Міністерство освіти і науки України

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

О.І. Толочко

ТЕОРІЯ АВТОМАТИЧНОГО КЕРУВАННЯ

Математичні методи та програмування в *MATLAB* для теорії керування

Навчальний посібник

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня бакаалвра
освітньою програмою «Електромеханічні системи автоматизації, електропривод та електромобільність»
спеціальності 141 «Електроенергетика, електротехніка та електромеханіка»,

Електронне мережне навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2025

Рецензент *Островерхов М.Я.,* доктор технічних наук, професор, завідувач кафедри теоретичної електротехніки

Відповідальний редактор *Ковбаса С.М.,* доктор технічних наук, доцент, завідувач кафедри автоматизації електромеханічних систем та електроприводу

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 7 від 8.05.2025 р.)
за поданням факультету електроенерготехніки та автоматики (протокол №11 від 28.04.2025 р.)
Електронне мережне навчальне видання Реєстр 24/25-493

Толочко О.І.
Теорія автоматичного керування. Математичні методи та програмування в *MATLAB* для теорії керування. [Електронний ресурс]: навч. посіб. для здобувачів ступеня бакалавра за освіт. програмою «Електромеханічні системи автоматизації, електропривод та електромобільність» спец. 141 «Електроенергетика, електротехніка та електромеханіка» / О.І. Толочко; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл). – Київ: КПІ ім. Ігоря Сікорського, 2025. – 296 с.

Навчальний посібник присвячено питанням застосування студентами спеціальності 141 «Електроенергетика, електротехніка та електромеханіка» математичних методів для розв‘язання інженерних задач в галузі теорії автоматичного керування, електротехніки та електромеханіки в середовищі системи програмування *MATLAB*. Посібник складається з 16 розділів, в яких викладені основи програмування в *MATLAB*, основи математичного моделювання в додатку Simulink пакету *MATLAB*, основи роботи з інструментами додатків *Extended Symbolic Toolbox* та *Control Toolbox* та математичні методи розв‘язання прикладних інженерних задач.

Навчальний посібник призначений для використання бакалаврами спеціальності 141 «Електроенергетика, електротехніка та електромеханіка» при вивченні дисципліни «Теорія автоматичного керування» та може бути використаним студентами навчальної освітньої програми «Електромеханічні системи автоматизації, електропривод та електромобільність» при вивченні дисциплін «Теоретичні основи електротехніки», «Моделювання електромеханічних систем», «Інтелектуальне керування та оптимізвція в електромеханічних системах» а також при виконанні курсових, бакалаврських та магістерських робіт.

АНГЛО-УКРАЇНСЬКИЙ СЛОВНИК ТЕРМІНОЛОГІЇ, ЩО ЗАСТОСОВУЄТЬСЯ В *MATLAB*

Англомовний термін (скорочення)	Переклад – пояснення
Desktop	Робочий стіл – головне вікно *MATLAB*
Workspace	Робочий простір – оперативна пам‘ять *MATLAB*
Folder	Папка
Tool	Інструмент
Toolbox	Ящик з інструментами – Папка з *m*-функціями певного призначення
Script-file	Файл-сценарій – головна програма
Function	Файл-функція
Built-in function	Вбудована функція – недоступна для перегляду
Keyword	Ключове слово – зарезервований термін, який не можна змінювати
Variable (var)	Змінна
Constant (const)	Констаната
Expression (expr)	Вираз
Statement (state)	Оператор
Command	Команда
Comment	Коментар
Character (char)	Символ
String (str)	Рядок символів
Number (num)	Число
Integer (int)	Ціле число
Array	Масив
List	Список
Input argument	Вхідний аргумент *m*-функції
Output argument	Вихідний аргумент *m*-функції – результат
Default	Значення параметру, що призначається при його відсутності за замовчанням
Prompt	Нагадування – запрошення до введення команди
Debug	Налагодження (програми)

ВСТУП

В теорії автоматичного керування (ТАК) та в її застосуваннях в електротехніці та електромеханіці існує безліч задач, що не можуть бути розв‘язані аналітичними методами або такий розв‘язок є занадто складним [1-7]. Вони потребують застосування чисельних математичних методів [8, 9], пов‘язаних з наявністю ітераційних циклів, або громіздких алгоритмів, які можуть бути успішно реалізовані тільки на ЕОМ при використанні сучасного програмного забезпечення.

До таких задач належать задачі математичного аналізу, задачі лінійної алгебри, операції зі степеневими поліномами, пошук визначених інтегралів, апроксимація нелінійних функцій, розв‘язання диференційних рівнянь, алгебраїчних та трансцендентних рівнянь, які широко використовуються в теорії автоматичного керування.

Наприклад, математичний опис лінійних динамічних об‘єктів подають у формі поліноміальних передавальних функцій або у вигляді вкторно-матричних алгебраїчних та диференційних рівнянь. Для аналізу таких систем необхідно знаходити корені алгебраїчних рівнянь, виконувати операції зі степеневими поліномами та операції лінійної алгебри. При частотному аналізі необхідно оперувати з комплексними числами, будувати різноманітні частотні характеристики. Якщо система має у своєму складі нелінійні елементи, характеристика вхід-вихід яких задана в табличному вигляді, то для їх математичного моделювання необхідно застосовувати апроксимацію або інтерполяцію. При дослідженні періодичних процесів треба вміти визначити їх гармонічний склад, щоб потім ефективно фільтрувати вищі гармоніки. При пошуку оптимальних траєкторій або оптимальних параметрів для систем керування електромеханічними об‘єктами необхідно вміти розв‘язувати нелінійні рівняння та їх системи.

Усі ці задачі можна вирішувати як за допомогою програм, написаних

будь-якою алгоритмічною мовою, що потребує від дослідника достатньо високої кваліфікації в галузі програмування та обчислювальної математики, так і за допомогою спеціалізованого програмного забезпечення, що дозволяє користувачу значно легше і скоріше отримувати результати у зручній формі завдяки наявності великої кількості різноманітних інструментів.

Серед спеціалізованих систем програмування однією з найбільш розповсюджених натепер є система *MATLAB* фірми *Mathworks* [10-15], що отримує у своєму складі об‘єктно-орієнтовану алгоритмічну мову, розвинутий графічний інтерфейс, засоби розв‘язання задач лінійної алгебри, математичного аналізу, оптимізації, обчислювальної математики, цифрової обробки сигналів, структурного математичного та віртуального фізичного моделювання, аналізу і синтезу систем автоматичного керування та багато інших інструментів (*Toolboxes*) [3-7, 16-22].

Для того, щоб скористуватися цими інструментами, треба спочатку ознайомитися з основами програмування в *MATLAB*: типами даних, операціями над даними, системою «допомоги», основними операторами, правилами синтаксису, графічними можливостями.

Особливістю пакета *MATLAB* є можливість застосування математичних методів не тільки шляхом написання програми на алгоритмічній мові, але й шляхом застосування блоків бібліотек додатку для структурного математичного моделювання *Simulink* [8, 23-25]. Оскільки цей додаток дуже широко використовується при моделюванні систем електроприводу, то в посібнику показано, як можна деякі практичні задачі розв‘язати обома способами.

У посібнику наведені приклади розв‘язання прикладних інженерних задач в галузі теорії автоматичного керування, електротехніки, електромеханіки, з використанням як стандартних інструментів *MATLAB,* так і власних функцій користувача. Приділено увагу діагностиці помилок та способам їх ліквідації,

методам визначення початкових наближень шуканих параметрів при застосуванні ітераційних методів їх уточнення.

Навички, надбані студентами при вивченні розділу «Автоматизація розрахунків та досліджень в середовищі пакету *MATLAB*» дисципліни «Теорія автоматичного керування», стануть їм в нагоді при вивченні дисциплін «Моделювання систем автоматичного керування», «Моделювання електромеханічних систем», «Адаптивне та робастне керування», «Інтелектуальне керування та оптимізація в електромеханічних системах».

1. ОСНОВИ РОБОТИ В *MATLAB*

1.1. Характеристика пакету

Систему програмування *MATLAB* використовують більш, ніж у 70 найвідоміших університетів світу, в т.ч. у Стенфордському, Каліфорнійському (США), Кембриджському (Англія), Кіото (Японія), Ейнтховенському технічному університеті (Нідерланди), в Масачусетському та Хельсінкському технологічних інститутах, а також в науково-дослідних центрах НАСА, в компаніях *Aerospace Corp., Boeing Aerospace, General Dynamics Corp., IBM, Lockheed, Siemens AG* та ін.

Історично *MATLAB* розроблявся фірмою *MathWorks* (США, Нейтік, штат Масачусетс) як діалогове середовище для матричних обчислень (*MATrix LABoratory*). Поступово пакет був оснащений гарною графічною системою, доповнений засобами лінійної алгебри від *LinPack*, символьної математики від *Maple*, підсилений засобами обчислювальної математики, додатками для структурного (*Simulink*) та віртуального фізичного моделювання електротехнічних, електромеханічних та електронних пристроїв (*SimPowerSystem*) та різноманітними інструментами (*Toolboxes*), призначеними для ефективного розв'язання спеціальних задач. Серед цих інструментів для фахівців у галузі електротехніки та енергетики найбільш інтересними є засоби для аналізу та синтезу систем керування лінійними і нелінійними об'єктами (*Control, Nonlinear, Robust, Predictive Toolboxes*), розв'язання задач оптимізації (*Optim*), ідентифікації (*Ident*), використання нейронних мереж (*Neuro*) та нечіткої логіки (*Fuzzy*), майстерня реального часу (*Real Time Workshop*).

Перша версія *MATLAB* була написана на Фортрані (*Cleve Moler*) на початку 60-х років. До 3-еї версії пакет працював під ОС *MS DOS*, починаючи з 4-ої – під *Windows*). Версії під *Windows* написані на *C* (автори: інтерпретатор – *Steve Bangert*, графіка – *Steve Kleiman*, більшість функцій – *John Little* і *Cleve Moler*). Може конвертувати файли, написані на Фортрані та *C*.

Отже, до складу *MATLAB* входять інтерпретатор команд, графічна оболонка, текстовий редактор з відладчиком, бібліотеки команд, компілятор, символьне ядро пакета *Maple*, математичні бібліотеки *MATLAB* на *C/C++*, генератор звітів и багатий інструментарій (*Toolboxes*).

Інтерфейс MATLAB відповідає сучасним канонам (див. рис. 1.1). Після запуску пакету на екрані монітору з'являється робочий стіл (***Desktop***), що складається із заголовку, головного меню, панелі інструментів і комбінованого вікна. Останнє отримує 4 панелі: ***Command Window*** (вікно команд), ***Command History*** (історія команд), ***Workspace*** (робочий простір – оперативна пам'ять), ***Current Folder*** (поточна папка).

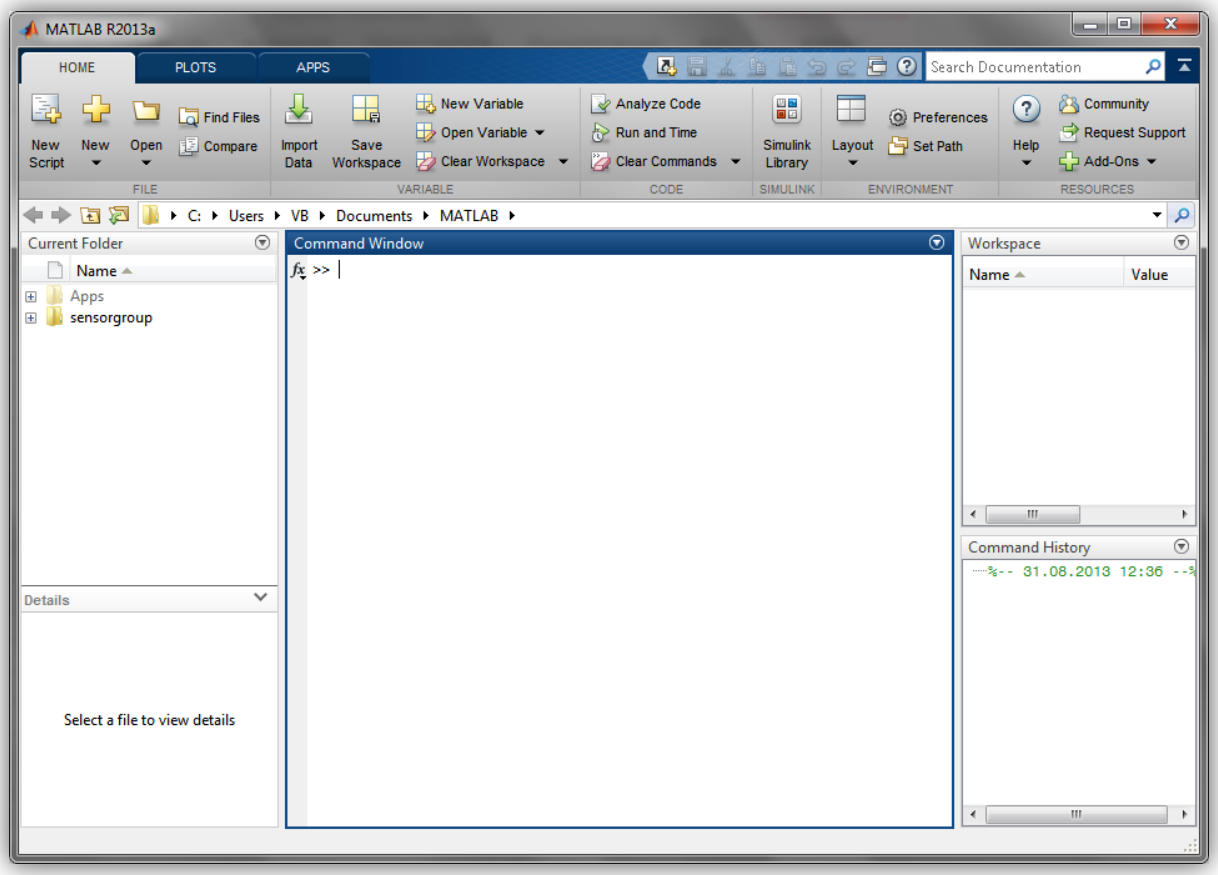


Рис. 1.1. Робочий стіл системи програмування *MATLAB 13a*

В *Command Window* набираються команди або оператори, які виконуються одразу після натискання клавіші *Enter*, видаються результати виконання цих команд, виводяться повідомлення системи.

Команди – це засоби (інструменти), що керують переферійним обладнанням. Оператори – це засоби, що здійснюють операції над данними.

Вікно *Workspace* відображує поточні змінні, та їх значення.

Вікно *Command History* зберігає всі команди, що набирає користувач у командному вікні.

Конфігурацію вікон на екрані та їх склад можна змінювати звичайними засобами, прийнятими у *Windows* та за допомогою команд меню *View*. Щоб відновити конфігурацію, прийняту за замовчанням, треба через функції меню пройти шлях: ***Desktop → Layout → Default.***

1.2. Режими роботи *MATLAB*

1.2.1. Режим командного інтерпретатора

Одним із найпростіших режимів роботи пакету *MATLAB* є режим командного інтерпретатора, в якому інструкції вводяться безпосередньо у вікні *Command Window* при наявності у командному рядку запрошення (*prompt*) «>>» натисканням клавіші *Enter*. При цьому виконується трансляція введеної команди у машинний код і її виконання без запам‘ятовування машинного коду (інтерпретація). Команди, розташовані в одному рядку, відокремлюються один від одного символами "," або ";". Якщо команда або оператор не поміщається в одному рядку, її можна продовжити в наступному рядку, використовуючи у якості оператору переносу три або більше точок поряд без пробілів ("…").

Деяка кількість введених команд запам‘ятовується в буфері, з якого їх можна викликати по черзі у зворотному п орядку клавішею "↑", та відображається у вікні *Command History*, з якого їх можна виконати повторно щигликом миші.

Після виконання введеної команди інтерпретатор готовий до прийому наступної команди, про що свідчить наявність у новому рядку командного вікна символу запрошення "»".

Командне вікно *MATLAB* при необхідності може бути очищено командою clc (*clear screen*).

Для видалення поточних змінних та їх значень (очистка робочого простору) необхідно виконати команду

clear

Щоб зберегти сеанс роботи в режимі командного інтерпретатора у файлі, можна перед початком сеансу виконати команду

diary *FileName*

або

diary (‘*FileName*’),

а в кінці сеансу – команду

diary off

В результаті таких дій у поточній папці буде створено текстовий файл-блокнот, в який будуть занесені як введені команди, так і результати їх виконання, за виключенням графічної інформації. У наступному сеансі можна вивести цей файл у вікно *Command Window* командою

type *FileName*

Вміст файлу можна скопіювати у буфер (^*c*) та включити (^*v*) у файл звіту **.doc* або **.docx*.

1.2.2. Програмний режим

Щоб зберегти послідовність операторів, що виконують певний алгоритм, у пам‘яті як єдине ціле, придатне для подальшого використання та редагування у вікні текстового редактора *edit* створюють файли, що зберігаються під маскою **.m* і називаються *m*-файлами. ***M-файли розподіляються на script-файли та m-функції. Script-файли – це головні програми, які у багатьох сучасних програмних продуктах називають «сценаріями».***

Перед формуванням програмних файлів треба створити власну папку і зробити її доступною для системи MATLAB. Для цього необхідно відкрити Провідник (*Explorer*) за шляхом *C:\Users\UserName\Documents\MATLAB*, де зберігаються документи пакету *MATLAB*, і створити за цим шляхом нову папку користувача, призначену для збереження власних файлів, наприклад, ivanov24.

Потім створену папку пиєднують до списку дозволених в MATLAB папок командою меню Set Path→Add Folder…→Save.

Для створення *script*-файлу треба клацнути мишею по віртуальній кнопці *New Script* у командному вікні MATLAB або виконати клавішну команду ^N або ввести з командного рядка команду edit. В усіх перелічених випадках на екрані з'явиться вікно текстового редактора, зображене на рис. 1.2.

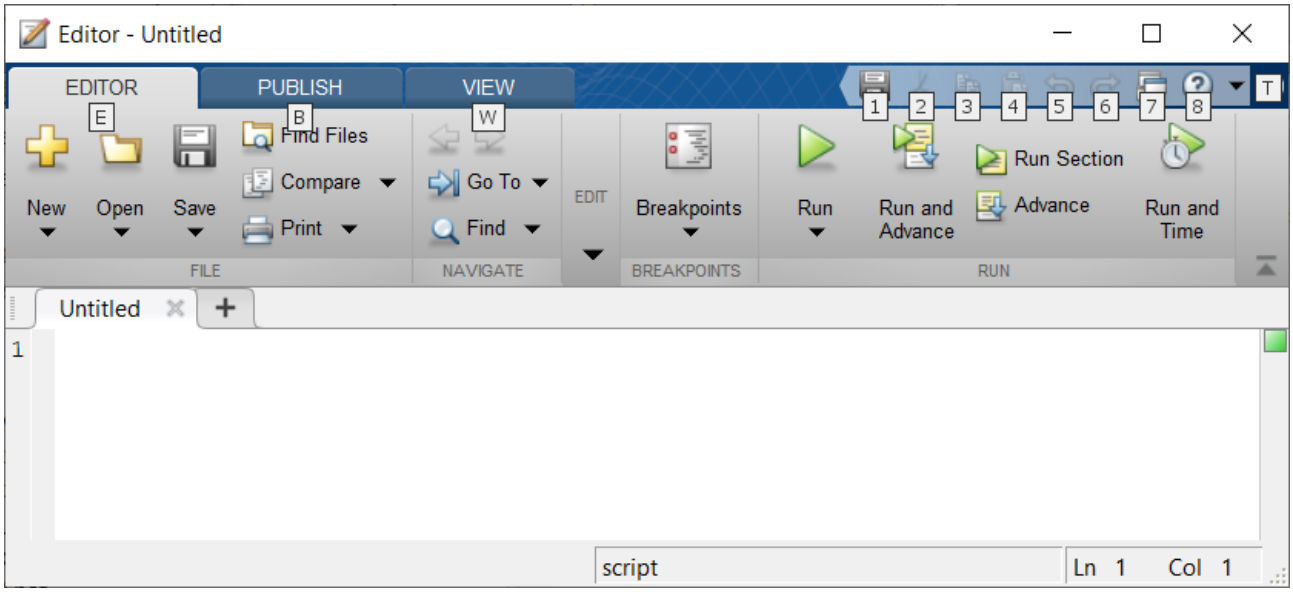


Рис. 1.2. Вікно текстового редактора

Введення інформації виконується за тими ж правилами, що і в командних рядках, а саме:

- в кожному рядку можна набрати одну (один) або декілька команд або операторів, що відокремлюються одне від одного комою «,» або крапкою з комою «;»; останній символ має сенс ставити тільки після оператору присвоєння з метою заборонити виведення результату цієї операції на екран після її виконання; після останнього оператора у рядку кому можна не ставити;
- перехід на наступний рядок виконується клавішею *Enter*;
- елементи операторів відокремлюються один від одного пробілами або спеціальними символами до яких належать знаки операцій, апострофи та

дужки; там де стоять спеціальні знаки можна вставляти довільну кількість пробілів;

- у будь-якому рядку можна записувати коментарі, ознакою початку яких є знак «%», а кінцем – кінець рядка;
- щоб розмістити один оператор у декількох рядках, у кінці кожного рядка ставлять три крапки «…»;
- у кінці *script*-файлу не треби ставити оператор end.

Після завершення набору *script*-файлу його необхідно зберегти у створеній заздалегідь папці клавішею *Save as*... та запустити на виконання клавішею *Run* або просто натиснути *F5* на клавіатурі, або набрати у командному рядку ім‘я файлу та натиснути клавішу *Enter*. Якщо у *script*-файлі будуть помилки, *MATLAB* повідомить про це у командному вікні з посиланням на конкретний рядок програми. Після усунення помилок в командне вікно (*Command Window*) будуть виведені результати виконання програми.

При створенні *script*-файлів дотримуйтесь таких рекомендацій:

- починайте файл з коментарів, у яких поясніть призначення програми, вкажіть своє прізвище і групу, наведіть за своїми міркуваннями іншу корисну інформацію, наприклад, номер лабораторної роботи, дату, тощо;
- до введення основних операторів доцільно набрати команди clc – очистити екрану, clear all – очистити пам‘ять, close all – закрити усі графічні фігури (якщо ви працюєте з графічними операторами), format compact – установити компактний стиль виведення інформації;
- якщо ви хочете бачити на екрані не тільки результати виконання операторів, але й самі оператори, скористайтеся командою echo on; відмінити її може команда echo off;
- доповнюйте програму коментарями, що пояснюють хід її виконання.

1.3. Файлова система пакету *MATLAB*

Серед основних папок системи *MATLAB* слід звернути увагу на такі папки:

- *bin* – командна папка, що утримує у сові файл запуску пакета *matlab.exe*;
- *simulink* – папка додатку для структурного математичного моделювання;
- *toolbox* – сукупність папок (*folder*) інструментів з файлами **.m*, що називаються *m*-функціями, призначеними для розв'язання різноманітних математичних задач, і одним *m*-файлом коментарів (*Contents.m*), в якому коротенько (в один рядок) описано призначення кожної функції даної папки.

Будь-яку функцію (крім вбудованих – *built-in function*) можна завантажити у текстовий редактор або вивести на екран командою

type *fun*,

де *fun* – ім'я функції.

Кожна функція починається з ***оператора заголовку функції***, який складається з ключового слова function та отримує інформацію про ім'я функції та імена вхідних і вихідних параметрів. Далі ідуть коментарі (*Comments*), складені за певними правилами. Перший рядок коментаря отримує коротеньку (в один рядок) інформацію про призначення функції. Ця інформація заноситься у файл *Contents.m* та виводиться при виконанні команди

help *folder*,

де *folder* – ім'я відповідної папки. Наступні рядки коментаря складаються з більш детальної інформації про функцію та її параметри. Вони виводяться на екран при виконанні команди

help *fun*

Перериває зв'язок коментарів з формуванням останньої команди пустий рядок, за яким починається ***тіло функції***, що складається з операторів (*Statements*) алгоритмічної мови *MATLAB*. Ці оператори виконують діагностику вхідних даних і реалізують алгоритм перетворення вхідних аргументів (*inputs arguments*), у вихідні (*outputs arguments*). Розділ операторів може

супроводжуватись коментарями, що пояснюють призначення окремих операторів або фрагментів програмного модуля.

Отже кожна *m*-функція має такий формат:

function [<*list of outputs arguments*>] = *fun* (<*list of inputs arguments*>)
% <Contents string for **help folder**>
%< Text for **help fun**>
%<…>
%< *Text for **help fun***>

<*Statements*>

У вбудованих функціях розділ операторів є недосяжним для перегляду та коригування користувачем.

Для знайомства з можливостями пакету корисно ознайомитися з основними папками інструментів. Серед них в першу чергу слід звернути увагу на такі:

general – команди загального призначення;
ops – математичні операції та спеціальні символи;
lang – конструкції алгоритмічної мови;
helptools – функції допомоги;
elfun – елементарні математичні функції;
elmat – елементарні матриці та вектори та маніпуляції над ними;
datafun – аналіз даних та перетворення Фур‘є;
matfun – операції лінійної алгебри;
graph2d – двовимірна графіка;
graph3d – тривимірна графіка;
graphics – обробка графіків (допоміжні графічні функції);
specgraph – спеціальна графіка;
polyfun – операції над степеневими поліномами;
funfun – функції над функціями;
optimfun – функції оптимізації.

Щоб дізнатися про розташування будь-якого файла з ім‘ям *FileName* у файловій системі *MATLAB* достатньо виконати команду

which *FileName*

Наприклад, при виконанні команди

which polyfun

отримаємо

C:\Program Files\MATLAB\R2018a\toolbox\matlab\polyfun\polyval.m

1.4. Імена змінних, константи, формати виведення інформації

Імена змінних (ідентифікатори) в середовищі *MATLAB* повинні складатися з латинських букв, цифр та знаку підкреслення і починатися з букви, наприклад, x, X, Alpha1, y_min, c2dim, Number_of_Iter. Кількість символів у імен може бути довільною але *MATLAB* аналізує тільки перші символи кількістю 31 або 63 (в залежності від версії пакету) і розрізняє заголовні та прописні букви.

Арифметичні константи розподіляються на такі:

- цілі (-16, +5, 283);
- дійсні у форматі з фіксованою крапкою, тобто у природній формі, коли ціла частина відокремлюється від дробової «крапкою» (25.6, -138.54);
- дійсні у форматі з плаваючою точкою, коли мантиса числа *M* відокремлюється від його порядку *p* символом «e», тобто числу у формі $M\times10^{p}$ відповідає константа Mep. Наприклад, константи 0.25e-8 та -35e12 відображають числа 0.25×10^{-8} та -35×10^{12} відповідно;
- комплексні у трьох попередніх формах з позначенням уявної частини символами i або j (2.5-5i, -0.2+1e-3j).

MATLAB має ряд ***зарезервованих констант і змінних*** (табл.1.1).

Таблиця 1.1

Ім’я	**Значення**
i, j	sqrt(-1) – уявна одиниця

pi	число π, обчислюється як 4*atan(1) або imag(log(–1))
eps	машинна точність – 2.2204e-16
realmax	модуль максимального дійсного числа – 1.7977e+308
realmin	модуль мінімального дійсного число – 2.2251e-308
intmax	максимальне ціле число – 2147483647
intmin	мінімальне ціле число – -2147483648
namelengthmax	максимальна довжина імені – 63
Inf	Infinity – нескінченність (number/0)
NaN	Not a Number – невизначеність (0/0, inf/inf, inf-inf, 0^inf, ...)
ans	answer – результат останньої операції
end	найбільше значення індексу масиву

Для виведення чисел на екрані є різні формати. Потрібний формат може бути визначений в меню (*File/Preferences*) або за допомогою команди format. Найбільший інтерес мають формати, подані у табл. 1.2.

Таблиця 1.2

Формат	**Подання**
short e	Число в експоненціальній формі (з плаваючою крапкою) з мантисою із 5 цифр і показником степені із 3 цифр
short	Число у природній формі (з фіксованою крапкою) з 4 цифрами після крапки або в форматі *short e*, якщо формат *short* не підходить для відображення значення числа (за замовчанням)
long e	Число в експоненціальній формі з мантисою із 16 цифр і показником степені із 3 цифр
long	Число у природній формі з 16 цифрами після крапки або в форматі *long e*
rat	Число у вигляді раціонального дробового числа
loose	Інформація виводиться просторо – з пропуском одного рядка між рядками (за замовчанням)
compact	Інформація виводиться щільно (без пропуску рядків)

Строкові константи (рядки символів) складаються з будь-яких символів, розташованих між апострофами (‘Hello!’, ‘2+3=5?’, ’Напруга, В’).

1.5. Операція присвоєння

Символом операції присвоєння є "=". Операція ***явного присвоєння*** має формат

VarName = expr

де VarName – ім‘я змінної, expr – вираз. При виконанні такого оператора виконується обчислення виразу у правій частині оператора, а результат заноситься в комірку пам‘яті, виділеної для змінної, ім‘я якої записано у лівій частині. Результат виводиться на екран. Якщо оператор присвоєння закінчується символом «;», то виведення результату не виконується.

Вираз (*Expression*) являє собою сукупність констант, змінних та звернень до функцій, роз‘єднаних знаками операцій. Усі змінні, застосовані у правій частині у момент виконання оператора повинні мати значення. Окремим випадком виразу може бути константа, змінна або звернення до функції, наприклад,

```
» y = sind(30) % Натискаємо на Enter – бачимо результат
y =
   0.5000
»x = 4+3; % Натискаємо на Enter – результат не виводиться
»x          % Натискаємо на Enter – бачимо значення введеної змінної
x =
    7
```

Усі змінні зі своїми значеннями зберігаються у *робочому просторі* (*Workspace*) середовища *MATLAB*, тобто в його оперативній пам‘яті. Дані з можна видалити командою clear, а можна і зберегти у файлі **.mat* (за замовчанням *matlab.mat*) командою load.

При так званому ***неявному присвоєнні*** ліва частина оператора та знак «=» в конструкції оператора відсутній. У такому випадку результат операції присвоюється зарезервованій змінній з ім‘ям ans (від *answer* – відповідь), наприклад,

```
» 5.32*pi
ans =
  16.7133
» sin(5)
ans =
```

```
  -0.9589
```

Присвоєння змінним комплексних значень виконується аналогічно:

```
» z = 3+4*i;  x = 2.5-6*j;  y = 4*exp (i*pi/8);
» x
x =
  2.5000 - 6.0000i
» y
y =
  3.6955 + 1.5307i
```

При роботі зі скалярними арифметичними даними у виразах використовуються операції додавання або уточнення знаку +, віднімання або зміни знаку -, множення *, ділення / та підвищення у степінь ^, а також операції порівняння >, <, >=, <=. ==, ~= і логічні операції |, &. ~, які будуть розглянуті пізніше. Перелік всіх операцій та призначення спеціальних символів можна побачити при виконанні команди

help ops

Із функцій, що можуть входити у математичні вирази, у першу чергу розглянемо елементарні функції, що знаходяться у папці elfun. Для знайомства з ними наберемо команду

help elfun

В результаті отримуємо інформацію, з якої випливає призначення таких функцій:

- ***тригонометричні з аргументом у радіанах***: sin (x) – $\sin x$, cos (x) – $\cos x$, tan (x) – $\mathrm{tg}\,x$, cot (x) – $\mathrm{ctg}\,x$, sec (x) – $\sec x$, csc (x) – $\mathrm{cosec}\,x$;
- ***тригонометричні з аргументом у градусах (degree)***: sind (x), cosd (x), tand (x), cotd (x), secd (x), cscd (x);
- ***зворотно тригонометричні зі значенням у радіанах***: asin (x) – $\arcsin x$, acos (x) – $\arccos x$, atan (x) – $(-\pi/2)\le\mathrm{arctg}\,x\le\pi/2$, atan2 (y,x) – $(-\pi)\le\mathrm{arctg}\,y/x\le\pi$, acot (x) – $\mathrm{arcctg}\,x$, asec (x) – $\mathrm{arcsec}\,x$, acsc (x) – $\mathrm{arccosec}\,x$;

- ***зворотно тригонометричні зі значенням у градусах***: asind (x), acosd (x), atand (x), atand2 (y,x), acotd (x), asecd (x), acscd (x);
- ***гіперболічні***: sinh (x) – $\mathrm{sh}x=(e^x-e^{-x})/2$, cosh (x) – $\mathrm{ch}x=(e^x+e^{-x})/2$, tanh (x) – $\mathrm{th}x=(e^x-e^{-x})/(e^x+e^{-x})$, coth (x) – $\mathrm{cth}x=(e^x+e^{-x})/(e^x-e^{-x})$, sech (x) – $\mathrm{sech}x=2/(e^x-e^{-x})$, csch (x) – $\mathrm{cosech}x=2/(e^x+e^{-x})$;
- ***експоненційні, логарифмічні та степеневі***: exp (x) – e^x, log (x) – $\ln x$, log10 (x) – $\lg x$, log2 (x) – $\log_2 x$, pow2 (x) – 2^x, sqrt (x) – $\sqrt{x}$, nthroot (x,n) – $\sqrt[n]{x}$;
- ***функції комплексних аргументів*** $z=x+jy=Ae^{j\varphi}$: abs (z) – $|z|=\sqrt{x^2+y^2}=A$, angle (z) – $\varphi=\mathrm{arctg}(y/x)$, real (z) $-x$, imag (z) $-y$, complex (x,y) – $z=x+jy$, conj (z) $-x-jy=Ae^{-j\varphi}$.
- ***округлювання та залишки***: fix (x) – округлювання у напрямку до 0 (дробова частина відкидається); floor (x) – округлювання у напрямку до $-\infty$ (до найближчого меншого цілого); ceil (x) – округлювання у напрямку до $+\infty$ (до найближчого більшого цілого); round (x) – округлювання до найближчого цілого; rem (x,y) – залишок від цілочислового ділення x на y; sign (x) – знакова функція (дорівнює +1 при x>0, 0 при x=0 та -1 при x<0).

До цього переліку можна додати ще декілька функцій цілочислових аргументів із папки specfun: factor (n) – $n!$; gcd (m,n) – найбільший загальний дільник (*Greatest Common Divisor*); lcm (m,n) – найменший загальний множник (*Least Common Multiple*).

1.6. Генерація векторів та матриць

Основним типом даних в *MATLAB* є матриця, тобто двовимірний масив, який можна подати у вигляді таблиці з декількома рядками та стовбцями. Окремим випадком матриці є одновимірні масиви (вектор-рядок та вектор-стовбець) та скаляри, тобто одиночні дані.

При поелементному формуванні векторов і матриць їх елементи поміщають у квадратні дужки [ ], елементи одного рядка відокремлюють один від одного комами «,» або пробілами «␣«, а рядки – крапкою з комою «;» або переводом рядку клавішею *Enter*, наприклад,

```
» x = [2 3 –8];              % Вектор-рядок
» y = [5; 6.5; -2.23; 0];    % Вектор-стовбець
» A = [1,2,3; 4 5 6];        % Матриця 2*3
» B = [1 2; 3 4; 5 6];       % Матриця 3*2
```

1.6.1. Спеціальні вектори

Вектори-рядки з рівномірним розподіленням елементів можна утворювати двома способами.

1) Оператори

VarName = InitValue : Step : EndValue

та

VarName = InitValue : EndValue

називаються ***діапазонами***. Вони створюють вектори-рядки за заданими значеннями першого InitValue та останнього EndValue елементів та різницею Step між двома сусідніми елементами. У другому випадку за замовчанням (*default*) Step=1. Наприклад,

```
» z = 2:5
z =
    2    3    4    5
» w=-1:0.5:1
w =
  -1.0000   -0.5000    0    0.5000    1.0000
```

Кількість елементів у рядку Number можна підрахувати за формулою:

Number = ((EndValue – InitValue) / Step)+1

Рядки з однаковою кількістю елементів можна поєднувати у матриці:

```
» c=[1:4; 2:5]
c =
    1    2    3    4
    2    3    4    5
```

2) Оператор

VarName = linspace (InitValue, EndValue, Number)

створюють рівномірно розподілені вектори-рядки за заданими значеннями першого InitValue і останнього EndValue елементів та їх кількістю Number. За замовчанням Number = 100. Оператори linspace зручно використовувати при завданні діапазону зміни аргументу при побудові графіків, наприклад,

```
» a=linspace(3,7,10)
a =
   3.0000    3.4444    3.8889    4.3333    4.7778    5.2222    5.6667    6.1111    6.5556
7.0000

>> x=linspace(0,2.5)
x =
  Columns 1 through 10
         0    0.0253    0.0505    0.0758    0.1010    0.1263    0.1515    0.1768    0.2020
0.2273
  Columns 11 through 20
    0.2525    0.2778    0.3030    0.3283    0.3535    0.3788    0.4040    0.4293    0.4545
0.4798
  Columns 21 through 30
    0.5051    0.5303    0.5556    0.5808    0.6061    0.6313    0.6566    0.6818    0.7071
0.7323
  Columns 31 through 40
    0.7576    0.7828    0.8081    0.8333    0.8586    0.8838    0.9091    0.9343    0.9596
0.9848
  Columns 41 through 50
    1.0101    1.0354    1.0606    1.0859    1.1111    1.1364    1.1616    1.1869    1.2121
1.2374
  Columns 51 through 60
    1.2626    1.2879    1.3131    1.3384    1.3636    1.3889    1.4141    1.4394    1.4646
1.4899
  Columns 61 through 70
    1.5152    1.5404    1.5657    1.5909    1.6162    1.6414    1.6667    1.6919    1.7172
1.7424
  Columns 71 through 80
    1.7677    1.7929    1.8182    1.8434    1.8687    1.8939    1.9192    1.9444    1.9697
1.9949
  Columns 81 through 90
    2.0202    2.0455    2.0707    2.0960    2.1212    2.1465    2.1717    2.1970    2.2222
2.2475
  Columns 91 through 100
```

2.2727 2.2980 2.3232 2.3485 2.3737 2.3990 2.4242 2.4495 2.4747 2.5000

У таких рядках крок Step між двома сусідніми елементами можна підрахувати за формулою:

$$Step = (EndValue - InitValue) / (Number - 1)$$

Вектори рядки, рівномірно розподілені впродовж логарифмічної осі, можна сформувати оператором

VarName = logspace (pInit, pEnd, Number)

який створює вектори-рядок кількістю Number елементів (за замовчанням Number=50), перший з яких дорівнює 10^{pInit}, останній – 10^{pEnd}, а інші розраховуються у такий спосіб, щоб відношення кожного наступного елементу до попереднього було константою 10^{pStep}:

$$VarName(i) / VarName(i-1) = 10^{pStep} = const,$$

$$lg(VarName(i)) - lg(VarName(i-1)) = pStep = const,$$

$$i = [2, 3, \ldots, Number]$$

$$pStep = (pEnd - pInit) / (Number - 1)$$

Наприклад,

```
>> w = logspace (-1,3,5)   % [10^(-1), 10^0, 10^1, 10^2, 10^3]
w =
   1.0e+03 *
    0.0001   0.0010   0.0100   0.1000   1.0000
>> pw=log10(w)
pw =
   -1    0    1    2    3
>> k = logspace (log10(2), log10(2^9), 9)  % [2^1, 2^2, 2^3, 2^4, 2^5, 2^6, 2^7, 2^8, 2^9]
k =
   2.0000    4.0000    8.0000   16.0000   32.0000   64.0000  128.0000  256.0000  512.0000
>> pk = log10(k)
pk =
   0.3010   0.6021   0.9031   1.2041   1.5051   1.8062   2.1072   2.4082   2.7093
>> omega = logspace(-1,2)    % вектор із 50 елементів: omega (1)=10^(-1),
                             %  omega (50)=10^2
omega =
```

```
  Columns 1 through 10
    0.1000    0.1151    0.1326    0.1526    0.1758    0.2024    0.2330    0.2683    0.3089
0.3556
  Columns 11 through 20
    0.4095    0.4715    0.5429    0.6251    0.7197    0.8286    0.9541    1.0985    1.2649
1.4563
  Columns 21 through 30
    1.6768    1.9307    2.2230    2.5595    2.9471    3.3932    3.9069    4.4984    5.1795
5.9636
  Columns 31 through 40
    6.8665    7.9060    9.1030   10.4811   12.0679   13.8950   15.9986   18.4207   21.2095
24.4205
  Columns 41 through 50
   28.1177   32.3746   37.2759   42.9193   49.4171   56.8987   65.5129   75.4312   86.8511
100.0000

>> p_omega=log10(omega)
p_omega =
  Columns 1 through 10
   -1.0000   -0.9388   -0.8776   -0.8163   -0.7551   -0.6939   -0.6327   -0.5714   -0.5102   -
0.4490
  Columns 11 through 20
   -0.3878   -0.3265   -0.2653   -0.2041   -0.1429   -0.0816   -0.0204    0.0408    0.1020
0.1633
  Columns 21 through 30
    0.2245    0.2857    0.3469    0.4082    0.4694    0.5306    0.5918    0.6531    0.7143
0.7755
  Columns 31 through 40
    0.8367    0.8980    0.9592    1.0204    1.0816    1.1429    1.2041    1.2653    1.3265
1.3878
  Columns 41 through 50
    1.4490    1.5102    1.5714    1.6327    1.6939    1.7551    1.8163    1.8776    1.9388
2.0000
```

1.6.2. Спеціальні матриці

Деякі ***спеціальні матриці*** створюються такими функціями:

zeros – матриця, у якої всі елементи дорівнюють 0;

ones – матриця, у якої всі елементи дорівнюють 1;

rand – матриця, складена із випадкових чисел з рівномірним розподілом;

randn – матриця, складена із випадкових чисел з нормальним розподілом.

Усі вони мають однаковий формат, який розглянемо на прикладі функції zeros:

zeros(n) – формує квадратну матрицю розміром $n \times n$;

zeros(m, n) – формує прямокутну матрицю розміром $m \times n$;

zeros(1, n) – формує вектор-рядок із n елементів;

zeros(m, 1) – формує вектор-стовбець із m елементів;

zeros(size(A)) – формує матрицю, такого ж розміру, як матриця A.

Наприклад,

```
» Z = zeros(2)
Z =
    0    0
    0    0
» zeros(1,3)
ans =
    0    0    0
» ones(size(Z))
ans =
    1    1
    1    1
» rand(2,3)
ans =
   0.8147    0.1270    0.6324
   0.9058    0.9134    0.0975
»v = randn(1,5)
v =
   3.0349    0.7254   -0.0631    0.7147   -0.2050
```

До функцій утворення спеціальних матриць та векторів, також належать

eye (n) – формує одиничну діагональну матрицю розміром n×n:

diag (A) – формує з елементів головної діагоналі матриці A вектор-стовбець;

diag (X) – формує діагональну матрицю, застосовуючи у якості головної діагоналі вектор X:

```
» E = eye(3)
E=
    1    0    0
    0    1    0
    0    0    1
» A = [1 2 3; 4 5 6; 7 8 9];
» V = diag(A)
V =
    1
```

```
     5
     9
» Av =  diag(V)
Av =
     1     0     0
     0     5     0
     0     0     9
```

До елементів масивів звертаються за ім‘ям масиву та списку індексів, поміщеному між круглими дужками, наприклад, C(2,3) – елемент 2-го рядка та 3-го стовбця матриці C, x(3) – 3-ій елемент вектору x. Останнє значення будь-якого індексу можна позначати як end: C(end,1) – останній елемент першого стовбця матриці C.

Використання замість одного з індексів символу «:» означає, що цей індекс перебігає усі свої значення. Це дозволяє звертатися до окремих рядків та стовбців матриць: H(:,3) – 3-ій стовбець матриці H, H(1:3,:) – перші 3 рядки матриці, H(:,[2,4]) – 2-ий та 4-ий стовбці матриці.

Пуста (порожня) матриця [ ] у правій частині оператору присвоєння знищує (вилучає) елементи, зазначені у лівій частині, наприклад, операція A(2,:)=[ ] викидає з матриці A другий рядок.

1.7. Контрольні питання та завдання

1. Перелічите інструменти пакету *MATLAB*, що можуть використовуватися в інженерних розрахунках та наукових досліджень в галузі електротехніки та електромеханіки.
2. З яких вікон складається робочий стіл (*Desktop*) пакету *MATLAB*?
3. В якому вікні вводяться команди та оператори *MATLAB* для миттєвого їх виконання? Який вигляд має запрошення на введення команди?
4. У яких текстових файлах зберігається коротенька інформація про призначення функцій в папках з інструментами *MATLAB*?
5. Як відокремлюються одна від одної команди в одному рядку? Як записати одну команду в декількох рядках?
6. Як очистити командне вікно?

7. Де зберігаються команди, що вводяться користувачем у командному вікні? Як можна викликати останню введену команду із буфера для її повтореня. Як очистити буфер команд?
8. Як зберегти сеанс роботи командного інтерпретатора у файлі? Як вивести його у командне вікно?
9. Де зберігаються значення змінних? Як очистити оперативну пам‘ять?
10. Які особливості роботи у програмному режимі? Як створити папку
11. Як установити формат виведення числової інформації?
12. Перелічите основні елементарні математичні функції пакету *MATLAB*.
13. Як утворюються матриці та вектори в MATLAB?
14. Які спеціальні вектори та матриці ви знаєте? Наведіть приклади їх використання.
15. Апробуйте команди version, ver, syntax, demo, help, doc, which, type, path, addpath, echo; опишіть їх призначення.

2. ОПЕРАЦІЇ З МАТРИЦЯМИ ТА ВЕКТОРАМИ

2.1. Арифметичні операції

У системі *MATLAB* реалізовано два типи арифметичних операцій: поелементні (Array) та матричні (Matrix). Інформацію про ці операції можна отримати виконанням команди

```
>> help ops
  Operators and special characters.
   Arithmetic operators.
    plus      - Plus                          +
    uplus     - Unary plus                    +
    minus     - Minus                         -
    uminus    - Unary minus                   -
    mtimes    - Matrix multiply               *
    times     - Array multiply                .*
    mpower    - Matrix power                  ^
    power     - Array power                   .^
    mldivide          - Backslash or left matrix divide     \
    mrdivide - Slash or right matrix divide         /
    ldivide   - Left array divide                   .\
    rdivide   - Right array divide                  ./
```

Матричні операції виконуються за правилами лінійної алгебри над масивами відповідних для цих операцій розмірів. Операндами ***поелементних операції*** можуть бути масиви однакових розмірів, скаляри, або скаляр і масив.

Операції «+» та «-» у матричній і поелементній формах не відрізняються одна від одної і позначаються однаково. Операції множення, правостороннього та лівостороннього ділення та підвищення у степінь у матричній формі позначаються як «*», «/», «\» та «^» відповідно, а у поелементній формі – як «.*», «./», «.\» та «.^», тобто зображення таких операцій утворюються додаванням символу «.» перед основним символом операції.

Розглянемо результати операцій `Z=X+Y` для двох матриць розміром $m \times n$ та `D=X+c` для матриці `D` розміром $m \times n$ і скалярної змінної `c:`

У першому випадку це буде матриця того ж розміру з елементами

$$Z_{i,j} = X_{i,j} + Y_{i,j}\ ; \quad i=1, 2, \ldots, m; \quad j=1,2,\ldots,n, \tag{2.1}$$

а у другому –

$$D_{i,j} = a_{i,j} + c\ ; \quad i=1, 2, \ldots, m; \quad j=1, 2, \ldots, n, \tag{2.2}$$

наприклад, при виконанні програми

```
X = [1 5 8; 4 7 2]
Y = [-3 2 6; 7 1 9]
Z = X+Y
c =10
D = X+c
```

отримаємо

```
X =
     1     5     8
     4     7     2
Y =
    -3     2     6
     7     1     9
Z =
    -2     7    14
    11     8    11
c =
    10
D =
    11    15    18
    14    17    12
```

За таким же алгоритм здійснюється операція віднімання.

За аналогією виконуються операції поелементного множення E=X.*Y, ділення F=X./Y та піднесення у степінь G=X.^Y над масивами однакового розміру:

$$E_{i,j} = X_{i,j} * Y_{i,j}\ ;\ F_{i,j} = X_{i,j} / Y_{i,j}\ ;\ G_{i,j} = X_{i,j} \wedge Y_{i,j}\ ; \quad i=1, 2, \ldots, m; \quad j=1,2,\ldots,n, \tag{2.3}$$

наприклад, при виконанні операцій

```
E = X.*Y
F = X./Y
G = X.^Y
```

отримаємо:

```
E =
    -3    10    48
    28     7    18
F =
   -0.3333    2.5000    1.3333
    0.5714    7.0000    0.2222
```

```
G =
       1        25     262144
   16384         7        512
```

Якщо один з операндів є скаляром c, то можливі такі поелементні операції: X*c, c *X ($X_{i,j}$* c=c * $X_{i,j}$), X./c ($X_{i,j}$ / c), c./ X (c /$X_{i,j}$), X.^c ($X_{i,j}$ ^ c), c.^ X (c ^ $X_{i,j}$), наприклад,

```
>> X*2
ans =
    2   10   16
    8   14    4
>> X./2
ans =
   0.5000   2.5000   4.0000
   2.0000   3.5000   1.0000
>> 2./X
ans =
   2.0000   0.4000   0.2500
   0.5000   0.2857   1.0000
>> X.^2
ans =
    1   25   64
   16   49    4
>> 2.^X
ans =
    2   32  256
   16  128    4
```

Матричним добутком **C** матриці **A** розміром $m\times k$ на матрицю **B** розміром $k\times n$, є матриця **C** розміром $m\times n$, елементи C_{ij} якої вираховуються як сума добутків елементів i-го рядка першого множника **A** на відповідні елементи j-го стовпчика другого множника **B**:

$$\underset{m\times k}{\mathbf{A}}\cdot\underset{k\times n}{\mathbf{B}}=\underset{m\times n}{\mathbf{C}}=\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1k} \\ a_{21} & a_{22} & \dots & a_{2k} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mk} \end{bmatrix}\cdot\begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{11} & b_{22} & \dots & b_{2n} \\ \dots & \dots & \dots & \dots \\ b_{k1} & b_{k2} & \dots & b_{kn} \end{bmatrix}=\begin{bmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \dots & \dots & \dots & \dots \\ c_{m1} & c_{m2} & \dots & c_{mn} \end{bmatrix}, \quad (2.4)$$

де

$$c_{ij}=a_{i1}b_{1j}+a_{i2}b_{2j}+....+a_{ik}b_{kj}=\sum_{l=1}^{k}a_{il}b_{lj}, \quad i=1, 2, \dots, m; \ j=1, 2, \dots, n. \qquad (2.5)$$

З формули (2.5) випливає, що внутрішні розміри матриць-аргументів

повинні збігатися, тобто другий множник повинен мати стільки рядків, скільки стовпчиків має перший множник. Результат цієї операції матиме таку кількість рядків, як перший множник, і таку кількість стовпців, як другий множник.

В *MATLAB* операцію (2.5) можна записати як

C (i,j) = A (i,:) * B (:,j)

Для добутку матриць, у загальному випадку, несправедливий переставний (комутативний) закон, тобто **A*****B**≠ **B*****A**.

Отже, операція «*» може здійснювати множення таких аргументів:

- двох матриць з однаковими внутрішніми розмірами C=A*B (див. формули (2.4), (2.5));
- матриці **A** розміром *m*×*k* на вектор-стовпець **b** розміром *k*×*1* (d = A*b),, результат – вектор-стовпець **d** розміром *m* ×1:

$$d_i = a_{i1}b_1 + a_{i2}b_2 + ... + a_{ik}b_k = \sum_{j=1}^{k} a_{ij}b_j \quad , \qquad i = 1, 2, \dots , m; \tag{2.6}$$

- вектору-рядка **a** розміром *1*×*k* на матрицю **B** розміром *k*×*n* (e=a*B), результат – вектор-рядок e розміром *1*×*n:*

$$e_j = a_1b_{1j} + a_2b_{2j} + ... + a_kb_{kj} = \sum_{i=1}^{k} a_ib_{ij}\,, \quad j = 1, 2, \dots , n; \tag{2.7}$$

- вектору-рядка **a** розміром *1*×*k* на вектор-стовпець **b** розміром *k*×*1* (g=a*b), результат – скалярний добуток *g*:

$$g = a_1b_1 + a_2b_2 + ... + a_kb_k = \sum_{i=1}^{k} a_ib_j\,, \qquad j = i; \tag{2.8}$$

- вектору-стовпчика **b** розміром *n*×*1* на вектор-рядок **a** розміром *1*×*n* (D=b*a), результат – квадратна матриця **D** розміром *n*× *n*:

$$d_{ij} = a_ib_j\,, \qquad i=1, 2, \dots, k; \qquad j=1, 2, \dots, k; \tag{2.9}$$

- скаляра на скаляр, результат – скаляр.

Результат операції (2.8) називають ***скалярним добутком*** двох векторів однакового розміру. Для його розрахунку можна скористатися функцією dot із папки specfun: g = dot (a,b). В такому випадку не має значення є вектори-

аргументи рядками чи стовпцями.

Наприклад, при виконанні програми

```
A = [2 8 -3 4; -6 5 7 1]                % Матриця 2*4
B = [ 5 1 0; 2 -7 9; 3 4 8; 5 6 7]    % Матриця 4*3
C = A*B                                  % Матриця 2*3
% C1 = B*A  -  Помилка в розмірностях матриць-множників
b = [2; 8; 3 ;-1 ]       % Вектор-стовпець із 4 елементів
d =A*b                   % Вектор-стовпець із 2 елементів
a = 1:4                  % Вектор-рядок із 4 елементів
e = a*B                  % Вектор-рядок із 3 елементів
g = a*b                  % Скаляр ( 1*1)
D = b*a                  % Матриця 4*4
```

отримаємо:

```
A =
     2     8    -3     4
    -6     5     7     1
B =
     5     1     0
     2    -7     9
     3     4     8
     5     6     7
C =
    37   -42    76
     6    -7   108
b =
     2
     8
     3
    -1
d =
    55
    48
a =
     1     2     3     4
e =
    38    23    70
g =
    23
D =
     2     4     6     8
     8    16    24    32
     3     6     9    12
    -1    -2    -3    -4
```

У відповідності до поняття про добуток матриць, у цілу додатну степінь *k* можна піднести тільки квадратну матрицю, тобто результатом операції A^k буде

$$\mathbf{A}^k = (\underbrace{((\mathbf{A}*\mathbf{A})*\mathbf{A})*\ldots*\mathbf{A}}_{k \text{ співмножників}}), \qquad (2.10)$$

тобто операції D^3 та D*D*D над матрицею D з попереднього прикладу дадуть однаковий результат:

```
ans =
      1058       2116       3174       4232
      4232       8464      12696      16928
      1587       3174       4761       6348
      -529      -1058      -1587      -2116
```

Крім того, у *MATLAB* передбачені операції правобічного та лівобічного ділення масивів, в яких насправді операція ділення на задану матрицю замінюється операцією множення на обернену до неї матрицю, тобто для квадратних матриць A та B, вектору-стовпця b та вектору-рядка c можливі такі операції: A/B (еквівалентно A*B^(-1) або A*inv (B)), A\B (еквівалентно A^(-1)*B або inv (A)*B), c/A (еквівалентно c*A^(-1) або c*inv (A)), A\b (еквівалентно A^(-1)*b або inv (A)*b). Нагадаю, що обернена матриця – це квадратна матриця, результатом матричного множення якої на вихідну матрицю (теж квадратну) є діагональна одинична матриця. Наприклад, при виконанні програми

```
% Правобічне та лівобічне матричне ділення
 echo off
  A = [1 6 4; 7 2 9; 3 5 6]
  B = [3 7 8; 2 5 1; 9 4 6]
  b = [ -2; 5; -3]
  c = [6 0 1]
  echo on
  A/B
  A\B
  c/A
  A\b
```

отримаємо

```
  A =
     1     6     4
     7     2     9
     3     5     6
  B =
     3     7     8
     2     5     1
     9     4     6
  b =
```

```
    -2
     5
    -3
c =
     6     0     1

   A/B
ans =
    0.5858    0.5439   -0.2050
    0.6569   -1.1757    0.8201
    0.6653   -0.0251    0.1172
    A\B
ans =
  -40.4286   18.1429    0.5714
  -16.2857    8.4286    1.7143
   35.2857  -15.4286   -0.7143
    c/A
ans =
   24.1429   11.8571  -33.7143
   A\b
ans =
   21.7143
    8.1429
  -18.1429
```

Результати не зміняться, якщо наведені операції замінити визначеними вище еквівалентними операціями.

2.2. Визначення розмірів матриць та векторів

Розмір векторів та матриць у *MATLAB* визначається такими функціями:

`L = length(X)` – довжина вектору `X`;

`m = size(A,1)` – кількість рядків у матриці `A`, тобто розмір матриці за першим індексом;

`n = size(A,2)` – кількість стовбців у матриці `A`, тобто розмір матриці за другим індексом;

`[m,n] = size(A)` – кількість рядків `m` та стовбців `n` у матриці `A`.

2.3. Маніпуляції над матрицями

Якщо у матриці $\mathbf{A}$ розміром $m \times n$ замінити рядки відповідними стовпцями, то одержимо матрицю $\mathbf{A}^T$ розміром $n \times m$, яка має назву транспонованої у відношенні до матриці $\mathbf{A}$.

Отже,

$$a_{i,j}^{T} = a_{j,i} \; ; (i=1,2,..., n; j=1,2,..., m). \qquad (2.11)$$

У *MATLAB* ця операція позначається символом «′».

Функції, що здійснюють деякі маніпуляції над матрицями знаходяться у папці elmat:

```
>> help elmat
  Elementary matrices and matrix manipulation.
Matrix manipulation.
    reshape  - Reshape array.
    diag     - Diagonal matrices and diagonals of matrix.
    fliplr   - Flip matrix in left/right direction.
    flipud   - Flip matrix in up/down direction.
    rot90    - Rotate matrix 90 degrees.
```

Операція **A(:)** створює з елементів матриці **A** розміром $m \times n$ вектор стовбець, складений із послідовно приєднаних один до одного стовпчиків вихідної матриці.

2.4. Базові функції математичного аналізу над векторами та матрицями

Базові функції математичного аналізу знаходяться серед інструментів папки datafun:

```
>> help datafun
    Data analysis and Fourier transforms.
    Basic operations.
    sum        - Sum of elements (сума елементів).
    cumsum     - Cumulative sum of elements (накопичення суми).
    prod       - Product of elements (добуток елементів).
    cumprod    - Cumulative product of elements (накопичення добутку).
    max        - Largest component (найбільший елемент).
    min        - Smallest component (найменший елемент).
    sort       - Sort in ascending order (сортування за збільшенням).
    sortrows   - Sort rows in ascending order (сортування рядків за
збільшенням).
```

mean - Average or mean value (середнє арифметичне).
median - Median value (серединне значення).
std - Standard deviation (середньоквадратичне відхилення).
var - Variance (дисперсія).
Finite differences.
diff - Difference and approximate derivative (прямі різниці або апроксимовані похідні).

Особливістю наведених вище функцій, як і багатьох інших, полягає у тому, що для матриць вони виконуються для кожного стовпчика, а не для матриці у цілому. Щоб виконати відповідну операцію для кожного рядка, треба транспонувати вихідну матрицю і отриманий результат, наприклад, Amean_rows = (mean(A'))'. Щоб виконати бажану операцію над матрицею у цілому, треба звернутися до цієї функції двічі, наприклад, S = sum (sum(A)).

Функції max та min можуть працювати як з одним аргументом, так і з двома. У другому випадку обидва аргументи повинні мати однаковий розмір, тому що відповідна операція виконується поелементно.

Серединним значенням при непарній кількості елементів є значення середнього елемента упорядкованого вектору, а при парній кількості – середнє арифметичне двох середніх елементів.

Середньоквадратичне значення (Standard deviation) вектору $s(x)$ обчислюється функцією std (x) за формулою

$$s(x) = \sqrt{\frac{1}{n-1} \cdot \sum_{i=1}^{n} (x_i - x_{mean})^2} \,. \qquad (2.12)$$

Дисперсія (variance) вектору розраховується функцією var (x) як квадрат середньоквадратичного відхилення:

$$D(x) = s^2(x) \,. \qquad (2.13)$$

Першими прямими різницями одномірного масиву

$$\mathbf{V} = [v_1 \quad v_2 \quad v_3 \quad \ldots \quad v_{n-2} \quad v_{n-1} \quad v_n]$$

називають вектор

$$\Delta\mathbf{V} = [\Delta v_1 \;\; \Delta v_2 \;\; \ldots \; \Delta v_{n-2} \;\; \Delta v_{n-1}] = [v_2 - v_1 \;\; v_3 - v_2 \;\; \ldots \; v_{n-1} - v_{n-2} \;\; v_n - v_{n-1}]. \qquad (2.14)$$

Другі прямі різниці – це прямі різниці від перших прямих різниць, тобто

$$\Delta^2\mathbf{V} = \left[\Delta^2 v_1 \quad \ldots \quad \Delta^2 v_{n-2}\right] = \left[\Delta v_2 - \Delta v_1 \quad \ldots \quad \Delta v_{n-1} - \Delta v_{n-2}\right] = \\ = \left[v_3 - 2v_2 + v_1 \quad \ldots \quad v_n - 2v_{n-1} + v_{n-2}\right]. \tag{2.15}$$

Приклади використання базових функцій математичного аналізу над одномірними масивами демонструє наступна програма:

```
% Базові функції математичного аналізу над векторами:
 echo off
  x = [1    6 14  7  -2 9 13 5 0]
  y = [23 7 -8 12   5 1  9  4 3]
  echo on
  Sx = sum (x)
  Sx_cum = cumsum (x)
  Py = prod (y)
  Py_cum = cumprod (x)
  xmax = max (x)
  [ xmax, imax] = max (x)
  ymin = min (y)
  [ ymin, imin] = min (y)
  xy_max = max (x, y)
  x_mean = mean (x)
  y_mean = sum (y) / length (y)
  xs = sort (x)
  ys = sort (y)
  x_med = median (x)
  y_med = median (y)
  x_std = std (x)
  y_std = sqrt (sum ( (y - mean (y)).^2) / (length (y) -1))
  Dx = var (x)
  Dy = std(y)^2
  dx = diff (x)
  dy = y (2 : end) -  y (1 : end-1)
  d2x = diff (x, 2)
  d2y = diff (diff (y) )
  d3x = diff (d2x)
```

При її виконанні маємо такі результати:

```
x =
    1    6    14    7    -2    9    13    5    0
y =
   23    7    -8    12    5    1    9    4    3
  Sx = sum (x)
Sx =
   53
  Sx_cum = cumsum (x)
Sx_cum =
    1    7    21    28    26    35    48    53    53
  Py = prod (y)
```

```
Py =
    -8346240
  Py_cum = cumprod (x)
Py_cum =
  Columns 1 through 7
         1          6         84        588      -1176     -10584    -137592
  Columns 8 through 9
     -687960           0
  xmax = max (x)
xmax =
    14
  [ xmax, imax] = max (x)
xmax =
    14
imax =
     3
  ymin = min (y)
ymin =
    -8
  [ ymin, imin] = min (y)
ymin =
    -8
imin =
     3
  xy_max = max (x, y)
xy_max =
    23     7    14    12     5     9    13     5     3
  x_mean = mean (x)
x_mean =
    5.8889
  y_mean = sum (y) / length (y)
y_mean =
    6.2222
  xs = sort (x)
xs =
    -2     0     1     5     6     7     9    13    14
  ys = sort (y)
ys =
    -8     1     3     4     5     7     9    12    23
  x_med = median (x)
x_med =
     6
  y_med = median (y)
y_med =
     5
  x_std = std (x)
x_std =
    5.5777
  y_std = sqrt (sum ( (y - mean (y)).^2) / (length (y) -1))
y_std =
    8.4377
```

```
  Dx = var (x)
Dx =
   31.1111
  Dy = std(y)^2
Dy =
   71.1944
  dx = diff (x)
dx =
     5     8    -7    -9    11     4    -8    -5
  dy = y (2 : end) -  y (1 : end-1)
dy =
   -16   -15    20    -7    -4     8    -5    -1
  d2x = diff (x, 2)
d2x =
     3   -15    -2    20    -7   -12     3
  d2y = diff (diff (y) )
d2y =
     1    35   -27     3    12   -13     4
  d3x = diff (d2x)
d3x =
   -18    13    22   -27    -5    15
```

Приклади використання базових функцій математичного аналізу над матрицями демонструє наступна програма:

```
% Базові функції математичного аналізу над матрицями:
echo off
X = [1 5 8; 4 7 2]
Y = [-3 2 6; 7 1 9]
echo on
SX_col = sum (X)
SX = sum (sum (X))
PY_col = prod (Y)
Py = prod (PY_col)
Xmax_col = max (X)
Xmax = max (max (X))
Ymin_col = min (Y)
Ymin = min (Ymin_col)
XY_max = max (X, Y)
Xs_col = sort (X)
Ys_row = sortrows (Y)
```

При її виконанні маємо такі результати:

```
X =
     1     5     8
     4     7     2
Y =
    -3     2     6
     7     1     9
  SX_col = sum (X)
SX_col =
```

```
     5    12    10
  SX = sum (sum (X))
SX =
    27
  PY_col = prod (Y)
PY_col =
   -21     2    54
  Py = prod (PY_col)
Py =
      -2268
  Xmax_col = max (X)
Xmax_col =
     4     7     8
  Xmax = max (max (X))
Xmax =
     8
  Ymin_col = min (Y)
Ymin_col =
    -3     1     6
  Ymin = min (Ymin_col)
Ymin =
    -3
  XY_max = max (X, Y)
XY_max =
     1     5     8
     7     7     9
  Xs_col = sort (X)
Xs_col =
     1     5     2
     4     7     8
  Ys_row = sortrows (Y)
Ys_row =
    -3     2     6
     7     1     9
```

2.5. Контрольні питання та завдання

1. Перелічите арифметичні операції, що виконуються над масивами однакового розміру поелементно. Наведіть приклади.
2. Що таке матричний добуток? Що таке скалярний добуток? За якими формулами вони розраховується? Які розміри повинні мати аргументи та результати цих операцій?
3. Як і над якими операндами виконується матрична операція піднесення у цілу степінь?

4. Які математичні операції виконують операції A/B, A\B, c/A, A\b? Як повинні бути узгоджені розміри матриць A, B та векторів c, b в цих операціях?
5. Як програмно визначити розміри матриць та векторів?
6. Які маніпуляції над матрицями та векторами можна здійснити в програмному середовищі *MATLAB*?
7. Що таке транспонована матриця? Як здійснити транспонування?
8. Яка різниця між поворотом матриці та її відображенням? Як здійснити ці операції?
9. Як витягнути матрицю в один стовпчик? А в один рядок?
10. Поясніть, як працює функція reshape?
11. Які операції може здійснити функція diag?
12. Перелічите базові функції математичного аналізу векторів і матриць та їх призначення. Якими є особливості виконання цих функцій для матриць?
13. Скільки вхідних та вихідних параметрів можуть мати функції max і min? Що це за параметри?
14. Чим відрізняються функції cumsum та cumprod від функцій sum та prod?
15. Чим відрізняється серединне значення елементів масиву від середнього? Як вони визначаються в середовищі *MATLAB*?
16. Як розрахувати середньо-квадратичне відхилення елементів масиву від середнього арифметичного значення?
17. Що таке дисперсія? Яка функція її розраховує?
18. Що таке прямі різниці? Що означає їх порядок? Як пов‘язані між собою розміри результуючого та вихідного масивів? Як їх розрахувати?
19. Виконати операції над матрицями

$$\mathbf{A}=\begin{bmatrix} 2 & 3{,}1 \\ 4{,}5 & 10{,}7 \\ 7 & 1 \end{bmatrix};\ \mathbf{B}=\begin{bmatrix} 7{,}5 & 11 & 1{,}7 \\ 5 & 4 & 2 \end{bmatrix},\ \mathbf{C}=\begin{bmatrix} 1 & 0 \\ 7 & 8 \\ 5 & 6 \end{bmatrix},\ \mathbf{D}=\begin{bmatrix} 7{,}4 & 5 \\ 9 & 8 \end{bmatrix}$$

у відповідності з виразами, наведеними у другому стовпчику таблиці 2.1. Для матриці **K** виконати дії з колонки 3 табл. 2.1. Отримати вектор **V** шляхом витягування матриці **H** по рядкам у 1 рядок та упорядковати його за збільшенням елементів (вектор **Vu**). Знайти для вектору **V** перші, другі та треті прямі різниці, а також середнє, серединне і середньо-квадратичне значення.

Таблиця 2.1

Вар.	**ЗАВДАННЯ**	
1	K= A*D+B^T-C	Знайти суми елементів кожного рядка
2	K= (A+C)^T.*B	Знайти добутки елементів кожного стовпчика
3	K= (A-C)*(D*B)	Знайти середнє арифметичне першого рядка
4	K= (B^T-A-C)*D	Знайти максимальний елемент останнього стовпчика
5	K= (C*D-A)^T+B	Знайти мінімальний елемент матриці та його позицію
6	K= (C-A)*D^T*B	Упорядкувати стовпці за зменшенням
7	K= (A.*C)*B	Упорядкувати рядки за збільшенням
8	K= A*(D*C^T-B)	Перегорнути матрицю у горизонтальній площині
9	K= C*D-A-B^T	Перегрупувати матрицю 3×2 у матрицю 2×3
10	K= A*D^2+C	Перегорнути матрицю у вертикальній площині
11	K= (A+C)*B	Обернути матрицю на 180°
12	K= (A*B)^2	Знайти суму максимальних елементів рядків
13	K= D*C^T+B	Знайти добуток мінімальних елементів стовпчиків
14	K= A*D+B^T-C	Знайти кумулятивні суми рядків
15	K= A*(D*C^T-B)	Знайти кумулятивні добутки стовпчиків

3. ПОБУДОВА ТА ОФОРМЛЕННЯ БАЗОВИХ ДВОВИМІРНИХ ГРАФІКІВ ФУНКЦІЙ

3.1. Загальні відомості

Графічні засоби пакета *MATLAB* орієнтовані не на створення графічних примітивів (прямокутників, кіл, тощо), а на побудову дво- і тривимірних графіків функціональних залежностей у самому різноманітному вигляді. Графіки будуються в окремих вікнах, що мають назву *Figure*.

Початкову інформацію про основні функції двовимірної графіки можна отримати виконанням команди *help graph2d*

```
>> help graph2d
Two dimensional graphs.
Elementary X-Y graphs.
   plot         - Linear plot.
   loglog       - Log-log scale plot.
   semilogx     - Semi-log scale plot.
   semilogy     - Semi-log scale plot.
   polar        - Polar coordinate plot.
   plotyy       - Graphs with y tick labels on the left and right.
Axis control.
   axis       - Control axis scaling and appearance.
   grid       - Grid lines.
   hold       - Hold current graph.
   subplot    - Create axes in tiled positions.
Graph annotation.
   title      - Graph title.
   xlabel     - X-axis label.
   ylabel     - Y-axis label.
   text       - Text annotation.
   gtext      - Place text with mouse.
```

3.2. Побудова двовимірних графіків в Декартових координатах

Графіки, побудовані за допомогою функцій *plot*, *semilogx*, *semilogy* та *loglog* відрізнятимуться один від одного тільки масштабуванням осей (лінійне,

напівлогарифмічне або логарифмічне). Звернення до цих функцій має однаковий формат. Тому розглянемо способи їх застосування на прикладі функції *plot*.

Домовимося, що в робочому просторі (*Workspace*) пакета *MATLAB* існують значення таких перемінних: x, y, – вектори-рядки із n елементів; u – вектор-стовбець із m елементів; A, B, C – матриці розміром m×n; z – комплексне число, Z – вектор комплексних чисел. Тоді

plot (y) – будує графік функції $y(i) = f(i)$ по точках з абсцисами $i = [1, 2, \ldots, n]$ та ординатами $[y(1), y(2), \ldots, y(n)]$;

plot (x,y) – будує графік функції $y(i) = f(x(i))$;

plot (Z) – те ж саме, що і plot(real(Z),imag(Z));

plot (A) – будує в одній системі координат n графіків, кожний з яких уявляє собою залежність елементів одного стовпчика матриці A від порядкового номеру рядка;

plot (u,A) – те ж саме, але у функції елементів вектору-стовбця u.

plot(x,A) – будує в одній системі координат m графіків, кожний з яких уявляє собою залежність елементів одного рядка матриці A від елементів вектору-рядка x.

Одним графічним оператором можна вивести в одній системі координат декілька графіків у такий спосіб:

plot (x1,y1, x2,y2, …, xN,yN)

Якщо треба побудувати графік функції, заданої аналітично, у лінійному масштабі, то вектор аргументів зручно задавати оператором

x = linspace (x0, xk)

Приклад 3.1. *Побудувати в одній системі координат графіки функцій* $y = \sin 2x \cdot \cos x$ *та* $z = e^{1/(x+1)}$ *на інтервалі* $x \in [0,10]$.

```
x =linspace(0,10);
y = sin (2*x) .* cos (x);
plot (x,y)
z = exp (1./(x+1)); A = [y; z]; plot (x, A)
```

Графік, отриманий при виконанні цієї програми подано на рис. 3.1. Цей графік можна імпортувати у Microsoft Word або у будь-який графічний редактор, наприклад, у Microsoft Office Visio, скориставшись через меню графічного вікна командою Edit→Copy Figure. У графічному редакторі можна скоріше і якісніше, ніж у середовищі пакета MATLAB доповнити графік допоміжними підписами та відкоригувати зображення.

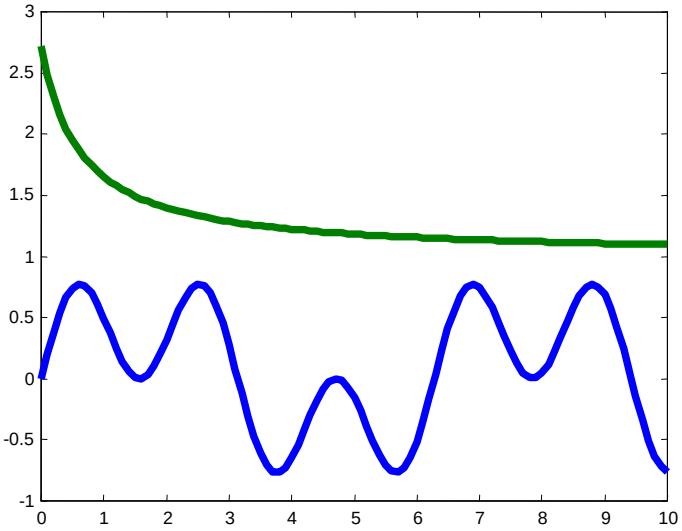


Рис. 3.1. Графіки функцій до прикладу 3.1

3.3. Керування стилями, кольорами, товщиною ліній та розмірами маркерів на графіках

Кожну пару векторів, що застосовуються для виводу одного з графіків, можна доповнити ***строковим параметром*** Style, який може мати у своєму складі ***від одного до чотирьох символів***, що визначатимуть колір та способи зображення маркера вузлових точок таблиці та лінії, що поєднує сусідні точки:

plot(x1,y1,Style1, x2,y2,Style2,…, xN,yN,StyleN)

Можливі значення символів у параметрі Style та їх сенс відображено у табл. 3.1. Отже, параметр Style може виглядати так: 'r', 'm:', 'o', 'k-.x', тобто він може визначати одну, дві або три властивості графіка із табл. 3.1.

Кольори, наведені у табл. 3.1 отримані змішуванням трьох основних кольорів (червоного, зеленого та синього) так званої **rgb**-палітри однакової (максимальної) інтенсивності.

Таблиця 3.1

Колір лінії	**Тип маркера**	**Тип лінії**
b – ***b****lue* (синій)	**.** – крапка	- – суцільна (*solid*)
g – ***g****reen* (зелений)	**o** – коло	**:** – дрібний (точковий) пунктир (*dotted*)
r – ***r****ed* (червоний)	**x** – хрестик	-- – штрихова (*dashed*)
y – ***y****ellow* (жовтий)	**+** – плюс	-. – штрих-пунктирна (*dashdotted*)
m – ***m****agenta* (пурпурний)	***** – зірка	
c – ***c****yan* (блакитний)	**s** – квадрат (***s****quare*)	
w – ***w****heit* (білий)	**d** – ромб (***d****iamond*)	
k – *blac****k*** (чорний)	**v** – трикутник вершиною вниз	
	^ – трикутник вершиною угору	
	< – трикутник вершиною вліво	
	> – трикутник вершиною вправо	
	p – п‘ятикутна зірка (***p****entagram*)	
	h – шестикутна зірка (***h****exagram*)	

У пакеті *MATLAB* кольори можна задавати вектором-рядком із 3-х чисел в діапазоні від 0 до 1, які визначають інтенсивність змішуваних основних кольорів: [red green blue]. 1 означає максимальну інтенсивність, а 0 – нульову.

Кольорам з табл. 3.1 відповідають такі значення **rgb**-векторів:

[0 0 1] – blue [1 0 1] – magenta

[0 1 0] – green [1 1 0] – yellow

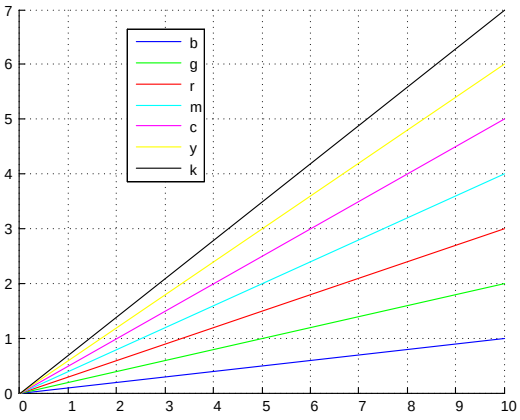
[1 0 0] – red [0 0 0] – black

[0 1 1] – cyan [1 1 1] – white

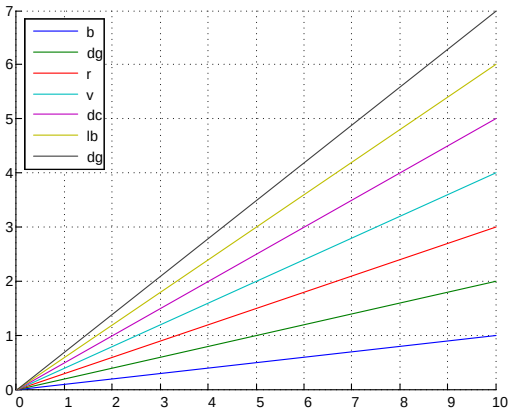
Треба відзначити, що кольори *cyan, magenta*, та *yellow* виявляються занадто яскравими та погано видними на білому фоні, особливо при чорно-білому друку. Тому за замовчанням в *MATLAB* при зображенні декількох графіків в одній системі координат, що виводяться одним оператором plot застосовують інші кольори, а саме:

[0 0 1] – blue
[0 0.5 0] – dark green
[1 0 0] – red
[0 0.75 0.75] – dark cyan
[0.75 0 0.75] – violet
[0.75 0.75 0] – light brown
[0.25 0.25 0.25] – dark grey

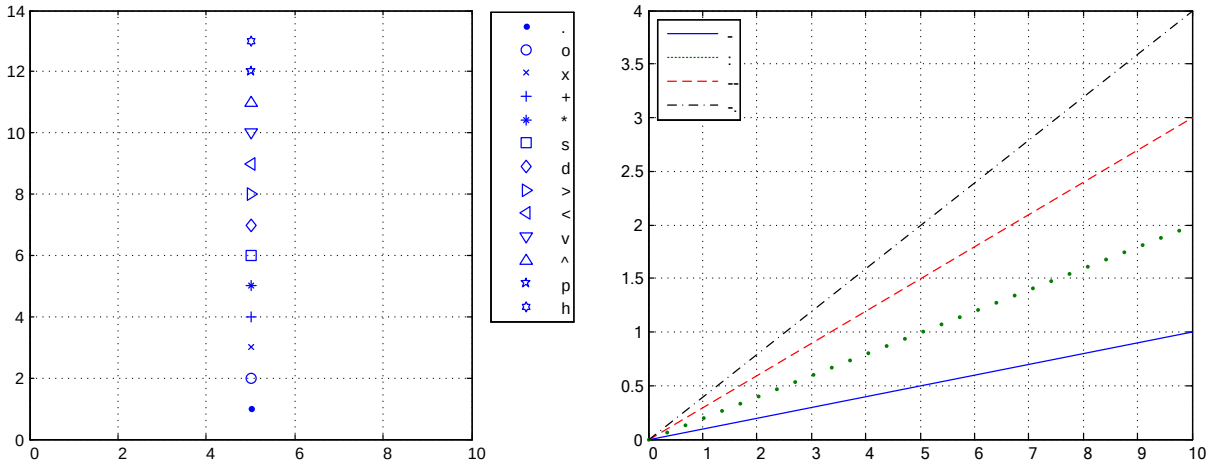
На рис. 3.2 показані графіки, які демонструють кольори, відображені у табл. 3.1 (а), послідовність кольорів, прийнята у MATLAB за замовчанням (б), стилі зображення маркерів графіків (в) та стилі зображення ліній на графіках (г).



а)



б)



в) г)

Рис. 3.2. Кольори та стилі зображення графіків

Окремі властивості (*Properties*) графіків можна змінювати, застосовуючи засоби так званої дескрипторної (низькорівневої) графіки, яка є елементом об‘єктно-орієнтованого програмування, застосовуючи функцію plot у такому форматі:

plot (x,y, PropName1,PropValue1, PropName2,PropValue2, …)

Інформацію про імена властивостей (*PropertyName*) та їх значення за замовчанням (*PropertyValue*) можна побачити після виконання функції

get (gca)

де gca означає ***G**raphic **C**urrent **A**xes* (поточна система координат).

Наведемо перелік деяких властивостей, що виводяться при виконанні цієї команди:

Box = on
Color = [1 1 1]
ColorOrder = [(7 by 3) double array]
FontName = Helvetica
FontSize = [10]
FontWeight = normal
GridLineStyle = :
LineStyleOrder = -
LineWidth = [0.5]
NextPlot = replace
XLim = [0 10]
XScale = linear
XTick = [(1 by 11) double array]
YAxisLocation = left
YLim = [1 2.8]

```
YScale = linear
YTick = [1 1.2 1.4 1.6 1.8 2 2.2 2.4 2.6 2.8]
```

З цього списку видно, що товщина лінії ('LineWidth') за замовченням має значення 0.5, ім'я шрифту ('FontName') для виводу текстів – Helvetica, розмір шрифту ('FontSize') – 10 пт., жирність шрифту ('FontWeight') – нормальна ('normal'), шкала осей абсцис і ординат – лінійна, розташування осі ординат – ліворуч, а палітра чергування кольорів ('ColorOrder') є матрицею розміром 7×3, складеною з векторів рядків, наведених вище.

Щоб конкретизувати значення якогось параметру, треба виконати команду

```
get (gca, PropName)
```

Наприклад,

```
>> MC = get (gca, 'ColorOrder')
MC =
       0         0    1.0000
       0    0.5000         0
  1.0000         0         0
       0    0.7500    0.7500
  0.7500         0    0.7500
  0.7500    0.7500         0
  0.2500    0.2500    0.2500
>> get (gca, 'XTickLabel')
ans =
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
```

Змінити колір і товщину лінії можна так:

```
plot(x1,y1, 'LineWidth', 2, 'Color' ,[0.8 0.8 0])
```

або

```
h = plot (x1,y1), set (h , 'LineWidth', 2, 'Color' ,[0.8 0.8 0])
```

Наявність лівостороннього параметру ***handle of the graphic objects*** при зверненні до функції plot дає можливість отримати не тільки сам графік, а ще і його властивості.

Наприклад, при виконанні операторів

```
x = linspace (0,2*pi,10); y = sin(x);
H = plot (x,y,'o-'), grid
```

отримуємо не тільки графік, а ще й інформацію про його властивості:

```
H =
  Line with properties:
              Color: [0 0.4470 0.7410]
          LineStyle: '-'
          LineWidth: 0.5000
             Marker: 'o'
         MarkerSize: 6
Show all properties
```

Ці властивості можна змінити, наприклад, збільшити розмір маркера:

```
set (H, 'MarkerSize', 8)
```

Властивостей графічних вікон і їх об‘єктів дуже багато. Тому не треба намагатися змінювати властивості, сенс яких є не зрозумілим.

3.4. Нанесення на графік заголовків, позначень осей, текстових пояснень та коментарів

Для ***нанесення заголовку*** в пакеті *MatLab* призначена функція title, для ***позначення осей*** – функції xlabel та ylabel, а для ***нанесення інших текстів*** (коментарів, позначень графіків, тощо) – функції text та gtext.

Перші три функції виводять тексти у наперед заданих позиціях: заголовок зверху, а позначення осей уздовж осей. Тому у найпростішому варіанті вони можуть застосовувати тільки один вхідний аргумент – 'Текст', наприклад,

```
title ('Вольт-амперна характеристика діода'),
xlabel ('Струм,А'), ylabel('Напруга,В')
```

Для функції text треба ще задати координати виводу (у системі координат, що застосовується при побудові графіка): text (x,y,'текст'), а функція gtext ('текст') позиціонує текст за допомогою миші. підпис осі абсцис.

За замовчанням усі тексти виводяться стилем Helvetica нормальної жирності (normal), розмір шрифту 10 пт. За бажанням ці параметри можна змінити, додавши після параметру 'Текст' конструкцію 'FontName', 'НазваШрифту', 'FontSize', РозмірШрифту, 'FontWeight', 'ЖирністьШрифту, наприклад,

```
Xlabel ('Струм,А', 'FontName','Arial', 'FontSize',12, 'FontWeight','bold')
```

Розташування заголовка та позначень осей продемонстровано на рис 3.3.

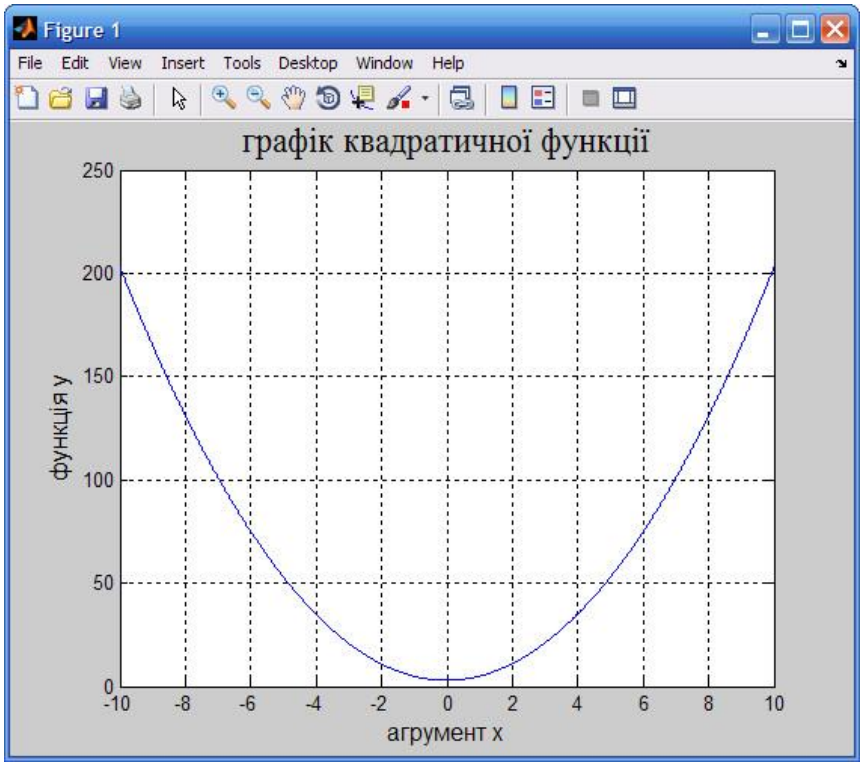


Рис. 3.3. Приклад оформлення підписів графіка

Цей графік отримано в результаті виконання наступної програми:

```
x=linspace(-10,10);
y=2*x.^2+3;
plot(x,y)
grid on
xlabel('агрумент x','FontName','Arial','FontSize', 12)
ylabel('функція y','FontName','Arial','FontSize', 12)
title('графік квадратичної функції', 'FontName','Times New Roman','FontSize',16)
```

При наявності в одній системі координат декількох графіків, зображених різними кольорами або стилями, доцільно додавати до них табличку з поясненнями, яка створюється функцією

legend ('text1', 'text2', …, 'textN', Pos)

Кількість текстових параметрів, що отримують коментарі, пояснення або позначення графіків, дорівнює кількості графіків. Останній параметр визначає позицію розташування легенди і може приймати такі значення: 1, 2, 3, 4 –

номер квадранта, -1 – найбільш зручне місце у полі графіків, 0 – поза полем графіків.

На рис. 3.2 параметр Pos у функції legend має для графіка а) значення 0, для графіка в)значення -1, а для графіків б) і г) значення 2. Якщо визначена позиція виведення легенди виявиться невдалою, її можна змінити в інтерактивному режимі, перетягнувши легенду в інше місце за допомогою миші.

Мишка застосовується при зверненні ще до однієї графічної функції, яка визначає чисельні значення координат виділених за допомогою миші точок графіка:

```
[x,y] = ginput(n)
```

де n – кількість точок.

3.5. Керування системою координат

Нанесенням на графік координатної сітки керує команда grid у таких модифікаціях: grid on – увімкнути режим зображення сітки, grid off – вимкнути цей режим, grid – змінити стан команди.

За аналогією команда hold керує ***режимом затримки графіка***, побудованого попереднім оператором plot, при виконанні наступного оператора plot, тобто hold on вмикає цей режим, hold off вимикає його, а hold змінює поточний стан команди. У стані hold off, який застосовується за замовчанням, у результаті виконання операцій

```
plot (x1,y1),  plot (x2,y2)
```

відкривається графічне вікно, у нього виводиться перший графік, який миттєво замінюється другим графіком.

Щоб побачити обидва графіки послідовно в одному вікні, треба між функціями plot поставити оператор pause, який затримує виконання наступної операції до натискування будь-якої клавіши.

Вивести обидва графіки разом в одне вікно можна сукупністю операторів

```
plot (x1,y1), hold on, plot (x2,y2)
```

або

plot (x1,y1, x2,y2)

Якщо два графіки в одному вікні мають суттєво різні масштаби, то для їх зображення зручно застосувати функцію

plotyy (x1,y1, x2,y2)

яка виводить дві осі ординат (зліва для першого графіка і зліва для другого, наприклад, графік, зображений на рис. 3.4, є результатом виконання програми

```
x=linspace(-5,5);
y1=x.^2-12*x;  y2=100*x.^2-2*x-3;
plotyy(x,y1,x,y2)
```

Функції

xlim ([x_left x_reight]),

ylim ([y_bottom y_top])

та

axis ([x_left x_reight y_bottom y_top])

встановлюють ***межі системи координат***.

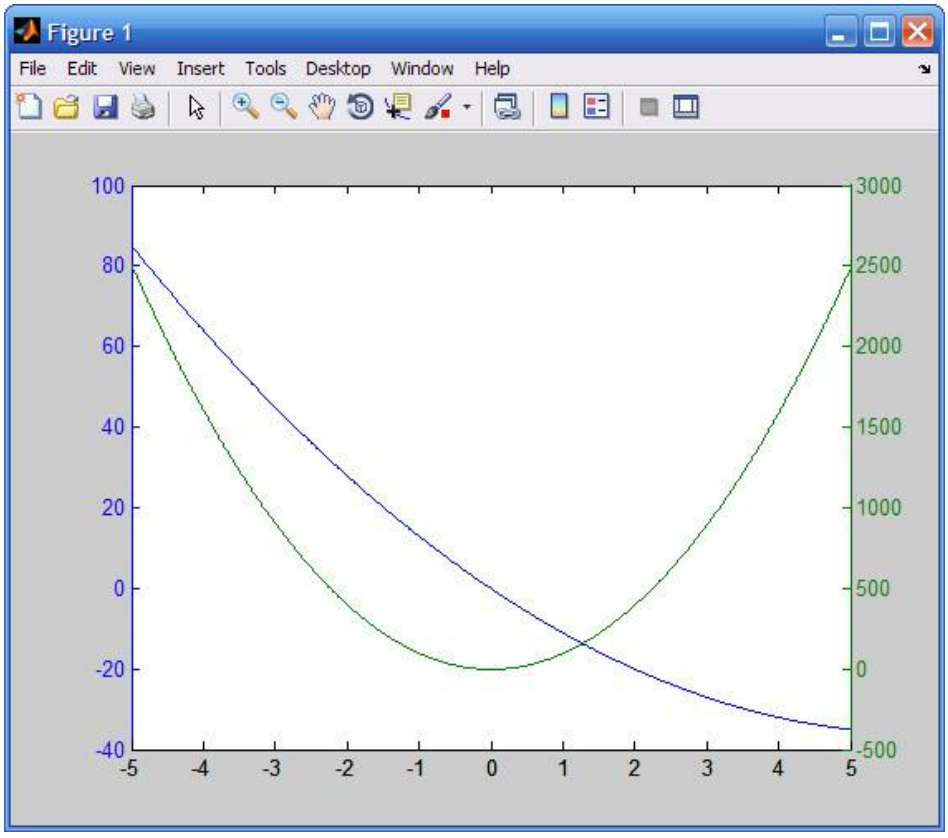


Рис. 3.4. Приклад суміщення двох графіків з різними масштабами

3.6. Керування графічними вікнами

Для керування графічними вікнами використовують такі команди:

figure – відкриває нове графічне вікно з черговим номером;

figure (n) – відкриває або активізує графічне вікно з заданим номером;

close – закриває активне графічне вікно;

close (n) – закриває графічне вікно з заданим номером;

close all – закриває усі графічні вікна;

shg – показує активне графічне вікно на передньому плані.

Кожне графічне вікно може бути розбите на декілька підвікон. Для цього використовується команда

subplot (m,n,p)

яка створює «матрицю» підвікон розміром m×n, здійснює їх наскрізну нумерацію за рядками і активізує підвікно з номером p $(1 \le p \le m \cdot n)$, у якому будуть виконуватися дії, передбачені наступним графічним оператором.

Для переходу у інше графічне підвікно треба знову виконати оператор subplot, змінюючи у ньому тільки значення останнього параметру, тобто номер активного підвікна, наприклад.

subplot (1,2,1), plot(x1,y1), subplot (1,2,2), plot(x2,y2)

Підвікна, розташовані поруч, можна поєднувати в єдине підвікно, наприклад,

subplot (2,3,[1,2]),…, subplot (2,3,[4,5]),…, subplot (2,3,[3;6]),…

Наведемо приклад побудови графіків у підвікнах

```
figure                  %відкриття загального графічного вікна
subplot(2,3,1)    %відкриття першого підвікна
x=linspace(-10,10);     %завдання аргументу
y=2*x.^2-4*x-3;   %обчислення першої функції
plot(x,y)               %побудова першої функції у першому підвікні
grid on                 %активація сітки
subplot(2,3,2)    %відкриття другого підвікна
y=-5*x.^2+8*x-3; %обчислення другої функції
plot(x,y), grid     %побудова другої функції у другому підвікні
subplot(2,3,3);   % відкриття третього підвікна
y=x.^2+3*x-1;           % обчислення третьої функції
```

```
plot(x,y),grid      %побудова третьої функції у третьому підвікні
subplot(2,3,4)      % відкриття четвертого підвікна
y=3*x+10;           % обчислення четвертої функції
plot(x,y), grid     %побудова четвертої функції у четвертому підвікні
subplot(2,3,5)      % відкриття п'ятого підвікна
y=sin(x);           % обчислення п'ятої функції
plot(x,y),grid      %побудова п'ятої функції у п'ятому підвікні
subplot(2,3,6)      % відкриття шостого підвікна
y=cos(x);           % обчислення шостої функції
plot(x,y),grid      %побудова шостої функції у шостому підвікні
```

Результат виконання програми показаний на рис. 3.5.

У кожному з графічних підвікон можна робити підписи осей, графіків, змінювати типи ліній і т.п. Якщо виникла помилка і необхідно перебудувати який-небудь графік, то потрібно знову звернутися до конкретного підвікна командою subplot і далі провести необхідні зміни.

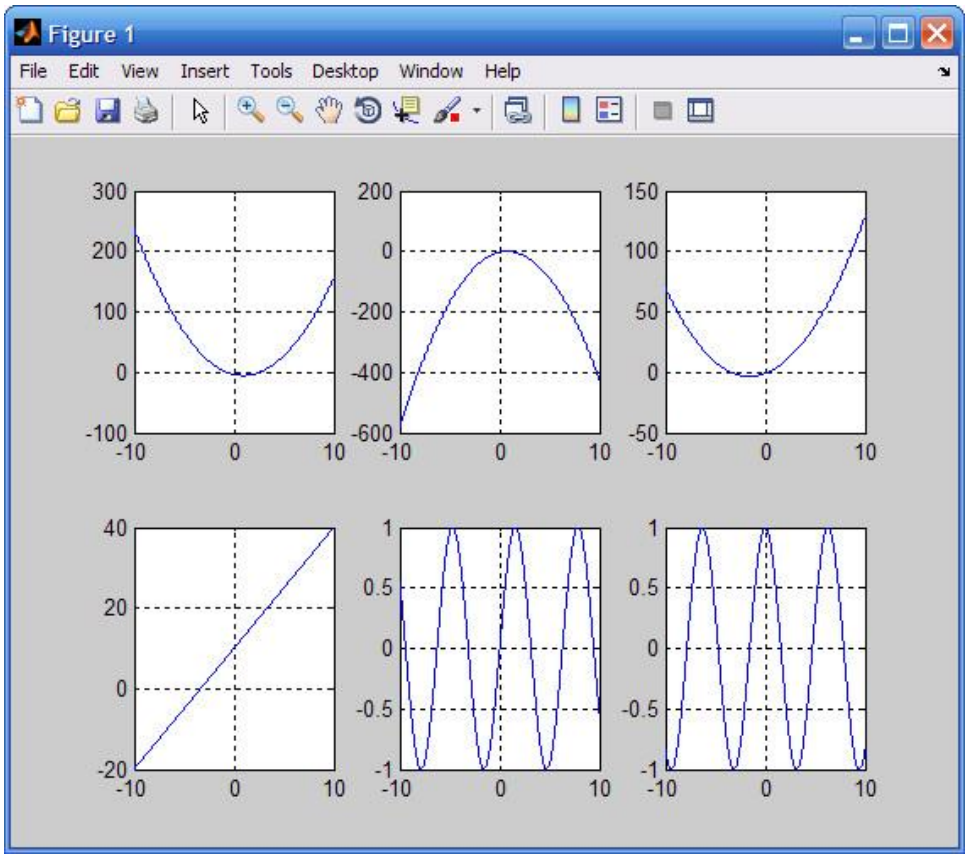


Рис. 3.5. Приклад побудови графіків у графічних підвікнах

3.7. Побудова графіків у полярних координатах

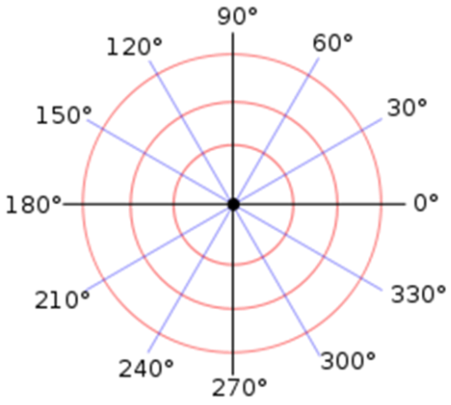


Рис. 3.6. Полярна система координат

Деякі графіки зручно будувати не в Декартових, а у полярних координатах (див. рис. 3.6).

Полярна система координат (СК) – це двовимірна система координат, в якій кожна точка на площині визначається двома числами: ***полярним кутом*** **φ** та ***полярним радіусом*** **ρ**:

$$\rho = f(\varphi). \tag{3.1}$$

Полярна СК задається ***нульовим променем*** або ***полярною віссю***. Точка, з якої виходить цей промінь називається началом координат або ***полюсом***. Радіальна і кутова координати можуть приймати значення від 0 до ∞. Кутова координата може бути як додатною, так і від‘ємною. Додатному куту відповідає поворот полярного радіуса проти годинникової стрілки, а від’ємному – за годинниковою стрілкою. Для побудови графіків у полярних координатах використовують функцію

```
polar (fi,ro)
```

Дуже часто полярні графіки задають так званими параметричними формулами:

$$\begin{cases} x = f_x(R,\alpha), \\ y = f_y(R,\alpha). \end{cases} \tag{3.2}$$

Тоді після розрахунку Декартових координат треба розрахувати ще полярний радіус і полярний кут:

```
ro = hypot (x, y),      fi = atan2(y, x),
```

після чого скористатися функцією `polar`. Графік функції, визначеної параметрично, можна побудувати і функцією `plot (x,y)`, але у цьому разі, щоб не спотворити пропорції графіка, треба доповнити його командою `axis equal`.

Додаткова інформація про обробку та редагування графіків за допомогою інтерфейсу графічних вікон, засоби спеціальної та 3-вимірної графіки відображена у додатках А, Б і В відповідно. Вона призначена для самостійної роботи студентів і може бути використана для поліпшення якості та інформативності графіків і різноманітності їх представлення.

3.8. Контрольні питання та завдання

1. Яка команда використовується для побудови двовимірного графіка?
2. Які типи ліній можна застосувати при побудові графіків? Який тип ліній призначається за замовчанням? Як його змінити?
3. Назвіть основні типи маркерів та поясніть, як вони наносяться.
4. Які кольори можна призначати символами у рядках символів, що записуються після основних аргументів графічних функцій?
5. Що таке *rgb*-палітра? Які значення мають вектори *rgb*-палітри, для кольорів, що можуть бути призначені символами?
6. Які кольори і в якій послідовності призначаються за замовчанням при побудові декількох графіків однією графічною функцією? Яка матриця кольорів це забезпечує?
7. Як вивести на екран імена властивостей графічних вікон, які можна змінювати засобами низькорівневої графіки та значення цих параметрів, що призначаються за замовчанням? Як їх змінити? Які властивості змінюють найчастіше? Наведіть приклади.
8. Назвіть команди підпису осей та формат їх написання.
9. Які команди керування системами координат ви знаєте?
10. Як масштабуються координатні осі? Як нанести легенду? Як керувати її розташуванням?
11. Які команди керування графічними вікнами ви знаєте?
12. Що таке полярна система координат?
13. Яку команду треба виконати, щоб круг, побудований у Декартових координатах не був схожим на еліпс?

4. ОПЕРАЦІЇ З ПОЛІНОМАМИ

4.1. Загальні поняття

Степеневим поліномом (СП) у математиці називають функцію, що має вигляд:

$$P_n(x) = a_0 x^n + a_1 x^{n-1} + \ldots + a_{n-1} x + a_n = \sum_{i=0}^{n} a_i x^{n-i}, \qquad (4.1)$$

де n – степінь поліному;

A=[a_0 , a_1 , ... , a_n] – вектор коефіцієнтів;

x – незалежна змінна.

В пакеті *MATLAB* інформація про СП зберігається у вигляді матриці-рядка коефіцієнтів, упорядкованих за зменшенням степені незалежної змінної, тобто вектору **A**=[2 5 0 –8] відповідає поліном $2x^3 + 5x^2 - 8$.

Слід мати на увазі, що нумерація елементів масивів у MATLAB завжди починається з одиниці.

Вивести СП на екран можна за допомогою функції poly2str

poly2str (A, x_char),

яка перетворює вектор коефіцієнтів A і символьне зображення незалежної змінної x_char у рядок символів (перетворення ***poly****nom to* ***str****ing*), наприклад,

```
>> a=[1 0 -5 2 3];
>> poly2str (a,'x')
```

виводить у командне вікно такий результат:

```
ans =
  x^4 - 5 x^2 + 2 x + 3
```

Застосовуючи функцію disp (var) можна вивести на екран значення змінної без її імені:

disp (poly2str (A, x_char))

наприклад,

```
>> disp (poly2str (a,'x'))
  x^4 - 5 x^2 + 2 x + 3
```

Можна доповнити рядок символів, утворюваний функцією poly2str, додатковою символьною інформацією:

```
>> disp (['P', int2str(length(a)-1),'(x) =', poly2str(a,'x')])
P4(x) =   x^4 - 5 x^2 + 2 x + 3
```

В останньому прикладі int2str – функція ***int**eger **to** **str**ing*, що перетворює цілі числа в рядок символів. До речі усі функції перетворення даних в отримують у своїх іменах посередині символ ‘2’, що замінює слово ***to***.

Вивести на екран два СП у вигляді дробу можна за допомогою функції

printsys (B, A, x_char)

де чисельник – поліном з коефіцієнтами B, а знаменник – поліном з коефіцієнтами A. Таку функцію зручно використовувати для виведення передавальних функцій систем (sys походить від system), заданих у поліноміальній формі, наприклад,

```
>> a=[2 4 -3 -1 0 7]; b=[4 2 6 9 -1 2 1];
>> printsys(a,b,'s')
num/den =
        2 p^5 + 4 p^4 - 3 p^3 - 1 p^2 + 7
  ---------------------------------------------------------------
  4 p^6 + 2 p^5 + 6 p^4 + 9 p^3 - 1 p^2 + 2 p + 1
```

4.2. Розрахунок значення степеневого поліному

Поліном (4.1) можна перетворити до вигляду:

$$P_n(x)=(...(((a_0x+a_1)x+a_2)x+a_3)x+...+a_n)\,. \qquad (4.2)$$

Алгоритм вирахування $P_n(x)$, складений на підставі виразу (4.2), називається ***схемою Горнера***. У відповідності до цієї схеми поліном *i*-го порядку обчислюється за формулою:

$$P_i = P_{i-1}x + a_i\,. \qquad (4.3)$$

Якщо покласти $P_0 = a_0$ та виконати операцію (4.3) n разів при $i = 1,2,..,n$, то можна отримати бажане значення.

У математиці доведено, що для поліномів загального вигляду не можна побудувати алгоритм більш ощадний у розумінні кількості операцій (*n* сумувань та *n* множень), ніж схема Горнера.

У *MATLAB* значення СП з коефіцієнтами A вираховує функція

Y = polyval (A, X),

де X – точки, у яких треба обчислити значення СП (можуть задаватися скалярною величиною або масивом будь-якого розміру);

Y – масив значень степеневого поліному.

Розмірність і розмір вихідної змінної Y співпадає з відповідними параметрами вхідного аргументу X, наприкад,

```
>> y = polyval ([ 2 5 0 -8 4], [2 1]);
y =
     60     3
```

тобто у поліном $2x^4+5x^3-8x+4$ по черзі підставлено: $x_1=2$ та $x_2=1$; в результаті отримано $y_1=2\cdot2^4+5\cdot2^3-8\cdot2+4=60$ та $y_2=2\cdot1^4+5\cdot1^3-8\cdot1+4=3$.

4.3. Операції зі степеневими поліномами

Операції з СП складаються з двох частин: 1) формування нового СП, який є результатом операції, тобто розрахунок його коефіцієнтів; 2) розрахунок значень нового СП при заданих значеннях незалежної змінної.

4.3.1. Алгебраїчне підсумовування поліномів

Нехай ми маємо два СП: $P_n(\mathbf{A},x)$ та $P_m(\mathbf{B},x)$. А результатом деякої операції є ще один СП $P_k(\mathbf{C},x)$.

Для ***алгебраїчного підсумовування поліномів*** $P_n(\mathbf{A},x)$ та $P_m(\mathbf{B},x)$ треба спочатку виконати відповідну операцію над масивами коефіцієнтів **A** та **B**:

$$P_k(\mathbf{C},x)=P_n(\mathbf{A},x)\pm P_m(\mathbf{B},x),$$

де k = max (m,n) – порядок результуючого поліному.

Зважаючи на те, що поліноми-аргументи в загальному випадку мають різний розмір, попередньо треба доповнити менший за розміром масив до

більшого нулями зліва. Функція формування коефіцієнтів **C** поліному, який є алгебраїчною сумою поліномів з масивами коефіцієнтів **A** та **B** може мати вигляд:

```
function C=polysum(A,B)
   m =length(A);     n =length(B);
   if m>n, r= m-n; B=[zeros(1,r),B];
     elseif n>m, r=n-m; A=[zeros(1,r),A];
   end
   C=A+B;
end
```

Приклад використання функції polysum:

```
>> p=[1 2 3 5 0 6]; q=[5 4 1 2];
>> poly2str(p,'x')
ans =
  x^5 + 2 x^4 + 3 x^3 + 5 x^2 + 6
>> poly2str(q,'x')
ans =
  5 x^3 + 4 x^2 +  x + 2
>> s=polysum(p,q);  % Підсумовування СП
>> r=polysum(p,-q); % Віднімання СП
>> poly2str(s,'x')
ans =
  x^5 + 2 x^4 + 8 x^3 + 9 x^2 +  x + 8
>> poly2str(r,'x')
ans =
  x^5 + 2 x^4 - 2 x^3 +  x^2 - 1 x + 4
```

Існує ще один підхід для розв‘язання поставленої задачі. Він полягає у перетворенні векторів коефіцієнтів степеневих поліномів у символьні (*symbolic*) вирази за допомогою функції poly2sym, виконання з ними необхідної операції та заключного зворотного перетворення результуючого символьного виразу у вектор коефіцієнтів за допомогою функції sym2poly.

```
>> a = [2 3 4];
>> b = [5 2 6 9];
>> Pa = poly2sym (a)
Pa =
2*x^2 + 3*x + 4
>> Pb = poly2sym (b)
Pb =
5*x^3 + 2*x^2 + 6*x + 9
>> Pc = Pa + Pb
Pc =
```

```
5*x^3 + 4*x^2 + 9*x + 13
>> c = sym2poly (Pc)
c =
   5    4    9   13
```

Для виконання більш складних операцій MATLAB має вбудовані функції conv, deconv, polyder та деякі інші.

4.3.2. Множення поліномів

C = conv(A,B) – обчислює коефіцієнти СП, який є ***добутком двох поліномів*** з векторами коефіцієнтів A і B , тобто

$$P_{m+n}(\mathbf{C}, x) = P_m(\mathbf{A}, x) \cdot P_n(\mathbf{B}, x) \qquad (9.4)$$

Приклад використання функції conv:

$$\left(x^3+2x^2+5\right)\cdot\left(2x^2+3x+1\right)=$$
$$=2x^5+3x^4+x^3+4x^4+6x^3+2x^2+10x^2+15x+5=$$
$$=2x^5+7x^4+7x^3+12x^2+15x+5$$

```
>> a=[1 2 0 5];  b=[2 3 1];  conv(a,b)
ans =
   2    7    7   12   15    5
>> poly2str(conv(a,b),'x')
ans =
  2 x^5 + 7 x^4 + 7 x^3 + 12 x^2 + 15 x + 5
```

4.3.3. Ділення поліномів

[D,R] = deconv(A,B) – обчислює коефіцієнти полінома D, який є ***часткою від ділення*** полінома з вектором коефіцієнтів A на поліном з вектором коефіцієнтів B; R – коефіцієнти СП, який є остачею від ділення. При цьому справедливе відношення A = conv(D,B) + R .

Приклад використання функції deconv:.

Ділене | *Дільник*

$$6x^4-5x^3+6x^2-2x+1 \quad \big| \quad 3x^3+4x^2-2x-3$$
$$\underline{6x^4+8x^3-4x^2-6x} \qquad 2x-4\tfrac{1}{3} - \textit{Результат}$$
$$-13x^3+10x^2+4x+1$$
$$\underline{-13x^3-17\tfrac{1}{3}x^2+8\tfrac{2}{3}x+13}$$
$$27\tfrac{1}{3}x^2-4\tfrac{2}{3}x-12 - \textit{Залишок}$$

```
>> a=[6 -5 6 -2 1]; b=[3 4 -2 -3];
>> [d,r]=deconv(a,b);
>> poly2str(d,'x')
ans =
  2 x - 4.3333
>> poly2str(r,'x')
ans =
  27.3333 x^2 - 4.6667 x – 12
```

4.3.4. Диференціювання поліномів

B = polyder(A) – обчислює коефіцієнти B полінома, який є похідною від полінома з коефіцієнтами A, тобто

$$P_{n-1}(\mathbf{B},x)=\frac{dP_n(\mathbf{A},x)}{dx}. \qquad (4.5)$$

C = polyder(A,B) – обчислює коефіцієнти C полінома, що утворюється при диференціюванні добутку поліномів з векторами коефіцієнтів A і B (C = polyder (conv(A,B)):

$$P_{m+n-1}(\mathbf{C},x)=\frac{d(P_m(\mathbf{A},x)\cdot P_n(\mathbf{B},x))}{dx}. \qquad (4.6)$$

[Q,D] = polyder(A,B) – обчислює коефіцієнти поліномів у чисельнику Q та у знаменнику D виразу, що утворюється при диференціюванні частки двох поліномів з векторами коефіцієнтів A і B. Отже, згідно з правилами визначення похідних від частки двох функцій:

Q = conv (B,polyder(A)) - conv(A,polyder(B)), D = conv (B,B).

Приклад використання функції polyder:

```
>> a=[5 4 -7 4 3];
>> disp (['P(x) = ', poly2str(a,'x')
    P(x) = 5 x^4 + 4 x^3 - 7 x^2 + 4 x + 3
>> b= polyder(a) ;  disp (['dP(x)/dx = ', poly2str(b,'x')
    dP(x)/dx = 20 x^3 + 12 x^2 - 14 x + 4
>> c=[6 -2 -1 12];
>> [p,q]=polyder(a,c)
p =
   30   -20    19   184   105  -156    51
q =
   36   -24    -8   148   -47   -24   144
>> printsys(p,q,'x')
```

```
num/den =
  30 x^6 - 20 x^5 + 19 x^4 + 184 x^3 + 105 x^2 - 156 x + 51
  ------------------------------------------------------------------------
   36 x^6 - 24 x^5 - 8 x^4 + 148 x^3 - 47 x^2 - 24 x + 144
```

4.4. Зв‘язок між коефіцієнтами степеневих поліномів та їх нулями

Нулями степеневих поліномів є корені поліноміальних (нелінійних алгебраїчних) рівнянь:

$$a_0 x^n + a_1 x^{n-1} + \ldots + a_{n-1} x + a_n = 0. \qquad (4.7)$$

Рівняння (4.7) має *n* коренів, які можуть мати як дійсні, так і комплексні значення. Якщо всі коефіцієнти СП – є дійсними числами, то кожний із комплексних коренів матиме комплексно-спряжену пару.

Розрахунок коренів нелінійних алгебраїчних рівнянь в пакеті *MATLAB* здійснює функція

```
X = roots(A),
```

де A – вектор коефіцієнтів СП, X – вектор-стовбець коренів.

Зворотна операція полягає у визначенні коефіцієнтів полінома за його нулями, що базується на формулі

$$P_n(x) = a_0 x^n + a_1 x^{n-1} + \ldots + a_{n-1} x + a_n = a_0 (x - x_1)(x - x_2)\ldots(x - x_n). \qquad (4.8)$$

Окремим випадком формули (4.8) є теорема Вієта для квадратного рівняння. Після розкриття скобок можна отримати вирази для кожного коефіцієнта СП через його нулі. Таку операцію в пакеті *MATLAB* здійснює функція

```
An=poly(X)
```

результатом виконання якої є коефіцієнти полінома у рівнянні (4.7), унормовані за коефіцієнтом при вищому степені незалежної змінної, тобто

$$\mathbf{A}_{\text{н}} = [1,\ a_1/a_0,\ \ldots,\ a_n/a_0]. \qquad (4.9)$$

Нормування виконується, тому що зворотна задача має безкінечну кількість розв‘язків, які отримуються множенням СП з коефіцієнтами (4.9) на будь-яке число.

```
% Приклад використання функцій roots та poly:
>> a=[2 5 6 -3 2];
>> x = roots(a)
x =
   -1.5054 + 1.4281i
   -1.5054 - 1.4281i
    0.2554 + 0.4087i
    0.2554 - 0.4087i
>> a1 = poly(x)
a1 =
    1   2.5   3   -1.5   1
>> a2 = a(1)*poly(x)
a2 =
    2   5   6   -3   2
```

Розглянуті функції доцільно застосовувати при еквівалентних перетвореннях структурних схем з динамічними ланками, поданими у віигляді передавальних функцій у поліноміальній формі, для побудови частотних характеристик та при дослідженні систем на сталість.

4.5. Застосування операцій над поліномами в теорії автоматичного керування

Розглянуті операції над поліномами можна застосовувати в теорії автоматичного керування при еквівалентних перетвореннях структурних схем з динамічними ланками, поданими у вигляді передавальних функцій у поліноміальній формі, при синтезі систем автоматичного керування поліноміальними методами, при розрахунку частотних характеристик та при дослідженні систем на сталість.

Розглянемо декілька прикладів.

Приклад 4.1. *Знайти еквівалентну передавальну функцію системи, структурна схема якої подана на рис. 4.1, та перевірити її на сталість методом визначення полюсів.*

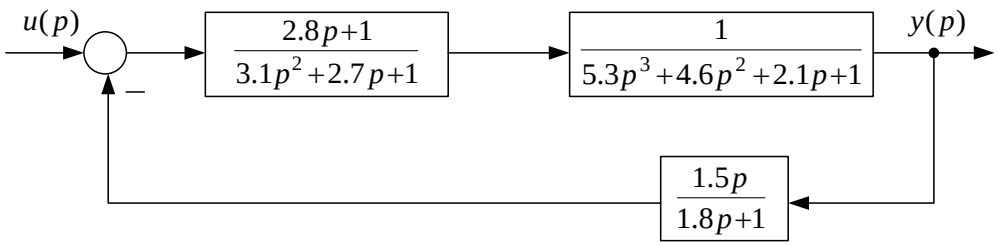


Рис. 4.1. Структурна схема до прикладу 4.1

Введемо позначення

$$W_1(p)=\frac{H_1(p)}{G_1(p)}=\frac{2.8p+1}{3.1p^2+2.7p+1.2},$$

$$W_2(p)=\frac{H_2(p)}{G_2(p)}=\frac{1}{5.3p^3+4.6p^2+2.1p+1}, \qquad W_3(p)=\frac{H_3(p)}{G_{31}(p)}=\frac{1.5p}{1.8p+1}.$$

Тоді

$$W(p)=\frac{H(p)}{G(p)}=\frac{W_1(p)W_2(p)}{1+W_1(p)W_2(p)W_3(p)}=\frac{H_1(p)H_2(p)G_3(p)}{H_1(p)H_2(p)H_3(p)+G_1(p)G_2(p)G_3(p)}.$$

На базі наведеної інформації розробляємо програму:

```
clc, clear all, close all
H1 = [2.8 1];  G1 = [3.1 2.7 1];
H2 = 1;  G2 = [5.3 4.6 2.1 1];
H3 = [1.5 0];  G3 = [1.8 1];
H4=conv(H1,H2);
G4=conv(G1,G2);
H=conv(H4,G3);
G=polysum(conv(H4,H3), conv(G4,G3));
printsys(H,G,'p')
P=roots(G)
compass(P)
figure, plot(real(P), imag(P),'x','MarkerSize',20), hold on
for k=1:length(P)
   plot([0;real(P(k))],[0;imag(P(k))],'--')
end
axis equal, grid on
xlabel('Re(p)'), ylabel('Im(p)')
```

В результаті виконання цієї програми отримуємо такі результати:

```
num/den =
                          5.04 p^2 + 4.6 p + 1
  ---------------------------------------------------------------------------------------
  29.574 p^6 + 67.856 p^5 + 72.184 p^4 + 48.296 p^3 + 26.21 p^2 + 8.1 p + 1
P =
  -0.8440 + 0.4507i
  -0.8440 - 0.4507i
  -0.0124 + 0.6270i
  -0.0124 - 0.6270i
  -0.2908 + 0.0967i
  -0.2908 - 0.0967i
```

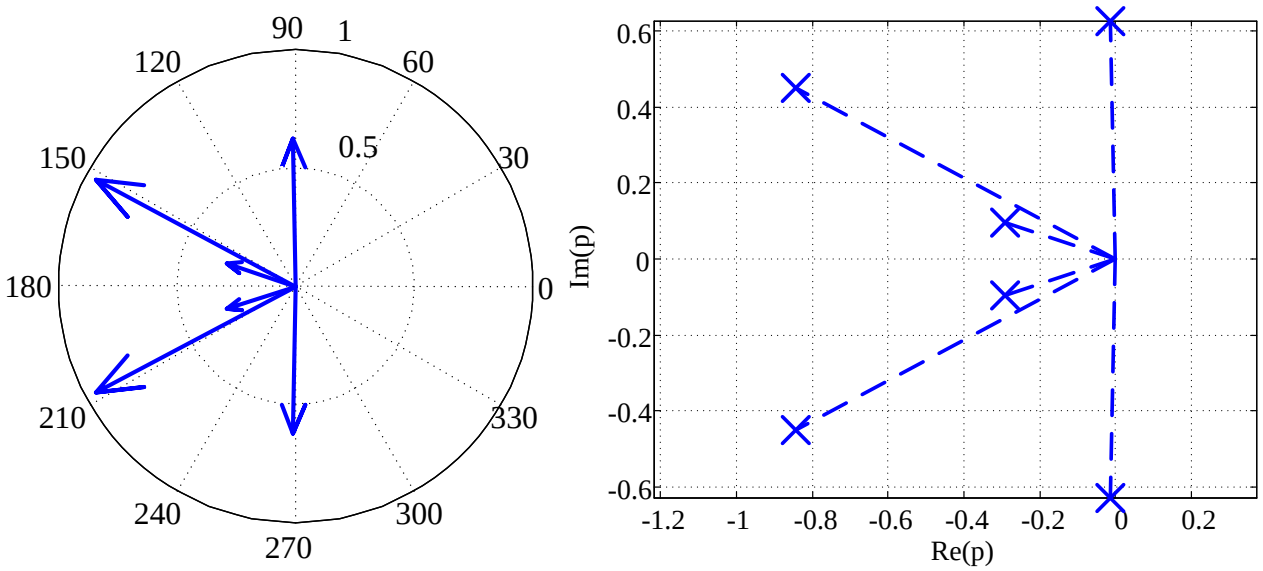


Рис. 4.2. Графіки до прикладу 4.1

Як бачимо, усі полюси мають від'ємні дійсні частини, з чого випливає, що досліджувана система є сталою. Однак домінуюча пара полюсів знаходиться майже на межі стійкості. Тому перехідні процеси у цій системі будуть мати коливальний характер з дуже повільним затуханням.

4.6. Контрольні питання та завдання

1. Що таке степеневий поліном? Якими параметрами він характеризується?
2. У якому порядку треба розташувати коефіцієнти полінома для подальшої роботи з ним у середовищі пакета *MATLAB*?
3. Як вивести на екран вираз для степеневого полінома?
4. Виконайте вручну з поясненнями операції:

   conv([1,2],[3,4]), polyder([3,-2,5.-8]), polyval([1,2,3],[4,5]),
   poly2str(C,'x')],
   roots([0.5 1.5 1]), poly(roots([0.5 1.5 1])
5. Порівняйте отримані Вами результати з результатами, що надає комп'ютер.
6. Які задачі теорії автоматичного керування потребують виконання операцій над степеневими поліномами?
7. Виконати завдання, наведене у прикладі 4.1, для деякого фрагменту структурної схеми, заданої у курсовій роботі з дисципліни «Теорія автоматичного керування», використовуючи операції над поліномами.

8. Знайти нулі та полюси отриманої після перетворень підсистеми та зобразити їх на комплексній площині (нулі кружечками, а полюси хрестиками.

5. ОСНОВНІ ПРАВИЛА РОБОТИ В СЕРЕДОВИЩІ ДОДАТКУ *Simulink* ПАКЕТУ *MATLAB*

5.1. Запуск *Simulink*. Огляд бібліотек

Simulink можна запустити з командної строки пакета *MATLAB* однойменною командою

>>simulink

де >> – запрошення *MATLAB* до вводу команди, або спеціальною кнопкою *Simulink Library* на панелі інструментів, що показана на рис. 5.1.

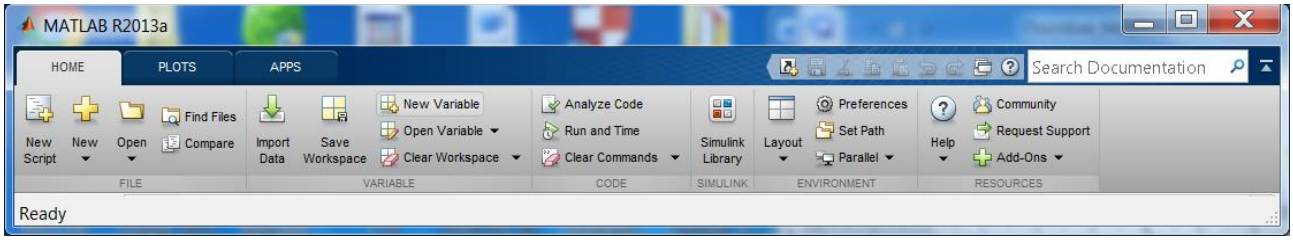


Рис. 5.1. Фрагмент командного вікна системи *MATLAB*

При цьому на екрані відкривається вікно з бібліотеками блоків, представлене на рис. 5.2, яке має назву *Simulink Library Browser (Навігатор Бібліотек)*.

Кожну бібліотеку можна розкрити подвійним натиском миші на піктограмі бібліотеки в правій частині вікна або вибором назви бібліотеки в лівій частині вікна. При цьому піктограми блоків обраної бібліотеки заміщають піктограми бібліотек.

Натиском правою кнопкою миші на імені бібліотеки через меню, що випадає, можна відкрити будь-яку бібліотеку у вигляді, звичному для користувачів більш старих версій (*MATLAB 4.0 - MATLAB 5.1*, *Simulink2 - Simulink3*).

Simulink зі старим інтерфейсом можна також відкрити з командної строки командою

>>simulink3

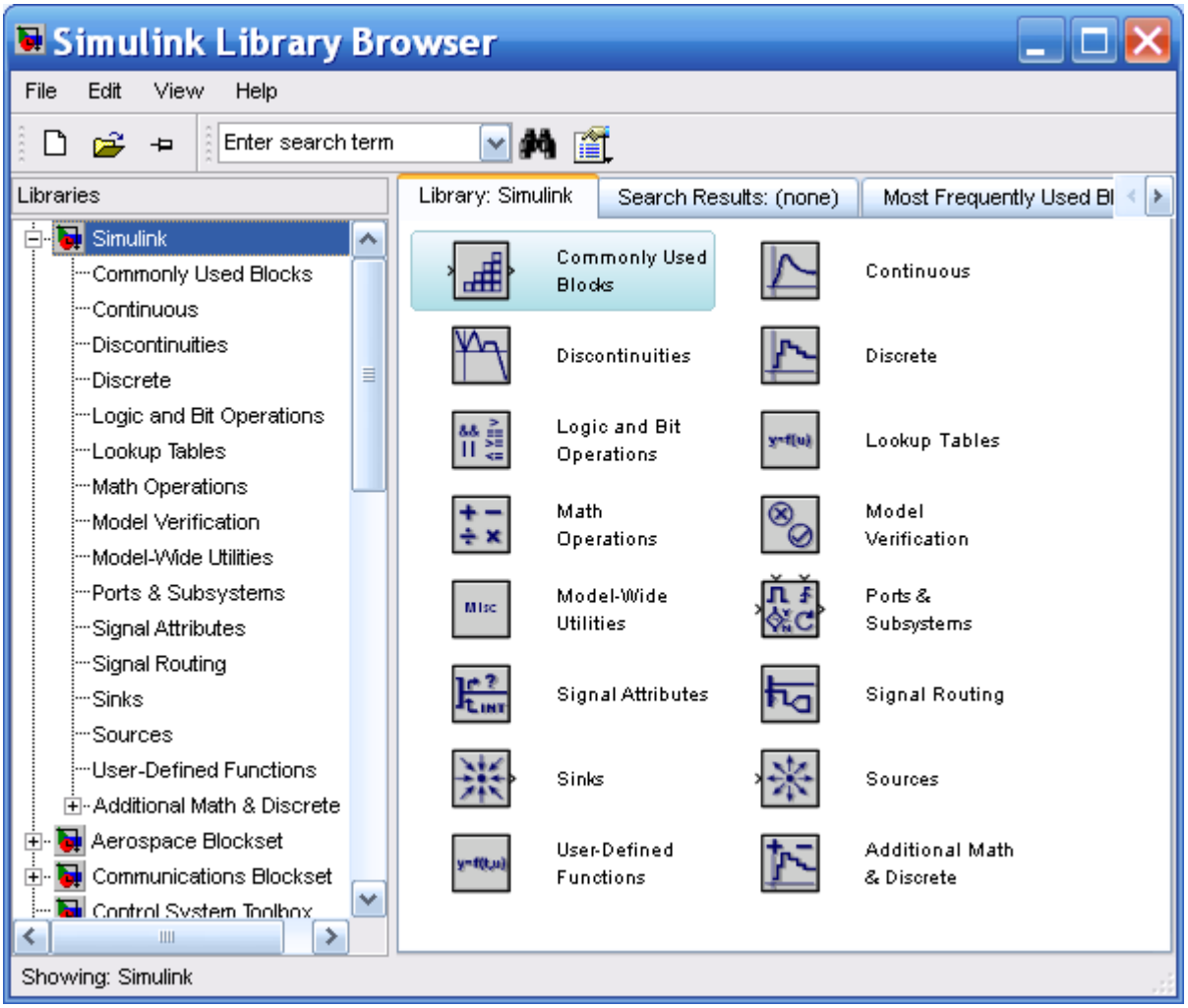


Рис. 5.2. Вікно *Simulink Library Browser*

При цьому відкривається вікно, зображене на рис. 5.3.

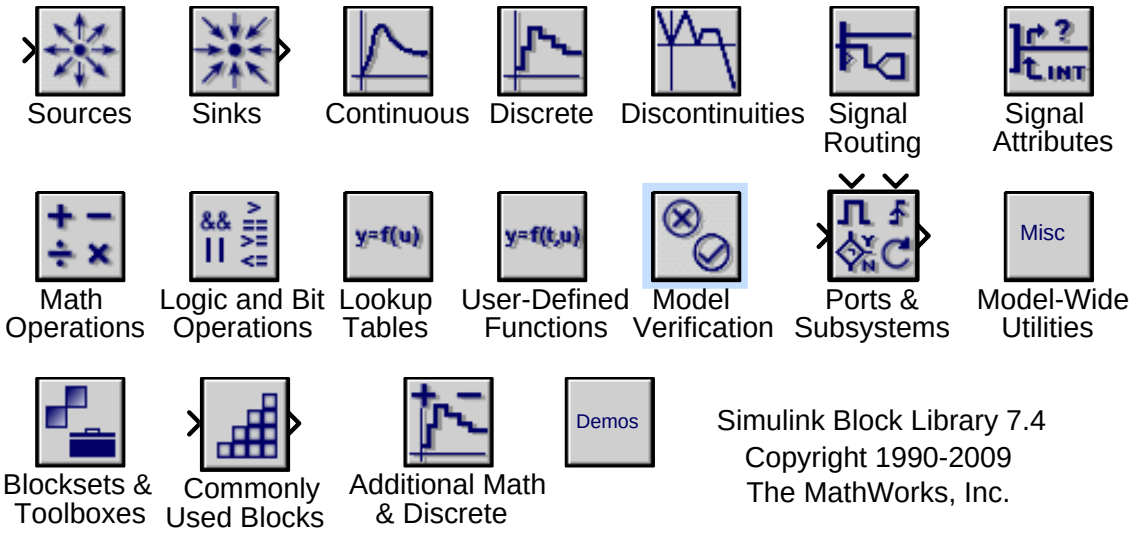


Рис. 5.3. Вікно бібліотек блоків *Simulink*

З цього вікна кожну бібліотеку можна розкрити подвійним натиском миші по її піктограмі.

До основних *Simulink-бібліотек* відносяться

Sources – джерела вхідних сигналів;

Sinks – блоки реєстрації та візуалізації;

Continuous – неперервні лінійні динамічні ланки;

Math Operations – математичні операції;

Ports & Subsystems – порти та підсистеми.

При подвійному натиску мишею по піктограмі блоку відкривається вікно введення його параметрів, у якому відображуються ім‘я блоку, його тип, короткий опис і рядки введення параметрів із супроводжуючими їх запитами. Більш докладну інформацію про блок можна побачити, звернувшись до функції *Help*. Піктограма блоку найчастіше відображує його динамічні або статичні властивості і реагує на зміну параметрів. Як приклад, на рис. 5.4 наведено вікно введення параметрів блоку *Transfer Function.*

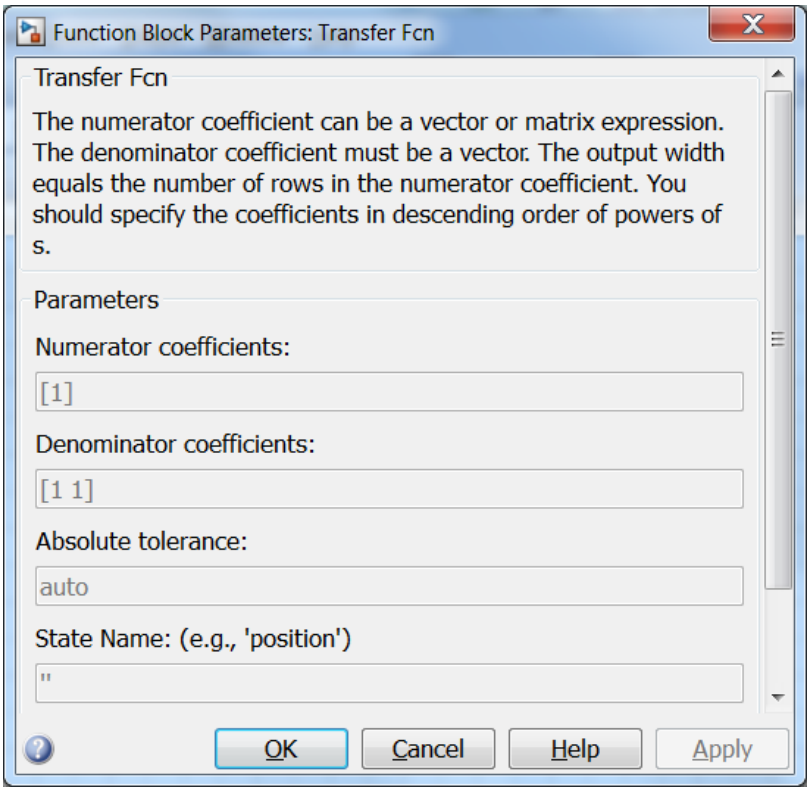


Рис. 5.4. Вікно введення параметрів блока *Transfer Fcn*

Параметри блоків можуть бути константами, перемінними, функціями або виразами, припустимими в *MATLAB*. Будь-які перемінні, від яких залежить параметр, повинні бути визначені в робочому середовищі до початку процесу моделювання, а іншому випадку *Simulink* сигналізує про помилку в цьому блоку.

Клавіша Apply дозволяє оцінити результат зміни параметрів, не закриваючи вікна їхнього введення, ***клавіша OK*** закриває вікно введення параметрів із запам‘ятовуванням виконаних змін, ***клавіша Cancel*** закриває вікно введення параметрів зі скасуванням внесених змін.

Для знайомства з можливостями додатку *Simulink* та особливостями роботи деяких блоків можна скористуватися демонстраціями, які зручно запускати з вікна *MATLAB Demo Window*. Воно з’являється на екрані при виконанні команди

```
demo
```

або

```
demo simulink
```

Для збереження моделей, як і інших файлів користувача, у *Windows* передбачена папка *C:\Users\...\Documents\Matlab*. ***Усі компоненти шляху до файлів MATLAB можуть складатися тільки з латинських букв, цифр і знаку підкреслення та починатися з латинської букви.***

Файли моделей і бібліотек додатка *Simulink* повинні мати маску **.mdl* або **.slx*. Починаючи з версії *MATLAB-2012a*, розширення *mdl* замінено на *slx*. Вони є текстовими файлами особливої структури і містять інформацію про структуру та параметри моделей, достатню для їхнього графічного відображення і моделювання.

Відкрити вікно для створення нової моделі можна не тільки командою *File → New → Model (^N)* будь-якого *Simulink-вікна*, але й кнопкою *New → Simulink → Simulink Model* пакета *MATLAB*.

Вже існуючі текстові файли і моделі відкриваються командою *Open* з вікон бібліотек та моделей *Simulink*, а також з командного рядка *MATLAB* введенням у ній імені файлу без розширення. ***Оскільки MATLAB не розрізняє імена змінних і різноманітних файлів, то всі вони повинні мати оригінальні імена***, які мають відрізнятися також від зарезервованих імен системи *MATLAB.*

Якщо при спробі відкрити модель або виконати яку-небудь програму з командного рядка на екран виводиться повідомлення «*Undefined function or variable 'name'»* (невизначена функція або перемінна), то це означає, що користувач або помилився при наборі імені файлу, або цей файл знаходиться в недоступній для системи *MATLAB* директорії (у більш пізніх версіях *MATLAB* приєднання нових директорій до списку доступних виконується системою програмування автоматично при зверненні до будь-якого файлу цієї директорії або після відповіді на питання «*Change current Folder*?»).

5.2. Основні команди *Simulink*

5.2.1. Команди, що виконуються з вікна *Simulink Library Browser*

Меню вікна *Simulink Library Browser* містить такі функції: *File* – робота з файлами, *Edit* – редагування, *View* – вигляд вікна; *Help* – допомога. Кожну функцію цього меню можна вибрати мишею або клавішами <→>, <←> і <*Enter*>.

Функція File містить наступні операції: *New...* ▸ – створити вікно для введення нової моделі (*Model (^N)*) або бібліотеки (*Library*), *Open... (^O)* – відкрити модель, *Close* – закрити вікно *Simulink Library Browser*, *Preferences…* – переваги (настроювання середовища).

Команда Preferences дозволяє виконувати настроювання середовища *MATLAB* і *Simulink*. При відсутності достатнього доступу нею краще не користуватися.

5.2.2. Команди, що виконуються з вікон *Simulink*-моделей

Меню вікон *Simulink*-моделей відрізняється від меню вікна *Simulink Library Browser* і отримує функції та кнопки, кількість та перелік яких

обирається у вкладці *Editor Defaults* вікна *Simulink Properties*. Одна з можливих конфігурацій вікна *Simulink*-моделі наведена на рис. 5.5.

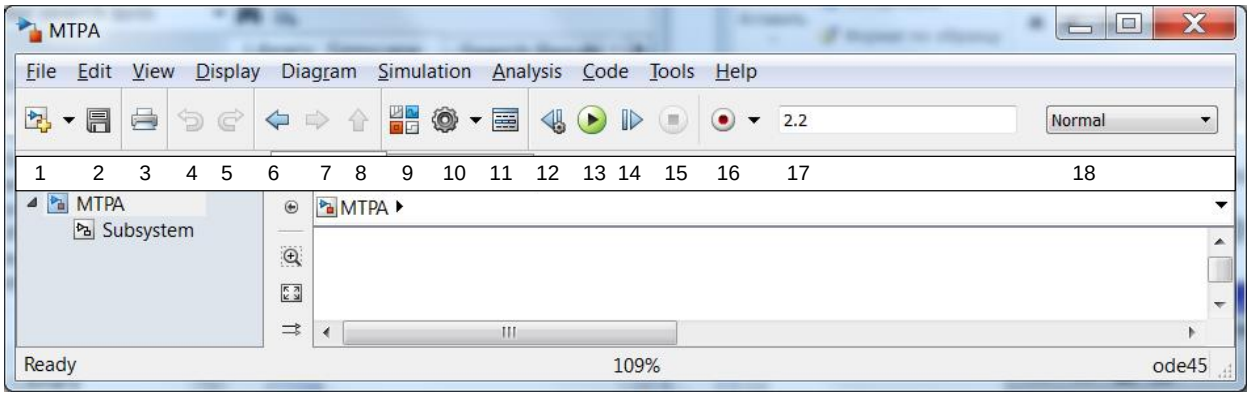


Рис. 5.5. Меню та кнопки вікна *Simulink*-моделі

Перші п'ять кнопок виконують стандартні операції *Windows:* 1 – *New Model* (відкрити нове *Simulink*-вікно), 2 – *Save* (зберегти), 3 – *Print* (надрукувати), 4,5 – Undo, Redo (скасувати, відновити результат останнього редагування).

Інші клавіши виконують специфічні операції *Simulink*: 13 – *Run* (почати моделювання); 15 – *Stop simulation* (зупинка моделювання); 17 – *Simulation Stop Time* (встановлення часу моделювання).

У полі заголовка вікна рис. 5.5 відображено шлях до моделі у файловій системі. Символ “*” у кінці шляху означає, що після редагування моделі не виконано її запис у файл.

Функція File вікна моделі містить такі операції роботи з файлами: *New* ▸ – створити вікно для введення нової моделі (*Model ^N*) або бібліотеки (*Library*); *Open... (^O)* – відкрити модель; *Close (^W)* – закрити модель; *Save (^S)* – зберегти модель у колишньому файлі; *Save As..* – зберегти модель у новому файлі; *Model Properties* ▸ – властивості моделі; *Exit MATLAB* – вихід із системи *MATLAB*.

Вікно Model Properties, яке відкривається при виборі відповідної операції, містить вкладки *Main, Callbacks, History* і *Description*. Вкладка ***History*** містить дані про модифікацію моделі.

Вкладка Callbacks (див. рис. 5.6) дозволяє визначити команди, що будуть автоматично виконані перед завантаженням моделі (*Model pre-load function:*), при її ініціалізації (*Model initialization function:*), перед стартом моделювання (*Simulation start function:*), після його закінчення (*Simulation stop function:*) і перед збереженням моделі у файлі (*Model pre-save function:*).

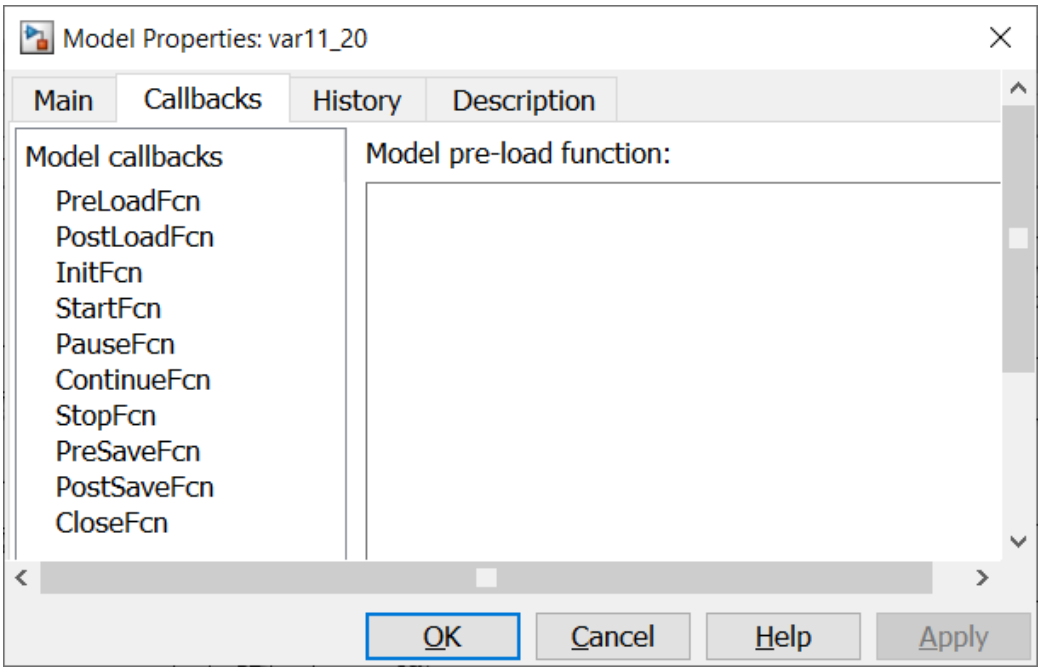


Рис. 5.6. Вкладка *Callbacks* вікна *Model Properties*

Функція Edit вікна моделі містить як стандартні *Windows*-операції редагування, так і деякі специфічні операції: *Undo (^Z)* – відмінити останню зміну; *Redo (^Y)* – відновити останню зміну; *Cut (^X)* – переміщення обраних об'єктів у буфер; *Copy (^C)* – копіювання обраних об'єктів у буфер; *Paste (^V)* – копіювання змісту буфера в обране місце графічного вікна; *Delete (Del)* – вилучення обраних об'єктів; ***Select All (^A)* – вибір всіх об'єктів в активному вікні; *Copy Current View to Clipboard* – копіювання моделі в буфер**.

Функція View вікна моделі містить **команди керування масштабом зображення *Zoom in, Zoom out, Fit system to view* і *Normal (100)***.

Найбільш споживаною операцією функції ***Display*** є *Signals & Ports*, за допомогою якої виконуються операції *Signal Dimensions, Wide Nonscalar Lines, Port Data Types* та деякі інші.

Важливими операціями функції ***Diagram*** є *Format* та *Rotate & Flip*. До команд форматування моделі належать команди

Font Style – установлення типу та розміру шрифту для текстових написів (для цього треба попередньо виділити блок, а не його ім'я);

Show Block Name – сховати/показати ім'я блоку.

До команд повороту елементів моделі належать команди

Clockwise (^R) – повернути блок на 90° за годинниковою стрілкою,

Flip Block (^I) – повернути блок із праворуч на ліворуч або зверху вниз,

Flip Block Name – змінити положення імені блоку (для горизонтально розташованих блоків воно може розташовуватися вгорі або внизу, а для розташованих вертикально – праворуч або ліворуч).

Функція ***Simulation*** керує наступними режимами:

Model Configuration Parameters...(^E) – установка і редагування параметрів моделювання,

Start (^T) – старт моделювання.

Вікно параметрів моделювання має декілька вкладок (панелей), з яких найчастіше використовують вкладки *Solver* та *Data Import/Export.*

Ці панелі подані на рис. 5.7 та рис. 5.8 відповідно.

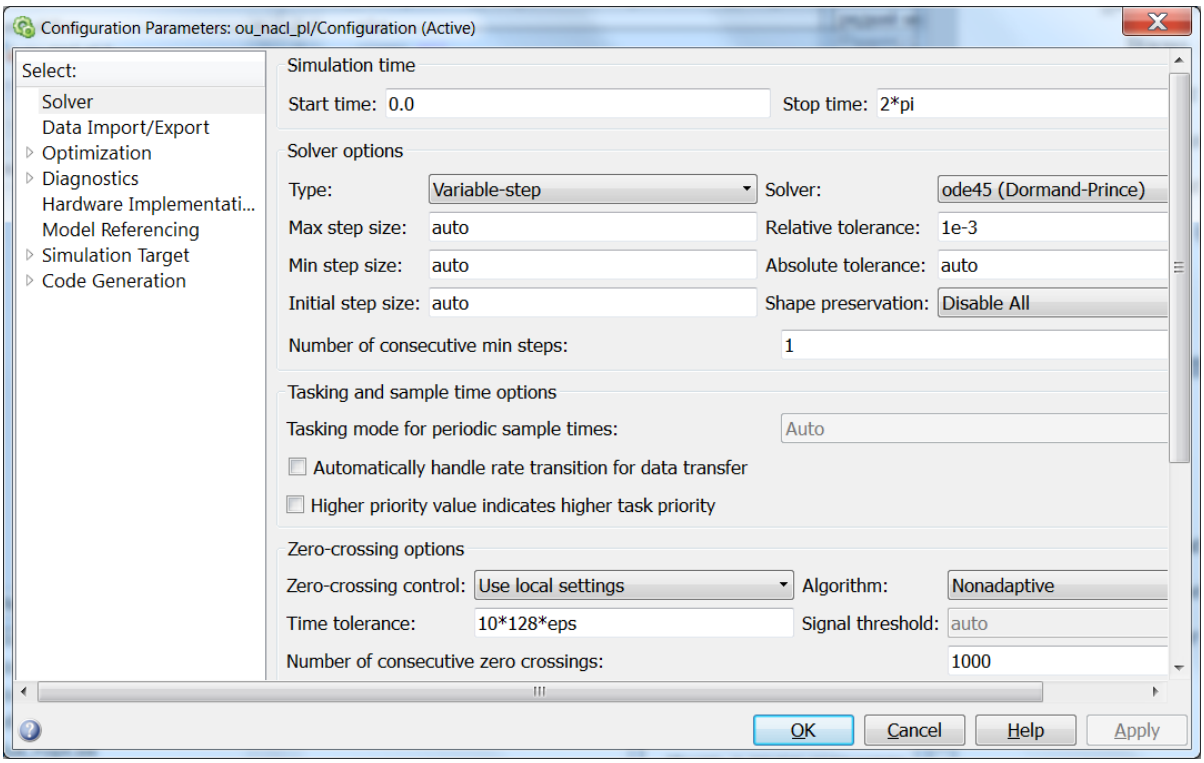


Рис. 5.7. Вкладка *Solver* вікна параметрів симуляції

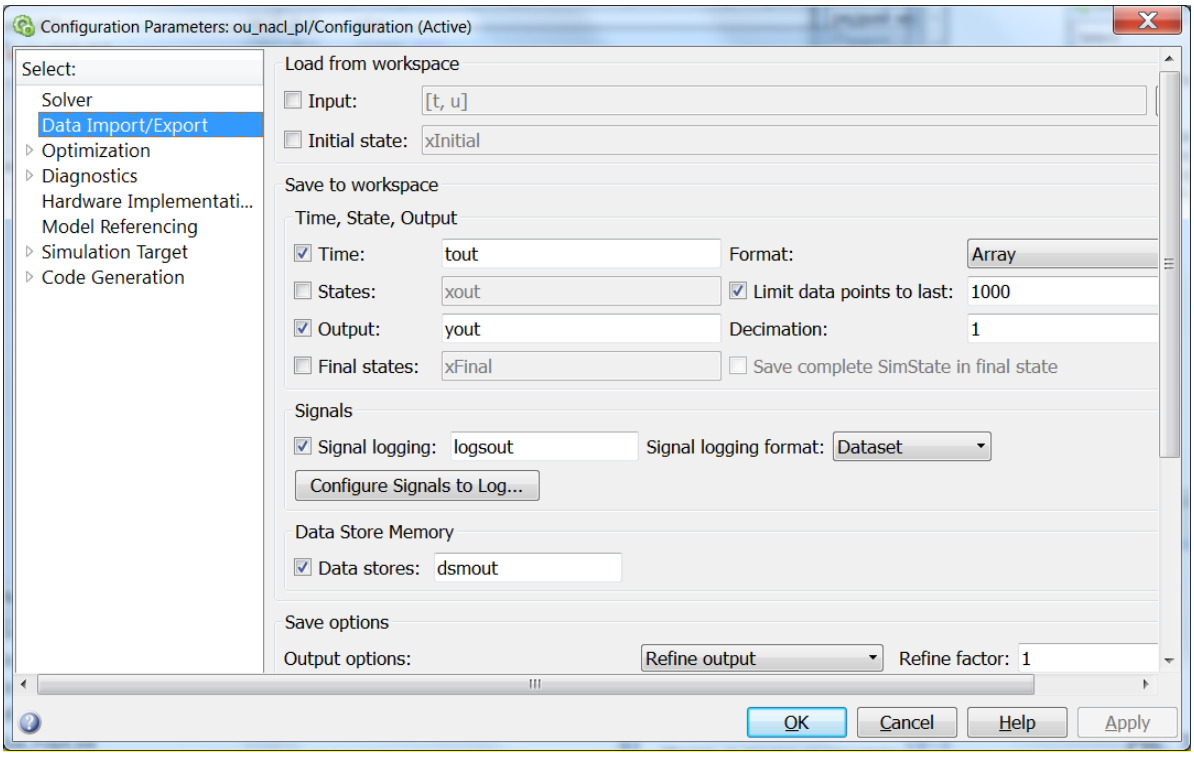


Рис. 5.8. Вкладка *Data Import/Export* вікна параметрів симуляції

Вони визначають метод і параметри симуляції (розв‘язання рівнянь, що описують математичну модель).

Панель Solver передбачає введення таких основних параметрів:

Start time – час початку моделювання;

Stop time – час закінчення моделювання;

Type – тип розв‘язання: *Fixed/Variable-Step* (з фіксованим/змінним кроком);

Solver – метод розрахунку перехідних процесів (для неперервних систем – метод розв‘язання звичайних диференційних рівнянь з початковими умовами).

Серед ***методів розв‘язання ДР з фіксованим кроком (Fixed Step)*** в *Simulink* пропонуються ***методи Рунге-Кутта першого-п‘ятого порядків***, а саме:

*ode*5 – метод Рунге-Кутта 5-го порядку (*Dormand-Prince formula*);

*ode*4 – метод Рунге-Кутта 4-го порядку (*fourth-order Runge-Kutta formula*);

*ode*3 – метод Рунге-Кутта 3-го порядку (*Bogacki-Shampine formula*);

*ode*2 – модифікований метод Ейлера (*improved Euler’s formula / Heun’s method*);

*ode*1 – метод Ейлера (*Euler’s method*).

Для методів з фіксованим кроком треба задати значення цього параметру в полі *Fixed step size*.

Серед ***методів розв‘язання ДР зі змінним кроком (Variable Step)*** в *Simulink* пропонуються:

*ode*45 – комбінований метод Рунге-Кутта (4-5)-го порядку з автоматичним вибором кроку;

*ode*23 – комбінований метод Рунге-Кутта (2-3)-го порядку з автоматичним вибором кроку;

*ode*113 – метод Адамса-Моултона-Бешфорта (метод прогнозу та корекцій змінного порядку);

*ode*15*s* – метод Гіра;

*ode*23*s* – метод Розенброка;

*ode*23*t* – метод трапецій;

*ode*23*tb* – комбінований метод, що використовує на першому етапі формулу трапецій, а на другому – зворотну диференційну формулу другого порядку.

Для методів з фіксованим кроком треба задати значення цього параметру в полі *Fixed step size*.

Для методів зі змінним кроком задаються такі параметри:

Max step size – максимальний крок; в режимі *auto* він розраховується за формулою $h_{\max} = (t_{stop} - t_{start})/50$;

Min step size – мінімальний крок;

Initial step size – початковий крок;

Relative tolerance – відносна точність (за замовченням 10^{-3});

Absolute tolerance – абсолютна точність (за замовченням 10^{-6});

Refine factor – коефіцієнт збільшення кількості точок моделювання (ціле число); для методу *ode*45 рекомендовано значення 4 (максиальне значення – 10), для інших методів – 1.

Вибір методів розв‘язання ДР та їх параметрів є непростою задачею, що потребує від користувача знань в області обцислювальної математики та певного досвіду, але для користувачів-початківців для моделювання лінійних неперервних САК можна рекомендувати використання методу *ode23s* з праметрами, що призначаються за замовчанням.

Підвищити якість візуалізації перехідних процесів за рахунок корекції вектору часу шляхом додавання до нього бажаних точок можна установкою параметра *Output options* на панелі *Data Import/Export* у стан *Produce additional output*. Більшу кількість точок вихідних сигналів моделі можна отримати і зменшенням максимального кроку чисельного інтегрування або параметрів точності *Relative tolerance* та *Absolute tolerance*, але таке рішення менш ефективне з точки зору затрат машинного часу. Для методу *ode*45, що призначається за замовчанням, якість візуалізації можна покращити

збільшенням параметру *Refine factor* (ціле число від 1 до 10). На панелі *Data Import/Export* у першу чергу зверніть увагу на доцільність усунення «прапорця» перед параметром *Limit Data Point to Last*., який в активному стані залишає в розрахунках перехідних процесів при візуалізації тільки 1000 останніх значень.

5.3. Основні прийоми створення та редагування моделей

Перед початком формування структурної схеми необхідно відкрити для неї нове вікно (*Simulink* → *File* → *New* → *Model (^N)* або *MATLAB* → *File* → *New* → *Model*). При цьому відкривається вікно з заголовком "*Untitled*". При запису в файл (*File* → *Save / Save as…*) у якості заголовку вікна буде фігурувати призначене користувачем ім'я файлу, яке за замовченням отримає розширення *slx*.

Вже існуючу модель можна завантажити з меню *Simulink* або *MATLAB* (*File* → *Open*) та із командного рядка *MATLAB* введенням в ній імені файлу, в якому збережена модель, без поширення.

В основу створення та редагування моделей покладено, як і в більшості графічних *Windows*-додатків, ***принцип drag-and-drop (перетягни та покинь)***.

Копіювання блоків із бібліотек або з будь-якої вже існуючої моделі у вікно створеного файлу після їх відкриття виконується мишею за допомогою операції *drag* (тягнути при натиснутій лівій клавіші миші). Аналогічно здійснюється переміщення блоків всередині вікна. Копіювання блоків всередині вікна виконується операцією *drag right* (тягнути при натиснутій правій клавіші миші). При цьому до імені нового блока додається цифра, що відображає порядковий номер копіювання, чим забезпечується унікальність імені кожного блока однієї моделі.

Імена блоків можна змінювати безпосереднім редагуванням. При цьому не можна їх дублювати або залишати блок без імені (ім'я – пустий рядок), але можна сховати ім'я (*Diagram* → *Format* → *Hide Block Name*). Зворотна операція здійснюється командою *Format* → *Show Block Name*. Для завершення

редагування імені треба зробити щиглик мишею зовні поля введення тексту. Після цього ім‘я аналізується системою та чи приймається, чи відкидається нею з виведенням повідомлення. Для зміни шрифту імені (*Format* → *Font*...) необхідно попередньо виділити сам блок, а не його ім‘я.

Іменами можна відмічати і лінії зв‘язку. Для цього необхідно клацнути двічі лівою кнопкою миші по обраній з‘єднувальній лінії та ввести текст в утворене поле. Отриманий у такий спосіб надпис буде, на відміну від блока *Note*, прив‘язаний до лінії зв‘язку.

При пересуванні блока, до якого вже приєднані лінії зв‘язку, останні витягуються або скорочуються для збереження зв‘язків; кінці зв‘язків, не приєднані до блоку, що пересовується не змінюють своє положення. Перемістити блок в межах одного вікна без ліній зв‘язку (висунути блок з моделі) можна операцією <*Shift*>+*drag*.

Для виконання якої-небудь дії з будь-яким об‘єктом моделі (блоком, лінією зв‘язку, сукупністю блоків та/або зв’язків, тобто фрагментом моделі, усією моделлю) його треба спочатку відмітити. Поодинокий об‘єкт виділяється щигликом лівої клавіші миші (*click*). Фрагмент моделі можна виділити такими способами:

- помітити вибірково блоки або зв‘язки, не відпускаючи клавішу *Shift* (<*Shift*> + *click*);
- помітити підряд, тобто. заключити за допомогою мишки у прямокутник;
- помітити все (*Edit* → *Select All* – ^*A*).

Про те, що виділення відбулося, свідчать маленькі незафарбовані квадратики, розташовані в кутах обраних блоків та поблизу кінців обраних зв‘язків.

З поміченим блоком можна виконувати такі операції:

- зміна розміру – *drag* за кут;
- знищення – <*Delete*>;

- усі доступні операції меню *Diagram: Format* (зміни кольорів та шрифтів, приховування імен), *Rotate & Flip* (повороти) та *Mask* (маскування).

Подвійний щиглик по піктограмі блока розкриває текстове вікно для введення його параметрів.

Параметри блока можуть бути подані як в чисельному вигляді, так і у вигляді імен змінних або математичних виразів. В останньому випадку до початку моделювання змінним повинні бути присвоєні значення. Це можна зробити з командного рядка *MATLAB* або виконанням командного файлу або у рядку *Init Fn* вкладки *Callbacks* вікна *Model Properties* (див. рис. 5.6).

Зв‘язки встановлюються між вхідними та вихідними портами блоків. Стрілка на зв‘язку показує напрямок потоку даних. Утворюються зв‘язки у такі способи:

- довільно кусочно-неперервно (*drag* від порту з перериванням цієї операції в бажаних точках зламу);
- з автоматичним розташуванням точок зламу.

Автоматичне розташування точок зламу забезпечується при виконанні операції *drag* від порту до порту без переривання, а також при швидкому з‘єднанні блоків, яке виконується у такий спосіб: помічається початковий блок або блоки, натискається клавіша *Ctrl* і помічається кінцевий блок.

Для ***розгалуження лінії зв‘язку*** використовують операцію *drag right* або ^ *drag left* або *drag* будь-якою клавішею але у зворотному напрямку, тобто від вхідного порту до точки розгалуження.

При проведенні ліній зв‘язку не варто намагатися влучити точно в порт; лінія приєднається до порту, якщо відпустити клавішу миші при находженні графічного курсора всередині блока або поблизу порту (на відстані менше 5 піксель).

З поміченими лініями зв‘язку можна виконувати такі операції:

- паралельне пересування – *drag*, ухопившись за середину сегменту зв‘язку;
- зміна кута між сегментами зв‘язку – *drag*, ухопившись за вузол;

- додатковий злам сегмента зв‘язку – <*Shift*>+ *drag*, ухопившись за бажану точку зламу;
- ділення зв‘язку – *drag*, ухопившись за першу ліву мітку зв‘язку або *drag right* від будь-якої точки зв‘язку;
- знищення – <*Delete*>.

З поміченими фрагментами моделі або з усією моделлю можна виконувати всі ті операції, що і з окремими блоками та/або зв‘язками.

Для того, щоб зробити розташування об‘єктів моделі більш зручним, вікна *Simulink* мають невидиму сітку 5х5 пікселів, до вузлів або до ліній якої прив‘язуються всі об‘єкти. Дискретне пересування виділених об‘єктів можна виконати клавішами <→>, <←>, <↑>, <↓>.

При редагуванні моделі ***графічний курсор змінює свою форму***, сигналізуючи користувачу про ту дію, до виконання якої підготовлена система:

– готовий для наступних дій;

– готовий для нанесення обмежувального прямокутника (виділення групи поруч розташованих об‘єктів);

– готовий для пересування блока, сегмента лінії, або виділеного фрагмента;

– готовий до проведення лінії зв‘язку;

– готовий для перенесення або для створення нової точки зламу лінії;

– готовий до зміни розміру блока.

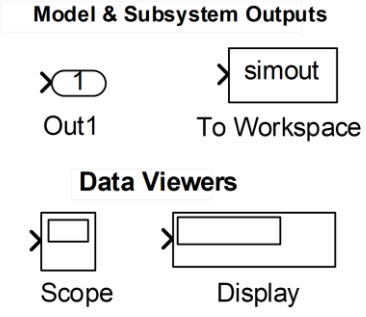


Рис. 5.9. Основні блоки бібліотеки *Sinks*

5.4. Основні засоби реєстрації та візуалізації сигналів у середовищі *Simulink*

Для реєстрації та візуалізації сигналів у *Simulink* існує бібліотека *Sinks*, основні блоки якої подані на рис. 5.9. Вони призначені для запам'ятання результатів моделювання *(To Workspace, Out)*, а також для їх графічного *(Scope)* або чисельного *(Display)* відображення.

Усі вони мають тільки входи і не мають виходів.

Розглянемо параметри, які є спільними для декількох блоків бібліотеки *Sinks*.

Параметр Limit data point to last у блоках *Scope*, *To Workspace*, *Out* визначає максимальну кількість точок для збереження сигналу (відлік ведеться від останньої розрахованої точки). Для того, щоб не втратити інформацію, значення параметра *Limit data point to last* можна установити рівним ∞ (константа *Inf*) або прибрати прапорець, що робить цю опцію активною.

Параметр Save format у блоках *Scope*, *To Workspace*, *Out* визначає формат запам'ятовування даних.

Інформація може бути збережена в наступних форматах: *Structure with time* (*Структура з часом моделювання*), *Structure* (*Структура без часу моделювання*) і *Array* (*Масив*).

Найпростішим типом даних є ***Array***. У цьому випадку кожен сигнал записується в окремий стовбець матриці. Один рядок матриці містить стан усіх сигналів у конкретний момент часу. Кількість рядків при d=1 дорівнює числу кроків моделювання. Якщо дані збережені в матриці y, то виділити з неї j-ий сигнал можна операцією y(:,j), а всі сигнали в i-й зареєстрований момент часу – операцією y(i,:).

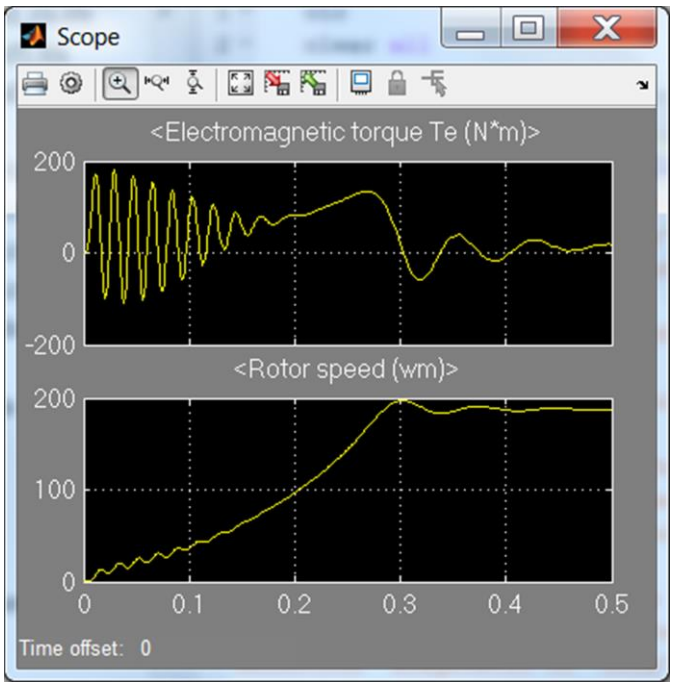


Рис. 5.10. Вікно блока *Scope*

Блок ***Scope* (*Осцилограф*)** у процесі моделювання відображає вихідні сигнали приєднаних до нього блоків. Для того, щоб побачити графіки перехідних процесів, необхідно відкрити його вікно (див. рис. 5.10), що поряд з полем виведення графіків містить такі важливі «кнопки» (рис. 5.11):

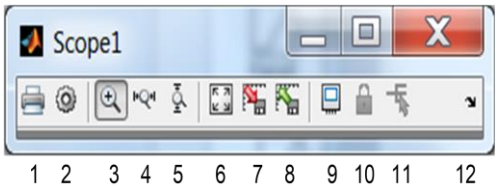


Рис. 5.11. Кнопки вікна блоку *Scope*

1 – *Print* (друкування);

2 – *Parameters* (настроювання параметрів);

3 – *Zoom* (рівномірна зміна масштабу по обох осях);

4 – *Zoom X-axis* (зміна масштабу по осі *X*);

5 – *Zoom Y-axis* (зміна масштабу по осі *Y*);

6 – *Autoscale* (автоматичне масштабування);

7 – *Save current axes settings* (запам'ятовування системи координат);

8 – *Restore saved axes settings (*відновлення системи координат, наприклад, після зміни масштабу однією з клавіш *Zoom*);

12 – *Dock Scope* (осцилограф виводить графіки без кнопкової панелі).

Звичайно після закінчення процесу моделювання для того, щоб побачити графіки в нормальному масштабі, натискають кнопку *Autoscale*, а потім запам'ятовують діапазони систем координат кнопкою *Save current axes settings*.

Якщо результати автоматичного масштабування за якимись причинами не влаштовують користувача, то він може змінити межі координатних осей графіків вручну за допомогою кнопок 3-5 візуально або установкою границь системи координат у чисельному вигляді.

Масштабування зображення в обраному виміру кнопками можна виконувати двома способами. При першому способі після активізації відповідної кнопки курсор миші підводять до бажаної ділянки графіка і клацають на ньому лівою клавішею. При кожнім натисканні масштаб буде збільшуватися, що приведе до відображення у вікні всі меншого і меншого фрагмента графіка. При другому способі фрагмент графіка, що хочуть збільшити, виділяють за допомогою курсору. Повернення до результатів попереднього масштабування виконується командою контекстуального меню *Zoom out*, а повернення до вихідного масштабу – кнопками *Autoscale* або *Restore saved axes settings*.

Для установки меж графіка по осі ординат у чисельному вигляді необхідно викликати контекстуальне меню щигликом правої кнопки миші в площині системи координат, вибрати з нього функцію *Axes Properties* і у вікні, що відкрилося, установити *параметри* Y-min *і* Y-max. У цьому ж вікні можна установити заголовок системи координат (параметр *Title*), замінивши коментар *%<SignalLabel>* бажаним текстом або вставивши цей текст перед коментарем.

Діапазон часу, що є спільним для всіх систем координат, можна змінити у ***вікні Parameters***, що відкривається однойменною кнопкою й утримує 2 вкладки: *General* і *Data history.*

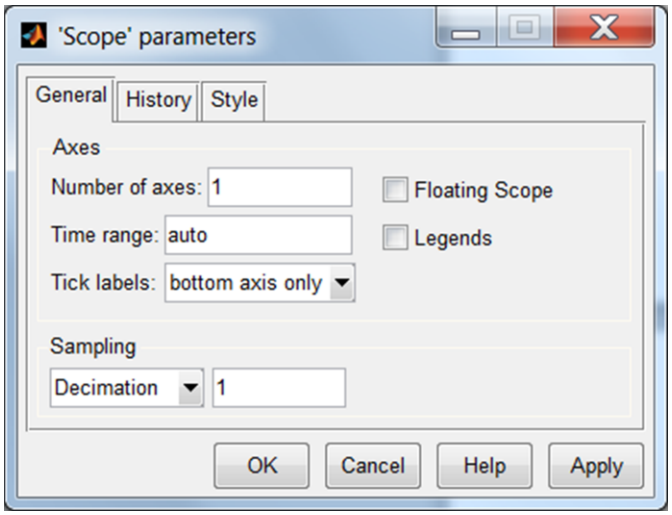


Рис. 5.12. Вкладка *General* вікна параметрів блоку *Scope*

За допомогою ***вкладки Style*** (рис. 5.14) назначаються такі параметри:

- *Figure Colour* – колір графічної фігури;
- *Axes Colour* – колір системи координат (колір фону і колір ліній градуювання);
- *Line* – стиль зображення ліній, їх товщина та колір;

За допомогою ***вкладки General*** (див. рис. 5.12) установлюються такі параметри:

- *Number of axes* – кількість систем координат, що визначає і кількість вхідних портів;
- *Time range* – діапазон часу;

За допомогою ***вкладки History*** (див. рис. 5.13) назначаються параметри *Limit data point to last.*

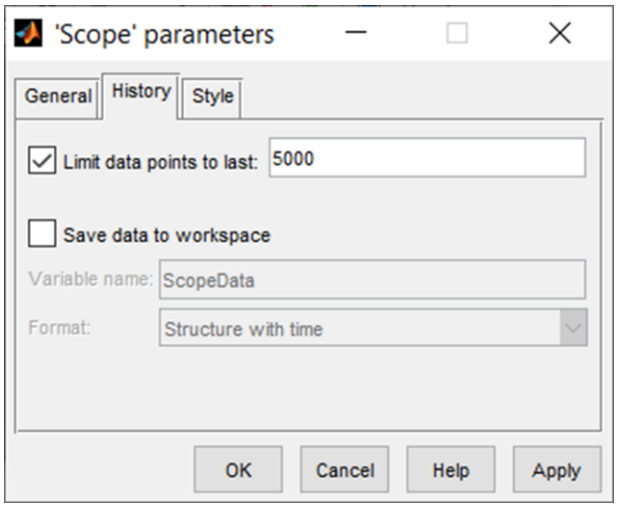


Рис. 5.13. Вкладка *History* вікна параметрів блоку *Scope*

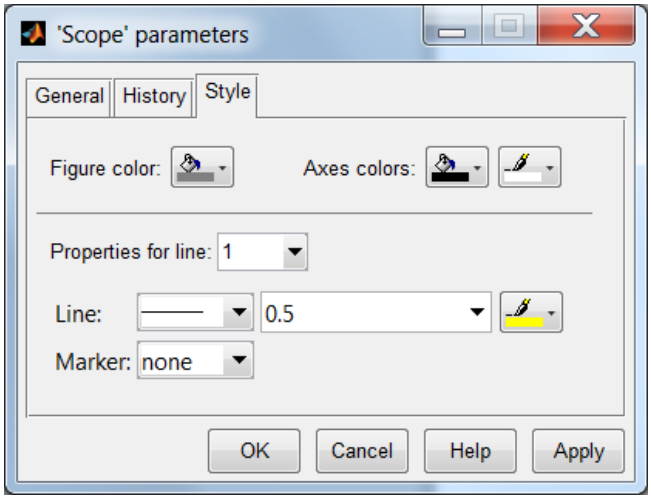


Рис. 5.14. Вкладка *Style* вікна параметрів блоку *Scope*

- *Marker* – стиль зображення маркерів точок графіка.

Останні три параметри встановлюються окремо для кожного графіка, вибір якого здійснюється параметром *Parameters for line.*

Розмір і пропорції вікна *Scope* можна змінювати довільно.

Скалярний сигнал зображується завжди жовтим кольором, а для складових векторного сигналу використовуються за замовчанням повторювані циклічно 6 кольорів: *жовтий, малиновий, блакитний, червоний, зелений, синій*. Фон зображення за замовчанням *чорний*.

Приклади використання блоку *Scope* для візуалізації двох синусоїдальних сигналів різної амплітуди і частоти показано на рис. 5.15.

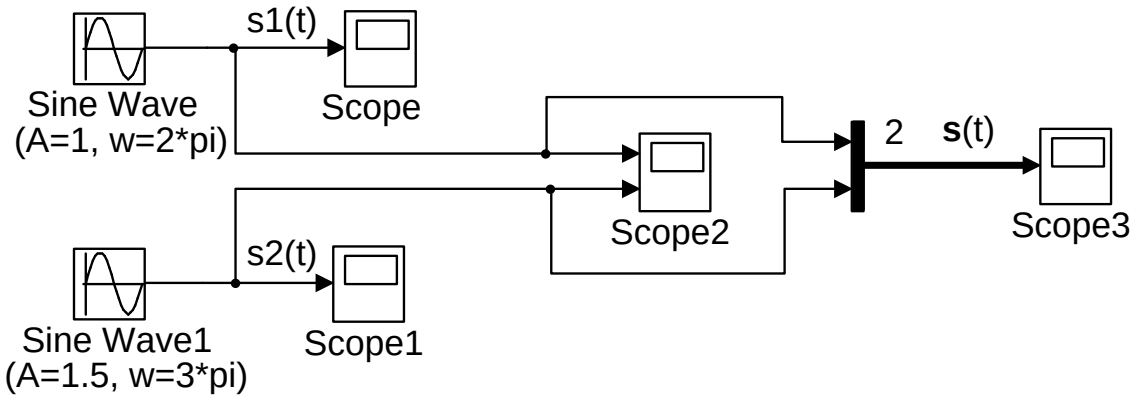


Рис. 5.15. Модель візуалізації двох сигналів блоками *Scope*

Вигляд відкритих після симуляції осцилографів показано на рис. 5.16.

Блоки *Scope* та *Scope1* відображують синусоїди у різних вікнах, а блоки *Scope1* та *Scope3* – в одному вікні. Різниця між останніми полягає в тому, що *Scope2* відкриває 2 системи координат, кожна у своєму підвікні, що досягається установленням у полі параметру *Number of axes* вкладки *General* вікна

Parameters значення 2, а *Scope3* зображує обидва сигнали, попередньо об‘єднані блоком *Mux* (Мультиплексор) в один векторний сигнал, в одній системі координат.

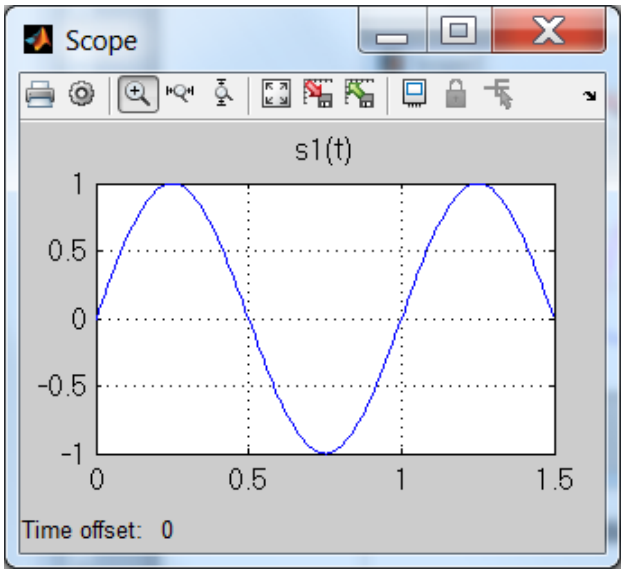
Блок ***Display* (*Дисплей*)** застосовується для виведення чисельних значень сигналів у заданому форматі, що обирається за допомогою параметра *Format* і може приймати наступні значення:

short – 4 значущі цифри у форматі з фіксованою крапкою;

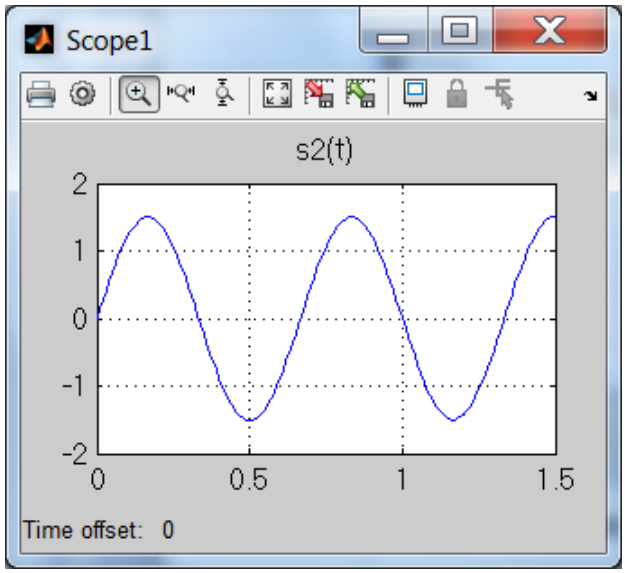
long – 14 значущих цифр у форматі з фіксованою крапкою;

short-e – 5 значущих цифр у мантисі чисел із плаваючою крапкою;

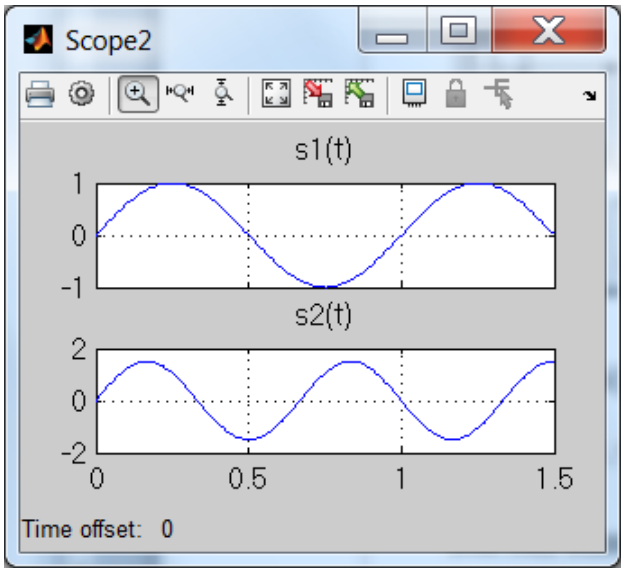
long-e – 16 значущих цифр у мантисі чисел із плаваючою крапкою.

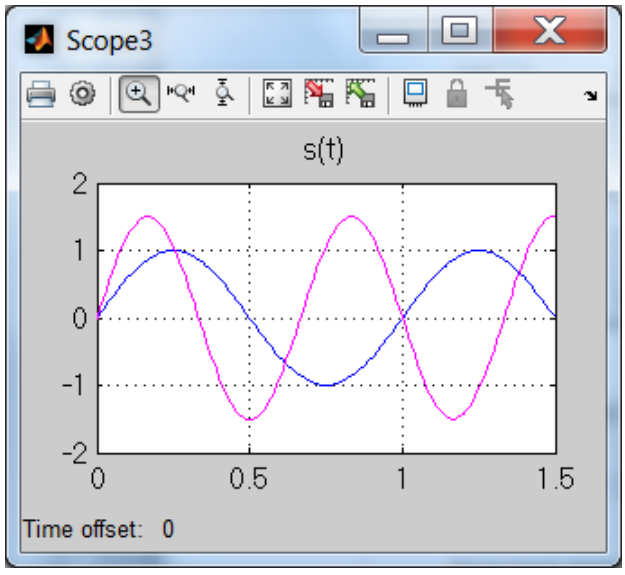


а)



б)





в) *г)*

Рис. 5.16. Результати візуалізації синусоїд блоками *Scope* (а), *Scope1* (б), *Scope2* (в), *Scope3* (г)

Скалярні і векторні сигнали, що надходять до блоку, можуть мати як дійсні, так і комплексні значення.

Якщо розмір блоку *Display* малий для відображення всієї інформації, то в його правому нижньому куті з'являється чорний трикутник, що вказує, у якому напрямку варто розтягти піктограму блоку.

Приклад використання блоку наведено на рис. 5.17.

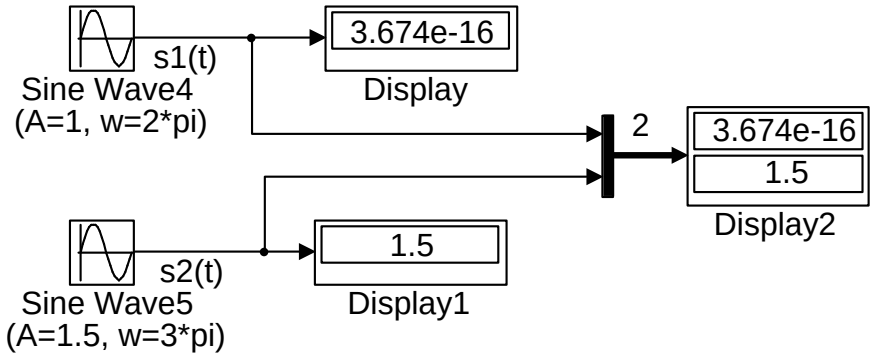


Рис. 5.17. Приклад використання блоку *Display*

Оскільки зазвичай моделювання здійснюється дуже швидко і з малими проміжками часу між розрахунками, то користувач може побачити тільки останнє значення змінної після закінчення симуляції.

5.5. Блоки запам‘ятовування сигналів

Блоками ***Out* (*Вихідний Порт*)** відзначають виходи моделей і підсистем.

У підсистемах вони необхідні для зв'язку підсистеми з моделлю більш високого рівня, а в моделях вищого рівня – для запам‘ятовування приєднаних до них сигналів з метою подальшого використання для побудови графіків або для аналізу чисельних даних.

Для виконання останньої операції необхідно відкрити через функцію *Simulation* вікно *Model Configuration Parameters*, вибрати в ньому вкладку *Data Import/Export* та в розділі *Save to workspace* встановити прапорці в полях *Output*

та *Time* і визначити імена змінних, у яких зберігатиметься інформація. За замовчанням у полі *Output* прописано ім‘я yout , а у полі *Time* – ім‘я tout, які, за бажанням користувача, можна змінити на будь-які інші (рис. 5.8).

Цей блок зручно використовувати для фіксації векторних сигналів.

Пізніше буде показано, що блоки *Out* використовують також для інформування додатків, призначених для аналізу і синтезу систем автоматичного керування (наприклад, *Control Toolbox*) про можливі точки знімання вихідних сигналів.

Скалярні і векторні сигнали, що надходять на вихідні порти можуть мати як дійсні, так і комплексні значення.

Параметри *Output when disabled* і *Initial output* мають значення тільки в підсистемах, виконуваних за умовою.

Для того, щоб отримати графіки, аналогічні графікам, побудованим за допомогою блоків *Scope* та *XY Graph*, використовуючи інформацію, записану у змінні tout, yout, з застосуванням блоку *Out*, треба спочатку встановити значення параметру *Format* в *Array*, а потім виконати таку послідовність операторів алгоритмічної мови *MATLAB*:

```
figure (1), plot (tout, yout (:,1)), title (‘s1(t)’), grid on
figure (2), plot (tout, yout (:,2)), title (‘s2(t)’), grid on
figure (3), subplot (2,1,1), plot (tout, yout (:,1)), title (‘s1(t)’), grid on
subplot (2,1,2), plot (tout, yout (:,2)), title (‘s2(t)’), grid on
figure (4), plot (tout, yout), legend (‘s1(t)’, ‘s2(t)’), grid on
figure (5), plot (yout (:,1), yout (:,2)), title (‘s2(s1)’), grid on
```

Блок ***To Workspace* (*У Пам'ять*)** запам'ятовує дані, що надходять на його вхідний порт у перемінній з ім'ям, завданим у поле параметра *Variable name*. Його зручно застосовувати для фіксації змінних з різними іменами.

Для запису інформації у форматі *Array* (*Масив*) треба не забути змінити значення за замовченням параметру *Save Format* (див. рис. 5.18).

За замовченням параметр *Variable Name* має значення simout.

Приклад реєстрації сигналів блоками та *Out* та *To Workspace* наведено на рис. 5.19.

Результати не зміняться, якщо у наведеній вище програмі замінимо yout (:,1) на s1, yout (:,2) на s2 та yout на [s1 s2].

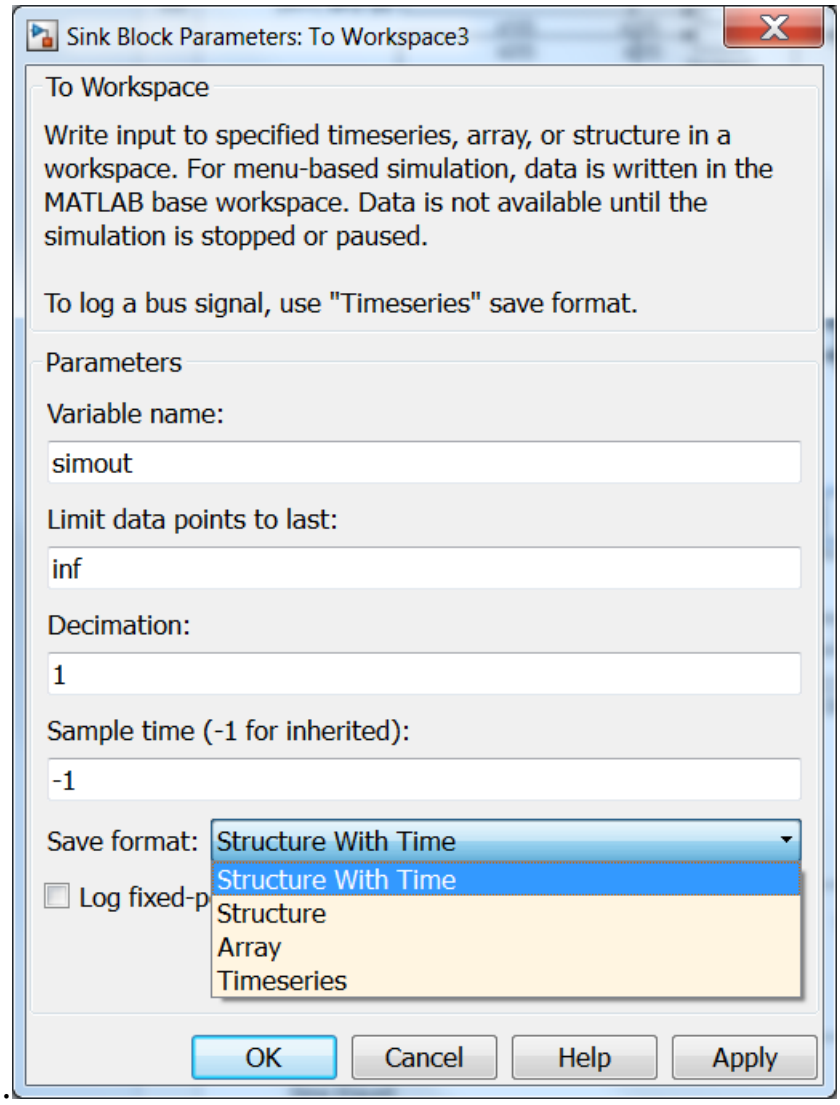


Рис. 5.18. Вікно параметрів блоку *To Workspace*

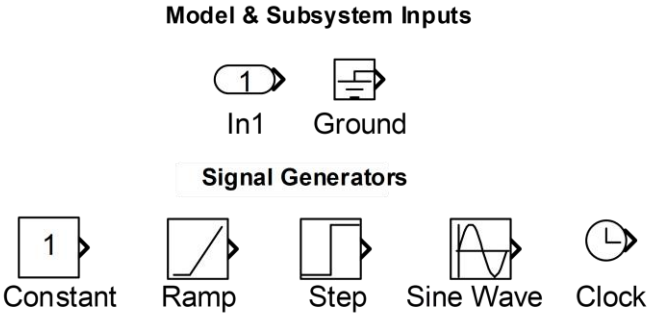


Рис. 5.20. Основні джерела вхідних сигналів бібліотеки *Sources*

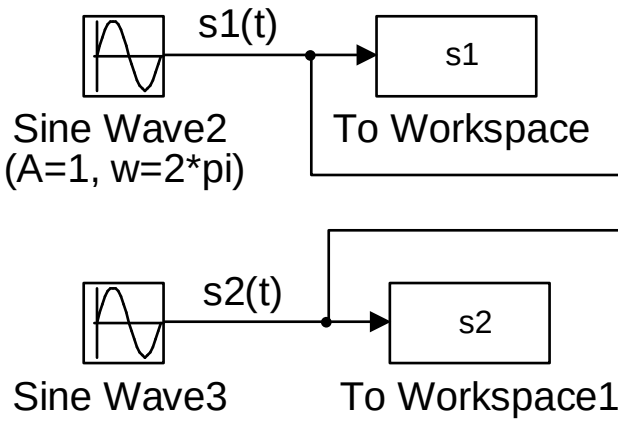


Рис. 5.19. Приклад моделі для запам‘ятовування сигналів блоками *Out* та *To Workspace*

5.6. Формування найпростіших вхідних сигналів блоками бібліотеки *Sources*

Основу формування вхідних сигналів складають блоки бібліотеки джерел *Sources*. Піктограми деяких з них подані на рис. 5.20. До цієї бібліотеки входить група блоків, що не мають вхідних портів, а мають тільки виходи.

Виходом блоку ***Constant* (*Константа*)** є скалярне постійне значення c, визначене параметром *Constant value* і не залежне від часу $y = c = const$ або вектор констант у вигляді рядка $\mathbf{y} = [c_1 \quad c_2 \quad \ldots \quad c_n]$ або стовпця $\mathbf{y} = [c_1 \quad c_2 \quad \ldots \quad c_n]^{\mathrm{T}}$.

Джерело ***Step* (*Стрибок, Сходинка*)** забезпечує стрибкоподібну зміну вихідного сигналу між двома постійними рівнями y_{In} *(Initial value)* і y_{Fin} *(Final value)* у заданий момент часу t_s *(Step time)*:

$$y_{Step}(t) = \begin{cases} y_{In} \text{ при } t \le t_s, \\ y_{Fin} \text{ при } t > t_s. \end{cases} \qquad (5.1)$$

Загальний вигляд скалярного ступінчатого сигналу показано на рис. 5.21.

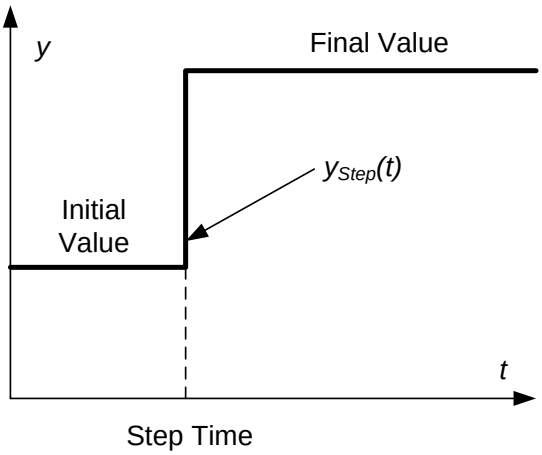


Рис. 5.21. Загальний вигляд скалярного ступінчатого сигналу

Блок ***Clock*** **(*Годинник*)** забезпечує відлік і відображення часу моделювання: $y(t)=t$.

Джерело ***Ramp*** **(*Рампа, Похила Площина*)** формує сигнал, що лінійно наростає або спадає від значення y_0 (*Initial output*), починаючи з моменту часу t (*Start time*) з коефіцієнтом пропорційності k (*Slope*):

$$y(t)=\begin{cases} y_0 \ \text{ при } \ t \le t_S, \\ y_0 + kt \ \text{ при } \ t > t_S. \end{cases} \tag{5.2}$$

Загальний вигляд такого сигналу показано на рис. 5.22.

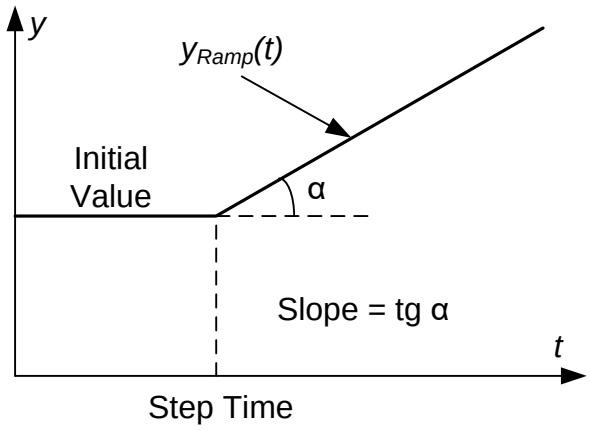


Рис. 5.22. Загальний вигляд скалярного сигналу *Ramp*

Блок ***Sine Wave*** **(*Синусоїда*)** генерує синусоїду у функції часу відповідно до рівняння:

$$y(t) = A\sin(\omega t + \varphi_0) + y_0, \qquad (5.5)$$

де

A – амплітуда (*Amplitude*);

ω – кругова частота, рад/с (*Frequency*);

φ_0 – фазовий зсув, рад. (*Phase*);

y_0 – вертикальний зсув (*Bias*).

Блок ***In* (*Вхідний Порт*)** помічає входи моделі. Основним параметром блоку *In* є номер порту (*Port number*), що відображається в його піктограмі. Усі порти в межах одного вікна повинні мати послідовну нумерацію. Номера портів формуються автоматично в порядку їхньої появи у вікні. При видаленні порту з вікна порти, що залишилися в ньому, перенумеровуються таким чином, щоб у послідовності номерів не було пропусків. Номера портів можна змінювати і вручну.

5.7. Моделювання неперервних лінійних динамічних систем у середовищі *Simulink*

У більшості випадків на першому етапі досліджень роблять такі припущення, які дозволяють вважати САУ лінійними та стаціонарними. Такі системи в англомовній науково-технічній літературі називають *LTI-systems* – *Linear TimeInvariant systems*. Методи аналізу та синтезу таких системи є найпростішими та добре розвинутими.

Simulink є середовищем для структурного математичного моделювання. Це означає, що математичний опис досліджуваних об‘єктів має бути поданим у вигляді структурних схем, в яких динамічні властивості систем відображаються за допомогою передавальних функцій, отриманих із диференційних рівнянь.

Структурні схеми уявляють собою графічне відображення математичного опису у вигляді з‘єднаних між собою односпрямованими лініями зв‘язку окремих блоків. У структурних схемах диференційні рівняння (ДР) відображуються у вигляді передавальних функцій ПФ), а алгебраїчні – у вигляді пропорційних ланок та алгебраїчних суматорів

Щоб зрозуміти, як ДР перетворюються на ПФ, розглянемо лінійну динамічну неперервну (***continuous***) систему з одним входом та одним виходом (***SISO – Single Input Single Output***), яка описується звичайним диференційним рівнянням (ДР) *n*-го порядку з постійними коефіцієнтами та n початковими умовами:

$$\begin{aligned} &A_n \frac{d^n y(t)}{dt^n} + A_{n-1} \frac{d^{n-1} y(t)}{dt^{n-1}} + \ldots + A_1 \frac{dy(t)}{dt} + A_0 y(t) = \\ &= B_m \frac{d^m u(t)}{dt^m} + B_{m-1} \frac{d^{m-1} u(t)}{dt^{m-1}} + \ldots + B_1 \frac{du(t)}{dt} + B_0 u(t), \end{aligned} \tag{5.5}$$

$$m \le n, \tag{5.6}$$

$$y(0) = y_0, \quad y'(0) = y'_0, \quad y''(0) = y''_0, \quad \ldots, \quad y^{(n-1)}(0) = y^{(n-1)}{}_0,$$

де

t – незалежна змінна, якою в даному випадку є час;

$\frac{d}{dt}$ – оператор диференціювання;

$u(t)$ – відома функція часу, тобто відомий вхідний сигнал (керуюча дія або збурення);

$y(t)$ – невідома функція часу, тобто вихідний сигнал;

n – порядок системи (порядок вищої похідної вихідного сигналу);

m – порядок вищої похідної вхідного сигналу;

$\mathbf{A} = \begin{bmatrix} A_n & A_{n-1} & \ldots & A_1 & A_0 \end{bmatrix}$ – вектор коефіцієнтів при похідних вихідного сигналу;

$\mathbf{B} = \begin{bmatrix} B_m & B_{m-1} & \ldots & B_1 & B_0 \end{bmatrix}$ – вектор коефіцієнтів при похідних вхідного сигналу.

Щоб отримати передавальну функцію (ПФ) такої системи, напишемо ДР (5.5) в операторній формі. Для цього замінюємо оператор диференціювання *d/dt* оператором Лапласа s, а сигнали у просторі часу $y(t)$, $u(t)$ – зображеннями цих сигналів у просторі оператора Лапласа $y(s)$, $u(s)$ (замість s для оператора Лапласа часто використовують позначення p):

$$\frac{d}{dt} \to s, \qquad y(t) \to y(s), \qquad u(t) \to u(s). \tag{5.7}$$

В результаті такого перетворення отримуємо:

$$\begin{aligned} &A_n s^n y(s) + A_{n-1} s^{n-1} y(s) + ... + A_1 s y(s) + A_0 y(s) = \\ &= B_m s^m u(s) + B_{m-1} s^{m-1} u(s) + ... + B_1 s u(s) + B_0 u(s). \end{aligned} \tag{5.8}$$

Як бачимо, похідні за часом від вхідного та вихідного сигналів у рівнянні (5.5) перетворились на добутки оператора Лапласа у відповідній степені на зображення цих сигналів у рівнянні (5.8). Тепер у лівій частині рівняння (5.8) можна винести за дужки $y(s)$, а у правій – $u(s)$:

$$y(s)(A_n s^n + A_{n-1} s^{n-1} + ... + A_1 s + A_0) = u(s)(B_m s^m + B_{m-1} s^{m-1} + ... + B_1 s + B_0). \tag{5.9}$$

З операторного рівняння знаходимо ПФ системи як відношення зображення вихідного сигналу до зображення вхідного сигналу:

$$W(s) = \frac{y(s)}{u(s)} = \frac{B_m s^m + B_{m-1} s^{m-1} + ... + B_1 s + B_0}{A_n s^n + A_{n-1} s^{n-1} + ... + A_1 s + A_0} = \frac{H_m(s)}{G_n(s)}. \tag{5.10}$$

Чисельник $H_m(s)$ і знаменник $G_n(s)$ ПФ (5.10) уявляють собою степеневі поліноми. Поліном у знаменнику називають характеристичним (англ. ***denominator***, скорочено ***den***), а поліном у чисельнику – поліномом впливу (англ. ***numerator***, скорочено ***num***).

Порядок динамічної ланки визначається порядком характеристичного поліному, що збігається з порядком відповідного диференційного рівняння.

Рівняння

$$G_n(p) = A_n p^n + A_{n-1} p^{n-1} \cdot + ... + A_1 p + A_0 = 0 \tag{5.11}$$

називають ***характеристичним рівнянням (Characteristic equation, Eigenvalue equation)***, а його корені – ***полюсами системи (Poles)***:

$$\mathbf{P} = \left[p_1,\ p_2, ...,\ p_n \right]^T. \tag{5.12}$$

Корні рівняння

$$H_m(p) = B_m p^m \cdot B_{m-1} p^{m-1} + ... + B_1 p + B_0 = 0 \tag{5.13}$$

називають ***нулями системи (Zeros)***:

$$\mathbf{Z}=\left[z_1,\, z_2,\, ...,\, z_m\right]^T. \qquad (5.14)$$

Поліноміальні ПФ часто нормують за вільними членами поліномів (або при їх відсутності за коефіцієнтами при молодших степенях оператору Лапласа):

$$W(p)=\frac{y(p)}{u(p)}=k\,\frac{b_m p^m\cdot b_{m-1}p^{m-1}+...+b_1 p+1}{a_n p^n+a_{n-1}p^{n-1}+...+a_1 p+1}. \qquad (5.15)$$

де

$$a_i=\frac{A_i}{A_0},\quad b_i=\frac{B_i}{B_0},\quad k=\frac{B_0}{A_0}. \qquad (5.16)$$

Ланки з різними передавальними функціями мають спеціальні назви, наприклад,

- інтегральна $W_{\text{i}}(s)=\frac{1}{s}$;
- аперіодична $W_{\text{a}}(s)=\frac{k}{Ts+1}$;
- коливальна $W_{\text{к}}(s)=\frac{k}{T^2s^2+2\xi Ts+1}$.

Ще одна форма запису поліноміальної ПФ отримується шляхом нормування її поліномів за коефіцієнтами при старших степенях оператору Лапласа

$$W(s)=\frac{y(s)}{u(s)}=K\,\frac{s^m+\beta_{m-1}s^{m-1}+...+\beta_1 s+\beta_0}{s^n+\alpha_{n-1}s^{n-1}\cdot+...+\alpha_1 s+\alpha_0}, \qquad (5.17)$$

де

$$\alpha_i=\frac{A_i}{A_0}=\frac{a_i}{a_n},\quad \beta_i=\frac{B_i}{B_m}=\frac{b_i}{b_m},\quad K=\frac{B_m}{A_n}=k\,\frac{b_m}{a_n}. \qquad (5.18)$$

З (5.17) отримують ПФ у формі розкладання на нулі-полюси (***Zero-Pole***), користуючись властивістю тотожної рівності поліномів

$$s^n+\alpha_{n-1}s^{n-1}\cdot+...+\alpha_1 s+\alpha_0=(s-s_1)(s-s_2)...(s-s_n), \qquad (5.19)$$

$$s^m+\beta_{m-1}s^{m-1}+...+\beta_1 s+\beta_0=(s-z_1)(s-z_2)...(s-z_n). \qquad (5.20)$$

Після підстановки (5.19) та (5.20) у ПФ (5.17) отримуємо

$$W(p)=\frac{y(s)}{u(s)}=K\frac{(s-z_1)(s-z_2)...(s-z_n)}{(s-p_1)(s-p_2)...(s-p_n)}. \quad (5.21)$$

У *Simulink* лінійні неперервні динамічні ланки моделюють блоки бібліотеки *Continuous*. (див. рис. 5.23).

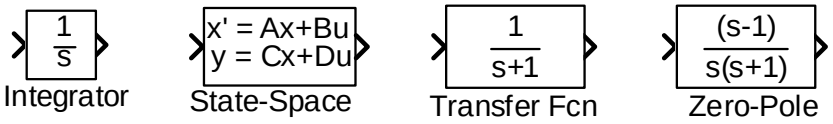


Рис. 5.23. Основні блоки з бібліотеки неперервних динамічних блоків (*Continuous*)

Блок ***Integrator (Інтегратор)*** з одиничним підсиленням є найпростішою динамічною ланкою першого порядку з ПФ

$$W_{int}(s)=\frac{x(s)}{u(s)}=\frac{1}{s} \quad (5.22)$$

яка здійснює операцію інтегрування з початковою умовою:

$$x(t)=\int_0^t u(t)dt+x_0 \quad (5.23)$$

При виборі режиму *internal* для опції *Initial condition source* початкові умови призначаються в полі параметра *Initial condition*. Інші параметри, додаткові можливості та режими роботи цього блоку будуть розглянуті в дисципліні «Моделювання систем автоматичного керування».

Блок ***Transfer Fcn*** імітує об‘єкт з математичним описом у вигляді передавальної функції в поліноміальній формі (5.10).

Вікно визначення параметрів блоку *Transfer Fcn* показано на рис. 5.24.

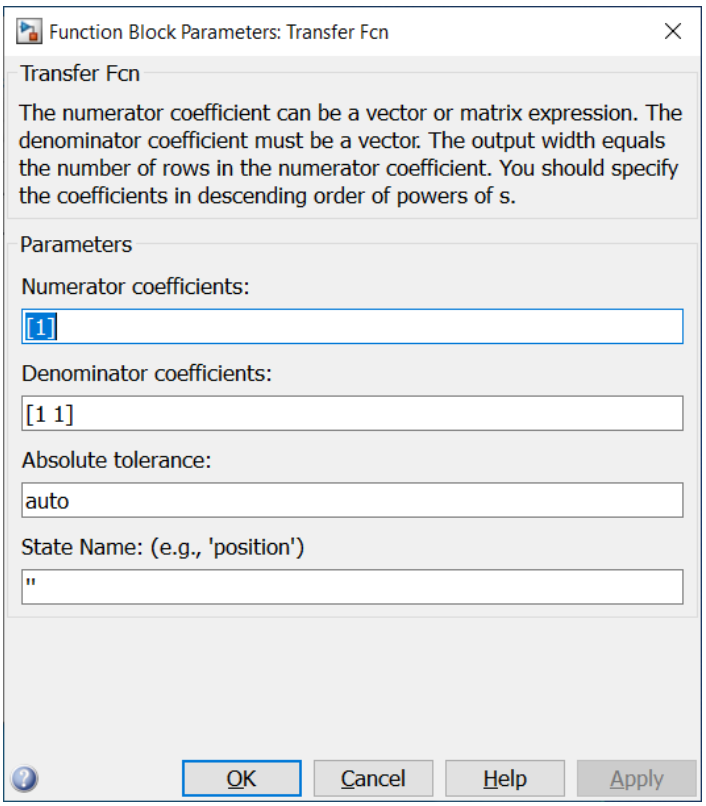


Рис. 5.24. Вікно визначення параметрів блоку *Transfer Fcn*

Основними вхідними параметрами, що визначають вигляд відтворюваної ПФ, є вектори коефіцієнтів степеневих поліномів чисельника (*Numerator coefficients*) і знаменника (*Denominator coefficients*), упорядковані за спаданням степенів оператора Лапласа *s*. Порядок знаменника повинний бути більшим від порядку чисельника або дорівнювати йому. Блок має нульові початкові умови.

Коефіцієнти можуть бути введені трьома способами:

1) константами і скалярними змінними, об'єднаними квадратними дужками у вектор;

2) змінною без дужок;

3) змінною або виразом у круглих дужках.

При першому способі в піктограмі блоку відображаються поліноми з коефіцієнтами у вигляді введених констант і змінних при відповідних степенях оператора Лапласа *s*. Наприклад, введення вектору [1 -2 *T* 4] відобразиться в блоці як

$$s^3 - 2s^2 + Ts + 4 \ ,$$

а уведення вектора [*b*0, *b*1, *b*2] – як

$$b0s^2 + b1s + b2.$$

Змінні, що використані в цих векторах, повинні одержати значення в робочому середовищі *MATLAB* до початку процесу моделювання.

При другому способі поліном відображається у вигляді *змінна(s)*. Наприклад, введення *num* відображається як *num(s)*, введення *An* – як *An(s)* та т.і.

При третьому способі *Similink* шукає значення змінних, які входять до виразу, у робочому просторі *MATLAB,* обчислює значення виразу і відображає його так само, як і при першому способі введення. Наприклад, якщо поліном визначено у діалоговому вікні як (*G*+*B*), а перемінні *G* і *B* визначено в *MATLAB* як *G*=[4 1 1], *B*=[0 1 0], то відображення полінома в блоці має вид: $4s^2+2s+1$.

Значення змінних повинне існувати в робочому середовищі в будь-який момент зображення блоку, інакше в піктограмі блоку відобразяться три знаки питання: *???*.

Блок ***Zero-Pole*** (***Нулі-Полюси***) імітує ПФ (5.17) у форматі розкладення на нулі-полюси з такими параметрами:

K – посилення (*Gain*);

Z – вектор нулів (*Zeros*);

P – вектор полюсів (*Poles*).

***При відсутності нулів їх задають пустою матрицею* [].**

Нулі і полюси можуть мати не тільки дійсні, але і комплексно-спряжені значення. Вони можуть вводитися тими ж трьома способами, що і вектори степеневих багаточленів блока *Transfer Fcn*.

При наявності комплексно спряжених пар нулів або полюсів в піктограмі блока співмножники передавальної функції виводяться не у вигляді добутку двох поліномів першого порядку з комплексними вільними членами, а у вигляді одного поліному другого порядку з дійсними коефіцієнтами. Наприклад, якщо

$$p_{1,2} = [\alpha + j\beta,\ \alpha - j\beta], \qquad (5.24)$$

то відповідний множник характеристичного поліному знаходиться як

$$G_{1,2}(s) = (s + \alpha - j\beta)(s + \alpha + j\beta) = (s + \alpha)^2 + \beta^2 = s^2 + 2\alpha s + (\alpha^2 + \beta^2). \qquad (5.25)$$

Для зв‘язку динамічних елементів між собою у складі лінійних динамічних систем застосовуються пропорційні ланки та алгебраїчні суматори. В вони реалізуються лінійними арифметичними блоками бібліотеки *Math Operations* (рис. 5.25).

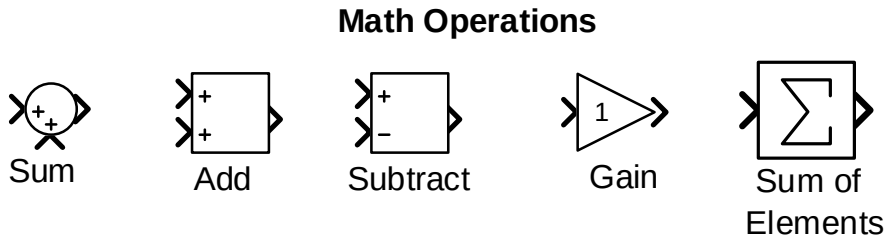


Рис. 5.25. Лінійні математичні блоки бібліотеки *Math Operations*

Блоки ***Sum (Суматор), Add (Додавати)*** та ***Subtract (Віднімати)*** є взаємозамінними і можуть легко перетворюватись один у інший зміною параметрів (див. рис. 5.26) ***Icon Shape (Форма Іконки)*** та ***List of signs (Послідовність знаків операцій)***.

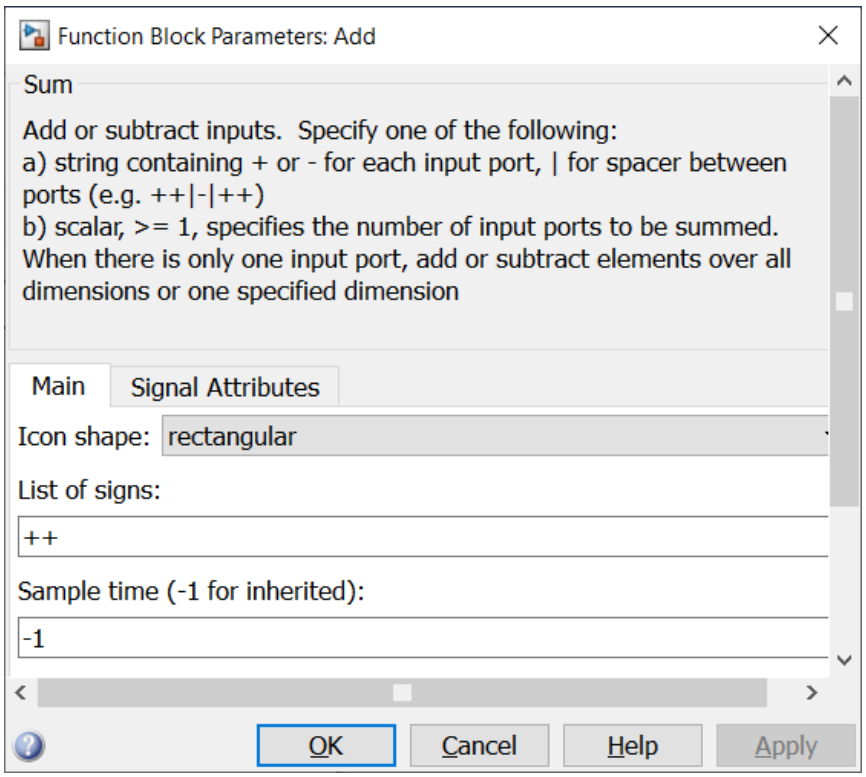


Рис. 5.26. Вікно визначення параметрів блоку *Sum*

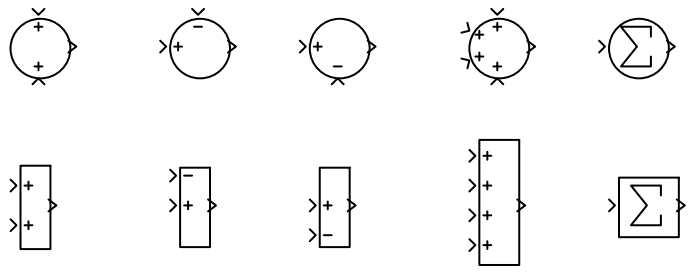
Кожний з перелічених вище блоків може мати круглу (*round*) або прямокутну (*rectangular*) форму. Усі вони підсумовують вхідні сигнали з відповідними знаками, які встановлюються в полі параметра *List of signs* (входи нумеруються зверху вниз для прямокутного блоку та проти годинної стрілки для круглого блоку). Комбінація знаків “+”, ”–“ описує параметри кожного конкретного порту, де кількість портів відповідає кількості знаків, використаних у комбінації. Для пропуску позиції чергового порту у списку знаків використовують символ „|”.

Якщо замість списку знаків визначити кількість входів, то усі вони будуть підсумовуючими. При одному вході блок підсумовує з відповідним знаком елементи вхідного вектору:

$$y = sum(u) = \sum_{i=1}^{k} u_i \,. \tag{5.26}$$

При цьому в піктограмі блоку відображається символ Σ.

Можливі різновиди досліджуваних блоків в залежності від значень їх параметрів наведені на рис. 5.27.



а) *б)* *в)* *г)* *д)*

Рис. 5.27. Блоки *Sum* та *Add* при таких значеннях параметру *List of signs*:
а) ++; *б)* –+|; *в)* |+–; *г)* 4; *д)* 1 або +

Блок ***Gain* (*Підсилення*)** зі скалярним значенням основного параметру і скалярним входом є пропорційною ланкою, яка обчислює вираз

$$y = k \cdot u \,. \tag{5.27}$$

Піктограма блоку відображає скалярний коефіцієнт підсилення в тій же формі, у якій він визначений при введенні (змінна або константа). Якщо коефіцієнт заданий у вигляді змінної в круглих дужках, то усередині блоку відображається її значення. Якщо значення змінної або її ім'я, або вираз занадто великі для того, щоб відобразити їх у межах блоку, то піктограма відображає символи "-*K*-". Щоб побачити вираз або значення коефіцієнта підсилення, необхідно збільшити розміри блоку.

5.8. Контрольні питання та завдання

1. Як можна запустити *Simulink*?
2. Які *Simulink-бібліотек* ви знаєте?
3. Яке призначення клавіш у нижній частині вікон параметрів блоків *Simulink*?
4. Як подивитись демонстрації *Simulink*?
5. Яке розширення мають файли моделей *Simulink*?
6. Як відкрити нове вікно, в якому можна досліджувати блоки *Simulink* та складати з них *Simulink*-моделі?
7. Як зберегти модель після редагування? Як впевнитися у тому, що результат останнього редагування збережено?
8. Як запустити процес симуляції моделі?
9. Який час моделювання встановлено за замовчанням? Як його змінити?
10. Яке призначення вкладки *Callbacks* вікна *Model Properties*?
11. Як скопіювати модель в буфер?
12. Як можна змінити орієнтацію блоків моделі та розташування їх імен?
13. Як змінити розмір текстів у моделі?
14. Які методи розв'язання диференціальних рівнянь пропонує *Simulink*? Який метод пропонується застосовувати початківцям при моделюванні лінійних неперервних систем?
15. Як можна підвищити щільність розрахунків для покращення якості візуалізації перехідних процесів?

16. Як надати імена лініям зв‘язку в моделях?
17. Як помітити фрагменти моделі перед виконання над ними певних дій? Які дії можна виконати з поміченими фрагментами моделі?
18. Як розгалузити лінію зв‘язку?
19. Яку форму може приймати графічний курсор у вікні моделі? Про готовність до яких дій свідчить його форма?
20. Які блоки *Simulink* використовують для фіксації та візуалізації результатів моделювання? В якій бібліотеці вони знаходяться?
21. Які блоки *Simulink* використовують для формування найпростіших вхідних сигналів? В якій бібліотеці вони знаходяться?
22. Як отримують з диференціального рівняння передавальну функцію?
23. Як називають в *Simulink* поліноми в чисельнику та знаменнику ПФ? Як позначають оператор Лапласа?
24. Як виглядає ПФ у форматі розкладення поліномів на множники (розкладення на нулі-полюси). Як її отримують?
25. Які лінійні динамічні *Simulink*-блоки застосовують при моделюванні неперервних систем автоматичного керування? В якій бібліотеці вони знаходяться?
26. Які *Simulink*-блоки здійснюють операції підсилення та алгебраїчного підсумовування? В якій бібліотеці вони знаходяться?

6. ФУНКЦІЇ РОЗДІЛУ *Control System Toolbox* ПАКЕТУ *MATLAB*

Розділ *Control System Toolbox* програмного пакету *MATLAB* призначений для створення, перетворення, аналізу та синтезу лінійних стаціонарних систем (*LTI-systems*). Основні функції цього розділу наведені в табл. 6.1.

При виконанні команди help control отримаємо перелік інструментів цього розділу. Наведемо деякі з них:

```
Graphical User Interfaces.
   ltiview       - LTI Viewer (time and frequency response analysis).
   sisotool      - SISO Design Tool (interactive compensator tuning).
Linear models and Model conversion.
   tf            - Create transfer function (TF) models and Conversion to transfer function.
   zpk           - Create zero/pole/gain (ZPK) models and Conversion to zero/pole/gain.
   ss            - Create state-space (SS) models and Conversion to state-space.
 Data extraction.
   tfdata        - Extract numerators and denominators.
   zpkdata       - Extract zero/pole/gain data.
   ssdata        - Extract state-space matrices.
 System interconnection.
   parallel      - Connect models in parallel (see also overloaded +).
   series        - Connect models in series (see also overloaded *).
   feedback      - Connect models with a feedback loop.
 System dynamics.
   dcgain        - Steady-state (D.C.) gain.
   pole          - System poles.
   zero          - Zeros and gain of SISO system.
   tzero         - Invariant zeros of MIMO system.
   order         - System order (number of states).
   pzmap         - Pole-zero map.
   damp          - Natural frequency and damping of system poles.
 Time-domain analysis.
   step          - Step response.
   stepinfo      - Step response characteristics (rise time, ...)
   impulse       - Impulse response.
   initial       - Free response with initial conditions.
   lsim          - Response to user-defined input signal.
   lsiminfo      - Linear response characteristics.
   gensig        - Generate input signal for LSIM.
 Frequency-domain analysis.
   bode          - Bode diagrams of the frequency response.
   bodemag       - Bode magnitude diagram only.
```

```
   nyquist       - Nyquist plot.
   margin        - Gain and phase margins.
   bandwidth     - System bandwidth.
Model simplification.
   minreal       - Minimal realization and pole/zero cancellation.
State-space (SS) models.
   canon         - Canonical forms of state-space models.
   ctrb          - Controllability matrix.
   obsv          - Observability matrix.
Demonstrations.
   Type "demo toolbox control" for a list of available demos.
```

6.1. Властивості лінійних стаціонарних об’єктів

Як відомо, алгоритмічна мова програмного пакету *MATLAB* є об’єктно-орієнтованою. Для роботи з лінійними ста ціонарними системами (*LTI-systems*) у пакеті *Control System Toolbox* визначено батьківський клас об’єктів – *LTI-objects* (*Linear Time Invariant objects*), який поєднує у собі лінійні неперервні (*continuous*) або дискретні (*discrete*) одномірні (*SISO*) та багатомірні (*MIMO*) системи з різними формами математичного опису. Відповідно до цього розроблені дочірні об’єкти (підкласи) класу *LTI*. Найбільш важливими з них є (випустимо з розгляду дискретні системи та системи із запізненням):

1) ss (*State Space*) – *LTI*-об’єкти з математичним описом (МО) у вигляді матричних диференційних рівнянь та рівнянь виходу у просторі стану;
2) zpk (*Zero-Pole-Gain*) – розкладені на множники, що визначаються нулями та полюсами системи;
3) tf (*Transfer Function polynomial*) – *LTI*-об’єкти у формі передавальної функції, чисельник і знаменник якої представлені у вигляді поліномів.

Для поєднання всіх параметрів та властивостей (*Properties*) лінійної стаціонарної системи у єдине ціле при розробці *LTI*-об’єктів використано тип даних, що зветься масивом структур. Структури можуть отримувати у собі

різнотипні поля (*fields*), до яких, у загальному вигляді, звертаються за форматом

StructureName.FieldsName

який відповідно до специфіки LTI-об’єктів набуває виду

ObjectsName.PropertiesName

При поєднанні структур об‘єктів у двомірний масив (матрицю) першому індексу (номеру рядка) відповідає номер виходу системи, а другому індексу (номеру стовпця) – номер входу, отже звернення

ObjectsName (i, j)

виділяє з багатомірної системи (*MIMO*) одномірну підсистему (*SISO*), яка характеризує взаємозв’язок *i*-го виходу багатомірної системи з її *j*-м входом.

Імена властивостей та типи їх значень виводяться при виконанні команд

get (tf), get(zpk), get(ss)

Властивості *LTI*-об’єктів підрозділяються на загальні для всіх підкласів, які називаються родовими, та специфічні властивості, що характеризують лише конкретний підклас.

До родових властивостей *LTI*-об’єктів належать:

Ts – період дискретності або переривання або квантування за часом (від англ. *Sample Time*) у секундах, який для неперервних систем має значення Ts = 0 (призначається за замовчанням);

InputName, OutputName – імена вхідних та вихідних сигналів; задаються при наявності одного входу або виходу рядком символів, наприклад, ′Струм′, а при декількох входах або виходах – векторами комірок рядків символів, наприклад, {′M′, ′Ія′, ′W′}; значення елементів векторів за замовчанням – ′′ (пустий рядок);

Специфічні властивості ss-об’єктів:

A – матриця стану; B – матриця входу; C – матриця виходу; D – матриця прямого зв’язку входу з виходом.

Специфічні властивості zpk-об’єктів:

Z – нулі системи; P – полюси системи; K – коефіцієнт ПФ у вигляді розкладення на нулі-полюси.

Специфічні властивості tf-об’єктів:

num – коефіцієнти поліномів у чисельниках ПФ; den – коефіцієнти характеристичного поліному системи;

Загальна специфічна властивість tf-об’єктів та zpk-об’єктів – Variable, яка визначає ім’я оператора, що утворює передавальні функції; може приймати значення 's', 'p', 'z', 'z^-1'.

6.2. Створення лінійних стаціонарних об’єктів у системі *MATLAB*

Створити *LTI*-об’єкти в різних формах їх математичного опису, можна за допомогою *MATLAB*-функцій ss, zpk та tf:

```
sys_ss = ss (A, B, C, D)
sys_zp = zpk (Z, P, K)
sys_ tf = tf (num, den)
```

Ці ж самі функції можуть перетворювати існуючі об‘єкти в інший формат:

```
sys_ tf = tf (sys_ss),  sys_zp = zpk (sys_ tf), …
```

Наприклад,

```
>> h = tf ( [2 1], [ 1 2 3 10] ),    h_zp = zpk (h)
h =
            2 s + 1
  -----------------------------
  s^3 + 2 s^2 + 3 s + 10
 Continuous-time transfer function.

h_zp =
                2 (s+0.5)
  -------------------------------------------
  (s+2.445) (s^2 - 0.4454s + 4.089)
      Continuous-time zero/pole/gain model.
```

Отримати вхідні аргументи для *LTI*-об‘єкта в форматі ss можна з його - *Simulink*-моделі, в якій входи і виходи помічені відповідними портами (блоки *In* і *Out*):

[A, B, C, D] = linmod (ModelName)

Усі перераховані функції можуть бути доповнені парами властивість-значення (′PropertyName′, value). Ім’я властивості (PropertyName) може бути поданим у повному або у скороченому вигляді, достатньому для ідентифікації параметру. В іменах властивостей можна використовувати як великі, так і малі літери англійського алфавіту. Наприклад,

```
>> w = tf ( [1 0], [1 2 10] , 'var', 'p')
w =
     p
-----------------
p^2 + 2 p + 10
```

Послідовність операторів, у якій використовуються масиви комірок

```
num = { [1 3]; [1 2] };
den = { [1 1]; [1 4 5] };
H = tf (num, den)
```

утворює *LTI*-об’єкт H з одним входом та двома виходами, що описується передавальними функціями

```
H =
 From input to output...
       s + 3
1:  ---------
       s + 1

           s + 2
2:  --------------------
       s^2 + 4 s + 5
Continuous-time transfer function.
```

Точно такий же результат можна отримати виконанням операторів

```
h11 = tf ([1 3], [1 1]);
h21 = tf ([1 2], [1 4 5]);
H = [h11; h21]
```

де використано вертикальну конкатенацію двох одномірних об’єктів.

Вхідним та вихідним сигналам можна давати імена, наприклад,

```
>> H = tf (num, den, 'InputName', 'E', 'OutputName', {'M', 'w'})
H =
 From input E to output...
       s + 3
M :  ---------
       s + 1


           s + 2
 w:  -------------------
       s^2 + 4 s + 5
Continuous-time transfer function.
```

6.3. Отримання інформації про лінійні стаціонарні об’єкти

Отримати інформацію про окремі властивості *LTI*-об’єктів можна такими шляхами:

- за допомогою функції get;
- за допомогою функцій ssdata, tfdata, zpkdata;
- зверненням до полів структури.

До функції get можна звертатися за такими форматами:

```
get (sys)
Value = get (sys, ’PropertyName’)
```

Функція get (sys) виводить перелік усіх властивостей моделі з їх значеннями.

Функція get (sys, 'PropertyName') визначає інформацію про властивість ’PropertyName’ *LTI*-об’єкта з ім’ям sys.

Функція set (sys, 'PropertyName') виводить перелік припустимих значень заданої властивості.

Специфічні властивості ***одномірної системи*** простіше за все визначати за допомогою функцій другої групи:

```
[A, B, C, D] = ssdata (sys,’v’)
[Z, P, K] = zpkdata (sys,’v’)
```

```
[num, den] = tfdata (sys,'v')
```

Кількість вихідних параметрів можна зменшувати, послідовно відкидаючи їх, починаючи з останнього. Звернення до функцій без вихідних (лівосторонніх) параметрів присвоює скалярній зарезервованій змінній ans значення першого з лівосторонніх параметрів.

6.4. Редагування лінійних стаціонарних об'єктів у системі *MATLAB*

Редагування *LTI*-об'єктів можна виконати двома способами:

- за допомогою функції set;
- присвоюванням значень елементам структури.

Функція set встановлює властивості *LTI*-об'єктів і має такий синтаксис:

```
set (sys,'Property1',Value1,'Property2',Value2,…)
```

6.5. Спрощення *LTI*-моделей у системі *MATLAB*

Мінімальну реалізацію *LTI*-моделей (декомпозицію Калмана) можна отримати за допомогою функції minreal, яка виконує скорочення однакових нулів та полюсів системи:

```
sys_min = mineral (sys)
sys_min = minreal (sys, tol)
[sys_min, P] = minreal (sys, tol)
```

де tol – максимально припустима похибка перетворення (за замовчанням tol = sqrt (eps), eps – зарезервована константа *MATLAB*, що дорівнює 2.2204e-016).

6.6. Розрахунок та побудова реакцій систем на типові та довільні вхідні сигнали

Поширення *Control System Toolbox* дозволяє розрахувати і побудувати перехідні (step) та вагові (impulse) характеристики, реакцію системи на

ненульові початкові умови (initial) і виконати моделювання лінійних систем при довільних зовнішніх діях (lsim).

Функції initial, step, impulse та lsim мають такі загальні властивості:

- при зверненні до них з лівосторонніми (вихідними аргументами) вони розраховують значення відповідних сигналів без побудови графіків, а при зверненні до них без вихідних аргументів – будують відповідні графіки без збереження у пам'яті результатів розрахунку;
- вихідними аргументами функцій можуть бути [y, t, x], [y, t] та y, де y – масив вихідних координат, t – вектор-стовпець моментів часу, x – масив змінних стану;
- параметри y та x є тривимірними; для масиву y перший індекс визначає момент часу, другий – номер виходу, а третій – номер входу; позбавитися від розмірності зі значенням 1 можна застосовуючи функцію squeeze:

  y = squeeze (y),

  яка для системи з одним входом або з одним виходом перетворить тривимірний сигнал у матрицю, а для одномірної системи – у вектор-стовпець; останнє перетворення можна виконати ще простіше: y = y (:);
- параметр t може бути як присутнім, так і відсутнім у списку вхідних (правосторонніх) аргументів;
- параметр t у списку вхідних аргументів може бути заданим скаляром t_end або рівномірно розподіленим вектором t0:dt:t_end, де t_end – час закінчення розрахунку, t0 – час початку розрахунку (за замовчанням t0=0), dt – крок зміни часу (за замовчанням він назначається автоматично на підставі аналізу параметрів моделі); при відсутності параметра t у списку вхідних аргументів всі його складові назначаються відповідною функцією системи *MATLAB* автоматично;
- для будь-якої багатомірної системи кожний графік від деякого входу до деякого виходу будується у власній системі координат власного підвікна загальної графічної фігури.

Якщо спочатку розрахувати перехідні функції, а потім побудувати їх командою plot, то можна гнучкіше змінювати вигляд графіків.

Розраховані значення перехідних функцій можна використати для визначення показників якості перехідних процесів.

Більшість з показників якості перехідної функції можна отримати у такий спосіб:

```
S = stepinfo (sys)
```

або

```
S = stepinfo (sys,'RiseTimeLimits', [0,1],  'SettingTimeThreshold',0.05)
```

За замовчанням RiseTimeLimits = [0.1, 0.9] тобто параметр RiseTime визначає час зростання в межах $0{,}1y_\infty \le y \le 0{,}9y_\infty$. При RiseTimeLimits = [0,1] параметр RiseTime визначає час першого перетину перехідною функцією лінії усталеного значення: $0 \le y \le y_\infty$.

Параметр визначає SettingTimeThreshold точність визначення часу перехідного процесу (SettingTime або Steady State Time). За замовчанням SettingTimeThreshold = 0.02. Це означає, що часом закінчення перехідного процесу вважають той час, коли перехідна функція входить в зону $0{,}98y_\infty \le y \le 1{,}02y_\infty$ і більше з неї не виходить. При призначенні SettingTimeThreshold = 0.05 ця зона збільшується: $0{,}95y_\infty \le y \le 1{,}05y_\infty$, а значення параметру SettingTime зменшується.

Аналогічно до перехідних функцій можна отримати і вагові функції досліджуваної системи. Показники якості вагової функції можна знайти у такий спосіб:

```
[t, y] = impulse (sys); Simp = lsiminfo (t, y,0)
```

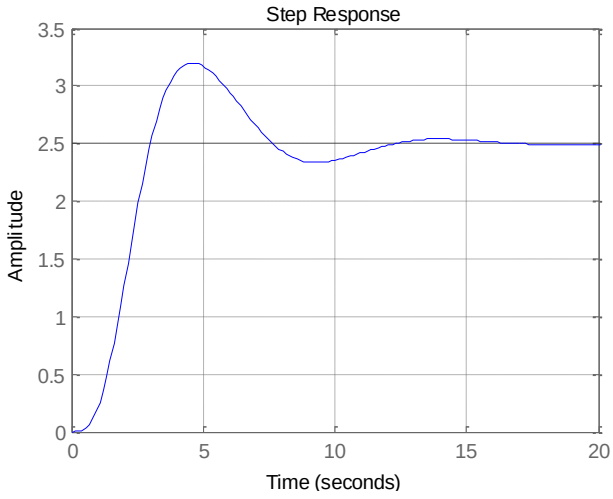
Наприклад, при виконанні програми

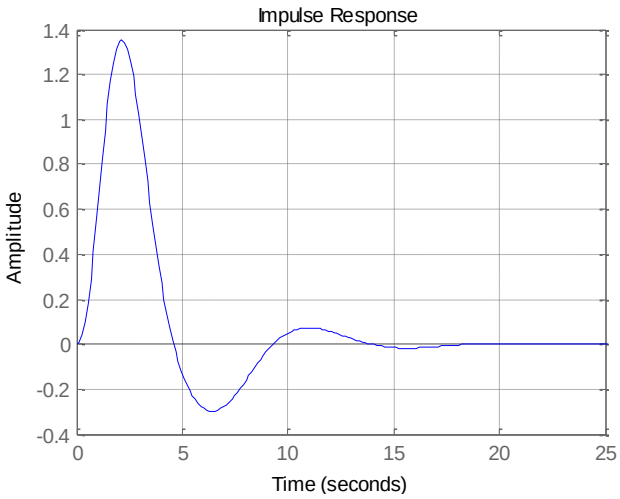
```
sys = tf ([1 5], [1 2 5 3 2 ])

figure (1), step (sys), grid
S1 = stepinfo (sys)
S2 = stepinfo (sys,'RiseTimeLimits', [0,1],  'SettingTimeThreshold',0.05)
figure (2), impulse (sys), grid
```

[t, y] = impulse (sys); S_i = lsiminfo (t, y,0) % 0 - усталене значення вагової функції

отримаємо:

```
sys =
             s + 5
  -----------------------------------
  s^4 + 2 s^3 + 5 s^2 + 3 s + 2
 Continuous-time transfer function.
S1 =
        RiseTime: 1.7023
    SettlingTime: 11.4267
     SettlingMin: 2.3036
     SettlingMax: 3.1994
      Overshoot: 27.9763
     Undershoot: 0
            Peak: 3.1994
        PeakTime: 4.5467
S2 =
        RiseTime: 2.9966
    SettlingTime: 10.3660
     SettlingMin: 2.3344
     SettlingMax: 3.1994
      Overshoot: 27.9763
     Undershoot: 0
            Peak: 3.1994
        PeakTime: 4.5467
S_i =
    SettlingTime: 12.9519
             Min: -0.2972
         MinTime: 6.4189
             Max: 1.3558
         MaxTime: 2.1396
```





а) б)

Рис. 6.1. Перехідна (а) та вагова (б) функції досліджуваної системи

Команду initial можна застосовувати тільки для *ss*-об'єктів.

Для використання команди lsim треба до вхідних параметрів додати ще узгоджений з вектором часу t вектор вхідного сигналу u:

lsim (sys, u, t)

Наприклад, лінійний сигнал, що змінюється на інтервалі $t \in [0,10]$ з одиничним прискоренням, можна сформувати операторами t =0:0.1:10; u=t.

6.7. Функції аналізу *LTI*-об'єктів у системі *MATLAB*

До складу пакету прикладних програм *Control System Toolbox* системи *MATLAB* включені функції для аналізу динамічних та статичних характеристик *LTI*-моделей:

dcgain – коефіцієнт підсилення в усталеному режимі;

pole – полюси системи;

zero – нулі одномірної системи;

tzero – нулі багатомірної системи;

order – порядок системи (кількість станів);

pzmap – карта розташування нулів та полюсів;

damp – частоти та коефіцієнти демпфірування.

Дію цих функцій покажемо на прикладі системи, розглянутої в попередньому підрозділі:

```
sys = tf ([1 5], [1 2 5 3 2 ])
k = dcgain (sys)
P = pole (sys)
Z = zero (sys)
sys_zp = zpk (sys)
n = order (sys)
damp (sys)
figure, pzmap (sys), sgrid, axis equal
```

Отримані результати:

```
sys =
```

```
            s + 5
  -------------------------------------
  s^4 + 2 s^3 + 5 s^2 + 3 s + 2
k =
    2.5000
P =
  -0.6887 + 1.7636i
  -0.6887 - 1.7636i
  -0.3113 + 0.6790i
  -0.3113 - 0.6790i
Z =
    -5

sys_zp =
                          (s+5)
  ------------------------------------------------------------
  (s^2 + 0.6225s + 0.558) (s^2 + 1.377s + 3.585)
 Continuous-time zero/pole/gain model.

n =
     4
```

Eigenvalue	Damping	Frequency
-3.11e-01 + 6.79e-01i	4.17e-01	7.47e-01
-3.11e-01 - 6.79e-01i	4.17e-01	7.47e-01
-6.89e-01 + 1.76e+00i	3.64e-01	1.89e+00
-6.89e-01 - 1.76e+00i	3.64e-01	1.89e+00

(Frequencies expressed in rad/seconds)

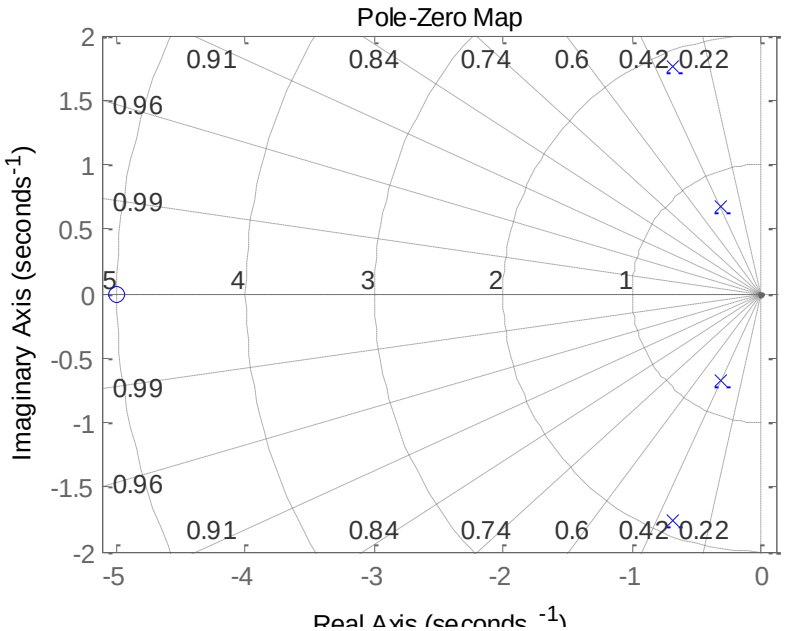


Рис. 6.2. Карта розташування нулів та полюсів

При зверненні до функції damp без лівосторонніх аргументів отримуємо таблицю параметрів Eigenvalue, Damping та Frequency .

З аналізу числових результатів видно, що власні числа матриці стану Eigenvalue є полюсами системи P (вони відрізняються тільки порядком і форматом виведення).

Частоти полюсів Frequency в рад/с (позначимо ω_p) є їх амплітудами:

$$\omega_p = A_p = |P| = \sqrt{\mathrm{Re}^2(P) + \mathrm{Im}^2(P)}\,. \qquad (6.1)$$

Коефіцієнти демпфірування Damping (позначають ξ або d) пов‘язані зі складовими полюсів формулою

$$\xi = \frac{|\mathrm{Re}(P)|}{|P|} = \frac{|\mathrm{Re}(P)|}{\omega_p}\,. \qquad (6.2)$$

Команда sgrid наносить на карту розташування нулів та полюсів кола рівних амплітуд (частот) та проміні рівних коефіцієнтів демпфірування (загасання). Отже, усі полюси або нулі, розташовані на одному колі, мають однакову амплітуду, що дорівнює радіусу кола, а всі полюси, розташовані на одному промені, мають однакові коефіцієнти демпфірування, що дорівнюють косинусу кута, між цим променем та від‘ємною частиною дійсної осі. Полюси, розташовані на уявній осі мають нульовий коефіцієнт демпфірування, що визначає систему, яка знаходиться на межі стійкості.

6.8. Розрахунок та побудова частотних характеристик

Перелік найбільш важливих функцій для дослідження частотних характеристик *LTI*-моделей наведено у табл. А1.

Функції bode та nyquist мають такі загальні властивості:

- при зверненні до них з лівосторонніми (вихідними) аргументами вони розраховують значення відповідних сигналів без побудови графіків, а при

зверненні до них без вихідних аргументів – будують відповідні графіки без збереження у пам’яті результатів розрахунку;

- розміри масивів mag (амплітуда) та phase (фаза) дорівнюють “кількість виходів” × “кількість входів” × “довжина вектору частот”;
- параметр w (частота) може бути як присутнім, так і відсутнім у списку вхідних (правосторонніх) аргументів;
- параметр w у списку вхідних аргументів може бути заданим вектором рівномірно розподіленим у логарифмічному масштабі у такий спосіб: w = logspace(w0, w_end) або {w0, w_end} (за замовчанням він назначається автоматично на підставі аналізу параметрів моделі);
- параметри досліджуваної системи можуть бути задані у початку списку вхідних аргументів у вигляді *LTI*-об’єкту будь-якого підкласу sys;

Вихідними аргументами функції bode можуть бути [mag, phase, w], [mag, phase], та mag, а вихідними аргументами функції nyquist – параметри [Re, Im, w], [Re, Im], та Re, де mag – масив амплітуд частотних характеристик у абсолютних одиницях (не в дБ), phase – масив фаз частотних характеристик у градусах, w – вектор-стовпець частот у рад/с, Re – дійсні складові годографу Найквіста, Im – уявні складові годографу Найквіста;

Функція margin обчислює запаси стійкості за фазою і модулем, а також відповідні частоти для *LTI*-моделей одномірних розімкнених систем. Значення запасів стійкості вказують, на скільки частотна характеристика розімкненої системи віддалена від критичної точки (-1, j0).

Функція margin, яку можна застосовувати тільки для *SISO*-об’єктів, без лівосторонніх аргументів будує логарифмічні частотні характеристики розімкненої системи з вказівкою запасів її стійкості. Вихідними аргументами її у максимальній кількості є [Gm, Pm, Wcg, Wcp], де Gm – запас стійкості за амплітудою, Pm – запас стійкості за фазою , Wcg – частота перетину ЛФХ рівня -180°, Wcp –частота зрізу. Якщо існують декілька точок перетину амплітудною

характеристикою осі частот і фазовою рівня -180°, то повертаються найменші значення відповідних запасів стійкості.

6.9. Контрольні питання та завдання

1. Яке призначення розділу *Control System Toolbox* програмного пакету *MATLAB*? Що таке *LTI*-системи*?*
2. Які дочерні об'єкти (підкласи) існують у батьківському класі *LTI-objects*?
3. Як подивитися імена властивостей лінійних стаціонарних об'єктів в *MATLAB*?
4. Що таке родові властивості? Назвіть їх імена?
5. Що таке специфічні властивості?
6. Як утворити різні підкласи *LTI*-об'єктів? Наведіть приклади.
7. Як виділити з *LTI*-об'єктів їх окремі властивості? Наведіть приклади.
8. Як змінити властивості *LTI*-об'єктів? Наведіть приклади.
9. Які функції призначені для розрахунку та побудови реакцій системи на типові та довільні сигнали?
10. Яка різниця між застосуванням функцій initial, step, impulse та lsim сигнали з лівосторонніми аргументами та без них? Які сигнали можуть розрахувати ці функції? Яку структуру мають розраховані дані? Як зменшити розмірність 3-вимірних масивів?
11. Як мозна задати час розрахунку перехідного процесу?
12. Як визначити основні показники якості перехідної та вагової функцій?
13. Які функції папки *control* визначають статичні та динамічні характеристики *LTI*-системи? Що це за характеристики?
14. Яка таблиця виводиться на екран привиконанні функції damp (sys)?
15. Як побудувати карту розташування нулів та полюсів на комплексній площині? Як нанести на неї лінії рівних амплітуд та лінії рівних коефіцієнтів демпфірування?
16. Які функції папки *control* розраховують та будують частотні характеристики?

7. ВИЗНАЧЕННЯ АНАЛІТИЧНИХ ВИРАЗІВ ДЛЯ ЧАСОВИХ ФУНКЦІЙ *LTI*-СИСТЕМ

7.1. Загальна характеристика методів розрахунку перехідних процесів

Графіки зміни сигналів САК у часі називають ***перехідними процесами***. Для їх побудови необхідно мати математичний опис системи. Як відомо, неперервні *LTI*-системи описуються звичайними лінійними диференціальними рівняннями з початковими умовами. Розв‘язком таких рівнянь є функції вихідних сигналів САК в залежності від часу.

З теорії диференційних рівнянь відомо, що визначення невідомого вихідного сигналу *y(t)* при відомому вхідному сигналі *u(t)* зводиться до знаходження суми вільної $Y(t)$ та вимушеної $\tilde{y}(t)$ складових загального рішення ДР:

$$y(t) = Y(t) + \tilde{y}(t). \qquad (7.1)$$

Перша складова визначається як загальне рішення однорідного диференціального рівняння (рівняння (5.8) при $u = 0$), а друга – як часткове рішення неоднорідного ДР (рівняння (5.8) при $u \neq 0$)). Однорідні ДР характеризують власні рухи систем під дією ненульових початкових умов, а неоднорідні – вимушені рухи під дією зовнішніх впливів. Такий спосіб знаходження аналітичних виразів для розрахунку перехідних процесів називають ***класичним***.

Для визначення загального рішення необхідно знайти полюси системи, які є коренями характеристичного рівняння і залежать тільки від коефіцієнтів характеристичного полінома.

Загальне рішення однорідного рівняння визначається типом полюсів, зокрема:

- якщо всі полюси дійсні і різні, то

$$Y(t) = \sum_{i=1}^{n} C_i e^{p_i t}; \qquad (7.2)$$

- для одного дійсного полюса

$$Y(t) = Ce^{pt}; \qquad (7.3)$$

- при наявності двох однакових (кратних) полюсів ($p_1 = p_2 = p$)

$$Y(t) = (C_1 + C_2 t)e^{pt}; \tag{7.4}$$

- для однієї пари комплексно-спряжених полюсів ($p_{1,2} = \alpha \pm j\beta$) експоненти вільної складової за формулою Ейлера ($e^{j\beta t} + e^{-j\beta t} = 2\cos(\beta t)$) зводяться до гармонік:

$$Y(t) = e^{\alpha t}(C_1 \cos\beta t + C_2 \sin\beta t) = Ae^{\alpha t}\sin(\beta t + \varphi); \tag{7.5}$$

$$A = \sqrt{C_1^2 + C_2^2}, \qquad \varphi = \operatorname{arctg}\frac{C_1}{C_2}. \tag{7.6}$$

У формулах (7.2)-(7.6) C_i – сталі інтегрування, які визначаються з початкових умов.

Частинне рішення неоднорідного ДР $\tilde{y}(t)$ залежить від типу вхідного сигналу. Воно визначає вимушений рух системи та її поведінку в усталеному режимі.

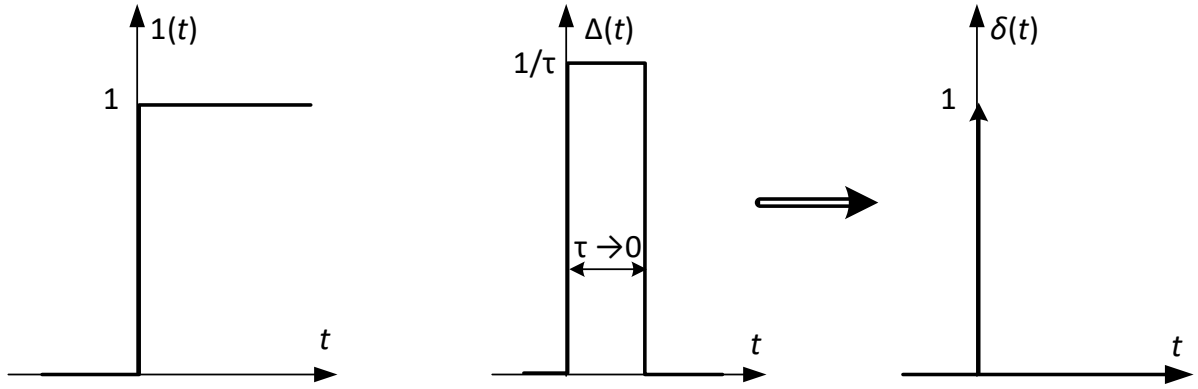
В реальних системах вхідний сигнал може бути будь-якою функцією часу. Тому для порівняння перехідних процесів в різних системах розглядають їх динаміку при так званих ***типових вхідних впливах***, в якості яких розглядаються ***одинична функція Хевісайда*** $1(t)$, (стрибкоподібний сигнал одиничної амплітуди)

$$1(t) = \begin{cases} 0 \text{ при } t<0, \\ 1 \text{ при } t\geq 0, \end{cases} \tag{7.7}$$

графік якої показано на рис. 7.1а, та ***δ-функція Дірака*** $\delta(t)$ (імпульс нескінченної амплітуди та одиничної площини):

$$\delta(t) = \lim_{\tau\to 0}\Delta(t) = \begin{cases} 0 \text{ при } t\neq 0, \\ \infty \text{ при } t=0, \end{cases} \tag{7.8}$$

де функція $\Delta(t)$ є прямокутним імпульсом шириною τ і висотою $1/\tau$ (див. рис. 7.1б). Досить часто її позначають так, як показано на рис. 7.1в.



а) б) в)

Рис. 7.1. Графіки типових вхідних сигналів:
одиничної функції Хевісайда (а) і δ-функції Дірака (б, в)

Із визначення δ-функції випливає, що вона є похідною від одиничної ступінчатої функції:

$$\delta(t)=\frac{d1(t)}{dt} \qquad \rightarrow \qquad 1(t)=\int_0^t \delta(t)dt\,. \tag{7.9}$$

Реакцію системи на одиничну функцію Хевісайда називають ***перехідною функцією*** $h(t)$ (***Step Response***), а реакцію на δ-функцію Дірака – ***ваговою функцією*** $w(t)$ (***Impulse Response***).

Із взаємозв‘язку між розглянутими типовими вхідними сигналами (7.9) випливає зв‘язок між реакціями САК на ці сигнали:

$$w(t)=\frac{dh(t)}{dt} \qquad \rightarrow \qquad h(t)=\int_0^t w(t)dt\,. \tag{7.10}$$

В теорії керування перехідну і вагову функції *LTI*-об‘єктів визначають з його передавальної функції, застосовуючи ***зворотне перетворення Лапласа***:

$$h(t)=L^{-1}\left\{\frac{W(\mathrm{s})}{s}\right\}, \qquad w(t)=L^{-1}\{W(s)\}\,. \tag{7.11}$$

7.2. Часові характеристики аперіодичної та коливальної ланок

Аперіодична і коливальна ланки є основою аналізу систем більш високого порядку, тому що

- характеристичний поліном (ХП) замкненої системи будь-якого порядку можна представити у вигляді добутку ХП аперіодичних і коливальних ланок;
- полюси замкненої системи складаються із сукупності полюсів цих елементарних ланок, а від їх розташування залежить стійкість системи і степінь її коливальності;
- якщо розкласти передавальну функцію замкненої системи на суму простих дробив, кожний з яких має характеристичний поліном аперіодичної або коливальної ланки, то на основі відомих формул для перехідної та вагової характеристики можна знайти аналітичний вираз для відповідних характеристик системи в цілому.

7.2.1. Аперіодична ланка

З передавальної функції (ПФ) аперіодичної ланки

$$W_a(p)=\frac{k}{T_a p+1}=\frac{k\omega_a}{p+\omega_a}, \qquad \omega_a=\frac{1}{T_a}, \tag{7.12}$$

записуємо її характеристичне рівняння (ХР)

$$p+\omega_a=0, \tag{7.13}$$

і визначаємо її полюс

$$p=-\omega_a, \tag{7.14}$$

який є дійсним від‘ємним числом з амплітудою ω_a, що збігається з дійсною частиною полюса.

ПФ (7.12) відповідають перехідна функція та вагова функції

$$h_a(t)=k\left(1-e^{-\omega_a t}\right), \qquad w_a(t)=k\omega_a e^{-\omega_a t}. \tag{7.15}$$

В середовищі пакету *MATLAB* вирази (7.15) можна отримати за допомогою функції ilaplace, що знаходиться в папці інструментів *Extended Symbolic Toolbox,* і здійснює зворотне перетворення Лапласа в аналітичній (символьній) формі.

Складемо програму, що знаходить вирази (7.15) за формулами (7.11):

```
syms s T  k om     % Опис символьних змінних
```

```
Wa = k*om / (s+om)      % ПФ аперіодичної ланки
wa = ilaplace (Wa)      % Вагова функція
ha = ilaplace (Wa/s)    % Перехідна функція
ha = collect (ha, k)    % Приведення подібних
```

В результаті її виконання отримуємо:

```
Wa =
(k*om)/(om + s)
wa =
k*om*exp(-om*t)
ha =
k - k*exp(-om*t)
ha =
(1 - exp(-om*t))*k
```

Графіки функцій $h_a(t)/k = h_a(t)/h_\infty$ та $w_a(t)/(k\omega_a) = w_a(t)/w_a(0)$, побудовані за рівняннями (7.15), зображено на рис. 7.2.

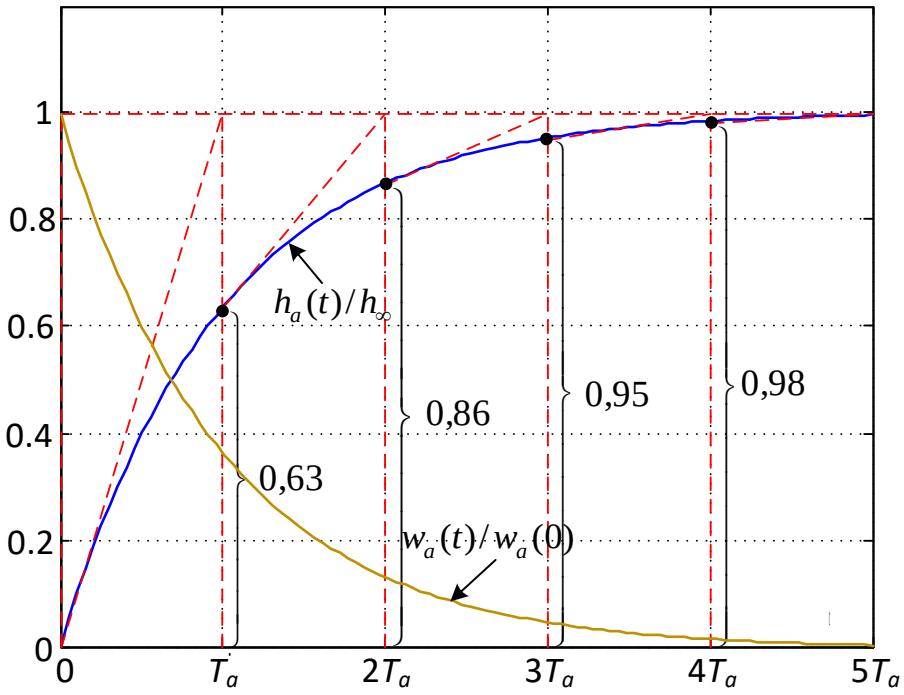


Рис. 7.2. Перехідна та вагова функції аперіодичної ланки

На ньому показані значення перехідної функції в моменти часу, кратні сталій часу T_a, з аналізу яких визначають ***час перехідного процесу (steady state time)*** з точністю 5% і 2%:

$$t_{sa(5\%)} = 3T_a = 3/\omega_a, \qquad t_{sa(2\%)} = 4T_a = 4/\omega_a. \tag{7.16}$$

Таку методику застосовують, тому що перехідна і вагова функція досягають усталених значень $h_{a\infty}=1,\ w_{a\infty}=0$ при $t_s=\infty$.

На рис. 7.2 зображені також дотичні до перехідної функції в моменти часу, кратні сталій часу аперіодичної ланки, які завжди перетинають лінію усталеного значення через час T_a.

Ця властивість випливає з наступних рівнянь:

$$h_a'(t)=w_a(t)=\omega_a e^{-\omega_a t},\ h_a(t+T_a)=h_a(t)+T_a h_a'(t)=k\left(1-e^{-\omega_a t}\right)+kT_a\omega_a e^{-\omega_a t}=k. \quad (7.17)$$

Її часто застосовують для експериментального визначення сталої часу аперіодичної ланки.

7.2.2. Коливальна ланка

ПФ коливальної ланки має вигляд

$$W_k(p)=\frac{k}{T_k^2 p^2+2\xi T_k p+1}=\frac{k\omega_k^2}{p^2+2\xi\omega_k p+\omega_k^2},\quad \omega_k=\frac{1}{T_k}, \quad (7.18)$$

де $0\le\xi<1$ – ***коефіцієнт демпфування (damping ratio) коливальної ланки.***

З характеристичного рівняння .

$$p^2+2\xi\omega_k p+\omega_k^2=0, \quad (7.19)$$

знаходимо полюси

$$p_{k1,2}=\omega_k\left(-\xi\pm j\sqrt{1-\xi^2}\right)=\alpha\pm j\beta=\omega_k e^{\pm j\varphi}. \quad (7.20)$$

Складові комплексно-спряжених полюсів пов‘язані між собою відношеннями

$$\alpha=-\omega_k\xi,\quad \beta=\omega_k\sqrt{1-\xi^2},\quad \omega_k=\sqrt{\alpha^2+\beta^2},\quad \varphi=\arccos\frac{\alpha}{\omega_k}=\arccos(-\xi). \quad (7.21)$$

Перехідна і вагова функції коливальної ланки мають вигляд:

$$h_k(t)=k\left[1-e^{-\alpha t}\left(\cos(\beta t)+\frac{\alpha}{\beta}\sin(\beta t)\right)\right],\quad w_k(t)=k\frac{\omega^2}{\beta}e^{-\alpha t}\sin(\beta t), \quad (7.22)$$

або з урахуванням рівнянь (7.21)

$$h_k(t)=1-e^{-\xi\omega_k t}\left(\cos\left(\omega_k t\sqrt{1-\xi^2}\right)+\frac{\xi}{\sqrt{1-\xi^2}}\sin\left(\omega_k t\sqrt{1-\xi^2}\right)\right), \quad (7.23)$$

$$w_k(t)=k\frac{\omega_k}{\sqrt{1-\xi^2}}e^{-\xi\omega_k t}\sin\left(\omega_k t\sqrt{1-\xi^2}\right). \quad (7.24)$$

Отримати вирази (7.22) можна при виконанні програми

```
syms s k om al bet t              % Опис символьних змінних
s1=-al+j*bet,  s2=- al-j*bet  % комплексно спряжені полюси коливальної ланки
Wk=k*om^2/(s-s1)/(s-s2)     % ПФ коливальної ланки
Wk=simplify(Wk)                 % спрощеня ПФ
wk=ilaplace(Wk)                 % Вагова функція
disp ('wk(t)=') , pretty(wk)
hk = int(wk,0,t)                  % Перехідна функція як визначений інтеграл від вагової
hk = subs (hk, al^2+bet^2, om^2)     % Підстановка
hk = collect (hk, k)        % Приведення подібних
disp ('hk(t)=') , pretty(hk)
```

В результаті її виконання отримуємо:

```
s1 =
- al + bet*i
s2 =
- al - bet*i
Wk =
(k*om^2)/((al - bet*i + s)*(al + bet*i + s))
Wk =
(k*om^2)/(al^2 + 2*al*s + bet^2 + s^2)
wk =
(k*om^2*exp(-al*t)*sin(bet*t))/bet
wk(t) =
         2
  k om  exp(-al t) sin(bet t)
  ---------------------------------
             bet

hk =

(k*om^2)/(al^2 + bet^2) - (k*om^2*exp(-al*t)*(bet*cos(bet*t) + al*sin(bet*t)))/(bet*(al^2 +
+bet^2))

hk =

k - (k*exp(-al*t)*(bet*cos(bet*t) + al*sin(bet*t)))/bet

hk =
(1 - (exp(-al*t)*(bet*cos(bet*t) + al*sin(bet*t)))/bet)*k
```

```
hk(t) =
  /     exp(-al t) (bet cos(bet t) + al sin(bet t)) \
 | 1 - ---------------------------------------------- | k
  \                      bet                         /
```

Графік перехідної та вагової функцій коливальної ланки та огинаючих її експонент

$$e_{1,2}(t)=1\pm\exp\left(-\frac{\alpha}{\beta}\omega_k t\right) \qquad (7.25)$$

при $k=1$, $\omega_k=1$ і $\xi=0,25$ представлено на рис. 7.3.

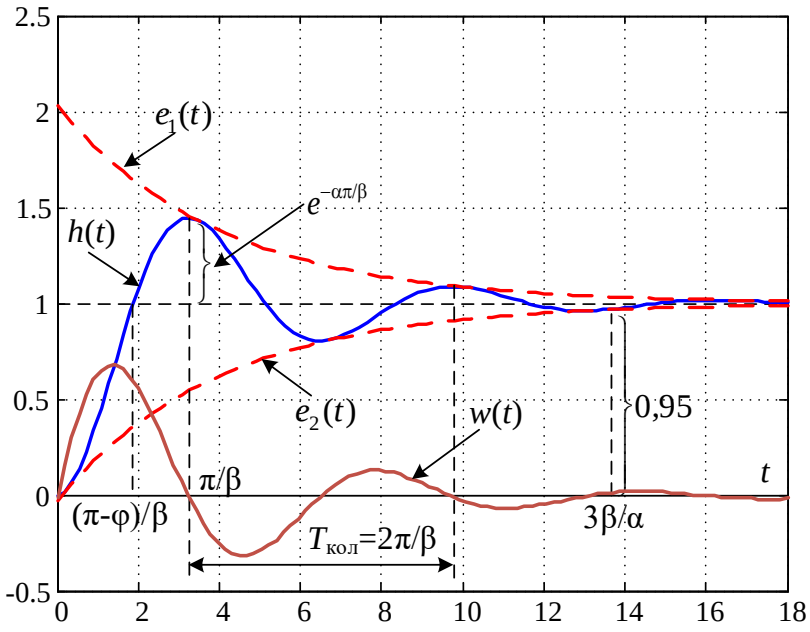


Рис. 7.3. Перехідна функція коливальної ланки та її огинаючих експонент

При коливальному характері перехідного процесу показників якості перехідних процесів значно більше, ніж при аперіодичному. Крім ***часу перехідного процесу (steady state time)***, який визначається за рівняннями, аналогічними до (7.10):

$$t_{sk(5\%)}\approx 3/\alpha, \qquad t_{sk(2\%)}\approx 4/\alpha \qquad (7.26)$$

до них відносяться:

- ***час першого узгодження перехідної функції з усталеним значенням (rise time)***, що визначається з рівняння $h_k(t_r)=1$

$$t_r = \frac{\pi - \psi}{\beta}, \qquad \psi = \operatorname{arctg}\frac{|\beta|}{|\alpha|}; \tag{7.27}$$

- ***період коливань***

$$T_{\text{кол}} = \frac{2\pi}{\beta}; \tag{7.28}$$

- ***час досягнення перехідною функцією максимального значення (peak time)***, що дорівнює половині періоду коливань:

$$t_m = \frac{T_{\text{кол}}}{2} = \frac{\pi}{\beta}; \tag{7.29}$$

- максимум перехідної функції

$$h_m = h(t_m) = k\left[1 + \exp\left(-\frac{\alpha\pi}{\beta}\right)\right] = k\left[1 + \exp\left(-\frac{\xi\pi}{\sqrt{1-\xi^2}}\right)\right]; \tag{7.30}$$

- другий максимум перехідної функції

$$h_{m2} = h(t_m + T_{\text{кол}}) = k\left[1 + \exp\left(-\frac{3\alpha\pi}{\beta}\right)\right]; \tag{7.31}$$

- ***перерегулювання перехідної функції (overshot або overshoot)***

$$\sigma \approx \exp\left(-\frac{\lambda\pi}{\beta}\right) = \exp\left(-\frac{\xi\pi}{\sqrt{1-\xi^2}}\right); \tag{7.32}$$

- ***ступінь загасання***

$$\psi_d = \left(1 - \frac{h_{m2} - h_\infty}{h_m - h_\infty}\right) = \left(1 - \frac{\exp(3\pi\alpha/\beta)}{\exp(\pi\alpha/\beta)}\right) = 1 - \sigma^2. \tag{7.33}$$

Останні 2 параметри найчастіше розраховують у процентах.

Графіки залежності перерегулювання перехідної функції коливальної ланки і ступені її загасання від коефіцієнту демпфування у процентах, зображені на рис. 7.4.

З рис. 7.4 видно, при зменшенні коефіцієнта демпфування перерегулювання збільшується, а ступінь загасання зменшується. При $\xi=0$, коли комплексно-спряжена пара полюсів розташовується на уявній осі, тобто

на межі стійкості, перехідна функція здійснює не загасаючі коливання (ступінь загасання дорівнює 0) з перерегулюванням 100%. При $\xi \to 1$ комплексно-спряжена пара полюсів вироджується у пару кратних дійсних полюсів, а перехідна функція має аперіодичний характер ($\sigma = 0$). При $0{,}65 < \xi < 0$ ступінь загасання перехідної функції складає практично 100%. Це означає, що навіть при наявності невеликого перерегулювання другого максимуму функції не існує, тобто перехідна функція здійснює тільки 1 коливання.

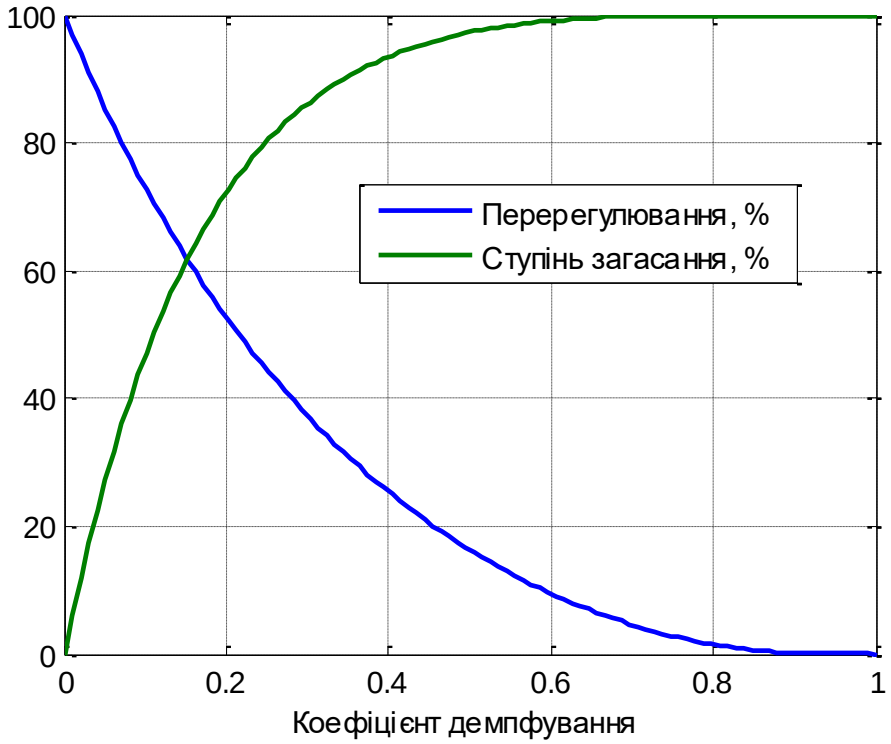


Рис. 7.4. Графік залежності перерегулювання перехідної функції коливальної ланки та ступені її загасання від коефіцієнту демпфування

Ці висновки підтверджуються аналізом сімейства перехідних функцій коливальних ланок з різними коефіцієнтами демпфування, представленим на рис. 7.5.

Коливальну ланку з нульовим демпфуванням ($\xi = 0$) називають консервативною. Її ПФ має вигляд

$$W_{kon}(p) = \frac{k}{T_k^2 p^2 + 1} = \frac{k\omega_k^2}{p^2 + \omega_k^2}. \qquad (7.34)$$

Вона має 2 комплексно-спряжених полюси з нульовою дійсною частиною

$$p_{1,2} = \pm j\omega_k , \qquad (7.35)$$

що розташовуються на дійсній осі (межа стійкості), а перехідна і вагова функції описуються рівняннями

$$h_{kon}(t)=k\left[1-\cos(\omega_k t)\right], \qquad w_{kon}(t)=h_{kon}'(t)=k\omega\cdot\sin(\omega_k t). \qquad (7.36)$$

Графіки цих функцій подані на рис. 7.6.

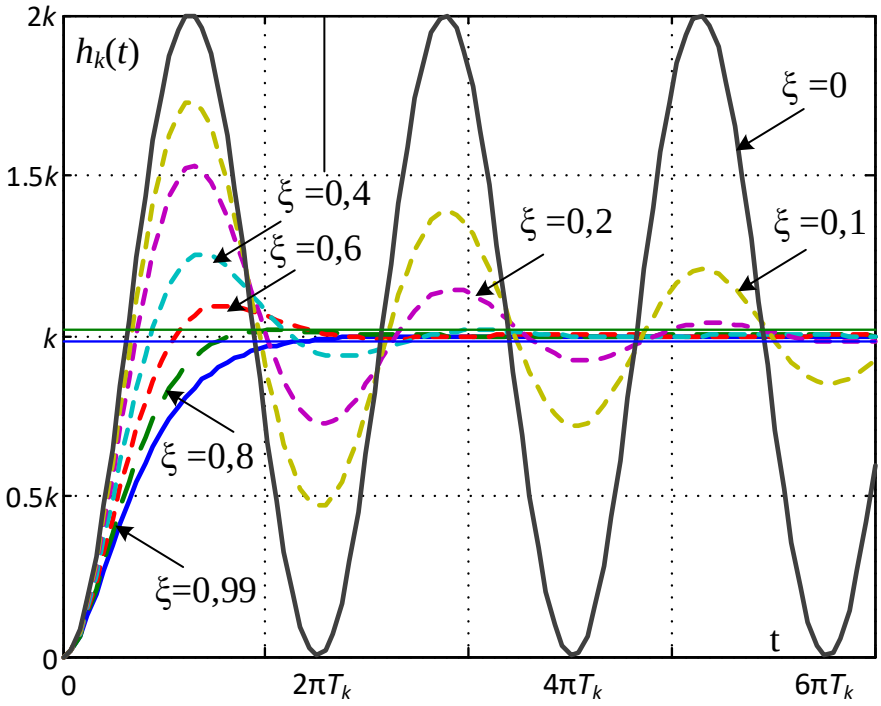


Рис. 7.5. Сімейство перехідних функцій коливальних ланок з різними коефіцієнтами демпфування

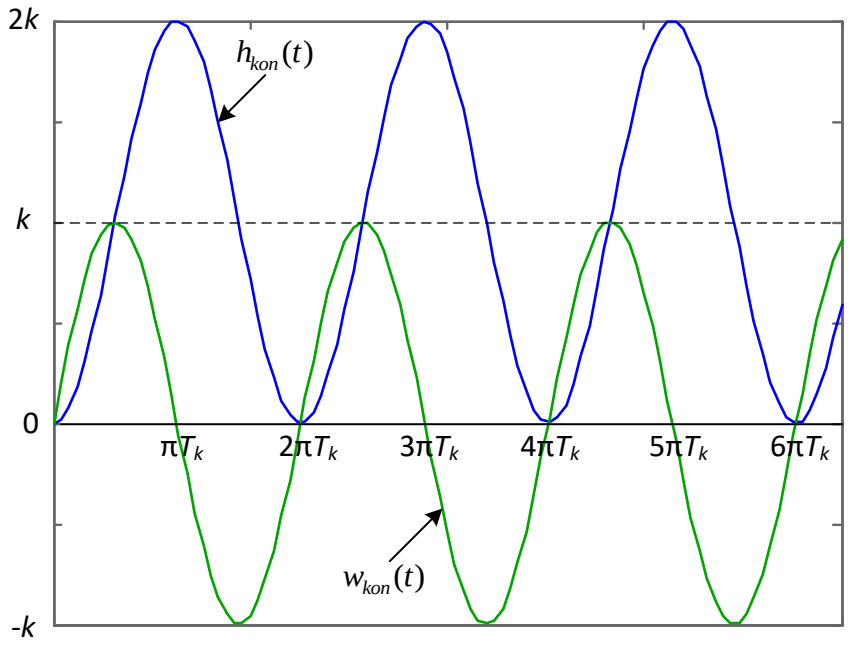


Рис. 7.6. Перехідна та вагова функції консервативної ланки

7.3. Зв‘язок між розташуванням полюсів та показниками якості перехідних процесів

На рис. 7.7 зображено розташування дійсного полюса і пари комплексно-спряжених полюсів на комплексній площині та позначені їх основні параметри.

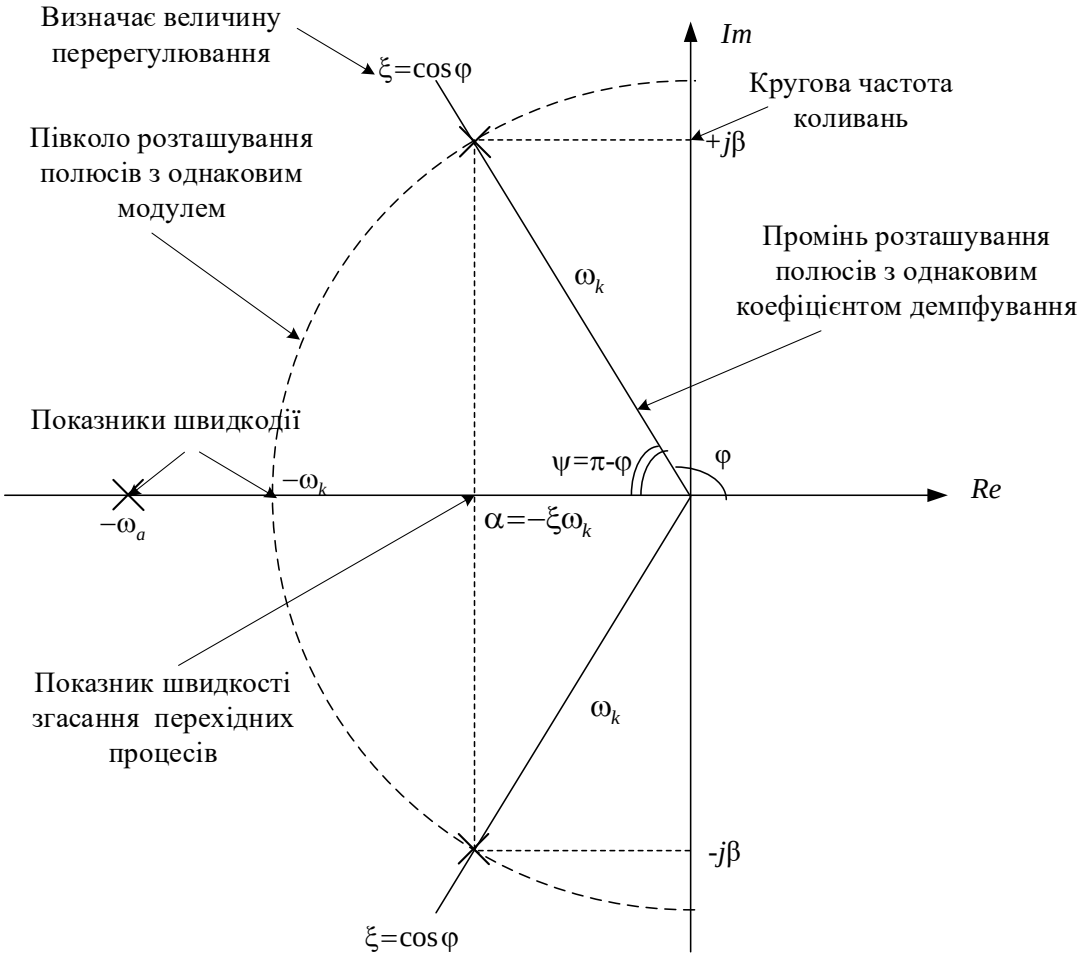


Рис. 7.7. Розташування полюсів аперіодичної та коливальної ланки на комплексній площині

Як бачимо, полюс аперіодичної ланки розташований на дійсній осі в точці $-\omega_a$, а полюси коливальної ланки – на перетині півкола з радіусом ω_k і променів, розташованих під кутами $\pm\varphi=\pm\arccos(\xi)$ до дійсної осі.

З рівняння (7.23) випливає, що ***форма кожної із коливальних складових перехідних функцій залежить тільки від кута*** φ***, тобто від коефіцієнта демпфування*** ξ***. Чим менше коефіцієнт демпфування, тим більше перерегулювання перехідної функції і тим менше ступінь її загасання.***

Від модулів полюсів (відстані їх від центру системи координат Re-Im) залежить тільки масштаб часу (швидкодія) окремої складової перехідного процесу: чим більший модуль, тим більша швидкодія цієї складової.

З порівняння карти розташування полюсів з графіками перехідних функцій випливає: час загасання перехідного процесу залежить від дійсної частини полюсів

$$t_{sa(2\%)} = 4/\omega_a \ , \qquad t_{sk(2\%)} \approx 4/\alpha ,$$

період коливань – від уявної частини

$$T_{\text{кол}} = \frac{2\pi}{\beta},$$

перерегулювання і коливальність перехідних процесів – від кута, що утворюється між променем, на якому розташований полюс і дійсною віссю

$$\sigma = \exp\left(-\frac{\xi\pi}{\sqrt{1-\xi^2}} \right), \qquad \xi = -\cos\varphi .$$

З аналізу рис. 7.7 випливає спосіб градуювання карти розташування нулів-полюсів променями однакових коефіцієнтів демпфування, показаний на рис. 7.8.

Для цього треба

- побудувати у лівій півплощині комплексної площини півколо одиничного радіусу;
- обрати на дійсній осі точки, абсциси яких дорівнюють бажаним значенням коефіцієнту демпфування;
- із кожної точки провести прямі, паралельні уявній осі, до перетину з півколом;
- побудувати промені, що проходять через центр координат і точки перетину.

Таке градуювання застосовує при побудові карти нулів-полюсів функція pzmap.

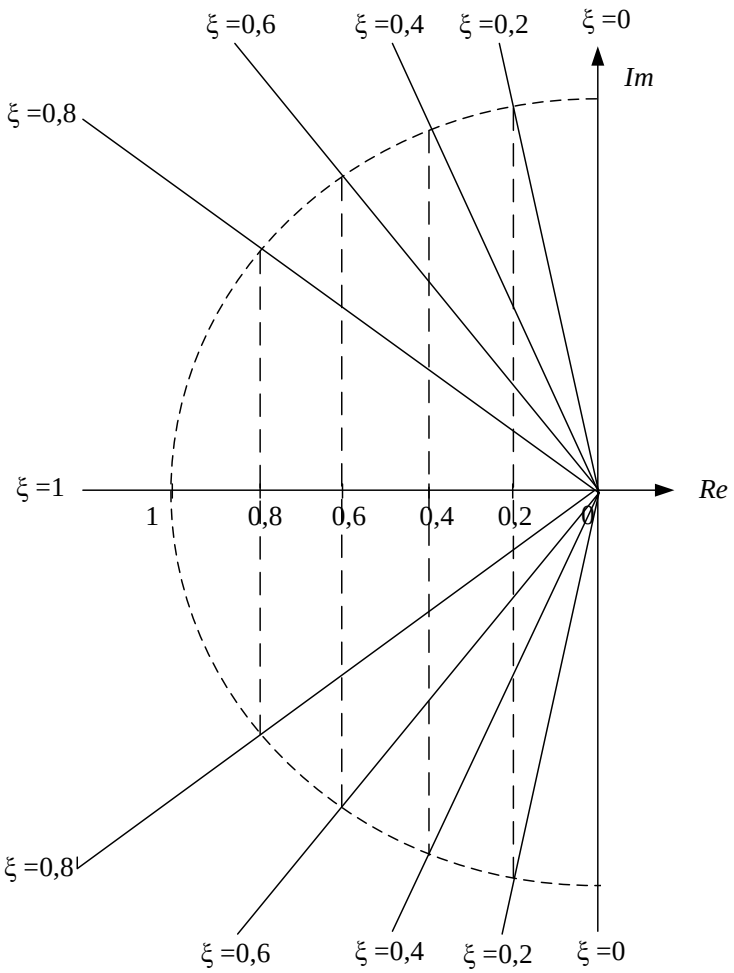


Рис. 7.8. Градуювання карти розташування нулів-полюсів лініями однакового демпфування

Наприклад, при виконанні оператора

```
obj = tf ([1 1], [1 4 8 4 1])
[P, Z] = pzmap (obj)
pzmap (obj), grid on
```

отримуємо

```
obj =
                s + 1
  -----------------------------------
s^4 + 4 s^3 + 8 s^2 + 4 s + 1
Continuous-time transfer function.
P =
 -1.7071 + 1.7071i
-1.7071 - 1.7071i
 -0.2929 + 0.2929i
-0.2929 - 0.2929i
Z =
   -1
```

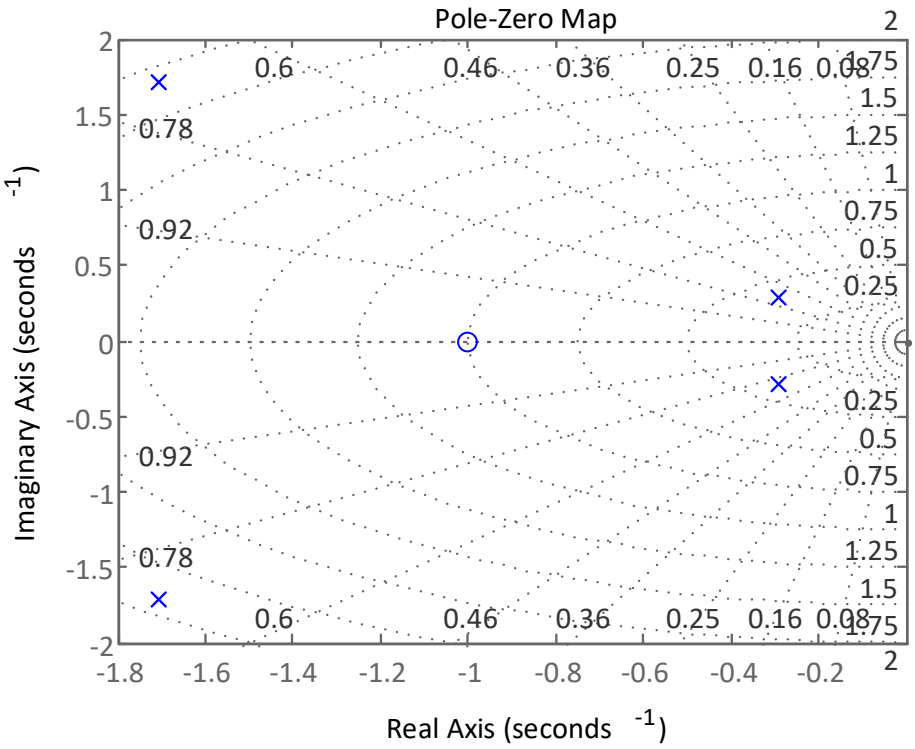


Рис. 7.9. Карта розташування полюсів об’єкта tf ([1 1], [1 4 8 4 1])

За розташуванням полюсів аналізують ***стійкість системи (Stability)***.

Відомо, що ***у стійкої системи (Stable System) усі полюси розташовуються у лівій півплощині комплексної площини, тобто дійсні частини усіх полюсів є від’ємними.***

Якщо хоча б одна пара комплексно-спряжених полюсів системи розташована на уявній осі (має нульові дійсні частини), то система знаходиться на межі стійкості. Виключенням з цього правила э розташування полюса в центрі координат**,** який зумовлений наявністю інтегральної ланки. Такий полюс називають нейтральним.

Кожний полюс системи впливає на якість її перехідних процесів, але ступінь їх впливу залежить від відстані полюсів від уявної осі: чим менше ця відстань, тим більший його внесок у перебіг перехідного процесу. Тому при аналізі систем зазвичай виділяють ***домінуючий дійсний полюс*** або ***домінуючу пару комплексно-спряжених полюсів*** за критерієм найбільшої близькості їх до уявної осі і аналізують властивості системи першого або другого порядку, яка приблизно відображає властивості досліджуваної системи.

Отже, визначивши домінуючий полюс або домінуючу пару комплексно-спряжених полюсів, можна приблизно оцінити показники перехідних процесів системи в цілому. Точність оцінювання буде тим більше, чим більше відношення відстані домінуючих полюсів від уявної осі до відстаней від неї інших полюсів.

7.4. Визначення аналітичних виразів для перехідних функцій систем з передавальними функціями довільного вигляду

За відомими перехідною та ваговою функціями аперіодичної ланки досить просто знайти перехідні функції систем, ПФ яких відрізняється від ПФ аперіодичної ланки чисельником. Наприклад, для системи з ПФ

$$W_d(p)=\frac{k_d p}{T_a p+1} \tag{7.37}$$

$$h_d(t)=k_d w_a(t)/k=k_d\omega_a \exp(-\omega_a t), \tag{7.38}$$

а для системи з ПФ

$$W_f(p)=\frac{k(T_f p+1)}{T_a p+1}=\frac{k}{T_a p+1}+\frac{kT_f p}{T_a p+1} \tag{7.39}$$

$$h_f(t)=h_a(t)+T_f w_a(t)=k\left(1-(T_a-T_f)\omega_a \exp(-\omega_a t)\right). \tag{7.40}$$

У такий же спосіб можна визначити вирази перехідних функцій для систем другого порядку з ПФ

$$\frac{k_d p}{T_k^2 p^2+2\xi T_k p+1},\quad \frac{k(T_f p+1)}{T_k^2 p^2+2\xi T_k p+1},\quad \frac{k_d p(T_f p+1)}{T_k^2 p^2+2\xi T_k p+1},\ldots$$

Для того, щоб знайти вирази для перехідних і вагових функцій систем більш високого порядку застосовують ***розкладання передавальних функцій загального вигляду на суму простих дробів***. Таке розкладання називають ***Partial-Fraction Decomposition*** або паралельна декомпозиція, тому що уданому випадку ПФ високого порядку еквівалентно замінюється паралельним з‘єднанням декількох ланок першого та (або) другого порядку.

При відсутності в системі кратних полюсів таке розкладення має вигляд:

$$W(p)=\frac{y(p)}{u(p)}=\frac{R_1}{(p-p_1)}+\frac{R_2}{(p-p_2)}+...+\frac{R_n}{(p-p_n)}. \qquad (7.41)$$

Перетворення Transfer Function Polynomial → Partial-Fraction Decomposition виконує в *MATLAB* функція

```
[R, P] = residue (num, den)
```

Приклад 7.1. *Знайти перехідну функцію системи з ПФ*

$$W(p)=\frac{3(p+1)}{p^3+9p^2+40p+100}$$

шляхом розкладання її на суму простих дробив.

Знайдемо елементи розкладання в середовищі MATLAB:

```
>>[R, P] = residue ([3 3], [1 9 40 100])
R =
  -0.48 +       0i
   0.24 -  0.195i
   0.24 + 0.195i
P =
  -5 + 0i
  -2 + 4i
  -2  - 4i
```

За цими даними складаємо ПФ

$$W(p)=\frac{-0,48}{p+5}+\frac{0,24-0,195j}{p+2-4j}+\frac{0,24+0,195j}{p+2+4j}$$

Щоб позбавитись від комплексних чисел, об'єднаємо 2 останні ланки з комплексно спряженими полюсами і коефіцієнтами в одну:

$$W(p)=\frac{-0,48}{p+5}+\frac{0,48p+2,52}{p^2+4p+20}.$$

Цю операцію можна здійснити в MATLAB за допомогою таких операторів:

```
den_k = conv ([1 -P(2)], [1 -P(3)])
num_k = conv ([1 -P(2)], R(3))+conv ([1 -P(3)], R(2))
```

що дають результат

```
den_k =
          1          4         20
num_k =
       0.48       2.52
```

У такий спосіб ми перетворюємо вихідну поліноміальну функцію 3-го порядку у суму дробив 1-го та 2-го порядків.

За принципом суперпозиції перехідну функцію досліджуваної системи можна знайти як суму перехідних і вагових функцій аперіодичної і коливальної ланок з відповідними коефіцієнтами.

$$h(t)=-\frac{0,48}{5}\left(1-e^{-5t}\right)+\frac{2,52}{20}\left[1-e^{-2t}\left(\cos(4t)+\frac{2}{4}\sin(4t)\right)\right]+\frac{0,48}{20}\cdot\frac{20}{4}e^{-2t}\sin(4t).$$

Після зведення подібних маємо

$$h(t)=0,03+0,096e^{-5t}+e^{-2t}(-0,126\cos(4t)+2,337\sin(4t).$$

У правильності виконаного розкладення можна впевнитися, виконавши зворотні операції:

```
den = conv ([1 5], [1 4 20])
num = -0.48*[1 4 20] +conv ([1 5], [0.48 2.52])
```

що дають результат

```
den =
     1     9    40   100
num =
     0     3     3
```

Інший спосіб полягає у порівнянні результатів моделювання системи з вихідною ПФ та її паралельної декомпозиції.

Отже, ПФ у форматі *Partial-Fraction Decomposition* зазвичай використовують для виведення аналітичних виразів перехідних функцій.

7.5. Контрольні питання і завдання

1. У чому полягає класичний метод розрахунку перехідних процесів?
2. Наведіть приклад однорідного та неоднорідного диференціального рівняння (ДР) 2-го порядку.
3. Від яких параметрів системи залежить загальне рішення однорідного ДР? Як вони визначаються?
4. Напишіть формули для загального рішення однорідного ДР для системи з одним дійсним полюсом та для систем з однією парою комплексно-спряжених полюсів.
5. Від чого залежить частинне рішення неоднорідного ДР?
6. Які вхідні сигнали називаються типовими? Як називають реакції на ці сигнали?

7. Якими математичними виразами пов‘язані між собою одинична функція Хевісайда і δ-функції Дірака, перехідна і вагова функції?
8. Якими математичними залежностями пов‘язані з передавальною функцією перехідна і вагова функції?
9. Напишіть передавальну, перехідну і вагову функції аперіодичної ланки. Визначте її полюс.
10. Чому дорівнюють значення перехідної функції аперіодичної ланки в моменти часу, кратні її сталій часу?
11. Як можна експериментально визначити параметри аперіодичної ланки?
12. Напишіть передавальну, перехідну і вагову функції коливальної ланки. Визначте її полюси і зв‘язки між складовими полюсів.
13. Напишіть формули для часу перехідного процесу, періоду коливань і перерегулювання перехідної функції коливальної ланки.
14. Які полюси називають домінуючими? Як впливає розташування домінуючих полюсів на якість перехідних процесів?
15. Для САК, заданої в курсовій роботі, знайти нулі і полюси, побудувати карту розташування нулів і полюсів на комплексній площині, визначити домінуючий полюс або домінуючу пару комплексно-спряжених полюсів і знайти приблизні показники якості перехідної функції. Порівняти їх з дійсними показниками.
16. Як градуюється в *MATLAB* карта розташування нулів-полюсів?
17. Визначити рівняння перехідної функції для ланки з передавальною функцією

$$W(p)=\frac{k(T_f p+1)}{T_k^2 p^2+2\xi T_k p+1}$$

18. Зробіть розкладення на суму простих дробив ПФ за керувальною дією системи, досліджуваної в курсовій роботі. Напишіть рівняння перехідної функції системи через перехідні та вагові функції аперіодичної та коливальної ланок. Зберіть модель, що складається з паралельного з‘єднання

виявлених елементарних ланок і отримайте перхідні функції окремих доданків і їх суму.

8. ПОБУДОВА ЧАСТОТНИХ ХАРАКТЕРИСТИК

8.1. Визначення частотних характеристик

Важливу роль в аналізі динамічних систем відіграють характеристики, що отримані на основі математичного опису у частотній області. Для визначення частотних характеристик (ЧХ) необхідно знайти частотну передавальну функцію.

Частотну передавальну функцію W(jω) системи можна отримати підстановкою $p = j\omega$ у звичайну передавальну функцію *W(p)* (ω – частота, $j=\sqrt{-1}$ – уявна одиниця):

$$W(j\omega)=W(p)\big|_{p=j\omega}. \tag{8.1}$$

Таку функцію завжди можна перетворити до вигляду

$$W(j\omega) = W_{re}(\omega) + jW_{im}(\omega) \tag{8.2}$$

або

$$W(j\omega) = A(\omega)e^{j\varphi(\omega)}. \tag{8.3}$$

Форма запису (8.2) називається ***алгебраїчною***. Її складовими є ***дійсна*** $W_{re}(\omega)$ та ***уявна*** $W_{im}(\omega)$ частини. Форма запису (8.3) називається ***показниковою або експоненційною формою***. Її складовими є ***амплітуда*** $A(\omega)$ та ***фаза*** $\varphi(\omega)$.

Складові формул (8.2) та (8.3) пов‘язані між собою співвідношеннями:

$$A = |W(j\omega)| = \sqrt{W_{re}^2 + W_{im}^2}; \tag{8.4}$$

$$\varphi = \arg(W(j\omega)) = \operatorname{arctg}(W_{im} / W_{re}); \tag{8.5}$$

$$W_{re} = A\cos\varphi; \tag{8.6}$$

$$W_{im} = A\sin\varphi. \tag{8.7}$$

Підстановкою (8.6) та (8.7) в рівняння (8.2) отримують тригонометричну форму запису комплексних чисел:

$$W = A(\cos\varphi + j\sin\varphi). \qquad (8.8)$$

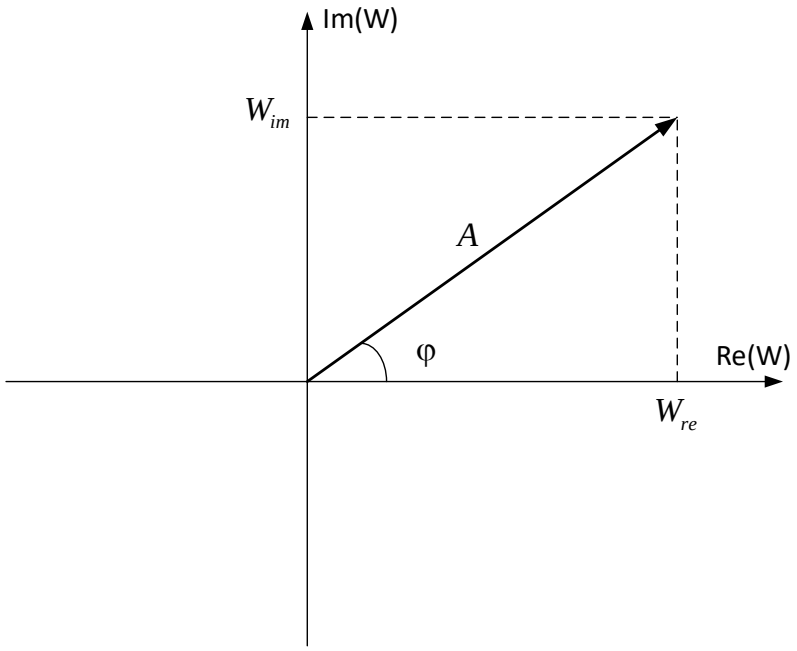


Рис. 8.1. Зображення комплексних чисел на комплексній площині

На комплексній площині комплексні числа зображуються у вигляді векторів, як це показано на рис. 8.1.

При частотному аналізі сигналів та динамічних систем застосовують такі частотні характеристики:

$W_{re}(\omega)$ – ***дійсна частотна характеристика*** (***ДЧХ***)***;***

$W_{im}(\omega)$ – ***уявна частотна характеристика*** (***УЧХ***);

$A(\omega)$ – ***амплітудно- частотна характеристика*** (***АЧХ***);

$\varphi(\omega)$ – ***фазо- частотна характеристика*** (***ФЧХ***);

$A(\varphi)$ – ***амплітудно-фазова частотна характеристика*** (***АФЧХ***);

$20\lg A(\lg\omega)$ – ***логарифмічна амплітудно- частотна характеристика*** (***ЛАЧХ***);

$\varphi(\lg\omega)$ – ***фазо-частотна характеристика*** (***ЛФЧХ***).

АФЧХ називають ***годографом або діаграмою Найквіста*** (***Nyquist Plot***).

ЛАЧХ та ЛФЧХ називають також ***діаграмами Боде*** (***Bode Plots***).

Амплітудні та фазові частотні характеристики відображують властивості реакції досліджуваного динамічного об‘єкту на синусоїдальний вплив в усталеному режимі. Наприклад, якщо на вхід об‘єкту подати синусоїдальний сигнал

$$u(t)=A_{in}\sin(\omega t+\varphi_{in}), \tag{8.9}$$

то після закінчення перехідного процесу на виході системи спостерігатиметься синусоїдальний сигнал з тією ж частотою але з іншими амплітудою та фазою:

$$y(t)=A_{out}\sin(\omega t+\varphi_{out}). \tag{8.10}$$

Відношення амплітуди вихідної синусоїди до амплітуди вхідної синусоїди є амплітудою частотних характеристик, а різниця фаз (фазовий зсув) – фазою частотних характеристик:

$$A(\omega)=\frac{A_{out}}{A_{in}}, \qquad \varphi(\omega)=\varphi_{out}-\varphi_{in}. \tag{8.11}$$

При застосуванні формули

$$L(\omega)=20\lg A(\omega) \tag{8.12}$$

отримують амплітуду в децибелах (скорочено дБ або dB).

8.2. Використання функцій ядра пакету *MATLAB* для побудови частотних характеристик

Для розрахунку комплексної ПФ, по-перше, необхідно створити вектор частот. Для побудови ДЧХ, УЧХ, АЧХ і ФЧХ його доцільно зробити рівномірно розподіленим впродовж лінійної осі на інтервалі $\omega\in[0...\omega_k]$ за допомогою оператора

```
w = linspace (0, wk);
```

або

```
w = 0:dw:wk;
```

Якщо ПФ системи має невисокий порядок, то далі позначаємо

```
p = j*w;
```

і підставляємо розраховану змінну p у ПФ, використовуючи поелементні

математичні операції. Наприклад, якщо $W(p)=10/(0,2\,p^2+0,3p+1)$, то вектор комплексних чисел, що змінюють свої значення у функції частоти, отримують виконанням оператора

```
W = 10 ./ (0.2*p.^2+0.3*p+1);
```

Для ПФ більш високого порядку, для яких вже знайдені коефіцієнти поліномів чисельника num та знаменника den, краще застосувати функції, що здійснюють операції зі степеневими поліномами:

```
H = polyval (num, p);
G = polyval (den, p);
W = H ./ G;
```

Розрахувати параметри отриманих комплексних чисел за формулами (8.4)-(8.7) можна за допомогою функцій real, imag, abs, angle:

```
U = real (W);      V = imag (W);
A = abs (W);     Phi = angle (W);
Phi_deg = Phi ./pi * 180;
```

Останній оператор перетворює фазу в радіанах у фазу в градусах.

За результатами наведених розрахунків можна побудувати перелічені вище ЧХ у такий спосіб:

```
figure (1),  plot (w, U, w, V), legend ('ДЧХ','МЧХ')
figure (2), subplot (2,1,1),  plot (w, A), grid on, title ('АЧХ')
subplot (2,1,2),  plot (w, Phi_deg), grid on, title ('ФЧХ')
```

Для побудови діаграм Боде, які найчастіше будують для розімкнених систем, елементи вектору частот рівномірно розподіляють впродовж логарифмічної осі на інтервалі $\lg(\omega_{\log})\in\left[d_0 \ldots d_k\right]$, що відповідає діапазону частот $\omega_{\log}\in\left[10^{d_0} \ldots 10^{d_k}\right]$, для чого застосовують функцію logspace:

```
w_lg = logspace (d0, dk);
```

а графіки будують у напівлогарифмічних осях функцією semiogx:

```
p_lg = j*w_lg;
Hr = polyval (num_r, p_lg);
```

```
Gr = polyval (den_r, p_lg);
Wr = Hr ./ Gr;
Ar = abs (W_lg);     L = 20*log10(Ar);
Phir = angle (Wr);    Phir_deg = Phir ./pi * 180;
figure (3), subplot (2,1,1),  semilogx (w_lg, L), grid on, title ('ЛАЧХ')
subplot (2,1,2),  plot (w_lg, Phir_deg), grid on, title ('ЛФЧХ')
```

Через те, що при використанні логарифмічної осі частот точка, що відповідає $\omega = 0$, знаходиться зліва у нескінченості ($\lg 0 = -\infty$), нижня межа діапазону частот при побудові ЛЧХ має не нульове, а достатньо мале але скінченне значення.

АФЧХ можна будувати як в Декартових, так і в полярних координатах:

```
plot (U, V),  axis equal
polar (A, Phi)
```

Найскладнішим в даному випадку є формування вектору частот, який в ідеалі повинен мати діапазон від 0 до ∞. Щоб годограф змінювався плавно, необхідно дуже ретельно підбирати значення частот і переходити до границь змінних, щоб розрахувати їх значення при $\omega \to \infty$.

8.3. Використання функцій *Control System Toolbox* та *Signal Toolbox* пакету *MATLAB*

У розділах *Control System Toolbox* та *Signal Toolbox* знаходяться функції, які в багатьох випадках спрощують побудову і розрахунок частотних характеристик. Обов'язковим вхідним (правобічним) параметром цих функцій повинен бути *LTI*-об'єкт sys у будь-якій формі або його параметри, що визначають математичний опис системи. Простіше за все утворити такий об'єкт через коефіцієнти поліномів чисельника та знаменника ПФ:

```
sys = tf (num, den)
```

Функції bode і nyquist без лівосторонніх (вихідних) параметрів

```
bode (sys)                    nyquist (sys)
bode (sys, w)                 nyquist (sys, w)
```

одразу будують діаграми Боде і Найквіста без збереження у пам’яті результатів їх розрахунку. Частота w у списку вхідних аргументів може бути задана вектором рівномірно розподіленим у логарифмічному масштабі таким чином: w = logspace (log10(w0), log10(wk)) або {w0, wk}. За замовчанням він назначається автоматично на підставі аналізу параметрів моделі.

Описані вище функції можна застосовувати з лівосторонніми параметрами mag – амплітуда від англ. *magnitude*, phase – фаза в градусах, re – дійсна частина, im – уявна частина:

```
[mag, phase, w] = bode (sys)        [re, im, w] = nyquist (sys)
[mag, phase] = bode (sys, w)        [re, im] = nyquist (sys, w)
```

У цьому разі вони тільки розраховують дані для побудови ЧХ, а користувач використовує цю інформацію для побудови будь-яких ЧХ у будь якому вигляді.

Вихідні параметри розглянутих функцій mag, phase, re, im є тривимірними масивами розміром “кількість виходів” × “кількість входів” × “довжина вектору частот”, що треба враховувати при застосуванні цих параметрів при побудові ЧХ.

Зазначимо, що при аналізі динамічних об‘єктів зазвичай ДЧХ, УЧХ, АЧХ і ФЧХ будують для замкнених систем, а діаграми Боде і Найквіста для розімкнених

Наприклад, для деякої відомої системи, що утворює у замкненому стані об‘єкт , а в розімкненому – об‘єкт , розглянуті вище ЧХ можна побудовати у такий спосіб:

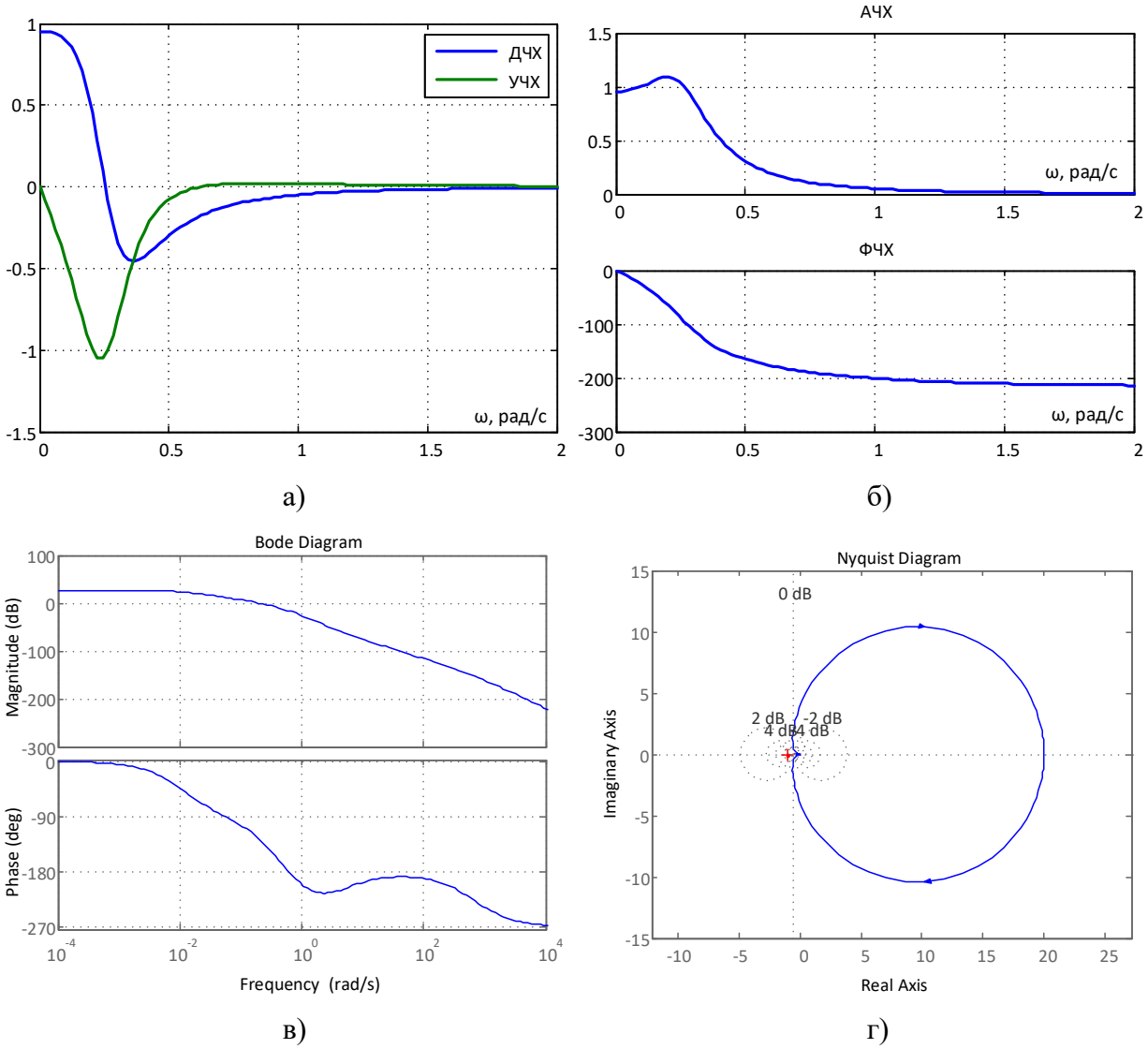
```
w = linspace (0, wk);
[U, V] = nyquist (sys, w);
figure (1)
plot (w, U(:),  w,V(:), 'LineWidth', 2), grid on, legend ('ДЧХ','УЧХ')

[Mag, Phi_deg] = bode (sys, w);
figure (2)
subplot (2,1,1), plot (w, Mag(:), 'LineWidth', 2), grid on, title ('АЧХ')
subplot (2,1,2), plot (w, Phi_deg(:),  'LineWidth', 2), grid on, title ('ФЧХ')
```

figure (3), bode (sysr, {log10(w0r) log10(wkr)}), grid on
figure (4), nyquist (sysr), grid on

У якості прикладу, на рис. 8.2 наведені частотні характеристики системи, ПФ якої в розімкненому і замкненому станах мають вигляд:

$$W_{\text{р}}(p) == \frac{5p+20}{0{,}5p^4+325p^3+347{,}05p^2+92{,}45p+1}; \qquad (8.13)$$

$$W_{\text{зі}}^{g}(p) = \frac{W_{\text{р}}(p)}{1+W_{\text{р}}(p)} = \frac{5p+20}{0{,}5p^4+325p^3+347{,}05p^2+97{,}45p+21}. \qquad (8.14)$$



а) б)

в) г)

Рис. 8.2. Частотні характеристики системи з передавальними функціями в розімкненому та замкненому станах (8.13) і (8.14) відповідно:

а) ДЧХ і УЧХ замкненої системи; б) АЧХ і ФЧХ замкненої системи; в) діаграми Боде розімкненої системи; г) діаграма Найквіста розімкненої системи

8.4. Зв'язок між АЧХ і ФЧХ та усталеною реакцією систем на синусоїдальні сигнали

Якщо подавати на вхід будь-якої лінійної динамічної системи синусоїдальний вхідний сигнал

$$u(t) = u_m \sin(\omega t + \varphi_u), \tag{8.13}$$

то в усталеному режимі вихідний сигнал також набуде синусоїдальної форми і матиме ту ж саму частоту, що і вхідний сигнал. Але амплітуда та фаза вихідної синусоїди в усталеному режимі

$$y(t) = y_m(\omega)\sin(\omega t + \varphi_y(\omega)), \tag{8.14}$$

відрізнятимуться від відповідних показників вхідної синусоїди і прийматимуть різні значення в залежності від частоти вхідного сигналу.

АЧХ являє собою залежність від частоти відношення амплітуди вихідного сигналу до амплітуди вхідного, або амплітуди вихідного сигналу при одиничній амплітуді вхідного ($u_m = 1$), тобто

$$A(\omega) = \frac{y_m}{u_m}(\omega) = y_m(\omega)\Big|_{u_m=1}, \tag{8.15}$$

а ФЧХ – різницю між фазами цих сигналів, або фазу вихідного сигналу при нульовій фазі вхідного ($\varphi_u = 0$***)***, тобто

$$\varphi(\omega) = \varphi_y(\omega) - \varphi_u = \varphi_y(\omega)\Big|_{\varphi_u=0}. \tag{8.16}$$

Розглянемо ще раз АЧХ та ФЧХ з рис. 8.2б. Позначимо на АЧХ точку резонансного максимуму $[\omega_р, A(\omega_р)]$, в якій АЧХ має максимальне значення і визначимо по ФЧХ фазу при цій частоті $\varphi(\omega_р)$, як це показано на рис. 8.3.

Виберемо довільно ще одну частоту $\omega = 0.5$, на якій амплітуда и фаза значно відрізняються від цих параметрів при $\omega = \omega_р$. З графіків рис. 8.3 отримуємо:

$$\omega_р = 0,2;\ \ M_р = A(\omega_р)/A(0) = 1,1/0,95 = 1,158;\ \ \varphi(\omega_р) = -65°;$$

$$A(0,5) = 0,3;\ \ \varphi(0,5) = -166°;\ \ A(0) = 0,95.$$

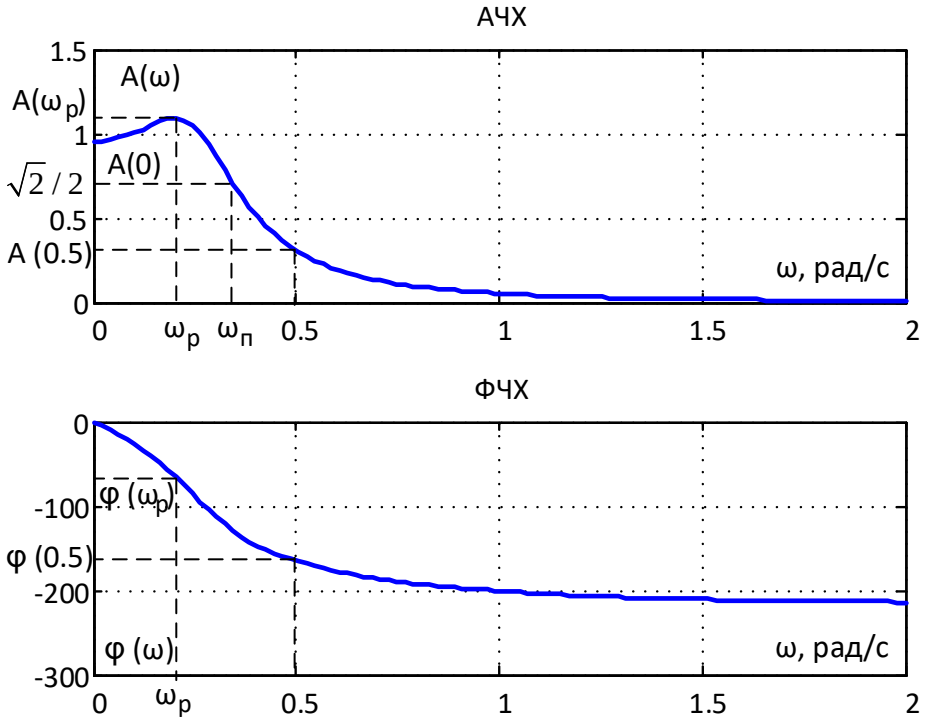


Рис. 8.3. АЧХ та ФЧХ для дослідження зв'язку між ними та усталеною реакцією системи на синусоїдальний вплив

Значення резонансного максимуму Mp та частоти, на якій він досягається wp, для замкненої системи sysz в середовищі *MATLAB* можна знайти як інфінітивну норму за допомогою функції norm:

```
[Mp, wp] = norm (sysz, Inf)
```

Величина резонансного максимуму $M_р$ ***впливає на перерегулювання перехідної функції, на коливальність перехідних процесів та на запас стійкості за фазою.*** Зокрема, перерегулювання можна оцінити шляхом розрахунку його за емпіричною формулою

$$\sigma \approx 1{,}3\frac{M_{\text{р}}-1}{M_{\text{р}}}\ . \qquad (8.19)$$

Значення перерегулювання, розраховане за формулою (5.38), для досліджуваної системи складає 17,7%.

Тепер подаємо на вхід моделі системи з ПФ (8.14) по черзі сигнали:

$$g_1(t)=\sin(0t+\pi/2)=1=\text{const}\,, \qquad (8.20)$$

$$g_2(t)=\sin(\omega_{\text{р}}t)=\sin(0{,}2t)\,, \qquad (8.21)$$

$$g_3(t)=\sin(0{,}5t)\,. \qquad (8.22)$$

В результаті симуляції отримаємо графіки рис. 8.4-8.6.

З рис. 8.4 бачимо, що усталене значення перехідної функції збігається з початковим значенням АЧХ, а перерегулювання складає 16,5%, тобто абсолютна похибка значення, розрахованого за формулою (8.19), складає 1,2%, що є цілком задовільно для приблизної оцінки цього параметру.

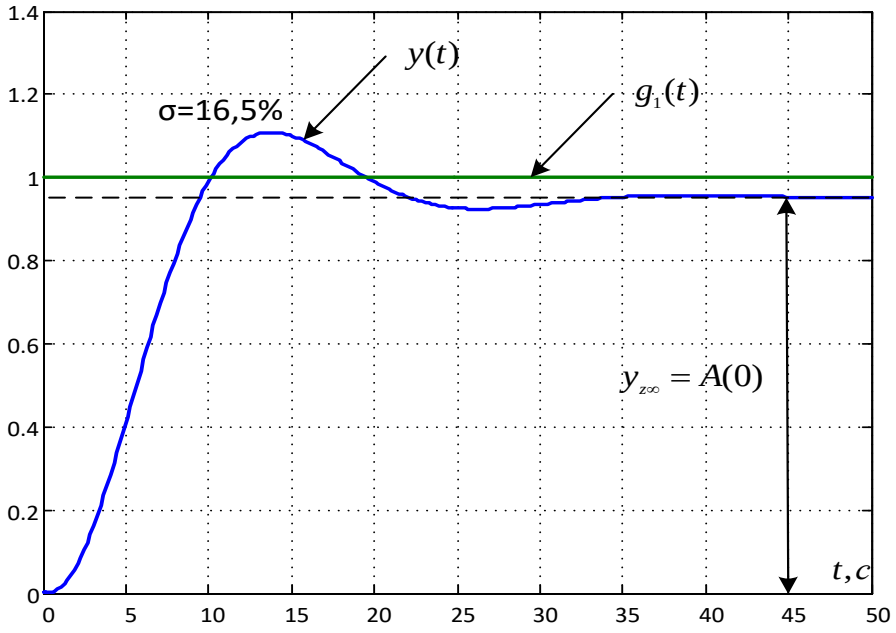


Рис. 8.4. Реакція досліджуваної системи на сигнал $g_1(t)=1(t)$

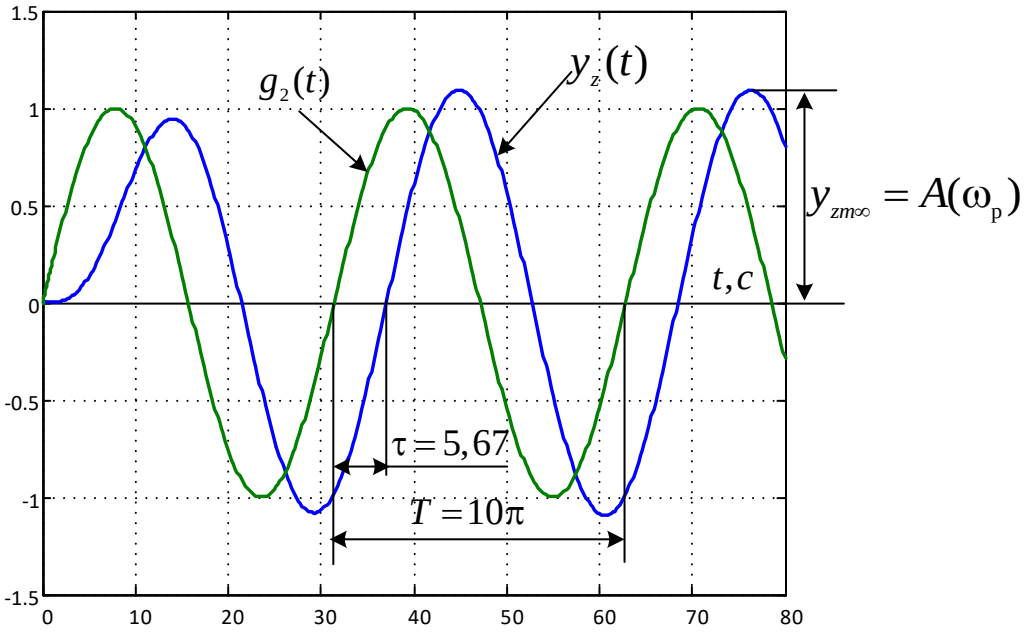


Рис. 8.5. Реакція досліджуваної системи на сигнал $g_2(t) = \sin(0,2t)$

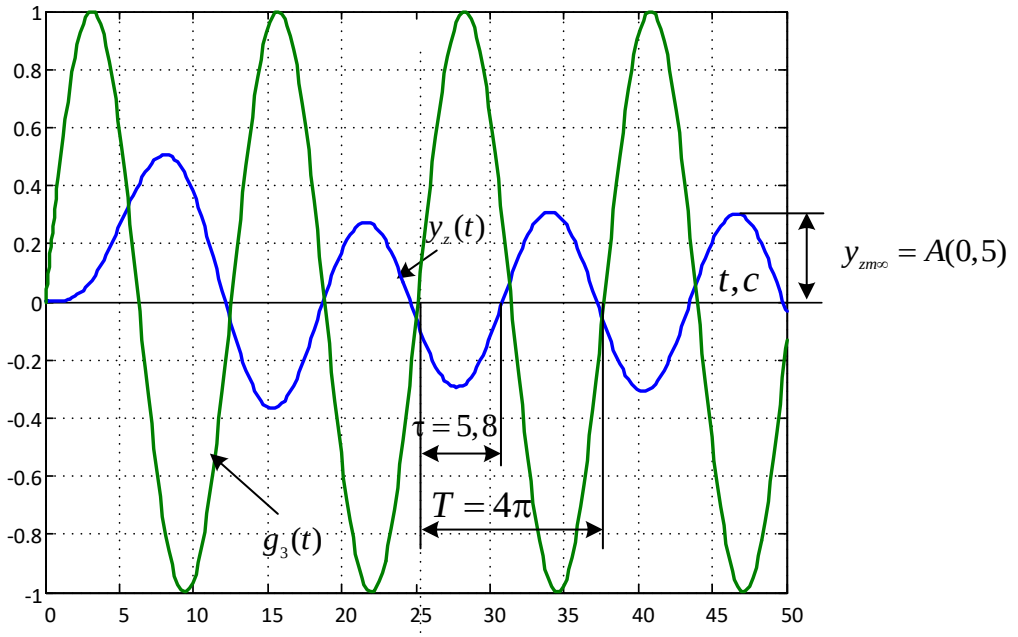


Рис. 8.6. Реакція досліджуваної системи на сигнал $g_3(t) = \sin(0,5t)$

З порівняння графіків рис.8.5 і 8.6 видно, що при однаковій амплітуді і нульовому зсуву фази вхідного сигналу, амплітуда і фаза вихідного сигналу в усталеному режимі змінюється в широкому діапазоні в залежності від частоти вхідного сигналу у відповідності з АЧХ та ФЧХ замкненої системи. Період усталених коливань визначається виразом

$$T = \frac{2\pi}{\omega} \ . \tag{8.23}$$

Знайдене з графіків запізнення вихідного сигналу від вхідного в усталеному режимі дає змогу розрахувати фазовий зсув вихідної синусоїди відносно вхідної з пропорції

$$\frac{\tau}{T} = \frac{\tau\omega}{2\pi} = \frac{\varphi(\omega)}{360} \ . \tag{8.24}$$

Розраховані у такий спосіб фази збігаються з відповідними значеннями ФЧХ.

Задачею будь-якої системи автоматичного керування є відпрацювання з мінімальним спотворенням вхідного низькочастотного сигналу та максимально можливе послаблення високочастотних сигналів, які є перешкодами. З такої позиції ***замкнена САК повинна мати властивості фільтра низьких частот***. ***Межу між високими і низькими частотами називають частотою пропускання*** $\omega_п$, а діапазон частот $[0, \omega_п]$ – смугою пропускання. Частота та смуга пропускання – це досить умовні поняття. Зазвичай ***смугою пропускання називають діапазон частот, у якому амплітуда вихідного сигналу є не меншою, ніж*** $\sqrt{2}/2 \approx 0,707$ ***від амплітуди вхідного сигналу***, як це показано на АЧХ рис. 5.12, тобто

$$A(\omega_п) = \sqrt{2}/2. \tag{8.25}$$

В *MATLAB* цю частоту w_prop можна знайти за допомогою функції bandwidth:

```
W_prop = bandwidth (sysz)
```

8.5. Контрольні запитання та завдання

1. Що таке частотні характеристики?
2. Які частотні характеристики ви знаєте?
3. Що таке частотна передавальна функція? Як розрахувати її значення в заданому діапазоні частот? Які числа отримаємо при розрахунку?

4. Які форми представлення комплексних чисел ви знаєте? Як пов‘язані між собою складові комплексних чисел?
5. Які *m*-функції ядра пакету *MATLAB* застосовуються для визначення складових комплексних чисел? В якій папці вони знаходяться?
6. Як визначити діапазон зміни частоти для побудови логарифмічних частотних характеристик?
7. В яких системах координат можна будувати амплітудно-фазові частотні характеристики?
8. Які можливості функцій bode і nyquist?
9. Що таке резонансний максимум? Яка *m*-функція його визначає?
10. Який параметр у часовій області визначає амплітуда АЧХ при нульовій частоті?
11. Який взаємозв‘язок між круговою частотою та періодом синусоїди?
12. Як визначити фазовий зсув вихідної синусоїди відносно вхідної із часу запізнення?
13. Що таке частота пропускання? Яка *m*-функція його визначає?

9. ОСНОВИ ПРОГРАМУВАННЯ В СЕРЕДОВИЩІ *MATLAB*

9.1. Характеристика лінгвістичних можливостей пакету *MATLAB*

Пакет *MATLAB* має у своєму складі розвинену алгоритмічну мову. З основними операторами цієї мови можна ознайомитися за допомогою команди

```
>>help lang
```

Наведемо скорочений список лінгвістичних конструкцій, що виводить ця команда з коментарями українською мовою:

Programming language constructs (конструкції алгоритмічної мови).

Control flow.

if - Conditionally execute statements (умовний оператор).

else - Execute statement if previous IF condition failed (початок гілки «false»).

elseif - Execute if previous IF failed and condition is true (початок альтернативної гілки «true»).

end - Terminate scope of control statements (кінець складеного оператора).

for - Repeat statements a specific number of times (заголовок циклу з відомим числом повторень).

while - Repeat statements an indefinite number of times (заголовок ітераційного циклу з передумовою.

break - Terminate execution of WHILE or FOR loop (переривання циклу).

switch - Switch among several cases based on expression (ключ до оператора вибору).

case - SWITCH statement case (оператор вибору).

otherwise - Default SWITCH statement case (початок гілки «інакше» оператора вибору).

try - Begin TRY block (оператор спроби).

catch - Begin CATCH block (початок гілка «інакше» оператора спроби).

error - Display message and abort function (переривання виконання програмного файлу з виведенням повідомлення про помилку).

Evaluation and execution.

eval - Execute string with MATLAB expression (виконувати рядок символів як вираз).

feval - Execute the specified function (виконувати рядок символів як функцію).

run - Run script (виконати script-файл).

Message display.

disp - Display array (виведення рядку символів або масиву).

Interactive input.

input - Prompt for user input (введення даних з клавіатури).

До перелічених вище операторів можна додати функції menu (меню), sprintf (записує форматовані дані в рядок символів), pause (пауза).

9.2. Характеристика основних операторів

Оператор консольного вводу input має формат:

VarName = input ('Prompt')

При виконанні цього оператора на екран виводиться запрошення Prompt у вигляді текстового повідомлення та перехід до режиму очікування користувачем введення значення змінної з клавіатури, що закінчується натисканням клавіші *Enter*, після чого введене значення присвоюється змінній з іменем VarName та зберігається у робочому просторі, наприклад,

```
» n = input ('Input  length of vector   n=')
Input  length of vector   n= <пауза до введення значення> 5 <Enter>
n =
    5
```

Оператор неформатованого виводу disp має формат

disp (VarName)

Він виводить значення змінної на екран без відображення її імені:

```
» k=1:10;
» disp(k)
   1    2    3    4    5    6    7    8    9   10
```

При виведенні одним оператором змішаних (символьних і числових) даних числові дані необхідно перетворити у рядок символів, що досягається використанням функцій int2str (integer) та num2str (number):

```
» disp (['Довжина вектору складає ', int2str(length(k)), ‘ елементів’])
Довжина вектору складає 10 елементів
» R =10.6; disp (['R=’, num2str(R), ‘ Ом - активний  опір кола'])
R=10.6 Ом - активний  опір кола
```

Функція затримки pause призупиняє виконання програми до натискання користувачем будь-якої клавіші. Оператор pause (n) виконує затримку виконання наступного оператора на n секунд.

Функцію menu, що має формат

VarName = menu ('Title’, 'Opt1', 'Opt2', …, ’ OptN’)

можна розглядати як різновидність оператору введення.

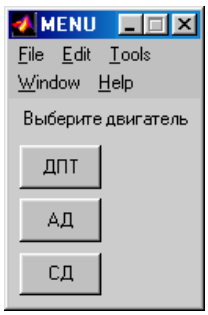


Рис. 9.1

Він генерує на екрані графічне вікно з кнопками, що надає користувачеві можливість вибору мишею однієї із заданих опцій. Порядковий номер обраної опції присвоюється змінній, ім‘я якої VarName зазначено у лівій частині оператора, наприклад, при виконанні оператора

```
» k = menu ('Оберіть двигун', 'ДПТ', 'АД', 'СД')
```

на екрані з‘являється вікно, зображене на рис. 9.1, і програма переходить у режим очікування вибору. При виборі опції ДПТ змінній k присвоюється значення 1, при виборі АД – значення 2, а при виборі СД – 3.

Складений оператор циклу з відомою кількістю повторень for…end має формат:

```
for VarName = Values % заголовок циклу
    % операторы тіла циклу:
    Stat1; Stat2; … StatN;
end % кінець циклу
```

Він організує виконання операторів Stat, що складають тіло циклу, послідовно для усіх значень Values змінної циклу VarName, після чого керування програмою передається оператору записаному після оператора кінця циклу end.

Значення змінної циклу можна задавати векторами будь-яких арифметичних констант (цілих і дійсних додатних та від'ємних чисел з додатним та від'ємним кроком, з рівномірним розподіленням впродовж дійсної або логарифмічної осі або розташованих у довільному порядку, комплексних чисел), а також рядками символів, наприклад,

```
for i = 1:10
for  x = -2.5:0.5:4
for k = 12:-2:0
for m = [-4:1, 3:4]
for  w = logspace (-2,3,6)
for  a = [3 18 -10 0, pi]
```

Параметр циклу може бути також матрицею. У цьому разі він по черзі приймає значення кожного стовбця матриці. Цикли можуть бути вкладеними. Тоді при кожному значенні змінної зовнішнього циклу змінна внутрішнього циклу перебирає послідовно усі свої значення.

Приклад 9.1. *Сформувати вектор X із 3-х елементів шляхом введення його з клавіатури.*

Складаємо програму

```
clc
for i = 1:3
   X(i) = input (['x(' int2str(i) ')=']);
end
X
```

В результаті її виконання отримуємо:

```
x(1)=5
x(2)=-8
x(3)=12
X =
    5   -8   12
```

Приклад 9.2. *Сформувати матрицю розміром 4×5 за формулою* $A_{ij}=2+ij$

Складаємо програму

```
A=zeros(4,5);
for i=1:4
   for j=1:5
      A(i,j)=2+i*j;
   end
end
A
```

В результаті її виконання отримуємо:

```
A =
    3    4    5    6    7
    4    6    8   10   12
    5    8   11   14   17
    6   10   14   18   22
```

Складений умовний оператор if…else…end, що використовують для програмування розгалужених алгоритмів, має формат:

```
if  LogExpression
      Statements1; % оператори гілки «так»:
else
      Statements2; % оператори гілки «ні»:
end % кінець оператору
```

Він обчислює логічний вираз LogExpression і аналізує його результат. Якщо результатом є 1 (істина, англ. *true*), то виконуються оператори гілки «так» Statements1, інакше (тобто, якщо результатом є 0 – хибність, англ. *false*) виконуються оператори гілки «ні» Statements2, наприклад,

```
if  x>0,  y = sqrt(x)
else y = x^2
end
```

При багатократному розгалуженні оператори if вкладаються один в інший. Щоб запобігти використання зайвого оператору end, замість конструкції else if в такому випадку використовують конструкцію elseif, наприклад,

```
if  x>0,  sg = 1
   elseif x<0, sg = -1
   else sg=0
```

```
end
```

У логічних виразах зазвичай застосовують операції відношення (порівняння) та/або логічні операції. Результатами таких операцій є істина (*true*) та хибність (*false*). У *MATLAB* їм відповідають логічні константи 1 та 0.

До операцій відношення належать:

> – більше; >= – більше або рівно; < – менше; <= – менше або рівно; = = – рівно; ~= – не рівно.

При порівнянні двох масивів однакового розміру результатом є масив того ж розміру, складений із нулів та одиниць, як результатів поелементних операцій відношення. Якщо ж в одним із операндів є скаляр, то кожний елемент масиву по черзі порівнюється зі скаляром:

```
» 2>1
ans =
    1
» 1>=3
ans =
    0
» A = [1 8 3; 6 2 5];  B = [4 3 1; 2 6 8];
» A>3
ans =
    0    1    0
    1    0    1
» A<B
ans =
    1    0    0
    0    1    1
```

Логічні операції виконуються над логічними даними, тобто операндами логічних операцій в мажуть бути тільки 0 (хибність) та 1 (істина) або їх сполучення.

До основних логічних операцій в MATLAB належать:

& – "І" (логічне множення, або кон‘юнкція), в алгебрі логіки найчастіше позначається як ∧;

| – "АБО" (логічне додавання, або диз‘юнкція), в алгебрі логіки найчастіше позначається як ∨;

xor – «виключне АБО»;

~ – «НЕ» (логічне заперечення, або інверсія), позначається символом ¬.

Операції | та & є двоаргументними, а ~ – одноаргументна.

Результати наведених операцій при різних сполученнях аргументів наведені у табл. 9.1.

Таблиця 9.1

x	y	x&y	x|y	x xor y	~x
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Логічні операції в пакеті *MATLAB* доповнюються такими логічними функціями:

any(x) – повертає 1, якщо деякі з елементів масиву x не нулі, і 0 у противному випадку;

all(x) – повертає 1, якщо усі елементи масиву x не нулі, і 0 у противному випадку;

find(x) – повертає індекси ненульових елементів масиву x.

Наприклад, при виконанні програми

```
x = [1 5 2 -7 4 -9]
if any(x<0)
   k = find(x<0),  x(k) = x(k).^2,  y = sqrt(x)
end
z = all(y>1)
if all(x~=0) v = log10(x), end
```

отримуємо

```
x =
    1    5    2   -7    4   -9
k =
    4    6
x =
    1    5    2   49    4   81
```

```
y =
   1.0000    2.2361    1.4142    7.0000    2.0000    9.0000
z =
    0
v =
         0    0.6990    0.3010    1.6902    0.6021    1.9085
```

Складений оператор вибору switch…case…otherwise…end, що застосовують при програмуванні багатократно розгалужених процесів, має у найпростішому випадку такий формат:

```
switch  VarName
        case Value1, Statements1
        case Value2, Statements2
        …
        case ValueN, StatementsN
        otherwise  StatementsAlternative
end
```

Для такого оператору гілка Statements1 виконується, якщо VarName = Value1, гілка Statements2 – якщо VarName = Value2, гілка StatementsN – якщо VarName = ValueN та гілка StatementsAlternative – коли ключова змінна отримує будь-яке інше значення. Конструкція otherwise не є обов‘язковою.

У більш складних випадках ключова змінна і її значення можуть задаватися виразами (Expression).

Складений оператор спроби try…catch…end, що застосовується при налагодженні програми має такий формат:

```
try  StatementeBase
catch  StatementAlternative
end
```

Оператор-альтернатива StatementAlternative виконується у тому випадку, коли при виконанні оператора-спроби фіксуються помилки. Текст повідомлення про помилку можна вивести за допомогою команди lasterr, наприклад,

```
» A=[1 2; 3 4]; B=[2 4];
```

```
» try, X=A\B, catch, lasterr, X=A\B', end
ans =
Error using ==> \
Matrix dimensions must agree.
X =
    0
    1
```

Складений оператор ітераційного циклу з передумовою while…end має формат:

```
while  LogExpression   % заголовок циклу
    % операторы тіла циклу:
     Stat1; State2; … StateN;
end % кінець циклу
```

Він організує виконання операторів stat, що складають тіло циклу, до тих пір, поки значення логічного виразу LogExpression є істиною (1). Вихід із циклу відбувається, коли значення цього виразу стає хибним (0). Оператори тіла циклу повинні змінювати значення хоча б однієї змінної, що входять до логічного виразу в заголовку циклу, або отримувати у своєму складі умовний оператор з оператором break у гілці «так».

Приклад 9.3. *Знайти мінімальне ціле число n, що задовольняє умові* $n!>10^{12}$*:*

```
M = 1e12; n = 1; F = 1;
while F<= M,  n = n+1;  F = F*n;  end,  F, n
```

В результаті отримуємо:

```
F =
  1.3077e+12
n =
   15
```

Приклад 9.4. *Скласти програму, яка в залежності від вибору опції, виконаного мишею за допомогою сформованого меню, виконує одну з 3-х програм, а при виборі опції* Exit *завершує виконання програми.*

Розв'язання цієї задачі має вигляд

```
while 1
```

```
    k = menu (‘Оберіть файл для виконання’, ‘File 1’, ‘File 2’, ‘File 3’, ‘Exit’)
    switch  k
        case 1, FileName1
        case 2, FileName2
        case 3, FileName3
        otherwise  break
    end
end
```

Оператор обчислення рядку символів як виразу eval (від англ. *evaluate*) має формат

eval (String)

Наприклад,

```
» eval ('sqrt(2)')
ans =
   1.4142
```

Оператор виконання функції, ім‘я якої задано рядком символів, feval має формат

feval (String, Arg1, Arg2, …, ArgN)

Наприклад,

```
» feval('sin',pi/2)
ans =
    1
```

Приклади, які доведуть доцільність використання операторів eval та feval, які за своєю конструкцією більш схожі на функції, будуть наведені пізніше.

9.3. Основні правила створення і виконання *m*-функцій

M-функції застосовуються для програмування типових алгоритмів, які показують, як із формальних вхідних параметрів утворити вихідні, або як використати вхідні параметри для побудови графіків тощо. Тобто *m*-функція уявляє собою нібито розгорнуту формулу, у якій формальні вхідні параметри є тільки позначеннями, тобто умовними іменами аргументів.

Щоб отримати конкретний результат за допомогою *m*-функції треба виконати заміну формальних вхідних аргументів фактичними, які при

зверненні до функції повинні мати конкретні значення, і виконати усі операції з цими параметрами.

Для створення *m*-функції треба клацнути мишею по віртуальній кнопці *New* у командному вікні MATLAB та обрати опцію *Function*. В результаті на екрані з'явиться вікно текстового редактора зі сформованим у ньому форматом функції, зображене на рис. 9.2.

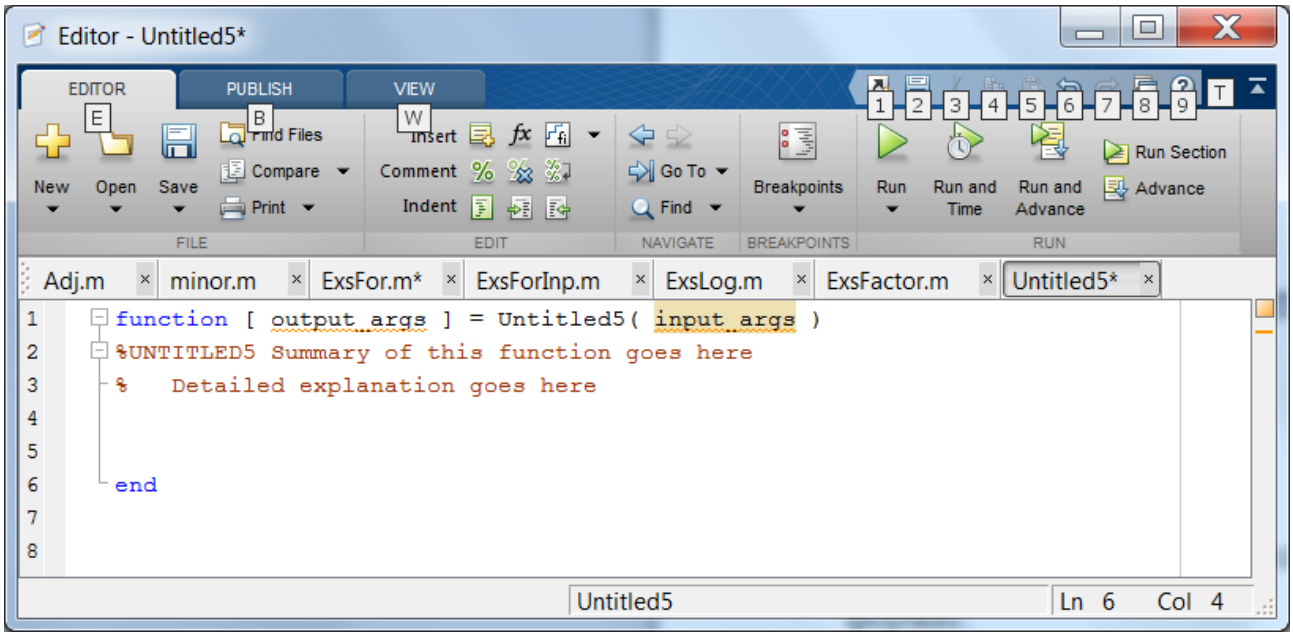


Рис. 9.2. Вікно текстового редактору для створення *m*-функції

Як бачимо, *m*-функція має формат

```
% Заголовок функції:
function [ List of ResultsNames ] = FunName (List of InputArgsNames )
% Необов'язкові коментарі:
% Contents String (global information of this function)
% Detailed information of this function
…

% оператори тіла функції
 Stat1; Stat2; …; StatN;
end  % кінець функції
```

Імена змінних у списку результатів виконання функції List of ResultsNames (в *MATLAB* їх називають вихідними аргументами – *output*

arguments) та у списку аргументів функції List of InputArgsNames (в *MATLAB* їх називають вхідними аргументами – *input arguments*) відокремлюються одне від одного комами «,».

Оператори та коментарі в *m*-функціях вводяться за тими ж правилами, які використовуються при створенні *script*-файлів. Зберігати *m*-функцію треба обов‘язково у файлі, ім‘я якого співпадає з іменем функції FunName. Такий файл автоматично отримує розширення «.*m*».

На відміну від *script*-файлу *m*-функцію не можна виконати командою run. Для виконання функції до неї необхідно звернутися з будь-якого програмного файлу або з командного рядка по імені та списку фактичних аргументів у круглих дужках.

Якщо функція не потребує наявності вхідних або вихідних параметрів, вони в заголовку функції позначаються у вигляді пустого вектору [ ], або зовсім опускаються.

Функції в MATLAB можуть використовуватися з різною кількістю вхідних і вихідних параметрів, а також для різнотипних параметрів. Наприклад, розглянемо різні варіанти використання базової функції математичного аналізу min:

```
>>a = -45; b = 12;
>>c = min (a,b)
c =
  -45
>>X = [2 7 -5 3];  Y= [15 6 22 -1];
>>Xmin = min(X)
Xmin =
   -5
>> [Xmin,Imin] = min(X)
Xmin =
   -5
Imin =
    3
>>Z = min (X,Y)
Z =
    2    6   -5   -1
>>A = [1 3 0; 4 2 8], B = [5 1 2; -3 7 4]
```

```
A =
    1    3    0
    4    2    8
B =
    5    1    2
   -3    7    4
>>Amin = min(A)
Amin =
    1    2    0
>> [Amin,imin] = min(A)
Amin =
    1    2    0
imin =
    1    2    1
>>A_min = min(min(A))
A_min =
    0
>>C=min(A,B)
C =
    1    1    0
   -3    2    4
```

Щоб створити такі функції, в них застосовують стандартні *m*-функції nargin та nargout, значеннями яких є кількість вхідних та вихідних фактичних аргументів відповідно.

Для того, щоб користувач правильно формував фактичні вхідні аргументи, в тілі функції повинні бути присутніми оператори, що аналізують типи та розміри цих параметрів, та при наявності помилок припиняють виконання програми з виведенням повідомлення про помилку.

Приклад 9.5. *Скласти функцію* maxi(x)*, яка в заданому векторі x визначає максимальний елемент та його позицію.*

```
function [xm,im] = maxi(x)
    [m,n]=size(x);
    if (m~=1)&(n~=1)  % Якщо x - матриця
      error('Помилка в розмірності аргументу');
    end
    k = length(x);
    if k==1, xm=x; im=1; break; end
    xm=x(1);
    for i=2:k
      if x(i)>xm  xm = x(i); end
```

```
    end
    if nargout==2     % Два вихідних параметри
      im = find(x==xm);
    end
end
```

9.4. Контрольні питання та завдання

1. В якій папці можна подивитися оператори алгоритмічної мови *MATLAB*?
2. Як здійснити діалогове введення даних в *MATLAB*? Наведіть приклади використання оператору input.
3. Наведіть приклади заголовків циклів з відомою кількістю повторювань.
4. Як програмуються в *MATLAB* розгалужені процеси? Наведіть приклади використання умовного оператору та оператору вибору.
5. Які операції відношення, логічні операції і функції Ви знаєте?
6. Як програмуються в *MATLAB* ітераційні цикли?
7. Наведіть приклад застосування функції menu.
8. Чим відрізняються файли-функції від файлів-сценаріїв?
9. Як утворюються заголовки функцій?
10. Під якими іменами записують функції у файл?
11. Як створюються функції з варійованою кількістю вхідних та вихідних параметрів?
12. Як звертаються до функцій при програмуванні? Як виконати функцію?
13. Скласти функцію користувача згідно з завданням, наведеним у табл. 9.2. Застосувати її для розв‘язання декількох довільних тестових задач. Перевірити правильність отриманого результату.

Таблиця 9.2

№	Задача *m*-функції	Перевірка
1	Розрахувати матричний добуток $\mathbf{C}=\mathbf{AB}$ матриці **A** розміром $m\times k$ та матиці **B** розміром $k\times n$ за формулою (2.4).	C=A*B
2	Для квадратної матриці **A** розрахувати $\mathbf{B}=\mathbf{A}^k$ з використанням операції * (див. 2.8).	B = A^k

3	Для квадратної матриці **A** розміром $n\times n$ та вектору-стовпчику **B** розміром $n\times 1$ розрахувати матрицю керованості $\mathbf{Q}_c=[\mathbf{B}\ \mathbf{AB}\ \mathbf{A}^2\mathbf{B}...\mathbf{A}^{n-1}\mathbf{B}]$ з використанням операції *.	Qc = ctrb (A,B)
4	Для квадратної матриці **A** розміром $n\times n$ та вектору-рядку **C** розміром $1\times n$ розрахувати матрицю керованості $\mathbf{Q}_o=[\mathbf{C}\ \mathbf{CA}\ \mathbf{CA}^2\ ...\mathbf{CA}^{n-1}]^T$ з використанням операції *.	Qo = obsv (A,B)
5	Розв‘язати систему лінійних алгебраїчних рівнянь методом Крамера з використанням функції	X = A\B
6	Розрахувати обернену матрицю шляхом розв‘язання систем лінійних рівнянь для кожних із стовпчиків матриць $\mathbf{A}^{-1}$ та одиничної діагональної матриці **I** матричного рівняння $\mathbf{A}^{-1}\mathbf{A}=\mathbf{I}$.	Ai = inv (A)
7	Розрахувати середнє квадратичне значення вектору за формулою (2.20).	D = var (x)
8	Розрахувати дисперсію вектору за формулою (2.21).	
9	Розрахувати середнє геометричне значення вектору за формулою (2.22).	
10	Розрахувати k-ті прямі різниці вектору **V**.	dVk = diff (V,k)
11	Розрахувати суму квадратичних відхилень двох векторів однакової довжини $F=\sum_{i=1}^{n}(y_i-x_i)^2$	
12	Розрахувати визначений інтеграл табличної функції методом трапецій за формулою $Z=\frac{1}{2}\sum_{i=1}^{n-1}(y_{i+1}+y_i)(x_{i+1}-x_i)$	Z = trapz (x,y)
13	Скласти функцію, що обчислює значення степеневого полінома за його коефіцієнтами і значеннями незалежної змінної за формулою (4.2) (схема Горнера)	Y = polyval (A, X)
14	Скласти власну функцію, аналогічну одноаргументній функції polyder	B = polyder(A)
15	Скласти функцію, що розраховує коефіцієнти чисельника і знаменника замкненого контуру за відповідними коефіцієнтами ланки в прямому каналі і в каналі зворотного зв‘язку	sz=feedback(sp,sz)

При виконанні останнього пункту дотримуйтесь таких рекомендацій:

- не давайте своїм функціям імена, що збігаються з зарезервованими іменами

пакету *MATLAB*;

- організуйте у функціях перевірку розмірів вхідних параметрів; у разі невідповідності їх умовам задачі організуйте виведення в командне вікно повідомлення про помилку за допомогою функції *error*.
- для розв‘язання тестових задач складіть *script-file*.

10. ОПЕРАЦІЇ МАТРИЧНОГО АНАЛІЗУ ТА ЛІНІЙНОЇ АЛГЕБРИ

До найчастіше застосованих операцій матричного аналізу та лінійної алгебри відносяться:

- розрахунок визначників;
- визначення мінорів та алгебраїчних доповнень матриці;
- визначення союзних та приєднаних матриць;
- обертання матриць;
- розв‘язання систем лінійних рівнянь.

Усі перелічені вище операції виконуються для квадратних матриць, тобто

$$\underset{n\times n}{\mathbf{A}}=\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \qquad (10.1)$$

10.1. Методи розрахунку визначника матриці

Визначник матриці або ***детермінант матриці*** – це одна із найважливіших характеристик квадратної матриці, що застосовується при розв‘язанні багатьох задач лінійної алгебри. Він розраховується з усіх перестановок елементів матриці розміром $n\times n$ як сума добутків із n елементів. Кожний доданок складається з добутку елементів у такий спосіб, щоб у ньому були присутні тільки по одному елементу з кожного рядка і з кожного стовпця. Він може позначатися як Δ, $\Delta(\mathbf{A})$, $\det(\mathbf{A})$, $|\mathbf{A}|$.

Визначники матриць другого та третього порядків зазвичай розраховують на основі їх визначення за відомими схемами:

$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}=a_{11}a_{22}-a_{21}a_{12},$$

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{21}a_{32}a_{13} - a_{31}a_{22}a_{13} - a_{21}a_{12}a_{33} - a_{32}a_{23}a_{11}.$$

Для матриць більш високого порядку ці схеми стають занадто громізькими та неекономічними з точки зору кількості операцій множення та алгебраїчного сумування. В цьому випадку для розрахунку визначників зазвичай використовують ***прямий хід методу Гауса***.

Прямий хід методу Гауса полягає в еквівалентному перетворенні матриць коефіцієнтів до трикутного вигляду шляхом почергового обнулення елементів матриці, розташованих нижче головної діагоналі, починаючи з першого стовпчика і закінчуючи передостаннім.

Еквівалентним називають таке перетворення матриці, яке не приводить до зміни її визначника.

Обнуління k-го елементу (k = 1, 2,..., n-1) i-го рядка (i = k+1, k+2,...,n) виконують шляхом заміни усіх коефіцієнтів i-го рівняння різницею між попередніми коефіцієнтами цього рівняння та відповідними коефіцієнтами k-го рівняння, помноженими на масштабний множник

$$p = \frac{a_{ik}}{a_{kk}}. \qquad (10.1)$$

У результаті коефіцієнти i-го рівняння після k-го перетворення приймають наступні значення:

$$a_{ik}^{(k)} = 0, \qquad a_{ij}^{(k)} = a_{ij}^{(k-1)} - p a_{kj}^{(k-1)}, \ (j = k+1, k+2, ..., n). \qquad (10.2)$$

При послідовному обнулінні коефіцієнтів мінімальної похибки округлення можливо досягти перестановкою рівнянь у такий спосіб, щоб модулі коефіцієнтів a_{kk} були максимально можливими. Цей етап метода Гауса зветься вибором головного елемента. Відповідно до вищеподаного, схема алгоритму прямого ходу може мати вигляд, наданий на рис. 10.1.

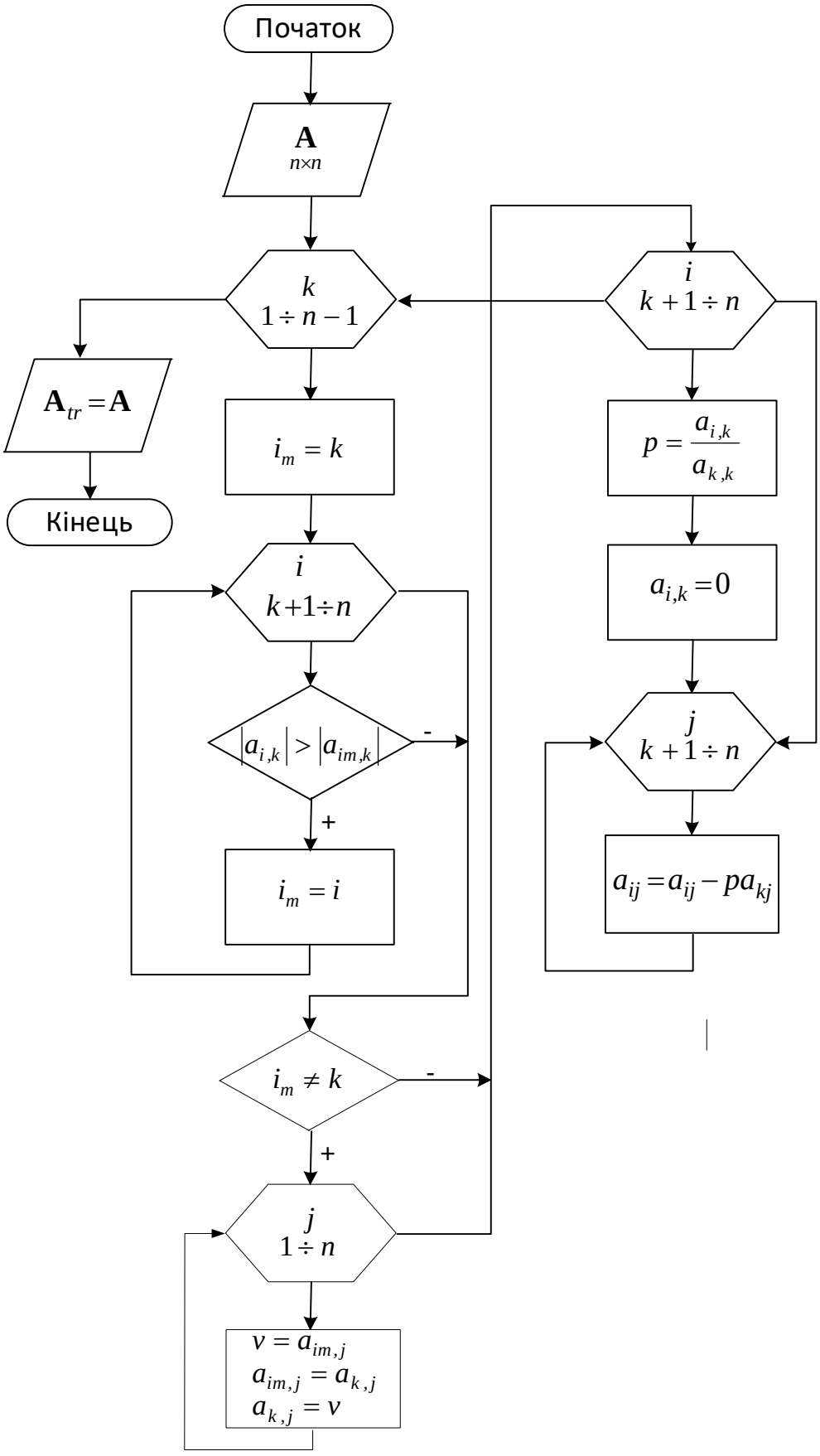


Рис. 10.1. Прямий хід метода Гауса

У результаті прямого ходу вихідна квадратна матриця (10.1) без врахування перестановки рядків отримує трикутний вигляд:

$$\underset{n\times n}{\mathbf{A}_{tr}}=\begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \dots & a_{2n}^{(1)} \\ 0 & 0 & a_{33}^{(2)} & \dots & a_{3n}^{(2)} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & a_{nn}^{(n-1)} \end{pmatrix} \tag{10.3}$$

Визначник перетвореної матриці дорівнює добутку діагональних елементів:

$$\left|\mathbf{A}_{tr}\right|=\left|\mathbf{A}\right|=a_{11}\cdot a_{22}^{(1)}\cdot a_{33}^{(2)}\cdot\ldots\cdot a_{nn}^{(n-1)}. \tag{10.4}$$

У *MATLAB операцію розрахунку визначника матриці виконує функція* det (A).

10.2. Методи розв‘язання систем лінійних рівнянь

Система *n* лінійних рівнянь з *n* невідомими має вигляд:

$$\begin{cases} a_{11}\cdot x_1+a_{12}\cdot x_2+\ldots+a_{1n}\cdot x_n=b_1, \\ a_{21}\cdot x_1+a_{22}\cdot x_2+\ldots+a_{2n}\cdot x_n=b_{2,} \\ \quad\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}\cdot x_1+a_{n2}\cdot x_2+\ldots+a_{nn}\cdot x_n=b_{n.} \end{cases} \tag{10.5}$$

Її можна записати і у матричній формі:

$$\mathbf{AX}=\mathbf{B}, \tag{10.6}$$

де $\mathbf{A}=\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}$ – квадратна матриця коефіцієнтів;

$\mathbf{B}=\begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{pmatrix}$ – вектор вільних членів; $\mathbf{X}=\begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix}$ – шуканий вектор коренів.

Способи рішення систем лінійних рівнянь поділяються на дві групи:

- точні методи (метод обернення матриці коефіцієнтів, правило Крамера, метод Гауса та інші);
- ітераційні методи (Ньютона, Зейделя, простих ітерацій та інші).

Якщо матриця **A** неособлива, тобто її визначник не дорівнює нулю, то система має єдине рішення:

$$\mathbf{X}=\mathbf{A}^{-1}\mathbf{B}, \tag{10.7}$$

де $\mathbf{A}^{-1}$ – матриця, обернена до матриці **A**.

Обчислення коренів за формулою (10.7) називається ***методом обернення матриці коефіцієнтів***.

Згідно з ***правилом Крамера*** корні обчислюються за формулами:

$$x_1=\frac{\Delta_1}{\Delta}, x_2=\frac{\Delta_2}{\Delta},\ldots,x_n=\frac{\Delta_n}{\Delta}, \tag{10.8}$$

де Δ – визначник матриці **A**;

Δ_i – визначники матриць, отримані з матриці **A** шляхом заміни її *i*-го стовпця вектором вільних членів **B**.

Обидва з перелічених вище методів використовують на практиці тільки при рішенні "вручну" систем рівнянь невисокого порядку. При $n>3$ ці методи стають дуже трудомісткими та не економічними.

Найбільш поширеним методом розв'язання систем лінійних рівнянь є ***метод Гауса***, який в даному випадку розбивається на два етапи: ***прямий хід та зворотний хід***.

Алгоритм прямого ходу відрізняється від зображеного на рис. 10.1 тільки тим, що він виконується не над квадратною матрицею, а над так званою розширеною матрицею розміром $n\times(n+1)$, отриманої шляхом додавання до матриці коефіцієнтів стовпчика вільних членів:

$$\mathbf{A}_{\text{р}}=\begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} & b_1 \\ a_{21} & a_{22} & \ldots & a_{2n} & b_2 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ a_{n1} & a_{n2} & \ldots & a_{nn} & b_n \end{pmatrix}. \tag{10.9}$$

У результаті прямого ходу система рівнянь (10.5) отримує вигляд

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n = b_1, \\ a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 . + \ldots + a_{2n}^{(1)}x_n = b_2^{(1)}, \\ a_{33}^{(2)}x_3 . + \ldots + a_{2n}^{(2)}x_n = b_3^{(2)}, \\ \ldots\ldots\ldots\ldots \\ a_{nn}^{(n-1)}x_n = b_n^{(n-1)}. \end{cases} \qquad (10.10)$$

Зворотний хід методу Гауса полягає у почерговому визначенні коренів з *n*-го по 1-ий з перетвореної системи рівнянь:

$$x_n = \frac{b_n}{a_{nn}}; \qquad x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{ij}x_j}{a_{ii}}, \ (i = n-1, n-2, \ldots., 1). \qquad (10.11)$$

Обчислення коренів системи лінійних рівнянь у пакеті MATLAB можна виконати за допомогою ***операції лівобічного ділення*** X=A\B або застосуванням функції або множенням оберненої матриці на вектор вільних членів: X=inv(A)*B.

Перші два із запропонованих способів є кращими, тобто більш швидкодіючими і у багатьох випадках більш точними, оскільки вони застосовують найбільш ефективні методи розв‘язання системи, здійснюючи вибір методу на основі аналізу матриці коефіцієнтів.

Слід відзначити, що точність розв‘язку не буде абсолютною, не зважаючи на еквівалентність перетворень та відсутність ітераційних обчислень. Вона спотворюється похибками округлення, які виникають при виконанні арифметичних операцій внаслідок обмеженої довжини комірок пам‘яті. Для перевірки точності розв‘язку доцільно обчислити і вивести на екран значення ***вектору нев’язок***

$$\mathbf{F} = \mathbf{AX} - \mathbf{B} \qquad (10.12)$$

У разі вірного розв’язання системи нев’язки (неузгодженості) повинні бути близькими до нуля. До того, як розв‘язувати систему рівнянь, слід знайти визначних матриці коефіцієнтів. Якщо він дорівнює 0, система не має рішення.

Наведена операція лівостороннього ділення може розв‘язати приблизно (методом найменших квадратів) навіть недовизначену та перевизначену системи лінійних рівнянь, тобто системи, в яких кількість рівнянь є меншою або більшою ніж кількість невідомих змінних, наприклад,

```
>> A=[1 2 3; 7 5 4]
A =
    1    2    3
    7    5    4
>> B=[5;1]
B =
    5
    1
>> x=A\B
x =
  -1.0000
        0
   2.0000
>>F=A*x-B
ans =
  1.0e-15 *
        0
         0.8882
```

Прикладом використання систем лінійних рівнянь в теорії автоматичного керування є розрахунок усталених режимів. Як відомо, математичний опис лінійних динамічних систем у просторі станів має вигляд:

$$\begin{cases} \mathbf{X}' = \mathbf{AX} + \mathbf{BU}, \\ \mathbf{Y} = \mathbf{CX} + \mathbf{DU}, \end{cases} \tag{10.13}$$

де $\mathbf{U}, \mathbf{Y}, \mathbf{X}$ – вектори вхідних і вихідних сигналів та вектор стану і вектор похідних від змінних стану; $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ – матриці стану, входу, виходу та прямогозв‘язку входу з виходом. В усталеному режимі $\mathbf{X}' = \mathbf{0}$. Тому для розрахунку усталених значень змінних стану необхідно розв‘язати систему лінійних рівнянь

$$\mathbf{AX}_{уст} = \mathbf{BU}, \tag{10.14}$$

а потім підставити знайдені значення

$$\mathbf{X}_{уст} = \mathbf{A}^{-1}\mathbf{BU} \tag{10.15}$$

у друге рівняння системи (10.13):

$$\mathbf{Y}_{уст}=\mathbf{C}\mathbf{X}_{уст}+\mathbf{D}\mathbf{U}. \qquad (10.16)$$

10.3. Методи розрахунку приєднаної матриці

Визначення приєднаної матриці $\mathrm{Adj}(\mathbf{A})$ (***adjacency matrix***), яка використовується для розрахунку оберненої матриці, пов‘язанно з поняттями мінор, алгебраїчне доповнення та союзна матриця.

Мінором M_{ij} елементу a_{ij} визначника квадратної матриці n-го порядку називають визначник матриці $(n-1)$-го порядку, отриманої із вихідної матриці викреслюванням з неї елементів i-го рядка та j-го стовпчика.

У якості прикладу наведемо програму, що утворює квадратну матрицю розміром 5×5 із випадкових чисел та обчислює для неї мінор M_{32}:

```
a = rand(5)
ap32 = a;
ap32(3,:) = [ ];  ap32 (:,2) = [ ]
Ma32=det(ap32)
```

При виконанні цієї програми отримаємо

```
a =
   0.7577   0.7060   0.8235   0.4387   0.4898
   0.7431   0.0318   0.6948   0.3816   0.4456
   0.3922   0.2769   0.3171   0.7655   0.6463
   0.6555   0.0462   0.9502   0.7952   0.7094
   0.1712   0.0971   0.0344   0.1869   0.7547
ap32 =
   0.7577   0.8235   0.4387   0.4898
   0.7431   0.6948   0.3816   0.4456
   0.6555   0.9502   0.7952   0.7094
   0.1712   0.0344   0.1869   0.7547
Ma32=
  -0.0157
```

Алгебраїчне доповнення A_{ij} може відрізняється від мінору M_{ij} тільки знаком:

$$A_{ij}=(-1)^{i+j}M_{ij}. \qquad (10.17)$$

Матриця, складена із алгебраїчних доповнень всіх елементів вихідної матриці, називається ***союзною матрицею***. Вона позначається як $\tilde{\mathbf{A}}$. Матриця, транспонована до союзної, називається ***приєднаною матрицею***:

$$\mathrm{Adj}(\mathbf{A}) = \tilde{\mathbf{A}}^T. \qquad (10.18)$$

Вона використовується для визначення оберненої матриці:

$$\mathbf{A}^{-1} = \frac{\mathrm{Adj}(\mathbf{A})}{\det(\mathbf{A})}. \qquad (10.19)$$

У пакеті *MATLAB* не існує функції, що розраховує приєднану матрицю, але можна створити власну функцію, наприклад,

```
function A_adj=Adj(A)
   [m,n]=size(A);
   A_adj=zeros(m,n);
   for i=1:m
      Ap=A;
      Ap(i,:)=[];
      for j=1:n
         B=Ap;
         Ap(:,j)=[];
         MA(i,j)= det(Ap);
         A_adj(j,i)=MA(i,j)*(-1)^(i+j);
         Ap=B;
      end
   end
```

Використовуючи цю функцію для матриці із випадкових чисел, отримаємо

```
>>A=randn(5), A_adj=Adj(A)
A =
   -0.4093    0.0780   -1.3853    1.9085    0.6901
   -0.1609    1.3244    0.3105    0.1222    0.5558
    0.4093   -0.2132   -0.2495    1.0470   -1.1203
   -0.9526   -0.1345    0.5037   -0.2269   -1.5327
    0.3173   -1.1714   -0.8927   -0.1625   -1.0979
A_adj =
    1.9550   -0.9161   -2.8814    3.1894   -0.7474
    0.3104   -6.2334   -0.2569   -0.0613   -2.6127
    1.2930    4.6154   -2.0778   -0.3226    5.7197
   -1.0943    2.5952   -2.6672    0.0006    3.3466
   -0.6554    2.2490    1.5254    1.2495    2.0782
```

Перевірити правильність отриманого результату можна згідно з формулою (10.15) оператором A_adj = inv(A)*det(A).

10.4. Методи розрахунку оберненої матриці

Оберненою матрицею $\mathbf{X}=\mathbf{A}^{-1}$ у відношенні до вихідної квадратної матриці **A** називається така квадратна матриця яка, будучи помноженою на вихідну, дає одиничну діагональну матрицю **E** того ж розміру:

$$\mathbf{A}\mathbf{A}^{-1}=\mathbf{A}^{-1}\mathbf{A}=\mathbf{E}, \tag{10.20}$$

або у розгорнутій формі:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \cdot \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{12} & x_{22} & \dots & x_{2n} \\ \dots & \dots & \dots & \dots \\ x_{n1} & x_{n2} & \dots & x_{nn} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{pmatrix} \tag{10.21}$$

Як видно з (10.5), елементи *k*-го стовпця оберненої матриці **X** можна визначити розв'язуючи систему *n* лінійних рівнянь з *n* невідомими.

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \cdot \begin{pmatrix} x_{1,k} \\ x_{2,k} \\ \dots \\ x_{n,k} \end{pmatrix} = \begin{pmatrix} e_{1,k} \\ e_{2,k} \\ \dots \\ e_{n,k} \end{pmatrix}, \tag{10.22}$$

де $e_{i,k} = \begin{cases} 0, & при \ \ i \neq k, \\ 1, & при \ \ i = k, \end{cases}$ $k = 1, 2, ..., n.$

Отже, для визначення усіх елементів оберненої матриці необхідно розв'язати *n* систем рівнянь. Цей підхід часто використовують при машинних розрахунках. Розв'язувати системи рівнянь можна будь-яким з відомих методів, наприклад, методом Гауса. Обернення часто виконують також використовуючи формулу (10.19).

У *MATLAB для обернення матриць використовують функцію* inv(A).

При використанні методу обернення можна також додатково перевірити точність розрахунку оберненої матриці шляхом обчислення матричного

добутку вихідної матриці коефіцієнтів **A** та оберненої. Результатом повинна бути одинична діагональна матриця, наприклад, при виконанні програми

```
A=rand(5,5)
Ao=inv(A)
E=A*Ao
tol=eye(5)-E
```

отримаємо

```
A =
    0.8147    0.0975    0.1576    0.1419    0.6557
    0.9058    0.2785    0.9706    0.4218    0.0357
    0.1270    0.5469    0.9572    0.9157    0.8491
    0.9134    0.9575    0.4854    0.7922    0.9340
    0.6324    0.9649    0.8003    0.9595    0.6787
Ao =
    3.1375   -0.8078   -1.8788   -4.2194    5.1680
   -8.6076    3.5314    2.8907   13.7204  -14.3665
   -6.2824    3.7220    3.6132   10.0084  -12.4190
   13.6173   -6.8822   -6.3938  -23.5288   27.5825
   -2.5292    1.0729    2.4193    5.8870   -7.2671
E =
    1.0000    0.0000         0         0   -0.0000
   -0.0000    1.0000    0.0000   -0.0000   -0.0000
    0.0000   -0.0000    1.0000   -0.0000    0.0000
    0.0000         0   -0.0000    1.0000    0.0000
    0.0000   -0.0000         0   -0.0000    1.0000
tol =
   1.0e-14 *
    0.0666   -0.0333         0         0    0.0888
    0.0278   -0.0222   -0.0042    0.2026    0.1776
   -0.2220    0.0888    0.1776    0.6217   -0.2665
   -0.1776         0    0.0444    0.2665   -0.1776
   -0.0444    0.0222         0    0.2220   -0.2665
```

10.5. Розрахунок усталених режимів в електричних колах

У якості прикладу використання операцій лінійної алгебри для розв‘язання інженерних задач в галузі електротехніки розглянемо розрахунок струмів у розгалужених лінійних електричних колах. При вивченні теоретичних основ електротехніки (ТОЕ) математичний опис таких кіл складається методом законів Кірхгофа. У цьому разі кількість лінійних рівнянь у системі і кількість

невідомих струмів дорівнює кількості гілок. Для зменшення порядку системи рекомендують застосовувати методи контурних струмів, контурних напруг, еквівалентного генератора. Але при використанні сучасних ЕОМ та сучасного програмного забезпечення швидкість розв‘язання системи лінійних рівнянь є дуже високою, і потреби у зниженні їх порядку практично не існує.

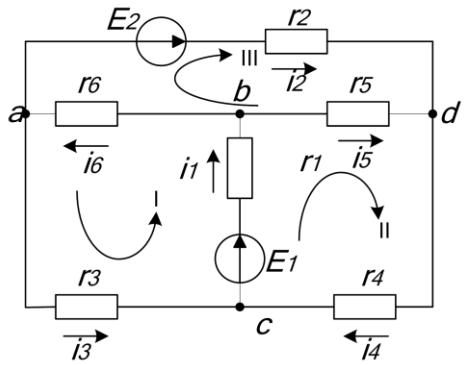


Рис. 10.2. Розгалужене електричне коло

Приклад 10.1. *Знайти струми в гілках розгалуженого електричного кола, зображеного на рис. 10.2 при відомих значеннях опорів резисторів та параметрів джерел струмів та ЕРС: r1=8 Ом; r2=10 Ом; r3=5 Ом; r4=7 Ом; r5=12 Ом; r6=3 Ом; E1=100 В; E2=30 В. Правильність результату перевірити методом балансу потужнотей.*

Досліджувана система має 6 гілок та 4 вузли, з них незалежними є 3 вузли. Позначимо їх буквами a, b, c. До незалежних вузлів додаємо 3 незалежних контури: I, II, III. *Виберемо напрямки струмів у гілках та додатні напрямки у незалежних контурах. Після цього складаємо систему 6 лінійних рівнянь з 6 невідомими: 3 за першим законом Кірхгофа і 3 за другим:*

$$\begin{array}{rl} a: & \left\{ \begin{array}{l} i_6 = i_2 + i_3, \\ b: \\ i_1 = i_6 + i_5, \\ c: \\ i_3 + i_4 = i_1, \\ \text{I:} \\ E_1 = i_1 r_1 + i_6 r_6 + i_3 r_3, \\ \text{II:} \\ E_1 = i_1 r_1 + i_4 r_4 + i_5 r_5, \\ \text{III:} \\ E_2 = i_2 r_2 + i_6 r_6 - i_5 r_5. \end{array} \right. \end{array} \qquad (10.23)$$

Формалізуємо систему (10.23) згідно з загальним виглядом системи лінійних рівнянь:

$$\begin{cases} 0\cdot i_1 \ -1\cdot i_2 \ -1\cdot i_3 \ +0\cdot i_4 +0\cdot i_5 +1\cdot i_6 = \ 0, \\ 1\cdot i_1 \ \ +0\cdot i_2 +0\cdot i_3 +0\cdot i_4 -1\cdot i_5 -1\cdot i_6 = \ 0, \\ -1\cdot i_1 +0\cdot i_2 +1\cdot i_3 \ +1\cdot i_4 +0\cdot i_5 +0\cdot i_6 = \ 0, \\ r_1\cdot i_1 +0\cdot i_2 +r_3\cdot i_3 \ +0\cdot i_4 +0\cdot i_5 +r_6\cdot i_6 = E_1, \\ r_1\cdot i_1 +0\cdot i_2 +0\cdot i_3 \ +r_4\cdot i_4 +r_5\cdot i_5 +0\cdot i_6 = E_1, \\ 0\cdot i_1 +r_2\cdot i_2 +0\cdot i_3 \ +0\cdot i_4 -r_5\cdot i_5 +r_6\cdot i_6 = E_2. \end{cases} \qquad (10.24)$$

З (10.24) складаємо матрицю коефіцієнтів та вектор вільних членів

$$\mathbf{A}=\begin{pmatrix} 0 & -1 & -1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & -1 & -1 \\ -1 & 0 & 1 & 1 & 0 & 0 \\ r_1 & 0 & r_3 & 0 & 0 & r_6 \\ r_1 & 0 & 0 & r_4 & r_5 & 0 \\ 0 & r_2 & 0 & 0 & -r_5 & r_6 \end{pmatrix}, \qquad \mathbf{B}=\begin{pmatrix} 0 \\ 0 \\ 0 \\ E_1 \\ E_1 \\ E_2 \end{pmatrix}. \qquad (10.25)$$

Тоді програма для розрахунку струмів у гілках схеми рис 10.2 матиме вигляд:

```
clc, clear all
r1=8; r2=10; r3=5; r4=7; r5=12; r6=3;
e1=100; e2=30;
A=[ 0 -1 -1  0   0  1;
     1  0  0  0  -1 -1;
    -1  0  1  1   0  0;
    r1  0 r3  0   0 r6;
    r1  0  0 r4  r5  0;
     0 r2  0  0 -r5 r6];
B=[0; 0; 0; e1; e1; e2];
i=A\B
E=[e1;e2;0;0;0;0];
P=sum(E.*i)
R=[r1; r2; r3; r4; r5; r6];
dP=sum(i.^2.*R)
```

В результаті її виконання отримуємо:

```
i =
   7.5975
   2.4584
   3.9806
   3.6169
   1.1585
   6.4390
P =
  833.5034
```

dP =

833.5034

Як бачимо, сумарна потужність джерел ЕРС співпадає з потужністю втрат електроенергії на активних опорах, тобто баланс потужностей виконується, що свідчить про правильність розрахунку струмів. Якщо баланс потужностей не виконується, то треба шукати похибку в рівняннях, складених за законами Кірхгофа.

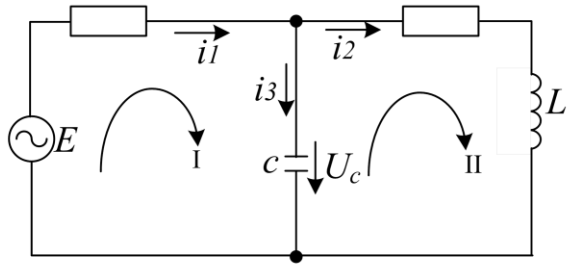


Рис. 10.3. Розгалужене електричне коло змінного струму

Приклад 10.2. *Побудувати графіки струмів та напруги на кондестаторі в гілках розгалуженого електричного кола змінного струму, зображеного на рис. 10.3, на одному періоді усталеного режиму при* $E=E_m\sin(\omega t)$, $E_m=220$ *В,* $\omega=2\pi f$, $f=50$*Гц,* $r_1=10$ *Ом;* $r_2=20$ *Ом;* $L=0.1$ *Гн;* $C=30$ *мкФ.*

Щоб побудувати графіки усталених синусоїдальних сигналів, необхідно розрахувати їх амплітуди і фазові зсуви відносно ЕРС джерела. Ці величини розраховуються, як і у попередньому прикладі шляхом розв'язання системи лінійних алгебраїчних рівнянь, складених за законами Кірхгофа. Різниця полягає в тому, що значення опорів гілок будуть комплексними числами:

$$z_1=R_1,\quad z_2=R_2+j\omega L,\quad z_3=\frac{1}{j\omega C}. \tag{10.26}$$

З урахуванням (10.26) згідно з законами Кірхгофа досліджуване електричне коло в усталеному режимі описується системою лінійних рівнянь 3-го порядку з комплексними коефіцієнтами:

$$\begin{cases} i_1=i_2+i_3, \\ E_m=i_1z_1+i_2z_2, \\ i_2z_2=i_3z_3. \end{cases} \tag{10.27}$$

Формалізуємо систему (10.27) згідно з загальним виглядом системи лінійних рівнянь:

$$\begin{cases} 1\cdot i_1-1\cdot i_2-1\cdot i_3=0, \\ z_1\cdot i_1+z_2\cdot i_2+0\cdot i_3=E_m, \\ 0\cdot i_3+z_2\cdot i_2-z_3\cdot i_3=0. \end{cases} \tag{10.28}$$

З (10.24) складаємо матрицю коефіцієнтів та вектор вільних членів

$$\mathbf{A}=\begin{pmatrix} 1 & -1 & -1 \\ z_1 & z_2 & 0 \\ 0 & z_2 & -z_3 \end{pmatrix}, \qquad . \tag{10.29}$$

Після розв'язання системи рівнянь (10.29) усталені струми отримаємо також у вигляді комплексних чисел, що складаються з дійсної та уявної частин:

$$i(j\omega)=i_{re}+ji_{im}. \qquad (10.30)$$

Їх треба перетворити до експоненційної форми

$$i(j\omega)=A_i e^{j\varphi_i}, \qquad (10.31)$$

де

$$A_i=\sqrt{i_{re}^2+i_{im}^2}, \qquad \varphi_i=\operatorname{arctg}\frac{i_{im}}{i_{re}}. \qquad (10.32)$$

Після цього можна записати рівняння усталеного періодичного режиму у функції часу:

$$i_{ycm}(t)=A_i\sin(\omega t+\varphi_i). \qquad (10.33)$$

Тоді програма для розрахунку струмів у гілках схеми рис 10.3 матиме вигляд:

```
clc, clear all
R1 = 10;  R2 = 20;  L = 0.1;  C = 30e-6;
f = 50;  T = 1/f;  w = 2*pi*f;
z1 = R1;  z2 = R2+j*w*L;  z3 = 1/(j*w*C);
Em = 220;
A = [ 1  -1  -1;
      z1  z2   0;
       0  z2  -z3]
B = [0; Em; 0]
i = A\B
im = abs(i)
phi = angle(i)
Uc = Em-i(2)*R2
Ucm = abs(Uc)
phi_c = angle(Uc)
t = linspace(0,T);
E = Em*sin(w*t);
Uc = Ucm*sin(w*t+phi_c);
plot (t,[E;Uc]), legend('E(t)','Uc(t)')
i1 = im(1)*sin(w*t+phi(1));
i2 = im(2)*sin(w*t+phi(2));
i3 = im(3)*sin(w*t+phi(3));
figure, plot (t,[i1;i2;i3]), legend('i1(t)','i2(t)','i3(t)')
```

В результаті її виконання отримуємо:

```
A =
  1.0e+02 *
```

```
   0.0100 + 0.0000i  -0.0100 + 0.0000i  -0.0100 + 0.0000i
   0.1000 + 0.0000i   0.2000 + 0.3142i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.2000 + 0.3142i   0.0000 + 1.0610i
B =
     0
   220
     0
i =
   3.0261 - 2.1932i
   3.2328 - 3.9815i
  -0.2067 + 1.7882i
im =
    3.7373
    5.1287
    1.8002
phi =
   -0.6272
   -0.8888
    1.6859
Uc =
   1.5534e+02 + 7.9630e+01i
Ucm =
  174.5637
phi_c =
    0.4737
```

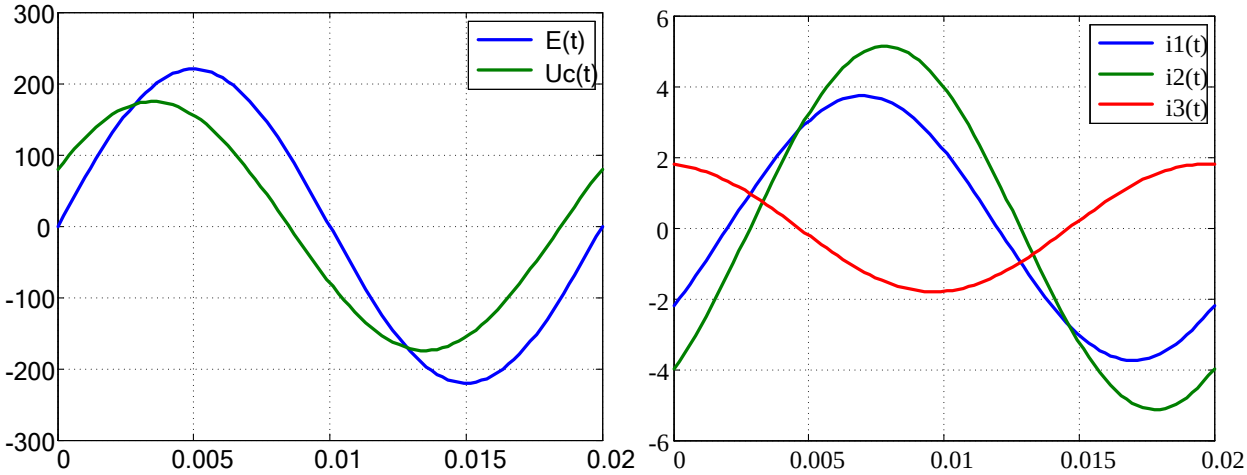


Рис. 10.4. Розв'язок задачі з прикладу 10.2.

10.6. Контрольні питання та завдання

1. Що таке мінор та алгебраїчне доповнення? Як їх розрахувати в *MATLAB*?
2. Що таке союзна та приєднана матриці?.

3. Що таке визначник матриці? Який метод використовують для його розрахунку? Поясніть алгоритм прямого ходу методу Гауса.
4. Які методи розв‘язання систем лінійних рівнянь ви знаєте?
5. Наведіть приклад використання систем лінійних рвнянь в теорії автоматичного керування.
6. Як розрахувати приєднану матрицю?
7. Що таке обернена матриця? Як її розрахувати?
8. Як розрахувати струми в гілках розгалуженого електричного кола постійного струму?
9. Як побудувати графіки усталених синусоїдальних сигналів в гілках розгалуженого електричного кола змінного струму?

11. АПРОКСИМАЦІЯ ТА ІНТЕРПОЛЯЦІЯ ТАБЛИЧНИХ ФУНКЦІЙ СТЕПЕНЕВИМИ ПОЛІНОМАМИ

11.1. Загальні поняття про апроксимацію та інтерполяцію

Існують три способи визначення функцій: аналітичний, табличний і графічний.

При моделюванні електротехнічних комплексів часто доводиться мати діло з ланками, статична характеристика яких уявляє собою нелінійну функціональну залежність, що не має точного або достатньо простого аналітичного опису (наприклад, криві намагнічування двигунів, статичні механічна та електромеханічна характеристики асинхронного двигуна, залежність зусилля різу летючих ножиць від кута повороту ножів та т.і.). Такі характеристики на практиці отримують експериментальним шляхом. У довідниках їх приводять у вигляді таблиць (див. табл. 11.1) або у вигляді графіків, побудованих за табличними даними (див. рис. 11.1).

Таблиця 11.1

i	1	2	3	…	n
x_i	x_1	x_2	x_3	…	x_n
y_i	y_1	y_2	y_3	…	y_n

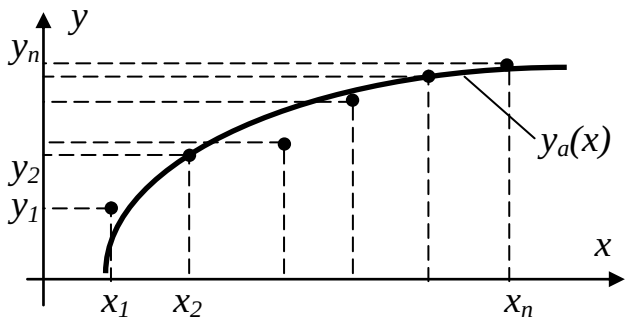


Рис.11.1

В ЕОМ інформацію про табличні функції $y_i(x_i)$ зберігають у вигляді масивів даних $\mathbf{X}=[x_1\ x_2\ \ldots\ x_n]$ та $\mathbf{Y}=[y_1\ y_2\ \ldots\ y_n]$. Табличні значення аргументів x_i називають в***узлами таблиці***, а відповідні значення функції – ***значеннями функції у вузлах***.

Оскільки будь-яка таблична функція $y_i(x_i)$ за своїм характером дискретна, то виникає ***задача розрахунку значення такої функції в довільній точці інтервалу її існування, яка в загальному випадку не співпадає з вузлом***

таблиці. Така задача може бути розв‘язана шляхом ***заміни табличної функції деякою аналітичною функцією, достатньо близькою до вихідної табличної***. Таку аналітичну функцію $y_a(x)$ називають ***апроксимуючою***. ***Апроксимація*** (***від лат. approximo – наближатися)*** – це приблизне вираження будь-яких залежностей через інші, звичайно більш простіші.

Для виконання апроксимації необхідно конкретизувати вид апроксимуючої функції. В обчислювальній математиці для апроксимації найчастіше застосовують такі функції:

1) степеневі поліноми та їх комбінації

$$y_a(x)=\sum_{i=0}^{n} a_i x^{n-i}, \qquad y_a(x)=\frac{\sum_{i=0}^{m} b_i x^{m-i}}{\sum_{i=0}^{k} b_i x^{k-i}};$$

2) лінійні комбінації експоненціальних функцій:

$$y_a(x)=\sum_{i=1}^{n} A_i e^{\alpha_i x};$$

3) комбінації періодичних функцій, що породжують ряди Фур‘є

$$y_a(x)=\sum_{i=1}^{n}\left(A_i \cos\omega_i x + B_i \sin\omega_i x\right);$$

4) нелінійні функції, які за допомогою лінеаризуючих перетворень вдається звести до степеневого поліному (найчастіше за все 1-го порядку):

$$c_0+\frac{c_1}{x}, \qquad \frac{x}{c_0+c_1 x}, \qquad c_0 c_1^{x}, \qquad c_0+c_1\ln x.$$

Апроксимацію тригонометричними функціями виконують для періодичних функцій. Для неперіодичних функцій найважливішим класом апроксимуючих функцій є алгебраїчні степеневі поліноми. Їх легко обчислювати (схема Горнера), виконувати над ними арифметичні операції, а також інтегрувати та диференціювати. До того ж при зміні порядку поліному та його коефіцієнтів суттєво змінюється вигляд графіку поліноміальної функції.

Згідно з ***апроксимаційоною теоремою Вейерштраса***, якщо $f(x)$ – неперервна функція на інтервалі $[a;b]$, то для будь-якого $\varepsilon>0$ існує поліном $P(x)$ такий, що

$$\max_{x\in[a;b]}|f(x)-P(x)|<\varepsilon. \qquad (11.1)$$

Але поліном, що задовольняє умові (11.1), зазвичай має зависокий порядок, і не існує універсального алгоритму його синтезу. Як бачимо, в загальному випадку апроксимуюча функція може не проходити через жодну з вузлових точок таблиці. Тому на практиці зазвичай застосовують ***апроксимацію методом найменших квадратів***, яка забезпечує мінімальну суму квадратів відхилень апроксимуючої функції від табличної у вузлових точках:

$$\Phi=\sum_{i=1}^{n}(y_a(x_i)-y_i)^2. \qquad (11.2)$$

Існує ще інший підхід для розв‘язання поставленої задачі. Він полягає у тому, що критерієм близькості апроксимуючої і табличної функцій вважають їх рівність у вузлових точках:

$$y_a(x_i)=y_i,\ \ i=1,2,\ldots,n. \qquad (11.3)$$

Апроксимуюча функція, що відповідає умові (11.3), називається ***інтерполюючою***. Процес пошуку такої функції на заданому інтервалі зміни аргументу $[x_0;x_n]$ називають ***інтерполяцією*** (від лат. *inter* – внутри, *pole* – узел), а зовні цього інтервалу – ***екстраполяцією*** (від лат. *extra* – зовні).

Розрізняють ***глобальну і локальну інтерполяцію***. ***Глобальний поліноміальний інтерполянт (англ. global fit) уявляє собою степеневий поліном, порядок якого є на одиницю меншим за кількість вузлів табличної функції з такими коефіцієнтами, що задовольняють умовам:***

$$P_{n-1}(x_i)=C_0x_i^{n-1}+C_1x_i^{n-2}+\ldots+C_{n-2}x+C_{n-1}=y_i,\ \ i=1,2,\ldots,n. \qquad (11.4)$$

Система лінійних рівнянь (11.4) має рішення, при тому тільки одне. При зменшенні порядку полінома система стає недовизначеною, а при збільшенні порядку – перевизначеною. В обох випадках вона немає розв‘язку.

При великій кількості табличних точок розрахунок глобального інтерполянту або його коефіцієнтів є трудомісткою задачею. До того ж, ці зусилля є не завжди оправданими, тому що рівність інтерполюючої і табличної функції у вузлах не гарантує плавної зміни інтерполянта між вузлами. Тому при розв‘язанні інженерних задач звичайно використовують ***інтерполяцію рухомими поліномами*** невисокого порядку. ***Таку інтерполяцію ще називають кусковою, локальною, або кусково-поліноміальною***.

Вади глобальної поліноміальної інтерполяції уперше виявив Рунге в 1901р. Він намагався інтерполювати функцію резонансного типу $y(x)=\frac{1}{1+25x^2}$ при рівномірному розбитті інтервалу інтерполювання. В результаті виявилось, що при підвищенні порядку глобального поліноміального інтерполянта підвищується похибка інтерполювания між окремими вузловими точками на «крилах» резонансної кривої, як це показано на рис. 11.2.

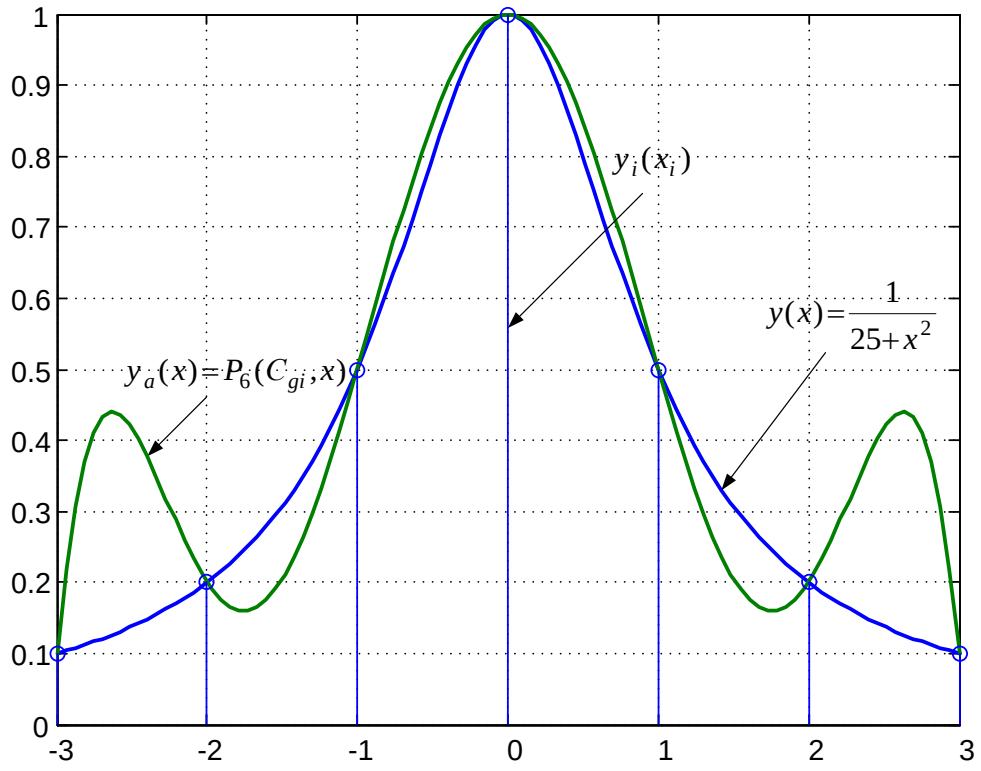


Рис. 11.2. Демонстрація вад глобальної поліноміальної інтерполяції

Рунге показав, що явище хвилястості інтерполянта можна усунути розподіленням абсцис вузлів не рівномірно, а дещо інакше. Але універсальна

схема розподілення вузлів таблиці не була знайдена. Звідси і виникла ідея кусково-поліноміальної інтерполяції.

Її сенс розглянемо на прикладі інтерполяції рухомими поліномами 1-го порядку, яка забезпечує так звану ***кусково-лінійну апроксимацію*** табличної функції (див. рис. 11.3).

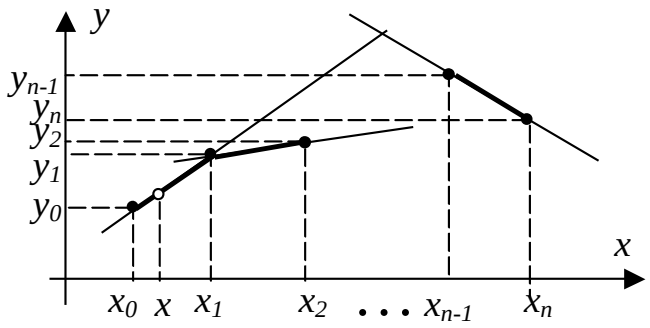


Рис. 11.3. Кусково-лінійна апроксимація

У цьому разі при пошуку $y(x)$ у довільній точці інтервалу визначення табличної функції кожного разу використовують рівняння прямої лінії, що проходить через 2 сусідніх вузла.

Тому задача локальної інтерполяції розбивається на 2 етапи: 1) пошук початкового вузла інтерполяції, тобто вузла, розташованого найближче до точки інтерполяції ліворуч (інтерполювання вперед) або праворуч (інтерполювання назад); 2) конструювання локального полінома за обраною інтерполяційною формулою. Недоліком кусково-лінійної апроксимації є наявність зламів апроксимуючої функції у вузлових точках.

Цей недолік стає майже непомітним при застосуванні інтерполяції рухомими поліномами 2-3 порядків.

Але найбільш гладку апроксимацію забезпечує ***інтерполяція кубічними сплайнами*** (від англ. spline – рейка).

Для проведення гладких кривих через вузлові точки креслярі здавна застосовували гнучкі рейки (лекала), виготовлені з пружного матеріалу. Механічні сплайни створюють, підвішуючи грузила у вузлах інтерполяції гнучкої рейки. Через деякий час вона сама по собі приймає форму кубічного сплайна, що забезпечує мінімум потенціальної енергії пружного матеріалу.

Математична ***теорія сплайн-функцій*** почала розвиватися в кінці 1960-х років.

З використанням положень теорії балок доведено, що форма, яку приймає механічний сплайн між кожною парою сусідніх вузлів, описується кубічним поліномом і те, що сусідні поліноми, а також їх перші та другі похідні з‘єднуються між собою неперервно.

При розв‘язанні задачі інтерполяції кубічними сплайнами табличної функції, заданої в n точках, необхідно вирішити n–1 систему чотирьох лінійних рівнянь з чотирма невідомими, в результаті чого визначаються параметри сплайнів. Розрахунок інтерпольованих значень виконують за схемою Горнера.

Недоліком сплайн-інтерполяції є складність розрахунків, що потребують значного машинного часу, та низька точність екстраполяції.

11.2. Апроксимація степеневими поліномами методом найменших квадратів

Апроксимація табличної функції

$$y_i = f(x_i); \qquad i=1, 2, \dots, n \tag{11.5}$$

степеневими поліномами методом найменших квадратів (МНК) полягає у визначенні коефіцієнтів степеневого поліному

$$y_a(x)=P_k(x)=c_0+c_1x+c_2x^2+\ldots+c_kx^k=\sum_{j=0}^{k}c_jx^j, \tag{11.6}$$

що забезпечують мінімізацію функціоналу

$$\Phi=\sum_{i=1}^{n}\left(P_k(x_i)-y_i\right)^2=\sum_{i=1}^{n}\left(\sum_{j=0}^{k}c_jx_i^j-y_i\right)^2. \tag{11.7}$$

Порядок апроксимуючого поліному повинен відповідати умові

$$k\le n-1. \tag{11.8}$$

Умова (11.8) випливає із узагальнення відомих положень: «через 2 точки можна провести пряму, при тому тільки 1» і «через 3 точки можна провести параболу, при тому тільки 1». Оскільки пряма описується поліномом першого порядку, а парабола – поліномом другого порядку, то узагальнення буде таким:

«***через n точок на площині можна провести криву, що описується степеневим багаточленом порядку n-1, при тому тільки 1***».

Для визначення потрібного вектору коефіцієнтів $\mathbf{c}=[c_0\ c_1\ c_2\ \dots\ c_k]$ необхідно розв‘язати систему лінійних рівнянь (k+1)-го порядку:

$$\frac{\partial\Phi}{\partial c_0}=0;\qquad \frac{\partial\Phi}{\partial c_1}=0;\quad \dots;\quad \frac{\partial\Phi}{\partial c_k}=0, \tag{11.9}$$

де

$$\frac{\partial \Phi}{\partial c_m}=\sum_{i=1}^{n}2\left(\sum_{j=0}^{k}c_j x_i^j-y_i\right)x_i^m=2\sum_{i=1}^{n}\sum_{j=0}^{k}c_j x_i^{j+m}-2\sum_{i=1}^{n}y_i x_i^m;\ m=0, 1, \dots, k.$$

Після перетворень система (11.9) набуває вигляду:

$$\begin{cases} c_0 n+c_1\sum\limits_{i=1}^{n}x_i+c_2\sum\limits_{i=1}^{n}x_i^2+\dots+c_k\sum\limits_{i=1}^{n}x_i^k=\sum\limits_{i=1}^{n}y_i; \\ c_0\sum\limits_{i=1}^{n}x_i+c_1\sum\limits_{i=1}^{n}x_i^2+c_2\sum\limits_{i=1}^{n}x_i^3+\dots+c_k\sum\limits_{i=1}^{n}x_i^{k+1}=\sum\limits_{i=1}^{n}y_i x_i; \\ \dots \\ c_0\sum\limits_{i=1}^{n}x_i^k+c_1\sum\limits_{i=1}^{n}x_i^{k+1}+c_2\sum\limits_{i=1}^{n}x_i^{k+2}+\dots+c_k\sum\limits_{i=1}^{n}x_i^{2k}=\sum\limits_{i=1}^{n}y_i x_i^k. \end{cases} \tag{11.10}$$

Вона уявляє собою систему (k+1) лінійних рівнянь з (k+1) невідомими $\mathbf{C}=[c_0\ \ c_1\ \dots\ c_k]$. З (11.9) бачимо, що елементи матриці коефіцієнтів **A** і вектора вільних членів **B** цієї системи можна описати формулами:

$$\begin{cases} a_{mj}=\sum\limits_{i=1}^{n}x_i^{m+j}, \\ b_m=\sum\limits_{i=1}^{n}x_i^m\cdot y_i, \end{cases} \tag{11.11}$$

де m=0, 1, …, k; j=0, 1, …, k.

Після розрахунку коефіцієнтів (11.11) систему (11.10) можна вирішити будь-яким з відомих методів.

Апроксимацію методом найменших квадратів часто застосовують для вирівнювання (згладжування, англ. ***fitting***) табличних функцій, що були

отримані в ході експерименту, а також для зменшення об'єму інформації про табличні функції при невисоких вимогах до точності розрахунку.

На практиці дуже рідко застосовують апроксимацію поліномами при $k>5$, тому що в багатьох випадках матриця коефіцієнтів у системі рівнянь (11.10) при підвищенні її порядку стає погано обумовленою (тобто її визначник стає дуже малим), і похибки розв'язку системи стають занадто великими.

В *MATLAB* для знаходження коефіцієнтів апроксимуючого поліному методом найменших квадратів використовується функція polyfit*:*

C = polyfit (Xt, Yt, k),

де C – вектор-рядок коефіцієнтів апроксимуючого полінома, упорядкований за зменшенням степені *x*;

Xt, Yt – вектори аргументів та значень табличної функції;

k – порядок апроксимуючого полінома.

Приклад 11.1. *Апроксимувати отриману експериментально Вольт-Амперну характеристику степеневим поліномом першого порядку.*

Текст script-файла може мати такий вигляд:

```
clc;
% Табл. функція задається двома векторами однакового розмміру:
It=0:.1:1;
Ut=[0 0.12 0.24 0.34 0.38 0.48 0.55 0.6 0.69 0.75 0.8];
d=polyfit(It,Ut,1);     % Визначення коефіцієнтів апрокс. полінома
ds=poly2str(d,'I')
I=linspace(min(It),max(It)); % Задаємо 100 точок на інтервалі існування табличної функції
U=polyval(d,I);             % Розраховуємо значення апрокс. функції в цих точках
Uat=polyval(d,It);          % Розраховуємо значення апрокс. функції у вузлових точках
func=sum((Uat-Ut).^2)   % Розраховуємо мінімізований функціонал
figure
plot(It,Ut,'*',I,U) % Будуємо графіки табличної та апрокс. функцій в одному вікні
grid on
```

Результати работи програми мають вигляд:

```
ds =
  0.77818 I + 0.060909
func =
   0.0099
```

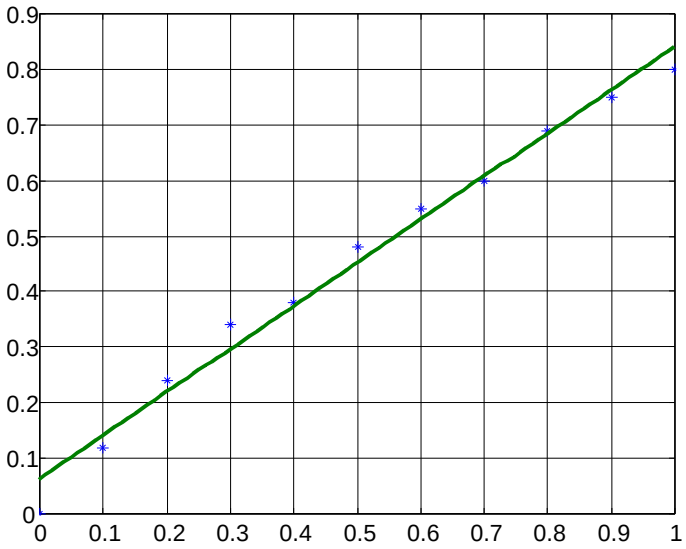


Рис. 11.4. Лінійна апроксимація експериментальної вольт-амперної характеристики до прикладу 11.1

11.3. Глобальне та локальне інтерполювання табличних функцій степеневими поліномами

Задача інтерполяції полягає у формуванні поліноміальної функції k-ої степені, яка приймає на деякому інтервалі $[x_j, x_{j+k}]$ значення, що співпадають зі значеннями табличної функції $y_i(x_i)$ $(i=j, j+1, ..., j+k)$ у $(k+1)$-й вузлових точках:

$$P_k(x_j)=y_j,\ P_k(x_{j+1})=y_{j+1},\ ...,\ P_k(x_{j+k})=y_{j+k} \qquad (11.12)$$

та в розрахунку значень знайденої функції y точках, відмінних від вузлових $(x \neq x_i)$.

Точку x_j звуть початковим вузлом інтерполяції. Якщо $i=0,1,2,\ldots,n$, а $k=n$, то знайдена функція є глобальним інтерполянтом, бо в цьому випадку його значення співпадають зі значеннями початкової функції в усіх вузлах:

$$P_n(x_i)=a_0+a_1x_i+a_2x_i^2+\ldots+a_nx_i^n=\sum_{j=0}^{n}a_jx_i^j\ ,\quad i=0,1,2,\ldots,n\,. \qquad (11.13)$$

Поліном, що задовольняє вимогам (11.13), називають ***канонічним***. Його коефіцієнти знаходять шляхом розв‘язання системи $n+1$ лінійних рівнянь з

$n+1$ невідомими:

$$\begin{cases} a_0 + a_1 x_0 + a_2 x_0^2 + \ldots + a_n x_0^n = y_0; \\ a_0 + a_1 x_1 + a_2 x_1^2 + \ldots + a_n x_1^n = y_1; \\ \ldots \\ a_0 + a_1 x_n + a_2 x_n^2 + \ldots + a_n x_n^n = y_n. \end{cases} \tag{11.14}$$

Визначник матриці коєфіцієнтів системи (11.14)

$$\mathbf{A} = \begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^n \\ & & \ldots & & \\ 1 & x_n & x_n^2 & \ldots & x_n^n \end{pmatrix} \tag{11.15}$$

називають ***визначником Вандермонда***. Доведено, що його можна розрахувати за формулою

$$\det(\mathbf{A}) = \prod_{\substack{i,j=0 \\ i \neq j}}^{n} (x_i - x_j) \neq 0. \tag{11.16}$$

Як і у випадку апроксимації методом найменших квадратів, матриця коефіцієнтів (11.15) при зростанні кількості табличних точок, а відповідно і степені поліному стає погано обумовленою. Тому частіше замість глобальної інтерполяції канонічним поліномом на інтервалі $[x_0, x_n]$ на кожній з ділянок $[x_0, x_k], [x_1, x_{k+1}], \ldots, [x_{n-k}, x_n]$ використовують різні (локальні) поліноми.

При локальній інтерполяції застосовують дещо інший підхід до визначення потрібного полінома, ніж при апроксимації методом найменших квадратів та глобальній інтерполяції, коли спочатку здійснюють пошук коефіцієнтів полінома, а потім значення полінома розраховують з використанням схеми Горнера. В даному випадку коефіцієнти поліномів залишаються невідомими, а формування поліномів відбувається за інтерполяційними формулами, в яких вихідними даними є координати вузлових точок табличної функції. Існують інтерполяційні формули Ньютона, Лагранжа, Бесселя, Стірлінга та деякі інші.

До найбільш часто вживаних інтерполяційних формул належить ***формула Лагранжа***. До її переваг можна віднести:

- можливість застосування як при рівномірному, так і при нервномірному розташуванні абсцис вузлових точок;
- входження координат вузлових точок у формулу в явному вигляді;
- існування зручних алгоритмів формування поліному *k*-ого порядку з поліному (*k*-1)-ого порядку.

Інтерполяційна формула Лагранжа має вигляд:

$$y(x) \approx L_k(x) = \\ = \sum_{m=j}^{j+k} y_m \frac{(x-x_j)(x-x_{j+1})\ldots(x-x_{m-1})(x-x_{m+1})\ldots(x-x_{j+k})}{(x_m-x_j)(x_m-x_{j+1})\ldots(x_m-x_{m-1})(x_m-x_{m+1})\ldots(x_m-x_{j+k})} \qquad (11.17)$$

Формулу (11.17) можна застосувати для пошуку *y(x)* на інтервалі $[x_j, x_{j+k}]$, але найбільшу точність вони забезпечують поблизу початкового вузла інтерполяції x_j: $x \in [x_j, x_{j+1}]$.

Тому, перед тим як використовувати інтерполяційні формули, необхідно знайти номер початкового вузла інтерполяції.

У технічних розрахунках звичайно застосовують лінійну або квадратичну інтерполяцію. У такому випадку формула (11.17) набуває таких форм:
при k=1

$$y(x) \approx L_1(x) = y_i \frac{(x-x_{j+1})}{(x_j-x_{j+1})} + y_{j+1} \frac{(x-x_j)}{(x_{j+1}-x_j)}; \qquad (11.18)$$

при k=2

$$y(x) \approx L_2(x) = y_j \frac{(x-x_{j+1})(x-x_{j+2})}{(x_j-x_{j+1})(x_j-x_{j+2})} + \\ + y_{j+1} \frac{(x-x_j)(x-x_{j+2})}{(x_{j+1}-x_j)(x_{j+1}-x_{j+2})} + y_{j+2} \frac{(x-x_j)(x-x_{j+1})}{(x_{j+2}-x_j)(x_{j+2}-x_{j+1})} \qquad (11.19)$$

Формула (11.18) являє собою рівняння прямої, яка проходить через точки (x_j, y_j), (x_{j+1}, y_{j+1}), а (11.19) – рівняння квадратичної параболи, яка проходить

через точки (x_j, y_j), (x_{j+1}, y_{j+1}), (x_{j+2}, y_{j+2}).

Алгоритмічно розв‘язання задачі кусково-поліноміальної інтерполяції складається з двох частин: 1) визначення початкового узла інтерполяції; 2) застосування інтерполяційної формули.

Для інтерполювання в пакеті MATLAB використовується функція interp1*:*

Y=interp1(Xt, Yt, X, method),

де Xt, Yt – вектори аргументів та значень табличної функції;

X – точка або масив точок, у яких необхідно обчислити значення інтерпольованої функції;

method – метод інтерполювання, може приймати наступні значення:

- 'linear' – лінійна інтерполяція (кусково-лінійна апроксимація);
- 'cubic' – кубічна інтерполяція;
- 'spline' – кубічна сплайн-інтерполяція;
- 'nearest' – інтерполяція за найближчим сусіднім вузлом (*nearest neighbour*).

Слід зауважити, що для методів 'linear' та 'nearest' режим екстраполяції за замовченням виключено. Щоб його включити, треба додати до списку вхідних параметрів в кінці ще один параметр: 'extrap'.

Наведемо приклад програми, що здійснює різні види інтерполяції, та результати її виконання:

```
clc, clear all, close all
xt=0:10;
yt=[0 3.5 2.5 3.2 4 4.5 5 6 7 8 8.25];
x=linspace(min(xt)-0.25,max(xt)+1,200);
yl=interp1(xt,yt,x,'linear','extrap');
yn=interp1 (xt,yt,x,'near','extrap');
yc=interp1 (xt,yt,x,'cubic');
ys=interp1 (xt,yt,x,'spline');
stem(xt,yt,'k--'), hold on, grid on
plot(x,yl,'r'), stairs(x,yn,'b')
figure
stem(xt,yt,'k--'), hold on, grid on
plot(x,yc)
figure
stem(xt,yt,'k--'), hold on, grid on
plot(x,ys), ylim([-2 10])
```

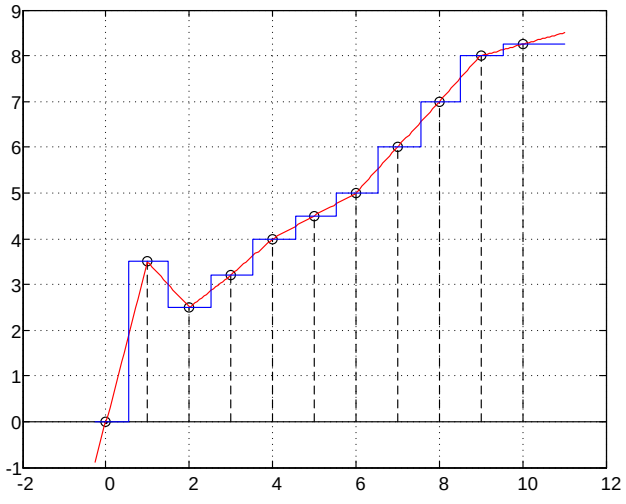


Рис. 11.5. Інтерполювання методами

'linear' та 'nearest'

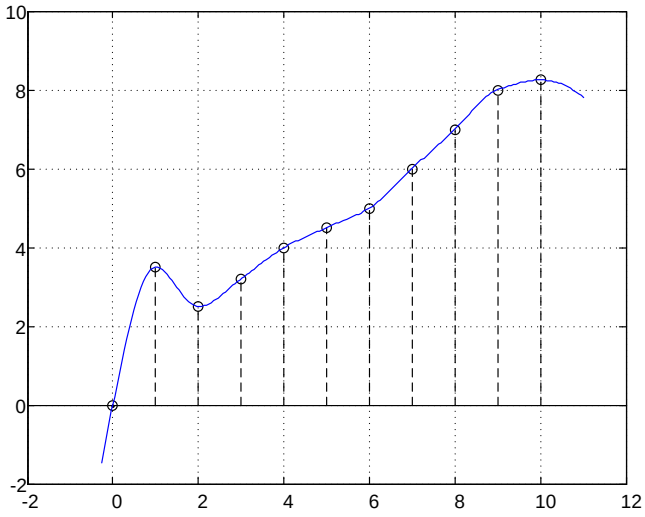


Рис. 11.6. Інтерполювання методом 'cubic'

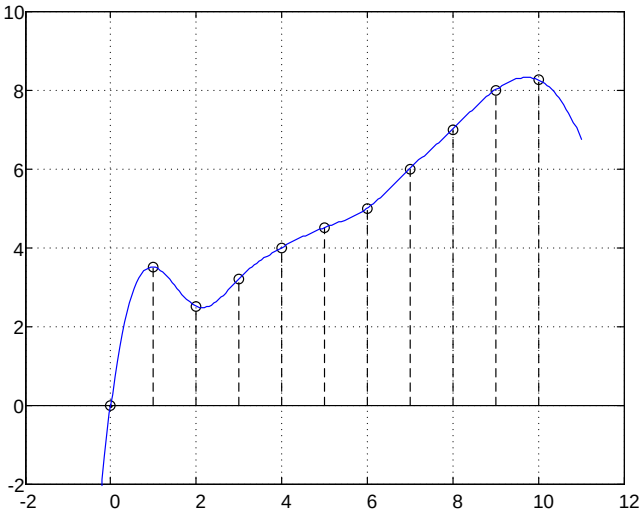


Рис. 11.7. Інтерполювання методом 'spline'

Наведені графіки наочно демонструють переваги і недоліки різних видів інтерполяції. Слід звернути увагу на недоцільність використання екстраполяції, результати якої дуже сильно залежать від типу інтерполяції, а точність не можливо оцінити із-за відсутності інформації про поведінку функції поза інтервалом її визначення.

У Simulink кусково-лінійну інтерполяцію табличної функції за формулою (11.18) виконує блок ***1D-Look-Up Table*** **(*Одномірний Функціональний Перетворювач*)** з параметрами *Breakpoints* (табличні значення вхідного сигналу) та *Table data* (табличні значення вихідного сигналу)**.** Іконка блоку ***1D-Look-Up Table*** та вікно введення його параметрів зображені на рис. 11.8 і 11.9 відповідно.

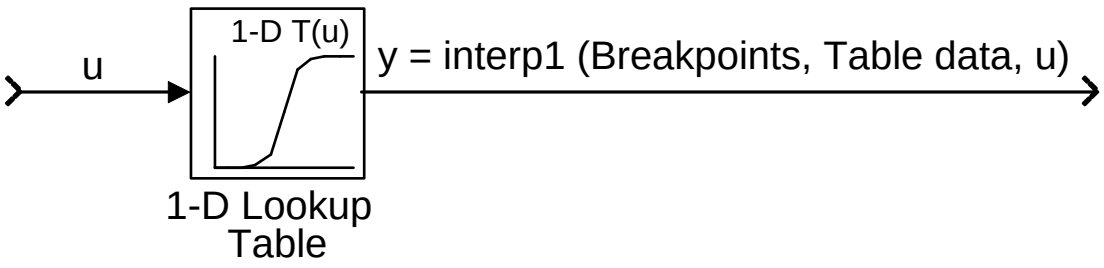


Рис.11.8. Застосування локальної інтерполяції при моделюванні нелінійних блоків у *Simulink*

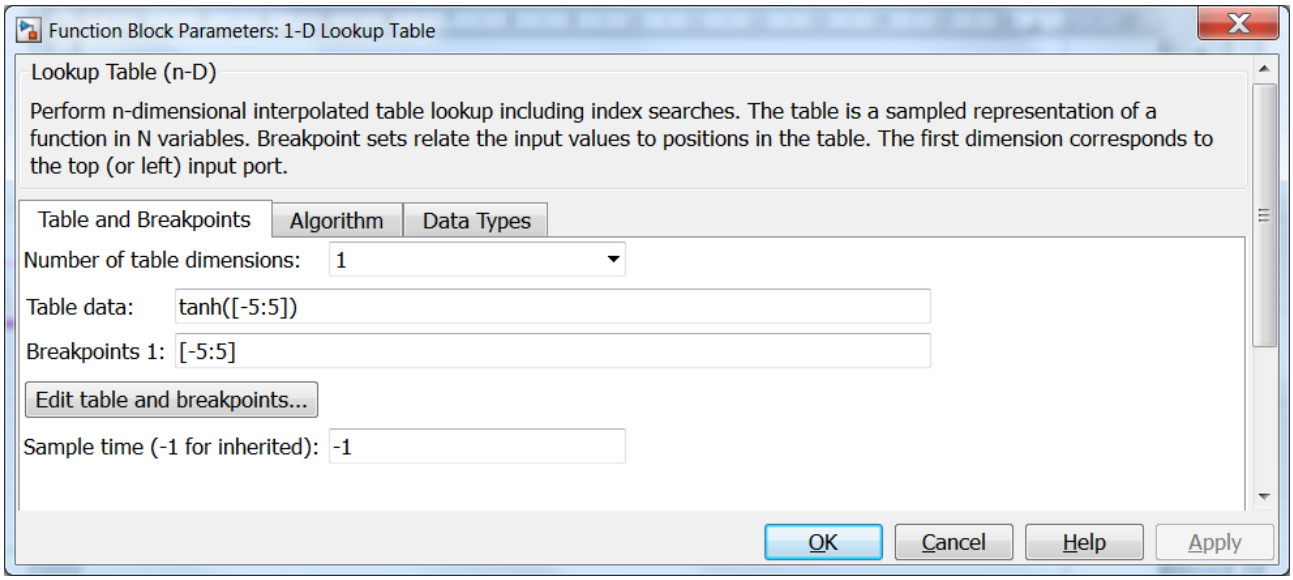


Рис.11.9. Вікно параметрів блоку *1D-Look-Up Table*

Блок *1D-Look-Up Table* може виконувати не тільки кусково-лінійну апроксимацію заданої табличної функції, але й інтерполювання кубічними сплайнами, для чого треба обрати відповідний параметр, к вкладці *Algorthm*, які показана на рис. 11.10.

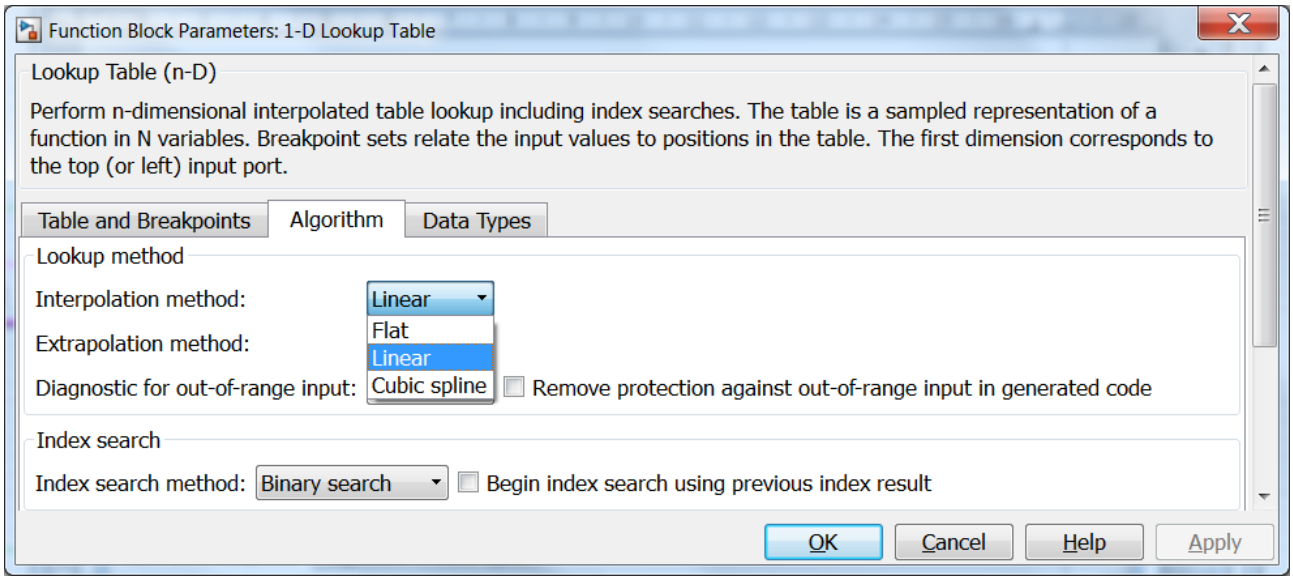


Рис.11.10. Вкладка *Algorthm* вікна параметрів блоку *1D-Look-Up Table*

Блок одновимірної інтерполяції легко перетворюється у багатовимірний шляхом використання параметру *Number of table dimensions* (див. рис. 11.11).

Інтерполювання широко використовується не тільки для моделювання нелінійних елементів з експериментально знятими характеристиками «вхід-вихід», але й у системах оптимального керування електричними двигунами, коли в режимі *off-line* визначають залежність якоїсь координати електроприводу від однієї або декількох інших координат, яка забезпечує досягнення мети керування, а потім цю залежність використовують в алгоритмі керування. Наприклад, в системах векторного керування асинхронним двигуном для підтримки ККД на максимально можливому рівні необхідно змінювати потокозчеплення ротора у функції електромагнітного моменту та швидкості двигуна.

11.4. Контрольні питання та завдання

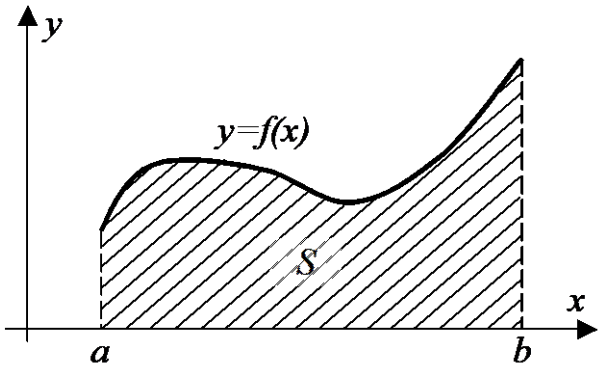
1. Які задачі вирішують методи апроксимації та інтерполювання. Яка між ними принципова різниця?
2. У чому полягає сутність методу найменших квадратів?
3. Як залежить максимальний порядок апроксимуючого полінома від кількості точок, у яких задана таблична функція?
4. За яких умов мінімізований функціонал при використанні МНК стає нульовим?
5. У яких випадках не слід намагатися підвищувати порядок апроксимуючого СП до максимально можливого?
6. Яким буде результат операції polyfit(1:4,[1 4 9 16],2)?
7. Яка різниця між глобальним та локальними інтерполянтами?
8. Що таке кусково-лінійна інтерполяція?
9. Чому досить часто замість глобальної інтерполяції використовують інтерполяцію рухомими локальними поліномами невисокого порядку?
10. Яка різниця між кубічною інтерполяцією та інтерполяцією кубічними сплайнами?

11. Запишіть рівняння прямої, що проходить через дві точки з відомими координатами, та рівняння параболи, що проходить через три точки з відомими координатами.
12. Як реалізувати кусково-лінійну апроксимацію та інтерполяцію кубічними сплайнами в *Simulink*?

12. ЧИСЕЛЬНЕ ІНТЕГРУВАННЯ

12.1. Загальні відомості

Завдання, у яких потрібне обчислення інтегралів, з'являються практично у всіх галузях прикладної математики. Основу машинних алгоритмів розрахунку визначених інтегралів складає їх геометричний сенс (див. рис. 11.1):



$$\int_a^b f(x)dx = S$$

Рис. 12.1. Геометричний сенс визначеного інтеграла

Більшість із методів чисельного інтегрування (ЧІ) засновані на тому, що інтервал [*a, b*] роздріблюють на частки, на кожній з котрих крива, яка описується підінтегральною функцією *f*(*x*), замінюється якоюсь іншою кривою, для якої обчислення інтегралу здійснюється за достатньо простими формулами, а потім усі площі складаються разом.

У випадку заміни підінтегральної функції інтерполюючими поліномами отримують так звані ***квадратурні формули***. Ці формули для рівновіддалених вузлів інтерполяції називають ***формулами Ньютона-Котеса***.

У залежності від порядку інтерполюючого полінома розрізняють такі методи ЧІ:

- $k = 0$ – метод прямокутників,
- $k = 1$ – метод трапецій,
- $k = 2$ – метод Сімпсона, або метод квадратичних трапецій.

Кожний з методів розглянемо для таких трьох випадків:

1) підінтегральна функція є табличною при нерівномірному розподіленні вектору аргументів:

$$y_i = f(x_i), \qquad i_i = 0,1,2,\ldots,n, \qquad Z = \int_{x_0}^{x_n} y_i(x_i)dx; \tag{12.1}$$

$$\Delta x_i = x_{i+1} - x_i = var; \tag{12.2}$$

2) підінтегральна функція є табличною при рівномірному розподіленні вектору аргументів:

$$\Delta x = x_{i+1} - x_i = h = const; \tag{12.3}$$

3) підінтегральна функція задана аналітично: $y=f(x)$

$$y = f(x), \qquad Z = \int_a^b y(x)dx; \tag{12.4}$$

$$x_0 = a, \quad x_n = b, \quad h = \frac{b-a}{n}, \quad x_i = a + ih, \quad i_i = 0, 1, 2, ..., n. \tag{12.5}$$

У формулах (12.1)-(12.5) n – кількість ділянок (відрізків), на які розбито інтервал інтегрування. Кількість точок у векторах абсцис та ординат завжди буде перевищувати кількість ділянок на одиницю.

12.2. Метод прямокутників

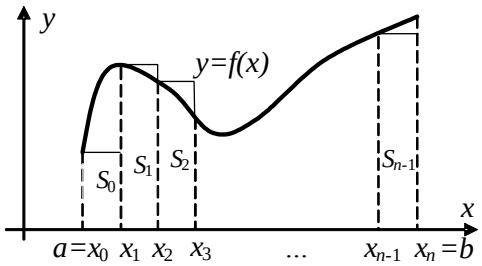


Рис. 12.2. Ілюстрація методу прямокутників

При k=0 зачення інтерпольованої функції на кожній з ділянок інтервалу інтегрування є константою (див. рис. 12.2): $y_i = P_0(x_i) = const,\ i = 0,1,\ldots,n-1,$ а графіки уявляють собою прямі лінії, паралельні осі абсцис.

Тому площина S (рис. 12.1), що дорівнює визначеному інтегралу, приблизно розраховується як сума площин n прямокутників.

Площа i-го прямокутника визначається як $S_{\text{пр}i} = y_i(x_{i+1} - x_i)$ або при виконанні умови (12.3) – як $S_{\text{пр}i} = hy_i$. Тоді значення інтегралів для 3-х досліджуваних випадків можна розрахувати за формулами

$$Z_{\text{пр}} \approx \sum_{i=0}^{n-1} y_i(x_{i+1} - x_i), \quad Z_{\text{пр}} \approx h\sum_{i=0}^{n-1} y_i, \quad Z_{\text{пр}} \approx \frac{b-a}{n}\sum_{i=0}^{n-1} f(a+ih). \tag{12.6}$$

Формули (12.6) надзвичайно просто записуються на алгоритмічній мові *MATLAB*, наприклад, перша з цих формул з урахуванням відсутності в *MATLAB* нульових індексів виглядатиме так:

```
Zpr = sum (y(1:end-1).*diff(x))
```

12.3. Метод трапецій

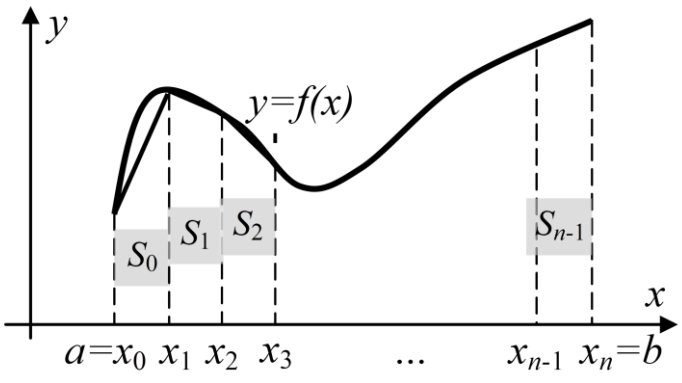


Рис. 12.2. Ілюстрація методу трапецій

При k =1 графіки локальних інтерполянтів уявляють собою прямі лінії, що з‘єднують між собою дві сусідні вузлові точки табличної функції, а сукупність фігур, на які приблизно розбивається площина S, уявляє собою сімейство трапецій.

Площа i-ї трапеції при виконанні умови (12.2) визначається як $S_{\text{тр}i}=\frac{y_i+y_{i+1}}{2}\left(x_{i+1}-x_i\right)$, а при виконанні умови (12.3) – як $S_{\text{тр}i}=\frac{y_i+y_{i+1}}{2}h$ або– як. Відповідно значення інтегралів для досліджуваних випадків можна розрахувати за формулами

$$Z_{\text{тр}}\approx\sum_{i=0}^{n-1}\frac{y_i+y_{i+1}}{2}(x_{i+1}-x_i),\qquad Z_{\text{тр}}\approx h\left(\frac{y_0+y_n}{2}+\sum_{i=1}^{n-1}y_i\right),\tag{12.7}$$

$$Z_{\text{пр}}\approx\frac{b-a}{n}\left(\frac{f(a)+f(b)}{2}+\sum_{i=1}^{n-1}f(a+ih)\right).\tag{12.8}$$

Для прикладу запишемо формули (12.7) на алгоритмічній мові *MATLAB*:

```
Ztr = ((y(1)+y(end))/2+sum (y(2:end-1)))
Ztr = h*((y(1)+y(end))/2+sum (y(2:end-1)))
```

12.4. Метод Сімпсона

При k=2 графіки локальних інтерполянтів уявляють собою параболи, що проходять через 3 сусідні вузли табличної функції, а площа S замінюється на суму параболічних трапецій. Основою кожної з таких трапецій є 2 відрізки розбиття осі абсцис. Тому ***для використання методу Сімпсона інтервал інтегрування завжди розбивають на парну кількість відрізків***:

$$n=2m, \tag{12.9}$$

де m – кількість параболічних трапецій.

Ще одним обмеженням, що накладається на табличні значення підінтегральної функції при застосуванні методу Сімпсона, є обов‘язкове виконання умови (12.3), тобто ***інтервал інтегрування неоюхідно ділити на відрізки однакової довжини***. В цьому випадку площа однієї параболічної трапеції розраховується за формулою

$$S_{\text{кві}}=\frac{h}{3}\left(y_i+4y_{i+1}+y_{i+2}\right), \tag{12.10}$$

а значення інтеграла – за формулами

$$Z_{\text{C}}\approx\frac{h}{3}\left[y_0+y_n+2\left(y_2+y_4+\ldots+y_{n-2}\right)+4\left(y_1+y_3+\ldots+y_{n-1}\right)\right], \tag{12.11}$$

$$Z_{\text{C}}\approx\frac{b-a}{3n}[\,f(a)+f(b)+2(\,f(a+2ih)+f(a+4ih)+\ldots+f(a+(n-2)h)+$$

$$+4(\,f(a+ih)+f(a+3ih)+\ldots+f(a+(n-1)h)]. \tag{12.12}$$

Формула (12.11) програмується в *MATLAB*, одним оператором присвоєння

```
Zs = h/3*(y(1)+y(end)+2*sum (y(3:2:end-2))+4*sum (y(2:2:end-1)))
```

12.5. Оцінювання похибок чисельного інтегрування

З наведених рисунків видно, що усі розглянуті чисельні методи є приблизними, причому точність розрахунків залежить від кроку чисельного інтегрування h, порядку методу та вигляду підінтегральної функції. Похибка

методів Ньютона-Котеса приблизно визначають як інтеграл від останнього члена інтерполяційного поліному. Інформація про похибки відображена в табл. 12.1.

Таблиця 12.1

Метод	Похибки	
	Таблична функція	Аналітична функція
прямокутників	$\frac{b-a}{2}\Delta y_{\max}$	$M_1 h \frac{b-a}{2}$
трапецій	$\frac{b-a}{12}\Delta^2 y_{\max}$	$M_2 h^2 \frac{b-a}{12}$
Сімпсона	$\frac{b-a}{180}\Delta^4 y_{\max}$	$M_4 h^4 \frac{b-a}{180}$

В таблиці позначено:

M_i – максимальні похідні i-го порядку на інтервалі $[a; b]$;

$\Delta^i y_{\max}$ – максимальні значення прямих різниць i-го порядку на інтервалі $[a; b]$.

Отже, метод Сімпсона при розбитті інтервалу інтегрування на n елементарних відрізків забезпечує приблизно таку ж точність, як метод трапецій при розбитті на $2n$ відрізків або метод прямокутників при розбитті на $4n$ відрізків.

У першому наближенні для забезпечення заданої точності інтегрування є можна обирати крок інтегрування за формулою:

$$h \approx \sqrt[p]{\varepsilon}, \qquad (12.12)$$

де p=1 для метода прямокутників, p=2 для метода трапецій, p=4 для метода Сімпсона.

12.6. Методи чисельного інтегрування з автоматичним вибором кроку

Для забезпечення заданої точності інтегрування часто використовують алгоритм з автоматичним вибором кроку (АВК), у якому застосовують такий підхід: обчислюють значення інтегралу за допомогою одного з методів з деяким

початковим кроком *h*, а потім повторюють ці ж самі обчислення з удвічи меньшим кроком *h*/2. Якщо вийде так, що

$$\left|Z(h)-Z\left(h/2\right)\right|\le\varepsilon, \qquad (12.13)$$

де ε – максимально допустима похибка інтегрування, то обчислювальний процес закінчують, у протилежному випадку звертаються до подальшого роздріблення кроку.

Кінцеве значення інтеграла можна уточнити, використовуючи ***формулу Рунге***, згідно з якою похибка квадратурних формул має вигляд:

$$R_0=Ah^p. \qquad (12.14)$$

Тоді уточнене значення інтеграла після обчислення його з кроком *h*

$$Z=Z\left(h\right)+R_0=Z\left(h\right)+Ah^p,$$

а після обчислення його з половинним кроком *h*/2 –

$$Z=Z\left(h/2\right)+A\left(h/2\right)^p.$$

Прирівнюючи праві частини двох останніх виразів, отримуємо

$$Z\left(h\right)+Ah^p=Z\left(h/2\right)+A\left(h/2\right)^p,$$

або

$$Z\left(h\right)+R_0=Z\left(h/2\right)+\frac{R_0}{2^p},$$

звідкіля можна розрахувати похибку через результати двох останніх ітерацій алгоритму чисельного інтегрування з АВК:

$$R_0=\frac{2^p}{2^p-1}\left[Z\left(h/2\right)-Z\left(h\right)\right]. \qquad (12.15)$$

З урахуванням (2.15) отримують ***формулу уточнения Річардсона***:

$$Z=Z_{n-1}+\frac{Z_n-Z_{n-1}}{2^p-1}. \qquad (12.16)$$

12.7. Обчислення визначених інтегралів у пакеті MATLAB

Для обчислення визначених інтегралів від функцій, заданих аналітичними виразами, у пакеті MATLAB використовуються функції quad, quadl, quadv,

quadgk та integral, що знаходяться у папці funfun.

Функції quad, quadl та quadv мають однаковий синтаксис і вирішують одну і ту ж задачу різними методами: quad використовує квадратуру Симпсона (метод параболічних трапецій), quadl – квадратуру Лобатто (метод 8-го порядку), quadv – векторизована версія функції quad.

Звернення до цих функцій має формат:

[z, Fcnt] = quad (Fun, a, b, tol, trace)

Обов'язковими параметрами є Fun – ідентифікатор функції, a, b – межі інтегрування та z – значення інтеграла.

До основного вихідного параметру z можна додати параметр Fcnt – кількість обчислень підінтегральної функції під час чисельного інтегрування, а до основних вхідних параметрів tol – точність (максимально припустима похибка), яка за замовчанням дорівнює 10^{-6} та параметр трасування trace, який при ненульовому значенні виводить на екран послідовність ітерацій (за замовчанням trace=0). Щоб задати trace=1 задати, не змінюючи значення параметра tol за замовчанням, його подають у вигляді пустого масиву ([]), наприклад,

```
[z, k] = quad (@exp, -1, 1, [], 1)
```

В результаті виконання такого оператора отримуємо

```
 9  -1.0000000000  5.43160000e-001    0.2654022193
11  -0.4568400000  11.13680000e-001   0.9457948315
13  -0.4568400000  4.56840000e-001    0.3667183447
15   0.0000000000  4.56840000e-001    0.5790762145
17   0.4568400000  5.43160000e-001    1.1392056316
19   0.4568400000  2.71580000e-001    0.4927283573
21   0.7284200000  2.71580000e-001    0.6464772595
z =
   2.3504
k =
   21
```

Для оператора

```
[z, k] = quadl (@exp, -1, 1, [], 1)
```

отримаємо

```
18    -1.0000000000    1.00000000e+000     2.3504023875

z =
   2.3504
k =
   18
```

Дещо по-іншому звертаються до функцій quadgk та integral, які застосовують адаптивні квадратурні формули Гауса-Кронрода

[z, ErrBnd] = quadk (Fun, a, b, 'AbsTol', Atol, 'Reltol', Rtol)

Вони відрізняються тим, що межі інтеграла можуть приймати значення Inf та -Inf, а підінтегральна функція може втрачати неперервність на інтервалі ЧІ.

Для обчислення визначених інтегралів від функцій, заданих таблично, у пакеті MATLAB використовуються функції trapz та cumtrapz, що знаходяться у папці datafun.

12.8. Контрольні питання та завдання

1. Який принцип покладено в основу чисельних методів розрахунку визначених інтегралів?
2. Які методи чисельного інтегрування Ви знаєте? Як вони пов'язані з методами інтерполювання?
3. За якими формулами можна розрахувати визначені інтеграли?
4. Від чого залежить точність розрахунку визначених інтегралів? Як її забезпечити? Як її задати при використанні стандартних *MATLAB*-функцій?
5. Якими *MATLAB*-функціями зручно користуватися для розрахунку визначених інтегралів?
6. Як подивитись послідовність ітерацій при використанні стандартних *MATLAB*-функцій?
7. Що розраховують функції trapz та cumtrapz?

13. ТРИГОНОМЕТРИЧНА АПРОКСИМАЦІЯ ПЕРІОДИЧНИХ ФУНКЦІЙ

13.1. Основні визначення

Функція часу $f(t)$ називається періодичною, якщо для неї справедливі умови:

$$f(t)=f(t+mT),\ m=1, 2, 3, \dots , \tag{13.1}$$

де T – період.

Періодичні функції краще за все апроксимувати лінійною комбінацію синусоїд і косинусоїд, що породжують ***ряди Фур'є:***

$$f(t)=\frac{a_0}{2}+\sum_{k=1}^{\infty}(a_k\cos(\omega kt)+b_k\sin(\omega kt))=\frac{a_0}{2}+A_k\sum_{k=1}^{\infty}\cos(\omega kt+\varphi_k)), \tag{13.2}$$

де $\omega =2\pi/T$ – кругова частота першої гармоніки;

k – порядковий номер гармоніки;

$$A_k=\sqrt{a_k^2+b_k^2}\ , \qquad \varphi_k=arctg\frac{b_k}{a_k} \tag{13.3}$$

– амплітуди і фази відповідних гармонік.

Обмежившись у формулі (13.2) деякою кінцевою кількістю гармонік m

$$k=1, 2, \dots, m\ , \tag{13.4}$$

одержують апроксимуючий гармонічний поліном $Q_m(t)$:

$$Q_m(t)\approx\frac{a_0}{2}+\sum_{k=1}^{m}(a_k\cos(\omega kt)+b_k\sin(\omega kt))=\frac{a_0}{2}+A_k\sum_{k=1}^{m}\cos(\omega kt+\varphi_k)), \tag{13.5}$$

Визначення коефіцієнтів a_0, a_k, b_k тригонометричного полінома називають ***гармонічним аналізом*** періодичних функцій, а розрахунок тригонометричного поліному (13.5) у заданих точках при відомих коефіцієнтах окремих гармонік називають ***гармонічний синтезом***.

Коефіцієнти ряду Фур'є (13.2) визначаються за формулами:

$$a_0=\frac{2}{T}\int_0^T f(t)dt\ , \qquad a_k=\frac{2}{T}\int_0^T f(t)\cos(\omega kt)dt\ , \qquad b_k=\frac{2}{T}\int_0^T f(t)\sin(\omega kt)dt\ . \tag{13.6}$$

Множину абсолютних величин амплітуд A_k гармонічних складових періодичних полігармонічних сигналів називають ***спектром амплітуд***, сукупність фаз φ_k – ***спектром фаз*** (див. формули (13.3)), а множину величин A_k^2 – ***спектром потужності***. Аналіз частотних спектрів сигналів називають ***спектральним аналізом***.

13.2. Гармонічний аналіз та синтез періодичних функцій

Для виконання гармонічного аналізу поділимо інтервал $[0,T]$ на n рівних відрізків:

$$\Delta t = T/n, \qquad t_i = i\cdot\Delta t, \qquad y_i = f(t_i), \qquad i=0,1,2,\ldots,n \tag{13.7}$$

і використаємо для чисельного інтегрування (ЧІ) метод прямокутників:

$$Z_{np} = h\sum_{i=0}^{n-1} y_i\,,$$

З урахуванням (13.7) при ЧІ виразів (13.6) отримуємо

$$\frac{2}{T}\Delta t = \frac{2}{T}\cdot\frac{T}{n} = \frac{2}{n}, \qquad \omega k t = \frac{2\pi}{T}\cdot k\cdot\frac{iT}{n} = \frac{2\pi ki}{n}. \tag{13.8}$$

Тоді коефіцієнти тригонометричного полінома можна приблизно розрахувати за формулами

$$\begin{cases} a_0 \approx \dfrac{2}{n}\displaystyle\sum_{i=0}^{n-1} y_i, \\ a_k \approx \dfrac{2}{n}\displaystyle\sum_{i=0}^{n-1} y_i \cos\dfrac{2\pi ki}{n}, \\ b_k \approx \dfrac{2}{n}\displaystyle\sum_{i=0}^{n-1} y_i \sin\dfrac{2\pi ki}{n}. \end{cases} \tag{13.9},$$

де $k=1,2,3,\ldots,m$ – номера гармонічних складових;

m – кількість врахованих гармонік.

В обчислювальній математиці доведено, що функція $Q_m(t)$ з коефіцієнтами (13.9) апроксимує вихідну періодичну функцію (1) ***методом найменших квадратів***.

При $n=2m$ функція $Q_m(t)$ стає ***тригонометричним інтерполянтом***, тобто для того, щоб тригонометрична функція у вузлових точках мала такі ж значення, як і вихідна таблична функція, ***треба мати, як мінімум 2 точки табличної функції на 1 гармоніку***.

В MATLAB відсутні функції, що виконують гармонічний аналіз періодичних функцій у вище викладеному вигляді, але такі функції можна скласти самостійно за формулами (13.8) та (13.5) з урахуванням відсутності нульових індексів:

```
function [a0, a, b] = k_fur (y, m);
% гармонічний аналіз:
  y(end) = []; n = length(y);
  a0 = 2/n*sum(y);
  for k = 1:m
     Sa = sum(y.*cos(2*pi*k*(0:n-1)/n);
     Sb = sum(y.*sin(2*pi*k*(0:n-1)/n);
  end
  a = 2/n*Sa;
  b = 2/n*Sb
end
function Q = f_fur (t, T, a0, a, b);
% гармонічний синтез:
  f=2*pi/T; % частота основной гармоники
  m = length (a); % кол-во гармоник
  n = length (t);
  for I = 1:n
     Q(i) = a0/2+sum (a.*cos(f*[1:m]*t(i)) + b.*sin(f*[1:m]*t(i)));
  end
end
```

Наведемо приклад використання наведених функцій для апроксимації та глобальної інтерполяції заданої на періоді табличної функції:

```
clc, close all, clear all,
format compact
yt = [7.38 6.76 5.22 3.47 2.07 1.16 0.64 0.36 0.16 0.13 0.16 0.23 0.37 0.64 1.16 …
       2.08 3.48 5.22  6.76 7.38];
T = 1, w = 2*pi/T
n = length(yt)-1,
m_max = ceil(n/2)
tt = linspace(0,T,n+1);
t = linspace(0,T);
stem(tt,yt,'k--'),  grid on,  hold on
col = 'br',  i = 1;
for m = [2 m_max]
  [a0,a,b] = k_fur (yt,m)
   ya = f_fur (t,T,a0,a,b);
```

```
    plot (t,ya,col(i)),  hold on,  grid on
    i = i+1;
end
title ('Тригонометрична апроксимація')
legend ('Qa2(t)','Qi(t)', 'yt(tt)',0)
A = sqrt(a.^2+b.^2);
fi = atan2(b,a);
P = A.^2;
yy = zeros(m,100);
for k = 1:m
    yy(k,:) = A(k)*cos(k*w*t+fi(k));
end
figure
plot ([0,T],[a0/2,a0/2], t,yy)
title ('Складові сигналу Qi(t)')
figure
subplot(1,3,1), stem(1:m_max,A),  grid on,
title ('Спектр амплітуд')
subplot(1,3,2), stem(1:m_max,P),  grid on,
title ('Спектр потужностей')
subplot(1,3,3), stem(1:m_max,fi),  grid on,
title ('Спектр фаз')
```

Результати виконання цієї програми відображені на рис.13.1, 13.2, 13.3.

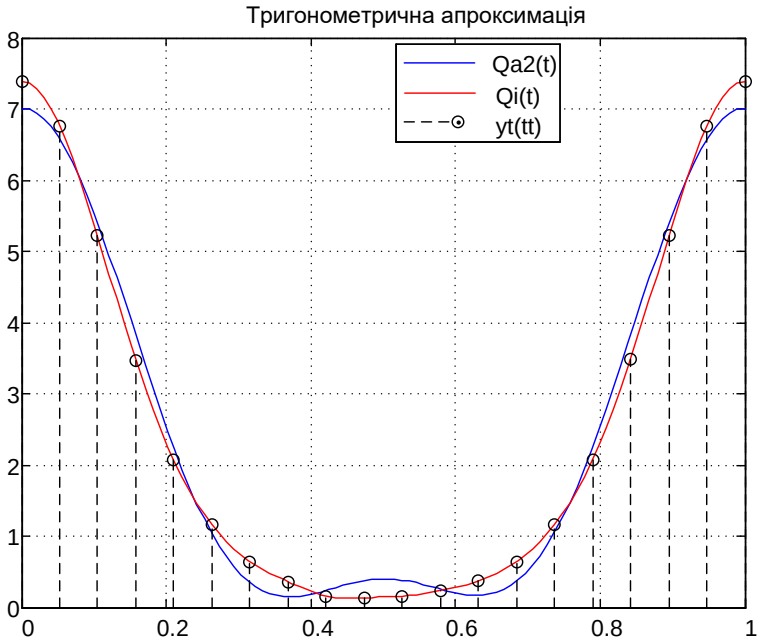


Рис. 13.1. Результати тригонометричної апроксимації досліджуваної періодичної функції

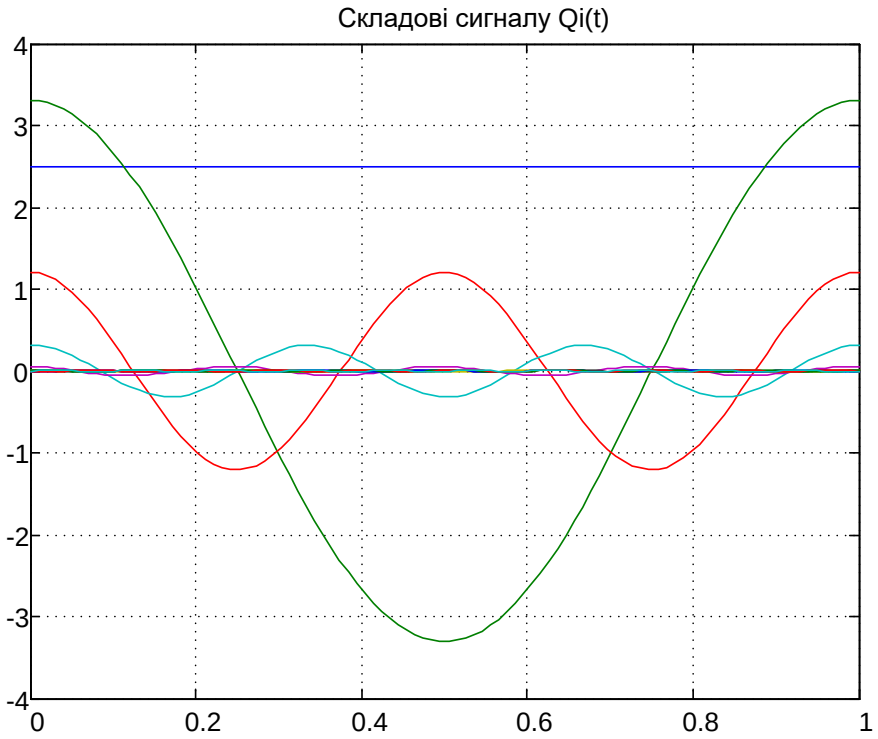


Рис. 13.2. Графіки окремих гармонічних складових тригонометричного інтерполянта досліджуваної періодичної функції

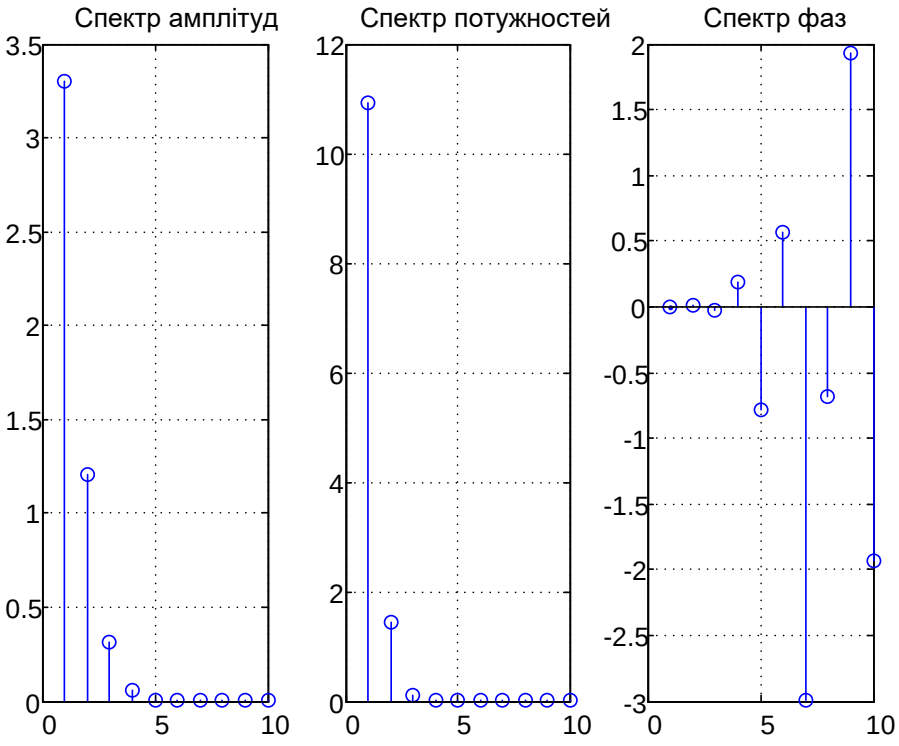


Рис. 13.3. Результати спектрального аналізу

З розглядання графіків рис. 13.2-13.3 випливає, що досліджуваний періодичний сигнал досить точно може бути апроксимованим трьома першими гармоніками.

13.3. Дискретне перетворення Фур‘є

Періодичні функції можна розкладати у ***комплексний ряд Фур‘є***:

$$f(t) = \sum_{k=-\infty}^{\infty} C_k e^{j\omega kt} \approx \sum_{k=--m}^{m} C_k e^{j\omega kt}, \qquad (13.10)$$

де C_k – комплексні коефіцієнти, що знаходять за формулою

$$C_k = |C_k| \cdot e^{j\varphi_k} = \frac{1}{T}\int_0^T f(t) \cdot e^{-jk\omega t} dt = \frac{1}{T}\int_0^T f(t) \cdot (\cos k\omega t - j \sin k\omega t) dt. \qquad (13.11)$$

Із порівняння (13.6) та (13.10) випливає ***взаємозв‘язок між комплексними та дійсними коефіцієнтами відповідних рядів Фур‘є***:

$$C_k = \begin{cases} \frac{1}{2}(a_k - jb_k) \text{ при } k = 1, 2, \ldots, \\ \frac{1}{2}a_0 \text{ при } k = 0, \\ \frac{1}{2}(a_k + jb_k) \text{ при } k = -1, -2, \ldots \end{cases} \qquad (13.12)$$

Відповідно

$$A_k = 2|C_k|, \quad k = 0, 1, 2, \ldots, m. \qquad (13.13)$$

Значення функції $f(t)$ у дискретні моменти часу можна визначити за допомогою підстановок у формулу (13.10) виразів (13.8):

$$y_i = \sum_{k=-m}^{m} C_k e^{j2\pi ki/n}, \qquad (13.14)$$

де

$$C_k = \frac{1}{n}\sum_{i=1}^{n} y_i e^{-j2\pi ki/n}. \qquad (13.15)$$

Остання формула утворює так зване ***пряме дискретне перетворення Фур‘є***.

В (13.14) кількість гармонік обирають максимально можливою, що визначається співвідношенням

$$n = 2m + 1, \tag{13.16}$$

а усі комплексні коефіцієнти переміщують в область додатних індексів:

$$y_i = \sum_{k=1}^{n} C_k e^{j2\pi ki/n}. \tag{13.17}$$

Формула (13.17) відповідає ***зворотному дискретному перетворенню Фур'є***.

Обчислення за формулами (13.15) та (13.17) потребує виконання n^2 операцій з комплексними числами. Зазвичай у досліджуваних масивах $n \geq 1000$. Тоді час, необхідний на здійснення досліджуваних перетворень стає занадто великим та унеможливлює обробку сигналів у реальному часі.

У 60-х роках Кулі та Тьюкі запропонували метод розрахунку коефіцієнтів Фур'є, який дозволив зменшити кількість операцій до $n \cdot \log_2(n)$. Він отримав назву ***Швидке Перетворення Фур'є (англ. FFT – Fast Fourier Transformation)***. Ефективність цього алгоритму настільки велика, що його застосовано в усіх програмних продуктах, які здійснюють спектральний аналіз та синтез виміряних полігармонічних сигналів.

В пакеті *MATLAB* функції fft і ifft, що здійснюють пряме та зворотне швидке перетворення Фур'є, знаходяться у папці datafun. Вони виконують операції

```
Cf=fft(y),   y=ifft(Cf)
```

за формулами

$$C_{fk} = \sum_{i=1}^{n} y_i e^{-j2\pi ki/n}, \qquad y_i = \frac{1}{n} \sum_{k=1}^{n} C_{fk} e^{j2\pi ki/n}. \tag{13.18}$$

Як бачимо, формули (13.19) відрізняються тим, що коефіцієнт $1/n$ переміщено з формули прямого перетворення у формулу зворотного перетворення Фур'є, тобто вектори комплексних коефіцієнтів $\mathbf{C}_f$ та $\mathbf{C}$ пов'язані між собою залежністю

$$\mathbf{C}=\mathbf{C}_f/n. \qquad (13.19)$$

Отримані в *MATLAB* за допомогою операції C=fft (y)/n комплексні коефіцієнти розташовуються у векторі з додатними індексами у такій послідовності:

$$\mathbf{C}=[C_0, C_1, C_2, ..., C_m, C_{-m}, ..., C_{-2}, C_{-1},]. \qquad (13.20)$$

Тому дійсні коефіцієнти Фур‘є з урахуванням виразів (13.12), (13.19), способу розташування коефіцієнтів (13.20) і відсутності нульового індексу в *MATLAB* можна визначити за допомогою такої *m*-функції:

```
function [a0,a,b] = k_furd (y,m);
  y(end) = []; n = length(y);
  C=fft (y) / n
  a0 = 2*C(1);
  a =  2*real (C(2:m+1));
  b = -2*imag (C(2:m+1));
end
```

Результати використання цієї функції та наведеної раніше функції k_fur будуть повністю тотожними.

Взагалі швидкі дискретні перетворення Фур‘є використовують для спектрального аналізу великих за обсягом експериментальних даних, в яких важко визначити навіть період основної гармоніки із-за випадкових перешкод.

Для прикладу сформуємо сигнал, що складається із синусоїди частотою 50 Гц з амплітудою 220 і синусоїди частотою 120 Гц з амплітудою 50, спотворимо його випадковим шумом з нормальним розподіленням та побудуємо спектр амплітуд, використовуючи функцію fft і рівняння (13.13):

```
clc, clear all, close all
fd = 1000;                  % Частота дискретизації
Td = 1/fd;                  % Період дискретизації
n = 1024;                   % Кількість вимірів
t = (0:n-1)*Td;             % Вектор часу
% Формування сигналу із 2-х синусоїдальних гармонік
x = 220*sin(2*pi*50*t) + 50*sin(2*pi*120*t);
y = x + 100*randn(size(t));   % Зашумлення основного сигналу
plot(t(1:1000),y(1:1000)), grid on
title('Виміряний сигнал')
xlabel('t,s')
```

```
    % Спектральний аналіз
    C = fft(y)/n;          % Комплексні коефіцієнти ряду Фурє
    f = fd/2*linspace(0,1,n/2+1); % Діапазон частот для спектрального аналізу
A = 2*abs(C(1:n/2+1));  % Амплітуди гармонік виміряного сигналу
    figure
    plot(f,A), grid on
    title('Спектр амплітуд виміряного сигналу')
    xlabel('Частота, Гц')
    ylabel('A(f)')
```

Графічні результати виконання програми наведені на рис. 13.4, 13.5.

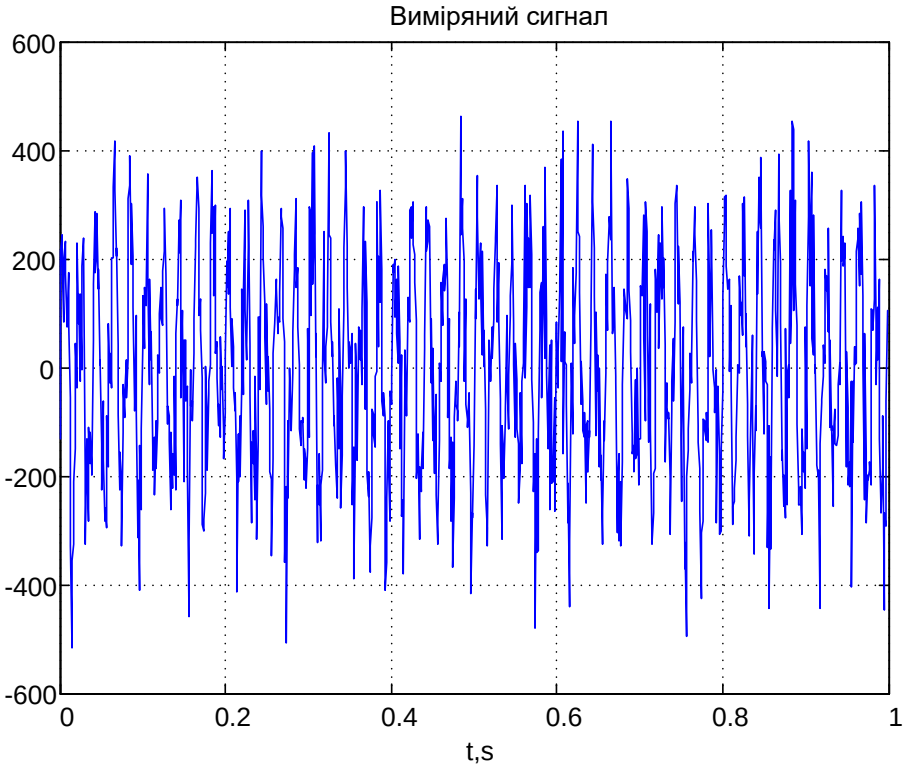


Рис. 13.4. Виміряний сигнал з наявністю шумів

Дивлячись на вихідний сигнал у функції часу (рис. 13.4), не можливо ідентифікувати його частотні компоненти. Це можна зробити у частотнім діапазоні шляхом використання швидкого дискретного перетворення Фур'є (рис.13.5).

Основною причиною не зовсім точного визначення амплітуд є наявність перешкод у вихідному сигналі. Інша причина полягає в тому, що сигнал має обмежену довжину. Збільшення кількості вимірюваних значень функцій зазвичай підвищує точність спектрального аналізу.

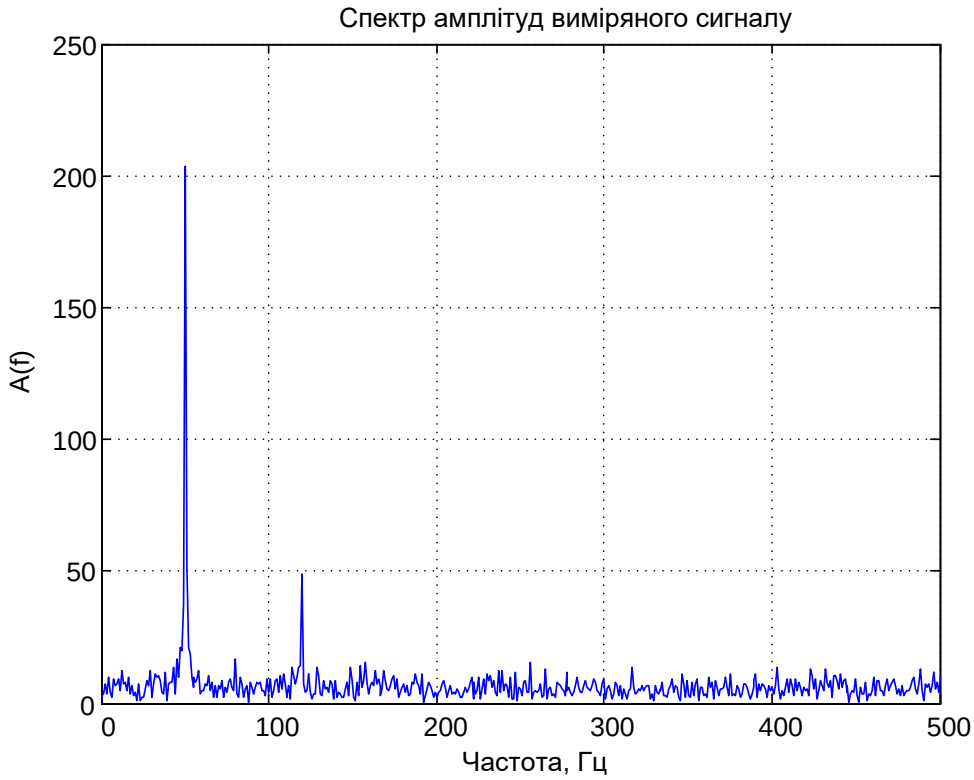


Рис. 13.5. Амплітудний спектр виміряного сигналу

Історія розвитку спектрального аналізу настільки цікава, а ***сфери його практичного застосування*** настільки різноманітні, що я раджу усім прочитати статтю, наведену в додатку А.

13.4. Контрольні питання та завдання

1. Що таке гармонічний аналіз, гармонічний синтез, спектральний аналіз, дискретне перетворення Фур‘є?
2. Як пов‘язані між собою амплітуди гармонік та коефіцієнти комплексного ряду Фур‘є?
3. За якими формулами можна розрахувати амплітуди та фази гармонічних складових?
4. Як виконати гармонічний аналіз за допомогою дискретного перетворення Фур‘є?
5. Скільки точок табличної функцію повинно приходитися на 1 гармоніку, щоб тригонометрична функція стала глобальним інтерполянтом?
6. З якою метою виконують спектральний аналіз?

14. ЧИСЛОВЕ РОЗВ‘ЯЗАННЯ ДИФЕРЕНЦІЙНИХ РІВНЯНЬ

14.1. Загальні поняття

Рішення диференційних рівнянь складає фундамент математичного моделювання різних пристроїв, процесів, систем. В електротехніці та похідних від неї дисциплінах рішення диференційних рівнянь використовують при розрахунках перехідних процесів.

Диференційні рівняння (ДР) розподіляються на ***звичайні диференційні рівняння*** (ЗДР), які отримують одну незалежну змінну і похідні від цієї змінної, і *ДУ* ***диференційні рівняння у часткових похідних***, що складаються з похідних від декількох змінних. ЗДР поділяються на ***звичайні диференційні рівняння*** **з початковими умовами** та ***звичайні диференційні рівняння з граничними умовами***.

Усі перелічені вище види ДР розв‘язуться суттєво різними методами. В даному посібнику розглядаються тільки так звана ***задача Коші***: пошук часткових рішень ЗДР та їх систем з початковими умовами.

Звичайне диференційне рівняння n-го порядку має вигляд:

$$F(t,y,y',y'',...,y^{(n)})=0, \qquad (14.1)$$

де t – незалежна змінна;

$y(t)$ – невідома функція незалежної змінної;

$y'(t)=\frac{dy(t)}{dt}$, $y''(t)=\frac{d_2y(t)}{dt^2}$, $y^{(n)}(t)=\frac{d_ny(t)}{dt^n}$ – її похідні.

Для знаходження ***частинного рішення*** рівняння (14.1) повинні бути відомі ***n початкових умов***:

$$y(t_0)=y_0,\ y'(t_0)=y'_0,...,y^{(n-1)}(t_0)=y_0^{(n-1)}. \qquad (14.2)$$

14.2. Огляд чисельних методів розв‘язання диференційних рівнянь

Числове розв‘язання диференційного рівняння полягає у визначенні таблиці значень $y_i(t_i)$ $(i=0,1,2,...,k)$ на деякому інтервалі $[t_0,t_k]$. Різницю між двома сусідніми табличними значеннями аргументу називають ***кроком інтегрування***:

$$h=t_{i+1}-t_i. \tag{14.3}$$

Для використання більшості з чисельних методів необхідно початкове диференційне рівняння *n*-го порядку (14.1) перетворити у ***систему*** *n* ***диференційних рівнянь першого порядку у нормальній формі Коші***, тобто рівнянь, розв‘язаних відносно похідних:

$$\begin{cases} x_1' = f_1(t,x_1,x_2,...,x_n), \\ x_2' = f_2(t,x_1,x_2,...,x_n), \\ \quad\quad\quad \dots \\ x_n' = f_n(t,x_1,x_2,...,x_n), \end{cases} \tag{14.4}$$

$$x_1(t_0)=x_{10}, x_2(t_0)=x_{20},...,x_n(t_0)=x_{n0}, \tag{14.5}$$

або у векторній формі:

$$\mathbf{X}'=\mathbf{F}(t,\mathbf{X}),\mathbf{X}(t_0)=\mathbf{X}_0. \tag{14.6}$$

Змінні стану $x_1,x_2,...,x_n$ та їх початкові умови у процесі перетворення із форми (14.1) у форму (14.4) однозначно пов’язуються з невідомою функцією x та її похідними.

Присутність незалежної змінної у правих частинах рівнянь (14.4) не є обов‘язковою.

Чисельний розв‘язок системи (14.4) на інтервалі $[t_0,t_k]$ є вектор незалежних змінних та відповідна матриця нев ідомих залежних змінних

$$\mathbf{t}=\begin{bmatrix} t_0 \\ t_1 \\ t_2 \\ \dots \\ t_k \end{bmatrix}, \qquad \mathbf{x}=\begin{bmatrix} x_{01} & x_{02} & \dots & x_{0n} \\ x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \dots & \dots & \dots & \dots \\ x_{k1} & x_{k2} & \dots & x_{kn} \end{bmatrix}. \tag{14.7}$$

Елементи матриці залежних змінних знаходять по рядкам, починаючи з рядка початкових умов Кожний рядок матриці уявляє значення усіх невідомих змінних у певний момент часу, а кожен стовбець – значення однієї невідомої змінної на усьому інтервалі зміни незалежної змінної.

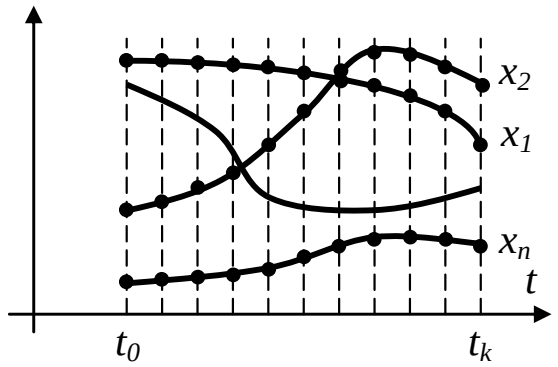


Рис. 12.1. Розв‘язок системи ДР

Чисельну інформацію (14.7) використовують для побудови графіків функцій $y_1(t), y_2(t), ..., y_n(t)$. При цьому аналітичний розв‘язок залишається невідомим. Якщо незалежною змінною є час, то отримані графіки називають ***графіками перехідних процесів*** (див. рис. 14.1).

Точність розв‘язку залежить від кроку ЧІ, тобто відстані між сусідніми інтервалами часу.

Серед чисельних методів розв‘язання ДР розрізняють одноступінчаті та багатоступінчаті.

В одноступінчатих методах для розрахунку значень змінних стану у точці $t + h$, тобто. $x_i(t + h)$, необхідна інформація про одне попереднє значення розв‘язку $x_i(t)$. Одними з найпоширеніших одноступінчатих методів є ***методи Рунге-Кутта***. Порядок методів Рунге-Кутта визначається кількістю розрахунків значень похідних на одному кроці ЧІ. Методи Рунге-Кутта можна використовувати як з постійним кроком, так і з автоматичним вибором кроку (АВК), який обирається з умов забезпечення заданої точності розв‘язку. Підвищити точність можна використанням комбінованих методів Рунге-Кутти різних порядків, які дають можливість визначити приблизно величину похибки і компенсувати її.

Багатоступінчаті методи для виконання одного кроку ЧІ потребують інформації про значення невідомих змінних у декількох попередніх точках часу, кількість яких визначає порядок методу. Серед багатокрокових методів

виділяють велику групу під загальною назвою ***«методи прогнозу і корекції» (Predictor-Corrector Methods).*** Найбільш відомими серед них є ***методи Адамса, Бешфорта, Мілна, Гіра, Розенброка***. Оскільки на першому кроці ЧІ відомими є тільки початкові умови, то методи прогнозу і корекції «запускаються до роботи» методами Рунге-Кутта відповідного порядку. При використанні методів прогнозу і корекції знайдений при першому наближенні розв‘язок потім уточнюється за допомогою ітераційних алгоритмів, в яких кількість ітерацій залежить від заданої точності розв‘язку.

При виборі певного методу розв‘язання ДР слід виходити з того, що одноступінчаті методи потребують менше оперативної пам‘яті, але працюють при заданій точності повільніше, ніж багатоступінчаті методи такого ж порядку. Також треба мати на увазі, що багатоступінчаті методи більш пристосовані для розв‘язання так званих ***жорстких (stiff) ДР***, для яких характерно сполучення швидкої та повільної динаміки, тобто відношення найбільшої сталої часу до найменшої є великим числом.

14.3. Методи Рунге-Кутта

До числа найбільш розповсюджених чисельних методів рішення диференційних рівнянь належать ***методи Рунге-Кутта (Runge-Kutta methods).***

Вони узгоджуються з розкладом функції $x(t)$ у ряд Тейлора навколо точки x_i аж до членів, що містять у собі h^p:

$$x_{i+1} \approx x_i + h \cdot x'_i + \frac{h^2}{2!} \cdot x''_i + \ldots + \frac{h^p}{p!} \cdot x^{(p)}_i . \qquad (14.8)$$

Показник степені p при h в останньому доданку ряду Тейлора визначає порядок методу.

Метод Рунге-Кутти першого порядку називають ***методом Ейлера (Euler’s method)***, другого порядку – ***модифікованим методом Ейлера (improved Euler’s formula)***, або ***методом Ейлера-Коші,*** або ***Heun’s method***, п‘ятого порядку – методом ***Дормана-Прінса (Dormand-Prince)***. Коли мова йде

про метод Рунге-Кутта без визначення його порядку, то найчастіше мають на увазі метод ***Рунге-Кутти 4-го порядку***, який застосовують найчастіше, завдяки досягненню компромісу між складністю, швидкодією та точністю методу.

Відповідно до ***метода Ейлера*** один крок рішення системи диференційних рівнянь (14.4) з початковими умовами (14.5) виконується за формулою:

$$\mathbf{X}(t+h)=\mathbf{X}(t)+h\cdot\mathbf{F}(t,\mathbf{X}(t))\,. \tag{14.9}$$

Метод Ейлера-Коши потребує обчислення вектору похідних (правих частин диференційних рівнянь) $\mathbf{F}(t,\ \mathbf{Y})$ у двох точках:

$$\begin{cases}\mathbf{K}_1=\mathbf{F}(t,\mathbf{X}(t)),\\ \mathbf{K}_2=\mathbf{F}(t+h,\ \ \mathbf{X}(t)+\mathbf{K}_1 h);\end{cases} \tag{14.10}$$

$$\mathbf{X}(t+h)=\mathbf{X}(t)+\frac{h}{2}\cdot(\mathbf{K}_1+\mathbf{K}_2). \tag{14.11}$$

Відповідно при використанні метода Рунге-Кутта четвертого порядку вектор похідних на кожному кроці чисельного інтегрування обчислюється чотири рази :

$$\begin{cases}\mathbf{K}_1=\mathbf{F}(x,\mathbf{X}(t)),\\ \mathbf{K}_2=\mathbf{F}(t+\dfrac{h}{2},\mathbf{X}(t)+\dfrac{h}{2}\mathbf{K}_1),\\ \mathbf{K}_3=\mathbf{F}(t+\dfrac{h}{2},\mathbf{X}(t)+\dfrac{h}{2}\mathbf{K}_2),\\ \mathbf{K}_4=\mathbf{F}(t+h,\mathbf{X}(t)+h\mathbf{K}_3);\end{cases} \tag{14.13}$$

$$\mathbf{X}(t+h)=\mathbf{X}(t)+\frac{h}{6}(\mathbf{K}_1+2\mathbf{K}_2+2\mathbf{K}_3+\mathbf{K}_4)\,. \tag{14.14}$$

Обчислення за наведеними раніше формулами продовжують до того часу, доки не буде досягнуто кінець інтервалу $[t_0,t_k]$.

Похибка методів Рунге-Кутта визначається виразом:

$$\varepsilon\approx k\cdot h^p.$$

Значення коефіцієнта k залежить від системи, що розв‘язується.

14.4. Розв‘язання диференційних рівнянь з використанням алгоритмічної мови пакета *MATLAB*

Серед методів розв‘язання ДР з фіксованим кроком в *Simulink* пропонуються методи Рунге-Кутта першого-п‘ятого порядків, а саме:

ode5 – метод Рунге-Кутта 5-го порядку (*Dormand-Prince formula*);

ode4 – метод Рунге-Кутта 4-го порядку (*fourthe-order Runge-Kutta formula*);

ode3 – метод Рунге-Кутта 3-го порядку (*Bogacki-Shampine formula*);

ode2 – модифікований метод Ейлера (*improved Euler’s formula / Heun’s method*);

ode1 – метод Ейлера (*Euler’s method*).

Серед методів розв‘язання ДР з заданою точністю, в *Simulink* і в *MATLAB* пропонуються комбіновані методи Рунге-Кутти та деякі методи прогнозу та корекцій, а саме:

ode45 – комбінований метод Рунге-Кутта (4-5)-го порядку з автоматичним вибором кроку, призначений для розв‘язання нежорстких ДР;

ode23 – комбінований метод Рунге-Кутта (2-3)-го порядку з автоматичним вибором кроку, призначений для розв‘язання нежорстких та слабко жорстких ДР;

ode113 – багатокроковий метод Адамса-Моултона-Бешфорта (метод прогнозу та корекцій змінного порядку), призначений для розв‘язання нежорстких ДР; може бути більш ефективним, ніж ode45, при високій точності;

ode15s – багатокроковий метод (1-5)-го порядку (за замовчанням 5-го), що базується на формулах чисельного диференціювання (*NDFs*), але може використовувати і формули зворотного диференціювання (*BDFs*) у відповідності з методом Гіра, спрямований на розв‘язання жорстких ДР;

ode23s – однокроковий метод, що базується на модифікованій формулі Розенброка 2-го порядку; для деяких жорстких ДР виявляється при невисокій точності виявляється більш ефективним, ніж ode15s;

ode23t – метод трапецій з використанням „вільного" інтерполянта для розв'язання помірно жорстких ДР;

ode23tb – комбінований метод, що використовує на першому етапі формулу трапецій, а на другому – зворотну диференційну формулу другого порядку; як і метод ode23s, може бути при невисокій точності більш ефективним, ніж *ode*15*s*, для розв'язання жорстких ДР.

Функції ode1, …, ode5 можна обирати тільки при розрахунку перехідних процесів у середовищі *Simulink*, а усі інші із вище перелічених – як у середовищі *Simulink*, так і у програмах, написаних алгоритмічною мовою пакета *MatLab.*

Звернення до усіх цих функцій має однаковий формат (за виключенням імені функції). Розглянемо варіанти їх використання на прикладі функції ode23. У мінімальній конфігурації звернення до цієї функції виглядає так:

[t,X] = ode23 (odeFun, Tspan, X0),

де

[t,X] – вектори, які містять рішення (t – вектор-стовбець часу, X – матриця вихідних змінних, в якій кількість рядків дорівнює кількості елементів у векторі часу, а кількість стовпчиків – порядку системи диференційних рівнянь n = length(X0), тобто кількості невідомих змінних); *i*-й рядок матриці X(i,:) містить у собі значення усіх невідомих змінних в *i*-й момент часу (i = 1:length(t)), а j-й стовпчик матриці X(:,j) містить у собі значення однієї змінної у кожний момент часу (j=1:n);

Tspan = [t0 tf] – інтервал часу, який визначається початковим t0 та кінцевим tf часами інтегрування (t0 – необов'язковий параметр, який за замовченням

дорівнює 0, тоді Tspan=tf) або монотонно зростаючий чи монотонно спадаючий вектор часу: Tspan = [t0 t1 t2 … tf];

X0 – вектор початкових умов;

odeFun – дескриптор *m*-функції (*function handle*), у якій міститься опис системи диференційних рівнянь, тобто формули для розрахунку вектору перших похідних від невідомих змінних, або, як їх називають, функції правих частин системи диференційних рівнянь у нормальні й формі Коші (див. рівняння (14.4)) у заданий момент часу *t*.

Дескриптор функції конструюється символом @, за яким прямує ім‘я функції: @FunName. Замість нього можна використати ім‘я функції у вигляді рядка символів ‘FunName’.

Обов‘язкова частина функції правих частин має такий формат:

```
function  dX=MainRight (t,X)
  dX(1)=…;
  dX(2)=…;
  …
  dX(n)=…;
  dX=dX(:);
end
```

Останній рядок функції витягає вектор похідних у стовбець. Час t у цій функції є скалярною змінною, а X – вектор невідомих змінних саме у цей момент часу. Дізнатися про додаткові параметри методу, що визначаються за замовчанням, та про можливі значення цих параметрів можна за допомогою функції odeset без параметрів:

```
» odeset
AbsTol: [ positive scalar or vector {1e-6} ]
BDF: [ on | {off} ]
Events: [ on | {off} ]
InitialStep: [ positive scalar ]
Jacobian: [ on | {off} ]
JConstant: [ on | {off} ]
JPattern: [ on | {off} ]
Mass: [ {none} | M | M(t) | M(t,y) ]
MassSingular: [ yes | no | {maybe} ]
MaxOrder: [ 1 | 2 | 3 | 4 | {5} ]
```

```
MaxStep: [ positive scalar ]
NormControl: [ on | {off} ]
OutputFcn: [ string ]
OutputSel: [ vector of integers ]
Refine: [ positive integer ]
RelTol: [ positive scalar {1e-3} ]
Stats: [ on | {off} ]
Vectorized: [ on | {off} ]
```

Значення опцій за замовчанням подані у фігурних дужках. Найбільший інтерес мають параметри AbsTol (абсолютна похибка), InitialStep (початковий крок), MaxStep (максимальний крок), RelTol (відносна похибка).

Змінити значення опцій можна за допомогою присвоєння вихідному параметру options значення функції odeset з чергуванням вхідних параметрів «ім‘я властивості» та «значення властивості» і доповнення виклику функції ode... сформованим у такий спосіб вхідним параметром. Наприклад, наведена нижче послідовність операторів

```
opt = odeset('RelTol',1e-4,'AbsTol',[1e-5 1e-5 1e-5], ‘MaxStep’, 0.01);
[t,y]=ode113(@mydiffur, 1.2, zeros (1,3), opt);
```

розв‘язує систему ДР 3-го порядку, описану у функції mydiffur, з нульовими початковими умовами на інтервалі часу $t \in [0.1,2]$ методом Адамса-Моултона-Бешфорта з відносною точністю 10^{-4}, абсолютною точністю по кожній змінній стану 10^{-5} і максимальним кроком 0.01.

Інформацію про поточні значення опцій після їх зміни можна побачити за допомогою оператора odeget, наприклад:

```
» odeget(opt,'RelTol')
ans =
   0.0001
```

14.5. Приклад розрахунку перехідних процесів в електричному лінійному колі в пакеті *MATLAB*

14.5.1. Умова задачі

Математичний опис лінійних електричних кіл створюють, використовуючи закони Ома та Кірхгофа з урахуванням тієї обставини, що напруги і струми на резисторі індуктивності та конденсаторі пов‘язані між собою співвідношеннями:

$$U_R(t) = RI_R(t), \qquad U_L(t) = L\frac{dI_L(t)}{dt}, \qquad I_c(t) = c\frac{dU_c(t)}{dt}. \tag{14.15}$$

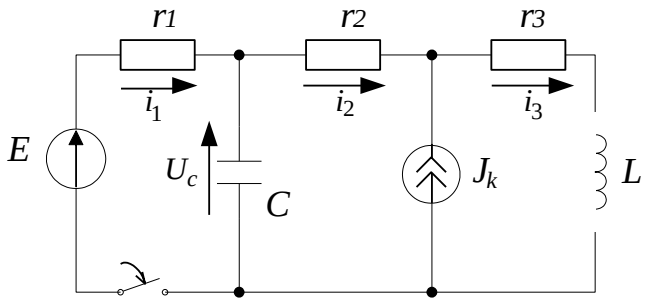


Рис. 14.2. Схема електричного кола

Як приклад, розглянемо розгалужену схему, зображену на рис. 14.2, для якої треба розрахувати перехідні процеси, тобто знайти $i_1(t)$, $i_2(t)$, $i_3(t)$, $U_C(t)$ при замиканні ключа при таких параметрах: E = 100 В, J_k = 5 А, C = 200 мкФ, L = 0,3 Гн, r_1 = 20 Ом, r_2 = 50 Ом, r_3 = 70 Ом.

14.5.2. Математичний опис електричного кола

Математичний опис схеми при замкненому стані ключа, згідно з законами Кірхгофа, має вигляд:

$$E = i_1 r_1 - U_c, \tag{14.16}$$

$$E = i_1 r_1 + i_2 r_2 + i_3 r_3 + L\frac{di_3}{dt}, \tag{14.17}$$

$$i_1 + C\frac{dU_c}{dt} = i_2, \tag{14.18}$$

$$i_2 + J_k = i_3. \tag{14.19}$$

Рівняння (14.17), (14.18) – диференційні; (14.16), (14.19) – алгебраїчні рівняння, які часто називають рівняннями зв’язку.

Для використання чисельних методів розв‘язання ДР перетворимо їх у нормальну форму Коші:

$$\begin{cases} \dfrac{dU_C}{dt} = (i_2 - i_1)/C, \\ \dfrac{di_3}{dt} = (E - i_1 r_1 - i_2 r_2 - i_3 r_3)/L. \end{cases} \tag{14.20}$$

З цих рівнянь видно, що досліджувана динамічна система описується системою двох диференційних рівнянь першого порядку.

Об‘єднаємо невідомі змінні, від яких визначено похідні, в один вектор:

$$\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} U_c \\ i_3 \end{bmatrix} \tag{14.21}$$

Ці сигнали утворюють вектор змінних стану. Їх кількість визначає порядок системи ДР.

З рівнянь зв‘язку визначаємо інші невідомі сигнали через змінні стану:

$$\begin{cases} i_1 = (E + U_C)/r_1, \\ i_2 = i_3 - J_k. \end{cases} \tag{14.22}$$

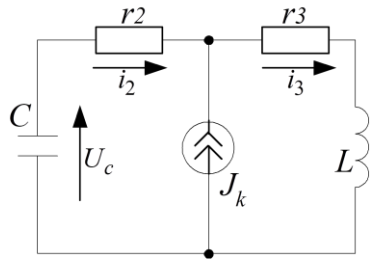


Рис. 14.3. Електричне коло у початковому стані

Розрахуємо початкові умови для заданої схеми, які збігаються з усталеними значеннями відповідних сигналів при розімкненому стані ключа. Така схема зображена на рис. 14.3.

В усталеному режимі усі похідні дорівнюють 0, звідси

$$\begin{cases} i_3(0) = J_k, \\ U_C(0) = -i_3 r_3 = -J_k r_3. \end{cases} \tag{14.23}$$

14.5.3. Розрахунок перехідних процесів програмним методом

Для розрахунку перехідних процесів програмним методом за допомогою *m*-функцій, розглянутих у попередньому підрозділі, розробимо на основі виразів (14.20)-(14.22) функцію, що буде обчислювати праві частини ДР (14.20), тобто вектор похідних від змінних стану. При цьому треба дотримуватися таких правил:

- імена змінних, значення яких відомі і будуть визначені у головній програмі, опишемо як глобальні, що дасть можливість використовувати їх у даній функції;
- присвоїмо невідомим змінним у скалярному вигляді значення індексованих змінних стану (див. (14.21));
- напишемо рівняння зв‘язку (14.22) у скалярній формі;
- напишемо диференційні рівняння (14.21);
- перетворимо вектор похідних у стовбець.

Після кожного з операторів присвоєння ставимо «крапку із зап‘ятою, щоб запобігти виведення зайвої проміжної інформації на екран.

```
% Математичний опис електричного кола
function dx=right1(t,x)
  global E Jk R L C  % Глобальні зміні
  % Вираз невідомих змінних через змінні стану в скалярній формі
  Uc=x(1);  i3=x(2);
  % Визначення допоміжних змінних із рівнянь зв‘язку в скал. формі
  i1=(E+y(1))/R(1);   i2=y(2)-Jk;
  % Диференційні рівняння в нормальній формі Коші
  dx(1)=(i2-i1)/C;
  dx(2)=(E-i1*R(1)-i2*R(2)-y(2)*R(3))/L;
  dx=dx(:);    %Витягування вектору похідних у стовбець
end
```

Створену функцію слід зберегти у файлі, ім‘я якого співпадає з ім‘ям функції, тобто right1.m.

Після цього розробляємо головну програму (*script file*). При цьому будемо дотримуватися такої послідовності:

- імена змінних, значення яких потребують як головна програма, так і функція правих частин, описуємо як глобальні, після чого визначаємо вихідні данні та розраховуємо початкові умови;
- задаємось інтервалом часу розрахунку, який може бути уточнений після декількох спроб;
- розв‘язуємо ДР, визначені у функції одним із методів, описаних у попередньому підрозділі; у такий спосіб визначаємо вектор-стовбець часу та

матрицю змінних стану;

- присвоїмо векторам невідомих змінних ДР значення відповідних стовбців матриці змінних стану;
- розрахуємо з рівнянь зв‘язку вектори допоміжних змінних;
- на підставі отриманих чисельних даних побудуємо графіки перехідних процесів.

Після операторів, що визначають вихідні дані, «крапку із зап‘ятою» можна не ставити, а після операторів, що розраховують перехідні процеси, «крапку із зап‘ятою» краще поставити, щоб запобігти виведенню на екран значних масивів даних, які використовуються для побудови графіків.

```
% Текст головної програми
global E Jk R L C      % Глобальні змінні
E=100,  Jk=5,  C=200e-6,  L=0.3,  R=[20 50 70]     % Вихідні данні
X0=[-Jk*R(3) Jk]      % Початкові умови
tk=0.1       % Час перебігу перехідних процесів
 [t,X]=ode23 (@right1,  tk,  X0);     % Розв‘язання  системи ДР
Uc=X(:,1);   I3=X(:,2);     % Невідомі змінні ДР
I1=(X(:,1)+E)/R(1);   I2=X(:,2)-Jk;   Ic=I2-I1;    % Допоміжні змінні
% Побудова графиків перехідних процесів
figure, subplot(121), plot(t,[I1 I2 I3 Ic]), grid on, legend('i1','i2','i3','ic');
subplot(122), plot(t,Uc), grid on
```

Результат роботи програми наведено на рис. 14.4.

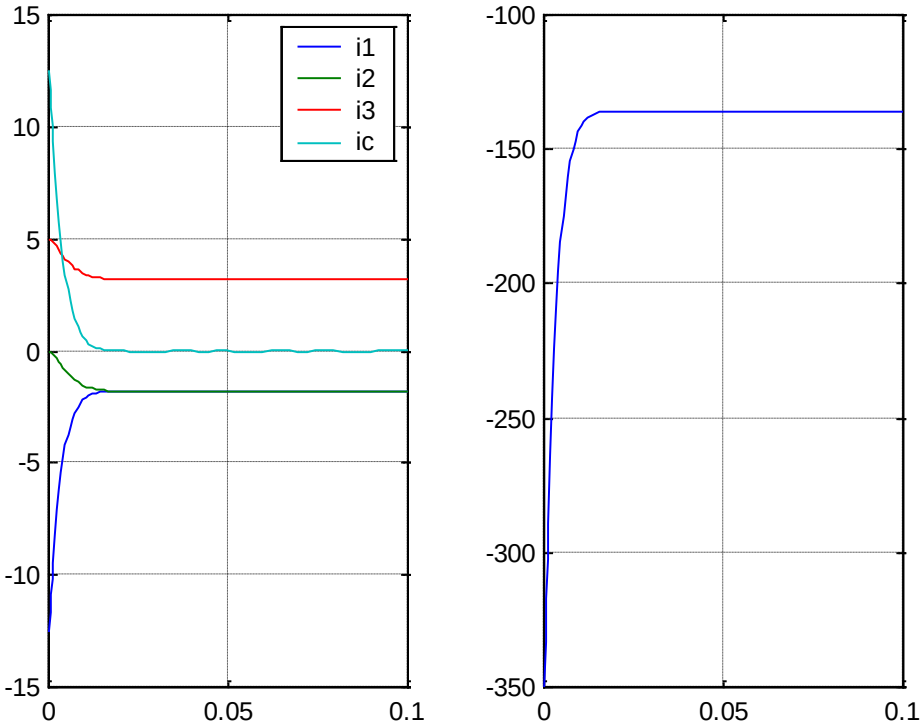


Рис. 14.4. Графіки розрахованих перехідних процесів

З розглядання цих графіків видно, що інтервал часу, на якому здійснюється розрахунок перехідних процесів, можна зменшити принаймні до 0.025с.

Якщо ЕРС та/або струми джерел змінюються у часі, то закон зміни цих сигналів треба описати у функції правих частин у скалярному вигляді, а в головній програмі – у векторному.

Наприклад, якщо

$$E(t) = E_m \sin(\omega t), \quad E_m = 220V, \quad \omega = 2\pi f, \quad f = 50Hz,$$

то у головній програмі зміняться перші 2 рядки

```
global Em  w Jk R L C      % Глобальні змінні
Em=220,  f=50,  w=2*pi*f, Jk=5,  C=200e-6,  L=0.3,  R=[20 50 70]
```

та перед розрахунком допоміжних змінних приєднається оператор

```
E=Em*sin (w*t);
```

Точно такий же оператор додаємо і у функцію правих частин ДР.

Для формування періодичних сигналів прямокутної та трикутної форми, у тому числі і пилкоподібної, у *Signal Processing Toolbox* існують функції square та sawtooth.

Функція square(t) генерує прямокутний періодичний сигнал одиничної амплітуди з періодом 2*pi. Необов‘язковий параметр duty cycle визначає у процентах долю періоду, коли вихідний сигнал має додатне значення. За замовченням значення цього параметру становить 50%.

Функція sawtooth(t) генерує трикутний періодичний сигнал одиничної амплітуди з періодом 2*pi. Необов‘язковий параметр *width* визначає долю періоду, коли вихідний сигнал досягає максимального значення. За замовченням значення цього параметру становить 1. При цьому вихідний сигнал має пилкоподібну форму з крутим заднім фронтом. При width = 0 він стає пилкоподібним із крутим переднім фронтом, а при width = 0.5 – набуває форму трикутної симетричної хвилі.

Для генерації перших 10 періодів пилкоподібного та прямокутного періодичних сигналів з частотою 50 Hz із кроком 1/10 kHz, можна скласти таку програму:

```
fs = 10000;
t = 0:1/fs:0.2;
x1 = sawtooth(2*pi*50*t);
x2 = square(2*pi*50*t);
subplot(211),plot(t,x1), axis([0 0.2 -1.2 1.2])
xlabel('Time (sec)');ylabel('Amplitude'); title('Sawtooth Periodic Wave')
subplot(212),plot(t,x2), axis([0 0.2 -1.2 1.2])
xlabel('Time (sec)');ylabel('Amplitude'); title('Square Periodic Wave')
```

Результати її виконання відображені на рис. 14.5.

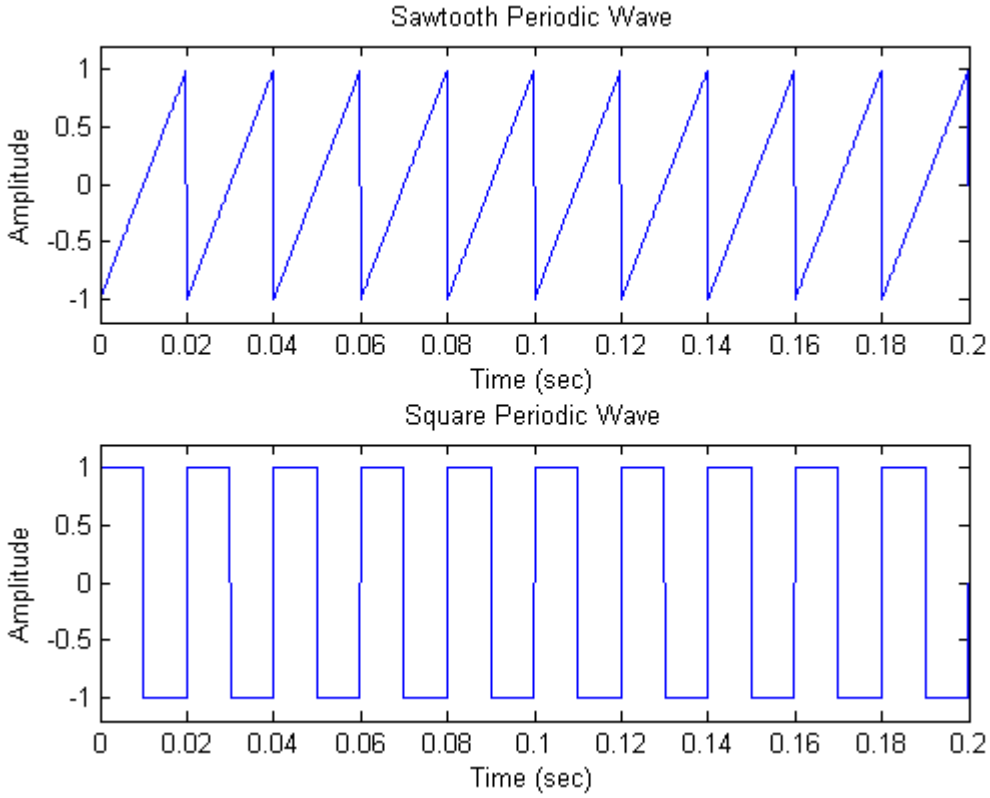


Рис. 14.5. Графіки типових періодичних сигналів

Для формування одиночних (неперіодичних) трикутних, прямокутних та Гаусовських імпульсів у *Signal Processing Toolbox* існують функції tripuls та rectpuls відповідно**.**

Імпульси формуються симетричними відносно точки $t = 0$. Ширина імпульсів за замовчанням дорівнює 1. Її можна змінити додатковим параметром.

Зразок використання цих функцій наведено нижче:

```
fs = 10000; Ts = 1/fs;
t = -0.1:Ts:0.1;
x1 = tripuls (t, 20e-3);
x2 = rectpuls (t, 20e-3);
subplot (211), plot (t, x1), ylim ([-0.2 1.2])
xlabel('Time (sec)'); ylabel('Amplitude'); title('Triangular Aperiodic Pulse')
subplot (212), plot (t, x2), ylim ([-0.2 1.2])
xlabel ('Time (sec)'); ylabel ('Amplitude'); title ('Rect Aperiodic Pulse')
```

Їх вигляд показано на рис. 14.5

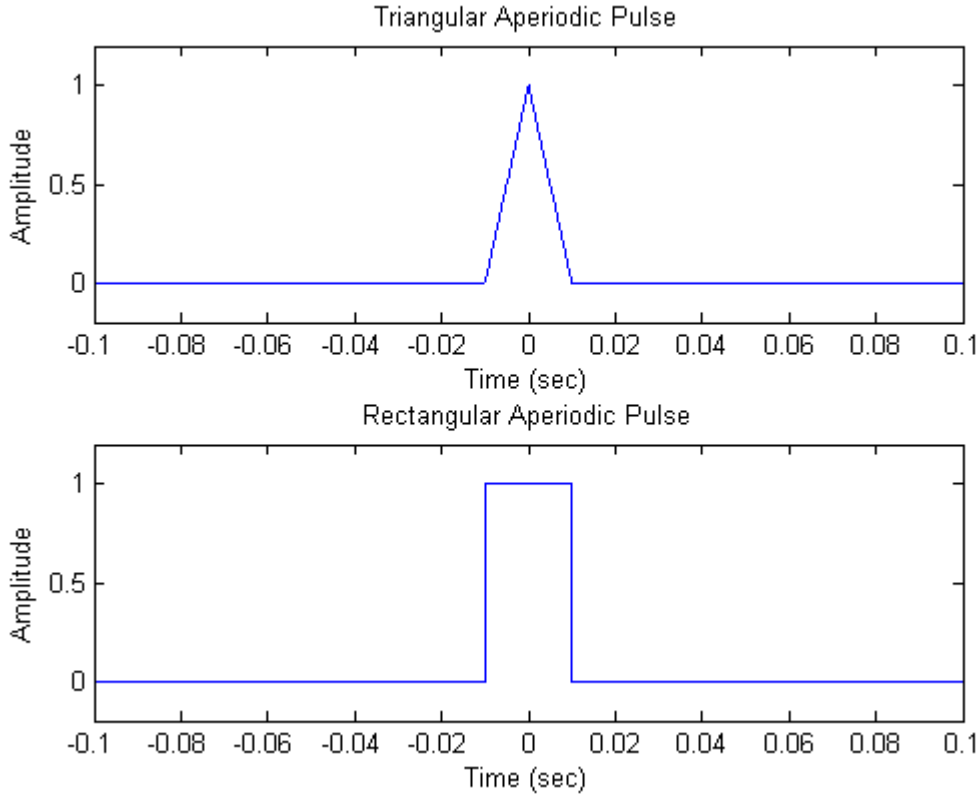


Рис. 14.6. Графіки трикутного та прямокутного імпульсів

14.5.4. Розрахунок перехідних процесів методом віртуального фізичного моделювання

Перевагою віртуального фізичного моделювання є те, що для його застосування не потрібно складати математичний опис досліджуваної системи. Ця функція покладається на програмне забезпечення. Задачею користувача в даному разі є складання принципової електричної схеми у блоках віртуальної бібліотеки *SimPowerSystem*, що потребує деякого досвіду.

Досліджувана електрична схема в *SPS*-блоках подана на рис. 14.7.

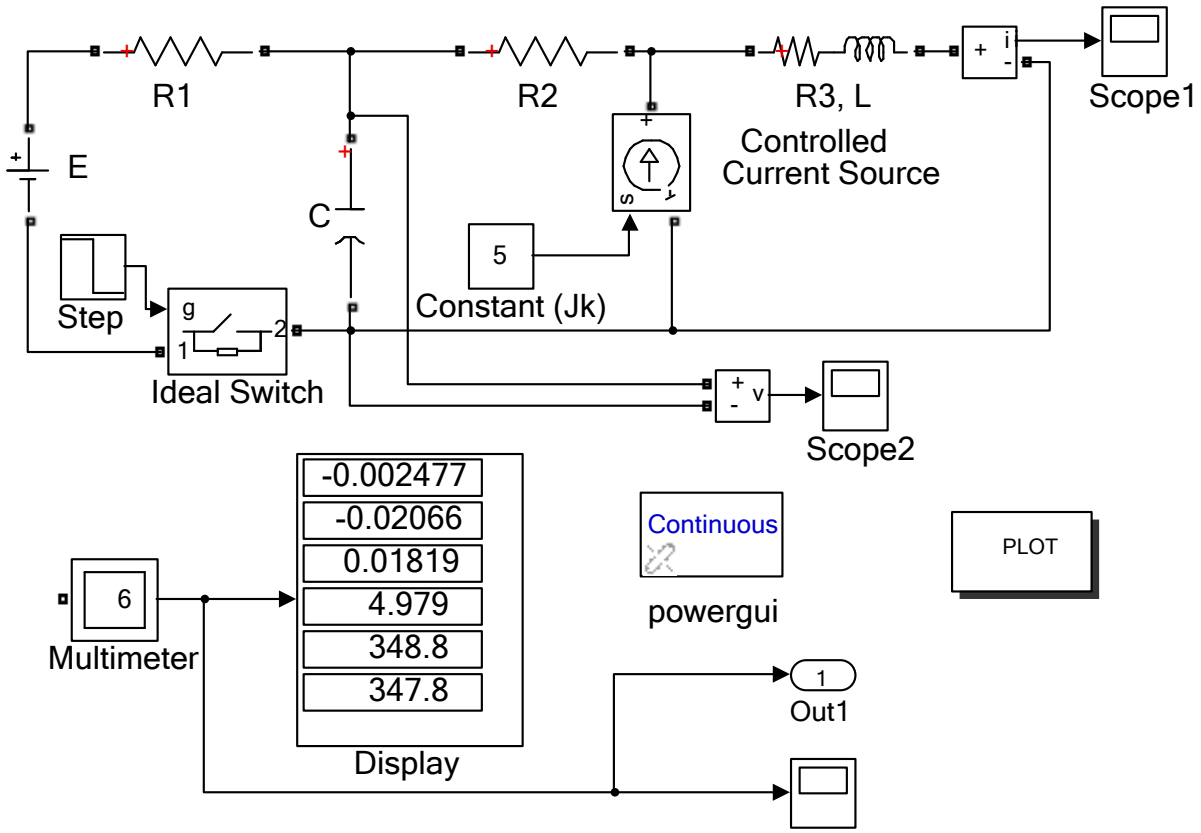


Рис. 14.7. Віртуальна фізична модель досліджуваного електричного кола

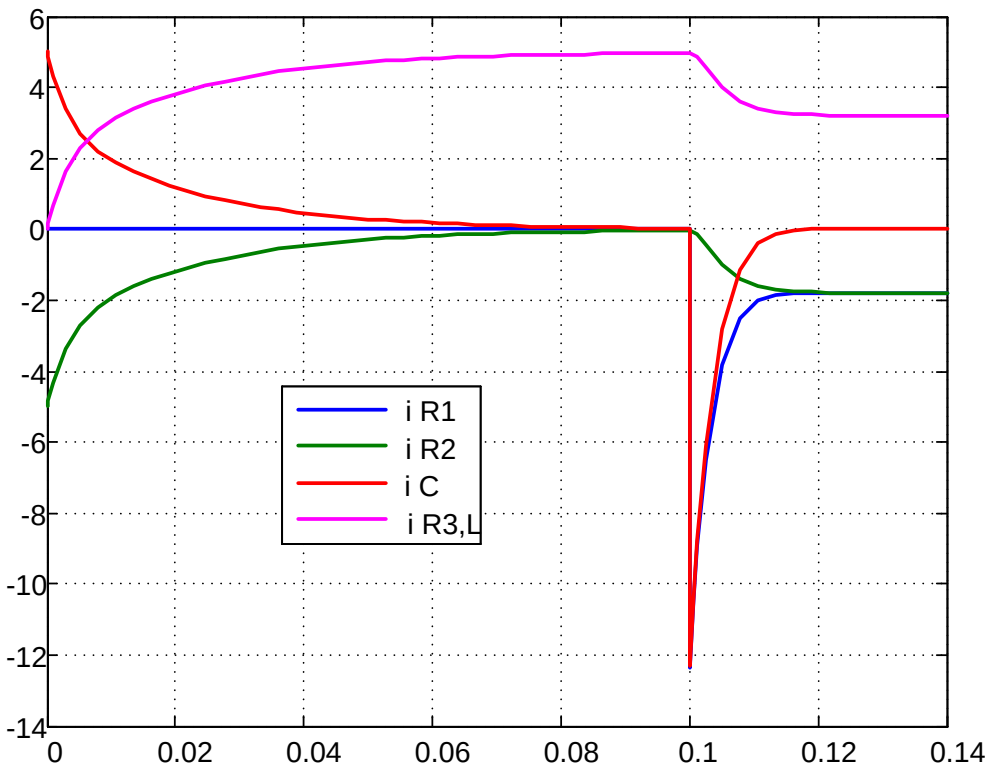


Рис.14.8. Перехідні процеси струмів досліджуваного електричного кола при розмиканні та замиканні ключа

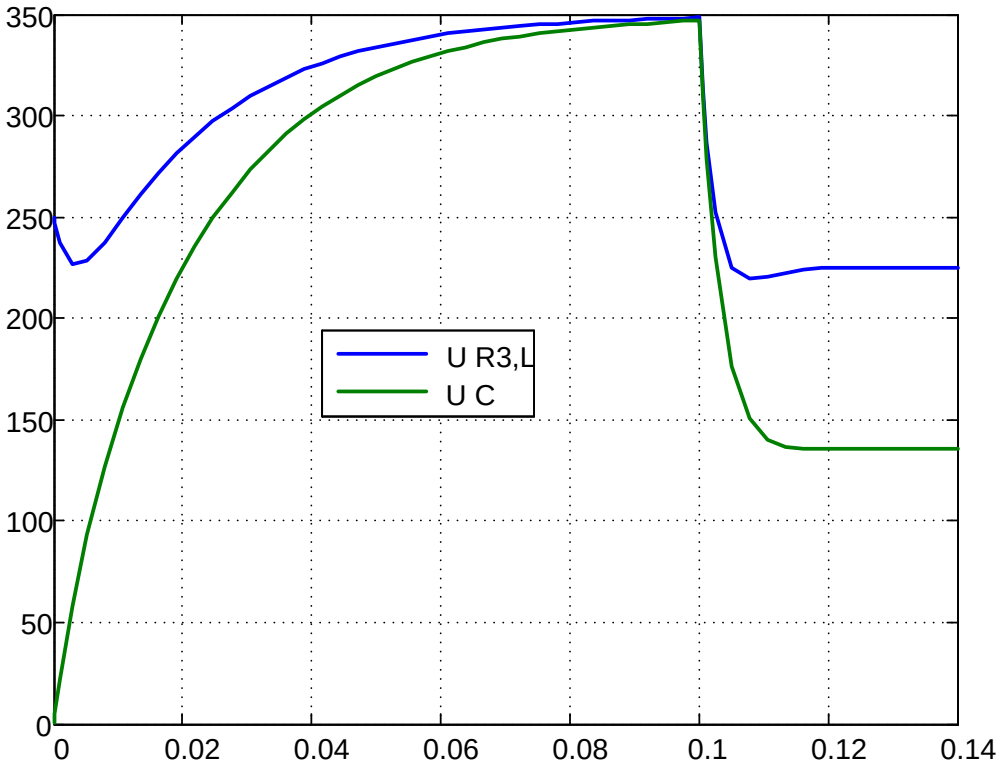


Рис.14.9. Перехідні процеси напруг досліджуваного електричного кола при розмиканні та замиканні ключа

Simulink-блоки у моделі рис. 14.7 використані тільки для фіксації результатів, формування вихідного сигналу джерела струму та керування ключовим елементом *Ideal Switch*. На рисунку продемонстровано можливість вимірювання сигналів різними інструментами.

Графіки перехідних процесів, отриманих за результатами симуляції досліджуваної моделі показані на рис. 14.8, 14.9.

Графіки на останніх рисунках відрізняються від графіків на рис. 14.4 кількістю сигналів та тим, що перші з них отримані при розмиканні ключа та його замиканні, а другі – тільки при замиканні.

14.6. Приклад розрахунку перехідних процесів в нелінійному електричному колі

Якщо у досліджуваних динамічних об‘єктах присутні елементи з нелінійними характеристиками «вхід-вихід», то диференційні рівняння, що їх описують, також стають нелінійними. Для цих випадків треба мати або

аналітичний опис статичних характеристик нелінійних ланок, або таблиці відповідності аргументів та значень цих характеристик. За даними таких таблиць можна виконувати апроксимацію нелінійних залежностей методом найменших квадратів інтерполювати їх на кожному кроці чисельного інтегрування. Оператори, що описують статичні нелінійні характеристики, додають до рівнянь зв‘язку як у функцію правих частин ДР, так і у головну програму.

У якості прикладу розглянемо електричну схему, подану на рис. 14.10, при $U = 80\ \text{V}$, $R = 4\ \text{Om}$. Залежність магнітного потоку котушки індуктивності від струму намагнічування внаслідок ефекту насичення сталі є нелінійною. Графік цієї залежності має вигляд рис. 14.11.

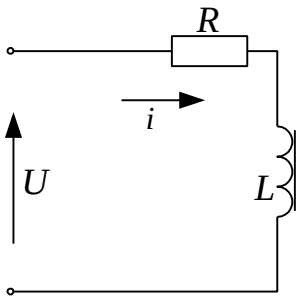


Рис. 14.10. Нелінійне коло

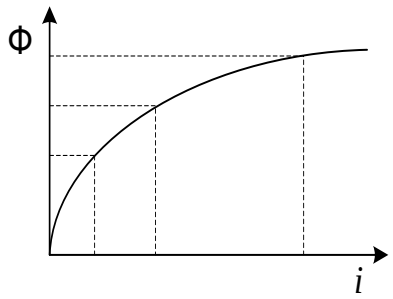


Рис. 14.11. Крива намагнічування

Вона задана табличною функцією $\Phi_T(i_T)$, наприклад,

$$\begin{cases} i_T = [0\ \ 0.02\, 0.042\, 0.067\, 0.098\, 0.141\, 0.205\, 0.3\, 0.447\, 0.669\, 11.48\ ] * 20, \\ \Phi_T = [0:0.1:1.1] * 3.2. \end{cases} \tag{14.24}$$

У такому разі індуктивність

$$L = \frac{d\Phi}{di} \tag{14.25}$$

буде змінною величиною.

Згідно із другим законом Кірхгофа напруга в електричній схемі визначається за формулою

$$U = iR + L\frac{di}{dt}. \tag{14.26}$$

Після підстановки (14.25) у (14.26) отримаємо

$$U = iR + \frac{d\Phi}{di} \cdot \frac{di}{dt} = iR + \frac{d\Phi}{dt},$$

або у нормальній формі Коші

$$\frac{d\Phi}{dt} = U - iR, \tag{14.27}$$

де

$$i = f(\Phi) \tag{14.28}$$

– нелінійне рівняння зв‘язку,

$$\Phi(0) = 0 \tag{14.29}$$

– початкова умова.

Для відомої табличної залежності $\Phi_T = f(i_T)$ можна розрахувати $\Phi(i)$ для будь-якого значення i за допомогою апроксимації чи інтерполяції. При апроксимації кривих намагнічування звичайно використовують степеневі поліноми 3-го або 5-го порядку (обов’язково непарного, тому що функція $\Phi(i)$ центрально-симетрична).

Програмні модулі для розв‘язання поставленої задачі при використанні для математичного опису кривої намагнічування апроксимації та інтерполяції наведені у табл. 14.1.

Таблиця 14.1

Апроксимація	Інтерполяція
Функції розрахунку правих частин ДР	

function dy = frn_a (t,y)
  global R U A
  F=y(1);
  i = polyval(A,F);
  dy(1) = U-i*R;
end
	function dy = frn_i (t,y)
	  global R U it Ft
	  F = y(1);
	  i = interp1(Ft,it,F);
	  dy(1) = U-i*R;
	end

Головна програма	
global R U A U=… R=… y0=… tk=… it = […] Ft = […] C = polyfit (it, Ft, 5); [t,y] = ode23(@frn_a, [0 tk], y0); I = polyval (C, y); plot (t, y, t, I)	global R U it Ft U=… R=… y0=… tk=… it = […] Ft = […] [t,y] = ode23 (@frn_i, [0 tk], y0); I = interp1 (Ft, it, y); plot (t, y, t, I)

Функція i = interp1(Ft,it,F) виконує кусочно-лінійну апроксимацію кривої намагнічування. Іншими словами можна сказати, що вона інтерполює задану таблицею нелінійну залежність рухомими степеневими поліномами 1-го порядку. Такий спосіб не завжди задовольняє вимогам щодо точності розрахунків, тому що інтерпольована у такий спосіб залежність у вузлах таблиці має точки зламу, а графік похідних від інтерпольованої кривої має розриви першого роду.

Підвищити точність інтерполювання можна застосуванням інтерполяції рухомими степеневими поліномами 3-го порядку i=interp1 (Ft, it, F, 'cubic') або інтерполяції кубічними сплайнами i = interp1 (Ft, it, F, 'spline').

У середовищі *Simulink* кусочно-лінійну апроксимацію здійснює блок *Look-Up-Table* з бібліотеки *Functions&Tables*. Параметрами цього блоку є вектор аргументів (*Vector of input values*) та вектор значень (*Table data*) табличної функції.

Для використання кубічної інтерполяції або інтерполяції сплайнами застосовують блок *MATLAB Fcn* з параметром *MATLAB function:* interp1 (Ft, it, u, 'cubic') або interp1 (Ft, it, u, 'spline') відповідно. Для використання апроксимації степеневим поліномом 5-го порядку застосовують блок *Polynomial* з параметром *Polynomial Coefficients:* C = polyfit (it, Ft, 5).

Simulink-модель досліджуваної схеми зображена на рис. 14.12.

Процес ініціалізації цієї моделі та розрахунок перехідних процесів можна виконати за допомогою такої програми:

```
clc, clear all, close all
Ft = [0:0.1:1.1]*12.2
it = [0 0.02 0.042 0.067 0.098 0.141 0.205 0.3 0.447 0.669 1 1.48 ]*20
U = 80, R = 4
C = polyfit (Ft,it,5)
col = 'bkr';
for k = 1:3
   sim ('OMM_NonLinSim')
   figure (1)
   plot (tout,I,col(k)), hold on, grid on
```

```
    figure (2)
    plot (tout,F,col(k)), hold on, grid on
end
figure (1), legend ('I(t)'),
figure (2), legend ('F(t)')
```

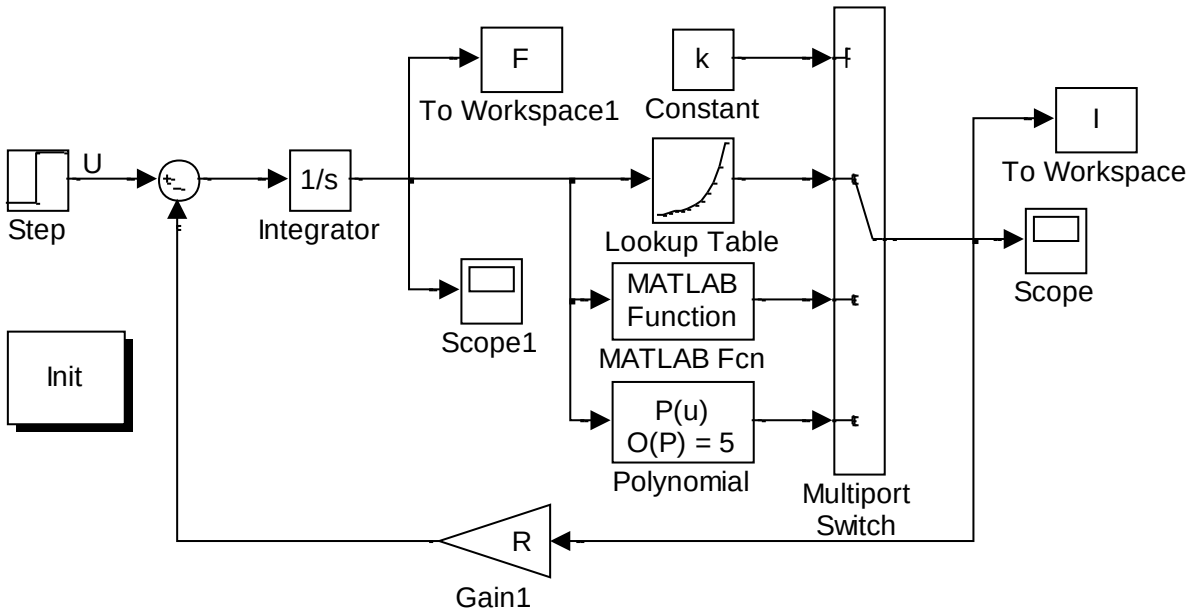


Рис. 14.12. Структурна модель досліджуваного нелінійного кола

Результат виконання наведеної програми наведено на рис. 14.13.

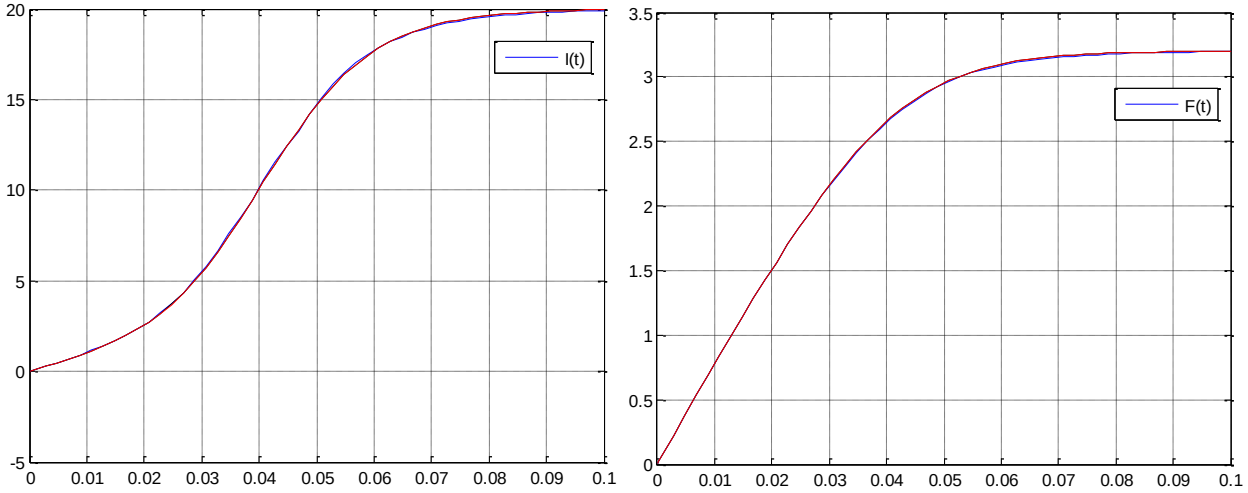


Рис. 14.13. Перехідні процеси в досліджуваному нелінійному електричному колі

14.7. Контрольні питання та завдання

1. Що таке диференційне рівняння? У чому полягають його аналітичне рішення та чисельний розв‘язок?
2. Охарактеризуйте сутність методів Рунге-Кутта.
3. Як треба перетворити ДР перед його чисельним розв‘язанням?
4. Як називаються методи Рунге-Кутта різних порядків?
5. Запишіть формулу одного кроку розв‘язання ДР першого порядку методом Ейлера. З яким методом чисельного інтегрування функцій він збігається?
6. Запишіть формулу одного кроку розв‘язання ДР першого порядку методом Ейлера-Коші. З яким методом чисельного інтегрування функцій він збігається?
7. Які функції пакету *MATLAB* виконують розв‘язання систем ДР першого порядку в нормальній формі Коші? Як ними користуватися?
8. Від чого залежить точність розв‘язання ДР чисельними методами? Як її можна підвищити?
9. Які параметри чисельних методів розв‘язання ДР можна змінювати? Як це зробити?

15. ЧИСЛОВЕ РОЗВ‘ЯЗАННЯ ТРАНСЦЕНДЕНТНИХ ТА АЛГЕБРАЇЧНИХ РІВНЯНЬ

15.1. Загальні відомості

В цьому розділі йдеться про пошук коренів рівняння

$$F(x)=0 \tag{15.1}$$

де $F(x)$ – нелінійна функція однієї змінної загального вигляду.

Якщо ця функція є степеневим поліномом ($F(x)=P_n(x)$, то рівняння (15.1) стає ***алгебраїчним рівнянням відповідного степеню***. ***У трансцендентних рівняннях*** функція $F(x)$ уявляє собою сукупність будь-яких математичних операцій над будь-якими функціями: степеневими, тригонометричними, експоненціальними, логарифмічними і т.п.

Числове розв’язання нелінійних рівнянь поділяється на два етапи: ***відділення коренів*** та ***уточнення їх початкових наближень ітераційними методами***.

Відділення коренів полягає у пошуку початкового наближення шуканого кореня x_0 або інтервалу існування кореня $[a, b]$. на якому функція, нуль якої треба знайти, змінює знак, що математично можна виразити умовою

$$F(a)\cdot F(b)<0. \tag{15.2}$$

Задача відділення коренів складно піддається формалізації, тому що інтервалом визначеності багатьох функцій є $[-\infty,+\infty]$ та кількість дійсних коренів не завжди є відомою. У ручному режимі інтервали існування коренів можна здійснювати шляхом побудови графіків функцій у різних діапазонах зміни аргументу.

Уточнення коренів виконується ***ітераційними методами***, тобто ***методами поступових наближень***. При використанні ітераційних методів розв‘язок знаходять приблизно, з ***заданою точністю***, яка задається у вигляді ***максимально припустимої абсолютної або відносної похибки***.

Найбільш розповсюдженими ***методами уточнення коренів*** є методи ***бісекцій, хорд, дотичних та метод простих ітерацій***.

15.2. Метод бісекцій

Метод бісекцій, який ще називають ***методом діхотомії***, або ***методом половинного ділення***, полягає у послідовного поділу відрізка, який містить корінь, навпіл, як це показано на рис. 15.1:

$$x = \frac{a+b}{2}. \tag{15.3}$$

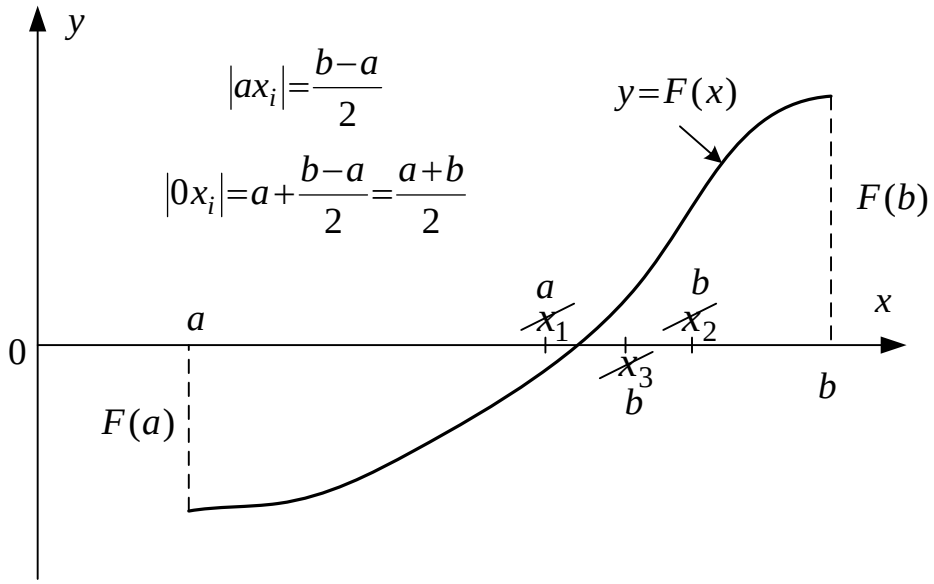


Рис. 15.1. Ілюстрація до методу половинного ділення

Для кожного наступного ділення обирається та половина відрізку, на кінцях якої функція має протилежний знак. При цьому інтервал існування кореня звужується за рахунок зміни однієї з його меж: лівої ($a=x$) або правої ($b=x$).

Ітераційний процес закінчується, якщо виконується умова:

$$b - a \le \varepsilon_x, \tag{15.4}$$

де ε – задана точність обчислення кореня ($\varepsilon << 1$). Часто вимагають, щоб одночасно виконувалася умова:

$$|F(x)| \le \varepsilon_y. \tag{15.5}$$

Метод бісекцій – простий та надійний спосіб пошуку коренів рівняння $F(x)=0$. Він збігається для будь-яких неперервних функцій $F(x)$, у тому числі і

для таких, що не диференціюються. Швидкість збіжності невелика. Заради досягнення точності ε необхідно витратити

$$N = \log_2\left(\frac{b-a}{\varepsilon_x}\right) \tag{15.6}$$

ітерацій. Це означає, що задля отримання кожних 3 вірних десяткових знаків необхідно зробити близько 10 ітерацій.

Якщо на відрізку $[a,b]$ знаходиться кілька коренів, то процес збігається до одного з них. Метод непридатний для пошуку кратних коренів парного порядку.

15.3. Метод хорд

Метод хорд, або ***метод пропорційних відрізків***, полягає у послідовному поділенні відрізка $[a,b]$, який має корінь, на частини, пропорційні значенням функції на кінцях відрізку:

$$\frac{F(a)}{F(b)} = \frac{a-x}{b-x}, \tag{15.7}$$

звідки

$$x = \frac{aF(b) - bF(a)}{F(b) - F(a)}. \tag{15.8}$$

Геометрично це еквівалентно заміні графіка функції $F(x)$ хордою, яка пройде через точки $(a,F(a))$ та $(b,F(b))$, як це показано на рис. 15.2.

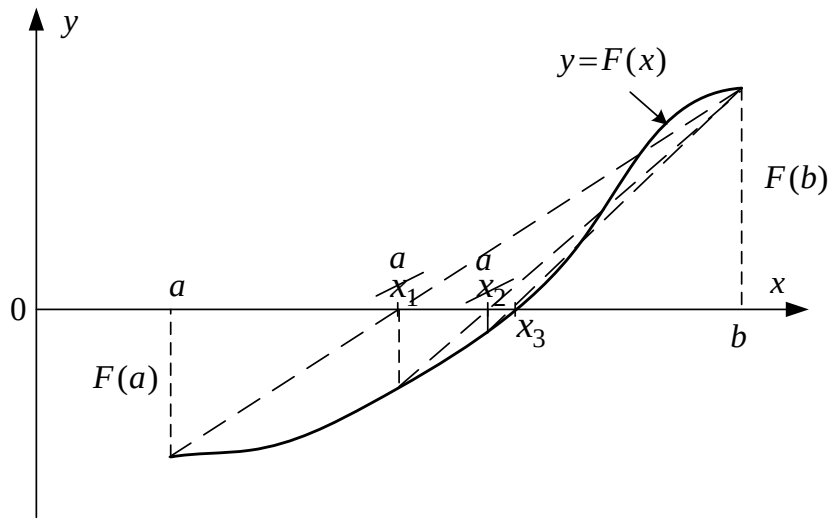


Рис. 15.2. Ілюстрація до методу хорд

Для закінчення ітераційного процесу замість умови (15.4) використовують умову:

$$\left|x_{i+1} - x_i\right| \le \varepsilon, \tag{15.9}$$

де x_{i+1}, x_i – відповідно, останнє обчислене і попереднє до нього наближення кореня. У решті цей метод походить на метод бісекцій, але забезпечує більш швидку збіжність.

15.4 Метод дотичних

Метод дотичних, або ***метод Ньютона***, полягає у послідовній апроксимації функції $F(x)$ дотичними до кривої у точці попереднього наближення $(x_i, F(x_i))$, які перетинають ось абсцис у точці наступного наближення x_{i+1}, як це показано на рис. 15.3.

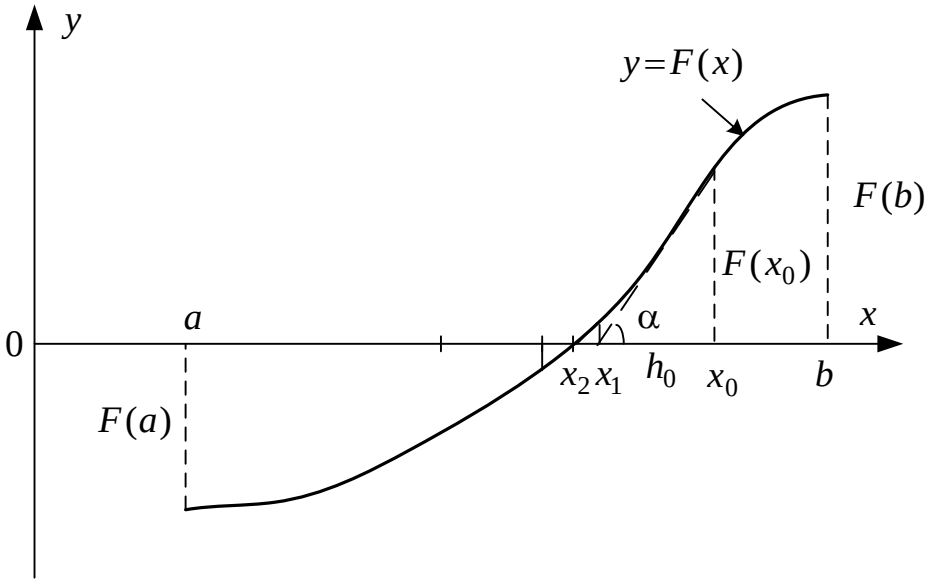


Рис. 15.3. Ілюстрація до методу дотичних

З рис. 14.3 видно, що

$$tg\alpha = F'(x_i) = \frac{F(x_i)}{h_i} = \frac{F(x_i)}{x_{i+1} - x_i},$$

звідкіля випливає:

$$x_{i+1} = x_i - h_i = x_i - \frac{F(x_i)}{F'(x_i)}. \qquad (15.10)$$

Послідовність $[x_0, x_1, x_2, ...]$ збігається до дійсного значення кореня рівняння $F(x)=0$, якщо його початкове наближення лежить у інтервалі $[a,b]$ ($F(a)*F(b)<0$), на якому похідні $F'(x)$ та $F''(x)$ тримають власний знак та виконується умова:

$$F(x_0) \cdot F''(x_0) > 0. \qquad (15.11)$$

Ітерації припиняють при виконанні умов (15.9) та (або) (15.5).

Метод Ньютона ефективний, якщо початкове наближення для кореня обрано з урахуванням умови (15.11) та поблизу кореня графік функції має велику крутизну. У сприятливих випадках число вірних десяткових знаків у черговому наближенні подвоюється, тобто процес збігається дуже швидко.

Недоліком метода дотичних є необхідність розраховувати у кожній точці не тільки значення функції, але і значення похідних.

15.5. Метод простих ітерацій

Метод простих ітерацій полягає у заміні вихідного рівняння $F(x)=0$ еквівалентним йому рівнянням:

$$x = \varphi(x) \qquad (15.12)$$

та обчисленні послідовності

$$x_{i+1} = \varphi(x_i), \qquad (15.13)$$

(i=1,2,3,...), яка збігається при $i \to \infty$ до точного рішення.

Ітерації припиняють при виконанні умови (15.9). Достатньою та необхідною умовою збіжності метода є:

$$|\varphi'(x)| < 1. \qquad (15.14)$$

Збіжні ітераційні процеси зображені на рис. 15.4, а розбіжні – на рис. 15.5.

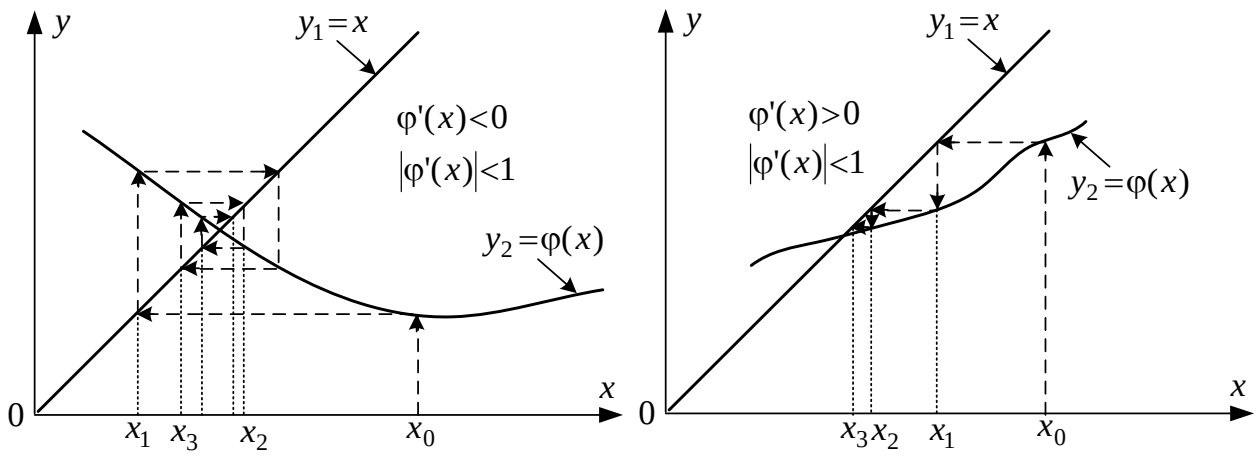


Рис. 15.4. Ілюстрація до методу простих ітерації
при збіжності ітераційного процесу

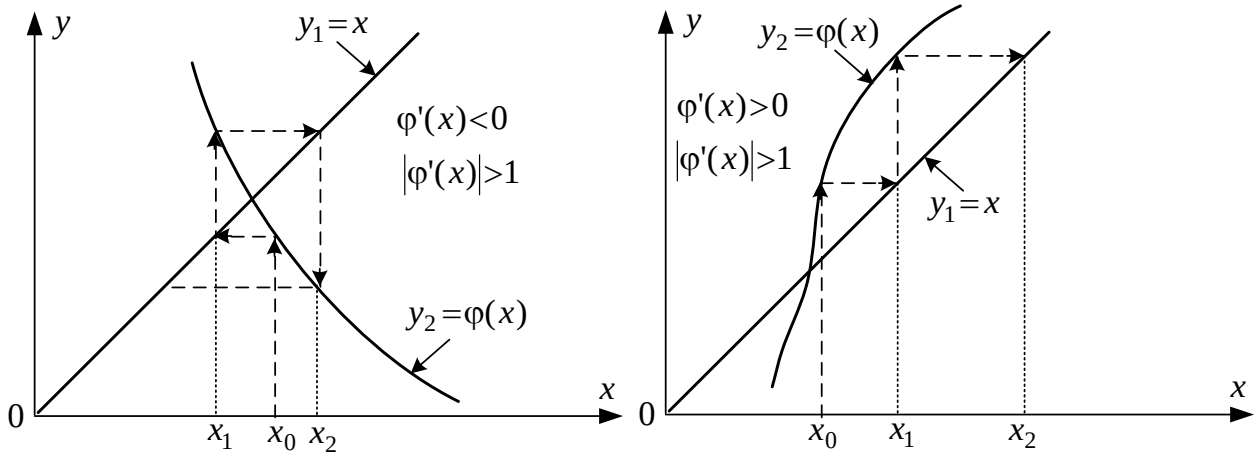


Рис. 15.5. Ілюстрація до методу простих ітерації
при розбіжності ітераційного процесу

Швидкість збіжності зростає зі зменшенням $|\varphi'(x)|$.

15.6. Розв‘язання трансцендентних та алгебраїчних рівнянь у середовищі пакета *MATLAB*

У пакеті MATLAB рішення нелінійних рівнянь виконує функція fzero, яка знаходиться у папці optim (*Optimisation Toolbox*). Звернення до неї з мінімальною кількістю параметрів має вигляд:

```
x = fzero(Fun,x0)
```

де Fun – або ідентифікатор функції (*function handle*), що утворюється за форматом @FunName, або ім‘я функції у вигляді рядка символів ‘FunName’,

або звернення до *inline*-функції у вигляді inline ('expr'), або так звана анонімна функція *(anonymous function)* у форматі @(x) expr; x0 – початкове наближення кореня або діапазон існування коренів ([a b]), на якому функція хоча б один раз змінює знак.

Ця функція знаходить єдиний нуль заданої функції в околі заданого початкового наближення або на заданому інтервалі.

Отже, отримати розв‘язок рівняння $\sin(x)=0$ в околі точки 3 можна такими способами:

```
x = fzero (@sin,3)
x = fzero ('sin',3)
x = fzero (inline('sin(x)'),3)
g = inline('sin(x)'),  x = fzero (g,3)
x = fzero (@(x) sin(x), 3)
```

Щоб мати можливість втручатися у процес рішення, змінюючи алгоритм, точність розв‘язку, максимальну кількість ітерацій, тощо, список вхідних параметрів можна доповнити параметром options:

```
x = fzero (Fun,x0,Options)
```

елементи якого треба попередньо визначити за допомогою функції optimset:

```
options = optimset ('param1',value1,'param2',value2,...)
```

Щоб визначитися з іменами параметрів, які можна змінювати, їх можливими значеннями та значеннями, що призначаються «за замовченням» (*default*), треба звернутися до функції optimset без вхідних та вихідних параметрів. Значення за замовчанням подаються у круглих дужках.

Для того, щоб оцінити точність розв‘язку, можна до вихідних параметрів додати дійсне значення функції Fval, яке буде близьким до нуля але ненульовим.

Якщо при пошуку нуля функції виникли проблеми, корисним буде вихідний параметр ExitFlag, який може приймати такі значення:

- 1 – «нуль» функції знайдено з заданою точністю;
- -1 – при заданій кількості ітерацій задана точність не досягнута;

- -3 – при пошуку інтервалу існування нуля функція отримала значення NaN або Inf;
- -4 – при пошуку інтервалу існування нуля функція отримала комплексне значення;
- -5 – процес пошуку нуля не збігається в одну точку;
- -6 – не знайдено інтервал, на якому функція змінює знак.

Для того, щоб дізнатися інформації про використані чисельні методи пошуку нуля, кількість ітерацій, кількість обчислень значень функцій, треба скористуватися ще одним вихідним параметром Output.

Отже у максимальній конфігурації звернення до функції fzero має формат:

[x,Fval,ExitFlag,Output] = fzero(Fun,x0,Options)

Наприклад, при виконанні оператора

```
[x,f,ex,out] = fzero(@sin,3)
```

отримаємо такі результати

```
x =
   3.1416
f =
  1.2246e-016
ex =
    1
out =
   intervaliterations: 3
         iterations: 5
          funcCount: 12
          algorithm: 'bisection, interpolation'
        message: 'Zero found in the interval [2.83029, 3.16971]'
```

Щоб прослідкувати процес збіжності розв‘язку, тобто побачити результати послідовності ітерацій, слід встановити параметр Display у стан Iter. Наприклад, в результаті виконання операції’

```
[x,f] = fzero (Fun,x0,optimset('Display','Iter'))
```

отримаємо

```
Search for an interval around 3 containing a sign change:
 Func-count    a          f(a)             b          f(b)         Procedure
```

```
     1           3        0.14112          3         0.14112      initial interval
     3       2.91515     0.224515     3.08485    0.0567094       search
     5          2.88     0.258619        3.12     0.021591       search
     7       2.83029     0.306295     3.16971    -0.0281093      search

  Search for a zero in the interval [2.83029, 3.16971]:
   Func-count    x          f(x)             Procedure
     7        3.16971    -0.0281093        initial
     8        3.14118    0.000417192       interpolation
     9        3.14159   -5.41432e-008      interpolation
    10        3.14159    1.45473e-015      interpolation
    11        3.14159    1.22465e-016      interpolation
    12        3.14159    1.22465e-016      interpolation

  Zero found in the interval [2.83029, 3.16971]
x =
    3.1416
  f =
 1.2246e-016
```

Для рішення алгебраїчних рівнянь в пакеті MATLAB є функція roots.

X = roots(P) – обчислює вектор коренів X полінома з коефіцієнтами P.

15.7. Контрольні питання

1. На які етапи ділиться процес чисельного розв‘язання рівнянь?
2. Як можна визначити початкові наближення коренів або діапазони їхнього існування?
3. Які методи ітераційного уточнення коренів Ви знаєте? Які у кожного з них недоліки та переваги.
4. Що Ви можете сказати про кількість ітерацій при застосуванні кожного з методів?
5. Які методи ітераційного уточнення коренів потребують знання початкового наближення кореня, а які – знання діапазону його існування?
6. Як математично визначити факт перетину деякою функцією осі абсцис?
7. Яка *MATLAB*-функція розв‘язує трансцендентні рівняння з однією змінною?
8. Як визначити інформацію про метод розв‘язання рівняння функцією fzero?
9. Як вивести на екран послідовність ітерацій?

16. ЧИСЛОВЕ РОЗВ'ЯЗАННЯ СИСТЕМ НЕЛІНІЙНИХ РІВНЯНЬ

16.1. Формулювання задачі

Система n рівнянь з n невідомими у векторно-матричній формі має вигляд:

$$\mathbf{F(X)} = \mathbf{0}, \ \mathbf{X} = [x_1, x_2, \ldots x_n], \tag{16.1}$$

а у скалярній формі –

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) = 0, \\ f_2(x_1, x_2, \ldots, x_n) = 0, \\ \ldots \\ f_n(x_1, x_2, \ldots, x_n) = 0. \end{cases} \tag{16.2}$$

Задача полягає у пошуку вектору коренів заданої системи за їх початковими наближеннями ітераційними методами з заданою точністю. Початкові значення коренів обираються на основі деяких знань про вирішувану задачу або методом «спроб і помилок» на етапі відділення коренів. Для систем 2-го порядку відділення коренів можливо графічним методом. При «поганих» початкових наближеннях ітераційний процес може бути таким, що розбігається.

До найбільш відомих ***ітераційних методів*** розв'язання систем нелінійних рівнянь належать ***метод простих ітерацій, метод Зейделя та метод Ньютона***.

16.2. Метод звичайних ітерацій

Для застосування цього метода треба початкову систему рівнянь перетворити до вигляду

$$\begin{cases} x_1 = \varphi_1(x_1, x_2, \ldots, x_n), \\ x_2 = \varphi_2(x_1, x_2, \ldots, x_n), \\ \ldots \\ x_n = \varphi_n(x_1, x_2, \ldots, x_n). \end{cases} \tag{16.3}$$

Якщо відомі початкові наближення коренів

$$\mathbf{X}^{[0]} = (x_1^{[0]}, x_2^{[0]}, \ldots, x_n^{[0]}), \tag{16.4}$$

то для їх уточнення використовують формули

$$\begin{cases} x_1^{[k]} = \varphi_1(x_1^{[k-1]}, x_2^{[k-1]}, \ldots, x_n^{[k-1]}), \\ x_2^{[k]} = \varphi_2(x_1^{[k-1]}, x_2^{[k-1]}, \ldots, x_n^{[k-1]}), \\ \ldots \\ x_n^{[k]} = \varphi_n(x_1^{[k-1]}, x_2^{[k-1]}, \ldots, x_n^{[k-1]}), \end{cases} \tag{16.5}$$

де k=1, 2, 3, ...– номер ітерації.

Ітерації закінчують при досягненні умови:

$$\max_i = \left| x_i^{[k]} - x_i^{[k-1]} \right| \le \varepsilon, \tag{16.6}$$

де ε – максимально припустима похибка результатів.

Достатні умови збіжності ітераційного процесу мають вигляд:

$$\sum_{j=1}^{n} \frac{\partial \varphi_j(x_1^{[0]}, x_2^{[0]}, \ldots, x_n^{[0]})}{\partial x_i} < 1, \tag{16.7}$$

або

$$\sum_{j=1}^{n} \frac{\partial \varphi_i(x_1^{[0]}, x_2^{[0]}, \ldots, x_n^{[0]})}{\partial x_j} < 1. \tag{16.8}$$

Вони повинні виконуватися для усіх значень i (i=1, 2, ..., n).

16.3. Метод Зейделя

Метод Зейделя відрізняється від метода звичайних ітерацій тільки формулами уточнення коренів:

$$\begin{cases} x_1^{[k]} = \varphi_1(x_1^{[k-1]}, x_2^{[k-1]}, \ldots, x_n^{[k-1]}), \\ x_2^{[k]} = \varphi_2(x_1^{[k]}, x_2^{[k-1]}, \ldots, x_n^{[k-1]}), \\ \ldots \\ x_n^{[k]} = \varphi_n(x_1^{[k]}, x_2^{[k]}, \ldots, x_n^{[k-1]}). \end{cases} \tag{16.9}$$

У більшості випадків він забезпечує більш швидку збіжність ітераційного процесу. Але обидва розглянуті методи є дуже чутливими до початкових

наближень та обмеженими необхідністю попередніх перетворень та виконання умов (16.8).

16.4. Метод Ньютона

Метод Ньютона є похідним від методу дотичних для одного рівняння.

Вектор прирощень коренів $\Delta\mathbf{X}$ на кожному кроці ітераційного процесу визначається шляхом розв‘язання системи n лінійних рівнянь з n невідомими:

$$\mathbf{J}^{[k-1]}\cdot\Delta\mathbf{X}=-\mathbf{F}(\mathbf{X}^{[k-1]}), \tag{16.10}$$

де $\mathbf{F}(\mathbf{X})$ – вектор лівих частин початкової системи рівнянь (16.1), $\mathbf{J}$ – ***матриця Якобі, або Якобіан***, яка уявляє собою ***транспоновану матрицю градієнтів***:

$$\mathbf{J}=\frac{\partial\mathbf{F}(\mathbf{X})}{\partial\mathbf{X}}=\begin{bmatrix}\frac{\partial f_1(\mathbf{X})}{\partial x_1} & \frac{\partial f_1(\mathbf{X})}{\partial x_2} & \cdots & \frac{\partial f_1(\mathbf{X})}{\partial x_n}\\ \frac{\partial f_2(\mathbf{X})}{\partial x_1} & \frac{\partial f_2(\mathbf{X})}{\partial x_2} & \cdots & \frac{\partial f_2(\mathbf{X})}{\partial x_n}\\ \cdots & \cdots & \cdots & \cdots\\ \frac{\partial f_n(\mathbf{X})}{\partial x_1} & \frac{\partial f_n(\mathbf{X})}{\partial x_2} & \cdots & \frac{\partial f_n(\mathbf{X})}{\partial x_n}\end{bmatrix} \tag{16.11}$$

Уточнення коренів здійснюють за формулою:

$$\mathbf{X}^{[k]}=\mathbf{X}^{[k-1]}+\Delta\mathbf{X}. \tag{16.12}$$

Ітерації завершують при здійсненні умови (16.6). Для жорсткішого контролю треба разом з умовою (16.6) перевіряти умову

$$\max_i\left|f_i(x_1^{[k]},x_2^{[k]},\ldots,x_n^{[k]})\right|\le\varepsilon_f. \tag{16.13}$$

16.5. Розв‘язання систем нелінійних рівнянь у середовищі пакета *MATLAB*

У пакеті MATLAB рішення систем нелінійних рівнянь виконує функція fsolve, яка за способом свого вживання подібна до функції fzero (див. попередню роботу), разом з якою вона відноситься до інструментів оптимізації (*Optimization Toolbox*). При максимальній кількості параметрів звернення до неї має формат

```
[x,Fval,ExitFlag,Output] = fsolve (Fun,x0,Options)
```

Різниця полягає у діагностичних повідомленнях, що є значеннями вихідного параметру ExitFlag, та у способі визначення опціональних параметрів. Якщо для функції fzero вони визначаються функцією optimset, то для функції fsolve – функцією optimoptions:

optimoptions(Solver,'Param1',Val1,'Param2',Val2,...)

Наприклад, на звернення

```
optimoptions('fsolve')
```

отримаємо відповідь:

```
ans =
  fsolve options:
  Options used by current Algorithm ('trust-region-dogleg'):
  (Other available algorithms: 'levenberg-marquardt', 'trust-region-reflective')

   Set by user:
     No options set by user.

   Default:
         Algorithm: 'trust-region-dogleg'
    DerivativeCheck: 'off'
       Diagnostics: 'off'
     DiffMaxChange: Inf
     DiffMinChange: 0
           Display: 'final'
    FinDiffRelStep: 1.4901e-08
       FinDiffType: 'forward'
       FunValCheck: 'off'
          Jacobian: 'off'
       MaxFunEvals: '100*numberOfVariables'
           MaxIter: 400
         OutputFcn: []
          PlotFcns: []
            TolFun: 1.0000e-06
              TolX: 1.0000e-06
          TypicalX: 'ones(numberOfVariables,1)'
```

Щоб проглянути послідовність ітерацій, тобто змінити стан параметра , треба виконати оператор

x = fsolve (Fun,x0, optimoptions('Display','Iter'))

Алгоритм роботи функції fsolve базується на мінімізації суми квадратів складових функції Fun ***методами Гауса-Ньютона і Левенберга-Марквардта***.

Функцію fsolve можна використовувати і як альтернативу функції fzero для пошуку кореня одного нелінійного рівняння.

Приклад 16.1. *Розв'язати систему рівнянь*

$$\begin{cases} x_1 + x_2 - \sin(\pi x_1) = 0, \\ x_1 - x_2 - \cos(\pi x_2) = 0. \end{cases} \qquad (16.14)$$

Складемо функцію, що розраховуватиме значення правих частин системи у вигляді вектору-стовпця:

```
function y = myFunVec1(x)
    y = [x(1)+x(2)-sin(pi*x(1));
         x(1)-x(2)-cos(pi*x(2))];
end
```

Розробимо програму для графічного визначення початкових наближень коренів. Для цього з першого рівняння системи (16.14) знайдемо залежність $x_{21}(x_1) = y_1 = \sin(\pi x_1) - x_1$, *а з другого рівняння системи (16.14) – залежність* $x_{12}(x_2) = y_2 = \cos(\pi x_2) + x_2$ *та побудуємо в одній системі координат графіки* $x_{21}(x_1)$ *та* $x_2(x_{12})$:

```
clc, clear all, close all
x1=linspace(-1,1.5);
y1=sin(pi*x1)-x1;  plot(x1,y1,'LineWidth',2);
xlim([-1 1.5]), ylim([-1 1.5])
axis equal; grid on; hold on
x2=x1; y2=cos(pi*x2)+x2;  plot(y2,x2,'r','LineWidth',2);
xlabel('xl'); ylabel('x2');
title('Графічне визначення початкових наближень коренів')
```

Точки перетину побудованих графіків (див. рис. 16.1) є коренями досліджуваної системи. Початковими наближеннями коренів зробимо координати малих точок, зображених на рис. 16.1, та уточнимо їх за допомогою функції fsolve.

Уточнені корені зображені на рис. 16.1 великими точками. Для того, щоб отримати такий рисунок, необхідно доповнити попередню програму такими операторами:

```
x1x2 = [-0.7  -0.7;    % Матриця початкових значень
        -0.3  -0.3;
         0.3   0.8;
         0.7   0.6;
           1  -0.3];   % Матриця початкових значень
m=size(x1x2,1)         % Кількість шуканих коренів
for i=1:m
   x1=x1x2(i,1); x2=x1x2(i,2);
```

```
    plot(x1,x2, 'Marker','.', 'MarkerSize', 10);
    x = fsolve(@myFunVec1,[x1,x2]);
    plot (x(1), x(2), 'Marker', '.', 'MarkerSize', 20);
    plot ( [ x1 x(1) ] , [ x2 x(2) ] , 'k-' ) ;
end
```

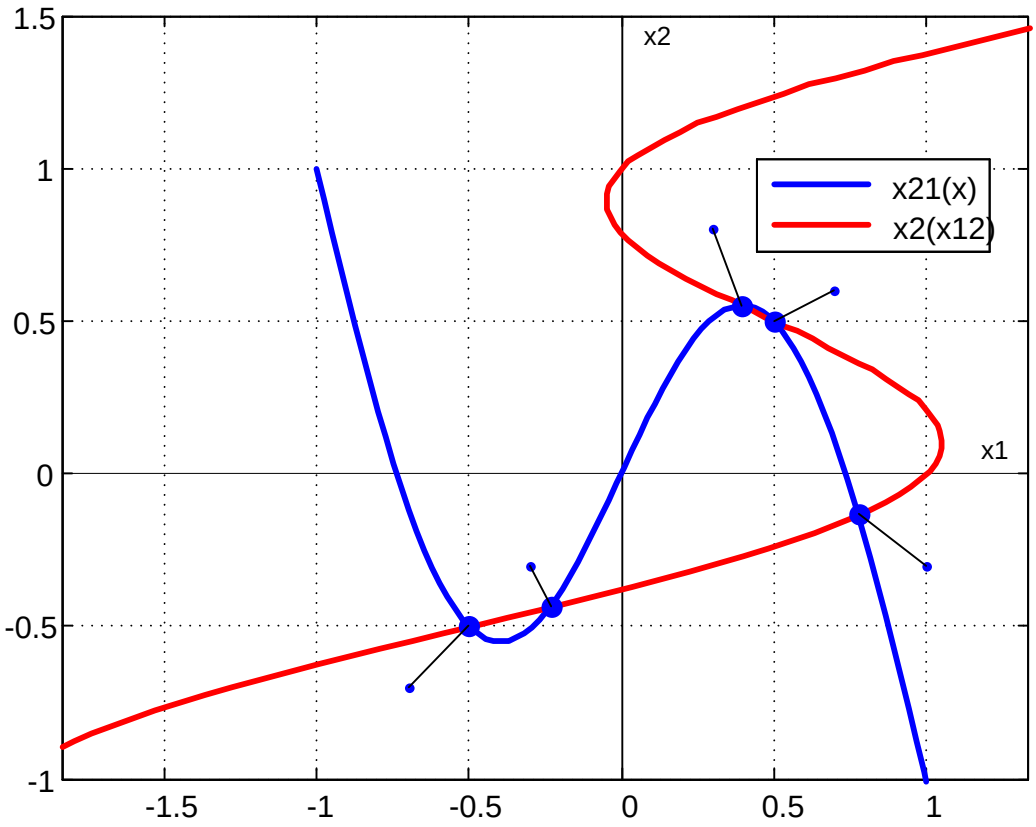


Рис. 16.4. Ілюстрація процесу розв‘язання системи нелінійних рівнянь (16.14)

Тепер розглянемо більш ретельно процес пошуку останнього кореня шляхом звернення до функції fsolve *з початковими умовами* x0 = [1; -0.3]:

```
>> [x,f] = fsolve (@myFunVec1, [1; -0.3])
Equation solved.
fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.
<stopping criteria details>
x =
    0.7776
   -0.1345
f =
   1.0e-11 *
    0.0570
    0.2051
```

Щоб отримати більше інформації про процес розв'язання

```
>> [x,f,flag,Info] = fsolve (@myFunVec1, [1; -0.3], optimset('Display','Iter'))
```

Iteration	Func-count	f(x)	Norm of step	First-order optimality	Trust-region radius
0	3	0.99725		3.61	1
1	6	0.00687815	0.24921	0.222	1
2	9	4.98015e-06	0.0280089	0.00541	1
3	12	4.34206e-12	0.000817837	4.5e-06	1
4	15	4.5321e-24	7.86843e-07	4.12e-12	1

```
x =
   0.7776
  -0.1345
f =
  1.0e-11 *
   0.0570
   0.2051
flag =
    1
Info =
      iterations: 4
       funcCount: 15
       algorithm: 'trust-region-dogleg'
       firstorderopt: 4.1223e-12
      message: [1x691 char]
```

У функції Fun *може бути передбачена можливість розрахунку матриці Якобі за аналітичними формулами, які для даного прикладу згідно з (16.14)мають вигляд:*

$$\mathbf{J}=\frac{\partial \mathbf{F}(\mathbf{X})}{\partial \mathbf{X}}=\begin{bmatrix} \frac{\partial f_1(\mathbf{X})}{\partial x_1} & \frac{\partial f_1(\mathbf{X})}{\partial x_2} \\ \frac{\partial f_2(\mathbf{X})}{\partial x_1} & \frac{\partial f_2(\mathbf{X})}{\partial x_2} \end{bmatrix}=\begin{bmatrix} 1-\pi\cdot\cos(\pi x_1) & 1 \\ 1 & -1+\pi\cdot\sin(\pi x_2) \end{bmatrix} \qquad (16.15)$$

Це може суттєво зменшити кількість звернень до функції Fun *і, тим самим, прискорити ітераційний процес. У цьому разі необхідно змінити значення вибіркового параметру* 'Jacobian' *з* 'off' *(за замовчанням) на* 'on'*, а формули для обчислення Якобіана додати до тіла функції* Fun *з двома вихідними параметрами*:

```
function [y,J] = myFunVec2(x)
```

```
    y=[x(1)+x(2)-sin(pi*x(1));
       x(1)-x(2)-cos(pi*x(2))];
    if nargout == 2
       J = [1-pi*cos(pi*x(1))   1;
             1   -1+pi*sin(pi*x(2))];
      end
end
```

В результаті зменшиться кількість обчислень значень функцій на кожній ітерації:

```
>> opt = optimoptions('fsolve','Jacobian','on','Display','Iter');
>> [x,f,flag,Info] = fsolve (@myFunVec2, [1;-0.3], opt)
```

Iteration	Func-count	f(x)	Norm of step	First-order optimality	Trust-region radius
0	1	0.99725		3.61	1
1	2	0.00687815	0.24921	0.222	1
2	3	4.98014e-06	0.0280089	0.00541	1
3	4	4.34183e-12	0.000817836	4.5e-06	1
4	5	4.32711e-24	7.86823e-07	4.04e-12	1

```
x =
   0.7776
  -0.1345
f =
  1.0e-11 *
   0.0550
   0.2006
flag =
    1
Info =
      iterations: 4
       funcCount: 5
       algorithm: 'trust-region-dogleg'
   firstorderopt: 4.0403e-12
         message: [1x691 char]
```

Причиною цього ефекту є позбавлення від необхідності розраховувати матрицю Якобі чисельними методами, замінюючи диференціали змінних їх приростеннями.

16.6. Контрольні питання

1. Які дані треба мати для розв‘язання систем нелінійних рівнянь?
2. Які методи ітераційного уточнення коренів Ви знаєте?

3. Поясніть сутність методу Ньютона.
4. Що таке матриця Якобі? Як її можна розрахувати?
5. Яка *MATLAB*-функція розв‘язує трансцендентні рівняння з однією змінною?
6. Яка різниця у зверненні до функцій fzero та fsolve?
7. Як узнати значення параметрів функції fsolve, що призначаються за замовчанням?
8. Як визначити інформацію про метод розв‘язання рівняння функцією fsolve?
9. Як вивести на екран послідовність ітерацій?
10. Як підвищити швидкодію розв‘язання системи нелінійних рівнянь функцією fsolve?

ЛІТЕРАТУРА

1. Попович М.Г., Ковальчук О.В. Теорія автоматичного керування: Підручник. – 2-ге вид., перероб. і доп. – К.: Либідь, 2007. – 656 с.

2. Phillips C., Harbor R. Feedback control systems, Prentice-Hall, 2000, 658 p.

3. Lutz H., Wendt W. Taschenbuch der Regelungstechnik. – Tun und Frankfurt am Main: Verlag Harry Deutsch, 2000. – 1148 S.

4. Solving control engineering problems with MATLAB (MATLAB curriculum series) / K. Ogata. – Englewood Cliffs, NJ: Prentice Hall, 1994. – 359 p.

5. Ogata K. Modern control engineering, Prentice-Hall, 2010. – 905 p.

6. Dingyü Xue, YangQuan Chen, Derek P. Atherton. Linear Feedback Control Analysis and Design with MATLAB. – Philadelphia: Siam, 2007. – 356 p.

7. Теорія автоматичного керування. Курсова робота: навчальний посібник / КПІ ім. Ігоря Сікорського; уклад.: О.І. Толочко, С.М. Пересада, Б.І. Приймак. – Київ : КПІ ім. Ігоря Сікорського, 2022. – 163 с.

8. Лазарєв Ю.Ф. Основи програмування в середовищі MatLAB: Навч. посібник. – К.: НТУУ "КПІ", 2003. – 424 с.

9. Лазарєв Ю.Ф. Довідник з MATLAB / Електронний навчальний посібник з курсового і дипломного проектування. – К.: НТУУ "КПІ", 2013. – 132 с.

10. Brian D. Hahn, Deniel T. Valentine. Essentiel MATLAB vor Engineers and Scientists. MATLAB examples: 7-th editions. – Elsevier Ltd, Academic Press, 2019. – 412 p.

11. Manassah Jamal T. Elementary mathematical and computational tools for electrical and computer engineers using MATLAB. – CRC Press LLC, 2001. – 349 p.

12. Helmut Bode. MATLAB-Simulink. Analyse und Simulation dynamischer Systeme: 2. Auflage. – Wiesbaden: Teubner Verlag, 2006. – 319 S.

13. Pietruszka W.D. MATLAB und Simulink in der Ingenieurpraxis. Modelbildung, Berechnung und Simulation: 2. Auflage. – Wiesbaden: Teubner Verlag, 2006. – 402 S.

14. Гаєв Є.О., Нестеренко Б.М. Універсальний математичний пакет MATLAB і типові задачі обчислювальної математики / Навчальний посібник. – К.: НАУ, 2004. – 176 с.

15. Задачин В.М., Конюшенко І.Г. Чисельні методи. Навчальний посібник. – Харків. Вид. ХНЕУ ім. С. Кузнеця, 2014. – 180 с.

16. Математичні методи в електромеханіці: навч. посіб. для студентів спеціальності 141 «Електроенергетика, електротехніка та електромеханіка» / Укл. О.І. Толочко. – Київ: КПІ ім. Ігоря Сікорського, 2020. – 212 с.

17. Математичні методи в електромеханіці та теорії керування. Методичні вказівки до лабораторних робіт для студентів спеціальності 141 – "Електроенергетика, електротехніка та електромеханіка" спеціалізації "Електромеханічні системи автоматизації та електропривод" / Укл.: О.І.Толочко. – Київ: КПІ ім. Ігоря Сікорського, 2016. – 81 с.

18. Фельдман Л.П., Петренко А.І., Дмитрієва О.А. Чисельні методи в інформатиці. – К.: Видавнича група BHV, 2006. – 480 с.

19. Островерхов М.Я., Пижов В.М. Моделювання електромеханічних систем в Simulink: Навч. посібник для студентів вищих навчальних закладів. – К.: ВД «Стилос», 2008. – 528 с.

20. Моделювання систем автоматичного керування: підручник / О.І. Толочко; КПІ ім. Ігоря Сікорського. – Київ: КПІ ім. Ігоря Сікорського, 2024. – 328 с.

21. Моделювання систем автоматичного керування. Методичні вказівки до лабораторних робіт: навч. посіб. / О. І. Толочко; КПІ ім. Ігоря Сікорського. – Київ: КПІ ім. Ігоря Сікорського, 2023. – 250 с.

22. Control System Toolbox. User's Guide, The MathWorks, Release 2015b, 2015.

23. Краснопрошина А.А., Репнікова Н.Б., Ільченко А.А. Сучасний аналіз систем керування з використанням MATLAB, Simulink, Control System: Навчальний посібник. – К.: "Корнійчук", 1999. – 144 с.

24. Лозинський А., Мороз В., Паранчук Я. Розв‘язування задач електромеханіки в середовищі пакетів MathCAD і MATLAB: навч. посібник. – Львів: Видавництво Державного університету «Львівська політехніка», 2000. – 166 с.

25. Толочко О.І. Аналіз та синтез електромеханічних систем зі спостерігачами стану. – Донецьк: НОРД-ПРЕС, 2004. – 298 с.

ДОДАТОК А

ОБРОБКА ТА РЕДАГУВАННЯ ГРАФІКІВ ЗА ДОПОМОГОЮ ФУНКЦІЙ ГРАФІЧНИХ ВІКОН

А1. Деякі функції меню графічного вікна

Для редагування графіків, підпису осей, та ін. можна використовувати не тільки команди *MATLAB*, а ще й функції меню та «кнопки» інструментів графічного вікна. Типове графічне вікно має меню, панель інструментів та поле для побудови графіка. Верхня частина цього вікна показана на рис. А.1. На рис. А.2 та рис. А.3 зображені панелі інструментів, які виводяться відповідними функціями команди *View*.

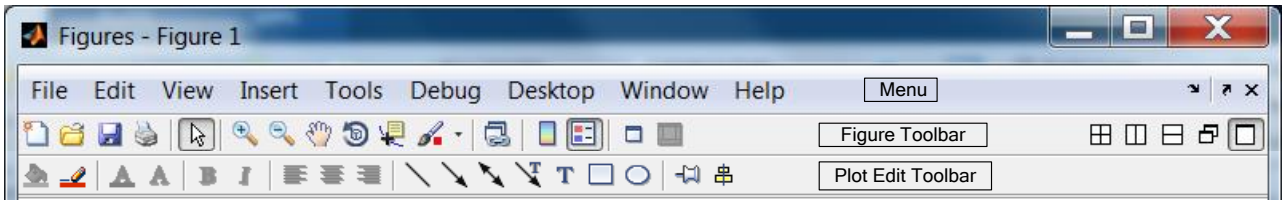


Рис. А1. Заголовок, меню та інструменти графічного вікна

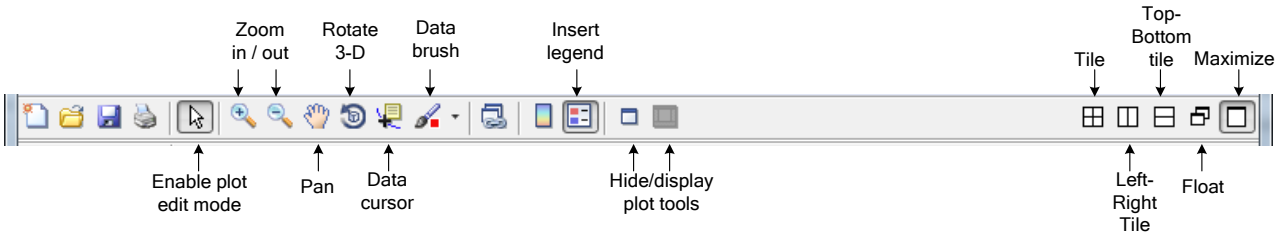


Рис. А2. Панель *Figure Toolbar* графічного вікна

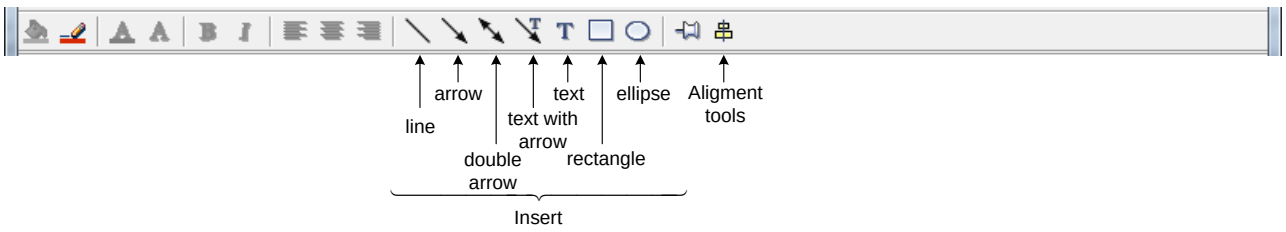


Рис. А3. Панель *Plot Edit Toolbar* графічного вікна

На рис. А4 відображене вікно, що відкривається при натисканні на кнопку *Alignment Tools* (вирівнювання).

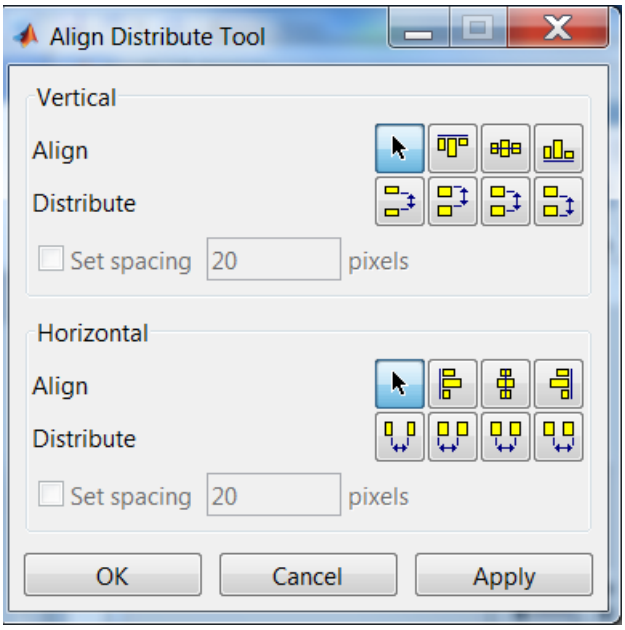


Рис. А4. Вікно, призначене для вирівнювання графічних об‘єктів

Перелічимо найважливіші пункти меню графічного вікна, виключивши із них загально відомі, що не потребують пояснень.

1) ***File*** (Файл):

- ***Close (^W)*** –закриття графічного вікна без збереження;
- ***Save (^S)*** – збереження графічного вікна у файлі з розширенням ***.fig***;
- ***Preferences → Figure Copy Template*** – налаштування графічного вікна;
- ***Print Preview*** – попередній перегляд графіка перед друком;
- ***Print*** – друк вмісту графічного вікна.

2) ***Edit*** (Редагування):

- ***Copy Figure*** – копіювання вмісту графічного вікна у буфер обміну;
- ***Copy Options*** – налаштування копіювання вмісту графічного вікна (доступні також у пункті меню ***File→ Preferences***);
- ***Figure Properties…*** – редагування властивостей графічного вікна;
- ***Axis Properties…*** – редагування властивостей системи координат;
- ***Current Object Properties…*** – редагування властивостей виділеного об’єкта;

- ***Colormap…*** – редагування карти кольорів (для тривимірних графіків та рисунків).

3) ***Viev*** (Вигляд):

- ***Figure Toolbar*** – активація/дезактивація панелі для редагування графічного вікна;
- ***Camera Toolbar*** – активація/дезактивація панелі керування камерою вигляду графічного вікна;
- ***Plot Edit Toolbar*** – активація/дезактивація панелі редагування графіків.

4) ***Insert*** (Вставка):

- ***Xlabel***, ***Ylabel***, ***Zlabel*** – вставка підпису осей абсцис, ординат, аплікат;
- ***Title*** – вставка підпису графіка;
- ***Legend*** – вставка легенди графіків;
- ***Line*** – вставка лінії;
- ***Arrow*** – вставка стрілки;
- ***Text Arrow*** – вставка стрілки з текстом;
- ***Double Arrow*** – вставка подвійної стрілки;
- ***TextBox*** – вставка тексту;
- ***Rectangle*** – вставка прямокутника;
- ***Ellipse*** – вставка еліпса;
- ***Axes*** – вставка системи координат.

5) ***Tools*** (Інструменти):

- ***Edit Plot*** – режим редагування графіка;
- ***Zoom In***, ***Zoom Out*** – збільшення та зменшення зони перегляду графіка;
- ***Pan*** – захват та рух графіка разом з координатною віссю;
- ***Rotate 3D*** – обертання графіка у тривимірній площині за допомогою миші;
- ***Data Cursor*** – визначення координат точок, виділених курсором;
- ***Brush*** – дає можливість виділити фрагменти графіка іншим кольором та іншою товщиною;

- ***Reset View*** – скидання всіх масштабних змін;
- ***Options*** – опції масштабу та захвату;
- ***Data Statistics*** – деякі статистичні дані по кожному з графіків (мінімум, максимум, середнє значення, інтервал і т.п.).

6) ***Desktop*** (Робочий стіл):

- ***dock figure*** – закріплює графік у вікні редагування властивостей графіка або на робочому столі;
- ***undock figure*** – відкріпляє графік від робочого стола або видаляє його з вікна редагування.

7) ***Window*** (Вікно) – переключення між різними графічними вікнами, якщо їх більше одного.

8) ***Help*** (Допомога): ***Graphics help, Plotting tools, Annotation Graph, Printing und exporting***

A2. Редагування вмісту графічного вікна

Одним із способів перейти у цей режим є вибір опції ***Property editor*** із сукупності команд функції ***View*** основного меню графічного вікна. Після цього під графіком відкривається вкладка *Property editor – Figure*, зображена рис. А.5. У такий стан *editor* повертається при щиглику мишею на фоні графічної фігури, що оточує систему координат.

Рис. А5. Вкладка *Property editor – Figure* графічного вікна

Вона дозволяє змінити колір фону вікна, надати або змінити його ім‘я та обрати палітру, що має сенс для тривимірних графіків та рисунків або фотографій.

Якщо клацнути кнопкою миші всередині системи координат або на одній з її осей, то відкривається вкладка *Property editor – Axes* (див. рис. А.6).

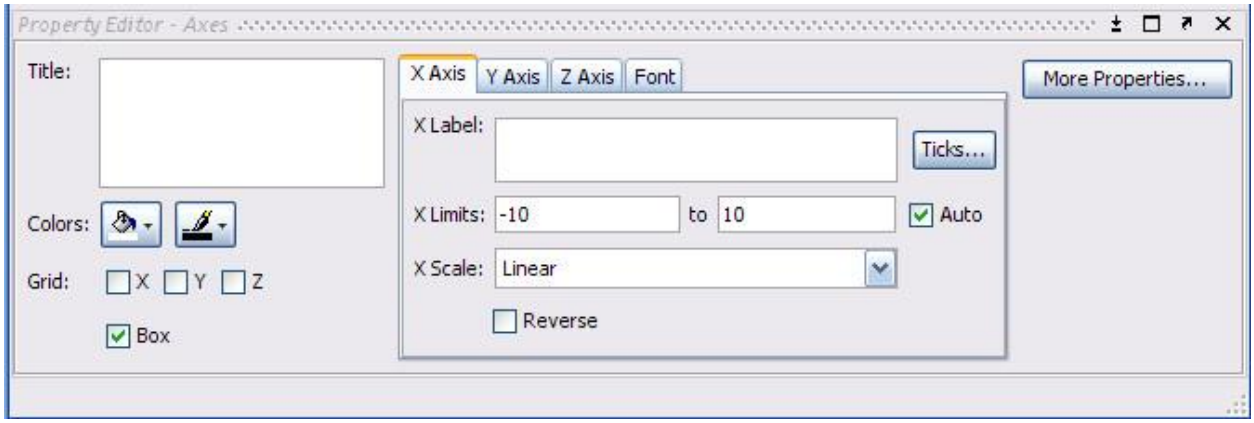


Рис. А6. Вкладка *Property editor – Axes* графічного вікна

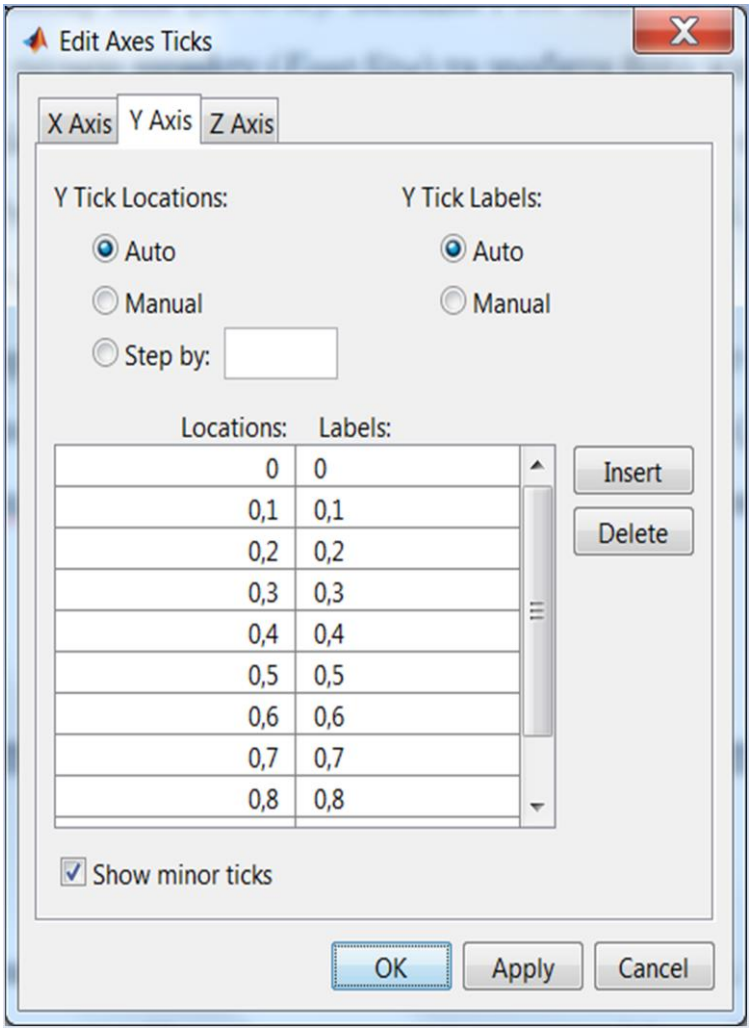


Рис. А7. Вікно редагування міток на осях

Вкладка ***Property editor - Axes*** дозволяє редагувати координатну сітку, а саме: створити назву графіка (*Title*), створити сітку по всім осям (*Grid*) та змінити її колір і колір заливки (*Colors*), вставити мітки осей (*X label*, *Y label*, *Z label*), змінити межі осей (*X limits*, *Y limits*, *Z limits*) та їх поділки (кнопка *Ticks*). Вигляд вікна *Edit Axes Ticks* показаний на рис. А.7. змінити лінійну шкалу на логарифмічну (*Scale*) та реверсувати координатну вісь (*Reverse*).

Функція *Font* задає ім’я шрифту підпису осей (*Font Name*), розмір шрифту (*Font Size*) та зробити його жирним або курсивом (*Weight* та *Angle*). Для більш детальних опцій редагування осей слід натиснути кнопку *More Properties*.

Щиглик мишею на лінії або будь-якій точці графіка відкриває вкладку *Property editor – Lineseries* (рис. А8).

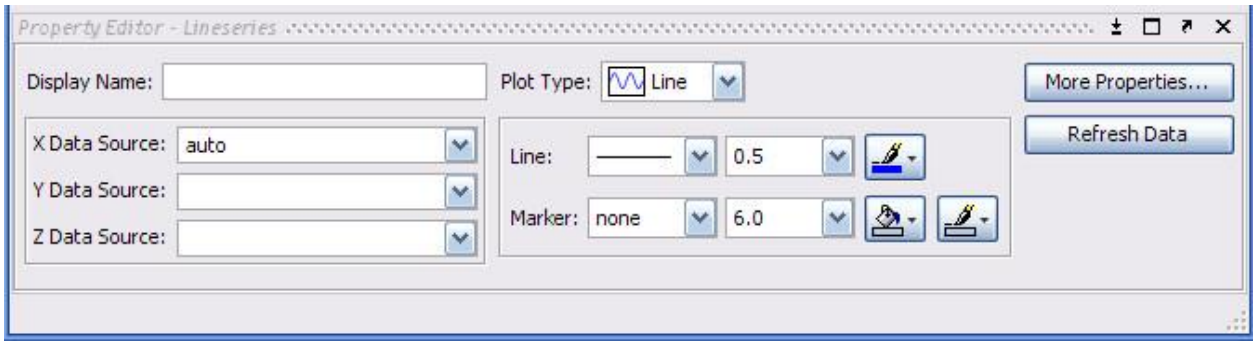


Рис. А8. Вкладка *Property editor- Lineseries* графічного вікна

Вкладка *Property editor - Lineseries* дозволяє редагувати сам графік, а саме: ввести текст для легенди (*Display Name*). вибрати тип графіка (*Plot Type*), тип лінії, її товщину та колір (*Line*), тип маркера, його розмір, колір та заливку (*Marker*). Для відображення більш детальних опцій слід натиснути кнопку *More Properties*.

А3. Апроксимація графіків в діалоговому режимі

Графіки, побудовані за табличними значеннями аргументів і функцій, отриманими експериментально, неточно відображають результати експериментів внаслідок наявності похибок вимірювання і дискретного характеру інформації. Похибки виникають і при розв‘язанні математичних та прикладних науково-технічних задач чисельними методами. Зокрема це стосується розв‘язання диференційних рівнянь ітераційними методами з автоматичним вибором кроку чисельного інтегрування. В цьому разі кількість розрахованих точок навіть при досить високій точності їх розрахунку іноді виявляється недостатньою для якісної візуалізації.

Для згладжування таких результатів в пакеті *MatLab* передбачена можливість апроксимації табличних функцій.

Приклад А1. *За даними табл. А.1 побудувати графік функції P(v) та в разі необхідності апроксимувати табличну функцію степеневим поліномом методом найменших квадратів і побудувати графік апроксимуючої функції.*

Таблиця А1.

Швидкість повітряного потоку, м/с	10	13	15.9	18.5	21.5	24.4	27.1	29.5	31.5
Потужність, Вт	25	62.5	125	175	250	375	500	625	800

Після виконання програми

```
V=[10 13 15.9 18.5 21.5 24.4 27.1 29.5 31.5];
P=[25 62.5 125 175 250 375 500 625 800];
plot(V,P)
```

на екрані отримуємо графік, зображений на рис. А9.

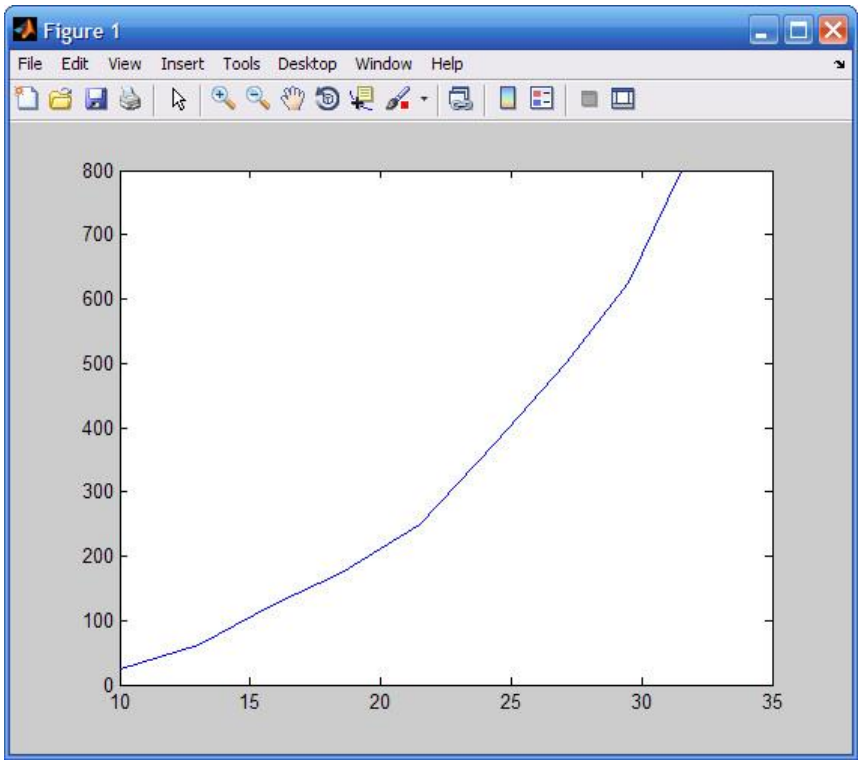
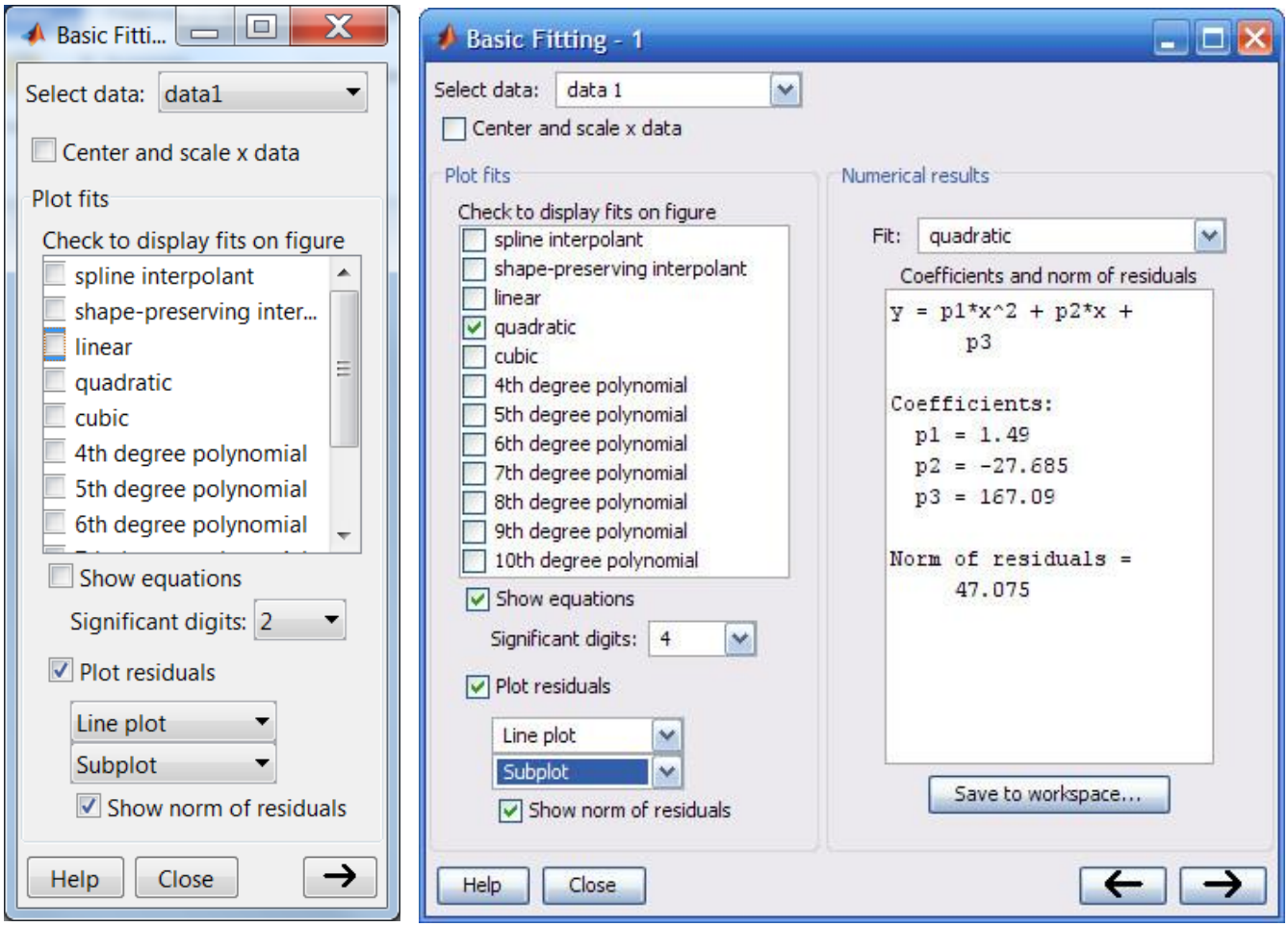


Рис. А9. Графік, побудований за даним табл. А1

Як бачимо, графік має видимі точки зламу. Щоб позбавитися від цього недоліку скористаємося опцією Basic Fitting меню Tools. При цьому

відкриється вікно, зображене на рис. А10а. В ньому пропонуються інтерполяція кубічними сплайнами (Spline Interpolant) та апроксимація степеневими поліномами 1-го (Linear), 2-го (Quadratic), 3-го (Cubic) та 4-10-го порядку (4th-10th degree polynomial), що здійснюється методом найменших квадратів



а) *б)*

Рис. А10. Вікно Basic Fitting до (а) та після (б) установки параметрів

На рис. А10б показано вікно згладжування після вибору у ньому методу апроксимації Quadratic, установки прапорця у полі Show Equations, вибору кількості цифр у коефіцієнтах апроксимуючої параболи (Significant digits).

Для порівняння апроксимованої залежності і реального графіка можна вивести їх різницю, поставивши прапорець Plot Residuals, в якому вибрати тип

лінії (Bar, Scatter або Line) та спосіб відображення (Subplot – вікно розіб'ється на 2 підвікна або Separate Figure – відкриється друге графічне вікно). Також можна поставити прапорець на Show norm of residuals (показати суму квадратичних відхилень апроксимуючої функції від вихідної табличної у вузлових точках).

В результаті отримаємо графік, показаний на рис. А11.

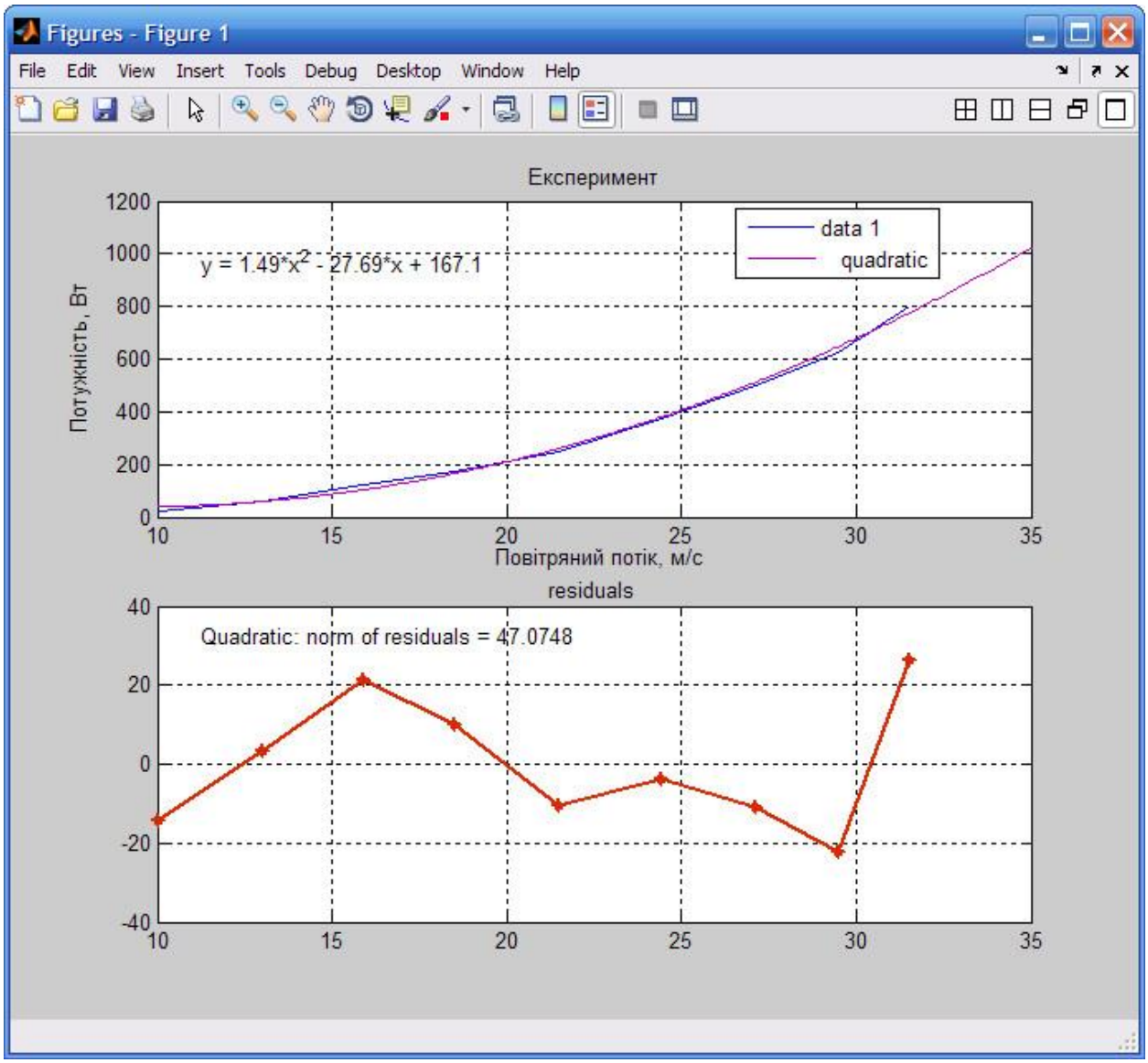


Рис. А11. Вихідний та апроксимований графіки до прикладу А.1 та їх різниця

Текстові об'єкти *MATLAB* можна створювати і коригувати засобами видавницького текстового редактора *TeX*. Він може створювати верхні та нижні

індекси, спеціальні символи та грецькі літери. Але набагато простіше це зробити у будь-якому графічному редакторі, наприклад, *Visio*, який входить до складу програм *Microsoft Office*.

А4. Редагування тексту за допомогою функцій TeX

Текстові об‘єкти *MATLAB* можна створювати і коригувати засобами видавницького текстового редактора *TeX*. Він може створювати верхні та нижні індекси, спеціальні символи та грецькі літери. Для активації функцій редактора *TeX* необхідно перед написанням тексту поставити один з ключів, приведених у табл. А.2. Можна застосовувати як один, так і кілька поспіль ключів.

Розглянемо приклад застосування функцій *TeX*. Нехай є знятий графік перехідного процесу кутової швидкості та моменту двигуна $\omega = f(t)$ та $M = f(t)$, показаний на рис. А.12. Потрібно підписати осі 14-м шрифтом *Times New Roman*, а сам графік жирним курсивом та 16-м шрифтом. Літеру ω зробити грецькою за допомогою функції *TeX*. На графік нанести сітку та стрілками показати швидкість і момент (напис зробити 12-м шрифтом). Лінії зробити товщиною 2 і різними кольорами.

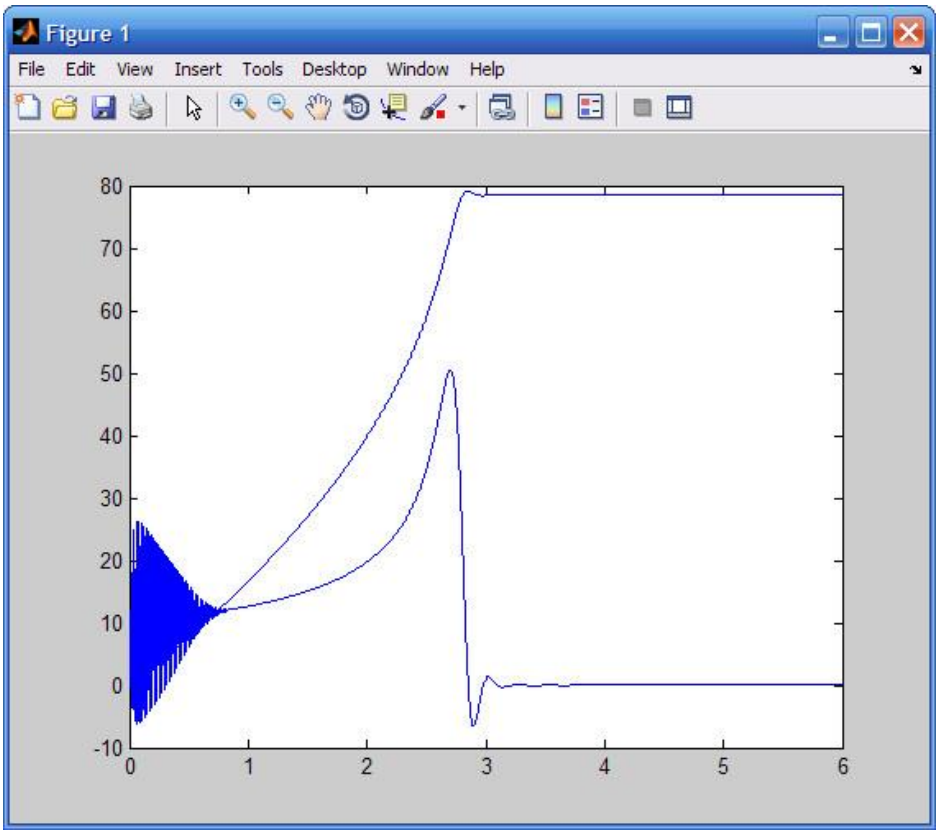


Рис. А12. Вихідні графіки перехідних процесів

Таблиця А2

Редагування тексту			
Значення	***Команда***		
Виділення курсивом	\it Текст		
Виділення жирним	\bf Текст		
Жирний курсив	\bf\it текст		
Ім’я та розмір шрифту	\fontname {ІмяШрифту}\fontsize {Розмір} текст		
Верхній індекс	текст^{індекс}		
Нижній індекс	текст_{індекс}		
Грецькі символи			
Значення	***Команда***	***Значення***	***Команда***
Альфа (α)	\alpha	Ро (ρ)	\rho
Бета (β)	\beta	Сігма (σ)	\sigma
Гама (γ)	\gamma	Тау (τ)	\tau
Дельта (δ)	\delta	Фи (φ)	\phi
Епсілон (ε)	\epsilon	Чи (χ)	\chi
Ета (η)	\eta	Пси (ψ)	\psi
Тета (θ)	\theta	Омега (ω)	\omega
Капа (κ)	\kappa	Велика Гама (Γ)	\Gamma
Лямбда (λ)	\lambda	Велика Дельта (Δ)	\Delta
Мю (μ)	\mu	Велика Тета (Θ)	\Theta
Ню (υ)	\nu	Велика Лямбда (Λ)	\Lambda
Ксі (ξ)	\xi	Велика Фі (Φ)	\Phi
Спеціальні символи			
Значення	***Команда***		
Менше-рівно ($\leq$)	\leq		
Більше-рівно ($\geq$)	\geq		
Двостороння стрілка ($\leftrightarrow$)	\leftrightarrow		
Стрілка вліво ($\leftarrow$)	\leftarrow		
Стрілка вправо ($\rightarrow$)	\rightarrow		
Плюс-мінус ($\pm$)	\pm		
Нескінченність ($\propto$)	\propto		
Часткова похідна (∂)	\partial		
Стрілка униз ($\downarrow$)	\downarrow		

Стрілка угору (↑)	\uparrow

Відкриваємо вікно *Property editor – Axes*. У полі *Title* пишемо

\bf\it\fontname{TimesNewRoman}\fontsize{16} Графік перехідного процесу швидкості та моменту

Потім ставимо прапорець *Grid* на осях X та Y. Далі у полі *Xlabel* пишемо

\fontname{TimesNewRoman}\fontsize{14} Час перехідного процесу t, с

У полі *Ylabel* пишемо

\fontname{TimesNewRoman}\fontsize{14}Швидкість \omega, рад/с Момент M, Н*м

Далі два рази натискаємо на криву швидкості і обираємо вкладку *Property editor – Lineseries*. Виставляємо там товщину лінії – 2. Те ж саме повторюємо для графіка моменту. Змінюємо також колір цієї графічної функції на червоний. За допомогою меню *Insert→Text Arrow* вставляємо стрілки з текстом \fontsize{12}Швидкість\omega для швидкості та \fontsize{12}Момент M для моменту. Закриваємо вкладку і отримуємо графік, показаний на рис. А.13.

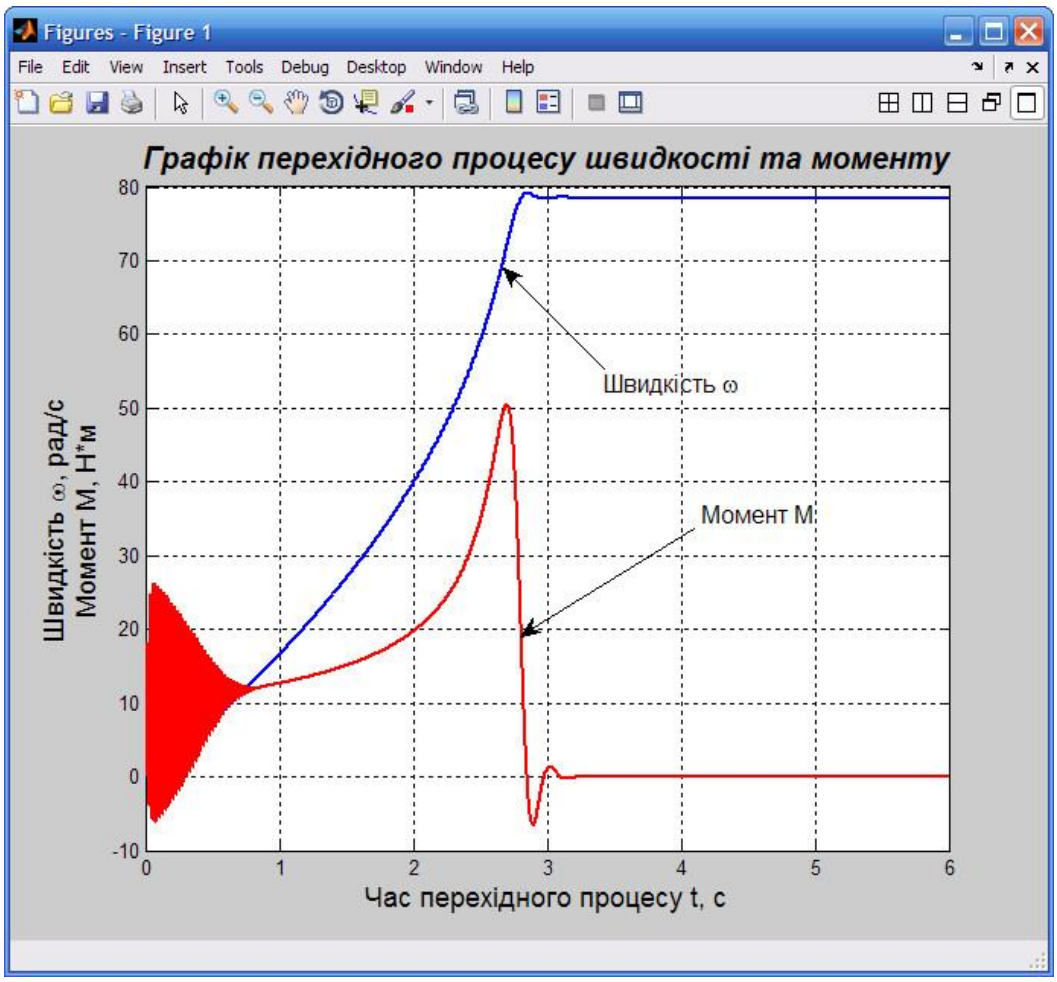


Рис. А13. Приклад оформлення графіків з використанням функцій *TeX*

А5. Питання та завдання для самоперевірки

1. Назвіть основні параметри та пункти меню графічного вікна.
2. Як відкрити режим редагування графічного вікна?
3. Як змінити параметри координатної сітки?
4. Як змінити параметри ліній графіка?
5. Як змінити параметри графічного вікна?
6. Як додати на графік стрілки з текстом та інші об’єкти?
7. Як будуються графіки за знятими експериментальними даними?
8. Як виконати апроксимацію табличної функції?
9. Які типи апроксимації існують?
10. Оформити будь-який графік (можна декілька) з використанням якомога більшої кількості інструментів графічних вікон.

ДОДАТОК Б
СПЕЦІАЛЬНА ГРАФІКА

Б1. Характеристика засобів спеціальної графіки пакету *MATLAB*

До найбільш часто застосованих засобів спеціальної графіки (див. help specgraph) відносяться

bar – Bar graph (стовпчикова діаграма);
barh – Horizontal bar graph.
stairs – Stairstep plot (ступінчатий графік);
stem – Discrete sequence or "stem" plot (решітчаста функція);
scatter – Scatter plot (точкова діаграма);
area – Filled area plot;
fill – Filled 2-D polygons;
pie – Pie chart (кругова діаграма);
hist – Histogram (гістограма);
rose – Angle histogram plot (кутова гістограма);
compass – Compass plot (векторна діаграма);
fplot – Plot function;
ezplot – Easy to use function plotter;
ezpolar – Easy to use polar coordinate plotter.

Стовпчикові діаграми досить часто застосовують для наочного порівняння деяких кількісних показників за роками, наприклад, народжуваність, продуктивність підприємств, кількість землетрусів, тощо. Іще їх зручно використовувати при відображенні випадкових процесів.

Кругові діаграми найчастіше застосовують для наочного відображення деяких показників у процентах, коли площа повного кола приймається за 100%, а площі секторів, пофарбовані у різні кольори, відображають частки цілого.

Ступінчаті графіки застосовують при зображенні цифрових сигналів, коли інформація дискретно змінюється у дискретні моменти часу.

Решітчасті функції відображають властивості ідеальних імпульсних сигналів.

Гістограми будують під час обробки результатів вимірювань. Для цього вектор вимірювань сортирують, діапазон результатів ділять на декілька рівних інтервалів і на кожному з них підраховують кількість вимірювань. Залежність кількості вимірювань на кожному інтервалі від діапазонів цих інтервалів, подану у вигляді стовпчикової діаграми, називають гістограмою. Гістограма дає уявлення про найбільш достовірний діапазон виміряних значень.

Зображення векторів комплексних чисел у полярній системі координат можна отримати функцією compass(z) або функцією compass(x,y), де x – вектор дійсних частин, а y – вектор уявних частин комплексних чисел.

Точкові діаграми використовують здебільш для побудови діаграм розсіювання, які ще називають діаграмами розкиду даних або скаттер-діаграмами.

Основні види спеціальних графіків, побудовані при виконанні програми

```
clc, close all
x=0:1:10;
y=(x-4).^2.+2;
for i=1:8
   subplot(4,2,i)
   switch i
      case 1, bar(x,y), xlim([0 10]), title('bar(x,y), xlim([0 10])')
      case 2, barh(x,y),ylim([0 10]), title('barh(x,y),ylim([0 10])')
      case 3, stairs(x,y), title('stairs(x,y)')
      case 4, pie(sqrt(y(6:10))), title('pie(sqrt(y(6:10)))')
      case 5, stem(x,y), title('stem(x,y)')
      case 6, scatter(x,y), title('scatter(x,y)')
      case 7, area(x,y), title('area(x,y)')
      case 8, fill(x,y,'g'), title('fill(x,y)')
   end
end
```

показані на рис. Б1.

Якщо функція, графік якої треба побудувати, описана у вбудованій *m*-функції або в *m*-функції користувача, то для її побудови можна використати функції fplot, ezplot та ezpolar.

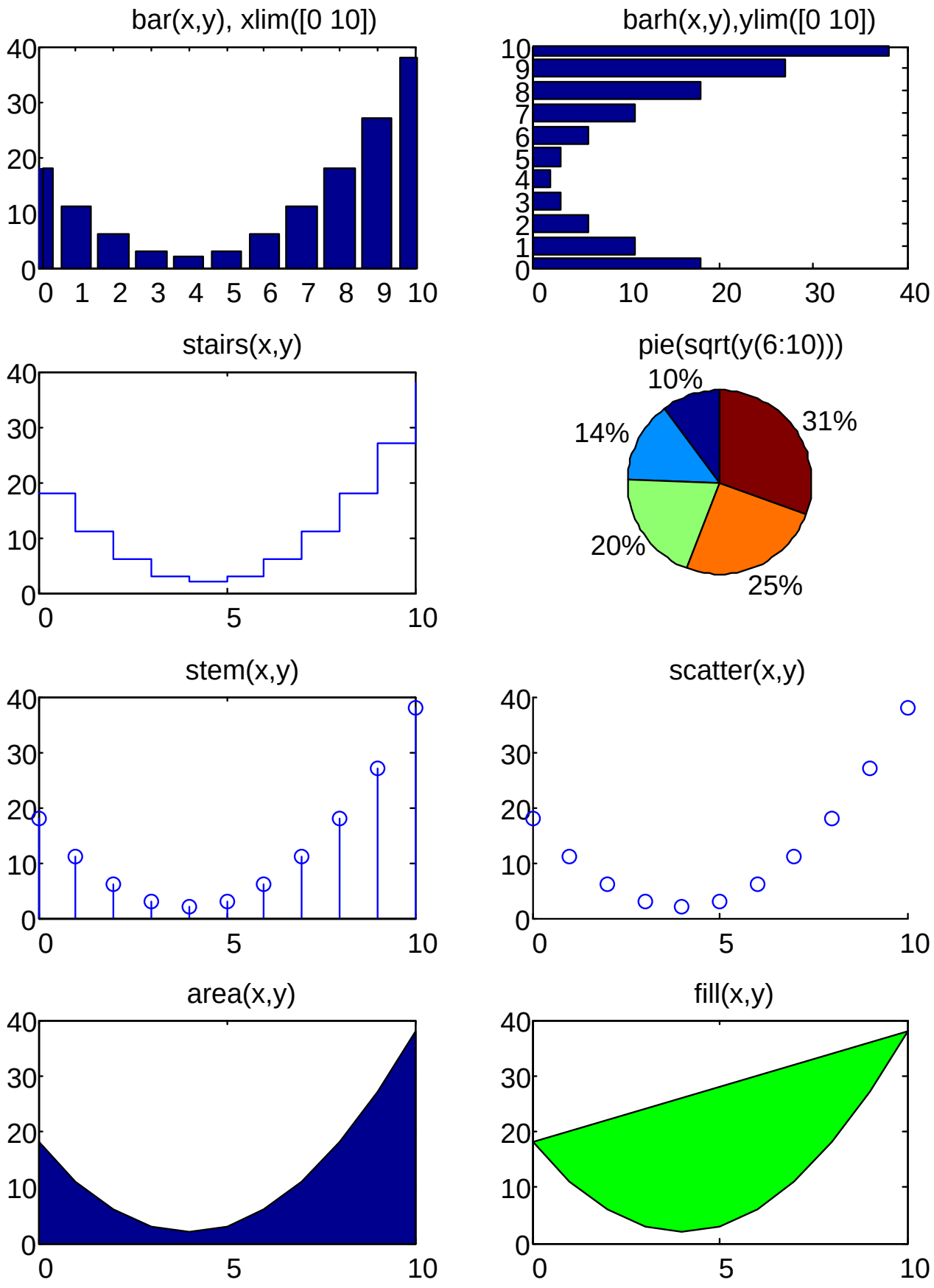


Рис. Б1. Види спеціальних графіків

Функцію fplot зазвичай застосовують у форматах fplot (fun, limits) та fplot (fun, limits, Style).

Функція ezplot має більше варіацій:

- ezplot (fun) будує графік функції fun(x) на інтервалі $-2\pi<x<2\pi$;
- ezplot (fun, [xmin, xmax]) будує графік функції fun(x) на інтервалі $x_{\min}<x<x_{\max}$;
- ezplot (fun2) будує графік функції fun2(x,y)=0 на інтервалах $-2\pi<x<2\pi$, $-2\pi<y<2\pi$;
- ezplot (fun2, [xmin, xmax, ymin, ymax]) будує графік функції fun2(x,y)=0 на інтервалах $x_{\min}<x<x_{\max}$, $y_{\min}<y<y_{\max}$;
- ezplot (funx, funy) будує параметрично визначений графік функції funy(t)(funx(t)) в діапазоні $0<t<2\pi$
- ezplot (funx, funy [tmin, tmax]) будує параметрично визначений графік функції funy(t)(funx(t)) в діапазоні $t_{\min}<t<t_{\max}$.

Функція ezpolar будує в полярних координатах графік функції fun(fi). Вона застосовується у варіантах ezpolar (fun) та ezpolar (fun,[a,b]). За замовчанням $0<\varphi<2\pi$.

Параметр fun у цих функціях може задаватися декількома способами:

- виразом ‘Expression’ або ім‘ям функції ‘FunName’ у вигляді рядка символів;
- посиланням на функцію @FunName;
- анонімною функцією @(x) Expression.

Для прикладу наведемо програму і результат її виконання (рис. Б.2).

```
for i=1:6,      subplot(3,2,i)
   switch i
      case 1, fplot(@tanh,[-2 2]), grid
      case 2, fplot('myfun_sg',[-20 20]), grid
      case 3, ezplot(@cos), grid
      case 4, ezplot('(x-4)^2',[0 10]), grid
      case 5, ezpolar(@(t) 1+cos(t))
      case 6, ezpolar('sin(7*fi/4)', [0 8*pi])
   end
```

```
function y=myfun_sg(x)
    y(:,1) = 200*sin(x(:))./x(:);
    y(:,2) = x(:).^2;
end
```

```
end
```

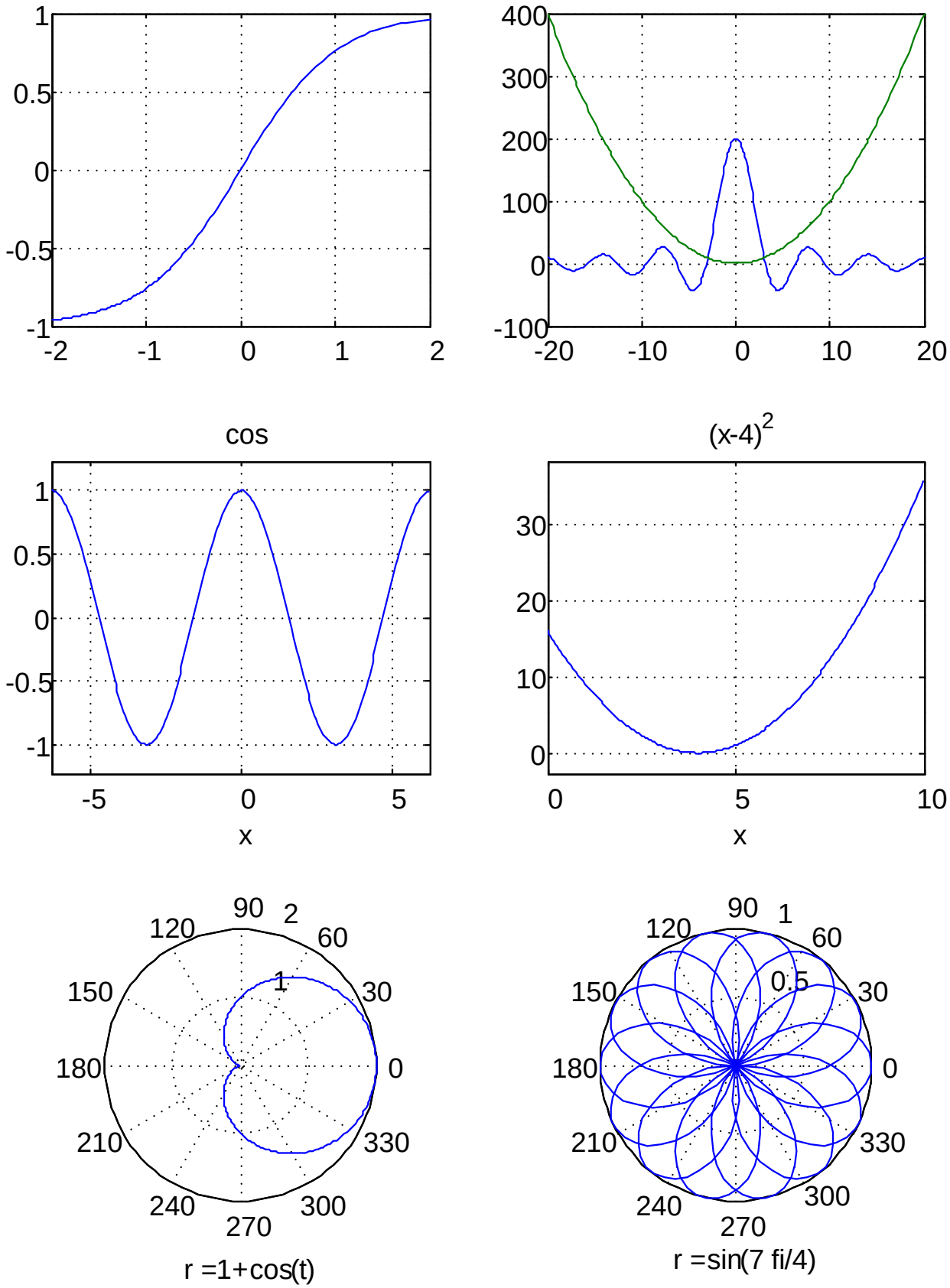


Рис. Б2. Приклади використання функцій fplot, ezplot та ezpolar

Б2. Питання та завдання для самокотролю

1. У якій папці знаходяться функції спеціальної графіки?
2. Як візуалізувати одномірний масив даних у вигляді стовпчикової діаграми, кругової решітчастої, точкової та ступінчатої функції?
3. Поясніть різницю між функціями area та fill.
4. Для чого застосовують функції fplot та ezplot? Яка між ними різниця?
5. Якими способами можна визначати функцію для операторів fplot, ezplot та ezpolar? Наведіть приклади.

ДОДАТОК В
ТРИВИМІРНА ГРАФІКА

В1. Загальна характеристика

З переліком функцій тривимірної графіки можна ознайомитися, виконавши команду help graph3D. Наведемо найбільш важливі з них:

```
Elementary 3-D plots.
  plot3        - Plot lines and points in 3-D space.
  mesh         - 3-D mesh surface.
  surf         - 3-D colored surface.
  fill3        - Filled 3-D polygons.

Color control.
  colormap    - Color look-up table.
  caxis        - Pseudocolor axis scaling.
  shading     - Color shading mode.
  hidden      - Mesh hidden line removal mode.
  brighten    - Brighten or darken color map.
  colordef    - Set color defaults.

Lighting.
  surfl       - 3-D shaded surface with lighting.
  lighting    - Lighting mode.
  material    - Material reflectance mode.
  specular    - Specular reflectance.
  diffuse     - Diffuse reflectance.
  surfnorm    - Surface normals.

Axis control.
  grid        - Grid lines.
  box         - Axis box.
  hold        - Hold current graph.
  subplot     - Create axes in tiled positions.
  xlim        - X limits.
  ylim        - Y limits.
  zlim        - Z limits.
```

```
Graph annotation.
  title      - Graph title.
  xlabel     - X-axis label.
  ylabel     - Y-axis label.
  zlabel     - Z-axis label.
  text       - Text annotation.
  gtext      - Mouse placement of text.
```

Інформацію про спеціальні засоби тривимірної графіки, можна отримати командою help specgraph, наприклад,

```
Specialized 3-D graphs.
  bar3           - 3-D bar graph.
  bar3h          - Horizontal 3-D bar graph.
  comet3         - 3-D comet-like trajectories.
…ezgraph3        - General purpose surface plotter.
  ezmesh         - Easy to use 3-D mesh plotter.
  ezmeshc        - Easy to use combination mesh/contour plotter.
  ezplot3        - Easy to use 3-D parametric curve plotter.
  ezsurf         - Easy to use 3-D colored surface plotter.
  meshc          - Combination mesh/contour plot.
  meshz          - 3-D mesh with curtain.
  pie3           - 3-D pie chart.
  ribbon         - Draw 2-D lines as ribbons in 3-D.
  scatter3       - 3-D scatter plot.
  stem3          - 3-D stem plot.
  surfc          - Combination surf/contour plot.
  trisurf        - Triangular surface plot.
  trimesh        - Triangular mesh plot.
  waterfall      - Waterfall plot.
```

При побудові тривимірних графіків можливо застосування майже усіх допоміжних графічних функцій, які використовуються при побудові двовимірних графіків (див. розділ 3), а також редагування графіків за допомогою функцій меню графічних вікон (див. розділ 4).

Тривимірні графіки мають набагато більше властивостей, ніж двовимірні, наприклад, «точка зору» (*Viewpoint control*), «розташування камери» (*Camera control*), підсвічування (*Light control*), але ці питання в даному посібнику не розглядаються.

В2. Побудова точок та ліній у тривимірному просторі

Для побудови точок та ліній у 3-вимірному просторі у середовищі пакета MATLAB використовують функцію plot3 (x,y,z), яка будує кусково-лінійну функцію, з'єднуючи між собою точки з координатами (x_i, y_i, z_i) для $i = 1, 2, ..., n$ відрізками прямих. Вектори x,y,z повинні мати однаковий розмір, наприклад, при виконанні операторів

```
t = linspace (0,8*pi,160);
plot3 (sin(t), cos(t), sqrt(t)),  grid on
xlabel ('sin(t)'),  ylabel ('cos(t)'),  zlabel ('sqrt(t)'),
title ('Спираль')
```

отримуємо графік спіралі, зображений на рис. В1.

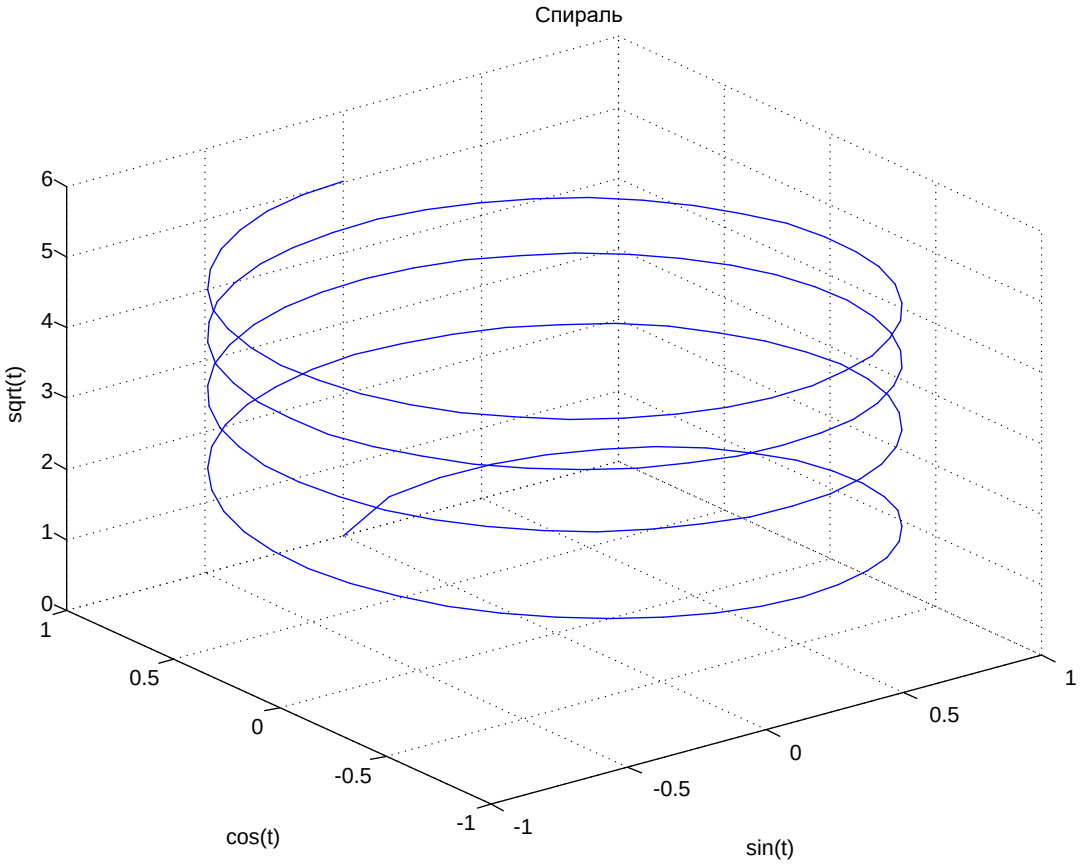


Рис. В1. Тривимірний графік спіралі

В3. Побудова каркасів та поверхонь у тривимірному просторі

Функції mesh (X,Y,Z), surf (X,Y,Z) і surfl (X,Y,Z) мають параметрами не вектори, а матриці і будують не лінії, а поверхні.

Щоб скористатися цими функціями, спочатку створюють вектори, що задають діапазони зміни значень абсцис і ординат, наприклад,

$$\mathbf{x} = [x_1 \quad x_2 \quad \ldots \quad x_n], \qquad \mathbf{y} = [y_1 \quad y_2 \quad \ldots \quad y_m]; \tag{B.1}$$

потім створюють 2 матриці розміром $m \times n$: матрицю $\mathbf{X}$ із m рядків вектору $\mathbf{x}$ і матрицю $\mathbf{Y}$ із n стовпчиків вектору $\mathbf{y}$:

$$\mathbf{X} = \begin{bmatrix} x_1 & x_2 & \ldots & x_n \\ x_1 & x_2 & \ldots & x_n \\ \vdots & \vdots & \ddots & \vdots \\ x_1 & x_2 & \ldots & x_n \end{bmatrix}, \qquad \mathbf{Y} = \begin{bmatrix} y_1 & y_1 & \ldots & y_1 \\ y_2 & y_2 & \ldots & y_2 \\ \vdots & \vdots & \ddots & \vdots \\ y_m & y_m & \ldots & y_m \end{bmatrix}. \tag{B.2}$$

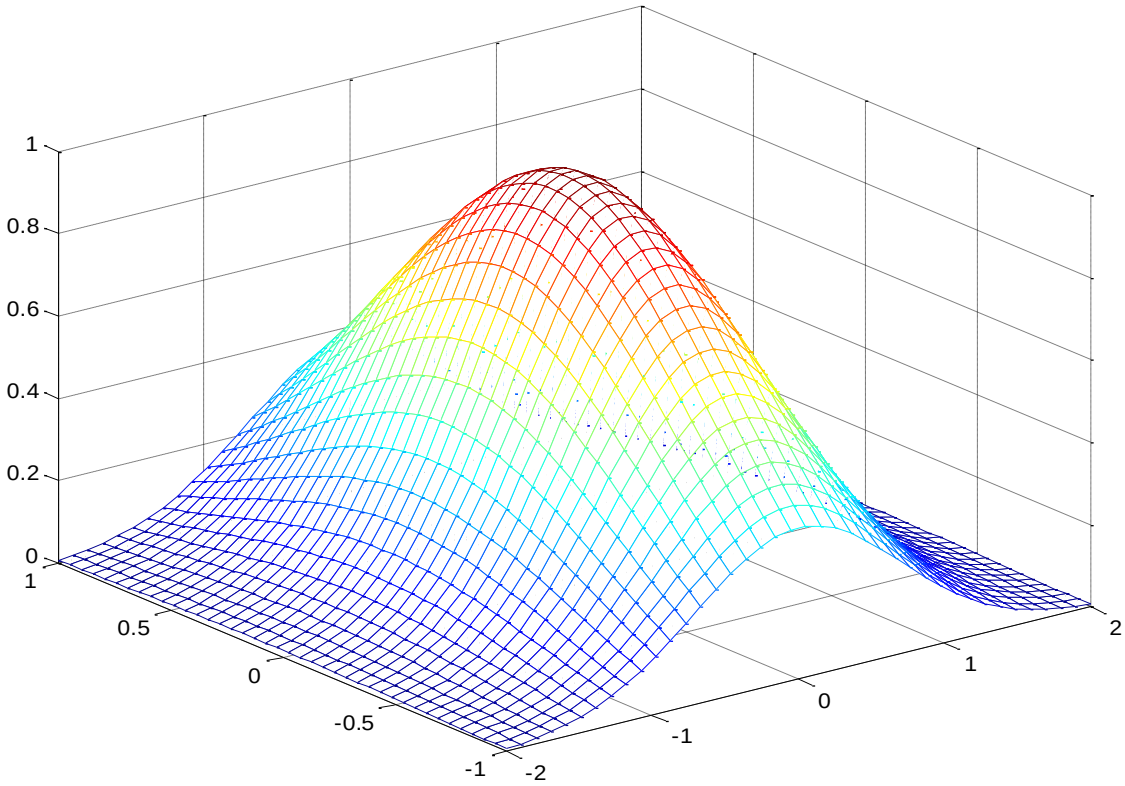
Після цього розраховують матрицю Z і будують графіки. Наприклад, при виконанні програми

```
x = linspace (-2,2,40);
y = linspace (-1,1,40);
[X,Y] = meshgrid (x,y);
Z = exp(-X.^2-Y.^2);
mesh (X,Y,Z),  grid on
figure,  surf (X,Y,Z),  grid on
```

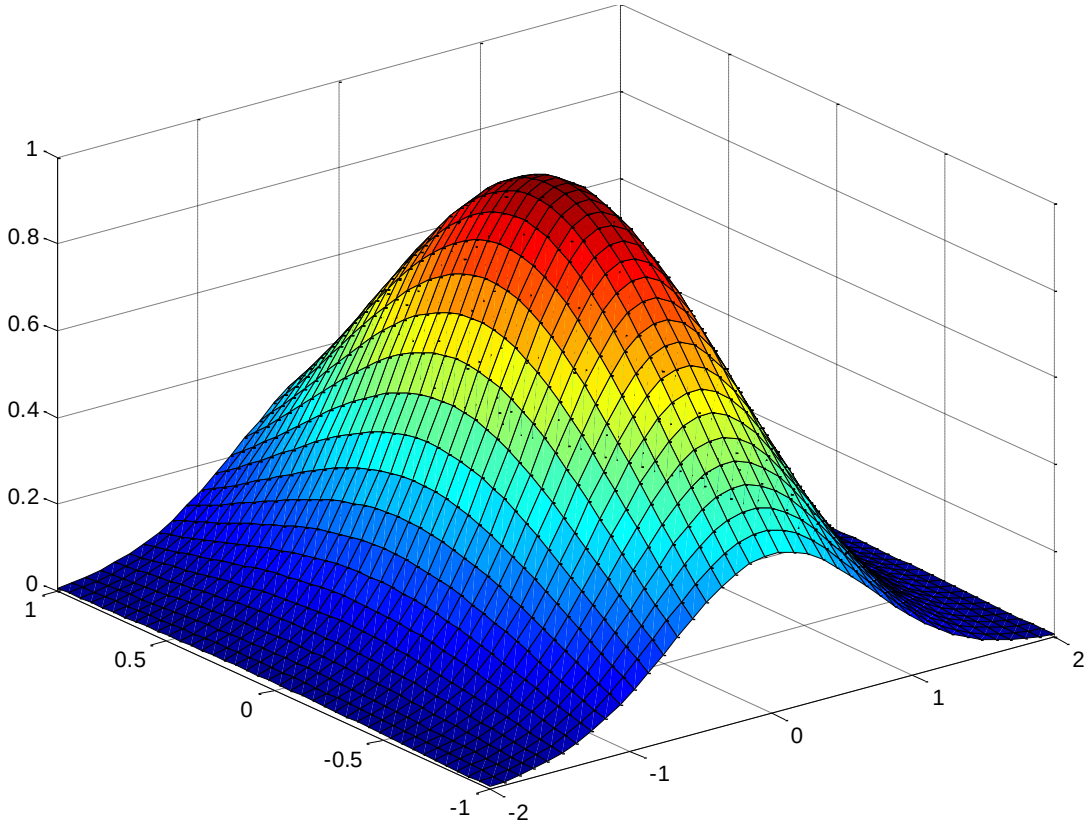
отримуємо графіки, подані на рис. В2. Вони демонструють різницю між двома досліджуваними графічними функціями.

Функція mesh будує каркасно-ребристу (*wireframe mesh*) поверхню, а функція surf – непрозору сітчасту поверхню.

Для кращого сприйняття "об‘ємності" зображення різних ребер або клітин автоматично фарбується у різні кольори. Крім того, здійснюється видалення невидимих ліній. Щоб створити ефект прозорості каркасно-ребристої поверхні, застосовують команду hidden off. Функція surfl зафарбовує сітчасту поверхню більш природно.



а)



б)

Рис. В2. Графіки залежності $z = e^{-x^2-y^2}$, побудовані функціями mesh (а) та surf (б)

Функція surf може застосовуватися в режимі інтерполяції тіні на гранях графіка shading interp. За замовчанням використовується стиль shading faceted, можна також стилем shading flat.

Команда colorbar будує поруч з графіком стовпчик з поточною кольоровою гамою, маркування якого демонструє прив‘язку кольору до значень аплікат z.

Функції surfc і meshc може будує не тільки відповідні поверхні, але й проекції на площину *xy* лінії однакового рівня.

Наприклад, при виконанні програми

```
clc, clear all, close all
[X,Y] = meshgrid ([-2:0.1:2]);
Z = X.*exp(-X.^2 -Y.^2);
surfc(X,Y,Z), shading flat
colorbar
```

отримуємо графік функції $z=xe^{-x^2-y^2}$ у вигляді непрозорої поверхні з проекціями , зображений на рис. В3.

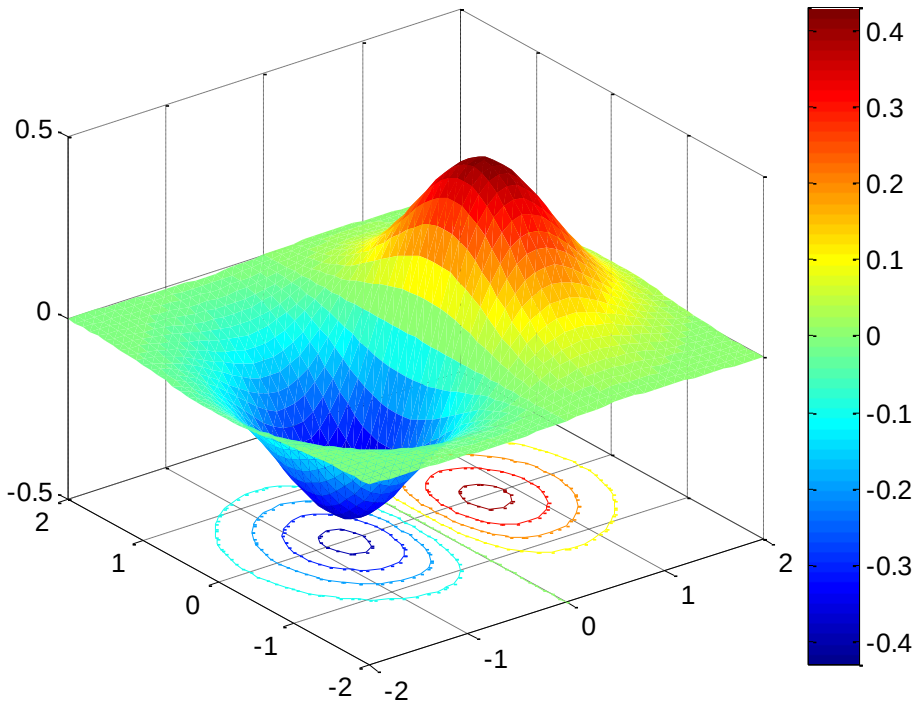


Рис. В3. Графік залежності $z=xe^{-x^2-y^2}$, побудований функцією surfc

Також існує можливість міняти кольорову гаму (палітру, карту кольорів) графіка за допомогою функції

colormap (MapName)

Перелік палітр можна подивитися за допомогою команди help graph3d:

Color maps.
hsv - Hue-saturation-value color map (кольори веселки).
hot - Black-red-yellow-white color map (чергування чорного, червоного, жовтого та білого кольорів).
gray - Linear gray-scale color map (відтінки сірого).
bone - Gray-scale with tinge of blue color map (сірі кольори з відтінком голубого).
copper - Linear copper-tone color map (відтінки міді).
pink - Pastel shades of pink color map (розові кольори).
white - All white color map (біла палітра).
flag - Alternating red, white, blue, and black color map (контрастне чергування червоного, білого, синього та чорного кольорів.
lines - Color map with the line colors.
colorcube - Enhanced color-cube color map.
vga - Windows colormap for 16 colors.
jet - Variant of HSV.
prism - Prism color map.
cool - Shades of cyan and magenta color map (відтінки голубого та фіолетового кольорів).
autumn - Shades of red and yellow color map (відтінки червоного та жовтого кольорів).
spring - Shades of magenta and yellow color map (відтінки пурпурного та жовтого кольорів).
winter - Shades of blue and green color map (відтінки голубого та зеленого).
summer - Shades of green and yellow color map (відтінки зеленогот а жовтого).

За замовчанням використовується палітра hsv, у якій кольори плавно змінюються від синього до червоного.

Кожна палітра уявляє собою матрицю інтенсивності rgb-кольорів розміром 64×3, яку можна побачити після введення імені палітри. Графічна інтерпретація інтенсивності трьох змішуваних кольорів виводиться командою

rgbplot (MapName)

В4. Основи контурної графіки

Контурна графіка використовується для побудови ліній одного рівня (наприклад, на географічних картах). Перелік *m*-функцій контурної графіки виводить команда help specgraph:

```
Contour and 2-1/2 D graphs.
  contour     - Contour plot.
  contourc    - Contour computation.
  contourf    - Filled contour plot.
  contour3    - 3-D Contour plot.
  clabel      - Contour plot elevation labels.
  ezcontour   - Easy to use contour plotter.
  ezcontourf  - Easy to use filled contour plotter.
  pcolor      - Pseudocolor (checkerboard) plot.
  voronoi     - Voronoi diagram.
```

Графіки можуть бути двовимірними (contour) и тривимірними (contour3). Для застосування функцій контурної графіки необхідно попереднє формування матриць X, Y, Z, оператором аналогічно тому, як це виконується при використанні функцій mesh і surf.

Звернення до основних функцій контурної графіки мають формат:

contour (X,Y,Z), contour3 (X,Y,Z)

За замовчанням весь діапазон зміни аплікат ділиться на 8 рівних відрізків і на графік виводяться 9 ліній.

Для того, щоб зрізи функції виконувалися в *n* рівновіддалених один від одного рівнях, використовують функції

contour (X,Y,Z,n), contour3 (X,Y,Z,n)

Для того, щоб зрізи функції виконувалися на заданих рівнях, параметр n замінюють вектором V, що складається з відповідних значень функції:

contour (X,Y,Z,V), contour3 (X,Y,Z,V)

Наприклад, якщо до попередньої програми додати оператори

figure, contour(X,Y,Z), grid on, figure, contour3(X,Y,Z,40)

то отримаємо графіки, зображені на рис. В5.

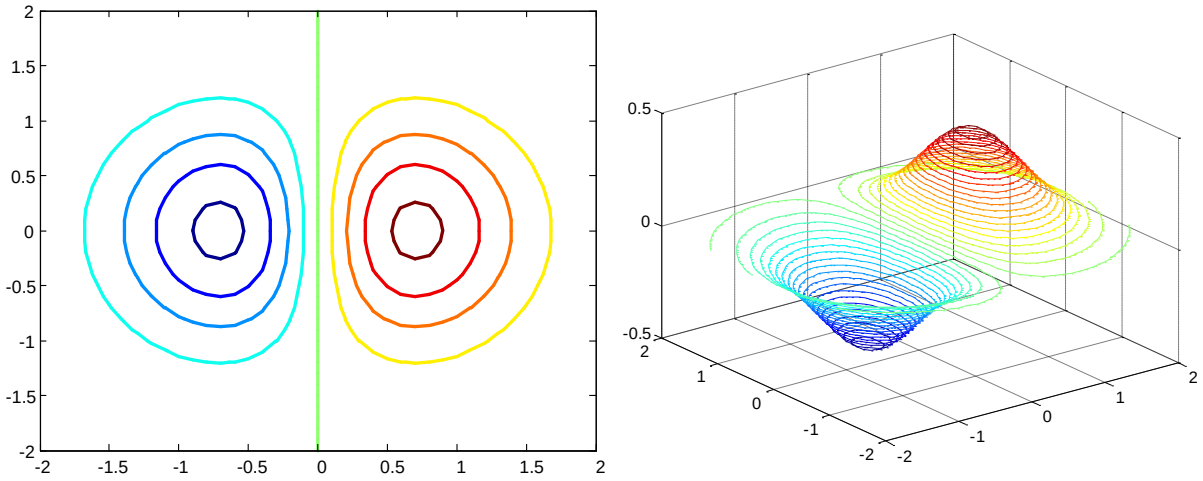


Рис. В5. Дво- та тривимірні контурні лінії функції $z=xe^{-x^2-y^2}$

Контурну графіку доцільно використовувати для визначення початкових наближень коренів систем рівнянь другого порядку та екстремумів двоаргументних функцій, як це буде показано у розділах 15 і 16.

В5. Питання та завдання для самоперевірки

1. У яких папках знаходяться функції тривимірної графіки?
2. Поясніть різницю між операторами plot3, mesh та surf.
3. Як створити матриці даних для операторів mesh та surf.
4. Які стилі зображення 3-вимірних поверхонь ви знаєте?
5. Як змінити кольорову палітру 3-вимірного графіка?
6. Що таке контурна графіка? Для чого її використовують?

Як вивести матрицю кольорів палітри та її графічну інтерпретацію**?**

ЗМІСТ