МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет кораблебудування
імені адмірала Макарова

І. Л. МИХЕЛЄВ, С. О. СЛОБОДЯН

ОСНОВИ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Навчальний посібник

Рекомендовано Міністерством освіти і науки України

Миколаїв ♦ НУК ♦ 2014

УДК 004.422:519.6(075.8)
ББК 73:22.18я73
М 69

Автори:
І. Л. Михелєв, канд. техн. наук, доцент;
С. О. Слободян, канд. техн. наук, доцент

Рецензенти:
А. Я. Казарєзов, доктор технічних наук, професор, завідувач кафедри економічної теорії та економетрії Чорноморського державного університету імені П. Могили;
І. І. Коваленко, доктор технічних наук, професор кафедри програмного забезпечення автоматизованих систем Національного університету кораблебудування імені адмірала Макарова

Рекомендовано Міністерством освіти і науки України як навчальний посібник для студентів вищих навчальних закладів (*лист № 1/11–11792 від 22.07.2013*)

Михелєв І. Л.
М 69 Основи інформаційних технологій : навчальний посібник / І. Л. Михелєв, С. О. Слободян. – Миколаїв : НУК, 2014. – 168 с.

ISBN 978–966–321–276–0

Посібник містить програму першої та другої частин курсу "Основи інформаційних технологій та програмування", основи теоретичного матеріалу, алгоритми реалізації базових обчислювальних процесів на ЕОМ, приклади розв'язання задач, алгоритми та приклади реалізації класичних чисельних методів.

Призначено для студентів, що вивчають курс "Основи інформаційних технологій та програмування".

УДК 004.422:519.6(075.8)
ББК 73:22.18я73

ISBN 978–966–321–276–0

ЧАСТИНА ПЕРША

ОСНОВИ ПРОГРАМУВАННЯ

Людина завжди намагалась збільшити швидкість і точність своїх обчислень. Ще задовго до першого тисячоліття до н. е. у Середземномор'ї та на Сході були поширені найпростіші пристрої для обчислень: дощечки, покриті шаром піску, з камінчиками або бусинками. Такі рахункові дощечки називались *абак*. Багато років вони удосконалювались, поліпшувалася техніка їх використання. При цьому ніяких нових обчислювальних пристроїв до кінця XVI сторіччя створено не було. У 1594 році Джон Непер винайшов логарифми, і з'явились нові обчислювальні пристрої – "*палички Непера*". Декількома роками пізніше Вільям Отред удосконалив процес обчислень, сполучивши дві логарифмічні шкали. Таким чином, з'явилася *логарифмічна лінійка*. У 1642 році Блез Паскаль винайшов механічну обчислювальну машину, яка могла виконувати операції додавання, а приблизно 40 років потому Готфрід Лейбніць розширив можливості машини Паскаля, додавши операцію множення.

Жоден з цих пристроїв не був автоматичним, оскільки всі вони вимагали постійного втручання людини. Але поява цих машин порушила питання про можливості створення цілком автоматичної обчислювальної машини, яка була б здатна виконувати обчислення без втручання людини. У 1834 році Ч. Беббідж вперше запропонував проект автоматичної машини для обчислень, назвавши її аналітичною. Беббіджу так і не вдалось побудувати свою аналітичну машину, однак саме його ідеї заклали фундамент, на якому згодом з'явились електронно-обчислювальні машини (ЕОМ).

Перша цілком автоматична обчислювальна машина була винайдена Айкеном при участі фірми *IBM* у 1944 році. Вона одержала назву *ASCC* (*Automatic Sequence Controlled Calculator* – автоматичний послідовний керований обчислювач) і була електромеханічною. За сучасними стандартами

ця машина була вкрай повільною, оскільки витрачала на множення порядку чотирьох секунд. Перша цілком електронна обчислювальна машина була створена в США у 1946 році і названа *ENIAC* (*Electronic Numerical Integrator and Calculator* – електронний числовий інтегратор і обчислювач). Машина *ENIAC* була приблизно в тисячу разів швидшою за *ASCC*, але тримати в робочому стані її 18000 електронних ламп протягом тривалого часу було досить складно.

Останні десятиліття відзначені стрімким розвитком обчислювальної техніки. Зменшуються розміри комп'ютерів, споживання ними електроенергії, а швидкість обчислень зростає. На сьогоднішній день персональні комп'ютери (ПЕОМ) ефективно використовуються у всіх сферах людської діяльності, що вимагає поширення комп'ютерної грамотності та використання ПЕОМ у навчальному процесі.

Широке і різноманітне застосування ПЕОМ висуває високі вимоги до їх програмного забезпечення. У той же час процес створення програм відноситься до однієї з найбільш складних сфер творчої діяльності людини та вимагає зусиль і спеціальних знань.

При створенні програм застосовуються різні алгоритмічні мови програмування, серед яких мова *Pascal*, що завоювала популярність серед користувачів, займає особливе місце. Ця мова програмування була вперше запропонована в 1970 році професором Ніклаусом Віртом (Цюріх, Швейцарія) і названа на честь Блеза Паскаля. Особлива роль мови *Pascal* у програмуванні полягає в тому, що вона заклала основи сучасної методології програмування і повною мірою відображає найбільш важливі концепції алгоритмів.

1. ОСНОВНІ ПРИНЦИПИ РОЗРОБКИ АЛГОРИТМІВ І ПРОГРАМ

1.1. Етапи розв'язання задач на ЕОМ

Розв'язання задачі на ЕОМ складається з декількох етапів, серед яких основними є наступні:

1) постановка задачі;

2) математична постановка задачі (формалізація);

3) вибір (або розробка) методу розв'язання задачі;

4) розробка алгоритму (алгоритмізація);

5) складання програми (власне програмування);

6) відлагодження програми;

7) обчислення та обробка результатів.

При *постановці задачі* увагу необхідно приділити виявленню кінцевої мети і виробленню загального підходу до досліджуваної проблеми. Необхідно вивчити загальні властивості явища або об'єкта, які розглядаються, виявити, чи існує розв'язок задачі, чи він єдиний. Правильно сформулювати задачу найчастіше не менш складно, ніж її розв'язати.

Формалізація, як правило, зводиться до побудови математичної моделі досліджуваного явища або об'єкта. Необхідно визначити обсяг і специфіку вхідних даних, ввести систему умовних позначень, вибрати відповідний математичний апарат.

При *виборі методу розв'язання задачі* слід враховувати складність формул того чи іншого чисельного методу і необхідну точність обчислень. Крім похибки самого чисельного методу, необхідно зважати на похибки, пов'язані з обмеженою точністю представлення чисел у пам'яті ЕОМ, а також на ті умовні передумови (припущення), які вводяться на етапі формалізації.

Розробка алгоритму полягає в розкладанні обчислювального процесу на можливі складові частини і визначення порядку їх виконання.

Складання програми полягає в представленні алгоритму в такій формі, яка допускається для введення в ЕОМ з ціллю наступного перекладу на машинну мову.

Відлагодження програми полягає в пошуку синтаксичних і логічних (змістовних) помилок. Гарантією правильності розв'язку задачі на ЕОМ може служити, наприклад: а) перевірка виконання умов задачі; б) якісний аналіз задачі; в) перерахування вручну або іншим чисельним методом.

Тільки після того як з'явиться повна впевненість у правильності одержуваних за допомогою ЕОМ-програми результатів, можна виконувати масові розрахунки і аналіз отриманих результатів.

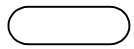
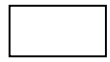
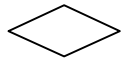
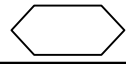
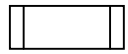
1.2. Схеми алгоритмів

Схема алгоритму – це його графічне представлення, доповнене елементами словесного запису.

Кожен пункт алгоритму відображається на схемі деякою геометричною фігурою, яка називається *блоком* або *блочним символом*. Блоки на схемах з'єднуються *лініями потоку інформації*. В середині блоків або поруч з ними роблять словесні записи, необхідні для уточнення функцій, які виконуються у блоках.

Розглянемо наведені у табл. 1.1 основні умовні графічні позначення (блоки), що застосовуються при складанні схем алгоритмів.

Таблиця 1.1. **Блочні символи найбільш частого застосування**

Назва блоку	Символ	Функція, що виконується
Початок – кінець або вхід – вихід		Позначає початок або кінець алгоритму, зупинку, вхід або вихід у стандартних алгоритмах
Блок введення – виведення		Загальне позначення введення або виведення даних незалежно від їх фізичного носія
Блок обчислень		Обчислювальна дія або послідовність обчислювальних дій
Логічний блок		Вибір напряму виконання алгоритму в залежності від певної умови або умов
Блок модифікації або заголовок циклу		Виконання дій, що змінюють пункти алгоритму (реалізація циклічного процесу)
Заздалегідь визначений процес		Використання стандартного алгоритму або алгоритму власної розробки
З'єднувач		Зв'язок між перерваними лініями потоку інформації у межах однієї сторінки
		Зв'язок між частинами блок-схеми, які розташовані на різних листах

Як **приклад** розглянемо схему відомого алгоритму обчислення дійсних коренів квадратного рівняння вигляду $ax^2 + bx + c = 0$ (рис. 1.1).

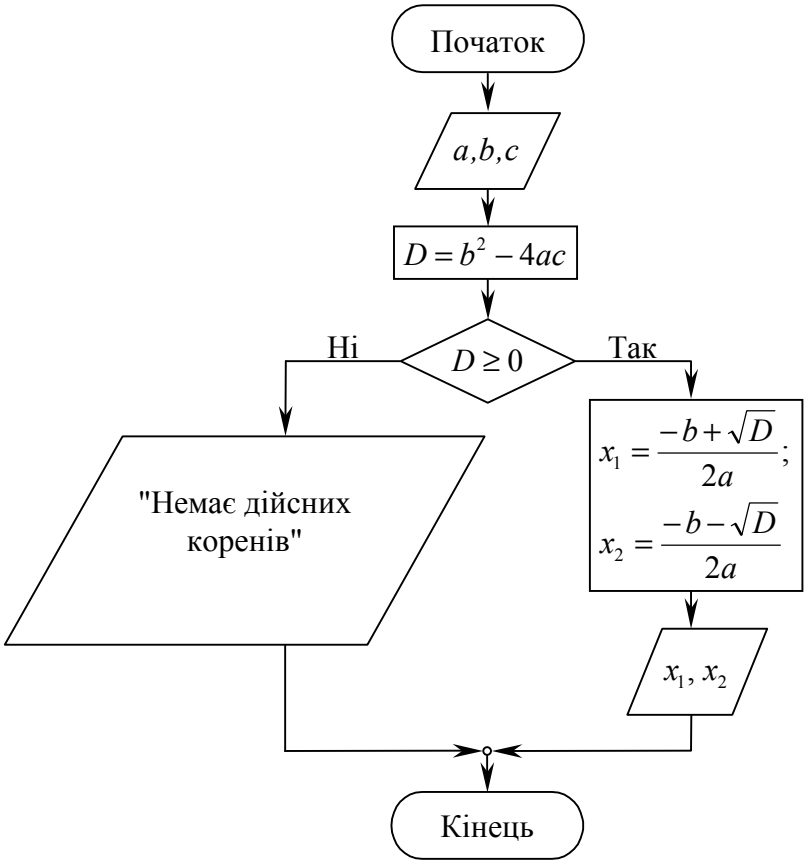


Рис. 1.1. Блок-схема алгоритму обчислення дійсних коренів квадратного рівняння

Для розв'язання цього рівняння на ЕОМ необхідно ввести значення коефіцієнтів a, b, c в оперативну пам'ять комп'ютера і потім обчислити значення дискримінанта. Якщо дискримінант більше або дорівнює нулю, то необхідно визначити корені рівняння і вивести їх (скажімо, на екран ЕОМ). Якщо значення дискримінанта від'ємне, то на екрані ЕОМ повинен з'явитися напис – "Немає дійсних коренів".

Схема є наочним і простим засобом представлення алгоритму. При цьому не накладається ніяких обмежень на ступінь деталізації алгоритму на схемі.

2. НАЙПРОСТІШІ КОНСТРУКЦІЇ МОВИ PASCAL

2.1. Основні поняття

При складанні програми мовою програмування можна використовувати тільки ті символи, які передбачені алфавітом мови.

Алфавіт мови Pascal складають:

а) великі та малі букви російського алфавіту (крім букв Ё і ё):
А, Б, В, Г, Д, Е, …, Я; а, б, в, г, д, е, …, я;

б) великі та малі букви англійського алфавіту, включаючи знак підкреслювання:
_, A, B, C, D, E, F, …, Z; a, b, c, d, e, f, …, z;

в) арабські цифри:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9;

г) спеціальні символи:

```
 *   –   /   \   {   }   (   )   [   ]   >   <   .   ,   :   ;   '   $   #
```

д) складені символи:

присвоєння	:=
порівняння	<> <= >=
підмножини	..
дужки	(* *) (. .)

Замість фігурних дужок { } можна використовувати дужки з зірочками (* *), а замість квадратних дужок [ ] – дужки з крапками (. .).

З окремих символів мови утворюються *слова*. Під словом у мові *Pascal* розуміють число, рядок або ім'я. Слова в тексті програми відокремлюються одне від одного пропусками або спеціальними символами та утворюють *словосполучення* і *вирази*. Якщо між словами знаходиться спеціальний (або складений) символ, то пропуск між ними можна не робити. Не можна ро-

бити пропуск в середині слова, а між словами їх може бути декілька. У залежності від призначення слова розділяються на службові та імена.

Службове слово (або *ключове слово*) складається з нерозривної послідовності букв і служить в основному для позначення операторів та розділів програми. У табл. 2.1 надається список ключових слів мови *Pascal*.

Таблиця 2.1. **Службові слова мови *Pascal***

absolute	external	nil	shl
and	file	not	shr
array	for	of	string
begin	forward	or	then
case	function	overlay	to
const	goto	packed	type
div	if	procedure	until
do	in	program	var
downto	inline	record	while
else	label	repeat	with
end	mod	set	xor

Ім'я (або *ідентифікатор*) – це послідовність символів та цифр, яка починається з букви і не є ключовим словом. Символом в ідентифікаторі може бути кожна (маленька або велика) буква англійського алфавіту і символ підкреслення. Кількість символів, що складають ідентифікатор, не повинна перевищувати 127.

Існують *стандартні* (*заздалегідь зарезервовані*) *ідентифікатори*. Вони служать для позначення констант, функцій і процедур. **Наприклад,** константа $\pi = 3{,}14159265\ldots$ в мові *Pascal* позначається стандартним ім'ям Pi; ClrScr – ідентифікатор процедури без параметрів, яка очищує екран ЕОМ і переміщує курсор у верхній лівий кут екрана; Sin – ім'я стандартної функції $y = \sin x$, яка повертає в основну програму значення синуса відповідного аргументу.

Імена деяким об'єктам (змінним, функціям і т. п.) може призначати і сам користувач (програміст). При цьому необхідно дотримуватись правил, які випливають із визначення ідентифікатора.

Приклади імен, складених без помилок: X; Y15; Gamma; A_1_3.

Приклади помилок в іменах:

5Dx – ім'я починається з цифри;
K.2 – ім'я містить неприпустимий символ (крапку);
Al Fa – ім'я містить неприпустимий символ (пропуск);
A+B – ім'я містить неприпустимий символ (знак "+").

Число записується в *Pascal*-програмі в десятковій системі числення і може бути цілим або дійсним, додатним або від'ємним, при цьому знак "+" можна не писати.

Приклади правильно записаних цілих чисел: –250; 134; 0; +7; 07.

Дійсні числа можна записувати в двох формах – *звичайній* і *показниковій* (*напівлогарифмічній*). При запису в звичайній формі число містить десяткову крапку (замість звичної десяткової коми при математичному запису числа). У показниковій формі зручно записувати дуже великі або дуже маленькі цифри. **Наприклад,** число $6{,}735 \cdot 10^{-12}$ мовою *Pascal* має вигляд: 6.735E–12. Цифри, що знаходяться перед буквою E, називаються *мантисою* числа, а цифри, що знаходяться після E, – *порядком*.

Приклади правильно записаних дійсних чисел:

звичайна форма – 7.545; –0.002123; 0.0; –24.32; +2.1;

показникова форма – –0.123E5; 10E7; –23E-05; 12.445E-2; 0.123E; –12E00.

Приклади неправильно записаних дійсних чисел:

звичайна форма –	15.	– цифра закінчується крапкою;
	.001	– цифра починається з крапки;
	12.13.25.03	– цифра містить декілька крапок;
показникова форма –	–5.E-12	– мантиса закінчується крапкою;
	–E12	– відсутня мантиса.

Рядок – це послідовність символів, яка міститься в одинарних лапках (апострофах). У послідовності між двома апострофами можуть знаходитися будь-які символи, включаючи пропуск і сам апостроф. Якщо в рядок необхідно включити знак апострофа, то його записують двічі. Рядки використовуються для роботи з текстами (наприклад, для виведення тексту на екран ЕОМ).

Приклади правильно записаних рядків:

'A' ; '234 AZaz' ;

' Лабораторна робота "Механічні властивості стали" '.

Приклади неправильно записаних рядків:

Color'	– немає початкового апострофа;
студент	– відсутні апострофи;
'123.3'46	– апостроф повинен стояти наприкінці рядка;
' + . , – ' '	– не вистачає ще одного апострофа.

Вираз – це конструкція, яка визначає дії, що повинні бути виконані для обчислення деякої величини. Вирази складаються з операндів (констант, змінних, функцій), знаків операцій і круглих дужок. Для обчислення числового значення виразу використовуються *арифметичні операції*. Правила запису знаків арифметичних операцій мовою *Pascal* ілюструє табл. 2.2.

Арифметичні операції є двомісними (бінарними), тобто вони виконуються із залученням двох операндів. При цьому знак "–" може бути знаком одномісної (унарної) операції, наприклад, –2.350.

Порядок виконання операцій в арифметичному виразі визначається їх *пріоритетом*: операції додавання і віднімання виконуються після операцій

множення і ділення; операції з однаковим пріоритетом виконуються в порядку написання у програмі зліва направо; якщо вираз міститься в дужках, то він виконується в першу чергу.

Таблиця 2.2. **Знаки арифметичних операцій мови *Pascal***

Знак	Математичний запис	Запис мовою *Pascal*
+	$a+b$	a+b
–	$a-b$	a–b
*	$a \cdot b$	a*b
/	$a:b$	a/b

Як **приклад** розглянемо запис нескладного арифметичного виразу мовою *Pascal*:

$$\frac{x(x^2-x+1)}{2x+(3{,}5-x)}-2{,}32\cdot 10^{-7} \Rightarrow \text{X*(X*X–X+1)/(2*X+(3.5–X))–2.32E–7}$$

Крім наведених у табл. 2.2 стандартних арифметичних операцій над цілими і дійсними операндами, існують операції, які застосовуються виключно тільки для цілих операндів:

Div – цілочислове ділення з відкиданням, тобто знаходження цілої частини числа, як результату ділення двох цілих операндів. **Наприклад,** записаний мовою *Pascal* вираз 5 div 2 буде обчислений, як "дві цілих", а значення виразу 5 div 6 дорівнює нулю;

Mod – операція обчислення остатку від цілочислового ділення двох цілих операндів. **Наприклад,** значення виразу 5 mod 2 дорівнює одиниці, а значення виразу 16 div 6 дорівнює 4.

Операндами у виразі, крім констант та змінних, можуть бути *стандартні функції* з аргументом різного типу. Зазначені функції наведені в табл. 2.3.

Таблиця 2.3. **Стандартні математичні функції мови *Pascal***

Математичне позначення	Запис мовою *Pascal*	Примітка
$\lvert x\rvert$	Abs(X)	Обчислює модуль аргументу X. Тип аргументу може бути цілим або дійсним, а тип результату співпадає з типом аргументу
x^2	Sqr(X)	Обчислює значення X в другому степені. Тип X може бути цілим або дійсним, а тип результату співпадає з типом аргументу
$\sqrt{x}$	Sqrt(X)	Обчислює квадратний корінь із додатного аргументу X, який може бути дійсним або цілим. Результатом завжди є дійсне число

Продовж. табл. 2.3

Математичне позначення	Запис мовою *Pascal*	Примітка
$\sin x$	Sin(X)	Обчислюються тригонометричні функції аргументу X. Значення аргументу задається в радіанах і може бути цілим або дійсним, а результат завжди дійсний
$\cos x$	Cos(X)	
arctgx	Arctan(X)	
e^x	Exp(X)	Піднесення числа e = 2,71828... до степеня X. Степінь може бути цілий або дійсний, а результат завжди дійсний
$\ln x$	Ln(X)	Обчислюється натуральний логарифм ненульового додатного аргументу X. Тип X може бути цілий або дійсний, а результат має дійсний тип

Як видно з табл. 2.3, набір стандартних функцій мови *Pascal* досить обмежений, тому ті функції, що не ввійшли в таблицю, треба обчислювати за допомогою стандартних. **Наприклад:** $\operatorname{tg} x = \frac{\sin x}{\cos x} \Rightarrow$ Sin(x)/Cos(x);

$\csc x = \frac{1}{\sin x} \Rightarrow$ 1/Sin(x); $\operatorname{sh} x = \frac{e^x - e^{-x}}{2} \Rightarrow$ 0.5*(Exp(x)–Exp(–x));

$x^y = e^{\ln x^y} = e^{y \cdot \ln x} \Rightarrow$ Exp(y*Ln(x)); $\log_a x = \frac{\ln x}{\ln a} \Rightarrow$ Ln(x)/Ln(a), і т. п.

Крім наведених у табл. 2.3 стандартних функцій, існують функції тільки дійсного аргументу, які перетворюють його в число цілого типу:

Trunc(X) – обчислює цілу частину дійсного аргументу X. **Наприклад:** значення виразу Trunc(2.123) дорівнює 2, а значення виразу Trunc(3.9999) дорівнює 3;

Round(X) – визначає заокруглене дійсне значення X за формулою *Round*(*x*) = *Trunc*(*x* + 0,5) при $x \geq 0$, а при $x < 0$ *Round*(*x*) = *Trunc*(*x* – 0,5). **Наприклад:** значення виразу Round(2.4999) дорівнює 2, а значення виразу Round(3.5) дорівнює 4.

2.2. Прості стандартні типи даних

З попереднього викладення вже стало зрозуміло, що кожен елемент даних у програмі є або константою, або змінною.

Константа – стала величина, значення якої не можна змінити в процесі виконання програми. Константи поділяються на *арифметичні*, *рядкові* та *логічні*. Вище ми розглянули правила запису цілих і дійсних констант (чисел), а також правила запису рядкових констант (рядків).

Логічні константи визначаються іменами True і False. Перше ім'я представляє логічне поняття "істина", а друге – "хибність".

Константи в програмі можуть бути представлені або безпосередньо своїм значенням (числом, рядком), або ім'ям (стандартним: Pi, False, True, MaxInt і т. п.; або заданим користувачем).

Змінна – це величина, значення якої може змінюватися в процесі виконання програми. Змінні в програмі можливо представляти тільки іменами.

У програмі можна використовувати дані різних типів. На рис. 2.1 показана структура даних мови *Pascal*.

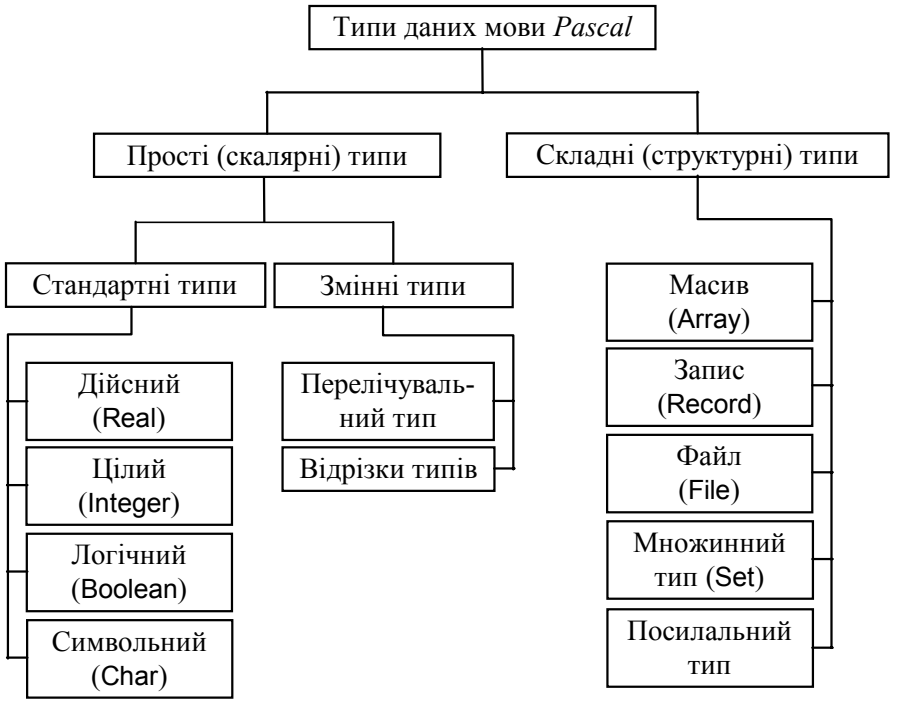


Рис. 2.1. Типи даних мови *Pascal*

У даному підрозділі зупинимося тільки на простих (скалярних) стандартних типах даних. Ці типи називаються стандартними, тому що вони визначаються стандартними іменами: цілий тип – Integer; дійсний тип – Real; логічний тип – Boolean; символьний тип – Char.

Цілий тип (Integer). Величина цілого типу є елементом підмножини цілих чисел. У мові *Pascal* існує стандартна константа MaxInt, значення якої є найбільше ціле значення, що допускається у даній ЕОМ. Для персональних комп'ютерів значення MaxInt дорівнює 32767, а значення цілих чисел знаходяться у діапазоні –32768…32767. Кожне значення типу Integer займає 2 байти[1] пам'яті ЕОМ.

[1] Байт – мінімальний дискрет даних, який можна записати у пам'ять комп'ютера. Розмір типів наведено для мови *Turbo Pascal*.

Дійсний тип (Real). Величина дійсного типу є елементом підмножини дійсних чисел. У підмножину входить число нуль, а також додатні та від'ємні числа, абсолютне значення яких знаходиться у діапазоні $10^{-36}…10^{38}$. Кожне дійсне число займає 6 байт і зберігається в пам'яті комп'ютера з точністю до 10 значущих цифр.

Логічний тип (Boolean). Як вже відзначалося на початку підрозділу, існує два значення логічного типу: True і False. Логічна змінна, що приймає одне з цих двох значень, займає в пам'яті ЕОМ 1 байт. При цьому встановлено, що *True* > *False*.

У мові *Pascal* існують стандартні функції логічного типу:

Odd(X) – результат виконання цієї функції дорівнює True, якщо цілий аргумент X – непарне число, і False – у противному випадку;

Eoln(F) – результат дорівнює True, якщо при читанні текстового файлу[2] F досягнуто кінець поточного рядка, і False – в іншому випадку;

Eof(F) – результат дорівнює True, якщо при читанні файлу F досягнуто його кінця, і False – в іншому випадку.

Значення True і False звичайно можна отримати в результаті виконання операцій порівняння (>, <, =, >=, <=). **Наприклад,** значення логічного виразу 12 < 3 є False, а 0.123 > –23.1 – True.

Необхідно розуміти, що до операндів дійсного типу не варто застосовувати операцію "=", оскільки така рівність швидше за все ніколи не виконається через неточність представлення дійсних чисел у пам'яті ЕОМ і неминучих помилок заокруглення при обчислюванні виразів дійсного типу. Таким чином, відношення A = B варто замінити відношенням Abs(A–B) < Eps, де ідентифікатор Eps визначає деяку малу величину, що характеризує ступінь близькості величин A і B.

Крім операцій порівняння, в мові *Pascal* існують три логічні операції, які можна застосовувати тільки з операндами логічного типу. Це одномісна (унарна) операція Not, а також бінарні операції And і Or. Математичні позначення і назви цих операцій ілюструє табл. 2.4, а їхнє виконання – табл. 2.5.

Таблиця 2.4. **Логічні операції мови *Pascal***

Математичне позначення	Назва	Запис на мові *Pascal*
¬	Ні (заперечення)	Not
∧	І (логічне множення або кон'юнкція)	And
∨	Або (логічне додавання або диз'юнкція)	Or

[2] Поняття файлу та текстового файлу подано в підрозд. 4.2.

Таблиця 2.5. **Значення логічних операцій мови *Pascal***

Значення операнда		Результат			
A	**B**	**Not A**	**Not B**	**A And B**	**A Or B**
False	False	True	True	False	False
False	True	True	False	False	True
True	False	False	True	False	True
True	True	False	False	True	True

Як **приклад** складемо логічний вираз, який визначає приналежність деякої точки з координатами (x, y) квадратній області $-\pi \le x, y \le \pi$. Він має вигляд:

```
(X >= –Pi) and (X <= Pi)  And  (Y >= –Pi) and (Y <= Pi)
```

Логічні операції мають більший пріоритет, ніж операції порівняння, і тому операції порівняння в даному прикладі відділяються круглими дужками. Пріоритет виконання логічних операцій такий: спочатку виконується операція заперечення Not, потім And і в останню чергу – Or.

Символьний (*літерний*) *тип* (Char). Величина типу Char є елементом підмножини символів коду *ASCII* [3]. Кожен символ займає в пам'яті ЕОМ 1 байт.

Кожне значення символьного типу має свій заздалегідь зарезервований порядковий номер. До значення типу Char, а також до його порядкового номера можуть бути застосовні стандартні функції:

Ord(X) – визначає цілий порядковий номер аргументу X в упорядкованій сукупності значень цього аргументу. Тип X може бути будь-яким скалярним типом крім дійсного. **Наприклад,** порядковий номер символу "#", обчислений як Ord(**'#'**), дорівнює 35; Ord(**'A'**) дорівнює 65; Ord(**'a'**) дорівнює 97 і т. п. Два значення логічного типу також упорядковані: Ord(True) дорівнює 1, а Ord(False) дорівнює 0;

Chr(X) – функція, обернена до Ord(X). Вона визначає символ (якщо він існує), порядковим номером якого є аргумент X. Таким чином, результат виконання функції має символьний тип. **Наприклад,** 5-м порядковим номером визначається символ **'♣'**; 42-м -**'*'**; 65-м – **'A'** і т. п.;

Succ(X) – визначає значення (якщо воно існує), що слідує за аргументом X, тип якого може бути будь-яким простим типом, крім дійсного. Тип результату співпадає з типом аргументу. **Наприклад,** вираз Succ(False) визначає логічне значення True; Succ(**'Ю'**) визначає символьну константу **'Я'**; Succ(19) визначає ціле число 20;

[3] *ASCII* – слово, утворене з початкових букв фрази *American Standard Code for Information Interchange* (Американський стандартний код для інформаційних обмінів). Це угода про стандарт представлення символів у пам'яті ЕОМ.

Pred(X) – визначає значення (якщо воно існує), що знаходиться перед аргументом X, тип якого може бути будь-яким простим типом крім дійсного. Тип результату співпадає з типом аргументу. **Наприклад,** вираз Pred(True) визначає логічне значення False; Pred(***'S'***) визначає символьну константу ***'R'***; Pred(–5) визначає ціле число –6.

Розглянуті прості типи даних мають дві характерні властивості: *неподільність та упорядкованість своїх значень*. Справді, кожна символьна константа або кожне ціле число представляє собою об'єкт, який не розпадається на окремі компоненти (символи або прості цифри). З іншого боку, сукупність цих об'єктів упорядкована.

2.3. Структура програми мовою Pascal

Програма, яка написана мовою *Pascal*, складається із *заголовка* і власне програми, яка називається *блоком*. Блок складається з декларативної та виконавчої частин. У декларативній частині описуються імена констант, типи, мітки, змінні, підпрограми. Усі декларації можуть бути розташовані у програмі в довільному порядку. Виконавча частина складається зі *складного* (*складеного*) оператора Begin-End і містить оператори програми. Нижче наводиться загальна структура *Pascal*-програми.

Треба відзначити, що в програмі обов'язково повинен бути присутній тільки останній розділ, всі інші розділи, а також заголовок програми, у випадку непотрібності можуть бути відсутніми.

	Program <*ім'я*>;	**ЗАГОЛОВОК**	
Декларативна частина програми	**Label** <*мітка1*>, <*мітка2*>, ..., <*міткаN*>;	БЛОК	Розділ міток
	Const <*ім'я_константи1*>=<*константа1*>; <*ім'я_константи2*>=<*константа2*>; ...; <*ім'я_константиM*>=<*константаM*>;		Розділ констант
	Type <*ім'я_типу1*>=<*тип1*>; <*ім'я_типу2*>=<*тип2*>; ...; <*ім'я_типуK*>=<*типK*>;		Розділ типів
	Var <*ім'я_змінної1*>, ..., <*ім'я_змінноїD*> : <*тип1*>; ...; <*ім'я_змінної1*>, ..., <*ім'я_змінноїS*> : <*типL*>;		Розділ змінних
	Procedure <*заголовок_процедури*>; <*блок_процедури*>;		Розділ процедур
	Function <*заголовок_функції*>; <*блок_функції*>;		Розділ функцій
Виконавча частина програми	**Begin** <*оператор1*>; <*оператор2*>; ...; <*операторG*> **End.**		Розділ операторів

Елементи списків (мітки, змінні) відокремлюються один від одного спеціальним символом "**,**", а розділи та оператори – спеціальним символом "**;**". Наприкінці програми ставиться крапка.

У будь-яке місце програми може бути поміщений коментар. *Коментар* – це текст, який починається лівою фігурною дужкою і закінчується найближчою правою фігурною дужкою {<*коментар*>}. При виконанні програми коментар ігнорується (не виконується). Коментарі полегшують читання програми, пояснюють виконувані дії. Присутність коментарів у програмі свідчить про гарний стиль програмування.

Підкреслимо, що *всі змінні, константи, мітки і підпрограми користувача, які використовуються в Pascal-програмі, обов'язково повинні бути описані у відповідних розділах її декларативної частини.* Правила таких описів будуть представленні у процесі вивчення матеріалу.

2.4. Оператор присвоєння

Оператор присвоєння є основним оператором будь-якої мови програмування. За його допомогою виконуються операція обчислення значення деякого виразу і запис його в пам'ять комп'ютера. Синтаксис[4] оператора присвоєння має вигляд:

<*Ідентифікатор*>:=<*Вираз*>;

де *Ідентифікатор* – ім'я змінної або функції, а знак ":=" – знак операції присвоєння.

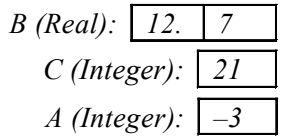
Наприклад: X1:=(–b+Sqrt(D))/(2*a); L:=R*Alfa; Symbol:=**'S'**[5].

Не зважаючи на те, що операція присвоєння дозволяє обчислити значення деякого виразу, її не слід плутати з математичною операцією рівності ("="). Так, з математичної точки зору, вираз $\beta = \beta + \pi$ беззмістовний, тому що $\pi \neq 0$, а з погляду програмування, запис Betta:=Betta+Pi має чітко визначений зміст: поточне значення змінної Betta збільшується на Pi радіан. Таким чином, *оператор присвоєння дозволяє замінити поточне значення змінної, яка знаходиться ліворуч від знака* ":=" *на нове значення, що визначається виразом праворуч.*

Ідентифікатор (змінна або функція) в операторі присвоєння може бути будь-якого типу (крім файлового), але обов'язково ідентичним з типом виразу. При цьому припускається присвоєння змінній (функції) дійсного типу значення виразу цілого типу. Розглянемо наступний **приклад**. Нехай в оперативній пам'яті ЕОМ виділені комірки (місця) для збереження двох цілих

[4] Під поняттям "синтаксис" тут і надалі будемо розуміти правила, які визначають формат операторів або команд мови *Pascal*.

[5] В якості рекомендації відмітимо, що змінним необхідно давати імена, пов'язані зі змістом розв'язуваної задачі. Це полегшує розуміння програми іншими людьми.



і одного дійсного чисел. У результаті операції A:=C в комірці *A* поточне значення (–3) заміниться новим значенням (21), а комірки *B* і *C* збережуть свої значення; у результаті операції B:=C в комірці *B* поточне значення (12,7) заміниться новим значенням (21,0), а при виконанні операції C:=B виникне помилка, через невідповідність типів – змінній цілого типу не можна присвоїти значення дійсного типу.

2.5. Введення та виведення даних. Оператори введення та виведення

Введення та виведення даних пов'язані з обміном інформацією між оперативною пам'яттю ЕОМ і зовнішніми носіями інформації. Введення в програму вхідних даних і виведення результатів здійснюється за допомогою стандартних файлів: Input (для введення) і Output (для виведення). Пристрої для введення та виведення можуть бути найрізноманітнішими. Поки будемо вважати, що введення даних в оперативну пам'ять комп'ютера може здійснюватися за допомогою клавіатури, а виведення відбувається на екран ЕОМ.

Введення даних може здійснюватися декількома способами.

1. У розділі опису констант перелічуються константи та задаються їхні значення.

```
…
Const A=2.5; X=–7.35; K=–0.53; T=225;
…
```

Завдяки такому опису в оперативній пам'яті ЕОМ виділяється місце для збереження дійсних констант A, X, K і цілої константи T, після чого, в пам'ять записуються відповідні значення вказаних констант. Змінити значення константи в програмі не можна і в цьому сенсі таке введення даних нераціональне.

2. У розділі опису змінних записуються відповідні імена із зазначенням типу (при цьому виділяється місце для збереження значень цих змінних в оперативній пам'яті), а в розділі операторів іменам присвоюються значення, що повинні відповідати описаному типу. Зрозуміло, якщо виникне необхідність змінити значення вхідних даних, то кожен раз доведеться робити виправлення в самій програмі – це теж не є раціональним.

```
…
Var   A,X,K:Real;
      T:Integer;
Begin
  A:=2.5; X:=–7.35;
  K:=–0.53; T:=225;
…
```

3. Для раціонального введення даних в оперативну пам'ять ЕОМ використовують *оператори введення* Read і ReadLn.

Синтаксис оператора Read має вигляд:

Read(<*змінна_1*>, <*змінна_2*>, …, <*змінна_N*>);

де Read (читати) – службове слово.

При виконанні оператора Read ЕОМ "чекає" на послідовне введення значень елементів списку введення (значень усіх змінних, які перелічені в дужках оператора). При цьому *тип поточного значення повинен відповідати типу змінної, яка має бути заздалегідь описана у розділі опису змінних* Var. Послідовне введення може здійснюватися в один рядок (через натискання клавіші "Пропуск") або в стовпець (через натискання клавіші "Введення (*Enter*)").

Розглянемо два однакових за дією фрагменти *Pascal*-програм. Зрозуміло, що виконання операторів, які знаходяться після Read, буде здійснюватися лише після того, як користувач введе з клавіатури чотири числа: три дійсних і одне ціле. Таким чином, в результаті описів у розділі Var, виділяються комірки в оперативній пам'яті ЕОМ для зберігання змінних дійсного типу A, X, K і змінної цілого типу T, а після виконання оператора Read ці комірки будуть заповнені відповідними (введеними користувачем) значеннями.

```
…
Var A,X,K:Real;
    T:Integer;
Begin
 Read(A,X,K,T);
…
```

```
…
Var A,X,K:Real;
    T:Integer;
Begin
 Read(A,X); Read(K,T);
…
```

Оператор введення ReadLn виконується так само, як і оператор Read, але потім здійснюється перехід до початку наступного рядка файлу Input. Синтаксис цього оператора аналогічний синтаксису оператора Read:

ReadLn(<*список_введення*>);

Для виведення даних у мові *Pascal* використовуються *оператори виведення* Write і WriteLn. Синтаксис оператора Write має вигляд:

Write(<*вираз_1*>, <*вираз_2*>, …, <*вираз_N*>);

де Write (писати) – службове слово.

Цей оператор виконує виведення даних, які перераховані в його дужках, у стандартний файл Output. При цьому елементами списку виведення можуть бути змінні, вирази та рядкові константи. **Наприклад,** після виконання операторів

```
X:=Pi; Y:=Sin(X);
Write('X=', X, ', a SinX=', Y);
```

або

```
X:=Pi;
Write('X=', X, ', a SinX=', Sin(X));
```

на екрані ЕОМ з'явиться напис:

X=3.1415926500E+00,□a□SinX=3.5870471038E–09[6].

[6] Символом □ тут і надалі будемо позначати пропуск.

Треба відмітити, що *значення* X *і* Y, *які виводяться за допомогою оператора виведення, повинні бути заздалегідь визначені в програмі,* в противному випадку вони автоматично будуть визначені як нуль.

Як видно, значення X і Y виводяться на екран у *стандартній* (*показниковій*) *формі*, яка не завжди зручна для читання та аналізу. Тому в мові *Pascal* передбачені засоби управління виведенням: *ширина поля виведення* і *точність виведення дійсного числа.* **Наприклад,** після виконання оператора

```
                точність
±□□□□□□.□□□□□□
ширина поля
```

```
Write('X=', X:10:4, ', a SinX=',Y:10:4);
```
або
```
Write('X=',X:10:4,', a SinX=',Sin(X):10:4);
```

на екрані ЕОМ з'явиться напис: X=□□□□3.1416,□a□SinX=□□□□0.0000.

Таким чином, на дійсні числа X і Y відведено десять позицій (ширина поля виведення), серед яких чотири позиції (точність) – на дробову частину. Якщо ширина поля виявиться більшою, ніж необхідно (як у нашому випадку), то значення виведеться з пропусками, а якщо ширина поля виявиться недостатньою, то в рядку виведення автоматично додадуться необхідні позиції.

Оператор WriteLn виконує майже ті ж дії, що й оператор Write і має аналогічний синтаксис:

WriteLn(<*список_виведення*>);

Відмінність між цими операторами полягає в тому, що оператор WriteLn після виведення останнього елемента зі списку виведення робить перехід до початку нового рядка файлу Output. Як **приклад** розглянемо два фрагменти програми. У результаті виконання першого фрагмента, на екрані ЕОМ з'явиться напис:

```
…
Const R=5.5;
Var S,L:Real;
Begin
 S:=Pi*Sqr(R);L:=2*Pi*R;
 Write('Площа-', S:8:3);
 Write('Довжина-',L:8:3);
…
```

```
…
Const R=5.5;
Var S,L:Real;
Begin
 S:=Pi*Sqr(R);L:=2*Pi*R;
 WriteLn('Площа-', S:8:3);
 WriteLn('Довжина-',L:8:3);
…
```

Площа□□95.033□Довжина□□34.558

Результат виконання другого фрагмента програми має вигляд:

Площа□□95.033
□Довжина□□34.558

2.6. Найпростіша лінійна програма

Лінійна програма зазвичай складається для обчислення значення деякого виразу, або значень декількох виразів. Для цього необхідно ввести в пам'ять ЕОМ вхідні дані, потім обчислити значення виразу (або декількох виразів), а результати обчислень вивести на екран комп'ютера. Таким чином, найпростіша лінійна програма складається з операторів введення, присвоєння і виведення.

Як **приклад** складемо програму обчислення дійсних коренів квадратного рівняння вигляду $ax^2 + bx + c = 0$ у випадку додатного значення дискримінанта цього рівняння.

У заголовку вказано ім'я програми – Example1 (Приклад 1). У другому рядку описаний стандартний бібліотечний модуль Crt. Це зроблено для того, щоб одержати доступ до безпараметричної процедури цього модуля – ClrScr (*Clear* – чистий, *Screen* – екран), яка дозволяє очистити екран ЕОМ. У розділі описів, крім цього, описані з указівкою типу всі змінні, які приймають участь в обчислювальному процесі.

```
Program Example1;
Uses Crt;
Var a,b,c,D,X1,X2:Real;
BEGIN
 ClrScr;
 WriteLn('Введіть вхідні дані: a, b і c');
 ReadLn(a,b,c);
   D:=Sqr(b)–4*a*c;
   X1:=(–b+Sqrt(D))/(2*a);
   X2:=(–b–Sqrt(D))/(2*a);
 ClrScr;
 WriteLn('Результати:');
 WriteLn('X1=', X1:10:4,' X2=', X2:10:4);
 ReadLn
END.
```

У розділі операторів записані оператори, послідовне виконання яких призводить до розв'язку задачі. Спочатку очищується екран, а потім на ньому з'являється рядкова константа ***'Введіть вхідні дані: a, b і c'***. Далі виконання програми припиняється до того часу, поки користувач не введе з клавіатури конкретні числові значення змінних a, b і c, необхідні для обчислення значень дискримінанта D та коренів X1, X2. Після чергового очищення екрана, на ньому з'явиться напис ***'Результати:'***, а на наступному рядку – значення коренів квадратного рівняння з точністю до чотирьох цифр після коми. Останній оператор у даній програмі – оператор ReadLn без списку введення. Використання цього оператора дозволяє припинити виконання програми до того часу, поки не буде натиснута клавіша "*Enter*", а після її натискання відбудеться повернення в редактор *Pascal*-програм.

Зауважимо: якщо значення D виявиться від'ємним, то це призведе до помилки виконання програми.

2.7. Функції користувача

У підрозд. 2.1 відзначалося, що для обчислення математичних виразів у мові *Pascal* передбачений обмежений набір стандартних функцій (див. табл. 2.3), а якщо функція не є стандартною, то її доводиться виражати через комбінацію стандартних функцій. **Наприклад,** для запису мовою *Pascal* математичного виразу $y = \text{th}(\text{tg}\, x)$ гіперболічний і тригонометричний тангенси необхідно перетворити до такого вигляду:

$$\text{th}\, x = \frac{\text{sh}\, x}{\text{ch}\, x} = \frac{\frac{e^x - e^{-x}}{2}}{\frac{e^x + e^{-x}}{2}} = \frac{e^x - e^{-x}}{e^x + e^{-x}};\quad \text{tg}\, x = \frac{\sin x}{\cos x} \Rightarrow \text{th}(\text{tg}\, x) = \frac{e^{\frac{\sin x}{\cos x}} - e^{-\frac{\sin x}{\cos x}}}{e^{\frac{\sin x}{\cos x}} + e^{-\frac{\sin x}{\cos x}}},$$

а відповідний запис мовою *Pascal* має вигляд:

```
Y:=( Exp(sin(x)/cos(x))–Exp(–sin(x)/cos(x)) )/( Exp(sin(x)/cos(x))+
                +Exp(–sin(x)/cos (x)) );
```

Як видно, надмірна деталізація виразу для Y призводить до втрати простоти та зрозумілості, ускладнює читання виразу. Крім того, якщо зазначену функцію доведеться обчислювати кілька разів у різних місцях програми, то це призведе до збільшення тексту програми, погіршить її структуру. Одним із засобів, який дозволяє уникнути вказаних недоліків і спростити написання *Pascal*-програми, є *підпрограма-функція*.

Підпрограма-функція вводиться за допомогою свого опису у відповідному місці *Pascal*-програми (див. підрозд. 2.3). За своєю структурою функція аналогічна основній програмі та теж складається із *заголовка* і *блоку*. Блок підпрограми-функції та основної програми аналогічні, а заголовок функції має вигляд:

Function <*Ім'я*> (<*Список_параметрів*>):<*Тип*>;

де Function (функція) – службове слово;
Ім'я – ім'я (ідентифікатор) функції;
Список_параметрів – перелік *формальних* параметрів (вхідних даних або аргументів) функції з указівкою їхнього типу;
Тип – тип результату, тобто тип значення, яке здобуває функція.

Необхідно відзначити, що опис функції у відповідному розділі описів *Pascal*-програми ніякої дії не спричиняє. Щоб виконати функцію, необхідно у потрібному місці програми зробити *звернення* до неї. Звернення до функції робиться аналогічно зверненню до стандартних функцій (див. табл. 2.3):

у необхідному місці записується ім'я функції, а в дужках вказуються *фактичні* параметри.

Як **приклад** наведена *Pascal*-програма Example2 для обчислення значення функції $y = \text{th}(\text{tg}\, x)$ в деякій точці a.

У розділі описів програми Example2 (крім опису стандартного модуля Crt, змінних A і Y, які беруть участь в обчисленнях) описані дві підпрограми-функції з іменами: tg і th. У заголовку цих підпрограм як формальний параметр використовується ім'я x, що визначає дійсний аргумент функцій. Самі значення функцій також визначені (описані) як дійсні. У блоках цих підпрограм, які складаються з одного розділу – розділу операторів, обчислюються відповідні вирази, значення яких присвоюються іменам функцій.

В основній програмі (після очищення екрана і введення змінної A) здійснюється звернення (виклик) до створених нами функцій. У тілі функції tg відбувається заміна формального параметра x на фактичний A, виконується обчислення виразу Sin(A)/ /Cos(A) і в основну програму *повертається* результат tg.

```
Program Example2;
Uses Crt;
Var Y,A:Real;
Function tg(x:Real):Real;
Begin
  tg:=Sin(x)/Cos(x)
End;
Function th(x:Real):Real;
Begin
  th:=(Exp(x)–Exp(–x))/(Exp(x)+Exp(–x))
End;
BEGIN
  ClrScr;
  WriteLn('Введіть значення A');
  ReadLn(A);
  Y:=th(tg(A));
  WriteLn('При A=',A:10:4,', значення Y=',Y:10:4);
  Readln
END.
```

Отриманий результат аналогічно використовується як фактичний параметр у підпрограмі-функції th: формальний параметр x замінюється на фактичний tg, виконується обчислення виразу (Exp(tg)–Exp(–tg))/(Exp(tg)+Exp(–tg)) і в основну програму повертається результат th. Нарешті цей результат присвоюється імені змінній Y та виводиться на екран ЕОМ.

По закінченні даного підрозділу окремо підкреслимо основні положення, необхідні для грамотного та ефективного використання апарата підпрограм-функцій:

1) число і тип формальних і фактичних параметрів повинні строго співпадати;

2) результат виконання функції – одне скалярне значення, яке передається в основну програму як значення імені цієї функції.

Більш докладний опис апарата підпрограм міститься в підрозд. 3.3.

3. НЕЛІНІЙНІ ОБЧИСЛЮВАЛЬНІ ПРОЦЕСИ І БАЗОВІ УПРАВЛЯЮЧІ КОНСТРУКЦІЇ ДЛЯ ЇХ РЕАЛІЗАЦІЇ НА ЕОМ

3.1. Розгалужені обчислювальні процеси. Оператори мови Pascal для їх програмної реалізації

Обчислювальний процес називають *розгалуженим*, якщо в залежності від виконання деяких умов він реалізується по одному з декількох, заздалегідь передбачених (можливих) напрямків. Кожен такий окремий напрямок називається *віткою обчислень*.

Для програмної реалізації розгалужених обчислювальних процесів у мові *Pascal* використовуються три оператори:

1) оператор умовного переходу;
2) оператор варіанта;
3) оператор безумовного переходу.

3.1.1. Оператор умовного переходу

Оператор умовного переходу – це такий оператор передачі управління в програмі, який дозволяє перейти з одного місця програми в інше, тобто передати управління і у такий спосіб змінити послідовний порядок виконання операторів. Передача управління відбувається в результаті виконання деякої умови (тобто в залежності від значення логічного виразу, який описує цю умову). Якщо умова виконується (значення логічного виразу дорівнює True), то здійснюється перехід до першої вітки обчислень (рис. 3.1). У противному випадку, якщо умова не виконується (значення логічного виразу дорівнює False), виконується друга вітка обчислень.

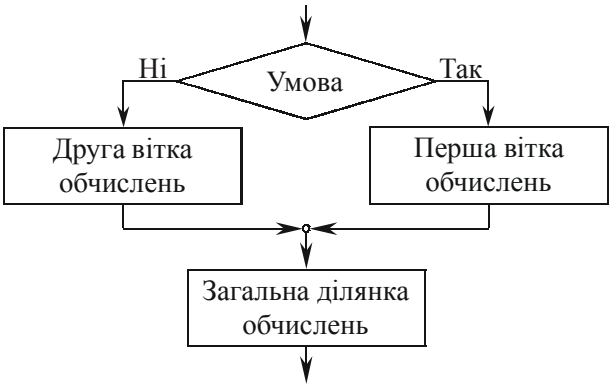


Рис. 3.1. Розгалужений обчислювальний процес, який містить дві вітки обчислень

У мові *Pascal* загальний синтаксис оператора умовного переходу має вигляд:

If <*Логічний_вираз*> **Then** <*Оператор_1*>
Else <*Оператор_2*>;

де If (якщо), Then (то), Else (інакше) – службові слова;
Оператор_1 і *Оператор_2* – будь-які[7] оператори мови *Pascal*.

Як **приклад** використання умовного оператора, розглянемо фрагмент *Pascal*-програми обчислення значення функції $y(x) = \begin{cases} \ln(x-5), & x > 5; \\ e^x, & x \le 5. \end{cases}$ в деякій точці x.

```
…
ReadLn(x);
If x>5 Then y:=Ln(x–5)
       Else y:=Exp(x);
<Оператор_N>;
…
```

Якщо вводиться значення змінної $x \in (5; +\infty)$, то значення логічного виразу x>5 дорівнює True і, отже, виконується операція y:=Ln(x–5) блоку Then (*Оператор_1*), а операція y:=Exp(x) блоку Else (*Оператор_2*) ігнорується. Якщо $x \in (-\infty; 5]$, то значення логічного виразу x>5 дорівнює False і виконується *Оператор_2*, а *Оператор_1* ігнорується. Далі у будь-якому випадку виконується оператор, який знаходиться в програмі за оператором умовного переходу (*Оператор_N*).

Логічний вираз в операторі умовного переходу може бути і більш складним. Нагадаємо, що для його формування можна використовувати логічні операції Not, And і Or (див. підрозд. 2.2).

[7] Під оператором поки будемо розуміти *простий* (тобто один єдиний) оператор мови *Pascal*. Так, наприклад, один оператор присвоєння, один оператор введення, один оператор виведення і т. п. є простими операторами.

Розглянута форма оператора умовного переходу називається *повною*, однак широко використовується й інша – *скорочена* форма, яка не містить блока Else. Схема виконання оператора If у скороченій формі для реалізації обчислювального процесу, що містить одну вітку обчислень, ілюструє рисунок 3.2, а його синтаксис має вигляд:

If <*Логічний_вираз*> **Then** <*Оператор*>;

Якщо умова виконується (значення логічного виразу дорівнює True), то здійснюється перехід до першої вітки обчислень (виконується *Оператор*). У противному випадку, якщо умова не виконується (значення логічного виразу дорівнює False), відразу здійснюється перехід до загальної ділянки обчислень (до оператора, який слідує за оператором If).

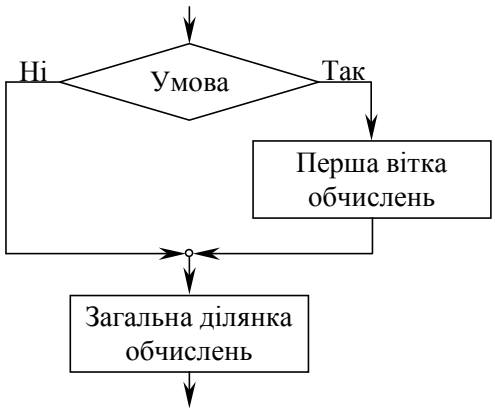


Рис. 3.2. Розгалужений обчислювальний процес, що містить одну вітку обчислень

Як **приклад** розглянемо фрагменти програм для розв'язання задачі визначення максимального числа з двох заданих:

```
…
ReadLn(A,B);
If A>B Then Max:=A
       Else Max:=B;
<Оператор_N>;
…
```

```
…
ReadLn(A,B);
Max:=A;
If B>A Then Max:=B;
<Оператор_N>;
…
```

Даний приклад ілюструє програмну реалізацію двох еквівалентних обчислювальних алгоритмів[8]. У першому випадку використовується повна форма оператора умовного переходу, а в другому – скорочена.

В другому випадку після введення двох чисел A і B максимальним з них вважається A. Якщо виявляється, що B > A, то максимальним числом стає B, у противному випадку (якщо значення логічного виразу B > A дорівнює False) відразу здійснюється перехід до оператора, якій знаходиться в програмі за оператором If (*Оператор_N*) і максимальним числом залишається A.

Напевно, у читача вже виникло запитання: "Якщо ми припустили, що оператори блоків Then і Else прості, як же за допомогою оператора умовного переходу програмно реалізувати обчислювальний алгоритм розгалу-

[8] Два обчислювальних алгоритми будемо називати еквівалентними, якщо однаковим вхідним даним вони зіставляють один і той же результат, але при цьому спосіб його одержання різний.

ження у випадку, коли яка-небудь вітка обчислень містить не одну, а кілька дій?" Спеціально для таких випадків у мові *Pascal* введене поняття *складного* (*складеного*) оператора. Складний оператор складається з двох та більше простих операторів, об'єднаних в операторні дужки Begin-End.

Як **приклад** розглянемо фрагмент *Pascal*-програми, який реалізує фрагмент алгоритму, представлений блок-схемою (рис. 3.3).

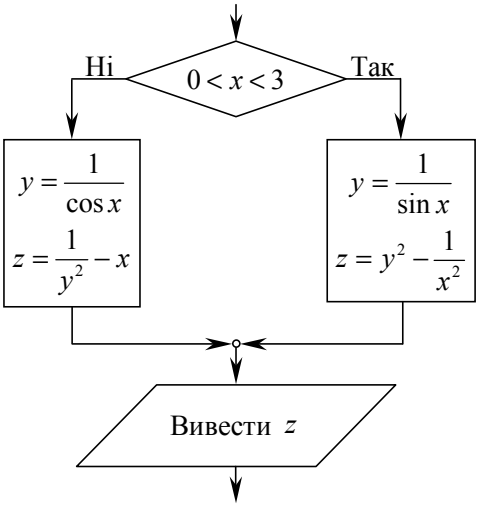


Рис. 3.3. Фрагмент блок-схеми алгоритму

```
…
If (x>0) And (x<3) Then
                    Begin
                      y:=1/Sin(x);
                      z:=Sqr(y)–1/Sqr(x)
                    End
                  Else
                    Begin
                      y:=1/Cos(x);
                      z:=1/Sqr(y)–x
                    End;
WriteLn('Z=', Z:10:3);
…
```

Бачимо, що у блоках Then і Else знаходяться по два оператори присвоєння, тому їх необхідно було об'єднати в складений оператор.

Відзначимо, що елементами складного оператора можуть бути будь-які оператори мови *Pascal*, у тому числі умовні, складні та інші.

Якщо в блоці Then або Else оператора умовного переходу також використовується умовний оператор, це означає, що оператор умовного переходу має *вкладену конструкцію*.

Як **приклад** складемо програму обчислення усіх (дійсних і комплексних) коренів квадратного рівняння вигляду $ax^2 + bx + c = 0$. Схема алгоритму розв'язання цієї задачі представлена на рис. 3.4.

Бачимо, що за введеними вхідними даними a, b і c обчислюється дискримінант квадратного рівняння D. Якщо він дорівнює нулю (як зазначено у підрозд. 2.2 логічний вираз $D = 0$ варто записувати в програмі у вигляді $|D| < \varepsilon$), то визначаються і виводяться на екран ЕОМ однакові дійсні корені $x_1 = x_2 = \frac{-b}{2a}$. У противному випадку, можливі два варіанти: 1) $D > 0$, коли різні дійсні корені квадратного рівняння визначаються за формулою

$x_{1,2} = \dfrac{-b \pm \sqrt{D}}{2a}$. 2) $D < 0$, коли маємо різні комплексні корені $x_{1,2} = \dfrac{1}{2a}\left[-b \pm i\sqrt{|D|}\right]$, де i – уявна одиниця ($i^2 = -1$).

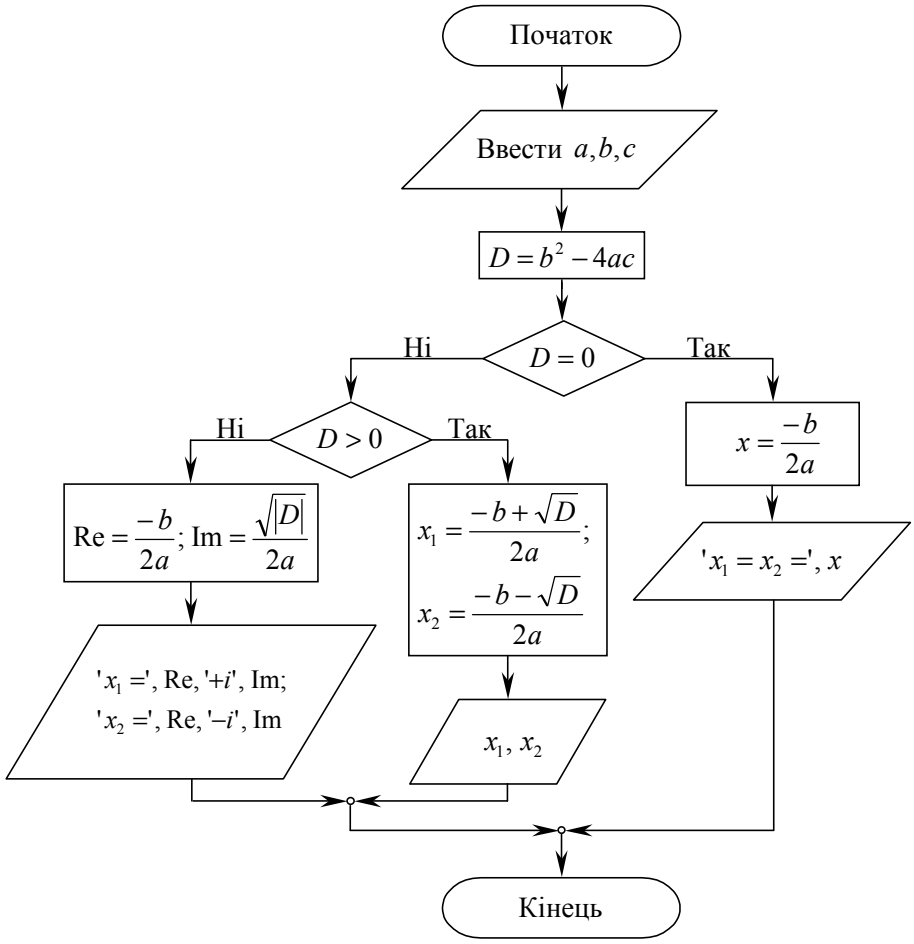


Рис. 3.4. Блок-схема алгоритму обчислення коренів квадратного рівняння

```
Program Example3;
Uses Crt;
Const E=0.00001;
Var a,b,c,D,X1,X2,X,Re,Im:Real;
BEGIN
  ClrScr;
  WriteLn('Введіть коефіцієнти квадратного рівняння');
  Write('a='); ReadLn(a);
  Write('b='); ReadLn(b);
```

```
  Write('c='); ReadLn(c);
  D:=Sqr(b)–4*a*c;
  ClrScr;
  WriteLn('Корені квадратного рівняння:');
  If Abs(D)<E Then
                  Begin
                    X:=–b/(2*a);
                    WriteLn('X1=X2=', X:4:3)
                  End
                Else
                  If D>0 Then
                           Begin
                             X1:=(–b+Sqrt(D))/(2*a);
                             X2:=(–b–Sqrt(D))/(2*a);
                             WriteLn('X1=', X1:4:3);
                             WriteLn('X2=', X2:4:3)
                           End
                         Else
                           Begin
                             Re:=–b/(2*a); Im:=Sqrt(Abs(D))/(2*a);
                             WriteLn('X1=', Re:4:3, '+i*', Im:4:3);
                             WriteLn('X2=', Re:4:3, '–i*', Im:4:3)
                           End;
  Readln
END.
```

Оскільки при знаходженні коренів квадратного рівняння можливі три різних розв'язання, виникає необхідність реалізовувати розгалужений обчислювальний процес, який містить три вітки обчислень. Він реалізується за допомогою оператора умовного переходу вкладеної конструкції.

Відзначимо, що вибрати ту або іншу вітку обчислення при програмній реалізації алгоритму завжди можна за допомогою оператора умовного переходу. Однак при великій кількості таких віток доведеться використовувати умовний оператор з великим ступенем (глибиною) вкладеності. Теоретично глибина вкладеності не обмежується, але з її збільшенням зростає складність і погіршується структура програми.

3.1.2. Оператор варіанта

Оператор варіанта можна вважати узагальненням оператора умовного переходу, оскільки він дає можливість виконати один з багатьох різних операторів у залежності від значення деякого виразу, який називається *селектором*. Як і умовний оператор, оператор варіанта використовується в повній і скороченій формах.

Синтаксис оператора варіанта в *повній формі* має вигляд:

Case <*Селектор*> **Of**
<*Список_міток_1*>: <*Оператор_1*>;
<*Список_міток_2*>: <*Оператор_2*>;
…
<*Список_міток_N*>: <*Оператор_N*>
Else <*Оператор*>
End;

де Case (вибір), Of (з), Else, End – службові слова;
Селектор – вираз будь-якого скалярного типу, крім дійсного;
Оператор – будь-який оператор мови *Pascal*;
Список міток (*або список констант вибору*) – список значень селектора або одне його значення. Зазначені константи розділяються в списку комами і повинні мати той же тип, що і селектор.

Оператор Case призводить до виконання того оператора, якому передує константа вибору, що дорівнює значенню селектора, або діапазон вибору, в якому знаходиться значення селектора. Якщо такої константи вибору або діапазону вибору не існує, то виконується оператор блоку Else. Далі в будь-якому випадку виконується оператор, що знаходиться в програмі за оператором варіанта.

Як **приклад** наведемо блок-схему алгоритму (рис. 3.5) і відповідну їй *Pascal*-програму ідентифікації введеного з клавіатури символу.

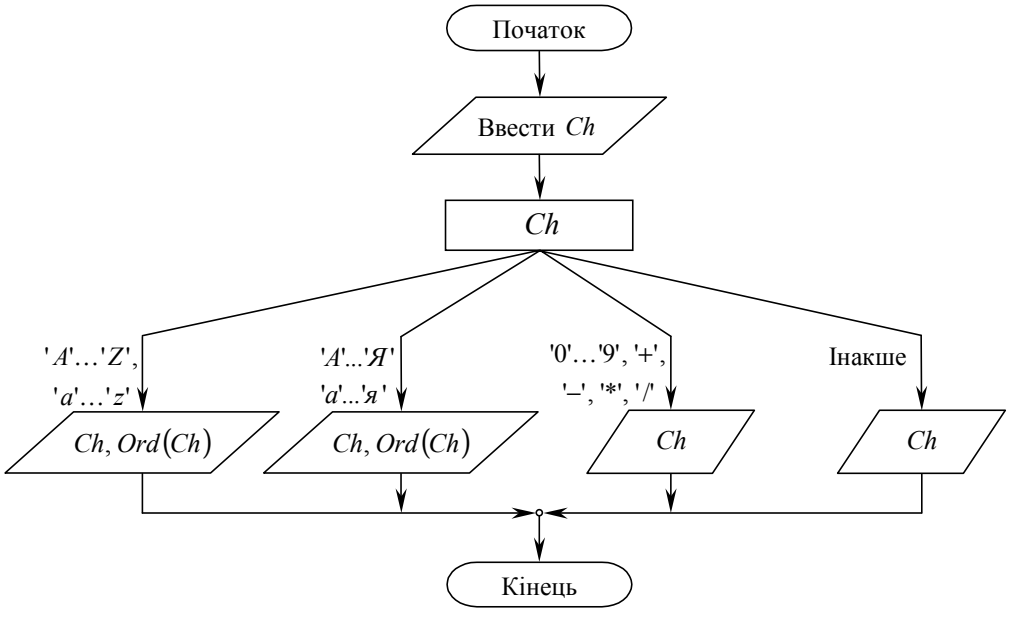


Рис. 3.5. Блок-схема алгоритму візуалізації введеного символу

```
Program Example4;
Uses Crt;
Var Ch:Char;
BEGIN
  ClrScr;
  WriteLn('Наберіть будь-який символ на клавіатурі');
  ReadLn(Ch);
  Case Ch of
     'A'..'Z','a'..'z':
          Begin
            WriteLn('Це символ латинського алфавіту ', Ch);
            WriteLn('Його порядковий номер ', Ord(Ch))
          End;
     'А'..'Я', 'а'..'п', 'р'..'я':
          Begin
            WriteLn('Це символ російського алфавіту ', Ch);
            WriteLn('Його порядковий номер ', Ord(Ch))
          End;
     '0'..'9':          WriteLn('Це цифра ', Ch);
     '+','–', '*','/':  WriteLn('Це знак арифметичної операції ', Ch)
     Else WriteLn('Це спеціальний символ ', Ch)
  End;
  ReadLn
END.
```

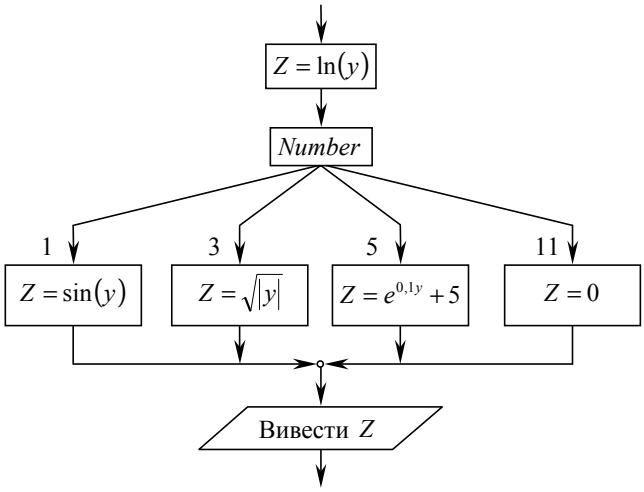
У цій програмі селектором виступає змінна символьного типу Ch. Якщо введене значення цієї змінної (введений символ) міститься в діапазоні великих або малих літер латинського алфавіту, то на екрані ЕОМ з'являться повідомлення про це, сам символ, а також повідомлення про його порядковий номер відповідно до таблиці *ASCII*-кодів. Аналогічні результати з'являться на екрані, якщо значення селектора Ch виявиться в діапазоні великих або малих літер російського алфавіту. Якщо введений символ – цифра з діапазону 0…9, то на екрані можна буде бачити відповідну цифру, а якщо Ch відповідає одному зі значень списку **'+'**, **'–'**, **'*'**, **'/'**, то на екран ЕОМ виведеться один зі знаків арифметичних операцій. Якщо ж введений символ не відповідає жодному зі значень даного списку констант вибору і не входить ні в один з перерахованих вище діапазонів вибору, то на екран виведеться повідомлення про те, що введений символ є спеціальним. Таким чином, оператор варіанта Case дозволив ефективно реалізувати розгалужений обчислювальний процес, якій містить п'ять віток обчислень.

Оператор варіанта в *скороченій формі* не містить конструкцію Else, отже, його синтаксис має вигляд:

Case <*Селектор*> **Of**
  <*Список_міток_1*>: <*Оператор_1*>;
  <*Список_міток_2*>: <*Оператор_2*>;
  ...
  <*Список_міток_N*>: <*Оператор_N*>
End;

Дія даного оператора аналогічна дії оператора варіанта в повній формі. Оператор варіанта в скороченій формі вибирає для виконання той оператор, одна з міток якого дорівнює поточному значенню селектора. Після виконання обраного оператора, управління в програмі передається до оператора, що знаходиться після оператора Case.

Як **приклад** складемо фрагмент *Pascal*-програми, який реалізує алгоритм, зображений на рис. 3.6.




```
…
Z:=Ln(y);
Case Number of
    1:  Z:=Sin(y);
    3:  Z:=Sqrt(Abs(y));
    5:  Z:=Exp(0.1*y)+5;
    11: Z:=0
End;
WriteLn('Z=', Z:10:3);
…
```

Рис. 3.6. Фрагмент алгоритму вибору

У залежності від значення селектора, значення функції $z = f(y)$ обчислюється за відповідною формулою: якщо Number=1, то $z = \sin(y)$; якщо Number=3, то $z = \sqrt{|y|}$; якщо Number=5, то $z = e^{0,1y} + 5$; якщо Number=11, то $z = 0$. Далі обчислене значення функції виводиться на екран ЕОМ, а якщо селектор приймає відмінне від перерахованих вище значень, то на екран виводиться значення функції $z = \ln y$.

3.1.3. Оператор безумовного переходу

Оператор безумовного переходу передає управління тому оператору в *Pascal*-програмі, якому передує мітка. Загальний синтаксис цього оператора має вигляд:

GoTo <*Мітка*>;

де GoTo (йти до) – службове слово;
Мітка – ціле число без знака, один символ або рядкова константа. Мітка повинна бути описана в розділі Label, як мітка оператора (див. підрозд. 2.3).

Як **приклад** розглянемо роботу *Pascal*-програми Example5.

Як видно, за заданим значенням аргументу X (на початку він дорівнює A) обчислюється значення функції Y:=Sin(X), після чого аргумент збільшується на величину h і здійснюється безумовний перехід на рядок програми, позначений міткою D. Таким чином, розв'язується задача *табулювання функції*, тобто за аналітичним виразом для функції $y = f(x)$ обчислюється таблиця її значень на відрізку $[A; B]$ з кроком h.

```
Program Example5;
Uses Crt;
Label D;
Var X,Y,A,h:Real;
BEGIN
   ClrScr;
   WriteLn('Введіть значення A і h');
   ReadLn(A,h);
   X:=A;
D: Y:=Sin(X);
   WriteLn('X=', X:3:3,' Y=', Y:3:3);
   X:=X+h;
   GoTo D
END.
```

Реалізація алгоритму табулювання в програмі Example5 здійснена неграмотно, оскільки, з теоретичної точки зору, ця програма буде працювати нескінченно. Отже, даний алгоритм не відповідає одній з тих основних вимог, які висуваються до алгоритмів, – вимозі *результативності*, тобто за скінченне число кроків він не забезпечує остаточного результату.

Процес реалізації багаторазових повторень однакових обчислень, до якого зводиться розв'язання задачі табулювання функції, називається циклічним, а правила його реалізації будуть представленні у наступному підрозділі.

Зазначимо, що застосування оператора GoTo в мові *Pascal* є необов'язковим і навіть небажаним. Справа в тім, що присутність цього оператора в програмі погіршує її структуру, наочність і читабельність, програму стає важко відлагоджувати і модифікувати. Оператор безумовного переходу іноді доводиться використовувати при виникненні особливих ситуацій у програмі.

3.2. Циклічні обчислювальні процеси. Оператори мови Pascal для їх програмної реалізації

Обчислювальний процес називають *циклічним*, якщо він містить багаторазові обчислення за однаковими математичними залежностями, але для різних значень величин (змінних), які входять у ці залежності. Багаторазово повторювані ділянки обчислень називаються *циклом*, а змінні, що модифікуються в циклі, називаються *змінними циклу*.

Існує алгоритм, якого необхідно строго дотримуватися для грамотної реалізації циклічних обчислень. Він має назву *алгоритм циклічної структури*. На рис. 3.7 показаний загальний вигляд циклічного алгоритму.

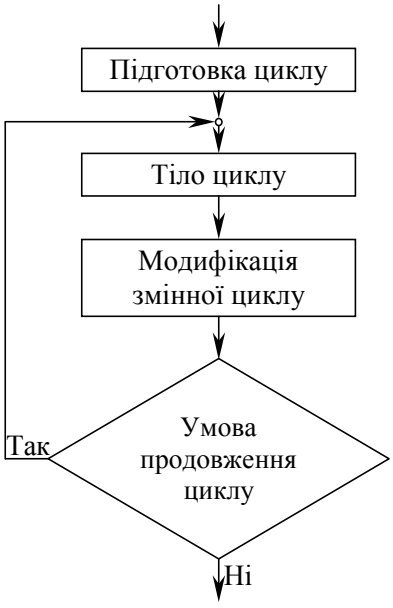


Рис. 3.7. Схема алгоритму циклічної структури

Підготовка циклу полягає у завданні початкових значень змінним циклу перед найпершим його виконанням.

Тіло циклу містить дії, які повторюються в ньому. Відзначимо, що самі дії завжди однакові, при цьому їхні багаторазові повторення здійснюються при різних значеннях змінних циклу.

Модифікацією називається зміна значень змінних циклу перед кожним новим повторенням (перед кожною новою ітерацією) циклу.

Управління циклом полягає в перевірці умови продовження або закінчення циклу. Згідно зі схемою 3.7 перевіряється умова продовження циклу з наступним переходом на початок тіла циклу, коли умова виконується, або виходом з циклу, коли умова не виконується.

Тепер знову звернемося до приклада з п. 3.1.3. Алгоритм табулювання функції, реалізований у програмі Example5, є прикладом циклічного алгоритму, оскільки містить:

1) підготовку циклу (змінній X за допомогою оператора присвоєння X:=A задається початкове значення, яке дорівнює A);

2) тіло циклу (обчислення значення функції Y:=Sin(X) з наступним виведенням на екран значень X і Y);

3) модифікацію змінної циклу (змінна циклу X на кожній ітерації циклу збільшує своє значення на величину кроку h).

Як видно, у даному алгоритмі відсутній механізм управління циклом, оскільки перехід на початок тіла циклу здійснюється без перевірки умови продовження або закінчення циклу. Наслідком відсутності одного з основних пунктів алгоритму циклічної структури є теоретично нескінченне табулювання усюди визначеної функції $y = \sin(x)$.

Для програмної реалізації циклічних обчислювальних процесів у мові *Pascal* існують спеціальні оператори – *оператори циклу*, застосування яких виключає необхідність використовувати оператор GoTo і тим самим дозволяє створювати ефективні, компактні та читабельні програми.

3.2.1. Оператор циклу з параметром

Даний оператор доцільно використовувати, якщо число повторень у циклі заздалегідь відомо. Тому *оператор циклу з параметром* інакше ще називають *оператором циклу з відомим числом повторень*.

Загальний синтаксис даного оператора має вигляд:

For <*Параметр_циклу*>:=<*Вираз_1*> [**To** / або / **DownTo**] <*Вираз_2*> **Do** <*Оператор*>;

де For (для), To (до), DownTo (вниз до), Do (зробити або виконати) – ключові (службові) слова;
Параметр_циклу – змінна будь-якого простого (скалярного) типу, крім дійсного;
Вираз_1 і *Вираз_2* – вирази, які можуть приймати тільки скалярні значення, тип яких повинен співпадати з типом *Параметра_циклу*;
Оператор – будь-який оператор мови *Pascal*.

У структурно-логічних блок-схемах (див. підрозд. 1.2), цикл з параметром зображується за допомогою блоку модифікації (заголовка циклу). Усередині даного блоку параметру циклу призначається ім'я, а також початкове та кінцеве його значення. Крок зміни параметра циклу дорівнює одиниці.

Як **приклад** складемо *Pascal*-програму, яка реалізує алгоритм виведення всіх великих букв латинського алфавіту, представлений блок-схемою на рис. 3.8.

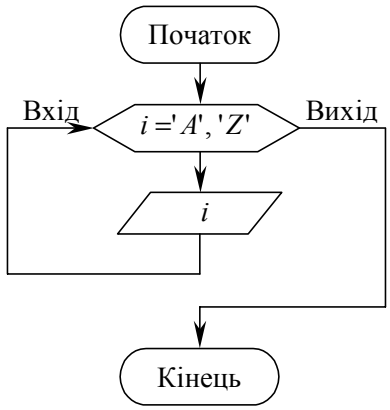


Рис. 3.8. Блок-схема алгоритму виведення символів латинського алфавіту

```
Program Example6;
Uses Crt;
Var i:Char;
BEGIN
  ClrScr;
  For i:='A' To 'Z' Do Write(i,' ');
  ReadLn
END.
```

У результаті виконання даної програми на екрані ЕОМ з'явитися рядок:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Даний приклад ілюструє окремий випадок, коли параметр циклу одночасно є і змінною циклу – символ алфавіту, який змінюється від початкового значення ***'A'*** (підготовка циклу) до кінцевого значення ***'Z'***. Тобто управління циклом полягає в тому, що останнє повторення єдиного оператора тіла циклу відбувається, коли значення i дорівнює ***'Z'***. Модифікація змінної циклу теж здійснюється автоматично: усі символи виводяться в порядку їх слідування, як би з одиничним кроком. Зрозуміло, що число повторень у даному циклі дорівнює числу букв латинського алфавіту – 26.

При використанні в циклі For службового слова DownTo кожне значення *Параметра_циклу* буде зменшуватися, починаючи від *Виразу_1* до *Виразу_2*, з кроком, який дорівнює –1. Таким чином, якщо в програмі Example6 організувати цикл

```
For i:='Z' DownTo 'A' Do Write(i,' ');
```

то на екрані ЕОМ з'являться символи латинського алфавіту в зворотному порядку:

Z Y X W V U T S R Q P O N M L K J I H G F E D C B A

Слід зазначити, що цикл For може взагалі не виконатись, якщо при використанні в його заголовку ключового слова To початкове значення параметра циклу (*Вираз_1*) буде більше кінцевого значення (*Вираз_2*). При використанні DownTo, навпаки, якщо *Вираз_1* < *Вираз_2*, то цикл не виконається жодного разу.

Тепер розглянемо задачу про обчислення скінченної суми, необхідність у розв'язанні якої часто виникає у найрізноманітніших розділах природознавства. Нехай потрібно обчислити $S = \sum_{k=1}^{n} \frac{k^2+1}{k}$. З практичної точки зору обчислення значення S за цією формулою здійснюється підсумовуванням елементів послідовності, які отримуються підстановкою у формулу загального члена $\frac{k^2+1}{k}$ значень параметра $k = 1, 2, 3, \ldots, n$. Тобто $S = \frac{1^2+1}{1} + \frac{2^2+1}{2} + \ldots + \frac{n^2+1}{n}$.

Ідея алгоритму обчислення скінченної суми полягає в обчисленні поточного доданка числової послідовності $y = \frac{k^2+1}{k}$ з наступним додаванням його до суми попередніх доданків за формулою $S := S + y$. Зрозуміло, що при обчисленні першого доданку сума попередніх дорівнює нулю.

Схему алгоритму таких обчислень наведено на рис. 3.9, а текст *Pascal*-програми має наступний вигляд:

```
Program Example7;
Uses Crt;
Var k,n:Integer;
      S,y:Real;
BEGIN
  ClrScr;
  Write('Введіть кількість доданків у сумі n=');
  ReadLn(n);
  S:=0;
  For k:=1 To n Do
  Begin
    y:=(Sqr(k)+1)/k;
    S:=S+y
  End;
  WriteLn('S=', S:3:3);
  ReadLn
END.
```

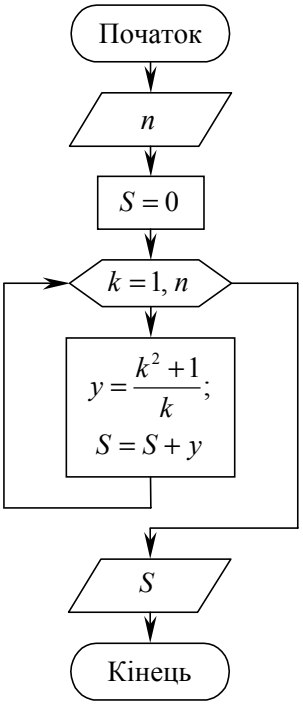


Рис. 3.9. Блок-схема алгоритму обчислення суми $S = \sum_{k=1}^{n} \frac{k^2+1}{k}$

Перед початком накопичування суми у комірку пам'яті ЕОМ з ім'ям S записується 0. На першій ітерації циклу (параметр k дорівнює 1) за формулою $y = \frac{1^2 + 1}{1} = 2$ обчислюється перший доданок y, після цього за допомогою оператора присвоєння поточне значення S замінюється на нове ($S = 0 + 2 = 2$). На другій ітерації циклу (коли k дорівнює 2) за формулою $y = \frac{2^2 + 1}{2} = 2{,}5$ обчислюється другий доданок, який знову додається до суми, що міститься в комірці з ім'ям S (до першого доданку за формулою $S = 2 + 2{,}5 = 4{,}5$). Описаний процес буде продовжуватися до того часу, поки не буде доданий останній доданок, що обчислюється, коли параметр циклу k дорівнює n.

Відзначимо важливу обставину: оскільки в циклі виконуються два оператори, то їх необхідно було об'єднати в складений оператор за допомогою операторних дужок Begin-End[9].

Досить часто доводиться зустрічатися з випадками, коли число повторень у циклі заздалегідь невідомо, однак його можна обчислити на базі вхідних даних для розв'язання задачі. Табулювання функції є класичним прикладом такої задачі. Отже, необхідно обчислити всі значення функції $y = \sin(x)$ на відрізку $[a; b]$ з кроком h. Зрозуміло, що кількість усіх точок x, у яких необхідно обчислювати функцію y можна визначити за формулою $N = \frac{b - a}{h} + 1$.

Схему алгоритму розв'язання даної задачі наведено на рис. 3.10, а текст *Pascal*-програми наведено поряд з рис. 3.10.

Оскільки параметром циклу не може бути змінна дійсного типу x, довелося ввести додаткову змінну цілого типу i, яка визначає номер поточного повторення в циклі. Кількість повторень у циклі повинна бути тільки цілою, тому при її обчисленні використовується функція перетворення типу Round (див. підрозд. 2.1).

Результати роботи даної програми мають наступний вигляд:

Введіть вхідні дані: a = –3 b = 2.5 h = 0.5

x = –3.000 y = –0.141	x = –1.000 y = –0.841	x = 1.000 y = 0.841
x = –2.500 y = –0.598	x = –0.500 y = –0.479	x = 1.500 y = 0.997
x = –2.000 y = –0.909	x = 0.000 y = 0.000	x = 2.000 y = 0.909
x = –1.500 y = –0.997	x = 0.500 y = 0.479	x = 2.500 y = 0.598

[9] Нагадаємо, що поняття складеного або складного оператора було введено в п. 3.1.1 при розгляді синтаксису оператора умовного переходу. Усі міркування з цього приводу справедливі та мають місце при реалізації циклічних процесів за допомогою оператора циклу For.

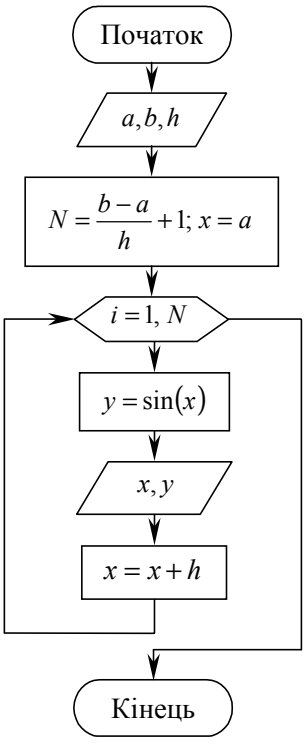


Рис. 3.10. Блок-схема алгоритму табулювання функції $y = \sin(x)$

```
Program Example8;
Uses Crt;
Var i,N:Integer;
      a,b,h,x,y:Real;
BEGIN
  ClrScr;
  WriteLn('Введіть вхідні дані:');
  Write('a='); ReadLn(a);
  Write('b='); ReadLn(b);
  Write('h='); ReadLn(h);
  N:=Round((b–a)/h)+1;
  x:=a;
  For i:=1 To N Do
  Begin
    y:=Sin(x);
    WriteLn('x=', x:6:3,' y=', y:6:3);
    x:=x+h
  End;
  ReadLn
END.
```

При реалізації обчислювальних алгоритмів часто зустрічаються і такі випадки, коли число повторень у циклі заздалегідь невідомо і не може бути обчислено за допомогою вхідних даних для розв'язання задачі. При цьому відома умова закінчення або продовження обчислювального процесу. Спеціально для цих випадків у мові *Pascal* існують *оператор циклу з передумовою* і *оператор циклу з післяумовою*.

3.2.2. Оператори циклу з передумовою і післяумовою. Ітераційні цикли

Загальний синтаксис *оператора циклу з передумовою* має вигляд:

While <*Умова*> **Do** <*Оператор*>;

де While (поки), Do (виконувати, робити) – службові слова;
Оператор – будь-який оператор мови *Pascal*;
Умова – умова продовження циклу, яка у тексті програми записується як логічний вираз.

Виконання оператора While-Do проілюстровано на схемі, зображеній на рис. 3.11.

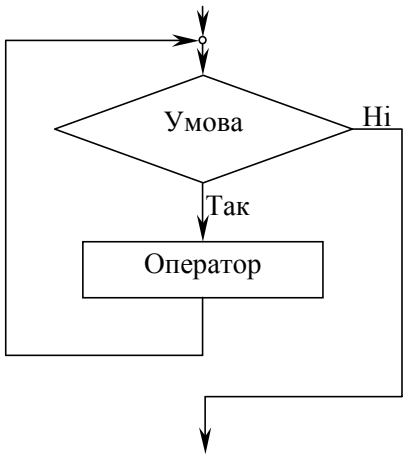


Рис. 3.11. Схематична ілюстрація виконання оператора циклу з передумовою

Бачимо, що виконання *Оператора* в циклі відбувається до того часу, поки виконується *Умова*. Коли *Умова* перестає виконуватись, то відбувається вихід із циклу, тобто управління в програмі передається до оператора, що знаходиться після циклу. Якщо умова продовження циклу не виконується із самого початку, то даний цикл не буде виконаний жодного разу.

Як **приклад** складемо програму для розв'язання наступної задачі. Дано два цілих числа a і b. Знайти кількість цілих чисел з відрізка $[a; b]$, які кратні N, та вивести їх на екран ЕОМ.

```
Program Example9;
Uses Crt;
Var a,b,S,N,Kol,Number:Integer;
BEGIN
  ClrScr;
  WriteLn('Введіть a,b,N'); ReadLn(a,b,N);
  Number:=a; Kol:=0;
  While Number<=b Do
  Begin
    If Number Mod N = 0 Then
                          Begin
                            Write(Number, ' ');
                            Kol:=Kol+1
                          End;
    Number:=Number+1
  End;
  WriteLn;
  WriteLn('Число значень, кратних ', N,' ,у межах від ',a,' до ',b,
  ' дорівнює ',Kol);
  ReadLn
  END.
```

Для розв'язання цієї задачі в програмі Example9 за допомогою оператора While-Do організовано цикл, у якому відбувається перебирання усіх чи-

сел (Number) від a до b з кроком одиниця. Якщо поточне число кратне N[10], воно виводиться на монітор ЕОМ і, крім цього, обчислюється (накопичується) значення кількості (Kol) таких чисел аналогічно алгоритму обчислення суми, реалізованому в програмі Example7.

Оскільки в циклі виконуються два оператори (оператор умовного переходу в скороченій формі та оператор присвоєння), то їх необхідно було об'єднати до складеного оператора за допомогою операторних дужок Begin-End.

Результати роботи програми Example9 мають вигляд:

```
Введіть a,b,N
13  69  5
15 20 25 30 35 40 45 50 55 60 65
Число значень, кратних 5 у межах від 13 до 69 дорівнює 11
```

Загальний синтаксис *оператора циклу з післяумовою* має вигляд:

Repeat
<*Оператор_1*>; <*Оператор_2*>;
<*Оператор_3*>; …; <*Оператор_N*>
Until <*Умова*>;

де Repeat (повторювати), Until (до того часу, поки) – службові слова;
Оператор_1, *Оператор_2*, …, *Оператор_N* – будь-які оператори мови *Pascal*;
Умова – умова виходу з циклу, яка у тексті програми записується як логічний вираз.

Виконання оператора Repeat-Until проілюстровано на схемі, зображеній на рис. 3.12.

Дія оператора Repeat-Until аналогічна дії оператора While-Do. Як видно, виконання всіх операторів у циклі буде відбуватися до того часу, поки не виконається *Умова*. Коли *Умова* виконається, то відбудеться вихід з циклу з передачею управління тому оператору в програмі, який слідує після оператора циклу. Відзначимо, що перевірка умови виходу з циклу здійснюється після чергової його ітерації, тому цикл Repeat-Until у будь-якому випадку виконається щонайменше один раз.

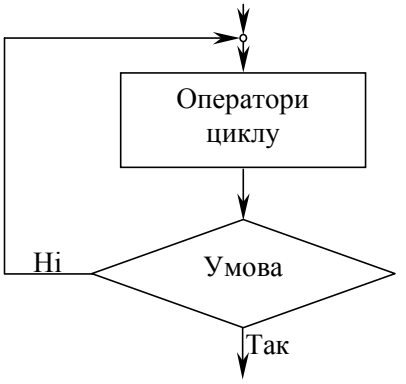


Рис. 3.12. Схематична ілюстрація виконання оператора циклу з післяумовою

[10] Якщо остача від ділення двох цілих операндів Number і N дорівнює нулю, то це означає, що число Number кратне числу N. З операцією Mod, яка дозволяє знаходити зазначену остачу, ми познайомилися в підрозд. 2.1.

Як **приклад** наведемо цикл Repeat-Until для розв'язання попередньої задачі.

```
…
Number:=a; Kol:=0;
Repeat
If Number Mod N = 0 Then
            Begin
              Write(Number,' ');
              Kol:=Kol+1
            End;
Number:=Number+1
Until Number>b;
…
```

Дія даного циклу еквівалентна дії циклу з передумовою із програми Example9.

Службові слова Repeat-Until за дією аналогічні операторним дужкам Begin-End, тобто між ними можна помістити групу операторів, відділених один від одного крапкою з комою.

Зазначимо, що розв'язання розглянутої задачі можна було ефективно здійснити і за допомогою оператора циклу For. Для цього необхідно було б визначити кількість повторень у циклі, як загальне число цілих чисел з відрізка $[a; b]$.

Як вже було зазначено, існує досить великий клас обчислювальних алгоритмів, у яких кількість повторень однакових обчислень в принципі не може бути заздалегідь визначена. У цих випадках, в процесі реалізації циклічних обчислень утворюється послідовність деяких значень $z_1, z_2, z_3, \ldots, z_n$, яка повинна збігатися до границі відповідного значення S, тобто $\lim\limits_{n\to\infty} z_n = S$. Кожне значення даної числової послідовності z_i обчислюється з урахуванням попереднього значення z_{i-1} і є в порівнянні з ним більш точним наближенням до бажаного результату S. Цикли, які реалізують алгоритм послідовних наближень до результату (до границі послідовності), мають назву *ітераційних*.

Критерієм закінчення послідовних наближень до результату є умова $|z_n - z_{n-1}| \le \varepsilon$, де ε – прийнятна похибка одержуваного результату. Значення n називають кількістю ітерацій, необхідних для одержання результату, а сам результат S ототожнюють або з z_n, або з $\frac{z_n + z_{n-1}}{2}$ в залежності від характеру збіжності ітераційного процесу.

Розв'язання задачі про обчислення суми нескінченного ряду є класичною ілюстрацією реалізації ітераційного циклічного процесу. Отже, потрібно знайти суму членів ряду[11] $e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^{n-1}}{(n-1)!}$ з заданою похибкою $\varepsilon = 10^{-4}$.

[11] Зазначимо, що виклик у програмі таких стандартних функцій, як Sin(x), Cos(x), Ln(x), і т. п., саме і зводиться до реалізації алгоритмів підсумовування відповідних функціональних рядів, тільки з більшим ступенем точності.

Як видно, розв'язання даної задачі зводиться до обчислення суми $S = 1 + \sum_{n=2}^{\infty} y_n(x)$, де $y_n(x) = \frac{x^{n-1}}{(n-1)!}$. Аналогічно обчисленню скінченної суми, розглянутому в п. 3.2.1, в даному випадку необхідно обчислювати поточні значення доданків $y_n(x)$ і здійснювати накопичування суми за формулою $S_n = S_{n-1} + y_n$, де $n = 2, 3, 4, \ldots, \infty$, а доданок S_1 дорівнює першому члену ряду y_1 і дорівнює одиниці ($S_1 = y_1 = 1$).

Для спрощення обчислювання доданків y_n представимо степінь x^{n-1} у вигляді добутку $x^{n-1} = \underbrace{x \cdot x \cdot x \cdot \ldots \cdot x}_{(n-1)\text{ разів}} = \prod_{i=1}^{n-1} x$, а факторіал $(n-1)! = 1 \cdot 2 \times \times 3 \cdot \ldots \cdot (n-1) = \prod_{i=1}^{n-1} i$. Таким чином, $y_n = \prod_{i=1}^{n-1} \frac{x}{i}$, отже, $S = 1 + \sum_{n=2}^{\infty} \prod_{i=1}^{n-1} \frac{x}{i}$.

Алгоритм обчислення $\prod$ аналогічний алгоритму обчислення $\sum$ і полягає у послідовному накопичуванні добутку множників. Для знаходження $y_n = \prod_{i=1}^{n-1} \frac{x}{i} = P$ необхідно обчислити поточний множник x/i і помножити його на добуток попередніх множників за формулою $P := P \cdot x/i$. Зрозуміло, що при обчисленні першого множника добуток попередніх дорівнює одиниці.

Алгоритм обчислення шуканої суми $S = 1 + \sum_{n=2}^{\infty} \prod_{i=1}^{n-1} \frac{x}{i}$ реалізований в *Pascal*-програмі Example10. У даній програмі використані прості змінні: n – номер члена ряду (номер поточної ітерації); y – поточне значення члена ряду, яке визначається як добуток $P = \prod_{i=1}^{n-1} \frac{x}{i}$; S – поточне значення суми n членів ряду.

Як видно, і добуток P, і сума S в даній програмі обчислюються (накопичуються) в одному циклі. Це можливо, оскільки в добутку P поточний множник з номером n відрізняється від попереднього в x/i разів, причому зв'язок між порядковим номером члена ряду і знаменником множника x/i визначається рівністю $i = n - 1$.

```
Program Example10;
Uses Crt;
Var x,y,Eps,S,P:Real;
      n:Integer;
BEGIN
  ClrScr;
  WriteLn('Введіть x,Eps'); ReadLn(x,Eps);
  n:=1; y:=1; WriteLn(n:2,' y=',y:10:6);
  S:=0; P:=1;
  While Abs(y)>Eps Do
  Begin
    S:=S+y;
    n:=n+1; P:=P*x/(n–1); y:=P;
    WriteLn(n:2,' y=', y:10:6)
  End;
  WriteLn('Exp(', x:2:2, ')=', S, 'Ехрточне(', x:2:2, ')=', Exp(x));
  ReadLn
END.
```

В цьому ж циклі для візуалізації ітераційного процесу передбачено виведення номера ітерації і значення відповідного члена ряду. Коли поточне значення члена ряду неістотне за абсолютним значенням ($|y_n| \le \varepsilon$), його не додають до загальної суми S, хоча і виводять на екран ЕОМ. Крім отриманого результату, для перевірки виводиться значення стандартної функції e^x.

Результати роботи даної програми мають вигляд:

```
Введіть x, Eps   1  1.0E–4
1 y = 1.000000 | 6 y = 0.008333
2 y = 1.000000 | 7 y = 0.001389
3 y = 0.500000 | 8 y = 0.000198
4 y = 0.166667 | 9 y = 0.000025
5 y = 0.041667 | Exp(1.00) = 2.7182539683E+00 Ехрточне(1.00) = 2.7182818285E+00
```

Як видно, з похибкою, яка не перевищує задане значення $\varepsilon = 10^{-4}$, можна обмежитися 8 членами розкладання в ряд Тейлора функції e^x в точці $x = 1$, тобто $e = 2{,}7183 \pm 10^{-4}$.

Необхідно ще раз підкреслити, що, оскільки в мові *Pascal* немає стандартних функцій для обчислення степеня і факторіала, довелось звести їх обчислення до обчислення відповідних добутків, тобто здійснити так зване *непряме обчислення*.

При організації непрямих обчислень можливо та доцільно використовувати й інший підхід. Отже нам необхідно здійснити накопичування суми за ітераційною формулою $S_n = S_{n-1} + y_n$. Представимо y_n у вигляді $y_n = y_{n-1} \cdot \varphi_n$.

Підставимо у формулу для y_n замість n величину $(n-1)$ і дістанемо співмножник φ_n у вигляді $\varphi_n = \frac{y_n}{y_{n-1}} = \frac{x^{n-1}\ (n-2)!}{(n-1)!\, x^{n-2}}$. Таким чином $\varphi_n = \frac{x}{n-1}$, отже

рекурсивна[12] схема $\left.\begin{array}{c} y_1 = 1;\ y_n = \frac{y_{n-1} \cdot x}{n-1}; \\ S_1 = 1;\ S_n = S_{n-1} + y_n; \\ n = 2, 3, 4, \ldots, \infty \end{array}\right\}$ визначає алгоритм непрямого обчислення суми ряду, який був реалізований у програмі Example10.

Для управління роботою циклів усіх трьох розглянутих видів (For-Do, Repeat-Until і While-Do) у сучасних версіях мови *Pascal* передбачені додаткові можливості. Реалізуються ці можливості за допомогою стандартних процедур – Break і Continue.

Якщо процедура Continue міститься в тілі циклу, то частина цього циклу, яка знаходиться після Continue, ігнорується і виконується наступна ітерація циклу.

Якщо в тілі циклу знаходиться процедура Break, то вона викличе негайне завершення роботи циклу. Далі буде виконуватися оператор, який знаходиться за циклом.

Проілюструємо сказане на простих **прикладах**.

```
…
x:=15;
For i:=1 To 5 Do
Begin
  WriteLn('x=', x);
  If i<4 Then Continue;
  x:=x+1
End;
…
```

Зрозуміло, що з п'яти ітерацій циклу тільки остання буде здійснюватися з урахуванням модифікованого значення змінної циклу x.

```
x = 15
x = 15
x = 15
x = 15
x = 16
```

[12] Рекурсією називається спосіб завдання функції, за яким значення обчислюваної функції для аргументу з більшим номером виражається відомим чином через значення цієї ж функції для аргументу з меншим номером.

```
…
While True Do
Begin
  ClrScr;
  Write('Введіть символ ');
  ReadLn(S);
  If S=' ' Then Break;
  WriteLn('Ви набрали – ', S);
  ReadLn
End;
WriteLn('Ви набрали символ "пропуск".
         Кінець програми!');
…
```

Виконання нескінченного циклу[13] для візуалізації набраного символу буде продовжуватися до того часу, поки користувач не натисне на клавіатурі ЕОМ пропуск.

3.2.3. Вкладені цикли

У підрозд. 3.2 поки розв'язувались відносно прості *одновимірні* задачі, такі, як табулювання функції з *однією* змінною, обчислення *одинарної* скінченної суми і т. п. Для розв'язання більш складних багатовимірних задач такого типу необхідно використовувати так звані *вкладені* або *складні цикли*, які утворюються, якщо в тілі циклу так само використовується циклічна структура. Цикл, що містить у собі циклічну структуру, називається *зовнішнім* відносно неї, а зазначена циклічна структура, яка знаходиться в тілі зовнішнього циклу, називається *внутрішньою*.

З відносним ступенем умовності вкладений цикл можна характеризувати рівнем його вкладеності. На рис. 3.13 зображений складний цикл другого рівня вкладеності, а на рис. 3.14 – першого.

[13] Оскільки в циклі з передумовою перевіряється умова продовження циклу, а в циклі з післяумовою умова його закінчення, то конструкції

```
While True Do
Begin
…
End;
```

```
Repeat
…
Until False;
```

дозволяють реалізовувати нескінченні цикли, управляти роботою яких можна тільки за допомогою процедур Break і Continue. Цикл **Repeat … Until** KeyPressed; також може працювати нескінченно, до того часу, поки користувачем не буде натиснута будь-яка клавіша клавіатури обчислювальної системи.

З рис. 3.13 видно, що зовнішній цикл з параметром *i* має рівень 0, внутрішній цикл з параметром *j* – рівень 1, а внутрішній цикл з параметром *k* має рівень 2. Зрозуміло, що цикл з параметром *j* є внутрішнім відносно циклу з параметром *i*. В той же час він є зовнішнім відносно циклу з параметром *k*. Рівні вкладеності циклів, зображених на рис. 3.14, визначаються так: зовнішній цикл з параметром *i* має рівень 0, внутрішній цикл з параметром *j* – рівень 1, внутрішній цикл с передумовою також 1-го рівня вкладеності.

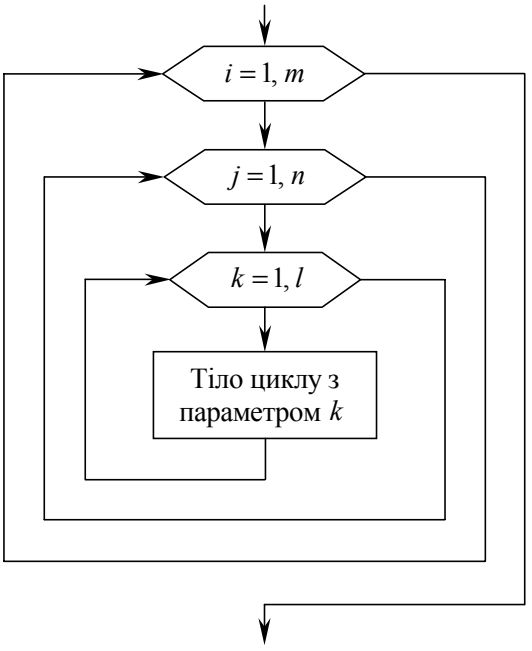


Рис. 3.13. Цикл 2-го рівня вкладеності

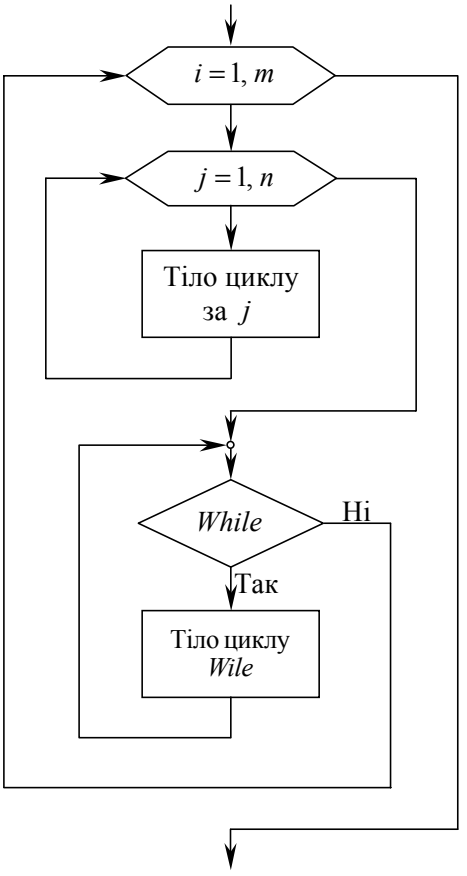


Рис. 3.14. Цикл 1-го рівня вкладеності

З теоретичної точки зору максимальний рівень вкладеності циклу необмежений, однак, із практичної – він не може бути нескінченним і обмежується обсягом оперативної пам'яті обчислювальної системи.

Параметри циклів різних рівнів змінюються не одночасно. Спочатку, при фіксованих початкових значеннях параметрів циклів з меншим рівнем, усі свої значення змінить параметр циклу найвищого рівня вкладеності. Потім на один крок зміниться значення параметра циклу передостаннього рівня і знову повністю виконається самий внутрішній цикл.

Циклічний процес буде продовжуватися до того часу, поки не змінить усі свої значення параметр самого зовнішнього циклу. Загальне число ітерацій вкладеного циклу можна легко визначити. Так, для циклу, зображеного на рис. 3.13, $N = m \cdot n \cdot l$. Якщо в циклі, зображеному на рис. 3.14, число повторень простого циклу з передумовою дорівнює l, то $N = m \cdot (n + l)$.

По закінченні теоретичної частини даного підрозділу сформулюємо основне правило організації вкладених циклів: *внутрішні та зовнішні прості цикли можуть бути будь-якими з трьох існуючих видів, при цьому всі оператори внутрішнього циклу повинні цілком знаходитися в тілі зовнішнього циклу*.

В інженерній і науковій діяльності дуже часто доводиться зустрічатися з розв'язанням двовимірних задач, програмна реалізація яких вимагає організації подвійних циклів. Як **приклад** розглянемо розв'язання двох таких розповсюджених задач.

1. У межах прямокутної області $\begin{Bmatrix} 0 \le x \le a \\ 0 \le y \le b \end{Bmatrix}$ потрібно побудувати поверхню, яка описується функцією двох змінних $Z = \sin \pi\xi \cos \pi\eta$, де $\xi = \frac{x}{a}$; $\eta = \frac{y}{b}$ – безрозмірні координати, границі зміни яких, вочевидь, лежать у межах від 0 до 1 ($0 \le \xi; \eta \le 1$).

Зрозуміло, що дана задача зводиться до табулювання функції двох змінних $Z(\xi; \eta)$, у результаті розв'язання якої буде отримана таблиця значень цієї функції, за якою можна буде побудувати шукану поверхню. Табулювання двовимірної функції Z можна, наприклад, організувати так. При фіксованому початковому значенні $\xi = 0$ обчислювати значення Z для всіх η, які нас цікавлять: $Z(0; 0)$, $Z(0; h_\eta)$, $Z(0; 2h_\eta)$, …, $Z(0; nh_\eta = 1)$. Після цього необхідно збільшити аргумент ξ на один крок і знову звести задачу до розглянутої в п. 3.2.1 задачі табулювання функції з одною змінною за допомогою обчислення значень $Z(h_\xi; 0)$, $Z(h_\xi; h_\eta)$, $Z(h_\xi; 2h_\eta)$, …, $Z(h_\xi; 1)$. Останній ланцюжок обчислень здійснюється для кінцевого значення $\xi = 1$: $Z(mh_\xi = 1; 0)$, $Z(1; h_\eta)$, $Z(1; 2h_\eta)$, …, $Z(1; 1)$. Для проведення описаних обчислень необхідно задатися значеннями кроків по осях ξ і η (h_ξ і h_η), а для програмної реалізації даного алгоритму організувати складний цикл першого рівня вкладеності: зовнішній – для зміни значення змінної ξ і внутрішній – для зміни η. У якості зазначених зовнішнього і внутрішнього циклів можна вибрати цикли з відомим числом повторень, кількість яких визначається відповідно як $m = 1/h_\xi + 1$ і $n = 1/h_\eta + 1$.

Pascal-програма, яка відповідає словесно описаному алгоритму, має такий вигляд:

```
Program Example11;
Uses Crt;
Var i,j,m,n:Integer;
      Hx,Hy,x,y,Z:Real;
BEGIN
  ClrScr;
  Write('Введіть значення Hx= '); ReadLn(Hx);
  Write('Введіть значення Hy= '); ReadLn(Hy);
  m:=Round(1/Hx)+1; n:= Round (1/Hy)+1;
  x:=0;
  For i:=1 To m Do
  Begin
    y:=0;
    For j:=1 To n Do
    Begin
      Z:=Sin(Pi*x)*Cos(Pi*y);
      Write(Z:6:3, ' ');
      y:=y+Hy
    End;
    x:=x+Hx;
    WriteLn
  End;
  ReadLn
END.
```

При вхідних даних $h_\xi = h_\eta = 0{,}1$ за допомогою даної програми можна одержати наступну таблицю для функції $Z(\xi; \eta)$:

ξ \ η	**0**	**0,1**	**0,2**	**0,3**	**0,4**	**0,5**	**0,6**	**0,7**	**0,8**	**0,9**	**1**
0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0,1	0.309	0.294	0.250	0.182	0.095	0.000	–0.095	–0.182	–0.250	–0.294	–0.309
0,2	0.588	0.559	0.476	0.345	0.182	0.000	–0.182	–0.345	–0.476	–0.559	–0.588
0,3	0.809	0.769	0.655	0.476	0.250	0.000	–0.250	–0.476	–0.655	–0.769	–0.809
0,4	0.951	0.905	0.769	0.559	0.294	0.000	–0.294	–0.559	–0.769	–0.905	–0.951
0,5	1.000	0.951	0.809	0.588	0.309	0.000	–0.309	–0.588	–0.809	–0.951	–1.000
0,6	0.951	0.905	0.769	0.559	0.294	0.000	–0.294	–0.559	–0.769	–0.905	–0.951
0,7	0.809	0.769	0.655	0.476	0.250	0.000	–0.250	–0.476	–0.655	–0.769	–0.809
0,8	0.588	0.559	0.476	0.345	0.182	0.000	–0.182	–0.345	–0.476	–0.559	–0.588
0,9	0.309	0.294	0.250	0.182	0.095	0.000	–0.095	–0.182	–0.250	–0.294	–0.309
1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

2. Обчислити значення математичного оператора $S = \sum_{i=1}^{m} i^2 \prod_{j=1,3,5,\ldots}^{n} \frac{i+j}{j^2}$.

Даний подвійний оператор для заданих значень m і n з математичної точки зору можна обчислити так:

$$S = 1^2 \frac{1+1}{1^2} \cdot \frac{1+3}{3^2} \cdot \frac{1+5}{5^2} \cdot \ldots \cdot \frac{1+n}{n^2} + 2^2 \frac{2+1}{1^2} \cdot \frac{2+3}{3^2} \cdot \frac{2+5}{5^2} \cdot \ldots \cdot \frac{2+n}{n^2} +$$
$$+ 3^2 \frac{3+1}{1^2} \cdot \frac{3+3}{3^2} \cdot \frac{3+5}{5^2} \cdot \ldots \cdot \frac{3+n}{n^2} + \ldots + m^2 \frac{m+1}{1^2} \cdot \frac{m+3}{3^2} \cdot \frac{m+5}{5^2} \cdot \ldots \cdot \frac{m+n}{n^2}.$$

Як видно, процес обчислення значення S можна трактувати як алгоритмічний процес накопичування суми, кожен доданок якої є добутком. Для програмної реалізації таких обчислень необхідно організувати вкладений цикл: зовнішній – для обчислення суми і внутрішній – для розрахунку доданків цієї суми (для накопичування добутку при фіксованому значенні індексу i). Зовнішній цикл доцільно організувати, як цикл із відомим числом повторень, а внутрішній цикл – наприклад, як цикл з передумовою.

Pascal-програма Example12 реалізує описаний обчислювальний алгоритм.

```
Program Example12;
Uses Crt;
Var i,j,m,n:Integer;
      S,P:Real;
BEGIN
  ClrScr;
  Write('Введіть значення m= ');
  ReadLn(m);
  Write('Введіть значення n= ');
  ReadLn(n); S:=0;
  For i:=1 To m Do
  Begin
    P:=1; j:=1;
    While j<=n Do
    Begin
      P:=P*(i+j)/Sqr(j);
      j:=j+2
    End;
    S:=S+Sqr(i)*P
  End;
  WriteLn('S=', S:10:4);
  ReadLn
END.
```

3.3. Заздалегідь визначений обчислювальний процес. Підпрограми-процедури і підпрограми-функції для його реалізації на ЕОМ

Циклічна форма повторюваності, яка досить докладно описана в підрозд. 3.2, характерна тим, що однакові дії багаторазово виконуються на *одному етапі обробки інформації*. Досить часто зустрічається інша форма повторюваності, коли одна послідовність дій виконується на *різних етапах обробки інформації*. Якщо представити такий алгоритм, то в різних його місцях будуть зустрічатися фрагменти (пункти алгоритму), однакові за виконуваними діями, які відрізняються тільки значеннями вхідних даних. Програма, складена за таким алгоритмом, буде містити однакову послідовність операторів, яка відповідає кожному повторюваному фрагменту. На рис. 3.15 наведена графічна ілюстрація розділу операторів програми, у різних місцях якого містяться повторювані фрагменти обчислень P. Зрозуміло, що така програма програє в наочності та читабельності програмам без повторюваних фрагментів, крім того, у цій програмі зростає імовірність появи помилок, її складніше відлагоджувати.

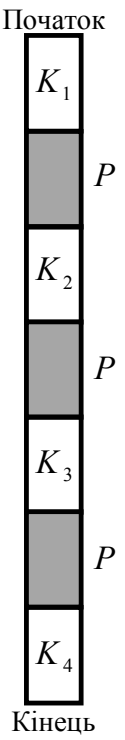


Рис. 3.15

Для більш ефективної реалізації алгоритмів з описаною повторюваністю в мові *Pascal* введене поняття *підпрограми*. Підпрограма являє собою самостійну програмну одиницю, у вигляді якої оформлюється повторювана група операторів P. Підпрограма записується тільки один раз, причому окремо від розділу операторів, а у відповідних місцях цього розділу забезпечується тільки звернення (посилання) до неї. Таким чином, тепер програма буде складатися тільки з неповторюваних фрагментів обчислень K_1, K_2, K_3, K_4, а графічну ілюстрацію її роботи зображено на рис. 3.16.

Підпрограми дозволяють розділити програму на окремі логічно завершені та взаємопов'язані компоненти, що дає можливість виконувати їх розробку різним програмістам відносно незалежно один від одного. Використання апарата підпрограм[14] дозволяє скоротити об'єм і поліпшити структуру *Pascal*-програми, зменшити імовірність виникнення помилок та спростити процес її відлагодження.

[14] Використання підпрограм у процесі створення програм іноді називають процедурним способом програмування.

У мові *Pascal* підпрограми реалізуються у вигляді *процедур* і *функцій*, причому стандартні підпрограми вже неодноразово використовувалися при складанні найрізноманітніших програм.

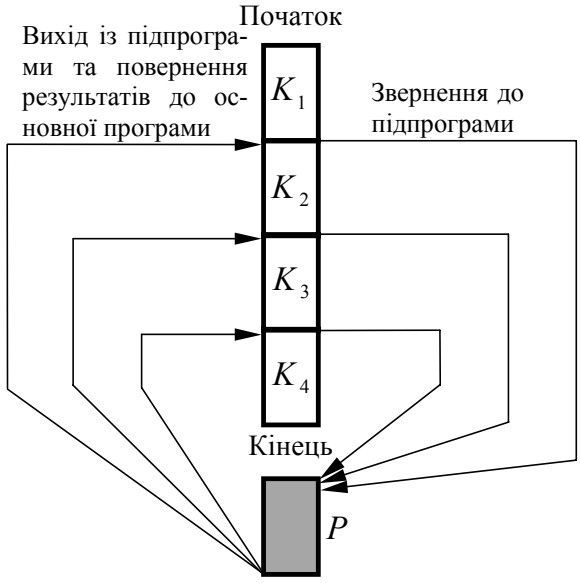


Рис. 3.16

Наприклад, стандартна процедура WriteLn дозволяла виводити список виведення на екран ЕОМ. Безпараметрична процедура ClrScr очищувала екран і переводила курсор у верхній лівий його кут. Стандартна функція Exp(b) дозволяла обчислити значення математичної функції $y = e^x$ заданого аргументу $x = b$. Ці підпрограми працювали для нас у режимі "чорної шухляди", тобто на вхід необхідно було подати вхідні дані (це потрібно для підпрограм з параметрами), щоб на виході одержати очікуваний результат. Алгоритм перетворення вхідних даних у результат нам не був відомий.

Зараз розглянемо задачі, у яких потрібно на базі стандартних підпрограм створювати свої завершені програмні компоненти – *підпрограми користувача*. У підрозд. 2.7 вже наводились механізми розробки і використання підпрограм-функцій. Ці підпрограми дозволяють обчислювати значення однієї величини, яке присвоюється імені функції. Якщо ж необхідно обчислювати не одне, а декілька значень або взагалі нічого не потрібно обчислювати, а треба виконати необчислювальні дії (наприклад, виведення тексту на екран ЕОМ), то доцільно використовувати підпрограму-процедуру.

Підпрограма-процедура описується у відповідному розділі *Pascal*-програми і аналогічно підпрограмі-функції складається із заголовка та блоку. Заголовок процедури має такий вигляд:

Procedure <*Ім'я*> (<*Список_параметрів*>);

де Procedure (процедура) – службове слово;
Ім'я – ім'я процедури, яке призначається згідно з загальними правилами побудови ідентифікаторів;
Список_параметрів – список імен (*формальних параметрів*) для позначення вхідних даних і результатів роботи процедури із зазначенням їх типів.

У мові *Pascal* можливо створювати безпараметричні процедури користувача. Для таких процедур заголовок має вигляд **Procedure** <*Ім'я*>;.

Блок процедури, аналогічно блоку функції, також складається з розділів опису (міток, констант, типів, змінних, процедур і функцій) та розділу операторів. Наприкінці розділу операторів процедури ставиться крапка з комою.

Опис підпрограми-процедури (так само як і підпрограми-функції) сам по собі ніяких дій не викликає. Для виконання підпрограми потрібно у відповідному місці основної програми помістити звернення до неї, що здійснюється за допомогою *оператора виклику процедури*, який має вигляд:

<*Ім'я*> (<*Список_аргументів*>);

де *Ім'я* – ім'я процедури, до якої відбувається звернення;
Список_аргументів – список конкретних значень, виразів або імен, які підставляються на місце формальних параметрів процедури при її виконанні.

Аргументи, що перераховані в списку оператора процедури, називаються *фактичними параметрами*. При виклику процедури формальні параметри заміняються на фактичні в порядку їх слідування, тому число та тип формальних і фактичних параметрів повинні строго співпадати.

Як **приклад** розглянемо задачу обчислення дійсних коренів біквадратного рівняння вигляду $K_1x^4 + K_2x^2 + K_3 = 0$.

Для розв'язання біквадратного рівняння скористаємося підстановкою $t = x^2$, яка дозволить звести його до квадратного рівняння $K_1t^2 + K_2t + K_3 = 0$. Якщо дискримінант D квадратного рівняння більше або дорівнює нулю, то можна знайти його дійсні корені t_1 і t_2. Якщо значення $t_{1,2} \geq 0$, то обчислити корені біквадратного рівняння можна за формулою $x_{1,2,3,4} = \pm\sqrt{t_{1,2}}$.

Для програмної реалізації описаної схеми розв'язання будемо використовувати підпрограму-процедуру для обчислення дійсних коренів квадратного рівняння вигляду $at^2 + bt + c = 0$. На рис. 3.17,***б*** наведено структурно-логічну блок-схему цього алгоритму. На відміну від схеми алгоритму основної програми дана схема містить *вхід* (на вхід подаються вхідні дані) і *вихід* (на виході вказуються результати роботи підпрограми).

Блок-схема основного алгоритму – алгоритму розв'язання біквадратного рівняння – зображена на рис. 3.17,***а***.

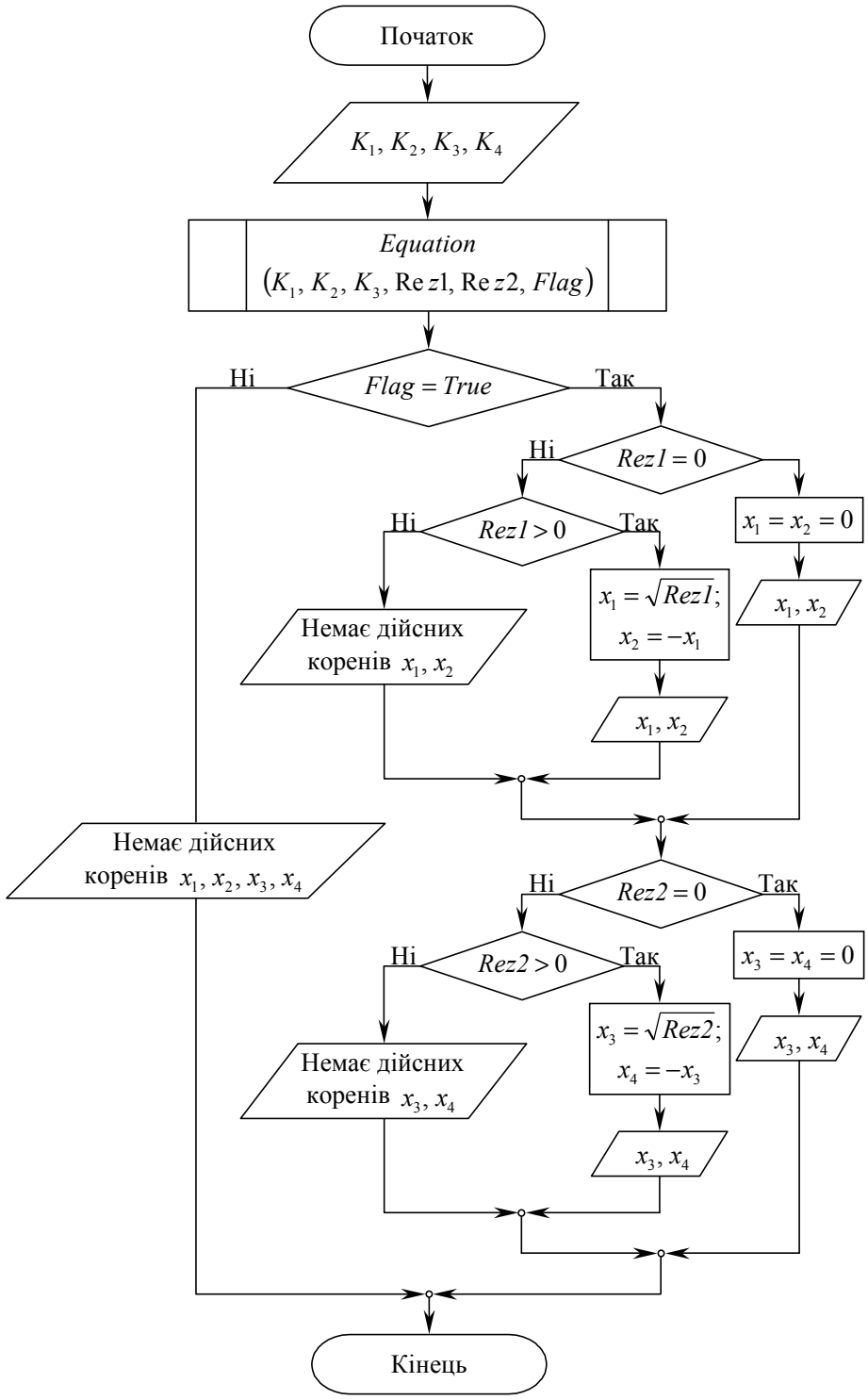


Рис. 3.17,***а***. Загальна блок-схема алгоритму

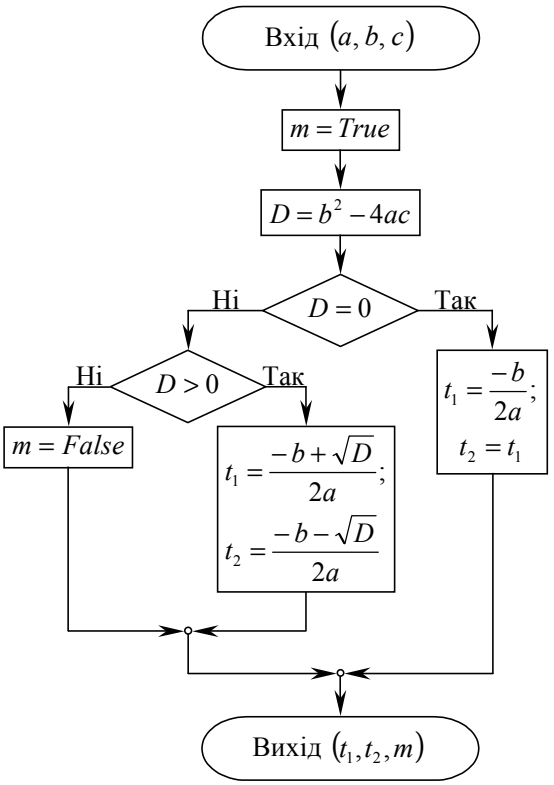


Рис. 3.17, ***б***. Блок-схема алгоритму *Equation* для обчислення коренів квадратного рівняння

Pascal-програма, яка відповідає даним алгоритмам, має наступний вигляд:

```
Program Example13;
Uses Crt; Const E=0.00001;
Var K1,K2,K3,Rez1,Rez2,X1,X2,X3,X4:Real; Flag:Boolean;
Procedure Equation(a,b,c:Real; Var t1,t2:Real; Var m:Boolean);
Var D:Real;
Begin
  m:=True; D:=Sqr(b)–4*a*c;
  If Abs(D)<E Then
                Begin
                     t1:=–b/(2*a); t2:=t1
                End
              Else
                If D>0 Then
                          Begin
                            t1:=(–b+Sqrt(D))/(2*a);
                            t2:=(–b–Sqrt(D))/(2*a)
                          End
                        Else m:=False
End;
BEGIN
  ClrScr; WriteLn('Введіть коефіцієнти біквадратного рівняння:');
  Write('K1= '); ReadLn(K1);
  Write('K2= '); ReadLn(K2); Write('K3= '); ReadLn(K3);
  Equation (K1, K2, K3, Rez1, Rez2, Flag);
  If Flag Then
            Begin
              If Abs(Rez1)<E Then
                              Begin
                                X1:=0; X2:=0; WriteLn('X1=', X1:3:3, ' X2=', X2:3:3)
                              End
                            Else
                              If Rez1>0 Then
                                          Begin
                                            X1:=Sqrt(Rez1); X2:=-X1;
                                            WriteLn('X1=', X1:3:3, ' X2=', X2:3:3)
                                          End
                                        Else WriteLn('Не має дійсних коренів X1, X2');
              If Abs(Rez2)<E Then
                              Begin
                                X3:=0; X4=0; WriteLn('X3=', X3:3:3, ' X4=', X4:3:3)
                              End
                            Else
                              If Rez2>0 Then
                                          Begin
                                            X3:=Sqrt(Rez2); X4:=-X3;
                                            WriteLn('X3=', X3:3:3, ' X4=', X4:3:3)
                                          End
                                        Else WriteLn('Не має дійсних коренів X3, X4')
            End
          Else WriteLn('Рівняння не має дійсних коренів!');
  ReadLn
END.
```

Результати роботи даної програми:

```
Введіть коефіцієнти біквадратного рівняння:
K1 = 1.52
K2 = –3.39
K3 = 1.23
X1 = 1.332 X2 = –1.332
X3 = 0.675 X4 = –0.675
```

В даній програмі розв'язання квадратного рівняння здійснюється за допомогою процедури Equation. У заголовку цієї процедури перераховані (описані) формальні параметри:

a, b і c – сталі коефіцієнти при невідомому *t* квадратного рівняння, які є вхідними даними процедури. Відзначимо, що змінні, які використовуються для передачі вхідних даних у підпрограму (процедуру або функцію), часто називаються *параметри-значення*;

m – необхідна ознака наявності у квадратному рівнянні дійсних коренів (якщо $D \geq 0$, то m=True, у противному випадку m=False);

t1, t2 – дійсні корені квадратного рівняння, які можна обчислити в процедурі тільки коли m=True. Параметри t1, t2 і m перелічуються після ключового слова Var і є *параметрами-змінними* – результатами виконання процедури.

Склад блоку (або тіла) процедури такий:

а) описова частина, де визначається додаткова змінна D (дискримінант рівняння), яка бере участь у процесі обчислення результатів. Змінна D не є результатом, а також не відноситься до вхідних даних, отже, вона необхідна і має зміст тільки усередині даної процедури і називається *локальною змінною*. Значення локальної змінної недоступне в основній програмі.

б) розділ операторів, у якому реалізуються алгоритми розв'язання квадратного рівняння і визначення ознаки m.

Для розв'язання поставленої задачі в основній програмі процедура Equation викликається один раз. При цьому в тілі процедури відбувається заміна формальних параметрів a, b, c, t1, t2, m на фактичні K1, K2, K3, Rez1, Rez2, Flag. Далі виконується послідовність дій, яка передбачена операторами процедури, після чого в основній програмі (відразу після оператора виклику процедури) будуть доступні результати Rez1, Rez2 і Flag.

У залежності від отриманих результатів можливі наступні розв'язки біквадратного рівняння:

1. Дискримінант *D* квадратного рівняння менше нуля (Flag=False) або $D \geq 0$ (Flag=True), але корені Rez1 і Rez2 від'ємні. У цих випадках біквадратне рівняння не має дійсних коренів.

2. Корені квадратного рівняння додатні ($Rez1, Rez2 \geq 0$), тоді маємо чотири дійсних корені біквадратного рівняння: X1, X2, X3, X4.

3. Якщо Rez1>0, а Rez2<0, то маємо два дійсних корені: X1, X2.

4. Якщо Rez1<0, а Rez2>0, то маємо два дійсних корені: X3, X4.

Ще раз підкреслимо найбільш важливі положення, які необхідно враховувати при використанні апарата підпрограм.

Формальні параметри являють собою змінні, які номінально (формально) присутні в процедурі. Вони визначають місце і тип підстановки фактичних параметрів. Фактичні параметри, навпаки, являють собою реальні об'єкти в програмі. Ними замінюють формальні параметри в тілі процедури при її виклику, і саме з ними виконуються дії, передбачені операторами в тілі процедури.

Таким чином, виклик процедури можна розглядати як передачу параметрів: вхідні дані передаються на вхід процедури, а результати передаються на вихід (повертаються в основну програму). При цьому параметри-значення (вхідні дані) передаються *за значенням*, а параметри-змінні (результати) – *за посиланням*. Основна відмінність цих способів передачі параметрів полягає в тім, що присвоювання значення параметру-змінної всередині процедури одночасно здійснюється і для відповідного фактичного параметра. Таким чином, параметр-змінна не може бути константою або виразом. Параметри-значення в процедурі аналогічні локальним змінним, тобто всі зміни, які можуть одержувати ці параметри в процесі виконання процедури, недоступні в основній програмі, оскільки не викликають ніяких змін відповідних фактичних параметрів. Параметр-значення може бути константою, значенням або виразом.

Описані способи передачі параметрів можна проілюструвати при роботі наступної програми:

```
Uses CRT;
Var X,Y,Z:Real;
Procedure W(X,Y:Real; Var Z:Real);
Var Buff:Real;
Begin
  Buff:=X; X:=Y; Y:=Buff;
  Z:=X–Y;
  WriteLn('В середині процедури:');
  WriteLn('X=', X:3:3, ' Y=', Y:3:3, ' Z=',Z:3:3)
End;
BEGIN
  ClrScr;
  X:=1; Y:=2;
  W(X,Y,Z);
  WriteLn('В основній програмі:');
  WriteLn('X=', X:3:3, ' Y=', Y:3:3, ' Z=', Z:3:3);
  ReadLn
END.
```

В середині процедури:
X = 2.000 Y = 1.000 Z = 1.000
В основній програмі:
X = 1.000 Y = 2.000 Z = 1.000

У даній програмі імена формальних і фактичних параметрів співпадають. В принципі це допускається, оскільки не виникає помилок при передачі даних. Однак імена формальних і фактичних параметрів доцільно вибирати різними, так як це було зроблено в програмі Example13. Це робить програму більш наочною і свідчить про гарний стиль програмування.

Імена локальних і глобальних змінних вибирати однаковими небажано, оскільки можуть виникнути помилки в програмі, пов'язані з плутаниною імен.

Взагалі поняття локальної та глобальної змінної введене в мові *Pascal* у зв'язку з присутністю в ній поняття блочної структури – аналогії з поняттям вкладеної структури для операторів умовного переходу та циклу. Блочну структуру утворюють програма разом з описаними в ній процедурами і функціями. Блок, що містить у своєму розділі описів інший блок (процедуру або функцію), називається *зовнішнім*. Блок, який міститься в розділі описів зовнішнього блоку, називається *внутрішнім*. Змінні та інші об'єкти, описані в розділі описів внутрішнього блоку, є локальними об'єктами цього блоку і недоступні для зовнішніх блоків. Об'єкти, описані в програмі або зовнішньому блоці, є глобальними по відношенню до внутрішніх блоків і доступні в них.

На рис. 3.18 зображена вкладена структура з чотирьох блоків. Видно, що змінна B3 є локальною відносно блоку 4 і недоступна в блоках 3, 2 і 1. Змінні A1 і B1 є локальними в блоці 2, а їх значення недоступні ні в основній програмі, ні в блоках 3 і 4. Змінні B і C, описані в основній програмі, є глобальними для усіх внутрішніх блоків (2, 3, 4), отже, їхні значення доступні у всій програмі. Змінна A, описана в основній програмі, є одночасно і глобальною,

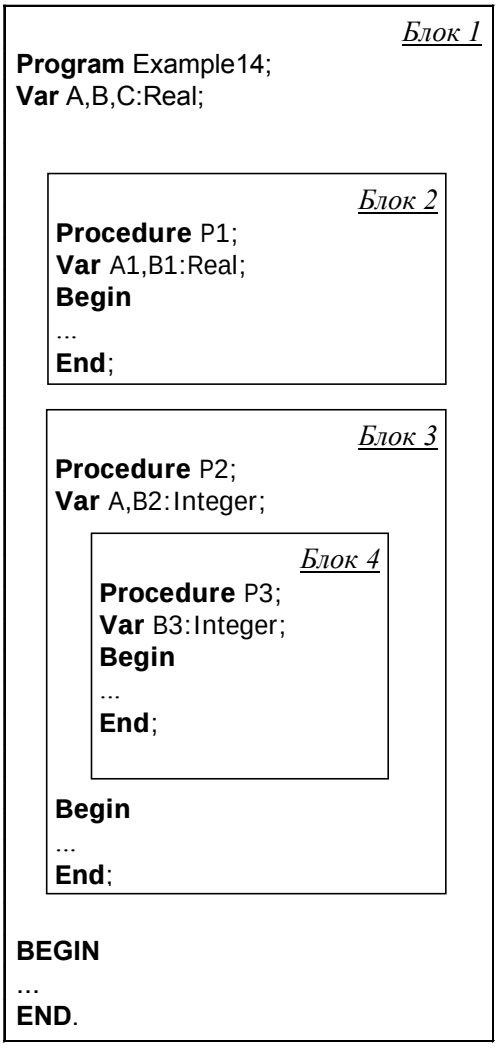


Рис. 3.18. Вкладена структура з 4-х блоків

і локальною. Глобальною вона є тільки для блоку 2, і її значення доступне як у цьому блоці, так і в основній програмі. У блоці 3 глобальна змінна A перевизначається і стає локальною для блоку 3 і глобальною для блоку 4. Змінна A може змінити своє значення в блоках 3 і 4, однак це не спричинить зміни значення глобальної змінної A у блоках 1 і 2.

Наприкінці даного розділу розглянемо розв'язання ще однієї задачі, в якій потрібно обчислити значення степеня $Z = x^y$, де значення $x \neq 0$, а показник степеня y може бути будь-яким цілим числом.

Відомо, що для обчислення степеня в мові *Pascal* не передбачені стандартна функція або процедура, отже, вираз для Z потрібно перетворити з використанням відомих стандартних функцій. У силу обмеженої області визначення функції $\ln(x)$, яка використовується для зазначених перетворень, вираз для визначення значення степеня Z буде відносно громіздким:

$$Z = \begin{cases} e^{y \ln x}, & x > 0; \\ e^{y \ln|x|}, & x < 0;\ y \bmod 2 = 0; \\ -e^{y \ln|x|}, & x < 0;\ y \bmod 2 = 1. \end{cases}$$

Вираз для степеня Z можна значно спростити, якщо попередньо обчислювати степінь з натуральним показником як добуток: $a^n = \prod_{i=1}^{n} a = P,$ де $a = x$, а $n = 0, 1, 2, 3, \ldots, |y|$. Таким чином, $Z = \begin{cases} P, & y \geq 0; \\ \dfrac{1}{P}, & y < 0. \end{cases}$

При складанні програми обчислення степеня $Z = x^y$, будемо використовувати підпрограму для знаходження степеня з натуральним показником $P = \prod_{i=1}^{n} a.$ Оскільки значення P є одним скалярним значенням, то для його обчислення можна використовувати як підпрограму-процедуру, так і підпрограму-функцію (що більш доцільно).

Розв'язання з використанням підпрограми-функції:

```
Program Example15_a;
Uses Crt;
Var X:Real;Y:Integer;
Function P(a:Real;n:Integer):Real;
Var i:Integer; S:Real;
Begin
  S:=1;
  For i:=1 To Abs(n) Do S:=S*a;
  P:=S
End;
BEGIN
  ClrScr;
  WriteLn('Введіть X,Y');
  ReadLn(X,Y);
  If Y<0 Then WriteLn('Степінь Z=', 1/P(X,Y):3:3)
         Else WriteLn('Степінь Z=', P(X,Y):3:3);
  ReadLn
END.
```

Розв'язання з використанням підпрограми-процедури:

```
Program Example15_b;
Uses Crt;
Var X,Z:Real;Y:Integer;
Procedure Power(a:Real;n:Integer; Var P:Real);
Var i:Integer;
Begin
  P:=1;
  For i:=1 To n Do P:=P*a
End;
BEGIN
  ClrScr;
  WriteLn('Введіть X,Y');
  ReadLn(X,Y);
  If Y<0 Then Power(1/X,Abs(Y),Z)
         Else Power(X,Y,Z);
  WriteLn('Степінь Z=', Z:3:3);
  ReadLn
END.
```

4. СКЛАДНІ ТИПИ ДАНИХ

Всі дані (константи або змінні), які використовувались в попередніх розділах для реалізації обчислювальних процесів розв'язання найрізноманітніших задач, називались *простими* (або *скалярними*). Цю назву вони отримали завдяки своїм головним властивостям: неподільності та упорядкованості своїх значень (див. підрозд. 2.2). Кажучи простішою мовою, *будь-яка константа або змінна скалярного типу може прийняти одне єдине значення, що присвоюється її імені*.

Часто зустрічаються ситуації, коли для обчислень необхідно використовувати досить багато змінних одного простого типу. Нехай для розв'язання гіпотетичної метеорологічної задачі необхідно використовувати в якості вхідних даних значення середніх температур кожного дня всього року. Для програмної реалізації простого алгоритму обчислення середньорічної температури в рамках даної задачі необхідно описати 365 констант (**Const** T1=–12.5; T2=–10.4; T3=–1.2; T4=–3.5; ..., T365=1.3;) і обчислити Tsr:=(T1+T2+T3+ ... +T365)/365;. Як видно, використання змінних простого типу для розв'язання таких задач є нераціональним, оскільки призводить до істотного збільшення програмного коду, погіршуючи наочність програми та ускладнюючи процес її відлагодження.

Для таких випадків мова *Pascal* надає можливість використовувати велику кількість змінних однакового типу за допомогою призначення цій множені одного *загального* (*колективного*) імені. Такі множини значень або змінних з одним загальним ім'ям одержали назву *складних* або *структурних* типів даних. Відзначимо, що можливість надати одне загальне ім'я цілій множині елементів має велике значення в програмуванні, оскільки з'являється можливість автоматизувати процеси введення, виведення та обробки значень цих елементів.

Існує чотири основні різновиди складних типів даних, які відрізняються за *способом* об'єднання окремих компонентів у структуру і *типом* компонент, які входять у цю структуру (див. рис. 2.1): регулярний тип (масиви), комбінований тип (записи), файловий тип (файли), множинний тип (множини). У даному розділі ми познайомимося тільки з масивами і текстовими файлами.

4.1. Одновимірні та багатовимірні масиви

У самому загальному випадку *масив* – це упорядкована сукупність елементів однакового типу.

Масиви бувають *одновимірними* (*лінійними*) і *багатовимірними*.

Прикладом одновимірного масиву може служити шеренга солдатів. Елементами цієї шеренги є солдати, які упорядковані в ній, наприклад, за зростом. Крім того, кожному солдату в шерензі відповідає деякий порядковий номер.

З математичної точки зору одновимірний масив це вектор (множина):

$$X = \{x_1, x_2, x_3, \ldots, x_n\},$$

де X – ім'я (ідентифікатор) масиву; x_i – компоненти або елементи масиву; $i = 1, 2, 3, \ldots, n$ – індекси (номери) компонентів масиву.

Як і прості типи даних, складний тип даних масив повинен бути обов'язково описаний у *Pascal*-програмі. Загальна форма опису масиву має такий вигляд:

<*Ім'я*>**:Array**[<*Тип_індексів*>] **Of** <*Тип_елементів*>;

де: *Ім'я* – ідентифікатор масиву;
Array (масив), Of (із) – службові слова;
Тип_індексів – може бути будь-яким скалярним типом, крім Real;
Тип_елементів – будь-який тип даних.

Як **приклад** опишемо таким способом масив $X = \{2{,}5;\ 0{,}25;\ 7{,}5;\ -3{,}9\}$. Тобто X – це масив з чотирьох елементів дійсного типу. У результаті такого опису, в оперативній пам'яті комп'ютера виділяється місце для збереження лінійної послідовності цих елементів.

```
Var X:Array[1..4] Of Real;
```

В деяких випадках зручно описувати масив у іншій формі. Спочатку в розділі типів (Type) описують тип масиву (створюють шаблон), а потім у розділі опису (Var) змінної присвоюють створений тип. **Наприклад**, потрібно описати два одновимірних масиви $A = \{2;\ 5;\ 3\cdot10^5;\ 7;\ 15;\ -21\}$ і $B = \{-10;\ 30;\ 100\}$.

```
Type Mas=Array[1..6] Of Integer;
Var A,B:Mas;
```

Масиви A і B мають однаковий тип, але різну вимірність. Тобто на масив B виділяється більше місця в пам'яті ЕОМ, ніж необхідно. Це можна робити, якщо достатньо ресурсів комп'ютера.

У деяких випадках доцільно в розділі Type створювати свій нестандартний тип індексів, а масив описувати з використанням створеного типу.

```
Type Month=(September, October, November);
Var T:Array[Month] Of Real;
```

В даному **прикладі** описаний масив середньомісячних температур осінніх місяців року.

Введення елементів одновимірного масиву також може здійснюватися різними способами:

1. Введення елементів масиву як констант.

```
Const A:Array [1..5] Of Integer=(–5,–4,–3,–2,–1);
```

Строго кажучи, це одночасно і опис масиву, і введення значень його компонентів. У цьому випадку масив цілих чисел можна тільки читати, оскільки змінювати значення констант у програмі не можна.

2. Введення елементів масиву за допомогою операторів присвоєння.

```
Var A:Array [1..5] Of Integer;
…
A[1]:=–5; A[2]:=–4; A[3]:=–3;
A[4]:=–2; A[5]:=–1;
```

Бачимо, що вибір окремого компонента одновимірного масиву здійснюється вказівкою імені масиву, слідом за яким у квадратних дужках вказується індекс цього компонента. Зрозуміло, що значення індексу повинне лежати в діапазоні, який визначається типом індексів при описі масиву.

Відзначимо, що представлений спосіб введення є нераціональним. У випадку великої кількості елементів у масиві таке введення буде захаращувати текст програми. Крім того, якщо елементи масиву A є вхідними даними для розв'язання задачі, то масові розрахунки за такою програмою будуть вимагати постійних змін програмного коду, а це неприпустимо.

3. Введення масиву за допомогою операторів введення з перерахуванням усіх його елементів.

```
Var A:Array [1..5] Of Integer;
…
ReadLn(A[1], A[2], A[3], A[4], A[5]);
```

Алгоритм введення елементів одновимірного масиву, який реалізований у даному фрагменті, уже володіє властивістю *масовості*[15], однак проблема наочності та читабельності такої програми залишається невирішеною. Отже, описаний спосіб введення теж не можна назвати раціональним.

[15] Масовістю називається властивість алгоритму бути застосовним для різноманітних наборів вихідних даних, на яких цей алгоритм визначений. Масовість, поряд з результативністю і детермінованістю являє собою одну з основних вимог, що висуваються до обчислювальних алгоритмів.

4. Введення елементів масиву за допомогою оператору циклу з параметром.

Даний спосіб введення є самим раціональним. Алгоритм його реалізації наведено на рис. 4.1.

Оскільки всі елементи масиву мають одне загальне ім'я, то в циклі досить організувати тільки зміну значення індексу поточного елемента.

Відзначимо, що введення елементів одновимірних масивів здійснюється аналогічно введенню.

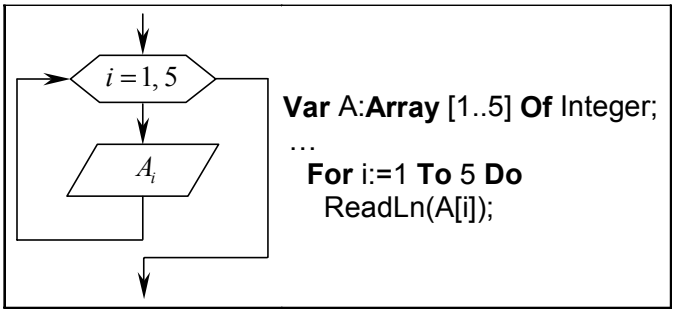


Рис. 4.1. Алгоритм введення елементів лінійного масиву

Як **приклад** розглянемо задачу про визначення екстремальних значень серед елементів одновимірного масиву, яка формулюється так. Дано одновимірний масив A вимірності n. Потрібно знайти, наприклад, мінімальний елемент даного масиву.

Алгоритм розв'язання даної задачі представлений на рис. 4.2, а його реалізація здійснюється в програмі Example16.

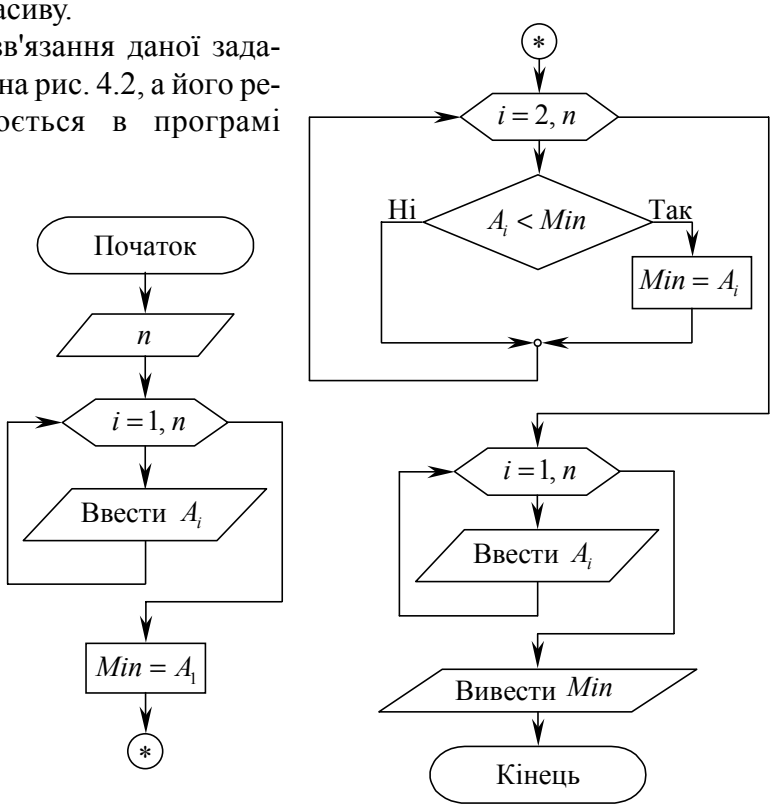


Рис. 4.2. Блок-схема розв'язання задачі про визначення мінімального елемента в масиві

Після введення вимірності та значень елементів масиву, пошук мінімального серед них здійснюється таким чином. Спочатку як мінімальний елемент вибирається перший елемент у масиві $Min = a_i$. Після цього послідовно перебираються і порівнюються з *Min* елементи, що залишилися. Якщо деяке значення a_i, $i = 2, 3, 4, \ldots, n$ виявиться менше, ніж *Min*, то це поточне значення *Min* замінюється на a_i. Після того як мінімальний елемент знайдений, здійснюється виведення одновимірного масиву і значення *Min* на екран ЕОМ.

```
Program Example16;
Uses Crt;
Var A:Array[1..100] Of Real;
    i,n:Integer;
    Min:Real;
BEGIN
  ClrScr;
  Write('Введіть вимірність масиву n= ');
  ReadLn(n);
  WriteLn('Введіть елементи масиву');
  For i:=1 To n Do
  Begin
    Write('A[', i, ']=');
    ReadLn(A[i])
  End;
  Min:=A[1];
  For i:=2 To n Do
    If A[i]<Min Then Min:=A[i];
  ClrScr;
  WriteLn('Заданий масив:');
  For i:=1 To n Do Write(A[i]:3:3, ' ');
  WriteLn;
  WriteLn('Min=', Min:3:3);
  ReadLn
END.
```

Пошук максимального елемента можна здійснити, якщо зробити в даній програмі наступні виправлення:

```
…
Max:=A[1];
For i:=2 To n Do
  If A[i]>Max Then Max:=A[i];
…
```

У деяких випадках елементи масиву не задаються як вхідні данні, а визначаються в процесі обчислень за рекурсивною формулою вигляду

$$x_i = f(x_{i-1}) \text{ або } x_{i+1} = f(x_i).$$

Як **приклад**, розглянемо задачу про знаходження тих елементів одновимірного масиву, які володіють визначеними властивостями. Нехай елементи одновимірного масиву *A* визначаються за формулою

$$\left.\begin{array}{c} a_1 = n; \\ a_{i+1} = \dfrac{14{,}9\sin(a_i) - |a_i|}{-i}, \end{array}\right\}, \text{ де } i = 1, 2, 3, \ldots, n.$$

Потрібно знайти, наприклад, середнє арифметичне від'ємних елементів масиву.

Легко помітити, що кількість елементів у масиві A дорівнює $n+1$, оскільки $a_1 = n$, а інші n елементів (a_2, a_3, …, a_{n+1}) визначаються рекурсивно за формулою $a_{i+1} = \left(14{,}9\sin(a_i) - |a_i|\right)/-i$.

Для розв'язання задачі треба знайти суму S та кількість Kol від'ємних елементів масиву A. Якщо в масиві немає від'ємних елементів ($Kol = 0$), то задача не має розв'язку. Якщо ж $Kol \neq 0$, то шукане середнє арифметичне можна обчислити, як S/Kol.

Текст програми, яка реалізує необхідні обчислення, має такий вигляд:

```
Program Example17;
Uses Crt;
Var A:Array[1..100] Of Real;  i,n,Kol:Integer;  S:Real;
BEGIN
  ClrScr; Write('Введіть значення n= '); ReadLn(n); A[1]:=n;
  For i:=1 To n Do A[i+1]:=(14.9*Sin(A[i])–Abs(A[i]))/–i;
  S:=0; Kol:=0;
  For i:=1 To n+1 Do
    If A[i]<0 Then
                Begin
                  S:=S+A[i]; Kol:=Kol+1
                End;
  WriteLn('Заданий масив:');
  For i:=1 To n+1 Do Write(A[i]:3:3, ' '); WriteLn;
  If Kol=0 Then WriteLn('У масиві немає від'ємних елементів')
             Else
             Begin
               S:=S/Kol; WriteLn('S= ', S:3:3)
             End;
  ReadLn
END.
```

Поряд з одновимірними широко використовуються *двовимірні масиви*, у яких положення елементів визначається вже двома індексами.

Зала кінотеатру може бути прикладом двовимірного масиву. Щоб відшукати потрібне крісло в ньому, необхідно знати номер ряду та номер місця в цьому ряду.

З математичної точки зору, двовимірний масив – це матриця вимірності $m \times n$:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix},$$

або скорочено: $A = \{a_{ij}\}$, де $i = 1, 2, 3, \ldots, m$ – номер рядка, $j = 1, 2, 3, \ldots, n$ – номер стовпця. Таким чином, деякий елемент матриці a_{ij} знаходиться на перетинанні i-го рядка та j-го стовпця.

З погляду розташування елементів у пам'яті ЕОМ, двовимірний масив являє собою чисту абстракцію. Оскільки оперативна пам'ять одновимірна, то і двовимірний масив розміщується в ній у вигляді лінійних послідовностей рядків або стовпців. Якщо елементи масиву розміщуються в пам'яті за рядками, то формується наступна послідовність: $a_{11}, a_{12}, \ldots, a_{1n}, a_{21}, a_{22}, \ldots, a_{2n}, \ldots, a_{m1}, a_{m2}, \ldots, a_{mn}$. Якщо масив розміщується в пам'яті за стовпцями, то послідовність елементів у ній має вигляд: $a_{11}, a_{21}, \ldots, a_{m1}, a_{12}, a_{22}, \ldots, a_{m2}, \ldots, a_{1n}, a_{2n}, \ldots, a_{mn}$.

Для резервування необхідного місця в оперативній пам'яті ЕОМ двовимірний масив необхідно описати у відповідних розділах опису *Pascal*-програми. Аналогічно одновимірному масиву двовимірний масив

$$A = \begin{pmatrix} 1{,}2 & -3{,}5 & 0{,}1 \\ 7{,}8 & 11{,}2 & -1{,}3 \\ 2{,}5 & -2{,}9 & 0{,}7 \\ -3{,}5 & 12{,}9 & 1{,}7 \end{pmatrix}$$ можна описати різними способами:

1. Опис масиву в розділі опису змінних.

```
Var A:Array[1..4,1..3] Of Real;
```

Тобто A – це матриця, яка складається з чотирьох рядків і трьох стовпців, а її елементами є значення дійсного типу.

2. Опис масиву з використанням створеного типу (створеного шаблону).

```
Type Matrix=Array[1..4,1..3] Of Real;
Var A:Matrix;
```

3. Опис двовимірного масиву на базі одновимірного.

```
Type Vector=Array[1..3] Of Real;
Var A:Array[1..4] Of Vector;
```

Як видно, двовимірний масив A описано, як одновимірний, елементами якого також є одновимірні масиви. Тобто змінна A є двовимірним масивом з чотирьох рядків, кожен з яких складається з трьох елементів.

Введення елементів двовимірного масиву може здійснюватися аналогічно введенню елементів лінійного масиву декількома способами: за допомогою операторів присвоєння і за допомогою операторів введення з перерахуванням всіх елементів. Однак самим раціональним є введення елементів у циклі, а оскільки положення елемента двовимірного масиву визначається двома індексами, то цикл для введення повинен бути подвійним. Схема алгоритму введення елементів двовимірного масиву і відповідний фрагмент *Pascal*-програми для його реалізації наведені на рис. 4.3.

Видно, що для кожного поточного рядка, номер якого i змінюється в зовнішньому циклі, оператор введення виконується n разів – для кожного

номера стовпця *j*, який змінюється у внутрішньому циклі. Таким чином, реалізується алгоритм введення елементів масиву за рядками.

Виведення елементів двовимірних масивів здійснюється аналогічно введенню.

У наведеному нижче **прикладі** здійснюються введення і виведення елементів квадратної матриці A, вимірності k і пошук добутку всіх елементів, які лежать вище головної діагоналі.

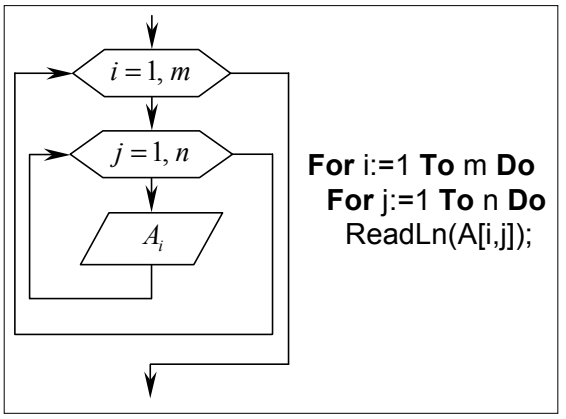


Рис. 4.3. Алгоритм введення компонентів двовимірного масиву

У програмі Example18 здійснюється введення з клавіатури ЕОМ елементів квадратної матриці за рядками з візуалізацією індексів поточного елемента.

```
Program Example18;
Uses Crt;
Var A:Array[1..100,1..100] Of Real;
      i,j,k:Integer;
      P:Real;
BEGIN
  ClrScr;
  Write('Введіть порядок матриці k= ');
  ReadLn(k);
  WriteLn('Введіть елементи матриці за рядками');
  For i:=1 To k Do
    For j:=1 To k Do
    Begin
      Write('A(', i, ';', j, ')=');
      ReadLn(A[i,j])
    End;
  P:=1;
  For i:=1 To k Do
    For j:=1 To k Do
      If j>i Then P:=P*A[i,j];
  ClrScr;
  WriteLn('Вхідний масив:');
  For i:=1 To k Do
  Begin
    For j:=1 To k Do Write(A[i,j]:5:3, ' ');
    WriteLn
  End;
  WriteLn('P=', P:5:3);
  ReadLn
  END.
```

У процесі виведення матриці A на екран елементи її поточного рядка розділяються пропуском, а наступний рядок матриці за допомогою безпараметричної процедури WriteLn виводиться з нового рядка екрана.

Розв'язання задачі ґрунтується на тому, що в елементів, які розташовані вище головної діагоналі, індекс стовпця більше індексу рядка. Таким чином, елемент матриці a_{ij} включається в добуток, якщо $j > i$.

Можна здійснити більш раціональне розв'язання, якщо помітити, що у елементів, які лежать вище головної діагоналі, індекс стовпця j лежить у межах від $i + 1$ до k:

```
…
For i:=1 To k–1 Do
  For j:=i+1 To k Do
    P:=P*A[i,j];
…
```

Результати роботи програми мають вигляд:

Введіть порядок матриці k = 3
Введіть елементи матриці за рядками

```
A(1;1) = 2.5
A(1;2) = 1
A(1;3) = 2
A(2;1) = –3.6
A(2;2) = 1.2
A(2;3) = 3
A(3;1) = –0.5
A(3;2) = 12.4
A(3;3) = 8.6
```

```
Вхідний масив:
2.500    1.000    2.000
–3.600   1.200    3.000
–0.500   12.400   8.600
P = 6.000
```

Розглянемо ще один **приклад**. Елементи прямокутної матриці A вимірності $m \times n$ визначаються за формулою $a_{ij} = \dfrac{i + \sin(\pi / j)}{\ln(i + j) - j}$. Знайти найбільший елемент k-го стовпця ($k \le n$) і поміняти його місцями з першим елементом матриці.

Алгоритм розв'язання даної задачі наведено на рис. 4.4, а *Pascal*-програма, яка реалізує цей алгоритм, називається Example19.

Рекомендується самостійно розібратися з особливостями алгоритму і реалізації розв'язання даної задачі.

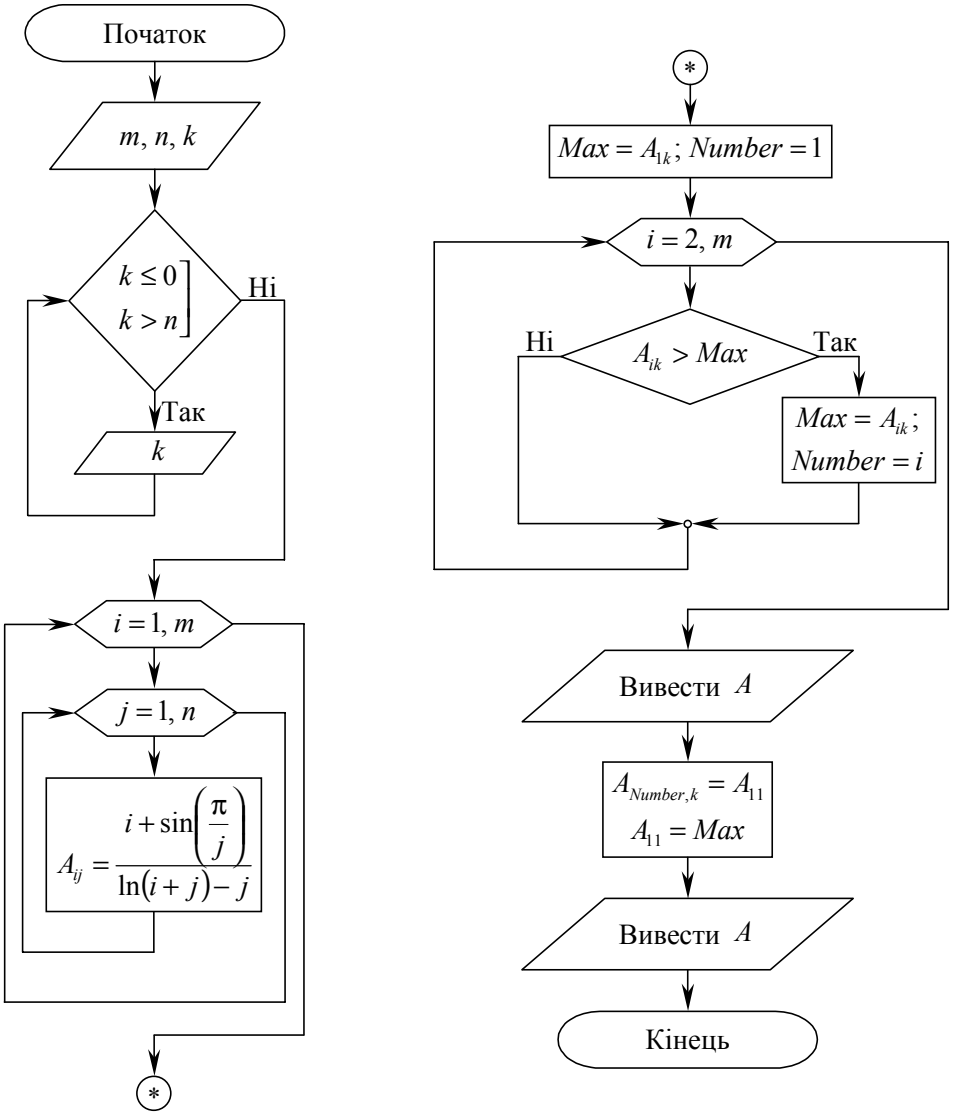

Початок
m, n, k
k ≤ 0
k > n
Ні
Так
k
i = 1, m
j = 1, n
$A_{ij} = \frac{i + \sin\left(\frac{\pi}{j}\right)}{\ln(i+j) - j}$
*
*
Max = A1k; Number = 1
i = 2, m
Ні
Aik > Max
Так
Max = Aik;
Number = i
Вивести A
ANumber,k = A11
A11 = Max
Вивести A
Кінець

Рис. 4.4

```
Program Example19;
Uses Crt;
Var A:Array[1..100,1..100] Of Real;
      i,j,m,n,k,Number:Integer;
      Max:Real;
BEGIN
  ClrScr;
  Write('Введіть кількість рядків матриці A       m= ');
  ReadLn(m);
  Write('Введіть кількість стовпців матриці A   n= ');
  ReadLn(n);
  Write('У якому стовпці шукати максимальний елемент? k= ');
  ReadLn(k);
  While (k>n) Or (k<=0) Do
  Begin
    WriteLn(k, '-го стовпця не існує.');
    Write('Знову введіть k= ');
    ReadLn(k)
  End;
  For i:=1 To m Do
    For j:=1 To n Do A[i,j]:=(i+Sin(Pi/j))/(Ln(i+j)–j);
  Max:=A[1,k]; Number:=1;
  For i:=2 To m Do
    If A[i,k]>Max Then
                    Begin
                      Max:=A[i,k];
                      Number:=i
                    End;
  WriteLn('Вхідна матриця A:');
  For i:=1 To m Do
  Begin
    For j:=1 To n Do Write(A[i,j]:8:3, ' ');
    WriteLn
  End;
  A[Number,k]:=A[1,1]; A[1,1]:=Max;
  WriteLn('Отримана матриця:');
  For i:=1 To m Do
  Begin
    For j:=1 To n Do Write(A[i,j]:8:3, ' ');
    WriteLn
  End;
  ReadLn
END.
```

Результати роботи програми Example19 мають наступний вигляд:

```
Введіть кількість рядків матриці A        m = 6
Введіть кількість стовпців матриці A      n = 6
У якому стовпці шукати максимальний елемент? k = 8
8-го стовпця не існує.
Знову введіть k = 4

Вхідна матриця A:
–3.259     –2.219     –1.156     –0.714     –0.495     –0.370
20.281     –4.888     –2.061     –1.226     –0.847     –0.638
7.766      –10.242    –3.200     –1.805     –1.228     –0.920
6.563      –24.011    –4.616     –2.451     –1.637     –1.217
6.315      –110.927   –6.372     –3.166     –2.072     –1.527
6.343      88.115     –8.553     –3.951     –2.532     –1.849

Отримана матриця:
–0.714     –2.219     –1.156     –3.259     –0.495     –0.370
20.281     –4.888     –2.061     –1.226     –0.847     –0.638
7.766      –10.242    –3.200     –1.805     –1.228     –0.920
6.563      –24.011    –4.616     –2.451     –1.637     –1.217
6.315      –110.927   –6.372     –3.166     –2.072     –1.527
6.343      88.115     –8.553     –3.951     –2.532     –1.849
```

4.2. Текстові файли

Уявімо собі ситуацію, коли для виконання деякої програми потрібно використовувати великий обсяг вхідних даних. Або іншу ситуацію, коли у ході виконання програми одержуються результати (їх теж може бути багато), які необхідні надалі, наприклад, для розрахунків по іншим програмам. Вводити з клавіатури велику кількість даних, так само як і записувати на аркуш паперу багато результатів, щонайменше заняття стомлююче. Зручним способом збереження інформації такого типу є запис її на носій. Різновидами цих носіїв є жорсткі диски ("вінчестери"), оптичні диски (*CD*, *DVD*, *Blu-ray Disc*), карти пам'яті та *USB*-флеш-накопичувачі ("флешки"). Для читання і запису інформації з носіїв використовуються такі пристрої, як оптичні приводи та зчитувачі флеш-карток різних типів. У мові *Pascal* для роботи з носіями інформації передбачені спеціальні об'єкти – *файли*.

При написанні програми фізична природа файлу не приймається до уваги. Прив'язка реальних файлів до програми здійснюється операційною системою. Таким чином, з погляду програмування, файл можна уявити у вигляді сукупності елементів однакового типу.

Хоча загальні визначення файлу і масиву однакові, необхідно розуміти принципові відмінності, які існують між цими складними типами даних:

1. кількість елементів у файлі заздалегідь не визначається.

2. елементи файлу не мають індексів, тому прочитати деякий елемент файлу можна тільки перебравши по черзі всі попередні елементи.

Важливе місце серед даних файлового типу займають *текстові файли*. На відміну від файлів інших типів, текстові файли є не просто послідовністю елементів одного простого типу, а складаються із символів, що об'єднуються в рядки. Зрозуміло, що ці рядки можуть бути різної довжини, отже, текстові файли можуть оброблятися тільки послідовно. Розділення на рядки в текстовому файлі здійснюється за допомогою спеціальної *ознаки кінця рядка*. З цією ознакою пов'язана стандартна функція логічного типу Eoln (див. підрозд. 2.2). У самому кінці будь-якого файлу, тобто наприкінці останнього рядка текстового файлу, записана *ознака кінця файлу*, з якою пов'язана функція Eof.

Оскільки будь-який файл з погляду програмування є абстрактною моделлю фізичного набору даних, то для простоти і наочності будемо уявляти структуру організації текстового файлу так, як це показано на рис. 4.5.

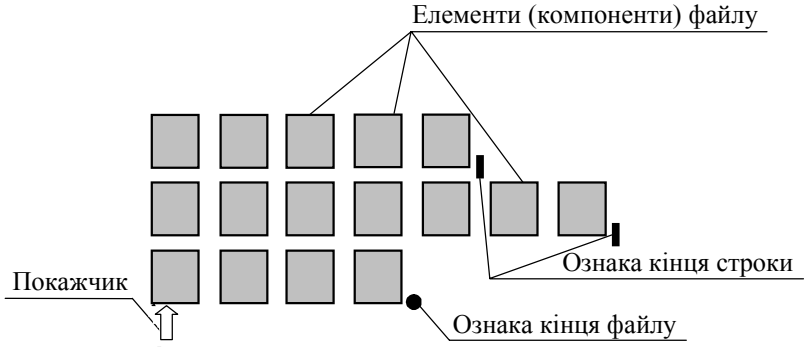


Рис. 4.5. Графічна інтерпретація текстового файлу

Операції над текстовими файлами можна здійснювати тільки за допомогою спеціальних функцій або процедур. Перед цим файл повинен бути зв'язаний з набором даних, які знаходяться на носії інформації.

Файл необхідно відкрити перед початком роботи з елементами, що зберігаються в ньому. Відзначимо, що *текстовий файл можна відкрити або тільки для запису, або тільки для читання*.

Перед закінченням виконання програми файл повинен бути закритий, оскільки завершення програми не спричиняє автоматичного закриття файлу, відкритого програмним способом.

Розглянемо основні процедури, необхідні для роботи з текстовими файлами.

Процедура Assign

Дана процедура зв'язує складний тип даних – файл із даними, які знаходяться на носії інформації у вигляді фізичного текстового файлу.

Загальний синтаксис процедури має вигляд:

Assign (<*Файлова_змінна*>, <*Рядковий_вираз*>);

де *Файлова_змінна* – ім'я файлової змінної типу Text;
Рядковий_вираз – рядковий вираз, який являє собою шлях до фізичного файлу.

Припустимо **наприклад**, що на флеш-накопичувачі, у каталозі *DAN* зберігається текстовий файл із даними, який має назву *id.dat* (*H:\DAN\id.dat*). Тоді наведений фрагмент *Pascal*-програми зв'язує файлову змінну F з існуючим зовнішнім файлом *id.dat*.

```
Var F:Text;
...
Assign(F, 'H:\DAN\id.dat');
```

Відзначимо, що процедура Assign не повинна використовуватися для відкритого файлу.

Процедура Reset

Дана процедура дозволяє відкрити текстовий файл для читання. Перед її виконанням файлова змінна повинна бути зв'язана з існуючим зовнішнім текстовим файлом за допомогою процедури Assign.

Загальний синтаксис процедури має вигляд:

Reset (<*Файлова_змінна*>);

Якщо при виконанні процедури Reset зовнішній файл *id.dat* відсутній у зазначеному місці (*H:\DAN*) або флешка не вставлена в *USB*-порт, то це призведе до помилки в програмі. Якщо файл F уже відкритий, то він закриється, а потім відкриється знову. При цьому поточна позиція покажчика (голівки читання або запису елементів файлу) переміститься на початок файлу.

```
Var F:Text;
...
Assign(F, 'H:\DAN\id.dat');
Reset(F);
```

Процедура Rewrite

Дана процедура дозволяє відкрити текстовий файл для запису. Перед її виконанням файлова змінна повинна бути зв'язана з існуючим зовнішнім текстовим файлом за допомогою процедури Assign.

Загальний синтаксис процедури має такий вигляд:

Rewrite (<*Файлова_змінна*>);

Якщо при виконанні процедури Rewrite зовнішній файл *id.dat* відсутній у директорії *DAN* на флеш-накопичувачі, то він буде створений автоматично. Якщо ж файл *id.dat* існує в зазначеному місці, то виконання даної процедури призведе до його знищення, а потім створення заново. В обох випадках буде

```
Var F:Text;
...
Assign(F, 'H:\DAN\id.dat');
Rewrite(F);
```

створено порожній файл (порожній набір даних), а поточна позиція покажчика встановиться на початок цього файлу.

Процедура Append.

Дана процедура дозволяє відкрити текстовий файл для розширення (доповнення). Перед її виконанням файлова змінна повинна бути зв'язана з існуючим зовнішнім текстовим файлом за допомогою процедури Assign.

Загальний синтаксис процедури має вигляд:

Append (<*Файлова_змінна*>);

```
Var F:Text;
...
Assign(F, 'H:\DAN\id.dat');
Append(F);
```

Якщо зовнішнього файлу *H:\DAN\id.dat* не існує, то виконання даної процедури призведе до помилки в програмі. Якщо файл *id.dat* уже відкрито, то він спочатку закриється, потім відкриється знову, а поточна позиція покажчика переміститься на кінець файлу.

Процедури читання з файлу і запису у файл

За аналогією обміну інформацією між стандартними файлами оперативної пам'яті ЕОМ і зовнішніми носіями інформації (див. підрозд. 2.5), процедури введення та виведення використовуються і для роботи з текстовими файлами. Процедури Read або ReadLn – для читання елементів з файлу, а процедури Write або WriteLn – для запису значень у текстовий файл.

Загальний синтаксис усіх зазначених процедур однаковий і має такий вигляд:

[**Read(Ln)** / або / **Write(Ln)**] (<*Файлова_змінна*>,

<*Список_введення_або_список_виведення*>);

При виконанні процедур читання необхідно, щоб зовнішній файл був відкритий за допомогою процедури Reset. Всі елементи в списку оператора Read або ReadLn повинні збігатися за типом із відповідними елементами текстового файлу. Оператор Read зчитує елементи і записує їх в оперативну пам'ять ЕОМ у порядку їхнього слідування в текстовому файлі (послідовно). Оператор ReadLn виконується так само, як і Read, але після зчитування всіх елементів, що знаходяться в списку цього оператора, здійснюється перехід до початку наступного рядка текстового файлу.

При виконанні процедур запису необхідно, щоб зовнішній файл був відкритий за допомогою процедури Rewrite або Append. Оператор Write послідовно записує в рядок файлу значення змінних зі свого списку. Оператор WriteLn виконується так само, як і оператор Write, але після запису

в рядок всіх елементів його списку, покажчик переміщується до початку нового рядка текстового файлу.

Процедура Close

Після виконання даної процедури текстовий файл закривається, при цьому зберігаються всі зміни, які були зроблені з файлом.

Загальний синтаксис процедури має вигляд:

Close (<*Файлова_змінна*>);

Після виконання наведеної послідовності операторів, в існуючому каталозі *H:\DAN* буде створено текстовий файл *id.dat*, у який запишеться рядкова константа ***'PASCAL'***. Потім файл *id.dat* буде закрито. Якщо перед виконанням процедури Close файл не був відкритий, то його стан не зміниться. Нагадаємо, що закінчення виконання програми не спричиняє автоматичного виклику процедури Close.

```
Var F:Text;
...
Assign(F, 'H:\DAN\id.dat');
Rewrite(F); Write(F, 'PASCAL');
Close(F);
```

Процедура Erase

Ця процедура дозволяє стерти текстовий файл з носія інформації. Перед її виконанням файлова змінна повинна бути зв'язана з існуючим зовнішнім текстовим файлом за допомогою процедури Assign.

Синтаксис процедури має наступний вигляд:

Erase (<*Файлова_змінна*>);

Якщо зовнішнього файлу *H:\DAN\id.dat* не існує, то виконання даної процедури призведе до помилки в програмі. Рекомендується, щоб перед виконанням процедури Erase файл F був закритий.

```
Var F:Text;
...
Assign(F, 'H:\DAN\id.dat');
Erase(F);
```

Процедура Rename

Ця процедура дозволяє змінити назву текстового файлу. Перед її виконанням файлова змінна повинна бути зв'язана з існуючим зовнішнім текстовим файлом за допомогою процедури Assign.

Синтаксис процедури має такий вигляд:

Rename (<*Файлова_змінна*>, <*Рядковий_вираз*>);

Якщо зовнішнього файлу *H:\DAN\id.dat* не існує, то виконання даної процедури призведе до помилки в програмі. Рядковий вираз в процедурі Rename визначає нове ім'я існуючого зовнішнього файлу, тому воно не може співпадати з іменами вже існуючих файлів.

```
Var F:Text;
...
Assign(F, 'H:\DAN\id.dat');
Rename(F, 'H:\DAN\id.bak');
```

Як **приклад** роботи з текстовими файлами розглянемо розв'язання наступної задачі. Нехай дано текстовий файл, у якому знаходиться фрагмент російського тексту. Потрібно знайти кількість букв російського алфавіту в кожному рядку даного тексту, а також відносну частоту, з якою символи "а" і "А" зустрічаються серед російських букв усього тексту. При розв'язанні задачі необхідно враховувати як малі, так і великі букви російського алфавіту.

Зрозуміло, що відносна частота зустрічі відповідних символів у тексті визначається за формулою S_a/S, де S_a – кількість символів "а" і "А" у файлі, а S – загальна кількість символів російського алфавіту у файлі.

Текст програми для розв'язання даної задачі та результати її роботи наведені нижче.

```
Program Example20;
Uses Crt;
Var Si,S,Sa,N:Integer;
    Symbol:Char; G:Text;
BEGIN
  ClrScr;
  Assign(G, 'H:\DOC\text.txt');
  Reset(G);
  N:=0; Sa:=0; S:=0;
  While Not Eof(G) Do
  Begin
    Si:=0; N:=N+1;
    While Not Eoln(G) Do
    Begin
      Read(G, Symbol);
      Case Symbol Of
        'А'..'Я', 'а'..'п', 'р'..'я': Si:=Si+1
      End;
      If (Symbol='А') Or (Symbol='а') Then Sa:=Sa+1
    End;
    WriteLn(N, '-й рядок містить ', Si, 'символів російського
алфавіту.');
    S:=S+Si; ReadLn(G)
  End;
  WriteLn('У файлі H:DOC\text.txt ', N, ' рядків і ', S, ' букв
російського алфавіту.');
  WriteLn('Символ "А" і "а" зустрічається у файлі ', Sa, ' разів.');
  WriteLn('Його відносна частота зустрічі Sa/S= ', Sa/S:4:4, '.');
  Close(G);
  ReadLn
END.
```

Для файлу *H:\DOC\text.txt* з вхідними даними, результати роботи даної програми мають наступний вигляд:

Файл H:\DOC\text.txt

Объект, который называется файлом, представляет собой абстрактную модель физического набора данных и существует вне программы. Абстрагирование от физической природы данных является существенной особенностью языка Pascal.
Оно дает возможность пользователю сосредоточить свое внимание на алгоритме решения задачи, не вдаваясь в детали организации файлов в той или иной операционной системе.

1-й рядок містить 45 символів російського алфавіту.
2-й рядок містить 51 символів російського алфавіту.
3-й рядок містить 52 символів російського алфавіту.
4-й рядок містить 37 символів російського алфавіту.
5-й рядок містить 55 символів російського алфавіту.
6-й рядок містить 52 символів російського алфавіту.
7-й рядок містить 36 символів російського алфавіту.
У файлі A:DOC\text.txt 7 рядків і 328 букв російського алфавіту.
Символ "A" і "a" зустрічається у файлі 29 разів.
Його відносна частота зустрічі Sa/S = 0.0884.

Розглянемо ще один **приклад**. Нехай існує текстовий файл *H:\id.dan*, у якому записані значення елементів двох одновимірних масивів *A* і *B* однакового порядку. Необхідно сформувати з масивів *A* і *B* двовимірний масив *D*, елементи якого визначаються за формулою $d_{ij} = (a_i + b_j)/2$. Отриманий масив *D* дописати у файл вхідних даних *H:\id.dan*.

Файл H:\id.dan

1 2 3 4 5
8 6 4 2 0

Зрозуміло, що для розв'язання даної задачі необхідно визначити вимірність *n* записаних у файл одновимірних масивів. Для цього можна прочитати елементи масиву *A* з першого рядка текстового файлу і знайти їх кількість *n*. Після цього, в іншому циклі з відомим числом повторень *n*, прочитати елементи масиву *B* з другого рядка файлу.

Порядок шуканого масиву *D* буде *n*×*n*, а його елементи d_{ij} визначаються в подвійному циклі при варіюванні індексів *i* та *j*. Наприклад:

$$d_{24} = \frac{a_2 + b_4}{2} = \frac{2+2}{2} = 2;\ \ d_{51} = \frac{a_5 + b_1}{2} = \frac{5+8}{2} = 6{,}5;$$ і т. п.

Текст програми для розв'язання даної задачі і результати її роботи наведені нижче:

```
Program Example21;
Uses Crt;
Type Vector=Array[1..50] Of Real;
Var A,B:Vector;
      D:Array[1..50] Of Vector;
      i,j,N:Integer;
      F:Text;
BEGIN
  Assign(F, 'H:\id.dan'); Reset(F);
  N:=0;
  While Not Eoln(F) Do
  Begin
    N:=N+1;
    Read(F, A[N])
  End;
  For i:=1 To N Do Read(F, B[i]);
  Append(F); WriteLn(F);
  WriteLn(F, 'Отриманий масив D');
  For i:=1 To N Do
  Begin
    For j:=1 To N Do
    Begin
      D[i,j]:=(A[i]+B[j])/2;
      Write(F, D[i,j]:3:3, ' ')
    End;
    WriteLn(F);
  End;
  Close(F);
  ClrScr; WriteLn('Результати див. у H:\id.dan');
  ReadLn
END.
```

Доповнений файл H:\id.dan

```
1 2 3 4 5
8 6 4 2 0
Отриманий масив D
4.500 3.500 2.500 1.500 0.500
5.000 4.000 3.000 2.000 1.000
5.500 4.500 3.500 2.500 1.500
6.000 5.000 4.000 3.000 2.000
6.500 5.500 4.500 3.500 2.500
```

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ ПО РОЗДІЛУ ПЕРШОМУ

1. Охарактеризувати основні етапи розв'язання задач на ЕОМ.
2. Що таке схема алгоритму? Які геометричні фігури використовуються при складанні схеми, що вони символізують?
3. Які типи даних існують в мові *Pascal*?
4. Назвіть порядок виконання операцій у виразі.
5. Назвіть вісім стандартних математичних функцій мови *Pascal*. Які особливості використання нестандартних функцій?
6. Яка структура програми на мові *Pascal*?
7. Як працює оператор присвоєння?
8. Яка різниця між математичною операцією рівності та операцією присвоєння?
9. Як працюють процедури введення і виведення?
10. Яким чином можна управляти формою виведення даних?
11. Дати характеристику розгалужених обчислювальних процесів.
12. Наведіть синтаксис оператора If.
13. Як працює оператор If?
14. Наведіть синтаксис оператора Case.
15. Як працює оператор Case?
16. Наведіть синтаксис оператора GoTo.
17. Як працює оператор GoTo?
18. Як виконується умовний оператор, якщо в ньому знаходиться інший умовний оператор?
19. Які особливості використання логічних операцій = та ≠ в мові *Pascal*.
20. Дати характеристику циклічних обчислювальних процесів.
21. Навести загальний вигляд алгоритму циклічної структури.
22. Наведіть синтаксис оператора While-Do.

23. Як працює оператор циклу While-Do?
24. Наведіть синтаксис оператора Repeat-Until.
25. Як працює оператор циклу Repeat-Until?
26. Наведіть синтаксис оператора For-Do.
27. Як працює оператор циклу For-Do?
28. Чим цикл Repeat-Until відрізняється від циклу While-Do?
29. Дані якого типу можна вживати як параметр циклу For-Do?
30. Охарактеризуйте роботу процедур Break та Continue для управління циклом з параметром.
31. Наведіть характеристику ітераційного циклу.
32. Охарактеризувати поняття складеного оператора. Навести приклади його застосування при реалізації розгалужених та циклічних алгоритмів.
33. Що таке складний тип даних? Чім він відрізняється від скалярного типу?
34. Який тип можуть мати елементи масиву?
35. Як елементи масиву розміщуються в пам'яті ЕОМ?
36. Як можуть бути описані одновимірні та двовимірні масиві?
37. Як здійснюється доступ до елементів масиву в мові *Pascal*?
38. Як здійснюється введення та виведення одновимірного масиву?
39. Як здійснюється введення та виведення двовимірного масиву?
40. Що таке файл, чим він відрізняється від масиву?
41. Наведіть графічну інтерпретацію файлу.
42. Які дані можуть бути елементами файлу?
43. Як зв'язати файлову змінну з іменем текстового файлу на носіях ЕОМ?
44. В чому різниця між операторами Reset та Rewrite?
45. Як додати новий елемент у кінець текстового файлу?
46. Чим відрізняються процедури Read і ReadLn при роботі з текстовим файлом?
47. Чим відрізняються процедури Write і WriteLn при роботі з текстовим файлом?
48. Як програмним чином можна стерти та перейменувати текстовий файл?
49. Наведіть графічну інтерпретацію заздалегідь визначеного обчислювального процесу.
50. Що таке підпрограма?
51. Охарактеризуйте опис та виклик підпрограми-процедури.
52. Охарактеризуйте опис та виклик підпрограми-функції.
53. Чим відрізняється підпрограма процедура від підпрограми функції?
54. Яка різниця між формальними і фактичними параметрами?
55. Що таке глобальна і локальна змінні?

СПИСОК ЛІТЕРАТУРИ ДО РОЗДІЛУ ПЕРШОГО

1. **Вирт, Н.** Систематическое программирование. Введение [Текст] / Н. Вирт. – М. : Мир, 1977.

2. **Дал, У.** Структурное программирование [Текст] : Пер. с англ. / У. Дал, Э. Дайкстра, К. Хоор. – М. : Мир, 1975.

3. **Йенсен, К.** Паскаль. Руководство для пользователя [Текст] / К. Йенсен, Н. Вирт. – М. : Финансы и статистика, 1989.

4. **Марченко, А. И.** Программирование в среде Turbo Pascal 7.0 [Текст] / А. И. Марченко, Л. А. Марченко. – К. : Юниор, 1997.

5. **Новичков, В. С.** Паскаль [Текст] / В. С. Новичков, Н. И. Парфилова, А. Н. Пылькин. – М. : Высшая школа, 1990.

6. Справочник по процедурам и функциям Borland Pascal With Object 7.0. [Текст] / И. Дериев [и др.]. – К. : Диалектика, 1993.

ЧАСТИНА ДРУГА

ОСНОВИ ЧИСЕЛЬНИХ МЕТОДІВ

Інженерна практика в наші дні досить часто зустрічається з математичними задачами, точний (аналітичний) розв'язок яких дістати досить складно або зовсім неможливо. У цих випадках звертаються до наближених обчислень за допомогою чисельних методів.

Із появою ЕОМ, а в останні роки – персональних комп'ютерів, обчислювальні можливості математики суттєво зросли: задачі, в яких механічні обчислення займали багато часу, зараз розв'язуються за лічені хвилини або години.

Таким чином, знання основ обчислювальної математики та вміле використання ЕОМ значною мірою визначають кваліфікацію сучасного інженера, тому програмування та обчислювальна математика займають важливе місце у його загальноосвітній підготовці, а відповідні курси читаються в усіх університетах України.

Існує два принципово різних напрямки у методиці зазначеної підготовки: програмування та обчислювальна математика читаються як два окремі курси, або як один курс. У НУК основи програмування та обчислювальна математика вивчаються в рамках курсу "Основи інформаційних технологій та програмування", який читається студентам усіх технічних спеціальностей.

Дана частина навчального посібника присвячена другому розділу зазначеного курсу, але й може розглядатися окремо, оскільки алгоритми реалізації викладених чисельних методів наведено у вигляді структурно-логічних блок-схем, отже, безвідносно до конкретної операційної системи та мови програмування.

За своїм змістом друга частина посібника поділена на шість глав, присвячених чисельним методам математичного аналізу. Розглядаються мето-

ди чисельного інтегрування, методи розв'язання нелінійних рівнянь, метод розв'язання систем лінійних алгебраїчних рівнянь, інтерполяція та апроксимація функцій, а також методи наближеного розв'язання звичайних диференціальних рівнянь першого порядку.

Кожне основне математичне твердження підкріплено розв'язанням відповідного прикладу. Крім того, всі наведені алгоритми розв'язання різних задач на ЕОМ за допомогою чисельних методів ілюструються конкретними чисельними результатами, отриманими після програмної реалізації цих алгоритмів.

1. ЧИСЕЛЬНЕ ЗНАХОДЖЕННЯ ЗНАЧЕННЯ ВИЗНАЧЕНОГО ІНТЕГРАЛУ. НАЙПРОСТІШІ КВАДРАТУРНІ ФОРМУЛИ ТА ЇХ РЕАЛІЗАЦІЯ НА ЕОМ

У багатьох наукових та технічних задачах інтегрування функції є складовою частиною вирішення певної проблеми. Обчислення площ та об'ємів, визначення статичних моментів і моментів інерції тіл, обчислення значення роботи, здійсненої деякими силами, та цілий ряд інших задач призводять до необхідності інтегрування функцій.

З курсу математичного аналізу відомо, якщо функція $f(x)$ неперервна на відрізку $[a, b]$, то визначений інтеграл від цієї функції існує та може бути обчислений за формулою Ньютона-Лейбніца:

$$\int_a^b f(x)\,dx = F(b) - F(a), \qquad (1.1)$$

де $F(x)$ – первісна функції $f(x)$, тобто $F'(x) = f(x)$.

Слід зазначити, що формула (1.1) не отримала широкого практичного застосування, оскільки для більшості елементарних функцій $f(x)$ первісну $F(x)$ не вдається виразити через елементарні функції. Крім того, часто підінтегральна функція задається у вигляді таблиці або графіка. Ці обставини призводять до необхідності наближеного обчислення значення визначеного інтегралу з використанням певного чисельного методу.

Чисельне знаходження значення визначеного інтегралу базується на його геометричній інтерпретації. Обчислення інтегралу $I = \int_a^b f(x)\,dx$ геометрично

зводиться до обчислення площі криволінійної трапеції *aABb*, обмеженої функцією $f(x)$, віссю абсцис та прямими $x = a$ та $x = b$ (рис. 1.1).

Формули наближеного інтегрування називаються *квадратурними формулами*. Розглянемо найпростіші з них.

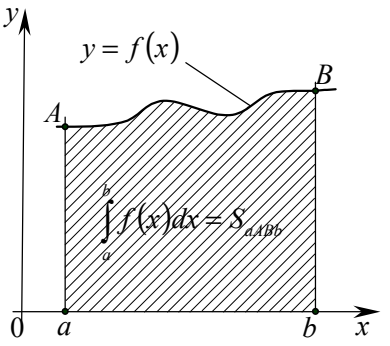


Рис. 1.1

1.1. Формули прямокутників

Розділимо відрізок $[a, b]$ на n рівних частин. При цьому отримаємо послідовність точок (вузлів інтегрування) $x_0 = a, x_1, x_2, \ldots, x_{n-1}, x_n = b$. Відстань між сусідніми вузлами називається кроком інтегрування $h = \dfrac{b-a}{n}$. Таким чином, $x_i = a + ih$, $i = 0, 1, 2, \ldots, n$.

Для обчислення інтегралу, площу криволінійної трапеції *aABb* наближено замінюємо сумою площ елементарних прямокутників з висотами, які дорівнюють значенням підінтегральної функції у лівих кінцях часткових відрізків $[x_i, x_{i+1}]$ (рис. 1.2):

$$I = S_{aABb} = S_1 + S_2 + S_3 + \ldots + S_{n-1} + \\ + S_n = hf(x_0 = a) + hf(x_1) + hf(x_2) + \\ + hf(x_3) + \ldots + hf(x_{n-2}) + hf(x_{n-1}).$$

Таким чином, остаточно маємо:

$$I = h\left[f(a) + \sum_{i=1}^{n-1} f(x_i) \right]. \qquad (1.2)$$

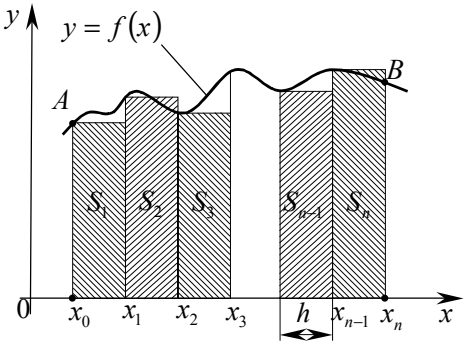


Рис. 1.2

Формула (1.2) має назву формули методу *лівих прямокутників*, а обчислення за нею легко можна реалізувати на ЕОМ згідно з алгоритмом, зображеним на рис. 1.3.

Аналогічно можна дістати формулу методу *правих прямокутників* для наближеного обчислення значення визначеного інтегралу (рис. 1.4):

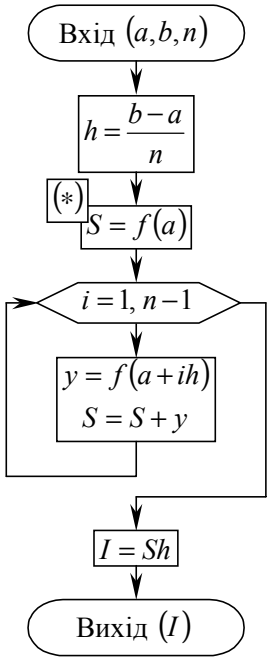


Рис. 1.3

$$I = S_{aABb} = \sum_{i=1}^{n} S_i = hf(x_1) + hf(x_2) + hf(x_3) + \\ + \ldots + hf(x_{n-1}) + hf(x_n = b).$$

$$I = h\left[f(b) + \sum_{i=1}^{n-1} f(x_i) \right]. \qquad (1.3)$$

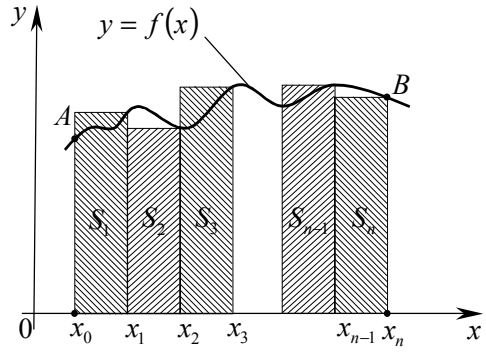


Рис. 1.4

Схема обчислення інтегралу за формулою (1.3) аналогічна наведеній на рис. 1.3 та відрізняється лише тим, що в обчислювальному блоці $(*)$ необхідно записати $S = f(b)$.

З рис. 1.2 та 1.4 можна помітити: якщо інтервал h достатньо малий, а функція $y = f(x)$ у межах цього інтервалу достатньо гладка (строго кажучи, вона повинна бути диференційована на цьому інтервалі, тобто $f(x) \in C[x_i, x_{i+1}]$), то похибка R заміни цієї реальної функції прямою $y = \text{const}$ буде наближатися до нуля за умови, що $h \to 0$.

Приклад 1.1. За допомогою формул методів лівих та правих прямокутників обчислити $\int_2^{10} \frac{x^2+1}{x}\,dx$, покладаючи $n = 4$.

Розв'язання. Знаючи межі інтегрування $a = 2$, $b = 10$, знаходимо крок $h = \frac{b-a}{n} = 2$. Тоді вузлами інтегрування будуть точки $x_0 = a = 2$, $x_1 = 4$,

$x_2 = 6$, $x_3 = 8$, $x_4 = b = 10$, а значення підінтегральної функції $f(x) = \dfrac{x^2+1}{x}$ у цих точках дорівнюють: $f(x_0) = 2{,}5$; $f(x_1) = 4{,}25$; $f(x_2) = 6{,}167$; $f(x_3) = 8{,}125$; $f(x_4) = 10{,}1$.

Тепер знаходимо чисельне значення інтегралу, використовуючи формулу лівих прямокутників (1.2):

$$I = h\left[f(a) + \sum_{i=1}^{3} f(x_i)\right] = h\left[f(a) + f(x_1) + f(x_2) + f(x_3)\right] =$$
$$= 2\left[2{,}5 + 4{,}25 + 6{,}167 + 8{,}125\right] = 42{,}084.$$

Якщо обчислення визначеного інтегралу виконувати за формулою методу правих прямокутників (1.3), то дістанемо:

$$I = h\left[f(b) + \sum_{i=1}^{3} f(x_i)\right] = h\left[f(b) + f(x_1) + f(x_2) + f(x_3)\right] =$$
$$= 2\left[10{,}1 + 4{,}25 + 6{,}167 + 8{,}125\right] = 57{,}284.$$

Даний інтеграл можна обчислити точно за формулою Ньютона–Лейбніца та визначити похибку наближеного інтегрування:

$$\int_2^{10} \frac{x^2+1}{x}\,dx = \int_2^{10} x\,dx + \int_2^{10} \frac{1}{x}\,dx = \frac{x^2}{2}\Bigg|_2^{10} + \ln|x|\Bigg|_2^{10} = 49{,}609;$$

$$\delta = \left|\frac{I_{\text{точн.}} - I_{\text{наближ.}}}{I_{\text{точн.}}}\right| \cdot 100\,\%; \quad \delta_1 = \left|\frac{49{,}609 - 42{,}084}{49{,}609}\right| \cdot 100\,\% = 15{,}169\,\%;$$

$$\delta_2 = \left|\frac{49{,}609 - 57{,}284}{49{,}609}\right| \cdot 100\,\% = 15{,}471\,\%.$$

1.2. Формула трапецій

Наближене значення визначеного інтегралу можна знайти більш точно, якщо площу фігури $aABb$ обчислити як суму площ елементарних трапецій. Для цього у межах всіх відрізків $[x_i, x_{i+1}]$ дугу графіка підінтегральної функції $y = f(x)$ необхідно замінити хордою, що стягує цю дугу (рис. 1.5):

$$I = S_{aABb} = \sum_{i=1}^{n} S_i = \frac{h}{2}\left[f(a) + f(x_1)\right] + \frac{h}{2}\left[f(x_1) + f(x_2)\right] +$$
$$+ \frac{h}{2}\left[f(x_2) + f(x_3)\right] + \ldots + \frac{h}{2}\left[f(x_{n-2}) + f(x_{n-1})\right] +$$
$$+ \frac{h}{2}\left[f(x_{n-1}) + f(b)\right] = \frac{h}{2}\left[f(a) + f(b) + 2f(x_1) + 2f(x_2) + \right.$$
$$\left. + 2f(x_3) + \ldots + 2f(x_{n-2}) + 2f(x_{n-1})\right].$$

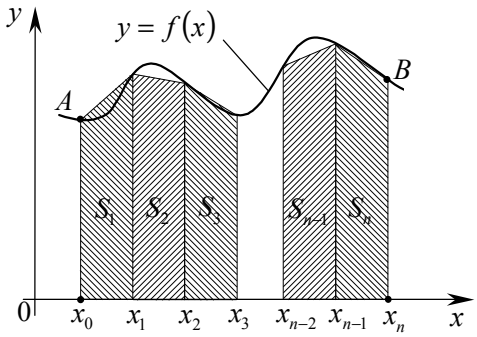


Рис. 1.5

Таким чином, остаточно маємо:

$$I = h\left[\frac{f(a) + f(b)}{2} + \sum_{i=1}^{n-1} f(x_i)\right]. \quad (1.4)$$

Формула (1.4) має назву формули методу *трапецій*. Наближені обчислення визначеного інтегралу за нею можуть бути реалізовані на ЕОМ згідно з алгоритмом, який наведено на рис. 1.3, де в блоці (∗) необхідно положити $S = (f(a) + f(b))/2$.

У рамках методу трапецій, аналогічно методам прямокутників, похибка заміни функції $y = f(x)$ прямою $y = ax + b$ буде наближатися до нуля, якщо $h \to 0$. Оцінка похибки R для одиничного кроку методу трапецій сформульована у вигляді наступної теореми, що представлена тут без доведення.

Нехай $f(x) \in C^2[x_i, x_{i+1}]$. Тоді похибка квадратурної формули трапеції має вигляд:

$$R(h, f) = -\frac{h^3}{12} f''(\xi),$$

де ξ – деяка точка з інтервалу $[x_i, x_{i+1}]$.

При поширенні цієї оцінки на весь відрізок інтегрування $[a, b]$, тобто на всі часткові відрізки $[x_i, x_{i+1}]$, кількість яких дорівнює n, отримаємо

$$R_n(h, f) = -\frac{h^3 n}{12} f''(\xi),\ x \in [a, b].$$

З огляду на те, що $hn = b - a$, дістанемо остаточний вигляд формули для оцінки похибки інтегрування в рамках даного методу:

$$|R_n(h, f)| \le M \frac{h^2|b-a|}{12}, \qquad (1.5)$$

де $M = \max\limits_{x\in[a, b]} |f''(x)|$.

Із оцінки (1.5) випливає, що квадратурна формула трапецій виявляється точною для поліномів першого степеня $f(x) = ax + b$, оскільки для них похідна другого порядку $f''(x)$ дорівнює нулю.

Приклад 1.2. За допомогою формули методу трапецій обчислити інтеграл прикладу 1.1, покладаючи $n = 4$. Оцінити точність отриманого значення.

Розв'язання. За допомогою формули (1.5) спочатку дамо оцінку похибки методу. Для цього обчислимо другу похідну підінтегральної функції: $f''(x) = \frac{2}{x^3}$. На відрізку інтегрування [2, 10] функція $f''(x)$ всюди додатна, причому її значення обмежене зверху: $f''(x) < 0{,}25$. Таким чином, маємо:

$$|R_n(h, f)| \le 0{,}25 \frac{2^2 \cdot 8}{12} = 0{,}667.$$

Значення підінтегральної функції у вузлових точках обчислені у прикладі 1.1 та представлені у вигляді таблиці:

i	0	1	2	3	4
x_i	2	4	6	8	10
$f(x_i)$	2,5	4,25	$6\frac{1}{6}$	8,125	10,1

Таким чином, маємо:

$$I = h\left[\frac{f(a)+f(b)}{2} + \sum_{i=1}^{3} f(x_i)\right] = 2\left[\frac{12{,}6}{2} + 4{,}25 + 6{,}167 + 8{,}125\right] = 49{,}684.$$

В кінцевому результаті маємо $I = 49{,}684 \pm 0{,}667$.

Слід звернути увагу на те, що формула (1.5) дає тільки верхню оцінку похибки, а так як функція $f''(x) = \frac{2}{x^3}$ швидко спадає, то модуль реальної похибки повинен бути значно меншим. Насправді: $\Delta = |49{,}609 - 49{,}684| = 0{,}075 < 0{,}667$.

1.3. Узагальнений алгоритм обчислення значення визначеного інтегралу методами лівих прямокутників, правих прямокутників та трапецій

Оскільки формули (1.2)–(1.4) однакові з точністю до значення першого доданка у дужках (для метода лівих прямокутників він дорівнює $f(a)$, для метода правих прямокутників – $f(b)$, а для метода трапецій – $\frac{f(a)+f(b)}{2}$), можна запропонувати узагальнений алгоритм чисельного знаходження визначеного інтегралу, якщо виділити алгоритм обчислення суми $\sum_{i=1}^{n-1} f(x_i)$, (він є загальним для всіх трьох методів) як самостійний. Назвемо зазначений алгоритм *Sum*, а його блок-схему зобразимо на рис. 1.6.

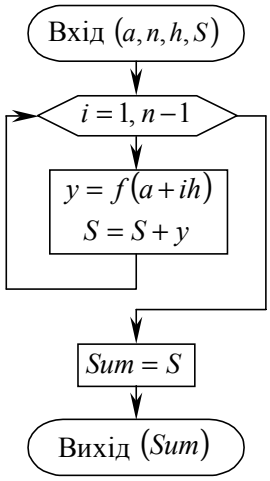


Рис. 1.6

Вибір того чи іншого чисельного метода повинен визначати сам користувач, наприклад так, як це показано на рис. 1.7.

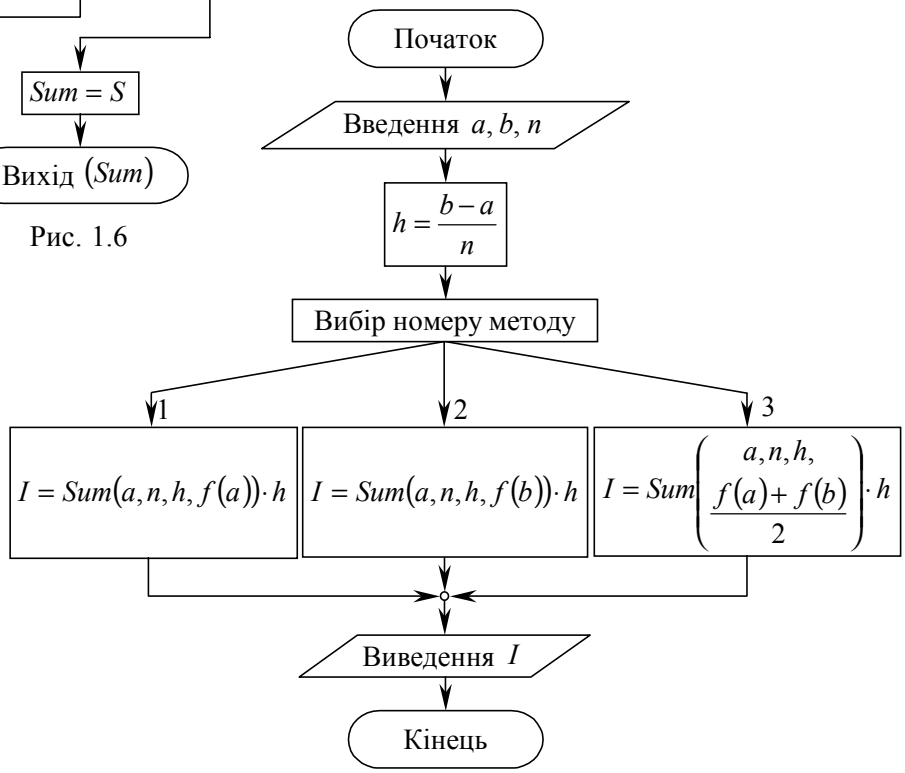


Рис. 1.7

Принцип побудови узагальненого алгоритму базується на наступному. Якщо на вхід алгоритму *Sum* замість формального параметра *S* подається фактичний параметр $f(a)$, то реалізується метод лівих прямокутників. Якщо замість *S* підставити $f(b)$, то визначений інтеграл обчислюється у рамках методу правих прямокутників. При підстановці як фактичного параметра значення $\frac{f(a)+f(b)}{2}$ інтеграл буде обчислений методом трапецій.

1.4 Формула Сімпсона

Точність наближеного інтегрування помітно зросте, якщо при обчисленні елементарних площ S_i, на які розбивається площа трапеції $aABb$, підінтегральну функцію $y = f(x)$ замінити параболою другого степеня вигляду $y = ax^2 + bx + c$, значення якої у вузлах x_i співпадають зі значеннями початкової підінтегральної функції, тобто $f(x_i) = ax_i^2 + bx_i + c$. Оскільки параболу можна провести, як мінімум, по трьох точках (рис. 1.8), то відрізок $[a, b]$ необхідно розбити на парне число частин $(2n)$.

Розглянемо довільну параболічну трапецію, зображену на рис. 1.9. Її площа

$$S = \int_{-h}^{h} (ax^2 + bx + c)\,dx = \frac{2}{3}ah^3 + 2ch.$$

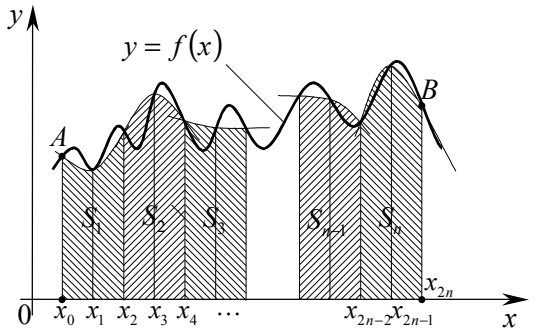


Рис. 1.8

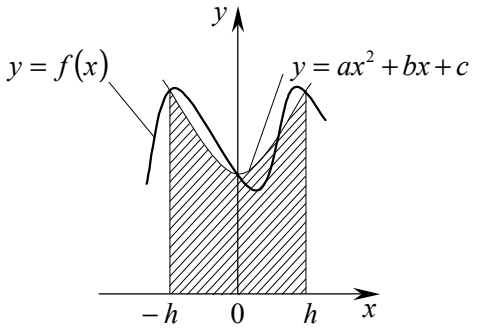


Рис. 1.9

Для знаходження невідомих коефіцієнтів *a* та *c*, які входять у формулу для обчислення *S*, скористаємося обставиною, що у вузлах $-h$, 0 та *h* значення параболи $y = ax^2 + bx + c$ та підінтегральної функції $y = f(x)$ співпадають:

$$\left.\begin{aligned} f(-h) &= ah^2 - bh + c \\ f(0) &= c \\ f(h) &= ah^2 + bh + c \end{aligned}\right\} \Rightarrow c = f(0);\ \ a = \frac{f(-h) - 2f(0) + f(h)}{2h^2}.$$

Таким чином, площа параболічної трапеції визначається формулою

$$S = \frac{h}{3}\left[f(-h) + 4f(0) + f(h)\right].$$

За допомогою отриманої формули можна обчислити всі елементарні площі S_i (див. рис. 1.8), а потім взяти їх суму для знаходження S_{aABb}:

$$I = S_{aABb} = \sum_{i=1}^{n} S_i =$$

$$= \frac{h}{3}\left(\begin{array}{c} f(a) + 4f(x_1) + f(x_2) + f(x_2) + 4f(x_3) + f(x_4) + f(x_4) + \\ + 4f(x_5) + f(x_6) + \ldots + f(x_{2n-4}) + 4f(x_{2n-3}) + \\ + f(x_{2n-2}) + f(x_{2n-2}) + 4f(x_{2n-1}) + f(b) \end{array}\right).$$

Остаточно отримаємо:

$$I = \frac{h}{3}\left(\begin{array}{c} f(a) + f(b) + 4[f(x_1) + f(x_3) + f(x_5) + \ldots + f(x_{2n-3}) + \\ + f(x_{2n-1})] + 2[f(x_2) + f(x_4) + f(x_6) + \ldots + \\ + f(x_{2n-4}) + f(x_{2n-2})] \end{array}\right)$$

або

$$I = \frac{h}{3}\left(f(a) + f(b) + 4\sum_{i=1,3,5\ldots}^{2n-1} f(x_i) + 2\sum_{i=2,4,6\ldots}^{2n-2} f(x_i)\right), \qquad (1.6)$$

де $h = \dfrac{b-a}{2n}$.

Формула (1.6) має назву формули *Сімпсона* або формули *парабол*. Алгоритм її практичної реалізації на ЕОМ наведений на блок-схемі, яка зображена на рис. 1.10.

Як видно, використання допоміжного параметра k дозволило нам обійтися одним циклом для підсумовування значень $f(x_i)$ з різними коефіцієнтами. Для кожної ітерації циклу з параметром i, значення $k = (-1)^{i+1}$, $i = 1, 2, 3, \ldots, 2n-1$. Отже, коефіцієнт $(3 + k)$ дорівнює 4 для $i = 1, 3, 5, \ldots, 2n-1$, а для $i = 2, 4, 6, \ldots, 2n-2$ цей коефіцієнт дорівнює 2.

Далі представимо без доведення теорему про оцінку похибки R для одиничного кроку методу Сімпсона.

Нехай $f(x) \in C^4[x_i, x_{i+2}]$. Тоді похибка квадратурної формули Сімпсона має вигляд:

$$R(h, f) = -\frac{h^5}{90} f^{(\mathrm{IV})}(\xi),$$

де ξ – деяка точка з інтервалу $[x_i, x_{i+2}]$.

Аналогічно тому, як це було зроблено в підрозд. 1.2, поширимо дану оцінку на всі часткові відрізки $[x_i, x_{i+2}]$ і внаслідок цього отримаємо формулу для оцінки похибки знаходження визначеного інтегралу:

$$\left|R_n(h, f)\right| \le M \frac{h^4 |b-a|}{180}, \qquad (1.7)$$

де $M = \max\limits_{x \in [a, b]} \left| f^{(\mathrm{IV})}(x) \right|$.

З (1.7) випливає, що визначений інтеграл, обчислений за квадратурною формулою Сімпсона, буде точними, якщо підінтегральна функція є поліномом не вище третього степеня.

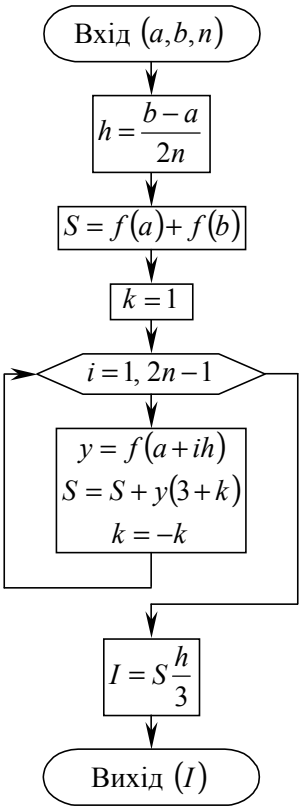


Рис. 1.10

Приклад 1.3. Обчислити методом Сімпсона інтеграл прикладу 1.1, покладаючи $n = 2$. Оцінити точність отриманого значення.

Розв'язання. Обчислимо $h = \frac{b-a}{2n} = 2$ та за допомогою формули (1.7) оцінимо похибку методу. На відрізку [2, 10] функція $f^{(\mathrm{IV})}(x)$ всюди додатна, а її значення обмежено зверху: $f^{(\mathrm{IV})}(x) < 0{,}75$. Таким чином, $\left|R_n(h, f)\right| \le 0{,}75 \frac{2^4 \cdot 8}{180} = \frac{8}{15}$.

Значення підінтегральної функції у вузлових точках обчислені в прикладі 1.1, отже за формулою (1.6) маємо:

$$I = \frac{h}{3}\left(f(a) + f(b) + 4[f(x_1) + f(x_3)] + 2f(x_2)\right) =$$
$$= \frac{2}{3}\left(2{,}5 + 10{,}1 + 4[4{,}25 + 8{,}125] + \frac{37}{3}\right) = 49{,}622.$$

$$I = 49{,}622 \pm \frac{8}{15}; \quad \Delta = \left|49{,}609 - 49{,}622\right| = 0{,}013 < \frac{8}{15}.$$

1.5. Обчислення інтегралу з заданим ступенем точності. Метод подвійного перерахунку

Іноді доводиться розв'язувати задачі чисельного обчислення значення визначеного інтегралу з заданим наперед ступенем точності ε. Для цього необхідно прогнозувати необхідну кількість n елементарних площ S_i, які складають загальну площу криволінійної трапеції S_{aABb}.

Нехай задана гранична припустима похибка чисельного інтегрування ε. Для того щоб виконувалася нерівність $|R_n(h, f)| \le \varepsilon$, достатньо поставити вимогу: $M\dfrac{h^2|b-a|}{12} \le \varepsilon$, враховуючи оцінку (1.5) для методу трапецій, або $M\dfrac{h^4|b-a|}{180} \le \varepsilon$, враховуючи (1.7) для методу Сімпсона. Отже, для методу трапецій

$$n \ge \sqrt{\frac{\max\limits_{x\in[a,\,b]}|f''(x)|\cdot|b-a|^3}{12\varepsilon}}, \tag{1.8}$$

а для методу Сімпсона

$$n \ge \sqrt[4]{\frac{\max\limits_{x\in[a,\,b]}|f^{(\mathrm{IV})}(x)|\cdot|b-a|^5}{2880\varepsilon}}. \tag{1.9}$$

Приклад 1.4. Методами трапецій та Сімпсона обчислити інтеграл $\int\limits_0^1 x^2 \sin x\,dx$ з заданим ступенем точності $\varepsilon = 10^{-2}$.

Розв'язання.

1. Згідно з методом трапецій, необхідне число елементарних площ n визначається за формулою (1.8):

$$\max_{x\in[a,\,b]}|f''(x)| = \max_{x\in[0,\,1]}\left|(2-x^2)\sin x + 4x\cos x\right| = 3{,}003;$$

$$n \ge \sqrt{\frac{3{,}003\cdot 1^3}{12\cdot 10^{-2}}} \Rightarrow n \ge 5{,}002.$$

Тепер можна визначити необхідний крок $h = \dfrac{1}{5{,}002} = 0{,}200.$

Вузли інтегрування і значення підінтегральної функції у вузлових точках обчислимо у вигляді наступної таблиці:

i	0	1	2	3	4	5
x_i	0	0,2	0,4	0,6	0,8	1
$f(x_i)$	0	0,008	0,062	0,203	0,459	0,841

По даним таблиці знаходимо чисельне значення інтегралу за формулою (1.4):

$$I = h\left[\frac{f(a)+f(b)}{2} + \sum_{i=1}^{4} f(x_i)\right] =$$

$$= 0{,}2\left[\frac{0{,}841}{2} + 0{,}008 + 0{,}062 + 0{,}203 + 0{,}459\right] = 0{,}231.$$

Остаточно дістанемо: $I = 0{,}23 \pm 0{,}01$.

2. Згідно з методом Сімпсона, число n визначається за формулою (1.9), а значення визначеного інтегралу – за формулою (1.6):

$$\max_{x\in[a,\,b]} \left|f^{(\mathrm{IV})}(x)\right| = \max_{x\in[0,\,1]} \left|(x^2-12)\sin x - 8x\cos x\right| = 13{,}579;$$

$$n \geq \sqrt[4]{\frac{13{,}579\cdot 1^5}{2880\cdot 10^{-2}}} \Rightarrow n \geq 0{,}829.$$

$h \leq \dfrac{1}{2\cdot 0{,}829} \leq 0{,}603$. Приймаємо $h = 0{,}5$.

i	0	1	2
x_i	0	0,5	1
$f(x_i)$	0	0,120	0,841

$$I = \frac{h}{3}\left[f(a) + 4f(x_1) + f(b)\right] = \frac{0{,}5}{3}\left[0 + 4\cdot 0{,}120 + 0{,}841\right] = 0{,}220.$$

Остаточно дістанемо: $I = 0{,}22 \pm 0{,}01$.

Невелике розходження у значенні I, яке обчислено у прикладі 1.4 різними методами, виникло через те, що в рамках методу трапецій не було потреби округляти крок h, а в рамках методу Сімпсона значення h довелось округлити так, щоб число кроків стало цілим. Таким чином, при реалізації чисельного методу інтегрування з заданим наперед ступенем точності на ЕОМ вибір

відповідного кроку являє собою окрему нескладну задачу. Крім того, формули (1.8) та (1.9) дозволяють прогнозувати похибку відповідного методу інтегрування тільки у тих випадках, коли підінтегральну функцію задано аналітично.

Виходячи з цих обставин, на практиці доцільним буде застосовувати метод *подвійного перерахунку*, згідно з яким шуканий інтеграл обчислюють двічі: при поділі відрізка інтегрування на n частин (I_n) та на $2n$ частин (I_{2n}).

Для оцінки похибки методу трапецій можна використовувати просту формулу. Нехай $R_n^n(h, f)$ та $R_n^{2n}(h, f)$ – похибки інтегрування за формулою трапецій відповідно при n та $2n$ відрізках поділу. Враховуючи оцінку (1.5), можна скласти рівність $\dfrac{R_n^n}{R_n^{2n}} = \left(\dfrac{h_n}{h_{2n}}\right)^2$, де h_n та h_{2n} – кроки інтегрування в першому та другому випадках. Зрозуміло, що $(h_n/h_{2n})^2 = 4$, тоді $R_n^n = 4R_n^{2n}$. Якщо I – істинне значення визначеного інтегралу, то $I = I_n + R_n^n$ та $I = I_{2n} + R_n^{2n}$, звідки випливає, що $\left|I_n - I_{2n}\right| = 3\left|R_n^{2n}\right|$. Таким чином, якщо задана гранична припустима похибка інтегрування ε, то достатньо, щоб $\left|R_n^{2n}\right| \le \varepsilon$, отже,

$$\left|I_n - I_{2n}\right| \le 3\varepsilon. \tag{1.10}$$

Аналогічні міркування дають можливість оцінити похибку методу Сімпсона згідно з методом подвійного перерахунку за формулою

$$\left|I_n - I_{2n}\right| \le 15\varepsilon. \tag{1.11}$$

На рис. 1.11 показана принципова схема алгоритму чисельного інтегрування з заданим ступенем точності. Після реалізації на ЕОМ вказаної схеми для методів трапецій та Сімпсона були отримані наступні результати обчислення інтегралу $\int_0^1 x^2 \sin x\,dx$ з точністю $\varepsilon = 10^{-5}$:

Метод трапецій:		Метод Сімпсона:	
$n = 2$	$I = 0{,}27030$;	$n = 2$	$I = 0{,}22306$;
$n = 4$	$I = 0{,}23487$;	$n = 4$	$I = 0{,}22323$;
$n = 8$	$I = 0{,}22614$;	$n = 8$	$I = 0{,}22324$.
$n = 16$	$I = 0{,}22397$;		
$n = 32$	$I = 0{,}22343$;		
$n = 64$	$I = 0{,}22329$;		
$n = 128$	$I = 0{,}22326$;		
$n = 256$	$I = 0{,}22325$.		

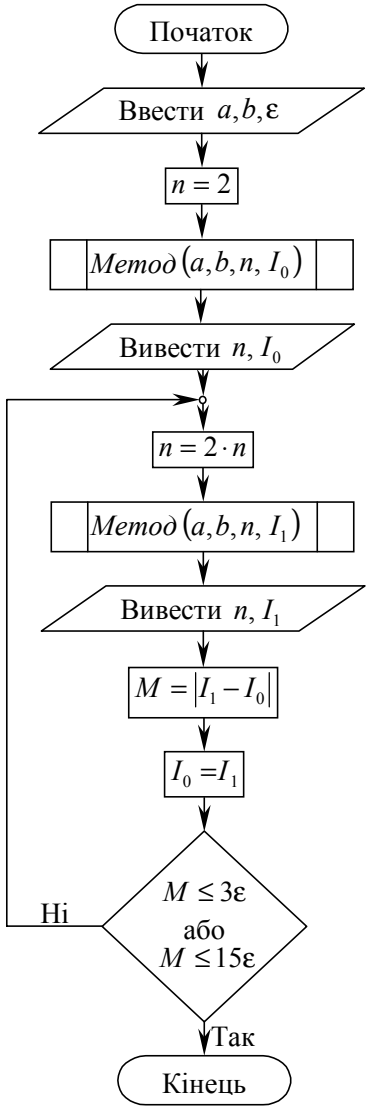


Рис 1.11

Бачимо, що зі збільшенням n похибка інтегрування зменшується (особливо швидко за методом Сімпсона). Але це твердження має суто теоретичне значення, так як у процесі практичних обчислень методом подвійного перерахунку, починає сильно прогресувати питома вага похибки округлення, значення якої з деякого моменту стає порівнянним з точністю результату інтегрування.

2. ЧИСЕЛЬНІ МЕТОДИ РОЗВ'ЯЗАННЯ НЕЛІНІЙНИХ РІВНЯНЬ ІЗ ОДНІЄЮ ЗМІННОЮ ТА ЇХ РЕАЛІЗАЦІЯ НА ЕОМ

2.1. Лінійні алгебраїчні, нелінійні алгебраїчні та трансцендентні рівняння. Необхідність наближеного розв'язання таких рівнянь

Відомо, що існують *лінійні* та *нелінійні* функції однієї змінної.

Найважливішою властивістю лінійних функцій $y = F(x)$ є те, що вони задовольняють принципу *суперпозиції*. Цей принцип полягає в наступному: якщо аргумент x являє собою лінійну суперпозицію точок $x = c_1x_1 + c_2x_2 + c_3x_3 + \ldots + c_nx_n$ (де c_i – константи), то значення y також може бути зображене у вигляді лінійної суперпозиції значень $F(x_1), F(x_2), F(x_3), \ldots, F(x_n)$. Наприклад, подана лінійна функція $y = 2x$. Якщо $x = 2x + 0{,}5x + 0{,}7x = 3{,}2x$, то лінійна суперпозиція дозволяє записати рівність $y = 4x + x + 1{,}4x = 6{,}4x$.

Нелінійні функції $y = F(x)$ не задовольняють принципу суперпозиції, наприклад, для $y = x^2 - (c_1x + c_2x)^2 \neq c_1x^2 + c_2x^2$.

Лінійне або нелінійне рівняння з однією змінною в загальному вигляді можна записати так:

$$F(x) = 0, \tag{2.1}$$

де функція $F(x)$ визначена і неперервна на скінченному або нескінченному інтервалі (A, B).

Таким чином, рівняння (2.1) називається *лінійним*, якщо $F(x)$ – лінійна функція. Якщо $F(x)$ – нелінійна функція, то рівняння (2.1) є *нелінійним*.

Коренем або *розв'язком* рівняння (2.1) називається будь-яке число $x_0 \in (A, B)$, що перетворює функцію $F(x)$ на нуль, тобто $F(x) = 0$.

Число x_0 називається *коренем k-ї кратності*, якщо при $x = x_0$ разом із функцією $F(x)$ перетворюються на нуль її похідні до $(k-1)$-го порядку включно:

$$F(x_0) = F'(x_0) = F''(x_0) = ... = F^{(k-1)}(x_0) = 0.$$

Однократний корінь називається *простим*.

Два рівняння $F(x) = 0$ і $G(x) = 0$ називаються *рівносильними* (*еквівалентними*), якщо будь-який розв'язок кожного з них є розв'язком і для іншого, тобто множини розв'язків цих рівнянь співпадають.

Розглянемо загальний вигляд лінійного рівняння з однією змінною: $kx + b = 0$. Воно має єдиний розв'язок $x = -b/k$.

Для нелінійних рівнянь можливі наступні варіанти розв'язків:

1. єдиний розв'язок, наприклад $5x^2 = 0 \Rightarrow x = 0$;

2. нескінченна множина розв'язків, наприклад $\sin x - 1 = 0 \Rightarrow x = 0{,}5\pi + 2\pi k$, де k належить множині цілих чисел ($k \in Z$);

3. немає розв'язків, наприклад $\sin x - 1{,}2 = 0$;

4. декілька $n \geq 2$ розв'язків, наприклад $x^2 - 4 = 0 \Rightarrow x_1 = 2$, $x_2 = -2$.

Нелінійні рівняння з одним невідомим підрозділяються на *алгебраїчні* та *трансцендентні*.

Нелінійне рівняння (2.1) називається алгебраїчним, якщо функція $F(x)$ є алгебраїчною. Шляхом алгебраїчних перетворень із будь-якого алгебраїчного рівняння можна дістати рівняння в канонічній формі:

$$F(x) = P_n(x) = a_0x^n + a_1x^{n-1} + a_2x^{n-2} + ... + a_n = 0,$$

де $a_0, a_1, a_2, \ldots, a_n$ – коефіцієнти рівняння, а x – змінна.

Показник n називається *степенем* алгебраїчного рівняння.

Відзначимо, що алгебраїчна функція $F(x)$ може бути:

1. *раціональною*:

а) *ціло-раціональною*, наприклад $F(x) = \dfrac{2x^2}{3} + \dfrac{x-1}{2}$;

б) *дрібно-раціональною*, наприклад $F(x) = \dfrac{x-2}{x^3+1} - \dfrac{1}{x} + 1$;

2. *ірраціональною*, наприклад $F(x) = \dfrac{3x^4 - 2x - \sqrt[5]{x+2}}{x^3 + \sqrt{x+1}}$.

Відомо, що будь-яке нелінійне алгебраїчне рівняння має, принаймні, один корінь – дійсний або комплексний. При цьому виявляється, що дістати точні значення коренів можна лише в окремих випадках, зазвичай коли для цього є деяка проста формула.

Так, для обчислення коренів квадратного рівняння вигляду $ax^2 + bx + c = 0$ застосовується відома формула $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$.

Для розв'язання кубічного рівняння вигляду $x^3 + px + q = 0$ застосовується формула $x = \sqrt[3]{-\frac{q}{2} + \sqrt{\frac{q^2}{4} + \frac{p^3}{27}}} + \sqrt[3]{-\frac{q}{2} - \sqrt{\frac{q^2}{4} + \frac{p^3}{27}}}$. Відразу відмітимо, що практичне застосування цієї формули ускладнено необхідністю використання апарата комплексних чисел.

Для розв'язання рівняння четвертого степеня також існує формула, яка виражає значення коренів через коефіцієнти цього рівняння. Однак вона є настільки складною, що практично не застосовується.

Слід зазначити, що в 1824 р. норвезький математик Абель довів, що рівняння п'ятого степеня $x^5 + ax^4 + bx^3 + cx^2 + dx + e = 0$ не можна розв'язати алгебраїчно. Іншими словами, не можна виразити корені цього рівняння через величини a, b, c, d, e за допомогою шести алгебраїчних дій (додавання, віднімання, множення, ділення, піднесення до степеня, добування кореня). У 1830 р. французький математик Галуа довів, що ніяке загальне рівняння, степінь якого більше за 4, не можна розв'язати алгебраїчно.

Виходячи з вищесказаного, можна зробити висновок, що більшість нелінійних алгебраїчних рівнянь точно розв'язати неможливо, внаслідок цього для обчислення коренів таких рівнянь доводиться застосовувати наближені (чисельні) методи.

Рівняння (2.1) називається *трансцендентним*, якщо функція $F(x)$ не алгебраїчна. Прикладами трансцендентних рівнянь є $x - 10\sin x = 0$; $2^x - 2\cos x = 0$; $\lg(x + 5) = x^2$ і т. п.

У деяких випадках розв'язання трансцендентних рівнянь можна звести до розв'язання алгебраїчних рівнянь. Але все-таки переважна більшість нелінійних трансцендентних рівнянь із однією змінною не розв'язуються за допомогою алгебраїчних перетворень (точними методами) і на практиці їх розв'язують чисельними методами.

У даному розділі представлені чисельні методи розв'язання нелінійних рівнянь із однією змінною, що є одними з найважливіших серед методів прикладного аналізу, необхідність у використанні яких виникає у багатьох розділах фізики, механіки, техніки та інших областях.

Розв'язати нелінійне рівняння чисельно – це означає *встановити, чи має воно корені, скільки їх, а також дістати значення коренів із заданим*

ступенем точності. Задача чисельного визначення дійсних коренів рівняння (2.1) зазвичай складається з двох етапів:

1. *Відокремлення коренів* (або *визначення відрізків ізоляції коренів*), тобто знаходження достатньо малих відрізків $[a, b]$ розглядуваної області (A, B), у межах яких знаходиться одне значення кореня.

2. *Уточнення коренів*, тобто їх обчислення із заданим ступенем точності у межах відрізків $[a, b]$ відповідним чисельним методом.

2.2. Відокремлення коренів нелінійного рівняння. Реалізація процесу відокремлення на ЕОМ

Перший етап чисельного розв'язання рівняння (2.1) полягає у *відокремлені коренів*, тобто у встановленні відрізків, які містять тільки один корінь. Відокремлення коренів у багатьох випадках можна виконати графічно. Оскільки дійсні корені рівняння (2.1) – це точки перетинання графіка функції $F(x)$ з віссю абсцис, достатньо побудувати графік $F(x)$ і позначити на цій осі відрізки, що містять по одному кореню. Побудову графіків у деяких випадках можна значно спростити, якщо замінити рівняння (2.1) на рівносильне: $F_1(x) = F_2(x)$. У цьому випадку будуються графіки функцій $y = F_1(x)$ та $y = F_2(x)$, а потім на осі Ox відмічаються відрізки, що локалізують абсциси точок перетинання цих графіків.

Приклад 2.1. Здійснити *графічне відокремлення коренів* рівняння $\sin 2x - \ln x = 0$ на відрізку $[0; 3]$.

Розв'язання. Побудуємо окремо графіки функцій $y = \sin 2x$ і $y = \ln x$ (рис. 2.1). Абсцисою точки A (точки їх перетинання) є корінь x_0 розглянутого рівняння, отже, $x_0 \in [1; 0{,}5\pi]$.

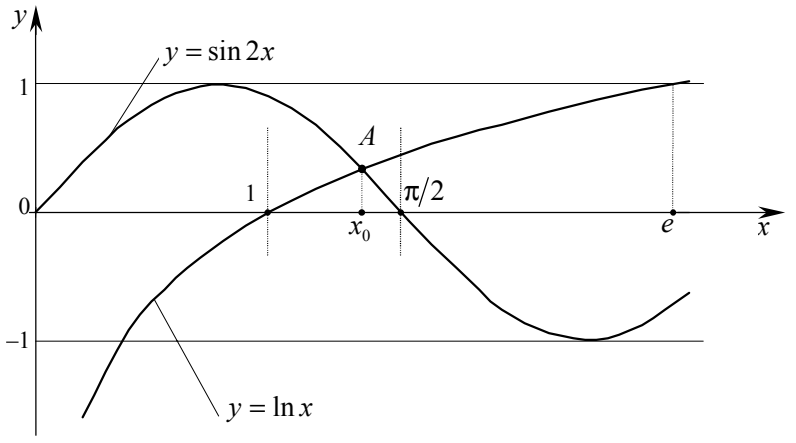


Рис. 2.1

У випадку, коли результати графічного відокремлення коренів викликають сумнів або графічне відокремлення здійснити важко, застосовується так зване *обчислювальне відокремлення кореня*.

При обчислювальному відокремленні кореня використовуються наступні математичні твердження:

1. Якщо неперервна на відрізку $[a, b]$ функція $F(x)$ приймає на його кінцях значення різних знаків (тобто $F(a) \cdot F(b) < 0$), то рівняння (2.1) має на цьому відрізку щонайменш один корінь.

2. Якщо функція $F(x)$ до того ж ще строго монотонна (тобто $F'(x)$ у межах $[a, b]$ зберігає свій знак), то корінь на відрізку $[a, b]$ буде єдиним.

Справді, значення функції $F(x) = \sin 2x - \ln x$ із прикладу 2.1 на відрізку $[1{,}2;\ 1{,}6]$: $F(1{,}2) > 0$; $F(1{,}6) < 0$. Крім цього, $F'(x \in [1{,}2;\ 1{,}6]) = 2\cos 2x - \dfrac{1}{x} < 0$.

При відокремленні коренів можна ефективно використовувати ЕОМ. Для цього необхідно протабулювати $F(x)$ на $[A, B]$ з деяким кроком h і з урахуванням твердження 1, вказати всі відрізки $[a, b] \subset [A, B]$, які містять по одному кореню. На рис. 2.2 наведена принципова схема алгоритму описаного табулювання, у якому функція $F(x)$ повинна використовуватися з урахуванням її області визначення D.

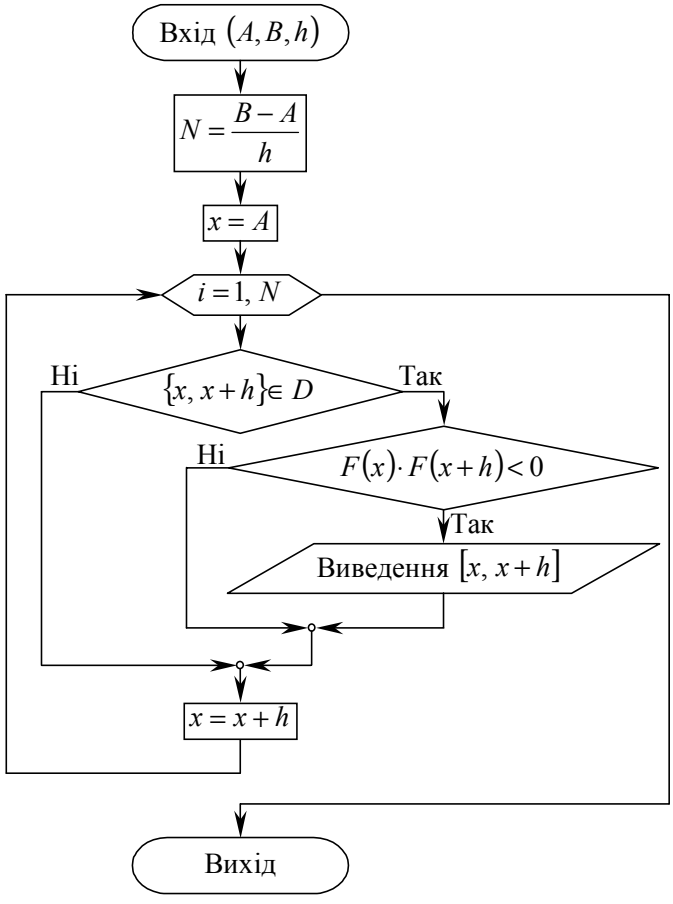


Рис. 2.2

Очевидно, що надійність такого підходу до відокремлення коренів залежить від характеру функції $F(x)$ і величини кроку h (чим менше крок, тим вище ймовірність того, що неперервна функція $F(x)$ на малому інтервалі ізоляції кореня $[a, b]$ буде монотонна).

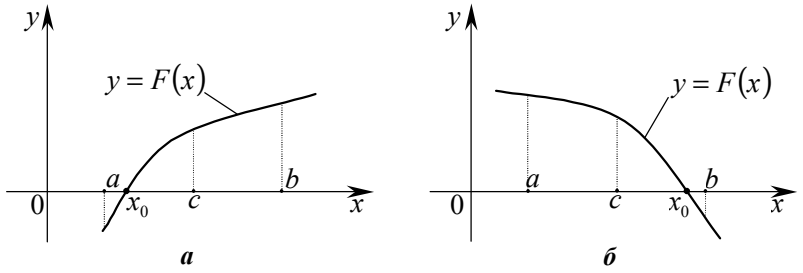
Після реалізації даної схеми на ЕОМ для функції $F(x) = x^3 - \cos \pi x$ при $A = -1000$, $B = 1000$, $h = 0{,}2$ були отримані наступні результати:

$$x_0^1 \in [-1{,}2; -1]; \quad x_0^2 \in [-0{,}6; -0{,}4]; \quad x_0^3 \in [0{,}4; 0{,}6].$$

2.3. Метод половинного ділення і його реалізація на ЕОМ

Даний метод відомий у літературі ще й під назвами *метод дихотомії, різновид методу проб* і *метод бісекції*.

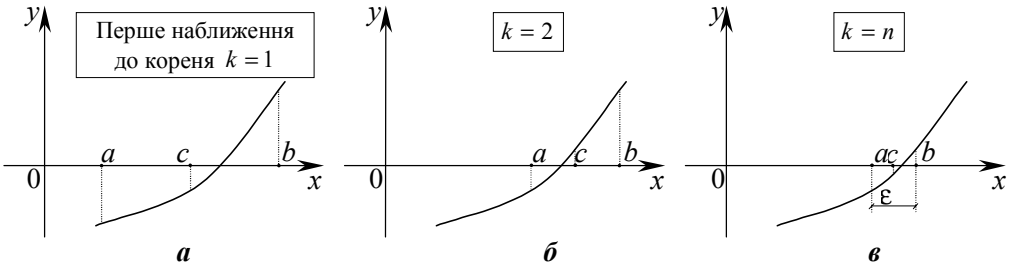
Отже, нехай нелінійне рівняння (2.1) має на відрізку ізоляції кореня $[a, b]$ єдиний корінь, причому функція $F(x)$ на цьому відрізку неперервна. Розділимо відрізок $[a, b]$ навпіл точкою $c = (a + b)/2$. Якщо $F(c) \neq 0$ (що практично ймовірніше), то можливі два випадки: або $F(x)$ змінює знак на відрізку $[a, c]$ (рис. 2.3,***а***), або на відрізку $[c, b]$ (рис. 2.3,***б***). Обираючи в кожному випадку той із відрізків, на якому функція змінює знак, і продовжуючи процес половинного ділення далі, можна дійти до якзавгодно малого відрізка, який містить корінь рівняння.



а ***б***

Рис. 2.3

Для реалізації описаного алгоритму локалізації кореня методом половинного ділення зручно (і доцільно) користуватися тільки початковими позначеннями (a, b, c). Графічна інтерпретація алгоритму з використанням початкових позначень проілюстрована на рис. 2.4.



а ***б*** ***в***

Рис. 2.4

З рис. 2.4 видно, що метод половинного ділення можна використовувати як метод розв'язання нелінійного рівняння із заданим ступенем точності ε.

Дійсно, якщо на деякому етапі процесу половинного ділення отриманий відрізок $[a, b]$ довжини ε, що містить корінь (див. рис. 2.4,*в*), то, прийнявши наближено $x_0 = (a + b)/2 = c$, дістанемо похибку, яка не перевищує значення ε/2. Отже, процес половинного ділення для обчислення кореня з точністю ε необхідно продовжувати до того часу, поки не виконається умова $(b - a)/2 < \varepsilon$.

Приклад 2.2. Методом половинного ділення уточнити корінь рівняння $\sin 2x - \ln x = 0$ на відокремленому відрізку $[1; 1{,}6]$ з точністю $\varepsilon = 10^{-2}$.

Розв'язання. Ручні обчислення у рамках методу половинного ділення зручно організувати у вигляді наступної таблиці:

k	a	b	c	$\varepsilon = (b-a)/2$	$F(a)$	$F(c)$	$F(a)\cdot F(c)$
1	1,000	1,600	1,300		0,909	0,253	$> 0 \Rightarrow a = c$
2	1,300	1,600	1,450	0,150	0,253	–0,132	$< 0 \Rightarrow b = c$
3	1,300	1,450	1,375	0,075	0,253	0,063	$> 0 \Rightarrow a = c$
4	1,375	1,450	1,413	0,038	0,063	–0,035	$< 0 \Rightarrow b = c$
5	1,375	1,413	1,394	0,019	0,063	0,014	$> 0 \Rightarrow a = c$
6	1,394	1,413	**1,403**	0,009 < 0,01			

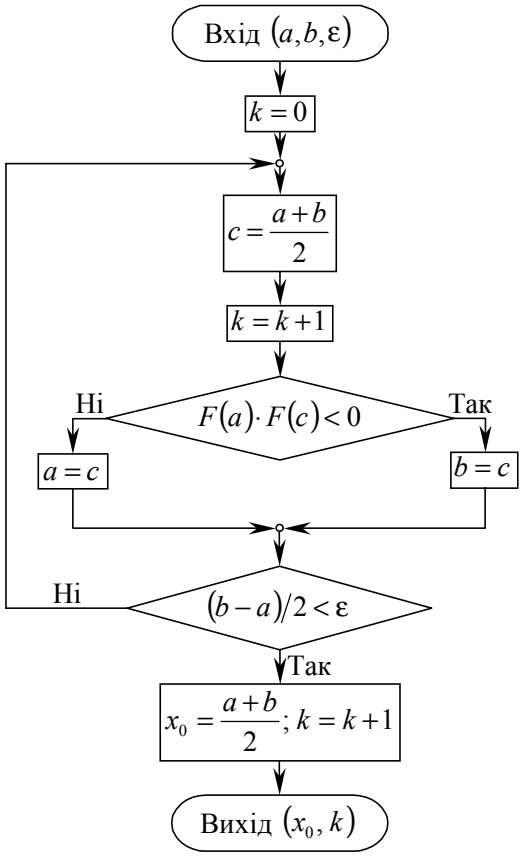


Рис. 2.5

Таким чином, $x_0 = 1{,}40 \pm 0{,}01$, а $|F(1{,}40)| = 0{,}00$.

Приклад 2.2 показує, що для обчислення кореня нелінійного рівняння навіть із невеликою точністю, у рамках методу половинного ділення доводиться виконувати досить багато рутинних обчислень. Отже, алгоритм даного методу доцільно реалізовувати на ЕОМ.

На рис. 2.5 зображена блок-схема алгоритму уточнення одного кореня рівняння (2.1) до заданого ступеня точності ε методом половинного ділення. Зі схеми видно: якщо навіть на певному наближенні з номером k станеться $F(c) = 0$, це не призведе до збою алгоритму.

Після реалізації даного алгоритму на ЕОМ результати розв'язання задачі прикладу 2.2 при $\varepsilon = 10^{-6}$ отримані у вигляді:

Корінь рівняння: $x_0 = 1{,}399429$.
Кількість наближень: $k = 20$.
$F(x_0) = -0{,}000001$.

2.4. Метод простої ітерації і його реалізація на ЕОМ

Нехай дано нелінійне рівняння (2.1). Потрібно визначити дійсний корінь даного рівняння на відомому відрізку його ізоляції $[a, b]$.

Замінимо рівняння $F(x) = 0$ на рівносильне[1]

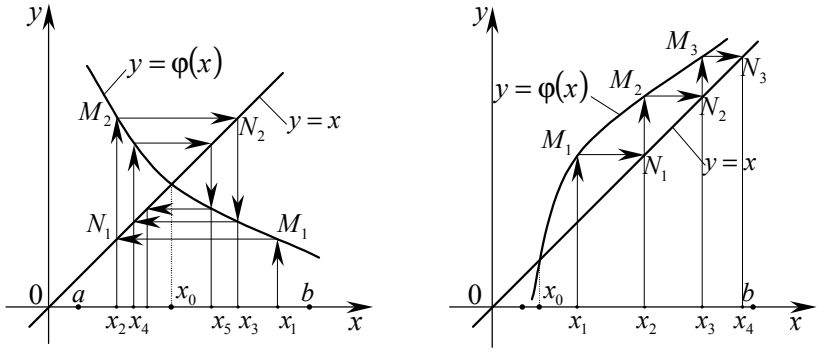
$$x = \varphi(x). \tag{2.2}$$

Перше наближення до істинного значення x_0 обирається із відрізка ізоляції кореня довільним образом, тобто $x_1 \in [a, b]$ Для знаходження другого наближення підставимо значення x_1 у праву частину рівняння (2.2): $x_2 = \varphi(x_1)$. Третє і наступне наближення виконуються аналогічно: $x_3 = \varphi(x_2)$; $x_4 = \varphi(x_3)$; …; $x_k = \varphi(x_{k-1})$.

Таким чином, *метод простої ітерації* (або *метод послідовних наближень*) здійснюється за рекурсивною формулою:

$$\left.\begin{array}{c} x_1 \in [a, b];\ x_k = \varphi(x_{k-1}), \\ k = 2, 3, 4, \ldots, n. \end{array}\right\}. \tag{2.3}$$

Процес послідовних наближень має просту геометричну інтерпретацію (див. рис. 2.6). Довільно оберемо значення $x_1 \in [a, b]$, тоді $x_2 = \varphi(x_1)$. Для того, щоб відкласти значення x_2 на осі абсцис, проведемо паралельно їй лінію M_1N_1. Ордината точки N_1 дорівнює ординаті точки M_1, а в силу того, що N_1 лежить на бісектрисі координатного кута $y = x$, ордината N_1 дорівнює абсцисі N_1. Таким чином, $\varphi(x_1) = M_{1y}$, $M_{1y} = N_{1y} = N_{1x} = x_2 \Rightarrow \varphi(x_1) = x_2$. Якщо провести $M_2N_2 \parallel Ox$ і повторити аналогічні міркування, то можна переконатися, що на рис. 2.6 зображені $x_3 = \varphi(x_2)$, $x_4 = \varphi(x_3)$, …



а *б*

Рис. 2.6

[1] Відмітимо, що заміна (2.1) ⇒ (2.2) здійсненна завжди. Наприклад, рівняння вигляду (2.1) $2x - \sin x = 0$ легко перетворюється у рівняння вигляду (2.2): $x = 0{,}5 \sin x$. Рівняння $2^x - \cos x = 0$ перетворюється на (2.2), якщо зобразити $0 = x - x$. Тоді $x = x - 2^x + \cos x$ або $x = x + 2^x - \cos x$.

З рис. 2.6 видно, що послідовні обчислення за схемою (2.3) у деяких випадках дозволяють дістати уточнене значення кореня (рис. 2.6,***а***), а у деяких випадках, навпаки, кожне нове значення x_k віддаляється від істинного значення x_0 (рис. 2.6,***б***). Говорячи більш строго, при розв'язанні нелінійного рівняння методом ітерацій можливі два випадки:

1. Ітераційна послідовність $x_1, x_2, x_3, \ldots, x_k$ *збігається*, тобто має границю, яка буде коренем рівняння ($\lim\limits_{k\to\infty} x_k = x_0$). У цьому випадку метод простої ітерацій є збіжним (див. рис. 2.6,***а***).

2. Метод ітерацій є *розбіжним*, коли послідовність $x_1, x_2, x_3, \ldots, x_k$ *розбігається*, тобто не має границі (див. рис. 2.6,***б***).

Існування $\lim\limits_{k\to\infty} x_k = x_0$ означає, що істинне значення кореня x_0 може бути отримано шляхом реалізації нескінченного ітераційного процесу. Однак на практиці ітерації не можуть продовжуватися нескінченно, що призводить до необхідності на певному етапі зупиняти процес обчислень, припускаючи при цьому деяку похибку методу ε.

Умова, за якої ітераційний процес збігається, і критерій оцінки точності методу простої ітерації виражає наступна теорема, наведена без доведення.

Нехай на відрізку ізоляції кореня $[a, b]$ існує єдиний корінь рівняння $x = \varphi(x)$, а функція $\varphi(x)$ визначена і диференційована на цьому відрізку, причому всі її значення $\varphi(x) \in [a, b]$. Тоді метод простої ітерації збігається при довільному виборі початкового наближення x_1, якщо виконується умова

$$\left|\varphi'(x)\right| \le q < 1. \tag{2.4}$$

Якщо дійсне число q визначається зі співвідношення (2.4), то критерієм закінчення ітераційного процесу уточнення кореня із заданим ступенем точності ε є умова

$$\left.\begin{array}{c} \left|x_k - x_{k-1}\right| \le \dfrac{\varepsilon(1-q)}{q}, \\ k = 2, 3, 4, \ldots, n. \end{array}\right\}. \tag{2.5}$$

Відзначимо, що умова (2.4) даної теореми не є необхідною. Це означає, що метод ітерацій може виявитися збіжним і при невиконанні даної умови.

Для практичної оцінки точності, значення q, яке входить в умову (2.5), обирається максимальним за модулем серед усіх значень похідної $\varphi'(x)$ на відрізку ізоляції кореня, тобто $q = \max\limits_{[a, b]}\left|\varphi'(x)\right|$. Чим менше виявляється значення q, тим скоріше збігається метод ітерацій.

Приклад 2.3. Методом простої ітерації уточнити корінь рівняння $\sin 2x - \ln x = 0$ на відокремленому відрізку [1; 1,6] з точністю $\varepsilon = 10^{-2}$.

Розв'язання. Перетворимо дане рівняння до ітераційного вигляду $x = x - \sin 2x + \ln x$ і перевіримо умову збіжності ітераційного процесу (2.4). Маємо: $|\varphi'(x)| = |1 - 2\cos 2x + 1/x|$. За допомогою калькулятора легко перевірити, що $q = |\varphi'(1{,}6)| = 3{,}622 > 1$, отже, метод ітерацій для даного рівняння буде розбіжним.

Якщо на відрізку ізоляції кореня $F(x)$ змінюється монотонно, за допомогою нескладних перетворень завжди можна домогтися того, щоб для рівняння $x = \varphi(x)$ метод ітерацій збігався. Наведемо один із можливих способів таких перетворень.

Помножимо рівняння (2.1) на деяке дійсне число $p \neq 0$ і перетворимо отримане рівняння до ітераційного вигляду: $pF(x) = 0 \Rightarrow x = x - pF(x)$. Задовольняючи це рівняння умові (2.4), дістанемо: $|1 - pF'(x)| < 1$. Таким чином, для забезпечення збіжності методу послідовних наближень можна вимагати, щоб виконувалася умова $0 < \max\limits_{[a,\,b]}\{pF'(x)\} \le 1$.

Для нашого рівняння $F'(x) = 2\cos 2x - 1/x$.

Протабулюємо отриману функцію на відрізку ізоляції кореня:

1	1,1	1,2	1,3	1,4	1,5	1,6
–1,832	–2,086	–2,308	–2,483	–2,599	–2,647	–2,622

Отже, якщо обрати значення $p = -1/3$, умова для забезпечення збіжності $0 < \max\limits_{[a,\,b]}\{pF'(x)\} \le 1$ буде виконана.

Будемо здійснювати схему ітерацій (2.3), враховуючи (2.5) для рівняння $x = x + 1/3(\sin 2x - \ln x)$, помітивши, що $q = \max\limits_{[a,\,b]}|\varphi'(x)| = 1 - |p| \cdot \min\limits_{[a,\,b]}|F'(x)| =$

$= 1 - \dfrac{1{,}832}{3} = 0{,}389$. Тоді $\dfrac{\varepsilon(1-q)}{q} = \dfrac{10^{-3}(1-0{,}389)}{0{,}389} = 0{,}016$.

Як перше наближення до кореня обираємо, наприклад, $x_1 = 1$ і ведемо розрахунок у таблиці.

k	x_k	x_{k+1}	$\lvert x_{k+1} - x_k \rvert$
1	1,000	1,303	0,303
2	1,303	1,385	0,082
3	1,385	1,397	0,012 < 0,016
4	**1,397**		

Таким чином, на четвертій ітерації дістанемо:

$x_0 = 1{,}40 \pm 0{,}01$, а $|F(1{,}40)| = 0{,}00$.

Алгоритм реалізації методу простої ітерації у випадку, коли він збігається до істинного значення кореня x_0 із заданою точністю ε, наведений на схемі, зображеної на рис. 2.7. На вхід даного алгоритму як параметр необхідно подавати значення $q = \max\limits_{[a,\,b]} |\varphi'(x)|$ тільки в тому випадку, якщо $q < 1$. Алгоритм обчислення q наведений на рис. 2.8.

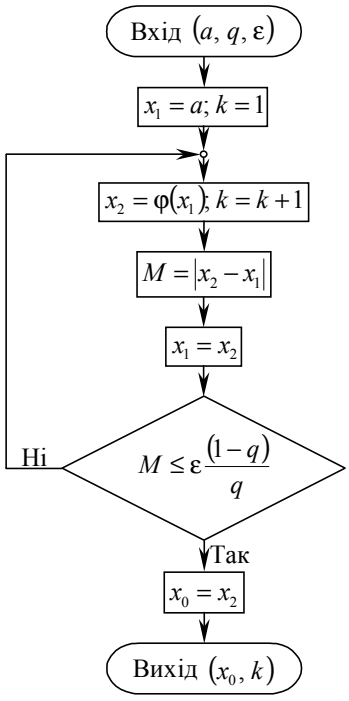


Рис. 2.7

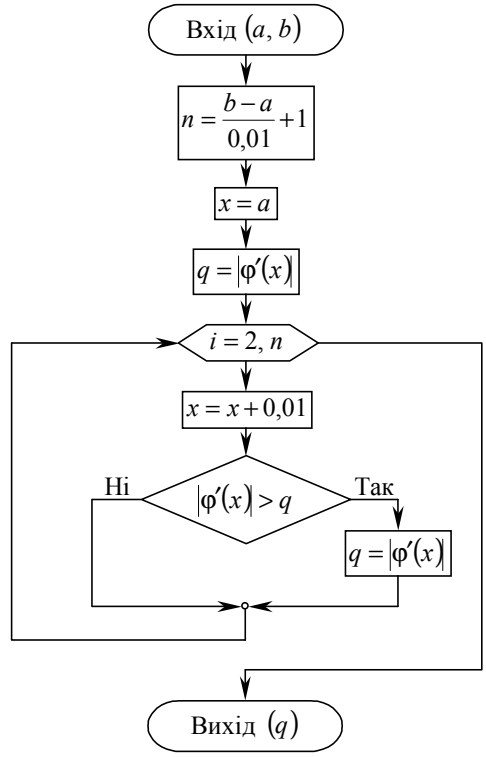


Рис. 2.8

Після реалізації даних алгоритмів на ЕОМ, розв'язок задачі приклада 2.3 при $\varepsilon = 10^{-6}$ отриманий у вигляді:

Корінь рівняння: x_0 = 1,399429. Кількість наближень: k = 9. $F(x_0)$ = = 0,000000.

2.5. Метод хорд і метод дотичних

Перш ніж приступити до розгляду даних методів, нагадаємо деякі корисні математичні твердження.

Функція $y = F(x)$ називається *зростаючою*, якщо зі зростанням аргументу x значення функції y збільшується. Якщо зі зростанням аргументу значення функції зменшується, то вона є *спадною*.

Функція називається *монотонною*, якщо в межах заданого відрізка вона або тільки зростає, або тільки спадає.

Якщо перша похідна $F'(x)$ у межах деякого відрізка додатна і зберігає свій знак, то функція $F(x)$ на цьому відрізку монотонно зростає. Якщо у межах деякого відрізка $F'(x)$ від'ємна і зберігає свій знак, то $F(x)$ на цьому відрізку монотонно спадає.

Якщо друга похідна $F''(x)$ у межах деякого відрізка додатна і зберігає свій знак, то графік функцій $F(x)$, побудований на цьому відрізку, є опуклим униз. Якщо у межах деякого відрізка $F''(x)$ від'ємна і зберігає постійний знак, то графік $F(x)$ буде опуклим уверх.

Як вже зазначалось в підрозд. 2.2, для існування на відрізку $[a, b]$ одного єдиного значення кореня необхідно і достатньо, щоб $F(a) \cdot F(b) < 0$ і функція $F(x)$ у межах $[a, b]$ була монотонна.

Якщо додатково поставити вимогу, щоб функція $F''(x)$ також була монотонною у межах $[a, b]$, то можливі наступні чотири випадки поведінки $F(x)$ на відрізку ізоляції кореня (див. рис. 2.9).

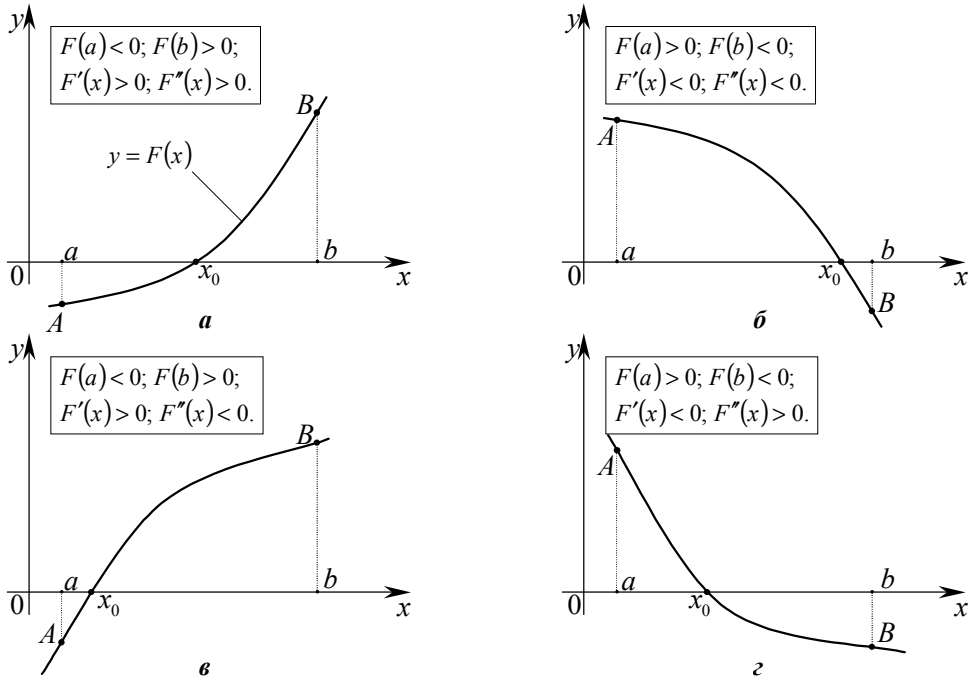


Рис. 2.9

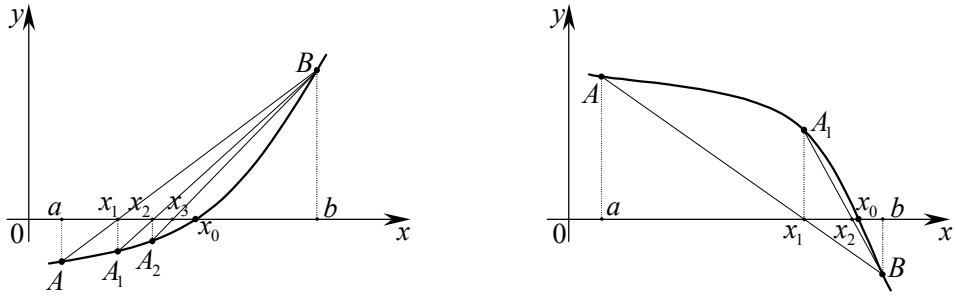
Метод хорд і його реалізація на ЕОМ

Даний метод іноді зустрічається в літературі під назвами *метод помилкового припущення* і *метод пропорційних частин*.

Нехай дано нелінійне рівняння (2.1), де $F(x)$ – неперервна на відрізку ізоляції кореня $[a, b]$ функція, яка має на ньому неперервні та знакопостійні похідні першого і другого порядків.

Ідея *методу хорд* полягає в тому, що на відрізку $[a, b]$ дуга кривої $y = F(x)$ стягується хордою AB і як наближене значення кореня береться точка перетинання AB з віссю Ox.

Розглянемо окремо випадки, зображені на рис. 2.9,***а***,***б***. Видно, що перша і друга похідна від функції $F(x)$ мають однакові знаки, тобто $F'(x) \cdot F''(x) > 0$:



З аналітичної геометрії відомо, що рівняння прямої, яка проходить через дві точки $A(x_1, y_1)$ і $B(x_2, y_2)$, має вигляд $\dfrac{(x - x_1)}{(x_2 - x_1)} = \dfrac{(y - y_1)}{(y_2 - y_1)}$. У нашому випадку для хорди AB маємо $A(a, F(a))$ і $B(b, F(b))$, отже

$$\frac{x - a}{b - a} = \frac{y - F(a)}{F(b) - F(a)}.$$

Для знаходження x_1 (точки перетинання хорди з віссю Ox) підставимо в це рівняння значення $y = 0$ і дістанемо

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

Знайдена формула має назву *формули методу хорд*. Тепер корінь рівняння лежить у межах відрізка $[x_1, b]$. Якщо точність обчисленого значення x_1 недостатня, то необхідно виконати наступне наближення до кореня, застосовуючи метод хорд до відрізка $[x_1, b]$:

$$x_2 = x_1 - \frac{F(x_1)(b - x_1)}{F(b) - F(x_1)}.$$

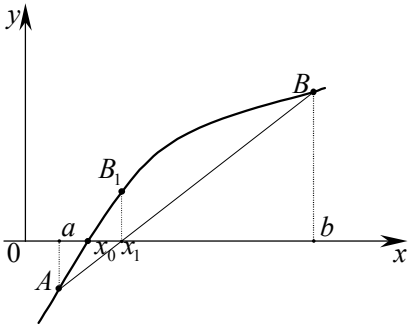
Продовжуючи процес наближень далі, знаходимо:

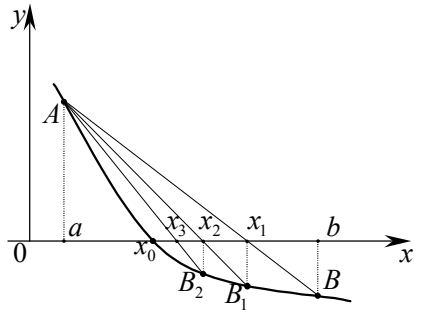
$$x_3 = x_2 - \frac{F(x_2)(b-x_2)}{F(b)-F(x_2)},\quad x_4 = x_3 - \frac{F(x_3)(b-x_3)}{F(b)-F(x_3)}, \ldots,$$
$$x_k = x_{k-1} - \frac{F(x_{k-1})(b-x_{k-1})}{F(b)-F(x_{k-1})}.$$

Таким чином, коли $F'(x) \cdot F''(x) > 0$, обчислення в рамках методу хорд здійснюється за рекурсивною формулою:

$$\left.\begin{array}{c} x_1 = a - \dfrac{F(a)(b-a)}{F(b)-F(a)};\ x_k = x_{k-1} - \dfrac{F(x_{k-1})(b-x_{k-1})}{F(b)-F(x_{k-1})}, \\ k = 2, 3, 4, \ldots, n. \end{array}\right\} \qquad (2.6)$$

Тепер розглянемо випадки, зображені на рис. 2.9,***в***,***г***, коли $F'(x) \cdot F''(x) < 0$:





За аналогією з графічною інтерпретацією методу половинного ділення, зображеною на рис. 2.4, можна сказати наступне: при уточненні кореня методом хорд (коли $F'(x) \cdot F''(x) < 0$) рухомою точкою є точка b відрізка ізоляції кореня, а точка a залишається нерухомою. Виходячи з цього, складемо рівняння хорди AB з точки B у точку A:

$$\frac{x-b}{a-b} = \frac{y-F(b)}{F(a)-F(b)}.$$

Підставляючи в це рівняння $y = 0$, отримаємо x_1 як точку перетину хорди з віссю Ox:

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

Тепер шуканий корінь x_0 належить відрізку $[a, x_1]$, а для подальшого уточнення кореня знову застосуємо метод хорд на цьому відрізку:

$$x_2 = x_1 - \frac{F(x_1)(x_1 - a)}{F(x_1) - F(a)}.$$

Продовжуючи процес наближень далі, знаходимо:

$$x_3 = x_2 - \frac{F(x_2)(x_2 - a)}{F(x_2) - F(a)}, \quad x_4 = x_3 - \frac{F(x_3)(x_3 - a)}{F(x_3) - F(a)}, \ \ldots,$$
$$x_k = x_{k-1} - \frac{F(x_{k-1})(x_{k-1} - a)}{F(x_{k-1}) - F(a)}.$$

Таким чином, коли $F'(x) \cdot F''(x) < 0$, обчислення в рамках методу хорд здійснюється за наступною схемою:

$$\left.\begin{array}{c} x_1 = b - \dfrac{F(b)(b-a)}{F(b) - F(a)}; \; x_k = x_{k-1} - \dfrac{F(x_{k-1})(x_{k-1} - a)}{F(x_{k-1}) - F(a)}, \\ k = 2, 3, 4, \ldots, n. \end{array}\right\} \qquad (2.7)$$

Процес обчислень за формулами (2.6) або (2.7) продовжується доти, поки ми не дістанемо наближений корінь із заданою точністю ε. Без доведення відзначимо, що для оцінки похибки методу хорд можна користуватися формулою

$$\frac{|F(x_k)|}{\min\limits_{[a,\,b]}|F'(x)|} \le \varepsilon. \qquad (2.8)$$

Необхідно відмітити, що вибір тої чи іншої рекурсивної формули для уточнення кореня в рамках методу хорд можна здійснити без обчислення $F'(x)$ користуючись очевидним правилом: *якщо на деякому кінці відрізка* $[a, b]$ *знак функції* $F(x)$ *не співпадає зі знаком другої похідної* $F''(x)$ *то цей кінець є рухомим*. Іншими словами, якщо $F(a) \cdot F''(x) < 0$, то рухомим є кінець a і обчислення необхідно здійснювати за формулою (2.6). У протилежному випадку, якщо $F(a) \cdot F''(x) > 0$, рухомим кінцем є b і обчислення здійснюються за формулою (2.7).

Приклад 2.4. Методом хорд уточнити корінь рівняння $\sin 2x - \ln x = 0$ на відокремленому відрізку $[1; 1{,}6]$ з точністю $\varepsilon = 10^{-2}$.

Розв'язання. Визначимо знак другої похідної і встановимо, за якою формулою необхідно проводити обчислення:

$$F''(a = 1) = -2{,}673 \Rightarrow F''(x) < 0. \quad F(a = 1) = 0{,}909 \Rightarrow F(a) > 0.$$

Таким чином, $F(a) \cdot F''(x) < 0 \Rightarrow$ рухомим є кінець $a \Rightarrow$ обчислення здійснюємо за схемою (2.6) доти, поки $\dfrac{|F(x_k)|}{\min\limits_{[a,\,b]}|F'(x)|} \le \varepsilon$.

Протабулюємо функцію $F'(x)$ на відрізку [1; 1,6] (див. приклад 2.3) і дістанемо min = 1,832. Умову закінчення процесу уточнення кореня для практичних обчислень зручно записати у вигляді $|F(x_k)| \le \varepsilon \cdot \min \le 0{,}018$.

$$x_1 = a - \frac{F(a)(b-a)}{F(b)-F(a)} = 1 - \frac{F(1)\cdot 0{,}6}{F(1{,}6)-F(1)} = 1{,}379;\ \ F(1{,}379) = 0{,}052 > 0{,}018.$$

$$x_2 = x_1 - \frac{F(x_1)(b-x_1)}{F(b)-F(x_1)} = 1 - \frac{F(1{,}379)\cdot 0{,}221}{F(1{,}6)-F(1{,}379)} = 1{,}399;\ \ F(1{,}399) = 0{,}002 < 0{,}018.$$

Таким чином, друге наближення до кореня дозволяє записати остаточну відповідь:

$$x_0 = 1{,}40 \pm 0{,}01, \quad \text{а} \quad |F(1{,}40)| = 0{,}00.$$

Алгоритм реалізації методу хорд для розв'язання рівняння (2.1) із заданою точністю ε наведений на схемі, зображеної на рис. 2.10. На вхід даного алгоритму як параметр необхідно подавати значення $\min = \min\limits_{[a,\,b]}|F'(x)|$, яке попередньо обчислюється за схемою, наведеною на рис. 2.11.

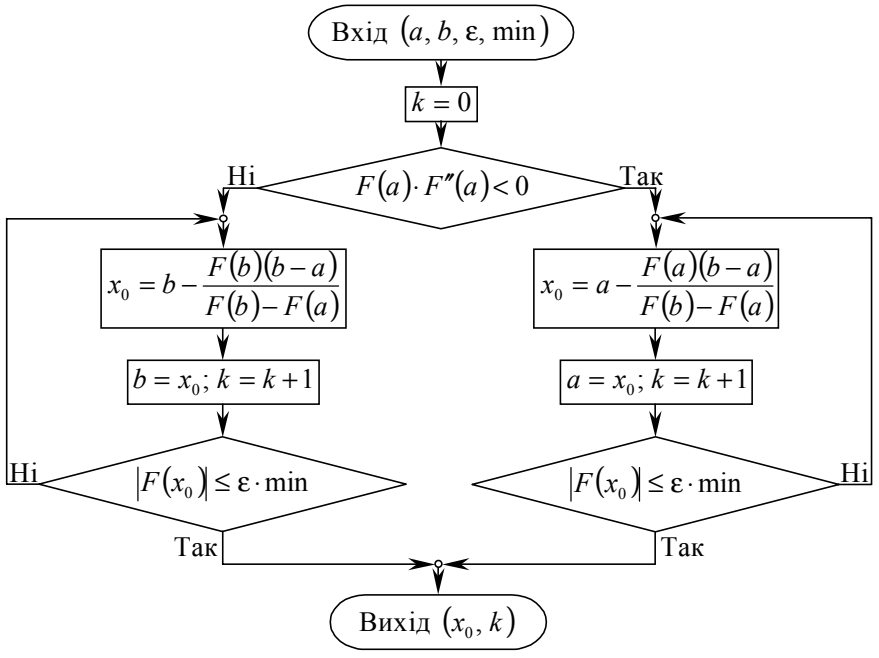


Рис. 2.10

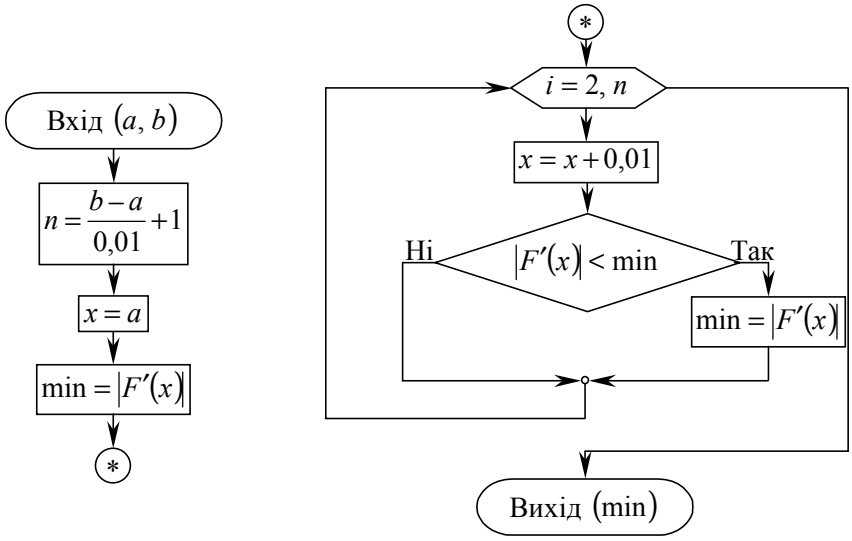


Рис. 2.11

Після реалізації даних алгоритмів на ЕОМ результат розв'язання задачі приклада 2.4 при ε = 10^{-6} отриманий у вигляді:

Корінь рівняння: x_0 = 1,399429. Кількість наближень: k = 4. $F(x_0)$ = = 0,000000.

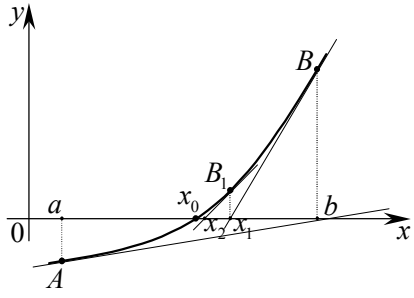
Метод дотичних і його реалізація на ЕОМ

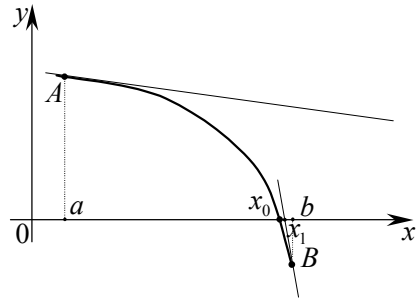
Метод дотичних (*метод Ньютона*) аналогічний методу хорд.

Нехай дано нелінійне рівняння (2.1), де $F(x)$ – неперервна на відрізку ізоляції кореня $[a, b]$ функція, яка має на ньому неперервні та знакопостійні похідні першого і другого порядків.

Геометричний зміст методу дотичних полягає в тому, що на малому відрізку $[a, b]$ дуга кривої $y = F(x)$ замінюється дотичною до цієї кривої і як наближене значення кореня береться точка перетинання цієї дотичної з віссю Ox.

Розглянемо окремо випадки, коли $F'(x) \cdot F''(x) > 0$ (див. рис. 2.9,***а***,***б***):





Проведемо дотичну до кривої $y = F(x)$ у точці B (якщо провести дотичну у точці A, то вона перетне Ox в деякій точці, яка не належить відрізку $[a, b]$).

З аналітичної геометрії відомо, що рівняння дотичної до кривої у деякій точці $C(x_c, y_c)$ має вигляд: $y - y_c = k(x - x_c)$, де $k = y'(x_c)$ – кутовий коефіцієнт. У нашому випадку для дотичної у точці $B(b, F(b))$ маємо

$$y - F(b) = F'(b)(x - b).$$

Для знаходження x_1 положимо $y = 0$ і дістанемо

$$x_1 = b - \frac{F(b)}{F'(b)}.$$

Знайдена формула має назву *формули методу дотичних*. Тепер корінь рівняння лежить у межах відрізка $[a, x_1]$. Якщо точність обчисленого значення x_1 недостатня, то необхідно виконати наступне наближення до кореня, застосовуючи метод дотичних на відрізку $[a, x_1]$:

$$x_2 = x_1 - \frac{F(x_1)}{F'(x_1)}.$$

Продовжуючи процес наближень далі, знаходимо:

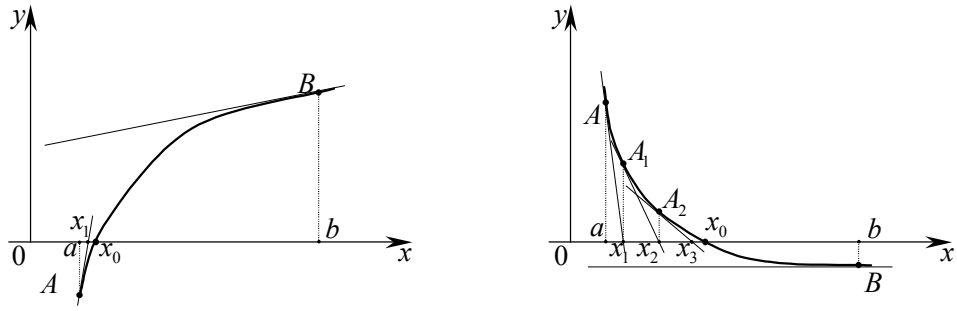
$$x_3 = x_2 - \frac{F(x_2)}{F'(x_2)},\ \ x_4 = x_3 - \frac{F(x_3)}{F'(x_3)}, \ldots,$$
$$x_k = x_{k-1} - \frac{F(x_{k-1})}{F'(x_{k-1})}.$$

Таким чином, коли $F'(x) \cdot F''(x) > 0$ обчислення в рамках методу дотичних здійснюються за рекурсивною формулою

$$\left.\begin{array}{c} x_1 = b - \dfrac{F(b)}{F'(b)};\ x_k = x_{k-1} - \dfrac{F(x_{k-1})}{F'(x_{k-1})}, \\ k = 2, 3, 4, \ldots, n. \end{array}\right\} \qquad (2.9)$$

За схемою (2.9) обчислюється послідовність наближених значень $x_1, x_2, x_3, \ldots, x_k$, кожен наступний член якої ближче до кореня x_0, ніж попередній. При цьому рухомим є кінець b відрізка ізоляції кореня, а кожен член зазначеної послідовності залишається більше істинного кореня x_0, тобто x_k – наближене значення кореня x_0 з перевишкою. Відмітимо, що в той же самий час метод хорд за схемою (2.6) дає значення x_k з недостачею, причому рухомим є кінець a.

У другому випадку, коли $F'(x) \cdot F''(x) < 0$ (див. рис. 2.9,*в*,*г*), міркування аналогічні, тільки дотичну до кривої $y = F(x)$ необхідно проводити у точці $A(a, F(a))$:



Таким чином,

$$\left.\begin{aligned} x_1 = a - \frac{F(a)}{F'(a)};\ x_k = x_{k-1} - \frac{F(x_{k-1})}{F'(x_{k-1})}, \\ k = 2, 3, 4, \ldots, n. \end{aligned}\right\} \qquad (2.10)$$

За схемою (2.10) обчислюється послідовність наближених значень x_1, $x_2, x_3, \ldots, x_k$, кожен наступний член якої ближче до кореня x_0, ніж попередній. При цьому рухомим кінцем відрізка ізоляції кореня є a, внаслідок чого x_k – наближене значення кореня x_0 з недостачею. Відзначимо, що в той же самий час метод хорд за схемою (2.7) дає значення x_k з перевишкою, причому рухомим кінцем є b.

Схеми (2.9) і (2.10) відрізняються тільки значенням початкового наближення до кореня x_1. При виборі початкового наближення (аналогічно методу хорд) можна керуватися наступним правилом: *за початкову точку у виразі для* x_1 *слід вибирати той кінець відрізка* $[a, b]$, *в якому знак функції* $F(x)$ *співпадає зі знаком її другої похідної*.

Для оцінки похибки приблизно обчисленого кореня x_k можна користуватися формулою (2.8). Без доведення відзначимо, що ця формула є загальною для методів хорд і дотичних.

Приклад 2.5. Методом дотичних уточнити корінь рівняння $\sin 2x - \ln x = 0$ на відокремленому відрізку $[1; 1{,}6]$ з точністю $\varepsilon = 10^{-2}$.

Розв'язання. Для розв'язання даної задачі будемо користуватися результатами обчислень, отриманих у прикладі 2.4.

Оскільки $F(a) \cdot F''(x) < 0$, то

$$x_1 = b - \frac{F(b)}{F'(b)} = 1{,}6 - \frac{F(1{,}6)}{F'(1{,}6)} = 1{,}398; \quad F(1{,}398) = 0{,}003 < 0{,}018.$$

Таким чином, уже перше наближення до кореня дозволяє записати:

$$x_0 = 1{,}40 \pm 0{,}01, \quad \text{а} \quad |F(1{,}40)| = 0{,}00.$$

Алгоритм реалізації методу дотичних із заданою точністю ε з урахуванням алгоритму, зображеного на рис. 2.11, наведено на рис. 2.12.

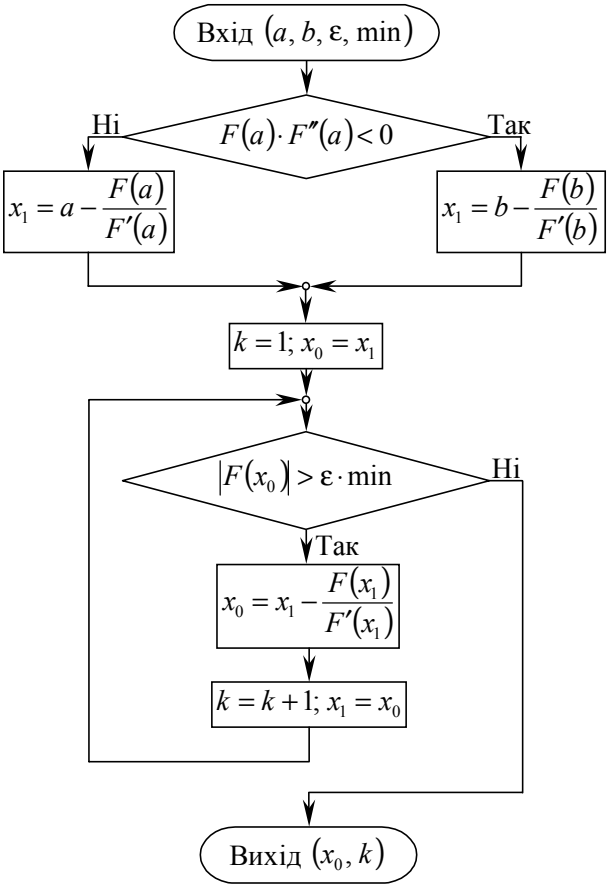


Рис. 2.12

Після реалізації даних алгоритмів на ЕОМ результат розв'язання задачі приклада 2.5 при $\varepsilon = 10^{-6}$ отриманий у вигляді:

Корінь рівняння: x_0 = 1,399429. Кількість наближень: k = 2. $F(x_0)$ = = –0,000000.

3. ЧИСЕЛЬНЕ РОЗВ'ЯЗАННЯ СИСТЕМ ЛІНІЙНИХ АЛГЕБРАЇЧНИХ РІВНЯНЬ. МЕТОД ГАУССА І РЕАЛІЗАЦІЯ ЙОГО НА ЕОМ

Обчислювальні методи лінійної алгебри відіграють особливу роль у чисельному аналізі. Це обумовлено, по меншій мірі, двома причинами.

По-перше, більшість лінійних задач математичного аналізу, диференціальних та інтегральних рівнянь зводяться до розв'язання задач лінійної алгебри, насамперед до розв'язання *систем лінійних алгебраїчних рівнянь* (СЛАР). Таким чином, чисельні методи розв'язання СЛАР часто є інструментом чисельного розв'язання великого класу математичних, а отже, і науково-технічних задач.

По-друге, відомо, що більшість методів розв'язання нелінійних задач також зводяться до розв'язання деякої послідовності лінійних систем. Справа у тому, що більшість нелінійних моделей в малому околі деякого розв'язку можна описати лінійними моделями. Відповідно, першим кроком розв'язання нелінійних задач є дослідження лінеаризованих моделей та застосування чисельних методів лінійної алгебри (зокрема, чисельних методів розв'язання СЛАР).

Нехай подана система лінійних алгебраїчних рівнянь n-го порядку:

$$\left.\begin{aligned} &a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n = b_1,\\ &a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots + a_{2n}x_n = b_2,\\ &a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \ldots + a_{3n}x_n = b_3,\\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots,\\ &a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \ldots + a_{nn}x_n = b_n. \end{aligned}\right\} \qquad (3.1)$$

Дана СЛАР може бути записана в матричному вигляді:

$$\mathbf{A} \cdot \mathbf{X} = \mathbf{B}$$

де $\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \dots & \dots & \dots & \ddots & \dots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{pmatrix}$ – квадратна матриця вимірністю $n \times n$ *коефіцієнтів при невідомих системи* a_{ij}, де індекс $i = 1, 2, 3, \dots, n$ визначає номер рядка матриці, а $j = 1, 2, 3, \dots, n$ – номер стовпця;

$\mathbf{B} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \dots \\ b_n \end{pmatrix}$ – n-вимірний вектор-стовпець *вільних членів* (*правих частин*) системи b_i;

$\mathbf{X} = (x_1 \ x_2 \ x_3 \ \dots \ x_n)$ – n-вимірний вектор-рядок *невідомих* системи x_j.

Розв'язком СЛАР (3.1) називається будь-яка сукупність чисел (*коренів*) $x_1 = c_1, x_2 = c_2, x_3 = c_3, \dots, x_n = c_n$, яка будучи підставлена на місце невідомих системи, перетворює всі ці рівняння у тотожності.

Якщо система (3.1) має розв'язок, то вона називається *сумісною*, якщо немає розв'язку – *несумісною* чи *суперечною*.

Сумісна СЛАР може мати одне або кілька розв'язків. Якщо СЛАР має один єдиний розв'язок, то вона називається *визначеною*, а якщо більше одного розв'язку, то *невизначеною*.

Якщо коефіцієнти одного з рівнянь системи є лінійною комбінацією коефіцієнтів інших рівнянь, то система *невизначена*.

Проілюструємо зазначене наступним прикладом. Нехай подана СЛАР 2-го порядку $\left.\begin{matrix} x_1 + x_2 = 1, \\ 3x_1 + 3x_2 = 3. \end{matrix}\right\}$ Видно, що коефіцієнти першого рівняння отримані з коефіцієнтів другого рівняння лінійним перетворенням $a_{1j} = a_{2j} - 2$. Внаслідок цього визначник матриці коефіцієнтів при невідомих системи $\det \mathbf{A} = \begin{pmatrix} 1 & 1 \\ 3 & 3 \end{pmatrix} = 0$ і, отже, система має нескінченну множину розв'язків, яку можна отримати за формулами $\left.\begin{matrix} x_1 = t, \\ x_2 = 1 - t, \end{matrix}\right\}$ де t – довільний параметр.

Дві системи лінійних рівнянь однакового порядку називаються *еквівалентними*, якщо вони обидві несумісні, або ж сумісні й мають однакові розв'язки.

Існує багато чисельних методів розв'язання СЛАР, які можна поділити на дві групи.

1. *Точні* (або *прямі*) методи, які характеризуються тим, що дають змогу дістати розв'язок через скінченне число арифметичних операцій. Якщо всі ці операції виконуються без помилок округлення, то розв'язок заданої системи буде точним.

До прямих методів відносять: метод Крамера, метод Гаусса, метод квадратного кореня, метод головного елемента та ін. Вони використовуються на практиці, як правило, якщо порядок значень a_{ij} і b_i не вище 10^3.

2. *Наближені* (*ітераційні*) методи дозволяють діставати корені СЛАР із заданою точністю. До цих методів відносять: метод простої ітерації, метод Зейделя, метод релаксації, метод ортогоналізації та ін. На практиці вони використовуються, якщо порядок чисел системи 10^6.

Зрозуміло, що внаслідок неминучих помилок округлення, які виникають при розв'язанні СЛАР як вручну, так і з допомогою ЕОМ, результати навіть точних чисельних методів виявляються наближеними. Для того щоб оцінити *відхил* (абсолютну похибку) отриманого розв'язку, в систему підставляють корені $x_j = c_j$, а потім знаходять різницю між лівою та правою частинами СЛАР:

$$\left.\begin{aligned} z_1 &= a_{11}c_1 + a_{12}c_2 + a_{13}c_3 + \ldots + a_{1n}c_n - b_1, \\ z_2 &= a_{21}c_1 + a_{22}c_2 + a_{23}c_3 + \ldots + a_{2n}c_n - b_2, \\ z_3 &= a_{31}c_1 + a_{32}c_2 + a_{33}c_3 + \ldots + a_{3n}c_n - b_3, \\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots, \\ z_n &= a_{n1}c_1 + a_{n2}c_2 + a_{n3}c_3 + \ldots + a_{nn}c_n - b_n. \end{aligned}\right\}$$

Якщо модуль максимального відхилу $z_k = \max\limits_{1\le i\le n}|z_i|$ менше за 10^{-10}, то вважають, що СЛАР розв'язана з достатнім ступенем точності.

Найбільш поширеним прямим методом розв'язання СЛАР є *метод Гаусса* (*метод послідовного виключення невідомих*). Виключення невідомих, на якому базується даний метод, виконується за допомогою наступних *елементарних перетворень*:

1. переставлення двох рівнянь системи;

2. множення обох частин рівняння системи на будь-яке відмінне від нуля число;

3. додавання до обох частин одного рівняння відповідних частин другого рівняння, помножених на будь-яке відмінне від нуля число.

В лінійній алгебрі доведено, що ці елементарні перетворення призводять дану СЛАР до еквівалентної, оскільки виконання елементарних перетворень рівносильне виразу одного невідомого системи через інше.

При розв'язанні СЛАР (3.1) методом Гауса вводиться поняття *головного елемента*. При виключенні невідомого x_1 – це елемент a_{11}, який повинен бути максимальним за модулем у першому стовпці системі. Якщо це не так, то перше рівняння і те рівняння системи, в якому $|a_{i1}| = \max$, необхідно поміняти місцями. Пошук головного елементу та відповідне переставлення рядків у рамках методу Гаусса називається *методом виключення невідомих з частковим упорядкуванням за рядками*. Доцільність використання саме цього методу стане зрозумілою дещо пізніше, коли буде обговорюватися питання створення алгоритму для практичної реалізації методу Гаусса на ЕОМ.

Будемо виключати невідоме x_1 із усіх рівнянь системи (3.1), крім першого. Для цього (після часткового упорядкування) розділимо кожне рівняння цієї системи на коефіцієнт a_{i1} при x_1 та віднімемо перше рівняння від усіх інших.

а) Ділення всіх рівнянь системи (3.1) на a_{i1}:

$$\left.\begin{aligned}
&x_1 + \frac{a_{12}}{a_{11}}x_2 + \frac{a_{13}}{a_{11}}x_3 + \ldots + \frac{a_{1n}}{a_{11}}x_n = \frac{b_1}{a_{11}},\\
&x_1 + \frac{a_{22}}{a_{21}}x_2 + \frac{a_{23}}{a_{21}}x_3 + \ldots + \frac{a_{2n}}{a_{21}}x_n = \frac{b_2}{a_{21}},\\
&x_1 + \frac{a_{32}}{a_{31}}x_2 + \frac{a_{33}}{a_{31}}x_3 + \ldots + \frac{a_{3n}}{a_{31}}x_n = \frac{b_3}{a_{31}},\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots,\\
&x_1 + \frac{a_{n2}}{a_{n1}}x_2 + \frac{a_{n3}}{a_{n1}}x_3 + \ldots + \frac{a_{nn}}{a_{n1}}x_n = \frac{b_n}{a_{n1}}.
\end{aligned}\right\} \qquad (3.2)$$

б) Віднімання першого рівняння від усіх інших рівнянь системи (3.2):

$$\left.\begin{aligned}
&x_1 + \frac{a_{12}}{a_{11}}x_2 + \frac{a_{13}}{a_{11}}x_3 + \ldots + \frac{a_{1n}}{a_{11}}x_n = \frac{b_1}{a_{11}},\\
&\left(\frac{a_{22}}{a_{21}} - \frac{a_{12}}{a_{11}}\right)x_2 + \left(\frac{a_{23}}{a_{21}} - \frac{a_{13}}{a_{11}}\right)x_3 + \ldots + \left(\frac{a_{2n}}{a_{21}} - \frac{a_{1n}}{a_{11}}\right)x_n = \frac{b_2}{a_{21}} - \frac{b_1}{a_{11}},\\
&\left(\frac{a_{32}}{a_{31}} - \frac{a_{12}}{a_{11}}\right)x_2 + \left(\frac{a_{33}}{a_{31}} - \frac{a_{13}}{a_{11}}\right)x_3 + \ldots + \left(\frac{a_{3n}}{a_{31}} - \frac{a_{1n}}{a_{11}}\right)x_n = \frac{b_3}{a_{31}} - \frac{b_1}{a_{11}},\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots,\\
&\left(\frac{a_{n2}}{a_{n1}} - \frac{a_{12}}{a_{11}}\right)x_2 + \left(\frac{a_{n3}}{a_{n1}} - \frac{a_{13}}{a_{11}}\right)x_3 + \ldots + \left(\frac{a_{nn}}{a_{n1}} - \frac{a_{1n}}{a_{11}}\right)x_n = \frac{b_n}{a_{n1}} - \frac{b_1}{a_{11}}.
\end{aligned}\right\} \qquad (3.3)$$

Позначивши нові коефіцієнти при невідомих системи (3.3) як a_{ij}^1, а праві частини – b_i^1 ($i, j = 2, 3, 4, \ldots, n$), запишемо:

$$\left.\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n &= b_1, \\ a_{22}^1 x_2 + a_{23}^1 x_3 + \ldots + a_{2n}^1 x_n &= b_2^1, \\ a_{32}^1 x_2 + a_{33}^1 x_3 + \ldots + a_{3n}^1 x_n &= b_3^1, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots, & \\ a_{n2}^1 x_2 + a_{n3}^1 x_3 + \ldots + a_{nn}^1 x_n &= b_n^1. \end{aligned}\right\} \qquad (3.4)$$

Аналогічно виключенню невідомого x_1 із другого та наступних рівнянь системи (3.1) можна виключити невідоме x_2 із третього та наступних рівнянь системи (3.4). Для цього до системи рівнянь $(n-1)$-го порядку, яка дістається, якщо з системи (3.4) викреслити перше рівняння, після часткового упорядкування необхідно застосувати перетворення "**а**" та "**б**":

$$\left.\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n &= b_1, \\ x_2 + \frac{a_{23}^1}{a_{22}^1}x_3 + \ldots + \frac{a_{2n}^1}{a_{22}^1}x_n &= \frac{b_2^1}{a_{22}^1}, \\ x_2 + \frac{a_{33}^1}{a_{32}^1}x_3 + \ldots + \frac{a_{3n}^1}{a_{32}^1}x_n &= \frac{b_3^1}{a_{32}^1}, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots, & \\ x_2 + \frac{a_{n3}^1}{a_{n2}^1}x_3 + \ldots + \frac{a_{nn}^1}{a_{n2}^1}x_n &= \frac{b_n^1}{a_{n2}^1}. \end{aligned}\right\}$$

$$\left.\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n &= b_1, \\ x_2 + \frac{a_{23}^1}{a_{22}^1}x_3 + \ldots + \frac{a_{2n}^1}{a_{22}^1}x_n &= \frac{b_2^1}{a_{22}^1}, \\ \left(\frac{a_{33}^1}{a_{32}^1} - \frac{a_{23}^1}{a_{22}^1}\right)x_3 + \ldots + \left(\frac{a_{3n}^1}{a_{32}^1} - \frac{a_{2n}^1}{a_{22}^1}\right)x_n &= \frac{b_3^1}{a_{32}^1} - \frac{b_2^1}{a_{22}^1}, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots, & \\ \left(\frac{a_{n3}^1}{a_{n2}^1} - \frac{a_{23}^1}{a_{22}^1}\right)x_3 + \ldots + \left(\frac{a_{nn}^1}{a_{n2}^1} - \frac{a_{2n}^1}{a_{22}^1}\right)x_n &= \frac{b_n^1}{a_{n2}^1} - \frac{b_2^1}{a_{22}^1}. \end{aligned}\right\} \qquad (3.5)$$

Позначивши нові коефіцієнти при невідомих системи (3.5) як a_{ij}^2, а праві частини – b_i^2 ($i, j = 3, 4, 5, \ldots, n$), запишемо:

$$\left.\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n &= b_1, \\ a_{22}^1 x_2 + a_{23}^1 x_3 + \ldots + a_{2n}^1 x_n &= b_2^1, \\ a_{33}^2 x_3 + \ldots + a_{3n}^2 x_n &= b_3^2, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots, & \\ a_{n3}^2 x_3 + \ldots + a_{nn}^2 x_n &= b_n^2. \end{aligned}\right\}$$

Аналогічні перетворення необхідно проводити до того часу, поки з останнього рівняння системи не буде виключено x_{n-1}. Результатом перетворень буде еквівалентна системі (3.1) наступна СЛАР *трикутного вигляду*:

$$\left.\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n &= b_1, \\ a_{22}^1 x_2 + a_{23}^1 x_3 + \ldots + a_{2n}^1 x_n &= b_2^1, \\ a_{33}^2 x_3 + \ldots + a_{3n}^2 x_n &= b_3^2, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots, & \\ a_{nn}^{n-1} x_n &= b_n^{n-1}. \end{aligned}\right\} \qquad (3.6)$$

Із системи (3.6) послідовно знаходять значення всіх невідомих, починаючи з останнього:

$$\begin{gathered} x_n = \frac{b_n^{n-1}}{a_{nn}^{n-1}} \Rightarrow x_{n-1} = \frac{b_{n-1}^{n-2} - a_{n-1,\,n}^{n-2} x_n}{a_{n-1,\,n-1}^{n-2}} \Rightarrow \\ \Rightarrow x_{n-2} = \frac{b_{n-2}^{n-3} - a_{n-2,\,n-1}^{n-3} x_{n-1} - a_{n-2,\,n}^{n-3} x_n}{a_{n-2,\,n-2}^{n-3}} \Rightarrow \ldots \Rightarrow \\ \Rightarrow x_1 = \frac{b_1 - a_{12}x_2 - a_{13}x_3 - \ldots - a_{1n}x_n}{a_{11}}. \end{gathered} \qquad (3.7)$$

Таким чином, процес розв'язання СЛАР (3.1) методом Гаусса включає в себе два етапи. Перший етап полягає у послідовному виключенні невідомих і призводить до системи вигляду (3.6), називається *прямим ходом*. Другий етап складається з послідовного знаходження значень невідомих (коренів системи) за формулами (3.7) і називається *зворотнім ходом*.

Приклад 3.1. За допомогою схеми Гаусса розв'язати наступну систему лінійних алгебраїчних рівнянь:

$$\left.\begin{aligned} 2x_1 - x_2 + x_3 &= 5,\\ 5x_1 + x_2 - x_3 &= 2,\\ 4x_1 - x_2 - x_3 &= 3. \end{aligned}\right\}$$

Розв'язання. Застосуємо прямий хід методу Гаусса з частковим упорядкуванням за рядками:

Знаходимо максимальний за модулем елемент у першому стовпці та міняємо місцями відповідні рядки:

$$\left.\begin{aligned} 5x_1 + x_2 - x_3 &= 2,\\ 2x_1 - x_2 + x_3 &= 5,\\ 4x_1 - x_2 - x_3 &= 3. \end{aligned}\right\}$$

$\rightarrow$

Виключаємо невідоме x_1 з другого і третього рівнянь:

$$\left.\begin{aligned} x_1 + \frac{1}{5}x_2 - \frac{1}{5}x_3 &= \frac{2}{5},\\ x_1 - \frac{1}{2}x_2 + \frac{1}{2}x_3 &= \frac{5}{2},\\ x_1 - \frac{1}{4}x_2 - \frac{1}{4}x_3 &= \frac{3}{4}. \end{aligned}\right\} \Rightarrow \left.\begin{aligned} x_1 + \frac{1}{5}x_2 - \frac{1}{5}x_3 &= \frac{2}{5},\\ -\frac{7}{10}x_2 + \frac{7}{10}x_3 &= \frac{21}{10},\\ -\frac{9}{20}x_2 - \frac{1}{20}x_3 &= \frac{7}{20}. \end{aligned}\right\}$$

Оскільки 7/10 > 9/20, рядки не переставляємо:

$$\left.\begin{aligned} x_1 + \frac{1}{5}x_2 - \frac{1}{5}x_3 &= \frac{2}{5},\\ -\frac{7}{10}x_2 + \frac{7}{10}x_3 &= \frac{21}{10},\\ -\frac{9}{20}x_2 - \frac{1}{20}x_3 &= \frac{7}{20}. \end{aligned}\right\}$$

$\rightarrow$

Виключаємо невідоме x_2 з третього рівняння:

$$\left.\begin{aligned} x_1 + \frac{1}{5}x_2 - \frac{1}{5}x_3 &= \frac{2}{5},\\ x_2 - x_3 &= -3,\\ x_2 + \frac{1}{9}x_3 &= -\frac{7}{9}. \end{aligned}\right\} \Rightarrow \left.\begin{aligned} x_1 + \frac{1}{5}x_2 - \frac{1}{5}x_3 &= \frac{2}{5},\\ x_2 - x_3 &= -3,\\ \frac{10}{9}x_3 &= \frac{20}{9}. \end{aligned}\right\}$$

Нарешті зворотнім ходом дістанемо $x_3 = 2 \Rightarrow x_2 = -3 + 2 = -1 \Rightarrow x_1 = \frac{2}{5} -$

$$-\frac{1}{5}(-1) + \frac{1}{5}\cdot 2 = 1.$$

Таким чином, $x_1 = 1$; $x_2 = -1$; $x_3 = 2$.

Перевіримо правильність отриманих коренів шляхом обчислення відхилів:

$$\left.\begin{aligned} z_1 &= 2 - 5\cdot 1 + 1 + 2 \equiv 0,\\ z_2 &= 5 - 2\cdot 1 - 1 - 2 \equiv 0,\\ z_3 &= 3 - 4\cdot 1 - 1 + 2 \equiv 0. \end{aligned}\right\}$$

Потреба у чисельній реалізації на ЕОМ методу Гаусса пов'язана в основному з вимірністю СЛАР. Розв'язок СЛАР малої вимірності можна дістати аналітично у вигляді формул (як це було зроблено у прикладі 3.1). Якщо вимірність СЛАР $n \geq 5$, то аналітичний розв'язок дістати важко через велику кількість та громіздкість перетворень, і саме в таких випадках доводиться використовувати ЕОМ.

Для розробки алгоритму розв'язання СЛАР методом Гаусса перетворимо системи (3.3) та (3.5) до наступного вигляду:

$$\left.\begin{aligned}
&x_1 + \frac{a_{12}}{a_{11}} x_2 + \frac{a_{13}}{a_{11}} x_3 + \ldots + \frac{a_{1n}}{a_{11}} x_n = \frac{b_1}{a_{11}},\\
&\left(a_{22} - a_{12}\frac{a_{21}}{a_{11}}\right)x_2 + \left(a_{23} - a_{13}\frac{a_{21}}{a_{11}}\right)x_3 + \ldots + \left(a_{2n} - a_{1n}\frac{a_{21}}{a_{11}}\right)x_n =\\
&= b_2 - b_1\frac{a_{21}}{a_{11}},\\
&\left(a_{32} - a_{12}\frac{a_{31}}{a_{11}}\right)x_2 + \left(a_{33} - a_{13}\frac{a_{31}}{a_{11}}\right)x_3 + \ldots + \left(a_{3n} - a_{1n}\frac{a_{31}}{a_{11}}\right)x_n =\\
&= b_3 - b_1\frac{a_{31}}{a_{11}},\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots,\\
&\left(a_{n2} - a_{12}\frac{a_{n1}}{a_{11}}\right)x_2 + \left(a_{n3} - a_{13}\frac{a_{n1}}{a_{11}}\right)x_3 + \ldots + \left(a_{nn} - a_{1n}\frac{a_{n1}}{a_{11}}\right)x_n =\\
&= b_n - b_1\frac{a_{n1}}{a_{11}}.
\end{aligned}\right\} \quad (3.3^1)$$

$$\left.\begin{aligned}
&a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n = b_1,\\
&x_2 + \frac{a^1_{23}}{a^1_{22}} x_3 + \ldots + \frac{a^1_{2n}}{a^1_{22}} x_n = \frac{b^1_2}{a^1_{22}},\\
&\left(a^1_{33} - a^1_{23}\frac{a^1_{32}}{a^1_{22}}\right)x_3 + \ldots + \left(a^1_{3n} - a^1_{2n}\frac{a^1_{32}}{a^1_{22}}\right)x_n = b^1_3 - b^1_2\frac{a^1_{32}}{a^1_{22}},\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots,\\
&\left(a^1_{n3} - a^1_{23}\frac{a^1_{n2}}{a^1_{22}}\right)x_3 + \ldots + \left(a^1_{nn} - a^1_{2n}\frac{a^1_{n2}}{a^1_{22}}\right)x_n = b^1_n - b^1_2\frac{a^1_{n2}}{a^1_{22}}.
\end{aligned}\right\} \quad (3.5^1)$$

Як видно з (3.31) та (3.51), при обчисленні нових коефіцієнтів при невідомих a_{ij}^1, a_{ij}^2, a_{ij}^3, …, які входять у систему трикутного вигляду (3.6), виключається можливість виконання неприпустимої операції ділення на нуль. Справа в тому, що проведені перетворення (3.3.) $\Rightarrow$ (3.3[1]) та (3.5) $\Rightarrow$ (3.5[1]) дозволяють виконувати ділення тільки на головні елементи a_{11}, a_{22}^1, a_{33}^2, …, а оскільки абсолютне значення головного елементу є максимальним за модулем у відповідному стовпці, то воно не може дорівнювати нулю.

На першому кроці (при виключенні невідомого x_1) головним елементом є a_{11}. На наступному кроці (при виключенні x_2) головним буде a_{22}^1 потім a_{33}^2, a_{44}^3, …, $a_{n-1,n-1}^{n-2}$. Введемо додаткову змінну k для позначення номера головного елементу ($k = 1, 2, 3, \ldots, n-1$).

Таким чином, нові коефіцієнти при невідомих та праві частини СЛАР на першому та наступних кроках прямого ходу можна алгоритмічно визначити за формулами

$$\left.\begin{gathered} a_{ij}^k = a_{ij}^{k-1} - a_{kj}^{k-1}\frac{a_{ik}^{k-1}}{a_{kk}^{k-1}};\ a_{ik} = 0; \\ b_i^k = b_i^{k-1} - b_k^{k-1}\frac{a_{ik}^{k-1}}{a_{kk}^{k-1}}, \\ k = 1, 2, 3, \ldots, n-1; \\ i, j = k+1, k+2, k+3, \ldots, n. \end{gathered}\right\} \qquad (3.8)$$

Згідно формул (3.7), зворотній хід також можна виразити алгоритмічно:

$$\left.\begin{gathered} x_n = \frac{b_n^{n-1}}{a_{nn}^{n-1}};\ x_i = \frac{b_i^{i-1} - \sum\limits_{j=i+1,i+2,\ldots}^{n} a_{ij}^{i-1}x_j}{a_{ii}^{i-1}}, \\ i = n-1, n-2, n-3, \ldots, 1. \end{gathered}\right\} \qquad (3.9)$$

На рис. 3.1 наведено структурно-логічну схему алгоритму для знаходження коренів системи лінійних алгебраїчних рівнянь методом Гаусса з частковим упорядкуванням за рядками. Розрахунок абсолютної похибки отриманих коренів здійснюється за формулою $z_i = \sum\limits_{j=1}^{n} a_{ij}x_j - b_i$ ($i = 1, 2, 3, \ldots, n$), згідно з алгоритмом на рис. 3.2.

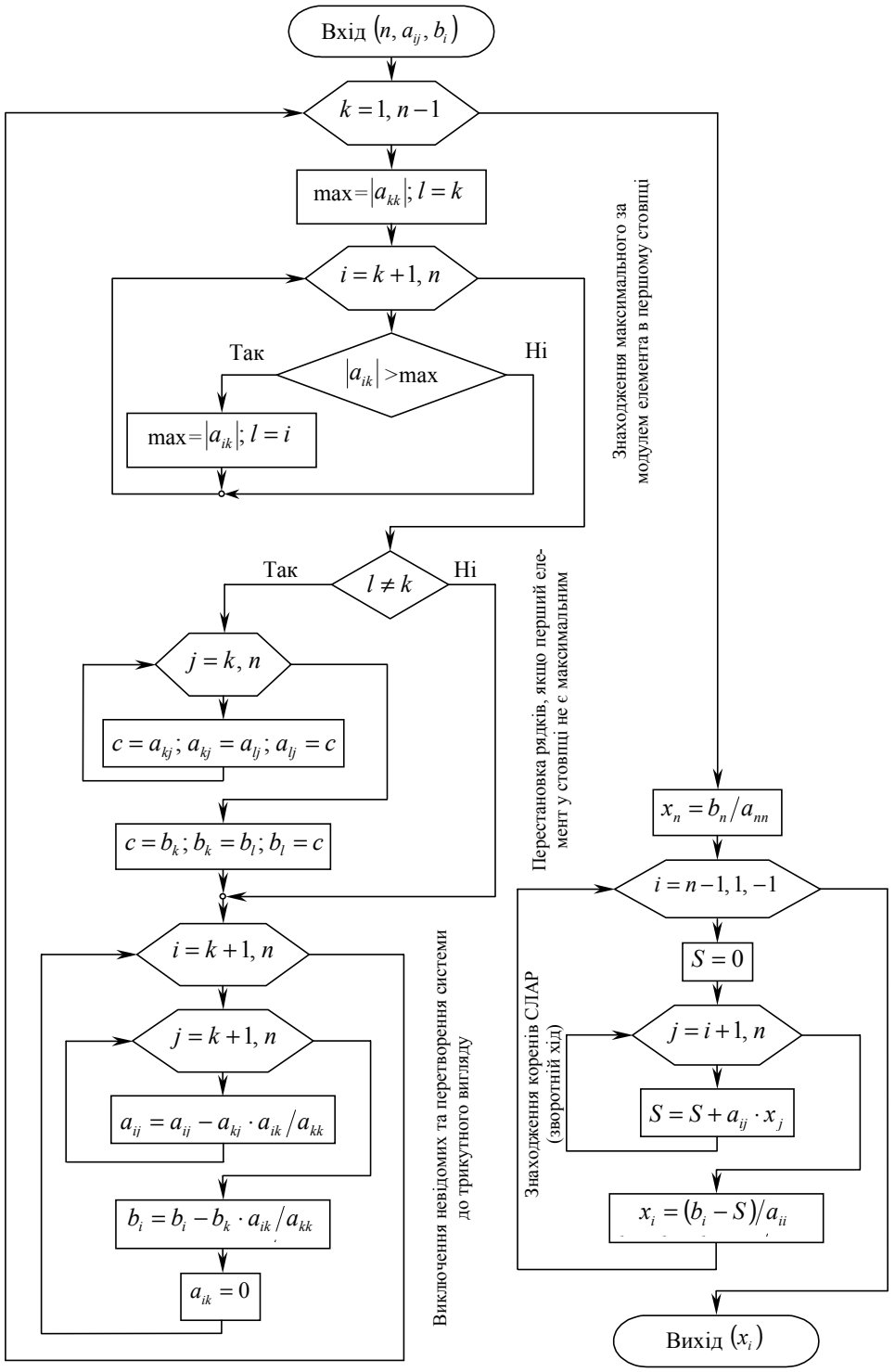

Вхід (n, a_{ij}, b_i)
$k = 1, n-1$
$\max = |a_{kk}|; l = k$
$i = k+1, n$
Так
Ні
$|a_{ik}| > \max$
$\max = |a_{ik}|; l = i$
Знаходження максимального за модулем елемента в першому стовпці
$l \neq k$
Так
Ні
Перестановка рядків, якщо перший елемент у стовпці не є максимальним
$j = k, n$
$c = a_{kj}; a_{kj} = a_{lj}; a_{lj} = c$
$c = b_k; b_k = b_l; b_l = c$
$i = k+1, n$
$j = k+1, n$
$a_{ij} = a_{ij} - a_{kj} \cdot a_{ik} / a_{kk}$
$b_i = b_i - b_k \cdot a_{ik} / a_{kk}$
$a_{ik} = 0$
Виключення невідомих та перетворення системи до трикутного вигляду
$x_n = b_n / a_{nn}$
$i = n-1, 1, -1$
$S = 0$
$j = i+1, n$
$S = S + a_{ij} \cdot x_j$
$x_i = (b_i - S) / a_{ii}$
Знаходження коренів СЛАР (зворотній хід)
Вихід (x_i)

Рис. 3.1

Після реалізації даних алгоритмів на ЕОМ для СЛАР

$$\left.\begin{aligned}
12{,}345x_1 - 11{,}234x_2 + 5{,}678x_3 + 11{,}345x_4 &= -12{,}345,\\
-2{,}123x_1 + 34{,}567x_2 + 11{,}234x_3 + 0{,}234x_4 &= 23{,}456,\\
0{,}234x_1 + 12{,}998x_2 - 11{,}234x_3 + 12{,}456x_4 &= -23{,}456,\\
9{,}234x_1 - 12{,}456x_2 + 3{,}456x_3 - 0{,}345x_4 &= -1{,}234,
\end{aligned}\right\}$$

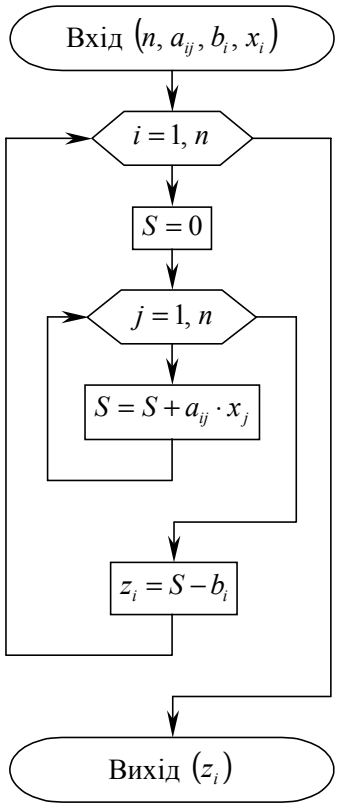


Рис. 3.2

отримуємо наступні результати:

$x_1 = -2{,}0820325983\text{E}-01;$

$x_2 = 2{,}9750067942\text{E}-01;$

$x_3 = 1{,}1570646408\text{E}+00;$

$x_4 = -1{,}1460926536\text{E}+00.$

$z_1 = 0{,}0000000000\text{E}+00;$

$z_2 = 2{,}9103830457\text{E}-11;$

$z_3 = -2{,}9103830457\text{E}-11;$

$z_4 = 1{,}8189894035\text{E}-12.$

Якщо наближені значення коренів, обчислених за схемою Гаусса, є достатньо точними, то їх уточнювати непотрібно.

У випадку необхідності можна уточнити розв'язок системи. Для цього достатньо розв'язати СЛАР із початковою матрицею коефіцієнтів, але з новим стовпцем правих частин, які складаються з відхилів (при цьому можна користуватися тією ж схемою Гаусса). Отримані значення поправок потім додають до раніше знайдених значень невідомих.

4. ІНТЕРПОЛЯЦІЯ ФУНКЦІЙ. ІНТЕРПОЛЯЦІЙНИЙ ПОЛІНОМ ЛАГРАНЖА. ОБЧИСЛЕННЯ ЗНАЧЕННЯ ПОЛІНОМУ ЛАГРАНЖА НА ЕОМ

4.1. Способи завдання функцій

Для аналізу різноманітних об'єктів та явищ, з якими людина зустрічається у своїй практичній діяльності, доводиться використовувати апарат математичних функцій. Математичні функції необхідні для описування досліджуваних об'єктів чи явищ, виявлення в них форм зв'язку.

Зупинимося на таких формах зв'язку, для яких деяка величина y, що характеризує процес або явище, залежить від сукупності незв'язаних між собою величин $x_1, x_2, x_3, \ldots, x_n$, причому така залежність характеризується тим, що кожному набору $(x_1, x_2, x_3, \ldots, x_n)$ відповідає єдине значення величини y. Вказана однозначна відповідність називається *функціональною залежністю*, а сама змінна величина y – *функцією*, залежною від n незалежних аргументів $x_1, x_2, x_3, \ldots, x_n$. Формально таку функціональну залежність прийнято записувати у вигляді $y = f(x_1, x_2, x_3, \ldots, x_n)$.

Якщо величина y є функцією однієї незалежної величини x, то говорять про функцію однієї змінної $y = f(x)$.

Наприклад, площа кругу $S = \pi R^2$ є функцією незалежної змінної – радіусу кола R, тобто $S = f(R)$. Об'єм прямокутного паралелепіпеду $V = abh$ є вже функцією трьох змінних і т. п.

Зі шкільного курсу математичного аналізу відомі три способи завдання функціональної залежності: 1. аналітичний; 2. графічний; 3. табличний.

Найбільш загальним та зручним є *аналітичний спосіб завдання* функціональної залежності. Наприклад, зв'язок пройденого шляху та часу при рівноприскореному русі матеріальної точки виражається як $S = v_0 t + 0{,}5at^2$. Позитивною властивістю аналітичного способу завдання функції є можливість діставати значення y для будь-якого фіксованого аргументу x із

довільною точністю. До недоліків цього способу можна віднести те, що доводиться виконувати всю послідовність розрахунків, крім того, аналітичний спосіб завдання функції не є наочним.

Графічний спосіб завдання функціональної залежності є наочним і полягає у побудові графіку функції $y = f(x)$. Графіком даної функції $y = f(x)$ називається геометричне місце точок площини xOy, координати яких задовольняють рівнянню $y = f(x)$.

Табличний спосіб завдання функціональної залежності особливо поширений у фізиці, техніці, економіці й частіше за все виникає у результаті експерименту чи спостережень. Нехай, наприклад, у результаті спостереження встановлена кількість виробів, що випускається в оброблювальному цеху на поточний час робочого дня:

Поточний час (t_i)	0	1	1,5	2	2,5	3	4	5,5	6
Кількість виробів на t_i час (N)	0	51	77	109	123	160	198	278	302

Таким чином, даною таблицею зображена функція $N(t)$. Перевагою табличного способу завдання функції є те, що для кожного значення незалежної змінної, зафіксованого у таблиці, можна одразу без вимірів та розрахунків знайти відповідні значення функції. Недоліком табличного способу є те, що неможливо задати функцію на всій області її визначення, тобто завжди знайдуться такі значення незалежної змінної, яких немає в таблиці.

4.2. Математична постановка задачі інтерполяції

Як вже зазначалося в підрозд. 4.1, в економіці та техніці часто доводиться стикатися з необхідністю обчислювати значення функції $y = f(x)$ в точках x, значення яких відрізняються від значень аргументу, зафіксованих у таблиці. Крім того, в деяких випадках, не дивлячись на те, що аналітичний вираз функції $y = f(x)$ відомий, він є досить складним і незручним для математичних перетворень. Подібні задачі на практиці формалізуються як математичні *задачі інтерполяції*.

Отже, нехай відомі значення деякої функції $f(x)$ утворюють наступну таблицю. Для того щоб з'явилась можливість обчислювати будь-які значення цієї функції, застосовують спеціальний прийом: за початковою інформацією (за таблицею) будують наближену функцію $F(x)$, що близька до початкової функції $f(x)$ та аналітичний вираз якої придатний для обчислень і математичних перетворень:

x	x_0	x_1	x_2	…	x_n
y	y_0	y_1	y_2	…	y_n

$$f(x) = F(x). \tag{4.1}$$

Процес обчислення значень функції в точках x, які відрізняються від точок, зафіксованих в таблиці, називають *інтерполяцією функції* $f(x)$. Точки $x_0, x_1, x_2, \ldots, x_n$, розміщені в таблиці, прийнято називати *вузлами інтерполяції*.

Якщо аргумент x, для якого визначається наближене значення функції, належить відрізку $[x_0, x_n]$, то задача обчислення функції $F(x)$ називається *інтерполяцією у вузькому сенсі*. Якщо аргумент знаходиться за межами відрізку інтерполяції $[x_0, x_n]$, то задача визначення $F(x)$ називається *екстраполяцією*.

Класичний підхід до розв'язання задачі побудови наближеної функції $F(x)$ базується на вимозі строгої відповідності значень функцій $f(x)$ і $F(x)$ у вузлах інтерполяції, тобто

$$\left.\begin{array}{c} F(x_0) = y_0;\ F(x_1) = y_1; \\ F(x_2) = y_2; \ldots; F(x_n) = y_n. \end{array}\right\} \qquad (4.2)$$

Геометрично задача інтерполяції для функції однієї змінної $y = f(x)$ полягає у побудові кривої $y = F(x)$, яка проходить через точки площини $M_0(x_0; y_0)$, $M_1(x_1; y_1)$, $M_2(x_2; y_2)$, …, $M_n(x_n; y_n)$. Однак вже з рис. 4.1 інтуїтивно зрозуміло, що через дані точки можна провести нескінченну кількість кривих. Таким чином, задача знаходження функції $y = F(x)$ за скінченним числом точок функції $y = f(x)$ дуже невизначена. Іншими словами, критерію близькості початкової та наближеної функцій (4.2) недостатньо для побудови функції $y = F(x)$.

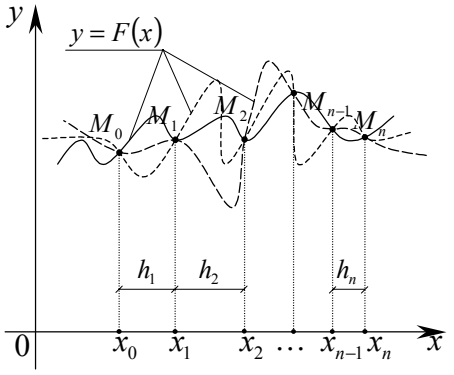


Рис. 4.1

Дана задача стає однозначною, якщо для функції $f(x)$ заданої $n + 1$ своїми значеннями, вибрати інтерполяційну функцію $F(x)$ конкретного вигляду, наприклад, поліном степеня n – $P_n(x)$.

Поліном $P_n(x)$, який задовольняє умовам (4.1), називається *інтерполяційним поліномом*, а відповідні формули – *інтерполяційними формулами*.

У випадку, коли $F(x)$ вибирається у класі степеневих функцій, інтерполяція називається *параболічною*.

Іноді доцільно використовувати інші види інтерполяції. Якщо інтерполяційна функція $f(x)$ є періодичною, то як клас інтерполяційних функцій $\{F(x)\}$ вибирають клас тригонометричних функцій. У деяких випадках як клас $\{F(x)\}$ доцільно вибирати раціональні функції.

4.3. Інтерполяційний поліном Лагранжа. Обчислення на ЕОМ значення інтерполяційного поліному в точці

Найбільш загальною формулою параболічної інтерполяції є інтерполяційна формула Лагранжа. Задача параболічної інтерполяції в цьому випадку формулюється наступним чином. На відрізку $[x_0, x_n]$ у вузлах інтерполяції $x_0, x_1, x_2, \ldots, x_n$ задається функція $f(x)$ своїми $n + 1$ значеннями. Треба знайти функцію $F(x)$ у вигляді поліному степеня n таким чином, щоб у вузлах інтерполяції його значення співпадали зі значеннями заданої функції $f(x)$, тобто щоб вони задовольняли рівняння (4.2).

Необхідно відразу зазначити, що така постановка задачі дозволяє знаходити інтерполяційну функцію навіть у тому випадку, коли вузли інтерполяції на відрізку $[x_0, x_n]$ довільно відстоять один від одного (вузли інтерполяції *нерівновіддалені*, тобто *крок інтерполяції* $h = x_{i+1} - x_i \neq \text{const}$).

Запишемо поліном $P_n(x)$ у вигляді:

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n, \tag{4.3}$$

де a_i $(i = 0, 1, 2, \ldots, n)$ – шукані постійні коефіцієнти.

Визначимо невідомі коефіцієнти поліному a_i, підставивши (4.3) у (4.2):

$$\left.\begin{aligned} y_0 &= a_0 + a_1 x_0 + a_2 x_0^2 + \ldots + a_n x_0^n, \\ y_1 &= a_0 + a_1 x_1 + a_2 x_1^2 + \ldots + a_n x_1^n, \\ y_2 &= a_0 + a_1 x_2 + a_2 x_2^2 + \ldots + a_n x_2^n, \\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots, \\ y_n &= a_0 + a_1 x_n + a_2 x_n^2 + \ldots + a_n x_n^n, \end{aligned}\right\} \tag{4.4}$$

де x_i, y_i $(i = 0, 1, 2, \ldots, n)$ – табличні значення аргументу та функції $f(x)$

Система (4.4) – це система лінійних алгебраїчних рівнянь $(n + 1)$-го порядку. Невідомі a_i цієї системи можна дістати, наприклад, за формулами Крамера:

$$a_0 = \frac{\Delta_0}{\Delta}, a_1 = \frac{\Delta_1}{\Delta}, a_2 = \frac{\Delta_2}{\Delta}, \ldots, a_n = \frac{\Delta_n}{\Delta}.$$

СЛАР (4.4) визначена (має єдиний розв'язок), якщо визначник Δ матриці, побудованої з коефіцієнтів при невідомих системи (4.4)

$$\begin{vmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ 1 & x_2 & x_2^2 & \dots & x_2^n \\ \dots & \dots & \dots & \ddots & \dots \\ 1 & x_n & x_2^2 & \dots & x_n^n \end{vmatrix}, \qquad (4.5)$$

відмінний від нуля.

Визначник матриці (4.5) відомий в алгебрі як *визначник Вандермонда*. Він дорівнює добутку всіляких різниць вузлів $x_1, x_2, \dots, x_n$: $\Delta = \prod_{0 \le i \le j \le n} (x_i - x_j)$.

Таким чином, $\Delta \neq 0$, оскільки всі вузли інтерполяції x_i різні.

Поліном (4.3) можна записати, не розв'язуючи систему (4.4), у вигляді

$$P_n(x) = y_0 \frac{(x-x_1)(x-x_2)\dots(x-x_n)}{(x_0-x_1)(x_0-x_2)\dots(x_0-x_n)} + y_1 \frac{(x-x_0)(x-x_2)\dots(x-x_n)}{(x_1-x_0)(x_1-x_2)\dots(x_1-x_n)} +$$
$$+ y_2 \frac{(x-x_0)(x-x_1)\dots(x-x_n)}{(x_2-x_0)(x_2-x_1)\dots(x_2-x_n)} + \dots + y_n \frac{(x-x_0)(x-x_1)\dots(x-x_{n-1})}{(x_n-x_0)(x_n-x_1)\dots(x_n-x_{n-1})},$$

або скорочено у вигляді

$$\left. P_n(x) = \sum_{i=0}^{n} y_i \prod_{\substack{j=0 \\ j \neq i.}}^{n} \frac{x-x_j}{x_i-x_j}, \right\} \qquad (4.6)$$

Поліном (4.6) називається *інтерполяційним поліномом Лагранжа*, причому його значення співпадають зі значеннями функції $f(x)$ у вузлах інтерполяції x_i.

Наведемо без доведення теорему, що дозволяє оцінити похибку приближення поліному (4.6) до функції $f(x)$ у точках, які відрізняються від вузлів x_i.

Нехай функція $f(x)$ на відрізку $[x_0, x_n]$ має всі похідні до $(n+1)$-го порядку включно, тобто $f(x) \in C^{n+1}[x_0, x_n]$. Тоді оцінку абсолютної похибки інтерполяційної формули Лагранжа можна виконати за формулою

$$\left|R_n(x)\right| = \left|f(x) - P_n(x)\right| \le \frac{M_{n+1}}{(n+1)!}\left|\omega_{n+1}(x)\right|, \qquad (4.7)$$

де $\omega_{n+1}(x) = (x-x_0)(x-x_1)(x-x_2)\dots(x-x_n)$, $M_{n+1} = \max\limits_{[x_0, x_n]}\left|f^{n+1}(x)\right|$.

Приклад 4.1. З'ясувати, з якою точністю можна обчислити значення sin(1,5) за допомогою інтерполяційної формули Лагранжа для функції $y = \sin x$, якщо взяти як вузли інтерполяції точки $x_0 = 0$, $x_1 = 0{,}5\pi$, $x_2 = \pi$.

Розв'язання. Дістанемо компоненти формули (4.7) при $n = 2$:

$$M_3 = \max_{[0,\,\pi]} \left| -\cos(x) \right| = 1; \quad \omega_3(1{,}5) = 1{,}5(1{,}5 - 0{,}5\pi)(1{,}5 - \pi); \quad 3! = 6.$$

Таким чином, верхня оцінка $\left| R_2(1{,}5) \right| \le \frac{1}{6} \left| 1{,}5(1{,}5 - 0{,}5\pi)(1{,}5 - \pi) \right| = 0{,}029.$

За формулою (4.6) знаходимо $\sin(1{,}5) = \frac{-6(1{,}5 - \pi)}{\pi^2} = 0{,}998,$ а розрахунки на калькуляторі дають $\sin(1{,}5) = 0{,}997$. Отже, реальна абсолютна похибка $\left| R_2(1{,}5) \right| = \left| 0{,}997 - 0{,}998 \right| = 0{,}001.$

Приклад 4.2. Побудувати інтерполяційний поліном Лагранжа для функції, заданої у вигляді таблиці, та дістати його значення в точці $x = 4$.

x	1	2	3	5
y	1	5	14	81

Розв'язання. Підставляємо вхідні дані у формулу (4.6) і отримуємо:

$$P_3(x) = 1\frac{(x-2)(x-3)(x-5)}{(1-2)(1-3)(1-5)} + 5\frac{(x-1)(x-3)(x-5)}{(2-1)(2-3)(2-5)} + \\ + 14\frac{(x-1)(x-2)(x-5)}{(3-1)(3-2)(3-5)} + 81\frac{(x-1)(x-2)(x-3)}{(5-1)(5-2)(5-3)}.$$

Таким чином, $P_3(4) = 36{,}5$. Однак, не знаючи модуля четвертої похідної $\left| f^{\mathrm{IV}}(x) \right|$ на інтервалі [1,5], не можна оцінити похибку отриманого інтерполяційного значення.

Цей приклад показує, що для *формальної побудови інтерполяційного поліному* достатньо мати лише таблицю значень функції $f(x)$, а оцінка похибки потребує додаткової інформації про поведінку $f(x)$ та її похідних на інтервалі інтерполяції.

На рис. 4.2 зображена блок-схема алгоритму формальної інтерполяції функції в деякій точці z. Після реалізації даного алгоритму на ЕОМ результат розв'язання задачі прикладу 4.2 отриманий у вигляді

$$P_n(4{,}000) = 36{,}500.$$

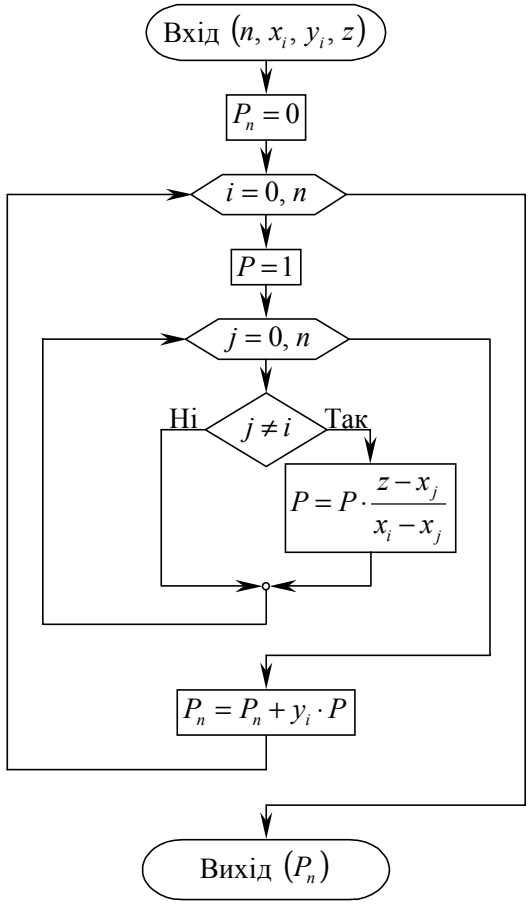

Вхід (n, x_i, y_i, z)
$P_n = 0$
$i = 0, n$
$P = 1$
$j = 0, n$
Ні
$j \neq i$
Так
$P = P \cdot \frac{z - x_j}{x_i - x_j}$
$P_n = P_n + y_i \cdot P$
Вихід (P_n)

Рис. 4.2

5. ОБРОБКА ЕКСПЕРИМЕНТАЛЬНИХ ДАНИХ ЗА ДОПОМОГОЮ ЕОМ

Аналіз різних процесів, які виникають у фізиці, техніці, економіці, призводить до необхідності виявлення найбільш суттєвих факторів, що впливають на досліджуваний процес. Крім того, виникає необхідність у виборі форм зв'язку між факторами та в оцінці параметрів отриманих рівнянь зв'язку.

Виявити різні фактори, які впливають на досліджуваний процес, у деяких випадках можливо тільки в результаті вимірів, тобто в процесі експерименту. З іншого боку, експеримент використовується і для перевірки адекватності ідеалізованої математичної моделі реальному процесу. Функціональні залежності, отримані в процесі експерименту у вигляді таблиць, прийнято називати *емпіричними*.

Аналітичну побудову емпіричних залежностей можна здійснити різними методами, серед яких найбільш поширеним є *метод найменших квадратів*.

5.1. Апроксимація функції методом найменших квадратів

Нехай у результаті вимірів у процесі експерименту отримана таблиця деякої функціональної залежності $y = f(x)$. Необхідно знайти формулу, що буде виражати цю залежність аналітично.

x	x_1	x_2	x_3	…	x_n
y	y_1	y_2	y_3	…	y_n

Можна, звичайно, застосувати метод інтерполяції, тобто побудувати інтерполяційний поліном Лагранжа $P_n(x)$, значення якого у вузлах $x_1, x_2, x_3, \ldots, x_n$ будуть точно співпадати з відповідними значеннями $y_1, y_2, y_3, \ldots, y_n$ із таблиці. Однак, точна відповідність значень у вузлах може і не означати того, що характери поведінки початкової та наближеної (інтерполяційної) функцій будуть співпадати. Більш того, точна

відповідність значень у вузлах виглядає зовсім невиправданою за умови, що значення функції $f(x)$ отримані в результаті вимірів, отже, є сумнівними.

Зараз поставимо задачу так, щоб із самого початку враховувався *характер* початкової функції. Необхідно знайти функцію *заданого вигляду*

$$y = F(x), \tag{5.1}$$

яка у вузлах $x_1, x_2, x_3, \ldots, x_n$ набуває значень $\tilde{y}_1, \tilde{y}_2, \tilde{y}_3, \ldots, \tilde{y}_n$, якомога ближчих до табличних $y_1, y_2, y_3, \ldots, y_n$.

Практично вигляд наближеної (*апроксимуючої*) функції (5.1) можна визначити наступним чином. Згідно з вхідною інформацією (за таблицею) будується *точковий графік* початкової функції $f(x)$, а потім проводиться плавна крива, яка за можливістю найкращим чином відображає характер розміщення точок (див. рис. 5.1). За отриманою в такий спосіб кривою встановлюється вигляд апроксимуючої функції (зазвичай з числа простих за виглядом аналітичних функцій).

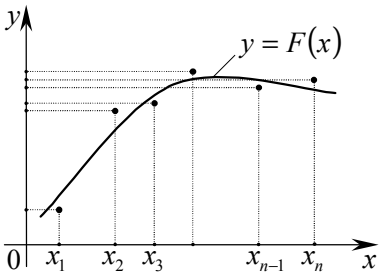


Рис. 5.1

Розглянемо один із поширених способів знаходження апроксимуючої функції $y = F(x)$.

Вимогу близькості табличних значень $y_1, y_2, y_3, \ldots, y_n$ та значень апроксимуючої функції $\tilde{y}_1, \tilde{y}_2, \tilde{y}_3, \ldots, \tilde{y}_n$ можна тлумачити таким чином. Сукупність значень функції з таблиці та сукупність значень функції $F(x)$ будемо розглядати як координати двох точок n-вимірного простору $M(y_1, y_2, y_3, \ldots, y_n)$ та $\tilde{M}(\tilde{y}_1, \tilde{y}_2, \tilde{y}_3, \ldots, \tilde{y}_n)$. Апроксимуюча функція $F(x)$ заданого вигляду визначається з умови мінімуму відстані між цими точками:

$$\sqrt{(y_1 - \tilde{y}_1)^2 + (y_2 - \tilde{y}_2)^2 + (y_3 - \tilde{y}_3)^2 + \ldots + (y_n - \tilde{y}_n)^2} = \min.$$

Значення цього кореня буде мінімальним, якщо підкореневий вираз наближатиметься до мінімуму. Скорочено це можна записати так:

$$\sum_{i=1}^{n} (y_i - \tilde{y}_i)^2 = \min. \tag{5.2}$$

Отже, задача знаходження апроксимуючої функції тепер може бути сформульована наступним чином. Для функції $f(x)$, заданої у вигляді таблиці, знайти функцію $F(x)$ заданого вигляду таким чином, щоб сума квадратів (5.2) була найменшою. Ця задача носить назву апроксимації функції $f(x)$ *методом найменших квадратів*.

В залежності від характеру точкового графіку, використовують наступні апроксимуючі функції:

1) $y = ax + b$ – лінійна;

2) $y = ax^2 + bx + c$ – параболічна;

3) $y = ax^m$ – степенева;

4) $y = ae^{mx}$ – показникова;

5) $y = \dfrac{1}{ax + b}$ – дробово-лінійна;

6) $y = a \ln x + b$ – логарифмічна;

7) $y = a\dfrac{1}{x} + b$ – гіперболічна;

8) $y = \dfrac{x}{ax + b}$ – дробово-раціональна.

Тут a, b, c, m – невідомі параметри. Коли вигляд функції $F(x)$ встановлено, задача апроксимації зводиться до визначення значень цих параметрів виходячи з формули (5.2).

5.2. Знаходження апроксимуючої функції у вигляді лінійної функції. Реалізація лінійної апроксимації на ЕОМ

Будемо шукати апроксимуючу функцію у вигляді $F(x, a, b) = ax + b$. Отже, $\tilde{y}_i = F(x_i, a, b)$, де $i = 1, 2, 3, \ldots, n$, а рівняння (5.2) запишеться у вигляді

$$\sum_{i=1}^{n} [y_i - F(x_i, a, b)]^2 = \Phi(a, b) = \min.$$

Для знаходження мінімуму функції $\Phi(a, b)$ скористаємося необхідною умовою існування екстремуму:

$$\left.\begin{aligned} \frac{\partial \Phi(a, b)}{\partial a} &= 0, \\ \frac{\partial \Phi(a, b)}{\partial b} &= 0, \end{aligned}\right\}$$

яка дозволяє записати

$$\left.\begin{aligned} -2\sum_{i=1}^{n}\left\{[y_i - F(x_i, a, b)]\frac{\partial F(x_i, a, b)}{\partial a}\right\} &= 0, \\ -2\sum_{i=1}^{n}\left\{[y_i - F(x_i, a, b)]\frac{\partial F(x_i, a, b)}{\partial b}\right\} &= 0. \end{aligned}\right\}$$

У нашому випадку $F(x_i, a, b) = ax_i + b$, $\dfrac{\partial F(x_i, a, b)}{\partial a} = x_i$, а $\dfrac{\partial F(x_i, a, b)}{\partial b} = 1$, отже,

$$\left.\begin{aligned} \sum_{i=1}^{n} (y_i x_i - a x_i^2 - b x_i) = 0, \\ \sum_{i=1}^{n} (y_i - a x_i - b) = 0. \end{aligned}\right\}$$

Після винесення за знак суми членів, які не залежать від індексу підсумовування i, отримаємо

$$\left.\begin{aligned} a\sum_{i=1}^{n} x_i^2 + b\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i x_i, \\ a\sum_{i=1}^{n} x_i + bn = \sum_{i=1}^{n} y_i. \end{aligned}\right\} \qquad (5.3)$$

Система (5.3) – СЛАР другого порядку відносно невідомих коефіцієнтів a та b. Якщо позначити

$$\left.\begin{aligned} S_1 = \sum_{i=1}^{n} x_i;\ S_2 = \sum_{i=1}^{n} x_i^2; \\ S_{10} = \sum_{i=1}^{n} y_i;\ S_{11} = \sum_{i=1}^{n} y_i x_i, \end{aligned}\right\} \qquad (5.4)$$

то легко дістати аналітичний розв'язок (5.3) у вигляді

$$\left.a = \frac{nS_{11} - S_1 S_{10}}{nS_2 - S_1^2};\ \ b = \frac{S_{10} - aS_1}{n}.\right\} \qquad (5.5)$$

Приклад 5.1. Для функції, заданої у вигляді таблиці, знайти апроксимуючу функцію вигляду $F(x) = ax + b$.

x	1	2	3	5
y	1,1	1,9	3,2	5

Розв'язання. За формулами (5.4) знаходимо: $S_1 = 1 + 2 + 3 + 5 = 11$, $S_2 = 1 + 4 + 9 + 25 = 39$, $S_{10} = 1{,}1 + 1{,}9 + 3{,}2 + 5 = 11{,}2$, $S_{11} = 1{,}1 + 3{,}8 + 9{,}6 + 25 = 39{,}5$.

За формулами (5.5) знаходимо:

$$a = \frac{4 \cdot 39{,}5 - 11 \cdot 11{,}2}{4 \cdot 39 - 121} = 0{,}994, \quad b = \frac{11{,}2 - 0{,}994 \cdot 11}{4} = 0{,}066.$$

Таким чином, $F(x) = 0{,}994x + 0{,}066$.

Зрозуміло, що значення знайденої у прикладі 5.1 функції $F(x)$ у вузлах x_i будуть відрізнятися від відповідних табличних значень y_i. Значення різниць $\varepsilon_i = y_i - F(x_i)$ називаються *відхиленнями* виміряних значень $f(x)$ від обчислених значень $F(x)$. Для знайденої емпіричної формули згідно з початковою таблицею можна дістати суму квадратів відхилень $\sigma = \sum_{i=1}^{n} \varepsilon_i^2$, а з двох різних наближень однієї й тієї ж табличної функції кращим слід вважати те, для якого σ має найменше значення.

На рис. 5.2 зображена блок-схема алгоритму обчислення коефіцієнтів a та b для лінійної апроксимації функції, заданої у вигляді таблиці. Зазначимо, що наведений алгоритм у певній мірі є універсальним, оскільки знаходження апроксимуючої функції у вигляді якоїсь елементарної функції з двома параметрами з підрозд. 5.1 іноді може бути зведено до знаходження параметрів лінійної функції. Покажемо, яким чином це можна зробити.

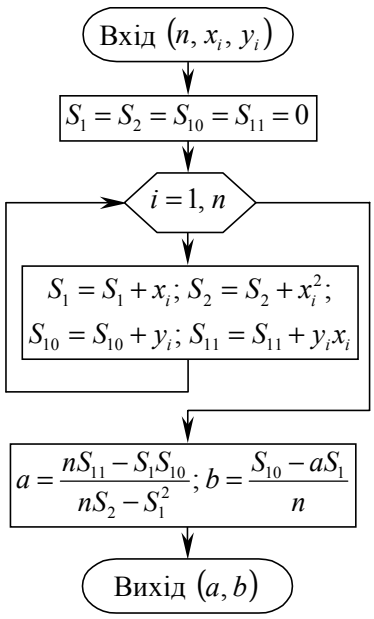


Рис. 5.2

Степенева функція $F(x) = ax^m$

Якщо припустити, що в початковій таблиці значення x_i та y_i додатні, можна злогарифмувати рівняння $F(x) = ax^m$ за умови $a > 0$:

$$\ln F(x) = m \ln x + \ln a.$$

За зроблених вище припущень, для знаходження наближеної функції у вигляді степеневої, алгоритм лінійної апроксимації використовується так:

1. на вхід алгоритму подаються n, $\ln x_i$, $\ln y_i$:
2. на виході отримуємо a та b;
3. у вираз для $F(x)$ підставляємо $m = a$, $a = e^b$.

Показникова функція $F(x) = ae^{mx}$

Злогарифмуємо $F(x) = ae^{mx}$ при $a > 0$:

$$\ln F(x) = mx + \ln a.$$

Таким чином, для знаходження наближеної функції у вигляді показникової при $y_i > 0$ алгоритм лінійної апроксимації використовується так:

1. на вхід алгоритму подаються n, x_i, $\ln y_i$;
2. на виході отримуємо a та b;
3. у вираз для $F(x)$ підставляємо $m = a$, $a = e^b$.

Дробово-лінійна функція $F(x) = \dfrac{1}{ax + b}$

Запишемо функцію $F(x)$ у вигляді

$$\frac{1}{F(x)} = ax + b.$$

Таким чином, для знаходження наближеної функції у вигляді дробово-лінійної при $y_i \neq 0$ алгоритм на рис. 5.2 використовується так:

1. на вхід алгоритму подаються n, x_i, $1/y_i$;
2. на виході отримуємо a та b, які потім підставляємо у вираз $F(x) = 1/(ax + b)$.

Логарифмічна функція $F(x) = a \ln x + b$

Якщо припустити, що в початковій таблиці значення $x_i \geq 0$, для знаходження наближеної функції у вигляді логарифмічної, алгоритм лінійної апроксимації використовується так:

1. на вхід алгоритму подаються n, $\ln x_i$, y_i;
2. на виході отримуємо a та b, які потім підставляємо у вираз $F(x) = a \ln x + b$.

Гіперболічна функція $F(x) = a\dfrac{1}{x} + b$

Для знаходження наближеної функції у вигляді гіперболи алгоритм лінійної апроксимації при $x_i \neq 0$ використовується так:

1. на вхід алгоритму подаються n, $1/x_i$, y_i;
2. на виході отримуємо a та b, які потім підставляємо у вираз для $F(x)$.

Дробово-раціональна функція $F(x) = \dfrac{x}{ax + b}$

Запишемо функцію $F(x)$ у вигляді

$$\frac{1}{F(x)} = b\frac{1}{x} + a.$$

Таким чином, для знаходження наближеної функції у вигляді дробово-раціональної, при $x_i, y_i \neq 0$ алгоритм на рис. 5.2 використовується так:

1. на вхід алгоритму подаються n, $1/x_i$, $1/y_i$;
2. на виході отримуємо a та b;
3. у вираз для $F(x)$ підставляємо $b = a$, $a = b$.

Приклад 5.2. Для функції, заданої табличним способом, знайти найкращу апроксимуючу функцію серед елементарних функцій із двома параметрами.

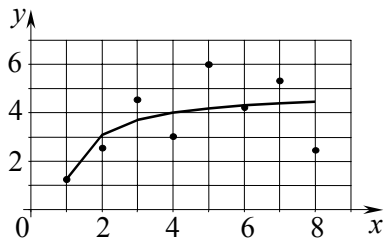
x	1	2	3	4	5	6	7	8
y	1,25	2,56	4,56	3,04	6,01	4,23	5,34	2,45

Побудувати точковий графік початкової функції та графік вибраної апроксимуючої функції.

Розв'язання. Після реалізації алгоритму лінійної апроксимації у вигляді підпрограми та відповідних викликах такої підпрограми в основній програмі були отримані наступні результати:

Лінійна	$a = 0{,}2890$	$b = 2{,}3793$	$\sigma = 14{,}834161905$;
Степенева	$m = 0{,}4992$	$a = 1{,}7128$	$\sigma = 14{,}062208574$;
Показникова	$m = 0{,}1053$	$a = 2{,}0672$	$\sigma = 17{,}063358711$;
Дробово-лінійна	$a = -0{,}0461$	$b = 0{,}5495$	$\sigma = 24{,}251111878$;
Логарифмічна	$a = 1{,}3603$	$b = 1{,}8768$	$\sigma = 11{,}934597612$;
Гіперболічна	$a = -3{,}6677$	$b = 4{,}9261$	$\sigma = 10{,}216915042$;
Дробово-раціональна	$a = 0{,}1334$	$b = 0{,}6145$	$\sigma = 13{,}298537713$.

Оскільки мінімальне значення σ відповідає гіперболічній функції, як апроксимуючу вибираємо функцію $F(x) = -3{,}6677/x + 4{,}9261$.



5.3. Знаходження апроксимуючої функції у вигляді квадратного тричлену. Реалізація квадратичної апроксимації на ЕОМ

Якщо шукати апроксимуючу функцію у вигляді функції з трьома параметрами $F(x, a, b, c) = ax^2 + bx + c$, то $\tilde{y}_i = F(x_i, a, b, c)$, а рівняння (5.2) запишеться у вигляді

$$\sum_{i=1}^{n} [y_i - F(x_i, a, b, c)]^2 = \Phi(a, b, c) = \min.$$

Необхідна умова існування екстремуму для $\Phi(a, b, c)$

$$\left.\begin{aligned} \frac{\partial \Phi(a, b, c)}{\partial a} = 0, \frac{\partial \Phi(a, b, c)}{\partial b} = 0, \\ \frac{\partial \Phi(a, b, c)}{\partial c} = 0, \end{aligned}\right\}$$

дозволяє записати:

$$\left.\begin{aligned} -2\sum_{i=1}^{n}\left\{[y_i - F(x_i, a, b, c)]\frac{\partial F(x_i, a, b, c)}{\partial a}\right\} = 0, \\ -2\sum_{i=1}^{n}\left\{[y_i - F(x_i, a, b, c)]\frac{\partial F(x_i, a, b, c)}{\partial b}\right\} = 0, \\ -2\sum_{i=1}^{n}\left\{[y_i - F(x_i, a, b, c)]\frac{\partial F(x_i, a, b, c)}{\partial c}\right\} = 0. \end{aligned}\right\}$$

В нашому випадку $F(x_i, a, b, c) = ax_i^2 + bx_i + c$, $\frac{\partial F(x_i, a, b, c)}{\partial a} = x_i^2$, $\frac{\partial F(x_i, a, b, c)}{\partial b} = x_i$, $\frac{\partial F(x_i, a, b, c)}{\partial c} = 1$, отже,

$$\left.\begin{aligned} \sum_{i=1}^{n}(y_i x_i^2 - ax_i^4 - bx_i^3 - cx_i^2) = 0, \\ \sum_{i=1}^{n}(y_i x_i - ax_i^3 - bx_i^2 - cx_i) = 0, \\ \sum_{i=1}^{n}(y_i - ax_i^2 - bx_i - c) = 0. \end{aligned}\right\}$$

Після винесення за знак суми членів, які не залежать від індексу підсумовування i, отримаємо

$$\left.\begin{aligned} a\sum_{i=1}^{n} x_i^4 + b\sum_{i=1}^{n} x_i^3 + c\sum_{i=1}^{n} x_i^2 &= \sum_{i=1}^{n} y_i x_i^2, \\ a\sum_{i=1}^{n} x_i^3 + b\sum_{i=1}^{n} x_i^2 + c\sum_{i=1}^{n} x_i &= \sum_{i=1}^{n} y_i x_i, \\ a\sum_{i=1}^{n} x_i^2 + b\sum_{i=1}^{n} x_i + cn &= \sum_{i=1}^{n} y_i. \end{aligned}\right\} \qquad (5.6)$$

Система (5.6) – СЛАР третього порядку відносно невідомих коефіцієнтів a, b та c. Якщо позначити

$$\left.\begin{aligned} S_1 = \sum_{i=1}^{n} x_i;\ S_2 = \sum_{i=1}^{n} x_i^2;\ S_3 = \sum_{i=1}^{n} x_i^3;\ S_4 = \sum_{i=1}^{n} x_i^4; \\ S_{10} = \sum_{i=1}^{n} y_i;\ S_{11} = \sum_{i=1}^{n} y_i x_i;\ S_{12} = \sum_{i=1}^{n} y_i x_i^2, \end{aligned}\right\} \qquad (5.7)$$

то можна отримати аналітичний розв'язок (5.6) за формулами Крамера у вигляді

$$\left.\begin{aligned} \Delta &= S_4(S_2 n - S_1^2) - S_3(S_3 n - S_1 S_2) + S_2(S_1 S_3 - S_2^2); \\ \Delta_1 &= S_{12}(S_2 n - S_1^2) - S_3(S_{11} n - S_1 S_{10}) + S_2(S_1 S_{11} - S_2 S_{10}); \\ \Delta_2 &= S_4(S_{11} n - S_1 S_{10}) - S_{12}(S_3 n - S_1 S_2) + S_2(S_3 S_{10} - S_2 S_{11}); \\ \Delta_3 &= S_4(S_2 S_{10} - S_1 S_{11}) - S_3(S_3 S_{10} - S_2 S_{11}) + S_{12}(S_1 S_3 - S_2^2); \\ a &= \frac{\Delta_1}{\Delta};\ b = \frac{\Delta_2}{\Delta};\ c = \frac{\Delta_3}{\Delta}. \end{aligned}\right\} \qquad (5.8)$$

Алгоритм обчислення коефіцієнтів a, b та c за формулами (5.8) з урахуванням (5.7) зображено на рис. 5.3.

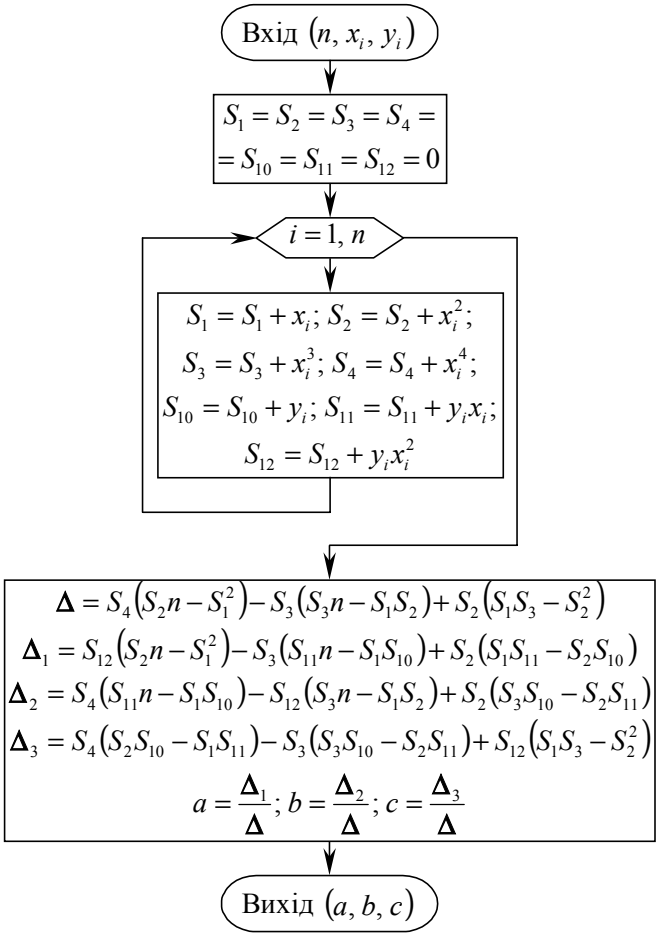


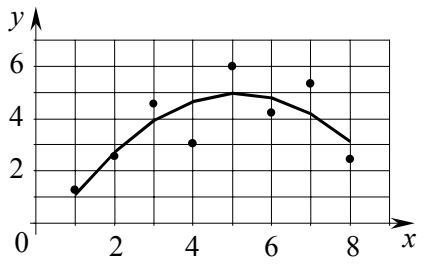
Рис. 5.3

Приклад 5.3. Для функції прикладу 5.2, заданої у вигляді таблиці, знайти апроксимуючу функцію вигляду $F(x) = ax^2 + bx + c$. Побудувати точковий графік початкової функції та графік апроксимуючої функції.

Розв'язання. Після реалізації алгоритму квадратичної апроксимації на ЕОМ були отримані наступні результати:

$$F(x) = -0{,}2251x^2 + 2{,}3151x - 0{,}9975;$$

$$\sigma = 6{,}3201595238.$$



6. НАБЛИЖЕНЕ РОЗВ'ЯЗАННЯ ЗВИЧАЙНИХ ДИФЕРЕНЦІАЛЬНИХ РІВНЯНЬ ПЕРШОГО ПОРЯДКУ. РОЗВ'ЯЗАННЯ ЗАДАЧІ КОШІ ЗА ДОПОМОГОЮ ЕОМ

6.1. Загальні поняття про диференціальні рівняння. Задача Коші

Для опису різноманітних фізичних, технічних або економічних задач використовуються математичні моделі. Більшість математичних моделей зображують диференціальні рівняння чи системи таких рівнянь. Так, математичною моделлю коливання матеріальної точки є звичайні диференціальні рівняння. Однак якщо описувати коливання не точки, а тіла, то математичною моделлю цього фізичного процесу будуть вже диференціальні рівняння з частинними похідними.

Рівняння, в якому невідома функція знаходиться під знаком похідної чи диференціала, називається *диференціальним рівнянням*.

Якщо невідома функція, що входить у диференціальне рівняння, залежить тільки від однієї незалежної змінної, то таке диференціальне рівняння називається *звичайним*. Наприклад, звичайними диференціальними рівняннями є $t^2 \dfrac{d^2 y(t)}{dt^2} = 2y(t)$ або $y''(x) - 2y'(x) + x^2 = 1$.

Якщо ж невідома функція, яка входить у диференціальне рівняння, є функцією двох або більше незалежних змінних, то таке диференціальне рівняння називається *рівнянням із частинними похідними*. Прикладами диференціальних рівнянь із частинними похідними є $\dfrac{\partial^2 u(x,t)}{\partial t^2} = a^2 \dfrac{\partial^2 u(x,t)}{\partial x^2}$ або

$$D_1 \frac{\partial^4 w(x,y)}{\partial x^4} + 2D_3 \frac{\partial^4 w(x,y)}{\partial x^2 \partial y^2} + D_2 \frac{\partial^4 w(x,y)}{\partial y^4} = p(x,y).$$

Порядок диференціального рівняння визначається за найвищим порядком похідної (чи диференціала), що входить у рівняння. Так, наприклад, рівняння $t^2 \frac{d^2 y(t)}{dt^2} = 2y(t)$, $y''(x) - 2y'(x) + x^2 = 1$ та $\frac{\partial^2 u(x,t)}{\partial t^2} = a^2 \frac{\partial^2 u(x,t)}{\partial x^2}$ є рівняннями другого порядку, а рівняння $D_1 \frac{\partial^4 w(x,y)}{\partial x^4} + 2D_3 \frac{\partial^4 w(x,y)}{\partial x^2 \partial y^2} + D_2 \frac{\partial^4 w(x,y)}{\partial y^4} = p(x,y)$ – четвертого порядку.

У даному розділі розглядаються тільки звичайні диференціальні рівняння першого порядку, які містять у загальному випадку незалежну змінну x, невідому функцію $y(x)$ та її похідну (або диференціал) першого порядку $y'(x)$ (або $\frac{dy(x)}{dx}$):

$$y' = f(x, y). \tag{6.1}$$

Розв'язком чи інтегралом диференціального рівняння (6.1) називається будь-яка диференційована функція $y = \varphi(x)$, що задовольняє даному рівнянню, тобто така, після підстановки якої в рівняння (6.1) воно перетворюється в тотожність. Графік функції $y = \varphi(x)$ називається *інтегральною кривою* цього рівняння.

Розв'язок диференціального рівняння (6.1), який містить одну незалежну довільну сталу C, називається *загальним розв'язком* або *загальним інтегралом* цього рівняння. Загальний розв'язок з геометричної точки зору зображує *сукупність інтегральних кривих* (див. рис. 6.1).

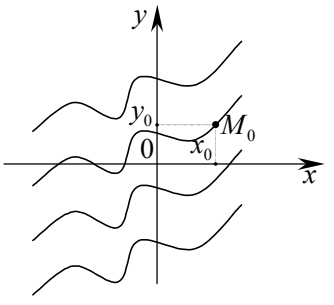


Рис. 6.1

Частинний розв'язок диференціального рівняння (6.1) можна отримати із загального при певному значенні сталої C.

Значення сталої C можна визначити за допомогою так званих *початкових умов*. Задача з початковими умовами називається *задачею Коші* і ставиться таким чином: знайти розв'язок $y = \varphi(x)$ рівняння (6.1), який задовольняє додатковій умові – при $x = x_0$, $y = \varphi(x_0) = y_0$. Геометрично задача Коші для рівняння (6.1) полягає в тому, що з усієї множини інтегральних кривих (рис. 6.1) потрібно знайти ту, яка проходить через точку M_0 з координатами $x = x_0$, $y = y_0$.

Приклад 6.1. Розв'язати задачу Коші:

$$\left.\begin{array}{l} y'(x) = x; \\ x_0 = 0;\ y_0 = 1. \end{array}\right\}$$

Розв'язання. Знайдемо загальний розв'язок: $y' = x \Rightarrow \dfrac{dy}{dx} = x \Rightarrow dy = xdx \Rightarrow$

$\Rightarrow \int dy = \int x\,dx \Rightarrow y = 0{,}5x^2 + C$. Якщо тепер у загальний розв'язок підставити початкові дані, то отримаємо $C = 1$. Отже, частинний розв'язок, який задовольняє початковій умові, зображує функцію $y = 0{,}5x^2 + 1$. Геометрично це означає, що з усієї множини парабол, які зображують загальний розв'язок, вибираємо одну єдину, що проходить через точку $M_0(0,1)$.

Методами точного інтегрування можна розв'язати порівняно невелику частину диференціальних рівнянь, які зустрічаються на практиці. Тому доводиться використовувати наближені методи, що, в залежності від форми зображення розв'язку, можна поділити на дві групи:

1. *аналітичні методи* – дають наближений розв'язок диференціального рівняння у вигляді аналітичного виразу;

2. *чисельні методи* – дають наближений розв'язок у вигляді таблиці.

У даному розділі розглянемо два наближених чисельних методи розв'язання диференціальних рівнянь вигляду (6.1) – це методи Ейлера і Рунге–Кутта.

6.2. Розв'язання задачі Коші методом Ейлера. Реалізація методу Ейлера на ЕОМ

Метод Ейлера є порівняно грубим чисельним методом розв'язання диференціальних рівнянь і застосовується в основному для орієнтовних розрахунків. Однак ідеї, покладені в основу даного методу, виявилися базовими для ряду інших, більш точних методів.

Отже, нехай потрібно розв'язати задачу Коші на деякому відрізку $[a, b]$:

$$\left.\begin{array}{l} y' = f(x, y); \\ x = x_0;\ y(x_0) = y_0. \end{array}\right\} \qquad (6.2)$$

Розіб'ємо відрізок $[a, b]$ на n рівних частин (рис. 6.2) і одержимо послідовність точок $x_0, x_1, x_2, \ldots, x_n$, тобто $x_i = a + ih$, де $i = 0, 1, 2, \ldots, n$, а $h = (b - a)/n$ – крок інтегрування.

Виберемо деякий k-й інтервал $[x_k, x_{k+1}]$ і зінтегруємо на ньому диференціальне рівняння (6.2):

$$\int_{x_k}^{x_{k+1}} y'(x)\,dx = \int_{x_k}^{x_{k+1}} f(x, y)\,dx \Rightarrow \int_{x_k}^{x_{k+1}} f(x, y)\,dx =$$

$$= y(x)\Big|_{x_k}^{x_{k+1}} = y(x_{k+1}) - y(x_k).$$

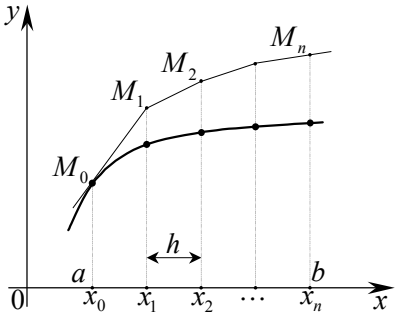


Рис. 6.2

Позначаючи $y(x_k) = y_k$ та $y(x_{k+1}) = y_{k+1}$ дістанемо

$$y_{k+1} = y_k + \int_{x_k}^{x_{k+1}} f(x, y)\,dx. \tag{6.3}$$

Коли припустити, що в інтегралі рівняння (6.3) підінтегральна функція $f(x, y)$ на відрізку $[x_k, x_{k+1}]$ не змінюється і дорівнює початковому значенню в точці x_k, тобто $f(x, y) = \text{const} = f(x_k, y_k)$, то дістанемо

$$y_{k+1} = y_k + f(x_k, y_k) \int_{x_k}^{x_{k+1}} dx = y_k + f(x_k, y_k)(x_{k+1} - x_k).$$

Оскільки $x_{k+1} - x_k = h$, а згідно з (6.2) $f(x_k, y_k) = y'_k$, остаточно отримаємо

$$y_{k+1} = y_k + hy'_k. \tag{6.4}$$

Рекурсивна формула (6.4) називається *формулою методу Ейлера*, яка з урахуванням початкової умови дозволяє побудувати таблицю наближених значень шуканої функції $y = \varphi(x)$ на відрізку $[a, b]$ з кроком h.

Якщо формулу (6.4) записати у вигляді $y_{k+1} - y_k = y'_k(x_{k+1} - x_k)$, то можна побачити, що на відрізку $[x_k, x_{k+1}]$ інтегральна крива приблизно заміняється прямолінійним відрізком, який виходить із точки $M_k(x_k, y_k)$ з кутовим коефіцієнтом $y'_k = f(x_k, y_k)$. Таким чином, замість істинної інтегральної кривої $y = \varphi(x)$ ми одержуємо ламану лінію з вершинами в точках $M_0(x_0, y_0)$, $M_1(x_1, y_1)$, $M_2(x_2, y_2)$, …, $M_n(x_n, y_n)$. Перша ланка цієї ламаної дотикається істинної інтегральної кривої в точці M_0 (див. рис. 6.2), а з кожним наступним кроком похибка систематично зростає.

Приклад 6.2. Методом Ейлера розв'язати задачу Коші $\left.\begin{matrix} y' = xy; \\ x_0 = 0;\ y_0 = 1 \end{matrix}\right\}$ на відрізку [0,1] з кроком $h = 0{,}25$. Оцінити максимальну похибку.

Розв'язання. Подане диференціальне рівняння має точний аналітичний розв'язок:

$$y' = xy \Rightarrow \frac{dy}{y} = x\,dx \Rightarrow \int \frac{dy}{y} = \int x\,dx \Rightarrow \ln|y| = 0{,}5x^2 + C.$$

Якщо тепер у загальний розв'язок підставити початкові дані, то одержимо $\ln 1 = C \Rightarrow C = 0$. Отже, частинний розв'язок, що задовольняє початковій умові, має вигляд: $y = e^{0{,}5x^2}$.

Тепер обчислимо за формулою Ейлера наближені значення частинного розв'язку і порівняємо ці значення з точними. Розрахунок зручно вести за допомогою наступної таблиці:

Наближений розв'язок				Точний розв'язок		Похибка
k	x_k	y_k	$hy'_k = hx_k y_k$	x_k	$y_k = e^{0{,}5x_i^2}$	δ, %
0	0	1 $y_{k+1} = y_k + hy'_k$	0	0	1	0
1	0,25	1	0,0625	0,25	1,0317	3,073
2	0,5	1,0625	0,1328	0,5	1,1331	6,231
3	0,75	1,1953	0,2241	0,75	1,3248	9,775
4	1	1,4194		1	1,6487	**13,908**

Для оцінки похибки методу Ейлера існує аналітична формула, яку можна знайти в літературі, присвяченій чисельним методам. Однак вона має в основному теоретичне значення, тому тут не наводиться. Для досягнення припустимої похибки ε, на практиці, як правило, застосовують метод подвійного перерахунку (застосування його к чисельному інтегруванню викладено в підрозд. 1.5). Розрахунок у рамках методу Ейлера ведеться двічі: при поділі відрізка [a, b] спочатку на n частин, а потім на $2n$ частин. Обчислення припиняються, коли виконується умова $|y(x_{2n}) - y(x_n)| \le \varepsilon$. Зазначимо, що оскільки подвійний перерахунок ведеться без реальної оцінки похибки, то після виконання умови $|y(x_{2n}) - y(x_n)| \le \varepsilon$ дістається деяке асимптотичне значення[2] $\varphi(b)$, точність якого невідома.

[2] Під асимптотичним тут слід розуміти таке значення інтегралу диференційного рівняння в точці b, яке (при малому значенні ε), не можна суттєво уточнити за допомогою наступного збільшення кількості n розбивок відрізка інтегрування на рівні частини.

Алгоритм розв'язання задачі Коші (6.2) на відрізку $[x_0, b]$ з кроком h у рамках методу Ейлера наведено на рис. 6.3. Метод подвійного перерахунку на базі даного алгоритму можна реалізувати згідно з блок-схемою, зображеною на рис. 6.4.

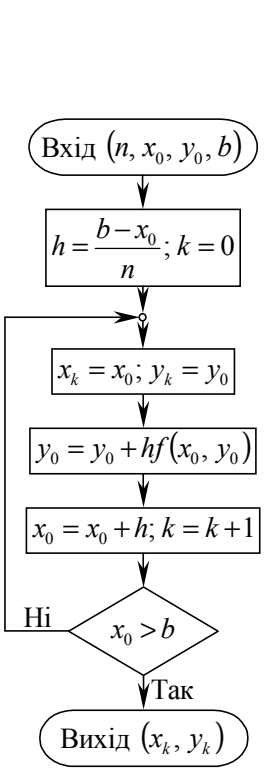


Рис. 6.3

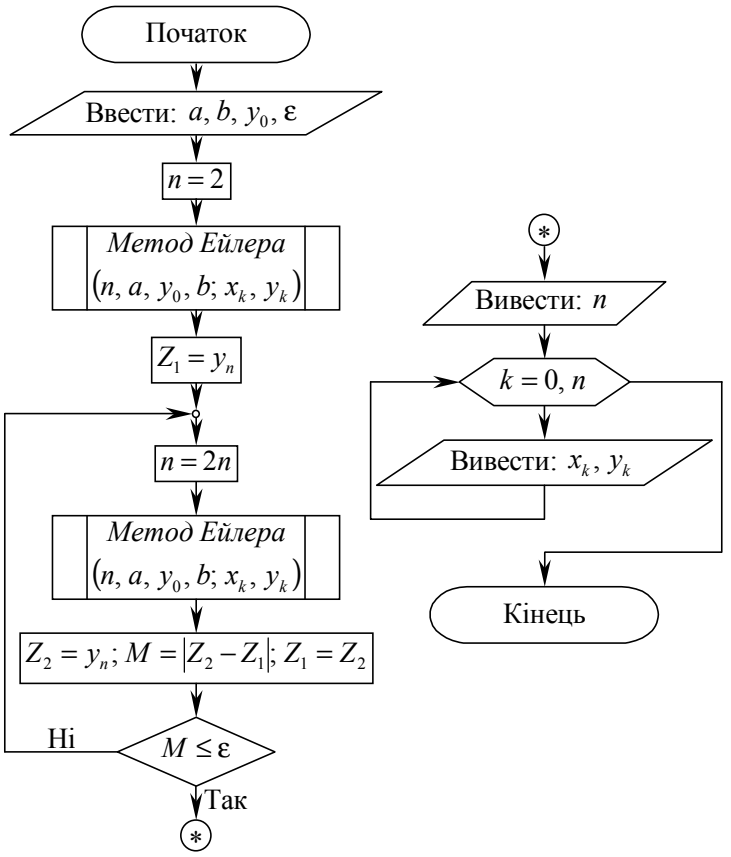


Рис. 6.4

Приклад 6.3. Методом Ейлера розв'язати задачу Коші з прикладу 4.2 із заданою асимптотичною точністю $\varepsilon = 10^{-2}$.

Розв'язання. Після реалізації алгоритму подвійного перерахунку, зображеного на рис. 6.4, одержимо таблицю для шуканої функції.

k	x_k	y_k
0	0,00	1,00
…	…	…
32	0,25	1,03
…	…	…
64	0,50	1,13
…	…	…
96	0,75	1,32
…	…	…
128	1,00	1,64

6.3. Розв'язання задачі Коші методом Рунге–Кутта. Реалізація методу Рунге–Кутта на ЕОМ

Метод Рунге–Кутта є одним із методів підвищеної точності та має багато спільного з методом Ейлера.

Нехай на відрізку $[a, b]$ потрібно знайти чисельний розв'язок задачі Коші (6.2). Розіб'ємо відрізок $[a, b]$ на n рівних частин точками $x_i = a + ih$, де $i = 0, 1, 2, \ldots, n$, $h = (b - a)/n$. У методі Рунге–Кутта, як і в методі Ейлера, значення шуканої функції $y = \varphi(x)$ послідовно обчислюються за формулою (6.4).

Якщо позначити $hy'_k = y_{k+1} - y_k = \Delta y_k$, то формула (6.4) запишеться у вигляді

$$y_{k+1} = y_k + \Delta y_k. \tag{6.5}$$

Розкладемо тепер шукану функцію $y(x)$ у ряд Тейлора в околі точки з координатою x_k:

$$y(x) = y(x_k) + \frac{y'(x_k)}{1!}(x - x_k) + \frac{y''(x_k)}{2!}(x - x_k)^2 + \frac{y'''(x_k)}{3!}(x - x_k)^3 + \\ + \frac{y^{\mathrm{IV}}(x_k)}{4!}(x - x_k)^4 + \ldots + \frac{y^n(x_k)}{n!}(x - x_k)^n + \frac{y^{n+1}(\xi)}{(n+1)!}(x - x_k)^{n+1}, \tag{6.6}$$

де x – будь-яке значення аргументу з околу точки x_k, причому $x \neq x_k$, а ξ – точка, розташована між x та x_k.

Якщо в околі точки x_k взяти точку x_{k+1} і, з огляду на те, що $x_{k+1} - x_k = h$, обмежитися членами ряду (6.6) до h^4 включно, дістанемо

$$\Delta y_k = y'_k h + \frac{h^2}{2} y''_k + \frac{h^3}{6} y'''_k + \frac{h^4}{24} y^{\mathrm{IV}}_k, \tag{6.7}$$

де похідні $y'_k, y''_k, y'''_k, y^{\mathrm{IV}}_k$ визначаються послідовним диференціюванням рівняння $y'_k = f(x_k, y_k)$.

Оцінка похибки (6.7) випливає з формули залишкового члена відрізку ряду Тейлора (6.6): $\max\limits_{x_k \le x \le x_{k+1}} \left|\varphi(x) - y(x)\right| \le \dfrac{h^5}{120} \max\limits_{x_k \le x \le x_{k+1}} \left|y^{\mathrm{V}}(x)\right|$. Таким чином, похибка методу Рунге–Кутта на кожному кроці інтегрування це величина порядку $h^5 - O(h^5)$.

Легко помітити, що коли обмежитися членами ряду (6.6) до h включно, то дістанемо формулу методу Ейлера (6.4). Зрозуміло, що похибка методу Ейлера на кожному кроці інтегрування є величина $O(h^2)$.

Замість безпосередніх обчислень за формулою (6.7) у методі Рунге–Кутта вибираються чотири числа:

$$\left.\begin{aligned} K_1^k = hf(x_k, y_k);\ K_2^k = hf\left(x_k + \frac{h}{2}, y_k + \frac{K_1^k}{2}\right);\\ K_3^k = hf\left(x_k + \frac{h}{2}, y_k + \frac{K_2^k}{2}\right);\ K_4^k = hf(x_k + h, y_k + K_3^k). \end{aligned}\right\} \qquad (6.8)$$

Можна довести: якщо числам $K_1^k, K_2^k, K_3^k, K_4^k$ надати відповідну вагу $\frac{1}{6}, \frac{1}{3}, \frac{1}{3}$, і $\frac{1}{6}$, то середньозважене цих чисел, тобто дорівнює значенню Δy_k, яке обчислюється за формулою (6.7).

Таким чином, обчислення таблиці наближених значень розв'язку задачі Коші у рамках методу Рунге–Кутта здійснюється з урахуванням (6.8) за рекурсивною формулою:

$$y_{k+1} = y_k + \frac{1}{6}(K_1^k + 2K_2^k + 2K_3^k + K_4^k). \qquad (6.9)$$

Відзначимо, що метод Рунге–Кутта на всьому відрізку інтегрування $[a, b]$ має точність $O(h^4)$. Оцінити реальну похибку цього методу достатньо важко, тому (так само як і в методі Ейлера) застосовують метод подвійного перерахунку для досягнення відповідної точності. Розрахунок у рамках методу Рунге–Кутта ведеться двічі: при поділі відрізка $[a, b]$ спочатку на n частин, а потім на $2n$ частин. Обчислення припиняються тоді, коли виконується умова $|y(x_{2n}) - y(x_n)| \le 15\varepsilon$.

Приклад 6.3. Методом Рунге–Кутта розв'язати задачу Коші з прикладу 6.2.

Розв'язання. Обчислення за формулою (6.9) з урахуванням (6.8) зобразимо у вигляді наступної таблиці:

k	x_k	y_k	K_1^k	K_2^k	K_3^k	K_4^k	$\frac{1}{6}\left(K_1^k + 2K_2^k + 2K_3^k + K_4^k\right)$
0	0,0000	1	0	0,0313	0,0317	0,0645	0,0317
		$y_{k+1} = y_k + \Delta y_k$					
1	0,2500	1,0317	0,0645	0,0997	0,1014	0,1416	0,1014
2	0,5000	1,1331	0,1416	0,1881	0,1918	0,2484	0,1916
3	0,7500	1,3248	0,2484	0,3170	0,3245	0,4123	0,3239
4	1,0000	1,6487	0,4122	0,5217	0,5371	0,6831	

Алгоритм розв'язання задачі Коші (6.2) на відрізку $[x_0, b]$ з кроком h у рамках методу Рунге–Кутта наведено на рис. 6.5. Метод подвійного перерахунку на базі даного алгоритму реалізується згідно з блок-схемою, зображеною на рис. 6.6.

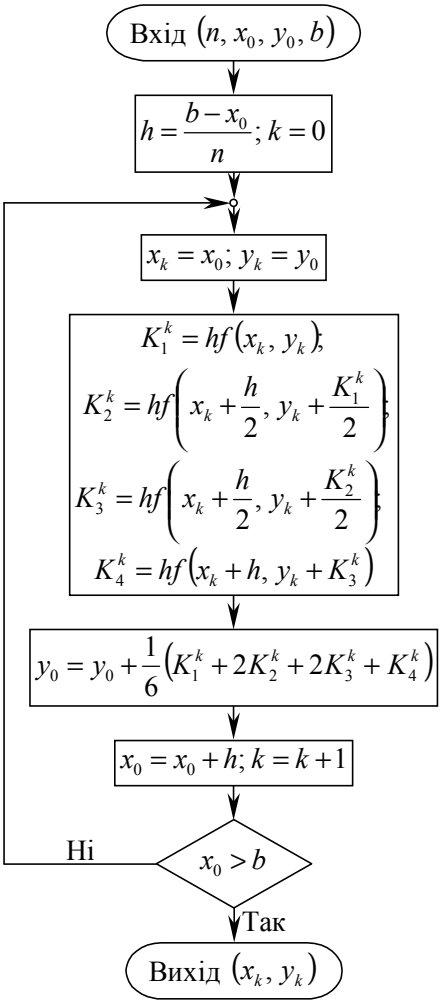


Рис. 6.5

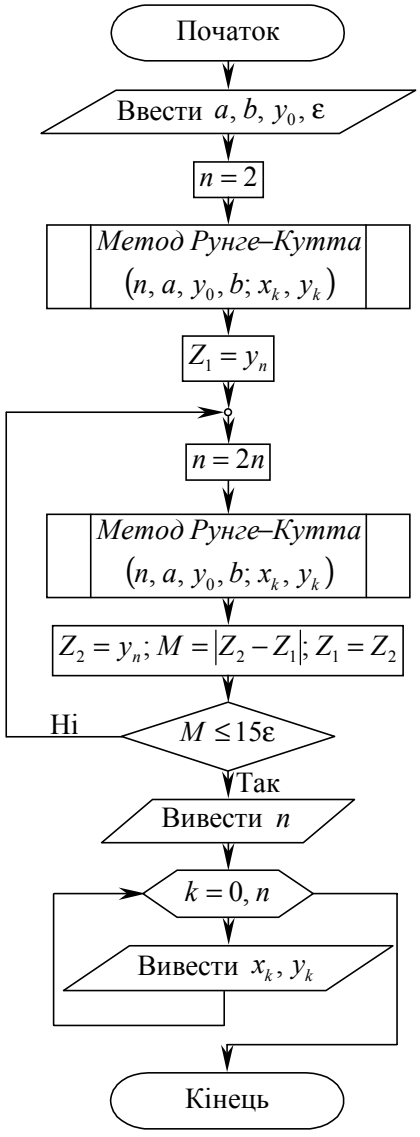


Рис. 6.6

рис. 6.6, одержимо наступну таблицю розв'язку задачі Коші з прикладу 4.2 із точністю $\varepsilon = 10^{-5}$:

k	x_k	y_k
0	0,00000	1,00000
1	0,12500	1,00784
2	0,25000	1,03174
3	0,37500	1,07284
4	0,50000	1,13315
5	0,62500	1,21569
6	0,75000	1,32478
7	0,87500	1,46640
8	1,00000	1,64872

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ ПО РОЗДІЛУ ДРУГОМУ

1. Назвіть задачі, які потребують обчислення значення визначеного інтегралу. Запишіть формулу Ньютона–Лейбніца для точного інтегрування.

2. Чому доводиться використовувати методи наближеного інтегрування?

3. Який геометричний зміст визначеного інтегралу?

4. Виведіть формулу методу лівих прямокутників.

5. Наведіть алгоритм обчислення значення визначеного інтегралу за методом лівих прямокутників.

6. Виведіть формулу методу правих прямокутників.

7. Наведіть алгоритм обчислення значення визначеного інтегралу за методом правих прямокутників.

8. Виведіть формулу методу трапецій. Як оцінюється похибка інтегрування в рамках методу трапецій?

9. Наведіть алгоритм обчислення значення визначеного інтегралу за методом трапецій.

10. Завдяки яким обставинам будується узагальнений алгоритм обчислення значення визначеного інтегралу методами лівих прямокутників, правих прямокутників і трапецій?

11. Виведіть формулу методу Сімпсона. Як оцінюється похибка інтегрування в рамках методу Сімпсона?

12. Наведіть алгоритм обчислення значення визначеного інтегралу за методом Сімпсона.

13. Який із відомих вам методів наближеного інтегрування має найбільшу точність і чому?

14. Якими способами можна обчислити значення визначеного інтегралу з заданим наперед ступенем точності?

15. Наведіть алгоритм методу подвійного перерахунку для обчислення значення визначеного інтегралу із заданою точністю.

16. Які лінійні та нелінійні рівняння Ви знаєте? Чим лінійне алгебраїчне рівняння відрізняється від нелінійного?

17. Чому виникає необхідність використовувати наближені методи для знаходження коренів нелінійного рівняння?

18. Охарактеризуйте етапи наближеного розв'язання нелінійних рівнянь?

19. Які властивості має відрізок ізоляції кореня? Якими способами можна його знайти?

20. Якими способами можна здійснити графічне відокремлення кореня нелінійного рівняння?

21. На базі якого класичного алгоритму здійснюється обчислювальне відокремлення коренів? Зобразіть блок-схему алгоритму обчислювального відокремлення коренів.

22. Дайте математичний опис методу половинного ділення. Як оцінюється похибка кореня нелінійного рівняння, уточненого в рамках цього методу?

23. Наведіть алгоритм методу половинного ділення і дайте йому геометричну інтерпретацію.

24. Запишіть розрахункові формули методу простої ітерації для уточнення кореня нелінійного рівняння. Охарактеризуйте збіжні та розбіжні ітераційні процеси уточнення кореня.

25. Наведіть алгоритм методу ітерацій для розв'язання нелінійного рівняння з заданим ступенем точності.

26. Виведіть розрахункові формули методу хорд. Як оцінюється похибка кореня нелінійного рівняння, уточненого в рамках цього методу?

27. Наведіть алгоритм методу хорд для розв'язання нелінійного рівняння з заданим ступенем точності.

28. Виведіть розрахункові формули методу дотичних. Як оцінюється похибка кореня нелінійного рівняння, уточненого в рамках цього методу?

29. Наведіть алгоритм методу дотичних для розв'язання нелінійного рівняння з заданим ступенем точності.

30. На базі алгоритмів методу хорд і дотичних самостійно розробіть алгоритм комбінованого методу хорд і дотичних враховуючи, що похибка зазначеного методу визначається аналогічно похибці методу половинного ділення.

31. Назвіть задачі, які призводять до необхідності розв'язання СЛАР. Чому виникає потреба у використанні ЕОМ для знаходження коренів СЛАР?

32. На які етапи розпадається процес розв'язання СЛАР методом Гаусса?

33. Які елементарні перетворення покладені в основу прямого ходу методу Гаусса?

34. Запишіть розрахункові формули для знаходження коефіцієнтів при невідомих і правих частин СЛАР трикутного вигляду.

35. Охарактеризуйте етапи алгоритму прямого ходу методу Гаусса з частковим упорядкуванням.

36. Запишіть математичні формули, покладені в основу алгоритму зворотного ходу методу Гаусса.

37. Наведіть блок-схему алгоритму обчислення відхилів, які виникають у ході розв'язання СЛАР.

38. Що таке функціональна залежність? Охарактеризуйте способи завдання функції.

39. Опишіть математичну постановку задачі інтерполяції функцій.

40. Опишіть процес виводу інтерполяційної формули Лагранжа. Як оцінюється абсолютна похибка інтерполяційної формули Лагранжа?

41. Наведіть алгоритм формальної інтерполяції функцій у точці.

42. Опишіть математичну постановку задачі апроксимації функції.

43. Виведіть розрахункові формули для знаходження коефіцієнтів лінійної апроксимуючої функції.

44. Наведіть алгоритм лінійної апроксимації.

45. Як алгоритм лінійної апроксимації використовується для знаходження апроксимуючої функції у вигляді будь-якої елементарної функції з двома параметрами?

46. Виведіть розрахункові формули для знаходження коефіцієнтів апроксимуючої функції у вигляді квадратного тричлена.

47. Наведіть алгоритм квадратичної апроксимації.

48. Сформулюйте задачу Коші.

49. Виведіть формулу методу Ейлера для розв'язання задачі Коші. Яка геометрична інтерпретація методу Ейлера?

50. Наведіть алгоритм методу Ейлера.

51. Охарактеризуйте метод подвійного перерахунку для розв'язання задачі Коші методом Ейлера. Наведіть відповідну блок-схему алгоритму.

52. Опишіть процес виводу формули методу Рунге–Кутта для розв'язання задачі Коші.

53. Наведіть алгоритм методу Рунге–Кутта.

54. Охарактеризуйте метод подвійного перерахунку для розв'язання задачі Коші методом Рунге–Кутта. Наведіть відповідну блок-схему алгоритму.

55. Чому метод Рунге–Кутта є методом підвищеної точності?

СПИСОК ЛІТЕРАТУРИ ДО РОЗДІЛУ ДРУГОГО

1. **Бахвалов, Н. С.** Численные методы [Текст] / Н. С. Бахвалов. – М. : Наука, 1973.

2. **Березин, И. С.** Методы вычислений [Текст] / И. С. Березин, Н. П. Жидков. – М. : Физматгиз, 1966.

3. **Боглаев, Ю. П.** Вычислительная математика и программирование [Текст] / Ю. П. Боглаев. – М. : Высш. шк., 1990.

4. **Демидович, Б. П.** Основы вычислительной математики [Текст] / Б. П. Демидович, И. А. Марон. – М. : Наука, 1970.

5. **Демидович, Б. П.** Численные методы анализа [Текст] / Б. П. Демидович, И. А. Марон, Э. З. Шувалова. – М. : Наука, 1967.

6. **Заварыкин, В. М.** Численные методы [Текст] / В. М. Заварыкин, В. Г. Житомирский, М. П. Лапчик. – М. : Просвещение, 1991.

7. **Мак-Кракен, Д.** Численные методы и программирование на Фортране [Текст] / Д. Мак-Кракен, У. Дорн. – М. : Мир, 1977.

8. **Форсайт, Дж.** Машинные методы математических вычислений [Текст] / Дж. Форсайт, М. Мальком, К. Моллер. – М. : Мир, 1980.

ЗМІСТ

Навчальне видання

МИХЕЛЄВ Ігор Леонідович
СЛОБОДЯН Сергій Олегович

ОСНОВИ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

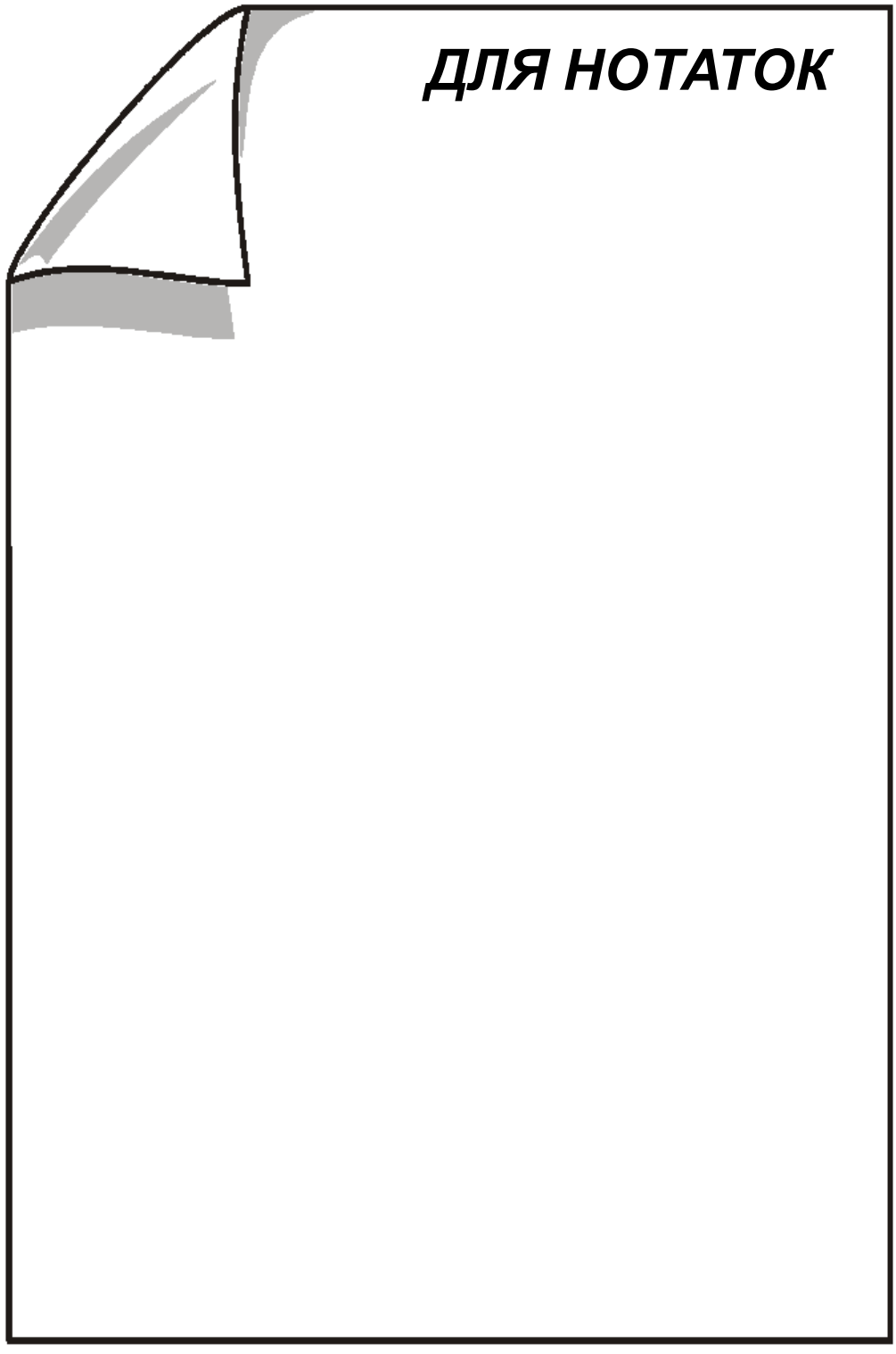
Навчальний посібник

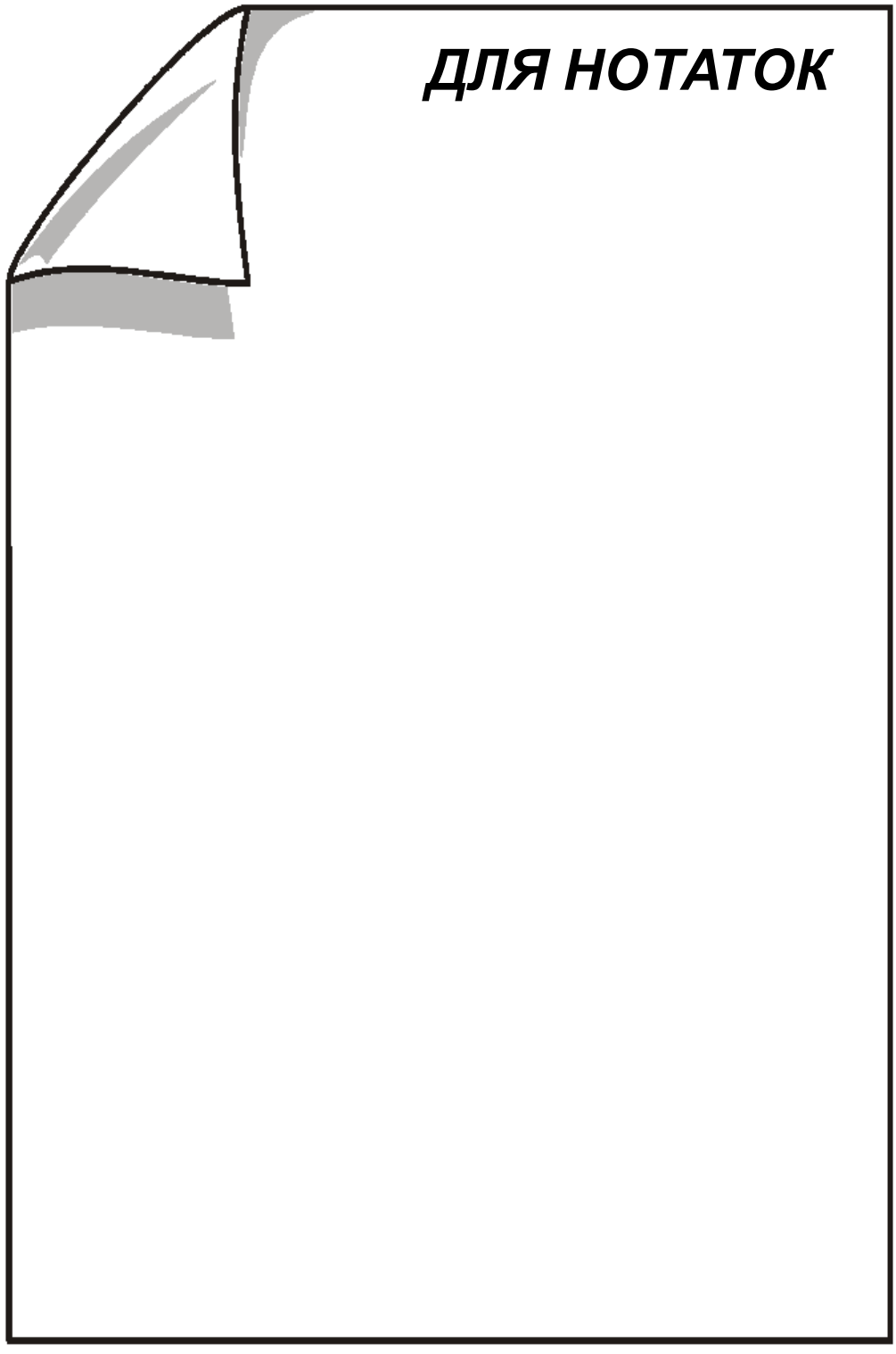
Комп'ютерне верстання *В. Г. Мазанко*
Коректор *М. О. Паненко*

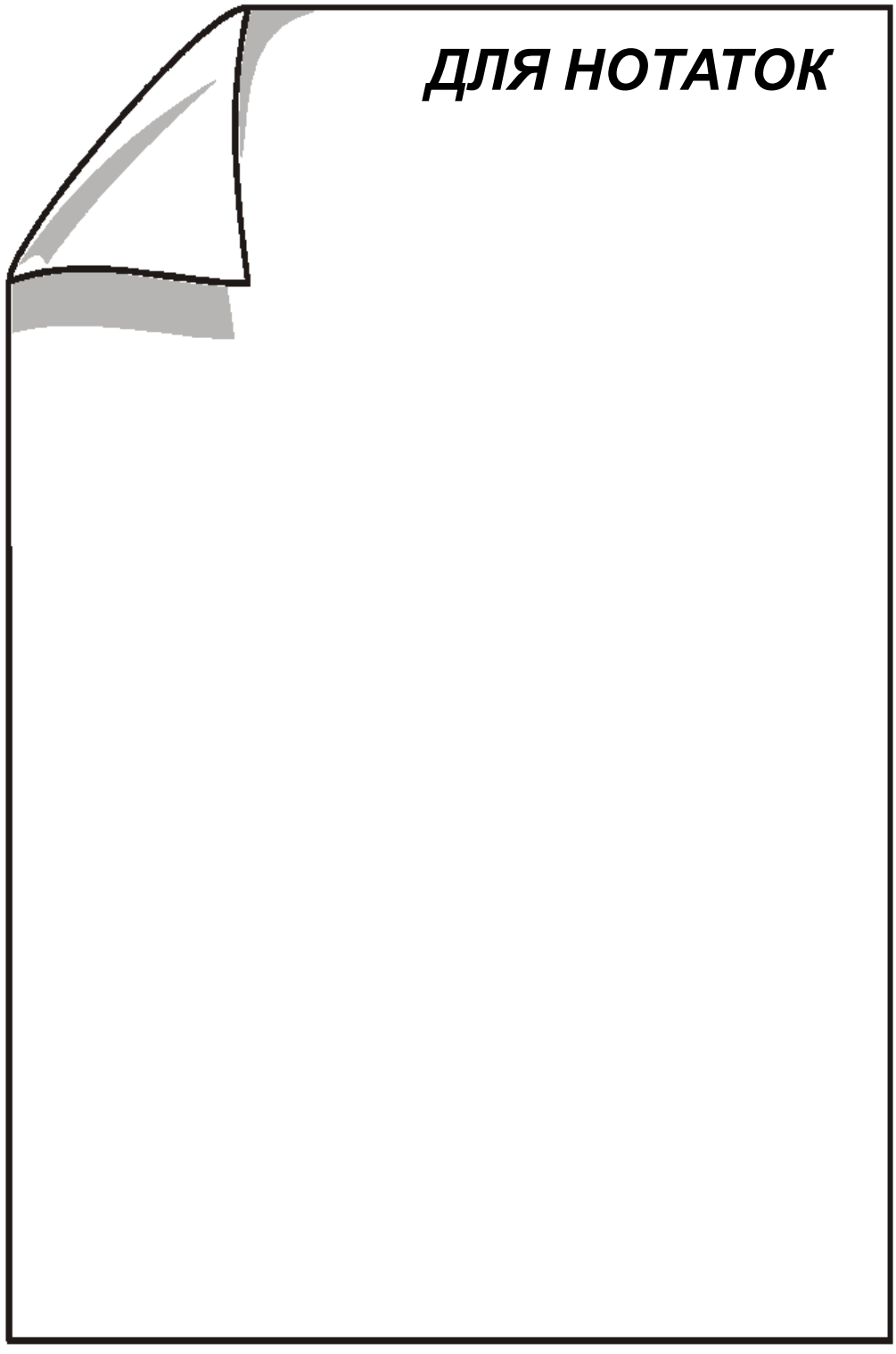
Формат 70×100/16. Ум. друк. арк. 13,7. Тираж 100. Вид. № 28. Зам. № 43.

Видавець і виготівник Національний університет кораблебудування
імені адмірала Макарова
просп. Героїв Сталінграда, 9, м. Миколаїв, 54025
E-mail : publishing@nuos.edu.ua

Свідоцтво суб'єкта видавничої справи ДК № 2506 від 25.05.2006 р.


ДЛЯ НОТАТОК


ДЛЯ НОТАТОК


ДЛЯ НОТАТОК