МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

О. Ю. Заковоротний, Т. О. Орлова, Д. В. Гриньов

КОМП’ЮТЕРНА МАТЕМАТИКА

Навчальний посібник

Затверджено
редакційно-видавничою
радою НТУ «ХПІ»,
протокол №1 від 15.02.2024 р.

Харків
НТУ «ХПІ»
2024

УДК 004.89
З 19

Рецензенти:

В. Д. Ковальов, д-р техн. наук, проф., Лауреат премії Кабінету Міністрів України, ректор Донбаської державної машинобудівної академії;

А. А. Коваленко, д-р техн. наук, проф., завідувач кафедри електронних обчислювальних машин, Харківський національний університет радіоелектроніки.

Рекомендовано Вченою радою Національного технічного університету «Харківський політехнічний інститут» як навчальний посібник для студентів зі спеціальності 123 «Комп'ютерна інженерія»
(протокол № 3 від 29.03.2024 р.)

Заковоротний О. Ю.

З 19 Комп'ютерна математика : навчальний посібник / О. Ю. Заковоротний, Т. О. Орлова, Д. В. Гриньов. – Харків : НТУ «ХПІ», 2024. – 375 с.

ISBN 978-617-05-0465-4

Розглянуто методи розв'язання задач вищої математики засобами комп'ютерної техніки, що дозволяє оволодіти навичками графічного подання розв'язків математичних задач, закріплює знання з таких галузей математики, як лінійна алгебра, векторний та спектральний аналіз, диференціальне та інтегральне числення. Посібник ілюструє теоретичний матеріал великою кількістю рисунків та прикладів, що допомагає краще зрозуміти та застосовувати наведений матеріал.

Призначено для студентів денної та заочної форм навчання зі спеціальності 123 «Комп'ютерна інженерія».

Іл. 160. Табл. 12. Бібліог. 18 назв.

УДК 004.89

ISBN 978-617-05-0465-4

РОЗДІЛ 1

СИСТЕМИ КОМП'ЮТЕРНОЇ МАТЕМАТИКИ

1.1. Огляд існуючих систем комп'ютерної математики

Комп'ютер, виправдовуючи свою назву (у перекладі з англ. "обчислювач"), працював як потужний калькулятор, що програмується, здатний швидко і автоматично (за веденою програмою) виконувати складні і громіздкі арифметичні та логічні операції над числами.

Успіхи обчислювальної математики і чисельні методи, що постійно вдосконалюються, дозволяють вирішити в такий спосіб будь-яку математичну задачу стосовно будь-якої галузі знань. Важливо, що результат обчислень представляється одним кінцевим числом в арифметичному вигляді, тобто з допомогою десяткових цифр. Іноді результат є безліччю (масивом, матрицею) таких чисел, але сутність уявлення від цього не змінюється - результат у вигляді кінцевого десяткового арифметичного числа.

Однак такий результат часто не задовольняв професійних математиків, і ось чому. Переважна більшість результатів нетривіальних математичних обчислень у класичній математиці традиційно записується у символьній формі: з використанням спеціальних загальновідомих чисел: π, е тощо, а ірраціональні значення – за допомогою радикала. Вважається, що інакше має місце важлива втрата точності.

Інший класичний приклад, що викликає зауваження математика – вираз, знайомий будь-якому школяру $\sin^2(\mathrm{x}) + \cos^2(x)$ завжди дорівнює одиниці, а в комп'ютері або буде спроба обчислити цей вираз (з неминучими помилками округлення), або буде видано повідомлення про невизначеність аргументу *x* і всі подальші дії будуть припинені.

Природно, за стрімким удосконаленням комп'ютерних систем, людині в комп'ютерних розрахунках захотілося більшого: чому не змусити комп'ютер виконувати перетворення традиційними для математики методами (дрібно-раціональні перетворення, підстановки, спрощення, рішення рівнянь, диференціювання тощо.).

Їх прийнято називати перетвореннями у символьному вигляді чи аналітичними перетвореннями, а результат отримувати не як раніше - як одне число, а у вигляді формули.

До цього моменту практично всі галузі людської діяльності виявилися охопленими кожна своїм власним математичним апаратом та обзавелися власними пакетами прикладного програмного забезпечення (ППЗ). При цьому всім знадобився універсальний математичний інструмент, орієнтований на широке коло користувачів, які не є професіоналами в математиці, ні програмістами, вихованими вузькоспеціальними, малозрозумілими для більшості кінцевих користувачів комп'ютерними мовами.

Це спричинило створення комп'ютерних систем символьної математики, розрахованих на широкі кола користувачів - непрофесіоналів у математиці. Так почалася з середини 60-х років XX століття ера систем комп'ютерної математики (СКМ).

Наприкінці 60-х років на вітчизняних ЕОМ серії "Світ", розроблених під керівництвом академіка В. Глушкова, була реалізована СКМ мовою програмування "Аналітик", що мала всі можливості символьних обчислень, втім, з вельми скромними, за нинішніми поняттями, характеристиками.

Звичайно, навіть найпростіші неінтелектуальні комп'ютерні математичні довідники представляють великий практичний інтерес - адже жодна здібна людина не в змозі вмістити у своїй голові всі математичні закони і правила, створені за багатовікову історію людства.

Дані про особливості існуючих СКМ наведено у табл. 1.1.

Таблиця 1.1 – Сучасні СКМ та їх можливості

СКМ	Призначення та можливості	Недоліки
Mathcad 13, 14	Система універсального призначення в основному для непрофесійних математиків та для освітніх потреб всіх рівнів. Продуманий інтерфейс представлення даних у традиційній математичній формі та дивовижна графіка на всіх етапах роботи, включаючи введення. Введення за допомогою вибору з панелей інструментів або меню практично без використання клавіатури. Потужний та вичерпний набір операторів та функцій. Безліч прикладів, електронних книг та бібліотек, готових розв'язань практичних завдань. Ядро символьних обчислень імпортовано із СКМ Maple. Надання серверних послуг професійного пакета. Легкість перенесення документа до	Досить примітивні засоби програмування. Дорожнеча електронних книг та бібліотек, наявність тільки англомовних версій самого пакету та додаткових бібліотек (книг). Ускладнена символьна обробка диференціальних рівнянь. Не створюється підсумковий виконуваний *. exe-файл; для запуску документа потрібна наявність пакету СКМ Mathcad. Труднощі при виконанні тригонометричних перетворень
Maple Version R4-R6	Університетська вища освіта та наукові розрахунки. Могутнє ядро символьних обчислень - можливості аналогічні СКМ Mathcad, що містить до 3000 функцій. Найпотужніша графіка. Зручна довідкова система. Засоби форматування документів	Підвищені вимоги до апаратних ресурсів. Відсутність синтезу звуків. Орієнтація на досвідчених користувачів та фахівців з математики. Усі недоліки аналітичних дій аналогічні СКМ Mathcad
Mathematica 5/7	Вища освіта та наукові розрахунки. Найбільш розвинена система символьної математики. Єдина СКМ, що забезпечує символьне розв'язання диференціальних рівнянь. Сумісність із різними комп'ютерними платформами. Унікальна тривимірна графіка. Підтримка синтезу звуку. Програмний синтез звуків Розвинені засоби форматування документів.	Високі вимоги до апаратних ресурсів. Надмірний захист від копіювання. Слабкий захист від некоректних завдань. Орієнтація на досвідчених користувачів. Введення завдань унікальною мовою функціонального програмування. Незвичайна індикація функцій запуску обчислень.
MATLAB 7.* та вище	Освіта (у тому числі технічна), наукові розрахунки, чисельне моделювання, та розрахунки, орієнтовані застосування матричних методів, у якій скаляр сприймається як матриця 1×1. Унікальні матричні засоби, безліч чисельних методів, описова (дескрипторна) графіка, висока швидкість обчислень, легкість адаптації до завдань користувача завдяки безлічі пакетів розширення системи. Розвинена мова програмування з можливостями об'єктно-орієнтованого програмування (ООП), сумісність з алгоритмічною мовою Java.	Високі вимоги до апаратних ресурсів. Слабкі можливості символьних обчислень. Відносно висока вартість. Введення завдань унікальною мовою програмування

Розглянемо внутрішню архітектуру СКМ. На рис. 1.1 представлено загальну програмну архітектуру.

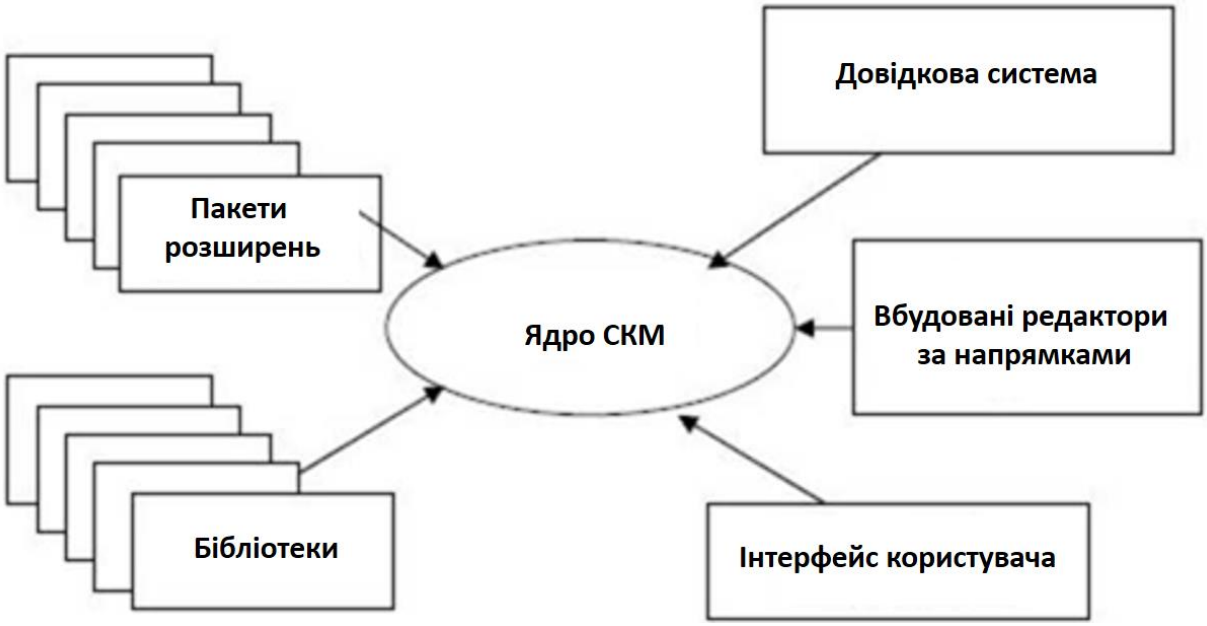


Рис. 1.1. Архитектура узагальненої СКМ.

Центральна частина – ядро системи СКМ реалізує алгоритм функціонування СКМ, забезпечує спільне функціонування її частин, організує прийом й інтелектуальну обробку запиту користувача та виклик потрібної процедури решения. У ядрі міститься велика кількість вбудованих функцій та операторів системи. Їхня кількість у сучасних СКМ може досягати багатьох тисяч. Наприклад, ядро системи Mathematica 4 містить дані більш ніж 5000 лише інтегралів, хоча для інтегрування використовуються лише кілька вбудованих функцій.

Пошук та виконання функцій та процедур, вбудованих у ядро СКМ, виконується швидко, якщо їх там не надто багато. Тому обсяг ядра обмежують, але до нього додають вбудовані в СКМ бібліотеки процедур та функцій, що використовуються відносно рідко. При цьому загальна кількість доступних користувачеві математичних функцій ядра та цих вбудованих бібліотек сягає багатьох тисяч.

Кардинальне розширення можливостей СКМ та їх пристосованість до потреб конкретних користувачів для поглибленого вирішення певного кола завдань (наприклад, задач теоретичної та прикладної статистики, векторного аналізу) досягається за рахунок встановлення зовнішніх пакетів розширення. Ці пакети, які купують окремо, роблять можливості СКМ практично безмежними.

Всі ці бібліотеки, пакети розширень та довідкова система сучасних СКМ (назвемо їх інструментами СКМ) містять не тільки і не просто знання в галузі математики, накопичені за багато століть її розвитку (цим нікого не здивуєш: саме такі можливості характерні для широко поширеного класу інформаційно-пошукових систем). Але ці інструменти надзвичайно автоматично і творчо використовують такі знання для вирішення завдань, де потрібно вибрати і вміти застосувати один, єдиний з багатьох десятків, неочевидний метод вирішення. Наприклад, СКМ можуть миттєво знайти невизначений інтеграл або відразу повідомити про неможливість його представлення елементарними функціями - завдання непросте, навіть для професійного математика. Не менш вражає і те, що якщо після отримання шуканої формули перейти до початку документа і задати параметрам, що входять до цієї формули, конкретні числові значення, миттєво буде отримано її чисельний результат. До складу будь-якої СКМ входить набір редакторів (на рис. 1.1 вони названі редакторами за напрямами): текстовий, формульний, графічний редактори, засоби підтримки роботи в мережі та HTML(XML)-засоби, пакети анімації та аудіозасоби.

Завдяки всім цим можливостям СКМ можуть бути віднесені до програмних продуктів найвищого на сьогоднішній день рівня – інтелектуального. Такі програми нині поєднуються терміном " бази знань ". Сучасні СКМ, на думку визнаних вчених, надає недосвідченому користувачеві можливості випускника математичного вишу в галузях чисельних методів розрахунку, математичного аналізу, теорії матриць та інших загальних розділів вищої математики, які дають змогу отримати конструктивні результати.

Звичайно, в абстрактних розділах математики, наприклад функціонального аналізу, СКМ поки що є мало корисними, але в прикладних завданнях, для яких СКМ і створювалися, такі розділи математики зазвичай не задіяні.

1.2. Знайомство з системою комп'ютерної математики MATLAB

Серед систем комп'ютерної математики, що бурхливо розвиваються, в першу чергу орієнтованих на чисельні розрахунки, особливо виділяється матрична математична система MATLAB. Через велику кількість пакетів розширення MATLAB, що поставляються з системою (у новітній реалізації MATLAB їх вже більше 85) ця система є і найбільшою з СКМ, орієнтованих на персональні комп'ютери. Обсяг її файлів перевищує 3,5 Гб. Система фактично стала світовим стандартом у галузі сучасного математичного та науково-технічного програмного забезпечення.

Ефективність MATLAB обумовлена насамперед її орієнтацією на матричні обчислення з програмною емуляцією паралельних обчислень та спрощеними засобами завдання циклів. Останні версії системи підтримують 64-розрядні мікропроцесори та багатоядерні мікропроцесори, що забезпечує найвищі показники швидкості обчислень і швидкості математичного імітаційного моделювання.

У MATLAB успішно реалізовані засоби роботи з багатовимірними масивами, великими і розрідженими матрицями та багатьма типами даних. Система пройшла багаторічний шлях розвитку від вузько спеціалізованого матричного програмного модуля, що використовується тільки на великих ЕОМ, до універсальної інтегрованої СКМ, яка орієнтована на персональні комп'ютери. MATLAB має потужні засоби діалогу, графіки та комплексної візуалізації обчислень.

Система MATLAB пропонується розробниками (корпорація The MathWorks) як лідируюча на ринку, в першу чергу на підприємствах військово-промислового комплексу, в енергетиці, в аерокосмічній галузі та в автомобілебудуванні, мова програмування високого рівня для технічних обчислень, що розширюється великою кількістю пакетів прикладних програм - Розширень. Найвідомішим із них стало розширення Simulink, що забезпечує блочне імітаційне моделювання різних систем та пристроїв. Але навіть без пакетів розширення MATLAB є потужним операційним середовищем для виконання величезної кількості математичних, науково-технічних розрахунків, обчислень, створення користувачами своїх пакетів розширення, бібліотек процедур та функцій. Нові версії системи мають вбудований компілятор і дозволяють створювати файли, що виконуються.

Типовий комплекс MATLAB + Simulink (рис. 1.2) містить інструментальні «скриньки» Toolboxes з великою кількістю пакетів розширення MATLAB та BloGksets для розширення можливостей системи візуально-орієнтованого блокового імітаційного моделювання динамічних систем Simulink. Вони купуються обрано та окремо від системи MATLAB + Simulink. У розробці пакетів розширення для MATLAB беруть участь багато наукових шкіл світу та провідні університети. Багато пакетів охоплюють великі напрями науки і техніки, такі як оптимізація відгуку нелінійних систем, моделювання пристроїв та систем механіки та енергетики, обробка сигналів та зображень, вейвлети, біоінформатика, генні алгоритми, нечітка логіка, нейронні мережі тощо.

MATLAB – одна з найстаріших, ретельно опрацьованих та перевірених часом систем автоматизації математичних та науково-технічних розрахунків, побудована на розширеному представленні та застосуванні матричних операцій. Це позначилось у назві системи - **MAT**rix **LAB**oratory (матрична лабораторія). Застосування матриць як основних об'єктів системи сприяє різкому зменшенню числа циклів, які дуже поширені при виконанні матричних обчислень звичайними мовами програмування високого рівня та полегшення реалізації паралельних обчислень.

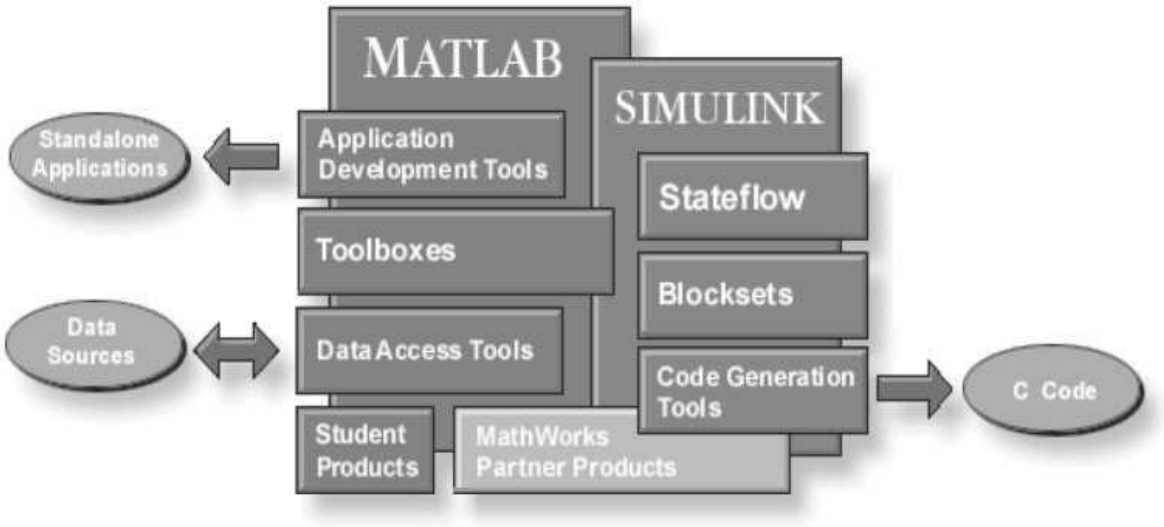


Рис. 1.2. Типовий комплекс MATLAB + Simulink.

Одним з основних завдань при створенні системи MATLAB завжди було надання користувачам потужної мови програмування, орієнтованої на технічні та математичні розрахунки та здатного перевершити можливості традиційних мов програмування, які багато років використовувалися для реалізації чисельних методів. При цьому особлива увага приділялася як підвищенню швидкості обчислень, так і адаптації системи до вирішення різноманітних завдань користувачів.

MATLAB реалізує три важливі концепції програмування:

• процедурне модульне програмування, засноване на створенні модулів – процедур та функцій;

• об'єктно-орієнтоване програмування, особливо цінне у реалізації графічних засобів системи;

• візуально-орієнтоване програмування, спрямоване створення засобів графічного інтерфейсу користувача GUI (Graphics User Interface).

Мова програмування MATLAB належить до класу інтерпретаторів. Це означає, що будь-яка команда системи розпізнається (інтерпретується) за її ім'ям (ідентифікатором) і негайно виконується в командному рядку, що забезпечує

легку перевірку частинами будь-якого програмного коду. Одночасно інтерпретуючий характер мови програмування MATLAB означає, що з перших рядків опису засобів цієї системи фактично описується її мова програмування.

Важливими перевагами системи є її відкритість та можливість доповнення. Більшість команд і функцій системи реалізовані як М-файлів текстового формату (з розширенням *.m*) і файлів мовою C/C++, причому всі файли можуть модифікуватися. Користувачеві дана можливість створювати не лише окремі файли, а й бібліотеки файлів для реалізації специфічних завдань. Будь-який набір команд у довідці можна виконати за допомогою команди Evaluate Selection контекстного меню правої клавіші миші.

1.2.1. Формати чисел

Для встановлення певного формату представлення чисел використовується команда:

```
>> format name
```

де `name` – ім'я формату. Для ілюстрації різних форматів розглянемо вектор, що містить два елементи-числа:

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

У різних форматах їх представлення матимуть такий вигляд:

```
format short  1.3333                  0.0000
format shorte 1.3333E+000             1.2345E-006
format long   1.333333333333338      0.000001234500000
format longe  1.333333333333338E+000 1.234500000000000E-006
format bank   1.33                    0.00
```

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

1.2.2. Константи та системні змінні

Константа – це попередньо визначене числове або символьне значення, представлене унікальним ім'ям (ідентифікатором). Числа (наприклад, 1, –2 та 1.23) є безіменними числовими константами.

Інші види констант у MATLAB прийнято називати системними змінними, оскільки, з одного боку, вони задаються системою під час її завантаження, а з іншого – можуть перевизначатися. Наприкла, можна встановити системної змінної eps інше значення, наприклад *eps*=0.0001. Проте важливо те, що значення за замовчуванням задаються відразу після завантаження системи. Тому невизначеними, на відміну звичайних змінних, системні змінні неможливо знайти ніколи.

Символьна константа – це ланцюжок символів, укладених в апострофи, наприклад:

```
'Hello my friend!'
'Привіт'
'2+3'
```

Якщо в апострофи вміщено математичний вираз, він не обчислюється і розглядається просто як ланцюжок символів. Так що '2+3' не повертатиме число 5. Однак за допомогою спеціальних функцій перетворення символьні вирази можуть бути перетворені на числові. Відповідні функції перетворення будуть розглянуті надалі.

1.2.3. Знищення визначених змінних

У пам'яті комп'ютера змінні займають певне місце, яке називається робочою областю (workspace). Для очищення робочої області використовується clear функція в різних формах, наприклад:

• clear – знищення визначень усіх змінних;

• clear *x* – знищення визначення змінної x;

• clear *a, b, c* – знищення визначень кількох змінних.

Знищена (стерта у робочій області) змінна стає невизначеною. Використовувати невизначені змінні не можна, і такі спроби супроводжуватимуться видачею повідомлень про помилку. Наведемо приклади завдання та знищення змінних:

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

Зверніть увагу на те, що спочатку вибірково стерта змінна *V*, а потім командою *clear* без параметрів стерти всі інші змінні.

Невизначені змінні використовуються під час виконання символьних обчислень. Спеціально, для виконання таких обчислень, система MATLAB не призначена. Однак, вони можливі за допомогою пакета розширення символьної математики *Symbolic Math*.

1.3. Оператори, функції та вирази системи комп'ютерної математики MATLAB

1.3.1. Оператори та їх пріоритет

Оператори, функції та вирази з ними становлять основу обчислювальних засобів будь-якої системи комп'ютерної математики.

Оператори вводяться спеціальними знаками у математичних висловлюваннях для зазначення дій, які мають виконуватися. Найбільш поширеними та однаковими у всіх системах комп'ютерної математики є арифметичні оператори: + (додавання), – (віднімання), * (множення), / (поділу) та зведення в ступінь ^. До операторів відносяться круглі (), квадратні [ ] і фігурні {} дужки, розділова точка, кома, двокрапка, точка з комою та ін. Оператори здійснюють ті чи інші дії над об'єктами, які називаються операндами.

Операнди можуть бути числами, константами, змінними та математичними виразами. Наприклад, у виразі (2 + 3) + 5 операторами є знаки + і дужки (), а операндами – константи 2 і 3 першого оператора складання і вираз (2 + 3) і константа 5 другого оператора складання. Аналогічно у вираженні (*a* + *b*) – *c* операндами будуть змінні *a, b* та *c*.

Слід зазначити, що у математичних виразах оператори мають загальноприйнятий пріоритет, тобто порядок виконання операторів у складному вираженні. Нижчим пріоритетом володіють оператори складання та віднімання. Вищий пріоритет у операторів множення, поділу, потім зведення у ступінь, виконання логічних операцій тощо. Для зміни пріоритету операцій у математичних виразах використовуються круглі дужки. Вирази в дужках виконуються насамперед незалежно від пріоритету операцій, що входять до них.

Поняття пріоритету полегшує однозначну інтерпретацію математичних виразів. Наприклад, у виразі 2+3*5 спочатку буде обчислено 3*5, а потім до результату додасться 2. У результаті буде обчислено значення 17. А у виразі

(2+3)*5 спочатку буде обчислено вираз у дужках (2+3), потім воно буде помножено на 5, тому результат буде 25. Таким чином, дужки дозволяють змінювати пріоритет операцій. Ступінь вкладення дужок у сучасних системах комп'ютерної математики не обмежена.

1.3.2. Функції та їх класифікація

Функція – об'єкт математичного виразу, який має унікальне ім'я (ідентифікатор), що виконує деяке перетворення своїх вхідних даних, представлених списком вхідних параметрів. Суть цього перетворення відповідає певній функціональній залежності значення, що повертається функцією, від вхідних параметрів функції. Наприклад, функція *sin*(*x*) повертає значення, яке є синусом вхідного параметра *x*. Таким чином, ознакою функції є повернення нею певного значення.

Вхідні параметри спочатку є формальними і представляються іменами деяких змінних. Особливістю функції є повернення її значення у відповідь на звернення до функції на ім'я із зазначенням фактичних параметрів у списку параметрів функцій. Фактичні параметри можуть бути різними константами, певними змінними і навіть математичними виразами, що обчислюються.

Наприклад, *sin*(*x*) є синтаксичною формою запису математичної функції синуса – *sin*(*x*). При цьому *x* – формальний параметр. А вже у виразі *sin*(1.0) числова константа 1.0 є фактичним параметром у вигляді дійсного числа, причому *sin*(1.0) повертає чисельне значення синуса кута 1 радіан. Функція *atan*2(*x, y*) є прикладом функції, що має список із двох формальних параметрів – *x* та *y*.

Як правило, в системах символьної математики важливо, як записаний фактичний параметр. Наприклад, число 1 або 1.0 є дійсним, про що вказує розділова точка. Якщо число представлене у вигляді 1, воно розглядається як ціле. Більшість систем символьної математики не обчислює вирази виду *sin*(l) чи *sin*(π/2), а виводить їх у вихідному вигляді. Це пов'язано з тим, що такий вид дає

значення функції набагато більше інформації, ніж її обчислене значення.

Завдяки властивості повернення значень функції вони застосовуються для побудови математичних виразів поряд з операторами. Наприклад, математичний вираз 2**sin*(*x*) містить функцію *sin*(*x*) та оператор множення *. Математичні вирази можуть бути як дуже простими (на зразок наведеного), так і дуже складними, що включають оператори інтегрування, диференціювання та інші спеціальні оператори та функції, а також складну багаторівневу систему дужок.

Функції зазвичай діляться на чотири типи:

- вбудовані в ядро системи визначені функції, або внутрішні функції;
- функції користувача, наприклад виду *f*(*x*, *y*, *z*):=(*x*^2+*y*^2)/*z*^2;
- бібліотечні функції, що викликаються з пакетів або бібліотек розширення системи;
- функції, задані у вигляді програмного модуля.

Крім того, функції можуть класифікуватися за характером вироблених ними перетворень вхідних параметрів. Вони діляться на алгебраїчні, тригонометричні, зворотні тригонометричні, гіперболічні, зворотні гіперболічні, спеціальні тощо.

Функції-процедури це функції матричної системи MATLAB, які можуть повертати безліч значень, причому найчастіше у вигляді масивів. Такі функції записуються у такому вигляді:

```
[R1,R2,...,Rn]=name_function(I1, I2,..., I3),
```

де у круглих дужках міститься список вхідних параметрів, а квадратних дужках - список вихідних параметрів. При виконанні такої функції змінні *R*1, *R*2, ..., *Rn* стають визначеними, набувають заданих функцією значення і їх можна використовувати в подальших розрахунках, наприклад, обчисленнях 2**R*1, *R*1+*R*2 і так далі. А сам по собі виклик такий функції у складі математичного виразу стає неможливим чи викликає повернення лише одного значення.

З погляду програміста, такі функції є процедурами, у яких використані операції присвоєння ряду змінних *R*1, *R*2, ..., *Rn* певних значень та дотримуються умови локалізації змінних зі списку вхідних параметрів у тілі процедури.

1.3.3. Математичні вирази

Математичні вирази – це складні (комбіновані) об'єкти, що складаються з операторів, операндів та функцій зі списками їх параметрів. Наприклад, у виразі (2 + 3) * *sin*(*x*) дужки ( ) та знаки + та * є операторами, константи 2 та 3 – операндами, *sin*(*x*) – вбудованою функцією, а *x* – вхідним параметром функції. Для оператора множення вираз (2 + 3) та функція *sin*(*x*), по суті теж вираз, що є операндами.

У системах для численних розрахунків математичні вирази записуються в природному вигляді, і в розборі їхньої структури немає особливої необхідності.

Інша справа системи символьної математики. Вони під час обчислень виразів перетворюються, тобто видозмінюються в міру виконання розрахунків. Це може призводити до дуже несподіваних наслідків, наприклад, коли найскладніше вираз спрощується до 0 або 1, а зовні нескладний вираз розгортається так, що не міститься на десятці сторінок екрана.

1.4. Оператори та функції матричної системи MATLAB

Система MATLAB виділяється з інших систем тим, що її оператори та функції мають операнди у вигляді векторів та матриць. Навіть операнд у вигляді одного числа сприймається як матриця розміру 1×1. Крім того, ця потужна система має безліч відмінностей у синтаксисі операторів та функцій. Наведемо визначення операторів та функцій даної системи у вигляді, як вони у ній використовуються.

1.4.1. Арифметичні оператори та функції

У табл. 1.2. наведено список арифметичних операторів із синтаксисом їх застосування. При цьому, кожен оператор має тотожну йому за призначенням функцію. Наприклад, оператор матричного множення має функцію *mtimes*(*M*1, *M*2).

Таблиця 1.2 – Список арифметичних операторів.

Функция	**Название**	**Оператор**	**Синтаксис**
plus	плюс	+	*M*1+*M*2
uplus	унарний плюс	+	+*M*
minus	мінус	–	*M*1–*M*2
uminus	унарний мінус	-	–*M*
mtimes	матричне множення	*	*M*1**M*2
times	поелементне множення масивів	.*	*A*1.**A*2
mpower	зведення у ступінь матриці	^	*M*1^x
power	поелементне зведення ступінь масиву	.^	*A*1^x
mldivide	зворотнє ділення матриць (з права на ліво)	\	*M*1*M*2
mrdivide	ділення матриць зліва направо	/	*M*l/*M*2
idivide	поелементне ділення масивів з права на ліво	.\	*A*1.*A*2
rdivide	поелементне ділення масивів з ліва на право	./	*A*1./*A*2
kron	тензорне множення Кронекера	*kron*	*kron*(*X*,*Y*)

Наведемо приклади застосування арифметичних операторів і функцій:

```
» A=[1 2 3];
» B=[4 5 6];
» B-A
ans = 3 3 3
» minus(B, A)
ans = 3 3 3
» A.^2
ans = 1 4 9
» power(A, 2)
ans = 1 4 9
» A.\B
ans =
4.0000 2.5000 2.0000
```

```
» ldivide(A, B)
ans =
4.0000 2.5000 2.0000
» rdivide(A, B)
ans =
0.2500 0.4000 0.5000
```

Відповідність операторам та командам системи MATLAB функцій є одним із важливих аспектів програмування у цій системі, що дозволяє використовувати у програмах одночасно як елементи операторного і функціонального програмування.

1.4.2. Оператори відносин

Оператори відносини у MATLAB записуються у вигляді, який наведено у табл. 1.3.

Таблиця 1.3 – Список арифметичних операторів.

Функція	**Назва**	**Оператор**	**Приклад**
eq	дорівнює	==	*x==y*
ne	не дорівнює	~=	*x~=y*
lt	менше ніж	<	*x<y*
gt	більше за	>	*x>y*
le	менше ніж або дорівнює	<=	*x<=y*
ge	більше ніж або дорівнює	>=	*x>=y*

Дані оператори виконують поелементне порівняння векторів або матриць однакового розміру та повертають значення логічної одиниці 1 (True), якщо елементи ідентичні, та значення логічного нуля 0 (False) в іншому випадку. Якщо операнди - дійсні числа, то застосування операторів та функцій відношення тривіально:

```
» eq(2,2)
ans = 1
» 2==2
```

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

Слід зазначити, що оператори <, <=, > і >= при комплексних операндах використовують порівняння лише дійсні частини операндів - уявні відкидаються. У той же час оператори == і ~= ведуть порівняння з урахуванням як дійсної, і комплексної частин операндов.

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

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

```
»M > [0 1;1 0]
ans =
0 1
0 1
```

Таким чином, спектр застосування операторів відношення в системі MATLAB ширший, ніж у звичайних мовах програмування, їх можна застосовувати не тільки до числа, але і (за особливими правилами) до елементів векторів, матриць та масивів іншого типу. Можливе застосування цих операторів і до символьних виразів:

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

У цьому випадку символи, що входять у вирази, представляються своїми кодами ASCII, які і порівнюються фактично. Рядки сприймаються як вектори з цими кодами. Все це треба враховувати при використанні керуючих структур мови програмування, які широко використовують оператори відносин.

1.4.3. Логічні оператори та функції

Логічні оператори та відповідні їм функції служать для поелементних логічних операцій над елементами однакових за розміром масивів (табл. 1.4.).

Таблиця 1.4 – Список арифметичних операторів

Функція	**Назва**	Позначення
and	Логічне І (*and*)	&
or	Логічне АБО (*or*)	|
not	Логічне НІ (*not*)	~
xor	Виключне АБО (*exclusive*)	
any	Вірно, якщо всі елементи вектора дорівнюють нулю	
all	Не вірно, якщо всі елементи вектора не дорівнюють нулю	

Робота цих операторів пояснюється наведеними нижче прикладами:

```
» A=[1 2 3];
» B=[1 0 0];
» and(A, B)
ans = 1 0 0
» or(A, B)
ans = 1 1 1
» A&B
ans = 1 0 0
»A|B
ans = 1 1 1
» not(A)
ans = 0 0 0
» not(B) ans = 1 1 1
»~B ans = 0 1 1
» xor(A,B) ans = 0 1 1
» any(A) ans = 1
» all([0 0 0])  ans = 0
» all(B)  ans = 0
» and('abc','O12')  ans = 1 1 1
```

Зверніть увагу, що аргументами логічних операторів можуть бути числа та рядки. При аргументах - числах логічний 0 відповідає нулю, а будь-яке відмінне від нуля число сприймається як логічна одиниця 1. При рядках діє зазначене правило - порівнюються ASCII коди символів.

1.5. Спеціальні символи MATLAB

До класу операторів у системі MATLAB належать деякі спеціальні символи. Нижче наведено опис їхнього повного набору.

: (двокрапка) – формування векторів та підматриць. Символ : один з операторів, що найчастіше використовуються в системі MATLAB. Він застосовується для формування векторів та виділення з них підвекторів. Оператор: використовує такі правила для індексування векторів:

j : *k* – те саме, що і [*j*, *j*+l, ..., *k*];

j : *k* – порожній вектор, якщо *j* > *k*;

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

j : *i* : *k* – порожній вектор, якщо *i* > 0 і *j* > *k* або якщо *i* <0 і *j* < *k* де *i*, *j* та *k* – скалярні величини.

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

A(: *j*) – це *j*-ий стовпець з *A*.

A(*i*, :) – це *i*-ий рядок з *A*.

A(:, :) – еквівалент двомірного масиву. Для матриць це аналогічно *A*.

A(*j* : *k*) – це *A*(*j*), *A*(*j*+l), …, *A*(*k*).

A(: *j*: *k*) – це *A*(:,*j*), *A*(:,*j*+l), …, *A*(:,*k*).

A(:, :, *k*) – це *k*-а сторінка тривимірного масиву *A*.

A(*i*, *j*, *k*, :) – вектор чотиривимірного масиву *A*. Вектор включає *A*(*i*, *j*, *k*, 1), *A*(*i*, *j*, *k*, 2), *A*(*i*, *j*, *k*, 3) і так далі.

A(:) – записує усі елементи масиву *A* у стовпець.

( ) (круглі дужки) – використовуються для завдання порядку виконання операцій в арифметичних виразах, вказівки послідовності аргументів функції та вказівки індексів елементів векторів або матриць. Якщо *X* і *V* – вектори, то *X*(*V*) можна записати як [*X*(*V*(1)), *X*(*V*(2)), ..., *X*(*V*(*n*))]. Елементи вектора *V* повинні бути цілими числами, щоб використовуватись для індексування елементів. Помилка генерується в тому випадку, якщо індекс елемента менше одиниці або більше, ніж *size*(*X*). Такий принцип індексування дійсний і для матриць. Якщо вектор *V* має *m* компонентів, а вектор *W* - *n* компонентів, то *A*(*V*, *W*) буде матрицею розміру *m*×*n*, сформованою з елементів матриці *A*, індекси якої елементи векторів *V* і *W*.

[ ](квадратні дужки) – використовуються для формування векторів та матриць. Наприклад [6.9 9.64 *sqrt*(–1)] – вектор, що містить три елементи, розділених пробілами. [6.9, 9.64, *i*] – такий самий вектор, а [1+*j* 2–*j* 3] і

[1 +*j* 2 –*j* 3] – різні вектори: перший містить три елементи, а другий п'ять. [11 12 13; 21 22 23] – матриця розміру 2×3. Крапка з комою розділяє перший і другий рядки.

A = [ ] – зберігає порожню матрицю в *A*.

A(*m*, :) = [] – видаляє рядок *m* із матриці *A*.

A(:,*n*) = [] – видаляє стовпець *n* із матриці *A*.

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

. (десяткова точка) – використовується для відокремлення дробної частини чисел від цілої. Наприклад: 314/100, 3.14 та .314*e*1 – одне й те ж число.

. (Виділення поля структури) – виділення поля структури. Наприклад, *A*.(*field*) та *A*(*i*).*field*, де *A* – структура, виділення поля структури з ім'ям "*field*".

.. (батьківський каталог) – перехід по дереву каталогів на один рівень вгору.

... (продовження) – три або більше точок наприкінці рядка вказують на продовження рядка.

, (кома) – використовується для поділу індексів елементів матриці та аргументів функції та для відділення операторів мови MATLAB. При відділенні операторів у рядку кома повинна замінюватись на точку з комою для недопущення виведення на екран результату обчислень.

; (крапка з комою) – використовується всередині круглих дужок для поділу рядків і для недопущення виведення на екран результату обчислень.

% (знак відсотка) – використовується для вказівки логічного кінця рядка. Текст після знака відсотка сприймається як коментар і ігнорується.

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

= (Знак присвоювання) – використовується для присвоєння значень в арифметичних виразах.

' (поодинока лапка) – текст у лапках представляється як вектор символів з компонентами, що є ASCII кодами символів. Поодинока лапка всередині тексту позначається двома лапками. Наприклад:

```
» a = 'привіт''друзям'
a =
привіт друзям
```

'(транспонування матриці) – транспонування матриць, наприклад *A*' – транспонована матриця *A*. Для комплексних матриць транспонування доповнюється комплексним поєднанням.

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

[,] (Горизонтальна конкатенація). Так, [*A*, *B*] – горизонтальна конкатенація (об'єднання) матриць *A* та *B*, які повинні мати однакове число рядків. [*AB*] діє аналогічно. Горизонтальна конкатенація може бути використана для будь-якого числа матриць в межах одних дужок [*A*, *B*, *C*]. Горизонтальна та вертикальна конкатенації можуть бути об'єднані разом [*A*, *B*; *C*].

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

( ), { } (Привласнення підмасиву). Так, *A*(*I*)=*B* надає значення елементів масиву елементам масиву *A*, які визначаються вектором індексів *I*. Масив *B* повинен мати однакове число елементів з масивом *I* або може бути скаляром.

A*(*I, J*) = *B – надає значення масиву *B* елементам прямокутної підматриці *A*, які визначаються векторами індексів *I* і *J*. Масив повинен мати length(*I*) рядків і *length*(*J*) стовпців.

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

1.6. Системні змінні та константи MATLAB

Як зазначалося раніше, до складу об'єктів MATLAB входить ряд системних змінних та констант, значення яких встановлюються системою під час її завантаження або автоматично формуються у процесі обчислень. Нижче наведено всі ці об'єкти.

ans – результат виконання останньої операції. Змінна *ans* створюється автоматично, коли не визначені вихідні аргументи будь-якого оператора. Наприклад:

```
» cos([0:2*pi])
ans =
1.0000 0.5403 -0.4161 -0.9900 -0.6536 0.2837 0.9602
```

str = computer – повертає рядок з інформацією про тип комп'ютера, на якому встановлено систему MATLAB.

[*str, maxsize*] = *computer* – повертає ціле число *maxsize*, що містить максимально допустиме число елементів матриці для цієї версії MATLAB. Наприклад:

```
» [str,maxsize] = computer
str =
PCWIN
maxsize = 268435455
```

eps **–** повертає інтервал між числом 1.0 та наступним найбільшим числом з плаваючою комою, яке сприймається як відмінне від 1.0. Значення eps визначає заданий за замовчуванням поріг для функцій pinv і rank, а також деяких інших функцій. На машинах з плаваючою арифметикою *eps* = 2^{-52}, що становить приблизно 2.22*e*-16. Наприклад:

```
» eps
ans =
2.2204e-016
```

f = flops – повертає загальну кількість операцій з плаваючою комою. *flops*(0) скидає лічильник операцій на нуль. Наприклад:

```
» f = flops
f =
8
```

i **–** уявна одиниця або *sqrt*(–1), яка використовується для введення комплексних чисел. Так як *i* – функція, вона може бути скасована і використовувати як змінна. Це дозволяє використовувати її як індекс, наприклад, у циклі *for*. При потребі символ *i* можна використовувати без знака множення при комплексній константі. Як уявна одиниця можна також використовувати символ *j*. Наприклад:

```
» w=3+5i
w =
3.0000 + 5.0000I
```

Inf **-** повертає позитивну нескінченність. Наприклад:

```
» 4/0
Warning: Divide by zero.
ans =
Inf
```

***Inputname* (*argnum*) –** повертає назву змінної робочої області вікна, що відповідає номеру аргументу argnum. Ця команда може використовуватися лише усередині тіла функції. Якщо вхідний аргумент не має назви (наприклад, якщо це - вираз замість змінної), команда inputname повертає порожній рядок ('').

j* –** уявна одиниця. Символ *j* можна використовувати як альтернативну уявну одиницю. Як уявна одиниця (або sqrt(–1)) ***j використовується для введення комплексних чисел. Т.к. *j* - функція, вона може бути скасована та використовуватися як змінна. Це дозволяє використовувати її як індекс, наприклад, у циклі *for*. При необхідності символ *j* можна використовувати без знака множення, коли буде задана комплексна константа. Наприклад:

```
» s=4-3j
s =
4.0000 - 3.0000!
```

***NaN* –** повертає нечислову величину, наприклад, операції, які мають невизначені числові результати. Наприклад:

```
» s=0/0
Warning: Divide by zero.
s =
NaN
```

***msg = nargchk*(*low, high, number*)** – повертає повідомлення про помилку, якщо число менше ніж *low* або більше ніж *high*, інакше повертається порожній рядок. Функція *nargchk* часто використовується всередині М-файлу для перевірки правильності числа аргументів. Наприклад:

```
» msg = nargchk(4,9,5)
msg =
[]
» msg = nargchk(4,9,2)
msg =
Not enough input arguments.
```

nargin – повертає кількість аргументів функції. Всередині тіла М-файлу функції *nargin* і *nargout* вказують відповідно число вхідних або вихідних параметрів (аргументів), заданих користувачем. Поза тілом М-файлу функції *nargin* і *nargout* показують, відповідно, число вхідних або вихідних параметрів цієї функції. Число аргументів негативне, якщо функція має змінну кількість аргументів.

***nargin*('*fun*')** - повертає кількість оголошених входів для функції *fun* М-файлу або –1, якщо функція містить змінну кількість вхідних аргументів.

nargout – повертає кількість вихідних параметрів, визначених функції.

***nargout*('*fun*')** – повертає кількість оголошених виходів для функції *fun* М-файлу.

pi – число π (відношення довжини кола до її діаметру). Наприклад:

```
» pi
ans =
3.1416
```

realmax – повертає найбільше число у форматі з плаваючою комою, що подається на конкретному комп'ютері. Будь-яке значення відповідає системної змінної inf, тобто машинної нескінченності Наприклад:

```
» n = realmax
n =
1.7977e+308
```

realmin – повертає найменше нормалізоване позитивне число у форматі з плаваючою комою, що подається на конкретному комп'ютері. Будь-яке менше сприймається як нуль. Наприклад:

```
» n = realmin
n =
2.2251 e-308
```

***varargout* = *foo*(*n*) –** повертає список вихідних параметрів змінної довжини функції *foo*.

y = *function bar*(*varargin*) - приймає змінну кількість аргументів на функцію *bar*.

Функції *varargin* та *varargout* використовуються тільки всередині М-файлу функції для завдання довільних аргументів функції. Ці змінні повинні бути останніми у списку входів або виходів, а її написання допускається лише малими літерами.

Контрольні запитання:

1. Які сучасні системи комп'ютерної математики використовуються у наш час, які у них є функціональні можливості та недоліки?
2. Наведіть складові типової архітектури системи комп'ютерної математики та опишіть їх призначення?
3. Наведіть складові архітектури системи комп'ютерної математики MATLAB та опишіть їх основні функціональні можливості?
4. Які формати чисел існують у системі комп'ютерної математики MATLAB, чим вони відрізняються та в чому принципова відмінність обчислень у різних форматах?
5. Які типи змінних та констант використовуються у системі комп'ютерної математики MATLAB, у чому їх різниця і особливості застосування?
6. Які оператори використовуються у системі комп'ютерної математики MATLAB та які пріоритети існують при їх застосуванні?
7. Наведіть класифікацію функції системи комп'ютерної математики MATLAB та поясніть різницю між ними?
8. Перелічите арифметичні оператори та функції, що існують у системі комп'ютерної математики MATLAB та наведіть приклади їх застосування?

9. Перелічите оператори відносин, що існують у системі комп'ютерної математики MATLAB та наведіть приклади їх застосування?

10. Перелічите логічні оператори та функції, що існують у системі комп'ютерної математики MATLAB та наведіть приклади їх застосування?

11. Які спеціальні символи використовуються у системі комп'ютерної математики MATLAB та поясніть особливості їх застосування?

РОЗДІЛ 2

ЕЛЕМЕНТАРНІ ФУНКЦІЇ СИСТЕМИ КОМП'ЮТЕРНОЇ МАТЕМАТИКИ MATLAB

Орієнтація математичної системи MATLAB на матричні (і векторні) операції веде до ряду тонкощів у завданні елементарних функцій та їх застосуванні. Вони пов'язані з тим, що аргументами таких функцій, при використанні системи MATLAB, є не поодинокі змінні, а матриці та вектори. Тому і результат, що повертається, може бути вектором-рядком, вектором-стовпцем, матрицею або масивом. Це дещо ускладнює опис навіть найпростіших функцій.

Елементарні функції СКМ представлені згрупованими за функціональним призначенням. Усі функції можуть використовуватися в конструкції виду *y = func* (*x*), де *func* – ім'я функції. Зазвичай у такій формі задається інформація про функції у системі MATLAB. Ми, однак, використовуватимемо для функцій, що повертають одиночний результат, простішу форму *func*(*x*). Форма [*y*, *z*, ...] = *func* (*x*, ...) використовуватиметься лише у випадках, коли функція повертає множинний результат, вказаний у квадратних дужках.

На прикладі системи MATLAB ми наведемо визначення більшості спеціальних математичних функцій.

2.1. Алгебраїчні та арифметичні функції

У системі MATLAB визначено такі арифметичні функції:

***factor*(*n*)** – повертає вектор-рядок, що містить прості множники числа *n*;

***gcd* (*A*, *B*)** – повертає масив, що містить найбільші спільні дільники відповідних елементів масивів цілих чисел *A* та *B*;

[*G, C, D*] = *gcd* (*A, B*) – повертає масив найбільших загальних дільників *G* та масивів *C* і *D*, які задовольняють рівняння: *A*(*i*).**C*(*i*) + *B*(*i*).**D*(*i*) = *G*(*i*);

***lcm*(*A, B*)** – повертає вектор найменших загальних кратних відповідних позитивних елементів цілих масивів *A* і *B* однакового розміру;

***primes*(*n*)** – повертає вектор рядок простих чисел, менших або рівних *n*.

Приклади застосування цих функцій:

```
» f = factor(221)
»f =13 17
» A=[2 6 9];
» B=[2 3 3];
» gcd(A, B)
ans = 2 3 3
»[G,C,D]=gcd(A, B)
G = 2 3 3
C = 0 0 0
D = 1 1 1
» A=[1 3 5 4];
» B=[2 4 6 2];
»lcm(A, B)
ans = 2 12 30 4
» p = primes(25)
P = 2 3 5 7 11 13 17 19 23
```

Є три функції для обчислення логарифмів: *log*(*X*) – натуральних, *log*2(*X*) – за основою 2 та *log*10(*X*) – за основою 10. Логарифм обчислюється для кожного елемента масиву *X*:

```
»X=[1.2 3.34 5 2.3];
» log(X)
ans = 0.1823 1.2060 1.6094 0.8329
» X=[2 4.678 5; 0.987 1 3];
»[F,E] = log2(X)
F =
0.5000 0.5847 0.6250
0.9870 0.5000 0.7500
E =
2 3 3
```

```
0 1 2
»X=[1.4 2.23 5.8 3];
» log10(X)
ans = 0.1461 0.3483 0.7634 0.4771
```

Для обчислення ступенів та квадратного кореня служать такі функції:

*pow*2(*Y*) – повертає масив, де кожен елемент є 2^Y;

*pow*2(*F*, *E*) – обчислює $X=F*2^E$ для відповідних елементів *F* і *E* з аргументами як масивів дійсних і цілих чисел, відповідно;

sqrt(*X*) – повертає квадратний корінь кожного елемента масиву *X*. Для негативних та комплексних елементів *X* функція *sqrt*(*X*) обчислює комплексний результат. Наприклад:

```
» d=pow2(pi/4, 2)
d = 3.1416
» A=[25 21.23 55.8 3];
» sqrt(A)
ans = 5.0 4.6076 7.4699 1.7321
```

2.2. Функції числової апроксимації, округлення та знаку

Ряд спеціальних функцій служить для виконання операцій апроксимації чисел, округлення числових даних і аналізу їх знака.

[*N, D*] = *rat*(*X*) – повертає масиви *N* і *D*, так що *N*./*D* апроксимує елементи масиву *X* з точністю l.*e*-6**norm*(*X*(:), l);

[*N, D*] = *rat*(*X, tol*) – повертає масиви *N* і *D*, так що *N*./*D* апроксимує елементи *X* з точністю *tol*;

***rat*(*X*)** без вихідних аргументів просто видає на екран ланцюговий дріб;

***rats*(*X, strlen*)** – повертає ряд, отриманий шляхом спрощеної раціональної апроксимації елементів масиву *X*. *strlen* – довжина рядка, що повертається. Функція повертає знак " * ", якщо отримане значення не може бути надруковане у рядку, довжина якого задана значенням *strlen*. Типово *strlen* =13;

***fix*(*A*)** – повертає масив *A* з елементами, округленими до найближчого до нуля цілого числа. Для комплексного *A* дійсні та уявні частини округляються окремо;

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

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

***rem*(*X*,*Y*) –** повертає *X* - *fix*(*X*./*Y*).**Y*, де *fix*(*X*./*Y*) – ціла частина від *X*/*Y*;

***round*(*X*)** – повертає округлені до найближчого цілого елементи масиву *X*. Для комплексного *X* дійсні та уявні частини округляються окремо;

***sign*(*X*)** – повертає масив даних про знаки елементів *X* тієї ж мірності як і *X* (1 – якщо відповідний елемент більше 0; 0 – якщо відповідний елемент дорівнює 0; –1 – якщо відповідний елемент менше 0). Для комплексних ненульових *X*, *sign*(*X*) = *X*./*abs*(*X*).

Приклади, наведені нижче, ілюструють застосування цих функцій:

```
»[g,j]=rat(pi,1e-10)
g =312689
j = 99532
» A=[1/3 2/3; 4.99 5.01]
A =
0.3333 0.6667
4.9900 5.0100
» fix(A)
ans =
0 0
4 5
» A=[-1/3 2/3; 4.99 5.01]
A=
-0.3333 0.6667
4.9900 5.0100
» floor(A)
```

```
ans =
-1 0
4 5
»a=-1.789;
» ceil(a)
ans =
-1
» a=-1.789+i*3.908;
» ceil(a)
ans =
-1.0000 + 4.0000i
» X=[25 21 23 55 3];
» Y=[4 8 23 6 4];
» rem(X,Y)
ans =
1 5 0 1 3
» X=[5.675 21.6+4.8974*i 2.654 55.8765];
» round(X)
ans =
6.0000 22.0000 + 5.0000i 3.0000 56.0000
» X=[-5 21 2 0 -3.7];
» sign(X)
ans =-1 1 1 0 -1
```

2.3. Функції комплексного аргументу

Для роботи з комплексними величинами у MATLAB використовуються такі функції:

***angle*(*Z*)** – повертає масив аргументів комплексних чисел (у радіанах), обчислюваних кожного елемента масиву комплексних чисел *Z*. Кути перебувають у діапазоні [–π; π];

***imag*(*Z*) –** повертає масив уявних частин всіх елементів масиву *Z*;

***real*(*Z*)** – повертає масив дійсних частин всіх елементів комплексного масиву *Z*;

***conj*(*Z*) –** повертає масив комплексно-сполучених чисел щодо чисел в масиві *Z*. Якщо *Z* комплексне, то *conj*(*Z*) = *real*(*Z*) – *i* * *imag*(*Z*).

Робота з комплексними числами пояснюється такими прикладами:

```
» Z=3+i*2
Z =
3.0000 + 2.0000i
»theta = angle(Z)
theta = 0.5880
» R = abs(Z)
R = 3.6056
» Z =R.*exp(i*theta)
Z =3.0000 + 2.0000i
»Z=[1+i, 3+2i, 2+3i];
» imag(Z)
ans =1 2 3
»Z=[1+i, 3+2i, 2+3i];
» real(Z)
ans =
1 3 2
» conj(2+3i)
ans =
2.0000 - 3.0000i
```

2.4. Тригонометричні та гіперболічні функції

У системі MATLAB визначено звичайні тригонометричні та зворотні тригонометричні функції. Однак ці функції обчислюються для кожного елемента масиву *X*. Вхідний масив *X* може складатися з комплексних величин. Нагадуємо, що всі кути у функціях задані радіанами.

2.4.1. Тригонометричні функції

sin(*X*) – синус;

cos(*X*) – косинус;

tan(*X*) – тангес;

cot(*X*) – котангенс;

sec(*X*) – секанс;

csc(*X*) – косеканс.

Приклади:

```
» X=[1 2 3];
» cos(X)
ans = 0.5403 -0.4161 -0.9900
» Y = cot(2)
Y = -0.4577
» X=[2 4.678 5; 0.987 1 3];
» Y = csc(X)
Y =
1.0998 -1.0006 -1.0428
1.1985 1.1884 7.0862
» X=[pi/10 pi/3 pi/5];
» sec(X)
ans = 1.0515 2.0000 1.2361
» X=[pi/2 pi/4 pi/6 pi];
» sin(X)
ans = 1.0000 0.7071 0.5000 0.0000
» X=[0.08 0.06 1.09]
X = 0.0800 0.0600 1.0900
»tan(X)
ans = 0.802 0.0601 1.9171
```

2.4.2. Зворотні тригонометричні функції

asin(*X*) – арксинус;

acos(*X*) – арккосинус;

atan(*X*) – арктангенс;

acot(*X*) – арккотангенс;

asec(*X*) – арксеканс;

acsc(*X*) – арккосеканс.

Ще одна функція *atan2*(*Y, X*) – повертає масив *P* тієї ж мірності, що *X* і *Y*, який містить поелементно арктангенси дійсних частин *Y* і *X*. Уявні частини ігноруються. Елементи *P* перебувають у інтервалі [-π, π]. Специфічний квадрант визначено функціями *sign*(*Y*) та *sign*(*X*). Це відрізняє отриманий результат від *atan*(*Y*/*X*), який обмежений інтервалом [-π/2, π/2].

Приклади:

```
» acos([0.5 1 2])
ans = 1.0472  0  0+1.3170i
» Y=acot(0.1)
Y = 1.4711
» Y = acsc(3)
Y = 0.3398
» Y=asec(0.5)
Y = 0 + 1.3170i
» Y = asin (0.278)
Y = 0.2817
» Y=atan(1)
Y =0.7854
» atan2(1,2)
ans = 0.4636
```

2.4.3. Гіперболічні функції

sinh(*X*) – гіперболічний синус;

cosh(*X*) – гіперболічний косинус;

tanh(*X*) – гіперболічний тангенс;

coth(*X*) – гіперболічний котангенс;

sech(*X*) – гіперболічний секанс;

csch(*X*) – гіперболічний косеканс.

Приклади:

```
» cosh(X)
ans = 1.5431 3.7622 10.0677
» Y = coth(3.987)
1.0007
» X=[2 4.678 5; 0.987 1 3];
» Y = csch(X)
0.2757 0.0186 0.0135
0.8656 0.8509 0.0998
» X=[pi/2 pi/4 pi/6 pi];
```

```
» sech(X)
ans =
0.3985 0.7549 0.8770 0.0863
» X=[pi/8 pi/7 pi/5 pi/10];
» sinh(X)
ans =
0.4029 0.4640 0.6705 0.3194
» X=[pi/2 pi/4 pi/6 pi/10];
»tanh(X)
ans =
0.9172 0.6558 0.4805 0.3042
```

2.4.4. Зворотні гіперболічні функції

asinh(*X*) – гіперболічний арксинус;

acosh(*X*) – гіперболічний арккосинус;

atanh(*X*) – гіперболічний арктангенс;

acoth(*X*) – гіперболічний арккотангенс;

asech(*X*) – гіперболічний арксеканс;

acsch(*X*) – гіперболічний арккосеканс.

Приклади:

```
»Y = acosh (0.7)
Y = 0 + 0.7954i
»Y = acoth (0.1)
Y = 0.1003 + 1.5708i
» Y = acsch(1)
Y = 0.8814
» Y = asech(4)
Y = 0 + 1.3181i
» Y = asinh (2.456)
Y = 1.6308
»X=[0.84 0.16 1.39];
» atanh (X)
ans =
1.2212  0.1614  0.9065+1.5708i
```

2.5. Побудова таблиць значень функції однієї змінної

Відображення функції у вигляді таблиці зручно, якщо є досить невелика кількість значень функції. Розглянемо приклад: нехай потрібно вивести у командне вікно таблицю значень функції

$$y(x) = \frac{\sin^2 x}{1+\cos x} + e^{-x}\ln x$$

в точках 0,2; 0,3; 0,5; 0,8; 1,3; 1,7; 2,5.

Завдання вирішується у два етапи.

1. Створюється вектор-рядок x, що містить координати заданих точок.

2. Обчислюються значення функції *y*(*x*) від кожного елемента вектора *x* і записуються отримані значення вектор-рядок *y*.

Значення функції необхідно знайти для кожного з елементів векторного рядка *x*, тому операції у виразі функції повинні виконуватися поелементно.

```
» x = [0.2 0.3 0.5 0.8 1.3 1.7 2.5]
x =
    0.2000 0.3000 0.5000 0.8000 1.3000 1.7000 2.5000
» y = sin(x).^2./(1+cos(x))+exp(-x).*log(x)
y =
    -1.2978 -0.8473 -0.2980 0.2030 0.8040 1.2258 1.8764
```

Зверніть увагу, що при спробі використання операцій зведення в ступінь ^, поділу / та множення * (які не відносяться до поелементних) виводиться повідомлення про помилку вже при зведенні *sin*(*x*) у квадрат:

```
» y = sin(x)^2/(1+cos(x))+exp(-x)*log(x)
??? Error using ==> ^
Matrix must be square.
```

У MATLAB операції * і ^ застосовуються для перемноження матриць відповідних розмірів та зведення квадратної матриці у ступінь.

Часто потрібно вивести значення функції в точках відрізка, що віддаляються один від одного на рівну відстань (крок). Припустимо, необхідно вивести таблицю значень функції $y(x)$ на відрізку [1, 2] з кроком 0.2. Можна, звичайно, ввести вектор рядок значень аргументу x = [1, 1.2, 1.4, 1.6, 1.8, 2.0] з командного рядка і обчислити всі значення функції так, як описано вище. Однак, якщо крок буде не 0.2, а, наприклад, 0.01, то буде велика робота з введенням вектора x.

У MATLAB передбачено просте створення векторів, кожен елемент яких відрізняється від попереднього на величину, тобто на крок. Для введення таких векторів служить двокрапка (не плутайте з індексацією за допомогою двокрапки). Наступні два оператори призводять до формування однакових векторних рядків. Умовно можна записати

```
» x = [1, 1.2, 1.4, 1.6, 1.8, 2.0]
x =
    1.0000 1.2000 1.4000 1.6000 1.8000 2.0000
» x = [1:0.2:2]
x =
    1.0000 1.2000 1.4000 1.6000 1.8000 2.0000
```

де x = [початкове значення: крок: кінцеве значення].

Необов'язково піклуватися про те, щоб сума передостаннього значення кроку дорівнювала кінцевому значенню, наприклад, при виконанні наступного оператора присвоєння

```
» x = [1:0.2:1.9]
x =
    1.0000 1.2000 1.4000 1.6000 1.8000
```

вектор-рядок заповниться до елемента, який не перевищує задане нами кінцеве значення. Крок може бути і негативним:

```
» x = [1.9:-0.2:1]
x =
    1.9000 1.7000 1.5000 1.3000 1.1000
```

У разі негативного кроку для отримання непустого векторного рядка початкове значення має бути більшим за кінцевий.

Для заповнення вектор-стовпця елементами, що починаються з нуля і закінчуються 0.5 з кроком 0.1, слід заповнити вектор-рядок, а потім використовувати операцію транспонування:

```
» x = [0:0.1:0.5]'
x =
    0
    0.1000
    0.2000
    0.3000
    0.4000
    0.5000
```

Зверніть увагу, що елементи вектору, що заповнюється за допомогою двокрапки, можуть бути лише дійсними, тому для транспонування можна використовувати апостроф замість крапки з апострофом.

Крок, що дорівнює одиниці, допускається не вказувати при автоматичному заповненні:

```
» x = [1:5]
x =
    1 2 3 4 5
```

Приклад. Нехай потрібно вивести таблицю значень функції

$$y(x) = e^{-x}\sin(10x)$$

на відрізку [0, 1] з кроком 0,05.

Для виконання цього завдання необхідно зробити такі дії:

1. Сформувати вектор-рядок *x* за допомогою двокрапки.

2. Обчислити значення *y*(*x*) від елементів *x*.

3. Записати результат у вектор-рядок *y*.

5. Вивести *x* та *y*.

```
» x = [0:0.05:1];
» y = exp(-x).*sin(10*x);
» x
x =
    Columns 1 through 7
    0 0.0500 0.1000 0.1500 0.2000 0.2500 0.3000
    Columns 8 through 14
    0.3500 0.4000 0.4500 0.5000 0.5500 0.6000 0.6500
    Columns 15 through 21
    0.7000 0.7500 0.8000 0.8500 0.9000 0.9500 1.0000
» y
y = Columns 1 through 7
    0 0.4560 0.7614 0.8586 0.7445 0.4661 0.1045
    Columns 8 through 14
    -0.2472 -0.5073 -0.6233 -0.5816 -0.4071 -0.1533 0.1123
    Columns 15 through 21
    0.3262 0.4431 0.4445 0.3413 0.1676 -0.0291 -0.2001
```

Вектор-рядки *x* та *y* складаються з двадцяти одного елемента і не вміщуються на екрані в один рядок, тому виводяться частинами. Оскільки *x* і *y* зберігаються у двомірних масивах розмірністю один на двадцять один, то виводяться по стовпчикам, кожен із яких складається з одного елемента. Спочатку виводяться стовпці з першого по сьоме (*columns* 1 *through* 7), потім – з восьмого по чотирнадцятий (*columns* 8 *through* 14) і, нарешті, – з п'ятнадцятого по двадцять перший (*columns* 15 *through* 21). Наочнішим і зручнішим є графічне уявлення функції.

2.6. Функції порозрядної логічної обробки даних

Ряд функцій системи MATLAB визначено для порозрядної логічної обробки даних.

***bitand*(*A*, *B*) –** повертає порозрядне логічне І двох невід'ємних цілих аргументів *A* і *B*. Приклад:

```
» f = bitand (7,14)
f = 6
```

***bitcmp*(*A*, *n*)** – повертає порозрядне доповнення аргументу *A* як *n*-бітове ціле число у форматі чисел з плаваючою комою (*floating-point integer* або скорочено *flint*). Приклад:

```
» g=bitcmp(6,4)
g = 9
```

***bitor*(*A*, *B*)** – повертає порозрядне логічне АБО двох невід'ємних цілих аргументів *A* і *B*. Приклад:

```
» v = bitor (12,21)
v = 29
```

bitmax – повертає максимальне ціле число без знака з плаваючою комою для вашого комп'ютера. Це значення визначається, коли всі біти встановлені. Приклад:

```
» bitmax
ans =
9.0072e+015
```

***bitset*(*A*, *bit*)** – встановлює біт у позиції *bit* аргументу *A* в 1. *A* повинен бути невід'ємним цілим, а параметр *bit* – номер між 1 і кількістю біт у цілому числі *A* з плаваючою комою.

***bitset*(*A, bit, v*)** – встановлює біт у позиції *bit* рівним значенню *V*, яке має бути або 0 або 1. Приклад:

```
» d = bitset (12,2,1)
d =
14
```

***bitshift*(*A, n*)** – повертає значення аргументу *A*, зрушене на *n* біт. Якщо $n > 0$, це аналогічно до множення на $2n$ (лівий зсув). Якщо $n < 0$, це аналогічно поділу на $2n$ (правий зсув). Приклад:

```
» f = bitshift (4,3)
f =
32
```

***bitget*(*A, bit*)** – повертає значення біта позиції *bit* операнда *A*. *A* повинен бути невід'ємним цілим і *bit* – номер між 1 і числом біт в цілому числі формату з плаваючою комою, представленому *A*. Приклад:

```
» disp(dec2bin(23))
10111
»C = bitget(23,5:-1:1)
C =
1 0 1 1 1
```

***bitxor*(*A, B*)** - повертає порозрядне логічне виключне АБО двох аргументів *A* і *B*. Приклад:

```
» x = bitxor (12, 31)
x =
19
```

Аргументи цих функцій повинні мати цілі значення. Для гарантії цього в сумнівних випадках корисно використовувати функції *ceil*, *fix*, *floor* та *round*.

2.7. Функції обробки множин

У MATLAB визначено ряд функцій для обробки множин:

***intersect*(*a, b*) –** повертає перетин множин для двох векторів *a* і *b*, тобто загальні елементи векторів *a* та *b*. Результативний вектор відсортований за зростанням. Якщо вхідні масиви – не вектори, вони розцінюються як вектори-стовпці *a* = *a*(:) чи *b* = *b*(:).

***intersect*(*a, b, 'rows'*)** – повертає рядки, загальні для *a* та *b*, коли *a* та *b* - матриці з однаковим числом стовпців.

[*c, ia, ib*] = *intersect*(*a, b*) – також повертає вектор-стовпець індексів *ia* та *ib*, але так, що *c* = *a*(*ia*) та *c* = *b*(*ib*) (або *c* = *a*(*ia*,:) та *c* = *b*(*ib*,:)). Приклад:

```
»A = [17 26]; B = [7 2 3 4 6 1];
»[c, ia, ib] = intersect (A, B)
C = 1 2 6 7
ia =1 3 4 2
ib = 6 2 5 1
```

***ismember*(*a, S*)** – повертає вектор тієї ж довжини, що і вихідний, що містить логічну одиницю, де елемент вектора належить множині *S*, і логічний нуль для всіх інших.

***ismember*(*A, S, 'rows'*)** – повертає вектор, що містить логічну одиницю, де рядки матриці є також рядками матриці *S*, і 0 у всіх інших випадках. *A* і *S* – матриці з однаковим числом стовпців. Приклад:

```
» set = [0 1 3 5 7 9 11 15 17 19];
» a = [1 2 3 4 5 6 7 8];
» k = ismember(a,set)
k =
1 0 1 0 1 0 1 0
```

***setdiff*(*a, b*)** – повертає різницю множин, тобто ті елементи вектора, які не містяться у векторі *b* (різниця множин). Результуючий вектор сортується за

зростанням.

***setduff*(*a, b,* '*rows*')** – повертає ті рядки з матриці *a*, які містяться в матриці *b*. *a* та *b* – матриці з однаковим числом стовпців.

[*c, i*] = *setdiff*(...) – повертає вектор індексів *index* так, що *c* = *a*(*i*) або *c* = *a*(*i*,:).

Якщо вхідний масив *a* не вектор, то він розцінюється як вектор-стовпець *a*(:). Приклад:

```
» a = [2 3 5 7 8 9 10 13 20];
» b = [1 4 5 6 8 9 4]
» c = setdiff(a, b)
c =
2 3 7 10 13 20
```

***setxor*(*a, b*)** – повертає результат логічного виняткового АБО для векторів *a* та *b*. Результуючий вектор відсортовано.

***setxor*(*a, b,* '*rows*')** – повертає рядки, які є перетинами матриць *a* і *b*. Де *a* та *b* - матриці з однаковим числом стовпців. Якщо вхідний масив *a* не вектор, то він розцінюється як вектор-стовпець *a*(:). Приклад:

```
» a = [-1 0 1 Inf -Inf NaN];
» b = [-2 pi 0 Inf];
» c = setxor(a, b)
c =
-Inf -2.0000 -1.0000 1.0000 3.1416 NaN
```

***union*(*a, b*)** – повертає вектор об'єднаних значень *a* і *b* без повторюваних елементів. Результуючий вектор сортується у порядку зростання.

***union*(*a, b,* '*rows*')** – повертає об'єднані рядки з *a* і *b*, що не містять повторень (*a* і *b* матриці з однаковим числом стовпців).

[*c, ia, ib*] = union(...) – повертає *ia* та *ib* - вектори індексів, такі, що *c* = *a*(*ia*) та *c* = *b*(*ib*), або для об'єднаних рядків *c* = *a*(*ia*, :) та *c* = *b*(*ib*,:). Не векторний масив *a* розцінюється як вектор-стовпець *a*(:). Приклад:

```
» a = [2,4,-4,9,0]; b = [2,5,4];
» [c, ia, ib] = union(a, b)
c = -4  0  2  4  5  9
ia = 3  5  4
ib = 1  3  2
```

***unique*(*a*)** – повертає такі ж значення елементів, як і *a*, але не містять повторень. Результуючий вектор сортується в порядку зростання. Не векторний масив розцінюється як вектор-стовпець *a* = *a*(:).

***unique*(*a*, '*rows*')** – повертає унікальні рядки *a*.

[*b*, *i*, *j*] = *unique*(...) – додатково повертає *i* та *j* – вектори індексів, такі, що *b* = *a*(*i*) та *a* = *b*(*j*) (або *b* = *a*(*i*,:) та a = *b*(*j*,:)). Приклади:

```
» b=[-2,3,5,4,1,-6,2,2,7]
b =
-2  3  5  4  1  -6  2  2  7
»[c,i,j]=unique(b)
c =
-6 -2  1  2  3  4  5  7
i =
6  1  5  8  2  4  3  9
j =
2  5  7  6  3  1  4  4  8
» a=[-2,3,5; 4,1,-6; 2,2,7;-2,3,5]
a =
-2   3   5
4   1   -6
2   2   7
2   3   5
» c=unique(a,’rows')
c =
-2   3   5
2   2   7
4   1   -6
```

Контрольні запитання:

1. Перелічите алгебраїчні та арифметичні функції, що існують у системі комп'ютерної математики MATLAB та наведіть приклади їх застосування?
2. Перелічите функції числової апроксимації, округлення та знаку, що існують у системі комп'ютерної математики MATLAB та наведіть приклади їх застосування?
3. Перелічите функції комплексного аргументу, що існують у системі комп'ютерної математики MATLAB та наведіть приклади їх застосування?
4. Перелічите тригонометричні функції, що існують у системі комп'ютерної математики MATLAB та наведіть приклади їх застосування?
5. Перелічите зворотні тригонометричні функції, що існують у системі комп'ютерної математики MATLAB та наведіть приклади їх застосування?
6. Перелічите гіперболічні функції, що існують у системі комп'ютерної математики MATLAB та наведіть приклади їх застосування?
7. Перелічите зворотні гіперболічні функції, що існують у системі комп'ютерної математики MATLAB та наведіть приклади їх застосування?
8. У чому особливості побудови таблиць значень функції однієї змінної та наведіть приклад такої побудови?
9. Перелічите функції порозрядної логічної обробки даних, що існують у системі комп'ютерної математики MATLAB та наведіть приклади їх застосування?
10. Перелічите функції обробки множин, що існують у системі комп'ютерної математики MATLAB та наведіть приклади їх застосування?

РОЗДІЛ 3

МАТРИЧНІ ФУНКЦІЇ СИСТЕМИ MATLAB

Система комп'ютерної математики MATLAB орієнтована виконання матричних обчислень і містить найповніший набір відповідних операторів і функцій. Вони дозволяють професійно розробляти базовані на матричних методах системи моделювання та проектування різноманітних пристроїв та систем.

3.1. Створення матриць у системі MATLAB

MATLAB дозволяє створювати матриці поєднанням (конкатенацією) вже існуючих матриць. Для цього є функція *cat*:

***cat*(*dim*, *A*, *B*)** – об'єднує масиви *A* і *B* відповідно до специфікації розмірності *dim* і повертає об'єднаний масив.

***cat*(*dim*, *A*1, *A*2, *A*3, *A*4, ...)** – поєднує всі вхідні масиви (*A*1, *A*2, *A*3, *A*4 і т.д.) відповідно до специфікації розміру *dim* і повертає об'єднаний масив.

***cat*(2, *A*, *B*)** – це те саме що і [*A*, *B*], а cat(1, *A*, *B*) це те ж саме що і [*A*; *B*].

Приклад:

```
» A =[2,4;3,5];B=[8,7;9,0];C=cat(1,A,B)
C =
2 4
3 5
8 7
9 0
```

Для створення матриць, які складаються з інших матриць, служать такі функції:

***repmat*(*A*, *m*, *n*) –** формує матрицю, що складається з $m \times n$ копій матриці *A*.

***repmat*(*A, n*) –** формує матрицю, що складається з *n*×*n* копій матриці *A*.

***repmat*(*A*, [*m n*]) –** дає результат, подібний до *repmat*(*A*, *m*, *n*).

***repmat*(*A*,[*m n p ...*])** - формує багатовимірний масив (*m n p*...), що складається з копій матриці *A*.

***repmat*(*a, m, n*) –** формує *m*×*n* матрицю зі значеннями скаляра *a*. Це обчислюється набагато швидше, ніж *a*×*ones*(*m*, *n*).

Приклад:

```
F =
3       2
43     32
» repmat(F,2,3)
ans =
3    2    3    2    3    2
43   32   43   32   43   32
3    2    3    2    3    2
43   32   43   32   43   32
```

***reshape*(*A, m, n*) –** повертає матрицю розміру *m*×*n*, сформовану *A* шляхом послідовної вибірки по стовпцях. Якщо *A* не має *m*×*n* елементів, то видається повідомлення про помилку.

***reshape*(*A, m, n, p, ...*)** або ***B* = *reshape*(*A*, [*m n p...*])** – повертає масив *N* розмірностями з такими ж елементами як *A*, але має розмір *m*×*n*×*p*.... Добуток *m*×*n*×*p*... має дорівнювати значення *prod*(*size*(*x*)).

***reshape*(*A, siz*)** – повертає масив *N* розмірностями з такими ж елементами як *A*, але перебудований до *siz*, вектору, що визначає розмір масиву.

Приклад:

```
» F=[3,2,7,4;4,3,3,2;2,2,5,5]
F =
3    2    7    4
4    3    3    2
2    2    5    5
»reshape(F,2,6)
```

```
ans =
3    2    3    7    5    2
4    2    2    3    4    5
```

Для створення матриць із заданою діагоналлю *diag*:

X = diag(v, k) – повертає квадратну матрицю *X* порядку *n* + *abs*(*k*) з елементами *v* на *k*-ій діагоналі, при *k* = 0 – це головна діагональ, при *k* > 0 – це верхня діагональ, при *k* < 0 – це нижня діагональ. Тут *v* – вектор, що складається з *n* компонентів, інші елементи – нулі.

X = diag(v) поміщає вектор v на головну діагональ (те, що й у попередньому випадку при *k* = 0).

v = diag(X, k) – повертає вектор-стовпець, що складається з елементів *k*-ої діагоналі матриці *X*.

v = diag(X) – повертає головну діагональ матриці *X* (те, що й у попередньому випадку при *k* = 0).

Приклади:

```
» v=[2,3];
» X=diag(v,2)
X =
0    0    2    0
0    0    0    3
0    0    0    0
0    0    0    0
» X=[2,5,45,6;3,5,4,9;7,9,4,8;5,66,45,2];
» V=diag(X,0)
V =
2
5
4
2
```

Для створення “магічної” матриці служить функція: ***magic(n)*** – повертає матрицю розміру *n*×*n*, що складається з цілих чисел від 1 до n^2 у якій суми елементів за рядками, стовпчиками і головним діагоналям дорівнюють тому

самому числу. Порядок матриці має бути більшим або дорівнює 3. Наприклад:

```
» M=magic(4)
M =
16   2    3    13
5    11   10   8
9    7    6    12
4    14   15   1
```

3.2. Матричні функції перестановки

Багато матричних обчислень засновані на перестановках елементів матриць. Наступні функції реалізують основні види перестановок:

B = fliplr(A) – переставляє стовпці матриці *A* щодо вертикальної осі. Приклад:

```
»F=[1,2,3;5,45,3]
F =
1    2    3
5    45   3
» fliplr(F)
ans =
3    2    1
3    45   5
```

B = flipud(A) **–** переставляє рядки матриці відносно горизонтальної осі. Приклад:

```
» F=[3,2,12;6,3,2]
F =  3    2    12
6    3    2
» flipud(F)
ans =
6    3    2
3    2    12
```

***perms*(*v*) –** повертає матрицю *P*, рядки якої є всі можливі перестановки елементів вектора *V*. Матриця містить *n*! рядків та *n* стовпців. Приклад:

```
» V=[1 4 6]
v = 1 4 6
» P=perms(v)
P =
6    4    1
4    6    1
6    1    4
1    6    4
4    1    6
1    4    6
```

3.2.1. Функція, що забезпечує поворот матриці

***rot*90(*A*) –** здійснює поворот матриці на 90 градусів проти годинникової стрілки;

***rot*90(*A, k*)** – здійснює поворот матриці на величину 90×*k* проти годинникової стрілки, де *k* – ціле число. Приклад:

```
»M=[3,2,7;3,3,2;1,1,1]
M =
3    2    7
3    3    2
1    1    1
» rot90(M)
ans =
7    2    1
2    3    1
3    3    1
```

3.2.2. Функції обчислення добутків та сум для елементів матриць

Операції добутку для елементів матриць реалізовані такими функціями:

***prod*(*A*)** – повертає добуток елементів масиву, якщо *A* – вектор, або повертає вектор рядок, що містить добутки елементів кожного стовпця, якщо *A* – матриця.

***prod*(*A, dim*) –** повертає добуток елементів масиву стовпчиками або рядками

матриці залежно від значення скаляра *dim*. Приклад:

```
» A=[1 2 3 4; 2 4 5 7; 6 8 3 4]
A =
1    2    3    4
2    4    5    7
6    8    3    4
» B=prod(A)
B =
12  64  45  112
```

***cumprod*(*A*) –** повертає добуток із накопиченням. Якщо *A* – вектор, *cumprod*(*A*) повертає вектор, що містить добуток з накопиченням елементів вектора *A*. Якщо *A* – матриця, то *cumprod*(*A*) повертає матрицю того ж розміру, як і *A*, що містить добуток з накопиченням кожного стовпця матриці *A*.

***cumprod*(*A, dim*) –** повертає добуток із накопиченням елементів рядками або стовпцями залежно від значення скаляра *dim*. Наприклад, *cumprod*(*A*,1) дає приріст першому індексу (рядки), таким чином виконуючи множення по стовпцях матриці *A*. Приклад:

```
» A=[1 2 3; 4 5 6; 7 8 9]
A =
1    2    3
4    5    6
7    8    9
» B = cumprod(A)
B =
1    2    3
4    10   18
28   80   162
B = cumprod(A,1)
B =
1    2    3
4    10   18
28   80   162
```

***cross*(*U, V*) –** повертає векторний добуток векторів *U* та *V* у тривимірному просторі. Тобто *W* = *U* × *V*, де *U* та *V* – зазвичай вектори з трьома елементами.

***cross*(*U, V, dim*)** – повертає векторний добуток багатовимірних масивів *U* і *V* за розмірністю, визначеною скаляром *dim*. *U* і *V* повинні мати той самий розмір - *size*(*U*, *dim*) та *size*(*V*, *dim*) і дорівнювати 3. Приклад:

```
» a = [6 5 3];
» b = [1 7 6];
» c = cross(a,b)
C = 9 -33 37
```

Визначено також наступні функції підсумовування елементів масивів:

***sum*(*A*) –** повертає суму елементів масиву, якщо *A* – вектор або повертає вектор-рядок, що містить суму елементів кожного стовпця, якщо *A* – матриця.

***sum*(*A, dim*) –** повертає суму елементів масиву по стовпцям або рядкам матриці в залежності від значення скаляра *dim*. Приклад:

```
» A=magic(4)
A =
16   2    3    13
5    11   10   8
9    7    6    12
4    14   15   1
» B = sum(A)
B =
34   34   34   34
```

***B* = *cumsum*(*A*) –** виконує підсумовування з накопиченням. Якщо *A* – вектор, *cumsum*(*A*) повертає вектор, що містить підсумовування з накопиченням елементів вектора *A*. Якщо *A* – матриця, *cumsum*(*A*) повертає матрицю того ж розміру, що містить підсумовування з накопиченням кожного стовпця матриці *A*.

***B* = *cumsum*(*A, dim*)** - виконує підсумовування з накопиченням елементів за розмірністю, точно визначеною скаляром *dim*. Наприклад, *cumsum*(*A*,1) виконує підсумовування по стовпцях. Приклад:

```
» A=magic(4)
A =
16   2    3    13
5    11   10   8
9    7    6    12
4    14   15   1
» B = cumsum(A)
B =
16   2    3    13
21   13   13   21
30   20   19   33
34   34   34   34
```

3.2.3. Функції виділення трикутних частин матриць

При виконанні ряду матричних обчислень виникає потреба у виділенні трикутної частини матриць. Наступні функції забезпечують таке виділення:

***tril*(*X*) –** повертає нижню трикутну частину матриці *X*;

***tril*(*X*, *k*) –** повертає нижню трикутну частину матриці *X* починаючи з *k*-ої діагоналі. При $k = 0$ – це головна діагональ, при $k > 0$ – це верхня діагональ, при $k < 0$ – це нижня діагональ. Приклад:

```
» M=[3,1,4;8,3,2;8,1,1]
M =
3    1    4
8    3    2
8    1    1
»tril(M)
ans =
3    0    0
8    3    0
8    1    1
```

***triu*(*X*) –** повертає верхню трикутну частину матриці *X*;

***triu*(*X*, *k*) –** повертає верхню трикутну частину матриці *X* починаючи з *k*-ої діагоналі. При $k = 0$ це головна діагональ, при $k > 0$ – це верхня діагональ, при $k < 0$ це нижня діагональ. Приклад:

```
» M=[3,1,4;8,3,2;8,1,1]
M =
3    1    4
8    3    2
8    1    1
»triu(M)
ans =
3    1    4
0    3    2
0    0    1
```

3.2.4. Обчислення спеціальних матриць

У практиці матричних обчислень широко використовують матриці спеціального виду. Нижче наведено функції, що використовуються при роботі з такими матрицями.

***compan*(*u*) –** повертає супроводжуючу матрицю, перший рядок якої – *u*(2:*n*) / *u*(1), де *u* – вектор поліноміальних коефіцієнтів. Власні значення *compan*(*u*) – коріння багаточлена. Приклад для багаточлена $x^3 + x^2 - 6x - 8$ вектор поліноміальних коефіцієнтів *r*:

```
» r=[1,1,-6,-8]
r=
1    1    -6   -8
» A=compan(r) % супроводжуюча матриця
A =
-1   6    8
1    0    0
0    1    0
» eig(compan(r)) % корні многочлена
ans =
-2.0000
2.5616
-1.5616
```

[*A, B, C,* ...] = *gallery*('*tmfun*', *P*1, *P*2, ...) – повертає тестові матриці, визначені рядком *tmfun*, де *tmfun* – це ім'я сімейства матриць, вибране зі списку. *P*1, *P*2, … – вхідні параметри, потрібні для конкретного сімейства матриць. Число

використовуваних параметрів *P*1, *P*2, ... змінюється від матриці до матриці, галерея зберігає більше 50 різних тестових матричних функцій, корисних для тестуючих алгоритмів та інших цілей. Приклад:

```
» A=gallery('dramadah',5,2)
A =
1    1    0    1    0
0    1    1    0    1
0    0    1    1    0
0    0    0    1    1
0    0    0    0    1
```

Існує ціла низка функцій для отримання матриць Адамара *hadamard*(*n*), Ганкеля *hankel*(*n*), Гільберта *hilb*(*n*), Паскаля *pascal*(*n*), Гепліца та Уїлкінсона *wilkinson*(*n*). Це досить рідкісні матриці, так що обмежимося поданням функції Адамра:

***H* = *hadamard*(*n*)** – повертає матрицю Адамара порядку *n*. Матриці Адамара застосовують у різних галузях, що включають комбінаторику, чисельний аналіз, обробку сигналів. Приклад:

```
» H = hadamard(4)
H =
1    1    1    1
1    -1   1    -1
1    1    -1   -1
1    -1   -1   1
```

3.3. Матричні функції для вирішення завдань лінійної алгебри

Розглянемо найчастіше використовувані функції лінійної алгебри. Їх набір у MATLAB досить ретельний і дозволяє вирішувати найсерйозніші завдання цього важливого розділу математики.

Число обумовленості матриці визначає чутливість розв'язання системи лінійних рівнянь до похибок даних. Наступні функції дозволяють визначити числа обумовленості матриць.

***cond*(*X*) –** повертає число обумовленості, засноване на другій нормі, відношення найбільшого сингулярного значення *X* до найменшого. Значення *cond*(*X*) і *cond*(*X*, *p*) близьке до 1 вказує добре обумовлену матрицю.

***C* = *cond*(*X*, *p*)** - повертає число обумовленості матриці, засноване на *p*-нормі: *norm*(*X*, *p*) × *norm*(*inv*(*X*), *p*), де *p* = 1 – число обумовленості матриці, засноване на першій нормі, *p* = 2 – число обумовленості, засноване на 2-нормі, *p* = "*fro*" – на нормі Фробеніуса (Frobenius) та *p* = *inf* на нескінченній нормі. Приклад:

```
» d=cond(hilb(4))
d = 1.5514e+004
```

***condeig*(*A*) –** повертає вектор чисел обумовленості для значень *A*. Ці числа обумовленості – зворотні величини косинусів кутів між лівими і правими власними векторами.

[*V*, *D*, *s*] = *condeig*(*A*) – еквівалентно [*V*, *D*] = *eig*(*A*); *s* = *condeig*(*A*).

Великі числа обумовленості мають на увазі, що *A* близька до матриці з кратними власними значеннями. Приклад:

```
» d=condeig(rand(4))
d =
1.0766
1.2298
1.5862
1.7540
```

***rcond*(*A*) –** оцінює обернену величину обумовленості матриці *A* першої норми. Якщо *A* добре обумовлена, то *rcond*(*A*) близько 1.00, якщо погано зумовлена близько 0.00. Порівняно з *cond*, *rcond* ефективніший, але менш достовірний метод оцінки обумовленості матриці. Приклад:

```
» s=rcond(hilb(4))
s = 4.6461 e-005
```

Для знаходження визначника і рангу матриць в MATLAB є наступні функції:

***det*(*X*) –** повертає визначник квадратної матриці *X*. Якщо *X* містить лише цілі елементи, то результат також ціле число. Використання *det*(*X*) = 0 як тест на виродженість матриці дійсний лише матриці малого порядку з цілими елементами. Приклад:

```
» A=[2,3,6;1,8,4;3,6,7]
A =
2   3   6
1   8   4
3   6   7
» det(A)
ans = -29
```

***rank*(*A*) –** повертає ранг матриці як число сингулярних значень, які є більшими, ніж заданий за умовчанням допуск;

***rank*(*A, tol*) –** повертає ранг матриці як число сингулярних значень, які є більшими, ніж допуск, заданий значенням змінної *tol*. Приклад:

```
» rank(hilb(11))
ans = 10
```

Для обчислень норми матриць призначені такі функції:

***norm*(*A*) –** повертає найбільше сингулярне значення *A*, *max*(*svd*(*A*));

***norm*(*A, p*) –** повертає різні види норм залежно від p: (*p*=1, 2, *inf*, "*fro*"). Приклад:

```
» A=[2,3,1;1,9,4;2,6,7]
A =
2   3   1
```

```
1    9    4
2    6    7
» norm(A,1)
ans =
18
```

Обчислення ортонормованого базису матриці забезпечують такі функції:

***orth*(*A*) –** повертає ортонормований базис матриці *A*. Стовпці *B* визначають той ж простір, що і стовпці матриці *A*, але стовпці *B* ортогональні, тобто *B*'×*B* = *eye*(*rank*(*A*)). Кількість стовпців матриці визначає ранг матриці *A*. Приклад:

```
» A=[2 4 6;9 8 2;12 23 43]
A =
2    4    6
9    8    2
12   23   43
» B=orth(A)
B =
0.1453   -0.0414  -0.9885
0.1522   -0.9863  0.0637
0.9776   0.1597   0.1371
```

***null*(*A*)** – повертає ортонормований базис для нульового (порожнього) простору *A*.

```
» null(hilb(11))
ans =
0.0000
-0.0000
0.0009
-0.0099
0.0593
-0.2101
0.4606
-0.6318
0.5276
-0.2453
0.0487
```

У лінійній алгебрі часто використовується приведення матриць до тієї чи іншої трикутної форми. Воно реалізується такими функціями:

***rref*(*A*) -** здійснює приведення матриці до трикутної форми, використовуючи метод виключення Гауса з частковим вибором провідного елемента. За замовчуванням приймається значення порога допустимості для незначного елемента стовпця рівним (*max*(*size*(*A*))×*eps*×*norm*(*A*,*inf*)).

[*R, jb*] = *rref*(*A*) – також повертає вектор *jb*, отже *r* = *length*(*jb*) може бути оцінкою рангу матриці *A*, *x*(*jb*) – пов'язані змінні в системі лінійних рівнянь виду *Ax* = *b*, *A*(:,*jb*) – базис матриці *A*, *R*(1:*r*, *jb*) – одинична матриця розміру *r* × *r*.

[*R, jb*] = *rref*(*A, tol*) – здійснює приведення матриці до трикутної форми, використовуючи метод виключення Гауса з частковим вибором провідного елемента для заданого значення порога допустимості *tol*.

***rrefmovie*(*A*) –** показує покрокове виконання процедури приведення матриці до трикутної. Приклади:

```
» s=magic(3)
s =
8    1    6
3    5    7
4    9    2
» rref(s)
ans =
1    0    0
0    1    0
0    0    1
```

Кут між двома підпросторами обчислює функція:

***theta* = *subspace*(*A, B*)** – повертає кут між двома підпросторами, точно визначеними між стовпцями матриць *A* і *B*. Якщо *A* і *B* – вектори-стовпці одиничної довжини, то кут обчислюється за формулою *acos*(*A*'×*B*). Якщо деякий фізичний експеримент описується масивом *A*, а друга реалізація цього експерименту – масивом *B*, *subspace*(*A, B*) вимірює кількість нової інформації,

отриманої з другого експерименту. Приклад:

```
» H = hadamard(20); A = H(:,2:4); B = H(:,5:8);
» subspace(A,B)
ans =
1.570
```

Слід матриці обчислюється функцією:

***trace*(*A*)** – повертає слід матриці (сума елементів діагоналі). Приклад:

```
» a=[2,3,4;5,6,7;8,9,1]
a =
2   3   4
5   6   7
8   9   1
»trace(a)
ans = 9
```

3.4. Набір матричних функцій

Набір матричних функцій у MATLAB:

***expm*(*X*)** – повертає експоненту від матриці *X*. Комплексний результат виходить якщо *X* має непозитивні власні значення.

***logm*(*X*)** – повертає логарифм матриці: функцію, обернену *expm*(*X*). Результат виходить комплексним, якщо *X* має негативні значення.

***sqrtm*(*X*)** – повертає квадратний корінь *X*. Результат виходить комплексним, якщо X має негативні власні значення.

***funm*(*X*,'*function*')** – дозволяє обчислити будь-яку функцію від матриці, якщо має ім'я, складене з латинських букв. Матриця *X* має бути квадратною. Команди *funm*(*X*,'*sqrt*') та *funm*(*X*,'*log*') еквівалентні командам *sqrtm*(*X*) та *logm*(*X*). Команди *funm*(*X*,'*exp*') і *expm*(*X*) обчислюють однакову функцію, але використовуючи різні алгоритми. Приклад:

```
» S=[1,0,3;1,3,1;4,0,0]
S=
1    0    3
1    3    1
4    0    0
» a=expm(S)
a =
31.2203  0        23.3779
38.9659  20.0855  30.0593
31.1705  0        23.4277
```

3.5. Функції для розріджених матриць

Матриці, що містять велику кількість елементів з нульовими значеннями, прийнято називати розрідженими матрицями. Вони широко використовуються при вирішенні прикладних завдань. Наприклад, моделювання електронних та електротехнічних лінійних ланцюгів часто призводить до появи в матричному описі топології схем сильно розріджених матриць. Для таких матриць створено ряд функцій, що забезпечують ефективну роботу з ними та усувають тривіальні операції з елементами матриць.

Розглянемо елементарні розріджені матриці та функції системи MATLAB, що відносяться до них.

Функція *spdiags* розширює можливості вбудованої функції *diag*. Можливі чотири операції, що розрізняються кількістю вхідних аргументів.

[*B, d*] = *spdiags*(*A*) – витягує всі ненульові діагоналі з матриці розміру $m \times n$. B - матриця розміру $min(m, n)$, стовпці якої є ненульовими діагоналями *A*. *d* - вектор довжини *p*, цілочисельні елементи якого точно визначають номери діагоналей матриці *A*.

***B* = *spdiags*(*A, d*) –** витягує діагоналі, визначені вектором *d*.

***A* = *spdiags*(*B, d, A*) –** замінює діагоналі матриці *A*, визначені вектором *d* зі стовпцями матриці.

***A* = *spdiags*(*B, d, m, n*) –** створює розріджену матрицю розміру *m*×*n*, розміщуючи відповідні стовпці матриці уздовж діагоналей, що визначаються вектором *d*. Приклад:

```
» A=
[1 3 4  6  8 0 0;
 7 8 0  7  0 0 5;
 0 0 0  0  0 9 8;
 7 6 54 32 0 9 6];
» d=[1 3 2 2]
» B = spdiags(A,d)
B=
3    6    4    4
0    0    7    7
0    9    0    0
0    6    9    9
```

***S* = *speye*(*m, n*)** – формує розріджену матрицю розміру *m*×*n* з одиницями на головній діагоналі та нульовими недіагональними елементами.

***S* = *speye*(*n*)** – рівносильна *speye*(*n, n*). Приклад:

```
» S = speye(4)
S=
1    0    0    0
0    1    0    0
0    0    1    0
0    0    0    1
```

Матриця *R* = *sprand*(*S*) має таку ж структуру, як і розріджена матриця *S*, та її елементи розподілені за рівномірним законом.

***R* = *sprand*(*m, n, density*)** - повертає випадкову розріджену матрицю розміру *m*×*n*, яка має приблизно density *m*×*n* рівномірно розподілених ненульових елементів (0 _ *density* _ 1).

***R* = *sprandn*(*S*)** – повертає матрицю зі структурою розрідженої матриці *S*, але з елементами, розподіленими за нормальним законом із середнім, що дорівнює 0 і

дисперсією, що дорівнює 1.

R = sprandn(m, n, density) – повертає випадкову розріджену матрицю розміру $m \times n$, що має приблизно *density* $m \times n$ нормально розподілених ненульових елементів (0 _ *density* _ 1).

sprandsym(S) – повертає випадкову симетричну матрицю, нижні піддіагоналі і головна діагональ якої має ту ж структуру як і матриця *S*. Її елементи розподілені за нормальним законом із середнім, що дорівнює 0 і дисперсією, що дорівнює 1.

sprandsym(n, density) - повертає симетричну випадкову розріджену матрицю розміру $n \times n$, яка має приблизно *density* $n \times n$ ненульових елементів; кожен елемент сформований як суми нормально розподілених випадкових чисел (0 _ *density* _ 1).

3.6. Алгоритми упорядкування матриць

Упорядкування – ще одна характерна для розріджених матриць операція. Її алгоритм реалізується кількома наведеними нижче функціями.

p = colmmd(S) – повертає вектор упорядкованості стовпців розрідженої матриці *S*. Для несиметричної матриці *S* вектор упорядкованості стовпців *p* такий, що *S*(:, *p*) матиме більш розріджене розкладання, ніж *S*. Таке впорядкування автоматично застосовується при виконанні операцій \ та / при вирішенні систем лінійних рівнянь із розрідженими матрицями. Можна використати команду *spparms*, щоб змінити деякі опції та параметри, пов'язані з евристикою в алгоритмі.

j = colperm(S) – повертає вектор перестановок *j*, такий, що стовпці матриці *S*(:, *j*) будуть упорядковані за зростання числа ненульових елементів. Якщо *S* – симетрична матриця, то *j* = *colperm*(*S*) повертає вектор перестановок *j*, такий як стовпці та рядки *S*(*j, j*) упорядковані за зростанням ненульових елементів. Якщо матриця *S* позитивна та визначена, тоді корисно застосовувати цю функцію перед виконанням розкладання Холецького. Приклад:

```
» S=sparse([2,3,1,4,2],[1,3,2,3,2],[4,3,5,6,7],4,5);
»full(S)
ans =
0    5    0    0    0
4    7    0    0    0
0    0    3    0    0
0    0    6    0    0
»t=colperm(S)
t = 4 5 1 2 3
» full(S(:,t))
ans =
0    0    0    5    0
0    0    4    7    0
0    0    0    0    3
0    0    0    0    6
```

***p* = *dmperm*(*A*) –** повертає вектор максимальної відповідності *p* такий, що коли вихідна матриця має повний стовпцевий ранг, то *A*(*p*, :) – квадратна з ненульовою діагоналлю. Матриця *A*(*p*, :) називається декомпозицією Далмейджа Мендельсона або *DM*-декомпозицією.

Якщо *A* – матриця, лінійна система *Ax* = *b* може бути вирішена приведенням *A* до верхньої блокової трикутної форми, з ненаведеним діагональним блоком. Рішення може бути знайдено шляхом зворотної перестановки.

[*p*, *q*, *r*] = *dmperm*(*A*) – знаходить перестановку рядків *p* та перестановку стовпців *q* квадратна матриці *A*, таку що *A*(*p*, *q*) – матриця в блоці верхньої трикутної форми. Третій вихідний аргумент *r* – цілий вектор, що описує межі блоків. *K*-ий блок матриці *A*(*p*, *q*) має індекси *r*(*k*) : *r*(*k*+1) – 1.

[*p*, *q*, *r*, *s*] = *dmperm*(*A*) – знаходить перестановки *p* і *q* та вектори індексів *r* і *s*, так що матриця *A*(*p*, *q*) перетворюється у верхню трикутню форму. Блок має індекси (*r*(*i*) : *r*(*i*+1) – 1, *s*(*i*) : *s*(*i*+1) –1).

***p* = *randperm*(*n*)** – повертає випадкові перестановки цілих чисел 1: *n*. Приклад:

```
» randperm(6)
```

```
ans = 2 4 3 6 5 1
```

Можна використати команду ***spparms*,** щоб змінити деякі опції та параметри, пов'язані з евристикою в алгоритмі.

Алгоритм упорядкування для симетричних матриць ґрунтується на алгоритмі впорядкування за розрідженістю стовпців. Фактично *symamd*(*S*) тільки формує матрицю *K* з структурою ненульових елементів, такий що *K*'×*K* має ту ж ненульову структуру, що і *S*, і потім викликає алгоритм упорядкування по розрідженості стовпців для *K*. Приклад:

```
» B=bucky;p= symamd(B);
» R=B(p,p);
» subplot(1,2,1 ),spy(B);subplot(1,2,2),spy(R)
```

На рис. 3.1 наводиться приклад застосування функції *symamd* до елементів розрідженої матриці.

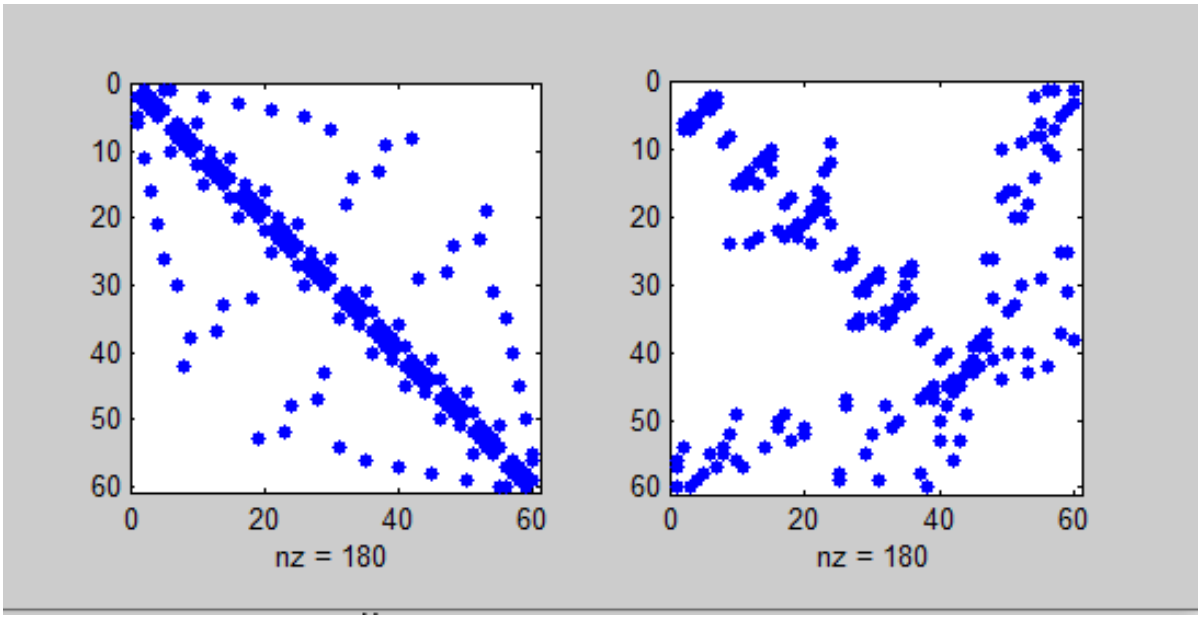


Рис. 3.1. Приклад застосування функції *symamd.*

***r* = *symrcm*(*S*)** – повертає вектор упорядкованості для симетричної матриці *S* і називається упорядкуванням Катхілла-Маккі. Причому формується така

перестановка *r*, що *S*(*r*, *r*) концентруватиме ненульові елементи поблизу діагоналі. Упорядкування застосовується для симетричних та несиметричних матриць.

Для речової симетричної розрідженої матриці *S* власні значення *S*(*r*, *r*) збігаються з власними значеннями *S*, але обчислення *eig*(*S*(*r*, *r*)) буде витрачатися менше часу, ніж *eig*(*S*). Приклад:

```
» B=bucky;p=symrcm(B);
» R=B(p,p);
» subplot(1,2,1),spy(B);subplot(1,2,2),spy(R)
```

На рис. 3.2. наведено приклад концентрації ненульових елементів розрідженої матриці поблизу головної діагоналі.

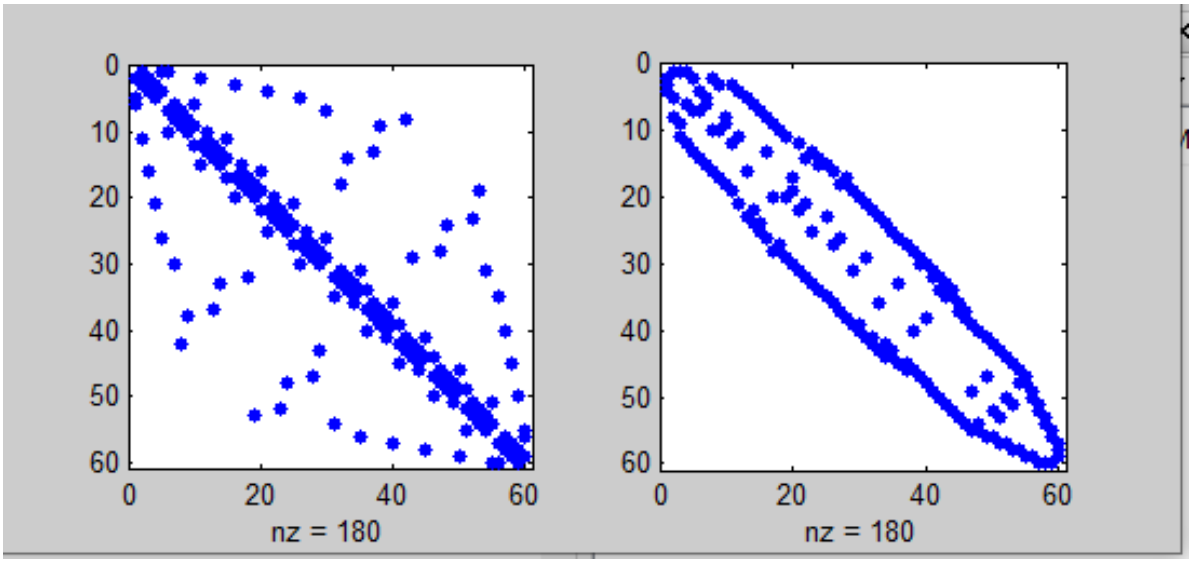


Рис. 3.2. Приклад застосування функції *symrcm*.

Система комп'ютерної математики MATLAB пропонує користувачам унікальний набір матричних операторів та функцій – помітно повніший, ніж у інших математичних систем. Це відкриває найширші можливості у вирішенні всіх видів математичних завдань, у яких використовуються сучасні матричні методи. Зокрема, наведені вище матричні функції, використовуються в пакеті математичного моделювання блочно-заданих систем *Simulink*, пакеті обробки

сигналів *Signal*, пакеті створення нейронних мереж *Network* і в ряді інших пакетів розширення системи MATLAB.

Контрольні запитання:

1. Перелічите функції для створення матриць, що існують у системі комп'ютерної математики MATLAB та наведіть приклади їх застосування?
2. Перелічите типи функцій перестановки елементів матриць, що існують у системі комп'ютерної математики MATLAB та поясніть різницю між ними?
3. Які функції забезпечують поворот матриць у системі комп'ютерної математики MATLAB та наведіть приклади їх застосування?
4. Які функції обчислення добутків елементів матриць використовуються у системі комп'ютерної математики MATLAB та наведіть приклади їх застосування?
5. Які функції обчислення сум елементів матриць використовуються у системі комп'ютерної математики MATLAB та наведіть приклади їх застосування?
6. Перелічите функції для обчислення спеціальних матриць, що існують у системі комп'ютерної математики MATLAB та наведіть приклади їх застосування?
7. Які існують функції виділення трикутних частин матриць у системі комп'ютерної математики MATLAB та наведіть приклади їх застосування?
8. Перелічите матричні функції для вирішення завдань лінійної алгебри, що існують у системі комп'ютерної математики MATLAB та наведіть приклади їх застосування?
9. Перелічите функції для роботи із розрідженими матрицями, що існують у системі комп'ютерної математики MATLAB та наведіть приклади їх застосування?
10. Які існують алгоритми та функції для упорядкування матриць у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?

РОЗДІЛ 4

ОСНОВИ ПРОГРАМУВАННЯ В СИСТЕМІ КОМП'ЮТЕРНОЇ МАТЕМАТИКИ MATLAB

4.1. Засоби програмування системи MATLAB. М-файли

MATLAB, будучи найпотужнішою системою для чисельних розрахунків, має розвинену мову програмування, що легко адаптується до завдань користувача. Програмами в системі MATLAB є файли М текстового формату, що містять запис програм у вигляді програмних кодів. Будь-яке таке визначення сценарію розрахунків чи нової функції відразу стає частиною засобів мови.

4.1.1. Ієрархія типів даних у системі MATLAB

Оскільки в основу мови системи MATLAB закладено об'єктно-орієнтоване програмування, то важливе місце при цьому має ієрархія типів даних (рис. 4.1).

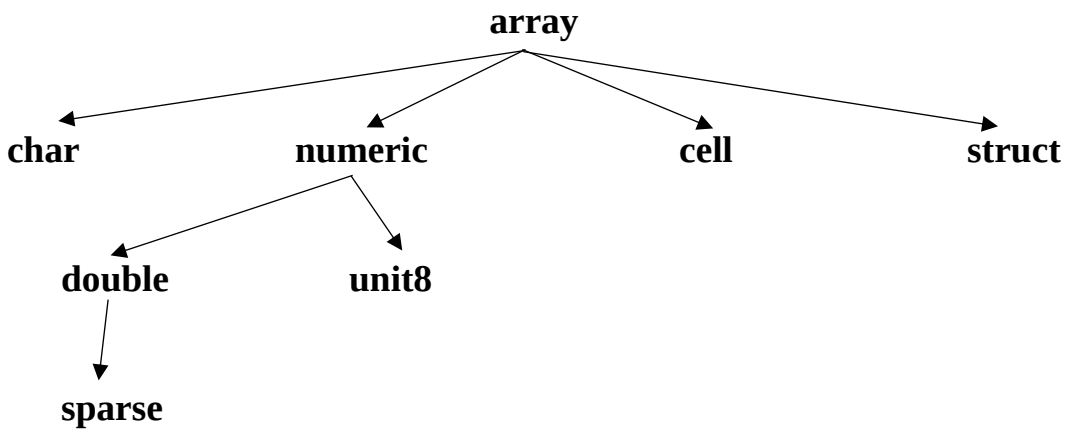


Рис. 4.1. Діаграма типів даних системи MATLAB

Таким чином, у MATLAB визначено наступні 6 основних типів даних, що є багатовимірними масивами:

double – числові масиви з числами подвоєної точності;

char – малі масиви з елементами-символами;

sparse – розріджені матриці з елементами – числами подвоєної точності;

cell – масиви комірок, які можуть бути масивами;

struct – масиви записів з полями, які можуть містити масиви;

*unit*8 – масиви 8-розрядних цілих чисел без знаків (математичні операції з ними не передбачені).

Крім того, передбачено ще один тип даних *UserObject* – типи даних (об'єкти), що визначаються користувачем. Типи даних *double, char* і *sparse* були розглянуті раніше. Що стосується чисел класу unite, то вони мають десяткові значення від 0 до 255, займають у пам'яті приблизно третину того, що займає десяткове число з подвійною точністю, і застосовуються в основному у службових цілях.

Кожному типу даних можна співвіднести деякі характерні їм операції. Дочірні типи даних, розташовані на наведеній діаграмі нижче батьківських типів, успадковують від останніх їх методи, що проявляється як ознака успадкування характеристик об'єктів. Оскільки в ієрархії типів даних зверху знаходяться дані типу array, це означає, що всі види даних у MATLAB – масиви.

Мова програмування системи MATLAB увібрала всі засоби, необхідні для реалізації різних видів програмування:

- процедурного;
- операторного;
- функціонального;
- логічного;
- структурного (модульного);
- об'єктно-орієнтованого;
- візуально-орієнтованого.

Переважна більшість функцій та команд мови MATLAB є цілком закінченими модулями, обмін даними між якими відбувається через їх вхідні параметри, хоча можливий і через глобальні змінні основного модуля програми.

Програмні модулі оформлені у вигляді текстових М-файлів, які зберігаються на жорсткому диску та підключаються до програм за потребою. Важливо, що на відміну від багатьох мов програмування застосування тих чи інших модулів не вимагає попереднього оголошення, а для створення та налагодження самостійних модулів MATLAB має всі необхідні засоби. Переважна більшість команд та функцій системи MATLAB представляється у вигляді таких модулів.

Об'єктно-орієнтоване програмування також представлене у системі MATLAB. Особливо воно корисне під час програмування завдань графіки. Що стосується візуально-орієнтованого програмування, то в MATLAB воно застосовується в основному в пакеті моделювання заданих блоками пристроїв та систем - Simulink та деяких інших пакетів розширення.

Важливе значення має двоякість операторів та функцій. Багато операторів мають свої аналоги як функцій. Так, оператор + має аналог у вигляді функції *sum*. Команди, записані у вигляді *Command argument,* мають нерідко форму запису і як функції *Command*('*argument*').

Приклади:

```
» help sin
SIN Sine.
SIN(X) is the sine of the elements of X.
Overloaded methods help sym/sin.m
» help('sin')
SIN Sine.
SIN(X) is the sine of the elements of X.
Overloaded methods
help sym/sin.m
» type('sin')
sin is a built-in function.
» type sin
sin is a built-in function.
```

Зазначена двоякість лежить в основі вибору між процедурним та функціональним типом програмування, кожен з яких має своїх шанувальників та

противників і може (тією чи іншою мірою) підходити для вирішення різних класів завдань. При цьому, перехід від одного типу програмування до іншого можливий у межах однієї програми і відбувається настільки природно, що більшість користувачів навіть не замислюються над тим, яким типом (або стилем) програмування вони переважно користуються.

4.1.2. M-файл сценарій

Файл сценарію, що містить текст програми, іменований також *Script*-файлом, є просто записом серії команд без вхідних і вихідних параметрів. Він має таку структуру:

```
%Основний коментар
%Додатковий коментар
Тіло файлу з будь-якими виразами
```

Важливі такі властивості файлів-сценаріїв:

- вони не мають вхідних та вихідних аргументів;
- працюють з даними із робочої області;
- у процесі виконання не компілюються;
- є зафіксованою у вигляді файлу послідовністю операцій, абсолютно аналогічною тій, що використовується в сесії.

Основним коментарем є перший рядок текстових коментарів, а додатковим – наступні рядки. Основний коментар виводиться під час виконання команд *lookfor* і `help Ім'я_каталогу`. Повний коментар виводиться під час виконання команди `help Ім'я_файла`. Розглянемо наступний файл-сценарій:

```
%Plot with color red
%Будує графік синусоїди лінією червоного кольору
%з виведеною масштабною сіткою в інтервалі [xmin,xmax]
x=xmin:0.1:xmax;
plot(x,sin(x),'r')
grid on
```

Перші три рядки – коментарі, решта – тіло файлу. Зверніть увагу на можливість завдання коментарів не англійською мовою. Знак %(відсотку) у коментарях повинен перебувати у першій позиції рядка. В іншому випадку команда *help name* не сприйматиме коментар як коментар (іноді це потрібно) і повертатиме повідомлення виду – *No help comments found in name.m*.

Візьміть до уваги, що такий файл не можна запустити без попередньої підготовки – завдання значень змінним *xmin*, *xmax*, використовуваним у тілі файла. Це наслідок першої якості файлів сценаріїв – вони працюють із даними з робочої області. Змінні, що використовуються у файлах-сценаріях, є глобальними, тобто вони діють однаково в командах сесії і всередині програмного блоку, яким є файл-сценарій. Тому задані у сесії значення змінних використовуються й у тілі файла. Імена файлів-сценаріїв не можна використовувати як параметри функцій, оскільки файли-сценарії не повертають значень.

4.1.3. Структура та властивості М-файл-функцій

М-файл-функція є типовим об'єктом мови програмування системи MATLAB. Одночасно він є повноцінним модулем з погляду структурного програмування, оскільки містить вхідні та вихідні параметри та використовує апарат локальних змінних. Структура такого модуля з одним вихідним параметром:

```
function yar=f_name(Список_параметрів)
%Основний коментар
%Додатковий коментар
Тіло файлу з будь-якими виразами
var=вираз
```

М-файл-функція має такі властивості:

- він починається з оголошення типу - *function*, після якого вказується ім'я змінної *var* – вихідного параметра, ім'я самої функції та список її вхідних параметрів;

- функція повертає своє значення і може використовуватись у вигляді *name* (Список_параметрів) у математичних виразах;
- усі змінні, що є в тілі файл-функції, є локальними, тобто діють лише в межах тіла функції;
- файл-функція є самостійним програмним модулем, який спілкується з іншими модулями через свої вхідні та вихідні параметри;
- правила виведення коментарів ті самі, що й у файл-сценаріїв;
- файл-функція є засобом розширення системи MATLAB;
- при виявленні файл-функції він компілюється і потім виконується, створені машинні коди зберігаються у робочій області системи MATLAB.

Остання конструкція *var* = вираз вводиться, якщо треба, щоб функція повертала результат обчислень. Якщо М-файл-функція завершується рядком з точкою з комою (;), то повернення значення функції використовується програмний оператор *return*.

Наведена форма файл-функції характерна для функції з одним вихідним параметром. Якщо вихідних параметрів більше, вони вказуються в квадратних дужках після слова *function*. При цьому структура модуля має вигляд:

```
function [var1 ,var2,...]=f_name(Список_параметрів)
%Основний коментар
%Додатковий коментар
Тіло файлу з будь-якими виразами
var1=вираз
var2=вираз
..........
```

Така функція багато в чому нагадує повноцінну процедуру. Її не можна використовувати прямо в математичних виразах, оскільки вона повертає не єдиний результат, а безліч результатів – за кількістю вихідних параметрів. Тому, як зазначалося, ця функція використовується як окремий елемент програм виду: `[var1, var2,...]=f_name(Список_параметрів)`

Після його застосування змінні виходу *var*1, *var*2,... стають певними та їх можна використовувати у наступних математичних виразах та інших сегментах програми. Якщо функція використовується як `f_name(Список_параметров)`, то повертається значення лише першого вихідного параметра - змінної var1.

Отже, з наведеного зрозуміло, що змінні у файл-сценаріях є глобальними, а файл-функціях – локальними. При цьому застосування глобальних змінних програмних модулів здатне створювати побічні ефекти. Застосування локальних змінних усуває цю можливість та відповідає вимогам структурного програмування.

Однак передача даних з модуля у модуль в цьому випадку відбувається тільки через вхідні і вихідні параметри, що вимагає ретельного планування цієї передачі. При створенні файл-функцій часом бажано застосування глобальних змінних. Відповідальність за це має брати на себе програміст, який створює програмні модулі.

Команда `global varl var2 ...` дозволяє оголосити змінні модулі – функції глобальними. Таким чином, усередині функції можуть використовуватися і такі змінні, якщо це потрібно за умовами вирішення конкретного завдання.

Починаючи з версії 5.0, у функції СКМ MATLAB можна включати підфункції. Вони оголошуються та записуються в тілі основних функцій та мають ідентичну їм конструкцію. Не слід плутати ці функції з внутрішніми функціями, вбудованими в ядро системи MATLAB. Нижче наведено приклад функції з такою підфункцією:

```
function [mean,stdev] = statv(x)
%STATV Interesting statistics.
%Приклад функції зі вбудованою підфункцією
n = length(x);
mean = avg(x,n);
stdev = sqrt(sum((x-avg(x,n)).^2)/n);
function m = avg(x,n) %Mean subfunction
m = sum(x)/n;
```

У цьому прикладі середнє значення елементів вектору *x* обчислюється за допомогою підфункції *avg*(*x, n*), тіло якої записано в тілі основної функції *statv*. Приклад використання функції *statv* представлений нижче:

```
» V=[1 2 3 4 5]
V =
     1     2     3     4     5
» [a,m]=statv(V)
a =
     3
m =
    1.4142
» statv(V)
ans =
     3
» help statv
STATV Interesting statistics.
```

Підфункції визначені та діють локально, тобто лише в межах М-файлу, що визначає основну функцію. Команда *help name* виводить коментар, що стосується лише основної функції, тоді як команда *type name* виводить повністю вміст М-файлу. Так що задані в деякому М-файл підфункції не можна використовувати ні в командному режимі роботи, ні в інших М-файлах.

При зверненні до функції інтерпретатор системи MATLAB передусім переглядає М-файл щодо виявлення підфункцій. Якщо їх виявлено, то вони задаються як локальні функції. Завдяки локальному впливу підфункцій їх імена можуть збігатися з іменами основних функцій системи. Якщо функції та підфункції повинні використовувати загальні змінні, їх треба оголосити глобальними як у функції, так і в її підфункціях.

Для запису М-файлів використовуються каталоги, які називаються батьківськими каталогами. Вони містять групи файлів певного функціонального призначення, наприклад, за статистичними розрахунками, матричними операціями, обчисленням певних класів функцій і так далі.

Однак починаючи з версії MATLAB 5.0 з'явилася можливість у батьківських каталогах створювати приватні каталоги – *private*. М-файли, що входять до них, доступні тільки файлам батьківського каталогу. Файли приватних каталогів переглядаються в першу чергу інтерпретатором системи MATLAB. Застосування приватних каталогів дозволяє змінювати вихідні файли, зберігаючи оригінали в батьківському каталозі в незмінному вигляді.

Якщо ви вирішили відмовитися від зміненого файлу, достатньо стерти його в приватному каталозі. Така можливість пов'язана з тим, що інтерпретатор при виявленні імені М-файлу перш за все переглядає приватний каталог файлу і інтерпретує знайдений у ньому файл. І лише якщо файл не знайдено, шукається файл у батьківському каталозі.

4.1.4. Дії для встановлення шляхів

Починаючи з 6-ї версії MATLAB є можливість визначення поточного каталогу та шляхів пошуку. Встановлення цих властивостей здійснюється або за допомогою відповідних діалогових вікон, або команд з командного рядка.

Поточний каталог визначається в діалоговому вікні *Current Directory* робочого середовища.

Поточний каталог обирається зі списку. Якщо його немає в списку, його можна додати з діалогового вікна *Browse for Folder*, викликаного натисканням на кнопку, розташовану праворуч від списку. Вміст поточного каталогу відображається у таблиці файлів.

Визначення шляхів пошуку здійснюється у діалоговому вікні *Set Path* навігатора шляхів, доступ до якого здійснюється з пункту *Set Path* меню *File* робочого середовища.

Для додавання каталогу натисніть кнопку *Add Folder* і в діалоговому вікні *Browse for Path* виберіть потрібний каталог. Додавання каталогу з його підкаталогами здійснюється при натисканні на кнопку *Add with Subfolders*. Шлях до доданого каталогу з'являється у полі MATLAB *search path*. Порядок пошуку

відповідає розташуванню шляхів у цьому полі, першим проглядається каталог, шлях до якого розміщено вгорі списку. Порядок пошуку можна змінити або взагалі видалити шлях до якогось каталогу, для чого виділіть каталог у полі MATLAB *search path* і визначте його положення за допомогою таких кнопок:

Move to Top – помістити вгору списку;

Move Up – перемістити вгору на одну позицію;

Remove – видалити зі списку;

Move Down – перемістити вниз на одну позицію;

Move to Bottom – розмістити вниз списку.

Після внесення змін слід зберегти інформацію про шляхи пошуку, натиснувши кнопку *Save*. За допомогою кнопки *Default* можна відновити стандартні установки, а *Revert* призначена для повернення до збережених.

4.1.5. Команди для встановлення шляхів

Дії встановлення шляхів в MATLAB 6.x дублюються командами. Поточний каталог встановлюється командою *cd*, наприклад, *cd c:\users\igor*. Команда *cd*, викликана без аргументу, виводить шлях до поточного каталогу. Для встановлення шляхів служить команда *path*, що викликається з двома аргументами:

path(*path, 'c:\users\igor'*) – додає каталог *c:\users\igor* з нижчим пріоритетом пошуку, а *path*(*'c: \users\igor', path*) – додає каталог *c:\users\igor* з найвищим пріоритетом пошуку.

Використання команди *path* без аргументів призводить до відображення на екрані списку шляхів пошуку. Видалити шлях зі списку можна за допомогою команди *rmpath* (*'c:\users\igor'*), яка видаляє шлях до каталогу *c:\users\igor* зі списку шляхів.

Зауваження. Не видаляйте без необхідності шляхи до каталогів, особливо до тих, у яких ви не впевнені. Видалення може призвести до того, що частина функцій, визначених у MATLAB, стане недоступною.

4.1.6. Робота з помилками

Часто під час обчислень виникають помилки. Наприклад, ми стикалися з проблемою обчислення функції *sin*(*x*)/*x* при *x* = 0 – виходить помилка виду "ділення на нуль". При появі помилки обчислення можуть завершитися достроково та виводиться повідомлення про помилку. Слід зазначити, що не всі помилки породжують зупинку обчислень. Деякі супроводжуються лише видачею попереджувальної інформації.

Такі ситуації повинні враховуватись програмістом, відзначатися як помилкові та, за можливості, усуватися. Для виведення повідомлення про помилку служить команда *error*('Повідомлення про помилку'), у виконанні якої обчислення перериваються і видається повідомлення про помилку, задане у апострофах. Нижче наведено приклад завдання функції *sd*(*x*)=*sin*(*x*)/*x*, в якому встановлено повідомлення про помилку:

```
function f=sd(x)
if x==0 error('Помилка – ділення на 0'), end
f=sin(x)/x
```

Для виявлення ситуації про помилку використано оператора умовного переходу if, який буде описаний детально дещо пізніше. Результат виконання цієї функції представлений нижче:

```
» sd(1)
f =
0.8415
ans =
0.8415
» sd(0)
??? Error using ==> sd
Помилка – ділення на 0
```

Якщо зупинка програми з появою помилки небажана, може використовуватися команда виведення попереджуючого повідомлення:

```
warning('Попереджувальне повідомлення')
```

Ця команда виводить повідомлення, що стоїть в апострофах, але не перешкоджає подальшій роботі програми. Ознакою того, що є помилкою, а що – попередженням, є символи ??? та слово *Warning* у відповідних повідомленнях.

Досвідчені програмісти повинні передбачати ситуації з появою помилок та виключати їх. Наприклад, при $x = 0$ вираз $sin(x)/x = 0/0 = 1$, отже було досить замість його обчислення використовувати значення 1.

У цьому простому прикладі можна легко скласти функцію *sd*0, що виключає обчислення $sin(x)/x$ при $x = 0$:

```
function f=sd0(x)
if x==0 f=1; else f=sin(x)/x; end
return
```

При цьому обчислення пройдуть коректно за будь-якого *x*:

```
»sd0(1)
ans =
0.8415
» sd0(0)
ans =
1
```

Для виведення повідомлення про останню зроблену помилку є функція *lasterr*:

```
» aaa
??? Undefined function or variable 'aaa'.
» 2+3
ans =
5
» 1/0
Warning: Divide by zero.
ans =
Inf
```

```
» lasterr
ans =
Undefined function or variable 'aaa'.
```

Як неважко помітити, функція *lasterr* повертає текстове повідомлення, яке слідує за знаками ??? повідомлення про помилку.

У програмах можуть міститись обробники помилок, які направляють хід обчислень в потрібне русло, навіть якщо помилка з'являється. Але для цього потрібні засоби індикації та обробки помилок. Основними з них є функції *eval* та *lasterr*. Про функцію *lasterr* вже говорилося, а функція

```
Eval('try', 'catch')
```

на відміну від раніше розглянутої форми має два вхідні аргументи. Один - це рядковий вираз, що перетворюється на виконувану форму і виконується за відсутності помилки. Якщо ж помилка, то рядок 'catch' дає звернення до функції обробки помилки.

4.1.7. Функції підрахунку числа вхідних та вихідних аргументів

При створенні функцій зі спеціальними властивостями дуже корисні дві наведені нижче функції:

nargin – повертає кількість вхідних параметрів цієї функції,

nargout – повертає число вихідних параметрів даної функції.

Нехай, наприклад, хочемо створити функцію, яка обчислює суму квадратів п'яти аргументів *x*l, *x*2, *x*3, *x*4 і *x*5. Створюємо функцію з ім'ям *sum*2_5:

```
function f=sum2_5(x1,x2,x3,x4,x5);
f=x1^2+x2^2+x3^2+x4^2+x5^2;
```

Перевіремо її у роботі:

```
» sum2_5(1,2,3,4,5)
ans =
55
```

```
» sum2_5(1,2)
??? Input argument 'x3' is undefined.
Error in ==> C:\MATLAB\bin\sum2_5.m
On line 2 ==> f=x1^2+x2^2+x3^2+x4^2+x5^2;
```

Отже, за всіх п'яти аргументів функція працює коректно. Але якщо аргументів менше ніж п'ять, вона видає повідомлення про помилку. За допомогою функції nargin можна створити функцію *sum*2_5*m*, яка працює коректно за будь-якої кількості заданих вхідних аргументів у межах від 1 до 5:

```
function f=sum2m_5(x1,x2,x3,x4,x5);
n=nargin;
if n==1 f=x1^2; end
if n==2 f=x1^2+x2^2; end
if n==3 f=x1^2+x2^2+x3^2; end
if n==4 f=x1^2+x2^2+x3^2+x4^2; end
if n==5 f=x1^2+x2^2+x3^2+x4^2+x5^2; end
```

У цій функції використовується умовний оператор *if-end*, який буде описано далі. Але і так зрозуміло, що завдяки застосуванню функції *nargin* і оператора умовного виразу обчислення йдуть за формулою з кількістю доданків, що дорівнює кількості вхідних аргументів. Це видно з наведених нижче прикладів:

```
» sum2_5m(1)
ans =
1
» sum2_5m(1,2)
ans =
5
» sum2_5m(1,2,3)
ans =
14
» sum2_5m(1,2,3,4)
ans =
30
» sum2_5m(1,2,3,4,5)
ans = 55
» sum2_5m(1,2,3,4,5,6)
```

```
??? Error using ==> sum2_5m
Too many input arguments.
```

Отже, при зростанні числа вхідних операторів від 1 до 5 обчислення проходять правильно. При більшій кількості операторів виводиться повідомлення про помилку. Це вже діє вбудована в інтерпретатор MATLAB система діагностики помилок.

4.1.8. Особливості виконання М-файл-функцій

М-файл-функції можуть використовуватися як у командному режимі, так і у інших М-файлах. При цьому необхідно вказувати всі вхідні та вихідні параметри. Винятком є випадок, коли вихідний параметр єдиний – у разі функція повертає єдиний результат і можна використовувати у математичних виразах. При використанні глобальних змінних вони повинні бути оголошені у всіх М-файлах, що використовуються у вирішенні заданої задачі та у всіх вбудованих підфункціях, що входять до них.

Імена функцій мають бути унікальними. Це пов'язано з тим, що при виявленні кожного нового імені MATLAB перевіряє, чи це ім'я не є іменем змінної, підфункції в даному М-файлі, приватної функції в каталогах *private* і ім'ям функції в шляху доступу до цієї функції. Якщо остання зустрічається, то буде виконано саме цю функцію. У новій версії MATLAB можливе перевизначення функції, але навряд чи це рекомендується робити переважній більшості користувачів системою.

Якщо аргумент функції використовується лише для обчислень та його значення не змінюються, то аргумент передається посиланням, що зменшує витрати пам'яті. В іншому випадку аргумент передається значенням. Для кожної функції виділяється своя (робоча) область пам'яті, окрема від області, що надається системі MATLAB. Глобальні змінні належать низці областей пам'яті. Модифікація яких змінює вміст усіх областей пам'яті.

Під час вирішення завдань із великим обсягом даних може відчуватися нестача оперативної пам'яті. Ознакою цього стає поява повідомлення про помилку "Out of memory". У цьому випадку корисне застосування кількох заходів:

- стирання даних, що стали непотрібними - насамперед великих масивів;
- збільшення розмірів файлу підкачки *Windows*;
- зменшення розміру даних, що використовуються;
- зняття обмежень на розміри пам'яті, що використовується;
- збільшення ємності пам'яті.

Чим більша ємність ОЗУ комп'ютера, на якому використовується система MATLAB, тим менша ймовірність виникнення зазначеної помилки. Досвід показує, що навіть при вирішенні завдань помірної складності ємність ОЗУ має бути не менше за 64 Мбайт або вищою.

4.1.9. Створення P-кодів

Коли зустрічається сценарій чи функція як М-файл, то щоразу виконується трансляція файлів, створює так звані *P*-коди (псевдокоди). Вона пов'язана із синтаксичним контролем сценарію або функції, що трохи сповільнює обчислення. MATLAB дозволяє створювати *P*-коди сценаріїв та функцій за допомогою команди. Тимчасові *P*-коди зберігаються в пам'яті до використання команди *clear* або завершення сеансу роботи.

```
pcode имя_М-файла
```

Особливо корисне застосування цієї команди у тому випадку, коли використовується складна дескрипторна графіка та засоби створення *GUI*. У цьому випадку виграш за швидкістю виконання обчислень може бути помітним. Перехід до *P*-кодів корисний у тому разі, коли користувач бажає приховати створений ним М-файл і реалізовані у ньому ідеї та алгоритми. Файл з *P*-кодами має розширення. Розмір файлу з *P*-кодами зазвичай більший, ніж М-файлу.

Розглянемо наступний приклад – створимо файл-сценарій *pp.m*:

```
told=cputime;
x=-15:.0001:15;
plot(x,sin(x))
t=cputime-told
```

Ця програма будує графік функції *sin*(*x*) за великою кількістю точок. Крім того, вона обчислює час у секундах виконання цього сценарію. При першому пуску отримаємо:

```
» PP
t = 0.4400
```

Тепер виконаємо створення *P*-кодів і знову виконаємо програму:

```
» pcode pp
»PP
t = 0.3900
» PP
t = 0.3300
```

Неважко помітити, що після перетворення на *P*-коди час побудови графіка дещо зменшився. Але набагато важливіше, що тепер можна стерти файл *pp.m* (але залишити *pp.p*) і знову пустити програму. Ваші дуже цікаві колеги навряд чи розберуться з тим, що записано в машинних кодах файлу *pp.p*, хоча за допомогою спеціальних програм (декомпіляторів) така можливість існує.

4.2. Умовний оператор *if-elseif-else-end*

Умовний оператор *if* у загальному вигляді записується так:

```
if Умова
Інструкції_1
elseif Умова
```

```
Інструкції_2
else
Інструкції_3
End
```

Ця конструкція припускає кілька окремих варіантів. У найпростішому типу *if-end*

```
if Умова Інструкції end
```

Поки Умова повертає логічне значення 1 (тобто вірно), виконуються «Інструкції», що становлять тіло структури *if-end*. У цьому оператор *end* вказує кінець переліку інструкцій. Інструкції у списку поділяються оператором , (кома) або ; (крапка з комою). Якщо «Умова» не виконується (дає логічне значення 0), «Інструкції» не виконуються.

Ще одна конструкція

```
if Умова Інструкції_1 else Інструкції_2 end
```

виконує «Інструкції_1» якщо виконується «Умова» та «Інструкції_2» в іншому випадку. Умови записуються наступним чином:

```
Вираз_1 Оператор_відносини Вираз_2,
```

причому як `Оператор_відносини` використовуються такі оператори: ==, <, >, <=, >= або ~=. Всі ці оператори представлені подвійними символами без пробілу з-поміж них.

Ми вже неодноразово показували застосування цієї загальновідомої структури управління в програмних модулях. Читач може випробувати свої варіанти програм з умовним оператором.

Приклад використання оператора *if-elseif-else*, в якому аналізується значення змінної *a* та виводиться повідомлення про величину *a*, наведено у наступному лістингу

```
function ifdem(a)
% приклад використання оператора if-elseif-else
if (a == 0)
    warning('a дорівнює нулю')
elseif a == 1
    warning('a дорівнює одиниці')
elseif a == 2
    warning('a дорівнює двум')
elseif a >= 3
    warning('a, більше або дорівнює трьом')
else
    warning('a менше трьох, та не дорівнює нулю, одиниці,
двум')
end
```

4.3. Цикли типу *for-end*

Цикли типу *for-end* зазвичай використовуються для організації циклічних обчислень із заданим числом циклів, що повторюються. Конструкція такого циклу має вигляд:

```
for var=Вираз, Інструкція Інструкція end.
```

Оператор призначений для виконання заданого числа дій, що повторюються. Найпростіше використання оператора *for* здійснюється наступним чином:

```
for count = start:step:final
% Далі йде текст програми із команд MATLAB
end
```

Тут *count* – змінна циклу, *start* – її початкове значення, *final* – кінцеве значення, а *step* – крок, на який збільшується *count* при кожному наступному заході в цикл. Цикл закінчується, як тільки значення *count* стає більшим за *final*. Змінна циклу може набувати як цілі, так й речові значення будь-якого знака. Розберемо застосування оператора циклу для деяких характерних прикладів.

Нехай потрібно вивести сімейство кривих для $x \in [0, 2\pi]$, яке задано функцією, яка залежить від параметра $y(x,a) = e^{-ax}\sin x$, для значення параметру від –0,1 до 0,1.

```
% файл-програма для побудови сімейства кривих
x = [0:pi/30:2*pi];
for a = -0.1:0.02:0.1
    y = exp(-a*x).*sin(x);
    hold on
    plot(x, y)
end
```

В результаті виконання з'явиться графічне вікно (рис. 4.2), яке містить потрібне сімейство кривих.

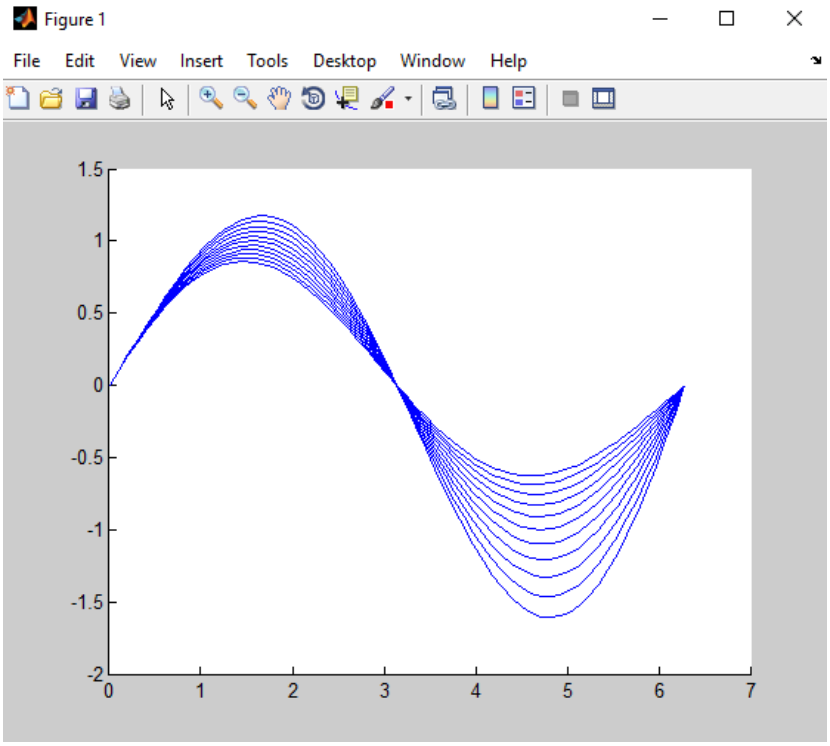


Рис. 4.2. Результат виконання.

Зауваження. Редактор М-файлів автоматично пропонує розташувати оператори всередині циклу з відступом від лівого краю. Використовуйте цю можливість для зручності роботи з текстом програми.

Напишемо файл-програму для обчислення суми:

$$S = \sum_{k=1}^{10} \frac{1}{k!}.$$

Алгоритм обчислення суми використовує накопичення результату, тобто спочатку сума дорівнює нулю ($S = 0$), потім змінну k заноситься одиниця, обчислюється $1/k!$, додається до S і результат знову заноситься в S. Далі k збільшується на одиницю, і процес триває, поки останнім доданком не стане $1/10!$ Файл-програма *Fordem*2, наведена в наступному лістингу, обчислює суму, яку шукає.

Лістинг файл-програми *Fordem*2 для обчислення суми

```
% ФАЙЛ-ПРОГРАМА ДЛЯ ОБРАХУВАННЯ СУМИ 1/1!+1/2!+ … +1/10!
S = 0;
% накопичення суми в циклі
for k = 1:10
    S = S + 1/factorial(k);
End
% вивід результату
S
```

Результат з'явиться у командному вікні, т.я. в останньому рядку файл-програми *S* міститься без крапки з комою для виведення значення змінної *S*

```
S =
    1.7183
```

Зверніть увагу, що решта рядків файл-програми, які могли б спричинити виведення на екран проміжних значень, завершуються крапкою з комою для запобігання виведення їх у командне вікно.

Перші рядки з коментарями не випадково відокремлені порожнім рядком від решти тексту програми. Саме вони виводяться на екран, коли користувач за допомогою команди *help* з командного рядка отримує інформацію про те, що робить *Fordem*2

```
» help Fordem2
 ФАЙЛ-ПРОГРАМА ДЛЯ ОБРАХУВАННЯ СУМИ 1/1!+1/2!+ … +1/10!
```

Під час написання файл-програм та файл-функцій не нехтуйте коментарями!

Усі змінні, що використовуються у файл-програмі, стають доступними у робочому середовищі. Вони є так званими глобальними змінними. З іншого боку, у файл-програмі можуть бути використані всі змінні, введені в робочому середовищі.

Розглянемо завдання обчислення суми, схожу на попередню, але яка залежить від змінної *x*

$$S = \sum_{k=1}^{10} \frac{x^k}{k!}.$$

Для обчислення цієї суми у файл-програмі *Fordem*2 потрібно змінити рядок усередині циклу *for* на

```
S = S + x.^k/factorial(k);
```

Перед запуском програми слід визначити змінну *x*, *y* командному рядку за допомогою наступних команд:

```
» x = 1.5;
» Fordem2
S =
    3.4817
```

У якості *x* може бути вектор або матриця, так як у файл-програмі *Fordem*2 при накопиченні суми використовувалися поелементні операції.

Перед запуском *Fordem*2 потрібно обов'язково надати змінній *x* деяке значення, а для обчислення суми, наприклад з п'ятнадцяти доданків, доведеться внести зміни до тексту файл-програми. Набагато краще написати універсальну файл-функцію, яка має вхідні аргументи значення *x* і верхньої межі суми та вихідне значення *S*(*x*). Файл-функція *sumN* наведена у наступному лістингу.

```
function S = sumN(x, N)
% файл-функція для обрахування суми
% x/1!+x^2/2!+ … +x^N/N!
% S = sumN(x, N)
S = 0;
% накопичення суми у циклі
for m = 1:1:N
    S = S + x.^m/factorial(m);
end
```

Про використання функції *sumN* можна дізнатися, набравши в командному рядку *help sumN*. У командне вікно виведуться перші три рядки з коментарями, відокремлені від тексту файл-функції порожнім рядком.

Зверніть увагу, що змінні файл-функції не є глобальними (*m* у файл-функції *sumN*). Спроба перегляду значення змінної *m* із командного рядка призводить до повідомлення про те, що *m* не визначено. Якщо в робочому середовищі є глобальна змінна з тим же ім'ям, визначена з командного рядка або файл-програмі, то вона ніяк не пов'язана з локальною змінною файл-функції. Як правило, краще оформляти власні алгоритми у вигляді файл-функцій для того, щоб змінні, які використовуються в алгоритмі, не змінювали значення однойменних глобальних змінних робочого середовища.

Цикли *for* можуть бути вкладені один в одного, причому змінні вкладених циклів повинні бути різними. Цикл *for* виявляється корисним при виконанні подібних дій, що повторюються, у тому випадку, коли їх число заздалегідь визначено. Обійти це обмеження дозволяє більш гнучкий цикл *while*.

4.4. Оператор циклу *while*

Розглянемо приклад обчислення суми, схожий на приклад із попереднього пункту. Потрібно знайти суму ряду для заданого x (розкладання в ряд $\sin x$):

$$S(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!}.$$

Суму можна накопичувати доти, поки доданки є не надто маленькими, скажімо більше за модулем 10^{-10}. Циклом for тут не обійтися, оскільки заздалегідь невідома кількість доданків. Вихід полягає у застосуванні циклу *while*, який працює, поки виконується умова циклу:

```
while умова циклу
команди MATLAB
end
```

У цьому прикладі умова циклу передбачає, що поточний доданок $x^k / k!$ більше 10^{-10}. Для запису цієї умови використовується знак більше (>). Текст файл-функції *mysin*, що обчислює суму ряду, наведено у наступному лістингу.

Лістинг файл-функції *mysin*, що обчислює синус розкладанням у ряд

```
function S = mysin(x)
% Обчислення сінусу разкладенням у ряд
% y = mysin(x), -pi<x<pi
S = 0;
k = 0;
while abs(x.^(2*k+1)/factorial(2*k+1))>1.0e-10
    S = S + (-1)^k*x.^(2*k+1)/factorial(2*k+1);
    k = k + 1;
end
```

Зверніть увагу, що цикл *while*, на відміну від *for*, не має змінної циклу, тому довелося до початку циклу *k* присвоїти нуль, а всередині циклу збільшувати *k* на одиницю.

Умова циклу *while* може містити операції відносин, наведені в табл. 4.1.

Завдання складніших умов проводиться із застосуванням логічних операторів.

Таблиця 4.1 – Операції відносин

Позначення	Операція відносин
==	Рівність
<	Менше
>	Більше
<=	Менше чи дорівнює
>=	Більше чи дорівнює
~=	Не дорівнює

Наприклад, умова $-1 \le x < 2$ полягає в одночасному виконанні двох нерівностей $x \ge -1$ та $x < 2$ та записується за допомогою логічного оператора *and*

```
and(x >= -1, x < 2)
```

або еквівалентним чином із символом *&*

```
(x >= -1) & (x < 2)
```

Логічні оператори та приклади їх використання наведені у табл. 4.2.

Таблиця 4.2 – Логічні оператори

Оператор	Умова	Запис у MATLAB	Еквівалентний запис
Логічне "І"	$x < 3$ та $k = 4$	*and*(x < 3, k == 4)	(x < 3) & (k == 4)
Логічне "АБО"	$x = 1,\ 2$	*or*(x == 1,x == 2)	(x == 1) | (x == 2)
Заперечення "НІ"	$a \ne 1.9$	*not*(a == 1.9)	~(a == 1.9)

При обчисленні суми нескінченного ряду є сенс обмежити кількість доданків. Якщо ряд розходиться через те, що його члени не прагнуть до нуля, то умова на мале значення поточного доданку може будь-коли виконатися і програма зациклиться. Виконайте підсумовування, додавши в умову циклу при файл-функції *mysin* обмеження на кількість доданків:

```
while(abs(x.^(2*k+1)/factorial(2*k+1))>1.0e-10)&
(k<=10000))
```

або в еквівалентній формі

```
while and(abs(x.^(2*k+1)/factorial(2*k+1))>1.0e-10),
k<=10000)
```

Організація повторюваних дій як циклів робить програму простою і зрозумілою, проте часто потрібно виконати той чи інший блок команд залежно від умов, тобто використовувати розгалуження алгоритму.

4.5. Оператор переривання циклу *break*

При організації циклічних обчислень слід дбати про те, щоб усередині циклу не виникло помилок. Наприклад, нехай заданий масив *x*, що складається з цілих чисел, і потрібно сформувати новий масив *y* за правилом $y(i) = x(i + 1) / x(i)$. Очевидно, що завдання може бути вирішено за допомогою циклу *for*. Але, якщо один із елементів вихідного масиву дорівнює нулю, то при розподілі вийде *inf* і наступні обчислення можуть виявитися марними. Запобігти цій ситуації можна виходом із циклу, якщо поточне значення *x(i)* дорівнює нулю. Наступний фрагмент програми демонструє використання оператора *break* для переривання циклу:

```
for x = 1:20
z = x-8;
    if z==0
        break
    end
    y = x/z
end
```

Як тільки змінна *z* набуває значення 0, цикл переривається.

Оператор *break* дозволяє достроково перервати виконання циклів *for* та *while*. Поза цими циклами оператор *break* не працює.

Якщо оператор *break* застосовується у вкладеному циклі, він здійснює вихід лише з внутрішнього циклу.

4.6. Конструкція перемикача *switch-case-otherwise-end*

Для здійснення множинного вибору або розгалуження може застосовуватись оператор *switch*. Він є альтернативою оператору *if-elseif-else*. Загалом застосування оператора розгалуження *switch* виглядає наступним чином:

```
switch вираз
     case значення 1
          команди MATLAB
     case значення 2
          команди MATLAB
……..……
     case значення N
          команди MATLAB
     case {значення N+1, значення N+2, …}
          команди MATLAB
……..……
     case {значення NM+1, значення NM+2,…}
     otherwise
          команди MATLAB
end
```

У цьому операторі спочатку обчислюється значення виразу (це може бути скалярне числове значення чи рядок символів). Потім це значення порівнюється зі значеннями: значення 1, значення 2, …, значення *N*, значення *N*+1, значення *N*+2, …, значення *NM*+1, значення *NM*+2,… (які можуть бути числовими чи рядковими). Якщо знайдено збіг, то виконуються команди MATLAB, які стоять після відповідного ключового слова *case*. В іншому випадку виконуються команди MATLAB, розташовані між ключовими словами *otherwise* та *end*.

Рядок із ключовим словом *case* може бути скільки завгодно, але рядок із ключовим словом *otherwise* повинен бути один.

Після виконання будь-якої з гілок відбувається вихід із *switch*, при цьому значення, задані в інших випадках, не перевіряються.

Застосування *switch* пояснює наступний приклад:

```
function demswitch(x)
a = 10/5 + x
switch a
    case -1
        warning('a = -1')
    case 0
        warning('a = 0')
    case 1
        warning('a = 1')
    case {2, 3, 4}
        warning('a дорівнює 2 або 3 або 4')
otherwise
        warning('a не дорівнює -1, 0, 1, 2, 3, 4')
end
» x = -4
demswitch(x)
a = -2
» x = 1
demswitch(x)
a = 3
Warning:a дорівнює 2 або 3 або 4
```

4.7. Створення паузи у обчисленнях

Для зупинки програми використовується оператор *pause*. Він використовується у таких формах:

pause – зупиняє обчислення до натискання будь-якої клавіші;

***pause*(*N*)** – зупиняє обчислення на *N* секунд;

pause on – вмикає режим створення пауз;

pause off – вимикає режим створення пауз.

Наступний приклад пояснює застосування команди *pause*:

```
for i=1:20;
x = rand(1,40);
y = rand(1,40);
z = sin(x.*y);
tri = delaunay(x,y);
trisurf(tri,x,y,z)
pause(1);
end
```

4.8. Поняття про об'єктно-орієнтоване програмування

Ми вже неодноразово згадували різні об'єкти мови програмування системи MATLAB. Це одна з ознак об'єктно-орієнтованого програмування системи. В основі об'єктно-орієнтованого програмування лежать три основні положення:

Інкапсуляція – об'єднання даних і програм та передача даних через вхідні і вихідні параметри функцій. В результаті з'являється новий елемент програмування – об'єкт.

Спадкування – можливість створення батьківських об'єктів та нових дочірніх об'єктів, що успадковують властивості батьківських об'єктів. Можливо також множинне успадкування, у якому клас успадковує властивості низки батьківських об'єктів і власної структури. На успадкування заснована система завдання типів даних, дескрипторна графіка та багато інших прийомів програмування. Приклади наслідування ми вже неодноразово помічали.

Поліморфізм – присвоєння деякій дії одного імені, яке надалі використовується по всьому ланцюжку об'єктів, що створюються зверху до низу. Причому кожен об'єкт виконує цю дію властивим йому способом.

Крім цих положень об'єктно-орієнтоване програмування в MATLAB допускає агрегування об'єктів, тобто об'єднання частин об'єктів чи низки об'єктів в одне ціле.

Об'єкт можна розглядати як деяку структуру, що належить до певного класу. У MATLAB визначено такі п'ять основних класів об'єктів:

double – числові масиви з елементами – числами подвійної точності;

sparse – двомірні числові або комплексні розріджені матриці;

char – масиви символів;

struct – масиви записів (структурні);

cell – масиви осередків.

З об'єктами цих класів ми багаторазово зустрічалися, особливо не обумовлюючи їхню приналежність до об'єктно-орієнтованого програмування. Для MATLAB взагалі характерно, що жодні класи об'єктів (зокрема, що заново створюються) не вимагають оголошення. Наприклад, створюючи змінну *name* = 'Петро', ми автоматично отримуємо об'єкт у вигляді змінної *name* класу *char*. Таким чином, для змінних приналежність до того чи іншого класу визначається їх значенням.

Для створення нових класів об'єктів є конструктори класів. Фактично це М-файли, ім'я яких збігається з ім'ям класів @Имя_- класу, але вони не матимуть символу @. Цим символом позначаються піддиректорії системи MATLAB, у яких є конструктори класів. Безліч таких директорій з прикладами конструкторів класів ви знайдете у піддиректоріях MATLAB та Toolbox.

Як приклад розглянемо піддиректорію *@SYM* у директорії *Toolbooks/Symbolic*. У цій піддиректорії можна знайти конструктори більш ніж сотні об'єктів пакета символьної математики. Наприклад, конструктор функції, що обчислює арктангенс, виглядає так:

```
» help @sym/atan.m
ATAN Symbolic inverse tangent.
»type @sym/atan.m
function Y = atan(X)
%ATAN Symbolic inverse tangent.
% Copyright (c) 1993-98 by The MathWorks, Inc.
% $Revision: 1.10 $ $Date: 1997/11/29 01:05:16 $
Y = maple('map','atan',X);
```

У разі для конструювання необхідного об'єкта використовується функція *maple*, дає вхід у ядро системи символьної математики *Maple V R4*, що у складі системи MATLAB за ліцензією фірми *MapleSoft*. Цей приклад, до речі, наочно показує, що користувач системи MATLAB може суттєво розширити кількість об'єктів класу *sym*, оскільки ядро системи *Maple V* містить набагато більше визначень, ніж у складі пакета символьної математики системи MATLAB. Для створення нових класів об'єктів є функція *class*, описана нижче.

Пакети прикладних програм системи MATLAB показують, що їх розробники з великим успіхом користуються об'єктно-орієнтованим програмуванням, часто створюючи нові класи об'єктів і нові об'єкти. Отже, М-файли системи є сукупністю наочних прикладів об'єктно-орієнтованого програмування мовою MATLAB. Це дає нам підставу обмежитися довідковим описом основних засобів такого програмування з приведенням мінімуму простих прикладів.

4.8.1. Створення класу чи об'єкта за допомогою функції class

Для створення класу об'єктів або їх ідентифікації служить функція *class*. Форми її застосування:

***class*(*OBJ*)** – повертає клас зазначеного об'єкта *OBJ*. Типи класів було зазначено вище. Додатково має клас об'єктів користувача – <*class_name*>.

***class*(*S, 'class_name'*)** – створює об'єкт класу *'class_name'* із використанням структури *S*.

***OBJ = class*(*S, 'class_name', PARENT*1*, PARENT*2*, ...*)** – створює об'єкт класу *'class_name'* на базі структури *S* і батьківських об'єктів *PARENT*1, *PARENT*2, ... При цьому створюваний об'єкт успадковують структуру та поля батьківських об'єктів. Об'єкту *OBJ* у разі властиво множинне успадкування.

Зверніть увагу на те, що ця функція зазвичай застосовується у складі М-файлів конструкторів класів об'єктів.

4.8.2. Контроль за ставленням об'єкта до заданого класу isa

Для контролю за належністю заданого об'єкта до деякого класу служить функція *isa*.

***isa*(*OBJ, 'ІМ'Я_Класу'*)** – повертає логічну 1, якщо *OBJ* належить класу із зазначеним ім'ям. Приклади застосування цієї функції:

```
» X=[1 2 3];
» isa(X,'char')
ans = 0
» isa(X,'double')
ans = 1
```

4.8.3. Інші функції об'єктно-орієнтованого програмування

Для отримання списку методів класу об'єктів використовується функція:

***M = methods*(*'class_name'*)** – повертає масив осередків із зазначенням методів, які стосуються заданого класу об'єктів. Наприклад:

```
» methods char
Methods for class char:
delete diff int
```

Наступні дві функції можуть використовуватися лише усередині конструкторів класів:

```
inferiorto('CLASS1 ','CLASS2',...)
```
та
```
superiorto(‘CLASSr,‘CLASS2',...)
```

Вони визначають нижчий та вищий пріоритети класів стосовно класу конструктора.

Будь-який оператор у системі MATLAB можна перевизначити шляхом завдання М-файлу з новим ім'ям у відповідному каталозі класів. Зокрема, всі арифметичні оператори мають уявлення у вигляді відповідних функцій.

4.9. Налагодження М-файлів у командному режимі

Навряд чи є програма з довжиною більш ніж десяток рядків, яка б запустилася відразу і видала потрібний правильний результат. Як правило, будь-яку програму треба налагоджувати в інтерактивному режимі, запускаючи її багато разів та аналізуючи отримані при кожній модифікації програми результати. Основним засобом налагодження М-файлів у системі MATLAB є вбудована підпрограма *M-File Editor/Debugger* з графічним інтерфейсом. Однак, MATLAB передбачає основні можливості налагодження і в командному режимі. Саме вони й розглянуті у цьому розділі.

Взагалі кажучи, налагодження програм – процес суто індивідуальний та творчий. Більшість користувачів середньої кваліфікації зазвичай налагоджують свої програми не звертаючись до спеціальних засобів налагодження, що вимагають для їх освоєння додаткового часу і навичок. Якщо алгоритм розв'язання завдання досить простий, після декількох модернізацій програми її вдається змусити працювати коректно.

Для цього часто достатньо ввести в програму перегляд проміжних обчислень, розблокувавши їх виведення зняттям операторів ; (точка з комою) або ввівши додаткові змінні, значення яких відображають перебіг обчислень. Після налагодження можна знову ввести оператори, що блокують висновок, і прибрати зазначені змінні.

Справжня необхідність застосування засобів налагодження виникає при відладкі серйозних програм досвідченими програмістами. Їм ці засоби, напевно, добре відомі з практики роботи з універсальними мовами програмування та їх засобами налагодження.

Для переходу в режим налагодження в командному режимі до М-файлу треба включити команду *keyboard.* Можна пустити її й у командному режимі:

```
» keyboard
K» type sw1
switch var
case {1,2,3}
disp('Перший квартал')
case {4,5,6}
disp('Другий квартал')
case {7,8,9}
disp('Третій квартал')
case {10,11,12}
disp('Четвертий квартал')
otherwise
disp('Помилка у завданні')
end
K» return
```

Ознакою переходу в режим налагодження стає поява комбінованого символу K». Він змінюється на символ після повернення командою *return* в звичайний командний режим роботи. Те саме станеться при використанні команди *dbquit*, що припиняє режим налагодження.

Один із способів налагодження М-файлів - розміщення в них точок зупинки. Однак у командному режимі не можна встановити такі точки за допомогою курсору мишки (як у *Windows*-наладчику). Тому треба мати текст програми з пронумерованими рядками. Він створюється за допомогою команди dbtype, що ілюструє такий приклад:

```
» keyboard
K» dbtype sw1
1 switch var
2 case {1,2,3}
3 disp('Перший квартал')
4 case {4,5,6}
5 disp('Другий квартал')
6 case {7,8,9}
7 disp('Третій квартал')
8 case {10,11,12}
9 disp('Четвертий квартал')
10 otherwise
```

```
11 disp('Помилка у завданні')
12 end
K»
```

Для встановлення контрольних точок у тестований М-файл використовуються такі команди:

dbstop in M-file at lineto – встановити контрольну точку в заданому рядку.

dbstop in M-file at subfun – встановити контрольну точку підфункції.

dbstop in M-file – встановити контрольну точку в М-файлі.

dbstop if error – встановити контрольну точку при повідомленні про помилку.

dbstop if warning – встановити контрольну точку під час попередження.

dbstop if NaN – встановити контрольну точку при результаті *NaN*.

dbstop if infnan – встановити контрольну точку при результаті *infnan*.

Можна використовувати спрощене введення цих команд, упускаючи слова *in*, *at* та *if*. При встановленні контрольної точки вона з'являється у вікні редактора М-файлів.

Для видалення контрольних точок використовується команда *dbclear* у тому ж синтаксисі, що і *dbstop*, наприклад:

dbclear M-file at lineto – видалити контрольну точку в заданій строчці.

Команда *dbstatus* виводить список установлених на цій сесії контрольних точок, наприклад:

```
K» dbstatus
Breakpoint for C:\MATLAB\bin\demo1.m is on line 2.
Breakpoint for C:\MATLAB\bin\sd.m is on line 3.
```

Після встановлення контрольних точок починається процес тестування М-файлу. Він полягає у виконанні одного чи кількох кроків програми з можливістю перегляду вмісту робочої області, тобто значень змінних, що змінюються під час виконання програми. Для покрокового виконання програми

використовується команда *dbstep* у таких формах:

dbstep – виконання чергового кроку;

dbstep nlines – виконання заданої кількості ліній програми

dbstep in – ця форма дозволяє користувачеві перейти до першої лінії, що виконується у функції, викликаної з М-файлу, де наступна виконувана лінія викликає іншу функцію М-файлу.

Для переходу від однієї зупинки програми до іншої використовується команда *dbcont*.

У точках зупинки користувач має можливість переглянути стан робочої області за допомогою раніше описаних команд *who* і *whos*.

Крім того, для переходу з М-файлу до робочої області з переміщеннями по робочих областях вгору або вниз використовуються команди:

dbdown – переміщення в стеку функцій, що викликаються зверху вниз, *dbup* – переміщення в стеку функцій, що викликаються знизу вгору.

Перебуваючи у робочій області, можна переглядати значення змінних, а й змінювати їх у ході налагодження програми. За допомогою команди *dbstack* можна переглядати стек функцій. Після завершення налагодження використовується команда *dbquit*.

У висновку ще раз звертаємо увагу на те, що всі можливості налагодження реалізовані в редакторі/налагоджувачі М-файлів з графічним інтерфейсом та зручними засобами візуалізації налагодження. До них відносяться виділення різним кольором різних елементів М-файлу (ключових слів, змінних, коментарів тощо), наочне уявлення контрольних точок, легкість їх встановлення та інше. Отже, описані вище прийоми налагодження в командному режимі виглядають дещо архаїчно. Очевидно, вони орієнтовані на користувачів, які звикли до командного режиму роботи з системою.

4.10. Профілювання М-файлів

Взагалі, досягнення працездатності програми лише один із етапів її налагодження. Не менш важливою є оптимізація програми за мінімумом часу її виконання або іноді мінімум обсягу оперативної пам'яті займаної програмою. Сучасні ПК, в яких використовується система MATLAB, мають достатні резерви пам'яті, тому розміри програми, як правило, не мають особливого значення. Набагато важливіша оптимізація програми за мінімумом часу виконання.

Оцінка часу виконання окремих частин програми називається її профільуванням. Для виконання цієї процедури служить команда *profile* із рядом опцій:

profile fun – запуск профілювання для функції *fun*.

profile report – виведення звіту з профілювання.

profile plot – графічне представлення результатів профілювання у вигляді діаграми Парето.

profile filename – профіль файлів із заданим ім'ям та шляхом.

profile report N – виведення звіту з профілювання заданих *N* ліній.

profile report frac – виводить звіт із профілювання тих рядків, відносна частка виконання яких у загальному часі виконання становить величину *frac* (від 0.0 до 1.0).

profile on – вмикання профілювання.

profile off – вимкнення профілювання.

profile reset – вимикання профілювання зі знищенням всіх даних.

INFO = profile – повертає структуру з наступними полями: *file* – повний шлях файлу, що профільується, *interval* – інтервали часу в секундах.

count – вектор вимірювань.

state – стан '*on*' (ввімкнений) або '*off*' (вимкнений) профілювача.

При графічному представленні профілювання горизонтальної осі вказуються

номери рядків, а вертикальної – час виконання. Спочатку з'являються рядки з великими часами виконання. Таким чином, програміст, який налагоджує програми, має можливість наочно оцінити де саме спостерігаються найбільші часи виконання.

4.11. Створення підсумкового звіту

Для створення сумарного звіту профілювання служить команда *profsumm*, яка використовується в декількох формах:

profsumm - виведення повного звіту про профільований М-файл. Виводяться дані про час виконання для рядків, сумарний час виконання яких становить 95% від загального часу або 10 рядків.

profsumm(*FRACTION*) – виводить частину звіту для рядків, відносний час виконання яких становить *FRACTION* (від 0.0 до 1.0) від загального часу виконання файлу.

profsumm(*N*) – виводиться звіт N рядків з максимальним часом виконання.

profsumm(*STR*) – виводить звіт лише за тими рядками, у яких зустрічається рядковий вираз *STR*.

profsumm(INFO), *profsumm*(*INFO, FRACTION*), *profsumm*(*INFO, N*) та *profsumm*(*INFO, STR*) – виводять підсумковий звіт за рядками, заданими масивом *INFO*.

Контрольні запитання:

1. Які типи даних існують у системі комп'ютерної математики MATLAB та яка між ними ієрархія?
2. Яка різниця між М-файл-функціями та М-файл-сценаріями, опишіть їх структуру, властивості та особливості застосування?

3. Перелічите засоби та команди для встановлення шляхів у системі комп'ютерної математики MATLAB та наведіть приклади їх застосування?

4. Перелічите функції для робота з помилками і попередженнями, що існують у системі комп'ютерної математики MATLAB та наведіть приклади їх застосування?

5. Перелічите функції підрахунку числа вхідних та вихідних аргументів, що існують у системі комп'ютерної математики MATLAB та наведіть приклади їх застосування?

6. Опишіть реалізацію умовного оператора *if* у системі комп'ютерної математики MATLAB та наведіть приклади його застосування?

7. Опишіть реалізацію циклу *for* у системі комп'ютерної математики MATLAB та наведіть приклади його застосування?

8. Опишіть реалізацію циклу *while* у системі комп'ютерної математики MATLAB та наведіть приклади його застосування?

9. Опишіть реалізацію перемикача *switch* у системі комп'ютерної математики MATLAB та наведіть приклади його застосування?

10. Поясніть особливості реалізації об'єктно-орієнтованого програмування у системі комп'ютерної математики MATLAB та наведіть приклади застосування функцій створення об'єкту чи класу?

11. Опишіть існуючі у системі комп'ютерної математики MATLAB засоби налагодження М-файлів?

12. Опишіть існуючі у системі комп'ютерної математики MATLAB засоби профілювання М-файлів?

РОЗДІЛ 5

ПОБУДУВАННЯ ГРАФІКІВ ФУНКЦІЙ У ПАКЕТІ MATLAB

Одна з головних переваг системи MATLAB - велика кількість засобів графіки, починаючи від команд побудови простих графіків функцій однієї змінної в декартовій системі координат і закінчуючи комбінованими та презентаційними графіками з елементами анімації, а також засобами проектування графічного інтерфейсу користувача *GUI*. Особливу увагу в системі приділено тривимірній графіці з функціональним забарвленням фігур і імітацією різних світлових ефектів.

5.1. Загальні можливості графіки

Основними відмінними рисами графіки стали:

- створення графіки в окремих вікнах;
- можливість виведення багатьох вікон;
- можливість переміщення вікна по екрану та зміна його розмірів;
- завдання різних координатних систем та осей;
- висока якість графіки;
- широкі можливості використання кольорів;
- легкість встановлення графічних ознак – атрибутів;
- зняття обмежень на кількість кольорів;
- велика кількість опцій у команд графіки;
- можливість отримання тривимірних фігур, що природно виглядають, та їх поєднань;
- простота побудови 3Д-графіків з їхньою проекцією на розташовану знизу площину;

- можливість побудови перерізів тривимірних фігур та поверхонь площинами;
- можливість імітації світлових ефектів під час освітлення фігур точковим джерелом світла;
- можливість створення анімаційної графіки;
- великий набір команд графіки;
- можливість створення об'єктів для типового інтерфейсу користувача.

З поняттям графіки у MATLAB пов'язане уявлення про графічні об'єкти, що мають певні властивості. Найчастіше про об'єкти можна забути, якщо ви не працюєте з об'єктно-орієнтованим програмуванням. Пов'язано це з тим, що більшість команд графіки, орієнтованої на кінцевого користувача, автоматично встановлює властивості графічних об'єктів та забезпечує відтворення графіки у потрібній системі координат, палітрі кольорів, масштабі тощо.

На нижньому рівні вирішення завдань використовується дескрипторна (описова) графіка (*Handle Graphics*). У ньому кожному графічному об'єкту ставиться у відповідність програмний модуль – описувач (дескриптор), на який можливі посилання під час використання графічного об'єкта. Дескрипторна графіка дозволяє здійснювати візуальне програмування об'єктів інтерфейсу - керуючих кнопок, текстових панелей і так далі. Ці великі можливості роблять графіку MATLAB однією з найкращих серед відомих систем комп’ютерної математики. Незважаючи на велику кількість команд такої графіки, синтаксис команд досить простий і легко засвоюється навіть початківцями.

5.2. Базові графічні об'єкти

Всі графіки, криві та поверхні, які можна побудувати у MATLAB, є комбінацією деякої кількості найпростіших відрізків багатокутників і т.д.

Комбінуючи їх належним чином, ми можемо отримати практично будь-який малюнок.

Найпростішою функцією є *line*. Вона будує відрізки та ламані лінії. Її першим аргументом є вектор *x* координат вузлів ламаної, а другим – вектор *y* координат вузлів. Для побудови просторової ламаної використовується третій аргумент вектор *z* координат вузлів. Після завдання векторів можна встановити властивості - товщину ліній, колір і т.д. у форматі

```
line(X,Y,Z,'PropertyName',PropertyValue,...)
```

Наприклад (рис. 5.1):

```
line([1 4 3 0 1],[0 3 4 1 0]);
axis equal
line([0 1 1 0 0 1 1 0 0],[0 0 1 1 0 0 1 1 0],
[0 1 2 3 4 5 6 7 8], 'Color','r','LineWidth',2)
grid on; % малює сітку на координатних площинах
t=0:pi/6:2*pi;
x=cos(t);
y=sin(t);
line(x,y);
axis equal
```

Команда *axis equal* робить позначки прирощень осей однакової довжини. Кожна команда *line* додає графічний об'єкт ламаною до поточного графічного вікна. Очищення вікна від старого малюнка може бути виконане командою – *clf*.

Можна також перед побудовою нового рисунка закрити графічне вікно кнопкою - хрестиком у правому верхньому куті вікна або командою – *close*.

Команда *rectangle* (рис. 5.2) рисує прямокутник, еліпс чи фігуру близьку їм формою. Команда `rectangle('Position',[x,y,w,h])` рисує прямокутник з вершиною у точці (*x*, *y*) шириною *w* заввишки *h*.

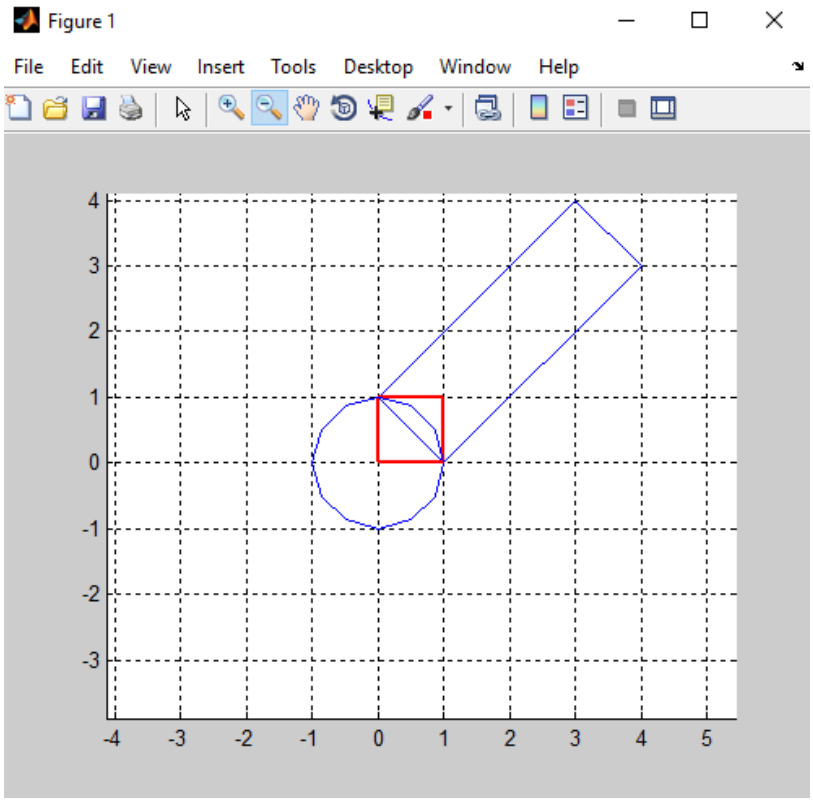


Рис. 5.1. Застосування функції *line*.

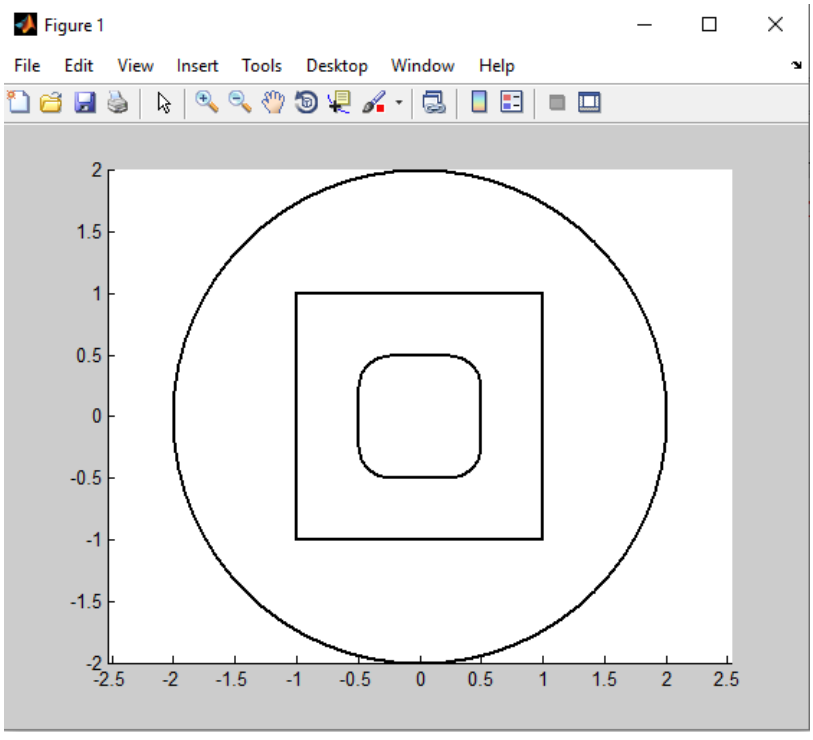


Рис. 5.2. Застосування команди *rectangle*.

```
clf
rectangle('Position',[-2 -2 4 4],'Curvature',[1,1],
'LineWidth',2);
rectangle('Position',[-1 -1 2 2],'LineWidth',2);
rectangle('Position',[-1/2 -1/2 1 1],'Curvature',
[0.5,0.5], 'LineWidth',2);
axis equal
```

Властивість *Curvature* визначає кривизну сторін (рис. 5.3).

```
rectangle('Position',[x,y,w,h],'Curvature',[cg, cv])
```

Якщо вона не задана або встановлена рівною [0 0], то відрізки сторін будуть прямі і ми отримуємо прямокутник. Завдання значень кривизни в інтервалі від 0 до 1 викривляє сторони фігури від прямолінійних (при *cg* = 0 або *cv* = 0) до еліптичних (при *cg* = 1 або *cv* = 1). Перше число в парі [*cg*, *cv*] задає кривизну горизонтальних сторін, друге – вертикальних. Слово кривизна використовується лише для позначення параметра функції і не є кривизною у точному математичному сенсі цього значення. Можна використовувати одне число, тоді воно відноситься до меншої сторони фігури.

```
clf
rectangle('Position',[-10,-5,20,10],'Curvature',[0.8,0.25],
'LineWidth',2);
rectangle('Position',[-10,-5,20,10],'Curvature',[0.8,0.5],
'LineWidth',2);
rectangle('Position',[-10,-5,20,10],'Curvature',[0.8,0.75],
'LineWidth',2);
rectangle('Position',[-7,-3,14,6],'Curvature',0.2,
'LineWidth',2);
rectangle('Position',[-7,-3,14,6],'Curvature',0.5,
'LineWidth',2);
rectangle('Position',[-7,-3,14,6],'Curvature',1,
'LineWidth',2);
rectangle('Position',[-5,-2,10,4],'Curvature',[1,1],
'FaceColor','c')
daspect([1,1,1])
xlim([-12,12])
ylim([-6,6]);
```

Номери кривих відповідають номеру рядка команди *rectangle* у коді (рис. 5.3).

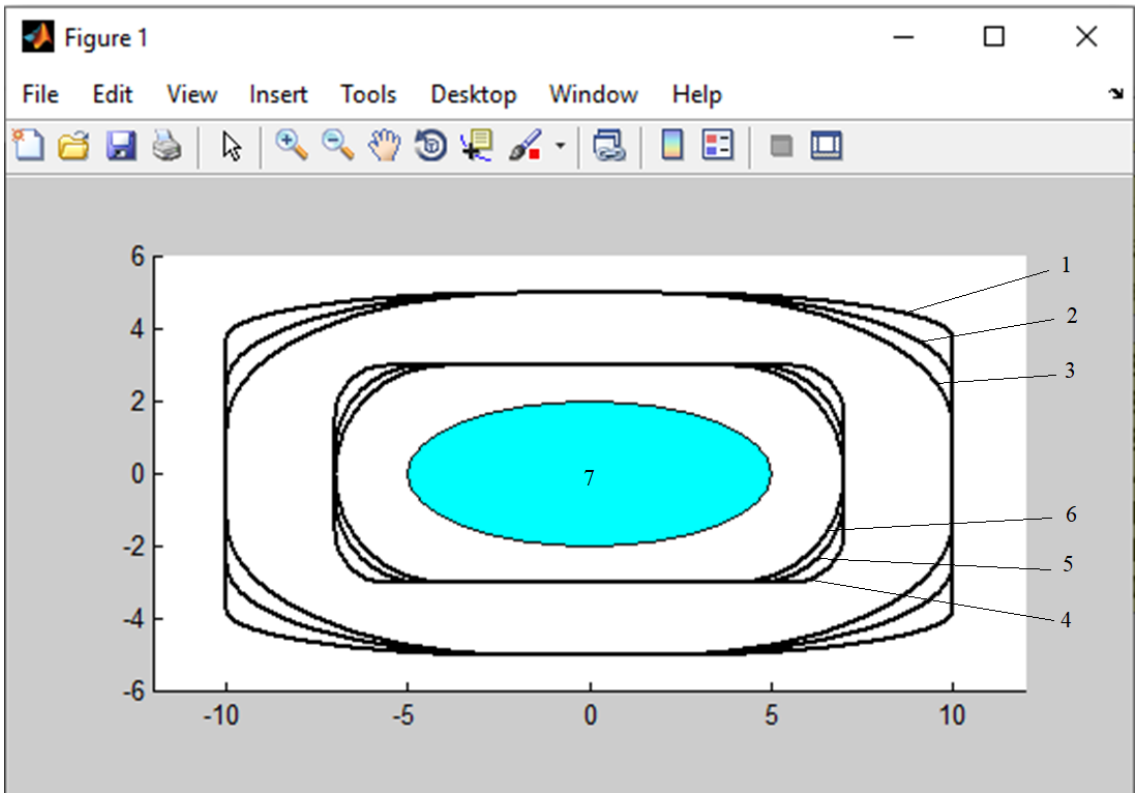


Рис. 5.3. Застосування команди *rectangle* з різними параметрами.

Команда *daspect*([1, 1, 1]) визначає однаковий масштаб одиниць по осях. А команда *daspect*([1, 3, 1]) означає, що одна одиниця по осі *x* дорівнює довжині трьом одиницям по осі *y* та одній одиниці по осі *z*, тобто по осі *y* відбувається стиск.

Опція '*FaceColor*', '*c*' призводить до побудови зафарбованої фігури кольором '*c*' – *cyan*. Команди *xlim*([-12, 12]) і *ylim*([-6, 6]) задають межі області, що відображається.

Команда *fill*(*x*, *y*) – малює зафарбований багатокутник з *x* координатами вершин, заданими у першому векторі, та *y* координатами – у другому. У цьому остання вершина має збігатися з першої. Третій аргумент задає колір заливки багатокутника

```
clf;
fill([1 2 1 0],[0 1 2 1],'g');
daspect([1 1 1])
```

Фігура показана на рис. 5.4.

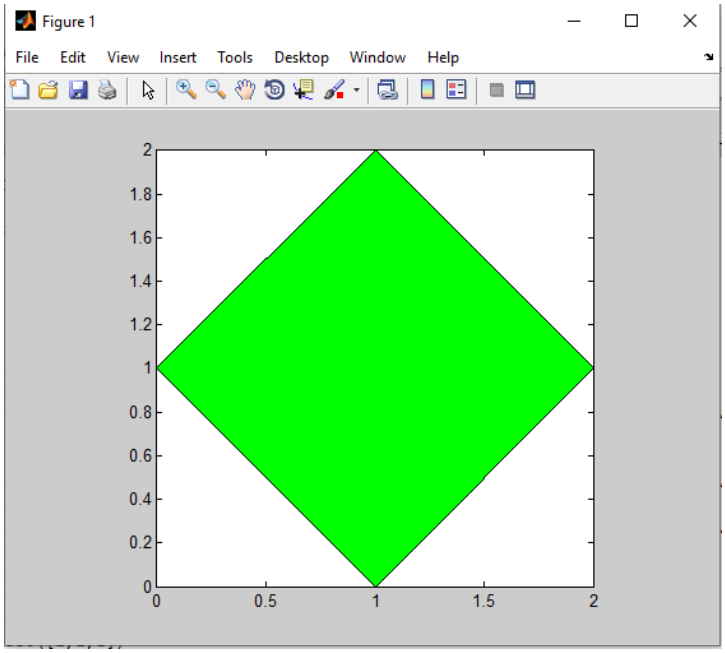


Рис. 5.4. Застосування команди *fill*.

Код, наведений далі, будує восьмикутник (рис. 5.5), у середину графічного вікна виводить текст 'Восьмикутник', а командне вікно виводить форматований текст з координатами однієї з його вершин.

```
clf
t= (1/16:1/8:1)'*2*pi;
x = sin(t);
y = cos(t);
fill(x,y,'g');
axis square
text(0,0,'Восьмикутник')
str=sprintf('Координати другої вершини: x(2)=%6.2f
y(2)=%6.2f',x(2),y(2));
disp(str);
```

Команда *text*(*x*, *y*, 'текст') у точку з координатами (*x*, *y*) виводить заданий текст.

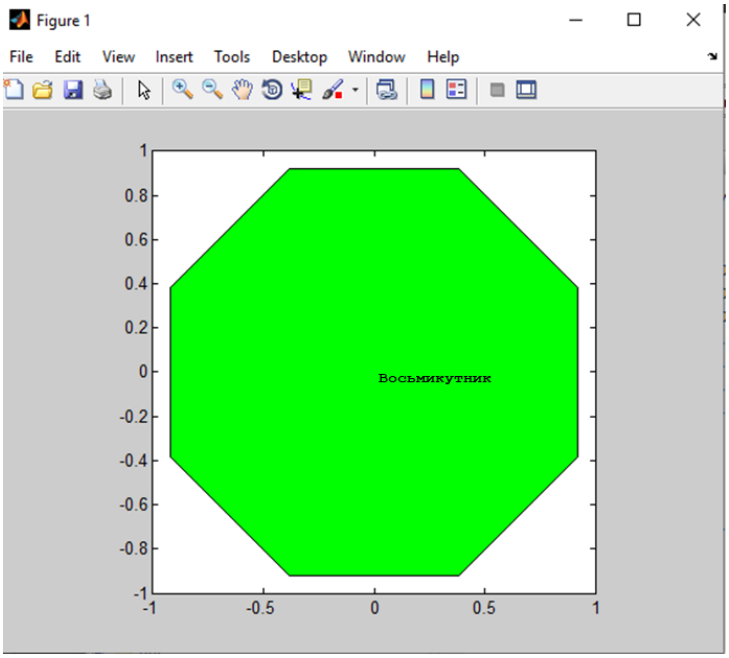


Рис. 5.5. Побудова восьмикутника.

Функція *sprintf* створює рядок, складений із тексту та відформатованих чисел. Формат задається символами % і літерою (у цьому прикладі *f*). Між відсотком та літерою можуть стояти уточнюючі символи – прапорці та числа, що визначають ширину поля виведення. Змінні, значення яких записуватимуться в рядок, є другим, третім і т.д. - аргументами функції *sprintf*. Першої послідовності символів % ... *f* відповідає другий аргумент функції *sprintf*(ім'я першої змінної), другий послідовності % ... *f* – третій аргумент (ім'я другої змінної) і т.д. У цьому прикладі значення *x*(2) та *y*(2) мають бути відображені шістьма цифрами, з яких дві мають стояти після десяткової крапки. Невикористані місця цифр заповнюються пробілами. Окрім літери *f* для відображення чисел використовується велика кількість інших символів – *g*, *d*, *e* тощо. З ними ви можете познайомитись у довідковій системі, наприклад, виконавши команду *doc sprintf*.

Команда *disp*(*str*) виводить рядок *str* у командне вікно. При формуванні текстової константи її слід укладати в одинарні лапки.

5.3. Побудова графіків функції однієї змінної

5.3.1. Графіки функцій у лінійному масштабі

MATLAB має добре розвинені графічні можливості для візуалізації даних. Розглянемо спочатку побудову найпростішого графіка функції однієї змінної з прикладу функції

$$y(x) = e^{-x}\sin(10x),$$

визначеної на відрізку [0, 1]. Виведення функції у вигляді графіка складається з наступних етапів:

1. Завдання вектору значень аргументу *x*.
2. Обчислення вектору значень функції *y*(*x*).
3. Виклик команди *plot* для побудови графіка.

Команди для завдання вектору *x* та обчислення функції краще завершувати крапкою з комою для недопущення виведення в командне вікно їх значень (після команди *plot* точку з комою ставити необов'язково, тому що вона нічого не виводить у командне вікно)

```
x = [0:0.05:1];
y = exp(-x).*sin(10*x);
plot(x, y)
```

Після виконання команд на екрані з'являється вікно *Figure*1 з графіком функції (рис. 5.6). Вікно містить меню, панель інструментів та область графіка. Надалі буде описано команди, спеціально призначені для оформлення графіка. Зараз нас цікавить сам принцип побудови графіків та деякі найпростіші можливості візуалізації функцій.

Для побудови графіка функції у робочому середовищі MATLAB повинні бути визначені два вектори однакової розмірності, наприклад, *x* і *y*. Відповідний масив *x* містить значення аргументів, а *y* - значення функції від цих аргументів. Команда *plot* з'єднує точки з координатами (*x*(*i*), *y*(*i*)) прямими лініями автоматично масштабуючи осі для оптимального розташування графіка у вікні. При побудові графіків зручно розташувати на екрані основне вікно MATLAB і вікно з графіком так, щоб вони не перекривалися.

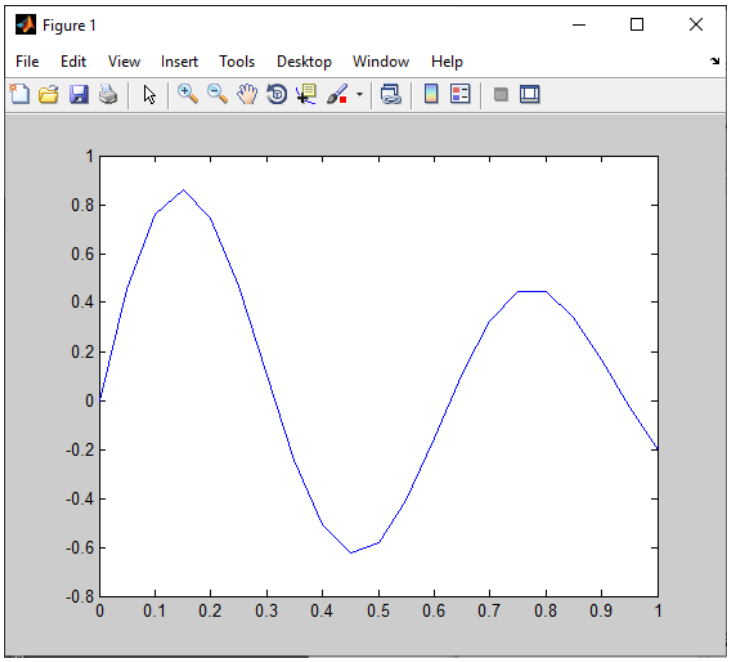


Рис. 5.6. Побудова графіку функції.

Побудований графік функції (рис. 5.6) має злами. Для точного побудови графіка функцію необхідно обчислити *y*(*x*) у більшій кількості точок на відрізку [0, 1], тобто встановити менший крок при введенні вектору *x*:

```
x = [0:0.01:4];
plot(x,sin(x)) ;
hold on;
plot(x,cos(x))
hold on
```

В результаті виходить графік функції у вигляді більш плавної кривої (рис. 5.7).

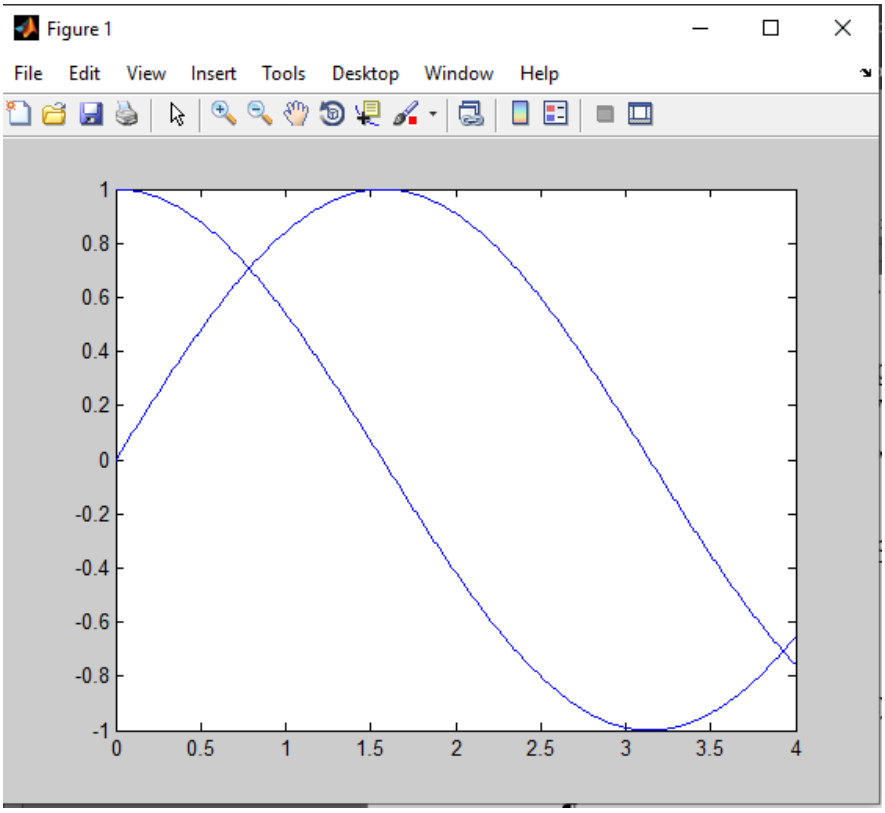


Рис. 5.7. Побудова більш плавної кривої графіку функції.

Порівняння кількох функцій зручно робити, відобразивши їх графіки на одних і тих самих осях. Наприклад, збудуємо на відрізку [–1, –0.3] графіки функцій $f(x)=\sin\left(\frac{1}{x^2}\right)$, $f(x)=\sin\left(\frac{1{,}2}{x^2}\right)$ за допомогою наступної послідовності команд (рис. 5.8):

```
x = [-1:0.005:-0.3];
f = sin(x.^-2);
g = sin(1.2*x.^-2);
plot(x, f, x, g)
```

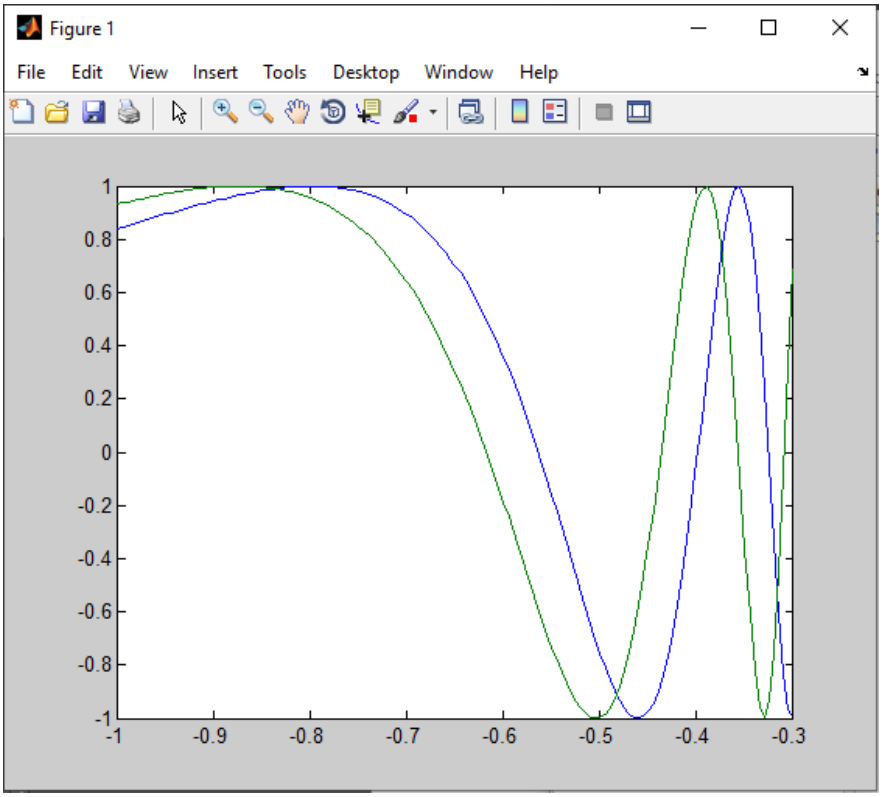


Рис. 5.8. Побудова графіків функцій на одних і тих самих осях.

Функції необов'язково повинні бути визначені на одному і тому ж самому відрізку (рис. 5.9). В цьому випадку при побудові графіків MATLAB вибирає максимальний відрізок, що містить решту. Важливо лише в кожній парі векторів абсцис та ординат вказати відповідні один одному вектори. Наприклад:

```
x1 = [-1:0.05:-0.3];
f = sin(x1.^-2);
x2 = [-1:0.05:0.3];
g = sin(1.2*x2.^-2);
plot(x1, f, x2, g)
```

Аналогічним чином вводячі в *plot* аргументами вектор абсцис і вектор ординат, можливо побудувати графіки довільного числа функцій.

Зауваження. Використання *plot* з одним аргументом вектором призводить до побудови "графіка вектору", тобто залежності значень елементів вектору від

своїх номерів. Аргументом *plot* може бути матриця, в цьому випадку на одні координатні осі виводяться графіки стовпців.

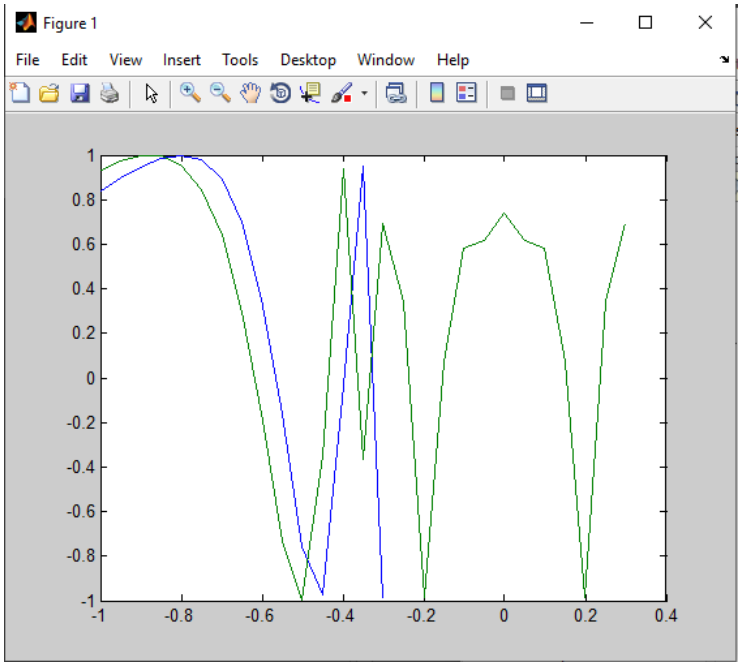


Рис. 5.9. Побудова графіків функцій на різних відрізках.

Іноді потрібно порівняти поведінку двох функцій, значення яких дуже відрізняються один від одного. Графік функції, з невеликими значеннями, практично зливається з віссю абсцис і встановити його вигляд не вдається. У цій ситуації допомагає функція *plotyy*, яка виводить графіки у вікно з двома вертикальними осями, що мають відповідний масштаб.

Порівняйте, наприклад, дві функції: $f(x) = x^{-3}$ і $F(x) = 1000 \cdot (x + 0{,}5)^{-4}$.

```
x = [0.5:0.01:3];
f = x.^-3;
F = 1000*(x+0.5).^-4;
plotyy(x, f, x, F)
```

При виконанні цього прикладу необхідно звернути увагу на те, що колір графіка збігається з кольором відповідної осі ординат (рис. 5.10).

Функція *plot* використовує лінійний масштаб по обох координатних осях. Однак MATLAB надає користувачеві можливість будувати графіки функцій однієї змінної у логарифмічному чи напівлогарифмічному масштабі.

5.3.2. Графіки функцій у логарифмічних масштабах

Для побудови графіків у логарифмічному та напівлогарифмічному масштабах служать такі функції:

- *loglog* (логарифмічний масштаб по обох осях);
- *semilogx* (логарифмічний масштаб по осі абсцис);
- *semilogy* (логарифмічний масштаб по осі ординат).

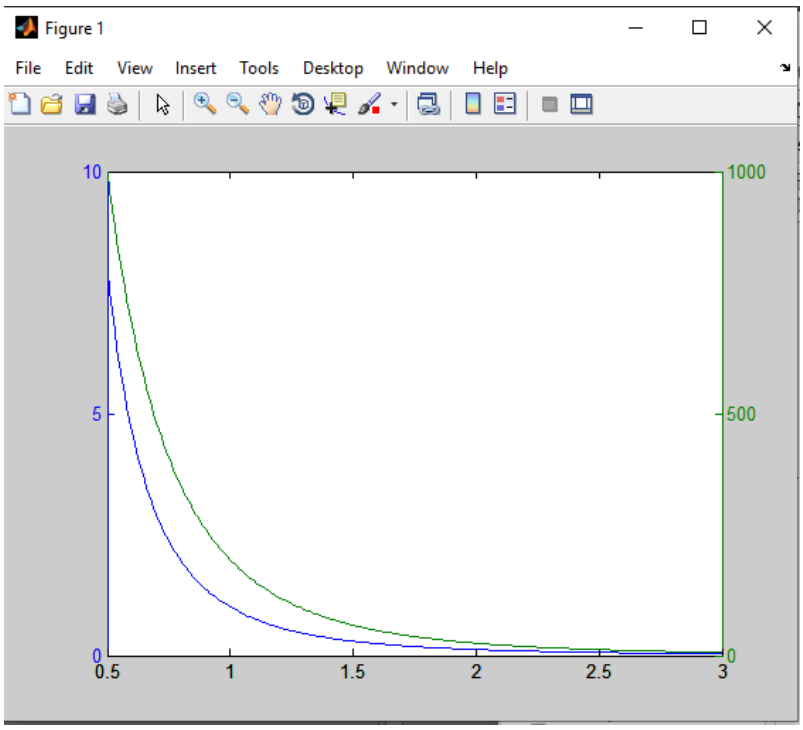


Рис. 5.10. Вивод графіків у вікно з двома різними масштабами.

Аргументи *loglog*, *semilogx* та *semilogy* задаються у вигляді пари векторів значень абсцис та ординат так само, як для функції *plot*, описаної в попередньому пункті. Побудуємо, наприклад, графіки функцій $f(x) = \ln(0{,}5x)$ та $g(x) = \sin(\ln(x))$ на відрізку [0.1, 10] в логарифмічному масштабі по осі x (рис. 5.11):

```
x = [0.1:0.01:10];
f = log(0.5*x);
g = sin(log(x));
semilogx(x, f, x ,g)
```

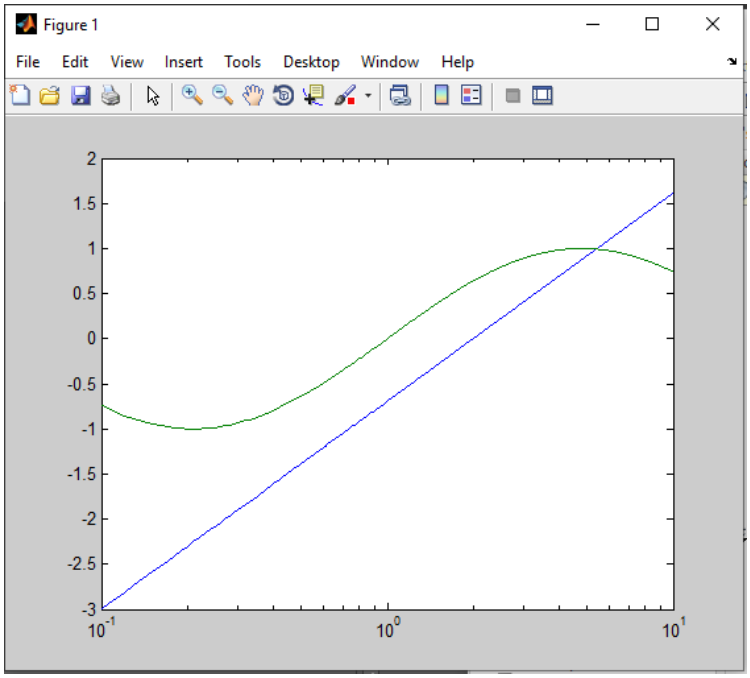


Рис. 5.11. Побудова графіків у логарифмічному масштабі.

5.3.3. Завдання властивостей ліній на графіках функцій

Побудовані графіки функцій мають бути максимально зручними для сприйняття. Часто потрібно нанести маркери, змінити колір ліній, а при підготовці до монохромного друку – встановити тип лінії (суцільна, пунктирна, штрих-пунктирна і т.д.). MATLAB надає можливість керувати видом графіків, побудованих за допомогою *plot*, *loglog*, *semilogx* і *semilogy*, для чого служить додатковий аргумент, що міститься за кожною парою векторів. Цей аргумент знаходиться між апострофів і складається з трьох символів, які визначають: колір, тип маркера та тип лінії. Використовується одна, дві чи три позиції, залежно від потрібних змін. У табл. 5.1 наведено можливі значення даного аргументу із зазначенням результату.

Таблица 5.1 – Возможные значения аргумента

Колір		Тип маркеру		Тип лінії	
y	жовтий	.	крапка	-	суцільна
m	рожевий	*o*	кружечок	:	пунктирна
c	блакитний	*x*	хрестик	-.	штрих-пунктирна
r	червоний	+	знак "плюс"	--	штрихова
g	зелений	*	зірочка		
b	синій	*s*	квадрат		
w	білий	*d*	ромб		
k	чорний	*v*	Трикутник вершиною вниз		
		^	Трикутник вершиною вгору		
		<	Трикутник вершиною вліво		
		>	трикутник вершиною вправо		
		p	п'ятикутна зірка		
		h	шестикутна зірка		

Якщо, наприклад, необхідно побудувати перший графік червоними точковими маркерами, а другий графік – чорною пунктирною лінією, слід використовувати команду *plo*t(*x, f,* '*r.*', *x, g,* '*k*:').

5.3.4. Оформлення графіків функцій

Зручність використання графіків багато в чому залежить від додаткових елементів оформлення: координатної сітки, підписів до осей, заголовка та легенди. Сітка наноситься командою *grid on*, підписи до осей розміщуються за допомогою *xlabel, ylabel*, заголовок дається командою *title*. Наявність кількох графіків на осях вимагає додавання легенди командою *legend* з інформацією про лінії. Усі перелічені команди застосовуються до графіків у лінійному, логарифмічному та напівлогарифмічному масштабах. Наступні команди виводять графіки зміни добової температури, які мають всю необхідну інформацію (рис. 5.12)

```
time=[0 4 7 9 10 11 12 13 13.5 14 14.5 15 16 17 18 20 22];
temp1=[14 15 14 16 18 17 20 22 24 28 25 20 16 13 13 14 13];
temp2=[12 13 13 14 16 18 20 20 23 25 25 20 16 12 12 11 10];
plot(time, temp1, 'ro-', time, temp2, 'go-')
grid on
title('Добові температури')
xlabel('Час (годин)')
```

```
ylabel('Температура (С)')
legend('10 травня, 11 травня')
```

При додаванні легенди слід врахувати, що порядок та кількість аргументів команди *legend* мають відповідати лініям на графіку. Останнім додатковим аргументом може бути положення легенди у вікні:

- –1 – поза графіком у правому верхньому кутку графічного вікна;
- 0 – вибирається найкраще положення в межах графіка так, щоб якнайменше перекривати самі графіки;
- 1 – у верхньому правому кутку графіка (це положення використовується за умовчанням);
- 2 – у верхньому лівому кутку графіка;
- 3 – у нижньому лівому кутку графіка;
- 4 – у нижньому правому кутку графіка.

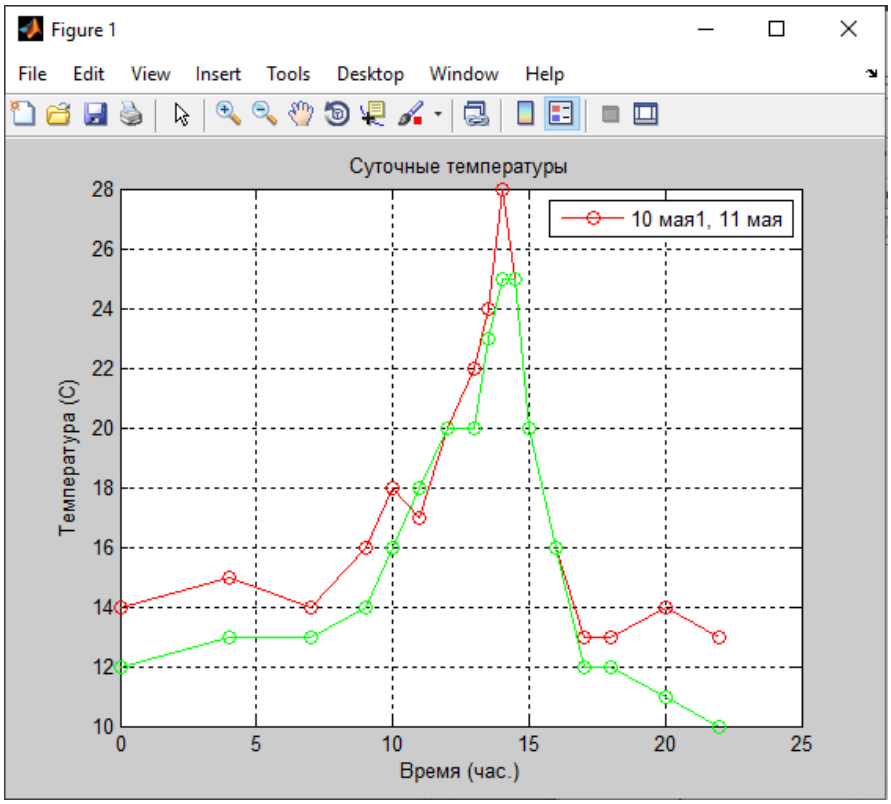


Рис. 5.12. Гафіки зміни добової температури.

У заголовку графіка, легенді та підписах осей допускається додавання формул та зміна стилів шрифту за допомогою формату *TeX*.

MATLAB виводить графіки різним кольором. Команда *plot* дозволяє легко встановити стиль і колір ліній, наприклад,

```
» plot(x,f,'k-',x,g,'k:')
```

здійснює побудову першого графіка суцільною чорною лінією, а другого – чорною пунктирною. Аргументи '*k*-' та '*k*:' задають стиль і колір першої та другої ліній. Тут *k* означає чорний колір, а дефіс або двокрапка – суцільну або пунктирну лінію.

5.3.5. Різні характеристики ліній

Функція *plot*(*X*1, *Y*1, *S*1, *X*2, *Y*2, *S*2, *X*3, *Y*3, *S*3, ...) – будує одному графіку ряд ліній, представлених даними виду (*X*',*Y*',*S*'), де *X*' і *Y*' - вектори, чи матриці, а *S*' рядок. За допомогою такої конструкції можлива побудова, наприклад, графіка функції лінією одного кольору, а точок на ній іншого кольору. Так, якщо треба побудувати графік функції лінією синього кольору з червоними точками, то спочатку треба задати побудову графіка з точками червоного кольору, а потім графіка лінії синього кольору.

За відсутності вказівки на колір ліній і точок на них він вибирається автоматично з шести кольорів (білий виключається). Якщо ліній більше шести, то вибір кольорів повторюється. Для монохромних систем лінії вирізняються стилем.

Нижче наведено приклад побудови графіків трьох функцій з різним стилем представлення кожного графіка.

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

Графіки функцій цього прикладу показані на рис. 5.13.

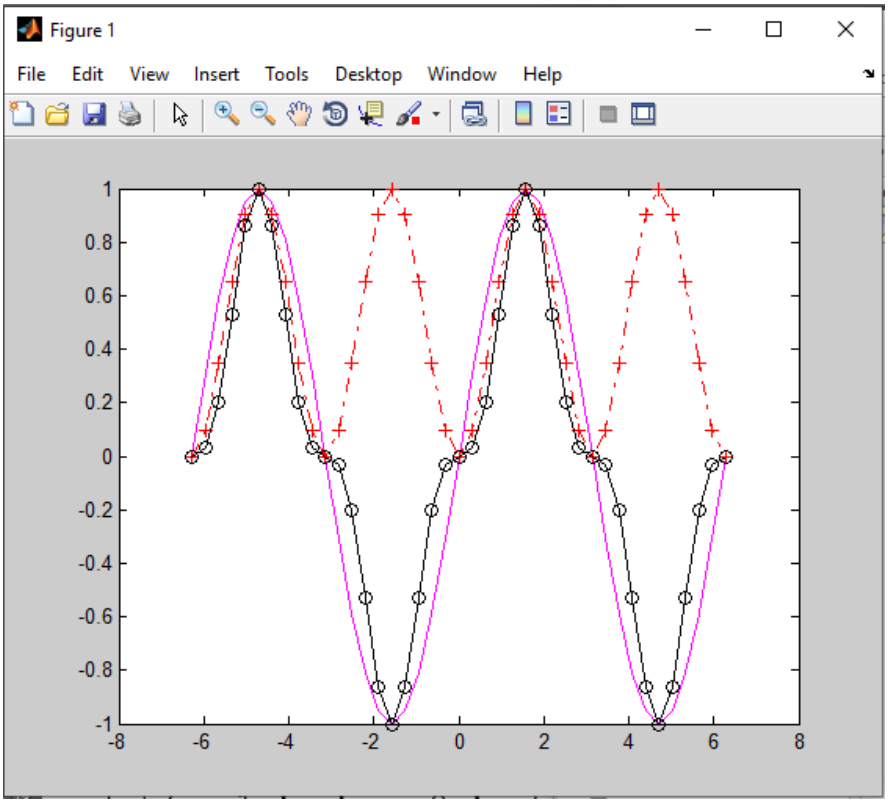


Рис. 5.13. Графікі функцій з різними стилеми.

Тут графік функції $y1$ будується суцільною фіолетовою лінією, графік $y2$ будується штрих-пунктирною лінією з точками у вигляді знака + червоного кольору, а графік $y3$ будується штриховою лінією з кружками чорного кольору.

5.4. Побудова графіків функцій двох змінних

Побудова графіка функції двох змінних у MATLAB на прямокутній області визначення змінних включає два попередні етапи:

1. Розбиття області визначення прямокутною сіткою.

2. Обчислення значень функції у точках перетину ліній сітки та запис їх у матрицю.

Побудуємо графік функції $z(x, y) = x^2 + y^2$ на області визначення у вигляді квадрата $x \in [0, 1]$, $y \in [0, 1]$. Необхідно розбити квадрат рівномірною сіткою (наприклад, з кроком 0.2) та обчислити значення функцій у вузлах, позначених точками.

Зручно використовувати два двомірні масиви *x* і *y*, розмірністю шість на шість, для зберігання інформації про координати вузлів. Масив *x* складається з однакових рядків, у яких записані координати *x*1, *x*2, ..., *x*6, а масив *y* містить однакові стовпці з *y*1, *y*2, ..., *y*6. Значення функції у вузлах сітки запишемо в масив *z* такої ж розмірності (6×6), причому для обчислення матриці *z* використовуємо вираз функції, але з поелементними матричними операціями. Тоді, наприклад, *z*(3,4) якраз дорівнюватиме значення функції *z*(*x*, *y*) у точці (*x*3, *y*4). Для генерації масивів сітки *x* і *y* за координатами вузлів у MATLAB передбачено функцію *meshgrid*, для побудови графіка у вигляді каркасної поверхні – функцію *mesh*. Наступні оператори призводять до появи на екрані вікна з графіком функції (рис. 5.14). Крапка з комою наприкінці операторів не ставиться, щоб проконтролювати генерацію масивів:

```
» [X, Y] = meshgrid(0:0.2:1,0:0.2:1)
X =
     0 0.2000 0.4000 0.6000 0.8000 1.0000
     0 0.2000 0.4000 0.6000 0.8000 1.0000
     0 0.2000 0.4000 0.6000 0.8000 1.0000
     0 0.2000 0.4000 0.6000 0.8000 1.0000
     0 0.2000 0.4000 0.6000 0.8000 1.0000
     0 0.2000 0.4000 0.6000 0.8000 1.0000
Y =
     0       0       0       0       0       0
     0.2000 0.2000 0.2000 0.2000 0.2000 0.2000
     0.4000 0.4000 0.4000 0.4000 0.4000 0.4000
     0.6000 0.6000 0.6000 0.6000 0.6000 0.6000
     0.8000 0.8000 0.8000 0.8000 0.8000 0.8000
     1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
Z = X.^2+Y.^2
```

```
Z =
     0       0.0400  0.1600  0.3600  0.6400  1.0000
     0.0400  0.0800  0.2000  0.4000  0.6800  1.0400
     0.1600  0.2000  0.3200  0.5200  0.8000  1.1600
     0.3600  0.4000  0.5200  0.7200  1.0000  1.3600
     0.6400  0.6800  0.8000  1.0000  1.2800  1.6400
     1.0000  1.0400  1.1600  1.3600  1.6400  2.0000
mesh(X,Y,Z)
```

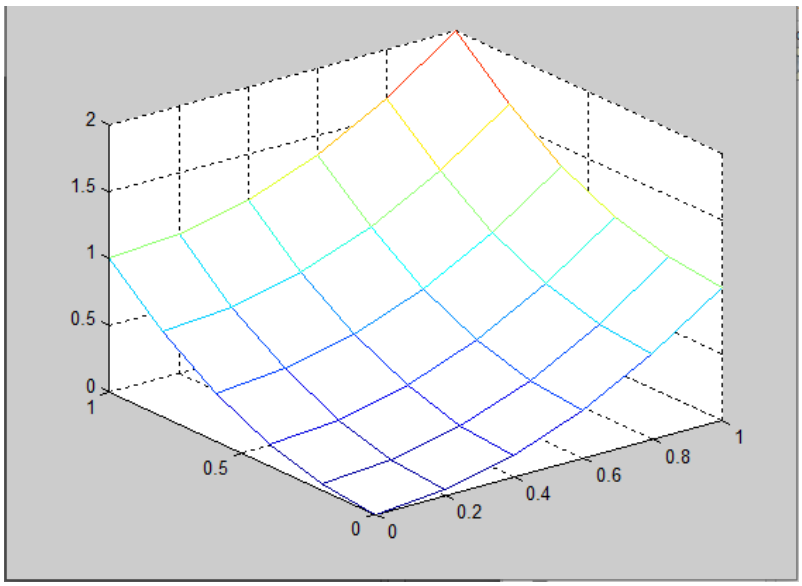


Рис. 5.14. Побудова графіку функції двох змінних.

5.5. Побудова тривимірних графіків за допомогою функцій сімейства *ez*

Поверхню сфери (рис. 5.15) за її параметричним рівнянням можна побудувати за допомогою наступних команд:

```
ezsurfc('cos(v)*cos(u)','cos(v)*sin(u)','sin(v)',
[-pi/2,pi/2,0,2*pi],21);
colormap(gray(128))
```

Використання функції *ezsurf* побудує ту саму поверхню але без контурних ліній на площині *XY* (рис. 5.16):

```
ezsurf('cos(v)*cos(u)','cos(v)*sin(u)','sin(v)',
[-pi/2,pi/2,0,2*pi],21)
```

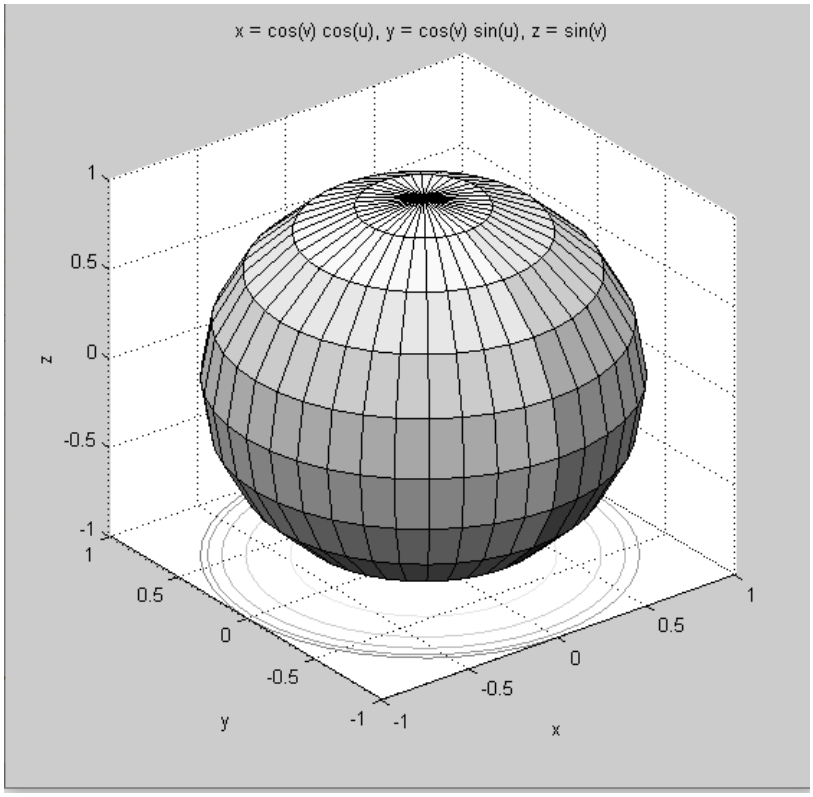


Рис. 5.15. Побудова поверхні сфери.

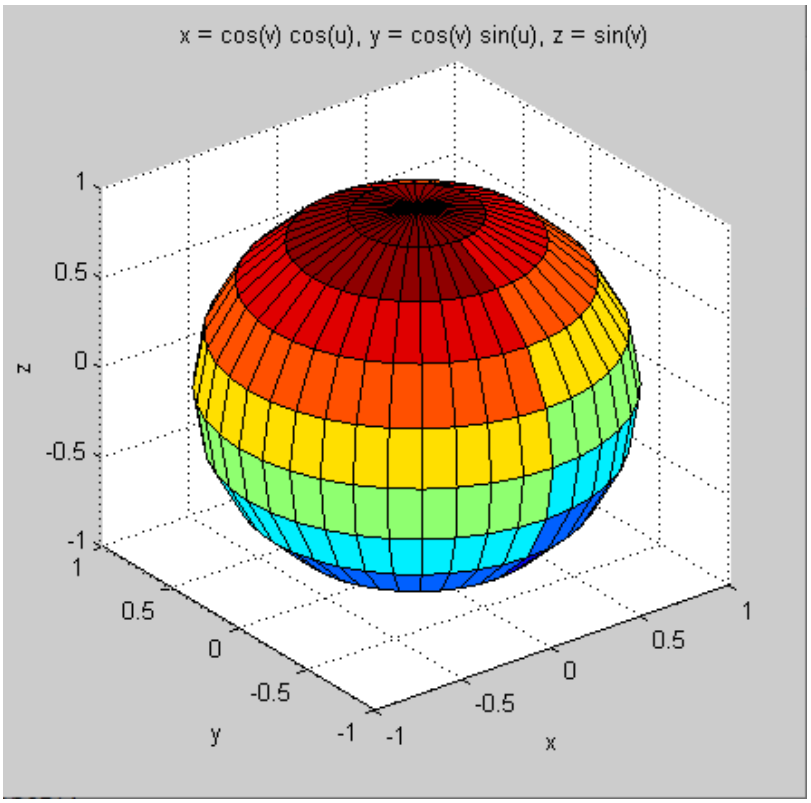


Рис. 5.16. Побудова поверхні сфери без контурних ліній.

Каркасну поверхню сфери (рис. 5.17) з віддаленими невидимими лініями можна отримати за допомогою функції *ezmesh* наступним чином:

```
ezmesh('cos(v)*cos(u)','cos(v)*sin(u)','sin(v)',
[-pi/2,pi/2,0,2*pi],21)
colormap([0 0 0])
axis equal
```

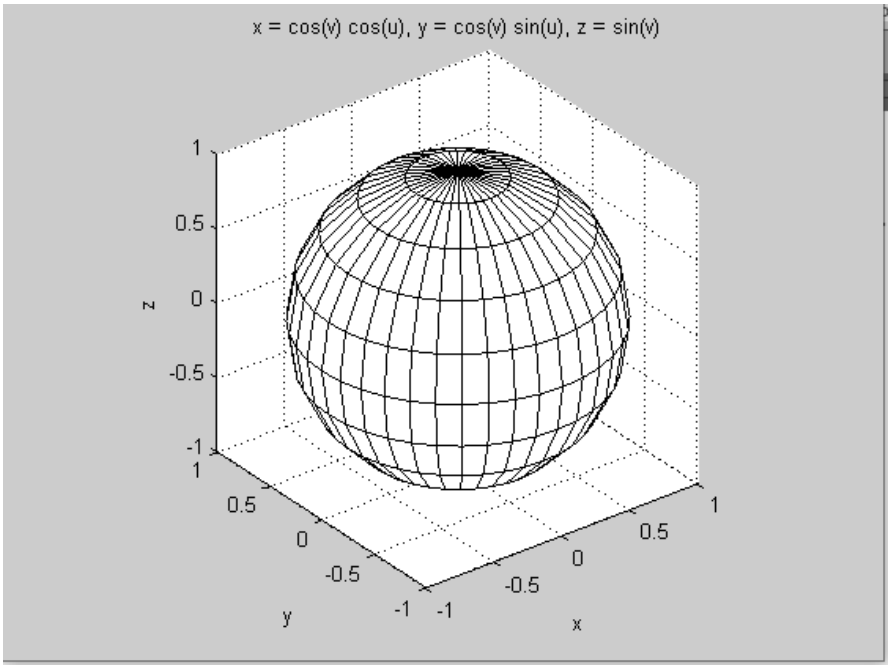


Рис. 5.17. Каркасна поверхня сфери.

Можливо будувати навіть розривні поверхні (рис. 5.18):

```
ezsurfc('(heaviside(x+1)-heaviside(x-1))*(heaviside(y+1)-
heaviside(y-1))',[-2,2],51)
ezmesh('(heaviside(x+1)-heaviside(x-1))*(heaviside(y+1)-
heaviside(y-1))',[-2,2],51)
colormap([0 0 0])
```

Можливо побудувати рівну область як поверхню із краєм у формі, наприклад, кола (рис. 5.19):

```
ezmesh('u','(abs(sqrt(1-u^2)+v)-abs(sqrt(1-u^2)-
v))/2','0',[-1,1,-1,1],21)
```

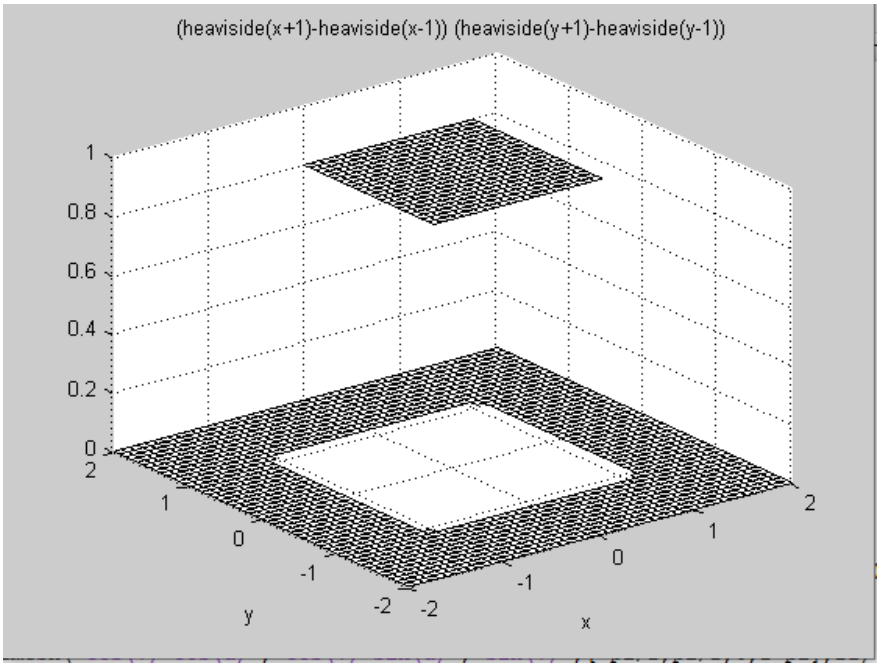


Рис. 5.18. Побудова розірваної поверхні.

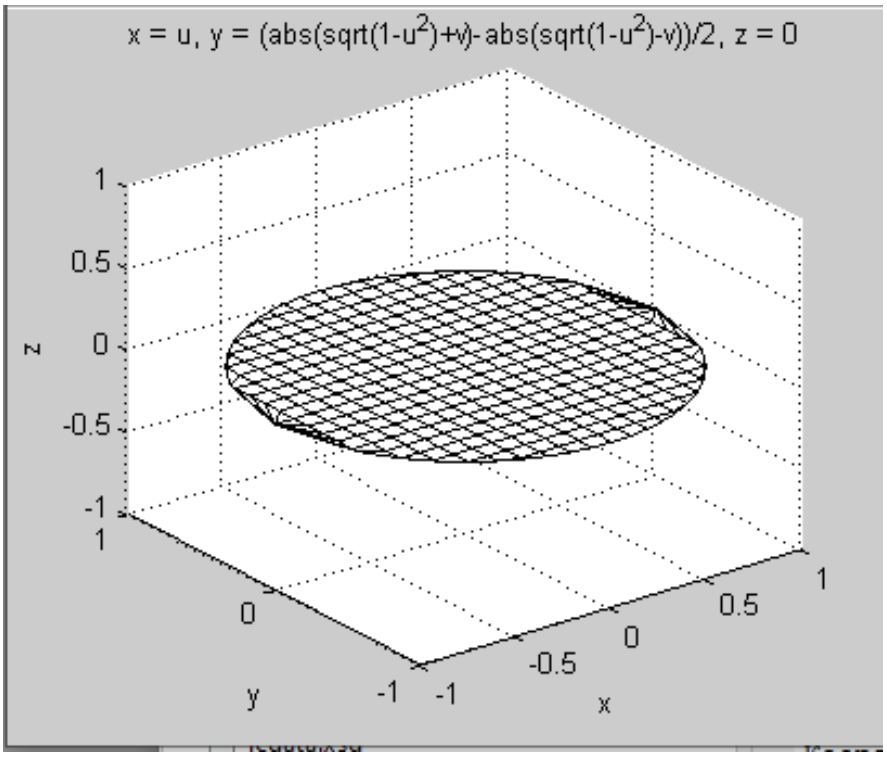


Рис. 5.19. Побудова поверхні із краєм у формі кола.

або таким чином (рис. 5.20)

```
ezsurf('u','(abs(sqrt(1-u^2)+v)-abs(sqrt(1-u^2)-
v))/2','0',[-1,1,-1,1],21)
```

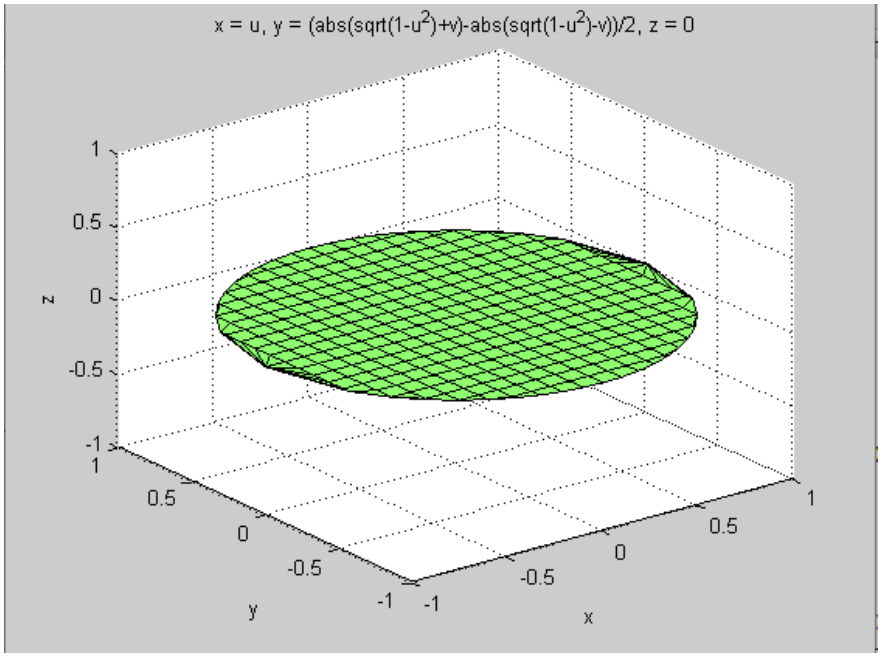


Рис. 5.20. Побудова поверхні із краєм у формі кола.

або з вогнутою поверхнею (рис. 5.21):

```
ezmesh('u','(abs(sqrt(1-u^2)+v)-abs(sqrt(1-u^2)-v))/2',
'u^2+((abs(sqrt(1-u^2)+v)-abs(sqrt(1-u^2)-v))/2)^2',
[-1,1,-1,1],21)
```

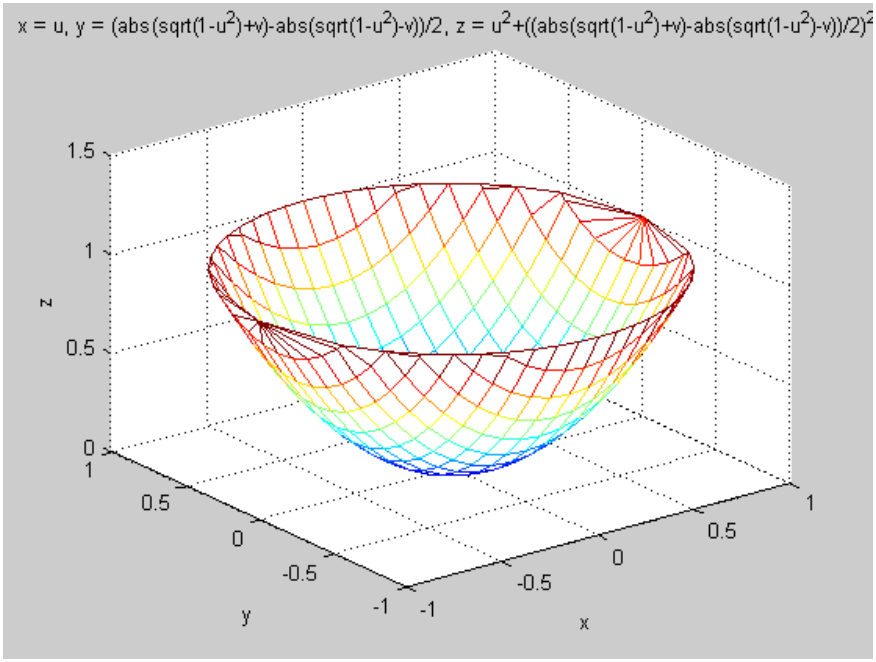


Рис. 5.21. Побудова вогнутої поверхні із краєм у формі кола.

Якщо областю зміни аргументів функції є коло, то простіше використовувати опцію цих функцій '*circ*' (рис. 5.22):

```
ezmesh('x*y','circ')
```

Функції групи ***ez…*** можна використовувати і зі стандартними файлами – функціями MATLAB, але при цьому у формулах слід використовувати знаки поелементних арифметичних операцій (з попередньою точкою). Наприклад створимо наступну М–функцію:

```
function x=xellipse(u,v)
x=2*cos(v).*cos(u);
```

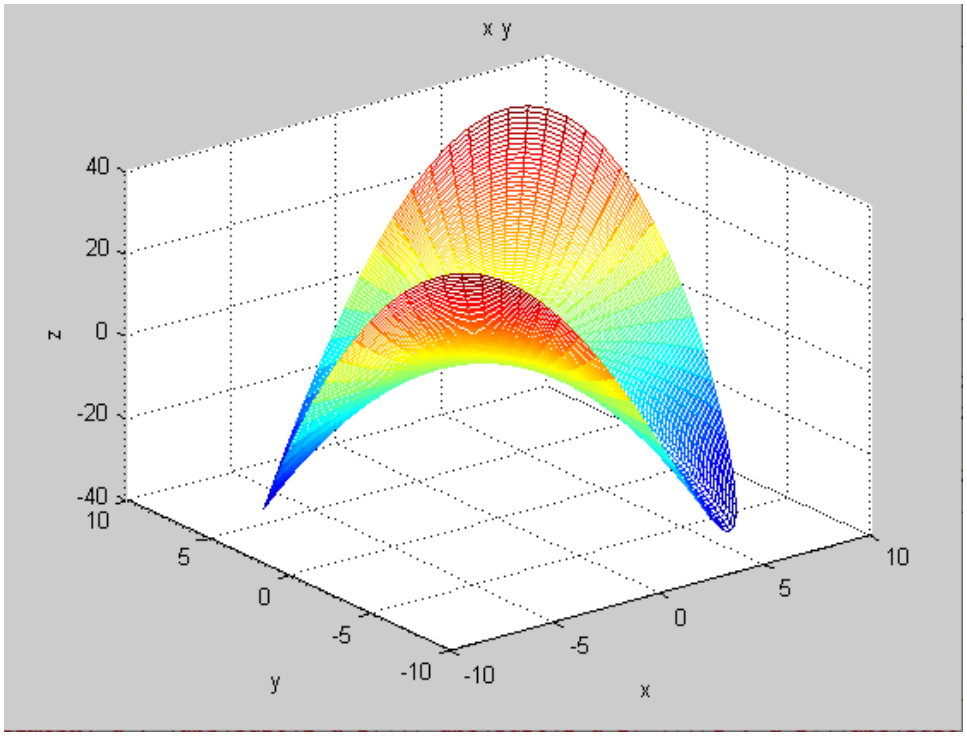


Рис. 5.22. Побудова поверхні із опцію '*circ*'.

Її графік можна побудувати в такий спосіб (рис. 5.23):

```
ezsurf('xellipse',[0 2*pi -pi/2 pi/2])
```

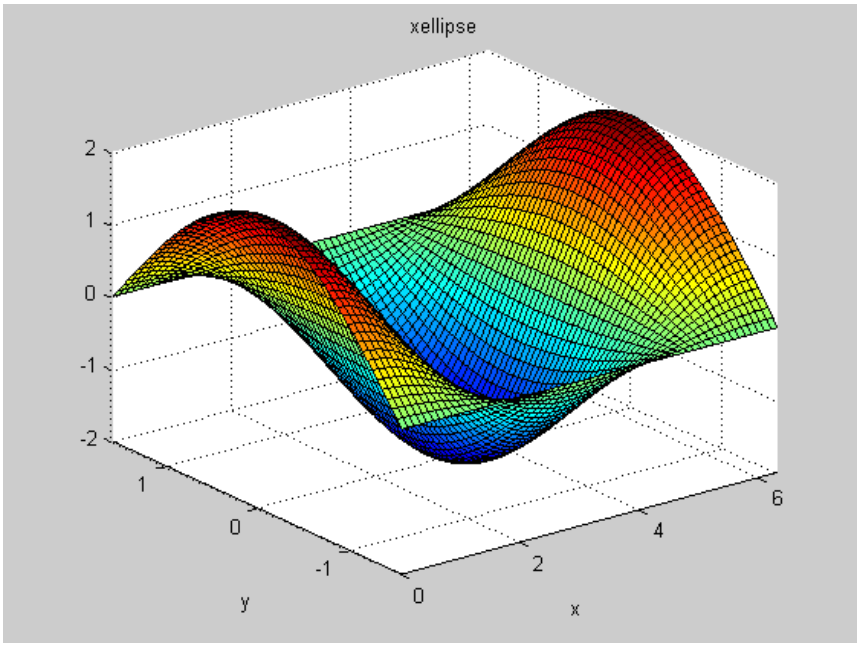


Рис. 5.23. Побудова поверхні використовуючи стандартні файли.

Ця функція має один недолік – замість числа 2 хотілося б мати аргумент, якому можна було б надавати довільні значення. Але використання функції з трьома аргументами під час виклику функції *ezsurf* призведе до помилки. Змінимо тіло функції наступним чином:

```
function x=xel(u,v,a)
x=a.*cos(v).*cos(u);
```

Щоб побудувати графік функції *xel*, треба створити локально функцію двох змінних @(*u*, *v*)*xel*(*u*, *v*, 2) і використовувати її при виклику графічної функції:

```
ezsurf(@(u,v)xel(u,v,2), [0 2*pi -pi/2 pi/2])
```

Створимо ще дві аналогічні функції:

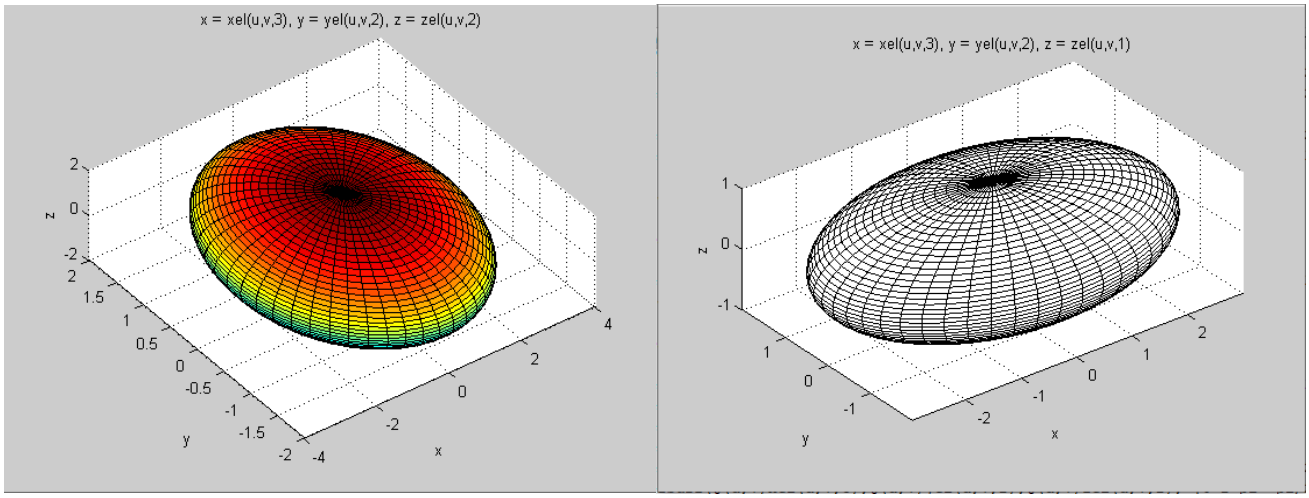
```
function y=yel(u,v,b)
y=b.*cos(v).*sin(u);
```
та
```
function z=zel(u,v,c)
z=c.*sin(v);
```

Тоді можна побудувати поверхню еліпсоїда за його параметричним рівнянням, підставляючи довільні значення півосей (рис. 5.24, *а*):

```
ezsurf(@(u,v)xel(u,v,3),@(u,v)yel(u,v,2),@(u,v)zel(u,v,2),
[0 2*pi -pi/2 pi/2])
```

або каркасну поверхню еліпса (рис. 5.24, *б*):

```
ezmesh(@(u,v)xel(u,v,3),@(u,v)yel(u,v,2),@(u,v)zel(u,v,1),
[0 2*pi -pi/2 pi/2])
axis equal
colormap([0 0 0])
```



а *б*

Рис. 5.24. Побудова поверхні еліпсоїда.

Функції, створені за допомогою дескриптора @, можна надавати ідентифікаторам та використовувати їх імена під час виклику графічних функцій (рис. 5.25):

```
x=@(u,v)xel(u,v,3) ;
y=@(u,v)yel(u,v,2);
z=@(u,v)zel(u,v,1);
ezmesh(x,y,z, [0 2*pi -pi/2 pi/2],21)
```

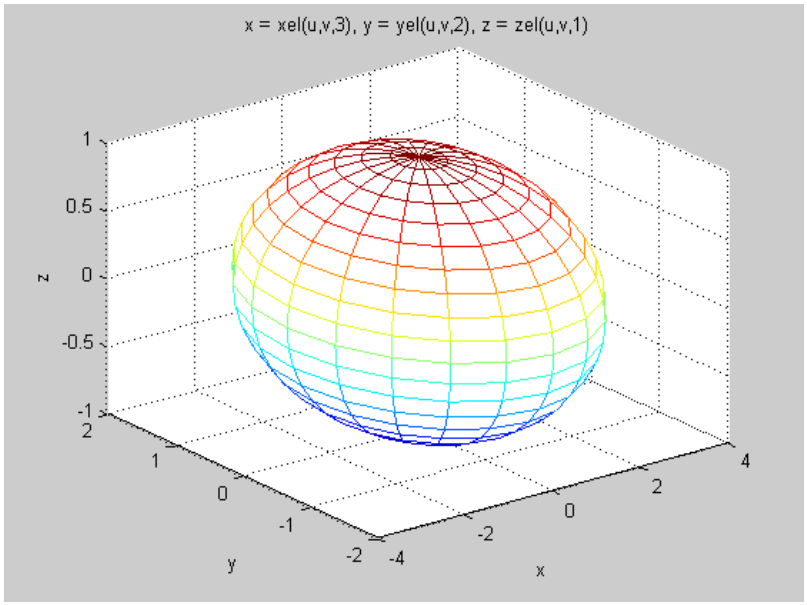


Рис. 5.25. Приклад побудови поверхні за допомогою дескриптора @.

Функції можна створювати відразу за місцем, без створення М-файлів та будувати графіки поверхонь (рис. 5.26)

```
x=@(u,v) u ;
y=@(u,v) (abs(sqrt(1-u.^2)+v)-abs(sqrt(1-u.^2)-v))/2;
z=@(u,v) u.^2+v.^2;
ezmesh(x,y,z,[-1 1],21)
```

або, використовуючи побудовані функції $x(u, v)$, $y(u, v)$,

```
ezmesh(x,y,'0',[-1 1],21)
```

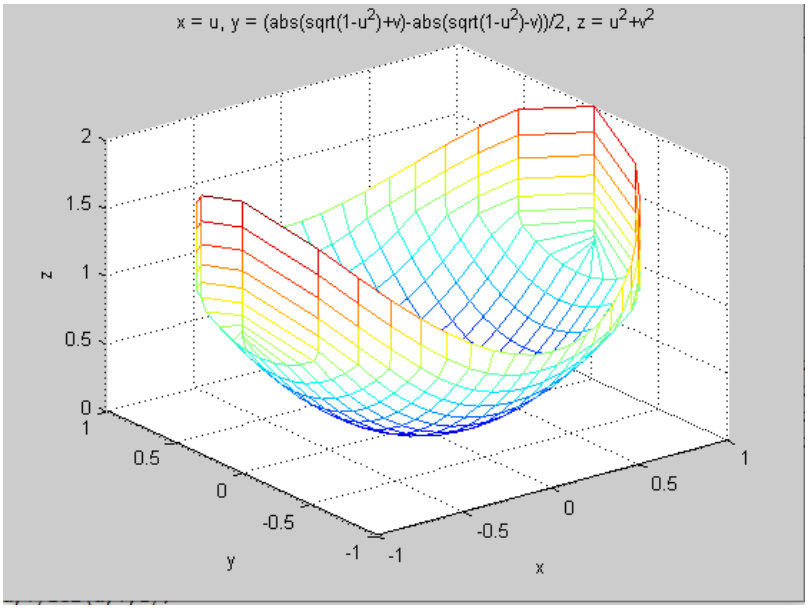


Рис. 5.26. Приклад побудови поверхні без створення М-файлу.

5.6. Розбиття графічного вікна

Побудуємо паралелепіпед за його параметричними рівняннями. Щоб поруч побудувати кілька графіків використовується функція *subplot*(*m, n, k*), де *m* – кількість рядів графіків, *n* – кількість стовпців, *k* – номер поточного графіка (той у який наступна графічна функція виводитиме результат). Графіки нумеруються по рядкам. Результат усіх побудов ми зібрали у наступний файл сценарій:

```
PR=@(x,a,w) (w+abs(x-a)-abs(x-a-w))/2;
subplot(2,2,1);ezplot(@(u)PR(u,0,1),[-1 3]);
xg=@(u) 1-PR(u,0,2)+PR(u,2,2);
yg=@(u) PR(u,0,1)-PR(u,1,2)+PR(u,3,1);
subplot(2,2,2);ezplot(xg,yg,[0 4]); axis equal;
xv=@(v) PR(v,0,1)-PR(v,2,1);
zv=@(v) PR(v,1,1);
subplot(2,2,3); ezplot(xv,zv,[0 3]); axis equal;
xs=@(u,v) xv(v).*xg(u);
ys=@(u,v) xv(v).*yg(u);
zs=@(u,v) zv(v);
subplot(2,3,6);
ezmesh(xs,ys,zs,[0 4 0 3],21);
colormap([0 0 0]);
axis equal;
```

Цей код необхідно набирати та виконувати по кілька рядків. Спочатку введіть та виконайте перші два рядки – отримайте графік функції *PR*(*u*,0,1), потім ще три рядки – отримайте квадрат, побудований за його параметричним рівнянням $x = xg(u)$, $y = yg(u)$.

Зауважимо, що корисно побудувати окремо графік кожної з координатних функцій *xg*(*u*) та *yg*(*u*). Наступні три рядки вводять та будують ламану у формі скоби. Останні чотири рядки вводять координатні функції параметричного рівняння поверхні паралелепіпеда і за ними функція *ezmesh* будує каркас поверхні. Після налагодження рядків коду скопіюйте їх у редактор М-файлів і збережіть під

ім'ям *parallelepiped.m*. Потім виконайте команду *parallelepiped*. В результаті виконання з’явиться графічне вікно (рис. 5.27) з чотирма різними графіками.

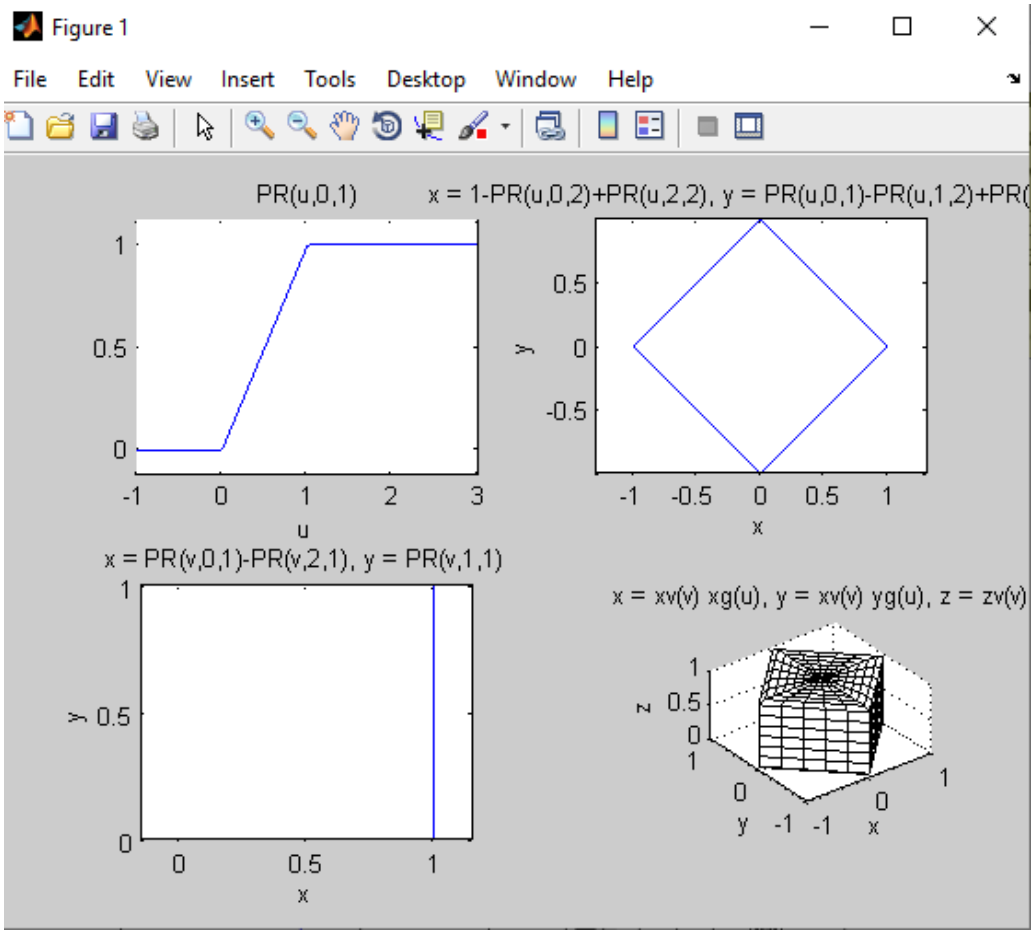


Рис. 5.27. Приклад побудови вікна з чотирма різними графіками.

Розглянемо ще один приклад разбиття графічного вікна (рис. 5.28):

```
x=-15:0.1:15;
subplot(2,2,1)
plot(x,sin(x))
subplot(2,2,2)
plot(sin(5*x),cos(2*x+0.2))
subplot(2,2,3)
plot(x,cos(x).^2)
subplot(2,2,4)
plot(x,sin(x)./x)
```

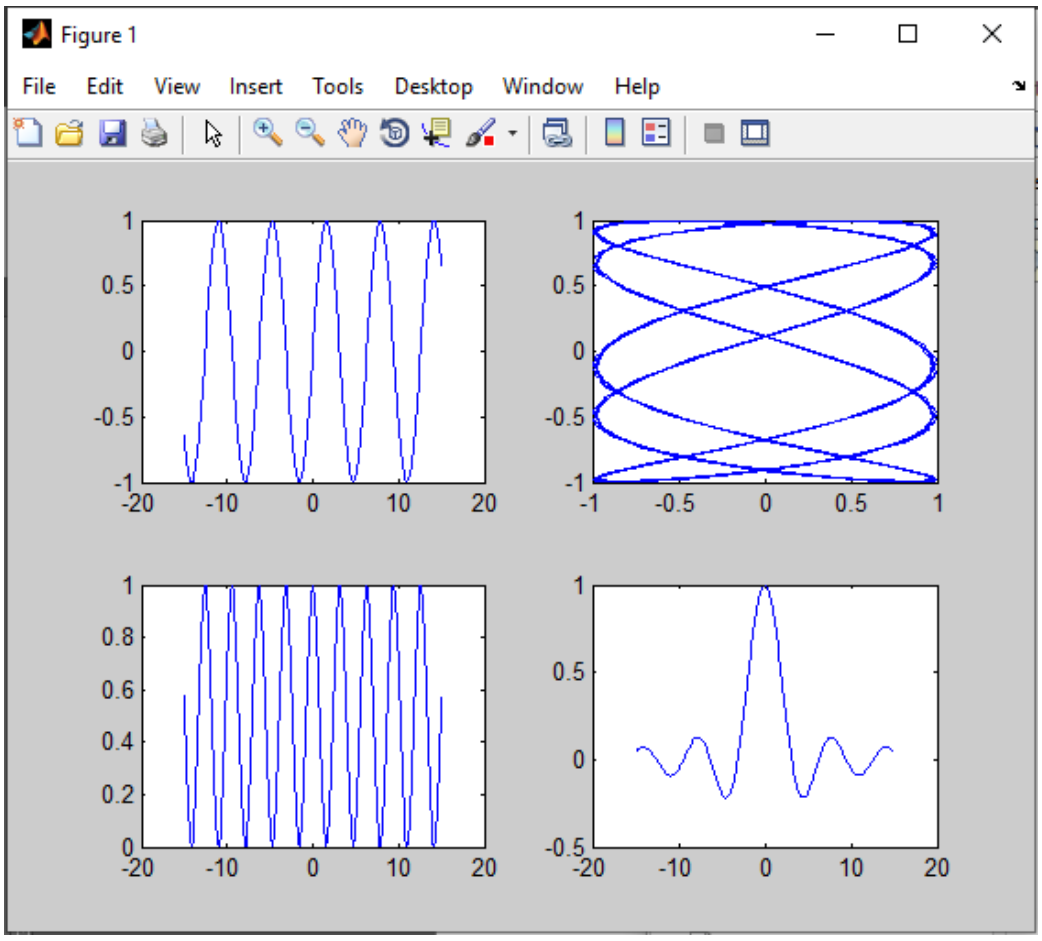


Рис. 5.28. Приклад побудови вікна з чотирма різними графіками.

Використання функції двох змінних *f*(*x*, *y*) при виклику функції *ezplot* дозволяє побудувати криву *f*(*x*, *y*)=0 за її неявним рівнянням

```
w=@(x,y) 1-x.^2-y.^2;
ezplot(w,[-1 1 -1 1]);
axis equal
```

У наступному прикладі будується квадрат за його неявним рівнянням

```
w(x,y)=0
w=@(x,y) 2-x.^2-y.^2-sqrt((1-x.^2).^2+(1-y.^2).^2);
ezplot(w,[-1.2,1.2])
```

Заодно, подивіться як виглядає поверхня функції *z* = *w*(*x*,*y*) (рис. 5.29)

```
ezmesh(w,[-1.2 1.2 -1.2 1.2])
colormap([0 0 0])
```

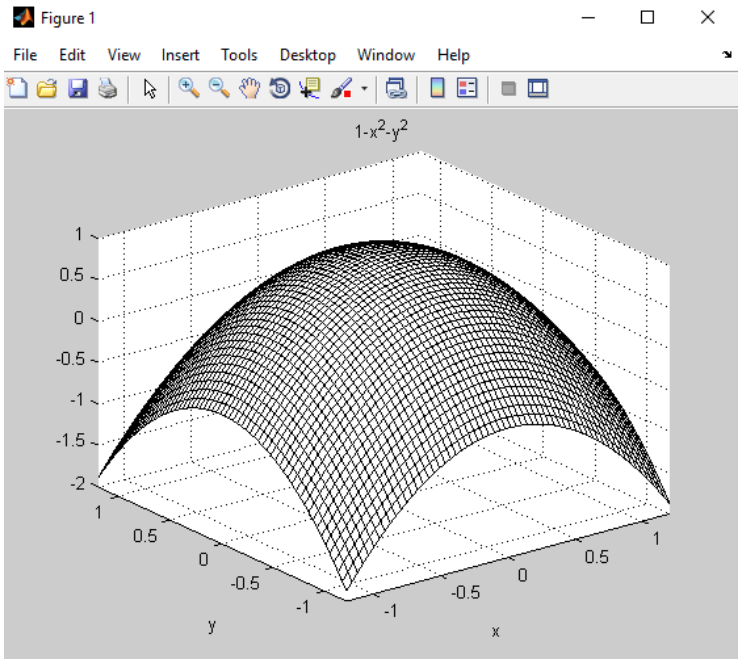


Рис. 5.29. Вигляд поверхні функції $z = w(x,y)$.

5.7. Додаткові графічні команди

Команда *newplot* створює порожнє графічне вікно. Команда *view* дозволяє переглянути тривимірний рисунок з іншого напрямку. Наприклад:

```
view(2) % вид зі стандартного напрямку 2
view(3) % вид зі стандартного напрямку 3
```

Команда *view*(*Azimuth*,*Elevation*) визначає вид з напрямку, що задається кутом *Azimuth*. Кут відлічується навколо осі *z* і кутом *Elevation*, який відлічується від площини *XY* (рис. 5.30). При цьому кути задаються у градусах. Наприклад *view*(60, –45).

Команда *colorbar* виводить поруч із рисунком стовпчик палітри кольорів.

Команда *shading interp* згладжує кольори на тривимірному графіку.

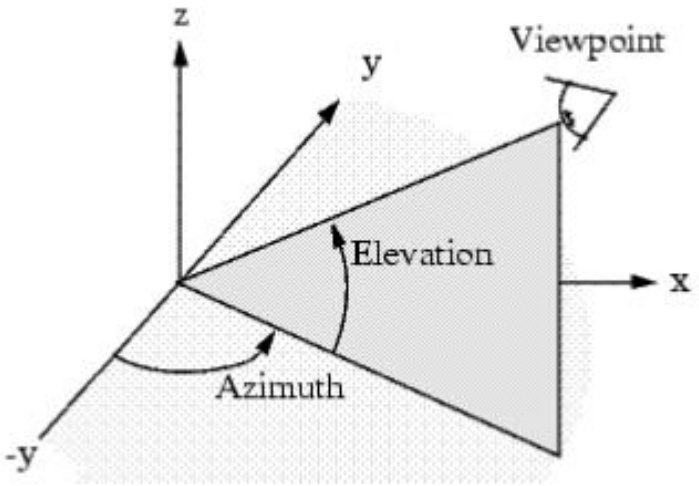
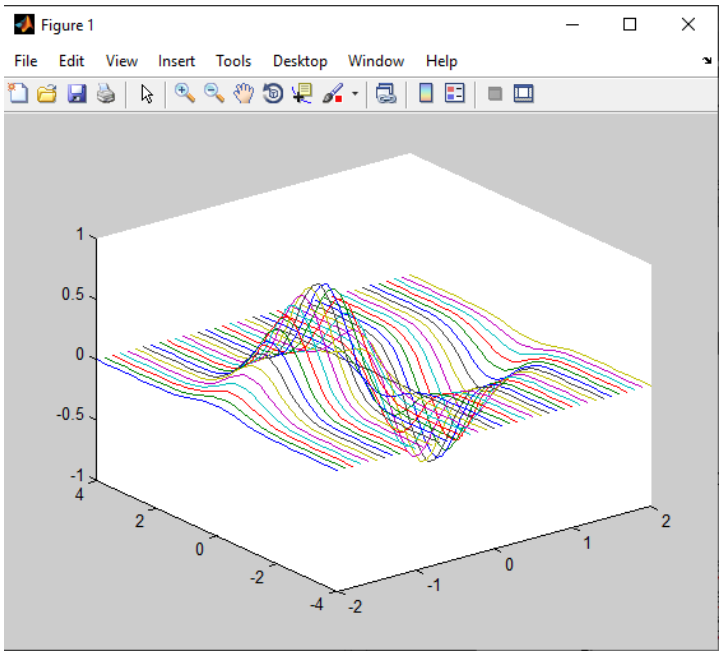


Рис. 5.30. Перегляд тривимірний рисунок з інших напрямків.

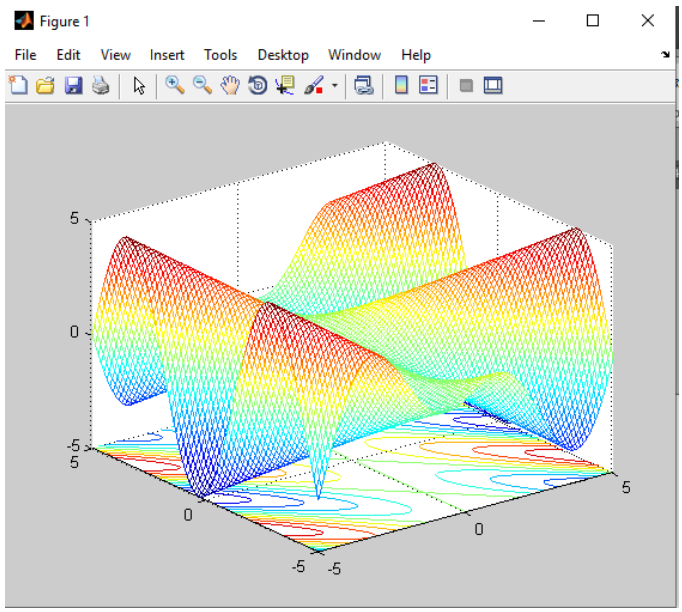
Команда *box on* відображає прямокутник або прямий паралелепіпед навколо графічної області. Команда *box off* стирає охоплюючий прямокутник або паралелепіпед. Зазвичай команду *box* корисно використовувати, коли будуються тривимірні фігури.

5.8. Приклади побудови поверхонь

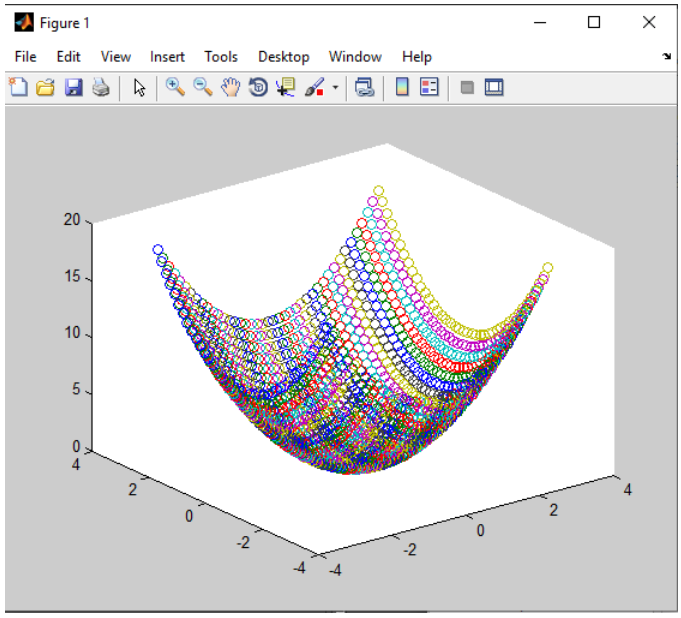
```
x=-2:0.1:2;
y=-4:0.2:4;
[X,Y]=meshgrid(x,y);
z=-2*X.*exp(-X.^2-Y.^2);
plot3(X,Y,z)
```



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

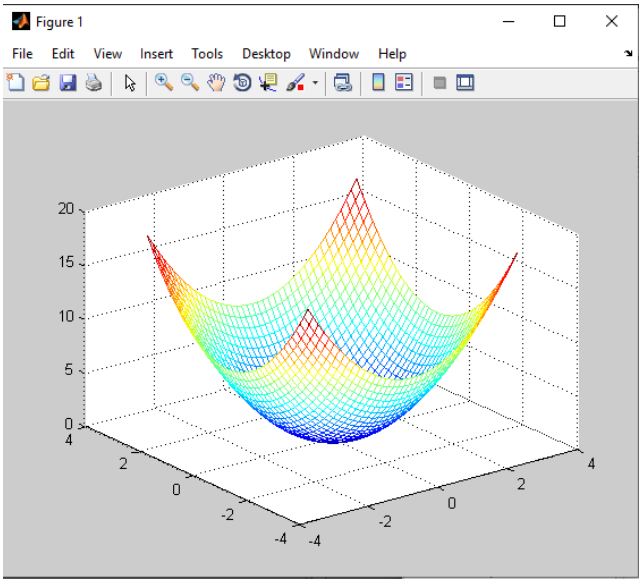



```
[X,Y]=meshgrid(-3:0.15:3);
Z=X.^2+Y.^2;
plot3(X,Y,Z,'o')
[X,Y]=meshgrid(-3:0.15:3);
Z=X.^2+Y.^2;
plot3(X,Y,Z,'o')
```



MATLAB дозволяє наносити на графік додаткову інформацію, зокрема, відповідність кольорів значенням функції. Сітка генерується за допомогою команди *meshgrid*, що викликається з двома аргументами. Аргументами є вектори, елементи яких відповідають сітці прямокутної області побудови функції. Можна використовувати один аргумент, якщо область побудови функції квадрат. Для обчислення функції слід використовувати поелементні операції.

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



Розглянемо основні можливості, що надаються MATLAB для візуалізації двох змінних функцій, на прикладі побудови поверхні графіка функції:

$$z(x,y) = 4\sin(2\pi x)\cdot\cos(1{,}5\pi y)\cdot(1-x^2)\cdot y\cdot(1-y)$$

на прямокутній області визначення $x \in [-1, 1]$, $y \in [0, 1]$.

Підготуємо матриці з координатами вузлів сітки та значеннями функції:

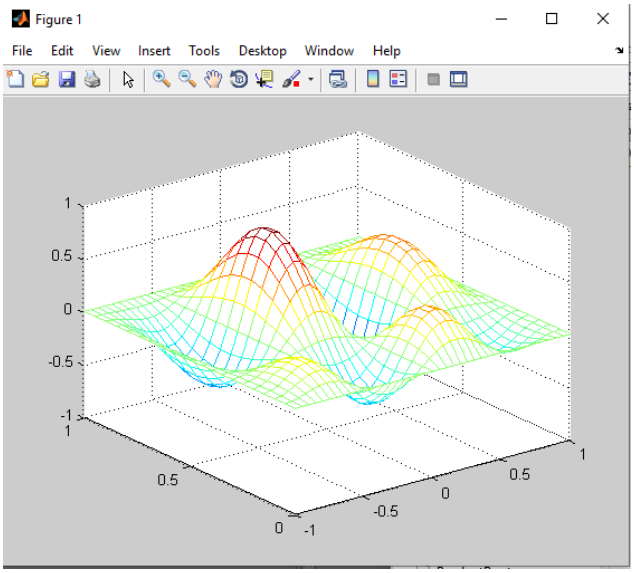
```
[X, Y] = meshgrid(-1:0.05:1, 0:0.05:1);
Z=4*sin(2*pi*X).*cos(1.5*pi*Y).*(1-X.^2).*Y.*(1-Y);
```

Для побудови каркасної поверхні використовується функція *mesh*, що викликається з трьома аргументами *mesh*(*X*, *Y*, *Z*). Колір ліній поверхні відповідає значенням функції. MATLAB малює лише видиму частину поверхні.

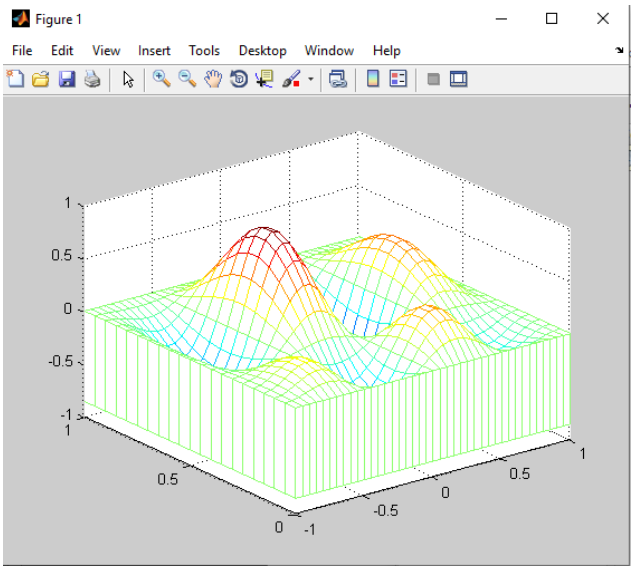
За допомогою команди *hidden off* можна зробити каркасну поверхню прозорою, додавши приховану частину. Команда *hidden on* видаляє невидиму частину поверхні, повертаючи графіку колишній вигляд.

Функція *surf* будує каркасну поверхню графіка функції та заливає кожну клітину поверхні певним кольором, що залежить від значень функції у точках, що відповідають кутам клітини. У межах кожної клітини колір незмінний.

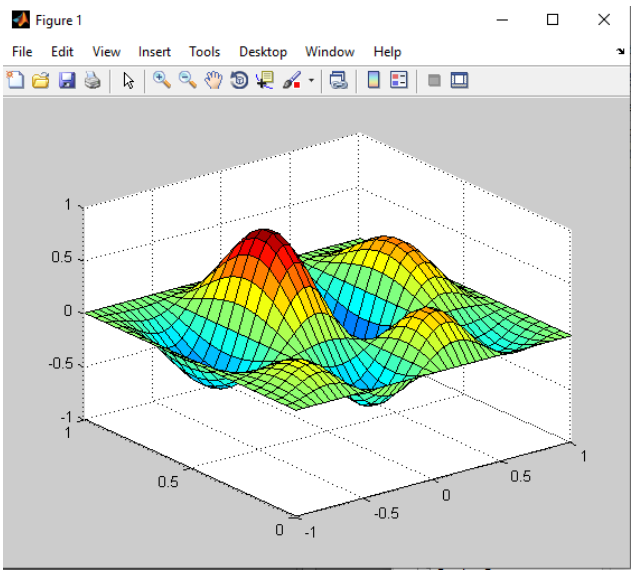
```
» mesh(X,Y,Z)
```




```
» meshz(X,Y,Z)
```




```
» surf(X,Y,Z)
```



Команда *shading flat* дозволяє прибирати каркасні лінії. Для отримання поверхні, плавно залитої кольором, що залежить від значень функції, призначена команда *shading interp*.

За допомогою *shading faceted* можна повернутись до поверхні з каркасними лініями.

Тривимірні графіки, одержувані за допомогою описаних вище команд, зручні для отримання уявлення про форму поверхні, проте за ними важко судити про значення функції. У MATLAB визначено команду *colorbar*, яка виводить поряд із графіком стовпець, що встановлює відповідність між кольором та значенням функції (рис. 5.31). Побудуйте за допомогою *surf* графік поверхні та доповніть його інформацією про колір:

```
surf(X,Y,Z)
colorbar
```

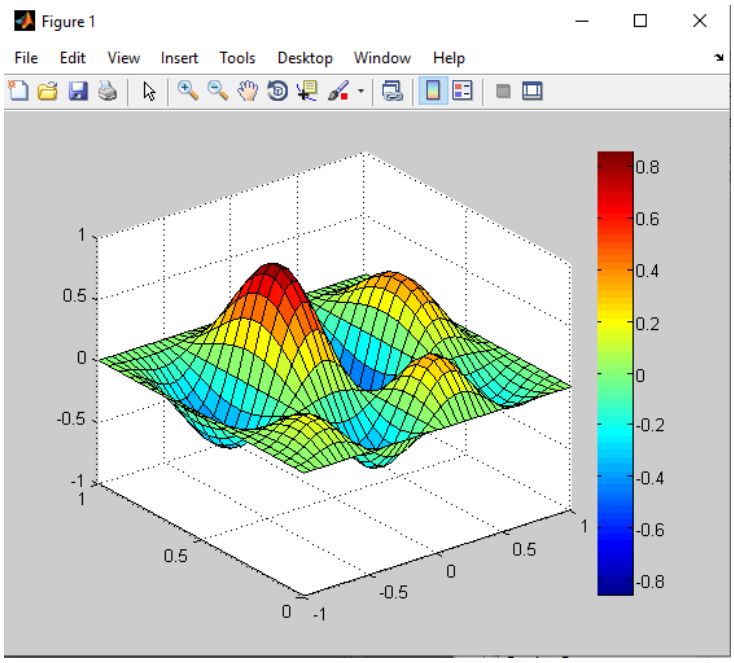


Рис. 5.31. Поверхня із стовпцем, що встановлює відповідність між кольором та значенням функції.

Команду *colorbar* можна застосовувати у поєднанні з усіма функціями, що будують тривимірні об'єкти.

Користуючись кольоровою поверхнею, важко зробити висновок про значення функції у тій чи іншій точці площини *xy*. Команди *meshc* або *surfc* дозволяють отримати більш точне уявлення про поведінку функції. Ці команди будують каркасну поверхню (рис. 5.32) або залиту кольором каркасну поверхню та розміщують на площині *xy* лінії рівня функції (лінії сталості значень функції):

```
surfc(X,Y,Z)
colorbar
```

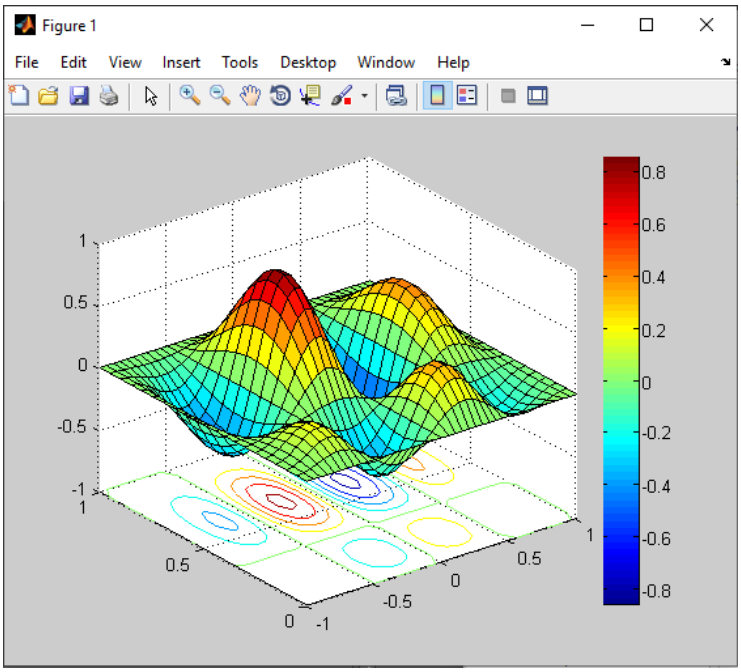


Рис. 5.32. Поверхня функції із лініями рівня.

MATLAB дозволяє побудувати поверхню, що складається з ліній рівня за допомогою функції *contour*3. Цю функцію можна використовувати так, як і описані вище *mesh, surf, meshc і surfc* з трьома аргументами. При цьому кількість ліній рівня вибирається автоматично. Є можливість задати четвертим аргументом в *contour*3 або число ліній рівня, або вектор, елементи якого дорівнюють значенням функції, що відображаються у вигляді ліній рівня. Завдання вектора (четвертого аргументу *levels*) зручне, коли потрібно досліджувати поведінку

функції певної області її значень (зріз функції). Побудуємо, наприклад, поверхню, що складається з ліній рівня (рис. 5.33), які відповідають значенням функції від 0 до 0,5 з кроком 0,01:

```
levels = [0:0.01:0.5];
contour3(X, Y, Z, levels)
colorbar
```

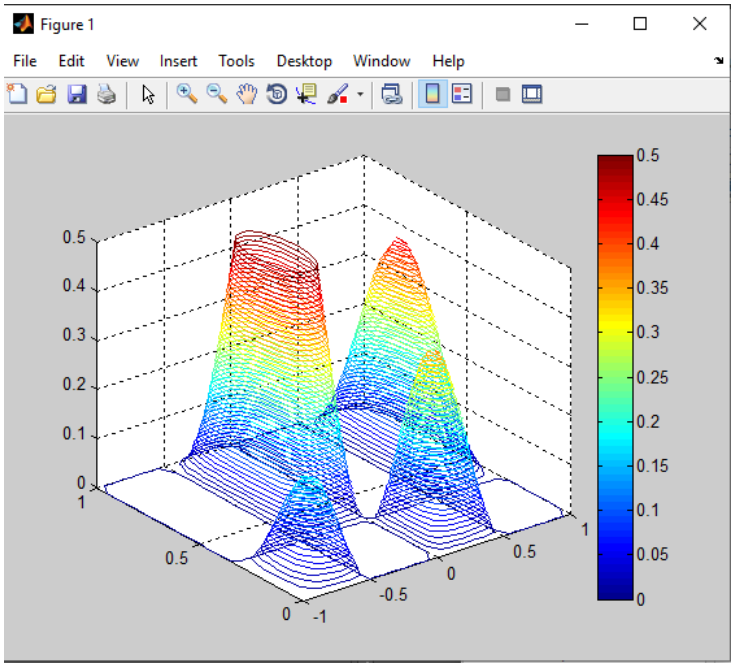


Рис. 5.33. Поверхня функції, що складається з ліній рівня.

5.9. Побудова контурних графіків функцій двох змінних

MATLAB надає можливість отримувати різні типи контурних графіків за допомогою функцій *contour* та *contourf*. Розглянемо їх можливості на прикладі функції

$$z(x,y) = 4\sin(2\pi x)\cdot\cos(1{,}5\pi y)\cdot(1-x^2)\cdot y\cdot(1-y).$$

Використання *contour* з трьома аргументами `contour(X,Y,Z)` призводить до графіка, на якому показані лінії рівня на площині *xy*, але без

вказівки числових значень на них. Такий графік є малоінформативним, не дозволяє дізнатися значення функції на кожній лінії рівня. Використання команди *colorbar* не дозволить точно визначити значення функції. Кожній лінії рівня можна надати значення, яке приймає на ній досліджувана функція, за допомогою визначеної в MATLAB функції *clabel*. Функція *clabel* викликається з двома аргументами: матрицею, що містить інформацію про лінії рівня та вказівником на графік, на якому слід нанести розмітку. Користувачеві не потрібно самому створювати аргументи *clabel*. Функція *contour*, викликана з двома вихідними параметрами, не тільки будує лінії рівня, а й знаходить необхідні для *clabel* параметри (рис. 5.34). Використовуйте *contour* з вихідними аргументами *CMatr* та *h* (у масиві *CMatr* міститься інформація про лінії рівня, а в масиві *h*-покажчики). Завершіть виклик *contour* крапкою з комою для не допущення виведення на екран значень вихідних параметрів та нанесіть на графік сітку:

```
[CMatr, h] = contour(X, Y, Z);
clabel(CMatr, h)
grid on
```

Додатковим аргументом функції *contour* (як і *contour*3, описаної вище) може бути число ліній рівня або вектор, що містить значення функції, яким потрібно побудувати лінії рівня.

Наочну інформацію про зміну функції дає заливка прямокутника на площині *xy* кольором, що залежить від значення функції у точках площини (рис. 5.35). Для побудови таких графіків призначено функцію *contourf*, використання якої не відрізняється від застосування *contour*. У наступному прикладі виводиться графік, який складається з двадцяти ліній рівня, а проміжки між ними заповнені кольорами, що відповідають значенням функції, що досліджується:

```
contourf(X, Y, Z, 20)
colorbar
```

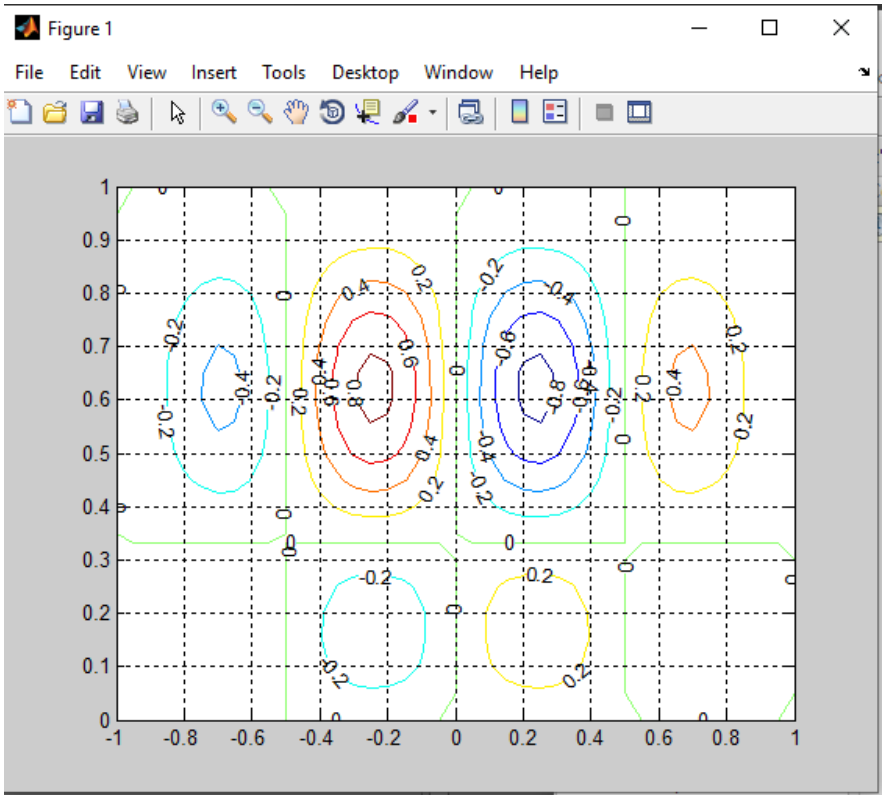


Рис. 5.34. Побудова контурних графіків функцій двох змінних.

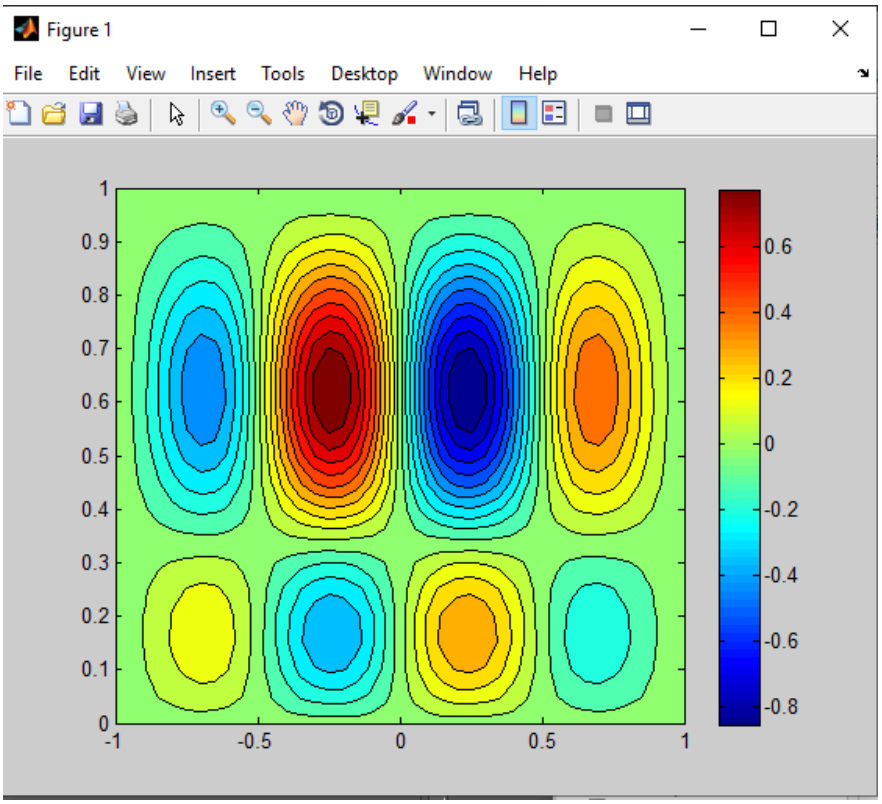


Рис. 5.35. Побудова контурних графіків функцій двох змінних із заливкою.

5.10. Оформлення графіків функцій

Простим та ефективним способом зміни колірного оформлення графіка є встановлення кольорової палітри за допомогою функції *colormap*. Наступний приклад демонструє підготовку графіка функції (рис. 5.36) для друку на монохромному принтері, використовуючи палітру *gray*:

```
surfc(X, Y, Z)
colorbar
colormap(gray)
title('Графік функції z(x,y)')
xlabel('x')
ylabel('y')
zlabel('z')
```

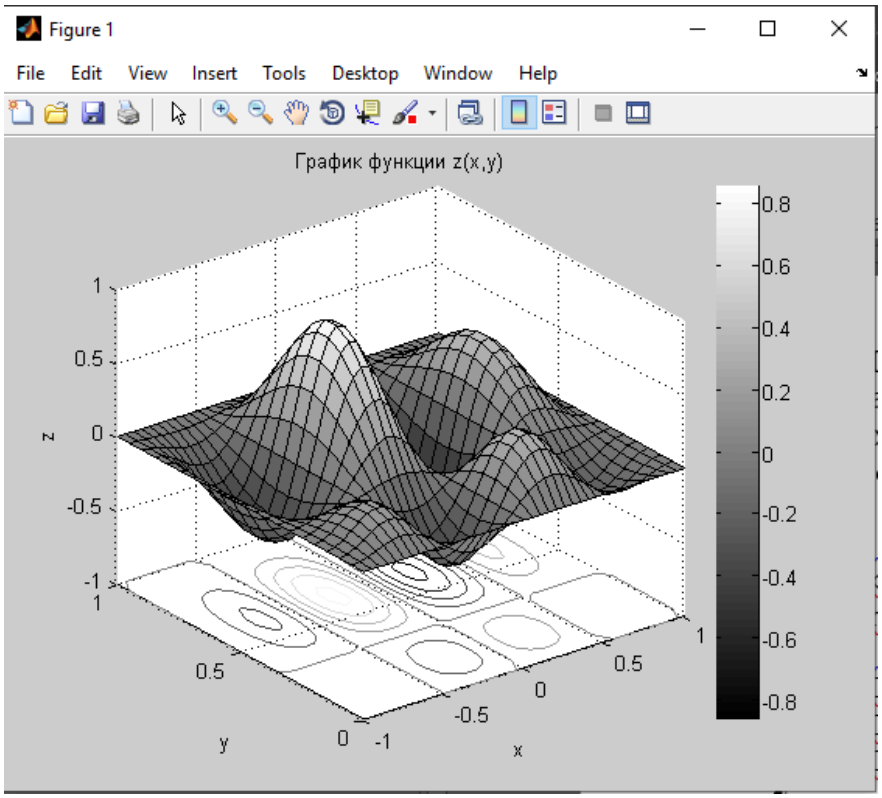


Рис. 5.36. Побудова графіка функції використовуючи палітру *gray*.

Команда *colormap*(*gray*) змінює палітру графічного вікна, тобто наступні графіки будуть виводитись у цьому вікні також у сірих тонах. Для відновлення початкового значення панелі слід застосувати команду *colormap*('*default*'). Кольорові палітри, доступні в MATLAB, наведені у табл. 5.2.

Таблиця 5.2 – Кольорові палітри MATLAB

Палітра	**Зміна кольору**
autumn	Плавна зміна: червоний- помаранчевий-жовтий
bone	Схожа на палітру *gray*, але із легким відтінком синього кольору
colorcube	Кожен колір змінюється від темного до яскравого
cool	Відтінки блакитного та пурпурового кольорів
copper	Відтінки мідного кольору
flag	Циклічна зміна: червоний-білий-синій-чорний
gray	Відтінки сірого
hot	Плавна зміна: чорний-червоний-помаранчевий-жовтий-білий
hsv	Плавна зміна кольорів веселки
jet	Плавна зміна: синій-блакитний-червоний-зелений-жовтий- червоний
pink	Схожа на палітру *gray*, але із легким відтінком коричневого кольору
prism	Циклічна зміна: червоний-помаранчевий-жовтий-зелений-синій-фіолетовий
spring	Відтінки пурпурового та жовтого
summer	Відтінки зеленого та жовтого
vga	Палітра *Windows* із шістнадцяти кольорів
white	Один білий колір
winter	Відтінок синього та зеленого

5.11. Виведення кількох графіків на одні осі

Для відображення кількох графіків функцій однієї змінної на одних осях використовувалися можливості функцій *plot, plotyy, semilogx, semilogy, loglog*. Вони дозволяють виводити графіки кількох функцій, задаючи відповідні векторні аргументи парами, наприклад *plot*(*x, f, x, g*). Однак для об'єднання тривимірних графіків їх не можна використовувати.

Для поєднання таких графіків призначена команда *hold on*, яку потрібно задати перед побудовою графіка. У наступному прикладі об'єднання двох графіків (площини та конуса) призводить до їх перетину. Конус визначається параметрично наступними залежностями:

$x(u,v) = 0{,}3 \cdot u \cdot \cos v;\ y(u,v) = 0{,}3 \cdot u \cdot \sin v;\ z(u,v) = 0{,}6 \cdot u;\ u,v \in [-2\pi, 2\pi]$.

Для графічного відображення конуса спочатку необхідно згенерувати за допомогою двокрапки вектор-стовпець і вектор-рядок, що містять значення параметрів на заданому інтервалі (важливо, що u - вектор-стовпець, а v - вектор-рядок):

```
u = [-2*pi:0.1*pi:2*pi]';
v = [-2*pi:0.1*pi:2*pi];
```

Далі формуються матриці *X*, *Y*, що містять значення функцій $x(u,v)$ і $y(u,v)$ у точках, що відповідають значенням параметрів. Формування матриць виконується за допомогою зовнішнього добутку векторів.

Зауваження. Зовнішнім добутком векторів $a = (a_1,...,a_j,...,a_N)$, $b = (b_1,...,b_k,...,b_M)$ називається матриця $C = (c_{jk}),\ \ j = \overline{1,N},\ \ k = \overline{1,M}$ розміром $N \times M$, елементи якої обчислюються за формулою $c_{jk} = a_j b_k$.

Вектор a є вектор-стовпцем і у MATLAB представляється у вигляді двомірного масиву розміру $N \times 1$. Вектор-стовпець b при транспонуванні переходить у вектор-рядок розміру $1 \times M$. Вектор-стовпець і вектор-рядок є матриці, у яких один із розмірів дорівнює одиниці. Фактично $C = ab^T$, де множення відбувається за правилом матричного добутку. Для обчислення матричного добутку MATLAB використовується оператор "зірочка". Визначимо зовнішній твір для двох векторів:

```
» a = [1;2;3];
» b = [5;6;7];
```

```
» C = a*b'
C =
     5   6   7
    10  12  14
    15  18  21
```

Сформуємо матриці *X, Y*, які потрібні для графічного відображення конуса:

```
X = 0.3*u*cos(v);
Y = 0.3*u*sin(v);
```

Матриця *Z* повинна бути того ж розміру, що і матриці *X* та *Y*. Крім того, вона повинна містити значення, що відповідають значенням параметрів. Якби у функцію входив добуток *i* та *v*, то матрицю *Z* можна було заповнити аналогічно матрицям *X* і *Y* за допомогою зовнішнього добутку. З іншого боку, функцію *z*(*u, v*) можна подати у вигляді $z(u,v) = 0{,}6 \cdot u \cdot g(v)$, де $g(v) \equiv 1$. Тому для обчислення *Z* можна застосувати зовнішній добуток векторів, де вектор-рядок має ту ж розмірність, що і *v*, але складається з одиниць:

```
Z = 0.6*u*ones(size(v));
```

Усі необхідні матриці для відображення конуса створені. Завдання площини виконується так:

```
[X,Y] = meshgrid(-2:0.1:2);
Z = 0.5*X+0.4*Y;
```

Тепер не складно записати і повну послідовність команд для побудови конуса і площини, що перетинаються:

```
u = [-2*pi:0.1*pi:2*pi]';
v = [-2*pi:0.1*pi:2*pi];
X = 0.3*u*cos(v);
Y = 0.3*u*sin(v);
Z = 0.6*u*ones(size(v));
surf(X, Y, Z)
```

```
[X,Y] = meshgrid(-2:0.1:2);
Z = 0.5*X+0.4*Y;
hold on
mesh(X, Y, Z)
hidden off
```

В результаті роботи програми отримаємо геометричну фігуру, наведену на рис. 5.37.

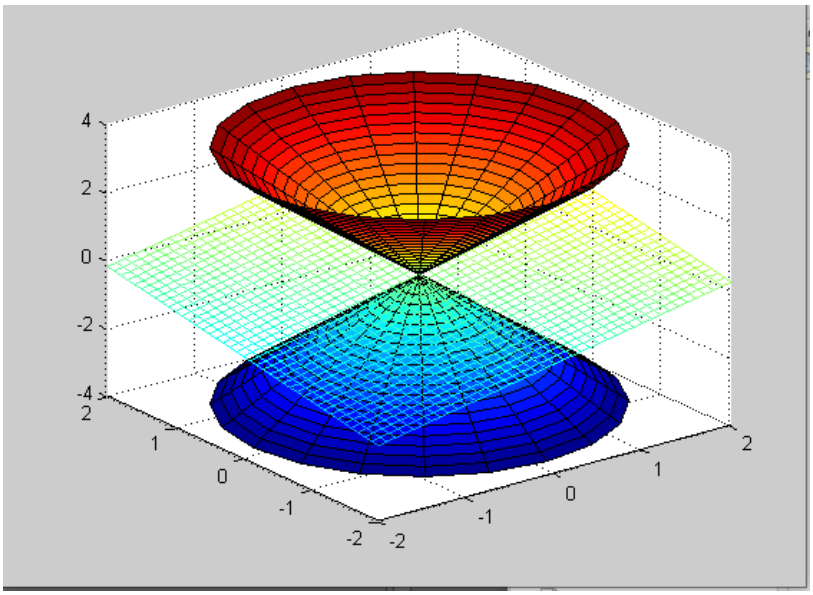


Рис. 5.37. Виведення кількох графіків на одні осі

Команда *hidden off* застосована для того, щоб показати частину конуса, що знаходиться під площиною.

Зверніть увагу, що команда *hold on* поширюється на всі наступні виведення графіків у поточне вікно. Для розміщення графіків у нових вікнах слід виконати команду *hold off*. Команда *hold on* може застосовуватися і для розташування декількох графіків функцій однієї змінної, наприклад,

```
» plot(x,f,x,g)
```

еквівалентно послідовності

```
» plot(x,f)
» hold on
» plot(x,g)
```

5.12. Логарифмічний масштаб координатної осі

Для побудови графіків функцій зі значеннями *x* та *y*, що змінюються в широких межах, нерідко використовуються логарифмічні масштаби замість лінійних. Розглянемо команди, що будують графіки у логарифмічному масштабі.

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

```
% Графік експоненційної функції у логарифмічному масштабі
x=logspace(-1,3);
loglog(x,exp(x)./x)
grid on
```

Зверніть увагу, що команда в кінці цього прикладу виводить координатну мірну сітку (рис. 5.38). Нерівномірне розташування ліній координатної мірної сітки свідчить про логарифмічний масштаб осей.

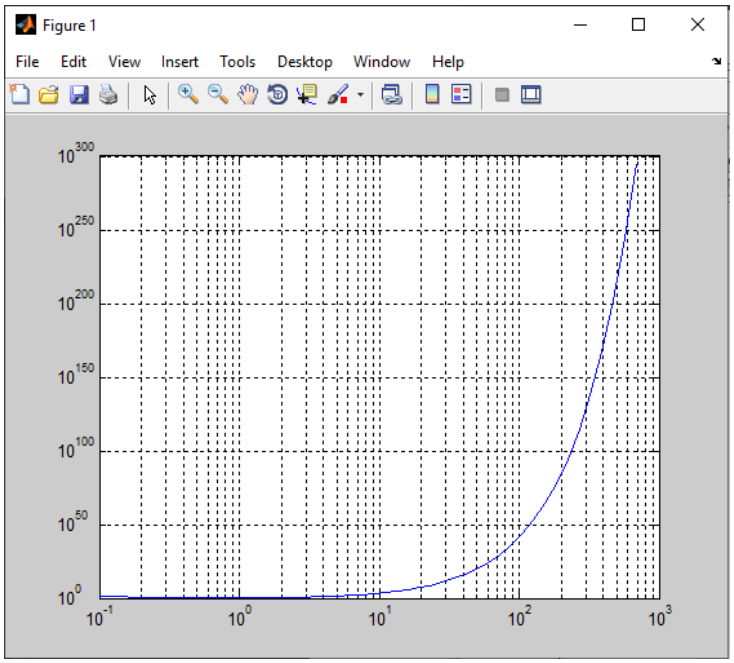


Рис. 5.38. Графік експоненти в логарифмічному масштабі по осі абсцис.

5.13. Стовпчикові діаграми та гістограми

Нижче представлені команди для побудови стовпчикових діаграм.

***bar*(*X, Y*) –** будує стовпчиковий графік елементів масиву *Y* у позиціях, що визначаються вектором *X*, з упорядкованими в порядку зростання значень елементами. Якщо *X* та *Y* – двомірні масиви однакового розміру, то стовпці будуються попарно з надбудовою один на одному.

***bar*(*Y*)** – будує графік значень елементів одномірного масиву *Y*. Фактично це попередня команда *X* = 1:*M*.

***bar*(*X, Y, width*)** або ***bar*(*Y, width*)** – команда аналогічна раніше розглянутим, але зі специфікацією ширини стовпців (при *width* > 1 стовпці перекриваються). За замовченням встановлено *width* = 0.8.

Можливе застосування цих команд і в наступному вигляді:

```
bar(...,'Специфікація')
```

для завдання специфікації графіків, наприклад типу ліній, кольору тощо за аналогією з командою *plot*.

Приклад побудови стовпчикової діаграми представлений нижче (рис. 5.39):

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

Наведемо ще один приклад (рис. 5.40).

```
y=[1 2 3; 4 5 6; 7 8 9];
bar(y)
```

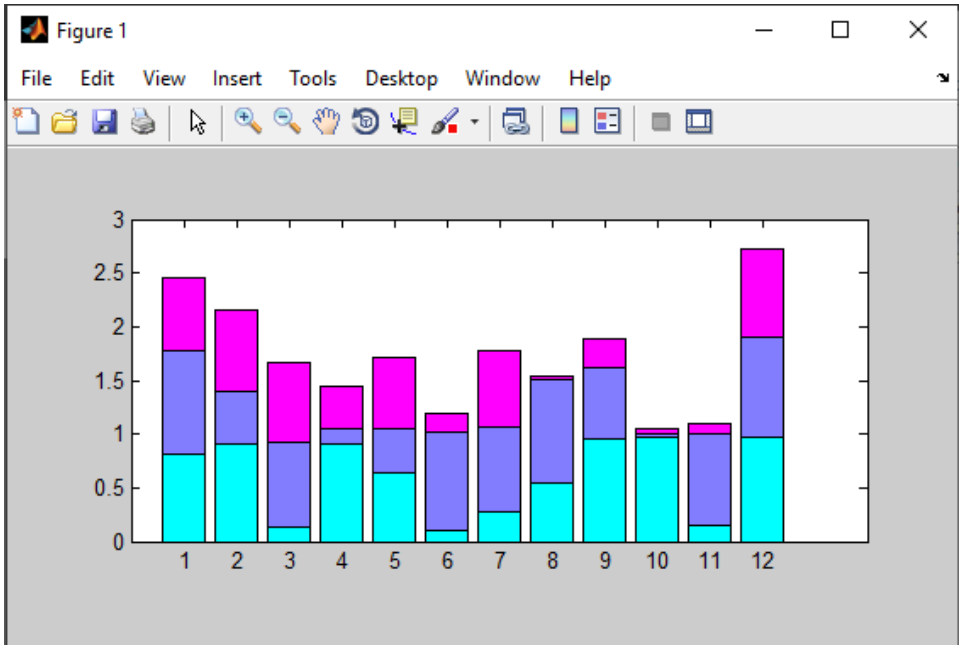


Рис. 5.39. Приклад побудови стовпчикової діаграми з вертикальними стовпцями.

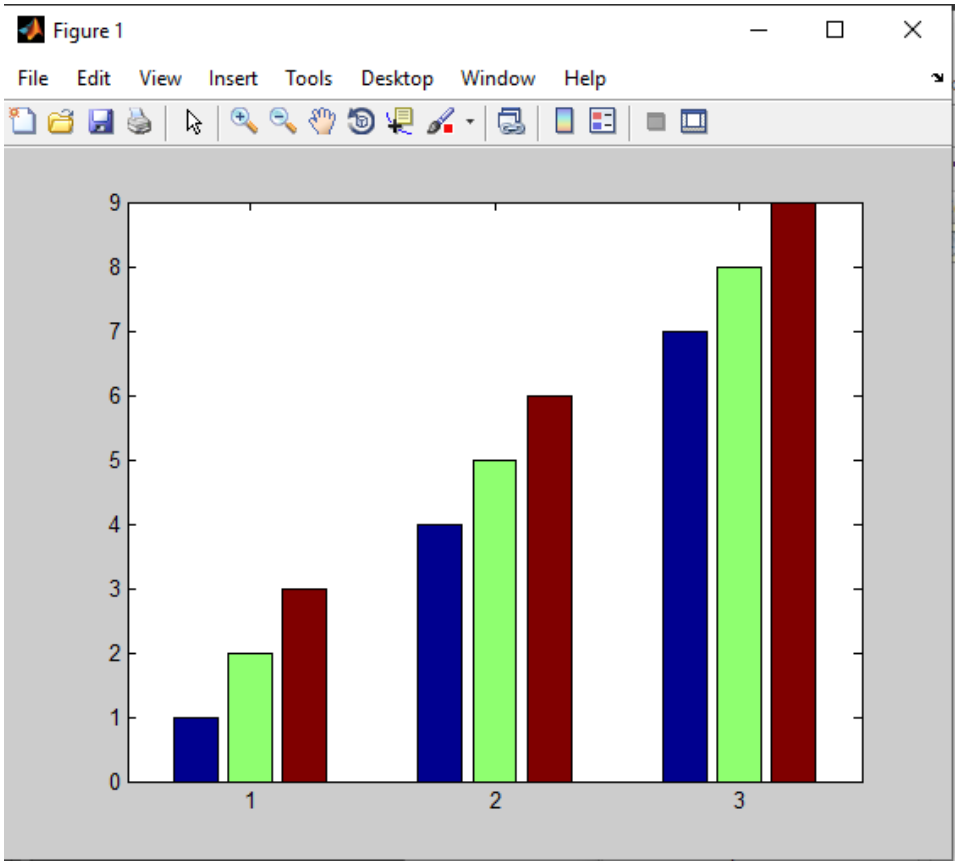


Рис. 5.40. Приклад побудови стовпчикової діаграми.

Команда *barh*(...) будує стовпчикові діаграми з горизонтальним розташуванням стовпців (рис. 5.41). Наприклад:

```
% Стовпчикова діаграма з горизонтальними стовпцями
subplot(2,1,1)
barh(rand(5,3),'stacked')
colormap(cool)
```

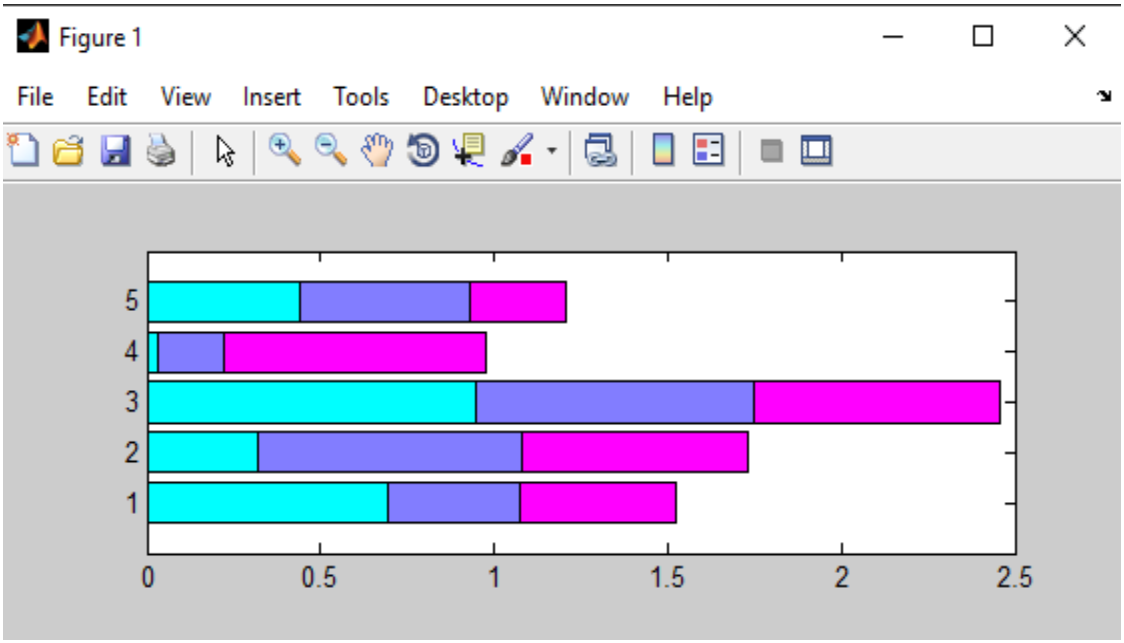


Рис. 5.41. Совпчикові діаграми з горизонтальним розташуванням стовпців.

Яке розташування стовпців вибрати, це вже залежить від користувача, який використовує дані команди для представлення своїх даних.

Класична гістограма характеризує число попадань значень елементів вектора *Y* в інтервали *M* з поданням цих чисел у вигляді стовпчиків відповідної висоти. Для отримання даних для гістограми служить функція *hist*, що записується у вигляді:

***N* = *hist*(*Y*)** – повертає вектор числа попадань для 10 інтервалів. Якщо *Y* матриця, то видаються дані щодо кількості влучень у її стовпці.

***N* = *hist*(*Y*, *M*)** – аналогічна вище розглянутої функції, але для числа попадань *M* (скаляр).

***N* = hist(*Y*, *X*)** – повертає число попадань елементів вектора *Y* з центральними відліками, заданими елементами вектора *X*.

[*N*, *X*] = *hist*(...) – повертає кількість попадань в інтервали та дані про центри інтервалів.

Команда *hist*(...) із синтаксисом (...), аналогічним наведеному вище, будує графік гістограми. Нижче наведено приклад її застосування - будується гістограма (рис. 5.42) для 1000 випадкових чисел і виводиться вектор з даними про числа їх влучень в інтервали, специфіковані вектором *x*:

```
%Побудова гістограми для 1000 випадкових чисел
x=-3:0.2:3;
y=randn(1000,1);
hist(y,x)
h=hist(y,x)
h =
Columns 1 through 13
3     0     1     6    10    10     3    22    34    45    53    57    63
Columns 14 through 26
72    88    73    80    86    66    58    49    29    19    17    15    11
Columns 27 through 31
6     3     6     3     2
```

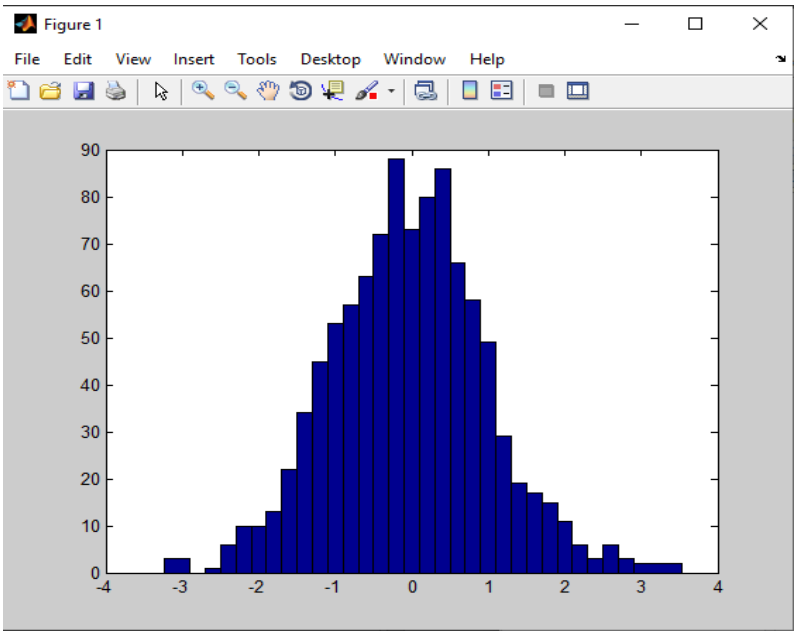


Рис. 5.42. Побудована гістограма.

Неважко помітити, що розподіл випадкових чисел, що розглядається, близький до нормального. Збільшивши їх число, можна спостерігати ще більшу відповідність цьому.

5.14. Побудова графіків спеціального виду

Сходові графіки являють собою сходи з огинаючою, представленою функцією *y*(*x*). Такі графіки використовуються, наприклад, для представлення функції *y*(*x*), поданої результатами низки вимірювань її значень. При цьому в проміжках між ними значення функції вважаються постійними та рівними величині останнього результату вимірювання.

Для побудови сходових графіків використовуються команди *stairs*:

***stairs*(*Y*) –** будує сходовий графік за даними вектора *Y*.

***stairs*(*X, Y*) –** будує сходовий графік за даними вектора *Y* з координатами *x* переходів від сходинки до сходинки заданими значеннями елементів вектора *X* (вони повинні йти у зростаючому порядку).

***stairs*(*..., S*)** – аналогічна по дії вище описаним командам, але будує графік лініями, стиль яких задається рядками *S*.

Наступний приклад демонструє побудову сходового графіка:

```
%Сходовий графік функції x^2
x=0:0.25:10;
stairs(x,x.^2);
```

Побудований для цього прикладу графік представлений на рис. 5.43. При цьому, відліки беруться через рівні проміжки горизонтальної осі.

Функція ***H=stairs*(*X,Y*)** повертає вектор ординат, а функція **[*XX,YY*]=*stair*(*X,Y*)** сама собою графік не будує, але повертає вектори *XX* і *YY*, які дозволяють побудувати графік з допомогою команди *plot*(*XX,YY*).

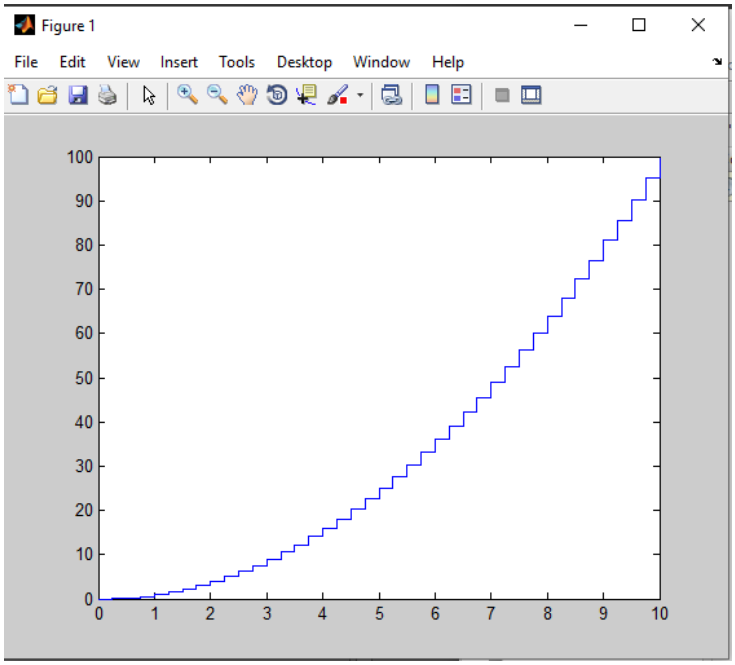


Рис. 5.43. Побудова сходового графіка.

Якщо дані для побудови функції визначені з помітною похибкою, то використовують графіки функцій типу *errorbar* – з оцінкою похибки кожної точки шляхом її подання у вигляді літери «I», висота якої дорівнює подвоєній похибці ординати точки. Команда *errorbar* використовується у вигляді:

***errorbar*(*X, Y, L, U*)** – будує графік значень елементів вектора *Y* залежно від даних у векторі *X* із зазначенням нижньої і верхньої меж похибки, заданих у векторах *L* і *U*.

***errorbar*(*X, Y, E*)** і ***errorbar*(*Y, E*)** – будує графіки функції *Y*(*X*) з зазначенням меж похибки у вигляді [*Y* – *E*, *Y*+*E*].

***errorbar*(..., '*LineSpec*')** – аналогічна описаним вище командам, але дозволяє будувати лінії зі специфікацією '*LineSpec*', аналогічною застосованою в команді *plot*.

Наступний приклад ілюструє застосування команди *errorbar*.

```
%Графік функції із зазначенням похибки кожної точки
x=-2:0.1:2;
```

```
y=erf(x);
e = rand(size(x))/10;
errorbar(x,y,e);
```

Побудований графік показано на рис. 5.44.

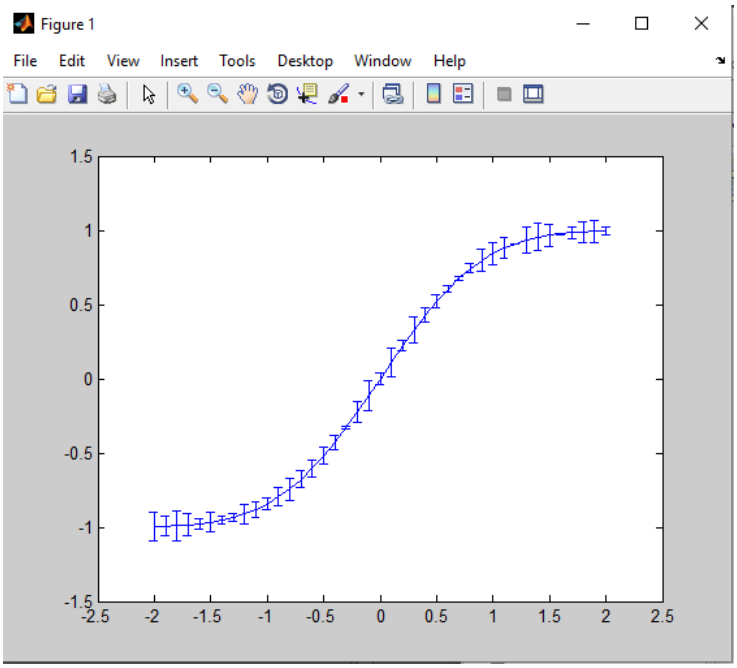


Рис. 5.44. Побудова графіку функції із зазначенням похибки кожної точки.

Функція, що записується у вигляді *H* = *errorbar*(...) повертає вектор ординат, використовуваних командою *errorbar*.

Ще один корисний вид графіка функції *y*(*x*) – її представлення дискретними відліками. Кожен відлік представляється вертикальною лінією, увінчаною кружком, причому висота межі відповідає координаті у точки.

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

***stem*(*Y*) –** будує графік функції з ординатами у векторі *Y* у вигляді відліків.

***stem*(*X, Y*) –** будує графік відліків з ординатами у векторі *Y* і абсцисами у векторі *X*.

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

stem(...,'linespec') – дає побудови, аналогічні описаним раніше командам, але зі специфікацією ліній, подібною до наведеної для функції *plot*.

Наступний приклад ілюструє застосування команди *stem*.

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

Отриманий для цього прикладу графік показаний на рис. 5.45.

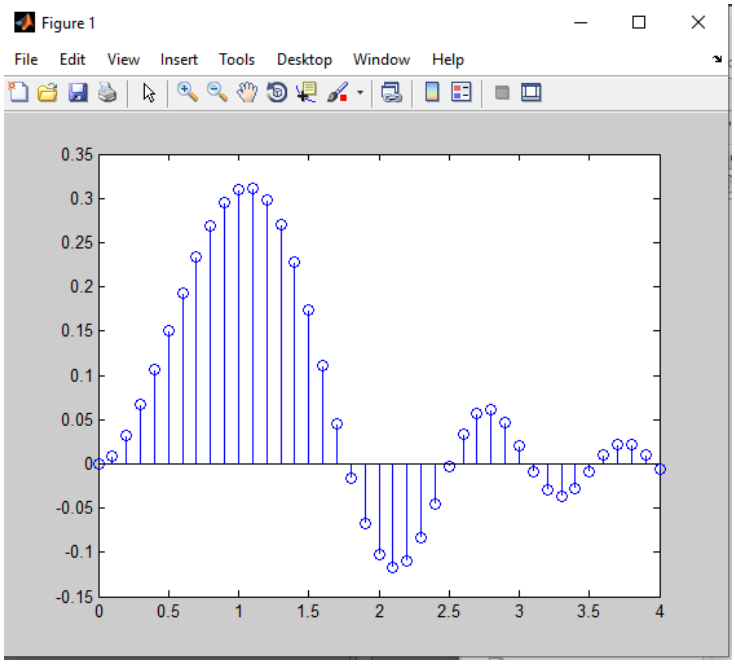


Рис. 5.45. Графік дискретних відліків функції.

Функція *H* = *stem*(...) повертає вектор висот точок поверхні.

5.15. Найпростіша анімація

При вивченні руху точки на площині або тривимірному просторі корисно не тільки побудувати траєкторію точки, але і стежити за рухом точки по траєкторії. MATLAB надає можливість отримати анімований графік, на якому кружок, що

позначає точку, переміщюється на площині або в просторі, залишаючи слід у вигляді лінії — траєкторії руху. Графік схожий на комету, що летить, з хвостом (рис. 5.46). Тому для побудови таких анімованих графіків застосовуються функції *comet* та *comet*3.

```
t=[0:0.001:10];
x=sin(t)./(t+1);
y=cos(t)./(t+1);
comet(x,y)
```

При виконанні останньої команди слідкуйте за тим, щоб вікно з графіком було поверх інших вікон. Зверніть увагу, що при зміні розмірів графічного вікна або його мінімізації і подальшому відновленні траєкторія руху пропадає. Це пов'язано зі способом, який застосовує MATLAB для побудови графіка.

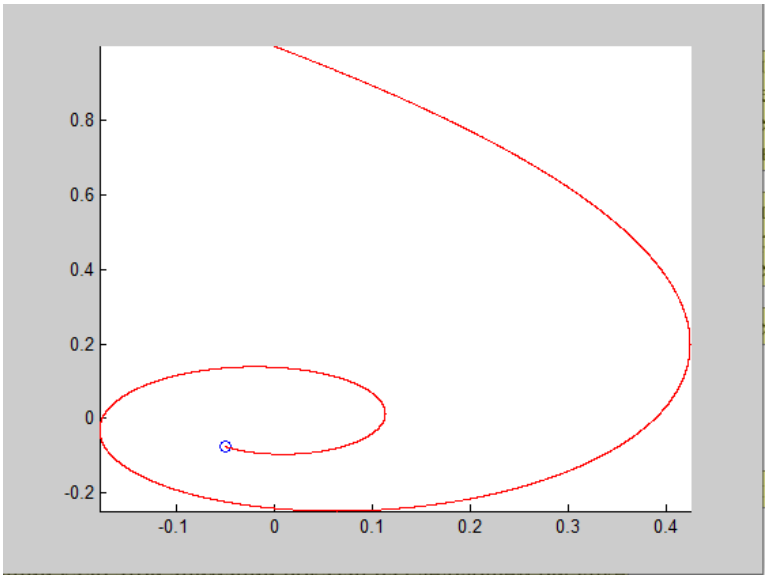


Рис. 5.46. Графік руху комети.

Швидкістю руху кружка можна керувати, задаючи різні кроки під час створення вектора. Використання *comet* з одним аргументом (вектором) призводить до побудови графіки значень компонентів вектора, що динамічно відображається, залежно від їх номерів.

Для побудови траєкторії точки, що переміщається у просторі, використовується функція *comet*3.

```
t=[0:0.1:100];
x=cos(t).*abs(t-50);
y=sin(t).*abs(t-50);
z=t;
comet3(x,y,z)
```

Останній кадр цієї анімації показано на наступному рис. 5.47.

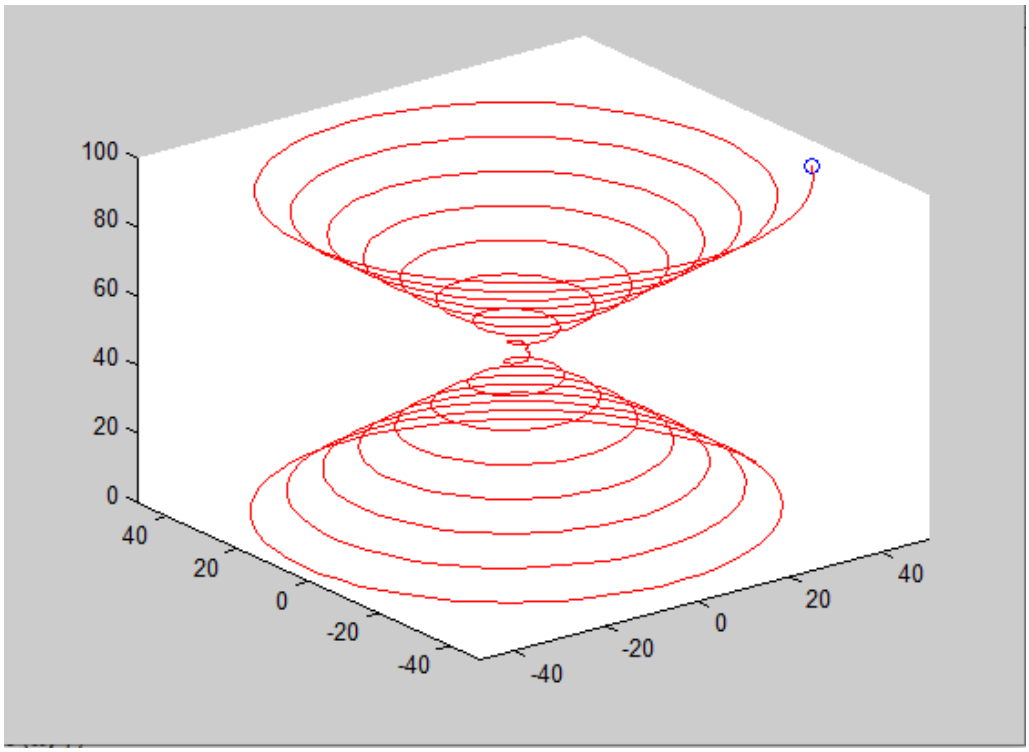


Рис. 5.47. Графік руху комети, що переміщається у просторі.

Графічна функція *ezplot*3 має опцію '*animate*', яка призводить до руху невеликого кружка по кривій, побудованої з її використанням (рис. 5.48):

```
ezplot3('sin(t)', 'cos(t)', 't',[0,6*pi],'animate')
```

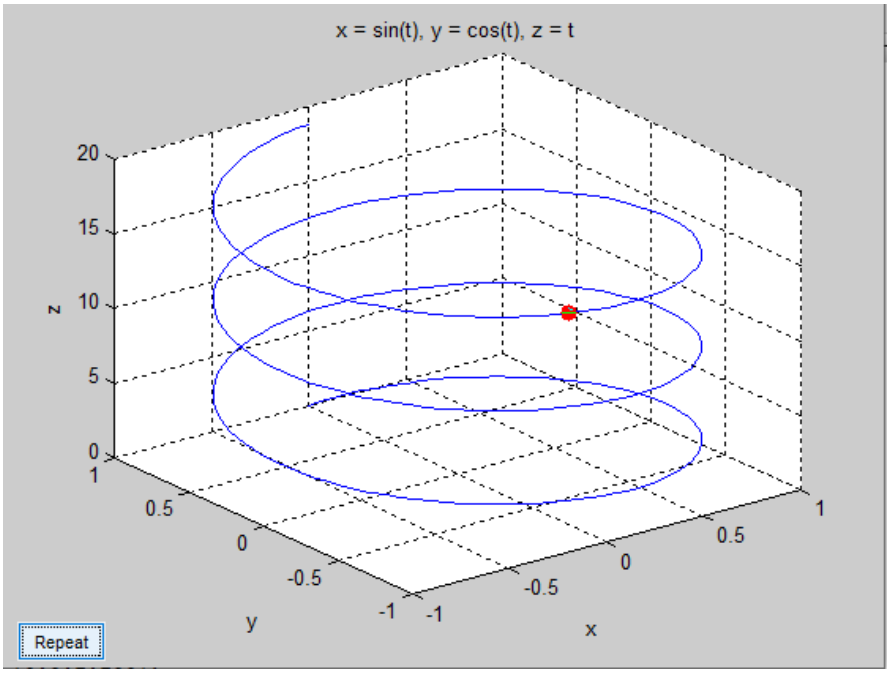


Рис. 5.48. Графік руху кружка по кривій.

Для побудови анімації у загальному випадку MATLAB передбачає використання функції *getframe*, яка захоплює графічні кадри і може зберігати їх у масиві. Інша функція *movie* програє такий масив у графічному вікні, створюючи тим самим анімацію. Ось простий приклад

```
figure
t=0:0.01:2*pi;
set(gca,'nextplot','replacechildren'); % встановлює
%режим заміни кадрів у графічному вікні
for j = 1:20
x=sin(j.*t).*exp(-t);
plot(t,x);
F(j)=getframe;  %  запам'ятовує поточний графічний кадр
end
movie(F,10) % Програє масив F десять разів
```

Усередині циклу може стояти будь-яка графічна функція (рис. 5.49). Створені з її допомогою кадри зберігатимуться в масив *F* і відтворюватимуться функцією *movie*.

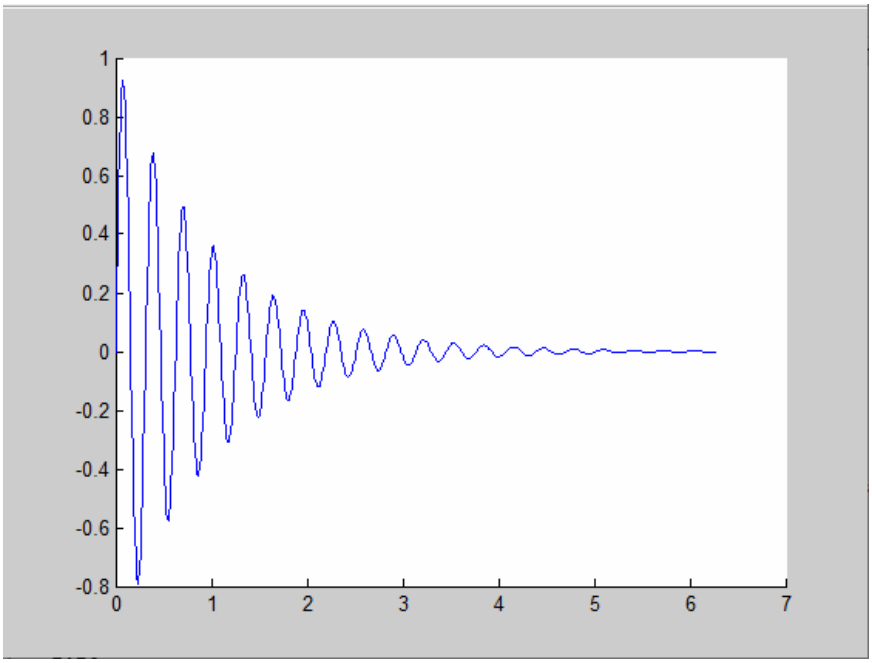


Рис. 5.49. Анімація графічної функції.

Наведимо ще один приклад із довідкової системи MATLAB. У прикладі створюється наступний файл сценарій и виконання команди *mymovie*:

```
Z= peaks;
surf(Z);
axis tight
set(gca,'nextplot','replacechildren');
% Record the movie
for j = 1:20
surf(sin(2*pi*j/20)*Z,Z)
F(j) = getframe;
end
movie(F,20) % Play the movie twenty times
```

Контрольні запитання:

1. Перелічите загальні можливості графіки у системі комп'ютерної математики MATLAB та наведіть приклади застосування базових графічних об’єктів?

2. Які особливості побудови графіків у лінійному масштабі, наведіть приклади побудови графіків двох функції на одних і тих самих осях але різних відрізках?

3. Які особливості побудови графіків у логарифмічному масштабі, наведіть приклади побудови графіків двох функції на одних і тих самих осях але з двома різними масштабами?

4. Які існують функції для побудови тривимірних графіків у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?

5. Які існують функції для розбиття графічного вікна у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?

6. Які існують можливості для побудови контурних графіків функцій двох змінних у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?

7. Які існують функцій для створення стовпчикових діаграм у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?

8. Які існують функцій для створення гістограм у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?

9. Які існують функцій для побудови графіків спеціального виду у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?

10. Які засоби анімації існують у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?

РОЗДІЛ 6

ФУНКЦІЇ ОБРОБКИ ЗОБРАЖЕНЬ. ПРЯМЕ І ЗВОРОТНЕ ПЕРЕТВОРЕННЯ РАДОНУ

Розділ знайомить з функціями MATLAB, що використовуються при обчисленні та візуалізації прямого та зворотного двовимірного перетворення Радона – математичного методу, що лежить в основі томографії. Багато описаних тут функцій відноситься до пакету розширення *Image Processing Toolbox*. Однак для розуміння матеріалу, що викладається, необхідні базові знання по обробці зображень у СКМ MATLAB.

6.1. Спеціальні графічні функції

Розуміння двох функцій MATLAB *radon* та *iradon*, що використовуються для обчислення прямого та зворотного перетворення Радону, неможливе без наочного подання результатів їхньої роботи. У цьому підрозділі розглядаються деякі графічні функції, які найчастіше використовуються в цьому випадку. До таких функцій відносяться:

```
imread('файл',...)  - читання графічного файлу
imfinfo('файл',...) - інформація про графічний файл
image(матриця,...)  - намалювати образ матриці
colormap(матриця(m x 3))
imshow(матриця,...)
imagesc(матриця,...)
surface(матриця,...)
surf(матриця,...)
mesh(матриця,...)
```

Графічний файл у MATLAB представляється у вигляді матриці з елементами, що є цілими числами, які описують індекси (номери) кольорів у

палітри кольорів (*colormap*). У MATLAB є функції читання графічних файлів та відображення отриманих матриць. Розглянемо для прикладу рис. 6.1, який знаходиться у графічному файлі *foto*32*x*32.*bmp*, розміром 32 на 32 пікселі.

Рис. 6.1. Рисунок розміром 32×32 пікселі.

Команда

```
A=imread('foto32x32.bmp')
```

читає та друкує матрицю *A* розміром 32×32. Значення елементів матриці є номерами кольорів відповідних пікселів. Ось шматочок цієї матриці

```
15   15   15   15   0    0    0    0    0    1    6    …
15   0    0    6    0    7    7    8    8    8    8    …
15   15   0    0    0    1    7    8    7    8    8    …
15   7    0    2    1    7    7    8    8    8    8    …
… … … … … … … … … … … … … … … … … … … … … … … … …
```

Команда

```
image(A);
```

по цій матриці рисує картинку у графічному вікні (рис. 6.2). При цьому рисунок не дуже гарної якості і для отримання прийнятного зображення нам потрібно змінити параметри його відображення.

Коли матриця містить цілі позитивні числа, її елементи інтерпретуються як індекси в карті кольорів. Функція *image*(*A*) будує її образ у поточній карті - кожному елементу матриці буде відповідати деякий прямокутник на рисунку, зафарбований кольором, номер якого відповідає рядку матриці кольорів (палітра). Є також додаткові аргументи, які керують роботою функції *image*.

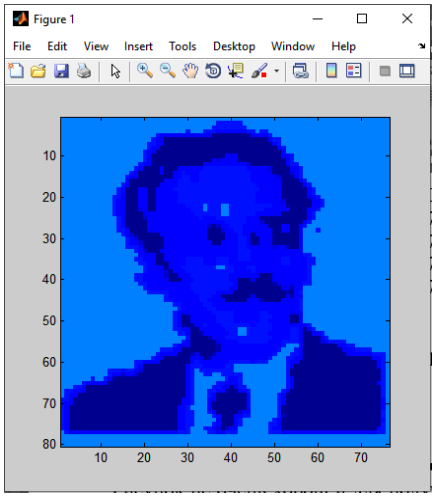


Рис. 6.2. Рисунок у графічному вікні.

Отриманим зображенням у графічному вікні можна керувати, використовуючи звичайні функції MATLAB, що керують графікою. Зокрема, у прикладі рисунок витягнутий по горизонталі. Команда

```
axis equal;
```

вирівнює горизонтальні та вертикальні розміри пікселів у графічному вікні (рис. 6.3). Вона встановлює коефіцієнт стиснення зображення однаковим у всіх осях.

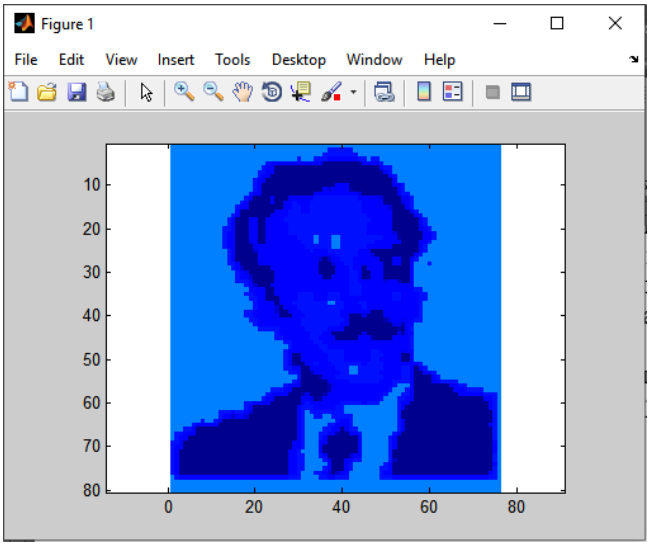


Рис. 6.3. Рисунок із однаковим стисненням зображення по всіх осях.

Той самий результат дає команда:

```
axis image;
```

але прямокутник, що охоплює зображення, у графічному вікні деформується під розмір зображення (рис. 6.4).

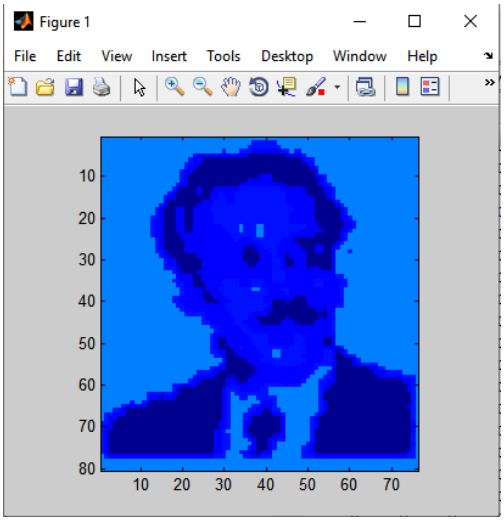


Рис. 6.4. Рисунок у графічному вікні по розміру зображення.

Кольори точок беруться із поточної карти кольорів. Карта кольорів (палітра) – це матриця, яка містить 3 стовпці і кілька (може бути багато) рядків. Наприклад, команда `gray(5)` створює наступну матрицю, яка може бути використана як палітра кольорів

```
ans =
         0         0         0
    0.2500    0.2500    0.2500
    0.5000    0.5000    0.5000
    0.7500    0.7500    0.7500
    1.0000    1.0000    1.0000
```

Карта кольорів *colormap* є m×3 матрицею дійсних чисел діапазон зміни яких від 0 до 1. Кожен її рядок є вектором *RGB*, що визначає один колір. Рядок карти кольорів з номером k визначає k-й колір у форматі [$r(k)$ $g(k)$ $b(k)$] з відповідними інтенсивностями червоного $r(k)$, зеленого $g(k)$ та синього $b(k)$ кольорів.

Прочитана з файлу матриця *A* містить числа від 0 до 15, для їх відображення задамо чорно-білу палітру з 16 відтінками сірого. Команда:

```
colormap(gray(16));
```

перетворює рис. 6.4 до такого вигляду (рис. 6.5).

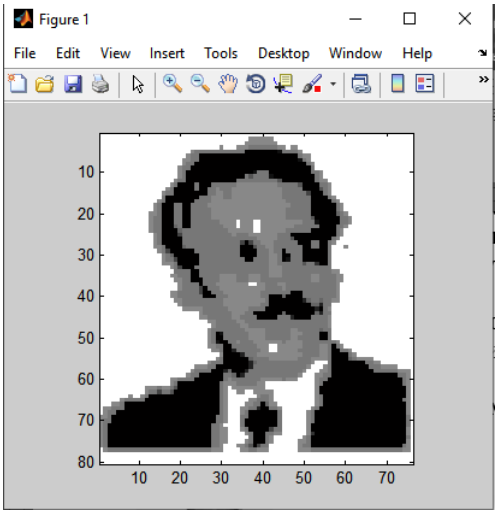


Рис. 6.5. Рисунок у чорно-білій палітрі з 16 відтінками сірого.

Зверніть увагу на тип елементів матриці *A* у вікні *Workspace* – *uint*8 (беззнакові, цілі 8 бітові, що можуть змінюватися в діапазоні від 0 до 255). Якщо виконати команду

```
B=A/15;
```

то операція поділу дасть тільки 0 або 1, а тип елементів матриці *B* не зміниться (розділити ціле на ціле дає ціле). Це дозволяє створити чорно-білий рис. 6.6. Для цього створимо палітру з двох кольорів – чорного (йому відповідає рядок 0, 0, 0) та білого (йому відповідає рядок 1, 1, 1).

```
% матриця палітри (два кольори – чорний та білий)
cm=[0 0 0;1 1 1];
image(B);
axis image;
colormap(cm);
```

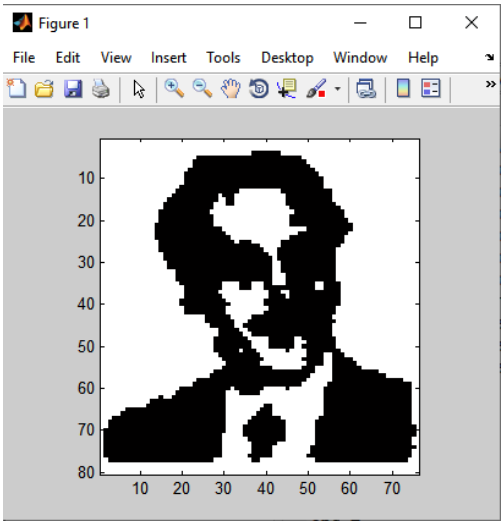


Рис. 6.6. Рисунок у чорно-білій палітрі.

Команда

```
colormap('default')
```

встановлює карту кольорів за замовчуванням.

Графічний файл може бути монохромним. Наприклад чорно-біла картинка (рис. 6.7.) розміром 32×32 пікселя – PeterMono1.bmp.

Рис. 6.7. Монохромний рисунок розміром 32×32 пікселя.

Команда

```
P=imread('PeterMono1.bmp')
```

прочитала та надрукувала матрицю 32×32 пікселя, складену тільки з нулів або одиниць. Команди:

```
image(P);
```

```
axis image;
colormap('default');
```

намалюють дуже темну картинку (рис. 6.8).

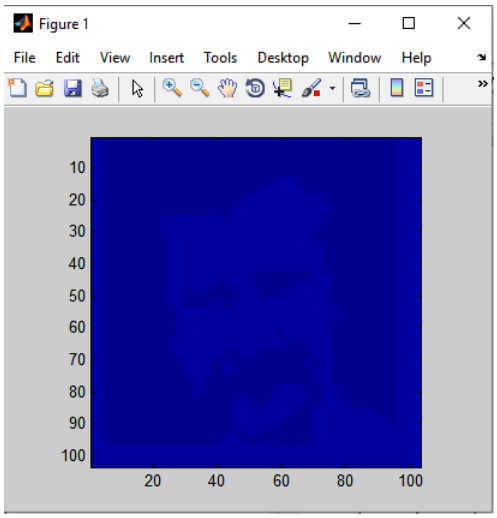


Рис. 6.8. Відображення монохромого рисунку у графічному вікні.

Але команди

```
cm=[0 0 0;1 1 1];
colormap(cm);
```

перетворять картинку на вигляд, який показаний на рис. 6.9.

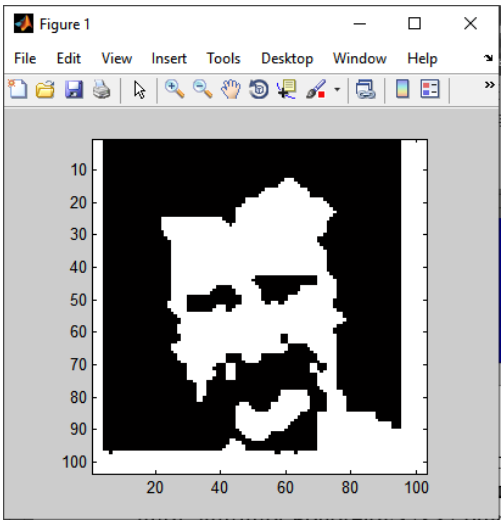


Рис. 6.9. Перетворений монохромний рисунок.

Щоб дізнатися формат графічного файлу, можна використовувати команду:

```
info = imfinfo('foto32x32.bmp')
```

Вона повертає структуру з великою кількістю полів, що містить різноманітну інформацію про графічний файл. Тут ми наводимо деякі поля цієї структури

```
info =
               Filename: 'C:\Users\Downloads\foto32x32.bmp'
            FileModDate: '04-Cep-2024 17:06:20'
               FileSize: 3318
                 Format: 'bmp'
          FormatVersion: 'Version 3 (Microsoft Windows 3.x)'
                  Width: 76
                 Height: 80
               BitDepth: 4
              ColorType: 'indexed'
        FormatSignature: 'BM'
     NumColormapEntries: 16
               Colormap: [16x3 double]
                RedMask: []
              GreenMask: []
               BlueMask: []
        ImageDataOffset: 118
      BitmapHeaderSize: 40
             NumPlanes: 1
       CompressionType: 'none'
            BitmapSize: 3200
         HorzResolution: 0
         VertResolution: 0
          NumColorsUsed: 0
     NumImportantColors: 0
```

Команда [*X*, *map*] = *imread*(*filename*) читає графічний файл у матрицю *X*, а його карту кольорів у матрицю *map*, елементи якої масштабуються до діапазону [0, 1].

```
[A,m] = imread('foto32x32.bmp');
image(A); colormap(m);
```

Команда:

```
[A,map,alpha] = imread(...)
```

у третьому параметрі повертає маску (матрицю з логічних нулів та одиниць), яка використовується для визначення інформації про прозорість точок образу (які точки образу відображати на екрані, а які ні; не відображати ті, для яких елементи *C* дорівнюють 1). Наприклад, цей параметр корисний для рисування зображення файлу іконки (рис. 6.10).

```
[A,B,C]=imread('Peter1.ico'); % A містить числа від 0 до 255.
image(A);
axis image;  % лівий рисунок
colormap(B); % середній рисунок
```

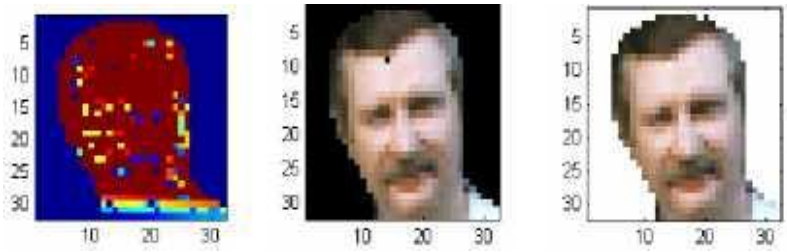


Рис. 6.10. Зображення файлу іконки.

Тепер зробимо тло рисунку білим.

```
B2=B;
B2(256,:)=[1 1 1]; % останній колір палітри зробити білим
%створимо матрицю з елементами 255
D = ones(size(A))*(length(B2)-1);
D(C == 0) = A(C == 0);
image(uint8(D)),
colormap(B2); % рисунок праворуч
axis image;
```

Пояснимо роботу рядка `D(C==0)=A(C==0)`. Якщо подивитися у вікні *Workspace*, то видно, що масив *C* є логічним. І при цьому він має такий самий розмір, як і масив *A*. Тут використовується логічне індексування у формі:

```
ім'я_матриці (логічний масив)
```

де логічний масив повинен мати той самий розмір, як і матриця. Результатом цієї операції логічного індексування є вектор, складений з елементів вихідної матриці, котрим у логічному масиві відповідні елементи дорівнюють логічній одиниці. Цей вектор може стояти як у правій частині оператора присвоєння, так і у лівій. Спочатку ми створили double масив *D* з чисел 255, а потім створили вектор однакової довжини *D*(*C*==0) і *A*(*C*==0). Другий вектор містить лише елементи матриці *A*, котрим у матриці *C* елементи дорівнюють нулю. Це означає, що відповідні точки образу мають бути намальовані. Значення цього вектора *A*(*C*==0) надаються елементам вектора *D*(*C*==0), тобто відповідним елементам матриці *D*. Якщо матриці *C* елемент був 1, то відповідний елемент матриці *D* змінюється, тобто залишається рівним 255. В результаті елементи матриці *A*, що відповідають точкам, що відображаються, скопіювалися в матрицю *D*, а відповідні точкам, що не відображаються, залишилися в матриці *D* рівними 255. Крім того, нам потрібно замінити останній (256-й) рядок матриці палітри, щоб вона відповідала білому кольору. Оскільки функція *image* відображає лише матриці з типом елементів *uint*8 (діапазон зміни чисел від 0 до 255), нам також знадобилося перетворення типу дійсної матриці *D* в тип *uint*8.

Команда *image*(*x, y, C*), де *x* та *y* є двоелементними векторами, які визначають діапазон зміни координат, рисує такий самий образ, як і команда *image*(*C*). Це буває корисно, наприклад, якщо ви бажаєте, щоб мітки осей відповідали реальним фізичним координатам графічного образу (рис. 6.11).

```
A = magic(5);
x = [-2.5 2.5];
y = [0.5 5.5];
image(x,y,A)
axis image,
colormap(jet(25))
```

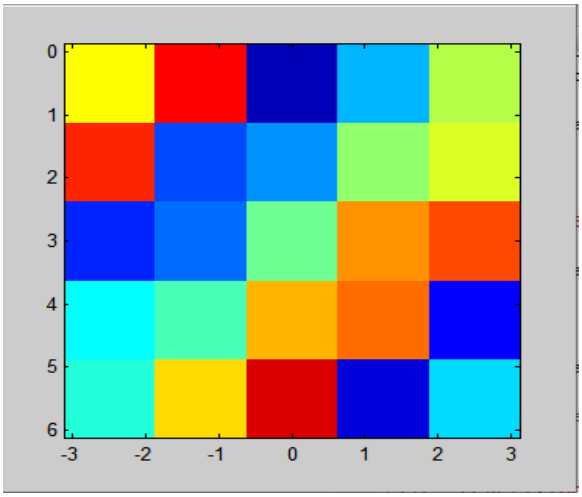


Рис. 6.11. Приклад, в якому координати графічного образу відповідають міткам осей.

Функція *imshow*(*M*) теж будує графічний образ напівтонової (тобто одноколірної з різними градаціями яскравості) матриці *M* елементи якої мають тип *double*. При цьому кожній точці матриці відповідає один піксель у графічному вікні. Варіантів використання цієї функції дуже багато. Вона є базовою графічною функцією *Image Processing Toolbox*.

```
% генерування нульової матриці розміром 4 x 4
>> I = zeros(4,4)
I =
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
>> I(2:3, 2:3) = 1 % зміна значень матриці
I =
     0     0     0     0
     0     1     1     0
     0     1     1     0
     0     0     0     0
imshow(I);   % побудова образу матриці
```

Тут одиниці у матриці *I* відповідають білим точкам рис. 6.12, а нулі – чорним. Але точок мало, тому рисунок маленький. Якби ми використовували функцію *image*, то треба було б виконати перетворення типу, наприклад, як

image(*uint*8(*I*)), але при цьому образ матриці був побудований за розміром графічного вікна.

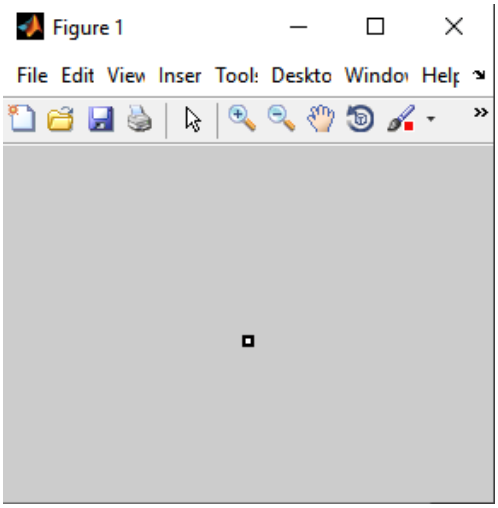
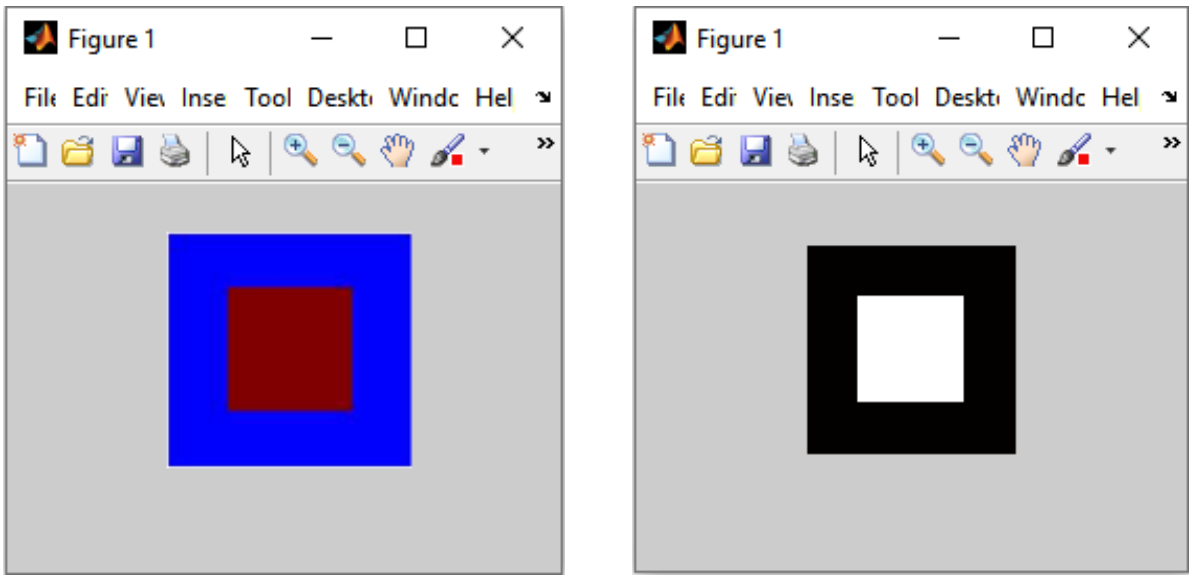


Рис. 6.12. Графічний образ напівтонової матриці.

Збільшимо розмір рис. 6.12, збільшивши розмір матриці (рис. 6.13):

```
I = zeros(100,100);
I(25:75, 25:75) = 1;
colormap('default');
imshow(I);    % рисунок 6.13, а
colormap(hot); % рисунок 6.13, б
```



а *б*

Рис. 6.13. Графічні образи, побудовані по матриці збільшеного розміру.

Слово *hot* є ім'ям палітри, імена інших існуючих палітр можна знайти у довідковій системі.

Ось ще приклад використання функції *imshow* (рис. 6.14).

```
moon = imread('moon.tif');
imshow(moon);
```

або можна одразу передати ім'я файлу

```
imshow('moon.tif');
imshow(I,[low high])
```

малює образ *I* у сірих відтінках з діапазоном зміни значень *I* рівним [*low high*].

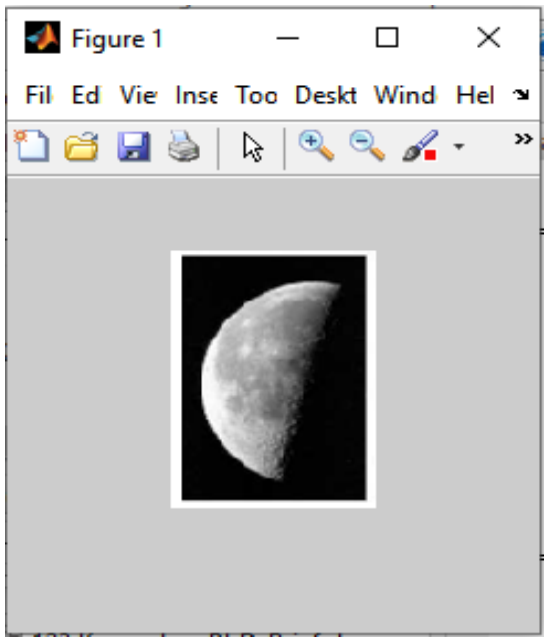


Рис. 6.14. Виведення зображення у графічному вікні.

Значення *low* та менші за нього малюються чорним кольором, значення *high* та більші за нього малюються білим. Значення між ними – малюються проміжними відтінками сірого, використовуючи рівні сірого, що визначаються за замовчуванням. Якщо використовується порожня матриця [] замість [*low high*], то *imshow* використовує [*min*(*I*(:)) *max*(*I*(:))], тобто мінімальне значення *I* відображається чорним, а максимальне – білим кольорами.

Вгадувати палітру чи використовувати палітру графічного файлу не завжди зручно. Тому в MATLAB є функція *imagesc,* яка масштабує матрицю під поточну палітру, а потім малює її образ.

Команда *imagesc*(*A*) відображає матрицю як графічний образ. Кожен елемент матриці відповідає прямокутній ділянці образу. Значення елементів є індексами рядків поточної матриці палітри, а трійка чисел цього рядка у форматі *RGB* визначає колір такого прямокутника. Команда *imagesc*(*x, y, A*) відображає матрицю *A* у вигляді набору кольорів (графічного образу) на координатній сітці з діапазоном, що визначається векторами *x* і *y*.

Іноді бажано відобразити матрицю як поверхню над прямокутною областю. Це можна зробити за допомогою функції *surface*(*Z*), де матриця *Z* містить дійсні числа, які інтерпретуватимуться як висота поверхні над площиною *XY*. Але матриці, прочитані із графічного файлу, містили лише цілі числа. Тут може знадобитися зміна типу даних. Наприклад, команди:

```
A = imread('foto32x32.bmp');
C = double(A)/16;
```

створять матрицю *double*(*A*), кожен елемент якої буде дійсним числом, які після поділу на 16 створять матрицю *C* з дійсними елементами в діапазоні від 0 до 1 (рис. 6.15).

```
C=double(A)/16;
surface(C); % рисує поверхню матриці з double елементами
surface(C*10);
colormap(hot);
```

Аналогічно працює команда *surf*(*Z*), яка створює 3-х мірний графік поверхні з координат точок, заданих у матриці *Z,* використовуючи *x*=1:*n* і *y*=1:*m*, де [*m,n*]=*size*(*Z*). Висоти точок поверхні *Z* є дійсними числами, а колір поверхні пропорційний висоті точок (рис. 6.16).

```
A = imread('foto32x32.bmp')
C=double(A)/16;
surfc(C);
colormap(gray(17));
```

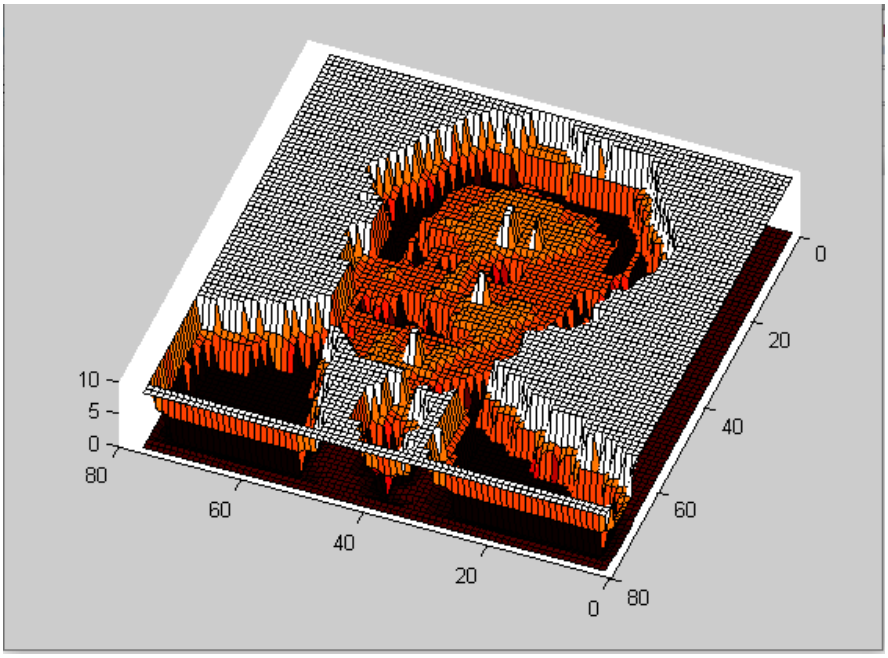


Рис. 6.15. Зображення побудоване за допомогою функції *surface*.

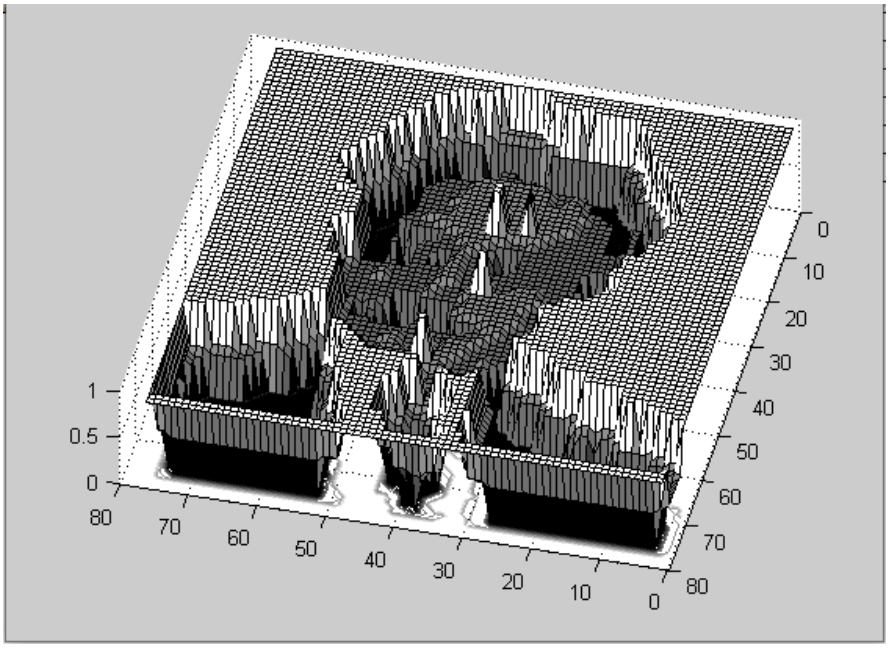


Рис. 6.16. Зображення побудоване за допомогою функції *surf*.

6.2. Математична постановка задачі комп'ютерної томографії

У 1917 році австрійський математик І. Радон отримав формулу звернення для відображення, що зіставляє функції f на площині з функцією R на безлічі всіх прямих на площині, що дорівнює інтегралам від f вздовж усіх прямих. Цей результат призвів до великих наслідків як у самій математиці, так і в її додатках. Найбільш важливим виявився додаток перетворення Радона до томографії. Термін "томографія" походить від двох грецьких слів: томос – перетин та графос – пишу і означає пошарове дослідження структури об'єктів. Існує кілька видів томографії: рентгенівська, електронно-променева, магнітно-резонансна, позитронно-емісійна, ультразвукова, сейсмічна, оптична томографія та ін. Але суть усіх видів томографії єдина: за набором «зображень» випромінювання, що пройшло крізь тіло, потрібно відновити внутрішню будову тіла.

Метод комп'ютерної томографії у 1961 р. запропонував американський нейрорентгенолог Вільям Ольдендорф, а у 1963 р. математик А. Кормак (США) провів лабораторні експерименти з рентгенівської томографії і показав здійсненність реконструкції зображення. У 1973 р. інженер-дослідник Г. Хаунсфілд (Великобританія) розробив першу комерційну систему – сканер головного мозку. 1979 року Г. Хаунсфілду та А. Кормаку за видатний внесок у розвиток комп'ютерної томографії було присуджено Нобелівську премію в галузі медицини.

Під час дослідження внутрішньої структури об'єкта його просвічують випромінюванням. Просвічуючи тіло з одного напрямку, одержують плоске (двовимірне) тіньове зображення тривимірного тіла. Просвічуючи тіло з іншого напрямку, отримують інше тіньове зображення тіла та додаткову інформацію про його внутрішню структуру. Просвічуючи тіло ще з одного напрямку, одержують нову інформацію і т.д. Маючи велику кількість проекційних знімків з різних напрямків, можна, з достатньою мірою точності, відновити внутрішню структуру

об'єкта, а точніше функцію щільності поглинання випромінювання.

Завданням томографії є відновлення тривимірної функції $д(x, y)$, щільності поглинання випромінювання. У такій постановці завдання дуже складне і в класичній томографії тривимірний об'єкт представляють як набір тонких зрізів. Усередині кожного зрізу щільність μ вважають функцією лише двох змінних. При дослідженні систему «джерело-приймач» влаштовують таким чином, щоб реєструвати дані на променях, що лежать у площині зрізу. Схема сканування одного шару представлена наступному рис. 6.17.

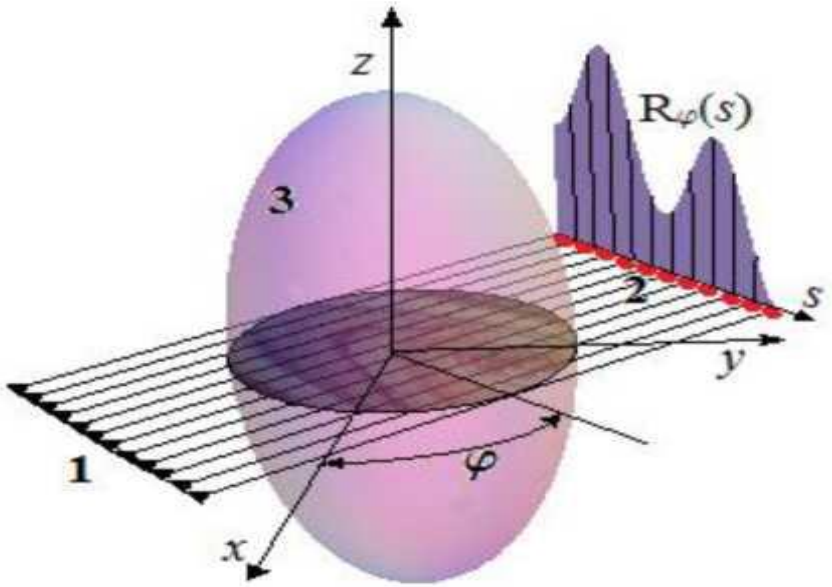


Рис. 6.17. Зондування тривимірного об'єкта
(1 – джерела; 2 – детектори; 3 – об'єкт)

Детектори реєструють дані і отримана за ними функція R залежить від однієї змінної s (при фіксованому напрямку зондування, що визначається кутом φ). Відновити по одній проекції $Rφ(s)$ функцію двох змінних $μ(x, y)$ неможливо. Для того, щоб отримати набір даних достатній для відновлення, застосовують зондування об'єкта з різних напрямків, варіюючи кут φ. Повертаючи систему «джерела-детектори», одержують безліч проекцій $R(s, φ)$ шару (параметр φ позначає кут зондування), якими можна відновити двовимірну функцію $μ(x, y)$. Схема сканування шару з різних напрямків показана на рис. 6.18.

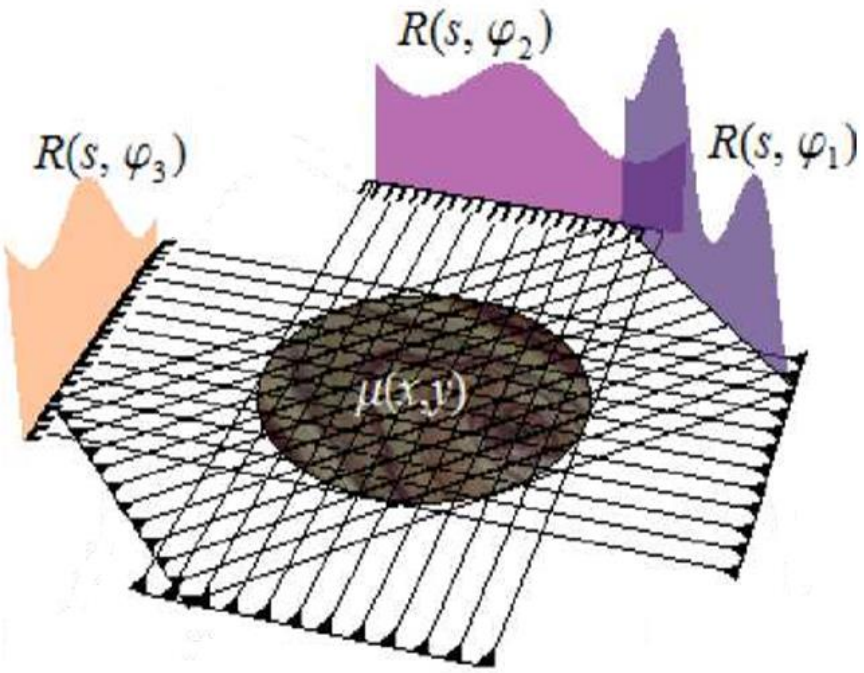


Рис. 6.18. Схема отримання проекцій $R(s, \varphi)$ одного шару.

Визначивши функцію $\mu(x, y)$ одного шару, систему «джерела-детектори» зрушують у бік осі z для отримання інформації про наступний близький шар і так далі. Потім, за двовимірними функціями $\mu_{zi}(x, y)$ у шарах z_i, де z – координата перпендикулярна до шара, отримують тривимірну функцію щільності поглинання $\mu(x, y, z)$. У цьому випадку основні проблеми виникають, щодо окремого шару, тобто при відновленні функції $\mu_z(x, y)$.

Подібний спосіб збирання даних становить основу реконструктивної томографії. На тіло, що досліджується, впливає випромінювання, що проникає всередину об'єкта. Воно взаємодіє з речовиною, що становить об'єкт, і на виході реєструється випромінювання через тіло. При обробці даних використовуються такі припущення: траєкторія променя вважається прямолінійною і виконується лінійний закон поглинання випромінювання в речовині.

У основі математичного апарату томографії лежить перетворення Радона.

6.2.1. Пряме перетворення Радону

Функція прямого перетворення Радона *radon* вимагає встановлення двох основних параметрів – вхідного зображення *I* і кутів *theta*. Вона має наступний спрощений синтаксис виклику

```
R = radon(I);
R = radon(I,theta);
```

Вихідне зображення задається у формі матриці *I*. Аргумент *theta* задає вектор кутів, для яких має виконуватись перетворення. Якщо аргумент *theta* не вказано, він приймається рівним 0:179. Значення *R*, що повертається, являє собою матрицю, в якій кожен стовпець є перетворенням Радону для кожного з кутів, що містяться у векторі *theta*. Матриця проекцій *R* має формат даних *double*.

Приклад 6.1. Створимо матрицю, що складається з одиниць та нулів та відобразимо її у графічному вікні (рис. 6.19).

```
I = zeros(100,100);
I(25:75,25:75) = 1;
imshow(I);
```

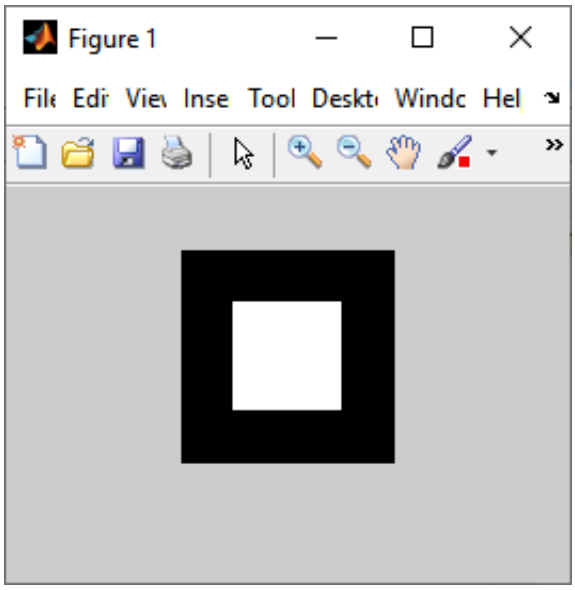


Рис. 6.19. Графічне відображення матриці.

Одиниці у матриці *I* відповідають білим точкам рисунка, а нулі – чорним. Тепер матимемо матрицю проекцій, тобто виконаємо перетворення Радону.

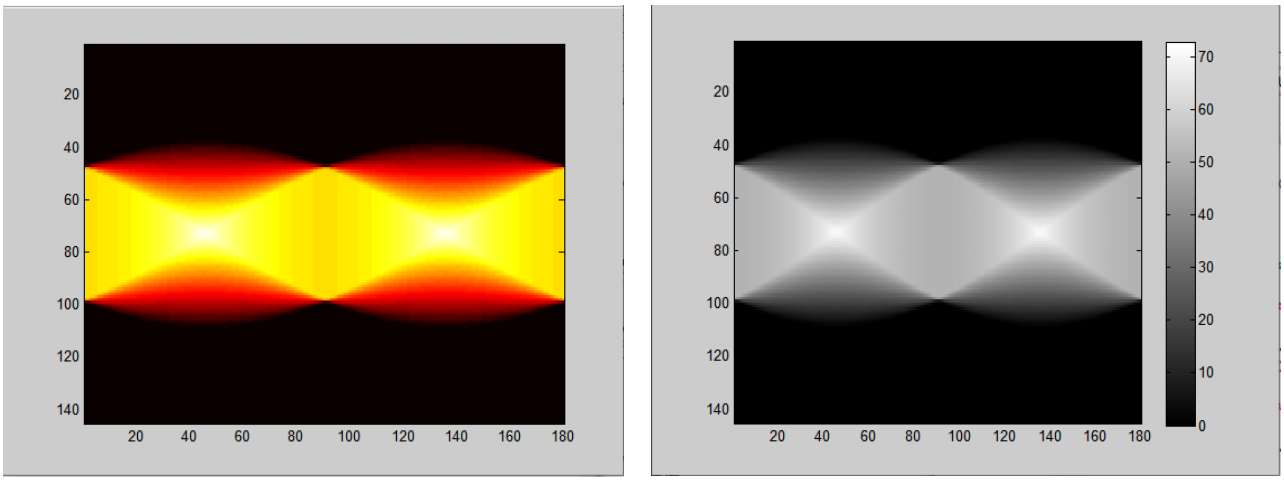
```
R=radon(I);
```

Оскільки матриця *R* містить дійсні елементи, а не типу *uint*8, перед побудовою її образу вона повинна бути масштабована під поточну палітру. Використовуємо для цього функцію *imagesc*(*R*) (рис. 6.20, *а*)

```
imagesc(R);
colormap(hot);
```

або (рис. 6.20, *б*)

```
colormap(gray(129));
colorbar;
```



а *б*

Рис. 6.20. Побудова образу матриці *R*.

Команда *colorbar* рисує у графічному вікні праворуч від образу поточну палітру кольорів. Команди:

```
[R,xp]=radon(I);
mesh(0:179,xp,R);
colormap(gray);
```

будують поверхню радонівського образу матриці *R* (рис. 6.21).

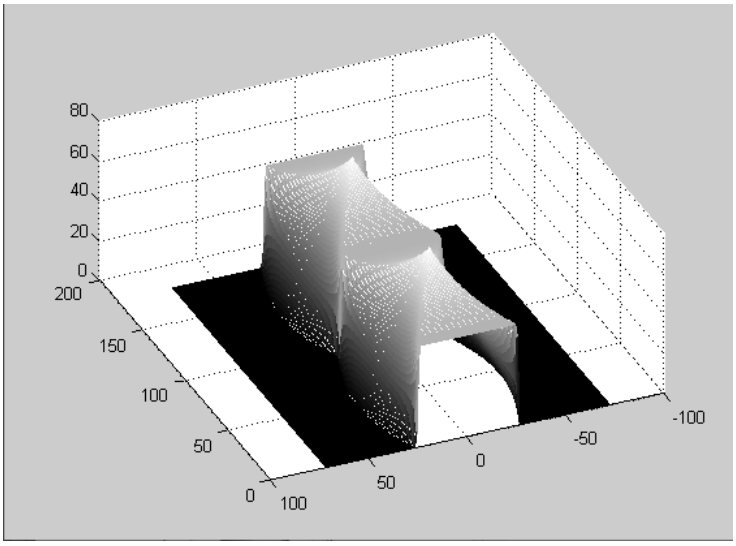


Рис. 6.21. Поверхня радонівського образу матриці *R*.

Приклад 6.2. Побудуємо перетворення Радона (ПР) характеристичної функції (ХФ) одиничного кола. Спочатку створимо матрицю, значення якої є ХФ одиничного кола. Розберемо методику на матриці невеликого розміру

```
[X,Y]=meshgrid(-1:0.5:1)
XY=X.^2+Y.^2
C=XY<=1  % створюємо логічну матрицю
Z=ones(size(C))  % одинична речова матриця
ZF=zeros(size(C))
```

Усі рядки ми не завершуємо крапкою з комою. Це призводить до виведення матриць у командне вікно. Для стислості ми тут не наводимо цих матриць. Тепер побачимо, як виділити потрібні елементи. Команда *ZF*(*C*) виділяє вектор - стовпець елементів нульової матриці *ZF*, для яких у логічній матриці значення *C* дорівнює одиниці.

```
ZF(C)' % відображаємо транспонований вектор-стовпець
0 0 0 0 0 0 0 0 0 0 0 0 0
```

Елементам нульової матриці *ZF*, що стоять на таких самих місцях, що в логічній матриці *C* стоять одиниці, надають значення матриці *Z* з аналогічних осередків (з тими ж номерами рядків і стовпців). Інші значення матриці *ZF* не змінюються. Це робить команда

```
ZF(C)=Z(C)
ZF =
     0     0     1     0     0
     0     1     1     1     0
     1     1     1     1     1
     0     1     1     1     0
     0     0     1     0     0
```

Відобразимо матрицю *ZF* (рис. 6.22) командами:

```
imagesc(ZF);
colormap(gray);
```

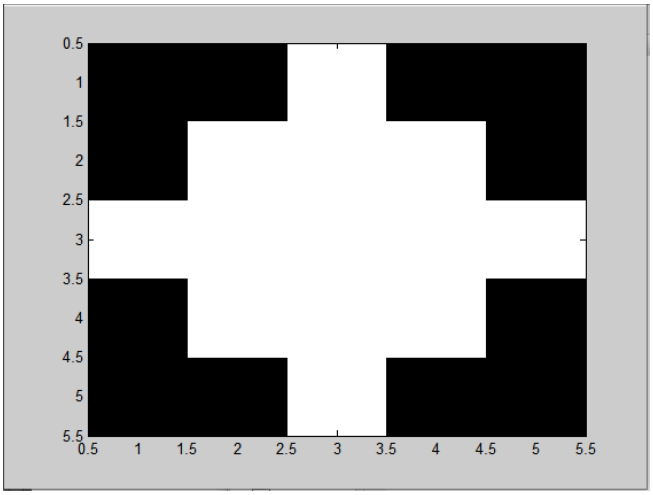
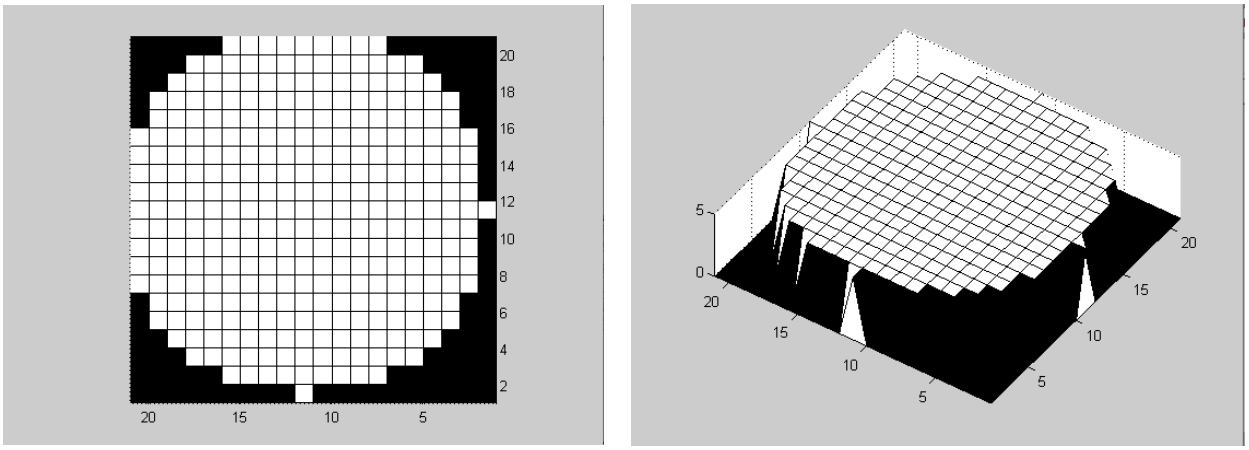


Рис. 6.22. Графічне відображення матриці *ZF*.

Далі зробимо те саме, але для матриць більшого розміру (рис. 6.23).

```
[X,Y]=meshgrid(-1:0.1:1);
XY=X.^2+Y.^2;
C=XY<=1;
Z=ones(size(C));
ZF=zeros(size(C));
ZF(C)=Z(C);
```

```
imagesc(ZF);      % рис.6.23, а
colormap(gray); axis image;
surf(ZF*5);       % рис.6.23, б
axis image; colormap(gray);
```



а *б*

Рис. 6.23. Графічне відображення матриць.

Тепер виконаємо ПР для характеристичної функції одиничного кола, використовуючи матрицю *ZF* (рис. 6.24).

```
R=radon(ZF,0:1:179);
imagesc(R);
colormap(gray);
colorbar;
```

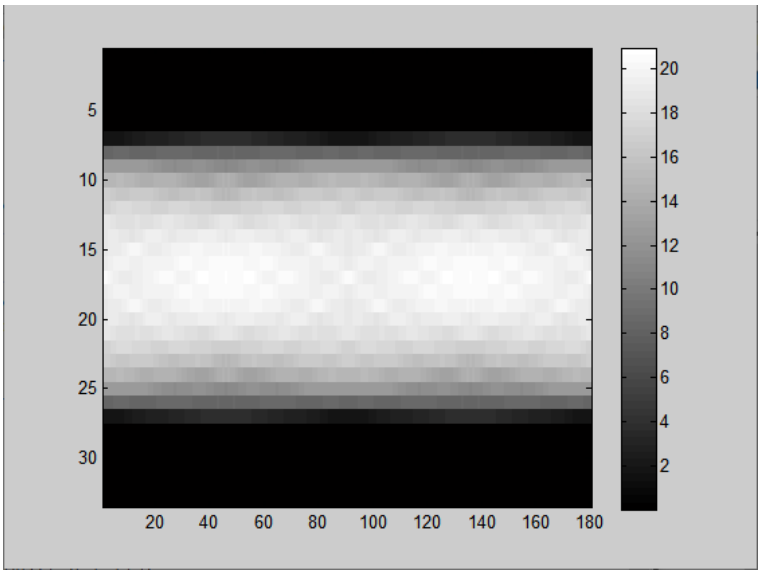


Рис. 6.24. Поверхня радонівського образу матриці *R*.

По горизонталі відкладено кути в градусах, а по вертикалі значення номерів рядків матриці *R* (їх кількість *n* визначена за умовчанням). Якщо виконати команди:

```
max(max(R))
ans =
    20.9740
```

то знайдемо максимальне значення в матриці *R*. Функція *colorbar* показала, що темним точкам образу відповідають нульові значення матриці *R*, а найсвітлішим - значення близькі до 20.

Команди:

```
[R,xp] = radon(ZF,1:179);
xp'
ans =
  Columns 1 through 10
-16   -15   -14   -13   -12   -11   -10    -9    -8    -7
  Columns 11 through 20
-6    -5    -4    -3    -2    -1     0     1     2     3
  Columns 21 through 30
 4     5     6     7     8     9    10    11    12    13
  Columns 31 through 33
14    15    16
```

показують, що обчислення виконані для 33 точок (значення *s* вздовж проекції при фіксованому куті) змінюється від –16 до +16. Крім того, зрозуміло, що початок координат *XY* зображення знаходиться в центрі матриці.

Ми розпочинали з характеристичної функції одиничного кола, яку представили матрицею над прямокутником [–1 1] × [–1 1]. Для приведення графіка проекції до реальних фізичних координат ми маємо показати діапазон зміни кута $[0, \pi]$ і радіального параметра $[-\sqrt{2}, \sqrt{2}]$ (рис. 6.25).

```
RU=uint8(R);
max(max(RU))
```

```
ans = 21
image([0 pi],[-sqrt(2) sqrt(2)],RU);
axis image;
colormap(gray(21));
```

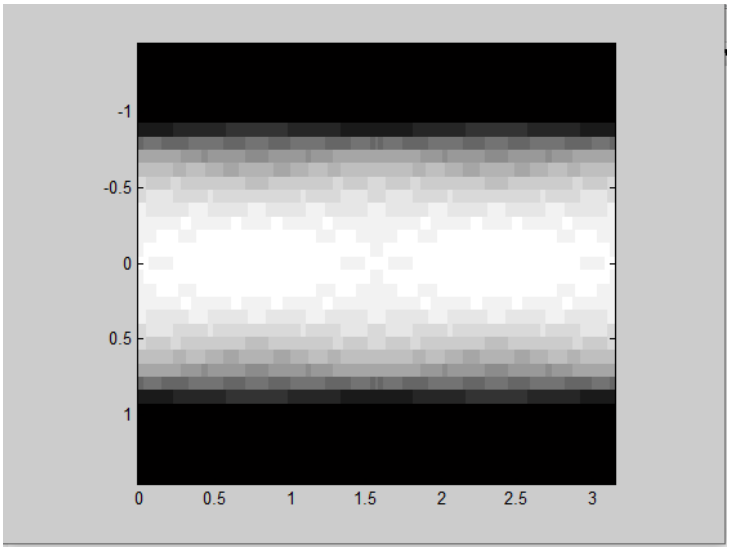


Рис. 6.25. Діапазон зміни кута і радіального параметра.

Розглянемо докладніше синтаксис функції *radon*. Він може бути одним із наступних:

```
R=radon(I, theta)
R=radon(I, theta, n)
[R, xp]=radon(...)
```

Функція *R=radon(I, theta)* виконує перетворення Радону напівтонового зображення, що задається матрицею *I*, і поміщає результат у матрицю проекцій *R*. Перетворення Радону являє собою обчислення проекцій зображення на прямі, що задаються кутами в градусах, що відраховуються проти годинникової стрілки від горизонталі. Ці кути передаються як вектор у параметрі *theta*. Якщо *theta* – скаляр, то *R* є вектор – стовпцем, що містить перетворення Радону для кута *theta*. Якщо *theta* – вектор, то *R* є матрицею, в якій кожен стовпець є перетворенням Радону для одного з кутів, що містяться у векторі *theta*. Якщо при виклику функції параметр *theta* опущений, то *theta* записуються значення кутів від 0 до 179 з кроком в 1°.

В форматі *R=radon*(*I, theta, n*) функція виконує перетворення Радону зображення *I*, значення кожної проекції обчислюється у *n* точках, і матриця *R* має *n* рядків. Якщо при виклику функції параметр *n* не заданий, то він автоматично обраховується за наступним рівнянням:

```
2*ceil(norm(size(I)-floor((size(I)-1)/2)-1))+3.
```

Якщо додатково визначити вихідний параметр *xp*[*R, xp*]=*radon*(...), то в ньому містяться значення координат у яких обчислювалися значення проекції. Значення в *k*-му рядку *R* відповідають координаті *xp*(*k*) .

Матриця проекцій *R* може розглядатися як напівтонове зображення та має формат представлення даних *double*.

Нагадаємо, що ПР двовимірної функції *f*(*x, y*) на вісь *s* являє собою лінійний інтеграл $R(s,\varphi)=\int_{-\infty}^{\infty} f(s\cos\varphi - t\sin\varphi, s\sin\varphi - t\cos\varphi)dt,$ де осі s та t задаються поворотом на кут φ проти годинникової стрілки (рис. 6.26):

$$\begin{bmatrix} s \\ t \end{bmatrix} = \begin{bmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

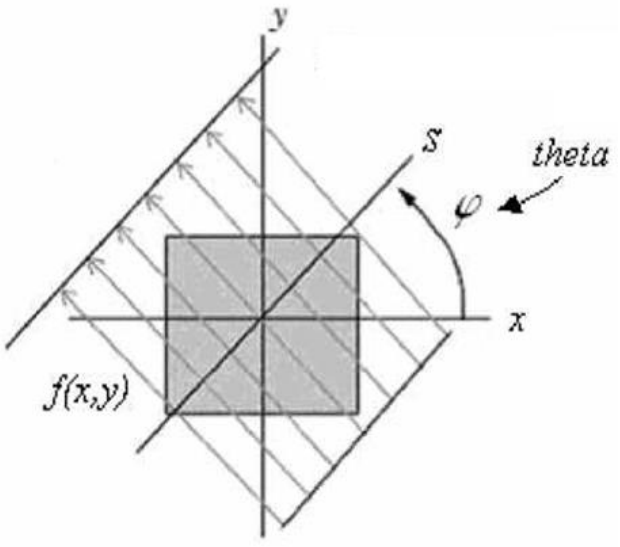


Рис. 6.26. Поворот на кут φ.

Вихідне напівтонове зображення *I* сприймається як двовимірна функція. Початок координат у системі відповідає центральному пікселю зображення *I* піксельної системи координат. Центральний піксель *I* можна визначити згідно з виразом *floor*((*size*(*I*) + l)/2).

Розглянемо, як у перетвореннях Радона обчислюються проекції (рис. 6.27). Визначимо проекції квадрату під кутом 0°, 15°, 30° и 45° из прикладу 6.1.

```
I = zeros(100,100);
I(25:75,25:75)= 1;
imshow(I);
[R,xp] = radon(I,[0 15 30 45]);
% Побудова графіків для кутів 0, 15, 30 та 45 градусів .
figure; plot(xp,R(:,1),'LineWidth',2);
title('R_{0^o}(s)');grid on;
figure; plot(xp,R(:,2),'LineWidth',2);
title('R_{15^o}(s)');grid on;
figure; plot(xp,R(:,3),'LineWidth',2);
title('R_{30^o}(s)');grid on;
figure; plot(xp,R(:,4),'LineWidth',2);
title('R_{45^o}(s)');grid on;
```

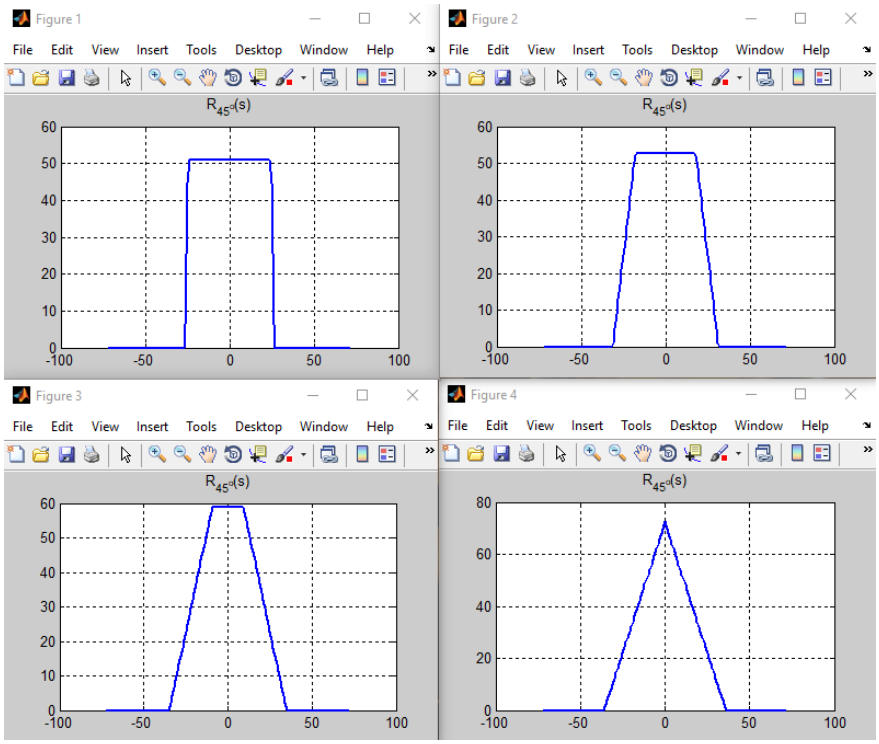


Рис. 6.27. Проекції квадрату під кутом 0°, 15°, 30° и 45°.

Усі графіки можна вивести в одному вікні (рис. 6.28):

```
plot(xp,R(:,1),xp,R(:,2),xp,R(:,3),xp,R(:,4),
'LineWidth',2);
grid on;
```

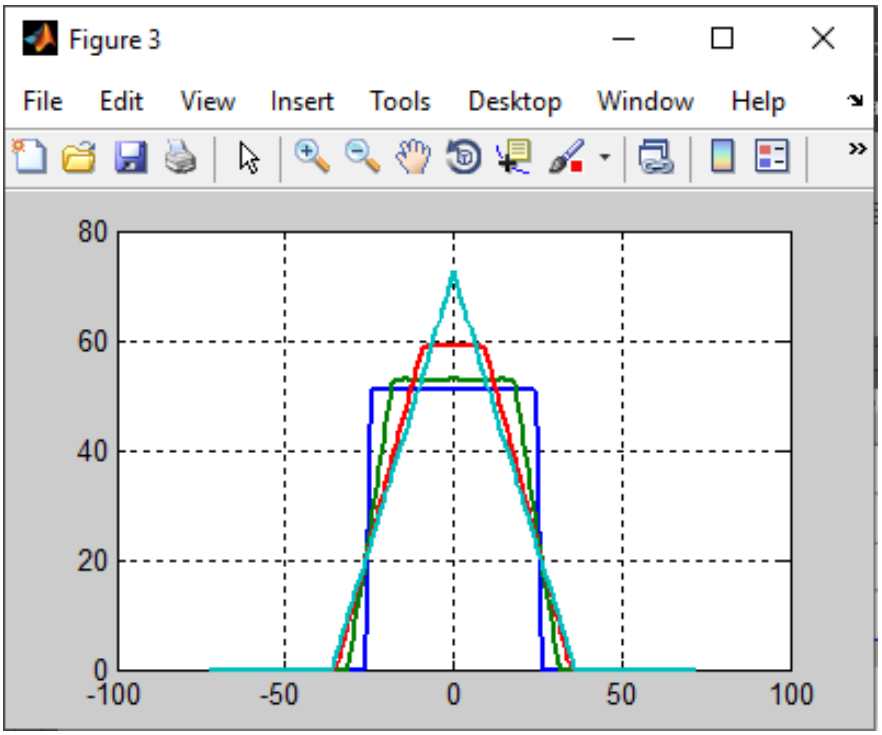


Рис. 6.28. Проекції квадрату під кутом 0°, 15°, 30° и 45° в одному вікні.

Перетворення Радона при великій кількості кутів часто будується у вигляді зображення матриці (рис. 6.29).

```
theta = 0:179;
[R,xp] = radon(I,theta);
imagesc(theta,xp,R);
title('R_{\psi} (S)');
xlabel('\psi (degrees)');
ylabel('S');
set(gca,'XTick',0:20:180);
colormap(hot);
colorbar;
```

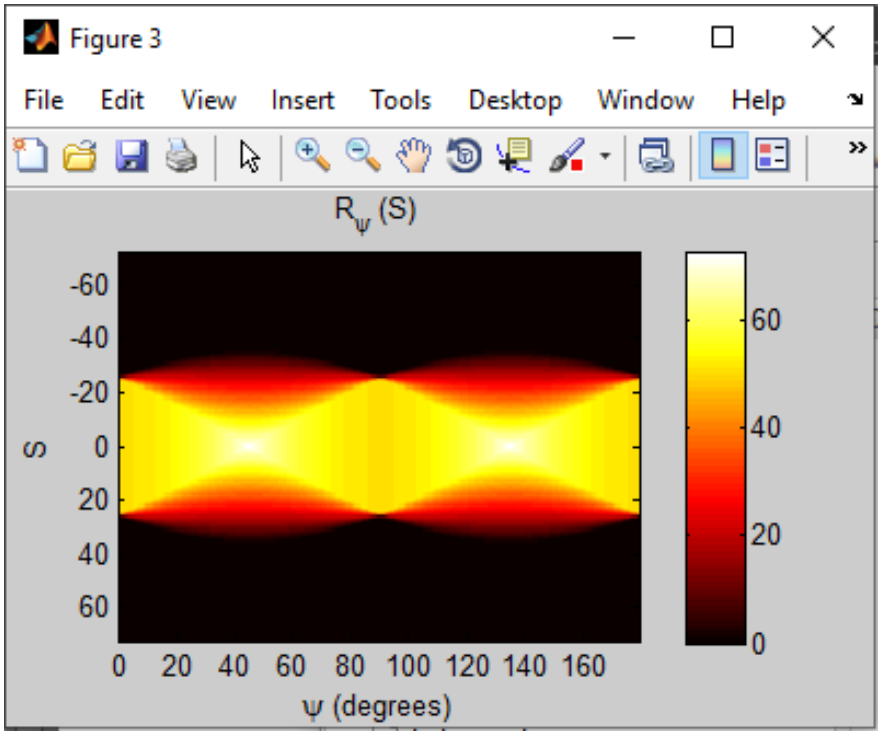


Рис. 6.29. Перетворення Радона у вигляді зображення матриці.

Приклад 6.3. Сформуємо невелике тестове зображення, що складається з трьох прямокутників різної яскравості (рис. 6.30), і побудуємо перетворення Радона (рис. 6.32).

```
% Приклад демонструє перетворення Радону та формування
% зображення.
I=zeros(100, 100);
I(20:80,20:80)=0.4;
I(30:70,45:55)=0.6;
I(45:55,45:55)=1;
imshow(I);
[R, xp]=radon(I, 0:179); % Пряме перетворення Радону.
% Виведення на екран проекцій на осі X, Y та під кутом в
45°.
```

Проекції зображення (рис. 6.30) на ось *X* і на осі з кутами 15°, 30°, 45°, 75° та на ось *Y* наведено відповідно на рис. 6.31. По горизонталі на всіх графіках відкладено радіальний параметр *s*.

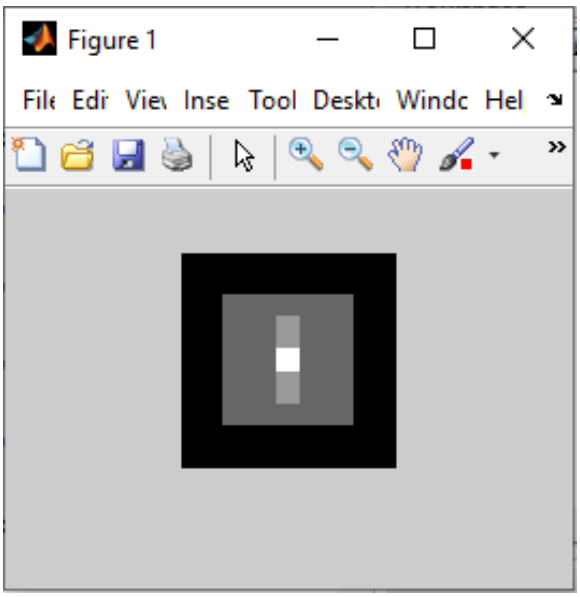


Рис. 6.30. Зображення трьох прямокутників різної яскравості.

```
figure, plot(xp,R(:,1),'LineWidth',2);
title('0 (градусів)');
grid on;
figure, plot(xp,R(:,16),'LineWidth',2);
title('15 (градусів)');
grid on;
figure, plot(xp,R(:,31),'LineWidth',2);
title('30 (градусів)');
grid on;
figure, plot(xp,R(:,46),'LineWidth',2);
title('45 (градусів)');
grid on;
figure, plot(xp,R(:,76),'LineWidth',2);
title('75 (градусів)');
grid on;
figure, plot(xp,R(:,91),'LineWidth',2);
title('90 (градусів)');
grid on;


figure, imshow( R,[],'XData',0:179,'YData', xp);
```

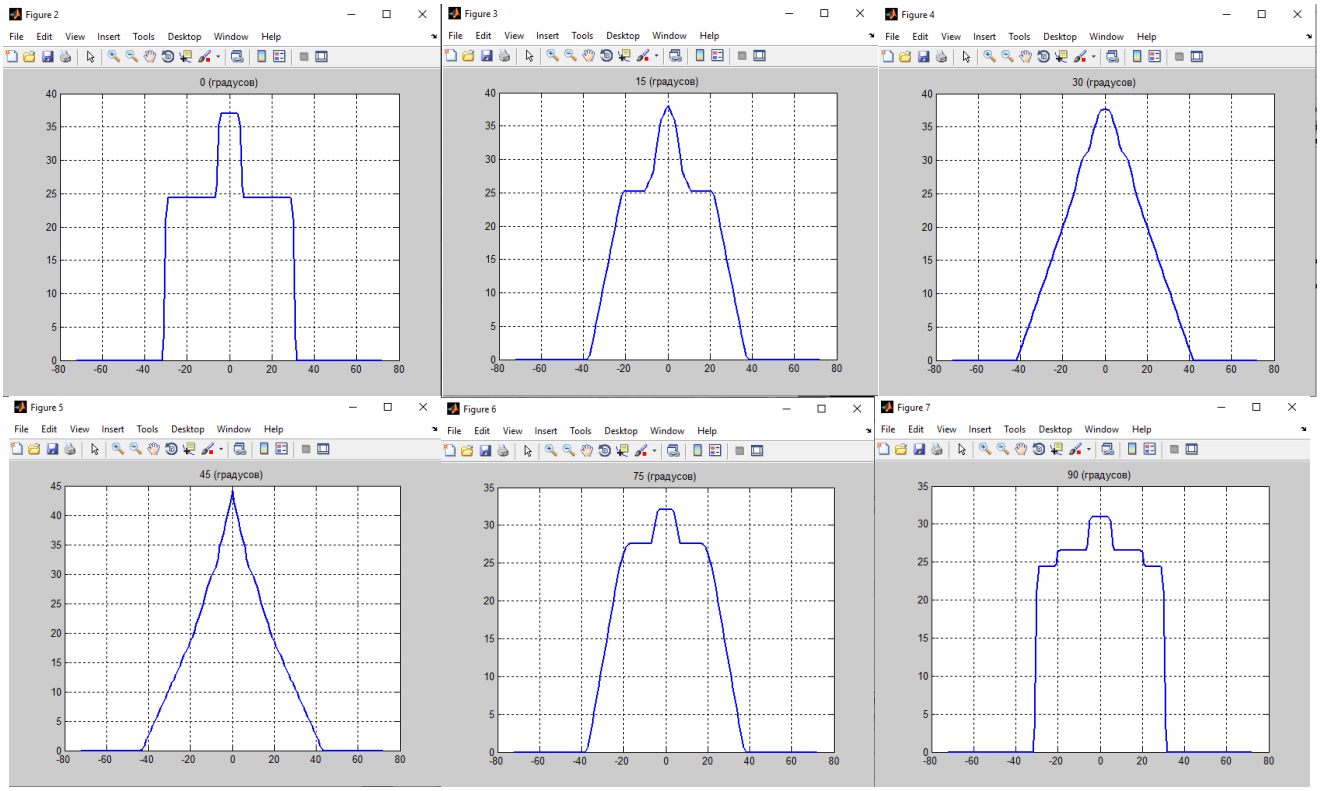


Рис. 6.31. Проекції зображення з кутами 0°,15°, 30°, 45°, 75°, 90°.

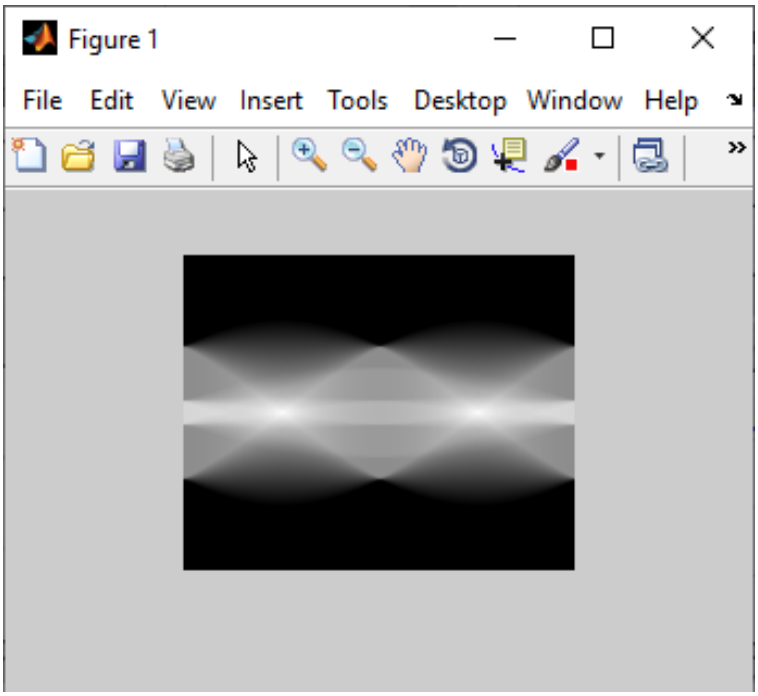


Рис. 6.32. Перетворення Радона.

6.2.2. Зворотне перетворення Радону

Функція прямого перетворення Радона *radon* для виконання вимагає встановлення двох основних параметрів – вихідного зображення *I* і кутів *theta*.

```
R = radon (I, theta);
```

Функція *iradon* реалізує відновлення зображення на основі проекційних даних, які містяться в масиві *R*.

```
IR = iradon (R, theta);
```

Параметр *theta* описує кути (у градусах), під якими отримано кожну проекцію

$$R(s,\varphi)=\begin{cases}2\sqrt{1-s^2}, |s|\le 1,\\ 0, |s|>1.\end{cases}$$

Як він виглядає, ми вже знаємо – це характеристична функція одиничного кола

$$f(x,y)=\begin{cases}1, x_0^2+y_0^2\le 1,\\ 0, x_0^2+y_0^2>1.\end{cases}$$

Перевіримо це за допомогою функції *iradon*. Створимо матрицю *R*, що відповідає функції *R* (*s*, φ) (рис. 6.33).

```
[Ph,S]=meshgrid(0:180, -2:0.1:2);
C=abs(S)<=1;
RSPh=zeros(size(C));
S2=S.^2;
RSPh(C)=2.*sqrt(1-S2(C).^2);
mesh(S,Ph,RSPh);
```

Тепер виконаємо зворотне перетворення Радону (рис. 6.34).

```
I=iradon(RSPh,0:180);
imagesc(I);                    % рис. 6.34, а
colormap(gray); mesh(I);       % рис. 6.34, б
```

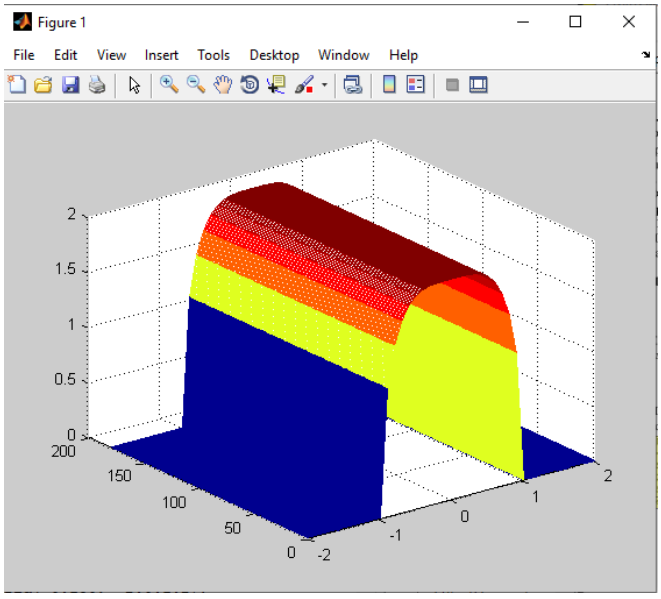
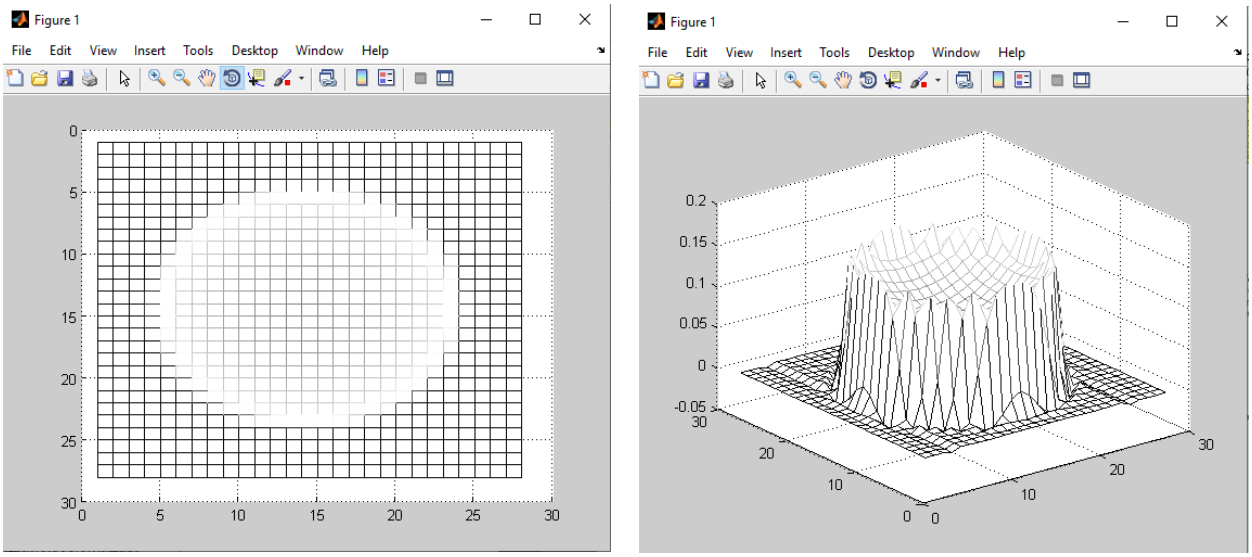


Рис. 6.33. Перетворення Радона.



а *б*

Рис. 6.34. Зворотне перетворення Радону.

Результат схожий на характеристичну функцію кола, але недостатня точність і, крім того, потрібне приведення до фізичних координат.

Розглянемо докладніше синтаксис функції *iradon*, який може бути одним із наступних:

```
I=iradon(P, theta)
I=iradon(P, theta, interp, filter, d, n)
[I, h]=iradon(...)
```

Функція *I=iradon*(*P, theta*) виконує реконструкцію зображення *I* за його проекційними даними, які містяться в масиві *P*. Рядки *P* є даними паралельно-променевих проекцій. У функції *iradon* центр обертання є центральною точкою проекцій і визначається за виразом *ceil*(*size*(*P*, 1)/2). Параметр *theta* описує кути (у градусах), під якими отримано кожну проекцію. Цей параметр може бути вектором, що містить кути або скаляр, що описує кут між проекціями *D_theta*. Коли *theta* представлений вектором, він повинен містити кути з рівномірною розбивкою. Коли *theta* представлений скаляром, що описує *D_theta*, тоді проекції беруть згідно з кутами *theta* = *m*×*D_theta*, де *m* = 0, 1, 2, ..., s*ize*(*P*, 2) – 1. Коли вихідними даними є порожня матриця ([]), тоді параметр *D_theta* за умовчанням дорівнює 180/*size*(*P*, 2).

Функція *iradon* використовує алгоритм фільтрації зворотних проекцій до виконання інверсного перетворення Радона. Фільтр проектується безпосередньо в частотній області та множиться на функцію перетворення Фур'є проекцій. Для прискорення обчислень функції перетворення Фур'є проводять спеціальні перетворення над проекціями.

Функція *I = iradon*(*P, theta, interp, filter, d, n*) містить опис параметрів, що використовуються при інверсних перетвореннях Фур'є. Існує також можливість точного визначення деяких комбінацій останніх чотирьох аргументів. Для пропущених параметрів функція *iradon* за замовчуванням встановлює деякі значення.

Параметр *interp* визначає тип інтерполяції, який використовується в *backprojection*. Наведемо список доступних опцій:

'nearest' – інтерполяція по найближчій околиці;

'linear' – лінійна інтерполяція (за умовчанням);

'spline' – сплайнова інтерполяція.

Параметр *filter* описує, який тип фільтра використовується для частотної фільтрації. Параметр *filter* є рядком, в якому описані кілька стандартних фільтрів:

'Ram-Lak' – усічений фільтр Рама-Лака (встановлюється за умовчанням). Частотний відгук цього фільтра дорівнює | *f* |. Одним із недоліків фільтра Рама-Лака є те, що він чутливий до шуму на проекціях. Тому він використовується у комбінаціях з іншими фільтрами.

'Shepp-Logan' – фільтр Шепа-Логана, помножений на фільтр Рама-Лака через фазову функцію.

'Cosine' – косинусний фільтр, помножений на фільтр Рама-Лака через косинусну функцію.

'Hamming' – фільтр Хеммінга, помножений на фільтр Рама-Лака через вікно Хеммінга.

'Hann' – фільтр Ханна, помножений на фільтр Рама-Лака через вікно Ханна.

Параметр *d* являє собою скаляр в діапазоні (0, 1] і служить для модифікації фільтра в плані масштабування частотної осі. За замовчуванням він дорівнює 1. Коли *d* менше 1, тоді фільтр стискає частотний діапазон до [0, *d*], нормує частоти; всі частоти, які більші за значення *d*, прирівнюються до 0.

Параметр *n* є скаляр, що описує число рядків і стовпців у відновленому зображенні. Коли параметр *n* не описаний, розміри визначаються виходячи з довжини проекцій:

```
n=2*floor(size(P, 1)/(2*sqrt(2))).
```

Після визначення параметра *n*, функція *iradon* відновлює зображення, не змінюючи масштаб даних. Коли проекції були обчислені за допомогою функції *radon*, розміри відновленого і вихідного зображень можуть не збігатися.

Функція [*I, h*]=*iradon*(...) повертає частотний відгук фільтра вектор *h*.

Вимоги до вихідних даних: усі вихідні та результуючі аргументи мають бути представлені у форматі *double*.

6.3. Приклад реалізації комп'ютерної томографії у MATLAB

Потрібно візуалізувати процес томографії з прикладу звичайної тривимірної функції. На рис. 6.35 показано як виглядатиме вікно програми в момент виконання.

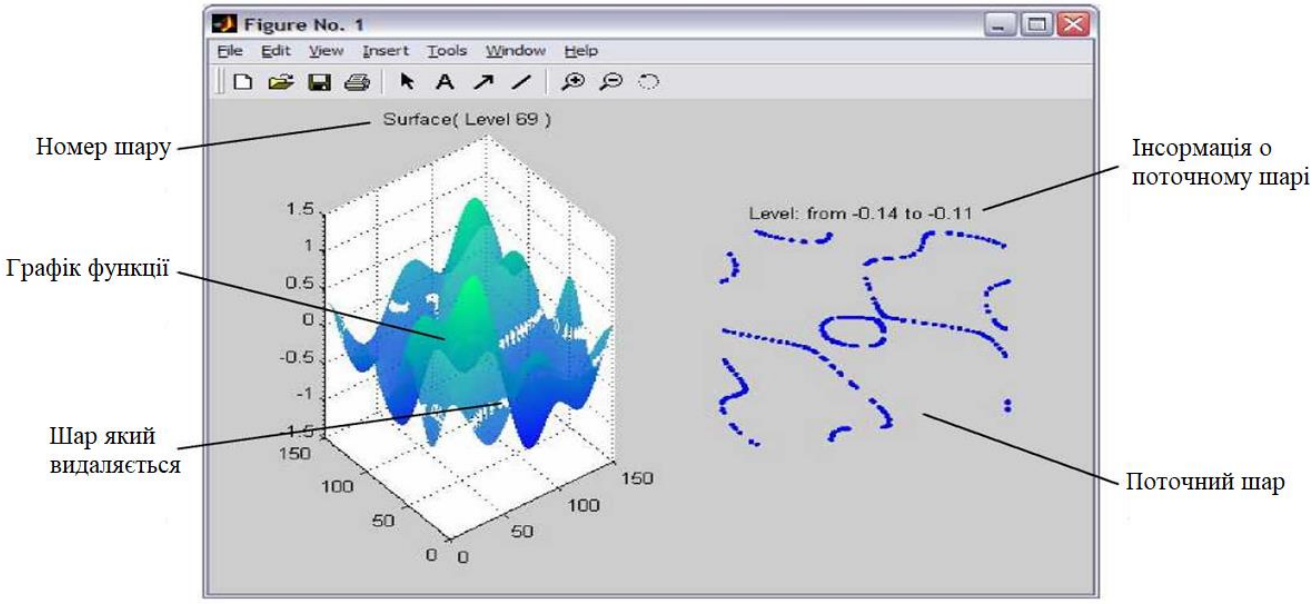


Рис. 6.35. Вікно програми.

Вікно програми складається із двох частин. Зліва розташовується задана функція, праворуч відображається поточний шар. Обидва графіки мають заголовки, що дають інформацію про поточний шар.

Спочатку розв'яжемо завдання без правої частини: навчимося рисувати поверхні, розберемося з вирізуванням шару, зробимо все це в циклі. Тільки після цього додамо другий графік у вікно програми.

Рисування поверхонь.

У системі MATLAB передбачено багато команд та функцій для побудови тривимірних графіків. Значення елементів числового масиву розглядаються як *z*-координати точок над площиною, що визначається координатами *x* та *y*.

Поверхні будуються за допомогою функцій *mesh* та *surf*. Вони відрізняються тим, що в *surf* можна змінювати колір як межі поверхні, так і осередків, з яких вона складається.

Однак для побудови графіка нам необхідно підготувати вихідні дані, тобто сітку значень *x* та *y*. Для цього використовується функція *meshgrid*. Ось приклад її використання

```
[x y]=meshgrid(0:1:2*pi)
x =
     0     1     2     3     4     5     6
     0     1     2     3     4     5     6
     0     1     2     3     4     5     6
     0     1     2     3     4     5     6
     0     1     2     3     4     5     6
     0     1     2     3     4     5     6
     0     1     2     3     4     5     6
y =
     0     0     0     0     0     0     0
     1     1     1     1     1     1     1
     2     2     2     2     2     2     2
     3     3     3     3     3     3     3
     4     4     4     4     4     4     4
     5     5     5     5     5     5     5
     6     6     6     6     6     6     6
```

Тепер, коли ми маємо *x* та *y*, ми можемо отримати *z* і побудувати поверхню (рис. 6.36).

```
[x y]=meshgrid(0:0.1:2*pi);
z=sin(x).*sin(y)+0.3*cos(2*x-3*y)-0.2*sin(x+2*y);
surf(x,y,z)
```

Вирізання шару.

Ідея для вирізування шару проста і заснована на тому факті, що в MATLAB крім числових є і нечислове значення *NaN* (*not a number*). Якщо ми зможемо замінити деякі значення *z* на нечислові, вони не будуть відображатися на екрані, і ми отримаємо ілюзію вирізаного шару.

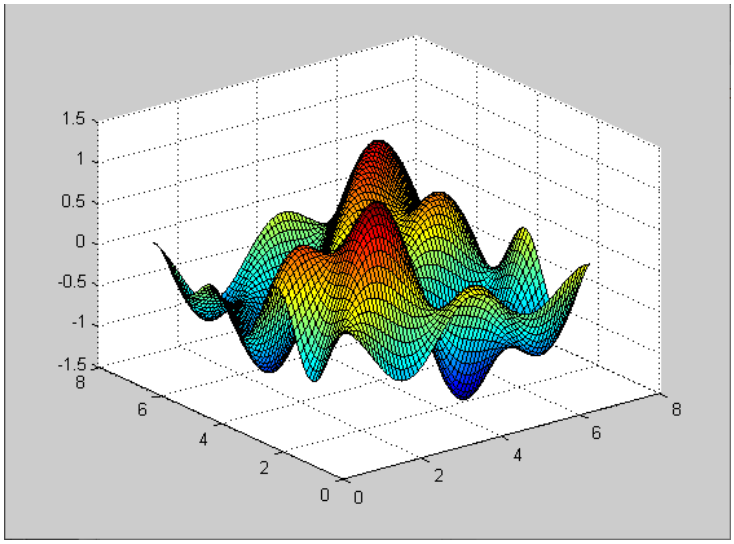


Рис. 6.36. Побудована поверхня.

Робиться це за допомогою функції *find*, яка знаходить значення за певною умовою.

```
z(find(abs(z)<0.05))=NaN;
surf(x,y,z)
```

У цьому рядку ми знаходимо в *z* лише ті точки, що лежать у діапазоні – 0.05 < *z* < 0.05 і змінюємо їх значення на *NaN*. Після такого звернення частина поверхні зникне (рис. 6.37).

Організація циклу.

Послідовне сканування шарів організуємо у циклі. Скануватимемо з нижнього краю поверхні *zMin* до верхнього краю поверхні *zMax*. Цикл буде займати кроки *Steps*. Значення цих змінних вводяться так:

```
Steps=50;
zMin=min(min(z));
zMax=max(max(z));
```

Також нам знадобляться крок *dx* та поточна точка *z*0:

```
dx=(zMax-zMin)/Steps;
z0=zMin;
```

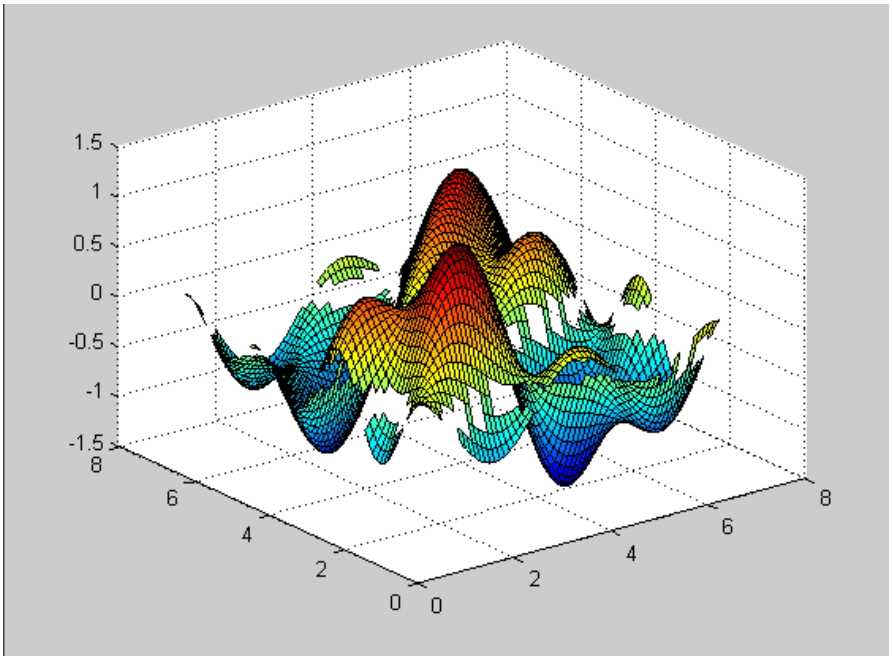


Рис. 6.37. Побудована поверхня з вирізаним шаром.

Якщо на кожному кроці циклу ми надаватимемо деяким точкам з z нечислові значення, то зрештою z зникне. Тому, на кожному кроці ми будемо тимчасово присвоювати значення z змінної Z і видаляти точки лише з Z.

Поєднавши все разом, отримуємо наступну програму.

```
[x y]=meshgrid(0:0.1:2*pi);
z=sin(x).*sin(y)+0.3*cos(2*x-3*y)-0.2*sin(x+2*y);
surf(x,y,z)
zMin=min(min(z));
zMax=max(max(z));
Steps=50;
dx=(zMax-zMin)/Steps;
z0=zMin;
for i=1:Steps
Z=z;
Z(find(abs(z-z0)<dx))=NaN;
surf(x,y,Z)
z0=z0+dx;
drawnow
pause
end
```

Наприкінці кожного кроку циклу ми змінюємо значення поточної точки *z*0 та оновлюємо графік за допомогою вбудованої функції *drawnow*.

Додавання другого графіка.

Щоб розмістити в одному вікні одразу кілька графіків, використовується функція *subplot*, яка розбиває графічне вікно на кілька підвікон. Графік виводиться в те вікно, яке є в даний момент активним.

Функція *subplot*(*m, n, k*) створює *m*×*n* підвікон (*m* – по горизонталі і *n* – по вертикалі), робить активним *k*-е по рахунку підокно і повертає унікальне число (дескриптор) цього вікна.

Дескриптори потрібні у тому, щоб відрізняти одне вікно від іншого, отже вміти переміщатися між цими вікнами.

Знаючи це, перед створенням поверхні введемо наступний рядок

```
hPlot=subplot(1,2,1);
```

Таким чином, ми розбили графічну область на дві частини (1 рядок, 2 стовпці), помістили фокус у перше вікно (рис. 6.38, ліворуч) і назвали це вікно *hPlot* (насправді це не назва, а дескриптор).

Залишилося розібратися з другим графіком (рис. 6.39). Для відображення точок з вирізаного шару, необхідно застосувати вбудовану функція *spy*. Вона замість ненульових значень виводить сині крапки. Назвемо це вікно *hLevel*.

```
hLevel=subplot(1,2,2);
spy(abs(z-z0)<dz)
```

Для переходу з одного підвікна до іншого використовується функція *axes*(*h*) (у MATLAB так називаються графіки з осями), де *h* – це дескриптор вікна. Остаточний код програми виглядає так (додані нові рядки виділені жирним):

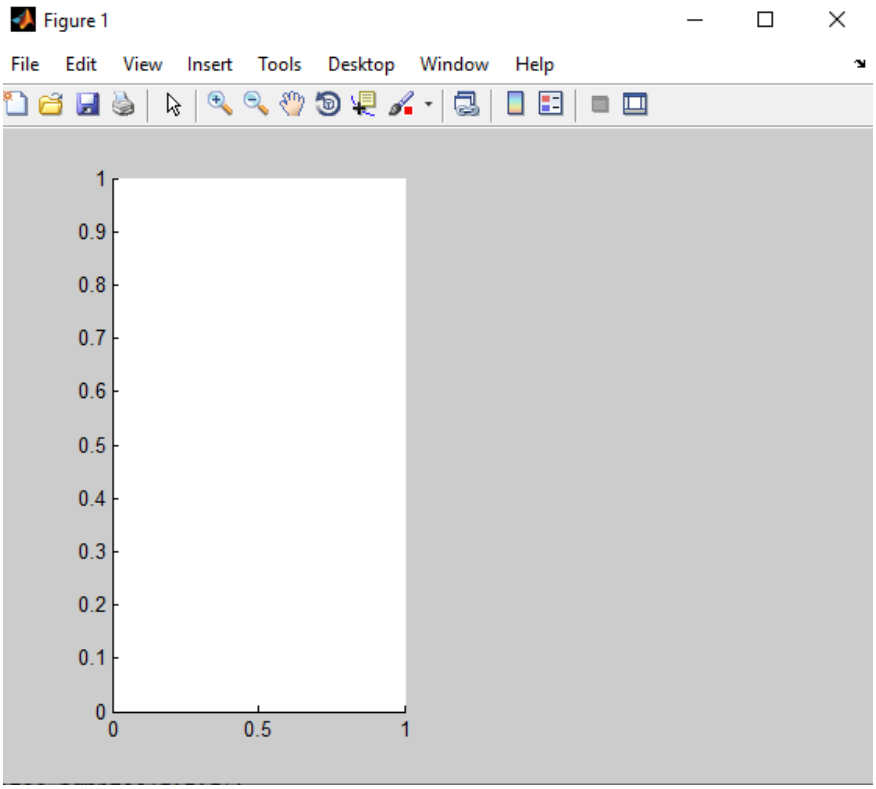


Рис. 6.38. Відображення лівої частини графічної області.

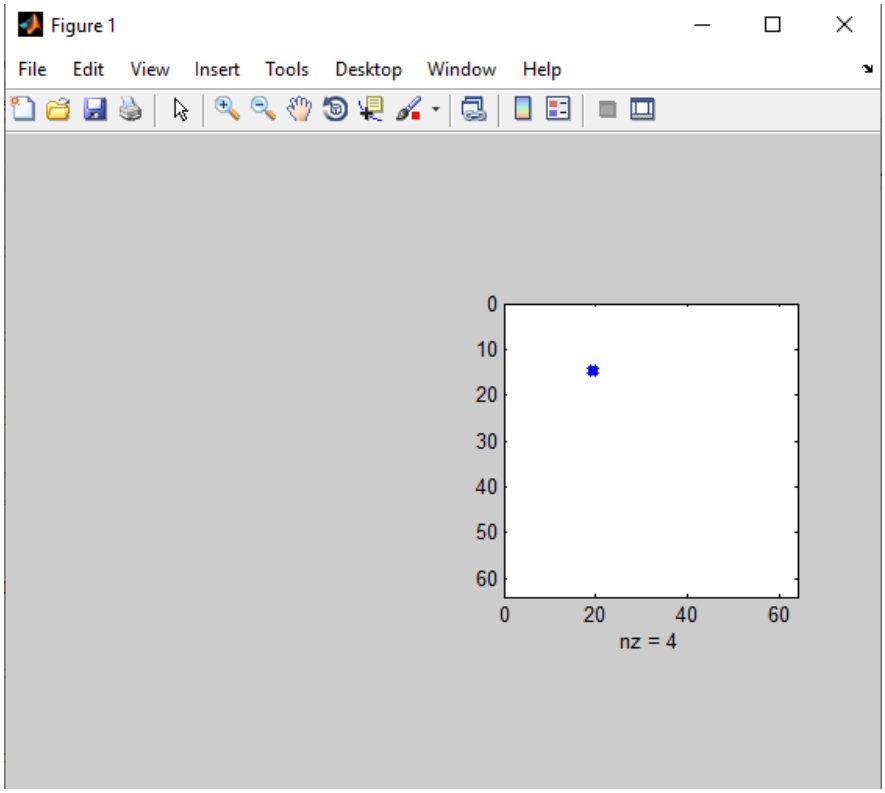


Рис. 6.39. Відображення правої частини графічної області.

```
[x y]=meshgrid(0:0.1:2*pi);
z=sin(x).*sin(y)+0.3*cos(2*x-3*y)-0.2*sin(x+2*y);
hPlot=subplot(1,2,1);
surf(x,y,z)
zMin=min(min(z));
zMax=max(max(z));
Steps=50;
dx=(zMax-zMin)/Steps;
z0=zMin;
hLevel=subplot(1,2,2);
spy(abs(z-z0)<dz)
for i=1:Steps
Z=z;
Z(find(abs(z-z0)<dx))=NaN;
axes(hPlot)
surf(x,y,Z)
axes(hLevel)
spy(abs(z-z0)<dz)
z0=z0+dx;
drawnow
end
```

Програма готова, проте, ще можемо покращити її, додавши інформативні заголовки та прикрасивши поверхню. У поверхні ми змінимо колірну карту, спосіб затінення та рівень прозорості.

Щоб змінити колірну карту поверхні, використовується функція *colormap*(ім'я_карти). Карти можна створювати самому, але краще скористатися однією з вбудованих у систему:

autumn	copper	hsv	prism
bone	flag	jet	spring
colorcube	gray	lines	summer
cool	hot	pink	white

За замовчуванням команда *shading faceted* встановлює рівномірне забарвлення комірок з нанесенням чорних граней. Нас таке не влаштовує, тому скористаємось двома іншими командами: *shading flat* та *shading interp*, які створюють згладжене затінення.

На довершення всього зробимо поверхню трохи прозорою: `alpha(0.8)`.

Ці три команди слід додати до циклу та в циклі після побудови поверхні за допомогою *surf*.

```
surf(x,y,z)
colormap(cool)
shading flat
alpha(0.8)
```

Заголовок активного графічного підвікна визначається функцією *title*('Рядок'). Для переведення числової інформації у рядкову використовуємо функцію *num2str*. Для додавання форматованих даних у рядку використовується функція *sprintf*.

```
[x y]=meshgrid(0:0.1:2*pi);
z=sin(x).*sin(y)+0.3*cos(2*x-3*y)-0.2*sin(x+2*y);
hPlot=subplot(1,2,1);
surf(x,y,z)
zMin=min(min(z));
zMax=max(max(z));
Steps=50;
dx=(zMax-zMin)/Steps;
z0=zMin;
hLevel=subplot(1,2,2);
spy(abs(z-z0)<dz)
for i=1:Steps
Z=z;
Z(find(abs(z-z0)<dx))=NaN;
axes(hPlot)
surf(x,y,Z)
colormap(cool)
shading flat
alpha(0.8)
axes(hLevel)
spy(abs(z-z0)<dz)
z0=z0+dx;
drawnow
end
```

В результаті виконання наведеного коду програми отримаємо наступний вигляд програми, що імітує реалізацію комп'ютерної томографії у MATLAB (рис. 6.40)

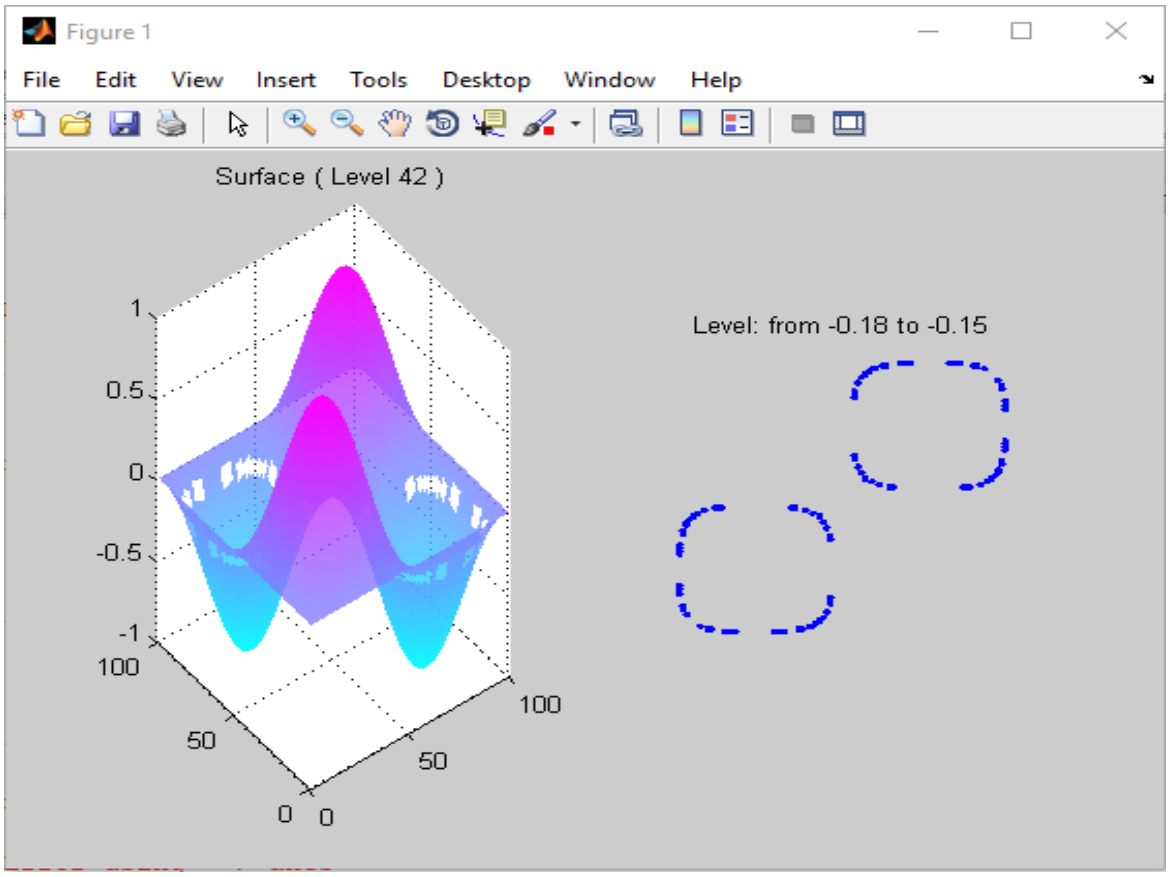


Рис. 6.40. Вікно програми.

Контрольні запитання:

1. Перелічите спеціальні графічні функції, що існують у системі комп'ютерної математики MATLAB та опишіть принципи їх застосування?
2. Які існують функції читання графічного файлу у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?
3. Які існують функції зміни розміру зображення у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?

4. Які існують засоби для зміни палітри кольорів зображення у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?

5. Які існують функції обробки графічних зображень у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?

6. Які існують функції побудови тривимірної поверхні зображення у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?

7. Поясніть загальні математичні підходи які використовуються у комп'ютерній томографії?

8. Які існують засоби для прямого перетворення Радона у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?

9. Які існують засоби для зворотного перетворення Радона у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?

10. Поясніть алгоритм побудови візуалізації процесу томографії у системі комп'ютерної математики MATLAB?

РОЗДІЛ 7

СИМВОЛЬНІ ОБЧИСЛЕННЯ У MATLAB

7.1. Оголошення символьних змінних та констант

У процесі символьних обчислень використовуються змінні та константи особливого типу, так звані символьні об'єкти. Хоча зазвичай у коді MATLAB тип змінних визначається динамічно і немає потреби оголошувати його явно, для символьних об'єктів справа інша. Для оголошення символьних змінних служить команда *syms*, яка, як аргументи, приймає імена змінних, перераховані через пропуск. Наприклад, так:

```
syms x y
% об'єкти, що оголошуються, позначають як дійсні змінні
syms a b real
```

Оголошення символьних констант здійснюється з допомогою функції *sym*. Вона може приймати, як аргумент, рядок, що містить спеціальні змінні, числовий вираз або виклик функції, як у прикладах нижче:

```
sym_pi = sym('pi')
sym_delta = sym('1/10')
sym_sqroot2 = sym('sqrt(2)')
```

Використання символьних констант корисно тим, що обчислення з ними виконуються точно (тобто без обчислювальних похибок) доти, доки не потрібно обчислити деяке числове значення. Зауважимо, що при виведенні вмісту робочого простору командою *whos* символьні змінні та константи відображаються як представники класу *sym object*.

7.2. Символьні вирази та маніпуляції над ними

Після оголошення символьні змінні можна використовувати приблизно так само, як і звичайні числові змінні. Зокрема, до них можливо застосувати оператори + – * / ^, за допомогою яких можна складати символьні вирази:

```
>> syms s t A
>> f = s^2 + 4*s + 5
f =
s^2 + 4*s + 5
>> g = s + 2
g =
s + 2
>> h = f*g
h =
(s^2 + 4*s + 5)*(s + 2)
>> z = exp(-s*t)
z =
exp(-s*t)
>> y = A*exp(-s*t)
y =
A*exp(-s*t)
```

У наведених командах символьні змінні *s, t, A* використовуються для складання символьних виразів, створюючи нові символьні змінні *f, g, h, z, y*. При цьому останні автоматично оголошуються символьними і їх значення не обчислюється і не перетворюється.

При маніпуляціях з символьними об'єктами часто буває корисно дізнатися, які незалежні змінні містяться у виразі, заданому рядком *S*. Для цього можна використовувати функцію *findsym*. Наприклад, продовжуючи приклад вище, можна виконати команду:

```
findsym(z)
ans =
    A, s, t
```

Зауважимо, що MATLAB завжди залишається передусім матричним процесором і тому, до символьних змінних можна вільно застосовувати матричний та векторний записи, відповідні вбудовані оператори і функції. Наведемо приклад:

```
n = 3;
syms x;
B = x.^((0:n)'*(0:n))
B =
[ 1,   1,   1,   1]
[ 1,   x, x^2, x^3]
[ 1, x^2, x^4, x^6]
[ 1, x^3, x^6, x^9]
```

Елементи рядків матриць при введенні відокремлюються пробілами або комами, а стовпців – крапкою з комою, так само як і при введенні звичайних матриць. В результаті утворюються символічні матриці та вектори, до яких застосовні матричні та поелементні операції, а також вбудовані функції. Розглянемо приклад введення та множення двох символьних матриць:

```
» syms a b c d e f g h
» A=[a b;c d]
A =
    [ a, b]
    [ c, d]
» B=[e f; g h]
B =
    [ e, f]
    [ g, h]
» C=A*B
C =
    [ a*e+b*g, a*f+b*h]
    [ c*e+d*g, c*f+d*h]
```

Функція *sym* дозволяє перетворювати значення числових змінних на символьні. Задамо числову матрицю *A* та перетворемо її на символьну матрицю *B*:

```
» A=[1.3 -2.1 4.9; 6.9 3.7 8.5]
A =
    1.3000  -2.1000  4.9000
    6.9000  3.7000  8.5000
» B=sym(A)
B =
    [ 13/10, -21/10, 49/10]
    [ 69/10, 37/10,  17/2]
```

При переході від числових виразів до символьних використовується запис числа у вигляді раціонального дробу. Використання раціональних дробів під час виконання символьних обчислень дозволяє завжди отримувати точний результат, який не містить похибки округлення. Переконатись у цьому можна на наступному простому прикладі. Встановемо формат *long* для відображення максимально можливої кількості значущих цифр для значень числових змінних та знайдемо суму чисел 10^{10} и 10^{-10}

```
» format long
» 1.0e10+1.0e-10
ans =
    1.000000000000000e+010
```

Тепер перетворемо цифри на символьні змінні і знову обчислемо суму:

```
» large=sym(1.0e10);
» small=sym(1.0e-10);
» s=large+small
s = 10000000000000000001/10000000000
```

Раціональна дріб є точним значенням суми. Зрозуміло, що символьні обчислення вимагають великих витрат часу в порівнянні зі звичайними.

Обчислення символьних виразів здійснюється за допомогою функції *vpa*, наприклад:

```
» c=sym('sqrt(2)');
» cn=vpa(c)
```

```
cn =
    1.41421356237309504880168872420969
```

За замовчанням зберігається тридцять дві цифри. Другий вхідний параметр функції *vpa* дозволяє задавати обчислення із заданим числом розрядів, наприклад, із 45 розрядами

```
» cn=vpa(c,45)
cn =
    1.41421356237309504880168872420969807856967187
```

Вихідне значення функції *vpa* є символьною змінною, але його можна використовувати у звичайних обчисленнях, наприклад

```
» cn=vpa(c,45)
cn =
    1.41421356237309504880168872420969807856967187
» cn+2
ans =
    3.41421356237309504880168872420969807856967187
» cn*2
ans =
    2.82842712474619009760337744841939615713934375
```

Результати арифметичних операцій, у даних випадках, отримаємо у символьних змінних. Для переводу символьних змінних у числові, тобто змінні типу *double array*, використовується функція *double*:

```
» cnd=double(cn)
cnd = 1.41421356237310
```

Продемонструємо деякі прості алгебраїчні маніпуляції, які можна здійснювати над символьними об'єктами. Функії:

***expand*(*S*)** – розкриває дужки у виразі *S*;

***factor*(*S*)** – розкладає на множники вираз *S*;

***simplify*(*S*) –** спрощує кожен елемент символьної матриці *S*;

***subs*(*S, oldvar, newvar*)** – замінює у виразі *S* кожне входження символьної змінної *oldvar* нової змінної *newvar*.

Наведемо приклади використання.

Розкриття дужок.

```
>> syms s;
>> A = s + 2;
>> B = s + 3;
>> C = A*B
C =
(s+2)*(s+3)
>> C = expand(C)
C =
s^2 + 5*s + 6
```

Розкладання на множники.

```
>> syms s;
>> D = s^2 + 6*s + 9;
>> D = factor(D)
D =
(s+3)^2
>> p = s^3 - 2*s^2 - 3*s + 10;
>> P = factor(p)
P =
(s+2)*(s^2 - 4*s + 5)
```

Скорочення загального множника (спрощення).

```
>> syms s;
>> H = (s^3 + 2*s^2 + 5*s + 10)/(s^2 + 5);
>> H = simplify(H)
H =
s+2
>> factor(s^3 + 2*s^2 + 5*s + 10)
ans =
(s+2)*(s^2 + 5)
```

Підстановка змінної. Маємо вираз $H(s) = \frac{s+3}{s^2+6s+8}$ де треба обчислити $G(s) = H(s) \mid s = s + 2$.

```
>> syms s;
>> H = (s + 3)/(s^2 + 6*s + 8);
>> G = subs(H, s, s+2)
G =
(s+5)/((s+2)^2 + 6*s + 20)
>> G = collect(G)
G =
(s+5)/(s^2 + 10*s + 24)
```

Таким чином $G(s) = \frac{s+5}{s^2+10s+24}$.

Зазначимо, що кінцевий результат символьних перетворень далеко не завжди виглядає так, як хотілося б користувачу і буває, що отриманий результат незручний для опрацювання.

Функція *subs* дозволяє зробити підстановку одного виразу до іншого. У загальному випадку функція викликається з трьома вхідними аргументами:

- ім'ям символічної функції;
- змінної, що підлягає заміні;
- та виразом, який слід підставити у виразі.

Функція *subs*, зокрема, полегшує введення громіздких символьних виразів. Наприклад:

```
» f=sym('((N^2-x^2)/(N+x)^2)+(sin(2*x)/A)*(sqrt(B*C))
+(A/B)^2+(cos(x)/C)^2')
» pretty(f)
   2    2           2    2                   1/2
  N  - x     cos(x)     A    sin(2 x) (B C)
  -------- + ------- + -- + -----------------
         2       2        2            A
  (N + x)       C        B
» f=subs(f,'A','sin(x)');
» f=subs(f,'B','tan(x)');
```

```
» f=subs(f,'C','cot(x)');
» pretty(f)
   2    2          2         2                              1/2
  N  - x     cos(x)    sin(x)    sin(2 x) (cot(x) tan(x))
  -------- + ------- + ------- + -------------------------
         2         2         2              sin(x)
  (N + x)    cot(x)    tan(x)
```

Спростимо отриманий вираз за допомогою функції *simple*:

```
» f2=simple(f)
f2 =
    (2*sin(x)*N+sin(2*x)*N+sin(2*x)*x)/(N+x)/sin(x)
» pretty(f2)
      2 sin(x) N + sin(2 x) N + sin(2 x) x
      -----------------------------------
           (N + x) sin(x)
```

Підстановка замість змінної її числового значення призводить до обчислення символьної функції від значення аргументу, наприклад

```
» f=sym('((N^2-x^2)/(N+x)^2)+(sin(2*x)/A)*(sqrt(B*C))
+(A/B)^2+(cos(x)/C)^2')
» pretty(f)
   2    2          2    2                   1/2
  N  - x     cos(x)    A    sin(2 x) (B C)
  -------- + ------- + -- + -----------------
         2         2        2            A
  (N + x)       C        B
» q=subs(f,'x',0)
q =
    1+A^2/B^2+1/C^2
» f3=sym('sin(x)+exp(x)+tan(x)');
» q1=subs(f3,'x',0)
q1 =
    1
```

Наведемо ще приклад застосування. Обчислення коефіцієнтів при ступенях незалежної змінної здійснюється за допомогою функції *collect*. Введіть поліном і відобразіть його у командному вікні за допомогою *pretty*:

```
» p=sym('(x+a)^4+(x-1)^3-(x-a)^2-a*x+x-3');
» pretty(p)
             3                  4           2
  x + (x - 1)  - a x + (a + x)  - (a - x)  - 3
```

Потім застосуйте до полінома функцію *collect*:

```
» pc=collect(p);
» pretty(pc)
   4               3        2        2
  x  + (4 a + 1) x  + (6 a  - 4) x  +
      3                4    2
+ (4 a  + a + 4) x + a  - a  - 4
```

За умовчанням, у якості незалежної змінної, в поліномі обирається *x*, проте вважається, що *a* – незалежна змінна, а *x* входить у коефіцієнти полінома, який залежить від *a*. Другий аргумент функції *collect* призначений для зазначення змінної, при ступенях якої слід знайти коефіцієнти

```
» pca=collect(p,'a');
» pretty(pca)
   4          3        2        2        3
  a  + (4 x) a  + (6 x  - 1) a  + (4 x  + x) a + x +
             3    2    4
+ (x - 1)  - x  + x  - 3
```

Функція *expand* представляє поліном сумою одночленів:

```
» pe=expand(p);
» pretty(pe)
   4      3         2  2    2         3
  a  + 4 a  x + 6 a  x  - a  + 4 a x  +
           4    3       2
+ a x + x  + x  - 4 x  + 4 x - 4
```

Аргументом *expand* може бути не тільки поліном, а й символьні вирази, що містить тригонометричні, експоненційні та логарифмічні функції:

```
» pk=sym('(sin(x)+cos(x))^3+(sin(x)+cos(x))^2+(sin(x)+
cos(x))')
pk =
     (sin(x)+cos(x))^3+(sin(x)+cos(x))^2+(sin(x)+cos(x))
» pk2=expand(pk);
» pretty(pk2)
      3           2                 2                 2
cos(x)  + 3 cos(x)  sin(x) + cos(x)  + 3 cos(x) sin(x) +
                                  3         2
+ 2 cos(x) sin(x) + cos(x) + sin(x)  + sin(x) + sin(x)
```

Аргументами функцій *expand, collect* може бути не тільки окремо поліном, тригонометрична, експоненційна, логарифмічна функції, але і їх поєднання, наприклад:

```
» p=sym('(x+a)^4+(x-1)^3+(sin(x)+cos(x))^4');
» pretty(p)
         3                     4           4
  (x - 1)  + (cos(x) + sin(x))  + (a + x)
» p1=collect(p)
p1 =
     x^4+(1+4*a)*x^3+(-3+6*a^2)*x^2+(3+4*a^3)*x+a^4-
-1+(sin(x)+cos(x))^4
» p2=expand(p1)
p2 =
     x^4+x^3+4*x^3*a-3*x^2+6*x^2*a^2+3*x+4*x*a^3+a^4-
-1+sin(x)^4+4*sin(x)^3*cos(x)+6*sin(x)^2*cos(x)^2+4*sin(x)*
*cos(x)^3+cos(x)^4
» pretty(p2)
   4      3        2  2        3    4    3      2
  a  + 4 a  x + 6 a  x  + 4 a x  + x  + x  - 3 x  + 3 x +
        4           3                 2       2
 + cos(x)  + 4 cos(x)  sin(x) + 6 cos(x)  sin(x)  + 4*
               3         4
 * cos(x) sin(x)  + sin(x)  - 1
```

Символьні поліноми розкладаються на множники функцією *factor*, якщо множники, що виходять, мають раціональні коефіцієнти:

```
» p=sym('7*x^5-56*x^4+105*x^3+140*x^2-532*x+336');
```

```
» p1=factor(p)
p1 =
     7*(x-2)*(x-3)*(x-4)*(x+2)*(x-1)
```

Функція *factor* може також представляти числа у вигляді добутку простих чисел

```
» syms a
» a=sym('2738470');
» a1=factor(a)
a1 = (2)*(5)*(7)*(19)*(29)*(71)
```

Зверніть увагу, що звернення

```
» a2=factor(2738470)
a2 = 2   5   7  19  29  71
```

виводить у командне вікно аналогічний результат, проте змінна *a*1 є символьною, а змінна *a*2 – дійсною, у чому нескладно переконатися за допомогою *whos*:

```
» whos a1 a2
   Name  Size    Bytes Class
    a1   1x1     176 sym object
    a2   1x6     48 double array
Grand total is 33 elements using 224 bytes
```

7.3. Обчислення символьних виразів та їх відображення

Незмінно настає момент, коли необхідно обчислити числове значення результату викладок, що виражені символьними об'єктами, чи нарисувати їх графік функції. Розглянемо, які можливості для цього надає MATLAB.

Обчислення значення виразу.

***double*(*S*)** – перетворює символьну матрицю *S*, замінюючи її елементи їх числовими значеннями. Матриця *S* не повинна містити символьних змінних.

Як випливає з опису функції, у символьному виразі перед обчисленням його чисельного значення необхідно зробити всі можливі підстановки, щоб позбавитися вільних символьних змінних. Розглянемо приклад:

```
>> syms s;
>> E = s^3 -14*s^2 + 65*s - 100);
>> F = subs(E, s, 7.1)
F = 13671/1000
>> G = double(F)
G = 13.6710
```

Рисування за допомогою функції *ezplot*.

Символьний вираз може бути нарисований безпосередньо за допомогою функції *ezplot*:

***ezplot*(*f*) –** будує графік функції *f*(*x*), де *f* є математичною функцією змінної *x*. Інтервал, на якому зображується графік, дорівнює [-2π, 2π].

***ezplot*(*f, xmin, xmax*)** – рисує графік на заданому інтервалі.

Наприклад, графік кубічного багаточлену $A(s) = s^3 + 4s^2 - 7s - 10$ на інтервалі [–1, 3] можна зобразити таким чином (рис. 7.1):

```
syms s;
A = s^3 + 4*s^2 - 7*s - 10;
ezplot(A, -1, 3)
ylabel('A(s)')
```

Розлянемо інший приклад (рис. 7.2):

```
» f=sym('x*sin(x^2)^3');
» ezplot(f)
```

Зауважимо, що у данному прикладі автоматично створюється відповідний заголовок. Другим аргументом може бути заданий вектор із межами відрізка, на якому потрібно побудувати графік функції:

```
» ezplot(f,[-6 6])
```

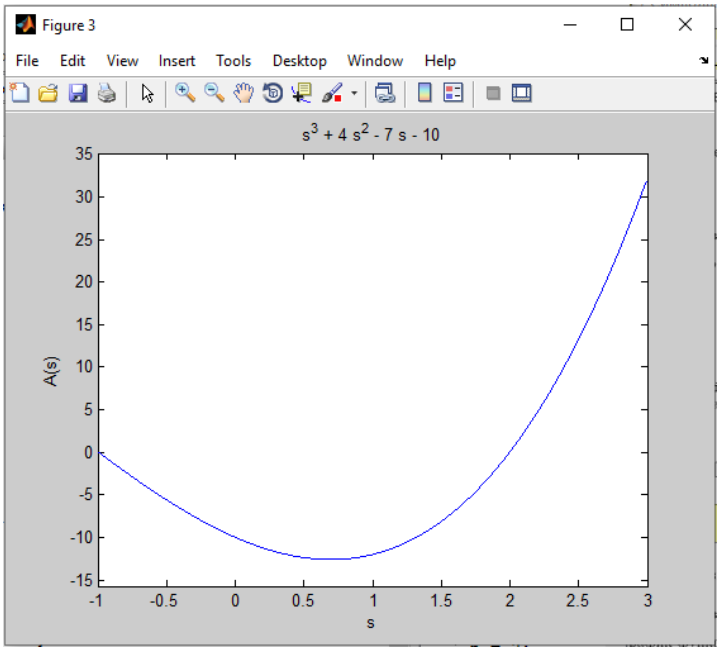


Рис. 7.1. Графік кубічного багаточлену.

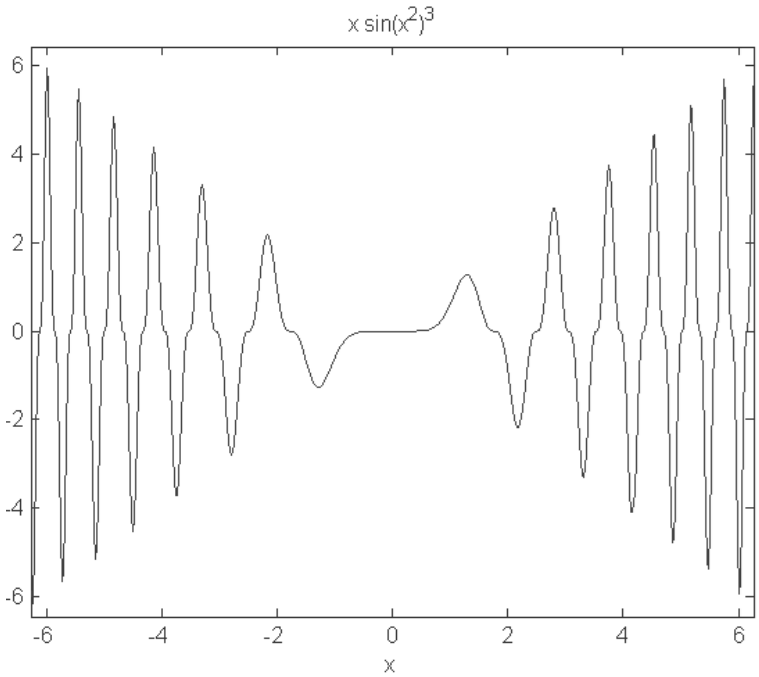


Рис. 7.2. Графік символічної функції

Функція *ezplot* має деякі відмінності від свого аналога – функції *fplot*, що застосовується до числових функцій. Зокрема, можлива вказівка символьної функції, яка залежить від двох аргументів:

```
» z=sym('x^2+a^3');
» ezplot(z,[-2 1 -3 4])
```

Межі зміни аргументів визначаються їх назвами. Перші два числа відповідають першому за алфавітом аргументу, а останні – другому. У прикладі *a* змінюється від –2 до 1, а *x* змінюється від –3 до 4. У графічне вікно виводиться лінія (рис. 7.3), що задовольняє виразу `x^2+a^3=0`.

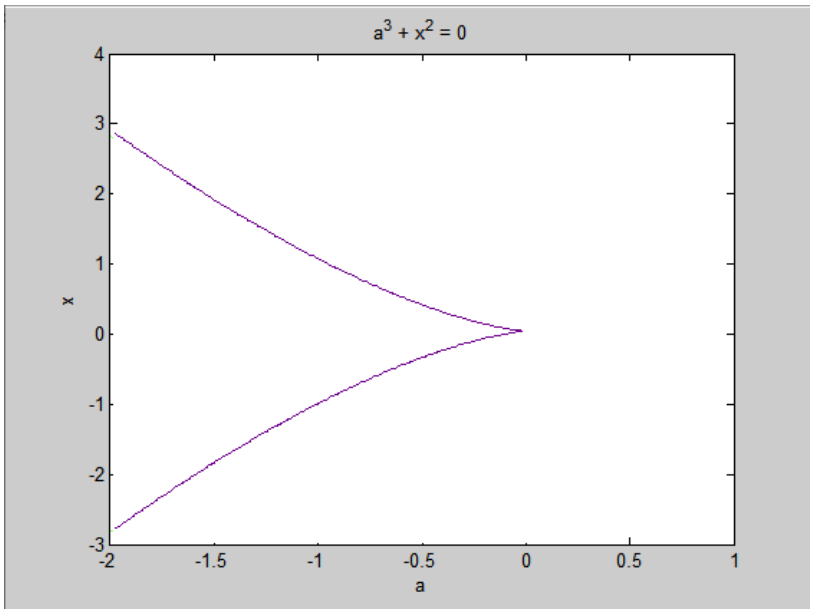


Рис. 7.3. Графік функції $x^2+a^3=0$.

Рисування за допомогою функції *plot*. Графік функції можна побудувати використовуючи функцію plot:

***plot*(*x, y, frmt*)** – рисує графік функції за точками, координати яких задані масивами *x* та *y*. Параметр *frmt* визначає зовнішній вигляд графіка.

Як приклад, ми нарисуємо графіки двох функцій. Одна буде задана явно, інша – символьним виразом. Одночасно продемонструємо деякі можливості налаштування зовнішнього вигляду графіка.

```
clear all; % Очищуємо робочий простір
syms x y; % Визначаємо символьні змінні
% Визначаємо функцію z(x,y) = (x^2-y^2)/(x^2+y^2)
z = (x^2 - y^2)/(x^2 + y^2);
x1 = linspace(0, pi); % Задаємо масив абсцис точок графіка
```

```
y1 = sin(x1); % Функція задається явним виразом М-коду
% Обчислюємо z(x,1) = (x^2-1)/(x^2+1) графік якої і нарисуємо
zy1 = subs(z, y, 1);
z1 = subs(zy1, x, x1); % Обчислюємо ординати точок
% Рисуємо графіки обох функцій різним кольором
plot(x1,y1,'b',x1,z1,'r')
% Змінюємо шрифт для написів
set(get(gcf,'CurrentAxes'),'FontSize', 10)
title('\bf2D-графік'); % Задаємо заголовок графіка
xlabel('\itx'); % Задаємо мітку осі абсцис
ylabel('\ity'); % Задаємо мітку осі ординат
da = daspect; % Отримуємо поточний масштаб графіка
da(1:2) = min(da(1:2)); % Вирівнюємо масштаб графіка
daspect(da); % Встановлюємо новий масштаб
xlim([0, pi]); % Встановлюємо межі по осі абсцис
```

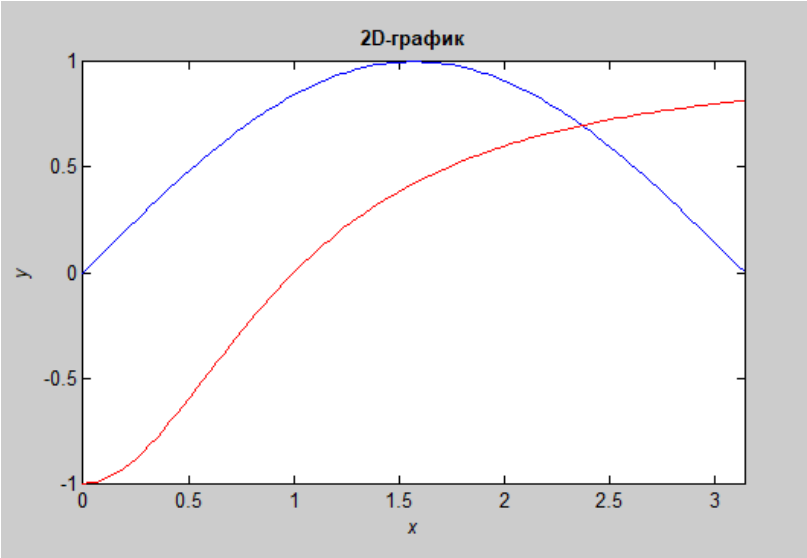


Рис. 7.4. Графік двох функцій.

Зазначимо, що зміна параметрів об'єктів за допомогою функцій *get, set* є типовою для MATLAB, особливо під час роботи з графікою. Приблизно так само можна нарисувати тривимірні графіки (деталі можна дізнатися в документації по функціях *plot3, mesh, surf*).

Інші графічні функції групи *ez…* також можна використовувати для побудови графіків символьних функцій. Ось кілька прикладів:

```
syms s t;
ezmesh(exp(-s)*cos(t),exp(-s)*sin(t),t,[0,8,0,4*pi])
syms u v;
ezmesh(u,(abs(sqrt(1-u^2)+v)-abs(sqrt(1-u^2)-v))/2,u,
[-1,1,-1,1],21)
syms x; ezplot(erf(x));
grid
syms t;
ezplot3(sin(t), cos(t), t,[0,6*pi])
syms t;
ezpolar(1+cos(t))
f=sym('x^2 +y^2');
ezsurf(f,[-1 1 -1 1])
```

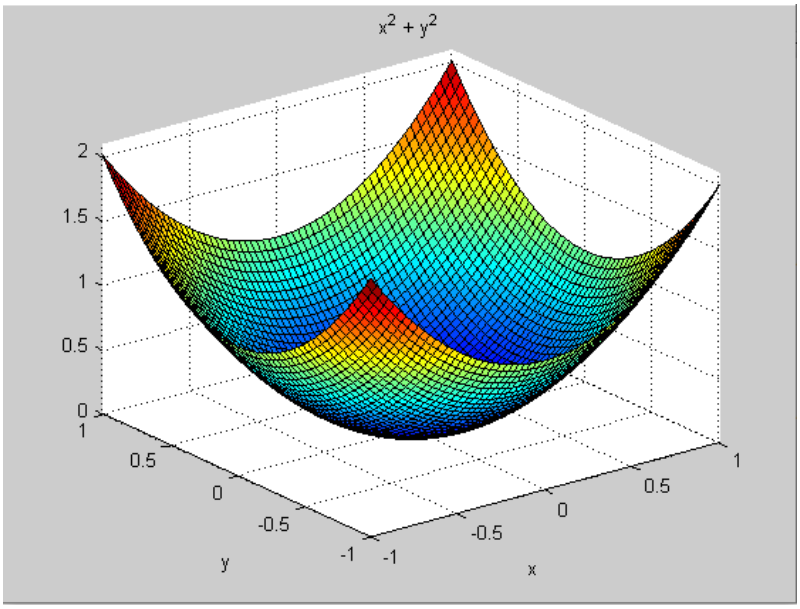


Рис. 7.5. Графік функції у полярних координатах.

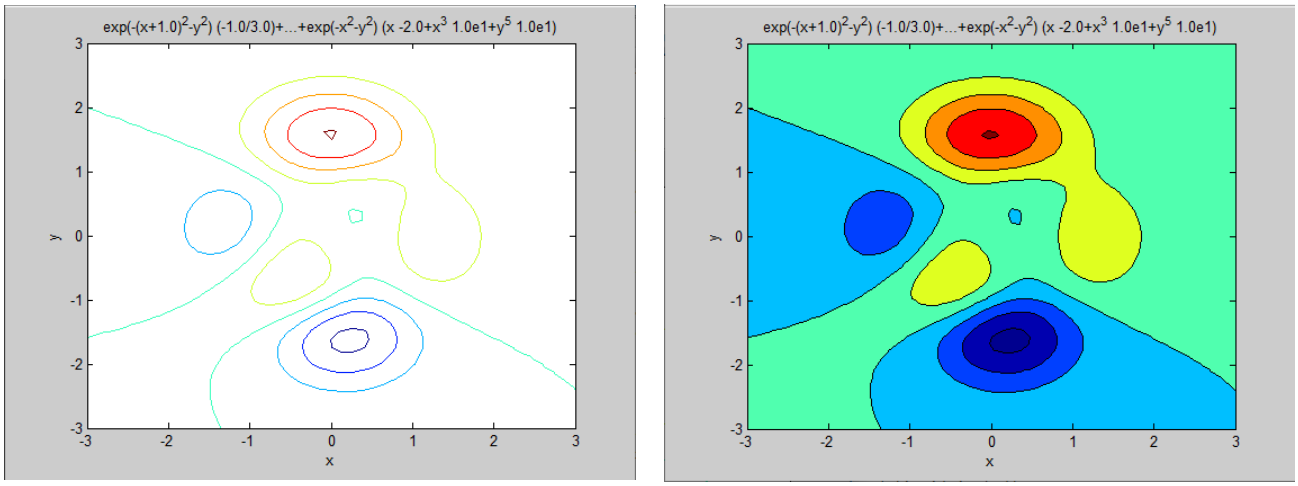
Функція *ezsurf* відображає графік символьної функції лише допустимих значень аргументів, інші значення відкидаються, що дозволяє досліджувати область визначення функції двох змінних. Наприклад:

```
ezsurf('asin(x^y)',[0 6 -7 7])
```

Функції *ezcontour* (рис. 7.6, *а*), *ezcontourf* (рис. 7.6, *б*) застосовуються для побудови контурних графіків – ліній рівня функцій двох змінних. Другий

параметр задає прямокутну область зміни аргументів, третій n – кількість осередків мережі $n \times n$ на яку розбивається область зміни аргументів (за умовчанням $n = 60$)

```
syms x y
f = 3*(1-x)^2*exp(-(x^2)-(y+1)^2) - 10*(x/5 - x^3 - y^5)*
*exp(-x^2-y^2) - 1/3*exp(-(x+1)^2 - y^2);
ezcontour(f,[-3,3],49)
ezcontourf( f, [-3,3, -3, 3],60)
```



а *б*

Рис. 7.6. Побудова контурних графіків двома різними функціями.

7.4. Розкладання в ряд Тейлора та визначення символьних виразів для сум

Розкладання математичних функцій у ряд Тейлора дозволяє виконувати функцію *taylor*, наприклад:

```
» f=sym('sin(x)');
» tf=taylor(f);
» pretty(tf)
   5     3
  x     x
  --- - -- + x
  120   6
```

За замовчуванням виводиться шість членів ряду розкладання на околиці точки нуль. Число членів розкладання можна встановити в другому додатковому параметрі функції *taylor*. Третій параметр вказує, за якою саме зі змінних слід розкладати у разі, коли символьна функція визначена від кількох змінних. Точка, в околиці якої проводиться розкладання, вказується у четвертому вхідному аргументі функції *taylor*, наприклад

```
» syms x;
» f=sym('sin(x)');
» tf=taylor(f, 5, x, pi/4);
» pretty(tf)
   1/2 / pi     \3    1/2 / pi     \2    1/2 / pi     \4
  2    | -- - x |    2    | -- - x |    2    | -- - x |
       \ 4      /         \ 4      /         \ 4      /
2           \ 4      /
  ---------------- - ---------------- + ---------------- +
         12                   4                  48
           1/2 / pi     \
    1/2   2    | -- - x |
  2            \ 4      /
+ ---- - ---------------
  2            2
```

Знаходження символьних виразів для сум, зокрема й нескінченних, дозволяє здійснити функцію *symsum*. Звернення до *symsum* у загальному випадку передбачає завдання чотирьох аргументів: доданку у символьній формі, яка залежить від індексу, самого індексу, верхньої і нижньої межі суми. Якщо до доданку входить факторіал, то слід застосувати до вираження факторіалу функцію *sym*. Знайдіть значення нескінченної суми, що є розкладанням до ряду функцій *sin*(*x*)

$$s = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!}$$

```
» syms k x
» s=symsum((-1)^(k)*x^(2*k+1)/sym('(2*k+1)!'),k,0,inf)
s = sin(x)
```

7.5. Визначення меж, диференціювання та інтегрування

Функція *limit* знаходить межу функції у певній точці, включаючи плюс чи мінус нескінченність. Першим вхідним аргументом *limit* є символьний вираз, другим – змінна, а третім – точка, в якій визначається межа. Нехай, наприклад, потрібно вирахувати

$$\lim_{x\to\infty}(1+\frac{1}{x})^{ax}$$

```
» syms a x
» limit((1+1/x)^(x*a),x,Inf)
ans = exp(a)
```

Функція *limit* дозволяє знаходити односторонні межі. Для знаходження межі праворуч слід зазначити четвертий додатковий аргумент *'right*', а ліворуч – '*left*'. Знайдемо рішення наступних двох завдань:

$$\lim_{x\to 0+}(10+x)^{1/x};\quad \lim_{x\to 0-}(10+x)^{1/x}.$$

```
» syms x
» limit((10+x)^(1/x),x,0, 'left')
ans = 0
» limit((10+x)^(1/x),x,0, 'right')
ans = inf
```

Зверніть увагу, що звичайної межі в точці нуль не існує:

```
» limit((10+x)^(1/x),x,0)
ans =
     NaN
```

Визначення похідної через межу дозволяє застосовувати обмеження для диференціювання функцій. Наприклад, знайдемо першу похідну функції *arctg*(*x*), використовуючи рівність

$$\frac{d}{dx}\text{arctg}(x)=\lim_{h\to 0}\frac{\text{arctg}(x+h)-\text{arctg}(x)}{h}$$

```
» syms h x
» L=limit((atan(x+h)-atan(x))/h, h, 0);
» pretty(L)
       1
    ------
         2
    1 + x
```

Обчислення похідних будь-якого порядку простіше проводити за допомогою функції ***diff*.** Символьний запис функції вказується в першому вхідному аргументі, змінна, за якою здійснюється диференціювання у другому, а порядок похідної в третьому. Застосуємо *diff* для обчислення першої та другої похідних функції *arctg*(*x*):

```
» P=diff('atan(x)',x,1);
» pretty(P)
    1
  ------
   2
  x  + 1
» P=diff('atan(x)',x,2);
» pretty(P)
      2 x
  - ---------
      2     2
    (x  + 1)
```

Символьне інтегрування є значно складнішим завданням, ніж диференціювання. *ToolBox Symbolic Math* дозволяє працювати як із невизначеними інтегралами, так і з визначеними. Невизначені інтеграли від символьних функцій обчислюються з допомогою *int.* Як вхідні аргументи вказуються символьна функція та змінна, за якою здійснюється інтегрування. Наприклад, нехай необхідно обчислити невизначений інтеграл $f = \int \exp(2x)dx$, тоді отримуємо:

```
» syms x
» f=sym('exp(2*x)');
```

```
» I=int(f,x)
» pretty(I)
    1/2 exp(2 x)
```

Очевидно, що функція *int* не дозволяє отримати невизначений інтеграл від довільної функції. Для знаходження певного інтеграла у символьному вигляді слід задати нижню та верхню межі інтегрування, відповідно, у третьому та четвертому аргументах *int*

```
» syms x a b
» f=sym('exp(2*x)');
» I=int(f,x,a,b);
» pretty(I)
    1/2 exp(2 b) - 1/2 exp(2 a)
```

Подвійні інтеграли обчислюються дворазовим застосуванням функції *int*. Наприклад, необхідно обчислити інтеграл $\int_c^d \int_a^b y\sin(x)dxdy$, тоді для його визначення необхідно задати символьні змінні *a, b, c, d, x,* у підінтегральну функцію *f* від *x* і *y* й проінтегрувати спочатку по одній змінній, а потім по іншій

```
» syms a b c d x y
» f=sym('y*sin(x)');
» Ix=int(f,x,a,b)
Ix =
    -y*cos(b)+y*cos(a)
» Iy=int(Ix,y,c,d)
Iy =
    1/2*(-cos(b)+cos(a))*(d^2-c^2)
» pretty(Iy)
      2    2
    (c  - d ) (cos(a) - cos(b))
  - --------------------------
                2
```

Аналогічним чином обчислюються будь-які кратні інтеграли у символьному вигляді.

7.6. Розв'язання рівнянь та систем рівнянь

7.6.1. Розв'язання рівнянь за допомогою функції solve

MATLAB можна використовувати для вирішення алгебраїчних і трансцендентних рівнянь та систем рівнянь, заданих у вигляді масиву символьних виразів. Основний інструмент для цього – функція *solve*. Вона може викликатися у різній формі:

```
solve(E1, E2, ..., EN)
solve(E1, E2, ..., EN, var1, var2, ..., varN)
```

Тут *E*1, *E*2, ..., *EN* – символьні вирази або змінні, в яких вони містяться, а *var*1, ..., *varN* – змінні, щодо яких слід розв'язати систему рівнянь *E*1 = 0, *E*2 = 0, ..., *EN*=0. Зрозуміло, перша форма виклику цієї функції допустима лише у разі, коли немає неоднозначностей щодо того, що саме слід знайти. Функція *solve* повертає єдиний символьний вираз, якщо рівняння (або система рівнянь) має єдине рішення і вектор рішень у іншому випадку. Якщо рівняння містить періодичні функції і може мати нескінченне число рішень, то функція обмежується тим, що повертає корні за один період в околиці нуля. Розглянемо приклади.

Єдине вирішення.

```
>> syms s;
>> E = s + 2;
>> s = solve(E);
s = -2
```

Декілька рішень.

```
>> syms s;
>> D = s^2 + 6*s + 9;
>> s = solve(D)
s =  [ -3]
     [ -3]
```

Рівняння з параметрами.

```
>> syms theta x z;
>> E = z*cos(theta) - x;
>> theta = solve(E, theta)
theta = acos(x/z)
```

Рівняння з комплексним коренем.

```
>> syms x;
>> E = exp(2*x) + 4*exp(x) - 32;
>> x = solve(E)
x =
[ log(-8)]
[ log(4)]
>> log(-8)
ans = 2.0794 + 3.1416i
>> log(4)
ans = 1.3863
```

Рівняння з безліччю рішень.

```
>> E = cos(2*theta) - sin(theta);
>> solve(E)
theta =
[-1/2*pi]
[ 1/6*pi]
[ 5/6*pi]
```

Нагадаємо, що чисельне (тобто наближене) значення рішення можна отримати, скориставшись функцією *double*.

7.6.2. Чисельне вирішення рівнянь, заданих символьним виразом

Зрозуміло, що не кожне рівняння можна вирішити аналітично. Для чисельного розв'язання системи нелінійних рівнянь (алгебраїчних чи трансцендентних) можна скористатися функцією *fsolve.* Ця функція у найпростішому випадку приймає два аргументи. Перший – анонімна функція чи ім'я функції, яка описує праву частину системи рівнянь, записаної у нормальній формі. Другий аргумент визначає початкове наближення для ітераційної

процедури пошуку рішення.

Процедура чисельного рішення часто є лише невеликим блоком у програмі, тому невідомо, як саме виглядає система рівнянь. Найпростіший, хоч і не найкращій спосіб полягає в тому, щоб «на льоту» створювати файл з описом необхідної задачі і потім передавати ім'я функції *fsolve*. Для нас цей спосіб цікавий тим, що знайомить з роботою файлової системи в MATLAB і надалі, при вирішенні задач варіаційного обчислення, буде використовуватися саме він.

Наведемо приклад, забезпечивши його коментарями. Отже, нехай необхідно вирішити систему рівнянь виду:

$$\begin{cases} r(\varphi - \sin\varphi) = 3; \\ r(1 - \cos\varphi) = 2. \end{cases}$$

Опишемо необхідні дії:

```
clear all;                          % Очищуємо пам'ять
eq1 = 'r * (phi - sin (phi)) - 3;  % Описуємо перше рівняння
eq2 = 'r*(1 - cos(phi)) - 2';      % Описуємо друге рівняння
% Починаємо створювати вміст майбутньої функції
s{1} = 'function y = MyFunction(x)'; % Заголовок функції
s{2} = 'r = x(1); phi = x (2); ';  % Визначаємо змінні
% Визначаємо результат
s{3} = ['y(1) = 'vectorize(eq1)'; '];
s{4} = ['y(2) = 'vectorize(eq2)'; ']; % виконання функції
% Визначаємо повне ім'я файлу
filename = fullfile(pwd, 'MyFunction.m');
fid = fopen (filename, 'w'); % Відкриваємо файл на запис
fprintf(fid, %s\n, s{:});          % Записуємо вміст
fclose(fid);                       % Закриваємо файл
% Вирішуємо рівняння чисельно та друкуємо результат
xinit = ones (2,1); % Задаємо початкове наближення
[xzero, yzero, eflag, opt] = fsolve('MyFunction', xinit);
fprintf('Кількість обчислень правої частини: %d\n',
opt.iterations);
fprintf('Знайдене рішення: \nr = %12.7f\nphi = %12.7f\n',
xzero);
delete (filename); % Видаляємо більше непотрібний файл
```

В результаті виконання описаної послідовності команд ми отримаємо наступний результат:

```
Кількість обчислень правої частини: 7
Знайдене рішення:
r = 1.0013267
phi = 3.0687767
```

Порівняємо результат обчислень із результатами наступної команди:

```
>> solve(eq1, eq2,'r, phi')
ans =
    phi: [1x1 sym]
      r: [1x1 sym]
```

7.7. Диференціювання та інтегрування

Диференціювання. Щоб знайти похідну символьного виразу, слід скористатися однією з таких форм функції *diff*:

***diff*(*S*)** – диференціює вираз *S* по незалежним змінним, визначеним функцією *findsym*;

***diff*(*S*, *t*)** - диференціює вираз *S* по змінній *t*;

***diff*(*S*, *n*)** - *n* разів диференціює вираз *S*;

***diff*(*S*, *t*, *n*)** - *n* разів диференціює вираз *S* по змінній *t*.

```
>> syms s n;
>> p = s^3 + 4*s^2 - 7*s - 10;
>> d = diff(p)
d =
3*s^2+8*s-7
>> e = diff(p, 2)
e =
6*s+8
>> g = s^n;
>> h = diff(g)
h = s^n*n/s
```

```
>> h = simplify(h)
h =
s^(n-1)*n
>>
>> f = exp(-(x^2)/2);
>> diff(f)
ans =
-x*exp(-1/2*x^2)
```

Інтегрування. Як і диференціювання, символьне обчислення інтегралів (як визначених, і невизначених) виконується однією функцією *int*:

***int*(*S*)** – інтегрує вираз *S* за незалежними змінними, визначеними функцією *findsym*. Якщо *S* – константа, то інтегрування виконується по *x*;

***int*(*S, t*) –** інтегрує вираз *S* по змінній *t*;

***int*(*S, a, b*) –** обчислює певний інтеграл на проміжку [*a, b*];

***int*(*S, t, a, b*) –** обчислює певний інтеграл за вказаною змінною.

Наведемо кілька прикладів:

```
>> syms x n a b t
>> int(x^n)
ans =
x^(n+1)/(n+1)
>> int(x^3 + 4*x^2 + 7*x + 10)
ans =
1/4*x^4+4/3*x^3+7/2*x^2+10*x
>> int(x, 1, t)
ans =
1/2*t^2-1/2
>> int(x^3, a, b)
ans =
1/4*b^4-1/4*a^4
>> int(cos(x))
ans =
sin(x)
>> int(exp(-x^2))
ans =
1/2*pi^(1/2)*erf(x)
```

7.8. Розв'язання звичайних диференціальних рівнянь

З рішенням звичайних диференціальних рівнянь (ЗДР) ситуація приблизно так ж, як і з рішенням алгебраїчних рівнянь: у деяких випадках їх можна вирішити аналітично, але частіше доводиться вирішувати чисельно. Нижче розглянемо, як це можна зробити.

Рішення ЗДР за допомогою функції *dsolve*.

Функція *dsolve* може знаходити як загальне рішення ЗДР, так і часткове рішення для заданих початкових або граничних умов. При цьому слід дотримуватись певних обмежень на форму запису рівняння (або системи рівнянь). Так, якщо невідома функція позначена символьною змінною *y*, її похідні слід позначати як *D*[*n*]*y*, де в квадратних дужках вказаний порядок похідної. Таким чином, похідна повинна позначатись у символьному вираженні *Dy*, друга похідна – *D*2*y* і т.д. Функція *dsolve* може викликатися з різним набором параметрів, залежно від типу завдання і порядку системи рівнянь. Деталі можна знайти у *help*, ми обмежимося невеликим прикладом.

Нехай необхідно вирішити задачу Коші виду:

$$y'' + 2y' + y = x\sin 2x\,;$$

$$y(0) = 1; y'(0) = -1.$$

Опишемо необхідні для цього дії.

```
clear all; % Очищуємо пам'ять
eq = 'D2y + 2*Dy + y - x*sin(2*x)'; % Описуємо рівняння
x0 = 0; % Початкові умови
y0 = 1;
y10 = -1;
ic1 = sprintf('y(%d) = %d', x0, y0); % Перетворимо їх
ic2 = sprintf('Dy(%d) = %d', x0, y10); % у символьні вирази
fprintf('Диференційне рівняння: \n%s=0\n', eq);
fprintf('Початкові умови: \n%s\n%s\n', ic1, ic2);
% Вирішуємо початкове завдання
Sol = dsolve(eq, ic1, ic2, 'x');
```

```
if isempty(Sol) % Якщо рішень немає, то ...
disp('Рішення не знайдені'); % друкуємо відповідь
else % В іншому випадку ...
n = length (Sol);
fprintf('Знайдено рішень: %d\n', n)
for i = 1:n % друкуємо всі рішення
fprintf('Рішення %d: \ny=%s\n', i, char(Sol(i)));
end
end
```

В результаті виконання описаної послідовності команд ми отримаємо наступний результат:

```
Диференціальне рівняння:
D2y + 2*Dy + y - x*sin(2*x)=0

Початкові умови:
y(0) = 1
Dy(0) = -1

Знайдено рішень: 1

Рішення 1:
y=129/(125*exp(x)) - (4*cos(2*x))/125 + (22*sin(2*x))/125 -
(8*x*cos(2*x))/25 - (4*x)/(25*exp(x)) - (6*x*sin(2*x))/25 +
(2*x^2*cos(2*x))/5 - (x^2*sin(2*x))/5 + (x*(4*cos(2*x) +
3*sin(2*x) - 10*x*cos(2*x) + 5*x*sin(2*x)))/25
```

7.9. Чисельне вирішення рівнянь, заданих у символьному вигляді

Чисельне рішення ЗДР здійснюється в MATLAB цілим набором різних функцій, які реалізують різні методи рішення і будуть розглянуті на наступних лекціях. Взагалі кажучи, використовуються вони приблизно так само, як ми до цього використовували *fsolve*: рівняння реалізується у вигляді функції, що обчислює праву частину, і ця функція передається як аргумент так званому солверу ЗДР (*ODE solver*). Крім того, вказується ряд параметрів та початкова умова для задачі Коші.

Щоб продемонструвати всю цю процедуру, продовжимо попередній приклад і вирішимо розглянуту задачу Коші чисельно, скориставшись солвером *ode*45, який реалізує метод Рунге-Кутти 4-5 порядку точності. Зауважимо, що чисельні методи рішення здебільшого вимагають представлення рівняння (системи рівнянь) як системи першого порядку. Це вимагатиме попередніх перетворень.

Наведений нижче код слід розуміти як продовження попереднього прикладу.

```
D2y = solve(eq, 'D2y');            % Вирішуємо щодо D2y
% Робимо заміну змінних
f2=subs(D2y,{sym('y'),sym('Dy')},{sym('y(1)'),sym('y(2)')})
% Починаємо створювати вміст майбутньої функції
s{1} = 'function dydx = MyEquation(x)'; % Заголовок функції
s{2} = 'dydx = zeros(2,1);';
s{3} = 'dydx(1) = y(2);';          % Перше рівняння системи
s{4} = ['dydx(2) = 'char(f2) ';']; % Друге рівняння системи
% Визначаємо повне ім'я файлу
filename = fullfile(pwd, 'MyEquation.m');
fid = fopen (filename, 'w');  % Відкриваємо файл на запис
fprintf(fid, %s\n, s{:});     % Записуємо вміст
fclose(fid);                  % Закриваємо файл
% Вирішуємо рівняння чисельно
xfinish = 5;                  % Задаємо кінцеву точку
xr = linspace(x0, xfinish, 20); % Створюємо масив абсцис
yinit = [y0; y10];            % Задаємо початкові умови
[xx, YY] = ode45('MyEquation', xr, yinit); % Вирішуємо ОДУ
% Рисуємо спочатку рішення, отримане аналітично за
% допомогою dsolve
% Створюємо масив координат для відображення рішення
xp1 = linspace(xo, xfinish);
yp1 = subs(Sol(1), sym('x'), xp1);
plot(xp1, yp1, 'b');          % Будуємо графік
hold on;                      % Заморожуємо полотно графіка
% Тепер будуємо графік чисельного рішення
plot(xx,YY(:,1),'r'); % Будуємо чисельне рішення щодо точок
hold off;             % Розморожуємо полотно графіка
% Задаємо заголовок графіка
```

```
title('\bfРозв'язання задачі Коші');
xlabel('\itx');           % Задаємо назви осей
ylabel('\ity');
xlim([x0, xfinish]);      % Задаємо межі по осі абсцис
delete (filename);        % Видаляємо не потрібний файл
```

Контрольні запитання:

1. Які існують функції для оголошення символьних змінних та констант у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?
2. Які існують засобу роботи з символьними виразами у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?
3. Які існують функції для розкриття дужок в символьних виразах у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?
4. Які існують функції для розкладання на множники символьних виразів у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?
5. Які існують функції для спрощення символьних виразів у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?
6. Які існують функції для підстановки змінних в символьних виразах у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?
7. Які існують функції для обчислення символьних виразів у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?
8. Які існують функції для відображення символьних виразів у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?
9. Які існують функції для розкладання в ряд Тейлора символьних виразів у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?
10. Які існують функції для визначення символьних виразів для сум у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?
11. Які існують функції для розв'язання рівнянь у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?

12. Які існують функції для визначення меж диференціювання та інтегрування у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?

13. Які існують функції для диференціювання та інтегрування у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?

14. Які існують засоби чисельного вирішення рівнянь, заданих у символьному вигляді у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?

РОЗДІЛ 8

ЗАСОБИ ОБРОБКИ ДАНИХ У СИСТЕМІ MATLAB

8.1. Поліноміальна апроксимація

Апроксимація – науковий метод, що полягає в заміні одних об'єктів іншими, в якомусь сенсі близькими до вихідних, але більш простими. Апроксимація дозволяє досліджувати числові характеристики та якісні властивості об'єкта, зводячи завдання до вивчення більш простих чи зручніших об'єктів.

Існує кілька методів поліноміальної апроксимації, які використовуються для наближення функцій поліномами. Ось деякі з найпоширеніших методів:

1. **Метод найменших квадратів**. Цей метод полягає у мінімізації суми квадратів відхилень між значеннями функції і апроксимуючим поліномом. Він є одним з найбільш поширених і використовується для розв'язання багатьох задач апроксимації.

2. **Метод Ньютона для інтерполяції**. Цей метод базується на поліномі Ньютона для апроксимації функцій. Він використовує інтерполяційний поліном ступеня n, який проходить через n+1 точок даних, і може бути використаний для апроксимації функції на всьому відрізку.

3. **Метод Лагранжа для інтерполяції**. Цей метод також базується на інтерполяційних поліномах, але використовує поліном Лагранжа, який може бути зручним у деяких випадках для апроксимації функцій.

4. **Метод Чебишева**. Цей метод використовує поліноми Чебишева, які є оптимальними в певному сенсі для апроксимації функцій. Він може забезпечити кращу апроксимацію, особливо на великому відрізку, ніж інші методи.

5. **Метод Гауса**. Цей метод використовує поліноми Гауса для апроксимації функцій. Вони дозволяють ефективно вирішувати задачі апроксимації на великих

відрізках та для функцій зі складними властивостями.

Розглянемо деякі докладніше.

Для поліноміальної апроксимації методом найменших квадратів у MATLAB використовується така функція:

***polyfit*(*x, y, n*)** – повертає вектор коефіцієнтів полінома $p(x)$ ступеня n, який найкраще апроксимує функцію $y(x)$. Результатом є вектор рядок довжиною $n+1$, що містить коефіцієнти полінома в порядку зменшення ступенів. Якщо число елементів векторів x і y дорівнює $n + 1$, то реалізується звичайна поліноміальна апроксимація, при якій графік полінома точно проходить через вузлові точки з координатами (x, y), що зберігаються у векторах x і y.

[*p, s*] = *polyfit*(*x, y, n*) – повертає вектор p з коефіцієнтами апроксимуючого полінома та структуру S для використання разом із функцією *polyval* для оцінки або передбачення похибки.

Приклад поліноміальної апроксимації функції $sin(x)$:

```
x=(-3:0.2:3)';
y=sin(x);
p=polyfit(x,y,3)
P =
    -0.0953 0.0000 0.8651 -0.0000
x=(-4:0.2:4)';
y=sin(x);
f=polyval(p,x);
plot(x,y,'o',x,f)
```

Побудований за цим прикладом рис. 8.1, дає наочне уявлення про точність поліноміальної апроксимації. Графік апроксимуючого полінома третього ступеня на рис. 8.1 показаний суцільною лінією, а точки вихідної залежності позначені кружками.

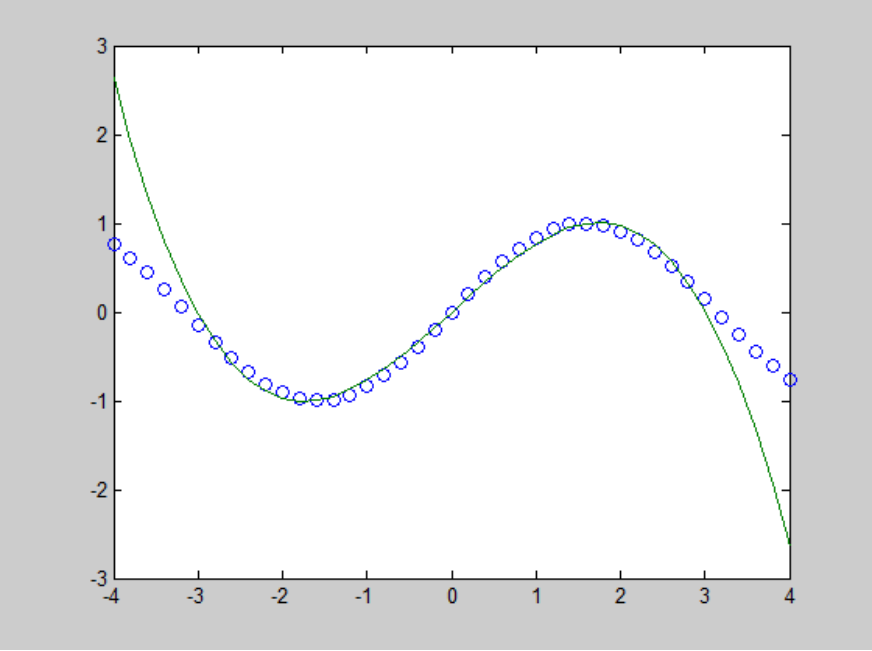


Рис. 8.1. Приклад використання функції *polyfit*.

8.2. Інтерполяція на нерівномірній сітці

Інтерполяція – в обчислювальній математиці знаходження невідомих проміжних значень певної функції, за наявним дискретним набором відомих значень, певним способом.

Для інтерполяції на нерівномірній сітці використовується функція *griddata*:

ZI = griddata(x, y, z, XI, YI) – повертає апроксимуючу матрицю поверхні виду $z = f(x,y)$, яка визначається векторами (x, y, z) з рівномірно або нерівномірно розподіленими елементами. Функція *griddata* апроксимує цю поверхню у точках, визначених векторами (*XI*, *YI*) у вигляді значень *ZI*. *XI* та *YI* зазвичай формують однорідну сітку (створену за допомогою функції *meshgrid*). *XI* може бути вектор-рядком, у цьому випадку він визначає матрицю з постійними стовпцями. Так само *YI* може бути вектор-стовпцем і тоді він визначає матрицю з постійними рядками.

[XI, YI, ZI] = griddata(x, y, z, xi, yi) – повертає апроксимуючу матрицю *ZI*, як описано вище, а також повертає матриці *XI* та *YI*, сформовані з вектор-стовпця *xi* і вектор-рядка *yi*, які аналогічні матрицям, що повертаються функцією *meshgrid*.

[...] = *griddata*(..., *method*) – використовує певний метод інтерполяції. Тут і в наступних формулах метод визначає тип апроксимуючої поверхні:

'*nearest*' – ступінчаста інтерполяція;

'*linear*' – лінійна інтерполяція (прийнята за умовчанням);

'*spline*' – кубічна сплайн-інтерполяція;

'*cubic*' – кубічна інтерполяція.

Метод '*cubic*' формують гладкі поверхні, в той час як '*linear*' і '*nearest*' мають розриви першої похідної та функції відповідно. Приклад:

```
x=rand(120,1)*4-2;
y=rand(120,1)*4-2;
z=x.*y.*exp(-x.^2-y.^2);
t=-2:0.1:2;
[X,Y]=meshgrid(t,t);
Z=griddata(x,y,z,X,Y);
mesh(X,Y,Z)
hold on
plot3(x,y,z,'ok')
```

Рис. 8.2 ілюструє застосування функції *griddata*. Цей приклад є ілюстрацією ефективності графічних засобів системи MATLAB. У ньому показано побудову вузлових точок-кружків на поверхні з функціональним забарвленням ліній її каркасу.

8.3. Одновимірна інтерполяція за таблицею

Для одновимірної інтерполяції таблично заданих функцій використовується процедура:

***interp*1(*x, Y, xi*)** – повертає вектор *yi*, що містить елементи, відповідні елементам *xi* та отримані інтерполяцією векторів *x* та *Y*. Вектор *x* визначає абсциси точок, в яких задані значення *Y*. Якщо *Y* – матриця, то інтерполяція виконується для кожного стовпця *Y* та *yi*, який має довжину *length*(*xi*) – *by* – *size*(*Y*, 2).

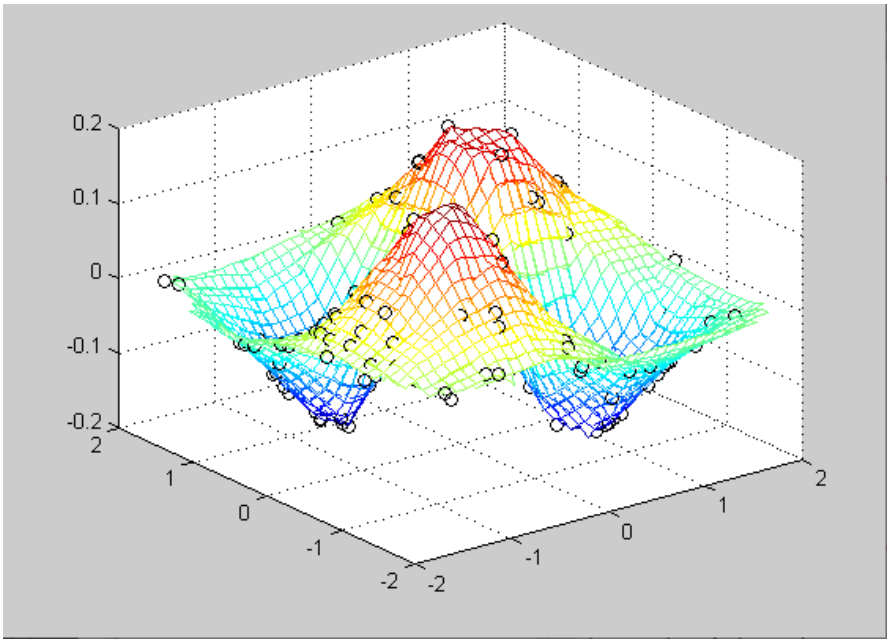


Рис. 8.2 Застосування функції *griddata*.

***interp*1(*x, Y, xi, method*)** – робить те саме, але дозволяє задати метод інтерполяції.

Усі методи інтерполяції вимагають, щоб значення *x* змінювалися монотонно. Для швидкої інтерполяції, коли *x* – вектор рівномірно розподілених точок, краще використовувати методи *'*linear', '*cubic', '*nearest', або '*spline'*. Зверніть увагу, що в цьому випадку найменуванню методу передує знак зірочки. Приклад (інтерполяція значень косинуса):

```
x=0:10;
y=cos(x);
xi=0:0.1:10;
yi=interp1(x,y,xi);
plot(x,y,'x',xi,yi,'g')
hold on
yi=interp1(x, y, xi, 'spline');
plot(x,y,'o',xi,yi,'m')
grid
hold off
```

Результати інтерполяції ілюструє рис. 8.3.

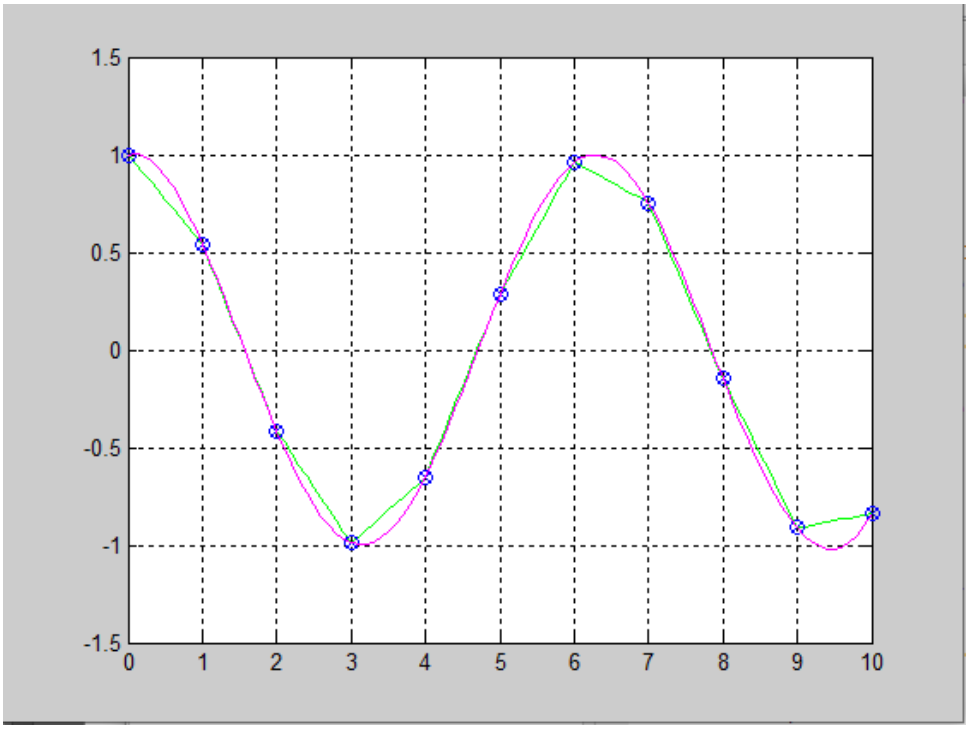


Рис. 8.3. Приклад застосування функції *interp*1.

Вузлові точки на рис. 8.3 позначені кружками з хрестиками. Одна з кривих відповідає лінійній інтерполяції, інша - сплайн-інтерполяції. Неважко відзначити, що сплайн-інтерполяція в даному випадку дає кращі результати, ніж лінійна інтерполяція. При останній точці просто з'єднуються один з одним відрізками прямих, так що графік інтерполюючої кривої при лінійній інтерполяції виходить не гладким.

8.4. Двовимірна інтерполяція за таблицею

Двовимірна інтерполяція значно складніше, ніж одномірна, розглянута вище. Для двовимірної інтерполяції таблично заданих функцій використовується процедура *interp*2:

***interp*2(*X, Y, Z, XI, YI*)** – повертає матрицю *ZI*, що містить елементи, відповідні елементам *XI* і *YI* та отриману інтерполяцією двовимірної функції,

заданої матрицями *X, Y* та *Z. X* та *Y* повинні бути монотонними та мати той самий формат (“*plaid*”), якби вони були отримані за допомогою функції *meshgrid*. Матриці *X* та *Y* визначають точки, в яких задано значення *Z. XI* та *YI* можуть бути матрицями, у цьому випадку *interp*2 повертає значення масиву *Z*, що відповідають точкам (*XI*(*i, j*), *YI*(*i, j*)). В якості альтернативи можливо передати вектор-рядок і вектор-стовпець *xi* і *yi* відповідно. У цьому випадку *interp*2 представляє ці вектори, ніби використовувалася команда *meshgrid*(*xi, yi*).

ZI = interp2(Z, XI, YI) – має на увазі, що *X* = 1 : *n* і *Y* = 1 : *m*, де [*m, n*] = *size* (*Z*).

ZI = interp2(Z, ntimes) – здійснює інтерполяцію рекурсивним методом із числом кроків *ntimes*.

ZI = interp2 (X, Y, Z, XI, YI, method) – дозволяє задати метод інтерполяції.

Усі методи інтерполяції вимагають, щоб *x* і *y* змінювалися монотонно і мали такий самий формат (“*plaid*”), як і функції *meshgrid*. Для більш швидкої інтерполяції, коли *x* і *y* – рівномірно розподілені, краще використовувати методи *'*linear', '*cubic', або '*nearest'*. Приклад:

```
[X,Y]=meshgrid(-3:0.25:3);
Z=peaks(X/2,Y*2);
[X1 ,Y1]=meshgrid(-3:0.1:3);
Z1 =interp2(X,Y,Z,X1,Y1);
mesh(X,Y,Z)
hold on
mesh(X1,Y1,Z1+15)
hold off
```

Рис. 8.4 ілюструє застосування функції *interp*2 для двовимірної інтерполяції (з прикладу функції *peaks*). В даному випадку поверхня знизу - двомірна лінійна інтерполяція, яка реалізується за умовчанням, коли не вказано параметр *method*.

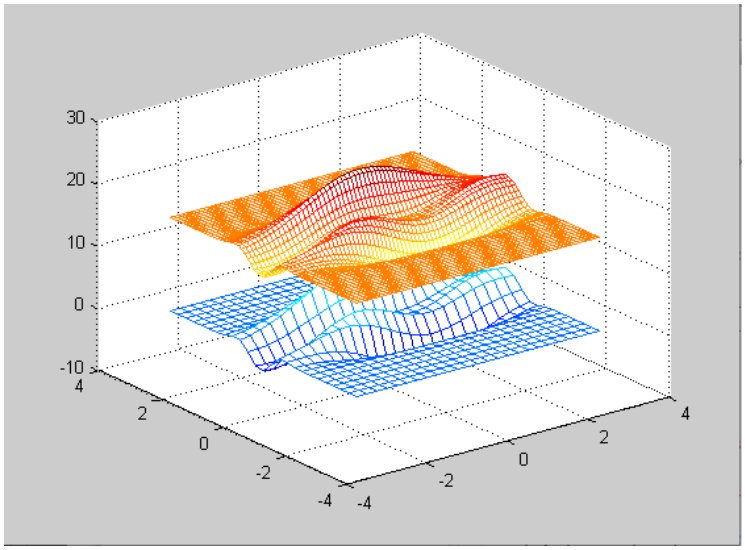


Рис 8.4. Застосування функції *interp*2.

8.5. Тривимірна інтерполяція за таблицею

Для тривимірної інтерполяції таблично заданих функцій використовується процедура:

***interp*3(*X, Y, Z, V, XI, YI, ZI*) –** повертає вектор інтерполяції *VI* як значення основної тривимірної функції *V* у точках матриць *XI, YI* та *ZI*. Матриці *X, Y* і *Z* визначають точки, в яких задано значення *V*. *XI, YI*, і *ZI* можуть бути матрицями, у цьому випадку *interp*3 повертає значення *Z*, що відповідають точкам (*XI*(*i, j*), *YI*(*i, j*), *ZI*(*i, j*)).

***VI = interp*3(*V,XI,YI,ZI*)** – має на увазі *X* =1 : *N*, *Y* = 1 : *M*, *Z* = 1 : *P*, де [*M, N, P*] = *size*(*V*).

***VI = interp*3(*V, ntimes*)** – здійснює інтерполяцію рекурсивним методом з числом кроків *ntimes*.

***VI = interp*3(*..., method*)** – дозволяє задати метод інтерполяції. Усі методи інтерполяції вимагають, щоб дані *X, Y* і *Z* змінювалися монотонно і мали такий самий формат (“*plaid*”), як і функції *meshgrid*. Для більш швидкої інтерполяції, коли *X* і *Y* і *Z* – рівномірно розподілені та монотонні, краще використовувати методи *'*linear', '*cubic'* або *'*nearest'*.

8.6. *N*-мірна інтерполяція за таблицею

MATLAB дозволяє навіть виконати інтерполяцію функцій, заданих у вигляді *n*-мірної таблиці. Для цього використовується процедура:

***interpn*(*X*1, *X*2, *X*3, ..., *V*, *Y*1, *Y*2, *Y*3, ...)** – повертає вектор інтерполяції *VI*, що представляє значення основної багатовимірної функції *V* у точках масивів *Y*1, *Y*2, *Y*3,... Для багатовимірної функції *V* має бути зазначено 2×*N*+1 аргументів, де *N* визначає розмірність. Масиви *X*1, *X*2, *X*3, ... задають точки, в яких задано значення *V*. *Y*1, *Y*2, *Y*3, ... можуть бути матрицями, у цьому випадку *interpn* повертає значення *VI*, що відповідають точкам (*Y*l(*i*, *j*),*Y*2(*i*, *j*), *Y*3(*i*, *j*), ...).

***VI* = *interpn*(*V*, *Y*1, *Y*2, *Y*3, ...)** – має на увазі *X*1 = l : *size*(*V*, l), *X*2 = l : *size*(*V*, 2), *X*3 = 1 : *size*(*V*, 3).

***VI* = *interpn*(*V*, *ntimes*)** – здійснює інтерполяцію рекурсивним методом з числом кроків *ntimes*.

***VI* = *interpn* (..., *method*)** – дозволяє вказати метод інтерполяції.

Через рідкісне використання такого виду інтерполяції приклади її застосування не наводяться.

8.7. Інтерполяція кубічним сплайном

У MATLAB сплайн-інтерполяція реалізується такою функцією:

***yi* = *spline*(*x*, *y*, *xi*)** – використовує вектори *x* і *y*, що містять значення функції в точках *x*, і вектор *xi*, що задає нові точки для знаходження елементів вектора *yi*, використовуючи кубічну сплайн-інтерполяцію.

***pp* = *spline*(*x*, *y*)** – повертає *pp*-форму сплайну, використовувану у функції *ppval* та інших сплайн-функціях.

Приклад (см. рис. 8.5):

```
x=0:10;
y=3*cos(x);
x1=0:0.1:11;
y1=spline(x,y,x1);
plot(x,y,'o',x1,y1,'--')
```

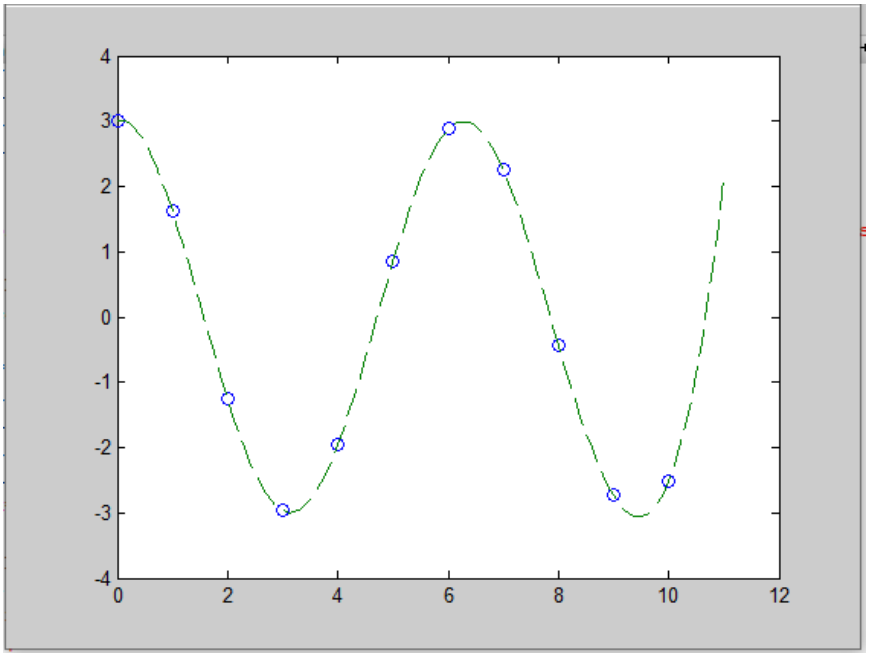


Рис. 8.5. Приклад застосування функції *spline*.

Сплайн-інтерполяція дає непогані результати для функцій, що не мають розривів та різких перепадів. Особливо хороші результати дає сплайн-інтерполяція для монотонних функцій.

У *Toolbox* системи MATLAB входить пакет розширення *Spline Toolbox* 2.0, що містить близько 70 додаткових функцій, що відносяться до реалізації сплайн-інтерполяції та апроксимації, а також графічного представлення сплайнами. Для виклику даних про цей пакет використовуйте команду *help splines*.

8.8. Засоби прямого перетворення Фур'є

MATLAB містить функції для виконання швидкого одновимірного та двовимірного дискретного перетворення Фур'є. Для одновимірного перетворення використовується така функція:

***fft*(*X*)** – повертає вектор швидкого перетворення Фур'є (ШПФ), використовуючи, як дані, елементи вектору *X*. Якщо *X*-матриця, то повертається результат ШПФ для кожного стовпця матриці.

***fft*(*X*, *n*)** – повертає ШПФ у *n* точках. Якщо довжина вектору *X* менше *n*, то елементи, що відсутні, заповнюються нулями. Якщо довжина *X* більша за *n*, то зайві елементи видаляються. Коли *X* – матриця, довжина стовпців коригується аналогічно.

***fft*(*X*, [ ], *dim*)** та ***fft*(*X*, *n*, *dim*)** – застосовують ШПФ до однієї з розмірностей масиву в залежності від величини *dim*.

Для ілюстрації застосування перетворення Фур'є створимо тричастотний сигнал на тлі сильного шуму, що створюється генератором випадкових чисел:

```
t=0:0.0005:1;
x=sin(2*pi*200*t)+0.8*sin(2*pi*150*t)+0.8*sin(2*pi*250*t);
y=x+2*randn(size(t));
plot(y(1:100),'b')
```

Цей сигнал має середню частоту 200 рад/с і два бічні сигнали з частотами 150 і 250 рад/с, що відповідає амплітудно-модульованому сигналу з частотою модуляції 50 рад/с і глибиною модуляції 0.8 (амплітуда бічних складових 0.8 від амплітуди центрального сигналу). З графіку сигналу (рис. 8.6) не видно, що корисний сигнал є амплітудно-модульованими коливаннями. Тому, що він місить шуми.

Тепер побудуємо графік спектральної щільності даного сигналу за допомогою прямого перетворення Фур'є, що по суті переводить часове подання сигналу (рис. 8.7) в частотне:

```
Y=fft(y,1024);
Pyy=Y.*conj(Y)/1024;
f=1000*(0:300)/1024;
plot(f,Pyy(1:301)),grid
```

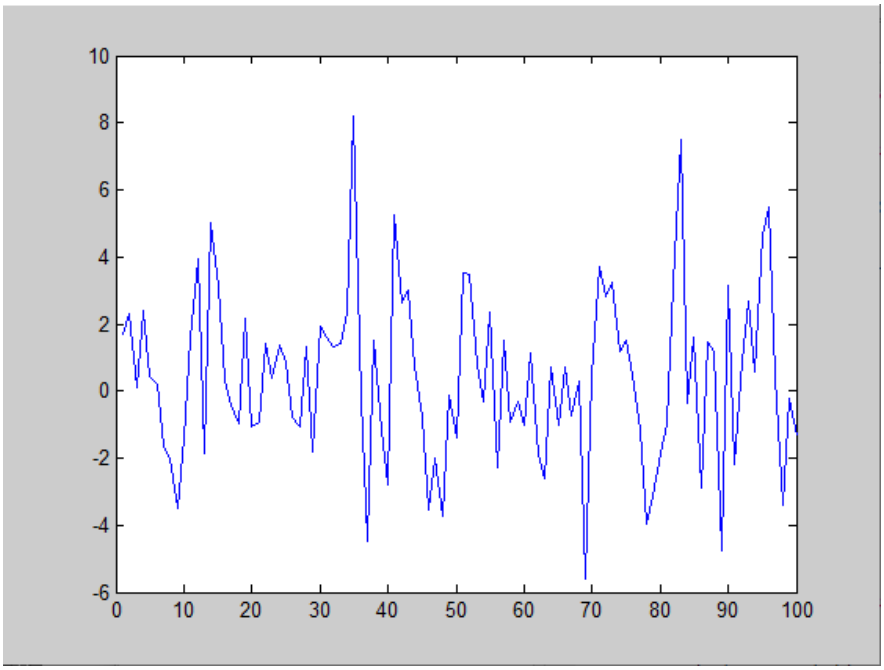


Рис. 8.6. Вигляд зашумленого сигналу.

Графік спектральної щільності сигналу, показаного на рис. 8.6, представлений на рис. 8.7.

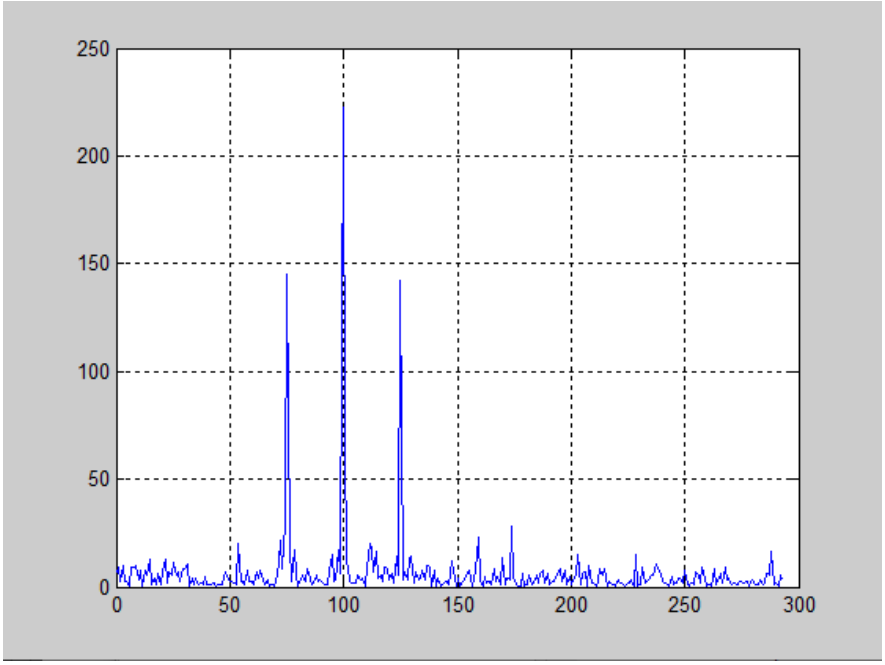


Рис. 8.7. Графік спектральної щільності сигналу.

З рис. 8.7 видно, що спектрограма сигналу має найбільш сильно виражений пік на середній частоті амплітудно-модульованого сигналу і два бічних піка. Всі ці три частотні складові сигналу добре виділяються на загальному тлі шуму. Таким чином, даний приклад наочно ілюструє техніку виявлення слабких сигналів на тлі шумів, що лежить в основі радіоприймальних пристроїв.

Для прямого двовимірного перетворення Фур'є використовується функція *fft*2:

***fft*2(*X*) –** повертає результат двовимірного ШПФ масиву даних *X*.

***fft*2(*X, m, n*)** – усікає масив *X* чи доповнює його нулями, щоб створити $m \times n$ матрицю перед виконанням перетворення Фур'є. Результат – матриця розміру $m \times n$.

Для прямого багатовимірного перетворення Фур'є також існує функція:

***fftn*(*X*)** - повертає результат *N*-вимірного дискретного перетворення для масиву *X* розміру *N*. Якщо *X* – вектор, вихід буде мати ту ж орієнтацію.

***fftn*(*X, SIZ*)** – повертає результат дискретного перетворення масиву *X* з обмеженням розміру, заданим змінної *SIZ*.

Функція ***Y* = *fftshift*(*X*)** перегруповує вихідні масиви функції *fft* і *fft*2, розміщуючи нульову частоту в центрі спектру, що іноді зручніше. Якщо *X* – вектор, то *Y* – вектор із циклічною перестановкою правої та лівої половини вихідного вектора. Якщо *X* – матриця, то *Y* – матриця, у якої квадранти I та III змінюються місцями з квадрантами II та IV. Розглянемо наступний приклад. Спочатку збудуємо графік спектральної щільності (рис. 8.8) при одномірному перетворенні Фур'є:

```
rand('state',0);
t=0:0.001:0.6;
x=sin(2*pi*50*t)+sin(2*pi*120*t);
y=x+2*randn(size(t))+0.3;
Y=fft(y);
Pyy=Y.*conj(Y)/512;
```

```
f=1000*(0:255)/512;
plot(f,Pyy(1:256))
grid
```

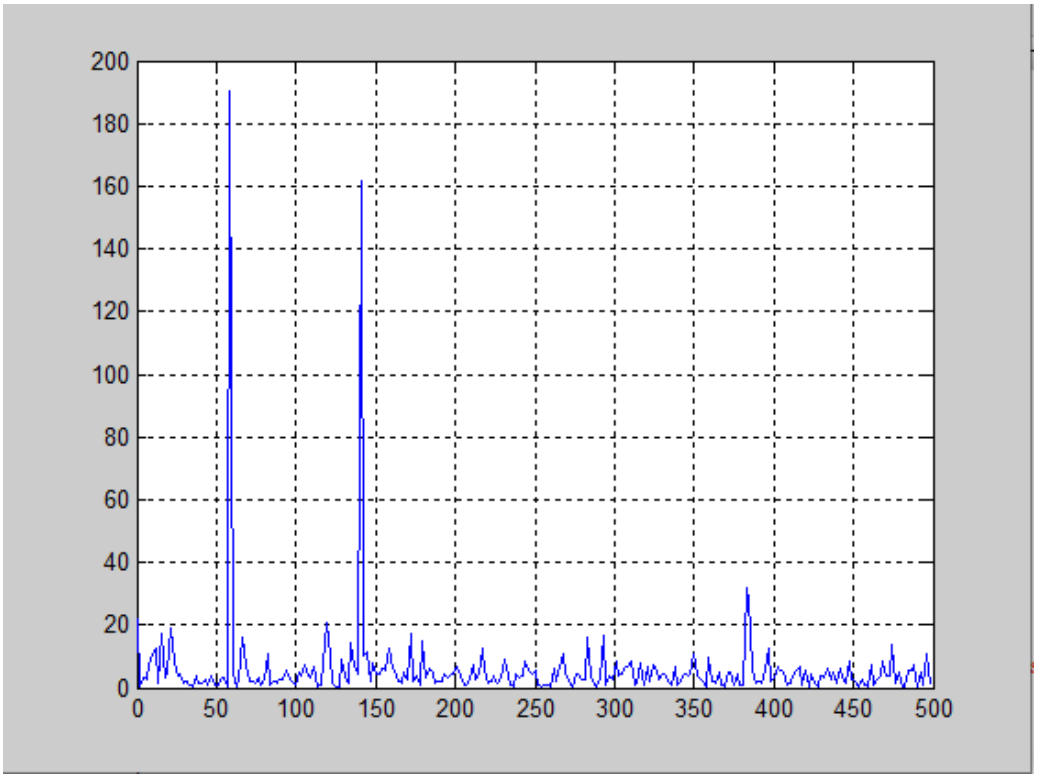


Рис. 8.8. Графік спектральної щільності сигналу після одномірного перетворення Фур'є.

Тепер скористаємось функцією *fftshift*:

```
Y=fftshift(Y);
Pyy=Y.*conj(Y)/512;
plot(Pyy),grid
```

Отриманий при цьому графік представлений на рис. 8.9.

8.9. Засоби зворотного перетворення Фур'є

В одновимірному випадку можливе зворотне перетворення Фур'є, яке реалізується наступною функцією:

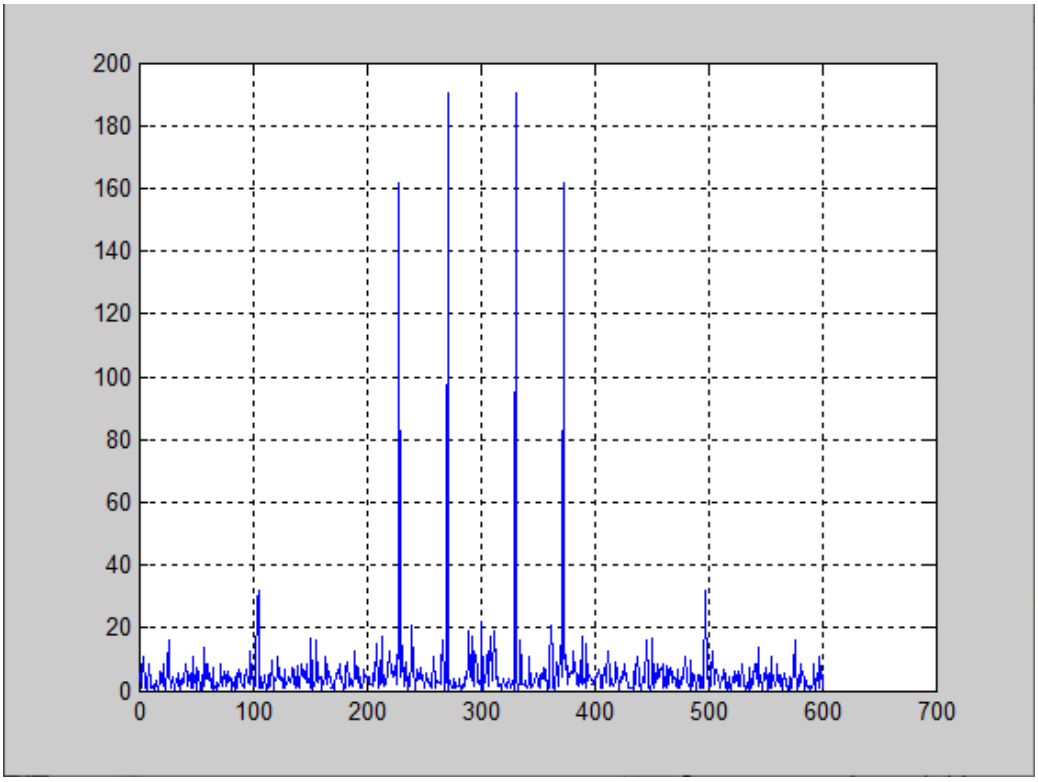


Рис. 8.9. Графік спектральної щільності того ж сигналу після застосування функції *fftshift*.

***ifft*(*F*)** – повертає результат ШПФ для даних вектору *F*. Якщо *F* – матриця, то *ifft* повертає зворотне перетворення Фур'є кожного стовпця цієї матриці.

***ifft*(*F*, *n*)** – повертає *n*-точкове швидке дискретне перетворення Фур'є вектору *F*.

***ifft*(*F*, [ ], *dim*)** і ***y* = *ifft*(*X*, *n*, *dim*)** – повертають зворотне дискретне перетворення Фур'є масиву *F* рядками або стовпцями залежно від значення скаляра *dim*.

Для будь-якого *X* *ifft*(*fft* (*x*)) дорівнює *x* з урахуванням похибки округлення. Якщо *x* – дійсний, *ifft*(*fft*(*x*)) може мати малі уявні частини. Приклад:

```
V=[1 1 1 1 0 0 0 0];
fft(V)
ans =
Columns 1 through 4
```

```
4.0000   1.0000 - 2.4142i      0    1.0000 - 0.4142i
Columns 5 through 8
0        1.0000 + 0.4142i      0    1.0000 + 2.4142i
ifft(fft(V))
ans =
1.0000    1.0000    1.0000    1.0000   0   0   0    0
```

Аналогічні функції є для двовимірного та багатовимірного випадків:

***ifft2*(*F*)** – повертає двовимірне дискретне зворотне швидке перетворення Фур'є для матриці *F*.

***ifft2*(*F*, *m*, *n*)** – повертає m×n зворотне швидке перетворення Фур'є для матриці *F*.

***ifftn*(*F*)** – повертає результат *A*-мірного зворотного дискретного перетворення Фур'є для *A*-мірного масиву *F*.

***ifftn* (*F*, *SIZ*)** - повертає результат зворотного дискретного перетворення для масиву *F* з обмеженням розміру, заданим змінною *SIZ*.

8.10. Функції згортки

У розділі розглянуті базові засоби щодо операцій згортки і фільтрації сигналів з урахуванням алгоритмів швидкого перетворення Фур'є. Багато додаткових операцій, що стосуються цієї області обробки сигналів, можна знайти в пакеті прикладних програм *Signal Processing Toolbox*.

Для двох векторів *x* і *y* визначено операцію згортки:

$$z(k) = \sum_{j=\max(1,k-n+1)}^{\min(k,m)} x(j)y(k-j+1).$$

В результаті її застосування виходить вектор *z* із довжиною (*m*+*n*+1), який на підставі відомої теореми про згортку можна вважати рівним *ifft*(*x*.**y*). Для здійснення згортки використовується функція *conv*(*x*, *y*).

Зворотня згортці функція визначена як [*q, r*] = *deconv*(*z, x*). Вона фактично визначає імпульсну характеристику фільтра. Якщо *z* = *conv*(*x, y*), то *q* = *y*, *r* = 0. Якщо *x* і *y* – вектори з коефіцієнтами поліномів, то згортка еквівалентна перемноженню поліномів.

Для двомірних масивів також існує функція згортки:

Z = *conv*2(*X, Y*) та *Z* = *conv*2(*X, Y,* '*option*').

Для двовимірних масивів *X* і *Y* з розміром *mx*×*nx* і *my*×*ny*, відповідно, результат двовимірної згортки породжує масив розміру (*mx*+*my*–1)×(*nx*+*ny*–1). У другій формі функції опція може мати таке значення:

'*full*' – повнорозмірна згортка (використовується за умовчанням),

'*same*' – центральна частина розміру mx×nx,

'*valid*' – центральна частина розміру (*mx*+*my*–1)×(*nx*+*ny*–1), якщо (*mx*×*nx*) > (*my*×*ny*).

8.11. Функції фільтрації

Для забезпечення дискретної одновимірної фільтрації може використовуватися функція *filter* у таких формах запису:

***filter*(*B, A, X*)** – фільтрує одновимірний масив даних *X*, використовуючи дискретний фільтр, що описується звичайним кінцево-різницевим рівнянням:

```
a(1)*y(n) = b(1)*x(n) + b(2)*x(n-1) + ... +
+b(nb+1)*x(n-nb) - a(2)*y(n-1) - ... - a(na+1)*y(n-na).
```

Якщо *a*(1) не дорівнює 1, то коефіцієнти рівняння нормалізуються щодо *a*(1). Коли *X* – матриця, *filter* оперує зі стовпцями *X*. Можлива фільтрація багатовимірного (розмірності *N*) масиву.

[*Y, Zf*]=filter(*B, A, X, Zi*) – виконує фільтрацію з урахуванням запізнення вхідного *Zi* та вихідного *Zf* сигналів.

***filter*(B, *A*, *X*, [ ], *DIM*)** або ***filter*(*B*, *A*, *X*, *Zi*, *DIM*)** – працюють у напрямку розмірності *DIM*.

Розглянемо типовий приклад фільтрації гармонійного сигналу і натомість інших сигналів – файл з ім'ям *filtdem.m* з пакета розширення *Signal Processing Toolbox*. На рис. 8.10 представлений кадр прикладу, на якому показано формування вхідної сукупності сигналів у вигляді трьох сигналів із частотами 5, 15 та 30 Гц. Показана часова залежність сигналу та фрагмент команд, які треба виконати для цього етапу перетворень.

```
Fs = 100;
t = (1:100)/Fs;
s1 = sin(2*pi*t*5);
s2=sin(2*pi*t*15);
s3=sin(2*pi*t*30);
s = s1+s2+s3;
plot(t,s);
xlabel('Time (seconds)');
ylabel('Time waveform');
```

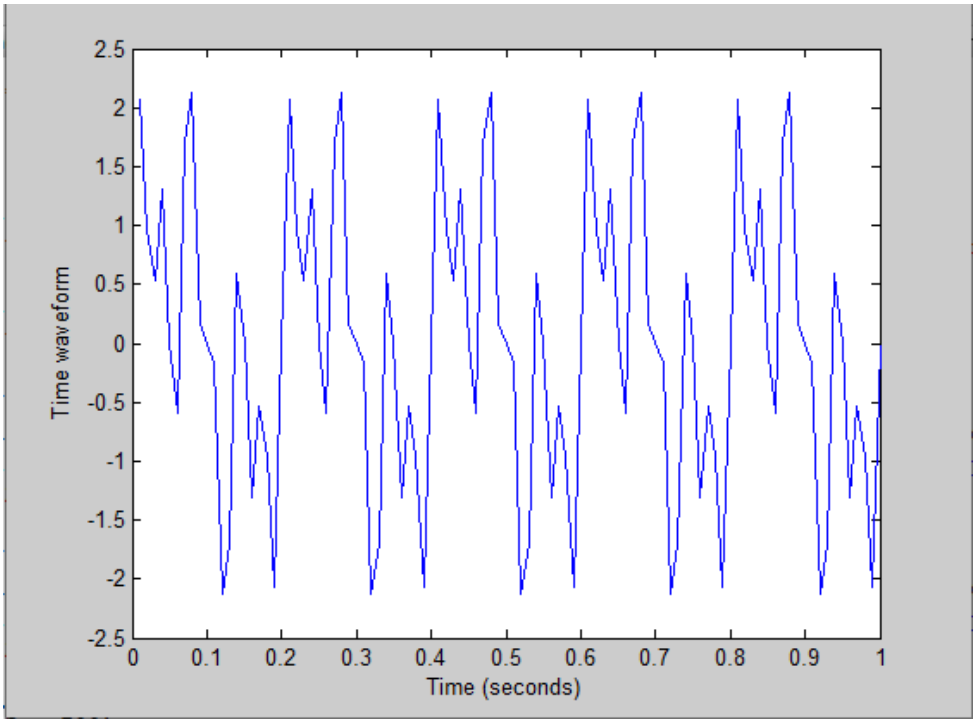


Рис. 8.10 Формування сигналу.

Рис. 8.11 ілюструє конструювання фільтра з досить плоскою вершиною амплітудно-частотної характеристики (АЧХ) та смугою частот, що забезпечує виділення сигналу з частотою 15 Гц та пригнічення сигналів із частотами 5 та 30 Гц. Для формування смуги пропускання фільтра використовується функція *ellip*, а для побудови АЧХ – функція *freqz* з пакету *Signal Processing Toolbox*. Це дозволяє побудувати графік АЧХ створеного фільтра.

```
[b,a] = ellip(4,0.1,40,[10 20]*2/Fs);
[H,w] = freqz(b,a,512);
plot(w*Fs/(2*pi),abs(H));
xlabel('Frequency (Hz)');
ylabel('Mag. of frequency response');
grid;
```

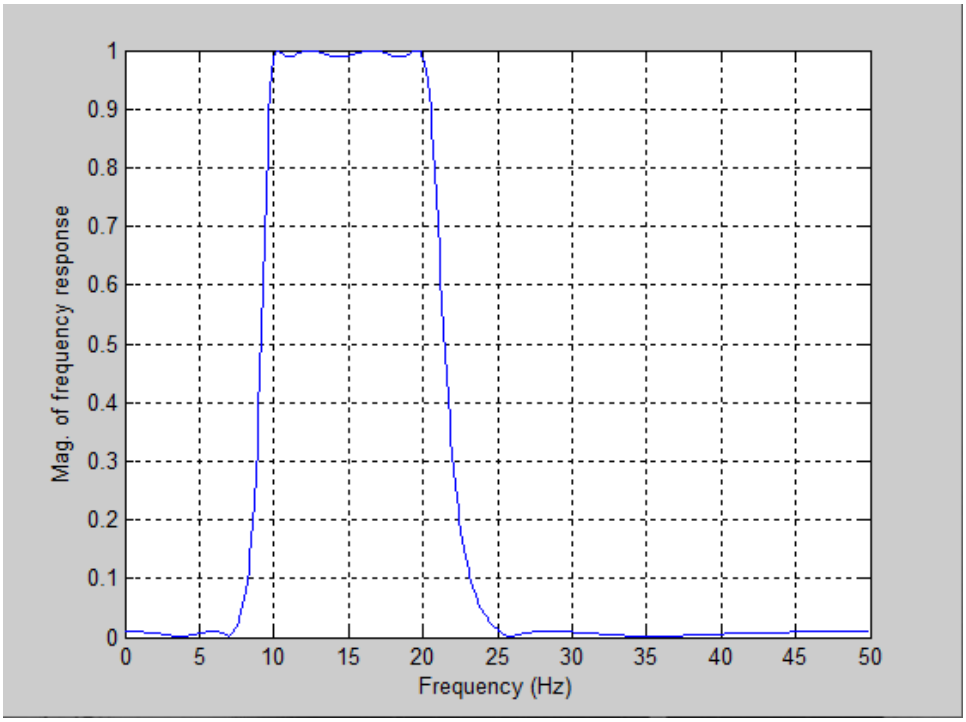


Рис. 8.11 Конструювання фільтра із заданою смугою пропускання та його АЧХ

Рис. 8.12 ілюструє ефективність виділення сигналу заданої частоти (15 Гц) за допомогою операції фільтрації – функції *filter*, описаної вище. Можна помітити дві обставини – отриманий стаціонарний сигнал практично синусоїдальний, що

свідчить про високий рівень фільтрації побічних сигналів. Однак наростання сигналу у часі йде досить повільно і займає кілька періодів частоти корисного сигналу. Характер наростання сигналу у часі визначається перехідною характеристикою фільтра.

```
sf = filter(b,a,s);
plot(t,sf);
xlabel('Time (seconds)');
ylabel('Time waveform');
axis([0 1 -1 1]);
```

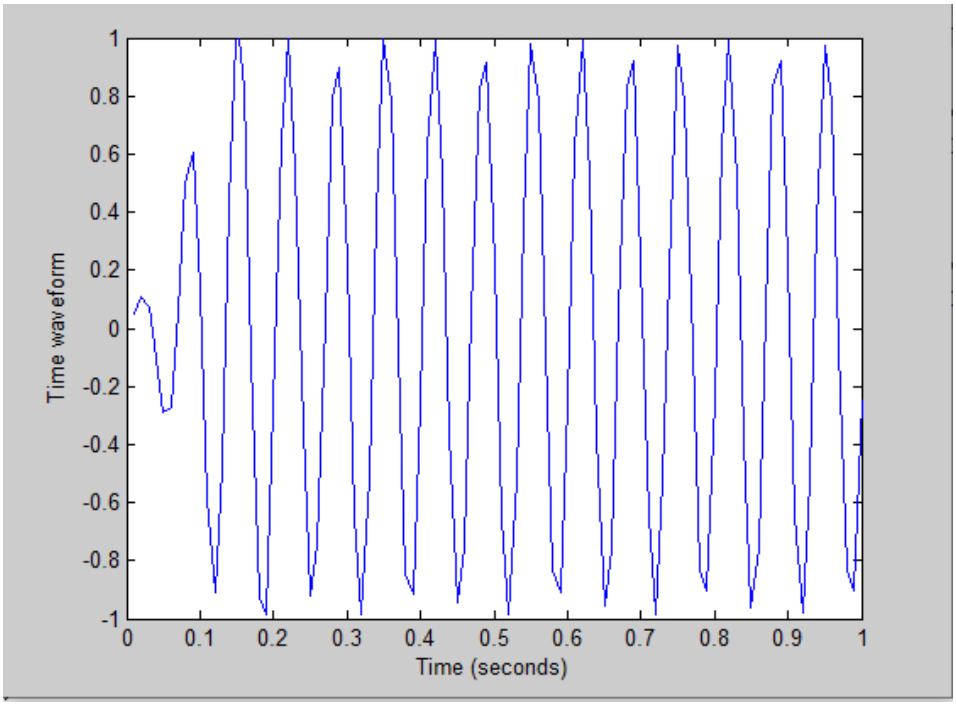


Рис. 8.12. Результат фільтрації у вигляді часової залежності сигналу на виході фільтра.

Рис. 8.13 показує спектр вихідного сигналу та спектр сигналу на виході фільтра (він показаний лініями іншого кольору). Для побудови спектрів використовується пряме перетворення Фур'є – функція *fft*.

```
S = fft(s,512);
SF = fft(sf,512);
```

```
w = (0:255)/256*(Fs/2);
plot(w,abs([S(1:256)' SF(1:256)']));
xlabel('Frequency (Hz)');
ylabel('Mag. of Fourier transform');
grid;
legend({'before','after'})
```

Цей приклад наочно ілюструє техніку фільтрації.

Для здійснення двовимірної фільтрації служить функція:

filter2(B, X) – фільтрує дані у двомірному масиві *X*, використовуючи дискретний фільтр, описаний матрицею *Y*. Результат *Y* має такий же розміри, як і *X*.

filter2(B, X, 'option') – виконує те саме, але з опцією, що впливає на розмір масиву *Y*: '*same*' – *size*(*Y*) = *size*(*X*) (діє за замовчуванням), '*valid*' – *size*(*Y*) < *size*(*X*), '*full*' – *size*(*Y*) > *size*(*X*).

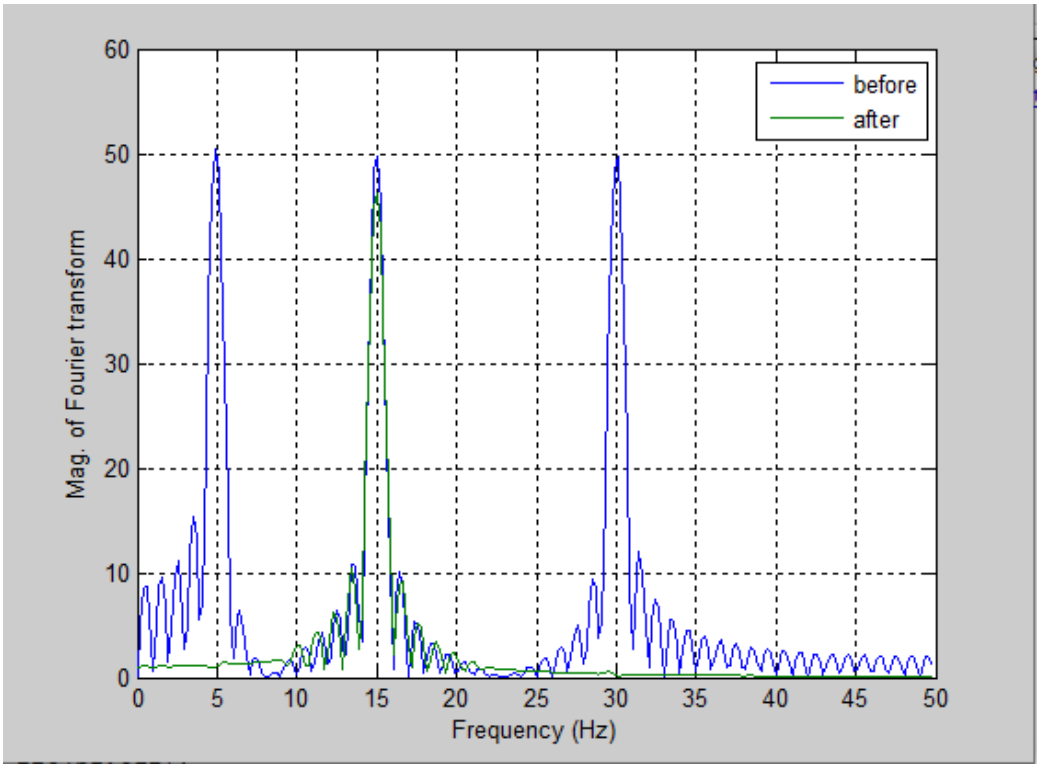


Рис. 8.13. Аналіз спектрів сигналів на вході та виході фільтра та побудова їх спектрів.

8.12. Корекція фазових кутів

Фазові кути одновимірних масивів реагують на розриви під час переходу через значення, кратні π. Функція *unwrap*(*P*) і *unwrap*(*P*, *cutoff*) усуває цей недолік одномірного масиву *P*, доповнюючи значення кутів у точках розриву значеннями ±2π. Якщо *P* – двомірний масив, то ця функція застосовується до стовпців. Параметр *cutoff* (за замовчуванням дорівнює π) дозволяє призначити будь-який критичний кут точок розриву. Функція використовується при побудові фазочастотних характеристик (ФЧХ) фільтрів.

Контрольні запитання:

1. Які існують методи поліноміальної апроксимації та чим вони розрізняються?
2. Які існують засоби для поліноміальної апроксимації методом найменших квадратів у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?
3. Які існують засоби для інтерполяції на нерівномірній сітці у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?
4. Які існують засоби для одновимірної інтерполяції за таблицею у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?
5. Які існують засоби для двовимірної інтерполяції за таблицею у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?
6. Які існують засоби для тривимірної інтерполяції за таблицею у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?
7. Які існують засоби для *N*-мірної інтерполяції за таблицею у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?

8. Які існують засоби для інтерполяції кубічним сплайном у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?

9. Які існують функції для прямого перетворення Фур'є у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?

10. Які існують функції для зворотного перетворення Фур'є у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?

11. Які існують функції згортки у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?

12. Які існують функції фільтрації у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?

13. Які існують функції для корекції фазових кутів у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?

РОЗДІЛ 9

ЧИСЛОВЕ РІШЕННЯ ЗАДАЧ ОПТИМІЗАЦІЇ

У процесі проектування та експлуатації автоматизованих систем управління регулярно виникають різноманітні завдання оптимізації, що визначають якість технологічних процесів. Поряд з цим сучасні технології характеризуються різноманітністю, складністю, підвищеною вимогою до якості продукції, що випускається, економічністю та надійностю роботи, гнучкістю системи. Як наслідок, зростають вимоги до алгоритмів управління та складність завдання проектування самої АСУ. Це робить невід'ємним використання різних САПР на вирішення завдань оптимізації.

Ця лекція присвячена опису пакета MATLAB *Optimization Toolbox* та його застосування у задачах проектування систем автоматичного управління. У рамках викладеного матеріалу розглянуто загальні аспекти використання пакета, такі як вибір цільових функцій та завдання обмежень, а також наведено огляд та приклади використання найпоширеніших функцій.

Optimization Toolbox надає широкий набір засобів для чисельного розв'язання задач оптимізації. Такі завдання розробниками було поділено на чотири групи:

- мінімізація,
- багатоцільова мінімізація,
- пошук рішень рівнянь,
- мінімізація квадратів (правдоподібність траєкторій).

Для кожного з цих завдань існує набір функцій, що реалізують той чи інший метод вирішення, поряд з цим кожна функція передбачає свою сферу застосування і свій набір параметрів. Вибір викладеного матеріалу обумовлений колом розв'язуваних у процесі завдань. В рамках лекції передбачається розглянути лише

загальні для всіх методів аспекти використання пакета, такі як вибір цільових функцій та завдання обмежень, а також навести огляд та приклади використання найпоширеніших функцій.

9.1. Завдання цільових функцій

Формулювання проблеми в задачах мінімізації найчастіше виглядає так min $f(x)$, де x – безліч можливих значень цільова функція $f(x)$, на які можуть бути накладені обмеження. Найчастіше цільова функція є скалярною, та її аргумент – скаляр чи вектор. Однак для деяких завдань мультиоб'єктної мінімізації, пошуку розв'язків рівнянь або мінімізації сум квадратів застосовується векторна або матрична цільова функція $f(x)$.

Більшість інструментів *Optimization Toolbox* розраховані на вирішення задачі мінімізації цільової функції. Якщо стоїть завдання максимізації деякого критерію, то розумним є використання додаткової цільової функції $g(x) = -f(x)$, яку потрібно мінімізувати.

Перейдемо до розгляду скалярної цільової функції, що задається у вигляді *m*-файлу, що містить однойменну *m*-функцію. Така функція може повертати одне, два чи три значення:

- власне саме значення цільової функції $f(x)$.
- значення цільової функції $f(x)$ та її градієнта$\nabla f(x)$.
- значення цільової функції $f(x)$ та її градієнта $\nabla f(x)$ та матрицю Гессе $h(x)=\partial^2 f / \partial x_i \partial x_j$.

Градієнт (рис. 9.1) – це похідна за простором, але, на відміну від похідної за одновимірним часом, градієнт є не скаляром, а векторною величиною.

Не всі методи вимагають використання градієнта цільової функції і для жодного з них не є обов'язковим застосування матриці Гессе, проте надання їх значень істотно прискорює процес оптимізації і підвищує точність.

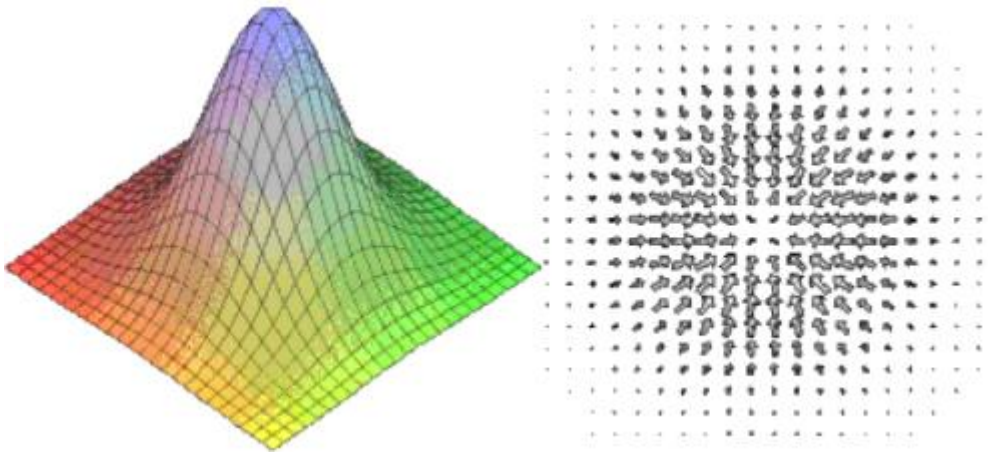


Рис. 9.1. Градієнт.

Як приклад розглянемо цільову функцію:

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2.$$

Тоді градієнт цієї функції має вигляд:

$$\nabla f(x) = \begin{bmatrix} -400(x_2 - x_1^2)x_1 - 2(1 - x_1) \\ 200(x_2 - x_1^2) \end{bmatrix},$$

та матриця Гессе:

$$H(x) = \begin{bmatrix} 1200x_1^2 - 400x_2 + 2 & -400x_1 \\ -400x_1 & 200 \end{bmatrix}.$$

Перейдемо до написання програми, що повертає значення цільової функції, її градієнта та матриці Гессе:

```
function [f g H] = myfun(x)
% Calculate objective f
f = 100*(x(2) - x(1)^2)^2 + (1-x(1))^2;
if nargout > 1 % gradient required
g    =    [-400*(x(2)-x(1)^2)*x(1)-2*(1-x(1));    200*(x(2)-
x(1)^2)];
if nargout > 2 % Hessian required
H = [1200*x(1)^2-400*x(2)+2, -400*x(1); -400*x(1), 200];
end
end
```

Тут значення змінної *nargout* визначає кількість параметрів, що повертаються і залежить від синтаксису виклику функції. Тепер перейдемо безпосередньо до розгляду задачі мінімізації цільової функції. Нижче наведено результат виконання процедури мінімізації.

```
options = optiset('GradObj', 'on', 'Hessian', 'on');
[x fval] = fminunc(@myfun, [10; 10], options)
x =
    1.0000
    1.0000
fval =
  1.2968e-014
  1.2968e-014
```

Далі наведено результати виміру швидкодії обчислень при використанні градієнта та матриці Гессе:

```
options = optiset('GradObj', 'on', 'Hessian', 'on');
tic;
[x fval] = fminunc(@myfun, [10; 10], options);
toc;
Elapsed time is 0.052151 seconds
```

тільки градієнта

```
options = optimset('GradObj','on','Hessian','off');
tic;
[x fval] = fminunc(@myfun, [10; 10], options)
toc;
Elapsed time is 0.068619 seconds
x =
    1.0000
    1.0000
fval = 7.1014e-012
```

і лише значення цільової функції.

```
options = optimset('GradObj','off','Hessian','off');
tic;[x fval] = fminunc(@myfun, [10; 10], options)
```

```
toc;
Elapsed time is 0.026089 seconds
x =
    1.0391
    1.0795
fval = 0.0015
```

Отже, у разі використання матриці Гессе дає виграш по швидкодії приблизно 30% за збереження точності результату. У разі відключення використання градієнта система видає попередження про перехід на метод лінійного пошуку. Обчислення займають менше часу, але значно зменшується точність результату.

Деякі методи оптимізації працюють із векторними або матричними цільовими функціями. Основна різниця між представленням таких цільових функцій у MATLAB полягає у поданні їх похідних. Замість градієнта для векторних функцій записується матриця Якобі. Так якщо x – вектор незалежних змінних, а $F(x)$ – векторна цільова функція, то матриця Якобі $J(x)$ набуває вигляду:

$$J_{ij}(x) = \frac{\partial F_i(x)}{\partial x_j}.$$

Для матричних цільових функцій матриця Якобі виходить перетворенням матриці цільової функції у вектор, шляхом перенесення стовпців в один ряд. Наприклад, матрична функція

$$F(x) = \begin{bmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \\ F_{31} & F_{32} \end{bmatrix},$$

може бути представлена як векторна функція

$$f(x) = \begin{bmatrix} F_{11} \\ F_{21} \\ F_{31} \\ F_{12} \\ F_{22} \\ F_{32} \end{bmatrix}.$$

Існує механізм анонімних функцій, які потрібно синтаксично повно визначати. Такі функції можуть бути корисні, коли цільова функція досить проста, і не передбачається використовувати її градієнти або матрицю Гессе. Наприклад, розглянута вище цільова функція досить проста для використання її в анонімній формі:

```
ann_myfun = @(x)(100*(x(2) - x(1)^2)^2 + (1 - x(1))^2)
```

Нижче наведено приклад використання анонімної функції. Як видно, її використання не впливає на точність обчислень.

```
options = optimset('GradObj','off','Hessian','off');
ann_myfun = @(x)(100*(x(2) - x(1)^2)^2 + (1 - x(1))^2);
tic;
[x fval] = fminunc(ann_myfun, [10; 10], options)
toc;
x =
    1.0391
    1.0795
fval =
    0.0015
Elapsed time is 0.062211 seconds.
```

Часто виникає ситуація, коли цільова функція містить додаткові параметри, крім тих, за якими її мінімізують. У той же час, запропоновані функції мінімізації можуть передати в цільову функцію тільки один параметр, що оптимізується. Існують три способи вирішення цього завдання.

Спосіб 1. Глобальні змінні.

Змінні, які потрібно передати на цільову функцію, оголошуються глобальними. Далі вони прямо викликаються із програми, що реалізує обчислення цільової функції. Цей метод украй не рекомендується до застосування у зв'язку з широко відомими проблемами, пов'язаними з іменами глобальних змінних та розмежуванням області видимості.

Спосіб 2. Анонімні функції.

Наступним способом передбачає використання анонімних функцій. Цільова функція задається з параметрами, а при оптимізації використовується покажчик на анонімну функцію, що викликає цільову та передає параметри. Перейдемо до прикладу. Нехай потрібно мінімізувати функцію

$$f(x, a, b) = (x_1 - a)^2 + (x_2 - b)^2$$

де параметри приймають такі значення $a = 3$ і $b = 5$. Створимо *m*-файл, що містить програму:

```
function f = myfun(x,a,b)
f = (x(1)-a)^2+(x(2)-b)^2;
end
```

Наведемо приклад використання анонімних функцій передачі параметрів.

```
a=3;
b=5;
x0=[1;1];
f=@(x)myfun(x,a,b);
[x fval] = fminunc(f,x0)
x =
     3
     5
fval =
     0
```

Спосіб 3. Вкладені функції.

В рамках реалізації даного підходу розробляється *m*-файл з функцією, що приймає *a*, *b*, *x*0 як вхідні параметри, і містить вкладену цільову функцію та виклик алгоритму оптимізації. Для розглянутого вище прикладу створимо наступний М-файл:

```
function [x,fval] = myfun(a,b,x0)
[x,fval] = fminunc(@NestedFun,x0);
```

```
function f = NestedFun(x)
f = (x(1)-a)^2+(x(2)-b)^2;
end
end
```

Нижче наведено приклад використання вкладених функцій.

```
[x,fval]=myfun(3,5,[1;1])
x =
     3
     5
fval =
     0
```

9.2. Завдання обмежень

Важливу роль у оптимізації грає завдання обмежень на аргумент. Наявність апріорної інформації та її використання для створення набору обмежень, дозволяють суттєво скоротити час обчислень, а в деяких завданнях це є обов'язковою умовою рішення.

Optimization Toolbox підтримує кілька видів обмежень, вибудовуючи їх за ієрархією:

1) Граничні умови – верхня і нижня межі окремих компонентів вектора аргументу $l < x < u$.

2) Лінійні рівності. Задаються як $A_{eq}x = b_{eq}$. Тут матриця $m \times n$ A_{eq} задає m рівностей для n-вимірного вектора аргументу.

3) Лінійні нерівності. Задаються як $Ax \le b$. Тут матриця $m \times n$ A визначає m рівностей для n-мірного вектора аргументу.

4) Нелінійні рівності. Задаються як $c_{eq}(x) = 0$. Тут функція $c_{eq}(x)$, може бути як скаляром так і вектором.

5) Нелінійні нерівності. Задаються як $c(x) \le 0$. Тут функція $c(x)$, може бути як скаляром так і вектором.

Передбачається, що нерівності завжди задані в один бік. За потреби зміни напряму можна використовувати множення на мінус одиницю. Деякі обмеження можуть бути записані у різний спосіб. Так замість $5x \le 20$ краще використовувати $x \le 4$. Розглянемо обмеження докладніше.

Граничні умови – найбільш простий вид обмежень, що задають верхню та нижню межі для компонентів вектора *x*. Як значення кордону можуть використовуватися вирази *Inf* і *–Inf*. Якщо потрібно вказати лише верхню або нижню межу, то другу компоненту обмеження можна опустити. Також, якщо для *n* вектору можна вказати лише перші *m* обмежень, решта буде автоматично заповнена значенням *Inf* з відповідним знаком.

Лінійні рівності та нерівності визначаються парою A_{eq}, b_{eq} або *A, b*. Наприклад, для завдання обмежень

$$\begin{aligned} -4x_1 + 5x_2 + x_3 &\le 11 \\ x_1 + 8x_2 - 2x_3 &\le 1 \\ 3x_1 - x_2 + 7x_3 &\ge 8 \end{aligned}$$

слід вказати

$$A = \begin{bmatrix} -4 & 5 & 1 \\ 1 & 8 & -2 \\ -3 & 1 & -7 \end{bmatrix} \quad B = \begin{bmatrix} 11 \\ 1 \\ -8 \end{bmatrix}.$$

Нелінійні обмеження. Нелінійні нерівності задаються як $c(x) \le 0$, де $c(x)$ – вектор, кожна компонента якого відповідає одному обмеженню. Аналогічно визначаються нелінійні рівності $ceq(x) = 0$. При використанні нелінійних обмежень можна задати так само градієнт функції обмежень, в деяких завданнях це може прискорити швидкодію або підвищити точність. Важливо, що функція нелінійних обмежень має обов'язково ставити і нерівності, і рівності. Якщо одне з них відсутнє, потрібно повертати порожній елемент. Наприклад, розглянемо наступне обмеження – аргументи цільової функції x_1 і x_2 повинні лежати в колі одиничного радіусу і вище за певну параболу:

$$\begin{cases} x_1^2 + x_2^2 \le 1 \\ x_2 \ge x_1^2 - 1 \end{cases}.$$

Для завдання такого обмеження має бути задана така функція:

```
function [c,ceq]=constr(x)
c(1) = x(1)^2+x(2)^2-1;
c(2) = x(1)^2 - x(2) - 1;
ceq = [];
end
```

Так як нелінійні рівності не задані, то функція повертає як другий вихідний параметр порожній елемент.

```
>> x=[2,3]
x =
     2     3
>> constr(x)
ans =
     7     0
```

Наступний приклад, взятий із довідкової системи MATLAB, ілюструє використання всіх можливих обмежень.

```
function myfun
x0 = [1; 4; 5; 2; 5];
lb = [-Inf; -Inf; 0; -Inf; 1];
ub = [ Inf; Inf; 20];
Aeq = [1 -0.3 0 0 0];
beq = 0;
A = [0 0 0 -1 0.1; 0 0 0 1 -0.5; 0 0 -1 0 0.9];
b = [0; 0; 0.2];
[x,fval,exitflag]=fmincon(@myobj,x0,A,b,Aeq,beq,lb,ub,
@myconstr)
end
%-----------------------------------------------
function f = myobj(x)
f = 6*x(2)*x(5) + 7*x(1)*x(3) + 3*x(2)^2;
end
```

```
%-------------------------------------------
function [c, ceq] = myconstr(x)
c = [x(1) - 0.2*x(2)*x(5) - 71 + 0.9*x(3) - x(4)^2 - 67];
ceq = 3*x(2)^2*x(5) + 3*x(1)^2*x(3) - 20.875;
end
```

Нижче наведено результат виконання програми.

```
myfun
x =
   -0.1607
   -0.5357
   20.0000
    2.2444
   22.4444
fval =
  -93.7847
exitflag = 1
```

На цьому ми закінчуємо розгляд завдання цільової функції та обмежень і переходимо до розгляду деяких завдань оптимізації, реалізованих у *Optimization Toolbox*.

9.3. Завдання оптимізації

Як було зазначено раніше, виділяються чотири типи завдань. Однак усередині кожного типу також є різні функції, призначені для вирішення конкретних проблем. Розглянемо ці типи завдань та функції, що в них входять (табл. 9.1 – табл. 9.4).

Таблиця 9.1 – Багатоцільова мінімізація.

Назва	**Формулювання**	**Функція**
Мінімакс	$\min_x \max_i F_i(x),\ A_{eq}x = b_{eq},\ Ax < b,\ l \le x \le u,$ $c(x) \le 0,\ c_{eq}(x) = 0$	*fminmax*
Досягнення мети	$\min_{x,\gamma} \gamma$ так, что $F(x) - weight \cdot \gamma \le goal$ $A_{eq}x = b_{eq},\ Ax < b,\ l \le x \le u,\ c(x) \le 0,$ $c_{eq}(x) = 0$	*fgoalattain*

Таблиця 9.2 – Мінімізація.

Назва	**Формулювання**	**Функція**
Скалярна мінімізація	$\min_x f(x),\ l < x < u$ – скаляр.	*fminbnd*
Мінімізація без обмежень	$\min_x f(x)$	*fminunc* *fminsearch*
Лінійне програмування	$\min_x f^T(x),\ A_{eq}x = b_{eq},\ Ax < b,\ l \le x \le u,$ f – вектор.	*linprog*
Квадратичне програмування	$\min_x \frac{1}{2}x^T Hx + f^T(x),\ A_{eq}x = b_{eq},\ Ax < b,$ $l \le x \le u$	*quadprog*
Мінімізація з обмеженнями	$\min_x f(x),\ A_{eq}x = b_{eq},\ Ax < b,\ l \le x \le u,$ $c(x) \le 0,\ c_{eq}(x) = 0$	*fmincon*
Напів-нескінченна (*semi-infinite*) мінімізація	$\min_x f(x),\ A_{eq}x = b_{eq},\ Ax < b,\ l \le x \le u,$ $c(x) \le 0,\ c_{eq}(x) = 0$ $K(x,w) \le 0$	*fseminf*
Бінарне програмування	$\min_x f^T(x),\ A_{eq}x = b_{eq},\ Ax < b,\ l \le x \le u,$	*bintprog*

Таблиця 9.3 – Рішення виразів.

Назва	**Формулювання**	**Функція**
Нелінійний вираз однієї змінної	$f(x) = 0$	*fzero*
Нелінійний вираз	$F(x) = 0$, n виразів, n змінних	*fsolve*

Таблиця 9.4 – Мінімізація квадратів (правдоподібність траєкторій).

Назва	**Формулювання**	**Функція**
Лінійний випадок найменших квадратів (ЛНК)	$\min_x \|Cx - d\|_2^2$	лівий поділ матриць
Ненегативний ЛНК	$\min_x \|Cx - d\|_2^2,$ $x \geq 0$	*Isqnonneg*
ЛНК с обмеженнями	$\min_x \|Cx - d\|_2^2,$ $A_{eq}x = b_{eq},\ Ax < b,\ l \leq x \leq u$	*Isqlin*
Нелінійний НК	$\min_x \|F(x)\|_2^2 = \min_x \sum_i F_i^2(x)$ при $l \leq x \leq u$	*Isqnonlin*
Нелінійна правдоподібність кривих	$\min_x \|F(x, xdata) - ydata\|_2^2$ $l \leq x \leq u$	*Isqcurvefit*

9.4. Лінійне програмування

Завдання лінійного програмування є одним із найпоширеніших завдань оптимізації. Подібні завдання зустрічаються, наприклад, в економічній кібернетиці, теорії ігор та інше. Розглянемо наступний приклад. Нехай задана цільова функція, яку потрібно максимізувати

$$f(x) = f^T x = 2x_1 + 5x_2 + x_3$$

і при цьому накладено такі обмеження

$$\begin{cases} x_1 - x_2 + x_3 \le 20 \\ 3x_1 + 2x_2 \le 34 \\ x_1 \ge 0 \\ x_2 \ge 0 \\ x_3 \ge 0 \end{cases}.$$

Для вирішення цього завдання будемо використовувати функцію

```
x = linprog(f,A,b,Aeq,beq,lb,ub,x0)
```

Нижче представлений код програми, що вирішує це завдання, та приклад її виконання.

```
function myfun
f=[-2; -5; -1];
A=[1 -1 1;3 2 0];
b=[20; 34];
lb=[0;0;0];
[x,fval]=linprog(f, A, b, [], [], lb)
end
```

Розв'язання задачі лінійного програмування

```
Optimization terminated.
x =
    0.0000
   17.0000
   37.0000
fval =
   -122.0000
```

9.5. Мінімізація без обмежень

Раніше ми розглянули цільову функцію виду

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2.$$

Модифікуємо її, ввівши параметр *a*

$$f(x) = 100(x_2 - x_1^2)^2 + (a - x_1)^2.$$

У цьому випадку мінімум функції досягається у точці

$$x = \begin{bmatrix} a \\ a^2 \end{bmatrix}, \; f(x) = 0.$$

Це завдання відноситься до завдань мінімізації нелінійної функції в умовах відсутності обмежень і для її вирішення в пакеті *Optimization Toolbox* пропонуються дві можливі команди:

```
x = fminsearch(fun,x0,options)
```

використовує метод без взяття похідних, і, використана раніше

```
x = fminunc(fun,x0,options)
```

В даний курс не входить опис теоретичних підстав для цих функцій, для кращого розуміння відмінностей між ними та розмежування областей їх застосування рекомендуємо звернутися до додаткової літератури. Ми лише розглянемо застосування цих функцій на вирішення, зазначеного вище прикладу.

Нижче наведено текст програми, що вирішує це завдання з використанням першої функції та результат вирішення.

```
function [x, fval] = myfun
a = sqrt(11);
x0=[-1;2];
f = @(x)100*(x(2)-x(1)^2)^2+(a-x(1))^2;
tic;
[x,fval] = fminsearch(f, x0)
toc;
end
```

Мінімізація з використанням прямого пошуку

```
x =
    3.3166
   11.0001
fval =
  6.3545e-011
Elapsed time is 0.013697 seconds.
```

Заміна функції мінімізації на `[x,fval] = fminunc(f, x0)` та виконання процедури мінімізації дозволяють отримати результат:

```
a = sqrt(11);
x0=[-1;2];
f = @(x)100*(x(2)-x(1)^2)^2+(a-x(1))^2;
tic;
[x,fval] = fminunc(f, x0)
toc;
x =
    3.3164
   10.9988
fval =
  3.4049e-008
Elapsed time is 0.398895 seconds.
```

Побудуємо графіки процесу мінімізації. У цьому прикладі передачі параметра в цільову функцію використовується механізм анонімних функцій, а збереження даних – вкладена функція.

```
function [x, fval] = myfun
a = sqrt(11);
x0=[-1;2];
f = @(x)100*(x(2)-x(1)^2)^2+(a-x(1))^2;
options = optimset('Display','off','OutputFcn',@outfun);
hist.x = [];
hist.fval = [];
[x,fval] = fminsearch(f, x0,options);
subplot(1,2,1);
plot(hist.fval);
```

```
subplot(1,2,2);
size(hist.x)
plot(hist.x(:,1),hist.x(:,2),'o');
figure;
hist.x = [];
hist.fval = [];
[x,fval] = fminunc(f, x0,options);
subplot(1,2,1);
plot(hist.fval);
subplot(1,2,2);
size(hist.x)
plot(hist.x(:,1),hist.x(:,2),'o');
function stop = outfun(x,optimValues,state)
stop = false;
switch state
case 'init'
hold on
case 'iter'
hist.fval = [hist.fval;
optimValues.fval];
hist.x = [hist.x; x'];
case 'done'
hold off
otherwise
end
end
end
```

На рис. 9.2 та рис. 9.3 наведено дані про те, як протікав процес мінімізації. Зліва наведено графік значень цільової функції, справа – значення аргументу. Видно, що хоча функція прямого пошуку витратила менше часу на мінімізацію, їй знадобилося на це майже втричі більше кроків.

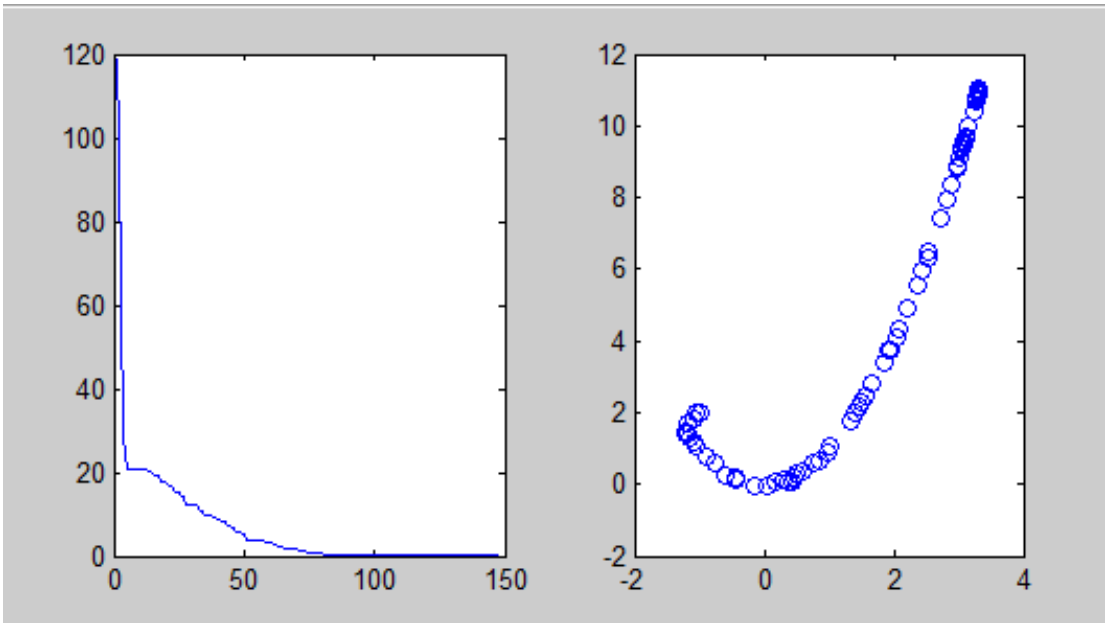


Рис. 9.2. Мінімізація з використанням функції *fminsearch.*

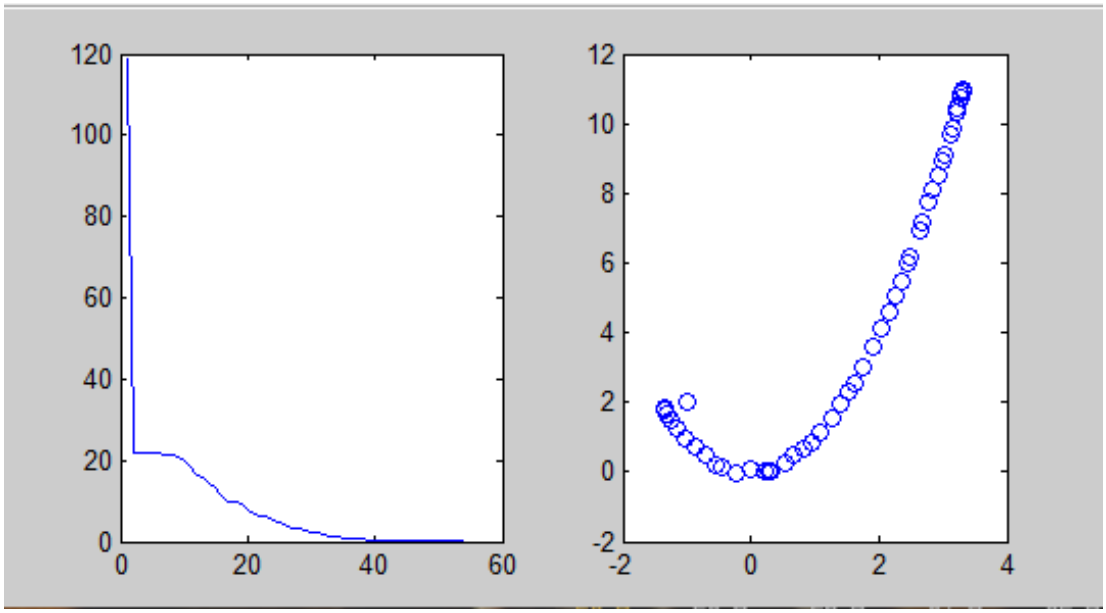


Рис. 9.3. Мінімізація з використанням функції *fminunc.*

9.6. Досягнення мети

Далі розглядається завдання наближення цільової функції до деякого заданого критерію. Формулювання виглядає так $\min \gamma$ так, що

$$\begin{cases} F(x) - weight \cdot \gamma \le goal \\ A_{eq}x = b_{eq} \\ Ax < b \\ l \le x \le u \\ c(x) \le 0 \\ c_{eq}(x) = 0 \end{cases}$$

Фактично йдеться про мінімізацію параметра γ, що відповідає за наближення цільової функції до заданої мети з урахуванням певних ваг. Очевидно, що варіювання вагових коефіцієнтів дозволяє регулювати процес розв'язання задачі. Так, якщо вибрати *weight* = |*goal*|, то забезпечуватиметься рівномірне наближення цільової функції до мети. Як приклад розглянемо лінійну систему, описану в просторі станів наступним чином

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx \end{cases}$$

де

$$A = \begin{bmatrix} -0.5 & 0 & 0 \\ 0 & -2 & 10 \\ 0 & 1 & -2 \end{bmatrix}, \; B = \begin{bmatrix} 1 & 0 \\ -2 & 2 \\ 0 & 1 \end{bmatrix}, \; C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Для даної системи пропонується використовувати керування у вигляді:

$$u = -Ky = -KCx.$$

Тоді замкнута система має вигляд:

$$\dot{x} = (A - BKC)x.$$

Ставиться завдання забезпечити для замкнутої системи цільовий набір власних значень:

$$goal = [-5 \; -3 \; -1]^T,$$

причому, щоб уникнути насичень пропонується обмежити кожен елемент матриці *K* в діапазоні від –4 до 4.

Вирішимо поставлене завдання, використовуючи цю функцію.

Зауваження. Використання такої постановки завдання дозволяє працювати тільки з дійсними числами, оскільки всі оптимізації функції не підтримують комплексних чисел. З іншого боку, можна використовувати певний узагальнюючий параметр оптимізації. Наприклад, смугу пропускання.

Нижче наведено текст відповідної програми, та результат її виконання.

```
function myfun
A = [-0.5 0 0; 0 -2 10; 0 1 -2];
B = [1 0; -2 2; 0 1];
C = [1 0 0; 0 0 1];
K0 = [-1 -1; -1 -1];  % Initialize controller matrix
goal = [-5 -3 -1];    % Set goal values for the eigenvalues
weight = abs(goal)    % Set weight for same percentage
lb = -4*ones(size(K0)); % Set lower bounds on the
controller
ub = 4*ones(size(K0)); % Set upper bounds on the controller
% Set display parameter
options = optimset('Display','off');
[K,fval,attainfactor] = fgoalattain(@(K)eigfun(K,A,B,C),K0,
goal,weight,[],[],[],[],lb,ub,[],options)

function F = eigfun(K,A,B,C)
F = sort(eig(A+B*K*C)); % Evaluate objectives
end
end
```

Результат оптимізації регулятора

```
weight =
     5     3     1
K =
   -4.0000   -0.2564
   -4.0000   -4.0000
fval =
   -6.9313
   -4.1588
   -1.4099
attainfactor =
   -0.3863
```

Значення `attainfactor` показує, що мета була перевиконана на 38%.

Припустимо, що стоїть завдання забезпечити точне досягнення мети

$$F(x) = goal.$$

Тоді можна вказати у структурі опцій оптимізації

```
options = optimset('GoalsExactAchieve',3);
```

Результат виконання, за точним досягненням цілей, представлений нижче.

```
weight =
     5     3     1
K =
   -1.5954    1.2040
   -0.4201   -2.9046
fval =
   -5.0000
   -3.0000
   -1.0000
attainfactor =
 -2.1142e-021
```

9.7. Пошук правдоподібних траєкторій

Завдання пошуку правдоподібних траєкторій є одним із ключових завдань у ідентифікації систем управління. Розглянемо наступний приклад. Нехай рух деякої системи описується рівнянням виду

$$y(x) = x_1 e^{-x_2 t} \sin(x_3 t).$$

Вихід системи вимірюється у деякі нерівномірні моменти часу, причому у вимірах присутній шумовий сигнал. Потрібно, на основі доступних вимірювань, отримати оцінку параметрів вектору. Для вирішення цього завдання будемо використовувати програму, текст якої представлений нижче.

```
function myfun
t=0.01;
while (t(end)<1),
t=[t t(end)+0.01+(rand()-0.5)/300];
end
xdata=t;
ydata=7*exp(-4.*xdata).*sin(18*xdata);
ydata=ydata+1.5*(rand(1,length(xdata))-0.5);
plot(ydata);
x0 = [1;1;1];
[x] = lsqcurvefit(@F,x0,xdata,ydata)
hold on;
plot(F(x,xdata),'r');
grid;
hold off;
function F = F(x,xdata)
F=x(1)*exp(-x(2)*xdata).*sin(x(3)*xdata);
end
end
```

На початку програми задається вектор тимчасових відліків, у які доступні вимірювання. Середній інтервал часу становить 0.01 секунди, отримані значення часу трактуються як вхідні. За отриманими значеннями будується функція, що досліджується, з використанням істинного вектору параметрів

$$x = \begin{bmatrix} 7 \\ -4 \\ 18 \end{bmatrix},$$

а результати обчислень адитивно зашумлюються. Далі викликається функція *lsqcurvefit* і будуються графіки вихідного процесу, отриманого за результатами оптимізації. Результати виконання функції представлені нижче, а графіки процесів на рис. 9.4.

```
>> myfun
x =  -0.0149
     1.6023
     0.0209
```

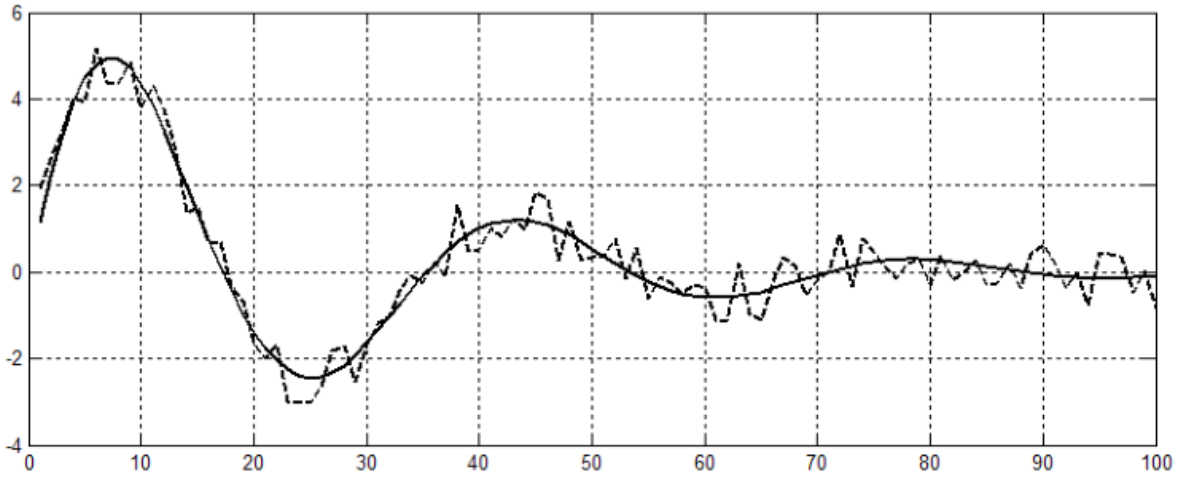


Рис. 9.4. Побудова правдоподібної кривої.

Контрольні запитання:

1. Сформулюйте математичну постановку задачі оптимізації та опишіть засоби для чисельного розв'язання цієї задачі?
2. Які засоби для завдання цільових функцій є у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?
3. Які засоби для завдання обмежень на аргумент є у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?
4. Які засоби для завдання оптимізації є у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?
5. Які засоби є у системі комп'ютерної математики MATLAB для вирішення завдання лінійного програмування, наведіть приклади їх застосування?
6. Які засоби є у системі комп'ютерної математики MATLAB для мінімізації функцій без обмежень, наведіть приклади їх застосування?
7. Які засоби пошуку правдоподібних траєкторій є у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?

РОЗДІЛ 10

РІШЕННЯ ДИФЕРЕНЦІЙНИХ РІВНЯНЬ У MATLAB

Багато прикладних завдань зводяться до вирішення звичайних диференціальних рівнянь (ЗДР) або систем таких рівнянь. Для деяких ЗДР можна побудувати формули «точного» рішення, наприклад, для рівнянь та систем із постійними коефіцієнтами. Елементи символьної математики, вбудовані у MATLAB, дозволяють знаходити аналітичний вид розв'язків таких рівнянь. Але й для них, якщо функції зовнішніх впливів складні (розривні, ламані або неінтегровані функції), побудова «аналітичних» рішень складна. Тому використання наближених методів є вкрай важливим і в MATLAB реалізовано велику кількість чисельних алгоритмів рішення ЗДР.

У цьому розділі ми наводимо основні відомості про функції MATLAB, призначені для вирішення ЗДР та їх систем, як у символьному, так і чисельному вигляді.

10.1. Обчислення похідних та інтегрування в системі MATLAB

10.1.1. Обчислення чисельних значень похідних кінцево-різницевим методом

Будучи, в основному, системою для чисельних розрахунків MATLAB реалізує обчислення похідних кінцево-різницевими методами. Вони реалізуються наступною функцією:

***diff*(*X*)** – повертає кінцеві різниці суміжних елементів масиву *X*. Якщо *X* – вектор, то функція *diff*(*X*) повертає вектор різниць сусідніх елементів [*X*(2) – *X*(1) *X*(3) – *X*(2) *X*(*n*) – *X*(*n*–1)], у якого кількість елементів на одиницю менша, ніж у вихідного вектору *X*. Якщо *X* – матриця, то *diff*(*X*) повертає матрицю

різниць стовпців: [$X(2 : m, :) – X(1 : m\text{-}1, :)$].

***diff* (*X, n*)** – повертає кінцеві різниці порядку *n*. Так, *diff*(*X*, 2) – це те саме, що і *diff*(*diff*(*X*)).

***Y = diff*(*X, n, dim*)** – повертає кінцеві різниці по рядках чи стовпцях для матриці X залежно від значення величини *dim*. Якщо порядок числа *n* дорівнює або перевищує величину *dim*, функція *diff* повертає порожній масив. Приклади:

```
» X=[1 2 4 6 7 9 3 45 6 7]
X =
1 2 4 6 7 9 3 45 6 7
» size(X)
ans = 1 10
» Y = diff(X)
Y =1 2 2 1 2 -6 42 -39 1
» size(Y)
ans = 1 9
» Y = diff(X,2)
Y = 1 0 -1 1 -8 48 -81 40
» X=magic(5)
X =
17  24  1   8   15
23  5   7   14  16
4   6   13  20  22
10  12  19  21  3
11  18  25  2   9
» Y = diff(X,2)
Y =
-25 20  0   0   5
25  5   0   -5  -25
-5  0   0   -20 25
```

10.1.2. Інтегрування засобами системи MATLAB

Для обчислення певних інтегралів система MATLAB має низку засобів. Так, наведені нижче функції виконують чисельне інтегрування методом трапецій та методом трапецій із накопиченням.

***cumtrapz*(*Y*)** – повертає значення інтеграла функції $y(x)$, ординати якої задані як елементи масиву *Y*, припускаючи, що крок інтегрування дорівнює одиниці

(інтегрування методом трапецій з накопиченням). У разі, коли крок відрізняється від одиниці, але є постійним, достатньо обчислити інтеграл та помножити його на величину кроку. Для векторів ця функція повертає вектор, що містить результат інтегрування з накопиченням елементів вектору *Y*. Для матриць – матрицю повертає того ж розміру, як і *Y*, що містить інтегрування з накопиченням для кожного стовпця матриці *Y*.

***cumtrapz*(*X,Y*) –** виконує інтегрування *Y* з накопиченням по змінній *X*, використовуючи метод трапецій. *X* і *Y* повинні бути векторами однієї і тієї ж довжини або *X* повинен бути вектор-стовпцем, а *Y* – матрицею.

***cumtrapz*(*..., dim*) –** виконує інтегрування із накопиченням елементів за розмірністю, точно визначеною скаляром *dim*. Довжина вектора *X* повинна дорівнювати *size*(*Y, dim*). Приклад:

```
» Y=magic(4)
Y =
16   2    3    13
5    11   10   8
9    7    6    12
4    14   15   1
» Z = cumtrapz(Y,1)
Z =
0         0         0         0
10.5000   6.5000    6.5000    10.5000
17.5000   15.5000   14.5000   20.5000
24.0000   26.0000   25.0000   27.0000
```

***trapz*(*Y*) –** повертає інтеграл, використовуючи інтегрування методом трапецій. Якщо *Y* – вектор, *trapz*(*Y*) – повертає масив інтегралів від елементів вектору *Y*, якщо *Y* – матриця, то *trapz*(*Y*) повертає вектор-рядок, що містить інтеграли від елементів кожного стовпця цієї матриці.

***trapz*(*X,Y*)** – повертає інтеграл від функції *Y* змінної *X*, використовуючи метод трапецій.

***trapz*(..., *dim*)** – повертає інтеграл рядками або стовпцями для вхідної матриці, залежно від значення змінної *dim*. Приклад:

```
» X=0:pi/70:pi/2;
» Y=cos(X);
» Z = trapz(Y)
Z = 22.2780
```

Наступна група функцій здійснюють інтегрування (одноразове та дворазове), використовуючи квадратурну формулу Ньютона-Котеса. Функції *quad* і *quad*8 використовують два різні алгоритми для обчислення певного інтеграла. Функція *quad* виконує інтегрування, використовуючи рекурсивне правило Сімпсона. Функція *quad*8 виконує інтегрування за допомогою квадратурних формул Ньютона-Котеса 8-го порядку.

***dblquad*('*fun*' ,*inmin, inmax, outmin, outmax*)** – обчислює та повертає подвійний інтеграл для підінтегральної функції *fun*(*inner, outer*), використовуючи квадратурну функцію *quad, inner* – внутрішня змінна, що змінюється від *inmin* до *inmax*, та *outer* – зовнішня змінна, що змінюється від *outmin* до *outmax*. Перший аргумент '*fun*' – рядок, який описує підінтегральну функцію. Ця функція має бути функцією двох змінних видів *fout = fun*(*inner, outer*). Функція повинна брати вектор *inner* та скаляр *outer* та повертати вектор *tout*, який є функцією, обчисленою в *outer* та у кожному значенні *inner*.

***result = dblquad*('*fun*', *inmin, inmax, outmin, outmax, tol, trace*)** – передає параметри *tol* і *trace* у функцію *quad*.

***result = dblquad*('*fun*', *inmin, inmax, outmin, outmax, tol, trace, order*)** – передає параметри *tol* і *trace* функції *quad* чи *quad*8 залежно від значення рядка *order*. Допустимі значення для параметра *order* є '*quad*' і '*quad*8' або ім'я будь-якого заданого користувачем квадратурного методу з таким же викликом і параметрами, як у функцій *quad* і *quads*. Наприклад:

Нехай М-файл *integ*1.*m* описує функцію 2**y**sin(*x*)+*x*/ 2*cos(*y*), тоді:

```
» result = dblquad('integ1',pi,2*pi,0,2*pi)
result = -78.9574
```

***quad*('*fun*', *a*, *b*) –** повертає певний інтеграл від заданої функції '*fun*' на відрізку [*a, b*].

***quad*('*fun*', *a*, *b*, *tol*)** – повертає певний інтеграл із заданою відносною похибкою *tol*. За замовчанням *tol* = l.*e*-3. Можна також використовувати вектор, що складається із двох елементів *tol* = [*rel_tol abs_tol*], щоб точно визначити комбінацію відносної та абсолютної похибки.

***quad*('*fun*', *a*, *b*, *tol*, *trace*)** – додатково при значенні *trace*, що не дорівнює нулю, будує графік, що показує хід обчислень інтеграла.

***quad*('*fun*', *a*, *b*, *tol*, *trace*, *P*1, *P*2,...)** – повертає певний інтеграл від функції, використовує параметри *P*1, *P*2,..., які безпосередньо передаються в певну функцію: *G* = *fun* (*X*, *P*1, *P*2, ...). Приклади:

```
» q = quad('exp',0,2,1e-4)
q = 6.3891
» q = quad('sin',0,pi,1e-3)
q = 2.0000
```

10.2. Символьне розв'язання диференціальних рівнянь

Функцією MATLAB, що вирішує ЗДР у символьному вигляді, є функція *dsolve*. Найпростіший формат її виклику наступний: *dsolve*('*equation*') або *var* = *dsolve*('*equation*'), де *'equation'* – рядок символів, що задає рівняння, а незалежною змінною, якщо вона не вказана, є *t*. Символ *D* у рядку рівняння позначає диференціювання незалежної змінної, представляючи, наприклад, оператор *d*/*dt*. Цифри, що йдуть за символом *D* (якщо вони вказані), позначають порядок похідної. Наприклад, *D*2 означає оператор d^2/dt^2. Будь-який символ, наступний без пропуску за оператором диференціювання, представляє ім'я

залежної змінної (яка не повинна містити символ *D*), наприклад, *D*3*y* означає d^3y/dt^3.

Арифметичні операції та функції у рядку, що становить рівняння, позначаються звичайним чином (без точок). Якщо початкові умови не задані, то рішення мають постійні інтегрування C_1, C_2 і т.д. Вираз *var*, що повертається, має символьний тип s*ym* (не плутати з рядковим типом *char*).

```
z=dsolve('Dx = -a*x') % вирішуємо рівняння dx/dt = -ax(t)
z=C1*exp(-a*t)    % отримуємо x(t) = C1*e^(-at)
```

Якщо у робочому просторі імен MATLAB є змінні *a* і C_1 (нехай, наприклад, *a* = 3, C_1 = 10), їх підстановку у рішення можна виконати командою

```
subs(z)
ans = 10*exp(-3*t)
```

Розв'яжемо рівняння $\frac{dz}{dt} = 4z(t) + 3\sin(t)$

```
dsolve('Dz = 4*z + 3*sin(t)')
ans = -3/17*cos(t)-12/17*sin(t)+exp(4*t)*C1
```

Отримаємо $C_1e^{4t} - \frac{3}{17}\cos t - \frac{12}{17}\sin t$. Як бачимо, у рішеннях використовується постійна C_1. Якщо порядок рівняння вищий, то у рішенні з'являються й інші незалежні постійні.

```
% вирішуємо рівняння y'' + 4y' + 3y + 2 = 0
dsolve('D2y+4*Dy+3*y+2=0')
ans =
exp(-3*t)*C2+exp(-t)*C1-2/3    % Отримаємо C1e^(-t) + C2e^(-3t) - 2/3
```

Функція *dsolve* шукає рішення у символьному вигляді та намагається повернути рішення у квадратурах. Якщо рівняння містить невизначену функцію,

то у рішенні з'являються невизначені інтеграли. Наприклад:

```
dsolve('Dy=f(t)')
ans = Int(f(t),t)+C1
dsolve('Dy-y=f(t)')
ans = (Int(f(t)*exp(-t),t)+C1)*exp(t)
dsolve('Dy-g(t)*y=f(t)')
ans = (Int(f(t)*exp(-Int(g(t),t)),t)+C1)*exp(Int(g(t),t))
```

Функція *dsolve* вміє працювати з деякими розривними функціями, наприклад:

```
dsolve('D2y-y=heaviside(t)')
ans =
exp(-t)*C2+exp(t)*C1+1/2*heaviside(t)*(-2+exp(-t)+exp(t))
```

Тут *heaviside* – функція Хевісайда, для якої в математичній літературі прийнято позначення $H(x)$ і визначається виразом

$$H(x)=\begin{cases}0, & x<0\\ 1, & x>0\end{cases}$$

Її значення у точці розриву у різних джерелах визначається по-різному. MATLAB у точці розриву функцію Хевісайду не визначає та повертає *NaN*.

У форматі виклику функції *dsolve* можна вказати інше ім'я незалежної змінної (не *t*) наступним чином

```
dsolve('equation','varname')
```

або

```
var = dsolve('equation','varname')
```

Наприклад:

```
dsolve('D2y+4*Dy+2*y+c=0','u')
ans = exp((-2+2^(1/2))*u)*C2+exp(-(2+2^(1/2))*u)*C1-1/2*c
```

Розв'яжемо рівняння Бернуллі $y' = y \cdot \operatorname{ctg} x + \frac{y^2}{\sin x}$. Маємо:

```
y=dsolve('Dy=y*cos(x)/sin(x)+y^2/sin(x)','x')
y = -sin(x)/(x-C1)
```

Зверніть увагу, що змінна *y* має символьний тип, а змінної *x* у робочому просторі немає. Щоб виконати перевірку, створимо символьну змінну *x* і символьний вираз *ss*, що представляє диференціальне рівняння.

```
syms x
ss=y*cos(x)/sin(x)+y^2/sin(x)-diff(y,x)
ss = 0
```

У символьний вираз *ss* значення *y*, яке повернула функція *dsolve*, було підставлено автоматично.

Загальний формат виклику функції *dsolve* наступний

```
r = dsolve('eq1,eq2,...', 'cond1,cond2,...', 'v')
r = dsolve('eq1','eq2',...,'cond1','cond2',..., 'v')
```

Тут *eq*l, *eq*2, ... – рядкові вирази, що становлять прості диференціальні рівняння, використовують для позначення незалежної змінної символ *v*; *cond*1, *cond*2, … - вирази, що задають початкові та/або граничні умови. Початкові/граничні умови задаються рядками виду '*y*(*a*)=*b*' або '*Dy*(*a*)=*b*', де *y* ім'я функції, а *a, b* – постійні. Ім'я незалежної змінної має задаватися останнім аргументом. Якщо аргумент '*v*' не заданий, ім'я незалежної змінної належить *t*. Якщо початкових/граничних умов недостатньо для однозначної рішення рівняння(ь), то у рішенні використовуються довільні постійні, що позначаються C_l, C_2, ... Вказувати явно ім'я шуканої функції не потрібно.

```
dsolve('Dy=1+y^2')
ans = tan(t+Cl)
```

Тут символ *y* використовується як ім'я шуканої функції і *t* – як незалежна змінна.

Як бачимо, рішенням є функція $tg(t + C_1)$. Для завдання початкової умови використовуємо другий аргумент

```
y = dsolve('Dy=1+y^2','y(0)=1')
y = tan(t+l/4*pi)
```

Зверніть увагу, що змінна *y* з'явилася у робочому просторі MATLAB, а незалежна змінна *t* – ні. Додамо символьну змінну *t* у робочий простір, виконавши команду *syms t.*

Тепер можна перевірити, що функція *tan*(*t*+*pi*/4) задовольняє диференціальному рівнянню, надрукувавши символьне вираз, що становить рівняння

```
diff(y,t)-1-y^2
ans = 0
```

Заміна символьної змінної *y* її виразом відбулася автоматично. Для перевірки початкової умови виконаємо команду підстановки

```
subs(y,t,0)
ans = l.0000
```

У наступному прикладі вирішимо крайову задачу для диференціального рівняння другого порядку $y'' = -a^2 y; y(0) = 1, y'(\pi/a) = 0$. Маємо:

```
dsolve('D2y = -a^2*y', 'y(0) = 1', 'Dy(pi/a) = 0')
ans = cos(a*t)
```

Далі наведени інші приклади:

```
dsolve('D2y+4*y=0','y(0)=0')
ans = Cl*sin(2*t)
dsolve('D2y+4*y=0','y(0)=0, Dy(0)=1')
ans =
l/2*sin(2*t)
```

```
dsolve('x*Dy=3*y+x^4*cos(x)','y(2*pi)=0','x')
ans =
x^3*sin(x)
```

Загальне рішення ДР 4-го порядку включає 4 невизначені константи

```
de='D4y+2*D2y-x=1';
dsolve(de,'x')
ans = -1/2*sin(2^(1/2)*x)*C2-1/2*cos(2^(1/2)*x)*C1+
1/12*x^3+1/4*x^2+C3*x+C4
dsolve('D4y=y')
ans = C1*exp(-t)+C2*exp(t)+C3*sin(t)+C4*cos(t)
```

Кожна незалежна початкова умова зменшує на одиницю кількість незалежних констант

```
dsolve('D4y=y','y(0)=0, Dy(0)=0')
ans =
(-C2-C4)*exp(t)+C2*exp(-t)+(2*C2+C4)*sin(t)+ C4* cos(t)
```

У граничних умовах можна задавати лінійну комбінацію значень функції та її похідних.

```
y=dsolve('D2y + y=0','y(0)=1, y(1)+Dy(1)=0');
pretty(y)
```

$$\frac{-\cos(1)+\sin(1)}{\cos(1)+\sin(1)}\sin(t)+\cos(t)$$

Рядки рівнянь та початкових/граничних умов можуть містити невизначені змінні. У цьому випадку вони інтерпретуються як символьні та входять до результуючого символьного рішення

```
dsolve('Dy = a*y', 'y(0) = b')     % вирішуєм задачу Коші
ans = b*exp(a*t)      % отримаєм y = be^at
f=dsolve('m*D2y-k*Dy=0','y(0)=0,y(1)=1','x')
f = -1/(-1+exp(k/m))+1/(-1+exp(k/m))*exp(k/m*x)
pretty(f)
```

$$-\frac{1}{-1+\exp\left(\frac{k}{m}\right)}+\frac{\exp\left(\frac{kx}{m}\right)}{-1+\exp\left(\frac{k}{m}\right)}$$

Функція *dsolve* може повернути результат трьома різними способами. Для одного рівняння та однієї результуючої змінної *dsolve* повертає рішення у вигляді символьного виразу (або вектор символьних виразів, якщо рівняння було нелінійним і для нього знайдено кілька рішень).

Для системи кількох рівнянь та рівної кількість вихідних змінних *dsolve* лексично впорядковує рішення і надає їх вихідним змінним.

Для системи рівнянь та однієї вихідної змінної *dsolve* повертає структуру з полями, що містять вирази рішень.

Наприклад, для наступного нелінійного диференціального рівняння функція *dsolve* поверне вектор із 4-х символьних виразів

```
Z=dsolve('(Dy)^2 + У^2 = 1') % вирішуємо рівняння (y')²+y²=1
z =
1
-1
sin(t-C1)
-sin(t-C1)
```

Якщо задати початкові умови, то рішень буде два

```
y = dsolve('(Dy^2 + y^2 = 1','y(0) = 0')
y =
-sin(t)
sin(t)
```

У наступному прикладі для системи двох рівнянь ми отримаємо рішення у вигляді двох символьних виразів *x* та *y*.

```
[x y]=dsolve('Dx = y', 'Dy = -x')
x = -C1*cos(t)+C2*sin(t)
y = C1*sin(t)+C2*cos(t)
```

Якщо приймаємо рішення в одну змінну, то вона буде структурою з двома полями, імена яких збігаються з іменами функцій, що шукаються.

```
q=dsolve('Du = v', 'Dv = -u')
q =
u: [1x1 sym]
v: [1x1 sym]
q.u
ans = -C1*cos(t)+C2*sin(t)
q.v
ans = C1*sin(t)+C2*cos(t)
```

Поводитись з результатом рішення ДР слід також, як з будь-якими символьними об'єктами. Наприклад, вирішимо крайову задачу для диференціального рівняння другого порядку $y'' = -4y; y(0) = 1, y'(\pi/2) = 0$, та побудуємо графік рішення

```
z=dsolve('D2y = - 4*y', 'y(0) = 1', 'Dy(pi/2) = 0')
z = cos(2*t)
ezplot(z)    % будуємо графік рішення
```

Нагадаємо, що функція *ezplot* будує графік символьного виразу (рис. 10.1), переданого їй як перший (і, можливо, єдиний) аргумент.

```
f=dsolve('4*D2y+0.8*Dy+2*y=20','y(0)=-1, Dy(0)=0','x')
f =
-11/7*exp(-1/10*x)*sin(7/10*x)-11*exp(-1/10*x)*
cos(7/10 *x)+10
ezplot(f,[0, 70]);
axis([0 70 0 20]);
gridon;
```

Для ДР 2-го порядку (рис. 10.2) фазові траєкторії будуються на поверхні (*y, y'*).

```
y=dsolve('D2y + y=0','y(0)=1, Dy(0)=1')
```

```
y = sin(t)+cos(t)
syms t;
z=diff(y,t)
z = cos(t)-sin(t)
% будуємо фазову траєкторію яка має форму кола
ezplot(y,z,[0,2*pi])
```

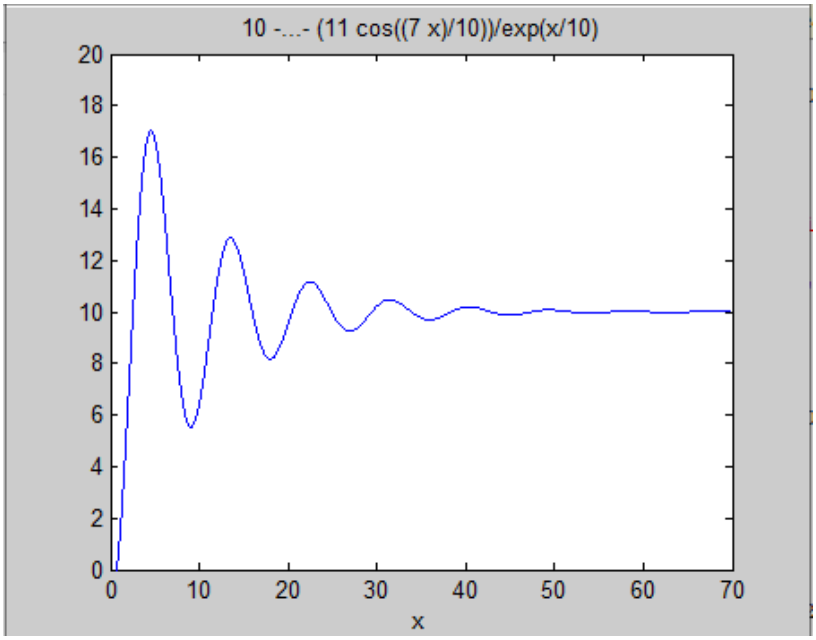


Рис. 10.1. Графік символьного виразу.

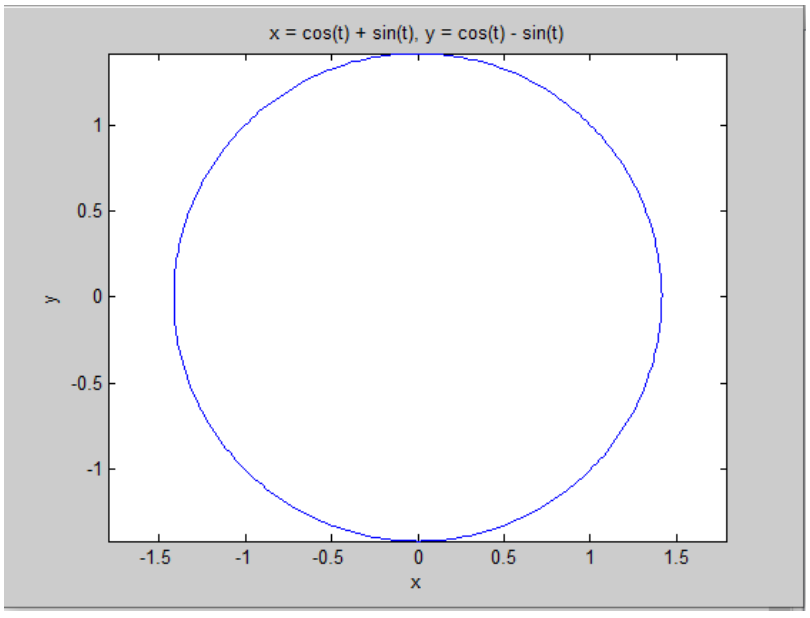


Рис. 10.2. Результат рішення ДР другого порядку.

У наступному прикладі ми будуємо фазову траєкторію системи

$$\frac{dx}{dt} = y - z; \frac{dy}{dt} = z - x; \frac{dz}{dt} = x - 2y,$$

яка проходить через точку (1, 0, 2). Для зручності весь код ми збираємо в єдину функцію:

```
function ex005
% приклад побудови фазової траєкторії системи 3-х рівнянь
sys='Dx = y-z, Dy = z-x, Dz=x-2*y';
cnd='x(0)=1,y(0)=0,z(0)=2';
[x,y,z]=dsolve(sys,cnd);
ezplot3(x,y,z,[-4 6]);
```

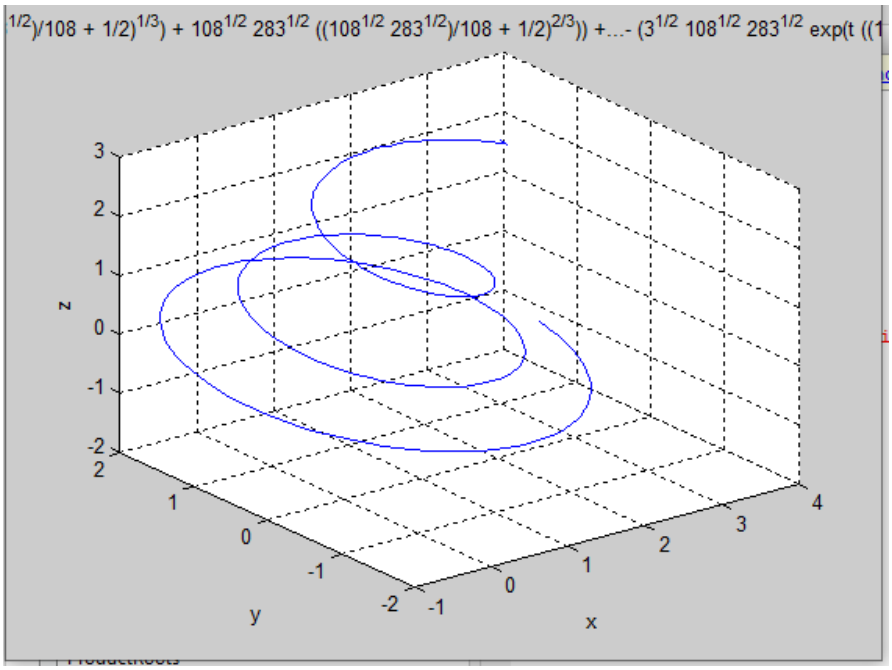


Рис. 10.3. Побудова фазової траєкторії системи з 3-х рівнянь.

У наступному прикладі ми будуємо фазові траєкторії (рис. 10.4) системи диференціальних рівнянь:

$$\frac{dx}{dt} = -x + z; \frac{dy}{dt} = -y - z; \frac{dz}{dt} = y - z,$$

що проходять при $t = 0$ через точки: (0, 1, 0), (0.05, 1, 0), (0.1, 1, 0), (0.2, 1, 0), (0.3, 1, 0).

```
function ex007
sys='Dx = -x+z, Dy = -y-z, Dz=y-z';
cnd={'x(0)=0,y(0)=1,z(0)=0',
     'x(0)=0.05,y(0)=1,z(0)=0',
     'x(0)=0.1,y(0)=1,z(0)=0',
     'x(0)=0.2,y(0)=1,z(0)=0',
     'x(0)=0.3,y(0)=1,z(0)=0'};
for k=1:5
[x,y,z]=dsolve(sys,char(cnd(k)));
X=subs(x,t,0:0.1:pi);
Y=subs(y,t,0:0.1:pi);
Z=subs(z,t,0:0.1:pi);
plot3(X,Y,Z);
hold on;
end
hold off;
grid on;
```

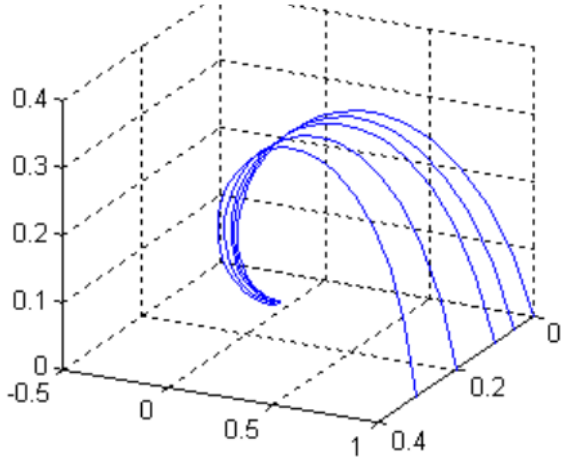


Рис. 10.4. Побудова фазових таєкторій.

Якщо система двох диференціальних рівнянь автономна, то на фазовій площині можна будувати поле напрямів. Для цього використовується функція ***quiver*(*X, Y, U, V*)** для якої *X*, *Y* – матриці координат точок фазової площини, а *U*, *V* – матриці координат векторного поля цих точках.

У наступному прикладі ми створили функцію *ex*010, яка вирішує систему рівнянь $\frac{dx}{dt} = y; \frac{dy}{dt} = x$, будує поле напрямів цієї системи (рис. 10.5) та будує чотири фазові траєкторії, що проходять через точки (0, 0.5), (0.5, 0), (–0.5,0), (0,–0.5).

```
function ex010
[X,Y] = meshgrid(-1:0.1:1);
quiver(X,Y,Y,X); % будуємо поле напрямків (вектор [Y,X] в
точці [X,Y])
hold on;
[x y]=dsolve('Dx = y, Dy = x','x(0)=0,y(0)=0.5');
ezplot(x,y,[-1.35 1.35]);
[x y]=dsolve('Dx = y, Dy = x','x(0)=0.5,y(0)=0');
ezplot(x,y,[-1.35 1.35]);
[x y]=dsolve('Dx = y, Dy = x','x(0)=-0.5,y(0)=0');
ezplot(x,y,[-1.35 1.35]);
[x y]=dsolve('Dx = y, Dy = x','x(0)=0,y(0)=-0.5');
ezplot(x,y,[-1.35 1.35]);
hold off;
```

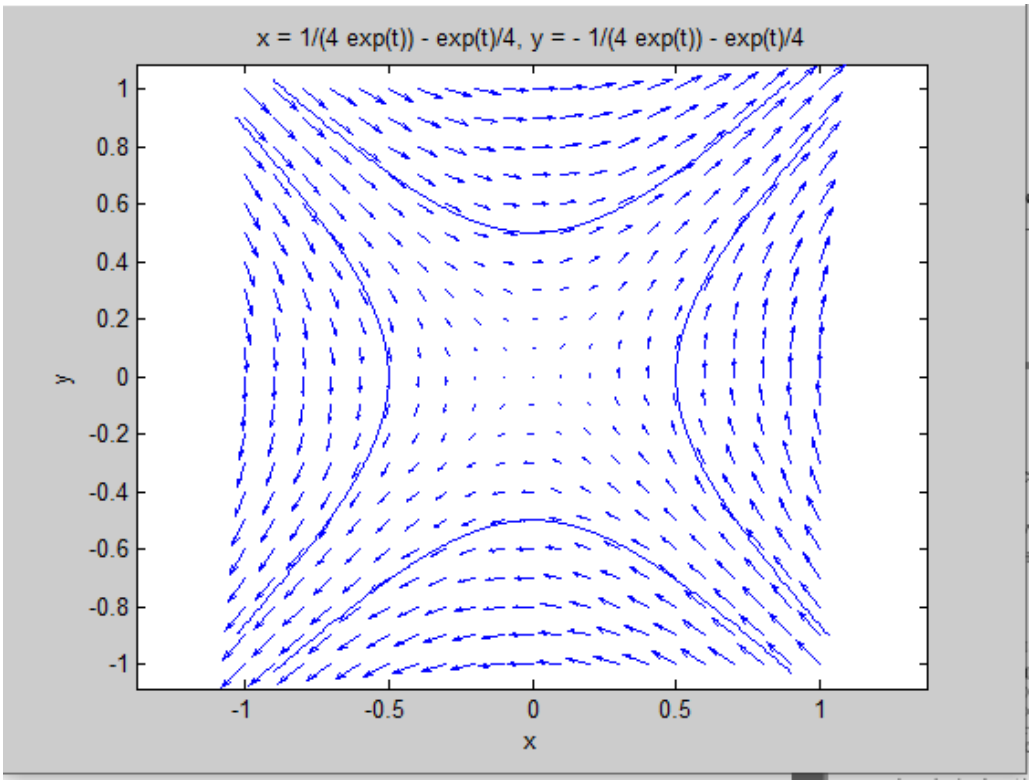


Рис. 10.5. Побудова фазових траєкторій та поля напрямків системи ДК.

Наведемо приклад побудови поля швидкостей та фазових траєкторій (рис. 10.6) для одного ДР 2-го порядку $f'' + f = 0$, яке заміною $y = f$, $z = f'$ може бути представлено системою $y' = z$, $z' = -y$, схожою на систему з попереднього прикладу.

```
function ex004
syms t;
newplot;
hold on;
for a=0: 0.2:2
x=dsolve('D2f+f=0','f(0)=1', ['Df(0)=' num2str(a)]);
y=diff(x,t);
ezplot(x,y,[0,2*pi]);
end;
[X,Y] = meshgrid(-2:0.2:2);
U=Y;
V=-X;
quiver(X,Y,U,V); % будуємо поле напрямків
hold off;
```

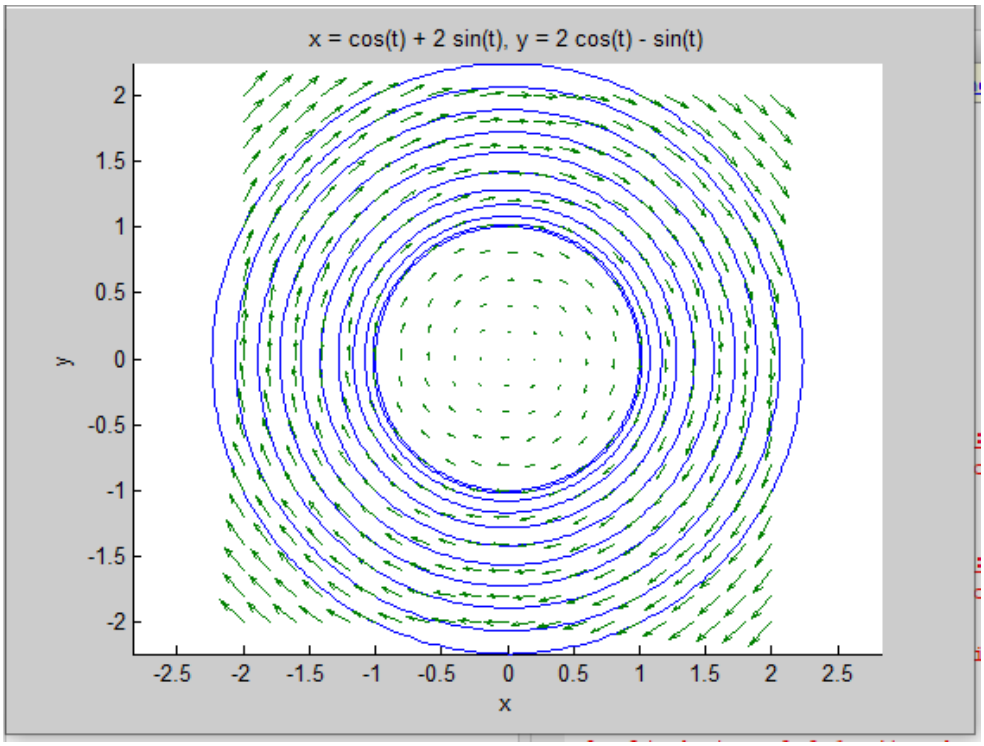


Рис. 10.6. Побудови фазових траєкторій та полів напрямів рівняння 2-го порядку.

Рішення багатьох ДР містить неелементарні (спеціальні) функції. Наприклад наступне однорідне ДР можна вирішити в елементарних функціях.

```
dsolve('Dy-x*y=0','x')
ans = C1*exp(1/2*x^2)
```

Але додавання неоднорідності ускладнює завдання і у вирішенні з'являється спеціальна функція – інтеграл помилок.

```
dsolve('Dy-x*y=1','x')
ans =
(1/2*pi^(1/2)*2^(1/2)*erf(1/2*2^(1/2)*x)+C1)*exp(1/2*x^2)
```

Подивитися графік функції *erf*(*x*) можна наступним способом

```
ezplot('erf(x)')
```

Якщо встановити початкову умову, то у рішенні не буде невизначених постійних і можна побудувати графік рішення

```
z=dsolve('Dy-x*y=1','y(0)=0','x')
ans =
1/2*exp(1/2*x^2)*pi^(1/2)*2^(1/2)*erf(1/2*2^(1/2)*x)
ezplot(z)
```

У наступному прикладі рішення виражається через функції Бесселя

```
dsolve('D2y-exp(x)*y=0','x')
ans =
C1*besseli(0,2*exp(1/2*x))+C2*besselk(0,2*exp(1/2*x))
```

У наступному прикладі рішення надається через функції Ейрі (рис. 10.7)

```
dsolve('D2y-x*y=0','x')
ans = C1*AiryAi(x)+C2*AiryBi(x)
ezplot('airy(x)')
```

Якщо рішення в явному вигляді знайти не вдається, функція *dsolve* намагається знайти рішення в неявному вигляді і при цьому може вивести відповідне повідомлення. У наступному прикладі ми вирішуємо рівняння Абеля і рішення отримуємо у неявній формі (у формі рівняння алгебри).

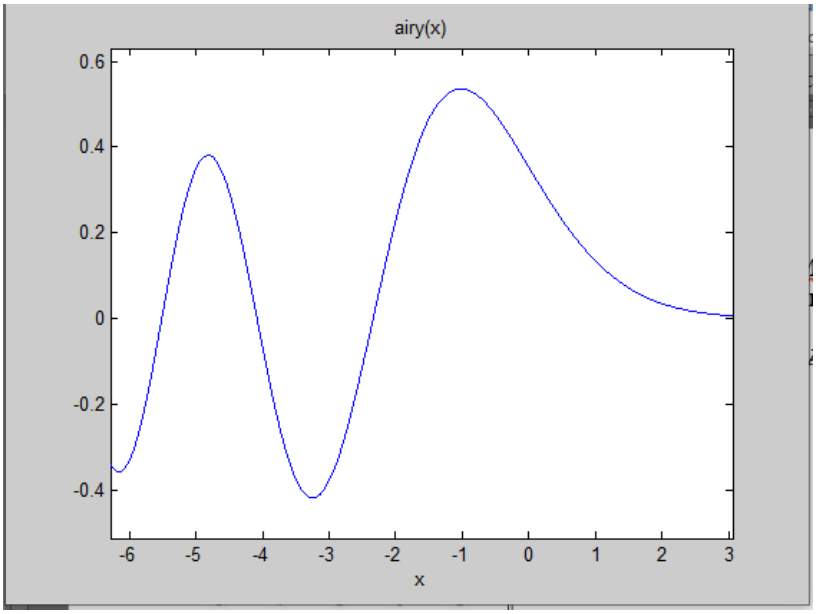


Рис. 10.7. Графік функції Ейрі *Ai*(*x*).

```
q=dsolve('Dy+x*y^3+y^2=0','x')
Warning: Explicit solution could not be found; implicit
solution returned.
In dsolve at 320
q = log(x)-C1+1/2*log(-1+y^2*x^2+x*y)-1/5*5^(1/2)*
*atanh((2/5*x*y+1/5)*5^(1/2))-log(x*y) = 0
```

Використовувати початкові умови визначення рішення у разі неможливо

```
dsolve('Dy+x*y^3+y^2=0','y(0)=1','x')
??? Error using ==> dsolve
Error, (in dsolve/IC) The 'implicit' option is not
available when giving Initial Conditions.
```

Однак можна виконати підстановку

```
syms C1;
subs(q,C1,1)
```

Якщо функція *dsolve* не може знайти рішення, вона виводить повідомлення і повертає порожній символьний об'єкт.

```
dsolve('D2y=sin(y)*sin(t)')
Warning: Explicit solution could not be found.
```

```
ans =
[ empty sym ]
de='D2y+y+y^3-cos(x)=0';
dsolve(de,'x')
Warning: Explicit solution could not be found.
In dsolve at 338
ans =
[ empty sym ]
```

10.3. Вирішення систем нелінійних рівнянь у системі MATLAB

Для чисельного розв'язання нелінійних рівнянь у MATLAB використовується функція *fzero*, що записується у різних видах:

***fzero*('*fun*')** – повертає уточнене значення *x*, у якому досягається нуль функції *fun*, представленої рядком, при початковому значенні аргумента. Значення, що повертається близьке до точки, де функція змінює знак, або *NaN*, якщо така точка не знайдена.

***fzero*('*fun*', *x*)** – повертає значення *x*, при якому *fun*(*x*)=0, із завданням інтервалу пошуку за допомогою вектору *x*=[*x*1 *x*2], такого що *f*(*x*(1)) відрізняється від знака *f*(*x*(2)). Якщо це не так, видається повідомлення про помилку. Виклик функції *fzero* з інтервалом гарантує, що *fzero* поверне значення, близьке до точки, де *fun* змінює знак.

***fzero*('*fun*', *x*, *tol*)** – повертає результат із відносною похибкою *tol*.

***fzero*('*fun*', *x*, *tol*, *trace*)** – видає на екран інформацію з кожної ітерації.

***fzero*('*fun*', *x*, *tol*, *trace*, *P*1, *P*2, ...)** – передбачає додаткові аргументи, що містяться у функцію *fun*(*x*, *P*1, *P*2, ...). При заданні порожньої матриці для *tol* або *trace* використовуються стандартні значення, наприклад *fzero*('*fun*', *x*, [], [], *P*1)

Для функції *fzero* нуль сприймається як точка, де графік функції *fun* перетинає вісь *x*, а не торкається її.

Наступний приклад показує наближене обчислення π/2 як рішення рівняння *cos*(*x*)=0:

```
» x = fzero('cos',[1 3])
x = 1.5708
```

У більш складних випадках рекомендується будувати графік функції *f*(*x*) для наближеного визначення коренів та інтервалів, в межах яких вони знаходяться. Нижче наведено приклад такого роду:

```
function f=fun1(x)
f=0.25*x+sin(x)-1;
```

Побудуємо графік функції (рис. 10.8):

```
» x=0:0.1:10;
plot(x,fun1(x));
grid on;
```

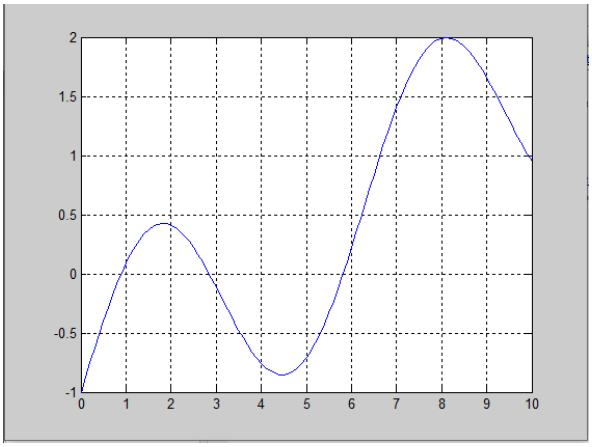


Рис. 10.8. Графік функції *funl*(*x*).

Значення коренів укладено, приблизно, в інтервалах [0.5 1], [2 3] та [5 6]. Знайдемо їх, використовуючи функцію *fzero*:

```
» x1=fzero('fun1',[0.5 1]).
x1 = 0.8905
» x2=fzero('fun1',[2 3])
x2 = 2.8500
» x3=fzero('fun1',[5,6])
X3 = 5.8128
» x3=fzero('fun1',5,0.001)
X3 = 5.8111
```

Зверніть увагу, що корінь *x*3 знайдено двома способами і що його значення в третьому знаку після точки відрізняються в межах заданої похибки *tol*=0.001. На жаль, знайти усі корені відразу функція *fzero* не в змозі.

10.4. Числове рішення задачі Коші в пакеті MATLAB

Даний матеріал присвячений опису можливостей, що надаються MATLAB, з чисельного розв'язання задачі Коші для звичайних диференціальних рівнянь чи систем таких рівнянь.

Задача Коші – одне з основних завдань теорії диференціальних рівнянь (звичайних та з частковими похідними); полягає у знаходженні рішення (інтеграла) диференціального рівняння, що задовольняє так званим початковим умовам (початковим даним).

Від крайових завдань задача Коші відрізняється тим, що область, в якій має бути визначене рішення, тут заздалегідь не вказується. Проте задачу Коші можна розглядати як одну з крайових задач.

Для цього призначені спеціальні функції, їх називають солвери (*solver*). У MATLAB їх кілька – для кожного типу ЗДР свій солвер. Вивчення методики їх застосування ми розіб'ємо на кілька кроків. Спочатку розглянемо застосування солверів на простих прикладах, потім розглянемо складніші завдання, що демонструють максимум можливостей солверів.

Задача Коші для одного звичайних диференціальних рівнянь (ЗДР) *n*-го порядку полягає у знаходженні функції, що задовольняє диференціальне рівняння виду $y^{(n)} = f\left(t, y, y', \dots, y^{(n-1)}\right)$ та початковим умовам $y(t_0) = = y_0; y'(t_0) = y_1, \dots, y^{(n-1)}(t_0) - y_{n-1}$. Перед рішенням рівняння має бути записане у вигляді системи ЗДР першого порядку

$$y'(t) = F(t, y), y(0) = y_0, \tag{10.1}$$

де y – вектор-функція, а y_0 – її початкове значення.

Якщо задана система ДУ деякі (або всі) рівняння якої мають порядок другий і вище, перед рішенням в MATLAB її також слід перетворити до виду (10.1).

Для вирішення задачі Коші (10.1) у MATLAB існує сім функцій (солверів): *ode*23, *ode*45, *ode*ll3, *ode*l5*s*, *ode*23*s*, *ode*23*t* та *ode*23*tb*.

***ode*45** – однокрокові явні методи Рунге-Кутта 4-го та 5-го порядку. Це класичний метод, який рекомендується для початкової спроби рішення. У багатьох випадках він дає добрі результати.

***ode*23** – однокрокові явні методи Рунге-Кутта 2-го та 3-го порядку. При помірних вимогах до жорсткості системи ЗДР та точності рішення цей метод може дати виграш у швидкості рішення.

***ode*113** – багатокроковий адаптивний метод Адамса-Башворта-Мултона змінного порядку, який може забезпечити високу точність рішення.

ode*15*s – багатокроковий адаптивний метод змінного порядку (від 1 до 5 за замовчуванням), який використовує формули чисельного диференціювання. Його варто використовувати, якщо солвер *ode*45 не забезпечує рішення.

ode*23*s – однокроковий метод, який використовує модифіковану формулу Розенброку 2-го порядку. Може забезпечити високу швидкість обчислень за низької точності.

ode*23*t – метод трапецій з інтерполяцією. Цей метод дає хороші результати під час вирішення завдань, що описують осцилятори з майже гармонійним вихідним сигналом.

ode*23*tb – неявний метод, який на початку рішення використовує метод Рунге-Кутта, а в подальшому використовує метод зворотного диференціювання 2-го порядку. При низькій точності цей метод може виявитися ефективнішим, ніж *ode*15*s*.

Методика їх використання однакова, включаючи способи завдання вхідних та вихідних аргументів. У загальному випадку виклик солвера для вирішення задачі Коші здійснюється так

```
[T, Y] = solver(odefun, [t0, tend], y0, options)
```

Тут *solver* – ім'я однієї з вищезгаданих функцій, *odefun* – дескриптор функції (або рядок з її ім'ям), що реалізує обчислення вектор – функції $F(t, y)$ – правої частини системи рівнянь (10.1). Функція $F(t, y)$ повинна бути створена заздалегідь і мати не менше двох вхідних аргументів. Її призначення – обчислювати праву частину системи для вектору y за значенням незалежної змінної t і повертати результати у вигляді вектор-стовпця. Вектор із двох чисел $[t_0, t_{end}]$ представляє діапазон зміни незалежної змінної (початкове та кінцеве значення). y_0 – вектор початкових значень шуканої вектор–функції. Необов'язковий аргумент структури *options* використовується для керування параметрами обчислювального процесу. У ній користувач може задати абсолютну та/або відносну похибку, якщо значення за замовчуванням (106 та 103) його не влаштовують.

Солвер повертає масив T з координатами вузлів (значень незалежної змінної), у яких знайдено рішення, і матрицю рішень Y, кожен стовпець якої є значенням компоненти вектор-функції рішення у вузлах. Значення функцій розташовані по стовпцях матриці, в першому стовпці - значення першої функції, що шукається, у другому – другий і т. д. Кожен рядок матриці Y представляє вектор рішення, який відповідає відповідному значенню незалежної змінної з вектору T. Якщо вихідні параметри не задані, то MATLAB зображує графіки знайдених рішень.

Покажемо застосування солвера ЗДР з прикладу, що став класичним, рішення рівняння Ван-дер-Поля, записаного як системи із двох диференціальних рівнянь:

$$y_1' = y_2;$$
$$y_2' = 2(1 - y_1^2)y_2 - y_1.$$
$$y_1(0) = 0;$$
$$y_2(0) = 1.$$

М-файл *vdp*100.*m*, містить систему ЗДР, та має вигляд:

```
function dy = vdp100(t,y)
dy = zeros(2,1); % вектор стовпець
dy(1) = y(2);
dy(2) = 2*(1 - y(1)^2)*y(2) - y(1);
```

Тоді рішення солвером *ode*15*s* і його графік (рис. 10.9) можна отримати, використовуючи наступні команди:

```
[T,Y] = ode15s('vdp100',[0 30],[2 0]);
plot(T,Y)
hold on;
gtext('y1')
gtext('y2')
```

Останні команди дозволяють за допомогою курсору мишки нанести на графіки рішень $y_1=y(1)$ і $y_2=y(2)$ написи, що позначають їх.

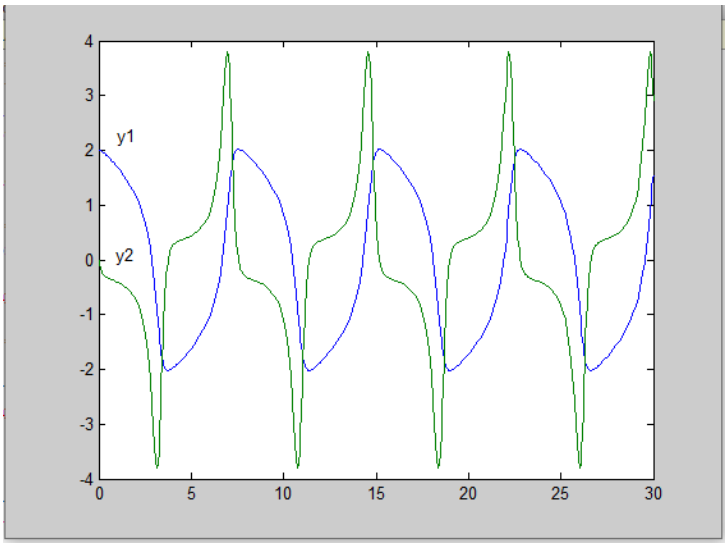


Рис. 10.9. Розв'язання системи диференціальних рівнянь чисельним методом.

Приклад 10.1. Розглянемо задачу Коші $y' = -5t^3 \cdot y,\ y(-1.5) = 0.001.$ Створюємо функцію:

```
function dydt = odefn1(t,y)
dydt=-5*t.^3.*y;
```

Викликаємо солвер та будуємо графік (рис. 10.10):

```
[T,Y]=ode23(@odefn1, [-1.5, 1.5], 0.01);
plot(T,Y,'-ok','MarkerSize',2);
grid on
```

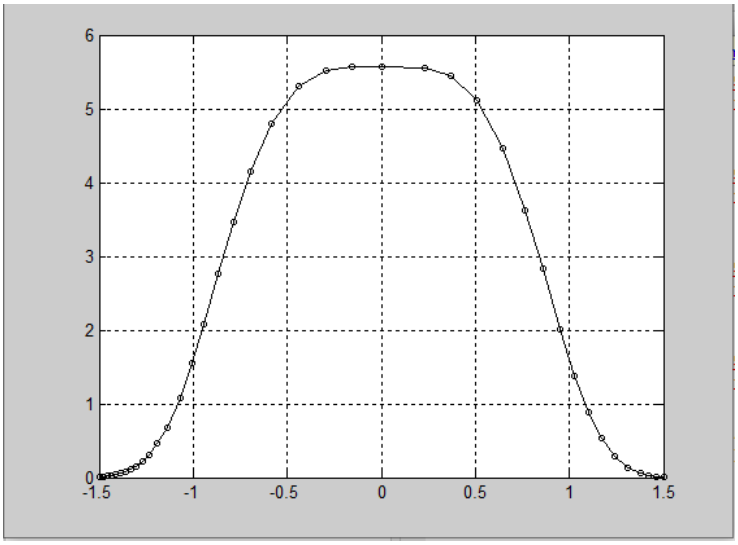


Рис. 10.10. Результат розв'язання задачі Коші з прикладу 10.1.

Пример 10.2. Вирішимо задачу Коші:

$$z'' + \frac{1}{5}z' + z = 0;\ z(0) = 0, z'(0) = 1.$$

Спочатку приведемо її до задачі Коші для системи ЗДР 1-го порядку. Позначимо $y_1 = z, y_2 = z'$. Тоді маємо таку задачу

$$\frac{dy_1}{dt} = y_2; \frac{dy_2}{dt} = -\frac{1}{5}y_2 - y_1; y_1(0) = 0, y_2(0) = 1.$$

Введемо вектор-функцію $Y = [y_1, y_2]^T$ та запишемо систему у векторному вигляді $Y' = F(Y), Y(0) = [0,1]^T$, де векторна функція у правій частині системи має вигляд $F(Y) = [y_2, -0{,}2 \cdot y_2 - y_1]^T$. У розгорнутому вигляді це виглядає так:

$$\begin{pmatrix} y_1' \\ y_2' \end{pmatrix} = F\begin{pmatrix} y_1 \\ y_2 \end{pmatrix}, \text{де } F\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} y_2 \\ -0{,}2 \cdot y_2 - y_1 \end{pmatrix} \text{ та } \begin{pmatrix} y_1(0) \\ y_2(0) \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Створимо функцію $F(Y)$ у MATLAB. Вона повинна містити перший аргумент – ім'я незалежної змінної, навіть якщо він не використовується для формування функції.

```
function F=odefn2(x,y)
F=[y(2);-0.2*y(2)-y(1)];
```

Для вирішення отриманої системи ЗДР викликаємо команду

```
[T,Y]=ode45('odefn2', [0, 20], [0,1]);
```

Тут другий аргумент [0, 20] функції *ode*45 визначає діапазон зміни незалежного аргументу, а третій [0, 1] – початкові значення вектор-функції. Функція *ode*45 повертає рішення у наступному вигляді

```
[T Y]
ans = 0       0   1.0000
0.0001   0.0001   1.0000
0.0001   0.0001   1.0000
…
19.7780  0.1025   0.0829
19.8890  0.1110   0.0694
20.0000  0.1179   0.0553
```

Перший стовпець представляє вибрані значення незалежного аргументу *T*, а другий і третій значення функцій y_1 і y_2. Будуємо графік рішення (рис. 10.11) (значення шуканої функції перебувають у першому стовпці матриці *Y*)

```
plot(T,Y(:,1),'-ok','MarkerSize',2);
gridon
```

Зазначимо, що солвер *ode*23 і *ode*45, використані нами в прикладах 1 і 2, найчастіше використовуються. *ode*23 реалізує алгоритми Рунге-Кутта 2-го та 3-го порядків, що працюють синхронно. Функція автоматично коригує довжину кроку, якщо обчислення обома методами сильно різняться на якомусь кроці. *ode*23 має сенс застосовувати у завданнях, в яких потрібно отримати рішення швидко з невисоким ступенем точності. Солвер *ode*45 заснований на формулах Рунге-Кутта четвертого та п'ятого порядку точності. Він дає найкращі результати і для більшості завдань їм варто скористатися насамперед. При виборі солвера для

розв'язання задачі необхідно враховувати властивості системи диференціальних рівнянь, інакше можна отримати неточний результат або витратити надто багато часу на розв'язання. Усі солвери намагаються знайти рішення з відносною 10^{-3} та абсолютною 10^{-6} точністю.

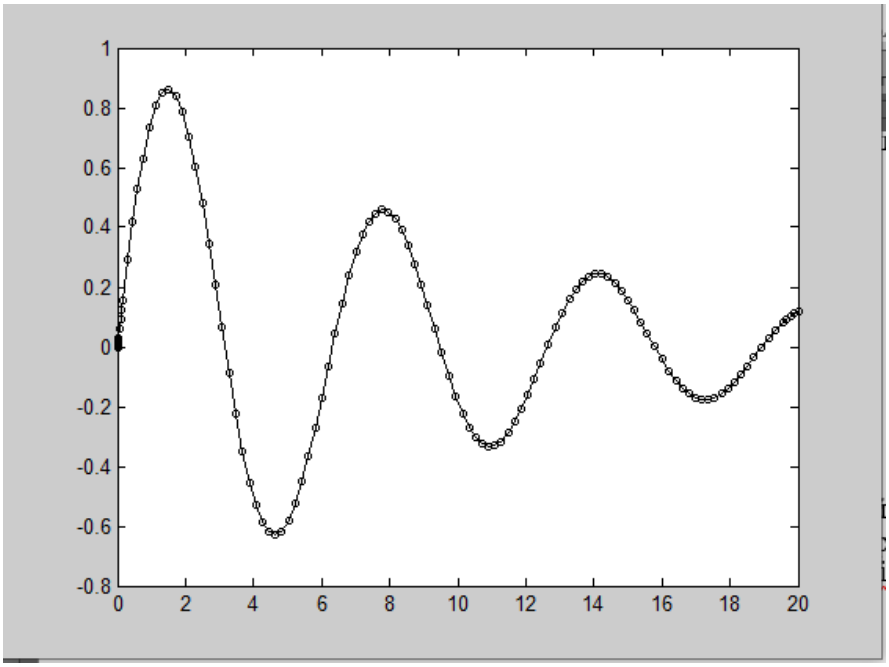


Рис. 10.11. Результат розв'язання задачі Коші з прикладу 10.2.

Приклад 10.3. Розв'яжемо систему рівнянь:

$$\begin{cases} y_1' = y_2 + Kx^2 \\ y_2' = -y_1 \end{cases}, \begin{pmatrix} y_1(0) \\ y_2(0) \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Створюємо функцію правої частини системи

```
function F=odefn3(x,y)
F=[y(2)+0.01*x.^2;-y(1)];
```

Вирішуємо систему ЗДР та будуємо графік

```
[X Y]=ode45('odefn3',[0,20],[0,1]);
plot(X,Y(:,1));
hold on;
plot(X,Y(:,2));
```

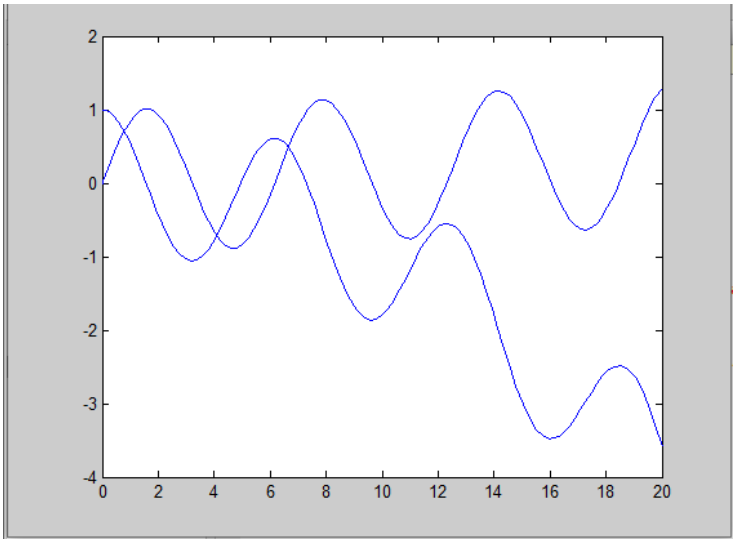


Рис. 10.12. Результат розв'язання задачі з прикладу 10.3.

Якщо ви захочете нанести мітки на графік, щоб розрізнити криві, можна виконати команди

```
hold on; gtext('y1'), gtext('y2')
```

та в інтерактивному режимі вкажіть місце для відображення першої мітки та другої (мітки вже показані на попередньому рисунку).

Солвери можуть знайти наближене рішення для заданих значень незалежної змінної, якщо як другий вхідний аргумент вказати вектор з цими значеннями, впорядкованими за зростанням.

```
[X Y]=ode45('odefn3',[0,5,10,15,20],[0,1])
X =
     0
     5
    10
    15
    20
Y =
         0    1.0000
   -0.8396    0.0486
   -0.3342   -1.8015
    0.9356   -2.9757
    1.2954   -3.5830
```

Інтерфейс солверів припускає звернення до них з одним вихідним аргументом:

```
sol = ode45(@odefn3,[0 ,20],[0,1]);
```

Такий виклик солвера призводить до утворення структури *sol* з інформацією про наближене рішення. Поле *x* структури *sol* містить вектор рядок значень незалежної змінної, а поле *y* – матрицю значень рішення, записаних по рядках. Інакше кажучи, *sol.x* еквівалентно вектору *T*, а *sol.y* – матриці *Y* з попереднього виклику. Тоді, наприклад, для побудови графіка рішення, отриманого в останньому прикладі, ми маємо виконати команду

```
plot(sol.x, sol.y(1,:));
```

Графік показаний на рис. 10.13.

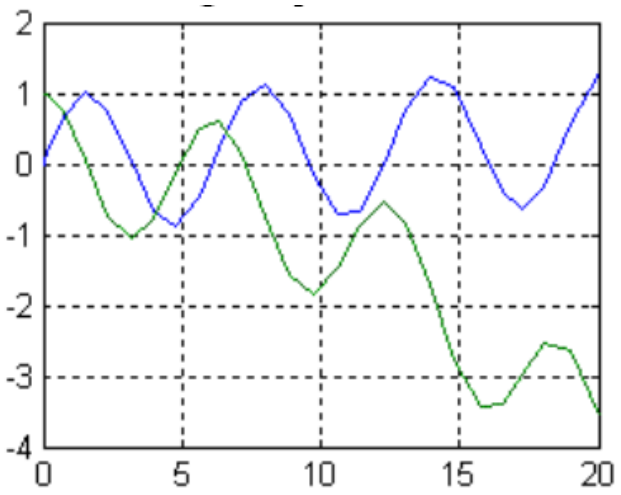


Рис. 10.13. Графік рішення.

Злами на графіку говорять про те, що за такого виклику солвера, вузлів, у яких знайдено рішення, недостатньо. Для визначення значень рішення у проміжних точках слід вдатися до інтерполяції. Це може ефективно виконати функцію *deval.* Її аргументами є структура *sol* і вектор з координатами незалежної змінної, котрим

слід обчислити значення. Знайдені значення компонент вектор функції записуються у вихідному аргументі. Наприклад, у наступному коді *yval*(1,:) зберігаються значення першої шуканої функції в точках вектору *xval*, *yval*(2,:) – другий.

```
xval= 0:0.1:20;
yval=deval(sol,xval);
plot(xval, yval(1,:), xval, yval(2,:));
grid on;
```

Графік показано на рис. 10.14. Як бачимо, графік рішення став гладшим. Зазначимо, що солвери самостійно вибирають вузли на заданому відрізку. Крім того, солвер *ode*45 ще сам виконує інтерполяцію у разі звернення до нього з двома вихідними аргументами.

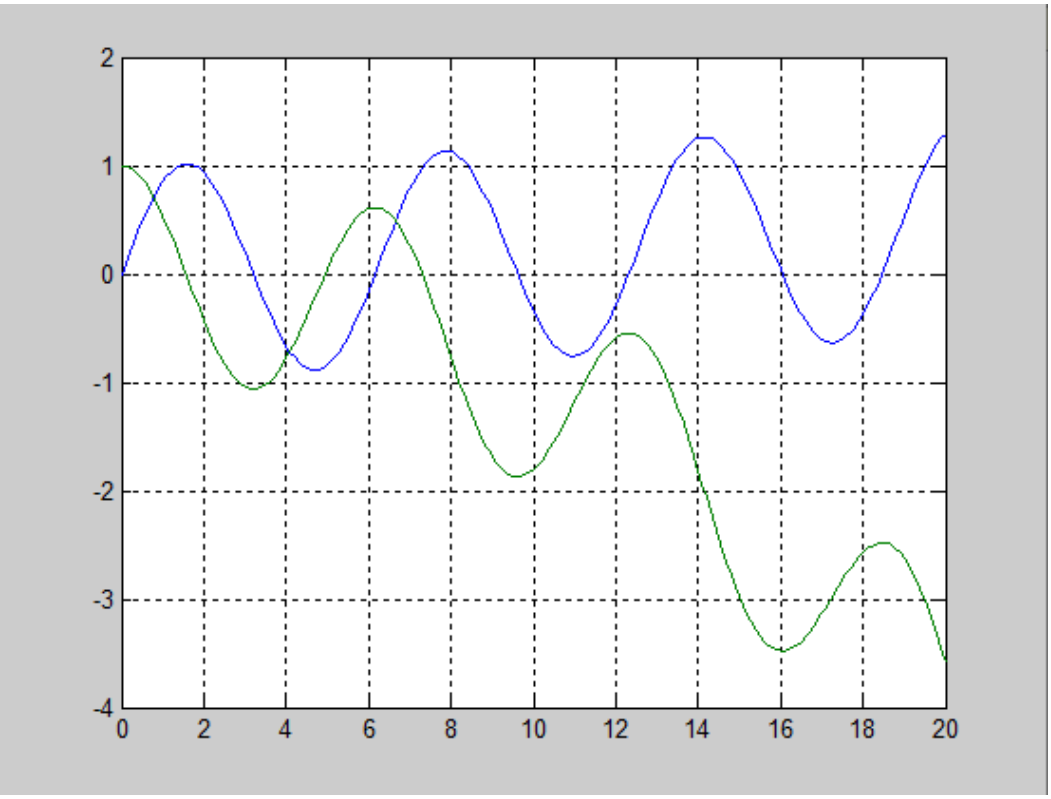


Рис. 10.14. Графік рішення.

Приклад 10.4. Досліджуємо розв'язання задачі Коші:

$$x'' + x^3 = \sin t\,; x(0) = 0, x'(0) = 0.$$

Перетворимо рівняння 2-го порядку на систему рівнянь 1-го порядку і весь код рішення зберемо в одну функцію *odefn*4. Вона не приймає жодного аргументу і не повертає жодного значення, але містить підфункцію *fun*(*x*,*y*), яка обчислює праву частину системи рівнянь.

```
function odefn4                    % рішення ЗДР 2-го порядку
Y0=[0; 0];                         % вектор початкових умов
[T,Y]=ode45(@fun,[0 50],Y0);       % вирішуємо систему
plot(T,Y(:,1),'k.-');              % графік рішення
grid on;
pause;
plot(Y(:,1),Y(:,2));               % фазова траєкторія

function F=fun(x,y)   % підфункція правої частини системи
F=[y(2); -y(1)^3+sin(x)];
```

На лівому рис.10.15 показаний графік розв'язання *x*(*t*), а правому – фазова траєкторія.

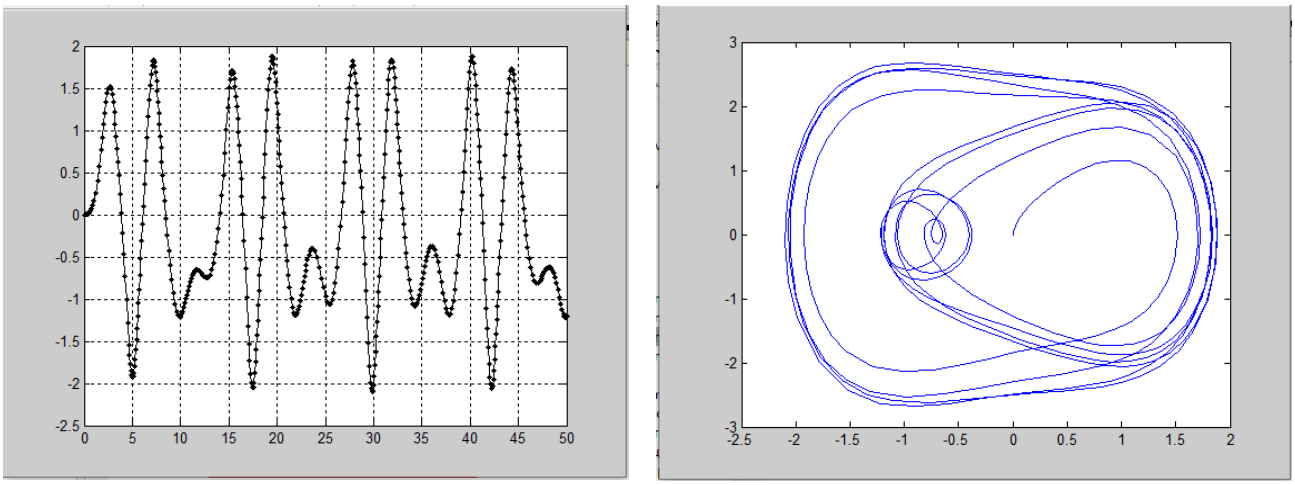


Рис. 10.15. Результат розв'язання задачі з прикладу 10.4.

Приклад 10.5. Досліджуємо розв'язання задачі Коші для системи рівнянь:

$$x' = y(t);\ y' = -0.01y(t) - \sin x(t)\,;x(0) = 0, y(0) = 2.1.$$

Весь код рішення зберемо в одну функцію *odefn5* з підфункцією *fun*(*x*, *y*), яка обчислює праву частину системи рівнянь.

```
function odefn5                    % рішення ЗДР 2-го порядку
Y0=[0; 2.1];                       % вектор початкових умов
[T,Y]=ode45(@fun,[0 100],Y0); % вирішуємо систему
plot(T,Y(:,1));
grid on;
pause;
plot(Y(:,1),Y(:,2));

function F=fun(x,y)   % підфункція правої частини системи
F=[y(2); -0.01*y(2)-sin(y(1))];
```

На рис. 10.16, ліворуч показаний графік розв'язання *x*(*t*), а праворуч – фазова траєкторія.

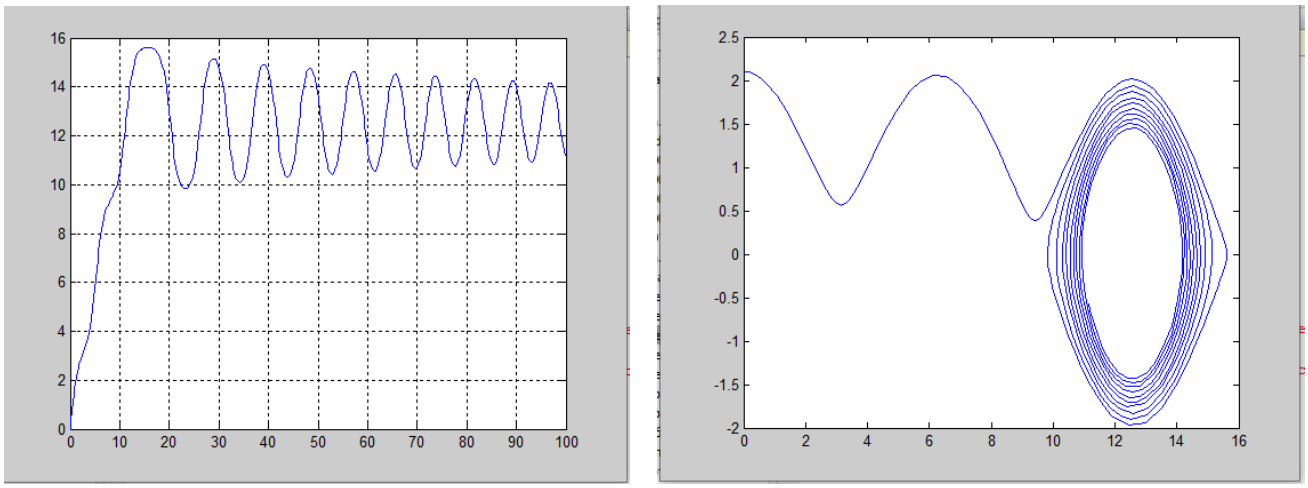


Рис. 10.16. Результат розв'язання задачі з прикладу 10.5.

Приклад 10.6. Досліджуємо розв'язання задачі Коші для системи рівнянь:

$$x' = y(t);\ y' = -x^3(t) + x(t); x(0) = 0, y(0) = 0.1.$$

Весь код рішення зберемо в одну функцію

```
function odefn6                    % розв'язання системи рівнянь
Y0=[0; 0.1];                       % вектор початкових умов
[T,Y]=ode45(@fun,[0 16],Y0);       % вирішуємо систему
plot(T,Y(:,1)); grid on;
pause;
plot(Y(:,1),Y(:,2));               % фазова траєкторія

function F=fun(x,y)  % підфункція правої частини системи
F=[y(2); -y(1)^3+y(1)];
```

На рис. 10.17 показані графіки функцій $x(t)$, $y(t)$.

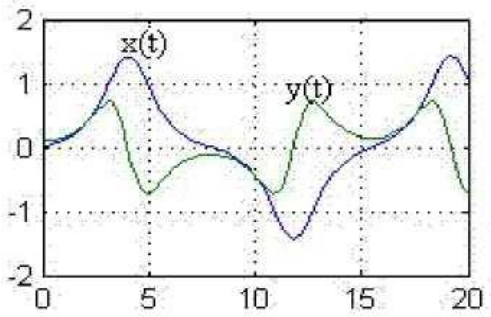


Рис. 10.17. Графіки функцій $x(t)$, $y(t)$.

На рис. 10.18, ліворуч показані графіки функцій $x(t)$, а на праворуч – фазова траєкторія.

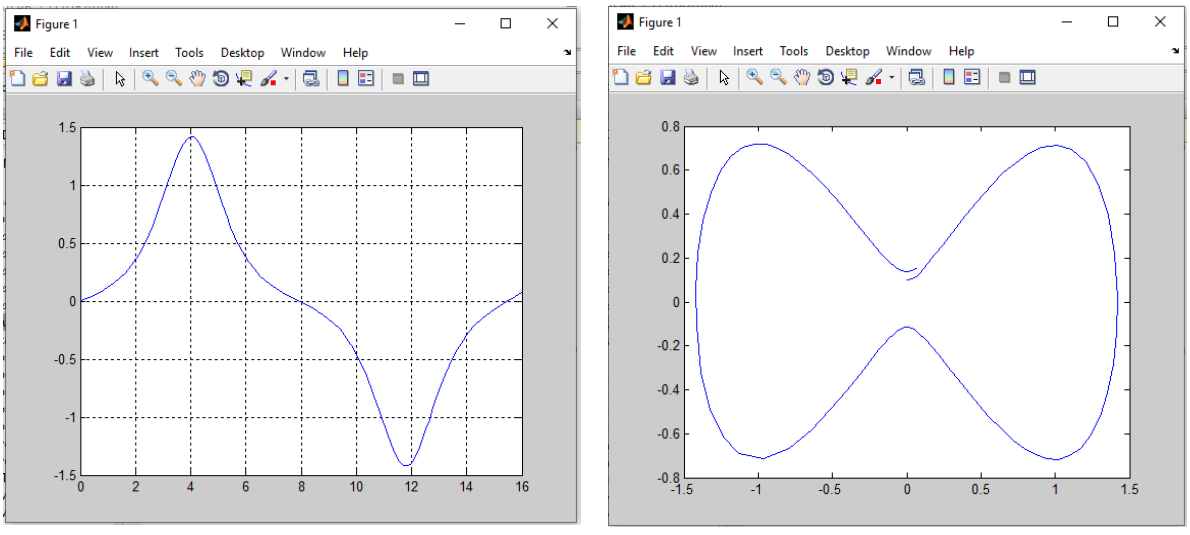


Рис. 10.18. Результат розв'язання задачі з прикладу 10.6.

Приклад 10.7. Розглянемо двовидову модель «хижак – жертва», вперше побудовану Вольтерра для пояснення коливань рибних уловів. Є два біологічні види, чисельністю в момент часу t, відповідно, $x(t)$ та $y(t)$. Особи першого виду є їжею для особин другого виду (хижаків). Чисельності популяцій у початковий час відомі. Потрібно визначити чисельність видів у довільний час.

Математичною моделлю завдання є система диференціальних рівнянь Лотки – Вольтерра:

$$\begin{cases} \dfrac{dx}{dt} = (a - by)x, \\ \dfrac{dy}{dt} = (-c + dx)y, \end{cases}$$

де a, b, c, d – позитивні константи. Проведемо розрахунок чисельності популяцій, якщо $a = 3$, $b = 3$, $c = 1$, $d = 1$, для двох варіантів початкових умов $x(0) = 2$, $y(0) = 1$ та $x(0) = 1$, $y(0) = 2$, котрим побудуємо фазові траєкторії. Весь код рішення зберемо в одну функцію

```
function odeLotVolt
% розв'язання системи рівнянь Лотки - Вольтерра
Y0 = [2; 1];                    % вектор початкових умов
[T,Y]=ode45(@fun,[0 7],Y0);     % вирішуємо систему
plot(Y(:,1),Y(:,2));            % фазова траєкторія
grid on;
hold on;
Y0=[1; 2];                      % вектор початкових умов
[T,Y]=ode45(@fun,[0 7],Y0);     % вирішуємо систему
plot(Y(:,1),Y(:,2));            % фазова траєкторія

function F=fun(x,y) % підфункція правої частини системи
F=[3*y(1).*(1-y(2)); y(2).*(y(1)-1)];
```

На рис. 10.19 видно, що чисельність популяцій змінюється періодично.

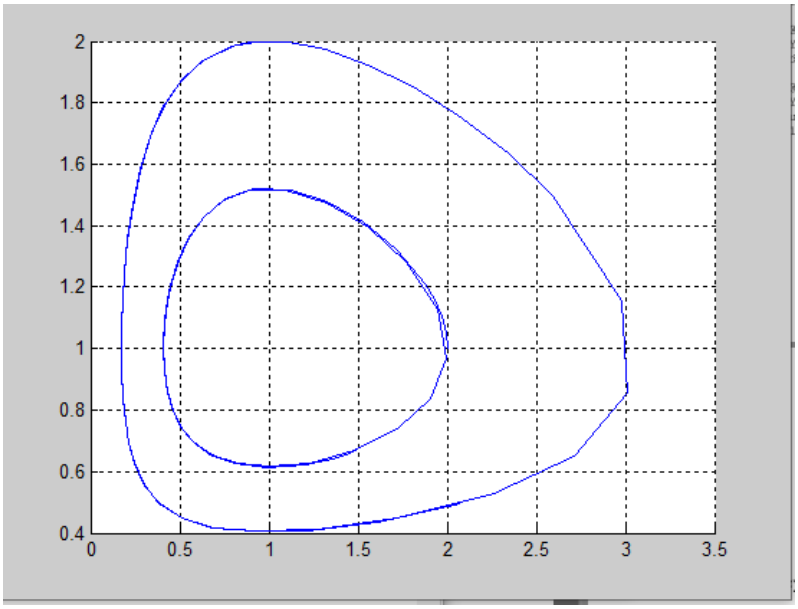


Рис. 10.19. Результат розв'язання задачі з прикладу 10.7.

Приклад 10.8. Розв'яжемо систему диференціальних рівнянь:

$$\frac{dx}{dt} = y; \frac{dy}{dt} = -y - x(z-1) - x^3; \frac{dz}{dt} = xy - z,$$

з початковими умовами $x(0) = 1$, $y(0) = 1$, $z(0) = 0$ і збудуємо її фазовий портрет.

Весь код рішення зберемо в одну функцію odefn8.m

```
function odefn8                    % рішення системи ЗДР
Y0=[1; 1; 0];                      % вектор початкових умов
[T,Y]=ode45(@fun,[0 25],Y0);       % вирішуємо систему
% 3D фазова траєкторія
plot3(Y(:,1),Y(:,2),Y(:,3),'LineWidth',2);
grid on;

F=fun(x,y) % підфункція правої частини системи
F=[y(2);
0.1*y(2)-y(1).*(y(3)-1)-y(1).^3;
y(1).*y(2)-0.1*y(3)];
```

Фазова траєкторія показана на наступному рис. 10.20.

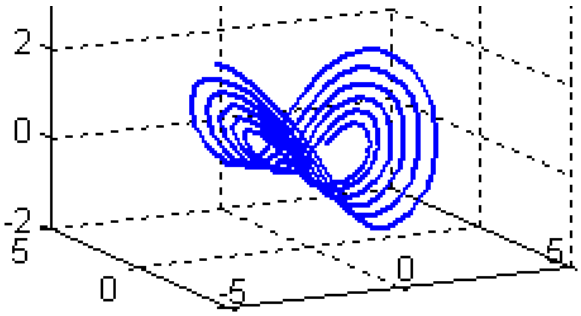


Рис. 10.20. Результат розв'язання задачі з прикладу 10.8.

Приклад 10.9. Досліджуємо поведінку математичного маятника. Нехай маса вантажу дорівнює одиниці, а стрижень, на якому підвішена маса, невагомий. Тоді диференціальне рівняння руху вантажу має вигляд:

$$\varphi'' + k\varphi' + \omega^2 \sin\varphi = 0,$$

де φ(t) – кут відхилення маятника від положення рівноваги (нижнє положення), параметр k характеризує величину тертя, $\omega^2 = \frac{g}{l}$ (g - прискорення вільного падіння, l – довжина маятника). Для визначення конкретного руху до рівняння руху слід додати початкові умови:

$$\varphi(0) = \varphi_0, \varphi'(0) = \varphi_0'.$$

Перетворимо рівняння до системи ЗДР 1-го порядку. Якщо позначити $u \equiv \varphi, v \equiv \varphi'$ то отримаємо

$$\begin{cases} u' = v, \\ v' = -kv - \omega^2 \sin(u), \end{cases}$$

$$u(0) = \varphi_0, v(0) = \varphi_0'.$$

Виберемо такі значення параметрів k=0.5, ω^2 = 10 та початкові значення φ = 0, φ_0' = 5.

Створюємо функцію

```
pend=@(t,y) [y(2); -0.5*y(2)-10*sin(y(1))];
```

Вирішуємо систему та будуємо графік (рис. 10.20, ліворуч):

```
[T,Y]=ode45(pend,[0:0.01:20],[0 5]);
plot(T,Y(:,1)); grid on;
```

Будуємо фазову траєкторію (рис. 10.21, праворуч)

```
plot(Y(:, 1),Y(:,2));
grid on; % розставляється розмітка лінійки
```

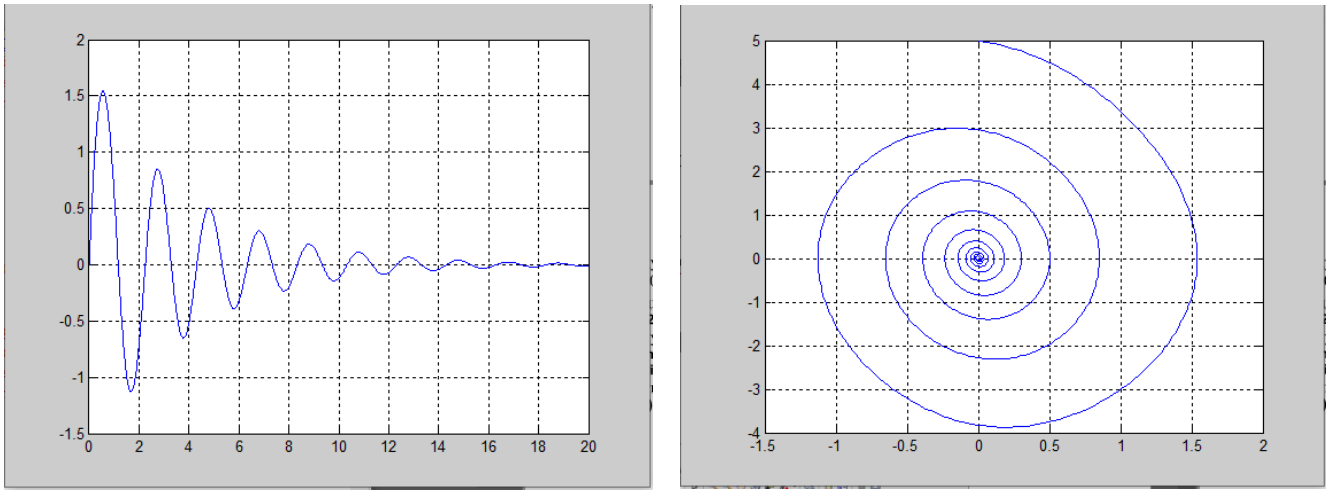


Рис. 10.21. Результат розв'язання задачі з прикладу 10.9.

Як видно з лівого графіка, максимальний кут відхилення маятника не перевищують $n/2$ і коливання маятника згасають.

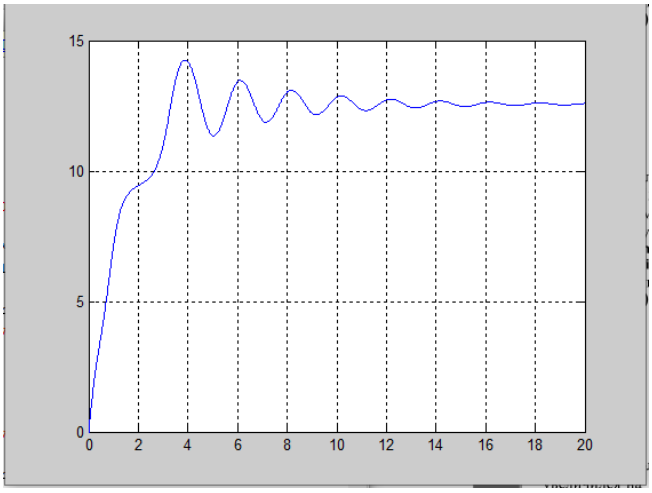
Збільшимо початкову швидкість до 10. Розв'язуємо задачу та будуємо графік розв'язання (рис.10.22, ліворуч)

```
[T,Y]=ode45(pend,[0:0.01:20],[0 10]);
plot(T,Y(:,1));
grid on;
```

Будуємо фазову траєкторію (рис.10.22, праворуч)

```
plot(Y(:, 1),Y(:,2));grid on;
```

Максимальне значення кута становить приблизно 14 радіан. Маятник зробив два повні оберти навколо точки закріплення (кут відхилення збільшився на $4n$), а потім коливання згасають в околиці значення $4n$ радіан (для маятника кут повороту $4n$ представляє те ж, що і 0 радіан, тобто положення рівноваги).



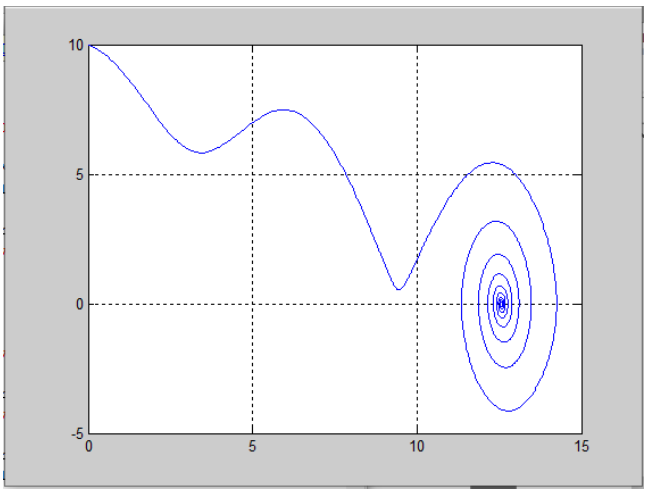


Рис. 10.22. Результат розв'язання задачі з прикладу 10.9 з новими умовами.

Побудуємо кілька графіків кута відхилення (рис. 10.23, ліворуч) і фазових траєкторій (рис. 10.23, праворуч), задаючи різну початкову швидкість.

```
clf;
hold on;      % графіки кута відхилення
for v=5:10
[T,Y]=ode45(pend,[0:0.01:20],[0 v]);
plot(T,Y(:,1)); grid on;
end;
pause;        % чекає натискання будь-якої клавіші
clf;
hold on;      % фазові траєкторіїfor v=5:10
[T,Y]=ode45(pend,[0:0.01:20],[0 v]);
plot(Y(:, 1),Y(:,2));
grid on;
end;
```

Як бачимо, початкова швидкість при v = 5, 6, 7 недостатня, щоб маятник пройшов верхню точку і зробив хоча б один повний оберт. При початковій швидкості v = 8, 9 маятник здійснює один повний оборот, а потім його коливання згасаю. При v = 10 маятник зміг виконати два повні обороти і тільки після цього його коливання стали згасати навколо положення рівноваги.

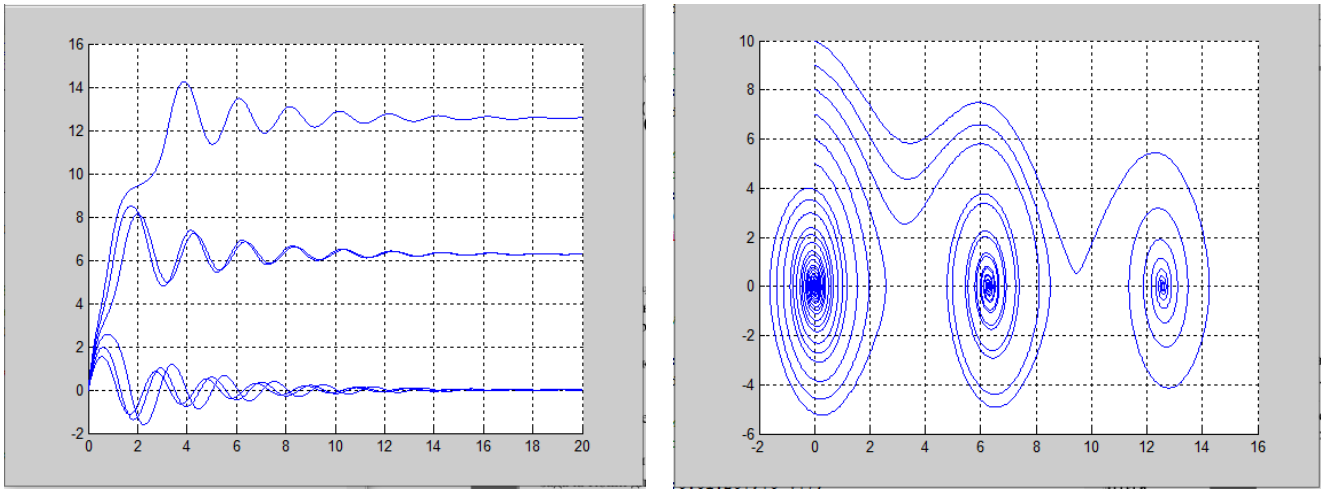


Рис. 10.23. Результат розв'язання задачі з різними кутами відхилення.

Приклад 10.10. Вирішимо ЗДР 3-го порядку:

$$z''' - x^2 z'' + xz' - z = 0$$

з початковими умовами $z(0) = 0; z'(0) = 2; z''(0) = -5$. Відповідна задача Коші для системи ЗДР 1-го порядку має вигляд:

$$\begin{cases} u' = v, \\ v' = \omega, \\ \omega' = x^2\omega - xv + u, \end{cases}$$

$$u(0) = 0, v(0) = 2, \omega(0) = -5.$$

Весь процес вирішення та візуалізації реалізуємо у наступній М-функції без аргументів:

```
function odeo3
% рішення ЗДР 3-го порядку
Y0 = [0; 2; -5];                % вектор початкових умов
[T,Y]=ode45(@fun,[0 1],Y0);     % вирішуємо систему
plot(T,Y(:,1),'k.-');           % будуємо графік рішення
grid on;
xlabel('\itt');
ylabel('\ity');
legend('координата',4);

function F=fun(x,y) % підфункція правої частини системи
F=[y(2); y(3); x.^2.*y(3)-x.*y(2)+y(1)];
```

Викликаючи цю функцію, отримуємо результат який наведено на рис.10.24.

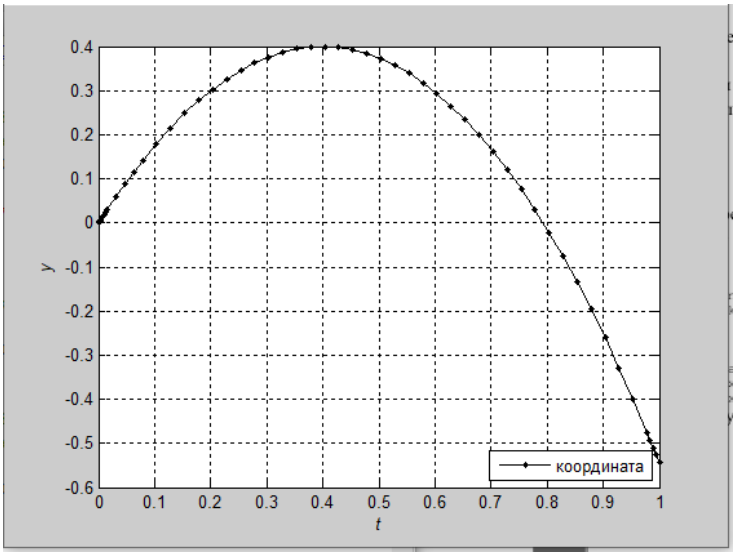


Рис. 10.24. Результат розв'язання задачі з прикладу 10.10.

Приклад 10.11. У довідковій системі MATLAB наводиться приклад розв'язання рівняння Ван-дер-Поля $z'' + z - K(1 - z^2)z' = 0$, рішення якого звичайними солверами дає незадовільний результат. Запишемо рівняння у вигляді системи:

$$\begin{cases} y_1' = y_2 \\ y_2' = -y_1 + K \cdot (1 - y_1^2) \cdot y_2 \end{cases}, \begin{pmatrix} y_1(0) \\ y_2(0) \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}.$$

Створимо функцію правої частини (приймемо K=1000):

```
function F=VanDerPol(x,y)
F=[y(2);-y(1)+1000*(1-y(1).^2).*y(2)];
```

Вирішуємо систему за допомогою солвера для жорстких систем та будуємо графік (рис. 10.25).

```
[X Y]=ode15s('VanDerPol',[0,3000],[2,0]);
plot(X,Y(:,1));
pause;
plot(X,Y(:,2));
```

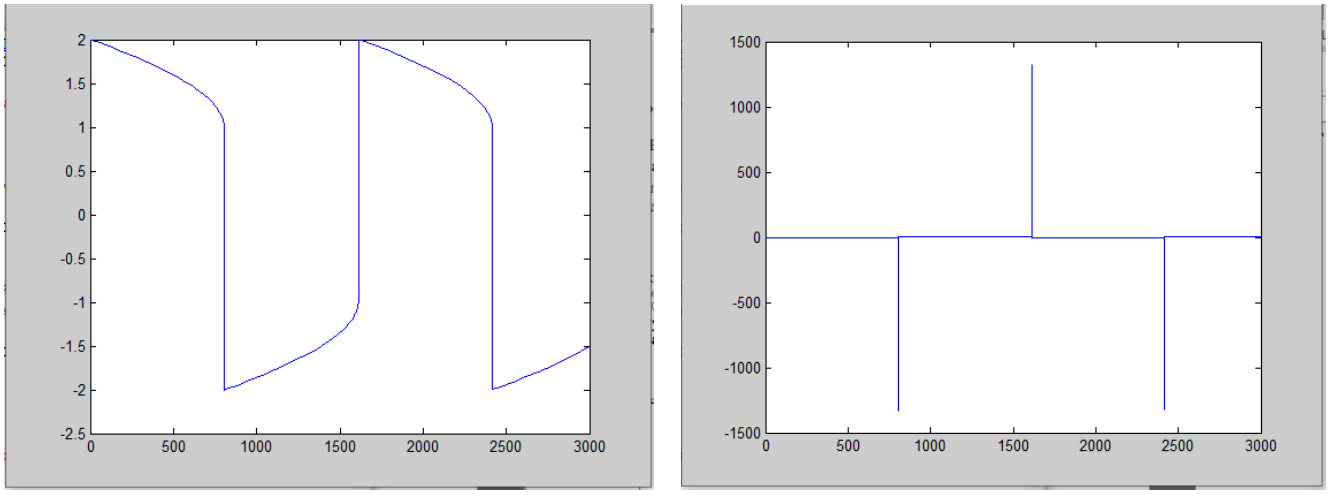


Рис. 10.25. Результат розв'язання задачі з прикладу 10.11.

Спробуйте вирішити це завдання з використанням солверів *ode*23, *ode*45 або *ode*113. Якщо вам не вистачить терпіння дочекатися закінчення розрахунку, то перервіть обчислення комбінацією клавіш *Ctrl – Break*. Справа в тому, що це завдання є прикладом так званих жорстких систем, для вирішення яких MATLAB має спеціальні солвери.

Якщо всі спроби застосування солверів *ode*45, *ode*23, *ode*113 не призводять до успіху, можливо, що розв'язувана система є жорсткою. Для вирішення жорстких систем підходить солвер *ode*15*s*, заснований на багатокроковому методі Гіра. Якщо потрібно вирішити жорстку задачу з невисокою точністю, то хороший результат може дати солвер *ode*23*s*, що реалізує однокроковий метод Розенброку другого порядку.

Отже, при вирішенні у MATLAB диференціальних рівнянь та систем з початковими умовами слід правильно вибирати солвер. Вихідне завдання може бути самою системою ЗДР. Але перед рішенням в MATLAB її слід перетворити на систему ЗДР 1-го порядку.

Приклад 10.12. Досліджуємо завдання про рух планети навколо Сонця під впливом тяжіння. Вона записується як системи ЗДР 2-го порядку:

$$\ddot{x} = -\frac{kx}{(x^2+y^2)^{\frac{3}{2}}}, \ddot{y} = -\frac{ky}{(x^2+y^2)^{\frac{3}{2}}},$$

де $x(t)$, $y(t)$ – координати планети, що рухається. Перетворимо вихідну систему до системи ЗДР 1-го порядку. Позначимо $z_1 = x, z_2 = \dot{x}, z_3 = y, z_4 = \dot{y}$. Тоді:

$$\begin{cases} z_1' = z_2 \\ z_2' = -\dfrac{kz_1}{(z_1^2+z_3^2)^{\frac{3}{2}}} \\ z_3' = z_4 \\ z_4' = -\dfrac{kz_3}{(z_1^2+z_3^2)^{\frac{3}{2}}} \end{cases}$$

Для модельного завдання виберемо k=1 і поставимо початкові умови

$$z_1(0) = 1, z_2(0) = 0, z_3(0) = 0, z_4(0) = 1.$$

Рішення зручно оформити у вигляді однієї функції з підфункцією для обчислення правої частини системи:

```
function planet                 % рішення системи
Y0=[1; 0; 0; 0.4];              % вектор початкових умов
[T,Y]=ode45(@fun,[0 20],Y0);    % вирішуємо систему
plot(Y(:,1),Y(:,3));            % графік траєкторії руху
grid on;
axis equal;
pause;
comet(Y(:,1),Y(:,3));           % анімація руху точки

function F=fun(x,y) % підфункція правої частини системи
F=[ y(2); -y(1)/(y(1).^2+y(3).^2).^(3/2);
    y(4); -y(3)/(y(1).^2+y(3).^2).^(3/2)];
```

На рис. 10.26 показана траєкторія руху планети, що отримується (крива з параметричним рівнянням $x(t)$, $y(t)$).

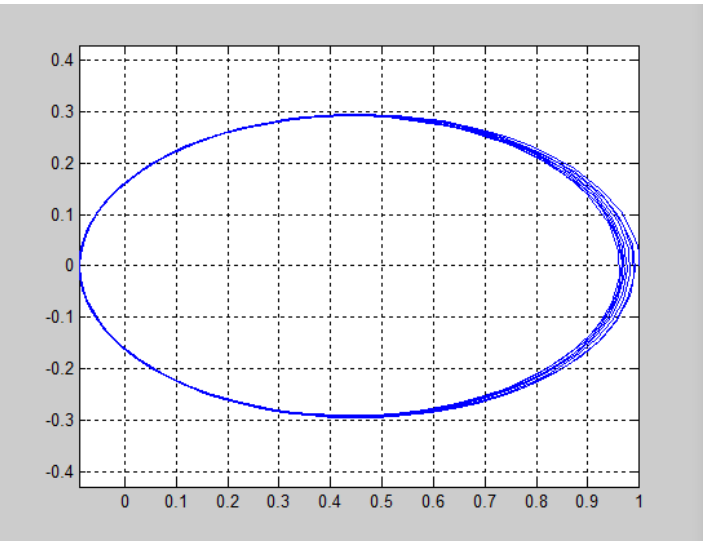


Рис. 10.26. Результат розв'язання задачі з прикладу 10.12.

Для того, щоб стежити за рухом точки траєкторією, ми використовували функцію *comet*. Вона дозволяє отримати анімований графік, на якому кружок, що позначає точку, переміщається на площині, залишаючи слід у вигляді лінії – траєкторії руху. Слідкуйте за тим, щоб вікно з графіком було поверх інших вікон.

Відомо, що траєкторія має бути замкненою кривою, а $x(t)$, $y(t)$ мають бути періодичними функціями. Однак ми бачимо деяке відхилення від замкнутості траєкторії. Це пояснюється похибкою обчислень. Підвищити точність обчислень можна наступними командами:

```
options=odeset('AbsTol',0.00000001, 'RelTol', 0.00001);
[T,Y]=ode45(@fun,[0 20],Y0, options); % вирішуємо систему
```

У коді функції *planet* замініть рядок з *ode*45 на ці два рядки.

```
function planet                          % рішення системи
Y0=[1; 0; 0; 0.4];                       % вектор початкових умов
options=odeset('AbsTol',0.00000001, 'RelTol', 0.00001);
[T,Y]=ode45(@fun,[0 20],Y0, options); % вирішуємо систему
plot(Y(:,1),Y(:,3));                     % графік траєкторії руху
grid on;
axis equal;
pause;
comet(Y(:,1),Y(:,3));                    % анімація руху точки
```

Як бачимо, при підвищенні абсолютної та відносної точності обчислень траєкторія виглядає замкнутою (рис. 10.27).

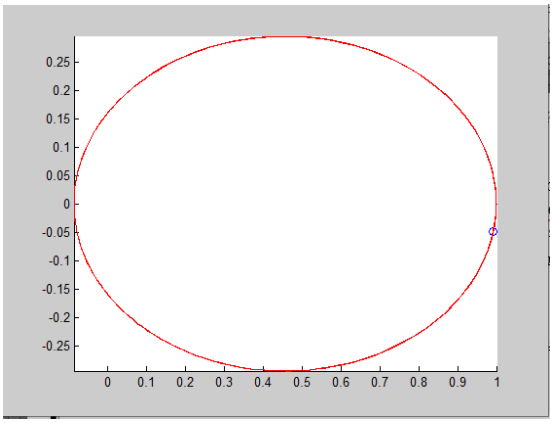


Рис. 10.27. Фазова траєкторія.

Тут ми викликали солвер *ode*45 із чотирма аргументами. Формат виклику з четвертим аргументом для всіх солверів однаковий

```
[T, Y] = solver(odefun, ,[t0, tend], y0, opts)
```

Він представляє структуру даних, що управляють ходом обчислювального процесу. Поля цієї структури слід заповнювати заздалегідь за допомогою функції *odeset*.

```
opts=odeset('name1',value1, 'name2', value2,...)
```

Вона створює нову структуру *opts*, в якій властивості із зазначеними іменами *name*1, *name*2, ... приймають такі значення *value*1,*value*2. У форматі:

```
opts=odeset(oldopts, 'name1',value1,'name2',value2,..)
```

ми змінюємо в існуючій структурі параметрів *oldopts* відповідні значення.

Можна керувати наступними параметрами солверів: точністю обчислень (параметри *RelTol, AbsTol, NormControl*), кроком інтегрування (параметри *InitialStep, MaxStep*), вихідними даними (параметри *OutputFcn, OutputSel, Refine,*

Stats), якобіаном (параметри *Jacobian*, матриць) мас і матрицею системи ЗДР (параметри *Mass, MStateDependence, MvPattern, MassSingular, InitialSlope*), подіями (параметр *Events*), два параметри тільки для *ode*15*s* (*MaxOrder, BDF*). З докладним призначенням параметрів солвер можна ознайомитися на сторінці довідки функції *odeset*. Приклади використання основних параметрів наведено нижче.

Функція *odeset* без параметрів повертає всі імена властивостей та їх допустимі значення

```
>> odeset
          AbsTol: [ positive scalar or vector {1e-6} ]
          RelTol: [ positive scalar {1e-3} ]
     NormControl: [ on | {off} ]
     NonNegative: [ vector of integers ]
       OutputFcn: [ function_handle ]
       OutputSel: [ vector of integers ]
          Refine: [ positive integer ]
           Stats: [ on | {off} ]
     InitialStep: [ positive scalar ]
         MaxStep: [ positive scalar ]
             BDF: [ on | {off} ]
        MaxOrder: [ 1 | 2 | 3 | 4 | {5} ]
        Jacobian: [ matrix | function_handle ]
        JPattern: [ sparse matrix ]
      Vectorized: [ on | {off} ]
            Mass: [ matrix | function_handle ]
MStateDependence: [ none | {weak} | strong ]
       MvPattern: [ sparse matrix ]
    MassSingular: [ yes | no | {maybe} ]
    InitialSlope: [ vector ]
          Events: [ function_handle ]
```

Найчастіше доводиться керувати точністю обчислень. Є два способи контролю точності залежно від значення параметра *NormControl*: по локальній похибці ε_i *i*-ої компоненти вектора рішень *yt* (*NormControl = off*) і по евклідовій нормі похибки (*NormControl = on*).

Крок інтегрування солвера визначається двома властивостями:

- *MaxStep* – задає максимальний крок (за умовчанням десята частина проміжку інтегрування);
- *InitialStep* - початковий крок і якщо він не заданий, то вибирається солвером самостійно.

Розглянемо приклад, де установки точності обчислень за замовчуванням вимагають зміни.

Приклад 10.13. Розв'яжемо диференціальне рівняння $y'' = -\frac{1}{t^2}$ на відрізку [*a*;100] з початковими умовами $y(a) = \ln(a)$, $y'(a) = 1/a$ при $a = 0.001$. Його точне рішення $y = \ln t$.

Наведемо завдання до системи ЗДР першого порядку та створимо функцію *example*13 для її вирішення, в якій будуємо графіки наближеного рішення з відносною точністю 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6} та графік точного рішення.

```
function example13
a=0.001;
Y0=[log(a); 1/a]; % вектор початкових умов
stl=str2mat('k-','k--','k-.','k:');
newplot; hold on; grid on;
for i=3:6
rt=10^(-i);
opts = odeset('RelTol', rt); % задаємо відносну точність
% знаходимо наближене рішення
[T,Y]=ode45(@ex7,[a 100],Y0,opts);
plot(T,Y(:,1),stl(i-2,:), 'LineWidth',2);
end
Z=log(T);
% додаємо криву точного рішення
plot(T,Z,'r', 'LineWidth',2);
legend('10^{-3}','10^{-4}','10^{-5}','10^{-6}','Location',
'SouthWest');
hold off;
function F=ex7(x,y) % функція правої частини системи
F=[y(2);-1./x.^2];
```

На рис. 10.28 червоною верхньою лінією показаний графік точного рішення та графіки наближеного рішення за різної відносної точності. Як бачимо, відносної точності 10^{-3} для даної задачі явно недостатньо (суцільна чорна нижня крива). Добре наближення до точного рішення вийшло тільки за *RelTol* = 10^{-6}.

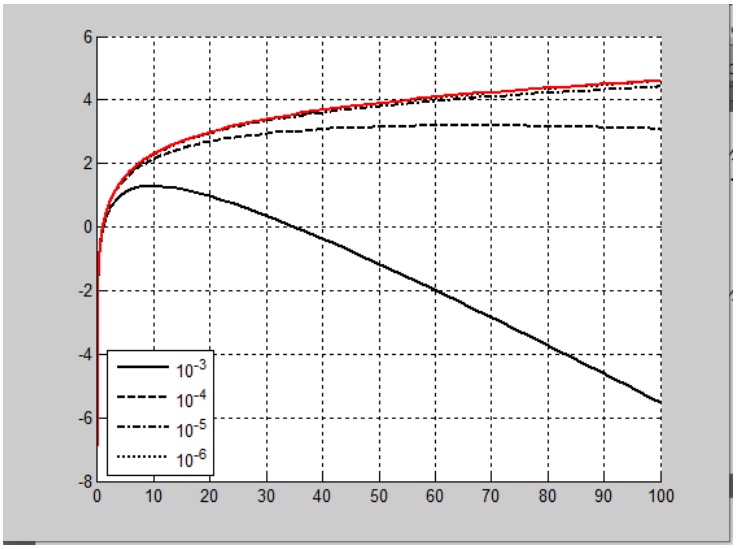


Рис. 10.28. Результат розв'язання задачі з прикладу 10.13.

Приклад 10.14. Розв'яжемо задачу Коші, яка описує рух тіла, кинутого з початковою швидкістю v_0 під кутом α до горизонту (рис. 10.29) в припущенні, що опір повітря пропорційний квадрату швидкості. У векторній формі рівняння руху має вигляд: $m\ddot{r} = -\gamma \cdot v|v| - mg$, де $r(t)$ – радіус-вектор тіла, що рухається, $v = r(t)$ – вектор швидкості тіла, γ – коефіцієнт опору, mg – вектор сили тяжіння тіла маси m; g – прискорення вільного падіння.

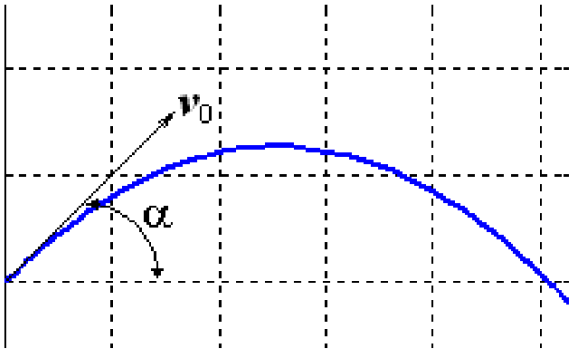


Рис. 10.29. Рух тіла, кинутого з початковою швидкістю під кутом до горизонту.

Особливість цього завдання полягає в тому, що рух закінчується наперед невідомий момент часу, коли тіло падає на землю.

Якщо позначити $k = \frac{\gamma}{m}$, то в координатній формі ми маємо систему рівнянь:

$$\ddot{x} = -k\dot{x}\sqrt{\dot{x}^2 + \dot{y}^2};$$

$$\ddot{y} = -k\dot{y}\sqrt{\dot{x}^2 + \dot{y}^2} - g,$$

до якої слід додати початкові умови: $x(0) = 0, y(0) = h$ (h початкова висота), $\dot{x}(0) = v_0 \cos\alpha, \dot{y}(0) = v_0 \sin\alpha$.

Припустимо $y(1) = x, y(2) = \dot{x}, y(3) = y, y(4) = \dot{y}$, тоді відповідна система ЗДР 1-го порядку набуде вигляду:

$$\begin{cases} y'(1) = y(2) \\ y'(2) = -ky(2)\sqrt{y(2)^2 + y(2)^2} \\ y'(3) = y(4) \\ y'(4) = -ky(4)\sqrt{y(2)^2 + y(2)^2} - g \end{cases}$$

Функція *bodyangle.m* для обчислення правої частини системи ЗДР має вигляд

```
function F=bodyangle(x,y)
k=0.01;
g=9.81;
F=[y(2);
-k.*y(2).*sqrt(y(2).^2+y(4).^2);
y(4);
-k.*y(4).*sqrt(y(2).^2+y(4).^2)-g];
```

Сценарій рішення *example*14_1.*m* нашого крайового завдання може мати вигляд (рис. 10.30)

```
% рух тіла кинутого під кутом до горизонту
alph=pi/4;              % кут кидання тіла
v0 = 1;                 % Початкова швидкість
h=0;                    % Початкова висота
tmax = 0.2;             % інтервал часу
% вектор початкових умов
```

```
Y0 = [0; v0. * cos (alph); h; v0. * sin (alph)];
% наближене рішення
[T, Y] = o de45 (@bodyangle, [0 tmax], Y0);
% графік кривої x(t), y(t)
plot(Y(:,1),Y(:,3), 'LineWidth',2);
axis equal; grid on;
```

Проте, визначення тривалості польоту нам доводиться підбирати значення t_{max}. Також доводиться «на око» визначати максимальну висоту та дальність польоту. У *ode* солверах MATLAB передбачена можливість визначення моментів настання подій, що відповідають деяким особливим значенням рішення та реакція на них. Для цього використовується спеціальна функція – обробник події. У процесі рішення MATLAB виявляє події та викликає користувальницький обробник.

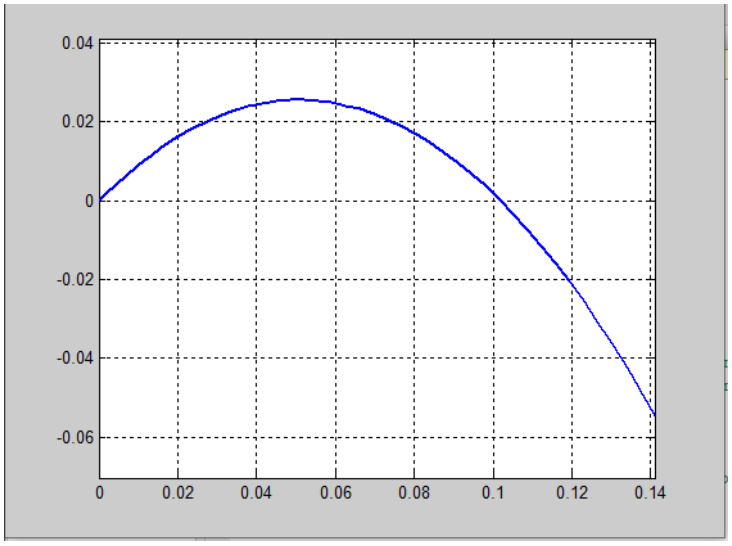


Рис. 10.30. Результат розв'язання задачі з прикладу 10.14.

Формат функції обробника події має бути наступним:

```
[value, isterminal, direction] = eventsfun(t, y)
```

Функція повинна повертати три вектори *value, isterminal, direction* однакової довжини, в яких *i*-та компонента відповідає зверненню в нуль деякого виразу, що

залежить від *t* і компонентів *y*(*k*) вектор-функції рішення системи ЗДР (аргументів *t, y* функції *events*). Компоненти цих векторів мають такий зміст:

***value*(*i*)** - вираз, складений з аргументу *t* і компонент *y*(*k*) вектор-функції рішення системи, яке всередині солвера буде перевірятися на звернення в нуль;

***isterminal*(*i*) = 1**, якщо інтегрування системи ЗДР слід зупинити при виконанні умови *value*(*i*)= 0, якщо зупиняти обчислення не потрібно;

***direction*(*i*) = 0**, якщо слід «відловлювати» всі нулі виразу *value*(*i*), + 1 – якщо слід реагувати на нулі при проходженні яких вираз *value*(*i*) зростає, *i* – 1 – якщо слід реагувати на нулі при проходженні яких *value*(*i*) зменшується.

Потім дескриптор цієї функції слід передати солверу, вказавши його у структурі параметрів *options* функції *odeset* як значення параметра *Events*.

```
options=odeset('Events', eventsfun)
```

Після цього солвер треба викликати з 5-ма вихідними параметрами

```
[T,Y,TE,YE,IE] = solver(odefun, tspan, y0, options)
```

Сенс додаткових параметрів буде описаний трохи нижче.

Для нашого прикладу функція *eventsfun* повинна реагувати на подію звернення в нуль координати *y* тіла (компоненти рішення *y*(3)) при її спаданні та звернення в нуль похідної *y* (компоненти рішення *y*(4)). З фізичних міркувань ясно, що похідна *y* звертається в нуль тільки в одній точці – у самій верхній точці траєкторії тіла, тому нам неважливо зменшується або зростає у цій точці.

Створимо наступну функцію *ex8events.m* обробник події

```
function [value,isterminal,direction] = ex8events(t,y)
% Визначати момент часу, коли висота тіла зменшується і
% стає рівною 0, і завершувати обчислення, і навіть
% визначити момент максимальної висоти (не зупиняючи
% обчислень), вона досягається коли швидкість по y дорівнює
% 0 визначати нулі для компонентів y(3) та y(4)
value = [y(3), y(4)];
```

```
isterminal = [1,0]; % зупиняти обчислення за y(3)=0
% функція y(3) зменшується, для y(4) будь-який напрямок
direction = [-1,0];
```

Подивимося, як використовувати цю функцію (рис. 10.31). Для цього скоригуємо файл сценарію в такий спосіб (сценарій *example14_2.m*)

```
alph=pi/4;          % кут кидання тіла
v0 = 1;             % Початкова швидкість
h=0;                % Початкова висота
% вектор початкових умов
Y0 = [0; v0. * cos (alph); h; v0.*sin(alph)];
% повідомляємо функцію обробник подій
options = odeset('Events', @ex8events);
% наближене рішення
[t,Y,te,ye,ie] = ode45(@bodyangle,[0 Inf],Y0,options);
% графік кривої x(t),y(t)
plot(Y(:,1),Y(:,3), '-bo', 'LineWidth',2);
axis equal;
grid on;
```

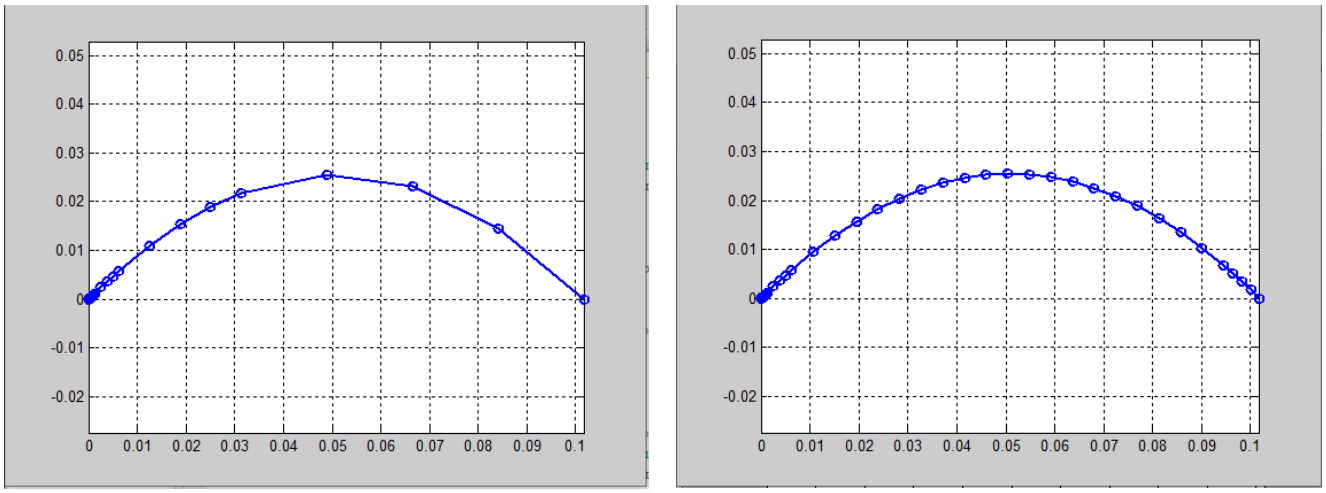


Рис. 10.31. Рух тіла кинутого під кутом до горизонту.

На рис.10.31, ліворуч показана траєкторія, обчислення якої закінчилися тоді, коли компонента рішення *y*(3), зменшуючи, стала рівною нулю. Видно, що наприкінці траєкторії крок обчислення (інтервал моментів часу) досить великий і забезпечує гладкість траєкторії. У такому разі можна використовувати параметр *MaxStep*, який, як зазначено вище, задає максимальний крок. Замінимо рядок

сценарію, що містить виклик функції *odeset* наступним рядком

```
options = odeset('Events',@ex8events,'MaxStep',0.025);
```

На графіку праворуч показано траєкторію, побудовану за модифікованим сценарієм. Гладкість кривої стала вищою.

```
alph=pi/4;          % кут кидання тіла
v0 = 1;             % Початкова швидкість
h=0;                % Початкова висота
% вектор початкових умов
Y0 = [0; v0. * cos (alph); h; v0.*sin(alph)];
% повідомляємо функцію обробник подій
options = odeset('Events',@ex8events,'MaxStep',0.025);
% наближене рішення
[t,Y,te,ye,ie] = ode45(@bodyangle,[0 Inf],Y0,options);
% графік кривої x(t),y(t)
plot(Y(:,1),Y(:,3), '-bo', 'LineWidth',2);
axis equal;
grid on;
```

Однак зменшення кроку за допомогою параметра *MaxStep* збільшує кількість кроків, що використовуються чисельним методом, та збільшує час розв'язання системи ЗДР. Є ще один параметр *Refine*, який керує кількістю точок рішення, що повертаються солвером на кожному інтервалі. Якщо *Refine* дорівнює 1, то солвер обчислює значення рішення лише у кінцевих точках інтервалів часу, використовуваних чисельним алгоритмом. Якщо *Refine* дорівнює $n > 1$, то солвер ділить кожен часовий відрізок на n підінтервалів і повертає рішення у кожній точці розподілу. При цьому використовуються внутрішні формули для обчислення рішення у точках тимчасового відрізка, а не застосовується алгоритм чисельного методу на дрібному розбитті. Це заощаджує час обчислення. Всі солвери, крім *ode*45 за замовчуванням, використовують значення *Refine* = 1, а *ode*45 за замовчуванням використовує значення 4.

Параметр *Refine* не працює, коли ви явно задаєте моменти часу *tspan*, які слід отримати рішення, тобто, коли замість аргументу *tspan* = [t_0, t_{end}] визначається вектор довжини *length*(*tspan*) > 2.

Замініть у файлі *example14_2.m* рядок, що задає параметри солвера, на наступний рядок

```
options = odeset('Events',@ex8events,'Refine',8);
```

і виконайте його.

Як бачимо (рис. 10.32), часові інтервали не стали рівними, проте гладкість кривої підвищилася.

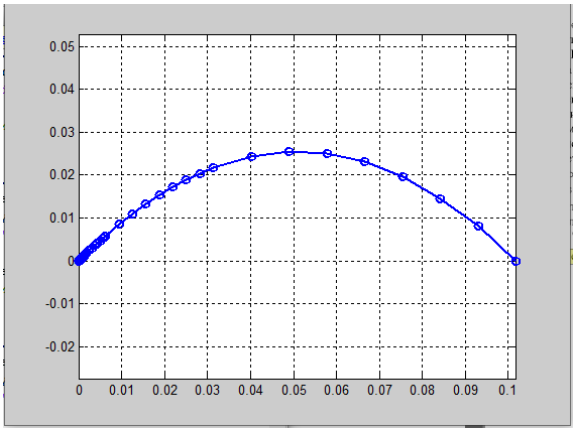


Рис. 10.32. Рух тіла кинутого під кутом до горизонту.

Зверніть увагу на те, що ми використовували солвер з 5-ма вихідними аргументами. Якщо солверу в четвертому аргументі *options* передається параметр *Events*, він (солвер) повертає п'ять параметрів, які можна прийняти так

```
[T,Y,TE,YE,IE] = solver(odefun, tspan, y0, options)
```

де додаткові параметри, що повертаються, мають наступний сенс: *TE* – вектор стовпець моментів часу, в які відбуваються події; *YE* – значення рішення у ці моменти часу; *IE* – індекси *k* компонент виразів *value*, для яких відбулися зазначені події. Так у нашому прикладі ми отримуємо:

```
te            % моменти настання події
te =
0.0721
0.1441
ye            % значення рішення у моменти настання подій
ye =
0.0509 0.7067  0.0255  0
0.1019 0.7063 -0.0000 -0.7068
ie
ie =
2
1
```

У прикладі функція *ex8events* обробник події *Events* повертає вектор [*y*(3), *y*(4)]. Тому вектор, що повертається *ie*=[2; 1] означає, що спочатку настала подія звернення в нуль функції *y*(4) (друга компонента вектора, що повертається функцією *ex8events*), а потім – *y*(3) (перша компонента). Ці події відбулися моменти часу *te*=[0.0721; 0.1441], а значення вектор – функції [*y*(1), *y*(2), *y*(3), *y*(4)] рішення в ці моменти знаходяться в матриці *ye*. Зокрема, найвища точка траєкторії має координати [0.0509 0.0255], тривалість польоту склала *tmax* = 0.1441 (друга компонента вектора *te*), а дальність польоту *xmax* = 0.1019 (перша компонента в другому рядку матриці *ye*).

Якщо солвер викликається з одним аргументом, що повертається у форматі

```
sol = solver(odefun, [t0, t1], y0, opts)
```

то структура *sol* матиме додаткові поля *sol.xe, sol.ye, і sol.ie*, відповідні параметрам *TE, YE, IE*, що повертаються, описаним вище. Наприклад, якщо у нашому прикладі для вирішення використати команду

```
sol = ode45(@bodyangle,[0 Inf],Y0,opts);
```

то будемо мати

```
sol.xe
```

```
ans =
0.0721    0.1441
sol.ye
ans =
0.0509    0.1019
0.7067    0.7063
0.0255    -0.0000
0         -0.7068
sol.ie
ans = 2  1
```

Продемонструємо можливості використання параметра *Events* на прикладі розв'язання задачі про м'ячик, що стрибає.

Приклад 10.15. Пружний м'ячик має початкове положення та швидкість. Опір повітря дуже малий, але енергія руху розраховується при відскоку м'яча від землі. Нехай при відскоку від землі вертикальна швидкість м'ячика становить 90% від вертикальної швидкості під час падіння.

Рівняння руху (без урахування опору повітря) має вигляд $m\ddot{r} = -mg$, де g – вектор прискорення вільного падіння, що має напрямок вертикально вниз. У покоординатній формі маємо $x'' = 0, y'' = -g$, та початкові умови $x(0) = x_0$, $y(0) = h$, $x'(0) = v_0 x, y'(0) = v_0 y$. Система розпадається на два незалежні рівняння, перше з яких має розв'язок $x(t) = x_0 + v_0 x \cdot t$. Це свідчить про те, що у горизонтальному напрямі рух відбувається із постійною швидкістю $v_0{}^x$. У нашому прикладі покладемо $x_0 = 0$. Друге рівняння також інтегрується, але хочемо показати, як можна використовувати параметр *Events*, і вирішуватимемо рівняння чисельно. У момент відскоку *tk* вертикальна швидкість *y* змінює знак і стає позитивною.

Функція правої частини системи ЗДР 1-го порядку має вигляд

```
function F=fun9(x,y)
g=9.81;
F=[y(2); -g];
```

Функція обробник події має вигляд

```
function [value,isterminal,direction] = events9(t,y)
%Визначати момент часу, в який висота тіла
% дорівнює 0 при її спаданні і завершувати обчислення
value = y(1);     % визначати нульову висоту
isterminal = 1;   % зупиняти обчислення за y(1)=0
direction = -1;   % висота y(1) зменшується
```

Сценарій для моделювання руху до першого приземлення м'ячика на землю можна записати таким чином (файл-сценарій *example*15_1.*m*):

```
h=0;                % Початкова висота
vox=1;              % Початкова швидкість вздовж осі x
voy=10;             % Початкова швидкість вздовж осі y
Y0=[h; voy];        % вектор початкових умов
% задаємо параметри
opts = odeset('Events',@events9,'MaxStep',0.1);
% y координата м'ячика
[t,Y,te,ye,ie] = ode45(@fun9,[0 Inf],Y0,opts);
X=vox.*t;           % x - координата м'ячика
% графік кривої x(t), y(t)
plot(X,Y(:,1), '-bo', 'LineWidth',2);
axis equal;
grid on;
```

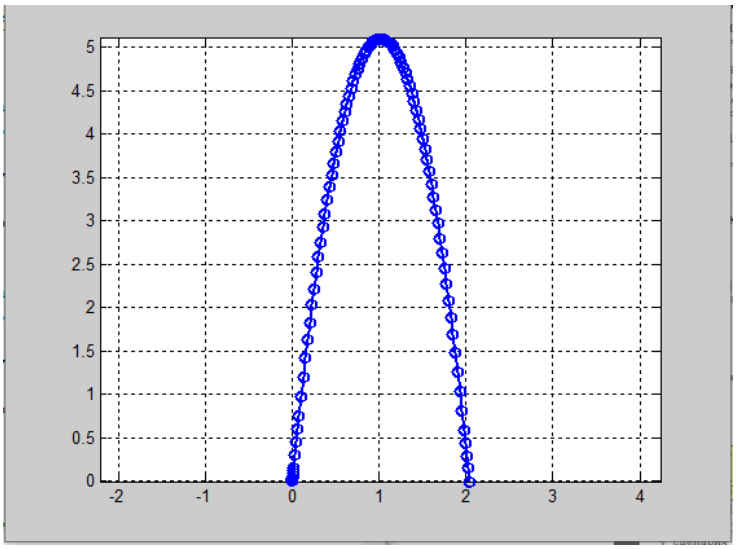


Рис. 10.33. Результат розв'язання задачі з прикладу 10.15.

Ми хочемо змоделювати рух при багаторазовому відскоку та приземленні м'ячика. Для цього виправимо сценарій таким чином (сценарій *example*15_2.*m*):

```
vox=1;              % Початкова швидкість вздовж осі x
voy=10;             % Початкова швидкість вздовж осі y
tstart = 0;
tfinal=Inf;
Y0 = [0; voy];      % вектор початкових умов
opts = odeset('Events',@events9,'MaxStep',0.1);
for i=1:6           % задаємо параметри
[t,Y,te,ye,ie]=ode23(@fun9,[tstart tfinal],Y0,opts);
X=vox.*t;           % y координата м'ячика
plot(X,Y(:,1),'-bo','LineWidth',2);%графік кривої x(t),y(t)
hold on;
tstart=te;
Y0 = [0; 0.9.*abs(ye(2))];
end
axis equal;
grid on;
hold off;
```

Отриманий графік траєкторії руху при векторі початкової швидкості $v(0) = (v_0^x, v_0^y) = (1, 10)$ показаний на наступному рис. 10.34.

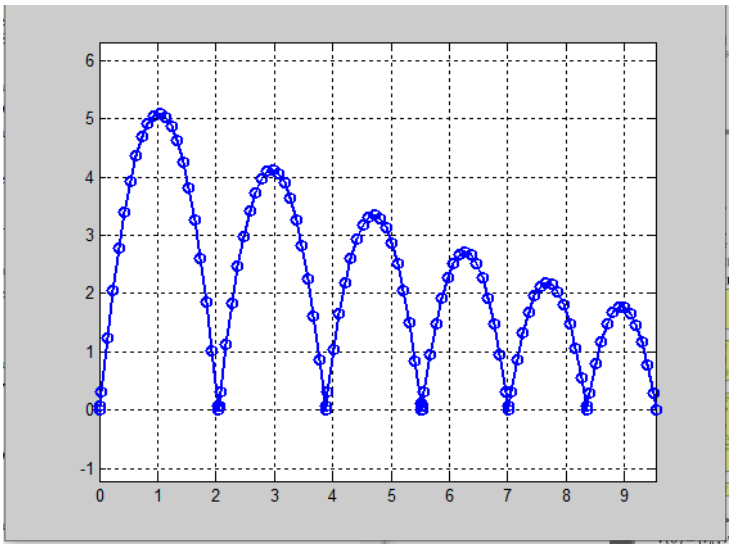


Рис. 10.34. Моделювання руху м'ячика, що стрибає.

Якщо ми хочемо будувати криві при різних значеннях початкової швидкості, замість сценарію зручно використовувати функцію з аргументами, що задають початкові значення

```
function funex15(vox,voy)
% моделювання руху м'ячика, що стрибає
% vox - початкова швидкість вздовж осі x
% voy - початкова швидкість вздовж осі y
% задаємо параметри
opts = odeset('Events',@events9,'MaxStep',0.01);
tfin=[];  % єдиний вектор моментів часу для побудови
анімації
yfin=[]; % єдиний вектор координат y для побудови анімації
tstart = 0;
tfinal=Inf;
Y0 = [0; voy]; % вектор початкових умов
for i=1:6
% Рішення
[t,Y,te,ye,ie] = ode23(@fun9,[tstart tfinal],Y0,opts);
X=vox.*t;
plot(X,Y(:,1), 'LineWidth',2); % графік ділянки кривої
hold on;
tstart=te; % Початковий момент для наступної ділянки
% початкові умови для наступної ділянки
Y0 = [0; 0.9.*abs(ye(2))];
tfin = [tfin t ']; % поєднуємо моменти часу
yfin=[yfin Y(:,1)']; % поєднуємо координати y
end
axis equal; grid on; hold off;
pause;
X=vox.*tfin;
comet(X, yfin); % анімація руху м'яча
```

Наведений код доповниний рядками для побудови анімації руху м'ячика за допомогою функції *comet*.

Приклад 10.16. Повернемося до прикладу 10.14, в якому ми вирішували задачу руху тіла, кинутого під кутом до горизонту. Створимо функцію *bodyanglek.m* правої частини системи, що має додатковий параметр – коефіцієнт

опору середовища *k*.

```
function F=bodyanglek(x,y,k)
g=9.81;
F=[y(2); -k.*y(2).*sqrt(y(2).^2+y(4).^2); y(4); -
k.*y(4).*sqrt(y(2).^2+y(4).^2)-g];
```

Якщо використовується функція обробник події параметра *Events*, вона також повинна мати цей додатковий аргумент.

```
function [value,isterminal,direction] = ex8eventsk(t,y,k)
% Визначати момент часу, в який висота тіла стає
% рівної 0 при її спаданні та завершувати обчислення, а
також
% визначити момент максимальної висоти (не зупиняючи
% обчислень), яка досягається коли швидкість по y дорівнює
0
value = [y(3), y(4)]; % визначати нулі для y(3) та y(4)
isterminal = [1,0];   % зупиняти обчислення за y(3)=0
direction = [-1,0];   % функція y(3) зменшується
                      % для y(4) без різниці
```

При виклику солвера ми також маємо передати додатковий аргумент. Тоді сценарій рішення *ex*16_1.*m* нашого крайового завдання буде мати вигляд:

```
% рух тіла кинутого під кутом до горизонту ex16_1.m
alph=pi/4;        % кут кидання тіла
v0 = 10;          % Початкова швидкість
% вектор початкових умов
Y0 = [0; v0. * cos (alph); 0; v0. * sin (alph)];
ak=[0.1 0.2 0.3 0.5]; % коефіцієнти опору середовища
opts    =    odeset('Events',@ex8eventsk,'Refine',16);    %
параметри
Newplot; hold on;
for i=1:4
[t,Y,te,ye,ie] = ode45(@bodyanglek,[0 Inf],Y0,opts,ak(i));
% Рішення
plot(Y(:,1),Y(:,3), 'LineWidth',2); % графік траєкторії
end
grid on; hold off;
```

Однак, найкращим рішенням є використання анонімних функцій. Ось приклад сценарію *exm*16_2.*m*, що вирішує те саме завдання, в якому ми щоразу з функції *bodyanglek*(*x, y, k*) з трьома аргументами створюємо функцію *ba*=@(*x,y*) *bodyanglek*(*x, y, ak*(*i*)) правої частини системи, що містить лише два аргументи.

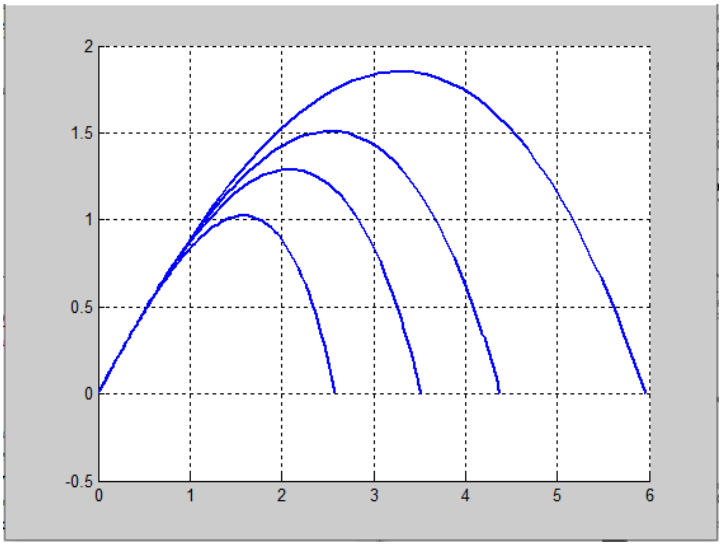


Рис. 10.35. Результат розв'язання задачі з прикладу 10.16.

```
% рух тіла кинутого під кутом до горизонту exm16_2.m
alph=pi/4; % кут кидання тіла
v0 = 10; % Початкова швидкість
Y0 = [0; v0. * cos (alph); 0; v0. * sin (alph)];
% вектор початкових умов
ak=[0.1 0.2 0.3 0.5]; % коефіцієнти опору середовища
opts = odeset('Events',@ex8events,'Refine',16); % параметри
Newplot; hold on;
for i=1:4
ba = @ (x, y) bodyanglek (x, y, ak (i));
% наближене рішення
[t,Y,te,ye,ie] = ode45(ba,[0 Inf],Y0,opts);
plot(Y(:,1),Y(:,3), 'LineWidth',2); % графік траєкторії
end
grid on;
hold off;
```

У цьому ми використовували колишній обробник подій – функцію *ex*8*events*(*t, y*) без додаткового параметра.

Раніше ми зазначали, що виклик солверів без значення, що повертається, призводить до побудови графіка рішення. Можливості виведення результату, що надаються солверами MATLAB, не вичерпуються лише в такий спосіб візуалізації рішення. Користувач може вибрати альтернативне графічне представлення результату або навіть створювати свої функції для побудови графіків. Для цього слід скористатися параметром *OutputFcn*. Його значення має бути дескриптором функції (або рядком з її ім'ям), яка виконує необхідні операції. Є кілька стандартних функцій:

- *odeplot* – побудова графіків компонентів рішення;
- *odephas*2 – побудова фазових траєкторій для двовимірних систем;
- *odephas*3 – побудова фазових траєкторій для тривимірних систем;
- *odeprint* – виведення числової інформації про рішення.

За замовчуванням параметр *OutputFcn* має значення дескриптора функції *odeplot*.

Фактично *OutputFcn* містить назву функції, яка виконується після кожного успішного кроку інтегрування. Нехай є завдання:

$$\begin{cases} y_1' = y_2 \\ y_2' = -y_2 + 5y_1 \ + \sin t \end{cases}, \begin{pmatrix} y_1(0) \\ y_2(0) \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

Створюємо функцію правої частини системи

```
function F=fun10(x,y)
F=[y(2);-y(2)-5.*y(1)+sin(x)];
```

Виконуємо команди

```
opts = odeset('OutputFcn',@odeplot);
ode45(@fun10,[0 20],[1;0], opts);
grid on;
```

На наступному рисунку зліва показаний отриманий графік. Такий же графік ми отримуємо при виклику солвера без параметрів, що повертаються, і без

вказівки значення параметра *OutputFcn*.

```
ode45(@fun10,[0 20],[1;0]);
grid on;
```

Наступні команди будують фазову траєкторію (рис. 10.36, графік праворуч):

```
opts = odeset('OutputFcn',@odephas2);
ode45(@fun10,[0 20],[1;0], opts);
grid on;
```

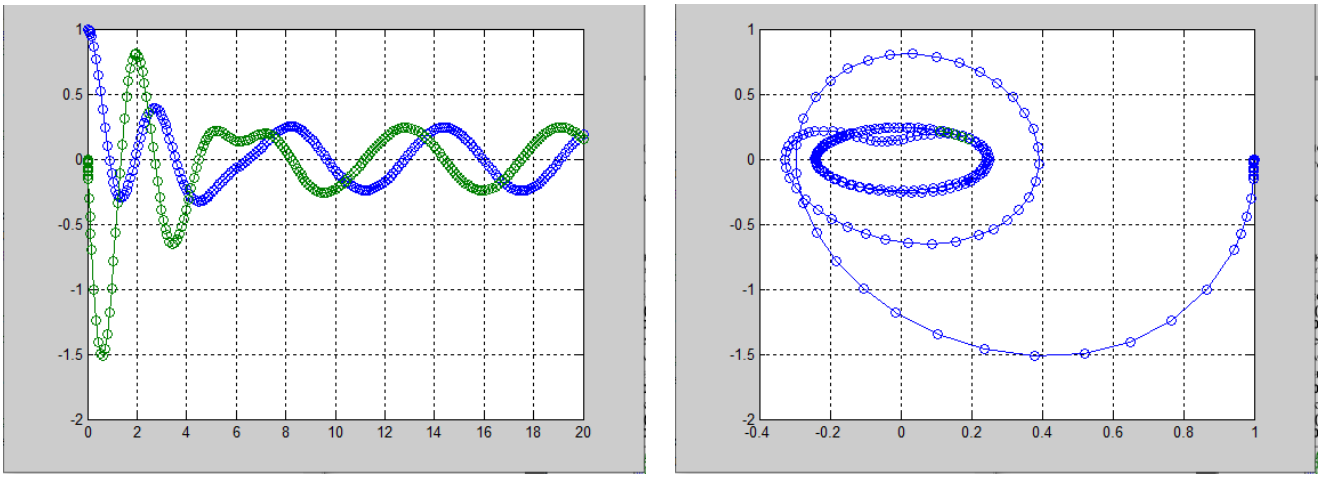


Рис. 10.36. Результат розв'язання задачі.

Для системи третього порядку, розглянутої на прикладі 10.10, з функцією правої частини

```
function F=fun5(x,y) % функція правої частини системи
F=[y(2); y(3); x.^2.*y(3)-x.*y(2)+y(1)];
```

команда

```
ode45(@fun5,[0 1],[0; 2; -5]);
grid on;
```

будує три графіки. Вони показані на малюнку ліворуч. А команди

```
opts = odeset('OutputFcn',@odephas3);
```

```
ode45(@fun5,[0 1],[0; 2; -5], opts);
grid on;
```

будують тривимірну фазову траєкторію (рис. 10.36, графік праворуч).

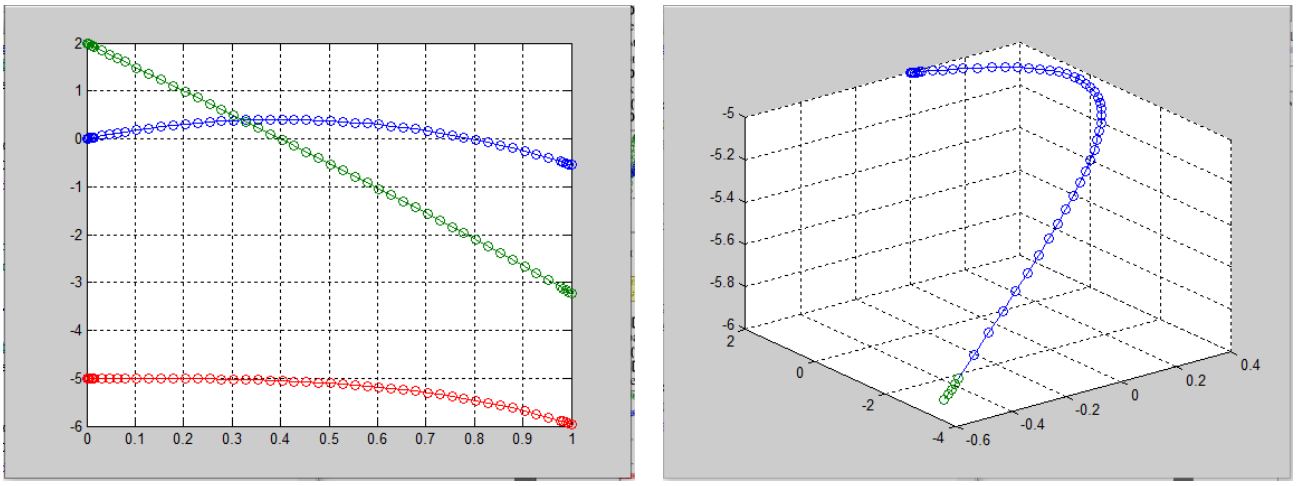


Рис. 10.37. Результат розв'язання задачі.

Приклад 10.17. Розв'яжемо систему рівнянь:

$$\begin{cases} y_1' = s(y_2 - y_1) \\ y_2' = y_1(r - y_3) - y_2 \\ y_3' = y_1 \cdot y_2 - by_3 \end{cases}$$

де *s*, *r*, *b* – параметри. Створимо функцію правої частини системи

```
function f=lor(t,y,s,r,b)
% права частина системи рівнянь Лоренца
f=[s.*(y(2)-y(1)); -y(2)+(r-y(3)).*y(1);
-b.*y(3)+y(1).*y(2)];
```

У вікні коду задаємо значення параметрів *s* = 10; *r* = 25; *b* = 3;

```
s=10; r=25; b=3;
```

Щоб виклик функції *ode*45 без значень, що повертаються, побудував графік двох компонентів рішення (а не трьох) задаємо параметр *OutputSel*, в якому вказуємо номери переданих компонентів (рис. 10.38):

```
opts = odeset('OutputSel',[2 3]);
ode45(@lor,[0 20],[1 -1 10], opts, s, r, b);
grid on;
```

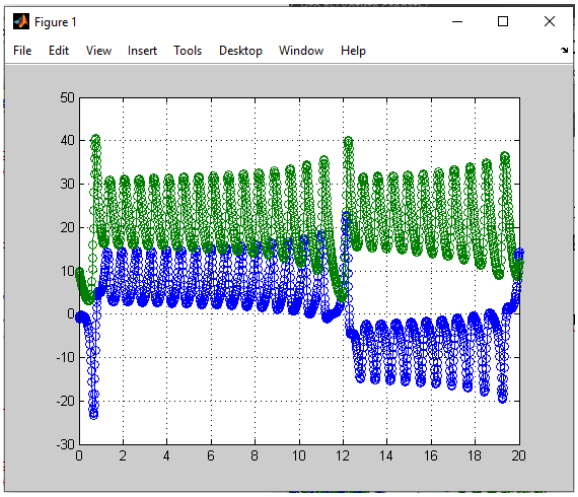


Рис. 10.38. Результат розв'язання задачі з прикладу 10.17.

Для побудови тривимірної фазової траєкторії (рис. 10.39, ліворуч) виконаємо команди:

```
opts = odeset( 'OutputFcn', @odephas3 );
ode45(@lor, [0 20], [1 -1 10], opts, s, r, b);
```

А для побудови двовимірної фазової траєкторії компонент 1 і 3 (рис. 10.39, праворуч) виконаємо команди:

```
opts=odeset('OutputSel',[1 3],'OutputFcn',@odephas2);
ode45(@lor, [0 20],[1 -1 10], opts, s, r, b );
```

Якщо використовувати значення, що повертаються, то попередні графіки можна побудувати, наприклад, таким чином (рис. 10.40):

```
[T,Y]=ode45(@lor,[0 20],[1 -1 10], [ ],s, r, b);
% Графік кривих 2 і 3
plot(T,Y(:,2:3),'LineWidth',2); grid on;
% 3d фазова траєкторія
plot3(Y(:,1), Y(:,2), Y(:,3));
grid on;
```

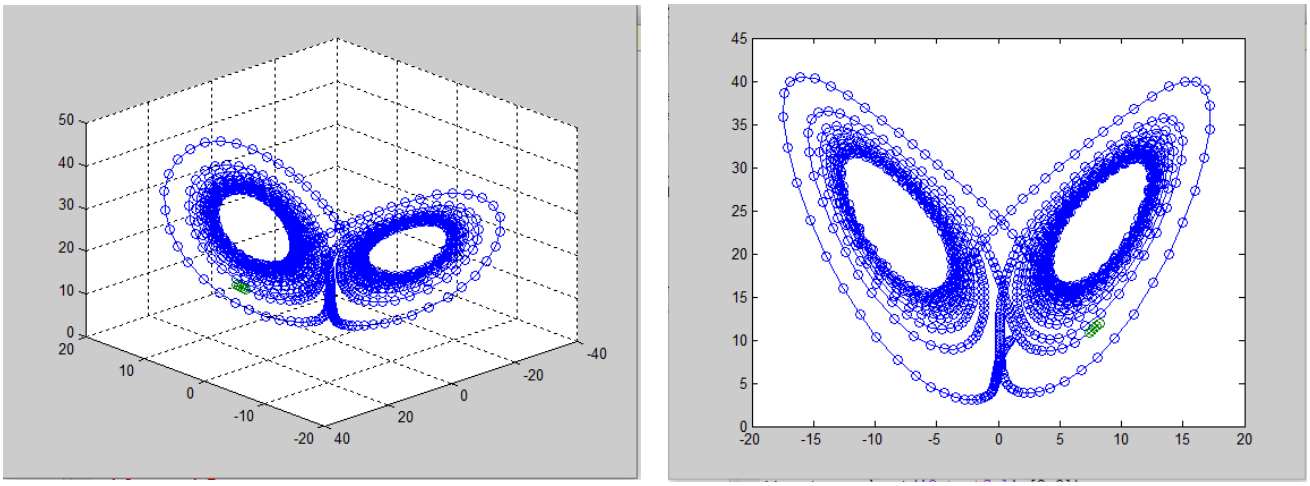


Рис. 10.39. Фазові траєкторії.

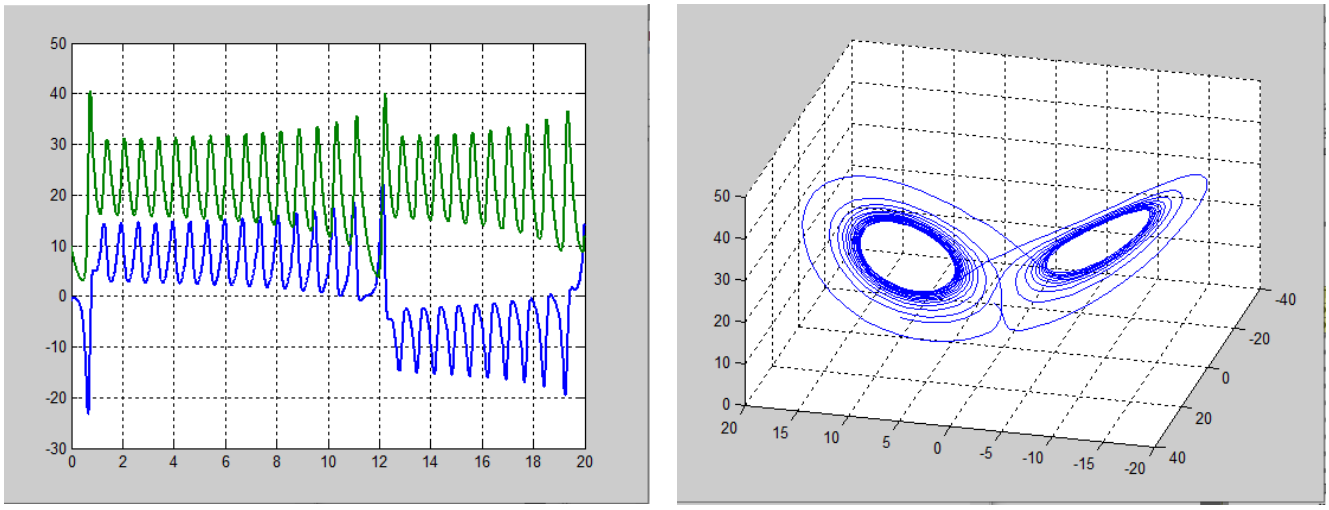


Рис. 10.40. Рішення та фазова траєкторія.

Рух точки фазової траєкторії компонент 1 і 3 можна виконати командою

```
comet(Y(:,1), Y(:,3));
```

А рух тривимірною фазовою траєкторією – командою

```
comet3(Y(:,1),Y(:,2), Y(:,3));
```

Якщо ми захочемо продовжити розрахунок, розпочавши рішення системи в точці в якій закінчився попередній розрахунок, слід кінцеві значення рішення першого розрахунку використовувати як початкові значення для наступного (рис. 10.41).

```
[TY , Y]=ode45(@lor,[0 20],[1 -1 10], [ ],s, r, b);
Z0=Y(end, :) % початкові значення другого розрахунку
[TZ , Z]=ode45(@lor,[20 40],Z0, [ ],s, r, b);
plot(TY,Y(:,2:3),'r', 'LineWidth',2); hold on;
plot(TZ,Z(:,2:3),'b', 'LineWidth',2); grid on;
hold off;
Z0 =
8.2178   14.3566  11.9137
```

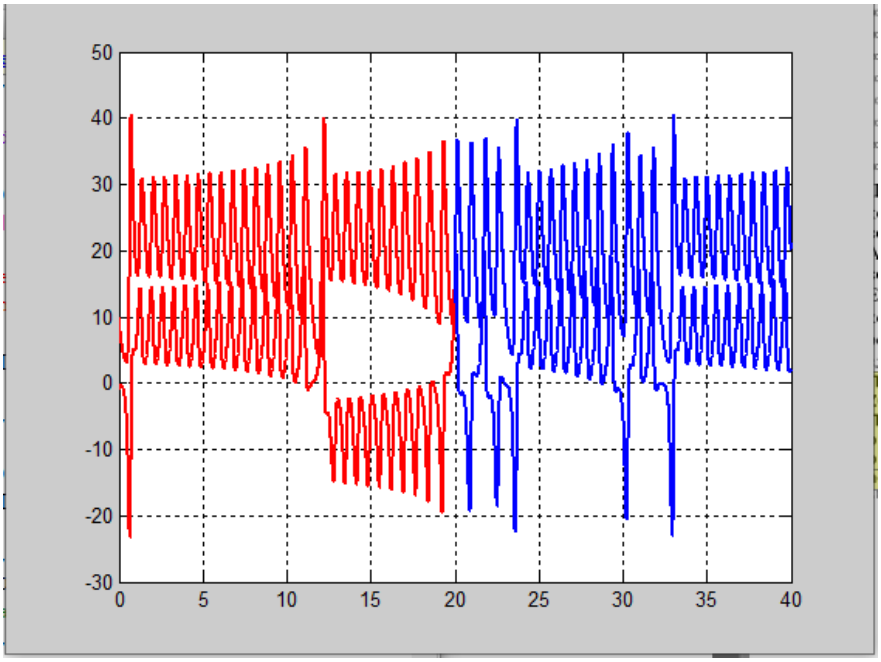


Рис. 10.41. Результат подальшого рішення.

Щоб переглянути положення деяких точок фазової траєкторії (рис. 10.42), виконаємо команди:

```
plot(Y(:,1), Y(:,3),'k',
Y(end-10,1),Y(end-10,3),'or',
Y(end-20,1),Y(end-20,3),'ob');
[Y(end-10,1),Y(end-10,3)]
ans =
2.6544    9.0760
```

Усі наведені команди зручно зібрати в сценарій.

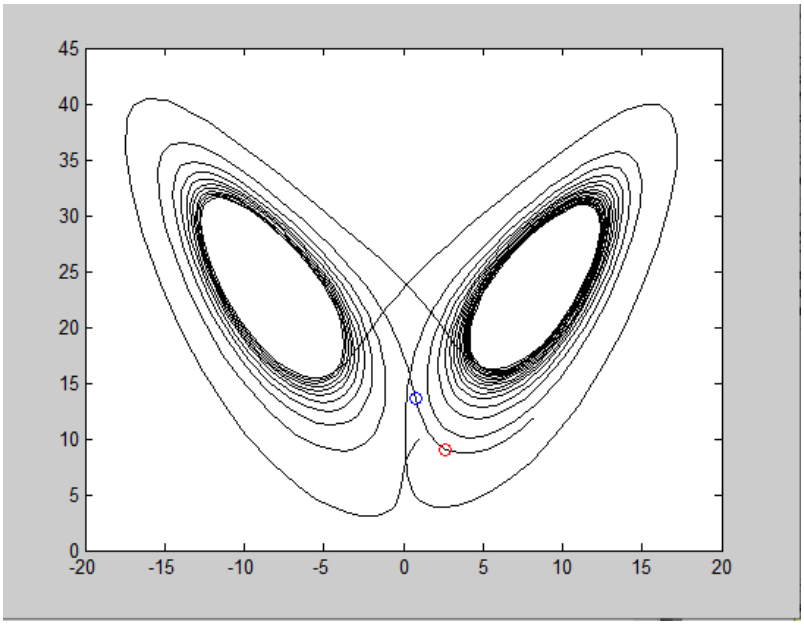


Рис. 10.42. Фазова траєкторія.

Користувач може створювати свої файл-функції для візуалізації рішення або обробки результатів кожного кроку чисельного інтегрування. І тому він має задати значення параметра *OutputFcn*, тобто надати йому значення дескриптора на свою функцію. Ця функція буде викликати солвер перед першим кроком інтегрування, після кожного кроку і в кінці. Залежно від значення, що повертається, солвер може зупиняти обчислення або продовжувати їх. Заголовок функції повинен мати вигляд

```
function status=outfun (t, y, flag)
```

Вхідний аргумент *flag* є рядковою змінною і може набувати одного з трьох значень:

'init' – передається солвером при першому виклику до початку інтегрування, параметр *t* є вектором двох елементів - меж відрізка інтегрування, а *y* – є вектором початкових значень;

' ' (порожній рядок) передається після кожного кроку інтегрування, параметр *t* є поточним значенням аргументу *y* – вектором наближених значень на

поточному кроці; аргумент *t* може містити кілька поточних значень тоді *y* - матриця, кожен стовпець якої містить компоненти рішення для відповідного моменту часу;

'done' передається після завершення чисельного рішення системи, *t* та *y* є порожніми масивами.

Коли *flag* є порожнім рядком, довжина вхідного аргументу визначається значенням параметра *Refine*. За умовчанням воно дорівнює 1 і всі солвери (крім *ode*45) на кожному кроці будуть передавати функції *outfun* у параметрі *t* лише одне значення незалежної змінної, а в параметрі *y* – вектор значень рішення в цей момент. Функція *outfun* повинна повертати у вихідному аргументі *status* 0 або 1. Якщо солвер виявляє, що функція *outfun* повернула 1, то процес рішення закінчується, а якщо 0 – то триває.

Цієї функції можна передавати значення не всіх координат вектора рішення *y*. Для вказівки вектора індексів компонента рішення, які слід передавати в *OutputFcn*, використовується параметр *OutputSel*. Його значенням має бути вектор із цілих чисел – номерів компонентів вектора рішення. За замовчанням передаються всі компоненти.

10.5. Функції, пов'язані із солверами

Існують дві функції, призначені для спільного використання з солверами *odextend* і *deval*. Функція *odextend* використовується для продовження рішення, отриманого за допомогою солвер, а *deval* – для отримання значень рішень у заданих точках.

Розглянемо рівняння, яке ми вирішили в прикладі 10.6. Функція правої частини системи має вигляд:

```
function F=funToEx(x,y) % функція правої частини системи
F=[y(2); -y(1)^3+y(1)];
```

Розв'яжемо завдання на відрізку [0, 8]

```
sol=ode45(@funToEx,[0 8],[0, 0.1]); % вирішуємо систему
```

Функція *odeextend* дозволяє отримати рішення на ширшій ділянці, наприклад, на відрізку [0 16].

```
sol2=odextend(sol,@funToEx,16);
plot(sol2.x,sol2.y(1,:));
```

Першим аргументом *odextend* є структура *sol*, яку повертає солвер під час вирішення задачі Коші. Другим – дескриптор функції *@funToEx* правої частини системи, третім – кінцеве значення незалежної змінної. Немає необхідності повторно передавати ім'я функції, яка обчислює праву частину системи ЗДР. Це ім'я зберігається у структурі *sol* і другим аргументом можна передавати порожній вектор. Крім того, у структурі зберігається ім'я солвера та його параметри. Тому завдання вирішується з використанням того ж самого солвера і тих самих параметрів.

```
sol=ode45(@funToEx,[0 8],[0, 0.1]);
solext=odextend(sol,[ ],16);
plot(solext.x,solext.y(1,:));
```

Допустимо використання наступних форматів виклику:

```
solext = odextend(sol, odefun, tfinal)
solext = odextend(sol, [], tfinal)
solext = odextend(sol, odefun, tfinal, yinit)
solext = odextend(sol, odefun, tfinal, [yinit, ypinit])
solext = odextend(sol, odefun, tfinal, yinit, option)
```

Аргумент *tfinal* вказує на нове кінцеве значення незалежного аргументу. Останнє значення рішення *sol.y*(:, *end*), отримане солвером першому етапі, є початковим значенням продовження рішення. Якщо ви хочете змінити це початкове значення або змінити параметри обчислювального процесу, слід

використовувати інші формати виклику функції *odextend*. Структура *solext,* яка повертається, має такі самі поля, як і вихідна структура *sol* і може використовувати таким же чином, як *sol*.

Функція *deval* рішення ЗДР, яке солвер повертає у структурі *sol*, обчислює у заданих точках. Формат її виклику наступний

```
sxint = deval(sol,xint)
sxint = deval(xint,sol)
sxint = deval(sol,xint,idx)
sxint = deval(xint,sol,idx)
[sxint, spxint] = deval(...)
```

Перші два виклики еквівалентні. Чисельне рішення задачі Коші або крайової задачі повертається у структурі *sol*, яка є одним із аргументів функції *deval*. Інший аргумент *xint* є точкою або вектором значень незалежної змінного, в яких ви бажаєте знати рішення. Елементи вектора *xint* повинні бути на відрізку [*sol.x*(1), *sol.x*(*end*)]. Вектор *sxint* містить значення для кожного елемента вектора *xint*.

Третій та четвертий виклики функції *deval* дозволяють у векторі *idx* передати номери тих компонентів вектора рішення, які ви бажаєте отримати. Виклик функції *deval* з двома параметрами, що повертаються дозволяє крім значення вектора *sxint* рішення задачі в зазначених точках, отримати значення похідної рішення *spxint* в цих точках, які виходять шляхом поліноміальної інтерполяції рішення. Наприклад, для вирішення *sol* останнього завдання можна отримати значення рішення в точках 1, 2, 3, 4 наступним чином:

```
deval(sol,[1,2,3,4])
ans =
0.1175 0.3582 0.8960 1.4182
0.1540 0.3606 0.7006 0.0413
```

Кожен стовпець містить вектор значень рішення в момент часу, вказаний у відповідному стовпчику *xint* вектора. Команда:

```
[xv,pp]=deval(sol,[1,2,3,4])
xv =
0.1175 0.3582 0.8960 1.4182
0.1540 0.3606 0.7006 0.0413
pp =
0.1540 0.3604 0.7010 0.0416
0.1158 0.3122 0.1736 -1.4352
```

повертає самі значення рішення та їх похідні. Зверніть увагу, що перший рядок похідних *pp* близький до другого рядка вектора *xv*. Це тому, що в нашому прикладі другий компонент рішення є похідною першої компоненти.

Контрольні запитання:

1. Які засоби для обчислення чисельних значень похідних кінцево-різницевим методом є у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?
2. Які засоби обчислення інтегралів є у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?
3. Які засоби символьного розв'язання диференціальних рівнянь є у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?
4. Які засоби чисельного розв'язання нелінійних рівнянь є у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?
5. Які засоби чисельного розв'язання задачі Коші для звичайних диференціальних рівнянь є у системі комп'ютерної математики MATLAB, наведіть приклади їх застосування?
6. Які існують функції (солвери) для вирішення задачі Коші у системі комп'ютерної математики MATLAB та чим вони відрізняються?

СПИСОК ЛІТЕРАТУРИ

1. Machine and Deep Learning Using MATLAB. Algorithms and Tools for Scientists and Engineers / *Kamal I. M. Al-Malah* // Wiley, 2023. 592 p.

2. Introduction to Intelligent Systems, Control, and Machine Learning Using MATLAB / *Marco Schoen* // Cambridge University Press, 2023. 450 p.

3. MATLAB Applications in Chemical Engineering / *Chyi-Tsong Chen* // Unknown Publisher, 2022. 576 p.

4. MATLAB and Simulink Crash Course for Engineers / *Eklas Hossain* // Springer International Publishing, 2022. 657 p.

5. Matlab for Beginners / *Peter Kattan* // PETRA BOOKS, 2022. 474 p.

6. Distribution System Modeling and Analysis with MATLAB and WindMil / *William H. Kersting, Robert Kerestes* // CRC Press, 2022. 496 p.

7. Fundamentals of Computational Intelligence / *O. Zakovorotniy, O. Lipchanska* // Laboratory workshop. Part 1. Kharkiv: NTU "KhPI", 2022. 160 p.

8. Fundamentals of Computational Intelligence / *O. Zakovorotniy, O. Lipchanska* // Laboratory workshop. Part 2. Kharkiv: NTU "KhPI", 2022. 152 p.

9. Visible Light Communication. Comprehensive Theory and Applications with MATLAB / *Suseela Vappangi, Vakamulla Venkata Mani, Mathini Sellathurai* // CRC Press, 2021. 502 p.

10. Programming and Engineering Computing with MATLAB 2021 / *Huei-Huang Lee* // SDC Publications, 2021. 532 p.

11. Condition Monitoring Algorithms in MATLAB / *Adam Jablonski* // Springer International Publishing, 2021. 527 p.

12. Multiphysics Modeling Using COMSOL 5 and MATLAB / *Roger W. Pryor* // Mercury Learning and Information, 2021. 626 p.

13. Fluid mechanics. Problem solving using MATLAB / *Raju, K. Srinivasa, Kumar, D. Nagesh* // PHI Learning Pvt. Ltd., 2020. 368 p.

14. Programming and Engineering Computing with MATLAB 2020 / *Huei-Huang Lee* // SDC Publications, 2020. 532 p.

15. Basics of MATLAB Programming / *R. Balaji* // Notion Press, 2020. 390 p.

16. MATLAB Fast Automation. Automate Your Work With MATLAB / *Jacob Sapir* // Amazon Digital Services LLC – Kdp, 2020. 85 p.

17. MATLAB Programming. Mathematical Problem Solutions / *Dingyü Xue* // De Gruyter,·2020. 318 p.

18. Differential Equation Solutions with MATLAB / *Dingyü Xue* // De Gruyter, 2020. 451 p.

ЗМІСТ

Навчальне видання

ЗАКОВОРОТНИЙ Олександр Юрійович
ОРЛОВА Тетяна Олександрівна
ГРИНЬОВ Денис Валерійович

КОМП’ЮТЕРНА МАТЕМАТИКА

Навчальний посібник
для студентів денної та заочної форм навчання зі спеціальності
123 «Комп’ютерна інженерія»

Роботу до видання рекомендував проф. В. Д. Дмитрієнко
Відповідальний за випуск проф. С. Ю. Леонов
В авторській редакції

План 2024 р. , п. 18

Підп. до друку 15.02.2024. Гарнітура Times New Roman. Обл.-вид. арк. 14,6.

Видавничий центр НТУ «ХПІ»
Свідоцтво про державну реєстрацію ДК № 5478 від 21.08.2017 р.
61002, Харків, вул. Кирпичова, 2.

Електронне видання