Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Харківський національний університет радіоелектроніки

В. І. Коробов
Ю. В. Ромашов
К. В. Стєпанова

ОСНОВИ ПРОГРАМУВАННЯ НАУКОВИХ ТА ІНЖЕНЕРНИХ РОЗРАХУНКІВ

Навчальний посібник

Харків – 2023

УДК 519.6: 519.7: 621.039: 681.3
К 68

Рецензенти:

С. В. Альохіна – доктор технічних наук, старший науковий співробітник відділу моделювання та ідентифікації теплових процесів Інституту проблем машинобудування ім. А.М. Підгорного НАН України;

Г. М. Жолткевич – доктор технічних наук, декан факультету математики і інформатики Харківського національного університету імені В. Н. Каразіна;

О. М. Цимбал – доктор технічних наук, професор кафедри комп'ютерно-інтегрованих технологій, автоматизації та мехатроніки Харківського національного університету радіоелектроніки.

Затверджено до друку рішенням Вченої ради
Харківського національного університету імені В. Н. Каразіна
(протокол № 2 від 24 січня 2022 року)

Коробов В. І.

К 68 Основи програмування наукових та інженерних розрахунків : навчальний посібник / В. І. Коробов, Ю. В. Ромашов, К. В. Стєпанова. – Харків : ХНУ імені В. Н. Каразіна, 2023. – 172 с.

Посібник підготовлений відповідно до основних положень програм з навчальної дисципліни «Програмування». Обговорюються базові поняття та концепції програмування, які ілюструються на прикладах наукових та інженерних розрахунків щодо атомної енергетики та їхньої автоматизації.

Посібник призначений для студентів різних курсів та рівнів вищої освіти спеціальностей 113 «Прикладна математика», 142 «Енергетичне машинобудування», 143 «Атомна енергетика», 151 «Автоматизація та комп'ютерно-інтегровані технології».

УДК 519.6: 519.7: 621.039: 681.3

© Харківський національний університет імені В. Н. Каразіна, 2023
© Коробов В. І., Ромашов Ю. В., Стєпанова К. В., 2023
© Дончик І. М., макет обкладинки, 2023

Навчальне видання

Коробов Валерій Іванович
Ромашов Юрій Володимирович
Стєпанова Катерина Вадимівна

ОСНОВИ ПРОГРАМУВАННЯ
НАУКОВИХ ТА ІНЖЕНЕРНИХ РОЗРАХУНКІВ

Навчальний посібник

Коректор *О. В. Анцибора*
Комп'ютерне верстання *В. В. Савінкова*
Макет обкладинки *І. М. Дончик*

Формат 60x84/16. Ум. друк. арк. 9,6. Наклад 100 пр. Зам. № 351/21.

Видавець і виготовлювач
Харківський національний університет імені В. Н. Каразіна,
61022, м. Харків, майдан Свободи, 4.
Свідоцтво суб'єкта видавничої справи ДК No 3367 від 13.01.2009
Видавництво ХНУ імені В. Н. Каразіна
Тел. 705-24-32

ЗМІСТ

ВСТУП

Ефективне використання можливостей сучасних комп'ютерів в сучасній індустрії забезпечується завдяки використанню відповідного програмного забезпечення, тому створення програм є актуальною задачею сьогодення. Індустрія створення комп'ютерних програм сьогодні є однією із найвагоміших галузей сучасної економіки розвинених країн світу.

До розробки програм різноманітного призначення залучають фахівців з прикладної математики, оскільки програми природно є ідеальними нематеріальними об'єктами, в яких використовуються абстрактні поняття, що мають математичне походження. Природно, що участь фахівців з прикладної математики у розробці програм можлива лише за умов володіння ними базовими поняттями, концепціями та прийомами програмування, які відповідають різноманітним завданням, для розв'язування яких створюються програми. Різноманіття концепцій та прийомів програмування дозволяє засвоїти їх лише протягом багаторічного навчання. Метою курсу програмування, що викладається на першому курсі для студентів, що навчаються в університеті за спеціальністю 113 «Прикладна математика», є ознайомлення із базовими поняттями, концепціями та прийомами програмування математичних обчислень, які історично є первинними та складають основу побудови алгоритмів окремих складових програм, а також сучасних концепцій програмування. Відповідно до цієї мети при вивченні курсу використовується мова програмування FORTRAN, яка була створена саме для математичних обчислень та широко використовується у розвинених країнах сьогодні. Даний курс також рекомендується для студентів спеціальностей 142 «Енергетичне машинобудування» та 143 «Атомна енергетика», а також спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології» для підготовки щодо програмування і автоматизації наукових та інженерних розрахунків.

1 ЗАГАЛЬНІ ПОНЯТТЯ ЩОДО ПРОГРАМУВАННЯ

Кожна галузь знань, включаючи й програмування, спирається на деякі загальні поняття та концепції, з використанням яких формулюються усі часткові результати, які в програмуванні мають вигляд прийомів програмування для розв'язування типових завдань щодо розробки програм.

1.1 Створення простішої програми для розрахунку

Комп'ютери та наявні в них програми є дуже розповсюдженими сьогодні і використовуються навіть у побуті, але сучасні буденні знання про комп'ютери та програми є недостатніми для розуміння вузькопрофесійних питань щодо прикладної математики, тому розглянемо далі питання про комп'ютери, комп'ютерні програми та мови програмування.

1.1.1 Загальні поняття про комп'ютери та мови програмування, Створення програм потребує наявності певних уявлень про комп'ютери та мови програмування.

1.1.1.1 Для розуміння, що являє собою комп'ютер без зайвих технічних подробиць щодо його конструктивної будови, слід володіти поняттям про інформацію. *Інформація* – це відомості про будь-які явища та процеси; *повідомленням* називають інформацію, що зафіксована в матеріальній формі [1]. Повідомлення представляють на паперових носіях, а також у вигляді файлів різних форматів, що розміщені на електронних носіях, жорстких дисках або флеш-картках тощо. *Неперервним (аналоговим) повідомленням* називають повідомлення, яке представлено у вигляді зміни з часом деякої фізичної величини, що характеризує розглядуваний процес [1]. В якості фізичної величини для створення повідомлення переважно використовують електричний струм. *Дискретним повідомленням* називають повідомлення, які формуються різними послідовностями елементів із фіксованого скінченного набору; дискретні повідомлення незалежно від характеру інформації в них можуть бути представлені в еквівалентній *цифровій формі* – формі представлення (кодування) інформації у послідовності цифр. Неперервні повідомлення можуть бути представлені з наперед заданою точністю у цифровій формі; *квантуванням* називають процес перетворення неперервного повідомлення в дискретну форму, що можна виконати із заданою точністю (похибкою).

Електронна обчислювальна машина (ЕОМ) – це електронний пристрій, призначений для перетворювання інформації. ЕОМ, що перетворюють безперервну інформацію, називають *електронними аналоговими обчислювальними машинами* (ЕАОМ); ЕОМ, що перетворюють цифрову інформацію, називають *електронними цифровими обчислювальними машинами* (ЕЦОМ). Через універсальність цифрової форми представлення інформації та суттєвий розвиток елементної бази ЕЦОМ мають більші можливості щодо впровадження та через це є суттєво більш розповсюдженими. Сьогодні під поняттям «комп'ютер» розуміють у першу чергу персональні ЕЦОМ, які конструктивно пристосовані для індивідуального користування.

1.1.1.2 Можливості комп'ютерів щодо ефективного перетворення цифрової інформації обумовлені не тільки високою швидкістю виконання дій над числами та великими обсягами пам'яті для зберігання проміжних результатів, а у першу чергу завдяки можливості автоматизації обчислювань на основі програмного управління, яке здійснюється за програмою, що в цифровій формі зберігається у пам'яті комп'ютера разом із вхідними даними та результатами. *Програма* – це послідовність команд, яка може виконуватися комп'ютером для одержання певних результатів [2–4]. За рахунок відповідного вибору команд та їхньої послідовності у програмі можна розв'язувати широке коло задач щодо перетворення інформації, що надає універсальності підходу, який засновано на програмному управлінні, та дозволяє використовувати комп'ютери для розв'язування різноманітних задач. Команди, що містяться у програмі, природно мають бути виключно із множини команд, які відомі процесору комп'ютера. Сучасні комп'ютери містять сотні команд, які є примітивними, що суттєво ускладнює написання програм безпосередньо на машинному коді. Для збільшення швидкості написання програм було запропоновано використовувати *мови програмування* – сукупність правил символьного запису програм, які дозволяють представляти програми у зрозумілому наочному текстовому вигляді, і за допомогою спеціальних програм – *компіляторів* (трансляторів) переводити програми, написані на мовах програмування, у відповідні їм послідовності команд процесора, тобто у програми, що безпосередньо виконуються на комп'ютері [3]. Якщо послідовність дій, що реалізується в програмі, уявляти *алгоритмом*, то програма буде записом цього алгоритму. При первинному знайомстві із програмуванням таких уявлень про алгоритми і програми є цілком достатньо, але слід пам'ятати, що теорія алгоритмів та програм (додаток А) є математизованою галуззю знань, що будується на більш точних формальних абстрактних уявленнях [5].

Різноманіття завдань, які розв'язують комп'ютерні програми, породжує різноманіття мов програмування, кожна з яких є більш зручною та пристосованою до створення програм певного класу. Історично першою

задачею, для розв'язування якої створювалися програми, було виконання математичних обчислень за заданими алгоритмами, для чого було створено мову програмування FORTRAN, назва якої відповідна словосполученню FORmula TRANslation [6]. Мова програмування FORTRAN є найстарішою із розповсюджених сьогодні мов програмування [7, 8]; більш сучасні мови програмування, такі як C, PASCAL, BASIC, PROLOG, LISP, створювалися як більш універсальні або більш спеціалізовані, ніж FORTRAN з урахуванням досвіду його використання, тому такі мови недоцільно використовувати при первинному знайомстві із програмуванням, для чого найбільш зручним є FORTRAN 77.

1.1.2 Основи мови програмування FORTRAN. Запис формул. Для запису програм мовою програмування FORTRAN використовують виключно доступні на клавіатурі наступні символи [4, 6, 9]:

26 великих літер латиниці A B C D … X Y Z
10 цифр 0 1 2 3 4 5 6 7 8 9
13 спеціальних символів ␣ . , = + - * / ( ) : ' $

Перший спеціальний символ означає пробіл, а більшість інших – зрозуміло із математики [6]. Текст програми розміщується у рядках таким чином, що кожен рядок містить одну дію, а послідовність дій визначається розташуванням рядків, таким чином, що дії виконуються послідовно від верхніх рядків до нижніх до закінчення програми [6, 9]. Текст програми у кожному рядку слід починати після 6 пробілів, які зарезервовано для розміщення службової інформації програми, про що буде розказано через декілька занять.

При виконанні навчальних завдань будемо використовувати вільне для користування програмне забезпечення на основі компілятора фірми WATCOM, що є загальнодоступним на сайті [10]. Фірма WATCOM відома як розробник найкращих компіляторів FORTRAN [11], що використовувалися 20 років тому і використовуються сьогодні для виконання обчислень при математичному моделюванні складних процесів у різноманітних технічних системах, наприклад в ядерних енергетичних установках. Кожна програма являє собою проект, що містить мінімум два файли, які доцільно зберігати на диску в окремій папці. Доцільно, щоб ім'я папки для зберігання проекту збігалося з ім'ям проекту. Перший із означених файлів – це файл проекту із розширенням *.WPJ, тобто WatcomProJect, що містить інформацію про інші файли, з яких складається проект. Далі будемо використовувати проекти, що складаються із файлів-джерел (sources), що містять програми, написані мовою програмування FORTRAN, та відповідно до цього мають розширення .FOR, тобто FORtran. Так, простіший перший навчальний (Educational) проект (Project) може складатися із двох файлів: EP1.WPJ та EP1.FOR.

Простіша програма виконує розрахунки за заданими формулами. Нехай, наприклад, слід визначити мінімальну товщину стінки S_R (мм) циліндричного корпуса ядерного реактора [12], яка витримає внутрішній тиск p (МПа) при внутрішньому діаметрі D (мм) та допустимому напруженні $[\sigma]$ (МПа) матеріалу, що можна здійснити за формулою [13] (додаток Б):

$$S_R = \frac{pD}{2[\sigma] - p}. \qquad (1.1)$$

Формула (1.1) є фактично алгоритмом перетворення цифрової інформації у вигляді трьох чисел p, D та $[\sigma]$ в число S_R. Слід зазначити, що p, D, $[\sigma]$ та S_R є дійсними числами, а також те, що у формулі (1.1) використовується ціле число, що дорівнює 2. Числа p, D, $[\sigma]$ та S_R можуть приймати будь-які значення, тобто є *змінними*, а число 2 не змінюється, тобто є *константою*.

Для роботи з інформацією в мовах програмування передбачені особливі об'єкти – змінні. *Змінна* являє собою об'єкт, який характеризується іменем, типом та значенням [9]. Для прикладу формули (1.1) величини p, D, $[\sigma]$ та S_R є змінними, оскільки вони мають імена, тип – дійсні числа та значення, які можуть задаватися у програмі. Символьна клавіатура комп'ютера не дозволяє записувати формули у формі звичайних дробів, як в (1.1), але для запису формул можна використовувати символьні позначення арифметичних дій. Так, формулу (1.1) можна записати за допомогою символьної клавіатури комп'ютера і змінних наступним чином:

```
SR=p*D/(2*SIGMA-P)                    (1.2)
```

Тут змінна з ім'ям SR відповідає S_R, а `SIGMA` – $[\sigma]$, а про інші можна легко здогадатися. Аналогічно (1.2) можна записувати і більш складні формули.

1.1.3 Створення програми розрахунку за заданою формулою. Розглянемо варіанти програм розрахунку за формулою (1.1). Для цього створимо проект EP1.WPJ, який має містити файл-джерело EP1.FOR. Зміст файлу EP1.FOR для виконання розрахунку за формулою (1.1) може мати вигляд:

Лістинг 1.1 – Розрахунок за формулоюъ

```
      PROGRAM EP1
      REAL SR,P,D,SIGMA
```

```
      P=10.0
      D=500.0
      SIGMA=250.0
      SR=P*D/(2*SIGMA-P)
      END
```

Кожна програма у FORTRAN має розпочинатися з оператора (команди, інструкції) PROGRAM, після якої слід вказати ім'я програми, і завершуватися оператором END. Після оператора PROGRAM одразу ж вводяться до розгляду змінні SR,P,D,SIGMA, які мають тип REAL, відповідний дійсним числам. Далі змінним присвоюються необхідні значення і здійснюється обчислення за формулою (1.1), що представлена у вигляді (1.2). Недоліком такої програми є те, що користувач не побачить на моніторі результатів розрахунків. Позбавлена від цього недоліку програма може мати наступний вигляд:

Лістинг 1.2 – Розрахунок за формулою, друк результату та призупинення

```
      PROGRAM EP1
      REAL SR,P,D,SIGMA
      P=10.0
      D=500.0
      SIGMA=250.0
      SR=P*D/(2*SIGMA-P)
      PRINT *,SR
      PAUSE
      END
```

Програма, наведена на лістингу 1.2, відрізняється від програми, що представлена на лістингу 1.1, наявністю рядків з інструкціями

```
PRINT *,SR                    (1.3)
  PAUSE                       (1.4)
```

Інструкція (1.3) друкує на моніторі значення змінної SR у форматі, що прийнятий за замовчуванням завдяки наявності символу «*». Інструкція (1.4) призупиняє роботу програми, доки не буде натиснуто клавішу ENTER на клавіатурі комп'ютера, що дозволяє користувачу побачити результати розрахунків та продовжити роботу з програмою.

Недоліком програми, що представлена на лістингу 1.2, є те, що користувач не може змінювати значення вихідних даних при виконанні програми. Позбавлена цього недоліку програма може мати наступний вигляд:

Лістинг 1.3 – Введення даних, розрахунок та друк результатів

```
      PROGRAM EP1
      REAL SR,P,D,SIGMA
      READ *,P
      READ *,D
      READ *,SIGMA
      SR=P*D/(2*SIGMA-P)
      PRINT *,SR
      PAUSE
      END
```

В програмі на лістингу 1.3 замість присвоювання відповідних змінних використано інструкцію READ, яка призупиняє роботу програми для введення з клавіатури значення вказаної в ній змінної, а після завершення введення продовжує виконання програми. Завдяки цьому користувач має можливість вводити необхідні йому дані для виконання розрахунку, і програма набуває властивості універсальності.

Робота із програмою (лістинг 1.3) передбачає, що користувач знає, в якому порядку слід вводити вихідні дані, та розуміє, що останнє число, яке друкує комп'ютер на екрані монітора, є результатом розрахунків; користувач також має знати, що для завершення роботи програми він має натиснути клавішу ENTER на клавіатурі. Користування такою програмою потребує її ретельного вивчення. Роботу із програмою можна суттєво полегшити друком коментарів, що підказують користувачу, що саме він має робити. Коментарі являють собою текст, а точніше набір символів. Для визначення наборів символів у мовах програмування передбачаються змінні та константи текстового (або символьного, або рядкового) типу. Символьні константи визначаються переліченням усіх символів, включаючи пробіли в одинарних лапках, наприклад 'SR='. Завдяки використанню констант символьного типу можна суттєво покращити програму, що наведена на лістингу 1.3, наступним чином:

Лістинг 1.4 – Розрахунок та друк результатів із поясненнями

```
      PROGRAM EP1
      REAL SR,P,D,SIGMA
      PRINT *,'INPUT P(MPA):'
      READ *,P
      PRINT *,'INPUT D(MM):'
      READ *,D
      PRINT *,'INPUT SIGMA(MPA):'
      READ *,SIGMA
      SR=P*D/(2*SIGMA-P)
      PRINT *,'SR=',SR
      PAUSE 'PRESS ENTER TO EXIT PROGRAM'
      END
```

Завдяки текстовим константам 'INPUT P(MPA):', 'INPUT D(MM):','INPUT SIGMA(MPA):', а також 'PRESS ENTER TO EXIT PROGRAM', значно спрощується робота користувача із програмою – користувач лише відповідає на питання комп'ютера і не повинен вивчати програму.

1.2 Виведення та введення даних системними пристроями

В операційній системі комп'ютерів передбачаються пристрої, призначені для виведення та введення даних; сучасні комп'ютери обов'язково комплектуються монітором, клавіатурою та маніпулятором – універсальними системними пристроями для виведення та введення даних. В кожній мові програмування для виведення та введення даних з використанням системних пристроїв передбачені відповідні можливості, які реалізуються схожим чином в різних мовах програмування. Особливості передбачених в мові програмування FORTRAN можливостей щодо виведення та введення даних за допомогою системних пристроїв є типовими для багатьох інших мов програмування, у тому числі C, тому являють значний інтерес.

1.2.1 Інструкції введення та виведення. В мові програмування FORTRAN передбачені інструкції PRINT та READ для виведення та введення даних за допомогою системних пристроїв – монітора та клавіатури відповідно [9]. Виведення та введення даних здійснюється у символьному вигляді.

1.2.1.1 Інструкція введення даних з клавіатури має вигляд:

```
READ FMT,INPLIST                    (1.5)
```

де FMT – опис формату, а INPLIST – список даних для введення з клавіатури, представлений у вигляді розділених комами імен змінних, значення яких мають вводитися з клавіатури.

Інструкція (1.5) призупиняє роботу програми та очікує введення даних з клавіатури відповідно за списком введення. Введені дані одразу ж відображуються в поточному рядку екрана монітора. Після закінчення введення, яке визначається натисканням клавіші ENTER, змінним зі списку введення присвоюються введені значення та програма продовжує роботу. Після завершення введення здійснюється перехід на наступний рядок екрана монітора, так що результати введення наступної інструкції (1.5) будуть відображатися у нижчому рядку екрана монітора. Шляхом опису формату визначається правило перетворення введених з клавіатури символів у значення змінних, що містяться у списку введення.

1.2.1.2 Інструкція для виведення даних на екран монітора має наступний вигляд:

```
PRINT FMT,OUTLIST                    (1.6)
```

де FMT – опис формату, а OUTLIST – список даних для виведення на екран монітора, представлений у вигляді розділених комами констант та (або) імен змінних, значення яких мають бути виведені на екран монітора.

Інструкція (1.6) за замовчуванням здійснює виведення даних зі списку виведення в один рядок на екрані монітора і після закінчення виведення автоматично «переходить» на початок наступного рядка, так що наступний оператор (1.6) здійснює виведення вже у наступний рядок. Шляхом опису формату визначається форма символьного представлення на екрані монітора даних, що містяться у списку виведення.

1.2.1.3 В інструкціях (1.5) та (1.6) можна використовувати безформатне виведення та введення, в якому форма символьного представлення даних на екрані монітора при введенні та правило перетворення введених з клавіатури символів у значення змінних, автоматично визначаються відповідно до типу даних, що містяться у списку даних для виведення та введення. Безформатному виведенню та введенню даних відповідає найпростіший опис формату, в якому FMT=*. Приклади використання інструкцій (1.5) та (1.6) для безформатного введення та виведення у програмі можуть бути проілюстровані у простішій програмі, у якій здійснюється введення даних та виведення введених даних на екран монітора:

Лістинг 1.5 – Приклад безформатного введення та виведення

```
      PROGRAM TESTREADPRINT
      REAL A,B,C
      READ *,A,B,C
      PRINT *,'A=',A, 'B=',B,'C=',C
      PAUSE 'PRESS ENTER TO EXIT THE PROGRAM'
      END
```

Безформатне введення та виведення визначено в лістингу 1.5 за допомогою символу «*». Список введення містить імена змінних A, B, C, значення яких мають бути введені з клавіатури через коми або(та) пробіли. Список для виведення містить символьні константи 'A=', 'B=', 'C=' та змінні A, B, C, які будуть виведені на екран монітора в окремому рядку.

Використання безформатного виведення та введення є найпростішим, але при цьому слід пам’ятати, що форма виведення та правила перетворення введених даних можуть відрізнятися на різних комп’ютерах.

1.2.2 Визначення формату виведення та введення. Для забезпечення однакового виведення та введення даних програмою при її роботі на різних комп'ютерах передбачене точне визначення форматів виведення та введення.

1.2.2.1 Формат введення являє собою текстову константу (або змінну текстового типу, що буде розглянута пізніше у розд. 4), в якій спеціальним чином закодовані правила перетворення введених з клавіатури символів у значення змінних, які містяться у списку введення [9]. Кількість та послідовність визначення таких правил має відповідати кількості змінних та їхній послідовності у списку введення; кожне таке окреме правило записується спеціальним чином у вигляді *дескриптора* – числово-літерного коду, який визначає відповідне правило перетворення, а дескриптори, що відповідають різним змінним зі спуску введення, розділяються комами. Текстова константа, що визначає формат введення або (та) виведення, обов'язково має починатися з символу "(" та закінчуватися символом ")".

Розглянемо деякі найбільш широко використовувані дескриптори перетворення дійсних чисел, передбачені у мові програмування FORTRAN.

При введені та виведенні дійсних чисел, коли відома їхня величина, зручно використовувати наступний формат чисел:

$$\underbrace{\pm NN\ldots N.\underbrace{NN\ldots N}_{d}}_{w}, \tag{1.7}$$

де $\pm$ – знак числа; $NN\ldots N$ – деяка послідовність цифр; «.» – десятинна точка; d – кількість значущих цифр після десятинної точки; w – загальна кількість символів для представлення числа, включаючи знак, цифри та десятинну точку.

Наприклад, якщо величина тиску p, що використовується при проектуванні ядерних енергоустановок, із фізичних уявлень обмежується умовами $0 \le p \le 300$ МПа та враховується із точністю, не меншою ніж $0{,}01$ МПа, то ця величина може бути представлена у вигляді (1.7), де $d = 2$ та $w = 6$.

Для введення та виведення дійсних чисел у вигляді (1.7) у мові програмування FORTRAN передбачені дескриптори F [9], які мають наступний вигляд:

```
Fw.d
```  (1.8)

де w – ціле число, що вказує на загальну кількість символів для представлення числа, включаючи знак, цифри та десятинну точку; d – ціле число, що вказує на кількість цифр після десятинної точки.

Більш універсальним є формат представлення дійсних чисел у вигляді

$$\pm 0.\underbrace{NN\ldots N}_{d}\cdot 10^{\pm NNN} \equiv \underbrace{\pm 0.\underbrace{NN\ldots N}_{d}E\pm NNN}_{w}, \tag{1.9}$$

де ± – знак числа та десятинного ступеня; $NN\ldots N$ – деяка послідовність цифр; NNN – послідовність трьох цифр; «.» – десятинна точка; d – кількість цифр після десятинної точки; w – загальна кількість символів для представлення числа, включаючи знак, цифри, десятинну точку, символ E – ознаку десятинного ступеня та три числа, що визначають десятинний ступінь.

Для введення та виведення дійсних чисел у вигляді (1.9) у мові програмування FORTRAN передбачені дескриптори E [9], які мають наступний вигляд:

```
Ew.d
```
(1.10)

де w – ціле число, що вказує на загальну кількість символів для представлення числа, включаючи знаки числа та десятинного ступеня, десятинну точку, а також символ-ознаку десятинного ступеня та три числа для визначення десятинного ступеня; d – ціле число, що вказує на кількість цифр після десятинної точки.

Для форматного введення та виведення також корисним є дескриптор X, який дозволяє пропускати задану кількість символів при введенні або задавати задану кількість пробілів при виведенні даних та має вигляд [9]:

```
wX
```
(1.11)

де w – ціле число, що вказує, яку кількість символів слід пропустити при введенні та скільки пробілів слід надрукувати у рядку при виведенні.

1.2.2.2 Приклади використання дескрипторів (1.8) та (1.10), (1.11) в інструкціях (1.5) та (1.6) для форматного введення та виведення у програмі можуть бути проілюстровані у простішій програмі, у якій здійснюється введення даних та виведення введених даних на екран монітора:

Лістинг 1.6 – Приклад форматного введення та виведення

```
      PROGRAM TESTFORMATREADPRINT
      REAL P1,P2
      READ '(F6.2,1X,E15.8)',P1,P2
      PRINT '(F6.2,1X,E15.8)',P1,P2
      PAUSE 'PRESS ENTER TO EXIT THE PROGRAM'
      END
```

Нехай при роботі програми, що представлена на лістингу 1.6, необхідно ввести значення $P1 = 23$ та $P2 = 8,74$. Тоді користувач має ввести:

```
023.00␣␣0.874E+1                    (1.12)
```

Головним в (1.12) є обов'язкова наявність двох пробілів між введеними числами, перший з яких виконує функцію відокремлення чисел та передбачений дескриптором X у форматі введення, а другий – займає місце пропущеного додатного знаку числа. Слід зазначити, що користуватися програмами із форматним вводом при використанні дескриптора E слід дуже обережно, оскільки помилки при введенні можуть привести до некоректної роботи програми. Слід рекомендувати для введення використовувати дескриптори F, а дескриптори E слід використовувати переважно для виведення даних.

1.2.3 Діалогові системи. Недоліком програм, що були представлені як приклади введення та виведення даних, є те, що при введенні та при виведені користувач програми повинен заздалегідь знати, що він має вводити та що виведено на екран монітора. В цих умовах користувач перед використанням програми повинен фактично повністю оволодіти алгоритмом програми, що потребує достатньо великих зусиль у випадку складних програм, в яких необхідне введення десятків значень вихідних даних та виведення великої кількості результатів розрахунків. Для допомоги користувачу програм доцільно розробляти діалогові системи, які за допомогою системних пристроїв виведення (екрана монітора) повідомляють користувача про те, що саме він має вводити безпосередньо перед необхідністю введення, а виведення результатів супроводжують необхідними поясненнями. Створення діалогових систем становить достатньо складну задачу [14, 15], тому далі розглянемо деякі простіші прийоми, які сприяють введенню вихідних даних та виведенню результатів.

1.2.3.1 Створення діалогових систем передбачає забезпечення зручного для сприйняття вигляду інформації, що виводиться на екран монітора для допомоги користувачу при введенні даних та при виведені результатів. Розглянемо деякі дескриптори, які допомагають представляти інформацію на екрані монітора у зручному для користувача вигляді.

Введення вихідних даних доцільно здійснювати після розміщення на екрані монітора необхідних коментарів, для чого може бути корисним дескриптор A [9], який має наступний вигляд:

```
Aw                                  (1.13)
```

де w – розмір поля даних в запису.

Допускається не вказувати розмір поля в (1.13), тоді розмір поля визначається автоматично відповідно до розміру поля введеного об’єкта або даних для виводу.

Корисним також є недрукований дескриптор \, який має відокремлюватися від інших дескрипторів за допомогою коми та ставитися наприкінці опису формату перед символом ")", якщо немає потреби до переходу на новий рядок. Така потреба буває необхідна для забезпечення зручності при введені вихідних даних, оскільки дозволяє розташовувати коментар до даних, що необхідно ввести, та введення цих даних в одному рядку. Приклад використання означеного недрукованого дескриптора \ та дескриптора (1.13) наведено нижче:

Лістинг 1.7 – Приклад побудови зручного введення та виведення

```
      PROGRAM SR
      REAL SR,P,D,SIGMA
      PRINT'(A,\)','INPUT PRESSURE, P(MPA):'
      READ'(F6.2)',P
      PRINT'(A,\)','INPUT DIAMETER, D(MM):'
      READ'(F6.2)',D
      PRINT'(A,\)','INPUT AVAILABLE STRESS, SIGMA(MPA):'
      READ '(F6.2)',SIGMA
      SR=P*D/(2*SIGMA-P)
      PRINT'(A,F6.2,1X,A)','P=',P,'(MPA)'
      PRINT'(A,F6.2,1X,A)','D=',D,'(MM)'
      PRINT'(A,F6.2,1X,A)','SIGMA=',SIGMA,'(MPA)'
      PRINT'(A,E15.8,1X,A)','SR=',SR,'(MM)'
      PAUSE 'PRESS ENTER TO EXIT PROGRAM'
      END
```

При роботі програми, текст якої представлено у лістингу 1.7, введення даних та виведення результатів є дуже зручним, оскільки користувач бачить, яку саме величину він має вводити та в якій розмірності, а при виведенні – має можливість проконтролювати введені дані та результат розрахунку із зазначенням розмірності.

Слід зазначити, що зручний дескриптор \ не передбачений в стандарті мови програмування FORTRAN 77, а є розширенням відповідного діалекту цієї мови програмування, який можна використовувати, якщо компіляція програми здійснюватиметься компілятором WATCOM [16].

1.2.3.2 У програмах іноді доводиться використовувати велику кількість однакових описів форматів для введення та виведення, тому іноді буває зручним окремо описати формат введення або виведення та посилатися на цей опис у різних місцях програми. Для цього передбачена не виконувана інструкція опису формату, яка має вигляд:

```
Label FORMAT FMT                                    (1.14)
```

де `Label` – унікальна мітка, яка дозволяє послатися на формат, FMT – опис формату.

Мітка являє собою цифри, які мають бути розташованими у рядку відповідної інструкції з 1-го по п’ятий символи; старші нулі та пробіли ігноруються. Формат опису має бути представлений змістом символьного опису відповідного формату. Приклад використання окремого опису формату у програмі може мати вигляд:

Лістинг 1.8 – Приклад використання окремого опису формату

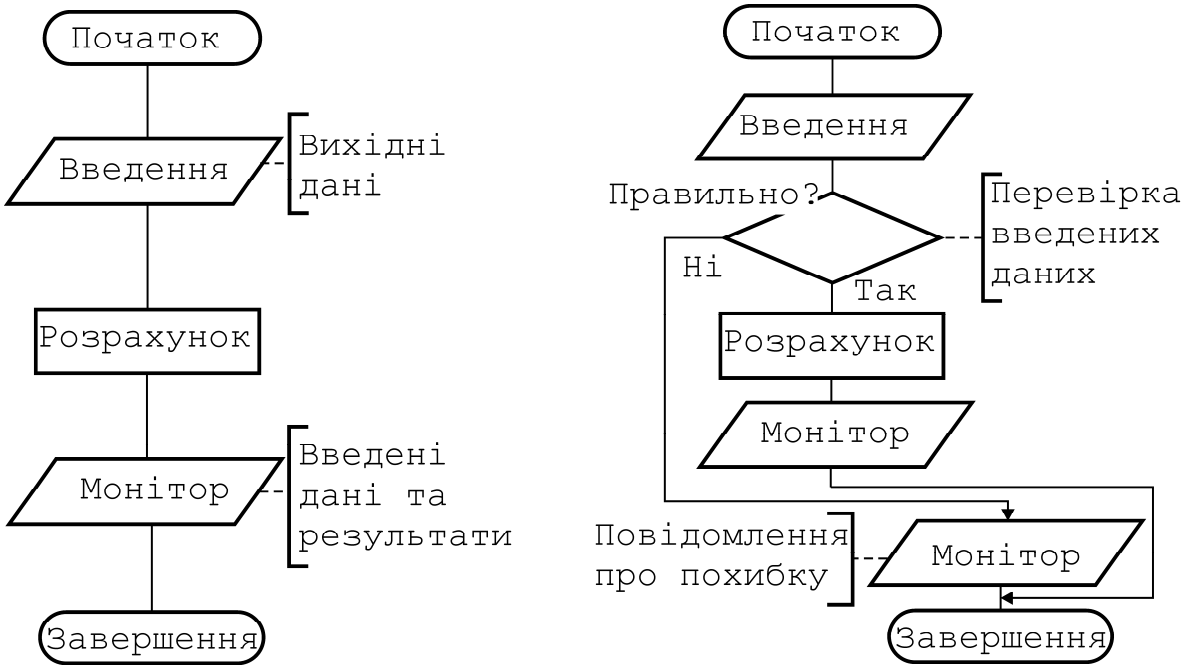
```
      PROGRAM SR
      REAL SR,P,D,SIGMA
      PRINT 100,'INPUT PRESSURE, P(MPA):'
      READ'(F6.2)',P
      PRINT 100,'INPUT DIAMETER, D(MM):'
      READ'(F6.2)',D
      PRINT 100,'INPUT AVAILABLE STRESS, SIGMA(MPA):'
      READ '(F6.2)',SIGMA
      SR=P*D/(2*SIGMA-P)
      PRINT 110,'P=',P,'(MPA)'
      PRINT 110,'D=',D,'(MM)'
      PRINT 110,'SIGMA=',SIGMA,'(MPA)'
      PRINT'(A,E15.8,1X,A)','SR=',SR,'(MM)'
100   FORMAT (A,\)
110   FORMAT (A,F6.2,1X,A)
      PAUSE 'PRESS ENTER TO EXIT PROGRAM'
      END
```

1.3 Управління порядком виконання інструкцій програми

В мовах програмування передбачається можливість зміни прийнятої за замовчанням послідовності виконання інструкцій за їхнім розташування в тексті програми, що буває необхідним в деяких випадках. Робота та, навіть, форма запису інструкцій керування є подібними в різних мовах програмування, тому відноситься до загальних понять програмування.

1.3.1 Умовні та безумовні переходи в програмах. В деяких випадках виникає потреба змінити прийняту за замовчанням послідовність виконання інструкцій за їхнім розташуванням в тексті програми. На перший погляд, такої потреби немає, оскільки будь-яка програма розрахунків має забезпечувати введення даних для розрахунку, здійснення розрахунку та виве-

дення результатів у заданій послідовності, що не має змінюватися, як на графічній схемі (flowchart) алгоритму (програми), що показана на рис. 1.1а та виконана (додаток В) за загальноприйнятими правилами [17–19]. В той же час, уявімо собі, що при роботі з програмою, схема якої представлена на рис. 1.1а, користувач помилково ввів з клавіатури неправильні вихідні дані. В цьому випадку програма продовжить виконання, тобто виконає розрахунки та виведе їхні результати відповідно до схеми (рис. 1.1а), що є значним недоліком програми, особливо у випадку, коли виконання розрахунків може здійснюватись декілька годин. Для виправлення цього недоліку введені вихідні дані слід перевірити і приступити до розрахунку лише за умов правильно введених даних, а у випадку неправильних вихідних даних слід повідомити користувача та зупинити роботу програми, як це показано на схемі, що представлена на рис. 1.1б. Таким чином, маємо потребу в умовному та безумовному зміненні послідовності виконання інструкцій програми. Умовне змінення необхідно для завершення програми при неправильно введених вихідних даних, а безумовне – для нормального завершення роботи програми (рис. 1.1б).



а) *б)*

Рисунок 1.1 – Схеми алгоритму програми без (а) та з переходами (б)

У наведеному прикладі (рис. 1.1б) необхідність зміни послідовності інструкцій обумовлена перевіркою правильності введених вихідних даних, але можна навести приклади, коли така необхідність обумовлена іншими причинами, наприклад, різними розрахунками у залежності від введених вихідних даних або від поточних проміжних результатів розрахунків. Наводити подібні приклади зараз не є потрібним, оскільки навіть наведеного

прикладу (рис. 1.1б) достатньо для ілюстрації необхідності використання умовних та безумовних переходів – змін послідовності виконання інструкцій програми.

Умовний перехід – це зміна послідовності виконання інструкцій програми відповідно до виконання деякої умови, яка визначається призначенням програми.

Безумовний перехід – це задана зміна послідовності виконання інструкцій програми незалежно від стану програми, що визначається сукупністю значень змінних.

Зміна послідовності виконання інструкцій програми здійснюється за допомогою спеціальних інструкцій переходу, що визначають, яку саме інструкцію слід виконувати із поточного місця програми. Для визначення інструкції програми, яку слід виконувати, в інструкціях переходу використовуються мітки, якими мають бути заздалегідь позначені необхідні інструкції програми. Таким чином, зміна послідовності виконання інструкцій програми може здійснюватися шляхом переходу до виконання лише інструкцій, що позначені мітками.

1.3.2 Інструкція безумовного переходу. Реалізація безумовного переходу є найпростішим способом змінення послідовності виконання інструкцій програми та здійснюється за допомогою інструкції [4, 6, 9]:

```
GOTO label
```
(1.15)

де `label` – унікальна мітка, якою заздалегідь помічена відповідна інструкція програми.

Інструкція (1.15) працює наступним чином, що після її виконання програма виконує не ту інструкцію, що міститься безпосередньо за інструкцією (1.15), а інструкцію, що позначена міткою `label`. Після цього програма послідовно виконує інструкції, які розташовані слідом за інструкцією, що позначена міткою `label`.

Слід зазначити, що стиль програмування, який використовує мітки та інструкції переходу, сьогодні вважається застарілим і не рекомендується для широкого використання. В той же час, навіть у найсучасніших мовах програмування, передбачається можливість використання міток та безумовних переходів. Це обґрунтовується тим, що в деяких випадках використання міток та переходів є найпростішим засобом реалізації завдань програми. Тобто використання міток та переходів є цілком допустимим у сучасному програмуванні за умов, що кількість міток та переходів у програмі є невеликою.

1.3.3 Простіші інструкції умовних переходів. Логічний тип. Умовні переходи дозволяють здійснювати змінення послідовності виконання

інструкцій програми у залежності від умов, яким задовольняють значення змінних та виразів, що містяться в програмі. Для здійснення умовних переходів у мові програмування FORTRAN та в багатьох інших мовах програмування передбачена інструкція, причому передбачено декілька видів такої інструкції. Далі розглянемо найпростіші два типи інструкції `IF`, а саме арифметичний та логічний типи [9]; більш складний блочний тип розглянемо на наступному занятті.

1.3.3.1 Арифметична інструкція IF забезпечує реалізацію умовного галуження для зміни послідовності виконання частин програми відповідно до виконання чи невиконання певних умов. Арифметична інструкція `IF` має вигляд:

```
IF (ArExpr) label1,label2,label3          (1.16)
```

де `ArExpr` – арифметичний вираз; `label1`, `label2` та `label3` – мітки, які позначають програмні інструкції.

Інструкція (1.16) працює наступним чином: спочатку визначається результат розрахунків за арифметичним виразом `ArExpr` і далі здійснюється перехід до інструкції, що позначена міткою `label1`, `label2` або `label3`, якщо результат `ArExpr` арифметичного розрахунку інструкції `IF` відповідно є від'ємним, дорівнює нулю, є додатним.

Розглянемо приклад використання інструкції (1.16) для перевірки правильного та коректного введення вихідних даних у програмі розрахунку мінімальної товщини стінки S_R (мм) циліндричного корпусу ядерного реактора (додаток Б), що має витримати внутрішній тиск p (МПа) при внутрішньому діаметрі D (мм) та допустимому напруженні $[\sigma]$ (МПа) матеріалу, що можна здійснити за розглянутою вище відомою [13] формулою (1.1), яка була розглянута раніше в якості прикладу. Відповідно до фізичного змісту усі вихідні дані розрахунку за формулою (1.1): внутрішній тиск p, внутрішній діаметр D та допустиме напруження $[\sigma]$ матеріалу, а також результат розрахунків – товщини стінки S_R мають бути додатними числами. В той же час, при введенні вихідних даних користувач програми може помилково ввести від'ємне значення напруження, та (або) внутрішнього діаметра, та (або) допустимого напруження матеріалу, і в кожному із цих випадків результат розрахунку за формулою (1.1) не матиме змістовного сенсу. При помилковому введенні вихідних даних результат розрахунків за формулою (1.1) може виявитися від'ємним, що покаже користувачу про помилки введення вихідних даних, але навіть при неправильно введених вихідних даних результат розрахунків може виявитися додатною величиною, і користувач не помітить похибок введення вихідних даних. З ураху-

ванням цих обставин, щоб уникнути помилок у розрахунках, є сенс перевіряти правильність та коректність введення вихідних даних. Наприклад, при програмуванні розрахунку за формулою (1.1) слід, принаймні, перевірити, чи є введені значення внутрішнього тиску p, внутрішнього діаметра D та допустимого напруження $[\sigma]$ матеріалу додатними, і якщо ні, то повідомити про це користувача програми та не здійснювати розрахунку некоректних даних.

Приклад програми, в якій виконується розрахунок за формулою (1.1) із попередньою перевіркою введених вихідних даних та повідомленням користувача програми у випадку неправильного їхнього введення, наведений на лістингу 1.9. У наведеному прикладі арифметичні інструкції вигляду (1.16) використані для перевірки правильного введення вихідних даних, які мають бути додатними, а безумовний перехід вигляду (1.15) використано, щоб програма не друкувала на екрані монітора повідомлення про похибку введених вихідних даних у випадку, коли такої помилки немає. Зазначимо, що дві із трьох міток в арифметичних інструкціях `IF` збігаються у наведеному прикладі програми.

Лістинг 1.9 – Приклад керування послідовністю виконання інструкцій

```
      PROGRAM SR
      REAL SR,P,D,SIGMA
      PRINT 100,'INPUT PRESSURE, P(MPA):'
      READ'(F6.2)',P
      IF (P) 210,210,220
220   PRINT 100,'INPUT DIAMETER, D(MM):'
      READ'(F6.2)',D
      IF (D) 210,210,230
230   PRINT 100,'INPUT AVAILABLE STRESS, SIGMA(MPA):'
      READ '(F6.2)',SIGMA
      IF (SIGMA) 210,210,240
240   SR=P*D/(2*SIGMA-P)
      PRINT 110,'P=',P,'(MPA)'
      PRINT 110,'D=',D,'(MM)'
      PRINT 110,'SIGMA=',SIGMA,'(MPA)'
      PRINT'(A,E15.8,1X,A)','SR=',SR,'(MM)'
      GOTO 300
210   PRINT*,'ERROR INPUTED DATA'
100   FORMAT (A$)
110   FORMAT (A,F6.2,1X,A)
300   PAUSE 'PRESS ENTER TO EXIT PROGRAM'
      END
```

Слід зазначити, що не усі можливі додатні вихідні дані щодо внутрішнього тиску p, внутрішнього діаметра D та допустимого напруження $[\sigma]$ матеріалу є коректними для виконання розрахунку за формулою (1.1). Дійсно, за умов, коли внутрішній тиск p є занадто великим у порівнянні із допустимим напруженням $[\sigma]$ матеріалу, результат розрахунку за формулою (1.1) буде некоректним, оскільки приведе до від'ємної товщини S_R стінки. Таким чином, введені вихідні дані щодо внутрішнього тиску та допустимого напруження мають задовольняти наступній умові:

$$2[\sigma] > p. \qquad (1.17)$$

Пропонується самостійно змінити програму, що наведена у лістингу 1.9, таким чином, щоб у випадку невиконання умови (1.17) розрахунок не здійснювався і користувач отримував повідомлення про некоректність введених вихідних даних.

1.3.3.2 Логічна інструкція `IF` забезпечує реалізацію умовного виконання інструкції програми, тобто виконання чи невиконання певної інструкції програми у залежності від виконання чи невиконання певних умов. Логічна інструкція `IF` має вигляд [6, 9]:

```
IF (LogicExpr) Instruct                    1.18)
```

де `LogicExpr` – логічний вираз, яким формується умова; `Instruct` – інструкція, яка має бути виконана при задовільнені мови, що формується виразом `LogicExpr`.

Логічний вираз – це вираз, результат якого має логічний тип. Логічний тип – це тип даних, які можуть приймати лише два значення: `.TRUE.` (так, істина), або `.FALSE.` (ні, неправда). Інструкція (1.18) працює наступним чином: спочатку визначається результат розрахунків за логічним виразом `LogicExpr` і далі, якщо результат `LogicExpr` дорівнює `.TRUE.`, здійснюється виконання інструкції Instruct; після цього програма продовжує виконання наступної інструкції. Зазначимо, що в якості `Instruct` може бути будь-яка інструкція, що виконується, у тому числі й інструкція безумовного переходу (1.15). Використання специфічного, незрозумілого для повсякденної свідомості, типу даних – логічного типу певним чином ускладнює початкове ознайомлення з інструкцією (1.18). Щоб уникнути початкового ускладнення при ознайомленні з інструкцією (1.18), покажемо на прикладі, як логічна інструкція (1.18) може цілком замінити арифметичну інструкцію (1.16). Для цього змінимо відповідним чином приклад програми, що наведений у лістингу 1.9, наприклад, наступним чином:

Лістинг 1.10 – Використання логічної інструкції IF

```
      PROGRAM SR
      REAL SR,P,D,SIGMA
      PRINT 100,'INPUT PRESSURE, P(MPA):'
      READ'(F6.2)',P
      IF (P.LE.0) GOTO 210
      PRINT 100,'INPUT DIAMETER, D(MM):'
      READ'(F6.2)',D
      IF (D.LE.0) GOTO 210
      PRINT 100,'INPUT AVAILABLE STRESS, SIGMA(MPA):'
      READ '(F6.2)',SIGMA
      IF (SIGMA.LE.0) GOTO 210
      SR=P*D/(2*SIGMA-P)
      PRINT 110,'P=',P,'(MPA)'
      PRINT 110,'D=',D,'(MM)'
      PRINT 110,'SIGMA=',SIGMA,'(MPA)'
      PRINT'(A,E15.8,1X,A)','SR=',SR,'(MM)'
      GOTO 300
210   PRINT*,'ERROR INPUTED DATA'
100   FORMAT (A$)
110   FORMAT (A,F6.2,1X,A)
300   PAUSE 'PRESS ENTER TO EXIT PROGRAM'
      END
```

Для перевірки умови $p \le 0$ у відповідній інструкції програми, що наведена у лістингу 1.10, в якості `LogicExpr` використовується логічний вираз:

```
P.LE.0
```
(1.19)

Символи `.LE.` в мові програмування FORTRAN символічно представляють операцію відношення «менше або дорівнює» (англ. Less or Equal). Неважко здогадатися, що вираз (1.19) відповідає перевірці умови $p \le 0$; у випадку, якщо ця умова виконується, результатом логічного виразу (1.19) буде значення `.TRUE.` і програма здійснить перехід за відповідною міткою, а у випадку невиконання цієї умови результатом логічного виразу (1.19) буде значення `.FALSE.` і програма не буде здійснювати перехід і виконуватиме наступну інструкцію. Якщо порівняти програми, що наведені у лістингу 1.8 та лістингу 1.9, то можемо зробити висновок, що використання логічної інструкції IF потребує меншої кількості міток, ніж використання арифметичної інструкції IF, що є більш зручним. Безумовно, що в мові програмування FORTRAN передбачена можливість різних операцій відношення, а не тільки операції типу (1.19), що буде розглянуто на наступному занятті.

Необхідний для використання в логічній інструкції `IF` логічний тип даних потребує спеціального вивчення, що буде здійснено протягом наступного заняття. Зараз слід уяснити наступне, що логічний тип – це спеціальним чином введений тип даних, які можуть приймати певні значення. Для роботи з логічним типом даних вводяться операції відношення, вихідні дані яких можуть мати будь-який тип, а вихідний результат має логічний тип, а також логічні операції, вихідні дані і результати яких мають виключно логічний тип. На наступному занятті розглянемо введення логічного типу даних, а також операції, які можна здійснювати з використанням логічного типу. Хоча логічний тип даних є досить специфічним та може бути незрозумілим для повсякденної свідомості, його введення є дуже корисним для розв'язування багатьох задач не тільки при розробці програм, але й при розв'язуванні інженерних задач. За умов розуміння практичної зручності від використання логічного типу даних, його вивчення не має містити зручностей.

1.4 Розширені можливості управління порядком виконання інструкцій

Існуючі мови програмування надають широких можливостей щодо керування порядком виконання інструкцій в програмах, але реалізація таких можливостей в різних мовах програмування здійснюється схожим чином. Керування порядком виконання інструкцій широко використовується в програмах різноманітного призначення, тому відповідні питання відносяться до загальних понять програмування.

1.4.1 Логічний тип даних. Розширені можливості керування порядком виконання інструкцій програм засновані на використанні особливого типу даних – логічного типу [6, 9].

Логічний тип даних `(LOGICAL)` – це тип даних, які можуть приймати лише два значення:

`.TRUE.` (1.20)

`.FALSE.` (1.21)

Логічний тип можна уявляти як два можливі варіанти відповіді на питання щодо правдивості деякого твердження, яке може бути або істинним (правдивим), або ні. Змістовний сенс значення (1.20) відповідає істинності деякого твердження, тобто є еквівалентним відповіді «так»; змістовний сенс значення (1.21) відповідає не істинному характеру деякого твердження, тобто є еквівалентним відповіді «ні».

Логічний тип використовується для перевірок, чи задовольняють значення змінних програми певним умовам. Наприклад, можна перевірити, чи є меншим значення деякої змінної порівняно із значенням іншої певної змінної. Для цього в мовах програмування передбачаються операції відношення між значеннями змінних та (або) констант. Такі операції відношення є символічною формою (кодуванням) представлення умов, яким мають задовольняти два значення:

```
VAR1 RELOP VAR2
```
(1.22)

де `VAR1` та `VAR2` – імена змінних, або константи, значення яких перевіряються на виконання деякого відношення; `RELOP` – символьний запис операції відношення.

Результатом операції відношення є логічний тип, який набуває значення (1.20), якщо відношення (1.22) виконується, та значення (1.21), якщо відношення (1.22) не виконується; зрозуміло, що порівнюватися мають значення одного типу даних. В якості `VAR1` та `VAR2` замість імен змінних та констант може виступати арифметичний вираз, що містить імена змінних та константи. В мові програмування FORTRAN передбачено шість операцій порівняння, які наведені в табл. 1.1.

Таблиця 1.1 – Операції відношення в мові програмування FORTRAN

Символьний запис операції	Змістовний сенс операції
`.LT.`	менше
`.LE.`	менше або дорівнює
`.EQ.`	дорівнює
`.NE.`	не дорівнює
`.GE.`	більше або дорівнює
`.GT.`	більше

Прикладом використання в простій програмі змінних логічного типу та деяких із операцій відношення, що наведені в табл. 1.1, може виступати повідомлення користувача про характер введеного числового значення:

Лістинг 1.11 – Використання логічної змінної

```
      PROGRAM PR4A
      REAL A
      LOGICAL COND
100   FORMAT (A$)
      PRINT 100,'INPUT VALUE, A:'
      READ'(F6.2)',A
      IF (A.LT.0) PRINT*,'A<0'
```

```
      COND=A.EQ.0
      IF (COND) PRINT*,'A=0'
      IF (A.GT.0) PRINT*,'A>0'
      PAUSE 'PRESS ENTER TO EXIT PROGRAM'
      END
```

Наведена в лістингу 1.11 програма повідомляє користувача про знак введеного ним числового значення дійсної змінної.

Більш складні, ніж наведені у табл. 1.1, умови можна формулювати із використанням логічних операцій – операцій над величинами логічного типу, результат яких також є логічним типом. Відомо п'ять основних логічних операцій, які наведені в табл. 1.2.

Таблиця 1.2 – Логічні операції

Символьний запис операції	Змістовний сенс операції
.NOT.	Логічне заперечення (ні)
.AND.	Кон'юнкція (і)
.OR.	Диз'юнкція (або)
.EQV.	Логічна еквівалентність
.NEQV.	Логічна нееквівалентність

Результати логічних операцій (табл. 1.2) наведені в табл. 1.3.

Таблиця 1.3 – Результати логічних операцій

Значення змінних		Результати логічних операцій	
V1	V2	V1.AND.V2	V1.OR.V2
.TRUE.	.TRUE.	.TRUE.	.TRUE.
.TRUE.	.FALSE.	.FALSE.	.TRUE.
.FALSE.	.TRUE.	.FALSE.	.TRUE.
.FALSE.	.FALSE.	.FALSE.	.FALSE.

Продовження таблиці 1.3

Значення змінних		Результати логічних операцій		
V1	V2	.NOT.V2	V1.EQV.V2	V1.NEQV.V2
.TRUE.	.TRUE.	.FALSE.	.TRUE.	.FALSE.
.TRUE.	.FALSE.	.TRUE.	.FALSE.	.TRUE.
.FALSE.	.TRUE.	.FALSE.	.FALSE.	.TRUE.
.FALSE.	.FALSE.	.TRUE.	.TRUE.	.FALSE.

Логічні операції (табл. 1.2 і табл. 1.3) дозволяють формулювати будь-які умови в одному виразі, і для цього виразі можна використовувати декілька логічних операцій, наприклад:

```
(V1.AND.V2).OR.(.NOT.V3)
```
(1.23)

де V1, V2 та V3 – змінні логічного типу; дужками визначається послідовність операцій.

Оскільки кожна зі змінних, що входить до виразу (1.23), може приймати лише два значення (1.20) або (1.21), то всього маємо 2^3 =8 можливих варіантів вихідних даних для виконання операцій у виразі (1.23). Результатом виразу (1.23) є значення логічного типу, яке буде визначатися відповідно до усіх можливих 8-ми варіантів значень змінних, як показано в табл. 1.4.

Таблиця 1.4 – Результати логічного виразу (1.23)

Значення змінних			Результати виразу
V1	V2	V3	
.TRUE.	.TRUE.	.TRUE.	.TRUE.
.TRUE.	.FALSE.	.TRUE.	.FALSE.
.FALSE.	.TRUE.	.TRUE.	.FALSE.
.FALSE.	.FALSE.	.TRUE.	.FALSE.
.TRUE.	.TRUE.	.FALSE.	.TRUE.
.TRUE.	.FALSE.	.FALSE.	.TRUE.
.FALSE.	.TRUE.	.FALSE.	.TRUE.
.FALSE.	.FALSE.	.FALSE.	.TRUE.

В якості логічних змінних у виразі (1.23) та подібних йому можуть виступати операції відношення (див. табл. 1.1), як, наприклад, у наступному фрагменті програми:

```
...
      REAL A,B,C
      LOGICAL COND
...
      COND=((A.LE.0).AND.(B.GT.C)).OR.(.NOT.(C.EQ.A))
...
      END
```

Завдяки дужкам можна однозначно розуміти послідовність виконання операцій відношення та логічних операцій, що наведена у прикладі при визначенні логічної змінної COND. Для заданих значень дійсних змінних A, B та C можна однозначно визначати результат – значення логічної змінної COND. Прикладом використання логічних операцій є програма:

Лістинг 1.12 – Використання логічних операцій

```
      PROGRAM PR4B
      REAL A,AMIN,AMAX
```

```
      LOGICAL COND
      PRINT '(A$)','INPUT VALUE, AMIN:'
      READ'(F6.2)',AMIN
      PRINT '(A$)','INPUT VALUE, AMAX:'
      IF (AMIN.GE.AMAX) GOTO 200
      READ'(F6.2)',AMAX
      PRINT '(A$)','INPUT VALUE, A:'
      READ'(F6.2)',A
      IF (A.LT.AMIN) PRINT*,'A<AMIN'
      IF (A.GT.AMIN) PRINT*,'A>AMIN'
      COND=(A.GE.AMIN).AND.(A.LE.AMAX)
      IF (COND) PRINT*,'AMIN<=A<=AMAX'
      GOTO 300
200   PRINT*,'ERROR AMIN>=AMAX'
300   PAUSE 'PRESS ENTER TO EXIT PROGRAM'
      END
```

Наведена в лістингу 1.12 програма пропонує користувачу ввести значення $A_{\min}$ та $A_{\max}$, повідомляє про помилку, якщо $A_{\min} \geq A_{\max}$, а в іншому випадку пропонує ввести значення A і повідомляє користувача про те, чи виконуються умови $A < A_{\min}$, $A > A_{\max}$ та $A_{\min} \leq A \leq A_{\max}$.

1.4.2 Структурна форма інструкції керування. Найбільш широкими можливостями щодо умовного керування послідовністю виконання інструкцій програми є структурна інструкція `IF`, яка дозволяє керувати виконанням послідовностей інструкцій у залежності від задоволення певної заданої умови. Така інструкція в тому чи іншому вигляді присутня в різних мовах програмування, а в FORTRAN має наступний вигляд:

```
IF (cond) THEN
instr for if
ELSE
instr for else
END IF
```

Тут `cond` – позначено логічну змінну, або вираз, якими визначають необхідну умову; `instr for if` позначено інструкції, які мають бути виконані за умов, коли значення cond дорівнює `.TRUE.`, а `instr for else` позначено інструкції, які мають виконуватися у протилежному випадку. Передбачено, що інструкція `ELSE` може бути відсутньою:

```
IF (cond) THEN
instr for if
END IF
```

Структурна форма інструкції керування дозволяє забезпечити виконування не одного, як в логічній інструкції IF, а декількох інструкцій при виконанні заданої певної умови. Слід підкреслити, що використання конструкцій вигляду IF–THEN–ELSE є характерним та здійснюється однаково в багатьох універсальних мовах програмування, таких як PASCAL, C, Python та багатьох інших. Такі конструкції із використанням логічного типу даних дозволяють керувати виконанням достатньо великої кількості послідовностей інструкцій у залежності від довільної кількості різноманітних досить складних умов, які можуть знадобитися при розробці програм різного призначення.

Використання структурної форми інструкції керування дозволяє зменшити кількість міток та переходів у програмі та завдяки цьому значно спрощує розуміння тексту програми, що можна побачити в наступному прикладі розрахункової програми:

Лістинг 1.13 – Використання блочної інструкції керування

```
      PROGRAM SR4
      REAL SR,P,D,SIGMA
100   FORMAT (A$)
      PRINT 100,'INPUT PRESSURE, P(MPA):'
      READ'(F6.2)',P
      IF (P.LE.0) THEN
      PRINT*,'ERROR INPUTED DATA, P<=0'
      GOTO 200
      ENDIF
      PRINT 100,'INPUT DIAMETER, D(MM):'
      READ'(F6.2)',D
      IF (D.LE.0) THEN
      PRINT*,'ERROR INPUTED DATA, D<=0'
      GOTO 200
      ENDIF
      PRINT 100,'INPUT AVAILABLE STRESS, SIGMA(MPA):'
      READ '(F6.2)',SIGMA
      IF (SIGMA.LE.0) THEN
      PRINT*,'ERROR INPUTED DATA, SIGMA<=0'
      GOTO 200
      ENDIF
      SR=P*D/(2*SIGMA-P)
110   FORMAT (A,F6.2,1X,A)
      PRINT 110,'P=',P,'(MPA)'
      PRINT 110,'D=',D,'(MM)'
      PRINT 110,'SIGMA=',SIGMA,'(MPA)'
```

```
      PRINT 110,'SR=',SR,'(MM)'
200   PAUSE 'PRESS ENTER TO EXIT PROGRAM'
      END
```

У програмі, що наведено в лістингу 1.13, здійснюється розрахунок товщини стінки циліндричної посудини високого тиску після перевірок правильного введення вихідних даних розрахунку. Використання блочної інструкції керування дозволило істотно зменшити кількість міток та переходів у програмі, хоча змістовна наповненість програми стала значно розширеною за рахунок формування точних повідомлень про характер помилок при введенні вихідних даних.

Слід зазначити, що в мові програмування FORTRAN передбачено більш розширений варіант структурної інструкції керування, який дозволяє враховувати декілька умов:

```
IF (cond1) THEN
instr for cond1
ELSE IF (cond2) THEN
instr for cond2
ELSE IF (cond3) THEN
instr for cond3
...


ELSE
instr for else cond1
END IF
```

Використання конструкції `ELSE-IF` є зрозуміло зручним для реалізації умовного керування виконанням інструкцій програми в певних випадках, особливо за великою кількістю умов.

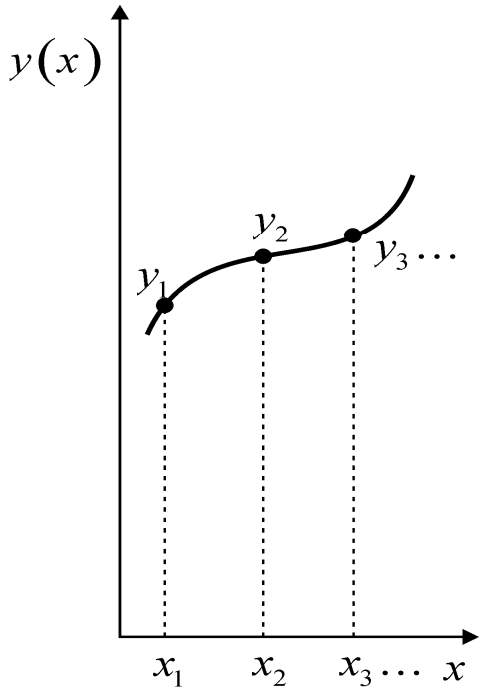
2 ЕЛЕМЕНТИ СТРУКТУРНОГО ПРОГРАМУВАННЯ

Широке впровадження комп'ютерів та програмних продуктів у наукові дослідження та в індустрію призвело до необхідності супроводу програм, яка передбачає виправлення помилок та внесення модифікаційних змін з метою забезпечення коректного та більш зручного подальшого користування. Для забезпечення можливостей щодо зручнішого розуміння тексту програм для пошуку помилок та для модифікації було запропоновано структурний підхід щодо програмування, в якому пропонується створювати програми у вигляді сукупності типових структур [20–22]. Розглянемо далі базові відомості, що необхідні для розуміння елементів структурного програмування та їхнього використання при розробці програм для виконання обчислень.

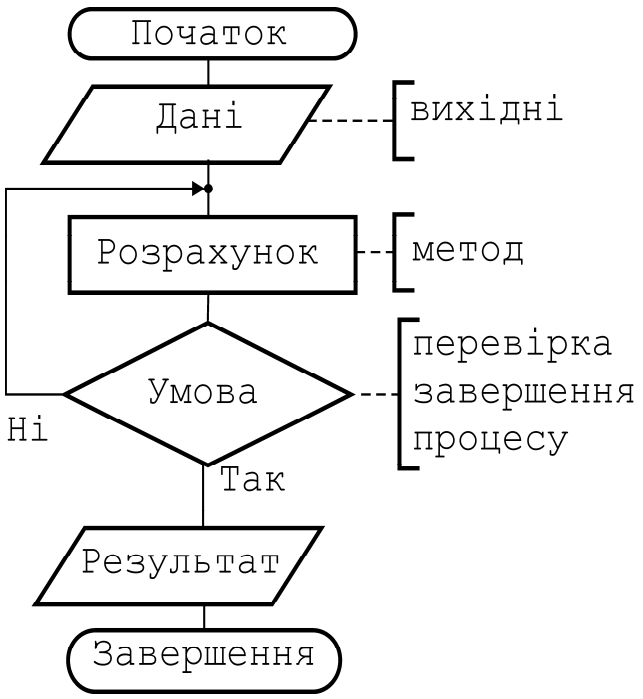
2.1 Циклічний повтор виконання послідовностей інструкцій

Циклічні повторення послідовності інструкцій є одним із елементів структурного програмування, тобто такі повторення є необхідними для створення програм різного призначення. Далі розглянемо базові поняття щодо циклічного виконання послідовностей інструкцій в програмах на прикладі типових обчислювальних методів, що використовуються для розв'язування задач прикладної математики в наукових, інженерних, економічних та інших розрахунках.

2.1.1 Зведення задач до циклічного виконання інструкцій. Якщо деяка математична задача не може бути розв'язана шляхом аналітичних перетворень, то можна скористатися обчислювальними методами [23, 24]. Для цього спочатку розв'язок задачі слід представити набором чисел, як, наприклад, вектор представляється ординатами, а функція – її значеннями в окремих точках (рис. 2.1а). Далі слід запропонувати обчислення, циклічне виконання яких приведе нас до чисел, що представляють розв'язок розглянутої задачі. Слід зазначити, що не завжди кількість циклічних повторень обчислень, необхідну для одержання розв'язку, можна визначити заздалегідь; іноді завершення циклу обчислень слід здійснювати за деякою відповідним чином встановленою умовою. Наприкінці слід розробити програму за схемою, що показана на рис. 2.1б, та виконати необхідні обчислення.



а)



б)

а) представлення розв'язку набором чисел;

б) блок-схема розрахунку

Рисунок 2.1 – Розв'язок математичної задачі обчислювальними методами

Реалізацію розглянутих загальних рекомендацій щодо використання циклічного повторення послідовностей інструкцій в програмах покажемо далі на прикладі розв'язування математичної задачі про пошук кореня рівняння у заданому інтервалі:

$$x:\ \ f(x)=0, x\in[a,b], \tag{2.1}$$

де $f(x)$ – функція довільного вигляду; a, b – границі інтервалу, в якому визначається корінь рівняння.

Необхідність використання обчислювальних методів для розв'язку задачі (2.1) обумовлена довільним виглядом функції $f(x)$, що не дозволяє використовувати аналітичні методи. Розв'язок задачі (2.1) є числом, тому будуємо обчислювальну схему для безпосереднього розв'язування цієї задачі – визначення значення кореня. Для цього скористаємося методом половинного ділення інтервалу, який засновано на відомому факті математичного аналізу, що при існуванні єдиного кореня рівняння (2.1) в інтервалі $[a,b]$ виконується умова $f(a)f(b)<0$ (рис. 2.2). В цих умовах маємо,

що середина $x_1 = (a+b)/2$ інтервалу $[a,b]$ має бути більш близькою до шуканого кореня, ніж один із країв цього інтервалу (рис. 2.2). До якого саме краю є ближчим корінь рівняння, можна встановити перевіркою умови

$$f(a)f(x_1) < 0. \qquad (2.2)$$

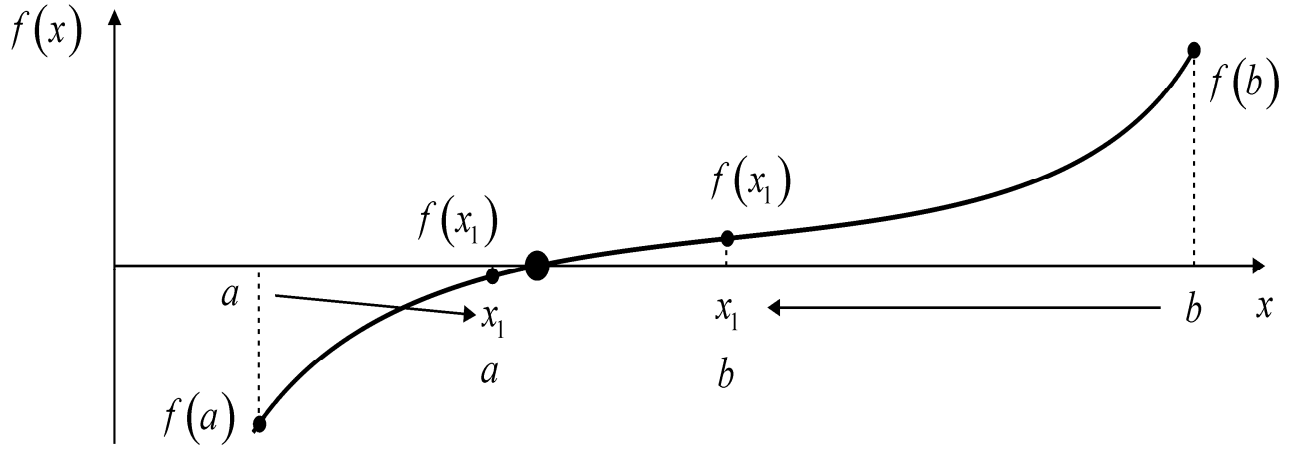


Рисунок 2.2 – Пошук кореня шляхом половинного ділення інтервалу

Дійсно, якщо ця умова виконується, то край a є ближчим до шуканого кореня, який далі можна визначати на меншому інтервалі $[a,b]$, де $b = x_1$; при невиконанні цієї умови край b інтервалу є ближчим до шуканого кореня, який можна далі визначати на меншому інтервалі $[a,b]$, де $a = x_1$. Далі слід знову визначити середину $x_1 = (a+b)/2$ нового інтервалу $[a,b]$, перевірити умову (2.2) і відповідним чином зменшити довжину $\varepsilon = b - a$ інтервалу $[a,b]$, яка є похибкою визначення кореня.

Довжину $\varepsilon = b - a$ інтервалу слід зменшувати, поки вона не стане достатньо малою, щоб представляти числове значення шуканого кореня. Таким чином, шляхом циклічного ділення інтервалу на дві частини наближаємося до кореня рівняння; схема алгоритму визначення кореня рівняння (2.1) представлена на рис. 2.3. Після того, як довжина $\varepsilon = b - a$ інтервалу $[a,b]$ буде достатньо зменшеною, розв'язок рівняння (2.1) можна визначити як $x = (a+b)/2$ з похибкою ε.

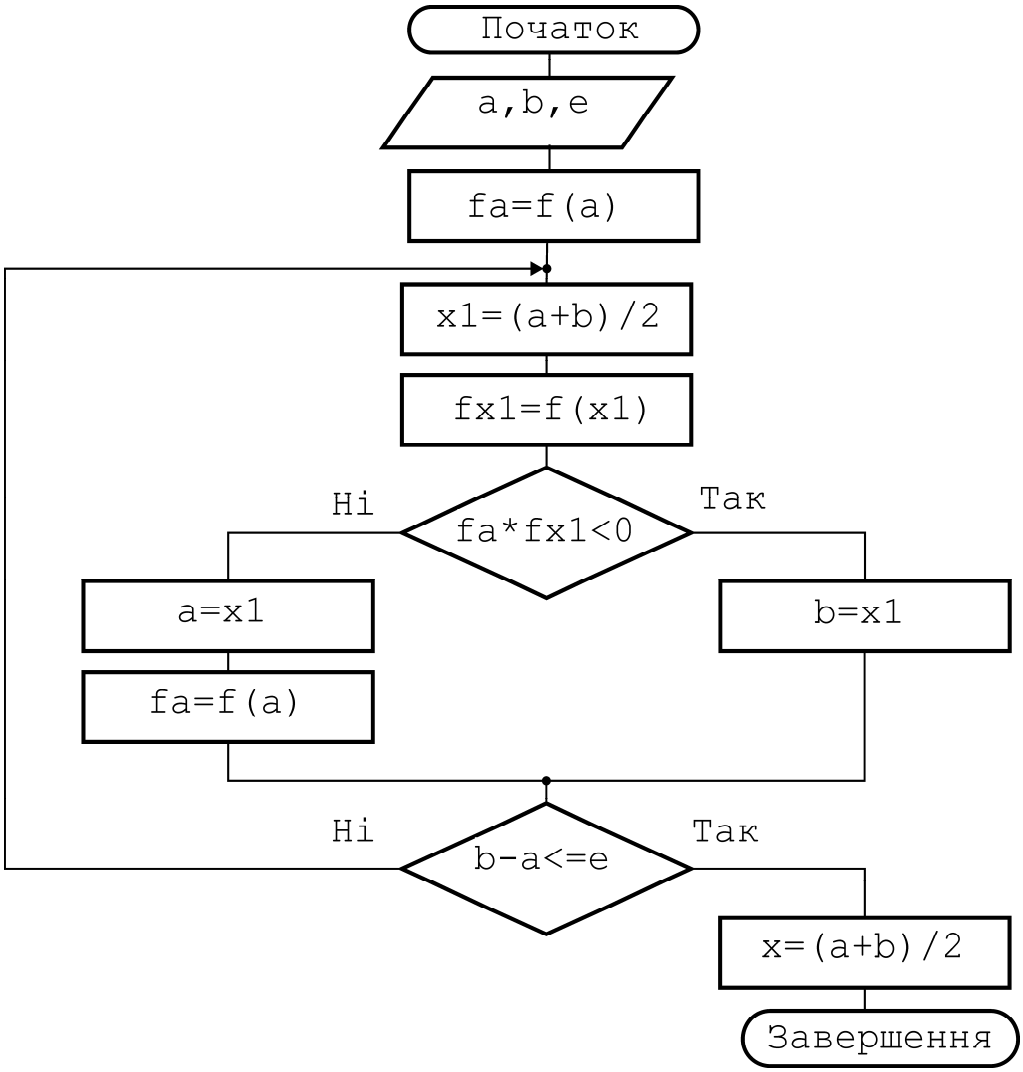


Рисунок 2.3 – Схема алгоритму методу половинного ділення

2.1.2 Циклічний повтор виконання інструкцій шляхом переходів. Розглянемо можливий варіант програми розв'язання рівняння (2.1), яка реалізується методом половинного ділення інтервалу, як представлено вище на рис. 2.3. В якості функції використовуємо наступну функцію:

$$f(x) = \mathrm{tg}(x) - x. \tag{2.3}$$

Розв'язки рівняння (2.1), що відповідає функції (2.3), можна наочно уявляти як ординати точок перетину графіків функцій $y = x$ та $y = \mathrm{tg}(x)$, як показано на рис. 2.4; бачимо (рис. 2.4), що рівняння (2.1) для функції (2.3) має нескінченну кількість коренів, тому далі розглянемо визначення кореня рівняння (2.1) для функції (2.3) в інтервалі, який має границі:

$$a = 3, \quad b = 5. \tag{2.4}$$

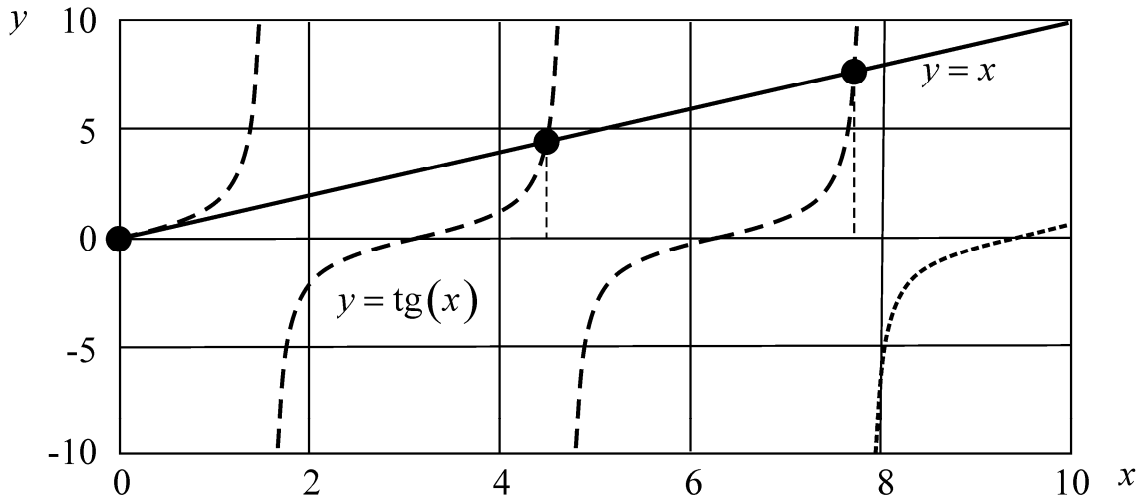


Рисунок 2.4 – Геометричне розв'язання рівняння

Текст програми, написаний мовою програмування FORTRAN, в якій здійснюється розв'язання рівняння (2.1) для функції (2.3) в інтервалі, що визначається границями (2.4), має містити дві інструкції керування та може мати наступний вигляд:

Лістинг 2.1 – Визначення кореня рівняння

```
      PROGRAM ROOT1
      REAL A,B,X1,X,FA,FX1,E
      A=3.0
      B=5.0
      E=1.0E-6
      FA=TAN(A)-A
100   X1=(A+B)/2
      FX1=TAN(X1)-X1
      IF ((FA*FX1).LT.0) THEN
      B=X1
      ELSE
      A=X1
      FA=TAN(A)-A
      END IF
      IF ((B-A).GT.E) GOTO 100
      X=(A+B)/2
      PRINT*,'X=',X
      PRINT*,'TAN(X)=',TAN(X)
      PAUSE 'PRESS ENTER TO EXIT PROGRAM'
      END
```

Представлена програма дозволяє одержати розв'язок рівняння (2.1) для функції (2.3) та границь інтервалу (2.4) з достатньо високою точністю.

В програмі не передбачається введення вихідних даних, тобто вона дозволяє одержувати один із коренів рівняння (2.1) для вихідних даних (2.3) та (2.4) з погрішністю $\varepsilon = 10^{-6}$, але в такій програмі нескладно передбачити введення вихідних даних таким чином, щоб можна було визначати будь-який із коренів рівняння (2.1) для функції (2.3) із заданою точністю, що можна запропонувати для самостійного виконання. Циклічне виконання інструкцій в програмі з лістингу 2.1 реалізовано шляхом використання міток та інструкцій умовного переходу.

Для обчислення тангенса, що міститься у функції (2.3), в програмі використано передбачену в FORTRAN стандартну функцію [6, 9]

```
TAN(X).
```
(2.5)

Стандартна функція надає результат у вигляді числового значення, обчисленого відповідно до імені функції для значень аргументів, що наводяться одразу ж після імені функції в круглих дужках. Значення аргументів стандартної функції можуть бути представленими у вигляді числової константи або імені змінної, значення якої слід розглядати як аргумент функції. Окрім стандартної функції тангенса (2.5) в мові програмування FORTRAN передбачені також інші стандартні функції, які можуть бути потрібними при обчисленнях [6, 9]:

Таблиця 2.1 – Деякі стандартні функції мови програмування FORTRAN

Математична функція	Запис на мові програмування
$\cos(x)$	`COS(X)`
$\sin(x)$	`SIN(X)`
$\arccos(x)$	`ACOS(X)`
e^x	`EXP(X)`
$\ln(x)$	`LOG(X)`
$\lvert x \rvert$	`ABS(X)`
$\lg(x)$	`LOG10(X)`
$\min(x_1, x_2)$	`MIN(X1,X2)`
$\mathrm{sh}(x)$	`SINH(X)`

Наведені в табл. 2.1 стандартні функції не вичерпують всього різноманіття стандартних функцій мови програмування FORTRAN, з якими можна ознайомитись зі спеціальної літератури [6, 9].

2.1.3 Підрахування кількості виконаних циклів розрахунків. Завершення циклічних розрахунків здійснюється шляхом перевірки відповідної умови, але може статися так, що ця умова не буде виконана, наприклад, через наближений характер обчислень, і в цьому випадку програма буде нескінченно здійснювати розрахунки. Для уникнення подібних випадків при виконанні циклічних розрахунків передбачають обчислення кількості виконаних циклів, щоб мати можливість обмежити кількість циклів обчислень. В мові програмування FORTRAN передбачені змінні цілого типу INTEGER, які дозволяють, наприклад, підраховувати кількість виконаних циклів обчислень, що мають бути представлені цілими числами.

Приклад програми, що розв'язує рівняння (2.1) для функції (2.3) в інтервалі з границями (2.4), наведений в лістингу 2.2. Для визначення кількості здійснених циклів розрахунків, а також для визначення максимально можливої кількості циклічних обчислень при наближеному розв'язуванні рівняння в цьому прикладі передбачені дві змінні цілого типу. Перед початком циклічних розрахунків змінну цілого типу – лічильник кількості циклів слід дорівняти до нуля, а при закінченні кожного циклу обчислень – збільшувати на одиницю. В умові, що визначає закінчення обчислень, окрім перевірки поточної довжини інтервалу визначення кореня, слід врахувати також умову, щоб поточна кількість виконаних циклів розрахунків не перевищувала заданого значення. Максимальна можлива величина кількості циклів розрахунку має бути достатньо великою величиною, яка має перевищувати кількість циклів, що потрібні для розв'язування рівняння із заданою точністю. При вдало побудованому алгоритмі розв'язування задачі кількість циклів розрахунків, необхідних для одержання розв'язку задачі із потрібною точністю, не буде дуже великою.

Лістинг 2.2 – Визначення кореня рівняння із підрахуванням кількості циклів обчислень

```
      PROGRAM ROOT2
      REAL A,B,X1,X,FA,FX1,E
      INTEGER I,N
      A=3.0
      B=5.0
      N=100
      E=1.0E-6
      FA=TAN(A)-A
      I=0
100   X1=(A+B)/2
      FX1=TAN(X1)-X1
      I=I+1
      IF ((FA*FX1).LT.0) THEN
```

```
      B=X1
      ELSE
      A=X1
      FA=TAN(A)-A
      END IF
      IF (((B-A).GT.E).AND.(I.LE.N)) GOTO 100
      X=(A+B)/2
      PRINT*,'X=',X
      PRINT*,'TAN(X)=',TAN(X)
      PRINT*,'I=',I
      PAUSE 'PRESS ENTER TO EXIT PROGRAM'
      END
```

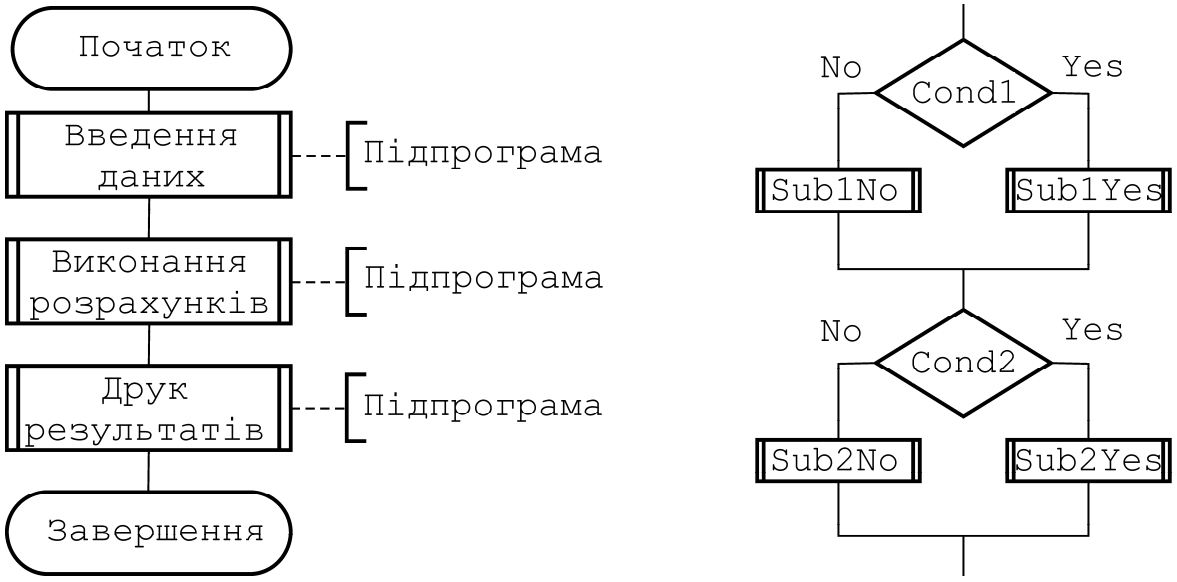
2.2 Використання підпрограм

В деяких програмах необхідно багато у різних місцях виконувати однакові послідовності деяких заданих інструкцій при різних значеннях деяких із змінних, і текст такої програми буде дуже громіздким, якщо в ній у великій кількості різних місць будуть «механічно» повторюватися однакові послідовностей деяких заданих інструкцій. В сучасних мовах програмування передбачається можливість одноразового окремого завдання послідовностей деяких заданих інструкцій для подальшого багаторазового їхнього використання в програмах [25–27], що є одним із елементів структурного програмування, яке забезпечує необхідні властивості створюваним програмам.

2.2.1 Поняття про підпрограми. Підпрограма є одним із способів організації послідовності інструкцій, що використовується в структурному програмуванні, і являє собою незалежну послідовність деяких заданих інструкцій, виконання якої можна здійснювати із будь-якого місця програми із подальшим поверненням до наступної інструкції цієї програми. Кажуть, що підпрограма «викликається», тобто програма передає керування відповідній підпрограмі. Після завершення свого виконання викликана підпрограма повертає керування назад до програми, що її викликала, таким чином, що керування буде передано інструкції програми, яка розташована слідом за інструкцією, що викликала підпрограму.

Є цілком зрозумілим, що використання підпрограм суттєво спрощує розробку програм, в яких необхідно в різних місцях виконувати певні задані однакові послідовності інструкцій, оскільки такі однакові послідовності інструкцій можна оформити у вигляді підпрограми та викликати у потрібних місцях програми. В той же час, використання підпрограм іноді є доцільним для раціоналізації будови програм шляхом їхнього структурування. Так, програма може містити три підпрограми, що виконуються послідовно: підпрограму введення даних, підпрограму розрахунків, підпрограму друку

результатів на екрані монітора (рис. 2.5а). Підпрограми також можна використовувати для спрощення структурної інструкції `IF` за рахунок більш компактного її представлення за рахунок використання коротких викликів підпрограм замість довгих послідовностей інструкцій, що мають виконуватися при виконанні тих чи інших умов (рис. 2.5б). Звичайно, що питання про доцільність використання підпрограм має розв'язуватися в кожному конкретному випадку.



а) *б)*

а) виділення етапів програми; *б) структуризація програми;*

Рисунок 2.5 – Приклади використання підпрограм

В деяких мовах програмування, наприклад у FORTRAN та PASCAL, передбачені два різновиди підпрограм: підпрограми-функції та підпрограми-процедури, а в деяких мовах програмування, наприклад в C, такого розділення підпрограм не передбачається і усі підпрограми є функціями. В деяких мовах програмування немає жодних інших конструкцій, окрім підпрограм-процедур та підпрограм-функцій, і будь-яка програма представляється у вигляді послідовностей підпрограм-процедур та підпрограм-функцій. В будь-якому випадку потрібно розуміти різницю між підпрограмами-процедурами та підпрограмами-функціями для використання у подальшому різних мов програмування. Підпрограми є окремими програмними одиницями та визначаються окремо від програми; в мові програмування FORTRAN підпрограми необхідно визначати після інструкції `END` завершення програми.

2.2.2 Підпрограми-функції. Поняття про підпрограму-функцію в програмуванні суттєво пов'язане із поняттям функції в математиці, яке формулюється наступним чином [28]: нехай маємо множину X, яка складається із елементів x, а також множину Y, яка складається із елементів y, і нехай

якимось способом f елементу $x \in X$ ставиться у відповідність елемент $y \in Y$, тоді відповідність $x \to y$, або $y = f(x)$ називають функцією із областю визначення X та областю значень Y.

У програмуванні під поняттям підпрограма-функція розуміють підпрограму, яка має певний результат у вигляді значення даних певного типу, що визначається певним чином за набором вхідних параметрів, що у загальному випадку являють собою множину значень даних певних типів. Тобто підпрограма-функція є аналогом математичної функції таким чином, що результат підпрограми-функції відповідає значенню функції в математичному сенсі, а множина вхідних даних підпрограми-функції – аргументу функції в математичному сенсі. При виклику підпрограми-функції керування передається відповідній підпрограмі, і в результаті по закінченні інструкцій цієї підпрограми до програми передається результат – значення функції. Передбачається, що підпрограми-функції визначаються окремо та можуть використовуватися у будь-якому місці програмі. Природно використовувати підпрограми-функції, коли в декількох місцях програми потрібно виконувати однакові послідовності інструкцій при різних значеннях деякого заданого набору змінних програми; в цьому випадку в якості аргументу функції доцільно передбачити відповідний заданий набір змінних, а саму функцію виконати у вигляді відповідної послідовності інструкцій.

В якості прикладу, де доцільно використовувати підпрограми-функції, можна навести розв'язування рівняння $f(x) = 0$ методом половинного ділення, в якому необхідно у двох місцях програми визначати значення функції $f(x)$ для різних значень аргументу x (рис. 2.6). Значущість проблеми двократного визначення значення функції $f(x)$ не є очевидною, коли ця функція має нескладний вигляд, який дозволяє записувати її в одному рядку шляхом однієї інструкції присвоювання, як, наприклад, у випадку, коли $f(x) = \mathrm{tg}(x) - x$. В той же час, навіть в цьому нескладному випадку довелося використати передбачену стандартну функцію `TAN(X)` для обчислення тангенса, і ситуація суттєво ускладнюється за умов відсутності можливостей обчислення тригонометричних функцій шляхом використання стандартних функцій. Таким чином, правило обчислення значення функції $f(x)$ в програмі, що здійснює розв'язування рівняння $f(x) = 0$, доцільно сформулювати окремо у вигляді підпрограми-функції та використовувати цю підпрограму в необхідних місцях програми.

Підпрограма-функція в мові програмування FORTRAN визначається за допомогою інструкції `FUNCTION`, після якої розташовують інструкції підпрограми-функції, які мають завершитися інструкцією `END`, що взагалі можна представити наступним чином [6, 9]:

```
type FUNCTION name(arguments)
...
name = ...
END
```

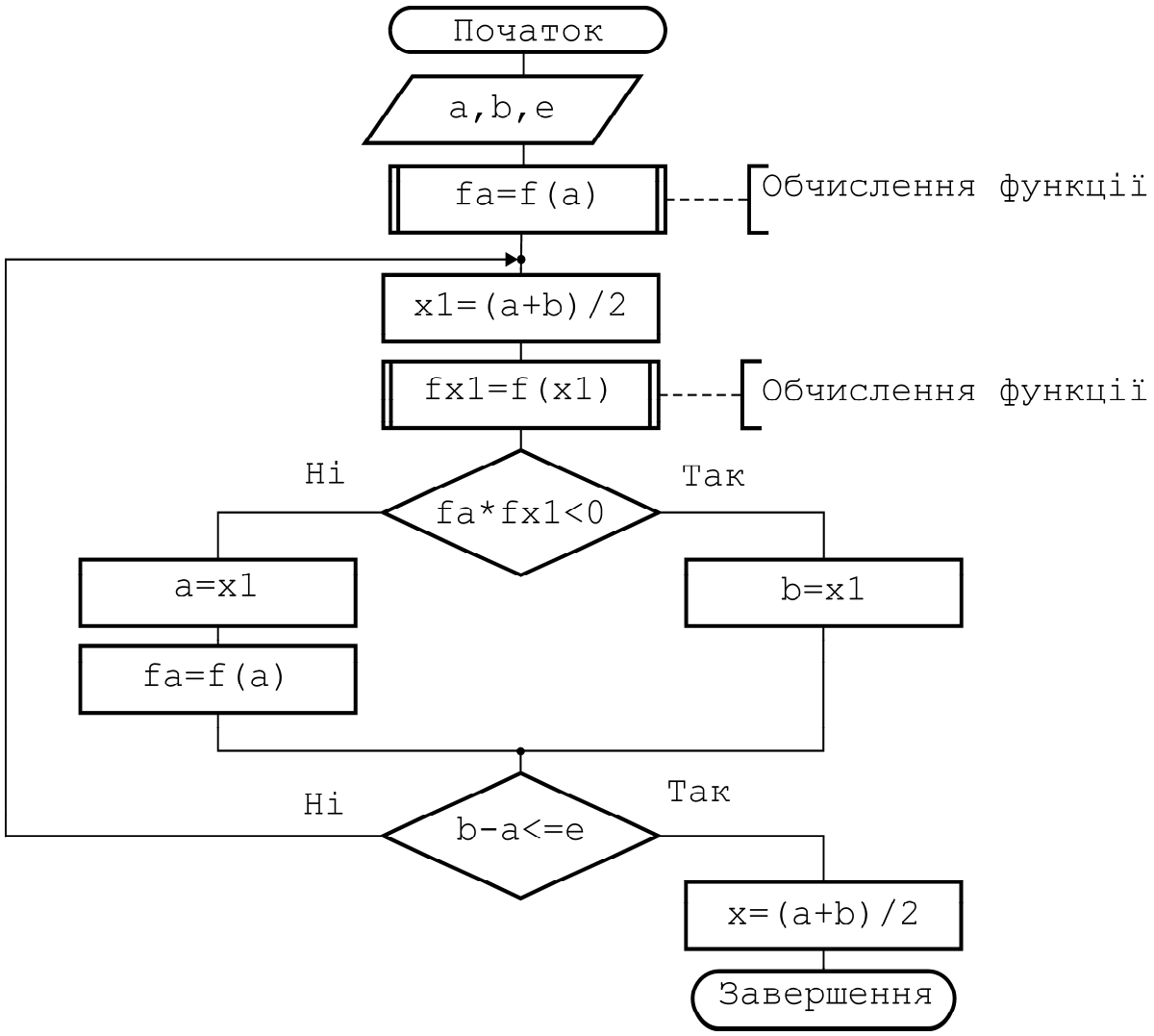


Рисунок 2.6 – Використання підпрограми-функції в алгоритмі методу половинного ділення

В описі підпрограми-функції слід визначити тип даних значення функції, що здійснюється за допомогою безпосереднього вказування типу `type`; слід також визначитися з іменем функції, яке має міститися в `name`, а також із вхідними параметрами (аргументами), що позначені як `arguments` та мають вказуватися через кому в дужках одразу ж після імені функції. Серед інструкцій підпрограми функції обов'язково має міститися інструкція присвоювання змінної, ім'я якої збігається із іменем функції, що необхідно для визначення значення функції. Типи вхідних параметрів підпрограми-функції мають бути визначені за допомогою відповідних

інструкцій, як це робиться в програмах; підпрограма-функція може мати також «локальні» змінні, тип яких визначається за допомогою інструкцій визначення, як у програмах. Якщо в програмі використовуються підпрограми-функції, то їхні імена мають бути визначені разом із змінними програми:

Лістинг 2.3 – Визначення кореня рівняння із використанням підпрограми-функції

```
      PROGRAM ROOT3
      REAL A,B,X1,X,FA,FX1,E,F
      A=3.0
      B=5.0
      E=1.0E-6
      FA=F(A)
100   X1=(A+B)/2
      FX1=F(X1)
      IF ((FA*FX1).LT.0) THEN
      B=X1
      ELSE
      A=X1
      FA=FX1
      END IF
      IF ((B-A).GT.E) GOTO 100
      X=(A+B)/2
      PRINT*,'X=',X
      PRINT*,'TAN(X)=',TAN(X)
      PAUSE 'PRESS ENTER TO EXIT PROGRAM'
      END
      REAL FUNCTION F(X)
      REAL X
      F=TAN(X)-X
      END
```

Виклик підпрограми-функції в програмі здійснюється шляхом звернення до її імені із переліченням в дужках через коми списку вхідних параметрів у вигляді констант та (або) імен змінних; результат підпрограми-функції може бути збережений у змінній, а також використаний у виразах різного типу.

У випадку, коли в програмі потрібно визначати корені рівняння для різних інтервалів, доцільно оформити визначення кореня рівняння у вигляді окремої підпрограми-функції, аргументами якої є границі інтервалу пошуку кореня та допустима похибка, як, наприклад, в лістингу 2.4. У наведеному прикладі границі інтервалу пошуку кореня є константами, але вони можуть бути також змінними, як і допустима похибка.

2.2.3 Підпрограми-процедури. Поняття про підпрограму-процедуру зводиться до простої послідовності інструкцій, які мають бути виконані при зверненні з програми; після виконання інструкцій підпрограми-процедури керування передається інструкції програми, яка розташована одразу ж за інструкцією, що викликала цю підпрограму-процедуру.

Підпрограма-процедура в мові програмування FORTRAN визначається за допомогою інструкції SUBROUTINE, після якої розташовують інструкції підпрограми-процедури, які мають завершитися інструкцією END, що взагалі можна представити наступним чином [6, 9]:

```
SUBROUTINE name(arguments)
...
END
```

В описі підпрограми-процедури слід визначити її ім'я name, за допомогою якого програма буде викликати цю підпрограму-процедуру, а також вхідні параметри arguments, що мають вказуватися через кому в дужках одразу ж після імені. Серед інструкцій підпрограми-процедури обов'язково мають міститися визначення типів вхідних параметрів, а також «локальних» змінних, тип яких має бути визначений відповідним чином, як у програмах. Виклик підпрограм-процедур [6, 9] здійснюється за допомогою інструкції CALL. Далі на лістингу 2.5 представлений приклад підпрограми-процедури, що одразу ж визначає два сусідні корені рівняння $\mathrm{tg}(x) - x = 0$.

Лістинг 2.4 – Визначення кореня рівняння, що оформлене у вигляді окремої підпрограми-функції

```
      PROGRAM ROOT4
      REAL E,X1,X2,ROOT
      E=1.0E-6
      X1=ROOT(3.0,5.0,E)
      X2=ROOT(7.0,8.0,E)
      PRINT*,'X1=',X1,';   ','TAN(X1)=',TAN(X1)
      PRINT*,'X2=',X2,';   ','TAN(X2)=',TAN(X2)
      PAUSE 'PRESS ENTER TO EXIT PROGRAM'
      END
      REAL FUNCTION ROOT(A,B,E)
      REAL A,B,E,X,FA,FX,F,AA,BB
      AA=A
      BB=B
      FA=F(AA)
100   X=(AA+BB)/2
      FX=F(X)
      IF ((FA*FX).LT.0) THEN
```

```
      BB=X
      ELSE
      AA=X
      FA=FX
      END IF
      IF ((BB-AA).GT.E) GOTO 100
      ROOT=(AA+BB)/2
      END
      REAL FUNCTION F(X)
      REAL X
      F=TAN(X)-X
      END
```

Лістинг 2.5 – Визначення двох сусідніх коренів рівняння з використанням спеціальної підпрограми-процедури

```
      PROGRAM ROOT5
      REAL X1,X2
      CALL ROOTS(4.0,5.0, 1.0E-6,X1,X2)
      PRINT*,'X1=',X1,';   ','TAN(X1)=',TAN(X1)
      PRINT*,'X2=',X2,';   ','TAN(X2)=',TAN(X2)
      PAUSE 'PRESS ENTER TO EXIT PROGRAM'
      END
      SUBROUTINE ROOTS(A,B,E,X1,X2)
      REAL A,B,X1,X2,E,ROOT
      X1=ROOT(A,B,E)
      X2=ROOT(A+3.0,B+3.0,E)
      END
      REAL FUNCTION ROOT(A,B,E)
      REAL A,B,E,X,FA,FX,F,AA,BB
      AA=A
      BB=B
      FA=F(AA)
100   X=(AA+BB)/2
      FX=F(X)
      IF ((FA*FX).LT.0) THEN
      BB=X
      ELSE
      AA=X
      FA=FX
      END IF
      IF ((BB-AA).GT.E) GOTO 100
      ROOT=(AA+BB)/2
      END
```

```
      REAL FUNCTION F(X)
      REAL X
      F=TAN(X)-X
      END
```

2.3 Створення програмних одиниць для багаторазового використання

При створенні великих програм окремі послідовності інструкцій зручно групувати в окремих програмних компонентах та складати великі програми із таких програмних компонент [6, 9]. Деякі із програмних компонент природно можуть бути використані в різних програмних проектах, як, наприклад, ті, що здійснюють стандартні математичні обчислення – визначення коренів рівнянь, визначених інтегралів і т.п. Використання готових програмних компонент, що розв'язують стандартні задачі, суттєво прискорює створення великих програмних систем, тому передбачається в усіх розвинених мовах програмування і є елементом структурного програмування. Програмні компоненти, що призначені для багаторазового використання, мають бути підготовлені за відповідними правилами та належним чином забезпечені описами.

2.3.1 Використання підпрограм в якості аргументів інших підпрограм. В деяких випадках буває зручним передавати підпрограму (функцію, або процедуру) як аргумент до іншої підпрограми [6, 9, 29]. Можливість розглядати підпрограму як аргумент іншої підпрограми надає надвеликих можливостей щодо створення бібліотек підпрограм різного призначення, які можуть бути використані в різних програмах.

Природним прикладом, в якому зручно передавати підпрограму як аргумент до іншої підпрограми, є розв'язування рівняння $f(x)=0$ в заданому інтервалі $[a,b]$. Дійсно, уявимо собі, що в програмі на проміжному етапі великих обчислень для забезпечення розрахунків необхідно розв'язувати два різні рівняння:

$$x:\ \ f_1(x)=0, x\in[a_1,b_1], \tag{2.6}$$

$$x:\ \ f_2(x)=0, x\in[a_2,b_2], \tag{2.7}$$

де $f_1(x)$ та $f_2(x)$ – задані функції; a_1, b_1 та a_2, b_2 – задані границі інтервалів, що містять корені відповідних рівнянь.

У випадку, коли $f_1(x)$ та $f_2(x)$ збігаються, визначення різних коренів рівняння можна здійснити однією функцією, що реалізує, наприклад, метод половинного ділення, в якій передбачити границі інтервалу в якості вхідних

параметрів – аргументів. Без використання функції в якості аргументу для різних функцій $f_1(x)$ та $f_2(x)$ слід створювати окремі функції для розв'язування рівнянь (2.3.1) та (2.3.2), але такі функції будуть відрізнятися виключно викликами функції у відповідних місцях. Якщо, окрім границь інтервалу, що містить корінь, також передбачити використання функції в якості аргументу функції, що здійснює розв'язок рівняння $f(x)=0$ в заданому інтервалі $[a,b]$ методом половинного ділення, то одна така функція може бути використана для розв'язування різних рівнянь, у тому числі (2.6) та (2.7), що буде значно економити обсяг програмного коду в програмах, які потребують розв'язування багатої кількості різних рівнянь.

Для забезпечення передачі підпрограми в якості аргументу до іншої підпрограми в списку вхідних параметрів (аргументів) цій іншій підпрограмі слід передбачити відповідний формальний параметр. При виконанні підпрограми, що містить підпрограму в якості аргументу, виклик цієї підпрограми-аргументу здійснюється як у звичайних програмах. При цьому у програмній компоненті, що викликає підпрограму, яка містить інші підпрограми в якості аргументів, усі підпрограми, що використовуються в якості аргументів, мають бути позначені інструкцією EXTERNAL, щоб відрізняти імена підпрограм-аргументів від імен змінних. З використанням такого підходу універсальна підпрограма-функція для визначення окремих коренів рівнянь (2.6) та (2.7) у випадку наступних вихідних даних: $f_1(x)=\operatorname{tg}x-x$, $a_1=4$, $b_1=5$ та $f_2(x)=x^3-6x^2+12x-8$, $a_2=1$, $b_2=3{,}5$ може бути створеною та використовуватися у різних програмах, як показано нижче:

Лістинг 2.6 – Програма із функціями-аргументами іншої функції

```
PROGRAM ROOT6
REAL X1,X2,ROOT
EXTERNAL F1,F2
X1=ROOT(4.0,5.0,1.0E-6,F1)
X2=ROOT(1.0,3.5,1.0E-6,F2)
PRINT*,'X1=',X1,';   F1(X1)=',F1(X1)
PRINT*,'X2=',X2,';   F2(X2)=',F2(X2)
PAUSE 'PRESS ENTER TO EXIT PROGRAM'
END
REAL FUNCTION F1(X)
REAL X
F1=TAN(X)-X
END
REAL FUNCTION F2(X)
REAL X
F2=X**3-6*X**2+12*X-8
```

```
      END
      REAL FUNCTION ROOT(A,B,E,F)
      REAL A,B,E,X,FA,FX,F,AA,BB
      AA=A
      BB=B
      FA=F(AA)
100   X=(AA+BB)/2
      FX=F(X)
      IF ((FA*FX).LT.0) THEN
      BB=X
      ELSE
      AA=X
      FA=FX
      END IF
      IF ((BB-AA).GT.E) GOTO 100
      ROOT=(AA+BB)/2
      EN
```

2.3.2 Створення проектів, що складаються із декількох файлів. Великі програми зручно представляти як сукупність відносно незалежних програмних компонент, що дозволяє, принаймні, розділити роботу програмістів шляхом зосередження на певних програмних компонентах та є притаманним сучасним технологіям та передбачено в усіх сучасних мовах програмування.

При розділені програми на програмні компоненти слід визначитися, яка саме із компонент отримуватиме керування на початку виконання програми; така програмна компонента називається головною програмою. В мові програмування FORTRAN інструкція головної програми міститься між інструкціями `PROGRAM` та `END`, а усі інші програмні компоненти – підпрограми отримують керування від основної програми. Так, програма, що наведена в лістингу 2.7, складається із чотирьох програмних компонент – основної програми та трьох підпрограм-функцій, що дозволило значно скоротити її обсяг за рахунок багаторазового використання інструкцій підпрограм-функцій. В той же час, нескладно уявити, що тексти великих програм в одному файлі будуть виглядати дуже громіздкими і деякі із програмних компонент зручно розташовувати в окремих файлах. Так, програму, що наведена в лістингу 2.7, зручно представляти у вигляді двох файлів: в першому має бути основна програма та функції, які визначають сутність задачі, що розв'язується програмою; в окремому файлі доцільно оформити програмну компоненту – підпрограму-функцію визначення кореня рівняння. Зручність такого розділення пов'язана із тим, що підпрограма-функція визначення кореня методом половинного ділення може бути

розроблена окремо і використовуватися як програмна компонента в різних програмах, і було б краще, щоб розроблений та перевірений код цієї підпрограми був огороджений від ненавмисних помилкових виправлень при маніпуляціях в тексті основної програми.

Забезпечення можливостей створення програм, що складаються із декількох файлів, при використанні мови програмування FORTRAN здійснюється засобами інтегрованого середовища розробки програм IDE (Integrated development Environment), які розробляються окремо від компіляторів та надають можливостей щодо роботи із програмами. В IDE Open WATCOM передбачається поняття про проект та джерела (sources). Проект має мати ім'я; уся інформація про проект міститься в файлі проекту, що має розширення WPJ, та в низці інших файлів (з різними розширеннями), які автоматично створюються IDE таким чином, що імена усіх файлів проекту збігається із іменем проекту (але ж мають різні розширення). Джерела (sources) містять дані про послідовності інструкцій програмних компонентів, підготовлені у вигляді окремого файлу, зміст якого має відповідати правилам мов програмування, а розширення має відповідати мові програмування; якщо джерело підготовлено мовою програмування FORTRAN 77, то його файл має мати розширення FOR. Імена файлів-джерел не обов'язково мають збігатися з іменем проекту. Розглянемо як приклад використання окремо розробленої підпрограми-функції для визначення коренів рівняння методом половинного ділення. Як найзручніший варіант можна запропонувати створення проекту ROOT7.WPJ, що буде містити два джерела – файли ROOT7.FOR та ROOTS.FOR, як схематично показано на рис. 2.7. У файлі ROOT7.FOR буде міститись основна програма та функції, якими визначаються рівняння, що розв'язуватимуться (лістинг 2.7); у файлі ROOTS.FOR буде міститись підпрограма-функція визначення кореня методом половинного ділення (лістинг 2.8) та, можливо, інші підпрограми, що пов'язані із визначенням коренів рівнянь.

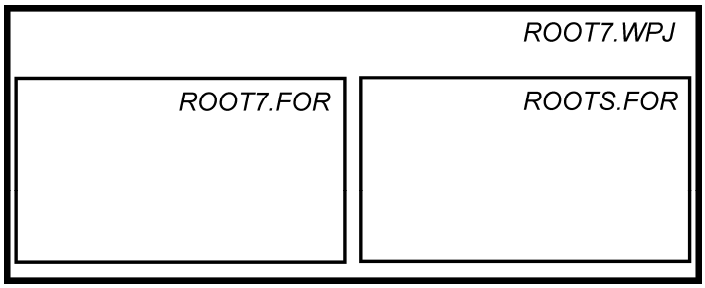


Рисунок 2.7 – Проект та його джерела

Лістинг 2.7 – Зміст файлу ROOT7.FOR

```
PROGRAM ROOT7
REAL X1,X2,ROOT12
EXTERNAL F1,F2
X1=ROOT12(4.0,5.0, 1.0E-6,F1)
X2=ROOT12(1.0,3.5, 1.0E-6,F2)
PRINT*,'X1=',X1,';   ','F1(X1)=',F1(X1)
```

```
      PRINT*,'X2=',X2,';   ','F2(X2)=',F2(X2)
      PAUSE 'PRESS ENTER TO EXIT PROGRAM'
      END
      REAL FUNCTION F1(X)
      REAL X
      F1=TAN(X)-X
      END
      REAL FUNCTION F2(X)
      REAL X
      F2= X**3-6*X**2+12*X-8
      END
```

Лістинг 2.8 – Зміст файлу ROOTS.FOR

```
      REAL FUNCTION ROOT12(A,B,E,F)
      REAL A,B,E,X,FA,FX,F,AA,BB
      AA=A
      BB=B
      FA=F(AA)
100   X=(AA+BB)/2
      FX=F(X)
      IF ((FA*FX).LT.0) THEN
      BB=X
      ELSE
      AA=X
      FA=FX
      END IF
      IF ((BB-AA).GT.E) GOTO 100
      ROOT12=(AA+BB)/2
      END
```

2.3.3 Створення бібліотек підпрограм та їхнє використання в проектах. Багато одних і тих же підпрограм можуть бути використані в різних проектах. Як приклад можна навести підпрограми, що здійснюють стандартні математичні обчислення та розв'язування нескладних задач; до таких підпрограм відноситься функція, що наведена у лістингу 2.8, яка здійснює розв'язування рівняння методом половинного ділення.

На відміну від простої підпрограми, що наведена на лістингу 2.8, деякі «стандартні» підпрограми можуть використовувати інші підпрограми, виконані в окремих файлах, і звичайно, що при приєднанні такої «стандартної» підпрограми до проекту слід приєднувати також усі інші підпрограми, які використовує ця приєднувана «стандартна» підпрограма, навіть якщо ці підпрограми не потрібні в проекті безпосередньо (рис. 2.8). Це буде дуже незручним, і для виправлення таких незручностей запропоновано технологію

використання статичних бібліотек підпрограм. Статична бібліотека – це файл-джерело, записаний у спеціальному форматі та представлений у вигляді файлу із розширенням LIB, в якому містяться підпрограми, які можна використовувати в проекті, а також «внутрішні» підпрограми, які не будуть використовуватися у проекті безпосередньо, але є необхідними для роботи тих підпрограм, що використовуватимуться в проекті (рис. 2.8). Звичайно, що приєднати до проекту один файл-LIB бібліотеки підпрограм значно простіше, ніж приєднувати до проекту окремо усі файли, необхідні для роботи відповідної бібліотеки підпрограм.

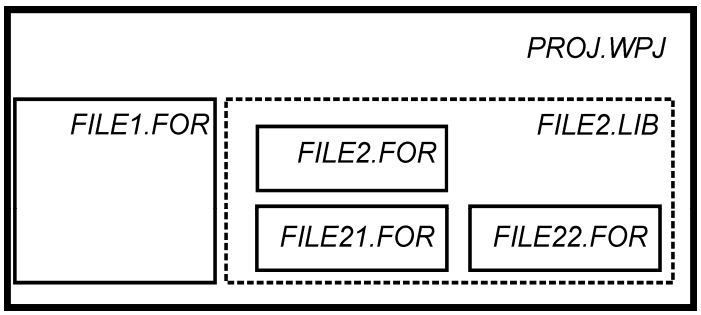


Рисунок 2.8 – Стандартна бібліотека та її використання

Бібліотека підпрограм є проектом, при створенні якого засобами IDE у відповідь на питання про мету (target) проекту в якості мети слід призначити статичну бібліотеку. Далі слід наповнити проект статичної бібліотеки необхідними джерелами – файлами із текстами необхідних підпрограм та здійснити компіляцію проекту, в якій буде створено файл із іменем проекту та розширенням LIB. Саме цей файл можна приєднувати до інших проектів, щоб використовувати присутні в ньому підпрограми. Слід зазначити, що зміст файлу-LIB бібліотеки підпрограм не можна змінювати і не можна з цього файлу одержати інформацію про підпрограми, які в ньому містяться. Для цього потрібно мати доступ до файлів-джерел, написаних мовою програмування, що були приєднані до проекту статичної бібліотеки при її компіляції. Тобто якщо в файлі-LIB бібліотеки підпрограм було виявлено помилку, наприклад, використання певної підпрограми приводить до помилкового результату, то необхідно внести зміни у відповідні файли-джерела, що написані мовою програмування та містяться в проекті цієї статичної бібліотеки, та заново компілювати бібліотеку. Ретельно перевірені статичні бібліотеки часто надають виключно у вигляді файлу-LIB, що не дозволить користувачу помилково змінити його, але файл-LIB бібліотеки підпрограм має супроводжуватися ретельним описом усіх підпрограм, які в ньому реалізуються, щоб користувачі-програмісти мали можливість використовувати ці підпрограми в своїх проектах.

Бібліотеки підпрограм, представлені у вигляді файлів із розширенням LIB, називають статичними бібліотеками, оскільки вони цілком додаються до файлу програми-проекту, яка їх використовує.

Бібліотеки підпрограм – файли-LIB додаються до проекту в якості джерела (source) разом із джерелами інших типів – файлів, написаних

мовою програмування, наприклад файлів-FOR. Підпрограми, що містяться у бібліотеці, яка додана до проекту в якості джерела, використовуються в програмних компонентах проекту як звичайні підпрограми.

В якості прикладу створення та використання статичної бібліотеки можна розглянути створення та використання статичної бібліотеки ROOTS.LIB, яка містить підпрограму-функцію `ROOT12(A,B,E,F)`, яка визначатиме корінь рівняння методом половинного ділення в заданому інтервалі із заданою погрішністю. Для створення цієї бібліотеки необхідно створити проект ROOTS.WPJ і в якості мети (target) проекту обрати Library[.lib], як це показано на рис. 2.9. Далі в цей проект слід додати в якості джерела файл ROOTS.FOR, наведений в лістингу 2.8, та здійснити компіляцію цього проекту. В результаті компіляції одержимо файл ROOTS.LIB, який можемо додавати до проектів в якості джерела. Для використання бібліотеки ROOTS.LIB створимо новий програмний проект ROOT8.WPJ, до якого в якості джерел додамо файл ROOT7.FOR, наведений на лістингу 2.7, та файл ROOTS.LIB бібліотеки. Далі можна здійснити компіляцію та запустити програму.

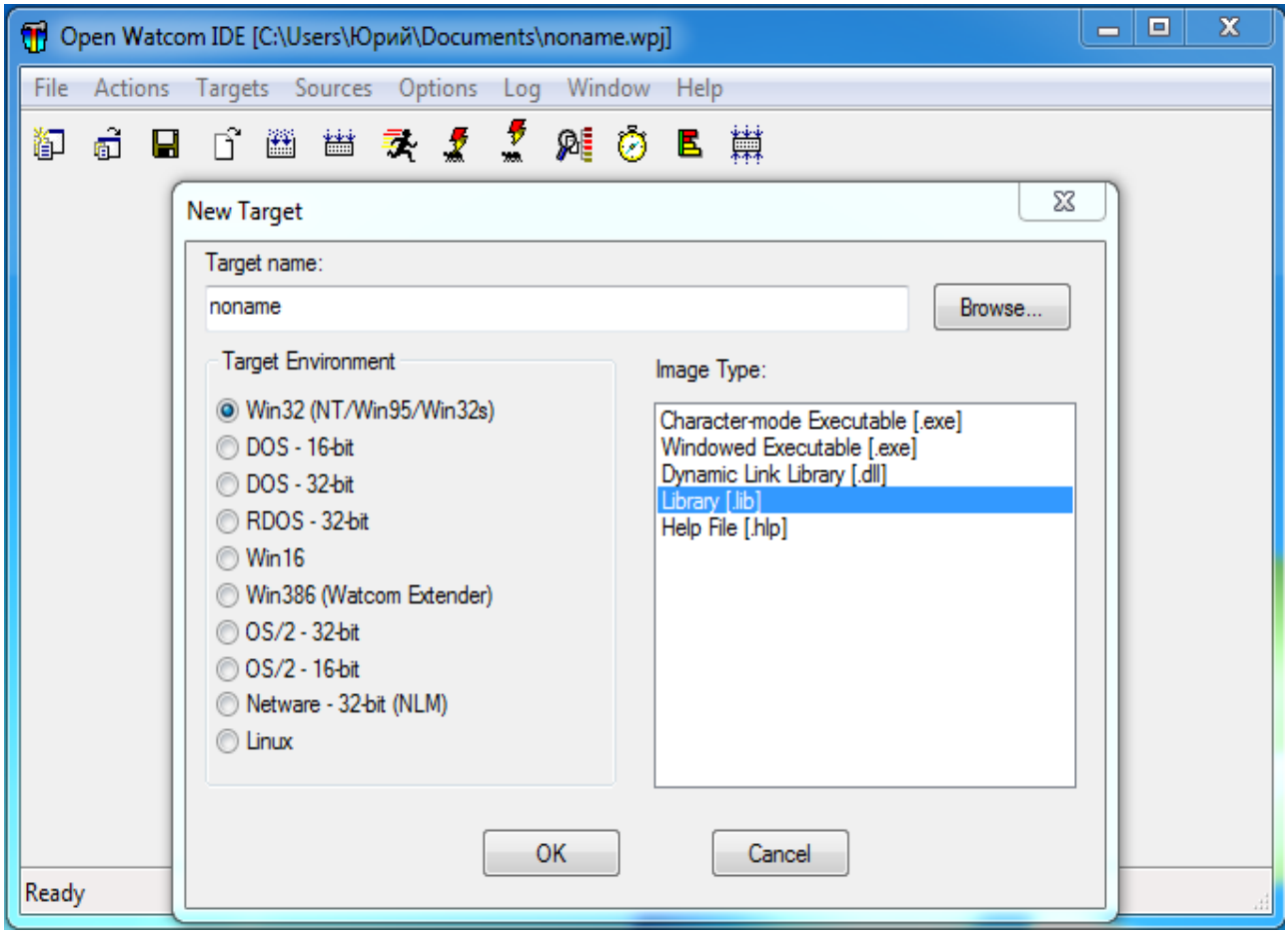


Рисунок 2.9 – Створення проекту статичної бібліотеки

2.4 Виконання заданої кількості циклів послідовностей інструкцій

Виконання заданої кількості циклів повторення деякої послідовності інструкцій взагалі можна здійснити шляхом підрахування кількості вже здійснених циклів та використання інструкцій переходу за умовою, в якій перевіряється кількість вже виконаних циклів. В той же час, у програмах різного призначення достатньо часто доводиться виконувати задану кількість разів деяку послідовність інструкцій, так що здійснення такого виконання відноситься до типових задач програмування. Для прощення програмування типових задач, пов'язаних із циклічним повторенням послідовності інструкцій задану кількість разів, в мовах програмування передбачені інструкції, що забезпечують виконання послідовності інструкцій задану кількість разів та записуються суттєво коротко у порівнянні із загальним підходом, який передбачає безпосереднє підрахування та перевірку кількості здійснених циклів.

2.4.1 Безпосереднє підрахування кількості циклів. При необхідності виконання послідовності інструкцій задану кількість разів принципово потрібно контролювати кількість разів, що вже виконані. Це можна здійснювати шляхом використання змінної цілочислового типу – лічильника циклу, значення якої слід збільшувати на одиницю після здійснення кожного циклу послідовностей інструкцій. Далі слід порівняти значення лічильника циклу із потрібною кількістю циклів і відповідно до результатів порівняння або здійснити наступний цикл, або закінчити циклічне виконання (рис. 2.10).

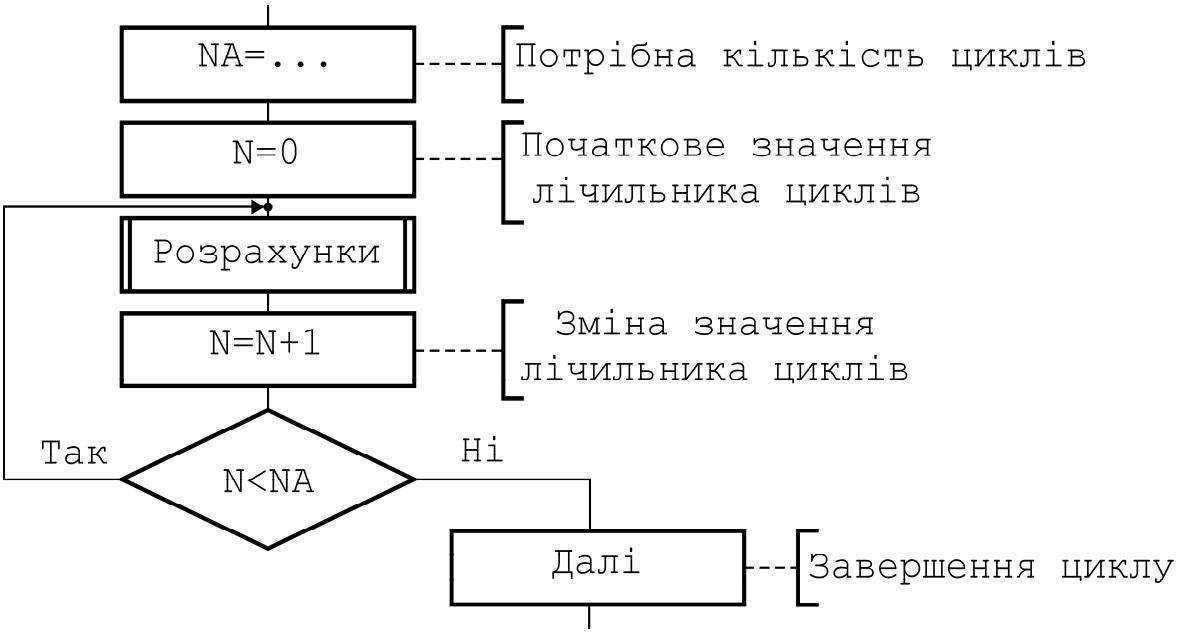


Рисунок 2.10 – Безпосереднє обчислення кількості циклів

Як приклад розглянемо програму обчислення визначеного інтеграла

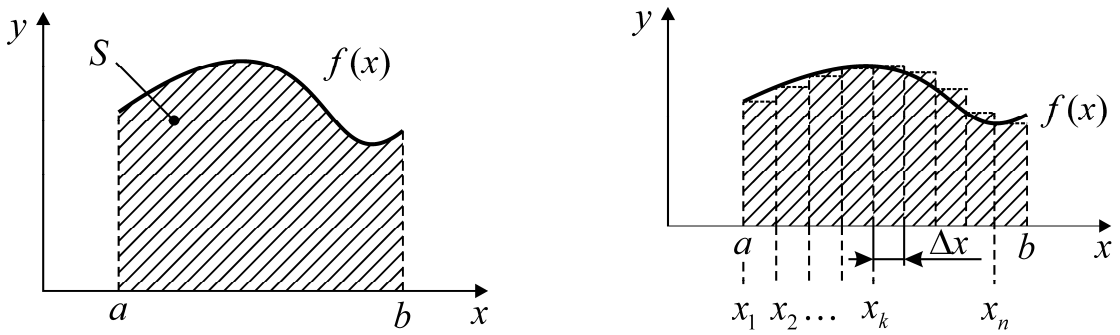
$$S = \int_a^b f(x)dx, \tag{2.8}$$

де a, b – задані границі інтервалу; $f(x)$ – задана функція.

Як добре відомо, геометричним образом визначеного інтегралу (2.8) є площа криволінійної трапеції, що обмежена графіком функції $f(x)$ та показана у вигляді заштрихованої області на рис. 2.11а. З урахуванням цього, розділимо інтервал інтегрування на n частин завдовжки (рис. 2.11б)

$$\Delta x = \frac{b-a}{n}. \tag{2.9}$$

Числове значення S інтеграла (2.8) наближено представимо у вигляді:



а) *б)*

а) геометричний сенс; *б) наближене обчислення;*

Рисунок 2.11 – Визначений інтеграл

$$S \approx S_n, S_n = \sum_{k=1}^{n} f(x_k)\Delta x, \tag{2.10}$$

де x_k – ліва координата частини із номером k інтервалу інтегрування (див. рис. 2.11б).

Безумовно результат розрахунку інтеграла (2.8) у вигляді (2.9), (2.10) матиме похибку, яка буде залежати від числа n; цю похибку можна геометрично уявити як суму площин не заштрихованих областей та площин областей заштрихованих прямокутників, які виходять за межі криволінійної трапеції на рис. 2.11б, що ілюструє невисоку відносну похибку наближеної формули (2.10) при обчисленні визначених інтегралів за умов, якщо число n

обране достатньо великим. Таким чином, програма наближеного обчислення значення визначеного інтеграла (2.8) за формулою (2.9), (2.10) зводиться до обчислення суми із великою кількістю схожих доданків, що можна реалізувати за схемою алгоритму, що представлений вище на рис. 2.10. Розглянемо далі приклад програми обчислення конкретного інтеграла (2.8) за формулою (2.9), (2.10), в якому $f(x)=x^3$, $a=0$, $b=1$; відповідна програма може мати вигляд, як на лістингу 2.9. Такий інтеграл можна підрахувати точно: його значення $S=0,25$; шляхом порівняння одержаного за наближеною формулою результату розрахунків програми із точним значенням можна оцінити похибку розрахунків та дослідити зв'язок між кількістю інтервалів n та цією похибкою.

2.4.2 Непряме підрахування кількості циклів. Оскільки виконання заданої кількості циклічних повторень деякої послідовності інструкцій є розповсюдженим у програмах різного призначення, то в мовах програмування передбачають можливість організації непрямого підрахування кількості циклів, для чого передбачені відповідні інструкції. В таких інструкціях вводиться змінна-лічильник та вказують початкове значення, скінченне значення й крок зміни лічильника.

Лістинг 2.9 – Безпосередній лічильник циклів та його використання для наближеного обчислення визначеного інтеграла

```
      PROGRAM INTEGRATION1
      REAL S,F,INTEGRAL1
      EXTERNAL F
      S=INTERGRAL1(0.0,0.1,150,F)
      PRINT*,'S=',S
      PAUSE 'PRESS ENTER TO EXIT'
      END
      REAL FUNCTION F(X)
      REAL X
      F=X**3
      END
      REAL FUNCTION INTEGRAL1(A,B,N,F)
      REAL S,DX,X,A,B,F
      INTEGER N,K
      DX=(B-A)/N
      K=0
      S=0
      X=A
100   S=S+F(X)
      K=K+1
      IF (K.LT.N) THEN
```

```
      X=X+DX
      GOTO 100
      END IF
      INTEGRAL1=S*DX
      END
```

Як представлено на лістингу 2.9, в мові програмування FORTRAN забезпечення циклічного виконання послідовностей інструкцій здійснюється так:

```
      ...
      DO label name = start,finish,step
      J=name
      ...
label CONTINUE
      ...
```

Послідовність інструкцій, яка має виконуватися в циклі, розпочинається одразу ж після інструкції `DO` та завершується інструкцією, на яку посилається мітка `label` інструкції `DO`. В інструкції DO також визначається ім'я name змінної-лічильника кількості циклів, її початкове значення `start`, кінцеве значення finish та крок `step` зміни. Для наочного розуміння тексту програми мітка `label` може посилатися на спеціальну інструкцію `CONTINUE`, яка не виконується і введена лише щоб визначати місця для передачі управління. Змінна циклу може бути цілою `(INTEGER)` або дійсною `(REAL)`; якщо крок не вказано, то він за замовченням буде дорівнювати одиниці.

Підпрограма-функція обчислення інтеграла (2.8) за наближеною формулою (2.9), (2.10) буде мати суттєво коротший вигляд завдяки непрямому обчисленню кількості циклів, як показано в лістингу 2.10, через відсутню необхідність безпосереднього підраховування поточного значення змінної-лічильника та перевірки умови досягнення її граничного значення. В схемах алгоритмів також існує спеціальне скорочене позначення циклічного виконання послідовностей інструкцій (рис. 2.12). В той же час слід пам'ятати, що насправді інструкції циклів реалізують схему, що наведена вище на рис. 2.10, але усі «стандартні» дії щодо підрахування лічильника та перевірки його значення реалізуються автоматично. При

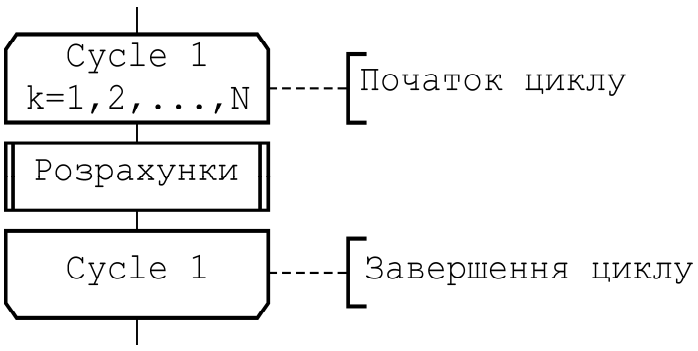


Рисунок 2.12 – Умовне позначення циклів на схемах алгоритмів

використанні інструкцій циклічного повторення з непрямим підрахуванням кількості циклів у послідовності циклічно виконуваних інструкцій можна використовувати значення змінної лічильника циклу, але значення цієї змінної змінювати заборонено.

Лістинг 2.10 – Непрямий лічильник циклів та його використання для наближеного обчислення визначеного інтеграла

```
      REAL FUNCTION INTEGRAL1(A,B,N,F)
      REAL S,DX,X,A,B,F
      INTEGER N,K
      DX=(B-A)/N
      S=0
      X=A
      DO 100 K=1,N
      S=S+F(X)
      X=X+DX
100   CONTINUE
      INTEGRAL1=S*DX
      END
```

2.4.3 Структурні інструкції керування для організації циклів. При створенні великих програм було підмічено, що велика кількість інструкції переходів суттєво ускладнює розуміння алгоритму програми із її тексту. Як альтернативу широкого використання міткам пропонується структурний підхід щодо розробки програм, відповідно до якого програма представляється послідовністю структур керування [20–22]. Стандартна мова програмування FORTRAN 77 є не дуже пристосованою до структурного програмування, тому що концепції структурного програмування були розвинені пізніше, ніж був створений стандарт мови програмування FORTRAN 77, який є результатом узагальнення досвіду багаторічного використання попередніх стандартів мови FORTRAN. Фірма WATCOM внесла додаткові можливості (розширення) в компілятор FORTRAN 77, які не передбачені стандартом цієї мови та дозволяють використовувати певні концепції структурного програмування, спрямовані насамперед на додаткові можливості організації циклів [16].

Розширенням стандартної мови програмування FORTRAN 77 є інструкція `END DO`, яку можна використовувати [16] замість стандартної інструкції `CONTINUE` для організації циклів без використання міток (табл. 2.2). Зазначимо, що інструкція `END DO` передбачена в стандарті FORTRAN 90 та у більш пізніх стандартах мови програмування FORTRAN, тому на цю інструкцію слід звернути особливу увагу [7, 8, 30].

Таблиця 2.2 – Порівняння стандартного та стурктурного DO-циклу

Приклад стандартного DO-циклу	Приклад структурного DO-циклу

`      DO 100 K=1,N`
`      S=S+F(X)`
`      X=X+DX`
`100   CONTINUE`
	`      DO K=1,N`
	`      S=S+F(X)`
	`      X=X+DX`
	`      END DO`

Разом із тим, окрім DO-циклу розширені можливості WATCOM FORTRAN 77 дозволяють реалізувати інші структури управління, серед яких доцільно згадати структури `WHILE – DO – END WHILE` та структури `LOOP – UNTIL`, які сьогодні є обов'язково передбаченими у сучасних мовах програмування [16]. Доведено, що використання структур керування типу `IF – THEN – ELSE`, а також `DO – END DO`, `WHILE – DO – END WHILE` та `LOOP – UNTIL` дозволяє написати будь-яку програму взагалі без використання міток та інструкцій переходу [20, 22].

Структура керування `WHILE – DO – END WHILE` дозволяє здійснювати повторення заданої послідовності інструкцій до тих пір, поки виконується деяка задана умова, та записується наступним чином [16]:

```
...
WHILE(logical expression)DO
...
END WHILE
...
```

В якості `logical expression` має виступати вираз, результат якого має тип `LOGICAL`, або змінна означеного типу, аналогічно тому, як це було розглянуто при вивченні інструкції `IF` на попередніх заняттях. Послідовність інструкцій, яка має виконуватися поки виконується умова, яка представлена у вигляді `logical expression`, має визначатися між інструкцією `WHILE` та інструкцією `END WHILE`.

Структура керування `LOOP – UNTIL` дозволяє здійснювати повторення заданої послідовності інструкцій до тих пір, поки не виконається деяка задана умова, та записується наступним чином [16]:

```
...
LOOP
...
UNTIL(logical expression)
...
```

В якості `logical expression` має виступати вираз, результат якого має тип `LOGICAL`, або змінна означеного типу, аналогічно тому, як

це було розглянуто при вивченні інструкції `IF` на попередніх заняттях. Послідовність інструкцій, яка має виконуватися до тих пір, поки не виконається умова, яка представлена у вигляді `logical expression`, має визначатися між інструкцією `LOOP` та інструкцією `UNTIL`.

В якості прикладів використання розширених інструкцій структур керування `WHILE – DO – END WHILE` та `LOOP – UNTIL` розглянемо цикл обчислення суми при наближеному обчисленні визначеного інтеграла, що використовувався вище в табл. 2.2; в табл. 2.3 наведені реалізації означеного циклу з використанням розширених інструкцій структур керування.

Таблиця 2.3 – Структурні реалізації циклу (розширені)

Приклад циклу WHILE	Приклад циклу UNTIL

`WHILE (X.LT.B) DO`
`S=S+F(X)`
`X=X+DX`
`END WHILE`
	`LOOP`
	`S=S+F(X)`
	`X=X+DX`
	`UNTIL (X.GE.B)`

Слід зазначити, що використання перелічених розширень мови програмування FORTRAN 77 можливе тільки за умов використання компілятора фірми WATCOM, тобто при спробі компіляції текстів програм із таким розширенням на інших компіляторах, наприклад фірм IBM або Microsoft, будуть повідомлення про помилки. Відсутність структур керування в стандартному FORTRAN 77 є його суттєвим недоліком, який сприяв подальшому розвитку мови програмування FORTRAN, сучасні стандарти якої містять усі притаманні сучасним мовам програмування можливості [7, 8, 30]. В той же час, необхідність удосконалення великої кількості програм, що написаних раніше на FORTRAN 77, змушує вивчати FORTRAN 77 і сьогодні. Більш того, FORTRAN 77 містить в собі узагальнення досвіду програмування з початку використання комп’ютерів [30] і тому є зручним та корисним для початкового навчання з програмування.

3 СТРУКТУРИ ДАНИХ

Взагалі дані можна уявляти собі як сукупність значень змінних різних заданих типів. У багатьох випадках дані являють собою характеристики певного об'єкта або певної ситуації. Для опису складних об'єктів (ситуацій) буває необхідно використовувати дуже велику кількість змінних різних типів, для чого бажано мати зручний спосіб представлення інформації про такі складні об'єкти у програмі. Структури даних – це особливий тип даних, який складається із великих обсягів даних одного типу або різних типів та надає можливості доступу до цих даних; структури даних складаються, звичайно, із попередньо введених простих типів даних. Завдяки використанню структур даних, інформацію про складний об'єкт можна представляти однією змінною, – структурою даних, – та мати доступ до усіх змінних, що визначають цей об'єкт.

3.1 Масиви

Простішою структурою даних є масив, який широко використовується в програмах різного призначення, особливо для наукових та інженерних розрахунків, та передбачений у різних виглядах в різних мовах програмування.

3.1.1 Загальні поняття про масиви та доцільність їх використання. Масив являє собою множину елементів одного типу, які об'єднані одним іменем таким чином, що кожному елементу масиву ставиться у відповідність ціле число або декілька упорядкованих цілих чисел. Доступ до елементу масиву здійснюється шляхом визначення імені масиву та індексу (індексів) його елементу.

Використання масивів надає можливості доступу до великої кількості даних через одну змінну, що буває зручним, а іноді просто необхідним, в деяких ситуаціях. В якості прикладу такої ситуації можна запропонувати розв'язування лінійних алгебраїчних рівнянь. Так, лінійне рівняння має вигляд:

$$ax = b, \tag{3.1}$$

де a, b – задані числові параметри рівняння та x – шукана невідома.

Запис одного рівняння (3.1) є дуже зрозумілим, але спроба розглянути систему вже двох лінійних рівнянь приводить до доцільності використання індексів для визначення числових значень параметрів рівнянь:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 &= b_1, \\ a_{21}x_1 + a_{22}x_2 &= b_2, \end{aligned} \tag{3.2}$$

де a_{11}, a_{12}, a_{21}, a_{22}, b_1, b_2 – задані числові параметри системи рівнянь та x_1, x_2 – шукані невідомі.

Систему рівнянь (3.2) можна записати і без індексних позначень із використанням необхідної кількості літер латинського алфавіту, яких в даному випадку буде потрібно вісім. Для систем з великою кількістю n лінійних алгебраїчних рівнянь, що є типовими для наукових та інженерних розрахунків, наприклад при визначенні температурного стану ядерного палива [31–34] (додаток Г), кількості літер алфавіту не вистачатиме для позначення усіх числових параметрів та невідомих і використання індексів є принциповим:

$$\begin{aligned} &a_{11}x_1 + a_{12}x_2 + \ldots + a_{1j}x_j + \ldots a_{1n}x_n = b_1, \\ &a_{21}x_1 + a_{22}x_2 + \ldots + a_{2j}x_j + \ldots a_{2n}x_n = b_2, \\ &\ldots \\ &a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nj}x_j + \ldots a_{nn}x_n = b_n. \end{aligned} \tag{3.3}$$

де позначення числових параметрів системи та шуканих невідомих аналогічно простішій системі (3.2).

Інформацію про числові параметри системи лінійних алгебраїчних рівнянь (3.3) зручно представити у вигляді таблиці **A** та колонки **b**, а інформацію про шукані невідомі – відповідно у вигляді колонки **x**:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}. \tag{3.4}$$

Таблицю чисел вигляду **A** в математиці називають матрицею, а колонки вигляду **b** та **x** – векторами ([35, 36], додаток Д). Доступ до кожного елементу векторів **b** та **x** можна здійснити за допомогою одного індексу, що вказує номер відповідного елементу, тобто сукупність елементів векторів можна визначити за допомогою цілочислового індексу, наприклад, індексу i, який має змінюватися від одиниці до n, що можна представити як відображення множини цілих чисел $i \in \{1,2,\ldots,n\}$ на множину значень елементів векторів **b** та **x**:

$$i \to b_i, \quad i \to x_i, \quad i = 1,2,\ldots n. \tag{3.5}$$

Доступ до кожного елементу матриці **A** можна здійснювати за допомогою упорядкованої пари цілочислових індексів, що визначають номер рядка та номер стовпця, наприклад індексів i та j, які мають змінюватися від одиниці до n, що можна представити у вигляді відображення множини упорядкованої пари $i \times j$ цілих чисел $i \in \{1,2,\ldots,n\}$ та $j \in \{1,2,\ldots,n\}$ на множину значень елементів матриці **A**:

$$i \times j \to a_{ij}, \quad i,j = 1,2,\ldots n. \tag{3.6}$$

Вимога до упорядкованості пари $i \times j$ індексів необхідна, щоб відрізняти елементи a_{ij} та a_{ji}.

Формальне визначення масиву в програмуванні здійснюється як відображення упорядкованої множини цілих чисел на множину значень заданого типу [17, 20]:

$$(i_1 \in I_1) \times (i_2 \in I_2) \times \ldots \times (i_N \in I_N) \to a_{i_1 i_2 \ldots i_N} \in D_a, \tag{3.7}$$

де i_1, i_2,..., i_N – цілочислові значення та I_1, I_2,..., I_N – області визначення цих значень; $a_{i_1 i_2 \ldots i_N}$ – елементи масиву та D_a – область їхнього визначення.

Цілочислові значення i_1, i_2,..., i_N, що визначають елемент масиву, називають індексами. Кожен індекс визначає окремий вимір масиву; кількість елементів масиву в деякому вимірі називають довжиною масиву в цьому вимірі. З точки зору визначення (3.7) вектори **b** та **x** із (3.4), (3.5) є масивами дійсних чисел із одновимірним масивом ($N = 1$) із довжиною 2, а матриця **A** із (3.4), (3.6) є двовимірним масивом дійсних чисел ($N = 2$) із довжиною 2 в кожному із вимірів. Зрозуміло, але слід підкреслити, що масив, як і будь-яка інша структура даних, складається із попередньо введених типів даних – цілочислових значень індексів та типу, що визначає область D_a визначення елементів масиву.

3.1.2 Завдання масивів та доступ до їхніх елементів. Для завдання масиву необхідно визначити тип його елементів, а також кількість вимірів та довжину в кожному із цих вимірів, що робиться відповідним чином в мовах програмування. В мові програмування FORTRAN це робиться наступним чином [6, 9, 29]:

```
type name(i1min:i1max,i2min:i2max,...,inmin:inmax)
```

де `type` – це визначення типу елементів масиву; `name` – ім'я масиву; `i1min` та `i1max` – мінімальне та максимальне значення індексу в пер-

шому вимірі масиву; `i2min` та `i2max` – мінімальне та максимальне значення індексу в другому вимірі масиву і т.д. до `inmin` та `inmax` – мінімальне та максимальне значення індексу у n-му вимірі масиву.

Кількість вимірів масиву визначається кількістю пар мінімальних і максимальних значень індексів в опису масиву, тобто є на одиницю більшою, ніж кількість ком в дужках опису масиву. Якщо мінімальне значення індексу в деякому напрямку дорівнює одиниці, то його можна не вказувати, а вказувати лише максимальне значення індексу без двокрапки. Так, одновимірні масиви X та B дійсних чисел завдовжки 20 елементів, а також двовимірний масив A завдовжки 20 елементів в кожному із вимірів можуть бути визначені наступним чином:

```
CCC   ARRAYS DEFINITIONS
      REAL X(0:19)
      REAL B(20)
      REAL A(20,20)
```

Масиви з елементами однакового типу можна визначати в одному рядку, що економить розмір тексту програми:

```
CCC   ARRAYS DEFINITIONS
      REAL X(0:19), B(20), A(20,20)
```

Доступ до елементів масивів здійснюється шляхом вказування імені масиву та індексів необхідного елементу в круглих дужках через коми; посилаючись таким чином на елемент масиву, його можна використовувати як звичайну змінну у арифметичних виразах:

```
...
      REAL X(0:19), B(20), A(20,20)
...
      X(0)=25.0
      A(1,1)=3.5
```

Підкреслимо, що елементи масивів можуть мати не тільки дійсний тип, а й будь-який стандартний (цілочисловий та логічний) тип, що є у FORTRAN.

Як приклад розглянемо програму (ліст. 3.1) розв'язування системи двох лінійних алгебраїчних рівнянь (3.2). Ця програма має розпочинатися із введення матриці та вектора правої частини системи лінійних алгебраїчних рівнянь, що можна легко здійснити за допомогою циклів `DO – END DO`; в програмі передбачається послідовне введення коефіцієнтів матриці та вектора правої частини для рівнянь системи. Розв'язування системи пропонується здійснити методом Крамера ([35], додаток Д). Для цього визначимо визначники:

$$\Delta = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}, \tag{3.8}$$

$$\Delta_1 = \begin{vmatrix} b_1 & a_{12} \\ b_2 & a_{22} \end{vmatrix} = b_1a_{22} - a_{12}b_2, \tag{3.9}$$

$$\Delta_2 = \begin{vmatrix} a_{11} & b_1 \\ a_{21} & b_2 \end{vmatrix} = a_{11}b_2 - b_1a_{21}. \tag{3.10}$$

Розв'язок одержимо за допомогою детермінантів (3.8)–(3.10) ([35], додаток Д):

$$x_1 = \frac{\Delta_1}{\Delta}, \quad x_2 = \frac{\Delta_2}{\Delta}. \tag{3.11}$$

Наприкінці програма (ліст. 3.1) представляє результати розрахунків.

Лістинг 3.1 – Розв'язування системи двох лінійних алгебраїчних рівнянь

```
      PROGRAM LAES2
      REAL A(2,2),B(2),X(2),DETA,DETA1,DETA2
      INTEGER I,J
      DO I=1,2
      DO J=1,2
      PRINT '(''A('',I1,'','',I1,'')=''$)',I,J
      READ *,A(I,J)
      END DO
      PRINT '(''B('',I1,'')=''$)',I
      READ *,B(I)
      END DO
      DETA=A(1,1)*A(2,2)-A(2,1)*A(1,2)
      DETA1=B(1)*A(2,2)-B(2)*A(1,2)
      DETA2=A(1,1)*B(2)-A(2,1)*B(1)
      X(1)=DETA1/DETA
      X(2)=DETA2/DETA
CCCCCC RESULTS OUTPUTTING
      DO I=1,2
      DO J=1,2
      PRINT '(F10.4,''X'',I1,\)',A(I,J),J
      IF (J.LT.2) PRINT'(''+''\)'
      END DO
      PRINT '(''='',F10.4)',B(I)
      END DO
```

```
DO I=1,2
PRINT '(''X('',I1,'')='',F10.4)',I,X(I)
END DO
PAUSE 'PRESS ENTER TO EXIT PROGRAM'
END
```

3.1.3 Використання масивів у підпрограмах. Масиви можна передавати в якості параметрів до підпрограм, що широко використовується в програмуванні. Якщо до підпрограми передається масив, то в цій підпрограмі має обов'язково міститись опис цього масиву, який має бути цілком відповідним до опису в програмній компоненті, що викликала підпрограму. Наведена вище на ліст. 3.1 програма може бути представлена у вигляді 4-х підпрограм: введення матриці та вектора системи рівнянь, розв'язку системи рівнянь, друку системи рівнянь та друку вектора:

Лістинг 3.2 – Підпрограми для системи двох лінійних алгебраїчних рівнянь

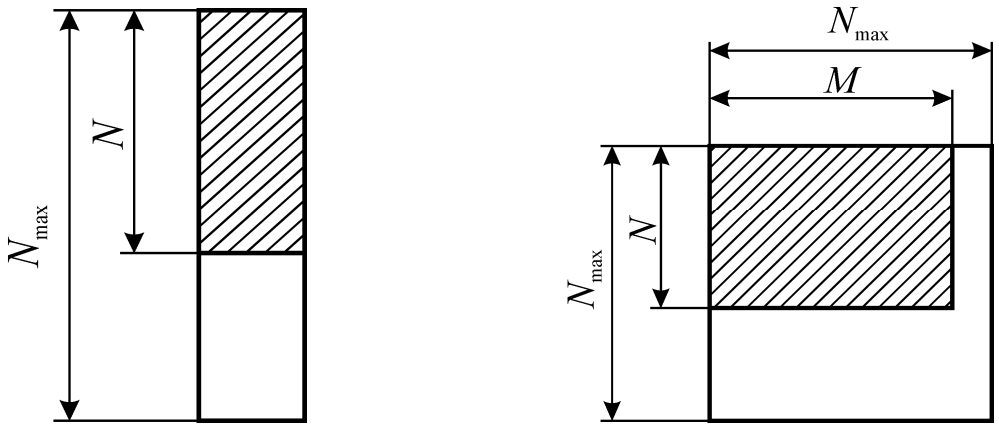
```
PROGRAM LAES2
REAL A(2,2),B(2),X(2)
CALL INPUTLAES(A,B)
CALL PRINTLAES(A,B)
CALL SOLVELAES(A,B,X)
CALL PRINTVECTOR(X)
PAUSE 'PRESS ENTER TO EXIT PROGRAM'
END
SUBROUTINE INPUTLAES(A,B)
REAL A(2,2),B(2)
INTEGER I,J
DO I=1,2
DO J=1,2
PRINT '(''A('',I1,'','',I1,'')='',$)',I,J
READ *,A(I,J)
END DO
PRINT '(''B('',I1,'')='',$)',I
READ *,B(I)
END DO
END
SUBROUTINE PRINTLAES(A,B)
REAL A(2,2),B(2)
INTEGER I,J
DO I=1,2
DO J=1,2
PRINT '(F10.4,''X'',I1,$)',A(I,J),J
```

```
IF (J.LT.2) PRINT'(''+''$)'
END DO
PRINT '(''=''',F10.4)',B(I)
END DO
END
SUBROUTINE SOLVELAES(A,B,X)
REAL A(2,2),B(2),X(2),D,D1,D2
D=A(1,1)*A(2,2)-A(2,1)*A(1,2)
D1=B(1)*A(2,2)-B(2)*A(1,2)
D2=A(1,1)*B(2)-A(2,1)*B(1)
X(1)=D1/D
X(2)=D2/D
END
SUBROUTINE PRINTVECTOR(X)
REAL X(2)
INTEGER I
DO I=1,2
PRINT '(''X(''',I1,''')=''',F10.4)',I,X(I)
END DO
END
```

3.2 Масиви змінної довжини та використання структур даних

Дуже часто може виникнути ситуація, коли довжини масиву мають визначатися в програмі у залежності від поточного стану певних змінних та навіть змінюватися протягом роботи програми. Розглянемо далі деякі базові питання щодо прийомів програмування масивів змінної довжини, яка визначається та може змінюватися протягом роботи програми.

3.2.1 Свідомо великі довжини масивів. Найпростішим, але нераціональним з точки зору використання пам'яті, способом роботи із масивами змінної довжини є створення масиву, довжина $N_{\max}$ якого заздалегідь перевищує довжини масивів, що будуть використовуватися у програмі, та визначення розмірів масиву в кожному окремому випадку. Реалізацію такого простішого підходу на прикладі одновимірного масиву змінної довжини N та двовимірного масиву змінної довжини $N \times M$ можна наочно уявляти, як це показано на рис. 3.1. Звісно, що при цьому пам'ять комп'ютера використовується неефективно, оскільки для розміщення масивів в деяких випадках може виділятися суттєво більше пам'яті, ніж це потрібно, і ця виділена пам'ять не може бути використана.



а) *б)*

а) одновимірний масив (вектор); *б) двовимірний масив (матриця);*

Рисунок 3.1 – Використання частини масиву в якості масиву

Відсоток M_{NU} невикористаної пам'яті при використанні масивів свідомо більшої довжини для забезпечення роботи із масивами змінної довжини для одновимірного та двовимірного масивів визначається відповідно так:

$$M_{NU} = \left(1 - \frac{N}{N_{\max}}\right) \cdot 100\%, \tag{3.12}$$

$$M_{NU} = \left(1 - \frac{N \cdot M}{N_{\max} \cdot N_{\max}}\right) \cdot 100\%. \tag{3.13}$$

Звісно, що використання масивів свідомо більшої довжини в якості масивів змінної довжини в більшості випадків веде до нераціонального використання пам'яті комп'ютера. В той же час, простота програмної реалізації робить такий підхід дуже привабливим, і його використання єцілком виправданим для випадків, коли M_{NU} є достатньо низьким, що для одновимірного масиву буде виконуватися, коли $N \sim N_{\max}$, а для одновимірного – коли $N \sim N_{\max}$ та $M \sim N_{\max}$. В деяких програмах існує можливість обрати $N_{\max}$ таким чином, щоб M_{NU} була достатньо малою, і завдяки простій програмній реалізації використання масивів свідомо більшої довжини в якості масивів змінної довжини в таких програмах є цілком виправданим.

Розглянемо далі приклад програми (ліст. 3.3), в якій здійснюється введення та друк довжин та елементів двовимірного масиву змінної довжини. Величину $N_{\max}$ зручно ввести із використанням інструкції `PARAMETER`, що дозволяє шляхом однократної зміни розширювати або зменшувати максимальну границю масивів [6, 9, 29]. Якщо деяка вели-

чина введена за допомогою інструкції PARAMETER, то її заборонено змінювати в програмі; в цій інструкції також можна вводити інші значення простих типів, не обов'язково границі довжин масивів. Значення, що введені за допомогою цієї інструкції PARAMETER, визначені виключно у відповідній програмній компоненті.

Лістинг 3.3 – Приклад використання масивів свідомо більшої довжини в якості масивів змінної довжини

```
      PROGRAM ARRAY3
      PARAMETER (NMAX=50)
      REAL M(NMAX,NMAX)
      INTEGER MN,MM
      CALL INPUTMATRIX(M,MN,MM,NMAX)
      CALL PRINTMATRIX(M,MN,MM,NMAX)
      PAUSE 'PRESS ENTER TO EXIT PROGRAM'
      END
      SUBROUTINE INPUTMATRIX(A,N,M,SIZE)
      INTEGER N,M,SIZE,I,J
      REAL A(SIZE,SIZE)
      PRINT '(''INPUT COUNT ROWS OF THE MATRIX: '',$)'
      READ *,N
      PRINT '(''INPUT COUNT COLS OF THE MATRIX: '',$)'
      READ *,M
      DO I=1,N
      PRINT '(''INPUT ROW '',I2,'':'')',I
      DO J=1,M
      PRINT
     1'(''INPUT ELEMENT('',I2,'','',I2,'')='',$)',I,J
      READ *,A(I,J)
      END DO
      END DO
      END
      SUBROUTINE PRINTMATRIX(A,N,M,SIZE)
      INTEGER N,M,SIZE,I,J
      REAL A(SIZE,SIZE)
      PRINT '(''COUNT ROWS OF MATRIX, ROWS='',I2)',N
      PRINT '(''COUNT COLS OF MATRIX, COLS='',I2)',M
      DO I=1,N
      PRINT '(''ROW = '',I2)',I
      DO J=1,M
      PRINT '(''ELEMENT('',I2,'','',I2,'')='',F10.4)'
     1,I,J,A(I,J)
```

```
      END DO
      END DO
      END
```

При обробці масивів у підпрограмах (див. ліст. 3.3) до цих підпрограм слід окрім імен масивів передавати значення їхніх довжин в кожному із вимірів, а також парметра, що визначає максимальний розмір масиву.

3.2.2 Точне виділення пам'яті. Звичайно, що не у всіх програмах можна знайти одне значення $N_{\max}$ таким чином, щоб M_{NU} була достатньо малою для усіх масивів змінної довжини, що будуть використовуватися у програмі. В цьому випадку зручним є використання масивів, довжини яких визначаються у програмі. В цьому випадку на початку програми при визначенні масиву слід визначити лише ім'я відповідної йому змінної та кількість вимірів за допомогою символів «:» через кому у круглих дужках одразу ж після імені змінної [16, 30]. Далі у програмі слід за допомогою інструкції ALLOCATE виділяти пам'ять для розміщення цього масиву відповідно до необхідних довжин у кожному із вимірювань та працювати як зі звичайним масивом [16, 30]. Після завершення роботи із таким масивом слід за допомогою інструкції DEALLOCATE звільнити пам'ять, що була необхідна для розташування масиву, і далі можна знову виділяти пам'ять на цей масив відповідно до нових потрібних довжин у кожному із напрямків [16, 30]. Для одновимірного та двовимірного масиву невизначеної довжини визначення, виділення пам'яті, робота та звільнення пам'яті може бути здійснено наступним чином:

```
      REAL B(:),A(:,:)
...
      ALLOCATE(B(5))
      ALLOCATE(A(2,7))
...
      B(1)=1.5
      A(2,5)=B(1)
...
      DEALLOCATE (A,B)
```

Тоді розглянуту програму (див. ліст. 3.3) можна реалізувати іншим чином:

Лістинг 3.4 – Приклад використання масивів невизначеної довжини

```
PROGRAM ARRAY4
REAL M(:,:)
INTEGER MN,MM
CALL IMPUTMATRIXSIZES(MN,MM)
```

```
 ALLOCATE(M(MN,MM))
 CALL INPUTMATRIXELEM(M,MN,MM)
 CALL PRINTMATRIX(M,MN,MM)
 DEALLOCATE(M)
 PAUSE 'PRESS ENTER TO EXIT PROGRAM'
 END
 SUBROUTINE IMPUTMATRIXSIZES(N,M)
 INTEGER N,M
 PRINT '(''INPUT COUNT ROWS (N) OF MATRIX: '',\)'
 READ *,N
 PRINT '(''INPUT COUNT COLS (M) OF MATRIX: '',\)'
 READ *,M
 END
 SUBROUTINE INPUTMATRIXELEM(A,N,M)
 INTEGER N,M,I,J
 REAL A(N,M)
 DO I=1,N
 PRINT '(''INPUT ROW '',I2)',I
 DO J=1,M
 PRINT
1'(''INPUT ELEMENT ('',I2,'','',I2,'')='',\)',I,J
 READ *,A(I,J)
 END DO
 END DO
 END
 SUBROUTINE PRINTMATRIX(A,N,M)
 INTEGER N,M,SIZE,I,J
 REAL A(N,M)
 PRINT '(''COUNT COLS OF MATRIX, N='',I2)',N
 PRINT '(''COUNT COLS OF MATRIX, M='',I2)',M
 DO I=1,N
 PRINT '(''ROW = '',I2)',I
 DO J=1,M
 PRINT
1'(''MATRIX('',I2,'','',I2,'')='',F10.4)'
2,I,J,A(I,J)
 END DO
 END DO
 END
```

При використанні масивів невизначеної довжини в підпрограмах в якості вхідних параметрів підпрограм слід передавати довжини масиву у всіх його вимірах, але в самій підпрограмі визначати таки масиви як

масиви із заданими довжинами, що відповідають відповідним вхідним параметрам підпрограми. У порівнянні із технологією програмування масивів змінної довжини на основі використання масивів із завідомо більшою довжиною, що розглядалося вище в п. 3.2.1, використання масивів невизначеної довжини є більш ефективним з точки зору використання пам'яті комп'ютера, але потребує від програміста пам'ятати про необхідність виділення та звільнення пам'яті у необхідних місцях програми. Слід зазначити, що простота програмування масивів є однією із переваг мови програмування FORTRAN перед іншими мовами програмування, але ця перевага є принциповою тільки при розробці програм матричних обчислень.

3.2.3 Використання структур даних. При роботі із масивами необхідно мати інформацію щодо довжин масивів в усіх вимірах, для чого, зазвичай, передбачають відповідну кількість змінних. Якщо в програмі використовується велика кількість масивів різної довжини, то іноді можна заплутатися в іменах змінних, що визначають довжини цих масивів. Шляхом спеціального підходу щодо вибору імен змінних можна уникнути плутанини щодо відповідності між іменами змінних та іменами масивів, довжину яких ці змінні визначають. В той же час, відповідно до концепції структурного програмування для визначення масивів бажано мати таку структуру даних, в якій визначені довжини масиву та його елементи, що дозволило б повністю представляти масиви у вигляді однієї змінної. Можливості щодо використання такого підходу надають розширення стандартної мови програмування FORTRAN 77, передбачені в компіляторі `WATCOM`, у вигляді спеціальних нестандартних інструкцій `STRUCTURE` та `RECORD`, які дозволяють створювати та використовувати в програмах складні структури [16], які містять в собі дані різного типу.

Нестандартна інструкція `STRUCTURE` створює новий тип даних, але не змінну, який може містити визначення змінних різного типу, кожна із яких називається полем структури та буде локально визначеною виключно усередині структури, в якій вона визначена; інструкція `END STRUCTURE` завершує визначення нового типу даних. Інструкція `RECORD` дозволяє створити змінну, що має нестандартний тип, визначений у вигляді структури даних за допомогою інструкції `STRUCTURE`. Доступ до полів структури даних здійснюється шляхом вказування імені змінної структури даних, символу «%» або «.», а також імені поля структури даних. Типові використання інструкцій роботи зі структурами даних показані нижче:

```
...
      STRUCTURE /name/
...
      INTEGER I,J
```

```
...
      END STRUCTURE
...
      RECORD /name/ varname
...
      varname.I=5
...
      varname%J=varname%I
...
```

При використанні структури даних для визначення двовимірного масиву до полів цієї структури даних доцільно ввести довжини масиву в кожному із його вимірів, а також масив із необхідною кількістю вимірів для зберігання елементів. Це дозволить в одній змінній, що має відповідний структурний тип, зберігати усі дані, які повністю визначають масив. Приклад можливого використання спеціально створених структур даних для зберігання двовимірного масиву – матриці є наступним:

Лістинг 3.5 – Використання структур даних для визначення матриці

```
PROGRAM ARRAY5
PARAMETER (NMAX=50)
STRUCTURE /MATRIX/
INTEGER ROWS,COLS
REAL ELEMENT(NMAX,NMAX)
END STRUCTURE
RECORD /MATRIX/ M
CALL INPUTMATRIX(M%ELEMENT,M%ROWS,M%COLS,NMAX)
CALL PRINTMATRIX(M.ELEMENT,M.ROWS,M.COLS,NMAX)
PAUSE 'PRESS ENTER TO EXIT PROGRAM'
END
```

Підпрограми, що використовуються в програмі, яка наведена на ліст. 3.5, є точно такими, як у програмі, що наведена вище на ліст. 3.3. Зрозуміло, що для забезпечення можливостей зміни розмірів масиву в програмі, що наведена на ліст. 3.5, використовується розглянутий в п. 3.2.1 прийом програмування, заснований на використанні масивів із завідомо більшою довжиною. На жаль, масиви невизначеної довжини, розглянуті в п. 3.2.2, не можуть бути використані в якості полів структур інструкції `STRUCTURE`, що є суттєвим обмеженням можливостей FORTRAN 77, навіть із розширеннями. В сучасних стандартах мови програмування FORTRAN та в інших сучасних мовах програмування передбачається можливість створення структур даних, що містять у тому числі й масиви невизначеної довжини, аналогічні розглянутим вище в п. 3.2.2, а також навіть і підпрограми-процедури та підпрограми-функції. Об'єднання в одному складеному типі

і даних, і підпрограм є центральною ідеєю об'єктно-орієнтованого програмування, яке є однією із сучасних технологій програмування, що дозволяє суттєво прискорювати розробку програм та їхнє супроводження протягом використання.

3.3 Типові прийоми програмування щодо обробки масивів

Розв'язування багатьох наукових та технічних задач зводиться до відповідної обробки спеціально побудованих масивів. Прикладами є обчислення визначника та рангу матриці [35, 36] (додаток Д), що необхідно при дослідженні питань щодо керованості систем звичайних диференціальних рівнянь ([37–41] (додаток Е). Розглянемо далі типові задачі щодо обробки двовимірного масиву розміром 3×4 наступного вигляду:

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & 1 & 1 & 1 \\ 2 & 1 & 1 & 2 & 3 \\ 4 & 2 & 2 & 3 & 4 \\ 2 & 1 & 1 & 1 & 1 \end{pmatrix}. \qquad (3.14)$$

В лістингу 3.7 наведено текст програми, в якій передбачається ініціалізація даних (3.14) двовимірного масиву за допомогою інструкції `DATA` та запропоновано підпрограму виведення двовимірного масиву на екран у зручному вигляді таблиці рядків та стовпчиків. Інструкція `DATA` дозволяє у зручній формі здійснити присвоєння значень та має вигляд:

`DATA` ***`vlist`***`/`***`clist`***`/`, (3.15)

де ***`vlist`*** – список змінних, значення яких мають бути присвоєні; *clist* – список констант, що відповідають значенням змінних із попереднього списку ***`vlist`***.

Зрозуміло, що значення змінних можуть бути визначеними шляхом безпосереднього присвоєння, але використання інструкції (3.15) просто додає додаткові зручності щодо тексту програми (ліст. 3.6).

Лістинг 3.6 – Ініціалізація значень та виведення масиву на екран

```
PROGRAM RANK
INTEGER N,M
PARAMETER (N=4,M=5)
REAL A(N,M)
DATA A/2*2.0,4.0,2.0,2*1.0,2.0,1.0,2*1.0,2.0,1.0
```

```
     1,1.0,2.0,3.0,1.0,1.0,3.0,4.0,1.0/
      CALL PRINTMATRIX(N,M,A)
      PAUSE 'PRESS ENTER TO EXIT'
      END
CCCCCC OUTPUTTING THE MATRIX TO THE SCREEN
      SUBROUTINE PRINTMATRIX(N,M,A)
      INTEGER N,M
      REAL A(N,M)
      PRINT '(A)','---------- MATRIX DATA :'
      PRINT 100,'COUNT ROWS=',N
      PRINT 100,'COUNT COLS=',M
      PRINT '(A)','..ELEMENTS='
      DO I=1,N
      DO J=1,M
      IF (J.LT.M) THEN
      PRINT '(1X,E12.5,\)',A(I,J)
      ELSE
      PRINT '(1X,E12.5)',A(I,J)
      END IF
      END DO
      END DO
      PRINT '(A)','------------------------'
100   FORMAT(A,I2)
      END
```

Слід додатково нагадати, що в наведеній програмі (див. ліст. 3.6) використано інструкцію PARAMETER, яка дозволяє визначати значення параметрів, які, на відміну від змінних, не можуть бути зміненими. Параметри зручно використовувати для визначення розмірів масивів.

3.3.1 Пошук максимального елементу рядка та стовпця. Для визначення максимального значення елементів рядка (стовпця) слід у циклі порівняти між собою значення елементів цього рядка (стовпця), як це зроблено в підпрограмі визначення максимального елементу рядка двовимірного масиву на ліст. 3.7. Підпрограма має отримати дані щодо розмірів масиву, елементів масиву та номера рядка, в якому слід знайти максимальний елемент, щоб в результаті визначити максимальне значення серед елементів заданого рядка та відповідний цьому максимальному значенню номер елементу рядка. У прикладі використання цієї підпрограми (ліст. 3.7) у відповідному циклі здійснюється визначення та виведення на екран монітора результатів щодо максимального елементу усіх рядків двовимірного масиву.

Для засвоєння навичок програмування подібних задач слід самостійно створити підпрограми для визначення максимального елементу стовпця, а також мінімального елементу рядка та стовпця двовимірного масиву.

3.3.2 Перестановка рядків масиву. Розв'язування деяких задач вимагає перестановки рядків двовимірного масиву. Для цього слід передбачити окрему допоміжну змінну, щоб зберегти дані від знищення, як у підпрограмі на ліст. 3.8. Ця підпрограма має отримати дані щодо розмірів масиву, елементів масиву, а також номерів рядків масиву, які слід переставити, і в результаті цього елементи заданих рядків будуть переставлені безпосередньо в заданому масиві. Використання цієї підпрограми показано на ліст. 3.8, де здійснюється виведення вихідного масиву, перестановка 2-го та 4-го рядків масиву та виведення масиву з переставленими рядками.

Для засвоєння навичок програмування подібних задач слід самостійно створити підпрограму перестановки стовпчиків двовимірного масиву.

Лістинг 3.7 – Пошук максимального елементу рядків двовимірного масиву

```
      PROGRAM RANK
      INTEGER N,M,NM
      PARAMETER (N=4,M=5)
      REAL A(N,M),AM
      DATA A/2*2.0,4.0,2.0,2*1.0,2.0,1.0,2*1.0,2.0,1.0
     1,1.0,2.0,3.0,1.0,1.0,3.0,4.0,1.0/
      CALL PRINTMATRIX(N,M,A)
      DO I=1,N
      CALL MAXROW(N,M,A,I,AM,NM)
      PRINT
     1'(''ROW='',I1,2X,''MAX='',E12.5,2X,''NMAX='',I1)'
     2,I,AM,NM
      END DO
      PAUSE 'PRESS ENTER TO EXIT'
      END
CCCCCC FINDING THE MAXIMUM ELEMENT OF THE ROW
      SUBROUTINE MAXROW(N,M,A,ROW,VALUE,NUMBER)
      INTEGER N,M,ROW,NUMBER
      REAL A(N,M),VALUE
      VALUE=A(ROW,1)
      NUMBER=1
      DO I=1,M
      IF (VALUE.LT.A(ROW,I)) THEN
```

```
  VALUE=A(ROW,I)
  NUMBER=I
  END IF
  END DO
  END
```

Лістинг 3.8 – Перестановка рядків двовимірного масиву

```
  PROGRAM RANK
  INTEGER N,M,NM
  PARAMETER (N=4,M=5)
  REAL A(N,M),AM
  DATA A/2*2.0,4.0,2.0,2*1.0,2.0,1.0,2*1.0,2.0,1.0
 1,1.0,2.0,3.0,1.0,1.0,3.0,4.0,1.0/
  CALL PRINTMATRIX(N,M,A)
  CALL PERMUTROWS(N,M,A,4,2)
  CALL PRINTMATRIX(N,M,A)
  PAUSE 'PRESS ENTER TO EXIT'
  END
  SUBROUTINE PERMUTROWS(N,M,A,R1,R2)
  INTEGER N,M,R1,R2
  REAL A(N,M),EL
  DO I=1,M
  EL=A(R1,I)
  A(R1,I)=A(R2,I)
  A(R2,I)=EL
  END DO
  END
```

3.3.3 Тотожні перетворення матриць. Матриці (додаток Д) широко використовуються для формулювання математичних задач, у тому числі проблем керованості (додаток Е). Числові характеристики матриці, такі як визначник або ранг (додаток Д), не змінюються при елементарних перетворюваннях [35, 36], таких як множення рядка на число, додавання до рядка іншого рядка та такі самі перетворення для стовпців. Такі властивості елементарних перетворень надають можливостей перетворювати матриці до спеціального вигляду, наприклад із нульовими елементами, що розташовані під діагональними елементами:

$$\mathbf{A}=\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & \dots & a_{1m} \\ 0 & a_{22} & a_{23} & a_{24} & a_{25} & \dots & a_{2m} \\ 0 & 0 & a_{33} & a_{34} & a_{35} & \dots & a_{3m} \\ 0 & 0 & 0 & a_{44} & a_{45} & \dots & a_{4m} \\ 0 & 0 & 0 & 0 & a_{55} & \dots & a_{5m} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & a_{nm} \end{pmatrix}. \qquad (3.16)$$

Матриці спеціального вигляду, такі, як, наприклад, (3.16), мають зрозумілі властивості, що дозволяють дуже просто визначати їхні характеристики, такі, як, наприклад, визначник та ранг (додаток Д). Перетворення матриці довільного вигляду до вигляду (3.16) дозволяє розв'язувати багато задач, у тому числі обчислення визначника та рангу, що показано на ліст. 3.9. У програмі (ліст. 3.9) використовується підпрограма, що розглянута вище на ліст. 3.6. Перетворення, що необхідні для обчислення рангу матриці, є ідентичними до перетворень, що необхідні для розв'язування системи лінійних алгебраїчних рівнянь, будуть більш докладно розглянуті у подальшому. Зараз рекомендується підготувати програму відповідно до ліст. 3.9 та подивитися результат її виконання, щоб побачити можливості щодо обчислення рангу матриці.

Лістинг 3.9 – Програма, яка здійснює перетворення, що необхідні для визначення рангу матриці

```
 PROGRAM RANK
 PARAMETER (N=4,M=5)
 REAL A(N,M),AM
 INTEGER NM
 DATA A/2*2.0,4.0,2.0,2*1.0,2.0,1.0,2*1.0
1,2.0,1.0,1.0,2.0,3.0,1.0
2,1.0,3.0,4.0,1.0/
 CALL PRINTMATRIX(N,M,A)
 DO I=1,N
 AM=ABS(A(I,I))
 NM=I
 DO J=I,N
 IF (AM.LT.ABS(A(J,I))) THEN
 AM=ABS(A(J,I))
 NM=J
 END IF
 END DO
 IF (NM.NE.I) THEN
```

```
DO J=I,M
AM=A(I,J)
A(I,J)=A(NM,J)
A(NM,J)=AM
END DO
END IF
IF (A(I,I).NE.0.0) THEN
DO J=I+1,N
AM=A(J,I)/A(I,I)
DO K=I,M
A(J,K)=A(J,K)-A(I,K)*AM
END DO
END DO
END IF
END DO
CALL PRINTMATRIX(N,M,A)
PAUSE 'PRESS ENTER TO EXIT'
END
```

3.4 Розв'язування системи лінійних алгебраїчних рівнянь

Багато задач із різних галузей знань може бути зведено до розв'язування систем лінійних алгебраїчних рівнянь, наприклад, визначення температури ядерного палива (додаток Г), напружено-деформованого стану тіл, що складають природні та технічні системи, швидкостей руху рідин та газів у природних та штучних системах, обробка статистичних даних, пошук оптимальних станів систем і багато інших. Простішим методом розв'язування системи лінійних алгебраїчних рівнянь є метод Гауса, який зводиться до перетворень матриць за спеціальними алгоритмами [23, 24, 42, 43]. Програмування методу Гауса при вивченні програмування є цілком виправданим через можливості щодо придбання простіших навичок та початкового досвіду реалізації алгоритмів обробки масивів, які є типовими для програм різного призначення.

3.4.1 Система лінійних алгебраїчних рівнянь та метод Гауса. В загальному вигляді систему лінійних алгебраїчних рівнянь зазвичай представляють у вигляді (3.3) та для скорочення запису використовують матрично-векторну форму (3.4). Для ще більшого скорочення запису систему лінійних алгебраїчних рівнянь представляють за допомогою розширеної матриці [35]:

$$\mathbf{A}_{\mathbf{b}} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{pmatrix}, \tag{3.17}$$

яка має розмір $n \times (n+1)$ та складається із матриці та вектора, що визначає цю систему. Ідея методу Гауса щодо розв'язування системи лінійних алгебраїчних рівнянь полягає у послідовному виключенні невідомих із використанням рівнянь системи: з другого та усіх подальших рівнянь за допомогою першого рівняння виключають першу невідому; далі із третього рівняння та усіх подальших рівнянь за допомогою другого рівняння виключають другу невідому і так далі до останнього рівняння, в якому залишатиметься одна невідома. Останнє рівняння з однією невідомою легко розв'язується, і останні невідомі послідовно визначаються, починаючи із передостанньої до першої.

Цілком зрозуміло, що метод Гауса можна уявляти як так званий «прямий хід» – тотожні перетворення розширеної матриці (3.17) до спеціального вигляду:

$$\mathbf{A}'_{\mathbf{b}} = \begin{pmatrix} 1 & a'_{12} & \dots & a'_{1n} & b'_1 \\ 0 & 1 & \dots & a'_{2n} & b'_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & b'_n \end{pmatrix}. \tag{3.18}$$

Тоді визначення розв'язку відповідної системи рівнянь, що тотожно перетворена до вигляду (3.18), можна здійснити наступним чином:

$$x_i = b'_i - \sum_{j=i+1}^{n} a'_{ij} x_j, \quad i = n, n-1, n-2, \dots, 1. \tag{3.19}$$

Відповідно до формул (3.19) для випадків $i = n, n-1, n-2$ маємо відповідно:

$$x_n = b'_n, \quad x_{n-1} = b'_{n-1} - a'_{n-1n} x_n, \quad x_{n-2} = b'_{n-2} - a'_{n-2n-1} x_{n-1} - a'_{n-2n} x_n. \tag{3.20}$$

Таким чином, програмування методу Гауса здійснюється в два етапи: 1) програмування «прямого ходу» – тотожних перетворень розширеної матриці (3.17) до спеціального вигляду (3.18); 2) програмування «оборотного ходу» – визначення розв'язку за формулами (3.19). У подальшому при

програмуванні методу Гауса систему лінійних алгебраїчних рівнянь представляємо у вигляді матриці та вектора правої частини, які будемо вводити за допомогою клавіатури за допомогою спеціальної підпрограми.

3.4.2 Перетворення прямого ходу методу Гауса. Нехай маємо матрицю та вектор правої частини системи лінійних алгебраїчних рівнянь загального вигляду (3.17). Щоб тотожно перетворити перший рядок розширеної матриці до спеціального вигляду (3.18), необхідно поділити кожний елемент першого рядка на елемент a_{11}, що математично у два кроки:

$$\overline{a}_{1j} = a_{1j},\ j = 1,2,...,n,\ \overline{b}_1 = b_1,\quad a_{1j} = \frac{\overline{a}_{1j}}{\overline{a}_{11}},\ j = 1,2,...,n,\ b_1 = \frac{\overline{b}_1}{\overline{a}_{11}}. \tag{3.21}$$

В результаті виконання перетворення (3.21) матриця (3.17) набуде наступного вигляду:

$$\begin{pmatrix} 1 & \alpha_{12} & \dots & \alpha_{1n} & \beta_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{pmatrix}, \tag{3.22}$$

де $\alpha_{1j} = a_{1j}/a_{11},\ j = 2,3,...,n,\quad \beta_1 = b_1/a_{11}$.

Перший крок (3.21) введення «проміжних» значень $\overline{a}_{1j}$ та $\overline{b}_1$ потрібен виключно з математичної точки зору, оскільки некоректно писати, наприклад, $b_1 = b_1/a_{11}$, оскільки наслідком цього одразу ж маємо $a_{11} = 1$. В той же час, при програмуванні маємо можливість нове значення змінної визначати довільним чином через її поточне значення, тому введення «проміжних» значень, як в (3.21), є непотрібним. При цьому в програмі слід виділити окрему змінну для зберігання a_{11} на початку перетворення (3.21), оскільки перший крок такого перетворення, як ми бачимо із (3.22), одразу ж приведе до $a_{11} = 1$. Програма, що реалізує перетворення (3.23), може мати вигляд:

Лістинг 3.10 – Перетворення рядка розширеної матриці

```
...
REAL A(N,N),B(N),AII
INTEGER J
...
AII=A(1,1)
DO J=1,N
A(1,J)=A(1,J)/AII
END DO
```

```
B(1)=B(1)/AII
...
```

Алгоритм, що наведений у ліст. 3.10, не спрацює у випадку, коли $a_{11}=0$, тому перед тим, як здійснювати цей алгоритм, слід визначити максимальний за модулем елемент першого стовпця матриці та переставити між собою перший рядок та рядок, що містить в собі максимальний за модулем елемент. Такі перетворення є тотожними для системи рівнянь і можуть бути реалізовані, наприклад, як показано далі у ліст. 3.11. Після перестановки рядків відповідно до алгоритму, що реалізований у ліст. 3.11, можемо перетворити перший рядок розширеної матриці до вигляду як у (3.18) відповідно до алгоритму, реалізованому в ліст. 3.10. Далі шляхом множення першого рядка на елемент $a_{j1},\ j=2,3,...,n$ та віднімання результату такого множення із відповідного рядка $j=2,3,...,n$ можемо привести матрицю (3.22) до наступного вигляду:

$$\begin{pmatrix} 1 & \alpha_{12} & \dots & \alpha_{1n} & \beta_1 \\ 0 & \alpha_{22} & \dots & \alpha_{2n} & \beta_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \alpha_{n2} & \dots & \alpha_{nn} & \beta_n \end{pmatrix}, \tag{3.23}$$

де $\alpha_{ij}=a_{ij}-\alpha_{1j}a_{i1},\ \beta_i=b_i-\beta_1 a_{i1},\ \ i,j=2,3,...,n$.

Лістинг 3.11 – Пошук максимального елементу та перестановка рядків

```
REAL A(N,N),B(N),AII,ABSA
INTEGER J,K
...
AII=ABS(A(1,1))
K=1
DO J=2,N
ABSA=ABS(A(J,1))
IF (ABSA.GT.AII) THEN
AII=ABSA
K=J
END IF
END DO
IF (K.NE.1) THEN
DO J=1,N
AII=A(1,1)
A(1,1)=A(K,1)
```

```
      A(K,1)=AII
      END DO
      AII=B(1)
      B(1)=B(K)
      B(K)=AII
```

В системі рівнянь з матрицею (3.23) у другому та усіх подальших рівняннях виключено першу невідому. Аналогічним чином виконаємо перетворення матриці

$$\begin{pmatrix} \alpha_{22} & \alpha_{23} & \dots & \alpha_{2n} & \beta_2 \\ \alpha_{32} & \alpha_{33} & \dots & \alpha_{3n} & \beta_{32} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \alpha_{n2} & \alpha_{n3} & \dots & \alpha_{nn} & \beta_n \end{pmatrix}, \tag{3.24}$$

яка має розмір $(n-1)\times n$. Далі будемо розглядати матрицю розміром $(n-2)\times(n-1)$ і т.д. впритул до матриці розміром 1×2. Звісно, що в програмі, яка реалізує метод Гауса, немає необхідності вводити масиви для усіх цих «проміжних» матриць, оскільки кожна із них є частиною вихідної матриці та розташована у відповідній частині масиву, в якому зберігається уся матриця системи. Для цього потрібно реалізувати цикл по кількості рівнянь та кожний наступний крок цього циклу реалізовувати для коефіцієнтів, індекси яких перевищують індекс цього циклу, як це реалізовано, наприклад, у вигляді підпрограми, що реалізує прямий хід методу Гауса:

Лістинг 3.12 – Прямий хід методу Гауса

```
      LOGICAL FUNCTION GAUSS1(A,B,N)
      INTEGER N,I,J,K
      REAL A(N,N),B(N),AII,ABSA
      DO I=1,N
CCC   MAX ELEMENT FINDING FOR COLUMN NUMBER I
      AII=ABS(A(I,I))
      K=I
      DO J=I+1,N
      ABSA=ABS(A(J,I))
      IF (ABSA.GT.AII) THEN
      AII=ABSA
      K=J
      END IF
      END DO
      IF (AII.EQ.0) THEN
      GAUSS1=.FALSE.
```

```
      GOTO 100
      END IF
CCC   REARRANGEMENT ROW I AND ROW K IF K NOT EQUAL I
      IF (K.NE.I) THEN
      DO J=I,N
      AII=A(I,J)
      A(I,J)=A(K,J)
      A(K,J)=AII
      END DO
      AII=B(I)
      B(I)=B(K)
      B(K)=AII
      END IF
CCC   DIVIDING ROW I
      AII=A(I,I)
      DO J=I,N
      A(I,J)=A(I,J)/AII
      END DO
      B(I)=B(I)/AII
CCC   NULING ROWS I+1,I+2,..., N IN COLUMN I
      DO K=I+1,N
      AII=A(K,I)
      DO J=I,N
      A(K,J)=A(K,J)-A(I,J)*AII
      END DO
      B(K)=B(K)-B(I)*AII
      END DO
      END DO
      GAUSS1=.TRUE.
100   END
```

Вибір підпрограми-функції для реалізації прямого ходу методу Гауса потрібен для повідомлення про випадок, коли детермінант матриці системи дорівнює нулю, який призводить до помилки ділення на нуль.

3.4.3 Перетворення зворотного ходу методу Гауса. Зворотний хід методу Гауса здійснюється за алгоритмом (3.3.3), який реалізуємо таким чином, що результат розв'язку системи лінійних алгебраїчних рівнянь буде розташований в масиві, який був спочатку передбачений для зберігання вектора правої частини системи. Відповідна підпрограма має вигляд:

Лістинг 3.13 – Зворотний хід методу Гауса

```
SUBROUTINE GAUSS2(A,B,N)
INTEGER N,I,J
```

```
REAL A(N,N),B(N)
DO I=N,1,-1
DO J=N,I+1,-1
B(I)=B(I)-A(I,J)*B(J)
END DO
END DO
END
```

Для перевірки роботи підпрограм GAUSS1 та GAUSS2 використовуємо системи лінійних алгебраїчних рівнянь із матрицею Гілберта [42, 43]

$$a_{ij} = \frac{1}{i+j-1}, \quad i, j = 1,2,...,n. \tag{3.25}$$

В якості вектора правої частини вибираємо наступний:

$$b_i = -\sum_{j=1}^{n} j \cdot a_{ij}, \quad i = 1,2,...,n. \tag{3.26}$$

Вектору (3.25) відповідає розв'язок

$$x_i = i, \quad i = 1,2,...,n. \tag{3.27}$$

Використовувати підпрограми GAUSS1 та GAUSS2 можна, наприклад, так:

Лістинг 3.14 – Використання підпрограм розв'язування систем лінійних алгебраїчних рівнянь

```
PROGRAM GAUSS
INTEGER N,I,J
REAL A(:,:), B(:)
LOGICAL GAUSS1
N=3
ALLOCATE(A(N,N),B(N))
DO I=1,N
B(I)=0
DO J=1,N
A(I,J)=1.0/(I+J-1)
B(I)=B(I)+J*A(I,J)
END DO
END DO
CALL PRINTSYSTEM(A,B,N)
```

```
      IF (GAUSS1(A,B,N)) THEN
      CALL PRINTSYSTEM(A,B,N)
      CALL GAUSS2(A,B,N)
      CALL PRINTSYSTEM(A,B,N)
      ELSE
      PRINT *,'DETERMINANT IS ZERO'
      END IF
      DEALLOCATE(A,B)
      PAUSE 'PRINT ENTER TO EXIT'
      END
      SUBROUTINE PRINTSYSTEM(A,B,N)
      INTEGER N,I,J
      REAL A(N,N),B(N)
      PRINT*,'SYSTEM:'
      DO I=1,N
      DO J=1,N
      PRINT '(F10.4,$)',A(I,J)
      END DO
      PRINT '(F10.4)',B(I)
      END DO
      END
      LOGICAL FUNCTION GAUSS1(A,B,N)
      ...
100   END
      SUBROUTINE GAUSS2(A,B,N)
      ...
      END
```

В цій програмі зворотний хід методу Гауса реалізується тільки за умов, коли при виконанні прямого ходу не виявилося, що детермінант матриці системи дорівнює нулю. В програмі передбачений друк вихідної матриці та матриці після прямого та зворотного ходу методу Гауса, що дозволяє спостерігати за роботою програми.

4 ЗБЕРІГАННЯ ДАНИХ ЗА ДОПОМОГОЮ ТЕКСТОВИХ ФАЙЛІВ

Текстовий файл є найбільш універсальним засобом обміну даними між різними програмами. На відміну від бінарних файлів, текстові файли надають можливостей обміну даними між програмами, що створені за допомогою різних мов програмування та компіляторів. Ідея текстового файлу полягає у представлені інформації у вигляді звичайного символьного тексту шляхом кодування алфавіту та спеціальних символів.

4.1 Константи та змінні текстового типу

Використання текстових файлів ґрунтується на поняттях про дані текстового типу, які можна представляти у вигляді текстових констант та змінних текстового типу. Дані текстового типу являють собою ланцюжок, що містить одну або декілька літер. Кожна літера текстових даних займає одну текстову одиницю пам'яті.

4.1.1 Константи текстового типу. Послідовність літер, що укладена в одинарних лапках, в мові програмування FORTRAN розглядається як константа текстового типу [9]. Одинарні лапки визначають початок та закінчення визначення даних константи текстового типу в тексті програми, наприклад

```
'INPUT DATA'
```

Довжина константи текстового типу – це кількість літер, що міститься в цій константі, тобто кількість літер, включаючи пробіли, що містяться між одинарними лапками, якими визначена константа; граничні одинарні лапки не входять до текстової константи та не враховуються при визначенні її довжини. Літери текстової константи є пронумерованими зліва направо таким чином, що крайня ліва літера має номер 1, наступна – номер 2 і т.д. до останньої літери.

Текстові константи можна друкувати за допомогою інструкції `PRINT` й використовувати для опису формату виведення даних в цій інструкції, а також для виведення повідомлень, наприклад в інструкції `PAUSE`:

Лістинг 4.1 – Використання текстових констант для виведення повідомлень

```
PROGRAM TEXT1
REAL A
```

```
A=25.4
PRINT '(A,F10.4)','A = ',A
PAUSE 'PRESS ENTER'
END
```

Це використовувалося раніше, як, наприклад, в п.п. 1.1.3 (див. лістинг 1.4). Слід зазначити, що символ лапки визначається всередині текстової константи за допомогою двох розташований поруч лапок:

```
'DATA AREN''T INPUTED'
```

Дві розташовані поруч лапки – це насправді один символ, тобто у наведеному прикладі довжина текстової константи становить 19, а не 20. Лапки можуть бути використані, наприклад, всередині текстових констант, які визначають формати введення та виведення в інструкціях `READ` та `PRINT`, що може бути зручним з точки зору читання тексту програми в деяких випадках. Так, в тексті програми, що наведена на лістингу 4.1, інструкцію `PRINT` можна використати по-іншому [9]:

```
PRINT '(''A='',F10.4)',A
```

4.1.2 Змінні текстового типу. Окрім констант текстового типу передбачені також змінні текстового типу, які містять в собі дані текстового типу, що можуть змінюватися при роботі програми [6, 9, 29, 30]. Для визначення змінних текстового типу в мові програмування `FORTRAN` передбачено інструкцію `CHARACTER`, яким визначається змінна текстового типу заданої довжини. Довжина змінної текстового типу вказується у вигляді цілочислової константи, що розташована після символу «*», який у свою чергу має бути розташований після інструкції `CHARACTER` або після імені змінної текстового типу, як, наприклад:

```
CHARACTER*5 A,B,C*10,D(6)*16
```

Визначник довжини у вигляді символу «*» та цілочислової константи, який розташований безпосередньо після інструкції `CHARACTER`, визначає довжину усіх змінних текстового типу, для яких далі довжина не вказана. Покажчик довжини, що розташований біля імені змінної, визначає довжину тільки цієї змінної. У наведеному прикладі введені дві змінні завдовжки 5, одна змінна завдовжки 10, та одновимірний масив завдовжки 6, елементи якого – текстові дані завдовжки 16.

Використання текстових змінних дозволяє, наприклад, здійснювати програмне формування виведення даних, як, наприклад, у програмі, що

наведено на лістингу 4.2. В цій програмі формат виведення даних визначається програмою та може змінюватися відповідним чином. Це дозволяє частково автоматизувати виведення вихідних даних шляхом створення спеціальної підпрограми та її подальшого використання в програмах різного призначення. В наведеному прикладі дані текстового типу передаються в якості «вхідних» параметрів до підпрограми. В наведеному прикладі довжина текстової змінної обрана таким чином, щоб вона перебільшувала максимальну довжину текстових даних, які будуть зберігатися в цій змінній.

Лістинг 4.2 – Частково автоматизоване виведення даних з використанням текстових змінних для визначення формату

```
PROGRAM TEXT2
REAL A,B
CHARACTER*20 VARNAME
A=1.2
B=3.5
VARNAME='(''A='',F10.4)'
CALL PRINTVAR(VARNAME,A)
VARNAME='(''B='',F10.4)'
CALL PRINTVAR(VARNAME,B)
PAUSE 'PRESS ENTER'
END
SUBROUTINE PRINTVAR(NAME,VALUE)
CHARACTER*20 NAME
REAL VALUE
PRINT NAME,VALUE
END
```

Змінні текстового типу допомагають спрощувати програмне керування введенням вихідних даних та виведенням результатів. Так, в лістингу 4.3 наведено програму, в якій користувач може здійснювати вибір одиниць вимірювання даних про лінійні розміри, а використання змінної текстового типу суттєво спрощує формування текстових повідомлень щодо роботи програми на екрані монітора. В цьому прикладі присвоєння значення змінної текстового типу здійснюється у залежності від значення заздалегідь введеної змінної цілочислового типу та використовується далі при формуванні повідомлень на екрані монітора для супроводу роботи комп'ютерної програми. Слід звернути увагу, що при визначенні формату за допомогою інструкції `FORMAT` немає необхідності у подвійних лапках. Значення змінних текстового типу можна вводити з клавіатури безпосередньо (лістинг 4.4).

Лістинг 4.3 – Частково автоматизоване формулювання повідомлень з використанням змінних текстового типу

```
PROGRAM TEXT3
```

```
      INTEGER INDEXDIM
      CHARACTER*3 NAMEDIM
      REAL VALDIM
      PRINT *,'AVAILABLE DIMENSION INDEX:'
      PRINT*,'1 - M'
      PRINT*,'2 - SM'
      PRINT*,'3 - MM'
      PRINT*,'4 - MKM'
      PRINT '(''INPUT DIMENSION INDEX (1-4): '',$)'
      READ '(I1)',INDEXDIM
      IF (INDEXDIM.EQ.1) NAMEDIM='M'
      IF (INDEXDIM.EQ.2) NAMEDIM='SM'
      IF (INDEXDIM.EQ.3) NAMEDIM='MM'
      IF (INDEXDIM.EQ.4) NAMEDIM='MKM'
      VALDIM=1.7
      PRINT 10,VALDIM,NAMEDIM
10    FORMAT ('SIZE = ',F10.4,'(',A3,')')
      PAUSE 'PRESS ENTER TO EXIT PROGRAM'
      END
```

Лістинг 4.4 – Введення з клавіатури змінних текстового типу

```
      PROGRAM TEXT4
      CHARACTER*3 NAMEDIM
      PRINT '(''INPUT DIMENSION NAME:'',$)'
      READ '(A3)',NAMEDIM
      PRINT '(A3)',NAMEDIM
      PAUSE 'PRESS ENTER TO EXIT PROGRAM'
      END
```

Для виведення та введення змінних текстового типу слід використовувати дескриптор A, який був розглянутий раніше в п.п. 1.2.3.1 (див. лістинг 1.7).

4.1.3 Дії над текстовими константами та змінними. Над даними текстового типу можна здійснювати певні дії, які дозволяють в результаті створювати нові дані.

Для перетворення даних текстового типу в мові програмування FORTRAN передбачена одна операція – конкатенація, тобто склеювання, яка до даних текстового типу додає праворуч інші дані текстового типу; конкатенація позначається комбінацією символів «//» [6, 9, 29, 30]. В програмі, що наведена на лістингу 4.5, за допомогою конкатенації здійснюється формування текстового повідомлення користувачу програми. Слід звернути увагу, що можна робити декілька конкатенацій в одному виразі; результатом таких дій буде приєднування праворуч до текстових даних інших текстових

даних відповідно до їхнього порядку конкатенації. Так, якщо маємо три змінні A, B та C текстового типу, то результатом конкатенації вигляду A//C//B будуть текстові дані, утворені приєднанням до даних A даних C та приєднання до цього результату ще й даних B. Якщо A='INPUTING', B='COMPLETE', а C=' DATA ', то в результаті складеної конкатенації D=A//C//B одержимо, що D='INPUTING DATA COMPLETE'.

Лістинг 4.5 – Формулювання повідомлення з використанням конкатенації даних текстового типу

```
PROGRAM TEXT5
CHARACTER NAMEDIM*3, MESSAGE*50
PRINT '(''INPUT DIMENSION NAME:'',$)'
READ '(A3)',NAMEDIM
MESSAGE='DIMENSION IS '//NAMEDIM
PRINT '(A50)',MESSAGE
PAUSE 'PRESS ENTER TO EXIT PROGRAM'
END
```

В мові програмування FORTRAN передбачено можливість витягування текстових даних із змінних текстового типу [6, 9, 29, 30]. Текстовий ланцюжок – це послідовність розташованих підряд літер, які входять до складу змінної текстового типу та можуть бути використаними як дані текстового типу. Текстовий ланцюжок формується наступним чином:

$$\texttt{var(beg:fin)}, \tag{4.1}$$

де var – ім'я змінної текстового типу або посилання на елемент масиву текстового типу; beg – цілочислове значення у вигляді константи або виразу, що визначає номер першого символу текстового ланцюжка, який він займає в текстовій змінній var; fin – цілочислове значення у вигляді константи або виразу, що визначає номер останнього символу ланцюжка, який він займає в текстовій змінній var.

Зазначимо, що beg та fin називають граничними виразами ланцюжка та те, що вони можуть визначатися програмно через значення цілочислових змінних або виразів. Пояснимо використання ланцюжків (4.1.1) на прикладах. Нехай для прикладу маємо змінну D текстового типу таку, що вона дорівнює 'INPUTING DATA COMPLETE'. Тоді ланцюжок D(1:8) буде дорівнювати 'INPUTING'; ланцюжок D(10:13) буде дорівнювати 'DATA'; ланцюжок D(15:22) буде визначатися як 'COMPLETE'. Ланцюжок, у якого параметри beg та fin збігаються, визначає символ, що міститься в текстовій змінній var на відповідному місці. На лістингу 4.6 наведено програму, в якій за допомогою використання ланцюжків здійсню-

ється введення з клавіатури текстової змінної та потім в окремих рядках друкуються окремі літери, що входять до цієї введеної текстової змінної у порядку, в якому вони розташовані в цій введеній текстовій змінній.

В програмі, що наведена на лістингу 4.6, в якості лічильника циклу виведення літер текстової змінної використана цілочислова змінна `I`, опис якої відсутній в тексті програми. В цьому зв'язку слід відзначити, що в мові програмування FORTRAN [9] передбачено визначення типу змінних за замовченням, відповідно до якого усі змінні, імена яких починаються з літер I, J, K, L, M, N, розглядаються як змінні цілочислового типу, якщо тип цих змінних не був визначений окремо в тексті програми за допомогою інструкцій визначення типу. Визначення типу за замовченням, або з контексту значень, які їм присвоюються в програмі, притаманний також деяким іншим мовам програмування, наприклад Python, хоча в таких розповсюджених мовах програмування, як PASCAL та C, тип даних для усіх без виключення змінних має визначатися в тексті програми.

Лістинг 4.6 – Друк літер текстової змінної

```
PROGRAM TEXT6
CHARACTER TEXTVAR*20
PRINT '('INPUT TEXT VARIABLE:'',$)'
READ '(A20)',TEXTVAR
DO I=1,20
PRINT '(A1)',TEXTVAR(I:I)
END DO
PAUSE 'PRESS ENTER TO EXIT PROGRAM'
END
```

Слід зазначити, що довжина текстових даних, які є результатом функції або формальним параметром підпрограми, може бути невизначеною та визначатися у залежності від результатів роботи програми. Текстові дані невизначеної довжини вводять у програму шляхом опису їх як звичайних текстових змінних, але на місці, де має визначатися їхня довжина, розташовують спеціальний описувач довжини у вигляді зірочки-множника у круглих дужках. Для роботи із текстовими даними в мові програмування FORTRAN передбачена дуже корисна стандартна функція [9, 30], яка дозволяє визначати довжину даних констант та змінних текстового типу, тобто аргумент. Ця функція має вигляд:

```
LEN(textvar),                                    (4.2)
```

де `textvar` – ім'я змінної текстового типу.

Використання текстових даних невизначеної довжини та функції (4.2) визначення довжини дозволяє створювати універсальні підпрограми

для обробки текстових даних. Наприклад, алгоритм програми, що наведена на лістингу 4.6, можна оформити у вигляді підпрограми, та використовувати цю підпрограму для відповідної обробки текстових даних різної довжини, наприклад, як це наведено нижче:

Лістинг 4.7 – Друк літер текстових змінних різної довжини

```
PROGRAM TEXT7
CHARACTER TVAR1*20, TVAR2*30
PRINT '(''INPUT TEXT VARIABLE (20):'',$)'
READ *,TVAR1
PRINT '(''INPUT TEXT VARIABLE (30):'',$)'
READ *,TVAR2
CALL PRINTLITERS(TVAR1)
CALL PRINTLITERS(TVAR2)
PAUSE 'PRESS ENTER TO EXIT PROGRAM'
END
SUBROUTINE PRINTLITERS(A)
CHARACTER*(*) A
DO I=1,LEN(A)
PRINT '(I2,''-'',A1)',I,A(I:I)
END DO
END
```

4.2 Базові поняття та інструкції, що зв'язані із файлами

Збереження даних у файлах на зовнішніх носіях, таких як магнітні диски та флеш-пам'ять, дозволяє зберігати великі обсяги даних, які значно перевищують обсяги оперативної пам'яті комп'ютерів. Результати деяких розрахунків містять настільки великі обсяги даних, що їх неможливо зберігати в оперативній пам'яті комп'ютера, і для їхнього зберігання слід використовувати зовнішні носії. Крім того, зберігання даних на зовнішніх носіях дозволяє багаторазово звертатися до даних, в той час як дані, що зберігаються в оперативній пам'яті, знищуються після вимкнення комп'ютера.

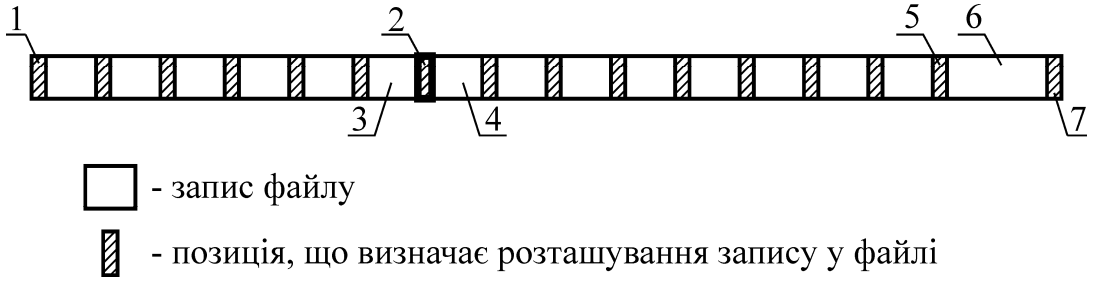
4.2.1 Записи та файли. Ключовим поняттям при роботі із файлами є поняття про запис – послідовність значень даних або літер, яка записується до файлу або читається із файлу при зверненні до файлу; кількість записів, що містяться у файлі, визначається кількістю звернень запису до цього файлу. Запис як поняття, що пов'язане із файлами, в мовах програмування, у тому числі в FORTRAN, – це є суто абстрактне поняття, яке не має жодного зв'язку із фізичним розташуванням записів файлу на

зовнішньому носії. Введення поняття про «логічний запис» як складовий елемент файлу в мовах програмування дозволяє програмістам уникнути необхідності щодо безпосереднього керування зовнішніми носіями, яке здійснюється автоматично операційною системою.

Існує два види записів: форматні та безформатні. Форматні записи в мові програмування `FORTRAN` – це текстові дані у вигляді набору літер, якими у зручній для людині формі можуть бути представлені різноманітні дані, будь то текстові коментарі, або числа, або дані логічного типу. Форматні записи є базовим елементом текстового файлу, який може відображатися багатьма програмами, у тому числі відомою програмою «Блокнот» фірми MicroSoft. Формат перетворення числових даних у текстову форму при записуванні до текстового файлу визначається інструкцією FORMAT або тотожним чином, як це робиться при виведенні даних на екран монітора за допомогою інструкції `PRINT`. Довжина форматного запису визначається кількістю літер. Безформатний запис містить числові, логічні та текстові дані, які зберігаються у деякому внутрішньому представленні, що залежить від процесора. Файли, що створюються з використанням без форматних записів, використовуються в тих випадках, коли слід зберігати дані на зовнішньому носії для подальшого читання без редагування. Довжина без форматного запису вимірюється в одиницях, що залежать від процесора, та визначається змістом списку виведення. Файли із безформатними записами не завжди можуть бути використаними в програмах, що створені різними компіляторами. Натомість форматні текстові файли можуть бути використаними в програмах, що створені на різних мовах програмування з використанням різних компіляторів.

Файл – це послідовність логічних записів, що зберігаються на зовнішньому носії (магнітному диску або флеш-пам’яті), тому між окремими записами існує відношення прямування. Метод доступу визначає порядок, в якому записи можуть бути записані до файлу або прочитані з файлу. Відомо два методи доступу до файлу: прямий та послідовний. При прямому доступі упорядкованість записів файлу визначається виключно їхніми номерами незалежно від порядку, в якому вони були записані до файлу; номер запису має вказуватися в інструкції, що записує або зчитує дані. Усі записи файлу з прямим доступом мають мати однакову довжину та мають бути виключно форматними, що відповідним чином обмежує використання файлів прямого доступу. При послідовному доступі упорядкованість записів файлу визначається послідовністю, в якій ці записи були поміщені до файлу (рис. 4.1); читання даних із записів файлу послідовного доступу можливе лише у порядку, в якому ці записи були поміщені до файлу. Для визначення порядку записів файлу послідовного доступу використовують поняття про початкову, поточну та кінцеву

позиції (точки); про попередню та про наступну записи; про запис закінчення файлу та про позицію (точку) закінчення файлу. Початкова позиція (точка) – це місце у файлі, що розташоване перед першим записом; поточна позиція (точка) – це місце у файлі, що розташоване одразу ж після запису, який був прочитаний із файлу або був записаний до файлу; кінцева позиція (точка) – це місце у файлі, що розташоване одразу ж після останнього запису (рис. 4.1). Попередній запис – це останній запис, що був прочитаний із файлу або записаний до файлу; наступний запис – це запис, який слідує за попереднім записом (рис. 4.1). Запис закінчення файлу – це спеціальний запис, який розташований одразу ж після кінцевої позиції та є ознакою про завершення файлу; позиція (точка) закінчення файлу – це місце у файлу, що розташоване після запису про закінчення файлу (рис. 4.1). Початкова, поточна, кінцева позиції та позиція закінчення файлу є уявними та фізично у файлі не існують – усі записи файлу слідують один за одним безпосередньо, але поняття про позиції записів у файлі є необхідним для програмування роботи із файлами послідовного доступу.



1 – початкова позиція (точка); 2 – поточна позиція (точка);
3 – попередній запис; 4 – наступний запис; 5 – кінцева позиція (точка);
6 – запис про закінчення файлу; 7 – позиція (точка) закінчення файлу

Рисунок 4.1 – Відношення упорядкованості
в файлі послідовного доступу

Звернення до файлу в мові програмування FORTRAN здійснюється через канал, який є цілочисловим значенням, що має дорівнювати або бути більшим нуля. Під'єднанням між каналом та файлом називається встановлення зв'язку, а точніше відповідності, між каналом та файлом. До каналу може бути під'єднаним одночасно тільки один файл, але після роз'єднання між каналом та файлом до каналу може бути під'єднаним інший файл. Встановлення та розірвання зв'язку між каналом та файлом здійснюється за допомогою інструкцій `OPEN` та `CLOSE` відповідно. Інструкція `OPEN`, яка встановлює зв'язок між каналом та файлом, є взагалі достатньо складною, оскільки вона може містити включно до дев'яти специфікацій, які мають бути переліченими через кому у круглих дужках одразу ж після інструкції, але більшу кількість цих специфікацій можна опустити, і вони приймуть

значення за замовченням, відповідні методу послідовного доступу до файлу. У найпростішому вигляді інструкція `OPEN` може мати тільки дві специфікації: номер каналу та ім'я файлу. Інструкція `CLOSE`, яка розриває зв'язок між каналом та файлом, має обов'язково містити лише одну специфікацію – номер каналу, хоча в цій інструкції взагалі передбачені чотири специфікації.

Специфікації щодо номера каналу в інструкціях `OPEN` та `CLOSE` мають бути визначеними у формі:

```
UNIT=number,                                    (4.3)
```

де `number` – номер каналу.

Специфікації щодо номера каналу в інструкції `OPEN` мають бути визначені у формі:

```
FILE=fname,                                     (4.4)
```

де `fname` – ім'я файлу у вигляді текстової константи або змінної.

Приклад використання інструкцій `OPEN` та `CLOSE` із визначенням мінімальної кількості необхідних специфікацій наведений у лістингу 4.8. В цьому прикладі встановлюється зв'язок між каналом з номером 3 та текстовим файлом з іменем MYFILE.TXT для додавання даних до цього файлу або зчитування даних з цього файлу.

Лістинг 4.8 – Встановлення та розірвання зв'язку між каналом та файлом

```
...
C     CREATING CONNECTION WITH FILE AND CHANNEL
      OPEN(UNIT=3,FILE='MYFILE.TXT')
CC    BEGINNING WORKING WITH FILE
...
CC    FINISHING WORKING WITH FILE
C     BREAKING CONNECTION WITH FILE AND CHANNEL
      CLOSE(UNIT=3)
...
```

Ім'я файлу може містити відомості про його розташування на зовнішньому носії, як це прийнято в операційних системах дискових типу, наприклад: "`D:\MYFOLDER\MYFILE.TXT`".

4.2.2 Запис даних до файлу послідовного доступу. Для запису даних до файлу послідовного доступу в мові програмування FORTRAN передбачена інструкція `WRITE`, яка має використовуватися наступним чином:

```
WRITE(controllist)datalist,          (4.5)
```

де `controllist` – список керуючої інформації; `datalist` – список даних для виведення до файлу.

Список керуючої інформації інструкції `WRITE` може містити до шести параметрів, які перелічуються через кому, але не всі з них обов'язково мають бути вказані. Обов'язково слід вказувати номер каналу та формат запису, який визначається як в інструкції `FORMAT`. Список даних для виведення до файлу має вигляд перелічених через кому даних у вигляді констант та (або) змінних; тип даних має відповідати послідовності, що передбачена в описі формату, як це робиться в інструкції `PRINT`. В якості прикладу використання файлу для зберігання даних на лістингу 4.9 наведено програму, в якій здійснюється збереження даних одновимірного масиву у файл.

Лістинг 4.9 – Збереження даних про одновимірний масив у текстовому файлі послідовного доступу

```
PROGRAM FILE1
REAL A(:)
INTEGER N
CHARACTER*7 FNAME
PRINT '(''INPUT COUNT ELEMENTS: '',$)'
READ '(I3)',N
ALLOCATE(A(N))
DO I=1,N
PRINT '(''INPUT ELEMENTS COUNT'',I3,'' : '',$)',I
READ '(F10.4)',A(I)
PRINT '(''A('',I3,'')='',F10.4)',I,A(I)
END DO
PRINT '(''INPUT FILE NAME: '',$)'
READ '(A7)',FNAME
OPEN(UNIT=1,FILE=FNAME//'.TXT')
WRITE(1,'(I3)') N
DO I=1,N
WRITE(1,'(F10.4)') A(I)
END DO
CLOSE(UNIT=1)
DEALLOCATE(A)
PAUSE 'PRESS ENTER TO EXIT'
END
```

Після завершення роботи програми, що наведена на лістингу 4.9, користувач комп'ютера може побачити на жорсткому диску в каталозі, де

розташований проект, текстовий файл, що містить дані про введений одновимірний масив. Ім'я файлу буде таким, який користувач ввів на відповідний запит програми; зміст цього файлу можна побачити за допомогою програми «Блокнот», яка має бути на кожному комп'ютері з операційною системою WINDOWS. Зазначимо, що цілочислова змінна `N`, яка визначає довжину масиву, позначена в програмі, а цілочислова змінна `I`, яка використовується як лічильник циклів, вводиться за замовченням.

4.2.3 Читання даних з файлу послідовного доступу. Для читання даних з файлу послідовного доступу в мові програмування FORTRAN передбачений спеціальний вигляд інструкції `READ`, яка має використовуватися наступним чином:

```
READ(controllist)datalist,                    (4.6)
```

де `controllist` – список керуючої інформації; datalist – список даних для виведення до файлу.

Список управляючої інформації інструкції `READ` є повністю аналогічним списку керуючої інформації інструкції `WRITE`, що розглянута у вигляді (4.5), та може містити до шести параметрів, які перелічуються через кому, але не всі з них обов'язково мають бути вказані. Обов'язково слід вказувати номер каналу та формат запису, який визначається як в інструкції `FORMAT`. Список даних для читання із файлу має вигляд перелічених через кому даних у вигляді констант та (або) змінних; тип даних має відповідати послідовності, що передбачена в опису формату, як це робиться у звичайній інструкції `READ`, яка використовується для введення даних з клавіатури. В якості прикладу використання файлу для зчитування даних на лістингу 4.10 наведено програму, в якій здійснюється зчитування даних одновимірного масиву з текстового файлу, ім'я якого має ввести за допомогою клавіатури користувач програми.

Лістинг 4.10 – Читання даних про одновимірний масив із файлу

```
PROGRAM FILE2
REAL B(:)
INTEGER N
CHARACTER*7 FNAME
PRINT '(''INPUT FILE NAME: '',$)'
READ '(A7)', FNAME
OPEN(UNIT=1,FILE=FNAME//'.TXT')
READ(1,'(I3)') N
ALLOCATE(B(N))
DO I=1,N
READ(1,'(F10.4)') B(I)
```

```
PRINT '(''B('',I3,'')='',F10.4)',I,B(I)
END DO
CLOSE(UNIT=1)
DEALLOCATE(B)
PAUSE 'PRESS ENTER TO EXIT'
END
```

Для коректної роботи програми, що наведена в лістингу 4.10, на жорсткому диску комп'ютера у папці, де зберігаються файли проекту, має міститися файл, ім'я якого користувач буде вводити на відповідний запит програми. Якщо такого файлу не буде або користувач помилиться при введенні імені файлу, то це призведе до фатальної помилки, що призведе до термінового завершення роботи програми. Результатом роботи програми буде створення масиву відповідно до даних, що містяться у вказаному користувачем файлі, та виведення елементів цього масиву на екран монітора, що дозволить перевірити правильну роботу програми щодо зчитування даних з файлу. Слід також зазначити, що файл, необхідний для роботи програми, наведеної на лістингу 4.10, може бути створений іншою програмою, наприклад, що наведена вище на лістингу 4.9, а також за допомогою будь-якої програми для редагування текстових файлів, наприклад у програмі «Блокнот». При створенні текстового файлу, необхідного для роботи програми, що наведена вище на лістингу 4.10, за допомогою інших програм та редакторів текстових файлів слід зміст створюваного файлу узгодити із форматом читання даних з цього файлу, а саме: у першому рядку має бути цілочислове значення, що має довжину 3 символи та визначає довжину одновимірного масиву, і якщо довжина менша 10, то перед числом, що визначає довжину, має бути два пробіли; кількість наступних рядків має дорівнювати вказаній у першому рядку довжині, а кожне числове значення має відповідати передбаченому у програмі (див. лістинг 4.10) формату `F10.4`, тобто для розміщення у символьному вигляді дійсного числа необхідно передбачити 10 символів, включаючи знак числа та десятинну крапку та чотири числа після цієї крапки, а також слід використовувати пробіли для невикористаних позицій.

Наведені тут відомості щодо збереження даних у файлах не охоплюють усіх можливостей, передбачених у мові програмування FORTRAN-77, але містять базові знання щодо роботи з файлами, притаманні різноманітним мовам програмування. Зазначимо, що використання файлів є ефективним засобом керування роботи програми за рахунок збереження у файлах окремих даних, необхідних для налаштування роботи програми. Наприклад, програма після свого запуску може зчитувати дані зі спеціально передбаченого файлу ініціалізації та продовжувати свою роботу відповідно до цих зчитаних даних. Це дозволяє змінювати роботу програми, не змінюючи її

текст, а змінюючи лише зміст файлу ініціалізації, що можна зробити будь-якою програмою-редактором текстових файлів, наприклад програмою «Блокнот». Такий нескладний прийом дозволяє створювати достатньо універсальні програми, користування якими не потребує перепрограмування та перекомпіляції програм. Зрозуміло, що файли можна використовувати у програмах також і для збереження як остаточних, так і проміжних результатів розрахунків, а в деяких випадках таке збереження є обов'язковим.

4.3 Використання текстових файлів для обміну даними

Забезпечення можливостей щодо обміну даними між програмами дозволяє використовувати різноманітні готові програми та створювати необхідні програми за допомогою різних мов програмування, що є більш пристосованими до розв'язування окремих задач. Наприклад, для розробки програм щодо виконання великих обсягів математичних розрахунків зручно використовувати мову програмування FORTRAN, але для вивчення результатів таких розрахунків зручно використовувати різноманітні спеціалізовані програми із досконалими графічними інтерфейсами, що дозволятимуть наочно представляти результати у вигляді двовимірних графіків тощо. Розглянемо далі приклад використання текстових файлів для зберігання результатів розрахунку у вигляді, зручному для подальшої побудови двовимірних графіків за допомогою інших програм, у тому числі стандартних програм роботи з електронними таблицями, таких як EXCEL.

4.3.1 Формування даних для побудови графіків. Двовимірний графік являє собою плоску криву, яка використовується для наочної ілюстрації залежностей між різними величинами. З математичної точки зору графік функції (далі розглядаємо функції, що є відображеннями множини дійсних чисел R на себе) являє залежність (рис. 4.2):

$$y = f(x),\ a \le x \le b,\quad (4.7)$$

де $x, y \in R$; f – деяка задана неперервна функція; a, b – границі області визначення цієї функції.

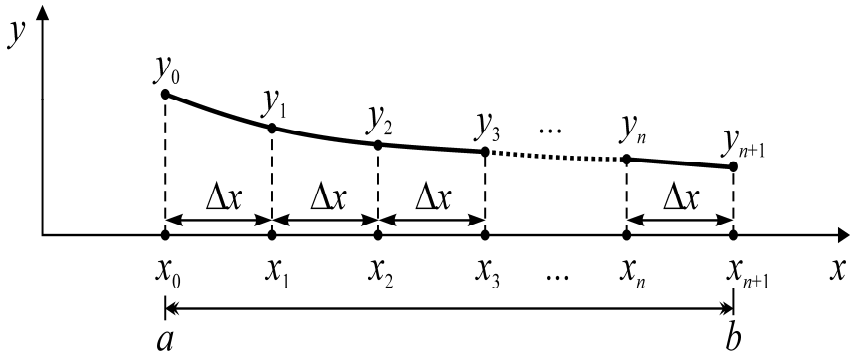


Рисунок 4.2 – Графік та дані для його побудови

Нагадаємо, що комп'ютер є перетворювачем дискретної інформації (див. п.п. 1.1.1.1), тому для зображення графіка (4.7) його слід представити в дискретній формі. Для цього замість неперервної змінної – ординати

$x:\ a \le x \le b$ розглянемо скінченну множину окремих значень цієї змінної в інтервалі $[a,b]$ (рис. 4.2) та визначимо елементи цієї множини наступним чином:

$$x_k = a + \Delta x \cdot k, \quad \Delta x = \frac{b-a}{n+1}, \quad k = 0,1,2,\ldots,n,n+1, \tag{4.8}$$

де x_k – елемент з номером k множини значень змінної x в інтервалі $[a,b]$; Δx – відстань між сусідніми елементами; n – кількість елементів в інтервалі (a,b).

Множину вигляду (4.8) часто використовують не тільки для побудови графіків, але і для наближеного розв'язування задач, як це обговорюється, наприклад, в роботах [44–46], і часто називають сіткою в інтервалі $[a,b]$, величину Δx – відповідно кроком сітки, а значення x_k – вузлами сітки. Сітка фактично являє дискретну (відцифровану) форму наближеного представлення неперервного інтервалу $[a,b]$, в якому визначена функція (4.7). Далі, щоб представити функцію (4.7) в дискретній формі, визначимо її значення у вузлах сітки (4.8):

$$y_k = f(x_k), \quad k = 0,1,2,\ldots,n,n+1. \tag{4.9}$$

Таким чином, неперервна функція (4.7) представлена у дискретній формі, а саме – у вигляді таблиці, що містить значення абсцис (4.8) та відповідних їм ординат (4.9). За цим, результати для побудови графіка функції (4.7) можна представити в текстовому файлі у вигляді значень (4.8) та (4.9); відповідна підпрограма представлена на ліст. 4.11. В цій підпрограмі передбачено, що дані графіка представлені у вигляді двох одновимірних масивів, що містять абсциси (4.8) та ординати (4.9); при формуванні текстового файлу передбачається, що абсциси та ординати друкуватимуться в окремому рядку та розділятимуться крапкою із комою, що є зручним для подальшого зчитування, оскільки дозволяє відокремити дані.

Лістинг 4.11 – Підпрограма запису даних графіка до файлу

```
SUBROUTINE SAVECHART(N,X,Y,FNAME)
INTEGER N
REAL X(N),Y(N)
CHARACTER*(*) FNAME
OPEN(UNIT=1,FILE=FNAME)
DO I=1,N
WRITE (1,FMT='(E15.7,'';'',E15.7)') X(I),Y(I)
END DO
```

```
CLOSE(UNIT=1)
END
```

На ліст. 4.12 показано приклад використання підпрограми, що наведена на ліст. 4.11, в якому здійснюється виведення в текстовий файл даних для побудови графіка функції

$$y = x^2, \quad a \le x \le b. \tag{4.10}$$

В цій програмі передбачено введення границь a та b інтервалу, який визначає область визначення функції (4.10), а також кількість n вузлів в інтервалі (a,b) та здійснюється визначення ординат (4.8) та абсцис (4.9) графіка функції (4.10) відповідно до введених даних. В результаті роботи програми буде створений текстовий файл із іменем `'CHART.TXT'`, в якому відповідним чином будуть міститися дані, що необхідні для побудови графіка функції (4.10); цей файл можна переглянути за допомогою програми «Блокнот».

Лістинг 4.12 – Підпрограма запису даних графіка до файлу

```
PROGRAM CHART
INTEGER N
REAL A,B,X(:),Y(:),DX,F
PRINT '(A,\)', 'INPUT A: '
READ *,A
PRINT '(A,\)', 'INPUT B: '
READ *,B
PRINT '(A,\)', 'INPUT N: '
READ *,N
ALLOCATE(X(N+2),Y(N+2))
DX=(B-A)/(N+1)
DO I=0,N+1
X(I+1)=A+I*DX
Y(I+1)=F(X(I+1))
END DO
CALL SAVECHART(N+2,X,Y,'CHART.TXT')
DEALLOCATE(X,Y)
PAUSE 'PRESS ENTER TO EXIT'
END

REAL FUNCTION F(X)
REAL X
F=X**2
END
```

4.3.2 Дані для побудови графіків поліномів Чебишева. Вище (див. ліст. 4.11 та ліст. 4.12) було розглянуто формування текстового файлу із даними для побудови графіка однієї функції (4.7). Як свідчить розглянутий вище приклад (див. ліст. 4.11), формування текстового файлу для побудови графіка функції в принципі не є нескладним за умови наявності даних (4.8) та (4.9), що представляють цю функцію в дискретній формі та представлені у розглянутому прикладі у відповідних одновимірних масивах, які містять абсциси та ординати точок графіка цієї функції. Розглянемо більш складний випадок, коли необхідно сформувати текстовий файл із даними для побудови графіків декількох функцій, що може бути необхідним, наприклад, для подальшого їхнього порівняння:

$$y_i = f_i(x), \quad i = 1,2,...,N, \quad a \le x \le b, \tag{4.11}$$

де $x, y_i \in R$; f_i – деяка задана неперервна функція; a, b – границі області визначення функцій $f_i(x)$, $i = 1,2,...,N$.

Щоб представити дані про функції (4.11) в дискретній формі, спочатку замість неперервної змінної – ординати x: $a \le x \le b$ із (4.11) розглянемо скінченну множину окремих значень цієї змінної в інтервалі $[a,b]$ (рис. 4.3), та визначимо елементи цієї множини у вигляді (4.8). Далі, щоб представвити функції (4.11) в дискретній формі, визначимо їхні значення у вузлах сітки (4.8):

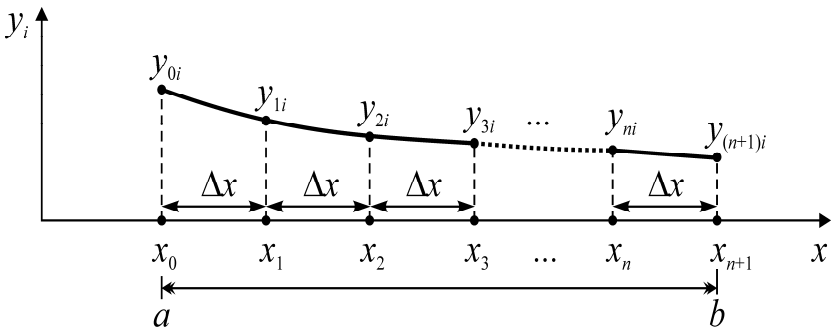


Рисунок 4.3 – Дані для побудови окремого графіка

$$y_{ji} = f_i(x_j), \quad j = 0,1,2,\ldots,n,n+1, \quad i = 1,2,...,N. \tag{4.12}$$

Таким чином, неперервні функції (4.11) представлені у дискретній формі, а саме – у вигляді таблиці, що містить значення абсцис (4.8) та відповідних їм ординат (4.12). За цим, результати для побудови графіків функцій (4.8) можна представити в текстовому файлі у вигляді значень (4.8) та (4.12); відповідна підпрограма представлена на ліст. 4.13. В цій підпрограмі передбачено, що дані графіка представлені у вигляді одновимірного масиву, що містить абсциси (4.8), та двовимірного масиву, що містить ординати (4.12); при формуванні текстового файлу передбачається, що абсциси та ординати друкуватимуться в окремому рядку та розділятимуться

крапкою із комою, що є зручним для подальшого зчитування іншими програмами, включаючи широко розповсюджені сьогодні програми електронних таблиць, оскільки дозволяє відокремлювати дані.

Лістинг 4.12 – Підпрограма запису даних декількох графіків до файлу

```
SUBROUTINE SAVECHARTN(NX,NY,X,Y,FNAME)
INTEGER N
REAL X(NX),Y(NX,NY)
CHARACTER*(*) FNAME
OPEN(UNIT=1,FILE=FNAME)
DO I=1,NX
WRITE (1,FMT='(F8.5,'';'',\)') X(I)
DO J=1,NY-1
WRITE (1,FMT='(F8.5,'';'',\)') Y(I,J)
END DO
WRITE (1,FMT='(F8.5)') Y(I,NY)
END DO
CLOSE(UNIT=1)
END
```

Як свідчить розглянутий вище приклад (див. ліст. 4.11), формування текстового файлу для побудови графіка функції в принципі не є нескладним за умови наявності даних (4.8) та (4.9), що представляють цю функцію в дискретній формі. У нескладних випадках, наприклад для функції (4.10), побудова даних (4.8), (4.9) не містила труднощів, що було показано на ліст. 4.12. Далі розглянемо більш складний та цікавий приклад, в якому слід побудувати поліноми Чебишева, що визначаються наступним чином [47]:

$$\begin{aligned} &T_{k+1}(x) = 2xT_k(x) - T_{k-1}(x), \quad T_0(x) = 1, \\ &T_1(x) = x, \quad k = 2,3,\ldots, -1 \le x \le 1. \end{aligned} \tag{4.13}$$

Поліноми Чебишева (4.13) пов’язані із багатьма фундаментальними математичними результатами [47]; завдяки своїм властивостям широко використовуються щодо розв’язування наукових та інженерних задач спектральними методами [48]. Зрозуміло, що поліноми Чебишева (4.13) можна побудувати за допомогою нескладних аналітичних перетворень та одержати:

$$\begin{aligned} &T_2(x) = 2x^2 - 1, \; T_3(x) = 4x^3 - 3x, \\ &T_4(x) = 8x^4 - 8x^2 + 1, \; \ldots \end{aligned} \tag{4.14}$$

Разом із тим, для обчислення поліномів Чебишева немає необхідності здійснювати аналітичні перетворення (4.13) та визначати кожен окремий поліном, як показано в (4.14). Дійсно, співвідношення (4.13) містять вичерпну інформацію, що дозволяє обчислювати будь-який поліном Чебишева для заданої ординати x, як це робиться у відповідному циклі в програмі, що представлена на ліст. 4.14. В цій програмі заздалегідь визначені та не можуть бути зміненими границі інтервалу, який визначає область визначення функцій (4.13), передбачається обов'язкове введення з клавіатури кількості n вузлів в інтервалі області визначення (див. рис. 4.3), а також обов'язкове введення з клавіатури кількості поліномів Чебишева (4.13), що мають бути обчисленими. Далі у циклі здійснюється визначення ординат та у внутрішньому циклі визначення відповідних ним абсцис, що відповідають необхідній кількості поліномів Чебишева. В результаті роботи програми буде створений текстовий файл із іменем 'CHEBYSHEV.TXT', в якому відповідним чином будуть міститися дані, що необхідні для побудови графіків поліномів Чебишева (4.13); зміст цього файлу можна переглянути за допомогою програми «Блокнот» або іншої програми відповідного призначення.

Лістинг 4.14 – Підпрограма запису даних графіка до файлу

```
PROGRAM CHEBYSHEV
INTEGER NX,NY
REAL A,B,X(:),Y(:,:),DX
A=-1.0
B=1.0
PRINT '(A,\)', 'INPUT NX: '
READ *,NX
PRINT '(A,\)', 'INPUT NY: '
READ *,NY
ALLOCATE(X(NX+2),Y(NX+2,NY))
DX=(B-A)/(NX+1)
DO I=0,NX+1
X(I+1)=A+I*DX
Y(I+1,1)=1.0
Y(I+1,2)=X(I+1)
DO J=3,NY
Y(I+1,J)=2*X(I+1)*Y(I+1,J)-Y(I+1,J-1)
END DO
END DO
CALL SAVECHARTN(NX+2,NY,X,Y,'CHEBYSHEV.TXT')
DEALLOCATE(X,Y)
PAUSE 'PRESS ENTER TO EXIT'
END
```

4.3.3 Використання сформованих даних для побудови графіків. Текстові файли із результатами розрахунків, що створюються програмами, можна використовувати в якості вихідних даних для інших програм, у тому числі для розповсюджених сьогодні програм обробки електронних таблиць з метою використання присутніх в них можливостей щодо обробки цих результатів, наприклад для їхньої візуалізації. В якості прикладу можливого використання текстових файлів із результатами розрахунків в інших програмах розглянемо далі варіанти використання текстового файлу із даними щодо графіків поліномів Чебишева, які створені розглянутою раніше програмою (див. ліст. 4.14) у сервісах Google (рис. 4.4), що відкриті до доступу через мережу Інтернет.

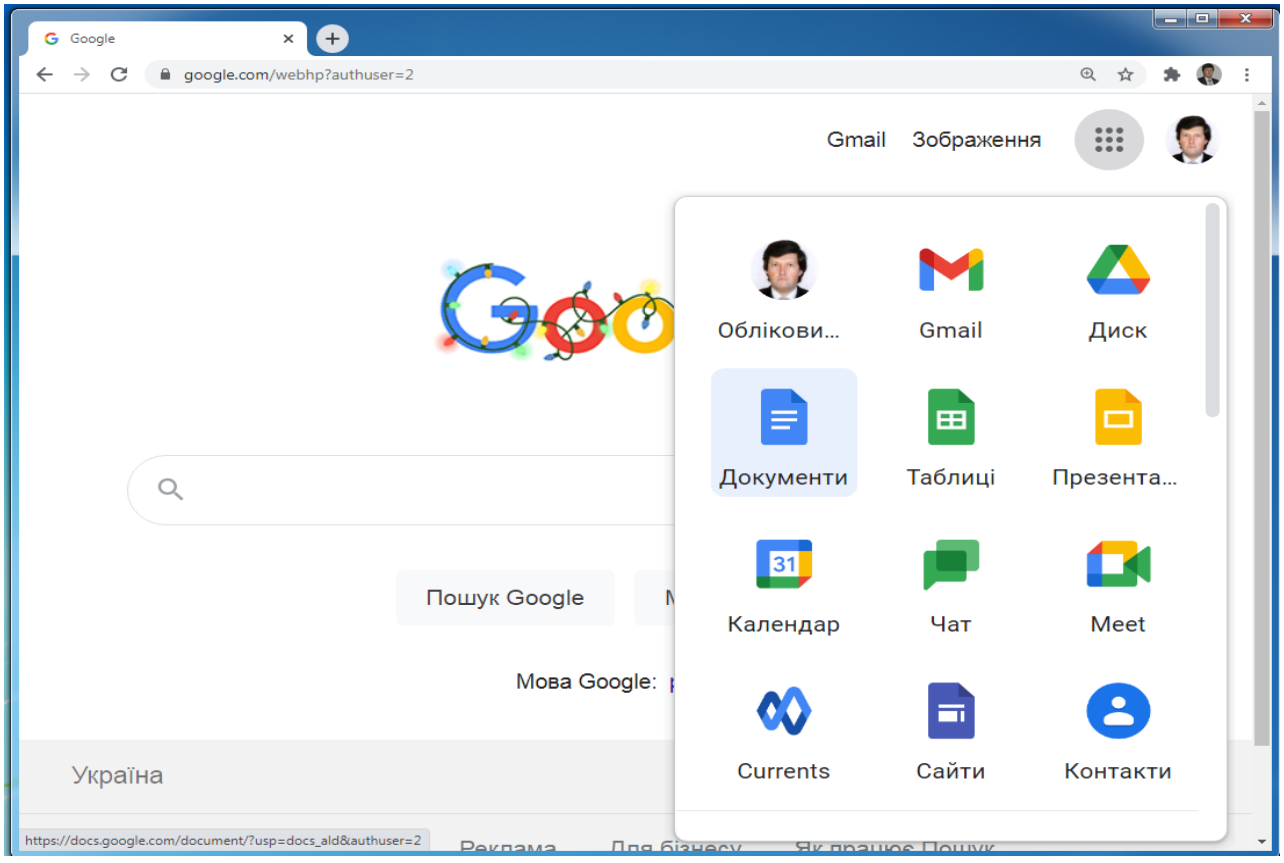


Рисунок 4.4 – Доступ до сервісу Google для роботи з документами

Сервіси Google надають великих можливостей щодо роботи з документами (рис. 4.4). Наприклад, спочатку можна відкрити засіб вибору файлів (рис. 4.5) та завантажити за його допомогою (рис. 4.6) створений програмою (див. ліст. 4.14) текстовий файл `'CHEBYSHEV.TXT'`. Далі у завантаженому текстовому файлі за допомогою відповідної команди (рис. 4.7; цю команду можна викликати у тому числі за допомогою комбінації клавіш CTRL+H) можна знайти усі символи «.» (крапка) та замінити їх на символ «,» (кома),

що потрібно для подальшого використання числових даних в Google таблицях (рис. 4.8). Слід створити нову таблицю та імпортувати (через меню файл) до неї текстовий файл `'CHEBYSHEV.TXT'`, що був попередньо виправленим та автоматично збереженим на Google диску; при імпортуванні цього текстового файлу (рис. 4.9) слід передбачити спеціальний тип роздільника як «;» (крапка з комою). Далі слід на листі таблиці виділити імпорттовані дані та вставити діаграму (рис. 4.10); результат набуде вигляду, як на рис. 4.11. Одержані результати повністю відповідають відомому вигляду поліномів Чебишева, що можна перевірити, наприклад, у довіднику [47]. Слід зазначити, що сервіси Google призначені для широкого кола користувачів і не є досить зручними для спеціалізованих наукових досліджень.

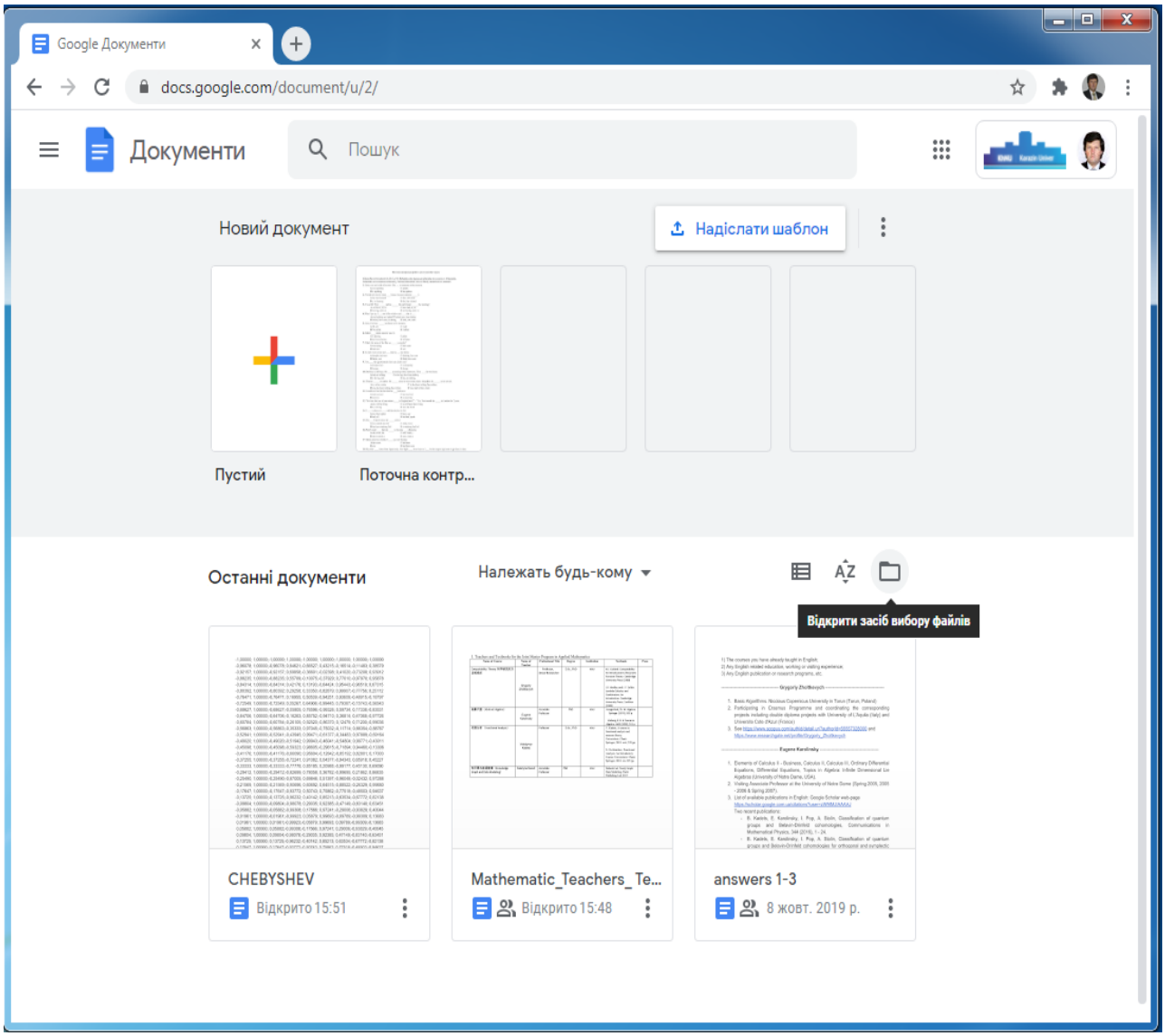


Рисунок 4.5 – Доступ до засобу вибору файлів

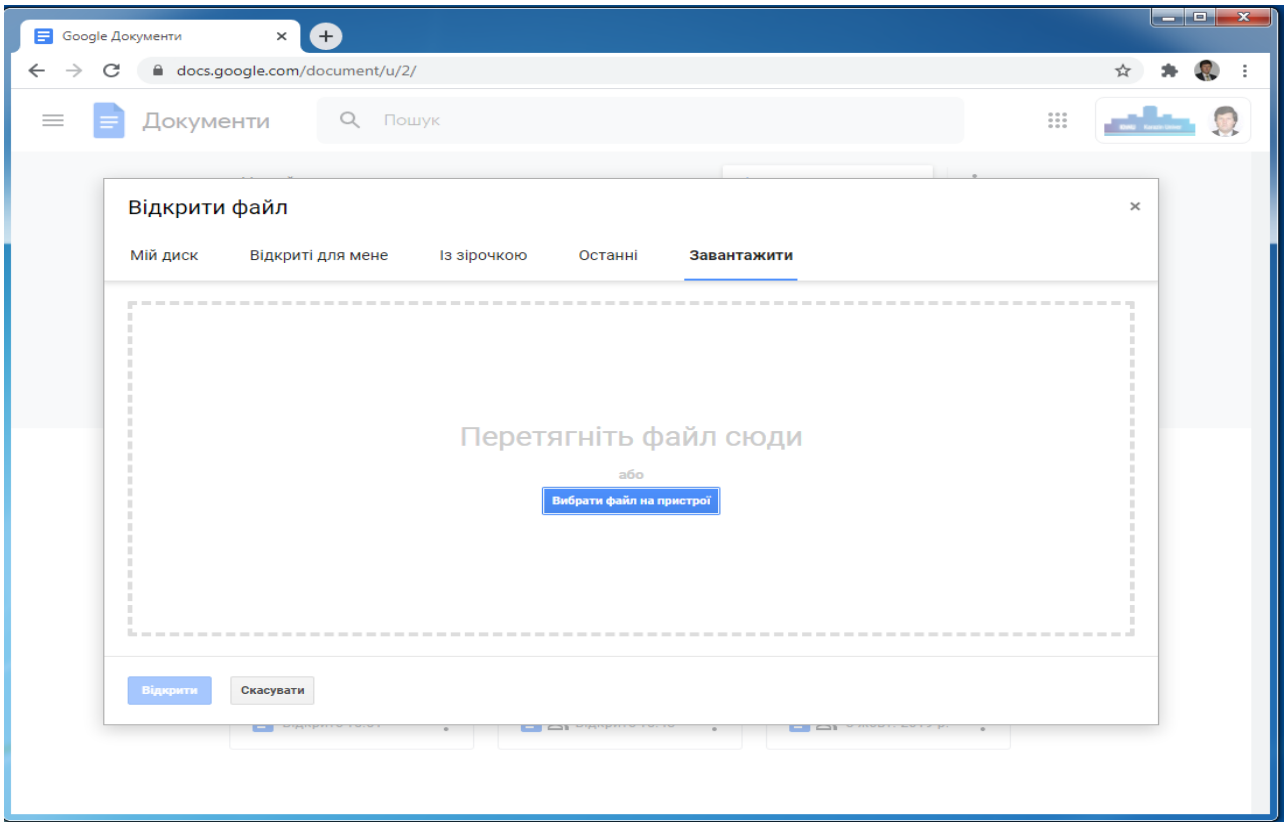


Рисунок 4.6 – Завантаження файлів з комп'ютера

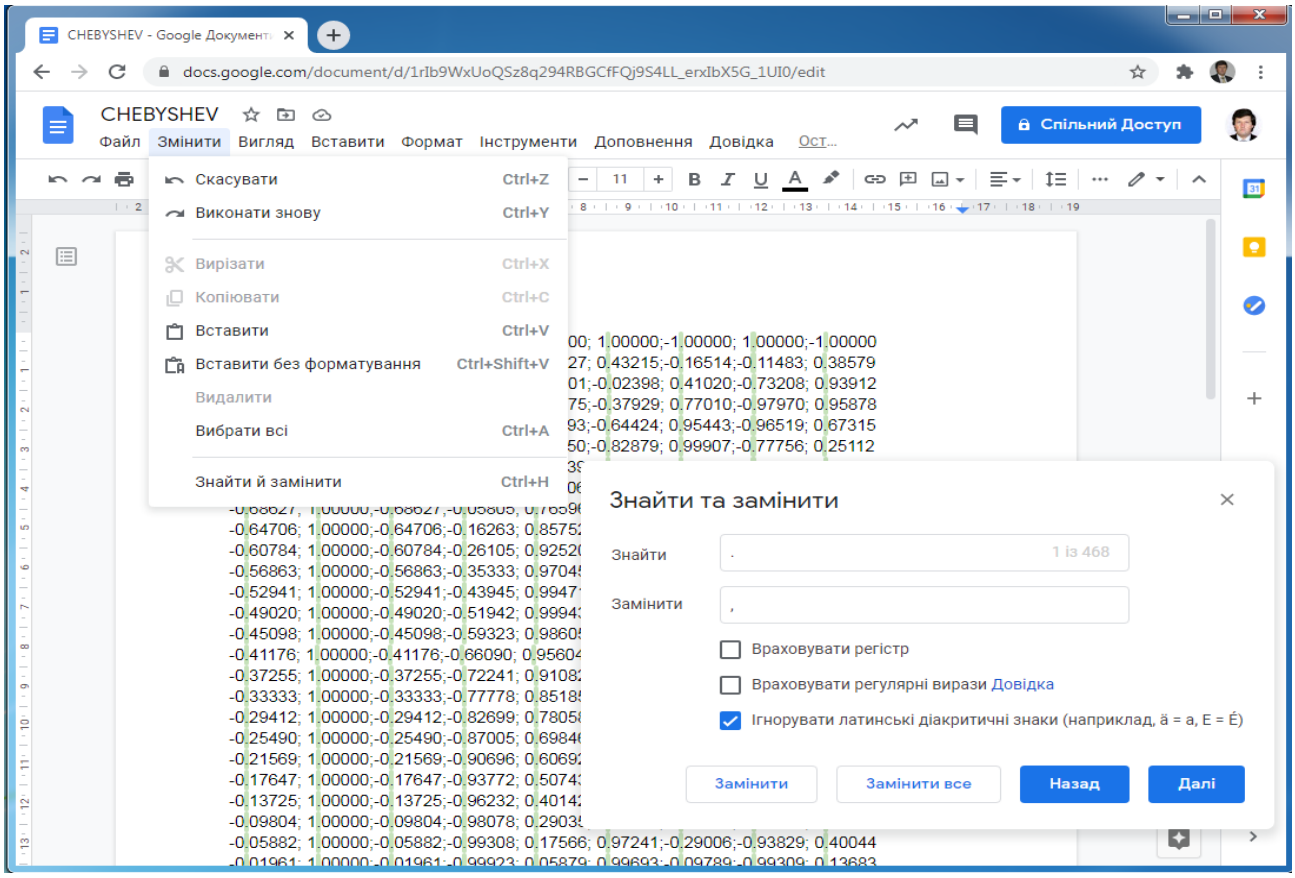


Рисунок 4.7 – Заміна символу в текстовому файлі

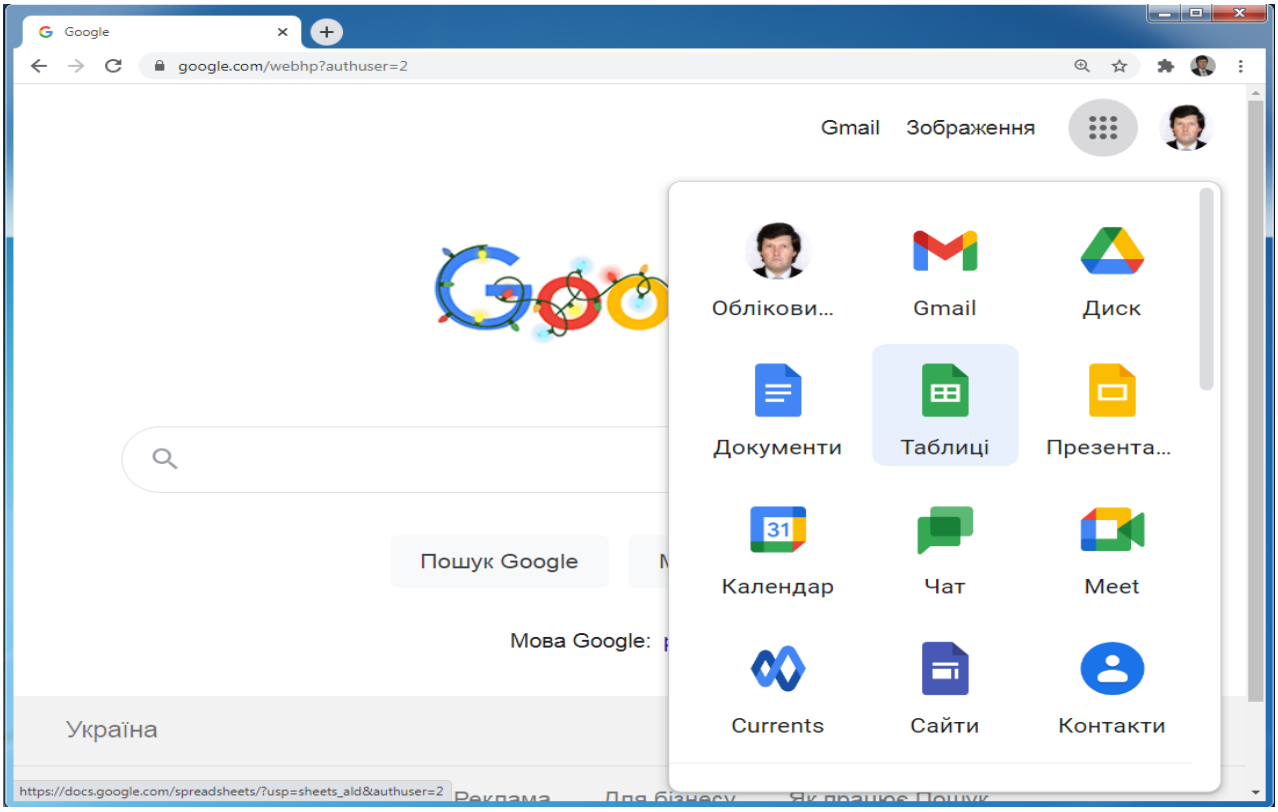


Рисунок 4.8 – Доступ до сервісу Google для роботи з таблицями

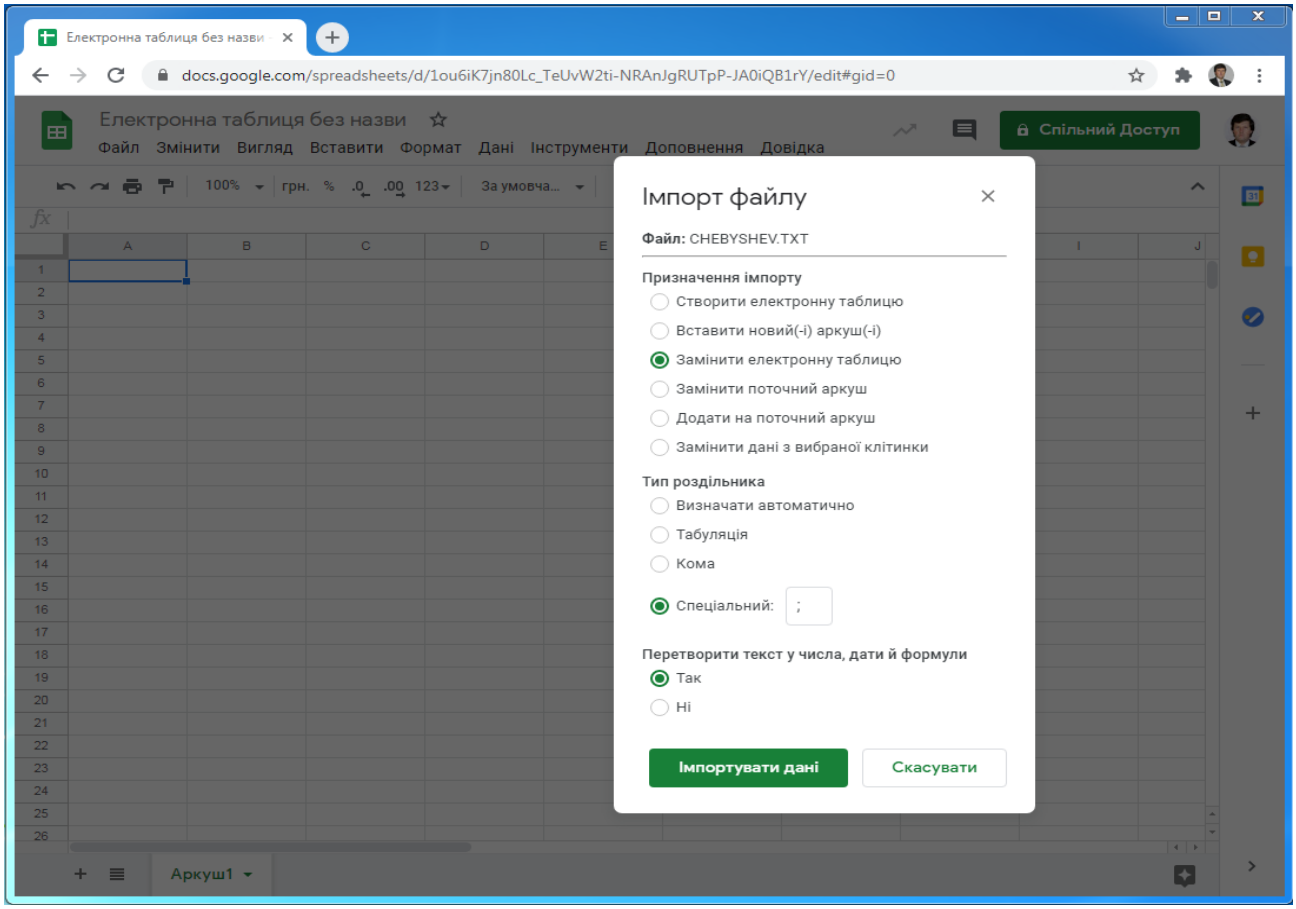


Рисунок 4.9 – Імпортування текстового файлу до Google таблиць

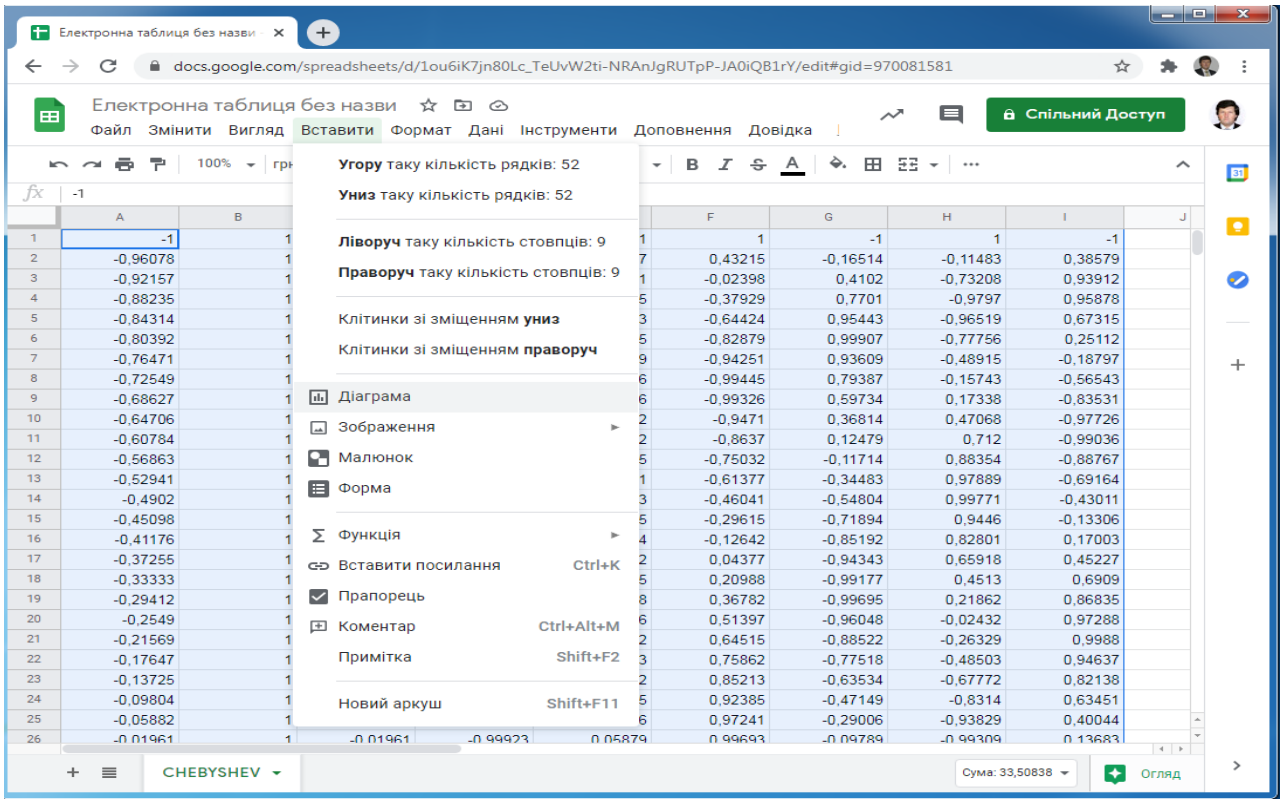


Рисунок 4.10 – Вставка діаграми в Google таблицю

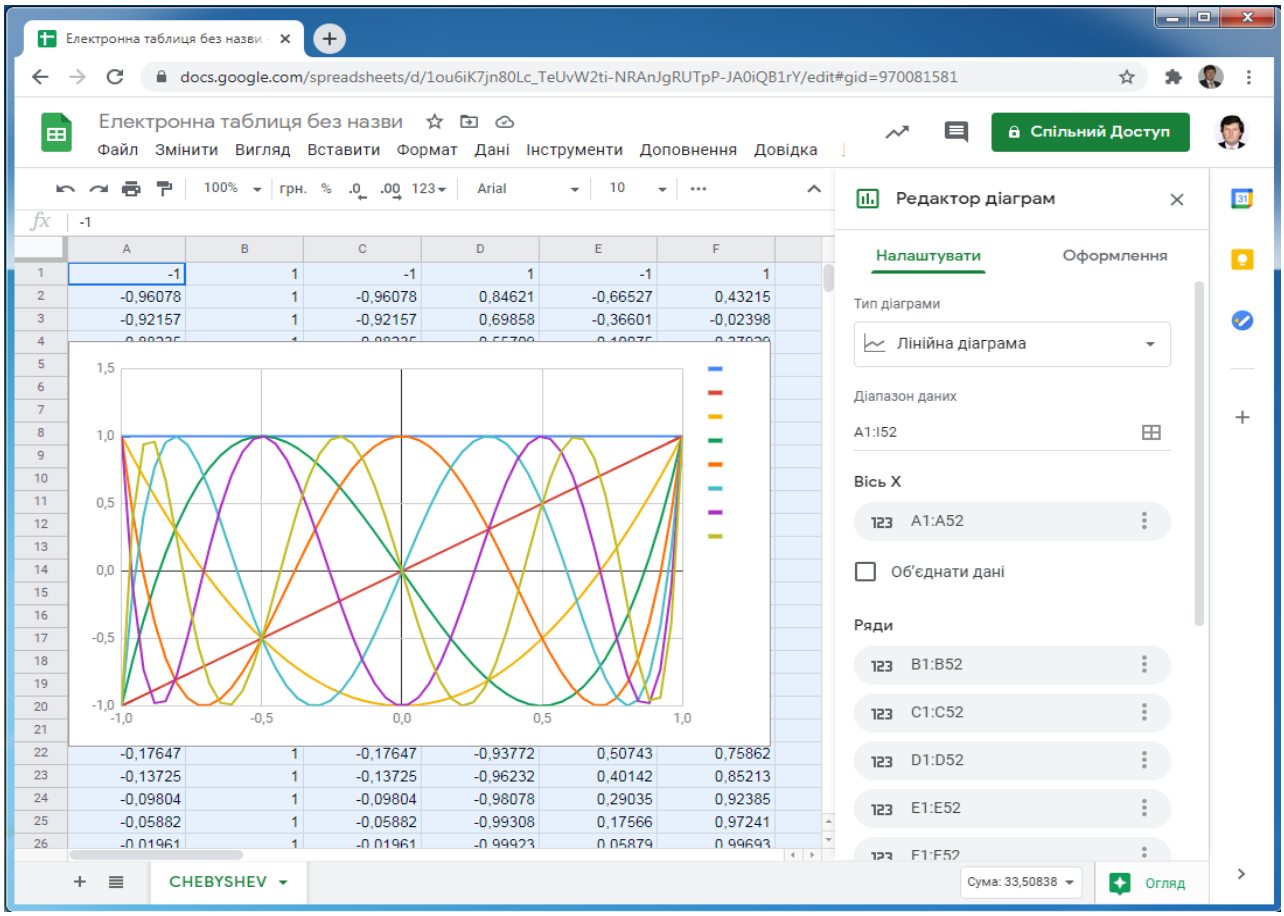


Рисунок 4.11 – Результат вставки діаграми в Google таблицю

4.4 Робота із файлами за допомогою підпрограм

При виконанні деяких наукових та інженерних розрахунків результати накопичуються протягом розрахунку та мають бути збережені у файл для подальшого використання. В таких випадках зручно мати набір підпрограм, що дозволяють автоматизувати процес запису результатів до текстового файлу. В якості прикладу розглянемо далі розв'язування задачі про інтегрування звичайного диференціального рівняння із початковою умовою, яке дозволяє моделювати різноманітні процеси, наприклад радіоактивний розпад ([49], додаток Ж), та широко використовується в інженерних та наукових розрахунках. Така задача формулюється наступним чином:

$$\frac{dx}{dt} = f(t,x), \quad x(0) = x_{(0)}, \quad t \geq 0, \tag{4.15}$$

де $x = x(t)$ – шукана функція; $f(t,x)$ – заданий закон швидкості шуканої функції; $x_{(0)}$ – задане (початкове) значення шуканої функції у заданій (початковій) точці.

Зрозуміло, що у загальному випадку для будь-якої функції $f(t,x)$ неможливо одержати розв'язок задачі (4.15) в аналітичній формі, хоча це можна зробити для окремих випадків функції $f(t,x)$. За цим, використання обчислювальних методів та розрахунки з використанням комп'ютерів є найбільш універсальним способом розв'язування задачі (4.15). Найбільш простим шляхом обчислювального розв'язування задачі (4.15) є метод Ейлера, який полягає у використанні наближеного виразу для похідної [23, 47]:

$$\frac{dx}{dt} = \lim_{\Delta t \to 0} \frac{x(t+\Delta t) - x(t)}{\Delta t} \Rightarrow \frac{dx}{dt} \approx \frac{x(t+\Delta t) - x(t)}{\Delta t}. \tag{4.16}$$

Наближене співвідношення (4.16) та диференціальне рівняння (4.15) дозволяють записати:

$$x(t+\Delta t) \approx x(t) + \frac{dx}{dt}\Delta t \Rightarrow x(t+\Delta t) \approx x(t) + f(t,x)\Delta t. \tag{4.17}$$

Наближене співвідношення (4.17) дозволяє запропонувати наступну схему розв'язування задачі (4.15): замість неперервної функції $x = x(t)$ визначатимемо її значення в заздалегідь обраних точках (рис. 4.4):

$$t = t_k, \quad t_k = k\Delta t, \quad k = 0,1,2,\ldots. \tag{4.18}$$

$$x_k \equiv x\left(t_k\right), \quad k = 0,1,2,\ldots, \tag{4.19}$$

де t_k – заздалегідь обрані точки, в яких має бути наближено визначена шукана функція $x = x\left(t\right)$; $\Delta t > 0$ – задана величина, яку називають крок інтегрування; x_k – значення шуканої функції у заданих точках, що мають бути визначеними для $k = 0,1,2,\ldots$

Для визначення значень (4.19) відповідно методу Ейлера використовуємо початкову умову та наближений вираз (4.17):

$$x_0 = x_{(0)}, \quad x\left(t_{k+1}\right) \approx x_k + f\left(t_k, x_k\right)\Delta t\,. \tag{4.20}$$

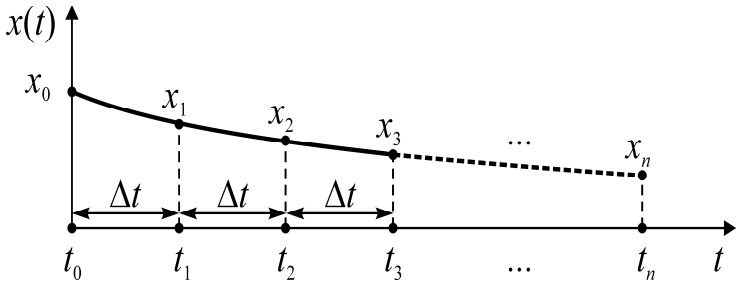


Рисунок 4.12 – Дискретне представлення розв'язку звичайного диференціального рівняння з початковою умовою

Зрозуміло, що похибка наближено знайдених значень (4.20) шуканої функції обумовлена наближеністю співвідношення (4.17) та залежатиме від кроку інтегрування Δt, який необхідно обирати достатньо малим. Таким чином, намагання розв'язати задачу (4.15) призвело до ітераційної схеми, в якій наступне значення (4.19) визначається за попереднім значенням, і при розв'язуванні цієї задачі необхідно зберігати у файл усі проміжні результати. Така схема є розповсюдженою щодо розв'язування задач вигляду (4.15), у тому числі для систем звичайних диференціальних рівнянь; різні покрокові методи розв'язування задач типу (4.15) відрізняються способом визначення наступного значення через попередні значення (4.19), а наведена схема (4.20) відповідає, як це було відзначено, методу Ейлера. За цих умов зручно мати набір підпрограм для запису результатів розв'язування задачі вигляду (4.15) та використовувати ці підпрограми в програмах розв'язування задачі вигляду (4.15) різним покроковими методами.

4.4.1 Створення підпрограм для роботи з файлами. Використання спеціальних заздалегідь створених підпрограм для роботи з файлами є зручним при виконанні складних розрахунків. Для роботи з файлами доцільно передбачити три типи підпрограм: відкриття файлу для запису, запис вихідних даних та результатів розрахунків до файлу, а також завершення роботи з файлом. Усі ці підпрограми зручно оформляти як функції логічного типу для забезпечення можливості щодо визначення помилок у процесі роботи з файлом та коректної роботи програми за умов наявності таких помилок.

У підпрограмі (ліст. 4.15) відкриття файлу для запису даних, у підпрограмах запису вихідних даних та результатів розрахунків (ліст. 4.16) та в підпрограмі завершення роботи із файлом (ліст. 4.17), що мають використовуватися при розв'язуванні задачі вигляду (4.15), можливість передбачити помилки при звертанні до файлу здійснюється за допомогою специфікації ERR повернення за помилкою, що передбачена в інструкції приєднання файлу до каналу та в інструкції запису (читання) даних до файлу. В цій специфікації вказується мітка інструкції, якій слід передати управління при виникненні помилки. Підпрограму (ліст. 4.17) завершення роботи із файлом, що має використовуватися при розв'язуванні задачі вигляду (4.15), зручно сумістити із записом кількості виконаних кроків інтегрування. Запис кількості кроків при завершенні розв'язування задачі обумовлений тим, що така кількість не завжди апріорі відома та може визначатися протягом розв'язування задачі, наприклад із додаткової умови.

Лістинг 4.15 – Підпрограма відкриття файлу для запису даних

```
      LOGICAL FUNCTION CPSAVE_OPEN(FILENAME,UNIT)
      CHARACTER*(*) FILENAME
      INTEGER UNIT
      CPSAVE_OPEN=.FALSE.
      OPEN(UNIT,FILE=FILENAME,ERR=100)
      CPSAVE_OPEN=.TRUE.
100   CONTINUE
      END
```

Лістинг 4.16 – Підпрограми запису вихідних даних та результатів до файлу

```
      LOGICAL FUNCTION CPSAVE_INPUTED(UNIT,DT,METHOD)
      CHARACTER*(*) METHOD
      INTEGER UNIT
      REAL DT
      CPSAVE_INPUTED=.FALSE.
      WRITE(UNIT,FMT='(A)',ERR=100)
     1 '**** CAUCHY PROBLEM SOLUTION ********'
      WRITE(UNIT,FMT='(''METHOD='',A)',ERR=100) METHOD
      WRITE(UNIT,FMT='(''DT='',E15.8)',ERR=100) DT
      WRITE(UNIT,FMT='(''T; X; DXDT'')',ERR=100)
      CPSAVE_INPUTED=.TRUE.
100   CONTINUE
      END
      LOGICAL FUNCTION CPSAVE_RESULT(UNIT,T,X,DXDT)
      INTEGER UNIT
      REAL T,X,DXDT
```

```
      CPSAVE_RESULT=.FALSE.
      WRITE(UNIT,
     1FMT='(E15.8,'';'',E15.8,'';'',E15.8)',ERR=100)
     1 T,X,DXDT
      CPSAVE_RESULT=.TRUE.
100   END
```

Лістинг 4.17 – Підпрограма завершення роботи із файлом результатів

```
      LOGICAL FUNCTION CPSAVE_CLOSE(UNIT,COUNTSTEPS)
      INTEGER UNIT,COUNTSTEPS
      CPSAVE_CLOSE=.FALSE.
      WRITE(UNIT,FMT='(I10.10)',ERR=100) COUNTSTEPS
      CPSAVE_CLOSE=.TRUE.
100   END
```

Наведений набір підпрограм (див. ліст. 4.15–4.17) може бути використаний при розв'язуванні задачі вигляду (4.15) будь-яким покроковим обчислювальним методом [23, 24, 47].

4.4.2 Використання підпрограм для збереження результатів у файл. Можливий варіант використання розглянутих раніше підпрограм роботи із файлами (див. ліст. 4.15–4.17) у підпрограмі розв'язування задачі (4.15) методом Ейлера (4.20) наведений на ліст. 4.18. Ця підпрограма оформлена у вигляді функції логічного типу, щоб повідомити про помилки роботи із файлом результатів. Вихідними даними для підпрограми (ліст. 4.18) розв'язування задачі вигляду (4.15) є початкові значення змінної інтегрування та шуканої функції; ім'я підпрограм обчислення швидкості шуканої функції, відповідної диференціальному рівнянню (4.15), а також визначення умови завершення інтегрування; значення кроку інтегрування та максимальна кількість кроків інтегрування; ім'я файлу для збереження результатів розрахунків.

Лістинг 4.18 – Підпрограма завершення роботи із файлом результатів

```
      LOGICAL FUNCTION CPSOLVE_EULER
     1(T0,X0,F,FINCOND,DT,COUNTSTEPS,RESFILE)
      REAL T0,X0,F,DT,X_K,DXDT_K,T_K
      LOGICAL FINCOND,CPSAVE_OPEN,CPSAVE_INPUTED,
     1CPSAVE_RESULT,CPSAVE_CLOSE
      INTEGER COUNTSTEPS
      CHARACTER*(*) RESFILE
      CPSOLVE_EULER=.TRUE.
      I=0
      CPSOLVE_EULER=CPSAVE_OPEN(RESFILE,1)
      IF (.NOT.(CPSOLVE_EULER)) GOTO 200
      CPSOLVE_EULER=CPSAVE_INPUTED(1,DT,'EULER')
```

```
      IF (.NOT.(CPSOLVE_EULER)) GOTO 100
      T_K=T0
      X_K=X0
      DXDT_K=F(T_K,X_K)
      CPSOLVE_EULER=CPSAVE_RESULT(1,T,X_K,DXDT_K)
      IF (.NOT.(CPSOLVE_EULER)) GOTO 100
      DO I=1,COUNTSTEPS
      T_K=T_K+DT
      X_K=X_K+DXDT_K*DT
      DXDT_K=F(T_K,X_K)
      IF (FINCOND(T_K,X_K,DXDT_K)) GOTO 100
      CPSOLVE_EULER=CPSAVE_RESULT(1,T_K,X_K,DXDT_K)
      IF (.NOT.(CPSOLVE_EULER)) GOTO 100
      END DO
100   CPSOLVE_EULER=CPSOLVE_EULER.AND.CPSAVE_CLOSE(1,I)
200   CONTINUE
      END
```

4.4.3 Виконання розрахунків із збереженням результатів у файл. Як приклад розглянемо дослідження процесу радіоактивного розпаду, яке зводиться до розв'язування наступної задачі ([49], додаток Ж):

$$\frac{dm}{dt} = -\lambda m, \quad m(0) = m_{(0)}, \tag{4.21}$$

де m – маса радіоактивної речовини; λ – характеристика інтенсивності процесу радіоактивного розпаду речовини; $m_{(0)}$ – початкова маса радіоактивної речовини.

Задача (4.21) може бути розв'язана аналітично та має розв'язок наступного вигляду([49], додаток Ж):

$$m = m_{(0)} \mathrm{e}^{-\lambda t}. \tag{4.22}$$

Далі розглянемо, наскільки наближений розв'язок задачі (4.21) методом Ейлера (4.20) буде збігатися із точним розв'язком (4.22) для наступних вихідних даних:

$$m_{(0)} = 1 \text{ кг}, \quad \lambda = \frac{\ln 2}{100} \text{ рік}^{-1}, \quad \Delta T = 0{,}1 \text{ рік}. \tag{4.23}$$

Максимальну кількість кроків інтегрування обмежимо значенням $n = 2000$, але інтегрування будемо здійснювати, доки початкова маса радіоактивної речовини не зменшиться удвічі.

Програма, що реалізує розв'язування задачі (4.21) про радіоактивний розпад, показана на ліст. 4.19; зрозуміло, що відповідний проект має містити також розглянуті раніше підпрограми (див. ліст. 4.15–4.18). В наведеній програмі (ліст. 4.19) передбачено визначення наближеного розв'язку задачі (4.21) методом Ейлера (4.20) та обчислення аналітичного розв'язку (4.22) із тим самим кроком, щоб було можна порівняти наближений розв'язок із точним. Слід зазначити, що розглянуті вище підпрограми (див. рис. 4.15–4.17), які створювалися для використання в підпрограмах наближеного розв'язування задачі вигляду (4.15), виявилися більш універсальними, про що свідчить їхнє використання в підпрограмі побудови аналітичного розв'язку (ліст. 4.19). В результаті роботи програми (ліст.4.19) буде створено два текстові файли, що будуть містити дані наближеного та точного аналітичного розв'язку задачі (4.21). За допомогою програми для роботи з текстовими файлами можна переглянути та порівняти відповідні результати розрахунків.

Лістинг 4.19 – Комп'ютерне моделювання радіоактивного розпаду

```
      PROGRAM RADIODECAY
      EXTERNAL F,FINCOND
      REAL F
      LOGICAL FINCOND, CPSOLVE_EULER
      PRINT *,'BEGINNING NUMERICAL SOLVING'
      CALL CPSOLVE_EULER
     1(0.0,1.0,F,FINCOND,0.1,2000,'RD_N.TXT')
      PRINT *,'FINISHING NUMERICAL SOLVING'
      CALL ANALYTSOLUTION
      PAUSE 'PRSS ETER TO EXIT'
      END
      REAL FUNCTION F(T,X)
      REAL T,X
      F=-LOG(2.0)*X/100
      END
      LOGICAL FUNCTION FINCOND(T,X,DXDT)
      REAL T,X,DXDT
      FINCOND=X.LT.(0.5)
      END
      SUBROUTINE ANALYTSOLUTION
      LOGICAL RES,CPSAVE_OPEN,CPSAVE_INPUTED,
     1CPSAVE_RESULT,CPSAVE_CLOSE,FINCOND
```

```
      REAL F,T,X,DXDT
      PRINT *,'BEGINNING ANALYTICAL SOLVING'
      I=0
      RES=CPSAVE_OPEN('RD_AN.TXT',1)
      IF (.NOT.(RES)) GOTO 200
      RES=CPSAVE_INPUTED(1,DT,'ANALYTICAL')
      IF (.NOT.(RES)) GOTO 100
      T=0.0
      X=1.0
      DXDT=F(T,X)
      RES=CPSAVE_RESULT(1,T,X,DXDT)
      IF (.NOT.(RES)) GOTO 100
      DO I=1,2000
      T=T+0.1
      X=1.0*EXP(-LOG(2.0)*T/100)
      DXDT=F(T,X)
      IF (FINCOND(T,X,DXDT)) GOTO 100
      RES=CPSAVE_RESULT(1,T,X,DXDT)
      IF (.NOT.(RES)) GOTO 100
      END DO
100   RES=RES.AND.CPSAVE_CLOSE(1,I)
200   CONTINUE
      PRINT *,'FINISHING ANALYTICAL SOLVING'
      END
```

ВИСНОВКИ

При вивченні курсу програмування основна увага була спрямована на вивчення базових понять та здобуття практичних навичок щодо програмування типових задач. Вивчення цього курсу дозволяє зробити наступні висновки.

Програмування являє собою галузь знань, що охоплює питання щодо розробки та створення комп'ютерних програм різного призначення. Показано, що розробка комп'ютерної програми вимагає від програміста певних знань щодо галузі призначення цієї програми. Завдяки цьому з урахуванням різноманіття галузей призначення програм досить складно підготувати програміста, який здатний створювати програми різного призначення, тому програмісти переважно спеціалізуються на створенні певних окремих класів програм.

Програма являє собою запис алгоритму, який вона реалізує. Поняття про алгоритми та їхні властивості є математизованою галуззю знань, вивчення якої потребує наявності досить суттєвої математичної підготовки. Хоча володіння математичною теорію алгоритмів не є обов'язковим для програмістів, що створюють відносно нескладні програми, типові для сучасної ІТ індустрії України, але воно забезпечує більшу конкурентоспроможність на відповідному ринку праці за рахунок наявності певної компетенції, обумовленої фундаментальними знаннями.

Для запису програм використовують зазвичай мови програмування. Одну програму можна записати різними мовами програмування. Мови програмування бувають універсальними та спеціалізованими; універсальні мови програмування відповідно до їхніх властивостей характеризуються поширеним використанням для розробки окремих класів програм. Мова програмування FORTRAN є науково орієнтованою та широко використовується при виконанні наукових та інженерних розрахунків, наприклад для виконання розрахунків на міцність конструкцій енергетичних установок, моделювання процесів тепло-масообміну та інших подібних задач. Ця мова програмування є найстарішою, оскільки первинним призначенням комп'ютерів було саме виконання наукових та інженерних розрахунків. Різноманіття мов програмування пов'язано із розширенням галузі використання комп'ютерів, які використовуються сьогодні не тільки для виконання наукових та інженерних розрахунків, але і для розв'язування нескладних побутових завдань та бізнес-задач, наприклад для придбання залізничних та авіаційних білетів, Інтернет-банкінгу та інших. Слід підкреслити, сучасна ІТ індустрія України спрямована на створення програмного забезпечення саме для розв'язування

побутових завдань та бізнес-задач, а не на виконання наукових та інженерних розрахунків.

Оскільки більш нові та сучасні мови програмування засновані на досвіді використання мови програмування FORTRAN, тому вивчення основ такої мови є доцільним у першу чергу для формування базових знань, необхідних для вивчення інших мов програмування. В той же час, володіння мовою програмування FORTRAN може бути безпосередньо корисним у випадку працевлаштування в науково-дослідницьких організаціях та установах, у тому числі та у першу чергу за кордоном, тому що ця мова програмування постійно розвивається та сьогодні широко використовується у європейських країнах та в США при виконанні наукових та інженерних розрахунків. Зрозуміло, що забезпечення широких можливостей працевлаштування вимагає володіння декількома мовами програмування, що виходить за межі даного навчального курсу і буде здійснюватися у подальшому навчанні при вивченні інших навчальних курсів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Каган Б.М. Электронные вычислительные машины и системы : учеб. пособие для вузов. – 3-е изд., перераб. и доп. – М. : Энергоатомиздат, 1991. – 592 с.

2. Фокс Д. Программное обеспечение и его разработка: математич. обеспеч. ЭВМ / пер. с англ. – М. : Мир, 1985. - 368 с.

3. Квиттнер П. Задачи, программы, вычисления / пер. М. М. Горбунов-Посадов, Д. А. Корягин, В. В. Красотченко ; ред. В. В. Мартынюк. - Москва : Мир, 1980. - 422 с.

4. Фортран: Программированное учебное пособие / под ред. чл.-кор. АН УССР Е.Л. Ющенко. – Киев : Вища школа, 1980. – 400 с.

5. Глушков В.М. Введение в кибернетику. – Киев : АН УССР, 1964. – 324 с.

6. Браух В. Программирование на Фортране 77 для инженеров / пер. с нем. – М. : Мир, 1987. – 200 с.

7. Chapman S.J. Fortran for Scientists and Engineers. – 4th ed. – New York : McGrawHill Education, 2018. – 1024 p.

8. Markus A. Modern Fortran in Practice. – New York : Cambridge University Pres, 2012. – 253 p.

9. Катцан Г. Язык Фортран 77 / пер. с англ. – М. : Мир, 1982. – 208 с.

10. http://openwatcom.org

11. https://en.wikipedia.org/wiki/Watcom

12. Денисов В. П., Драгунов Ю. Г. Реакторные установки ВВЭР для атомных электростанций. – Москва : ИздАТ, 2002. – 480 с.

13. Нормы расчета на прочность элементов оборудования и трубопроводов атомных энергетических установок (ПНАЭ Г-7-002-86) / Госатомэнергонадзор СССР. – М. : Энергоатомиздат, 1989. – 525 с.

14. Рыбаков Ф.И. Системы эффективного взаимодействия человека и ЭВМ. – М. : Радио и связь, 1985. – 200 с.

15. Кузнецов И.П. Кибернетические диалоговые системы. – М. : Наука, 1976. – 293 с.

16. Open Watcom FORTRAN 77. Language Reference. – Sybase, Inc, 2002–2008. – 300 p.

17. Baase S., Van Gelder A. Computer algorithms. Introduction to Design and Analysis. – 3rd ed. – New York: Addison-Wesley Longman, 2000. – 688 p.

18. Ершов А.П. Теория программирования и вычислительные системы. – Москва : Знание, 1972. – 64 с.

19. ГОСТ 19.701-90. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения / Единая система программной документации. – Москва : Стандартинформ, 2005. – 128 с. – С. 93–114.

20. Дал У., Дейксрта Э., Хоор К. Структурное программирование. – Москва : Мир, 1975. – 247 с.

21. Турский В. Методология программирования. – Москва : Мир, 1981. – 264 с.

22. Хьюз Дж., Митчом Дж. Структурный поход к программированию. – М. : Мир, 1980. – 276 с.

23. Калиткин Н.Н. Численные методы. – М. : Наука, 1978. – 512 с.

24. Мудров А.Е. Численные методы для ПЭВМ на языках Бейсик, Фортран и Паскаль. – Томск : МП «РАСКО», 1991. – 272 с.

25. Баррон Д. Введение в языки программирования. – М. : Мир, 1980. – 192 с.

26. Хьюз Ч., Пфлигер Ч., Роуз Л. Методы программирования: курс на основе Фортрана. – М. : Мир, 1981. – 336 с.

27. Кнут Д.Э. Искусство программирования. Том 1. Основные алгоритмы. – Москва : Вильямс, 2002. – 720 с.

28. Шилов Г.Е. Математический анализ. Функции одного переменного. Ч. 1, 2. – М. : Наука, 1969. – 528 с.

29. Ашкрофт Дж., Элдридж Р., Полсон Р., Уилсон Г. Программирование на Фортране 77. – М. : Радио и связь, 1990. – 272 с.

30. Меткалф М., Рид Дж. Описание языка программирования ФОРТРАН 90. – Москва : Мир, 1995. – 302 с.

31. Активные зоны ВВЭР для атомных электростанций / [Шмелев В. Д., Драгунов Ю. Г., Денисов В. П., Васильченко И. Н.]. – М. : Академкнига, 2004. – 220 с.

32. Romashov Yu., Chibisov D. Approximate estimates of the temperature state of ceramic nuclear fuel in cylindrical fuel elements and the influence of processes and parameters of a nuclear reactor core // Вісник НТУ «ХПІ» Тем. вип.: Енергетичні та теплотехнічні процеси та устаткування. – Харків : НТУ «ХПІ», 2019. – № 2. – С. 28–32.

33. Izonin I., Nevliudov I. and Romashov Yu. Computational Models and Methods for Automated Risks Assessments in Deterministic Stationary Systems // Proceedings of the 1st International Workshop on Computational & Information Technologies for Risk-Informed Systems (CITRisk 2020). – Kherson. – 2020. – P. 27–43.

34. Лыков А. В. Теория теплопроводности. – М. : Высш. шк., 1967. – 600 с.

35. Беклемишев Д.В. Курс аналитической геометрии и линейной алгебры. – М. : Наука. Гл. ред. физ.-мат. лит., 1987. – 320 с.

36. Пономаренко В.С., Малярець Л.М., Бойко А.В., Любчик Л.М., Ляшенко О.І., Станжицький О.М., Черняк О.І., Христіановський В.В., Янцевич А.А., Афанасьєва Л.М., Макаренко О.І., Денисова Т.В., Лебедєва І.Л., Мацкул В.М., Плахотник В.В., Сенчуков В.Ф., Шинкарик М.І. Вища математика. – Харків : Фоліо, 2014. – 672 с.

37. Коробов В.И. Метод функции управляемости : [монография]. – Москв : Институт компьютерных исследований; Ижевск : Регулярная и хаотическая динамика, 2007. – 576 с.

38. Кай Шэнь, Неусыпин К. А. Исследование критериев степеней наблюдаемости, управляемости и идентифицируемости линейных динамических систем // Мехатроника, автоматизация, управление. – Т. 17. – № 11. – 2016. – С. 723–731.

39. Воронов А. А. Устойчивость, управляемость, наблюдаемость. – М. : Наука, 1979. – XX с.

40. Эйкхофф П. Основы идентификации систем управления. – М. : Мир, 1975. – 680 с.

41. Брайсон, Хо Ю Ши. Прикладная теория оптимального управления. – М. : Мир, 1972. – 544 с.

42. Форсайт Дж., Молер К. Численное решение систем линейных алгебраических уравнений. – М. : Мир. – 168 с.

43. Уилкинсон, Райнш. Справочник алгоритмов на языке АЛГОЛ. Линейная алгебра. – М.: Машиностроение, 1976. – 389 с.

44. Самарский А. А. Теория разностных схем. – 3-е изд., испр. – М. : Наука. Гл. ред. физ.-мат. лит., 1989. – 616 с.

45. Флетчер К. Вычислительные методы в динамике жидкостей: В 2-х томах: Т. 1 / пер. с англ. – М. : Мир, 1991. – 504 с.

46. Флетчер К. Вычислительные методы в динамике жидкостей: В 2-х томах: Т. 2 / пер. с англ. – М. : Мир, 1991. – 552 с.

47. Корн Г. Справочник по математике (для научных работников и инженеров) / Г. Корн, Т. Корн; Пер. с англ. – М. : Наука, 1977. – 832 с.

48. Флетчер К. Численные методы на основе метода Галеркина / пер. с англ. Л. В. Соколовской; ред. В. П. Шидловского. – М. : Мир, 1988. – 352 с.

49. Климов А.Н. Ядерная физика и ядерные реакторы : учебник для вузов. – 2-е изд., перераб. и доп. – Москва : Энергоатомиздат, 1985. – 352 с.

ДОДАТКИ

Додаток А

ОСНОВИ ТЕОРІЇ АЛГОРИТМІВ ТА ПРОГРАМ
(довідково-реферативний)

Теорія алгоритмів та програм є галуззю знань, що відноситься до математики [5] та будується шляхом введення спеціальних понять і визначень та формулювання співвідношень, що їх пов'язують. Розглянемо відповідно [5] базові математичні поняття та результати, необхідні для розуміння сутності та для формального визначення поняття про алгоритми та програми.

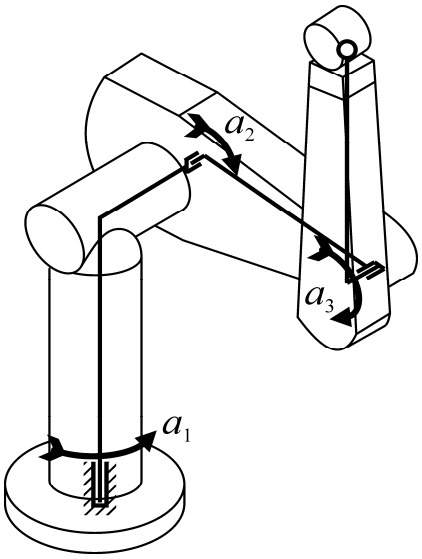


Рисунок А.1 – Маніпулятор та його ступені вільності

А.1 Абстрактним алфавітом називають будь-яку скінченну сукупність об'єктів, що називають буквами цього алфавіту [5]. Наприклад:

$$A = A(a_1, a_2, \ldots, a_n), B = B(b_1, b_2, \ldots, b_m), \qquad (А.1)$$

де A та B – алфавіти; $a_1, a_2, \ldots, a_n$ та $b_1, b_2, \ldots, b_m$ – букви алфавіту A та B відповідно.

Мовний алфавіт є прикладом окремого випадку абстрактного алфавіту, але насправді букви абстрактного алфавіту можуть мати різну природу, наприклад можуть відповідати ступеням вільності маніпулятора робото-технічного пристрою (рис. А.1) автоматизованої системи.

Словом в абстрактному алфавіті A називають будь-які упорядковані послідовності букв цього алфавіту [5], наприклад:

$$p(A) = a_3, a_1 a_n, a_1 a_3 a_6, a_1 a_3 a_2 a_6, \ldots, \qquad (А.2)$$

де $p(A)$ – слово в абстрактному алфавіті A.

Зміст слів визначається змістом абстрактного алфавіту; наприклад, у випадку, коли абстрактний алфавіт відповідає ступеням вільності (рис. А.1) маніпулятора, слова вигляду (А.2) такого алфавіту будуть відповідати складеним рухам цього маніпулятора.

А.2 Алфавітним оператором (або алфавітним відображенням) називають будь-яку відповідність (функцію), яка зіставляє словам в деякому

(вхідному для даного оператора) алфавіті слова в іншому (або в тому самому, вихідному для даного оператора) алфавіті [5]:

$$p(A) \to p(B). \tag{A.3}$$

А.2.1 Кодуванням називають алфавітний оператор, який зводиться до заміни букв вхідного алфавіту в словах їхніми кодами, які є відображеннями (кодуваннями) букв вхідного алфавіту у вихідному алфавіті:

$$a_i = b_{i1}, b_{i2} \ldots, b_{ik}, i = 1, 2, \ldots, n, \tag{A.4}$$

де a_i – буква вхідного алфавіту A (А.1) та $b_{ij}, j = 1, 2, \ldots, k$ – букви вихідного алфавіту B (А.1), що визначають відображення букви a_i.

Слово $p(B)$ у вихідному алфавіті B, яке відповідає слову $p(A)$ вхідного алфавіту A відповідно до кодування (А.4), називають [5] кодом цього слова $p(A)$. Зрозуміло, що сенс є у використанні оборотних кодувань, в яких усі слова $p(A)$ мають різні коди. Також зрозуміло, що оборотність кодування не забезпечується однією вимогою, щоб коди букв алфавіту A були різними, але забезпечується разом із вимогою, щоб коди усіх букв алфавіту A мали однакову довжину. [5]. Кодування, в якому коди букв вхідного алфавіту мають однакову довжину, називають нормальним [5]. Кодування дозволяє звести всі алфавітні відображення до алфавітних зображень в деякому одному заданому стандартному алфавіті, і такий стандартний алфавіт обирають із двох букв, що ототожнюють із цифрами 0 та 1 [5] (двійковий алфавіт). Якщо A – деякий алфавіт з n букв, а C – деякий стандартний алфавіт із l букв, то завжди можна обрати таке число k, що буде задовольняти умові:

$$l^k \geq n. \tag{A.5}$$

Завдяки нерівності (А.5) можемо закодувати усі букви алфавіту A словами довжини k в стандартному алфавіті C. Наприклад, нехай C є двійковий алфавіт, то $l = 2$, і, щоб закодувати алфавіт A, який містить три ($n = 3$) букви, які відповідають ступеням вільності маніпулятора (див. рис. А.1), потрібне число k, що задовольняє умові (А.5), буде наступним: $k = 2$. За цим, двійкові коди букв алфавіту A, що відповідні ступеням вільності маніпулятора (див. рис. А.1), можуть бути, наприклад, наступними:

$$a_1 = 00,\ a_2 = 01,\ a_3 = 10. \tag{A.6}$$

За допомогою коду (А.6) рух маніпулятора (див. рис. А.1), що складається із послідовного обертання навколо вертикальної осі, обертання навколо плеча та навколо ліктя, а потім у точно зворотному порядку, представиться так:

$$000110100100. \qquad (A.7)$$

А.2.2 Позначимо α нормальне кодування алфавіту A в стандартному алфавіті C; оборотне кодування C в A позначатимемо як α^{-1} (рис. А.2). Таким же чином позначимо β нормальне кодування алфавіту B в стандартному алфавіті C; оборотне кодування C в B позначатимемо як β^{-1} (рис. А.2). Розглянемо довільний алфавітний оператор φ із вхідним алфавітом A та вихідним алфавітом B (рис. А.2). Визначимо алфавітний оператор у вигляді послідовно виконуваних алфавітних операторів:

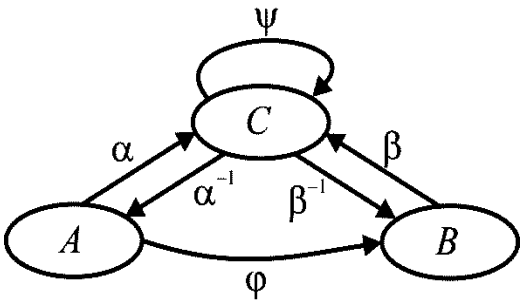


Рисунок А.2 – Представлення алфавітного оператора в стандартному алфавіті

$$\psi = \alpha^{-1}\varphi\beta. \qquad (A.8)$$

Зрозуміло, що алфавітний оператор (А.8) має вхідний та вихідний алфавіти, які збігаються із введеним стандартним алфавітом C (рис. А.2). З іншого боку, якщо маємо деякий алфавітний оператор ψ в стандартному алфавіті C, то можемо визначити відповідний йому алфавітний оператор, що діє із A в B (рис. А.2):

$$\varphi = \alpha\psi\beta^{-1}. \qquad (A.9)$$

Таким чином, будь-який алфавітний оператор можна звести до відповідного йому алфавітного оператора в стандартному алфавіті (рис. А.2).

А.3 Найбільш принциповою в теорії алгоритмів є проблема завдання довільного алфавітного оператора. Якщо область визначення алфавітного оператора є скінченною, то його завдання може бути зведене до побудови таблиці співвідношень між відповідними словами вхідного та вихідного алфавітів:

$$p(A) \to p(B): \begin{matrix} p_1(A) \to p_1(B) \\ p_2(A) \to p_2(B) \\ \vdots \\ p_N(A) \to p_N(B) \end{matrix}. \qquad (A.10)$$

У випадку нескінченної області визначення алфавітного оператора його завдання у вигляді таблиці співвідношень, зрозуміло, є принципово неможливим. В цьому випадку завдання алфавітного оператора може бути зведене до визначення правил, що дозволяють відобразити будь-яке слово вхідного алфавіту у відповідне йому слово вихідного алфавіту за скінченну кількість кроків такого відображення.

Алгоритмом називають сукупність скінченних систем правил, які визначають алфавітний оператор; тобто алгоритм є конструктивно заданою відповідністю між словами в деяких абстрактних алфавітах [5]. Якщо різні алгоритми визначають однакове алфавітне зображення, то такі алгоритми називають еквівалентними [5]. Можна розглянути задачу про загальні способи завдання алгоритмів, що дозволяють побудувати алгоритм, еквівалентний наперед заданому алгоритму; загальний спосіб завдання алгоритмів називають алгоритмічною системою [5]. Алгоритмічна система зазвичай містіть два типи об'єктів: елементарні оператори $O_1, O_2, \ldots$ (Operator) та елементарні розпізнавачі $R_1, R_2, \ldots$ (Recognizer) [5]. Елементарний оператор – це алфавітний оператор, який є простим для завдання, а елементарний розпізнавач визначає наявність певних заданих властивостей в словах. У загальному випадку завдання алгоритму зводиться до визначення послідовності елементарних операторів та елементарних розпізнавачів (рис. А.3).

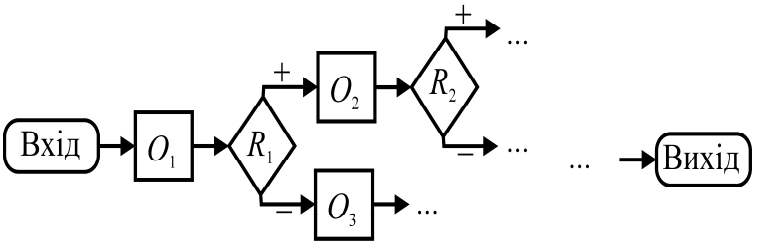


Рисунок А.3 – Загальне завдання алгоритму

Якщо деяке слово, що подане на вхід алгоритму (рис. А.3), потрапить через скінченне число кроків до вихідного вузла (рис. А.3), то кажуть, що такий алгоритм є застосовним до цього слова; якщо перетворення поданого на вхід алгоритму слова буде здійснюватися нескінченно довго, то кажуть, що такий алгоритм є незастосовним до цього слова [5].

А.3.1 В нормальних алгоритмах використовується тільки один тип оператора – оператор підстановки та один тип розпізнавача – розпізнавач входження [5].

Кажуть, що слово q міститься у заданому слові p, якщо слово p може бути представлено у вигляді $p = p_1 q p_2$, де p_1 та p_2 – деякі слова; при цьому якщо слово q не міститься в слові p_1, то входження слова q у слово p називають першим ліворуч [5]. Розпізнавач входження – це такий розпізнавач, який перевіряє, чи входить задане слово q у задане слово p [5]. Оператор підстановки – це алфавітний оператор, що в даному слові p

замінює словом q_2 задану замість першого ліворуч входження заданого слова q_1; оператор підстановки задається шляхом визначення букв q_1 та q_2, що розділені стрілкою: $q_1 \to q_2$ [5]. Алгоритми, що складаються виключно із операторів підстановки та розпізнавачів входження, називають узагальненими нормальними алгоритмами [5]. Нормальними алгоритмами називають такі узагальнені нормальні алгоритми, в яких результат кожного оператора підстановки $q_1 \to q_2$ перевіряється розпізнавачем входженьня слова q_1 та цей оператор підстановки виконується, доки це слово q_1 буде міститися в результаті дії цього оператора. Нормальний алгоритм є послідовністю операторів підстановки та розпізнавачів входження, як показано на рис. А.4.

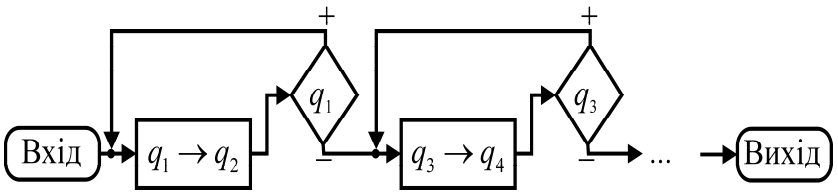


Рисунок А.4 – Нормальний алгоритм

Принцип нормалізації [5] полягає в тому, що для будь-якого алгоритму у довільному скінченному алфавіті можна побудувати еквівалентний цьому алгоритму нормальний алгоритм, тобто усі алгоритми можуть бути нормалізованими. Цей принцип підтверджується тим, що усі відомі сьогодні алгоритми є нормалізованими; були спеціальні намагання щодо побудови алгоритмів більш загального вигляду, але результати не вивели за межі класу нормальних алгоритмів [5]. Можливо, що у майбутньому вдасться запропонувати алгоритм, що не може бути нормалізованим, але нормалізовані алгоритми охоплюють значну частину алгоритмів, тому систему нормальних алгоритмів можна вважати універсальною алгоритмічною системою [5].

А.3.2 Кодування у стандартному, наприклад у двійковому, алфавіті може бути використане не тільки для кодування слів, але і для кодування оператора підстановки та розпізнавача входження, позначки початку та завершення алгоритму (рис. А.4). За допомогою такого кодування можемо записати схему алгоритму у вигляді слова в прийнятому стандартному алфавіті, яке називають зображенням даного алгоритму [5]. Була доведена теорема про універсальний нормальний алгоритм (А.А. Марков), із якої випливає можливість побудови машини, що може виконувати будь-який нормальний алгоритм, а з урахуванням принципу нормалізації – будь-який алгоритм; для цього зазначена машина має містити програму – зображення того нормального алгоритму, що має бути виконаним [5].

Слід розуміти, що нормалізація заданого алгоритму, тобто побудова еквівалентного йому нормального алгоритму, є досить складною задачею, тобто програмування машини, що здійснює безпосередньо універсальний нормальний алгоритм, є дуже складним та непрактичним. Машини, що

дозволяють виконати будь-який алгоритм, практично будуються на спеціальних алгоритмічних системах, що відрізняються від прийнятої для нормальних алгоритмів [5]. Існує багато можливих алгоритмічних систем, але щодо програмування комп'ютерів (цифрових обчислювальних машин) доцільно коротко повідомити про алгоритмічну систему Поста [5]. В алгоритмічній системі Поста використовується двійковий алфавіт та не скінченна інформаційна стрічка, що розділена на комірки, в кожній із яких можна розмістити одну букву (або 0 або 1); ті комірки, в яких записано 0, називають невідміченими, а в яких записано 1 – відповідно відміченими, причому відміченими можуть бути скінченна кількість комірок [5] (рис. А.5). Робота алгоритму здійснюється окремими (дискретними) кроками, на кожному з яких виконується один із наказів, що складає алгоритм; кожному кроку відповідає активна комірка, для першого наказу фіксується в якості активної деяка початкова комірка [5]. В алгоритмічній системі Поста накази, що складають алгоритм, можуть належати до наступних шести типів:

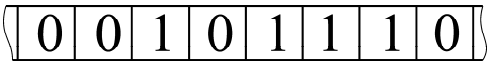


Рисунок А.5 – Фрагмент нескінченної стрічки Поста

1) відмітити актив ну комірку та перейти до заданого іншого наказу;

2) стерти відмітку активної комірки та перейти до заданого іншого наказу;

3) змістити активну комірку на крок праворуч та перейти до заданого іншого наказу;

4) змістити активну комірку на крок ліворуч та перейти до заданого іншого наказу;

5) якщо активна комірка відмічена, то перейти до заданого наказу, а якщо активна комірка не відмічена, то перейти до іншого заданого наказу;

6) зупинка, завершення роботи алгоритму.

Можна довести [5], що клас усіх алгоритмів, які еквівалентні системі Поста, збігається із класом усіх нормалізованих алгоритмів.

Слід зазначити, що існують такі класи задач, для розв'язування яких неможливо запропонувати єдиний універсальний алгоритм; проблеми розв'язування таких задач називають алгоритмічно нерозв'язними проблемами [5]. Прикладом алгоритмічно нерозв'язної проблеми є доведення тотожностей у звичайній алгебрі [5].

А.4 Комп'ютери (електронно-обчислювальні машини) є універсальними перетворювачами інформації і дозволяють автоматизувати майже усі галузі діяльності людини, хоча створювалися спочатку виключно для виконання великих обсягів розрахунків [5].

А.4.1 Електронна будова комп'ютерів має велике значення з точки зору характеристик (розміри, швидкість, об'єм пам'яті), але принциповим для забезпечення властивостей комп'ютера як універсального перетворю-

вача інформації є спеціальний принцип управління та набір можливих операцій [5]. Принцип управління, що забезпечує алгоритмічну універсальність комп'ютерів, заснований на розвитку та узагальненні алгоритмічної системи Поста [5].

Система Поста передбачає наявність нескінченної кількості комірок інформаційної стрічки, але в технічних пристроях сучасних комп'ютерів об'єм пам'яті принципово є обмеженим. За цих умов для забезпечення універсальності комп'ютерів в них передбачається два види пам'яті: швидкодіюча, але істотно обмежена в об'ємі внутрішня (оперативна) пам'ять, та повільна, але з великими обсягами, зовнішня пам'ять, що реалізується у вигляді спеціальних пристроїв, наприклад зовнішніх накопичувачів (вінчестерів). Комп'ютер є окремим випадком програмного автомату – пристрою, що має пам'ять у вигляді окремих комірок, робота якого управляється послідовністю командних слів (наказів), що розташуються в деяких із таких комірок; вказана послідовність наказів називається програмою [5]. За умов наявності необхідного набору команд, що дозволяє програмувати будь-який алгоритм, програмний автомат називають універсальним; комп'ютер є універсальним програмним автоматом, оскільки на ньому можна запрограмувати будь-який алгоритм за умов відсутності обмежень на об'єм пам'яті [5].

А.4.2 Програмуванням називають запис алгоритму у вигляді скінченної послідовності наказів (команд) комп'ютера [5], точніше процесора комп'ютера. Побудова програми шляхом безпосереднього кодування наказів універсального програмного автомату, тобто процесора комп'ютера, є дуже складною і практично можлива та здійснюється лише для окремих класів програм типу драйверів пристроїв тощо. У зв'язку із цим більш економний спосіб запису програм полягає у використанні самого комп'ютера для перетворення інформації про алгоритм у зручній формі у форму команд процесора [5]. Такий спосіб [5] складає основу автоматичного програмування з використанням універсальних програмуючих програм – трансляторів (компіляторів).

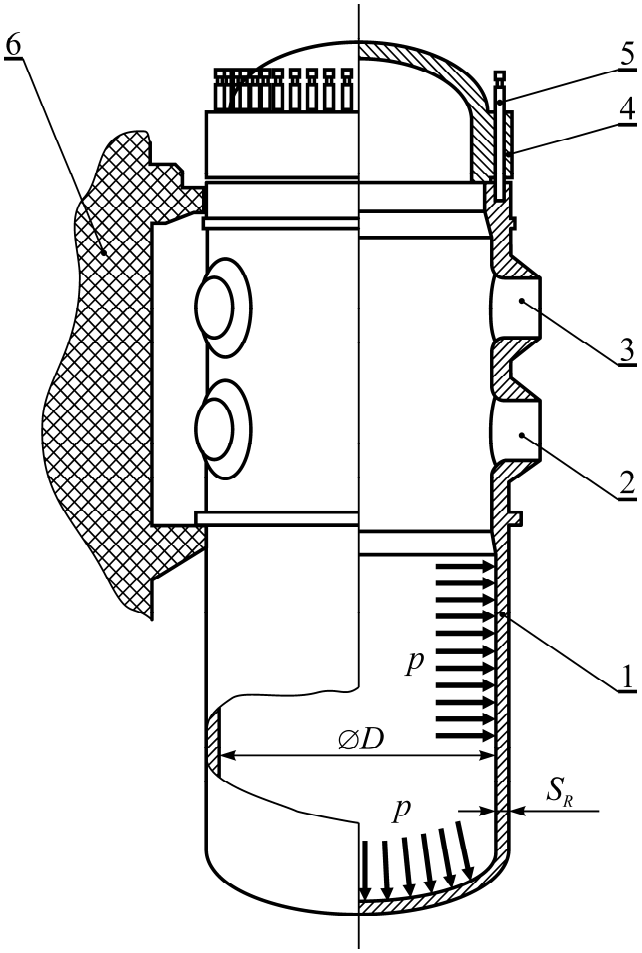
Компілятор являє собою запрограмований алгоритм перетворення програми, що записана формальною алгоритмічною мовою програмування, на мову команд процесора комп'ютера [5]. В якості формальної алгоритмічної мови програмування можна запропонувати мову схем нормальних алгоритмів, але це не полегшить програмування [5]. Тому були створені такі алгоритмічні мови програмування, що допускають більш просту для читання людиною-програмістом форму запису алгоритмів [5]; такими мовами програмування є FORTRAN, C, PASCAL, PYTHON та інші мови програмування.

Додаток Б

ВІДОМОСТІ ПРО ІНЖЕНЕРНИЙ РОЗРАХУНОК НА МІЦНІСТЬ КОРПУСІВ ЯДЕРНИХ РЕАКТОРІВ

Ядерний реактор являє собою пристрій, що призначений для організації та підтримання протягом часу керованої ланцюгової реакції ділення ядер. Ядерні реактори використовують в енергетиці для вироблення енергії на атомних електростанціях. Сьогодні на території України в стані експлуатації знаходяться 15 ядерних реакторів, які у 2019 р. забезпечували виробництво майже 54 % електроенергії, що відповідає 2-му місцю у світі після Франції, де цей показник становить майже 71 %, та при цьому значно випереджає за цим показником такі країни, як Росія та США (майже 20 %).

Б.1 Сьогодні відомо про експлуатацію 6-ти типів ядерних реакторів у різних країнах світу, але найбільш розповсюдженими (68 % загальної кількості діючих реакторів) є корпусні ядерні реактори з водою під тиском (PWR – Pressurized Water Reactor) в якості теплоносія та уповільнювача нейтронів; лише такими реакторами оснащені усі АЕС України. В ядерних реакторах PWR ядерне паливо, в якому має відбуватися ядерна реакція ділення, розташоване у герметичному корпусі (рис. Б.1); тепло, яке виділяється

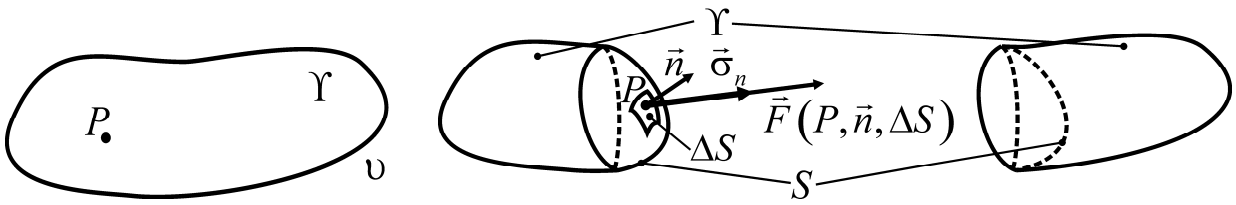


1 – корпус реактора;
2 – патрубок подачі теплоносія;
3 – патрубок виходу теплоносія;
4 – кришка;
5 – трипільний елемент кришки (шпилька);
6 – будівельні конструкції, що утримують реактор;
p – тиск теплоносія;
D – внутрішній діаметр корпусу;
S_R – товщина стінки корпусу

Рисунок Б.1 – Типова будова корпусного ядерного реактора

в ядерному паливі внаслідок реакції ділення, відводиться за допомогою теплоносія – води, яка прокачується скрізь паливо. Щоб уникнути кипіння водного теплоносія, його тиск p має бути досить великим; в реакторах ВВЕР тиск теплоносія $p \approx 15$ МПа [12]. Цілком зрозуміло, що досить високий тиск p водного теплоносія може зруйнувати корпус реактора (рис. Б.1), і, щоб уникнути такого руйнування для заданого тиску p, внутрішнього діаметру D та властивостей матеріалу корпусу реактора, слід визначити мінімально допустиму товщину S_R стінки (рис. Б.1), яка забезпечуватиме міцність корпусу при експлуатаційних навантаженнях.

Б.2 Визначення міцності конструкцій та деталей машин взагалі та зокрема міцності корпусу ядерного реактора суттєво використовує поняття механіки суцільних середовищ, у першу чергу саме поняття про суцільне середовище та про його внутрішні механічні напруження. Суцільне середовище – це така система, що складається із нескінченної кількості точок, які безперервно заповнюють деяку задану область – частину цього простору. В якості простору при розрахунку на міцність розглядають максимум трьохвимірний Евклідовий простір, відомий в геометрії; області в цьому просторі з геометричної точки зору – це звичайні геометричні фігури, що складаються із точок, у тому числі точки P, які утворюють у просторі область Υ з границею υ, як показано на рис. Б.2а. Хоча поняття



а) *б)*

а) геометричний образ; *б) внутрішнє механічне напруження*

Рисунок Б.2 – Суцільне середовище та внутрішнє механічне напруження

про суцільне середовище суперечить атомарній структурі речовин, але воно дозволяє врахувати можливість ототожнювати елементи конструкцій та деталі машин із певними геометричними фігурами, що є природним, оскільки вони безпосередньо спостерігаються такими, а атомарна структура речовин є недосяжною для безпосереднього спостереження. Міцність елементу конструкції, або деталі машини, що схематизується суцільним середовищем (рис. Б.2а), уявляється як обмеженість інтенсивності взаємодії між точками відповідного суцільного середовища. Для визначення інтенсивності взаємодії між точками суцільного середовища представимо (рис. Б.2б), що

суцільне середовище умовно розділено поверхнею S, яка проходить через точку P, для якої досліджується інтенсивність взаємодії з іншими точками; вектор $\vec{n}$ одиничної нормалі до поверхні S залежить від форми поверхні S. Навколо точки P розглянемо малу площадку ΔS на поверхні S та позначимо як $\vec{F}$ силу взаємодії. Вектором напруження $\vec{\sigma}_n$ в точці P на площадці із нормаллю $\vec{n}$ називаємо вектор, який визначається наступним чином:

$$\vec{\sigma}_n\left(P\right) = \lim_{\Delta S \to 0} \frac{\vec{F}\left(P, \vec{n}, \Delta S\right)}{\Delta S}. \qquad \text{(Б.1)}$$

Слід підкреслити, що існування межі (Б.1) приймається аксіоматично; вагомим припущенням є також те, що взаємодія площадки ΔS із іншими точками суцільного середовища зводитися тільки до сили (див. рис. Б.2б).

Зрозуміло, що напруження (Б.1) в точці суцільного середовища істотно залежить від орієнтації вектора нормалі площадки, що проходить через цю точку, тобто в точці суцільного середовища можна визначити нескінченну кількість напружень, які утворюють напружений стан в точці. Цікавою є та обставина, що для визначення напруженого стану в точці не є необхідним визначати усю нескінченну множину напружень в цій точці, а достатньо ввести координатну систему та визначити напруження лише на координатних площадках. У трьохвимірному просторі вектор визначається трьома компонентами та можна виділити три характерні координатні напрямки (площадки), тобто напружений стан визначається дев'ятьма скалярними параметрами, що представляють проекції векторів напруження на трьох координатних площадках. Це можна пояснити прикладом визначення напруженого стану циліндра (рис. Б.3), форма якого є досить близькою до циліндричної частини корпусу ядерного реактора (див. рис. Б.1). В цьому прикладі для визначення положення точки P замість прямокутних декартових координат xyz використані більш зручні циліндричні координати $r\theta z$. Отже, відповідно до таких циліндричних координат, в точці P маємо три координатні напрямки, відповідні радіальній r, окружній θ та осьовій z координатам (рис. Б.3). На кожній такій координатній площадці маємо вектор напруження, який може бути представлений трьома компонентами – проекціями на радіальний, окружний та осьовий координатні напрямки. Таким чином, напружений стан в точці циліндра може бути визначений дев'ятьма компонентами (рис. Б.3):

$$\begin{matrix} \sigma_{rr} & \sigma_{\theta r} & \sigma_{zr}, \\ \sigma_{r\theta} & \sigma_{\theta\theta} & \sigma_{z\theta}, \\ \sigma_{rz} & \sigma_{\theta z} & \sigma_{zz}. \end{matrix} \qquad \text{(Б.2)}$$

Компоненти напружень σ_{rr}, $\sigma_{\theta\theta}$ та σ_{zz} називають нормальними напруженнями, а усі інші компоненти – дотичними напруженнями; компоненту напружень σ_{rr} називають радіальними напруженнями, $\sigma_{\theta\theta}$ – окружними напруженнями, σ_{zz} – осьовими напруженнями. Можна показати, що компоненти напружень (Б.2) дозволяють визначити напруження на будь-якій площадці, що проходить через точку:

$$\begin{aligned} \sigma_{nr} &= \sigma_{rr} n_r + \sigma_{\theta r} n_\theta + \sigma_{zr} n_z, \\ \sigma_{n\theta} &= \sigma_{r\theta} n_r + \sigma_{\theta\theta} n_\theta + \sigma_{z\theta} n_z, \\ \sigma_{nz} &= \sigma_{rz} n_r + \sigma_{\theta z} n_\theta + \sigma_{zz} n_z, \end{aligned} \qquad \text{(Б.3)}$$

де σ_{nr}, $\sigma_{n\theta}$ та σ_{nz} – радіальна, окружна та осьова компоненти вектора (Б.1) напруження в точці P на площадці з нормаллю $\vec{n}$ (див. рис. Б.2б); n_r, n_θ та n_z – радіальна, окружна та осьова компоненти вектора $\vec{n}$ одиничної нормалі, що визначає площадку, на якій має бути визначене напруження в точці P.

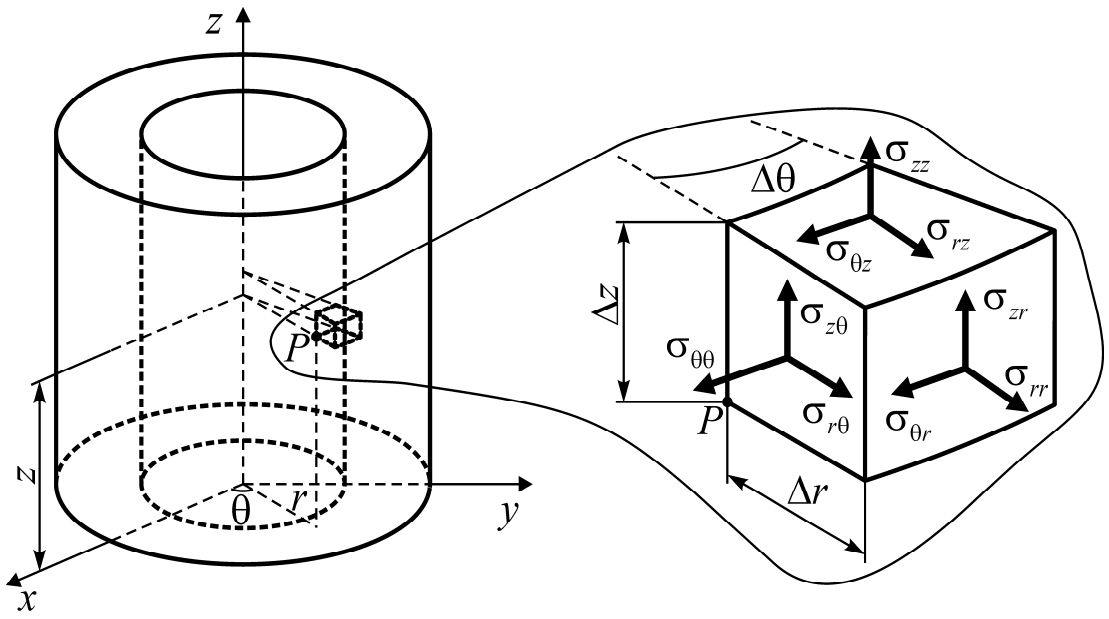


Рисунок Б.3 – Циліндр та репрезентативний об'єм навколо його точки

При розгляді прикладу щодо визначення напруженого стану циліндра (рис. Б.3) зрозуміло, що компоненти напружень істотно залежать від вибору координатної системи, але такий вибір є вільним, отже, для визначення напруженого стану можна використовувати різні координатні системи, і компоненти напружень на координатних площадках таких систем, зрозуміло, будуть різними. В той же час компоненти напружень в різних координатних системах пов'язані між собою таким чином, що компоненти напружень, визначені в одній координатній системі, можуть бути перераховані в компоненти напружень для іншої координатної системи, положення

якої задано відносно первинної координатної системи. Досить цікавим є можливість доведення того, що від вибору координатної системи для визначення напруженого стану не залежать корені наступного рівняння:

$$\begin{vmatrix} \sigma_{rr}-\sigma & \sigma_{\theta r} & \sigma_{zr} \\ \sigma_{r\theta} & \sigma_{\theta\theta}-\sigma & \sigma_{z\theta} \\ \sigma_{rz} & \sigma_{\theta z} & \sigma_{zz}-\sigma \end{vmatrix} = 0, \qquad (Б.4)$$

де σ – шуканий корінь рівняння; прямі дужки означають обчислення визначника відповідної матриці, про що можна дізнатися у додатку Д.

Слід підкреслити, що можливість визначати в координатних системах об'єкти, що не залежать від вибору координатних систем, є властивістю спеціальних математичних об'єктів – тензорів, тобто з математичної точки зору сукупність компонент напружень (Б.2) є тензором другого рангу, який в механіці суцільних середовищ називають тензором напружень. Важливою властивістю тензора напружень, що може бути доведена, є симетричність:

$$\sigma_{r\theta} = \sigma_{\theta r},\ \sigma_{rz} = \sigma_{zr},\ \sigma_{\theta z} = \sigma_{z\theta}. \qquad (Б.5)$$

Завдяки властивості (Б.5) симетрії тензора напружень усі три корені рівняння (Б.4) є дійсними числами, тобто їхні уявні частини дорівнюють нулю, що узгоджується із фізичним сенсом проблеми. Позначимо корені рівняння (Б.4) наступним чином:

$$\sigma_1 \geq \sigma_2 \geq \sigma_3, \qquad (Б.6)$$

де σ_1, σ_2 та σ_3 – корені рівняння (Б.4), які називають головними напруженнями.

Оскільки саме головні напруження (Б.6) насправді визначають напружений стан в точці суцільного середовища, що представляє деформівне тверде тіло, незалежно від вибору координатної системи для його розгляду, то умови міцності такого деформівного твердого тіла мають визначатися саме за допомогою цих головних напружень. Існує декілька теоретично та експериментально обґрунтованих теорій міцності, але для обґрунтування міцності корпусів ядерних реакторів використовується теорія максимального дотичного напруження, відповідно до якої умова міцності записується наступним чином:

$$\sigma_1 - \sigma_3 \leq [\sigma], \qquad (Б.7)$$

де $[\sigma]$ – допустиме напруження для матеріалу, яке визначається відповідно до норм розрахунку на міцність [13] за результатами випробувань зразків матеріалу на розрив при розтягуванні.

Б.3 Визначення напружено-деформованого стану корпусів ядерних реакторів у загальному випадку є досить складною задачею, але можливо одержати також наближені результати, придатні для виконання інженерних розрахунків. Дійсно, за умов симетрії навантаження маємо, що в циліндрі, який знаходиться під внутрішнім тиском (див. рис. Б.3а та рис. Б.4), дотичні напруження мають дорівнювати нулю:

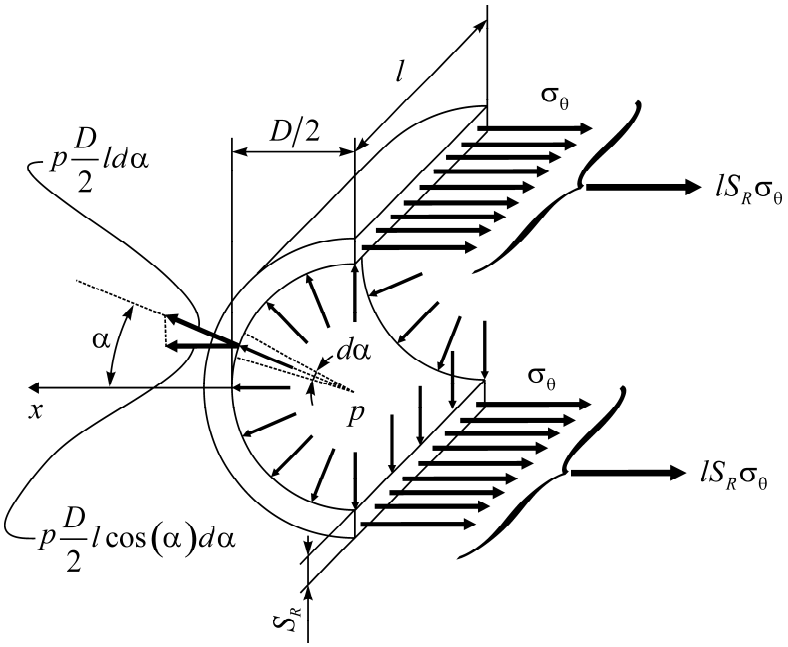


Рисунок Б.4 – Схематизація напруженого стану циліндра під дією внутрішнього тиску

$$\sigma_{r\theta}=0,\ \sigma_{rz}=0,\ \sigma_{\theta z}=0. \quad (Б.8)$$

Рівності (Б.7) разом із умовами (Б.5) значно спрощують розв'язування задачі про визначення напруженого стану циліндра під внутрішнім тиском, оскільки для цього залишається визначити лише нормальні напруження.

Щоб уникнути температурних напружень внаслідок теплового розширення, передбачають можливість вільного переміщення циліндра (корпусу реактора) уздовж поздовжньої осі z (див. рис. Б.3а), та завдяки цьому можна прийняти:

$$\sigma_{zz}=0. \quad (Б.9)$$

Радіальні напруження на внутрішньому радіусі циліндра визначаються тиском: $\sigma_{rr}=-p$, а на зовнішньому радіусі дорівнюють нулю: $\sigma_{rr}=0$, тому для радіального напруження в корпусі ядерного реактора можемо наближено прийняти середнє значення:

$$\sigma_{rr}=-p/2. \quad (Б.10)$$

Для наближеного визначення окружного напруження $\sigma_{\theta\theta}$ розглянемо умову рівноваги половини циліндра – рівність нулю сум проекцій усіх сил на вісь x (див. рис. Б.4):

$$l\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} p\frac{D}{2}\cos\alpha d\alpha - 2lS_R\sigma_{\theta\theta}=0, \quad (Б.11)$$

де l – довжина розрахункової ділянки циліндра (корпусу реактора) та α – кут між напрямом внутрішнього тиску та віссю x (див. рис. Б.4).

Після обчислення інтеграла умова (Б.11) дозволить визначити шукане окружне напруження у вигляді:

$$\sigma_{\theta\theta} = \frac{pD}{2S_R}. \qquad (Б.12)$$

Завдяки співвідношенням (Б.8)–(Б.10), (Б.12) маємо головні напруження (Б.6) у наступному вигляді:

$$\sigma_1 = \frac{pD}{2S_R},\ \sigma_2 = 0,\ \sigma_3 = -\frac{p}{2}. \qquad (Б.13)$$

Підставимо вирази (Б.13) до умови міцності (Б.7) та в результаті цього одержимо наступне:

$$\frac{pD}{2S_R} + \frac{p}{2} \le [\sigma]. \qquad (Б.14)$$

Далі залишається розв'язати нерівність (Б.14) відносно шуканої товщини S_R стінки циліндра, що дозволить одержати наступну умову міцності циліндричної частини корпусу ядерного реактора:

$$S_R \ge \frac{pD}{2[\sigma] - p}. \qquad (Б.15)$$

Одержана нерівність (Б.15) відповідна формулі (1.1), що використовувалася в прикладі щодо програмування простих розрахунків. Слід підкреслити, що відповідно до змісту нерівності (Б.15) маємо обмеження щодо вибору матеріалу для виготовлення циліндричної частини корпусу ядерного реактора:

$$[\sigma] > \frac{p}{2}. \qquad (Б.16)$$

Зрозуміло, що при проведенні розрахунку за формулою (1.1) слід спочатку перевіряти виконання умови (Б.16).

Додаток В

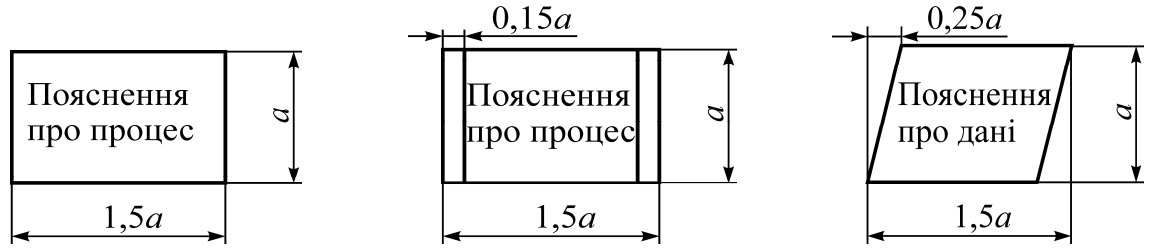
ЗАГАЛЬНІ ВІДОМОСТІ ПРО ГРАФІЧНІ СХЕМИ ПРОГРАМ

Будь-яка програма може бути реалізована за допомогою декількох мов програмування, якщо зрозуміло, що має представляти собою ця програма. Для наочного уявлення щодо алгоритмів, програм, даних та систем незалежно від мови програмування використовують графічні схеми, які дозволяють зрозуміти, яким чином має працювати програма та зв'язки між окремими алгоритмами та даними, а також схему роботи системи та взаємодії програм. Виконання таких графічних схем врегульовано стандартами, наприклад [19], для зрозумілого їхнього використання незалежно від розробника.

Відповідно до стандарту [19] прийнято розрізняти схеми даних, програми, роботи системи, взаємодії програм, а також ресурсів системи. Далі розглянемо тільки схеми програм, які відображають послідовність операцій в програмі та містять [19]: символи процесу, що вказують фактичні операції обробки даних, у тому числі визначники шляху з урахуванням логічних операцій; лінійні символи, що вказують потік управління; спеціальні символи, що спрощують написання та читання програми.

Для символьного визначення процесу на графічних схемах програм використовують прямокутник (рис. В.1а) а для позначення підпрограм – прямокутник із подвійними границями з боків (рис. В.1б); внутрішня область символу використовується для написання пояснень щодо визначуваного процесу [19]. В схемах програм також широко використовують символ (рис. В.1в) даних з невизначеним носієм [19]; такий символ використовують зазвичай для позначення перетворення даних у зручну для обробки форму, наприклад для позначення введення з клавіатури, та у зручну для відображення результатів обробки форму, наприклад для позначення виведення до монітора, як це зазначено в старому стандарті ГОСТ 19.003-80, що вже втратив чинність щодо обов'язкового використання, але певні його положення є досить корисними і сьогодні. Типові приклади використання в схемах програм символів, що показані на рис. В.1, показані вище на рис. 1.1а та на рис. 2.5а.

В схемах програм також доводиться широко використовувати символ процесу – розв'язування (рис. В.2а), який позначає вибір напрямку виконання алгоритму або програми у залежності від деяких змінних умов, тобто відображає розв'язок або функцію перемикаючого типу, що після обчислення умов, що показані всередині символу, за вхідними даними формує один із декількох альтернативних результатів, які позначені поряд із шляхами, що їх відображають [19].

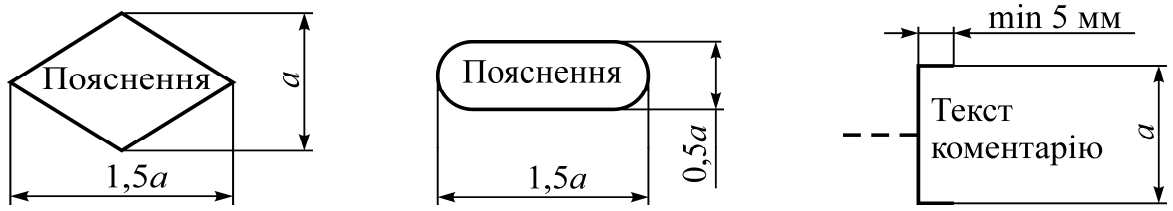


а) *б)* *в)*

а) процес; *б) зумовлений процес (підпрограма);* *в) дані*

Рисунок В.1 – Символи процесів (а, б) та даних (в), що використовуються в схемах програм

Також в схемах програм обов'язково використовують спеціальні символи – термінатор (рис. В.2б), що відображає вихід із програми до зовнішнього середовища та вхід до програми із зовнішнього середовища, тобто початок та завершення програми, а також коментар (рис. В.2в), що використовується для опису та пояснювальних записів для пояснень та приміток, пунктирна лінія якого зв'язана із відповідним символом або групою символів, що обведена такою ж пунктирною лінією. Типові приклади щодо використання в схемах програм символів, що наведені на рис. В.2, показані вище на рис. 1.1 та на рис. 2.1б, на рис. 2.3 та на рис. 2.5, на рис. 2.6 та на рис. 2.10.



а) *б)* *в)*

а) розв'язування; *б) термінатор;* *в) коментар*

Рисунок В.2 – Символ процесів (а) та спеціальні символи (б, в), що використовуються в схемах програм

Для визначення циклів на схемах програм використовується символ, що складається із двох частин, що відображають відповідно початок та кінець циклу (рис. В.3). Умови для ініціалізації, приросту, завершення розташовуються всередині символу на початку або завершенні циклу у залежності від розташування інструкції, що перевіряє відповідну умову. Приклад використання символів з рис. В.3 наведено вище на рис. 2.12.

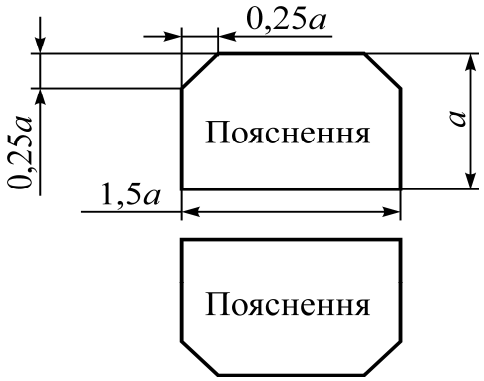


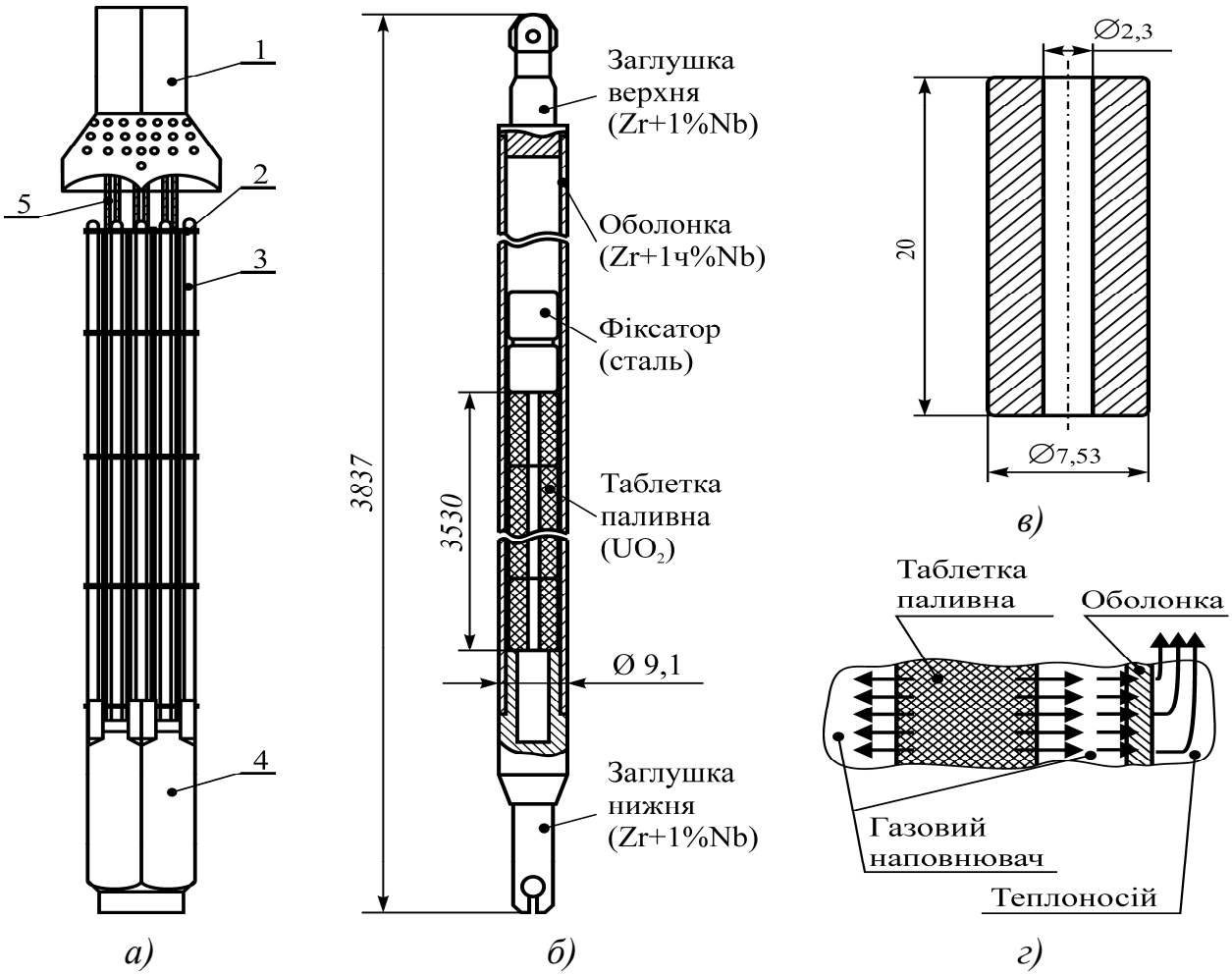
Рисунок В.3 – Символи для відображення початку та кінця циклу

Слід зазначити, що сучасні стандарти [19] не регламентують розміри для графічного відображення на схемах програм, але можна рекомендувати використовувати розміри символів відповідно старому ГОСТ 19.003-80, як це показано на рис. В.1 та рис. В.2, на якому розмір a слід обирати із послідовності 10, 15, 20 мм та припускається збільшення, кратне 5 мм.

Додаток Г

НАБЛИЖЕНЕ ВИЗНАЧЕННЯ ТЕМПЕРАТУРИ ЯДЕРНОГО ПАЛИВА

Ядерне паливо використовується в ядерних реакторах для нагрівання теплоносія за рахунок теплоти ланцюгової реакції ділення ядер з метою подальшого використання теплової енергії нагрітого теплоносія для видобутку електричної енергії та надання її побутовим та промисловим користувачам. На відміну від побутового уявлення про паливо як про деяку речовину (бензин, вугілля та інші), ядерне паливо являє собою досить складну конструкцію, що складається із багатьох деталей (рис. Г.1).



1 – головка; 2 – дистанційна решітка; 3 – тепловиділяючий елемент; 4 – хвостовик; 5 – трубчасті несучі конструкції, що утворюють напрямні канали системи управління та захисту, і центральна несуча труба

Рисунок Г.1 – Типові конструкції ядерного палива, що використовуються для енергетичних ядерних реакторів типу ВВЕР-1000, у вигляді тепловиділяючої збірки (а), що містить тепловиділяючі елементи (б) та паливні таблетки (в), а також теплові потоки від паливної таблетки до теплоносія (г)

Г.1 Окремий виріб ядерного палива називають тепловиділяючою збіркою (рис. Г.1а) [31] – скорочено ТВЗ, із ТВЗ формують активну зону ядерного реактора, в об'ємі якої має здійснюватися ланцюгова реакція ділення ядерного палива; активна зона ядерного реактора ВВЕР-1000 складається із 163 ТВЗ, що розташовані на одній висоті та рівномірно заповнюють циліндр із рівностороннім шестикутником у перерізі [31], тобто перезавантаження палива полягає у перезавантаженні за допомогою підйомно-транспортних машин – кранів ТВЗ (рис. Г.1а) активної зони ядерного реактора.

Типова сучасна несуча конструкція ТВЗ (рис. Г.1а) складається із головки–1 та хвостовика–4, що призначені для фіксації ТВЗ в активній зоні та з'єднані між собою трубчастими несучими конструкціями–5, що являють собою, наприклад, 18 напрямних каналів та центральну трубу в конструкції ТВЗ для ядерного реактора типу ВВЕР-1000 [31]. До несучих трубчастих конструкцій закріплюють дистанційну решітку–2 (рис. Г.1а), в якій закріплюються тепловиділяючі елементи, скорочено – твели (рис. Г.1а), в яких безпосередньо розташований паливний матеріал (рис. Г.2б) у вигляді керамічних виробів (рис. Г.1в) із діоксиду урану; ТВЗ для ядерного реактора типу ВВЕР-100 містить 312 твелів [31].

Твел (рис. Г.1б) являє собою циліндричну оболонку, яка ущільнюється на торцях за допомогою верхньої та нижньої заглушок. Всередині твелу розташовані вироби керамічного паливного матеріалу, виготовлені у вигляді таблетки – циліндра із центральним отвором (див. рис. Г.1в); внутрішня область твелу заповнюється інертним газом. При роботі ядерного реактора тепло, що виділяється в об'ємі паливних таблеток, крізь газовий наповнювач та оболонку відводиться до теплоносія, що рухається зовні оболонки твелу, а невелика частина цього тепла – в область центрального отвору паливної таблетки, як показано стрілками вище на рис. Г.1г.

Г.2 Теплопередача (див. рис. Г.1г) від паливної таблетки до теплоносія здійснюється шляхом теплопровідності [34], яка являє собою фізичний процес передачі кінетичної енергії атомів (молекул) від більш нагрітої частини субстанції, твердого тіла, газу, рідини тощо до її менш нагрітої частини. Теплопровідність досліджують шляхом побудови математичної моделі у вигляді диференціального рівняння та необхідних початкової умови та граничних умов [34]. Далі розглянемо побудову спрощеної математичної моделі стаціонарної, тобто незалежної від часу усталеної теплопровідності. Приймаємо, що потоки тепла (теплові потоки) в паливній таблетці спрямовані тільки уздовж радіального напрямку (рис. Г.2а) та не змінюються протягом часу, і завдяки такому радіальному тепловому потоку в паливній таблетці протягом часу встановлюється симетричне поле температури, тобто тепловий потік та температура (рис. Г.2а) паливної таблетки залежать тільки від радіальної координати:

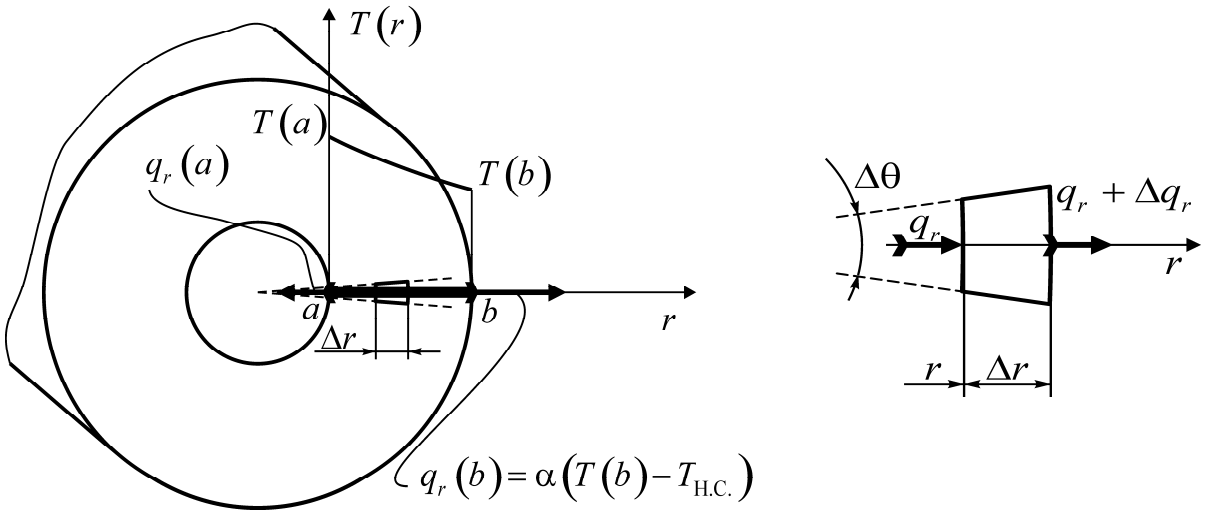
$$q_r = q_r(r),\ T = T(r), \qquad (Г.1)$$

де r – радіальна координата; q_r та T – радіальний тепловий потік та температура.

Розглядаємо теплові потоки в репрезентативному об'ємі паливної таблетки (рис. Г.2б). За одиницю часу на одиницю довжини паливної таблетки через її граничну поверхню r до репрезентативного об'єму потрапляє кількість теплоти $q_r r\Delta\theta$, через поверхню $r+\Delta r$ – відводиться кількість теплоти $(q_r+\Delta q_r)(r+\Delta r)\Delta\theta$, а в самому об'ємі внаслідок ланцюгової ядерної реакції утворюється кількість теплоти $Q(r)\cdot(r+\Delta r/2)\Delta\theta\Delta r$, де позначення відповідають рис. Г.2б. Отже, відповідно до закону збереження енергії (тепла) маємо:

$$q_r r\Delta\theta - (q_r+\Delta q_r)(r+\Delta r)\Delta\theta + Q(r)\cdot(r+\Delta r/2)\Delta\theta\Delta r = 0, \qquad (Г.2)$$

де $Q(r)$ – об'ємна потужність внутрішніх джерел тепла.



а) *б)*

а) температурне поле; *б) теплові потоки в репрезентативному об'ємі*

Рисунок Г.2 – Радіальна теплопровідність паливної таблетки

Після нескладних перетворень рівності (Г.2) матимемо:

$$\frac{q_r}{r+\Delta r/2} + \frac{\Delta q_r}{\Delta r}\frac{r+\Delta r}{r+\Delta r/2} = Q. \qquad (Г.3)$$

За умови, що $\Delta r \to 0$, рівність (Г.3) набуде вигляду диференціального рівняння:

$$\frac{dq_r}{dr} + \frac{q_r}{r} = Q(r). \qquad (Г.4)$$

Оскільки, відповідно до співвідношення (Г.1), окрім невідомого теплового потоку маємо також невідому температуру, то рівняння (Г.4) має бути доповненим відповідним рівнянням, що надасть можливості визначити температуру. Таким додатковим рівнянням є співвідношення закону теплопровідності Фур'є [34], яке має наступний вигляд:

$$q_r = -\lambda \frac{dT}{dr}, \tag{Г.5}$$

де λ – коефіцієнт теплопровідності матеріалу паливної таблетки, який є характеристикою матеріалу та має визначатися шляхом відповідних експериментальних досліджень із зразками цього матеріалу.

Система диференціальних рівнянь (Г.4), (Г.5) визначає теплопровідність у внутрішньому об'ємі паливної таблетки, а передачу тепла на границях $r = a$ та $r = b$ паливної таблетки (див. рис. Г.2б) визначають граничні умови. У випадку ядерного палива в якості граничних умов можна прийняти наступні:

$$q_r(a) = 0, \tag{Г.6}$$

$$q_r(b) = \alpha\left(T(b) - T_{\text{Н.С.}}\right), \tag{Г.7}$$

де α – коефіцієнт теплопередачі від паливної таблетки до теплоносія крізь газовий наповнювач та оболонку; $T_{\text{Н.С.}}$ – температура теплоносія.

Таким чином, у вигляді (Г.4)–(Г.7) маємо математичну постановку задачі про наближене визначення температури у таблетці керамічного ядерного палива (див. рис. Г.1в). При розв'язуванні задачі (Г.4)–(Г.7) зазвичай за допомогою співвідношення (Г.5) виключають із розгляду тепловий потік та розглядають математичну постановку задачі для диференціального рівняння теплопровідності із відповідними граничними умовами:

$$\frac{d^2T}{dr^2} + \frac{1}{r}\frac{dT}{dr} = -\frac{Q(r)}{\lambda}, \quad a < r < b, \tag{Г.8}$$

$$\frac{dT}{dr} = 0, \ r = a, \tag{Г.9}$$

$$\frac{dT}{dr} = -\frac{\alpha}{\lambda}\left(T - T_{\text{Н.С.}}\right), \quad r = b. \tag{Г.10}$$

Отже, наближене дослідження температурного стану ядерного палива зведено до розгляду математичної задачі (Г.8)–(Г.10) – до крайової задачі для звичайного диференціального рівняння теплопровідності (Г.8).

Г.3 Хоча розв'язок лінійної крайової задачі (Г.8)–(Г.10) можна одержати в аналітичному вигляді, така задача сформульована на основі суттєвих спрощень щодо теплопровідності в ядерному паливі. Зрозуміло, що у випадку

більш детального дослідження теплопровідності ядерного палива відповідна крайова задача буде більш складною, ніж (Г.8)–(Г.10), і буде неможливо одержати її точний розв'язок в аналітичному вигляді. Тому розглянемо далі наближене розв'язування крайової задачі (Г.8)–(Г.10) методом сіток [44, 45], який є досить універсальним обчислювальним методом, що може бути використаний для розв'язування різноманітних задач та реалізований за допомогою комп'ютерів, для чого потребується створення спеціального програмного забезпечення.

Основна ідея методу сіток щодо розв'язування крайової задачі (Г.8)–(Г.10) полягає у тому, що в інтервалі $a \le r \le b$ обирають сукупність $n+2$ координат (рис. Г.3):

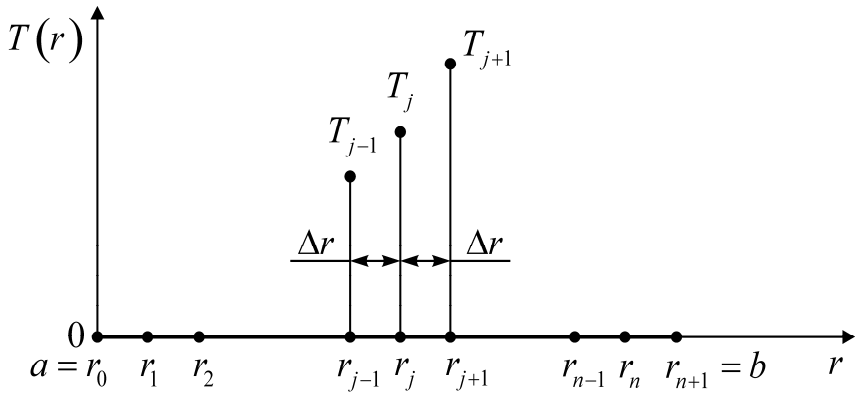


Рисунок Г.3 – Сітка та вузлові значення температури

$$r_0, r_1, r_2, \ldots, r_{j-1}, r_j, r_{j+1}, \ldots, r_{n-1}, r_n, r_{n+1}, \qquad (Г.11)$$

де $n > 0$, $r_0 = a$, $r_{n+1} = b$.

Точки області $a \le r \le b$, які визначаються координатами (Г.11), називають вузлами, а сукупність таких вузлів – сіткою в області $a \le r \le b$. Координати вузлів сітки (Г.11) обираємо таким чином, щоб відстань між вузлами була однаковою (рис. Г.3), тобто:

$$r_j = r_0 + j\Delta r, \quad j = 0,1,2,\ldots,n,n+1, \quad r_0 = a, \quad \Delta r = \frac{b-a}{n+1}, \qquad (Г.12)$$

де Δr – крок сітки (рис. Г.3).

Завдяки сітці (Г.12) маємо можливість ввести до розгляду вузлові значення шуканої температури:

$$T_j = T\left(r_j\right), \quad j = 0,1,2,\ldots,n,n+1. \qquad (Г.13)$$

Таким чином, розв'язування крайової задачі (Г.8)–(Г.10) може бути зведено до наближеного визначення вузлових значень (Г.13) шуканої температури, в чому саме і полягає метод сіток. Зрозуміло, що для достатньо точного визначення вузлових значень (Г.13) число n, яке визначає кількість вузлів сітки (Г.12), має бути досить великим.

Щоб визначити вузлові значення (Г.13) температури, розглянемо диференціальне рівняння (Г.8) у внутрішніх вузлах сітки (Г.12) з номерами $j = 1,2,\ldots,n$, а граничні умови (Г.9) та (Г.10) – у граничних вузлах

сітки з номерами $j=0$ та $j=n+1$ (див. рис. Г.3). Для цього використаємо наступні відомі [44, 45] формули наближеного диференціювання:

$$\frac{d^2T_j}{dr^2}=\frac{T_{j-1}-2T_j+T_{j+1}}{\Delta r^2}+\frac{1}{12}\frac{d^4T_j}{dr^4}\Delta r^2+\ldots, \qquad (\Gamma.14)$$

$$\frac{dT_j}{dr}=\frac{T_{j+1}-T_{j-1}}{2\Delta r}+\frac{1}{6}\frac{d^3T_j}{dr^3}\Delta r^2+\ldots, \qquad (\Gamma.15)$$

$$\frac{dT_j}{dr}=\frac{-3T_j+4T_{j+1}-T_{j+2}}{2\Delta r}-\frac{1}{3}\frac{d^3T_j}{dr^3}\Delta r^2+\ldots, \qquad (\Gamma.16)$$

$$\frac{dT_j}{dr}=\frac{3T_j-4T_{j-1}+T_{j-2}}{2\Delta r}-\frac{1}{3}\frac{d^3T_j}{dr^3}\Delta r^2+\ldots. \qquad (\Gamma.17)$$

Формули (Г1.4) та (Г.15) використаємо далі для розгляду диференціального рівняння (Г.8) у внутрішніх вузлах (Г.12) сітки, що мають номери $j=1,2,\ldots,n$; для розгляду граничної умови (Г.9) використовуємо формулу (Г.16) у вузлі з номером $j=0$, а граничної умови (Г.10) – формулу (Г.17) у вузлі з номером $j=n+1$ (див. рис. Г.3). Отже, граничну умову (Г.9), диференціальне рівняння (Г.8) та граничну умову (Г.10) за допомогою формул (Г.14)–(Г.17) наближеного диференціювання наближено представимо у наступному вигляді:

$$-3T_0+4T_1-T_2=0, \qquad (\Gamma.18)$$

$$\frac{T_{j-1}-2T_j+T_{j+1}}{\Delta r^2}+\frac{1}{r_k}\frac{T_{j+1}-T_{j-1}}{2\Delta r}=-\frac{Q_j}{\lambda},\quad j=1,2,\ldots,n, \qquad (\Gamma.19)$$

$$\frac{3T_{n+1}-4T_n+T_{n-1}}{2\Delta r}=\alpha\left(T_{n+1}-T_{\text{Н.С.}}\right), \qquad (\Gamma.20)$$

де $Q_j\equiv Q\left(r_j\right)$ – відоме вузлове значення густини потужності об'ємного тепловиділення.

За допомогою співвідношень (Г.18) та (Г.20) можемо визначити температури у граничних вузлах сітки:

$$T_0=\frac{4}{3}T_1-\frac{1}{3}T_2, \qquad (\Gamma.21)$$

$$T_{n+1}=-\frac{1}{3-2\alpha\Delta r}T_{n-1}+\frac{4}{3-2\alpha\Delta r}T_n-\frac{2\alpha\Delta r}{3-2\alpha\Delta r}T_{\text{Н.С.}}. \qquad (\Gamma.22)$$

Одержані вирази (Г.21) та (Г.22) дозволяють виключити вузлові значення температури T_0 та T_{n+1} із співвідношень (Г.19), відповідних $j=0$ та $j=n$, та після відповідних тотожних перетворень представити співвідношення (Г.19) у наступному вигляді:

$$A_1T_1 + B_1T_2 = F_1, \tag{Г.23}$$

$$A_jT_{j-1} + B_jT_j + C_jT_{j+1} = F_j,\ j = 2,3,\ldots,n-1, \tag{Г.24}$$

$$B_nT_{n-1} + C_nT_n = F_n, \tag{Г.25}$$

де для скорочення запису використані наступні позначення:

$$A_1 = \frac{4}{3}\left(\frac{1}{\Delta r^2} - \frac{1}{2r_1\Delta r}\right) - \frac{2}{\Delta r^2},$$

$$B_1 = \frac{1}{\Delta r^2} + \frac{1}{2r_1\Delta r} - \frac{1}{3}\left(\frac{1}{\Delta r^2} - \frac{1}{2r_1\Delta r}\right), F_1 = -\frac{Q_1}{\lambda}, \tag{Г.26}$$

$$A_j = \frac{1}{\Delta r^2} - \frac{1}{2r_j\Delta r}, B_j = -\frac{2}{\Delta r^2}, C_j = \frac{1}{\Delta r^2} + \frac{1}{2r_j\Delta r}, F_j = -\frac{Q_j}{\lambda}, j = 2,3,\ldots,n-1, \tag{Г.27}$$

$$B_n = \frac{1}{\Delta r^2} - \frac{1}{2r_n\Delta r} - \frac{1}{3 - 2\alpha\Delta r}\left(\frac{1}{\Delta r^2} + \frac{1}{2r_n\Delta r}\right),$$

$$C_n = \frac{4}{3 - 2\alpha\Delta r}\left(\frac{1}{\Delta r^2} + \frac{1}{2r_n\Delta r}\right) - \frac{2}{\Delta r^2}, \tag{Г.28}$$

$$F_n = -\frac{Q_j}{\lambda} + \frac{2\alpha\Delta r}{3 - 2\alpha\Delta r}\left(\frac{1}{\Delta r^2} + \frac{1}{2r_n\Delta r}\right)T_{H.C.}.$$

Співвідношення (Г.23)–(Г.25) являють собою систему лінійних алгебраїчних рівнянь, яка у компактній матрично-векторній формі (додаток Д) записується наступним чином:

$$\begin{pmatrix} A_1 & B_1 & 0 & 0 & \ldots & 0 & 0 & 0 \\ A_2 & B_2 & C_2 & 0 & \ldots & 0 & 0 & 0 \\ 0 & A_3 & B_3 & C_3 & \ldots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & A_{n-1} & B_{n-1} & C_{n-1} \\ 0 & 0 & 0 & 0 & \ldots & 0 & B_n & C_n \end{pmatrix} \begin{pmatrix} T_1 \\ T_2 \\ T_3 \\ \vdots \\ T_{n-1} \\ T_n \end{pmatrix} = \begin{pmatrix} F \\ F_2 \\ F_3 \\ \vdots \\ F_{n-1} \\ F_n \end{pmatrix}. \tag{Г.29}$$

Розв'язування системи лінійних алгебраїчних рівнянь (Г.29) дозволить визначити вузлові значення температури, відповідні вузлам сітки з номерами $j = 1,2,\ldots,n$, а температури у граничних вузлах з номерами $j = 0$ та $j = n+1$ можна буде визначити за формулами (Г.21), (Г.22). Бачимо, що матриця системи лінійних алгебраїчних рівнянь (Г.29) переважно нульові елементи, а ненульові її елементи зосереджені навколо діагоналі. Такі матриці називають стрічковими, і вони характерні при використанні методу сіток та у розглянутому випадку є наслідком використання формул (Г.14)–(Г.17) наближеного диференціювання.

Додаток Д

ПОЧАТКОВІ ПОНЯТТЯ ПРО МАТРИЦІ ТА ЇХНІ ВЛАСТИВОСТІ

Матриця **A** розміром $n \times m$ являє собою сукупність чисел, розташованих у вигляді таблиці з n рядками та m стовпчиками [35, 36]:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix}, \qquad (Д.1)$$

де a_{ij} $(i = 1,2,\dots,n,\ \ j = 1,2,\dots,m)$ – числа, що являють собою елементи матриці та знаходяться на перетині i-го рядка та j-го стовпця.

У загальному випадку елементами матриці можуть бути інші об'єкти: вектори, функції, їх похідні і т.п. Далі матриці позначатимемо великими жирним літерами латинського алфавіту $\mathbf{A}, \mathbf{B}, \mathbf{C}$ та іншими, а їх елементи – відповідними малими літерами з подвійними індексами a_{ij}, b_{ij}, c_{ij}, де індекс $i = \overline{1,m}$ відповідає номеру рядка матриці, а $j = \overline{1,n}$ – номеру її стовпця.

Д.1 Класифікація матриць здійснюється у залежності від співвідношення кількості рядків та стовпців, а також співвідношеннями між елементами. Матриця $\mathbf{A} = (a_{ij})_{n \times n}$, у якої кількість рядків дорівнює кількості n стовпців, тобто $m = n$, називається *квадратною матрицею порядку n* (або n-го порядку); у протилежному випадку $(m \neq n)$ матрицю називають *прямокутною*. *Матрицею-рядком* називається матриця розміру $1 \times n$, тобто така, що складається з одного рядка. *Матрицею-стовпцем* називається матриця розміру $m \times 1$, яка має один стовпець. *Нульовою матрицею* називають матрицю, всі елементи якої дорівнюють нулю. Вона позначається літерою **O**. *Головною діагоналлю* квадратної матриці називають сукупність її елементів з однаковими індексами

$$\left\{ a_{ii}, i = \overline{1,n} \right\} = \left\{ a_{11}, a_{22}, \dots, a_{nn} \right\}, \qquad (Д.2)$$

а *побічною* – сукупність елементів:

$$\left\{ a_{(n-i+1)i}, i = \overline{1,n} \right\} = \left\{ a_{1n}, a_{2(n-1)}, \dots, a_{n1} \right\}. \qquad (Д.3)$$

Симетричною називається квадратна матриця, у якої всі елементи, що розташовані симетрично відносно головної діагоналі, є попарно рівними між собою. Тобто для довільних її елементів при $i \neq j$ виконується співвідношення:

$$a_{ij} = a_{ji}, \tag{Д.4}$$

де $i = \overline{1,n}$, $j = \overline{1,n}$.

Верхньою (нижньою) трикутною матрицею називається ненульова квадратна матриця, усі елементи якої, що розташовані нижче (вище) головної діагоналі, дорівнюють нулю. *Діагональною* називають квадратну матрицю, всі елементи якої поза головною діагоналлю дорівнюють нулю:

$$\mathbf{D} = diag(d_1,\ldots,d_n) = \begin{pmatrix} d_1 & 0 & \ldots & 0 \\ 0 & d_2 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & d_n \end{pmatrix}. \tag{Д.5}$$

Одиничною матрицею називається діагональна матриця, усі елементи головної діагоналі якої дорівнюють одиниці:

$$\mathbf{E} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \cdots & 1 \end{pmatrix}. \tag{Д.6}$$

Позначення одиничної матриці зручно супроводжувати індексом, який указує на її *порядок*, отже, одинична матриця розміру $n \times n$ позначається $\mathbf{E}_n$. Якщо елементи кожного рядка матриці $\mathbf{A} = \left(a_{ij}\right)_{m\times n}$ записати в стовпець, не порушуючи порядку їхнього розташування, то отримаємо матрицю $\mathbf{A}^T = \left(a_{ji}\right)_{n\times m}$, яка називається *транспонованою* до матриці $\mathbf{A}$:

$$\mathbf{A}^T = \begin{pmatrix} a_{11} & a_{21} & \ldots & a_{m1} \\ a_{12} & a_{22} & \ldots & a_{m2} \\ \ldots & \ldots & \ldots & \ldots \\ a_{1n} & a_{2n} & \ldots & a_{mn} \end{pmatrix}, \tag{Д.7}$$

а сама операція переходу від матриці $\mathbf{A}$ до матриці $\mathbf{A}^T$ називається *транспонуванням* матриці. Зрозуміло, що двічі транспонована матриця дає вихідну матрицю: $\left(\mathbf{A}^T\right)^T = \mathbf{A}$, а симетрична матриця збігається зі своєю транспонованою. Матриці $\mathbf{A} = \left(a_{ij}\right)_{m\times n}$ і $\mathbf{B} = \left(b_{ij}\right)_{m\times n}$ однакового розміру називаються *рівними,* якщо їхні відповідні елементи рівні між собою:

$$\mathbf{A} = \mathbf{B} \Leftrightarrow \left(a_{ij} = b_{ij}, \quad \forall i = \overline{1,m}, \quad \forall j = \overline{1,n}\right). \qquad (Д.8)$$

Д.2 Над матрицями визначені такі *дії* (*операції*): множення матриці на скаляр, додавання матриць, множення матриць. Розглянемо, яким чином визначаються означені операції [35, 36].

Добутком матриці $\mathbf{A} = \left(a_{ij}\right)_{m\times n}$ *зі скаляром* $\lambda = \text{const}$ називається матриця $\mathbf{C} = \left(c_{ij}\right)_{m\times n}$, відповідні елементи якої є добутком елементів матриці $\mathbf{A}$ зі сталим множником λ:

$$\mathbf{C} = \lambda\mathbf{A} \Leftrightarrow c_{ij} = \lambda a_{ij}. \qquad Д.9)$$

Певна річ, якщо $\lambda = -1$, то отримуємо матрицю, яка називається *протилежною* (до) матриці $\mathbf{A}$: $-\mathbf{A} = (-1)\cdot\mathbf{A}$.

Сумою матриць $\mathbf{A} = \left(a_{ij}\right)_{m\times n}$ і $\mathbf{B} = \left(b_{ij}\right)_{m\times n}$ однакового розміру називають матрицю $\mathbf{C} = \mathbf{A} + \mathbf{B}$ того самого розміру, кожний елемент c_{ij} якої є сумою відповідних елементів вихідних матриць:

$$\mathbf{C} = \mathbf{A} + \mathbf{B} \Leftrightarrow \left(c_{ij} = a_{ij} + b_{ij}, \quad \forall i = \overline{1,m}, \quad \forall j = \overline{1,n}\right). \qquad (Д.10)$$

Очевидно, що згідно з означенням нульової матриці маємо: $\mathbf{A} + \mathbf{O} = \mathbf{A}$. А різницю двох матриць $\mathbf{C} = \mathbf{A} - \mathbf{B}$ можна розглядати як суму матриць, якщо до матриці A додати матрицю, яка протилежна матриці $\mathbf{B}$: $\mathbf{A} - \mathbf{B} = \mathbf{A} + (-1)\cdot\mathbf{B}$.

Добутком матриці $\mathbf{A}$ розміру $m\times k$ на матрицю $\mathbf{B}$ розміру $k\times n$ називається матриця $\mathbf{C}$ розміру $m\times n$, елементи якої обчислюються за наступною формулою: $c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + a_{i3}b_{3j} + \ldots + a_{ik}b_{kj}$. Відповідно до цього маємо таке визначення добутку матриць:

$$\mathbf{C} = \mathbf{A}\cdot\mathbf{B} \Leftrightarrow c_{ij} = \sum_{s=1}^{k} a_{is}b_{sj} \quad \forall i = \overline{1,m}, \quad \forall j = \overline{1,n}. \qquad (Д.11)$$

Співвідношення (Д.11) можна представити за допомогою відомого мнемонічного правила, яке називають зазвичай правилом «*рядок на стовпець*»; відповідно до цього мнемонічного правила, для того щоб обчислити елемент c_{ij} добутку матриць $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$, треба елементи i-го рядка матриці $\mathbf{A} = (a_{ij})_{m \times k}$ помножити на відповідні за номером елементи j-го стовпця матриці $\mathbf{B} = (b_{ij})_{k \times n}$ і знайти суму отриманих добутків. *Піднесенням до степеня* k називають множення матриці на себе k разів:

$$\mathbf{A}^k = \underbrace{\mathbf{A} \cdot \mathbf{A} \cdot \ldots \cdot \mathbf{A}}_{k \text{ разів}}, \qquad (Д.12)$$

тому ця операція визначена тільки для квадратних матриць. Піднесення до степеня є частковим випадком добутку матриць, тому має ті самі властивості.

Підсумовуючи, зазначимо, що операції над матрицями мають різний *пріоритет*. Наведемо перелік операцій у послідовності спадання пріоритету:

1) транспонування – найвищий пріоритет;

2) множення матриці на число та множення матриць (у частинному випадку, піднесення до степеня) – мають однаковий пріоритет;

3) додавання (віднімання) матриць – найнижчий пріоритет.

Д.3 Розглянемо далі властивості операцій над матрицями. Обидві операції – додавання матриць та множення матриці на скаляр називаються *лінійними операціями* над матрицями. Ці операції зводяться до відповідних арифметичних дій над числами і мають ті самі властивості, що й операції над числами. Наведемо ці властивості:

- $\mathbf{A} + \mathbf{B} = \mathbf{B} + \mathbf{A}$, $\lambda\mathbf{A} = \mathbf{A}\lambda$ – *комутативність* (переставний закон);
- $(\mathbf{A} + \mathbf{B}) + \mathbf{C} = \mathbf{A} + (\mathbf{B} + \mathbf{C}) = \mathbf{A} + \mathbf{B} + \mathbf{C}$, $\lambda(\mu\mathbf{A}) = (\lambda\mu)\mathbf{A}$ – *асоціативність* (сполучний закон);
- $\lambda(\mathbf{A} + \mathbf{B}) = \lambda\mathbf{A} + \lambda\mathbf{B}$, $(\lambda + \mu)\mathbf{A} = \lambda\mathbf{A} + \mu\mathbf{A}$ – *дистрибутивність* (розподільний закон).

Зауважимо, що з означення добутку матриць випливають такі властивості операції множення матриць:

- для того, щоб дві матриці $\mathbf{A}$ і $\mathbf{B}$ можна було перемножити, необхідно і достатньо, щоб кількість стовпців n першої матриці (множеного $\mathbf{A}$) дорівнювала кількості рядків m другої матриці (множника $\mathbf{B}$); відповідно, добутком матриці розміру $m \times k$ на матрицю розміру $k \times n$ буде матриця розміру $m \times n$;
- якщо матриці $\mathbf{A}$ і $\mathbf{B}$ є квадратними матрицями однакового розміру, то існують і добуток $\mathbf{A} \cdot \mathbf{B}$, і добуток BA, але може бути, що $\mathbf{A} \cdot \mathbf{B} \neq \mathbf{B} \cdot \mathbf{A}$, тобто для добутку матриць у загальному випадку не виконується переставний закон. Однак добутки одиничної матриці $\mathbf{E}$ з матрицею $\mathbf{A}$ (того самого

розміру) зліва і справа є рівними між собою і дорівнюють матриці $\mathbf{A}$: $\mathbf{A}\cdot\mathbf{E}=\mathbf{E}\cdot\mathbf{A}=\mathbf{A}$;

• для добутку матриць виконується асоціативний закон: $(\mathbf{A}\cdot\mathbf{B})\cdot\mathbf{C}=\mathbf{A}\cdot(\mathbf{B}\cdot\mathbf{C})$.

Д.4 Далі розглянемо важливі числові характеристики матриць, які широко використовуються при формулюванні результатів.

Д.4.1 Важливою характеристикою квадратної матриці є *визначник*, або *детермінант*, який позначають грецькою буквою Δ, або скороченням det (від лат. *determinans* – той, що визначає) з посиланням на відповідну матрицю, наприклад $\Delta\mathbf{A}$ або $\det\mathbf{A}$. Але перш ніж надати означення детермінанта, слід ввести до розгляду деякі допоміжні поняття. Нехай задана скінченна множина з n перших натуральних чисел $\{1, 2, 3, ..., n\}$. Крім природного розташування чисел за зростанням, їх можна упорядкувати також іншими способами. Будь-яке розташування чисел $1, 2, 3, ..., n$ у деякому певному порядку називається *переставленням* із n чисел, а самі числа називають *елементами* переставлення. Число різних можливих переставлень із n чисел дорівнює добутку $1\cdot 2\cdot ...\cdot n$, який позначається через $n!$ і читається: «*ен-факторіал*». Розглянемо два довільні елементи a і b переставлення з n елементів $(\{a, b\}\subset\{1, 2, 3, ..., n\})$. Кажуть, що елементи a і b утворюють *інверсію* (від грецьк. *inversio* – перевертання, переставляння), якщо $a>b$, але в переставленні a передує b, тобто порушується порядок розташування чисел за зростанням. Переставлення називається *парним* (*непарним*), якщо воно має в собі парне (непарне) число інверсій. Числа, які утворюють переставлення, звичайно беруться у круглі дужки. Так, переставлення з чотирьох чисел: (3, 1, 4, 2) містить три інверсії; їх утворюють три пари чисел: (3,1), (3,2) та (4,2), тобто це переставлення є непарним. Переставлення (4, 2, 1, 3) містить чотири інверсії: (4,2), (4,1), (4,3) та (2,1), отже, воно є парним.

Д.4.1.1 *Визначником*, або *детермінантом*, *n-го порядку* матриці $\mathbf{A}=(a_{ij})_{n\times n}$ називається число, яке дорівнює алгебраїчній сумі доданків виду $a_{1j_1}\cdot a_{2j_2}\cdot a_{3j_3}\cdot ...\cdot a_{nj_n}$, кожний з яких є добутком n елементів матриці, вибраних по одному і тільки по одному з кожного рядка і кожного стовпця, при цьому добуток береться зі своїм (з протилежним) знаком, якщо переставлення $(j_1, j_2, j_3, ..., j_n)$ з других індексів елементів є парним (непарним):

$$\det\mathbf{A}=\sum_{(j_1, j_2, ..., j_n)}(-1)^{s(j_1, j_2, ..., j_n)}a_{1j_1}\cdot a_{2j_2}\cdot ...\cdot a_{nj_n}, \qquad (Д.13)$$

де $s(j_1, j_2, ..., j_n)$ – кількість інверсій у переставленні $(j_1, j_2, ..., j_n)$, утвореному з других індексів елементів матриці з перших індексами, розташова-

ними у порядку зростання; доданки суми (Д.13) називаються *членами визначника*.

Визначники зображають, як і матрицю, у вигляді таблиці чисел, але в прямих дужках (а не в круглих чи в квадратних):

$$\det \mathbf{A} = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix}. \qquad (Д.14)$$

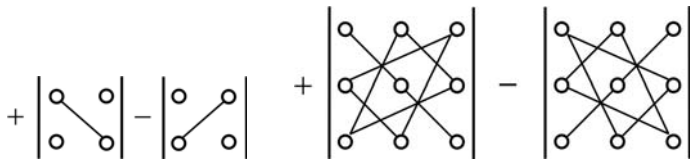
Відповідно до визначення (Д.13), визначником 1-го порядку матриці $\mathbf{A} = (a_{ij})_{1\times 1} = (a_{11})$ є сам елемент, з якого він складається:

$$\det \mathbf{A} = \left|a_{11}\right| = a_{11}. \qquad (Д.15)$$

Згідно з визначенням (Д.13) детермінант 2-го порядку дорівнює сумі добутку елементів головної діагоналі, взятого зі знаком «плюс», і добутку елементів побічної діагоналі, взятого зі знаком «мінус», оскільки переставлення з других індексів елементів першого члена не містить інверсій: $(1, 2)$, а переставлення з других індексів елементів другого члена – одну інверсію $(2, 1)$.

$$\det \mathbf{A} = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11} \cdot a_{22} - a_{12} \cdot a_{21}. \qquad (Д.16)$$

На рис. Д.1а наведено геометричну схему, за якою обчислюється $\det \mathbf{A}$ відповідно формулі (Д.16). На ній елементи визначника позначені кружками, а відрізками з'єднані елементи, які знаходяться у різних рядках і різних стовпцях. За визначенням (Д.13) детермінант третього порядку має шість членів: $3! = 1 \cdot 2 \cdot 3 = 6$ і обчислюється за формулою:



а) *б)*

а) 2-го порядку; *б) 3-го порядку*

Рисунок Д.1 – Геометрична схема формули обчислення визначника

$$\det \mathbf{A} = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{13}a_{22}a_{31} - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33}. \qquad (Д.17)$$

На рис. Д.1б зображена геометрична схема, за якою обчислюється визначник 3-го порядку, її називають «*правилом трикутників*». На цій схемі, як і вище, елементи визначника позначені кружками, а відрізками

з'єднуються елементи, які знаходяться у різних рядках і різних стовпцях; вони визначають шість трикутників (за кількістю членів детермінанта). У першої трійки членів переставлення з других індексів елементів парні, тому кожен із них береться зі знаком «плюс», а у другої – непарні, тому кожен береться зі знаком «мінус».

Як приклад наведемо обчислення визначника 3-го порядку:

$$\det \mathbf{A} = \begin{vmatrix} 1 & 0 & 5 \\ -2 & 3 & 1 \\ 3 & -4 & -3 \end{vmatrix} =$$

$$= 1\cdot 3\cdot(-3) + 0\cdot 1\cdot 3 + 5\cdot(-2)\cdot(-4) - 5\cdot 3\cdot 3 - 0\cdot(-2)\cdot(-3) - 1\cdot 1\cdot(-4) = -10.$$

Зрозуміло, що кількість членів детермінанта швидко зростає зі збільшенням його порядку. Наприклад, обчислення визначника четвертого порядку потребує підрахунку двадцяти чотирьох його членів: $4! = 1\cdot 2\cdot 3\cdot 4 = 24$, а при $n = 5$ вже маємо $5! = 120$. Поряд зі словосполученням «обчислення визначника» використовують «розкриття визначника» як синонім.

Д.4.1.2 Розглянемо далі найважливіші властивості визначників.

Властивість 1 (про визначник транспонованої матриці). Визначник транспонованої матриці $\mathbf{A}^T$ дорівнює визначнику вихідної матриці $\mathbf{A}$:

$$\det \mathbf{A}^T = \det \mathbf{A}. \qquad (Д.18)$$

Ця властивість вказує на рівноправність рядків і стовпців визначника, тому при подальшому розгляді ми будемо формулювати властивості визначника відносно дій над рядками, але ті самі властивості поширюються і на дії над стовпцями.

Властивість 2 (про зміну знака). Якщо у визначнику поміняти місцями два рядки, то отримаємо визначник, який має протилежний знак:

$$\Delta = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{k1} & a_{k2} & \cdots & a_{kn} \\ \cdots & \cdots & \cdots & \cdots \\ a_{s1} & a_{s2} & \cdots & a_{sn} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix} \Leftrightarrow \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{s1} & a_{s2} & \cdots & a_{sn} \\ \cdots & \cdots & \cdots & \cdots \\ a_{k1} & a_{k2} & \cdots & a_{kn} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix} = -\Delta, \ k \neq s, \qquad (Д.19)$$

де k і s – номери двох рядків, які міняються місцями.

Властивість 3 (про спільний множник елементів рядка). Спільний множник λ елементів рядка можна винести за знак (символ) визначника:

$$\Delta = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \dots & \dots & \dots & \dots \\ \lambda \cdot a_{k1} & \lambda \cdot a_{k2} & \dots & \lambda \cdot a_{kn} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix} = \lambda \cdot \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \dots & \dots & \dots & \dots \\ a_{k1} & a_{k2} & \dots & a_{kn} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}. \qquad (Д.20)$$

Відповідно (Д.20), якщо всі елементи будь-якого рядка визначника помножити на деяке число λ, то вихідний визначник помножиться на λ.

Властивість 4 (про рівність визначника нулю). Визначник дорівнює нулю, якщо:

- всі елементи деякого рядка дорівнюють нулю:

$$\left(\exists i : a_{ij} = 0,\ \forall j = \overline{1,n}\ \right) \Rightarrow \det \mathbf{A} = 0. \qquad (Д.21)$$

- два або більше рядків є однаковими або пропорційними:

$$\left(\exists \{i_1, i_2\} : a_{i_1 j} = a_{i_2 j} \text{ або } a_{i_1 j} = \lambda \cdot a_{i_2 j}, \forall j = \overline{1,n}\right) \Rightarrow \det \mathbf{A} = 0. \qquad (Д.22)$$

Властивість 5 (про подання визначника як суми двох визначників). Якщо елементи будь-якого рядка визначника (нехай це буде i-й рядок) є сумою двох доданків, то цей визначник можна подати у вигляді суми двох визначників того ж порядку, у яких елементи усіх рядків, окрім i-го, є елементами вихідного детермінанта, а елементами їхніх i-х рядків є доданки i-го рядка заданого визначника:

$$\begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{i1} + b_{i1} & a_{i2} + b_{i2} & \cdots & a_{in} + b_{in} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix} = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{i1} & a_{i2} & \cdots & a_{in} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix} + \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ \cdots & \cdots & \cdots & \cdots \\ b_{i1} & b_{i2} & \cdots & b_{in} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix}. \qquad (Д.23)$$

Властивість 6 (про інваріантність, тобто незмінність визначника). Визначник не зміниться, якщо до елементів будь-якого його рядка додати відповідні за номером елементи іншого рядка, помножені на будь-яке задане число k:

$$\Delta = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{i1} & a_{i2} & \cdots & a_{in} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix} = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{i1} + ka_{m1} & a_{i2} + ka_{m2} & \cdots & a_{in} + ka_{mn} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix}. \qquad (Д.24)$$

Дійсно, за попередньою властивістю (Д.23) отримуємо суму двох визначників, перший з яких дорівнює Δ, а другий, який містить два рядки з пропорційними елементами a_{mj} і ka_{mj} ($\forall j = \overline{1,n}$), та за властивістю (Д.22) дорівнюватиме нулю.

Д.4.1.3 Далі наведемо *теорему Лапласа,* яка має важливе теоретичне значення та може застосовуватися щодо обчислення визначників, порядок яких більше трьох. Для її формулювання та доведення необхідно означити нові поняття.

Мінором елемента a_{ij} визначника n-го порядку (від лат. *minor* – менший) називається визначник $(n-1)$-го порядку, який одержують з вихідного визначника вилученням i-го рядка і j-го стовпця, на перетині яких розташований даний елемент. Він позначається M_{ij}. *Алгебраїчним доповненням елемента a_{ij}* визначника n-го порядку називають його мінор, взятий зі знаком «плюс», якщо сума індексів $(i+j)$ є парним (непарним) числом, і зі знаком «мінус», якщо ця сума індексів непарна. Він позначається A_{ij}:

$$A_{ij} = \begin{cases} +M_{ij}, \text{ якщо } (i+j) \text{ парне}, \\ -M_{ij}, \text{ якщо } (i+j) \text{ непарне}, \end{cases} \Leftrightarrow A_{ij} = (-1)^{i+j} \cdot M_{ij}. \qquad (Д.25)$$

Для прикладу визначимо мінор і алгебраїчне доповнення елемента a_{12} визначника $\Delta = \begin{vmatrix} 1 & 2 & 3 \\ -1 & 0 & 4 \\ 4 & 3 & 5 \end{vmatrix}$. Для цього вилучаємо з вихідного визначника перший рядок та другий стовпець, на перетині яких розташований елемент a_{12}, і обчислюємо визначник 2-го порядку. Оскільки для елемента a_{12} сума індексів є непарною, то мінор і алгебраїчне доповнення цього елемента відрізняються лише знаком:

$$\Delta = \begin{vmatrix} \cancel{1} & \cancel{2} & \cancel{3} \\ -1 & \cancel{0} & 4 \\ 4 & \cancel{3} & 5 \end{vmatrix} \Rightarrow \left(M_{12} = \begin{vmatrix} -1 & 4 \\ 4 & 5 \end{vmatrix} = -21,\ A_{12} = (-1)^{1+2} \begin{vmatrix} -1 & 4 \\ 4 & 5 \end{vmatrix} = 21 \right).$$

Теорема Д.1. (***Теорема Лапласа про розкриття визначника***). Визначник n-го порядку дорівнює сумі добутків елементів будь-якого рядка або стовпця з їхніми алгебраїчними доповненнями:

$$\Delta = \begin{bmatrix} a_{i1}A_{i1} + a_{i2}A_{i2} + \ldots + a_{in}A_{in} & \forall i = \overline{1,n}; \\ a_{1j}A_{1j} + a_{2j}A_{2j} + \ldots + a_{nj}A_{nj} & \forall j = \overline{1,n}. \end{bmatrix} \qquad (Д.26)$$

З **теореми Д.1** випливає наступний наслідок: сума добутків елементів будь-якого рядка (стовпця) з алгебраїчними доповненнями відповідних (за номером) елементів іншого рядка або стовпця дорівнює нулю:

$$\begin{bmatrix} a_{i1}A_{k1} + a_{i2}A_{k2} + \ldots + a_{in}A_{kn} = 0 & \forall i = \overline{1,n} \quad \text{при} \quad k \neq i; \\ a_{1j}A_{1l} + a_{2j}A_{2l} + \ldots + a_{nj}A_{nl} = 0 & \forall j = \overline{1,n} \quad \text{при} \quad l \neq j. \end{bmatrix} \qquad (Д.27)$$

Так, за **теоремою Д.1** обчислення визначника n-го порядку можна завжди звести до обчислення n визначників $(n-1)$-го порядку, якими є алгебраїчні доповнення елементів обраного рядка чи стовпця. Аналогічно розкриття кожного з визначників $(n-1)$-го порядку зводяться до обчислення визначників $(n-2)$-го порядку, кількість яких становить $(n-1)$, і далі, поки не дійдемо до визначників першого порядку – елементів вихідного визначника. *Застосування теореми Лапласа із залученням властивості про інваріантність визначника*. Зрозуміло, що обсяг обчислень за формулою (Д.27) зменшується, якщо деякі з елементів обраного рядка дорівнюють нулю, оскільки відповідні їм алгебраїчні доповнення нема потреби обчислювати. Властивість 6 (*про інваріантність визначника*) дозволяє отримати всі елементи довільного рядка чи стовпця рівними нулю, окрім одного. Елемент a_{rs}, який залишається відмінним від нуля, називається *провідним* елементом, а відповідний рядок з номером r (або стовпець s), у якому стоїть a_{rs} і за допомогою якого саме і здійснюються перетворення, теж називається *провідним.*

Тепер розглянемо приклад та обчислимо визначник 3-го порядку (див. с. 152) за теоремою Лапласа та переконаємось в тому, що результати обчислень за визначенням (Д.13) і за **теоремою Д.1** збігаються. При цьому звертаємо увагу на те, що $a_{12} = 0$, тому доцільно здійснювати розкладання

визначника за елементами або першого рядка, або другого стовпця, оскільки у цих випадках матимемо лише два доданки. Якщо розкладати за елементами першого рядка, то отримуємо:

$$\begin{vmatrix} 1 & 0 & 5 \\ -2 & 3 & 1 \\ 3 & -4 & -3 \end{vmatrix} = a_{11}A_{11} + a_{12}A_{12} + a_{13}A_{13} = 1 \cdot \begin{vmatrix} 3 & 1 \\ -4 & -3 \end{vmatrix} + 5 \cdot \begin{vmatrix} -2 & 3 \\ 3 & -4 \end{vmatrix} =$$

$$= 1 \cdot (-9+4) + 5 \cdot (8-9) = 1 \cdot (-5) + 5 \cdot (-1) = -10.$$

Доцільно застосовувати теорему Лапласа до обчислення визначника трикутної матриці, оскільки визначник верхньої трикутної матриці послідовно розкривають за елементами 1-го стовпця кожного з n визначників, які для такої матриці є алгебраїчними доповненнями елементів головної діагоналі. Тоді вихідний визначник отримаємо у вигляді добутку елементів його головної діагоналі:

$$\Delta = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & a_{nn} \end{vmatrix} = a_{11} \begin{vmatrix} a_{22} & a_{23} & \cdots & a_{2n} \\ 0 & a_{33} & \cdots & a_{3n} \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & a_{nn} \end{vmatrix} = a_{11} \cdot a_{22} \begin{vmatrix} a_{33} & a_{32} & \cdots & a_{3n} \\ 0 & a_{44} & \cdots & a_{4n} \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & a_{nn} \end{vmatrix} =$$

$$= \ldots = a_{11} \cdot a_{22} \cdot a_{33} \cdot \ldots \cdot a_{nn}.$$

Таку ж формулу одержимо для обчислення визначника нижньої трикутної матриці, якщо на кожному кроці розкривати за елементами першого рядка алгебраїчні доповнення діагональних елементів.

Д.4.2 Розглянемо числову характеристику матриць довільного розміру.

Д.4.2.1 *Рангом матриці* $\text{rang}(\mathbf{A})$ називають максимальний порядок ненульових мінорів, що містить в собі матриця $\mathbf{A}$. Будь-який ненульовий мінор порядку $r = \text{rang}(\mathbf{A})$ називають *базовим*. Для відшукання рангу матриці зазвичай застосовують *метод елементарних перетворень матриці*:

- переставлення рядків (стовпців);
- множення всіх елементів деякого рядка (стовпця) на ненульове число;
- додавання до елементів деякого рядка (стовпця) відповідних елементів іншого рядка (стовпця), помножених на число;
- викреслення нульових рядків (стовпців);
- викреслення пропорційних рядків (стовпців).

Якщо матрицю $\mathbf{A}$ шляхом елементарних перетворень можна звести до матриці $\mathbf{B}$ (і навпаки), то їх називають *еквівалентними* та пишуть: $\mathbf{A} \sim \mathbf{B}$.

Теорема Д.2. Елементарні перетворення не змінюють рангу матриці.

Теорема Д.3. Будь-яка матриця може бути зведеною до трикутного або ступінчастого (трапецоїдного) вигляду шляхом елементарних перетворень.

Оскільки в рамках цього посібника ми активно та неодноразово обчислюємо ранг матриці, то наведемо тут легкий для сприйняття приклад знаходження рангу заданої матриці A з детальною демонстрацією елементарних перетворень (які, до речі, за **теоремою Д.2** не змінюють її рангу). Отже:

$$\mathbf{A}=\begin{pmatrix}1 & 1 & -3 & -1\\ 2 & -1 & -3 & -8\\ 1 & 2 & -4 & 1\\ 1 & -1 & -1 & -5\end{pmatrix}\underset{\substack{p2=p2-2\cdot p1\\ p3=p3-p1\\ p4=p4-p1}}{\sim}\begin{pmatrix}1 & 1 & -3 & -1\\ 0 & -3 & 3 & -6\\ 0 & 1 & -1 & 2\\ 0 & -2 & 2 & -4\end{pmatrix}\sim$$

$$\underset{\substack{p2=p2+3\cdot p3\\ p4=p4+2\cdot p3}}{\sim}\begin{pmatrix}1 & 1 & -3 & -1\\ 0 & 0 & 0 & 0\\ 0 & 1 & -1 & 2\\ 0 & 0 & 0 & 0\end{pmatrix}\underset{\substack{p1=p1-p3\\ p2=\varnothing\\ p4=\varnothing}}{\sim}\begin{pmatrix}1 & 0 & -2 & -3\\ 0 & 1 & -1 & 2\end{pmatrix}.$$

Бачимо, що максимальний порядок ненульового мінору матриці A дорівнює двом: $\operatorname{rang}(\mathbf{A})=2$.

Д.4.2.2 Перейдемо тепер до деяких понять, які використовуються як в теоретичних дослідженнях, пов'язаних із властивостями матриць, так і в прикладних задачах.

Квадратна матриця $\mathbf{A}$ називається *неособливою*, або *невиродженою*, якщо її визначник не дорівнює нулю: $\det\mathbf{A}\neq 0$. У протилежному випадку вона називається *особливою,* або *виродженою*.

Матриця $\mathbf{A}^*$, транспонована до матриці, складеної з алгебраїчних доповнень елементів матриці A, називається *приєднаною*, або *союзною*, до цієї матриці $\mathbf{A}$:

$$\mathbf{A}^*=\begin{pmatrix}A_{11} & A_{21} & \cdots & A_{n1}\\ A_{12} & A_{22} & \cdots & A_{n2}\\ \cdots & \cdots & \cdots & \cdots\\ A_{1n} & A_{2n} & \cdots & A_{nn}\end{pmatrix}. \qquad (Д.28)$$

Матриця $\mathbf{A}^{-1}$ називається *оберненою* до матриці $\mathbf{A}$, якщо добуток цієї матриці з матрицею $\mathbf{A}$ як зліва, так і справа дорівнює одиничній матриці:

$$\mathbf{A}^{-1}\cdot\mathbf{A}=\mathbf{A}\cdot\mathbf{A}^{-1}=\mathbf{E}. \qquad (Д.29)$$

Теорема Д.4 (*існування єдиності оберненої матриці*). Для будь-якої невиродженої квадратної матриці $\mathbf{A}$ існує єдина обернена матриця $\mathbf{A}^{-1}$:

$$\mathbf{A}^{-1} = \frac{1}{\det \mathbf{A}} \left(\mathbf{A}^{*} \right)^{T}. \qquad (Д.30)$$

За допомогою оберненої матриці розв'язують матричні рівняння вигляду:

$$\mathbf{A} \cdot \mathbf{X} = \mathbf{B} \text{ або } XA = B, \qquad (Д.31)$$

де $\mathbf{A}$ – задана невироджена квадратна матриця, $\mathbf{B}$ – також відома матриця узгодженого розміру, $\mathbf{X}$ – невідома матриця. Якщо помножити ці матричні рівняння на $\mathbf{A}^{-1}$ зліва та справа відповідно, а потім скористатися означенням оберненої матриці та властивістю, отримаємо наступні формули для відшукання розв'язку матричного рівняння:

$$\mathbf{A} \cdot \mathbf{X} = \mathbf{B} \Rightarrow \mathbf{X} = \mathbf{A}^{-1} \cdot \mathbf{B}; \ \mathbf{X} \cdot \mathbf{A} = \mathbf{B} \Rightarrow \mathbf{X} = \mathbf{B} \cdot \mathbf{A}^{-1} \text{ або } XA = B. \qquad (Д.32)$$

Д.4.2.3 За означенням добутку матриць у вигляді матричного рівняння (Д.31) (*матрична форма системи*) можна подати будь-яку *систему лінійних алгебраїчних рівнянь* (скорочено СЛАР) – сукупність m рівнянь першого степеня, тобто лінійних рівнянь, відносно n невідомих $x_1, x_2, ..., x_n$:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 \quad + \ldots \quad + a_{1n}x_n = b_1; \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots + a_{2n}x_n = b_2; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + \ldots + a_{mn}x_n = b_m \end{cases}, \qquad (Д.33)$$

де $\mathbf{A} = \left(a_{ij} \right)_{m \times n}$ – основна матриця системи; $\mathbf{B} = \left(b_i \right)_{m \times 1}$ – матриця-стовпець вільних членів; $\mathbf{X} = \left(x_j \right)^{T}_{1 \times n}$ – матриця-стовпець невідомих.

Розв'язком системи рівнянь називається такий набір чисел $(\alpha_1, \alpha_2, \alpha_3, ..., \alpha_n)$, який при його підстановці замість невідомих $(x_1, x_2, x_3, ..., x_n)$ у кожне рівняння системи перетворює ці рівняння у числові тотожності. Система лінійних алгебраїчних рівнянь називається *сумісною*, якщо вона має принаймні один розв'язок, і *несумісною,* якщо вона не має жодного розв'язку. Сумісну систему рівнянь називають *визначеною*, якщо вона має єдиний розв'язок, і *невизначеною*, якщо кількість розв'язків більша одного.

Система лінійних рівнянь називається *неоднорідною*, якщо хоча б один елемент матриці-стовпця вільних членів є відмінним від нуля. Якщо всі елементи b_i $(i=\overline{1,m})$ дорівнюють нулю, то така система лінійних рівнянь називається *однорідною.* Однорідна система рівнянь має вигляд:

$$\mathbf{A}\cdot\mathbf{X}=\mathbf{O}, \tag{Д.34}$$

де $\mathbf{A}=\left(a_{ij}\right)_{m\times n}$ – основна матриця системи; $\mathbf{X}=\left(x_j\right)^T_{1\times n}$ – матриця-стовпець невідомих; $\mathbf{O}=\left(0\right)_{m\times 1}$ – нульова матриця-стовпець.

Якщо ми приєднаємо справа (через вертикальну риску) до основної матриці A матрицю-стовпець вільних членів $\mathbf{B}=\left(b_i\right)_{m\times 1}$, то отримаємо матрицю $\mathbf{A}\,|\,\mathbf{B}=\left(a_{ij}\,\middle|\,b_i\right)_{m\times(n+1)}$, яку називають *розширеною матрицею* системи:

$$\mathbf{A}\,|\,\mathbf{B}=\left(\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1\\ a_{21} & a_{22} & \cdots & a_{2n} & b_2\\ \vdots & \vdots & \ddots & \vdots & \vdots\\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_m \end{array}\right). \tag{Д.35}$$

Для дослідження системи на сумісність та визначеність використовується

Теорема Д.5 (*Кронекера–Капеллі, критерій сумісності системи*). Система лінійних алгебраїчних рівнянь сумісна тоді і тільки тоді, коли ранг основної матриці системи дорівнює рангу її розширеної матриці:

$$\text{СЛАР сумісна} \;\Leftrightarrow\; \operatorname{rang}\left(\mathbf{A}\right)=\operatorname{rang}\left(\mathbf{A}\,|\,\mathbf{B}\right).$$

З теореми Кронекера–Капеллі і означення рангу випливає, що

- система несумісна тоді і тільки тоді, коли ранг основної матриці системи не дорівнює рангу її розширеної матриці;
- якщо ранги збігаються і дорівнюють кількості невідомих $(\operatorname{rang}\left(\mathbf{A}\right)=\operatorname{rang}\left(\mathbf{A}\,|\,\mathbf{B}\right)=n)$, то система має єдиний розв'язок;
- якщо ранги рівні між собою, але менші числа невідомих $(\operatorname{rang}\left(\mathbf{A}\right)=\operatorname{rang}\left(\mathbf{A}\,|\,\mathbf{B}\right)<n)$, то система має безліч розв'язків.

Основна матриця $\mathbf{A}=(a_{ij})_{m\times n}$ є складовою частиною розширеної матриці $\mathbf{A}\,|\,\mathbf{B}=(a_{ij}\,|\,b_i)_{m\times(n+1)}$, тому ранги матриць $\mathbf{A}$ і $\mathbf{A}\,|\,\mathbf{B}$ визначають не окремо один від одного, а досліджують розширену матрицю системи. Зрозуміло, що ранг основної матриці системи не може бути більше рангу

розширеної матриці. Корисним для застосування буде розгляд особливостей розв'язку однорідної системи рівнянь (Д.34):

- однорідна система рівнянь завжди сумісна, адже нульові значення невідомих задовольняють систему. Це узгоджується з теоремою Кронекера–Капеллі, оскільки ранг основної матриці системи не зміниться, якщо до неї приєднати нульовий стовпець вільних членів;
- розв'язок однорідної СЛАР, за яким $x_j = 0 \quad \forall j = \overline{1,n}$, називається *тривіальним*;
- якщо однорідна система має ***ненульовий*** розв'язок, тобто хоча б одне з чисел $x_j = \alpha_j \neq 0$, $j = \overline{1,n}$, то вона має нескінченну кількість розв'язків вигляду: $x_j = c\alpha_j$, де c – довільна стала, тобто така система є невизначеною;
- якщо $\mathbf{X}_1 = (\alpha_i)_{n\times 1}$ і $\mathbf{X}_2 = (\beta_i)_{n\times 1}$ є розв'язками однорідної системи: $\mathbf{A}\cdot\mathbf{X}_1 = \mathbf{O}$, $\mathbf{A}\cdot\mathbf{X}_2 = \mathbf{O}$, то їх лінійна комбінація $\mathbf{X} = c_1\mathbf{X}_1 + c_2\mathbf{X}_2$, де c_1 та c_2 – довільні числа, теж є розв'язком цієї системи. Це є наслідком властивостей матричних операцій; дійсно, завдяки властивостям, операціям над матрицями, матимемо: $\mathbf{A}\cdot\mathbf{X} = \mathbf{A}\cdot(c_1\mathbf{X}_1 + c_2\mathbf{X}_2) = c_1(\mathbf{A}\cdot\mathbf{X}_1) + c_2(\mathbf{A}\cdot\mathbf{X}_2) = \mathbf{O}$;
- важливим при дослідженні систем є випадок, коли *система має безліч розв'язків*, тобто ранги основної та розширеної матриць системи збігаються, але є меншими за кількість невідомих: $\operatorname{rang}(\mathbf{A}) = \operatorname{rang}(\mathbf{A}\,|\,\mathbf{B}) < n$.

Д.4.2.4 Виберемо в основній матриці системи базисний мінор r-го порядку (r – ранг матриці і $r < n$). Не порушуючи загальності, вважатимемо, що базисний мінор розташований у лівому верхньому куті основної матриці (у протилежному випадку рівняння можна поміняти місцями і змінити номери невідомих). Це значить, що r перших рівнянь системи є лінійно незалежними, тобто вони містять всю необхідну інформацію, а решту рівнянь можна отримати з перших r шляхом елементарних перетворень розширеної матриці. Тому доцільно залишити лише ті r рівнянь, які відповідають базисному мінору. У цих рівняннях невідомі $x_j\left(j = \overline{1,r}\right)$, коефіцієнти при яких утворюють базисний мінор, залишимо у лівих частинах рівнянь, а інші невідомі $x_j\left(j = \overline{r+1,n}\right)$ перенесемо у праві частини:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1r}x_r = b_1 - a_{1r+1}x_{r+1} - a_{1r+2}x_{r+2} - \ldots - a_{1n}x_n, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2r}x_r = b_2 - a_{2r+1}x_{r+1} - a_{2r+2}x_{r+2} - \cdots - a_{2n}x_n, \\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots \\ a_{r1}x_1 + a_{r2}x_2 + \cdots + a_{rr}x_r = b_r - a_{rr+1}x_{r+1} - a_{rr+2}x_{r+2} - \cdots - a_{rn}x_n. \end{cases} \qquad (Д.36)$$

Невідомі $x_j\left(j=\overline{1,r}\right)$, коефіцієнти при яких є елементами базисного мінору, називають *базисними,* а інші невідомі – *вільними*. Розв'яжемо рівняння (Д.33) відносно базисних невідомих та одержимо розв'язок системи у вигляді:

$$\begin{cases} x_1 = f_1(x_{r+1},\, x_{r+2},..., x_n, b_1, b_2,..., b_r), \\ x_2 = f_2(x_{r+1},\, x_{r+2},..., x_n, b_1, b_2,..., b_r), \\ ..... \\ x_r = f_r(x_{r+1},\, x_{r+2},..., x_n, b_1, b_2,..., b_r), \end{cases} \qquad (Д.37)$$

де $f_1, f_2,..., f_r$ – лінійні функції від вільних невідомих $x_{r+1}, x_{r+2},..., x_n$ і параметрів $b_1, b_2,..., b_r$ – вільних членів перших r рівнянь.

Розв'язок (Д.37) можна отримати за допомогою оберненої матриці, оскільки визначником основної матриці системи є базисний мінор. Розв'язок (Д.37), що містить вільні невідомі, через які виражаються базисні невідомі, називається *загальним розв'язком* системи. Частинним розв'язком системи називається розв'язок, який отримується із загального при конкретних (фіксованих) значеннях вільних невідомих. Частинний розв'язок системи, знайдений при нульових значеннях вільних невідомих, називають *базисним розв'язком*. Якщо в базисних розв'язках усі змінні набувають невід'ємних значень, то такий частинний розв'язок називають *опорним*.

Розглянемо розв'язання систем n лінійних рівнянь з n невідомими:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1; \\ a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n = b_2; \\ ..... \\ a_{n1}x_1 + a_{n2}x_2 + ... + a_{nn}x_n = b_n. \end{cases} \qquad (Д.38)$$

Теорема Д.6 (*правило Крамера*). Якщо визначник n-го порядку основної матриці системи алгебраїчних рівнянь (Д.38) відмінний від нуля ($\Delta \neq 0$), то вона має єдиний розв'язок, який знаходиться за формулою:

$$x_j = \frac{\Delta_j}{\Delta},\ \ j=\overline{1,n} \sim x_1 = \frac{\Delta_1}{\Delta}, x_2 = \frac{\Delta_2}{\Delta}, \ldots, x_n = \frac{\Delta_n}{\Delta}, \qquad (Д.39)$$

де Δ_j – визначник, який отримано з визначника основної матриці системи заміною j-го стовпця стовпцем вільних членів $(b_i)_{n\times 1}$.

Із формул (Д.39) випливає:

- правило Крамера виключає з розгляду випадок, коли $\Delta = 0$, оскільки якщо $\Delta = 0$, але хоча б один із визначників Δ_j, $j=\overline{1,n}$, відмінний від нуля, то система несумісна, адже принаймні один із дробів у (Д.25) втрачає сенс;

• якщо $\Delta = 0$ і $\Delta_j = 0 \ \ \forall j = \overline{1, n}$, то система невизначена і має безліч розв'язків.

Розглянемо приклад розв'язання СЛАР:

$$\begin{cases} 4x_1 + 3x_2 + 2x_3 = 33; \\ 3x_1 + 2x_2 + x_3 = 23; \\ x_1 + x_2 + 2x_3 = 12. \end{cases} \qquad (Д.40)$$

За правилом Крамера обчислимо визначник основної матриці системи:

$$\Delta = \begin{vmatrix} 4 & 3 & 2 \\ 3 & 2 & 1 \\ 1 & 1 & 2 \end{vmatrix} = -1 \neq 0 .$$

Оскільки матриця є невиродженою, то система рівнянь має єдиний розв'язок. Обчислюємо визначники, що відповідають невідомим:

$$\Delta_1 = \begin{vmatrix} 33 & 3 & 2 \\ 23 & 2 & 1 \\ 12 & 1 & 2 \end{vmatrix} = -5; \quad \Delta_2 = \begin{vmatrix} 4 & 33 & 2 \\ 3 & 23 & 1 \\ 1 & 12 & 2 \end{vmatrix} = -3, \quad \Delta_3 = \begin{vmatrix} 4 & 3 & 33 \\ 3 & 2 & 23 \\ 1 & 1 & 12 \end{vmatrix} = -2 .$$

За правилом Крамера знаходимо розв'язок системи:

$$x_1 = \frac{\Delta_1}{\Delta} = 5, \quad x_2 = \frac{\Delta_2}{\Delta} = 3 \quad x_3 = \frac{\Delta_3}{\Delta} = 2 .$$

Отже, розв'язком системи є трійка чисел: $x_1 = 5$, $x_2 = 3$, $x_3 = 2$.

Оскільки ми вже знаємо, що в ситуації, коли існує обернена матриця до основної матриці системи, розв'язок СЛАР (що містить n рівнянь відносно n невідомих) можна знайти за допомогою оберненої матриці, то покажемо тут, як це реалізувати до (Д.40) ще й методом оберненої матриці.

Обернена матриця існує, оскільки вище вже було встановлено, що визначник основної матриці системи не дорівнює нулю, а саме:

$$\Delta = \begin{vmatrix} 4 & 3 & 2 \\ 3 & 2 & 1 \\ 1 & 1 & 2 \end{vmatrix} = -1 \neq 0.$$

Визначимо тепер алгебраїчні доповнення і побудуємо союзну матрицю

$$\mathbf{A}^* = \begin{pmatrix} 3 & -4 & -1 \\ -5 & 6 & 2 \\ 1 & -1 & -1 \end{pmatrix}.$$

Помноживши матрицю A^* на сталу $1/\Delta$, отримаємо обернену матрицю:

$$\mathbf{A}^{-1} = \frac{\mathbf{A}^*}{\Delta} = \begin{pmatrix} -3 & 4 & 1 \\ 5 & -6 & -2 \\ -1 & 1 & 1 \end{pmatrix}.$$

Перевіримо правильність обчислень:

$$\mathbf{A} \cdot \mathbf{A}^{-1} = \begin{pmatrix} 4 & 3 & 2 \\ 3 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} -3 & 4 & 1 \\ 5 & -6 & -2 \\ -1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \mathbf{E},$$

$$\mathbf{A}^{-1} \cdot \mathbf{A} = \begin{pmatrix} -3 & 4 & 1 \\ 5 & -6 & -2 \\ -1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 4 & 3 & 2 \\ 3 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \mathbf{E}.$$

Отже, обернена матриця знайдена правильно. Тепер знаходимо розв'язок:

$$\mathbf{X} = \mathbf{A}^{-1} \cdot \mathbf{B} = \begin{pmatrix} -3 & 4 & 1 \\ 5 & -6 & -2 \\ -1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 33 \\ 23 \\ 12 \end{pmatrix} = \begin{pmatrix} 5 \\ 3 \\ 2 \end{pmatrix} \begin{pmatrix} 33 \\ 23 \\ 12 \end{pmatrix} = \begin{pmatrix} 5 \\ 3 \\ 2 \end{pmatrix}.$$

Таким чином, розв'язком системи буде єдина трійка чисел: $\begin{pmatrix} 5 \\ 3 \\ 1 \end{pmatrix}$.

Застосовують до розв'язання систем лінійних рівнянь і *метод послідовного виключення невідомих* (*метод Гаусса*) та його модифікацію – *метод повного виключення невідомих* (*метод Жордана–Гаусса*). Перевагою цього методу є те, що дослідження системи рівнянь на сумісність здійснюється у поєднанні з відшуканням її розв'язку. Суть методу Гаусса полягає у тому, що за допомогою елементарних перетворень розширену матрицю вихідної системи лінійних алгебраїчних рівнянь за скінчене число кроків (*ітерацій*) зводять до трикутного або трапецієподібного вигляду. Такі перетворення називаються *прямим ходом* методу Гаусса. Далі відбувається знаходження невідомих шляхом послідовної підстановки знайдених значень невідомих у рівняння з меншим номером i що називають *зворотним ходом* методу Гаусса. При застосуванні методу Гаусса прямий хід виконують на розширеній матриці або у спеціальній таблиці, де записані елементи розширеної матриці, тоді як зворотний – здійснюють на системі рівнянь.

Якщо внаслідок елементарних перетворень основна матриця системи набуває трикутного вигляду (її порядок збігається з кількістю невідомих),

то система рівнянь має єдиний розв'язок. У випадку трапецієподібної матриці система сумісна, але невизначена.

Якщо при виконанні прямого ходу методу Гаусса одержано суперечливе рівняння: $0 \cdot x_1 + 0 \cdot x_2 + ... + 0 \cdot x_n = b_i$, де $b_i \neq 0$, то така система несумісна.

Виконання зворотного ходу за методом Гаусса, якщо його розглядати як подальше перетворення розширеної матриці системи, еквівалентне зведенню основної матриці системи до одиничної. Саме у цьому полягає суть методу *повного виключення невідомих,* або *методу Жордана–Гаусса.*

Якщо за допомогою елементарних перетворень звести основну матрицю системи до одиничної матриці n-го порядку (у цьому випадку $\text{rang}(\mathbf{A}) = \text{rang}(\mathbf{A} \mid \mathbf{B}) = n$), тобто:

$$\begin{cases} x_1 = b_1^{(n)} \\ x_2 = b_2^{(n)} \\ \cdots \\ x_n = b_n^{(n)} \end{cases} \qquad \text{(Д.41)}$$

то отримаємо єдиний розв'язок системи $\mathbf{X}^T = (b_1^{(n)}, b_2^{(n)}, ..., b_n^{(n)})$.

Якщо ж $\text{rang}(\mathbf{A}) = \text{rang}(\mathbf{A} \mid \mathbf{B}) < n$, то результатом перетворень за методом Жордана – Гаусса є отримання в основній матриці системи одиничної матриці r-го порядку

$$\begin{cases} x_1 + a_{1r+1}^{(r)} x_{r+1} + \cdots + a_{1n}^{(r)} x_n = b_1^{(r)}, \\ x_2 + a_{2r+1}^{(r)} x_{r+1} + \cdots + a_{2n}^{(r)} x_n = b_2^{(r)}, \\ \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \\ x_r + a_{rr+1}^{(r)} x_{r+1} + \cdots + a_{rn}^{(r)} x_n = b_r^{(r)}. \end{cases} \qquad \text{(Д.42)}$$

Такий вигляд (Д.42) системи лінійних алгебраїчних рівнянь дає можливість легко виразити базисні невідомі через вільні.

Додаток Е

ОСНОВИ ТЕОРІЇ КЕРОВАНОСТІ ЛІНІЙНИХ ЗВИЧАЙНИХ ДИФЕРЕНЦІАЛЬНИХ РІВНЯНЬ

Перш ніж доторкнутися до азів теорії керованості, слід володіти основними поняттями та фактами теорії лінійних звичайних диференціальних рівнянь та основними способами їх розв'язання [36].

Е.1 Досить часто для опису досліджуваних процесів недостатньо одного диференціального рівняння, тому використовується їх система. *Системою лінійних диференціальних рівнянь* називається сукупність n лінійних диференціальних рівнянь першого порядку, кожне з яких містить одні й ті самі невідомі функції $y_1, y_2, ..., y_n$ від незалежної змінної x та їх похідні:

$$\begin{cases} y_1' + a_{11}y_1 + a_{12}y_2 + \ldots + a_{1n}y_n = q_1, \\ y_2' + a_{21}y_1 + a_{22}y_2 + \ldots + a_{2n}y_n = q_2, \\ \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\ y_n' + a_{n1}y_1 + a_{n2}y_2 + \ldots + a_{nn}y_n = q_n, \end{cases} \qquad \text{(Е.1)}$$

де $a_{ij} = a_{ij}(x)$ – *коефіцієнти при невідомих*; $q_i = q_i(x)$, $i = \overline{1,n}$, $j = \overline{1,n}$ – *вільні члени*.

Стисло систему (Е.1) записують так:

$$y_i' + \sum_{j=1}^{n} a_{ij} y_j = q_i, \; i = \overline{1,n}.$$

Якщо рівняння системи (Е.1) розв'язані відносно похідних, то система лінійних диференціальних рівнянь називається *нормальною*.

Якщо коефіцієнти системи (Е.1) є сталими величинами, то така система називається *системою диференціальних рівнянь зі сталими коефіцієнтами.* При $q_i \equiv 0$, $i = \overline{1,n}$ система називається *однорідною,* у протилежному разі –*неоднорідною*.

Набір функцій $\phi_j = \phi_j(x)$, $j = \overline{1,n}$ називають *розв'язком системи* на інтервалі (a,b), якщо підстановка їх у рівняння системи замість $y_j(x)$ перетворює усі рівняння на тотожності.

Загальним розв'язком системи диференціальних рівнянь (Е.1) називається сукупність n диференційовних функцій від незалежної змінної x, які містять n довільних сталих $C_1, C_2, \ldots, C_n$ – числових параметрів:

$$\begin{aligned} y_1 &= y_1\left(x, C_1, C_2, \ldots, C_n\right), \\ y_2 &= y_2\left(x, C_1, C_2, \ldots, C_n\right), \\ \ldots \quad &\ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\ y_n &= y_n\left(x, C_1, C_2, \ldots, C_n\right), \end{aligned} \tag{E.2}$$

і таких, що:

- вони задовольняють рівняння системи (E.1) при будь-яких дійсних значеннях параметрів C_i, $i = \overline{1,n}$;
- для заданих початкових умов: $y_1\left(x_0\right) = b_1$, $y_2\left(x_0\right) = b_2$, ... , $y_n\left(x_0\right) = b_n$

завжди знайдуться значення сталих, що визначають розв'язок системи, який задовольняє початкові умови.

Розв'язок системи, отриманий із загального розв'язку при фіксованих значеннях параметрів $C_1, C_2, \ldots, C_n$, називається *частинним*.

Під *задачею Коші для системи лінійних диференціальних рівнянь* розуміють знаходження такого частинного розв'язку системи (E.1), який задовольняє початкові умови: $y_1\left(x_0\right) = b_1$, $y_2\left(x_0\right) = b_2$, ... , $y_n\left(x_0\right) = b_n$.

E.2 Одним із способів розв'язання системи (E.1) є *метод виключення,* суть якого полягає у зведенні системи n диференціальних рівнянь 1-го порядку до одного диференціального рівняння n-го порядку відносно однієї з невідомих функцій (досягається шляхом диференціювання рівняння системи з подальшим використанням у вигляді підстановки інших рівнянь цієї ж системи в отриманий вираз). Маючи в розпорядженні розв'язок цього рівняння, знаходять інші невідомі функції.

У прикладних задачах часто невідомі функції є функціями часу t: $y_j = y_j(t)$; їх похідні за змінною t позначають через $\dot{y}_j$.

Стосовно нормальних систем диференціальних рівнянь зі сталими коефіцієнтами:

$$\dot{y}_j = \sum_{j=1}^{n} a_{ij} y_j, \quad j = \overline{1,n} \tag{E.3}$$

покажемо так званий *алгебраїчний метод* (*Ейлера***)** розв'язання системи лінійних диференціальних рівнянь. Нехай задано однорідну нормальну систему диференціальних рівнянь першого порядку:

$$\begin{cases} \dot{y}_1 = a_{11} y_1 + a_{12} y_2 + \ldots + a_{1n} y_n, \\ \dot{y}_2 = a_{21} y_1 + a_{22} y_2 + \ldots + a_{2n} y_n, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \dot{y}_n = a_{n1} y_1 + a_{n2} y_2 + \ldots + a_{nn} y_n, \end{cases} \tag{E.4}$$

де a_{ij} – сталі величини, $y_i = y_i\left(t\right)$ – невідомі функції $\left(i, j = \overline{1,n}\right)$.

Систему (Е.4) зручно стисло записувати у вигляді матричного рівняння:

$$\dot{\mathbf{Y}} = \mathbf{A} \cdot \mathbf{Y},$$

де $\dot{Y} = \begin{pmatrix} \dot{y}_1 \\ \dot{y}_2 \\ \vdots \\ \dot{y}_n \end{pmatrix}$, $A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}$, $Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$.

Суть методу Ейлера полягає в наступному: частинні розв'язки системи знаходять у вигляді сукупності показникових функцій:

$$y_1 = \alpha_1 e^{kt},\ y_2 = \alpha_2 e^{kt},\ ...,\ y_n = \alpha_n e^{kt},$$

де k, α_i $(i = \overline{1,n})$ – невизначені сталі, які підбирають так, щоб функції $\phi_i(t) = \alpha_i e^{kt}$ задавали розв'язок системи рівнянь.

Це приводить до задачі лінійної алгебри знаходження власних чисел і власних векторів матриці $\mathbf{A}$. Дійсно, підставляючи в систему (Е.4) замість y_i функції $\alpha_i e^{kt}$ $(i = \overline{1,n})$ і скорочуючи кожне рівняння на множник $e^{kt} \neq 0$, отримаємо систему однорідних лінійних алгебраїчних рівнянь відносно сталих $\alpha_1, \alpha_2, \dots, \alpha_n$:

$$\begin{cases} (a_{11} - k)\alpha_1 + a_{12}\alpha_2 + \dots + a_{1n}\alpha_n = 0, \\ a_{21}\alpha_1 + (a_{22} - k)\alpha_2 + \dots + a_{2n}\alpha_n = 0, \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ a_{n1}\alpha_1 + a_{n2}\alpha_2 + \dots + (a_{nn} - k)\alpha_n = 0 \end{cases} \qquad \text{(Е.5)}$$

з визначником

$$\Delta = \begin{vmatrix} a_{11} - k & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} - k & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} - k \end{vmatrix}. \qquad \text{(Е.6)}$$

Якщо k таке, що визначник $\Delta \neq 0$, то система рівнянь (Е.5) має тільки тривіальний розв'язок: $y_1 = y_2 = \dots = y_n = 0$. Нетривіальний розв'язок система (Е.5) матиме лише для тих k, при яких визначник (Е.6) системи дорівнює нулю. Отже, для визначення k приходимо до алгебраїчного рівняння n-го степеня:

$$\begin{vmatrix} a_{11} - k & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} - k & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} - k \end{vmatrix} = 0 \cdot \qquad \text{(Е.7)}$$

Це рівняння називається *характеристичним рівнянням системи диференціальних рівнянь*, його корені – *коренями характеристичного рівняння*. Розглянемо випадок дійсних і різних коренів рівняння (Е.7). Для кожного кореня k_i $(i=\overline{1,n})$ записують систему (Е.5) і визначають коефіцієнти $\alpha_1^{(i)},\alpha_2^{(i)},\ldots,\alpha_n^{(i)}$. Оскільки ранг матриці системи дорівнює $(n-1)$, то один із коефіцієнтів можна вибрати довільно. Звичайно його приймають рівним одиниці. Після розв'язання системи алгебраїчних рівнянь (Е.5) дістаємо частинні розв'язки системи диференціальних рівнянь:
для k_1:

$$y_1^{(1)}=\alpha_1^{(1)}e^{k_1t},\ y_2^{(1)}=\alpha_2^{(1)}e^{k_1t},\ ...,\ y_n^{(1)}=\alpha_n^{(1)}e^{k_1t};$$

для k_2:

$$y_1^{(2)}=\alpha_1^{(2)}e^{k_2t},\ y_2^{(2)}=\alpha_2^{(2)}e^{k_2t},\ ...,\ y_n^{(2)}=\alpha_n^{(2)}e^{k_2t};$$

.....

для k_n:

$$y_1^{(n)}=\alpha_1^{(n)}e^{k_nt},\ y_2^{(n)}=\alpha_2^{(n)}e^{k_nt},\ ...,\ y_n^{(n)}=\alpha_n^{(n)}e^{k_nt}.$$

Сукупність (систему) n частинних розв'язків:

$$\begin{bmatrix} y_1^{(1)}(t),\ y_1^{(2)}(t),\ ...,\ y_1^{(n)}(t), \\ y_2^{(1)}(t),\ y_2^{(2)}(t),\ ...,\ y_2^{(n)}(t), \\ ..... \\ y_n^{(1)}(t),\ y_n^{(2)}(t),\ ...,\ y_n^{(n)}(t) \end{bmatrix}$$

називають *фундаментальною системою розв'язків* на інтервалі (a,b), якщо $\forall t\in(a,b)$ відповідний визначник не дорівнює нулю:

$$\Delta(t)=\begin{vmatrix} y_1^{(1)}(t) & y_1^{(2)}(t) & ... & y_1^{(n)}(t) \\ y_2^{(1)}(t) & y_2^{(2)}(t) & ... & y_2^{(n)}(t) \\ ... & ... & ... & ... \\ y_n^{(1)}(t) & y_n^{(2)}(t) & ... & y_n^{(n)}(t) \end{vmatrix}\neq 0.$$

Відповідно, загальний розв'язок системи (Е.4) має вигляд:

$$y_j(t)=C_1\alpha_j^{(1)}e^{k_1t}+C_2\alpha_j^{(2)}e^{k_2t}+...+C_n\alpha_j^{(n)}e^{k_nt},\ j=\overline{1,n}, \qquad (Е.8)$$

де $C_1,C_2,\ldots,C_n$ – довільні сталі.

Для неоднорідної нормальної системи рівнянь

$$y_i'=\sum_{j=1}^{n}a_{ij}y_j+q_i,\ i=\overline{1,n}, \qquad (Е.9)$$

або в матричній формі

$$\dot{\mathbf{Y}} = \mathbf{A} \cdot \mathbf{Y} + \mathbf{Q},$$

де $\mathbf{Q} = (q_1, q_2, ..., q_n)^T$ – матриця вільних членів (відомих функцій) $q_i(t)$, структура загального розв'язку системи така сама, як і для одного неоднорідного рівняння n-го порядку зі сталими коефіцієнтами:

$$\mathbf{Y} = \overline{\mathbf{Y}} + \tilde{\mathbf{Y}},$$

де $\overline{\mathbf{Y}} = (\overline{y}_1, \overline{y}_2, ..., \overline{y}_n)^T$ – матриця загального розв'язку відповідної однорідної системи рівнянь; $\tilde{\mathbf{Y}} = (\tilde{y}_1, \tilde{y}_2, ..., \tilde{y}_n)^T$ – матриця будь-якого частинного розв'язку неоднорідної системи рівнянь.

Якщо функції $\overline{y}_i = \overline{y}_i(t, C_1, C_2, ..., C_n)$ знайдені, то для знаходження частинного розв'язку неоднорідної системи рівнянь – матриці $\tilde{Y}$ – можна застосувати метод варіації довільних сталих, що зводиться до розв'язання системи рівнянь відносно похідних функцій $C_i = C_i(t)$, $i = \overline{1,n}$:

$$\begin{cases} C_1'(t)y_1(t) + C_2'(t)y_2(t) + \ldots + C_n'(t)y_n(t) = q_1(t), \\ C_1'(t)y_1'(t) + C_2'(t)y_2'(t) + \ldots + C_n'(t)y_n'(t) = q_2(t), \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ C_1'(t)y_1^{(n-1)}(t) + C_2'(t)y_2^{(n-1)}(t) + \ldots + C_n'(t)y_n^{(n-1)}(t) = q_n(t). \end{cases} \qquad \text{(E.10)}$$

E.3 Спостережуваність, керованість та ідентифікованість грають важливу роль при синтезі систем управління динамічними об'єктами, оцінюванні стану та ідентифікації їх параметрів [38]. Поняття спостережливості й керованості дуальні (тобто якщо система повністю спостережувана, то побудована для цієї системи сполучена система буде повністю керована, крім того, справедливо і зворотне твердження). Отже, для визначення спостережливості можна побудувати спряжену систему, яка буде досліджуватися за допомогою будь-якого критерію керованості, що істотно розширює методологічний апарат для дослідження всіх вищезазначених характеристик. Одним з дуже популярних критеріїв є критерій Калмана [39–41], який відрізняється простотою і широко використовується в практичних застосуваннях.

Нехай об'єкт описується рівняннями вигляду

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A} \cdot \mathbf{x}(t) + \mathbf{B} \cdot \mathbf{u}(t), \\ \mathbf{z}(t) &= \mathbf{C} \cdot \mathbf{x}(t) + \mathbf{D} \cdot \mathbf{u}(t), \end{aligned} \qquad \text{(E.10)}$$

де $\dot{\mathbf{x}}(t) \equiv \dfrac{d\mathbf{x}(t)}{dt}$; $\mathbf{x}$ – n-вектор стану; $\mathbf{u}$ – m-вектор керування (вектор входу); $\mathbf{z}$ – l-вектор вимірювань (вектор виходу); $\mathbf{A}$ – $n \times n$-матриця

системи; $\mathbf{B}$ – $n \times m$-матриця керування; $\mathbf{C}$ – $l \times n$-матриця вимірювань (матриця виходу); $\mathbf{D}$ – $l \times m$-матриця прямого зв'язку (наскрізна матриця).

Тут слід зазначити, що матриця $\mathbf{D}$ часто є нульовою, це в свою чергу означає, що в системі немає явного прямого зв'язку. Система (Е.11) називається *повністю спостережуваною на інтервалі часу* $[0, t_1]$, якщо вектор стану $\mathbf{x}(t_0)$ можна визначити за відомим вектором вимірювань $\mathbf{z}(t)$. Складемо матрицю керованості (яку, до речі, також називають і матрицею спостережень) та позначимо її через $\mathbf{S}$:

$$\mathbf{S} = \left[\mathbf{B}\, \mathbf{A}\mathbf{B}\, \mathbf{A}^2\mathbf{B} \ldots \mathbf{A}^{n-1}\mathbf{B} \right].$$

Теорема Е.1 (*критерій Калмана*). Система (Е.11) є повністю спостережуваною, якщо ранг матриці спостереження $\mathbf{S}$ дорівнює порядку системи n:

$$\operatorname{rang}(\mathbf{S}) = n,$$

тобто якщо вимірювання $\mathbf{z}(t)$ містять достатню інформацію для визначення $\mathbf{x}(t_0)$.

Зауважимо, що у разі, якщо ранг матриці спостереження менше порядку системи, то за вимірюваннями $\mathbf{z}(t)$ можна оцінити лише частину вектора стану $\mathbf{x}(t)$.

Процес побудови функції керованості та обмеженого керування u, що вирішують проблему синтезу, детально викладено та супроводжується прикладами в монографії [37]: для одновимірного та багатовимірного керування слід звернутися до глави 1, параграфи 5 і 6 відповідно. Крім того, автором книги [37] В. І. Коробовим вирішена задача синтезу для певного класу нелінійних систем, які за допомогою заміни фазових змінних і введення нового керування відображаються на лінійні керовані системи.

Додаток Ж

ДИФЕРЕНЦІАЛЬНЕ РІВНЯННЯ РАДІОАКТИВНОГО РОЗПАДУ

Радіоактивний розпад – це процес мимовільної зміни хімічного стану нестійкої речовини, який відбувається через явище радіоактивності, що являє собою мимовільне випромінювання речовиною потоків елементарних частинок (нейтронів, електронів та інших), які відділяються від нестійких атомів цієї речовини [49]. Зрозуміло, що не всі речовини мають властивість радіоактивності, оскільки така властивість істотно ґрунтується на нестійкості атомів, яка притаманна лише деяким речовинам із великими масовим числом, наприклад урану.

Якщо в деякий момент часу t маємо N атомів радіоактивної речовини, то протягом часу Δt кількість таких атомів зменшиться внаслідок радіоактивного розпаду на величину ΔN внаслідок втрати атомами елементарних частинок, таких як електрони, нейтрони або альфа-частинки. Цілком природно, що розпад будь-яких атомів із наявної кількості однакових атомів є рівноімовірним, протилежний випадок свідчить про наявність відмінності атомів [49]. Тобто імовірність розпаду за одиницю часу є сталою [49]:

$$-\frac{1}{\Delta t}\frac{\Delta N}{N}=\lambda, \qquad (Ж.1)$$

де $\lambda>0$ – стала радіоактивного розпаду, яка є характеристикою речовини, а від'ємність враховує зменшення кількості атомів.

Співвідношення (Ж.1) можна тотожно перетворити до наступного вигляду:

$$\frac{\Delta N}{\Delta t}=-\lambda N. \qquad (Ж.2)$$

Межа співвідношення (Ж.2) при $\Delta t\to 0$ приводить до добре відомого [49] диференціального рівняння радіоактивного розпаду:

$$\frac{dN}{dt}=-\lambda N. \qquad (Ж.3)$$

Диференціальне рівняння (Ж.3) слід розглядати із початковою умовою, яка має визначити поточну кількість речовини у деякий момент часу, наприклад у початковий момент часу $t=0$:

$$N(0)=N_0, \qquad (Ж.4)$$

де N_0 – задана кількість атомів радіоактивної величини у момент часу $t=0$.

Розв'язуванням диференціального рівняння (Ж.3) із початковою умовою (Ж.4) можемо встановити кількість атомів радіоактивної речовини у будь-який момент часу $t \geq 0$. Таке розв'язування насправді є нескладним. Дійсно, диференціальне рівняння (Ж.3) можна представити у вигляді:

$$\frac{dN}{N} = -\lambda dt . \qquad (Ж.5)$$

Далі можна виконати інтегрування у співвідношенні (Ж.5) з урахуванням початкової умови (Ж.4):

$$\int_{N_0}^{N} \frac{d\mu}{\mu} = -\lambda \int_{0}^{t} d\tau \quad \Rightarrow \quad \ln \mu \Big|_{N_0}^{N} = -\lambda \tau \Big|_{0}^{t} \quad \Rightarrow \quad \ln N - \ln N_0 = -\lambda t . \qquad (Ж.6)$$

Після нескладних перетворень останнього співвідношення (Ж.6) маємо можливість одержати шуканий розв'язок у вигляді:

$$N(t) = N_0 e^{-\lambda t} . \qquad (Ж.7)$$

Точний розв'язок (Ж.7) повністю відповідний результату (4.22) і є цікавим для тестування програм розв'язування звичайних диференціальних рівнянь, що саме і було використано вище у п. 4.4.3. Можливість моделювання явища радіоактивного розпаду дозволяє використовувати його в археології та палеонтології для історичного датування знайдених артефактів.

Для визначення змісту константи λ, введеної у співвідношенні (Ж.1), визначимо час T, протягом якого розпадеться половина початкової кількості атомів радіоактивної речовини. Для цього використаємо зрозуміле співвідношення, що випливає із розв'язку (Ж.7):

$$\frac{N_0}{2} = N_0 e^{-\lambda T} . \qquad (Ж.8)$$

Із співвідношення (Ж.8) одержимо:

$$\lambda = \frac{\ln 2}{T} . \qquad (Ж.9)$$

Завдяки співвідношенню (Ж.9) маємо наочний зміст параметра λ; зазначимо, що введену до розгляду величину T називають періодом напівроспаду і саме цю величину наводять зазвичай у фізичних довідниках щодо властивостей радіоактивних речовин.