МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Київський національний університет будівництва і архітектури

Ю.В. Медведський

ПРОГРАМНІ КОМПЛЕКСИ ІНЖЕНЕРНИХ РОЗРАХУНКІВ

Конспект лекцій
для здобувачів першого (бакалаврського) рівня
вищої освіти спеціальності
193 «Геодезія та землеустрій»

Київ 2025

УДК 6339.5(075.8)
М42

Рецензент А. О. Анненков, д-р техн. наук, професор

Затверджено на засіданні кафедри інженерної геодезії, протокол № 8 від 27 березня 2024 року.

В авторській редакції.

Медведський Ю. В.

М42 Програмні комплекси інженерних розрахунків [Електронний ресурс]: конспект лекцій / Ю. В. Медведський. – Київ : КНУБА, 2025. – 132 с.

Розглянуто питання використанні теорії математики та розроблених на її основі алгоритмів під час виконання обчислень в геодезичних задачах та вимірюваннях, застосовуючи пакети прикладних програм для числового аналізу, систем комп'ютерної алгебри та засобів статистичного аналізу.

Призначено для здобувачів першого (бакалаврського) рівня вищої освіти спеціальності 193 «Геодезія та землеустрій».

УДК 6 339.5 (075.8)

ЗМІСТ

ВСТУП

Програмні комплекси інженерних розрахунків – це дисципліна, яка базується на використанні теорії математики та розроблених на її основі алгоритмів при виконанні обчислень в геодезичних задачах та вимірюваннях, застосовуючи пакети прикладних програм для числового аналізу, систем комп'ютерної алгебри та засобів статистичного аналізу.

Мета освітньої компоненти – ознайомити студентів з можливостями програмних засобів з автоматизації математичних розрахунків для вирішення наукових та прикладних задач.

Завдання освітньої компоненти полягає в проведенні розрахунків, що застосовуються в обчисленнях геодезичних задач, за допомогою пакети прикладних програм для числового аналізу та засобів мови програмування Python.

Компетентності здобувачів освітньої програми, що формуються в результаті засвоєння освітньої компоненти

Код	Зміст компетентності
Інтегральна компетентність	
ІК	Здатність розв’язувати складні спеціалізовані задачі геодезії та землеустрою
Загальні компетентності	
ЗК01	Здатність вчитися і оволодівати сучасними знаннями.
ЗК02	Здатність застосовувати знання у практичних ситуаціях.
ЗК06	Здатність використовувати інформаційні та комунікаційні технології.
ЗК07	Здатність працювати автономно.
Фахові компетентності	
СК05	Здатність застосовувати сучасне інформаційне, технічне і технологічне забезпечення для вирішення складних питань геодезії та землеустрою, кадастру.
СК06	Здатність виконувати дистанційні, наземні, польові та камеральні дослідження, інженерні розрахунки з опрацювання результатів досліджень, оформляти результати досліджень, готувати звіти при вирішенні завдань геодезії та землеустрою.

Програмні результати здобувачів освітньої програми, що формуються в результаті засвоєння освітньої компоненти

Код	Програмні результати
РН09	Збирати, оцінювати, інтерпретувати та використовувати геопросторові дані, метадані щодо об’єктів природного і техногенного походження, застосовувати статистичні методи їхнього аналізу для розв’язання спеціалізованих задач у сфері геодезії та землеустрою.
РН10	Обирати і застосовувати інструменти, обладнання, устаткування та програмне забезпечення, які необхідні для дистанційних, наземних, польових і камеральних досліджень у сфері геодезії та землеустрою.

У рамках цього курсу ***виокремлюють три змістових модулі***:

1. Початок роботи з MATLAB та MATHCAD.
2. Початок роботи з Python.
3. Основи статистики.

ЗМІСТОВНИЙ МОДУЛЬ 1. ПОЧАТОК РОБОТИ З MATLAB ТА MATHCAD.

Лекція 1. Початок роботи з Mathcad

План

1. Арифметичні обчислення. Використання змінних.
2. Створення матриць та найпростіші операції над ними.
3. Побудова виразів та графіків.
4. Робота з векторами.

1. Арифметичні обчислення. Використання змінних

MathCAD має *спеціалізовану вхідну мову програмування* дуже високого рівня, орієнтовану на математичні розрахунки. Тому, розглядаючи вхідну мову системи як мову програмування, ми можемо виділити типові поняття й об'єкти. До них відносяться ідентифікатори, константи, змінні, масиви й інші типи даних, оператори й функції, що управляють структурами й т.д. Чітке знання їхніх можливостей і правил застосування (синтаксис) досить корисне при рішенні завдань помірної й високої складності.

Алфавіт системи MathCAD містить:

- Великі й малі латинські літери;
- Великі й малі грецькі літери;
- Арабські цифри від 0 до 9;
- Системні змінні;
- Оператори;
- Імена вбудованих функцій;
- Спецзнаки;
- Рядкові й прописні букви кирилиці (при роботі з русифікованими документами).

До основних елементів мови відносяться *типи даних, оператори, функції користувача й керуючі структури*. До типів даних відносяться числові константи, звичайні й системні змінні, масиви (вектори й матриці) і дані файлового типу.

Для введення грецьких букв можна скористатися панеллю знаків **Greek**, що включає кнопкою  на панелі **Math**. Крім того, в MathCAD передбачена можливість набору грецьких букв за допомогою клавіш. Для цього досить набрати відповідну англійську букву й натиснути комбінацію клавіш **[Ctrl+G]**.

Числові константи

Константами називаються поіменовані об'єкти, що зберігають деякі значення, які не можуть бути змінені в ході виконання програми.

Числові константи задаються за допомогою арабських цифр, десяткової крапки (а не коми) і знаку - (мінус). Наприклад:

123 – цілочислова десяткова константа;

12.3 - десяткова константа із дробовою частиною;

$12.3*10^{-5}$ – десяткова константа з мантисою (12.3) і порядком –5.

В систему MathCAD включені дані строкового типу. *Строкова константа* - це рядок, який береться у лапки, наприклад: “My name”. У строкову константу можуть входити один або кілька символів або слів.

Змінні

Змінними називаються поіменовані об'єкти, що мають деяке значення, яке може змінюватися в ході виконання програми. Імена констант, змінних й інших об'єктів називають ***ідентифікаторами***. *Тип змінної* визначається її значенням; змінні можуть бути числовими, строковими, символьними тощо. Ідентифікатори в системі MathCAD мають практично будь-яку довжину, у них входять будь-які латинські й грецькі букви, а також цифри.

Оператори

Оператор – це спеціальний MathCAD -символ, що вказує на виконання певної операції з операндами. Вони можуть включати символи арифметичних операцій, знаки обчислення сум, добутків, похідної й інтеграла тощо. *Операндами* називаються величини, що беруть участь в різних операціях (додавання, ділення тощо). Після вказання операндів, оператори стають блоками, що виконуються в програмі. З різними видами операторів можна познайомитися в наступному розділі лекції.

Арифметичні оператори - призначені для виконання арифметичних дій над числовими величинами та конструювання математичних виразів (табл. 1.1).

Таблиця 1.1

Арифметичні оператори

Оператор	*Введення з клавіатури*	*Призначення оператора*
X :=Y	**X : Y**	Локальне присвоювання X значення Y
X ≡ Y	X ~Y	Глобальне присвоювання X значення Y
X =	**X =**	Результат обчислення значення X
-X	-X	Зміна знаку X
X + Y	X + Y	Підсумовування X та Y
X – Y	X – Y	Віднімання Y від X
X · Y	X * Y	Множення X на Y
X/Y	X / Y	Ділення X на Y
X^Y	X ^ Y	Піднесення X до степені Y
√ X	X \	Обчислення квадратного кореня з X
X !	X !	Обчислення факторіалу
(■)	‘	Введення пари круглих дужок із шаблоном
(	(	Введення відкриваючої дужки
)	)	Введення закриваючої дужки
X_n	X [n	Введення нижнього індексу n

Застосування розширених операторів значно полегшує рішення математичних завдань (табл. 1.2).

Таблиця 1.2

Розширені арифметичні оператори

Оператор	*Введення з клавіатури*	*Призначення оператора*
$\sum_{\blacksquare}^{\blacksquare}$	$	Обчислення суми
$\prod_{\blacksquare}^{\blacksquare}$	#	Обчислення добутку
$\frac{d}{d\blacksquare}\blacksquare$	?	Обчислення похідної
$\int_{\blacksquare}^{\blacksquare}\blacksquare\,d\blacksquare$	&	Обчислення визначеного інтегралу

Всі ***оператори відношень*** (табл. 1.3) можуть вводитися самостійно в місце розташування курсору. Необхідно відзначити, що вираз з логічним оператором повертає логічне значення, що відповідає виконанню або невиконанню умови, що задана оператором. Математично значення логічної одиниці і нуля збігаються зі значеннями числових констант 1 і 0.

Таблиця 1.3

Оператори відношень (логічні оператори)

Оператор	*Введення з клавіатури*	*Призначення оператора*
X > Y	X > Y	X більше Y
X < Y	X < Y	X менше Y
X ≥ Y	X Ctrl) Y	X більше або дорівнює Y
X ≤ Y	X Ctrl (Y	X менше або дорівнює Y
X ≠ Y	X Ctrl # Y	X не дорівнює Y
X = Y	X Ctrl = Y	X дорівнює Y

Вбудовані (стандартні) функції

MathCAD має безліч вбудованих функцій, які мають особливу властивість: у відповідь на звертання до них по імені із вказівкою аргументу вони повертають деяке значення у символьному, числовому, векторному або матричному вигляді. У систему MathCAD вбудований цілий ряд функцій, наприклад функція обчислення синуса **sin(*x*)** аргументу *x*, логарифма

натурального **ln(*x*)** аргументу *x* і тощо. Завдяки вбудованим функціям забезпечується розширення вхідної мови системи і її адаптація до завдань користувача.

Для вставлення стандартних функцій необхідно використати команду меню **Insert (Вставити) > Function (Функції)** або кнопку  С*тандартної панелі інструментів*.

Математичні вирази

Вирази являють собою складені за певними правилами комбінації констант, змінних, функцій, елементів масивів об'єднаних знаками математичних операцій.

Наприклад, у виразі:

Y:=2*ln(x)+1,

Y-змінна, **1** і **2** – числові константи, * і + - оператори, а **ln(x)** – вбудована функція з аргументом **x.**

Пріоритети виконання операцій

Вирази обчислюються зліва направо з урахуванням круглих дужок.

1. Дії в дужках.
2. Стандартні функції.
3. Піднесення до степеня (^).
4. Множення і ділення (* , /).
5. Ділення націло (\).
6. Залишок від ділення націло (mod).
7. Додавання і віднімання (+ , -).
8. Зчеплення рядків.
9. Операції порівняння.
10. Логічні операції

a. заперечення (Not);
b. логічне множення (And);
c. логічне додавання (Or).

Введення й редагування формул та тексту

У MathCAD -документі курсор введення має вигляд червоного хрестика. Цей хрестик вказує, у якому місці робочого аркуша буде зроблена наступна дія. Встановивши покажчик миші в потрібнім місці документа й

виконавши клацання, ви переміщуєте туди хрестик (можна використати стрілки, а не мишку). Покажчик у вигляді хрестика може приймати інші форми.

Він стає вертикальною рискою блакитного кольору $25 + \sin(\pi) = 25$ при введенні формули в області формул або при виборі вже існуючої формули. Переміщати цей блакитний курсор можна тільки за допомогою клавіш-стрілок.

Якщо при переміщенні червоного курсору-хрестика ви дістались в область формули, курсор автоматично приймає форму блакитного курсору формул.

Крім курсору формул у вашому розпорядженні перебуває курсор миші. За допомогою його можна тільки позиціювати курсор формул, як і курсор-хрестик, але не переміщати його.

При введенні текстової області (клавіша ["]) курсор-хрестик має вигляд вертикальної червоної риски. При цьому текстова область обмежена чорною рамкою.

Якщо ви вже вводите текст, забувши створити текстову область (**MathCAD** сприймає введений текст як формулу), та досить нажати клавішу пробілу, і MathCAD перетворить формулу в текст. Перетворення у зворотному напрямку неможливе.

Зупинимося докладніше на властивостях блакитного курсору формул. Для цього розглянемо приклад. Припустимо, що MathCAD не відома функція cosh (гіперболічний косинус), і нам необхідно ввести визначення:

$$f(x) = \frac{e^{x} + e^{-x}}{2}.$$

Введемо наступну послідовність символів:

f(x):((e^x)+(e^-x))/2.

Дужки тут необхідні: вони показують, до чого ставиться та.або інша операція. Якщо не вводити внутрішні дужки, то наступний за ***x*** вираз буде додано до показника степеня. Якщо ж опустити зовнішні дужки, то тільки другий доданок буде розділене на два. Однак в **MathCAD** передбачені економні методи редагування й введення. За допомогою клавіші пробілу можна збільшити область виділення - у блакитного курсору з'являється горизонтальний слід. Слід курсору дозволяє виділяти фрагменти формул уявними дужками таким чином, що наступна математична операція буде

виконуватися над всім виразом, відзначеним слідом курсору, тобто взятим в уявні дужки.

Присвоєння змінним значень

Звичайні змінні відрізняються від системних тим, що вони повинні бути попередньо визначені користувачем. Як оператор присвоєння використовується знак **:=** (знак локального присвоєння**)** Для введення стрілки можна використати клавішу **:** . До цього присвоєння змінна не визначена і її не можна використати. Область дії правіше і нижче цього оператора. За допомогою знака **≡** (три горизонтальні риски, вводиться клавішею **[~]** (тильда)) можна забезпечити глобальне присвоєння, тобто воно може виконуватися в будь-якому місці документа. Для виведення результату або для контролю значень змінних використовується звичайний знак рівності **=** (якщо виводиться числовий результат) або знак символьної рівності → (стрілка), якщо обчислення виконується в символьному виді. Для введення стрілки можна використати клавіші [**Ctrl+.**] або відповідну кнопку складальної панелі **Symbolic.**

При визначенні функцій користувача так само як і при визначенні змінних можуть бути використані знаки локального й глобального присвоєння. При цьому з використанням знака глобального присвоєння функція може бути визначена в будь-якому місці документа.

Визначення функцій користувача

Name_Func(*arg1,arg2,...,arg*):=Вираз,

Name_Func(*arg1,arg2,...,arg*) ≡ Вираз,

де **Name_Func** - ім'я функції; **arg1, ..., arg** - аргументи функції; **Вираз** - будь-який вираз, що містить доступні системі оператори й функції з операндами й аргументами, зазначеними в списку параметрів; **:= -** знак локального присвоєння**, ≡** - глобальне присвоєння.

Ранжировані змінні мають ряд фіксованих значень, або цілочисельних, або у вигляді чисел, з визначеним кроком від початкового значення до кінцевого.

Ранжировані змінні – це особливий клас змінних, котрий у системі **MathCAD** найчастіше заміняє керуючі структури, що називаються циклами.

Ранжировані змінні характеризуються ім'ям й індексом кожного свого елемента. Наприклад: **Ім'я:=*Nпочаткове .. Nкінцеве***, де **Ім'я** - ім'я змінної,

Nпочаткове - її початкове значення, ***Nкінцеве*** - кінцеве значення, **..** - символ, що вказує на зміну змінної в заданих межах (він вводиться знаком крапки з коми **;**). Якщо ***Nпочаткове < Nкінцеве*** , то крок зміни змінної буде +1, у протилежному випадку - (-1).

Для створення ранжированої змінної загального виду використовується вираз:

Ім'я:= ***Nпочаткове,(Nпочаткове +Крок).. Nкінцеве.***

де ***Крок***-заданий крок змінної.

Ранжировані змінні широко використовуються для подання числових значень функцій у вигляді таблиць, а також для побудови їхніх графіків. Будь-який вираз з ранжированими змінними після знаку рівності виводить таблицю результату.

Приклади виконання найпростіших обчислень наведено на рис. 1.1.

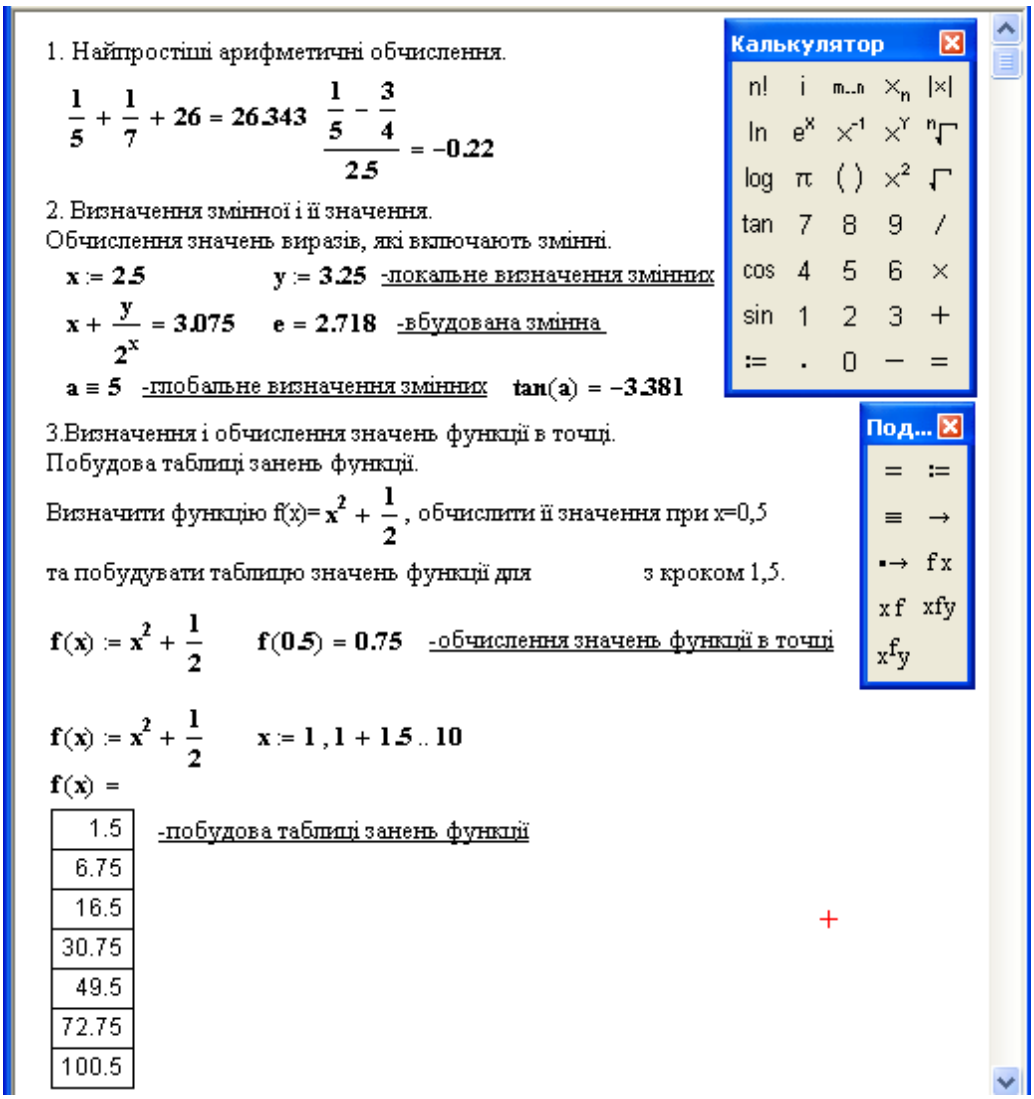


Рис. 1.1. Приклади виконання найпростіших обчислень

Корисно враховувати деякі властивості таблиць результату:

- Число рядків у них не може бути більше 50;
- Числа в таблицях можна задавати в необхідному форматі за допомогою операцій завдання формату чисел;
- При використанні в таблиці одиниць розмірності всі дані таблиці будуть містити одиниці розмірності.

2. Створення матриць та найпростіші операції над ними

Масив – це сукупність даних, що має унікальне ім'я, кінцеве число числових або символьних елементів, упорядкованих заданим чином, які мають певні адреси. У системі MathCAD використаються масиви двох типів: одновимірні (вектори) і двовимірні (матриці).

Індексація елементів масивів. Порядковий номер елемента, який є його адресою, називається індексом. Нижня границя індексації задається значенням системної змінної **ORIGIN,** що може приймати значення 0 або 1. Для зміни початку індексації можна прямо в документі присвоїти змінній **ORIGIN** відповідне значення або зробити це, через команду **Math** головного меню, підменю **Options** (Опції), використовуючи вкладку **Build-In Variables** (Вбудовані змінні).

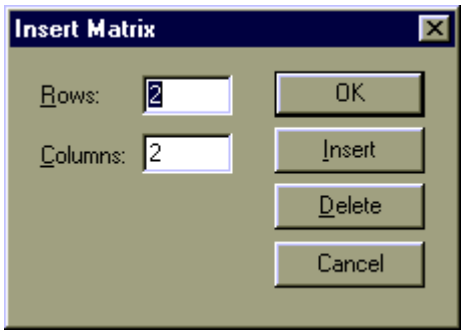


Рис. 1.2. Вставлення матриці

Вектори можуть бути двох типів: вектори – рядки й вектори – стовпці. Незважаючи на те що два цих вектори мають ті самі числові значення елементів, вони різні по типу й дадуть різні результати при векторних і матричних операціях.

Для введення векторів і матриць можна використати кнопку  панелі складальних математичних елементів **Matrix**, що у свою чергу, включається натисканням відповідної кнопки на панелі **Math**. Але набагато простіше використати сполучення клавишь **[Ctrl+M]**. Обидві вище зазначені дії

приводять до появи діалогового вікна **Insert Matrix**, у якому необхідно вказати число рядків і стовпців для введення матриці, або вектора.

У результаті в документі з'являється шаблон матриці, який можна заповнити необхідними даними. Перехід від символу до символу в середині шаблона відбувається за допомогою клавіші **Tab**(Табуляція). Масив можна визначити й вручну, поелементно. Для задання нижнього індексу використовується клавіша **[** (квадратна дужка) або кнопка  , що розміщується на  панелі . Якщо індекс подвійний (у матриці), то індекси вводяться через кому. Заповнення масивів може бути організоване за допомогою ранжированих змінних і функцій користувача.

Приклади виконання дій над матрицями та векторами наведено на рис. 1.3

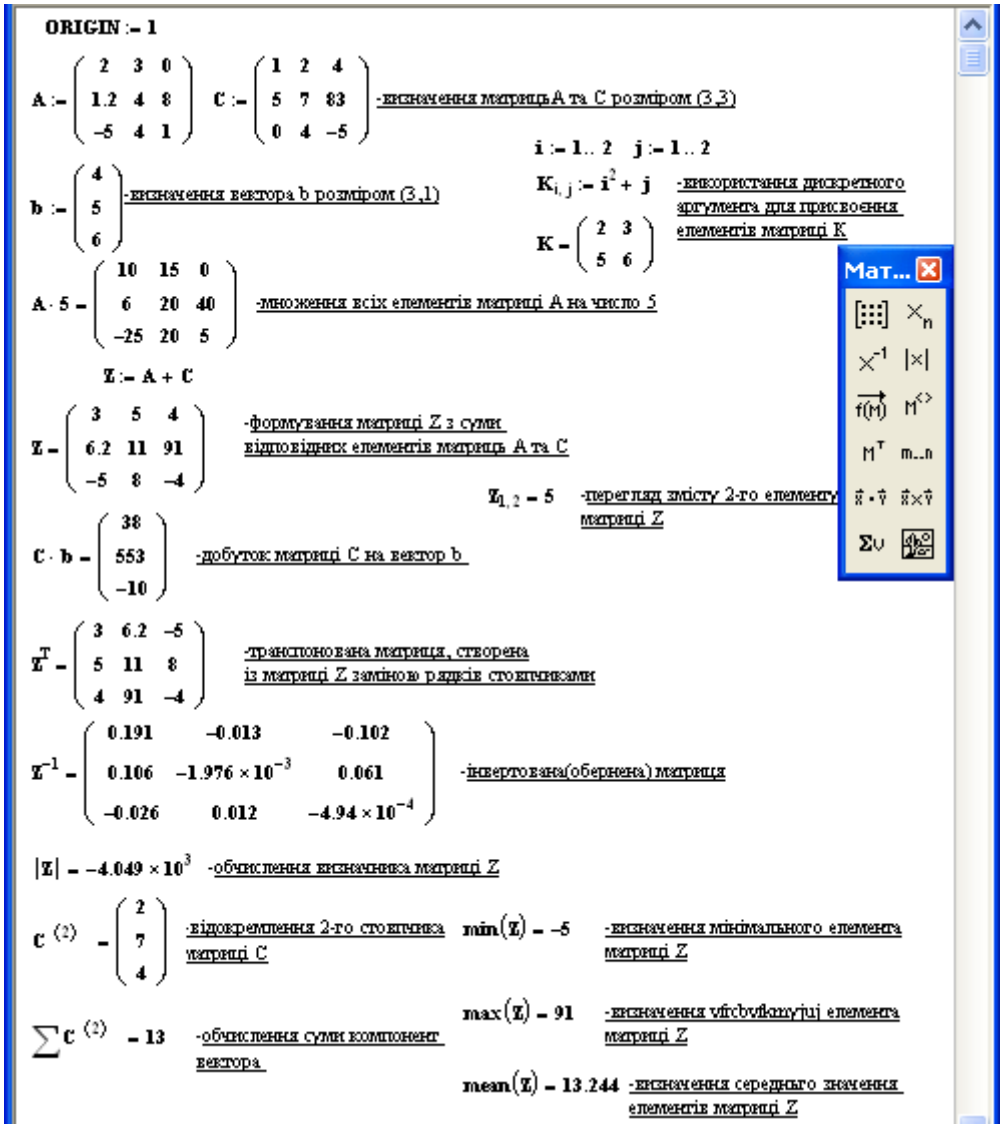


Рис. 1.3. Приклади виконання дій над матрицями та векторами

Для роботи з векторами та матрицями система MathCAD містить ряд операторів і функцій (табл. 1.4). Введемо наступні позначення: для векторів – **V**, для матриць – **M**, і для скалярних величин – **Z**.

Таблиця 1.4

Оператори і функції

Оператор	**Введення**	**Призначення оператора;**
V1+V2	V1+V2	Додавання двох векторів V1 і V2;
V1-V2	V1-V2	Різниця двох векторів V1 і V2;
-V	-V	Зміна знаку елементів вектора V;
-M	-M	Зміна знаку елементів матриці M;
V-Z	V-Z	Віднімання скаляра Z від вектора V;
Z*V, V*Z	Z*V, V*Z	Множення вектора V на скаляр Z;
Z*M, M*Z	Z*M, M*Z	Множення матриці M на скаляр Z;
V1*V2	V1*V2	Множення двох векторів V1 і V2;
M*V	M*V	Множення матриці M на вектор V;
M1*M2	M1*M2	Множення двох матриць M1 і M2;
V/Z	V/Z	Ділення вектора V на скаляр Z;
M/Z	M/Z	Ділення матриці M на скаляр Z;
M^{-1}	M^-1	Інверсія матриці M;
M^n	M^n	Піднесення матриці M до степені n;
$\sqrt{V}$	V\	Обчислення квадратного кореня з V;
| M|	| M	Обчислення визначника матриці M;
V^T	V Ctrl !	Транспонування вектора V;
M^T	M Ctrl !	Транспонування матриці M;
V	V Ctrl –	Векторизація вектора V;
M	M Ctrl –	Векторизація матриці M;
$M^{<n>}$	M Ctrl ^n	Виділення n-го стовпця матриці M;
V_n	V [n	Виділення n-го елемента вектора V;
$M_{m,n}$	M [(m,n)	Виділення елемента (m, n) матриці M.

Під поняттям “векторизація” мається на увазі одночасне проведення математичних операцій у їх скалярному вигляді над усіма елементами вектора або матриці. Це можна розуміти і як можливість паралельних обчислень.

Якщо **A** и **B** – вектори, то **A*B** дає скалярний добуток цих векторів. Але той же добуток під знаком векторизації створює новий вектор, кожен j-й елемент якого є добуток j-х елементів векторів **A** і **B**. Векторизація дозволяє виконувати скалярні оператори й функції з масивами.

Існує також ряд вбудованих векторних і матричних функцій (табл. 1.5).

Таблиця 1.5

Векторні функції, що входять у систему MathCAD

Функція	**Призначення функції**
lenght(V)	повертає довжину вектора;
last(V)	повертає індекс останнього елемента;
max(V)	повертає максимальний за значенням елемент;
min(V)	повертає мінімальний за значенням елемент;
Re(V)	повертає вектор дійсних частин вектора з комплексними елементами;
Im(V)	повертає вектор уявних частин вектора з комплексними елементами;
ε (*i, j, k*)	повністю асиметричний тензор розмірності три. ***i, j, k*** повинні бути цілими числами від 0 до 2 (або між >**ORIGIN** й **ORIGIN+2**, якщо **ORIGIN≠0)**. Результат дорівнює 0, якщо будь-які два аргументи рівні, 1 – якщо три аргументи є парною перестановкою (0, 1, 2), і мінус 1, якщо три аргументи є перестановкою (0, 1, 2), кратної 2 і некратної 4.

Для роботи з матрицями також існує ряд вбудованих функцій (табл. 1.6).

Таблиця 1.6

Вбудовані функцій

Функція	**Призначення функції**
Augment(M1, M2)	Поєднує в одну матриці **M**1 і **M**2, що мають однакове число рядків;
identity(*n*)	Створює одиничну квадратну матрицю розміром n*n;
stack(M1, M2)	Поєднує в одну матриці **M**1 і **M**2, що мають однакове число стовпців, розташовуючи **M**1 над **M**2;
submatrix(A,*ir,jr,ic,jc*)	повертає підматрицю, що складається із всіх елементів, що знаходяться в рядках від *ir* по *jr* і стовпцях з *ic* по *jc* (*irJjr* й *icJjc*);
diag(V)	Створює діагональну матрицю, елемент головної діагоналі якої – вектор **V**;
matrix(*m,n,*f)	Матрицю, у якій (*i,j*)-й елемент містить значення *f*(*i,j*), де *i*= 0, 1, …*m* і *j*=0, 1, …*n*;
Re(M)	Повертає матрицю дійсних частин матриці **M** с комплексними елементами;
Im(M)	Повертає матрицю уявних частин матриці **M** з комплексними елементами.

3. Побудова виразів та графіків

MathCAD дозволяє легко будувати двох - і тривимірні гістограми, двомірні графіки в декартових і полярних системах координат, тривимірні графіки поверхонь, лінії рівня поверхонь, зображення векторних полів, просторові криві.

Існує три способи побудови графіків у системі MathCAD:

- можна скористатися командою головного меню **Insert** (Вставити), вибравши в спадаючому меню команду **Graph** (Графік) і з списку - тип графіка;
- вибрати тип графіка на панелі **Graph** (Графік)  , що відкривається кнопкою на панелі **Math**;
- скористатися швидкими клавішами (вони передбачені не для всіх типів графіків).

Розглянемо більш докладно команди меню **Insert**(Вставити)->**Graph**(Графік) (ліворуч зображені відповідні кнопки складальної панелі **Math**):

 **X-Y Plot** (X-Y залежність) клавіша [@].

Призначена для побудови графіка функції однієї змінної в декартовій системі координат. Будується графік функції $y=f(x)$ у вигляді зв'язаних один з одним пар координат (x_i,y_i) при заданому проміжку зміни для i.

 **Polar Plot** (Полярні координати) клавіші [Ctrl+7].

Призначена для побудови графіка функції $r(q)$, заданої в полярних координатах, де полярний радіус r залежить від полярного кута q.

 **Surface Plot** (Поверхні) клавіші [Ctrl+2].

Призначена для подання функції $z=f(x,y)$ у вигляді поверхні в тривимірному просторі. При цьому повинні бути задані вектори значень x_i й y_j, а також визначена матриця виду $A_{i,j}=f(x_i,y_j)$. Ім'я матриці A вказується при заповненні рамки-шаблона. За допомогою цієї команди можна будувати параметричні графіки.

 **Contour Plot** (Контурний графік).

Будує діаграму ліній рівня функції виду $z=f(x,y)$, тобто відображає точки, у яких дана функція приймає фіксоване значення $z=const$.

 **3D Scatter Plot** (3D Точковий).

Служить для точкового подання матриці значень $A_{i,j}$ або відображення значень функції двох змінних у декартовій системі координат $z=f(x,y)$ у заданих точках. Ця команда може також використовуватися для побудови просторових кривих.

 **3D Bar Plot** (3D Діаграми).

Служить для подання матриці значень $A_{i,j}$ або відображення значень функції $z=f(x,y)$ у вигляді тривимірної гістограми.

 **Vector Field Plot** (Векторне поле).

Служить для подання двовимірних векторних полів $\boldsymbol{V}=(V_x, V_y)$. При цьому компоненти векторних полів V_x і V_y повинні бути представлені у вигляді матриць. За допомогою цієї команди можна побудувати поле градієнта функції $f(x,y)$.

3D Plot Wizard (виклик майстра для швидкої побудови 3-вимірного графіка).

При виборі цієї команди виникає ряд спливаючих вікон, у яких потрібно вибрати параметри побудови тривимірного графіка (задається тип тривимірного графіка, стиль його зображення, гама кольорів). Графік за замовчуванням будується на проміжку від -5 до +5 (по обох змінних).

Графік функції y=f(x) (рис. 1.4).

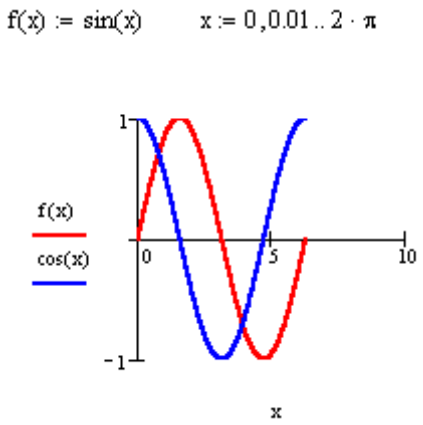


Рис. 1.4. Графік функції y=f(x)

При виконанні команди **Inset -> Graph -> Plot** у документі з'являється рамка-шаблон із двома незаповненими осередками для побудови графіка. (Клавіша [@]).

В осередку, що розташовується під віссю абсцис, вказується незалежна змінна ***x***. Її варто визначити заздалегідь як змінну, що приймає значення із проміжку (ранжирувана змінна).

В осередку поруч із віссю ординат необхідно задати функцію ***f(x)***, графік якої ми хочемо побудувати. Якщо ця функція була задана заздалегідь, то в осередок досить ввести ***f(x)***, у противному випадку варто ввести зображувану функцію в явному вигляді (наприклад, ***cos(x)***).

Після введення ***x*** і ***f(x)*** у графічній області з'являться ще чотири осередки, які не обов'язково заповнювати. MathCAD автоматично знаходить підходящі значення для x_{min}, x_{max}, y_{min}, y_{max}. Якщо ж запропоновані MathCAD значення вас не влаштовують, ви можете задати свої.

В MathCAD існує можливість будувати графік функції, не задаючи попередньо проміжок зміни незалежної змінної. За замовчуванням цей проміжок приймається рівним [-10, 10].

Для подання на одній діаграмі графіків декількох функцій необхідно виділити осередок поруч із віссю ординат і через кому ввести другу

функцію. За замовчуванням графік цієї функції буде представлений пунктирною лінією іншого кольору.

4. Робота з векторами

Тривимірні точкові графіки можна використати для побудови зображення просторових кривих. Просторові криві задаються, як правило, у вигляді *(x(t),y(t),z(t))*, де *t* являє собою неперервний дійсний параметр.

$$N := 36$$

$$i := 0 .. N - 1$$

$$x_i := \cos\left(\frac{i}{N} \cdot 6 \cdot \pi\right)$$

$$y_i := \sin\left(\frac{i}{N} \cdot 6 \cdot \pi\right)$$

$$z_i := \frac{i}{N} \cdot 3$$

Оскільки при побудові тривимірної точкової діаграми MathCAD дозволяє відображати на графіку тільки окремі точки та з'єднуючі їх лінії, необхідно спочатку визначити три вектори координат - x_i, y_i, z_i.

Просторова крива створюється командою **Insert -> Graph ->Scatter Plot** (3D Точковий). Можна використати складальну панель **Graph**, вибравши відповідну піктограму  . Для з'єднання точок необхідно на вкладці **Appearance** вікна форматування графіків вказати опцію **Line**.

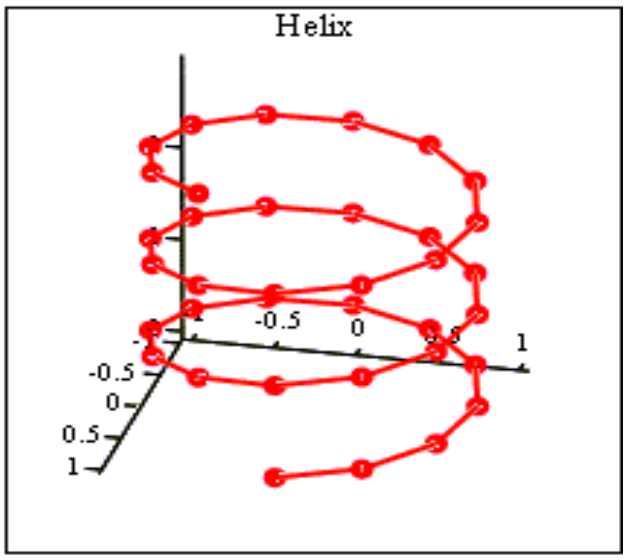


Рис. 1.5. Крива в просторі

Векторні та градієнтні поля

Команда **Insert -> Graph -> Vector Field Plot** (Векторне поле) служить для подання двовимірних векторних полів *v=(vx, vy)*.

$$v(x,y) := \begin{pmatrix} \frac{-y}{\sqrt{x^2+y^2}} \\ \frac{x}{\sqrt{x^2+y^2}} \end{pmatrix} \qquad N := 20 \qquad i := 0..N-1 \qquad j := 0..N-1$$

$$a := -2 \qquad b := 2 \qquad x_i := a + \frac{i}{N-1}\cdot(b-a)$$

$$c := -2 \qquad d := 2 \qquad y_j := c + \frac{j}{N-1}\cdot(d-c)$$

$$V_{i,j} := v\left(x_i, y_j\right)$$

$$vx_{i,j} := \left(V_{i,j}\right)_0 \qquad vy_{i,j} := \left(V_{i,j}\right)_1$$

При цьому векторне поле (рис. 1.6)необхідно початку визначити як функцію-вектор-функцію двох координат - *x* та *y*. Потім задаються вектори значень вузлових точок *x* та *y*. За допомогою цих векторів компоненти векторного поля *vx(x,y)* і *vy(x,y)* генеруються у вигляді матриць значень $vx_{i,j}$ і $vy_{i,j}$.

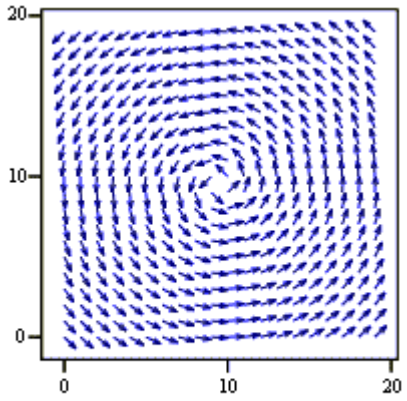


Рис. 1.6. Векторне поле

Подібним чином можна побудувати градієнтне поле скалярної функції *f(x,y)* (рис. 1.7). Градієнтне поле для функції двох змінних являє собою двовимірне векторне поле. Як і в інших випадках, зовнішній вигляд зображення векторного поля можна легко змінити, виконавши подвійне натискання миші в області графіка й змінивши необхідні опції в діалоговому вікні, що відкрилося, **3-D Plot Format**.

$$f(x,y) := \sin(x)\cdot\cos(y)$$

$$\text{grad}(x,y) := \begin{pmatrix} \frac{d}{dx} f(x,y) \\ \frac{d}{dy} f(x,y) \end{pmatrix} \quad \begin{matrix} Nx := 10 & a := -2 & b := 2 & i := 0..Nx-1 \\ Ny := 10 & c := -4 & d := 4 & j := 0..Ny-1 \end{matrix}$$

$$x_i := a + i\cdot\frac{b-a}{Nx-1} \qquad y_j := c + j\cdot\frac{d-c}{Ny-1}$$

$$F_{i,j} := f(x_i, y_j) \quad V_{i,j} := \text{grad}(x_i, y_j) \quad Vx_{i,j} := (V_{i,j})_0 \quad Vy_{i,j} := (V_{i,j})_1$$

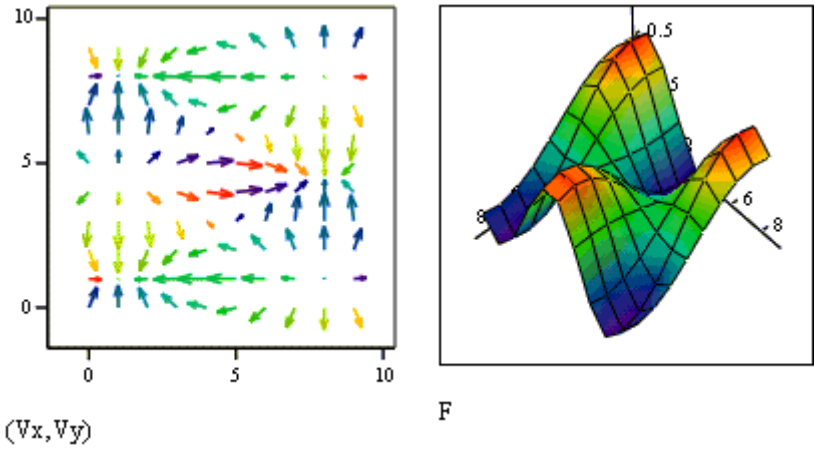


Рис. 1.7. Графік поверхні та її градієнтне поле

Запитання для самоконтролю

1. Як в Mathcad задаються числові змінні та вирази?
2. Які правила діють при виконанні арифметичних обчислень у Mathcad?
3. Як створити матрицю в Mathcad?
4. Які операції можна виконувати над матрицями?
5. Як створити вектор у Mathcad?
6. Чим відрізняється вектор від матриці у Mathcad?
7. Як побудувати графік функції?
8. Як задати вираз, що залежить від змінної?
9. Як побудувати 2D-графік у Mathcad?
10. Як змінити вигляд графіка (вісь, масштаб, підписи)?

Лекція 2. Основи роботи з Mathcad

План

1. Символьні розрахунки. Вирішення диференціальних рівнянь.
2. Використання циклів.
3. Використання конструкції if…elseif…else.

1. Символьні розрахунки. Вирішення диференціальних рівнянь

Введення в систему MathCAD символьних обчислень надає їй нові можливості, які були відсутні в колишніх версіях системи. Символьні обчислення в MathCAD можуть бути реалізовані трьома способами:

- З використання команд підменю позиції **Symbolics** (Символіка) головного меню.
- З використанням команд панелі **Symbolic** (Ключові слова символьних обчислень), що включається кнопкою  на математичній панелі інструментів. (Починаючи з версії MathCAD 4.0 для активних символьних обчислень застосовується термін *SMARTMATH*, що представляє комбінацію слів *smart* і *MathCAD*, що в буквальному значенні означає *розумний MathCAD*).
- З використанням команди **Optimization** позиції головного меню **Math**.

Команди меню Symbolics (Символьні операції)

Операції, що вимагаються від символьного процесора, знаходяться в підменю позиції **Symbolics** (Символьні операції)) головного меню.

Щоб символьні операції виконувалися, процесору необхідно вказати, над яким виразом ці операції повинні виконуватися, тобто потрібно виділити вираз. Для ряду операцій варто не тільки вказати вираз, над яким вони виконуються, але й виділити змінну, щодо якої виконується та або інша символьна операція.

Сам вираз в такому випадку не виділяється: адже й так ясно, що якщо маркер введення виділяє змінну якого-небудь виразу, то цей вираз вже відзначений наявністю в ньому виділеної змінної.

Слід зазначити деякі особливості при роботі з командами меню **Symbolics:**

- Для символьних обчислень вираз необхідно вказувати явно. Наприклад, неприпустимо вводити деяку функцію користувача *F(x)* і

намагатися знайти її похідні або інтеграл. Це важливе обмеження, про яке потрібно завжди пам'ятати. Однак воно виконується при виконанні обчислень за допомогою функцій системи *SmartMath***,** що описані далі; головне в тім, що для результату символьних обчислень у цьому випадку використовується оператор →.

- Іноді результат обчислень містить вбудовані в систему спеціальні мат. функції; у цьому випадку результат міститься в буфері обміну. Використовуючи команду **Paste** (Вставити) або клавішу F4, можна вставити вміст буфера обміну в документ, як текст і проаналізувати отриманий результат.
- До недоліків роботи з командами меню **Symbolics** варто віднести те, що це ручна робота, однокрокова. При подальшому використанні результатів символьних обчислень необхідно за допомогою операцій **Copy** (Копіювати) та **Past** (Вставити) присвоїти цей результат деякій змінній або функції. Крім того, при зміні формули, що піддається символьному перетворенню, результат (навіть при встановленому ***Автоматичному режимі обчислень***) не перераховується.
- Якщо операція нездійсненна - система виводить повідомлення про помилку або просто повторює виділений вираз (без змін).

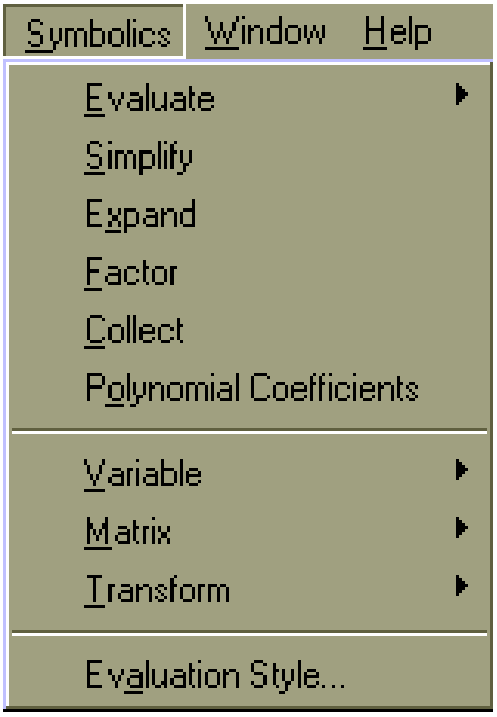


Рис. 2.1. Символьні операції

Операції над виділеним виразом

До операцій з виділеними виразом відносяться (табл. 2.1):

Таблиця 2.1

Операції з виділеними виразом

Операція	**Призначення операції**
Evaluate (Обчислити)	перетворити виразу з вибором виду перетворень із підменю
Simplify (Спростити)	спростити виділений вираз з виконанням таких операцій, як скорочення подібних доданків, зведення до загального знаменника, використання основних тригонометричних тотожностей тощо
Expand (Розкласти)	розкласти вираз [наприклад, для $(X+ Y) \cdot (X- Y)$ одержуємо X^2-Y^2]
Factor (Розкласти на множники)	розкласти число або вираз на множники [наприклад, $X^2- Y^2$ дасть $(X+ Y) \cdot (X- Y)$]
Collect (Зведення подібних доданків)	зібрати доданки, подібні до виділеного виразу, що може бути окремою змінної або функцією зі своїм аргументом (результатом буде вираз, поліноміальний щодо обраного виразу)
Polynomial Coefficients (Поліноміальні коефіцієнти)	знайти коефіцієнти полінома по заданій змінній

Операція **Evaluate** (Обчислити) містить підменю з наступними командами:

- **Evaluate Symbolically [Shift+F9]**(Обчислити в символах) — виконати символьне обчислення виразу;
- **Floating Point Evaluation...** (Із плаваючою точкою) - виконати арифметичні операції у виразі з результатом у формі числа із плаваючою точкою;
- **Complex Evaluation** (У комплексному вигляді) — виконати перетворення з поданням результату у комплексному вигляді.

Операції з виділеними змінними

Наступна група символьних операцій виконується над виразами, що вимагають вказівки змінної, стосовно якої виконується операція. Для цього досить встановити на змінній курсор введення. Саме вираз при цьому не вказується окремо, оскільки вказівка в ньому на змінну є одночасно і вказівкою на сам вираз. Якщо вираз містить інші змінні, то вони розглядаються як константи.

В пункті **Variable** (Змінна) об'єднані операції над виділеними змінними відносяться (табл. 2.2):

Таблиця 2.2

Операції над виділеними змінними

Solve (Розв'язання рівняння)	знайти значення виділеної змінної, при якій вираз що її містить стає рівним нулю (вирішити рівняння або нерівність щодо виділеної змінної)
Substitute (Заміна змінної)	замінити зазначену змінну вмістом буфера обміну;
Differentiate (Диференціювання)	диференціювати весь вираз, що містить виділену змінну, щодо цієї змінної (інші змінні розглядаються як константи);
Integrate (Інтегрування)	інтегрувати весь вираз, що містить виділену змінну, по цій змінній;
Expand to Series... (Розкласти в ряд)	знайти декілька перших членів розкладу функції в ряд Тейлора по виділеній змінній;
Convert to Partial Fraction (Розкласти на елементарні дроби)	розкласти на елементарні дроби вираз, який розглядається як раціональний дріб відносно виділеної змінної.

Оскільки символьні обчислення в MathCAD можна здійснювати в двох різних варіантах розглянемо їх на прикладах:

Приклад №1. *Розкласти вираз на співмножники за допомогою головного меню:*

1) Введіть вираз sin(2x);
2) Виділіть його повністю;
3) Виберіть в головному меню пункти **Symbolics => Expend** (Символьні операції => Розкласти)
4) Результат з'явиться трохи нижче у вигляді ще одного рядка.

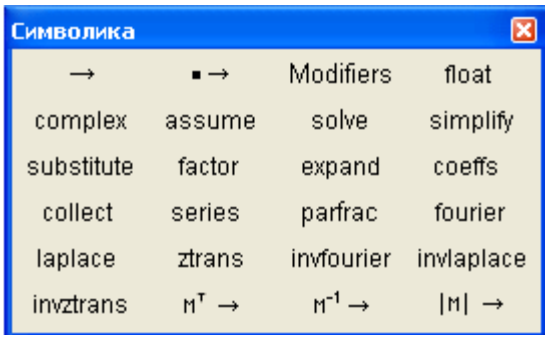
$$\sin(2 \cdot x)$$

$$2 \cdot \sin(x) \cdot \cos(x)$$

Приклад №2. *Розкласти вираз на співмножники за допомогою оператора →:*

1) Введіть вираз sin(2x)

2) Натисніть кнопку **Expand** (Розкласти) на панелі **Symbolic** (Символика).



$$\sin(2 \cdot x)\ \text{expand}, x \ \rightarrow 2 \cdot \sin(x) \cdot \cos(x)$$

Приклад №3. *Скорочення виразів (Simplify):*

1-спосіб за допомогою головного меню

1) Введіть вираз;

2) Виділіть вираз повністю, або тільки ту частину яку потрібно скоротити;

3) Виберіть команду **Symbolic=> Simplify**(Символика=> Спростити).

$$(x + 2 \cdot y) \cdot z - z^2 \cdot (x + 5 \cdot y) + z$$

$$z \cdot x + 2 \cdot z \cdot y - z^2 \cdot x - 5 \cdot z^2 \cdot y + z$$

2-спосіб за допомогою панелі **Symbolic**

$$(x + 2 \cdot y) \cdot z - z^2 \cdot (x + 5 \cdot y) + z\ \text{simplify}\ \rightarrow z \cdot x + 2 \cdot z \cdot y - z^2 \cdot x - 5 \cdot z^2 \cdot y + z$$

Приклад №4. *Знаходження коефіцієнтів полінома (Polynomial Coefficient) (Коефіцієнти полінома):*

1) Введіть вираз;

2) Натисніть кнопку ***Coeffs*** на панелі **Symbolic**;

3) Введіть в містозаповнювач після вставленого ключового слова ***Coeffs*** аргумент полінома.

$$(x+2\cdot y)\cdot z - z^2\cdot y\cdot(x+5\cdot y)\ \text{coeffs},z \rightarrow \begin{pmatrix} 0 \\ x+2\cdot y \\ -y\cdot x-5\cdot y^2 \end{pmatrix}$$

$$(x+2\cdot y)\cdot z - z^2\cdot y\cdot(x+5\cdot y)\ \text{coeffs},x \rightarrow \begin{pmatrix} 2\cdot z\cdot y-5\cdot z^2\cdot y^2 \\ z-z^2\cdot y \end{pmatrix}$$

Оптимізація обчислень досягається заміною складної функції або математичного виразу їхнім аналітичним поданням (якщо воно, звичайно, є). Для включення процесу оптимізації необхідно виділити вираз, що потрібно оптимізувати, і виконати команду **Optimization** позиції головного меню **Math**.

Ознакою оптимізації виразу (рис. 2.2) є поява після нього червоної зірочки. Крім того, клацнувши правою кнопкою миші й вибравши з контекстного меню команду **Show Popup** (Показати), можна спостерігати появу вікна з оптимізованим виразом.

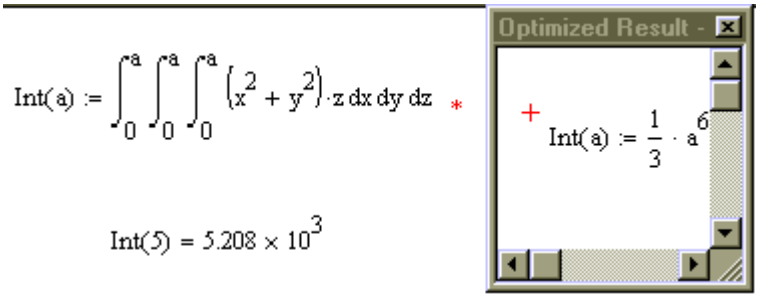


Рис. 2.2. Оптимізація виразу

Особливий виграш оптимізація може дати при багаторазовому обчисленні складних функцій, що містять інтеграли, похідні, суми, добутки й ряди.

Рівняння з одним невідомим.

Розглянемо алгебраїчне рівняння з однією невідомою ***f(x)=0***, наприклад, ***sin(x)²+cos(x)-1=0***.

Для рішення таких рівнянь MathCAD має вбудовану функцію ***root***, що, залежно від типу завдання, може включати два, або чотири аргументи й, відповідно, працює трохи по-різному.

- ***root(f(x),x);***
- ***root(f(x),x,a,b);***

де:

- ***f (x)*** - скалярна функція, що визначає рівняння ;
- ***x*** – аргумент функції відносно якого вирішується рівняння;

- ***a,b*** - границі інтервалу, всередині якого відбувається пошук кореня.

Перший тип функції ***root*** вимагає додаткового задання початкового значення (*guess value*) змінної ***x***. Для цього потрібно просто попередньо присвоїти ***x*** деяке числове значення. Пошук кореня буде проводитися поблизу цього числа. Таким чином, присвоєння початкового значення вимагає апріорної інформації про локалізацію (ізоляцію) кореня.

Приведемо приклад рішення дуже простого рівняння ***sin(x)²+cos(x)-1=0***, корені якого відомі заздалегідь (рис. 2.3).

$$x := 1.5$$

$$\mathrm{root}\left(\sin(x)^2 + \cos(x) - 1, x\right) = 1.57$$

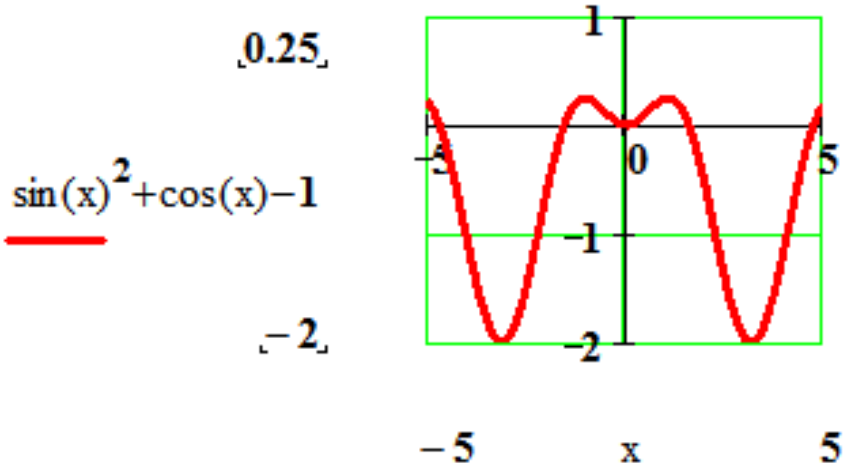


Рис. 2.3. Графічне рішення рівняння $\sin(x)^2+\cos(x)-1=0$.

Графік функції f(x)=**sin(x)²+cos(x)-1** і положення знайденого кореня показані на рис.13. Зверніть увагу, що, хоча рівняння має нескінченну кількість коренів ***xn*** (n=0,±1,±2,...), MathCAD знаходить (із заданою точністю) тільки один з них, ***x0***, що лежить найближче до ***x=1.5***. Якщо задати інше початкове значення, наприклад ***x=3***, то рішенням буде інший корінь рівняння.

2. Використання циклів

Оператор for

У циклі ***for*** кількість повторень циклу визначається змінною, яку потрібно задати на початку циклу. Ця змінна називається ***параметром циклу***. Розглянемо створення такого циклу:

- Встановіть курсор на вільне місце введення в програмі (праворуч від вертикальної риски);
- На панелі програмування натисніть кнопку ***for*** (Цикл for). З'явиться шаблон із трьома місцями для введення;
- Праворуч від слова ***for*** введіть ім'я змінної циклу. Після знака ∈ введіть діапазон зміни параметра циклу так само, як це робиться за допомогою дискретної змінної. Параметром циклу може бути ряд чисел, або вектор, або список скалярів, діапазонів, векторів, розділених комами;
- У поле, що залишилося (знизу, під словом ***for***) введіть вираз, що обчислюється в циклі;
- Якщо в циклі потрібно обчислити кілька виразів, то спочатку встановіть курсор на місце введення й натисніть кнопку **Add Line** стільки разів, скільки рядків буде містити цикл. Потім заповніть всі місця введення. Видаліть зайві місця введення (рис. 2.4).

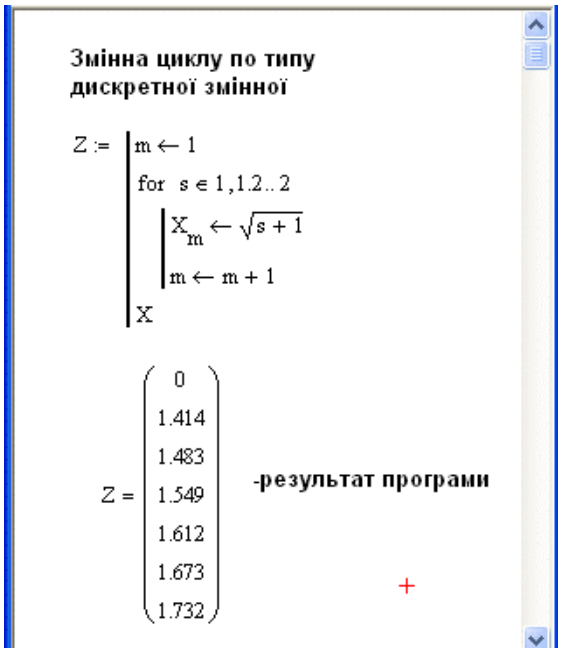


Рис. 2.4. Лічильний цикл

Оператори break, continue, return.

Ці оператори використовуються для керування роботою циклів і всієї програми в цілому:

- ***continue*** повертає розрахунок на початок циклу;
- ***break*** забезпечує вихід із циклу й продовження роботи програми;
- ***return*** забезпечує вихід із програми.

Робота цих операторів показана на рис. 2.5. Оператор ***continue*** у цьому прикладі створює список непарних чисел (вектор V), здійснюючи повернення на початок циклу, якщо залишок від ділення числа n на 2 (mod(n, 2)) дорівнює 0. Оператор ***break*** здійснює вихід із циклу й продовження роботи програми, якщо n < 0. Оператор ***return*** здійснює вихід із програми, якщо n = -10, із записом введеного коментаря (*вихід із програми*).

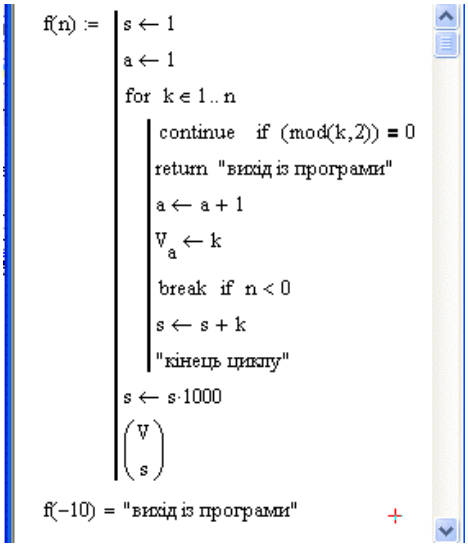


Рис. 2.5. Використання операторів циклу, continue, return, break

Примітка

Для вставки в програму операторів **continue, break**, **return**, як і інших операторів програмування, потрібно користуватися тільки панеллю програмування. Їх не можна набирати із клавіатури.

3. Використання конструкції if…elseif…else

Умовний оператор **if** складається з двох частин. Праворуч від ключового слова **if** записується логічний вираз (умова), який потрібно перевірити. Якщо він справджується, то виконується вираз ліворуч від оператора **if**. Якщо логічний вираз не виконується – нічого не відбувається, а виконання програми продовжується переходом до її наступного рядка. Вставити умовний оператор у програму можна в такий спосіб:

- введіть ліву частину виразу та оператор присвоєння;
- створіть наступний рядок програмного коду, натисканням на панелі **Programming** (Програмування) кнопки **Add Line** (Додати лінію);
- натисніть кнопку умовного оператора **if**;

- праворуч від умовного оператора if введіть умову. Користуйтесь логічними операторами, вводячи їх з панелі **Boolean** (Булеві оператори);
- вираз який повинен виконатись, якщо умова істина, введіть ліворуч від оператора **if**;

якщо в програмі передбачені додаткові умови, додайте в програму ще один рядок натисканням кнопки **Add Line** (Додати лінію*)* та введіть їх в такий же спосіб, використовуючи оператор **if** або **otherwise**.

Оператор **otherwise** використовується з одним або декількома операторами **if** і вказує на вираз, що буде виконуватись, якщо жодна з умов не виконується (рис. 2.6).

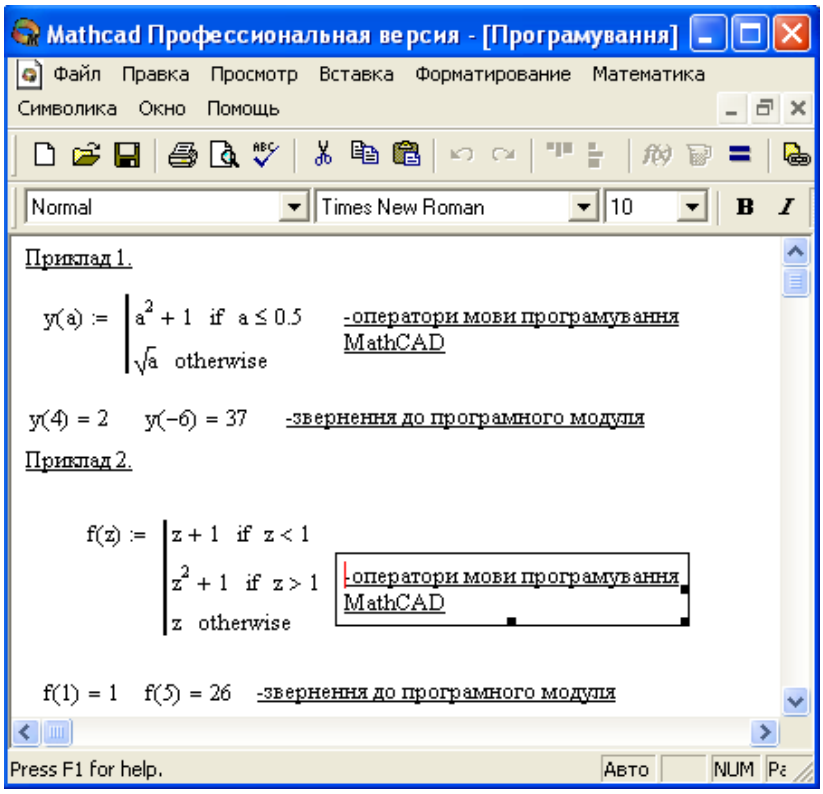


Рис. 2.6. Варіанти запису оператора if в MathCAD

Запитання для самоконтролю

1. Як виконується символьне інтегрування у Mathcad?
2. Як вирішуються диференціальні рівняння в середовищі Mathcad?
3. Як задати цикл у Mathcad?
4. Яка різниця між "while" і "for" циклами у Mathcad?
5. Як використовувати логічну конструкцію if у Mathcad?
6. Як працює комбінація if…elseif…else у логічних умовах?

7. Які типи даних використовуються в умовних виразах?
8. Як вивести результат на екран після виконання циклу?
9. Як перевірити істинність умови в Mathcad?
10. Як зупинити виконання обчислення при певній умові?

Лекція 3. Початок роботи з Matlab.

План

1. Арифметичні обчислення. Використання змінних.
2. Вектори та матриці. Введення, додавання і віднімання векторів. Введення матриць і найпростіші операції.
3. Візуалізація розрахунків. Робота з декількома графіками. Графічні об'єкти, редактор графіків. Двовимірні та трьохвимірні графіки

1. Арифметичні обчислення. Використання змінних

Пакет MatLab містить набір вбудованих функцій. Вбудовані математичні функції дозволяють знаходити значення різних виразів. MatLab надає можливість управління форматом виведення результату. Команди для обчислення виразів мають вигляд, властивий всім мовам програмування високого рівня. Наприклад, наберемо в командному рядку вираз:

```
>> 5+3
```

Натиснувши Enter отримаємо результат у командному рядку:

```
ans =
8
>>
```

Якщо при введенні у виразі використовуються десяткові дроби, то ціла частина відділяється від дробової *крапкою*.

Програма MatLab обчислює суму виразу, записує результат в спеціальну змінну *ans* і виводить її значення в командне вікно. Змінна *ans* створюється автоматично тоді, коли вираз, що обчислюється не присвоюється деякій змінній. Інформація про змінну *ans* відразу ж з'являється у вікні *Workspace* (рис. 4). У першому стовпці *Name* цього вікна записано ім'я змінної. Наступний стовпець *Value* показує значення змінної. Вміст стовпця *Size* демонструє основний принцип роботи MatLab. Програма MatLab всі дані представляє у вигляді масивів. Змінна *ans* є двовимірним масивом розміру один на один і займає 8 байт пам'яті, про що свідчить стовпець *Bytes*. В останньому стовпці *Class* зазначений тип змінної – *double array*, тобто масив, що складається з чисел подвійної точності. За замовчуванням всі числа представляються з подвійною точністю (*double*).

Після виводу результату у вікні *Command Window* наступним рядком після відповіді розташований командний рядок з символом >> і миготливим курсором. Це говорить про те, що навколишнє середовище MatLab готове до подальших обчислень.

Формати чисел

Вид, в якому виводиться результат обчислень, залежить від формату виведення, встановленого в MatLab. За замовчуванням MatLab видає числові результати в *нормалізованій формі* з чотирма цифрами після десяткової крапки. Але при роботі з числовими даними можна задавати різні *формати* представлення чисел. Однак в будь-якому випадку усі обчислення проводяться з граничною *подвійною* точністю. Для установки формату представлення чисел використовується команда >> format name – де *name* – ім'я формата.

Для числових даних *name* може мати наступні представлення: *short* – коротке представлення в фіксованому форматі (5 знаків), *short e* – коротке представлення в експоненційному форматі (5 знаків мантиси і 3 знаки порядку), *long* – довге подання у фіксованому форматі (15 знаків), *long e* – довге подання у експоненційному форматі (15 знаків мантиси і 3 знаки порядку), *hex* – подання чисел в шістнадцятирічному форматі, *bank* – представлення для грошових одиниць. Приклад: для ілюстрації різних форматів розглянемо вектор, що містить два елементи-числа:

```
>> x=[4/3   1.2345e-6]
```

Таблиця 3.1

Формати чисел

format short	1.3333	0.0000
format short e	1.3333e+000	1.2345e-006
format long	1.333333333333338	0.000001234500000
format long e	1.333333333333333e+000	1.234500000000000e-006
format bank	1.33	0.00

Задання формату позначається тільки на формі виведення чисел. Обчислення все одно відбуваються в форматі подвійної точності, а введення чисел можливе в будь-якому зручному для користувача вигляді. Необхідний формат виведення результату визначається користувачем з меню робочого середовища MATLAB. Виберемо в меню *File* пункт *Preferences*. На екрані з'явиться діалогове вікно *Preferences* (рис. 3.1).

Для установки формату виведення слід переконатися, що в списку лівої панелі діалогового вікна *Preferences* обраний пункт *Command Window*. Задання формату проводиться із списку *Numeric format* панелі *Text display* діалогового вікна. Натиснувши кнопку *OK* підтверджуємо установку вибраного формату і виконуємо закриття діалогового вікна.

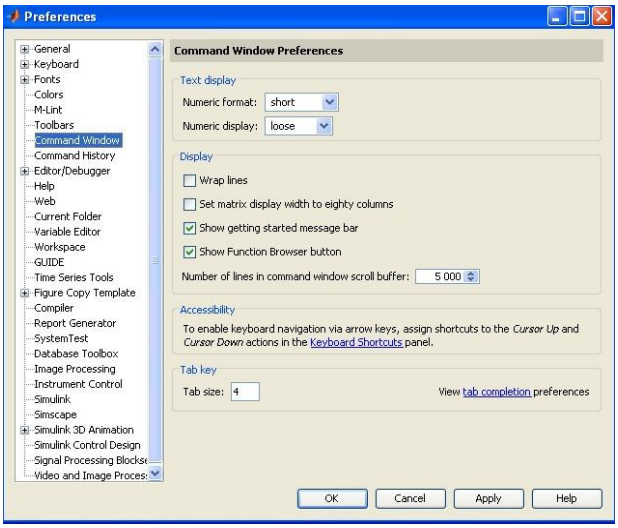


Рис. 3.1. Діалогове вікно *Preferences* MATLAB

Зараз встановлено короткий формат з плаваючою точкою *short* для виведення результатів обчислень, при якому на екрані відображаються тільки чотири цифри після десяткової крапки.

Наберемо в командному рядку 100/3 і натиснемо *Enter*. Результат виводиться у форматі *short*:

```
>> 100/3
ans =    33.3333
```

Цей формат виведення збережеться для всіх наступних обчислень. За потреби необхідно встановити інший формат.

Якщо потрібно отримати результат обчислень більш точно, то слід вибрати на панелі *Text display* із списку *Numeric Format* значення *long*. Результат буде відображатися в довгому форматі з плаваючою крапкою *long* з чотирнадцятьма цифрами після десяткової крапки.

Інформацію про формати можна отримати, набравши в командному рядку команду *help* з аргументом *format*:

```
>> help format
```

У командному вікні з'являється опис кожного з форматів. Задавати формат виведення можна безпосередньо з командного рядка за допомогою команди *format*. Наприклад, для установки довгого формату виведення результатів обчислень з плаваючою крапкою слід ввести команду *format long e* в командному рядку:

```
>> format long e
```

Команду слід вводити малими літерами. Це особливість вбудованої довідки *help*. MATLAB розрізняє великі та малі літери.

Для більш зручного сприйняття результату MATLAB виводить результат обчислень через рядок після виразу, що обчислюється. Однак іноді буває зручно розмістити більше рядків на екрані, для чого слід в діалоговому вікні *Preferences* вибрати *compact* із списку *Numeric display*. Додавання порожніх рядків забезпечується вибором *loose* із списку *Numeric display*.

Вбудовані елементарні функції

Вбудовані елементарні функції MatLab включають тригонометричні, гіперболічні, експоненціальні і логарифмічні функції, а також функції для роботи з комплексними числами і для округлення різними способами.

Таблиця 3.2

Основні математичні функції MatLab

sqrt(x)	обчислення квадратного кореня
exp(x)	піднесення до ступеня числа e
pow2(x)	піднесення до ступеня числа 2
log(x)	обчислення натурального логарифма
log10(x)	обчислення десяткового логарифма
log2(x)	обчислення логарифма за основою 2
sin(x)	синус кута x, заданого в радіанах
cos(x)	косинус кута x, заданого в радіанах
tan(x)	тангенс кута x, заданого в радіанах
cot(x)	котангенс кута x, заданого в радіанах
asin(x)	арксинус
acos(x)	арккосинус
atan(x)	арктангенс
acot(x)	арккотангенс
pi	число пі
round(x)	округлення до найближчого цілого
fix(x)	усічення дробової частини числа
floor(x)	округлення до меншого цілого
ceil(x)	округлення до більшого цілого
mod(x)	залишок від ділення с урахуванням знаку
sign(x)	знак числа
factor(x)	розкладання числа на прості множники
isprime(x)	істинно, якщо число просте
rand	генерація псевдовипадкового числа з рівномірним законом розподілу
randn	генерація псевдовипадкового числа з рівномірним законом розподілу
abs(x)	обчислення модуля числа

Приклади на округлення і залишок від ділення

- fix – округлення до найближчого цілого у напрямку до нуля:

```
>> fix(1.8)          >> fix(-1.8)
ans =               ans =
1                        -1
```

- floor, ceil – округлення до найближчого цілого у напрямку до мінус нескінченності або плюс нескінченності:

```
>> floor(3.2)          >> ceil(3.2)
ans =                  ans =
  3                      4
```

- round – округлення до найближчого цілого:

```
>> round(4.1)          >> round(4.5)
ans =                  ans =
  4                      5
```

- mod – залишок від цілочисельного ділення (зі знаком другого аргументу):

```
      >> mod(5,2)              >>
mod(5,-2) ans =                ans =
1                         -1
```

- rem – залишок від цілочисельного ділення (зі знаком першого аргументу):

```
>> rem(5,2)          >> rem(5,-2)
ans =                ans =
1                       1
```

Основним поняттям всіх математичних систем є *математичний вираз*. Він задає те, що повинно бути обчислено в числовому, рідше символьному вигляді. Приклади простих математичних виразів:

```
3 * 5
1.215 * sin(x) 5 + exp(4) / 3 sqrt(y) / 2
sin(pi/4)
```

Математичні вирази будуються на основі чисел, констант, змінних, операторів, функцій і різних спеціальних знаків.

Дійсні числа

Число являється найпростішим об’єктом мови MatLab, який представляє кількісні дані. Числа можна вважати константами, імена яких

співпадають з їх значеннями. Числа використовуються в загальноприйнятому представленні про них. Вони можуть бути цілими, дробовими, з фіксованою і плаваючою комою. Можливо представлення чисел в науковому форматі з вказанням мантиси і порядку числа. Приклади представлення чисел:

0
2 -3
2.301
0.00001
123.456e-24
-234.456e10

Як прийнято в більшості мов програмування в мантисі чисел ціла частина відокремлюється від дробової не комою, а крапкою. Для відокремлення порядку числа від мантиси використовується символ *e*. Знак «плюс» у чисел не проставляється, а знак «мінус» у числа називають *унарним мінусом*. Пробіли між символами в числах не допускаються.

В MatLab не прийнято ділити числа на цілі і дробові, короткі і довгі і т. д., як прийнято в більшості мов програмування, хоч задавати числа в таких формах можна. Операції над числами в MatLab виконуються у форматі, який прийнято вважати форматом з подвійною точністю.

Константи і системні змінні

Константа це попередньо визначене числове або символьне значення, представлене унікальним іменем. Наприклад, числа 1, -2, 1.23 являються безіменними числовими константами. Інші види констант в MatLab прийнято називати системними змінними, оскільки, з одного боку, вони задаються системою при її завантаженні, а з іншого можуть перевизначатися. Основні системні змінні, які застосовуються в системі MatLab:

- i або j уявна одиниця (корінь квадратний з -1);
- pi число π 3.1415926…;
- eps похибка операцій над числами з плаваючою крапкою (2^{-52});
- realmin найменше число з плаваючою крапкою (2^{-1022});
- realmax найбільше число з плаваючою крапкою (2^{1023});

2. Вектори та матриці. Введення, додавання і віднімання векторів. Введення матриць і найпростіші операції

MatLab – система, спеціально призначена для проведення складних обчислень з векторами, матрицями і масивами. При цьому вона за замовчуванням передбачає, що кожна задана змінна – це вектор, матриця або масив. Все визначається конкретним значенням змінної. Наприклад, якщо задано x=1, то це означає, що x – це вектор з одним елементом, який має значення 1. Якщо потрібно задати вектор із трьох елементів, то їх значення необхідно перелічити в квадратних скобках, розділюючи пробілами. Наприклад, присвоєння:

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

задає вектор *v*, який має три елемента із значеннями 1, 2 і 3. Після введення вектора система виводить його на екран дисплею.

Використання оператора двокрапка (:)

Часто виникає необхідність здійснити формування упорядкованих числових послідовностей. Такі послідовності потрібні для створення векторів або значень абсциси при побудові графіків. Для цього в MatLab використовується оператор : (двокрапка). Синтаксис оператора:

Початкове_значення : Крок : Кінцеве_значення

Дана конструкція формує зростаючу послідовність чисел, яка починається з початкового значення, змінюється із заданим кроком і завершується кінцевим значенням. Якщо Крок не задано, то він приймає значення 1. Якщо кінцеве значення вказано меншим, ніж початкове значення, то видається повідомлення про помилку. Розглянемо приклади застосування оператора.

```
>> 1:5 ans =
   1   2   3   4   5
>>i=0:2:10 i =
   0   2   4   6   8   10
```

Введення, додавання і віднімання векторів

Для збереження векторів використовуються масиви *a* і *b*. В командному рядку введемо масив *a*, використовуючи квадратні дужки і розділяючи елементи вектор-стовбця крапкою з комою. Крапка з комою в

кінці виразу використовується для скасування виведення результату на екран.

```
>> a = [1.3; 5.4; 6.9]
a =  1.3000
   5.4000
   6.9000
```

Так як введений вираз не завершено крапкою з комою, то MATLAB автоматично виведе значення змінної *a*. Тепер в командному рядку введемо другий вектор – масив *b* скасувавши виведення на екран.

```
>> b = [7.1; 3.5; 8.2];
```

Оскільки введений вираз завершено крапкою з комою, то виведення на екран значень змінної *b* скасовується. Обчислимо суму векторів *a* і *b*:

```
>> c = a+b
c =  8.4000
8.9000
15.1000
```

При складанні алгебраїчних виразів MatLab дозволяє використання традиційних знаків арифметичних операцій і символів спеціальних відносин (табл. 3.3).

Таблиця 3.3

Знаків арифметичних операцій і символів

Символи	Дія, що виконується
Операції над числовими величинами	
+	Покомпонентне додавання числових масивів однакової розмірності; додавання скалярної величини до кожного елементу масиву
-	Покомпонентне віднімання числових масивів однакової розмірності; віднімання скалярної величини з кожного елемента масиву
*	Множення матриць відповідно до правил лінійної алгебри (число стовпців першого співмножника має дорівнювати числу рядків другого співмножника); множення всіх компонентів масиву на скаляр
.*	Покомпонентне множення елементів масиву однакової розмірності

/	Ділення скаляра на скаляр; покомпонентне ділення всіх елементів масиву на скаляр
./	Покомпонентне ділення елементів масивів однакової розмірності
\	Ліве матричне ділення
.\	Покомпонентне ділення елементів (ліве поелементне ділення)
^	Зведення скаляра в будь-яку ступінь; обчислення цілого ступеня квадратної матриці
‘	Обчислення сполученої матриці
.‘	Транспонування матриці

Поелементні операції з векторами

В розглянутому раніше матеріалі вектор використовувався в якості аргументу математичних функцій, результатом яких був вектор з елементами, які дорівнюють значенням функції від відповідних елементів вхідного вектора. Таким чином, відбувалося *поелементне* обчислення функції яка викликається. Далі розглянемо можливості поелементної роботи з векторами, які знадобляться в подальшому для визначення власних функцій і побудови їх графіків.

Поелементні операції з векторами:

Введемо два вектор-рядки

```
>> v1 = [2 -3 4 1];
>> v2 = [7 5 -6 9];
```

Операція .* (перед знаком множення повинна бути крапка) приводить до поелементного множення векторів однакової довжини. В результаті отримуємо вектор з елементами, які дорівнюють добутку відповідних елементів вхідних векторів:

```
>> u = v1.*v2
u =  14   -15   -24     9
```

За допомогою .^ здійснюється поелементне піднесення до степеню:

```
>> p = v1.^2
p =     4     9    16     1
```

Показником степеня може бути вектор тієї ж довжини, що і зведений до степеня. При цьому кожен елемент першого вектора підноситься до степеню, який дорівнює відповідному елементу другого вектора:

```
>> p = v1.^v2
P =     128.0000  -243.0000  0.0002  1.0000
```

Ділення відповідних елементів векторів однакової довжини виконується з використанням ./ >> d = v1./v2

```
d =    0.2857   -0.6000   -0.6667    0.1111
```

Зворотне поелементне ділення (ділення елементів 2-го вектора на відповідні елементи 1-го) здійснюється за допомогою .\

```
>> dinv = v1.\v2 dinv =    3.5000   -1.6667   -1.5000    9.0000
```

Таким чином, крапка в MATLAB використовується не тільки для введення десяткових дробів, але і для вказівки на те, що ділення або множення або зведення до степеню масивів однакового розміру повинно бути виконано поелементно.

До поелементних операцій відносяться і операції з вектором і числом. Складання вектора і числа не приводить до повідомлення про помилку. MATLAB прибавляє число до кожного елемента вектора. Таке ж твердження відноситься і для віднімання. Додавання числа до кожного елемента вектора:

```
>> v = [4 6 8 10]; >> s = v+1.2 s =    5.2000    7.2000    9.2000   11.2000
```

Додавання до кожного елемента вектора числа:

```
>> v = [4 6 8 10]; >> s1 = 1.2+v s1 =    5.2000    7.2000    9.2000   11.2000
```

Віднімання числа з кожного елемента вектора

```
>> v = [4 6 8 10];
>> r1 = v-1.2
r1 =    2.8000    4.8000    6.8000    8.8000
```

Віднімання із числа кожного елемента вектора

```
>> r2 = 1.2-v r2 =   -2.8000   -4.8000   -6.8000   -0.8000
```

Множити вектор на число можна як справа,
так і зліва:

```
>> v = [4 6 8 10]; >> p1 = v*2
p1 =     8    12    16    20
>> p2 = 2*v
p2 =     8    12    16    20
```

Ділити за допомогою знака / можна вектор на число:

```
>> d = v/2
d =     2     3     4     5
```

Векторний добуток

Векторний добуток a × b визначений тільки для векторів із трьохвимірного простору, тобто які складаються із трьох елементів. Результатом також являється вектор із трьохвимірного простору. Для обчислення векторного добутку в MATLAB існує функція *cross*:

```
>> a = [1.2; -3.2; 0.7];
>> b = [4.1; 6.5; -2.9];
>> c = cross(a, b)
c =   4.7300
   6.3500
   20.9200
```

3. Візуалізація розрахунків. Робота з декількома графіками. Графічні об'єкти, редактор графіків. Двовимірні та трьохвимірні графіки

Функції однієї змінної $y(x)$ широко застосовуються в практиці математичних і інших розрахунків, а також у техніці комп'ютерного математичного моделювання. Для відображення функцій однієї змінної $y(x)$ використовуються графіки в декатровій (прямокутній) системі координат. При цьому будуються дві осі – горизонтальна X і вертикальна Y, і задаються координати x і y, які визначають вузлові точки функції $y(x)$. Ці точки об'єднуються одна з одною відрізками прямих, тобто при побудові графіку здійснюється лінійна інтерполяція для проміжних точок. Оскільки MATLAB – матрична система, сукупність точок $y(x)$ задається векторами X і Y однакового розміру.

Команда ***plot*** (X, Y) служить для побудови графіків функцій в декартовій системі координат. Синтаксис команди:

plot (X,Y) – будує графік функції $y(x)$, кординати точок (x, y) якої беруться із векторів однакового розміру Y і X. Якщо X або Y – матриця, то будується сімейство графіків за даними, які містяться в колонках матриці.

Розглянемо приклад, який ілюструє побудову графіків двох функцій sin (x) і cos (x), значення яких містяться в матриці Y, а значення аргументу x зберігається у векторі x (рис. 3.2, рис. 3.3). В робочому вікні створимо вектор x, обчислимо значення функції sin (x), введемо команду *plot* для побудови графіка:

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

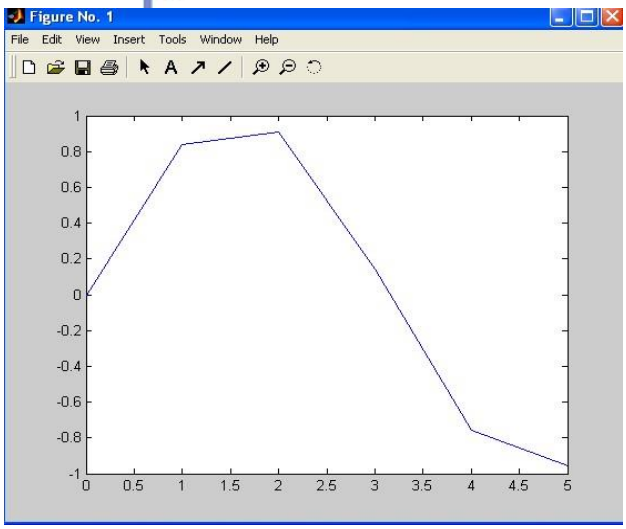


Рис. 3.2. Графік функції sin (*x*)

Тепер створимо вектор *x* і обчислимо значення функції cos (*x*), введемо команду *plot* для побудови графіка:

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

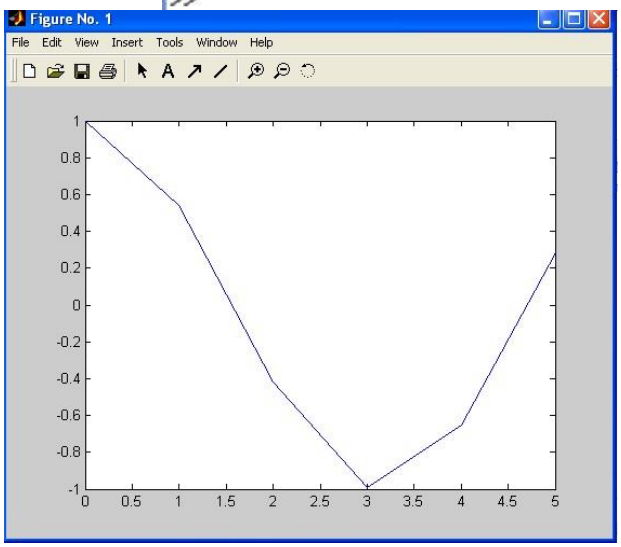


Рис. 3.3. Графік функції cos (*x*)

Далі графіки функцій sin (*x*) і cos (*x*) разташуємо в одному графічному вікні. Виклик функції *plot* з числовими парами *x* і *y* створює числові графіки. При відсутності вказівки на колір ліній і точок MATLAB автоматично

присвоює кожному графіку свій колір, тип, що дозволяє розрізняти задані набори даних (рис. 3.4).

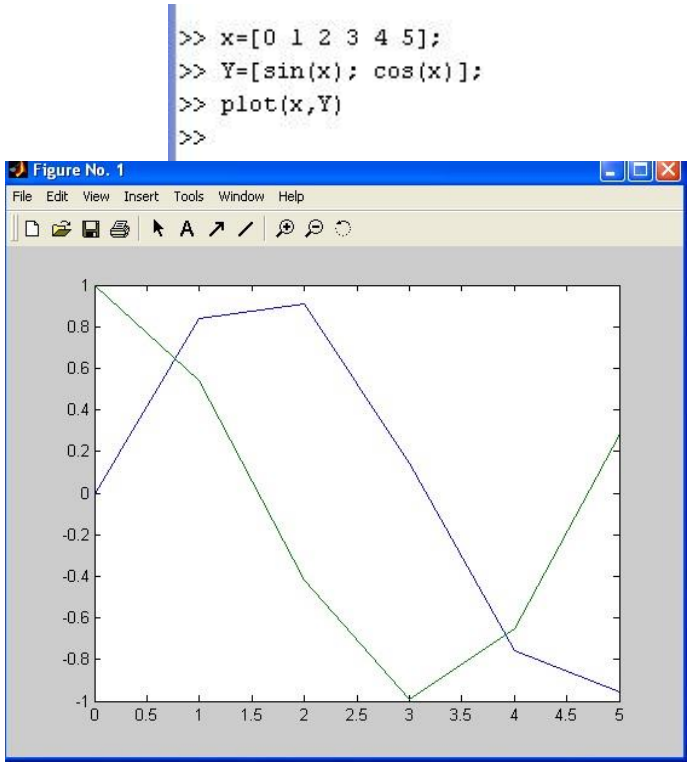


Рис. 3.4. Графіки функцій sin (*x*) і cos (*x*)

Для того, щоб відобразити графіки функцій в одному графічному вікні і кожен графік щоб мав свій колір необхідно вказати це в команді ***plot.***

Синтаксис: plot (X,Y,S). Тип лінії графіка можна задавати з допомогою строкової константи S (табл. 3.4).

Таблиця 3.4

Тип лінії графіка

Тип лінії	символ	Тип маркера	символ	Колір лінії	символ
суцільна	-	плюс	+	блакитний	c
розривна штрихова	–	кружок	o	малиновий	t
пунктирна	:	зірочка	*	жовтий	y
штрихпунктирна	-.	хрест	x	червоний	r
відсутність лінії	none	крапка	.	зелений	g

		квадрат	s	синій	b
		ромб	d	білий	w
		п'ятигранник	p	чорний	k
		шестигранник	h	фіолетовий	m
		стрілка вниз	v		
		стрілка вверх	^		
		стрілка вліво	<		
		стрілка вправо	>		

Таким чином, за допомогою строкової константи S можна змінювати колір лінії, представляти вузлові точки різними відмітками (крапка, коло, хрест, трикутник з різною орієнтацією вершин) и змінювати тип лінії графіка.

Розглянемо приклад побудови графіків трьох функцій з різним стилем представлення кажної із них (рис. 3.5):

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

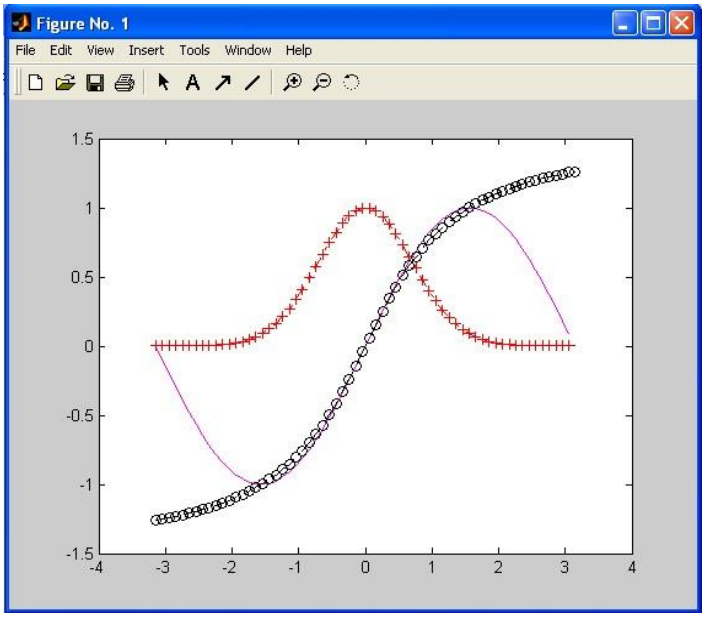


Рис. 3.5. Графіки трьох функцій

Графік функції *y*1 будується суцільною фіолетовою лінією, графік *y*2 будується штрих пунктирною лінією з точками у вигляді знака «плюс» червоного кольору, а графік *y*3 будується штриховою лінією з кружками чорного кольору.

Оформлення графіків

Додаткові прикраси графіка полягають у можливості забезпечити його заголовком, підписати осі, нанести координатну сітку і розмістити легенду:

title – виводить заголовок рафіка

xlabel, ylabel – підписує осі графіка

grid on – наносить координатну сітку

legend – розміщує легенду

Синтаксис legend (опис 1-го графіка, опис 2-го графіка, місце розміщення легенди в графічному вікні). Параметр місце розміщення легенди в графічному вікні може приймати наступні значення:

-1 – легенда розміщується поза полем графіка, вгорі праворуч;

0 – система вибирає краще місце в полі графіка, що не перекривається даними;

1 – легенда розміщується в правому верхньому куті (за замовчуванням – там же);

2 – легенда розміщується в лівому верхньому кутку поля графіка;

3 – легенда розміщується в лівому нижньому кутку поля графіка;

4 – легенда розміщується в правому нижньому кутку поля графіка.

Приклад оформлення графіка підписами (рис. 3.6):

```
>> x=0:0.2:6.28;
>> y1=sin(x);
>> y2=cos(x);
>> plot(x,y1,'-',x,y2,'*')
>> legend('sin','cos',4)
>> ylabel('y-axis')
>> xlabel('x-axis')
>> grid on
>> title('Function sin & cos')
>>
```

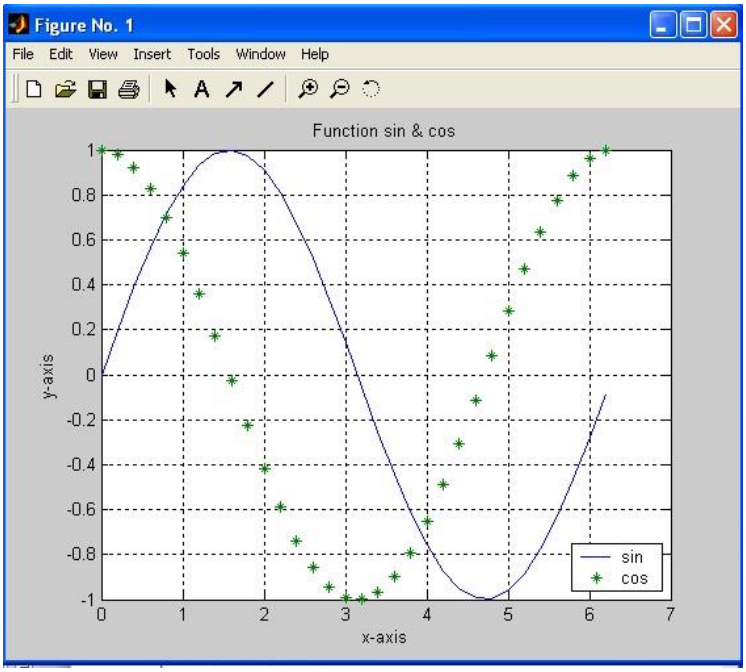


Рис. 3.6. Графік, оформлений підписами

Працюючи з графічними об'єктами виникають ситуації, коли необхідно розміщувати кожен з графіків в окреме графічне вікно. Команди функцій побудови графіків автоматично відкривають нове вікно зображення, якщо до цього його не було на екрані. Якщо ж воно існує, то команда використовує його за замовчуванням. Для відкриття нового вікна і вибору його за замовчуванням, необхідно набрати figure. Для того, щоб зробити існуюче вікно поточним треба в дужках вказати номер вікна: figure (n) де n – це номер в заголовку вікна. У цьому випадку результати всіх наступних команд будуть виводитися в це вікно. Якщо треба зберегти кілька графіків, то перед виведенням нового графіка необхідно відкрити нове вікно з наступним номером вікна, який вказується в дужках.

Для побудови циліндра у вигляді тривимірної фігури застосовується функція cylinder:

[X, Y, Z] = cylinder(R, N), яка створює масиви X, Y і Z, які описують циліндричну поверхню з радіусом R і числом вузлових точок N для подальшої побудови за допомогою функції surf(X,Y,Z).

Для побудови графіка використовується функція surf, яка будує каркасну поверхню і заливає кожну клітину поверхні певним кольором. Для побудови каркасної поверхні також можна використовувати функцію mesh.

Приклад побудови об’ємного циліндра (рис. 3.7).

```
>> [X,Y,Z]=cylinder(10,30);
>> surf(X,Y,Z,X)
```

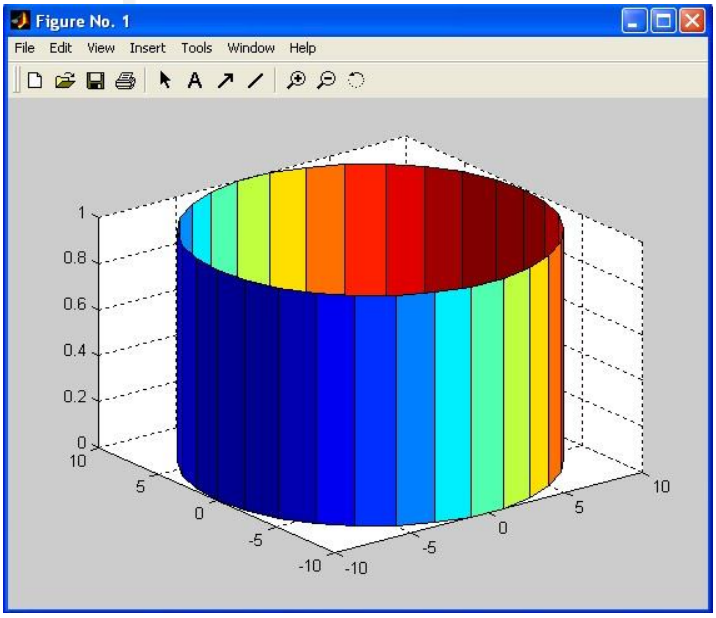


Рис. 3.7. Графік об’ємного циліндра

Забарвлення кольором визначено вектором X. Вид циліндра істотно залежить від графічної команди, що використовується для його побудови. Команда surf дає можливість задати функціональне забарвлення з кольором, який визначається вектором X, який задається четвертим параметром і робить уявлення циліндра досить наочним.

У команді surf змінимо функціональне забарвлення кольором, який визначимо вектором Y, який задаємо четвертим параметром (рис. 3.8):

```
>> [X,Y,Z]=cylinder(10,30);
>> surf(X,Y,Z,Y)
```

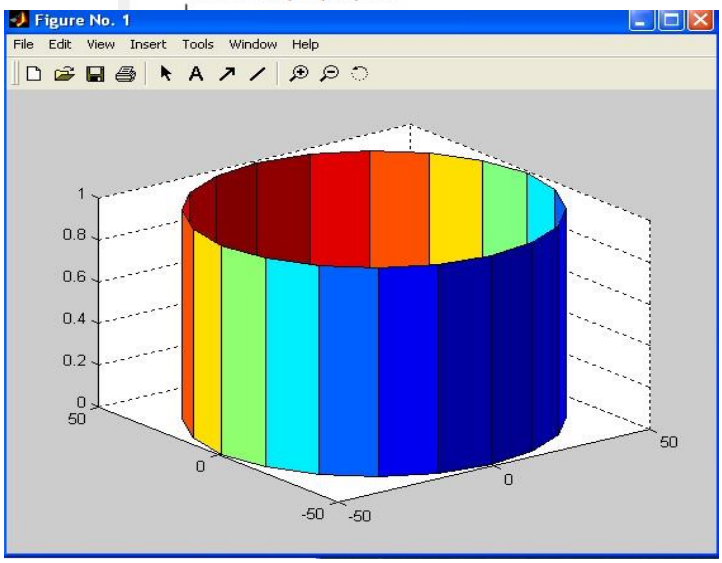


Рис. 3.8. Графік об’ємного циліндра

Забарвлення кольором визначено вектором Y. Потім у команді surf змінимо функціональне забарвлення кольором, який визначимо вектором Z, який задаємо четвертим параметром (рис. 3.9).

```
>> [X,Y,Z]=cylinder(10,30);
>> surf(X,Y,Z,Z)
```

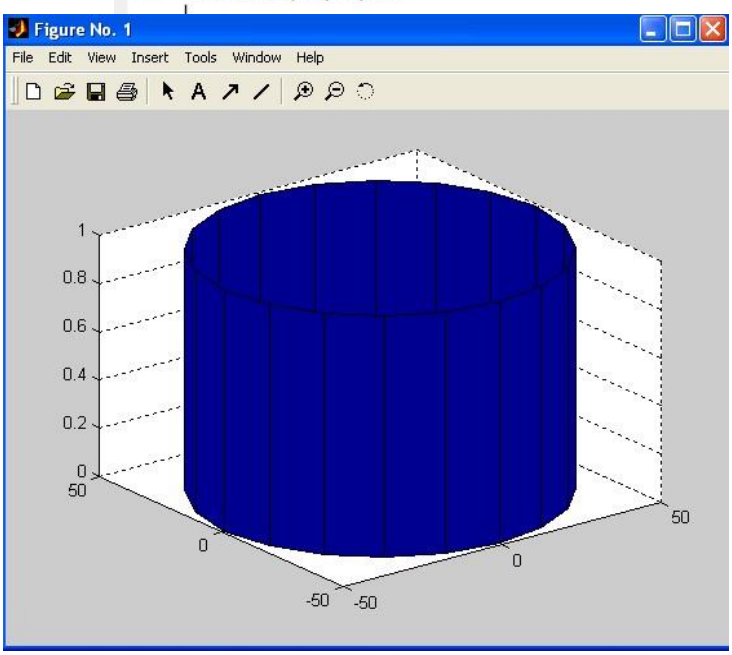


Рис. 3.9. Графік об’ємного циліндра

Забарвлення кольором визначено вектором Z

Високорівнева графіка. Побудова сфери як трьохмірної фігури

Для формування масивів X, Y та Z - координат точок сфери, як тривимірної фігури використовується функція sphere. Команда mesh (X, Y, Z, C) виводить на екран сітчасту поверхню значень масиву Z, визначених на безлічі значень масивів X і Y. Кольори вузлів поверхні задаються масивом C. [X,Y,Z] = sphere(N) – генерує матриці X, Y та Z розміру (N+1)*(N+1) для побудови сфери за допомогою команд surf (X,Y,Z) або surfl (X, Y, Z) чи mesh (X, Y, Z). Розглянемо приклад застосування функції sphere при N = 20, графік виконаний за допомогою surf (рис. 3.10):

```
>> [X, Y, Z] = sphere (30);
>> surf (X, Y, Z)
```

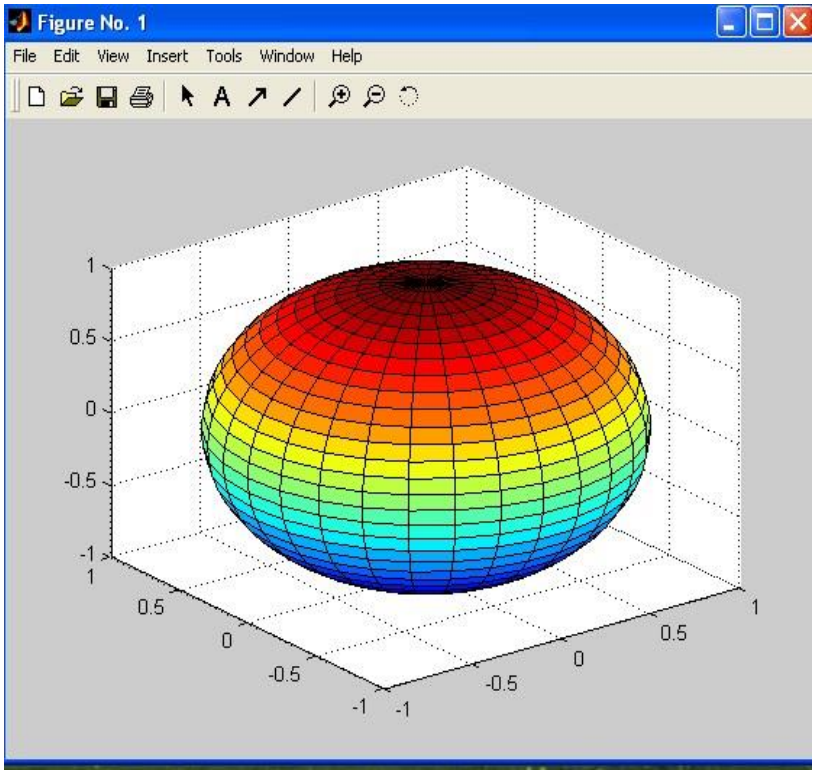


Рис.3.10. Графік сфери побудований за допомогою функції surf

Питання для самоконтролю

1. Як задати змінну у Matlab?
2. Які основні правила іменування змінних у Matlab?
3. Як ввести вектор у Matlab?
4. Як виконати додавання і віднімання векторів?
5. Як створити матрицю у Matlab?
6. Які операції підтримуються для матриць?
7. Як побудувати графік функції у Matlab?
8. Як додати кілька графіків на один малюнок?
9. Як побудувати 3D-графік у Matlab?
10. Як користуватися редактором графіків у Matlab?

Лекція 4. Функції та цикли в Matlab

План

1. Скрипти та функції
2. Функції sprint та printf
3. Конструкція if…elseif…else
4. Цикли for і while

1. Скрипти та функції

Поняття м-файлу

У систему MATLAB включена мова програмування високого рівня, за допомогою якої можна розв'язувати задачі векторної алгебри, лінійної алгебри і аналітичної геометрії за допомогою матричного запису, а також як автоматизувати, так і унаочнити процес дослідження впливу параметрів, що входять в рівняння, на положення і форму лінії і поверхні в координатному просторі.

Система MATLAB орієнтована на роботу в напівавтоматичному режимі, тобто необхідно в командному рядку ввести команду і передати її до ядра системи. Після перевірки синтаксису команда автоматично опрацьовується в ядрі і результат виводиться в командне і/або графічне вікно, після чого система переходить в режим очікування введення нової команди. Якщо необхідно переглянути результат виконання команди з іншими початковими даними, слід заново ввести в командний рядок всі необхідні команди, оскільки в системі MATLAB не передбачено можливості динамічного оновлення результату при зміні значення початкової змінної. Існує дві можливості автоматизації повторного введення серії команд. Перший спосіб полягає у використанні вікна Command History, в якому зберігаються введені раніше команди. Цей спосіб зручно використовувати для повторного виконання невеликої кількості команд. Проте в деяких випадках, залежно від поточних значень змінних, потрібно використовувати різні набори команд невідоме число разів. У таких випадках слід скористатися другим способом, заснованим на застосуванні *m*-файлів, де можуть міститися команди і управляючі структури мови MATLAB. Звертання до створеного *m*-файлу здійснюється за допомогою вказування (введення) його імені, а якщо необхідно заданням вхідних і вихідних параметрів в командному рядку.

М-файли являють собою текстові файли, що збережені з розширенням m. Оскільки m-файл є текстовим файлом, то його можна створювати за допомогою будь-якого текстового редактора. До складу системи MATLAB входить редактор Editor/Debugger, за допомогою якого можна як створювати m-файли, так і виконувати їх налагодження, що приводить до скорочення часу створення робочого mфайлу. Відкрити редактор Editor/Debugger для створення нового m-файлу можна за допомогою команди M-file, розташованої в підменю New головного меню File, або за допомогою "натиснення" на кнопку New на панелі інструментів робочого столу системи MATLAB.

М-файли підрозділяються на два типи: сценарії (script) і функції (function).

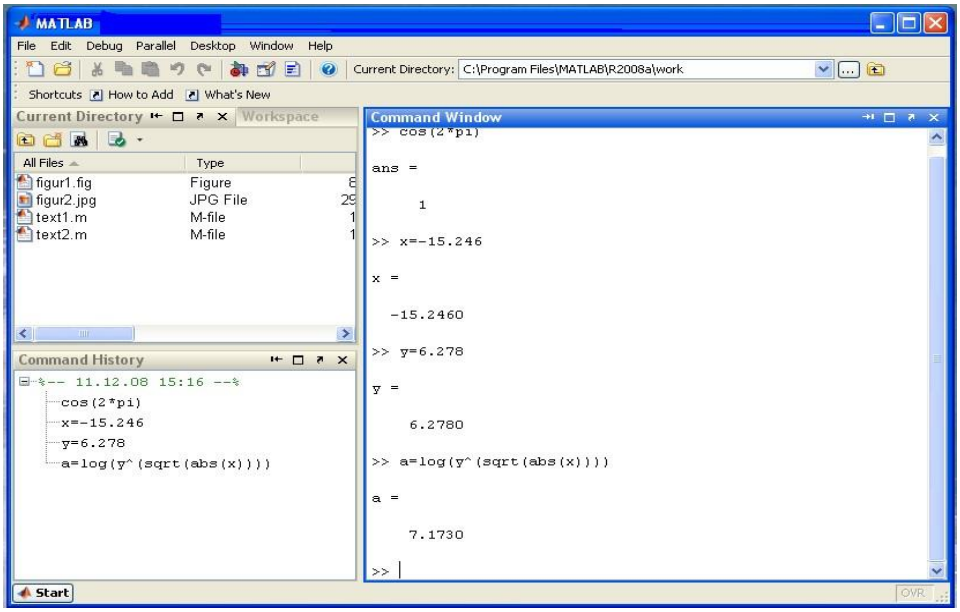


Рис. 4.1. Файл скрипт

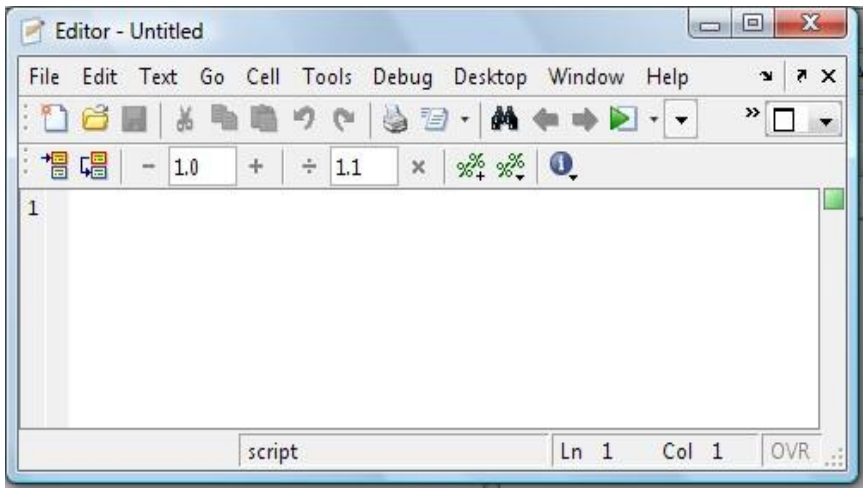


Рис. 4.2. Файл-сценарій

Файли-сценарії

Сценарії, що є найпростішим типом m-файлів, містять послідовності команд, які задаються в командному рядку, і коментарі, що починаються із знаку %. Особливостями сценаріїв є виконання команд в режимі інтерпретації, тобто команди перетворюються у код програми і виконуються порядково, використовуючи тільки змінні, розташовані в робочому просторі MATLAB.

Отже, необхідно бути уважним при створенні змінних під час виконання сценарію, оскільки нові змінні, імена яких співпадають з іменами змінних, розташованих в робочому полі, замінять їх, що може призвести до втрати необхідних для подальшої роботи даних.

Головною особливістю описування програм мовою MATLAB є те, що типи змінних на початку програми не декларуються, досить надати змінній значення певного типу.

Файл-функції

Функції разом з сценаріями також містять команди, але є складнішим типом m-файлів в порівнянні з сценаріями. Відмінними особливостями функцій від сценаріїв є:

можливість компіляції всієї функції у код програми з подальшим розміщенням його в пам’яті;

наявність власного робочого простору, де зберігаються локальні змінні;

наявність вхідних і вихідних параметрів.

Після виконання функції її робочий простір вилучається з пам’яті. Отже, значення всіх змінних, які були створені під час виконання функції і розташовувалися в її робочому просторі, вилучаються з пам’яті.

Приклад. Обчислити кути нахилу вектора *a = (-1;2;5)* до осей координат. Обчислення оформити у вигляді функції з ім'ям angles. Для цього необхідно звернутися до редактора *m*-файлів та ввести наступний набір команд:

```
function a=angles(A)
%Обчислення  кута нахилу вектора
A=acos(A./sqrt(sum(A.*A)))*180/pi
```

Тут перший рядок є заголовком функції. Він включає ім’я функції (в даному випадку angles), а також один вхідний (*A*) і один вихідний (*a*) параметри.

Наступний рядок з коментарем, який при обчисленні функції ігнорується.

Це необов'язкова частина файл-функції. Проте якщо виникне потреба отримати довідку про функцію, коментар допоможе пригадати, яке призначення даної функції:

```
>> help angles
%Обчислення  кута нахилу вектора
```

Таким чином, коментар, введений після заголовка функції, інтерпретується як опис функції. Наступний рядок - тіло функції - вираз, за яким обчислюється значення функції.

Після введення файл-функцію необхідно зберегти в поточному робочому каталозі, аналогічно до збереження файл-сценарію, але ім'я m-файлу, в якому зберігається файл-функція, обов'язково повинне співпадати з іменем функції.

Створену файл-функцію можна використовувати як в командному режимі (аналогічно до будь-якої вбудованої функції системи MATLAB), так і викликати з інших файл-програм або файл-функцій.

При виклику файл-функції потрібно вказати всі її вхідні і вихідні параметри.

Результати виконання функції angles () :

```
>> A=[-1 2 5];
>> angles(A)
A =
  100.5197   68.5833   24.0948
>> angles ([-1 2 5])
A =
  100.5197   68.5833   24.0948
```

2. Функції sprint та printf

Функція sprintf: форматне виведення у рядок

```
str = sprintf(fmt, arg1, arg2, ….);
```

Функція fprintf: форматне виведення на екран або в файл

```
fprintf(fmt, arg1, arg2, ….); % На екран
fprintf(fd,fmt, arg1, arg2, ...); % В файл
```

fd– дескриптор файлу

fmt- Рядок з описом формату виведення. Послідовності, що починаються на %, замінюються вхідними аргументами.

Таблиця 4.1

Параметри функції sprint та printf

Формат	Значення	Формат	Значення
%d	Ціле	**%f**	Десятковий дріб
%.10d	Ціле, доповнене до 10 знаків нулями зліва	**%.3f**	Десятковий дріб із 3 знаками після коми
%10d	Ціле, доповнене до 10 знаків пробілами ліворуч	**%10.3f**	Те саме, але доповнена до 10 знаків пробілами зліва
%s	Рядок	**%e**	Експонентний запис
\n	Новий рядок	**%15.3e**	Те саме, але з 3 знаками після коми та доповнена до 15 знаків пробілами зліва

Приклади використання зі скалярами та рядками:

```
>>fprintf('%d\n\n', 10);
10
>>fprintf('%10.3f, %5.1f\n',
2.1, 3.5);
        2.100, 3.5
>>fprintf('%15.3e\n', 123456);
        1.235e+05
>>fprintf('%s %e %f %d\n', 'ab', 1, 2, 3, 4);
ab1.000000e+00 2.000000 3
>>sprintf('%.3e', 1.3)
ans= 1.300e+00
>> fprintf(%X %o, 255, 63)
FF 77
```

Приклади використання з матрицями:

```
>>fprintf('%d\n\n', 10);
10
>>sprintf('<%.3f>',rand(1,4));
ans=<0.416><0.568><0.225><0.467>
>>sprintf('%d', [1 2 3; 4 5 6]);
ans = 142 536
>>fprintf('%d %d %d\n',
 [1 2 3; 4 5 6]')
1 2 3
4 5 6
```

3. Конструкція if…elseif…else

Оператор *if* може застосовуватися в простому вигляді, для виконання блоку команд при задоволенні деякої умови, або в конструкції *if-elseif-else* для написання алгоритмів, що розгалужуються. Синтаксис умовного оператора виглядає так:

```
if умова
команди MatLab
end
```

Якщо умова вірна, то виконуються команди MatLab, розміщені між *if* і *end*, а якщо умова невірна, то відбувається перехід до команд, розташованим після *end*.

У загальному вигляді оператор розгалуження представляє конструкцію *ifelseif-else*. Розглянемо на прикладі лістинг програми, яка демонструє роботу *ifelse*. Залежно від виконання тієї чи іншої умови працює відповідна гілка програми. Якщо всі умови невірні, то виконуються команди, розміщені після *else*. Гілок може бути скільки завгодно або тільки дві. Розглянемо приклад розгалуження, наведений на рис. 4.3.

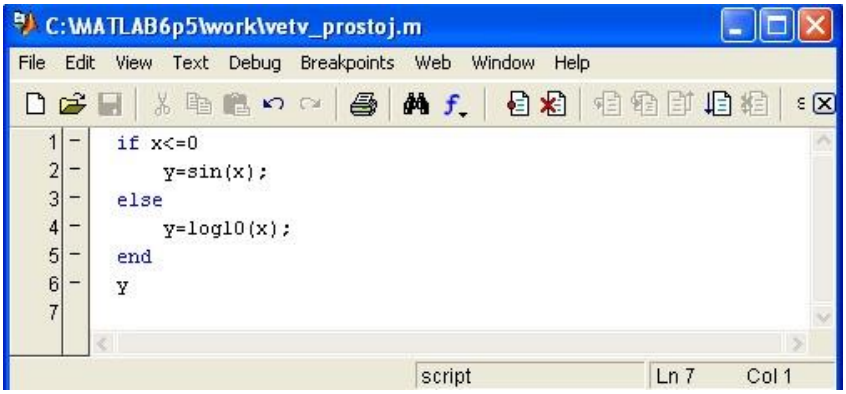


Рис. 4.3. Розгалужений обчислювальний процес

Наведений приклад має дві гілки розгалуження, тому для перевірки правильності написання програми необхідно почергово задавати змінній *x* значення, які відповідають умовам, заданим у прикладі. Результат розв’язання даного прикладу представлено на рис. 4.4.

У разі наявності двох гілок використовується завершальне *else*, а *elseif* пропускається. Оператор *if* повинен закінчуватися *end*. При наявності у прикладі трьох гілок, використовується конструкція *if-elseif-else*.

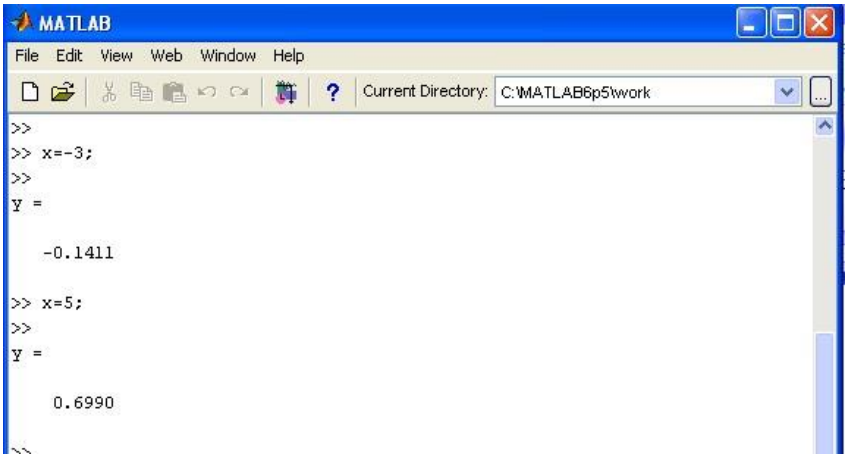


Рис. 4.4. Результат розв'язання прикладу,
отриманий при значенні x=-3 та при значенні x=5

4. Цикли for і while

Схожі і повторювані дії виконуються за допомогою операторів циклу *for* і *while*. Цикл *for* призначений для виконання повторюваних дій задану кількість разів – арифметичний цикл, а *while* - для дій, число яких заздалегідь не відомо, але відома *умова* продовження циклу.

Цикл for

Використання *for* здійснюється наступним чином:

```
for count = start : final
команди MatLab
end
```

Вираз *count* – це змінна циклу, *start* - початкове значення змінної циклу, *final* - кінцеве значення змінної циклу, а *step* - крок, на який збільшується *count* при кожному наступному заході в цикл. Цикл закінчується, як тільки значення *count* стає більше за *final*. Змінна циклу може приймати не тільки цілі, але і дійсні значення будь-якого знаку.

Приклад застосування циклу *for*. Нехай потрібно вивести графіки сімейства кривих для x що належать інтервалу [0,2□]. Сімейство кривих задано функцією $y(x, a) = e^{-ax} sinx$, що залежить від параметра a, для значень параметра a від -0,1 до 0,1 з кроком 0,02. Можна, звичайно, послідовно обчислювати $y(x,a)$ і будувати її графіки для різних значень a від -0,1 до 0,1. Але набагато зручніше використовувати цикл *for*.

Наберемо текст файл-програми в редакторі М-файлів. Для цього необхідно розкрити меню File робочого середовища MatLab, в пункті New вибрати підпункт M-file (рис. 4.5).

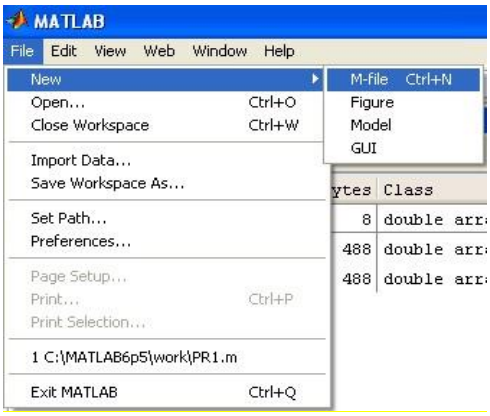


Рис. 4.5. Команди виклику вікна Untitled для набору програми

Після цього відкриється вікно редактора Мфайлів Untitled, в якому і виконується набір програми. Відкрити вікно Untitled для набору програми можна також натиснувши кнопку New M-file на панелі інструментів робочого середовища.

Новий файл для набору рядків програми відкривається у вікні Untitled редактора М-файлів, яке наведене на рис. 4.6.

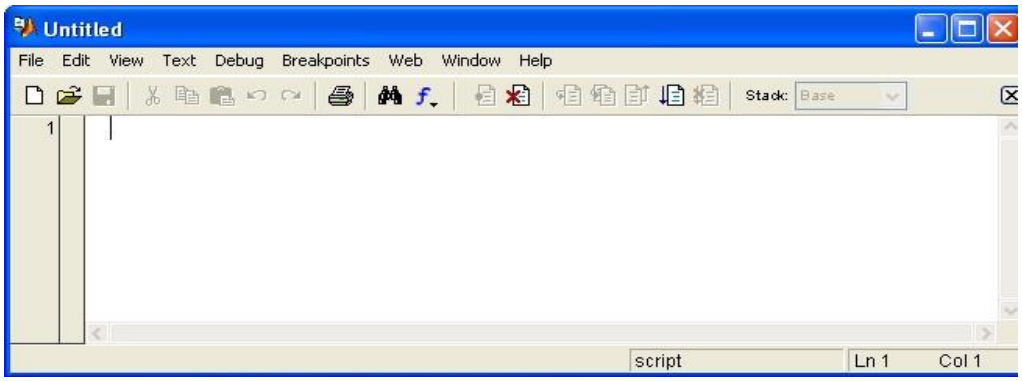


Рис. 4.6. Вікно Untitled редактора М-файлів MatLab

Наберемо у редакторі команди програми (рис. 4.7).

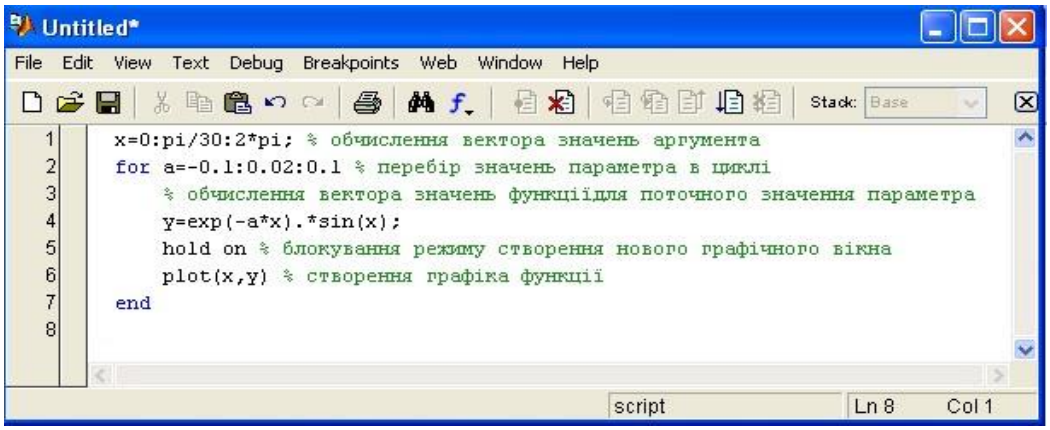


Рис. 4.7. Набір команд програми у вікні редактора

Набравши рядки програми необхідно зберегти файл з ім'ям PR1, вибравши в меню File редактора пункт Save as (рис. 4.8.).

Рис. 4.8. Вікно збереження файлу

На екрані з'явиться вікно для збереження файлу, в якому в рядку Ім’я файла необхідно задати ім'я файлу, наприклад PR1 (рис. 4.9).

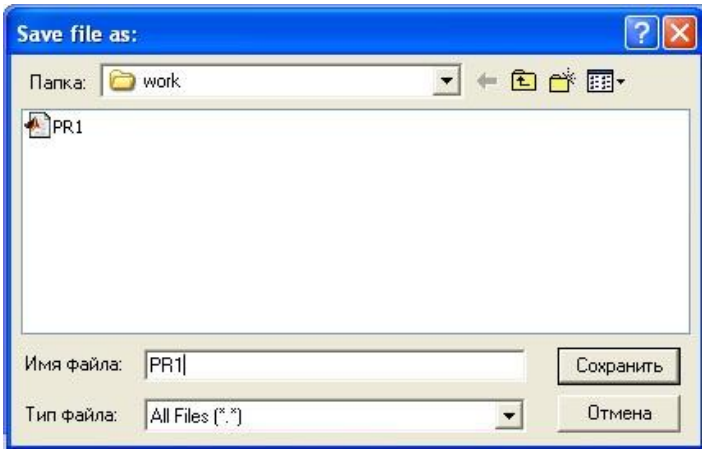


Рис. 4.9. Вікно для збереження файлу

Для запуску на виконання команд файлу слід вибрати пункт Run в меню Debug або просто натиснути кнопку F5, розташовану на клавіатурі (рис. 4.10).

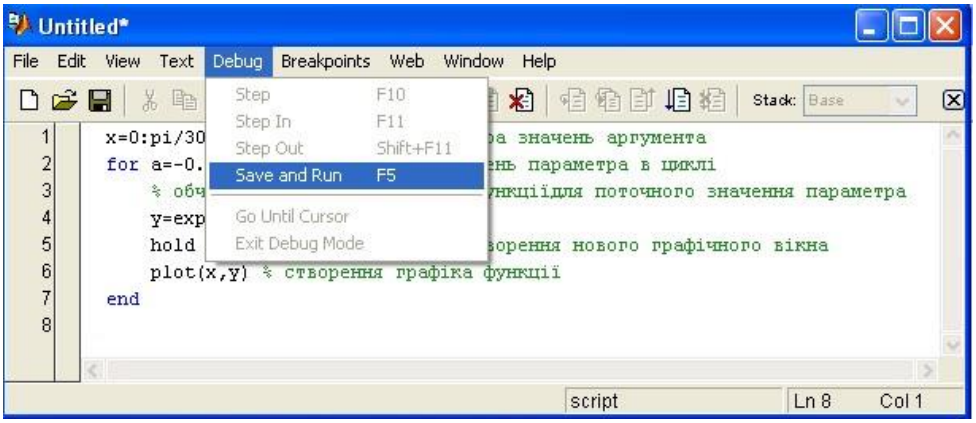


Рис. 4.10. Команди запуску програми на рішення

В результаті виконання програми PR1 з'явиться графічне вікно, зображене на рис. 4.11, яке містить потрібне сімейство кривих.

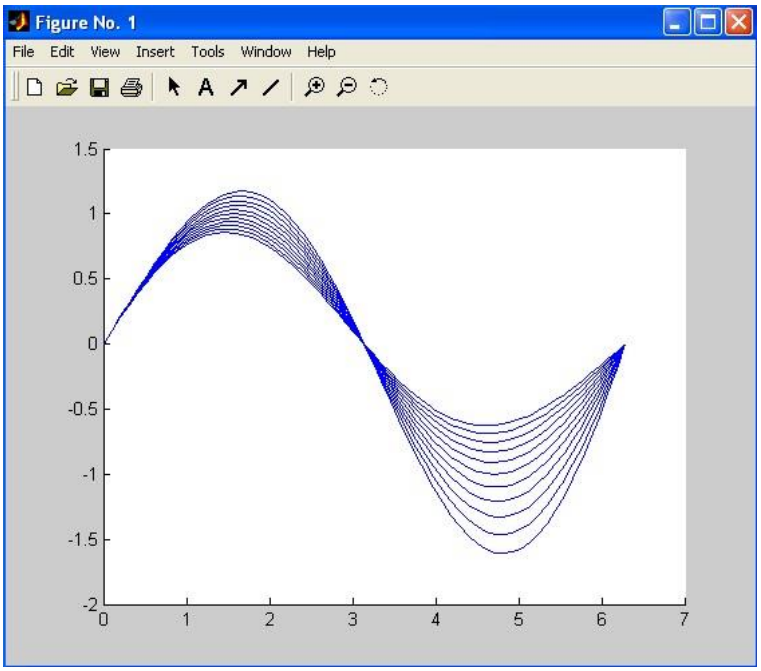


Рис. 4.11. Результат виконання програми PR1

Цикл while.

Цикл *for* використовується, коли визначено кінцеве число дій. Але існують алгоритми з заздалегідь невідомою кількістю повторень. Реалізувати їх дозволяє більш гнучкий цикл *while*.

Цикл *while* використовується для організації повторів однотипових дій у випадку, коли число повторів заздалегідь невідомо і визначається виконанням деякої умови. Цикл *while* працює, доки виконується умова циклу:

while умова повторення циклу команди MatLab end

Розглянемо приклад. Визначити суму і кількість ітерацій, при якій буде виконуватися рівність з точністю до 0,001.:

$$\sum_{n=0}^{\infty}\frac{1}{2^n}=1+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\ldots=2.$$

При складанні програми треба в першу чергу надати початкове значення змінній, в якій буде накопичуватися сума (s = 0) та надати початкове значення змінній, в якій буде вестись підрахунок кількості ітерацій (n = 0).

Слід звернути цвагу на те, що у цикла while, на відміну від for, немає змінної циклу, тому необхідно до початку циклу змінній *n*, яка буде вести

підрахунок кількості ітерацій (повторень у циклі) присвоїти початкове значення, тобто нуль, а всередині циклу збільшувати значення *n* на одиницю.

У заданому прикладі межа підсумовування нескінченність. Для рішення таких прикладів необхідно накопичувати суму доки виконується умова циклу.

Лістинг файл-функції, яка обчислює кількість ітерацій і суму ряду представлено на рис. 4.12.

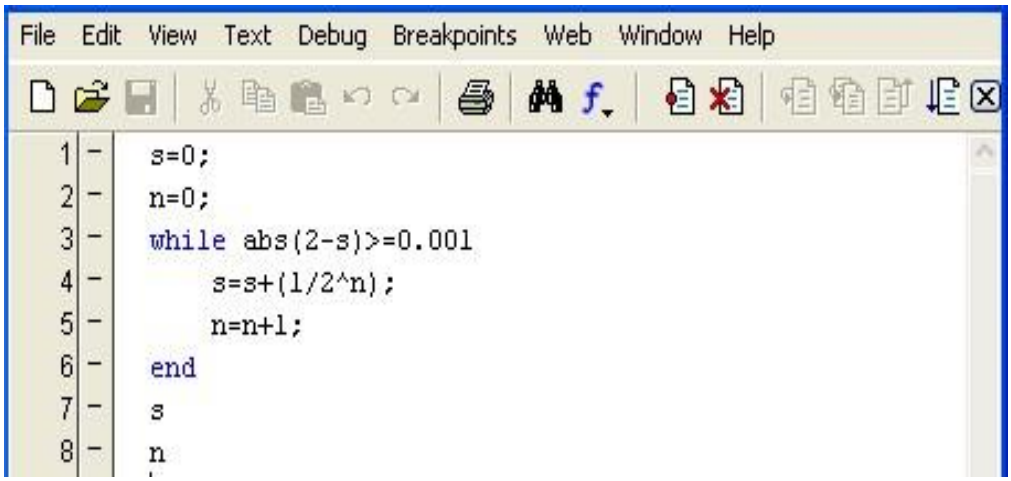


Рис. 4.12. Лістинг програми підрахунку суми та ітерацій

Запустивши програму на рішення отримаємо результат (рис. 4.13).

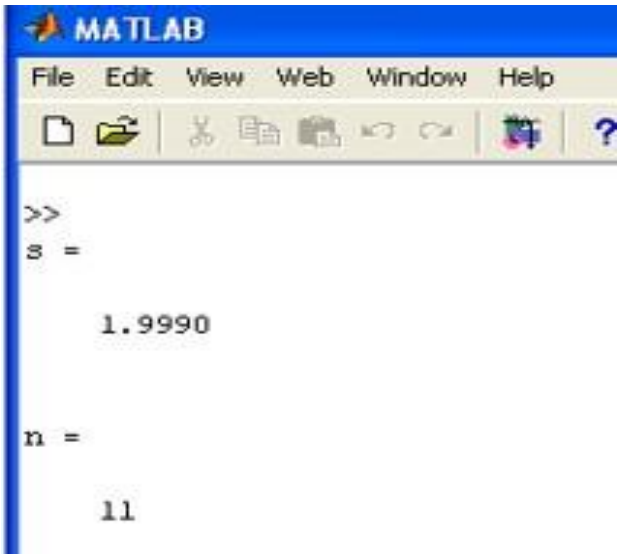


Рис. 4.13. Результат роботи програми

З результату можна зробити висновок, що за 11 ітерацій було накопичено суму, яка дорівнює 1,999.

Умова циклу *while* може містити логічний вираз, складений з операцій відношення і логічних операцій або операторів. Для задання умови повторення циклу допустимі операції відношення.

Запитання для самоконтролю

1. Що таке скрипт у Matlab і як його створити?
2. Як створюється функція у Matlab?
3. Як використовуються функції printf та sprintf?
4. У чому різниця між скриптом і функцією?
5. Як задається конструкція if…elseif…else?
6. Як використовуються логічні оператори в умовах?
7. Як працює цикл for у Matlab?
8. Як працює цикл while у Matlab?
9. Який синтаксис дострокового виходу з циклу?
10. Як уникнути нескінченного циклу в Matlab?

Лекція 5. Векторизація коду в Matlab

План

1. Найпростіші випадки векторизації. Функція sum
2. Функції zeros,ones,size,repmat,reshape
3. Логічні матриці
4. Функція image
5. Функції sort,unique

1. Найпростіші випадки векторизації. Функція sum

Будь-яке обчислення, яке *MathCAD* може виконати з поодинокими значеннями, він може виконувати поелементно з векторами і матрицями (табл. 5.1). Це можна реалізувати двома способами: послідовно виконуючи дії над кожним елементом масиву або використанням *оператора векторизації*. Для введення цього оператора необхідно над виділеним об'єктом натиснути одночасно клавіші *Ctrl+"-"* (обрати символ → з панелі *Matrix*). Векторизований об’єкт зображується з стрілкою нагорі.

Оператор векторизації змінює зміст операцій. Наприклад, якщо A деяка матриця. Тоді запис *exp(A)* є некоректним, оскільки аргументом функції *exp* повинна бути проста змінна, а не матриця. Застосування до цієї функції оператора векторизації призводить до обчислення функції *exp* від кожного елемента матриці і результатом також є матриця.

Таблиця 5.1

Оператори арифметичних дій з масивами

Дія	Вигляд	Опис
Множення	A*к	Множить кожен елемент A на скаляр к
	w*v	Скалярний добуток вектора-рядка на вектор стовпець
	A*B	Матричне множення: рядок – стовпець
	A*v	Множення матриця – вектор-стовпець
Ділення	A/к	Ділить кожен елемент масиву на к
Складання	A+B	Поелементне складання масивів
	A+к	Додавання до кожного елемента A числа к
Віднімання	A – B	Поелементне віднімання масивів
	A-к	Віднімання з кожного елемента A числа к
Векторне множення	u x v	Добуток векторів [Ctrl+8]. Результат - вектор

*Позначення: матриця – A, B, M, вектор – v, u, скаляр - к.

Виведення результатів

В системі передбачено відображення масивів в формі матриці та в формі таблиці (рис. 5.1).

$$b = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

+

a =

	0	1	2	3
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1

стиль «матриця» стиль «таблиця»

Рис. 5.1. Стилі виведення масивів

Функція sum

Застосування до вектору

```
>> x = [1 2 3 4 5];
>> sum(x)
15
```

Застосування до матриці

```
>> X = [10 20; 30 40]
X =
10 20
30 40
>> sum(X,1)
ans =
40 60

>> sum(X,2)
ans =
30
70

>> sum(X(:))
ans =
```

100

2. Функції zeros, ones, size, repmat, reshape

Функція **zeros** - створення матриці, заповненої одиницями

```
>> x = zeros(2,3)
x =
0 0 0
0 0 0
```

Функція **size** -отримання розміру матриці

```
>>size(zeros(2,3))
ans =
2 3
>> ones(size([1 2 3; 4 5 6]))
ans =
1 1 1
1 1 1
```

Функція **eye** - створення одиничної матриці

```
>> x = eye(2)
x =
1 0
0 1
```

Функція **ones** -створення матриці, заповненої одиницями

```
>>x = ones (2,3)
x =
1 1 1
1 1 1
>>x = ones ([2 3])
x =
1 1 1
1 1 1
```

Функція **repmat** - створює нову матрицю із копій матриці, розмножуючи її за принципом плитки на стіні

```
>> x = [1 2 3; 4 5 6];
>> repmat(x,1,2)
1 2 3 1 2 3
4 5 6 4 5 6
>> repmat(x,2,1)
1 2 3
4 5 6
1 2 3
4 5 6
```

Функція **reshape** - змінює розміри матриці, не змінюючи кількість елементів у них та їх порядок слідування (як у FORTRAN)

```
>> x = [1 2 3; 4 5 6]
x =
1 2 3
4 5 6
>> reshape(x,1,6)
ans =
1 4 2 5 3 6
>> reshape(x,3,2)
ans =
1 5
4 3
2 6
>> reshape(x(:),2,3)
ans =
1 2 3
4 5 6
```

3. Логічні матриці

У **Matlab** логічні матриці — це матриці, елементи яких можуть мати тільки два значення:

- 1 (логічне true)
- 0 (логічне false)

Застосовують для фільтрації даних, роботи з умовами (if, while), побудови масок для обробки зображень.

Використання логічних операторів - матриця створюється шляхом порівняння числових матриць:

```
A = [1 2 3; 4 5 6; 7 8 9];
B = A > 4  % Логічна матриця, де елементи A > 4
Результат:
B =
   0   0   0
   0   1   1
   1   1   1
```

Використання функції logical

C = logical([0 1 0; 1 0 1]) % Перетворення числової матриці на логічну

Використання вбудованих функцій:
Z = false(3,3); % 3×3 матриця нулів (false)
O = true(2,2); % 2×2 матриця одиниць (true)

Таблиця 5.2

Логічні операції (&, |, ~)

Оператор	Значення
~	«НЕ»
&	Матричне «І»
|	Матричне «АБО»

Таблиця 5.3

Логічні функції

Функція	Значення
any(x,1)	Векторне «АБО» по стовпцях матриці
any(x,2)	Векторне «АБО» за рядками матриці
all(x,1)	Векторне «І» по стовпцях матриці
all(x,2)	Векторне «І» за рядками матриці

A = [1 0 1; 0 1 0; 1 1 0];
B = [0 1 1; 1 0 1; 0 0 1];
C1 = A & B % Логічне "І"
C2 = A | B % Логічне "АБО"
C3 = ~A   % Логічне "НЕ"

Знаходження індексів (find)
idx = find(A) % Індекси ненульових (true) елементів

Використання логічних матриць для індексації
A = [10 20 30; 40 50 60; 70 80 90];
mask = A > 30; % Логічна матриця
B = A(mask)   % Витягуємо тільки елементи, що > 30

Перевірка наявності true значень
any(mask) % Чи є хоч одне true у векторі/матриці?
all(mask) % Чи всі елементи є true?

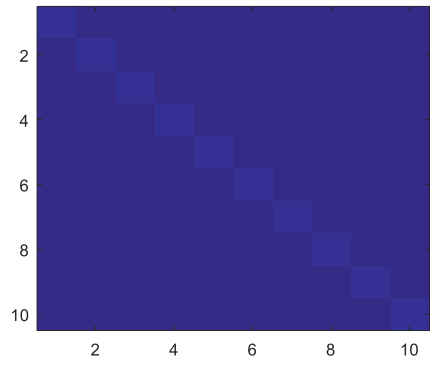
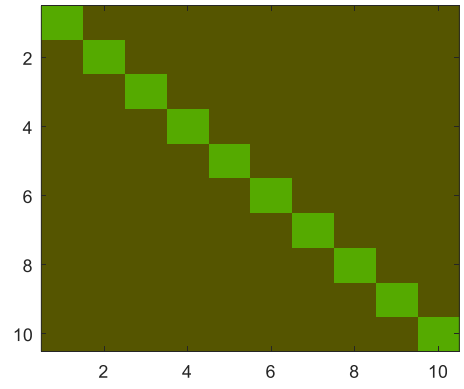
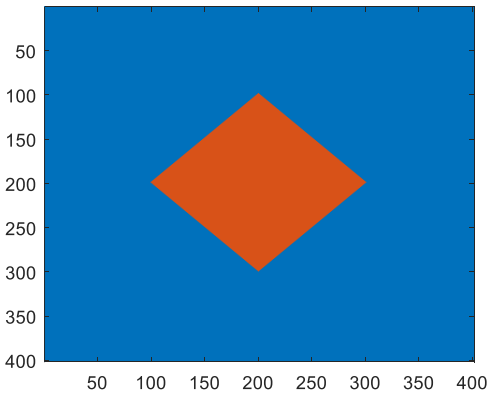
4. Функція image

Функція **image** використовується для відображення матриць як зображень. Вона безпосередньо візуалізує числові дані, відображаючи їх у вигляді кольорового зображення відповідно до поточної колірної карти (**colormap**).

Кольори визначаються палітрою кольорів. Номер кольору відраховується від одного. В табл. 5.4. показано приклад застосування функції image з різними логічними матрицями та функціями.

Таблиця 5.4

Приклад застосування функції image

Приклад із функцією eye	**Приклад із логічними матрицями**
>> M = eye(10) + 1; >> image(M);  >> colormap colorcube 	>> [X,Y] = meshgrid(-2:0.01:2); >> Z = ones (size (X)); >> Z(abs(X) + abs(Y) <= 1) = 2; >> image(Z) >> colormap lines 

Основний синтаксис

image(C) % Відображення матриці C як зображення
image(x, y, C) % Вказівка координат для зображення
h = image(...) % Отримання об'єкта Image

- **C** – матриця числових значень (відображаються як кольори).
- **x, y** – координати, що визначають положення пікселів.
- **h** – дескриптор об'єкта Image.

Відображення простої матриці

```
A = magic(5);  % Створює магічний квадрат 5×5
image(A);      % Відображає його як зображення
colormap(jet); % Використовує кольорову карту "jet"
colorbar;      % Додає шкалу кольорів
```

Масштабування кольорів

За замовчуванням image відображає кольори на основі значень матриці, які мають бути в межах [1, length(colormap)]. Якщо матриця містить значення поза цими межами, колір може бути неправильно відображений.

Рішення: Використовуйте imagesc, яка автоматично масштабує значення.

```
imagesc(A);  % Альтернативний спосіб для коректного відображення
colormap(hot);
colorbar;
```

Використання з RGB-зображеннями

Функція image може працювати з тривимірними масивами (m×n×3), що представляють **RGB-зображення**.

```
RGB = imread('peppers.png'); % Завантаження зображення
image(RGB);  % Відображення кольорового зображення
axis image;  % Збереження співвідношення сторін
```

Встановлення координатного простору

Зазвичай image використовує координати пікселів, але їх можна змінити:

```
x = [10 50]; y = [20 80]; % Вказуємо діапазон координат
image(x, y, A);
axis xy; % Фіксуємо порядок координат
```

Прозорість (AlphaData)

```
h = image(RGB);
h.AlphaData = 0.5; % Напівпрозорість 50%
```

Можливості та застосування

Переваги image:

- Висока швидкість відображення великих матриць.
- Можливість роботи з RGB та індексованими зображеннями.
- Гнучкі параметри кольорової карти.
- Простота використання для візуалізації числових даних.

Недоліки image:

- Не масштабує значення автоматично (треба використовувати imagesc).
- Не підходить для візуалізації 3D-даних.

5. Функції sort, unique

Функція sort

Функція sort використовується для сортування елементів вектора або матриці.

B = sort(A) % Сортує A у порядку зростання (за замовчуванням)
B = sort(A, 'descend') % Сортує A у порядку спадання
[B, idx] = sort(A) % Повертає відсортовані значення та їхні початкові індекси

Приклади:

1. Сортування вектора

```
A = [5 3 8 2 7];
B = sort(A)  % [2 3 5 7 8]
```

2. Сортування матриці по стовпцях (за замовчуванням)

```
A = [3 7 2; 8 1 5; 4 6 9];
B = sort(A)  % Сортує кожен стовпець окремо
Результат:
B =
    3    1    2
    4    6    5
    8    7    9
```

3. Сортування по рядках

```
B = sort(A, 2)  % Сортує кожен рядок окремо
Результат:
B =
    2    3    7
    1    5    8
    4    6    9
```

4. Отримання індексів сортування

```
A = [4 1 3 2];
[B, idx] = sort(A)
Результат:
B = [1 2 3 4]
idx = [2 4 3 1] % Початкові індекси значень
```

Функція unique

Функція unique використовується для знаходження унікальних значень у векторі або матриці.

B = unique(A) % Унікальні значення, відсортовані за зростанням

B = unique(A, 'stable') % Зберігає порядок появи елементів у A

B = unique(A, 'rows') % Унікальні рядки матриці

[B, ia, ic] = unique(A) % Додаткові вихідні аргументи

Приклади

1. Унікальні елементи вектора

```
A = [4 2 5 2 3 3 4 1];
B = unique(A)  % [1 2 3 4 5]
```

2. Збереження порядку

```
B = unique(A, 'stable')  % [4 2 5 3 1]
```

3. Отримання індексів

```
[B, ia, ic] = unique(A)
Результат:
B = [1 2 3 4 5]   % Унікальні значення
ia = [8 2 5 1 3]  % Перші входження
ic = [4 2 5 2 3 3 4 1] % Відповідність початковим індексам
```

4. Унікальні рядки матриці

```
A = [1 2; 3 4; 1 2; 5 6];
B = unique(A, 'rows')
Результат:
B =
    1    2
    3    4
    5    6
```

Запитання для самоконтролю

1. У чому перевага векторизації порівняно з циклами?
2. Як використовується функція sum для векторів?
3. Як працюють функції zeros і ones?
4. Як використовується функція size?
5. Яке призначення функції repmat?
6. Як змінити форму матриці за допомогою reshape?
7. Як створити логічну матрицю?
8. У чому особливість використання логічних умов у масивах?
9. Як відобразити зображення за допомогою функції image?
10. Як працюють функції sort і unique?

ЗМІСТОВИЙ МОДУЛЬ 2. ПОЧАТОК РОБОТИ З PYTHON

Лекція 6. Перші кроки з Python 3, функції та модулі

План

1. Арифметичні обчислення. Використання змінних.
2. Створення та використання скриптів

1. Арифметичні обчислення. Використання змінних

Комп'ютерна пам'ять складається з комірок. У кожній із них можна зберегти одне з 256 значень (від 0 до 255), тобто, це один байт інформації. 4 гігабайта пам'яті, наприклад, в смартфоні — це 4 мільярди таких комірок.

У кожної комірки є порядковий номер (адреса), за допомогою якого можна записати або прочитати інформацію. Припустимо, потрібно зберегти в пам'яті комп'ютера два числа – 25 і 6800. Уявити, як вони зберігаються в пам'яті, можна так
418414184241843418444184541846418474184841849418504185141852418534185441855418564185741858418594186041861418624186341864418654186641867418684186941870418714187241873418744187541876418774187841879418804188141882418834188441885680025

Щоб використовувати ці дані всередині програми, нам доведеться пам’ятати, за якою адресою ми їх зберегли. Як правило, даних у програмах на багато більше, ніж два числа. Тримати всі потрібні адреси в голові буде досить важко. Та і взагалі не потрібно, адже краще написати корисний код, ніж практикуватися в запису чисел.

Для спрощення роботи програмістами були придумані змінні.

Змінна — це ім'я, яке можна використовувати для доступу до даних. Погодьтесь, набагато легше тримати в голові ім'я **age**, ніж, наприклад, адресу комірки 41861.

Оператор присвоєння

З перших уроків ви вже знаєте , що інтерпретатор Python читає програму і виконує те, що там написано. Так от, якщо в програмі зустрінеться знак =, вийде таке:

1. Інтерпретатор знайде вільну клітинку пам'яті;
2. Запише в неї значення, яке знаходиться справа від знаку =
3. Прив’яже його з ім'ям, яке знаходиться зліва від знаку =

age = 25

Це зв'язування імені змінної і її значення називається **присвоюванням** , а знак =називається **оператором присвоювання** . Виходить, що присвоїти значення змінної в Python — це буквально написати ім'я змінної, оператор присвоювання та значення змінної. Саме в таком порядку.

Крім оператора присвоювання в Python є ще безліч різних операторів, про які буде розповідатися в наступних уроках. Зараз, однак, важливо розповісти про таке поняття як операнд .

Операнд - це те, з чим працює оператор. Наприклад, лівий операнд оператора присвоювання — це назва, а правий операнд — значення змінної.

Імена змінних

Щоб інтерпретатор Python міг розпізнати змінну, її назву не можна починати з цифри. Наприклад, 5var — це некоректне ім'я для змінної. А от в середині або в кінці назви, цифри цілком можна використовувати.

Для назв змінних використовуйте лише латинські букви та цифри. В імені змінної не можна використовувати пробіли, тому якщо назва складається з двох або більше слів, розділяйте їх знаком нижнього підкреслення,
наприклад, **my_age** або long_var_name_1.

Для цього стилю іменування програмісти навіть придумали окреме назву: Snake Case, тобто, зміїний реєстр. Тобто, **Foo_2**, FOO_2 і foo_2 — разні назви.

Ніколи не використовуйте символи **l**— строчну (маленьку) латинську букву «эль», **O** — прописну (велику) латинську букву «о» або **I** — прописну (велику) латинську букву «ай» для імені змінних з однієї букви. Тобто, l, O і I — погані імена для змінних. У деяких шрифтах ці символи не відрізняються від цифр, один від одного і нуля.

Крім цього, для назви змінних не можна використовувати ключові слова, а назви вбудованих функцій використовувати вкрай небажано.

Вивід змінних на екран

Для того, щоб вивести значення змінної на екран, можна використовувати саме такий спосіб, як в першій програмі. Тільки замість привітання потрібно написати ім'я змінної.

Тобто, якщо у нас у програмі є якась змінна, на екраі її значення можна вивести так:

```
a = 1
print(a)
```

Взагалі, **print** — це одна з вбудованих функцій мови Python.

Функції в програмуванні — це будь-які дії. Тобто , якщо ви напишете слово **print**, Python подумає, що потрібно діяти: у цьому випадку вивести що-то на екран.

Якщо, наприклад, потрібно вивести кілька змінних, можна використовувати функцію **print** кілька разів:

```
a = 1
b = 2
c = 3
print(a)
print(b)
print(c)
```

так і просто написати імена змінних в дужках, через кому.

```
a = 1
b = 2
c = 3
print(a, b, c)
```

У першому випадку значення кожної змінної виводиться з нового рядка, в другому — в одному рядку.

Відступи важливі

Якщо ви попробуєте запустити програму з «непотрібним» відступом для **print:**

```
a = 1
 print(a)
```

вона не працює і закінчиться помилкою:

```
IndentationError: unexpected indent
```

Додавання і віднімання

Змінним можна присвоювати не тільки прості значення, 13 або, наприклад, 22. Такі значення, до речі, **називаються літералами**. **Літерал** — це значення, на пряму записане в вихідному коді.

Так от, змінним можна присвоювати не тільки фіксовані значення - літерали. Їм можна присвоювати результат роботи інших операторів. У цій програмі, наприклад, **a** змінній буде присвоєний результат роботи оператора додавання, а змінній **b** — результат роботи оператора віднімання.

```
a = 13 + 5
b = 13 - 5
print(a, b)
```

Замість чисел можна поставити імена змінних і це теж працює. Ця програма, наприклад, виводить на екран **26:**

```
a = 13 + 5
b = 13 - 5
c = a + b
print(c)
```

У цих прикладах правим **операндом** для оператора присвоєння є віднімання , це є частина коду, яка вираховується у значення.

Множення і ділення

Пітон уміє не тільки додавати і віднімати. Він вміє множити і ділити! Наприклад, ось ця програма виводить на екран **12** і **2.4**

```
a = 2 * 6
b = a / 5
print(a, b)
```

Результат множення надрукувався без крапки (12), а результат ділення (2.4) — з крапкою між цілою та дробовою частиною.

Тобто, в Пітоні є як цілі числа, так і числа дробові. Виходить, що змінній можна присвоїти не тільки ціле, але і дробове значення, наприклад:

```
pi = 3.14
e = 2.7182
```

2. Створення та використання скриптів

Python-скрипти — це текстові файли з розширенням .py, які містять код, що виконується інтерпретатором Python. Вони використовуються для автоматизації завдань, створення програм, обробки даних тощо.

Щоб створити скрипт, потрібно:

1. Відкрити будь-який текстовий редактор (VS Code, PyCharm, Notepad++, або стандартний Блокнот).
2. Написати Python-код.
3. Зберегти файл з розширенням .py.

Приклад:

```
# my_script.py
print("Hello, world!")  # Виведе текст у консоль
```

Запуск Python-скрипту

Якщо Python встановлений, можна запустити скрипт через термінал або командний рядок:

python my_script.py

Або, якщо використовується python3:

python3 my_script.py

Також його можна запустити із середовища розробки:

- У **VS Code** натисніть Run або F5.
- У **PyCharm** відкрийте файл і натисніть Shift + F10.
- У **Jupyter Notebook** використовуйте !python my_script.py.

Ставити розширення .py у файлу не обов'язково, але це бажано робити. Можна сказати, що це гарний тон - створювати скрипти Python з таким розширенням. Скрипти часто використовується для автоматизації рутинних завдань:

- Робота з текстовими файлами
- Зміна та створення Excel-файлів
- Обробка PDF
- Обробка великих масивів даних
- Аналіз та візуалізація даних

Запитання для самоконтролю

1. Як задати змінну у Python?
2. Які основні типи даних використовуються у Python?
3. Як виконати арифметичні операції в Python?
4. Як створити скрипт у Python?
5. Як запустити скрипт у середовищі розробки?
6. Як імпортувати модуль у Python?
7. Яке призначення модуля math?
8. Як виводити дані на екран?
9. Як коментувати код у Python?
10. Як викликати допомогу щодо функцій у Python?

Лекція 7. Функції та модулі в Python 3

План

1. Знайомство з функціями та модулями
2. Логічні елементи, блоки if і цикли while

1. Знайомство з функціями та модулями

Зазвичай реалізація складних задач містить величезні фрагменти коду. Зручним способом організувати великий фрагмент коду в більш зручні фрагменти є створення функцій. Функції в Python є основою при написанні програм.

Іноді функцію порівнюють з "чорним ящиком", коли відомо, що на вході і що при цьому на виході, а нутрощі "чорного ящика" часто бувають приховані.

Існує велика кількість вбудованих функцій. Наприклад функція *abs()*, приймає на вхід один аргумент – об’єкт числового типу і повертає абсолютне значення для цього об’єкта.

```
>>> abs (-9)
9
```

Результат виклику функції можна присвоїти змінній, використовувати його в якості операндів математичних виразів, тобто складати більш складні вирази.

```
x = abs (-15)
y = x +5
print('y=', y)
```

Результатом запуску даного коду буде:

```
20
```

Крім складання складних математичних виразів Python дозволяє передавати результати виконання функцій в якості аргументів інших функцій без використання додаткових змінних:

```
print(abs (-15))
```

Отримаємо:

```
15
```

Спочатку визначається абсолютне значення цілого числа -15, а потім за допомогою функції *print()* виводиться на екран результат розрахунку.

Можна реалізовувати власні функції.

Функція – це іменований фрагмент коду, відокремлений від інших. Вона може приймає будь-яку кількість будь-яких вхідних параметрів і повертати будь-яку кількість будь-яких результатів.

З функцією можна зробити дві речі: визначити; – викликати.

Щоб визначити функцію, використовується наступна конструкція:

def ІМ'Я_ФУНКЦІЇ (ВХІДНІ_ПАРАМЕТРИ):
ФУНКЦІЯ

Імена функцій підкоряються тим же правилам, що і імена змінних (вони повинні починатися з літери або _ і містити тільки букви, цифри або _).

Функція може не містити параметри але круглі дужки все рівно необхідно вказувати:

```
def do_nothing():
...    pass
```

Тіло функції відділяється пробілами. Використання виразу *pass* відображає, що функція нічого не робить.

Щоб викликати функцію вказується її ім’я та дужки з параметрами:

```
do_nothing()
```

Всі дії в програмі виконуються послідовно зверху вниз. Це означає, що перш ніж використовувати ідентифікатор в програмі, його необхідно попередньо оголосити, присвоївши йому значення. Тому визначення функції має бути розташоване перед викликом функції.

Визначимо і викличемо функцію, яка не має параметрів і виводить на екран одне слово:

```
def salute():
    print('Hi!')
salute()
```

Отримаємо:

```
Hi!
```

Коли викликається функція *salute()*, Python виконує код, розташований всередині її опису. У цьому випадку він виводить одне слово і повертає управління основній програмі.

Параметри функції

Функція може приймати параметри та повертати значення. Параметри функції – звичайні змінні, якими функція користується для внутрішніх

розрахунків. Якщо параметрів декілька – вони перераховуються через кому [5, 10, 13].

Формальні параметри – параметри, що вказуються при оголошенні функції.

Фактичні параметри (аргументи) – параметри, що передаються в функцію при її виклику.

```
def print_numbers(limit):
    for i  in range (limit):
        print(i)

n=int(input('Введіть кількість елементів: '))

print_numbers(n)
```

Результатом запуску даного коду буде:

```
Введіть кількість елементів: 5
0
1
2
3
4
```

Значення, які передаються в функцію при виклику, називаються ***аргументами***. Коли функція викликається з аргументами, їх значення копіюються у відповідні параметри всередині функції.

Існують функції які просто щось виконують, наприклад вбудована функція *print()* яка виводить на екран певні значення:

```
n=15
print(n)
```

Отримаємо:

```
15
```

А є функції які повертають розраховане значення, що може бути привласнене змінній, наприклад вбудована функція *input():*

```
a=input('Введіть слово: ')
```

Отримаємо:

```
Введіть слово: Ні!
```

Для того щоб переглянути результат роботи такої функції потрібно скористатися функцією *print()*

```
a=input('Введіть слово: ')
print(a)
```

Отримаємо:

```
Hi!
```

При необхідності повернути результат роботи функції в програму, з якої вона викликалася, для її подальшого оброблення застосовується команда ***return***. Вираз, що стоїть після *return* буде повертатися в якості результату виклику функції.

Без аргументів *return* використовується для виходу з функції (інакше вихід відбудеться при досягненні кінця функції). В Python функції здатні повертати кілька значень одночасно.

```
def PrintRoots(a, b, c):
    D = b**2 - 4 * a * c
    import math
    x1 = (-b + math.sqrt(D)) / 2 * a
    x2 = (-b - math.sqrt(D)) / 2 * a
    return x1, x2
print (PrintRoots(1.0, 0, -1.0))
```

Результатом запуску даного коду буде:

```
(1.0, -1.0)
```

Крім того, результати виконання функції можна привласнювати відразу декільком змінним:

```
x1, x2 = PrintRoots(1.0, 0, -1.0)
print("x1 =", x1, "\nx2 =", x2)
```

Результатом запуску даного коду буде:

```
x1 = 1.0
x2 = -1.0
```

Всередині функції може міститися довільна кількість *return*. Однак спрацює лише один з них.

```
def traffic_light (color):
    if color == 'red':
        return "STOP!"
    elif color == "green":
```

```
        return "GO!"
    elif color == 'yellow':
        return "GET READY!"
    else:
        return "Broken traffic light!"
```

Викликавши функцію *traffic_light()*, передавши їй в якості аргументу рядок 'blue'.

```
result = traffic_light('blue')
print(result)
```

Функція зробить наступне:

- присвоїть значення 'blue' параметру функції color;
- пройде по логічному ланцюжку if-elif-else;
- поверне рядок;
- присвоїть рядок змінній result. Результатом буде:

```
'Broken traffic light!'
```

Функція може приймати будь-яку кількість аргументів (включаючи нуль) будь-якого типу. Вона може повертати будьяку кількість результатів (також включаючи нуль) будь-якого типу. Якщо функція не викликає *return* явно, буде отримано результат *None*.

```
def do_nothing():
    pass
```

Результатом буде:

```
None
```

None – це спеціальне значення в Python, яке заповнює собою порожнє місце, якщо функція нічого не повертає. Воно не є булевим значенням *False*, незважаючи на те що схоже на нього під час перевірки булевої змінної.

Аргументи функцій

Функція може приймати довільну кількість аргументів або не приймати їх зовсім. В функцію можна передавати не лише окремі об’єкти але і колекції/послідовності (список, кортеж та ін.). крім того, аргументи можуть бути позиційними, іменованими, обов’язковими та не обов’язковими.

Позиційні аргументи

Найбільш поширений тип аргументів – це ***позиційні аргументи***, чиї значення копіюються у відповідні параметри згідно з порядком проходження.

```
def func(a, b, c):
    return a+b*c
print(func(1, 2, 3))                    # a = 1, b = 2, c = 3
```

Отримаємо:

```
7
```

Незважаючи на поширеність аргументів такого типу, у них є недолік, який полягає в тому, що потрібно запам'ятовувати значення кожної позиції.

Іменовані аргументи

Щоб уникнути плутанини з позиційними аргументами, можна вказати аргументи за допомогою імен відповідних параметрів. Порядок проходження аргументів в цьому випадку може бути іншим:

```
print (func(a =2, b = 1, c = 3))
```

Отримаємо:

```
5
```

Можна об'єднувати позиційні аргументи та іменовані аргументи.

```
print (func(2, 2, c = 3))
```

Отримаємо:

```
8
```

Якщо викликати функцію, що має як позиційні аргументи, так і іменовані аргументи, то позиційні аргументи необхідно вказувати першими.

Модульність в Python

Якщо код програми є складним, розбиття її на окремі функції допомагає спростити його для візуального сприйняття. Якщо цього недостатньо, є сенс винести частину функцій та пов'язаних з ними оголошень за межі основного файлу програми.

Такі додаткові файли з кодом, що використовується в програмі, називаються модулями. Найчастіше вони містять оголошення функцій та констант, які далі можуть бути підключені (імпортовані) в головну програму і вільно в ній використовуватися. Об'єкти з модуля можуть бути імпортовані в інші модулі. Файл утворюється шляхом додавання до імені модуля розширення *.py*. При імпорті модуля інтерпретатор шукає файл спочатку в

поточному каталозі, потім в каталогах, зазначених у змінній оточення *PYTHONPATH*, потім в залежних від платформи шляхах за замовчуванням, а також в спеціальних файлах з розширенням *'.pth'*, які лежать в стандартних каталогах. Можна внести зміни в *PYTHONPATH* і в *'.pth'*, додавши туди свій шлях. Каталоги, в яких здійснюється пошук, можна подивитися в змінній *sys.path*.

Великі програми, як правило, складаються з стартового файлу – файлу верхнього рівня, і набору файлів-модулів. Головний файл займається контролем програми. У той же час модуль – це не тільки фізичний файл. Модуль являє собою колекцію компонентів. У цьому сенсі модуль – це простір імен, – *namespace*, і всі імена всередині модуля ще називаються атрибутами – такими, наприклад, як функції і змінні [5, 8, 9, 13].

Є велика кількість вбудованих модулів, що дозволяють виконувати складні математичні операції (math), працювати з датами (datatime)/часом (time), випадковими числами (random), операційною та файловою системами (os) та ін.

Модуль *math* надає додаткові функції для роботи з числами, а також стандартні константи. Перш ніж використовувати модуль, необхідно підключити його за допомогою інструкції:

Модуль *math* надає наступні стандартні константи: ***pi*** – повертає число n. ***e*** – повертає значення константи *e*.

```
import math

print(math.pi)
print(math.e)
```

Отримаємо:

```
3.141592653589793
2.718281828459045
```

2.

3. Логічні елементи, блоки if і цикли while

Основними алгоритмічними структурами є: слідування, розгалуження, цикл. ***Слідування*** – команди виконуються послідовно одна за іншою.

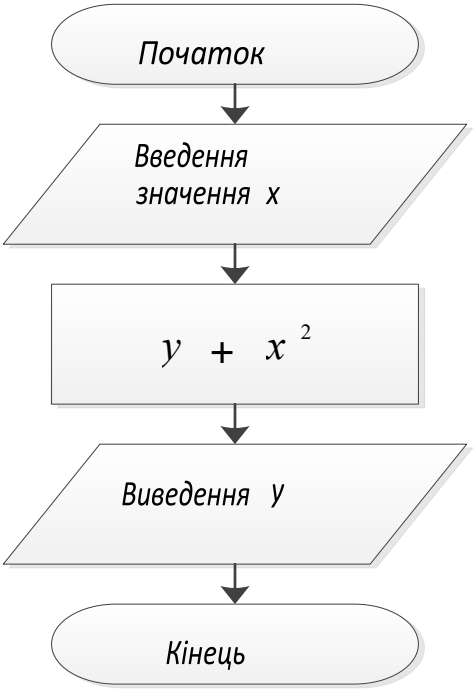


Рис. 7.1. Приклад блок-схеми реалізації лінійного алгоритму

Розгалуження – алгоритм, що містить хоча б одну умову в результаті перевірки якої може виконуватись розділення на декілька паралельних гілок. Кожна з гілок може містити також розділення, послідовні дії або цикли.

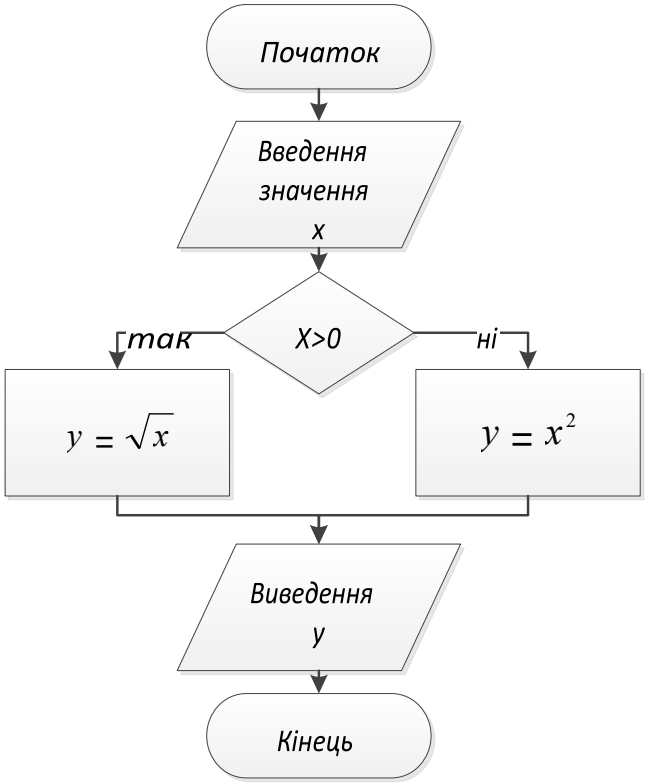


Рис. 7.2. Приклад блок-схеми реалізації алгоритму з розгалуженням

Цикл – інструкції що виконують одну і ту ж послідовність дій поки діє задана умова.

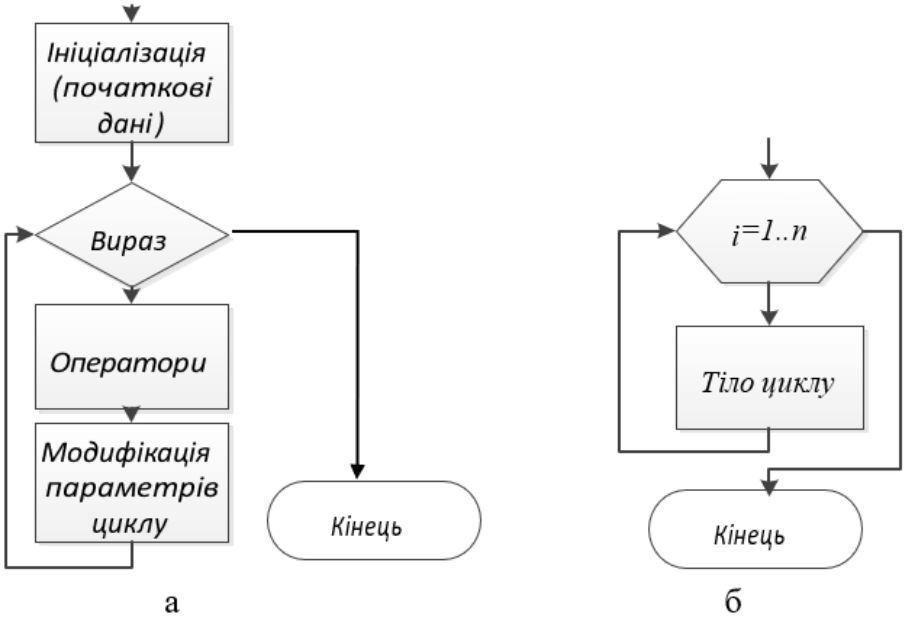


Рис. 7.3. Приклад блок-схеми реалізації циклічного алгоритму:
а – цикл з передумовою; *б* – цикл з лічильником

Реалізація алгоритмів з розгалуженням

Хід виконання програми може бути лінійним, тобто таким, коли вирази виконуються, починаючи з першого і закінчуючи останнім, по порядку, не пропускаючи жодного рядка коду. Але частіше буває зовсім не так. При виконанні програмного коду деякі його ділянки можуть бути пропущені.

Припустимо, в реальному житті людина живе за розкладом (можна сказати, розклад – це своєрідний "програмний код", який слід виконати). У її розкладі о 18.00 стоїть похід в басейн. Однак людині надходить інформація, що басейн не працює. Цілком логічно скасувати своє заняття з плавання. Тобто однією з умов відвідування басейну повинно бути його функціонування, інакше повинні виконуватися інші дії.

Схожа нелінійність дій може бути і в комп'ютерній програмі. Частина коду повинна виконуватися лише при певному значенні конкретної умови. Найпростішою в Python для опису розгалужуючої структури, де дії виконуються лише у випадку істинності умови, є така конструкція:

```
if ЛОГІЧНА_УМОВА:
    ПОСЛІДОВНІСТЬ_ВИРАЗІВ
```

Цю конструкцію на блок-схемі можна зобразити:

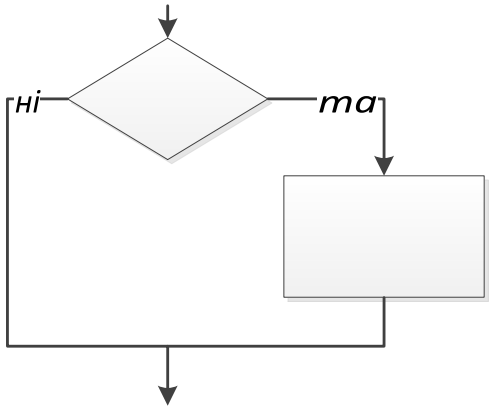


Рис. 7.4. Блок-схема реалізації конструкції if

Першим йде ключове слово ***if*** (англ. "Якщо"); за ним – логічний вираз; потім двокрапка, що позначає кінець заголовка оператора, а після неї – будь-яка послідовність виразів або тіло умовного оператора, яке буде виконуватися в разі, якщо умова в заголовку оператора істинна.

```
x = 2
if x > 0:
   print("x – додатнє")
if x < 0:
   print("x – від’ємне")
```

Результатом запуску даного коду буде:

```
x – додатнє
```

1. Привласнили значення *2* змінній *x*.
2. Зробили умовне порівняння за допомогою операторів *if*, виконуючи різні фрагменти коду в залежності від значень змінної *x*.
3. Викликали функцію *print()*, щоб вивести текст на екран.

Рядки *if* в Python є операторами, які перевіряють, чи є значення виразу (в даному випадку змінна x) рівним *True*.

print() – це вбудована в Python функція для виведення інформації. Вбудовані функції Python – це іменовані фрагменти коду, які виконують певні операції.

Кожен рядок *print()* відокремлений пробілами під відповідною перевіркою.

У більшості мов програмування символи начебто фігурних дужок (*{і}*) або ключові слова *begin* і *end* застосовується для того, щоб розбити код на розділи. У цих мовах хорошим тоном є використання відбиття пробілами, щоб зробити програму більш зрозумілою для себе та інших. Існують навіть інструменти, які допоможуть красиво вибудувати код.

Гвідо ван Росум при розробці Python вирішив, що виділення пробілами буде досить, щоб задати структуру програми і уникнути уведення дужок. Python відрізняється від інших мов тим, що пробіли в ньому використовуються для того, щоб задати структуру програми.

Як правило, використовують чотири пробіли для того, щоб виділити кожен підрозділ, хоча можна використовувати будь-яку кількість пробілів, Python чекає, що всередині одного розділу буде застосовуватися однакова кількість пробілів.

З огляду на це, в конструкції *if* код, який виконується при істинності умови, повинен обов'язково мати відступ вправо. Решта коду (основна програма) повинен мати той же відступ, що і слово *if*.

Зустрічається і більш складна форма розгалуження: *if-else*. Якщо умова при інструкції *if* є хибною, то виконується блок коду при інструкції *else*:

if ЛОГІЧНА_УМОВА:
ПОСЛІДОВНІСТЬ_ВИРАЗІВ_1
else:
ПОСЛІДОВНІСТЬ_ВИРАЗІВ_2

Цю конструкцію на блок-схемі можна зобразити:

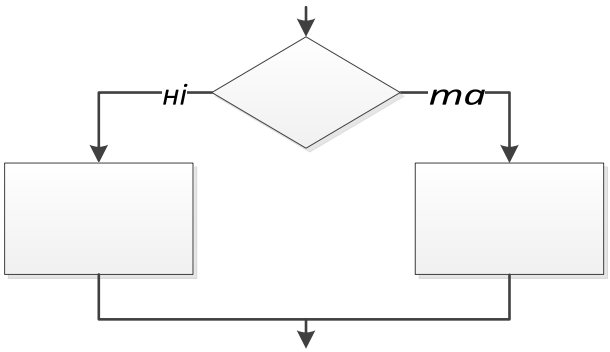


Рис. 7.5. Блок-схема реалізації конструкції if-else

Працює ця конструкція наступним чином. Спочатку перевіряється перша умова і, якщо вона істинна, то виконується перша послідовність виразів. Якщо умова не виконується потік виконання переходить до рядка, який йде після *else*.

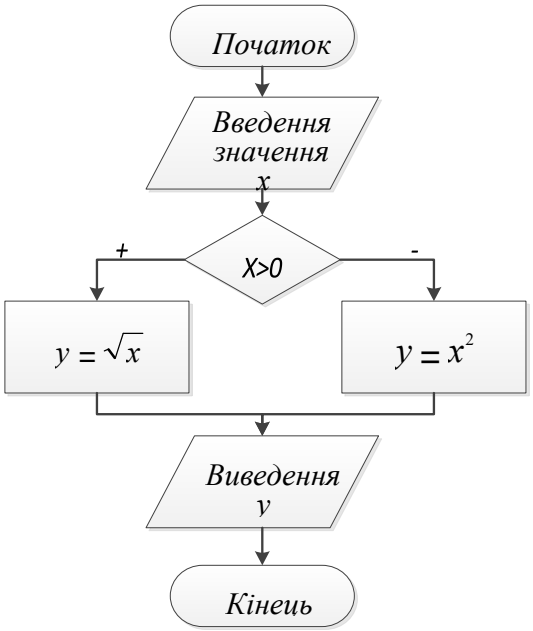


Рис. 7.6. Приклад блокконструкції if-схеми реалізації -else

```
x = int(input("Введіть x="))
if x > 0:
    y=x**0.5
else:
    y=x**2
 print("y =", y)
```

Отримаємо:

```
Введіть x= 9 y = 3.0
```

Альтернативні гілки програми

Логіка програми що виконується може бути складнішою, ніж вибір однієї з двох гілок.

Умовний оператор *if* має розширений формат, що дозволяє перевіряти кілька незалежних одна від одної умов і виконувати один з блоків, поставлених у відповідність з цими умовами. У загальному вигляді оператор виглядає так:

```
if ЛОГІЧНА_УМОВА_1:
    ПОСЛІДОВНІСТЬ_ВИРАЗІВ_1
elif ЛОГІЧНА_УМОВА_2:
    ПОСЛІДОВНІСТЬ_ВИРАЗІВ_2
elif ЛОГІЧНА_УМОВА_3:
    ПОСЛІДОВНІСТЬ_ВИРАЗІВ_3
...
else:
   ПОСЛІДОВНІСТЬ_ВИРАЗІВ_N
```

Цю конструкцію на блок-схемі зображено на рис. 7.7.

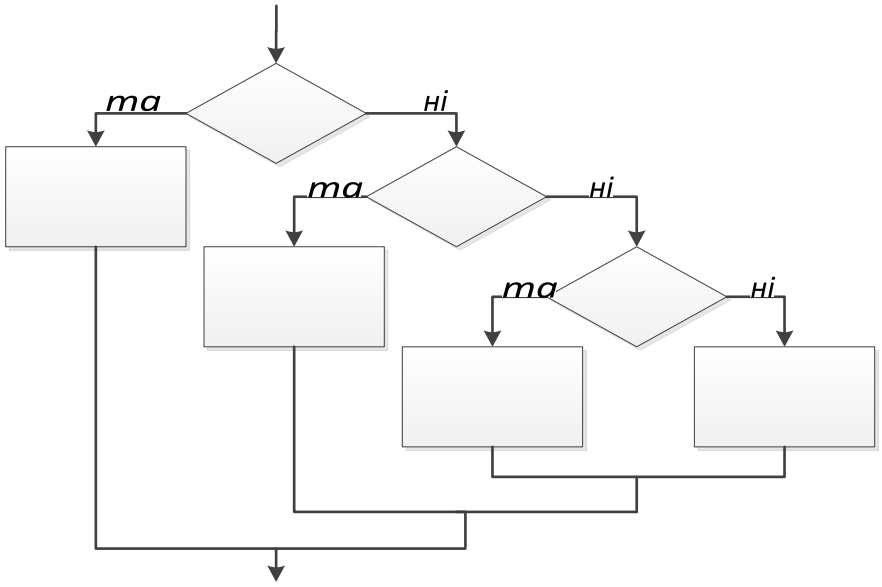


Рис. 7.7. Блок-схема реалізації конструкції if-elif-else

Працює ця конструкція таким чином. Спочатку перевіряється перша умова і, якщо вона істинна, то виконується перша послідовність виразів. Після цього потік виконання переходить до рядку, який йде після умовного оператора (тобто за послідовністю виразів *N*). Якщо перша умова рівна *False*, то перевіряється друга умова (наступна після *elif*), і в разі його істинності виконується послідовність 2, а потім знову потік виконання переходить до рядка, наступного за оператором умови.

Аналогічно перевіряються всі інші умови. До гілки програми *else* потік виконання доходить тільки в тому випадку, якщо не виконується жодна з умов.

Ключове слово *elif* походить від англ. "*Else if*" – "інакше якщо". Тобто умова, яка слідує після нього перевіряється тільки тоді, коли всі попередні умови хибні.

```
if not True:
   print("1")
elif not (1+1==3):
   print("2")
else:
   print("3")
```

Результатом запуску даного коду буде:

```
3
```

Оператор pass

В процесі роботи над програмою слід намагатися після кожної зміни мати працюючу програму, але іноді не завжди відразу відомо, що необхідно виконати якщо умова приймає істинне значення, а що в протилежному випадку. В Python інструкції з розгалуженням, або цикли, або функції з порожнім тілом заборонені, тому в якості тіла використовується "порожній оператор" ***pass***.

Припустимо, що заплановано використання умовного оператора з декількома умовами, але встигнуто написати тільки один з блоків умовного оператора. При цьому постає питання, як її налагодити, якщо програма не виконується через синтаксичну помилку.

```
if not True:
    print("1")
elif not (1+1==3):
elif not (1+1==4):
elif not (1+1==5):
```

Блоки для випадків, коли значення not (1+1==3), not (1+1==4), not (1+1==5), ще не написані, тому програма не виконується через помилку SyntaxError: expected an indented block.

Ключове слово *pass* можна вставити на місце відсутнього блоку:

```
if not True:
    print("1")
elif not (1+1==3):
    pass
elif not (1+1==4):
    pass
elif not (1+1==5):
    pass
```

Реалізація циклічних алгоритмів

У реальному житті ми досить часто зустрічаємося з циклами. У комп'ютерних програмах поряд з інструкціями розгалуження (тобто вибором шляху дії) також існують інструкції циклів (повторення дії). Якби інструкцій циклу не існувало, довелося б багато разів вставляти в програму один і той же код поспіль стільки раз, скільки потрібно виконати однакову послідовність дій.

Цикли – це інструкції, які виконують одну й ту ж саму послідовність дій, поки діє задана умова.

Кожен циклічний оператор має тіло циклу – якийсь блок коду, який інтерпретатор буде повторювати поки умова повторення циклу буде залишатися істинною.

В програмуванні розрізняють такі види циклів:

- Цикл з передумовою – виконує дії поки умова є істинною (while).
- Цикл з післяумовою – спочатку виконуються команди, а потім перевіряється умова (do…while).
- Цикл з лічильником – виконується задану кількість разів (for).
- Сумісний цикл – виконує команди для кожного елементу із заданого набору значень (for…which).

В Python присутні лише цикл з передумовою (while) та цикл for, що поєднує в собі два види – цикл з лічильником та сумісний цикл.

Оператор циклу while

Універсальним організатором циклу в мові програмування Python є конструкція *while* [5, 6, 7, 8, 9, 11]. Слово "while" з англійської мови перекладається як "поки" ("поки логічний вираз повертає істину, виконувати певні операції"). Конструкцію *while* на мові Python можна описати наступною схемою:

while УМОВА_ПОВТОРЕННЯ_ЦИКЛУ:
ТІЛО_ЦИКЛУ

Дуже схоже на оператор умови – тільки тут використовується інше ключове слово: *while* (англ. "Поки").

Дану конструкцію як правило використовують при повторі уведення даних користувачем, поки не буде отримано коректне значення:

```
correct_choice = False
while not correct_choice:
   choice = input("Введіть число 1 або 2:")
   if choice == "1" or choice == "2":
      correct_choice = True
   else:
      print ("Не правильно введено число, повторіть введеня")
```

Результатом запуску даного коду буде:

```
Введіть число 1 або 2:3
Не правильно введено число, повторіть введеня.
Введіть число 1 або 2:4
Не правильно введено число, повторіть введеня
Введіть число 1 або 2:1
>>>
```

1. Визначили логічну змінну *correct_choice*, присвоївши їй значення *False*.
2. Оператор циклу перевіряє умову *not correct_choice*: заперечення False – істина. Тому починається виконання тіла циклу: виводиться запрошення *«Enter your choice, please (1 or 2):»* і очікується уведення користувача.
3. Після натискання клавіші Enter введене значення порівнюється з рядками *«1»* і *«2»*, і якщо воно дорівнює одній з цих значень, то змінній *correct_choice* присвоюється значення *True*. В іншому випадку програма виводить повідомлення *«Не правильно введено число, повторіть введеня»*.
4. Оператор циклу знову перевіряє умову і якщо вона як і раніше істинна, то тіло циклу повторюється знову, інакше потік виконання переходить до наступного оператору, і інтерпретатор виходить з циклу і виконання програми припиняється або виконуються дії що прописані поза тілом циклу.

Ще один варіант використання оператора циклу – обчислення формул із змінним параметром.

$$\sum_{i=1}^{n} i^3 = 1^3 + \cdots + n^3$$

В цьому випадку, параметром, що змінюються є *i*, причому *i* послідовно приймає значення в діапазоні від *1* до *n*.

За допомогою оператора циклу *while* рішення буде виглядати так:

```
n = int(input("Введіть n: "))
sum = 0
i = 1
while i <= n:
    sum += i**3
    i += 1
print ("sum = ", sum)
```

Результатом запуску даного коду буде:

```
Введіть n: 5
sum =  225
```

1. Необхідно ввести *n* – граничне значення *i*.
2. Ініціалізація змінної *sum* – в ній буде зберігатися результат, початкове значення – *0*.

3. Ініціалізація змінної *i* (лічильника *i*) – за умовою, початкове значення – *1*.
4. Починається цикл, який виконується, поки *i* <= *n*.
5. У тілі циклу в змінну *sum* записується сума значення з цієї змінної, отриманої на попередньому кроці, і значення *i*, піднесеної в куб.
6. Лічильнику *i* присвоюється наступне значення.
7. Після завершення циклу виводиться значення *sum* після виконання останнього кроку.

Наступне значення лічильника отримують додаванням до його поточного значення кроку циклу (в даному випадку крок циклу дорівнює *1*). Крок циклу при необхідності може бути від'ємним, і навіть дробовим. Крім того, крок циклу може змінюватися на кожній ітерації (тобто при кожному повторенні тіла циклу).

Нескінченні цикли

Іноді можна зіткнулися з проблемою нескінченного повторення блоку коду, що виникає, наприклад, через семантичну помилку в програмі:

```
i = 0
while i < 10:
    print (i)
```

Такий цикл буде виконуватися нескінченно, тому що умова *i <10* завжди буде істинною, адже значення змінної *i* не змінюється: така програма буде виводити нескінченну послідовність нулів.

У циклів немає обмеження кількості повторень тіла циклу, тому програма з нескінченним циклом буде працювати безперервно, поки не буде зроблено аварійної зупинки натисканням комбінації клавіш Ctrl+C, не зупинимо відповідний процес засобами операційної системи, або не будуть вичерпані доступні ресурси комп'ютера (наприклад, може бути досягнуто максимально допустимої кількості відкритих файлів), чи не виникне виняток. У таких ситуаціях кажуть, що програма зациклилася.

Знайти зациклення в програмі іноді буває не так-то просто, адже в реальних програмах зазвичай використовується багато циклів, кожен з яких потенційно може виконуватися нескінченно.

Проте, відсутність обмеження на кількість повторів тіла циклу дає нові можливості. Багато програм представляють собою цикл, в тілі якого проводиться відстеження та оброблення дій користувачів або запитів з

інших програмних і автоматизованих систем. При цьому такі програми можуть працювати без перебоїв дуже тривалі терміни, іноді роками.

Цикли – це дуже потужний засіб реалізації, але вони вимагають деякої уважності.

Якщо необхідно, щоб цикл виконувався до тих пір, поки щось не станеться, але точно не відомо, коли ця подія трапиться, можна скористатися нескінченним циклом, що містить оператор ***break***.

```
i = 1

while True:
    if i > 5:
        break  # Перериваємо цикл
    print (i)
    i += 1
```

Результатом запуску даного коду буде:

```
1
2
3
4
5
```

1. Виведемо 5 чисел починаючи з *1*. Змінній *i* привласнено початкове значення *1*.
2. Найпростішою умовою для нескінченного циклу є значення True.
3. Виконується перевірка значення змінної *i* зі значенням *5*. Якщо воно співпадає або є більшим, то виконується оператор *break* що переривається виконання циклу.
4. Виводиться на екран значення числа (виконується якщо не спрацював оператор *break).*
5. Збільшується значення змінної *i* на *1*

Іноді потрібно не переривати весь цикл, а лише пропустити по якійсь причині одну ітерацію. ***Continue*** дозволяє перейти до наступної ітерації циклу до завершуючи виконання всіх виразів всередині циклу.

Розглянемо приклад: виводяться на екран цілі числа в діапазоні від 1 до 10, крім чисел 3, 4, 5.

```
i=0
while i<=10:
   i+=1
   if 3<=i<=5:
        continue
   print(i)
```

Результатом запуску даного коду буде:

```
1
2
6
7
8
9
10
11
```

Альтернативна гілка else

Мова Python дозволяє використовувати розширений варіант оператора циклу:

while УМОВА_ПОВТОРЕННЯ_ЦИКЛУ:
ТІЛО_ЦИКЛУ
else:
АЛЬТЕРНАТИВНА_ГІЛКА_ЦИКЛУ

Поки виконується умова повторення тіла циклу, оператор *while* працює так само, як і в звичайному варіанті, але як тільки умова повторення перестає виконуватися, потік виконання направляється по альтернативній гілці *else* – так само, як в умовному операторі *if*, вона виконається всього один раз.

```
i = 0
while i < 3:
   print (i)
   i += 1
else:
   print ("кінець циклу")
```

Результатом запуску даного коду буде:

```
0
1
2
кінець циклу
```

Запитання для самоконтролю

1. Як оголосити функцію у Python?
2. Що таке аргументи функції?
3. Як задати значення за замовчуванням для аргументів?
4. Як повернути значення з функції?
5. Що таке область видимості змінної у функції?
6. Як використовується інструкція if у Python?
7. Як оформити конструкцію if…elif…else?
8. Як працює цикл while у Python?
9. Як достроково вийти з циклу?
10. Як імпортувати та використовувати власний модуль?

Лекція 8. Списки цикли та бібліотеки Python 3

План

1. Списки, цикли for, вбудована довідка.
2. NumPy, SciPy і Matplotlib

1. Списки, цикли for, вбудована довідка

Списки

Масив – набір фіксованої кількості елементів, що розміщені в пам'яті комп'ютера безпосередньо один за одним, а доступ до них здійснюється за індексом (номер даного елементу в масиві).

В Python для реалізації масиву використовуються списки. ***Список*** – тип даних, що представляє собою послідовність певних значень, що можуть повторюватись. Але на відміну від масиву – кількість елементів у списку може бути довільною.

Списки – гетерогенна, змінювана структура даних, що може містити елементи різних типів, що перераховані через кому та заключені в квадратні дужки. Це дозволяє створювати структури будь-якої складності і глибини.

Списки служать для того, щоб зберігати об'єкти в певному порядку, особливо якщо порядок або вміст можуть змінюватися. Можна змінювати список, додати в нього нові елементи, а також видалити або перезаписати існуючі. Можна змінити кількість елементів у списку, а також самі елементи. Одне і те ж значення може зустрічатися в списку кілька разів.

Список є об'єктом, тому може бути присвоєний змінній.

```
Int_list=[1, 2, 5, 8]
```

Отримаємо:

```
[1, 2, 5, 8]
```

Список можна створити з нуля або більше елементів, розділених комами і вкладених у квадратні дужки:

```
empty_list = [ ]
number_list = [1, 2, 3, 4, 5]
week_days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
print(empty_list)
print(number_list)
print(week_days)
```

Отримаємо:

```
[]
[1, 2, 3, 4, 5]
[‘Monday’, ‘Tuesday’, ‘Wednesday’, ‘Thursday’, ‘Friday’]
```

Крім того, за допомогою функції ***list()*** можна створити порожній список:

```
another_empty_list = list()
print (another_empty_list)
```

Отримаємо:

```
[]
```

Функція ***list()*** перетворює інші типи даних в списки.

```
# Рядок перетвориться в список, що складається з односимвольних рядків
q = list('cat')
print(q)
```

Отримаємо:

```
['c', 'a', 't']
```

Звернення до елементу

Список містить різні дані, звертатися до яких можна через ім’я списку та вказавши зміщення необхідного елементу:

```
letters_list = ['a', 'b', 'c']

print(letters_list)          # Виведення всього списку
print(letters_list[0])       # Виведення першого елементу списку
print(letters_list[1])       # Виведення другого елементу списку
print(letters_list[2])       # Виведення третього елементу списку
print(letters_list[-1])      # Виведення останнього елементу списку
print(letters_list[-2])      # Виведення передостаннього елементу списку
```

Результатом запуску даного коду буде:

```
a
b
c
c
b
```

Зсув повинен бути коректним значенням для списку – воно являє собою позицію, на якій розташовується присвоєне раніше значення. Якщо вказати позицію, яка знаходиться перед списком або після нього, буде згенеровано виняток (помилка).

```
letters_list = ['a', 'b', 'c']
print(letters_list[3])
```

Отримаємо:

```
Traceback (most recent call last):
 File "G:\lab.py", line 9, in <module>
   print(letters_list[3])
IndexError: list index out of range
```

За аналогією з отриманням значення списку за допомогою його зміщення можна змінити це значення:

```
letters_list = ['a', 'b', 'c']
print (letters_list)
letters_list[2] = 'C' print (letters_list)
```

Отримаємо:

```
['a', 'b', 'c']
['a', 'b', 'C']
```

Отримання елементів за допомогою діапазону зсувів

Можна отримати зі списку підпослідовність, використавши зріз списку:

```
letters_list =['a', 'b', 'c', 'd', 'e']
print(letters_list[0:2])    # Виведення елементів починаючи з 1-го до 2-го
print(letters_list[::2])    # Кожен непарний елемент
print(letters_list[::-2])   # Всі елем. з останнього зі зміщенням вліво на 2:
print(letters_list[::-1])   # Інверсія списку
```

Отримаємо:

```
['a', 'b']
['a', 'c', 'e']
['e', 'c', 'a']
['e', 'd', 'c', 'b', 'a']
```

Методи списків

Для додавання елементів в кінець списку – використовують метод ***append()***.

```
letters_list =['a', 'b', 'c', 'd', 'e']
print(letters_list)

letters_list.append('f')

print(letters_list)
```

Результатом запуску даного коду буде:

```
['a', 'b', 'c', 'd', 'e']
['a', 'b', 'c', 'd', 'e', 'f']
```

Можна об’єднати один список з іншим за допомогою методу ***extend()***.

```
letters_list =['a', 'b', 'c', 'd', 'e']
others_list = ['g', 'h', 'i']
print(letters_list)
print(others_list)

letters_list.extend(others_list)
print(letters_list)
```

Результатом запуску даного коду буде:

```
['a', 'b', 'c', 'd', 'e']
['g', 'h', 'i']
['a', 'b', 'c', 'd', 'e', 'g', 'h', 'i']
```

Можна також використовувати оператор +=:

```
letters_list =['a', 'b', 'c', 'd', 'e']
others_list = ['g', 'h', 'i']
print(letters_list)
print(others_list)

letters_list += others_list
print(letters_list)
```

Результат буде аналогічний.

Цикл for

В Python ітератори часто використовуються оскільки вони дозволяють проходити структури даних, не знаючи, наскільки ці структури великі і як реалізовані. Можливо пройти по послідовності таким чином:

```
numbers = [1, 2, 3, 4]
i = 0
while i <=len(numbers)-1:
    print(numbers[i])
     i+=1
```

Результатом запуску даного коду буде:

```
1
2
3
4
```

Однак існує більш характерний для Python спосіб вирішення цього завдання:

```
for element in numbers:
    print(element)
```

Результатом запуску даного коду буде:

```
1
2
3
4
5
```

Списки подібні до *numbers* є одними з ітераційних об’єктів в Python поряд з рядками, кортежами, словниками та деякими іншими елементами.

Ітераційні об’єкти – це ті, вміст яких можна перебрати в циклі. Ітерування по кортежу або списку повертає один елемент за раз. Ітерування по рядку повертає один символ за раз:

```
word = 'cat' for letter in word:
    print(letter)
```

Результатом запуску даного коду буде:

```
c
a
t
```

Цикл *for* дозволяє перебирати всі елементи вказаної послідовності (список, кортеж, рядок). Цикл спрацює рівно стільки разів, скільки елементів знаходиться в послідовності.

for ЗМІННА in ПОСЛІДОВНІСТЬ:
ТІЛО_ЦИКЛУ

Ім’я змінної в яку на кожному кроці буде розміщено елемент послідовності обирається довільно.

Оператори *break* та *continue* працюють так само як і в циклі *while*. Блок *else* дозволяє перевірити чи виконався цикл *for* повністю якщо ключове слово *break* не було викликане, то буде виконаний блок *else.* Це корисно, якщо потрібно переконатися в тому, що попередній цикл виконався повністю, замість того щоб рано перерватися:

```
numbers = []
for number in numbers:
    print('Цей список має деякий елемент', number)
    break
else:
# Відсутність переривання означає, що елемент відсутній

    print('Елементів немає в списку, чи не так?')
```

Результатом запуску даного коду буде:

```
'Елементів немає в списку, чи не так?'
```

В циклах використання блоку *else* може здатися нелогічним. Можна розглядати цикл як пошук чогось, в такому випадку *else* буде викликатися, якщо нічого не було знайдено.

Ітерування за кількома послідовностям за допомогою функції zip()

Щоб паралельно ітеруватися за кількома послідовностям одночасно можна використати функцію *zip()*:

```
days = ['Monday', 'Tuesday', 'Wednesday']
fruits = ['banana', 'orange']
drinks = ['coffee', 'tea', 'beer']
for day, fruit, drink in zip(days, fruits, drinks):
     print(day, ": drink", drink, "eat", fruit)
```

Результатом запуску даного коду буде:

```
Monday : drink coffee eat banana
Tuesday : drink tea eat orange
```

Функція *zip()* припиняє свою роботу, коли виконується найкоротша послідовність. Один зі списків (fruits) виявився коротшим за інші, тому результат було виведено лише для двох елементів.

Функція *zip()* – ітератор, який повертає кортежі, що складаються з відповідних елементів аргументів-послідовностей.

2. NumPy, SciPy і Matplotlib NumPy

Ми будемо широко використовувати структури масивів, які містяться в пакеті **numpy** . Ці масиви використовуються в багатьох пакетах Python, які використовуються в обчислювальній науці, аналізі даних і графічному аналізі (у пакетах, таких як **scipy** і **matplotlib**).

Масиви Numpy — це тип високоструктурованого списку, який можна використовувати для виконання звичайних числових і матричних обчислень.

Хоча є винятки, масиви numpy — це особливий тип списку з обмеженим розміром і з усіма записами одного типу.

Масиви Numpy дозволяють легко налаштовувати вектори, матриці та тензорні об’єкти вищого порядку для подальших маніпуляцій.

Важливо те, що Python оптимізував обробку масивів numpy для дуже швидкого та ефективного виконання обчислень.

Загалом, вони дуже зручні для обчислювальної фізики.

Щоб створити масиви numpy, нам спочатку потрібно імпортувати модуль numpy наступним чином:

```
import numpy as np
```

Перший оператор імпортує numpy as, перейменовує його на np. Ви можете перейменувати будь-який модуль таким чином, і це звичайне перейменування для numpy. Тоді окремі методи та об’єкти в numpy можна називати наступним чином: np.sin(), np.pi тощо.

Другий оператор імпортує все, що стосується numpy, і дозволяє вам просто використовувати sin(), pi тощо.

Перше твердження є кращим:

Він зберігає весь numpy у своєму власному *просторі імен,* тому ви завжди знаєте, що використовуєте функцію numpy замість своєї власної функції, і уникаєте конфліктів з іншими реалізаціями тих самих функцій в інших пакетах (scipy, sympy, math.. .)

Це стандартний метод у науковому співтоваристві Python. Будь-яка документація, яку ви знайдете в Інтернеті, ймовірно, використовуватиме import numpy as np.

Matplotlib

Побудова фігур є важливою частиною використання Python для дослідження фізики та навчання. Основний пакет побудови графіків на Python називається **matplotlib.** Дивна назва походить від того факту, що інтерфейс побудови змодельований за (невільною) обчислювальною програмою **matlab** . Matplotlib еволюціонував з моменту його створення та вплинув на код для багатьох нових пакетів для малювання, таких як **bokeh** , **plotly** , **ggplot** (версія Python).

Matplotlib можна імпортувати кількома способами.

```
import matplotlib
import pylab
import matplotlib.pyplot
```

Перший імпорт (matplotlib) імпортує весь пакет, другий імпортує лише основний інтерфейс побудови графіка (pyplot), третій імпортує (pylab) спеціальний пакет, який включає позначення (pyplot) і числові методи (numpy). *Зауважте, що використання PyLab зазвичай не рекомендується.* Одним із поширених способів імпорту matplotlib і numpy є використання

```
import numpy as np
import matplotlib.pyplot as plt
```

Це зробить побудову доступною як префікс plt, а numpy – як np. Найпростішою командою plot є:

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.show()
```

Цей код просто відобразить елементи списку [1, 2, 3, 4]проти елементів іншого списку [1, 4, 9, 16]. Залежно від вашої конфігурації Spyder або Jupyter, останньою командою у вашому кресленні може бути show(), яка активує вікно креслення.

Scipy

Scipy або *Scientific Python* є останнім пакетом у цій частині підручника. scipy — це набір функцій для виконання базового наукового

програмування та аналізу даних. Це також екосистема для багатьох інших наукових пакетів і забезпечує послідовний інтерфейс для функцій і дозволяє уникнути дублювання. Наприклад, scipy містить перетворення Фур'є (scipy.fftpack) і числове інтегрування (scipy.integrate). Він також надає основні функції для машинного навчання (scikit-learn) і обробки зображень (scikit-image).

Наприклад, щоб інтегрувати список чисел за допомогою scipy, ви можете використати функцію, яка називається trapz(для *trapezoid* , тип інтеграції, який використовується) з integrateпакета.

```
import scipy.integrate as integrate
result = integrate.trapz([0, 1, 2, 3, 4, 5])
print(result)
# 12.5
```

або якщо ви хочете інтегрувати sin(x) від 0 до π Ви можете використовувати функцію quad (скорочення від *квадратура*).

```
import numpy as np
import scipy.integrate as integrate
result = integrate.quad(np.sin, 0, np.pi)
print(result)
# (2.0, 2.220446049250313e-14)
```

Тут quad береться назва функції (наприклад np.sin*,)* і межі інтеграла, але без точок даних. Потім Scipy намагається інтегрувати функцію та повертає два значення. Перше значення – відповідь, друге значення – оцінка помилки у відповіді.

Оптимізація

Іншим поширеним використанням scipy є оптимізація за допомогою scipy.optimize пакета. Це пакет, який використовується для підгонки даних до наукових моделей або підгонки значень до поліномів. Одним із способів підгонки даних є використання curve_fit функції, яка приймає щонайменше три аргументи

- Функція , яку ви хочете встановити.
- Набір *незалежних даних.*
- *Набір* залежних даних .

Дані, які ви надаєте, curve_fit можуть бути зібрані, наприклад, під час експерименту. Ви також можете надати curve_fit додаткову допомогу, якщо ваша функція особливо складна. Наприклад

- Ви можете надати початкові припущення для своєї функції в ключовому слові p0.
- Ви можете вказати відносні чи абсолютні помилки *залежних даних* за допомогою ключового слова sigma(і absolute_sigma*)*.

Запитання для самоконтролю

1. Як створити список у Python?
2. Як додати та видалити елемент списку?
3. Як перебрати список за допомогою циклу for?
4. Як отримати довжину списку?
5. Як працює вбудована функція help()?
6. Які основні функції містить бібліотека NumPy?
7. Для чого використовується SciPy?
8. Як створити графік за допомогою matplotlib?
9. Як працюють масиви NumPy?
10. Як зчитати дані з CSV-файлу у Python?

ЗМІСТОВИЙ МОДУЛЬ 3. ОСНОВИ СТАТИСТИКИ

Лекція 9. Основні задачі математичної статистики.

План

1. Поняття середнього, моди та медіани, кореляція даних.
2. Середня квадратична, гранична та відносна похибки.

1. Поняття середнього, моди та медіани, кореляція даних

Виконання геодезичних робіт пов'язане з виконанням вимірювань різних величин - відстаней, перевищень, кутів тощо. Вимірювання можуть виконуватися безпосереднім порівнянням величини з одиницею міри (прямі вимірювання) і за допомогою її обчислень як функції інших безпосередньо виміряних величин (непрямі вимірювання).

Досвід показує, що результати вимірювань завжди містять деякі похибки. Це проявляється, наприклад, у разі багаторазового вимірювання однієї й тієї самої величини - одержувані результати завжди дещо різняться між собою, а отже, неминуче відрізняються від істинного значення вимірюваної величини, тобто містять похибки вимірювань.

Похибки поділяють на грубі, систематичні та випадкові. Вимірювання, що містять грубі похибки, викликані недбалістю спостерігача або несправністю приладів, виявляють і відкидають, бракують.

Систематичні похибки, які під час повторних вимірювань залишаються постійними або змінюються за певним законом, намагаються виключити, вводячи в результати вимірювань поправки, юстуючи (регулюючи) прилади, застосовуючи правильну методику вимірювань.

Випадкові похибки під час повторних вимірювань змінюються випадковим чином. Вони неминучі й виключити їх із результатів вимірювань неможливо, тому під час виконання й обробки вимірювань прагнуть лише послабити їхній вплив. Шляхи до такого послаблення вказує теорія помилок (похибок) вимірювань. Надалі будемо вважати, що результати вимірювань вільні від грубих і систематичних похибок і містять тільки випадкові.

Похибкою Δ вимірювання називають відхилення результату вимірювання від істинного значення вимірюваної величини:

$$\Delta = l - X,$$

де l - результат вимірювання;

X - істинне, точне значення вимірюваної величини.

У переважній більшості випадків випадкові похибки, будучи наслідком дії багатьох чинників (недосконалості приладів, змін в умовах вимірювань, недосконалості органів чуття спостерігача), підпорядковуються так званому закону нормального розподілу. При цьому малі за абсолютною величиною похибки трапляються частіше за великі, позитивні похибки трапляються так само часто, як і негативні, а середнє арифметичне з випадкових похибок під час зростання їхнього числа прагне до нуля.

У статистиці **середнє**, **мода** та **медіану** — називають мірами центральної тенденції. Вони показують загальні характеристики розподілу даних за певною змінною, дозволяють виявити одне значення (або кілька значень — якщо мода в розподілі не одна, але про це детальніше згодом), що описує весь розподіл. Можна також сказати, що середнє, мода та медіана — це окремі значення що представляють весь набір даних, типові для всіх значень у групі.

Міри центральної тенденції потрібні з наступних міркувань:

1. Щоб отримати загальну картину розподілу. Ми не можемо запам’ятати кожен факт, що стосується сфери дослідження.
2. Щоб отримати чітку картину щодо досліджуваної сфери для розуміння та отримання потрібних висновків.
3. Щоб отримати чіткий опис групи в цілому та мати змогу порівнювати дві або більше груп у термінах типової «поведінки».

Середнє (Mean)

Найвідомішою мірою центральної тенденції — і найбільш вживаною в повсякденному побуті — є середнє, або ж просте середнє, або ж арифметичне середнє (arithmetic mean) — просто середнє значення ряду даних.

Для його обчислення досить скласти разом всі значення в розподілі, і поділити на кількість спостережень. В Екселі чи Google Spreadsheets для цього є функція MEAN. Є різні математичні способи підрахунки середнього, але в усіх сучасних електронних таблицях та спеціальних програмних пакетах для роботи з даними і статистикою є ця функція, тож ми не будемо зупинятися на математичних викладках.

Є певні загальні правила для використання середнього, зокрема:

1. Середнє — це «центр тяжіння» розподілу, і кожне значення дає внесок у визначення середнього значення, коли поширення значень є симетричними довкола центральної точки.

2. Середнє значення більш стабільне, ніж медіана чи мода. Тому, коли потрібно знайти найбільш стабільну міру центральної тенденції, використовують середнє.

Переваги середнього:

1. Середнє визначене дуже жорстко, тому не виникає питань чи нерозуміння щодо його значення та суті.

2. Це найбільш поширена міра центральної тенденції, оскільки її легко зрозуміти.

3. Середнє легко підрахувати.

4. Враховує всі значення розподілу.

Обмеження чи недоліки середнього:

1. На значення середнього впливають екстремальні значення (відомий іронічний жарт про «середню температуру по лікарні»).

2. Часом середнім є значення, що не присутнє в розподілі.

3. Часом результатом можуть бути абсурдні значення. Наприклад, маємо 41, 44, та 42 учнів у 5а, 5б та 5в класах якоїсь школи. Виходить, що середня кількість учнів у 5 класах школи – 42,3(3). А так не буває.

Медіана (Median)

Медіану можна визначити як точку на ряді розподілу (впорядкований набір значень змінної для різних спостережень — наприклад від найменшого до найбільшого значення) — до цієї точки розташовано половина всіх значень, і після цієї точки теж половина значень. Тобто, медіана, це значення, що ділить впорядкований ряд навпіл. Якщо кількість значень непарна, то береться одне зі значень — те, що стоїть у розподілі рівно по центру.

Коли значень парна кількість, то беруть два центральні значення, і знаходять їхнє середнє.

Для чого використовують медіану?

1. Коли потрібно знайти точну середню точку, точку на «півдорозі» від найменшого значення до найбільшого.

2. Коли екстремальні значення впливають на середнє — медіана є найкращою мірою центральної тенденції.

3. Медіану використовують коли потрібно, щоб певні значення впливали на центральну тенденцію, але все, що про них відомо — що вони «нижче» або «вище» медіани

Переваги медіани:

1. Легко вирахувати та зрозуміти.
2. Для підрахунку медіани не потрібні всі значення в розподілі.
3. Екстремальні значення розподілу не впливають на медіану.
4. Її можна визначити і для «відкритих» категорій / класів інтервалів.

Обмеження медіани:

1. Вона не так жорстко визначена як середнє, оскільки її значення не так вираховується, як знаходиться (серед значень в розподілі).
2. Не враховує всі спостереження (значення для всіх спостережень).
3. З медіаною потім не можна робити алгебраїчні перетворення так, як із середнім.
4. Потребує впорядкування значень або класів інтервалів у висхідному чи спадному порядку.
5. Часом медіаною може бути значення, не присутнє у самому розподілі.

Мода (Mode)

Третя міра центральної тенденції — це **мода** — значення, що найчастіше зустрічається в розподілі. Як правило, вона представляє найбільш типове значення. На моду ніколи не впливають екстремальні значення в розподілі, а впливають – екстремальні частоти значень, наскільки часто те чи інше значення змінної зустрічається в розподілі.

Мода використовується:

1. Коли нам треба швидка і приблизна міра центральної тенденції.
2. Коли потрібна міра центральної тенденції, що має бути типовим значенням.

Переваги моди:

1. Мода показує найбільш поширене значення в розподілі.
2. На моду не впливають екстремальні значення – так як на середнє.

3. Моду можна визначити для відкритих інтервалів / категорій.
4. Допомагає аналізувати якісні дані.
5. Моду можна виявити просто побудувавши графік розподілу чи стовпчасту діаграму.

Обмеження:

1. Не включає до визначення / розрахунку всі спостереження розподілу, а лише концентрацію частот.
2. Подальші алгебраїчні перетворення неможливі – на відміну від середнього.
3. Буває важко визначити моду у випадку багатомодального чи бімодального розподілу

Порівняння змінних і кореляція

Найбільш наглядний приклад показати зв'язок між двома кількісними змінними – це діаграма розсіювання (рис. 9.1). На відміну від гістограм, які ми розглядали раніше – під час аналізу одномірних розподілів, на осі у показують не частоту того чи іншого значення змінної по осі x, а значення іншої змінної. Крапка на діаграмі означає одночасно значення двох змінних для одного спостереження («рядок» в таблиці даних).

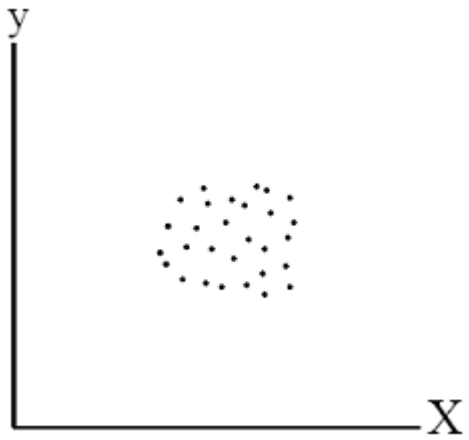


Рис. 9.1. Діаграма розсіювання

У кореляції є дві властивості – сила і напрям. Сила кореляції визначається числовим значенням, а напрям – тим, чи кореляція позитивна чи негативна.

Позитивна кореляція: обидві змінні міняються у тому ж напрямі. Тобто, якщо одна змінна зростає, друга зростає теж. Якщо одна спадає, то друга спадає так само. Наприклад, рівень освіти – скільки років людина

навчалася (в нормальних країнах) та річний заробіток корелюють між собою позитивно.

Негативна кореляція: змінні рухаються у протилежних напрямках. По мірі того, як одна змінна спадає, інша росте, і навпаки. Наприклад – кількість годин, проведених людиною уві сні та кількість годин неспання – корелюють негативно (що очевидно – чим більше спиш – тим менше часу лишається на яві, і навпаки) (рис. 9.2).

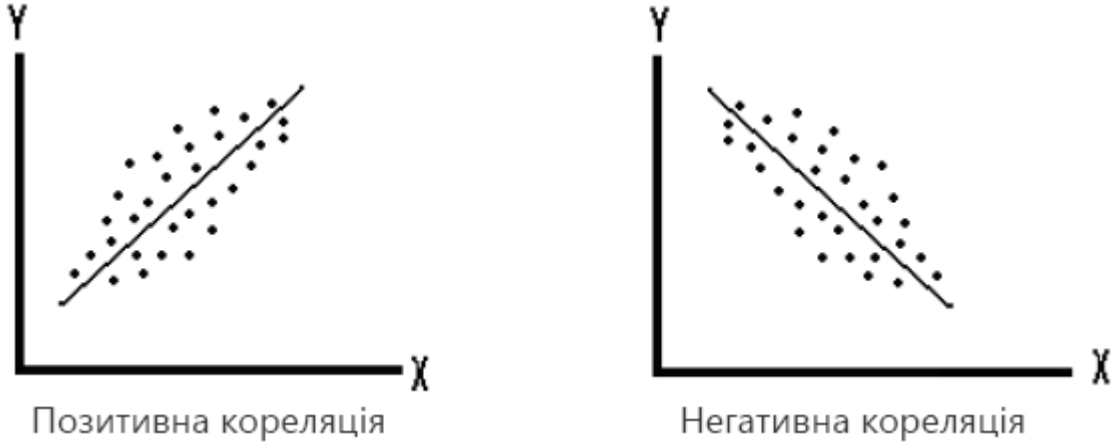


Рис. 9.2. Приклад позитивної і негативної кореляції змінних

Коефіцієнт кореляції показує ступінь, до якого дві змінні пов'язані (наскільки спільно чи подібно змінюються їх значення для різних спостережень) – тобто якої сили між ними може бути зв'язок. Значення коефіцієнта кореляції може бути від -1.0 до 1.0. Якщо вирахувана кореляція більша за 1 або менша за -1 – значить десь у підрахунках сталася помилка, адже 1 – означає абсолютну пряму (позитивну) кореляцію, а -1 – абсолютну зворотню (негативну) кореляцію.

Як підраховується коефіцієнт кореляції? Дорівнює сумі добутків відхилень, поділеній на добуток їх стандартних відхилень

Що означає, коли ми кажемо, що між двома змінними нема кореляції? Це означає, що між двома змінними немає прямого зв'язку. Наприклад, немає прямої кореляції між розміром взуття та зарплатою. Тобто, великі значення розміру взуття мають такі ж шанси зустрітися серед людей з високою зарплатою, як із низькою.

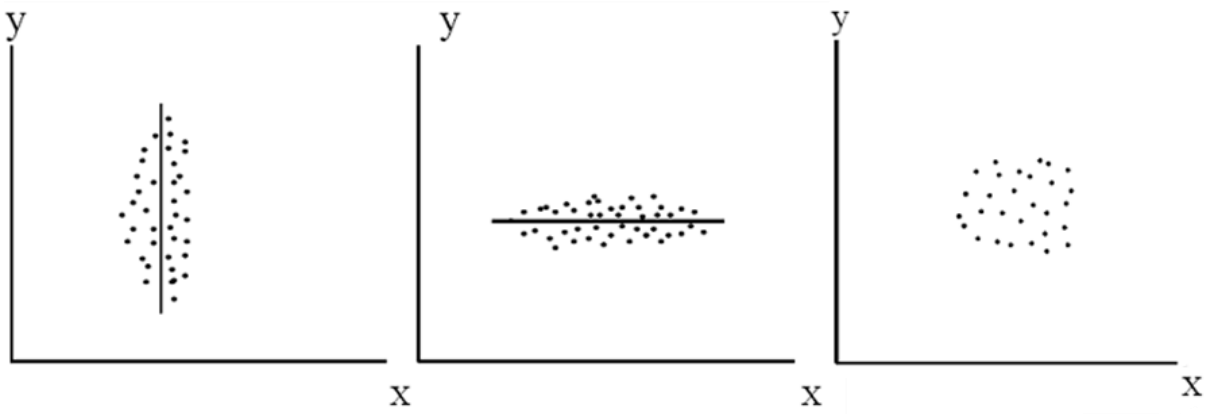


Рис. 9.3. Випадки дуже малої або відсутньої кореляції змінних

Кореляція і причинно-наслідковий зв'язок. Навіть якщо дві змінні виглядають пов'язаними між собою, це не значить, що одна спричинила іншу. Класичний приклад – це кореляція між ростом злочинності та споживанням морозива протягом літніх місяців у США. Дві змінні є пов'язані між собою, але жодне явище не є причиною іншого. Насправді, обидва явища спричинені підвищенням температури повітря, а не одне одним.

Важливо також пам'ятати, що кореляція – це міра лінійного зв'язку. При цьому, кореляція не говорить нам, яка змінна впливає на яку – кореляція лише показує наявність зв'язку, але впливу. Вимірюючи кореляцію, не можна сказати – це А впливає на Б, чи Б впливає на А.

Діаграма розсіяння для двох змінних може виглядати, наприклад, так:

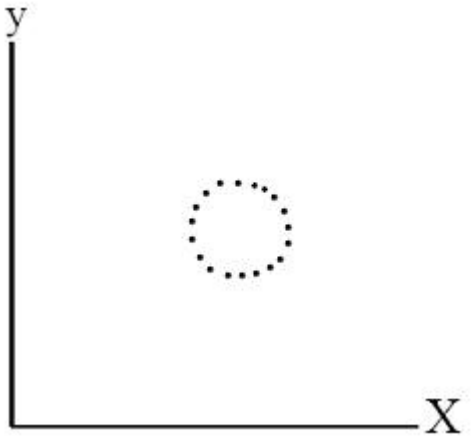


Рис. 9.4. Діаграма розсіяння для двох змінних

Для цих двох змінних кореляція буде дорівнювати нулю. Але це ще не означає, що зв'язку між змінними немає — просто він може бути не лінійним.

2. Середня квадратична, гранична та відносна похибки

Вимірювання, що виконуються, в однакових умовах, однаковими приладами, вважають рівноточними. У результаті багаторазових вимірювань одержувані значення вимірюваної величини можуть бути як більшими, так і меншими за її істинне значення.

Оцінка величини допущеної похибки є неодмінною частиною кожного вимірювання, оскільки дає змогу судити, наскільки отриманий результат близький до істинного значення величини. Вимірювання прийнято вважати точним і закінченим за умови зазначення допущеної помилки. Точність вимірювань характеризується середньою квадратичною похибкою вимірювання (СКП), яка визначається формулою Гауса:

$$m = \sqrt{\frac{\Delta_1^2 + \Delta_2^2 + \ldots + \Delta_n^2}{n}} = \sqrt{\frac{\left[\Delta^2\right]}{n}},$$

де Δ1, Δ2, ...Δn - випадкові похибки вимірювань.

Величина D = т носить назву дисперсії. Ця формула дає точне значення середньої квадратичної похибки за нескінченно великого числа п вимірювань. За малої кількості вимірювань, унаслідок випадкового характеру похибок Δі, обчислене за формулою значення т має свої похибки. Наближено середню квадратичну похибку визначення СКП можна оцінити за формулою

$$m_m \approx \frac{m}{\sqrt{2n}}$$

Формулою визначення СКП користуються під час дослідження точності геодезичних приладів і методів вимірювань, коли відомо істинне значення вимірюваної величини.

За нормального розподілу і досить великого числа вимірювань: 68,3% усіх похибок за своїм модулем не перевищують середньої квадратичної похибки т. Похибки, що перевищують 2m, зустрічаються рідко, а більші за 3m - ще рідше; при цьому 95,5% усіх похибок

за модулем менше 2m і 99,7% похибок менше 3m. Потроєнуту середню квадратичну похибку називають граничною:

$$\Delta пр = 3m.$$

Вимірювання, що містять помилки, більші за граничну, бракують.

Середня квадратична похибка дає характеристику точності вимірювання за абсолютною величиною і виражається в одиницях

вимірюваної величини. Для оцінки ступеня точності вимірювань користуються відносними похибками, що дорівнюють відношенню похибки до вимірюваної величини, яка виражається у вигляді дробу з одиницею в чисельнику:

$$\frac{m}{l} = \frac{1}{N}.$$

Запитання для самоконтролю

1. Що таке середнє значення і як його обчислити?
2. Як визначити моду та медіану?
3. Що таке кореляція і які її типи існують?
4. Як побудувати графік розсіювання?
5. Як знайти коефіцієнт кореляції?
6. Що таке середня квадратична похибка?
7. Як обчислити граничну похибку?
8. Як обчислити відносну похибку?
9. Як перевірити точність даних?
10. Які статистичні функції є у бібліотеках Python?

Лекція 10. Похибки, оцінка точності вимірів

План

1. Види похибок і правила округлення
2. Систематичні похибки.
3. Випадкові похибки.
4. Нормальний розподіл.
5. Оцінка точності функцій виміряних величин
6. Обробка результатів низки рівноточних вимірювань однієї величини

1. Види похибок і правила округлення

Похибки можна розділити на такі групи:

- Випадкова похибка – викликається великою кількістю причин у кожному вимірі (приклад – розкид між результатами титрування)
- Систематична похибка – обумовлені недосконалістю методу вимірювань (прилади, домішки у реактивах тощо).
- Грубі промахи - пов'язані з помилками експериментатора (неправильне читання показань приладу тощо).

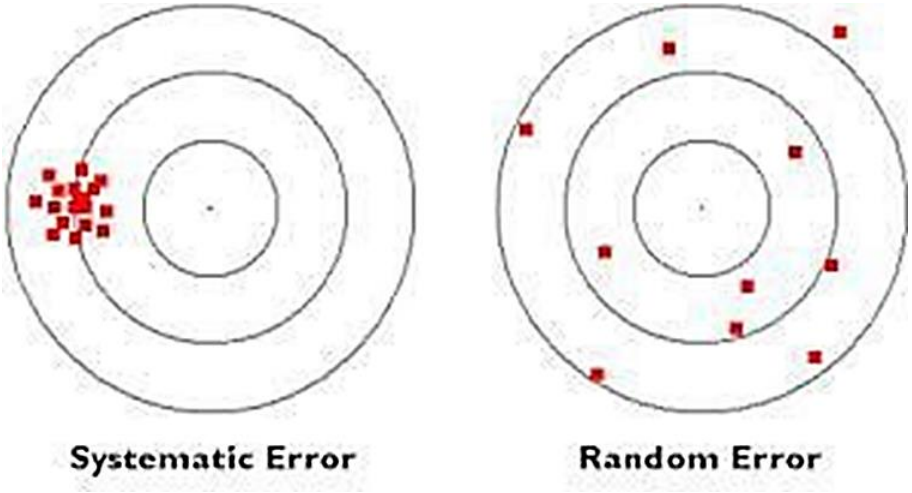


Рис. 10.1. Візуальна інтерпретація схематичної та випадкової похибка

Абсолютна похибка -різниця між істинним та виміряним значенням

$$\Delta x = x_{true} - x_{meas}$$

Відносна похибка – співвідношення абсолютної похибки до загального значення величини змінної.

$$x = \Delta x / x .$$

Правила округлення

Значущі цифри - всі цифри даного числа від першої зліва, не рівної нулю, до останньої справа.

Приклади:

- 123- 3 значущих цифра
- 0.012 - 2 значущих цифри
- 6.022*1023 - 4 значущих цифри
- 5*103 - 1 значна цифри. АЛЕ: 5000 - 4 значущих цифри!

Округлення до N-горозряду:

- Якщо N+1 - ий розряд < 5 -то відкинути всі цифри після N-го розряду
- Якщо N+1 -ий розряд ≥ 5 – то збільшити N-ий розряд на 1 і відкинути всі цифри після N-горозряду

Приклади:

- 123 -> 120
- 0.0458 -> 0.05
- 1.95 -> 2.0

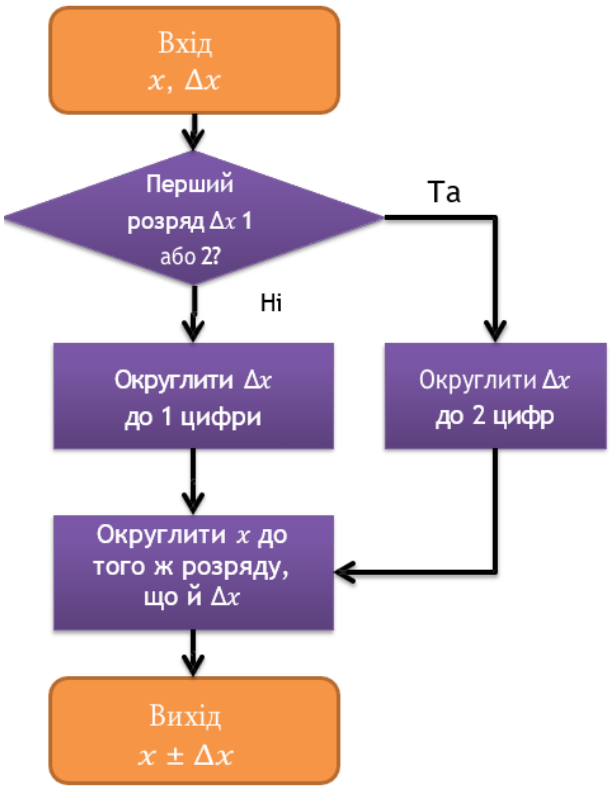


Рис. 10.2. Алгоритм округлення чисел

Приклад:

- $53216 \pm 348 \quad \rightarrow 5.32 \pm 0.03 \cdot 10$
- $0.0322 \pm 0.012 \rightarrow 3.2 \pm 1.2 \quad \cdot 10\text{-}2$
- $12.482 \pm 0.973 \rightarrow (12.5 \pm 1.0)$

Не можна округляти:

1. Проміжні обчислення(втрата точності)
2. Коефіцієнти регресії, отримані МНК (вони корельовані один з одним).

2. Систематичні похибки

Систематичні похибки— це похибки, які виникають постійно або за однакових умов під час вимірювань або обчислень. Вони мають певну закономірність і впливають на результат у визначеному напрямку (завжди збільшують або зменшують значення).

Основні джерела систематичних похибок:

- Інструментальні — пов'язані з неточностями приладів (наприклад, невірно відкалібрований інструмент).
- Методологічні — пов’язані з неправильним вибором або застосуванням методу вимірювання.
- Особистісні — помилки, що виникають через людський фактор (неправильне зчитування показників).
- Зовнішні фактори — зміни температури, тиску тощо, що постійно впливають на результати.

Як систематичні похибки впливають на математичні операції:

- Додавання і віднімання: Якщо всі вимірювання мають однакову систематичну похибку, то вона просто додається або віднімається від результату. Наприклад: Якщо температура постійно вимірюється з похибкою +2°C, то сума декількох вимірювань також матиме цю додаткову похибку.
- Множення і ділення: Систематична помилка масштабується разом із результатом. Наприклад: Якщо довжина постійно завищена на 5%, площа, яка обчислюється як добуток довжин, буде завищена приблизно на 10%.
- Піднесення до степеня: Помилка збільшується пропорційно степеню. Наприклад: Якщо помилка в довжині складає 1%, то при піднесенні довжини до квадрату помилка буде приблизно 2%.

Методи зменшення впливу систематичних похибок:

1. Калібрування приладів — регулярна перевірка та налаштування вимірювальних інструментів.
2. Метод порівнянь – вимірювання відносно відомих стандартів.
3. Застосування різних методів вимірювання – для перевірки результатів.
4. Корекція результатів – внесення поправок на відому систематичну похибку.

Складання систематичних похибок

Якщо величина отримана в результаті складання двох або більше вимірювань, систематична похибка результату дорівнює сумі систематичних похибок окремих вимірювань:

$$\Delta S = \Delta A + \Delta B,$$

де:

- ΔA і ΔB – систематичні похибки окремих вимірювань,
- ΔS– загальна систематична похибка.

Приклад 1. Якщо довжини двох відрізків виміряні з похибками $\Delta A=+2$ і $\Delta B=+3$, то загальна похибка виміряної суми довжин буде:

$$\Delta S=2+3=+5 \text{ мм}$$

Приклад 2. Вимірювання $A=50$ ($\Delta A=+2$), вимірювання $B=30$ ($\Delta B=+1$), сума величин:

$$A + B = 50 + 30 = 80$$

Сумарна похибка: $\Delta рез = 2 + 1 = +3$
Результат: 80 ($\Delta=+3$).

Віднімання систематичних похибок

При відніманні двох величин їх систематичні похибки також підсумовуються за модулем:

$$\Delta S = \Delta A + \Delta B$$

(оскільки навіть якщо знаки похибок різні, вони впливають на результат у бік зростання невизначеності).

Приклад. Якщо відстань між двома точками виміряна двома методами:

- перший дає похибку $\Delta A = +2$ мм,
- другий $\Delta B = -1.5$ мм,

то загальна систематична похибка буде:

$$\Delta S = |2| + |(-1.5)| = 3.5 \text{ мм}.$$

Систематичні помилки можуть накопичуватися при багаторазових вимірюваннях, що призводить до значних відхилень у кінцевому результаті. Тому в геодезії їх намагаються:

- компенсувати шляхом використання приладів з автоматичною корекцією,
- усувати за допомогою методів калібрування,
- враховувати через математичне коригування результатів.

Множення і ділення систематичних помилок

Для множення і ділення систематичні помилки перетворюються на **відносні похибки** (у відсотках), які потім додаються.

Формула для множення/ділення:

$$\frac{\Delta_{\text{рез}}}{R} = \frac{\Delta A}{A} + \frac{\Delta B}{B}$$

де R - результат операції.

Приклад. Вимірювання 1: A=50 (ΔA=+2, вимірювання 2: B=30 (ΔB=+1)

1. Знаходимо відносні похибки:

$$\frac{\Delta A}{A} = \frac{2}{50} = 0.04 = 4\%$$

$$\frac{\Delta B}{B} = \frac{1}{30} \approx 0.0333 = 3.33\%$$

2. Додаємо відносні похибки:

$$\Delta_{\text{відн}} = 4\% + 3{,}33\% = 7{,}33\%$$

3. Обчислюємо результат множення:

$$R = A \times B = 50 \times 30 = 1500$$

4. Знаходимо абсолютну похибку результату:

Результат: 1500 (Δ=+110)

3. Випадкові помилки

Випадкові помилки (рандомні похибки) виникають через неконтрольовані фактори, які по-різному впливають на результати вимірювань кожного разу. На відміну від систематичних, вони не мають постійного напряму (тобто можуть бути як у бік збільшення, так і зменшення).

Приклади джерел випадкових помилок:

- Коливання температури або тиску.
- Недосконаліть людського зору під час зчитування даних.
- Мікроколивання в електронних приладах.

При виконанні математичних операцій випадкові похибки обробляються за правилами комбінування похибок**.** Основна відмінність від систематичних похибок — використання квадратичного складання.

Складання випадкових помилок

Якщо результат геодезичного вимірювання отриманий як сума кількох величин, його похибка визначається як **середня квадратична помилка (СКП) окремих випадкових помилок**:

$$\sigma S = \sqrt{\sigma A^2 + \sigma B^2},$$

де σA, σB – середні квадратичні похибки окремих вимірювань,

- σS – сумарна похибка.

Приклад:

Якщо довжини двох відрізків виміряні з випадковими похибками σA = 3 мм та σB = 4 мм, то загальна похибка виміряної суми довжин буде:

$$\sigma S = \sqrt{3^2 + 4^2} = \sqrt{9 + 16} = \sqrt{25} = 5 \text{ мм}.$$

Віднімання випадкових помилок

При відніманні двох величин випадкові похибки також підсумовуються за кореневим середнім квадратів:

$$\sigma S = \sqrt{\sigma A^2 + \sigma B^2}$$

Приклад. Якщо ми маємо дві відстані з похибками σA = 2 мм та σB = 5 мм, то випадкова похибка їх різниці буде:

$$\sigma S = \sqrt{2^2 + 5^2} = \sqrt{4 + 25} = \sqrt{29} = 5.39 \text{ мм}$$

Отже, при складанні та відніманні випадкових похибок їх ніколи не віднімають алгебраїчно, а завжди обчислюють через СКП.

Випадкові помилки підпорядковуються **нормальному розподілу** та можуть бути додатними або від’ємними. При **складанні або відніманні величин** випадкові похибки **не додаються арифметично**, а підсумовуються **за квадратами**. Чим більше вимірювань, тим **точніше можна оцінити випадкову похибку** та зменшити її вплив на кінцевий результат.

Ця методика є основою при аналізі похибок у геодезичних розрахунках, наприклад, при визначенні координат точок, висотних відміток або довжин ліній.

4. Нормальний розподіл (розподіл Гауса)

Нормальний розподіл (розподіл Ґауса) – розподіл ймовірностей випадкової величини, що характеризується густиною ймовірності.

$$f(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right),$$

де μ — математичне сподівання, σ^2— дисперсія випадкової величини.

Параметр σ σ також відомий, як стандартне відхилення. Розподіл із μ = 0 та $\sigma^2 = 1$ називають стандартним нормальним розподілом.

Центральна гранична теорема стверджує, що нормальний розподіл виникає тоді, коли дана випадкова величина являє собою суму великого числа незалежних випадкових величин, кожна з яких відіграє незначну роль в утворенні всієї суми. Наприклад, відстань від влучення снаряду гармати до цілі при великій кількості пострілів характеризується саме нормальним розподілом.

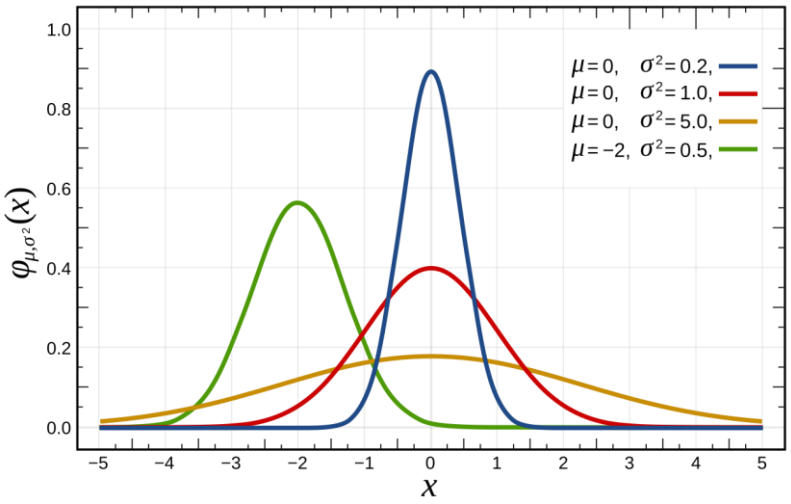


Рис. 10.3. Нормальний розподіл випадкової величини

Нормально розподілена випадкова величина позначається так: ξ~N(μ,σ2.).

Нормальний розподіл із функцією густини f(x) (математичним сподіванням μ і стандартним відхиленням σ >0) має такі властивості:

- Він симетричний відносно точки x = μ, яка одночасно є модою, медіаною і середнім значенням розподілу.
- Розподіл є одномодальним: його перша похідна додатна при x<μ, від'ємна при x>μ, і дорівнює нулю лише в точці x=μ.
- Площа, що обмежена під кривою і віссю x дорівнює одиниці.

5. Оцінка точності функцій виміряних величин

Нехай для визначення значення деякої величини *i* виміряно інші величини *x*, *y*, *z*..., з якими визначувана величина пов'язана функціональною залежністю

$$u = f(x, y, z,...).$$

Якщо середні квадратичні похибки виміряних величин дорівнюють *mx*, *my*, *mz*..., то середня квадратична похибка визначуваної величини виражається формулою

$$m_u^2 = \left(\frac{\partial f}{\partial x}\right)^2 m_x^2 + \left(\frac{\partial f}{\partial y}\right)^2 m_y^2 + \left(\frac{\partial f}{\partial z}\right)^2 m_z^2 + \dots \ . \tag{4}$$

У табл. 10.1 наведено конкретні варіанти формули (4) для окремих випадків функції *i*.

Таблиця 10.1

Функція	Середня квадратична помилка
U = k·x, де k – безпомилкове постійна величина; x – незалежна змінна	$mu = k \cdot mx$
U = x + y	$m_u = \sqrt{m_x^2 + m_y^2}$
U = x-y	$m_u = \sqrt{m_x^2 + m_y^2}$
U = k1·x+k2·y+…+kn·w	$m_u = \sqrt{k_1^2 \cdot m_x^2 + k_2^2 \cdot m_y^2 + \dots + k_n^2 \cdot m_w^2}$
U = x·y	$m_u = \sqrt{m_x^2 \cdot y^2 + m_y^2 \cdot x^2}$
U = x/y	$m_u = \sqrt{\left(m_x^2 \cdot y^2 + m_y^2 \cdot x^2 \right) / y^4}$

Приклад. Довжину лінії *d* виміряно по частинах. Відрізок, d_1 = 215,46 м виміряно із середньою квадратичною похибкою m_1 = 0,11 м, а відрізок d_2 = 113,25 з m_2 = 0,04 м. Необхідно оцінити точність вимірювання довжини всієї лінії $d = d_1 + d_2$ = 328,71 м.

Рішення. Середня квадратична похибка визначиться за формою (2) табл. 10.1:

$$m_d = \sqrt{m_1^2 + m_2^2} = \sqrt{0{,}11^2 + 0{,}04^2} = 0{,}12 \text{ м}.$$

Відносна похибка результату вимірювань:

$$\frac{m_d}{d} = \frac{0{,}12}{328{,}71} = \frac{1}{2700}.$$

6. Обробка результатів низки рівноточних вимірювань однієї величини

Рівноточними називають вимірювання, виконані приладами однакової точності, рівним числом прийомів, в однакових умовах, однаково досвідченими спостерігачами. Якщо для визначення деякої величини X виконано низку рівноточних вимірювань і отримано результати l1, l2,..., ln, то за остаточне значення приймають величину, що обчислюється як середнє арифметичне з усіх результатів:

$$L = \frac{l_1 + l_2 + \ldots + l_n}{n} = \frac{[l]}{n}. \qquad (6)$$

За кількості вимірювань n, що прагне до нескінченності, середнє арифметичне L прагне до істинного значення X вимірюваної величини.

Середня квадратична похибка кожного окремого вимірювання визначається за формулою:

$$m = \sqrt{\frac{[\upsilon^2]}{n-1}}, \qquad (7)$$

де $\upsilon = l - L$ - відхилення від середнього арифметичного.

Середньоквадратичну похибку арифметичної середини $m_L = M$ визначають за формулою

$$M = \frac{m}{\sqrt{n}}. \qquad (8)$$

Крім оцінки точності виконаних вимірювань, формула (8) може бути використана для розрахунку кількості вимірювань, необхідної для досягнення заданої точності M. Записуючи формулу (8) відносно n, отримаємо

$$n = \frac{m^2}{M^2}. \qquad (9)$$

Приклад. Під час дослідження мірної стрічки одна й та сама лінія виміряна 15 разів. Результати вимірювань наведено в табл. 3. Обчислити середнє значення довжини лінії й оцінити точність вимірювань.

Рішення. Обчислимо середнє значення виміряної лінії, для чого складемо і розділимо на 15 десяті та соті частки метрів.

$$L = 156 + \frac{3,75}{15} = 156 + 0,25 = 156,25 \text{ м.}$$

Обчислимо відхилення υ і складемо їх. Їхня сума має бути близькою до нуля (з точністю округлень). Знайдемо суму квадратів відхилень і за формулою (7) визначимо середню квадратичну похибку одного вимірювання:

$$m = \sqrt{\frac{340}{15-1}} = 4,9 \text{ см.}$$

Похибка результату вимірювань (середнього арифметичного) визначимо за формулою (8):

$$M = \frac{4,9}{\sqrt{15}} = 1,3 \text{ см.}$$

Відносна похибка одного вимірювання:

$$\frac{m}{L} = \frac{4,9}{15625} = 1:3200.$$

середнього арифметичного

$$\frac{M}{L} = \frac{1,3}{15625} = 1:12000.$$

Запитання для самоконтролю

1. Які види похибок існують у вимірюваннях?
2. Які правила округлення застосовуються при поданні результатів?
3. Що таке систематичні похибки?
4. Як виявити та усунути систематичні похибки?
5. Що таке випадкові похибки?
6. У чому суть нормального розподілу похибок?
7. Що таке розподіл Гауса і як він використовується?
8. Як оцінити точність функції виміряних величин?
9. Яка формула використовується для обробки рівноточних спостережень?
10. Як зменшити вплив похибок на результат?

СПИСОК РЕКОМЕНДОВАНИХ ДЖЕРЕЛ

1. Васильєв О. М. Програмування мовою Python / О.М. Васильєв. – Тернопіль : Навчальна книга-Богдан, 2021. – 503 с.
2. Маттес Ерік. Пришвидшений курс Python : практичний, проєктно-орієнтований вступ до програмування / Ерік Маттес ; з англійської переклала Ольга Бєлова. – Львів : Вид-во Старого Лева, 2021. – 556 с.
3. Основи програмування. Python. Частина 1 [Електронний ресурс]: підручник для студ. спеціальності 122 “Комп’ютерні науки”, спеціалізації “Інформаційні технології в біології та медицині” / А. В. Яковенко. – Київ : КПІ ім. Ігоря Сікорського, 2018. – 195 с.
4. Інформатика. Комп’ютерна техніка. Комп’ютерні технології / під ред. проф. Пушкаря О.І. – Київ : ВЦ „Академія”, 2003. – 704 с.
5. Савченко, В. М. Системний аналіз та математичне моделювання у GNU Octave : навч. посіб. / В. М. Савченко, О. Б. Маций, О. В. Мнушка. – Харків: ХНАДУ, 2020. – 128 с.
6. В.Б. Хоцкіна, І.Н. Вдовиченко Робота в пакеті MATLAB: навч. посібник. – Кривий Ріг: Державний університет економіки і технологій, 2023. – 130 с.
7. . Теорія ймовірностей, математична статистика та імовірнісні процеси: навч. посіб. / Джавала Л.Л., Слюсарчук Ю.М., Хром’як Й.Я., Цимбал В.М. – Львів: Вид-во Львів. політехніки, 2015. – 364 с.
8. Системний аналіз складних систем управління: навч. посіб. /А.П. Ладанюк, Я.В. Смітюх, Л.О. Власенко та ін. – Київ : НУХТ, 2013. – 274 с.
9. Математика: алгебра та початки аналізу. Частина І: навч. посіб. / О.В.Левчук, Л.С.Яхно, В.М.Кобзар. – Вінниця: ВНАУ, 2019. – 320 с.
10. MATLAB Source Codes [Електронний ресурс]. – Режим доступу: https://www.mathworks.com/help/index.html?s_tid=CRUX_lftnav
11. Python Tutorial [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/python-programming-language-tutorial/?ref=outind
12. Tutorial University of Toronto Computational Physics [Електронний ресурс]. – Режим доступу: https://computation.physics.utoronto.ca/tutorials/
13. Learn Python [Електронний ресурс]. – Режим доступу: https://www.c-sharpcorner.com/learn/learn-python
14. GNU Octave Wiki [Електронний ресурс]. – Режим доступу: https://wiki.octave.org/GNU_Octave_Wiki

Навчальне видання

МЕДВЕДСЬКИЙ Юрій Вікторович

ПРОГРАМНІ КОМПЛЕКСИ ІНЖЕНЕРНИХ РОЗРАХУНКІВ

Конспект лекцій

Комп’ютерне верстання *А.П. Селівестрової*

Ум. друк. арк. 7,67. Обл.-вид. арк. 8,25
Електронний документ. Вид № 93/V-25

Виконавець і виготовлювач

Київський національний університет будівництва і архітектури
Проспект Повітряних Сил, 31, Київ, Україна, 03037

Свідоцтво про внесення до Державного реєстру суб’єктів
видавничої справи ДК № 808 від 13.02.2002 р.