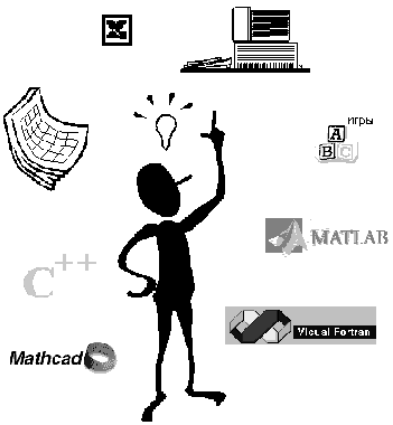
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний авіаційний університет

Гаєв Є.О., Нестеренко Б.М.

Універсальний математичний пакет MATLAB і типові задачі обчислювальної математики

Навчальний посібник



Київ 2004

УДК 004.94 (075.8)
ББК *з*973.23 я7
Г 134

Рецензенти: провідний наук. співр. Інституту механіки НАН України, д-р фіз.-мат. наук, проф. В.Ф. Мейш;
доцент кафедри механіки суцільних середовищ Київського Національного університету ім. Т.Г.Шевченка, канд. фіз.-мат. наук В.А. Каліон

Затверджено на засіданні кафедри комп'ютерізованих систем управління НАУ 20 січня 2003 року.

Гаєв Є.О., Нестеренко Б.М.
Г 134 Універсальний математичний пакет MATLAB і типові задачі обчислювальної математики. Навчальний посібник.– К.: НАУ, 2004. – 176 с.

Посібник спрямований на прискорене оволодіння універсальним математичним середовищем MATLAB і застосуванню його до розв'язування деяких типових задач чисельного моделювання. Коротко і наочно нагадано властивості цих задач і чисельні методи для них. Запропоноване "ручне" розв'язування за принципом "роби як я" дозволяє засвоїти алгоритмічну сторону методів. Ці ж задачі використано для роз'яснення основ програмування в MATLAB. Нарешті, викладено спеціалізовані команди MATLAB, які слід використовувати для систематичної роботи з даними задачами. Наведено як теоретичний матеріал, так і приклади розв'язування. Лабораторні завдання студентам для закріплення практичних навичок видаються окремо.

Для студентів старших курсів, які навчаються за спеціальністю „Системи управління та автоматики", а також для студентів інших спеціальностей, які вивчають та використовують сучасні інформаційні технології та математичне моделювання систем і процессів.

УДК 004.94 (075.8)
ББК *з*973.23 я7

ISBN 966-598-193-5

З М І С Т

Розділи, які помічено знаком , під час першого читання можна опустити

Вступ

Сьогоднішня доба в інформатиці така, що швидкісні і потужні ЕОМ (переважно архітектури IBM) стають дедалі доступнішими, багато хто зі студентів вже зараз має їх у персональному корисуванні. А з точки зору програмного забезпечення – більшість з майбутніх фахівців вже не будуть працювати з якоюсь окремою мовою програмування *Fortran*, *Pascal* або *C*++. На зміну останнім приходять інтегровані програмні продукти ("пакети") з так званим "дружнім інтерфейсом". Дизайнерська ідеологія розраховувати машину (праску, холодильник, пральну машину тощо), як кажуть, "*на чайника*" торкнулася вже галузі "розумних" машин: згадані програмні продукти дозволяють розв'язувати математичні задачі навіть людям, які в цих задачах не розуміються… Так би мовити "Не треба думати, програма підкаже!"

Відомо кілька таких інтегрованих пакетів – це MathCAD, Maple, MATLAB та інші. Одним з найвідоміших і найпоширених з них є універсальний математичний пакет MATLAB. В багатьох американських університетах, наприклад, вивчення MATLAB вже кілька років є обов'язковим для студентів першого – другого курсів. Університети України, інших країн СНД також почали включати інтегровані математичні пакети до навчальних програм, про що свідчить перелік щойно надрукованих посібників [19–29]. Кількість літератури з MATLAB, перелік якої подаємо окремо, також з кожним роком зростає [1–18].

Даний посібник відрізняється від названих книжок орієнтацією на програми вищої школи з інформаційних технологій та математичного моделювання систем і процесів.

"Нудні" математичні теми можна зробити цікавими, якщо пов′язати їх з комп′ютерними захопленнями сучасної молоді. Універсальний математичний пакет MATLAB такій вимозі відповідає.

Оволодіти MATLAB доволі складно. Даний посібник починає з початкового рівня (розділ 1) і доводить студента до достатньо складних задач, пов'язаних із звичайними диференціальними рівняннями. Для швидкого навчання

використано не формально-академічний підхід (алфавіт, операції, синтаксис і т.д.), а численні приклади і метод "*роби як я*".

Уявлення про перелік тем даного посібника дає *Зміст*. Матеріалом для роботи з MATLAB обрані деякі типові задачі математичного моделювання, а саме: розв'язання систем лінійних алгебраічних рівнянь і трансцендентних рівнянь, інтегрування функцій і розв'язання звичайних диференціальних рівнянь. Як відомо, вони часто зустрічаються в математичному моделюванні систем і процесів і тому включені до навчальних програм з багатьох дисциплін, які вивчаються у вищій школі. Для оволодіння таким програмним матеріалом вважаємо за доцільне їх "ручне" розв'язування; цьому присвячено розділ 2. Це дозволяє засвоїти основи того чи іншого математичного методу, зосередитись на умовах правильної, коректної постановки задачі, а також на алгоритмічній стороні обчислень, не відволікаючись на програмування. На даному етапі студенти використовують середовище MATLAB як швидкий калькулятор, як свого часу в курсах вищої математики використовували "кишенькові" програмовані мікро-ЕОМ *Б3-34*, *МК-52*, *МК-54*. У викладенні основ того чи іншого математичного методу ми спираємось на його наочне представлення, нагадуємо лише головні ідеї і властивості, які завжди потрібно пам'ятати студенту, інженеру і досліднику з посиланням на посібники, де можна знайти подробиці і обгрунтування.

У розділі 3 студенти повертаються до тих самих задач, але вже з метою їхнього програмування найпростішими засобами MATLAB. Якщо таке не передбачено програмою того чи іншого курсу – цей розділ можна пропустити. В курсі, де передбачається більш детальне вивчення програмування, або вивчення саме MATLAB, цей розділ слід використати. Таким є, наприклад, курс "*Інформаційні технології в комп'ютеризованих системах управління*". Також можливо, що студенти захочуть самостійно оволодіти цим простим матеріалом.

На кожну з типових математичних задач, які подані у *Змісті* посібника, в MATLAB є одна або декілька команд, які "автоматично" розв'язують відповідну задачу. Такі команди разом із прикладами використання наведені в розділі 4. Безумовно, саме "готові" засоби atLab слід застосовувати в практичній роботі – на

етапі дипломної роботи студента, в його наступній інженерній або науковій праці. Проте, тут слід розуміти природу конкретної задачі, умови коректності її формулювання для забезпечення наявності розв'язку, його єдиності, а також гарантування правильності обчислень. Всі ці методологічні питання спеціально обговорюються в розділі 4.

Даний посібник може бути корисний для студентів Національного авіаційного університету майже всіх спеціальностей, які вивчають різноманітні математичні курси. Посібник може бути також використаний фахівцями різних профілей, які бажають оволодіти MATLAB для використання у власній науковій роботі.

Даний посібник включає теоретичний матеріал. Посібник з лабораторних робіт видається окремо [3]. Це дозволяє використати різні методики навчання. Можна йти традиційним шляхом, від теорії (даний посібник) до вправ (посібник [3]). Але у багатьох випадках вивчення теми можна починати саме з лабораторної роботи, вміщеної в роботі [3], намагаючись самостійно знайти відповіді на поставлені питання. У такий спосіб слід спиратись на евристичний (дружній) інтерфейс MATLAB і на його розвинену довідкову систему (останнє ще дає додаткову практику в англійській мові…). Після цього теоретична частина – це перевірка Вашої роботи, відповіді на запитання, які залишилися.

Даний посібник разом із збірником лабораторних робіт [3] дозволяє оволодіти MATLAB у такому степені, що користувач зможе самостійно робити складні, розгалужені обчислення для власної навчальної або навіть професійної роботи.

У той же час слід зауважити, що посібник може вмістити лише основи роботи в середовищі MATLAB, і далеко не в повній мірі вичерпує можливості останнього. Тут ми обмежилися так би мовити одновимірною математикою. Подальшому викладенню MATLAB, його застосуванню до двовимірних математичних задач, плануємо присвятити наступні роботи.

1. АЗБУКА MATLAB

Комп'ютерна програма MATLAB* (назва походить від *Matrix Laboratory*, "Матрична лабораторія") – розвинена і універсальна система для наукових і інженерних розрахунків. Вона популярна у багатьох університетах і науково-виробничих фірмах світу. Система "розуміє" програми на мовах Fortran, C, C++, пов'язана з відомими текстовим редактором Microsoft Word і з табличним процесором Excel, що робить її зручною для виконання складних розрахунків (зокрема – розрахунків навчального характеру) та оформлення звітів про них, курсових та дипломних робіт. За досить тривалий час її розповсюдження (з 1984 р.) пакет MATLAB здобув собі стійку популярність. Ось чому обрано саме цю програму з кількох відомих.

Існують декілька модифікацій (версій) системи MATLAB. Найбільш відомими є версії серії MATLAB-5.x, для яких видані посібники [4-11,15,16]. В даному посібнику, однак, обрано більш сучасну на сьогоднішній день версію MATLAB2000 [2,12]. Її називають також MATLAB-6; відомі версії 6.0 і 6.5 У даному розділі пропонуємо оволодіти основами MATLAB крок за кроком шляхом певних вправ, від зовсім простих до складних за методом "*роби як я*". Наприкінці розділу студент придбає досить широкий круг навичок роботи в MATLAB, після чого можна перейти до розв'язування навчальних задач курсу вищої і прикладної математики для інженерних спеціальностей (розділ 2), або до основ програмування (розділ 3) та безпосередньо до арсеналу "готових" програм MATLAB (розділ 4).

1.1. ІНТЕРФЕЙС ПРОГРАМИ MATLAB

Як "театр починається з вішалки", так сучасна комп'ютерна програма починається і надає користувачеві всі свої можливості

*) Дехто називає цю програму "пакет" або "середовище". Ми використовуємо як синоніми такі терміни: "пакет", "середовище", "система".

через **інтерфейс** – графічну підпрограму для спілкування користувача з програмою основною.

Студентам з певним комп‘ютерним досвідом і з розвиненим почуттям “*Я сам*!” радимо розпочати з самостійного виконання лабораторної роботи 1 методичного посібника [3]. Дружній інтерфейс програми MATLAB і її *Help* підкажуть відповіді на запитання, запропоновані наведеним планом. Саме так часто роблять студенти, розбираючись з новою програмою. Перевірити вашу відповідь та знайти розв'язання нез'ясованим питанням зможете у наступному підрозділі.

Виконання роботи може бути також задане викладачем.

1.2. КОРОТКІ ВІДОМОСТІ ПРО КОМП’ЮТЕРНИЙ ПАКЕТ MATLAB

Після запуску MATLAB, користувач бачить на екрані комп’ютера вікно, що виглядає приблизно так, як зображено на рис. 1.1. Проте, вікно – саме воно разом із "кнопками меню" і "іконками" і утворює *інтерфейс програми* – може виглядати і трішки інакше. Все залежить від того, які опції користувач обрав з меню програми

Viev \ Desktop Layout ‣

Можливі різні варіанти того, як буде виглядати інтерфейс системи. Такими варіантами є: *Default* (без втручання користувача), *Command Window only* (лише вікно команд), *Simple* (найпростіший), *Short History* (вікно з історією команд користувача, скорочений варіант), *Tall History* (історія команд, повний варіант) та *Five Pannels* (усі можливі п’ять панелей з інформацією про ресурси MATLAB). Для наших цілей достатньо мати справу лише з *Command Window* – вікном команд, показаним на рис. 1.1. На тому ж рисунку показано ще вікно “Скорочена історія” роботи користувача (*Short History Window*). Така конфігурація вікон відповідає опції *Simple* (найпростіша).

У вікні праворуч, яке називається командним, бачимо так зване “*запрошення*” – пару символів “>>”, яка показує, куди треба вводити команди. На рис. 1.1 наведено кілька прикладів команд

MATLAB-середовища. Приклад 1 і 2 – це обчислення складного математичного виразу. Інформацію про це написано латинськими буквами після знака "**%**". Якби знака "%" не було, після натискання клавіши *Ввод* (*Enter*) система "облаяла" б користувача: "*Error: Missing operator, comma, or semicolon*" ("*Помилка: відсутній оператор, кома або кома з крапкою*"). Таким чином, знак "%" ставлять для того, аби система не помічала наступного тексту. Такий прийом використовують для написання *коментарів* до складної задачі. (Радимо писати коментарі латинськими, або великими українськими літерами – пояснення див. в розділі 3.3).

Приклади 1 – 5 (див. рис. 1.1) знайомлять із способами простих обчислень в системі MATLAB, а також із *алфавітом* і *синтаксисом* системи. Останні наближені до звичної нам форми математичних записів. Розглянемо кілька прикладів.

Приклад 1.1. Обчислити вираз

$$\frac{\sqrt{2+\sqrt{3}}}{\sqrt{2-\sqrt{3}}}+\frac{\sqrt{2-\sqrt{3}}}{\sqrt{2+\sqrt{3}}}.$$

У командному вікні після ">>" набираємо на спеціальній мові, яка наближена до звичайної, але зрозуміла MATLAB:

sqrt(*2*+*sqrt*(*3*))/*sqrt*(*2*-*sqrt*(*3*))+*sqrt*(*2*-*sqrt*(*3*))/*sqrt*(*2*+*sqrt*(*3*))

Натискуємо *Ввод*, і система дає відповідь: *ans =4* (або *4.0000*, залежно від замовленої точності).*)

Можна зробити і інакше, вводячи кілька команд у рядочку:

x=*2*;*y*=*3*;*sqrt*(*x*+*sqrt*(*y*))/*sqrt*(*x*-*sqrt*(*y*))+*sqrt*(*x*-*sqrt*(*y*))/*sqrt*(*x*+*sqrt*(*y*))

У такому випадку програма надає *x* значення *2* і *y* значення *3* і підставляє їх у наступний вираз. Натискуємо *Ввод* і система дає таку саме відповідь: *ans =4*.

*) Надалі під записом *Ввод* будемо розуміти дію – натискання клавіші *Ввод*.

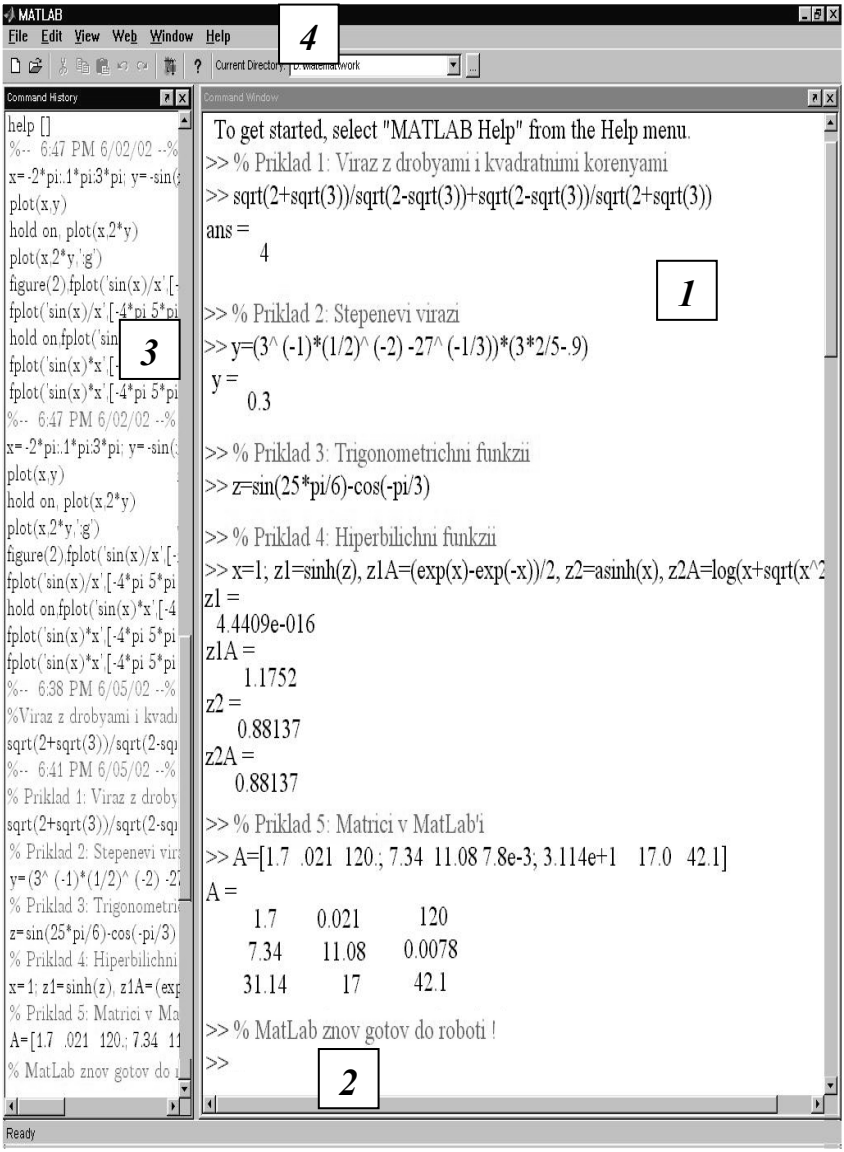


Рис. 1.1. Вигляд програми MatLab у конфігурації Simple: 1 – Вікно команд (розмір можна змінювати); 2 – "Запрошення" до вводу команд; 3 – Вікно "Коротка історія" (розмір можна змінювати); 4 – Меню з багатьма можливостями

Приклад 1.2. Обчислити

$$(3^{-1}(\frac{1}{2})^{-2} - 27^{-\frac{1}{3}})(3 \cdot \frac{2}{5} - 0.9).$$

У командному вікні набираємо:

y=(3^ (-1)(1/2)^ (-2) -27^ (-1/3))*(3*2/5 - .9).*

Після *Ввод* маємо відповідь: *y =0.3000.*

Увага: між двома множниками (виразами в дужках) ставити знак множення * обов′язково!

Приклад 1.3. Обчислити значення функції:

$$\sin(\frac{25\pi}{6}) - \cos(-\frac{\pi}{3}).$$

На мові MATLAB така формула виглядає як

*z=sin(25*pi/6) -- cos(-pi/3).*

Отримуємо результат *z = 4.4409e-016*. Практично, це *0*, але приклад 1.3 показує також, з якою точністю MATLAB обчислював функції *sin* і *cos*.

У прикладах 1.2 і 1.3 отримано величини заданих змінних *y* і *z*, а у прикладі 1.1 система дала назву самостійно – *ans* (від *answer*, відповідь). У подальшому система вже знає, що величини *y*, *z* і *ans* мають певні значення, так що на введення *y^2+z^2* або *sqrt(ans)* система видасть значення відповідно *0.09* та *2*. Тобто, запис із знаком рівняння " = " дозволяє надати змінній потрібне числове значення.

MATLAB виконав ці обчислення оскільки відповідні завдання (*оператори*) записані без синтаксичних помилок. Наприклад, кількість відкриваючих дужок обов′язково дорівнює кількості закриваючих. Спробуйте зробити помилки навмисно – і система відмовиться обчислювати, відразу вкаже на помилку: *Error*.

Цей перший досвід дозволяє заключити: в MATLAB імена змінних записуються буквами латинського алфавіту і можуть бути довільними. Лише певні імена, зокрема

ans, *pi*, *i*, *j*, *inf*, *NaN*,

зарезервовано для внутрішних потреб MATLAB: *pi*=π≈3,141592 (число π), $i = j = \sqrt{-1}$ (уявна одиниця), *inf*=∞ (infinity, нескінченність) і невизначеності $NaN = \frac{0}{0}$ або $NaN = \frac{\infty}{\infty}$ ("*nane*", not a number). Користувач не має права надавати такі імена власним змінним.

Запис математичних дій над величинами здійснюється за допомогою символів, вміщених в табл. 1.1.

Таблиця 1.1. Позначення математичних операцій в MATLAB

+	Додавання	/	Ділення
–	Віднімання	\	Ділення "справа наліво"
*	Множення	^	Піднесення до степеня
.*	Поелементне множення	./	Поелементне ділення

Приклад 1.3 показує, що імена *sin*( ), *cos*( ), *exp*( ), *log*( ), *log10*() стають іменами функцій (відповідно – синуса, косинуса, експоненти $e^{(\,)}$, логарифма натурального ln і логарифма десяткового lg), коли вони у дужках заключають імена змінних або математичні вирази. Імена інших функцій, відомих MATLAB, дещо відрізняються від звичних у вітчизняній літературі. Найголовніші з них наведені в табл. 1.2.

Таким чином, відомі математичні співвідношення

$$\operatorname{sh} x = \tfrac{1}{2}(e^{x} - e^{-x}) \qquad \text{та} \qquad \operatorname{arcsh} x = \ln(x + \sqrt{x^2 + 1})$$

на мові MATLAB будуть виглядати як

sinh(x)=(exp(x)-exp(-x))/2 та *asinh(x)=log(x+sqrt(x^2+1)).*

Приклад 1.4 на рис. 1.1 наводить обчисленняння цих функцій для *x=1*.

Увага: в MATLAB введено кілька власних математичних операцій – ділення "справа наліво" \ і поелементні множення .* і

ділення **.** /. Для чисел ділення *a/b* означає, як звичайно, $\frac{a}{b}$; а "*ділення справа наліво*" a\b означає $\frac{b}{a}$. Проте для матриць різниця між двома операціями більш суттєва – див. про це в розділі 4.1. Поелементні операції (операції з крапкою) **.*** і **./**, а також **.^** застосовуються лише до векторів і матриць однакового виміру – див. про них в розділах 1.3 і 1.5.

Таблиця 1.2. Основні математичні функції MATLAB

MATLAB	**Функція**	MATLAB	**Функція**
sin	Синус	*tan*	Тангенс
sinh	Синус гіперболічний	*tanh*	Тангенс гіперболічний
asin	Арксинус	*cot*	Котангенс
asinh	Арксинус гіперболічний	*acot*	Арккотангенс
cos	Косинус	*sqrt*	Корінь $\sqrt{\ }$
cosh	Косинус гіперболічний	*acoth*	Арккотангенс гіперболічний
acos	Арккосинус	*exp*	Експонента
acosh	Арккосинус гіперболічний	*log, log10*	Логарифми
abs	Абсолютна величина (модуль)	*real, imag*	Дійсна та уявна частини комплексного числа

Наступне правило MATLAB стосується запису чисел. Числа в звичній для нас формі

2,17 0,00217 $0{,}217\ 10^{1}$

(десяткова частина відокремлюється від цілої комою) для MATLAB треба писати як

2.17 0.00217 .217e+1,

або ще так: .217e+1 або .217e-2. Скільки значущих цифр буде видавати MATLAB у своїх відповідях – залежить від обраної користувачем форми (*short* або *long*, без суфіксів, або з суфіксами E, D), вказаної в пункті меню

File\Preferences… \Array Editor.

Команда MATLAB може складатись з кількох операторів. Тоді їх треба відокремлювати один від одного за допомогою коми "**,**" або крапки з комою "**;**". У першому випадку після *Ввод* отримаємо значення всіх змінних, що обчислювалися. У другому випадку обчислення будуть проведені, але надруковані не будуть. Це слід використовувати, аби виводити важливі дані (кінцеві, вузлові або відладочні результати обчислень) і не виводити другорядні.

Отже, можна починати практичну роботу в середовищі MATLAB.

1.3. ПРИЙОМИ РОБОТИ З MATLAB. ДОПОМОГА ВІД MATLAB

Ознайомимось ще з вікном MATLAB "Коротка історя команд" (ліворуч на рис. 1.1). У цьому вікні фіксуються час і дата даної і попередніх сесій роботи і всі команди, які робив користувач у даній сесії роботи (після запуску MATLAB), і навіть у попередніх сесіях. (Щоб очистити "Історію", треба правою кнопкою миші клацнути в довільному місці вікна і в меню, яке з′явиться, обрати "*Delete Entire History*").

На рис. 1.1 у вікні "*Command History*" видно лише початок команд, що виконувалися. Щоб побачити більше, треба курсор миші навести на смугу, що розділяє командне вікно і вікно "*Історія*", і коли курсор перетвориться у стрілочку "↔" натиснути ліву кнопку миші і "потягнути" смугу вправо. Так само можна, навпаки, розширити командне вікно.

Деякі спеціальні прийоми можуть бути корисними під час роботи з MATLAB. Часто треба повторити команду, яка виконувалася раніше. Для цього існує кілька способів:

1. Якщо треба повторити щойно виконану команду – не треба знов набирати її з клавіатури. Натисніть клавішу "стрілка

уверх”[↑]-- і остання команда з′явиться у командному рядочку знову. Натисніть “стрілку уверх” ще раз – з′явиться передостання команда. Такий прийом зручний для двох - п′яти останніх команд.

2. Якщо команда виконувалася давно в дану сесію, або навіть в попередню – знайдіть її у командному вікні (можливо, для цього потрібно піднятися по командному вікну вище: лівою[*)] кнопкою миші “натиснути” ковзунок на правій смузі вікна і “потягнути” його уверх). Виділіть потрібну команду або потрібну її частину (лівою кнопкою миші “протягніть” від початку до кінця потрібного тексту). Тепер треба скопіювати виділене у допоміжну пам′ять (спосіб 1: клацнути на виділеному правою кнопкою миші і з меню обрати *Copy*; спосіб 2: натиснути комбінацію клавіш < *Ctrl+C* >). Нарешті, повернутися до чергового Запрошення >> і вставити потрібний текст (права кнопка миші і обрати *Paste*, або натиснути комбінацію клавіш < *Ctrl+V* >). Можна і так зробити: курсор миші навести на виділений фрагмент, натиснути ліву кнопку миші, “потягнути текст” в кінець командного вікна до запрошення “>>” і відпустити.
3. Якщо потрібна команда знаходиться у вікні “*Command History*”, клацніть по ній лівою кнопкою миші і в такий спосіб виділіть її. Тепер натисніть ліву кнопку і “перетягніть” команду у командне вікно.

В короткому посібнику розповісти про MATLAB все – просто не можливо. Користуйтеся книжками [1-18], список яких наведено наприкінці. Окрім того, суттєву допомогу можна отримати від самого MATLAB прямо біля комп′ютера – так званий *Help*. Ось деякі способи.

Натискання на клавішу [*Help*] (праворуч у верхньому рядочку меню 4, рис. 1.1) і потім вибір *Using the Desktop* або *Using the Command Window* з меню, яке з′являється, призводить до появи кольорового вікна Help[**)] з поясненнями англійською мовою щодо,

[*)] Ролі лівої і правої кнопок миші такі самі, як, взагалі, в операційній системі Windows.

[**)] Або через головне меню *View \Help*

відповідно, загального вигляду програми MATLAB та, окремо, вікна команд. (В версії MATLAB 6.5 це зроблено трішки інакше).

Натискання на клавішу із знаком запитання **?** (нижній рядочок меню 4 на рис. 1.1, праворуч), або ж, через меню, *Help* \MATLAB Help призводить до появи того ж вікна *Help*, де можна розшукувати потрібну інформацію. Вікно складається з двох половин – панелі *Help*-навігатора (ліворуч) і панелі інформації, рис.1.2. Навігатор подає зміст довідкової системи. Як тільки курсором виділено потрібний розділ – інформація з нього відразу з′являється на правій панелі.

Для початкового ознайомлення доречно запустити демонстраційний "мультфільм": з меню обрати Help\Demos. Далі обрати одну з тем демонстраційного показу: Desktop Environment (інтерфейс програми MATLAB), Matrices (матриці), Numerics (деякі чисельні методи – підбір емпірічної функції, інтерполяція, диференціальні рівняння та інші), Graphics (побудова графіків), Language (мова програмування в MATLAB), Gallery (деякі "топологічні образи") або More Examples (графічні ілюстрації до деяких математичних задач). Письмові коментарі до "мультиків" – англійською мовою.

Нарешті, ще один спосіб отримати інформацію. Якщо, припустимо, ви знаєте, яку саме команду або функцію MATLAB бажаєте використати, але забули її формат – тоді просто з командного рядку даєте наказ:

help ІмʹяКоманди,

наприклад: *help plot* або *help log*. Отримаєте текст з підказкою відповідно про побудову графіків та логарифмічну функцію.

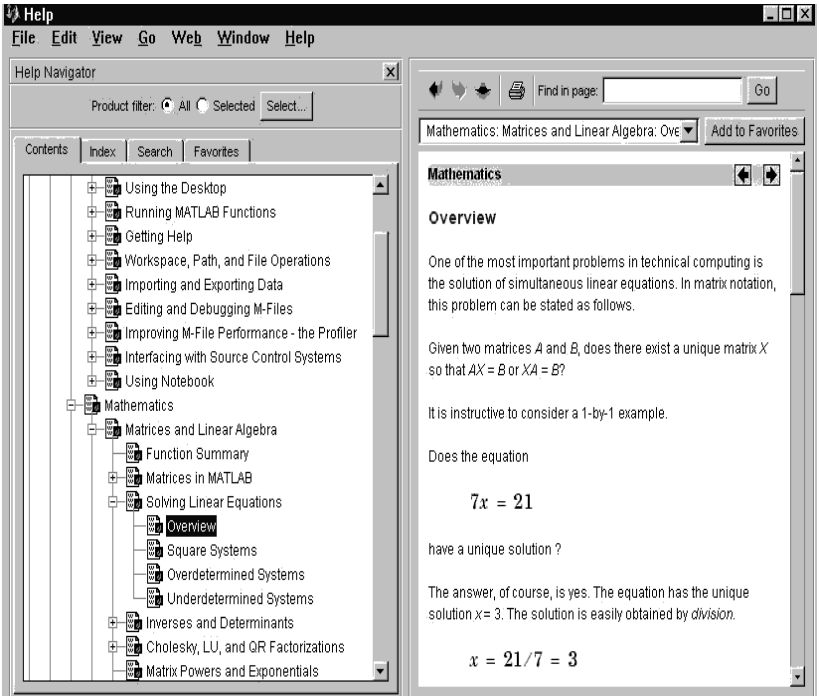


Рис. 1.2. Зовнішній вигляд вікна допомоги (Help). Воно відкрито на інформації про розвязок лінійних алгебраїчних рівнянь: панель навігатора (ліворуч) вказує зміст довідника, панель праворуч надає інформацію

Допоміжні матеріали, які для вас мають якесь особливе значення (плануєте використовувати найчастіше або детально вивчити пізніше), можуть бути відібрані у вашу особисту колекцію. Натисніть на кнопку *Add to Favorits* (Додати у Вибране) праворуч зверху (рис. 1.2), і надалі відповідна сторінка *Help* постійно зберігатиметься в розділі *Favorits* (Вибране) *Help*-навігатора (рис.1.2, ліва панель).

1.4. МАТРИЧНА АРИФМЕТИКА MATLAB

Матрична арифметика – основа MATLAB (пригадайте походження його назви). Тому наступний матеріал вартий особливої уваги!

Для запису числових матриць в MATLAB всі елементи матриці заключають у квадратні дужки, елементи одного рядка відокремлюють один від одного *пропуском* або *комою*, а рядок від рядка – *крапкою з комою* "**;**". Тоді, матрицю

$$A=\begin{pmatrix} 1.7 & 0.021 & 120 \\ 7.34 & 11.08 & 0.0078 \\ 31.14 & 17 & 42.1 \end{pmatrix}$$

можна записати у командному вікні MATLAB як

A=[*1.7 .021 120.* **;** *7.34 11.08 7.8e-3* **;** *3.114e+1 17.0 42.1*] (1.1)

(див. приклад 5 на рис. 1.1). Тоді натискання *Ввод* призведе до того, що матрицю буде переписано у звичній для нас формі у встановленому форматі для чисел (чотири знаки після десяткової точки у даному випадку).

Тепер зрозуміло, як передати системі вектор-рядок або вектор-стовпчик, припустимо, такі:

$$row1 = (1.7,\ 0.021,\ 120)\ , \qquad col1 = \begin{pmatrix} 1.7 \\ 7.34 \\ 31.14 \end{pmatrix}.$$

У командному вікні MATLAB пишемо відповідно:

row1=[*1.7 .021 120.*], *col1*=[*1.7; 7.34; 31.14*] (1.2)

Натискання клавіші *Ввод* дозволяє перевірити, чи вірно зрозумів вас MATLAB.

Система має також особливі імена для деяких спеціальних матриць. А саме:

zeros(m,n) – матриця вимірності *m*x*n* з самих нулів (нуль-матриця),

ones(*m,n*) – матриця вимірності *m*x*n* з самих одиниць,

eye(*n*) – матриця вимірності *n* з одиницями по головній діагоналі, усі інші елементи – нулі (одинична матриця).

Пояснимо це конкретними прикладами:

$$zeros(2,3)=\begin{pmatrix}0 & 0 & 0\\ 0 & 0 & 0\end{pmatrix},\quad ones(3,2)=\begin{pmatrix}1 & 1\\ 1 & 1\\ 1 & 1\end{pmatrix},$$

$$eye(4)=\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}.$$

Якщо вимірність матриці дуже велика – одного командного рядка може не вистачити. Щоб продовжити введення даних (або набір довгої команди) – наберіть знак продовження **...** (три крапки) і натисніть *Ввод*. Ці три крапки будуть сприйняті як особливий, службовий знак і помічені синім кольором. На новому рядку запрошення “>>” вже не буде. Продовжуємо на ньому введення інформації, і якщо треба – знов переходимо на наступний рядок за допомогою знака продовження **...** (трьох крапок). Коли ж, нарешті, введення закінчено, остаточно натискуємо клавішу *Ввод*. На новому рядку з'явиться знак запрошення “>>”, що свідчить про те, що MATLAB готовий до сприйняття наступної команди.

Проте для великих за розмірами матриць, які незручно набирати у вигляді (1.1), або навіть лише їхні рядки або стовпчики у вигляді (1.2), існують і інші засоби. MATLAB дозволяє вводити їх частинами. Припустимо, для рядків і стовпчиків (1.2) ввели спочатку

row1a=[*1.7 .021*], *row1b*=[*120.*], *col1a*=[*1.7*], *col1b*=[*7.34; 31.14*].

Тоді, виконанням команд

row1=[*row1a, row1b*], *i* *col1*=[*col1a; col1b*]

передаємо у пам′ять комп′ютера вектори *row1* і *col1* у відповідності саме із виразом (1.2).

Аналогічно, якщо за такими правилами ввести другий *row2* і третій *row3* рядки матриці (1.1) (або другий *col2* і третій *col3* стовпчики з виразу (1.1)), то сформувати матрицю *A* в цілому можна за допомогою команди

A=[*row1; row2; row3*] *або* *A*=[*col1, col2, col3*]

Звернемо увагу ще раз: знак “ **;** “ (крапка з комою) відокремлює кожний наступний рядок, знак “**,**” (або ніякого знака, пропуск) розділяє стовпчики!

Визначений вище *ідентифікатор A* буде для програми не якимось числом, а саме набором чисел, таблицею. На відміну від цього, запис $A(i\ ,\ j)$ посилається вже на конкретне число з набору, яке знаходиться на перетині рядка i і стовпчика j, за умов $i \le n$ і $j \le m$. Наприклад, $A(1,2)=0.025$, $A(2,3)=0.0078$. Важливо розуміти і більш складні посилання, наприклад – на частину певного рядка або стовпчика. Якщо “замовляємо” MATLAB-системі *A*(*1:2,2:3*) – тобто, "надати підтаблицю з рядками від *1* до *2* і із стовпчиками від *2* до *3*", то це буде матриця

$$\begin{pmatrix} 0.021 & 120 \\ 11.08 & 0.0078 \end{pmatrix}.$$

Ось ще приклади посилань на частини введеної матриці.

r1=A(:,*1*) – перший стовпчик матриці *A*, тобто “рядки – усі, а із стовпчиків обирати лише з індексом *1*”. *A1=A*(*2*, *2* **:** *end*) означає “елементи другого рядка, в яких другий індекс змінюється від 2 до кінця”. *A*(:,:), або *A*(*1* : *end*, *1* : *end*) дає саму матрицю *A*. Таким чином,

$$r1 = \begin{pmatrix} 1.7 \\ 7.34 \\ 31.14 \end{pmatrix}, \qquad A1 = (11.08 \quad 0.0078).$$

MATLAB вміє робити складні перетворення матриць – їхнє додавання, віднімання, множення, і робить це у відповідності з загальними правилами матричної алгебри [13,17,33]. Додавати і віднімати вектори і матриці ($A+B$ і $A-B$) можна, якщо їхні

виміри співпадають. Інакше MATLAB видасть повідомлення про помилку:

Error using ==> +
Matrix dimensions must agree.

При додаванні (відніманні) матриць названа операція здійснюється з кожною відповідною парою елементів матриць A і B, і з цих сум (різниць) утворюється нова матриця $S = A + B$ або $R = A - B$.

Результат множення векторів або матриць $A * B$ вимірності $i \times n$ і $n \times j$ є матриця вимірності $i \times j$, яка обчислюється за складним правилом. MATLAB його знає, а студент може пригадати з підручників [33].

Операція ділення A / B ні для векторів, ні для матриць у підручниках з математики не визначена. Проте, MATLAB-система вводить таку операцію, розуміючи під нею множення $A * B^{-1}$. Така операція визначена, коли B – квадратна невироджена матриця $n \times n$, а A -- матриця з довільним числом рядків і з n стовпчиками. Аналогічно визначається в MATLAB *операція ділення* "*справа наліво*": $A \backslash B = A^{-1} * B$. Припускається, що A – квадратна матриця $n \times n$, а B має вимірність $n \times k$, наприклад – вектор-стовпчик з n рядків. В такому випадку $A \backslash B$ є розв'язком системи лінійних рівнянь $Ax = b$, коли x і b є вектор-стовбці.

MATLAB вводить також "нелітературні" операції "*поелементне множення*" і "*поелементне ділення*" (операції з крапкою), коли матриці мають однакові виміри і операція застосовується до їхніх відповідних елементів, наприклад:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1k} \\ a_{21} & a_{22} & \dots & a_{2k} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nk} \end{pmatrix} .* \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1k} \\ b_{21} & b_{22} & \dots & b_{2k} \\ \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & b_{nk} \end{pmatrix} = \begin{pmatrix} a_{11} * b_{11} & a_{12} * b_{12} & \dots & a_{1k} * b_{1k} \\ a_{21} * b_{21} & a_{22} * b_{22} & \dots & a_{2k} * b_{2k} \\ \dots & \dots & \dots & \dots \\ a_{n1} * b_{n1} & a_{n2} * b_{n2} & \dots & a_{nk} * b_{nk} \end{pmatrix}.$$

Аналогічно, "*поелементне ділення*" двох матриць $n \times k$ – це нова матриця, яка позначається A **.** $/B$ і складається з відповідних пар a_{ij} / b_{ij}. Бачимо, що MATLAB сконструйовано саме для роботи з матрицями!

Дійсно, створити вектор (тобто матрицю $1\times n$) дуже просто. Для цього призначений оператор **:** (semicolon). Конструкція

$$\text{v}=3.5 : .15 : 5.0 ; \qquad (1.3)$$

створює вектор v з першим елементом *3.5*, другим елементом *3.65*, третім елементом *3.80*, і так далі з кроком *.15*, поки черговий елемент не наблизиться до *5*. Коли у виразі (1.3) присутній лише один семіколон, то крок обирається *1*, тобто вектор v*1=3.5* **:** *5.0* складається лише з елементів *3.5* і *4.5*.

Увага: в конструкціях (1.3) слід не забувати ставити крапку з комою **;** , інакше весь екран може "засипати" великою кількістю цифр!

Коли v вектор, то і змінні *x=sin*(v), *y=tan*(v), *z=exp*(v) стають векторами з елементами, що дорівнюють вказаним функціям від відповідних елементів вектора v. Наприклад, *z1=exp*(v1) є вектором з двох елементів *z1*=[$e^{3.5}$, $e^{4.5}$].

Проте, спроба в такий спосіб виконати команду

>> *z*= *sin*(v) / v або >> *z*=v**exp* (v)

призведе до повідомлення про помилку на зразок

??? *Error using* ==> *
Inner matrix dimensions must agree.

Чому не вдається? Адже ж ви замовили операції над векторами, а вони не для кожної пари операндів коректно визначені! Використайте ті ж самі операції, але "з крапкою" **.** / , **.*** – і отримаєте жаданий результат! Розуміння різниці між вказаними операціями буде до нагоди у розділі 1.5.

Комплексне число *z=x+**i**y* – це той самий вектор з двох елементів *z*={*x*,*y*}. Тому не дивно, що MATLAB вміє робити обчислення і з комплексними числами. При цьому останні можна записувати як *z=x+**i**y*, так і як *z=x+**j** y* (тобто уявну одиницю можна позначати як ***i***, так і ***j***). Легко перевірити комплексну арифметику:

```
>> (2+3j)+(3+2i)
   ans =   5.0000 + 5.0000 i
```

```
>> (2+3j)*(3+2j)
   ans =    0 +13.0000 i

>> sqrt(j)
   ans =   0.7071 + 0.7071 i
```

(у випадку комплексних чисел знак множення на уявну одиницю можна не ставити!)

1.5. ФУНКЦІЇ ВІД МАСИВІВ

Нагадаємо, що розділи посібника, відмічені знаком , під час першого читання можна пропустити. Зокрема, наведений тут матеріал буде вперше використаний в розділі 2.4, хоча "за логікою" продовжує підрозділ 1.4.

MATLAB орієнтований на систематичне використання масивів, зокрема – матриць. Це дозволяє розробляти більш загальні і короткі алгоритми, а також значно збільшити швидкість обчислення великих задач. Заради цього програми знаходження певних характеристик масивів, такі, як знаходження розмірів масиву або його найбільшого (найменшого) елементів тощо, включені в MATLAB як його “внутрішні” функції. Їх слід використовувати у наших обчисленнях.

Якщо треба знайти або використати **довжину вектора** (тобто кількість його елементів), то можна застосувати функцію *length*. Як і у випадку звичайної функції, за ім'ям функції у дужках вказують аргумент, у даному випадку – ім'я вектора, довжину якого шукаємо. Вище були визначені вектори *r1* і *A1*. Для них

$$length(r1)=3, \qquad length(A1)=2.$$

Таку функцію можна застосувати і до матриці. Отримаємо найбільшу з двох вимірностей матриці, наприклад

$$length(zeros(2,3))=3.$$

(Приклад використання див. в підрозділі 3.2.2).

Проте, відносно матриць часто треба знати її обидві вимірності, тобто вектор з двох чисел. Тому треба застосувати іншу функцію, *size*, у форматі

$$[M\ ,\ N] = size(A).$$

Виконання такої команди видає на екран величину змінних *M* і *N*, з яких перша, *M*, дає кількість рядків матриці *A*, а *N* – кількість стовпців. Деякі інші корисні формати застосування такої функції див. в довіднику *Help*, *help size*.

MATLAB вміє обчислювати визначники матриць і обернені матриці; для цього існують внутрішні MATLAB′івські функції

det(*A*) і *inv*(*A*).

Зауважимо ще раз, що MATLAB обов‘язково звертає увагу на відповідність вимірностей матриць, дії з якими йому запропоновані. Якщо він “залається”

Error using ==> +
Matrix dimensions must agree,

то це означає, що користувач запропонував йому некоректну математичну операцію, і MATLAB нагадує: “*Виміри матриць повинні погоджуватись*”. Якщо ж MATLAB дає попередження

Warning: Matrix is singular to working precision,

то це означає неможливість отримати обернену матрицю, оскільки її визначник дорівнює “ машиному нулю”, матриця вироджена.

Функції *max*(*A*) і *min*(*A*) у випадку, коли *A* є вектор, дають відповідно найбільший і найменший з його елементів. Так, *max*(*r1*)=*31.14*, *min*(*r1*)=*1.7*. Однак, ми не отримали положення знайденого елемента у векторі. Аби знайти ще й положення, існує і інший формат звертання до цих функцій: [*E*, *n*]. В такому випадку MATLAB визначить два числа – елемент вектора *E*, найбільший чи найменший, і також номер елемента у векторі *n*. Приклад: на команду [*E*, *n*]=*max*(*r1*) MATLAB відповість *E=31.14*, *n=3*. Результатом команди [*E*, *n*]=*min*(*r1*) буде *E=1.7*, *n=1*.

Доволі часто використовується функція *sum*(*A*). Коли *A* є вектором, то названа функція видає суму всіх його елементів. Якщо ж *A* є матрицею з *n* стовпчиків, то відповіддю буде вектор-рядок з *n* елементів, кожний дає суму елементів відповідного стовпчика *A*. Наприклад, якщо *A*=[*1 2 3*] або *A*=[*1*; *2*; *3*], то *S*=*sum*(*A*)=*6*; у випадку *A*=[*1 2 3* ; *4 5 6*] змінна *S*=*sum*(*A*) буде вектором-рядком *S* =[*5 7 9*].

Функція *норми вектора* $norm(\mathrm{v}, p)$ обчислюється для цілих $p=1$, $p=2$, $p=3$ за формулою

$$norm(\mathrm{v}, p) = \left(\sum_{i=1}^{n} | \mathrm{v}_i |^{p} \right)^{1/p}.$$

Очевидно, що $norm(\mathrm{v},1) = \sum | \mathrm{v}_i |$ – просто сума модулів компонент вектора, $norm(\mathrm{v},2) = \sum \mathrm{v}_i^2$ – сума квадратів. Такі функції стають у нагоді для розв'язування лінійних алгебраїчних рівнянь та для побудови ітераційних алгоритмів.

Ще одна внутрішня функція MATLAB може буди корисною. Фукція *diag(A)*, якщо вона застосовується до квадратної матриці *A* вимірності *n*x*n*, видає вектор-стовпчик вимірності *n*, який складається лише з елементів головної діагоналі *A*. Якщо ж функцію *diag*(v), застосувати до вектора v з *n* елементів, то отримаємо матрицю *n*x*n*, в якій всі елементи дорінюють нулю, окрім головної діагоналі, по якій розташовані елементи вектора v. Таким чином, матриця

$$D=diag(diag(A))$$

є квадратною матрицею тієї ж вимірності *n*x*n* що і матриця *A*, в якій елементи на головній діагоналі співпадають з відповідними елементами *A*, а усі останні дорівнюють нулю. Радимо перевірити роботу такої функції самостійно.

Повний перелік функцій над матрицями дивись у довіднику Help, наприклад – командою *help elmat* (Elementary matrices).

1.6. ПОБУДОВА ДВОВИМІРНИХ ГРАФІКІВ

Графічне зображення результатів розрахунків – найбільш інформативне і переконливе. Тому дуже важливо засвоїти хоча б частину тих можливостей, які надає MATLAB. Перш ніж вивчати наведену нижче теорію, пропонуємо спочатку спробувати будувати графіки самостійно шляхом виконання лабораторної роботи 2 за

наданим в посібнику [3] планом. Після цього – подивіться у теоретичну частину даного розділу.

Середовище MATLAB має кілька спеціальних команд побудови графіків. Графік одновимірної залежності – це є лінія на площині *x*O*y*, яка з′єднує певні точки P_i з координатами $\{x_i,y_i\}$ на площині. Це значить, що ми повинні вказати MATLAB таку систему точок. Після цього програма з′єднає їх за допомогою однієї із своїх команд, наприклад – *plot*. Покажемо це на прикладах.

Нехай потрібно побудувати графік функції *y=sin*(*x*) в діапазоні зміни аргументу $x \in [-\pi, 3\pi]$. Припустимо також, що нам достатньо мати 10 точок на цьому відрізку довжиною $l = 4\pi$, тобто крок між точками має бути $\Delta x = 0{,}4\pi$. У командному вікні MATLAB виконаємо таку послідовність команд:

*>> x =-pi : .4*pi : 3*pi , y = sin* (x)
>> plot(*x,y*)

Перша команда, як ми вже знаємо, утворює вектор чисел *x,* про що MATLAB, оскільки крапка з комою не поставлена, повідомить на зразок:

x=
Columns 1 through 6
3.1416 -1.885 -0.62832 0.62832 1.885 3.1416
Columns 7 through 11
4.3982 5.6549 6.9115 8.1681 9.4248.

Друга команда обчислить вектор з ім'ям *y* значень функції *sin*(*x*) від відповідних значень *x*. Ну а третя команда вже будує на площині *x*O*y* утворені в такий спосіб точки $\{x_i,y_i\}$ і з′єднує їх лінією. Результат див. на рис. 1.3: природньо, що графік доволі грубий, бо крок між точками осі O*x* є $0{,}4\pi \approx 1{,}3$. Графік буде виглядати краще, якщо всі наведені команди повторити, задавши в десять разів менший крок аргументу:

*>> x=-pi : .04*pi : 3*pi ; y=sin* (*x*) ; *plot*(*x,y*)

(дивись результат на рис. 1.4). **Увага:** тепер команди розділені між собою знаком **;** , аби не ускладнювати вигляд екрану виводом двічі

по 110 даних з масивів x і y. Останній графік наведено так, як він виглядає в MATLAB – разом із спеціальним вікном *Figure*.

Сказаного ще недостатньо для побудови графіка функції, скажімо, $y = \sin(x)/x$. Для утворення масиву значень функції y треба застосовивати не звичайне ділення, а "*ділення з точкою*" **./**. Тоді послідовність команд

```
>> x= - pi : .01*pi : 3*pi;  y=sin (x) ./ x ; plot(x,y)        (1.4)
```

дасть нам потрібний графік, побудований для області $x \in [-\pi, 3\pi]$.

А як побудувати два графіки на одному рисунку, припустимо, ще графік функції $y = x^{1/3}\sin(x)$? Спочатку потрібен масив чисел (назвемо його *z*)

```
z= sin (x) .* x  .^ (1/3) ,
```

(масив чисел *x* обчислювати не треба, бо він вже існує; по-друге, зверніть увагу на покомпонентне "*множення з точкою*" **.*** і покомпонентне "піднесення до *степеня*" **.^**, про які йшла мова в поясненнях до табл. 1.1.). Якщо тепер наказати MATLAB

```
>> z= (x) .^(1/3) .* sin (x); plot(x,z),
```

то побачимо новий графік замість старого в тому ж вікні *Figure 1* – така зміна теж інколи потрібна! А ось отримати одночасно два графіка (і так само три і більше) можна двома способами. В перший спосіб виконуємо команду *hold on* (включити *режим утримання*) перш ніж наказати *plot(x,z)*. (І навпаки, якщо потрібно у тому ж вікні зобразити нову криву замість попередніх, треба виключити режим утримання командою *hold off*). Другий спосіб дає команда

```
plot(x,y, x,z).
```

Тобто, в команді *plot* можливо замовляти кілька графіків – спочатку вказати перший масив пар точок (*x,y*), потім другий (*x,z*), третій і т.д. – скільки потрібно.

Дві лінії на графіку, однак, неможна відрізнити, бо вони "намальовані" лініями одного кольору. Попередню команду краще виконати у такому форматі:

plot(*x,y,'r', x,z,'g'*), *legend*(*'y=sin*(*x*)*/x', 'z=sin*(*x*) *x^{1/3}'*), **. . .**
title('*Дві криві на графіку*')

Тоді перша лінія на графіку буде червоною (від *red*), а наступна зеленою (від *green*). Команда *legend*, крім того, напише додаткове пояснення до кривих, а команда *title* дасть назву рисунку '*Дві криві на графіку*' (див. приклади нижче).

Отримані графіки можна зробити більш "красивими". Для покращення їхнього вигляду і збільшення інформативності можуть бути потрібними ще кілька команд. Наведемо їх:

grid – "включає" координатну сітку на рисунку;

xlabel('Довільна назва осі Ox'),
ylabel('Довільна назва осі Oy') – надписують осі відповідно O*x* і O*y* так, як замовлено в аргументі команд (лапки, прості **'** або подвійні **",** використовувати обов′язково!);

title("Довільна назва рисунка") – надписує над графіком вказану в кавичках назву.

axis([*- 40 30 -.4 1.5*]) "керує" розташуванням побудованих кривих між осями координат $X_{\min} \le x \le X_{\max}$, $Y_{\min} \le y \le Y_{\max}$. У даному випадку $X_{\min} = -40$, $X_{\max} = 30$, $Y_{\min} = -.4$, $Y_{\max} = 1{,}5$. **Увага:** лівий і правий кінці осі O*x*, нижній та верхній кінці O*y* заключати в прямокутні дужки **[** та **]**!

Нарешті, спробуйте побудувати графік функції (1.4) у вигляді "комети":

*>> N=1000; x= -10*pi:pi/N:10*pi; comet(x,sin(x)./x).*

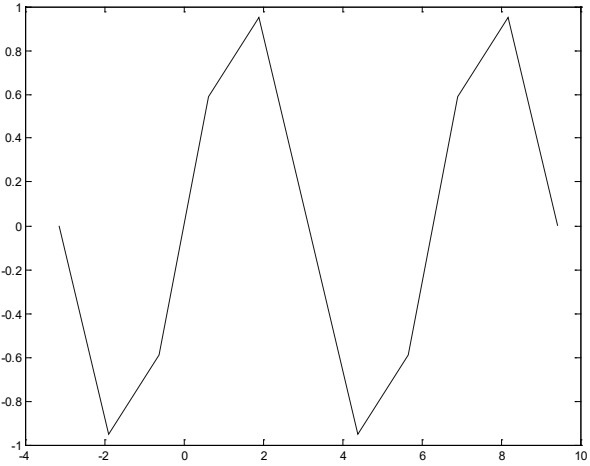


Рис. 1.3. Графік функції *y=sin*(*x*), побудований за одинадцятьма точками

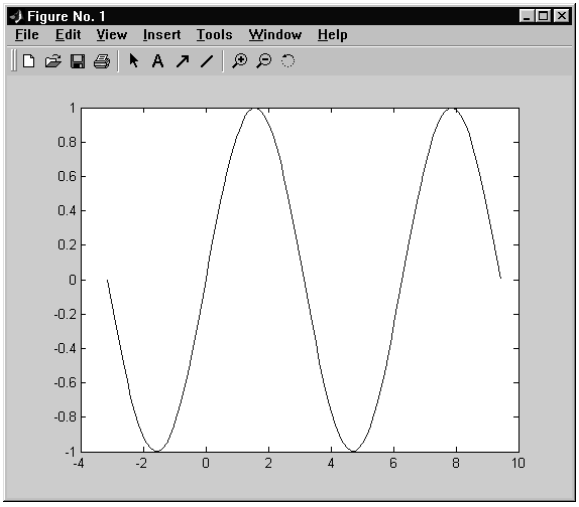


Рис. 1.4. Більш "гладкий" графік тієї ж функції *y=sin*(*x*), показаний разом з вікном *Figure* MatLab

(Ще раз спробуйте, але цього разу N=100). Красиво!

Інший спосіб зробити графік більш інформативним – використати розвинене меню, яке включено в *Figure* (див. рис. 1.4). Наприклад, *Insert* надає можливості вставити надписи під осями (підменю *Xlabel*, *YLabel*), дати рисунку загальну назву (*Title*), пояснити одну чи більше кривих текстом (*Legend*), вказати на щось важливе стрілкою (*Arrow*) та пояснити це текстом (*Text*). Кнопка меню  дозволяє збільшити якусь частину графіка, аби дослідити її більш детально (для цього лівою кнопкою миші обведіть прямокутником потрібний фрагмент). Є і інші можливості, які допитливий читач дослідить самостійно.

Що робити, аби зберігти вже побудовані графіки, а наступні – побудувати від них окремо? Виконайте попередньо команду

>> Figure.

Тепер з′явиться нове вікно для наступних графіків. Можна замовити вікно з певним номером: *Figure*(*3*), і т.п.

Ми бачили, що команда *plot* потребує певної підготовчої роботи – подумати про масив значень аргументу, крок між ними тощо. Всю цю роботу можна доручити “більш розумній” функції MATLAB *fplot*. Ось який у неї формат:

fplot (*‘tan*(*x*)*/x’* ,[*-pi/2 pi/2 0 4*] ,*’: r’*).

В першій позиції у лапках вказане ім′я функції, яку треба побудувати – у даному випадку *y=tg(x)/x*; друга позиція відведена для того, щоб вказати (у прямокутних дужках) цікаві для вас області зміни аргументу $-\pi/2 \le x \le \pi/2$ (це обов’язково!) і функції $0 \le y \le 4$ (а це можна опустити); нарешті, у третій позиції можна вказати (знов у лапках!) тип лінії і її колір (у даному випадку *′:′* означає пунктир, *′r′* відповідає червоному кольору кривої, від *red*). Можливі стилі можна знайти у Help по такому шляху:

Contents → *MATLAB* → *Using MATLAB* → *Graphics* → *Basic Plotting* → *Specifying Line Style*

або

Specifying the Color and Size of Lines

Змінити стиль і колір ліній можна також за допомогою кнопки  вікна *Figure*, після натискання якої клацнути лівою або правою кнопкою миші на потрібній кривій і обрати потрібне з меню, яке з′явиться.

Часто виникає потреба надрукувати побудований графік. Для цього натисніть кнопку  вікна *Figure*. Існує також багато способів вставити графік у ваш звітний Word-документ (дипломну роботу тощо). Перший спосіб: у вікні *Figure* виконати

Edit \ *Copy Figure*,

після чого перейти у *Word*-документ і зробити *Правка\Вставить*. Другий спосіб передбачає попереднє збереження рисунку у тому чи іншому графічному форматі: в меню по шляху *File\Export*… обрати той чи інший графічний формат (*bmp*, *jpg*, інший) та зберігти файл у вашій робочій директорії (папці). Тепер, або в інший зручний час рисунок можна вставити у *Word*-документ за допомогою

Вставка\Рисунок ► *\Из файла…*

Побудова графіків дозволяє глибше засвоювати математичні науки, і просто дає насолоду від роботи за комп′ютером! Аби збільшити ваши можливості, розглянемо ще параметричне задання кривих і побудову графіків у полярних координатах.

Нехай задано дослідити кардіоїду*, яка задається двома рівняннями залежності абсциси і ординати від параметра t:

$$x = 2a\cos t - a\cos 2t,$$
$$y = 2a\sin t - a\sin 2t$$

Команди MATLAB

```
>> a=1; t=0:.01:2*pi; x=2*a*cos(t)-a*cos(2*t);
>> y=2*a*sin(t)-a*sin(2*t);
>> plot(x,y), title('Параметрична крива'); axis equal
```

* Назва кривої зрозуміла з рис. 1.5

призводять до появи графіка, що нагадує серце і який показано на рис. 1.5.

Широкі можливості надає побудова графіків у полярній системі координат з використанням команди *polar*. Наприклад, відома крива *спіраль Архімеда* задається рівнянням $\rho = a\,\theta$. Її графік у полярних координатах можна побудувати за допомогою таких команд:

*>> Theta=0 : 10*pi/1000 : 10*pi*; *Rho=5*Theta*; *polar*(*Theta*,*Rho*)

Результат показано на рис. 1.6. На графіку відмічаються значення кута *Theta,* який відкладають від координатної осі проти годинникової стрілки, а також (штриховими колами) значення полярного радіуса *Rho*.

Інші приклади побудови графіків можна знайти у довіднику HELP до MATLAB.

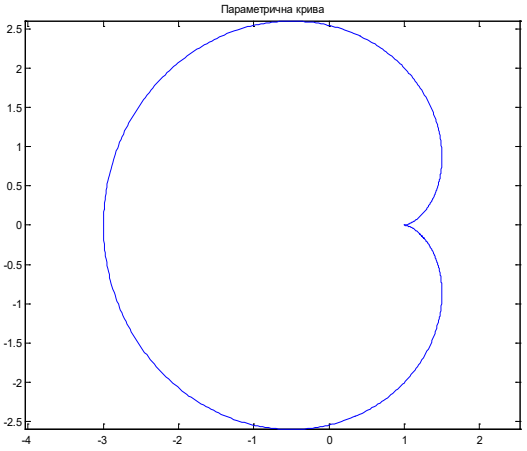


Рис. 1.5. Крива "кардіоїда", яка побудована за параметричним рівнянням

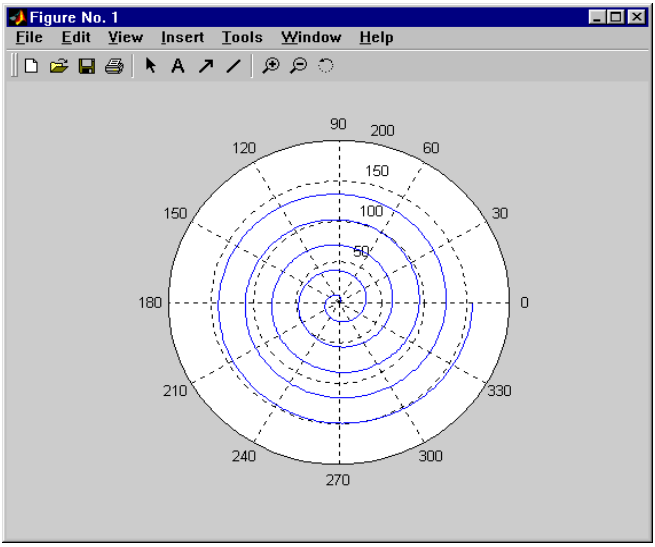


Рис. 1.6. Крива (спіраль Архімеда) у полярних координатах

1.6. ЧИСЕЛЬНІ І “АНАЛІТИЧНІ” ОБЧИСЛЕННЯ В MATLAB

Числа, вектори і матриці (масиви) – це **числові об‘єкти**, бо складаються з певної кількості чисел, і дії над ними проводяться за арифметичними алгоритмами. Логіка дій і, відповідно, алгоритми зовсім інші, коли ми проводимо аналітичні перетворення, такі, як, припустимо, піднесення до квадрату $(a+b)^2 = a^2 + 2ab + b^2$. Результат такого перетворення дійсний для довільних a і b.

Доречно згадати, що алгоритми **аналітичної математики** одними з перших у світі були розроблені в Інституті кібернетики Академії Наук України і реалізовані в ЕОМ “МИР-2”. Більш конкурентноспроможним виявився, однак, американський пакет ***Maple***, алгоритми якого пізніше увійшли і в MATLAB. Тут познайомимось з деякими корисними командами *Symbolic Math Toolbox* (допоміжного пакета *Символічна Математика*), що входить до складу MATLAB.

Різницю між числовими і символічними об‘єктами MATLAB можна показати на прикладі многочленів. Вектор $p=[a_1,a_2,\ldots,a_n]$ певні програми MATLAB сприймають як числовий об‘єкт, що представляє многочлен

$$p(x) = a_1 x^n + a_2 x^{n-1} + a_3 x^{n-2} + ... + a_n x + a_{n+1} \qquad (1.5)$$

(елементи вектора відповідають коефіцієнтам многочлена у порядку спадання степенів). Так, команда ***polyval***(*p,x₀*) (від *polynomial value*) обчислює цей многочлен в точці x_0, тобто отримує $p(x_0)$.

Команда (оператор) ***roots***(*p*) знаходить всі корені многочлена*) $p(x)$. Якщо, навпаки, задати вектор-стовпчик з коренів многочлена *r*, то команда ***poly*** знаходить вектор-рядок коефіцієнтів полінома. (Перевірте дію наведених тут і нижче команд на прикладах, запропонованих в лабораторній роботі № 3 [3]).

Нехай дано ще один многочлен $q=[b_1,b_2,...,b_n]$. Тоді команда ***conv***(*p,q*) формує новий многочлен *P*, що дорівнює добутку заданих, тобто $P(x)=p(x)q(x)$.

Все це – *обчислювальні* команди і алгоритми. Проте, добре було б діяти і за звичними нам правилами алгебри. Для цього перш за все слід оголосити аргумент *x* ***символьним об'єктом***, аби MATLAB не наполягав призначати йому якесь певне значення. Це можна зробити командою *sym* у форматі

*>> P=sym('a1*x^2+b1*x+c1')*

(назва команди опису змінної походить від *symbolic*). MATLAB відповість *P = a1*x^2+b1*x+c1*, тобто він не вважатиме коефіцієнти *a1*, *b1*, *c1* і змінну *x* невизначеними величинами, а сприймає *P* як якийсь цілісний об'єкт. Можна замовити MATLAB показати цей об'єкт у більш "*приємному*" (*pretty*) вигляді:

>> pretty(P)

Він дає:

$$a1\ x^2 + b1\ x + c1$$

Так само введемо об'єкт *Q= a2*x^2+b2*x+c2* – інший символічний многочлен другого степеня. Спробуємо помножити ці многочлени:

*) Відомості про корені многочленів див. в розділі 4.2 та в підручнику [32]

```
>> PQ=P*Q
```

Відповідь MATLAB: *PQ* = (*a1*x^2+b1*x+c1*)*(*a2*x^2+b2*x+c2*). Він просто замінив *P* і *Q* на їх символічні значення (вирази), тобто працює з цілісними об'єктами, утворивши символьний об'єкт *PQ*, ще "не знаючи", що з останнім можна поводитись за правилами алгебри. Команда *expand* (*розкласти*) роз'яснює йому це наше бажання:

```
>> PQ=expand(PQ)
  PQ =a1*x^4*a2+a1*x^3*b2+a1*x^2*c2+b1*x^3*a2+b1*x^2*b2+
                                  b1*x*c2+c1*a2*x^2+c1*b2*x+c1*c2
```

Як бачимо, MATLAB виконав множення, але ще не "розуміє", що в результаті, який отримано, можна привести подібні (з одинаковими степенями від *x*) члени. Підкажемо йому далі командою *collect*:

```
>> PQ=collect(PQ)
  PQ=a1*x^4*a2+(a1*b2+b1*a2)*x^3+(a1*c2+b1*b2+c1*a2)*x^2+
                                  (b1*c2+c1*b2)*x+c1*c2
```

Нарешті отримали символічний вираз *PQ*, що представляє добуток двох многочленів. Зрозуміло, такий результат дійсний для будь-яких коефіцієнтів *a1*, *a2*, *b1*, *b2*, *c1*, *c2*.

Багато корисного можна одержати, володіючи символьною математикою. Припустимо, треба пригадати, чому дорівнюють корені квадратного рівняння, які назвемо *roots*. Допоможе команда *solve*(*P*) (*розв'язати* рівняння *P=0*). Отримуємо, що масив *roots* складається з двох елементів:

```
>> roots=solve(P)
roots =
        [ 1/2/a1*(-b1 +(b1^2 -4*a1*c1)^(1/2))]
        [ 1/2/a1*(-b1 -(b1^2 -4*a1*c1)^(1/2))]
```

Візьмемо з нього перший елемент

```
>> r1=roots(1)
r1 = 1/2/a1*(-b1+(b1^2 -4*a1*c1)^(1/2))
```

і використаємо команду запису в більш звичній нотації:

```
>> pretty(r1)
```

Маємо в результаті відому формулу:

$$-b1 + (b1^2 - 4\,a1\,c1)^{1/2}$$
$$1/2 \text{ ---------------------------} \quad .$$
$$a1$$

Часто потрібно переходити від символьних об‘єктів до чисельних. Для цього слід використовувати команди ***sym2poly*** і ***poly2sym***. Перша перетворює символьний многочлен у числовий, наступна – навпаки, поліном як чисельний об‘єкт перетворює в символьний. Пояснимо на прикладі.

Раніше отримано *PQ* – многочлен у символьній формі з довільними (символічними) коефіцієнтами *a1*, *b1*,*c1* і *a2*, *b2*, *c2*. Надамо спочатку першій групі, а потім другій групі коефіцієнтів певних значень і підставимо (***subs****titute*) їх у многочлен:

```
>>  a1=1; b1=2; c1=3; PQ2 =subs(PQ)
PQ2 = x^4*a2+(b2+2*a2)*x^3+(c2+2*b2+3*a2)*x^2+(2*c2+
                                                3*b2)*x+3*c2
>> a2=2;b2=1;c2=3; PQ 1=subs(PQ2)
     PQ1 = 2*x^4+5*x^3+11*x^2+9*x+9
```

Перша команда створила символьний многочлен *PQ2*, що залежить лише від іншої групи коефіцієнтів, а друга команда створила многочлен *PQ1* вже з числовими коефіцієнтами. Однак, *PQ1* – все ще символьний об‘єкт. Знайдемо його чисельний аналог:

```
>> pq1=sym2poly(PQ1)
pq1 =    2    5   11    9    9
```

Отримано вектор *pr1*, складений з коефіцієнтів многочлена. Команда

```
PQ1=poly2sym(pq1),
```

навпаки, перетворить цей числовий вектор в символьний об'єкт.

Такими є перші кроки з символьною математикою. Таким чином, перш, ніж її використовувати, MATLAB треба створити символьні об‘єкти. Цьому служать команди *sym* і *syms*. Покажемо застосування символьної математики на подальших прикладах.

Приклад 1.5. *Знайти похідну функції* $y=\left(\frac{x}{1+x}\right)^x$

(*№ 663* із збірника задач Бермана [30]*)*.

Приклад 1.6. *Знайти первісну функції* $y=\left(\frac{1}{1+\sqrt{x+1}}\right)$

(*№ 1869* [30]).

Утворюємо два символьних об‘єкти

```
>> Berman663=sym( '(x/(1+x)^x)' ),  Berman1869=sym( '1/(1+sqrt(1+x))' )
```

Знаходимо похідну

```
>> d663=diff(Berman663)
     d663 = 1/((x+1)^x)-x/((x+1)^x)*(log(x+1)+x/(x+1))
```

і невизначений інтеграл:

```
>> int1869=int(Berman1869)
     int1869 = -log(x)+2*(x+1)^(1/2)-2*atanh((x+1)^(1/2))
```

Отримані вирази слід переглянути за допомогою команди *pretty*:

```
>> pretty(d663)
                                  /                  x   \
                                x |log(x + 1) + -------|
      1                           \                x + 1/
  -------- - ---------------------------
         x                        x
  (x + 1)                  (x + 1)
```

Можна бачити, що одержані вирази дещо відрізняються від відповідей підручника:

$$d663=\left(\frac{x}{x+1}\right)^x\left(\frac{1}{x+1}+\ln\frac{x}{x+1}\right),$$

$$\text{int}1869=2\left(\sqrt{x+1}-\ln(1+\sqrt{x+1})\right)+C\,.$$

Проте, у кожному випадку наші результати тотожні відповідям.

Можна спробувати доручити MATLAB самому знайти і запропонувати нам тотожні вирази. Так, команда *simple*(*d663*)

видасть певний перелік можливих тотожніх записів для похідної *d663*, кожний з яких використовує ті чи інші запрограмовані правила алгебри, тригонометрії, математичного аналізу тощо. Але не всі правила вдалося запрограмувати фірмі *MathWorks*…

Приклад 1.7. *Обчислити визначений інтеграл* $\int_0^\infty e^{-x^2}\,dx$ (інтеграл Гаусса).

Наводимо команди і відповіді MATLAB на них:

```
>> syms t k                                                    (1.6)
>> Gauss = int (exp(-t^2),0,inf)
       Gauss = 1/2*pi^(1/2)
>> pretty(Gauss)
                          1/2
                      1/2 pi
```

що є відомим результатом $\frac{1}{2}\sqrt{\pi}$.

Кожному студенту радимо перевірити себе на подібних задачах лабораторної роботи 3, наведеної в посібнику [3].

Приклад 1.8. *Обчислити наближено і точно суму знакозмінного ряду* $1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\ldots$.

Наближене значення обчислюємо, взявши 5 і 10 доданків у частинній сумі, а точне значення – використовуючи ідентифікатор inf (від *infinity*, *нескінченність*):

```
>> S5=1+symsum((-1)^k/k, 1, 5),  S10=1+symsum((-1)^k/k, 1, 10),
     S5 = 13/60
     S10 = 893/2520
>>  S=1+symsum((-1)^k/k,1,inf)
     S = 1-log(2)
```

Тобто п‘ять доданків дають 0.21667, десять – 0.35437, а точне значення суми ряду є $1-\ln 2 \approx 0.30685$. При цьому враховано, що символьну змінну *k* попередньо було вже введено в (1.6). (Значення змінних *S5*, *S10* і *S* MATLAB видав у "раціональній формі"; про

інші можливі формати див. *help sym*. В даному випадку слід скопіювати ці дробові значення, розмістити у командний рядочок і отримати значення у десятковій формі).

Приклад 1.9. *Розкласти функцію* $y = e^{-t^2}$ *в ряд Тейлора, отримавши 10 членів розкладу.*

```
>> taylor(exp( - t^2), 11)
     ans = 1- t^2+1/2*t^4 - 1/6*t^6+1/24*t^8 - 1/120*t^10
```

(Символьна змінна *t* була вже створена; команді *taylor* дано завдання створити *11*, а не *10* членів ряду, бо команда сприймає таке завдання як "*не більше ніж*…". У відповіді *ans* присутні насправді 6 доданків, бо коефіцієнти при непарних степенях – нулі).

Розглянемо можливості аналітичного розв‘язування трансцендентних і диференціальних рівнянь від однієї невідомої.

Приклад 1.10. *Розв‘язати рівняння* $\arcsin x = p(1-x)$ *для* $p > 0$.

Відомо, що це рівняння трансцендентне, аналітичного розв‘язку для нього не існує. "Знаючи" це, MATLAB швидко знаходить чисельні розв‘язки для визначеного наперед параметра p. Використовуємо команду *solve* (*розв‘язати*):

```
>> % Для p=.5 :
>> X1=solve('asin(x) = .5*(1-x)')
     X = .32916
>> % Для p= 4 :
>> X2=solve('asin(x) – 4*(1-x)')
     X = .77738
```

(**Зверніть увагу:** рівняння можна писати у різних формах; використовувати лапки обов‘язково).

Приклад 1.11. *Розв‘язати диференціальне рівняння* $y'' + 9y = 0$ [29], №*4301*.

Диференціальні рівняння і навіть системи диференціальних рівнянь запрограмовано розв'язувати команді *dsolve*. В її

аргументах перелічують рівняння системи і граничні умови (кожне – в лапках, через кому):

```
>> Berman4301=dsolve(' D3y+9*Dy=0 ')
      C1+C2*sin(3*t)+C3*cos(3*t)
```

Тобто рівняння записують в певній умовній формі: *Dy* означає першу похідну і *D3y* третю похідну функції *y*(*t*), аргументом вважають "час" *t*, якщо інше не вказано. Можна пересвідчитись у правильності отриманого розв‘язку, який включає три довільні константи. А ось як виглядали б завдання і розв‘язок у випадку граничних умов $y(0)=1,\quad y'(0)=1,\quad y(10)=1$:

```
>>Berman4301=dsolve('D3y+9*Dy=0', 'y(0)=1', 'Dy(0)=1', 'y(10)=1', 'x')
    Berman4301 =  1/3*(3*cos(30)+sin(30)-3)/(-1+cos(30)) –
                         1/3*sin(30)/(-1+cos(30))*cos(3*x)+1/3*sin(3*x)
```

Щоб полегшити прочитання розв‘язку, застосуємо знайому вже команду:

```
>> pretty(Berman4301)

        3 cos(30) + sin(30) - 3            sin(30) cos(3 x)
  1/3 ----------------------------  –  1/3 -----------------------  +  1/3 sin(3 x)
           – 1 + cos(30)                      – 1 + cos(30)
```

Бачимо, що MATLAB знайшов константи *C1*, *C2*, *C3* нібито користуючись відомими з вищої математики методами. Зверніть увагу, як в команді MATLAB було наказано саме *x* вважати незалежною змінною.

Однак поки ще не кожне рівняння або система рівнянь можуть бути розв‘язані аналітичною математикою MATLAB, не все зуміли запрограмувати фахівці фірми MathWorks. Може ви доповните? Оскільки готового результату вам не гарантовано – будьте готові трохи експериментувати з пакетом!

Аналітичну математику MATLAB з успіхом можна застосувати для вивчення перетворень Лапласа і перетворень Фур'є. Операційне числення (перетворення Лапласа) використовується в теорії автоматичного управління. Перетворення Фур'є широко застосовується в проектуванні антен, фільтрів радіо-

і телевізійних сигналів, в оптиці, квантовій фізиці, теорії ймовірностей тощо. Два приклади аналітичного обчислення цих перетворень в MATLAB:

Приклад 1.12. *Знайти перетворення Лапласа функції часу*

$$h(t) = \sin(f_0 t), \tag{1.7}$$

побудувати графіки прообразу і образу для значення параметру $f_0 = 5$ *Гц*.

Виконуємо у командному вікні:

```
>> syms h f0 t H p
>> h=sin(f0*t)
     h = sin(f0*t)
>> H=laplace(h,p)
     H = f0/(p^2+f0^2)
>> f0=5; h=subs(h), H=subs(H)
     h = sin(5*t)
     H = 5/(p^2+25)
>> subplot(211); ezplot(h), subplot(212); ezplot(H)
```

Перша команда створила символьні змінні *h*, *f0*, *t*, *H* і *p*, а друга команда надала *h* значення тригонометричної функції часу *t* з круговою частотою *f0*. Після цього команда *laplace* обчислила *H* як образ Лапласа від *h*(*t*); другий параметр команди *laplace* наказує їй використовувати символ *p* як аргумент новоствореної функції (за умовчанням таким аргументом MATLAB пише *s*). Функція, що отримана, є вірною для будь якого *f0*. Проте, для побудови графіків треба *f0* надати певного значення; конкретизовані функції бачимо після четвертої команди. Нарешті, п'ята команда будує графіки функцій *h*(*t*) і *H*(*p*) відповідно у верхній і нижній частинах графічного вікна (рис. 1.7).

Приклад 1.13. *Знайти перетворення Фур'є функції*

$$h(t) = e^{-a|t|} \cos(f_0 t), \tag{1.8}$$

прийнявши $a = 1$ *і* $f_0 = 10$ *Гц*.

Перетворення Фур'є виконаємо (в аналітичному вигляді) командою *fourier* і побудуємо графіки образу $H(f)$ (функції частоти) і прообразу $h(t)$ (функції часу):

```
>> syms h t H f
>> h=exp(-abs(t))*cos(10*t), H=fourier(h,f)
     h = exp(- abs(t))*cos(10*t)
     H = 1/(1+(f-10)^2)+1/(1+(f+10)^2)
>> subplot(211), ezplot(h), axis([ -5 5 -.8  1.1])
>> subplot(212), ezplot(H, [ -15 15]), axis([ -15 15 -.1  1.1])
```

Першою командою введені символічні змінні для функції *h* часу *t* і її Фур'є-образу *H* (функції частоти *f*). Друга команда створює символьний запис *h*(*t*) і аналітично обчислює її Фур'є-перетворення *H*(*f*); показано відповіді MATLAB. Третя і четверта команди будують відповідні графіки у верхній і нижній половинах графічного вікна (рис. 1.8). Перевіримо, чи поверне нас до *h*(*t*) обернене (*inverted*) перетворення Фур‘є $\mathrm{F}^{-1}[H(f)]$:

```
>> h1=ifourier(H)
    h1=1/2*(exp(-x)*Heaviside(x)+exp(x)*Heaviside(-x))*
                                        (exp(10*i*x) +exp(-10*i*x))
>> simplify(h1)
      ans=cos(10*x)*(exp(-2*x)*Heaviside(x)+1-
                                        Heaviside(x))*exp(x)
```

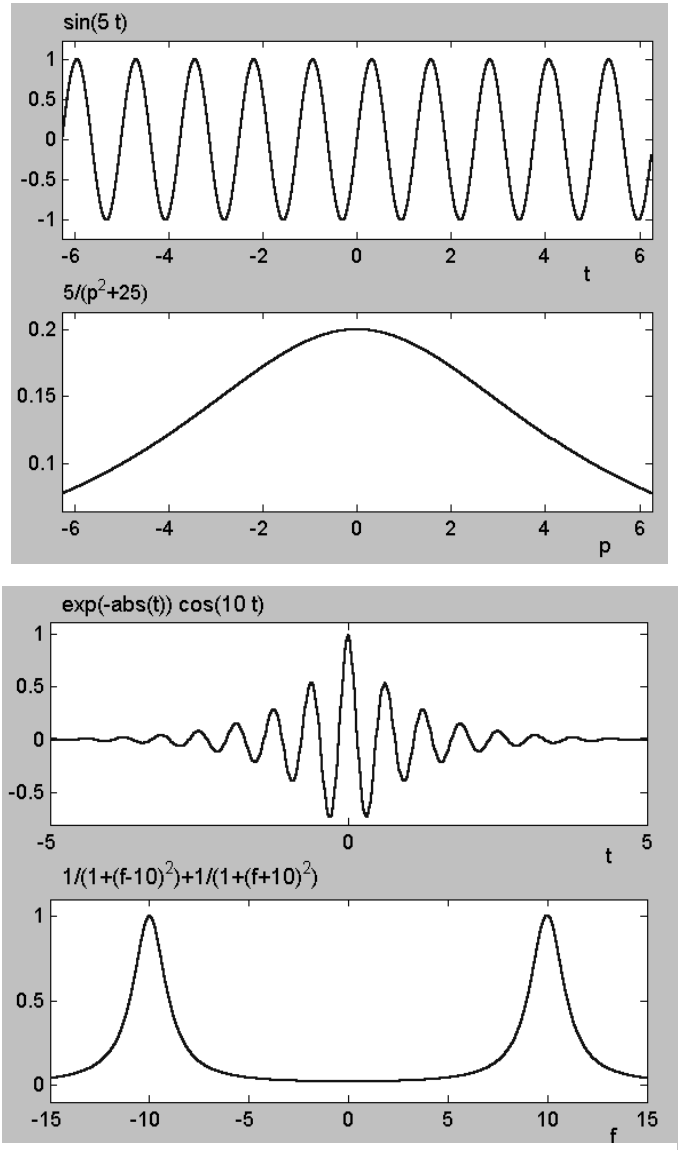


Рис. 1.8. Графіки функції (1.8) і її Фур‘є образу

Якщо спочатку отримали комплексну функцію, то командою *simplify* вдалося її трохи спростити, показати, що насправді вона є

дійсною. Останній вираз – найпростіший, який вдається отримати MATLAB. В ньому присутня відома функція *Heaviside*(*x*), що дорівнює *0* при $x < 0$ і дорівнює *1* при $x \geq 0$. …

Фур'є-образи деяких функцій, таких як $h(t) = \sin t$ або $h(t) = \cos t$, приводять до δ-функції *Dirac*(*f*) (усюди нуль, лише для $f = 0$ вона дорівнює нескінченності, ∞). MATLAB не вміє будувати графіків таких функцій (до речи, "навчити" MATLAB могло б бути темою курсової роботи студента).

Звертаємо увагу, що для побудови графіків символьно заданих функцій широко використана нова команда ***ezplot***. Вона не потребує, на відміну від ***plot***, створення вектора значень аргумента; ***ezplot*** це робить "сама", надаючи аргументу значення між -2π і 2π (приклад 1.11, рис. 1.7). Першим аргументом ***ezplot*** є ім'я функції (*h* або *H*); можна написати аналітичний вираз функції безпосередньо, беручи його в лапки – ***ezplot***(*'sin(x)/x'*). Наступним аргументом (другим, як тут, або третім) можна вказати область задання аргумента функції, наприклад: ***ezplot***(*H*,[– *3*pi*, *4*pi*]). Також можна другим аргументом ***ezplot*** вказувати функцію (теж у лапках!); таке завдання MATLAB зрозуміє як побудову графіка параметрично заданої функції. Приклад:

ezplot(' *sin*(*3*t*)**cos*(*t*) ', ' *sin*(*3*t*)**sin*(*t*) ', [*0*, *pi*]).

Сподіваємось, символічна математика MATLAB буде вам в нагоді для вивчення курсу вищої математики та в практичній роботі. Закріпити отримані знання допоможе виконання завдань лабораторної роботи 3 посібника [3].

1.8. КОНТРОЛЬНІ ПИТАННЯ ДО РОЗДІЛУ 1

1. Які "вікна" MATLAB ви знаєте? Яке з них, власне, потрібне для роботи? Як викликати вікно Help, як його використовувати?
2. Як викликати довідку про побудову графіків?
3. Для чого в MATLAB призначені коментарі, як вони вводяться?
4. Які використовуються формати для дійсних і комплексних чисел? Як вводяться масиви чисел – вектори і матриці?

5. Які імена можна надавати змінним в MATLAB? Чи можна надавати імена *ans*, *pi*, *i*, *j*? Як можна надавати змінним ті чи інші значення?
6. Які ви знаєте дії із змінними в MATLAB? За яких умов дії +, –, *, / і ^ можна застосовувати до матриць? Чи можна застосувати операції ^ і *sqrt* до матриць? Чим відрізняються деякі з названих операцій від їх відповідників "з крапкою" **.***, **./** і **.^**? Як отримати довідку про арифметичні оператори?
7. Як можна отримати значення тригонометричних функцій від заданих аргументів? Експоненти і логарифма? Гіперболічних функцій? Чи може бути аргументом названих функцій вектор або матриця? А комплексне число?
8. Як відокремлювати команди у командному вікні MATLAB, якщо їх декілька? Яка різниця між роздільниками "**,**" (кома) і "**;**" (крапка з комою)?
9. Як можна повторити команду, попередньо вже виконану?
10. Яке призначення функцій від масивів *length*, *size*, *inv*, *norm* і *diag*?
11. Що означає команда MATLAB на зразок
    x=-pi **:** *pi/100* **:** *3*pi* ? Навіщо вона потрібна для побудови графіків функцій?
12. Які команди для побудови графіків функцій ви знаєте? Чим відрізняються команди *plot*, *fplot*, *ezplot* і *comet*?
13. Яким чином команди *legend*, *title*, *grid*, *xlabel*, *ylabel*, *axis*, *insert* "прикрашають" графіки? Як можна задати або змінити колір кривих і позначки на них?
14. Для чого використовують команду *figure*?
15. Як ви можете пояснити різницю між числовими і символьними об'єктами MATLAB? Як знайти корені многочлена? Скільки їх має бути? Як знайти похідну і первісну заданої функції? Як обчислити визначений інтеграл? Суму ряду? Фур'є– або Лапас-образи заданої функції?

Як можна відобразити у *Word*-документі результати вашої роботи з програмою MATLAB – введені команди і отримані результати, побудовані графіки?

2. "РУЧНЕ" РОЗВ'ЯЗУВАННЯ ТИПОВИХ ЗАДАЧ МАТЕМАТИЧНОГО МОДЕЛЮВАННЯ

В даному розділі універсальний і потужний комп'ютерний пакет MATLAB використовується лише як швидкий і розвинений калькулятор, а головну увагу звернено на зміст математичних методів, що розглядаються. Відібрано методи обчислень, які найчастіше використовуються в типових задачах математичного моделювання, таких як системи лінійних алгебраїчних рівнянь (СЛАР), нелінійні рівняння з одним невідомим, знаходження емпіричних формул методом найменших квадратів, інтегрування функцій, початкові та крайові задачі для звичайних диференціальних рівнянь. Матеріал викладено у короткій, довідковій формі з посиланням на більш грунтовну літературу. Студенту важливо володіти ідеєю метода, розуміти коректність кожної даної задачі – за яких умов гарантовані існування та єдиність її розв'язку. Бо якщо в математичній моделі немає розв'язку – як таке зрозуміти фізично? Якщо ж розв'язків декілька – який з них використовувати в практиці? Важливо також знати умови збіжності методу, способи перевірки достовірності і точності результату.

"Ручне" обчислення за допомогою MATLAB насправді достатньо швидке і необтяжливе. Проте, студент орієнтований на те, аби всі дії проводити за єдиним алгоритмом. Таке виконання обчислень має підготувати вас до програмування тих самих задач засобами MATLAB, чому присвячений наступний розділ.

2.1. РОЗВ′ЯЗУВАННЯ СИСТЕМ ЛІНІЙНИХ АЛГЕБРАЇЧНИХ РІВНЯНЬ

Системи лінійних алгебраїчних рівнянь – це найпростіша з математичних задач. Ця задача, однак, має важливе самостійне значення, оскільки до неї безпосередньо приводять багато практичних моделей фізичних, технічних та економічних явищ. Крім того, до таких рівнянь та необхідності їхнього розв'язування призводять задачі більш складні. Системи лінійних алгебраїчних рівнянь – фундамент обчислювальної математики.

Будь-яка математична операція f, що може бути застосована до елементів x, y,… якоїсь множини, називається *лінійною*, якщо результатом застосування f до $x+y$ та до Cx (де C стала) буде відповідно $f(x)+f(y)$ та $C\,f(x)$. З курсу лінійної алгебри [32–39] відомо, що множення матриці A вимірності (m,n) на вектор-стовпець x вимірності $(n,1)$ є саме лінійною операцією, результатом якої є новий вектор y вимірності $(m,1)$. *Матричним рівнянням* називають запис

$$Ax=b, \tag{2.1}$$

де A матриця (m,n), b – деякий n-вимірний вектор-стовпець, а x – n-вимірний вектор-стовпець, що розшукується:

$$A=\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}, \qquad b=\begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_m \end{pmatrix} \qquad x=\begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix}. \tag{2.2}$$

Подане матричне рівняння може також розглядатись як *система m лінійних рівнянь з n невідомими*:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_m. \end{cases} \tag{2.3}$$

Розв'язком матричного рівняння (2.1), або **розв'язком системи** (2.3) називають такий вектор x, що обертає їх в тотожності. В більшості випадків мають справу *з квадратними матрицями*, $n = m$, тобто з випадком, коли *кількість рівнянь збігається з кількістю невідомих*. Умови існування та єдиності розв'язку такої задачі точно встановлені.

Розглянемо методи розв'язування таких систем лінійних рівнянь. Теоретично, розв'язати їх можна за допомогою визначників. З курсу вищої математики [33] відоме *правило* (*метод*) *Крамера*. За цим правилом, при розв'язуванні системи (2.1), *k*-та компонента x_k вектора x визначається згідно з формулою

$$x_k = \frac{\Delta_k}{\Delta}, \tag{2.4}$$

де Δ –визначник матриці A, Δ_k – визначник матриці A_k, в якій *k*-тий стовпчик замінюється вектором b. Інший спосіб (матричний) полягає в обчисленні оберненої матриці A^{-1}, після чого

$$x = A^{-1} b. \tag{2.4A}$$

Вказані теоретичні результати встановлюють умови існування і єдиності розв'язку даної задачі:

Матриця A має бути невиродженою, тобто $\Delta \neq 0$.

Однак для практичних задач, коли кількість рівнянь n велика (а в сучасних технічних або економічних задачах n дорівнює кількох тисяч) ці методи стають надто громіздкими і потребують значних затрат машинного часу, тому в

обчислювальній практиці застосовуються не часто. Розглянемо два методи, зручних саме для ЕОМ.

2.1.1. МЕТОД ГАУССА РОЗВ'ЯЗУВАННЯ СИСТЕМ ЛІНІЙНИХ РІВНЯНЬ

Цей метод відомий вам ще зі школи як *метод виключення невідомих*. Розглянемо систему рівняннь (2.3), в якому вільні члени будемо позначати як $a_{k,n+1} = b_k$. Розділимо перше рівняння системи на a_{11} і результат запишемо у вигляді

$$x_1 + a_{12}^{(1)} x_2 + a_{13}^{(1)} x_3 + ... = a_{1,n+1}^{(1)}. \qquad (2.5)$$

Тут і надалі верхній індекс означатиме номер перетворення, яке здійснене. Помножимо це рівняння на $-a_{21}$ і додамо його до другого рівняння системи (2.3). Коефіцієнти отриманого другого рівняння мають тепер вигляд $a_{2j}^{(1)} = a_{2j} - a_{21} a_{1j}^{(1)}$, де індекс j пробігає значення $j = 1,2,..., n+1$. При цьому перший коефіцієнт нового рівняння $a_{21}^{(1)}$ обертається в нуль. Так само поводимось із наступними рівняннями, тобто від k-го рівняння віднімаємо рівняння (2.5), попередньо помножене на a_{k1}. Його нові коефіцієнти мають вигляд

$$a_{kj}^{(1)} = a_{kj} - a_{k1} a_{1j}^{(1)} \text{ для всіх } j = 1,2,..., n+1.$$

Після виконання таких перетворень для всіх рівнянь системи, $k = 2,3,..., n$, маємо нову систему рівнянь

$$\begin{aligned} x_1 + a_{12}^{(1)} x_2 + a_{13}^{(1)} x_3 + ... + a_{1n}^{(1)} x_n &= a_{1,n+1}^{(1)} \\ a_{22}^{(1)} x_2 + a_{23}^{(1)} x_3 + ... + a_{2n}^{(1)} x_n &= a_{2,n+1}^{(1)} \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots & \\ a_{n2}^{(1)} x_2 + a_{n3}^{(1)} x_3 + ... + a_{nn}^{(1)} x_n &= a_{n,n+1}^{(1)} \end{aligned},$$

де всі коефіцієнти в першому стовпчику матриці $A^{(1)}$ дорівнюють нулю, за винятком $a_{11}^{(1)} = 1$.

На наступному кроці перше рівняння нової системи (відповідно, перший рядочок матриці $A^{(1)}$) лишаємо незмінним, а аналогічну процедуру застосовуємо до рівнянь від другого до n-го. Маємо формули для нових коефіцієнтів (i – номер рівняння, j – номер стовпчика):

$$a_{2j}^{(2)} = a_{2j}^{(1)} / a_{22}^{(1)}, \quad j = 2,3,...,n+1,$$
$$a_{ij}^{(2)} = a_{ij}^{(1)} - a_{i2}^{(1)} a_{2j}^{(2)}, \quad i = 3,4,...,n, \quad j = 3,4,...,n+1.$$

Таку саму процедуру повторюємо для всіх рядків матриці A. Так, для k-го кроку ($k = 1,2,...,n-1$) маємо для елементів k-го і наступних i-х рівнянь:

$$a_{kj}^{(k)} = a_{kj}^{(k-1)} / a_{kk}^{(k-1)}$$
$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - a_{ik}^{(k-1)} a_{kj}^{(k)}, \qquad (2.6)$$
$$\text{де } i = k+1,...,n, \quad j = k+1,...,n+1.$$

Після $n-1$ кроків система рівнянь є еквівалентною вихідній і набуває такого "трикутного" вигляду:

$$\begin{aligned}
x_1 + a_{12}^{(1)} x_2 + a_{13}^{(1)} x_3 + a_{14}^{(1)} x_4 + ... + a_{1n}^{(1)} x_n &= a_{1,n+1}^{(1)} \\
x_2 + a_{23}^{(2)} x_3 + a_{24}^{(2)} x_4 + ... + a_{2n}^{(2)} x_n &= a_{2,n+1}^{(2)} \\
x_3 + a_{34}^{(3)} x_4 + ... + a_{3n}^{(3)} x_n &= a_{3,n+1}^{(3)} \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots & \\
x_{n-1} + a_{n-1,n}^{(n-2)} x_n &= a_{n-1,n+1}^{(n-2)} \\
a_{n,n}^{(n-1)} x_n &= a_{n,n+1}^{(n-1)}
\end{aligned} \quad . \qquad (2.7)$$

На цьому ***прямий хід*** методу виключення Гаусса закінчено. З трансформованої системи (2.7) безпосередньо маємо x_n, а наступні компоненти розв'язку рекурентно отримуємо "знизу вгору":

$$x_n = \frac{a_{n,n+1}^{(n-1)}}{a_{n,n}^{(n-1)}}, \qquad x_{n-1} = \frac{a_{n-1,n+1}^{(n-2)} - a_{n-1,n}^{(n-3)} x_n}{a_{n-1,n-1}^{(n-1)}}, \quad \ldots, \tag{2.8}$$

$$x_2 = a_2^{(2)} - a_{23}^{(2)} x_3 - \ldots - a_{2n}^{(2)} x_n, \quad x_1 = a_1^{(1)} - a_{12}^{(1)} x_2 - \ldots - a_{1n}^{(1)} x_n.$$

Цей процес послідовного обчислення значень невідомих називається ***зворотним ходом*** методу Гаусса.

Алгоритмічний процес ***методу виключення Гаусса*** складається, таким чином, з двох етапів. Якщо на етапі прямого ходу виникає випадок, коли $a_{kk}^{(r)} = 0$, то k-й рядок неможливо використовувати для виключення елементів k-го стовпчика. В цьому випадку k-й рядок замінюється іншим, що лежить нижче. Так здійснити неможливо лише у тому випадку, коли всі елементи $a_{p,k}^{(r)} = 0$, $p > k$, а це відповідає тому, що вихідна матриця A є виродженою і система не має єдиного розв'язку.

Розглянемо тепер реалізацію метода Гаусса в середовищі MATLAB.

Приклад 2.1. Розглянемо СЛАР вигляду

$$\begin{cases} x_1 + 2x_2 + x_3 + 4x_4 = 13 \\ 2x_1 + 4x_3 + 3x_4 = 28 \\ 4x_1 + 2x_2 + 2x_3 + x_4 = 20 \\ -3x_1 + x_2 + 3x_3 + 2x_4 = 6. \end{cases} \tag{2.9}$$

Використовуючи систему MATLAB, подамо матрицю A і вектор-стовпець b у вигляді

$A = [1\ 2\ 1\ 4;\ 2\ 0\ 4\ 3;\ 4\ 2\ 2\ 1;\ -3\ 1\ 3\ 2]$

$b = [13\ 28\ 20\ 6]'$.

На екрані монітора після натискання клавіші *Enter* з'явиться таке зображення

```
A =   1   2   1   4
      2   0   4   3
      4   2   2   1
```

```
      -3    1    3    2
b =   13
      28
      20
      6
```

Покажемо, як у MATLAB побудувати простий алгоритм її розв′язування методом виключення Гаусса. Спочатку командою *Ae*=[*A*, *b*] створимо розширену матрицю (4,5), яка включає як коефіцієнти рівнянь, так і праві частини:

```
Ab =
   1    2    1    4   13
   2    0    4    3   28
   4    2    2    1   20
  -3    1    3    2    6
```

Тепер командою *Eq1=Ab*(*1*,**:**) створюємо допоміжний вектор-рядок, що відповідає першому рівнянню:

```
Eq1 =    1    2    1    4   13
```

Так само командами *Eq2=Ab*(*2*,:), *Eq3=Ab*(*3*,:), *Eq3=Ab*(*3*,:) створюємо вектор-рядки, що відповідають трьом останнім рівнянням:

```
Eq2 =    2    0    4    3   28

Eq3 =    4    2    2    1   20

Eq4 =   -3    1    3    2    6
```

Тепер можна починати еквівалентні перетворення рівнянь відповідно до прямого ходу методу Гаусса.

Після команди *Eq1=Eq1*/*Eq1*(*1*,*1*) перший коефіцієнт першого рівняння дорівнює одиниці. Виконуємо тепер три команди

*Eq2=Eq2-Eq1*Eq2*(*1,1*), *Eq3=Eq3-Eq1*Eq3*(*1,1*), *Eq4=Eq4-Eq1*Eq4*(*1,1*)

Система рівнянь перетворена до вигляду (2.5). Тепер рівняння з другого по четверте перетворюємо командами

Eq2=Eq2/Eq2(*2,2*) та
*Eq3=Eq3-Eq2*Eq3*(*2,2*), *Eq4=Eq4-Eq2*Eq4*(*2,2*).

(Можна бачити, що вони подібні до попередніх. Тому, клавішею ↑, стрілка вверх, у командному вікні MATLAB можна викликати попередню команду і відредагувати її).

Тепер перетворюємо рівняння з третього по четверте

Eq3=Eq3/Eq3(*3,3*), *Eq4=Eq4-Eq3*Eq4*(*3,3*).

Прямий хід на цьому завершено. Допоміжні вектори-рядки тепер мають вигляд

```
Eq1 =   1   2   1    4     13;
Eq2 =   0  -4   2   -5      2;
Eq3 =   0   0  -5  -7.5   -35
Eq4 =   0   0   0   -9    -18
```

Таким чином, за формулами (2.6) отримано трикутну систему рівнянь вигляду (2.7).

Проводимо зворотний хід методу за формулами (2.8). Відразу маємо *x4=Eq4*(*1,5*)/*Eq4*(*1,4*)=*2*. Далі:

x3=Eq3(*1,5*) – *Eq3*(*1,4*)**x4=4*,
x2=Eq2(*1,5*) – *Eq2*(*1,4*)**x4-Eq2*(*1,3*)**x3= –1*,
x1=Eq1(*1,5*) – *Eq1*(*1,4*)**x4* – *Eq1*(*1,3*)**x3* – *Eq1*(*1,2*)**x2=3*.

Вектор-стовпець *X*=[*x1*; *x2*; *x3*; *x4*] буде розв′язком вихідної системи рівнянь даного прикладу.

2.1.2. МЕТОД ІТЕРАЦІЙ ДЛЯ СИСТЕМ ЛІНІЙНИХ РІВНЯНЬ

У випадку великої кількості невідомих є ефективним *метод ітерацій*. Цей метод відіграє важливу роль у чисельній математиці і застосовується також для розв'язування інших задач. Сутність методу щодо СЛАР полягає ось у чому. Спочатку матричне рівняння (2.1) запишемо у формі

$$x = A_1 x + b_1. \qquad (2.10)$$

Еквівалентне перетворення рівняння (2.1) до форми (2.10) можна зробити багатьма способами (див. далі приклад 2.2). Якщо,

припустимо, діагональні елементи матриці A всі відмінні від нуля, то в кожному її рядочку можна "відщепити" x_k від діагонального елемента рівняння і перенести $(a_{kk}-1)\,x_k$ вліво.

Розв'язуємо тепер систему (2.10). Якщо деякий вектор $x=x^{(0)}$ обрати за наближений розв′язок системи (2.1) (*нульове наближення*), то з системи (2.10) отримуємо $x^{(1)} = A_1 x^{(0)} + b_1$ – *перше наближення*. Тепер підставляємо $x^{(1)}$ в праву частину системи (2.10) і отримуєм *друге наближення* $x^{(2)} = A_1 x^{(1)} + b_1$. В такий спосіб обчислюємо, швидко і легко, *одноманітно*, послідовність векторів $\{x^{(1)}, x^{(2)},..., x^{(k-1)}, x^{(k)}\}$, кожний наступний з яких пов′язаний з попереднім співвідношенням

$$x^{(k)} = A_1 x^{(k-1)} + b_1. \qquad (2.11)$$

Якщо така послідовність збігається до якогось граничного вектора x^*, то останній і дає шуканий розв′язок системи (2.1). Доведення цього майже очевидного факту див. в підручниках [32-38]. А якщо послідовність не збігається? Обговорення такої можливості див. нижче, а зараз продовжимо обговорення у такому припущенні.

Повторення обчислень за формулою (2.11), так звані *ітерації*, зупинемо на якомусь *k*-му кроці. Якщо *k-1*-ше і *k*-те наближення вже досягли граничного значення з точністю ε (тобто відрізняються від граничного на ε), то між собою вони відрізняються не більше як на 2ε. Звідси випливає практичний спосіб визначення, коли можна

зупиняти ітераційний процес (2.11) – відразу як різниця між послідовними наближеннями стане менше за допустиму похибку ε (саме ж ε визначають з міркувань практичної доцільності).

Але ж як оцінити "близкість" $x^{(k)}$ і $x^{(k-1)}$? Адже ж це вектори, а не числа! Є кілька способів визначення *відстані* між двома векторами $x^{(1)}$ і $x^{(2)}$, або кажуть ще *метрики*, і позначають символом ρ [33,35]. В перший спосіб обчислюємо суму відстаней між відповідними координатами векторів:

$$\rho_1(x^{(1)}, x^{(2)}) = \sum_{i=1}^{n} | x_i^{(2)} - x_i^{(1)} | . \tag{2.12}$$

У другий спосіб обчислюємо суму квадратів покоординатних відстаней:

$$\rho_2(x^{(1)}, x^{(2)}) = \sum_{i=1}^{n} (x_i^{(2)} - x_i^{(1)})^2 . \tag{2.13}$$

Отже, обираємо одну з цих метрик, і як тільки станеться $\rho(x^{(k)}, x^{(k-1)}) \le 2\varepsilon$, зупиняємо подальші ітерації.

Вище виходили з того, що "*послідовність наближень збігається*". А чи "зобов′язана" вона збігатись? Теорія показує [33,35], що послідовність наближень збігається не завжди. Проте, вона збігається обов′язково, якщо коефіцієнти матриці A_1 задовольняють умову

$$l = \max_{1 \le i \le n} \sum_{j=1}^{n} | \alpha'_{ij} | < 1 \qquad (2.14)$$

(сума елементів кожного рядка матриці A_1 менше одиниці), де $\alpha'_{ij} = -\frac{\alpha_{ij}}{\alpha_{ii}}$ [33].

Отже, застосування методу ітерацій повинно починатись з перевірки умови (2.14). В той же час, доведено, що будь-яку систему рівнянь завжди можна перетворити в еквівалентну систему з коефіцієнтами, які вже підкорюються умові збіжності (2.14), [36]. Як це робити практично – далі покажемо на прикладі.

Підсумуємо **алгоритм розв′язування системи лінійних алгебраїчних рівнянь** (2.1) методом ітерацій:

1. Перетворюємо систему (2.1) до вигляду (2.10) і обчислюємо $l = \sum_{j=1}^{n} | \alpha_{ij} |$.
2. Перевіряємо умову (2.14) $l < 1$. Якщо умова не виконується, повертаємось до етапу перетворення системи (2.1) до форми (2.10). Принципово відомо, що таке перетворення, яке призводить до матриці (2.10) і задовольняє умову (2.14), можливо.
3. Якщо матриця A_1, для якої $l < 1$, знайдена, то можна розпочинати ітеративний пошук розв′язку системи (2.1). Вважаємо, що встановлена допустима похибка розв′язку ε. Тоді обчислюємо

$\varepsilon_1 = \frac{1-l}{l}\varepsilon$ – допустиму похибку послідовних наближень, або приймаємо грубо $\varepsilon_1 = \frac{1}{2}\varepsilon$.

4. Обираємо початкове наближення $x_0 = \{\beta_1, \beta_2, ..., \beta_n\}$. Відомо, що навіть "погане" наближення не впливає на принципову збіжність процесу ітерацій, але швидкість збіжності може значно зменшитись, тобто кількість потрібних ітерацій зросте.
5. Обчислюємо наступне наближення за формулою $x_1 = A_1 x_0 + b_1$.
6. Оцінюємо "близкість" двох послідовних наближень x_0 і x_1 за допомогою норми (2.12).
7. Перевіряємо умову $\rho_1(x_0, x_1) \le \varepsilon_1$. Якщо вона не виконується – повторюємо кроки 4 – 7.
8. Якщо ж умова виконується – розв′язок отримано!

Приклад 2.2. Розв'язування СЛАР методом ітерацій в MATLAB.

$$\begin{aligned} &4x_1 + 2{,}4x_2 - 0{,}8x_3 + x_4 = 9, \\ &x_1 + 3x_2 - 1{,}5x_3 - x_4 = 8, \\ &0{,}4x_1 - 0{,}8x_2 + 4x_3 + 2x_4 = 20, \\ &2x_1 - x_2 - +2x_3 - 3x_4 = 7. \end{aligned} \qquad (2.15)$$

Перш ніж застосовувати метод ітерацій, дану систему треба перетворити до вигляду (2.10), але так, аби коефіцієнти матриці A_1 задовольняли умову збіжності (2.14). Задля цього спочатку за допомогою еквівалентних перетворень (додавання до якогось рівняння іншого, попередньо помноженого на якийсь множник) отримуємо систему вигляду $A' x = h'$, де діагональні елементи матриці A' перебільшують суму модулів інших елементів.

Домовимось про точність обчислень. Виходячи з коефіцієнтів заданої системи (2.15), під час їхнього отримання зберігали перший знак після коми, округлюючи другий знак. Тому, і під час даних обчислень достатньо зберігати два знаки після коми. На всяк випадок, зберігаємо три знаки, тобто точність обчислень покладено $\varepsilon = 0{,}001$.

Дану систему з коефіцієнтом $a_{11} = 4$ перетворимо наступним чином. У третьому рівнянні модуль коефіцієнта перед третім невідомим x_3 є 4 і перевищує суму модулів інших коефіцієнтів $|0{,}4| + |-0{,}8| + |2| = 3{,}2$; це рівняння лишаємо незмінним. Замість першого рівняння візьмемо суму першого з четвертим: тоді модуль коефіцієнта перед першим невідомим x_1, що дорівнюватиме 6, стане перевищувати суму модулів інших коефіцієнтів $|1{,}4| + |1{,}2| + |-2| = 4{,}6$. Замість другого візьмемо подвоєне друге мінус четверте; коефіцієнти щойно отриманого рівняння (помічаємо їх штрихом) такі, що виконується умова $|a'_{21}| + |a'_{23}| + |a'_{24}| = |0| + |-4| + |1| = 5 < |a'_{22}| = 7$. Замість четвертого рівняння візьмемо різницю між четвертим і третім; тепер четвертий коефіцієнт $a'_{44} = -5$ перевищує за модулем суми модулів інших, тобто $|a'_{41}| + |a'_{42}| + |a'_{43}| = |1{,}6| + |0{,}8| + |2| = 3{,}8$. Отримали в результаті таку систему рівнянь

$$\begin{aligned}
&(\text{I}) + (\text{IY}) && 6x_1 + 1{,}4x_2 + 1{,}2x_3 - 2x_4 = 16,\\
&2(\text{II}) - (\text{IY}) && 7x_2 - 5x_3 + x_4 = 9,\\
&(\text{III}) && 0{,}4x_1 - 0{,}8x_2 + 4x_3 + 2x_4 = 20,\\
&(\text{IY}) - (\text{III}) && 1.6x_1 - 0.2x_2 - 2x_3 - 5x_4 = -13
\end{aligned} \quad , \tag{2.15'}$$

де діагональні коефіцієнти домінують над сумою абсолютних величин інших. Нова система (2.15′) еквівалентна заданій системі (2.15), бо із способу її отримання з системи (2.15), колонки ліворуч, видно, що використані всі рівняння з системи (2.15) і жодне з

рівнянь системи (2.15′) не є наслідком інших. Тепер кожне рівняння № *i* розв′язуємо відносно x_i, *i=1,2,3* і *4*. Маємо систему:

$$
\begin{aligned}
x_1 &= -0{,}233\,x_2 - 0{,}2\,x_3 + 0{,}333\,x_4 + 2{,}667,\\
x_2 &= 0{,}714\,x_3 - 0{,}143\,x_4 - 0{,}143,\\
x_3 &= -0{,}1\,x_1 + 0{,}2\,x_2 - 0{,}5\,x_4 + 5,\\
x_4 &= 0{,}32\,x_1 - 0{,}04\,x_2 - 0{,}4\,x_3 + 2{,}6.
\end{aligned}
$$

Це і є та остаточна система, яка підлягає розв′язку методом ітерацій. Її можна записати у вигляді (2.10) з матрицею і вектором

$$
A' = \begin{pmatrix} 0 & -0.233 & -0.2 & 0.333 \\ 0 & 0 & 0.714 & -0.143 \\ -0.1 & 0.2 & 0 & -0.5 \\ 0.32 & -0.04 & -0.4 & 0 \end{pmatrix}, \qquad b' = \begin{pmatrix} 2.667 \\ 1.286 \\ 5 \\ 2.6 \end{pmatrix}.
$$

Для MATLAB даємо їм імена *A1* і *b1* і вводимо:

```
>> a1=[0  -0.233  -.2  .333] ,  a2=[0,  0,  .714, -.143] , ...
   a3=[-.1, .2, 0, -.5] ,          a4=[.32, -.04, -.4, 0] , ...
   b1=[2.667; 1.286; 5; 2.6] ,     A1=[a1; a2; a3; a4]  .
```

Тепер можна проводити обчислення. Спочатку виконуємо команди

N_iterations=0 і *x0=[0; 0; 0; 0]*.

Перша з них потрібна для підрахунку кількості ітерацій, а друга – задає початкове значення розв′язку, $x_1 = 0;\ x_2 = 0;\ x_3 = 0;\ x_4 = 0$.

Далі багатократно виконуємо складну команду

```
N_iterations=N_iterations+1, xi=A1*x0+b1,
distance=norm(x0 - xi,1), x0=xi                    (2.16)
```

Її виконання складається з виконання чотирьох окремих команд. Перша команда рахує кількість ітерацій, бо *N_iterations* приймає значення *1, 2, 3, 4* і т.д. Друга команда *xi=A1*x0+b1* обчислює чергове наближення розв′язку *xi*, бо кожного разу четвертою командою *x0=xi* воно передається змінній *x0*. А третя команда обчислює норму різниці між попереднім *x0* і щойно отриманним *xi*

наближеннями. Скопіювавши команду (2.16) за допомогою <*Ctrl+C*> і поставивши її у командний рядочок знов (за допомогою <*Ctrl+V* >), можемо багато разів виконувати на MATLAB потрібні обчислення. Слідкуємо за різницею між двома послідовними наближеннями *distance*, і зупиняємо обчислення, коли *distance* $\leq \varepsilon$. Дійсно, тепер чергове наближення виявляється досить близьким до точного розв′язку (2.15). Для цього виявилося достатньою кількість ітерацій *N_iterations=30.*

2.2. РОЗВ'ЯЗУВАННЯ ТРАНСЦЕНДЕНТНИХ РІВНЯНЬ

Досить часто виникає потреба знати корені нелінійного рівняння загального виду

$$f(x) = 0\,. \qquad (2.17)$$

Коли функція $y = f(x)$ є поліномом, рівняння (2.17) називається ***алгебраїчним***; для аналізу і розв'язування таких рівнянь розроблені спеціальні методи, див. [32,33]. У випадку алгебраїчного рівняння теорія завжди може відповісти на питання про існування і кількість розв'язків рівняння (2.17).

Якщо функція $f(x)$ складається з поліномів, тригонометричних, степеневих та інших елементарних або неелементарних функцій, то рівняння (2.17) називають *трансцендентним*. Інколи ми навіть не знаємо аналітичний вираз функції $f(x)$ (отримуємо її, скажімо, з чисельного роз'вязку якоїсь іншої складної задачі в залежності від параметра x) – все одно можна ставити питання про наявність коренів рівняння (2.17) та їхнє знаходження з потрібною для тих чи інших практичних цілей точністю. Далі коротко викладено кілька найбільш поширених методів.

2.2.1. ГРАФІЧНИЙ МЕТОД, ВІДОКРЕМЛЕННЯ КОРЕНІВ

"Шкільний" метод визначення наявності кореня та його грубого знаходження за допомогою побудови графіка функції $y = f(x)$ є найбільш простим і переконливим. Його застосування полегшується ефективністю сучасних комп′ютерів і великої кількості простих для застосування графічних програм. Наведемо два приклади.

Приклад 2.3. Нехай потрібно знайти корінь трансцендентного рівняння

$$\cos x = kx, \tag{2.18}$$

де параметер k – дійсне число. У даному випадку функцію $f(x) = \cos x - kx$ доцільно розглядати складеною з двох функцій $\varphi(x) = \cos x$ і $\psi(x) = kx$. Рівняння (2.18), очевидно, еквівалентне рівнянню $\varphi(x) = \psi(x)$. Графіки кожної з цих функцій неодноразово будували у школі: це буде, відповідно, косинусоїда і пряма лінія через початок координат і з тангенсом кута нахилу k. Графіки показані на рис. 2.1, де пряма побудована для кількох значень k. Можна зробити висновок, що кількість розв′язків даного рівняння залежить від значення k, а саме:

а) при $k = 0{,}5$ маємо лише один корінь рівняння, що лежить між 0 і $\pi/2$ (можна навіть узагальнити, що один корінь буде для усіх k, таких, що $-\frac{\pi}{2} < k < \frac{\pi}{2}$);

б) при $k = 0{,}1$ рівняння має аж сім коренів, які знаходяться у інтервалах: $x_1 \in (0, \frac{\pi}{2})$, $x_2 \in (\pi, 2\pi)$, $x_3 \in (2\pi, 3\pi)$, $x_4 \in (-\pi, 0)$, $x_5 \in (-2\pi, -\pi)$, $x_6 \in (-3\pi, -2{,}5\pi)$ і $x_7 \in (-3{,}5\pi, -3\pi)$.

Рис. 2.1. **До графічного розв′язування рівняння** $\cos x = kx$ **для кількох значень** k**: крива 1 – косинусоїда; 2 – пряма для** $k = 0{,}5$**; 3 – пряма для** $k = 0{,}1$**; 4 – пряма для** $k = -0{,}3$

в) при $k=-0{,}3$ маємо лише три корені: $x_1 \in (-\frac{\pi}{2}, 0)$, $x_2 \in (\frac{\pi}{2}, \pi)$ і $x_3 \in (\pi, 1.5\pi)$.

Таким чином, у кожному випадку графічна побудова дозволяє принципово з′ясувати кількість коренів рівняння, що досліджується, і приблизне знаходження коренів, тобто вказати інтервал $a \le x \le b$, де знаходиться лише один корінь. Така процедура називається *локалізацією* коренів.

Приклад 2.4. Нехай потрібно розв′язати нелінійне рівняння [32, с.168]

$$2x^5 - 100x^2 + 2x - 1 = 0. \qquad (2.19)$$

Як у попередньому прикладі, можна задану функцію розкласти на степеневу $\varphi(x) = 2x^5$ і параболу $\psi(x) = 100x^2 - 2x + 1$, графіки яких також вивчали у школі, і шукати перетин двох кривих. Натомість сучасний комп′ютер легко може побудувати графік функції $y = 2x^5 - 100x^2 + 2x - 1$ безпосередньо (див. розділ 1.5):

*>> fplot('2*x^5-100*x^2+2*x-1',[-2 4]); legend('y=2*x^5-100*x^2+2*x-1')*

і нам лишається дослідити його перетин з віссю *OX*: $y = 0$. На підставі графіка, показаного на рис. 2.2, робимо висновок, що один з коренів рівняння (2.19) лежить, без сумніву, на проміжку $x \in [3,4]$. Навіть можна оцінити більш точно, що корінь лежить між 3,5 і 3,8. Окрім того, перетин кривою осі *OX* можливий десь в околі $x = 0$, однак великий масштаб графіка не дозволяє зробити більш впевнений висновок.

Таким чином, для уточнення коренів треба дослідити дану функцію в більш вузьких діапазонах $3{,}5 < x < 3{,}8$ і, скажімо, $-0{,}1 < x < 0{,}1$:

fplot('*2*x^5-100*x^2+2*x-1*',[*-3,5 3,8*]); *legend*('*y=2*x^5-100*x^2+2*x-1*')

fplot('*2*x^5-100*x^2+2*x-1*',[*-.1 .1*]); *legend*('*y=2*x^5-100*x^2+2*x-1*')

Більш точно знайти корінь в першому діапазоні, а саме $x \approx 3{,}656$, і заключити, що в другому діапазоні кореня насправді немає дозволяє рис. 2.3. Слід звернути увагу на те, що існування коренів даного рівняння за межами дослідженого інтервалу $-2 < x < 4$ тут лишилося нез′ясованим. Треба або дослідити питання на інших інтервалах, або врахувати (в даному випадку степеневого рівняння) відомі теореми про кількість та характер розташування коренів поліномів [32].

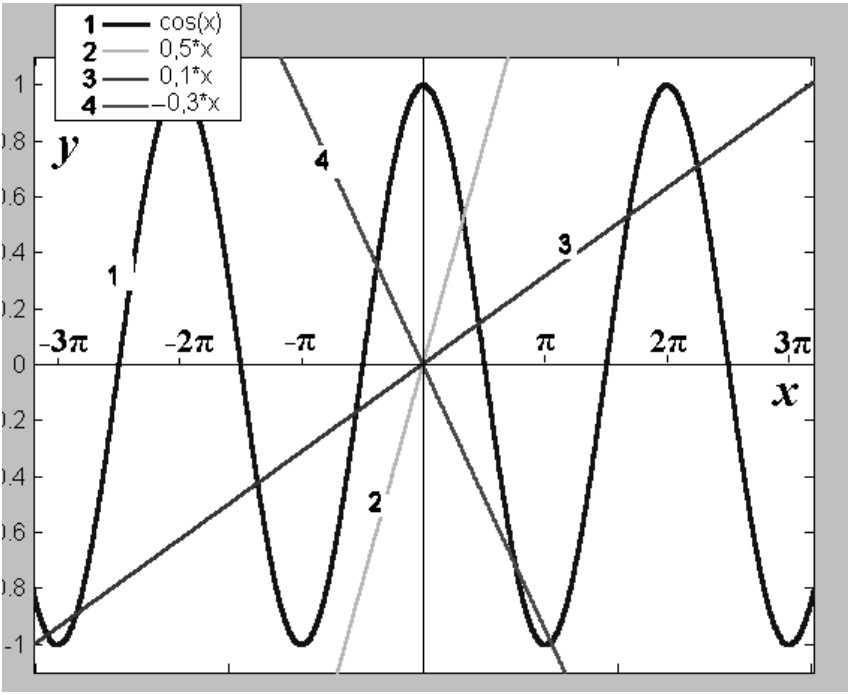
Як будувати графіки функцій на персональному комп′ютері –обговорювалось в розділі 1. В такий спосіб неможливо, однак, знаходити комплексні корені.

Побудову графіка часто використовують для того, аби лише грубо встановити, чи існує корінь рівняння на певному проміжку $[x_{\text{п}}, x_{\text{к}}]$. Таку процедуру називають *відокремленням кореня*. Приклад наведено вище. Важливо відмітити, що, оскільки *неперервна* функція $f(x)$ з рівняння (2.17) має різні знаки на кінцях інтервалу $x_{\text{п}}$ і $x_{\text{к}}$ (індекси обрано від слів "Початок" і "Кінець"), то критерієм існування на ньому коренів може служити умова

$$f(x_{\text{п}}) \cdot f(x_{\text{к}}) < 0. \qquad (2.20)$$

Її зручно використовувати в комп'ютерних програмах.

2.2.2. МЕТОД ПОДІЛУ ВІДРІЗКА НАВПІЛ



Не завжди можна скористатись нашим зором, і треба, буває, "доручити" програмі "самій" обрати рішення щодо наявності кореня рівняння. Для цього обчислювач повинен "алгоритмізувати" процес обчислень.

На першому етапі треба *відокремити корінь* рівняння (2.17), тобто знайти такі кінці інтервалу $X_{\text{п}}$ і $X_{\text{к}}$, для яких виконується умова (2.20). В багатьох випадках їх можна знайти графічним методом, як у попередньому розділі. Бувають випадки, коли кінці

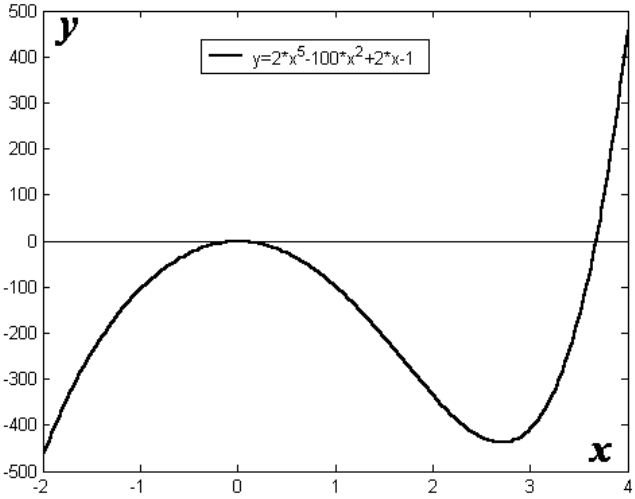


Рис. 2.2. Графічний пошук нулів рівняння (2.19): один корінь між 3 і 4; не ясно, чи перетинає крива вісь абсцис в околиці $x = 0$

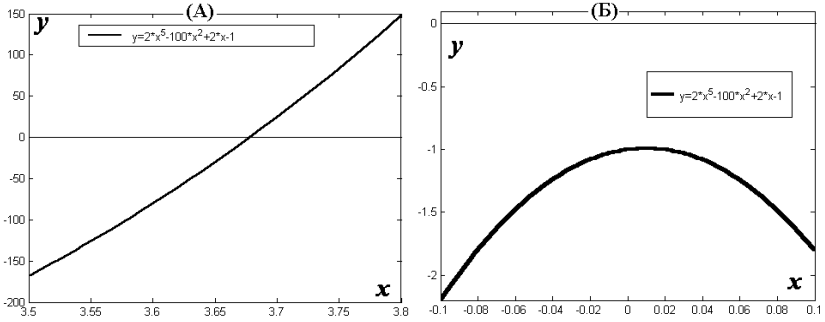


Рис. 2.3. Уточнення коренів рівняння (2.19) в діапазонах (А) $3{,}5 < x < 3{,}8$ і (Б) $-0{,}1 < x < 0{,}1$; показана також вісь *OX*. У другому випадку крива не перетинає вісь абсцис $y = 0$ – кореня на даному інтервалі немає

інтервалу $x_{\text{п}}$ і $x_{\text{к}}$ для певної задачі відомі теоретично. Часто, для неперервної функції $f(x)$, допомагає наступний алгоритм.

Крок 1, *початковий*: довільно обирають *початкове наближення* x_0 і *крок пошуку* $\Delta x = \Delta x_0$ і обчислюють функцію в точках x_0 і $x_1 = x_0 + \Delta x$. *Крок 2*, *визначення напрямку пошуку*: якщо $|f(x_1)| < |f(x_0)|$ (наступне значення ближче до нуля, ніж попереднє) – напрямок вірний, знак Δx не змінюють; якщо ж $|f(x_1)| > |f(x_0)|$ (ще далі віддалилися від нуля) – напрямок невірний, змінюють його на протилежний, тобто приймають $\Delta x = -\Delta x$. *Крок 3*, *локалізація кореня*: обчислюють послідовно $x_2 = x_1 + \Delta x$ і $f(x_2)$, $x_3 = x_2 + \Delta x$ і $f(x_3)$, …, $x_n = x_{n-1} + \Delta x$ і $f(x_n)$, кожного разу перевіряючи умову $f(x_{n-1}) \cdot f(x_n) < 0$, $n = 1,2,...$ Як тільки це сталося – щойно отримані точки і дають кінці інтервалу $x_{\text{п}} = x_{n-1}$ і $x_{\text{к}} = x_n$, на якому знаходиться корінь рівняння (2.17). Можна переходити до наступного етапу.

Треба мати на увазі, що описаний алгоритм не завжди дозволяє локалізувати корінь. Якщо на кроці 2 маємо рівність $f(x_1) = f(x_0)$, то визначитись з напрямком пошуку ще не можна. Тоді треба зробити ще крок у тому ж напрямку $x_2 = x_1 + \Delta x$, порівняти $f(x_2)$ і $f(x_1)$ і знову спробувати визначитись. На кроці 3 для довільної функції $f(x)$ не гарантовано, що умова (2.20) відбудеться – це означає, що у програмі треба передбачити "*аварійну зупинку*" для запобігання її нескінченної роботи.

На другому етапі, коли $x_{\text{п}}$ і $x_{\text{к}}$ вже відомі, треба *уточнити корінь*, тобто знайти його з потрібною точністю. Найбільш простим і надійним способом розв′язування нелінійних рівнянь вважають *метод ділення відрізка навпіл*. Наступні наближення обирають тепер за правилом

$$\Delta x = \Delta x / 2, \qquad x' = x_{\text{п}} + \Delta x. \tag{2.21}$$

Знов перевіряють умову (2.20). Оскільки для кінців інтервалу $x_п$ і $x_к$ вона виконана, то обов'язково буде виконуватись одне із співвідношень,

$$\text{або} \quad f(x_п) \cdot f(x') < 0, \qquad \text{або} \quad f(x') \cdot f(x_к) < 0. \qquad (2.22)$$

Перше з них означає, що корінь знаходиться у лівій половині інтервалу, тобто x' можна вважати тепер за кінець уточненого інтервалу, тобто покласти $x_к = x'$. У другому випадку корінь лежить у правій половині інтервалу, між новим початком $x_п = x'$ і кінцем $x_к$. Знаємо, тобто, новий, вдвічі вужчий інтервал $[x_п, x_к]$, де знаходиться корінь рівняння.

Уточнення кореня за формулою ділення навпіл (2.21) і логічними умовами (2.22) продовжуємо доти, поки довжина інтервалу Δx не стане меншою за потрібну точність ε. Виконання останньої умови означатиме, що корінь рівняння (2.19) з точністю ε знайдено.

Треба зазначити, що під час перевірки умов (2.22) може статися і "точне попадання в корінь" $f(x_п) \cdot f(x') = 0$. На цей доволі рідкісний випадок треба передбачити дострокове припинення обчислень, бо x' є корінь.

Графічне пояснення до методу поділу відрізка навпіл дано на рис. 2.4. Цей метод з кожним кроком уточнення дає удвічі краще наближення до кореня, що вважається доволі повільним. Деякі з наступних методів мають швидшу збіжність.

2.2.3. МЕТОД ІТЕРАЦІЙ РОЗВ'ЯЗУВАННЯ НЕЛІНІЙНИХ РІВНЯНЬ

Цей метод називають *методом простих ітерацій* або *послідовних наближень*. Спочатку рівняння (2.17) переписують у вигляді

$$x = \varphi(x), \tag{2.23}$$

де, наприклад, $\varphi(x) = f(x) + x$. Очевидно, що рівняння (2.17) і (2.23) еквівалентні: якщо $x = x_*$ корінь одного з них, то він же є і коренем іншого. Останнє рівняння дозволяє, якщо обрати деяке *початкове наближення* x_0, створити послідовність наближень

$$x_1 = \varphi(x_0),\ x_2 = \varphi(x_1), \ldots,\ x_n = \varphi(x_{n-1}), \ldots \tag{2.24}$$

$n = 1,2,3,\ldots$. Має місце твердження, яке легко довести [32,36,37]:

Теорема. Якщо ітераційний процес (2.24) збігається, то наближення збігаються саме до кореня $x = x_*$.

Якщо ітераційний процес збігається, то корінь, таким чином, існує. Але існування кореня ще не гарантує збіжності послідовних наближень (або – ітераційного процесу) (2.24). Геометричне пояснення до даної проблеми дозволяє дати рис. 2.5.

На рис. 2.5,А зображено ситуацію, коли ітераційний процес збігається. На графіку побудовано криву $y = \varphi(x)$ і пряму $y = x$. Точка їхнього перетину дає корінь $x = x_*$ рівняння (2.23), який треба знайти. Припустимо, обрано початкове наближення $x = x_0$. На підставі формул (2.24) обчислюємо $y_0 = \varphi(x_0)$ і такий само відрізок $x_1 = y_0$ відкладаємо на осі абсцис – це можна зробити відображенням від прямої $y = x$, як показано стрілками. Отримане значення $x_1 = \varphi(x_0)$ знову підставляємо у формулу (2.24) і аналогічним чином отримуємо значення $y_1 = \varphi(x_1)$, $x_2 = y_1 = \varphi(x_1)$, потім $x_3 = \varphi(x_2)$ і т.д. Значення x_1, x_2, x_3, x_4, ..., як бачимо, поступово наближуються до кореня x_*, який шукаємо. Тобто, одне з чергових наближень вже являє собою корінь з певною точністю. Наближення до кореня мало монотонний характер, всі x_i були ліворуч від кореня. Можна упевнитись на рис. 2.5,А, що якщо початкове наближення обрати

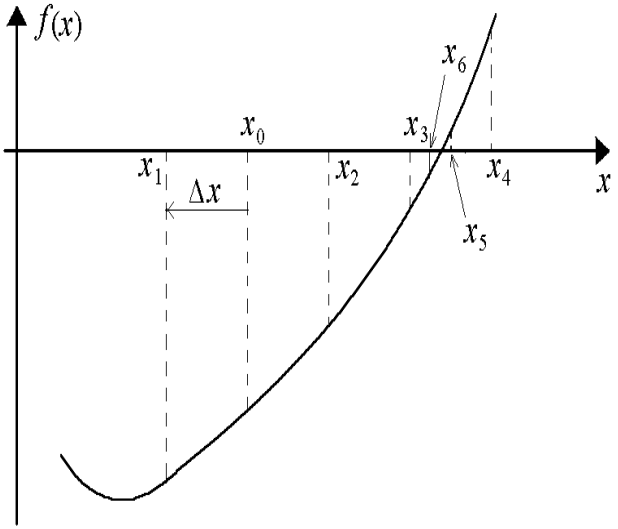


Рис. 2.4. Пояснення до методу поділу відрізка навпіл: точки $x_0 = x_п$ і x_1 -- визначення напрямку; $x_2, \ldots, x_4$ – локалізація кореня на першому етапі; x_5, x_6 і т.д. – уточнення кореня на другому етапі за формулами (2.22)

справа від x_*, то ітераційний процес все одно збігається, але x_i наближується до x_* справа наліво. У цьому випадку маємо *односторонню збіжність*.

Рис. 2.5,Б показує, що збіжність наближень x_1, x_2, x_3, x_4, . . . може мати місце і з двох боків – це так звана *двостороння збіжність*. Однак, збіжність може й не мати місце – такий випадок ілюструє рис. 2.5,В: початкове наближення x_0 обрано досить близьким до x_*; проте, наступні наближення x_2, x_3, x_4 і т.д. виявляються все далі і далі від x_*.

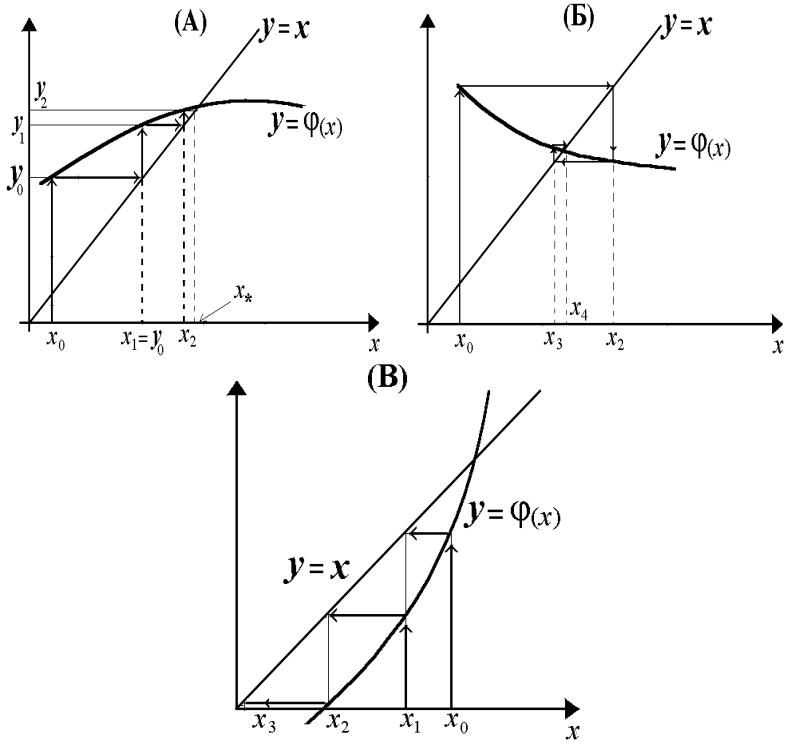


Рис. 2.5. Геометричне пояснення до збіжності односторонньої (А), двосторонньої (Б) та розбіжності (В) ітерацій (2.24) до розв'язку x_*

Від чого залежить збіжність простих ітерацій за формулою (2.24)? Наведений рис. 2.5 підказує, що має значення нахил кривої $y = \varphi(x)$ поблизу кореня $x = x_*$. Дійсно, легко встановити такий факт [33,35,36]:

Теорема. Для збіжності ітераційного процесу (2.24) достатньо, коли

$$|\varphi'(x)| < 1 \qquad (2.25)$$

для всіх x поблизу кореня†.

† Насправді формулювання теореми трохи складніше, [32,34,35].

На практиці умову (2.25) не завжди можна перевірити, бо корінь x_* невідомий. Інколи вдається переконатись, що умова (2.25) виконується на деякому відрізку $x \in [a, b]$, до якого корінь *має належати*. Для певних математичних моделей виконання умови (2.25) довести вдається. В таких випадках збіжність гарантовано. Але у більшості практичних ситуацій обчислювачі діють за гаслом "Спробувати, а потім перевірити!".

Перехід від рівняння (2.17) до рівняння у формі (2.23) можна здійснити багатьма способами. Часто застосовують такий практичний прийом, що дозволяє одночасно задовольнити достатню умову збіжності (2.25). Якщо має місце рівняння (2.17), то рівняння

$$x = x + \lambda\, f(x)$$

йому еквівалентне для будь-якого $\lambda \neq 0$. Коефіцієнт λ оберемо так, щоб задовольнити умову (2.25). Для функції $\varphi(x) = x + \lambda\, f(x)$ легко знайти похідну: $\varphi'(x) = 1 + \lambda\, f'(x)$. Припустимо, що корінь рівняння (2.17) має бути на відрізку $[a, b]$, на якому похідна $f'(x)$ додатня (тобто функція $f(x)$ монотонно зростає на $[a, b]$), змінюючись від найменшого значення $m > 0$ до найбільшого M. Можна перевірити [32,34], що умова збіжності задоволена, коли коефіцієнт λ обрати з інтервалу $(-2/m, 0)$. Таким чином, метод простих ітерацій дозволяє розв′язувати широкий клас рівнянь. Інша справа – швидкість збіжності. Не завжди вона задовільна, і це змушує розробляти й інші методи.

Приклад 2.5. Розв′язати рівняння $2^x + 5x - 3 = 0$.

Розв′язування. Таке трансцендентне рівняння не може бути розв’язане аналітично. Перепишемо його у вигляді $x = \frac{1}{5}(3 - 2^x)$. Це означає, що функція $\varphi(x)$ з рівняння (2.23) має вигляд

$\varphi(x) = \frac{1}{5}(3 - 2^x)$. Її похідною є функція $\varphi'(x) = -\frac{2^x}{5}\ln 2$. Оскільки функція $y = 2^x$ зростаюча, то на усьому проміжку $x \in (-\infty, 2]$ можна стверджувати, що

$$|\varphi'(x)| = \frac{2^x}{5}\ln 2 \le \frac{2^2}{5}\ln 2 = \frac{4\ln 2}{5} \approx 0{,}55 < 1.$$

Таким чином, для кожного x з області $-\infty < x < 2$ достатня умова збіжності (2.25) виконана. Можна обрати довільне початкове наближення X_0 з цієї області і проводити ітерації. Обчислення за допомогою MATLAB можна провести таким чином.

Спершу виконуємо команду і отримуємо

```
>> x0=3; i=0; x1=(3-2^x0)/5, R=x0-x1, x0=x1; i=i+1
    x1 =  -1     R =  4       i =  1
```

Далі викликаємо ту ж саму команду, але редагуємо її

```
>> x1=(3-2^x0)/5, R=x0-x1, x0=x1; i=i+1
```

Тепер змінна ***i*** починає працювати як лічильник циклів, а змінна *R* видає різницю між послідовними наближеннями. Кожний цикл закінчується тим, що щойно знайдене наближення *x1* передається змінній *x0* і розрахунок продовжується. Легко пересвідчитись, що різниця *R* стає менше за *0,001* за *7* ітерацій. Якщо така точність вас задовольняє – розрахунок можна закінчити; наближеним значенням кореня буде *x1= 0,3458*.

(Зверніть увагу: ітерації збігаються навіть для $x_0 = 20$ (за *7* ітерацій!), який виходить далеко із зазначеного діапазону і для якого умова (2.25) не виконується. Чи це не протирічить теоремі? **Відповідь:** умова (2.25) є *достатньою*, тобто гарантує збіжність, коли вона має місце. Однак, як бачимо, із збіжності ця умова зовсім не випливає, тобто – вона не є *необхідною*!)

2.2.4. МЕТОД ХОРД РОЗВ′ЯЗУВАННЯ РІВНЯНЬ

Попередній рис. 2.5 підказує зовсім інший спосіб поступового (ітеративного) наближення до кореня рівняння, ідею якого показано на рис. 2.6. Корінь $x = x_*$ рівняння (2.17) – це така точка на осі Ox, де крива $y = f(x)$ її перетинає. Нехай відокремленням кореня (шляхом геометричної побудови або алгоритмом, який описано в розділі 2.2.1) попередньо встановлено, що він лежить між $x_0 = a$ і $x_к = b$. Хорда, яка сполучає точки $A = (a, f(a))$ і $B = (b, f(b))$ на кривій, грубо наближає дану криву і має рівняння $\frac{y - f(a)}{x - a} = \frac{f(b) - f(a)}{b - a}$. Ця хорда перетинає вісь Ox в точці

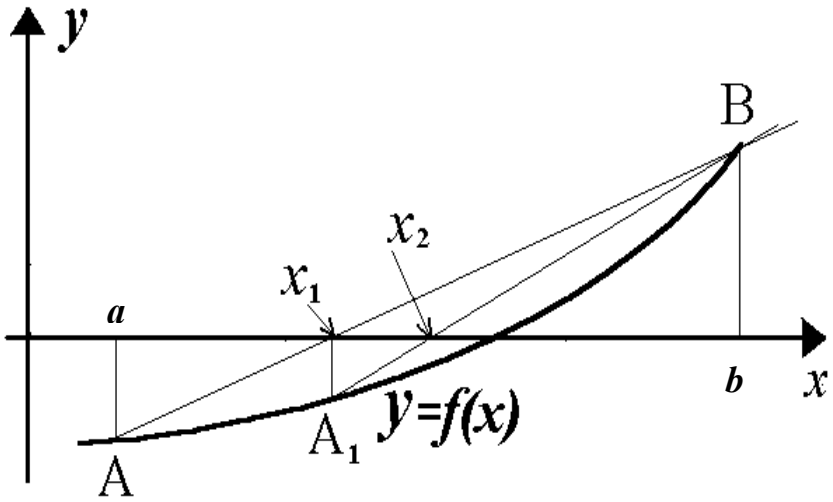


Рис. 2.6. Ітеративний пошук кореня за методом хорд

$$x_1 = a - \frac{f(a)}{f(b) - f(a)}(b - a).$$

Отримане значення вже ближче до кореня, аніж $x_0 = a$, і його приймаємо за нове наближення. Для наступного наближення будуємо нову хорду, що сполучає A_1 і B, і обчислюємо її точку перетину з віссю, і так далі, за формулою

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f(b) - f(x_{n-1})}(b - x_{n-1}), \quad n = 2,3,4,... \tag{2.26}$$

Такий метод отримав назву *метод хорд*. У ньому один з кінців інтервалу, на якому розшукується корінь, є нерухомий – у випадку рис. 2.6 таким є кінець $x = b$. Послідовність наближень $\{x_1, x_2, x_3, ...\}$ у методі хорд завжди монотонна. Буде ця послідовність зростаючою або спадною, який саме кінець інтервалу обирати за нерухомий – залежить від властивостей функції $y = f(x)$. Правило дає

Теорема [32,37]**.** Нехай на відрізку $[a, b]$ функція $y = f(x)$ неперервна разом із своїми похідними $f'(x)$ і $f''(x)$, причому $f(a)\, f(b) < 0$, а похідні зберігають сталі знаки. У формулі (2.26) за нерухомий беремо той кінець, в якому знак функції збігається із знаком другої похідної, $f(b)\, f''(b) > 0$. За таких умов послідовність наближень за формулою (2.26) збігається до кореня рівняння $f(x) = 0$. Похибку наближень оцінює формула

$$| x_* - x_n | \le \frac{M_1 - m_1}{m_1} | x_n - x_{n-1} |, \tag{2.27}$$

де M_1 і m_1 – найбільше і найменше значення першої похідної $f'(x)$ на проміжку $[a, b]$.

Формула (2.27) дозволяє зупинити ітераційний процес, коли $| x_n - x_{n-1} | \le \varepsilon$.

Існують й інші варіанти ітераційного знаходження коренів нелінійних рівнянь, з якими студент може познайомитись у книжках [32,35-38].

2.3. ПІДБІР ЕМПІРИЧНИХ ФУНКЦІЙ МЕТОДОМ НАЙМЕНШИХ КВАДРАТІВ

Проблема пошуку емпіричного рівняння чи не найчастіш зустрічається в практиці інженера або наукового працівника-експериментатора в будь-якій галузі науки і техніки. Йдеться ось про що.

Нехай в експерименті, науковому або інженерному, встановлено залежність між вхідним параметром x (аргументом) і вихідним параметром y (функцією). Результат такого експерименту фіксують у вигляді таблиці 2.1 з даними N дослідів:

Таблиця 2.1: Результати дослідів

Номер досліду	Значення x	Значення y
1	x_1	y_1
2	x_2	y_2
…	…	…
N	x_N	y_N

Ці ж дані можна представити у графічній формі, скажімо – значками o , як показано на рис. 2.7. Якщо послідовні експериментальні точки $\{x_i, y_i\}$, $i = 1,..., N$ з'єднати прямими, то отримаємо ламану лінію, яка буде тим "гірше", чим більше похибка експериментальних вимірів. Область використання таких даних обмежується лише умовами даного експерименту. Однак, вона буде значно ширшою, якщо знайти *аналітичну формулу* $y = f(x)$, яка наближує дані експериментальні точки. Тоді легко робити *інтерполяцію* даних експерименту (тобто наближено визначати значення функції y **між** сусідніми значеннями аргументу x_k

і x_{k+1}) і навіть *екстраполяцію* (прогноз **за межі** експерименту, тобто для $x < x_1$ або $x > x_N$). Знаходження такої ***емпіричної функції*** і є задачею даного розділу, а також розділу 4.7. Додамо, що задачі такого роду можуть виникнути і в теоретичній роботі, коли, скажімо, результати складного розрахунку треба подати наближеною, але простою і зручною в користуванні формулою. Інший випадок: вигляд формули встановлено теорією, як це було, наприклад, із законом всесвітнього тяжіння, але значення коефіцієнтів невідомі і мають бути знайдені зі спеціальних експериментів.

Викладемо ***метод найменших квадратів*** (***МНК***) розв'язання такої задачі. Хоча цей красивий метод запропоновано ще Гауссом і Лежандром в 19-му сторіччі [33,36], він плідно використовується у багатьох розділах математики і, безумовно, має складати науковий багаж сучасного фахівця.

На першому етапі побудови емпіричної формули треба

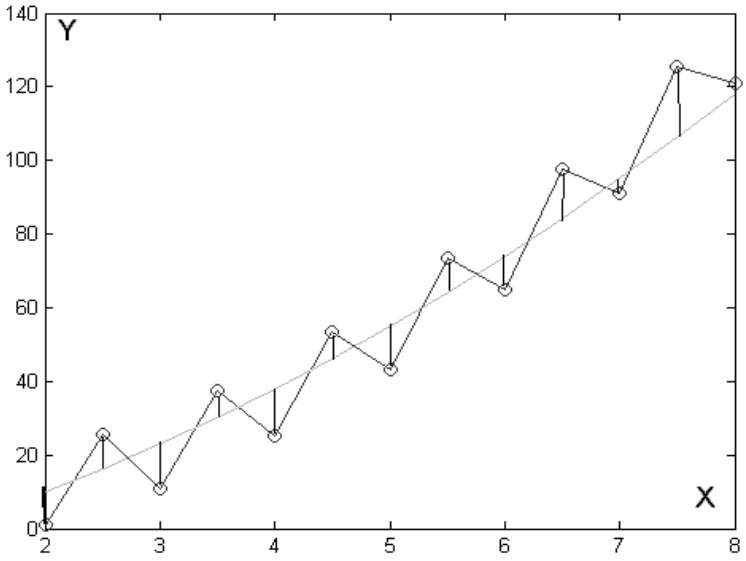


Рис. 2.7. Емпіричні дані, позначені значком o і з′єднані ламаною, разом із кривою, що їх наближує

встановити загальний вигляд цієї формули. Для цього треба поєднати візуальний вигляд вже побудованої ламаної з вашими знаннями про функції її їх графіки. Якщо ламана, що на графіку сполучає експериментальні точки, близька до прямої лінії, то аналітичну залежність слід шукати у вигляді лінійної функції $y = Ax + B$ з деякими коефіцієнтами A і B. Якщо ламана зростає або спадає приблизно як парабола $y = \pm Ax^2 + B$, то останню функцію і слід взяти як емпіричне рівняння. Більш загально, буває корисним і зручним дослідити степеневу функцію $y = \pm Ax^{\alpha} + B$ або многочлен n-го степеня $y = a_1 x^n + a_2 x^{n-1} + a_3 x^{n-2} + ... + a_{n-1} x + a_n$, які мають параметри (коефіцієнти) відповідно A, B, α і a_1, a_n, …, a_n. Якщо функція дуже швидко необмежено зростає – використовують показникову функцію $y = Ae^{\alpha x}$ (про показникову і логарифмічну функцію див. далі). Якщо візуально видно певну періодичність функції – доцільно звертатись до тригонометричних функцій $y = A\sin(\omega x)$, $y = B\cos(\omega x)$, або до їхніх сум. У "важких випадках" радимо переглянути довідник [31].

На другому етапі слід знайти "найкращі" значення параметрів вибраної емпіричної функції. Нехай обрана функція має вигляд

$$y = f(x; A_1, A_2, ..., A_M), \qquad (2.28)$$

тобто залежить, крім аргументу x, від M параметрів $A_1, A_2, ..., A_M$. Буває, що кількість параметрів співпадає із кількістю експериментальних даних, $M = N$. Тоді, підставляючи у формулу (2.28) пари експеріментальних значень, маємо N рівнянь з N невідомими

$$y_i = f(x_i; A_1, A_2, ..., A_N), \quad i = 1,2,..., N.$$

Існують різні способи *точно* провести криву (2.28) через експериментальні точки, див. підручники [32,36]. Тут викладемо принципово інший метод.

Дійсно, якщо експериментальні дані неминуче включають деяку похибку вимірювань, навіщо ж проводити криву (2.28) точно через відповідні точки?

Нехай крива (2.28) при певному виборі значень параметрів $A_1, A_2, ..., A_M$ і при i-му значенні аргументу x_i приймає значення функції $\tilde{y}_i$ (з "тільдою"), яке відрізняються від експериментального значення y_i, і так – для кожного з вимірювань $i = 1, ..., N$. Тоді сума всіх відхилень

$$\sum_{i=1}^{N} (\tilde{y}_i - y_i)^2 \overset{Def}{=} S(A_1, A_2, ..., A_M) \qquad (2.29)$$

відрізняється від нуля. Така сума називається *нев'язкою*[*)] і є функцією коефіцієнтів $A_1, A_2, ..., A_M$. Природньо обирати такі значення цих коефіцієнтів, за яких нев'язка має найменше значення. Геометрично це означає, що крива (2.28) проходить посередині між експериментальними точками. В цьому і полягає ідея **методу найменших квадратів**. Знайти мінімум функції (2.29) можна шляхом диференціювання нев'язки $S(A_1, A_2, ..., A_M)$ по кожному з параметрів і отримання M рівнянь з M невідомими.

Із загальною реалізацією методу найменших квадратів можна ознайомитись у книгах [33,35-38]. Тут обмежимось лише певними випадками.

1. Лінійне емпіричне рівняння.

Нехай залежність між аргументом і функцією шукаємо у вигляді лінійної функції

$$y = ax + b.$$

Тоді, нев'язка (2.29) має такий вигляд:

$$S(a, b) = \sum_{i=1}^{N} (y_i - ax_i - b)^2 .$$

[*)] Для користування MatLab корисно знати англійський еквівалент цього терміну – *residual*

Прирівняємо до нуля частинні похідні цієї функції по кожному з аргументів a і b, аби знайти мінімум нев′язки. Маємо два рівняння відносно a і b:

$$\frac{\partial S}{\partial a} = 2\sum_{i=1}^{N}(y_i - a\,x_i - b)(-x_i) = 0,$$

$$\frac{\partial S}{\partial b} = 2\sum_{i=1}^{N}(y_i - a\,x_i - b)(-1) = 0.$$

Після простих спрощень і перетворень приходимо до лінійної системи двох рівнянь

$$a\sum_{i=1}^{N} x_i^2 + b\sum_{i=1}^{N} x_i = \sum_{i=1}^{N} x_i\,y_i\ ,$$

$$a\sum_{i=1}^{N} x_i + nb = \sum_{i=1}^{N} y_i\ .$$

Можна довести строго, що така система **завжди має**, і притому **єдиний розв′язок**, якщо кількість вимірів $N>1$. Цей розв′язок легко знайти:

$$a = \frac{N\,S(x_i\,y_i) - S(x_i)\,S(y_i)}{N\,S(x_i^2) - S(x_i)^2},\quad b = \frac{S(y_i)\,S(x_i^2) - S(x_i)\,S(x_i\,y_i)}{N\,S(x_i^2) - S(x_i)^2},\qquad (2.30)$$

де позначено $S(x_i) = \sum_{i=1}^{N} x_i$ – сума перших степенів аргументів, $S(x_i^2) = \sum_{i=1}^{N} x_i^2$ – сума других степенів аргументів, $S(x_i\ y_i) = \sum_{i=1}^{N} x_i\ y_i$ – сума попарних добутків аргументів і функції і, нарешті, $S(y_i) = \sum_{i=1}^{N} y_i$.

Приклад 2.6. Нехай "експериментальні" дані подані таблицею 2.2 (власне, саме ці дані показано на рис. 2.7).

Таблиця 2.2. Результати умовного експерименту

x_i	2	2.5	3	3.5	4	4.5	5	5.5	6	6.5	7	7.5	8
y_i	1	25.5	11	37.5	25	53.5	43	73.5	65	97.5	91	125.5	121

Обчислюємо вказані вище суми: $S(x_i) = 65$; $S(y_i) = 770$; $S(x_i^2) = 370{,}5$ і $S(x_i\ y_i) = 4760$ (це можна обчислити "вручну", але легко робити в MATLAB командами на зразок*) *Sx=sum*(*x*), *Sx2=sum*(*x* **.^***2*), *Sxy=sum*(*x* **.*** *y*), якщо *x* і *y* – відповідні вектори даних). За формулами (2.30) маємо $a = 20$, $b = -40{,}77$. Таким чином, лінійним емпіричним рівнянням для даних табл.. 2.2 буде $y = 20x - 40{,}77$.

2. Емпірична формула у вигляді многочлена.

Так само легко знайти "найкращий" (у сенсі найменшої середньоквадратичної нев′язки) серед многочленів степеня $n < N$, що наближує задані емпіричні дані з N вимірювань. На цьому шляху отримаємо систему n лінійних рівнянь з n невідомими, яка завжди однозначно розв′язується. Покажемо це на прикладі емпіричного рівняння другого степеня $y = a x^2 + b x + c$.

При так обраному емпіричному рівнянні, функція нев′язки має вигляд

$$S(a,b,c) = \sum_{i=1}^{N} (y_i - a x_i^2 - b x - c)^2 .$$

Прирівнюючи до нуля частинні похідні $\frac{\partial S}{\partial a}$, $\frac{\partial S}{\partial b}$ і $\frac{\partial S}{\partial c}$, приходимо до системи трьох лінійних рівнянь з трьома невідомими:

$$a\sum_{i=1}^{N} x_i^4 + b\sum_{i=1}^{N} x_i^3 + c\sum_{i=1}^{N} x_i^2 = \sum_{i=1}^{N} x_i^2 y_i ,$$

*) про оператор MatLab′у *sum*(*a*) для вектора або матриці *a* див. розділ 1.4.

$$a\sum_{i=1}^{N} x_i^3 + b\sum_{i=1}^{N} x_i^2 + c\sum_{i=1}^{N} x_i = \sum_{i=1}^{N} x_i\, y_i \ ,$$

$$a\sum_{i=1}^{N} x_i^2 + b\sum_{i=1}^{N} x_i + Nc = \sum_{i=1}^{N} y_i \ .$$

Формула для розв′язку цієї системи має доволі громіздкий вигляд. Для знаходження чисельних значень коефіцієнтів a, b і c слід використати визначники, або операцію MATLAB

$$C=A\backslash b,$$

(див. розділ 4.1) де матриця $A = \begin{pmatrix} S(x_i^4) & S(x_i^3) & S(x_i^2) \\ S(x_i^3) & S(x_i^2) & S(x_i) \\ S(x_i^2) & S(x_i) & N \end{pmatrix}$ і вектор

$b = \begin{pmatrix} S(x_i^2\ y_i) \\ S(x_i\ y_i) \\ S(y_i) \end{pmatrix}$, а також введено додаткові позначення

$$S(x_i^4) = \sum_{i=1}^{N} x_i^4 \ , \quad S(x_i^3) = \sum_{i=1}^{N} x_i^3 \quad \text{і} \quad S(x_i^2\ y_i) = \sum_{i=1}^{N} x_i^2\ y_i \ . \quad (2.31)$$

Приклад 2.7. Нехай для тих самих "експериментальних" даних табл. 2.1 шукаємо емпіричне рівняння у вигляді квадратичної функції. Тоді, додатково до значень прикладу 2.6, маємо:

$$S(x_i^4) = 15\,234\,, \quad S(x_i^3) = 2\,307{,}5 \quad \text{і} \quad S(x_i^2\ y_i) = 31580\,.$$

Визначники системи рівнянь дорівнюють

$$\Delta_0 = \begin{vmatrix} S(x_i^4) & S(x_i^3) & S(x_i^2) \\ S(x_i^3) & S(x_i^2) & S(x_i) \\ S(x_i^2) & S(x_i) & N \end{vmatrix}, \ \Delta_0 = \begin{vmatrix} S(x_i^2 y_i) & S(x_i^3) & S(x_i^2) \\ S(x_i\, y_i) & S(x_i^2) & S(x_i) \\ S(y_i) & S(x_i) & N \end{vmatrix}$$

$$\Delta_2 = \begin{vmatrix} S(x_i^4) & S(x_i^2 y_i) & S(x_i^2) \\ S(x_i^3) & S(x_i y_i) & S(x_i) \\ S(x_i^2) & S(y_i) & N \end{vmatrix},\ \Delta_3 = \begin{vmatrix} S(x_i^4) & S(x_i^3) & S(x_i^2 y_i) \\ S(x_i^3) & S(x_i^2) & S(x_i y_i) \\ S(x_i^2) & S(x_i) & S(y_i) \end{vmatrix}$$

і для умов прикладу дорівнюють $\Delta_0 = 74\,011$, $\Delta_1 = 106\,620$, $\Delta_2 = 414\,050$ і $\Delta_3 = -725\,110$. Нарешті маємо коефіцієнти параболи: $a = \Delta_1 / \Delta_0 \approx 1{,}44$, $b = \Delta_2 / \Delta_0 \approx 5{,}59$ і $c = \Delta_3 / \Delta_0 \approx -9{,}80$. Аналогічно можуть бути знайдені коефіцієнти полінома степеня більшого, ніж $n = 2$.

3. Інші емпіричні формули.

Не для кожної емпіричної формули вдається легко підібрати коефіцієнти, бо система рівнянь може бути нелінійною. Проте, наведемо емпіричні формули, для яких ця задача розв′язується легко. Отже, можна порадити використовувати їх у практичній роботі.

3.1. Дробово-раціональні функції вигляду

$$y = a + \frac{b}{x}, \qquad y = \frac{1}{a\,x + b}, \qquad y = \frac{x}{a\,x + b}.$$

3.2. Степеневі, показникові і логарифмічні функції

$$y = a\,x^b, \qquad y = a\,b^x, \qquad y = a \lg x + b.$$

У такому випадку використовують стандартний прийом логарифмування, після чого ці функції стають лінійними відносно $\lg x$ і $\lg y$, [33].

Математичне середовище MATLAB дозволяє легко і швидко обчислювати ті громіздкі суми (2.31), знання яких потрібно для знаходження емпіричних рівнянь згідно з матеріалом розділу 4.7. Для цього доцільно використати оператор MATLAB *sum*(*v*) з вектором *v* як аргумент. Наприклад, для обчислення $S(x_i^2\ y_i) = \sum_{i=1}^{N} x_i^2\ y_i$ слід наказати : *Sx2y=sum*((*x* **.^***2*) **.*** *y*). **Увага:** використовуються "операції з крапкою" **.^** і **.*** !

Проте, універсальність математичного середовища MATLAB в тому і полягає, що воно включає широкий набір внутрішніх засобів для типових задач математичного моделювання.

2.4. ІНТЕГРУВАННЯ ФУНКЦІЙ.

Як відомо [34], визначеним ***інтегралом на проміжку*** $a \le x \le b$ від функції $f(x)$ називають граничне значення суми

$$\sum_{i=1}^{n} f(\bar{x}_{i-1})(x_i - x_{i-1}), \qquad \bar{x}_{i-1} \in [x_{i-1}, x_i]. \tag{2.32}$$

де точки $x_0 = a$, x_1, x_2, …, $x_n = b$ (*вузли*) розподіляють проміжок $[a, b]$ на відрізки довжиною $\Delta x_i = x_i - x_{i-1}$ (найчастіше – рівної довжини Δx), коли кількість точок n нескінченно зростає, а довжини відрізків прагнуть до нуля, $\Delta x \to 0$. Нагадаємо, що користуватись саме граничним значенням, яке позначають

$$\int_a^b f(x)dx, \tag{2.33}$$

замість формули (2.32) запропонували І. Ньютон і Г. Лейбніц, і з цим пов'язані найбільші успіхи фізико-математичних наук, починаючи з XVII сторіччя. Геометричний смисл інтеграла (2.33) – площа криволінійної трапеції, обмеженої ліворуч і

праворуч прямими $x = a$ і $x = b$, кривою $f(x)$ зверху і віссю Ox знизу.

У курсі вищої математики ви вивчили способи *інтегрування*, тобто пошуку аналітичних формул для інтегралів (2.33) від багатьох функцій $f(x)$. Проте, далеко не всі функції можуть бути "аналітично проінтегровані", наприклад – інтеграл $\int_a^b \frac{\sin x}{x} dx$ не може бути виражений у вигляді формули. Ще приклад – коли функціональна залежність f від x в аналітичному вигляді відсутня, а її отримано в результаті роботи якогось алгоритму. Практика поставляє багато саме таких задач. Для обчислення інтегралу від таких функції треба зробити "крок назад" від формули (2.33) до формули (2.32).

2.4.1. ФОРМУЛА ПРЯМОКУТНИКІВ ОБЧИСЛЕННЯ ІНТЕГРАЛІВ

Дійсно, виходячи з уявлення про інтеграл (2.33) як про площу, можна бачити, що формула (2.32) дає наближене значення цієї площі (рис. 2.8,А). При цьому i-й доданок в формулі (2.32) відповідає площі елементарного прямокутника $\Delta S_i = f(x_{i-1})(x_i - x_{i-1})$, який заштриховано на рисунку (використано сторону прямокутника f_{i-1} ліворуч від відрізка $\Delta x_i = x_i - x_{i-1}$). В даному випадку, для зростаючої функції , сума площ прямокутників наближає площу криволінійної трапеції "знизу".

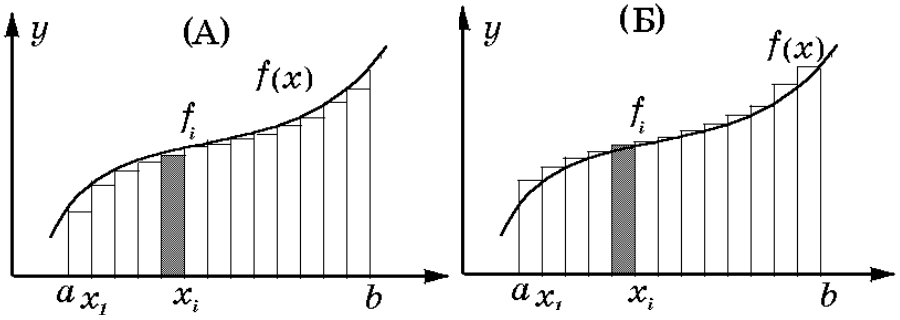


Рис. 2.8. Дві схеми інтегрування методом прямокутників:

(А) наближення "знизу"; (Б) наближення "зверху"

Якщо площу елементарного прямокутника визначити інакше, $\Delta S_i = f(x_i)(x_i - x_{i-1})$, за стороною f_i праворуч від відрізка $\Delta x_i = x_i - x_{i-1}$, то сума площ наближає інтеграл від даної функції "зверху". В кожному випадку такі формули дають спосіб наближеного обчислення інтеграла і мають назву "формули прямокутників". Інший варіант формули прямокутників отримаємо, якщо значення функції f брати у середині відрізку $[x_{i-1}, x_i]$. Тоді маємо $\Delta S_i = f(\frac{1}{2}(x_i + x_{i-1})) \cdot (x_i - x_{i-1})$, а для інтеграла в цілому –

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} f(\tfrac{1}{2}(x_i + x_{i-1})) \cdot (x_i - x_{i-1}).$$

Кожна з формул прямокутників зводить задачу інтегрування до арифметичних дій із значеннями функції f в певних точках інтервалу, вузлах, і дає доволі точні результати, якщо функція $f(x)$ є неперервною і змінюється "не дуже сильно". Вони

легко програмуються, а для рівномірного кроку інтегрування $\Delta x = \frac{b-a}{n}$, формули мають особливо простий вигляд. Наприклад, формула (2.32) перетворюється у таку:

$$\int_a^b f(x)dx \approx \Delta x \sum_{i=0}^{n-1} f(x_i). \qquad (2.34)$$

Таким чином, для обчислення інтеграла від $f(x)$ за допомогою MATLAB треба утворити масив вузлів поділу інтервала *x= a* **:** *Dx* **:** *b – Dx*, де *Dx* = (*b – a)/n* крок інтегрування, обчислити значення функції в них (весь масив значень утворюється лише однією командою*) *F=f*(*x*)), застосувати внутрішню функцію сумації *S=sum*(*F*) і, зрештою, помножити на *Dx*, тобто *Integral=Dx* * *S*.

2.4.2. ФОРМУЛА ТРАПЕЦІЙ

З попереднього матеріалу зрозуміло, що площі елементарних частин криволінійної трапеції можна наближати і інакше, що і дає підстави для альтернативних методів обчислення інтеграла (2.33). Майже найпоширенішим методом є "*метод трапецій*", коли кожну елементарну частину наближають трапецією з двома паралельними сторонами довжини f_{i-1} і f_i, висоти $\Delta x_i = x_i - x_{i-1}$,

*) аби формула для *f* правильно враховувала дії над матрицями в MatLab-середовищі, часто слід використати "команди з крапкою"

площа якої є $\Delta S_i = \frac{1}{2}(f_{i-1} + f_i)\Delta x_i$ (рис. 2.9). Маємо *формулу трапецій*

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} \frac{f_{i-1} + f_i}{2}\Delta x_i = \left\{\begin{matrix}\text{якщо всі} \\ \Delta x_i \text{ рівні}\end{matrix}\right\} = \frac{1}{2}\Delta x \sum_{i=1}^{n}(f_{i-1} + f_i)$$

Легко бачити її відмінність від двох наведених формул прямокутників. У формулі лівих прямокутників (2.34) підсумовуються всі елементи вектора $\{f_0, f_1, f_2, ..., f_{n-1}, f_n,\}$ крім останнього; в формулі правих прямокутників також підсумовуються всі елементи, але за виключенням "нульового". В формулі ж трапецій підсумовуються всі елементи, однак з цієї суми слід виключити половину "нульового" і останнього доданків. Дійсно, легко бачити, що, для рівномірного поділу відрізка інтегрування,

$$\int_a^b f(x)dx \approx \Delta x \sum_{i=0}^{n} f_i - \frac{f_0 + f_n}{2}\Delta x, \text{ де } \Delta x = \frac{b-a}{n}. \qquad (2.35)$$

Виходячи звідси, щойно наведений алгоритм чисельного інтегрування легко модернізувати. Проілюструємо це конкретним прикладом.

Приклад 2.8. Обчислити інтеграл $\int_0^{\frac{\pi}{2}} x\sin x\,dx$.

Іншими словами, обчислити площу заштрихованої фігури на рис. 2.10.

Чисельне обчислення в середовищі MATLAB потрибує лише кількох команд (наводимо їх з коментарями і відповіддю MATLAB):

```
>> a=0; b=pi/2;  % Введення границь інтегрування
>> % N – кількість поділів інтервалу інтегрування
>> N=5; Dx=(b-a)/N;
>> x=a:Dx:b; F=x .* cos(x);  % Вузли і значення заданої функції
>> Integral=Dx*(sum(F)-(F(1)-F(end))/2) % Метод трапецій

    Integral =   0.54959
```

Зрозуміло, що точність результату залежить від параметрів обчислювального алгоритму; у даному випадку такий параметр один, це – *N*, кількість поділів інтервалу інтегрування. Доцільно зробити ще кілька обчислень для більшого *N*. Результати наведено у табл. 2.3. Похибку слід порівнювати з точним значенням інтеграла *I=0.5708*. Останнє можна обчислити або аналітично (використовуючи "інтегрування частинами"), або можливості самого MATLAB (розділ 1.6). Маємо:

```
>> syms x I
>> I= int( x*cos(x) , 0 ,pi/2)
 I = 1/2*pi-1
>> 1/2*pi-1
ans = 0.5708
```

Таблиця 2.3. Дослідження точності інтегрування

N, Кількість поділів	Наближене значення інтеграла	Відностна похибка
5	*0,54959*	*3,7 %*
10	*0,56555*	*0,92 %*
50	*0,57059*	*0,04 %*

Можна бачити, що відносна похибка

$$\varepsilon = \left| \frac{\text{Точне значення} - \text{Наближене значення}}{\text{Точне значення}} \right| \cdot 100\%$$

швидко покращується із збільшенням *N*.

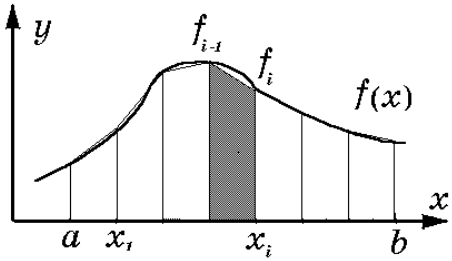


Рис. 2.9. Поясення до інтегрування методом трапецій

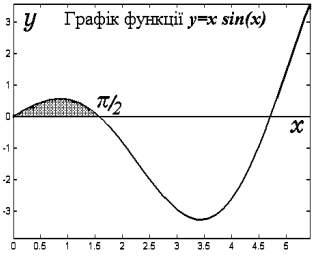


Рис. 2.10. До прикладу 2.8

2.4.3. ПОХИБКИ ПРИ ОБЧИСЛЕННІ ІНТЕГРАЛІВ

Користувач того чи іншого методу обчислень повинен давати собі (і можливим замовникам) раду щодо точності результату. При обчисленні інтегралів маємо нагоду показати, з чого може складатись похибка обчислень і як це враховувати в роботі.

Якщо метод обчислення (в даному випадку – інтегралів) обрано вірно, то має бути принципова можливість наближатись до "вірного значення" як завгодно близько шляхом "кращого" обрання параметрів методу. В останньому прикладі таким параметром методу інтегрування слугувало число N, кількість поділів проміжку інтегрування або – це зручніше – крок чисельного інтегрування $\Delta x = {(b-a)}/{N}$.

Похибкою (*абсолютною*) *розрахунку* R називають різницю між наближеним значенням, отриманим чисельним алгоритмом, і точним значенням. Абсолютна похибка має дві складові – одна виникає внаслідок заміни інтеграла сумою (похибка метода R_1), друга зумовлена похибками обчислення значень f_i і діями над ними (похибка округлення R_2). Очевидно, що для грубих f_i результат не буде точним навіть для найкращого метода. Така *неусувна похибка* оцінена як $|R_2| \le (b-a)\Delta f$, де Δf – абсолютна похибка обчислення підінтегральної функції [36].

А як обрання методу впливає на похибку? Чи гарантує метод, що зменшення Δx зменшує похибку? Для методів чисельного інтегрування доведено [36], що точність отриманих формул дійсно покращується, якщо функція $f(x)$ поводить себе "добре" у тому сенсі, що її перша або друга похідні обмежені по модулю, тобто $|f'(x)| \le M_1$, $|f''(x)| \le M_2$ на всьому проміжку інтегрування.

Як швидко зменшується похибка методу R_1 із зменшенням Δx? Доведено, що похибка формул лівих або правих прямокутників зменшується "зі швидкістю" $\frac{1}{2} M_1 (b-a) \cdot \Delta x$ (першого степеня Δx); похибка формули середніх прямокутників і формули трапецій зменшується як $kM_2 (b-a) \cdot \Delta x^2$ (другого степеня Δx), де $k = \frac{1}{24}$ для першої і $k = \frac{1}{12}$ для другої формул, тобто набагато швидше.

Наведені оцінки часто дозволяють визначити наперед кількість точок поділу n, яка забезпечує потрібну точність. Часто використовують таке практичне правило: інтеграл обчислюють двічі, з n і з $2n$ точками поділу; порівнюють обидва результати I_n і I_{2n} і оцінюють точність як кількість спільних десяткових знаків. При цьому точність обчислення проміжних величин, підінтегральної функції f_i, має бути на один-два порядки краще.

Слід мати на увазі, що інженер інколи стикається і з "поганими" інтегралами, для яких

названі методи не дають точного результату. Такими є, наприклад, інтеграли з теорії рядів Фур'є $\int_a^b f(t)\sin(kt)\,dt$, оскільки для великих k підінтегральна функція багато разів змінює знак на проміжку інтегрування, отже $f'(x)$ і $f''(x)$ майже необмежені. Для таких інтегралів розроблені спеціальні методи.

2.5. ЗВИЧАЙНІ ДИФЕРЕНЦІАЛЬНІ РІВНЯННЯ: ЗАДАЧА КОШІ

Більшість математичних моделей в техніці і природознавстві формулюються у вигляді диференціальних рівнянь, часто – одного звичайного диференціального рівняння (ЗДР) або системи таких рівнянь. З курсу вищої математики відомо, що диференціальне рівняння n-го порядку, або система рівнянь n-го порядку (наприклад – n рівнянь першого порядку) має нескінченну кількість розв'язків, які залежать, однак, від n довільних сталих $C_1,...,C_n$. Для однозначного розв'язування моделі, таким чином, її потрібно доповнити ще якоюсь інформацією. За характером такої інформації найчастіше виникають ***початкові задачі*** (їх ще називають на ім'я славетного французського математика *задачами Коші*) або *граничні задачі*. Кожна з таких задач потребує окремого розгляду, які будуть проведені у даному і наступному підрозділах. Задачі першого типу мають особливе

фундаментальне значення. З них і почнемо. Задачі другого типу відкладем до розділу 4.4.2.

Зустрічаються початкові задачі для системи n звичайних диференціальних рівнянь першого порядку і для одного ЗДР n-го порядку. Покажемо спочатку простий зв'язок між такими задачами.

Початковою задачею, або задачею Коші для системи ЗДР називають проблему знаходження таких функцій $y_1(x),\ldots, y_n(x)$ однієї змінної x, які для $x_0 \le x$ задовольняють наступну систему ЗДР і приймають певні числові значення $y_1, y_2, \ldots, y_n$ на початку області визначення $x = x_0$ (задовольняють *початкові умови*):

$$\begin{aligned}
&\frac{dy_1}{dx} = f_1(x, y_1,..., y_n),\\
&\frac{dy_2}{dx} = f_2(x, y_1,..., y_n),\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \cdots\cdots\\
&\frac{dy_n}{dx} = f_n(x, y_1,..., y_n),\\
&y_1(x_0) = y_1, \quad y_2(x_0) = y_2, \ \cdots\ , \ y_n(x_0) = y_n.
\end{aligned} \qquad (2.36)$$

Доведено теорему існування і єдиності розв'язку такої задачі за певних, не дуже обтяжливих умов на функції $f_i(x, y_1,..., y_n)$, зокрема – неперервності їхніх похідних. Тобто такі математичні моделі майже завжди змістовні. Зокрема, систему (2.36) для трьох

функцій $y_1(x)$, $y_2(x)$, $y_3(x)$ часто інтерпретують як задачу руху точки в тривимірному просторі у часі x. Аналогічно, система (2.36) для шести функцій – проблема руху двох точок у просторі і т.д.

Початковою задачею, або задачею Коші для звичайного диференціального рівняння (ЗДР) n-го порядку називають проблему знаходження такої функції $y(x)$ однієї змінної x, яка задовольняє дане ЗДР для $x_0 \le x$, і її похідні в початковій точці x_0 задовольняють початкові умови:

$$y^{(n)} = F(x, y', y'', ..., y^{(n-1)}), \quad y(x_0) = y_0, \quad y'(x_0) = y_1, \ldots, y^{(n-1)}(x_0) = y_{n-1}. \tag{2.37}$$

Існування і єдиність розв'язку такої математичної моделі суттєво залежить від властивостей функції F. У більшості випадків можна вважати, що задача (2.37) розв'язується однозначно, коли кількість початкових умов дорівнює порядку рівняння.

Відомо [33,35-37], що таке ЗДР може бути записане у вигляді еквівалентної *нормальної системи* диференціальних рівнянь і початкової задачі для неї:

$$\frac{dy}{dx} = y_1,$$
$$\frac{dy_1}{dx} = y_2,$$
$$\ldots\ldots \tag{2.38}$$

$$\frac{dy_{n-2}}{dx} = y_{n-1},$$

$$\frac{dy_{n-1}}{dx} = F(x, y, y_1, ..., y_{n-1}),$$

$$y(x_0) = y_0, \quad y_1(x_0) = y_1, \quad \ldots \quad , \quad y_{n-1}(x_0) = y_{n-1},$$

де перші $n-1$ рівняння фактично вводять $n-1$ допоміжну функцію $y_1(x), \ldots, y_{n-1}(x)$.

Задачу Коші для системи рівнянь (2.38), таким чином, можна вважати базовою. Для неї доречно перейти до векторної форми запису. Система (2.38) пов'язує похідну вектора-стовпчика Y з деякою векторною функцією від x і Y, тобто

$$\frac{dY}{dx} = F_*(x, Y), \qquad Y(0) = Y_0, \tag{2.39}$$

де елементами Y є названі функції:

$$Y(x) = \begin{pmatrix} y_1(x) \\ y_2(x) \\ \ldots \\ y_n(x) \end{pmatrix}, \quad Y_0 = \begin{pmatrix} y_1 \\ y_2 \\ \ldots \\ y_n \end{pmatrix}, \quad F_* = \begin{pmatrix} f_1 \\ f_2 \\ \ldots \\ f_n \end{pmatrix}.$$

У випадку одного рівняння (2.37) такими функціями будуть

$$f_i(x, y, y_1, ..., y_{n-1}) = y_{i+1} \quad \text{для} \quad i = 0, 1, ..., n-2$$

$$\text{і} \quad f_n(x, y, y_1, ..., y_{n-1}) = F(x, y, y_1, ..., y_{n-1}).$$

Розв'язок диференціального векторного рівняння (2.39) – це будь-яка векторна функція $Y(x)$, що

задовольняє написане векторне рівняння і компоненти якої приймають вказані значення на початку $x = x_0$. Це дозволяє надавати геометричної інтерпретації процедурі розрахунків.

2.5.1. МЕТОД ЕЙЛЕРА ІНТЕГРУВАННЯ ПОЧАТКОВОЇ ЗАДАЧІ

Розглянемо найпростіший і майже очевидний спосіб чисельного розв'язування задачі Коші (2.39), ще кажуть – спосіб інтегрування відповідного ЗДР. Початкова умова для задачі (2.39) показує, що *інтегральна крива*, що розшукується, має проходити через дану точку $\{x_0, y_0\}$ площини. Треба знайти значення функції $y(x)$ для наступних значень аргументу $x = x_1 = x_0 + h$, $x_2 = x_1 + h$, $x_3 = x_2 + h$ і т. д., що відстоять один від одного на *крок інтегрування* h (рис. 2.11).

З вищої математики відомо, що приріст функції у першій точці є $\Delta y \approx y'(x_0, y_0)\, h$, тобто, враховуючи рівняння, маємо $\Delta y \approx f(x_0, y_0)\, h$. Таким чином, по нульовій ординаті точки можемо наближено обчислити ординату її першого положення:

$$y_1 = y_0 + \Delta y \approx y_0 + f(x_0, y_0)\, h. \qquad (2.40)$$

Таке обчислення нагадує розрахунок руху матеріальної точки: адже поряд з положенням точки

y_0 відома її швидкість $V_1 = y'_x = f(x_0, y_0)$, яку множимо на час h для знаходження зсуву Δy.

Знаємо, таким чином, перше положення точки "1". Для нього обчислюємо нову "швидкість" точки $V_2 = y'_x = f(x_1, y_1)$. Тепер ці дані вважаємо за вихідні і, застосовуючи знову формулу (2.40), знаходимо друге положення точки, потім третє, четверте і т.д., як показано на рис. 2.11. "Траєкторія" точки і дає наближено шукану інтегральну криву. Розрахункову формулу (2.40) називають *формулою Ейлера*.

2.5.2. МЕТОД РУНГЕ-КУТТА ЧИСЕЛЬНОГО ІНТЕГРУВАННЯ

Точність формули Ейлера, однак, незадовільна. Похибка розрахунку весь час накопичується, оскільки вектор швидкості в кожній наступній точці ми знаємо наближено. Проте, ця формула підказує ідею набагато кращого методу, навіть *ряд методів Рунге-Кутта*.

Тепер, аби потрапити в точку "1", будемо намагатися зробити кілька попередніх напівкроків. Нехай, таких напівкроків два, k_1 і k_2, так що ординату чергової точки знаходимо за формулою

$$y_1 = y_0 + C_1 k_1 + C_2 k_2 ,$$

де перший напівкрок взято як раніше, $k_1 = h\, f(x_0, y_0)$, а другий – для зсунутого моменту часу і координати $k_2 = h\, f(x_0 + \alpha_2 h, y_0 + \beta_2 k_1)$. Було сформульовано [33,35-

37] задачу оптимального вибору сталих C_1, C_2 і α_2, β_2 і отримано кілька таких наборів констант, наприклад:

$$C_1 = 0,\ C_2 = 1,\ \alpha_2 = \beta_2 = \frac{1}{2} \qquad \text{і} \qquad C_1 = C_2 = \frac{1}{2},\ \alpha_2 = \beta_2 = 1.$$

Кожний з наборів дає доволі хороші розрахункові формули. Проте, найбільше застосування здобули формули з чотирма напівкроками.

За такими формулами положення кожної чергової точки знаходять за її координатами на попередньому кроці і на напівкроках

$$y_1 = y_0 + C_1 k_1 + C_2 k_2 + C_3 k_3 + C_4 k_4\ , \qquad (2.41)$$

де кожний напівкрок враховує попередній:

$$k_1 = h\ f(x_0, y_0), \qquad k_2 = h\ f(x_0 + \alpha_2 h, y_0 + \beta_{21} k_1),$$

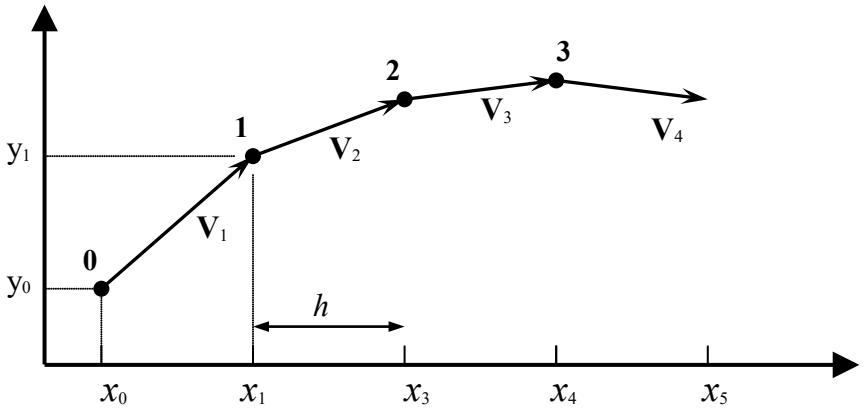


Рис. 2.11. Ідея методу Ейлера чисельного інтегрування диференціального рівняння

$$k_3 = h\ f(x_0 + \alpha_3 h, y_0 + \beta_{31} k_1 + \beta_{32} k_2) \qquad \text{і} \qquad (2.42)$$
$$k_4 = h\ f(x_0 + \alpha_4 h, y_0 + \beta_{41} k_1 + \beta_{42} k_2 + \beta_{43} k_3).$$

З багатьох можливих "оптимальних" наборів констант [36] наведемо лише один:

$$C_1 = C_4 = \frac{1}{6},\quad C_2 = C_3 = \frac{1}{3},\quad \alpha_2 = \beta_{21} = \frac{1}{2},$$
$$\alpha_3 = \frac{1}{2}, \beta_{31} = 0, \beta_{32} = \frac{1}{2},\quad \alpha_4 = 1, \beta_{41} = \beta_{42} = 0, \beta_{43} = 1. \tag{2.43}$$

Застосування методу Рунге-Кутта у такій формі ілюстровано наступним прикладом, де остання формула очевидним способом узагальнена для системи диференціальних рівнянь вигляду (2.38).

2.5.3. ПРИКЛАД ЗАСТОСУВАННЯ МЕТОДУ РУНГЕ-КУТТА

Приклад 2.9. *Розв'язати диференціальне рівняння* $y''+9y=0$ *за початкових* умов $y(0)=1,\quad y'(0)=1,\quad y''(0)=1$.
(взято з [30], №4301).

Перш за все задане рівняння слід написати у вигляді нормальної системи рівнянь першого порядку. Позначаємо $y' = y_1(x)$ і $y'' = y_2(x)$. Тепер вихідна і дві нові функції задовольняють систему рівнянь і початкові умови

$$\frac{dy}{dx} = y_1,\quad \frac{dy_1}{dx} = y_2,\quad \frac{dy_2}{dx} = -9y_1,$$
$$y(0) = 1,\qquad y_1(0) = 1,\qquad y_2(0) = 1\ .$$

Побудуємо за допомогою MATLAB чисельний розв'язок даної задачі з кроком *h=.4*. Праві частини заданих рівнянь є функціями, взагалі кажучи, від

компонент y, y_1, y_2 і аргументу x, які мають такий вигляд:

de0(*x , y0 , y1 , y2*)= *y1*, *de1*(*x , y0 , y1 , y2*)= *y2*, *de0*(*x , y0 , y1 , y2*)= –9**y1*

(назви функцій походять від *Differential Equation*; все, що має відношення до першого рівняння або першої, вихідної, невідомої y, позначємо індексом *0*). Надаємо вихідних значень аргументу, всім змінним і "лічильнику кроків" ***i***

>> x0=0; *y0=1*; *y1=1*; *y2=1*; *h=.4*; *i=0*;

і виконуємо перший напівкрок

>> x_1=x0; *k1_0=h*de0*(*x_1,y0,y1,y2*); *k1_1=h*de1*(*x0,y0,y1,y2*); . . .
*k1_2=h*de2*(*x0,y0,y1,y2*);

і другий напівкрок

>>x_2=x0+h/2; *y2_0=y0+k1_0/2*; *y2_1=y1+k1_1/2*; *y2_2=y2+k1_2/2*; . .
.
*k2_0=h*de0*(*x_2,y2_0,y2_1,y2_2*); *k2_1=h*de1*(*x_2,y2_0,y2_1,y2_2*);. .
.
*k2_2=h*de2*(*x_2,y2_0,y2_1,y2_2*);

В результаті маємо координати *x_m*, *ym_ j*, де обчислюємо коефіцієнти формул Рунге-Кутта для кожної з трьох невідомих (*m* номер напівкроку і *j* – номер рівняння, *j=0, 1, 3*; обидва індекси розділено символом _, "підкреслення"). Маємо також самі коефіцієнти *km_j*, згідно з формулами (2.42), (2.43). Узагальнення формул для системи рівнянь полягає у тому, що кількість коефіцієнтів співпадає з кількістю рівнянь, але для кожного рівняння коефіцієнти ідентичні. Слід також пояснити, що замість *de0*, *de1* і

de2 треба було б писати вищенаведені вирази для правих частин рівнянь. Проте, аби уникнути громіздкості записів, зручніше запрограмувати ці формули як програми-функції (див. розділ 3.2). За допомогою редактора *m*-файлів створюємо файли *de0.m*, *de1.m* і *de2.m*, завдяки чому вказані функції працюють як внутрішні функції MATLAB. Наведемо текст лише останнього файлу, бо інші аналогічні:

Лістінг 2.1 (*m*-файл *de2.m*)

```
function y=de2(x, y0, y1, y2)
% Права частина третього рівняння системи ЗДР № 4301 [30]
y= - 9*y1;
```

Аналогічно виконуємо третій напівкрок

```
>> x_3=x0+h/2; y3_0=y0+k2_0/2; y3_1=y1+k2_1/2; y3_2=y2+k2_2/2;...
  k3_0=h*de0(x_3,y3_0,y3_1,y3_2); k3_1=h*de1(x_3,y3_0,y3_1,y3_2); ...
  k3_2=h*de2(x_3,y3_0,y3_1,y3_2);
```

і четвертий напівкрок методу Рунге-Кутта:

```
>> x_4=x0+h; y4_0=y0+k3_0; y4_1=y1+k3_1; y4_2=y2+k3_2; ...
  k4_0=h*de0(x_4,y4_0,y4_1,y4_2);
k4_1=h*de1(x_4,y4_0,y4_1,y4_2); ...
  k4_2=h*de2(x_4,y4_0,y4_1,y4_2);
```

Для кожного з трьох рівнянь системи тепер отримано по чотири коефіцієнти *k1_j*, *k2_j*, *k3_j* і *k4_j*. Закінчуємо один крок методу Рунге-Кутта і за

формулою (2.41) отримуємо значення кожної з трьох невідомих функцій *z0*, *z1* і *z2* для аргументу $x = x_0 + h$:

```
>> z0=y0+(k1_0 +2*k2_0 +2*k3_0 +k4_0)/6; . . .
   z1=y1+(k1_1 +2*k2_1 +2*k3_1 +k4_1)/6; . . .
   z2=y2+(k1_2 +2*k2_2 +2*k3_2 +k4_2)/6;
```

На запитання про значення цих величин (робити це не обов'язково) MATLAB відповість:

```
>> z0, z1, z2
   z0 = 1.3744
   z1 = 0.6704
   z2 = -2.3696
```

Це – значення розв'язку $y(x)$ і його похідних у точці $x = x_0 + h$. Готуємо наступний крок методу Рунге-Кутта:

```
>> x0=x0+h; y0=z0; y1=z1; y2=z2;  i=i+1, . . .
   X(i)=x0; Y(1,i)=y0; Y(2,i)=y1;  Y(3,i)=y2;
```

"Переприсвоєння" у першому рядочку дозволяє повторити всю описану процедуру в незмінному вигляді; крім того, ***i*** тепер рахує кількість кроків з $\Delta x = h$. Другий рядочок "складає" отримані дані у вектор *X* для аргументу і двовимірний масив *Y* для розв'язку – рядочок *Y*(*1*, **:**) для функції $y(x)$ і рядочки *Y*(*1*, **:**) і *Y*(*1*, **:**) для похідних $y'(x)$ і $y''(x)$, обчислених у вузлах $x = x_0 + h$, $x_0 + 2h$, $x_0 + 3h$ і т.д.

Таких кроків зробимо 10 – отримали таблиці розв'язку для $0 \le x \le 4$. Побудуємо графіки функцій*), які розрахували:

```
>> figure(1);plot(X, Y(1, : ));
>> title('Solution  y1(x)'); legend('Numerics for h=.4')
>> figure(2);plot(X, Y(2, : ));
>> title('Derivative y1(x)'); legend('Numerics for h=.4')
>> figure(3);plot(X, Y(3, : )); title('Second Derivative y2(x)');
```

Вони представляють, відповідно, розв'язок заданого рівняння третього порядку, першу і другу похідні від нього; на рис. 2.12 криві *1* представляють лише дві перші функції. Зверніть увагу: всі криві проходять через точку (*0,1*), як це "замовлено" початковими умовами!

Обидві криві *1* на рис. 2.12 доволі ламані. Проте, порівняємо їх з графіками точного розв'язку, який було отримано в прикладі 1.7. З загального аналітичного розв'язку за допомогою початкових умов знаходимо значення трьох констант і остаточний вигляд розв'язку у формі $y = \frac{10}{9} + \frac{1}{3}\sin(3x) - \frac{1}{9}\cos(3x)$. Відповідно, похідна має рівняння $y' = \cos(3x) + \frac{1}{3}\sin(3x)$. Виконуємо команди:

*) Якщо надписи командою *title* зробити українською або російською мовами, можуть виникнути "непорозуміння" з MatLab. Краще потім змінити надписи за допомогою меню *Figure* (див. розділ 3.3)

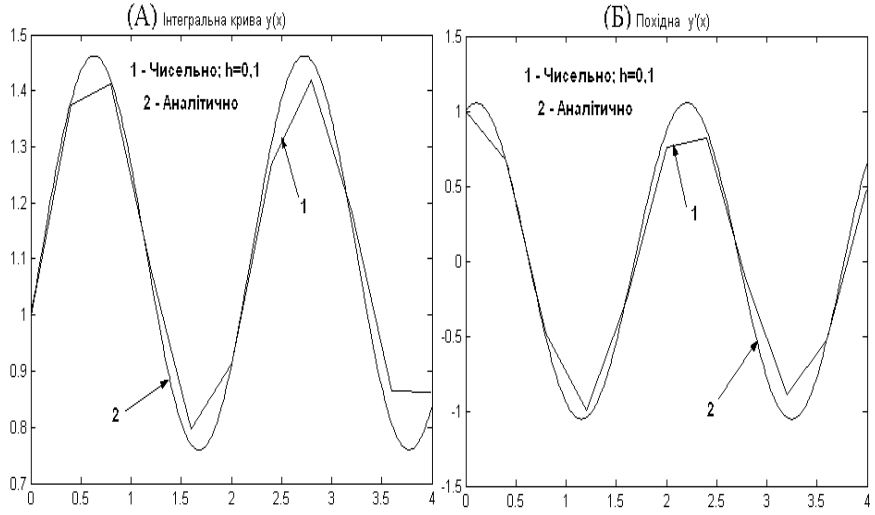


Рис. 2.12. Чисельний розв'язок рівняння прикладу 2.9 (крива 1) у порівнянні з аналітичним (крива 2). (А) – шукана функція, (Б) – її похідна

*>> figure(1); hold on; fplot('10/9+sin(3*x)/3 - cos(3*x)/9', [0 4], 'r')*
*>> figure(2); hold on; fplot('cos(3*x)+sin(3*x)/3', [0 4], 'r')*

Тепер можемо порівняти попередню ламану *1* з кривою *2*, що представляє точний розв'язок (червона на екрані). Чисельні результати доволі непогано відповідають аналітиці (рис. 2.12)!

Тепер цікаво було б повторити всі чисельні розрахунки з половинним кроком *h=.2*. Розпочати слід з команди

>> x0=0; y0=1; y1=1; y2=1; h=.2; i=0;

і надалі повторювати описані вище групи команд. Проте, це громіздка робота, в якій можливі помилки. Краще з цих команд створити *script*-програму, яку запишемо у файл *runge.m*:

Лістінг 2.2 (*m*-файл *runge.m*)

```
% Процедура одного кроку Рунге-Кутта
% для системи ЗДР з 3х з правими частями в de0, de1 і de2.
% Нагадування: спочатку слід надати початкових значень величинам
% x0=  ; y0=  ; y1=  ; y2=  ;
%  і кроку інтегрування h=;

% Перший напівкрок методу Рунге-Кутта:
x_1=x0;
k1_0=h*de0(x0,y0,y1,y2);
k1_1=h*de1(x0,y0,y1,y2);
 k1_2=h*de2(x0,y0,y1,y2);

% Другий напівкрок методу Рунге-Кутта:
x_2=x0+h/2;  y2_0=y0+k1_0/2;  y2_1=y1+k1_1/2;  y2_2=y2+k1_2/2;
k2_0=h*de0(x_2,y2_0,y2_1,y2_2);
k2_1=h*de1(x_2,y2_0,y2_1,y2_2);
k2_2=h*de2(x_2,y2_0,y2_1,y2_2);

% Третій напівкрок методу Рунге-Кутта
x_3=x0+h/2;  y3_0=y0+k2_0/2;  y3_1=y1+k2_1/2;  y3_2=y2+k2_2/2;
k3_0=h*de0(x_3,y3_0,y3_1,y3_2);
k3_1=h*de1(x_3,y3_0,y3_1,y3_2);
k3_2=h*de2(x_3,y3_0,y3_1,y3_2);

% Четвертий напівкрок методу Рунге-Кутта
x_4=x0+h;  y4_0=y0+k3_0;  y4_1=y1+k3_1;  y4_2=y2+k3_2;
k4_0=h*de0(x_4,y4_0,y4_1,y4_2);
k4_1=h*de1(x_4,y4_0,y4_1,y4_2);
k4_2=h*de2(x_4,y4_0,y4_1,y4_2);
```

```
% Завершення одного кроку методом Рунге-Кутта
z0=y0+(k1_0 +2*k2_0 +2*k3_0 +k4_0)/6;
z1=y1+(k1_1 +2*k2_1 +2*k3_1 +k4_1)/6;
z2=y2+(k1_2 +2*k2_2 +2*k3_2 +k4_2)/6;

% Нагадування: перш, ніж повторити цю процедуру,
% слід перепризначити:
% x0= x0+h;  y0=z0 ;  y1=z1;  y2=z2 ;
% Кінець програми Runge
```

Тепер (див. розділ 3.1) всі ці команди можна виконати однією командою *Runge* з командного рядочка. Ще розумніше виконувати таку складену команду

```
Runge; x0=x0+h;y0=z0;y1=z1;y2=z2; i=i+1, X(i)=x0;Y(1,i)=y0;
Y(2,i)=y1; Y(3,i)=y2;
```

кожного разу (поки ***i*** не покаже потрібну кількість кроків) викликаючи її "стрілкою угору". А наприкінці – така само побудова графіків.

Повторення розрахунку з кроком *h=.2* покаже, що нова крива ще ближче до аналітичної. А чисельний розв'язок з кроком *h=.1* вже майже співпадає з аналітикою (проте треба зробити вже 40 кроків, аби досягти $x = 4$).

Таким чином, зменшення *h* покращує точність результату. Часто, особливо коли аналітичний розв'язок відсутній, використовують *правило Рунге* для оцінки точності розрахунку задачі Коші. Для цього проводять два розрахунки, з кроком *h* і з кроком $h/2$; якщо обидві інтегральні криві співпадають з достатньою точністю – розв'язання задачі вважають закінченим. Якщо ж істотно

відрізняються – проводять новий розрахунок з кроком $h/4$ і так само порівнюють з передостаннім; якщо точність знов-таки незадовільна – продовжують зменшувати крок інтегрування. Чи збігається розрахунок при $h \to 0$ – це інколи важка проблема, яка може змусити обирати новий чисельний метод.

2.6. КОНТРОЛЬНІ ПИТАННЯ ДО РОЗДІЛУ 2

1. На які питання ви маєте відповісти, отримавши для розв'язання (а можливо – і для самостійного дослідження) ту чи іншу математичну задачу?
2. Що таке СЛАР? Які методи розв'язування СЛАР ви знаєте? У чому полягає метод Гаусса розв'язування систем лінійних алгебраїчних рівнянь? У чому суть методу ітерацій для СЛАР? Чому СЛАР мають фундаментальне значення для математичного моделювання фізичних, технічних, економічних явищ? За яких умов СЛАР не має розв'язку? Має нескінченну кількість розв'язків? Лише два розв'язки? Лише єдиний?
3. Що називають алгебраїчним рівнянням з однією невідомою? Скільки розв'язків може мати математична модель якогось явища у формі многочлена *n*-го степеня? У чому фундаментальність такого роду математичних задач?
4. Що називають трансцендентним рівнянням з однією невідомою? Чи існують якісь універсальні методи розв'язування таких задач? Скільки розв'язків може мати така задача?
5. Отримавши якесь трансцендентне рівняння $f(x) = 0$, як ви почнете його досліджувати? Чи може бути корисним побудова графіка функції $y = f(x)$?
6. Оцінивши значення того чи іншого кореня грубо, наближено, яким алгоритмом можно уточнити його більш точно? З якою точністю?

7. У чому полягає метод "поділу пополам"? А метод ітерацій? А метод хорд? Навіщо висувають ***умову збіжності*** методу і чи завжди збіжність гарантована?
8. Чому не вистачає аналітичних методів обчислення інтегралів, чому виникає потреба у їх чисельному обчисленні? Які ви знаєте методи чисельного обчислення визначених інтегралів? Як оцінюють точність (похибку) обчислення інтегралу?
9. Чому більшість математичних моделей формулюється через звичайні диференціальні рівняння (ЗДР)? Що таке *порядок* ЗДР або системи ЗДР? Які дві принципово різні задачі для ЗДР ви знаєте? Чому різновид задач для ЗДР, що вперше був докладно досліджений французьким математиком О. Коші, називають *початкові задачі*? У чому їх відмінність від *граничних задач* для ЗДР? Які умови і скільки слід брати до уваги, аби очікувати наявність розв'язку задачі і його єдиність?
10. У чому сутність методу Ейлера інтегрування задачі Коші для ЗДР? А суть методу Рунге-Кутта? Як можна оцінити похибку розв'язку задачі?
11. Як можна крайову задачу для ЗДР звести до початкової задачі? У якому відношенні спрощується дослідження крайової задачі у випадку лінійності ЗДР?

3. ОСНОВИ ПРОГРАМУВАННЯ В MATLAB

Система MATLAB дозволяє користувачеві надавати ім'я певним ланцюжкам команд таким чином, що це ім'я починає працювати як команда MATLAB, виконання якої в командному рядку викликає виконання того самого ланцюжка команд. Команди, створені користувачем, називаються ***програмами***, процес їхнього створення – ***програмуванням***. Якщо користувач створив певну команду (програму), то та в свою чергу може бути використана в якійсь більш складній програмі. Програмування, таким чином, дозволяє складні, розвинені й довготривалі обчислювальні процеси складати з більш простих.

Система MATLAB розуміє програми двох типів – програми-функції (*functions*) і програми-сценарії (*scripts*), які ми для скорочення будемо називати *функціями* і *скриптами*. Для їхнього створення MATLAB має спеціальний текстовий редактор, який зберігає написані програми у файлі із вказаним користувачем іменем і з розширенням "**.***m*". (Можна використовувати і якийсь інший текстовий редактор, та це навряд чи доцільно).

Даний розділ присвячено основам програмування в системі MATLAB. Дано приклади функцій і скриптів, рекомендації щодо роботи з вбудованим текстовим редактором MATLAB (його називають редактором *m* -файлів), команди логіки *if*, *else*, *ifelse* і команди управління обчислювальним процесом *for* . . . *end*, *while* . . . *end*. Запропоновано лабораторну роботу 5 [3], яка дозволяє закріпити навички програмування. Наведені приклади і завдання лабораторної роботи спираються на методи, викладені у попередньому, другому розділі.

Щоб запустити будь-яку програму MATLAB, необхідно набрати її ім‘я в командному рядку командного вікна MATLAB і натиснути клавішу *Ввод*. Приклади команд:

>> visualize *>> step1*(*z*)

>> step_pi(t) *>> MyDeterminant*(*A*),

де *visualize*, *step1*, *step_pi* і *MyDeterminant* – програми (команди), створення яких надано нижче. Зараз важливо сказати, що кожна програма для стороннього користувача (або для самого її автора після тривалого проміжку часу) – це "чорна скринька", яка створює

(говорять, "*видає*") певні величини (***вихідні*** дані) на підставі даних, які вже існують в математичному середовищі MATLAB або спеціальним чином вказані даній програмі (***вхідні*** дані).

Програма, яка видає дані *y*, *x1* та інші на підставі вже створених в математичному середовищі даних і не потребує ніяких вхідних даних, називається *скриптом*. Команда (програма) *visualize* є скриптом, бо не має вхідних параметрів. Програма, яка може використовувати вже створені дані, але потребує певних вхідних даних ***x***, a, b,…, називається *функцією*. Наприклад, *step1*(*z*) та інші є саме функціями. Різницю між двома видами програм MATLAB може пояснити рис. 3.1.

Таким чином, *скрипт* і *функція* є двома можливими різновидами програм MATLAB, яким надані певні іменами. Вони зберігаються у файлах з саме такими іменами. Розглянемо правила створення таких програм.

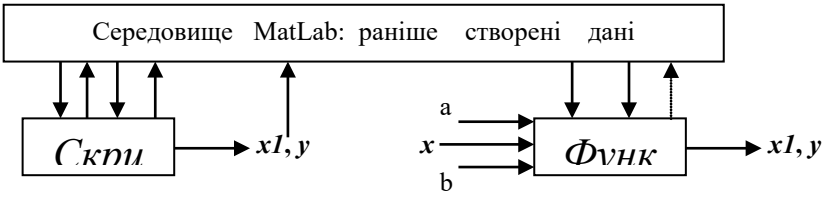


Рис. 3.1. Пояснення щодо різниці між двома типами програм

3.1. ПРОГРАМИ-СЦЕНАРІЇ

Скрипт, тобто програма-сценарій, – це послідовність команд середовища MATLAB, яка зберігається у файлі з певним іменем і розширенням "**.***m*". Створювати скрипти доцільно тоді, коли необхідно неодноразово повторювати відповідну послідовність команд.

Перш за все слід викликати редактор *m*-файлів. Для цього в меню MATLAB мишею обираємо

File \ New ► M-file,

або натискуємо іконку 🗋 . З'являється вікно редактора *m*-

файлів з порожньою сторінкою майбутнього *m*-файлу, який поки що озаглавлений "*Untitled*". Після набору кількох перших рядків майбутньої програми, новий *m*-файл слід зберігти. Для цього необхідно мишею у меню обрати ***File\Save*** або натиснути комбінацію "гарячих клавіш" <*Ctrs* + *S*>, і у вікні "***Save file as*:**" у першому рядку "*Ім'я файла*" замість запропонованої назви ***Untitled.m*** ввести бажане ім'я, наприклад ***script1.m*** (радимо користуватись лише латинськими літерами).

Далі можна продовжувати набір програми, час від часу зберігаючи її (***File \ Save*** або <*Ctrs* + *S*>). Оскільки ім'я вже надане, MATLAB зберігає набрані дані "мовчки", без посередництва вікна "***Save file as*:**". Лише у випадку, коли ви захочете змінити назву програми (файлу), слід через меню натиснути мишею "***File \ Save as . . .***" і у вікні "***Save file as*:**" замінити запропоновану назву файла на бажану.

Отже, пишемо програму-скрипт. Формат написання програм, взагалі кажучи, довільний, проте кожному слід виробити свій власний стиль і порядок, враховуючи надані нижче рекомендації. В першому і, можливо, в кількох наступних рядках слід написати **коментар** – власний текст після знаку **%** на початку кожного рядка. В коментарі доцільно коротко записати, для якої задачі створено програму, навести певні формули і звідки їх взято, навести прізвище автора програми і дату її створення, наприклад:

```
% Програма перетворення декартових координат {x,y,z}
% у сферічні {rho,theta,r}
% за формулами rho= …, theta=… i =…
% (формули взято з книги [Курс вищої математики]
%
% Розробник програми Б.М. Нестеренко
% Дата створення . . . .
```

Кожного разу, натискуючи на клавішу *Ввод* для переходу на новий рядок, редактор *m*-файлів нумерує рядки тексту, так що наведені рядки мають номери від *1* до *7*.

Написані на початку скрипту коментарі відіграють доволі важливу роль, допомогаючи сторонньому користувачеві узнати (або самому авторові – пригадати) призначення даної програми. Якщо надалі в командному рядочку MATLAB вводити

>> *help* ***script1*** **,**

то отримуємо записаний під коментарем текст від першого рядка до четвертого (див. вище). Якщо потрібно вивести і інформацію з двох наступних рядків – вилучіть рядок 5 з порожнім коментарем.

Далі слід написати **тіло програми** – послідовність команд системи MATLAB, яка власне й утворює програму. Наведемо приклад скрипту візуалізації трьох функцій (побудови графіків), які в MATLAB називаються, припустимо, *f0*, *f1* і *f2* і приклад яких наведено далі. В командному вікні виконуємо:

```
>> Xbegin=0; Xend=10;  N=1000;   % Початок і кінець табуляції
 >> Step= (Xend – Xbegin)/N;          % Крок табуляції                (3.1)
>> X= Xbegin : Step :  Xend ; % Вектор значень аргумента
>> F0=f0(X);  F1=f1(X); F2=f2(X);  % Вектори значень 3-х функцій
>> visualize
```

Коментарі до кожної команди[‡] (їх можна не писати) пояснюють призначення певних дій: у першому рядку задаються початок і кінець відрізку, на якому будемо обчислювати значення функцій (табулювати) а також кількість точок поділу *N*; далі обчислюємо крок табуляції *Step*; у третьому рядку створюємо вектор значень аргументу *X*, який складається з *N+1* елементів, починаючи з числа *Xbegin* і закінчуючи числом *Xend*; у четвертому рядку обчислюються три вектори, що містять значення функцій *f0*, *f1* і *f2* для відповідних аргументів (маємо на увазі, що функції *f0*, *f1* і *f2* MATLAB "знає": або ми пишемо конкретні функції *sin*, *tan*, *exp* тощо, або створено файли *f0.m*, *f1.m* і *f2.m*). Команда ***visualize*** в останньому рядку раніше була невідома MATLAB, але ми попередньо створили файл *visualize.m* з наступним змістом:

[‡] Як ми вже пояснювали (див. також розділ 3.3), коментарі краще писати латинськими літерами, бо в деяких версіях MatLab‘у виникають несподівані проблеми!

Лістінг 3.1А (*m*-файл *visualize.m*)

```
% Програма побудови трьох графіків функцій (візуалізації)
% для їхнього співставлення між собою

%Copyright Gayev Ye.A.
figure; plot(X, F0, 'r',   X, F1, 'b',   X,F2, 'g');
title('Співставлення графіків трьох функцій')                    (3.2)
grid on;
xlabel(' Незалежний аргумент x '); xlabel(' Функції від x ')
%-------------------------------------------------------
% Кінець програми visualize.
```

Прочитавши послідовність команд (3.2) з означеного файлу, MATLAB виконуватиме її кожного разу як єдину команду *visualize*. Відмітимо, що потрібні для її роботи дані *x, F0, F1, F2* MATLAB бере серед раніше створених даних. Якщо скрипт створює якісь нові дані, то вони поступають у середовище MATLAB, як показано на рис. 3.1, і стають доступними всім наступним командам.

Інші приклади скриптів надають програми *runge* розділу 2.5.3 та *MyFFT* розділу 4.6.

3.2. ПРОГРАМИ-ФУНКЦІЇ

Програми-функції, як і скрипти, створюють за допомогою редактора *m*-файлів (див. попередній підрозділ), і зберігають в *m*-файлі з певним іменем, але їх структура трохи відрізняється від структури скриптів.

Першим рядком програми-функції має бути текст вигляду

function ***ВихіднийАргумент*** *=FunctionName*(***ПерелікВхіднихАргументів***)

де *FunctionName* – довільне ім'я, яке дозволене в MATLAB. Рекомендовано зберігати програму-функцію у файлі з тим самим іменем *FunctionName*. Зберігати програму слід так само, як зберігають програми-сценарії.

Наступні рядки програми доцільно заповнити коментарями на зразок наведених вище. Вони мають

нагадувати користувачеві потрібну інформацію щодо функції за допомогою команди-запиту

>> help FunctionName

"*ПерелікВхіднихАргументів*", у якому вхідні аргументи перераховані через кому, використовується у подальшому **тілі програми-функції** для утворення нових змінних і надання їм певних значень за допомогою математичних операцій. Припустимо, що цей перелік виглядає таким чином:

$x1, x2, x3$.

Тоді вказані змінні можуть бути використані в тілі програми для створення нових змінних, наприклад,

Sum=x1 + x2; A= sqrt(x3); і так далі. . .

Наприкінці важливо надати потрібного значення ***Вихідному Аргументу***. Останній може бути одним іменем, наприклад, ***y***, і тоді перший рядок виглядає

$$function\ \boldsymbol{y} = FunctionName(\ x1, x2, x3\ )\ \ ,$$

а може складатись з кількох змінних, взятих у квадратні дужки, наприклад:

$$function\ [\ \boldsymbol{y1, y2}\ ]\ = FunctionName(\ x1, x2, x3\ )\ \ .$$

Приклад 3.1. Побудувати функції, які будуть репрезентувати частинні суми ряду Фур‘є

$$\frac{4}{\pi}\left(\frac{\sin x}{1}+\frac{\sin 3x}{3}+\frac{\sin 5x}{5}+\frac{\sin 7x}{7}+\ldots\right)\quad , \qquad (3.3)$$

який, як відомо [30,34], на відрізку $x\in[-\pi,\pi]$ наближує розривну функцію

$$f(x)=\begin{cases}-1, & якщо\ \ x\in[-\pi,0]\\ \\ 1, & якщо\ \ x\in(0,\pi)\end{cases}. \qquad (3.4)$$

Програма-функція для обчислення математичної функції $f_1(x) = \frac{4}{\pi}\frac{\sin x}{1}$ (перша частинна сума ряду) виглядає таким чином:

```
function F=F1(x)
% First partial sum of the Fourier series
%  4*(sin(x)/1 +sin(3*x)/3 +sin(5*x)/5  +...)/pi
%
% Copyright Гаєв Є.О.

F= 4*(sin(x)/1)/pi;
```

Програма функції $f_2(x) = \frac{4}{\pi}\left(\frac{\sin x}{1} + \frac{\sin 3x}{3}\right)$ (друга частинна сума ряду) має вигляд:

```
 function F=F2(x)
%Second partial sum of the Fourier series
%              4*(sin(x)/1 +sin(3*x)/3 +sin(5*x)/5  +...)/pi
%
% Copyright Гаєв Є.О.

F= F1(x)+4*(sin(3*x)/3)/pi;
```

Утворивши файли *F1.m* і *F2.m* з таким змістом, ми навчаємо MATLAB "розуміти" ці функції як його власні, внутрішні. Саме тому друга програма і працює, що вона посилається на *F1.m* як на внутрішню функцію. Можемо тепер: (*i*) обчислити значення новостворених функцій для будь-якого аргументу, навіть для векторного; (*ii*) обчисливши масив аргументів *x* і масив відповідних значень функції *F1* або *F2* командою *plot* можна побудувати їхні графіки; (*iii*) застосувати також команду автоматичної побудови графіку *fplot*. Ось приклад використання новостворених функцій для побудови графіків з командного вікна:

```
>> fplot('f1(x)', [-2*pi 2*pi],'b')
>> hold on; fplot('f2(x)', [-2*pi 2*pi],'g')
```

>> hold on; *fplot*('*f3*(*x*)', [*-2*pi 2*pi*],'*r*')

– отримаємо графіки першої (синя крива), другої (зелена), третьої (червона) і т.д. частинної суми ряда (3.3). Це одне з можливих розв'язків першого завдання лабораторної работи 3 з посібника [3]. Інші можливі розв'язки наведено нижче. (***Увага***: аби функція працювала для векторних аргументів, не забувайте пересвідчитись, що кожна операція тіла функції правильно сприймає вектори, тобто де потрібно – використано "операції із крапкою"!);

Вважаємо створеними аналогічні функції від аргументу x *F3.m, F4.m* і т. д. до *F10.m*, що являють собою відповідні частинні суми ряду (3.3) до десятої частинної суми включно. Цікаво дізнатися, чи дійсно вони наближають розривну граничну функцію (3.4) і наскільки близько. Звернемося до програмування функції (3.4).

Приклад 3.2. Запрограмувати розривну функцію (3.4) для відрізку $x \in [-\pi, \pi]$.

Наводимо програму (*Лістінг*), що частково вирішує проблему (далі зрозуміло, чому лише "частково"). Дамо функції (3.4) ім'я *step*(*x*), враховуючи її вигляд.

Лістінг 3.2А (*m*-файл *step.m*)

```
function  F=step(x)
% Функція ступінчата з рівнянням   (3.4)
%              / =-1;   якщо x \in [-pi, 0]
%   F(x)=
%              \ = 1,    якщо x \in (0, pi]
%
% Copyright Гаєв Є.О.
   if x<=0
      %   disp( ' SubProgram X<0' )
      F=-1;
   else
      %   disp( ' SubProgram X>0' )
      F=1;
   end
```

Щойно отримана функція середовища MATLAB *step* дозволяє одержати значення функції (3.4) лише для скалярного аргументу, наприклад *step*(*-.5*)=*-1*, *step*(*.5*)=*1*. Але, для вектора *t= - 3 : 3* замість, відповідно, семи значень маємо *step*(*t*) =*1* (тобто значення лише для останнього елементу вектора *t*)! Це не дозволить побудувати графік цієї функції командою *plot*, проте команда

>> fplot('*step'*,[*-5 5 -1.1 1.1*])

потрібний графік побудує. (Спробуйте! Подумайте ще раз над різницею між двома командами *plot* і *fplot*).

Запропонована проста програма включає невідомі ще читачеві команди середовища MATLAB. Так, зустрічаємо команду *disp*, корисну для налагоджування програм. Про неї, та чому вона "закоментована" знаком *%* (тобто система все одно її в обчисленнях не враховує) розповімо далі. А зараз – про оператори умовного виконання.

3.2.1. ОПЕРАТОРИ УМОВНОГО ВИКОНАННЯ

Оператори умовного виконання *if*, *else* та пов'язаний з ними *ifelse*– важливий інструмент програмування, що дозволяє надати програмі елементи штучного інтелекту. Загальна форма оператора умовного виконання є такою:

if *Умова 1*
Група операторів 1
elseif *Умова 2*
Група операторів 2
else
Група операторів 3
end
(3.5)

В складеному операторі (3.5) *Група операторів* – це послідовність будь-яких команд MATLAB (дії над величинами, побудова графіків тощо), а *Умова* – це якийсь математичний запис, що може бути ***вірним*** (*true*) або ***невірним*** (*false*). Ще кажуть, що *Умова* може приймати лише два логічних значення, відповідно ***1*** або ***0***.

Оператор (3.5) завжди має починатися з ***if*** і закінчуватися ***end***, і працює таким чином. Коли MATLAB, послідовно виконуючи всі попередні команди програми, дійшов до ***if***, з якого починається блок команд (3.5), то перш за все перевіряється *Умова 1*. Якщо вона має логічне значення ***1***, тобто ***вірно***, то MATLAB виконує *Групу операторів 1*, після чого приступає до операторів, що написані нижче за ***end***. Всі інші умови і оператори (3.5) тоді ніякої ролі не відіграють.

Якщо *Умова 1* приймає значення ***0*** (тобто ***false***, ***невірно***), то MATLAB пропускає без всякої уваги *Групу операторів 1* і звертається до перевірки *Умови 2*. Якщо остання має значення ***1*** (***true***, ***вірно***), то виконується *Група операторів 2* і далі знову "управління передається" на оператор, слідуючий за ***end***. *Група операторів 3* в цьому випадку ніякої ролі не відігравала.

Проте, коли *Умова 2* ***невірна*** (має значення ***0***, *false*), то *Група операторів 2* пропускається без уваги і MATLAB виконує *Групу операторів 3*, а потім знов-таки "покидає оператор" ***if … end*** і виконує наступний за ним.

З наведеного детального опису має стати зрозумілою і робота кожної із скорочених форм оператора умовного виконання (3.5):

```
if Умова                          if Умова 1
   Група операторів                  Група операторів 1
end                               else
                                     Група операторів 2
                                  end
```

В усіх випадках кожне ***if*** повинно закінчуватись відповідним ***end***! Редактор *m*-файлів автоматично розміщує складові цього оператора у вигляді "сходинок", аби покращити сприйняття тексту програми. Кожному радимо дотримуватись такого правильного стилю програмування! Корисно також переглянути приклади, подані в *Help*.

Наведена теорія дозволяє зрозуміти програму функції *step*. Якщо десь у командному рядку MATLAB зустріне вираз *step*(*t*), то він звертається до своєї робочої області пам‘яті і шукає там *m*-файл із іменем *step*. Не знайшов – видає інформацію

??? Undefined function or variable ' step '.

В такому випадку ви зробили щось невірне; можливе пояснення і виправлення – див. розділ 3.3. Знайшов MATLAB потрібну функцію – замість аргументу програми *x* підставляє надану величину, тобто *x=t*, і виконує запрограмовані дії: якщо надане значення *x* менше або дорівнює нулю, то вихідній змінній *F* надається значення *F= -1*, і з цим значенням виходить із "внутрішнього середовища" команди у загальне середовище MATLAB; якщо ж умова *x <= 0* була невірною, то вихідній змінній надається значення *F= 1*, що якраз і відповідає нижній гілці *x > 0* функції (3.4).

Вираз ***Умова***, яка є аргументом операторів *if* або *elseif*, може бути простим, що використовує один з можливих ***операторів відношення***

$$A == B, \quad A < B, \quad A <= B, \quad A > B, \quad A >= B, \quad A \sim= B. \tag{3.6}$$

A і *B* – це довільні арифметичні вирази, яким програма вже має надати числових значень. Тоді написані умови означають відповідно "*дорівнює*", "*менше*", "*менше або дорівнює*", "*більше*", "*більше або дорівнює*", "*не дорівнює*". Розглянемо, наприклад, останню умову *A~= B*. Вона має значення ***1*** (***true***), якщо *A* і *B* дійсно такі, що $A \neq B$; проте, умова приймає значення ***0*** (***false***), якщо *A = B*. Значення інших форм умови так само легко зрозуміти. Однак ***Умова*** може бути і складною, складаючись з виразів (3.6) за допомогою ***операторів логіки***

Умова1 **&** *Умова2*, *Умова1* **|** *Умова2*, **~** *Умова1*,

які читаються відповідно "*Умова1* **і** *Умова2*", "*Умова1* **або** *Умова2*", "**не** *Умова1*". Це означає, що перша з написаних вище складних умов (її називають ***логічним множенням*** або ***AND***) приймає значення ***1*** лише тоді, коли *Умова1* і *Умова2* мають значення ***1*** одночасно. Друга складна умова (***логічна сума*** або ***OR***)

приймає значення ***1*** у двох випадках: коли *Умова1* має значення ***1*** незалежно від *Умови2* і коли *Умова2* має значення ***1*** незалежно від *Умови1*. Третя складна умова (її називають ***запереченням*** або ***NOT***) приймає значення ***1*** коли *Умова1* має значення ***0***, і приймає логічне значення ***0*** коли *Умова1* має значення ***1***. Більш докладно співвідношення між вхідними і вихідним значеннями функцій логіки надає Табл.. 3.1.

Таблиця 3.1. Значення функцій логіки ***AND***, ***OR*** та ***NOT***

Значення *A* та *B*		*A & B*	*A | B*	*~ A*
1	***1***	***1***	***1***	***0***
1	***0***	***0***	***1***	***0***
0	***1***	***0***	***1***	***1***
0	***0***	***0***	***0***	***1***

Приклади використання складних логічних умов подано далі; ще більше прикладів можна знайти в довідниковій системі *Help* та у книгах [1-12]. Розглянемо ще нові оператори, які часто використовують у програмуванні.

3.2.2. ОПЕРАТОРИ УПРАВЛІННЯ

Існує кілька **операторів управління** обчислювальним процесом –

for ... end, *while… end*,
switch ... case ... otherwise ... end,
try ... catch ... end, *break*

та інші. Тут зупинимось на двох перших, найбільш поширених. Якщо, з набуттям досвіду програмування, виникне потреба в інших операторах, познайомитись з ними можна у довіднику *Help*.

Оператори управління потрібні для організації циклів повторювальних обчислень, наприклад – дозволяють модифікувати програму *step*, аби вона правильно працювала і для векторного аргументу. Для цього текст програми слід трохи змінити, як показано у наступному лістінгу, який радимо зберегти у *m*-файлі *step1*:

Лістінг 3.2Б (*m*-файл *step1.m*)

```
function  F0=step1(x)
% The same Step Function as in "STEP.m"
% but modified for vectorized arguments
%
% Розробник Нестеренко Б.М., 10.01.01
L=length(x);
 for i=1:L
  if x( i )<=0
   F0( i )=-1;
    else
   F0( i )=1;
    end
 end
```

Розглянемо роботу програми *step1*. Перш за все, вона використовує функцію *length* (див. розділ 1.4), що дозволяє надати *L* числового значення кількості елементів у вхідному векторі *x*. Таку саму кількість елементів повинен мати вихідний вектор *F0* (він може називатись і *F*, значення не має). Кожний з елементів вхідного вектора *x* аналізується по черзі у циклі *for* (7-й рядок програми) … *end* (13-й рядок програми). Такий оператор називають оператором з визначеною кількістю циклів**. Змінна циклу**[*)]** *i* приймає спочатку значення *1*, обирає *x*(*1*) з елементів вектора *x* і аналізує його так само, як в програмі *step*, надаючи вихідній змінній *F0*(*1*) відповідного значення *–1* або *1*. Потім таку саму роботу програма проводить для *x*(*2*) і надає певного значення *F0*(*2*), і так – для всіх *L* елементів вхідного вектора. Коли вся робота виконана, програма повертає управління в зовнішнє середовище MATLAB разом із змінною *F0*, яка стає вектором з *L* елементів. Тепер можна використати програму *plot* для побудови графіку функції (3.4):**

```
>> t= -3*pi : .01 : 3*pi;  F=step1(t);   plot(t,F);  axis( [ - 3  3  - 1.1  1.1 ] )
```

[*)] Оскільки символу *i* надаються певні значення 1, 2 і т.д., то він втрачає значення "по умовчуванню" $\sqrt{-1}$

Відмітимо, що змінною *L* можна користуватись усюди "в середині" програми *step1*, однак значення її, як і кожної іншої змінної, створеної в тілі програми-функції, зникає, як тільки управління повертається до командного рядка. Це зручно, бо жодна із зовнішніх змінних, навіть з таким самим іменем, не впливатиме на роботу функції. Проте існують програмні засоби для порушення цього загального правила (див. нижче).

Щойно створена програма *step1* будує функцію (3.4) фактично на інтервалі $x \in (-\infty, \infty)$, тобто надає значення *F= –1* для всіх $x \le 0$ і значення *F= +1* для всіх $x > 0$. Однак вона мала б задаватись рівнянням (3.4) лише на інтервалі $x \in [-\pi, \pi]$ і бути подовженою періодично (з періодом $T = 2\pi$) на всю числову вісь. Програма-функція такої математичної функції буде більш складною.

Приклад 3.3. Побудувати функцію (3.4), періодичну на числовій осі.

Перш за все слід **виробити алгоритм** знаходження значення $F(x)$ для довільного значення аргументу x. Пропонується такий алгоритм: якщо нам надано якесь число x, то перевіряємо його належність до інтервалу $[-\pi, \pi]$; якщо x належить цьому інтервалу – відповідна функція *step* або *step1* надасть $F(x)$ потрібного значення, *-1* або *1*. Нехай тепер x не належить інтервалу $[-\pi, \pi]$. Треба визначитись, якого значення набуде F, якщо функцію (3.4) періодично пересувати на 2π вліво та вправо. У цьому, фактично, і полягає алгоритм: якщо x додатній, то віднімаємо від нього період 2π і перевіряємо, чи потрапив результат в інтервал $[-\pi, \pi]$. Не потрапив? Ще раз віднімаємо і

перевіряємо належність результату до інтервалу. Зрозуміло, що рано чи пізно потрапить, але кількість операцій віднімання заздалегідь невідома. Як тільки результат віднімання потрапив у контрольний інтервал, то визначимо значення $F(x)$ вже створеним алгоритмом *step*.

Якщо ж вихідний x від'ємний, $x<0$, то додаємо до нього $T=2\pi$ і перевіряємо потрапляння в контрольний інтервал $[-\pi,\pi]$. Так робимо, аж поки не потрапимо, а тоді – визначимо $F(x)$ алгоритмом *step*.

Оскільки кількість операцій віднімання–додавання невідома заздалегідь, у програмі слід використати оператор *while*. Наведемо текст програми, збережений, скажімо, в " *step_pi.m*":

Лістінг 3.3 (*m*-файл *step_pi.m*)

```
function  F=step_pi(x)
% Periodic Step Function,
% i.e. periodical continuation of the Step Function
%          / =-1; якщо  x \in [-pi, 0]
% F(x)=
%          \ = 1, якщо  x \in (0, pi]
% to be compared  with partial sums of corresponding
Fourier Series

% Розробник Гаєв Є.О., 10.01.01
L=length(x);
for i=1:L  % Початок циклу № 1 по елементах вхідного вектора
  xx=x(i);
```

```
   if  (xx >= -pi) & (xx < pi)   % Перевірка положення xx відносно [-pi, pi]
     % disp('x потрапив в інтервал [-pi, pi]');
       F(i)=step(x(i));
   elseif  xx <= 0 %  Якщо x не потрапив та від'ємний
       % disp(' X < 0 ');
         nT=0;
       while ~ ((xx >= -pi) & (xx < pi))
% 2) Цикл доки xx потрапить в інтервал
                nT=nT+1;   xx=xx+2*pi ;
       end                                        % Кінець циклу 2)
       F(i)=step(xx);
   else                %   Якщо x не потрапив та додатній
       % disp(' X > 0 ');
             nT=0;
       while ~((xx >= -pi) &(xx < pi ))
% 3) Цикл доки xx потрапить
              nT=nT+1;   xx=xx-2*pi;
       end                                        % Кінець циклу 3)
      F(i)=step(xx);
   end   %  Закінчення перевірки положення xx відносно інтервалу [-pi, pi]
end           % Кінець циклу № 1
%
%
%-------------------------------------------------------------
%  SubFunction, допоміжна функція
%-------------------------------------------------------------
function  F=step(x)
% Step Function дорівнює -1  якщо x \in [-pi, 0]  u = 1, якщо x \in (0, pi]
```

```
if x<=0
   %   disp( ' SubProgram X<0' )
   F=-1;
else
   %   disp( ' SubProgram X>0' )
   F=1;
end
```

Пояснення до програми *step_pi*.

Після коментарів у рядках з першого по сьомий, роль яких вже пояснено, іде тіло програми. Оскільки ця програма вже доволі складна, слід починати її аналіз із з'ясування того, із скількох циклів та логічних блоків вона складається, тобто який оператор *end*, що має закінчувати блоки та цикли, до якого "відкриваючого оператора" *for*, *while* або *if* відноситься. Розташування операторів "сходинками", що його автоматично зробив редактор *m*-файлів, дозволяє легко встановити наступне:

- програма складається з зовнішнього циклу, що розпочинається з *for* в 11-му рядку і закінчується останнім *end* в 33-му рядку; названі оператори пояснено коментарями "Початок циклу № 1 по елементах вхідного вектора" і "Кінець циклу № 1".
- всередині цього зовнішнього циклу знаходиться оператор умовного виконання *if* (13-й рядок) **…** *end* (32-й рядок) в його повній формі (3.5).

- серед операторів, які виконуются після умови *elseif*, присутній цикл №2, що починається з *while* у 19-му рядку і закінчується *end* у 22-му.
- ще один "вкладений" цикл №3 виконується після умови *else*. Це також є цикл з невизначеною кількістю обертів *while* (27-й) **…** *end* (30-й рядок).

З урахуванням зазначеної структури програми *step_pi* розберемо її роботу. Цикл №1 виконується однаково для кожного з L елементів вхідного вектора. Для кожного i-го елемента $x(i)$ утворюємо службову змінну *xx* (на всякий випадок, аби не змінити x) і її вже аналізуємо. Якщо $xx \in [-\pi, \pi]$, то умова оператора виконана і у 15-му рядку програми відповідному елементу вихідного вектора *F(i)* надається значення відомою вже програмою *step*, управління передається "закриваючому" операторові *end* у 32-му рядку і починається аналіз *i+1*-го елемента вхідного вектора. Якщо ж потрапляння в контрольний інтервал не сталося, то діємо в залежності від того, від'ємне *xx* чи додатнє. У першому випадку виконується цикл № 2, у другому – цикл № 3. На виході з цих циклів змінній *F(i)* надається значення програмою *step*. Про роль "закоментованих" команд розповідається далі.

Тепер можна побудувати графік періодичної на всій числовій осі функції (3.4) такою командою:

```
>> t=-5*pi :8*pi/1000 :3*pi; F=step_pi(t); plot(t,F), axis([-5*pi, 4*pi, -1.1 1.1])
```

Цей графік показано кривою 0 на рис. 3.2. Стає зрозумілою дана тут назва програми-функції *step* (*сходинка*).

Після закінчення роботи програми *step_pi* в зовнішнє середовище видається вектор *F* такої самої довжини, як вхідний вектор *x*. А от змінні *L*, *xx*, *nT* (лічильник циклів) "забуваються" – у цьому суттєва відмінність програми-функції від програми-скрипту. Нижче буде дано приклад використання щойно створених функцій і скрипту.

Програми-функції *step1* і *step_pi* дають приклади використання двох типів операторів управління. Інформацію про інші, менш поширені, оператори управління разом з прикладами використання можна отримати командами допомоги *help switch*, *help try*, *help break.*

3.2.3. ПІДПРОГРАМИ. ГЛОБАЛЬНІ ЗМІННІ

Ще декілька важливих властивостей програм-функцій. В програмі- *step_pi* є звертання до функції *step*. Таке звертання *step*(*xx*) нічим не відрізняється від звертання до будь-якої внутрішньої функції системи MATLAB на зразок *sin*(*xx*) або *exp*(*xx*), припускається лише, що відповідна функція визначена в середовищі MATLAB. Але не варто визначати функцію *step*(*xx*), чи будь-яку іншу, якщо вона не має самостійного значення і вживається лише у зв‘язку з даною функцією. Тому в версіях системи MATLAB починаючи з v.6, введено можливість використовувати **підпрограми**, і саме

така можливість використана в Лістінгу 3.3. Наприкінці програми-функції *step_pi*, у тому ж файлі *step_pi.m* написано ще функцію *step*, яка і є підпрограмою і оформлена у відповідності з загальними правилами MATLAB (з єдиною відмінністю, що інформація про неї не буде видана командою *help step*).

Тим не менше, запитайте MATLAB командою *help step* – і отримаєте інформацію про іншу функцію *step*, навіть про кілька функцій *step*, що належать до пакетів *LTI*, *IDmodel*, *IDdata*. Це вказує на важливу для нас властивість MATLAB: потрібну для виконання програми-функції інформацію він шукає спочатку всередині її тексту (і тому, в нашому випадку, виконує саме нашу функцію *step*!) і лише потім, якщо не знайде, шукає ще десь.

Ще зверніть увагу: підпрограму *step* написано в найпростішому варіанті, розрахованому лише на поодинокі числа, а не вектори. Але ж складніше і не треба, бо звертання до неї *F*(*i*)=*step*(*xx*) іде для поодинокого числа *xx*!

Підпрограм може бути кілька в одному *m*-файлі. Порядок їхнього розташування ніякої ролі не відіграє.

Інколи виникає потреба порушити **конвенцію щодо локальної ролі змінних**, породжених усередині програми-функції. Наприклад, вам потрібно знати (або навіть використати в наступній програмі) скільки зсувів *nT* основного інтервалу

було зроблено. Тоді на початку *кожної* з програм треба перерахувати всі спільні, **глобальні** змінні, тобто поставити рядок з описом змінних на зразок

global nT X Y Z

Таку ж команду треба виконати в командному рядку, якщо передбачається використовувати там змінні *nT*, *X*, *Y* та *Z* як глобальні (спільні з якоюсь програмою).

Глобальні змінні використовують, наприклад, при створенні графічного інтерфейса користувача (GUI, Graphical User Interface), див. [2]. Подробиці можна знайти в довідковій системі *Help* за ключовим словом *global*, або прямо з командного вікна MATLAB:

help global та *help isglobal*.

3.3. НАЛАГОДЖЕННЯ ПРОГРАМ. ДЕЯКІ ПОРАДИ

Навіть у досвідчених програмістів написана вперше програма не завжди починає правильно працювати відразу. Ще треба її налагодити (англійською – *to debug*). Редактор *m*-файлів MATLAB v.6 пропонує тут певну допомогу. Наведемо деякі поради щодо перевірки і налагодження ваших програм. Наприкінці – кілька зауважень щодо використання української або російської мови в MATLAB.

Якщо після запуску вашого скрипту або використання програми-функції ви отримуєте замість очікуванного результату якийсь коментар, що розпочинається словом "*Error*:", то це свідчить про *синтаксичні* помилки у вашій програмі. Цей коментар видається, як правило, червоними літерами і включає певну підказку, де шукати помилку. Наприклад:

??? Error: *File*: *D*:*Matemat**work**Berm4271.m Line: 7 Column: 11

")" expected, *";" found.*

Виданий текст означає, що інтерпретатор MATLAB (який перекладає вашу програму на машинну мову) чекав від вашого тексту закриття дужечки (***")" expected***), але дійшов до кінця командного тексту (***";" found***), так і не знайшовши її. Інтерпретатор радить шукати помилку в 7-му рядку в 11-й позиції вказаного файлу (*Berm4271.m Line: 7 Column: 11*).

Інший можливий коментар

??? Undefined function or variable 'ya'.

означає, що інтерпретатор не може надати значення змінній *'ya'*: або ви забули написати оператор на зразок *ya=* , або програмі слід взяти її значення зовні, для чого змінну *ya* слід зробити глобальною. Можуть бути і більш складні причини; як їх знайти – див. поради нижче.

Коли всі синтаксичні помилки виправлено і програма ніби-то починає працювати, це ще не гарантує, що вона працює вірно. В більшості випадків слід пересвідчитись, що всі змінні обчислюються правильно і приймають вірні значення.

Складайте вашу велику програму з певних програмних модулів; кожний з модулів слід перевірити окремо. Перевіряйте модулі на штучних комбінаціях вхідних даних, обраних таким чином, що певні проміжні величини легко обчислюються вручну. Тоді в програмі безпосередньо за цими "контрольованими" величинами слід тимчасово ставити оператори виводу даних (оператори друку). Такі оператори використано, наприклад, в програмі *step* (Лістінг 3.2А):

disp(*'Текстовий_Рядок'*)

Вказана команда *disp* виводить на екран комп'ютера той *Текстовий_Рядок*, який в лапках стоїть аргументом команди. Така видача дозволяє зорієнтуватися, чи програма пройшла відповідне місце у ній, чи ні. Можливі і такі використання програми друку: *disp*(*A*), або *disp A*; буде надруковано числове значення величини *A*. Проте, той самий результат одержимо, якщо просто скажемо *A* в тексті програми, або після присвоєння значення не поставимо крапку з комою: *A=a*, *B=в*;. Такі "контрольні видачі" в програмі *step* (лістінг 3.2А) дозволили нам упевнитись, що оператор логіки *if … else … end* дійсно працює так, як передбачалося. Після перевірки програмного модуля контрольні видачі можна викреслити з текста програми або "закоментувати" знаком *%*. Це можна зробити або вручну, або за допомогою меню редактора програм ***Text\comment***, або, навпаки, за допомогою ***Text\uncomment*** викреслити цей знак. Редактор програм також дозволяє дізнатися значення, яке набула та чи інша змінна після виконання програми. Для цього треба навести на цю змінну курсор, виділити її і в меню радактора обрати ***Text\Evaluate Selection***. Команда *disp* може бути також використана для повідомлення користувачу про той чи інший випадок під час розрахунків. Так використано команду

disp('*Матриця не є квадратною! Помилка!*')

в програмі *MyDeterminant* розрілу 3.5.

Аналогічна команда виводу може допомогти в організації простого "діалогу" з програмою. Наприклад, виконання програми

X=input('*Please input value of X*')

призведе до того, що на екрані з'явиться запрошення ввести значення для *X*; далі програма чекає на таке введення, після цього надає *X* відповідне значення і продовжує роботу. Приклад з простим діалогом між програмою і користувачем, який здійснюється з командного рядочка, дає програми *MyFFT* (лістінг 4.6). Більш складний діалог через спеціальні графічні вікна можна організувати на основі *Graphical User Interface* (*GUI*). Приклад такого діалогу для пошуку емпіричного рівняння дано в розд. 4.7. Подробиці див. в книзі [2].

Не налагоджена нова програма може інколи "зациклитися", тобто нескінченно довго "обертатись" по певному колу команд без видач будь-якого результату. Так буває при обчисленнях за ітераційною схемою, коли програміст забуває передбачити випадок розбіжності ітерацій. Аби перервати роботу програми і зробити комп'ютер знов керованим, використовують комбінацію клавіш

< *Ctrl – C* >.

Читач, напевне, помітив, що лише частину коментарів до програм ми подали українською мовою. Ми змушені так робити, бо MATLAB не завжди "розуміє" українську або російську. Якщо ви спробуєте вводити у командному рядку навіть коментар

>> *% Проблемна Ситуація –*

все буде гаразд, аж поки ви не натиснене літеру ***c***-маленьке; з великим ***C*** – все гаразд. Використання коментарів з символами кирилиці в скриптах і програмах-функціях призводить до повідомлення про помилки, хоча їх немає насправді. Тому

користуйтеся латинськими літерами, поки програмісти не пристосують MATLAB до потреб нашого ринку. В меню графічного вікна *Figure*, однак, труднощів з кирилицею майже немає. Пункт меню ***Insert\Title*** дозволить написати такий, наприклад, заголовок над графіками: "*Обєм елiпсоїда*". (Проблема, як бачимо, лише з апострофом).

Якщо ви працюєте не на власному комп'ютері, а в комп'ютерному класі загального користування, у вас можуть виникнути проблеми зі збереженням ваших програм, даних і результатів розрахунків. Справа у тому, що Системний Адміністратор надасть вам лише обмежені *права доступу*. Часто ви будете мати право зберігати ваші файли лише в директорії *TEMP* та на дискеті *A*. Оскільки інші студенти також зберігають файли в *TEMP*, радимо вам суворо дотримуватися певного порядку. Доцільно створити в *TEMP* піддиректорію для всієї групи студентів, скажімо *FIT-203*, всередині якої кожний студент створює піддиректорії вже для власних файлів. Ці піддиректорії можна назвати прізвищами студентів – *Kozenko*, *Mironchenko* тощо. Не знищуйте файлів в чужих директоріях! Проте, на всяк випадок, робіть копії важливих файлів на власну дискету.

Може статися, однак, що MATLAB "не захоче" бачити створені вами програми в директорії, скажімо, *TEMP/Kozenko*, або на дискеті. Це означає, що дані директорії треба "прописати" в спеціальному переліку директорій *PATH*. Спробуйте самі (у разі потреби – просіть Системного Адміністратора) зробити наступне. В меню MATLAB оберіть ***File / SetPath…***. У вікні *SetPath*, що з'явиться, натисніть клавішу ***Add Folder…***. З'явиться "драбина" з переліком усіх директорій і дисків комп'ютера. Оберіть директорію з вашими файлами, або диск *A*, та натисніть *OK* – вашу директорію буде вказано в загальному переліку вікна *SetPath*. Якщо покинути це вікно за допомогою *Close*, ваше налагодження буде дійсним лише протягом даного сеансу роботи – і часто цього достатньо. А натискання *Save* потрібно в разі довготривалої роботи з MATLAB.

Якщо потрібно зберегти ваші результати розрахунків (змінні у вигляді масивів чи структур) для наступного сеансу роботи – в меню оберіть

File / Save Workspace As…

У вікні, яке з'явиться, оберіть директорію, де зберігаєте ваші дані, або диск *A*, і надайте файлу даних певне ім'я (доцільно використати або ваше прізвище, або дату роботи, скажімо, *Dezember7*). Буде утворено двійковий файл *Dezember7.mat*. Наступного разу для "пригадування" збережених даних просто оберіть з меню ***File/Open…***, або натисніть "іконку"  і оберіть потрібний файл. Як прочитати MATLAB дані з інших програм, див. в розділі 4.7.

Іноді доцільно зберегти не лише дані, а і певний запис ваших дій під час роботи з MATLAB, аби пізніше можна було б пригадати, як ви впорались з тією чи іншою задачею. MATLAB буде "нотувати" всі ваші дії в текстовому файлі *diary*, якщо з командного рядка виконати команду *diary*. "Щоденник" вашої роботи під час MATLAB-сеансу також доцільно назвати датою вашої роботи; тоді команду слід виконати у вигляді *diary*('*Dezember7'*), назва файлу в лапках.

3.4. ПРИКЛАД: АНАЛІЗ ЗБІЖНОСТІ РЯДУ ФУР'Є

Використаємо створені в даному розділі програми для аналізу (вже більш досконалого з точки зору програмування) математичної задачі лабораторної роботи 3 з методичного посібника [3], запропонованої раніше. Нагадаємо, що створено скрипт *visualize* і підпрограми-функції *F1*, *F2*, **…** *F10*, а також програму побудови періодичної функції (3.4), і все це дозволяє дослідити, чи дійсно ряд Фур'є (3.3) наближає розривну функцію (3.4), і як швидко збігаються до неї частинні суми ряду (3.3). Виконуємо в MATLAB послідовність команд аналогічно переліку (3.1), призначення яких пояснюється коментарями:

```
>> Xbegin=-3*pi ;  Xend=2*pi ;  N=1000 ;
>> Step= (Xend - Xbegin)/N ;
>> X= Xbegin : Step : Xend ;                % Аргумент
>> F0=step_pi(X); F1=f3(X); F2=f5(X); %Вектори значень 3-х функцій
>> visualize
>> F1=f8(X); F2=f10(X);                     % Вектори значень ще 2-х функцій
```

>> visualize

Результатом роботи будуть два графічних вікна з графіками, які показані на рис. 3.2. Стрілки і пояснюючі надписи на рисунках зроблено вже в самих графічних вікнах, так само збільшено товщину ліній. Лінія у вигляді сходинки, що задається рівнянням (3.4), помічена цифрою 0. Можна зробити такі математичні висновки.

Частинні суми ряду (3.3), хоча і є неперервними функціями, дійсно поступово наближають функцію (3.4), розривну в точках $x=(2k+1)\pi$, де k пробігає значення $k=\pm1,\pm2,\pm3,\ldots$. Старші частинні суми *f8*(*x*) і *f10*(*x*) краще наближають функцію (3.4), аніж частинні суми *f3*(*x*) і *f5*(*x*). Звідси, зрозуміло, не випливає, що з більш старшими частинними сумами наближення ще покращиться – строго таке можна показати лише математичним доведенням. Проте, геометричне дослідження на гатунок наведенного для, скажімо, *f20*(*x*) і *f30*(*x*) і т.д. може бути цікавим!

Критично налаштований читач може помітити, що навіть для старших частинних сум все одно лишаються помітні "сплески" наближуючих функцій в точках розриву. Так, це явище було вперше помічено відомим фізиком Гібсом [39]. А ви звернули на це увагу тоді, коли вивчали ряди Фур‘є у курсі вищої математики?

3.5. ПРИКЛАД: ОБЧИСЛЕННЯ ВИЗНАЧНИКІВ

Дамо ще один корисний приклад складної програми MATLAB. Обчислення визначника – поширена математична операція. Уявіть собі, що ви отримали завдання розробити програму для його обчислення. Ваша робота має складатись з таких етапів.

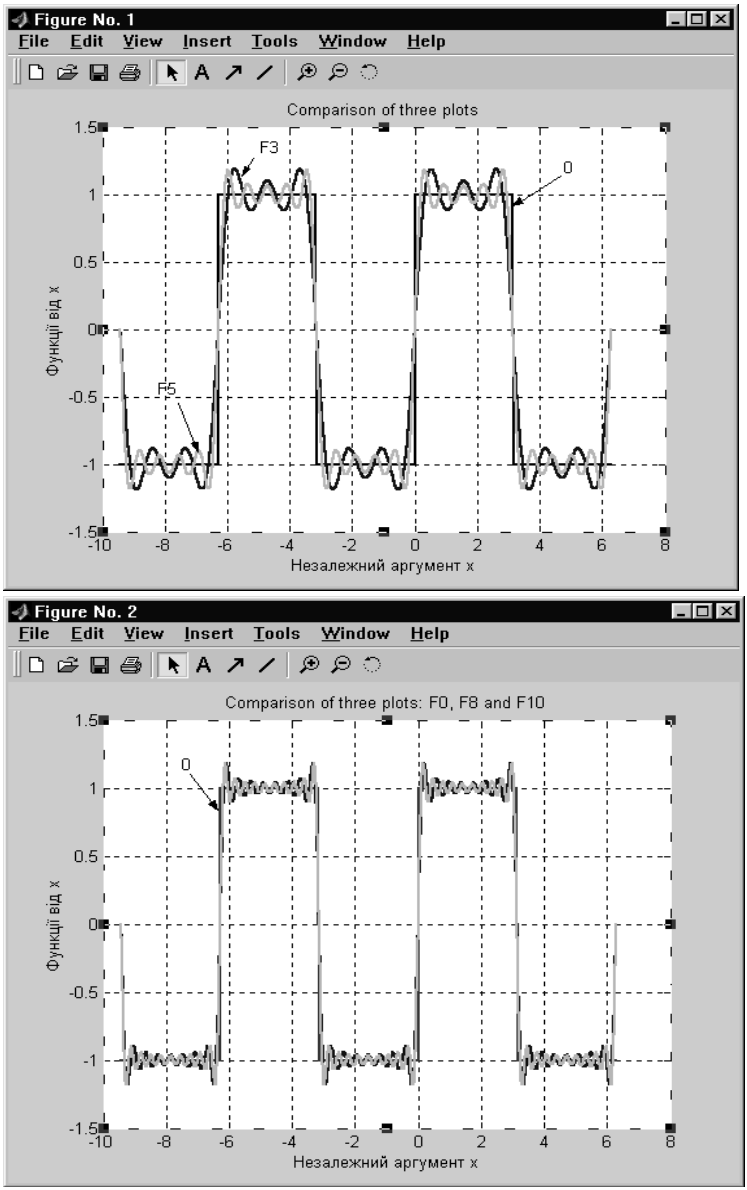


Рис. 3.2. Співставлення третьої і п‘ятої (уверху), восьмої і десятої (унизу) частинних сум ряду (3.3) з граничною функцією (3.4), позначеною номером *0*

Етап 1. Для кожної задачі, фізичної чи математичної, спочатку слід пригадати зміст та властивості понять, які вона використовує. Визначник матриці – це число, яке обчислюється за заданою таблицею чисел A:

$$\det(A) = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}. \qquad (3.6)$$

Відомо, що визначник не змінюється, якщо від елементів певного стовпчика відняти відповідні елементи іншого стовпчика, попередньо помножені на довільне число (так само для рядків). Якщо такими діями таблицю чисел перетворити таким чином, аби всі члени певного рядка, за винятком одного, скажімо a_{pq}, дорівнювали 0 (нулю), то визначник дорівнюватиме $(-1)^{p+q} a_{pq}$, помноженому на визначник меншого на одиницю порядку [34]. Далі так само перетворюємо цей останній визначник, аж поки отримаємо "визначник першого порядку", тобто число. Пояснимо таку техніку на прикладі [32, с.266]:

$$\Delta_5 = \begin{vmatrix} 3 & 1 & \underline{-1} & 2 & 1 \\ -2 & 3 & \underline{1} & 4 & 3 \\ 1 & 4 & \underline{2} & 3 & 1 \\ 5 & -2 & \underline{-3} & 5 & -1 \\ -1 & 1 & \underline{2} & 3 & 2 \end{vmatrix} = \left\{ \begin{array}{l} \text{Третій стовпчик,} \\ \text{помножений на 3,} \\ \text{додаєм до першого;} \\ \text{помножений на 1} \\ \text{до другого; і т.д.} \end{array} \right\} = \begin{vmatrix} 0 & 0 & \underline{\underline{-1}} & 0 & 0 \\ 1 & 4 & 1 & 6 & 4 \\ 7 & 6 & 2 & 7 & 3 \\ -4 & -5 & -3 & -1 & -4 \\ 5 & 3 & 2 & 7 & 4 \end{vmatrix} =$$

$$= -1 \cdot (-1)^{1+3} \begin{vmatrix} \underline{1} & 4 & 6 & 4 \\ \underline{7} & 6 & 7 & 3 \\ \underline{-4} & -5 & -1 & -4 \\ \underline{5} & 3 & 7 & 4 \end{vmatrix} = \left\{ \begin{array}{l} \text{З усіх стовпчиків} \\ \text{віднімаємо перший,} \\ \text{помножений на} \\ \text{множники 4, 6 і 4} \end{array} \right\} =$$

$$= - \begin{vmatrix} \underline{1} & 0 & 0 & 0 \\ \underline{7} & -22 & -35 & -25 \\ -4 & 11 & 23 & 12 \\ 5 & -17 & -23 & -16 \end{vmatrix} = -(-1)^{1+1} \begin{vmatrix} \underline{-22} & -35 & -25 \\ \underline{11} & 23 & 12 \\ \underline{-17} & -23 & -16 \end{vmatrix} = \left\{ \begin{array}{l} \text{Вихідним приймаємо} \\ \text{перший стовпчик,} \\ \text{помножений на} \\ \text{множники } \frac{35}{22} \text{ і } \frac{25}{22} \end{array} \right\} =$$

$$= - \begin{vmatrix} \underline{\underline{-22}} & 0 & 0 \\ 11 & 5{,}5 & -0{,}5 \\ -17 & 4{,}1 & 3{,}3 \end{vmatrix} = 22 \cdot (-1)^{1+1} \begin{vmatrix} 5{,}5 & \underline{-0{,}5} \\ 4{,}1 & \underline{3{,}3} \end{vmatrix} = 22 \begin{vmatrix} 0 & \underline{\underline{-0{,}5}} \\ 40{,}4 & \underline{\underline{3{,}3}} \end{vmatrix} =$$

$$= 22 \cdot (-0{,}5) \cdot (-1)^{1+2} \cdot \left| (40{,}4) \right| = \left\{ \begin{array}{l} \text{Останній} \\ \text{множник -} \\ \text{визначник від} \\ \text{матриці } 1 \times 1 \end{array} \right\} = 22 \cdot 0{,}5 \cdot 40{,}4 = \ 444{,}4$$

(Відносна похибка цього результату 0,4% зумовлена округленням деяких обчислень).

Сказане дозволяє "вручну" діяти за наступним алгоритмом.

Етап 2. "Ручне" обчислення заради кращого оволодіння алгоритмом. Єдина відмінність: при ручному обчислені за "основний" (підкреслений) обирався стовпчик, за можливістю, з одиницею на горі; тепер "основним" завжди буде перший стовпчик кожного чергового визначника. Наводимо алгоритм.

0. Робимо копію *Am* заданої матриці *A* оператором *Am= A*, аби не зруйнувати її в результаті перетворень.

1.1. Виконуємо команду

*p=1; q=p+1; Am(p:end,q)=Am(p:end,q)-Am(p:end,p)*Am(p,q)/Am(p,p)* (3.7)

Отримуємо увесь другий стовпчик *q=2,* перетворений за потрібним правилом.

1.2. Повторюємо команду (3.7), поклавши, однак, $q = q + 1$. Перетворено 3-й стовпчик матриці *Am*.

1.3. Так робимо доти, аж поки q не стане дорівнювати n, вимірності матриці *A*. Отримуємо в результаті нову матрицю *Am*, в якій в першому рядку всі нулі, за винятком $a_{11} \neq 0$. Кількість команд, яких було виконано, дорівнює *n-1*.

2. Повторюємо дії 1.1 – 1.3 за формулою (3.7), починаючи з *p=2*. В матриці *Am* маємо вже два "нульових" рядки. Команду (3.7) було повторено *n-2* разів.

3. Так само "отримуємо нулі" у третьому, четвертому і т.д. рядках. В результаті – матриця *Am* має "трикутний" вигляд з нулями вище головної діагоналі.

4. Визначник дорівнює добутку чисел, які стоять на головній діагоналі.

(Порівняйте результат з обчисленням за "внутрішньою" програмою MATLAB *D=det*(*A*)!)

Етап 3. Програмування в MATLAB

Створюємо *m*-файл із ім′ям *MyDeterminant.m* зі змістом:

```
function d=MyDeterminant(X)
% A Test Program for calculating Determinant of
% a Square Matrix A of its Dimension n
[n,m]=size(X); % size – внутрішня програма MATLAB
if n~=m % Перевірка, чи є матриця квадратною
   disp('Dimensions are not square! You are wrong!');
else % Якщо матриця дійсно квадратна
  Am=X;
for p=1:n
  for q=p+1:n
     Am(p:end,q)=Am(p:end,q) - Am(p:end,p)*Am(p,q)/Am(p,p);
%  pause % Закоментована команда для тестування даної програми
  end
end % Отримано матрицю з нулями вище головної діагоналі
v=diag(Am); det=prod(v)
end
```

Якщо з командного вікна наказати

d=MyDeterminant(*A*),

де *A* – дана матриця, то програма отримує вектор v з діагональними елементами перетвореної матриці і видає їх добуток – значення визначника *d*. Якщо в цій програмі "розкоментувати" команду *pause* і прибрати попередню кому з крапкою, то програма буде

виводити перетворену матрицю і зупинятись після обчислень кожного із стовпчиків. Натискання на *Ввод* продовжує обчислення. "Розкоментована" команда *pause* – ще одна корисна команда для налагодження програм; у даному випадку вона робить наочним поступове перетворення матриць.

Таким чином, в даному розділі ви навчилися програмувати прості скрипти і програми-функції, використовувати оператори умовного виконання *if – else – end* і оператори управління обчислювальним процесом *for – end*. На закінчення пропонуємо лабораторну роботу 4 з посібника [3] для закріплення навичок програмування.

3.6. КОНТРОЛЬНІ ПИТАННЯ ДО РОЗДІЛУ 3

1. Які якості методів обчислення, що розглянуті в розділі 2, слід використовувати для програмування обчислень? Які види програм існують в MATLAB? Чим вони відрізняються щодо вхідної і вихідної інформації? Як викликати редактор *m*-файлів для написання програм MATLAB? Як записують *коментарі* в програмах MATLAB, для чого вони призначені? Чого слід очікувати, запрошуючи *help* для щойно створеної програми?
2. Як має виглядати програма-сценарій (*script*)? Якщо у скрипті створено якусь нову змінну, чи зберігається її значення після закінчення роботи скрипту? Чи можна у скрипті використовувати змінну, створену зовні? Як програми-сценарії використовують в MATLAB?
3. Як має виглядати програма-функція, з чого слід її починати? Якщо у тілі функції створено якусь нову змінну, чи зберігається її значення після закінчення роботи скрипту? Чи можна у функції використовувати змінну, створену зовні? Як можна передавати значення змінних із зовні у функцію і, навпаки, з функції у загальне середовище MATLAB? Як у MATLAB слід звертатись до програми-функції? Скільки вхідних значень має функція? Скільки вихідних?
4. Які ви знаєте оператори умовного виконання? Які вони мають формати запису? Поясніть смислове значення ключових слів цих операторів *if*, *elseif*, *else*, *end*. Яких значень можуть набувати

умови, що пишуть безпосередньо за ключовими словами *if* і *elseif*? Чи можуть ці умови бути складними? Які логічні операції використовують для запису складних умов? Як значення складної умови залежить від значень операндів?

5. Які ви знаєте оператори управління обчислювальним процесом? Чому ключові слова тих чи інших циклів *for*, *while*, *switch*, *try* мають закінчуватись ключовим словом *end*?
6. У чому полягає налагодження програми?

4. ВНУТРІШНІ ПРОГРАМИ MATLAB ДЛЯ ТИПОВИХ ЗАДАЧ МАТЕМАТИЧНОГО МОДЕЛЮВАННЯ

Вище розглянуто розв'язування в MATLAB доволі складних задач (хоча і з "стандартного набору" вищої та обчислювальної математики). Середовище MATLAB використовувалося при цьому як розвинений калькулятор або як інтегрована система програмування таких задач. Проте MATLAB не можна було б вважати "універсальним математичним середовищем", якби він не мав внутрішніх засобів розв'язування таких саме задач. В практиці інженера, науковця або студента під час виконання курсової та дипломної роботи на старших курсах слід спиратися саме на ці засоби. Найчастіше за усе потрібну задачу можна розв'язати простою командою MATLAB або застосуванням певної функції із вхідними аргументами. Проте, насправді, ці засоби є складними програмами, що враховують усі можливі випадки і особливості задачі, яка розв'язується, і оптимізовані з урахуванням найновітніших досягнень світової математики і програмування. Сподіваємося, що читачі, які засвоїли матеріал розділів 2 і 3, тепер уявляють собі, які алгоритми сховано за "чорними скриньками" команд. І головне – усвідомлюють можливі складнощі, коли в даній задачі може не існувати розв'язку, або, навпаки, існують декілька; які додаткові умови треба (і навіщо) повідомити MATLAB для успішного вирішення проблеми на підставі стійкого алгоритму. Таке усвідомлення дає ключ до сприйняття можливих застережень MATLAB (*warnings*) або навіть повідомлень про помилки (*errors*).

В даному розділі повернемося до розглянутих вже типових задач математичного моделювання, але викладемо, нарешті, "внутрішні" засоби MATLAB. Саме їх – наголосимо – слід використовувати в практичній роботі; з них, як з цеглинок, складати розвинені програми для задач будь-якої складності. MATLAB пропонує також готові спеціалізовані пакети програм (команд) – ***Toolbox***'и – для певних професійних груп. Такими є

- *Power System Blockset* для електриків і енергетиків,
- *Filter Design Toolbox* для аналізу сигналів та розробки фільтрів певних властивостей,

- *Neural Network Toolbox* для певних задач медицини нейронів і штучного інтелекту,
- *Fuzzy Systems Toolboz* для задач з нечіткими множинами і нечіткою логікою,
- *Simulink* для "графічного моделювання" складних динамічних систем,
- *Stateflow, Real-Time Workshop , CDMA Reference Blockset* для алгоритмів безпровідної комунікації,
- *Communications Toolbox* для аналізу складних радіо- або телефонних сигналів,
- *Control System Toolbox* для розробки та аналізу систем управління,
- *Data Acquisition Toolbox* для обробки експериментальної (аналогової) інформації,
- *Statistics Toolbox* для задач математичної статистики,
- *GARCH Toolbox* для аналізу економічних питань,
- *Financial Toolbox* для прийняття рішень у фінансовій сфері і аналізу відповідної інформації та інші.

Багато з них "перетинаються" один з іншим, але кожен насичений спеціальними термінологією і методами. В даному посібнику немає місця для відповідних тем, радимо звертатися до книг у наведеному наприкінці списку [1,4,8]. А якщо ви раптом не можете знайти придатний готовий математичний інструментарій – не важко поступово розробити власний.

Отже, звернемось до типових задач математики і до універсальних команд MATLAB, що їх розв‘язують.

4.1. ЛІНІЙНА АЛГЕБРА І СИСТЕМИ ЛІНІЙНИХ АЛГЕБРАЇЧНИХ РІВНЯНЬ

Найпоширеніші задачі лінійної алгебри – обчислення визначника даної матриці A, а також матриці A^{-1}, оберненої до неї. З курсу вищої математики відомо, що це доволі трудомісткі обчислення, для проведення яких розроблена велика кількість методів. В MATLAB, однак, результат можна отримати командами

det(A) для $|A|=$ та *inv*(A) для A^{-1}.

(імена функцій *det* і *inv* походять від *determinant* та *inverse*). Якщо на першу операцію MATLAB дає відповідь

??? *Error using ==> det*
Matrix must be square.

то це означає, що матриця A не є квадратною (бо визначники існують лише для них). Якщо ми матрицю A не знаємо (вона отримана, припустимо, в результаті розрахунків), то узнати її виміри допоможе команда з розділу 1.4

[*n*,*m*]=*size*(A).

Матриця є квадратною, коли *n*=*m*.

Обернена до A матриця A^{-1} визначається як така, що $A^{-1} * A = E = A * A^{-1}$, де одинична матриця E вимірності *n* в MATLAB утворюється командою *eye*(*n*). Якщо запросити обернену матрицю від прямокутної, то маємо

??? *Error using ==> inv*
Matrix must be square. Матриця має бути квадратною!

Як відомо з вищої математики, обернена матриця не існує за умови $|A| = 0$. Якщо ж визначник матриці близький до нуля (матриця "погано обумовлена", singular matrix), то більшість методів обчислення стають нестійкими – їхній результат чомусь (може, помилка у математичній моделі?) суттєво залежить навіть від незначних похибок. MATLAB тоді попереджає:

Warning: Matrix is singular to working precision.

Якщо усе ж таки за фізичним змістом задачі матриця і має бути такою, пошукайте якийсь особистий метод для погано обумовлених матриць.

Приклад 4.1. Обчислити визначник і обернену матрицю для матриці $A = \begin{vmatrix} 1 & 2 & 3 & 4 \\ 1^2 & 2^2 & 3^2 & 4^2 \\ 1^3 & 2^3 & 3^3 & 4^3 \\ 1^4 & 2^4 & 3^4 & 4^4 \end{vmatrix}$. Обчислення можна провести за такою схемою:

```
>> st1=[1  2  3  4];          % перший рядок вихідної матриці
```

```
>> st2= st1 .^2;  st3= st1 .^3;  st4= st1 .^4;     % степені першого рядка
>> A=[st1 ;  st2 ; st3 ; st4 ];        % утворення вихідної матриці
>> D=det(A) ,  A1=inv(A)          % визначник і обернена матриця
      D=288
      A1=   4.0000   -4.3333    1.5000   -0.1667
           -3.0000    4.7500   -2.0000    0.2500
            1.3333   -2.3333    1.1667   -0.1667
           -0.2500    0.4583   -0.2500    0.0417
```

Обидва результати отримано! Перевіримо останній з них множенням на вихідну матрицю:

```
>> A1*A
ans =      1.0000    0.0000   -0.0000    0.0000
          -0.0000    1.0000    0.0000   -0.0000
           0.0000    0.0000    1.0000    0.0000
                0         0    0.0000    1.0000
```

тобто отримали одиничну матрицю.

Велика кількість задач з фізики, математики, економічного планування та інших важливих наук приводять до так званої **проблеми власних значень** та **власних векторів** даної квадратної матриці A. Проблема полягає у тому, щоб знайти такі скаляри λ і *нетривіальні* (тобто відмінні від нуля) вектори x, для яких множення матриці на x дає λx. Відомо, що проблема власних значень приводить до алгебраїчного рівняння степеня n, який дорівнює виміру матриці, тобто $n=length(A)$. Масив коефіцієнтів цього алгебраїчного рівняння (у вигляді вектора коефіцієнтів біля λ^k в порядку зменшення степеня від n до 0) дає команда $p=poly(A)$. Рівняння має n розв'язків (у сенсі, про який йдеться у розділі 4.2). Для розв'язування задачі MATLAB має функцію *eig* (від *eigenvalues*). Її застосування з одним вихідним аргументом $d = eig(A)$ дає вектор власних значень d; застосування з двома вихідними аргументами у формі $[V,D] = eig(A)$ – дає матрицю D з власними значеннями на головній діагоналі і одночасно матрицю V, стовбчиками якої є власні вектори. Метод обчислень програма "обирає самостійно", виходячи з властивостей матриці.

Приклад 4.2. Обчислити власні значення і власні вектори матриці з прикладу 4.1.

Командою [*V, D*]=*eig*(*A*) отримуємо дві матриці *V* і *D* з потрібними даними. Тоді *x1=V*(**:** ,*1*), …, *x4=V*(**:** ,*4*) – власні вектори і *lamb1=D*(*1,1*) … *lamb4=D*(*4,4*) – власні числа заданої матриці *A* з першого по четвертий. Можна пересвідчитись, що вектор *A*x1* дорівнюватиме *lamb1*x1*. Так само відповідають наведеному вище визначенню власних векторів і власних чисел *x2=V*(**:** ,*2*) і *lamb2=D*(*2,2*) та інші власні вектори і відповідні ним власні числа. (Не наводимо матриць, що отримано; радимо зробити ці обчислення самостійно).

Системи лінійних алгебраїчних рівнянь (СЛАР) – одна з найбільш поширених математичних моделей. Обмежимось тут випадком систем (2.3), коли кількість рівнянь *n* співпадає з кількістю невідомих *m*. В MATLAB можна реалізувати різні методи розв‘язування таких систем. Оскільки вже вміємо обчислювати визначники – можемо йти за правилом Крамера (2.4). Інший спосіб – за допомогою оберненої матриці.

Відомо, що СЛАР (2.4) можна записати в матричному вигляді в одній з форм

$$Ax = b \qquad \text{або} \qquad yA' = c. \qquad (4.1)$$

В першому випадку, що співпадає з (2.1), x і b є векторами вимірності $n \times 1$, тобто стовбцями. В другому випадку y і c є векторами з тих же елементів, але вимірності $1 \times n$, тобто рядками. Ці останні вектори-рядки можна отримати з векторів-стовбців операцією **транспонування** – перетворення стовпчика в рядок або навпаки, яка позначається штрихом над символами: $y = x'$ і $c = b'$ або $x = y'$ і $b = c'$. В залежності від того, яку форму зображення СЛАР обрано (невідома справа чи зліва), розв'язок системи в MATLAB реалізується командами

$$x=inv(A) * b \qquad \text{або} \qquad x' = b' * inv(A). \qquad (4.2)$$

Існування розв'язку гарантовано за умови $det(A) \neq 0$.

Формулами Крамера або командами (4.2) ми "нав'язуємо" MATLAB певний метод розв'язування матричних рівнянь (4.1). В

розділі 2, однак, пояснювалося, що такі методи можуть бути неефективними з точки зору швидкості виконання або використання пам'яті комп'ютера. Два інших методи було викладено в розділах 2.1.1 і 2.1.2 і запропоновано їх запрограмувати у розділах 3.5 і 3.6. Існують і інші методи. Деякі з них розроблено для матриць спеціального вигляду, наприклад, матриць, серед елементів яких багато нулів ("*рідкі*", *sparse* матриці). Розробники MATLAB "дозволили" йому самому обирати метод розв'язування, якщо звертатися до MATLAB командами:

$$x = A \setminus b \quad \text{і} \quad y = c / A, \tag{4.3}$$

відповідно до форми запису (4.1).

Ці команди, вперше наведені в табл.. 1.1, нагадують звичайне ділення чисел і були б очевидними, якби (4.1) були числовими рівняннями (матриця коефіцієнтів СЛАР A є завжди нібито в знаменнику, бо на неї "нахиляється" риска – *обернений слеш* або *слеш*). Для першої форми запису кількість стовбчиків чисельника і знаменника мають співпадати; отримуємо вектор-стовбчик. У другому записі має співпадати кількість рядків – і отримуємо вектор-рядок.

Команди *обернений слеш* і *слеш* у записах (4.3) – винайдені саме розробниками MATLAB, бо в звичайній математиці ділення матриць не визначено. Записи (4.3) викликають насправді той метод розв'язування, який програма "зважатиме" найкращим для вказаної матриці.

Приклад 4.3. Розв'язати систему рівнянь для даної в прикладі 4.1 матриці A і вектора-стовпчика вільних членів b=[*1 2 3*]′.

Для лівого рівняння слід використати операцію *обернений слеш*. Маємо:

```
>> x=A \ b
    x =   1.0000
         -0.5000
          0.3333
```

Такий же розв'язок отримаємо з використанням *слешу*, але з транспонованими даними:

```
>> y=b'/A'
y =    1.0000   -0.5000    0.3333
```

Перевірка результату: множення $A*x$ дає вектор b; множення $y*A'$ дає транспонований вектор b.

4.2. ЗНАХОДЖЕННЯ КОРЕНІВ МНОГОЧЛЕНІВ

Серед різноманітних нелінійних рівнянь особливе місце посідають алгебраїчні рівняння n-го степеня

$$a_0 x^n + a_1 x^{n-1} + a_2 x^{n-2} + ... + a_{n-1} x + a_n = 0 \qquad (a_0 \neq 0), \qquad (4.4)$$

де функція $P(x) = a_0 x^n + a_1 x^{n-1} + a_2 x^{n-2} + ... + a_{n-1} x + a_n$ називається **поліномом**, або **многочленом**). Такі рівняння дуже часто виникають у математичній фізиці. До них призводять, наприклад, пошук загального розв′язку однорідного диференціального рівняння зі сталими коефіцієнтами, задача на власні значення матриці та інші. Алгебраїчні рівняння (4.4) – це той рідкісний випадок нелінійних задач, коли математиці вдалося розробити загальну теорію. Тут удається повністю характеризувати проблему існування і кількості розв'язків. Наведемо тут основні відомості. Обмежуємося лише тим випадком, коли коефіцієнти рівняння a_0, a_1, …, a_n є дійсними числами.

Алгебраїчне рівняня n-го степеня вигляду (4.4) може мати як дійсні, так і комплексні **корені**. Якщо цікавитись лише дійсними коренями многочлена $P(x)$, то корисно пам′ятати, що їх може бути **не більше**, ніж степінь многочлена n (у тому числі, дійсних коренів може не бути взагалі).

Якщо комплексне число $x = a + bi$ є коренем многочлена $P(x)$, то спряжене комплексне число $\bar{x} = a - bi$ також є коренем цього полінома [32]. Звідси випливає таке твердження: **якщо степінь полінома непарний, то він обов′язково має хоча б один дійсний корінь**.

Найбільш узагальнено проблему вирішує **основна теорема алгебри**: алгебраїчне рівняння n-го степеня (4.4) має рівно n коренів, якщо дійсні і комплексні корені враховувати стільки разів, яка їх кратність**.**

Поняття *кратності кореня* требя пояснити. Якщо a є дійсним коренем многочлена (з дійсними коефіцієнтами) $P(x)$, то останній можна "націло розділити" на $x-a$; результатом при цьому буде новий поліном $P_1(x)$ степеня на 1 менше, тобто степеня $n-1$. Припустимо, таке "ділення націло" можливо для $(x-a)^k$, але вже не можливо для $(x-a)^{k+1}$, де k ціле (зрозуміло, що k більше за 1, оскільки a є корінь). Тоді такий корінь називають k-кратним дійсним коренем.

Для комплексних коренів $x=a+bi$ має місце аналогічне положення: поліном $P(x)$ ділиться націло на квадратний тричлен $x^2-2ax+(a^2+b^2)$ *). (Зрозуміло, що дискримінант останнього від'ємний, бо інакше він, і $P(x)$ разом з ним, мав би дійсні корені). Комплексний корінь називається k-кратним, якщо $P(x)$ ділиться на вказаний квадратний тричлен у степені k.

Наведеною теорією слід користуватись під час пошуку коренів поліномів. Для цього в математиці розроблені спеціальні методи, з якими можна ознайомитись в підручнику [31]. Проте, MATLAB з цією наукою "добре обізнаний" – його команда *roots* видає всі корені поліномів як дійсні, так і комплексні. При цьому аргументом команди має бути вектор, який складається з $n+1$-го коефіцієнта многочлена, взятих в порядку зменшення степеня (4.4). (Радимо також пригадати матеріал розділу 1.6).

Приклад 4.4. Обчислити корені многочлена

$$x^4-288x^3+2810x^2-2868x+288$$

*) Оскільки $P(x)$ має ділитись одночасно на $x-x_*$ і на $x-\overline{x}_*$, де $x_*=a+bi$ і $\overline{x}_*=a-bi$, і тому – на добуток $x-x_*$ і $x-\overline{x}_*$

(взято *характеристичний многочлен* матриці A з прикладу 4.1, тобто такий, що своїми коренями має її власні числа).

Будуємо вектор з п'яти коефіцієнтів *p*=[*1 -288 2810 -2868 288*] і командою *r=roots*(*p*) отримуємо вектор з чотирьох коренів заданого поліному

```
>> r=roots(p)
     r =  277.9265
            8.9315
            1.0292
            0.1127
```

На підставі сказаного в розділі 1.6 можна пересвідчитись, що поліном, побудований за такими коренями, приводить саме до заданого многочлена:

```
>> poly(r)

   ans =        1.0e+003 *
   0.0010  -0.2880   2.8100  -2.8680   0.2880
```

(в першому рядку видано множник 1000 для всіх наступних коефіцієнтів). Переведемо відповідь з чисельної форми в символічну і надамо їй більш красивої форми:

```
>> poly2sym(ans)
ans = x^4-288*x^3+2810*x^2-2868*x+288
>> pretty(ans)
```

$$x^4 - 288\,x^3 + 2810\,x^2 - 2868\,x + 288$$

4.3. КОРЕНІ НЕЛІНІЙНИХ РІВНЯНЬ З ОДНИМ НЕВІДОМИМ

На відміну від алгебраїчних рівнянь, загальна теорія нелінійних рівнянь відсутня. Можна лише сказати, що **одне рівняння з кількома невідомими** має, скоріше за все, багато розв'язків. Справді, всім невідомим, окрім одного, можемо надати довільні значення – отримуємо одне рівняння з одним невідомим; надамо інші значення – маємо інше рівняння, і так отримуємо довільну кількість рівнянь, кожне з яких може мати корінь. Проте, рівняння з кількома невідомими може і не мати розв'язку. Для

кожного окремого рівняння, таким чином, потрібне певне дослідження існування і єдиності розв'язку!

В другому розділі описано розв’язування нелінійних рівнянь діленням відрізка навпіл, а також методи ітерацій, Ньютона і метод хорд. Існують і інші методи, які реалізовано в MATLAB-програмі *fzero* (від *find zero* певної функції $f(x)$). В усіх можливих випадках передбачається, що окрім функції $f(x)$, для якої треба знайти корінь, програмі повідомляється ще певна інформація про наближене значення кореня *x0*. Ось найбільш поширений формат цієї команди:

x=fzero(**'***Функція***'**, *x0*)

Якщо *x0* є число, то програма сприймає його за наближене значеня кореня і шукає спочатку значення *x1*, де функція змінює знак, тобто виконується нерівність

$$Функція(x0)* Функція(x1) < 0. \qquad (4.5)$$

Потім програма уточнює корінь і видає на екран значення *x=корінь*. Якщо їй це не вдалося, програма видає *NaN* (*not a number*). Однак, не поспішайте з висновком, що корінь не існує (див. нижче).

Якщо *x0* задається як вектор з двох чисел – програма сприймає його за кінці інтервалу, на яких функція змінює знак (4.5). Тоді знаходження кореня Вам гарантовано. Якщо це не так, програма видасть

??? Error using ==> fzero
The function values at the interval endpoints must differ in sign.

Для задання функції-аргументу *Функція* є декілька засобів. Найпростіший – задання функції аналітичним виразом у лапках.

Приклад 4.5. Знайти корені трансцендентного рівняння $\cos x = k x$ з прикладу 2.1 для $k = 0.1$

1). Початкове значення кореня задамо навмання *x0=.5*. Застосування стандартної команди видає корінь рівняння:

```
>> x=fzero( ' cos(x) -.1*x ' , .5)
   x =  1.4276
```

Перевіряємо його підстановкою в функцію:

```
>> cos(x) -.1*x
   ans = -5.2550e-005,
```

тобто маємо нуль з точністю до четвертого знаку.

2). Візьмемо, однак, інше початкове значення *x0= -.5*:

```
>> x=fzero(' cos(x)-.1*x ', -.5)
    x =  -1.7463
```

Підстановка цього результату в задану функцію також засвідчує її обертання в нуль:

```
>> cos(x) -.1*x
    ans =  -5.5511e-017,
```

тобто той факт, що знайдений *x* дає ще один розв'язок заданого рівняння.

3). Ще корені отримуємо з іншими початковими наближеннями, наприклад:

```
>> x=fzero(' cos(x) -.1*x ', 4)
    x =   5.2671
```

Чим пояснити таку кількість знайдених коренів? Відповідь дає графічна побудова функцій $y = \cos x$ і $y = kx$, наведена на рис. 2.1. Вона виявляє, що залежно від значення k, рівняння може мати різну кількість розв'язків. Наведений приклад обгрунтовує, чому формальний розв'язок нелінійного рівняння MATLAB слід, де можливо, перевіряти графічною побудовою.

Відмітимо ще іншу форму команди розв'язування рівняння, з чотирма вихідними аргументами

```
[ x , fval , exitflag , output ] = fzero( ' Функція ', x0)
```

яка видає не лише корінь *x* рівняння *Функція=0*, а також значення функції у знайденому корені *fval*, кількість ітерацій і інформацію про обраний метод розв'язування в масиві *output*.

4.4. ЗВИЧАЙНІ ДИФЕРЕНЦІАЛЬНІ РІВНЯННЯ: ЗАДАЧІ КОШІ

Як ми вже знаємо з розділу 2.5, різноманітність математичних моделей з участю звичайних диференціальних рівнянь (ЗДР) в основному складається з задач двох видів – **задач з початковими умовами** (*Initial Value Problems*; їх ще називають задачами Коші) і **задач з крайовими умовами** (граничних задач, *Boundary Value Problems*). Чисельне розв'язування останніх часто

базується на методах розв'язування перших. В такому порядку і розглянемо їх тут.

4.4.1. ЗАДАЧІ З ПОЧАТКОВИМИ УМОВАМИ

Формулювання задачі звучить таким чином: знайти функції $y_1(t)$, $y_2(t)$, …, $y_n(t)$ однієї змінної t (її умовно можна називати **часом**), які задовольняють певну систему звичайних диференціальних рівнянь (її приймають у так званій *нормальній формі*, див. розділ 2.5.1)

$$y_1' = f_1(t, y_1, ..., y_n), \ \ldots, \ y_n' = f_n(t, y_1, ..., y_n) \qquad (4.7)$$

і систему початкових умов (значень функції, які розшукуються, в початковий момент t_0)

$$y_1(t_0) = y_{10}, \ \ldots, \ y_n(t_0) = y_{n0}. \qquad (4.8)$$

Для доволі необтяжливих умов для функцій f_1, f_2, …, f_n доведено, що розв'язок такої задачи існує в якомусь проміжку часу $[t_0, t]$ і він єдиний, якщо кількість рівнянь збігається з кількістю функцій, що розшукуються. Чисельні методи Рунге-Кутта для розв'язування задач Коші (4.7) – (4.8) було розглянуто вище; в MATLAB вони реалізовані в програмах *ode45* і *ode23* (назви є скороченням від *Ordinary Differential Equation*). *Help* вважає, що *ode45* слід використовувати як першу спробу. Програма *ode23* більш підходить для "чутливої" системи рівнянь. Для задач, ще більш чутливих до випадкових похибок, програма *ode113*, побудована на методі Адамса, є більш надійною. І для задач "дуже чутливих", наприклад, для так званих *жорстких рівнянь* (*stiff ODE*) кращими є програми *ode15s*, *ode23s*, *ode23t* і *ode23tb*. Всі названі програми використовуються одноманітно, що дозволяє тут обмежитися лише двома першими, *ode45* і *ode23*.

Єдина форма звернення до програми розв'язування початкової задачі має вигляд:

$$[t, y] = ode45(odeFUNCTION, Interval, y0); \qquad (4.9)$$

В такому запису вхідний аргумент *odeFUNCTION* – це посилання на n функцій в правих частинах рівнянь (4.7), пояснене далі. Другий вхідний аргумент *Interval* – вектор прийнамні з двома елементами. Якщо таких елементів лише два – програма інтерпретує їх як початкове t_0 і кінцеве значення часу $t_к$ і інтегрує рівняння (4.7) від початку до кінця. Якщо вектор *Interval* має більше ніж два елементи (у порядку зростання або зменшення!), то програма інтерпретує їх як контрольні проміжні точки і обов'язково видає значення розв'язку і в них. Третій вхідний аргумент *y0* – вектор з n початковими значеннями (4.8).

Зміст вихідних аргументів майже очевидний: *t* – вектор-стовпець з моментами часу між t_0 і $t_к$, для яких обчислено значення вихідного аргументу; *y* – матриця розв'язку, кожний рядок якої має n елементів $y_1(t)$, $y_2(t)$, …, $y_n(t)$ – значення розв'язку в момент часу з відповідного рядка *t*. Це означає, що кількість стовпчиків цієї вихідної матриці *y*(: ,*1*), *y*(: ,*2*), …, *y*(: ,*n*) відповідає довжині вектора *t* і дають вони якраз таблиці шуканих функцій. Матрицю виводити не слід, бо вона має бути занадто великою, тому у запису (4.9) не забувайте ставити крапку з комою. Проте, стовпці матриці дозволяють побудувати графіки розв'язку. Розглянемо приклади.

Приклад 4.6. Нехай треба розв'язати задачу Коші для рівняння

$$y' + \frac{1 - p\,x}{x^2}\,y = 1 \qquad (4.10)$$

для різних початкових значень $y(x_0) = y_0$, $x_0 = 0.2$ і значення параметра $p = 2$ (брати $x_0 = 0$ не можна, бо ліва частина тоді не визначена).

1). Перш за все слід надати рівнянню нормального вигляду: $y' = 1 - \frac{1 - p\,x}{x^2}\,y$. Праву частину цього рівняння програмуємо в файлі *Berm3956.m* (назва стає зрозумілою з подальшого):

Лістінг 4.1. Зміст файлу *Berm3956.m*

```
function dydt=Berm3956(t , y )
% Права частина ЗДР No.3956, Берман [30], тобто
% dydx+(1-2*x)*y/x^2  =1.
% Чисельний розв'язок треба отримати для початкових значень
% y(0)= y0  командою [t ,y]=ode45( ' berm3956 ' ,[x0  xK] , [y0]);
% Можна порівняти з аналітичним розв'язком
% запрограмованим у функції  Berm3956a
%  p – Параметр рівняння
global  p
dydt=1- ( 1 - p*t ) .* y ./ t .^ 2 ;
```

Програма складається фактично з трьох рядків, все інше – коментарі. Останній рядок програми описує, як пов'язати вхідні змінні t і y з вихідною змінною *dydt*. Потрібне для обчислень значення параметра p передається у програму із зовні за допомогою опису цієї змінної як *global*.

2). Далі надаємо значення параметру p і початкового значення шуканій невідомій функції, наприклад $p = 2$ і *y0=3*. Саме для них будуємо чисельний розв'язок на проміжку часу *Interval*=[**.** *2 2*] командами

```
>> global p ;  p=2 ; [t3 , Ynum3 ]=ode45('berm3956' , [ .2  2] , 3) ;
```

(Увага: параметр *p* об'явлено глобальним, аби передати його значення в середину програми-функції *berm3956*). Тепер можна побудувати графік розв'язку:

```
>> plot( t3 , Ynum3 )
```

3). Робити аналіз розв'язку цікаво не лише для поодинокої кривої, а для **родини розв'язків**, кожне з яких репрезентує певне початкове значення *y0*. Тому повторимо попередні дії для значень *y0=1*, *2* і *4*, відмічаючи відповідні пари { *t* , *Ynum*} додатковими номерами:

```
>> [t1, Ynum1]=ode45( 'berm3956' , [ .2  2] , 1);
>> [t2, Ynum2]=ode45('berm3956' , [ .2  2] , 2);
>> [t4, Ynum4]=ode45('berm3956' , [ .2  2] , 4);
>> plot(t1,Ynum1,'r', t2,Ynum2,'g', t3,Ynum3,'b', t4,Ynum4,'k')
>> legend(' y_0=1', '2', '3', ' y_0=4')
```

(всі наступні рази MATLAB вже пам'ятає глобальне значення *p*). Результат показано на рис. 4.1.

4). Обов'язково слід ставити питання про достовірність отриманого результату. Відкриємо "таємницю": ця задача із збірника Бермана [30] №3965 для $p = 2$, де і знаходимо загальний вигляд розв'язку $y = Cx^2 e^{1/x} + x^2$. Для довільного $y_0 = y(0,2)$ знаходимо сталу $C = \dfrac{y_0 - x_0^2}{x_0^2 e^{1/x_0}}$. Відповідну формулу запрограмуємо як функцію MATLAB:

Лістінг 4.2. Зміст файлу Berm3956a.m

```
function Y=Berm3956a(t, t0, y0)
% Аналітичний розв'язок ЗДР No.3956 [30] в залежності від часу t
% dydx=(1-2*x)*y/x^2,  y(t0)=y0
% для порівняння з чисельним розв'язком з Berm3956.m
C=(y0 - t0^2)/(t0^2 *exp(1/t0));  Y=C*t.^2.*exp(1./t) + t.^2;
```

Побудовою аналітичної кривої для $y_0=3$

```
>> Tan=.2:.05:2; Yan=Berm3956a(Tan, .2, 3);
>> hold on;  plot(Tan, Yan, 'r')
```

впевнимось, що аналітичний розв'язок цілком співпадає з чисельної кривою 3.

Приклад 4.7. Розв'язати диференціальне рівняння другого порядку $y'' + 2y' + 5y = -\dfrac{17}{2}\cos 2x$ залежно від початкових умов для функції $y(0) = y_0$ і її похідної $y'(0) = y_0'$.

(Задачу взято зі збірника задач [30], №4271; її аналітичним розв'язком є $y = e^{-x}(C_1 \cos 2x + C_2 \sin 2x) - \dfrac{1}{2}\cos 2x - 2\sin 2x)$.

1) Рівняння в нормальній формі має вигляд системи двох рівнянь

$$y' = z,$$

$$z' = -2z - 5y - \frac{17}{2}\cos 2x, \tag{4.11}$$

з початковими умовами $y(0) = y_0$ і $z(0) = y_0'$ (введено нову змінну $z = y'$). Тому праві частини системи зберігаємо у файлі *Berm4271.m.*

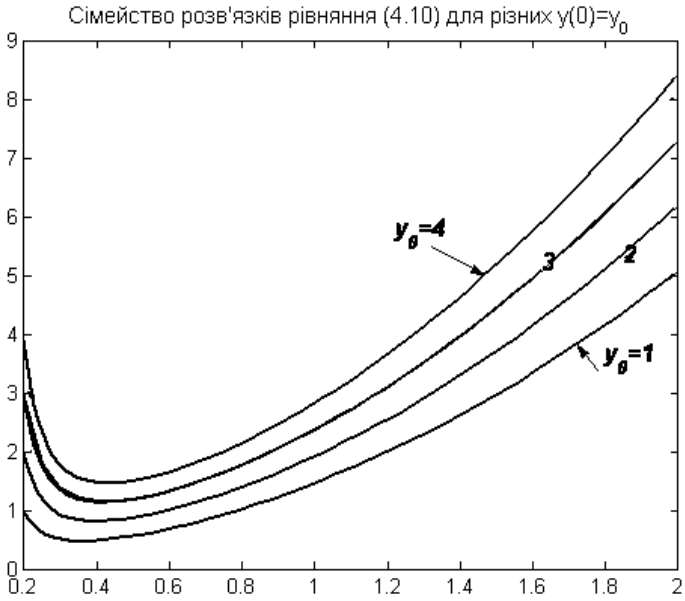


Рис. 4.1. Родина розв'язків рівняння (4.10) для початкових значень $y_0 = 1, 2, 3$ і 4, а також порівняння з аналітичним розв'язком для $y_0 = 2$

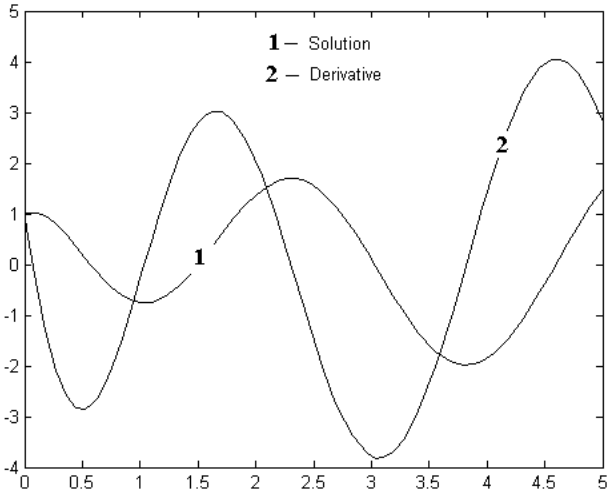


Рис. 4.2. Інтегральна крива рівняння (4.11) і її похідна

Лістінг 4.3. Зміст файлу *Berm4271.m*:

```
function dydt=Berm4271(t , y)
%Праві частини ЗДР 2-го порядку No.4271 [29], а саме
% dzdx=(-17/2)*cos(2*x)-2z-5y,    dydx=z,
% Чисельний розв'язок для початкових даних y(0)=c1, z(0)=c2
%  викликати командою
%  [t,y]=ode45('berm 4271', [0 5], [c1; c2]);
% Аналітичний розв'язок запрограмовано в Berm4271a .
dydt=[y(2); (-17/2)*cos(2*t)-2*y(2)-5*y(1)];
```

Призначення цієї програми-функції – передати зв'язок між незалежною змінною x (в програмі вона введена як *t*), двох шуканих змінних y і z (в програмі введені як *y*(*1*) і *y*(*2*)) з векторною змінною *dydt*, яка і є значеннями похідних згідно з рівняннями (4.11).

Тепер можна викликати одну з команд розв'язування задачі Коші для тих чи інших початкових значень *y0* і *z0*. Нехай обидві останні дорівнюють *1*:

```
>> y0=1; z0=1;  [t, y]=ode23('berm4271',  [0 5], [y0; z0]);
```

Графік шуканої функції і її похідної отримуємо командою

```
>> plot(t,y(:, 1),'g', t,y(:, 2),'b'); legend('Solution', 'Derivative')   .
```

Легко пересвідчитись, що ці криві співпадають з тими, що дає аналітичний розв'язок. З наведеної вище формули загального розв'язку знаходимо значення констант $C_1 = 0{,}5 + y_0$ і $C_2 = (z_0 + y_0 + 4{,}5)/2$, за яких виконуються задані початкові умови. Криві аналітичного розв'язку будуємо командами:

```
>> t=0:.1:5; Yan=Berm4271a(t,1,1);
>> hold on; plot(t, Yan(1,:),'r', t, Yan(2,:),'r')
```

Тут *Berm4271a* – функція MATLAB, створена у файлі *Berm4271a.m*:

Лістінг 4.4. Зміст файлу *Berm4271a.m*:

```
function Y=Berm4271a(x,y0,z0)
%Analytical Solution of ODE #4271 [29]
%dzdx=(-17/2)*cos(2*x)-2z-5y,    dydx=z. Solution is
```

```
%
% y=exp(-x)*(C1*cos(2*x)+C2*sin(2*x)) -cos(2*x)/3 -2*sin(2*x)
% For Constants found
% C1=y0+.5; C2=(y0+z0+4.5)/2
% Solution and its Derivative:
Yan=exp(-x).*(C1*cos(2*x)+C2*sin(2*x)) -cos(2*x)/2 -2*sin(2*x);
Zan=-exp(-x).*(C1*cos(2*x)+2*C1*sin(2*x)+C2*sin(2*x)-
    2*C2*cos(2*x))+sin(2*x)-4*cos(2*x);
Y=[Yan; Zan];
```

В практичній роботі дуже рідко існує можливість співставити чисельний розрахунок з аналітичним розв'язком. В такому випадку слід знайти інше рівняння або систему, які були б подібними до заданих, але простішими і мали б аналітичний розв'язок. Розв'язати їх двічі, аналітично і чисельно, і таким шляхом пересвідчитись у правильній роботі алгоритму і програми.

4.5. ЗВИЧАЙНІ ДИФЕРЕНЦІАЛЬНІ РІВНЯННЯ: ЗАДАЧІ З КРАЙОВИМИ УМОВАМИ

Розглянемо інший клас задач для звичайних диференціальних рівнянь. Доволі часто лише частина додаткових умов для пошуку інтегральної кривої ЗДР дана на початку інтервалу $x = a$, а інша частина відома на кінці інтервалу $x = b$. Це означає, що задача, на відміну від початкових задач (4.7), (4.8) або (2.37), тепер формулюється таким чином [33,37]:

$$y^{(n)} = F(x, y', y'', ..., y^{(n-1)}),$$
$$y(a) = y_{0a}, \quad y'(a) = y_{1a}, \quad \dots, y^{(n_1)}(a) = y_{n_1 a}, \qquad (4.12)$$
$$y(b) = y_{0b}, \quad y'(b) = y_{1b}, \quad \dots, y^{(n_2)}(b) = y_{n_2 b}.$$

(Може також бути, що на кінцях інтервалу задані не "чисті похідні", а їх лінійні комбінації $R_{ai}[y] = \alpha_{0i} y(a) + \alpha_{1i} y'(a) + ... + \alpha_{n-1,i} y^{(n-1)}(a)$ і $R_{bi}[y] = \sum_{j=0}^{n-1} \beta_{ji} y^{(j)}(b)$). Така задача може мати єдиний розв'язок,

може мати декілька або не мати зовсім; в підручнику [33], с. 210 наведено цікавий приклад, коли ЗДР другого порядку з однією парою умов має безліч розв'язків, і жодного – з іншою парою умов. Як тут зорієнтуватися?

4.5.1. МЕТОД СТРІЛЬБИ РОЗВ'ЯЗУВАННЯ КРАЙОВИХ ЗАДАЧ

Краще уявити властивості граничної задачи (4.12) часто удається за допомогою її зведення до послідовності задач Коші [37]. Нехай з багатьох інтегральних кривих заданого ЗДР, що проходять через точку A площини, для яких тобто задоволена умова $y(a) = A$ на лівому краю інтервала $[a, b]$ (рис. 4.3), підходящою є лише та, що проходить через точку B з координатами (b, y_{0b}), де $y_{0b} = B$ (штрихова крива). Якщо ЗДР має другий порядок, то воно може бути інтерпретовано як рівняння руху матеріальної точки на площині, що починає свій рух з точки A внаслідок, наприклад, пострілу під певним кутом до горизонту. Кут пострілу (похідна $y'(a)$), одначе, невідомий, а замість відсутньої умови задана точка B, тобто $y(b)$, через яку має проходити траєкторія матеріальної точки. З такої інтерпретації витікає наступний *алгоритм стрільби*.

В точці $x = a$ до заданих n_1 умов додаєм таку кількість довільних додаткових умов $n_2 = n - n_1$, щоб було можливим розпочати розрахунок інтегральної кривої методом Рунге-Кутта; у випадку рівняння другого порядку (рис. 4.3) – до даної умови $y(a) = y_{0a}$ додаємо умову $y'(a) = \theta_1$, де значення θ_1 обрано довільно. Отже, "стріляємо" з кутом нахилу θ_1. Припустимо, розрахунок методом Рунге-Кутта дав криву *1*, що пройшла нижче точки B. Проводимо ще один розрахунок задачі Коші з новим кутом нахилу $y'(a) = \theta_2$ (другий "постріл"); якщо нова крива ще нижче за першу – напрямок зміни кута обрано невірно. "Стріляємо" ще раз, з новим кутом θ, і по поведінці інтегральної кривої робимо черговий висновок щодо обранного θ. Якщо чергова крива пройшла зверху від B (як крива *2* на рис. 4.3) – значить, правильне значення θ знаходиться між двох, перевірених останніми. Тепер

можна уточнювати це значення, аж поки інтегральна крива пройде через точку B з потрібною точністю. Можна сказати і так, що за

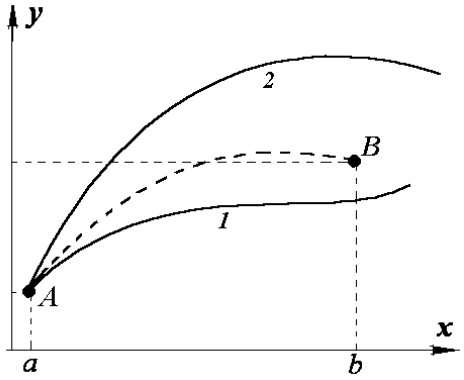


Рис. 4.3. Пояснення до методу стрільби: 1 – розв'язок ЗДР з недостачею, 2 – з перевищенням

допомогою послідовності задач Коші дану крайову задачу зведено до розв'язування відносно θ трансцендентних рівнянь вигляду $y(b;\theta) = B$. Потрібна така кількість рівнянь, скільки додано невизначених граничних умов в точці $x = a$, тобто n_2. Можна сподіватись, що математична модель (4.12) має єдиний розв'язок, якщо сума кількості умов на початку n_1 і на кінці інтервалу n_2 дорівнює порядку рівняння n.

Проте, таке правило необхідне, але не достатнє. Сказане є лише геометричною або механічною інтерпретацією проблеми розв'язування крайової задачі (4.12), яка зовсім не гарантує умов такого плавного переходу інтегральних кривих через точку B. "Гарантію" можна надати лише для певних класів рівнянь. Зручним "полігоном" для перевірки певних гіпотез часто слугує клас лінійних рівнянь. Отже, розглянемо задачу (4.12) для лінійних ЗДР, для прикладу – рівняння другого порядку:

$$L[y] \overset{Df}{=} y'' + p(x)y' + q(x)y = f(x), \qquad (4.13)$$

$$R_a[y] \overset{Df}{=} \alpha_0 y(a) + \alpha_1 y'(a) = A, \quad R_b[y] \overset{Df}{=} \beta_0 y(b) + \beta_1 y'(b) = B.$$

Лінійність операторів*) L і R означає $L[u+v]=L[u]+L[v]$ і $L[Cu]=CL[u]$. Відомо [34], що така властивість ЗДР дозволяє зображати кожну його інтегральну криву як лінійну комбінацію деяких базисних. Тому замість даної задачі розглянемо дві інші для допоміжних функцій $u(x)$ і $v(x)$:

$$\begin{aligned} L[u] &= f(x) \\ R_a[u] &= A, \\ u'(a) &= \theta, \end{aligned} \qquad (4.13,\text{А}) \qquad\qquad \begin{aligned} L[v] &= 0 \\ R_a[v] &= 0, \\ v'(a) &= \theta_1. \end{aligned} \qquad (4.13,\text{Б})$$

Обидва формулювання є задачами Коші, які можуть бути розв'язані вже відомими методами, наприклад – методом Рунге-Кутта. Неоднорідна задача (4.13,А) – "перший постріл" – дає траєкторію $u(x)$, яка, однак, не проходить через точку B, $R_b[u]\neq B$. Однорідна задача (4.13,Б) – "другий постріл" – дає траєкторію $v(x)$, яка не є тривіальним розв'язком (тотожнім нульом), бо $\theta_1\neq 0$; оберемо таке $\theta_1\neq 0$, аби $R_b[v]\neq 0$. Лінійність призводить до того, що функція $y=u+Cv$ задовольняє рівняння і першу граничну умову (4.13). А в точці $x=b$ відповідний оператор приймає значення $R_b[y]=R_b[u]+CR_b[v]$. На підставі двох проведених "пострілів" можна обрати таку сталу C, для якої $R_b[y]=B$, тобто задоволена і друга гранична умова (4.13):

*) Надпис *Df* над знаком прирівнювання означає "*Definition*", "*За визначенням*". Порівняйте дане визначення лінійності з таким визначенням в розділі 2.1

$$C = \frac{B - R_b[u]}{R_b[v]}.$$

Таким чином, розв’язуємо (чисельно або навіть аналітично) дві задачі Коші і із двох знайдених функцій конструюємо розв’язок граничної задачі (4.13). Розв'язок заданої задачі можливий, оскільки можна розв'язати обидві допоміжні початкові задачі.

4.5.2. МЕТОД СКІНЧЕННИХ РІЗНИЦЬ ДЛЯ КРАЙОВИХ ЗАДАЧ

Цей метод наближеного розв’язування крайових задач набув особливого поширення. Він полягає у заміні похідних, що входять до ЗДР, їх наближеними апроксімаціями. Покажемо реалізацію цього методу на прикладі лінійного ЗДР другого порядку (4.13).

Проміжок зміни незалежного аргументу розділимо на N частин *сіткою* з $N+1$-го *вузлів*

$$x_0 = a,\ x_1 = x_0 + \Delta x_1,\ x_2 = x_1 + \Delta x_2,\ \ldots, x_{N-1} = x_{N-2} + \Delta x_{N-1},$$
$$x_N = b,$$

в яких і будемо шукати наближені значення розв’язку

$$y_0 = y(x_0),\ y_1 = y(x_1),\ y_2 = y(x_2),\ \ldots,\ y_{N-1} = y(x_{N-1}),$$
$$y_N = y(x_N).$$

Для спрощення, сітку вважаємо рівномірною, $\Delta x_i = \Delta x = \frac{b-a}{N}$.

Апроксимацію першої похідної $y'(x)$ можемо прийняти як "*крок уперед*" або як "*крок назад*", відповідно

$$y'(x_i) \approx \frac{y_{i+1} - y_i}{\Delta x_{i+1}}, \qquad y'(x_i) \approx \frac{y_i - y_{i-1}}{\Delta x_i}. \tag{4.14}$$

Часто застосовують "*симетричну апроксимацію*" першої похідної $y'(x_i) \approx \frac{y_{i+1} - y_{i-1}}{2\Delta x_i}$. Так само маємо кілька варіантів для наближенного значення другої похідної у вузлах:

$$y''(x_i) \approx \frac{y'_{i+1} - y'_i}{\Delta x_{i+1}} \qquad \text{або} \qquad y''(x_i) \approx \frac{y'_i - y'_{i-1}}{\Delta x_i},$$

де перші похідні обираються за (4.14). Проте, найпоширенішим є застосування симетричної різниці

$$y''(x_i) \approx \frac{y'_{i+1} - y'_{i-1}}{2\Delta x} \approx \frac{\frac{y_{i+1} - y_i}{\Delta x} - \frac{y_i - y_{i-1}}{\Delta x}}{2\Delta x} = \frac{y_{i+1} - 2y_i + y_{i-1}}{2\Delta x^2} \tag{4.15}$$

Від того, яку з різницевих формул (4.14) і (4.15) використовувати, залежить остаточний вигляд *різницевого рівняння*, яким замінюємо точне рівняння (4.13). Обмежимося викладом однієї з можливих схем.

1. Неявна різницева схема. Похідні рівняння (4.13) замінюємо наступним різницевим рівнянням у кожному вузлі, за виключенням першого і останнього:

$$\frac{y_{i+1}-2y_i+y_{i-1}}{2\Delta x^2}+p_i\frac{y_{i+1}-y_{i-1i}}{2\Delta x}+q_iy_i=f_i,\qquad i=1,2,...,N-1.$$

Після приведення подібних отримуємо таку систему з $N-1$-го рівняння:

$$\begin{aligned}
A_1y_0+B_1y_1+C_1y_2&=f_1,\\
A_2y_1+B_2y_2+C_2y_3&=f_2,\\
A_3y_2+B_3y_3+C_3y_4&=f_3,\\
&\cdots\cdots\\
A_{N-2}y_{N-3}+B_{N-2}y_{N-2}+C_{N-2}y_{N-1}&=f_{N-2},\\
A_{N-1}y_{N-2}+B_{N-1}y_{N-1}+C_{N-1}y_N&=f_{N-1},
\end{aligned}\qquad(4.16)$$

де присутні $N+1$ невідоме $y_0, y_1, \ldots, y_{N-1}, y_N$ і де

$$A_i=\frac{1}{2\Delta x^2}-\frac{p_i}{2\Delta x},\qquad B_i=-(\frac{1}{\Delta x^2}-q_i),\qquad C_i=\frac{1}{2\Delta x^2}+\frac{p_i}{2\Delta x}.$$

Ще два рівняння отримуємо з граничних умов. Якщо на границях задано функції ($\alpha_1=0$ і $\beta_1=0$ в граничних умовах (4.13)), то такими рівняннями є $y_0=A$ і $y_N=B$; якщо задано похідні ($\alpha_0=0$ і $\beta_0=0$), то такими рівняннями, легко бачити, є $y_1-y_0=A\Delta x$ і $y_N-y_{N-1}=B\Delta x$; аналогічно отримуємо два додаткові

рівняння для крайових умов (4.13) загального вигляду.

Задачу приведено до системи лінійних алгебраїчних рівнянь. Однак не поспішаємо застосовувати один з методів розділу 2.1, оскільки основна кількість рівнянь доволі спеціального, *тридіагонального* вигляду. Найбільш ефективним тут є так званий *метод прогонки*. Цим методом спочатку обчислюють допоміжні коефіцієнти в порядку зростання індексів $i = 1,2,..., N$, а вже потім – шукані функції в порядку зменшення індексів $y_N, y_{N-1},..., y_1, y_0$. Не будемо тут викладати цей метод, відсилаємо до літератури [32,34,36].

2. Яка з різницевих схем краща? Зрозуміло, що застосування різних апроксимацій для першої і другої похідних дають різні різницеві рівняння, звідки і випливає поставлене питання. Відповісти на нього однією фразою неможливо, кожна схема може працювати добре для одних задач і не працювати для інших. Для пояснення треба нагадати про найважливіші властивості цього інструменту обчислень.

Зрозуміло, що першою вимогою до різницевої схеми має бути прагнення її розв'язку до точних значень інтегральної кривої ЗДР у вузлах, якщо Δx спрямовуємо до нуля. Якщо похибка методу прагне до нуля як певний степінь кроку інтегрування Δx^k, то цей степінь k називають *порядком апроксимації*. Наведена схема (4.17) має другий порядок

апроксимації. Різницева схема тим краще, чим порядок апроксимації вище.

Проте, в справу втручаються й інші фактори. Усі обчислення при розв'язанні СЛАР (4.17) проводяться з певною похибкою. Чим менше Δx, тим більше рівнянь в системі (4.17), тим більше можливостей накопичити похибку за рахунок округлення проміжних даних. Якість різницевої схеми перевіряють тим, що штучно вносять малі збурення величин і дивляться на їх розвиток крок за кроком. Добрими є лише *стійкі* різницеві схеми, в яких збурення суттєво не зростають.

Про методи дослідження різницевих схем, про конкретні схеми, рекомендовані для тих чи інших рівнянь і крайових задач для них можна прочитати в підручниках [32,34-36]. Там же описані і деякі інші методи чисельного розв'язування граничних задач для ЗДР.

4.5.3. РОЗВ'ЯЗУВАННЯ КРАЙОВИХ ЗАДАЧ В MATLAB

Можливі особливості крайових задач для звичайних диференціальних рівнянь, потреби гарантування точності і стійкості розрахунку враховано розробниками MATLAB в команді *bvp4c* (назва походить від *Boundary Value Problem*), призначеної для їх "автоматичного" розв'язування. Зауважимо, що попередня версія середовища MATLAB v.5 ще не могла розв'язувати такі задачі "автоматично", кожен користувач мав розробляти

власну програму, виходячи з того чи іншого методу. І на даному етапі MATLAB v.6 має лише одну версію *4c* такої команди*), на відміну від семи версій для задачі Коші (див. розділ 4.4). Це пояснюється більшою складністю алгоритмів для даного типу задач. В майбутніх версіях мають з'явитися нові команди для крайових задач для ЗДР.

Розглянемо, як працює названа команда *bvp4c*. Для правильного звертання до неї треба слідувати певним домовленостям, що закладені програмістами. Виходимо з того, що рівняння записано у вигляді нормальної системи з n рівнянь (2.36) або (2.38). Для правої частини такої системи спочатку треба написати програму-функцію з умовним іменем *ODE_right*, яка вихідний вектор-стовпчик пов'язує із змінними x, y_1, …, y_n системи ЗДР. Приклади таких програм вже наводилися в лістінгах 4.1 і 4.3.

Далі аналогічним чином слід створити програму-функцію для моделювання граничних умов, яку назвемо, наприклад, *bc_ODE*. Її вхідними аргументами мають бути два вектори (назвемо їх *Ya* і *Yb*), а результатом – вектор-стовпчик *res*. Перший вектор *Ya* відноситься до початку інтервалу $x = a$ і другий *Yb* – до кінця інтервалу $x = b$. Припустимо, на початку інтервалу ставлять n_1 умов і k-та з них виглядає як $(\alpha_1 y_1 + \alpha_2 y_2 + ... + \alpha_n y_n)|_{x=a} = A$. Тоді цю

*) Допитлий читач має знати, що названа команда MatLab'у реалізує *метод колокацій* [33]; проте йому притаманні всі риси чисельного методу скінченних різниць, описаного вище.

умову слід передавати в *k*-й елемент вектора *res* за схемою

$$res(k) = \alpha_1 Ya(1) + \alpha_2 Ya(2) + \ldots \alpha_n Ya(n) - A,$$

тобто роль $y_1(a)$ відіграє $Ya(1)$, роль $y_2(a) - Ya(2)$ і т.д. Аналогічно, якщо *p*-та умова ставиться на кінці інтервалу у вигляді $(\beta_1 y_1 + \beta_2 y_2 + \ldots + \beta_n y_n)|_{x=b} = B$, то у програмі-функції *bc_ODE* вона схематизується у вигляді

$$res(p) = \beta_1 Yb(1) + \beta_2 Yb(2) + \ldots \beta_n Yb(n) - B.$$

Програму для граничних умов можна записати як підпрограму у файлі правих частин *ODE_right.m*, як це показано на прикладі нижче.

Тепер слід приготувати початкове припущення щодо майбутнього розв'язку, яке назвемо *Sol_0* і яке складається з набору вузлів і початкових значень всіх шуканих змінних в цих вузлах. Не звертаємо особливої уваги на те, що *Sol_0* є новим, ще не поясненим типом даних, *структурою*, бо існує допоміжна програма *bvpinit*, яка утворить *Sol_0* автоматично. Для цього перш за все треба задати якусь початкову сітку вузлів *X_init* як, наприклад

X_init=[*0 1 2 3 4 5*], або як *X_init*= *linspace*(*0, 5*).

(Перший спосіб означає: "За початкові вузли взяти точки *x=1*, *2*, *3*, *4* разом з початком *x=0* і кінцем *x=5* інтервалу [*0* , *5*]"; другий спосіб доручає утворити сітку програмі *linspace*, яка вказаний інтервал ділить

на *100* частин). Далі треба повідомити про очікувані значення всіх n невідомих в цих вузлах за допомогою масиву *Y_init*. І тепер команда

$$>> Sol\_0 = bvpinit(X\_init, Y\_init)$$

утворить структуру початкового наближення. Масив *Y_init* можна задавати по-різному. Це може бути вектор з n сталими (за кількістю шуканих невідомих) – тоді *bvpinit* вважатиме кожну *i*-ту сталу за стале на всьому проміжку початкове значення *i*-ї невідомої $y_i(x)$. Це може бути також n функцій – тоді кожне $y_i(x)$ буде обчислюватись за ними. Від правильного вибору *Y_init*, його близкості до остаточного розв'язку залежитиме збіжність обчислювального процесу. Проте, часто можна починати з задання *Y_init* сталими!

Нарешті, все готове для остаточної команди побудови розв'язку запрограмованої краєвої задачі:

$$>> Solution = bvp4c(ODE\_right, bc\_ODE, Sol\_0);$$

В результаті виконання даної команди буде утворено структуру *Solution*, яка містить значення всіх n шуканих функцій у вузлах інтегрування. Вузли інтегрування, уточнені заради досягнення потрібної точності, містяться у векторі *Solution.x*, а функції – у колонках масиву *Solution.y*. Тобто звертатись до y_1 треба через *Solution.y*(*1*, **:**), до y_2 через *Solution.y*(*2*, **:**) і т.д. Наприклад, побудувати графік *i*-ї функції можна командою

>> plot(*Solution.x*, *Solution.y*(*i*,:)) .

Покажемо реалізацію такого MATLAB-алгоритму розв'язування граничних задач для ЗДР на прикладі.

Приклад 4.8. Розв'язати крайову задачу на інтервалі $x \in [0,5]$

$$y'' + 2y' + 5y = -\frac{17}{2}\cos 2x, \quad y(0) = 1, \quad y(5) = 1.$$

Нагадаємо (див. приклад 4.7), що функція правих частин, після перетворення рівняння до нормальної системи рівнянь (4.11), вже готова і зберігається у файлі "*Berm4271.m*". Функцію граничних умов збережемо у файлі "*bc_Berm4271.m*":

Лістінг 4.5. Зміст файлу *bc_Berm4271.m*:

```
function res=bc_Berm4271(Ya ,Yb)
%Function to express boundary conditions of the Example 4.8
res=[Ya(1)-1 ; Yb(1)-1];
```

Тепер розв'язуємо задану граничну задачу такою послідовністю команд:

```
>> % Утворення сітки вузлів і початкового наближення
>> Xmesh=linspace(0 , 5); Y_init=[1 1];
```

(зрозуміло, що якщо вже вважати початкове наближення сталим, то таким, що співпадає із заданими $y(0)$ і $y(5)$).

```
>> % Утворення структури початкового наближення
>> Sol_0=bvpinit( Xmesh , Y_init);
>> % Шукаємо структуру з даними розв'язку
>> Solution=bvp4c('Berm4271' , 'bc_Berm4271', Sol_0);
>> % Будуємо графіки розв'язку
>> plot(Solution.x, Solution.y(1 , :),'b', Solution.x,Solution.y(2 , :)
,'g')
>> legend( 'Function y(x)' , 'Derivative y1(x) ')
```

В результаті маємо графіки шуканої функції $y(x)$ та її похідної $y'(x)$, зображені на рис. 4.4. Зверніть увагу, що інтегральні криві поводять себе зовсім інакше, ніж у прикладі 4.7, рис. 4.2! Легко

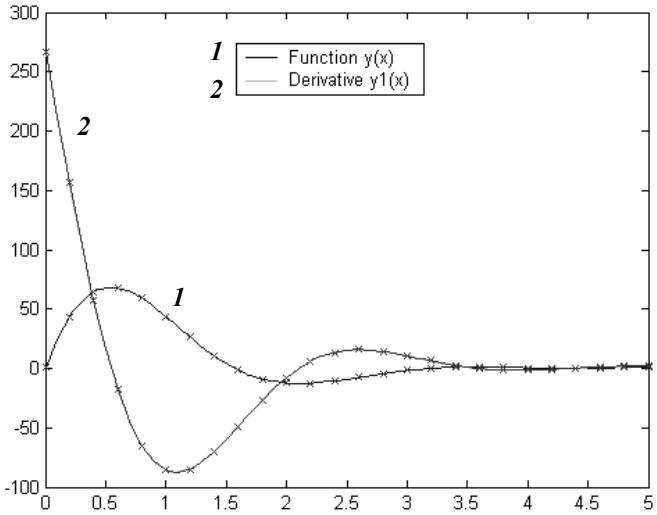


Рис. 4.4. Результати чисельного розв'язування граничної задачі прикладу 4.8 у порівнянні з аналітичним розв'язком (хрестики)

пересвідчитись, що крива *1* задовольняє граничну умову.

Природньо, що даний навчальний приклад обрано таким чином, аби була можливість перевірки точності чисельного розв'язку порівнянням з аналітичним. Так, у попередньому прикладі 4.7 наведена формула аналітичного розв'язку. Сталі C_1 і C_2, що входять до формули, мають бути такими, що інтегральна крива приймає значення y_a і y_b на кінцях інтервалу:

$$y(0) = C_1 - \frac{1}{2} = y_a,$$

$$y(b) = e^{-b}(C_1 \cos 2b + C_2 \sin 2b) - \frac{1}{2}\cos 2b - 2\sin 2b = y_b.$$

З цієї СЛАР знаходимо сталі. Результат у вигляді програми-функції аналітичного розв'язку даної крайової задачі програмуємо у файлі

Лістінг 4.4′. Зміст файлу *Berm4271a1.m*:

```
function Y=Berm4271a1(x,ya,yb)
% Analytical Solution of BVP for the ODE #4271 [29]
%dzdx=(-17/2)*cos(2*x)-2z-5y,    dydx=z. Solution is
% y=exp(-x)*(C1*cos(2*x)+C2*sin(2*x)) -cos(2*x)/3 -2*sin(2*x)
%
% For Constants found
b=5;   C1=ya+.5;
C2=(exp(b)*(yb+cos(2*b)/2+2*sin(2*b))-C1* cos(2*b))/sin(2*b);
% Solution and its Derivative:
Yan=exp(-x).*(C1*cos(2*x)+C2*sin(2*x)) -cos(2*x)/2 -2*sin(2*x);
Zan=-exp(-x).*(C1*cos(2*x)+2*C1*sin(2*x)+C2*sin(2*x)-
    2*C2*cos(2*x))+sin(2*x)-4*cos(2*x);
Y=[Yan; Zan];
```

Тепер на попередній рисунок накладаємо графіки аналітичних кривих при *ya=1* і *yb=1*:

```
>> t=0:.2:5;                    % Вектор аргументів кроком .2
>> Yan=Berm4271a1(t,1,1); % Вектор значень функції і похідної
>> %                            Будуємо графіки у вигляді точок x
>> hold on; plot(t,Yan(1,:),'rx', t,Yan(2,:),'rx')
```

Значки дуже точно попадають на криві, отримані чисельно!

В обчислювальній практиці можуть виникнути певні питання щодо передачі параметрів ЗДР, більш точного задання початкових наближень тощо. Відповіді на більшість з них можуть бути знайдені у *Help* і у подальшій літературі [1-18].

4.6. ДИСКРЕТНЕ ПЕРЕТВОРЕННЯ ФУР'Є

Дискретне перетворення Фур'є – це комп'ютерний аналог аналітичного перетвореня Фур'є, про яке йшлося в розділі 1.6. З

іншого боку – це вдалий винахід математики і радіофізики останніх п‘ятидесяти років, який у формі алгоритму **швидкого перетворення Фур'є** (***Fast Fourier Transform***, ***FFT***) знайшов надзвичайно широке використання в сучасних інформаційних технологіях. В цьому легко пересвідчитись, зазирнувши в довідково-пошукову систему MATLAB з ключовим словом *Fourier*. Проте, тут обмежимось лише коротким визначенням *FFT*.

Алгоритм ***fft***, швидке перетворення Фур'є – це "чорна скриня", яка на вході отримує вектор h з n дійсних чисел і на виході видає вектор H з такою ж кількістю n комплексних чисел. При цьому має місце така чудова обставина: як вектор, складений з дійсних частин останнього *real*(H), так і вектор з уявних частин *imag*(H) відображають частотний склад вхідного вектора. Наприклад, якщо h складається з середньомісячних температур січня за багато років, то H несе інформацію про періодичність цього процесу (навіть якщо виміри температури, природньо, мають якусь випадкову похибку)! З метою відобразити, що h належить до реальної функції часу, а H відображає частотний склад функції, зв‘язок між ними зображають як $h(t) \Leftrightarrow H(\omega)$. Якщо перша функція подається на вхід "чорної скрині" дискретно з рівномірним кроком Δt, то вихідну функцію слід інтерпретувати дискретною з кроком $\Delta\omega = \dfrac{2\pi}{N \cdot \Delta t}$. Таке співвідношення показане схематично на рис. 4.5.

Застосування аналітичного перетворення Фур‘є і швидкого перетворення Фур‘є вивчають в спеціальних курсах математики, радіофізики і інформатики. Тут ми обмежимось лише ілюстрацією застосування MATLAB.

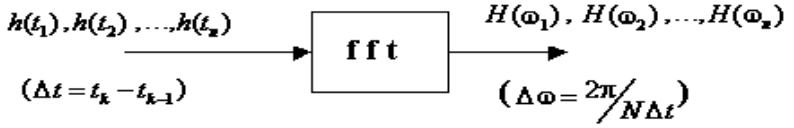


Рис. 4.5. Схематичне зображення програми *Fast Fourier Transform*

Уявіть собі, що ви отримали від вашого приятеля набір чисел на дискеті, який являє собою дискретизовані із кроком Δt значення функції вигляду

$$h(t) = a_1 \sin(\omega_1 t) + a_2 \cos(\omega_2 t) + a_3 \sin(\omega_3 t) + \ldots, \qquad (4.17)$$

але про частоти гармонік ω_1 , ω_2 , ω_3 і т.д. він сказати вам небажає. Та й що ж, узнаємо їх самі за допомогою *fft*!

Ваш приятель № 1 (або ви самі для приятеля № 2) утворював вхідні дані командами:

>> Om1=20; *Om2=40*; *Om3=50*; *dt=1/Om3*; *N=1000*; *t=0:dt:N*dt*;

*>> h=2*sin*(*Om1*t*) *-3*cos*(*Om2*t*) *+sin*(*Om3*t*);

тут *Om1*, *Om2* і *Om3* ті самі "таємні" частоти ω_1 , ω_2 і ω_3 , а коефіцієнти при гармоніках значення не мають; обрано певний крок часу *dt*, взято *N* таких кроків часу *t* і утворено вектор значень функції *h*(*t*). Вектори *h* і *t* збережено на дискеті і передано приятелеві № 2 (про те, як зберігати і надалі імпортувати дані йшлося в розділі 3.3).

(Приятель № 1 може зробити ще хитріше, а саме – додати до кожного точного "виміру" (4.17) випадкову похибку. Така похибка, розподілена за нормальним законом із середнім значенням *M* і дисперсією *D*, генерується командою MATLAB *randn*. Тобто ваш хитрий приятель має утворити вектор похибок *Error* і додати його до вхідних даних:

>> M=0; *d=1*;

*>> Error=M+d*randn*(*1*,*N+1*);

*>> h=2*sin*(*Om1*t*) *-3*cos*(*Om2*t*) *+sin*(*Om3*t*)*+Error*

(дисперсію тут взято $D= d^2=1$, але варто поекспериментувати з цією величиною).

Приятелю № 2 треба узнати приховані від нього частоти. Перш за все слід візуалізувати ці дані:

>> tt=t(1:100); hh=h(1:100); plot(tt,hh,'x',tt,hh)

(допоміжні вектори *tt* і *hh* створено, аби побачити лише частину кривої, бо інакше "за деревами не побачити лісу"). Графік показано на рис. 4.6,А. Чи можна тут піймати якусь закономірність?

Однак виконуємо швидке перетворення Фур‘є, за яке "відповідає" команда *fft*. Знаходимо реальну і уявну частини результату *H*:

>> H=fft(h); ReH=real(H); ImH=imag(H);

Тепер важливо правильно співставити отримані дані з аргументом ω за вказаною вище формулою:

*>> dt=t(2) - t(1); Omega=2*pi*(0:N)/(N*dt);*

Будуємо тепер результуючі графіки

>> figure; plot(Omega , ImH); figure; plot(Omega , ReH) .

Останній з них наведено на рис. 4.6. Розглянемо рисунок.

Ліві піки на ньому припадають якраз на частоти ω=20, 40 і 50 (аби в цьому упевнитись, відокреміть зону навколо цих піків після натискання кнопки  вікна *Figure*). Ось ви і відгадали "таємницю" приятеля, які саме частоти він загадав! Бачимо, що програма *fft* працює як "детектор" частоти коливань.

Однак, що ж то за піки праворуч? То – похибка *fft*. Кожний чисельний метод має неодмінні похибки. У даного метода вони виникають у вигліді фальшивих піків. Як відрізнити похибку від змістовної інформації? Пропонується такий метод.

З наданої інформації $\{h_i\}$, $\{t_i\}$ зробимо вибірку через один, тобто $\{h_1, h_3, h_5, \ldots\}$, $\{t_1, t_3, t_5, \ldots\}$, і ці "нові" дані піддамо швидкому перетворенню Фур’є:

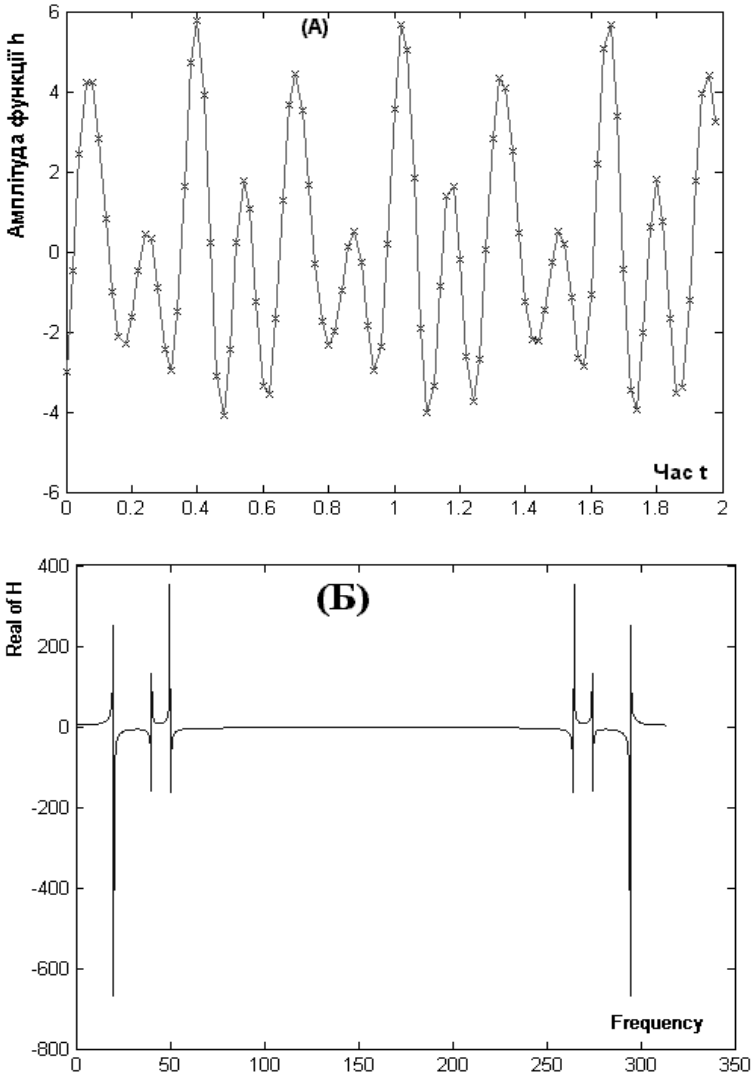


Рис. 4.6. (А) вхідна інформація про функцію (4.17);
(Б) приклад роботи "детектора частоти" *fft*

```
>> t1=t(1:2:end); h1=h(1:2:end); dt1=t1(2) -t1(1);
>> H1=fft(h1); ReH1=real(H1); ImH1=imag(H1);    .
```

Тепер утворюємо відповідний набір частот і будуємо графіки

```
>> N1=length(t1) -1; Omega1=2*pi*(0:N1)/(N1*dt1);
>> figure; plot(Omega1,ImH1); title('Аналіз підвибірки');
>> figure; plot(Omega1, ReH1); title('Аналіз підвибірки');    .
```

На нових графіках бачимо, що ліві піки залишилися на попередньому місці, а праві змінили своє положення. В такий спосіб можемо відрізнити правильні і помилкові піки.

Слід для повноти зазначити, що "детектор частоти" *fft* працює правильно лише за умов достатньо "представницької" вибірки вхідної інформації $\{h_i\}$, $\{t_i\}$. А саме, крок вибірки Δt має бути достатньо малим, $\Delta t < \pi / \omega_{\max}$, де $\omega_{\max}$ – найбільша з частот коливань (4.17), тобто у даному випадку $\omega_{\max} = \omega_3 = 50$ (теорема Котельникова-Шеннона). Студентам, які докладно вивчають перетворення Фур‘є, радимо провести чисельні "експерименти" з програмою *fft*, "граючись" розміром вибірки *N*, кроком *dt* в залежності від частот, що складають вхідну інформацію. Для такого експерименту пропонуємо скріпт *MyFFT*, що працює в діалозі з користувачем.

В файлі MyFFT збережено такий текст:

Лістінг 4.6. Зміст файлу *MyFFT.m*

```
% Program for testing sampling quality of h(t)=sin(2*pi*f0*t)
% depending on SamplingStep 'dt' and Number 'N'
% Copyrite of Dr. Ye.Gayev
disp(' );
disp('This Program makes the Discrete Fourier Transform of a Sinusoid ');
disp('and allows investigation of the effect of some parameters ');
disp(' h=Sin(2*Pi*f0*t)');
disp('Some Data are required to proceed'); disp(' ');
f0=input('Please input Frequency value of the Sinusuid, f0 \n'); T0=1/f0;
a=input('Input a real number a to divide Sinusoid"s Period To to M
\n'); dt=T0/a;
```

```
disp('Sampling Frequency is  '); fS=1/dt,
N=input('Input an integer N, number of sampling \n'); t=0:dt:N*dt;
N1=round(.5/dt); t1=0:dt:2*N1*dt; t2=0:dt/1000:2*N1*dt;
h=sin(2*pi*f0*t); h1=sin(2*pi*f0*t1); h2=sin(2*pi*f0*t2);
H=fft(h); freq=(0:N)/(N*dt); ReH=real(H); ImH=imag(H);
disp(' '); disp('Press <Enter> and enjoy the result!'); disp(' ');
pause % The Program waits until <Enter> be touched
figure(12); subplot(311);plot(t1,h1,'x-',t2,h2,'r--');
title('Original Signal (red) and Sampled Data (blue). Please compare!');
subplot(312); plot(freq,ReH); title('Real part of H'); subplot(313);
plot(freq,ImH); title('Imaginative part of H');
disp('What do you think about it?'); disp(' ');
```

Програма спочатку інформує користувача про своє призначення: "*Ця програма виконує Дискретне перетворення Фур'є синусоїди та дозволяє дослідити вплив деяких параметрів fft. Для роботи потрібні деякі дані*". Далі програма запрошує "*Введіть частоту цієї синусоїди, f0*". Після ввода частоти просить ввести "*Число a, долю періоду синусоїди* T_o" та "*Ціле N, кількість виборок з синусоїдальної функції*". Програма повідомляє також про "*Введену частоту*" та будує *Figure* за введеними значеннями *a* і *N*. Дослідження полягає у побудові кількох таких рисунків з різними параметрами *a* та *N*, у їх порівнянні і висновках про вплив названих параметрів на результат дискретного перетворення Фур'є.

Математичне середовище MATLAB дозволяє створювати і більш досконалі, більш приємні у користуванні діалоги. Приклад такого дано в наступному підрозділі.

4.7. ДІАЛОГОВЕ ВІКНО ДЛЯ ПОШУКУ ЕМПІРИЧНИХ РІВНЯНЬ

Обчислення коефіцієнтів емпіричних рівнянь, яке описано в розділі 2.3, корисно для засвоєння студентами фундаментальної значущості методу найменших квадратів. Однак для повсякденної роботи MATLAB пропонує готові засоби для швидкого знаходження емпіричних рівнянь з певного набору.

1. Треба, буває, *імпортувати* в MATLAB ті дані, що отримані "зовні", в якомусь, скажімо, експерименті. Сучасні вимірювальні прилади дають змогу зберегти дані експерименту в пам′яті комп′ютера в певному форматі, у файлі під певним іменем. Нехай ім′ям файла*) буде "*exper2.dat*" і цей файл в найпростішому, так званому *plain*-форматі, вміщує дані на зразок поданих у таб. 4.1 даних вимірів:

Таблиця 4.1. Зразок даних умовного експерименту

```
#Sampling time = 30.0 secs
#Position    U (av)      U/UR        u/UR        u/U
#--------    ------      ----        ----        ---
0               0       0            0           0
2.00000     3.49442     0.43861     0.10246     0.23360
6.00000     3.74956     0.47064     0.10301     0.21887
10.00000    3.96785     0.49804     0.10216    0.20513
14.00000    4.04939     0.50827     0.10477    0.20614
 ...          ...         ...         ...         ...
 ...          ...         ...         ...         ...
318.0000     8.07720     1.01383     0.00826    0.00815
322.0000     8.07030     1.01297     0.00774    0.00765
```

Три перші рядки (*headers*) вміщують текстову інформацію, а нижче пішли власне цифрові дані. Бачимо, що стовпчики з даними відокремлюються один від одного пропусками (*space*), але можливі і інші розділювачі (*coma*, *semicolon* **;** тощо).

За допомогою іконки  відкриваємо директорію пам′яті комп′ютера з потрібним файлом (дискету *A*, наприклад); у вікні "*Тип файлів*" треба замовити "*All_files*" і обрати потрібний файл "*exper2.dat*". Включається *Import Wizard* (*Помічник імпорту*), показаний на рис. 4.7. Вказуємо йому тип розділювача стовпчиків (*space*) і кількість рядків у заголовку (*Text header lines*), яка дорівнює ***3***. Вікна перегляду ліворуч і праворуч дозволяють упевнитись, чи вірно намагається MATLAB прочитати ваші дані.

*) Нагадаємо, що три символи імені після крапки, "*dat*" в даному випадку, вказують на формат файла. MatLab розуміє широкий спектр форматів, які використовуються різноманітними програмами – *Excel*, *SuperCalc* тощо

Натискуєм [illegible] (рис. 4.8) інформує, [illegible] директорії MATLAB [illegible]

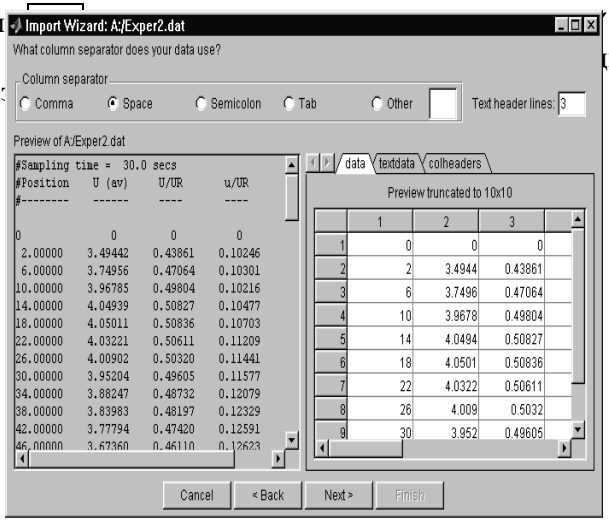


Рис. 4.7. Вигляд Wizard для імпортування з файла даних

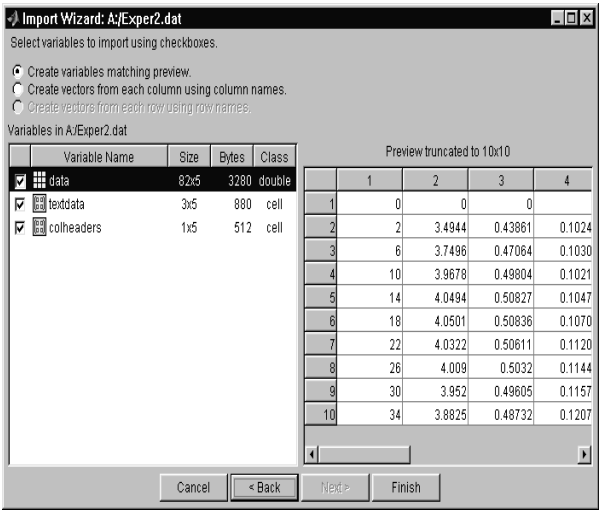


Рис. 4.8. Наступне і останнє вікно *Wizard* для імпорта даних. Лишається натиснути клавішу *Finish*

data, так само як і інформацію заголовку під іншими іменами. Треба натиснути *Finish*. Тепер MATLAB може працювати з імпортованими даними. Якщо подивитись тепер у його *робочий простір* (*Workspace*) за допомогою пунктів меню інтерфейса

View ‣ ***Workspace***,

то побачимо там вказані змінні *data* і *textdata*. Доцільно надати остаточне змістовне ім′я масиву даних, що введено:

>> Exp2=data; .

Тепер всі дані експерименту зберігаються в двовимірному масиві *Exp2*. Легко побудувати, наприклад, графік отриманих залежностей:

>>plot(*Exp2*(*1*:*end*,*3*),*Exp2*(*1*:*end*,*1*),'*r*-
',*Exp2*(*1*:*end*,*5*),*Exp2*(*1*:*end*,*1*),'*g*-')

(колонка 1 як спільна ордината двох графіків, колонки 3 і 5 по вісі абсцис). Можна тепер проводити пошук емпіричної формули для імпортованих даних.

2. Інформацію про пошук емпіричної формули (англійською – *curve fitting*) можна знайти у довідковій системі Help по шляху

Contents ‣ *MATLAB* ‣ *UsingMATLAB* ‣ *Mathematics* ‣
‣ *DataAnalysis and Statistics*

Тут наведено ряд розділів (*Column-Oriented Data Sets*, *Data Preprocessing*, *Regression and curve Fitting* і особливо *CaseStudy*:*Curve Fitting*), які детально тлумачать можливості у підборі емпіричного рівняння, що надаються MATLAB. Покажемо найголовніші.

Пошук емпіричної формули (рівняння) з використанням "внутрішніх" можливостей MATLAB проводиться одночасно з "візуалізацією" табличних даних, що зберігаються в змінній *Exp2*, тобто одночасно з побудовою графіка емпіричної функції. Нехай файл *Exp2* зберігає дані табл.. 4.1. Припускаємо, що аргументи x_i зберігаються у першому стовпчику файла *exp2*, тобто *x=exp2*(*1:end,1*), а результати вимірів – у другому стовпчику

y=exp2(*1:end,2*). Графік з′являється, як звичайно, після виконання команди

>> *plot*(*exp2*(*1*:*end* ,*1*), *exp2*(*1*:*end* , *2*), '*r* o')

(опція '*r* o' означає, що експериментальні точки будуть позначатись кружочками червоного кольору, *r*ed). Результати разом з вікном Figure бачимо на рис. 4.9. Натискуємо клавішу *Tools* і з меню, яке з′являється, обираємо *BasicFitting*, як показано на рис. 4.9. З′являється нове вікно *BasicFitting*, де слід відмітити бокси *linear* (лінійна формула) та (або) *quagratic* (квадратична формула), а також *ShowEquations* (показати рівняння). У графічному вікні відразу бачимо пряму, що наближує експериментальні дані, і її рівняння, так само як параболу з її рівнянням. Задачі пошуку того чи іншого наближуючого рівняння розв′язано. Кнопка *More* дозволяє отримати ще додаткову інформацію, наприклад – дізнатися про величину нев′язки (*residual*) при використанні першого чи другого з рівнянь і таким чином обрати найкраще.

Для "експериментальних" даних, які подано у табл. 2.2 прикладу 2.6 отримуємо таке лінійне наближення:

$$y = 20x - 41,$$

або таке параболічне наближення:

$$y = 1{,}4x^2 + 5{,}6x - 9{,}8.$$

Перше рівняння збігається з розрахунком прикладу 2.6. Криві, що отримано, "найкращим чином" проходять між усіх "експериментальних" точок. Як саме проходять – можна бачити з рис. 4.10.

Пошук емпіричних формул часто зустрічається в інженерній і дослідницькій роботі. Інші корисні формули, параметри яких можна знаходити даним методом, наведено в розділі 2.3. Завдання для закріплення цього важливого матеріалу дані в лабораторній роботі посібника [3].

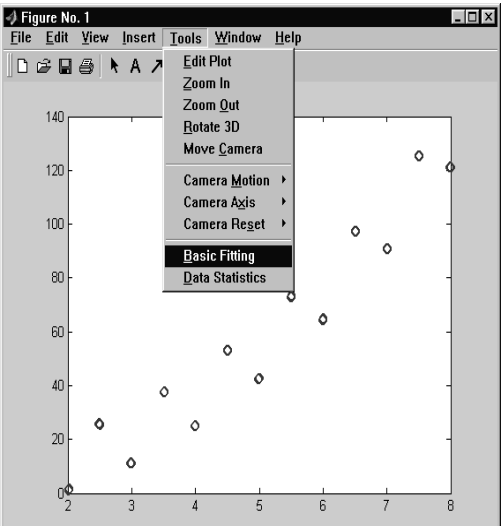


Рис. 4.9. Вікно Figure з експериментальними точками і відкритим підменю меню Tools, де обрано Basic Fitting

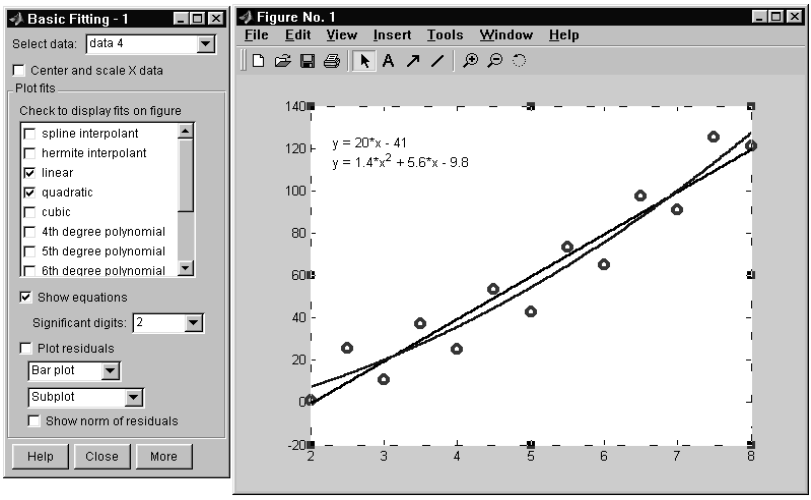


Рис. 4.10. Поява додаткового вікна Basic Fitting, де обираємо тип наближуючого рівняння – лінійне і квадратичне

4.8. КОНТРОЛЬНІ ПИТАННЯ ДО РОЗДІЛУ 4

1. Навіщо розроблені так звані "внутрішні програми (команди)" MATLAB? Чому ми використовуємо їх у даному розділі і не використовували в попередніх?
2. Які команди ("внутрішні програми") використовуються для розв'язування задач лінійної алгебри? Яким вимогам має відповідати вхідний аргумент "команд" *det* і *inv*? Що станеться, якщо аргумент не задовольняє такі вимоги? Якщо вхідна матриця має різні вимірності? Якщо її визначник дорівнює нулю? Якщо він близький до нуля?
3. Якою операцією MATLAB можна поміняти місцями рядочки і стовпці матриці? Як вона називається?
4. Назвіть кілька способів розв’язаня в MATLAB системи лінійних алгебраїчних рівнянь (СЛАР).
5. Навіщо розв’язувати алгебраїчні рівняння? Скільки коренів має таке рівняння? Як можна в MATLAB знайти корені многочлена?
6. Чому математики розділяють рівняння на лінійні і нелінійні? Що таке *лінійність*? Команда *fzero* працює з чисельними чи з символьними об’єктами? Чому в цій функції задають другий параметр? Що має видати ця команда, коли розв'язку рівняння не існує?
7. Які ви знаєте різновиди задач для звичайних диференціальних рівнянь? Скільки додаткових умов треба брати до уваги, розглядаючи диференціальне рівняння?
8. Які задачи розв'язують команди MATLAB *ode45*, *ode 23*, *bvp4c*? Скільки вхідних параметрів вони потребують, що ці параметри означають? Які обчислювальні методи насправді "приховано" в названих командах?
9. Як ви розумієте суть і призначення перетворення Фур'є? Про що свідчить перехід з "фізичного простору" в "частотний простір"? Що має бути "на вході" і що "на виході" команди *fft*?
10. Які "готові засоби" пропонує MATLAB для підбору параметрів, скажімо, квадратичної залежності отриманих експериментальних даних? Скільки параметрів треба знайти для такої залежності?

П І С Л Я М О В А

Отже, ви познайомилися з основами універсального математичного середовища MATLAB і з його застосуванням до деяких типових задач математичного моделювання, що вивчають у курсах вищої математики, прикладної математики і математичного моделювання Національного авіаційного університету. Лабораторні роботи [3] мають закріпити отримані знання. Сподіваємося, тепер ви погодитесь з тезами, висловленими у передмові до посібника, а саме:

- MATLAB – пакет зовсім не складний, і не так вже багато часу потрібно для оволодіння основами його мови;
- "нагородою" за витрачений час і зусилля є те, що (зізнаємось!) нудні і важкі задачі стандартних математичних курсів ви можете тепер розв'язувати, не залишаючи свого улюбленого комп'ютера, швидко і легко;
- основою MATLAB можна вважати його графічні команди. Вони надають вам можливості візуалізувати отримані розв'язки і "експериментувати" з ними. Отримали, наприклад, інтегральну криву задачі Коші для заданого значення граничної умови $y(0) = y_0$. Спробуйте тепер "погратися" з величиною y_0 – доволі неочикувані криві з’являться на екрані!
- Такі "ігри" за допомогою MATLAB є насправді зовсім не розвагою, а вашим власним

математичним дослідженням. Може, поталанить і вам відкрити, як Адамсу і Лєвєр'є, вашу планету Нептун, і як Діраку – частку позітрон*)!

Насамкінець хочемо вам сказати, що в цьому посібнику вистачило місця розказати лише приблизно про половину можливостей MATLAB. Студент, який хоче піти далі в оволодінні MATLAB, має познайомитись ще з вражаючою тривимірною графікою і поглибити свої навики в програмуванні – познайомитись із поняттями структури і об'єктно-орієнтованим програмуванням, програмуванням графічного інтерфейсу користувача (GUI). Тут вам допоможуть книги [2,4,6,7,11,15,16].

Так само математичні можливості MATLAB не вичерпуються розглянутими тут моделями і задачами. Адже MATLAB може бути вашим помічником і в моделях теорії ймовірностей і математичної статистики, в обробці звуку і зображень, при аналізі експерименту і в багатьох інших питаннях вашої майбутньої спеціальності. І ще, мабуть найважливіше, що обов'язково треба вивчити – це розв'язування рівнянь з частинними похідними. Про це і про багато іншого дізнаєтесь у книгах [1,4,5,8,9,14,17,18].

*) Дійсно, обидва відкриття були зроблені (відповідно, у 1846 і 1931 рр), так би мовити, чисельним експериментом

С П И С О К Л І Т Е Р А Т У Р И

ОСНОВНИЙ

1. **Андриевский Б.Р., Фрадлов А.Л.** Элементы математического моделирования в программных средах MATLAB и SciLab. – СПб: Наука, 2001. – 286 с.
2. **Ануфриев И.** Самоучитель MATLAB 5.3/6.x. – СПб.: БХВ-Петербург, 2002. – 736 с.
3. **Гаєв Є.О., Нестеренко Б.М.** Збірник лабораторних робіт. К.: НАУ, 2004 р. – 39 с.
4. **Говорухин В.**, **Цибулин В.** Компьютер в математическом исследовании. Учебный курс: Maple, MATLAB, LaTex. – СПб: Питер, 2001. – 624 с.
5. **Гультяев А.** MATLAB 5.2. Иммитационное моделирование в среде Windows. Визуализация, программирование, анализ данных. - СПб: "Корона", 1999. – 288 с.
6. **Дьяконов В.** MATLAB. Учебный курс. – "Питер", 2001. – 560 с.
7. **Дьяконов В.П., Абраменков И.В.** MATLAB 5.0/5.3. Система символьной математики. – М.: Нолидж, 1999. – 640 с.
8. **Ильин С.П**. Вариационное исчисление с применением MATLAB. – Харьков: ХПИ, 2001. – 107 с.
9. **Лавров К.Н., Цыплякова Т.П.** Финансовая аналитика. MATLAB-6. – Москва: Диалог-МИФИ, 2001. – 368 с.
10. **Лазарєв Ю.Ф.** Початки програмування в середовищі MATLAB. Навч. посібник. – К.: “Політехніка”, 2000. – 396 с.
11. **Лазарев Ю.** MATLAB 5.x. – К.: BVH, 2000. – 384 с.
12. **Мартынов Н.Н.** Введение в MATLAB 6.x.– М.:Кудиц-образ,2002.– 352 с.
13. **Методи** обчислень на персональному комп‘ютері. Ч. 1: Системи лінійних алгебраїчних рівнянь. (Уклад. Сулима І.М., Мейш В.Ф., Гаєв Є.О.). – К.: Нац. аграрн. ун-т, 2002. – 45 с.
14. **Мєтьюз Д.Г., Финк К.Д.** Численные методы. Использование MATLAB. – М.: «Вильямс», 2001. – 720 с.
15. **Потемкин В.Г.** Система инженерных и научных расчетов MATLAB 5.x, в 2х томах. – М.: МИФИ, 1999.

16. **Потемкин В.Г., Рудаков П.И.** MATLAB 5 для студентов. – М.: Диалог–МИФИ. 1999. – 448с.
17. **Проведение** математических расчетов с использованием системы MATLAB. Метод. пособие. – Харьков: НТУ "ХПИ", 2001. – 56 с.
18. **Чен К., Джиблин П., Ирвинг А.** MATLAB в математическом исследовании. – М.: Мир, 2001. – 346 с.

ДОДАТКОВИЙ

19. **Акастьолова Н.О., Джур О.Є.** Розв'язання інженерних та економічних задач в Excel. – Дн-ск: РВВ ДНУ, 2001. – 96 с.
20. **Артамонова Л.А., Халепа Н.В., Клочков В.Е.** Графические возможности среды MathCAD Plus 6.0. – Новомосковск, 1999. – 610 с.
21. **Дьяконов В.** Mathcad 2000. Учебный курс. – СПб: Питер, 2000. – 592 с.
22. **Дьяконов В.** Mathematica 4. Учебный курс. – СПб:Питер, 2001. – 656 с.
23. **Жилкин В.А.** Применение системы MATHCAD при решении задач прикладной математики. – Челябинск, 2001. – 73 с.
24. **Карабутов Н.**Н. Математическое моделирование и анализ систем в среде MATHCAD.Учебн. пособие. – М., 2001. – 79 с.
25. MATHCAD 6.0 PLUS. Финансовые, инженерные и научные расчеты в среде Windows95. – М.: Изд. дом «Филинъ», 1997. – 712 с.
26. **Петров Ю.А.** Современные математические пакеты. Учебн. пособие. – Комсомольск-на-Амуре, 2001. – 148 с.
27. **Плис А.И., Сливина Н.А.** MathCAD 2000: Математический практикум для экономистов и инженеров.– М.:Финансы и статистика,2002. – 656 с.
28. **Пономарев О.П. и др.** Использование математического пакета Maple Y при интегрировании дифференциальных уравнений. Учеб. пособие. – М.: РГОТ УПС, 2000. – 76 с.
29. **Скворцов В.** Explore and Derive. Исследуйте и получите. – Казань: КГТУ, 2001. – 60 с.
30. **Берман Т.Н.** Сборник задач по курсу математического анализа. – М.: Наука, 1985.

31. **Вірченко Н.О., Ляшко І.І.** Графіки елементарних та спеціальних функцій. – К.: Наук. думка, 1996. – 582 с.
32. **Демидович В.П., Марон И.А.** Основы вычислительной математики. – М.: Наука, 1966. – 664 с.
33. **Демидович Б.П., Марон И.А., Шувалова Э.З.** Численные методы анализа. – М.: Наука, 1967. – 368 с.
34. **Дубовик В.П., Юрик І.І.** Вища математика. Навч. посібник. – К.: А.С.К., 2001. – 648 с.
35. **Зеленский К.Х. и др.** Компъютерные методы прикладной математики. – К.: Дизайн, 1999. – 352 с.
36. **Лященко М.Я., Голованъ М.С.** Чисельні методи. – К.: Либідь, 1996. – 288 с.
37. **Методи обчислень:** Практикум на ЕОМ: Навч. посібник / Бурківська В.Л., Войцехівський С.О. та ін. – К.: Вища шк., 1995. – 303 с.
38. **Сулима И.М., Гавриленко С.И., Радчик И.А., Юдицкий Я.А.** Основные численные методы и их реализация на микрокалькуляторах. – К.: Вища шк., 1987. – 310 с.
39. **Анго А.** Математика для электро- и радиоинженеров. - М.: Наука, 1964. – 772 с.

Навчальний посібник

Гаєв Євген Олександрович
Нестеренко Борис Миколайович

УНІВЕРСАЛЬНИЙ МАТЕМАТИЧНИЙ ПАКЕТ MATLAB І ТИПОВІ ЗАДАЧІ МАТЕМАТИЧНОГО МОДЕЛЮВАННЯ

Редактор
Технічний редактор
Коректор

Піписано до друку Формат . Папір типограф.
Офсетний друк. Ум.-друк. с.
Наклад екз. Замовлення № .

Видавництво Національного авіаційного університету.
03058, Київ-58, проспект космонавта Комарова, 1