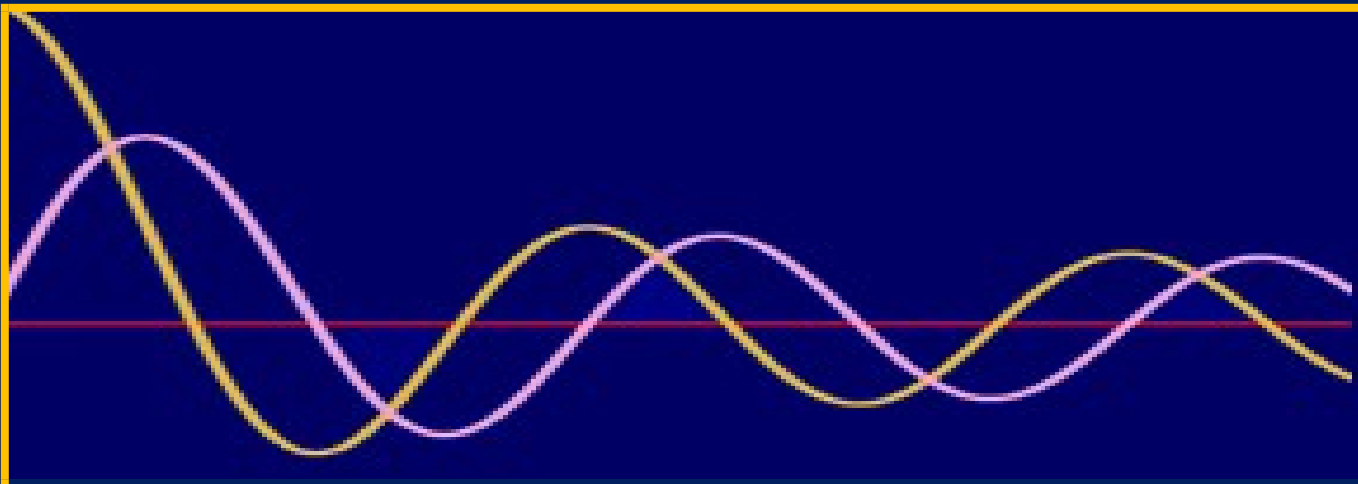
Для студентів мех.-мат. факультету

Б.П.ДОВГИЙ
Є. С. ВАКАЛ

ІНФОРМАЦІЙНІ СИСТЕМИ ТА ТЕХНОЛОГІЇ



Навчальний посібник

СПЕЦІАЛЬНІСТЬ
МАТЕМАТИКА

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

Б.П.ДОВГИЙ
Є. С. ВАКАЛ

ІНФОРМАЦІЙНІ СИСТЕМИ ТА ТЕХНОЛОГІЇ

Навчальний посібник

для студентів механіко-математичного-факультету

Київ – 2021

УДК 51-37:519.6(0.75.8)
Д 58

Рецензенти:
д-р фіз.-мат. наук, проф. О. В. Капустян,
канд. фіз.-мат. наук, доц. О. С. Тригуб

Рекомендовано до друку
вченою радою механіко-математичного факультету
(протокол № 1 від 27 серпня 2020 року)

Довгий Б.П.

Д58 Інформаційні системи та технології : навчальний посібник для студентів механіко-математичного факультету / Б.П.Довгий, Є. С. Вакал. – К.: Видавець Кравченко Я.О., 2021. – 111 с.

ISBN 978-617-7700-81-3

Навчальне видання містить теоретичні відомості, приклади завдань та методичні рекомендації до їх виконання із дисципліни "Інформаційні системи та технології", що викладається студентам механіко-математичного факультету Київського національного університету імені Тараса Шевченка. За змістом видання поділено на теми згідно з навчальною програмою дисципліни. Головна увага приділяється питанням застосування інформаційних систем і технологій для розв'язання задач, що виникають в галузі математики. Для їх глибшого і змістовнішого розуміння навчальне видання доповнено короткими поясненнями із відповідних розділів наближених обчислень та текстами програм. У посібнику як базовий інструмент навчання обрано пакет MATLAB, в якому реалізовано низку стандартних і спеціальних математичних операцій.

Для студентів, аспірантів і викладачів механіко-математичного факультету та факультету комп'ютерних наук та кібернетики. Посібник буде корисним для фахівців з обчислювальної математики, які використовують у наукових дослідженнях системи комп'ютерної математики.

УДК 51-37:519.6(0.75.8)

ISBN 978-617-7700-81-3

ПЕРЕДМОВА

Інформаційні технології (ІТ) – це процеси, які використовують сукупність засобів і методів збирання, опрацювання і передавання даних (первинної інформації) для отримання інформації нової якості про стан об'єкта, процесу або явища (інформаційного продукту). ІТ є процесом, який складається із чітко регламентованих правил виконання операцій, дій, етапів різного ступеня складності над даними, які зберігаються в комп'ютерах.

Інформаційна система (ІС) – це організаційно-впорядкована взаємопов'язана сукупність засобів і методів ІТ, які використовуються для збереження, обробки і отримання інформації з метою досягнення поставленого завдання. Таке поняття ІС передбачає використання ЕОМ і засобів зв'язку як основних технічних засобів обробки інформації, що реалізують інформаційні процеси передачі інформації, необхідної для прийняття рішення, розв'язання задачі із будь-якої предметної області.

Як бачимо, визначення ІС і ІТ є досить широкими поняттями, які відображають сучасне уявлення про процеси перетворення інформації. В залежності від конкретної області використання ІС можуть дуже сильно відрізнятися за своїми функціями, архітектурою, реалізацією.

У навчальному посібнику основну увагу приділено задачам, що виникають у галузі математики. В зв'язку з цим розглядаються питання застосування відповідних інформаційних систем і технологій, які дозволяють:

- представляти дані максимально наближено до математичної області;
- просто, і в той же час ефективно, розв'язувати широкий спектр різноманітних задач математики, як в чисельному так і в символьному вигляді;
- відображати отримані результати у вигляді різноманітних таблиць, діаграм, 2D, 3D-графіків і аналітичних виразів.

Сучасні інтегровані середовища, що ґрунтуються на цих засадах, є універсальними системами комп'ютерної математики, серед яких найвідомішими на сьогоднішній день є MATLAB, Mathcad, Maple, Mathematica, Statistica та Derive. У посібнику як базовий інструмент навчання обрано математичний пакет MATLAB, в якому реалізовано низку стандартних і спеціальних математичних операцій [1, 2, 7 – 9, 12, 14, 17]. Для їх глибшого і змістовнішого розуміння навчальне видання доповнено коротким теоретичним матеріалом із відповідних розділів наближених обчислень і завданнями для самостійного опрацювання.

ТЕМА 1
ОСНОВНІ ОБ'ЄКТИ СИСТЕМИ MATLAB

Математичний пакет MATLAB має власну мову програмування, інтерактивне середовище для розробки алгоритмів, візуалізації та аналізу числових розрахунків.

Для практичної роботи будемо використовувати програмний комплекс MATLAB 7.6.0 (R2008a). Виконання програм відбувається в режимі *інтерпретації* операторів.

Правила іменування. У системі MATLAB для іменування об'єктів використовуються *ідентифікатори*, які обов'язково починаються з букви і можуть містити літери латинського алфавіту, цифри й символ підкреслення "_". Ідентифікатор може мати скільки завгодно символів, але значущими є тільки перші (початкові) 31 символ. Доцільно використовувати для позначення об'єктів програми змістовні імена. MATLAB не допускає використання кирилиці в іменах об'єктів.

Типи даних. Майже всі дані в системі MATLAB, зі структурної точки зору, є масивами, отже:

скалярна змінна	–	матриця розмірності 1×1
вектор-рядок з N елементів	–	матриця розмірності 1×N
вектор-стовпець з N елементів	–	матриця розмірності N×1

Для маніпуляції об'єктами в системі MATLAB використовуються *дескриптори* функцій – скалярні величини, які зберігають вказівник на функцію і дозволяють передати її іншій функції або викликати локальну функцію ззовні основної функції.

1. Для *числових* типів даних існують наступна типізація (клас) і імена функцій *перетворення* типів:

`single, double`	– дійсний/комплексний 32 біт, 64 біт;
`int8, int16, int32, int64`	– знаковий цілочисельний 8 біт, 16 біт, 32 біт, 64 біт;
`uint8, uint16, uint32, uint64`	– беззнаковий цілочисельний 8 біт, 16 біт, 32 біт, 64 біт.

За замовчуванням використовується тип `double`, який має найбільшу точність представлення дійсного числа з діапазону від 10^{-308} до 10^{308} (приблизно 16 значущих цифр) і тому є універсальним типом. Однак, при необхідності економити пам'ять ЕОМ, можна вказувати самостійно бажаний тип, наприклад,

```
A = uint8([1,2,3,4,5])
```

Ще одним варіантом економії пам'яті ЕОМ, у випадку роботи із сильно розрідженими масивами, є використання класу `sparse`. Перетворення "повної" матриці `A` (з багатьма нулями) в "розріджену" матрицю `B` здійснюється за допомогою функції `B=sparse(A)`. Зворотне перетворення виконується функцією `full(B)`. Система MATLAB має великий арсенал вбудованих функцій для роботи з `sparse`-матрицями. Враховуючи певну специфіку використання цього типу, ми не будемо докладно зупинятися на його вивченні.

Константи числових типів даних записуються як і в інших алгоритмічних мовах, і тип константи визначається при її запису, наприклад,

```
17, -25                    % цілі
435.123, -2.345, .0001     % дійсні (запис із фіксованою крапкою)
1.248e-12, 1.26E8          % дійсні (запис в експоненціальній формі)
13-2.5i, -2.0345j          % комплексні
```

2. Для *логічного* типу використовується клас `logical` (і відповідна функція перетворення типу) , наприклад, означення логічної змінної:

```
a = logical(1); B = 1>0; B(2) = 1<0; B(3) = true; B(4) = false;
C = logical([1,0,0;0,1,0;0,0,1]);
```

Значенням логічних констант є 1 або 0, яким відповідає *Істина* і *Хибність*. Можна використовувати і позначення `true`, `false`. Дані цього типу мають розмір 1 байт і використовуються для побудови та обчислення *умов* й індексації масивів, наприклад, у випадку заданої вище логічної змінної `C` оператор `Y(C)=X(C)` або `Y(C==0) = X(C==0);` реалізує заміну недіагональних елементів матриці `Y` відповідними елементами матриці `X`.

Зазначимо, що коли елементи матриці `C` визначаються із цілих чисел 1, 0 (`C=[1,0,0;0,1,0;0,0,1];`), то перший варіант запису призводить до помилки.

3. Для *рядкового* типу використовується клас `char` (і відповідна функція перетворення типу). Константи-рядки записуються за допомогою обмежувача – апострофа, наприклад,

```
s='Hello'; x='Розв''язок рівняння Бюргерса'; al='{\alpha}';
```

У результаті отримуємо змінні рядкового типу `s,x` у вигляді лінійного масиву з елементами-символами (`s= ['H','e','l','l','o']`). Команда `a = double(s)` перетворює символьний масив у числовий, що дає `A = 72 101 108 108 111,` де елементи масиву `A` є кодами символів у певній *кодовій сторінці* (наприклад, "Windows−1251").

Команда `c = char(A)` виконує обернене перетворення числового вектора в рядок і дає `c = 'Hello'`. Для визначення кодової сторінки, встановленої на комп'ютері, необхідно виконати команду `slCharacterEncoding`, результатом виконання якої може бути, наприклад, таке повідомлення

```
ans =
windows-1251
```

З огляду на метод збереження рядків, при групуванні рядків в масив кожний рядок буде (при необхідності) доповнений справа пропусками (`' '`) до максимальної довжини рядка, наприклад, оператор `A = ['a','ab','abcd']` утворює масив `A` з рядками однакової довжини, рівної 4 : `A = ['a  ','ab  ','abcd']`.

4. Для *структурного* типу (тип *запис*) використовується клас `struct` (і відповідна функція перетворення типу). Цей тип даних групує дані різних типів із використанням контейнерів даних, які називаються *полями*. Доступ до полів структури виконується за допомогою нотації у формі *structName.fieldName*. Наприклад, ланцюжок команд

```
field1 = 'p1'; field2 = 'p2';
value1 = [1 2 3 4 5]; value2 = 'текст';
S = struct(field1,value1,field2,value2);
```

утворює структуру `S`. Для доступу до значення поля `p1`, можна використати один із наступних еквівалентних синтаксисів: `S.p1`, `S.(field1)`, `S.('p1')`. Створити структуру можна й іншим способом, наприклад, команди:

```
S(2).p1 = [1.2,3,-4,1,34]; S(2).p2 = '4'; S(2).p3 = C;
```

не тільки перетворюють *S* в масив, але і додають нове поле з іменем *p3*, при цьому значення *S(1).p3* порожнє! Різні поля можуть зберігати різні типи даних, у тому числі й структури, але в конкретному полі масиву структур має бути визначений фіксований тип.

При виведенні структури (скаляра) відображаються як імена полів, так і значення. При виведенні структури (масиву) виводиться лише інформація про те, що це – масив-структура, його розмірність й імена полів структури. Якщо необхідно вивести тільки імена полів, то можна використовувати функцію *fieldnames* з одним аргументом – іменем масиву структур, наприклад, *fieldnames(S)*. Для виведення значень відповідного поля масиву структур необхідно вказати *ім'я_масиву.ім'я_поля*, наприклад, *S.p2*.

Для видалення деякого поля з усіх елементів масиву структур, потрібно застосувати функцію *rmfield*, наприклад, *S = rmfield(S,'p3')* – видалення поля з ім'ям *p3* з масиву структур *S*.

Структури досить часто використовуються у вбудованих функціях MATLAB (див., наприклад, *spline*, *ode45*, *bvpinit*, *pdepe*).

5. Для типу *комірка* використовується клас *cell* (і відповідна функція перетворення типу). Цей тип даних групує дані різних типів і структур даних з використанням універсальних контейнерів даних, які доступні за допомогою індексу. Наприклад, команда

```
A={3+2i,[1 2 3],[1 2 3;4 5 6;7 8 9],'Hello'};
```

створить масив комірок із об'єктів різного типу. Таким чином, конструктор *{}* діє подібно конструктору *[]* для масивів, але він об'єднує в масив комірок дані різного типу. Ще приклад: для зберігання матриць однакової розмірності можна використовувати тривимірний масив, а для зберігання матриць різної розмірності – масив комірок

```
M{1}=magic(4); M{2}=magic(3);
M{3}=[1 2 3 4 5]; M{4}=[1 2 3;4 5 6];
```

Масив комірок зручно використовувати і для зберігання рядків, наприклад, оператор *A = {'a','ab','abcd'}* створює масив комірок *A* з рядками різної довжини. Звичайно, масиви комірок можуть бути і багатовимірні. Наприклад, за командою

```
B = cell(2,2);
```

задається матриця *B* розмірності 2×2 з комірок із порожніми (невизначеними) значеннями.

Таку ініціалізацію доцільно виконувати у випадку створення масиву комірок із наступним заповненням значень елементів у циклі, що підвищує швидкість виконання програмного коду.

MATLAB відображає масив комірок у скороченій формі. Для виведення всіх значень комірок, необхідно застосувати функцію *celldisp*, наприклад, *celldisp(A)*. Для відображення значення конкретної комірки – *A{2}*, а для виведення конкретного значення в комірці – *A{2}(2,3)*. Необхідно звернути увагу на різне відображення результатів виконання команд *A{2}* і *A(2)*:

```
A{2}
ans =
     1 2 3
A(2)
ans =
     [1x3 double]
```

Отримані результати показують, що індекс у фігурних дужках *A{k}* означає вміст (значення) відповідної комірки, а в круглих *A(k)* – це комірка з відповідним вмістом.

Для виведення структури масиву комірок у вигляді графічного зображення використовується функція *cellplot*, наприклад, *cellplot(A)*.

Система MATLAB не очищає масив комірок при виконанні оператора присвоєння і при цьому в пам'яті може залишатись стара інформація в незаповнених комірках. Тому корисно перед виконанням оператора присвоєння *A = …;* скористатись командою *clear A;*

У системі MATLAB тип *cell* зручно використовувати при програмуванні функції зі змінною кількістю аргументів. Для цього аргументом функції вказується ключове слово *varargin*, яке інтерпретується в тілі функції як масив комірок. Дану ситуацію буде проілюстровано при вивченні функцій.

У пунктах 1 – 5 розглянуто основні типи даних, які надалі будуть використовуватися. Зазначимо, що для визначення типу конкретного об'єкту необхідно застосовувати функцію *class* до імені цього об'єкту.

З появою нових релізів системи MATLAB з'являються нові типи даних, які корисні в деяких конкретних випадках. Звичайно, при цьому MATLAB доповнюється множиною інструментів для роботи з цими типами даних. В нових релізах також можуть оновлюватись і доповнюватись вбудовані функції для покращення ефективності роботи пакету.

Змінні. *Змінна* – це певна поіменована величина, яка здатна зберігати деякі, звичайно різні за значеннями, дані відповідних типів. Тип змінної заздалегідь не декларується, а визначається *виразом*, значення якого присвоюється або переприсвоюється змінній. Крім імен змінних і поіменованих констант користувача, в пакеті є *системні змінні*, які мають фіксовані імена і значення. Основні системні змінні, що використовуються в пакеті MATLAB, наведено в таблиці:

Основні системні змінні	
Ім'я	**Призначення**
i, j	уявна одиниця комплексного числа
pi	число π
eps	похибка операцій над числами з плаваючою крапкою
realmin	найменше число з плаваючою крапкою
realmax	найбільше число з плаваючою крапкою
inf	значення машинної нескінченності (∞)
ans	змінна, що зберігає результат останньої операції
NaN	вказівка на нечисловий характер даних (Not-a-Number), позначення невизначеного результату (наприклад, 0/0; *inf* / *inf*)
nargin	кількість вхідних аргументів у функції

Основні системні змінні	
Ім'я	**Призначення**
nargout	кількість вихідних аргументів у функції
computer	тип комп'ютера
flops	кількість операцій з плаваючою крапкою
version	номер версії MATLAB

Системні змінні можна перевизначити на термін поточної сесії (виконання програми), а відновити початкове значення можна оператором `clear`, наприклад,

```
eps = 1e-6;
…
clear eps;
```

Побудова виразів. *Вирази* використовуються для запису послідовності певних обчислень (дій/операцій) над даними для отримання значення конкретного типу. Зазвичай, побудова виразу кожного типу визначається строго, за індуктивними правилами. При цьому чітко визначається пріоритет виконання операцій, як для групи операцій (по типам), так і всередині групи.

В MATLAB визначено наступний пріоритет (у порядку "старший" – "молодший") за обчисленнями:

1. *арифметичні*;
2. *умови* (в пріоритеті):
   2.1. *відношення* (*предикати*);
   2.2. *логічні операції*.

Порядок виконання операцій в групі може бути змінено за допомогою круглих дужок.

При побудові виразів використовуються не тільки операції, але й *спеціальні символи,* наведені нижче:

Операції і спеціальні символи	
Позначення	**Призначення**
:	операція "двокрапка" (має декілька варіантів використання)
( )	дужки для групування операцій або аргументів; зазначення індексів
[ ]	конструктор для формування масивів
{ }	конструктор для формування масивів комірок; зазначення індексів у комірках
.	крапка (має декілька варіантів використання)
..	посилання на батьківський каталог директорії
...	текст рядка команди продовжується на наступний фізичний рядок
пропуск	розділовий знак (між словами, даними, змінними, константами)
end	в індексі масиву означає останній елемент/стовпець/рядок
,	розділовий знак
;	блокування виведення інформації на екран або поділ рядків у масиві
%	знак початку коментаря

В *арифметичних операціях* використовуються числові дані однакової розмірності. Якщо один із операндів є скаляром, а інший ні, то скаляр *поширюється* до розмірів іншого операнду. Набір арифметичних операцій у MATLAB складається з наступних операцій:

Арифметичні операції	
Позначення операції	**Призначення**
+	додавання
-	віднімання
*	добуток
.*	поелементний добуток
^	піднесення до степеня
.^	поелементне піднесення до степеня
\	"ліве" матричне ділення (якщо $AX = B$, де X – невідоме, тоді $X = A \setminus B$)
/	"праве" матричне ділення (якщо $XA = B$, де X – невідоме, тоді $X = B / A$)
.\	"ліве" поелементне ділення
./	"праве" поелементне ділення
'	операція транспонування для дійсних масивів і транспонування з одночасним комплексним спряженням для комплексних масивів
.'	операція транспонування

Для групи арифметичних операцій встановлено наступний пріоритет:

Пріоритет арифметичних операцій	
Рівень	**Операція**
1	" .' ", " .^ ", " ' ", " ^ "
2	унарні " + ", " – "
3	" .* ", " ./ ", " .\ ", " * ", " / ", " \ "
4	бінарні " + ", " – "
5	оператор "двокрапка" " : "

Усередині кожного рівня оператори мають однаковий пріоритет і обчислюються зліва направо.

Операції відношення (предикати) мають два операнди і використовуються для порівняння двох величин, у тому числі і масивів. Вони можуть записуватись у вигляді бінарної операції або звернення до функції і представлені в наступній таблиці:

Операції/функції відношення			
Позначення операції	**Функція**	**Призначення**	**Приклад**
==	*eq*	дорівнює	*x*==*y* або *eq*(*x*,*y*)
~=	*ne*	не дорівнює	*x*~=*y* або *ne*(*x*,*y*)
<	*lt*	менше	*x*<*y* або *lt*(*x*,y)
>	*gt*	більше	*x*>*y* або *gt*(*x*,*y*)
<=	*le*	менше або дорівнює	*x*<=*y* або *le*(*x*,*y*)
>=	*ge*	більше або дорівнює	*x*>=*y* або *ge*(*x*,*y*)

При обчисленні предиката ми отримуємо результат 1 – *Істина* або 0 – *Хибність*.

Зазначимо, що порівняння комплексних чисел операціями "<", "<=", ">", ">=" реалізується шляхом порівняння їх дійсних частин.

Логічні операції і функції застосовуються поелементно до даних і використовуються для побудови булевих виразів (умов):

Логічні операції і функції			
Позначення операції	**Функція**	**Призначення**	**Приклад**
~	*not*	заперечення	~*x* або *not*(*x*)
&	*and*	кон'юнкція	*x*&*y* або *and*(*x*,*y*)
|	*or*	диз'юнкція	*x*|*y* або *or*(*x*,*y*)
	xor	виключаюче "або"	*xor*(*x*,*y*)
	all	*Істина,* якщо всі елементи вектора ≠0	*all*(*x*)
	any	*Істина,* якщо не всі елементи вектора =0	*any*(*x*)
	exist	перевірка існування змінної чи функції	*exist*('*x*')
	find	пошук індексу ненульового елемента (індексів, що задовольняють певні умові)	*find*(*x*)
	isempty	*Істина,* якщо масив порожній	*isempty*(*x*)
	isinf	*Істина,* якщо елемент = ∞	*isinf*(*Inf*)
	isletter	*Істина,* якщо символом є буква	*isleter*('*a*')
	isnan	*Істина,* якщо елемент не є числовим	*isnan*(*NaN*)
	isreal	*Істина,* якщо елемент дійсний	*isreal*(1.2)
	isstr	*Істина,* якщо елементом є текстовий рядок	*isstr*('*qwerty*')

У виразах можливе звернення до *функцій*, які можуть бути *вбудованими* (внутрішніми) і *М–функціями* (зовнішніми). Зовнішні функції створюються користувачем і містять свої визначення у М–файлах. Вбудовані функції зберігаються у відкомпільованому ядрі системи MATLAB, внаслідок чого вони виконуються гранично швидко. Пакет MATLAB містить значну кількість елементарних і спеціальних функцій. Зі списком елементарних функцій можна ознайомитись за допомогою команди `help elfun`, а зі списком спеціальних функцій – за допомогою команди `help specfun`.

Перелік деяких вбудованих функцій наведено нижче:

Тригонометричні та гіперболічні функції			
Тригонометрична функція	**Обернена ригонометрична функція**	**Гіперболічна функція**	**Обернена гіперболічна функція**
sin(*x*)	*asin*(*x*)	*sinh*(*x*)	*asinh*(*x*)
cos(*x*)	*acos*(*x*)	*cosh*(*x*)	*acosh*(*x*)
tan(*x*)	*atan*(*x*)	*tanh*(*x*)	*atanh*(*x*)
cot(*x*)	*acot*(*x*)	*coth*(*x*)	*acoth*(*x*)

Трансцендентні функції	
Назва	**Призначення**
exp(*x*)	експонента *x*
log (*x*)	натуральний логарифм *x*
*log*2(*x*)	логарифм *x* за основою 2
*log*10(*x*)	логарифм *x* за основою 10
*pow*2(*x*)	експонента *x* за основою 2
sqrt(*x*)	корінь квадратний з *x*
*nextpow*2(*x*)	найближча степінь експоненти за основою 2 ≥ *x*

Функції роботи з комплексними числами	
Назва	**Призначення**
abs(*z*)	модуль *z* (|*z*|)
angle(*z*)	аргумент *z* (*arg*(*z*))
conj(*z*)	спряжене до *z* (*z**)
imag(*z*)	уявна частина *z* (*Im*(*z*))
real(*z*)	дійсна частина z (*Re*(*z*))

Функції заокруглення і модульна арифметика	
Назва	**Призначення**
abs(*x*)	абсолютна величина *x* (|*x*|)
fix(*x*)	ціла частина *x* ([*x*])
floor(*x*)	округлення *x* до найближчого меншого цілого (результат – ціле значення)
ceil(*x*)	округлення *x* до найближчого більшого цілого (результат – ціле значення)
round(*x*)	округлення *x* до цілого (результат – дійсне значення)
mod(*x*,*y*)	залишок від цілочисельного ділення *x* на *y* з урахуванням знаку
rem(*x*,*y*)	залишок від цілочисельного ділення *x* на *y*. Порівняйте: *mod*(5.345,2), *rem*(5.345,2); *mod*(–5.5,2), *rem*(–5.5,2)
sign(*x*)	знак числа

Теоретико-числові функції	
Назва	**Призначення**
factor(*n*)	розклад числа *n* на прості множники
isprime(*n*)	*Істина*, якщо число *n* просте
primes(*n*)	формування списку простих чисел, які не перевищують *n*
gcd(*n*,*m*)	найбільший спільний дільник *n*,*m*
lcm(*n*,*m*)	найменше спільне кратне *n*,*m*
rat(*n*)	подання числа *n* у вигляді ланцюгового дробу
rats(*n*)	наближене число у вигляді раціонального дробу вигляду *a/b*
perms(1:*n*)	можливі перестановки елементів заданого числового вектора
nchoosek(*n*,*k*)	число сполучень (C_n^k)

Функції для операцій з елементами матриць	
Назва	**Призначення**
diag(*A*)	створення або видобування діагоналі матриці. Якщо *A* є вектором, то створюється квадратна матриця, в якій елементи вектора *A* розміщені на головній діагоналі. Якщо *A* є матрицею, створюється вектор-стовпець, що складається з елементів головної діагоналі матриці *A*
fliplr(*A*)	відбиття стовпців матриці *A* відносно вертикальної осі
flipud(*A*)	відбиття рядків матриці *A* відносно горизонтальної осі
isreal(*A*)	*Істина*, якщо всі елементи матриці *A* дійсні
reshape(*A*,*m*,*n*)	зміна розмірності матриці без зміни кількості її елементів. (див. *help reshape*) Утворюється матриця розмірності *m*×*n*. Нехай *A* – матриця розмірності 3×4, тоді можна утворити матриці: *reshape*(*A*,1,12); *reshape*(*A*,12,1); *reshape*(*A*,2,6); *reshape*(*A*,3,[ ]); *reshape*(*A*,[ ],4);
*rot*90(*A*,*k*)	поворот матриці *A* на 90**k* градусів, де *k* = 1, –1, 2, –2,…. Якщо *k* відсутнє, то приймається *k* = 1
tril(*A*,*k*)	створення лівої трикутної матриці, починаючи з діагоналі (*головна* + *k*), де при *k*>0 – відлік вправо, а при *k*<0 – вліво від головної діагоналі. Якщо *k* відсутнє, то приймається *k*=0.
triu(*A*,*k*)	створення правої трикутної матриці, починаючи з діагоналі (*головна* + *k*), де при *k*>0 – відлік вправо, а при *k*<0 – вліво від головної діагоналі. Якщо *k* відсутнє, то приймається *k*=0
:	виділення стовпця/рядка в матриці або генерація елементів вектора
[*n*,*m*]=*size*(*A*)	визначення розмірності матриці *A* (кількість рядків і стовпців)
ndims(*A*)	визначення кількості вимірів у просторовому представленні масиву *A*
s=*length*(*A*)	максимальний розмір матриці *A* (*s* = *max*(*size*(*A*))) – максимум із кількості рядків і стовпців

Функції для роботи з рядками символів	
Назва	**Призначення**
abs('*i*')	код символу *i*
blanks(*n*)	створення рядка з *n* пробілів
deblank(*s*)	видалення пробілів у кінці рядка
double('*xyz*')	перетворення символів рядка '*xyz*' у числове значення
strcmp(*s*,*t*)	порівняння рядків *s* і *t*
upper(*s*)	переведення всіх символів рядка *s* у верхній регістр
lower(*s*)	переведення всіх символів рядка *s* у нижній регістр
setstr(*n*)	перетворення числа *n* у рядок
*int*2*str*(*n*)	перетворення цілого числа *n* у рядок
*num*2*str*(*x*)	перетворення дійсного числа *x* у рядок
*str*2*num*(*s*)	перетворення рядка *s* у число
findstr(*s*,*p*)	пошук місця рядка *p* у рядку *s*
strrep(*s*,*c*,*p*)	заміна в рядку *s* рядка *c* на рядок *p*
strcat(*s*,*c*)	горизонтальне об'єднання рядків
strvcat(*s*,*c*)	вертикальне об'єднання рядків
eval(*s*)	обчислення рядкових виразів

Функції часу та дати	
Назва	**Призначення**
clock	поточний час і дата у векторній формі [рік, місяць, день, година, хвилина, секунда]
date	поточна дата у формі рядка
now	поточний час і дата у формі числа
datestr(*d*,*n*)	перетворення числового формату дати *d* в один із 19 вихідних рядкових форматів дати і часу (*n* – номер формату від 0 до 18)
etime(*t*2,*t*1)	визначення тривалості проміжку часу (в сек), заданого векторами *t*1 і *t*2

Враховуючи певну специфіку спеціальних математичних функцій, наведемо тільки їх відповідні назви:

- функції Бесселя (`bessel`, `besseli`, `besselj`, `besselk`, `bessely`);
- бета-функція (`beta`, `betainc`, `betaln`);
- еліптичні функції Якобі (`ellipj`, `ellipke`);
- гама-функції (`gamma`, `gammainc`, `gammaln`);
- функції Лежандра (`legandre`).

Для детального ознайомлення з відповідною функцією можна використати команду `help` <*name_function*>, де <*name_function*> – ім'я функції.

ТЕМА 2
ОСНОВНІ СТРУКТУРИ КЕРУВАННЯ АЛГОРИТМІЧНОЇ МОВИ MATLAB

Враховуючи, що алгоритми, створювані користувачем, як правило, складаються із серії команд і можуть виконуватись не один раз, використання інтерактивного середовища для запуску і виконання команд у командному вікні не є найкращим варіантом роботи з пакетом MATLAB. Отже потрібно створювати програми. Для цього в системі MATLAB є своя алгоритмічна мова [7, 10, 12].

Ланцюг команд. Команди мови, як і команди у командному вікні, записуються по одній у фізичний рядок. Якщо після запису тексту команди відсутній розділювач ";", то після виконання цієї команди на екран буде виведено результат. За наявності знаку ";" здійснюється блокування виведення. Для запису декількох команд в одному фізичному рядку використовується розділювач "," (без блокування виведення) або розділювач ";". Частину довгого виразу можна перенести на наступний рядок за допомогою символу "…".

Оператор присвоєння. В системі MATLAB можна надавати змінним певні значення. З цією метою використовується оператор присвоєння, який має синтаксис: *x = E*, де *x* – ім'я змінної, а *E* – вираз.

Семантика присвоєння реалізується у наступний спосіб – обчислюємо значення (позначимо його через *e*) виразу *E* у правій частині оператора, яке і стає значенням змінної *x*. В MATLAB пам'ять під змінні заздалегідь не виділяється, тому, якщо змінна *x* не існує, то вона створюється. У випадку існування *x* попереднє значення змінної *x* стає недосяжним, а тип *x* приводиться до *class(e)*.

Приклад 1. Знаходження коренів сіткової функції.

```
%%pl2_1.m
clear all         % очистка робочої області
close all         % закриття усіх графічних вікон
clc               % очистка командного вікна
X = [-2 : 0.1 : 3];
Y = (X + 1) .*  ...   % приклад продовження команди на наступний рядок
    (X - 2);
% Знаходження коренів сіткової функції Y
n = length(X);  w = 1 : n - 1;
% пошук індексу масиву значень функції Y, для якого виконується умова,
% тоді значення елементу масиву X із цим індексом дає корінь
X(find(Y(w) .* Y(w + 1) < 0 | Y(w) == 0))
% Графічний розв'язок
% Виведення графіку функції і осі абсцис
plot(X, Y, X, zeros(n),'LineWidth',2), grid on
```

Розгалуження. У мові MATLAB дана *керуюча структура* представлена декількома варіантами, які мають свою специфіку виконання (семантику). Незалежно

від варіанту розгалуження його запис обов'язково має закінчуватися ключовим словом `end`. Тоді команда, розташована після нього, буде виконуватися наступною.

а) Синтаксис *оператора розгалуження* має вигляд:

```
if F
     P;
else
     Q;
end
```

Семантика: якщо після обчислення умови `F` отримали значення *Істина*, то виконується група команд `P`, а при значенні *Хибність* – група команд `Q`.

b) Синтаксис *спрощеного оператора розгалуження* –

```
if F
     P;
end
```

Семантика: якщо умова `F` істинна, то виконується група команд `P`, у протилежному випадку оператор не виконує жодної команди.

Зазначимо, що оператори розгалуження (як а) так і b)) можуть бути вкладеними.

c) Для спрощення запису вкладеності операторів розгалуження застосовується *оператор каскадного розгалуження*, який має наступний синтаксис:

```
if F_1
     P_1;
elseif F_2
     P_2;
     …
elseif F_n
     P_n;
else
     Q;
end
```

Семантика: послідовно перевіряємо значення умов `F_k` і перша з умов, яка істинна, приводить до виконання групи команд `P_k`. Якщо всі умови `F_k` хибні і наявна альтернативна частина `else`, то виконується група команд `Q`.

Приклад 2. Використання різних варіантів розгалуження.

```
%%pl2_2.m
% Приклади варіантів розгалуження
clear, clc
x = input('? x=');
% Спрощений оператор розгалуження
if ~isreal(x)
    x = real(x)
end
% Вкладене розгалуження
if x < -1
    y = -1;
else
    if x < 1
        y = x;
    else
        y = 1;
    end
end
```

```
disp(strcat('x=',num2str(x),'    y=',num2str(y)));
% Каскадне розгалуження
if x < -1
    y = -1;
elseif x < 1
    y = x;
else
    y = 1;
end
disp(strcat('x=',num2str(x),'    y=',num2str(y)))
% Каскадне розгалуження (інша форма запису)
if x < -1,     y = -1;
elseif x < 1,  y =  x;
else,          y =  1;
end
display(strcat('x=',num2str(x),'    y=',num2str(y)))
```

d) У деяких випадках зручно користуватись *оператором вибору*, який має такий синтаксис:

```
switch    E
    case    e_1
        P_1;
    case    e_2
        P_2;
   …
    case    e_n
        P_n;
    otherwise
        Q;
end
```

Тут `E` має бути виразом/змінною зі значенням скаляра або рядка символів, а `e_k` – скаляром, або рядком символів, або масивом комірок із скалярів чи рядків символів.

Семантика такої структури: послідовно перевіряється виконання умов `E == e_1`, `E == e_2`,… , і перша з умов `E == e_k`, яка істинна, призводить до виконання групи команд `P_k`. Якщо жодна з цих умов не справджується і наявна альтернативна частина `otherwise`, то виконується група команд `Q`.

Приклад 3. Використання оператора вибору для перевірки уведеного символу.

```
%%pl2_3.m
% Приклад з оператором вибору
clear all, close all, clc
% вводимо у вигляді запису рядкової константи ('d'|'2'|'('|'.'|'*')
x = input('Введіть символ =');
if isstr(x)
    x = x(1);
end
y = x;
if isletter(y)
    y = lower(y);
end
disp(strcat('Ви ввели символ="',x,'"'));
% Використання оператора вибору
```

```
switch y
    case {'a','b','c','d','e','f','g','h','i','j','k','l','m',...
          'n','o','p','q','r','s','t','u','v','w','x','y','z'}
        disp('Це буква англійського алфавіту');
    case {'0','1','2','3','4','5','6','7','8','9'}
        disp('Це цифра');
    case {'(',')','[',']','{','}'}
        disp('Це дужка');
    case '.'
        disp('Це крапка');
    otherwise
        disp('Інший символ');
end
```

Цикл. Для повторення виконання певної послідовності команд мова MATLAB має структуру керування, яка називається *циклом* або *ітерацією*. Існує декілька варіантів оператора циклу.

a) Синтаксис *оператора циклу з передумовою*:

```
while F      % заголовок циклу
    P;       % "тіло" циклу
end          % кінець циклу
```

Семантика такої структури: поки значення умови `F` є істинним, виконується група команд `P` тіла циклу. Як тільки значення `F` стає хибним, то здійснюється вихід з циклу, і керування передається командам, розташованим після оператора циклу, тобто після ключового слова `end`.

b) Синтаксис *оператора циклу за діапазоном значень (цикл з лічильником)*:

```
for k = e1 : e2 : e3  % заголовок циклу
    P;                % "тіло" циклу
end                   % кінець циклу
```

Тут `k` – змінна циклу (*лічильник*), `e1,e3` – початкове і кінцеве значення діапазону, `e2` – крок (при значенні `+1` його можна не вказувати) на який змінюється `k` після кожного виконання групи команд `P`. Цикл закінчується, якщо значення лічильника виходить за межі вказаного діапазону.

Наведений синтаксис діапазону відповідає арифметичній прогресії, проте можна використовувати й інші варіанти завдання діапазону:

- одновимірні масиви з числовими, символьними, рядковими елементами;
- одновимірні масиви комірок;
- двовимірні масиви.

Зверніть увагу (див. нижче приклад 4) на значення лічильника циклу `h`, які він отримує у випадку використання матриці в якості діапазону (варіанти завдання `H` і `H(:)`).

Приклад 4. Використання різних варіантів оператора циклу

```
%%pl2_4.m
% Приклади операторів циклу
clear all, close all, clc

% Цикл з передумовою
% Обчислення константи eps
e = 1;
while 1 + e ~= 1
    e = e./2;
end
```

```
e = 2 .* e;
disp({'обчислене eps=', e, 'eps=', eps})
pause(2); clc

n = 5; H = zeros(n);
% Цикл за діапазоном значень (з "лічильником")
for i = 1 : n
    for j = 1 : n
        H(i, j) = 1 ./ (i + j - 1);
    end
end
disp('Матриця Гільберта (обчислена)')
H
disp('Використання вбудованої функції hilb')
H = hilb(n)
pause(2); clc

disp('Діапазон - одновимірний масив чисел H')
H = [-34, 25.8, 123e2]
for h = H
    h
end
pause(2); clc

disp('H - числова матриця, діапазон = H')
H = [-34, 25.8; 123e2, 2; -4, 234.5]
for h = H
    h
end
disp('H - числова матриця, діапазон = H(:)')
for h = H(:)
    h
end
pause(2); clc

disp('H - масив комірок, діапазон = H')
H = {-34, 2+5.8i, 'abc123'};
for h = H
    h
end
disp('H - масив комірок, діапазон = H(:)')
for h = H(:)
    h
end
```

При програмуванні ланцюга команд у тілі циклу можуть використовуватися (зазвичай у комбінації зі спрощеним оператором розгалуження) наступні команди:

continue – пропуск усіх наступних команд (до *end*) і передача керування на першу команду після заголовка циклу;

break – завершення циклу і передача керування наступній команді після *end*.

Введення, виведення, деякі допоміжні команди. Для інтерактивного введення даних (під час виконання програми на MATLAB) використовують команду *введення*:

```
X = input('Текст')
```

або ланцюг команд `input('Текст'), X = ans.`

Семантика команди:

1. виводиться на екран інформація *Текст*;
2. вводиться з клавіатури довільний вираз MATLAB, при цьому константи вводяться так, як вони визначаються в мові;
3. значення цього виразу присвоюється змінній *X* (у другому варіанті системна змінна *ans* теж отримує це значення).

У найпростішому випадку виведення значення на екран здійснюється записом виразу, після якого відсутній символ " ; ". Проте існує можливість виведення даних за допомогою *команд виведення*:

```
disp(E);
display(E);
```

Тут через *E* позначено вираз. У результаті виконання цих команд на екран (у різному вигляді) виводиться значення *E*.

При оформленні діалогу з користувачем корисні команди:

format – впливає на спосіб представлення інформації на екрані (детальнішу інформацію можна отримати за командою *help format*);

pause – призупиняє виконання програми до натиснення довільної клавіші на клавіатурі;

pause(n) – призупиняє виконання програми на *n* секунд.

М-сценарії та їх виклик. Зазначимо, що програми, створені користувачем, умовно можна поділити на дві групи:

- *файли-сценарії*, які не мають вхідних параметрів (будемо називати їх *М-файли*);
- *файли-функції*, які мають вхідні параметри (*М-функції*).

М-функції розглядаються в наступних розділах посібника. Зараз зосередимося на М-файлах.

Текст для М-файла (позначимо його ім'я *myfile*) можна підготувати у різний спосіб:

- у командному вікні командою *edit myfile* ;
- за допомогою відповідного пункту меню або піктограми на панелі інструментів;
- у середовищі довільного текстового редактору (при збереженні треба вказати і розширення файла *myfile.m*).

Текст файла бажано починати з коментаря з інформацією про його призначення, після якого записати такі команди:

```
clear all          % очистка робочої області
close all          % закриття всіх графічних вікон
clc                % очистка командного вікна
```

Зазначимо, що у М-файлі доступна вся інформація, яка є в робочій області.

Виклик М-файла відбувається введенням його імені (без розширення), тобто аналогічно виклику команди користувача у командному вікні, наприклад, *pl2_4* .

ТЕМА 3

АЛГОРИТМИ РОЗКЛАДУ МАТРИЦЬ. ОБУМОВЛЕНІСТЬ МАТРИЦЬ. ПРОБЛЕМА ВЛАСНИХ ЗНАЧЕНЬ

Для розуміння змісту деяких вбудованих функцій MATLAB при роботі з матрицями, розглянемо необхідні короткі теоретичні відомості з лінійної алгебри і методів обчислень.

LU-розклад матриці *A* для розв'язання СЛАР *Ax=b*

Теорема 3.1. Якщо $\det(A) \neq 0$, то існує матриця *перестановок* P, така, що $P \cdot A$ має відмінні від 0 *кутові мінори* :

$$\Delta_1 = a_{1,1}, \ \Delta_2 = \begin{vmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{vmatrix}, \ \ldots, \ \Delta_n = \det(A), \ \Delta_i \neq 0, \ i = \overline{1,n}. \tag{3.1}$$

Елементарна матриця перестановок визначається як: $P_{k,j} = (E)_{j \leftrightarrow k \text{ рядки}}$.

Теорема 3.2. Нехай всі кутові мінори матриці A відмінні від 0, тоді матрицю A однозначно можна представити у вигляді добутку двох трикутних матриць:

$$A = L \cdot U, \quad L = \begin{pmatrix} l_{1,1} & 0 & \cdots & 0 \\ l_{2,1} & l_{2,2} & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ l_{n,1} & l_{n,2} & \cdots & l_{n,n} \end{pmatrix}, \quad U = \begin{pmatrix} 1 & u_{1,2} & \cdots & u_{1,n} \\ 0 & 1 & \cdots & u_{2,n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}. \tag{3.2}$$

Таким чином, розв'язування СЛАР $A \cdot \vec{x} = \vec{b}$ можна реалізувати у вигляді послідовності наступних кроків:

$$A \ \Rightarrow \ [P, \ L, \ U], \tag{3.3}$$

$$U \cdot \vec{x} = \vec{f}, \ \ \vec{f} = L^{-1} P \cdot \vec{b}. \tag{3.4}$$

Знаходження розкладу (3.3) називається *прямим ходом*, а (3.4) – *зворотним ходом методу Гаусса*. З точки зору ефективності обчислень у алгоритм (3.3), (3.4) потрібно додати процедуру зменшення впливу *похибок заокруглень*.

Для реалізації (3.3) – (3.4) використовуються:

А) Алгоритм без явного знаходження матриць *P*, *L*.

Для k – стовпця матриці $A^{(k-1)}$, $(A^{(0)} = A)$, де $k = 1,2,\ldots,n-1$,

1) знаходимо j_k – номер рядка, в якому знаходиться *провідний елемент*
$\left| a_{j_k,k}^{(k-1)} \right| = \max\limits_{k \le i \le n} \left(\left| a_{i,k}^{(k-1)} \right| \right) \& \ a_{j_k,k}^{(k-1)} \neq 0;$

2) якщо $j_k \neq k$, то виконуємо перестановку рядків j_k, k множенням на відповідну елементарну матрицю перестановок
$P_k = P_{k,j_k}, \quad P_k \cdot A^{(k-1)} = P_k \cdot \vec{b}^{(k-1)};$

3) виключаємо x_k з рядків $i = \overline{k+1,n}$ множенням на відповідну елементарну трикутну матрицю
$L_k \cdot P_k \cdot A^{(k-1)} = L_k \cdot P_k \cdot \vec{b}^{(k-1)};$
де

$$L_k = \begin{pmatrix} 1 & 0 & \cdots & 0 & \cdots & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots & \cdots & \cdots & \vdots \\ 0 & \cdots & l_{k,k} & 0 & \cdots & \cdots & 0 \\ 0 & \cdots & l_{k+1,k} & 1 & \ddots & \cdots & 0 \\ 0 & \cdots & l_{k+1,k} & 0 & 1 & \ddots & 0 \\ 0 & \cdots & \vdots & \vdots & \cdots & \ddots & \vdots \\ 0 & \ldots & l_{n,k} & 0 & \cdots & \cdots & 1 \end{pmatrix}, \quad l_{i,k} = \begin{cases} \dfrac{1}{a_{k,k}^{(k-1)}}, i = k, \\ \dfrac{-a_{i,k}^{(k-1))}}{a_{k,k}^{(k-1)}}, i = \overline{k+1, n}. \end{cases}$$

У результаті об'єднання проведених обчислень (прямий хід методу Гаусса), отримуємо (3.4), де

$$U = L_{n-1} \cdot P_{n-1}(\cdot \ldots (L_2 \cdot P_2 (L_1 \cdot P_1 \cdot A)) \ldots),$$

$$\vec{f} = \vec{b}^{(n-1)} \equiv P_{n-1} \cdot \ldots \cdot P_2 \cdot P_1 \cdot \vec{b}.$$

Знаходження компонент x_k (зворотний хід методу Гаусса) здійснюється за формулою: $x_k = f_k - \sum_{i=k+1}^{n} u_{k,i} x_i, \quad k = n, n-1, \ldots, 1.$

Б) Алгоритми з явним знаходження матриць *P*, *L*, *U* – *методи факторизації.*

1) Знаходимо розклад матриці $A \Rightarrow [P, L, U]$;
2) отримуємо дві СЛАР з трикутними матрицями

$$(L \cdot U) \cdot \vec{x} = \vec{f}, \quad \vec{f} = P \cdot \vec{b}, \quad \begin{cases} L \cdot \vec{y} = \vec{f}, \\ U \cdot \vec{x} = \vec{y}. \end{cases} \tag{3.5}$$

Як методи факторизації, так і метод Гаусса, мають певні варіанти реалізації, пов'язані зі специфікою матриці СЛАР :

a) якщо в (3.5) $P = E$, то такий варіант має назву *методу Холецького*;
b) якщо матриця A є ермітовою $A = A^*$ або симетричною $A = A'$, то такий варіант має назву *методу квадратного кореня*;
c) якщо A – *тридіагональна* матриця

$$\begin{cases} a_i x_{i-1} - c_i x_i + b_i x_{i+1} = -f_i, \\ i = \overline{1, n}, \quad a_1 = b_n = 0, \end{cases} \tag{3.6}$$

для якої виконуються умови

$$\begin{cases} |c_i| \geq |a_i| + |b_i|, \quad i = \overline{1, n}; \\ a_i \neq 0, \quad i = \overline{2, n}; \\ b_i \neq 0, \quad i = \overline{1, n-1}, \end{cases} \tag{3.7}$$

то такий варіант методу Гаусса має назву *методу монотонної правої прогонки*, і реалізація прямого ходу відбувається за рекурентними формулами:

$$\alpha_1 = \beta_1 = 0; \ z = c_i - a_i \alpha_i, \ \alpha_{i+1} = b_i / z, \ \beta_{i+1} = (f_i + a_i \beta_i) / z, \ i = \overline{1, n}, \tag{3.8}$$

а зворотного ходу – за рекурентними формулами:

$$x_n = \beta_{n+1}; \ x_{i-1} = \alpha_i x_i + \beta_i, \ i = n, n-1, \ldots, 2. \tag{3.9}$$

При виконанні умов (3.7) прогонка (3.8),(3.9) є *стійкою* (виконується умова $|\alpha_i| \leq 1, \ i = \overline{2, n}$).

Для випадку (3.6) є варіант *немонотонної прогонки* (пошук оптимального елемента по рядку), аналогічно існують ці варіанти для п'ятидіагональної матриці. Розглядаються також реалізації прогонки, які носять назви: *потокова*, *зустрічна*, *циклічна*, *для складних систем*, *матрична* [15, 18, 19].

QR-розклад матриці *A* для розв'язання СЛАР *Ax=b*

Лема. 3.1. Довільна дійсна квадратна матриця може бути представлена у вигляді добутку *ортогональної* і *правої трикутної* матриці:

$$A = Q \cdot R \equiv U' \cdot R, \quad U = U_{n-1} \cdot \ldots \cdot U_2 \cdot U_1; \qquad (U_i' \cdot U_i = U_i \cdot U_i' = E). \tag{3.10}$$

Метод послідовної побудови матриці U – *метод відбиття*. Розглянемо алгоритм цього методу (спрощений варіант, при умові, що всі кутові мінори не дорівнюють нулю). Маємо реалізацію :

1-й крок. Матриця $A = A_{n,n} = A^{(0)}$. Вибираємо вектор-стовпець $\vec{y} = (a_{1,1}, a_{2,1}, ..., a_{n,1})^{\mathrm{T}}$.

а) якщо $a_{2,1} = a_{3,1} = ... = a_{n,1} = 0$, то $A^{(1)} = A^{(0)}, \quad U_1 = E$;

б) якщо $a_{2,1}, a_{3,1}, ..., a_{n,1} \neq 0$, то $A^{(1)} = U_1 \cdot A^{(0)}, \quad U_1 = E - 2\vec{\omega} \cdot \vec{\omega}^{\mathrm{T}}$, де

$$\vec{\omega}^{\mathrm{T}} \cdot \vec{\omega} = (\vec{\omega}, \vec{\omega}) = 1, \quad \vec{\omega} = \frac{\vec{y} - \alpha \vec{e}_1}{\|\vec{y} - \alpha \vec{e}_1\|}, \quad \alpha = \|\vec{y}\| = \sqrt{(\vec{y}, \vec{y})}\,; \tag{3.11}$$

в) виконали *l*-1 -й крок ;

г) *l*-й крок. Для матриці розмірності *n*-*l*+1 $\begin{pmatrix} a_{l,l}^{(l-1)} & \ldots & a_{l,n}^{(l-1)} \\ \vdots & \ddots & \vdots \\ a_{n,l}^{(l-1)} & \ldots & a_{n,n}^{(l-1)} \end{pmatrix}$ вибираємо вектор-стовпець $\vec{y} = \left(a_{l,l}^{(l-1)}, a_{l+1,l}^{(l-1)}, ..., a_{n,l}^{(l-1)}\right)^{\mathrm{T}}$ і виконуємо дії, аналогічні 1-му кроку:

а) якщо $a_{l,l}^{(l-1)} = a_{l+1,l}^{(l-1)} = ... = a_{n,l}^{(l-1)} = 0$, то $A^{(l)} = A^{(l-1)}, \quad U_l = E$;

б) якщо $a_{l,l}^{(l-1)}, a_{l+1,l}^{(l-1)}, ..., a_{n,l}^{(l-1)} \neq 0$, то $A^{(l)} = U_l \cdot A^{(l-1)}$, а блочна матриця U_l визначається наступним чином:

$$U_l = \begin{pmatrix} E_{l-1} & 0 \\ 0 & U_{(n-l+1)} \end{pmatrix}, \quad U_{(n-l+1)} = E_{(n-l+1)} - 2\vec{\omega}_{(n-l+1)} \cdot \vec{\omega}^{\mathrm{T}}{}_{(n-l+1)},$$

де для обчислення вектора $\vec{\omega}_{(n-l+1)}$ використовуємо формули (3.11), але з урахуванням розмірності *n*-*l*+1.

Після виконання *n*-1 кроків отримаємо верхню трикутну матрицю R :

$$R = A^{(n-1)} = ... = U_{n-1} \cdot U_{n-2} \cdot .... \cdot U_1 \cdot A = U \cdot A, \quad U^{\mathrm{T}} \cdot R = A\,.$$

QR–розклад використовується як для розв'язання СЛАР, так і для побудови ітераційних процесів вирішення повної проблеми власних значень.

Оцінка похибки розв'язування СЛАР. Число *cond(A)*

Припустимо, що похибки, які вносяться у вхідні дані, малі.

Означення 1. Під *обумовленістю* обчислювальної задачі розуміють чуттєвість (залежність) її розв'язку до вхідних даних, а коефіцієнт, який зв'язує ці похибки, називають *мірою обумовленості задачі*.

Визначимо його для СЛАР :

$$A\vec{x} = \vec{b}, \quad \det(A) \neq 0\,. \tag{3.12}$$

Для (3.12) можна отримати

$$\vec{x} = A^{-1} \cdot \vec{b}\,. \tag{3.13}$$

Вважаючи A^{-1}, $\vec{b}$ змінними, формально продиференцюємо (3.13) :

$$d\vec{x} = dA^{-1}\cdot\vec{b} + A^{-1}\cdot d\vec{b} = \left[A\cdot A^{-1} = E,\ dA\cdot A^{-1} + A\cdot dA^{-1} = 0,\ dA^{-1} = -A^{-1}\cdot dA\cdot A^{-1}\right] =$$
$$= -A^{-1}\cdot\left(dA\cdot\left(A^{-1}\cdot\vec{b}\right) - d\vec{b}\right) = -A^{-1}\cdot\left(dA\cdot\vec{x} - d\vec{b}\right). \qquad (3.14)$$

В останній рівності перейдемо до норм, використаємо відповідні нерівності, в результаті чого отримаємо:

$$\left\|d\vec{x}\right\| \le \left\|A^{-1}\right\|\cdot\left(\left\|dA\right\|\cdot\left\|\vec{x}\right\| + \left\|d\vec{b}\right\|\right). \qquad (3.15)$$

Враховуючи співвідношення

$$\vec{b} = A\cdot\vec{x},\quad \left\|\vec{b}\right\| \le \left\|A\right\|\cdot\left\|\vec{x}\right\|,\quad \left\|\vec{x}\right\| \ge \frac{\left\|\vec{b}\right\|}{\left\|A\right\|},$$

і вводячи відносні похибки вхідних даних і розв'язку

$$\varepsilon_A = \frac{\left\|dA\right\|}{\left\|A\right\|},\quad \varepsilon_b = \frac{\left\|d\vec{b}\right\|}{\left\|\vec{b}\right\|},\quad \varepsilon_x = \frac{\left\|d\vec{x}\right\|}{\left\|\vec{x}\right\|},$$

отримаємо:

$$\frac{\left\|d\vec{x}\right\|}{\left\|\vec{x}\right\|} \le \left\|A^{-1}\right\|\cdot\left(\left\|dA\right\| + \frac{\left\|d\vec{b}\right\|}{\left\|\vec{x}\right\|}\right) \le \left\|A^{-1}\right\|\cdot\left(\left\|dA\right\| + \frac{\left\|d\vec{b}\right\|}{\left\|\vec{b}\right\|}\cdot\left\|A\right\|\right) \le \left\|A^{-1}\right\|\cdot\left\|A\right\|\cdot\left(\frac{\left\|dA\right\|}{\left\|A\right\|} + \frac{\left\|d\vec{b}\right\|}{\left\|\vec{b}\right\|}\right),$$

$$\varepsilon_x \le cond(A)\cdot\left(\varepsilon_A + \varepsilon_b\right), \qquad (3.16)$$

де $\mu_A = cond(A) = \left\|A\right\|\cdot\left\|A^{-1}\right\|$ – *число обумовленості* матриці A.

Зауваження 1. Матриці з $\mu_A >> 1$ – *погано обумовлені.*

Зауваження 2. Детальніші дослідження дають наступний вираз для (3.16):

$$\varepsilon_x \le \frac{\mu_A}{1 - \mu_A\cdot\varepsilon_A}\cdot\left(\varepsilon_A + \varepsilon_b\right).$$

Зауваження 3. Для μ_A маємо наступні співвідношення:

1^o $\mu_A \ge 1;$

2^o $\mu_A \ge \dfrac{\max\limits_i\left|\lambda_i(A)\right|}{\min\limits_i\left|\lambda_i(A)\right|};$

3^o $\mu(A\cdot B) \le \mu(A)\cdot\mu(B).$

Зауваження 4. Простою ознакою поганої обумовленості матриці є велика різниця норм вектор-стовпців / вектор-рядків. Загальне правило дій у цьому випадку таке: за допомогою множення стовпців / рядків на числа (степені 2) урівноважуємо ці норми, масштабуємо невідомі / перетворюємо праві частини. Після цього застосовуємо методи типу Гаусса. Якщо цього не достатньо, потрібно використовувати *регуляризацію*.

Алгебраїчна проблема власних значень

Означення 2. Нехай для матриці $A = \left(a_{i,j}\right)_{i,j=1}^{n}$ існує нетривіальний розв'язок однорідної системи :

$$A\cdot\vec{h} = \lambda\cdot\vec{h}, \qquad (3.17)$$

тоді числа λ_i називаються *власними числами* матриці A, а відповідні їм розв'язки $\vec{h}_i$ – *власними векторами*.

Означення 3. Задачу відшукання повного набору $\{\lambda_i(A),\ \vec{h}_i\}$ називають *повною проблемою власних значень* (ППВЗ) для матриці А. Відповідно, визначення частини спектру $\lambda_i(A)$ (найчастіше максимального / мінімального за модулем) і можливо відповідних власних функцій $\vec{h}_i$ називають *частковою проблемою власних значень.*

Часткова проблема власних значень. Розглянемо *степеневий* ітераційний метод/процес (СІП), який базується на відношенні Релея і для застосування якого необхідно, щоб матриця *A* мала *n* лінійно незалежних векторів.

Алгоритм. 1. Задаємо довільний вектор, відмінний від нуля ($\forall\ \vec{x}^{(0)} \neq 0$).

2. В циклі по k=0,1,…, $kmax$ повторюємо наступні дії :

а) $\vec{Y} = A \cdot \vec{x}^{(k)}, \quad y = \sqrt{(\vec{Y}, \vec{Y})}, \quad \lambda^{(k)} = y, \quad \vec{x}^{(k+1)} = \vec{Y}/y$;

б) після $k>m$ перевіряємо виконання умови закінчення СІП $\left|\lambda^{(k)} - \lambda^{(k-1)}\right| < \varepsilon$.

Тут позначено:

kmax – максимальна кількість ітерацій (для запобігання зациклюванню СІП),

m – кількість ітерацій без перевірки різниці наближень до максимального за модулем власного числа (0<*m*<*kmax*), $0 < \varepsilon < 1$ – бажана точність.

Результатом СІП є власне значення $\max\limits_i \left|\lambda_i(A)\right| = \lambda^{(k)}$ і відповідний йому власний вектор $\vec{h} = \vec{x}^{(k+1)}$. Збіжність СІП лінійна зі знаменником $q = |\lambda_2/\lambda_1|$. Збіжність СІП відсутня при:

а) $\lambda_1 \neq \lambda_2$ & $|\lambda_1| = |\lambda_2|$;

б) для власного значення λ_1 жорданова клітина має розмірність більше 1.

Для варіанту $A = A'$ СІП має назву методу *скалярних добутків* і в наведеному вище алгоритмі у пункті 2 змінюється дія а) на наступну:

$$\vec{Y} = \frac{\vec{x}^{(k)}}{\left\|\vec{x}^{(k)}\right\|}, \quad \vec{x}^{(k+1)} = A \cdot \vec{Y}, \quad \lambda^{(k)} = \left(\vec{Y}, \vec{x}^{(k)}\right),$$

що забезпечує квадратичну збіжність.

Для знаходження $\min\limits_i \left|\lambda_i(A)\right|$ використовується варіант СІП з оберненою матрицею $A \cdot \vec{Y} = \vec{x}^{(k)}, \quad y = \sqrt{(\vec{Y}, \vec{Y})}, \quad \beta^{(k)} = y, \quad \vec{x}^{(k+1)} = \vec{Y}/y$ і $\min\limits_i \left|\lambda_i(A)\right| = \dfrac{1}{\beta^{(k)}}$.

Повна проблема власних значень. Існують досконалі алгоритми і програми, що базуються на різних модифікаціях *QR-алгоритмів* за схемою

$$A^{(0)} = A, \quad A^{(k)} = Q^{(k+1)} \cdot R^{(k+1)}, \quad A^{(k+1)} = R^{(k+1)} \cdot Q^{(k+1)}, \quad \lim_{i \to \infty}\left(A^{(i)}\right) = \Lambda,$$

де Λ – *права канонічна форма Жордана*. Для варіанту $A = A'$ або $A = A^*$ алгоритм має назву *методу обертань* (*Якобі*).

ТЕМА 4
РОБОТА З МАТРИЦЯМИ, МАТРИЧНИЙ АНАЛІЗ

Усі дані MATLAB представляються у вигляді масивів. Дуже важливо розуміти, як їх використовувати. Без цього неможлива ефективна робота в MATLAB, зокрема побудова графіків, розв'язання задач лінійної алгебри, обробка даних тощо. У підрозділі описані основні операції над масивами.

Створення та використання масивів у системі MATLAB

Для створення масиву (стандартно двовимірної дійсної матриці) можна використати:

1. *конструктор масиву* `[]`, в якому записуються необхідні елементи. В загальному випадку це вирази зі скалярними даними або масивами, але можуть бути і порожні елементи. Елементи рядка відокремлюються символом "*пропуск*" або "`,`", елементи стовпця – символом "`;`";
2. *операцію* "`:`" для формування *діапазону* значень `e1:e2:e3` – послідовності елементів у вигляді арифметичної прогресії з початковим елементом `e1`, кроком `e2` і кінцевим елементом `e3`. При використанні дійсних параметрів в означенні послідовності слід мати на увазі, що значення `e3` визначається з урахуванням наближених обчислень;
3. *вбудовані функції:*
   - `linspace(e1,e3,n)` – для формування вектор-рядка з послідовності `n` елементів у вигляді арифметичної прогресії з початковим елементом `e1` і кінцевим елементом `e3` (який належить послідовності), а рівномірний крок визначається за формулою `(e3-e1)/(n-1);`
   - `logspace(e1,e3,n)` – для формування вектор-рядка з послідовності `n` елементів у вигляді геометричної прогресії з початковим елементом 10^{e1} і кінцевим елементом 10^{e3}.

     Якщо параметр `n` опущено, то вважається, що `n=100`;
4. *вбудовані функції:*
   - `eye(n,m,тип)` – для створення одиничної матриці ($m=n$) або матриці з одиницями на головній діагоналі ($m\neq n$);
   - `zeros(n,m,тип)` – для створення матриці *з нулів*;
   - `ones(n,m,тип)` – для створення матриці *з одиниць*,

   де `n,m` – кількість рядків, стовпців. При створенні квадратної матриці параметр `m` можна не вказувати. Параметр `тип` – необов'язковий. Зазвичай його використовують для означення типу вихідних даних і вказують у вигляді константи-рядка. Якщо його не вказано, то за замовчуванням вважаємо, що маємо тип `'double'`.
5. *вбудовані функції* для генерації спеціальних матриць: `compan, hadamard, hankel, hilb, invhilb, magic, pascal, rosser, toeplitz, wilkinson;`
6. *вбудовану функцію* `rand(n,m)` для генерації матриці розмірності `n×m` із псевдовипадкових чисел з діапазону від 0 до 1;

7. *вбудовану функцію* `meshgrid` для генерації матриць `X,Y` координатної сітки на базі одновимірних векторів `x,y` (використовуються при побудові 3D-графіків). Після виклику `[X,Y]=meshgrid(x,y)` отримуємо матриці наступного виду:

$$\mathrm{X}=\left[\underbrace{x;x;...;x}_{m}\right],x=\left[x_1,x_2,...,x_n\right];\quad \mathrm{Y}=\left[\overbrace{y,y,...,y}^{n}\right],y=\left[y_1;y_2;...;y_m\right].$$

Приклад 5. Використання різних варіантів створення масивів.

```
%%pl4_1.m
% Приклади створення масивів
clear all, close all, clc
b = [1; -1; 3], A = [11, 3, 2; -1 10 -1; 2, 2 7]
B = [], B = [diag(b), eye(3); diag(diag(A)), A]
y = 0 : 0.501 : 1.1
z = 0 : 1 : 5
x = linspace(0, 1.1, 3), y = linspace(-1, 1.3, 5), z = linspace(-2, 2)
p = logspace(0, 4, 5)
A = eye(2),  A = eye(2, 1),  A = eye(3, 2, 'int8')
A = zeros(2),  A = zeros(2, 3), A = zeros(3, 2, 'int8')
A = ones(2), A = ones(2, 3), A = ones(3, 2, 'uint8')
C = compan([1 2 3 4]), H = hadamard(4), A = hankel(C)
H = hilb(5), I = invhilb(5)
M = magic(5), P = pascal(5)
R = rosser('int8'), T = toeplitz(C), W = wilkinson(5)
A = rand(3,4)
[X, Y] = meshgrid(x, y); X, Y
```

Для створення масивів розмірності більше 2 можна використовувати:

1. функції `zeros, ones, rand` з відповідним набором параметрів. Наприклад, `A=rand(3,2,4)`;
2. процедуру "нарощування" існуючого масиву відповідними елементами. Наприклад, для матриці `A=[1,2;3,4]` після виконання команди `A(1,1,2)=0` отримаємо матрицю `A` розмірності 2×2×2 (причому матриця `A(:,:,2)` – нульова), а після виконання команди `A(:,:,3)=[0,20;70,2]` – матрицю `A` розмірності 2×2×3. Допустима і зворотна дія, наприклад, після команди `A(:,:,2)=[]` ми видалили нульову матрицю і отримали матрицю `A` розмірності 2×2×2;
3. функцію `cat`. Наприклад, після виконання команд `A=magic(3); B=pascal(3); C=cat(4,A,B)` отримаємо матрицю `C` розмірності 3×3×1×2.

Відлік (нумерація) елементів масиву за розмірностями починається з 1.

Для використання масиву із заданим іменем:

1. вказуємо ім'я масиву `A` в матричних операціях при побудові виразів і при отриманні матриці-результату, наприклад, `A=[-pi:pi/20:pi]; Y=sin(A)`;
2. вказуємо *A(індексний_вираз)* – для *доступу* до елемента(ів) матриці `A` при використанні в побудованому виразі або для *зміни значення* елемента (значень елементів) `A` при використанні в лівій частині команди присвоєння. Слід пам'ятати, що відсутність елемента у випадку *доступу*, призводить до виникнення фатальної помилки, а у випадку *зміни значення* – до добудови ("нарощування") масиву `A`.

Уведене тут позначення *індексний_вираз* має наступні смислові варіанти:

- *один* скалярний вираз/змінна/константа зі значенням `k` – для вказівки фіксованого (конкретного) `k`-го елементу одновимірного масиву `A`, наприклад, `A(1)`, `A(k)`, `A(k+1)`;
- *декілька* скалярних виразів/змінних/констант, які розділені символом "," (*список індексів*) зі значеннями `k1,k2,…,km` – для вказівки фіксованого (конкретного) елементу `m`-вимірного масиву `A`, наприклад, `A(1,2)`, `A(k,j)`, `A(k+1,j-1,m)`;
- у *списку індексів* використовується *знак операції* "`:`" – для вказівки усіх елементів цього рядка/стовпця/матриці/… масиву `A` (перші два варіанти для двовимірного, а наступні – для багатовимірних). Наприклад, якщо `A` матриця розмірності 10×20, то `A(:,2)` – одновимірний вектор-стовпець із елементів другого стовпця матриці `A`, а `A(5,:)` – одновимірний вектор-рядок із елементів п'ятого рядка матриці `A`;
- *end* (спеціальний символ) – для вказівки останнього елементу в даній вимірності масиву `A`. Може комбінуватись зі знаком операції "`:`". Наприклад, для одновимірного масиву: `A(end)`, `A(end-1)`, `A(k:end)`; для двовимірного масиву: `A(1,end)`, `A(:,end-2)`, `A(end,k)`, `A(end,end)`, `A(end)`;
- *один знак операції* "`:`" – для вказівки всіх елементів масиву `A` (якщо `A` двовимірний, то перебір виконується по стовпцях, а у випадку тривимірності, спочатку по стовпцях матриці `A(:,:,1)`, потім по стовпцях матриці `A(:,:,2)` і т. д.). Наприклад, порівняйте для матриці `A=[1,2;3,4]` вирази `A(:)` і `A(:,:)`;
- *масив зі значеннями цілого типу (індексний масив)* – для вказівки тих елементів масиву `A`, індекси яких дорівнюють значенню елементів індексного масиву. Наприклад, для матриці `A=rand(5,6)` команди `A([3:8])=0; A(2:3,4:6)=0;` обнуляють наступні елементи: `A(3,1)`, `A(4,1)`, `A(5,1)`, `A(1,2)`, `A(2,2)`, `A(3,2)`, `A(2,4)`, `A(2,5)`, `A(2,6)`, `A(3,4)`, `A(3,5)`, `A(3,6)`;
- *масив зі значеннями логічного типу (масив логічного індексування)* – для вказівки всіх елементів масиву `A`, індекси яких відповідають індексам масиву логічного індексування (МЛІ) і відповідний елемент МЛІ є *Істина*. Приклади: для отриманої в попередньому прикладі матриці, команда `b=A(0<A&A<0.5)` утворить вектор-стовпець `b` із елементів матриці `A`, для яких виконується умова `0<A&A<0.5`, а після виконання команди `B=A(A(:,1)>0, A(5,:)>0)` отримаємо матрицю `B` розмірності 2×5 (проаналізуйте утворення її елементів!).

Вбудовані функції MATLAB для роботи з матрицями

Розглянемо вбудовані функції системи MATLAB, які використовуються для знаходження числових характеристик матриць, виконання операцій над матрицями і векторами, розв'язування основних задач лінійної алгебри, представлення матриць в певних формах і вирішення питань проблеми власних значень. Імена більшості з цих функцій співпадають з математичними позначеннями, тому легко запам'ятовуються.

Визначник квадратної матриці знаходиться функцією `det(A)`.

Норма вектора /матриці знаходиться функцією `norm(A,p)`, де необов'язковий параметр `p` може приймати такі допустимі значення: `1,2,inf,'fro'`, де `'fro'` – фробеніусова норма. Якщо `p` опускаємо, то `p=2`. Нижче представлено таблицю

обчислення узгоджених норм для векторів/матриць через інші вбудовані функції MATLAB, що дає чітке уявлення про визначення цієї вбудованої функції

Для матриць:	**Для векторів:**
`norm(A) = norm(A,2) = max(svd(A))`	`norm(A) = norm(A,2) = sqrt(sum(A.^2))`
`norm(A,1) = max(sum(abs(A)))`	`norm(A,1) = sum(abs(A))`
`norm(A,inf) = max(sum(abs(A')))`	`norm(A,inf) = max(abs(A))`
`norm(A,'fro') = sqrt(sum(diag(X'*X)))`	`norm(A,-inf) = min(abs(A))`

Число обумовленості квадратної матриці знаходиться функцією `cond(A,p)`, де необов'язковий параметр `p` приймає такі самі значення, як і у функції `norm`. Для оцінки числа обумовленості використовується функція `rcond(A)`, яка обчислює значення `k1`, обернене значенню `cond(A,1)`. Якщо матриця `A` добре обумовлена, то значення `k1` близьке до одиниці, для погано обумовленої матриці значення `k1` близьке до нуля.

Ранг матриці знаходиться функцією `rank(A)`.

Слід квадратної матриці знаходиться функцією `trace(A)`.

Ортонормований *базис* матриці знаходиться функцією `orth(A)`. Виклик `H=orth(A)` дає матрицю `H`, для якої (математично) виконується рівність `H' *H = E`. Тут і надалі термін "математично" означає, що всі операції виконуються точно і похибки заокруглень відсутні. Тому, якщо необхідно виконати перевірку деякої математичної рівності `Л=П`, то записуємо її, наприклад, у вигляді, `abs(Л-П)<1e-P`, де `P` число з діапазону ≈[12,16].

Нуль-простір (ядро) для матриці – знаходиться функцією `null(A)`. Виклик `Z=null(A)` утворює матрицю `Z`, для якої (математично) виконуються рівності `Z' *Z=E`, `A*Z=0`.

Скалярний добуток векторів знаходиться функцією `dot(X,Y)`, де `X,Y` – вектори однакової розмірності.

Векторний добуток векторів знаходиться функцією `cross(X,Y)`, де `X,Y` – вектори розмірності 3.

Тензорний добуток векторів (добуток Кронекера) знаходиться функцією `kron(X,Y)`, де `X,Y` – прямокутні матриці відповідної розмірності.

Обернена матриця для квадратної матриці знаходиться функцією `inv(A)`. Виклик `B=inv(A)` дає матрицю `B`, для якої (математично) виконується рівність `B*A = A*B = E`.

Псевдообернена матриця (по Муру-Пенроузу) – знаходиться функцією `pinv(A)`. Виклик `B=pinv(A)` для квадратної (прямокутної) матриці `A` дає матрицю `B`, для якої (математично) виконуються рівності `A*B*A = A`, `B*A*B = B`. Тут `A*B`, `B*A` – ермітові матриці.

LU–розклад матриці `A` виконується за допомогою виклику `[L,U,P]=lu(A)` для квадратної матриці `A`. Тут `L` – нижня трикутна матриця з одиницями на головній діагоналі, `U` – верхня трикутна матриця, `P` – матриця перестановок. Для наявних матриць (математично) виконується рівність `P*A = L*U`.

QR–розклад матриці `A` виконується за допомогою виклику `[Q,R,P]=qr(A)` для квадратної матриці `A`. Тут `Q` – унітарна матриця (`Q*Q'=E`), `R` – верхня трикутна матриця, `P` – матриця перестановок. Для наявних матриць (математично) виконується рівність `A*P=Q*R`.

Розклад *Холецького* Якщо матриця `A` *симетрична* (елементи дійсні) або *ермітова* (елементи комплексні), то можна застосувати розклад Холецького за допомогою виклику функції `[R,p]=chol(A)`. Якщо `A` – додатно означена матриця, то `p=0`, а `R` – верхня трикутна і виконується рівність `R'*R=A`, в іншому випадку `p>0` і `R` – верхня трикутна матриця порядку `q=p-1` така, що `R'*R=A(1:q,1:q)`.

Сингулярний розклад матриці `A` (SVD – Singular value decomposition) виконується за допомогою виклику `[U,D,V]=svd(A)` для квадратної матриці `A`. Тут `D=diag(d)` – діагональна матриця, де `d` – вектор невід'ємних чисел ($d_1 \ge d_2 \ge \ldots \ge d_n$).

З'ясуємо математичний зміст функції `svd`, яка використовується в інших вбудованих функціях цієї групи функцій. Сингулярний розклад матриці `A` означає, що потрібно знайти ненульові вектори $\vec{v}$, $\vec{u}$ і невід'ємні числа `d`, для яких виконується наступна системи рівнянь:

$$\begin{cases} A\,\vec{v} = d\,\vec{u}, \\ A'\,\vec{u} = d\,\vec{v}. \end{cases} \tag{4.1}$$

Вектори $\vec{v}$, $\vec{u}$ називаються *правими і лівими сингулярними* векторами, числа `d` – *сингулярними* числами.

Допустимо, що вектори $\vec{v}$ утворюють стовпці унітарної матриці `V`, а $\vec{u}$ утворюють стовпці унітарної матриці `U`, числа `d` – діагональну матрицю `D`. Тоді рівняння (4.1) еквівалентні двом матричним рівнянням:

$$\begin{cases} A*V = U*D, \\ A'*U = V*D \end{cases}$$

або одному матричному розкладу:

$$A = U*D*V', \tag{4.2}$$

який і називається *сингулярним* розкладом матриці `A`.

Це представлення для квадратної матриці можна знайти і з інших міркувань.

Теорема 4.1. (Полярний розклад Келі) Довільну квадратну матрицю `A` можна представити у вигляді добутку симетричної `S` і ортогональної матриці `W`:

$$A = S*W;\ \ S = S',\ \ \ W*W' = E. \tag{4.3}$$

У свою чергу симетрична матриця `S` може бути приведена до діагонального виду за допомогою деякого ортогонального перетворення:

$$S = U*C*U';\ \ C = diag(c_i), c_i = \pm d_i, d_i \ge 0;\ \ C = D*T;\ \ T = diag(t_i), t_i = \pm 1,\ T*T' = E. \tag{4.4}$$

Підставимо (4.4) у (4.3) і отримаємо :

$$A = U*D*T*U'*W\ = U*D*V',\ \ V' = T*U'*W\ .$$

Приведення матриці до (*верхньої*) *форми Хессенберга* виконується функцією `[P,H]=hess(A)`. Якщо матриця `A` симетрична або ермітова, то матриця Хессенберга `H` буде тридіагональною. Необов'язковий параметр `P` – унітарна матриця перетворень. Для наявних матриць (математично) виконується рівність `A=P*H*P, P'*P=E`.

Жорданова форма матриці знаходиться функцією `jordan(A)`.

Приведення матриці до *ступінчастого* вигляду виконується функцією `[R,rb]=rref(A)`. Застосовується метод Гаусса з вибором провідного елемента по стовпцю (у випадку невиродженої квадратної матриці отримуємо матрицю `E`). Наприклад, послідовність команд

```
A=[1,1,1;1,2,2;1,2,3];  b=[1;2;3];  R=rref([A,b]);  R(:,4)
```

знаходить розв'язок наступної СЛАР

$$\begin{cases} 1x_1 + 1x_2 + 1x_3 = 1, \\ 1x_1 + 2x_2 + 2x_3 = 2, \\ 1x_1 + 2x_2 + 3x_3 = 3. \end{cases}$$

Власні значення і *вектори* квадратної матриці знаходяться функцією `eig(A)`. Виклик `[H,L]=eig(A)` утворює матрицю `H` власних векторів і діагональну матрицю `L` власних чисел, для яких (математично) виконується рівність `A*H = H*L`. Для матриць `H,L` виконуються співвідношення: `n=length(A); norm(H(:,k))=1, k=1,…,n; abs(L(1,1))≥…≥abs(L(n,n))`.

Характеристичний поліном квадратної матриці знаходиться функцією `poly(A)`. Виклик `P=poly(A)` дає вектор-рядок (стандартне зображення полінома в MATLAB), який і представляє характеристичний поліном виду

$$P = p_1\lambda^n + p_2\lambda^{n-1} + ... + p_n\lambda + p_{n+1}.$$

Наступні М-файли ілюструють деякі варіанти роботи з матрицями.

Приклад 6. Реалізувати *QR*–розклад матриці.

```
%%pl4_2.m
% Якщо для матриці A всі головні мінори <>0,
% то методом відбиття можна отримати розклад Q*R=A,
% де Q - ортогональна (QQ'=E), а R - верхня трикутна матриці
% err - код закінчення: при err=0 - розклад знайдено
clear all, close all, clc
A = rand(4);
n = length(A); R = A; Q = eye(n); err = 0;
for k = 1 : n-1
   if R(k,k) == 0
      err = 1; break
   end
   y = R(k:n, k); km = k-1; m = n-km;
   if any(y(2:m))
      al = norm(y); y(1) = y(1)-al;
      y = y./norm(y); U = eye(m)-2.*y*y';
      U = [eye(km), zeros(km,m); zeros(m,km), U];
      Q = U*Q; R = U*R; R(k+1:n,k) = 0;
   end
end
if ~err
   Q = Q', R,  abs(Q*R - A) < 1e-14  % перевірка
   [Q, R] = qr(A)  % використання вбудованої функції
end
```

Приклад 7. Різні варіанти розв’язання СЛАР.

```
%%pl4_3.m
% Приклади розв'язання СЛАР Ax=b
clear all, close all, clc
xx = [1; -2; 3]; % відомий розв'язок (модельна задача)
% формування вхідних данних: матриця A=A' & A>0
A=[ 7,  1, 1;...
    1, 10, 2;...
    1,  2, 4]
b = A * xx    %  вектор правої частини СЛАР

% Варіанти
x1 = A \ b;                % 1-й : матричне ліве ділення
```

```
[Q, R, P] = qr(A);
x2 = P * (R \ (Q' * b));  % 2-й : QR-розклад
[L, U, P] = lu(A);
x3 = U \ (L \ (P * b));   % 3-й : LU-розклад
R = rref([A, b]);
x4 = R(:, 4);             % 4-й : приведення до ступінчастої форми
x5 = [NaN; NaN; NaN];
if all(all(A == A'))      % 5-й : розклад Холецького
   [R, p] = chol(A);
   if p == 0
      x5 = R \ (R' \ b);
   end
end
opts.SYM = true; opts.POSDEF = true;
x6 = linsolve(A, b, opts);% 6-й : використання вирішувача – солвера
                          %       для СЛАР
x7 = inv(A) * b;          % 7-й : використання оберненої матрицю
% візуальне порівняння отриманих розв'язків :
[x1, x2, x3, x4, x5, x6, x7, xx]
```

Приклад 8. Знаходження власних значень і власних векторів матриць.

```
%%pl4_4.m
clear all, close all, clc
P = pascal(5) % побудова матриці Паскаля
m = length(P) - 1;
p = poly(P)   % характеристичний поліном матриці P
% використання конструктора [], операцій
% горизонтального і вертикального об'єднання масивів
% для побудови запису матриці P у формі Фробеніуса
F = [-p(2:end); [eye(m), zeros(m,1)]]
r = roots(p)  % власні числа матриці Паскаля
J = jordan(P) % жорданова форма для матриці Паскаля
% власні вектори і власні числа матриці Паскаля
[H1, L1] = eig(P);  %  для P
H1, L1        %  вивести на екран
l1 = sort(diag(L1), 'descend');  % по спаданню значень
[H2, L2] = eig(F);  %  для P у формі Фробеніуса
H2, L2        %  вивести на екран
l2 = sort(diag(L2), 'descend');
[r, l1, l2]   %  візуальне порівняння
l1 == l2      %  порівняння логічне
% порівняння з урахуванням наближених обчислень
abs(l1 - l2) < 1e-14
```

Враховуючи певну специфіку використання *функцій від матриць* (не слід плутати із застосуванням функцій до кожного елемента матриці), наведемо тільки їх виклик або назву:

- матрична експонента (e^A) (`expm(A), expm1, expm2, expm3`);
- матричний логарифм (`logm(A)`);
- матричний квадратний корінь ($A^{1/2}$) (`sqrtm(A)`);
- матричний поліном (`polyvalm(p,A)`);
- обчислення довільної матричної функції `f` (`funm(A,@f)`).

Поліноми та дії над ними

У MATLAB поліном задається і зберігається у вигляді вектора, елементами якого є коефіцієнти полінома. Перелік вбудованих функцій для роботи з поліномами наведено нижче:

Функції для роботи з поліномами	
Назва	**Призначення**
polyval(*P,x*)	обчислення значення полінома *n*-го степеня при заданому значенні змінної *x*: $y=P_1*x^n+\ldots+P_n*x+P_{n+1}$
roots(*P*)	обчислення коренів полінома *P*
conv(*P,Q*)	обчислення добутку поліномів *P,Q*
[*Q,R*]=*deconv*(*B,A*)	ділення поліномів *B,A*; частка від ділення повертається у вигляді вектора *Q*, залишок – у вигляді вектора *R*, так що виконується співвідношення *B=A*Q+R*
polyder(*P*)	обчислення похідної від полінома *P*
[*Q,R*]=*polyder*(*B,A*)	похідна від відношення поліномів *B/A* повертається у вигляді відношення поліномів *Q/R*
polyfit(*x,Y,n*)	апроксимація дискретної функції *Y(x)* поліномом степеня *n*
[*R,P,K*]=*residue*(*B,A*)	розклад $B(x)/A(x)$ на прості дроби: $B(x)/A(x) = R_1/(x-P_1)+ R_2/(x-P_2)+\ldots+ R_n/(x-P_n)+K(x)$. Якщо корінь P_j має кратність *t*, будуть утворені вирази $R_j/(x-P_j)+ R_{j+1}/(x-P_j)^2+\ldots+ R_{j+t-1}/(x-P_j)^t$

Аналіз даних

Система MATLAB містить значну кількість функцій для використання в різних розділах математичної статистики для аналізу даних. Крім цього, до складу системи входить спеціалізований пакет *Statistics Toolbox*. Зупинимося лише на тих функціях для аналізу та статистичної обробки даних, які найчастіше використовуються для даних, заданих у вигляді числових масивів:

Функції для аналізу і обробки числових масивів	
Назва	**Призначення**
max(*x*)	максимальна компонента масиву
min(*x*)	мінімальна компонента масиву
mean(*x*)	середнє значення елементів масиву
median(*x*)	серединне значення елементів масиву (медіана)
std(*x*)	стандартне відхилення елементів масиву
sum(*x*)	сума елементів масиву
prod(*x*)	добуток елементів масиву
cumsum(*x*)	сумування з накопиченням: $y_i = \sum_{j=1}^{i} x_i$
cumprod(*x*)	добуток з накопиченням: $y_i = \prod_{j=1}^{i} x_i$
sort(*x*), *sort*(*x, 'ascend'*); *sort*(*x, 'descend'*)	сортування елементів масиву за зростанням; сортування елементів масиву за спаданням

ТЕМА 5
ЗАДАЧІ ЧИСЛОВОГО АНАЛІЗУ

До бібліотеки MATLAB входять функції, що реалізують низку методів апроксимації та інтерполяції даних, числового диференціювання та інтегрування, оптимізації, розв'язання нелінійних рівнянь [3, 5, 7 – 9].

Алгоритми наближення функції поліномами

Нехай на відрізку [a, b] задано вузли (різні) – *вузли інтерполяції*

$$a = X_1 < X_2 < ... < X_n = b. \tag{5.1}$$

Крім того, в цих вузлах відомі значення функції $f(X_i) = Y_i$. Необхідно побудувати *інтерполяційний поліном* (ІП)

$$P_{n-1}(x) = \sum_{k=1}^{n} p_k x^{k-1} \tag{5.2}$$

такий, що

$$P_{n-1}(X_i) = Y_i, \quad i = \overline{1,n}. \tag{5.3}$$

Враховуючи (5.2), (5.3), отримуємо для невідомих коефіцієнтів p_k, $k = \overline{1,n}$, СЛАР із визначником Вандермонда, який не дорівнює нулю (з огляду на властивість вузлів (5.1)), а отже ІП існує та єдиний. Крім "поліноміальної" форми представлення (5.2) існують й інші форми запису ІП.

Форма Лагранжа дозволяє представити поліном (5.2) через значення функції у вузлах інтерполяції $f(X_i)$ у вигляді

$$L_{n-1}(x) = \sum_{k=1}^{n} f(X_k) l_k(x). \tag{5.4}$$

Для знаходження поліномів $l_k(x)$ маємо: $\sum_{k=1}^{n} f(X_k) l_k(X_i) = f(X_i)$, $i = \overline{1,n}$, а отже $l_k(X_i) = \begin{cases} 1, k = i \\ 0, k \neq i \end{cases}$, звідки отримуємо $l_k(x) = \dfrac{\omega(x)}{(x - X_k)\omega'(X_k)}$, де уведено позначення $\omega(x) = \prod_{i=1}^{n} (x - X_i)$.

Форма Ньютона є представленням полінома (5.2) у вигляді різницевого аналога формули Тейлора.

Означення 1. *Поділені різниці* 1*-го порядку* для функції $f(x)$ визначають наступним чином

$$f(X_i; X_j) \equiv f_{i;j} = \frac{f(X_i) - f(X_j)}{X_i - X_j} = f_{j;i}, \quad (i \neq j). \tag{5.5}$$

Поділені різниці вищих порядків визначаються рекурентно:

$$f(X_i; X_j; X_k) \equiv f_{i;j;k} = \frac{f_{i;j} - f_{j;k}}{X_i - X_k}, \ \ ..., f_{i;j;...;s;k} = \frac{f_{i;j;...;s} - f_{j;...;s;k}}{X_i - X_k}. \tag{5.6}$$

Інтерполяційний поліном (5.2) у формі Ньютона має вигляд

$$N_{n-1}(x) = f(X_1) + (x - X_1) f_{1;2} + (x - X_1)(x - X_2) f_{1;2;3} + ... + (x - X_1)(x - X_2)...(x - X_{n-1}) f_{1;2;...;n}. \tag{5.7}$$

Похибка інтерполяції визначається як $r_{n-1}(x) = f(x) - P_{n-1}(x)$. Якщо $f(x) \in C^{(n)}([a,b])$, то маємо $|r_{n-1}(x)| = |f(x) - P_{n-1}(x)| \le \frac{M_n \cdot |\omega(x)|}{n!}$, $M_n = \max\limits_{x\in[a,b]} |f^{(n)}(x)|$.

Збіжності $|r_{n-1}(x)| \underset{n\to\infty}{\longrightarrow} 0$ у загальному випадку немає, вона суттєво залежить від вузлів сітки і гладкості функції $f(x)$.

До набору вбудованих функцій MATLAB для поліноміальної апроксимації і інтерполяції входять:

Функції апроксимації та інтерполяції даних	
Назва	**Призначення**
P=polyfit(x,y,n)	інтерполяційний поліном *P* степеня *n* для функції *y*, заданій на сітці *x*
w=griddata(x,y,z,t,r,M)	двовимірна інтерполяція функції *z(x,y)* на нерівномірній сітці; *t, r* – точки сітки, на якій визначається вихідний масив *w*, *M* – метод: *'linear'* (за замовчуванням), *'cubic'*

У системі MATLAB можна скористатися символьними обчисленнями і отримати ІП як у формі (5.4) так і у формі (5.7) (див. приклади відповідного розділу). ІП у формі (5.2) легко одержати, використовуючи вбудовану функцію `polyfit`.

Приклад 9. Побудувати інтерполяційний поліном у формі Ньютона.

Нижче наведено М-файл, який ілюструє роботу розглянутих функцій і використовує функцію користувача `ip_N` для побудови ІП у формі Ньютона:

```
%%pl5_1.m
% Приклади апроксимації і інтерполяції даних
clear all, close all, clc
a = 0; b = 10; n =10;
x = linspace(a, b, n+1);
y = sin(x);
P = polyfit(x, y, n)      %  Інтерполяційний поліном
t = linspace(a, b, 10*n+1);
w = polyval(P, t);        %  Інтерполяція даних
% Графіки порівняння вихідної функції і інтерполяції
plot(x, y, t, w, t, sin(t))
% Побудова інтерполяційного поліному у формі Ньютона
[N, err] = ip_N(x, y);
if ~err
   N
end
```

```
function [N, err] = ip_N(X, Y)
% Побудова інтерполяційного поліному у формі Ньютона
% вхідні параметри:
%   X - вектор вузлів інтерполювання (відсутні кратні)
%   Y - вектор значень функції у вузлах інтерполювання
% вихідні параметри:
%   N - вектор, який є представленням інтерполяційного поліному Nm(t)
% err = 1- при перевірці вхідних даних знайдена помилка і
%           виконання функції аварійно закінчено
%       0- перевірка вхідних даних пройшла успішно.
%
N = sort(X); err=0;
if nargin<2
   err = 1;
```

```
   error('ip_N:NotEnoughInputs',...
         'Not enough input arguments. See ip_N.');
   return
end
n = length(X); n1 = n - 1;
% перевірка, чи є кратні вузли інтерполювання ?
c = 0; i = 1;
while ~c & i<n
   j = i;  i = i + 1; c = N(j) == N(i);
end
clear N;
if c
   err = 1;
   error('ip_N:NotEnoughInputs',...
         ' Є кратні вузли інтерполювання. See ip_N.');
   return
end
% обчислення поділених різниць;
% обчислення поліномів Wi(t)= t-X(i), i=1,...,n-1;
for i = 1 : n1
   for j = n : -1 : i+1
      Y(j) = (Y(j-1)-Y(j))./(X(j-i)-X(j));
   end
   W(i,1) = 1;  W(i,2) = -X(i);
end
% побудова інтерполяційного поліному Nm(t)
j = 1; N(j) = Y(n);
for i = n1 : -1 : 1
   N = conv(W(i,:), N);
   j = j+1; N(j) = N(j) + Y(i);
end
return
```

Числове диференціювання

Постановка задачі. Нехай $L_m(u(x)) = \sum_{j=0}^{m} a_j u^{(j)}(x)$ – лінійний диференціальний оператор m-го порядку. Необхідно його наближено обчислити в деякій точці $x_i \in \omega_h$, де ω_h – рівномірна або нерівномірна сітка.

Схема розв'язання. Заміняємо наближено $u(x) \approx L_n(x) \in \mathrm{P}_n(x) \;\Rightarrow\; u^{(j)}(x) \approx L_n^{(j)}(x)$.

Найпростіші формули числового диференціювання отримують, вибираючи певний *шаблон S* із точок рівномірної сітки ω_h:

Шаблон	Похідна	Позначення і обчислення	Назва похідної	Точність
$S[x_{i-1}, x_i]$	$u'(x_i)$	$u_{\bar{x},i} = \frac{u_i - u_{i-1}}{h}$	*ліва різницева похідна*	$O(h^1)$
$S[x_i, x_{i+1}]$	$u'(x_i)$	$u_{x,i} = \frac{u_{i+1} - u_i}{h}$	*права різницева похідна*	$O(h^1)$
$S[x_{i-1}, x_i, x_{i+1}]$	$u'(x_i)$	$u_{x,i} = \frac{u_{i+1} - u_{i-1}}{2h}$	*центральна різницева похідна*	$O(h^2)$
$S[x_{i-1}, x_i, x_{i+1}]$	$u''(x_i)$	$u_{\bar{x}x,i} = \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2}$	*друга різницева похідна*	$O(h^2)$

Для отримання вищої точності можна збільшити кількість точок шаблону або скористатися формулою *Рунге-Ромберга*.

Нехай для обчислення $\eta(x)$ використовується деяка наближена розрахункова формула $\xi(x,h)$ з порядком точності p

$$\eta(x) = \xi(x,h) + O(h^p) = \xi(x,h) + \varphi(x)\cdot h^p + O(h^{p+1}). \quad (5.8)$$

Використовуючи дві зв'язані сітки з кроками $h_1 = h$ і $h_2 = k\cdot h$ можна отримати для (5.8):

$$\varphi(x)\cdot h^p = \frac{\xi(x,k\cdot h) - \xi(x,h)}{1-k^p}, \quad (5.9)$$

$$\eta(x) = \xi(x,h) + \frac{\xi(x,k\cdot h) - \xi(x,h)}{1-k^p}. \quad (5.10)$$

Формула (5.9) – *перша формула Рунге-Ромберга.* Вона використовується для перевірки, чи знайдено результат з необхідною точністю ε. Формула (5.10) – *друга формула Рунге-Ромберга.* Вона використовується як для підвищення точності обчислень, так і для побудови нових розрахункових формул з порядком не нижче ніж $p+1$.

У системі MATLAB числове диференціювання представлено такими функціями:

Функції числового диференціювання	
Назва	**Призначення**
dx=diff(x)	обчислення скінчених різниць, тобто *dx*=[*x*(2)-*x*(1), *x*(3)-*x*(2), ..., *x*(*end*)-*x*(*end*-1)]
[*px,py*]=*gradient*(*z,hx,hy*)	обчислення дискретного аналога градієнта функції двох змінних *z*(*x,y*). Тут *hx,hy* – кроки диференціювання по *x* і по *y*, *px, py* – матриці, що містять компоненти градієнта
L=del2(Z)	обчислення "дискретного Лапласіана" для матриці *Z*

Приклад 10. Обчислити різницеві похідні, дискретний аналога градієнта функції двох змінних та оператора Лапласа.

```
%%pl5_2.m
% Приклади числового диференціювання
clear all, close all, clc
n = 101; x = linspace(0, 2*pi, n);
n1 = n - 1; h = x(2) - x(1); hh = h.*h;
y = sin(x); dy = cos(x);

% різницева перша похідна
Dy = diff(y);
dydx = Dy./h;           %  для точок x(1:end-1) - права різниця
dydx(n) = dydx(n1);     %  для точки x(end)     - ліва різниця
disp(strcat('1-ша похідна : h  =',num2str(h),...
       ' абсолютна похибка=',num2str(norm(dydx-dy,inf))))
plot(x, dy, x, dydx)
grid on, title('1-ша різницева похідна')
pause, close all, clear Dy dydx

% різницева друга похідна
%  для точок x(2:end-1) - центральна різниця
d2ydx2 = zeros(1,n-2);
for k = 2 : n1
```

```
    k1 = k-1;
    d2ydx2(k1) = (y(k1)-2*y(k)+y(k+1))./hh;
  end
  disp(strcat('2-га похідна : h^2=',num2str(hh),...
        ' абсолютна похибка=',num2str(norm(d2ydx2-(-y(2:n1)),inf))))
  plot(x, -y, x(2:n1), d2ydx2)
  grid on, title('2-га різницева похідна')
  pause, close all, clear x y d2ydx2

  f = @(x,y)(16 - x.^4 - y.^4);
  x = linspace(-2, 2, 21); hx = x(2) - x(1);
  y = linspace(-2, 2, 21); hy = y(2) - y(1);

  [X, Y] = meshgrid(x, y); Z = f(X, Y);
  [Px, Py] = gradient(Z, hx, hy);
  contour(Z), hold on
  quiver(Px, Py), hold off
  pause, close all, clear Px Py
  mesh(X, Y, Z), grid on
  pause, close all
  L = del2(Z); mesh(X, Y, L), grid on
  pause, close all
```

Отримаємо результат:

```
1-ша похідна : h  =0.062832 абсолютна похибка=0.031406
2-га похідна : h^2=0.0039478 абсолютна похибка=0.00032894
```

Числове інтегрування

Постановка задачі. Необхідно наближено обчислити визначений інтеграл

$$I=\int_a^b f(x)dx \quad \text{або} \quad I=\int_a^b \rho(x)u(x)dx, \tag{5.11}$$

де $\rho(x)>0$ – вагова функція, неперервна на (a,b), $f(x)|u(x)$ – підінтегральні функції достатньої гладкості на $[a,b]$.

Більшість методів числового інтегрування ґрунтуються на заміні (5.11) скінченною сумою – *квадратурними формулами* (КФ)

$$I_n=\sum_{k=1}^{n} A_k f_k \quad \text{або} \quad I_n=\sum_{k=1}^{n} A_k u_k, \tag{5.12}$$

де A_k – *квадратурні коефіцієнти*, x_k – *вузли* КФ.

Означення 1. Якщо граничні точки $x=a$, $x=b$ входять до системи вузлів КФ, то формули (5.12) називаються формулами *закритого* типу, інакше – *відкритого* типу.

Означення 2. КФ називається *чисельно стійкою*, якщо всі A_k невід'ємні.

Означення 3. *Похибкою* КФ називається величина $|I-I_n|=R_n$. Вона залежить від квадратурних коефіцієнтів і вузлів $\{A_k\}$, $\{x_k\}$, а також гладкості функції $f|u$.

Формули (5.12) в цілому містять $2n$ коефіцієнтів, тому є можливість будувати різні КФ, що ґрунтуються на певній стратегії:

- різні методи заміни підінтегральної функції $f|u$, як на всьому інтервалі $[a,b]$, так і на його складових $[x_{k-1},x_k]$;

- фіксація вибраного набору параметрів $\{x_k\}$ або $\{A_k\}$ з подальшим визначенням решти параметрів, що задовольняють певним додатковим умовам;
- забезпечення найменшої оцінки залишку R_n для всіх функції $u(x)$ в деякому класі U;
- забезпечення *точного* ($R_n = 0$) обчислення (5.11) формулою (5.12) для всіх функцій $u(x)$ з деякого класу U.

Найпростіші формули числового інтегрування отримують, вибираючи на $[a,b]$ фіксовану рівномірну сітку $\omega_h = \{x_k = a + (k-1)h, k = \overline{1,n}, h = (b-a)/(n-1)\}$ і користуючись адитивністю (5.11)

$$I = \int_a^b f(x)dx = \sum_{k=2}^{n} \int_{x_{k-1}}^{x_k} f(x)dx . \qquad (5.13)$$

Назва	**Обчислення** $I_n(k) = \int_{x_{k-1}}^{x_k} f(x)dx$	**Застосування складеної КФ** $I_n = \sum_{k=2}^{n} I_n(k)$	**Точність**
1. *КФ прямокутників* *а) лівих*	hf_{k-1}	$h\sum_{k=1}^{n-1} f_k$	$O(h^1)$
б) правих	hf_k	$h\sum_{k=2}^{n} f_k$	$O(h^1)$
в) центральних	$hf_{k-0.5}$, $x_{k-0.5} = 0.5(x_{k-1} - x_k)$	$h\sum_{k=2}^{n} f_{k-0.5}$	$O(h^2)$
2. *КФ трапецій*	$0.5h(f_{k-1} + f_k)$	$h\left(0.5(f_1 + f_n) + \sum_{k=2}^{n-1} f_k\right)$	$O(h^2)$
3. *КФ Сімпсона*	$\frac{0.5h}{3}(f_{k-1} + 4f_{k-0.5} + f_k)$	$\frac{h}{3}\left(f_1 + f_n + 4\sum_{k=2,4,...}^{n-1} f_k + 2\sum_{k=3,5,...}^{n-2} f_k\right)$	$O(h^4)$

Для отримання значення (5.11) із заданою точністю $0 < \varepsilon < 1$ використовують адаптивні алгоритми (як рекурентні так і рекурсивні), які будуються на апостеріорній оцінці похибки R_n за допомогою першої формули Рунге-Ромберга.

У системі MATLAB числове інтегрування представлено вбудованими функціями:

Функції числового інтегрування	
Назва	**Призначення**
*s=h.*trapz(y)*	метод трапецій для дискретної функції *y* з постійним кроком *h*
s=trapz(x,y)	метод трапецій для дискретної функції *y*, заданої у вузлах сітки *x*
s=quad(@f,a,b,tol)	метод Сімпсона
s=quadl(@f,a,b,tol)	метод Сімпсона (підвищеної точності) для функції $f(x)$, описаної у М-файлі *f.m*; [*a,b*] – інтервал інтегрування; *tol* – точність рекурсивного адаптивного алгоритму (необов'язковий параметр, за замовчуванням рівний 1*e*-6)
s=dblquad(@f,a,b,c,d)	метод Сімпсона для функції $f(x,y)$, описаної у М-файлі *f.m*; $x \in [a,b], y \in [c,d]$

Приклад 11. Обчислити визначений інтеграл (5.11) для a=1, b=2, $f(x) = x\sin(x)$ за формулами числового інтегрування, використовуючи вбудовані функцій MATLAB.

```
function pl5_3
% Приклади числового інтегрування
clear all, close all, clc

a = 1; b = 2;
x = linspace(a, b, 11); h = x(2) - x(1);
n = length(x); y = f(x); n1 = n - 1;

disp(' КФ прямокутників :')
Pl = h .* sum(y(1:n1));                    % лівих
Pr = h .* sum(y(2:n));                     % правих
Pz = h .* sum(f(0.5.*(x(1:n1)+x(2:n)))); % центральних
disp([Pl Pr Pz])

disp(' КФ трапецій :')
It1 = h .* trapz(y);          %  з рівномірним кроком
x = [1,1.15,1.2,1.253,1.35,1.5,1.65,1.8,1.9,1.93,2];
y = f(x); It2 = trapz(x, y);  %  з нерівномірним кроком
disp([It1 It2])

disp(' КФ Сімпсона (рекурсивний адаптивний алгоритм):')
Is = quad(@f, a, b, 1e-4);    % точність користувача  1e-4
Isd = quadl(@f, a, b);        % точність за замовчуванням  1e-6
disp([Is Isd])

disp(' Символьне обчислення визначеного інтегралу :')
syms z; F = f(z); I = vpa(int(F, z, a, b), 10)

disp(' Всі результати :')
Z = vpa([Pl, Pr, Pz, It1, It2, Is, Isd], 10)
disp(' абсолютна похибка :')
disp(abs(Z - I))

function s = f(t)
% означення підінтегральної функції
s = t .* sin(t);
```

Отримаємо результати:

```
 КФ прямокутників :
    1.3905    1.4882    1.4410
 КФ трапецій :
    1.4393    1.4388
 КФ Сімпсона (рекурсивний адаптивний алгоритм):
    1.4404    1.4404
 Символьне обчислення визначеного інтегралу :
I =
1.440422421
 Всі результати :
Z =
[  1.390478757,  1.488191144,  1.440966217,  1.439334951,  1.438761144,
1.440422419, 1.440422421]
 абсолютна похибка :
[0.049943663801121924994408374232 08, 0.047768723083225099468318575191006,
0.000543796086992058558262641554392 76, 0.001087470358948301740742437004883,
0.001661276497483717140557502034425 9,0.000000002409954909243783305328179267 2351,
0.000000000000065066008136938080497202463448048]
```

ТЕМА 6
АЛГОРИТМИ НАБЛИЖЕННЯ ФУНКЦІЙ ЛІНІЙНИМИ І КУБІЧНИМИ СПЛАЙНАМИ

Ураховуючи, що в загальному випадку відсутня збіжність системи інтерполяційних поліномів (ІП) на всьому відрізку наближення, було запропоновано будувати не один ІП на всьому відрізку $[a, b]$, а використовувати *кусково-поліноміальну інтерполяцію*. У цьому випадку на кожному інтервалі Δ_i застосовують поліном зазвичай низького степеня. Коефіцієнти кожного із поліномів цієї системи визначають із умов інтерполяції на сітці та деяких додаткових умов – відомостей про функцію, що інтерполюється.

Побудований у такий спосіб інтерполянт називають *сплайн-функцією* або просто *сплайном*. Перевага сплайнів перед звичайною інтерполяцією полягає у простоті обчислень полінома, стійкості процесу обчислень, збіжності.

Назва "сплайн" походить від англійського *spline*, що означає гнучку металеву лінійку, яка використовувалась як лекало (інструмент кресляра для проведення гладкої лінії через задані точки площини). Профіль лінійки, розташований між двома сусідніми точками, математично описує кубічний поліном.

Нехай на відрізку $[a, b]$ задана *сітка* з $n > 1$ вузлами

$$a = X_1 < X_2 < \ldots < X_{n-1} < X_n = b\,. \tag{6.1}$$

Уведемо позначення:

$$\begin{aligned} &\Delta_{[a,b]} = \bigcup_{i=1}^{n-1} \Delta_i, \\ &\Delta_i = [X_i, X_{i+1}], \\ &h_i = X_{i+1} - X_i,\ i = 1,\ldots,n-1. \end{aligned} \tag{6.2}$$

На сітці відомі значення деякої функції $u(x)$:

$$y_i = u(X_i),\ \ i = 1,\ldots,n.$$

Означення 1. Функцію $S_{m,q}(x),\ x \in [a, b]$ називають *сплайном степеня* $m > 0$ *дефекту* q $(0 \le q \le m)$, якщо

а) на кожному відрізку $\Delta_i,\ i = 1,\ldots,n-1$ вона є поліномом степеня m, тобто

$$S_{m,q}(x) = \sum_{k=0}^{m} s_{i,k}(x - X_i)^k,\ x \in \Delta_i; \tag{6.3}$$

б) функція $S_{m,q}(x)$ неперервна на $[a, b]$ разом зі своїми похідними до степеня $m - q$ включно:

$$S_{m,q}(x) \in C^{m-q}([a,b]). \tag{6.4}$$

Означення 2. Сплайн $S_{m,q}(x),\ x \in [a, b]$ називають *інтерполяційним* для функції $u(x)$ і позначають $S_{m,q}(x;u)$, якщо він задовольняє умову інтерполяції у вузлах сітки

$$S_{m,q}(X_i) = y_i,\ i = 1,\ldots,n\,. \tag{6.5}$$

Представлення (6.3) можна записати і в інших формах, наприклад, "поліноміальній" або у формі "лагранжіана".

Побудова інтерполяційного лінійного сплайна

Ураховуючи, що для ІС першого порядку виконуються умови (6.4) – (6.5), дефект лінійного сплайна дорівнюватиме одиниці. Для ІС $S_{1,1}(x;u), x \in \Delta_{[a,b]}$ використовуватимемо "сплайнове" зображення

$$S_{1,1}(x;u) = a_i(x - X_i) + b_i,\ x \in \Delta_i, i = 1,...,n-1. \tag{6.6}$$

Система (6.6) визначається коефіцієнтами $\{a_i, b_i\}_{i=1}^{n-1}$, кількість яких дорівнює 2(n–1). Для їх знаходження скористаємося зображенням (6.6) та умовами (6.4), (6.5). У результаті маємо систему

$$\begin{cases} b_i = y_i,\ \ i = 1,...,n-1, \\ a_{n-1}h_{n-1} + b_{n-1} = y_n,\ \ i = n, \\ a_{i-1}h_{i-1} + b_{i-1} = b_i,\ \ i = 2,...,n-1 \end{cases}$$

із кількістю рівнянь 2(n–1). Її розв'язок легко знайти в явній (алгебраїчній) формі:

$$\begin{cases} b_i = y_i, \\ a_i = \dfrac{b_{i+1} - b_i}{h_i} \equiv \dfrac{y_{i+1} - y_i}{X_{i+1} - X_i} \equiv y_{i;i+1},\ i = 1,...,n-1. \end{cases} \tag{6.7}$$

Тут ми формально позначили $b_n = y_n$, а також використали стандартне позначення $y_{i;i+1} = y(X_i; X_{i+1})$ для *поділених різниць першого порядку*.

Зауважимо, що подання одних коефіцієнтів (a_i) за допомогою інших (b_i) є типовим при побудові сплайнів.

Досить часто при застосуванні сплайн-функцій необхідно представити ІС на відрізку $[X_i, X_{i+1}]$ сітки у формі "лагранжіана":

$$\begin{aligned} &S_{1,1}(x;u) = y_i\rho_{i,1}(x) + y_{i+1}\rho_{i,2}(x),\ x \in \Delta_i, \\ &\rho_{i,1}(x) = (X_{i+1} - x)h_i^{-1};\ \rho_{i,2}(x) = (x - X_i)h_i^{-1}. \end{aligned} \tag{6.8}$$

Якщо увести позначення

$$t = (x - X_i)h_i^{-1},\ x \in \Delta_i,\ i = 1,...,n-1,\ t \in [0,1], \tag{6.9}$$

то сплайн (6.8) набуває вигляду

$$\begin{aligned} &S_{1,1}(x;u) = y_i\psi_1(t) + y_{i+1}\psi_2(t); \\ &x \in \Delta_i,\ i = 1,...,n-1,\ t \in [0,1]; \\ &\psi_1(t) = 1 - t,\ \psi_2(t) = t. \end{aligned} \tag{6.10}$$

Побудова інтерполяційного кубічного сплайна

Оскільки для сплайна третього порядку має виконуватись умова (6.4) із гладкістю до другої похідної включно, дефект кубічного сплайна дорівнюватиме 1.

Для ІС $S_{3,1}(x;u), x \in \Delta_{[a,b]}$ використовуватимемо "сплайнове" зображення

$$\begin{aligned} &S_{3,1}(x;u) = a_i(x - X_i)^3 + b_i(x - X_i)^2 + c_i(x - X_i) + d_i, \\ &\quad x \in \Delta_i,\ i = 1,...,n-1. \end{aligned} \tag{6.11}$$

Легко визначити зміст коефіцієнтів у (6.11):

$$d_i = S_{3,1}(X_i;u), c_i = S'_{3,1}(X_i;u), b_i = \frac{1}{2}S''_{3,1}(X_i;u), a_i = \frac{1}{6}S'''_{3,1}(X_i;u).$$

Система (6.11) визначається коефіцієнтами $\{a_i, b_i, c_i, d_i\}_{i=1}^{n-1}$, кількість яких дорівнює 4(n–1). Для їх знаходження використаємо:

1) умову інтерполяції (6.5), що дає систему $(n-1)+1$ рівнянь

$$\begin{cases} d_i = y_i,\ i = 1,\ldots,n-1, \\ a_{n-1}h_{n-1}^3 + b_{n-1}h_{n-1}^2 + c_{n-1}h_{n-1} + d_{n-1} = y_n,\ i = n; \end{cases} \tag{6.12}$$

2) умови гладкості сплайна (6.4) в усіх внутрішніх вузлах сітки

$$S_{3,1}^{(p)}(X_i - 0; u) = S_{3,1}^{(p)}(X_i + 0; u),\ p = 0,1,2,\ i = 2,\ldots,n-1,$$

що дає систему $3(n-2)$ рівнянь

$$a_i h_i^3 + b_i h_i^2 + c_i h_i + d_i = d_{i+1}, \tag{6.13}$$

$$3a_i h_i^2 + 2b_i h_i + c_i = c_{i+1}, \tag{6.14}$$

$$6a_i h_i + 2b_i = 2b_{i+1},\ i = 1,\ldots,n-2; \tag{6.15}$$

3) дві додаткові умови, необхідні для утворення замкненої системи рівнянь. Найчастіше використовують такі додаткові умови – *типи крайових умов для кубічного сплайна*:

$$\text{I. } S'_{3,1}(a;u) = A,\ S'_{3,1}(b;u) = B; \tag{6.16}$$

$$\text{II. } S''_{3,1}(a;u) = A,\ S''_{3,1}(b;u) = B; \tag{6.17}$$

$$\text{III. } S_{3,1}^{(p)}(a;u) = S_{3,1}^{(p)}(b;u),\ p = 1,2; \tag{6.18}$$

$$\text{IV. } S'''_{3,1}(X_j + 0;u) = S'''_{3,1}(X_j - 0;u),\ j = 2,n-1. \tag{6.19}$$

Система (6.12) – (6.15), доповнена одним з рівнянь (6.16) – (6.19), може бути розв'язана безпосередньо, але краще (з огляду на кількість операцій і об'єм оперативної пам'яті) визначити, наприклад, $\{a_i, b_i\}_{i=1}^{n-1}$ через $\{c_i\}_{i=1}^{n-1}$. При цьому ми отримаємо для $\{c_i\}_{i=1}^{n-1}$ СЛАР з тридіагональною матрицею, яка має діагональну перевагу.

Вбудовані функції системи MATLAB для роботи зі сплайнами

Для кусково-поліноміальної апроксимації табличних функцій $y(x)$ призначена функція `interp1`:

```
yy = interp1(x, y, xx, method),
```

де `x` – абсциси функції, що апроксимується; `y` – ординати функції, що апроксимується; `xx` – абсциси точок, в яких обчислюють значення апроксимуючих поліномів, що повертаються у вигляді вектора *yy*; `method` – спосіб апроксимації, що задається як рядок символів (насправді достатньо вказати лише перший символ).

Параметр `method` може набувати одного зі значень:

- `'nearest'` – інтерполяція за сусідніми точками, за якої значення у довільній точці дорівнює значенню у найближчій вузловій точці (апроксимація кусковими поліномами нульового степеня ("сходинками"): для довільного проміжного значення *xx* необхідно знайти найближче табличне x_i і за *yy* прийняти відповідне табличне значення y_i);
- `'linear'` – лінійна інтерполяція (апроксимація ламаними – кусковими поліномами першого степеня);
- `'spline'` – кубічний сплайн (апроксимація сплайнами – кусковими поліномами третього степеня);
- `'pchip'` або `'cubic'` – апроксимація кусковими поліномами Ерміта третього степеня.

Якщо параметр `method` опущено, то за замовчуванням використовують `'linear'`.

За апроксимації методом *'spline'* у вузлових точках неперервні перша і друга похідні, а у внутрішніх вузлах, сусідніх із кінцевими, неперервна також і третя похідна (умова (6.19)).

Назви методів *'spline'* та *'pchip'* є іменами функцій, до яких звертається функція *interp*1 (інші два методи *'nearest'* та *'linear'* опрацьовуються безпосередньо в *interp*1). Звернення до цих функцій має два різні формати:

1) *yy = spline(x, y, xx)* та *yy = pchip(x, y, xx)*;

2) *pp = spline(x, y)* та *pp = pchip(x, y)*.

У першому варіанті вхідні та вихідні параметри мають той самий зміст, що й при зверненні до *interp1*.

У другому варіанті обидві функції повертають *pp*-структуру, так звану кусково-поліноміальну форму (*ppform*, *pieceweise polinomial form*) – спеціальний внутрішній формат системи MATLAB для представлення сплайнів. Ці структури можуть задаватися як аргументи при зверненні до функції *ppval* для обчислення значень апроксимуючих функцій у формі *yy = ppval(pp, xx)*.

У системі MATLAB також реалізована багатовимірна інтерполяція, яка представлена наступними процедурами для інтерполяції функцій 2-х, 3-х і *n* змінних:

- *ZZ=interp2(X,Y,Z,XX,YY,method);*
- *UU=interp3(X,Y,Z,U,XX,YY,ZZ,method);*
- *UU=interpn(X1,X2,...,Xn,U,XX1,XX2,...,XXn,UU);*

Розглянемо практичне застосування вбудованих функції для роботи зі сплайнами.

Приклад 12. Наблизити функції $f(x)$, для яких відсутня збіжність системи інтерполяційних поліномів на всьому відрізку наближення :

1) $f(x) = 1/(1+25x^2)$

2) $f(x) = |x|$

за допомогою $S_{0,1}(x; f)$ – сплайна нульового степеня і зобразити графік табуляції вихідної функції $f(T)$ і сплайна $S_{0,1}(T; f)$ у точках масиву табуляції T.

```
%%pl6_1.m
% Приклади апроксимації і інтерполяції даних
% з використанням інтерполяційного сплайна порядку 0
% Виконати на рівномірній сітці наближення
% 1) функції f(x)=1/(1+25*x^2) на відрізку [-1,1];
% 2) функції f(x)=|x| на відрізку [-1,1].
clear all, close all, clc
SP = 51; a = -1; b = 1; X = linspace(a, b, SP);
T = linspace(a, b, 201);  % масив табуляції

% 1) приклад К.Рунге
f = @(x)(1./(1+25.*x.*x));
NF(1) = {'f(x)'}; F(1,:) = f(T); y = f(X);
NF(2) = {strcat('S0(x;f) -',int2str(SP))};
F(2,:) = interp1(X, y, T, 'nearest');
plot(T, F), grid on
title('Наближення f(x)=1/(1+25x^2) ІП S0(x;f)')
legend(NF,'Location','Best')
pause, close all
```

```
% 2) приклад С.Н.Бернштейна
f = @(x)(abs(x));
F(1,:) = f(T); y = f(X);
F(2,:) = interp1(X, y, T, 'nearest');
plot(T, F), grid on
title('Наближення f(x)=|x| ІП S0(x;f)')
legend(NF,'Location','Best')
pause, close all
```

Приклад 13. Розв'язати задачу з попереднього прикладу за допомогою $S_{3,1}(x;f)$ – кубічного сплайна.

```
%%pl6_2.m
% Приклади апроксимації і інтерполяції даних
% з використанням кубічного інтерполяційного сплайна
% Виконати на рівномірній сітці наближення
% 1) функції f(x)=1/(1+25*x^2) на відрізку [-1,1];
% 2) функції f(x)=|x| на відрізку [-1,1].
clear all, close all, clc
SP = 21; a = -1; b = 1; X = linspace(a, b, SP);
T = linspace(a, b, 201);  % масив табуляції

% 1) приклад К.Рунге
f = @(x)(1./(1+25.*x.*x));
NF(1) = {'f(x)'}; F(1,:) = f(T); y = f(X);
NF(2) = {strcat('S3(x;f) -',int2str(SP))};
F(2,:) = spline(X, y, T);
plot(T, F), grid on
title('Наближення f(x)=1/(1+25x^2) ІП S3(x;f)')
legend(NF,'Location','Best')
pause, close all

% 2) приклад С.Н.Бернштейна
f = @(x)(abs(x));
F(1,:) = f(T); y = f(X);
F(2,:) = spline(X, y, T);
plot(T, F), grid on
title('Наближення f(x)=|x| ІП S3(x;f)')
legend(NF,'Location','Best')
pause, close all
```

Приклад 14. Для дискретно-заданої функції побудувати кубічний сплайн і вивести на екран *pp*-структуру.

```
%%pl6_3.m
% Побудова кубічного сплайна для дискретної функції
clear all, close all, clc
x = -3 : 3; y = sign(x);
plot(x, y, 'o'), axis( [-3.5, 3.5, -1.5, 1.5] )
xlabel('x'), ylabel('y')
pp = spline(x, y)
X = pp.breaks
C = pp.coefs
```

Результат виведення на екран *pp*-структури (з доданими коментарями у правому стовпці для пояснення полів):

```
pp =

      form: 'pp'                 форма сплайна
    breaks: [-3 -2 -1 0 1 2 3]   масив точок розбиття
     coefs: [6x4 double]         масив коефіцієнтів сплайна
    pieces: 6                    кількість поліномів, що складають сплайн
     order: 4                    кількість коефіцієнтів в поліномі
       dim: 1                    розмірність


X =

    -3    -2    -1     0     1     2     3

C =

       0.25        -0.75          0.5          -1
       0.25            0        -0.25          -1
      -0.25         0.75          0.5          -1
      -0.25            0         1.25           0
       0.25        -0.75          0.5           1
       0.25  5.5511e-017        -0.25           1
```

Для детальнішого ознайомлення з основними питаннями теорії поліноміальних сплайнів та їх застосувань у прикладних задачах і способами використання вбудованих функцій пакету MATLAB для роботи зі сплайнами можна рекомендувати, наприклад, [4, 6, 20].

ТЕМА 7
ВИКОРИСТАННЯ М-ФУНКЦІЙ

Опис функції. Опис функції здійснюється в окремому файлі, ім'я якого має збігатися з ім'ям функції, яка в ньому розташована. Синтаксис цього файла (*ff.m*):

```
function [R1 R2 … Rn] = ff(x1,x2,…,xn)
%FF   Короткий зміст функції ff
% Детальний опис функції ff
% xi – вхідні формальні параметри
% Ri – вихідні формальні параметри
% коментар
%
<оператори>;  return
end
```

Для *фізичного* закінчення опису тіла функції можна використовувати ключове слово `end`, або просто завершення тексту файлу. Для *логічного* закінчення – використовується ключове слово `return`.

Усі поіменовані дані, які можуть використовуватись у тілі функції, є *локальними* або *формальними вхідними* або *вихідними параметрами*. Усі вихідні параметри мають бути визначені в тілі функції. Якщо деяка змінна *y* є вхідним і вихідним параметром, то вона повинна фігурувати в обох списках параметрів. Сукупність вхідних параметрів може бути порожньою – у цьому випадку можна опустити й дужки (). У випадку порожньої сукупності вихідних параметрів опускають і знак =. Якщо вихідний параметр один, то можна опустити й [ ].

Інформацію, розміщену в тексті коментаря М-функції, можна відобразити командою `help ff`, а весь текст М-функції – надрукувати командою `type ff`. Перший рядок коментаря (при належному оформленні!) має певний зміст. Він використовується командою `lookfor ff` для пошуку функції за назвою `ff`.

При програмуванні функцій буває корисно використовувати вбудовані функції:
`nargin` – кількість вхідних параметрів поточної функції;
`nargin(fun)` – кількість вхідних параметрів функції з ім'ям `fun`;
`nargout` – кількість вихідних параметрів поточної функції;
`nargout(fun)` – кількість вихідних параметрів функції з ім'ям `fun`.

Для створення змінної кількості вхідних / вихідних параметрів використовується ключове слово `varargin / varargout`, яке записують після позиційних вхідних / вихідних параметрів :

```
function [R1, R2, varargout] = fun(x1, x2, varargin)
```

або взагалі без позиційних вхідних / вихідних параметрів :

```
function [varargout] = fun(varargin)
```

У тілі функції `fun` до `varargin / varargout` можна звертатись як до масиву комірок, наприклад, передавати `varargin{:}` при зверненні до іншої функції.

У випадку визначення простої функції від аргументів *x1,x2,…,xn*, тобто такої, значення якої обчислюється у вигляді деякого виразу *Expr(x1,x2,…,xn)*, опис можна подати так:

а) за допомогою дескриптора *@* (інша назва *анонімна* функція)

```
ff=@(x1,x2,…,xn)(Expr(x1,x2,…,xn));
```

б) за допомогою команди *inline* (*inline*-функція)

```
ff=inline('Expr(x1,x2,…,xn)');
```

Зазначимо, що при описі обчислювальної частини функції (в тілі, у виразі *Expr*) потрібно завжди записувати крапку для вказівки поелементних арифметичних операцій. Крапку можна не ставити між числом (константою) і змінною у випадку запису арифметичної операції.

Для збереження інформації між викликами даної М-функції використовується атрибут *persistent*. Наприклад, *persistent t1 t2 t3*.

При описі функції в М-файлі допускаються такі можливості :

А) після опису *основної* функції *pl7_1* (саме вона і тільки вона доступна ззовні), можуть бути розташовані *підфункції,* опис яких починається ключовим словом *function*, а закінченням тіла є кінець файла *pl7_1.m*, або останній оператор перед початком опису нової підфункції або ключове слово *end*.

Продемонструємо схему опису і використання підфункцій.

```
function pl7_1(varargin)     % основна функція pl7_1 (файл pl7_1.m)
   n = length(varargin);
   if n == 1
      e = varargin{1};
   elseif n == 2
      e = f1(varargin{2});
   elseif n == 3
      e = f2(varargin{3});
   else
      [e, es] = f(varargin{4:n});
      es
   end
   e
end      % кінець опису pl7_1 (необов'язковий, якщо відсутні вкладені функції)

function [rf, varargout] = f(varargin) % підфункція f
   rf = [varargin{:}];
   varargout = {rf};
   rf = sum(rf);
end      % кінець опису f (необов'язковий, якщо відсутні вкладені функції)

function rf1 = f1(a)                   % підфункція  f1
   rf1 = a;
end      % кінець опису f1 (необов'язковий, якщо відсутні вкладені функції)

function rf2 = f2(a)                   % підфункція  f2
   rf2 = a;
end      % кінець опису f2 (необов'язковий, якщо відсутні вкладені функції)
```

Виклик в основній програмі має вигляд:

```
M = {1, 22, 33, 444, 555, 666};
pl7_1(M{1}); pl7_1(M{1:2}); pl7_1(M{1:3}); pl7_1(M{:})
```

Виклик підфункцій (*f1, f2, f*) можливий тільки з основної функції *pl7_1* або підфункцій, розміщених у даному файлі *pl7_1.m* .

Б) в описі *основної* функції можуть бути розташовані (в довільному місці) *вкладені функції,* опис яких починається ключовим словом `function`, а закінчення тіла обов'язково фіксується ключовим словом `end`:

```
function e = pl7_2(a, b, c) % основна функція pl7_2 (файл pl7_2.m)
   function rf = f(x)
      rf = f1(c) + x;
      function rf1 = f1(x)
         rf1 = f2(x) + b;
      end                  % кінець опису rf1, наявність обов'язкова!
      function rf2 = f2(x)
         rf2 = a + x;
      end                  % кінець опису rf2, наявність обов'язкова!
   end                     % кінець опису rf, наявність обов'язкова!
   e = f(5);
end                        % кінець опису pl7_2, наявність обов'язкова!
```

Виклик в основній програмі має вигляд:

```
pl7_2(1, 2, 3)
```

Використовувати (поєднувати) обидва ці варіанти в одній М-функції допускається тільки при умові, що останнім оператором основної функції і підфункцій обов'язково є `end`.

Локальні та глобальні змінні. *Локальними* називаються змінні, які використовуються у даній основній функції, підфункціях, або у вкладених функціях, причому область видимості обмежується тілом основної функції, окремим тілом кожної підфункції, або поширюється на всі вкладені функції. *Глобальними* називаються змінні, оголошені за допомогою атрибута `global`, наприклад,

```
global g1 g2 g3
```

Означення `global` має бути розміщено до першого використання відповідних змінних у тілі функції або в робочому просторі.

Виклик функції відбувається подібно іншим мовам програмування :

```
[F1 F2 … Fn] = ff(e1, e2, …, en);
ff(e1, e2, …, en);
```

Тут через `F1 F2 … Fn` позначено змінні для отримання результатів, а `e1, e2, …, en` – вирази, які є *фактичними* параметрами.

При першому зверненні до зовнішньої функції система MATLAB виконує її пошук за іменем М-файла на диску. Після того, як функцію знайдено, виконується її синтаксичний аналіз. У випадку знайденої помилки виводиться повідомлення в командне вікно, а при успішному завершенні аналізу створюється псевдокод, який завантажується в робочий простір і виконується. При повторному зверненні до функції вона буде знайдена в робочому просторі, а отже синтаксичний аналіз не проводиться, і функція виконується швидше. Видалити псевдокод із робочого простору можна командою

```
clear  ім'я_функції
```

Допускається використання функцій і змінних з однаковими іменами. Але тоді потрібно враховувати наступний порядок вибору системою MATLAB таких імен:

1. ім'я розшукується в поточному робочому середовищі (змінна з цим ім'ям, вкладена функція, анонімна функція або *inline*-функція);
2. розшукується підфункція в поточному М–файлі;

3. розшукується *приватна* функція. Приватні функції розміщені в директорії з ім'ям `private` і доступні тільки із функцій, розміщених у ній або у її батьківській директорії. Ця директорія не вказується в шляху доступу;
4. розшукується вбудована функція MATLAB;
5. виконується пошук М-функції в поточній директорії і шляхах доступу MATLAB.

Для визначення статусу імені *name* (змінної, функції) потрібно використати функцію *exist('name')*. Вона повертає, зокрема, такі числові значення:

- 0 – ім'я відсутнє;
- 1 – ім'я використовується як змінна робочого середовища;
- 2 – функція розташована в шляху пошуку файлів;
- 5 – функція є вбудованою (приватні функції і підфункції не ідентифікуються!);
- 7 – ім'я є директорією.

Повний перелік можливостей цієї функції можна отримати за допомогою команди *help exist*.

MATLAB має можливості використовувати імена функцій як аргументи (процедурний тип даних). При описі функції такі імена записуються звичайним способом, а при виклику – до фактичного імені додається дескриптор (*@*) або записується ім'я у вигляді текстової константи, наприклад, опис деякої функції *f*:

```
function […] = f(…,fun,…)
```

Тоді

```
e=Expr(fun(…),…);
```

– виклик функції *fun* (…) у складі виразу *Expr*;

```
E=Expr1(f(…,@myfun,…),…);
```

або

```
E=Expr1(f(…,'myfun',…),…);
```

– виклик функції *f* у складі виразу *Expr1*, де в якості фактичного параметра, який відповідає *fun*, використовуємо ім'я *myfun*.

Довільна функція MATLAB, наприклад, з ім'ям *myfun*, крім звичайного виклику *myfun(x1,x2,…,xn)*, може бути викликана вбудованою функцією *feval*. Перший фактичний вхідний аргумент має бути ім'ям *myfun* (у вигляді *'myfun'* або *@myfun*), а наступні фактичні вхідні аргументи – це перелік *x1,x2,…,xn* (через кому), наприклад,

```
a = myfun(x, y, z);
a = feval('myfun', x, y, z);
a = feval(@myfun, x, y, z);
```

Зазначимо відмінність функцій *eval* і *feval*. Перша приймає рядок з записаним виразом, який необхідно обчислити, а друга – ім'я функції у вигляді рядка, значення якої необхідно обчислити.

Приклад 15. Для кращого розуміння особливостей використання анонімних і *inline*-функцій при зверненні до функції *feval*, а також у тілі зовнішньої М-функції і у тілі М-файла, розглянемо М-функцію *p17_3a* і М-файл *p17_3b*, тіло яких складають однакові команди, проте робота деяких з цих команд відбувається по-різному. Потрібно звернути увагу на пріоритет використання імен функцій в М-функції і М-файлі, а також на форму використання параметра-функції в підфункції *P1*

і в зовнішній функції *P1*. В коментарях операторів указано, що форма запису деяких з них має недопустимий синтаксис (можливо в даній версії MATLAB), а деякі з операторів виконують звернення до зовнішніх функцій за наявності однакових імен в робочій області. Підсумовуючи, можна зробити висновок про необхідність ретельного тестування програм, які створює користувач.

```
function p17_3a      % зовнішня функція
clear all, close all, clc
f1 = @(x)(x.*x);      % анонімна функція
f2 = inline('2.*x');  % inline-функція
f1(5), feval(f1, 5)

% для виклику у вигляді feval(@f1, 5) - синтаксична помилка !
feval('f1', 5)        % у директорії пошуку є М-функція f1.m
feval(@sin, pi), feval('sin', pi)
f2(5), feval(f2, 5)

% для виклику в виді feval(@f2, 5) – синтаксична помилка !
feval('f2', 5)        % у директорії пошуку є М-функція f2.m
P1(f1, 1)             % виклик підфункції
%  P1(@f1, 2)  -  синтаксична помилка !
%  P1('f1', 3) -  виклик зі зверненням до М-функції f1.m !
P1(f2, 1)             % виклик підфункції
%  P1(@f2, 2)% -  синтаксична помилка !
%  P1('f2', 3) -  виклик зі зверненням до М-функції f2.m !
end

function e = P1(f,x)   % опис  підфункції
   e = f(x);           % виклик функції, заданої вхідним аргументом
end

%% М-файл p17_3b.m
clear all, close all, clc
f1 = @(x)(x.*x);      % анонімна функція
f2 = inline('2.*x');  % inline-функція
f1(5), feval(f1, 5)

% для виклику у вигляді feval(@f1, 5) - синтаксична помилка !
feval('f1', 5)        % у директорії пошуку є М-функція f1.m
feval(@sin, pi), feval('sin', pi)
f2(5), feval(f2, 5)

% для виклику у вигляді feval(@f2, 5) - синтаксична помилка !
feval('f2', 5)        % у директорії пошуку є М-функція f2.m
%  P1(f1, 1)   -  помилка при виклику зовнішньої функції
%  P1(@f1, 2)  -  синтаксична помилка !
%  P1('f1', 3) -  виклик зі зверненням до М-функція f1.m !
P1(f2, 1)             % виклик зовнішньої функції
%  P1(@f2, 2)  -  синтаксична помилка !
%  P1('f2', 3) -  виклик зі зверненням до М-функція f2.m !

function e = P1(f,x)   %  зовнішня функція
   e = f(x);   % виклик функції, заданої вхідним аргументом
end
```

Для опису *рекурсивної* функції (яка прямо або непрямо викликає сама себе) в MATLAB відсутні якісь спеціальні позначення і її виклик записується звичайним чином.

Приклад 16. Для рівняння $x=\sqrt[5]{x^2/3+20}$ знайти наближений розв'язок, використовуючи ітераційний метод Ейткена-Стеффенсена (написати відповідну зовнішню М-функцію).

Маємо типову ситуацію, при якій необхідно реалізувати діалогове введення даних для пошуку області розташування коренів рівняння $f(x)\equiv x-\sqrt[5]{x^2/3+20}=0$, а потім реалізувати вказаний ітераційний метод для рівняння виду $x=\varphi(x)$, де $\varphi(x)=\sqrt[5]{x^2/3+20}$. При реалізації метода Ейткена-Стеффенсена у вигляді зовнішньої функції (один раз написали і в подальшому можемо її використовувати!) потрібно продумати її інтерфейс, який задовольнить найвибагливішого користувача (з огляду набору вхідних і вихідних параметрів). Також потрібно подбати і про можливі помилки при написанні виклику цієї функції.

Нижче наведено текст програми і результати її роботи.

```
function pl7_4
%% PL7_4 Приклади опису і виклику функцій
clear all, close all, clc
x0 = search_region_root(@f);                % виклик підфункції
disp('Обчислення кореня методом Ейткена-Стеффенсена');
% 1) виклик зовнішньої функції зі "стандартним" набором
%    вихідних і вхідних параметрів
ep = 1e-8; x = x0; [x, it] = m_estef1(@fi, x, ep, 500);
disp('Виклик:  [x, it] = m_estef1(@fi, x, ep, 500)')
disp(strcat('Корінь=', num2str(x), ' ітерацій=', int2str(it),...
            ' точність=', num2str(ep)))
% 2) виклик зовнішньої функції зі "стандартним" набором
%    вихідних і скороченим вхідних параметрів
x = x0; [x, it] = m_estef1(@fi, x, ep);
disp('Виклик:  [x, it] = m_estef1(@fi, x, ep)')
disp(strcat('Корінь=', num2str(x), ' ітерацій=', int2str(it),...
            ' точність=', num2str(ep)))
% 3) виклик зовнішньої функції з мінімальним набором
%    вихідних і вхідних параметрів
x = x0; x = m_estef1(@fi, x);
disp('Виклик:  x = m_estef1(@fi, x)')
disp(strcat('Корінь=', num2str(x)))
% 4) виклик зовнішньої функції без вихідних
%    і з мінімальним набором вхідних параметрів
x = x0; m_estef1(@fi, x);
disp('Виклик:  m_estef1(@fi, x)')
disp(strcat('Корінь=', num2str(ans)))
% 5) виклик зовнішньої функції з "розширеним" набором
%    вихідних і мінімальним набором вхідних параметрів
x = x0*10; [x, it, e_tol, er, ou] = m_estef1(@fi, x);
disp('Виклик:  [x, it, e_tol, er, ou] = m_estef1(@fi, x)')
disp(strcat('Корінь=', num2str(x), ' ітерацій=', int2str(it),...
            ' точність=', num2str(e_tol),...
            ' Error=', int2str(er), ' ou=', int2str(ou)))
end

function x0 = search_region_root(f)       % підфункція
% Графічний пошук області існування кореня рівняння f(x)=0
% і завдання початкового наближення до кореня x0
e = 1;
```

```
while e
   a = input('? a='); b = input('? b='); h = input('? h=');
   X = [a : h : b]; plot(X, f(X)), grid on
   e = input('? продовжимо дослідження областей з коренями (1-так|0-ні) =');
end
x0 = input('? Початкове наближення=');
end

function e = fi(x);                          % підфункція
% завдання правої частини рівняння x = fi(x)
e = (x .* x / 3  + 20).^(1/5);
end

function e = f(x);                           % підфункція
% завдання функції f(x) = x-fi(x)
e = x - fi(x);
end

function [x, varargout] = m_estef1(f, x, varargin)
%% M_ESTEF1 м.Ейткена-Стеффенсена обчислення кореня x = f(x)
%  Метод Ейткена-Стеффенсена знаходження кореня
%  рівняння виду x = f(x)
% Вхідні параметри :
%  f - функція від скалярної змінної f(x);
%  x - початкове наближення;
%  varargin - список змінних параметрів:
%               e - точність (за замовчуванням =eps);
%             mki - max кількість ітерацій (mki=1000);
% Вихідні параметри :
%  x - останнє значення наближення;
%  varargout - список змінних параметрів:
%               k - кількість виконаних ітерацій;
%               e - використана точність;
%             err - true (|x(k)-x(k-1)|>e або k>mki
%                   false (помилка відсутня);
%          решта записаних вихідних параметрів = NaN

% перевірка кількості вхідних параметрів
if nargin < 2
   error('M_ESTEF1: Мало вхідних параметрів !');
   return
end
k = length(varargin);
switch k                  % визначення вхідних змінних параметрів
    case 0
        e = eps; mki = 1000;
    case 1
        e = varargin{1}; mki = 1000;
    otherwise  % k >= 2
        e = varargin{1}; mki = varargin{2};
end
k = 0; err = true;
% перевірка вхідних даних e і mki
if 0 >= e | e >= 1 | mki < 1
   return
end
```

```
% реалізація ітераційного процесу
while k < mki
   x1 = f(x); x2 = f(x1);
   x = (x .* x2 - x1 .* x1)./(x - 2 .* x1 + x2); k = k + 3;
   if abs(x2 - x) < e
      err = false; break
   end
end
% визначення вихідних змінних параметрів
if nargout > 1
   varargout(1)={k};     % кількість ітерацій
   if nargout > 2
      varargout(2)={e};  % точність
      if nargout > 3
         varargout(3)={err};
         if nargout > 4
            varargout(4:nargout)={NaN};
         end
      end
   end
end
```

Результат виконання М-функції *p17_3*:

```
? a=-2
? b=4
? h=0.1
? продовжимо дослідження областей з коренями (1-так|0-ні) =0
? Початкове наближення=1.8
Обчислення кореня методом Ейткена-Стеффенсена
Виклик:  [x, it] = m_estef1(@fi, x, ep, 500)
Корінь=1.8407 ітерацій=6 точність=1e-008
Виклик:  [x, it] = m_estef1(@fi, x, ep)
Корінь=1.8407 ітерацій=6 точність=1e-008
Виклик:  x = m_estef1(@fi, x)
Корінь=1.8407
Виклик:  m_estef1(@fi, x)
Корінь=1.8407
Виклик:  [x, it, e_tol, er, ou] = m_estef1(@fi, x)
Корінь=1.8406 ітерацій=1002 точність=2.2204e-016 Error=1 ou=NaN
```

ТЕМА 8
ІТЕРАЦІЙНІ МЕТОДИ РОЗВ'ЯЗАННЯ СЛАР ТА НЕЛІНІЙНИХ РІВНЯНЬ І СИСТЕМ

При розв'язанні СЛАР великої розмірності прямі методи стають неефективними. У таких випадках доцільніше застосовувати наближені ітераційні методи [3,11,16, 19].

Ітераційні методи розв'язання СЛАР

Нехай маємо СЛАР

$$A\vec{x} = \vec{f}, \tag{8.1}$$

де A має обернену матрицю A^{-1}.

Представимо матрицю A у вигляді $A = A_1 + D + A_2$, де $D = diag\,[a_{11}, \dots, a_{nn}]$, A_1 – нижня трикутна, A_2 – верхня трикутна матриці.

$$(A_1 + D + A_2)\vec{x} = \vec{f},$$
$$D\vec{x} = -(A_1 + A_2)\vec{x} + \vec{f},$$
$$\vec{x} = -D^{-1}(A_1 + A_2)\vec{x} + D^{-1}\vec{f}\,. \tag{8.2}$$

Метод Якобі. Схема методу :

$$\vec{x}^{k+1} = -D^{-1}(A_1 + A_2)\vec{x}^{(k)} + D^{-1}\vec{f}. \tag{8.3}$$

Якщо $A = A'$, $A > 0$ і $|a_{ii}| > \sum_{j=u\neq i}^{n} |a_{ij}|$ $i = \overline{1, n}$, то ітераційний процес (8.3) збігається.

Метод Зейделя. Схема методу:

$$\vec{x}^{(k+1)} = -D^{-1}A_1\vec{x}^{(k+1)} - D^{-1}A_2\vec{x}^{(k)} + D^{-1}\vec{f}\,. \tag{8.4}$$

Умова закінчення цих ІП :

$$\left\|\vec{x}^{(k+1)} - \vec{x}^{(k)}\right\|_c < \varepsilon\,.$$

Канонічна форма однокрокових ітераційних методів має вигляд:

$$B_{(k+1)}\frac{\vec{x}^{(k+1)} - \vec{x}^{(k)}}{\tau_{k+1}} + A\vec{x}^{(k)} = \vec{f}\,. \tag{8.5}$$

Якщо $B_{(k+1)} = E$, то ітераційний процес (8.5) називають *явним*, інакше – *неявним*. Якщо $B_{(k+1)} = B$, то ітераційний процес називається с*таціонарним.*

Метод простої ітерації. Схема методу

$$\frac{\vec{x}^{(k+1)} - \vec{x}^{(k)}}{\tau} + A\vec{x}^{(k)} = \vec{f}\,. \tag{8.6}$$

Метод Річардсона. Схема методу

$$\frac{\vec{x}^{(k+1)} - \vec{x}^{(k)}}{\tau_{k+1}} + A\vec{x}^{(k)} = \vec{f}\,. \tag{8.7}$$

Якщо $A = A'$, $A > 0$, то відомі алгоритми побудови оптимальних ітераційних параметрів для (8.6), (8.7), які базуються на значеннях границь спектра власних значень матриці СЛАР. Для (8.6) – це *оптимальний лінійний ІП*, а для (8.7) – *метод Річардсона з чебишовським набором параметрів*.

Метод релаксації (узагальнення методу Зейделя). Схема методу

$$(D+\omega A_1)\frac{\vec{x}^{(k+1)}-\vec{x}^{(k)}}{\omega}+A\vec{x}^{(k)}=\vec{f}\,. \tag{8.8}$$

При $0<\omega<1$ (8.8) називають методом нижньої релаксації, при $1<\omega<2$ – методом верхньої релаксації, при $\omega=1$ – методом Зейделя.

Якщо $A=A'$, $A>0$, то (8.8) збігається при $0<\omega<2$.

Запишемо (8.8) у покомпонентній формі:

$$\begin{cases} x_i^{(k+1)}=(1-\omega)x_i^{(k)}+\omega\dfrac{f_i}{a_{ii}}-\omega\left(\sum\limits_{j=1}^{i-1}\dfrac{a_{ij}}{a_{ii}}x_j^{(k+1)}+\sum\limits_{j=i+1}^{n}\dfrac{a_{ij}}{a_{ii}}x_j^{(k)}\right), i=\overline{1,n;} \\ k=0,1,\ldots, \left\|\vec{x}^{(k+1)}-\vec{x}^{(k)}\right\|_c<\varepsilon. \end{cases} \tag{8.9}$$

Методи побудовані на основі варіаційних підходів. Нехай в СЛАР (8.1) $A=A'$, $A>0$. Розглянемо стаціонарний ітераційний процес у вигляді (8.5) з $B=E$. Позначимо для кроку k нев'язку через

$$\vec{r}^{(k)}=A\vec{x}^{(k)}-\vec{f}\,, \tag{8.10}$$

тоді (8.5) можна переписати у вигляді

$$\vec{x}^{(k+1)}=\vec{x}^{(k)}-\tau_{k+1}\vec{r}^{(k)}\,. \tag{8.11}$$

Методом мінімальних нев'язок (ММН) називається ітераційний процес (8.11), в якому τ_{k+1} вибирається з умови $\min\left\|\vec{r}^{(k+1)}\right\|$ при заданому $\left\|\vec{r}^{(k)}\right\|$.

Помножимо (8.11) на A і віднімемо з обох частинах рівності $\vec{f}$:

$$A\vec{x}^{(k+1)}-\vec{f}=A\vec{x}^{(k)}-\vec{f}-\tau_{k+1}A\vec{r}^{(k)}, \quad \vec{r}^{(k+1)}=\vec{r}^{(k)}-\tau_{k+1}A\vec{r}^{(k)},$$

$$\left\|\vec{r}^{(k+1)}\right\|^2=(\vec{r}^{(k)}-\tau_{k+1}A\vec{r}^{(k)},\ \vec{r}^{(k)}-\tau_{k+1}A\vec{r}^{(k)})=\left\|\vec{r}^{(k)}\right\|^2-2\tau_{k+1}(\vec{r}_{(k)},A\vec{r}^{(k)})+\tau_{k+1}^2\left\|A\vec{r}^{(k)}\right\|^2.$$

Звідси бачимо, що $\left\|r^{(k+1)}\right\|$ досягає мінімуму, якщо

$$\tau_{k+1}=\frac{(A\vec{r}^{(k)},\vec{r}^{(k)})}{\left\|A\vec{r}^{(k)}\right\|^2}\,. \tag{8.12}$$

Алгоритм переходу від наближення $\vec{x}^{(k)}$ до наступного :

$$\begin{aligned} &\vec{r}^{(k)}=A\vec{x}^{(k)}-\vec{f}, \quad \vec{y}=A\vec{r}^{(k)}, \\ &\tau_{k+1}=\frac{(\vec{y},\vec{r}^{(k)})}{(\vec{y},\vec{y})}, \quad \vec{x}^{(k+1)}=\vec{x}^{(k)}-\tau_{k+1}\vec{r}^{(k)}. \end{aligned} \tag{8.13}$$

Метод найшвидшого спуску (МНС) відрізняється від ММН тільки формулою обчислення τ_{k+1}. Це значення знаходять з умови $\min\left\|\vec{z}^{(k+1)}\right\|_A$ при заданому $\vec{z}^{(k)}=\vec{x}^{(k)}-\vec{x}^{точ}$, що еквівалентно ортогональності нев'язок $\vec{r}^{(k)},\vec{r}^{(k+1)}$. Помножимо (8.11) скалярно на $\vec{r}^{(k)}$, отримаємо

$$0=(\vec{r}^{(k)},\vec{r}^{(k)})-\tau_{k+1}(A\vec{r}^{(k)},\vec{r}^{(k)})\,.$$

Звідси

$$\tau_{k+1}=\frac{(\vec{r}^{(k)},\vec{r}^{(k)})}{(A\vec{r}^{(k)},\vec{r}^{(k)})}\,. \tag{8.14}$$

Швидкість збіжності ММН, МНС така, як і у процеса (8.6) з оптимальним параметром τ.

Приклад 17. Розв'язати задану СЛАР різними ітераційними методами.

Нижче наведено текст відповідного М-файла і результати його роботи.

```
%%p18_1.m
% Приклади реалізації і використання ітераційних процесів
clear all, close all, clc
n = 10;

% Створюємо деяку матрицю
A = ones(n); x = logical(eye(n)); A(x) = [11:5:56];
b = ones(n,1);   %  Створюємо вектор правих частин
ep = 1e-10; mki = 10000;  %  Точність і max кількість ітерацій
%  Розв'язання СЛАР Ax=b

% Матричне ліве ділення
x = A \ b;

% Ітераційний метод Зейделя
xz = b; [xz, kz] = m_zejdel(A, b, xz, ep, mki);

% Ітераційний метод верхньої релаксації
w = 1.0125;    %  параметр релаксації
xr = b; [xr, kr] = m_relaxv(A, b, w, xr, ep, mki);

% Ітераційний метод мінімальних нев'язок
xn = b; [xn, kn] = m_minn(A, b, xn, ep, mki);

disp(' Кількість ітерацій :')
disp(strcat('м.Зейделя         =',int2str(kz)))
disp(strcat('м.верхн.релаксації=',int2str(kr)))
disp(strcat('м.min нев''язок   =',int2str(kn)))
disp('Розв''язки :')
S = sprintf('%c%c%c%c',' матр.лів.діл \','   метод Зейделя',...
                       ' верх.релаксації',' мінім. нев''язок');
disp(S)
for k = 1 : n
    S = sprintf('%16.6e%16.6e%16.6e%16.6e',x(k),xz(k),xr(k),xn(k));
    disp(S)
end
[S,er] = sprintf('%s%16.6e%16.6e%16.6e','Абсолют.похибка:',...
            norm(x-xz), norm(x-xr), norm(x-xn));
disp(S)

function [x, k] = m_zejdel(A, f, x, e, mki)
%%M_ZEJDEL   Метод Зейделя.
% Ітераційний метод Зейделя розв'язання СЛАР A*x=f
%  (реалізація матрично-векторної форми)
n = length(A); D = diag(A); D = 1./D; D = diag(D);
DA1 = -D * tril(A, -1); DA2 = -D * triu(A, 1); Df = D * f;
clear D, xn = x;
for k = 1 : mki
   xn = DA1 * xn + DA2 * x + Df;
   t = true; i = 0;
   while t & i < n
      i = i + 1; t = abs(xn(i) - x(i)) < e;
   end
   x = xn;
   if t
      break
   end
```

```
   end
   return

   function [x, k] = m_relaxv(A, f, w, x, e, mki)
   %%M_RELAXV      Метод релаксації.
   % Розв'язання СЛАР A*x=f ітераційним методом релаксації
   %    (матрично-векторна форма)
   A1 = tril(A, -1); O = diag(diag(A));
   O = inv(O + w .* A1);
   for k = 1 : mki
      r = w .* (f - A * x);
      r = O * r; x = x + r;
      if norm(r, inf) < e
         break
      end
   end
   return

   function [x, k] = m_minn(A, f, x, e, mki)
   %% M_MINN       Ітераційний метод min нев'язок для СЛАР
   % Розв'язання СЛАР  A*x=f ітераційним методом min нев'язок
   n = length(A);
   for k = 1 : mki
      r = A * x - f; y = A * r;
      t = dot(y, r) ./ dot(y, y); r = t .* r;
      t = true; i = 0;
      while t & i < n
         i = i + 1; t = abs(r(i)) < e;
      end
      x = x - r;
      if t
         break
      end
   end
   return
```

Результат виконання М-файла *p18_1.m*:

```
Кількість ітерацій :
м.Зейделя         =23
м.верхн.релаксації=11
м.min  нев'язок   =59
Розв'язки :
   матр.лів.діл \  метод Зейделя верх.релаксації мінім. нев'язок
   7.122632e-002   7.122632e-002   7.122632e-002   7.122632e-002
   4.748421e-002   4.748421e-002   4.748421e-002   4.748421e-002
   3.561316e-002   3.561316e-002   3.561316e-002   3.561316e-002
   2.849053e-002   2.849053e-002   2.849053e-002   2.849053e-002
   2.374211e-002   2.374211e-002   2.374211e-002   2.374211e-002
   2.035038e-002   2.035038e-002   2.035038e-002   2.035038e-002
   1.780658e-002   1.780658e-002   1.780658e-002   1.780658e-002
   1.582807e-002   1.582807e-002   1.582807e-002   1.582807e-002
   1.424526e-002   1.424526e-002   1.424526e-002   1.424526e-002
   1.295024e-002   1.295024e-002   1.295024e-002   1.295024e-002
Абсолют.похибка:   3.907242e-011   3.912723e-012   1.564702e-010
```

Ітераційні методи розв'язання нелінійних рівнянь

Знаходження коренів нелінійного рівняння

$$f(x)=0 \tag{8.15}$$

здійснюється у два етапи.

На першому етапі досліджується розташування коренів (у загальному випадку на комплексній площині) і визначаються області, в яких існує тільки один корінь і його кратність. Це дає нам початкове наближення до вибраного кореня.

На другому етапі в залежності від характеристик функції $f(x)$ будується певний ІП уточнення кореня.

У загальному вигляді ІП можна записати так

$$x^{(k+1)}=\Phi_{(k)}\left(x^{(k)},x^{(k-1)},...,x^{(k-m+1)}\right),\quad k=0,1,... \ . \tag{8.16}$$

Для характеристики ІП (8.16) вводять такі поняття:

- *p-й порядок методу* – якщо виконується рівність $r^{(k+1)}=C\cdot(r^{(k)})^p$, де $C=const$, $r^{(k)}=\left|x^{моч}-x^{(k)}\right|$ – похибка;
- *стаціонарний метод* – якщо $\Phi_{(k)}$ не залежить від *k*, інакше – *нестаціонарний*;
- *m-кроковий* $(m\geq 1)$ – якщо для отримання чергового елементу послідовності використовують *m* попередніх.

Метод простої ітерації Спочатку рівняння (8.15) приводиться до еквівалентного

$$x=s(x) \tag{8.17}$$

і ІП проводиться за правилом

$$x^{(k+1)}=s(x^{(k)}),\quad k=0,1,... \ . \tag{8.18}$$

Функція $s(x)$ зазвичай вибирається у вигляді

$$s(x)=x+\tau(x)f(x), \tag{8.19}$$

причому функція $\tau(x)$ не повинна змінювати знак в області пошуку кореня $U_r\left(x^{моч}\right)=\left\{x:\left|x-x^{моч}\right|\leq r\right\}$, а для $s(x)$ має виконуватись умова $\left|s'(x)\right|\leq q<1$ в області $U_r\left(x^{моч}\right)$.

Якщо $\tau(x)=\tau=const$, то отримаємо **метод релаксації**:

$$x^{(k+1)}=x^{(k)}+\tau\cdot f(x^{(k)}), \tag{8.20}$$

який збігається при умові $-2<\tau\cdot f'\left(x^{моч}\right)<0$.

Якщо в області $U_r\left(x^{моч}\right)$ виконуються умови $f'(x)<0\ \&\ 0<m_1<\left|f'(x)\right|<M_1$, то (8.20) збігається при значеннях $\tau\in\left(0,\ 2/M_1\right)$ і для оптимального значення маємо $\tau_{опт}=\dfrac{2}{m_1+M_1}$.

Ураховуючи, що для збіжного ІП мають місце нерівності :

$$\left|x^{моч}-x^{(k)}\right|\leq\frac{q}{1-q}\left|x^{(k)}-x^{(k-1)}\right|,\qquad \left|x^{моч}-x^{(k)}\right|\leq\frac{q^k}{1-q}\left|x^{(1)}-x^{(0)}\right|,$$

умову закінчення ІП вибирають у вигляді $\left|x^{(k+1)}-x^{(k)}\right|\leq\varepsilon,\quad 0<\varepsilon<1$.

Ітераційні процеси (8.18), (8.20) мають перший порядок (збігаються лінійно зі швидкістю геометричної прогресії зі знаменником *q*).

На основі лінійного ІП можна побудувати нестаціонарний ІП, який має квадратичну збіжність. Це так званий **процес Ейткена-Стеффенсена**. Його алгоритм: за відомим $x^{(k)}$ циклічно $k = 0,3,6,...$, виконуємо наступні розрахунки

$$\begin{cases} x^{(k+1)} = s(x^{(k)}), \quad x^{(k+2)} = s(x^{(k+1)}), \\ x^{(k+3)} = \dfrac{x^{(k)}x^{(k+2)} - (x^{(k+1)})^2}{x^{(k)} - 2x^{(k+1)} + x^{(k+2)}}. \end{cases} \tag{8.20}$$

Метод Ньютона (лінеаризації) Нехай функція $f(x)$ рівняння (8.15) двічі неперервно диференційована, тоді ІП

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})} \tag{8.21}$$

збігається до кореня (8.15) при $\forall \ \ x^{(0)}$, якщо скрізь $|f(x)\cdot f''(x)| < (f'(x))^2$.

Якщо $f'(x)$ і $f''(x)$ зберігають знак на $\Omega = U_r(x^{моч})$, а також виконується умова $f(x^{(0)})\cdot f''(x^{(0)}) \ge 0$, то ІП (8.21) збігається до кореня $x^{моч}$ монотонно і для похибки маємо нерівність $\left|x^{моч} - x^{(k)}\right| \le \dfrac{M_2}{2m_1}\left|x^{(k)} - x^{(k-1)}\right|^2$, $m_1 = \min\limits_{\Omega}|f'(x)|$, $M_2 = \max\limits_{\Omega}|f''(x)|$. ІП (8.21) має 2-й порядок. Виходячи із геометричних побудов, ІП (8.21) має назву ***методу дотичних***.

Для кореня $x^{моч}$ кратності *n*, ІП (8.21) записують у формі

$$f'(x^{(k)})\frac{x^{(k+1)} - x^{(k)}}{n} + f(x^{(k)}) = 0.$$

При заміні у (8.21) похідної на поділену різницю отримаємо двокроковий ІП за схемою **методу січних** :

$$x^{(k+1)} = x^{(k)} - \frac{\left(x^{(k)} - x^{(k-1)}\right)f(x^{(k)})}{f(x^{(k)}) - f(x^{(k-1)})} \tag{8.22}$$

та **методу Курчатова** :

$$x^{(k+1)} = x^{(k)} - \frac{2\left(x^{(k)} - x^{(k-1)}\right)f(x^{(k)})}{f(2x^{(k)} - x^{(k-1)}) - f(x^{(k-1)})}. \tag{8.23}$$

ІП (8.22), (8.23) мають порядок $p \approx 1.62$ і не вимагають обчислення похідної.

Ітераційні методи розв'язання систем нелінійних рівнянь

Знаходження коренів системи *N* нелінійних рівнянь

$$\vec{f}(\vec{x}) = 0 \tag{8.24}$$

за допомогою ітераційних методів проводиться як і для одного рівняння (8.15).

Метод простої ітерації. Спочатку система (8.24) приводиться до еквівалентної

$$\vec{x} = \vec{s}(\vec{x}) \tag{8.25}$$

і ІП проводиться за правилом

$$\vec{x}^{(k+1)} = \vec{s}(\vec{x}^{(k)}), \quad k = 0,1,... \ . \tag{8.26}$$

Вектор-функцію $\vec{s}(\vec{x})$ потрібно вибрати так, щоб для матриці $S = \left[S_{i,j}\right]_{i,j=1}^{N}$ з елементами $S_{i,j} = \dfrac{\partial s_i(x)}{\partial x_j}$ в околі $U_r(\vec{x}^{моч})$ виконувалась умова $\|S\|_* \le q < 1$. Умову

закінчення ІП (8.26) потрібно перевіряти для кожної компоненти: $\left|x_i^{(k+1)} - x_i^{(k)}\right| \le \varepsilon,\ \ \mathrm{i} = 1,2,...,\mathrm{N};\quad 0 < \varepsilon < 1$.

Метод простої ітерації в поєднанні з методом Зейделя. ІП проводиться за правилом

$$\begin{cases} x_1^{(k+1)} = s_1(x_1^{(k)}, x_2^{(k)}, ..., x_N^{(k)}), \\ x_2^{(k+1)} = s_2(x_1^{(k+1)}, x_2^{(k)}, ..., x_N^{(k)}), \\ ... \\ x_N^{(k+1)} = s_N(x_1^{(k+1)}, x_2^{(k+1)}, ..., x_{N-1}^{(k+1)}, x_N^{(k)}), \\ k = 0,1,... \end{cases} \tag{8.27}$$

Нелінійний метод Зейделя. ІП проводиться за правилом

$$\begin{cases} f_i(x_1^{(k+1)}, x_2^{(k+1)}, ..., x_i^{(k+1)}, x_{i+1}^{(k)}, ..., x_N^{(k)}) = 0, \\ i = 1,2,...,N; \\ k = 0,1,... \end{cases} \tag{8.28}$$

Метод Ньютона. ІП проводиться за правилом (у векторній формі)

$$F(\vec{x}^{(k)}) \cdot \vec{\delta} = -\vec{f}(\vec{x}^{(k)}),\quad \vec{x}^{(k+1)} = \vec{x}^{(k)} + \vec{\delta},\quad \mathrm{k} = 0,1,..., \tag{8.29}$$

де матриця $F(\vec{x}) = \left[\frac{\partial f_i(\vec{x})}{\partial x_j}\right]_{i,j=1}^{N}$, а умовою закінчення є $\left\|\vec{x}^{(k+1)} - \vec{x}^{(k)}\right\|_* \le \varepsilon,\quad 0 < \varepsilon < 1$.

Також використовують *гібридні* методи, коли зовнішні ітерації виконують одним методом, а внутрішні – іншим. Наприклад, *зовнішні* за Зейделем, а *внутрішні* - за Ньютоном: використовуємо (зовнішній) ІП (8.28), а (внутрішній) для знаходження компоненти $x_i^{(k+1)}$.

В пакеті MATLAB відсутня пряма реалізація розглянутих методів, але є функції, що реалізують низку методів оптимізації і розв'язання нелінійних рівнянь:

Функції мінімізації функцій і знаходження коренів	
Назва	**Призначення**
fmin('f',xn, xk); *fminbnd('f',xn, xk)*	знаходження мінімуму функції однієї змінної на інтервалі [x_n, x_k]; знаходження мінімуму функції однієї змінної на інтервалі [x_n, x_k] (у *V7.6*)
fmins('f',x0); *fminsearh('f',x0)*	знаходження мінімуму функції декількох змінних в околі точки x_0; знаходження мінімуму функції декількох змінних в околі точки x_0 (у *V7.6*)
fzero('f',x)	знаходження нуля функції поблизу точки *x*, наприклад: *fzero('x^3–27',1)*

Приклад 18. Для рівняння з поліномом у лівій частині ($f(x) \equiv x^4 - 4x^2 + 5x - 20 = 0$) знайти наближений розв'язок, застосовуючи метод Ньютона (зовнішня функція `m_njut1.m`), і порівняти результат з отриманим при використанні функцій `fzero` і `roots`.

Наведена нижче М-функція реалізує графічне дослідження області розташування коренів і діалогове введення початкового наближення при виконанні програми.

```
function p18_2
% Приклади знаходження коренів нелінійного рівняння
clear all, close all, clc
global F dF
F = [1, 0, -4, 5, -20];  % поліном (ліва частина рівняння)
```

```
dF = polyder(F);          % визначення похідної від полінома
e = 1;
while e
    a = input('? a='); b = input('? b='); h = input('? h=');
    X = [a : h : b]; plot(X, f(X)); grid on
    e = input('? продовжимо шукати область коренів (1-так|0-ні) =');
end
close all
x0 = input('? Початкове наближення=');
e = 1e-10;
[x, k] = m_njut1(@fdf, x0, e, 1000);
disp('Метод Ньютона :');
disp(strcat('ітерацій=',num2str(k),' точність=',num2str(e),...
            ' корінь=',num2str(x)))
x = fzero(@f, x0);
disp(strcat('За допомогою fzero=',num2str(x)))
R = sort(roots(F));
disp('Корені полінома :')
format long e
disp(R)

function s = f(x)      %  підфункція функції p18_2
global F
s = polyval(F, x);

function s = fdf(x)    %  підфункція функції p18_2
global F dF
s(1) = f(x);
s(2) = polyval(dF, x);

function [x, k] = m_njut1(fdf, x, e, mki)
%% M_NJUT1   м.Ньютона для знаходження кореня  f(x)=0.
% Метод Ньютона для знаходження кореня  f(x)=0
for k = 1 : mki
   o = fdf(x);  o = o(1)./o(2); x = x - o;
   if abs(o) < e
      break
   end
end
```

Результат виконання М-функції *p18_2.m*:

```
? a=-5
? b=5
? h=0.2
? продовжимо шукати область коренів (1-так | 0-ні) =0
? Початкове наближення=2.3
Метод Ньютона :
ітерацій=4 точність=1e-010 корінь=2.3458
За допомогою fzero=2.3458
Корені полінома :
     2.579351133365094e-001 -1.706666508420220e+000i
     2.579351133365094e-001 +1.706666508420220e+000i
     2.345840899606892e+000
    -2.861711126279911e+000
```

Приклад 19. Знайти наближений розв'язок системи нелінійних рівнянь (СНР)

$$\begin{cases} \cos(0.4y + x^2) + x^2 + y^2 = 1.6, \\ 1.5x^2 - \dfrac{y^2}{0.36} = 1. \end{cases}$$

Використаємо в цій задачі ітераційний метод Ньютона для СНР (зовнішня функція *m_njuts.m*). Наведена нижче М-функція реалізує графічне дослідження області розташування коренів і діалогове введення вектора початкових наближень до коренів.

```
function p18_3
%% Приклади знаходження розв'язків СНР
clear all, close all, clc
ezplot('cos(0.4*y+x^2)+x^2+y^2-1.6',[-2,2]), hold on
ezplot('1.5*x^2-y^2/0.36-1',[-2,2]), grid on
X0 = input('? Початкове наближення=');
kn = size(X0); n = kn(1);
% Створення структури для зберігання коренів :
X = zeros(1, n); S = struct('x', X, 'y', X, 'kit', X);
e = 1e-10; mki = 1000;  % параметри ітераційного процесу
disp(strcat('Точність для м.Ньютона=', num2str(e),...
            ' Обмеж.кільк.іт=', int2str(mki)));
for k = 1 : n
   X = (X0(k, :))';
   [X, it] = m_njuts(X, e, mki, @F, @dF);
   S.x(k) = X(1); S.y(k) = X(2); S.kit(k) = it;
end
s1 = '            Розв''язок системи НР\n';
s2 = '----------x-------|----------y-------|--іт-\n';
s = strcat(s1, s2);
fprintf(s);
n = length(S.x);
for k = 1 : n
   fprintf('%17.8f |%17.8f |%5d\n', S.x(k), S.y(k), S.kit(k));
end

function f = F(X)       % підфункція функції p18_3
% Обчислення лівої частини НС F(X)
x2 = X(1).^2; y = X(2); y2 = y.*y;
f = [cos(0.4 .* y + x2) + x2 + y2 - 1.6;...
     1.5 .* x2 - y2 ./ 0.36 - 1];

function J = dF(X)      % підфункція функції p18_3
% Обчислення Якобіана від F(X)
x = X(1); x2 = x.*x; y = X(2); y2 = y.*y;
s = -sin(0.4 .* y + x2);
J = [2 .* x .* (1 + s), 2.*y + 0.4 .* s;...
                3 .* x, -y./0.18];


function [X, k] = m_njuts(X, e, mki, ffun, dffun)
%% M_NJUTS   м.Ньютона для знаходження коренів системи F(X)=0.
% Знаходження розв'язку нелінійної системи рівнянь F(X) = 0
% ітераційним методом Ньютона  (dFi(X(n)/dXj)(X(n+1)-X(n))-F(X(n)),
% де значення X(0) відоме.
% Умова закінчення ітерацій : norm(X(n+1)-X(n))<e.
% вхідні дані :
```

```
%   X      - початкове наближення до розв'язку;
%   e      - точність (0<e<1);
%   mki    - максимальна кількість кроків ітерацій;
%   ffun   - вектор-стовпець (функція F(X));
%   dffun  - матриця Якобі (dFi(X)/dXj).
% вихідні дані:
%   X      - останнє наближення до розв'язку;
%   k      - кількість виконаних ітерацій.

if nargin<5
  error('M_NJUTS:Not Enough Inputs',...
        'Not enough input arguments.  See M_NJUTS.');
  return
end
if e <= 0 | e >= 1
  error('M_NJUTS:error e <= 0 | e >= 1.',...
        'e <= 0 | e >= 1.  See M_NJUTS.');
  return
end
k = 0; F = true;
while F & k < mki
   k = k + 1; O = dffun(X) \ ffun(X);
   X = X - O; F = norm(O) > e;
End
```

Результат виконання М-функції *p18_3.m*:

```
? Початкове наближення=[-1.1,0.5;1.1,0.5;-0.9,-0.3;0.9,-0.3]
Точність для м.Ньютона=1e-010 Обмеж.кільк.іт=1000
            Розв'язок системи НР
----------x-------|----------y-------|--іт-
      -1.03862924 |       0.47172595 |    4
       1.03862924 |       0.47172595 |    4
      -0.87456548 |      -0.23027589 |    5
       0.87456548 |      -0.23027589 |    5
```

ТЕМА 9
АЛГОРИТМИ РОЗВ'ЯЗАННЯ ЗАДАЧІ КОШІ ДЛЯ ЗДР

Розглянемо задачу Коші для ЗДР

$$\begin{cases} \dfrac{d\vec{u}(x)}{dx} = \vec{f}(x,\vec{u}(x)), \quad x \in (x_0, x_{end}], \\ \vec{u}(x_0) = \vec{u}_0 \end{cases} \tag{9.1}$$

і для її наближеного розв'язку будемо використовувати *чисельні* методи:

a) *різницеві* (*однокрокові* і *багатокрокові*);
b) *типу Рунге-Кутта*.

Як перші, так і другі методи бувають *явними* і *неявними*.

Різницеві однокрокові методи. Розглянемо спочатку скалярний варіант (9.1). Вводимо за змінною x сітку з певним кроком (рівномірним або нерівномірним), виконуємо апроксимацію лівої і правої частини рівняння (9.1) на шаблоні $S[x_n, x_{n+1}]$.

Явний метод Ейлера. Схема методу

$$y_{n+1} = y_n + h_n \cdot f(x_n, y_n), \quad y_0 = u_0, \quad n = 0,1,\ldots . \tag{9.2}$$

Неявний метод Ейлера. Схема методу

$$y_{n+1} = y_n + h_n \cdot f(x_{n+1}, y_{n+1}), \quad y_0 = u_0, \quad n = 0,1,\ldots . \tag{9.3}$$

Симетрична схема. Схема методу

$$y_{n+1} = y_n + 0.5h_n \cdot (f(x_n, y_n) + f(x_{n+1}, y_{n+1})), \quad y_0 = u_0, \quad n = 0,1,\ldots . \tag{9.4}$$

Для автоматичного вибору кроку використовують метод *Рунге* (враховуючи, що методи (9.2)–(9.3) мають точність $O(h)$, а метод (9.4) – точність $O(h^2)$).

Методи типу Рунге-Кутта. Характерною особливістю цих методів є те, що обчислення проводяться не тільки в точках сітки, але і в деяких проміжних точках (використовуючи спеціальні квадратурні формули)

$$y_{n+1} - y_n = \int_{x_n}^{x_{n+1}} f(\xi, u(\xi))d\xi = h_n \sum_{i=1}^{m} A_i \cdot K_i, \quad y_0 = u_0, \quad n = 0,1,\ldots .$$

Сім'я методів Рунге-Кутта **2-го порядку точності** (m=2). Схема методу

$$\begin{aligned} &A_1 = 1-\gamma, \quad A_2 = \gamma, \quad \theta = 0.5h_n/\gamma, \\ &K_1 = f(x_n, y_n), \quad K_2 = f(x_n + \theta, y_n + \theta K_1), \quad \gamma \in (0,1]. \end{aligned} \tag{9.5}$$

Найчастіше використовуються такі значення: $\gamma = 0.5$, $\gamma = 1$.

Методи Рунге-Кутта **4-го порядку точності** (m=4). Найпопулярніша з усіх схем методів має вигляд:

$$\begin{aligned} &A_1 = 1/6, \quad A_2 = 1/3, \quad A_3 = 1/3, \quad A_4 = 1/6, \\ &K_1 = f(x_n, y_n), \quad K_2 = f(x_n + 0.5h_n, y_n + 0.5h_n K_1), \\ &K_3 = f(x_n + 0.5h_n, y_n + 0.5h_n K_2), \quad K_4 = f(x_{n+1}, y_n + h_n K_3). \end{aligned} \tag{9.6}$$

Поняття жорстких систем диференціальних рівнянь

Стійкість умовна і абсолютна. Якщо розглядати модельну задачу Коші для (9.1) з $f(x,u(x)) = \lambda \cdot u(x)$, $x > 0$, $\lambda < 0$, то стійкість означає виконання нерівності $|u(x+h)| \le |u(x)|$, $h > 0$.

При використанні різницевого методу потрібно вимагати, щоб розв'язок різницевої задачі задовольняв нерівність, аналогічну нерівності для розв'язку відповідної диференціальної задачі.

Означення 1. Різницевий метод є *абсолютно стійким* для довільного $h>0$, і *умовно стійким*, якщо він стійкий при певних обмеженнях на крок h.

Для явного методу Ейлера: $|y_{n+1}| \le |1+\lambda h||y_n|$ і маємо обмеження

$$0 < h \le \frac{2}{|\lambda|}. \tag{9.7}$$

Аналогічне обмеження на крок можна отримати для методу Рунге-Кутта 2-го порядку точності ($\gamma = 1$), а для 4-го –

$$|\lambda| \cdot h \le 2.78.$$

Для неявного методу Ейлера $|y_{n+1}| \le \left|\frac{1}{1-\lambda h}\right| |y_n|$ і завжди маємо

$$\frac{1}{1-\lambda h} < 1 \text{ при } 0 < h \,\&\, \lambda < 0.$$

Із наведених прикладів видно, що явні методи умовно стійкі, а неявні – безумовно стійкі, але вимагають розв'язання нелінійних рівнянь.

Розглянуті методи без значних зусиль переносяться на системи ЗДР, але виникає проблема з їх стійкістю (обмеження типу (9.7)), пов'язана з різномасштабністю процесів, які описуються системою ЗДР. Проілюструємо це на простому прикладі системи двох рівнянь

$$\begin{cases} \dfrac{du_1(x)}{dx} + a_1 u_1(x) = 0, \\ \dfrac{du_2(x)}{dx} + a_2 u_2(x) = 0, \end{cases}$$
$$x > 0; \;\; a_{1,2} > 0, \;\; a_2 >> a_1$$

або записаній у векторній формі

$$\frac{d\vec{u}(x)}{dx} = A \cdot \vec{u}(x), \;\; A = \begin{pmatrix} -a_1 & 0 \\ 0 & -a_2 \end{pmatrix}, \;\; \vec{u}(x) = \begin{pmatrix} u_1(x) \\ u_2(x) \end{pmatrix}. \tag{9.8}$$

Ураховуючи, що $a_{1,2} > 0, \;\; a_2 >> a_1$, з деякого x ми повинні мати (аналітично) $\vec{u}(x) = \begin{pmatrix} u_1(x) \\ u_2(x) \end{pmatrix} \approx \begin{pmatrix} u_1(x) \\ 0 \end{pmatrix}$.

Проте чисельно, якщо використовувати явний метод Ейлера, ми маємо обмеження на крок у вигляді нерівності (9.7), що приводить до $h \cdot a_2 \le 2$.

Означення 2. Нехай матриця A – стала, квадратна, розмірності N. Система (9.8) називається *жорсткою*, якщо:

1) $\mathrm{Re}(\lambda_k) < 0, k = 1,2,...,N$ – система асимптотично стійка за Ляпуновим;

2) $S = \dfrac{\max\limits_k |\mathrm{Re}(\lambda_k)|}{\min\limits_k |\mathrm{Re}(\lambda_k)|}$ – *число жорсткості* системи велике.

У системі MATLAB є вбудовані функції (їх часто називають *солвери – вирішувачі*), які реалізують різні методи, наприклад,

- Гіра для жорстких ЗДР – `ode15s`;
- типу Рунге-Кутта 2-го, 3-го порядку для нежорстких ЗДР – `ode23`;
- Розенброка 2-го порядку для жорстких ЗДР – `ode23s`;
- типу Рунге-Кутта 4-го, 5-го порядку для нежорстких ЗДР – `ode45`.

Решта солверів має імена `ode113,ode23t,ode23tb` і `ode15i` для розв'язання систем виду $\vec{f}\left(x,\vec{u}(x),\frac{d\vec{u}(x)}{dx}\right)=0$. Методика використання, з точки зору вхідних/вихідних параметрів, всіх солверів однакова і має такий стандартний шаблон виклику:

```
[X, Y] = solver(odef, [x0, xend], Y0, options);
```

де :

- `X` – сітка розв'язку, `Y` – матриця розв'язку (стовпчики $Y(:,i)=u_i(X(:))$);
- `solver` – ім'я вбудованої функції MATLAB;
- `odef` – функція користувача (умовно позначимо її `fun`) з описом правої частини ЗДР (9.1), яка записується у вигляді дескриптора (`@fun`) або рядка з її ім'ям ('`fun`');
- `[x0,xend]` – відрізок, на якому шукаємо розв'язок ЗДР, якщо подаємо його у вигляді вектора `[x0:h:xend]`, то задаємо точки, в яких отримуємо розв'язок ЗДР;
- `Y0` – вектор початкових значень задачі Коші ЗДР;
- `options` – необов'язковий набір параметрів управління солверами.

Інтерфейс солверів допускає звернення до них з одним вихідним аргументом:

```
sol = solver(odef, [x0, xend], Y0);
```

Такий виклик солвера приводить до утворення *структури* `sol` з інформацією про отриманий розв'язок (детальніше див. `help` відповідного солвера).

Для реалізації сервісних операцій при роботі з солверами для ЗДР, MATLAB містить дві групи команд:

1. для підтримки опцій солвера (структури `sol`):

`odeset`	створити/змінити опції солвера
`odeget`	отримати опції солвера

2. для формування вихідної інформації солвера в певних форматах, прийнятних для подальшої візуалізації результату:

`odeplot`	інформація солвера як матриці від часу
`odephas2`	інформація для побудови двовимірної фазової площини
`odephas3`	інформація для побудови тривимірної фазової площини
`odeprint`	діалогове вікно виведення на друк

Розглянемо приклади застосування деяких з цих солверів.

Приклад 20. Знайти наближений розв'язок задачі Коші для системи ЗДР на відрізку [0,35] (рівняння Лотки-Вольтерри для моделювання процесу "хижак-жертва"):

$$\begin{cases} u' = u*(1-\alpha*v), \\ v' = v*(\beta*u-1), \\ u(0)=1, \\ v(0)=1. \end{cases}$$

Нижче наведено текст відповідної *M*-функції, результати роботи якої зображено на рис. 9.1–9.2.

```
function pl9_1
%% Приклади розв'язання задачі Коші для ЗДР
% (рівняння Лотки-Вольтерри) для моделювання
% процесу "хижак-жертва".
clear all, close all, clc
global alf bet

alf = 1; bet = 2.5;
Y0 = [1; 1];        % вектор початкових умов
[T, Y] = ode45(@f91, [0 25], Y0); % розв'язання системи
plot(T, Y(:,1), T, Y(:,2),'r','LineWidth',2), grid on
legend('u','v','Location','Best')
title('Модель "хижак-жертва" ( {\alpha} = 1, {\beta} = 2.5 ) ' )
xlabel('t'), ylabel('u,v'), pause, close all

plot(Y(:,1), Y(:,2), 'b','LineWidth',2), hold on
alf = 2.5; bet = 1;
[T, Y] = ode45(@f91, [0 25], Y0);
plot(Y(:,1), Y(:,2), 'r','LineWidth',2), grid on
title('Фазові траєкторії')
xlabel('u'), ylabel('v')
ch=legend('{\alpha}=1, {\beta}=2.5',...
    '{\alpha}=2.5,{\beta}=1','Location','Best')
set(ch,'Fontsize',12)

function F = f91(t, U)
% підфункція правої частини системи
global alf bet
F = [U(1).*(1 - alf .*U(2)); U(2).*(bet.*U(1) - 1)];
```

Результат виконання М-функції *pl9_1.m*:

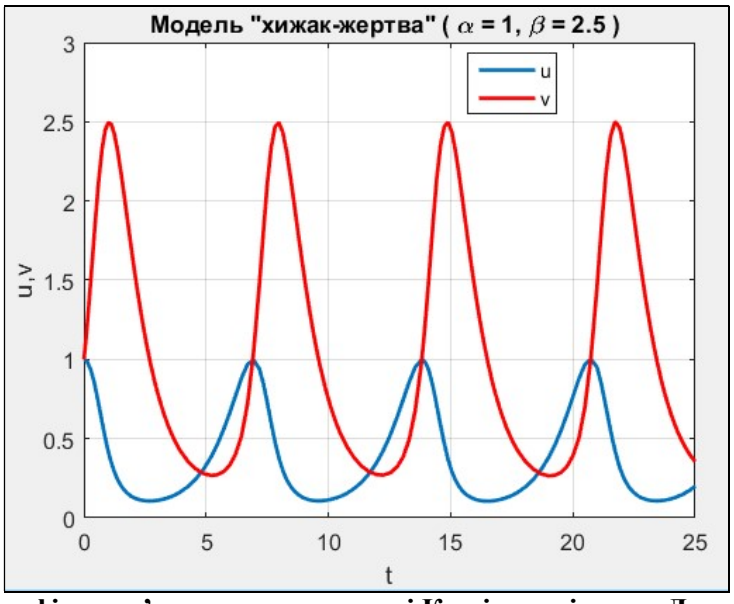


Рис. 9.1. Графік розв'язку системи задачі Коші для рівнянь Лотки-Вольтерри (параметри $\alpha = 1$; $\beta = 2.5$)

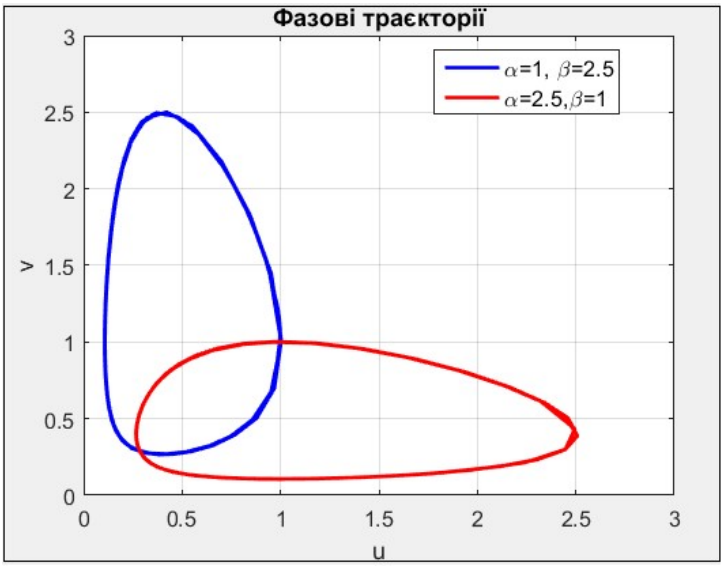


Рис. 9.2. Фазові траєкторії системи рівнянь Лотки-Вольтерри

Приклад 21. Знайти наближений розв'язок початково-крайової задачі для рівняння Кортевега-де Фріза (КдФ) з періодичними крайовими умовами

$$\begin{cases} \dfrac{\partial u(x,t)}{\partial t}+\dfrac{\partial}{\partial x}\left(\dfrac{u^2(x,t)}{2}\right)+b\dfrac{\partial^3 u(x,t)}{\partial x^3}=0, \\ x\in[l,r],\ 0<t\le T; \\ u(x,0)=u0(x); \\ u(l+\xi,t)=u(r+\xi,t),\ \xi\in[0,r-l] \end{cases}$$

при наступних параметрах: $b=0.1,\ l=0,\ r=2\pi,\ T=30,\ u0(x)=\sin(x)$. Для цієї задачі є декілька законів збереження, наприклад, *перший – закон збереження імпульсу*:

$$\int_l^r u(\varsigma,t)d\varsigma=\int_l^r u0(\varsigma)d\varsigma=C_1,$$

другий – закон збереження енергії:

$$\int_l^r u^2(\varsigma,t)d\varsigma=\int_l^r u0^2(\varsigma)d\varsigma=C_2.$$

Для наближеного розв'язання задачі використаємо метод *прямих* (*Роте*). Для цього введемо просторову сітку $\overline{\omega}_h=\{x_k=l+(k-1)h,\ k=1,2,...,n,\ r-1=(n-1)h\}$, а на прямих $x=x_k$ невідомі функції $U_k(t)=u(x_k,t)$. Всі диференціальні оператори в рівнянні КдФ замінимо на різницеві, врахуємо періодичність крайових умов і отримаємо наступну задачу Коші для системи ЗДР:

$$\begin{cases} \dfrac{d\vec{U}(t)}{dt}=\vec{F}(t,\vec{U}(t)), \\ 0<t\le T; \\ \vec{U}(0)=u0(\vec{X}),\quad \vec{X}=[x_1,x_2,...,x_n], \end{cases}$$

де

$$\vec{F}\left(t,\vec{U}(t)\right)=\begin{cases} f(k), & k=1, \quad kp=2, kpp=3, km=n-1, kmm=n-2; \\ f(k), & k=2, \quad kp=3, kpp=4, km=1, kmm=n-1; \\ f(k), & k=\overline{3,n-2}, \; kp=k+1, kpp=k+2, km=k-1, kmm=k-2; \\ f(k), & k=n-1, \quad kp=n, kpp=2, km=n-2, kmm=n-3; \\ f(k), & k=n, \quad kp=2, kpp=3, km=n-1, kmm=n-2; \end{cases}$$

$$f(k)=A*\left(U_{kp}(t)+U_k(t)+U_{km}(t)\right)*\left(U_{kp}(t)-U_{km}(t)\right)+$$
$$+B*\left(U_{kpp}(t)-2*\left(U_{kp}(t)-U_{km}(t)\right)-U_{kmm}(t)\right);$$

$$A=-1/6h, \; B=-b/2h^3.$$

Нижче наведено текст відповідного М-файла і результати його виконання (перевірка виконання закону збереження, рис. 9.3, 9.4).

```
function pl9_2
%% Приклади використання солверів для жорстких СЗДР
%  Застосування м.Роте для початково-крайової задачі
%  для рівняння Кортевега-де Фріза з періодичними
%  крайовими умовами на відрізку x Є [l,r].
clear all, close all, clc
global A B n3 n2 n1 n

% параметри крайової задачі :
T = 30; l = 0; r = 2.*pi; b = 0.1;
% завдання просторової сітки :
n = 93; x = linspace(l, r, n);
% завдання сітки отримання розв'язку :
m = 75; Tt = linspace(0, T, m);

U0 = @(x)(sin(x));    % функція початкових умов
Y0 = (U0(x))';        % вектор початкових умов

% допоміжні величини :
n1 = n-1; n2 = n-2; n3 = n-3; h = x(2)-x(1);
A = -1./(6.*h); B = -b./(2.*h.^3);

[t, Y] = ode23s(@pl9_2f, Tt, Y0);  % розв'язання задачі

% побудова сіток для виведення графіка розв'язку
[Mx, Mt] = meshgrid(x, t');
% виведення графіка наближеного розв'язку u(x,t)
surfl(Mx, Mt, Y)
shading interp, colormap pink
title('КЗ для рівняння Кортевега-де Фріза')
xlabel('x'), ylabel('t'),zlabel('u')
pause
contour(Mx, Mt, Y, 10)
xlabel('x'), ylabel('t')
colormap hsv
disp(strcat('Виконання 1-го закону збереження(імпульсу)=',...
            num2str(abs(h.*trapz(Y0-Y(m,:)')),13)))
disp(strcat('Виконання 2-го закону збереження (енергії)=',...
            num2str(abs(h.*trapz(Y0.^2-Y(m,:).^2')),13)))
end
```

```
function F = pl9_2f(t, U)      %  опис підфункції
% функція правої частини системи
    function f = fs    % вкладена функція
        f = A.*(U(kp)+U(k)+U(km)).*(U(kp)-U(km))+ ...
            B.*(U(kpp)-2.*(U(kp)-U(km))-U(kmm));
    end
    global A B n3 n2 n1 n
    F = zeros(n, 1);
    kmm = n2; km = n1; k = 1; kp = 2; kpp = 3; F(k) = fs();
    kmm = n1; km = n;  k = 2; kp = 3; kpp = 4; F(k) = fs();
    for k = 3 : n2
        kmm = k-2; km = k-1; kp = k+1; kpp = k+2; F(k) = fs();
    end
    kmm = n3; km = n2; k = n1; kp = n; kpp = 2; F(k) = fs();
    kmm = n2; km = n1; k = n;  kp = 2; kpp = 3; F(k) = fs();
end
```

Результат виконання М-функції *pl9_2.m*:

```
Виконання 1-го закону збереження(імпульсу)=3.403969715368e-006
Виконання 2-го закону збереження (енергії)=3.357457026136e-005
```

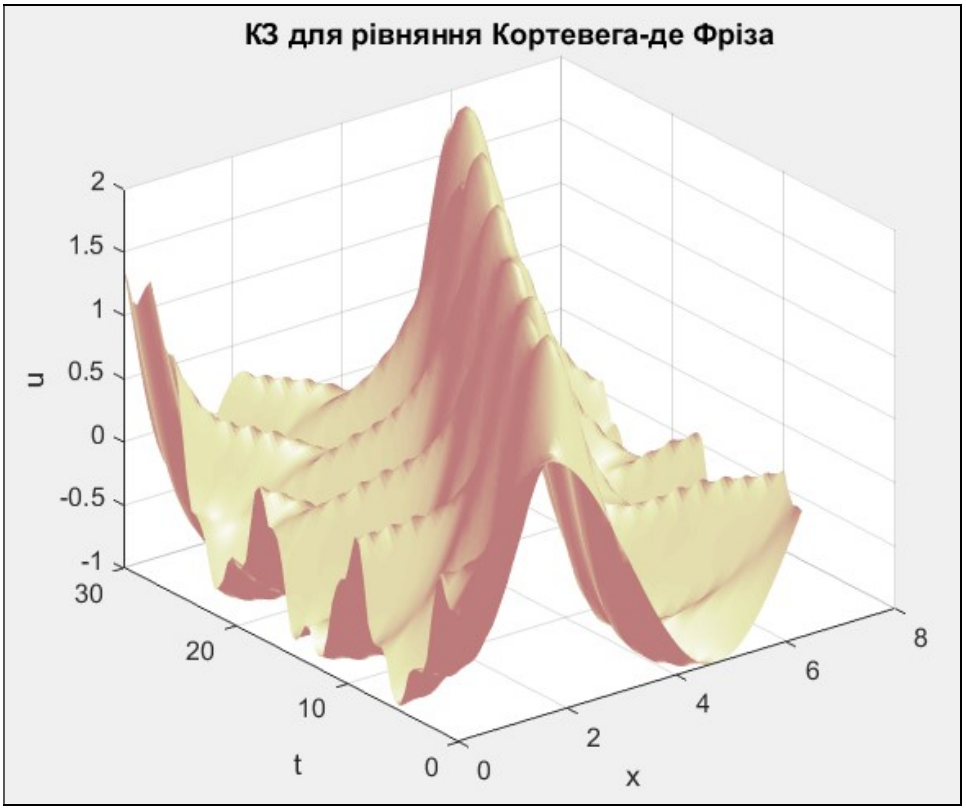


Рис. 9.3. Графік розв'язку КЗ для рівняння Кортевега-де Фріза (КдФ) з періодичними крайовими умовами

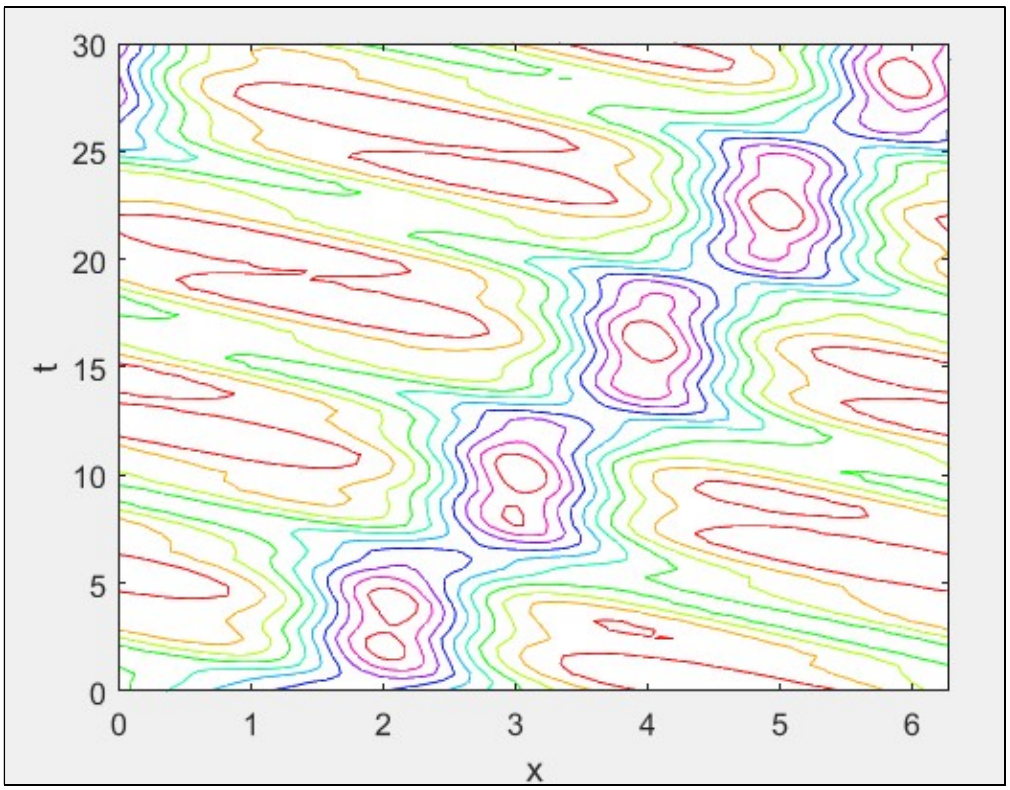


Рис. 9.4. Зображення ліній рівня для наближеного розв'язку $u(x,t)$

ТЕМА 10
ПОБУДОВА ГРАФІКІВ У MATLAB. РОБОТА З ФАЙЛАМИ

Побудова графіків функцій

Одна з переваг системи MATLAB – широкий спектр засобів графіки: від команд побудови простих графіків функцій однієї змінної в декартовій системі координат і до комбінованих презентаційних графіків з елементами анімації.

1. "Елементарна" графіка

Двовимірні графіки

plot(x,y, 'кмс')	Графік в лінійному масштабі, *x,y* – координати точок; коди: *к* – кольору, *м* – форми маркера, *с* – стилю лінії
loglog	Графік в логарифмічному масштабі (аналогічно *plot*)
semilogx	Графік в напівлогарифмічному масштабі по осі x (аналогічно *plot*)
semilogy	Графік в напівлогарифмічному масштабі по осі y (аналогічно *plot*)
polar	Графік в полярних координатах (аналогічно *plot*)

Символи управління кольором, стилем лінії, типом маркера					
1-й символ		**2-й символ**		**3-й символ**	
колір лінії		*форма маркера*		*стиль лінії*	
y	yellow (жовтий)	пробіл	відсутність маркера		відсутність лінії
m	magenta (малиновий)	.		–	суцільна
c	cyan (бірюзовий)	*O*		:	пунктирна
r	red (червоний)	*X*		–.	штрих-пунктирна
g	green (зелений)	+		– –	штрихова
b	blue (синій)	*			
w	white (білий)	*S*			
k	black (чорний)	*D*			
		V			
		^			
		<			
		>			
		P			
		H			

При побудові графіків використовуються додаткові команди

axis([*xl,xr,yl,yr*])	Масштабування і виведення осей координат
grid on|off	Управління відображенням сітки
title('*текст*')	Розміщення тексту над графіком (зверху)
legend()	Опис (поясненя) елементів графіка
text()	Розміщення тексту на графіку
xlabel('*текст*') (*ylabel*('*текст*'))	Розміщення тексту вздовж *X*-осі (*Y*-осі)
subplot(*n,i,j*)	Розбиття графічного вікна на *n* частин і позиціювання розміщення графіка в *i, j* комірці

Нижче наведено програму побудови графіків трьох функцій з різним стилем зображення кожної з них.

```
x=-2*pi:0.1*pi:2*pi;
y1=sin(x);
y2=sin(x).^2;
y3=x/6;
plot(x,y1,'-k',x,y2,'-.+r',x,y3,'--ob')
axis([-6 6 -1 1])
```

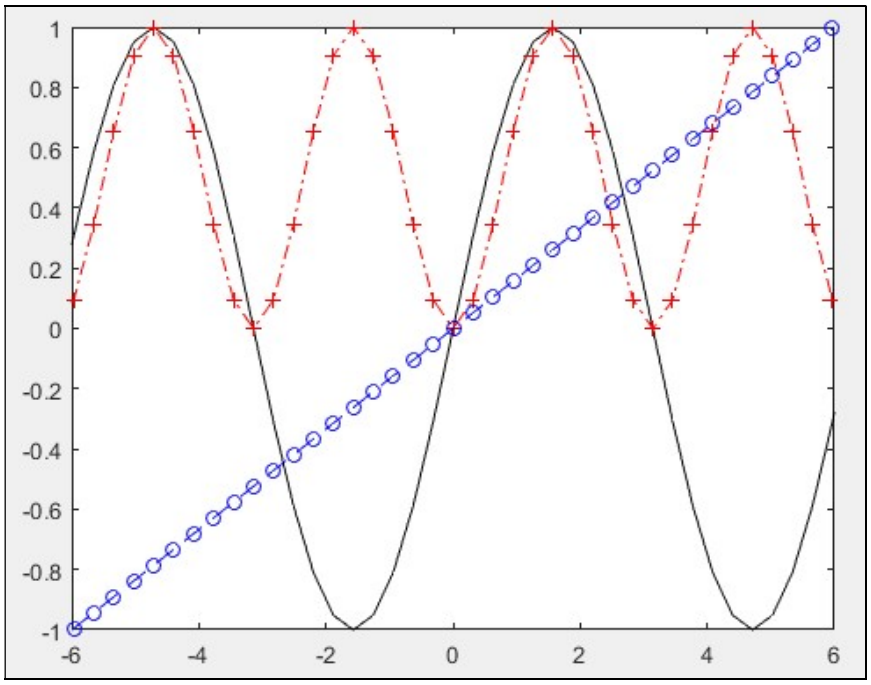


Рис. 10.1. Графіки функцій $y = \sin x$, $y = \sin^2 x$, $y = x$ на відрізку [-6, 6]

Тривимірні графіки

plot3(*x,y,z*)	Побудова ліній і точок в тривимірному просторі
contour	Зображення ліній рівня для тривимірної поверхні
contourc	Формування масиву опису ліній рівня
contour3	Зображення тривимірних ліній рівня
mesh(*X,Y,Z*)	Тривимірна сітчаста поверхня
meshc	Тривимірна сітчаста поверхня з проекцією ліній постійного рівня
meshz	Тривимірна сітчаста поверхня з площиною відліку на нульовому рівні
surf	Затінена сітчаста поверхня
surfc	Затінена сітчаста поверхня з проекцією ліній постійного рівня
surfl	Затінена сітчаста поверхня з підсвічуванням

Крім цих функцій, є багато додаткових функцій, що реалізують *управління кольором, завдання палітри кольорів, управління підсвічуванням, управління кутом перегляду, створення "твердої" копії та збереження графіка* тощо.

2. "Спеціальна" графіка

Двовимірні графіки

area	Зафарбування областей графіка
bar	Стовпчикова діаграма
barh	Стовпчикова діаграма з горизонтальним розташуванням
comet	Рух точки по траєкторії
compass	Графік векторів-стрілок, які виходять із початку координат
errorbar	Графік із зазначенням інтервалу похибки
ezplot	Побудова графіків з використанням діалогового вікна
feather	Графік векторів-стрілок, які виходять із рівновіддалених точок горизонтальної осі
fill	Зафарбування многокутника
hist	Побудова гістограми
pareto	Графік результатів профільованих програм
pie	Кругова діаграма
plotmatrix	Графік матриці
quiver	Графік поля напрямків
ribbon	Зображення ліній на тривимірному графіку
stairs	Ступінчастий графік
stem	Графік дискретних значень

Тривимірні графіки

*bar*3	Тривимірна стовпчаста діаграма
*bar*3*h*	Тривимірна стовпчикова діаграма з горизонтальним розташуванням
*comet*3	Рух точки по траєкторії в тривимірному просторі
contourf	Графік ліній рівня із зафарбованими областями
*fill*3	Зафарбування багатокутника в тривимірному просторі
*pie*3	Секторна діаграма
*quiver*3	Графік поля напрямків у тривимірному просторі
slice	Перетин функцій від трьох змінних
*stem*3	Графік дискретних значень у тривимірному просторі
trimesh	Тривимірна поверхня з трикутними комірками
trisurf	Тривимірна сітчаста поверхня з трикутними комірками
waterfall	Тривимірна поверхня без відображення ребер сітки

А ще доступні: *робота з графічними образами, анімаційні можливості, об'ємні графічні об'єкти, дескрипторна графіка, створення і управління осями координат, об'єкти дескрипторної графіки, операції над графічними об'єктами, графічний інтерфейс користувача (GUI), засоби проектування GUI, діалогові панелі, створення меню і кнопок інструментальної панелі і т. д.*

Використання бінарних і текстових (форматованих) файлів для введення, виведення

1. Відкриття і закриття файлу

fp=fopen(*namef,mode*)	Зв'язати змінну *fp* з файлом, що має ім'я *namef* на зовнішньому пристрої та відкрити його для обробки згідно з режимом *mode*. Рядок *mode* може мати значення: '*r*' - читання (існуючого); '*w*' – запис (у новий); '*a*' – доповнення існуючого. Наявність '+' означає можливість заміни записів у файлі (прямий доступ). Значення '*b*' – бінарний, '*t*' – текстовий файл.
fclose(*fp*)	Закрити файл *fp*.

2. Бінарні файли

[*A,count*]= *fread*(*fp,size,precssion*)	Виконує читання в масив *A* даних бінарного файла, виозначеного змінною *fp*. Аргумент *count* отримує число успішно прочитаних елементів. Допустимі значения size: *N* = читається *N* елементів у вектор-стовпець; *Inf* = читається до кінця файла; [*M,N*] = читаються елементи і записуються по стовпцях у матрицю розмірності *M*N* (допустимо задавати кількість стовпців *N* як *inf*). Якщо аргумент *size* опущено, то виконується читання усіх даних. Аргумент *precssion* визначає кількість біт для представлення даних: MATLAB C or Fortran Description *'uchar' 'unsigned char' unsigned integer,* 8 *bits.* *'schar' 'signed char' signed integer,* 8 *bits.* *'int*8*'* *'integer**1*'* *integer,* 8 *bits.* *'int*16*'* *'integer**2*'* *integer,* 16 *bits.* *'int*32*'* *'integer**4*'* *integer,* 32 *bits.* *'int*64*'* *'integer**8*'* *integer,* 64 *bits.* *'uint*8*'* *'integer**1*'* *unsigned integer,* 8 *bits.* *'uint*16*'* *'integer**2*'* *unsigned integer,* 16 *bits.* *'uint*32*'* *'integer**4*'* *unsigned integer,* 32 *bits.* *'uint*64*'* *'integer**8*'* *unsigned integer,* 64 *bits.* *'single' 'real**4*'* *floating point,* 32 *bits.* *'float*32*'* *real**4*'* *floating point,* 32 *bits.* *'double' 'real**8*'* *floating point,* 64 *bits.* *'float*64*'* *'real**8*'* *floating point,* 64 *bits.* Якщо аргумент *precssion* опущено, то *'uint*8*'*
count= *fwrite(fp,A,precssion*)	Записує елементи матриці *A* у файл *fp*. Дані записуються по стовпцях. *count* – лічильник успішно записаних даних; *fp* – ціле, файлова змінна при відкритті файла, або 1 для стандартного файла виведення, або 2 – для файла помилок; *precssion* визначає розмір і точність даних (як у *fread*).

3. Позиціонування файла

[*message,errnum*]=*ferror*(*fp*)	Отримати інформацію про помилку *I/O*.
st=feof(*fp*)	Чи досягнуто (*s*t = 1) кінець файла *fp*.
st=fseek(*fp,offset,origin*)	Встановити маркер файла *fp* у задану позицію. Маркер переміщується на *offset* байт вправо(>=0) | вліво (<0) від позиції, вказаної в *origin*, яка має значення : *'bof'* або –1 початок файла; *'cof'* або 0 поточна позиція; *'eof'* або 1 кінець файла.
pos=ftell(*fp*)	Визначити в байтах позицію *pos* маркера від початку файла *fp*. При pos= –1 – помилка.
frewind(*fp*)	Встановити маркер у початок файла *fp*.

4. Текстові (форматовані) файли

[*A,count*]= *fscanf*(*fp,format,size*)	Прочитати в матрицю *A* дані з файла *fp* під управлінням *format*. Значення *count*, *size* аналогічні оператору *fread*. Рядок *format* може містити пробіли, звичайні символи або символи специфікаторів перетворення (використання як у мові С): *%d* – знакове ціле; *%u* – ціле; *%f* – дійсне число з фіксованою крапкою; *%e* – дійсне число з плаваючою крапкою (експоненціальна форма); *%g* – дійсне число з фіксованою|плаваючою крапкою; *%s* – рядок; *%q* – рядок, обмежений подвійними лапками; *%c* – символи, включаючи символи пробілів; %[...] – найбільший рядок, із символами з множини, вказаної в []; %[^...] – найбільший рядок, в який не входять символи з множини, вказаної в []. Ширина поля регулюється для цілих даних (%5*d* – ціле з 5 цифр). Формати *%f*, *%e* підтримують форму %<ширина>.<точність>. Екранні символи *\n,\r,\t,\b,\f* також мають звичайний смисл.
count= *fprintf*(*fp,format,A,…*)	Записати дані *A*, … у файл *fp* під управлінням форматів, визначених у рядку *format*. Значення аргументу *count* аналогічне оператору *fwrite*.
line=fgetl(*fp*)	Прочитати рядок *line* із файла *fp*, видаливши символ кінця рядка (–1 при читанні кінця файла).
line=fgets(*fp*)	Аналогічно *fgetl*, але включається символ кінця рядка.

Інколи корисно використовувати функцію `sprintf` – форматне перетворення даних *A, B, C, ...* у рядок символів *S* :

```
[S, errmsg] = sprintf(format, A, B, C, ...)
```

Значенням необов'язкового аргументу `errmsg` є повідомлення про помилку (за наявності), або порожній рядок (при її відсутності).

Функції `fprintf` і `sprintf` – "векторизовані", тобто, коли вхідними даними є масив, то рядок формату `format` циклічно застосовується до елементів стовпця масиву, поки не будуть вичерпані всі елементи. Наприклад,

```
x =0 : 0.1 : 1;  y = [x, exp(-x)];
f=fopen('ex1file.txt', 'w');
fprintf(f, '%6.2f%13.6f\n', y);
fclose(f);
type ex1file.txt
```

Імпорт, експорт файлів

load namef *load namef ac*p*	Прочитати змінні з (бінарного) *namef.mat* файла; Прочитати змінні з "шаблонними" іменами *ac*p* із *namef.mat* файла. При завантаженні MAT-файла, нові значення однойменних змінних будуть записані замість існуючих.
save namef *a ab ac*p -append*	Записати змінні з вказаними іменами *a ab* і з "шаблонними" іменами *ac*p* у бінарний *namef.mat* файл. За наявності опції *append* можна додати вказані дані до існуючого MAT-файла.
A=csvread('*namef.csv*')	Прочитати числові елементи, які розділені "," з текстового файла *namef.csv* і записати в масив *A*.
csvwrite('*namef.csv*',*A*)	Записати числовий масив *A* у текстовий файл *namef.csv*, розділяючи елементи за допомогою ",".

Зазначимо, що для роботи з графічними форматами файлів використовуються функції *imread*, *imwrite*, звуковими файлами – *wavwrite, wavread*.

Збереження змінних робочої області. Команда *save* дозволяє зберегти вміст робочої області (Workspace) у бінарному MAT-файлі, який у подальшому можна завантажити командою *load*. Команда *save* також доступна як опція *Save Workspace* меню *File.*

Функція menu. Для створення меню використовується команда *menu* у вигляді:

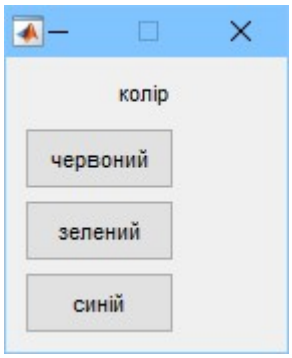
choice=menu('заголовок','вибір 1','вибір 2',…,'вибір n');

яка реалізує виведення графічного вікна меню із заголовком *заголовок* і кнопками вибору *вибір i*. Змінна *choice* отримує число 1,2,…,*n* – номер вибраної кнопки меню.

Проілюструємо роботу функції *menu* для побудови графіка деякої функції на заданому інтервалі кольором, вибраним із меню

```
k=menu ('колір','червоний','зелений','синій')
color=['r';'g';'b'];
x=-2:0.1:2;
y=x.*x;
plot(t,y,color(k))
```

Функція *menu* виводить на екран вікно у вигляді:



При натисненні на одну з кнопок із назвою кольору, функція *menu* повертає номер цієї кнопки (1,2 або 3), і цей номер стає значенням змінної *k.* Вектор-стовпець *color* містить атрибути кольору для побудови графіка. За номером *k* потрібний атрибут вибирається із масиву *color* і вказується в команді *plot*. У результаті, обравши потрібний пункт меню, одержимо графік функції, побудований вибраним кольором.

ТЕМА 11
ВИКОРИСТАННЯ СОЛВЕРІВ MATLAB ДЛЯ РОЗВ'ЯЗАННЯ КРАЙОВИХ ЗАДАЧ ДЛЯ ЗДР

Розглянемо крайову задачу для ЗДР, записану у вигляді системи диференціальних рівнянь першого порядку

$$\vec{U}'(x) = \vec{F}(x, \vec{U}(x)), \quad x \in [a,b] \tag{11.1}$$

і набору додаткових крайових умов

$$\vec{G}(\vec{U}(a), \vec{U}(b)) = 0. \tag{11.2}$$

Тут уведено позначення:

$\vec{U}(x) = [u_1(x), u_2(x), ..., u_n(x)]$ – вектор-функція шуканого розв'язку;

$\vec{F}(x, \vec{U}(x)) = [f_1(x, \vec{U}(x)), f_2(x, \vec{U}(x)), ..., f_n(x, \vec{U}(x))]$ – вектор-функція з описом правих частин системи (11.1), де $f_i(...)$ – відомі функції;

$\vec{G}(\vec{U}(a), \vec{U}(b)) = [g_1(\vec{U}(a), \vec{U}(b)), g_2(\vec{U}(a), \vec{U}(b)), ..., g_n(\vec{U}(a), \vec{U}(b))]$ – вектор-функція із завданням крайових умов задачі (11.1), де $g_i(...)$ – відомі функції.

Для задачі (11.1) – (11.2) будемо вважати, що питання існування та єдиності розв'язку досліджені. Для наближеного розв'язку (11.1) – (11.2) використовують різні наближені методи:

- *редукція до задачі Коші* (*балістичний метод*);
- чисельно-аналітичні – *метод Трефца*; *проекційні*, *варіаційні*; *асимптотичні* (розвинення у ряд за малим параметром);
- *метод скінченних різниць* (у різних модифікаціях побудови різницевої задачі).

У системі MATLAB є вбудована функція `bvp4c` для розв'язання задач типу (11.1) – (11.2). Стандартна схема її використання має вигляд :

```
a = «a»; b = «b»; m = «m»;
x_init = «сітка по x»; % наприклад, linspace(a, b, m);
u_init = «[A(m,n)]»;
sol_init = bvpinit(x_init, u_init);
sol = bvp4c(@F, @G, sol_init);
% Виведення результатів :
N = «N»; x = linspace(a, b, N);
Y = deval(sol, x);
```

Функція `bvpinit` задає деяку "підказку":

`x_init` – вектор-рядок розмірності m, який задає сітку за змінною x, наприклад, `linspace(a,b,m);`

`u_init` – матриця розмірності $m \times n$ або вектор-стовпець розмірності n. Елементи матриці $A(m,n)$ задають значення функцій $u_1(x), u_2(x), ..., u_n(x)$ у вузлах сітки. Якщо це – вектор-стовпець, то вказані значення для кожної функції $u_i(x)$ є сталими.

Вихідний параметр `bvpinit` – структура `sol_init` з полями `x,y`.

Перший вхідний параметр у `bvp4c` задає вказівник на функцію користувача з описом правої частини системи (11.1). Ця функція повинна бути описана наступним чином:

```
function dfdx = F(x, y)
dfdx = [f1(…);f2(…);…;fn(…)];
```

Тут `y` – вектор значень функцій $u_1(x), u_2(x), ..., u_n(x)$ у точці x. Вихідний параметр `dfdx` – вектор значень функцій $f_1(...), f_2(...), ..., f_n(...)$.

Другий вхідний параметр у `bvp4c` задає вказівник на функцію користувача з описом граничних умов (11.2). Ця функція повинна бути описана наступним чином:

```
function bc = G(ya, yb)
bc = [g1(…);g2(…);…;gn(…)];
```

Тут `ya, yb` – вектори, що задають значення функцій $u_1(x), u_2(x), ..., u_n(x)$ у граничних точках a і b відповідно. Вихідний параметр `bc` – вектор зі значень функцій $g_1(...), g_2(...), ..., g_n(...)$. Вихідний параметр `bvp4c` – структура `sol`, що описує знайдений розв'язок.

Функція `deval` обчислює значення знайденого розв'язку `sol` в точках `x` сітки виведення графіка. Вихідний параметр функції `deval` – матриця `Y`, де `Y(i,:)` $= u_i(x)$.

Приклад 22. Знайти наближений розв'язок модельної крайової задачі

$$\begin{cases} u''(x) = 6x(1-2x), \\ \quad x \in (0,1); \\ u(0) = 0,\ u(1) = 0 \end{cases}$$

і порівняти точний $u(x) = x^3(1-x)$ і знайдений розв'язок.

Нижче наведено текст відповідного *M*-файла, результати роботи якого зображено на рис. 11.1.

```
function pl11_1
%% Розв'язання крайової задачі :
% u"(x) = 6x(1-2x),
% u(0) = 0, u(1) = 0.
%    Модельна задача: u(x)=x^3*(1-x)
clear; clc

% Створюємо структуру sol_init початкових даних із m точок,
% рівномірно розподілених на відрізку [0, 1] зі значеннями y=[0; 0]:
a = 0; b = 1; m = 51; xx = linspace(a, b, m); yy = [0; 0];
sol_init = bvpinit(xx, yy);

sol = bvp4c(@F, @G, sol_init);  % Розв'язуємо задачу

% Виведення результатів :
N = 101; x = linspace(a, b, N);
Y = deval(sol, x);
u = @(x)(x .^ 3 .* (1 - x));    % Завдання точного розв'язку
xx = linspace(a, b, 21);
plot(x, Y(1,:),'k -', xx, u(xx),'*', x, Y(2,:),...
   'r :','LineWidth',2);
axis([a, b, -1.2, 0.7]); grid on
title('Розв''язання КЗ для ЗДР'), xlabel('x'), ylabel('Y(x)')
legend('y(x)', 'u(x)', 'y''(x)')
```

```
function dydf = F(x, y)
dydf = [y(2); 6.*x.*(1-2.*x)];

function bc = G(ya, yb)
bc = [ya(1); yb(1)];
```

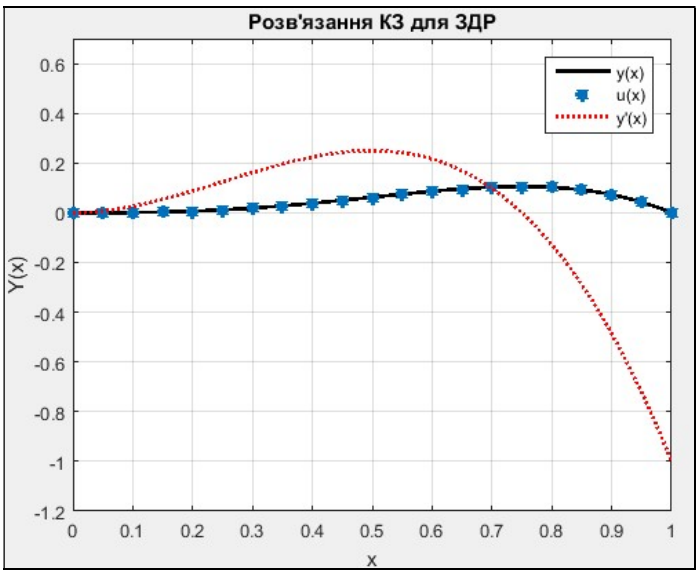


Рис. 11.1. Графік розв’язку модельної крайової задачі

Приклад 23. Розв’язати крайову задачу для нелінійного рівняння Треша:

$$\begin{cases} u''(x) = \alpha \cdot sh(\alpha \cdot u(x)), \\ \quad x \in (0,1); \\ u(0) = 0,\ u(1) = 1 \end{cases}$$

при наступних варіантах значень параметра $\alpha = [1.0, 5.0, 10.0, 15.0, 17.0]$.

Нижче наведено текст відповідного *M*-файла, результати роботи якого зображено на рис. 11.2.

```
function pl11_2
%% Розв'язання КЗ для нелінійного рівняння Треша:
% u"(x) = alfa*sinh(alfa*u(x)),
%     u(0) = 0, u(1) = 1.
%  за допомогою вбудованих функцій MATLAB.
clear; clc
global alfa
a = 0; b = 1; m = 51; xx = linspace(a, b, m); yy = [0; 1];
sol_init = bvpinit(xx, yy);
ALFA = [1.0, 5.0, 10.0, 15.0, 17.0]; na = length(ALFA);
N = 101; x = linspace(a, b, N); Y = zeros(na, N);
for i = 1 : na
    alfa = ALFA(i)
    sol = bvp4c(@F, @G, sol_init);
    y = deval(sol, x);
    Y(i,:) = y(1,:);
end
plot(x, Y,'LineWidth',2), grid on
xlabel('x'), ylabel('u(x)')
title('Розв''язання КЗ для рівняння Треша')
ch=legend(strcat('{\alpha}=',num2str(ALFA(:))),...
```

```
  'Location','Best')
set(ch,'Fontsize',12)

function dydf = F(x, y)
global alfa
dydf = [y(2); alfa.*sinh(alfa.*y(1))];

function bc = G(ya, yb)
bc = [ya(1); yb(1) - 1];
```

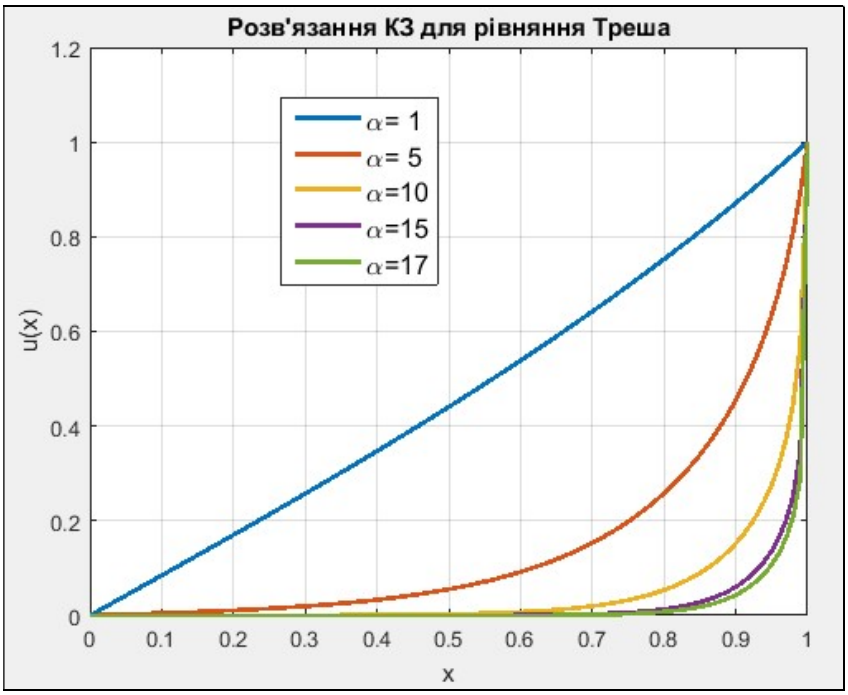


Рис. 11.2. Графіки розв'язку крайової задачі для нелінійного рівняння Треша при різних значеннях параметра α

Для більших значень α, на жаль, використання `bvp4c` не приносить результату. Таке "просте" на вигляд нелінійне рівняння є одним із тестових для перевірки методів розв'язання нестійких нелінійних крайових задач. Отже не для всіх задач можна використовувати стандартні програми.

Приклад 24. Розв'язати модельну крайову задачу [18, с. 154–155]:

$$\begin{cases} \left(K(x)*u'(x)\right)' = 0, \\ x \in (0,1), \\ u(0)=1,\ u(1)=0, \\ [u(t)] \equiv \left(u(t+0)-u(t-0)\right)=0, \\ [K(t)u'(t)]=0;\quad t \in (0,1), \end{cases} \quad \text{де } K(x)=\begin{cases} K_1,\ 0<x<t, \\ K_2,\ t<x<1, \end{cases} \quad K_{1,2}>K_0>0;$$

$$t = x_m + 0.5h,\ 1 < m < N\,.$$

Порівняти наближений розв'язок, отриманий на рівномірній сітці, з відомим точним розв'язком задачі:

$$u(x)=\begin{cases} 1-\alpha*x,\ 0\le x\le t, \\ \beta*(1-x),\ t<x\le 1, \end{cases} \quad \alpha = 1/\left(\gamma+(1-\gamma)*t\right),\ \beta=\alpha*\gamma,\ \gamma=K_1/K_2\,.$$

Нижче наведено текст відповідного *M*-файла, результати роботи якого зображено на рис. 11.3 – 11.4.

```
function pl11_3
%% Розв'язання модельної крайової задачі :
%      ( K(x)u'(x) )' = 0,
%       u(0) = 1, u(1)= 0.
%  з розривним коефіцієнтом K(x).
clear all, close all, clc
global t al be K1 K2
a = 0; b = 1; N = 51;  K1 = 2; K2 = 5; m = 33;
xx = linspace(a, b, N); t = xx(m) + 0.5.*(xx(2)-xx(1));
ga = K1./K2; al = 1./(ga + (1 - ga).*t); be = al.*ga;
% Створюємо структуру sol_init початкових даних із m точок,
%  рівномірно розподілених на відрізку [a, b]
%  зі значеннями y=[1; 0]:
yy = [1; 0]; sol_init = bvpinit(xx, yy);
sol = bvp4c(@F, @G, sol_init);   % Розв'язуємо задачу
% Виведення результатів :
n = 101; x = linspace(a, b, n);  Y = deval(sol, x);
nn = 21; xx = linspace(a, b, nn); u = xx;
for k = 1 : nn
    u(k) = U(xx(k));
end
plot(x, Y(1,:),'k -', xx, u, '*','LineWidth',2)
axis([a, b, -0.1, 1.1]), grid on
title('КЗ для (K(x)u''(x))''=0 (K-розривна)')
xlabel('x'), ylabel('Y(x)')
legend('Y','u(x)','Location','Best'), pause
u = Y(1, 1:5:end) - u;
disp(strcat('Похибка в нормі C=',num2str(norm(u))))
plot(xx, abs(u),'LineWidth',2), grid on
title('Абсолютна похибка')
xlabel('x'), ylabel('|Y-u(x)|')

function k = K(x)
% підфункція з описом коефіцієнта K
global t K1 K2
if x <= t
    k = K1;
else
    k = K2;
end

function dydf = F(x, y)
% підфункція опису правої частини системи
dydf = [y(2)./K(x); 0];

function bc = G(ya, yb)
% підфункція опису системи крайових умов
bc = [ya(1) - 1; yb(1)];

function u = U(x)
% підфункція опису точного розв'язку u(x)
global t al be
if x <= t
    u = 1 - al.*x;
else
    u = be.*(1 - x);
end
```

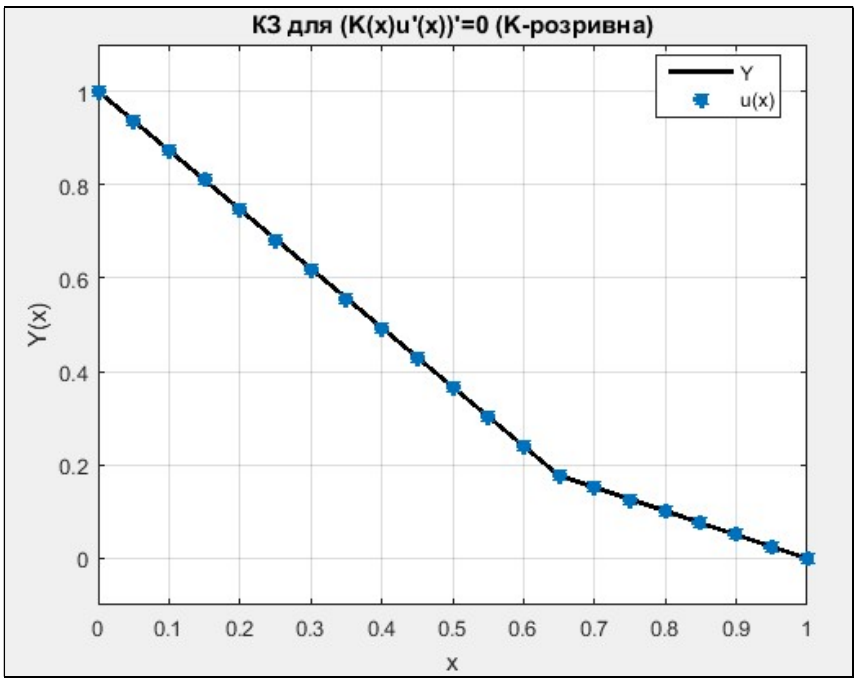


Рис. 11.3.Графік розв'язку КЗ з розривним коефіцієнтом

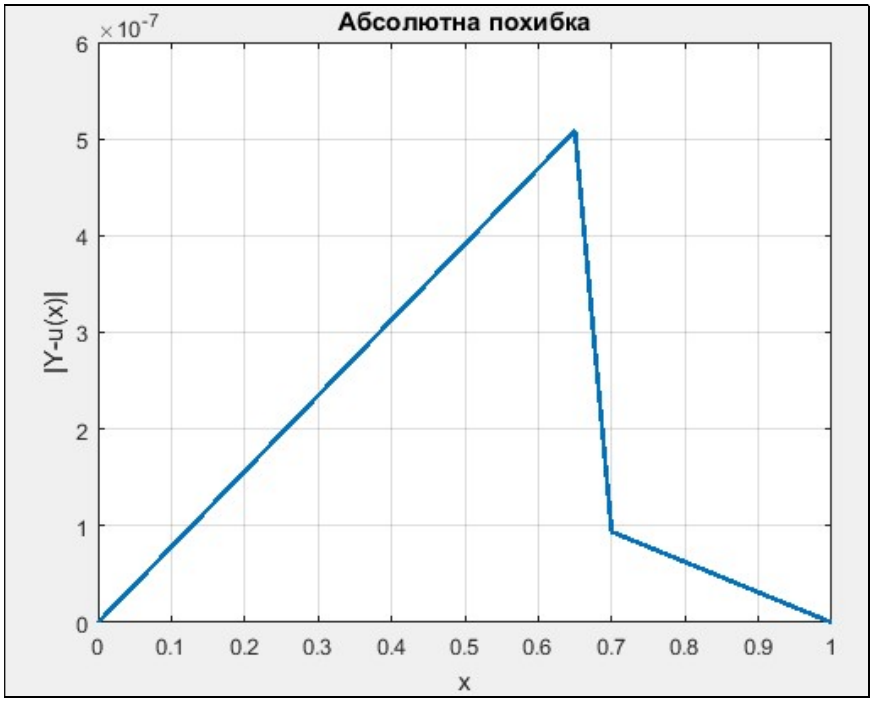


Рис. 11.4. Графік абсолютної похибки розв'язку КЗ з розривним коефіцієнтом

```
Похибка в нормі C=1.1294e-006
```

ТЕМА 12
ВИКОРИСТАННЯ СОЛВЕРІВ MATLAB ДЛЯ РОЗВ'ЯЗАННЯ ОДНОВИМІРНИХ КРАЙОВИХ ЗАДАЧ ДЛЯ СИСТЕМ ПАРАБОЛІЧНИХ І ЕЛІПТИЧНИХ ДРЧП

Розглянемо систему рівнянь

$$\vec{c}\left(x,t,\vec{U},\frac{\partial\vec{U}}{\partial x}\right)\frac{\partial\vec{U}}{\partial t}=x^{-m}\frac{\partial}{\partial x}\left(x^m\vec{f}\left(x,t,\vec{U},\frac{\partial\vec{U}}{\partial x}\right)\right)+\vec{s}\left(x,t,\vec{U},\frac{\partial\vec{U}}{\partial x}\right), \qquad (12.1)$$

$$x\in(a,\ b),\quad t\in(t0,\ tf]$$

(параметр $m=0,1,2$ відповідає плоскій, циліндричній, або сферичній симетрії), для якої виконуються початкові умови

$$\vec{U}(x,t_0)=\vec{U}_0(x) \qquad (12.2)$$

і набір крайових умов в граничних точках $x=a$ (при $m>0$ повинно бути $a\geq 0$), $x=b$ скінченного інтервалу $[a,\ b]$

$$\vec{p}\left(x,t,\vec{U}\right)+\vec{q}(x,t)\vec{f}\left(x,t,\vec{U},\frac{\partial\vec{U}}{\partial x}\right)=0. \qquad (12.3)$$

Тут уведено позначення:

$\vec{U}(x)=[u_1(x,t),u_2(x,t),...,u_n(x,t)]$ – вектор-функція шуканого розв'язку;

$\vec{c}\left(x,t,\vec{U},\frac{\partial\vec{U}}{\partial x}\right)$ – відома діагональна матриця (теплоємність), елементи якої приймають нульові або додатні значення. Елемент, рівний нулю, відповідає еліптичному рівнянню, інакше – параболічному рівнянню. В системі (12.1) повинно бути принаймні одне параболічне рівняння. Елемент $c(...)$, який відповідає параболічному рівнянню, може перетворюватися у нуль в ізольованих значеннях x, якщо ці значення – вузли сітки. Розриви розподілу функцій $c(...)$ і/або $s(...)$ за просторовою координатою допустимі при умові, що точки сітки представлені в кожному підінтервалі, де знаходяться розриви;

$\vec{f}\left(x,t,\vec{U},\frac{\partial\vec{U}}{\partial x}\right)$, $\vec{s}\left(x,t,\vec{U},\frac{\partial\vec{U}}{\partial x}\right)$ – відомі вектор-функції (потік, джерела) з описом коефіцієнтів рівняння системи (12.1);

$\vec{p}\left(x,t,\vec{U}\right)$, $\vec{q}(x,t)$ – відомі вектор-функції, причому елементи $q(...)$ або тотожно дорівнюють нулю, або ні при яких значеннях x, t не дорівнюють нулю. Крім того, із цих двох коефіцієнтів тільки $p(...)$ може залежати від $U(x,t)$.

У системі MATLAB є вбудована функція `pdepe` для розв'язку задач (12.1) – (12.3). Найпростіший синтаксис цієї функції:

```
sol = pdepe(m, @pdefun, @icfun, @bcfun, xmesh, tspan);
```

Аргументи функції `pdepe`:

- `m` – параметр, що описує симетрію задачі (його значення 0, 1 або 2);
- `pdefun` – функція користувача, яка обчислює значення коефіцієнтів *c, f, s* системи (12.1). Ця функція викликається за формою

```
[c, f, s] = pdefun(x, t, u, dudx).
```

Її вхідними параметрами є скалярні величини `x` і `t` і вектори `u` і `dudx`, які наближено відповідають розв'язку U та його частинній похідній по x;

- `icfun` – функція користувача, яка обчислює початкові умови (12.2). Ця функція викликається за формою

```
u = icfun(x);
```

- `bcfun` – функція користувача, яка обчислює значення функцій p та q граничних умов (12.3). Ця функція викликається за формою

```
[pl, ql, pr, qr] = bcfun(xl, ul, xr, ur, t).
```

Тут `ul` – наближений розв'язок на лівому кінці відрізка `xl = a`, `ur` – наближений розв'язок на правому кінці відрізка `xr = b`; `pl` і `ql` – відповідні масиви функцій p і q для лівої граничної точки `xl`, аналогічно `pr` і `qr` відповідні масиви функцій p і q для правої граничної точки `xr`;

- `xmesh` – вектор $[x_1, x_2, ..., x_N]$, $N > 3$, що задає координати сітки по x $a = x_1 < x_2 < ... < x_N = b$;
- `tspan` – вектор $[t0, t1, ..., tf]$ моментів часу для фіксації шуканої функції за координатою t. Елементи `tspan` повинні задовольняти умову $t0 < t1 < ... < tf$, кількість елементів `tspan` повинна бути не менше 3.

Функція `pdepe` формує розв'язок у вигляді багатовимірного масиву `sol`:

$$U_i(x,t) = \texttt{sol(:,:,i)},$$

при цьому перший індекс (j) відповідає номеру часового шару t_j, другий індекс (k) визначає номер вузла x_k, а третій індекс (i) дає номер компоненти вектора розв'язку, таким чином елемент `sol(j,k,i)` дає значение розв'язку $u_i(x_k,t_j)$. Інакше кажучи, елемент `Ui(j,k)=sol(j,k,i)` є наближенням U_i при `(t,x)= (tspan(j), xmesh(k))`.

Розглянемо деякі приклади використання М-функції `pdepe`.

Приклад 25. Розв'яжемо крайову задачу, яка містить еліптичну і параболічну складову системи (12.1):

$$\begin{cases} \dfrac{\partial}{\partial x}\left(\dfrac{\partial U_1(x,t)}{\partial x}\right) = 10 * U_2(x,t), \\ \dfrac{\partial U_2(x,t)}{\partial t} = \dfrac{\partial}{\partial x}\left(\dfrac{\partial U_2(x,t)}{\partial x}\right) + U_1(x,t) * U_2(x,t), \\ x \in (0,1), \quad t \in (0,1], \\ \dfrac{\partial U_1(0,t)}{\partial x} = 0, \quad U_1(1,t) = 1 - t, \\ U_2(0,t) = t, \quad U_2(1,t) = 0, \\ U_2(x,0) = 0.5 - |0.5 - x|. \end{cases}$$

Нижче наведено текст відповідного М-файла та результати його роботи (рис. 12.1–12.2).

```
function pl12_1
%% Розв'язання крайової задачі :
%      0 = U1" - 10*U2(x,t),
% dU2/dt = U2" + U1(x,t)*U2(x,t),
% U1'(0,t)= 0, U1(1,t) = 1 - t,
% U2(0,t) = t, U2(1,t) = 0.
% U2(x,0) = 0.5 - |x - 0.5|.
% за допомогою вбудованих функцій MATLAB.
```

```
clear; clc
m = 0; % плоска задача
xl = 0; xr = 1; x = linspace(xl, xr, 21); % сітка по x
t = linspace(0, 1, 21); % вектор по часу
% Розв'язуємо задачу
sol = pdepe(m, @koefpde, @initpde, @boundpde, x, t);

[X, T] = meshgrid(x, t);
% графіки розв'язку U1(x,t), U1(x,t)
u = sol(:, :, 1);
mesh(X, T, u); colormap gray; view(-8, 12);
title('Функція U1(x,t)');
xlabel('x'); ylabel('t'); pause;

u = sol(:, :, 2);
mesh(X, T, u); colormap gray; view(-15, 36);
title('Функція U2(x,t)');
xlabel('x'); ylabel('t');

function [c, f, s] = koefpde(x, t, u, DuDx)
% коефіцієнти рівняння
c = [0; 1];
f = [DuDx(1); DuDx(2)];
s = [-10 .* u(2); u(1) .* u(2)];

function u0 = initpde(x)
% початкова умова
u0 = [0; x .* (1 - x)];

function [pl, ql, pr, qr] = boundpde(xl, ul, xr, ur, t)
% коефіцієнти крайових умов
pl = [-t; ul(2)];
ql = [1; 0];
pr = [ur(1) + t - 1; ur(2)];
qr = [0; 0];
```

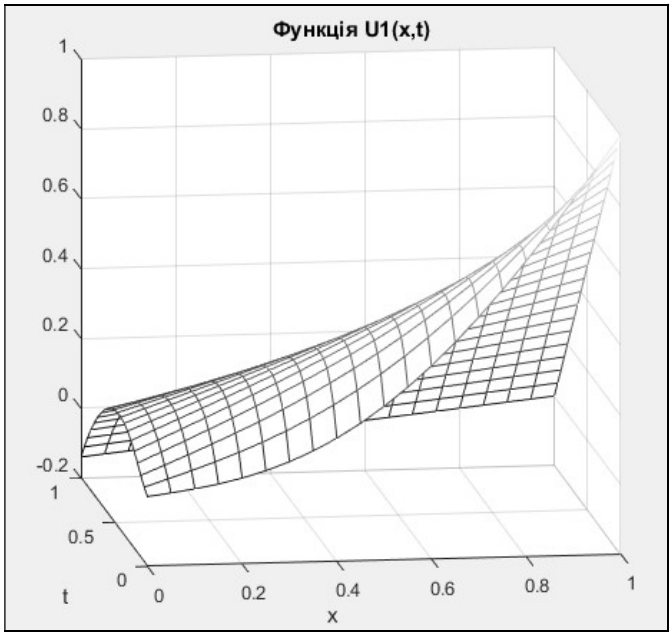


Рис. 12.1. Графік функції $U_1(x, t)$

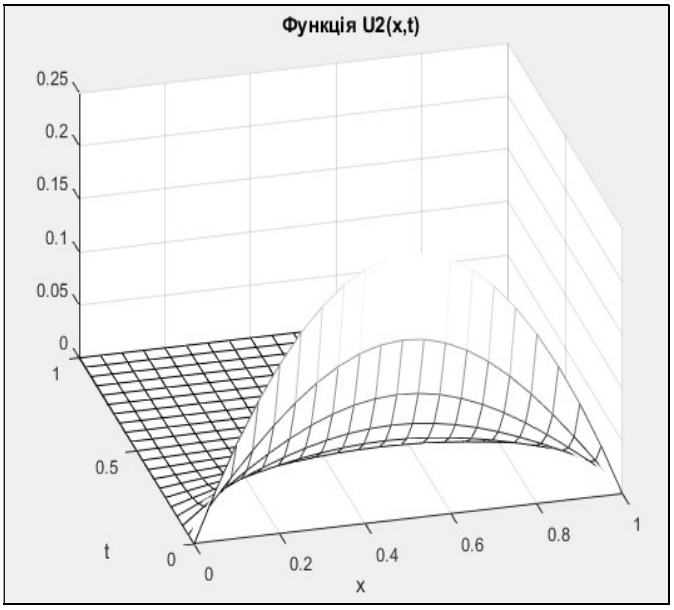


Рис. 12.2. Графік функції $U_2(x, t)$

Приклад 26. Розв'язати крайову задачу для нелінійного рівняння Бюргерса :

$$\begin{cases} \dfrac{\partial U(x,t)}{\partial t} + L \cdot \dfrac{\partial}{\partial x}\left(\dfrac{U^2(x,t)}{2}\right) = K \cdot \dfrac{\partial}{\partial x}\left(\dfrac{\partial U(x,t)}{\partial x}\right), \\ x \in (-3,3), \quad t \in (0,1]; \quad L = 0.5, \quad K = 1; \\ U(x,0) = \exp\left(-20 \cdot x^2\right); \\ U(-3,t) = 0, \quad U(3,t) = 0. \end{cases}$$

Нижче наведено текст відповідного М-файла та результати його роботи (рис. 12.3–12.4).

```
function pl12_2
%% Розв'язання крайової задачі :
% du/dt = K*u"(x,t) - 0.5*L*(u^2)',
% u(-3,t) = 0, u(3,t) = 0,
% u(x,0) = exp(-20*x^2).
% за допомогою вбудованих функцій MATLAB.
clear; clc

m = 0; % плоска задача
global K L
K = 1; L = 0.5;
xl = -3; xr = 3; x = linspace(xl, xr, 61); % сітка по x
t = linspace(0, 1, 21); % вектор по часу
% Розв'язуємо задачу
sol = pdepe(m, @koefpde, @initpde, @boundpde, x, t);
u = sol(:, :, 1);
% графік розв'язку
[X, T] = meshgrid(x, t);
mesh(X, T, u); shading interp; colormap gray;
title('Чисельний розв''язок'); xlabel('x'); ylabel('t');
view(31, 48); pause;
% проста анімація
```

```
fig = plot(x, u(1, :), 'erase', 'xor');
for k = 1 : length(t)
    set(fig, 'xdata', x, 'ydata', u(k, :));
    pause(.3);
end

function [c, f, s] = koefpde(x, t, u, DuDx)
% коефіцієнти рівняння
global K L
c = 1;
f = K .* DuDx;
s = -L .* u .* DuDx;

function u0 = initpde(x)
% початкова умова
u0 = exp(-20 .* x .* x);

function [pl, ql, pr, qr] = boundpde(xl, ul, xr, ur, t)
% коефіцієнти крайових умов
pl = ul; ql = 0;
pr = ur; qr = 0;
```

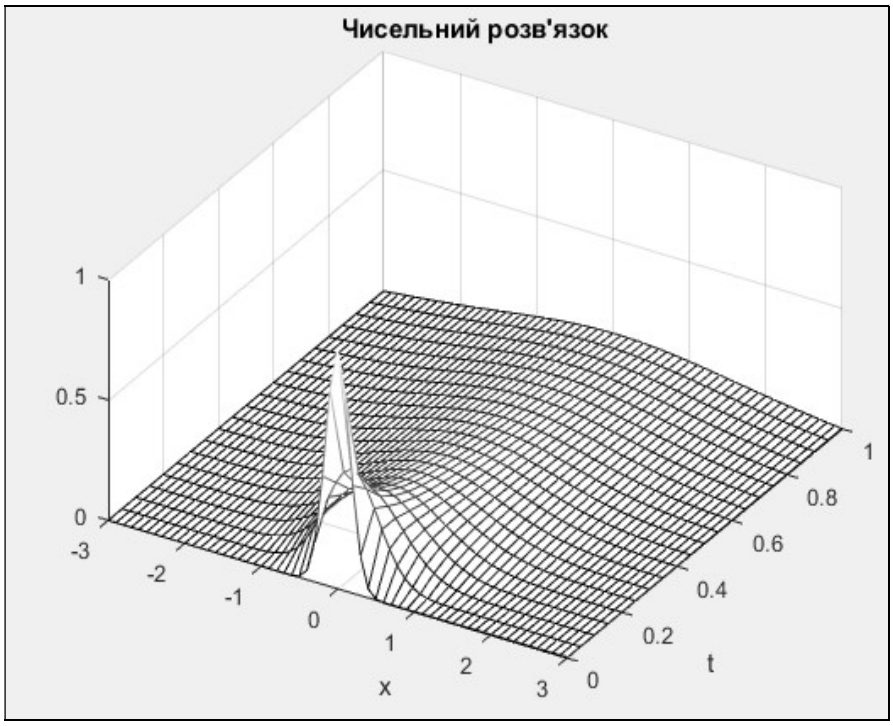


Рис.12.3. Графік наближеного розв’язку нелінійного рівняння Бюргерса

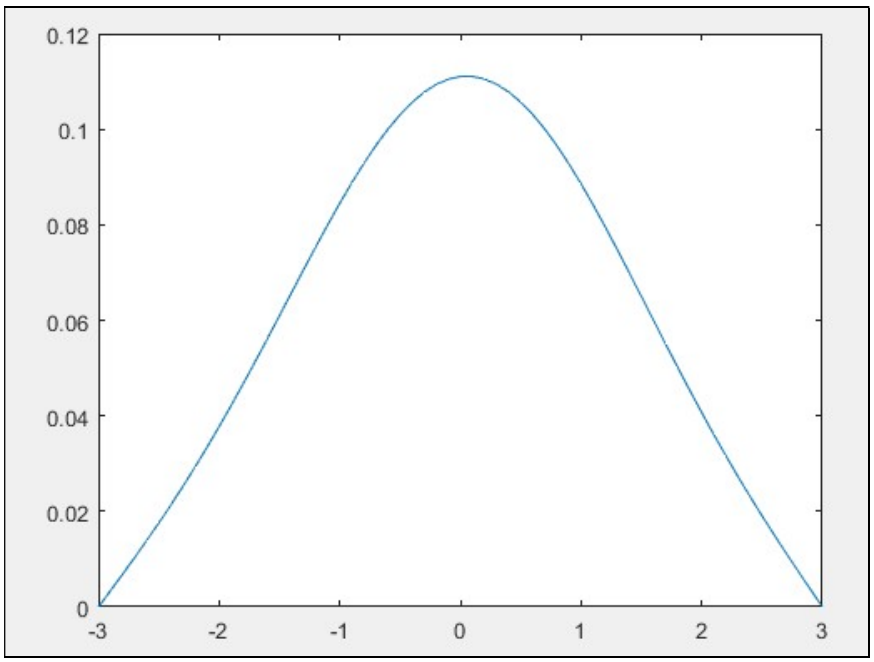


Рис.12.4. Кінцевий графік "анімації"

Приклади застосування `pdepe` можна знайти, наприклад, у посібнику [3, с. 108–116].

Математичне моделювання багатьох прикладних задач приводить до диференціальних рівнянь у частинних похідних (ДРЧП) та їх систем. Одним із найпоширеніших наближених методів їх розв'язання є метод скінченних елементів (МСЕ). Тому, крім створення власних, спеціалізованих програм, можна використовувати пакети розширення MATLAB, які багатократно збільшують ефективність використання цієї системи.

Одним із таких пакетів є пакет PDE ToolBox, призначений для розв'язання ДРЧП та їх систем за допомогою МСЕ.

Функції пакету PDE ToolBox досить складні і вимагають глибоких знань предметної області ДРЧП і МСЕ. Зупинимося на переліку деяких задач, розв'язання яких можна здійснити засобами цього пакету:

- стаціонарні і нестаціонарні задачі теплопровідності;
- задачі дифузії;
- задачі електростатики ;
- двовимірні задачі теорії пружності;
- задачі дифракції;
- розповсюдження акустичних і електромагнітних хвиль;
- знаходження власних коливань конструкцій;
- ламінарні течії рідин і газів;
- задачі теорії пластин і оболонок.

ТЕМА 13
СИМВОЛЬНІ ОБЧИСЛЕННЯ В СИСТЕМІ MATLAB

Пакет прикладних програм Symbolic Math Toolbox надає системі MATLAB принципово нові можливості – вирішення завдань у символьному (аналітичному) вигляді, реалізація точної арифметики довільної розрядності. Пакет базується на застосуванні ядра символьної математики однієї з найпотужніших систем комп'ютерної алгебри – Maple. Пакет Symbolic забезпечує: виконання символьного диференціювання та інтегрування, обчислення сум і добутків, розкладання в ряди Тейлора і Маклорена, операції з многочленами, обчислення їх коренів, розв'язання в аналітичному вигляді нелінійних рівнянь і задач для ЗДР, різноманітні символьні перетворення, підстановки і багато іншого. Пакет має команди прямого доступу до ядра системи Maple. Він дозволяє створювати процедури із синтаксисом мови програмування системи Maple і встановлювати їх у системі MATLAB.

Для ознайомлення з пакетом Symbolic можна використати наступні команди:

- *symintro* – початкове знайомство із Symbolic;
- *symcalcdemo* – демонстрація символьних обчислень;
- *symlindemo* – демонстрація застосування пакету Symbolic у задачах лінійної алгебри;
- *symvpademo* – демонстрація операцій арифметики з довільною розрядністю;
- *symrotdemo* – вивчення питань поворотів на площині;
- *symeqndemo* – демонстрація розв'язання рівнянь у символьному вигляді.

Для кожного із цих прикладів наявний М-файл, запуск якого відбувається за допомогою однієї із вказаних команд. Також можна роздрукувати ці файли (*type команда*), або взяти на редагування (*edit команда*) і копіюючи певні фрагменти команд, які зацікавили користувача, виконати їх у командному вікні.

Для отримання переліку доступних у MATLAB команд системи Symbolic Math Toolbox можна звернутися до довідкової системи або ввести команду

```
help symbolic
```

Щоб отримати довідку про символьну команду з пакета Symbolic (при збігу числових і символьних імен) треба використати префікс *sym* перед іменем команди

```
help sym/<ім'я_команди>
```

Щоб отримати довідку про команду системи Maple, необхідно звернутися до довідкової системи за допомогою спеціальної команди

```
mhelp <ім'я_команди>
```

Символьні об'єкти. Для роботи з командами системи Maple у MATLAB визначено новий тип *sym* – символьний об'єкт (symbolic object). Для проведення аналітичних операцій необхідно, щоб відповідні змінні були попередньо оголошені. Група символьних змінних може бути утворена за допомогою команди *syms*. Наприклад, команда

```
syms s1 s2;
```

описує змінні *s*1 і *s*2.

Інший спосіб опису символьного об'єкта надає команда *sym*

`>> s3=sym('s3')`	s3 = s3

Ця команда дозволяє також задати нову символьну змінну, зв'язавши її з певним виразом

`>> s4=sym(exp(s1)*s2+sqrt(2))`	*s*4 = *exp*(*s*1)*s2+2^(1/2)

У MATLAB відсутня математична нотація для подання формул, подібна тій, що використовується в системі Maple. Для виведення аналітичного виразу *s* в "математичній формі" застосовується функція `pretty(s)`, але її використання не є ефективним (в сенсі відображення формули).

Виявити символьні змінні у виразі дозволяє команда `findsym`:

`>> findsym(s4)`	*ans* = *s*1, *s*2

Для перетворення чисел у символьну форму у відповідності із вказаним параметром застосовується функція `sym` у вигляді

`sym(S,flag)`

Другий аргумент *flag* задається у такому вигляді:

'*f*' – число у форматі MATLAB (із плаваючою точкою);

'*r*' – число в раціональній формі;

'*e*' – число в раціональній формі з оцінкою відхилення від точного значення за допомогою константи *eps*;

'*d*' – число в розширеній десятковій формі із кількістю знаків, що визначається поточною величиною параметра *digits* (встановлюється функцією `digits(k)`).

Символьні операції дозволяють знаходити значення виразів і символьних констант із будь-якою точністю. Зокрема, значення раціональних дробів, функцій від цілих і раціональних аргументів, від числа π тощо можна отримати з будь-якою кількістю значущих цифр результату. Для цього використовується функція `vpa`.

Наприклад:

`>> vpa('sqrt(2)',50)`	*ans* = 1.4142135623730950488016887242096980785696718753769

Другий (необов'язковий) параметр функції `vpa` задає кількість значущих цифр, що виводяться. За замовчуванням утримуються 32 значущі цифри.

За допомогою команд `sym`, `syms` реалізується довизначення властивостей символьних змінних. Опишемо *x* як дійсну змінну `x=sym('x','real')` або `syms x real`. Для зняття накладених обмежень на можливі значення виконаємо команду `sym('x','unreal')` або `syms x unreal`.

Після оголошення символьних змінних можна утворювати символьні вирази (СВ), наприклад, використовуючи необхідні функції, операції "+", "–", "*", "/", "^" та "(", ")". Зауважимо, що MATLAB завжди залишається перш за все матричним процесором, отже для символьних змінних можна вільно застосовувати векторний і матричний запис і відповідні вбудовані оператори і функції. Так, наприклад, для створення символьної матриці необхідно створити символьні змінні, що будуть елементами матриці, а потім створити матрицю, явно визначивши її рядки і стовпці:

`>> syms a b c;` `>> A=[a b c;b c a;c a b]`	*A* = [*a*, *b*, *c*] [*b*, *c*, *a*] [*c*, *a*, *b*]

Інший спосіб побудови матриці:

`>> A=sym('[a b c;b c a;c a b]')`	$A =$ $[\ a, b, c]$ $[\ b, c, a]$ $[\ c, a, b]$

Команду *sym* можна використати також для створення СВ із абстрактною функцією (опис такої функції відсутній у поточному М-файлі або в підключених директоріях). Отриманий вираз можна використати для побудови нових СВ. Наприклад, створимо СВ, що повертає вираз правої скінченної різниці (для довільних *x*, *h*), а для числового значення цього СВ використаємо підстановку конкретних значень *x*, *h* і реальної функції *ff*.

Оформимо потрібну послідовність команд у вигляді М-функції:

`function pl13_1` `clear; clc; z = sym('ff(x)')`	$z =$ $ff\,(x)$
`syms x h;` `rdz=(subs(z,x,x+h)-z)/h`	$rdz =$ $(ff\,(x+h)–ff\,(x))/h$
`nh=0.01;nx=pi;ff=@(t)(sin(t));` `s=subs(subs(rdz,h,'nh'),x,'nx')`	$s =$ $(ff\,(nx+nh)–ff\,(nx))/nh$
`es=vpa(s)`	$es =$ $(ff\,(nx+nh)–1.*ff\,(nx))/nh$
`en=eval(s)`	$en =$ -0.99998

Символьні операції з виразами. В таблиці наведено деякі прості операції, за допомогою яких здійснюються аналітичні перетворення над символьними об'єктами.

Аналітичні операції з виразами	
Назва	**Призначення**
expand(*S*)	розкриття дужок у виразі *S*
factor(*S*)	розклад виразу *S* на множники
simplify(*S*)	спрощення виразу *S*
simple(*S*)	спрощення виразу *S* з переліком варіантів
subs(*S*,*ov*,*nv*)	заміна у виразі *S* кожного входження символьної змінної *ov* змінною *nv*
[*r*,*s*]=*subexpr*(*t*,*s*)	перетворення СВ *t* у вираз *r* за допомогою підстановки *s*
collect(*S*,*var*)	перетворення виразу *S* в поліном з обчисленням коефіцієнтів при степенях незалежної змінної *var*
[*n*,*d*]=*numden*(*S*)	перетворення виразу *S* в раціональну форму як відношення двох незвідних поліномів *n* і *d* з цілочисельними коефіцієнтами
horner(*P*)	перетворення символьного полінома *P* за схемою Горнера

Нижче наведено текст М-функції *pl13_2*, що використовує деякі з цих функції (зверніть увагу на отримані значення *sn*, *ss*):

`function pl13_2`
`clear; clc; syms x y;`
`a = 5; b = -1; c = 4;`
`S = (x+a)*(x+b)^2*(y-c)`
	$S =$
	$(x+5)*(x–1)\text{^}2*(y–4)$

`S1 = expand(S)`	$S1 =$ $x\text{^}3*y–4*x\text{^}3+3*x\text{^}2*y–12*x\text{^}2–9*x*y+36*x+5*y–20$
`S2 = subs(S1,y,1)`	$S2 =$ $–3*x\text{^}3–9*x\text{^}2+27*x–15$

`S3 = diff(S2,x)`	$S3 =$ $-9*x\text{^}2-18*x+27$
`S4 = factor(S3)`	$S4 =$ $-9*(x+3)*(x-1)$
`S5 = collect(S4,x)`	$S5 =$ $-9*x\text{^}2-18*x+27$
`S6 = horner(S5)`	$S6 =$ $27+(-18-9*x)*x$
`sn = subs(S5,x,1/3)`	$sn =$ 20
`ss = subs(S5,x,'1/3')`	$ss =$ 20

Приклад 27. Для функції $u(x)=\sin(\pi x),\ x\in[0,1]$ за m=3 точками побудувати інтерполяційний поліном у формі Лагранжа, потім перетворити його у форму Ньютона і побудувати відповідні графіки.

Нижче наведено текст відповідного М-файла

```
%%  pl13_3.m
clear; clc
syms x;
a = 0; b = 1; m = 3;
X = linspace(a, b, m); Y = sin(pi .*  X);
% Побудова інтерполяційного поліному у формі Лагранжа
w = prod(x - X); w1 = diff(w, x);
L = 0;
for k = 1 : m
   L = L + Y(k) .* (w / (x - X(k)) / subs(w1, x, X(k)));
end
L
% Побудова інтерполяційного поліному у формі Ньютона
N = expand(L); N = horner(N)
Nn = vpa(N,10)
XX = linspace(a, b, 10 * m + 1);
YL = subs(L, x, XX); YNn = subs(Nn, x, XX);
plot(XX, YL, XX, YNn, X, Y, 'b*');
legend('ІП ф.Лагранжа','ІП ф.Ньютона(ч)',...
   'точний розв''язок','Location','Best');
xlabel('x'); ylabel('sin({\pi}*x)');
grid on;
```

Отримаємо такий результат (звертаємо увагу на значення N і Nn):

```
L =
-4*x*(x-1)+4967757600021511/2028240960365167042394725128601 6*x*(x-1/2)
N =
(16225927682921335842382041026661 7/40564819207303340847894502572032-
81129638414606676728031405122553/2028240960365167042394725128601 6*x)*x
Nn =
(4.000000000-4.000000000*x)*x
```

Символьні операції математичного аналізу. Основні команди, що реалізують низку важливих операцій математичного аналізу, наведено в таблиці:

Команди математичного аналізу	
Назва	**Призначення**
diff (*S*),*diff* (*S,y*), *diff* (*S,n*),*diff* (*S,x,n*)	Обчислення *n*–тої похідної (за замовчуванням – першої) від СВ *S* за незалежною змінною *x*
jacobian(*F,v*)	Обчислення матриці Якобі для вектор-функції *F* (стовпця) за вектором змінних *v* (рядком)
int(*F*),*int*(*F,x*), *int*(*F,a,b*),*int*(*F,x,a,b*)	Обчислення інтегралу від символьної функції *F*. Тут *x*, *a*, *b* – необов'язкові змінні: *x* – змінна інтегрування, *a*, *b* – межі інтегрування
limit(*F*), *limit*(*F,a*), *limit*(*F,x,a*), *limit*(*F,x,a,'left'*), *limit*(*F,x,a,'right'*)	Обчислення границі символьно заданої функції *F*. Тут *a* – граничне значення незалежної змінної (за замовчуванням *a* = 0). Необов'язкові параметри '*left*', '*right*' указують, що обчислюється границя функції зліва або справа
symsum(*S*),*symsum*(*S,x*), *symsum*(*S,a,b*), *symsum*(*S,x,a,b*)	Обчислення суми нескінченного ряду за змінною *x*. У випадку скінченного ряду задаються початкове і кінцеве значення номера члена ряду *a* і *b*
taylor(*F*), *taylor*(*F,a*), *taylor*(*F,n*), *taylor*(*F,x*), *taylor*(*F,n,x,a*)	Розкладання символьно заданої функції *F* у ряд Тейлора. Для будь-якого варіанта розкладання можна задавати точку *a* і змінну *x*, відносно яких записується розклад, а також число членів ряду (розклад містить члени ряду до *n*–1-го порядку). За замовчуванням обчислюється розклад у точці *a* = 0 до п'ятого степеня (*n* = 6)
g=*finverse*(*f*), *g*=*finverse*(*f,x*)	Знаходження функції, оберненої до *f*, відносно заданої змінної *x*. Для функції однієї змінної *x* може бути опущено
compose(*f,g*)	Суперпозиція функцій. Результатом є функція *f* (*g*(*y*)), де *f* = *f* (*x*), *g* = *g*(*y*)

Приклад 28. За допомогою метода Ньютона знайти наближений розв'язок нелінійної СР

$$\begin{cases} x^2 y^2 = 1, \\ x - \dfrac{y}{2} = 2. \end{cases} \tag{13.1}$$

Нижче наведено текст відповідного М-файла.

```
%%  pl13_4.m
clear; clc; close;
syms x y;
global v
v = [x, y];
% Побудова графіка для системи рівнянь
F = funf(v);
ezplot(F(1),[-5,5,-5,5]); hold on;
ezplot(F(2),[-5,5,-5,5]); grid on; pause;
% Уведення наближень до коренів та їх уточнення
e = 1e-8; mki = 100;
while 1
   k = input('Наближення до кореня  № =');
   if k
      x0 = input('? x0=');
      y0 = input('? y0=');
      X = [x0; y0];
     [X, k] = njuts(@fun, X, e, mki);
     fprintf('Ітерацій =%3u Корінь = (%.8f,%.8f)\n', k, X);
   else
      break
   end
end
```

```
close;

function F = funf(v)
% Опис системи рівнянь
F = [v(1).^2.*v(2).^2-1; v(1)-v(2)/2-2];

function [F, dF] = fun(v)
% Система рівнянь і Якобіан
F = funf(v); dF = jacobian(F,v);

function [X, k] = njuts(fdf, X, e, mki)
% Метод Ньютона знаходження розв'язку системи
%   для символьно заданої нелінійної СР F(X)=0
global v
n = length(X); [F, J] = fdf(v);
for k = 1 : mki
   A = J; b = F;
   for i = 1 : n
      A = subs(A, v(i), X(i));
      b = subs(b, v(i), X(i));
   end
   d = A\b; X = X - d;
   if norm(d) < e
      break
   end
end
```

Отримаємо такий результат (показано 1-й корінь):

```
Наближення до кореня №=1
? x0=-0.5
? y0=-5
Ітерацій=  6 Корінь=(-0.22474487,-4.44948974)
```

Приклад 29. Знайти похибку апроксимації ($u_{\bar{x}x,i} - u''(x_i)$) функції $u''(x_i)$ другою різницевою похідною $u_{\bar{x}x,i} = \dfrac{u(x_i - h) - 2u(x_i) + u(x_i + h)}{h^2}$, використовуючи розклад $u(x_i \pm h)$ у ряд Тейлора по h.

```
%%  pl13_5.m
clear; clc
syms x h;
zmh = sym('u(x-h)');
z0h = sym('u(x)');
zph = sym('u(x+h)');
sm = taylor(zmh, 7, x, h);
sp = taylor(zph, 7, x, h);
Oh = simplify((sm-2*z0h+sp)/h^2-diff(z0h,x,2));
Oh = subs(Oh, x, 'Xi'); Oh = collect(Oh);
z = char(Oh); k = findstr(z,'h'); z = z(k(end-1)+3:end);
Oh = simple(sym(z))
```

Отримаємо такий результат:

```
Oh =
1/12*@@(D,4)(u)(Xi)*h^2
```

Приклад 30. Проілюструємо застосування функцій математичного аналізу.

Нижче наведено текст відповідного М-файла.

```
%%  pl13_6.m
clear; clc
syms x a b n;

% Г.М.Фихтенгольц "Курс диф.и инт.исч", т.1, стр.65
u = sqrt(n)*(sqrt(n+1)-sqrt(n))
limu = limit(u,n,inf)
clear u limu

% Г.М.Фихтенгольц "Курс диф.и инт.исч", т.1, стр.153
u = a^(1/x), ua = subs(u, a, 6);
limul = limit(ua, x, 0, 'left')
limur = limit(ua, x, 0, 'right')
u = subs(u, a, [3, 6, 13, 17, 25]);
for i = 1 : length(u)
   ezplot(u(i), [-15,15]); hold on;
end
grid on; pause; close;
clear u ua limul limur i

% Г.М.Фихтенгольц "Курс диф.и инт.исч", т.1, стр.238
u = x^2*cos(a*x)
up50 = diff(u, x, 50)
clear u up50

% Г.М.Фихтенгольц "Курс диф.и инт.исч", т.2, стр.30
u = 1/sqrt((x-a)*(b-x))
iu = simplify(int(u, x))
U = simplify(diff(iu)) % перевірка
clear u iu U

% Г.М.Фихтенгольц "Курс диф.и инт.исч", т.2, стр.136
u = sin(x)/(1+cos(x)^2)
iu = simplify(pi/2*int(u, x, 0, pi))
clear u iu

% Г.М.Фихтенгольц "Курс диф.и инт.исч", т.2, стр.325
u = n*x^(n-1)
S = symsum(u, n, 1, inf)

ezplot(S, [-15,15]); grid on; pause; close;
clear u S
% просто приклади
u = exp(x), Ou = finverse(u), S = compose(Ou, u)
ezplot(u, [-4, 4]); grid on; pause; close;
ezplot(Ou, [0.01, 4]); grid on; pause; close;
ezplot(S); grid on; pause; close;
clear u Ou S
```

Символьні операції з матрицями. Для алгебраїчних операцій із символьними об’єктами використовуються команди з іменами, аналогічними тим, що застосовуються для числових даних. Проте, виявив, що аргументом команди є символьний об’єкт, MATLAB запускає відповідну процедуру пакету Symbolic Math Toolbox.

Функції для операцій з елементами матриць	
Назва	**Призначення**
diag	створення або видобування діагоналі матриці
tril	формування лівої трикутної матриці
triu	формування правої трикутної матриці
inv	утворення оберненої матриці
det	обчислення визначника (детермінанта) квадратної матриці
rank	обчислення рангу матриці
jordan	обчислення канонічної форми Жордана
rref	зведення матриці до верхньої трикутної форми
null	нуль-простір матриці
eig	обчислення власних значень і власних векторів матриці
svd	сингулярний розклад матриці
poly	обчислення характеристичного полінома матриці
expm	обчислення матричної експоненти

Враховуючи, що ці функцій неодноразово використовувалися із числовими даними, покажемо застосування деяких з них.

Приклад 31. Знайти:

а) загальне рівняння площини в просторі, яка проходить через точки $M_i(x_i, y_i, z_i),\ i = 1, 2, 3$;

б) нормальний вектор площини;

в) відповідь на питання: чи проходить площина через точку $O(0, 0, 0)$.

Нижче наведено текст відповідного М-файла.

```
%%  pl13_7.m
clear; clc
syms x y z;
n = 3; M = [1, 2, 3; -4, 5, 6;7, -8, 9];
% Чи лежать точки на одній прямій ?
p = (M(3,:)-M(1,:))./(M(2,:)-M(1,:));
if p-p(1)==0
   disp('Точки лежать на одній прямій !')
else
   p = [x,y,z]; A = [p,1;M,ones(n,1)];
   d = det(A); P = sym(strcat(char(d),'=0'));
   disp('Рівняння площини :'); disp(P);
   N = zeros(1,n); A = eye(n); D = subs(d, p, N);
   for k = 1 : n
      N(k) = simplify(subs(d, p, A(k,:)) - D);
   end
   disp('Координати нормального вектора площини :'); disp(N);
   if D
      b = ' не';
   else
      b = '';
   end
   b = strcat('Площина',b,' проходить через O(0,0,0)');
   disp(b);
end
```

Отримаємо такий результат:

```
Рівняння площини :
48*x+48*y+32*z-240=0
Координати нормального вектора площини :
     48    48    32
Площина не проходить через O(0,0,0)
```

Приклад 32. Застосування деяких функцій для роботи з матрицями.

Нижче наведено текст відповідного М-файла.

```
%%  pl13_8.m
clear; clc
syms a;
% Показати, що Жорданові матриці
%   для матриць A і A' однакові
A = [2, 0, 1, 5; 1, 0, a, 3; 1, 0, 2, -1;1, 1, 1, 1]
Ja = jordan(A)
B = A.';
Jb = jordan(B)
if all(Ja == Jb)
    disp('Так, однакові)
else
    disp('Ні, неоднакові)
end
clear Ja Jb
% Показати, що для трикутної матриці A розмірності n
%  з нульовими діагональними елементами маємо A^n = 0
B = triu(A, 1);
C = B^length(B)
```

Розв'язання рівнянь. Для аналітичного розв'язання алгебраїчних рівнянь використовується команда `solve`, виклик якої має вигляд

```
solve(ex1,ex2,...,exN,var1,var2,...,varN)
```

Тут `ex1,ex2,...,exN` – завдання рівнянь у виглядів рядків, `var1, var2,...,varN` – невідомі (символьні змінні). При розв'язанні системи рівнянь для збереження результату створюється структура, поля якої відповідають іменам невідомих. Можна також використовувати варіант виклику з невідомими в якості вихідних параметрів:

```
[var1,var2,...,varN] = solve(ex1,ex2,...,exN)
```

Приклад 33. Знайти розв'язок :

а) системи (13.1);

б) системи

$$\begin{cases} \cos(0.4y + x^2) + x^2 + y^2 = 1.6, \\ 1.5x^2 - \dfrac{y^2}{0.36} = 1. \end{cases} \tag{13.2}$$

в) системи

$$\begin{cases} x^2 + 2y^2 - 3z^3 = 3, \\ 8x^3 - 3x^2 y + z = 77, \\ xyz^2 + 8/x = 2. \end{cases} \qquad (13.3)$$

Нижче наведено текст М-файла для знаходження розв'язків систем (13.1)–(13.3).

```
%%  pl13_9.m
clear all, close all, clc
% Система а)
ezplot('x^2*y^2-1',[-5,5]), hold on;
ezplot('x-y/2-2',[-5,5]), grid on
[x, y] = solve('x^2*y^2-1=0', 'x-y/2-2=0');
s1 = '         Розв''язок системи ';
s2 = '----------x-------|----------y-------\n';
s = strcat(s1, ' а)\n', s2);
fprintf(s);
n = length(x);
for k = 1 : n
   fprintf('%17.8f |%17.8f\n', double(x(k)), double(y(k)));
end
pause, close all
% Система б)
ezplot('cos(0.4*y+x^2)+x^2+y^2-1.6',[-2,2]), hold on
ezplot('1.5*x^2-y^2/0.36-1',[-2,2]), grid on
S = solve('cos(0.4*y+x^2)+x^2+y^2-1.6=0',...
          '1.5*x^2-y^2/0.36-1=0', 'x', 'y');
s = strcat(s1,' б)\n',s2);
fprintf(s);
n = length(S.x);
for k = 1 : n
   fprintf('%17.8f |%17.8f\n', double(S.x(k)), double(S.y(k)));
end
pause, clear all, close all
% Система в)
[x, y, z] = solve('x^2+2*y^2-3*z^3-3=0',...
                  '8*x^3-3*x^2*y+z-77=0',...
                  'x*y*z^2+8/x-2=0');
s1 = '                  Розв''язок системи в)\n';
s2 = '----------x-------|----------y-------|----------z-------\n';
s = strcat(s1, s2);
fprintf(s);
n = length(x);
for k = 1 : n
   fprintf('%17.8f |%17.8f |%17.8f\n', double(x(k)),...
           double(y(k)), double(z(k)));
end
```

Нижче наведено результат роботи М-файла *pl13_9*.

```
        Розв'язок системи а)
------------x-------|------------y-------
       1.70710678 |      -0.58578644
       0.29289322 |      -3.41421356
       2.22474487 |       0.44948974
      -0.22474487 |      -4.44948974
```

Аналізуючи графік і чисельний розв'язок системи (13.1) робимо висновок, що всі її корені знайдено.

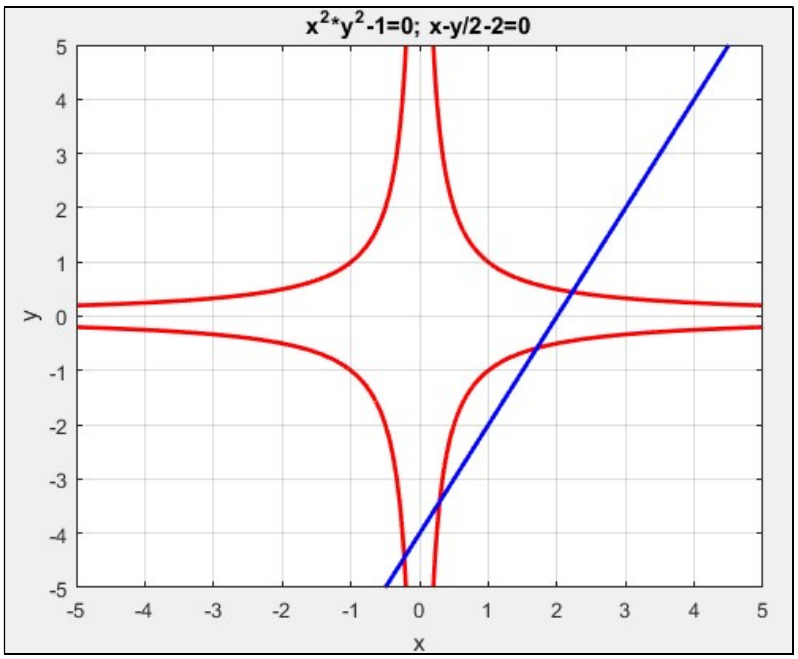


Рис. 13.1. Графічне розв’язання системи (13.1)

```
          Розв'язок системи б)
------------x-----|-----------y--------
      -0.87456548 |       -0.23027589
       0.87456548 |       -0.23027589
```

Аналіз графіка і чисельного розв’язку системи (13.2) показує, що не всі корені можуть бути знайдені вбудованою функцією *solve*.

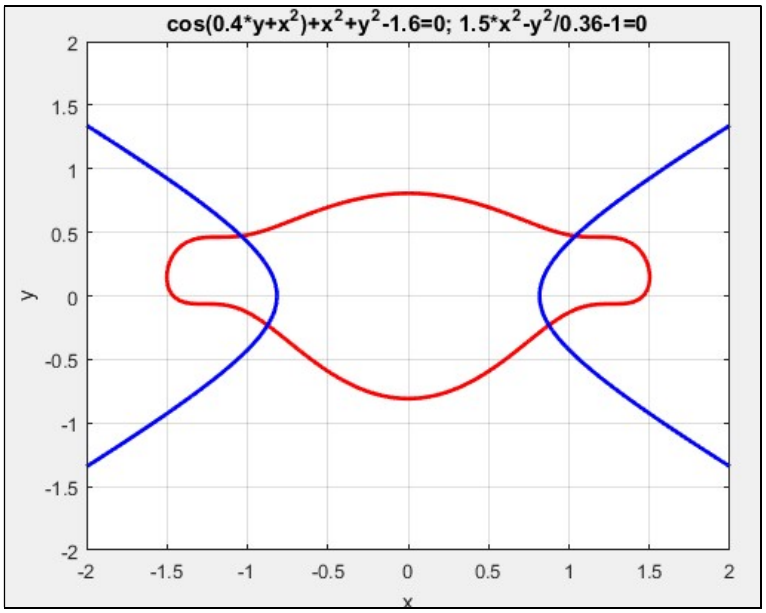


Рис. 13.2. Графічне розв’язання системи (13.2)

Частина зі знайдених розв’язків системи (13.3) (перший корінь легко перевіряється !):

```
             Розв'язок системи в)
--------x------|-----------y------|-----------z--------
     2.00000000 |      -1.00000000 |       1.00000000
     0.00127683 |    -827.46179873 |      76.99595295
     2.21246671 |       0.64475439 |      -0.32685410
     2.05779696 |      -0.56761935 |      -0.78278359
```

Розв'язання звичайних диференціальних рівнянь і систем звичайних диференціальних рівнянь. Для розв'язання ЗДР і систем ЗДР в MATLAB використовується функція *dsolve*. Команда *dsolve(eq1,eq2,…)* знаходить аналітичний розв'язок СДР із початковими умовами. Вони задаються рівностями *eq1,eq2,…* (спочатку вказуються рівняння із символьних змінних або рядки, що задають рівняння, потім – початкові умови). За замовчуванням незалежною змінною вважається *t*, можна використовувати й іншу змінну, додавши її в кінець списку параметрів команди *dsolve*. Для позначення похідних у рівняннях використовується символ *D*, комбінації *D2,D3,…* застосовуються для позначення другої, третьої і наступних похідних. Початкові умови задаються у вигляді рівностей *'x(a)=b'* або *'Dx(a)=b'*, де *x* – незалежна змінна, *a* і *b* – константи. Якщо кількість початкових умов менша, ніж кількість ДР, то в розв'язку будуть присутні довільні константи *C1,C2,….* . Розв'язок (вихідний параметр) представляється структурою з полями за числом невідомих функцій.

Нижче наведено приклади застосування функції *dsolve* для розв'язання систем ЗДР.

```
%%  pl13_10.m
clear; clc
%   A.M.Самойленко,С.А.Кривошея,Н.А.Перестюк
% "Дифференциальные уравнения:примеры и задачи",стр.172 Пр.2.72
S = dsolve('Dy=u','Du=v','Dv=3/t*v-6/t^2*u+6/t^3*y+sqrt(t)'),
S.y

%    -#- ,стр.128 Пр.2.16
S = dsolve('Dy=u','Du=-(u^2+u^4)/(2*y)','y(0)=1','u(0)=2');
simplify(S.y)

%    -#- ,стр.276 Пр.4.31
S = dsolve('Dx=1-2/t*x','Dy=(1+2/t)*x+y-1');
S.x
S.y
```

Отримано наступні результати (зазначимо, що для Пр.2.16 знайдено і комплексний розв'язок):

```
ans =
8/15*t^(7/2)+1/6*t^3*C1+1/2*t^2*C2+t*C3
ans =
4/5+1/5*(15*t+1)^(2/3)
1/2*(16+(-84*t-12*i*t*3^(1/2)-8)^(2/3))/(7+i*3^(1/2))
ans =
1/3*t+1/t^2*C2
ans =
-1/3*(t^3+3*C2-3*exp(t)*C1*t^2)/t^2
```

Засоби візуалізації символьних обчислень. Розширення MATLAB командами Maple призвело до появи нових графічних функцій :

Графічні команди пакета Symbolic Math	
Назва	**Призначення**
ezplot	побудова графіка
funtool	графічний калькулятор функції однієї змінної
ezpolar	побудова графіка в полярних координатах
ezcontour	побудова контурних графіків (ліній рівня)
ezcontourf	побудова контурних графіків (ліній рівня) з заповненням
ezmesh	побудова сітчастої поверхні
ezmeshc	побудова сітчастої поверхні з проекцією ліній рівня
ezsurf	побудова поверхні
ezsurfc	побудова поверхні з проекцією ліній рівня
ezplot3	побудова тривимірних кривих

Деякими з цих функцій (`ezplot`) ми активно користувалися у попередніх прикладах. Використання інших вбудованих функцій можна знайти в посібнику [3]. Серед цих функцій є інтерактивна – графічний калькулятор функцій однієї змінної. Виклик `funtool` виводить три графічні вікна: в перших двох відбувається виведення графіків функцій *f*, *g*. Третє вікно містить інтерфейс користувача, а саме

- рядки введення:
  - опису функцій *f*, *g*,
  - інтервалу представлення незалежної змінної *x*,
  - параметра перетворень *a*;
- систему кнопок для виконання різних операцій побудови і перетворення функцій.

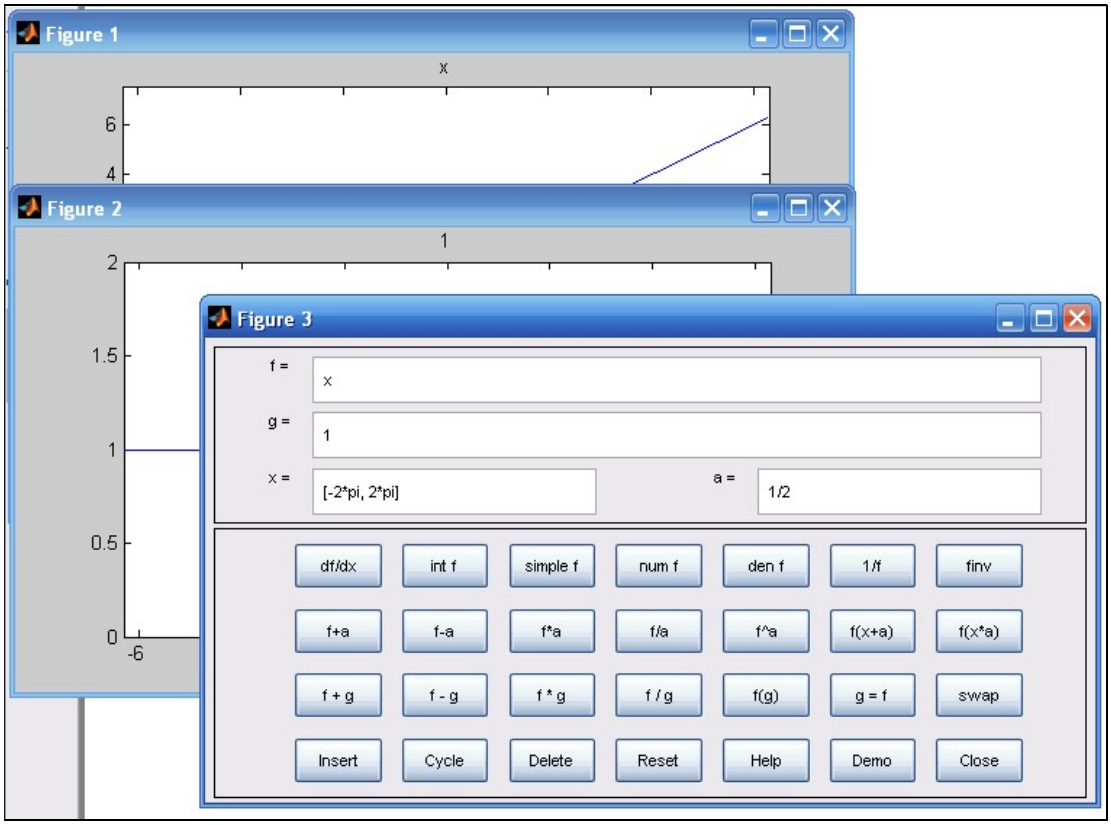


Рис. 13.3. Вікна графічного калькулятора

КОНТРОЛЬНІ ЗАПИТАННЯ І ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ ЗА МАТЕРІАЛАМИ ТЕМ 1–13

Тема 1

1. Дати означення ідентифікатора, указати для яких цілей його використовують. Навести приклади допустимих і недопустимих ідентифікаторів.
2. Навести структуру типів даних MATLAB.
3. Перерахувати основні класи даних у системі MATLAB.
4. Навести приклади допустимого запису числових констант.
5. Навести можливі варіанти запису комплексних чисел.
6. Навести приклади імен функцій перетворення числового типу.
7. Навести приклади запису логічних констант і створення значень логічних змінних.
8. Указати правила запису констант рядкового типу та особливості зберігання рядків у масивах.
9. Записати букви β, γ, λ, μ, σ у вигляді констант рядкового типу (вказівка: скористатися малими грецькими літерами в системі LaTex).
10. Записати інформацію (про себе) у вигляді структури з полями: Прізвище, Ім'я, По батькові, Дата народження, Стать.
11. Записати попередню інформацію (п.10) у вигляді класу `cell`.
12. Пояснити різницю зберігання рядків у звичайному масиві і в масиві комірок.
13. Пояснити, як можна ідентифікувати клас об'єкту в MATLAB.
14. Пояснити, якими характеристиками відрізняються звичайні і системні змінні.
15. Чи правильно (синтаксично) записано вираз

$$(A*b-c)+2.45-\sin(x)<0.5\&b==c|\exp(x)>1\ ?$$

16. Якщо відповідь у п.15 ствердна, то вказати пріоритет виконання операцій.
17. Перерахувати основні групи вбудованих функцій.
18. Як можна визначити список елементарних функцій?
19. Як можна визначити список спеціальних функцій?
20. Як можна деталізувати інформацію про конкретну вбудовану функцію?

Тема 2

1. Пояснити синтаксичні правила запису ланцюга команд в MATLAB.
2. Пояснити синтаксис і семантику оператора присвоєння в MATLAB.
3. Написати приклад оператора розгалуження в MATLAB.
4. Написати приклад спрощеного оператора розгалуження в MATLAB.

5. За допомогою оператора каскадного розгалуження записати обчислення

$$y=\begin{cases}0, & |x|\geq 2,\\ -x, & -1\leq x\leq 1,\\ x+2, & -2<x<-1,\\ 2-x, & 1<x<2.\end{cases}$$

6. Для змінної $G\in[1,12]$ надрукувати відповідні знаки гороскопу, використовуючи оператор вибору в MATLAB.
7. Пояснити синтаксис і семантику оператора циклу з передумовою в MATLAB.
8. Пояснити синтаксис і семантику оператора циклу за діапазоном значень в MATLAB.
9. Навести приклад запису оператора введення в MATLAB.
10. Записати приклади варіантів виведення інформації в MATLAB.

Тема 3

1. Нехай виконуються умови теореми 3.2 з теми 3. Записати алгоритм методу Холецького знаходження матриць L, U, де $A=L\cdot U$, використовуючи рекурентний процес [19, с. 56–57] .
2. Записати розрахункові формули методу квадратного кореня для симетричної матриці $A=A'$ [19, с. 71–72].
3. Написати М-файл *tem31.m* для знаходження $\min\limits_i|\lambda_i(A)|$, використовуючи степеневий ітераційний метод ($A\neq A'$).
4. Написати М-файл *tem32.m* для знаходження $\max\limits_i|\lambda_i(A)|$, використовуючи метод скалярних добутків ($A=A'$).

Тема 4

1. Нехай виконуються умови теореми 3.2 з теми 3. Написати М-файл *tem41.m* для знаходження розв'язку СЛАР $A\vec{x}=\vec{b}$ з реалізацією методу Холецького знаходження матриць L, U, де $A=L\cdot U$, використовуючи рекурентний процес [19, с. 56–57] .
2. Написати М-файл *tem42.m* реалізації методу квадратного кореня для ермітової матриці $A=A^*$ [19, с. 71–72].

Тема 5

1. Чи будуть інтерполяційні поліноми, побудовані за формулами (5.2)–(5.3); (5.4) і (5.7) однаковими ?

2. Задано два вектори $\vec{X},\vec{Y}$ – вузли і значення функції *f*, для якої $f(X_i)=Y_i$. Чи можна побудувати інтерполяційний поліном у формі Ньютона не використовуючи додаткових векторів ?
3. Для правої різницевої похідної $u_{x,i}$, центральної різницевої похідної $u_{x,i}$ і другої різницевої похідної $u_{\bar{x}x,i}$, використовуючи формулу Тейлора, знайти вираз локальної похибки

$$\psi_h(x_i)=\left|\frac{u(x_{i+1})-u(x_i)}{h}-u'(x_i)\right|,\quad \psi_h(x_i)=\left|\frac{u(x_{i+1})-u(x_{i-1})}{2h}-u'(x_i)\right|,$$

$$\psi_h(x_i)=\left|\frac{u(x_{i-1})-2u(x_i)+u(x_{i+1})}{h^2}-u''(x_i)\right|$$

в точці x_i.

4. Застосувати формули Рунге-Ромберга до квадратурної формули (КФ) трапецій. Чи отримаємо теж КФ ? Якщо "так", то яку ?
5. Для КФ центральних прямокутників, використовуючи формулу Тейлора, знайти вираз локальної похибки

$$\psi_h(x_{i-0.5})=\left|hf(x_{i-0.5})-\int_{x_{i-1}}^{x_i}f(x)dx\right|,\quad h=x_i-x_{i-1},\quad x_{i-0.5}=0.5(x_{i-1}+x_i).$$

Тема 6

1. Що спільного і відмінного в означеннях функцій $S_{m,q}(x)$ і $S_{m,q}(x;u)$?
2. Задано два вектори $\vec{X},\vec{Y}$ – вузли і значення функції *f*, для якої $f(X_i)=Y_i$. Написати М-файл *tem62.m* для побудови інтерполяційного сплайна $S_{0,1}(x;u)$.
3. Записати розрахункові формули інтерполяційного сплайна $S_{1,1}(x;u)$ у представленнях:
   a) "сплайнове";
   b) "поліноміальне";
   c) у формі "лагранжіана".
4. Написати М-файл *tem64.m* для наближення на відрізку $x\in[0,\pi]$ функції $f(x)=\exp(-0.1x^2)\cos(2x)$ сплайнами $S_{0,1}(x;u)$, $S_{1,1}(x;u)$, $S_{3,1}(x;u)$, використовуючи вбудовані функції MATLAB.

Тема 7

1. Який синтаксис має заголовок функції при :
   a) відсутності вхідних параметрів;
   b) відсутності вихідних параметрів;
   c) збігу вхідного параметра *X* і вихідного параметра *X*;
   d) наявності тільки одного вхідного параметра *X* і тільки одного вихідного параметра *Y*.
2. Як із командного вікна відобразити інформацію:
   a) розміщену в тексті коментаря М-функції з ім'ям *myfun*?

b) щодо тексту М-функції з ім'ям *myfun*?
c) отриману при пошуку в робочій директорії М-функції з ім'ям *myfun*?

3. Навести приклад опису функції за допомогою:
   a) файла М-функції з ім'ям *myfun*;
   b) анонімної функції з ім'ям *myfun*;
   c) inline-функції з ім'ям *myfun*.
4. Указати відмінності опису і виклику підфункції і вкладеної функції.
5. Навести приклади існуючих варіантів:
   a) використання імен функцій як аргументів при виклику функцій;
   b) виклику довільної функції MATLAB.

Тема 8

1. Написати алгоритм методу Якобі у покомпонентній формі запису.
2. Написати алгоритм методу Зейделя у покомпонентній формі запису.
3. За допомогою функції знаходження мінімуму функції однієї змінної знайти дійсні корені рівняння $x^3 - 3.2x + 1.5 = 0$.
4. Розв'язати рівняння $2x^3 - 3.1x^2 + 1.4 = 0$ за допомогою:
   a) функції `fzero`;
   b) функції `roots`.
5. За допомогою функції знаходження мінімуму функції декількох змінних знайти розв'язки системи рівнянь

$$\begin{cases} \cos(0.4y + x^2) + x^2 + y^2 = 1.6, \\ 1.5x^2 - \dfrac{y^2}{0.36} = 1. \end{cases}$$

Тема 9

1. Знайти розв'язок задачі Коші

$$\begin{cases} \dfrac{dy}{dx} = -\dfrac{2xy^2}{1+x^2} + \dfrac{x^2}{1+y^2x}, x \in (0,2], \\ y(0) = 0.5 \end{cases}$$

   неявним методом Ейлера з рівномірним кроком $h = 0.01$.
2. Задачу Коші з п.1 розв'язати симетричною схемою.
3. Розв'язати задачу Коші з п.1, використовуючи солвер MATLAB `ode23`. Побудувати графіки розв'язків задач із пп.1–3 на одному рисунку.
4. Знайти вектор-функцію $\vec{U}(x) = (u_1(x), u_2(x))^T$, розв'язавши задачу Коші

$$\begin{cases} \vec{U}'(x) = A * \vec{U}, \quad x \in (0, 2], \\ \vec{U}(0) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \end{cases} \quad \text{де} \quad A = \begin{pmatrix} 998 & 1998 \\ -999 & -1999 \end{pmatrix}.$$

Побудувати для порівняння на одному рисунку графіки розв'язків $u_1(x)$ і $u_2(x)$, отриманих солверами `ode45` і `ode23s`. Для матриці A знайти число обумовленості.

Тема 10

1. Побудувати графіки розв'язку і фазових траекторій для задачі Коші

$$\begin{cases} \vec{U}'(x) = A * \vec{U}, \quad x \in (0,2], \\ \vec{U}(0) = \begin{pmatrix} 0.1 \\ 1 \end{pmatrix}, \end{cases} \quad \text{де} \quad A = \begin{pmatrix} 998 & 1998 \\ -999 & -1999 \end{pmatrix},$$

використовуючи солвер `ode23s`.

2. Побудувати графік функції і лінії рівня для

$$z(x,y) = \frac{\sin(x)\sin(y)}{xy}, \quad x \in [-10,10], \; y \in [-10,10].$$

Тема 11

1. Використовуючи солвер `bvp4c`, розв'язати і побудувати графік розв'язку для крайової задачі

$$\begin{cases} \vec{U}'(x) = A * \vec{U}, \quad x \in (0,2), \\ u_1(0) = 0, u_2(2) = 2, \end{cases} \quad \text{де} \quad A = \begin{pmatrix} 3 & -1 \\ -1 & 19 \end{pmatrix}.$$

2. Використовуючи солвер `bvp4c`, розв'язати і побудувати графік розв'язку для крайової задачі

$$\begin{cases} u^{(4)}(x) - x^2 u'(x) + 4u(x) = \sin(1-x), \quad x \in (0,1), \\ u(0) = 2, \quad u(1) = 1, \quad u'(1) = 2, \quad u''(1) = 0. \end{cases}$$

Тема 12

1. Використовуючи солвер `pdepe`, розв'язати і побудувати графік розв'язку з проекцією ліній рівня для крайової задачі

$$\begin{cases} \dfrac{\partial}{\partial x}\left(\dfrac{\partial U_1(x,t)}{\partial x}\right) + 3U_1(x,t) = 2U_2(x,t) + \sin(x-t), \\ \dfrac{\partial U_2(x,t)}{\partial t} = \dfrac{\partial}{\partial x}\left(\dfrac{\partial U_2(x,t)}{\partial x}\right) + U_1(x,t)U_2(x,t)\exp(x-t), \\ x \in (0,1), \quad t \in (0,1], \\ U_1(0,t) = 0, \quad U_1(1,t) = 1-t, \\ U_2(0,t) = t, \quad U_2(1,t) = 0, \quad t \in [0,1], \\ U_2(x,0) = 0.5 - |0.5 - x|, \quad x \in [0,1]. \end{cases}$$

Тема 13

Написати відповідні М-файли, використовуючи символьні обчислення

1. Створити символьний вираз, що повертає другу скінченну різницю для довільних $x, h, f(x)$, а для отримання числового значення цього виразу використати підстановку $x = 1$, $h = 0.05$, $f(x) = \cos(x)$.
2. Задано деякі символьні поліноми $P_N(y)$, $Q_M(x)$. Виконати підстановку $y = Q_M(x-5)$ у поліном $P_N(y)$. Для отриманого символьного виразу знайти похідну і подати її у формі полінома з коефіцієнтами при степенях незалежної змінної і у формі Горнера.
3. Знайти похибку апроксимації $(u_{x,i} - u'(x_i))$ функції $u'(x_i)$ правою різницевою похідною $u_{x,i} = \dfrac{u(x_i + h) - u(x_i)}{h}$, використовуючи розклад $u(x_i + h)$ у ряд Тейлора по h.
4. Задано символьну змінну x і числову матрицю $A = \begin{pmatrix} 1 & -1 & 1 \\ 1 & 1 & -1 \\ 2 & -1 & 0 \end{pmatrix}$. Створити матрицю для отримання характеристичного рівняння (ХР). Знайти ХР за допомогою функції `det( )`. Для знаходження коренів ХР використати розклад полінома на множники.
5. Знайти графічний розв'язок і корені системи $\begin{cases} 41x^2 + 3y^2 = 123, \\ 6x^2 + 31y^2 = 186. \end{cases}$
6. Знайти загальний розв'язок ДР $y'' + y = 4\sin(x)$.
7. Побудувати графіки для розв'язку ДР з п.6 при $C_1 = 1, C_2 = 1$ і $C_1 = 0, C_2 = 0$.
8. Побудувати для розв'язку ДР з п.6 фазові траекторії при $C_1 = C_2 = 0$.

СПИСОК ЛІТЕРАТУРИ

1. *Ануфриев И. Е.* MATLAB 7 / И. Е. Ануфриев, А. Б. Смирнов, Е. Н. Смирнова. – СПб.: БХВ-Петербург, 2005. – 1104 с.
2. *Вакал Є. С.* Класифікація рівнянь із частинними похідними з використанням системи MATLAB / Є. С. Вакал, Ю. Є. Вакал. – К.: Основа, 2017. – 104 с.
3. Використання математичного пакета MATLAB для розв'язування прикладних задач : навч. посіб. / Б. П. Довгий, Є. С. Вакал, Ю. Є. Вакал, А. В. Попов. – К.: Фітосоціоцентр, 2012. – 78 с.
4. *Довгий Б.П.* Сплайн-функції та їхнє застосування : навч. посіб. / Б. П. Довгий, А. В. Ловейкін, Є. С. Вакал, Ю. Є. Вакал. – К.: ВПЦ "Київський університет", 2017. – 120 с.
5. *Довгий Б.П.* Методи обчислень : методичні вказівки до лабораторних робіт із використанням пакета MATLAB / Б. П. Довгий, Є. С. Вакал, Ю. Є. Вакал. – К.: ВПЦ "Київський університет", 2017. – 60 с.
6. *Завьялов Ю. С.* Методы сплайн-функций / Ю. С. Завьялов, Б. И. Квасов, В. Л. Мирошниченко. – М.: Наука, 1980. – 352 с.
7. *Дьяконов В. П.* Математические пакеты расширения MATLAB : специальный справочник / В. П. Дьяконов, В. В. Круглов. – СПб.: Питер, 2001. – 480 с.
8. *Кетков Ю. Л.* MATLAB 7: программирование, численные методы / Ю. Л. Кетков, А. Ю. Кетков, М. М. Шульц. – СПб.: БХВ-Петербург, 2005. – 752 с.
9. *Кривилев А. В.* Основы компьютерной математики с использованием системы MATLAB / А. В. Кривилев. – М.: Лекс-Книга, 2005. – 496 с.
10. *Лазарев Ю. Ф.* Моделирование процессов и систем в MATLAB / Ю. Ф. Лазарев. – СПб. : Питер; Киев: Изд. группа BHV, 2005. – 512 с.
11. *Ляшко И.И.* Методы вычислений / И.И. Ляшко, В.Л. Макаров, А.А. Скоробогатько. – К.: Вища школа, 1977. – 408 с.
12. *Мартынов Н. Н.* Введение в MATLAB 7 / Н. Н. Мартынов. – М.: КУДИЦ-ОБРАЗ, 2005. – 416 с.
13. *Перестюк М. О.* Збірник задач з математичної фізики / М. О. Перестюк, В. В. Маринець, В. Л. Рего. – Кам'янець-Подільський: Аксіома, 2012. – 252 с.
14. *Половко А. М.* MATLAB для студента / А. М. Половко, П. Н. Бутусов. – СПб.: БХВ-Петербург, 2005. – 321 с.
15. *Попов В. В.* Методи обчислень : конспект лекцій для студентів механіко-математичного факультету / В. В. Попов. – К.: ВПЦ "Київський університет", 2012. – 303 с.
16. *Попов В. В.* Лабораторні роботи та домашні завдання для самостійної роботи з дисципліни "Методи обчислень" для студ. мех.-мат. ф-ту / В. В. Попов, Б. П. Довгий, Є. С. Вакал. – К.: Укр. фітосоціолог. центр, 2006. – 32 с.
17. *Потемкин В. Г.* Справочник по MATLAB [Електронный ресурс] / В. Г. Потемкин. – Режим доступа : *http://www.nsu.ru/matlab/ MatLab_RU/ ml/book2/index.asp.htm* (15.01.2003).
18. *Самарский А. А.* Теория разностных схем / А. А. Самарский. – М.: Наука, 1977. – 656 с.
19. *Самарский А. А.* Численные методы / А. А. Самарский, А. В. Гулин. – М.: Наука, 1989 – 432 с.
20. *Стечкин С. Б.* Сплайны в вычислительной математике / С. Б. Стечкин, Ю. Н. Субботин. – М.: Наука, 1976. – 248 с.

ЗМІСТ

Навчальне видання

ДОВГИЙ Борис Павлович
ВАКАЛ Євген Сергійович

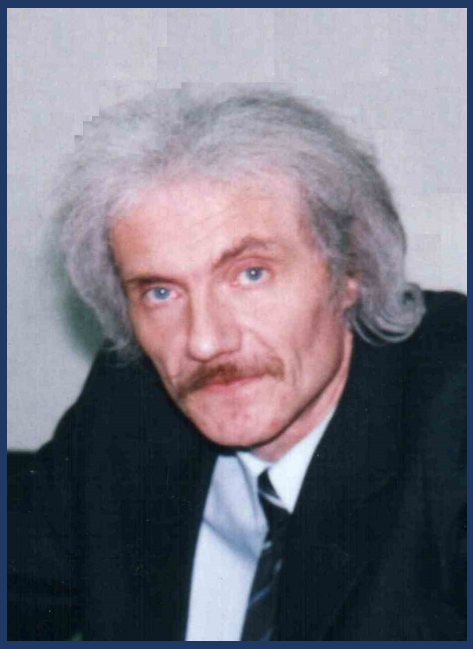
ІНФОРМАЦІЙНІ СИСТЕМИ ТА ТЕХНОЛОГІЇ

Навчальний посібник
для студентів механіко-математичного-факультету

Підписано до друку 12.01.21
Формат 60х84$^{1/8}$. Папір офсетний.
Гарнітура: Times
Умовн. друк. арк.: 13,86. Обл.-вид. арк.:9,45
Наклад: 300 прим. Замовлення №П-2021-04
Віддруковано з оригіналів замовника.

Видавник і виготовлювач ФО-П Кравченко Я.О.
Свідоцтво №ДК6078 від 13.03.2018 р.
02100, м. Київ, вул. Будівельників, буд. 32/2
e-mail: 5619531@ukr.net www.metodichka.in.ua

ДОВГИЙ Борис Павлович – доцент, кандидат фізико-математичних наук, доцент кафедри математичної фізики Київського національного університету імені Тараса Шевченка. Наукові інтереси пов'язанні із застосуванням методів обчислювальної математики до розв'язання крайових задач нелінійної оптики та прикладних задач економічної теорії. Читає лекції з нормативних курсів "Методи обчислень", "Інформатика і програмування", "Інформаційні системи і технології" та низки спеціальних курсів з інформаційних технологій і обчислювальних методів. Автор понад 60 наукових та навчально-методичних праць.



ВАКАЛ Євген Сергійович – доцент, кандидат фізико-математичних наук, доцент кафедри математичної фізики Київського національного університету імені Тараса Шевченка. Основні напрямки наукової діяльності стосуються сучасних інформаційних технологій, теорії крайових задач для нелінійних рівнянь з розривними розв'язками та числових методів. Читає лекції з нормативних курсів "Рівняння математичної фізики", "Методи математичної фізики", "Інформаційні системи і технології" та низки спеціальних курсів, пов'язаних з теорією наближених методів та новітніми розробками в галузі інформатики. Автор понад 100 наукових та навчально-методичних праць.