимо спочатку магічну матрицю розміру 3×3:

```
>> A=magic(3)
A=
8   1   6
3   5   7
4   9   2
```

Тепер можна побудувати матрицю, що містить чотири матриці:

```
>> B=[A A+16;A+32 A+16]
B =
8   1   6    24   17    22
3   5   7   19    21   23
4   9   2    20   25   18
40   33    38    24   17  22
35    37    39   19   21   23
36    41    34   20   25   18
```

Отримана матриця має розмір 6x6. Обчислимо суму її стовпців:

```
>> sum(B)
 ans =
126     126     126     126     126     126
```

Сума однакова по всім стовпців. Для обчислення суми рядків використовуємо команду

```
>> sum(B.')
ans =
78     78     78    174     174     174
```

Запис **B.'** означає транспонування матриці B, тобто заміну рядків стовпцями. Цього разу сума виявилася різною. Це відкидає припущення, що матриця B теж є магічною. Для **істинно магічної матриці суми стовпців та рядків мають бути однаковими**:

```
>> D=magic(6)
D=
35    1     6     26     19      24
3    32     7     21     23      25
31   9     2      22     27      20
8    28    33    17     10      15
30    5    34    12     14      16
4    36    29    13     18      11
```

```
>> sum(D)
ans=
111 111 111 111 111 111
>> sum(D.')
ans=
111 111 111 111 111 111
```

Більш того, для магічної матриці однаковою є і сума елементів за основними діагоналями (головною діагоналі та головною анти-діагоналі).

1.10. Видалення стовпців та рядків матриць

Для формування матриць та виконання ряду матричних операцій виникає необхідність видалення окремих стовпців та рядків матриці. Для цього використовуються порожні квадратні дужки [ ]. Для матриці M:

```
>> M=[1 2 3; 4 5 6; 7 8 9]
M =
1    2    3
4    5    6
7    8    9
```

Видалимо другий стовпець з використанням оператору **:** (двокрапка):

```
>> M(:,2)=[ ]
1    3
4    6
7    9
```

Використовуючи оператор **:** (двокрапка), видалимо другий рядок:

```
>> M(2,:)=[ ]
M =
1    3
7    9
```

1.11. Оператори відношення та їх функції

Оператори *відношення* служать для порівняння двох величин, векторів чи матриць. Усі оператори відношення мають два операнди, наприклад x і y, і записуються, як показано в табл.1.2.

Дані оператори виконують поелементне порівняння векторів або матриць однакового розміру і повертають значення 1 (**True**), якщо елементи ідентичні, і значення 0 (**Fals**e) інакше.

Таблиця 1.2 — Оператори та функції відношення

Функція	**Назва**	**Оператор**	**приклад**
Eq	Дорівнює	= =	x = = y
Ne	Не дорівнює	~ =	x ~ = y
Lt	Менше ніж	<	x<y
Gt	Більше ніж	>	x>y
Le	Менше або дорівнює	<=	x<=y
Ge	Більше або дорівнює	>=	x>=y

Якщо операнди — дійсні числа, то застосування операторів відносини тривіально:

```
>> eq(2,2)
ans =1
>> 2==2
ans =1
>> ne(1,2)
ans =1
>> 2~=2
ans =0
>> 5 > 3
ans =1
>> le(5,3)
ans =0
```

Слід зазначити, що оператори <, <=, > і >= при комплексних операндах використовують порівняння лише дійсні частини операндів — уявні відкидаються. У той самий час оператори == і ~= ведуть порівняння з урахуванням як дійсної, так і уявної частин операндів. Наступні приклади пояснюють це:

```
>> (2+3i)>=(2+i)
ans=1
>> (2+3i)>(2+i)
ans=1
>> abs(2+3i)>abs(2+i)
ans =1
>> (2+3i)==(2+i)
ans =0
>> (2+3i)>(2+i)
ans =1
```

Якщо один із операндів — скаляр, відбувається порівняння всіх елементів другого операнда-масиву зі значенням цього скаляра:

```
>>M=
-1    0
1     2
>> M>=0
ans =
0     1
1     1
```

У загальному випадку оператори відносини порівнюють два масиви одного розміру та видають результат у вигляді масиву того ж розміру:

```
>> M>[0 1;.1 0]
ans =
0    0
0     1
```

Таким чином, спектр застосування операторів відношення в системі Octave ширший, ніж у звичайних мовах програмування, оскільки операнда є не тільки числа, але і вектори, матриці і масиви. Можливе застосування операторів ставлення і до символьних виразів:

```
>> 'b'>'a'
ans =1
>> 'abc'= ='abc'
ans =
1     1     1
>> 'cba'<'abc'
ans =
0     0     1
```

У цьому випадку символи, що входять до виразів, видаються своїми ASCII-кодами. Рядки сприймаються як вектори, що містять значення кодів. Усе це треба враховувати під час використання керувальних структур мови програмування, у яких широко застосовуються оператори відносини.

1.12. Оператори логіки

Оператори логіки та відповідні їм функції служать для реалізації поелементних логічних операцій над елементами однакових за розміром масивів (табл.1. 3).

Робота операторів пояснюється наведеними нижче прикладами:

```
>>A=[1 2 3];
>>B=[1 0 0];
>> and(A,B)
ans =
1    0    0
>> or(A,B)
ans =
1    1     1
>> A&B
ans =
1    0    0
>> A|B
ans=
1     1     1
>> not(A)
ans =
0    0    0
```

```
>> not(B)
ans =
0     1     1
>> ~B
ans=
0    1    1
>> xor(A,B)
ans =
0     1     1
>> any(A)
ans =1
>> all([0 0 0])
ans =0
>> all(B)
ans =0
>> and('abc','012')
ans =
1     1     1
```

Таблиця 1.3 — Оператори логіки та функції Octave

Функція	**Назва**
And	Логічне І (AND) &
Or	Логічне АБО (OR) |
Not	Логічне НЕ (NOT) ~
Xor	Виключне АБО (EXCLUSIVE OR)
Any	Правильно, якщо всі елементи вектора дорівнюють нулю
All	Правильно, якщо всі елементи вектора не дорівнюють нулю

Зверніть увагу, що аргументами логічних операторів можуть бути числа та рядки. При аргументах-числах логічний нуль відповідає числовому нулю, а будь-яке відмінне від нуля число сприймається як логічна одиниця. Для рядків діє зазначене правило — кожен символ рядка представляється своїм ASCII-кодом.

1.13. Спеціальні символи

До класу операторів у системі Octave належать також спеціальні символи. Вони призначені для створення найрізноманітніших об'єктів вхідної мови, мови програмування системи та надання їм різних форм. У табл. 1.4 наведено опис повного набору спеціальних символів.

Розглянемо їх докладніше:

: (двокрапка) — формування під-векторів та під матриць з векторів та матриць. Оператор **:** — один з операторів, що найчастіше використовуються в системі Octave.

Оператор **:** використовує такі правила для створення векторів:

j:k — те саме, що і [j,j+l,...,k];

j:k — порожній вектор, якщо j>k;

j:i:k — те саме, що і [j, j+i ,j+2i .....k];

j:i:k — порожній вектор, якщо i>0 та j>k або якщо i<0 та j<k, де 1, j та k — скалярні величини.

Нижче показано, як вибирати за допомогою оператора **:** рядки, стовпці та елементи з векторів, матриць та багатовимірних масивів:

A(:, j) — це j-й стовпець з A;

A(i,:) — це і-й рядок з A;

A(:,:) — еквівалент двовимірного масиву. Для матриць це аналогічно A;

A(j:k) — це A(j), A(j+l),...,A(k);

A(:,j:k) — це A(:,j),A(:,j+l),...,A(:,k);

A(:,:,k) — це k-а сторінка тривимірного масиву A;

A(i,j,k,:) — вектор, виділений із чотиривимірного масиву A. Вектор включає елементи A(1, j.k.l), A(i. j,k,2), A(i. j,k.3) і т. п.;

A(:) — записує всі елементи масиву A як стовпця.

Таблиця 1.4 — Спеціальні символи Octave

Позначення	**Назва**	**Категорія**
:	Двокрапка	colon
( )	Круглі дужки	paren
[ ]	Квадратні дужки	paren
{ }	Фігурні дужки	paren
.	Десятирічна крапка	punct
.	Виділення поля структури	punct
..	Батьківський каталог	punct
...	Продовження рядка	punct
,	Розділювач	punct
;	Крапка з комою	punct
% #	Коментар	punct
i	Виклик команди операційної системи	punct
=	Привласнення	punct
'	Лапки	punct
'	Транспонування	transpose
'	Транспонування з комплексним поєднанням	ctranspose
[,]	Горизонтальна конкатенація	horzcat
[;]	Вертикальна конкатенація	vertcat
( ), { }..	Привласнення підмасиву	subsasgn
( ), { }..	Посилання на підмасив	subsref
	Індекс підмасиву	subsindex

Символи **( )** (круглі дужки) використовуються для визначення порядку виконання операцій в арифметичних виразах, вказівки послідовності аргументів функції та вказівки індексів елемента вектора або матриці. Якщо X і V — вектори, то X(V) можна записати у вигляді [X(V(D), X(V(2)).....X(V(n))].) Елементи вектора V повинні бути цілими числами, щоб їх можна було використовувати як індекси елементів масиву X. Помилка генерується в тому випадку, якщо індекс елемента менше одиниці або більше, ніж розмір size (X). Такий же

принцип індексування дійсний і для матриць, вектор W — n компонентів, то A(V,W) буде матрицею розміру m×n, сформованою з елементів матриці A, індекси якої елементи векторів V і W.

Символи **[ ]** (квадратні дужки) використовуються для формування векторів та матриць:

[6.9 9.64 sqrt(-l)] — вектор, що містить три елементи, розділених пробілами;

[6.9. 9.64 i] — такий самий вектор;

[1+j 2-j 3] та [1 +j 2 -j 3] — різні вектори: перший містить три елементи, а другий п'ять;

[11 12 13; 21 22 23] — матриця розміру 2×3. Крапка з комою розділяє перший і другий рядки.

Ще кілька прикладів:

A = [ ] — зберігає порожню матрицю A;

A(m,:) = [] — видаляє рядок m із матриці A;

A(:,n) = [ ] — видаляє стовпець n з матриці A.

Символи **{}** (фігурні дужки) використовуються для формування масивів осередків. Наприклад, {magic(3) 6.9 'hello'} — масив осередків із трьома елементами.

Символ **.** (десяткова крапка) використовується для відокремлення дробової частини чисел від цілої. Наприклад, 314/100, 3.14 і .314e1 — одне й те число.

Крім того, символ крапки **.** використовується для виділення полів структур. Наприклад, A.(field) і A(i).field, де A — структура означає виділення поля структури з ім'ям «field».

Нижче наведено призначення інших спеціальних символів Octave:

.. (верхній каталог) — перехід по дереву каталогів на один рівень вгору;

... (продовження) — три або більше крапок наприкінці рядка вказують на продовження рядка;

; (крапка з комою) — використовується всередині круглих дужок для поділу рядків матриць, а також наприкінці операторів для заборони виведення на екран результату обчислень;

, (кома) — використовується для поділу індексів елементів матриці та аргументів функції, а також для поділу операторів мови Octave. При поділі операторів у рядку кома може замінюватися на точку з комою з метою заборони виведення на екран результату обчислень;

%,# (знак відсотків) — використовується для вказівки логічного кінця рядка. Текст, що знаходиться після знака відсотка, сприймається як коментар та ігнорується (за винятком російськомовних коментарів, які нерідко ведуть до помилкових команд);

! (знак оклику) — є покажчиком введення команди операційної системи. Рядок, що йде за ним, сприймається як команда операційної системи;

= (знак рівності) — використовується для присвоєння значень в арифметичних виразах;

' (одинична лапка, апостроф) — текст у лапках представляється як вектор символів з компонентами, що є ASCII кодами символів. Лапка всередині рядка задається двома лапками. Наприклад:

```
>> a='Hello''my friend'
a =Hello'my friend
```

' (транспонування з комплексним сполученням) — транспонування матриць, наприклад **A'** — транспонована матриця А. Для комплексних матриць транспонування доповнюється

комплексним сполученням. Рядки транспонованої матриці відповідають стовпцям вихідної матриці;

.' (транспонування) — транспонування масиву, наприклад **A.'** — транспонований масив A. Для комплексних масивів операція сполучення не виконується;

[,] — горизонтальна конкатенація Так, **[A,B]** – горизонтальна конкатенація (об'єднання) матриць A та B. A та B повинні мати однакову кількість рядків. [A] діє аналогічно. Горизонтальна конкатенація може бути використана для будь-якого числа матриць в межах одних дужок: **[A,B,C]**. Горизонтальна та вертикальна конкатенації можуть використовуватись одночасно: **[A,B;C]**;

[;] — вертикальна конкатенація. Так, **[A;B]** – вертикальна конкатенація (об'єднання) матриць A і B. A та B повинні мати однакову кількість стовпців. Вертикальна конкатенація може бути використана для будь-якого числа матриць в межах одних дужок: **[A;B;C]**. Вертикальна та горизонтальна конкатенації можуть використовуватись одночасно: **[A;B,C]**;

(),{} — привласнення під масиву. Наведемо кілька прикладів:

A(I)=B – присвоює значення елементів масиву елементам масиву A, які визначаються вектором індексів I. Масив B повинен мати таку ж розмірність, як і масив I, або може бути скаляром;

A(I,J)=B – присвоює значення масиву елементам прямокутної під матриці A, які визначаються векторами індексів I і J. Масив B повинен мати length(I) рядків і length(J) стовпців;

A{I}=B, де A — масив осередків, де I — скаляр, поміщає копію масиву B у задану комірку масиву A. Якщо I має більше одного елемента, то з'являється повідомлення про помилку.

1.14. Функції часу та дати

Ряд функцій необхідно для повернення поточного часу та дати:

calendar(d) — повертає календар на місяць, в який потрапляє день, заданий аргументом d (дні відраховуються від початку літочислення);

calendar — повертає матрицю розміром 6×7, що містить календар на поточний місяць. Календар починається з неділі (перший стовпець) і завершується суботою;

calendar(y,m) — повертає календар на місяць, заданий аргументом m і рік, заданий аргументом;

Виклик функції без надання результату видає календар на екран.

Приклади наведено нижче.

```
>> calendar
```

Mar 2024

S	M	Tu	W	Th	F	S
0	0	0	0	0	1	2
3	4	5	6	7	*8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	0	0	0	0	0	0

```
>> calendar(700477)
```

Nov 1917

S	M	Tu	W	Th	F	S
0	0	0	0	1	2	*3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	0
0	0	0	0	0	0	0

Функція **fix(clock)** заокруглює число секунд до цілого значення. Приклад наведено нижче:

```
>> c=clock
c =
2.0240e+03   3.0000e+00   8.0000e+00   1.0000e+00   4.0000e+00
1.8478e+01
>> fix(clock)
ans =
2024     3     8     1     5    18
```

Функція **cputime** — повертає час роботи процесора (в секундах), використаний системою Octave з моменту її запуску. Це число може вийти за рамки внутрішньої установки часу, і тоді відлік часу починається заново.

Приклад:

```
>> t1=cputime; w=surf(peaks(30)); cputime-t1
ans =
0.2800
```

str = date — повертає рядок, що містить дату у форматі дд-ммм-рррр (день-місяць-рік). Приклад:

```
>> d = date
d =
24-Mar-2024
```

Функція **datenum** — перетворює рядок дати на порядковий номер дати, який відраховується з деякого початкового дня (01.01.00);

datenum(str) — перетворює дату, задану рядком str, на порядковий номер дати. Рядок string повинен мати один із наступних форматів: 0, 1, 2, 6, 13, 14, 15 або 16, визначених для функції datestr;

datenum(Y,M,D) — повертає порядковий номер дати для відповідних масивів елементів Y, M та D (рік, місяць, день). Масиви Y, M та D повинні мати однакову розмірність (при цьому будь-які з них можуть бути скалярами);

datenum(Y,M,D,H,MI,S) — повертає порядковий номер дати для відповідних масивів елементів Y, M, D, H, MI та S (рік, місяць, день, години, хвилини, секунди). Масиви Y, M, D, H, MI та S повинні мати однакову розмірність (при цьому будь-які з них можуть бути скалярами).

Приклад представлено нижче:

```
>> n1 = datenum('26-Nov-1998')
n1 =730085
>> Y=[1998,2000];M=[1,12];D=23;N=datenum(Y,M,D)
N =
729778     730843
```

Функція **datestr(D, dateform)** — перетворює кожен елемент масиву порядкових номерів дати D в рядок. Аргумент dateform визначає формат результату; dateform може бути номером або рядком відповідно до табл.1.5.

Функція **datevec(A)** — перетворює вхідні величини масив розмірності n×6, кожен рядок якого являє собою вектор [Y,M,D,H,MI,S]. Перші п’ять елементів вектора — цілі числа. Масив A може складатися або з рядків, що задовольняють формат функції **datestr**, або з скалярних величин, створених функціями **datenum** і **now**;

[Y, M, D, H, MI, S] = datevec(A) — повертає компоненти вектора дати як індивідуальні змінні.

Таблиця 1.5 — Формати подання дати

Dateform (номер)	**Dateform (рядок)**	**Приклад**
0	'dd-rmiM-yyyy HH:MM:SS'	1l-Mar-1995 03:45
1	'dd-mmM-yyyy'	0l-Mar-1995
2	'mm/dd/yy'	03/01/95
3	'mmm'	Mar
4	'm'	M
5	'mm'	3
6	'mm/dd'	03/01
7	'dd'	1
8	'ddd'	Wed
9	'd'	W
10	'YYYY'	1995
11	'YY	95
12	'mmmyy'	Mar95
13	'HH:MM:SS'	15:45:17

Будь-який компонент вхідного вектора, який не вписується в нормальний діапазон дат, перетворюється на наступний діапазон (наприклад дата, що не існує June 31 перетворюється на July 1). Допускаються значення нульового місяця та нульового дня.

```
>> n = datevec('11/31/98')
n=
1998     12      1      0      0      0
```

```
>> n = datevec(710223)
n =
1944     7     10     0     0
```

Функція **eomday (Y, M)** — повертає останній день року та місяця, заданих відповідно елементами масивів Y та M.

Приклад (знаходження високосних років двадцятого століття):

```
>> y = 1900:1999;
>> E = eomday(y,2);
>> y(find(E==29))
ans=
Columns 1 through 6
1904 1908 1912 1916 1920 1924
Columns 7 through 12
1928 1932 1936 1940 1944 1948
Columns 13 through 18
1952 1956 1960 1964 1968 1972
Columns 19 through 24
1976 1980 1984 1988 1992 1996
```

Функція **etime(t2,t1)** — повертає тривалість проміжку часу (в секундах), що задається векторами t1 та t2. Вектори повинні задовольняти формат, що видається функцією clock;

T = [рік місяць день година хвилини секунди].

Функція працює некоректно, якщо в поточний проміжок часу потраплять межі місяця або року, що трапляється вкрай рідко і виправляється при повторі операції. Приклад (обчислюється час, що витрачається на швидке перетворення Фур'є з 2048 точками):

```
>> x = rand(2048,1); t = clock; fft(x); etime(clock,t); etime (clock,t)
ans =0.0500
```

now — повертає поточний час та дату у формі числа. Використання **rem(now,1)** повертає лише час, а **floor(now) —** лише дату. Приклад:

```
>> t1 = now, t2 = rem(now,1)
t1 =7.3009e+005
t2 =0.6455
```

tic — запускає таймер;

toc — виводить час, що минув із моменту запуску таймера;

t = toc — повертає час у змінну t.

Приклад:

```
>> tic, surf(peaks(50)); toc
elapsed_time =0.7600
```

Функція **[N,S] = weekday(D)** — повертає день тижня у вигляді числа N та у вигляді рядка S для кожної дати масиву D.

Приклад:

```
>> D=[728647,735730]; [N,S] = weekday(D)
N =
2     1
S=
Mon     Sun
```

1.15. Елементарні математичні функції

У системі Octave визначено такі елементарні математичні функції:

abs(X) — повертає абсолютну величину кожного числового елемента вектора $\vec{X}$. Якщо X містить комплексні числа, **abs(X)** обчислює модуль кожного числа.

Приклади:

```
>> abs(-5) = 5
>> abs(3+4i) =5
>> abs([1 -2 1 3i 2+3i ])
ans =
1.0000     2.0000     1.0000     3.0000     3.6056
```

exp(X) — повертає експоненту кожного елемента X. Для комплексного числа z = x + i*y функція exp(z) обчислює комплексну експоненту: exp(z)=exp(x)*(cos(y)+i*sin(y)).

Приклади:

```
>> exp([1 2 3])
ans =
2.7183    7.3891    20.0855
>> exp(2+3i)
ans =
-7.3151 + 1.0427i
```

Функція **factor(n)** — повертає вектор-рядок, що містить прості множники числа n. Для масивів ця функція не застосовується.

Приклад:

```
>> f = factor(221)
f =
13    17
```

G=gcd(A, B) — повертає масив, що містить найбільші спільні дільники відповідних елементів масивів цілих чисел A і B.

Функція **gcd (0,0)** повертає значення 0, в інших випадках масив G, що повертається, містить позитивні цілі числа;

[G, C, D] = gcd(A, B) — повертає масив найбільших загальних дільників G та масивів C та D, які задовольняють рівнянню A(i).*C(i) + B(i) .*D(i) = G(i). Корисні для виконання елементарних ермітових перетворень.

Приклади:

```
>> A=[2 6 9];
>> B=[2 3 3];
>> gcd(A,B)
ans =
2    3    3
>> [G,C,D]=gcd(A,B)
G =
2    3    3
C =
0    0    0
D=
1    1    1
```

lcm(A,B) — повертає найменші загальні кратні для відповідних парних елементів масивів A та B. Масиви A та B повинні містити

позитивні цілі числа та мати однакову розмірність (будь-який з них може бути скаляром).

Приклад:

```
>> A=[1 3 5 4];
>> B=[2 4 6 2];
>> lcm(A,B)
ans =
2    12    30    4
```

log(X) — повертає натуральний логарифм елементів масиву X. Для комплексного чи негативного z, де $z=x+y\times i$, обчислюється комплексний логарифм як $\log(z)=\log(|(z)|)+i\times atan2(y,x)$. Функція логарифму обчислюється для кожного елемента масиву. Область визначення функції включає комплексні та негативні числа, що може призвести до непередбачуваних результатів при некоректному використанні.

Приклад:

```
>> X=[1.2 3.34 5 2.3];
>> log(X)
ans=
-0.1823 1.2060 1.6094 0.8329
```

log2(X) — повертає двійковий логарифм елементів масиву X;

[F,E] = log2(X) — повертає масив дійсних значень F і масив цілих чисел E. Елементи масиву F зазвичай знаходяться у діапазоні 0.5< abs(F) < 1. Для дійсних X повертаються масиви F, що задовольняють рівняння виду $X=F.\times 2.\times E$. Для нульових значень X повертаються F = 0 та E = 0.

Приклад:

```
>> X=[2 4.678 5;0.987 1 3];
>> [F,E] = log2(X)
F =
0.5000 0.5847 0.6250
0.9870 0.5000 0.7500
```

```
E =
2   3   3
0   1   2
```

log10(X) — повертає десятковий логарифм для кожного елемента X. Область функції включає комплексні числа, що здатне призвести до непередбачених результатів при некоректному використанні:

```
>> X=[1.4 2.23 5.8 3];
>> log10(X)
ans =
0.1461 0.3483 0.7634 0.4771
```

mod(x,y) — повертає x mod y;

mod(X, Y) — повертає залишок від поділу X на Y (тобто, X - Y.*floor(X./Y)) для ненульового Y, X — в іншому випадку. Якщо операнди X та Y мають однаковий знак, функція mod(X, Y) повертає той самий результат, що mod(X, Y). Однак (для позитивних X та Y) mod(-x,y) = rem(-x,y)+y.

Приклади:

```
>> M = mod(5,2)
M =1
>> mod(10,4)
ans =2
```

pow2(Y) — повертає масив X, де кожен елемент є 2^Y ;

pow2(F,E) — обчислює $X=F*2^E$ для відповідних елементів F та E. Аргументи F та E — масиви дійсних та цілих чисел відповідно:

```
>> d=pow2(pi/4,2)
d =3.1416
```

p=nextpow2(A) – повертає такий показник ступеня p, що 2^p є abs(A). Ця функція ефективно застосовується для **швидкого перетворення Фур'є**. Якщо A не є скалярною величиною, то nextpow2 повертає значення nextpow2(length(A)).

Приклади:

```
>> x=[2 6 7 8 9 3 4 5 6 7 7 8 4 3 2 4];
>> length(x)
ans =16
>> p = nextpow2(ans)
p=4
>> x=4;
>> p = nextpow2(x)
p=2
>> x=45;
>> p = nextpow2(x)
p=6
```

Функція **primes(n)** повертає вектор-рядок простих чисел, менших або рівних n. Приклад:

```
>> p = primes(25)
p =
2 3 5 7 11 13 17 19 23
```

[N,D] = rat(X) — повертає масиви N і D, такі, що N./D апроксимує X з точністю 1e-6*norm(X(:),!). Навіть при тому, що всі числа з плавучою комою — раціональні числа, іноді бажано апроксимувати їх дробами, у яких чисельник та знаменник є по можливості малими цілими числами. Завдяки функції **rat** це можна зробити;

[N,D]=rat(X,tol) — повертає масиви N і D, такі що N./D апроксимує X з точністю tol.

rat(X) без вихідних параметрів просто видає на екран масив ланцюгових дробів;

rat(X,strlen) — повертає рядок, отриманий шляхом спрощеної раціональної апроксимації елементів X. Аргумент strlen – довжина рядка, що повертається. Функція повертає знак «*», якщо отримане значення не може бути надруковане у рядку, довжина якого задана strlen. За умовчанням strlen=13. Той самий алгоритм апроксимації використовується у командному вікні Octave при заданні раціонального формату виведення командою format rat.

Приклад:

```
>> [g,j]=rat(pi,1e-10)
g=
312689
j =
99532
```

sqrt(A) — повертає квадратний корінь кожного елемента масиву X. Для негативних та комплексних елементів X функція sqrt(X) обчислює комплексний результат.

Приклад:

```
>> A=[25 21.23 55.8 3];
>> sqrt(A)
ans =
5.0 4.6076 7.4699 1.7321
```

На рис. 1.6 представлені графіки низки поширених алгебраїчних функцій. Ці графіки отримані в результаті виконання наступного файлу сценарію:

```
pkg load symbolic
syms x
subplot(2,2,1),ezplot(x*2,[-5 5]),xlabel(''),grid on,
subplot(2,2,2),ezplot(exp(x),[-2 2]),xlabel(''),grid on,
subplot(2,2,3),ezplot(log(x),[0 5]),grid on,
subplot(2,2,4),ezplot(sqrt(x),[0 10]),grid on,
```

Графіки дають наочне уявлення про поведінку представлених ними функцій. Зверніть увагу на застосування графічної команди ezplot з пакету Symbolic Math ToolBox (вона відрізняється від звичайної команди ezplot Octave відсутністю укладання символьних змінних в ''), команди syms, що також входить у пакет Symbolic Math Toolbox і задає символьну змінну x, і дещо незвичайне застосування команди **xlabel ('')**.

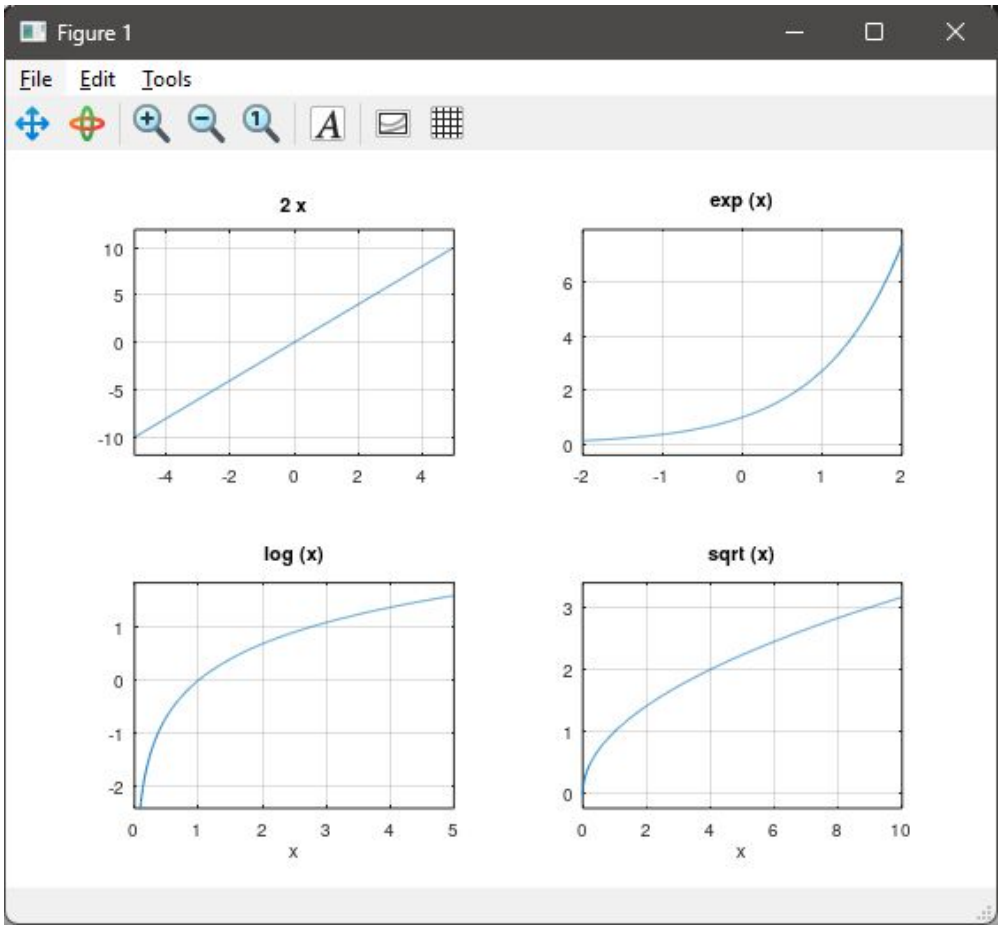


Рисунок 1.6 — Приклади подання ряду графіків алгебраїчних функцій

Ця команда з аргументом у вигляді порожнього рядка знімає висновок позначення горизонтальної осі двох верхніх графіках. Якщо цього не зробити, символ «x» виявиться накладеним на найменування функцій нижніх графіків, яке команда ezplot виводить над графіками автоматично.

У системі Octave визначено також тригонометричні та зворотні тригонометричні функції. Функції обчислюються у кожному з елементів масиву. Вхідний масив припускає комплексні значення. **Нагадування: всі кути у функціях задаються у радіанах** (див. розділ 3).

Імена функцій, що працюють з аргументом (повертають результати) у градусах, закінчуються на букву d (degree - градус у тригонометрії)!

Функція **acos(X)** — повертає арккосинус для кожного елемента X. Для дійсних значень X в області [-1, 1] acos(X) повертає дійсне значення з діапазону діапазону [0, pi], для дійсних значень X поза області [-1, 1] acos(X) повертає комплексне число.

Приклади:

```
>>Y = acos(0.5)
Y=1.0472
>> acos([0.5 1 2])
ans =
1.0472 + 0i        0 + 0i        0 + 1.3170i
```

acot(X) — повертає арккотангенс для кожного елемента X.

Приклад:

```
>> Y=acot(0.l)
Y=1.4711
```

acsc(X) — повертає арккосеканс для кожного елемента X:

```
>> Y= acsc(3)
Y=0.3398
```

asec(X) — повертає арксеканс для кожного елемента X:

```
>> Y=asec(0.5)
Y = 0 + 1.3170i
```

asin(X) — повертає арксинус для кожного елемента X. Для дійсних значень X в області [-1, 1] asin(X) повертає дійсне число з діапазону [-pi/2, pi/2], для дійсних значень X поза області [-1, 1] asin(X) повертає комплексне число.

Приклад:

```
>> Y= asin (0.278)
Y=0.2817
```

atan(X) — повертає арктангенс для кожного елемента X. Для дійсних значень X atan(X) знаходиться в області [-pi/2, pi/2]:

```
>> Y=atan(1)
Y =0.7854
```

atan2(Y, X) — повертає масив P тієї ж розмірності, що X та Y, та містить поелементно арктангенси відношення дійсних частин Y та X. Уявні частини ігноруються. Елементи P знаходяться у інтервалі [-pi, pi]. Специфічний квадрант визначено функціями sign(Y) та sign(X). Це відрізняє отриманий результат від atan(Y/X), який обмежений інтервалом [-pi/2, pi/2]:

```
>> atan2(l,2)
ans =0.4636
```

cos(X) — повертає косинус для кожного елемента X:

```
>>X=[1 2 3];
>> cos(X)
ans =0.5403     -0.4161     -0.9900
```

cot(X) — повертає котангенс для кожного елемента X:

```
>> Y = cot(2)
Y=-0.4577
```

csc(X) — повертає косеканс для кожного елемента X:

```
>> X=[2 4.678 5;0.987 1 3];
>> Y = csc(X)
Y =
1.0998     -1.0006     -1.0428
1.1985     1.1884     7.0862
```

sec(X) — повертає масив тієї ж розмірності, що і X, який складається з секансів елементів X:

```
>> X=[pi/10 pi/3 pi/5];
>> sec(X)
ans =
1.0515     2.0000     1.2361
```

sin(X) — повертає синус для кожного елемента X:

```
>> X=[pi/2 pi/4 pi/6 pi];
>> sin(X)
ans =
1.0000     0.7071     0.5000     0.0000
```

tan(X) — повертає тангенс кожного елемента X:

```
>> X=[0.08 0.06 1.09]
X=
0.0800 0.0600 1.0900
>> tan(X)
ans=
0.802     0.0601     1.9171
```

Наступний файл сценарію дозволяє спостерігати графіки чотирьох тригонометричних функцій (рис. 1.7):

```
>>pkg load symbolic
>>syms x
>>subplot(2,2,1),ezplot(sin(x),[-5 5]),xlabel(''),grid on
>>subplot(2,2,2),ezplot(tan(x),[-5 5]),xlabel(''),grid on
>>subplot(2,2,3),ezplot(asin(x),[-1 1]),grid on
>>subplot(2,2,4),ezplot(atan(x),[-5 5]),grid on
```

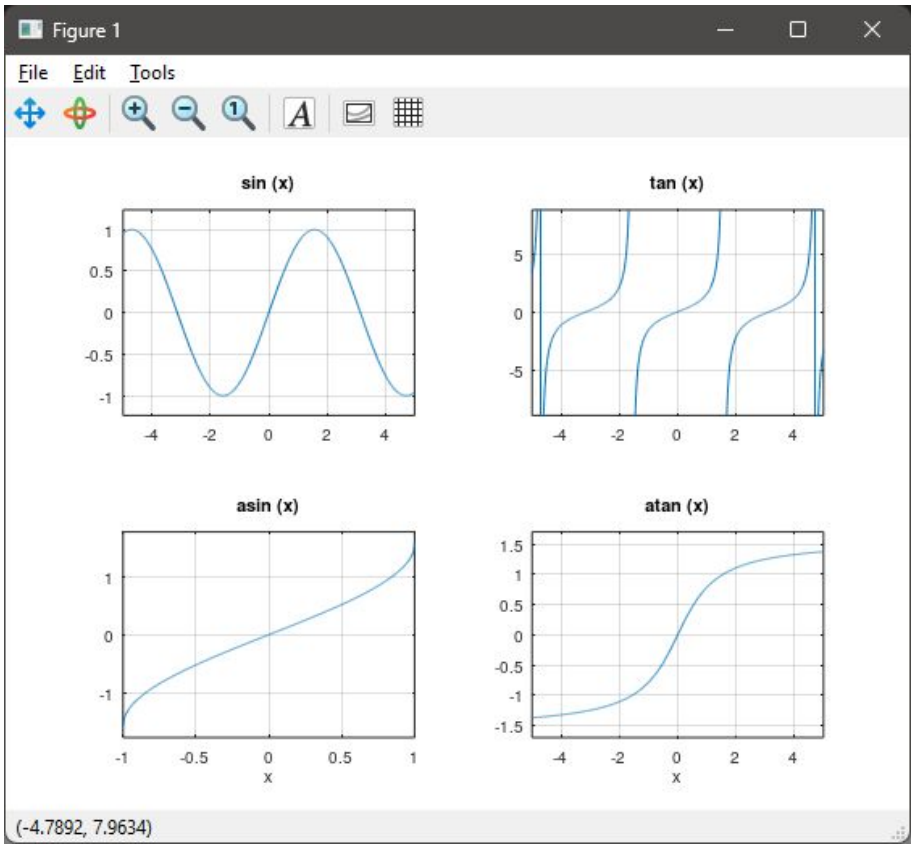


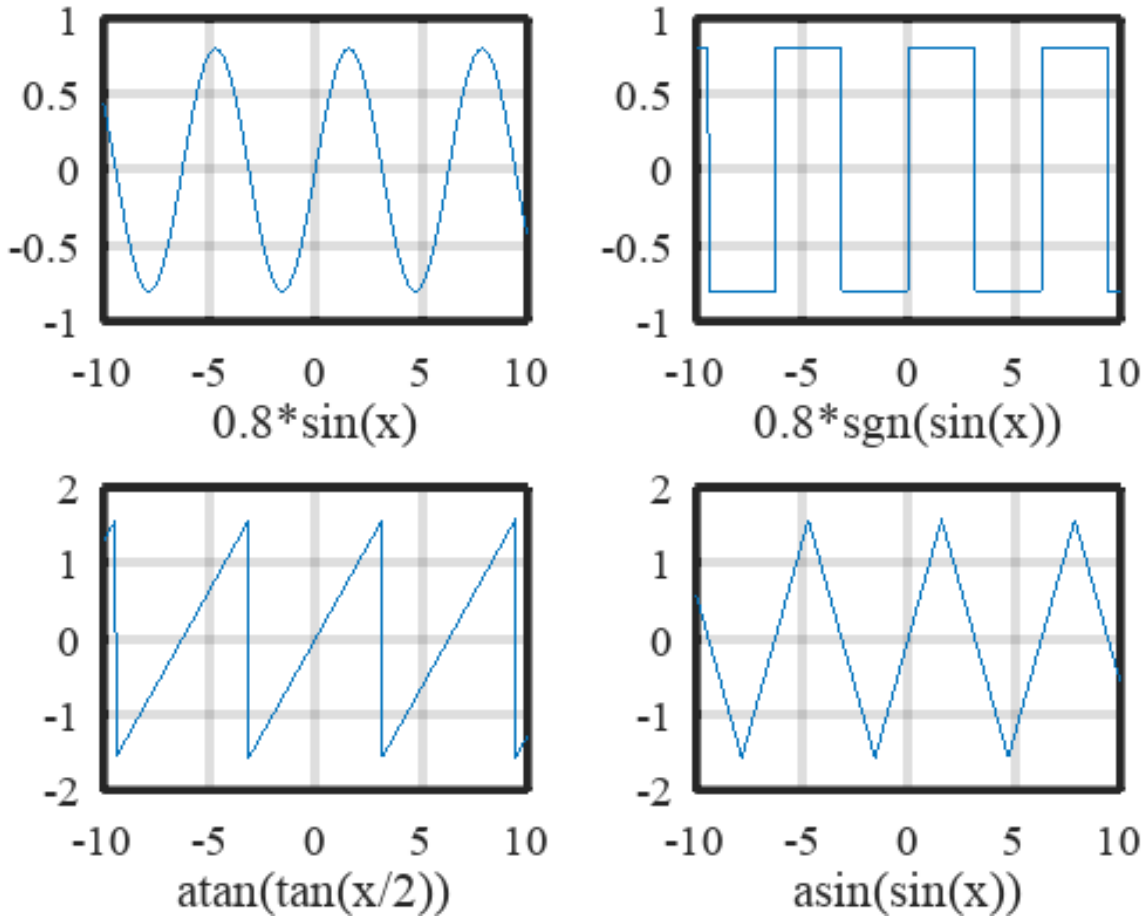
Рисунок 1.7 — Приклади графіків тригонометричних функцій у Octave

Оскільки багато тригонометричних функцій періодичні, з'являється можливість формування з них цікавих комбінацій —

типових тестових сигналів, що використовуються при моделюванні радіоелектронних пристроїв. Наступний файл-сценарій дозволяє побудувати графіки для таких комбінацій, що створюють із синусоїди три найбільш поширені сигнали: прямокутні, пилкоподібні та трикутні імпульси: [У пакеті розширення Signal Processing Toolbox є спеціальні функції для генерації таких сигналів — square та sawtooth.].

```
>>x=-10:0.01:10;
>>subplot(2,2,1),plot(x,0.8*sin(x)),xlabel('0.8*sin(x)')
>>subplot(2,2,2),plot(x,0.8*sign(sin(x))),xlabel('0.8*sgn(sin(x))')
>>subplot(2,2,3),plot(x,atan(tan(x/2))),xlabel('atan(tan(x/2))')
>>subplot(2,2,4),plot(x,asin(sin(x))),xlabel('asin(sin(x))')
```

Відповідні графіки подано на рис. 1.8.



Рисунки 1.8 — Графіки синусоїди, прямокутних, пилкоподібних та трикутних коливань

Додатковий ряд графіків, отриманих комбінаціями елементарних функцій, показано на рис. 1.9. Ці графіки будуються за наступним файлом-сценарієм:

```
>>x=-15:0.01:15;
>>subplot(2,2,1),plot(x,sin(x).^3),xlabel('sin(x)^3')
>>subplot(2,2,2),plot(x,abs(sin(x))),xlabel('abs(sin(x))'),axis([-15 15 -1 1]),
>>subplot(2,2,3),plot(x,tan(cos(x))),xlabel('tan(cos(x))')
>>subplot(2,2,4),plot(x,csch(sec(x))),xlabel('csch(sec(x))')
```

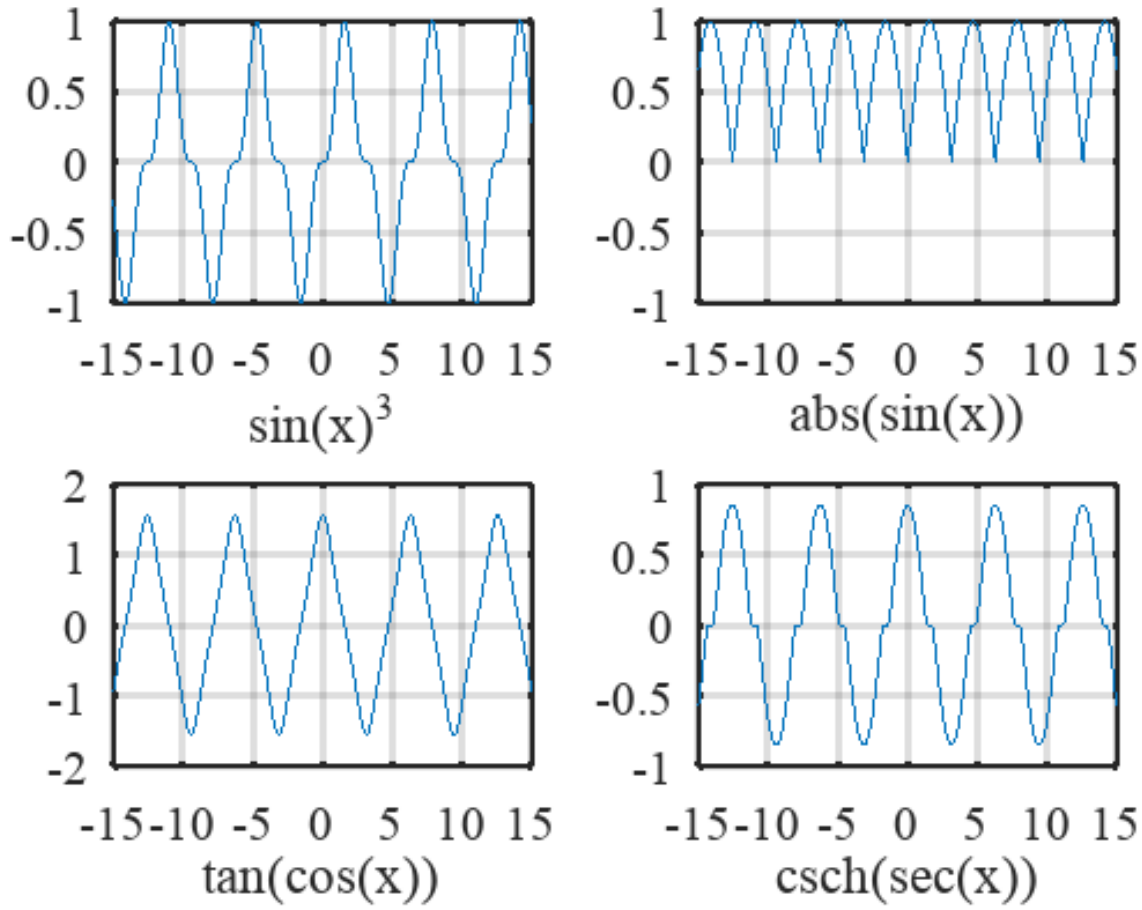


Рисунок 1.9 — Графіки періодичних сигналів без розривів

Ці графіки моделюють сигнали, одержувані, наприклад, при випрямленні синусоїдальної напруги (або струму) і при проходженні синусоїдальних сигналів через нелінійні електричні ланцюги.

Поряд із тригонометричними функціями в математичних розрахунках часто використовуються і гіперболічні функції. Нижче

наведено список таких функцій, визначених у системі Octave. Функції обчислюються для кожного елемента масиву. Вхідний масив припускає комплексні значення.

Усі кути в тригонометричних функціях вимірюються у радіанах. Імена функцій, що працюють з аргументом (повертають результати) у градусах, закінчуються на букву d (degree — градус у тригонометрії).

Функція **acosh(X)** — повертає гіперболічний арккосинус для кожного елемента X:

```
>>Y= acosh (0.7)
Y =
0 + 0.7954i
```

acoth(X) — повертає гіперболічний арккотангенс для кожного елемента X.

Приклад:

```
>>Y = acoth (0.1)
Y=
0.1003 + 1.5708i
```

acsch(X) — повертає гіперболічний арккосеканс для кожного елемента X:

```
>> Y = acsch(1)
Y =0.8814
```

asech(X) — повертає гіперболічний арксеканс для кожного елемента X.

Приклад:

```
>> Y = asech(4)
Y =0 + 1.3181i
```

asinh(X) — повертає гіперболічний арксинус для кожного елемента X:

```
>> Y = asinh (2.456)
Y =1.6308
```

atanh(X) — повертає гіперболічний арктангенс для кожного елемента X:

```
>> X=[0.84 0.16 1.39];
>> atanh (X)
ans =
1.2212 +      0i   0.1614 +      0i   0.9065 + 1.5708i
```

cosh(X) — повертає гіперболічний косинус для кожного елемента X:

```
>> X=[1 2 3];
>> cosh(X)
ans =

1.5431     3.7622     10.0677
```

coth(X) — повертає гіперболічний котангенс для кожного елемента X.

Приклад:

```
>> Y = coth(3.987)
Y =1.0007
```

csch(x) — повертає гіперболічний косеканс для кожного елемента X:

```
>> X=[2 4.678 5;0.987 1 3];
>> Y = csch(X)
Y =
0.2757     0.0186     0.0135
0.8656     0.8509     0.0998
```

sech(X) — повертає гіперболічний секанс для кожного елемента X:

```
>> X=[pi/2 pi/4 pi/6 pi];
>> sech(X)
ans =
0.3985     0.7549     0.8770     0.0863
```

sinh(X) — повертає гіперболічний синус для кожного елемента X:

```
>> X=[pi/8 pi/7 pi/5 pi/10];
>> sinh(X)
ans =0.4029     0.4640     0.6705     0.3194
```

tanh(X) — повертає гіперболічний тангенс для кожного елемента X:

```
>> X=[pi/2 pi/4 pi/6 pi/10];
>>tanh(X)
ans =0.9172     0.6558     0.4805     0.3042
```

Наступний m-файл-скрипт надає можливості отримати графіки ряду гіперболічних функцій:

```
>>pkg load symbolic
>>syms x
>>subplot(2,2,1),ezplot(sinh(x),[-4 4]),xlabel(''),grid on
>>subplot(2,2,2),ezplot(cosh(x),[-4 4]),xlabel(''),grid on
>>subplot(2,2,3),ezplot(tanh(x),[-4 4]),grid on
>>subplot(2,2,4),ezplot(sech(x),[-4 4]),grid on
```

Гіперболічні функції, на відміну тригонометричних, є періодичними. Вибрані для графічного представлення функції надають приклади характерної нелінійності.

В іншому файлі використані команди для побудови графіків ряду зворотних гіперболічних функцій:

```
>>pkg load symbolic
>>syms x
>>subplot(2,2,1),ezplot(asinh(x),[-4 4]),xlabel(''),grid on;
>>subplot(2,2,2),ezplot(acosh(x),[0 4]),xlabel(''),grid on
>>subplot(2,2,3),ezplot(atanh(x),[-1 1]),grid on
>>subplot(2,2,4),ezplot(asech(x),[0 1]),grid on
```

Особливістю даного класу функцій, як зворотний гіперболічний синус і тангенс, є симетричний характер, що дає наближення ряду типової нелінійності.

1.16. Функції округлення та знака

Ряд спеціальних функцій застосовуються для виконання операцій округлення числових даних і аналізу їх знака.

fix(A) — повертає масив A з елементами, заокругленими до найближчого до нуля цілого числа. Для комплексного A дійсні та уявні частини округляються окремо.

Приклади:

```
>> A=[1/3 2/3; 4.99 5.01]
A =
0.3333    0.6667
4.9900    5.0100
>> fix(A)
ans =
0    0
4    5
```

floor(A) — повертає A з елементами, що представляють найближче менше або дорівнює відповідному елементу A ціле число. Для комплексного A дійсні та уявні частини перетворюються окремо.

Приклади:

```
>> A=[-1/3 2/3; 4.99 5.01]
A =
-0.3333  0.6667
4.9900  5.0100
>> floor(A)
ans =
-1 0 4 5
```

ceil(A) — повертає найближче більше або рівне A ціле число. Для комплексного A дійсні та уявні частини округляються окремо.

Приклади:

```
>> a=-1.789;
>> ceil(a)
ans =-1
>> a=-1.789+i*3.908;
>> ceil(a)
ans =
-1.0000 + 4.0000i
```

rem(X,Y) — повертає X = fix(X./Y).*Y, де fix(X./Y) — ціла частина X/Y.

Якщо операнди X та Y мають однаковий знак, функція **rem(X, Y)** повертає той самий результат, що mod(X,Y). Однак (для позитивних X та Y) **rem(-x,y) = mod(-x,y)-y**. Функція **rem** повертає результат між 0 і

sign(X)*abs(Y). Якщо Y=0, функція **rem** повертає NaN. Аргументи X та Y повинні бути цілими числами. Через неточне представлення в комп'ютері чисел з плавучою комою використання речових (або комплексних) вхідних аргументів може призвести до непередбачених результатів:

```
>> X=[25 21 23 55 3];
>> Y=[4 8 23 6 4];
>> rem(X,Y)
ans=1 5 0 1 3
```

round(X) — повертає заокруглені до найближчого цілого елементи масиву X. Для комплексного X дійсні та уявні частини округляються окремо:

```
>> X=[5.675 21.6+4.897*i 2.654 55.8765];
>> round(X)
ans =
6.0000    22.0000 +5.0000i    3.0000    56.0000
```

sign(X) — повертає масив Y тієї ж розмірності, що і X, де кожен із елементів Y дорівнює:

1, якщо відповідний елемент X більше 0;

0, якщо відповідний елемент X дорівнює 0;

-1, якщо відповідний елемент X менше 0. Для ненульових дійсних та комплексних X – sign(X)=X./abs(X).

Приклад:

```
>> X=[-5 21 2 0 -3.7];
>> sign(X)
ans =
-1 1 1 0 -1
```

1.17. Функції комплексного аргументу

Для роботи з комплексними числами та даними в Octave використовуються такі функції:

angle(Z) повертає аргумент комплексного числа в радіанах для кожного елемента масиву комплексних чисел Z. Кути знаходяться в діапазоні [-pi; +pi]. Для комплексного Z модуль та аргумент обчислюються наступним чином: R = abs(Z) — модуль, theta = angle(Z) — аргумент. При цьому формула Z = R.*exp(i*theta) дає перехід від показової форми подання комплексного числа до алгебраїчної.

Приклади:

```
>> Z=3+i*2
Z =
3.0000 + 2.0000i
>> theta = angle(Z)
theta =0.5880
>> R = abs(Z)
R =3.6056
>> Z =R.*exp(i*theta)
Z =
3.0000 + 2.0000i
```

imag(Z) — повертає уявні частини всіх елементів масиву Z:

```
>> Z=[1+i, 3+2i, 2+3i];
>> imag(Z)
ans =
1    2    3
```

real(Z) — повертає речові частини всіх елементів комплексного масиву Z:

```
>> Z=[1+i 3+2i  2+3i];
>> real(Z)
ans =
1 3 2
```

conj(Z) — повертає число, комплексно-сполучене аргументом Z. Якщо Z комплексне, то conj(Z) = real(Z) - i x imag (Z):

```
>> conj(2+3i)
ans=
      2.0000 — 3.0000i
```

Таким чином, вміння виконувати елементарні математичні операції, визначати змінні, використовувати вбудовані функції системи та задавати власні, працювати з масивами даних — це ази роботи у системі Octave.

Контрольні запитання до розділу 1

1. У чому полягає основна відмінність між Matlab та Octave?

2. У чому полягає перевага та недоліки Octave порівняно з Matlab?

3. Що спільного між Matlab та Octave?

4. Який синтаксис команд у Octave, що формують вектор-рядок, вектор- стовпець та матрицю?

5. Запишіть за правилами Octave вираз $y = \sin(\omega t + \varphi)$.

6. Запишіть за правилами Octave вираз $y = e^{x-2}(5x - 3)$.

7. Які додаткові параметри має команда **format** та на що ця команда впливає?

8. Які функції в системі Octave, що виконують роботу з датою та часом?

9. Якими методами і командами в Octave можливо задавати масиви та матриці?

10. Які функції виконують команди **rand, randn** та **magic**?

11. Які операції над масивами та матрицями можливо робити в системі Octave та вкажіть функції, які для цього використовують?

12. Запишіть функцію для визначення максимального за модулем елемента масиву.

13. Задайте дві матриці таким чином, щоб їх можна було складати і знайти їхню суму.

14. Задайте дві матриці таким чином, щоб їх можна було поелементно перемножити, і знайти їх поелементний добуток.

15. Які тригонометричні функції розглянуто та як вони записуються в системі Octave?

16. У чому особливість визначення кутів тригонометричних функцій в системі Octave?

17. Складіть таблицю значень функції $y = \sin(x)$, x змінюється від 0 до 90^o з кроком 5^o.

18. Поясніть призначеність наступних функцій в системі Octave: **exp, log, log10, log2, sqrt**?

Розділ 2. Робота з графічними функціями в Octave

Для візуалізації вхідних даних та отриманих результатів GNU Octave має у своєму розпорядженні потужні графічні бібліотеки. В Octave є набір різноманітних засобів для графічного відображення даних. Зокрема, програма дозволяє відображати графіки залежностей для змінних прямо з робочого простору. Побудову та зміну графіків можна здійснювати за допомогою спеціальних інструментів або за допомогою команд. Можливо як відображення графіків залежності від однієї змінної на координатній площині, так і побудова поверхонь для функцій від двох змінних у тривимірному просторі.

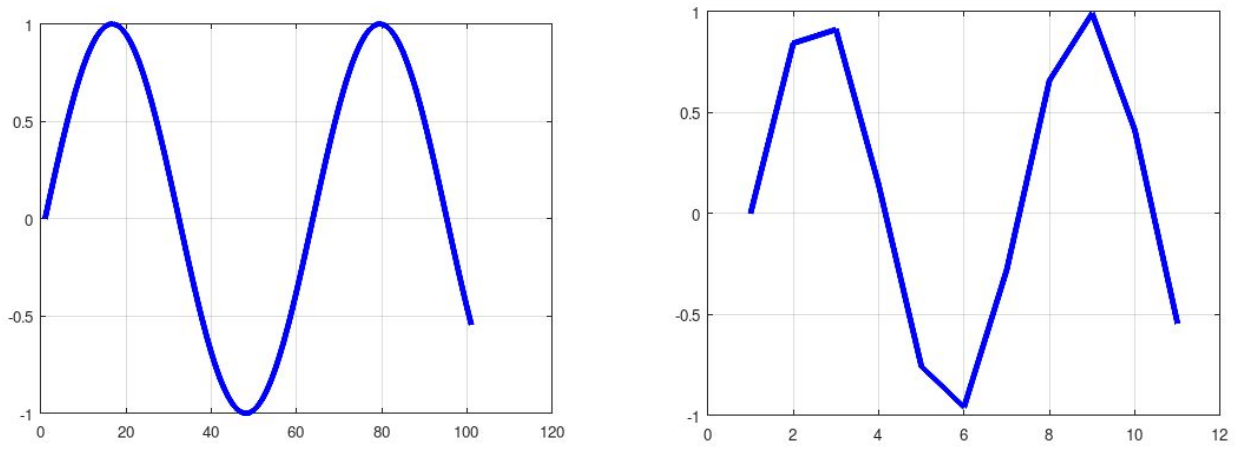
2.1. Засоби двовимірної графіки в Octave

У режимі безпосередніх обчислень доступні всі можливості системи. Широко використовується, наприклад, побудова графіків різних функцій, що дають наочне уявлення про їхню поведінку в широкому діапазоні зміни аргументу. При цьому графіки будуються в окремих вікнах, що масштабуються і переміщуються.

Розглянемо найпростіший приклад — побудова двовимірного графіка синусоїди. **Слід пам'ятати, що Octave (як і інші СКМ) будує графіки функцій ряду точок, з'єднуючи їх відрізками прямих, тобто здійснюючи лінійну інтерполяцію функції в інтервалі між суміжними точками.** Задамо інтервал зміни аргументу *x* від 0 до 10 з кроком 0.1. Для побудови графіка достатньо спочатку задати вектор x=0:0.1:10, а потім використовувати команду побудови графіків plot(sin(x)) (рис. 2.1).

Пояснення щодо інтервалу зміни незалежної змінної *x* від 0 до 10 з кроком 0.1.полягає у наступному. Команда **plot** будує неправдивий графік функції sin(x): лише задане числом елементів

вектора *x* з кількістю заданих точок. Ці точки потім просто з'єднуються відрізками прямих, тобто здійснюється кусково-лінійна інтерполяція графіка даних. При 100 точках отримана крива оком сприймається як цілком плавна (рис.2.1, *а*), але при 10-20 точках вона виглядатиме з відрізків прямих (рис.2.1, *б*).



а *б*

Рисунок 2.1 — Приклад побудови графіка синусоїди

Графіки в Octave розташовані в окремих вікнах, які називаються графічними вікнами. З першого погляду видно відмінності графічного вікна, показаного на рис.2.1, від командного вікна Octave. У головному меню вікна з'явилася позиція Tools (Інструменти), що дозволяє вивести чи приховати інструментальну панель, розташовану у верхній частині вікна графіка на рис. 2.1. Засоби цієї панелі (розгляд наведено у наступній частині посібнику) дозволяють легко керувати параметрами графіків і експортувати їх до інших додатків.

При застосуванні ще однієї команду **plot** після її виконання графік нової функції буде розміщено поверх попереднього. Ця особливість пов'язана не з роботою самої функції **plot**, а з роботою графічного вікна. Тому всі функції, пов'язані з побудовою графіків,

застосовуються зазвичай у парі з функціями, що визначають режим роботи графічного вікна.

Спробуємо побудувати графіки відразу трьох функцій: sin(x), cos(x) і sin(x)/x. Насамперед зазначимо, що ці функції можуть бути позначені змінними, які не мають явної вказівки аргументу у вигляді y(x):

```
>>y1=sin(x); y2=cos(x); y3=sin(x)/x;
```

Така можливість обумовлена тим, що це змінні є векторами, як і змінна x. Тепер можна використовувати одну із низки форм команди **plot**:

plot(a1,f1,a2,f2,a3,f3,...),

де al, a2, a3,.„ — вектори аргументів функцій (у нашому випадку всі вони – *x*), а f1, f2, f3,... — вектори значень функцій, графіки яких будуються у одному вікні.

В даному випадку для побудови графіків зазначених функцій необхідно записати наступне:

```
>> plot(x,y1,x,y2,x,y3)
```

Очікується, що Octave у цьому випадку побудує, як завжди, точки графіків цих функцій та з'єднає їх відрізками ліній. Але при виконанні цієї команди отримаємо лише два графіки та крапки замість третього. Не виключений навіть збій у роботі програми. Причина цього казусу вже обговорювалась у попередньому прикладі: при обчисленні функції **y3=sin(x)/x,** якщо **x** є масив (вектор), то не можна використовувати оператор звичайного поділу **/**.

Цей приклад ще раз наочно вказує на те, що чисто поверхове застосування навіть такої потужної системи, як Octave, іноді призводить до прикрих збоїв. Для отримання графіка необхідно обчислювати відношення sin(x) до x за допомогою оператора поелементного поділу масивів **./** (рис.2.2).

В Octave побудовано графіки всіх трьох функцій, при цьому у вікні командного режиму з'явилося попередження про поділ на 0 — у момент, коли x=0. Це означає, що команда **plot** «не знає» про те, що невизначеність **sin(x)/x=0/0** можна усунути і дає 1. Це недолік практично всіх систем для чисельних обчислень.

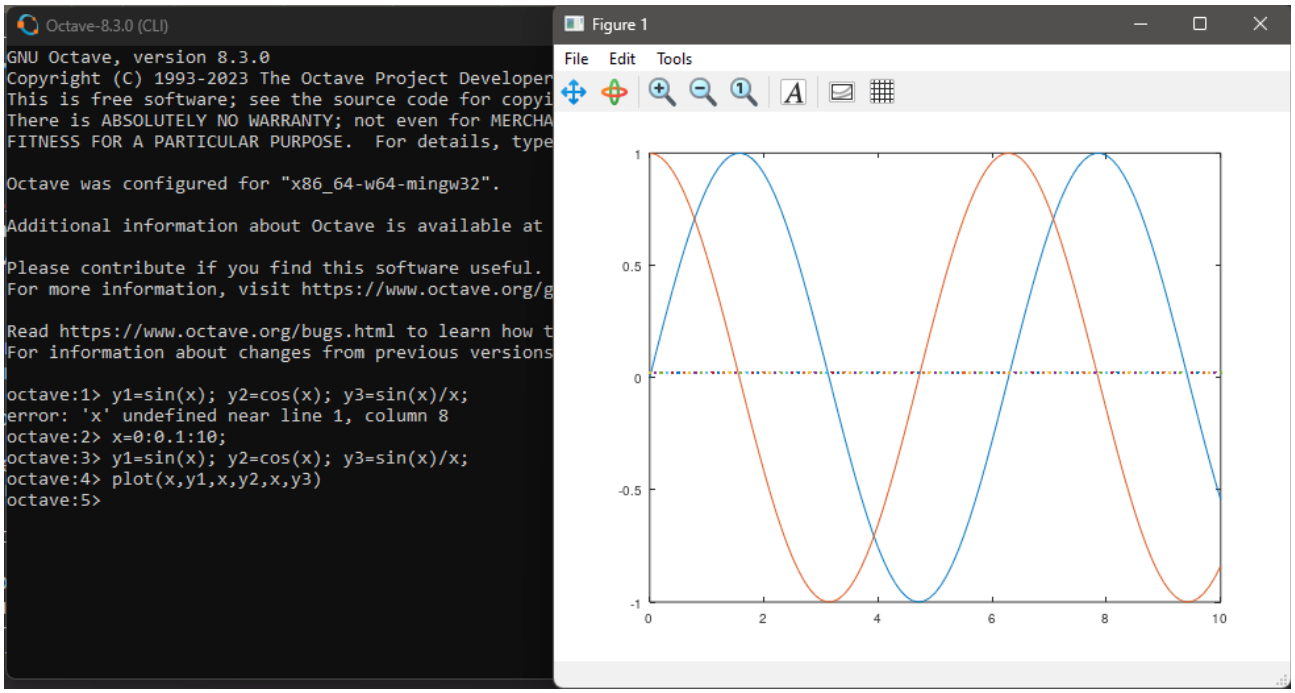


Рисунок 2.2 — Візуалізація графіків трьох функцій

Зрозуміло, Octave має засоби для побудови графіків таких функцій, як **sin(x)/x**, які мають усувні невизначеності. Покажемо, як це робиться за допомогою іншої графічної команди — **fplot**:

fplot('f(x)', [xmin xmax]).

Ця функція дозволяє будувати функцію, задану в символьному вигляді, в інтервалі зміни аргументу від xmin до xmax без фіксованого кроку зміни x. Один із варіантів її застосування демонструє рис. 2.3:

```
>> clear all;
>> fplot(sin(x)/x,[-15, 15]), grid on;
```

У процесі обчислень з'являється попередження про помилку (поділ на 0). Але графік будується правильно, при x=0 sinx/x=1.

Зверніть також увагу на дві команди: **clear** (очистити) – очищення графічного вікна і **grid on** (сітка) — включення відображення сітки, яка будується пунктирними лініями.

На рис. 2.3 представлено також меню File (Файл) вікна графіки, що містить типові файлові операції. Однак вони не стосуються файлів документів, а стосуються файлів графіків. Зокрема, можна присвоювати ім'я рисунків з графіками для запису на диск.

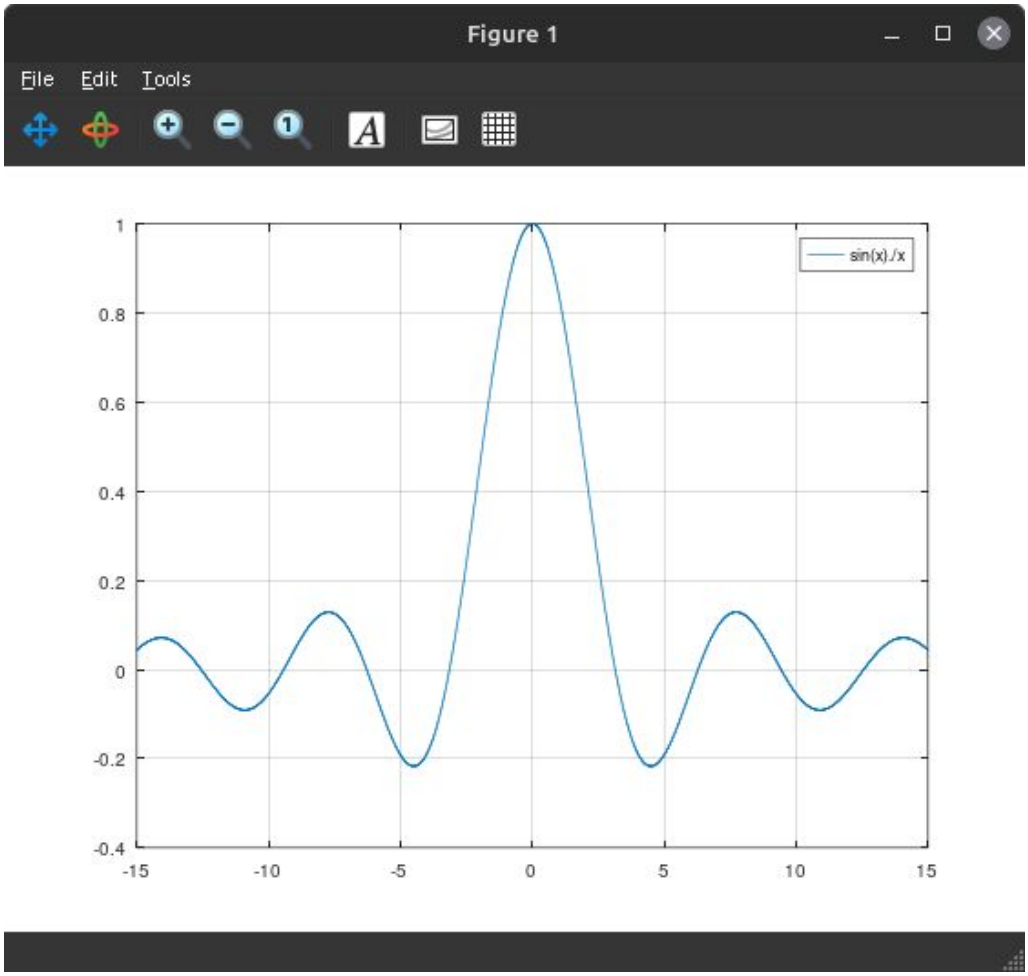


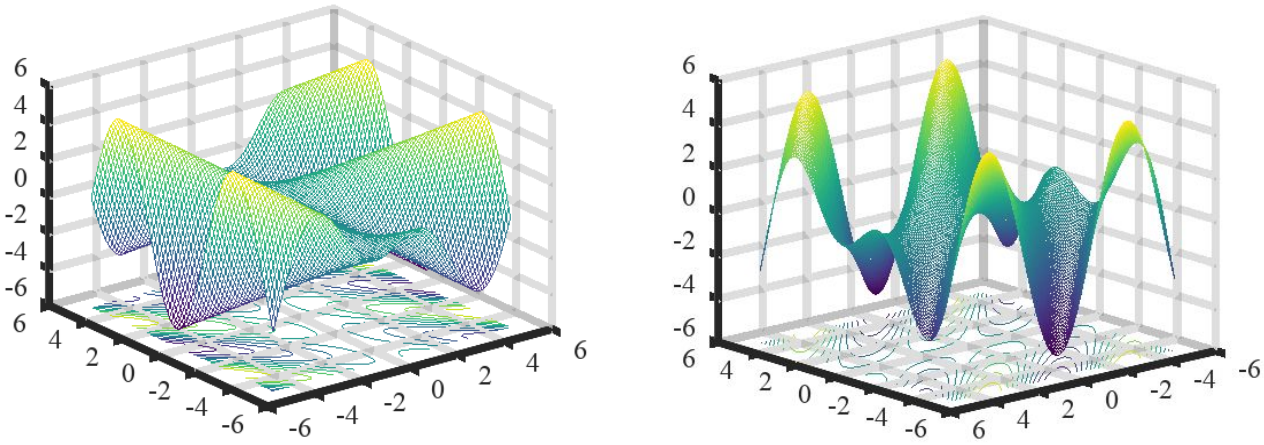
Рисунок 2.3 — Побудова графіка sin(x)/x функцією **fplot**

Так само просто забезпечується побудова графіків складних поверхонь. Потрібно лише знати, якою командою реалізується той чи інший графік. Наприклад, для побудови графіка поверхні та її проєкції у вигляді контурного графіка на площину під поверхнею достатньо використовувати такі команди (див. розділ 2.8):

```
>> [X,Y]=meshgrid(-5:0.1:5);
>> Z=X.*sin(X+Y);
>> meshc(X,Y,Z)
```

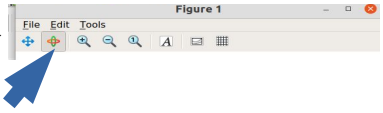
Реалізація побудови тривимірних графіків поверхні з забарвленням, заданою вектором Z, та її проєкцією на площину XY, функцією **meshc(X,Y,Z)**, показано на рис. 2.4.

Octave надає можливість повертати побудований графік (рис.2.4, *а*) та спостерігати під різними кутами (рис.2.4, *б*).



а *б*

Рисунок 2.4 — Графік поверхні та проєкції на площину

Для обертання графіка достатньо активізувати останню праворуч кнопку панелі інструментів із зображенням пунктирного кола зі стрілкою. Тепер, ввівши курсор миші в область графіка і натиснувши ліву кнопку миші, можна круговими рухами змусити графік обертатися разом з паралелепіпедом, що його обрамляє. Також на панелі інструментів, натиснувши на піктограму 

, можна виконати обертання графіка.

Приклад побудови популярного графіка в системі Octave — сомбреро представлено на рис.2.5. Для цього, ввівши команду sombrero, отримаємо відповідний графік.

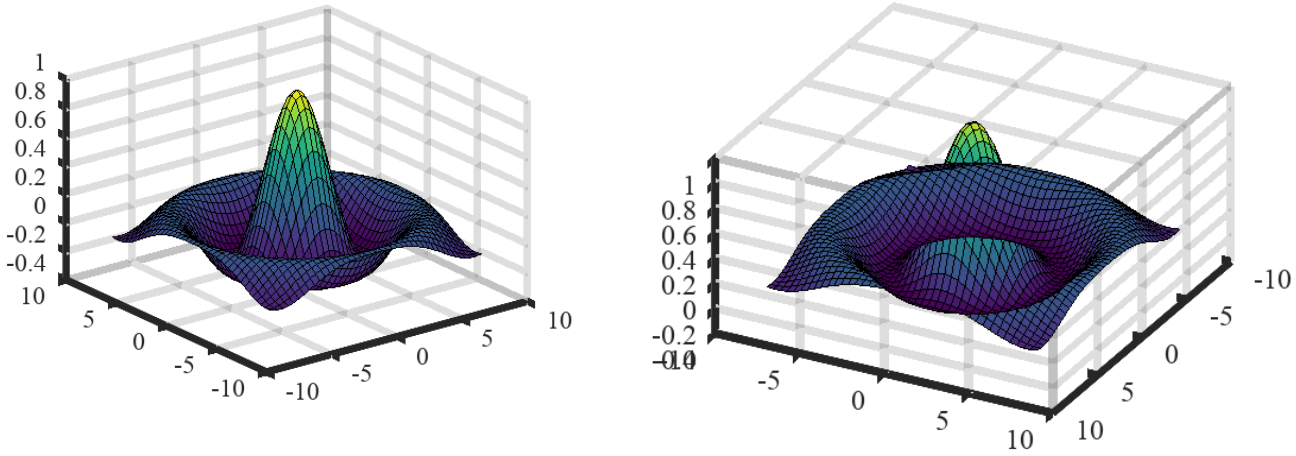


Рисунок 2.5 — Результати обертання тривимірної фігури мишею

В Octave можна обертати і двовимірні графіки, спостерігаючи поворот площини, в якій побудовані. Жодного програмування таке обертання не вимагає.

2.2. Редагування та форматування графіків

Для редагування графіка потрібно натиснути пункт **Tools** (Sycnhevtynb) контекстного меню. В цьому меню можна керувати графіком за допомогою контекстного меню, вмикати, вимикати сітку, масштабувати та переміщувати графік. Вигляд цього меню при курсорі, розташованому в області тривимірного графіка поза побудованими, як приклад, тривимірними графічними об'єктами, показаний на рис. 2.6. За допомогою миші можна візуалізувати значення на графіку. При натисканні на кнопку з літерою *A* та виділенні області на графіку відкриється меню анотації (див. рис. 2.6). Це надає можливості наносити будь які пояснювальні написи (кнопка з літерою *A*) і т. п. на графік.

Місце напису фіксується виділенням мишею. На рис. 2.7 показано відформатований графік із текстовим блоком, створеним таким чином у лівій верхній частині поля графіка.

Також показано контекстне меню правої клавіші миші, що пояснює вибір розміру символів напису (та інші можливості цього меню). Нагадуємо, що це меню з'являється при натисканні правої кнопки миші на заданому об'єкті. У цьому меню є всі команди, доступні для цього об'єкту в даній ситуації.

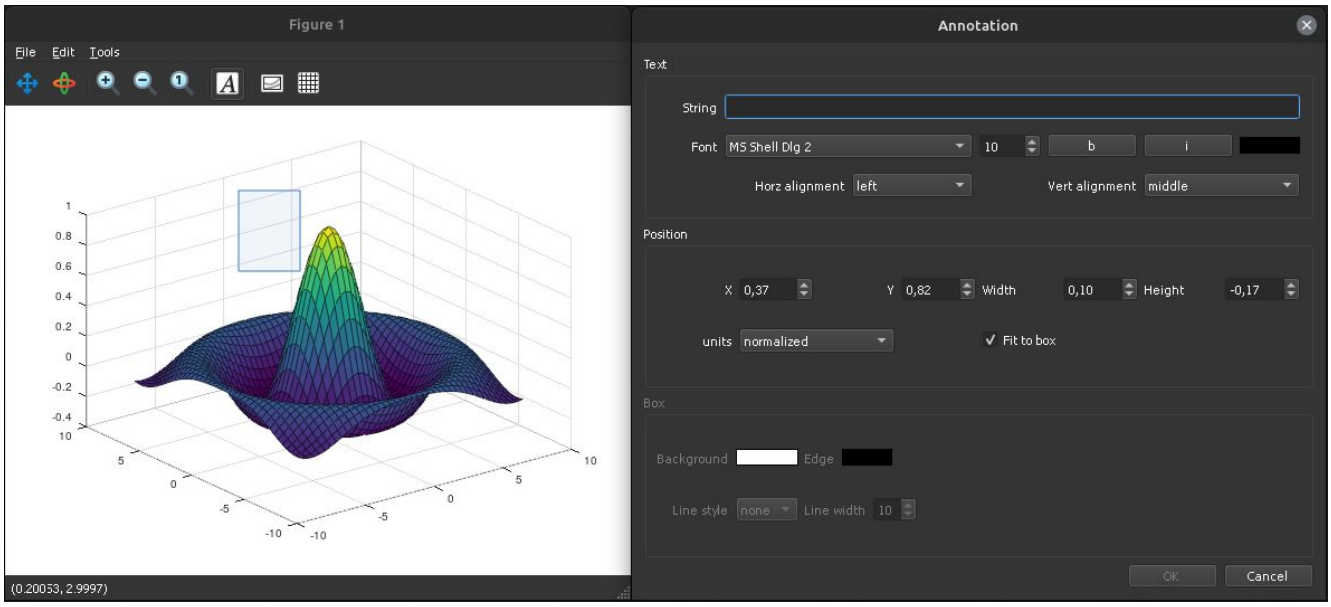


Рисунок 2.6 — Додавання тексту на графік

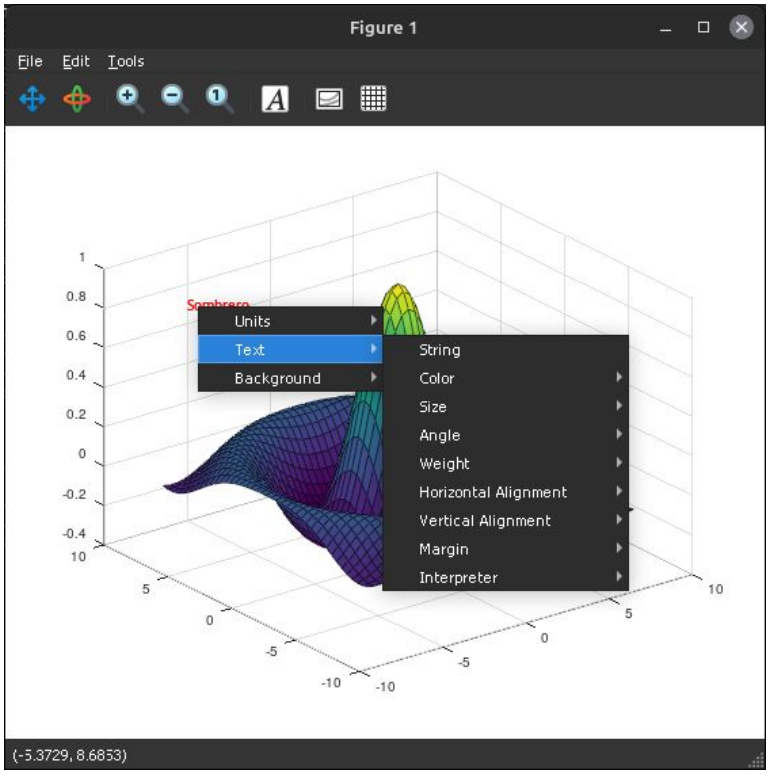


Рисунок 2.7 — Редагування нанесеного напису на побудованому графіку

Графіки в системі Octave будуються просто оманливо. Пов'язано це з тим, що багато характеристик графіків встановлені за умовчанням. До таких властивостей відносяться виведення або приховування координатних осей, положення їх центру, колір лінії графіка, її товщина і т. п. Властивості та вид графіків можна змінювати в широких межах за допомогою відповідних параметрів команд графіки. Однак це потребує розуміння деталей мови програмування та дескрипторної графіки системи Octave.

У новій версії Octave для зміни властивостей графіків (їх форматування) використовуються принципи візуального контролю над стилем (видом) всіх об'єктів графіків. Це дозволяє легко, просто і наочно надати графікам належного вигляду перед записом їх у вигляді файлів на диск. У цій частині реалізовані окремі принципи візуально-орієнтованого програмування графічних засобів.

Додатково можна змінити розміри графіка (див. меню **Tools** (Інструменти) та його команди **Zoom In** (Збільшити) та **Zoom Out** (Зменшити)), почати поворот графіка мишею (команда **Rotate 3D**). Слід зазначити, що існують доступні можливості форматування графіків і програмним методом, шляхом завдання відповідних графічних команд, параметрів і примітивів. Наприклад, команда **text(x,y, 'legend')** дозволяє задати напис 'legend' з початком, що має координати (x, y). Якщо після першого апострофа перед текстом помістити параметр **\leftarrow**, напис (легенда) з'явиться після стрілки з вістрям, зверненим вліво. Аналогічно параметр **\rightarrow** після напису задає виведення стрілки після напису з вістрям, зверненим праворуч. Ця можливість дозволяє помічати як криві, так і окремі точки на них. Можливе також застосування команди **legend('s1', 's2',...)**, що виводить легенду звичайного виду – відрізки ліній графіків з написами 's1', 's2' і т. п. (рис.2.8).

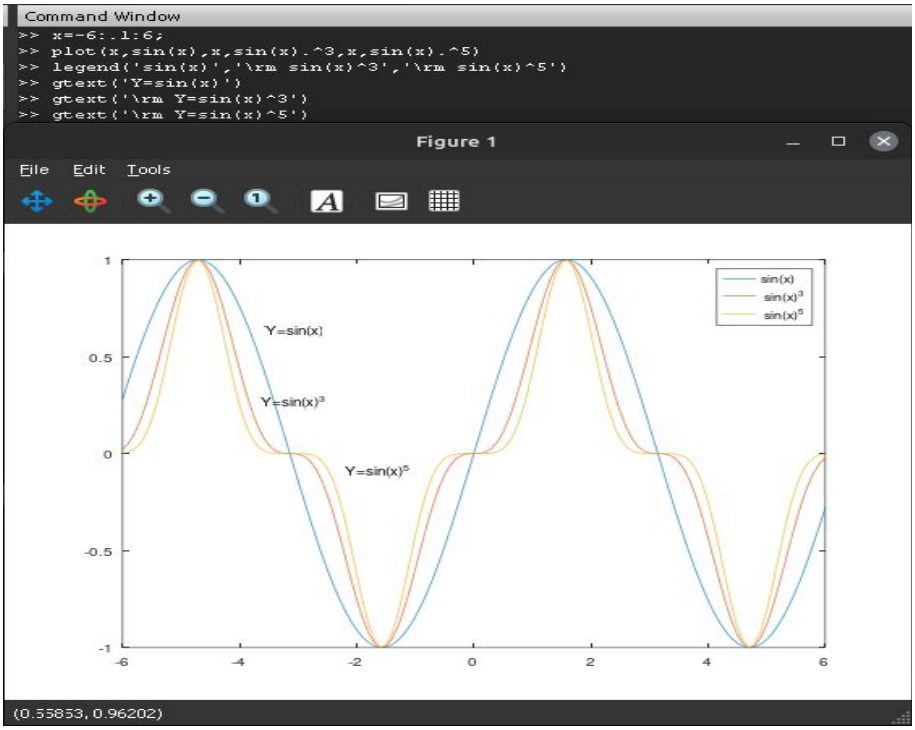


Рисунок 2.8 — Подання двовимірного графіка з позначенням залежностей та легендою

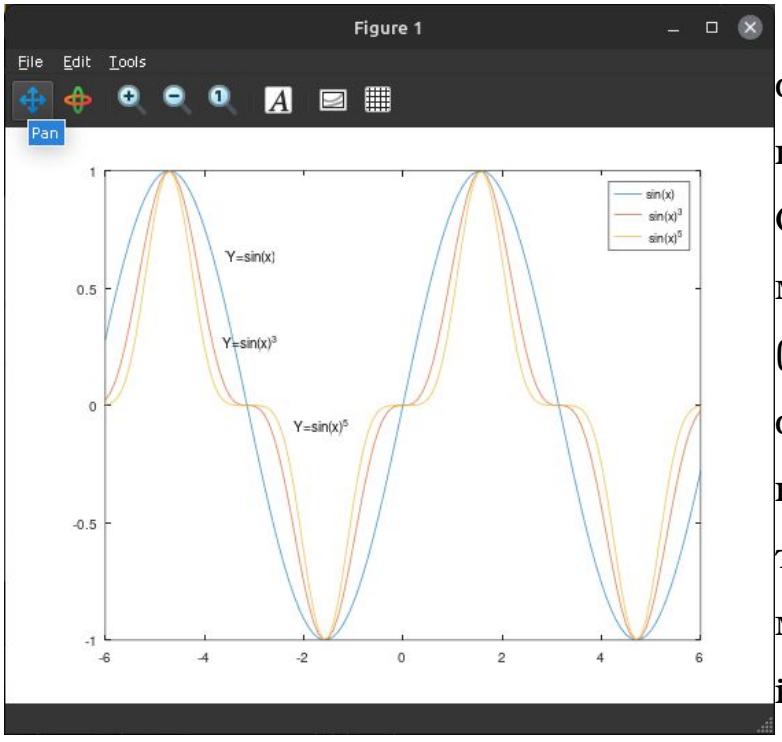


Рисунок 2.9 — Приклад переміщення графіка

Зазвичай графік займає фіксоване становище у центрі графічного вікна. Однак у вікні графіків, у меню зверху є команда **Pan** (Панорамування), що знімає фіксацію положення координатних осей графіка та дозволяє рухати його мишею разом з осями. Це ілюструє рис. 1.9. При переміщенні графіка його легенда та колірна діаграма залишаються незмінними.

2.3. Побудова графіків у декартовій системі координат

Декартова (прямокутна) система координат задається двома перпендикулярними прямими, — осями координат. Горизонтальна пряма X - вісь абсцис,вертикальна Y - вісь ординат. Точка перетину - початком координат. Чотири кути, утворені осями координат, є координатними кутами. Положення точки у прямокутній системі координат визначається значенням двома величинами — координатами точки. Якщо точка має координати x і y, то x — абсциса точки, y — ордината. Рівняння, що зв'язує координати x та y, є рівнянням лінії, якщо координати будь-якої точки цієї лінії задовольняють йому. Величина y — функція змінної величини x, якщо кожному з тих значень, які може набувати x, відповідає одне чи кілька певних значень y. При цьому змінна величина x називається аргументом функції **y = f(x)**. Функція вважається заданою, якщо для кожного значення аргументу існує відповідне значення функції.

Найчастіше використовують такі способи задання функцій: табличний — числові значення функції вже задані та занесені в таблицю;

графічний — значення функції задані за допомогою лінії (графіка), у якої абсциси зображують значення аргументу, а ординати - відповідні значення функції;

аналітичний — функція задається однією чи кількома формулами (рівняннями); при цьому, якщо залежність між x і y виражена рівнянням, дозволеним щодо y, то говорять про явно задану функцію, в іншому випадку функція вважається неявною.

Сукупність всіх значень, які може приймати в умовах поставленої задачі аргумент x функції **y = f(x)**, називається областю визначення цієї функції. Сукупність значень y, які приймає функція f(x), називається безліччю значень функції.

При графічному зображенні функцій задаються координати x і y, що визначають вузлові точки функції f(x). Ці точки з'єднуються одна з одною відрізками прямих, тобто під час побудови графіка, як зазначено у 2.1, здійснюється лінійна інтерполяція для проміжних точок. Оскільки Octave — матрична система, сукупність точок y(x) визначається векторами X і Y однакового розміру.

Команда **plot** служить для побудови графіків функцій у декартовій системі координат. Ця команда має низку параметрів, що розглядаються нижче.

plot (X, Y) — будує графік функції y(x), координати точок (x,y) якої беруться з векторів однакового розміру Y і X. У разі застосування X або Y — матриці будується сімейство графіків за даними, що містяться в колонках матриці.

Наведений нижче приклад ілюструє побудову графіків двох функцій — sin(x) та cos(x) (рис. 2.10), значення функції яких містяться в матриці Y, а значення аргументу x зберігаються у векторі X:

```
>> x=[0 1 2 3 4 5];
>> Y=[sin(x); cos(x)];
>> plot(x,Y)
```

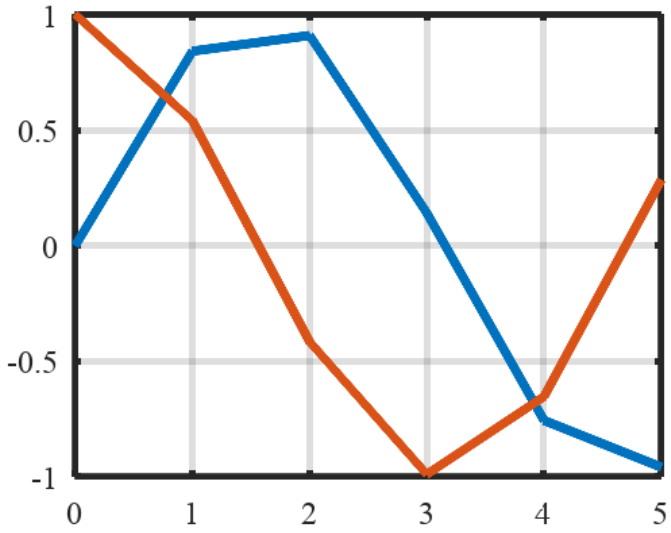


Рисунок 2. 10 — Графіки двох функцій у декартовій системи координат

В даному випадку чітко видно, що графік складається з відрізків.

Для забезпечення функції, що відображається, у вигляді гладкої кривої, необхідно збільшити кількість вузлових точок. Розташування може бути довільним.

Команда **plot(Y)** — будує графік y(i), де значення беруться з вектора Y, а і являє собою індекс відповідного елемента. Якщо Y містить комплексні елементи, виконується команда **plot (real(Y). imag(Y))**. В інших випадках уявна частина даних ігнорується.

Приклад використання команди **plot(Y)**:

```
>> x=-2*pi:0.02*pi:2*pi;
>> y=sin(x)+i*cos(3*x);
>> plot(y)
```

Відповідний графік показано на рис. 2.11.

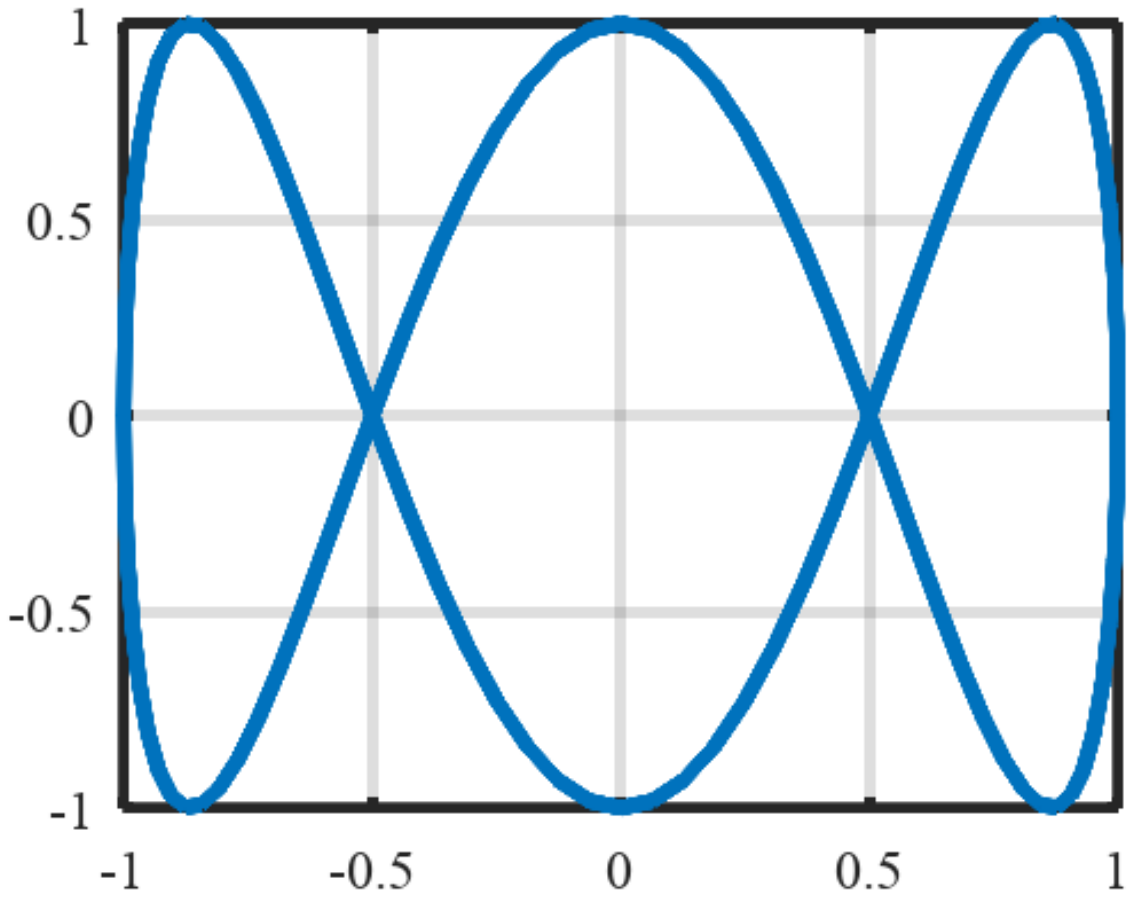


Рисунок 2.11 — Графік функції, що представляє вектор Y з комплексними елементами

Команда **plot(X,Y,S)** — аналогічна команді plot(X,Y), але тип лінії та її колір графіка можна задавати за допомогою рядкової константи S (табл.2.1).

Таблиця 2.1 — Значення константи S

Колір лінії	
y	Жовтий
м	Фіолетовий
c	Блакитний
r	червоний
g	Зелений
в	Синій
w	Білий
k	Чорний
Тип маркеру	
.	Крапка
o	Окружність
x	Хрест
+	Плюс
*	Зірочка
s	Квадрат
d	Ромб
v	Трикутник (вниз)
^	Трикутник (вгору)
<	Трикутник (ліворуч)
>	Трикутник (вправо)
p	П'ятикутник
h	Шестикутник
Тип лінії	
-	Суцільна
:	Короткий пунктир
-.	Штрих-пунктир
--	Довгий пунктир

Таким чином, за допомогою рядкової константи S можна змінювати колір лінії, представляти вузлові точки різними відмітками (точка, коло, хрест, трикутник з різною орієнтацією вершини тощо) і змінювати тип лінії графіка.

Команда **plot (X1,Y1, S1, X2, Y2, S2, X3, Y3, S3,...)** — ця команда будує на одному графіку ряд ліній, представлених даними виду (X..Y..S.), де X і Y – вектори чи матриці, а S – строкові. За допомогою такої конструкції можлива побудова, наприклад, графіка функції лінією, колір якої відрізняється від кольору вузлових точок. Так, якщо треба побудувати графік функції лінією синього кольору з червоними точками, спочатку треба задати побудову графіка з точками червоного кольору (без лінії), наступний крок - побудова графіка з лінією синього кольору (без точок).

За відсутності вказівки на колір ліній та точок відбувається автоматичний вибір з таблиці кольорів (білий виключається). При кількості ліній більше шести вибір кольорів повторюється. Для монохромних систем лінії вирізняються стилем.

Розглянемо приклад побудови графіків трьох функцій з різним стилем уявлення кожної з них:

```
>> x=-2*pi:0.1*pi:2*pi;
>> y1=sin(x);
>> y2=sin(x).^2;
>> y3=sin(x).^3;
>> plot(x,y1,'-m',x,y2,'-.+r',x,y3,'--ok')
```

Графіки функцій цього прикладу показано на рис. 2.12. Графік функції **y1** будується суцільною фіолетовою лінією, графік **y2** — штрих-пунктирною лінією з точками у вигляді знака «плюс» червоного кольору, графік **y3** — штриховою лінією з кружками чорного кольору.

При вивченні руху точки на площині Octave дозволить побудувати графік руху та простежити за рухом. Побудувати анімаційний ролик можна за допомогою функції **comet(x, y)**, яка дозволить побачити рух точки вздовж кривої **y(x)** на площині. Для руху точки на площині вздовж синусоїди достатньо запровадити команди:

```
>>x=0: pi/30:6*pi;
>>y=sin(x);
>>comet(x, y)
```

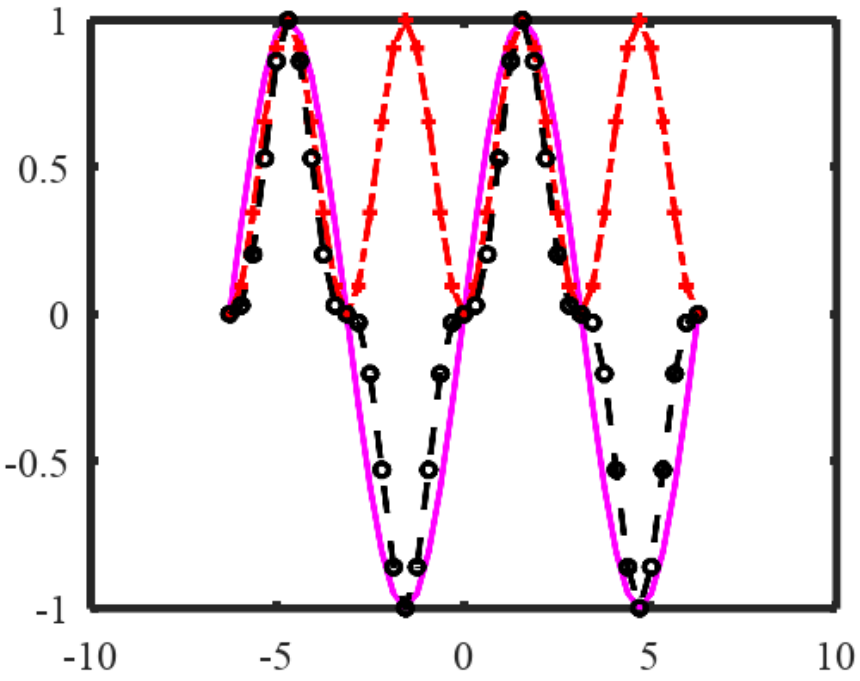


Рисунок 2.12 — Графіки трьох функцій на одному рисунку з різним стилем ліній

Для побудови графіків функцій зі значеннями x і y, що змінюються у широких межах, часто використовуються логарифмічні масштаби. Розглянемо команди, які застосовуються у таких випадках.

Команда **loglog(...)** — синтаксис команди аналогічний раніше розглянутому функції **plot(...)**. Логарифмічний масштаб використовується для координатних осей X і Y. Нижче наведено приклад застосування цієї команди:

```
>> x=logspace(-1,3);
>> loglog(x,exp( sin (x)))
>> grid on;
```

На рис.2.13 представлено графік функції **exp(sin(x))** у логарифмічному масштабі. Зверніть увагу на те, що командою **grid on**

будується координатна сітка. Зауважимо, що команда **grid off** прибирає сітку з графіка.

Нерівномірне розташування ліній координатної сітки свідчить про логарифмічний масштаб осей.

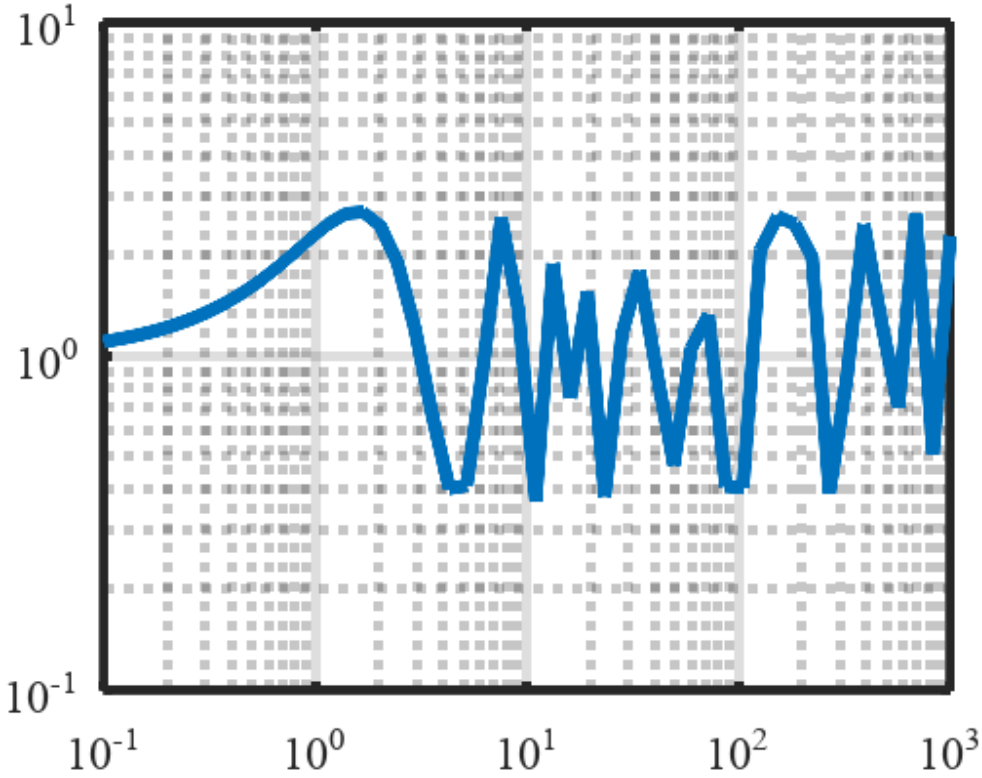
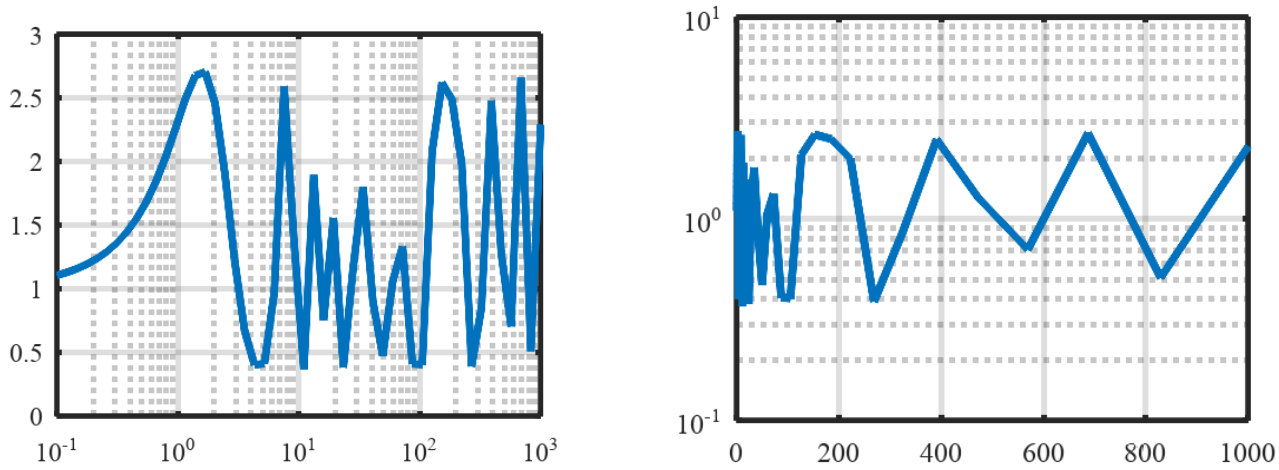


Рисунок 2.13 — Графік функції **exp(sin(x))** у логарифмічному масштабі



а *б*

Рисунок 2.14 — Графік функції **exp(sin(x))** у напівлогарифмічному масштабі

У деяких випадках застосовується напівлогарифмічний масштаб графіків: по одній осі задається логарифмічний масштаб, а за іншою — лінійний. Для побудови графіків функцій у напівлогарифмічному масштабі використовуються наступні команди:

semilogx(...) — будує графік функції у логарифмічному масштабі (основа 10) по осі X та лінійному по осі Y (рис.2.14, *а*);

semilogy(...) — будує графік функції в логарифмічному масштабі осі Y і лінійному осі X (рис.2.14, *б*).

Запис параметрів (...) виконується за аналогією до функції **plot(...)**.

2.4. Діаграми та гістограми в Octave

Гістограми та стовпчаста діаграма відображають значення прямокутниками (стовпцями). Їх використовують для порівняння та візуалізації відмінностей у значеннях за категоріями. Відмінність гістограм та стовпчастих діаграм полягає у використанні різних видів шкал у змінних. Гістограми використовуються для змінних з безперервними шкалами (наприклад, рік народження), а стовпчасті діаграми — для дискретних (наприклад, Київ, Харків, Львів). Для гістограм по осі X розташовуються значення даних, а осі Y відображається щільність розподілу. Для стовпчастих діаграм по осі X розташовуються категорії, а по осі Y - кількість значень, що відповідає іншій змінній для цього значення категорії. Висота стовпців пропорційна значенню осі Y.

Візуальна різниця між гістограмами та стовпчастими діаграмами полягає у наявності або відсутності простору між стовпцями значень. У бар-чартах є невелика відстань між стовпцями, у гістограмі — відсутня. Гістограма ділить безперервне значення

змінної на умовні групи. Стовпчаста діаграма використовується для змінних, які вже розбиті на групи (категоріальні змінні).

Стовпцеві діаграми відбивають зміст деякого вектора V. У цьому кожен елемент вектора представляється стовпцем, висота якого пропорційна значенню елемента. Стовпці нумеруються і масштабуються по відношенню до максимального значення найвищого стовпця. Для побудови такого графіка застосовується команда **bar(y)** (рис. 2.15).

Функція **bar(y)** виводить елементи масиву **y** у вигляді гістограми. Масив x виступає у ролі масиву номерів елементів масиву y.

Функція **bar(x, y)** виводить гістограму елементів масиву **y** у вигляді стовпців на позиціях, що визначаються масивом **x**, елементи якого мають бути упорядковані у порядку зростання.

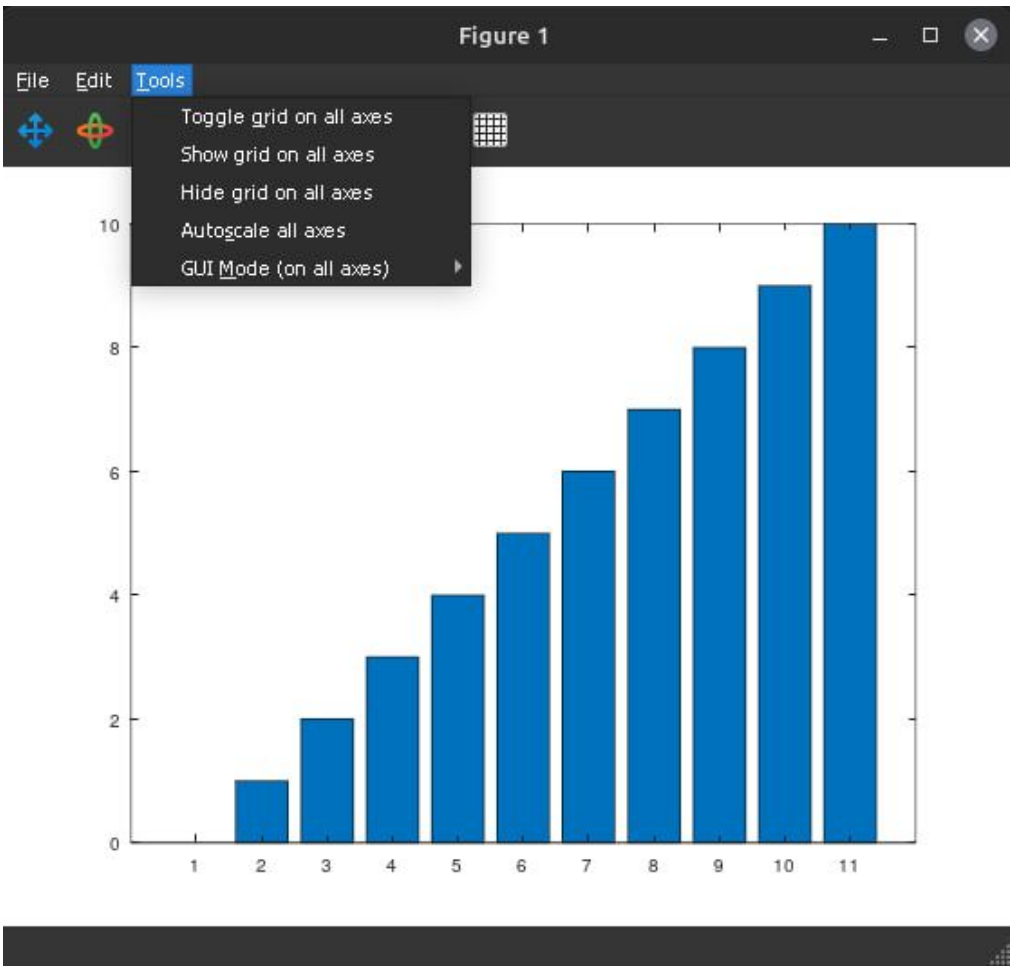


Рисунок 2.15 — Приклад стовпцевої діаграми

Побудова стовпцевих діаграм визначається наступними командами:

bar(x, Y) — стовпцевий графік елементів вектора Y (або групи стовпців для матриці Y) зі специфікацією положення стовпців, заданою значеннями елементів вектора x, які повинні йти монотонно у зростальному порядку;

bar(Y) — будує графік значень елементів матриці Y, але для побудови графіка використовується вектор x=1:m;

bar(x,y,width) або **bar(y,width)** — команда аналогічна раніше розглянутим, але зі специфікацією ширини стовпців (при width>1 стовпці в одній і тій же позиції перекриваються). За замовчуванням встановлено width= 0.8.

Приклад побудови з коментарем (позначено символом %) стовпцевої діаграми матриці розміром 12×3 наводиться нижче (рис.2.16):

```
>> % Стовпцева діаграма з вертикальними стовпцями
>> subplot(2,1,1),bar(rand(12,3),'stacked'),colormap(cool)
```

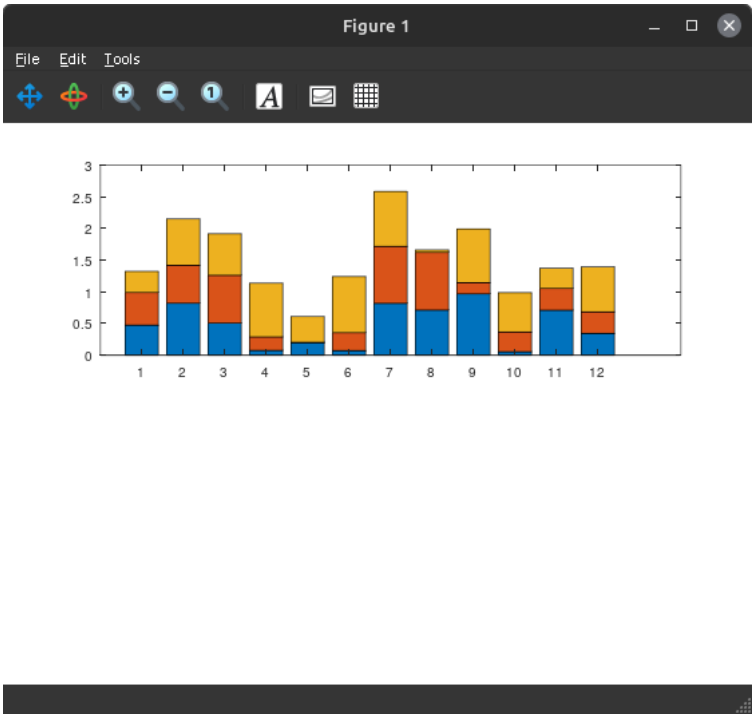


Рисунок 2.16 — Приклад діаграми з вертикальними стовпцями

Крім команди **bar(...)** існує аналогічна їй за синтаксисом команда **barh(...)** для побудови стовпцевої діаграми з горизонтальним розташуванням стовпців (рис. 2.17):

>> subplot(2,1,1),barh(rand(5,3),'stacked'), colormap(cool)

Рисунок 2.17 — Стовпцева діаграма з горизонтальними стовпцями

Вибір розташування стовпців визначає користувач, який використовує ці команди для представлення даних.

Класична гістограма характеризує числа попадань значень елементів вектора Y в інтервалів M з поданням цих чисел у вигляді стовпцевої діаграми. Для отримання даних для гістограми служить функція **hist**:

N=**hist(Y)** — повертає вектор чисел попадань для 10 інтервалів, що вибираються автоматично. У разі застосування Y — матриці, видається масив даних про кількість попадань для кожного з її стовпців;

N=**hist(Y,M)** — використовується інтервал M (M — скаляр);

N=**hist(Y,X)** — повертає числа попадань елементів вектора Y у інтервали, центри яких задані елементами вектора X;

[N,X]=**HIST(...)** — повертає числа попадань в інтервали та дані про центри інтервалів.

У наступному прикладі наведено сценарій побудови гістограми для 1000 випадкових чисел з виведенням вектора з даними про числа їх влучень в інтервали, задані вектором x (рис.2.18):

```
>> x=-3:0.2:3;
>> y=randn(1000,1);
>> hist(y,x)
>> h=hist(y,x)
h =
Columns 1 through 12
0 0 3 7 8 9 11 23 33 43 57 55
Columns 13 through 24
70 62 83 87 93 68 70 65 41 35 27 21
Columns 25 through 31
12 5 6 3 2 1 0
```

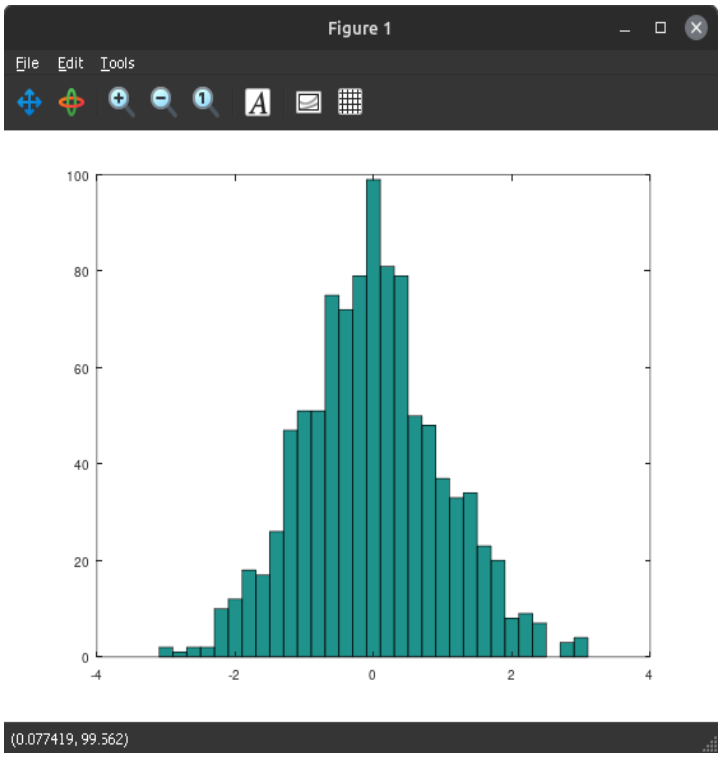


Рисунок 2.18 — Приклад побудови гістограми

Неважко зауважити, що розподіл випадкових чисел близький до нормального закону. Збільшивши їх кількість, можна спостерігати ще більшу відповідність цьому закону.

Застосовуються також спеціальні типи гістограм. Так, діаграми Ганта використовуються для побудови діаграм проектів та графіків. Використання кольорових смуг різної довжини відображає не тільки дати початку та закінчення проекту, але й важливі події, завдання, контрольні точки та їх часові рамки. Сучасні діаграми Ганта можуть ілюструвати взаємозв'язку залежностей між видами діяльності.

Кутові гістограми **у полярній системі** координат (див. розділ 2.6) знаходять застосування в індикаторах станцій радіолокації, для відображення «троянд» вітрів і при побудові інших спеціальних графіків. Для цього використовується ряд команд типу **rose(...):**

rose(PHI) — будує кутову гістограму для 20 інтервалів за даними вектора PHI;

rose(PHI,N) — будує кутову гістограму для N інтервалів у межах кута від 0 до 2*pi за даними вектора PHI;

rose(PHI,X) — будує кутову гістограму за даними вектора PHI зі специфікацією інтервалів, вказаною у векторі X.

Наступний приклад ілюструє застосування команди rose з побудовою графіка (рис.2.19):

```
>> rose (1:100,12)
```

Функція **H=rose(...)** будує графік і повертає вектор дескрипторів графічних об’єктів, а функція **[T,R]=rose(...)** сама собою графік не будує, але повертає вектори T і R, які потрібні команді **polar (T,R)** для побудови подібної гістограми.

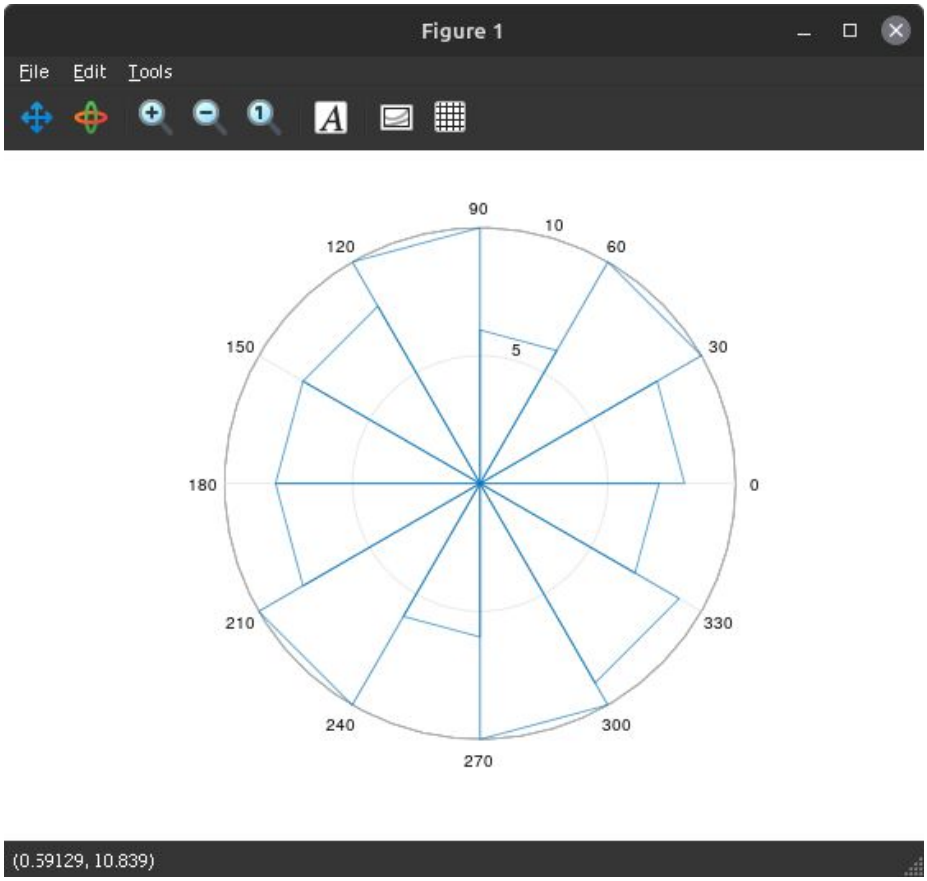


Рисунок 2.19 — Кутова гістограма

2.5. Східчасті графіки та оцінка похибки побудови функції

Східчасті графіки візуально являють собою сходи, що огинають представлену функцію y(x). Такі графіки використовуються, наприклад, для відображення процесів квантування функції (x), представленої рядом своїх відліків. При цьому в проміжках між відліками значення функції вважаються постійними та рівними величині останнього відліку.

Для побудови східчастих графіків у системі Octave використовуються команди групи **stairs**:

stairs(Y) — будує східчастий графік за даними вектора Y;

stairs(X,Y) — будує східчастий графік за даними вектора Y з координатами x переходів від сходинки до сходинки, заданими значеннями елементів вектора X;

stairs(...,S) — аналогічна по дії вищеописаним команд, але будує графік лініями, стиль яких задається рядками S.

Наступний приклад ілюструє побудову східчастого графіка (рис.2.20):

```
>> x=0:0.25:10;
>> stairs(x,x.^2);
```

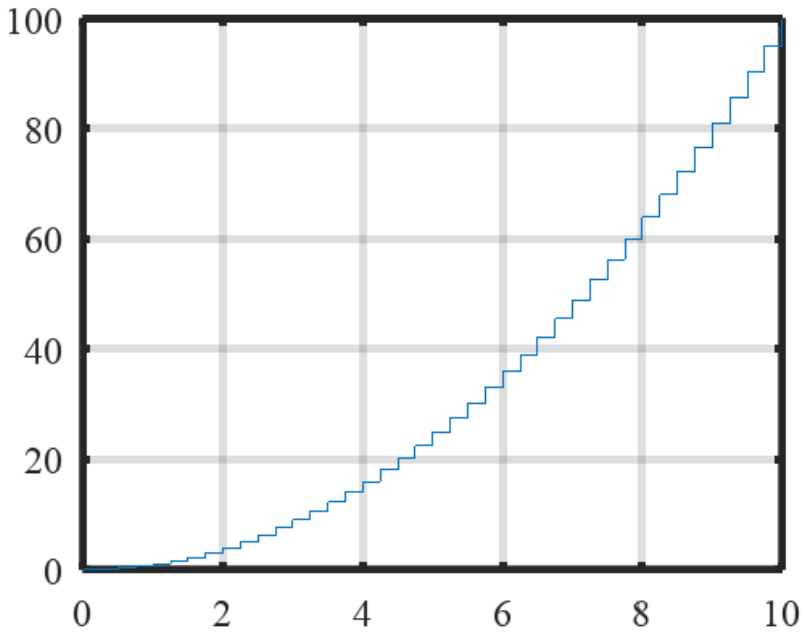


Рисунок 2.20 — Приклад східчастого графіка

Зверніть увагу на те, що відліки беруться через рівні проміжки горизонтальної осі. Якщо, наприклад, відображається функція часу, то stairs має вигляд квантованої за часом функції.

Функція **H=stairs(X,Y)** повертає вектор дескрипторів графічних об'єктів, тобто графік не будує, а повертає вектори XX і YY, які дозволяють побудувати графік з допомогою команди **plot(XX,YY)**.

Ще один вид графіка функції y(x) — представлення дискретними відліками. Цей вид графіка застосовується, наприклад, для опису квантування сигналів. Кожен відлік є вертикальною рискою, увінчаною кружком, причому висота риски відповідає y-координаті точки.

Для побудови графіка такого виду використовуються команди **stem(...)**:

stem(X,Y) — будує графік відліків з ординатами у векторі Y та абсцисами у векторі X;

stem(.... 'LINESPEC') — дає побудови, аналогічні раніше наведеним командам, але зі специфікацією ліній LINESPEC, подібної специфікації, наведеної для функції plot;

stem(Y) — будує графік функції з ординатами у векторі Y у вигляді відліків;

stem(....,'filled') — будує графік функції із зафарбованими маркерами.

Наступний приклад ілюструє застосування команди **stem** (рис.2.21):

```
>> x=0:0.1:4;
>> y=sin(x.^2).*exp(-x);
>> stem(x,y)
```

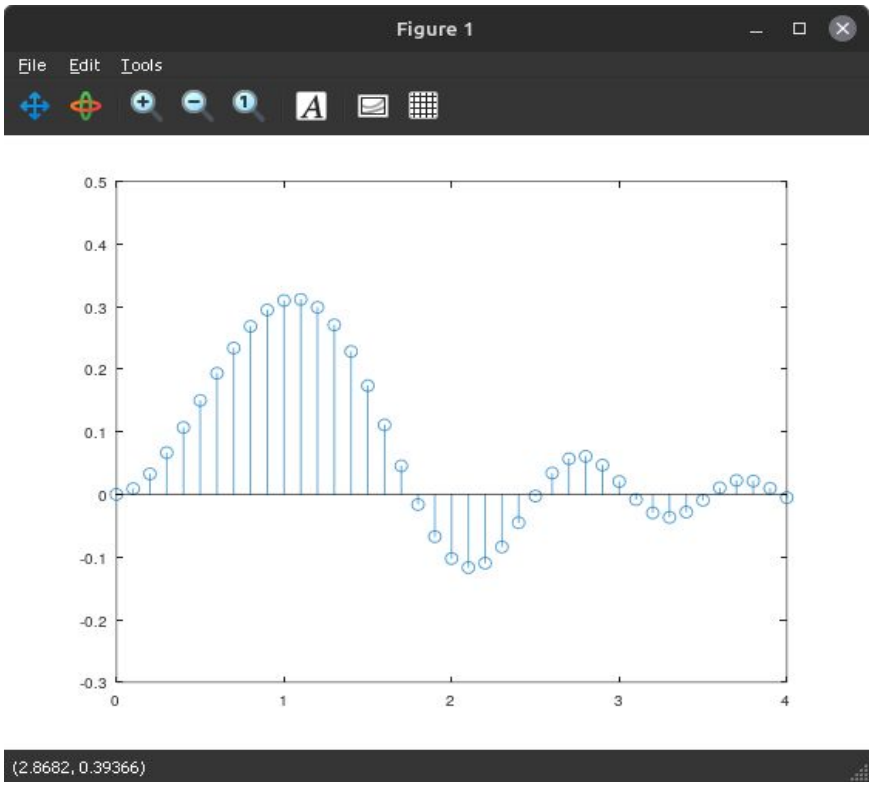


Рисунок 2.21 — Приклад графіка дискретних відліків функції

Якщо дані для побудови функції визначені з помітною похибкою, то використовують графіки функцій типу **errorbar** з оцінкою похибки кожної точки шляхом її подання у вигляді літери **I**, висота якої відповідає заданій похибці подання точки.

Команда **errorbar** використовується у такому вигляді:

errorbar(X,Y,L,U) — будує графік значень елементів вектора Y в залежності від даних, що містяться у векторі X, із зазначенням нижньої та верхньої меж значень, заданих у векторах L і U;

errorbar(X,Y,E), H=errorbar(Y,E) — будує графіки функції Y(X) із зазначенням цих меж у вигляді [Y-E Y+E], де E — похибка;[функція H=errorbar(...) повертає вектор дескрипторів графічних об'єктів];

errorbar(..., 'LineSpec') — аналогічна описаним вище командам, але дозволяє будувати лінії зі специфікацією **LineSpec**, аналогічної специфікації, застосованої в команді **plot**.

Наступний приклад ілюструє застосування команди **errorbar** (рис.2.22):

```
>> x=-2:0.1:2;
>> y=erf(x);
>> e=rand(size(x))/10;
>> errorbar(x,y,e)
```

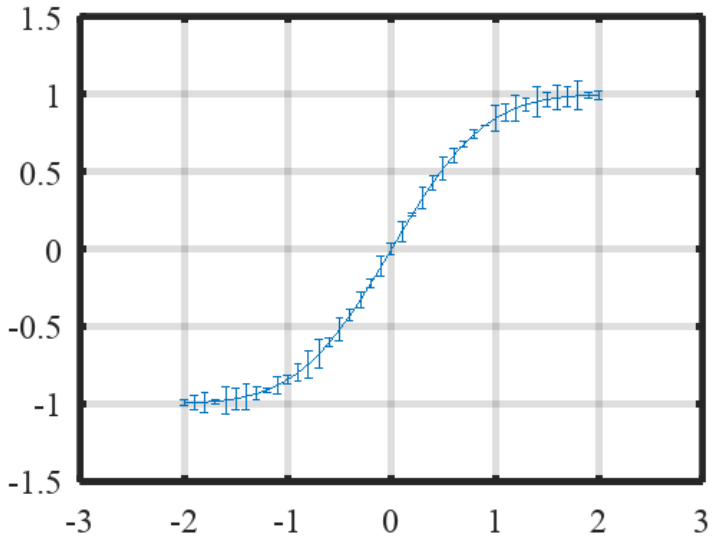


Рисунок 2.22 — Графік функції **erf(x)** із діапазонами похибки

2.6. Графіки у полярній системі координат

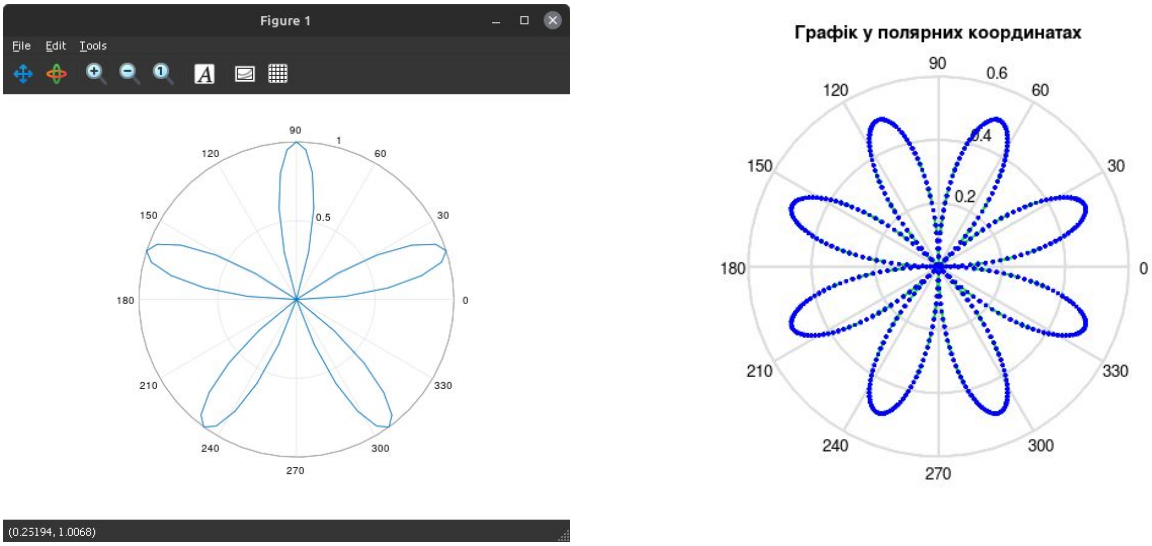
У полярній системі координат будь-яка точка представляється як кінець радіус-вектора, що виходить із початку системи координат, що має довжину ρ та кут φ. Для побудови графіка функції ρ(φ) використовуються наведені нижче команди. Кут φ зазвичай змінюється від 0 до 2*pi. Для побудови графіків функцій у полярній системі координат використовуються команди типу **polar(...)**:

polar(PHI, RHO) — будує графік у полярній системі координат, що є положенням кінця радіус-вектора з довжиною RHO і кутом PHI;

polar(PHI,RHO,S) – аналогічна до попередньої команди, але дозволяє задавати стиль побудови за допомогою рядкової константи S за аналогією з командою **plot**.

Сценарій виконання команд та графік функції (рис.2.23, *а*):

```
>> t=0:pi/50:2*pi;
>> polar(t,sin(5*t))
```



а *б*

Рисунок 2.23 — Графіки функцій у полярній системі координат

Графіки функцій у полярних координатах можуть мати досить різноманітний вигляд, часом нагадуючи такі об'єкти природи, як сніжинки чи кристали льоду на склі.

2.7. Графіки векторів

Іноді бажано уявлення ряду радіус-векторів у їх звичайному вигляді, тобто у вигляді стрілок, що виходять з початку координат і мають кут і довжину, які визначаються дійсною і уявною частиною комплексних чисел, що представляють ці вектори. Для цього застосовується команда **compass**:

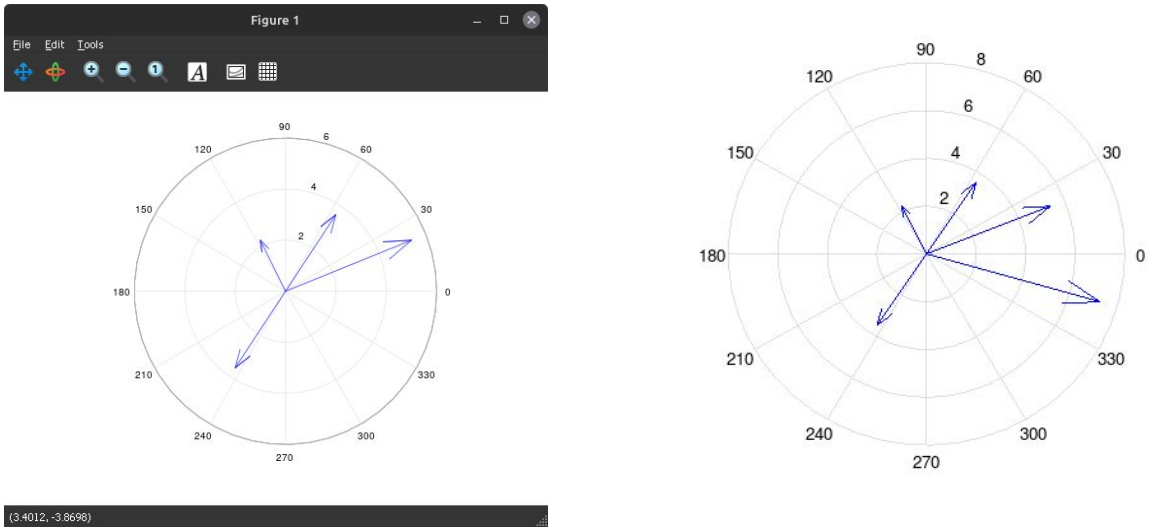
compass(U,V) — будує графіки радіус-векторів з компонентами (U,V), що представляють дійсну та уявну частини кожного з радіус-векторів;

compass (Z) — еквівалентно **compass (real (Z), imag(Z))**;

compass(U,V,linespec) і **compass(Z,linespec)** — аналогічні представленим вище командам з додаванням специфікації ліній побудови **linespec**, подібну до описаної для команди **plot**.

У наступному прикладі показано використання команди **compass (рис.2.24):**

```
>> Z=[-1+2i -2-3i 2+3i 5+2i];
>> compass(Z)
```



а *б*

Рисунок 2.24 — Представлення графічно радіус-векторів

Функція **H=compass(...)** будує графік і повертає дескриптори графічних об'єктів.

Іноді корисно відображати комплексні величини виду **z = x + y** у вигляді проекції радіус-вектора на площину. Для цього використовується сімейство графічних команд класу **feather**:

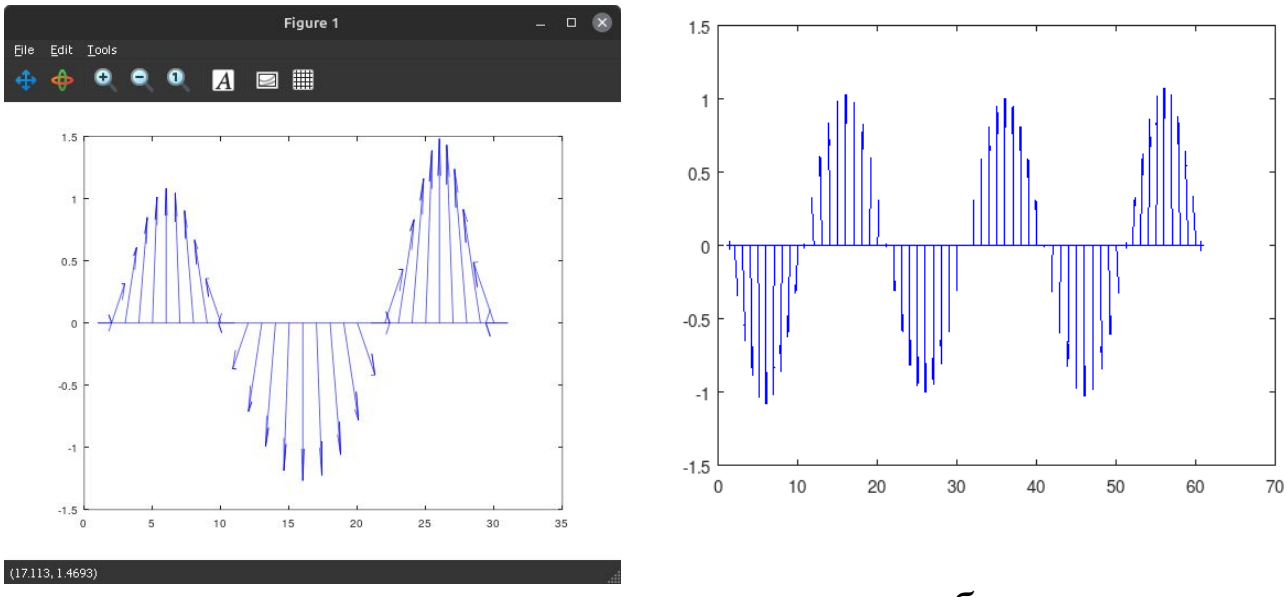
feather(U,V) — будує графік проекції векторів, заданих компонентами U та V, на площину;

feather(z) — для вектора z з комплексними елементами здійснює побудови, аналогічні **feather(real(z),imag(z))**;

feather(..., S) — з додаванням специфікації ліній, заданою рядковою константою S за аналогією з командою **plot**.

Приклади застосування команди **feather представлено у** наступних сценаріях (рис.2.24 *а, б* відповідно):

```
>> x=0:0.1*pi:3*pi;
>>y=0.05+i;
>> z=exp(x*y);
>> feather(z)
>>x=-3*pi:0.1*pi:3*pi;
>>y=0.05+i;
>>z=sinh(x*y);
```



а *б*

Рисунок 2.24 — Проєкції радіус-векторів комплексної величини на площину

2.8. Тривимірні графіки

У графіка поверхні (тривимірному або 3D-графіка) положення точки визначається значеннями трьох координат.

Так, **двовимірні контурні графіки** призначено для уявлення на площині функції двох змінних виду **z(x, y)** за допомогою ліній рівного рівня. У разі перетинання тривимірної поверхні площинами, розташованими паралельно одна одній. При цьому контурний графік є сукупністю спроектованих на площину (x, y) ліній перетину поверхні **z(x,y)** площинами.

Для побудови контурних графіків використовуються команди **contour**:

contour(Z) — будує контурний графік за даними матриці Z з автоматичним завданням діапазонів зміни x та y;

contour(X,Y,Z) — будує контурний графік за даними матриці Z із зазначенням специфікацій для X і Y; **contour(Z,N)** і **contour(X,Y,Z,N)** — дає побудови, аналогічні раніше описаними командами, із завданням N ліній рівного рівня (**за умовчанням N=10**);

contour(Z, V) та **contour(X,Y,Z,V)** — будують лінії рівного рівня для висот, вказаних значеннями елементів вектора V;

contour(Z,[-v v]) або **contour(X,Y,Z,[-v v])** — обчислює одиничний контур для рівня v;

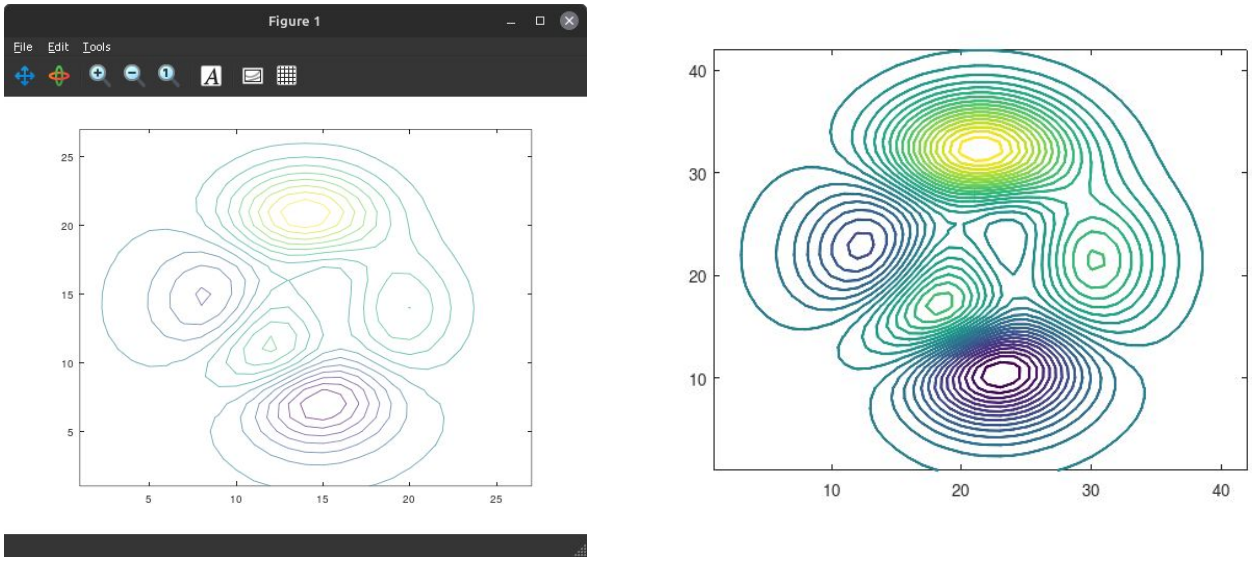
[C,H]=contour(...) — повертає дескриптори — матрицю C і вектор-стовпець H. Вони можуть використовуватися як вхідні параметри для команди **clabel**;

contour(... 'LINESPEC') — дозволяє використовувати перелічені вище команди із зазначенням специфікації ліній, якими йде шикування.

Приклади побудови контурного графіка поверхні, заданої у системі у готовому вигляді функцією **peaks** (рис. 2.25, *а, б*) відповідно:

```
>> z=peaks(27);
>> contour(z,15)

>> z=peaks(42);
>> contour(z,31)
```



а б

Рисунок 2.25 — Двовимірні контурні графіки, побудовані за допомогою команди **contour**

Графіки такого виду часто використовуються в топографії для представлення об'ємного рельєфу місцевості. Для оцінки висот контурних ліній використовується їхнє функціональне забарвлення.

Тривимірний контурний графік являє собою розташовані в просторі лінії рівного рівня, отримані при розшаруванні тривимірної фігури поруч площин, що січуть, розташованих паралельно опорній площині фігури.

На відміну від двовимірного, **лінії рівного рівня тривимірного контурного графіка відображаються в аксонометрії**. Для отримання тривимірних контурних графіків використовується команда **contour3**:

contour3(...) — має синтаксис, аналогічний команді **contour(...)**, але будує лінії рівного рівня в аксонометрії з використанням

функціонального забарвлення (забарвлення змінюється вздовж осі Z).

Корисні форми запису цієї команди:

contour3(Z) — будує контурні лінії для поверхні, заданої масивом Z, без урахування діапазону зміни x та y;

contour3(Z,n) — будує те, що попередня команда, але з використанням n площин перетину (за замовчуванням n=10);

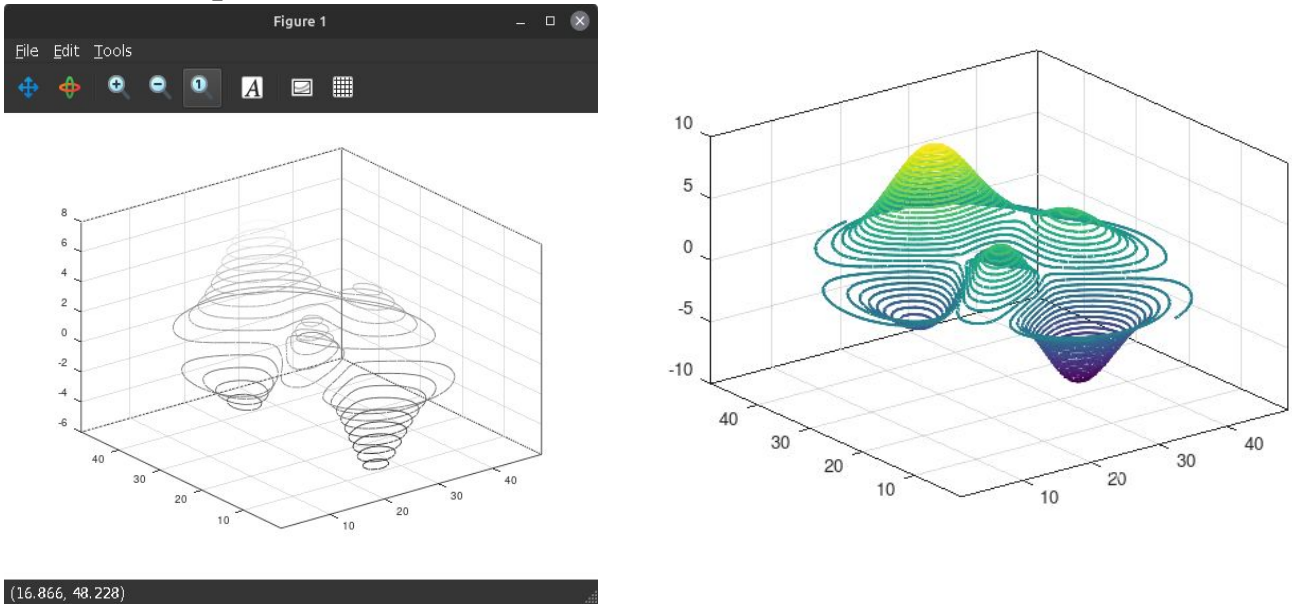
contour3(X,Y,Z) — будує контурні лінії на поверхні, заданої масивом Z, з урахуванням зміни x і y. Двовимірні масиви X та Y створюються за допомогою функції **meshgrid**;

contours(X,Y,Z,n) — будує те саме, що попередня команда, але з використанням n перерізів площини.

Приклад застосування команди **contour3** з побудовою 20 та 40 ліній рівня відповідно (рис.2.26 *а, б*):

```
>> contour3(peaks,20)
>> colormap(gray)

>> contour3(peaks,40)
>> colormap(blue)
```



а *б*

Рисунок 2.26 — Тривимірні контурні графіки для функції **peaks**

Величина z називається функцією двох величин x та y, якщо кожній парі чисел, які можуть бути значеннями змінних x та y, відповідає одне або декілька певних значень величини z. У цьому змінні x і y називають аргументами функції **z(x, y)**. Пари тих чисел, які можуть бути значеннями аргументів **x, y** функції **z(x, y)**, у сукупності складають область визначення цієї функції. Для побудови графіка двох змінних **z = f(x, y)** необхідно виконати такі дії:

1.Сформувати для побудови графіка прямокутну сітку, проводячи прямі, паралельні осям $y=y_j$ та $x=x_i$;

2.Обчислити значення $z_{ij}=f(x_i,y_j)$ у всіх вузлах сітки;

3.Звернутися до функції побудови поверхні, передаючи їй як параметри сітки та матрицю $Z=\{z_{i,j}\}$ значень у вузлах сітки.

Специфіка побудови тривимірних графіків вимагає непросто завдання низки значень x і y, тобто векторів x і y. Вона вимагає **визначення X і Y двовимірних масивів — матриць**. Для формування прямокутної сітки в Octave та створення таких масивів служить функція **meshgrid**. Здебільшого вона використовується разом із функціями побудови графіків тривимірних поверхонь: **plot3**, **mesh, surf**.

Функція **meshgrid** записується у таких формах:

[X,Y] = meshgrid(x) — аналогічна **[X,Y] = meshgrid(x,x)**;

[X,Y,Z] = meshgrid(x,y,z) — повертає тривимірні масиви, що використовуються для обчислення функцій трьох змінних та побудови тривимірних графіків;

[X,Y] = meshgrid(x,y) — перетворює область, задану векторами x і y, масиви X і Y, які можуть бути використані для обчислення функції двох змінних і побудови тривимірних графіків. Рядки вихідного масиву X є копіями вектора x; а стовпці Y — вектора y.

Приклад:

```
>> [X,Y] = meshgrid(1:4,13:17)
X=
1 2 3 4
1 2 3 4
1 2 3 4
1 2 3 4
1 2 3 4
Y=
13 13 13 13
14 14 14 14
15 15 15 15
16 16 16 16
17 17 17 17
```

Наведемо ще один приклад застосування функції **meshgrid**:

```
>> [X,Y] = meshgrid(-2:.2:2,-2:.2:2);
```

Такий виклик функції дозволяє встановити опорну площину для побудови тривимірної поверхні при зміні сітки від -2 до 2 з кроком 0.2.

Функція **ndgrid** є багатовимірним аналогом функції **meshgrid**:

[X1,X2,X3,…]=ndgrid(x1,x2,x3….) — перетворює область, задану векторами x1, x2, x3…, у масиви X1, X2, X3…, які можуть бути використані для обчислення функцій декількох змінних і багатовимірної інтерполяції, *i*-я розмірність вихідного масиву X_i є копією вектора x_i;

[X1,X2….]=ndgrid(x) — аналогічна **[X1,X2 ….] = ndgrid (x, x, …)**.

Приклад застосування функції **ndgrid** представлено нижче (рис.2.27):

```
>> [X1,X2] = ndgrid(-2:.2:2, -2:.2:2)
>> Z = X1.*exp(-X1.^2-X2.^2);
>> mesh(Z)
```

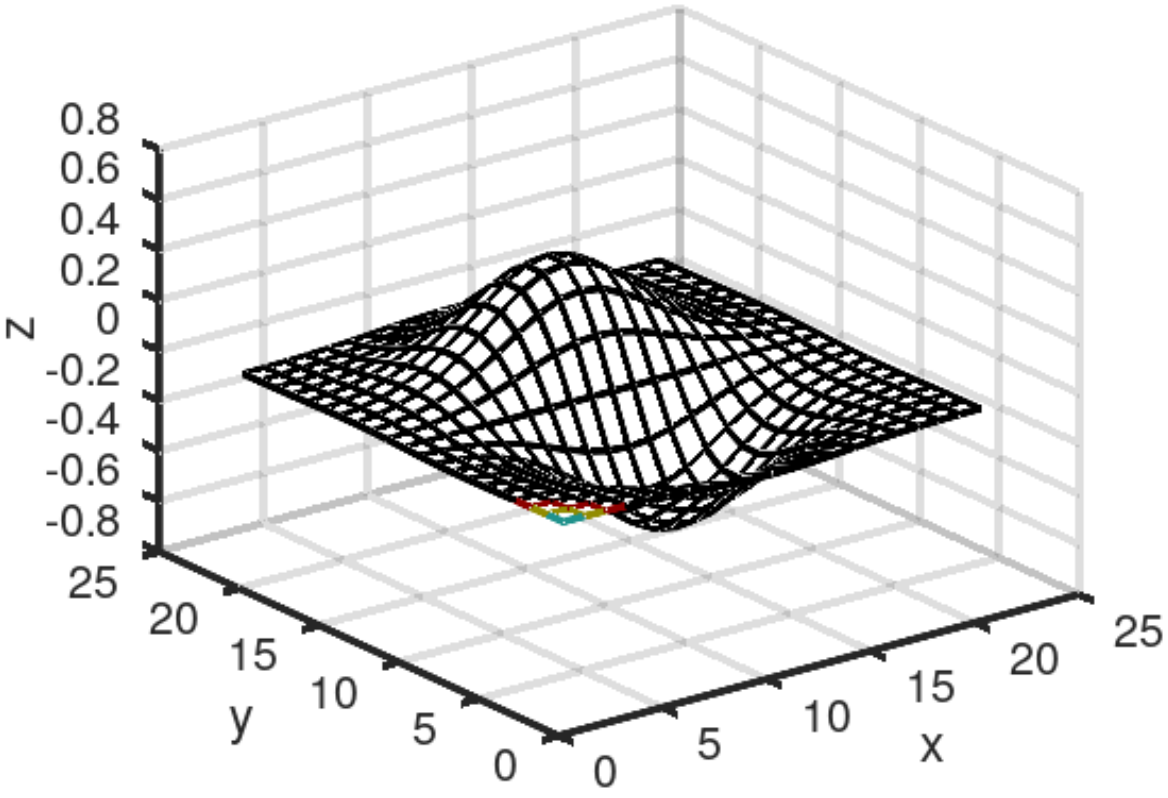


Рисунок 2.27 — Приклад використання команди **ndgrid**

Для побудови поверхні розглянемо команду **plot3(...),** яка є аналогом команди **plot(...)**, але належить до функції двох змінних z(x, y). Ця команда дозволяє побудувати аксонометричне зображення тривимірних поверхонь і представлена такими формами:

plot3(x,y,z) — будує масив точок, представлених векторами x, y та z, з'єднуючи їх відрізками прямих (має обмежене застосування);

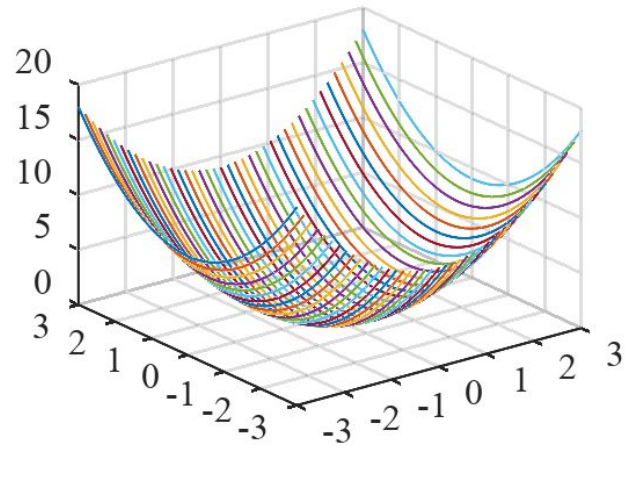
plot3(X,Y,Z), де X, Y і Z — три матриці однакового розміру, будує точки з координатами X(i.:), Y(i,:) та Z(i,:) і з'єднує їх відрізками прямих.

Нижче наведено приклад побудови тривимірної поверхні, що описується функцією z(x,y)=x^2+y^2 (рис.2.28, *а*):

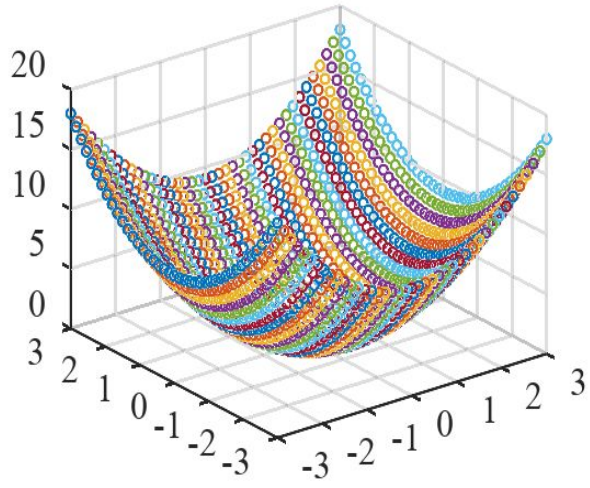
```
>> [X,Y]=meshgrid(-3:0.15:3);
>> Z=X.^2+Y.^2;
>> plot3(X,Y,Z)
```

Команда **plot3 (X, Y, Z, S)** — забезпечує побудови, аналогічні розглянутим раніше, але зі специфікацією стилю ліній і точок, що відповідає специфікації команди **plot**. Нижче наведено приклад застосування цієї команди для побудови поверхні кружками (рис.2.28, *б*):

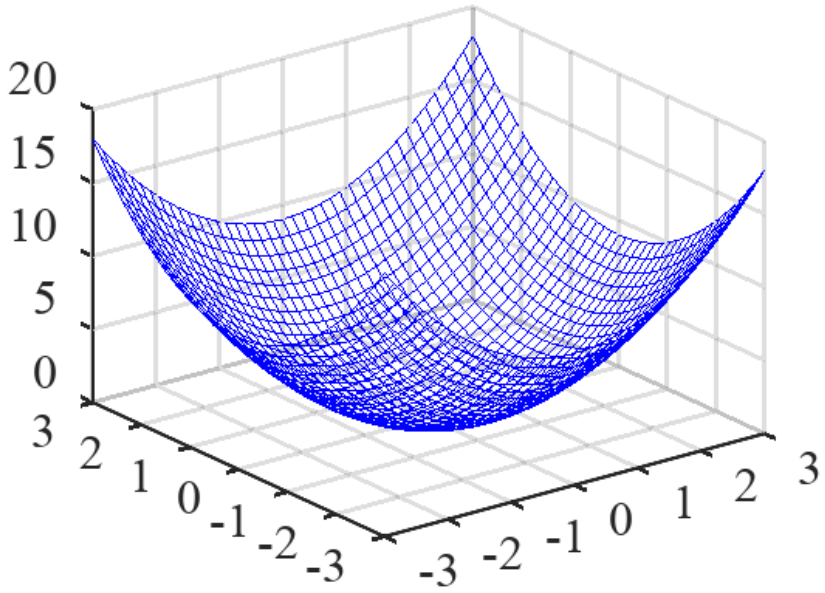
```
>> [X,Y]=meshgrid(-3:0.15:3);
>> Z=X.^2+Y.^2;
>> plot3(X,Y,Z.'o')
```



а



б



в

Рисунок 2.28 — Поверхні, побудовані із застосуванням команди **plot3(X,Y,Z)** (*а*), **plot3 (X, Y, Z, S)** (*б*) та **plot3(X,Y,Z,'-b',Y,X,Z,'-b')** (*в*)

Команда **plot3(x1,y1,z1,s1,x2,y2,z2,s2,x3,y3,z3,s3,...)** дозволяє побудову на одному рисунку графіки кількох функцій **z1(xl ,yl)**, **z2(x2,y2)** зі специфікацією ліній і маркерів кожної їх (рис.2.28, *в*):

```
>> [X,Y]=meshgrid([-3:0.15:3]);
>> Z=X.^2+Y.^2;
>> plot3(X,Y,Z,'-b',Y,X,Z,'-b')
```

У цьому випадку будуються два графіки однієї й тієї ж функції із взаємно перпендикулярними лініями, що утворено. Графік має вигляд сітки без забарвлення її осередків (нагадує дротяний каркас фігури).

Найбільш представницькими та наочними є сітчасті графіки поверхонь із заданим або функціональним забарвленням. Для побудови застосовується команда **mesh**:

mesh(X,Y,Z,C) — виводить у графічне вікно сітчасту поверхню **Z(X,Y)** з кольорами вузлів поверхні, заданими масивом;

mesh(X,Y,Z) — аналог попередньої команди при C=Z. В даному випадку використовується функціональне забарвлення, при якому колір задається висотою поверхні.

Можливі також форми команди **mesh(x,y,Z)**, **mesh(x,y,Z,C)**, **mesh(Z)** та **mesh(Z,C)**.

Функція **mesh** повертає дескриптор для класу об'єкта **surface**.

Приклад графіка із застосуванням команди **mesh(X,Y,Z)** представлено на рис. 2.29. Поверхня на рис. 2.29, *а* побудована за сценарієм для поверхні, наведеної на рис.2.28, *а*. Відмінність полягає у застосуванні команди **mesh(X,Y,Z)** замість **plot3(X,Y,Z)**:

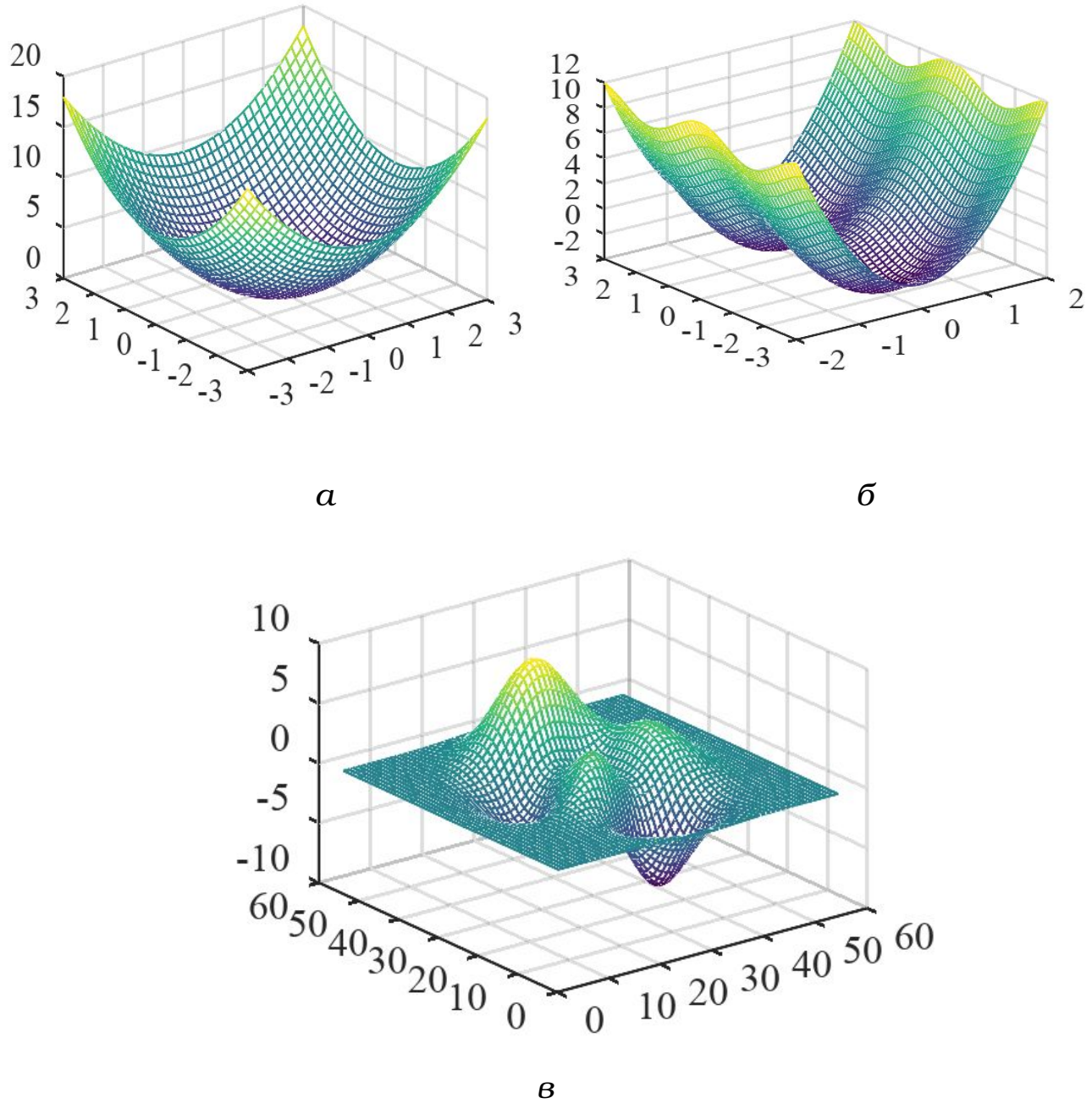
рис. 2.29, *а:*

```
>> [X,Y]=meshgrid(-3:0.15:3);
>> Z=X.^2+Y.^2;
>> mesh(X,Y,Z)
```

На рис. 2.29, *б* подано графік поверхні функції z=3x^2-sin(y^2):

```
>>[X, Y]= meshgrid(- 2:0.1:2, -3:0.1:3);
```

```
>>Z=3*X.*X-2*sin(Y).^2
>>mesh(X, Y, Z)
```



а *б*

в

Рисунок 2.29 — Результати роботи команди **mesh(X,Y,Z)**

Функціональне фарбування ліній поверхні помітно посилює наочність її уявлення (порівняйте рис. 2.28, *а* та рис.2.29, *а*).

Octave має кілька графічних функцій, що повертають матричний образ поверхонь [1-2]. Наприклад, функція **peaks(N)** повертає матричний образ поверхні з рядом піків та западин. Такі функції

зручно використовуватиме для перевірки роботи графічних команд тривимірної графіки. Для функції **peaks** можна навести такий приклад (рис.29, *б*):

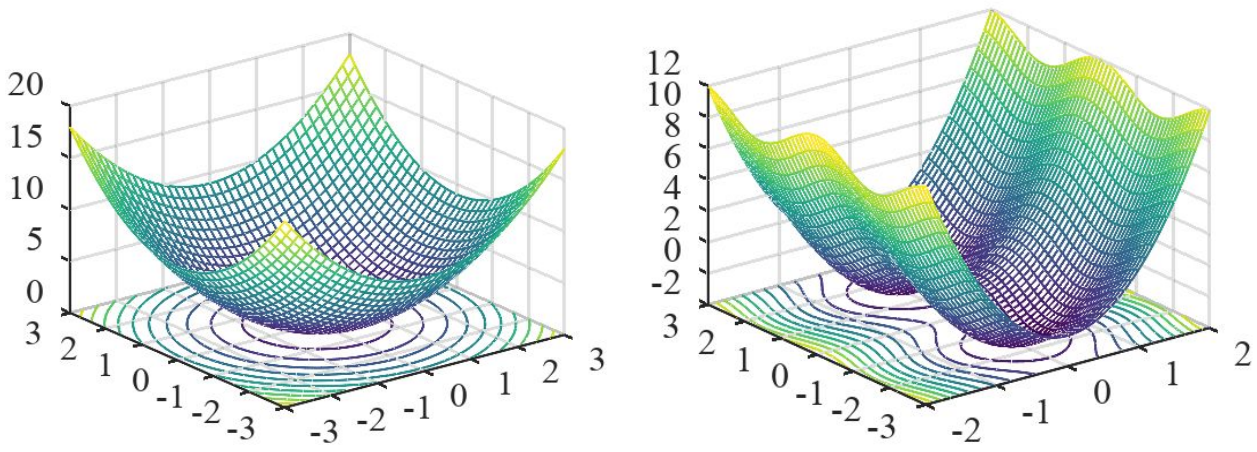
```
>> z=peaks(25);
>> mesh(z);
```

Іноді графік поверхні корисно поєднати з контурним графіком її проекції на площину, розташованому під поверхнею.

Для цього використовується команда **meshc**:

meshc(...) — аналогічна **mesh(...)**, але крім графіка поверхні дає зображення її проекції як ліній рівного рівня (графіка типу **contour**).

На рис.2.30 наведено приклад застосування цієї команди для поверхонь, наведених на рис.2.29, *а, б* відповідно. Графіки такого типу дають краще уявлення про особливості поверхні.



а *б*

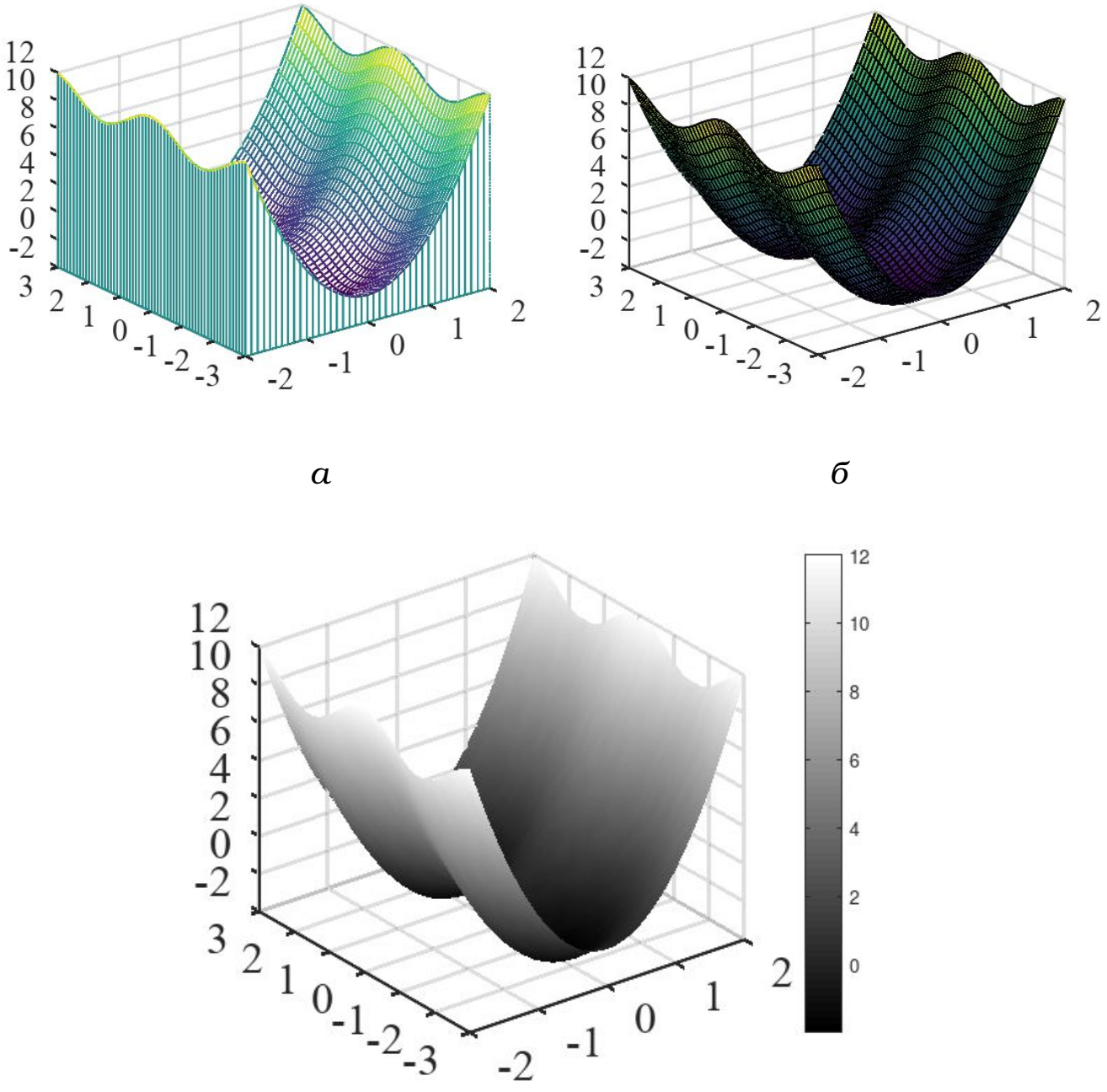
Рисунок 2.30 — Графіки поверхонь та їх проекції на розташовану нижче площину

Для побудови поверхні з численних стовпців застосовуються команди класу **meshz:**

meshz(...) — аналогічна **mesh(...)**, але будує поверхню у вигляді стовпчиків. Наступний приклад ілюструє застосування команди **mesh**

для побудови параболоїда (рис.2.31, *а*):

```
>> [X,Y]=meshgrid([-3:0.15:3]);
>> Z=X.^2+Y.^2;
>> meshz(X,Y,Z)
```



а *б*

в

Рисунок 2.31 — Приклад представлення побудови параболоїда командами **meshz(X,Y,Z)** та **surf(X,Y,Z)**

Особливо наочне уявлення про поверхні дають сітчасті графіки, що використовують функціональне забарвлення осередків. Наприклад, колір забарвлення поверхні **z(x, y)** може бути поставлений у відповідність до висотою z поверхні з вибором для малих висот темних тонів, а для великих – світлих. Для побудови таких поверхонь використовуються команди класу **surf (...)**:

surf(X,Y,Z,C) — будує кольорову параметричну поверхню за даними матриць X, Y і Z з кольором, що задається масивом C;

surf(X,Y,Z) — аналогічна попередній команді, де C = Z, так що колір задається висотою того чи іншого осередку поверхні;

surf(x,y,Z) та **surf(x,y,Z,C)** з двома векторними аргументами x і y — вектори x і y замінюють перші два матричні аргументи і повинні мати довжини **length(x)=n** та l**ength(y)=m**, де **[m,n]=size(Z)**. І тут вершини областей поверхні представлені трійками координат (x(j), y(j), Z(1,j)). Зауважимо, що **x** відповідає стовпцям **Z**, а **y** відповідає рядкам.

surf(Z) і **surf(Z,C)** використовують x=1:n та y=1:m. І тут висота Z — однозначно певна функція, задана геометрично прямокутною сіткою;

h=surf (...) — будує поверхню і повертає дескриптор об'єкта класу **surface**.

Команди **axis**, **caxis**, **color-map**, **hold**, **shading** та **view** задають координатні осі та властивості поверхні, які можуть використовуватися для більшої ефектності показу поверхні або фігури.

Простий приклад побудови параболоїда з функціональним забарвленням осередків командою **surf(X,Y,Z**) показано на рис.2.31, *б*).

Можна помітити, що завдяки функціональному забарвленню графік поверхні набагато виразніший, ніж при побудовах без такого

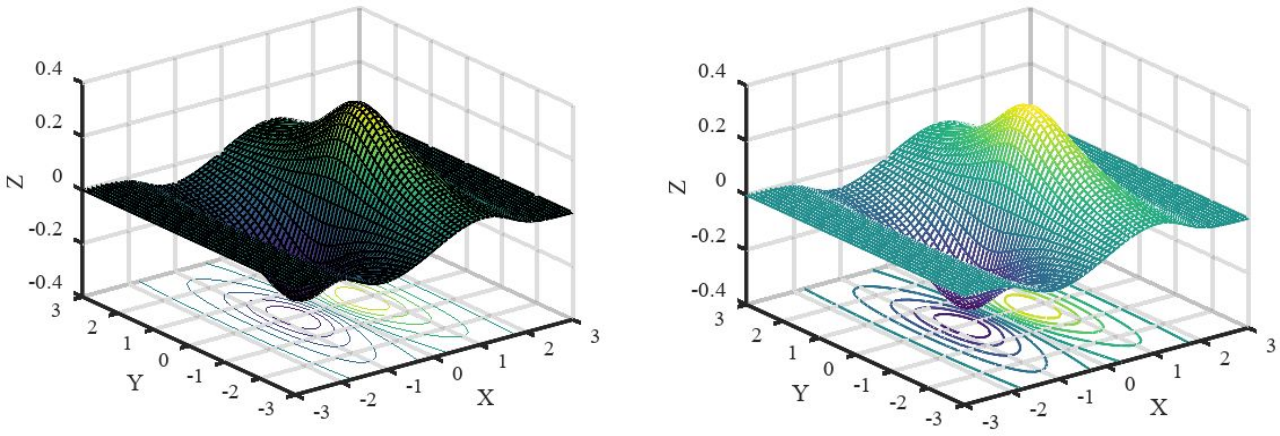
забарвлення (порівняйте рис.2.31, *а* та рис.2.31, *б).* Навіть у тому випадку, коли кольоровий графік друкується у чорно-білому вигляді.

У наступному прикладі використовується функціональне забарвлення параболоїда відтінками сірого кольору з виведенням шкали відтінків кольорів (рис.2.31, *в*) з додаванням команд до сценарію рис.2.31, *б*:

```
>> colormap(gray)
>> shading interp
>> colorbar
```

У цьому прикладі команда **colormap(gray)** задає забарвлення тонами сірого кольору, а команда **shading Interp** забезпечує усунення зображення сітки і задає інтерполяцію кольорів об'ємної поверхні.

Зазвичай застосування інтерполяції для фарбування надає поверхням і фігурам більш реалістичний вигляд. В той же час фігури каркасного вигляду дають точніші кількісні дані про кожну точку.



а *б*

Рисунок 2.32 — Реалізація команд для побудови поверхні та її проекції на опорну площину

Для підвищення наочності уявлення поверхонь можна використовувати додатковий графік ліній рівного рівня, що

отримується шляхом проектування поверхні на опорну площину графіка (під поверхнею). Для цього використовується команда **surfc**:

surfc(...) — забезпечує додаткову побудову контурного графіка проекції фігури на опорну площину (рис.2.32, *а*).

На рис. 2.32 з одночасним нанесенням на графіки позначення осей показано приклад побудови ліній рівного рівня командою **surfc** (рис.2.32, *а*) та **meshc** (рис.2.32, *б*).

Приклад застосування команди **surfc** для побудови поверхні, що описується функцією **peaks**, із застосуванням інтерполяції кольорів та побудовою колірної шкали реалізовано у наступному прикладі (рис.2.33):

```
>> [X,Y]=meshgrid([-3:0.1:3]);
>> Z=peaks(X,Y);
>> surfc(X,Y,Z)
>>xlabel('X');ylabel('Y');zlabel('Z')
>> colorbar
```

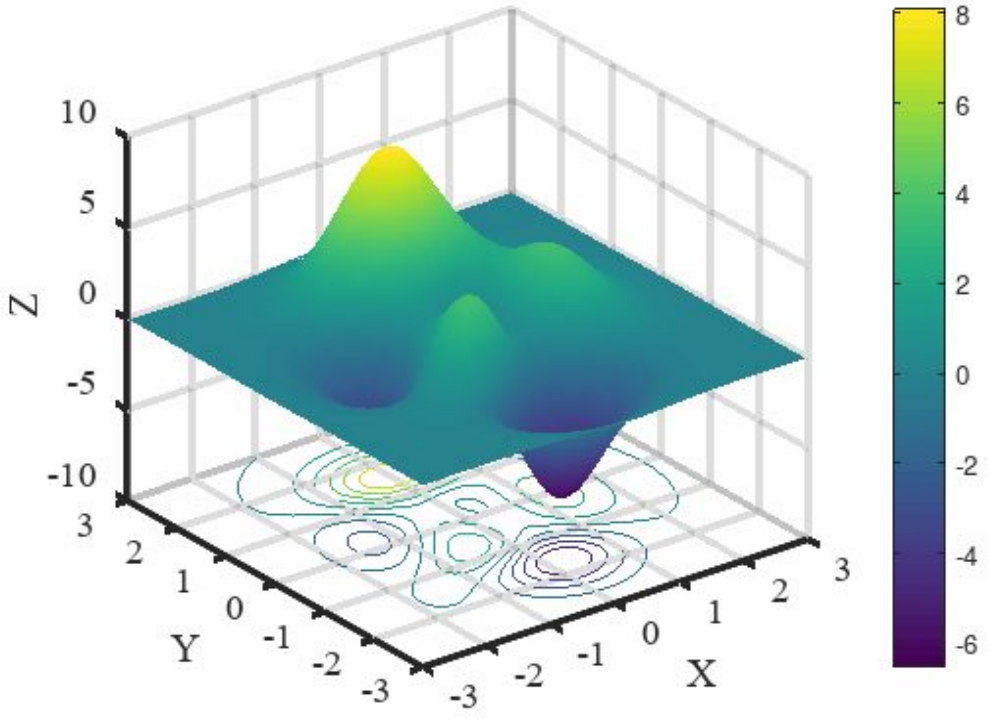


Рисунок 2.33 — Графік функції **peaks** з проекцією та шкалою кольорів

Графіки складних поверхонь з інтерполяцією відтінків кольорів виглядають більш реалістичними, ніж графіки сітчастого вигляду і графіки без інтерполяції кольорів.

Неважко помітити певну логіку у назвах графічних команд. Ім'я команди складається з основного слова та суфіксу розширення. Наприклад, всі команди побудови поверхонь мають основне слово **surf** (скорочення від surface — поверхня) та суфікси: **c** — для контурних ліній поверхні. Це правило полегшує запам'ятовування численних команд графіки.

Графіки перерізів функцій трьох змінних будує команда **slice** (у перекладі — "скибочка"):

slice(X,Y,Z,V,Sx,Sy,Sz) — будує плоскі перерізи об'ємної фігури V у напрямку осей x,y,z с позиціями, що задаються векторами Sx, Sy, Sz. Масиви X, Y, Z задають координати V і повинні бути монотонними і тривимірними (як повертаються функцією **meshgrid**) з розміром MxNxP. Колір точок перерізів визначається тривимірною інтерполяцією в об'ємній фігурі V;

slice(X,Y,Z,V,XI,YI,ZI) — будує перерізи об'ємної фігури V поверхнею, визначеної масивами XI, YI, ZI;

slice (... 'method') — при побудові задається метод інтерполяції, який може бути одним із наступних: '**linear**', '**cubic**' або '**nearest**'. За умовчанням використовується лінійна інтерполяція — '**linear**';

slice(V,Sx,Sy,Sz) або **slice(V,XI,YI,ZI)** — мається на увазі X=1:N, Y=1:M, Z=1:P;

H=slice(...) — будує перетин і повертає дескриптор об'єкта класу surface.

Графік прикладу, наведеного нижче, представлено на рис. 2.34.

```
>> [x,y,z]= meshgrid(-2:.2:2, -2:.25:2, -2:.16:2);
>> v = sin(x).*exp(-x.^2-y.^2-z.^2);
```

```
>> slice(x,y,z,v,[-0.6 0.8],0,[-2, 2])
```

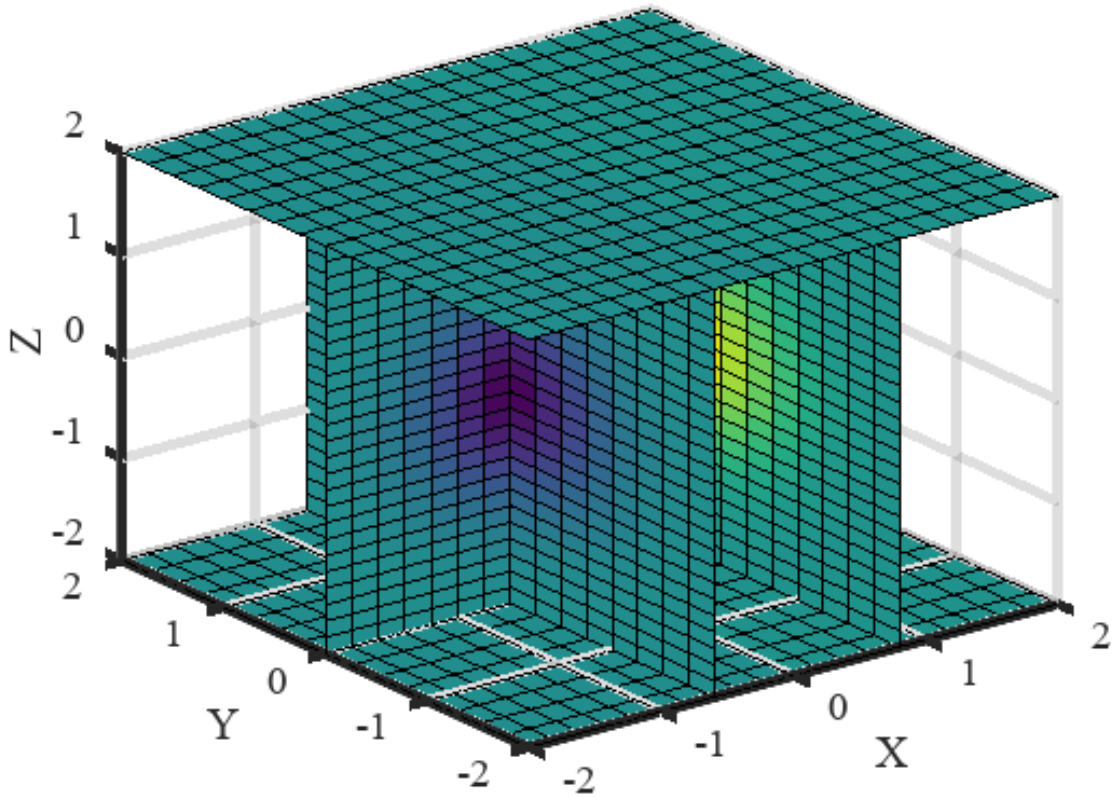


Рисунок 2.33 — Графік перерізів функцій трьох змінних із застосуванням команди **slice**

Контрольні запитання до розділу 2

1. Які можливості має графічне середовище в Octave?

2. У чому полягає особливість побудови графіків функцій ряду точок?

3. Які можливості для побудови графіків функцій, які мають усувні невизначеності, в Octave?

4. Вкажіть, яку функцію в системі Octave виконують команди: loglog, semilogx та semilogy, в чому їх відмінність?

5. У чому особливість команди plot в Octave, які параметри має ця команда та які подібні команди застосовуються при побудові

графіків?

6. Яку функцію виконує команда subplot в Octave, надайте приклади використання?

7. У чому проявляється особливість команди **polar** в Octave?

8. Що виконують команди: **contourf, contour, contour3, contourc** в Octave та чим відрізняються від подібних команд?

9. Які можливості команди **meshgrid** в Octave?

10. Що виконують та в чому відмінність команд **mesh, meshc, meshz** в Octave?

11. Що виконують та в чому відмінність команд **surf, surfc, surfl** в Octave?

12. Які додаткові параметри має команда axis для чого вона використовується?

13. Які функції виконують команди: grid, hold, box, figure, як вони застосовуються в Octave?

14. Яку функцію виконує команда **colorbar**, як вона використовується в Octave?

15. У чому особливість ліній рівного рівня тривимірного контурного графіка?

Розділ 3. Базові елементи модульного принципу програмування в GNU Octave

Одним із сучасних принципом програмування є принцип модульності, згідно з яким окремі частини програми, призначені на вирішення локальних завдань, організовуються в підпрограми [1—5]. У GNU Octave цей принцип забезпечується використанням **файлів-сценаріїв** та **функцій**. У програмних модулях кожен момент часу виконується одна функція (*F*). Якщо у тілі функції *F* у виразі зустрічається виклик — ім'я іншої функції (*G*), між ними встановлюється тимчасовий (динамічний) зв'язок: виконання першої функції припиняється до того часу, доки виконатися друга. Цей принцип виконання називається **вкладеністю функцій викликів** і може бути повторений багаторазово.

3.1. Визначення функцій

Функції в Octave можна визначати в скриптах і в самому інтерпретаторі (у MATLAB — тільки в спеціальних m файлах).

Функція — це фрагмент програмного коду, який можна викликати з інших частин програми. Має вхідні та вихідні параметри (аргументи), а також свою область видимості для змінних.

Функція, як і програма, призначена для багаторазового використання: має вхідні параметри і не виконується без попереднього завдання. Зазначимо, що у мові Octave регістр написання букв враховується. Наприклад, **function** — правильно, **Function** — неправильно.

Функція має заголовок виду

function name1 [, name2, ...] = fun(var1 [, var2, ...]),

де name1 [, name2, ...] — список **вихідних параметрів**, тобто

змінних, яким буде присвоєно кінцевий результат обчислень,

fun — ім'я функції, var1 [, var2, …] — **вхідні параметри**. Таким чином, найпростіший заголовок функції виглядає так:

function name = fun(var)

Виклик програм у Octave здійснюється з командного рядка. Іменовані функції зберігаються у файлах. Програму можна запустити на виконання, вказавши ім'я файлу, у якому вона збережена. Звернення до функції здійснюється так само, як і до будь-якої іншої вбудованої функції системи, тобто із зазначенням вхідних та вихідних параметрів. Викликати функцію можна з командного рядка або використовувати як один із операторів програми.

У загальному вигляді функція має наступний синтаксис:

function [ret-list] = name(arg-list)
команди …
endfunction

де ret-list — перелік змінних, з розрахованими значеннями, arg-list — перелік змінних, що передаються в функцію як вхідні дані (обидва списки параметрів не є обов'язковими). Важливо: при стиканні GNU Octave з невідомим ім'ям функції у командному рядку, починається перегляд усіх директорій, визначених у **LOADPATH**. У разі збігу цієї назви з аналогічним ім'ям файлу в якомусь каталозі, починається його "автоматичне" завантаження та виконання. Таким чином, обов'язковою є умова **ідентичності назви функції та імені m-файлу,** в якому вона визначена (наприклад, функція MyFunction повинна описуватися у файлі MyFunction.m). Крім того, файли-функції повинні починатися з ключового слова **function**!

При створенні та збереженні функції слід пам'ятати, що її ім'я повинне спів падати з ім'ям файлу.

Усі імена змінних усередині функції, а також імена зі списку

вхідних та вихідних параметрів, сприймаються системою як локальні, тобто ці змінні вважаються певними лише усередині функції.

Так, функція з без вихідних параметрів (є аналогом поняття "процедура") записується у вигляді:

```
%Файл Sum_Vectors.m
%Складання векторів
function Sum_Vectors(v1, v2)
  % функція disp(arg) виводить на екран аргумент arg
  disp("Sum = "), disp(v1 .+ v2);
  endfunction
```

Приклад реалізації функції:

```
%Складання векторів
function Sum_Vectors(v1, v2)
 % функція disp(arg) виводить на екран аргумент arg
v1=0:10
v2=0:10
disp("Sum = "), disp(v1 .+ v2)
Endfunction
```

Результат розрахунку:

```
v1 =
   0   1   2   3   4   5   6   7   8   9  10
v2 =
   0   1   2   3   4   5   6   7   8   9  10
Sum =
   0   2   4   6   8  10  12  14  16  18  20
```

У програмуванні завжди застосовується програмний код, з великою кількістю текстових коментарів. Коментар починається або зі знака дієз `#' або символу відсотка `%' і продовжується до кінця рядка. Інтерпретатор Octave ігнорує залишок рядка після знака дієз або відсоткового символу. Натискання клавіші Enter призводить до активізації наступного командного рядка.

Якщо функція завершується символом «;», результат роботи функції не виводиться на екран. В іншому випадку результат буде виведено.

Прикладом функції з одним вхідним та одним вихідним

параметром є функція перетворення радіан на градуси (див. розділ 1) :

```
%функція g= RadToGrad(r)
r=pi/2 %вхідне, задане, значення в радіанах
g= r*180/pi %вихідний параметр у градусах
Endfunction
Результат розрахунку:
r= 1.5708
g = 90
```

Приклад функції з одним входом та одним виходом для побудови кругової діаграми (рис. 3.1) із застосуванням вбудованої функції pie(a) показано нижче:

```
function diagr()
a=[1.3 2.7 1.9 1.8 0.7];
hp=pie(a);
% функція побудови кругової діаграми
grid on
set(gca,'FontSize',14,'fontangle','Normal','fontname','Times New Roman','LineWidth',4);
```

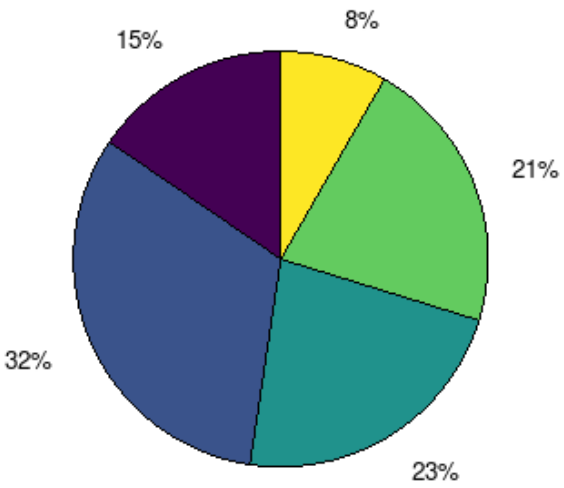


Рисунок 3.1 — Приклад реалізації вбудованої функції у тіло основної для побудови кругової діаграми

У функції function matr_21() генерується два одновимірних масиви x і y. Із викликом вбудованої у тіло функції meshgrid формується два двомірних масиви X і Y для обчислення значень функції, які містяться в масиві Z: $Z=\exp(X.^2)+\sin(Y.^2)$. Цей масив застосовується для побудови поверхні (рис. 3.2).

```
function matr_21()
x=0:0.1:1.5; y=0:0.1:2.7;
[X Y] = meshgrid(x, y);
Z=exp(X.^2)+sin(Y.^2);
figure(1),surf(X,Y,Z)
set(gca,'FontSize',16,'fontangle','Normal','fontname','Times New Roman','LineWidth',4);
xlabel('x');set(gca,'FontSize',16);
ylabel('y');set(gca,'FontSize',16);
zlabel('z');set(gca,'FontSize',16);
```

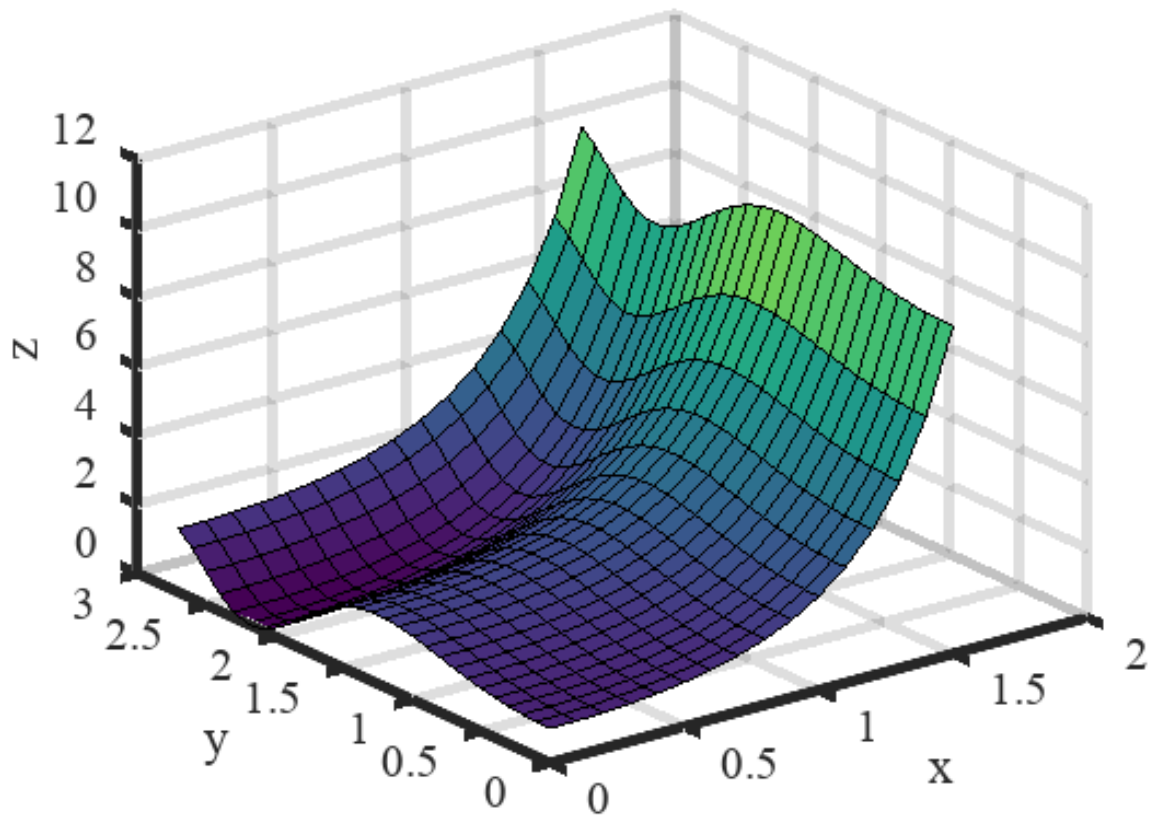


Рисунок 3.2 — Поверхня, задана значеннями функції $Z=\exp(X.^2)+\sin(Y.^2)$ у вузлових точках

Функція з кількома вхідними та вихідними параметрами представлена на прикладі розв'язання квадратного рівняння виду $ax^2+bx+c=0$ (для дійсних та комплексних коренів):

```
% Файл Roots_Equations.m
%Рішення квадратного рівняння
function [x1,x2] = Roots_Equations(a,b,c)
D = b^2 - 4 * a * c %визначення дискриминанту
x1 = (-b - sqrt(D)) / (2 * a) %визначення кореня рівняння
x2 = (-b + sqrt(D)) / (2 * a) %визначення кореня рівняння
endfunction
```

Приклад:

а) дійсні корені:

```
% Файл Roots_Equations.m
function [x1,x2] = Roots_Equations(a,b,c)
a=7; b=4; c=0;
D = b^2 - 4 * a * c %визначення дискримінанту
x1 = (-b - sqrt(D)) / (2 * a) %визначення кореня рівняння
x2 = (-b + sqrt(D)) / (2 * a) %визначення кореня рівняння
```

Результат розрахунку:

```
D = 16
x1 = -0.5714
x2 = 0
```

б) комплексні корені:

```
% Файл Roots_Equations.m
function [x1,x2] = Roots_Equations(a,b,c)
a=8; b=2; c=17;
D = b^2 - 4 * a * c %визначення дискримінанту
x1 = (-b - sqrt(D)) / (2 * a) %визначення кореня рівняння
x2 = (-b + sqrt(D)) / (2 * a) %визначення кореня рівняння
```

Результат розрахунку:

```
D = -540
x1 = -0.1250 - 1.4524i
x2 = -0.1250 + 1.4524i
```

Функцію для розв'язання кубічного рівняння наведено нижче:

```
function  [x1,x2,x3]=cub(a,b,c,d)
  a=3;
   b=-2;
   c=-1;
   d=-4;
r=b/a;
s=c/a;
t=d/a;
p=(3*s-r^2)/3;
q=2*r^3/27-r*s/3+t;
D=(p/3)^3+(q/2)^2;
u=(-q/2+sqrt(D))^(1/3);
v=(-q/2-sqrt(D))^(1/3);
y1=u+v;
y2=-(u+v)/2+(u-v)/2*i*sqrt(3);
y3=-(u+v)/2-(u-v)/2*i*sqrt(3);
x1=y1-r/3% Перший корінь
x2=y2-r/3%Другий корінь
x3=y3-r/3%Третій корінь
```

Результат розрахунку:

```
x1 = 1.4905
x2 = -0.4119 + 0.8514i
x3 = -0.4119 — 0.8514i
```

Під рекурсією в програмуванні розуміється виклик функції з її тіла. Класичними рекурсивними алгоритмами є піднесення числа до цілого позитивного степеня, обчислення факторіала, інше. У рекурсивних алгоритмах функція викликає себе до виконання будь-якої умови. Як приклад наведено обчислювання n-го числа. Якщо нульовий елемент послідовності дорівнює нулю, перший — одиниці, а кожен наступний є сумою двох попередніх, то це послідовність Фібоначчі (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, . . .).

Функція для визначення числа у послідовності Фібоначчі наведено нижче.

```
function F=fibonachi(n)
if(n==0) | (n==1)
F=n;
else
F=fibonachi(n-1)+fibonachi(n-2);
end
end
```

В термінальному режимі — у командному вікні виклик функції fibonachi(2) з n=2 надає результат:

ans = 1

Функції fibonachi(4) з n=4 надає результат:

ans = 3

Анонімні функції зберігаються у змінних. Синтаксис:

func = @(arg1,...,argn) expression.

Приклад:

```
sqr = @(x) x.^2
len = @(x,y) sqrt(x.^2 + y.^2)
```

3.2. Особливості програмного режиму в Octave

Script-файл (файл скрипту) — це файл, який містить (майже) будь-яку послідовність команд Octave. Файл скрипту читається і оцінюється так само, якщо введено кожну команду в командному рядку Octave, і надає зручний спосіб виконання послідовності команд, які логічно не належать функції [1—2].

На відміну від файлу функції, **файл скрипту не повинен починатися з ключового слова функції**. Навіть якщо файл скрипту не починається з функції ключового слова, можна визначити більше однієї функції в одному файлі скрипту і завантажити (але не виконати) всі з них одночасно. Якщо це так, то Octave вважатиме, що це файл функції, і що визначає одну функцію, яка повинна бути

оцінена відразу після її визначення. **Файл сценарію** також відрізняється від файлу функції тим, що змінні, вказані у файлі сценарію, не є локальними змінними, а знаходяться в тій же області дії, що інші змінні, видимі в командному рядку.

Так, для розв'язку системи лінійних алгебраїчних рівнянь у вікні інтерпретатора Octave послідовно введемо такі команди:

```
>> %Визначення матриці
>> % Коефіцієнти системи лінійних рівнянь
>> A=[7 9 11;4 -5 8;13 -10 21];
>> % вектор правих частин СЛАР
>>  b=[15;31;28];
>> %Рішення системи методом зворотної матриці
>> x=A^(-1)*b
x =
  -21.7429
    0.3143
   14.9429
>>% Перевірка
>>A * x
ans =
   15
   31
   28
```

Виконання розв'язку системи лінійних алгебраїчних рівнянь у програмному режимі (у вікні Editor (Редактор)):

```
1
2  %%Рішення СЛАР методом зворотної матриці
3
4  A=[7 9 11;4 -5 8;13 -10 21]%% Коефіцієнти системи лінійних рівнянь
5  b=[15;31;28];% вектор правих частин СЛАР
6  x=A^(-1)*b%Рішення системи методом зворотної матриці
7  A*x%Перевірка
8
```

```
>> SLAE
A =
    7    9   11
    4   -5    8
   13  -10   21
```

```
x =
  -21.7429
    0.3143
   14.9429

ans =
   15
   31
   28
```

Використання командного рядка, як правило, виправдане лише при невеликих програмах. Якщо ж алгоритм вирішення завдання досить громіздкий і його необхідно налагоджувати, то є сенс скористатися **скриптами**.

Скрипт є звичайним текстовим файлом з **розширенням *.m**, що містить будь-який допустимий системою набір команд. Це **аналог модуля** (власне звідси назва розширення файлу) [1—3].

Важливо: у GNU Octave розрізняються m-файли двох різновидів: **файли-сценарії (Script Files)** та **Функції GNU Octave**. Основна відмінність: у вигляді **скриптів** оформляються основні **програми**, які управляють від початку до остаточного виконання процесу обчислення. **Функції** — окремі **підпрограми**, розраховані, переважно, на неодноразове використання. **Файли-сценарії не повинні починатися з ключового слова function!**

При написанні скриптів Octave у ряді випадків зручно використовувати функції очищення. До таких функцій відносяться: clc; - Очищення командного вікна; clear all; — очищення області змінних та імен функцій користувача; close all; — Закриття раніше відкритих вікон графіка. Ці три функції слід розміщувати на початку скрипту.

Приклад файлу сценарію з підпрограмами розрахунку електростатичного поля силового кабелю коаксіальної конструкції з однорідною ізоляцією представлено нижче.

Коаксіальним кабелем називається система двох циліндричних провідників різного радіуса: r_1 — внутрішній (у разі силового кабелю — струмопровідна жила), r_2 — зовнішній (у разі силового кабелю — металевий екран), розміщених один усередині іншого так, що їхні осі збігаються (рис.3.3).

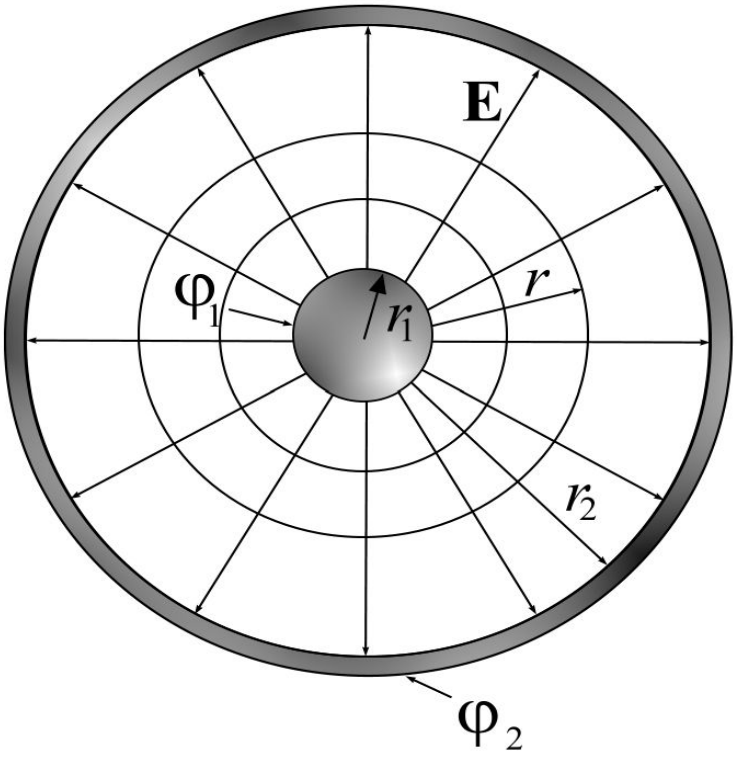


Рисунок 3.3 — Модель коаксіального кабелю з силовими лініями електростатичного поля та еквіпотенціальними поверхнями

Величина потенціалу φ не змінюється вздовж осьової координати Z у циліндричній системі координат. Через циліндричну симетрію електростатичного поля величина потенціалу не змінюється і вздовж координати Θ.

Рівняння Лапласа та його розв'язання для напруженості E електростатичного поля мають вигляд:

$$\frac{1}{r}\frac{d}{dr}\left(r\frac{d\varphi}{dr}\right)=0 \quad , (3.1) \qquad E=\frac{\varphi_1-\varphi_2}{r\log\frac{r_2}{r_1}}=\frac{U}{r log\frac{r_2}{r_1}} \quad , (3.2)$$

де φ_1, φ_2 — потенціали внутрішнього провідника (струмопровідної жили) та металевого екрана відповідно.

За умови, що потенціал зовнішнього провідника дорівнює нулю $\varphi_2=0$ (металевий екран заземлено) потенціал внутрішнього провідника дорівнює прикладеній напрузі U: $\varphi_1=U$.

На рис.3.4 подано розрахункову модель з вузловими точками для визначення значень напруженості електростатичного поля по товщині однорідної ізоляції (рис. 3.5) силового кабелю на лінійну напругу 20 кВ.

```
clear all;
Ul=20e+03;Uf=20e+03/sqrt(3);%Лінійна та фазна напруги, В
S=240e-06;%Переріз струмопровідної жили, м2
r1=sqrt(S/pi)%радіус струмопровідної суцільної жили, м
deli=6e-03;%Товщина ізоляції, м
r2=r1+deli%Радіус по ізоляції, м
X1=0; Y1=0;  N1=400; %Координати центру жили та число вузлів
%для побудови струмопровідної жили
X2=0; Y2=0;  N2=400;  N=N1+N2; %Координати центру екрана та
%число вузлів для його побудови та загальна кількість вузлів (точок)
[X1, Y1, DL1]= OCL(X1,Y1,r1,N1);%Підпрограма для визначення
%елементарної довжини дільниці для жили
[X2, Y2, DL2]= OCL(X2,Y2,r2,N2);%Підпрограма для визначення
%елементарної довжини дільниці для екрану
X=[X1; X2]; Y=[Y1; Y2];%Загальна матриця координат
figure(1),plot(X,Y,'bo-'),hold on;axis equal;%Візуалізація
%розрахункової моделі кабелю з відповідною кількістю вузлів
Esr=Uf/(r2-r1) %Середнє значення напруженості поля
% Формування циклу для визначення напруженості поля по
%товщині ізоляції
N=50;
for i=1:N
rt(i)=r1+(r2-r1)*(i-1)/N% Поточне значення радіуса,м
```

```
Et(i)=Uf/(rt(i)*log(r2/r1));%Поточне значення напруженості поля, %В/м
    end% завершення циклу визначення напруженості %електростатичного поля по товщині ізоляції
figure(2), plot(rt,Et,'rx-','markersize', 6,'LineWidth',6);grid on; hold on; %Побудова графіка розподілення напруженості електростатичного %поля по товщині ізоляції
set(gca,'FontSize',24,'fontangle','Normal','fontname','Times New Roman','LineWidth',6);
    xlabel('rt, m');set(gca,'FontSize',24);
    ylabel('E, V/ m');set(gca,'FontSize',24);
```

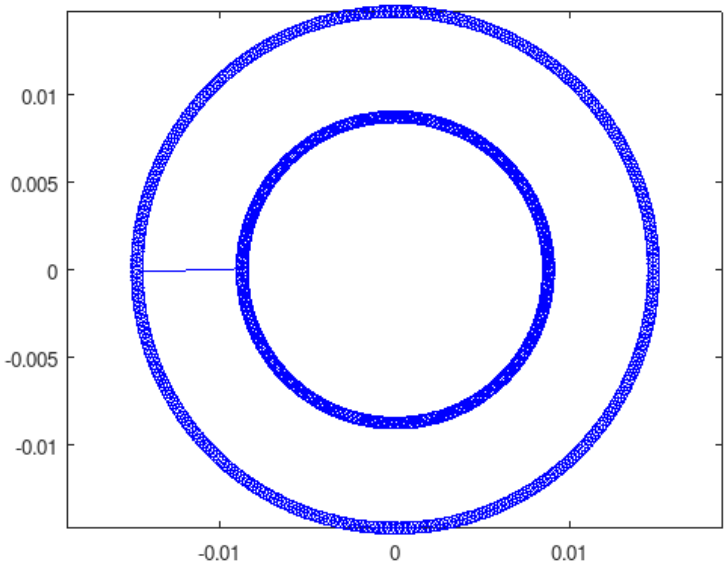


Рисунок 3.4 — Розрахункова модель силового кабелю

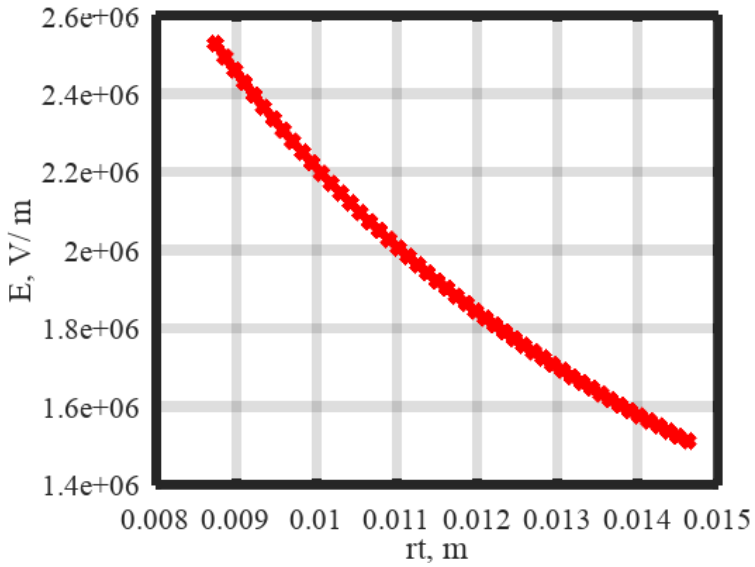


Рисунок 3.5 — Залежність напруженості електростатичного поля по товщині ізоляції силового кабелю напруги 20 кВ

3.3. Базові основи програмування в Octave

Цей короткий розділ не може і не охоплюватиме всі аспекти програмування за допомогою Octave. Кількість доступної літератури величезна та безкоштовна [1—2].

Одним із **основних операторів**, призначених для розгалуження в більшості мов програмування, є **умовний оператор**.

Оператор **if** — серце логіки Octave. Почнемо з дуже простого прикладу:

```
>> a = true
a=1
>> if(a)
>
```

Зверніть увагу, що консоль для введення змінює символи «>>» на символ «>» або відсутність у програмі GUI. Це означає, що починається наступна частина оператора **if** — це його тіло. Тіло умови виконується тільки в тому випадку, якщо логічний вираз у визначенні оператора **if** істинно. В даному випадку a=true, тому інструкції будуть виконуватися. Але в більшості випадків не можна заздалегідь знати істинно логічний вираз або хибно.

При налаштуванні програми у вигляді командного рядка можливо використовувати ряд операторів **if** для виконання операції під час виконання певних умов.

Синтаксис для програм у текстовому файлі такий самий:

```
# example.m
a = true
 b = false
 if(a)
   «a = true!»
endif
 if(b)
   «b = true!»
endif
```

Оператор **else** визначає блок операторів, який виконається, якщо логічний вираз у визначенні оператора **if** помилково:

```
>> if(a)
'a true!'
else
'a is false ...'
endif
ans = a is false ...
```

У кожному операторі **if** обчислюється логічний вираз. Якщо вираз істинний, то виконується тіло цього оператора **if** . Якщо це хибно, програма переходить до наступного оператора **if** . Якщо жодне з логічних виразів не істинно, виконується тіло оператора **else** (якщо воно є).

Існує звичайна, скорочена та розширена форма цього оператора в мову програмування Octave. Звичайний умовний оператор має вигляд:

```
if умова
оператори1
else
оператори2
end
```

Тут умова — логічний вираз, оператори1, оператори2 - оператори мови або вбудовані функції Octave. Звичайний оператор **if** працює за наступним алгоритмом: якщо умова істинна, то виконуються оператори1, якщо хибно — оператори2. Розширений умовний оператор застосовують, коли однієї умови для прийняття рішення недостатньо:

```
if умова1
оператори1
elseif  умова2
оператори2
…
elseif умова n
операториN
```

```
else
оператори
end
```

Розширений оператор **if** працює в такий спосіб. Якщо умова 1 істинно, виконуються оператори 1, інакше перевіряється умова 2, якщо воно істинно, виконуються оператори 2, інакше перевіряється умова N. Якщо жодна з умов по гілках **elseif** не виконується, то виконуються оператори по гілці **else**.

Оператор циклу з передумовою у мові програмування Octave має вигляд:

```
while вираз
оператори
end
```

Працює цикл із передумовою в такий спосіб. Обчислюється значення виразу. Якщо воно є істинним, виконуються оператори. В іншому випадку цикл закінчується, і керування передається оператору, що йде за тілом циклу. Вираз обчислюється перед кожною ітерацією циклу. Якщо при першої перевірки вираз хибно, цикл не виконається жодного разу. Вираз має бути змінним чи логічним.

Для запису циклу з певною кількістю повторень застосовують оператор

```
for параметр = початкове_значення:крок:кінцеве_значення
оператори
end
```

Виконання циклу починається з присвоєння параметру початкового циклу значення. Потім слідує перевірка, чи не перевершує параметр циклу кінцеве значення. Якщо результат перевірки ствердний, цикл вважається завершеним, і керування передається наступному за тілом циклу оператору. В іншому випадку оператори виконуються в циклі. Далі параметр змінює своє значення на значення кроку. Знову проводиться перевірка значення параметра

циклу та алгоритм повторюється. Якщо крок циклу дорівнює 1, то оператор записують так

```
for параметр = початкове_значення:кінцеве_значення
оператори
end
```

Оператори передачі керування примусово змінюють порядок виконання команд. У мові програмування Octave таких операторів є два. Оператори **break** і **continue** використовують лише усередині циклів. Так, оператор **break** здійснює негайний вихід із циклів **while**, **for** та управління передається оператору, що перебуває безпосередньо за циклом. Оператор **continue** починає нову ітерацію циклу, навіть якщо попередня не була завершено. У мові програмування Octave є безліч функцій роботи з рядками (табл.3.1).

У файлі з розширенням .m, крім основної функції, ім'я якої збігається з ім'ям файлу, можуть бути так звані підфункції. Ці функції доступні лише всередині файлу. Таким чином, загальну структуру функції можна так:

```
%Початок основної функції m-файлу,
% ім'я котрої повинно співпадати з ім'ям файлу, у котрому
   зберігається
function [y 1 ,y 2 ,…y n ]=name_function(x 1 ,x 2 ,…,x m )
% Серед операторів основної функції можуть бути оператори
% виклику підфункцій f1, f2, f3, …,fl
оператор1;
оператор2;
…
операторm;
end; % закінчується основна програма
function [y1,y2,…yn ]=f1(x1,x 2,…,x m ) % початок першої підфункції
операторі
end % кінець першої підфункції
function [y1,y2,…yn ]=f2(x1,x2 ,…,xm ) % початок другої підфункції
операторі
end % кінець другої підфункції
…
```

function [y1,y2 ,…yn]=fn(x1,x 2,…,x m) % початок n-ї підфункції
оператори
end % кінець n-ї підфункції підфункції

Така структура близька до структури програм мовою C+. Вона не допускає вкладеності функцій одна в одну.

Однак у Octave можливий і інший синтаксис, у якому дозволено використання вкладених функцій.

Оператор кінця блоку може залежати від того, якого типу цей блок: замість одного end в MATLAB в Octave рекомендується використовувати endif, endfor, endwhile і т.п.

Таблиця 3.1 — Формати подання дати

Функція	Пояснення
char(node)	Повертає символ за кодом code
deblank(s)	Формується новий рядок шляхом видалення прогалин наприкінці рядку s
int2str(x)	Перетворення чисел, що зберігаються в масиві (матриці) **x** до цілого типу та запис результатів у масив символів
findstr(str,substr)	Повертає номер позиції, починаючи з str
sprintf(format, x)	Формує рядок з чисел, що зберігаються в числовому змінної x відповідно до формату format (sprintf 'X= %4.2e',x)
sscanf(s, format)	Функція повертає з рядка **s** числове значення чи масив значень відповідно до формату

Octave менш суворий до правил розбиття довгих команд на кілька рядків: явно вказувати розбиття необхідно лише в тому випадку, коли трактування рядків неоднозначне.

В Octave є можливість передавати ім'я функції як вхідний параметр, що суттєво розширює можливості програмування. Ім'я

функції передається як рядок, а її обчислення здійснюється за допомогою функції **feval**. Функція **feval** надає альтернативний спосіб обчислення значення функції. Параметрами функції **feval** є: рядок з іменем викликаної функції, як ім'я може бути вбудована функція чи визначена користувачем функція; параметри цієї функції розділені комою.

Для вимірювання часу роботи блоку коду можна скористатися командами **tic** та **toc**. Відбувається вимір часу блоку коду, розташованого між рядками з цими функціями:

```
tic
блок коду
toc
```

Іноді необхідно переглянути інформацію по всіх змінних або за якоюсь певною змінною. Це можна зробити через область змінних (див. рис.1.1) або скористатися командою **who** або **who()**, яка виводить список певних змінних, або командою **whos** або **whos()** — список змінних із зазначенням їх розміру та обсягу займаної пам'яті.

Наступний приклад ілюструє дію цих команд

```
>> A=[1 2; 3 4];
>> b='line';
>> c=5;
>> d=1+i;
>> who
Variables in the current scope:
A b c d
>> whos
Variables in the current scope:
   Attr        Name        Size              Bytes  Class
   ====        ====        ====              =====  =====
               a           2x2                  32  double
               b           1x4                   4  char
               c           1X1                   8  double
    c          d           1X1                  16  double
Total is 1 element using 16 bytes
```

Контрольні запитання до розділу 3

1. У який спосіб забезпечується принцип модульності у GNU Octave?
2. Яка обов'язкова умова визначення функції у GNU Octave?
3. Як саме визначається коментар у програмному коді?
4. Які ознаки має файл скрипту?
5. Чим відрізняється файл сценарію від файлу функції?
6. Які ознаки має файл сценарію?
7. В чому полягає необхідність застосування функцій очищення у GNU Octave?
8. Назвіть особливості умовного оператору.
9. У яких випадках застосовують оператор **else**?
10. Чим обумовлено застосування функція **feval**?
11. Чим відрізняються команди who (who()) та whos (whos())?
12. У чому полягає особливість операторів break і continue ?

Розділ 4. Візуалізація скалярних та векторних полів

Поняття поля має важливе значення для опису процесів, що вивчаються у фізиці, електродинаміці, у тому числі електроізоляційній, кабельній та оптоволоконній техніці. Розрізняють **скалярні** та **векторні** поля [9—10, 12].

При заданому полі в деякій області Ω простору кожній точці P відповідає певне значення деякої скалярної чи векторної величини. Якщо кожній точці P деякої області Ω простору поставлено у відповідність значення скалярної функції $u(P)$, то поле називається **скалярним.** У прямокутній системі координат положення точки описується її декартовими координатами x, y, z. Отже, завдання скалярного поля u рівнозначне завдання скалярної функції $u=u(x, y, z)$ трьох змінних. Найпростішим прикладом скалярного поля є **температурне поле** неоднорідно нагрітого тіла, оскільки кожній точці тіла можна поставити у відповідність певне значення температури. Прикладами скалярного поля є густина **розподілення електричного заряду**, **потенціал системи заряджених часток**.

Завдання **векторного поля** в області Ω простору означає, що в ній визначено векторну функцію $\vec{F}(P)$ положення точки $P(x, y, z)$ у просторі декартової системи координат:

$$\overline{F}(P) = P(x,y,z)\overline{i} + Q(x,y,z)\overline{j} + R(x,y,z)\overline{k} \ , \qquad (4.1)$$

де $P(x,y,z)$, $Q(x,y,z)$ *та* $R(x,y,z)$ — скалярні функції,

$\vec{i}, \vec{j}, \vec{k}$ - одиничні вектори (орти) у напрямку осей x, y та z у тривимірному випадку.

Так, наприклад, в області, заповненій рідиною, що тече з деякою

швидкістю з різною в різних точках (так звані не ньютонівські рідини: розплав полімерів в екструдері при нанесенні ізоляції на струмопровідну жилу) кожній точці можна поставити у відповідність **вектор швидкості. У такому випадку** вийде **векторне поле швидкостей рідини**, що рухається.

На одиничний електричний заряд, поміщений у точку, діють з певною силою електричні заряди, розподілені в деякій області. Саме ці сили (**напруженість електричного поля Е**) утворюють **векторне поле**, яке називається **електростатичним полем** - поле, що створюється нерухомими у просторі і незмінними в часі електричними зарядами.

Скалярне або векторне поле називаються **стаціонарним**, якщо величина, що характеризує поле, залежить тільки від положення точки у просторі, але не залежить від часу. Якщо ж аналізована величина залежить також і від часу, то поле називається **нестаціонарним**.

4.1. Поверхні і лінії рівня скалярного поля

Часто функцію $u(P)$, що задає скалярне поле, незалежно від її фізичного змісту, називають **потенціалом**.

Для **наочного зображення скалярного поля** використовуються поверхні рівня і лінії рівня.

Рівняння поверхні рівня скалярного поля $u(P)$ має вигляд

$$u(x, y, z)=C. \qquad (4.2)$$

Очевидно, що жодні дві поверхні рівня $u(x, y, z)=C_1$ і $u(x, y, z)= C_2$ не можуть мати загальних точок, якщо $C_1 \neq C_2$. В іншому випадку це означало б, що поле одночасно приймає різні значення в одній і тій

же точці простору.

Поверхня рівня являє собою безліч всіх точок простору з однаковими значеннями скалярного поля $u(P)$. У будь-якій точці поверхні рівня поле u приймає одне й те саме значення C.

Оскільки $u=u(x, y, z)$ — однозначна функція, то кожній точці відповідає одне значення функції, тому через кожну точку поля проходить тільки одна поверхня рівня.

Поверхні рівня називають також **еквіпотенціальними поверхнями**, тобто поверхнями рівного потенціалу.

Лінії рівня використовуються для зображення **скалярного поля**, заданого в пласкій області Ω. Кожна лінія рівня являє собою безліч точок площини з однаковими значеннями поля. Якщо область Ω розташована в площині x0y, то лінії рівня скалярного поля u описуються рівнянням $u(x, y)=C$ (рис.4.1, рис. 4.2).

За допомогою ліній рівня зображують, наприклад, рельєф місцевості на географічних картах, з'єднуючи одну криву точки, що мають однакову висоту над рівнем моря.

Для деяких поверхонь та ліній рівня використовуються спеціальні назви:

поверхні постійної температури називаються **ізотермічними або ізотермами**;

поверхні постійного тиску називаються **ізобаричними або ізобарами**;

поверхні постійного потенціалу називаються **еквіпотенційними** (рис.4.1, рис. 4.2);

поверхні постійної енергії називаються **ізоенергетичними**.

Еквіпотенціальні лінії знаходять широке застосування при розгляді плоско паралельних електричних полів [13—15].

Еквіпотенціальні лінії є лініями, вздовж яких електричний

потенціал φ — **енергетична характеристика електричного поля**, яка показує, яку **потенційну енергію** має одиничний позитивний заряд у даній точці, в усіх точках постійний. Фізично це означає, що робота, яка виконується силами електричного поля по перенесенню одиничного позитивного заряду з довільної точки даної лінії рівня в точку, потенціал якої прийнятий рівним нулеві, буде однакова. Робота з переміщення заряду уздовж еквіпотенціальної поверхні дорівнює нулю.

Еквіпотенціальні лінії зображують одновимірні області, в яких електричний потенціал, створений одним або декількома сусідніми зарядами, має постійну величину. Це означає, що якщо заряд знаходиться в будь-якій точці на заданій рівнопотенціальної лінії, не буде потрібно ніяких робіт, щоб перемістити його з однієї точки в іншу на тій же лінії.

Еквіпотенціальні лінії стаціонарного електричного поля замкнені ($rot\,\dot{E}=rot\,(grad\,\varphi)=0$).

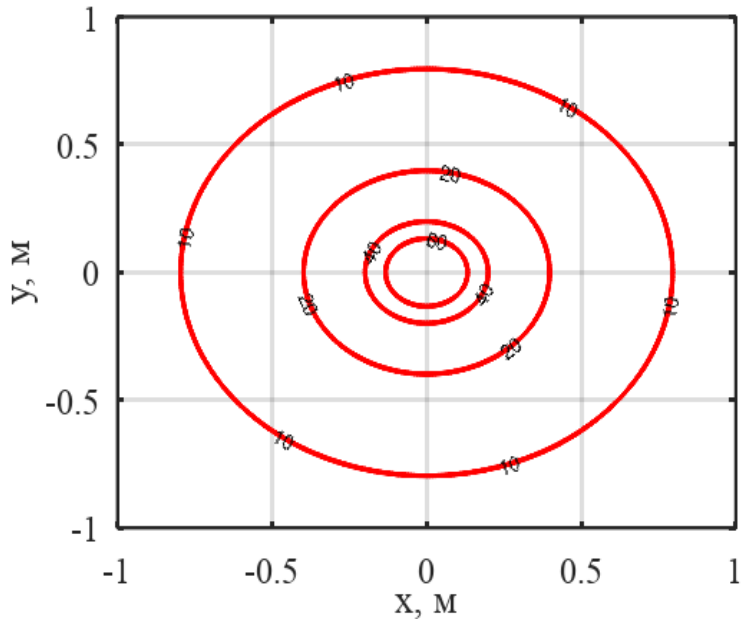
Еквіпотенціальні лінії стаціонарного магнітного поля за умови: густина струму $\vec{\delta}=0$ та $rot\,\dot{H}=rot\,(grad\,\varphi_m)=0$ (φ_m — магнітний потенціал), - починаються і закінчуються на струмах (аналогічно як і силові лінії електричного поля починаються і закінчуються на зарядах).

Візуалізацію засобами Octave електростатичного поля **точкового заряду** виконано для потенціалу φ (рис.4.1): скалярне уявлення простіше за векторне.

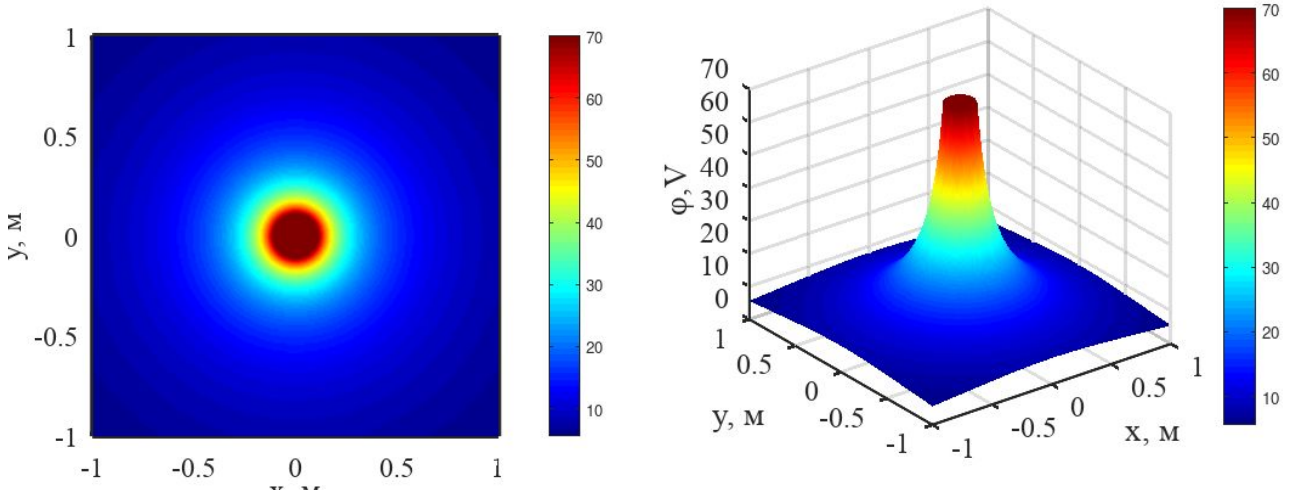
Потенціал точкового заряду q у вакуумі визначається на підставі:

$$\varphi=\frac{q}{4\,\pi\varepsilon_0 r}\ , \qquad (4.3)$$

де ε_0 =8,85 10^{-12} Ф/м — електрична стала, r — відстань від заряду-джерела до точки, для якої розраховується потенціал.



а



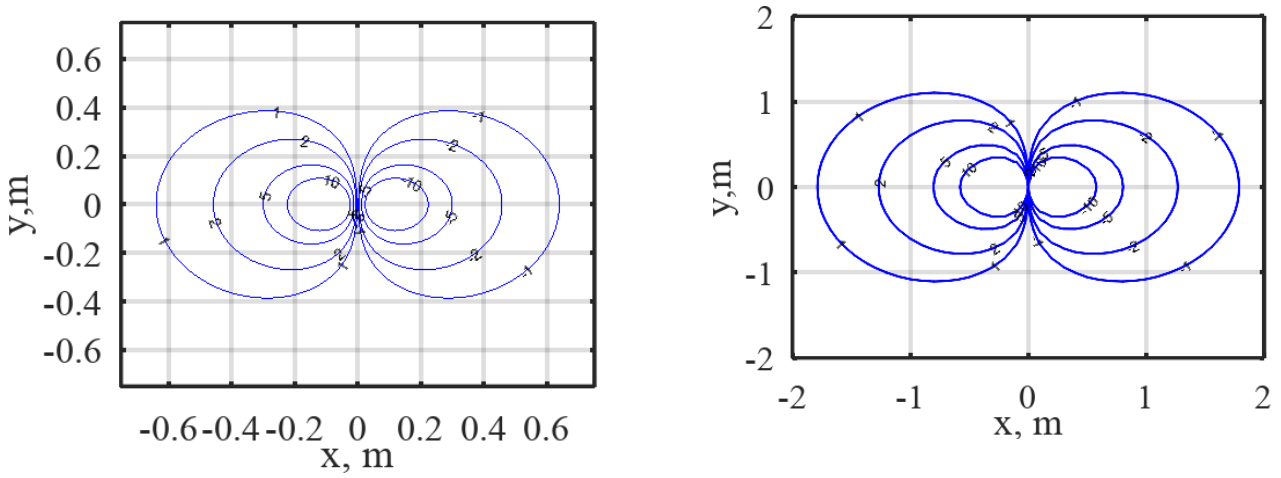
б

Рисунок 4.1 — Візуалізація еквіпотенційних ліній (*а*) та поверхонь (*б*) потенціалу φ точкового електричного заряду 100 пКл

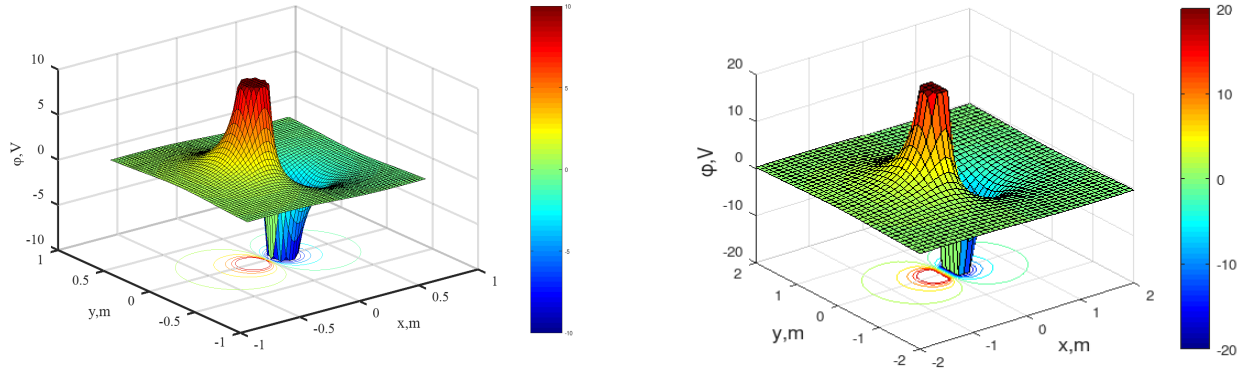
За умови залежності поля $\varphi=\varphi(x,y)$ від двох змінних поле є пласким. Симетрія задачі дозволяє тривимірному простору перейти до площини: точковий електричний заряд розташовано на початку

координат. Для побудови на площині функції $\varphi=\varphi(x,y)$ застосовано команду **meshgrid,** яка формує сітку вузлів на площині у вигляді двовимірних масивів x та y. Команда **contour** дозволяє побудувати лінії рівня для масиву даних електричного потенціалу φ, що містить значення потенціалу у вузлах сітки.

На рис. 4.2 представлено візуалізацію електростатичного поля диполя за допомогою команд **meshgrid** та **contour**. Сукупність ліній рівного потенціалу φ дає наочне зображення електричного поля (рис. 4.1, рис. 4.2), що полегшує його вивчення.



а



б

Рисунок 4.2 — Еквіпотенційні лінії (*а*) та поверхні електричного потенціалу φ (*б*) електричного диполя із зарядом 25 пКл - ліворуч і 200 пКл — праворуч

Електричний диполь являє ідеалізовану систему, що складається з точкових та рівних за абсолютною величиною позитивного та негативного електричних зарядів. Диполь - електричне нейтральний, проте породжує електричне поле. З урахуванням **принципу суперпозиції електростатичних полів** потенційна енергія електростатичної взаємодії системи зарядів розраховується на підставі її обчислення кожної пари зарядів [16—18].

В заданій точці простору різні заряджені частинки створюють власні електричні поля: загальна напруженість поля в цій точці дорівнює геометричній сумі напруженостей полів частинок. **Принцип суперпозиції (накладання) полів** означає, що електричні поля під час накладання не впливають одне на одне. Принцип суперпозиції дозволяє обчислити напруженість поля довільної системи зарядів, а не тільки точкових.

Для електростатичного поля рівняння Максвелла у вакуумі — лінійні. Електродинамічний принцип суперпозиції не є непорушним законом природи, а є усього лише наслідком лінійності рівнянь Максвелла, тобто рівнянь класичної електродинаміки. Порушення принципу суперпозиції спостерігається, зокрема, при розповсюдженні електромагнітних хвиль оптичного діапазону у оптичних волокнах, що генеруються лазером. За певної інтенсивності (напруженості електричної складової електромагнітної хвилі) у оптичному волокні виникають нелінійні ефекти.

4.2. Поле градієнтів

Векторне поле може мати дуже складний характер, змінюючись за величиною та напрямом при переході від однієї точки до іншої. Для наочного представлення векторного поля можна накреслити у різних точках простору вектори, що показують напрями поля у цих точках

(рис.4.3).

Якщо скалярні поля описуються лініями та поверхнями рівня, то форму векторного поля можна охарактеризувати векторними лініями.

Для геометричної характеристики векторного поля розглядають векторні лінії.

Векторною* (силовою) *лінією поля $\overline{F}(P)$ ***називається лінія,*** *дотична до якої в кожній її точці збігається з вектором* $\overline{F}(P)$ *, що визначає поле в цій точці.*

Так, **силова лінія електричного поля** $\vec{E}$ — уявно проведена в полі лінія, дотична до якої в кожній точці збігається з напрямом вектора напруженості електричного поля (або вектора магнітної індукції $\vec{B}$ (вектора напруженості магнітного поля $\vec{H}$) у разі магнітного поля). Лінії вектора $\vec{E}$ називають силовими лініями внаслідок того, що напруженість електричного поля чисельно дорівнює силі, яка діє в даній точці поля на одиничний додатній заряд. Напрямки вектора цієї сили та вектора $\vec{E}$ завжди збігаються.

Силові лінії електричного поля починаються на тілі, зарядженому додатно, і закінчуються на негативно зарядженому, що випливає із рівняння Максвелла (див. розділ 4.4)

$$div\,\vec{D}=\rho \ , \qquad (4.4)$$

де $\vec{D}=\varepsilon_0*\varepsilon*\vec{E}$ — вектор електричного зміщення, ε_0=8,85 10^{-12} Ф/м — електрична стала, ε — діелектрична проникність середовища.

Силові лінії магнітного поля замкнені самі на себе, тому що $div\,\vec{B}=0$.

Для додатного заряду векторними (силовими) лініями будуть промені, що виходять з заряду. Слід зауважити, що ці лінії є не

обов'язково прямими. Якщо поле створюється не одним, а кількома зарядами, то лінії напруженості поля мають форму кривих (рис. 4.3).

Для магнітного поля векторними лініями будуть лінії, що виходять з північного полюса і закінчуються в південному.

Просторові області, цілком складені з векторних ліній, називають векторними трубками. В кожній точці P поверхні векторної трубки вектор $\boldsymbol{a}(M)$ лежить на дотичній площині до поверхні.

Градієнт являє векторну величину, яка визначає в кожній точці простору не лише швидкість зміни, а й напрямок найшвидшої зміни функції, що залежить від координат. При замінах координат градієнт перетворюється інакше, ніж вектор, і тому його не можна розглядати як справжній вектор.

У математиці градієнтом скалярної функції називають швидкість зміни функції, взяту у бік її найбільшого зростання.

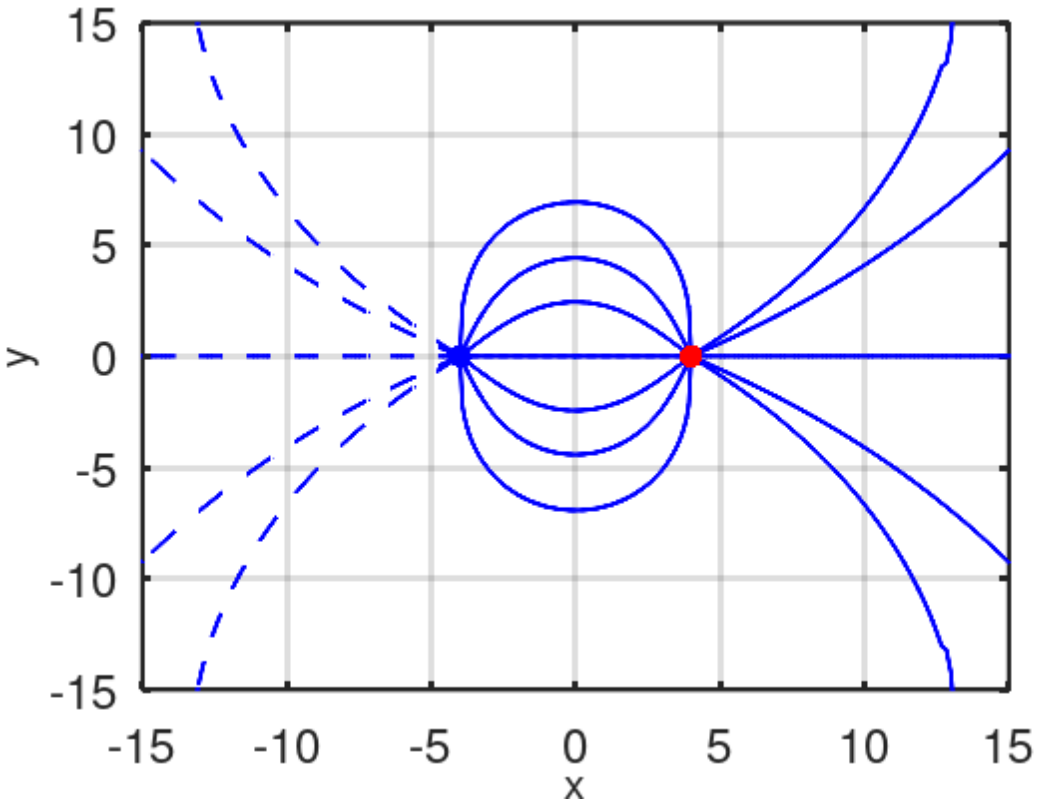


Рисунок 4.3 — Силові лінії системи двох точкових зарядів

Для скалярного поля $u = u(x, y, z)$ з частковими похідними $\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial u}{\partial z}$ вектор $\frac{\partial u}{\partial x}\bar{i} + \frac{\partial u}{\partial y}\bar{j} + \frac{\partial u}{\partial z}\bar{k}$ є **градієнтом** (**полем градієнта**) скалярного поля і позначається:

$$grad\ u = \nabla u = \frac{\partial u}{\partial x}\bar{i} + \frac{\partial u}{\partial y}\bar{j} + \frac{\partial u}{\partial z}\bar{k}\ , \tag{4.5}$$

де $\nabla = \left(\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial u}{\partial z}\right)$ (4.6)

— оператор Гамільтона або оператор набла — векторний диференціальний оператор першого порядку, компоненти якого є частковими похідними за координатами.

Операцію переходу від скалярного поля до його градієнта та операцію переходу від векторного поля до його дивергенції часто позначають Гамільтона оператором.

Градієнт скалярного поля — вектор, у напрямі якого похідна поля максимальна і дорівнює

$$|grad\ u| = \sqrt{\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial u}{\partial z}\right)^2}\ . \tag{4.7}$$

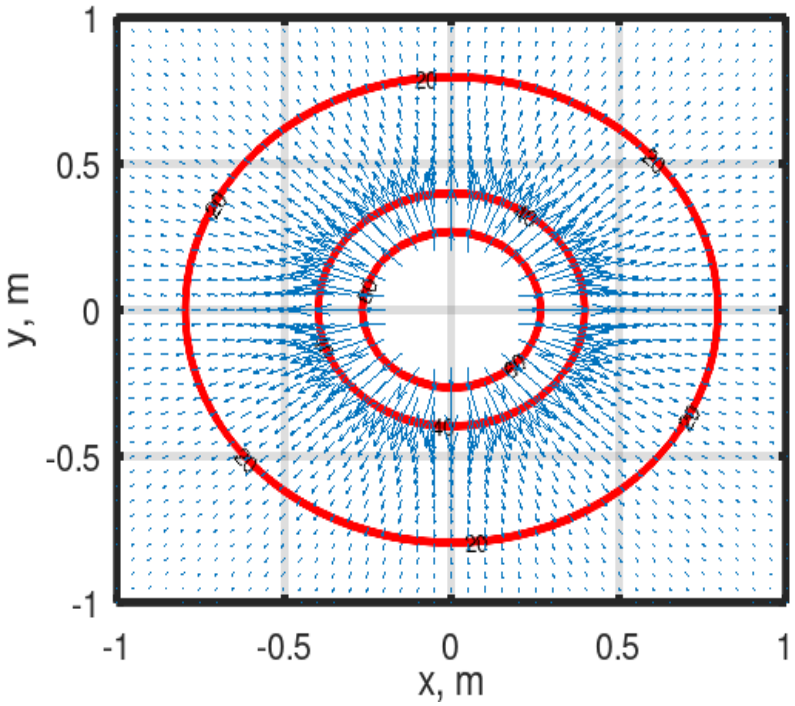
У цьому полягає фізичний зміст градієнта: напрям градієнта є напрямок якнайшвидшого зростання функції.

Векторною лінією поля градієнта або лінією градієнта є крива, дотична до якої в кожній точці збігається з напрямом grad u.

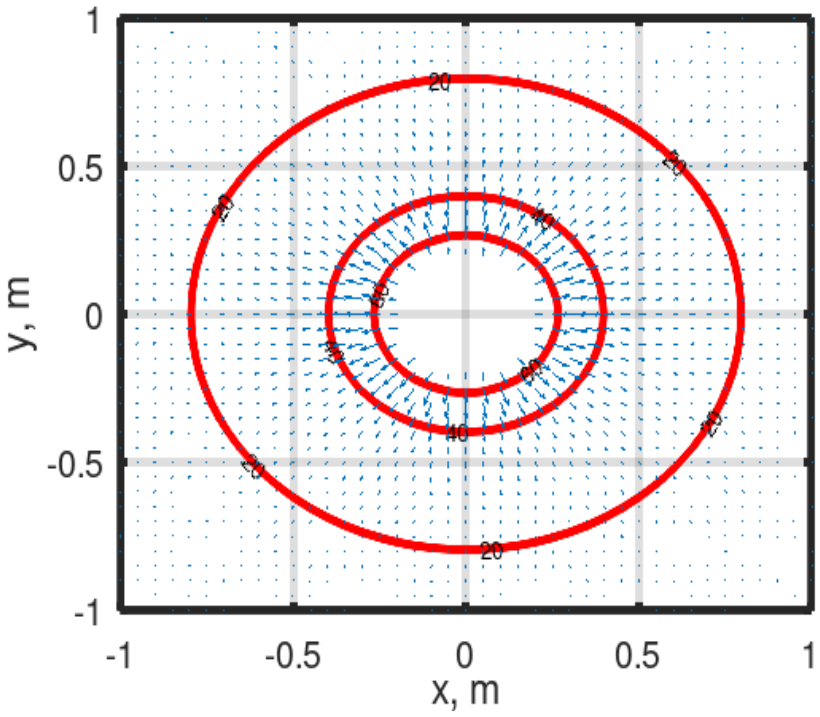
З властивостей градієнта поля u випливає, що лінія градієнта — це крива, вздовж якої функція $u(x, y, z)$ максимально зростає (або максимально спадає).

Крім того, **лінії градієнта завжди перпендикулярні до**

поверхні рівня поля *u* (рис.4.4).



а



б

Рисунок 4.4 — Приклад одночасного зображення скалярного у вигляді ліній рівня потенціалу та векторного у вигляді векторів напруженості електростатичного поля одиночного електричного заряду 200 пКл

Формула (4.5) — **інваріантна**: результат не залежить від того, в якій системі координат представити.

Важливо, що операція «градієнт» застосовується до скалярного полю, а в результаті отримаємо векторне поле.

Градієнт *u(x,y,z)* — векторне поле з безліччю силових ліній: стрілок відповідної довжини та напрямком.

При зміні системи координат стрілки не зміняться. Зміняться координати цих стрілок у новій системі координат: вектори, як об'єкти, залишаться незмінними.

На площині математичний зміст градієнта полягає у завданні в кожній точці (*x,y*) напряму, у якому функція *u(x, y)* зростає найшвидше. Абсолютне значення градієнта (тобто довжина вектора в кожній точці) визначає локальну швидкість зміни *u(x, y).*

Операція градієнт застосовується до багатьох фізичних законів або подання рівнянь, що пов'язують величини. Наприклад, для електростатичного поля, яке характеризується силовою характеристикою — напруженістю $\vec{E}$ та енергетичною характеристикою – потенціалом φ, справедливо:

$$\vec{E} = -grad(\varphi) . \qquad (4.8)$$

Так, у центрі показаної області (*x,y*) на рис. 4.4 напруженість електростатичного поля змінюється швидко: значення градієнта максимальні, при віддаленні від центру функція *E(x,y)* змінюється повільно. Вектор напруженості $\vec{E}$ спрямований туди, де потенціал змінюється найкрутіше. Модуль $\vec{E}$ тим більше, чим швидше цей потенціал змінюється.

Важливо: еквіпотенціальні і силові лінії в будь-якій точці поля перетинаються під прямим кутом (рис.4.4).

Для побудови графіків полів градієнта застосовуються команди **quiver**:

quiver(X,Y,U,V) — будує графік поля градієнтів як стрілок кожної пари елементів масивів X і Y, причому елементи масивів U і V вказують напрям і розмір стрілок;

quiver(U, V) — будує вектори швидкості в рівно розташованих точках на площині (x, y);

quiver(U,V,S) або **quiver(X,Y,U,V,S)** — автоматично масштабує стрілки по сітці і витягує їх за значенням S (рис.4.4,*а*). Використовуйте S=0 для побудови стрілок без автоматичного масштабування (рис.4.4,*б*);

quiver(...,LINESPEC) — використовує для векторів вказаний тип лінії. Вказані в LINESPEC маркери малюються біля основ, а не на кінцях векторів. Для скасування будь-якого виду маркера використовуйте специфікацію '.'. Специфікації ліній, кольорів та маркерів були детально описані у розділі, присвяченому команді **plot**;

quiver(.... 'filled') — дає графік із зафарбованими маркерами.

Приклади застосування команди **quiver** наведено на рис.4.5:

рис.4.5, *а, б*

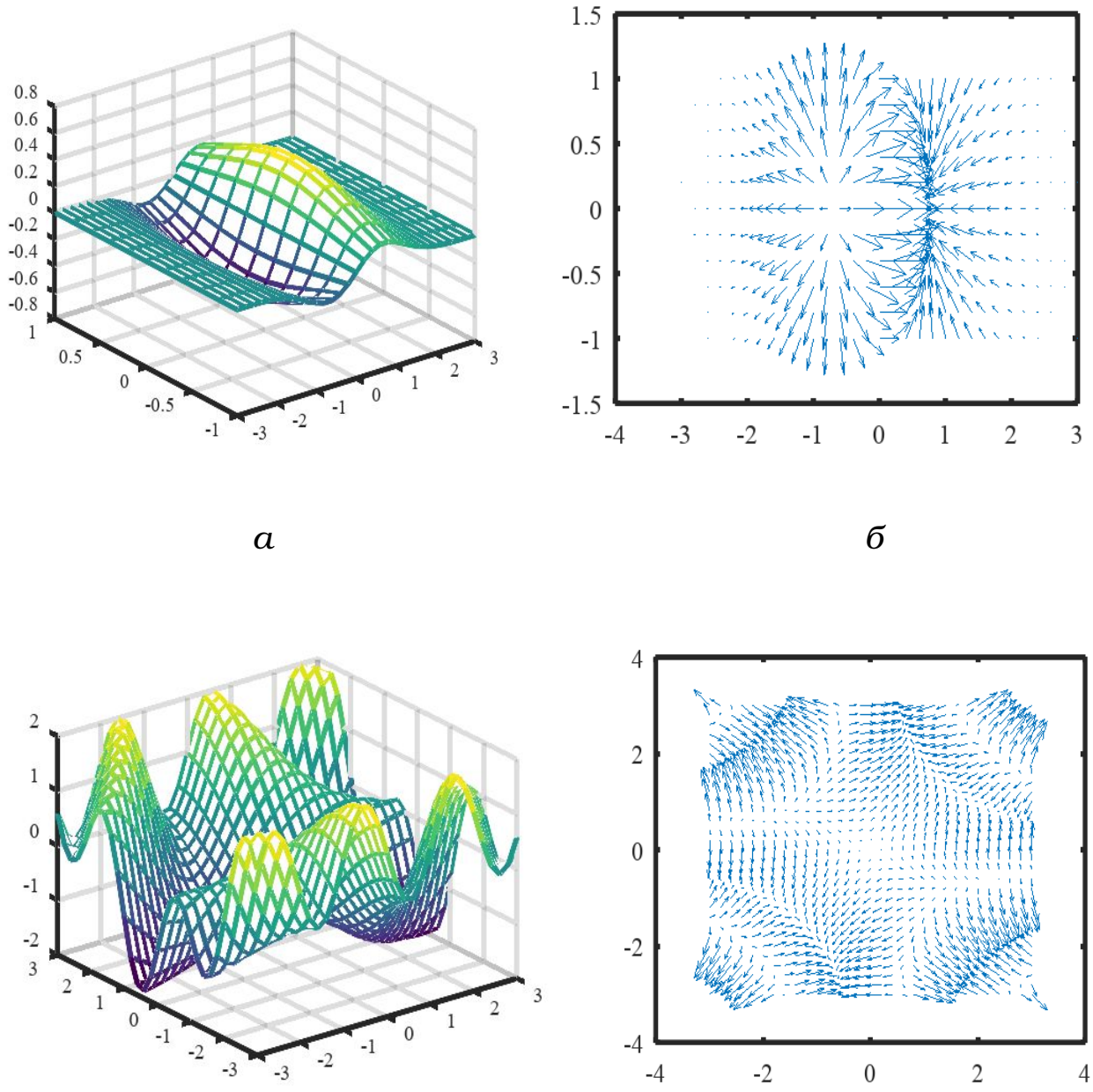
```
>> x = -3:.2:3; y = -1:.2:1;
>> [xx,yy] = meshgrid(x,y);
>> zz = xx.*exp(-xx.^2-yy.^2);
>> [px,py] = gradient(zz,2,2);
>> quiver(x,y,px,py,2);
```

рис.4.5, *в,г*:

```
>>x = -3:.2:3; y = -3:.2:3;
>>>>[xx,yy] = meshgrid(x,y);
>>zz =sin(xx.*yy)-cos(yy);
>>[px,py] = gradient(zz,2,2);
>>quiver(x,y,px,py,2,'filled'),
```

Неважко помітити, що уявлення поля градієнтів стрілками дає

наочне уявлення про лінії поля, вказуючи області, куди ці лінії впадають і звідки вони походять. Можна вважати, що джерела – це точки, звідки векторні лінії починаються, а стоки – точки, де векторні лінії закінчуються (рис.4.5, *б, г*).



а *б*

в *г*

Рисунок 4.5 — Функції (*а,в*) та поле їх градієнтів (*б,г*)

На рис. 4.6 представлено поверхню, лінії рівня функції $z=\exp\left(-\left(x^2+y^2\right)\right)$ та поле її градієнта із застосуванням команди

quiver(X,Y,U,V).

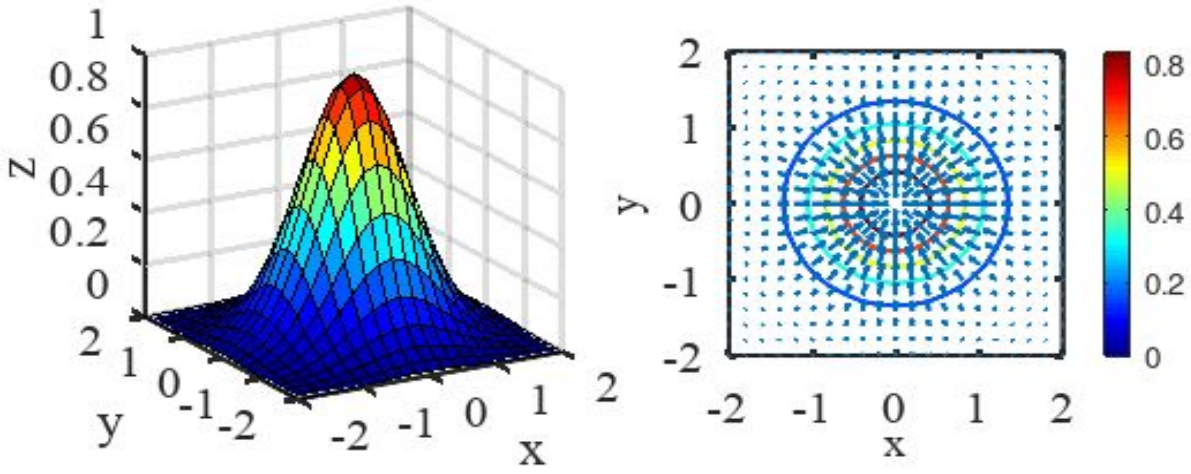


Рисунок 4.6 — Приклад побудови поверхні функції $z=\exp\left(-\left(x^2+y^2\right)\right)$ та її градієнту із застосуванням команд **contour** и **quiver**

4.3. Візуалізація диференціальних характеристик векторного поля

Для опису **скалярних полів** застосовується **градієнт** — вектор, інваріантний щодо вибору системи координат, похідна за напрямом максимальної зміни скалярного поля.

Зміни **векторного поля** в 1-му наближенні характеризуються двома величинами [6-8]:

1. **Дивергенцією (або розбіжністю) поля** — скаляром, який характеризує зміну інтенсивності (щільності) поля,

2. **Вихором (ротором)** поля — вектором, який є векторною характеристикою «обертальної складової» векторного поля (його «скручування»).

Градієнт скалярного поля, дивергенція та вихор векторного поля зазвичай називають основними **диференціальними характеристиками** у теорії поля.

Дивергенція (або розбіжність) — скалярна величина, що

характеризує потужність джерел чи стоків векторного поля у кожній його точці. Вона дорівнює межі потоку векторного поля через замкнуту поверхню, віднесеного до об'єму, укладеного всередині поверхні при стягуванні поверхні в точку. Якщо векторне поле (формула (4.1)) в області задано трьома скалярними функціями *P, R, Q* — проєкціями на координатні вісі, то дивергенцію поля у довільній точці знаходять як скалярний добуток оператору набла (4.6) на (4.1)

$$(\nabla \cdot \overline{F}) = \left(\left(\frac{\partial}{\partial x}\bar{i} + \frac{\partial}{\partial y}\bar{j} + \frac{\partial}{\partial z}\bar{k} \right) \cdot (P\bar{i} + Q\bar{j} + R\bar{k}) \right) = \frac{\partial P}{\partial x} + \frac{\partial Q}{\partial y} + \frac{\partial R}{\partial z} \ . \quad (4.9)$$

Такий добуток — **дивергенція** і позначається $div\ \overline{F}$:

$$div\ \overline{F} = \frac{\partial P}{\partial x} + \frac{\partial Q}{\partial y} + \frac{\partial R}{\partial z}; \quad \nabla \cdot \overline{F} = div\ \overline{F} \ . \quad (4.10)$$

На рис. 4.7 представлено контури рівних значень дивергенції векторного поля $\overline{F}$, яке задано у циліндричній системі координат:

$$\overline{F} = \rho \exp(-\rho/\alpha)^2 \times \dot{i}_\rho, \quad (4.11)$$

де $\vec{i}_\rho$ — орт радіального напрямку в циліндричній системи координат,

з параметрами: α=3, в рівнозначному діапазоні зміни значень функції по осі *x* та осі *y* від -2 до +2 з кроком 0,2.

Для обчислення знайденої аналітично дивергенції

$$div\,\overline{F} = 2\exp(-\rho/\alpha)^2 \times [1 - (\rho^2/\alpha^2)] \quad (4.12)$$

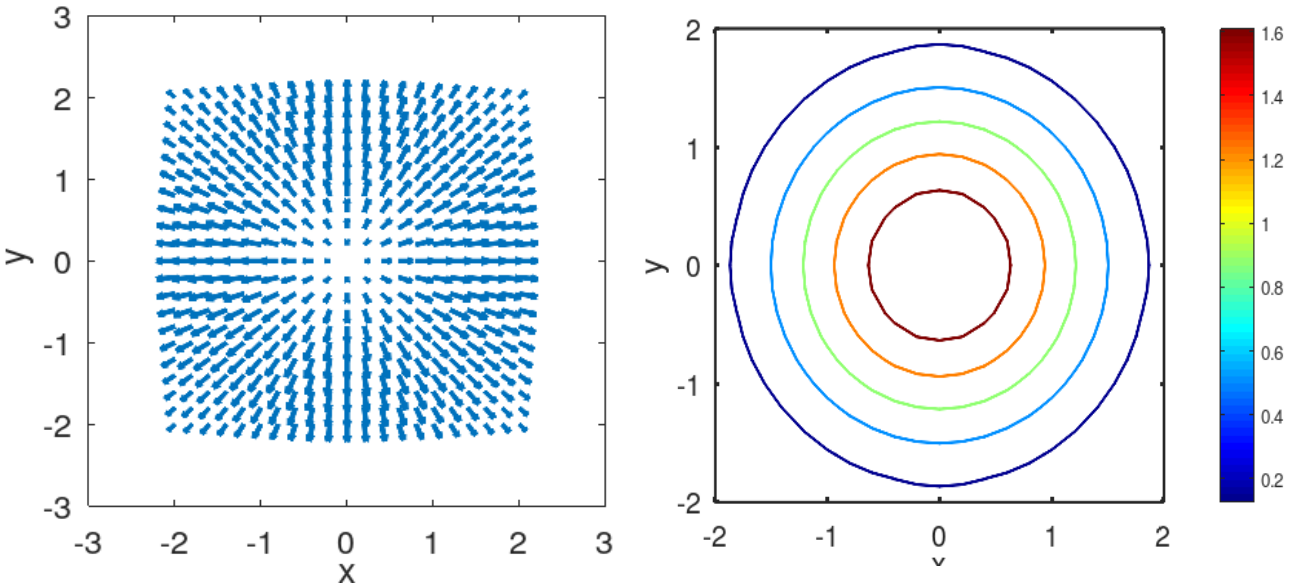
застосовано команду **divergence** у декартовій системі координат з урахуванням зв'язку між циліндричною та декартовою системами координат: $\rho=\sqrt{x^2+y^2}$.

Електричний струм характеризується вектором об'ємної густини струму $\vec{j}$ і силою струму I. Об'ємна густина електричного струму дорівнює заряду, що проходить в одиницю часу через одиничну поверхню перпендикулярно до ліній струму.

Відповідно до рівняння безперервності в диференціальній формі

$$div\,\vec{j}=-\frac{\partial\rho}{\partial t} \qquad (4.13)$$

витоки чи стоки векторного поля $\vec{j}$ — електричні заряди.



а *б*

Рисунок 4.7 — Векторне поле (*а*) та контури рівних значень дивергенції (*б*) із застосуванням команд **quiver та contour**

Лінії струму виходять із точок, з яких витікає додатній заряд, і закінчуються в точках, до яких він стікається: точки, де дивергенція

додатна, - *витоки*, точки з від'ємною дивергенцією — *стоки* поля.

Якщо припустити, що об'ємна густина ρ електричного заряду в об'ємі незмінна в часі, то похідна за часом дорівнюватиме нулю:

$$div\,\vec{j}=0\,, \qquad (4.14)$$

Поле, яке характеризується незмінними у часі **векторними чи скалярними величинами, називається постійним, чи стаціонарним.** Відповідно до (4.14) на постійному струмі векторне полі густини струму $\vec{j}$ немає витоків і стоків: силові лінії електричного поля замкнені.

Дивергенція електростатичного поля у середовищі з діелектричною проникністю ε, що має силову природу, визначає також положення джерел поля, які в цьому випадку називаються електричними зарядами, та дорівнює **об'ємній густині електричного заряду ρ** [16-18]:

$$div\,\vec{E}=\frac{\rho}{\varepsilon_o\,\varepsilon}\quad, \qquad (4.15)$$

де ε_0 =8,85*10^{-12} Ф/м — електрична стала.

Дивергенція вектора напруженості електричного поля пропорційна густині об'ємного заряду. Зазначене співвідношення називають теоремою Гаусса для вектора напруженості електричного поля в диференціальній формі.

Якщо **дивергенція поля дорівнює нулю**, то джерел та стоків у цього поля немає, або вони зрівноважені. Таке поле називають **соленоїдальним**.

Так, ***в будь-якій точці магнітного поля дивергенція вектора магнітного поля - магнітна індукція*** $\vec{B}$ (напруженість магнітного поля $\vec{H}$: $\vec{B}=\mu_0\mu\vec{H}$, де $\mu_0=4\pi\cdot10^{-7}$ Гн/м — магнітна стала, μ — магнітна проникність середовища) ***дорівнює нулю:***

$$div\,\vec{B}=0\,. \qquad (4.16)$$

Формула (4.16) відображає факт відсутності магнітних монополів (скалярних магнітних зарядів, подібних до електричних зарядів) і пов'язану з цим **соленоїдальність магнітного поля – силові лінії вектора магнітної індукції завжди замкнені. У такому полі немає ані джерел, ані стоків.**

Оскільки в природі немає магнітних зарядів, магнітне поле завжди є **вихровим**, і його силові лінії завжди замкнені. Силові лінії постійного магніту хоч і виходять із його полюсів (ніби мають джерела всередині), насправді замикаються всередині магніту. Тому, розрізав магніт надвоє, не вдасться отримати два окремі магнітні полюси.

Потенціальні (безвихрові або градієнтні) поля характеризують скалярними функціями: електричним потенціалом φ електричного поля та магнітним потенціалом $\varphi_м$ магнітного поля постійного струму.

Необхідною та достатньою умовою **потенційності векторного поля** є рівність у цій області нулю **ротора поля (вектора вихора.**

Векторний добуток, що означає соленоїдальність поля, визначається з урахуванням (4.6) та (4.1)

$$[\nabla\times\overline{F}]=\left(\left(\frac{\partial}{\partial x}\overline{i}+\frac{\partial}{\partial y}\overline{j}+\frac{\partial}{\partial z}\overline{k}\right)\cdot\left(P\overline{i}+Q\overline{j}+R\overline{k}\right)\right)=\begin{vmatrix}\overline{i} & \overline{j} & \overline{k}\\ \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z}\\ P & Q & R\end{vmatrix} \qquad (4.17)$$

має назву **ротор**, і визначається:

$$rot\ \overline{F}=[\nabla\times\overline{F}]=\begin{vmatrix}\overline{i} & \overline{j} & \overline{k}\\ \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z}\\ P & Q & R\end{vmatrix}=\overline{i}\begin{vmatrix}\frac{\partial}{\partial y} & \frac{\partial}{\partial z}\\ Q & R\end{vmatrix}-\overline{j}\begin{vmatrix}\frac{\partial}{\partial x} & \frac{\partial}{\partial z}\\ P & R\end{vmatrix}+\overline{k}\begin{vmatrix}\frac{\partial}{\partial x} & \frac{\partial}{\partial y}\\ P & Q\end{vmatrix}. \quad (4.18)$$

Ротор — вектор, координати якого визначаються визначником третього порядку, перший рядок — одиничні вектори (орти) координатних осей, другий — оператори частинного диференціювання в такому ж порядку, як і орти осей, третій — координати функції, яка визначає векторне поле $\overline{F}$.

З практичної точки зору **ротор векторного поля характеризує обертальну здатність поля в даній точці**: найбільша саме в площині, перпендикулярній ротору.

На рис. 4.9 представлено контури рівних значень ротора векторного поля $\overline{F}$, яке задано у циліндричній системі координат:

$$\vec{F}=\beta\times\rho\exp(-\rho/\alpha)^2\times\dot{i}_{\varphi}, \quad (4.19)$$

де $\dot{i}_{\varphi}$ — орт кутового (азимутального) напрямку в циліндричній системи координат,

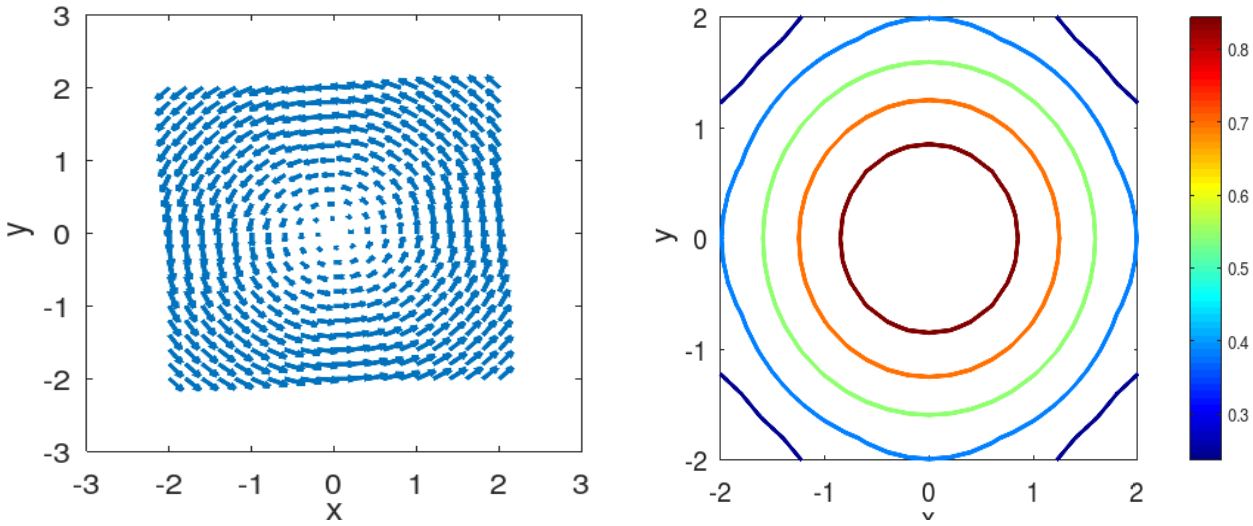
з параметрами: α=3, β=1, в рівнозначному діапазоні зміни значень функції по осі *x* та осі *y* від -2 до +2 з кроком 0,2.

Для обчислення аналітично знайденого ротора застосовано команду **curl (curl — позначення ротора у англомовній технічній літературі)** у декартовій системі координат з урахуванням зв'язку між циліндричною та декартовою системами координат: $\rho=\sqrt{x^2+y^2}$

$$rot\,\vec{F}=2\times\beta\exp(-\rho/\alpha)^2\times[1-(\rho^2/\alpha^2)]\dot{z}_0\ . \quad (4.20)$$

Порівняння аналітичних значень ротора з числовим показує високу точність збігу analitical_curl = 1.5908, numerical_curl=1.59083.

На рис. 4.10 показано контури рівних значень ротора векторного поля $\vec{F}$, яке задано на підставі (4.11) з параметрами: α=-0,5, в рівнозначному діапазоні зміни значень функції по осі x та осі y від -1 до +1 з кроком 0,1.



а *б*

Рисунок 4.9 — Векторне поле (*а*) та контури рівних значень ротора (*б*) із застосуванням команд **quiver та contour**

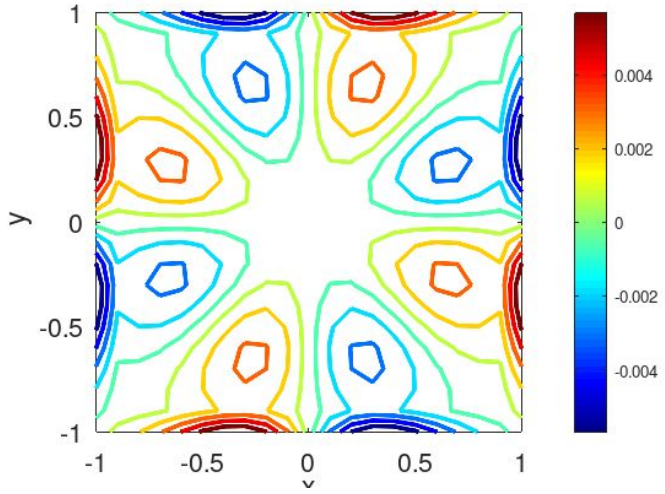


Рисунок 4.10 — Результати визначення значень ротора із застосуванням команди **curl** з представленням контурів рівних значень командою **contour**

4.4. Узагальнені властивості та характеристики електромагнітних полів

Відповідно до розглянутих характеристик поля можна виділити три **часткових види електромагнітного поля** [16—17]**.**

1. **Безвихрове і потенціальне електростатичне поле** характеризується **відсутністю руху заряду** $\frac{dV}{dt}\cdot q=0$, тобто утворено **нерухомими та незмінними у часі зарядами** $\frac{\partial \mathrm{q}}{\partial \mathrm{t}}=0$. Це означає, що **густина струму** $\vec{j}=0$, відповідно **магнітне поле відсутнє:** $\vec{B}=0$, $\vec{H}=0$.

Таке поле **визначається** наступними рівняннями Максвелла у диференційній формі:

$$rot\,\vec{E}=0\,, \tag{4.21}$$

$$div\,\vec{D}=\rho \tag{4.22}$$

,

з урахуванням співвідношень для потенціалів полів

$$\vec{E}=-grad\,\varphi \tag{4.23}$$

та рівняннями Пуассона — за наявності або Лапласа — за відсутності зарядів ρ у середовищі з діелектричною проникністю ε відповідно

$$\nabla^2=\frac{\rho}{\varepsilon_0\cdot\varepsilon}\,,\quad \nabla^2=0\,, \tag{4.24}$$

де $\nabla^2=\Delta=\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}+\frac{\partial^2}{\partial z^2}$ — скалярний оператор, оператор Лапласа у

декартовій системі координат.

Рівняння (4.21) означає: **електричне поле — не вихрове.** Для потенціального поля ротор в кожній точці — нульовий вектор. Лінії напруженості електричного поля мають початок і кінець (див. рис. 4.3). Електричне поле починається на позитивних і закінчується на негативних зарядах. **Електричне поле має стоки й витоки** (рівняння 4.22). **Градієнт потенціалу — векторна величина** (рівняння (4.23)), Знак «-» означає, що **вздовж силової лінії потенціал спадає.**

2. **Стаціонарне електричне поле постійного струму** створюється зовнішніми джерелами в провідних середовищах та супроводжується проходженням постійного струму. При цьому у провіднику не накопичується електричний заряд (ρ=0), а на поверхні густина заряду є постійною (ρ=const).

Система основних рівнянь **електричного поля постійного струму** за відсутності струму переносу записується у вигляді:

$$rot\,\vec{E}=0\,, \tag{4.25}$$

$$div\,\vec{j}=0 \tag{4.26}$$

$$\vec{j}=\gamma\vec{E}\,, \tag{4.27}$$

де γ — питома електрична провідність середовища (См/м).

Рівняння (4.25) характеризує це поле як потенціальне; (4.26) — 1-й закон Кірхгофа, що співвідноситься із законом для електричних кіл: сума вузлових струмів дорівнює нулю; рівняння (4.27) представляє закон Ома у диференціальній формі без наявності зовнішніх джерел.

3. **Магнітне поле постійного струму** утворюється в провіднику

та навколишньому просторі під час проходження постійного струму по провіднику. Це поле не змінюється в часі, тому немає взаємного впливу електричного і магнітного полів:

$$rot\,\vec{H}=\vec{\vec{j}} \qquad ,(4.24)$$

$$div\,\vec{B}=0 \qquad (4.25)$$

з матеріальним рівнянням щодо зв'язку між вектором магнітної індукції та напруженістю магнітного поля:

$$\vec{B}=\mu_0\mu\vec{H} \quad . \qquad (4.26)$$

Рівняння (4.24) — узагальнення закону повного струму та характеризує магнітне поле як **вихрове: будь-який електричний струм з густиною** $\vec{j}$ **створює вихрове магнітне поле.**

Рівняння (4.25) доводить про безперервність (замкненість) ліній вектора магнітної індукції.

Таким чином, **електростатичне поле — потенційне, магнітостатичне - соленоїдальне.** Довільне **електромагнітне поле** є сумою цих полів. І взагалі: будь-яке **векторне поле** виходить додаванням **потенційного** та **соленоїдального** полів.

Зв'язки між характеристиками **електромагнітного поля** для будь-якої окремої точки простору в будь-який окремо взятий момент часу встановлюються на підставі **рівнянь Максвелла в диференціальній формі для нерухомих середовищ.**

Перше рівняння Максвелла являє собою диференціальну форму запису закону повного струму, який складається зі струму провідності

$\vec{j}$ та струму зміщення $\vec{j}_{зм}=\frac{\partial \vec{D}}{\partial t}=\varepsilon_0 \varepsilon \frac{\partial \vec{E}}{\partial t}$: струм провідності і змінне у часі електричне поле збуджують у просторі вихрове магнітне поле

$$rot\,\vec{H}=\vec{\vec{j}}+\frac{\partial \vec{D}}{\partial t} \qquad ,(4.27)$$

де вектор електричної індукції $\vec{D}=\varepsilon_0 \times \varepsilon \times \vec{E}$. (4.28)

Струм провідності відповідає руху зарядів, струм зміщення може існувати у вакуумі, де цей струм рухом зарядів не супроводжується.

Друге рівняння Максвелла — закон електромагнітної індукції в диференціальній формі: будь-яка зміна у часі магнітного поля породжує вихрове електричне поле:

$$rot\,\vec{E}=-\frac{\partial \vec{B}}{\partial t} \qquad (4.29)$$

Електричне поле може збуджуватись не тільки електричними зарядами, а також і змінним магнітним полем.

Третє рівняння Максвелла є узагальненою теоремою Гаусса:

$$div\,\vec{D}=\rho\,. \qquad (4.30)$$

Четверте рівняння Максвелла базується на принципи безперервності магнітних силових ліній:

$$div\,\vec{B}=0 \qquad (4.31)$$

До рівнянь Максвелла додаються матеріальні рівняння (4.26) та (4.28) відповідно:

$$\vec{B}=\mu_0\mu\vec{H} \text{ та } \vec{D}=\varepsilon_0\times\varepsilon\times\vec{E}.$$

Система рівнянь (4.27) — (4.31) є повною для однозначного визначення електромагнітного поля у всіх точках простору тільки у випадку, якщо задані початкові значення векторів $\vec{E}$ і $\vec{H}$ (для часу t= 0) та визначені граничні умови для миттєвих значень векторів змінного електромагнітного поля.

1. На межі поділу двох діелектриків дотичні (тангенціальні) складові напруженостей електричного поля незмінні [16—18]:

$$E_{1\tau}=E_{2\tau}. \quad (4.32)$$

2. Нормальні складові вектора електричної індукції неперервні за відсутності вільних електричних зарядів з поверхневою густиною σ:

$$D_{1n}=D_{2n} \quad (4.33)$$

3. За умови наявності вільних електричних зарядів з поверхневою густиною σ на межі поділу діелектричних середовищ нормальна складова електричного поля має розрив, що дорівнює поверхневій густині заряду:

$$D_{1n}-D_{2n}=\sigma. \quad (4.34)$$

4. Нормальна складова вектора магнітної індукції на межі є безперервною:

$$B_{1n}-B_{2n}=0 \quad (4.35)$$

5. Дотичні складові вектора напруженості магнітного поля незмінні за умови відсутності поверхневої густини струму:

$$H_{1\tau}=H_{2\tau} \tag{4.36}$$

4.5. Практична реалізація візуалізації електромагнітного поля при протіканні змінного струму у проводі циліндричної форми

Для визначення розподілу густини струму за перерізом проводу, по якому тече відмінний від нуля повний змінний струм, застосуємо рівняння Максвелла для квазістаціонарного режиму електромагнітного поля. Умови застосування квазістаціонарного наближення для опису електродинамічних процесів у провідних системах включають нерівності, що забезпечують зазначені спрощення виду рівнянь Максвелла та накладають обмеження на частоту зміни поля та струмів, на розміри систем та електрофізичні показники речовини. Змінення за часом t електромагнітного поля характеризується його головними монохроматичними Фур'є-гармоніками

$$\vec{E}=E_0\exp(j\omega t) \quad , \quad \vec{H}=H_0\exp(j\omega t) \tag{4.37}$$

із коловою частотою $\omega=2\pi f$ та періодом T, що приблизно дорівнює характерному часу зміни поля. Поле змінюється повільно, тому електричну провідність γ, діелектричну ε та магнітну μ проникності можна вважати постійними і рівними їх статичним значенням. Середовище — ізотропне.

Важливо, квазістаціонарний режим охоплює електромагнітні поля, що порівняно повільно змінюються. Це дозволяє **знехтувати струмами зміщення:**

$$\vec{j}_{зм}=\frac{\partial \vec{D}}{\partial t}=\varepsilon_0 \varepsilon \frac{\partial \vec{E}}{\partial t}=0 \qquad (4.38)$$

Для металевих проводів нехтування струмами зміщення припустимо у широкому діапазоні частот. Саме це дозволяє визначати електричні параметри провідних систем: повітряних ліній електропередавання, силових кабелів, симетричних та коаксіальних кабелів зв'язку.

У **квазістаціонарному наближенні** система рівнянь Максвелла включає рівняння (4.29): $rot\,\vec{E}=-\frac{\partial \vec{B}}{\partial t}$ — **магнітне поле, що повільно змінюється, індукує появу у просторі вихрового електричного поля: за наявності змінного магнітного поля електричне поле втратило властивість потенціальності** (4.25).

У електростатичному наближенні напруженість електричного поля всередині проводів має дорівнювати нулю. Відмінна від нуля напруженості електричного поля $\vec{E}$ призвела б до виникненню струму. Протікання струму у проводі пов'язане з дисипацією (втратами) енергії і тому не може саме собою (без зовнішніх джерел енергії) підтримуватись у стаціонарному стані. Звідси випливає, що всі заряди у проводі повинні бути розподілені по його поверхні. Наявність зарядів в об'ємі проводу неодмінно призвела б до виникнення електричного поля в ньому. Розподіл зарядів по поверхні може бути здійснено таким чином, щоб створювані ними всередині проводу поля взаємно компенсувалися. **Завдання**

електростатики проводу зводиться до визначення електричного поля в порожнечі, поза проводу, та до визначення розподілу зарядів по поверхні проводу. Електростатичне поле — нормальне до поверхні проводу в кожній її точці. Оскільки $\vec{E} = -grad\,\varphi$, потенціал поля має бути постійним вздовж усієї поверхні проводу. Іншими словами, **поверхня однорідного проводу являє собою еквіпотенційну поверхню електростатичного поля.**

При обчисленні розподілу густини струму у перерізі прямолінійного проводу електричне поле паралельно його осі. Вектор магнітного поля $\vec{H}$ лежить у площині, перпендикулярній до осі. Розглянемо провід кругового перерізу. Через симетрію проводу на його поверхні напруженість електричного поля залишається постійною у часі. За такої граничної умови рівняння $\mathrm{div}\,\vec{E} = 0$, $rot\,\vec{E} = 0$ у просторі поза проводом мають рішення лише у випадку E=const у всьому просторі. За аналогічних причин і магнітне поле навколо проводу буде таким самим, яким воно було б навколо проводу при протіканні по ньому постійного струму, рівному даному миттєвому значенню змінного струму.

Всередині проводу електричне поле задовольняє рівнянню

$$\frac{1}{4\pi\varepsilon_0\mu_0\gamma}\Delta\vec{E} = \frac{\partial\vec{E}}{\partial t} \qquad .(4.39)$$

У циліндричній системі координат з віссю oz вздовж осі проводу електричне поле $\vec{E}$ має лише z-компоненту і залежить тільки від координати радіусу *r*.

З урахуванням (4.37) для періодичного електромагнітного поля, що змінюється з коловою частотою ω, для поздовжньої електричної складової диференціальне рівняння має вигляд:

$$\frac{1}{r}\frac{\partial}{\partial r}\cdot\left(r\frac{\partial E}{\partial r}\right)+k^2\vec{E}=0 \qquad (4.40)$$

де $k=\sqrt{j\omega\mu_0\mu\gamma}=\frac{\sqrt{2}}{\Delta}$ (4.41)

— коефіцієнт вихрових струмів, модуль якого має розмірність 1/м, Δ — глибина проникнення електромагнітної хвилі, м.

Рішення (4.40) залишається кінцевим при *r*=0 (на осі проводу) та записується у вигляді:

$$E=E_z=A\,J_o(kr)\exp(-j\omega t) \qquad , (4.42)$$

де А — постійна інтегрування, $J_o(kr)$ — функція Бесселя першого роду нульового порядку.

Густина струму провідності відповідно до закону Ома у диференційній формі (4.27) змінюється аналогічно, за (4.42).

З урахуванням (4.29) та (4.26) азимутальна магнітна складова електромагнітного поля визначається

$$H_\varphi=-\frac{1}{j\omega\mu_0}\frac{\partial E_z}{\partial r} \qquad (4.43)$$

Рішення для *Hφ* записується у вигляді:

$$H_\varphi=-jA\sqrt{\frac{4\pi\gamma j}{\omega}}J_1(kr)\exp(-j\omega t) \qquad ,(4.44)$$

де $J_1(kr)$ — функція Бесселя першого роду першого порядку.

Постійна інтегрування *A* визначається з врахуванням умови: на

поверхні проводу напруженість магнітного поля визначається на підставі закону Ампера:

$$H_{\varphi}=\frac{I}{2\pi r_a} \qquad (4.45)$$

де I — струм, що протікає по проводу, радіус котрого дорівнює r_a.

На рис. 4.11 — рис. 4.12 представлено лінії рівня та поверхні абсолютних значень густини струму провідності (рис.4.11, *а;* рис.4.12, *а),* лінії рівня та поверхні абсолютних значень напруженості магнітного поля (рис.4.11, *б*; рис.4.12, *б*), векторне магнітне поле (рис.4.11, *в;* рис.4.12, *в*) при протіканні змінного струму амплітуди 1А частоти 50 Гц та 150 Гц (рис.4.11, *б*) у мідному проводі перерізом 1200 мм^2 відповідно.

На рис. 4.13 для мідного проводу перерізом 240 мм^2 показано розподіл густини струму (рис. 4.13, *а*) та напруженості магнітного поля (рис. 4.13, *б*) при протіканні струму амплітуди 1А частоти 50 Гц.

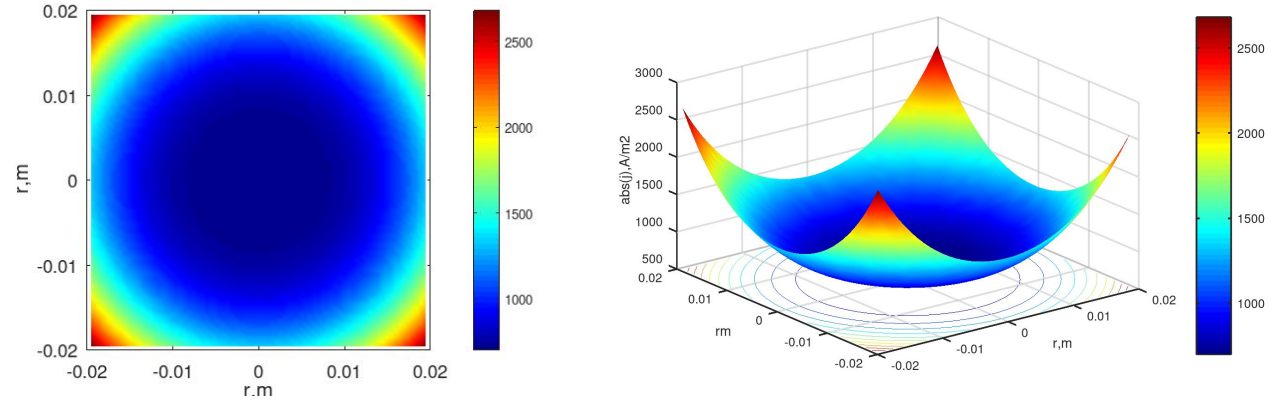
При протіканні в однорідному проводі постійного струму провідності густина струму однакова в різних точках перерізу проводу.

При протіканні змінного струму I густина струму виявляється не однаковою за розподілом по перерізу: найбільша - на поверхні та найменша - на осі провідника.

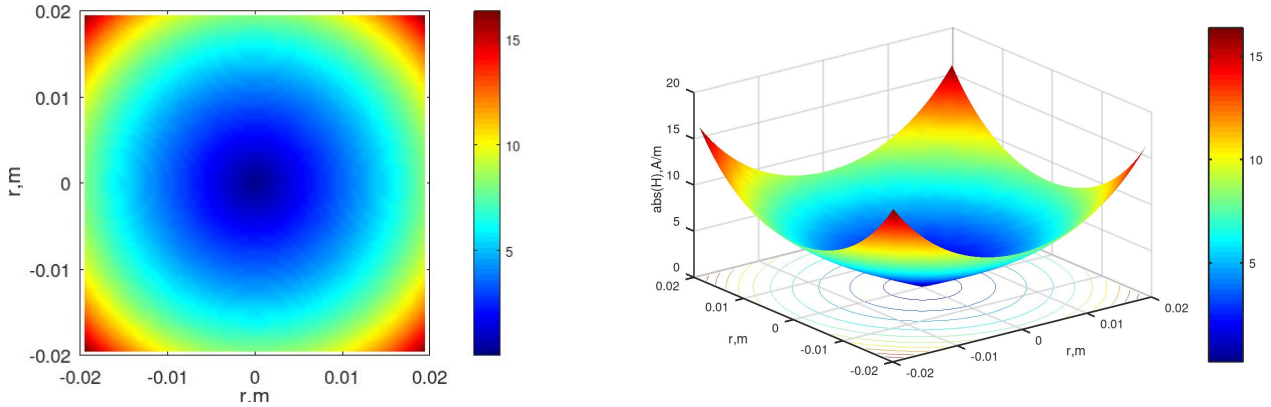
Змінний струм створює всередині провідника **вихрове магнітне поле** (рис. 4.11, *в* — рис.4.13, *в)*, силові лінії якого лежать у площині, перпендикулярній до осі провідника.

Припустимо, що струм у проводі посилюється. Тоді магнітне поле $\vec{H}$, що зростає, призводить до появи **вихрового електричного поля** $\vec{E}$, яке у поверхні провідника спрямовано однаково зі струмом I, а на

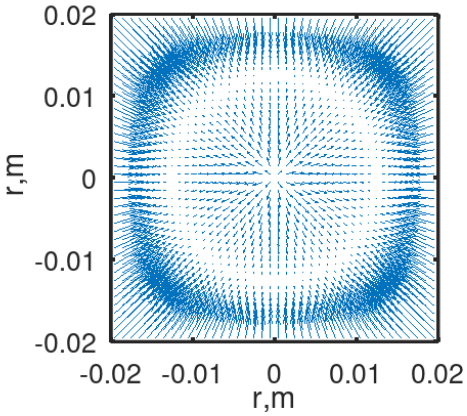
осі проводу — протилежно до напрямку струму. Це поле буде посилювати струм на поверхні і послаблювати його на осі.



а

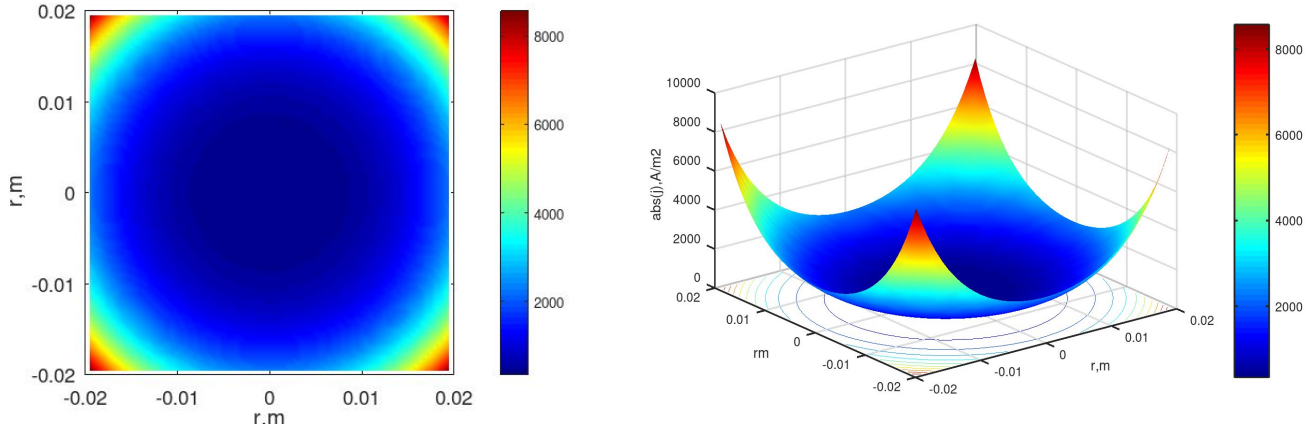


б

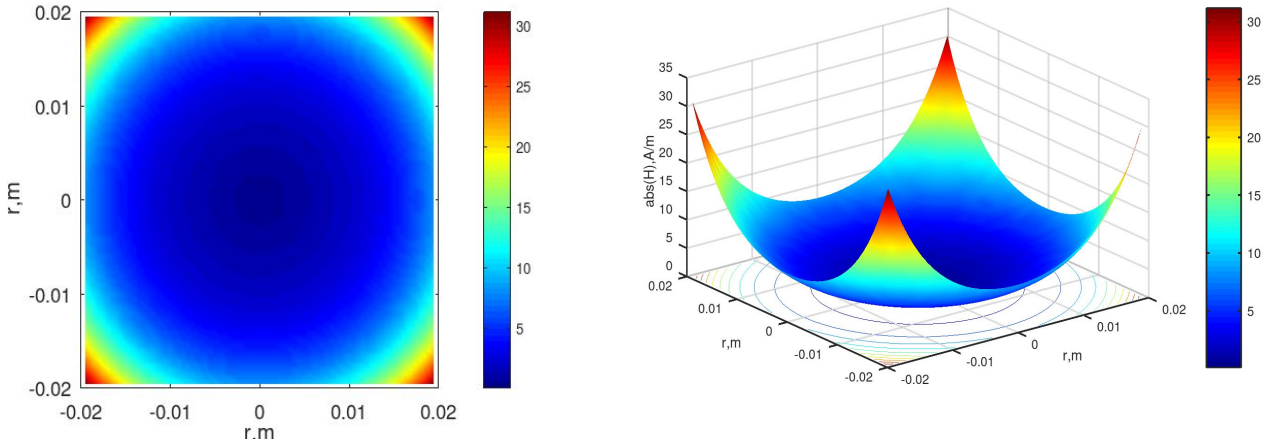


в

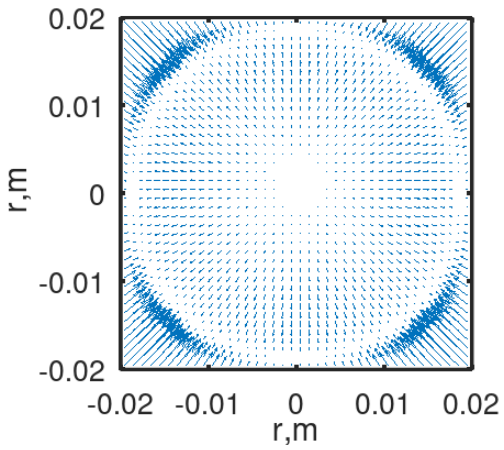
Рисунок 4.11 — Розподіл густини струму (*а*) та напруженості магнітного поля (*б*) при протіканні струму амплітуди 1А частоти 50 Гц у мідному проводі перерізом 1200 мм^2



а

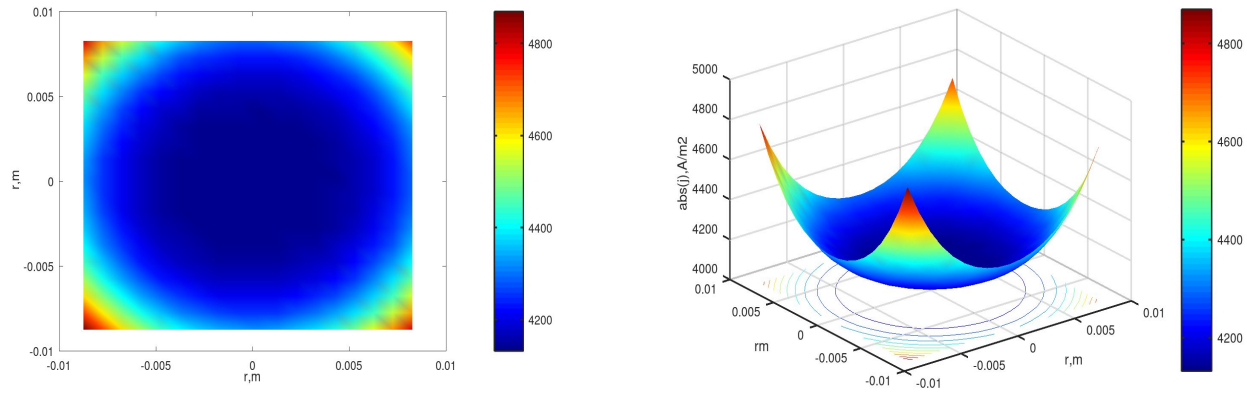


б

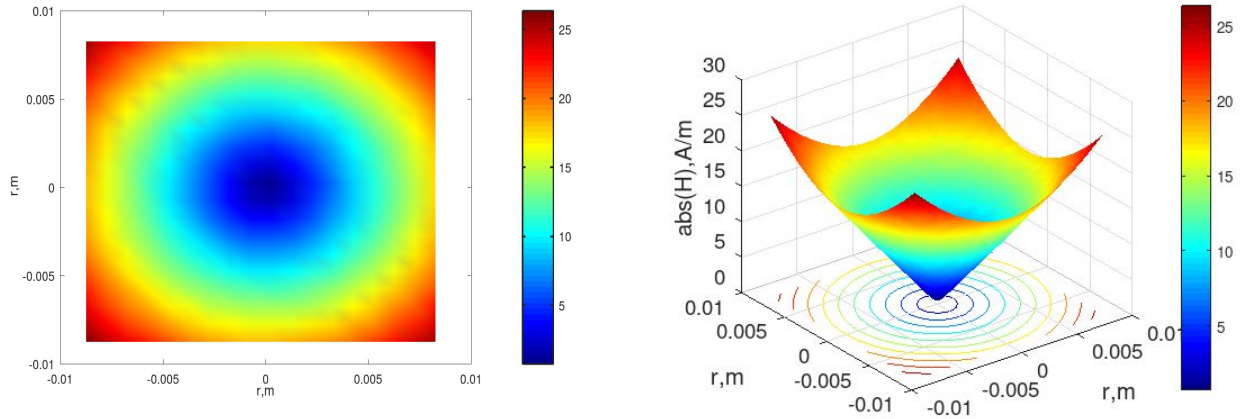


в

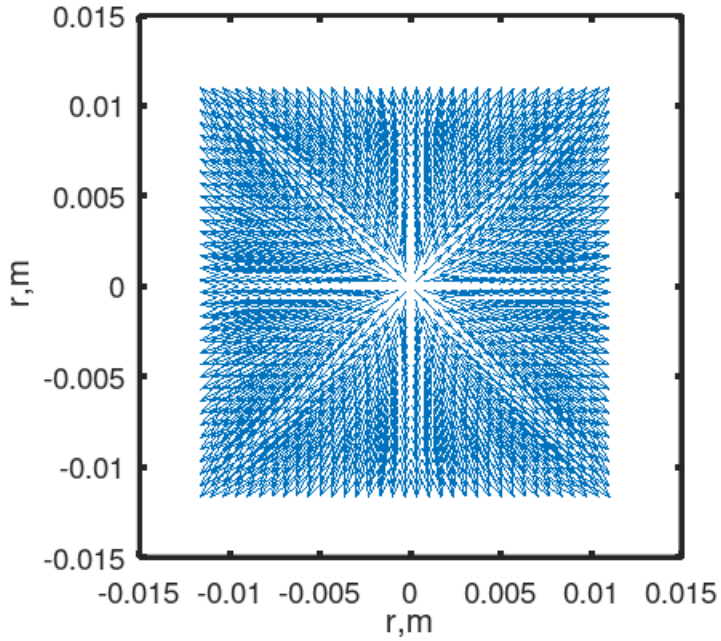
Рисунок 4.12 — Розподіл густини струму (*а*) та напруженості магнітного поля (*б*) при протіканні струму амплітуди 1А частоти 150 Гц у мідному проводі перерізом 1200 мм^2



а



б



в

Рисунок 4.13 —Протікання змінного струму амплітуди 1 А частоти 50 Гц у мідному проводі перерізом 240 мм^2

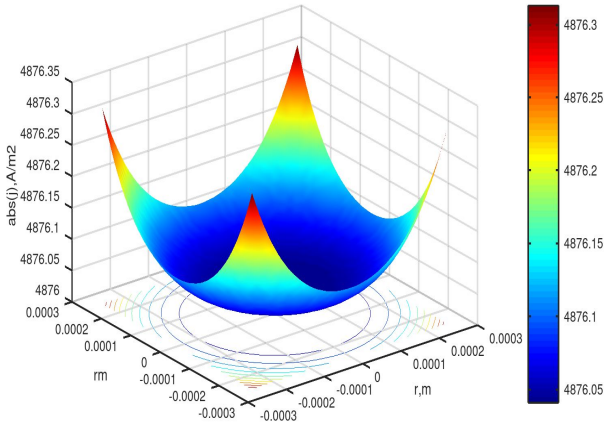
Уявимо тепер, що струм I зменшується. У цьому випадку магнітне поле $\vec{H}$ зменшується та обумовлює появу електричного поля $\vec{E}$, яке буде спрямоване протилежно в порівнянні з розглянутим випадком: у поверхні — протилежно струму, на осі — збігатися з напрямком струмом. В обох випадках, і при посиленні, і при ослабленні струму, **вихрове електричне поле** на осі провідника перешкоджатиме, а на поверхні сприятиме змінам струму, а отже, на осі провідника змінний струм буде менше, ніж на поверхні. Ця нерівномірність тим більше, чим більший діаметр проводу (порівняйте рис. 4.11 та рис. 4.13) і чим більша частота змінного струму (порівняйте рис. 4.11 та рис. 4.12).

Цей ефект стає особливо помітним на високій частоті (рис. 4.14). Зазначене явище пояснюється виникненням **вихрового магнітного поля електромагнітної індукції**. При збільшенні частоти струм практично існує тільки в тонкому поверхневому шарі (рис. 4.14, *г*). **Це явище отримало назву скін-ефекту** (від англійської — skin — шкіра, поверхневий шар).

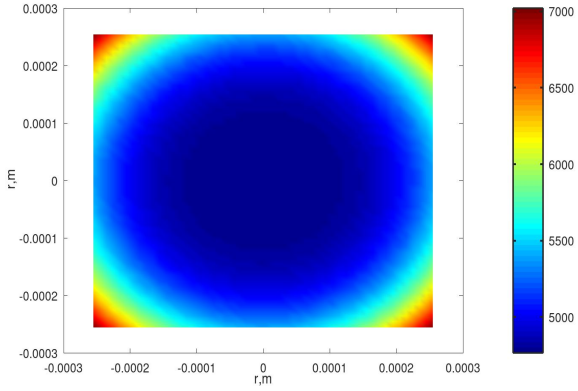
Внаслідок скін-ефекту електричний струм на високій частоті тече переважно крізь поверхневий шар проводу (рис. 4.14). Це призводить до зменшення ефективного перерізу проводу: порівняйте, наприклад, розподіл густини струму для частоти 1000 Гц та частоти 1 МГц. Товщина, на якій густина струму зменшується приблизно на 36,9% від значення струму на поверхні проводу, і є товщина скін-шару $\Delta = \frac{\sqrt{2}}{\sqrt{\omega \mu_0 \mu \gamma}}$.

Як наслідок, електричний опір проводу збільшується. За наявності скін- ефект струм існує практично тільки в поверхневому шарі, і магнітного поля усередині провідника немає. Зменшується

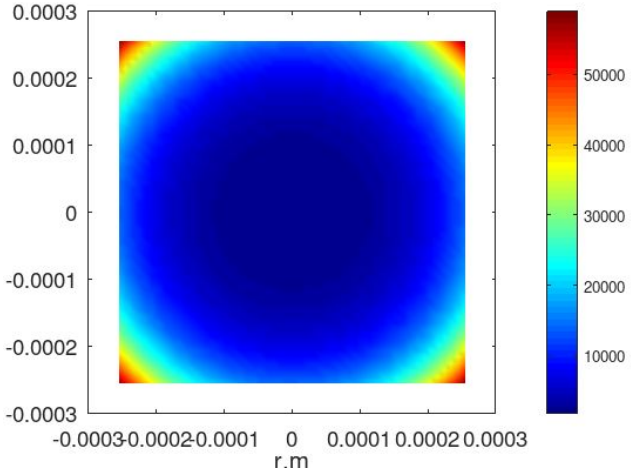
об'єм, котрий займає магнітне поле. Відповідно, індуктивність провідника із зростанням частоти зменшується.



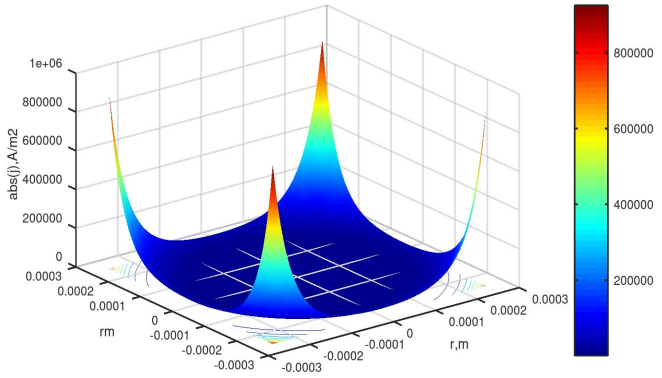
f=1000 Гц



f=100 кГц



f=1 МГц



f=5 МГц

Рисунок 4.14 — Прояв скін-ефекту при протіканні змінного струму амплітуди 1 мА у мідному проводі діаметром 0,511 мм

Явище поверхневого ефекту використовують у спеціальних провідниках — ліцендратах, які складаються з багатьох тонких провідників, ізольованих між собою. При протіканні змінного струму високої частоти спостерігається суттєвий перерозподіл густини струму з витисненням на поверхню. Це дозволяє виготовляти ліцендрат у вигляді трубок з тонкими стінками з нанесенням на провідник з довільного матеріалу тонкого шару металу з високою

провідністю (золото, срібло). Внутрішні мідні провідники спеціальних коаксіальних кабелів, зокрема, для стільникового зв'язку, виготовляють також у вигляді трубки.

Контрольні запитання до розділу 4

1. Яке поле називається скалярним? Наведіть приклади.

2. У чому проявляється особливість векторного поля? Наведіть приклади.

3. Як математично визначається рівняння поверхні рівня скалярного поля $u(P)$?

4. Обґрунтуйте причини назви рівняння поверхні еквіпотенціальною?

5. Що являють собою еквіпотенціальні лінії?

6. Чим обумовлено замкненість еквіпотенціальних ліній стаціонарного електричного поля ?

7. Дайте визначення векторної (силової) лінії поля.

8.Який принцип покладено у побудову силових ліній електричного поля диполя?

9. Якою величиною є градієнт та що визначає?

10. Як математично позначається перехід від скалярного поля до його градієнта та операцію переходу від векторного поля до його дивергенції?

11. Що означає інваріантність градієнта скалярного поля ?

12. Яка функція (команда) застосовується для побудови полів градієнта?

13. Якими величинами характеризуються зміни векторного поля?

14. Назвіть основні диференціальні характеристики у теорії поля.

15. Якою величиною є дивергенція та що характеризує?

16. Яке поле називається постійним, стаціонарним

17. Чому дорівнює дивергенція електростатичного поля у середовищі з діелектричною проникністю ε?

18. Яке поле називається соленоїдальним?

19. Чому магнітне поле завжди є вихровим?

20. Як називається векторний добуток, що визначає соленоїдальність поля?

21. Перелічте основні особливості електростатичного поля.

22. У чому полягає особливість стаціонарного електричного поля постійного струму?

23. Назвіть умову утворення магнітного поля постійного струму.

24. Що являє собою електромагнітне поле?

25. Перелічте граничні умови для миттєвих значень векторів змінного електромагнітного поля?

26. За яких умов можливо знехтувати струмами зміщення?

27. Чому дорівнює напруженість електростатичного поля всередині провідних середовищ?

28. Що являє собою поверхня провідного середовища в електростатичному полі?

29. Які причини обумовлюють перерозподіл густини електричного струму по перерізу проводу при протиканні змінного струму?

Розділ 5. Обробка експериментальних даних

Обробка даних експерименту, у тому числі і при проведенні лабораторних занять, одне з тих завдань, з якими зустрічаються абсолютно всі студенти незалежно від їх спеціалізації [20—21]. Проте саме поняття «обробка експериментальних даних» настільки широко, що у кожному конкретному випадку потрібно пояснення, що саме під ним мається на увазі.

Наприклад, це може бути встановлення закономірності, котрій підпорядковуються дані, підбір формули, здатної їх описати; пролонгація отриманих значень в область, недоступну для експериментальних досліджень; отримання даних про значення досліджуваних величин у проміжних точках. Залежно від цього різняться математичні постановки задачі обробки експериментальних даних, а також методи та засоби її вирішення [20—21].

5.1. Інтерполяція та екстраполяція

Інтерполяція (від лат. interpolatio — підновлення, зміна) в математиці, метод відновлення (звичайно, наближеного) функції за значеннями самої функції і, можливо, деяких її похідних на кінцевій множині точок.

До чисельного інтерполювання доводиться звертатися, коли обчислюють значення функцій, заданих у вигляді таблиці. При складному аналітичному вигляді функції або отриманої експериментально на підставі інтерполяції значення набувають значень функції, яка інтерполює.

Потрібно розуміти, що через фіксований набір точок експериментальних даних, можливо провести нескінченно велику кількість кривих, тобто завдання інтерполяції немає однозначного

рішення. Для того щоб виділити якесь рішення, необхідно вказати метод, за допомогою котрого проводитиметься інтерполяція.

Основні застосовані методи наведено у табл. 5.1.

— **Інтерполяція методом найближчого сусіда**: значення величини, яка досліджується, в проміжній точці приймається рівним значенню величини в найближчій відомій точці. Різновидом методу також є методи «сусіда праворуч» та «сусіда зліва». Цей метод є найбільш грубим, застосовувати його має сенс тільки при наявності великого обсягу даних з малим кроком для знаходження проміжних значень безпосередньо поблизу експериментальних точок.

— **Лінійна інтерполяція**: передбачається лінійна залежність значення вимірюваної величини на відрізках між експериментальними точками. Сфера застосування цього методу така ж, як і методу найближчого сусіда.

— **Інтерполяція сплайнами.** Сплайном називається функція, безперервна разом зі своїми похідними у всій області визначення даних і задається на відрізку між двома алгебраїчними точками багаточленом. Тобто сплайн являє набір багаточленів, різних для кожної ділянки, що інтерполюється, але підбираються таким чином, що для кожної експериментальної точки збігаються значення самої величини та її похідної, що обчислюється ліворуч і праворуч. Вимога безперервності похідної забезпечує плавний перехід один в одне сплайнів для різних ділянок. Найбільш наочне уявлення сплайна – гнучка пружна лінійка, закріплена в експериментальних точках. На практиці найчастіше застосовують кубічний сплайн через однозначності його побудови.

— **Інтерполяція поліномами.** Для побудови кривої, що проходить через всі експериментальні точки, підбирається поліном відповідно- го ступеня. На відміну від інтерполяції сплайнами, де

поліном для кожної ділянки підбирається окремо, у разі поліноміальної інтерполяції поліном єдиний всім ділянок. Підібрати такий поліном можливо завжди — кількість коефіцієнтів у поліномі має відповідати числу експериментальних точок. Однак навіть у цьому випадку завдання має нескінченну кількість рішень, оскільки сам поліном можна вибирати у різний спосіб. Найчастіше на практиці застосовуються поліноми у формі Лагранжу, Ньютона, Ерміту. Застосування поліноміальної інтерполяції потребує особливої обережності: треба розуміти, що високий степінь полінома, швидше за все, не має нічого спільного з реальною залежністю, яка описує експериментальні дані. Крім того, використовування поліномів може призводити до сильних осциляцій функцій у проміжних точках.

Методи інтерполяції використовуються для наближеного інтегрування в машинній графіці, при чисельному розв'язанні диференціальних рівнянь [22—24].

В GNU Octave одновимірна інтерполяція реалізована за допомогою однойменної функції **interp1**. У найпростішому випадку виклик функції має вигляд:

Y1 = interp1 (X, Y, X1).

X і **Y** — вектор-рядки або вектор-стовпці експериментальних даних (**X** містить значення аргументу, **Y** - значення функції); **X1** — вектор, що містить значення аргументів у точках, для яких необхідно визначити значення функції. Функція повертає вектор **Y1**, що містить обчислені значення функції. **Y** може бути масивом, у цьому випадку інтерполяція буде проведена для кожного стовпця окремо, **Y1** у такому разі також буде масивом. У загальному випадку функція **interp1** викликається з розширеним набором аргументів, що дозволяє уточнити параметри процесу інтерполяції:

YI = interp1 (X, Y, XI, METHOD).

Таблиця 5.1 — Параметри METHOD та особливості застосування

Значення параметру METHOD	Пояснення	Зауваження
linear'	Лінійна інтерполяція. Значення в проміжній точці визначаються на основі лінійної інтерполяції двох сусідніх точок. Цей метод застосовується по визначенню	Необхідно як мінімум 2 точки. Потребує більше пам'яті та часу обчислювання, ніж метод найближчого сусіда
'nearest'	Метод найближчого сусіда. Значення в обчислюваній точці приймаються рівними значенню у найближчій експериментальній точці	Потрібно як мінімум 2 точки. Найбільш швидкий метод
'next'	Метод сусіда праворуч. Значення у обчислюваній точці приймаються рівними значенню в найближчій експериментальній точці праворуч	Потрібно як мінімум 2 точки. Швидкість така сама, як у методі 'nearest'
'previous'	Метод сусіда ліворуч. Значення у обчислюваній точці приймаються рівними значенню в найближчої експериментальній точці ліворуч	Потрібно як мінімум 2 точки. Швидкість така сама, як у методі 'nearest'
'pchip'	Кубічна інтерполяція	Необхідно як мінімум 4 точки. Потребує більше пам'яті та часу порівняно з лінійною інтерполяцією
'cubic'	Те саме, що і 'pchip'	Експериментальні точки повинні бути рівномірно розподілені
'makima'	Інтерполяція на основі модифікованої формули Акіма для кубічної інтерполяції поліномами Ерміта. Інтерполяція заснована на застосуванні шматкової поліноміальної функції ступеня не вище 3	Вимагає мінімум 2 точки. Інтерполяція менш гладка, ніж у методі 'spline', але більше гладка порівнянно із застосуванням в методі pchip. Швидкість обчислення вище, ніж у методі 'spline', але нижче, ніж у 'pchip'. Вимоги до пам'яті такі ж, як у 'spline'
'spline'	Кубічна інтерполяція сплайнами	Потрібні щонайменше 4 точки. Потрібно більше пам'яті і часу, ніж 'pchip'

Для ілюстрації розглянемо інтерполяцію сьома точками синусоїди різними способами. Нижче наводиться текст програми інтерполяції синусоїди різними методами (рис. 5.1).

```
%Програма з реалізацією інтерполяції різними методами
%тригонометричної функції
% очищення всіх параметрів
clear all
% Сім точок синусоїди
x=[0,45,90,135,180,225,270];
y=sind(x);
% Аргумент для побудови
x1=0:0.1:270;
%Обчислення різними методами інтерполяції
yNear= interp1(x,y,x1,'nearest');
yLinear=interp1(x,y,x1,'linear');
ySpline=interp1(x,y,x1,'spline');
yCubic=interp1(x,y,x1,'pchip');
% побудова графіків
figure(1)
xlabel('x');set(gca,'FontSize',18);
ylabel('sin(x)');set(gca,'FontSize',18);
set(gca,'FontSize',16,'fontname','Times New Roman','LineWidth',3);
grid on;
hold on;
plot(x,y,'Marker','s','LineWidth',3,'LineStyle','none');
plot(x1,yNear,'LineWidth',2,'LineStyle','-');
plot(x1,yLinear,'LineWidth',2,'LineStyle','-.');
plot(x1,ySpline,'LineWidth',2,'LineStyle',':');
plot(x1,yCubic,'LineWidth',2,'LineStyle','--');
legend('sin(x)','Інтерполяція найближчого сусіда','Лінійна',
%Сплайнами','Кубічна');
```

Дуже часто необхідно обчислити значення величини в області, для якої відсутні експериментальні дані. Для цього необхідно виконати **екстраполяцію функції**. Процедура екстраполяції, як і інтерполяції не має єдиного рішення. Метод екстраполяції потрібно вибирати із фізичних міркувань. В іншому випадку одержувані значення можуть бути дуже далекі від реальних. Крім того, не рекомендується проводити екстраполяції у віддалені області.

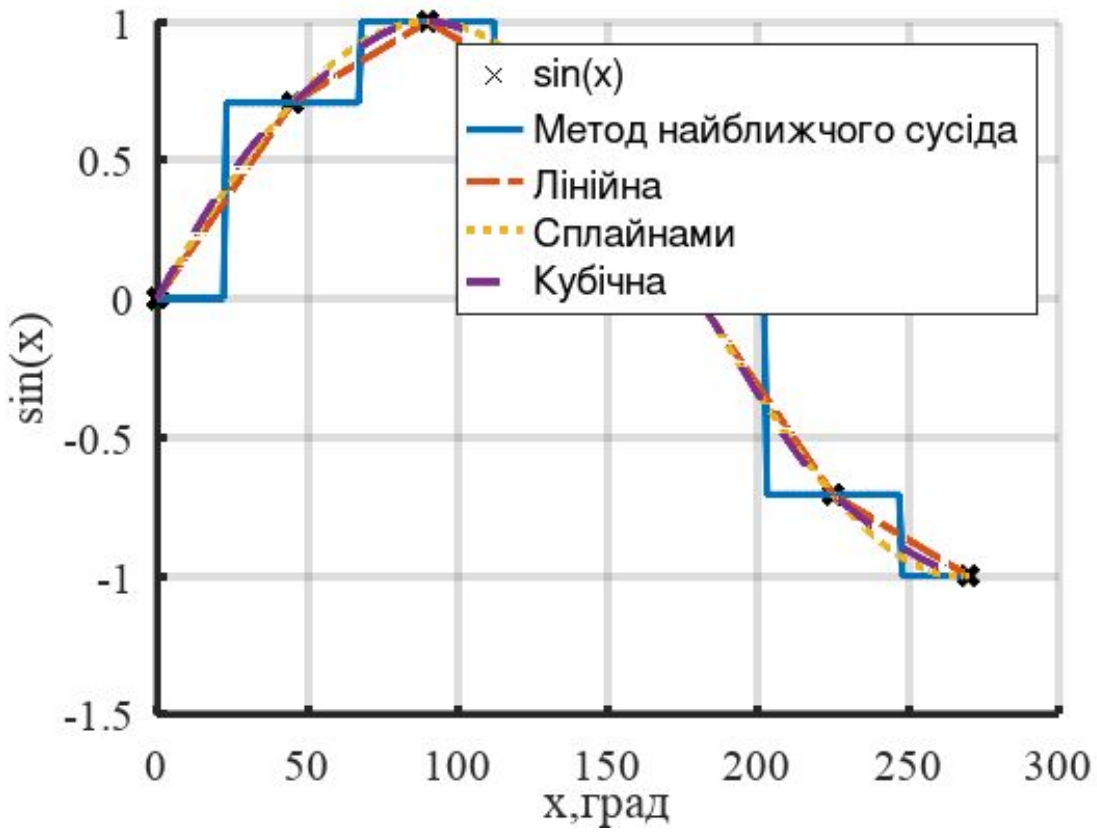


Рисунок 5.1 — Інтерполяція синусоїди різними методами

У математиці екстраполяція - це різновид апроксимації, при якій оцінювання значення змінної проводиться не всередині інтервалу її зміни (інтерполяція), а поза ним. При цьому екстраполяція більшою мірою, ніж інтерполяція, піддається впливу невизначеності та ризику отримати некоректні результати [22-24].

Існує кілька методів екстраполяції, вибір найбільш відповідного з яких залежить від апріорних відомостей про процес, який сформував точки даних, що спостерігаються — чи є функція гладкою, безперервною або періодичною. Методи екстраполяції, в основному, ті самі, що й інтерполяції.

Процедура екстраполяції реалізується за допомогою функції **interp1**, якою додатково вказується метод екстраполяції.

YI = interp1 (X, Y, XI, METHOD, EXTRAPOLATION).

У параметра EXTRAPOLATION є лише два можливі значення:

— **'extrap'** — у цьому випадку екстраполяція буде проведена тим самим методом, що і інтерполяція;

— «**число**» (вказується без лапок) — у цьому випадку за межами відрізка з відомими данними функції буде присвоєно постійне значення, рівне зазначеному аргументу.

Розглянемо екстраполяцію функції синус переліченими вище методами (рис. 5.2).

```
% Очищення всіх параметрів
clear all
% Сім точок синусоїди
x=[0,45,90,135,180,225,270];%Значення аргументу у градусах
y=sind(x);
%точки синусоїди для порівняння
xSin=[0,45,90,135,180,200,225,270,315,360];
ySin=sind(xSin);
% Аргумент для побудови
x1=0:0.1:360
% Екстраполяція різними методами
yNear= interp1(x,y,x1,'nearest','extrap');
yLinear=interp1(x,y,x1,'linear','extrap');
ySpline=interp1(x,y,x1,'spline','extrap');
yCubic=interp1(x,y,x1,'pchip','extrap');
% Побудова графіка
figure(1)
xlabel('x,град');set(gca,'FontSize',18);
ylabel('sin(x');set(gca,'FontSize',18);
set(gca,'FontSize',16,'fontname','Times New Roman','LineWidth',3);
grid on;
hold on;
plot(xSin,ySin,'kx-','LineWidth',3,'LineStyle','none');
plot(x1,yNear,'LineWidth',2,'LineStyle','-');
plot(x1,yLinear,'LineWidth',2,'LineStyle','-.');
plot(x1,ySpline,'LineWidth',2,'LineStyle',':');
plot(x1,yCubic,'LineWidth',2,'LineStyle','--');
legend('sin(x)','Метод найближчого сусіда','Лінійна' ,'Сплайнами','Кубічна ');
```

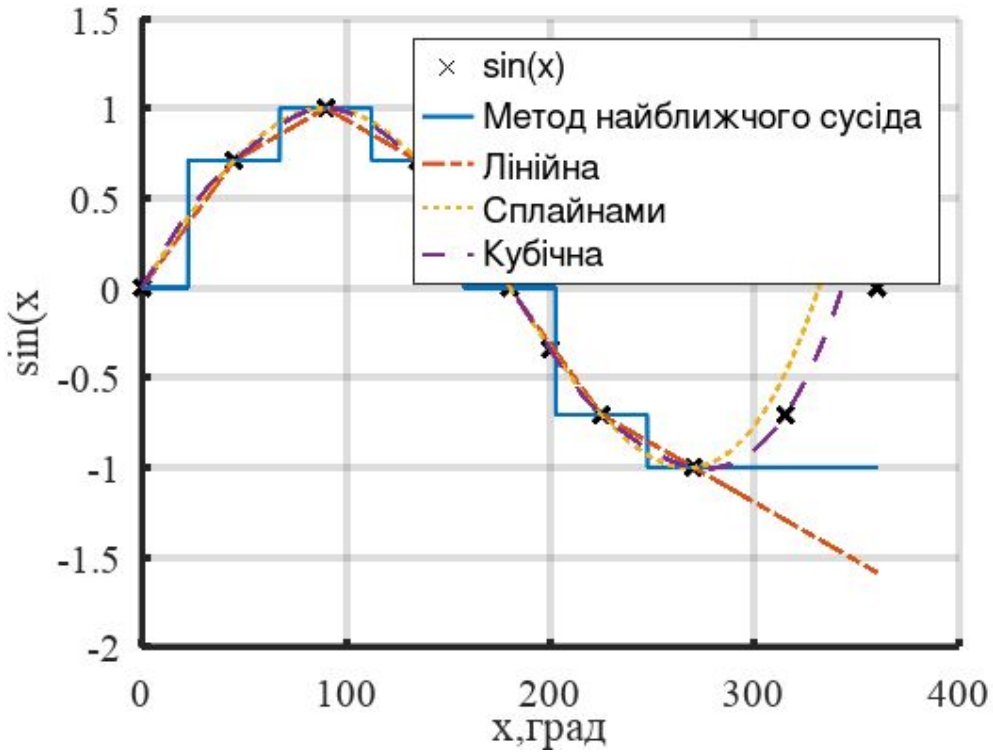


Рисунок 5.2 — Екстраполяція синусоїди різними методами

Видно, що поблизу області визначення екстраполяція дає значення, близькі до точних, але з видаленням точність прогнозу зменшується.

5.2. Лінійна та нелінійна апроксимація

Одним з типових завдань обробки даних є їх апроксимація. У цьому випадку загальний вигляд аналітичної залежності, якої описуються дані, відомі, але параметри цього розподілу не визначено. Завдання апроксимації полягає у визначені параметрів так, щоб загальна аналітична залежність найкращим чином відповідала даним експерименту. Найкращим варіантом, напевно, буде проходження аналітичної кривої точно через усі експериментальні точки. Але через похибки при проведенні експерименту може виявитися, що не всі експериментальні точки виявляться на аналітичній залежності

(більше того, може виявитися, що жодна з експериментальних точок не потрапляє на аналітичну залежність). Можна вимагати проходження аналітичної кривої через максимально можливу кількість експериментальних точок, але у такого підходу свій недолік - незрозуміло, як вибирати точки, в яких потрібен збіг, а також відхилення в точках, що залишилися, можуть бути настільки значні, що проведення апроксимації буде позбавлено сенсу.

В даний час прийнято, що найкращою функцією, що апроксимує, буде та, для якої сума квадратів відхилень в експериментальних точках буде мінімальною. Припустимо, що експериментально виміряні значення функції дорівнюють $\mathbf{y_1}$, $\mathbf{y_2}$, ... $\mathbf{y_n}$. Теоретичний розрахунок дає для цих точок значення $\mathbf{Y_1}$, $\mathbf{Y_2}$,...$\mathbf{Y_n}$. Найкращою функцією, що апроксимує, буде та, для якої сума

$$S=\sum_{i=1}^{n}(y_i-Y_i)\rightarrow \min \qquad (5.1)$$

даватиме найменше можливе значення. Метод, заснований на мінімізації цієї суми, отримав назву **методу найменших квадратів** (МНК). Комп'ютерна реалізація методу найменших квадратів сильно залежить від того, в якому вигляді обрана функція, що апроксимує.

Завдання складається з двох етапів:

1. За результатами експерименту визначити зовнішній вигляд залежності, що підбирається.
2. Підібрати коефіцієнти залежності $Y=f(x,a_1,a_2,\ldots,a_n)$.

Достатньою умовою мінімуму функції $S(a_1,a_2,\ldots,a_n)$ є рівність нулю всіх її похідних. Тому завдання пошуку мінімуму функції S еквівалентне вирішенню системи алгебраїчних рівнянь

$$\frac{\partial S}{\partial a_1}=0,\frac{\partial S}{\partial a_2}=0,\frac{\partial S}{\partial a_3}=0,....,\frac{\partial S}{\partial a_n}\ . \qquad (5.2)$$

Якщо параметри a_i входять у залежність $Y=f(x,a_1,a_2,...,a_n)$ лінійно, отримаємо систему (5.3) з n лінійних рівнянь з n невідомими

$$\begin{aligned}
&\sum_{i=1}^{n}2(y_i-f(x,a_1,a_2,...,a_n))\frac{\partial f}{\partial a_1}=0\\
&\sum_{i=1}^{n}2(y_i-f(x,a_1,a_2,...,a_n))\frac{\partial f}{\partial a_2}=0 \qquad (5.3)\\
&.....\\
&\sum_{i=1}^{n}2(y_i-f(x,a_1,a_2,...,a_n))\frac{\partial f}{\partial a_n}=0
\end{aligned}$$

Так, наприклад, параметри лінійної функції $Y=a_1+a_2x$, що апроксимує, визначаються на підставі

$$a_1=\frac{1}{n}(\sum_{i=1}^{n}y_i-a_2\sum_{i=1}^{n}x_i)$$

. (5.4)

$$a_2=n\sum_{i=1}^{n}y_i*x_i-\sum_{i=1}^{n}y_i\frac{\sum_{i=1}^{n}x_i}{n\sum_{i=1}^{n}x_i^2-(\sum_{i=1}^{n}x_i)^2}$$

Параметри апроксимації експериментальної залежності поліномом k -го степеню $Y=\sum_{i=1}^{k+1}(a_ix^{(i-1)})$ визначаються системою у матричному вигляді:

$$Ca=q, \qquad (5.5)$$

де C — елементи матриці:

$$C_i, j = \sum_{k=1}^{n} \left(x_k^{(i+j-2)} \right), i = 1, \ldots, k+1; \; j = 1, \ldots, k+1 \;\; , \tag{5.6}$$

q — вектор:

$$q_i = \sum_{k=1}^{n} \left(y_k x_k^{(i-1)} \right), i = 1, \ldots, k+1 \;\; . \tag{5.7}$$

Розв'язок системи (5.5) дозволяє визначити параметри залежності, що апроксимує:

$$Y = \sum_{i=1}^{k+1} \left(a_i x^{(i-1)} \right) \;\; . \tag{5.8}$$

Математично завдання підбору коефіцієнтів залежності зводиться до визначення коефіцієнтів a_i з умови (5.1).

В Octave її можна вирішувати кількома способами:

1. використовувати функцію **polyfit**,
2. розв'язати задачу оптимізації за допомогою функції **sqp**,
3. сформувати та вирішити систему лінійних алгебраїчних рівнянь для визначення коефіцієнтів a_i.

Поліноміальна апроксимація реалізована в GNU Octave за допомогою функції **polyfit**. Ця функція має три варіанти виклику:

p = polyfit(x,y,n);

[p,S] = polyfit(x,y,n);

[p,S,mu] = polyfit(x,y,n).

Список вхідних аргументів у всіх випадках однаковий – запитуються значення аргументу, значення функції та степінь

багаточлена для апроксимації. Степінь багаточлена рекомендується брати не вище ніж наявне число експериментальних точок. Список значень, що повертається, може змінюватись:

— у найпростішому випадку повертається вектор ***p***, що містить n+1 елемент. Компонентами вектора є коефіцієнти апроксимувального полінома, розташовані в порядку від найбільшого ступеня до найменшого, тобто поліном матиме вигляд

$$p(x)=p_1\cdot x^n+p_2\cdot x^{(n-1)}+...+p_x x+p_{(n+1)};, \qquad (5.9)$$

— у розширеному варіанті буде повернуто структуру S, котра застосовується для оцінки помилок;

— у загальному випадку буде повернено двохелементний вектор **mu**, що включає середнє **mu(1)** і стандартну помилку **mu(2)**.

Отримані дані можуть бути використані для подальших визначень функцією **polyval**, яка також має кілька варіантів використання:

y = polyval(p,x);

[y,delta] = polyval(p,x,S);

y = polyval(p,x,[],mu);

[y,delta] = polyval(p,x,S,mu).

У найпростішому випадку функція приймає як аргументи значення коефіцієнтів полінома, так і значення аргументу, для яких потрібно обчислити значення. Функція **polyval** повертає обчислені значення полінома у заданих точках. При включенні до списку аргументів додаткових параметрів також буде обчислена змінна **delta** з метою оцінки стандартного відхилення. В інтервалі **y ± delta** знаходиться як мінімум половина обчислених значень величини **y**. Проведемо апроксимацію синусоїди за допомогою поліномів різного

степеня.

```
% Очищення всіх параметрів
clear all
% Сім точок синусоїди
x=[0,45,90,135,180,225,270];%Значення аргументу у градусах
y=sind(x);
x1=0:1:270;
% Апроксимація поліномами різного степеню
p1= polyfit(x,y,1);
p2= polyfit(x,y,2)
p3= polyfit(x,y,3)
p5= polyfit(x,y,5);
% Обчислення значень функцiЇ, що апроксимує
y1=polyval(p1,x1);
y3=polyval(p3,x1);
y5=polyval(p5,x1);
%y10=polyval(p7,x1);
% побудова графiків
figure(1)
grid on;
hold on;
xlabel('x');set(gca,'FontSize',18);
ylabel('sin(x)');set(gca,'FontSize',18);
set(gca,'FontSize',16,'fontname','Times New Roman','LineWidth',3);
plot(x,y,'Marker','s','LineWidth',3,'LineStyle','none');
plot(x1,y1,'LineWidth',2);
plot(x1,y3,'LineWidth',2);
plot(x1,y5,'LineWidth',2);
legend('sin(x)','n=1' ,'n=3','n=5');
```

Слід звернути увагу на те, що якщо степінь полінома спів падає з кількістю експериментальних точок, то вдається досягти точного співпадіння аналітичних та експериментальних значень. Проте це не говорить про кращу відповідність функції, яка апроксимує, експерименту.

З рис.5.3 видно, що, незважаючи на точний збіг для полінома сьомого степеня, краще передає властивості поліном третього степеня. Збіг експериментальних та розрахункових даних

забезпечується не збільшенням числа коефіцієнта у функції, а її фізично обґрунтованим підбором.

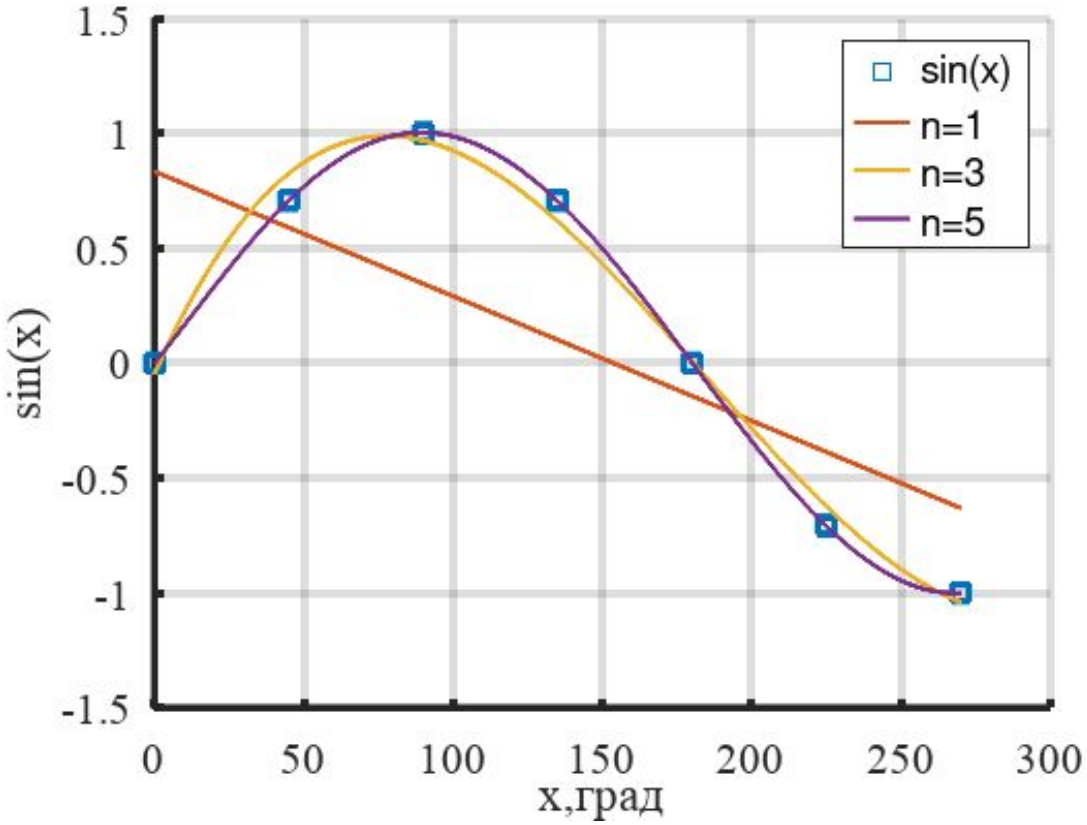


Рисунок 5.3 — Апроксимація синусоїди поліномом

Для вирішення завдань **підбору аналітичних залежностей** за експериментальними даними **методом найменших квадратів** можна з функцією **polyfit** додатково використовувати функції Octave:

corr(x,y) — функція обчислення коефіцієнта кореляції (x — масив абсцис експериментальних точок, y — масив ординат експериментальних точок);

mean(x) — функція визначення середньо арифметичного.

Так, для експериментальної лінійної залежності електричної ємності кабелю від вмісту вологи визначені значення ємності кабелю при вмісті вологи 42% (рис. 5.4, *а*) та 62% (рис.5.4, *б*) із застосуванням функції **polyfit(x,y,n)**.

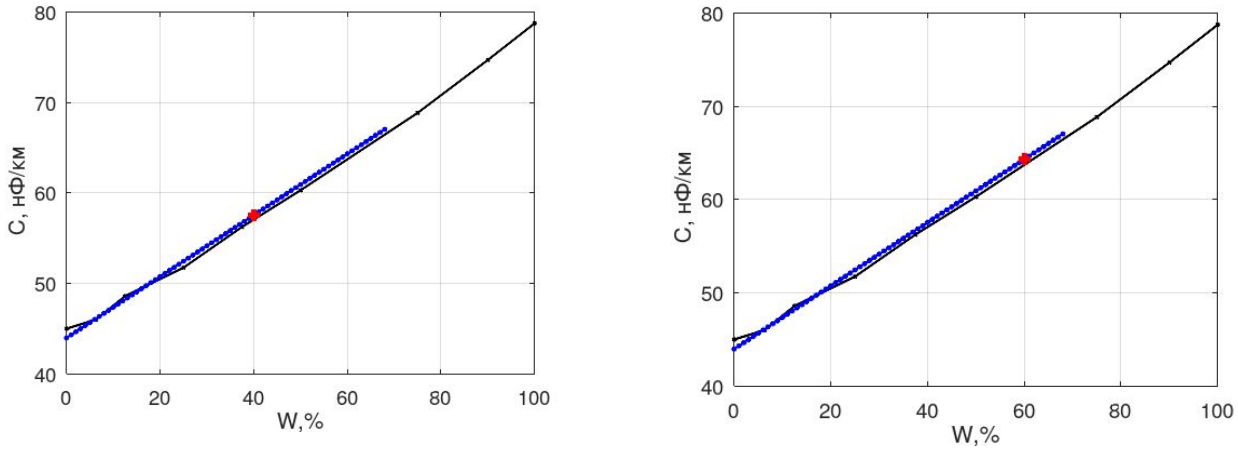


Рисунок 5.4 — Підбор за експериментальними даними аналітичної залежності функцією **polyfit(x,y,n)**

На рис. 5.4 в одній графічній області представлені експериментальна залежність (чорним кольором), лінії регресії поліномів $y=a_1*x+a_2$ (синім кольором) та значення ємності в шуканій точці (червоним кольором). Коефіцієнт кореляції дорівнює 1 для двох випадків. Коефіцієнт кореляції ***r*** є мірою зв'язку для лінійної форми залежності.

Для експериментальних даних, що представлено на рис. 5.5, задача апроксимації ускладнюється: не піддається лінеаризації, тобто не зводиться до лінійної регресії. У такому випадку розв'язується оптимізаційна задача пошуку мінімуму функції, що апроксимує, у вигляді $Y=a\,x^b\exp(cx)$, параметри b та c котрої входять нелінійно в експериментальну залежність (рис.5.5, крива 1).

Застосовано алгоритм методу найменших квадратів за допомогою функції **sqp**.

Коефіцієнт кореляції r застосовується лише у випадках, коли між даними існує прямолінійний зв'язок. У разі нелінійного зв'язку для виявлення тісноти зв'язку між змінними ***y*** та ***x*** у разі їх нелінійної залежності застосовується **індекс кореляції *R***.

Індекс кореляції знаходиться у межах від 0 до 1. За наявності функціональної залежності **індекс кореляції близький до 1.** За відсутності зв'язку ***R*** практично дорівнює нулю. Індекс кореляції ***R*** застосовується як для лінійної, так і для нелінійної залежності. При прямолінійному зв'язку коефіцієнт кореляції за своєю абсолютною величиною дорівнює індексу кореляції: **|*r*| = *R***.

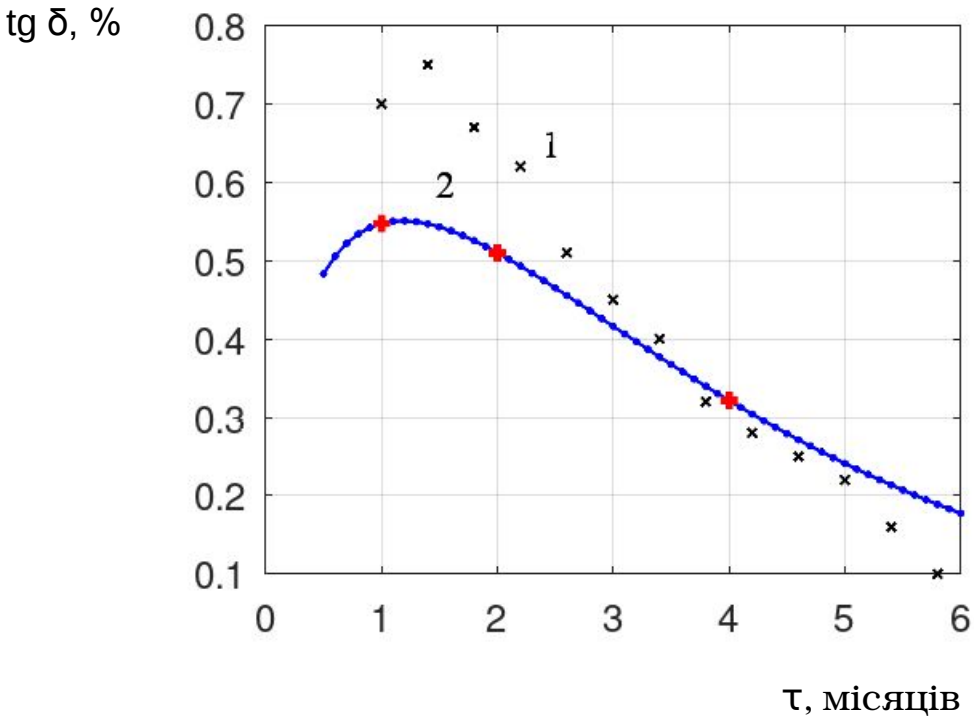


Рисунок 5.5 — Степінь відповідності між експериментальними даними та аналітичного залежністю функцією **polyfit(x,y,n)**

На рис. 5.5 представлено результати апроксимації (крива 2) експериментальної залежності (крива 1) тангенсу кута діелектричних втрат для частоти 100 Гц в процесі теплової сушки при температурі 50°С зразка кабелю після експлуатації в умовах підвищеної вологості на АЕС.

Пошук мінімуму функції $Y = a\,x^{b}\exp(cx)$ дозволив отримати функцію, яка апроксимує експериментальну нелінійну залежність,

$Y = -0.2726\,x^{0.2062}\exp(-0.2278\,x)$ з параметрами: *a*=-0,2726; *b*=0,2062; *c*=-0,2278. Значення сумарної квадратичної похибки становить 0,01423. Індекс кореляції дорівнює ***R*=0,9947**.

5.3. Реалізація алгоритму Левенберга - Марквардта в Octave

При більш складному виборі функції, яка апроксимує, аналітичний пошук її коефіцієнтів (особливо у разі багатьох змінних) в принципі неможливий, проте рішення може бути знайдено чисельними методами. І тут на підставі відповідного методу реалізується ітераційний процес, що дозволяє підібрати параметри апроксимувальної функції, що забезпечують мінімізацію суми [22—24].

Одним з перших методів пошуку був метод Ньютона та його модифікації.

Особливості методу Ньютона: швидко сходиться (квадратична збіжність). У разі кількох коренів може бути знайдено будь-яке з них (залежить від початкового наближення). Але швидкість збіжності (і сама збіжність) може залежати від початкового наближення

Згодом з'явилося сімейство методів на основі градієнтного спуску. В даний час для оптимізації часто застосовують методи з урахуванням генетичного алгоритму. У кожного з методів є свої переваги і недоліки, тому для оптимізації часто застосовують **комбіновані алгоритми**. Одним з таких алгоритмів є алгоритм Левенберга – Марквардта [25-26], що комбінує метод Ньютона і метод градієнтного спуску.

Метод Левенберга – Марквардта відрізняється від методу Ньютона більш повільнішою сходимістю. Перевага – менш вимогливий до початкового наближення. Широко застосовується практично. За цим методом кожна ітерація включає підбір кроку:

значення ітерація виконується за початковим, малим значенням кроку. Якщо значення функції зросло, то крок збільшується. У протилежному випадку приймається отримане значення та відбувається перехід до наступної ітерації.

У математиці та обчисленнях алгоритм Левенберга – Марквардта (LMA або просто LM), також відомий як загасаючі найменші квадрати (DLS), використовується для вирішення нелінійних задач найменших квадратів. **Ці проблеми мінімізації виникають, зокрема, за апроксимації кривої методом найменших квадратів**. LMA використовується в багатьох програмних додатках для вирішення загальних задач апроксимації кривої. Однак, як і в багатьох алгоритмах припасування, LMA знаходить лише локальний мінімум, який не обов'язково є глобальним мінімумом. LMA інтерполює між алгоритмом Гаусса – Ньютона (GNA) та методом градієнтного спуску. LMA надійніший, ніж GNA. Це означає, що в багатьох випадках він знаходить рішення, навіть якщо воно починається дуже далеко від кінцевого мінімуму. Для коректних функцій і коректних початкових параметрів LMA зазвичай трохи повільніше, ніж GNA. Алгоритм був вперше опублікований в 1944 Кеннетом Левенбергом, коли він працював у Франкфордському армійському арсеналі. Він був повторно відкритий у 1963 році Дональдом Марквардтом, який працював статистиком у DuPont, та незалежно Жираром, Вінним та Моррісоном.

Розглянемо реалізацію алгоритму апроксимації на прикладі апроксимації залежності в'язкості η трансформаторної оливи від температури T формулою Френкеля [27]:

$$\left(\eta = \eta_0 \exp\left(A/T\right)\right) \qquad ,(5.9)$$

де η_0 — в'язкість при температурі 20° С, А — енергія активації, Т — температура.

Динамічна в'язкість η (Па·с) визначає опір рідини потоку - це тангенціальна сила на одиницю площі, необхідна для переміщення однієї площини повз іншу з одиничною швидкістю на одиничній відстані один від одного. Коли одна площина рухається повз іншу в рідині, між двома шарами встановлюється градієнт швидкості. В'язкість можна розглядати як коефіцієнт опору, пропорційний цьому градієнту.

Кінематична в'язкість — це відношення динамічної в'язкості η рідини до густини ρ рідини при однаковій температурі. Це міра опору потоку рідини під дією сили тяжіння. Одиницею кінематичної в'язкості є $м^2/с$.

У табл. 5.1 представлено значення динамічної в'язкості рідин, які часто застосовуються при атмосферному тиску.

Таблиця 5.1

Рідина	Значення η, 10^{-3} Па· с				
Температура рідини,° С	0° С	20° С	50° С	70° С	100° С
Ацетон	-	0.32	0.25	-	-
Бензин	0.73	0.52	0.37	0.26	0.22
Вода	29221	43101	0.55	0.41	0.28
Гліцерин	12100	1480	180	59	13
Гас	43133	43221	0.95	0.75	0.54
Кислота уксусна	-	43132	0.62	0.50	0.38
Масло касторове	-	987	129	49	-
Ртуть	-	19725	14611	-	45292

Масло трансформаторне є спеціально розробленим діелектриком, який виконує кілька ключових функцій в електроенергетиці. Воно служить не тільки для ефективної ізоляції обмоток трансформаторів від інших металевих елементів, але також має здатність ефективно охолоджувати працююче обладнання та запобігати корозії внутрішніх компонентів. Таким чином, трансформаторне масло відіграє ключову роль у підтримці надійної та безаварійної роботи електричних систем. Трансформаторне масло отримують фракційною перегонкою і подальшим аналізом сирої нафти. Трансформаторне масло складається з органічних сполук, а саме нафтенів, ароматичних вуглеводнів і олефінів. Всі ці сполуки є вуглеводнями, тому трансформаторне масло є чистим вуглеводневим мінеральним маслом.

На початку програми очищається екран, простір імен, вимикається посторінковий висновок, потім підключаються необхідні математичні пакети і визначається інтерфейс побудови графіків (GNU Octave може використовувати як власні, так і зовнішні засоби побудови графіків). Після цього йдуть два вектори, що містять експериментальні дані про залежність в'язкості від температури у градусах Кельвіна. Наступним кроком визначається функція, що обчислює в'язкість у відповідності з рівнянням Френкеля (рис. 5.6). Ця функція повинна задовольняти деяким умовам:

— як вхідні дані вона повинна містити не тільки температуру, а й коефіцієнти, пов'язані з константами у рівнянні Френкеля.

У наведеному прикладі це **koefficient(1)**, що відповідає η_0, і **koefficient(2)**, що відповідає *A*;

— як вихідні параметри функція повинна містити відповідні значення в'язкості.

Потім визначаються значення параметрів, які будуть

використані як початкові для процесу ітерації. До вибору цих параметрів необхідно поставитися відповідально: якщо початкові дані взяті дуже далеко від значень, що забезпечують мінімум, тобто раціональний процес може або не зійтися, або вимагати значної кількості ітерацій. Крім того, якщо функція, яка апроксимує, має декілька мінімумів, існує велика ймовірність, що замість загального мінімуму буде знайдено локальний. Після викликається функція **leasqr**, що забезпечує мінімізацію суми квадратів відхилень за алгоритмом Левенберга – Марквардт (рис.5.6)

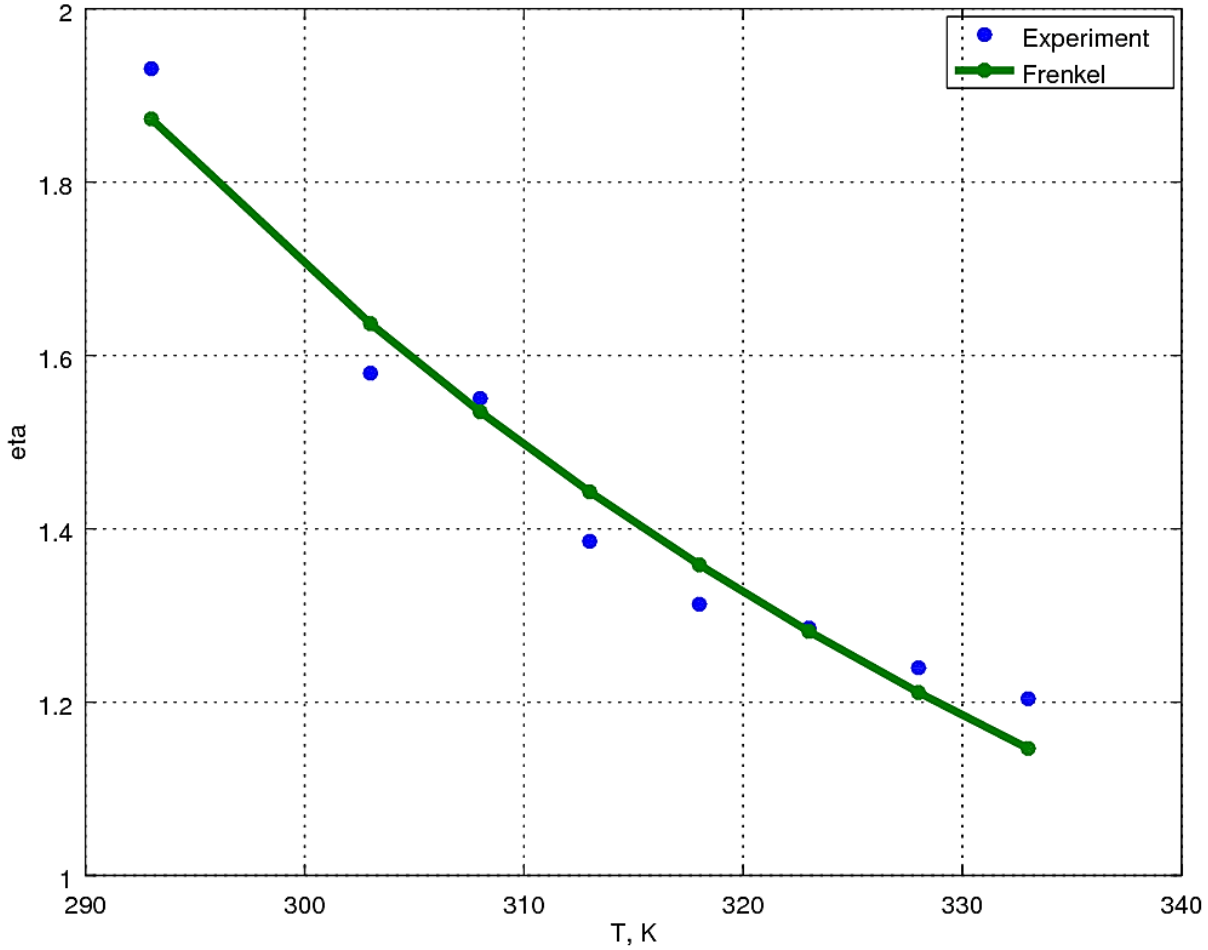


Рисунок 5.6 — Апроксимація експериментальних даних із застосуванням функції **leasqr**

Алгоритм Левенберга – Марквардта, що комбінує метод Ньютона і метод градієнтного спуску, реалізовано в GNU Octave за допомогою

функції **polyfit**:

[yfit pfit cvg iter]=leasqr (x, y, pin, F, stol, niter).

Для представленого прикладу аргументи функції — це:

x — вектор незалежних змінних (аргументів) функції;

y — вектор залежних змінних (значень) функції;

pin —початкові значення параметрів, що оптимізуються;

F — ім'я функції чи покажчик її. Структура функції обов'язково повинна бути наступною: **y = f(x, p)**. Параметри апроксимуючої функції, що підлягають оптимізації, повинні бути передані їй як аргументи. Функція **leasqr** обчислюватиме дані параметри таким чином, щоб сума квадратів відхилень між експериментальними значеннями функції **y** та обчисленими значеннями **yfit** мала мінімальне значення відповідно до (5.1);

stol — максимально допустиме відхилення для ітераційного процесу;

niter — максимальна кількість допустимих ітерацій до завершення.

Функція повертає такі параметри:

yfit — значення функції, що відповідають вхідним незалежним змінним **x**, які обчислені при оптимізованому значенні параметра **p**;

pfit – значення параметра **p**, які забезпечують мінімізацію суми квадратів відхилень;

cvg — параметр контролю; cvg=1 — ітераційний процес зійшовся; cvg = 0 — у протилежному випадку;

iter — кількість досконалих ітерацій.

Можливе і детальніше налаштування роботи функції **leasqr** із визначенням додаткових параметрів. На рис. 5.6 представлено результати мінімізації суми квадратів відхилень за алгоритмом Левенберга – Марквардт із застосуванням функції **leasq**.

Контрольні запитання до розділу 5

1. У чому сенс інтерполяції шляхом найближчого сусіда?

2. В яких випадках виправдане застосування методу інтерполяції шляхом найближчого сусіда?

3. За допомогою якої функції в GNU Octave реалізовано одновимірну інтерполяцію?

4. У який спосіб в GNU Octave реалізується поліноміальна інтерполяція?

5. У чому особливість функції **interp1** процедура екстраполяції експериментальних даних?

6. Що таке апроксимація та чим відрізняється від інтерполяції?

7. Як реалізовано метод найменших квадратів у GNU Octave?

8. У чому особливість застосування алгоритму методу найменших квадратів за допомогою функції **sqp?**

9. Чим відрізняються коефіцієнт та індекс кореляції?

10. Які особливості алгоритму Левенберга – Марквардта?

11. У який спосіб відбувається реалізація алгоритму Левенберга – Марквардта у GNU Octave?

СПИСОК ЛІТЕРАТУРИ

1.Eaton J.W., BatemanD., Hauberg S., Wehbring R. GNU Octave A high-level interactive language for numerical computations. Boston: MA, 2017. –1004 p.

2. Jesper Schmidt Hansen GNU Octave Beginner's Guide / Jesper Schmidt Hansen, Packt Publishing Ltd., 32 Lincoln Road, Olton, Birmingham, B27 6PA, UK. 2011 – 280 p.

3. Stahel Andreas Octave and MATLAB for Engineers. Bern University of Applied Sciences, Switzerland. Version of 10th September, 2020 . - 321 p.

4. Katey Birtcher, Stephen Merken, Nate McFadden, S. T. Sambandamm, Greg Harris MATLAB® A Practical Introduction to Programming and Problem Solving. Boston University, 2019 – 610 p.

5. Holly Moore MATLAB for Engineers. Salt Lake Community College, Pearson Education Inc, 2022 – 672 p.

6. Ashwin Pajankar, Sharvani Chandu GNU Octave by Example: A Fast and Practical Approach to Learning GNU Octave. Pittsburgh, PA, USA, 2020 – 173 p.

7. Spiegel, Murray R. Mathematical Handbook of Formulas and Tablets. New York: McGraw Hill Book Company, 1968.

8. Octave Programming Tutorial / Vectors and matrices. – онлайн ресурс: https://en.wikibooks.org/wiki/Octave_Programming_Tutorial/Vectors_and_matrices

9. Железнякова Е. Ю., Норік Л. О. Вища математика в GNU Octave [Електронний ресурс]: навчально-практичний посібник. – Харків : ХНЕУ ім. С. Кузнеця, 2024. – 276 с.

10. Чікіна Н.О., Антонова І.В. Функції декількох змінних. Скалярні поля: навчально-методичний посібник для студентів технічних спеціальностей усіх форм навчання. – Харків: НТУ «ХПІ», 2023. – 84 с.

11. Савченко В.М., Маций О.Б., Мнушка О.В. Системний аналіз та математичне моделювання в GNU Octave: навчально-практичний посібник. – Харків: ХНАДУ, 2020. 128 с.
12. Рубан А. І., Гогоці Ю. Г., Гусак О. Г. Теорія поля: підручник. – Суми: Сумський державний університет, 2023. – 279 с.
13. Мілих В.І., Шавьолкін О.О. Електротехніка, електроніка та мікропроцесорна техніка. Підручник. - Київ: Каравела, 2018. – 688 с.
14. Мілих В.І. Електромагнітні поля, параметри та процеси в електротехнічних пристроях. Підручник. - Харків: ФОП Панов А.М., 2020. – 396с.
15. Болюх В.Ф. Основи електротехніки, електроніки та мікропроцесорної техніки: навч. посіб / В.Ф. Болюх, В.Г. Данько, Є.В. Гончаров; за ред. В.Г.Данька: НТУ «ХПІ» – Харків: Планета-Прінт, 2019. – 248 с.
16. Карпов Ю. О., Ведміцький Ю. Г., Кухарчук В. В. Теоретичні основи електротехніки. Електромагнітне поле: підручник. Херсон: ОЛДІ-ПЛЮС, 2014. - 407с.
17. Багацька О. В., Бутрим О. Ю., Колчигін М. М. Теоретична електродинаміка: підручник. Харків: ХНУ імені В. Н. Каразіна, 2017. - 414 с.
18. Hippel R. Dielectrics and Waves. Boston, London: Artech House, 1954. - 280 p.
19. Поплавко Ю.М. Фізика диелектриків. К: НТУУ «КПІ», 2015. - 572 с.
20. Безпрозванних Г.В., Мірчук І.А. Синтез технологічних режимів охолодження та радіаційного опромінення електричної ізоляції кабелів: Монографія, видавництво ТОВ «Друкарня «Мадрид», м. Харків, 2021 р. - 179 с.
21. Безпрозванних Г.В., Рогинський О.В. Конструктивно-технологічні рішення підвищення електричних характеристик високовольтної

композитної електроізоляційної системи електричних машин. Монографія, видавництво ТОВ «Друкарня «Мадрид», м. Харків, 2023 р. - 138 с.
22. Nocedal J., Wright, S. J. Numerical Optimization. 2006. Springer New York. http://dx.doi.org/10.1007/978-0-387-40065-5.
23. Rao Singiresu S. Engineering Optimization: Theory and Practice. - New Jersey, USA: John Wiley & Sons, 2009. – 813 p.
24. Rahmat-Samii Yahya, Eric Michielssen Electromagnetic optimization by genetic algorithm. – New York, USA: John Wiley & Sons, 1999. – 480 p.
25. Levenberg K. A method for the solution of certain non-linear problems in least squares. Quarterly of Applied Mathematics. 1944. V. 2. N 2. P.164-168. https://doi.org/10.1090/qam/10666
26. Marquardt D. W. An Algorithm for Least-Squares Estimation of Nonlinear Parameters. J. Soc. Indust. Appl. Math. 1963. V. 11. N 2. P. 431-441.
27. Поплавко Ю.М., Переверзєва Л.П., Воронов С.О., Якименко Ю.І. Фізичне матеріалознавство. Частина друга. Діелектрики. Київ: НТУУ «КПІ», 2007. – 592 с.

Навчальне видання

БЕЗПРОЗВАННИХ Ганна Вікторівна

МОСКВІТІН Євген Сергійович

ПРИКЛАДНЕ ПРОГРАМУВАННЯ В GNU Octave

Навчальний посібник
для студентів спеціальності
141 «Електроенергетика, електротехніка
та електромеханіка»

Відповідальний за випуск доц. Кєссаєв О. Г.

Роботу до видання рекомендував проф. Мілих В. І.

В авторській редакції

План 2024 р., поз. 127

Підписано до друку __.__.2025 р.
Друк. цифровий. Гарнітура Bookman Old Style. Ум. друк. арк. 201

Видавничий центр НТУ «ХПІ».
Свідоцтво про державну реєстрацію ДК № 5478 від 21.08.2017 р.
61002, Харків. вул. Кирпичова, 2

Електронне видання