Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Алгоритми та методи обчислень

Лабораторний практикум

Рекомендовано Методичною радою КПІ імені Ігоря Сікорського
як навчальний посібник для здобувачів ступеня бакалавра
за освітньою програмою «Комп'ютерні системи і мережі»
спеціальності 123 Комп'ютерна інженерія

Укладачі: М. А. Новотарський, В. М. Порєв

Електронне мережеве навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2025

УДК 681.3.06

Укладачі: *Новотарський Михайло Анатолійович,* д-р техн. наук, проф.,
Порєв Віктор Миколайович, канд.техн. наук, доцент.

Рецензент *Дичка І. А.,* д-р техн. наук, проф., декан факультету прикладної математики, науковий керівник кафедри програмного забезпечення комп'ютерних систем КПІ ім. Ігоря Сікорського.

Відповідальний редактор *Стіренко С. Г.*, д-р техн-наук, проф.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 7 від 08.05.2025 р.)
за поданням вченої ради факультету інформатики та обчислювальної техніки
(протокол № 10 від 28.04.2025 р.)

Алгоритми та методи обчислень. Лабораторний практикум [Електронний ресурс]: лаб. практикум: навч. посіб. для здобувачів ступеня бакалавра за освіт. програмою «Комп'ютерні системи і мережі» спец. 123 Комп'ютерна інженерія / КПІ ім. Ігоря Сікорського; уклад.: М. А. Новотарський, В.М. Порєв – Електрон. текст. дані (1 файл). – Київ: КПІ ім. Ігоря Сікорського, 2025. – 120 с.

Навчальний посібник охоплює теоретичний матеріал та практичні завдання, які необхідні для виконання лабораторного практикуму з дисципліни «Алгоритми та методи обчислень». Практикум може виконуватися у комп'ютерному класі кафедри або дистанційно на персональних комп'ютерах студентів. В роботі описано 5 лабораторних робіт, тематика яких у повній мірі охоплює теми курсу «Алгоритми та методи обчислень».

До кожної роботи наведено методичні рекомендації з теорії, яка використовується при виконанні лабораторних робіт, наведені деякі алгоритми вирішення завдань, вимоги до виконання роботи, індивідуальні завдання та контрольні питання.

УДК 681.3.06

Реєстр. № НП 24/25-517. Обсяг 5,5 авт. арк.
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Берестейський, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

©КПІ ім. Ігоря Сікорського, 2025

ЗМІСТ

ВСТУП……..……..…….....6

Розділ 1. ЛАБОРАТОРНА РОБОТА №1……..…...8

1.1. Теоретичні відомості до лабораторної роботи №1……...8

1.1.1. Зображення алгоритму у вигляді блок-схеми……...…….....8

1.1.2. Класифікація алгоритмів за блок-схемою…….……...……....11

1.2. Вибір варіанту завдання на лабораторну роботу……..…….……13

1.3. Вимоги до програмного забезпечення……17

1.4. Зміст звіту……...17

1.5. Контрольні запитання……..……17

1.6. Інформаційні ресурси……...18

Розділ 2. ЛАБОРАТОРНА РОБОТА №2…….…….…….…19

2.1. Теоретичні відомості до лабораторної роботи №2……19

2.1.1. Алгоритми сортування…….……..……20

2.1.2. Опис алгоритмів сортування…….……..……..24

2.1.2.1. Бульбашкове сортування…….……..…24

2.1.2.2. Сортування вставками…….……..27

2.1.2.3. Сортування шейком……..…….……31

2.1.2.4.Сортування злиттям……32

2.1.2.5. Сортування Боуза-Нельсона……..34

2.1.2.6. Швидке сортування……37

2.1.2.7. Сортування Шелла…….……39

2.1.2.8. Сортування Хоара……..…….41

2.1.2.9. Пірамідальне сортування…….……..……44

2.1.2.10. Сортування підрахунком……45

2.1.2.11. Порозрядне сортування……...46

2.2. Вибір варіанту завдання на лабораторну роботу……....47

2.3. Вимоги до програмного забезпечення……..……..…48

2.4. Зміст звіту……..48

2.5.Контрольні запитання……..…….……48

2.6. Інформаційні ресурси ……....…….……..……..49

Розділ 3. ЛАБОРАТОРНА РОБОТА №3……..…….50

3.1.Теоретичні відомості до лабораторної роботи №3…….……...….50

3.1.1. Інтерполяційний многочлен Лагранжа ……....52

3.1.2. Інтерполяційний многочлен Ньютона…….….53

3.1.3. Схема Ейткена…….54

3.2. Методи оцінки похибки інтерполяції . Оцінка похибки методу
Теоретична оцінка похибки інтерполяції……..……57

3.3. Чисельний експеримент……...62

3.4. Вимоги до звіту з лабораторної роботи……..……66

3.5. Контрольні запитання……...66

3.6. Інформаційні ресурси ……...…….……..67

Розділ 4. ЛАБОРАТОРНА РОБОТА №4…….……..68

4.1. Теоретичні відомості до лабораторної роботи №4……68

4.1.1. Загальні поняття та визначення……...……..68

4.1.2. Метод половинного ділення…….……69

4.1.3. Метод хорд……..……70

4.1.4. Метод Ньютона (метод дотичних)……...…..……75

4.1.5. Комбінований метод…….……..80

4.1.6. Метод ітерацій (метод послідовних наближень)…….…83

4.2. Порядок виконання роботи…….……...…….…...…..86

4.3. Варіанти для виконання лабораторної роботи……..…….86

4.4. Контрольні запитання……..…….88

4.5. Інформаційні ресурси…….…….…….……89

Розділ 5. ЛАБОРАТОРНА РОБОТА №5……..…….90

5.1. Теоретичні відомості до лабораторної роботи №5……90

5.1.1. Основні поняття та визначення…..……...……90

5.1.2. Класифікація методів розв’язання СЛАР на комп’ютері…..……...…..91
5.1.3. Особливості методів Гауса…….…..…….93
5.1.3.1. Метод Гауса з послідовним виключенням невідомих……..93
5.1.3.2 Метод Гауса з вибором головного елемента…….……97
5.1.3.3. Метод Гауса з одиничними коефіцієнтами…….……...98
5.1.4. Загальний підхід до розв’язання СЛАР наближеними методами……100
5.1.5. Умови збіжності ітераційного процесу…….…105
5.1.6. Метод послідовних наближень (метод Якобі)……...107
5.1.7. Метод Гауса-Зейделя……...…….110
5.1.8. Метод верхньої релаксації……...112
5.2. Варіанти виконання лабораторної роботи……...……...113
5.3. Контрольні запитання…….117
5.4. Інформаційні ресурси.…….…….……..118
Перелік використаних джерел……..119

ВСТУП

Навчальний посібник містить навчально – методичні матеріали по виконанню лабораторних робіт з дисципліни «Алгоритми та методи обчислень», яка входить до переліку нормативних освітніх компонент освітньо-професійної програми першого (бакалаврського) рівня вищої освіти «Комп'ютерні системи та мережі» за спеціальністю 123 Комп'ютерна інженерія.

Відповідна навчальна дисципліна містить такі теми: основні поняття про алгоритми, основи теорії алгоритмів, чисельні методи обчислень.

Метою лабораторного практикуму з курсу «Алгоритми та методи обчислень» є засвоєння на практиці сучасних методів та технологій розробки і оцінювання алгоритмів. Підготовка студентів у рамках даного лабораторного практикуму включає набуття навичок вибору та використання методів алгоритмізації, методів обчислень, які є стійкими до похибок. Студенти отримують практичний досвід застосування ефективних методів обчислення математичних задач, аналізу одержаних наближених розв'язків, створення високоефективних алгоритмів і програм, що враховують особливості реалізації обчислень з використанням комп'ютерів. В результаті отримані знання сприяють розвитку логічного та аналітичного мислення студентів.

Лабораторна робота №1 є вступною лабораторною роботою, яка має на меті отримання навичок створення блок-схем лінійного алгоритму, розгалуженого алгоритму та циклічного алгоритму. Відповідно до варіанту завдання необхідно сформувати блок-схему конкретного алгоритму, яка максимально наближена до програмної реалізації. Виконання лабораторної роботи має включати розробку програми з використанням загально вживаних алгоритмічних мов програмування. Стиль написання програми має обов'язково включати графічний інтерфейс користувача. Основні вимоги до інтерфейсу задані у описі вимог до виконання лабораторної роботи.

Лабораторна робота №2 присвячена отриманню практичних навичок визначення обчислювальної складності на прикладі алгоритмів сортування. Суть роботи полягає у написанні програми сортування у відповідності з варіантом. У

програмі необхідно передбачити вимірювання часу сортування з максимально можливою точністю. Аналіз результатів має включати побудову графіка часу виконання алгоритму від розміру вхідного масиву даних. Отриманий графік порівняти з графіком теоретично відомої обчислювальної складності.

Лабораторна робота №3 призначена для ознайомлення з практичним використанням інтерполяційних многочленів Лагранжа, Ньютона та Ейткена. Також в лабораторній роботі застосовуються методи оцінки похибки інтерполяції. Теоретичні відомості до даної лабораторної роботи включають блок-схеми побудови алгоритмів у відповідності з варіантами. Варіанти завдань включають аналітичні представлення функцій, табличне представлення яких необхідно інтерполювати.

Лабораторна робота №4 має на меті ознайомлення з методами розв'язування нелінійних рівнянь на комп'ютері та застосування даних методів до практичного вирішення задач. В ході виконання лабораторної роботи студенти набувають умінь і навичок при програмуванні та налагодженні програм для розв'язування нелінійних рівнянь на комп'ютері. У лабораторні роботі передбачено використання таких методів: метод хорд, метод дотичних, метод половинного ділення, комбінований метод та метод ітерацій.

Лабораторна робота №5 присвячена вивченню методів розв'язування лінійних систем алгебраїчних рівнянь. В ході виконання лабораторної роботи студент має скласти блок-схему алгоритму для розв'язування системи лінійних алгебраїчних рівнянь у відповідності з індивідуальним варіантом виконання роботи. Наступний крок виконання лабораторної роботи полягає у створенні програми для розв'язування системи лінійних алгебраїчних рівнянь.

Таким чином, даний комплекс лабораторних робіт дозволяє студентам не тільки закріпити отримані теоретичні знання, а й отримати необхідні практичні навички х програмування задач, для яких дані методи застосовуються.

Розділ 1. ЛАБОРАТОРНА РОБОТА № 1

Тема: «Поняття алгоритму. Задавання алгоритмів у вигляді блок-схем».

Мета: Навчитися створювати блок-схеми лінійного алгоритму; розгалуженого алгоритму та циклічного алгоритму за допомогою редактора блок-схем **afce** або іншого довільного редактора.

Завдання: Відповідно до варіанту завдання розробити блок-схеми обчислення виразів для лінійного алгоритму, алгоритму, що розгалужується та циклічного алгоритму. У відповідності до блок-схеми створити програму обчислення виразу алгоритмічною мовою, узгодженою з викладачем.

1.1. Теоретичні відомості до лабораторної роботи №1

Алгоритм – чіткий опис послідовності дій, які необхідно виконати при розв’язуванні задачі. Можна сказати, що алгоритм описує процес перетворення вхідних даних у результати, тому для розв’язування будь-якої задачі необхідно:

- Ввести вхідні дані.
- Перетворити вхідні дані в результати (вихідні дані).
- Вивести результати.

Розробка алгоритму розв'язання задачі – це розбиття задачі на послідовно виконувані етапи, причому результати виконання попередніх етапів можуть використовуватися при виконанні подальших. При цьому мають бути чітко вказані як зміст кожного етапу, так і порядок виконання етапів. Окремий етап алгоритму являє собою або іншу, більш просту задачу, алгоритм розв'язання якої відомий (розроблений заздалегідь), або повинен бути достатньо простим і зрозумілим без пояснень.

1.1.1. Зображення алгоритму у вигляді блок-схеми

Блок-схемою називається наочне графічне зображення алгоритму, коли окремі його етапи зображуються за допомогою різних геометричних фігур – блоків, а зв'язки між етапами (послідовність виконання етапів) вказуються за допомогою стрілок, що з'єднують ці фігури. Конфігурація і розміри блоків, а також порядок

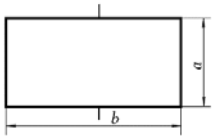
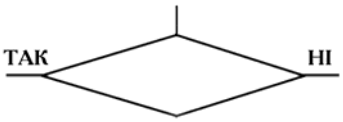
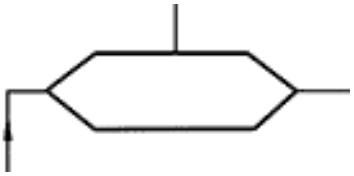
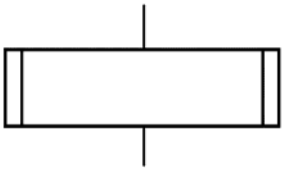
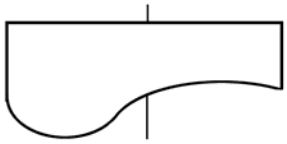
графічного оформлення блок-схем регламентовані державним стандартом "Схеми алгоритмів, програм, даних і систем", а саме: 19.701-90 (ISO 5807-85).

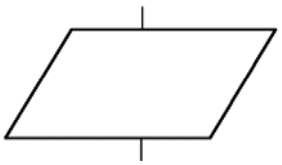
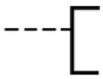
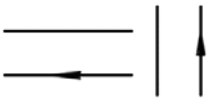
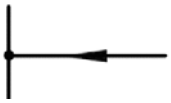
У табл. 1.1 наведені найбільш часто використовувані блоки, зображені елементи зв’язків між ними і дано коротке пояснення до них. Блоки і елементи називають елементами блок-схем.

Представлених в таблиці елементів цілком достатньо для зображення алгоритмів, які необхідні при виконанні студентських робіт. При з'єднанні блоків слід використовувати тільки вертикальні і горизонтальні лінії потоків. Горизонтальні потоки, що мають напрямок справа наліво, і вертикальні потоки, що мають напрямок знизу вгору, повинні бути обов'язково позначені стрілками. Інші потоки можуть бути помічені або залишені не поміченими. Лінії потоків повинні бути паралельні лініям зовнішньої рамки або границь аркуша.

Таблиця 1.1

Елементи блок-схем

Назва	**Елемент**	**Коментар**
Процес		Обчислювальна дія або послідовність обчислювальних дій
Рішення		Перевірка умови
Модифікація		Заголовок детермінованого циклу
Зумовлений процес		Звернення до процедури
Документ		Виведення даних, друк даних

Введення / Виведення		Введення / Виведення даних
З'єднувач		Розрив лінії потоку
Початок, Кінець		Початок, кінець, пуск, зупинка, вхід і вихід в допоміжних алгоритмах
Коментар		Використовується для розміщення написів
Горизонтальні і вертикальні потоки		Лінії зв'язків між блоками, напрям потоків
Злиття		Злиття ліній потоків
Міжсторінковий з'єднувач		Немає

Відстань між паралельними лініями потоків повинна бути не менше 3 мм, між іншими елементами схеми – не менше 5 мм. Горизонтальний і вертикальний розміри блоку повинні бути кратними 5 мм (ділитися на 5 без остачі). Співвідношення горизонтального і вертикального розмірів блоку b / a = 1.5 є основним. При ручному виконанні блоку допустимо відношення b / a = 2. Блоки "Початок", "Кінець" і "З’єднувач" мають висоту a/2, тобто вдвічі менше основної висоти блоків. Для розміщення блоків рекомендується поле аркуша розбивати на горизонтальні і вертикальні (для розгалужених схем) зони. Для зручності опису блок-схеми кожен її блок слід пронумерувати. Зручно використовувати наскрізну нумерації блоків. Номер блоку розташовують у розриві в лівій верхній частині рамки блоку.

1.1.2. Класифікація алгоритмів за блок-схемою

В залежності від структури алгоритми бувають: лінійні, що розгалужуються та циклічні.

Лінійні алгоритми зазвичай представляють найпростішими блок-схемами, що описують відносно прості обчислення або процеси. Етапи виконання окремих етапів в таких алгоритмах є незмінними і виконуються послідовно один за одним (Рис. 1.1).

Введення даних

Обчислення

Виведення результату

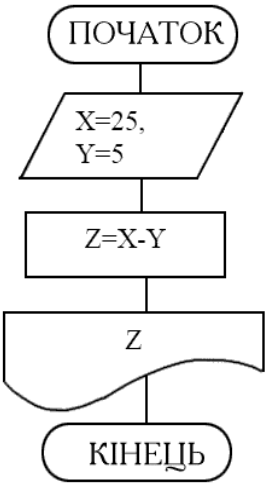


Рис. 1.1. Приклад лінійного алгоритму

Алгоритми, що розгалужуються представляють блок-схемами, що дозволяють обирати виконання процесу у тому чи іншому напрямку в залежності від умов, заданих заздалегідь, або визначаються в процесі виконання програми.

На рис.1.2 зображено блок-схему алгоритму, який розгалужується.

Циклічні алгоритми представляють блок-схемами, які дозволяють вибирати як напрямок виконання процесу, так і в залежності від заданих умов циклічно (повторно) виконувати певні процеси або операції. Більшість алгоритмів є циклічними.

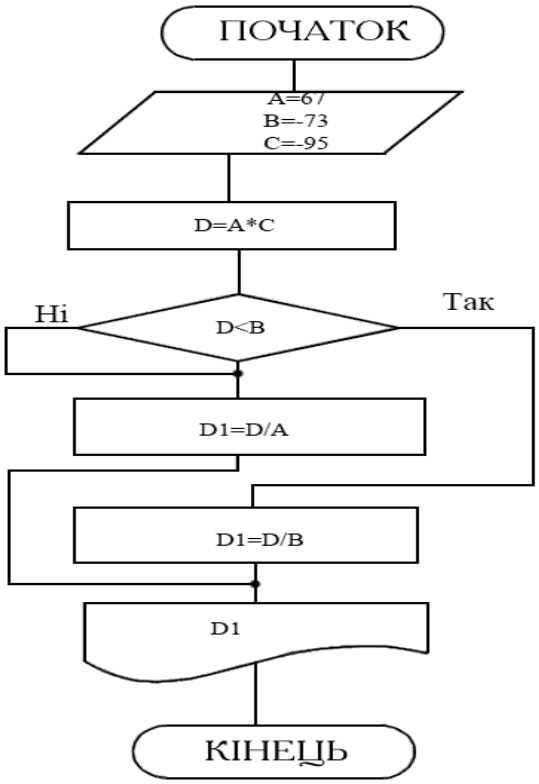


Рис. 1.2. Блок-схема алгоритму, що розгалужується

На рис. 1.3. показано приклад обчислення значення виразу К1=a+sin2b, якщо змінна *a* змінюється в діапазоні від 1 до 10 з кроком 1. Це приклад блок-схема циклічного алгоритму.

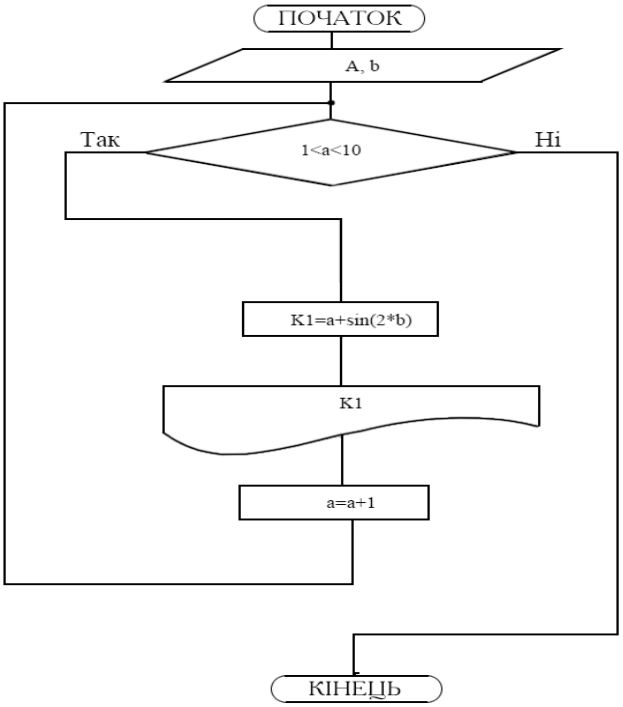


Рис. 1.3. Приклад блок-схеми обчислення виразу

1.2. Вибір варіанту завдання на лабораторну роботу

Всі змінні, які не мають значень, вводять з клавіатури.

Відповідно до номера у списку групи вибрати варіант завдання для виконання лабораторної роботи. Варіанти завдань зведені в таблицю 1.2.

Таблиця 1.2

Варіанти завдань до лабораторної роботи №2

	Лінійний	Що розгалужується	Циклічний
1	$Y1=\sqrt{(a+c)}+\frac{1}{(a+c)}$	Знайти корені рівняння: $57.567x^2-11.675x-34.114=0$	Обчислити: $f=\sum_{i=0}^{10}\left(a_i^2+56\cdot c_i\cdot f\cdot g_i\right)$
2	$Y1=\lceil a-z\rceil+\frac{\lceil a+z\rceil}{6}$	Знайти значення виразу $y=2a^2+x$ для $a>10$, та $y=2a^2-x$ у іншому випадку	Обчислити $f=\prod_{i=1}^{5}\frac{\left(a_i^3-b_i^2\right)}{a_i\cdot b_i}$
3	$Y1=(a\cdot c)^2+(b\cdot c)^3+(c\cdot c)^4$	Якщо $\frac{b}{z}>d$ то $y=\sin(w\cdot f)$ в іншому випадку $y=\cos(w\cdot f)$	Обчислити $f=\frac{a!\cdot b}{(a-b)!}$
4	$Y1=\left(\frac{a}{d}\right)^2+\left(\frac{b}{d}\right)^3+\left(\frac{c}{2}\right)^4$	Якщо $f=0$, то $y=\lg(l\cdot k)+d\cdot\sin(w\cdot k)$ в іншому випадку $y=\cos(w\cdot k)+\lg(l\cdot k)$	Обчислити та вивести на друк значення кожного $f=\sqrt{a^2+b^2}-(a+b)^2$ a - змінюється в діапазоні від –4 до 18 з кроком 1
5	$Y1=\frac{(a+x)^2}{5}+\frac{(a+c)^3}{2}$	Якщо $k\cdot c>p$ то $y=\sin^2(c\cdot a)$ інакше якщо $k\cdot c<p$ то $y=\cos^2(k\cdot a)$	Обчислити $f=\sum_{i=1}^{n}\left(\prod_{j=1}^{p}\left(a_i^2+b_j^3\right)\right)$
6	$Y1=(d\cdot a)^b+(b\cdot c)^{\frac{1}{d}}$	Обчислити $y=\frac{4\cdot r-r\cdot x}{4\cdot x-r\cdot x}$ передбачити запобігання діленню на нуль.	Обчислити $f=\sum_{a=0}^{n}\left(\sum_{b=0}^{p}\left(a^b+b^a\right)\right)$
7	$Y1=\frac{(a^2-c^2)}{7}+\left(\frac{a!}{c!(a-c)!}\right)$	Якщо $x>0$ $y=k\cdot x^2+c\cdot x+b$ інакше $y=b\cdot x^2+c\cdot x+k$	Обчислити $f=\sum_{i=1}^{n}\left(\prod_{j=1}^{n}a_i+a_j^2\right)$
8	$Y1=\left(\frac{b\cdot\sqrt{c}}{2^b}\right)-\left(\frac{c\cdot\sqrt{b}}{2^c}\right)$	Якщо $x>0$ то $y=a\cdot x^3-b\cdot x^2$ інакше якщо $x\le 0$ то $y=b\cdot x^3+a\cdot x^2$	Обчислити $f=\prod_{j}^{n}j!-\sum_{i}^{n}i!$
9	$Y1=\sqrt[3]{5+c\cdot\sqrt{b+5\sqrt{a}}}$	Обчислити якщо $k>10$ $y=\sqrt{k\cdot\sqrt{d^2}+d\cdot\sqrt{k^2}}$ інакше	Обчислити $f=\prod_{i=1}^{n}\left(a_i+b_{i+1}\right)+\sum_{i}^{n}\left(a_{i+1}\cdot b_i\right)$

	Лінійний	Що розгалужується	Циклічний
		$y=(k+d)^2$	
10	$Y1=\frac{\sqrt{a}+b^2}{\sqrt{b}-a^2}+\sqrt{\frac{a\cdot b}{c\cdot d}}$	Обчислити якщо $k<10$ $y=(a+c)^4+(a-c)^2$ інакше $y=(a-c)^3+(a+c)^2$	$f=\sum_{i=1}^{p}\sum_{j=1}^{p}\sum_{k=1}^{p}\left(i\cdot\left(i\cdot j\cdot\left(i\cdot j\cdot k\cdot\sqrt{(a+b)}\right)\right)\right)$
11	$Y1=\sqrt{\sin\left(\frac{a}{6}\right)+\cos\left(\frac{b}{6}\right)+}$ $+\sqrt{2\sin\left(\frac{a}{6}\right)\cos\left(\frac{b}{6}\right)}$	Обчислити $y=k\cdot x^2\cdot\lg(k\cdot x)+\sqrt{x}$ і передбачити запобігання від'ємного значення під знаком логарифма.	Обчислити $f=\prod_{a=1}^{n}\sum_{k=1}^{a}\left(a^k+\frac{b}{k}\right)$
12	$Y1=2\cdot\cos\left(b\frac{c}{2}\right)+$ $+2\sin\left(c\frac{b}{2}\right)$	Обчислити $y=\sqrt{\frac{d-b-kj}{23\cdot\sqrt{g\cdot f}+6\cdot\sqrt{v\cdot c}}}$, передбачити: - неможливість обчислення значення у разі від'ємного значення підкореневого виразу; - запобігання ділення на нуль.	Обчислити $f=\sum_{i=0}^{50}\left(a_i^2+56\cdot c_i\cdot\sqrt{f_i\cdot g_i}\right)$
13	$Y1=\left(\frac{a}{b}\right)^3+\left(\frac{c}{2}\right)^2$	Обчислити, якщо $b,\ c,\ d>4$ $y=\frac{a}{b}+\frac{a\cdot b}{c}+\frac{a\cdot b\cdot c}{d}$ інакше якщо $b\le 4$ то $y=\frac{b}{a}+\frac{c\cdot b}{b}+\frac{d\cdot b\cdot c}{c}$ інакше якщо $c\le 4$ $y=\frac{b}{c}+\frac{a\cdot b}{b}+\frac{d\cdot b\cdot a}{a}$	Обчислити $f=\sum_{i=1}^{n}\sqrt{d_i+\sqrt{c_i+\sqrt{a_i+b_i}}}$
14	$Y1=\frac{(a+b\cdot x)^2}{c+d}+\frac{c+d}{(a+b\cdot x)^3}$	Обчислити, якщо $x>0$, то $y=\frac{123\cdot x^3+124\cdot y^4}{125\cdot z^5}$, якщо $y>0$ то $y=\frac{123\cdot y^3+124\cdot z^4}{125\cdot x^5}$, якщо $z>0$ то $y0\frac{123\cdot z^3+124\cdot y^4}{125\cdot x^5}$.	Обчислити $f=\sum_{i=1}^{n}\sum_{j=1}^{n}\left(a^i+b^j\right)$
15	$Y1=\left(a+\frac{b}{x}\right)^3+\left(b+\frac{a}{x}\right)^5$	Якщо $i=2n+1$ то $y=25\cdot c_i^2-\sqrt{\frac{2\cdot i}{34\cdot x^2}}+4\cdot\sqrt{\frac{i}{45x^2}}$ Інакше, якщо $i=2n$, то $y=25\cdot x^2-\sqrt{\frac{2\cdot i}{34\cdot c_i^2}}+4\cdot\sqrt{\frac{i}{45c_i^2}}$	Обчислити $f=\prod_{i=1}^{n}\left(a_i^3-b_i^3\right)+\sum_{i=1}^{n}\left(a_i^3+b_i^3\right)$

	Лінійний	Що розгалужується	Циклічний
16	$Y1=\frac{\left(\frac{a}{b}+\frac{b}{a}\right)\cdot\left(\frac{a}{c}+\frac{c}{a}\right)}{a+b+c}$	Якщо $i=2n+1$ то $y=\frac{d\cdot m^5-d^5m}{i\cdot d}$ Інакше, якщо $i=2n$ то $y=\frac{i\cdot d}{d\cdot m^3-d^3m}$	Обчислити $f=\sum_{a=0}^{n}a^2+56\cdot c_a\cdot f\cdot$ якщо a змінюється з кроком 1
17	$Y1=\frac{\left(\frac{a}{b}-5\right)}{2}+\sqrt{\frac{\left(\frac{b}{a}+5\right)}{3}}$	Якщо $i \bmod 3=0$ то $y=\frac{q^i\cdot d}{\sqrt{v+x}}+\frac{p^i\cdot d}{\sqrt{c+h}}$ інакше $y=\frac{v\cdot d}{\sqrt{q^i+x}}+\frac{c\cdot d}{\sqrt{p^i+h}}$	Обчислити $f=\prod_{a=1}^{n}\prod_{b=1}^{n}\frac{\left(a^4+b^4\right)}{\left(a^4-b^4\right)}$, де a змінюється з кроком 0,25
18	$Y1=\sin(a+b)-$ $-\left(\cos(a-b)\right)^2$	Якщо $i \bmod 5<3$ $y=\frac{g^{g+i}}{nb^{k+i}}$ інакше $y=\sqrt[4]{\frac{nb^{g+i}}{g^{k+i}}}$	$f=\sqrt{\frac{\sum_{i=1}^{n}\left(m_i+c_i\right)}{\sum_{j=1}^{n}\left(p_j+c_j\right)}}$
19	$Y1=\lg\left(\frac{a}{b}\right)+\ln\left(\frac{b}{a}\right)$	Якщо $i \bmod 3\le 1$ Обчислити вираз $y=\sqrt{\frac{a^{b\cdot i}}{c^{\frac{i}{a}}}}+\sqrt{\frac{b^{c\cdot i}}{a^{\frac{i}{c}}}}$ інакше $y=\frac{a+b}{(c+a)^i}+\left(i+c^i\right)$	$f=\sum_{i=1}^{n}\left(\frac{p_i+i}{\sum_{j=1}^{n}\left(\frac{p_i+p_j}{p_j-j}\right)}\right)$
20		Якщо $\sqrt{b^2-4ac}>0$ то $y=ax^2+bx+c$ інакше $y=cx^2-ax+b$	$f=\sum_{b=1}^{n}\frac{\sum_{i=1}^{n}a_i}{n}+b$
21	$Y1=\frac{\lg(a+b)}{\ln(a-b)}+\frac{\ln(a+b)}{\lg(a-b)}$	Якщо $\sqrt{a^2+b^2}>10$ то $y=\sqrt{\frac{a^2+b^2}{a^2-b^2}}$ інакше $y=\sqrt{\frac{a^2-b^2}{a^2+b^2}}$	Знайти значення виразу $f=\sum_{i=1}^{n}\left(i!-(i-1)!\right)$
22	$Y1=\sin\left(\frac{a+b}{(a+b)^2}\right)+$ $\cos\left(\frac{a+b}{(a+b)^2}\right)$	Якщо $r\cdot g\ne 0$ $y=\frac{x+24\cdot x^2}{r\cdot g}$ інакше $y=\sqrt{r+x}+g\cdot x$	Обчислити $f=\frac{\sum_{i=1}^{n}n!}{\sum_{i=1}^{n}\left(a_i+b_i\right)}$

	Лінійний	Що розгалужується	Циклічний
23	$Y1=\sin\left(\frac{a}{b}\right)+\sin^2\left(\frac{a}{b}\right)+$ $+\cos\left(x^2\right)+\cos\left(\sqrt{x}\right)$	Якщо $r>0$ то $y=\frac{\pi r^2}{(2\pi r+21r)}$ інакше $y=\frac{c^2+b^2}{\pi r^2}$	Обчислити $f=\frac{\prod_{i=1}^{n}a_i+\sum_{j=1}^{p}b_j}{\sum_{i=1}^{n}\sum_{j=1}^{p}(a_i+b_j)}$
24	$Y1=\left(\frac{a+b}{a-b}+\frac{c+s}{c-s}\right)^2$	Якщо $d>0$ то $y=\pi\cdot d\cdot h+\sqrt{\pi+23d}$ інакше $y\frac{\sqrt{\pi\lvert d\rvert}}{129\cdot h}$	Обчислити $f=\sum_{j=0}^{p}\left(a_j+\sum_{i=1}^{n}\frac{(a_j+b_i)}{a_jb_i}\right)$
25	$Y1=\left(a+\frac{b}{c}\right)^d+\left(d+\frac{b}{c}\right)^a$	Якщо $x>0$ то $y=\sin(x)+\sqrt{\cos(x)}$ інакше $y=\sqrt[2]{\frac{(a+x)}{(b-x)}}$	Обчислити $f=\frac{1}{n_1}+\frac{1}{n_2}+\ldots+\frac{1}{n_m}$ m=25
26	$Y1=\frac{(z+y)^{g+h}}{(z+x)^{\frac{g+h}{gh}}}$	Якщо $v\cdot b\neq 0$ то $y=\sqrt{24\cdot g-s\cdot b-(y\cdot t)/(v\cdot b)}$	Обчислити $f=\frac{1}{n_1}\cdot\frac{1}{n_2}\cdot\ldots\cdot\frac{1}{n_m}$ *m*=15
27	$Y1=s^{\frac{a}{b}+c}+d^{\frac{c}{b}+a}$	Якщо $\sin(x)>0$ то $y=\frac{\cos(x)}{\sin(x)}+\sqrt[3]{\sin(x)}$ інакше якщо $\sin(x)=0$ то $y=\frac{\cos^2(x)}{0.15}$ інакше якщо $\sin(x)<0$, то $y=\frac{\cos(x)}{\sin(x)}+\sqrt[3]{\cos(x)}$	Обчислити $f=\frac{n_1\cdot n_2\cdot\ldots\cdot n_m}{n_1+n_2+\ldots+n_m}$ m=45
28	$Y1=\frac{\sqrt{a+b}}{(c+2\cdot b)}+\frac{\sin(a)}{\sqrt{a^2-b^2}}$	Якщо $\frac{n!}{r!(n-r)!}>2345$ то $y=\frac{n!}{(n-r)!}$ інакше $y=\frac{n^r}{(n-r)!}$	$f=\frac{\sum_{a=1}^{5}a^3}{\prod_{b=1}^{5}b^6}$
29	$Y1=\frac{(c+27\cdot g)^{(g+27\cdot c)}}{234\cdot c\cdot g}$	Якщо $x>23$ то $y=\frac{x+2\cdot z}{z-2\cdot x}+\frac{\sin(x)}{21}$ інакше $y=\frac{1+x}{1+\frac{x}{23}}+\frac{\cos(x)}{21}$	Вивести на друк числа від 1 до 2000 з кроком, що дорівнює сумі попереднього та попереднього помноженого на три. Перший крок дорівнює одиниці.

	Лінійний	Що розгалужується	Циклічний
30	$Y1=\sin^2(a+b)+$ $+\cos^2(a-b)$	Якщо $\frac{a}{b}>1$ то $y=\frac{a-b}{a+b}+\sqrt{\frac{(a-b)^2}{c}}$ інакше $y=\sqrt{\frac{a-b}{a+b}+\sqrt{\frac{a^b}{25}}}$	$f=\prod_{i=1}^{n}\left(a_i\sum_{j=1}^{p}b_j\right)$

1.3. Вимоги до програмного забезпечення

- Модульна структура програми;
- Уведення даних із клавіатури та з зовнішнього файлу;
- Перевірка коректності введених даних;
- Інтерфейс у стилі GUI.

1.4. Зміст звіту

- Титульний лист;
- Тема завдання;
- Завдання;
- Блок-схеми алгоритмів;
- Роздруківки блок-схем;
- Роздруківка тексту програми;
- Роздруківка результатів виконання програми;
- Аналіз результатів.

1.5. Контрольні запитання

- Дати визначення алгоритму.
- З яких основних етапів складається алгоритм розв’язування задачі за допомогою комп’ютера?

- Які основні елементи блок-схем ви знаєте?
- Перерахуйте загальні властивості алгоритмів.
- Які існують способи задавання алгоритмів.

1.6. Інформаційні ресурси

1. Марченко Р. Що таке алгоритми: кроки приклади, конструкції, мислення: https://dan-it.com.ua/uk/blog/shho-take-algorytmy-kroky-pryklady-konstrukcziyi-myslennya/
2. Лінійні алгоритми: https://ua.izzi.digital/DOS/193559/276443.html
3. Що таке алгоритми з розгалуженням: https://ua.izzi.digital/DOS/327580/432019.html
4. Циклічні алгоритми: https://step.org.ua/konspekt/algoritm/tema5

Розділ 2. ЛАБОРАТОРНА РОБОТА № 2

Тема: «Обчислювальна складність алгоритмів сортування».

Мета: Закріплення навичок практичної оцінки алгоритмічної складності логічних алгоритмів на прикладі алгоритмів сортування.

Завдання: Використовуючи відповідний до варіанту алгоритм сортування написати програму сортування масиву даних. Застосовуючи дану програму, дослідити часову складність алгоритму сортування та порівняти її з теоретичною алгоритмічною складністю.

Порядок виконання лабораторної роботи

- Написати програму сортування у відповідності з вибраним варіантом.
- У програмі передбачити вимірювання часу сортування з максимально можливою точністю.
- Створити 10 тестових масивів даних різного розміру, які підлягають сортуванню.
- Побудувати графік залежності часу виконання алгоритму від розміру вхідного масиву даних.
- Побудувати графік теоретично відомої обчислювальної складності.
- Порівняти одержані графіки часової складності та обчислювальної складності вашого алгоритму сортування.

2.1. Теоретичні відомості до лабораторної роботи №2

Критерії оцінки алгоритму

Основний критерій оцінки алгоритмів – залежність числа операцій конкретного алгоритму від обсягу вхідних даних. Ми можемо порівняти два алгоритми за швидкістю зростання числа операцій від зростання обсягу вхідних даних. Саме швидкість зростання відіграє ключову роль.

Золоте правило для алгоритмів. Програємо в кількості операцій – виграємо в обсязі використовуваної пам'яті і навпаки, виграємо в кількості операцій – програємо в обсязі використовуваної пам'яті (*пам'яті потрібно **більше***).

Обчислювальна складність

Обчислювальна складність алгоритму — це *функція*, що визначає *залежність обсягу роботи*, виконуваної деяким алгоритмом, *від обсягу вхідних даних*. Обсяг роботи зазвичай вимірюють категоріями часу. Обсяг даних визначають представленими обчислювальними ресурсами. Час визначається кількістю елементарних кроків, необхідних для розв'язування задачі. Простір визначається обсягом пам'яті (біти, байти) або місцем на носієві даних (доріжки, сектори).

Асимптотична обчислювальна складність Θ

Якщо $f(n)$ і $g(n)$ – деякі функції, то запис $f(n)=\Theta(g(n))$ означає, що найдуться такі $c_1, c_2 > 0$ й таке n_0, що $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ для всіх $n \ge n_0$.
У цьому випадку говорять, що $g(n)$ є асимптотично точною оцінкою для fn.
Скорочений запис: $\exists(c_1, c_2 \ge 0), n_0 : \forall(n \ge n_0) \Rightarrow |c_1 g(n)| \le |f(n)| \le |c_2 g(n)|$

Асимптотичні обчислювальні складності O та Ω.

Запис $f(n)=\Theta(g(n))$ містить у собі дві оцінки: верхню й нижню. Їх можна розділити.

Якщо $f(n)$ та $g(n)$ – деякі функції, то запис $f(n)=O(g(n))$ означає, що знайдеться така константа $c>0$ й таке n_0, що $f(n) \le cg(n)$ для всіх $n \ge n_0$.

Скорочено: $\exists c, n_0 : \forall(n \ge n_0) \Rightarrow |f(n)| \le |cg(n)|$.

Запис $f(n)=\Omega(g(n))$ означає, що знайдеться така константа $c>0$ й таке n_0, що $0 \le cg(n) \le f(n)$ для всіх $n \ge n_0$.

$$\exists c, n_0 : \forall(n \ge n_0) \Rightarrow |cg(n)| \le |f(n)|$$

2.1.1. Алгоритми сортування

Алгоритмічну складність алгоритмів будемо розглядати на прикладі алгоритмів сортування.

Сортування — це впорядкування елементів у списку. У випадку, коли елемент має кілька полів, поле, що служить критерієм порядку, називається ключем сортування. На практиці в якості ключа часто виступає число, а в інших полях зберігаються будь-які дані, що не впливають на роботу алгоритму.

Розглянемо найпоширеніші алгоритми сортування для чисел, хоча вони можуть бути поширені на рядки й списки елементів, про яких говорилося в загальнім визначенні.

Класифікація алгоритмів сортування

При класифікації враховують наступні основні характеристики алгоритмів.

- *Стійкість* (stability) – стійке сортування не міняє взаємного розташування елементів, які дорівнюють один одному.
- *Природність поведінки* – ефективність методу при обробці вже впорядкованих або частково впорядкованих даних. Алгоритм поводиться природно, якщо враховує цю особливість вхідної послідовності.
- *Використання операції порівняння*. Алгоритми, які при сортуванні порівнюють елементи між собою, називають алгоритмами, які базуються на порівняннях. Алгоритмічна складність для гіршого випадку у цих алгоритмах становить $O\left(n^2\right)$, але вони відрізняються гнучкістю застосування. Для спеціальних випадків (типів даних) існують ефективніші алгоритми.

Ще однією важливою властивістю алгоритму є сфера його застосування. При цьому розрізняють два основні типи впорядкування.

1. *Внутрішнє сортування* використовує масиви, які цілком містяться в оперативній пам'яті з довільним доступом до будь-якого елементу масиву. Дані зазвичай впорядковуються на тому ж місці, без додаткових витрат. У сучасних персональних комп'ютерах широко застосовується підкачування та кешування пам'яті. Алгоритм сортування повинен добре поєднуватися з цими операціями.

2. *Зовнішнє сортування,* яке застосовується для запам'ятовувальних пристроїв великого об'єму, з послідовним доступом (упорядкування файлів). При цьому в кожний момент доступний тільки один елемент, а витрати на перемотування в порівнянні з пам'яттю невиправдано великі. Це накладає додаткові обмеження на алгоритм і приводить до методів упорядкування, що зазвичай використовують додатковий дисковий простір. Крім того, доступ до даних на носії проводиться набагато повільніше, ніж операції з оперативною пам'яттю. Обсяг даних настільки великий, що не дозволяє розмістити інформацію в оперативній пам’яті.

Відомі алгоритми сортування та їх обчислювальні складності

До алгоритмів ***стійкого сортування*** належать наступні.

1. *Бульбашкове сортування* (англ. Bubble sort) — обчислювальна складність алгоритму: $O\left(n^2\right)$. Суть алгоритму: для кожної пари індексів проводиться обмін, якщо елементи розташовані не один за одним.

2. Сортування *перемішуванням* (Шейкерне, Cocktail sort, bidirectional bubble sort) – обчислювальна складність алгоритму: $O\left(n^2\right)$.

3. Сортування *вставками* (Insertion sort) — обчислювальна складність алгоритму: $O\left(n^2\right)$. Суть алгоритму: визначають місце елемента в упорядкованому списку та вставляють його туди.

4. Комбіноване сортування — обчислювальна складність алгоритму: $O\left(n^2\right)$; поєднує методи бульбашкового сортування та сортування вставками.

5. *Блочне* сортування (Кошикове, Bucket sort) — обчислювальна складність алгоритму: $O(n)$. Потрібно $O(k)$ додаткової пам'яті та знання природи даних, які підлягають сортуванню.

6. Сортування *підрахунком* (Counting sort) — обчислювальна складність алгоритму: $O(n+k)$; потрібно $O(n+k)$ додаткової пам'яті (існує три варіанти).

7. Сортування *злиттям* (Merge sort) — обчислювальна складність алгоритму: $O(n \cdot \log n)$; потрібно $O(n)$ додаткової пам'яті; упорядковують дві половини списку окремо (першу й другу), а потім — зливають їх воєдино.

8. Сортування *за допомогою двійкового дерева* (англ. Tree sort) — обчислювальна складність алгоритму: $O(n \cdot \log n)$; потрібно $O(n)$ додаткової пам'яті.

Алгоритмами *нестійкого сортування* є наступні:

- Сортування *вибором* (Selection sort) — обчислювальна складність алгоритму: $O\left(n^2\right)$; виконується пошук найменшого або найбільшого елемента та переміщення його на початок або кінець упорядкованого списку.
- Сортування *Шелла* (Shell sort) — обчислювальна складність алгоритму: $O(n \cdot \log n)$; спроба поліпшити сортування вставками.

- Сортування *гребінцем* (Comb sort) — обчислювальна складність алгоритму $O(n \cdot \log n)$.

- *Пірамідальне* сортування (Сортування купи, Heapsort) — обчислювальна складність алгоритму: $O(n \cdot \log n)$; список перетворюється на купу, береться найбільший елемент і додається в кінець списку.

- *Плавне* сортування (Smoothsort) — обчислювальна складність алгоритму: $O(n \cdot \log n)$.

- *Швидке* сортування (Quicksort), у варіанті з мінімальними витратами пам'яті складність алгоритму: $O(n \cdot \log n)$ — середня обчислювальна складність, $O(n^2)$ — найгірший випадок; широко відомий як найшвидший з відомих для впорядкування великих списків з випадковим розміщенням елементів; початковий набір даних розбивають на дві половини так, що будь-який елемент першої половини списку упорядкований щодо будь-якого елемента другої половини списку; потім алгоритм застосовують рекурсивно до кожної половини. При використанні $O(n)$ додаткової пам'яті сортування стає стійким.

- *Порозрядне* сортування — обчислювальна складність алгоритму: $O(n \cdot k)$; потрібно $O(k)$ додаткової пам’яті.

- *Сортування перестановкою* — $O(n \cdot n!)$ O($n \cdot n$!) — найбільша обчислювальна складність. Для кожної пари виконується перевірка правильного порядку та генеруються всі можливі перестановки початкового масиву.

Таблиця 2.1

Характеристики основних алгоритмів сортування

Назва	Порівнянь	Присвоювань	Пам'ять	Час
Бульбашкою	$O(N^2)$	$O(N^2)$	0	—
Вибором	$O(N^2)$	$O(N)$	0	—
Пірамідальне	$O(N \log N)$	$O(N \log N)$	0	16437
Швидке	$O(N \log N)$	$O(N log\ N)$	$O(\log N)$	5618
Злиттям	$O(N \log N)$	$O(N log\ N)$	$O(N)$	7669
Підрахунком	0	$O(N)$	$O(MAXV)$	322
Порозрядне	0	$O(kn + km)$	$O(N + km)$	1784

2.1.2. Опис алгоритмів сортування

2.1.2.1. Бульбашкове сортування

Цей метод – найпоширеніший і простий. Він вимагає мінімального обсягу пам'яті для даних, але витрати часу на його реалізацію великі. При впорядкуванні виконуються наступні операції:

1) елементи масиву порівнюються *попарно*: перший з другим; другий з третім; $i-$й з $(i+1)-$м;

2) якщо вони розміщені неправильно (при *впорядкуванні* за зростанням перший має бути меншим за другий або дорівнювати йому), то елементи міняємо місцями.

За один такий перегляд масиву при сортуванні за зростанням мінімальний елемент «виштовхнеться», принаймні, на одне місце вгору (уперед), а максимальний – переміститься в самий кінець (униз). Таким чином, мінімальний елемент, як легка бульбашка повітря у рідині поступово «спливе» вгору (на початок послідовності). Звідси – назва методу.

Оскільки останній елемент після першого перегляду масиву виявиться на своєму місці, то наступний перегляд можна закінчити раніше, тобто останніми при другому перегляді будуть порівнюватися $(n-2)-$й і $(n-1)-$ й елементи. Таким чином, при $k-$му перегляді достатньо порівняти елементи від першого до $(n-k)-$го.

За $(n-1)$ перегляд відбудеться повне упорядкування масиву при будь-якому початковому розташуванні елементів у ньому.

Алгоритм бульбашкового сортування з переглядом з початку в кінець

```
1. Задати масив з n чисел
2. Для номера_перегляду (k) від 1 до n-k виконати
2.1. Для номера_елемента (i) від 1 до n - k виконати
   Якщо елементи i-й і (i+1)-й стоять неправильно, то
   поміняти їх місцями;
3. Вивести відсортований масив.
4. Закінчити.
```

```
procedure SortPuz (var Arr : array of Integer; n : Integer);
```

```
var
  i : Integer;
  Temp : Integer;
  Flag : Boolean;
begin
  repeat
    Flag := False;
    for i := 0 to n - 1 do
      if Arr [i] > Arr [i + 1] then begin
        Temp := Arr [i];
        Arr [i] := Arr [i + 1];
        Arr [i + 1] := Temp;
        Flag := True;
      end;
  until Flag = False;
end;
```

```
procedure SortPuz (var Arr : array of Integer; n : Integer);

1.Arr[1..n]:масив цілих чисел
2. Flag = False;
3. Цикл  від i= 0 до n - 1:
     Якщо Arr [i] > Arr [i + 1]:
        Arr [i]↔Arr [i + 1]
        Flag := True ⇔

4.Йдемо на 3 Flag = False
```

Існує різновид цього алгоритму, який використовує перегляд масиву з кінця до початку. При впорядкуванні виконуються наступні операції:

1) елементи порівнюються *попарно*: останній з передостаннім (n та $(n-1)$); передостанній з третім з кінця ($(n-1)$ та $(n-2)$); $(i+1)$ – й – з i – м;

2) якщо вони розміщені неправильно (при *впорядкуванні* за зростанням менший за номером елемент має бути меншим за елемент з більшим номером або дорівнювати йому), то елементи міняємо місцями.

За перший такий перегляд масиву при сортуванні за зростанням мінімальний елемент «виштовхнеться», на одну позицію вгору (уперед), а максимальний – переміститься принаймні на одну позицію вниз. Таким чином, мінімальний елемент, як легка бульбашка повітря у рідині «спливає» вгору (на початок послідовності), а «важкі» елементи поступово осідають вниз. Звідси – назва методу.

Оскільки перший елемент після першого перегляду масиву виявиться на своєму місці, то наступний перегляд можна закінчити раніше, тобто останніми

при другому перегляді будуть порівнюватися $2-$ й і $3-$ й елементи. Таким чином, при $k-$му перегляді достатньо порівняти елементи від останнього до $k-$го. За $(n-1)$ перегляд відбудеться повне упорядкування масиву при будь-якому початковому розташуванні елементів у ньому.

Алгоритм бульбашкового сортування з кінця до початку

```
1. Задати масив з n чисел.
2. Для номера_перегляду (k) від n до k виконати
2.1. Для номера_елемента (i) від n до k виконати
   Якщо елементи i-й і (i+1)-й стоять неправильно, то
   поміняти їх місцями;
3. Вивести відсортований масив.
4. Закінчити.
```

Скорочення кількості переглядів поліпшує обчислювальну складність методу. З алгоритму можна зрозуміти, що якщо на одному з переглядів не було перестановок, то їх не буде й потім – дані вже відсортовані, а процес слід закінчити. Такий підхід дає значну економію часу при роботі з великими «майже відсортованими» масивами.

У прискореному алгоритмі будемо використовувати булеву змінну «Перестановка», якій спочатку присвоюється значення «false», тому що перестановок елементів не було. Потім, якщо в процесі перегляду масиву деякі елементи мінялися місцями, то змінна одержує значення «true». Це дозволить перервати виконання зовнішнього циклу за умови, що на черговому перегляді перестановки не виконувалися.

Алгоритм прискореного бульбашкового сортування з початку в кінець

```
1. Задати масив з n чисел.
  2.1 Номер_перегляду (k) = 1.
  2.2. Повторювати
       2.2.1. «Перестановка» = false.
       2.2.2. Для i від 1 до n-k виконати
              Якщо елементи i-й і (i+1)-й стоять
  неправильно, то
              • Поміняти місцями i-й і (i+1)-й елементи;
              • «Перестановка» = true.
       2.2.3. k = k+1
       Повторювати поки «Перестановка»=true.
  3. Вивести відсортований масив.
  4. Закінчити.
```

Існує алгоритм прискореного сортування, що використовує прохід по масиву з кінця до початку. Цей алгоритм використовує подібний до попереднього принцип скорочення кількості переглядів. Отже, як і у попередньому випадку, з алгоритму зрозуміло, що якщо на одному з переглядів не було перестановок, те їх не буде і при подальшому перегляді – дані вже відсортовані, а процес слід закінчити. Такий підхід дає значну економію часу при роботі з великими «майже відсортованими» масивами.

У даному прискореному алгоритмі також використовують булеву змінну «Перестановка», якій спочатку присвоюється значення «false», тому що перестановок елементів не було. Потім, якщо в процесі перегляду масиву деякі елементи мінялися місцями, то змінна одержує значення «true». Це дозволить перервати виконання зовнішнього циклу за умови, що на черговому перегляді перестановки не виконувалися.

Алгоритм прискореного бульбашкового сортування з кінця до початку

```
1. Задати масив з n чисел.
  2.1 Номер_перегляду (k) = n.
  2.2. Повторювати
       2.2.1. «Перестановка» = false.
       2.2.2. Для i від т до k виконати
            Якщо елементи (i+1)-й і (i)-й стоять
неправильно, то
             • Поміняти місцями (i+1)-й і-й елементи;
             • «Перестановка» = true.
       2.2.3. k = k -1
    Повторювати поки «Перестановка»=true.
  3. Вивести відсортований масив.
  4. Закінчити.
```

2.1.2.2. Сортування вставками.

Різновид сортування простими вставками

Це досить простий, але ефективний у деяких випадках метод. Спочатку вважаємо, що перший елемент перебуває на своєму місці. Далі, починаючи із другого, кожний елемент порівнюємо з усіма елементами, які стоять перед ним, і якщо вони стоять неправильно, то міняємо місцями. Таким чином, новий

елемент порівнюється та міняється місцями не з усім масивом, а тільки доти, поки не знайдеться елемент, менший за нього. Тому розглянутий алгоритм приблизно у два рази швидший за бульбашкове сортування. Для вже частково відсортованих масивів такий метод є найкращим.

Розглянемо псевдокод даного алгоритму. На вхід процедурі сортування подається масив A [1 .. n], що складається з елементів послідовності A[1], A[2], . . . , A [n], які потрібно відсортувати. A.length – розмір початкового масиву. Для сортування не потрібно залучення додаткової пам'яті, крім постійної величини для одного елемента, оскільки виконується перестановка в межах масиву. В результаті роботи процедури у вхідному масиві виявляється необхідна результуюча послідовність

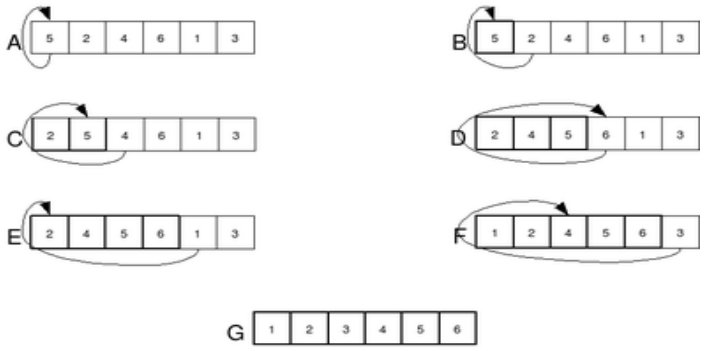


Рис. 2.1. Ілюстрація роботи алгоритму сортування вставками

Алгоритм сортування вставками

```
for j = 2 to A.length  //цикл проходів
    key = A[j]
    i = j - 1
    while i > 0 and A[i] > key
         A[i+1] = A[i] //зсуваємо елементи вправо
         i = i - 1
    A[i+1] = key // розмістили ключовий елемент на місце
```

Алгоритм мовою Pascal

```
const N = 255;
type TArray = array [1..N] of integer;
procedure InsertSort(var x: TArray);
var
  i, j, buf: integer;
begin
  for i := 2 to N do
  begin
    buf := x[i];
```

```
    j := i - 1;
    while (j >= 1) and (x[j] > buf) do
    begin
      x[j + 1] := x[j];
      j := j - 1;
    end;
    x[j + 1] := buf;
  end;
end;
```

Алгоритм мовою Java

```
public static void insertionSort2(int[] m, int a, int b) {
    int t;
    int i, j;
    for (i = a; i < b; i++) {
        t = m[i];
        for (j = i - 1; j >= a-1 && m[j] > t; j--){
            m[j + 1] = m[j];
        }
        m[j + 1] = t;
    }
}
```

Різновид сортування бінарними вставками

Алгоритм сортування простими вставками можна легко поліпшити, користуючись тим, що готова послідовність a [1], ..., a [i-1], в яку потрібно включити новий елемент, вже впорядкована. Тому місце включення можна знайти значно швидше, застосувавши бінарний пошук, який досліджує середній елемент готової послідовності і продовжує поділ навпіл, поки не буде знайдено місце включення. Модифікований алгоритм сортування називають сортуванням бінарними вставками. Обчислювальна складність алгоритму дорівнює $O(n \cdot \log n)$. Реалізація алгоритму бінарними вставками:

Алгоритм сортування бінарними вставками

```
void sort_bin_insert (int *a, int n) // Сортування бінарними
вставками
{ int x,left,right,sred;
  for (int i=1; i<n; i++)
  {
    if (a[i-1]>a[i])
     {
```

```
      x=a[i];      // x - елемент для вставки
      left=0;// ліва границя відсортованої частини масиву
      right=i-1; // права границя відсортованої частини масиву
       do {
        sred = (left+right)/2; // sred - нова "середина"
послідовності
        if (a[sred]<x ) left= sred+1;
         else right=sred-1;
       } while (left<=right); // пошук ведеться поки ліва
границя не виявиться справа від правої границі
      for (int j=i-1; j>=left;j--) a[j+1]= a[j];
      a[left]= x;
     }
  }
}
```

Різновид сортування з пошуком максимального елемента

Зазвичай застосовується для масивів, що не містять повторюваних елементів.

Алгоритм:

1) вибрати максимальний елемент масиву;

2) поміняти його місцями з останнім елементом (після цього найбільший елемент буде стояти на своєму місці);

3) повторити пункти 1-2 з рештою n-1 елементами.

Приклад. Дано масив з п'яти елементів: 5,2,7,4,6. Відсортувати за зростанням методом простого вибору. Порядок сортування показаний в таблиці 2.2.

Таблиця 2.2

Порядок кроків сортування

	Елементи масиву					*Примітки*
Початковий масив	5	2	7	4	6	max=7
Крок 1	5	2	**6**	4	7	76; max=6
Крок 2	5	2	4	6	7	46; max=5
Крок 3	4	2	5	6	7	45; max=4
Крок 4	2	4	5	6	7	42

Алгоритм сортування вибором на Java

```
public static void sort (int[] arr) {
      for (int min=0;min<arr.length-1;min++) {
            int least = min;
            for (int j=min+1;j<arr.length;j++) {
                if(arr[j] < arr[least]) {
                      least = j;
                 }
```

```
            }
            int tmp = arr[min];
            arr[min] = arr[least];
            arr[least] = tmp;
      }
}
```

Різновид сортування з пошуком мінімального елемента

При кожному перегляді масиву знаходимо мінімальний елемент і міняємо його з першим елементом. При другому проході починаємо з другого елемента та міняємо знайдений мінімальний елемент з другим і т. д.

```
procedure SortMin (var Arr : array of Integer; n : Integer);
var
  i, j : Integer;
  Min, Pos, Temp : Integer;
begin
  for i := 0 to n - 1 do begin
    Min := Arr [i];
    Pos := i;
    for j := i + 1 to n do
      if Arr [j] < Min then begin
        Min := Arr [j];
        Pos := j;
      end;
    Temp := Arr [i];
    Arr [i] := Arr [Pos];
    Arr [Pos] := Temp;
  end;
end;
```

2.1.2.3. Сортування шейкером

Якщо дані сортують не в оперативній пам'яті, а на жорсткому диску, то кількість переміщень елементів суттєво впливає на час роботи. Цей алгоритм зменшує кількість таких переміщень. За один прохід із усіх елементів вибираються мінімальний і максимальний. Потім мінімальний елемент міститься на початок масиву, а максимальний – у кінець. Таким чином, за кожний прохід два елементи стають на свої місця, тому знадобиться $n/2$ проходів, де n – кількість елементів.

На кожному проході масив проглядається зліва направо до середини, а потім – навпаки (справа наліво). У результаті, як при сортуванні «бульбашкою», при

перегляді зліва направо максимальний елемент стає на своє місце, а при зворотному – мінімальний.

Алгоритм сортування за шейкером

```
void sort_sheiker (int *m, int n) // Шейкер-сортування
{
  int left=1;                          // ліва границя
  int right=n-1;                   // права границя
  int j,t,k=n-1;
  do         // Зворотний прохід "бульбашковий" від правої
границі до лівої
  {
    for (j=right;j>=left; j--)
      if (m[j-1]>m[j])
      {
            t=m[j-1]; m[j-1]=m[j]; m[j]=t;
            k=j;// фіксування індекса останнього обміну
        }
    left=k+1;             // Ліва границя
    //Прямий прохід "бульбашковий" від лівої границі до
правої
    for (j=left; j<=right; j++)
      if (m[j-1]>m[j])
      {
         t=m[j-1]; m[j-1]=m[j]; m[j]=t;
         k=j;// фиксування індекса останнього обміну
      }
    right=k-1;// права границя
   } while (left<=right);// До того часу, поки ліва границя
не стане большою за  праву границю
}
```

Фіксування індексів останнього обміну при проходах зліва направо і справа наліво пришвидшує «шейкер»-сортування.

2.1.2.4. Сортування злиттям

Цей метод є одним з найпростіших і швидких. Особливістю його є те, що він працює з елементами масиву послідовно, завдяки чому може використовуватися, наприклад, для даних на жорсткому диску. Крім того, алгоритм, може бути ефективний для сортування таких структур, як зв'язані списки. Він був запропонований Джоном фон Нейманом у 1945 році.

Алгоритм використовує методику «розділяй і пануй», яка може бути представлена у вигляді трьох стадій:

1) поділ — дані розбивають на дві, три або більше складових, обробку яких здійснюють простими методами;

2) рекурсія — застосовують для обробки отриманих складових;

3) злиття — результати дій над складовими «зливають» у єдине ціле.

Основою алгоритму є *підзадача злиття* двох відсортованих масивів в один, теж відсортований.

Власне сортування злиттям

Процедура злиття використовує два відсортовані масиви. Очевидно, що масив з одного елемента по визначенню є відсортованим. Тоді сортування можна здійснити відповідно до рис. 2 у такий спосіб.

• Розбити елементи масиву на пари й здійснити злиття елементів кожної пари, одержавши відсортовані ланцюжки довжини 2 (крім, можливо, одного елемента, для якого не найшлося пари);

• Розбити відсортовані ланцюжки на пари, і здійснити злиття ланцюжків кожної пари;

• Якщо число відсортованих ланцюжків більше одиниці, перейти до кроку 2.

Цей алгоритм можна реалізувати у рекурсивний спосіб або за допомогою звичайного циклу. Рекурсивний алгоритм буде таким.

```
//Рекурсивна функція сортування частин масиву
public static int[] sort(int[] arr){
      if(arr.length < 2) return arr;
      int m = arr.length / 2;
      int[] arr1 = Arrays.copyOfRange(arr, 0, m);
      int[] arr2 = Arrays.copyOfRange(arr, m, arr.length);
      return merge(sort(arr1), sort(arr2));
}
// злиття двох масивів в один відсортований
 public static int[] merge(int[] arr1,int arr2[]){
      int n = arr1.length + arr2.length;
      int[] arr = new int[n];
      int i1=0;
      int i2=0;
      for(int i=0;i<n;i++){
            if(i1 == arr1.length){
```

```
                arr[i] = arr2[i2++];
            }else if(i2 == arr2.length){
                arr[i] = arr1[i1++];
            }else{
                if(arr1[i1] < arr2[i2]){
                    arr[i] = arr1[i1++];
                }else{
                    arr[i] = arr2[i2++];
                }
            }
        }
        return arr;
}
```

2.1.2.5. Сортування Боуза- Нельсона

Цей метод реалізує один зі способів злиття. Він зручний для сортування великих масивів, що перебувають у зовнішній пам’яті з послідовним доступом (наприклад, вінчестерах). Масив *A* розбивається на дві половини: *B* і *C*. Ці половини зливаються в упорядковані пари, поєднуючи перші елементи з *B* і *C* у першу пару, другі – у другу і т. д. Отриманому масиву надається ім'я *A*, після чого операція повторюється. При цьому пари зливаються в упорядковані четвірки. Попередні кроки повторюються: четвірки зливаються у вісімки і т. д., поки не буде впорядкований увесь масив, тому що довжини частин щораз подвоюються. Якщо розмір масиву непарний, або на деякому кроці з’являться неповні частини, то виконують окремо злиття початків, кінців і центральної частин. Описаний алгоритм можна проілюструвати наступним прикладом.

Приклад. Початкова послідовність: *A* = 44 55 12 42 94 18 06 67

Крок 1

Перша половина *B* = 44 55 12 42

Друга половина *З* = 94 18 06 67

A = 44 94 18 55 06 12 42 67 – для наочності непарні пари виділені.

Крок 2

Перша половина *B* = 44 94 18 55

Друга половина *З* =**06 12** 42 67

A = 06 12 44 94 18 42 55 67 – виділені впорядковані четвірки.

Крок 3

Перша половина *B* = 06 12 44 94

Друга половина *З* = 18 42 55 67

A = 06 12 18 42 44 55 67 94 – відсортований масив.

Судячи з опису, алгоритм найпростіше реалізувати за допомогою рекурсії, причому, на відміну від традиційного сортування злиттям, він не вимагає додаткової пам'яті.

1. Злиття

1.1. Початок рекурсії. Якщо j + r ≤ n, то

1.1.1. якщо m = 1, то

а) Якщо A [j]> A [j + r], то поміняти місцями A [i] і A [j + r].

1.2. інакше

1.2.1. m = m / 2.

1.2.2. Виконати злиття «початків» від j до m з відстанню r.

1.2.3. Якщо j + r + m ≤ n, то виконати злиття «кінців» від j + m до m з відстанню r.

1.2.4. Виконати злиття «центральних частин» від j + m до m з відстанню m - r.

2. Основна частина рекурсії

2.1. m = 1.

2.2. Повторювати

2.3. Цикл злиття списків однакового розміру

2.3.1. j = 1.

2.3.2. Поки j + m ≤n

1) Виконати злиття (j, m, m);

2) j = j + 2m

2.3.3. Подвоєння розміру списку перед початком нового проходу: m = 2m.

Кінець циклу 2.2, що реалізує всі злиття, поки m <n.

3. Вивести масив A.

4. Закінчити.

Число порівнянь і пересилань при реалізації сортування методом Боуза-Нельсона в найгіршому випадку пропорційне `n log n`, тобто складність алгоритму `O(n log n)`. Додаткова пам'ять не потрібна.

```
1   Const n=200;
2
3   Type
4   tipkl=word;
5   tip = Record
6   kl: tipkl;
7   z:Array[1..50] of real
8   End;
9
10  Var
11  A: Array[1..n] of tip;
12  j:word;
13
14  Procedure Bose (Var AA; voz:Boolean);
15  Var
16  m,j:word; x:tip; {tip - тип сортованих
17  записів}
18  A: Array [1..65520 div Sizeof(tip)] of
19  tip Absolute AA;
20  Procedure Sli(j,r,m: word); { r -
21  відстань між початками частин, які
22  зливаються, а m - їх розмір, j -
23  найменший номер запису}
24  Begin
25  if j+r<=n Then
26  If m=1 Then
27  Begin
28  If voz Xor (A[j].kl < A[j+r].kl) Then
29  begin
30  x:=A[j];
31  A[j]:= A[j+r];
32  A[j+r]:=x
33  end
34  end
35  else
36  begin
37  m:=m div 2;
38  Sli(j,r,m); {Злиття "початків"}
39  If j+r+m<=n Then
40  Sli(j+m,r,m); {Злиття "кінців"}
```

```
41  Sli(j+m,r-m,m) End {Злиття  центральної
42  частини}
43  end{блоку Sli};
44  begin
45  m:=1;
46  Repeat
47  j:=1; {Цикл злиття списків однакового
48  розміру: }
49  While j+m<=n do
50  begin
51  Sli(j,m,m);
52  j:=j+m+m
53  End;
54  m:=m+m {Подвоєння розміру списка перед
55  початком нового проходу}
56  Until m >= n {Кінець циклу, який
57  реалізує все дерево злиттів}
58  end{блоку Bose};
59  begin
60  Randomize;
61  For j:=1 to n do
62  begin
63  A[j].kl:= Random(65535);
    Write(A[j].kl:8);
    end;
    Readln;
    Bose(A,true);
    For j:=1 to n do
    Write(A[j].kl:8);
    Readln
    END.
```

2.1.2.6. Швидке сортування

Метод використовує той факт, що число перестановок у масиві може різко скоротитися, якщо міняти місцями елементи, що перебувають на великій відстані. Він реалізує метод «розділяй і пануй». Для цього звичайно в середині масиву вибирається один елемент (опорний). Далі ліворуч від опорного розташовують усі елементи, менше його, а праворуч – більше. Потім той же приймання застосовують до половинок масиву. Процес закінчується, коли в частинах масиву залишиться по одному елементу.

В алгоритмі використовуються два різноспрямовані процеси. Перший виконується від початку масиву й шукає елемент, більший опорного. Другий -

працює з кінця й шукає елемент, менший опорного. Як тільки такі елементи знайдені, проводиться їхній обмін місцями. Далі пошук триває з того місця, де процеси зупинилися.

Таким чином, коли процеси зустрічаються, будь-який елемент у першій частині менше кожного в другий. Це значить, що їх уже порівнювати один з одним не прийде. Залишається тільки провести таку ж операцію стосовно отриманих половин, і так далі, поки в черговій частині масиву не залишиться один елемент. Це буде означати, що масив відсортований. Очевидно, що найбільш зручний спосіб реалізації розглянутого методу – рекурсивний.

Загальний алгоритм можна представити так.

- З масиву вибирається деякий опорний елемент, *Опорний* = $a[i]$, $i=n/2$.
- Запускається процедура поділу масиву, яка переміщає всі елементи, менші, або рівні *Опорному*, вліво від нього, а всі більші або рівні *Опорному*, – вправо.
- Тепер масив складається з двох підмасивів, причому довжина лівого менша, або дорівнює довжині правого.
- Для обох підмасивів, якщо у них більше двох елементів, рекурсивно виконується та ж процедура.

Наприкінці отримаємо повністю відсортовану послідовність.

Рекурсивний алгоритм процедури швидкого сортування

```
void Qsort(int a[],int L,int R) // або ...int *a
{ int i=L,j=R,w,x;
  x=a[(L+R)/2];
  do { while (a[i]<x) i++;
       while (a[j]>x) j--;
       if(i<=j)
           {w=a[i]; a[i]=a[j]; a[j]=w;
           i++; j--;}
  } while (i<j);
  if (L<j) Qsort(a,L,j);
  if (i<R) Qsort(a,i,R);
}
void main()
{
     const int nmax=20;
     int n, a[nmax];
     input(a,n);
```

```
cout<<"Початковий масив:\n";
print(a,n);
Qsort(a,0,n-1);
cout<<"Відсортований масив:\n";
print(a,n);
```

2.1.2.7. Сортування Шелла

Метод є прискореним варіантом сортування вставками. При цьому прискорення досягають за рахунок збільшення відстані, на яку переміщають елементи. Початковий масив ділять на *d* частин, що містять $\frac{n}{d}$ елементів кожний (останній підмасив може бути коротшим). Підмасиви містять елементи з номерами $[0, d, 2d$ і т. д.], $[1, d+1, 2d+1$ і т. д.], $[2,\ d+2, 2d+2$ і т. д.] і т. д.

Спочатку порівнюють й упорядковують за допомогою алгоритму вставок елементи, що розташовані один від одного на відстані *d*, тобто номери, що мають, 0 і 1, d і $d+1$, $2d$ і $2d+1$ і т. д. Потім процедуру повторюють при менших значеннях *d*, наприклад, $\frac{d}{2}$. Завершується алгоритм упорядкуванням елементів при *d* = 1. Ми розглянемо сортування Шелла для початкового значення *d,* що дорівнює *n* /2, і будемо послідовно зменшувати його вдвічі.

Незважаючи на те, що сортування Шелла в багатьох випадках повільніше, ніж швидке сортування, воно має ряд переваг:

- відсутність потреби в пам'яті під стек;
- відсутність деградації при невдалих наборах даних — швидке сортування легко деградує до $O\left(n^2\right)$, що гірше, ніж гірший гарантований час для сортування Шелла.

Нехай даний список $A=\left(32,95,16,82,24,66,35,19,75,54,40,43,93,68\right)$. Виконаємо його сортування методом Шелла, як значення d візьмемо $5,3,1$.

На першому кроці сортуються підсписки A, складені із усіх елементів A, що відрізняються на 5 позицій, тобто підсписки $A_{5,1}=\left(32,66,40\right)$, $A_{5,2}=\left(95,35,43\right)$, $A_{5,3}=\left(16,19,93\right)$, $A_{5,4}=\left(82,75,68\right)$, $A_{5,5}=\left(25,54\right)$.

В отриманому списку на другому кроці знову сортують підсписки з віддалених на 3 позиції елементів.

Процес завершується звичайним сортуванням вставками одержаного списку. Середній час роботи алгоритму залежить від довжин проміжків — d, на яких будуть перебувати елементи початкового масиву з розміром N на кожному кроці алгоритму.

Принцип сортування Шелла

```
void sort_shell(int a[], int max)       // сортування Шелла
  {
    for (int gap = max/2; gap>0; gap/=2)    // вибір інтервалу
      for (int i=gap; i<max; i++)           // прохід масива
// порівняння пар, які розміщені на gap одна від одної
        for (int j=i-gap; j>=0 && less(a[j+gap],a[j]); j-=gap)
          swap(a[j], a[j+gap]);
  }
```

Алгоритм сортування Шелла

```
#include <iostream>
using namespace std;
void print (int *m, int n)
{
     for (int i=0;i<n;i++)
          cout<<m[i]<<" ";
     cout<<"\n";
}
void ShellSort(int data[], int rzm_data)
{  int step,i,j,c;
    step = rzm_data;           // Крок пошуку та вставки
    do
    {  // Обчислення нового кроку
        step = step / 3 + 1;
// Виконуємо сортування простими вставками з заданим кроком
        int k=0;
         while (k<step) //сортуємо з кроком step послідовності,
перший елемент яких номер від 0 до step-1
         {
         for (i=k+step; i<rzm_data; i=i+step) //переміщуємося з
кроком step по поточній послідовності
              {
              c = data[i];
              j = i-step;
```

```
              while (j >= 0 && data[j] > c) //шукаємо місце
вставки елемента, якщо він менший за попередній
                 {  data[j+step] = data[j];
                    j = j-step;
                 }
              data[j+step] = c; //виконуємо вставку
         }
          k++; //переходимо до наступної послідовності для
сортування
            }
     }
    while (step !=1);
}
void input(int *m, int &n)   // Им'я масиву є адресою його
початку.
{                                    // або
void input(int m[], int &n)
     cout<<"Введіть кількість елементів масиву ";
     cin>>n;
     for (int i=0;i<n;i++)
     {
     cout<<"a["<<i<<"]=";
     cin>>m[i];// Конструкція a[i] еквівалентна *(a + i)
     }

}

void main()
{
     const int nmax=20;
     int n, a[nmax];
     input(a,n);
     cout<<"Початковий масив:\n";
     print(a,n);
     ShellSort(a,n);
     cout<<"Відсортований масив:\n";
     print(a,n);
}
```

2.1.2.8. Сортування Хоара

Сортування Хоара – це метод, який визнано одним із кращих методів сортування серед існуючих. У методі Хоара спочатку виділяють базовий елемент, щодо якого ключі з більшою вагою перекидаються вправо, а з меншою вліво. Базовий елемент порівнюється із протилежним елементом.

Як базовий елемент зручно брати крайні елементи.

Розглянемо приклад. Дана множина {9,6,3,4,10,8,2,7}

Беремо 9 як базовий елемент. Порівнюємо 9 з протилежним елементом, у цьому випадку це 7. 7 менше, ніж 9, отже, елементи міняються місцями.

{**7**,6,3,4,10,8,2,**9**}

Далі починаємо послідовно порівнювати елементи з 9, і міняти їх місцями залежно від порівняння.

{7,**6**,3,4,10,8,2,**9**}

{7,6,**3**,4,10,8,2,**9**}

{7,6,3,**4**,10,8,2,**9**}

{7,6,3,4,**9**,8,2,**10**} – 9 і 10 міняємо місцями.

{7,6,3,4,**8**,**9**,2,10} – 9 і 8 міняємо місцями.

{7,6,3,4,8,**2**,**9**,10} – 2 і 9 міняємо місцями.

Після такого перекидання елементів увесь масив розбивається на дві підмножини, розділені елементом 9.

{7,6,3,4,8,2}

{10}

Далі по вже відпрацьованому алгоритму сортуються ці підмножини. Підмножина з одного елемента природно можна не сортувати. Вибираємо в першій підмножині базовий елемент 7.

{7,6,3,4,8,2}

{**2**,6,3,4,8,**7**} – міняємо місцями 2 і 7.

{2,**6**,3,4,8,**7**}

{2,6,**3**,4,8,**7**}

{2,6,3,**4**,8,**7**}

{2,6,3,4,**7**,**8**} – міняємо місцями 7 і 8

Одержали знову дві підмножини.

{2,6,3,4}

{8}

А далі все відбувається аналогічно.

У результаті можна створити таку програму:

```
int HoareSort(int *array, int left, int right)
{
      long base, opposite, p;
      int c;
      base=left;
      opposite=right;
      while (base!=opposite){
            if ((array[base]>array[opposite])^(base>opposite)){

                  c=array[base];
                  array[base]=array[opposite];
                  array[opposite]=c;

                  p=base;
                  base=opposite;
                  if (p<opposite)
                        opposite=p+1; else opposite=p-1;
            } else {
                  if (base<opposite)
                        opposite--; else opposite++;
            };
      };

      if (left<base-1) HoareSort(array,left,base-1);
      if (base+1<right) HoareSort(array,base+1,right);
};
```

Принцип сортування Хоара:

```
void sort_hoare(int a[], int l, int r)   // сортування Хоара
  {
    int i=l, j=r, step=-1, condition=1;
    if (l>=r) return;                  // сортувати немає чого

    do {                  // сортуємо з лівої границі до правої
      if (condition == less(a[j],a[i]))
        {
          swap(a[i], a[j]);              // перестановка чисел
          swap(i, j);                 // обмін місцями індексів
          step = -step;                // а зараз - в інший бік
          condition = !condition;// заміна умови на протилежну
        }
      j += step;                         // переміщуємо індекс
    } while (j!=i);                // поки індекси не зійдуться
    sort_hoare(a, l, i-1);                // ліва підмножина
    sort_hoare(a, i+1, r);                // права підмножина
```

```
}
```

2.1.2.9. Пірамідальне сортування

Алгоритм пірамідального сортування має обчислювальну складність $O(n\log n)$. Будемо вибирати з масиву найбільший елемент, і записувати його на початок уже відсортованої частини масиву (сортування вибором у зворотному порядку). Тобто, відсортований масив буде будуватися від кінця до початку. Така хитрість необхідна для того, щоб не було необхідності в додатковій пам'яті та для прискорення роботи алгоритму — несортована частина буде розташовуватися на початку масиву, а відсортована частина перебуватиме в кінці.

Властивість несортованої частини (купи) максимумів: елементи з індексами $i+1$ та $i+2$ не більші за елемент з індексом i (природно, якщо $i+1$ та $i+2$ лежать у межах купи). Нехай n — розмір купи, тоді друга половина масиву (елементи від $\frac{n}{2}+1$ до n) задовольняють властивості купи. Для інших елементів викликаємо функцію «проштовхування» по купі, починаючи з $\frac{n}{2}$ *до* 0.

```
    void down_heap(int a[], int k, int n)
{
  int temp=a[k];
  while (k*2+1<n){
    y=k*2+1;
    if (y < n-1 && a[y] < a[y+1]) y++;
    if (temp >= a[y]) break;
    a [k] =a [y];
 }
 a[k]=temp;
}
```

Ця функція одержує посилання на масив, номер елемента, який необхідно проштовхнути, та розмір купи. У неї є невеликі відмінності від звичайних функцій роботи з купою. Номер мінімального предку зберігається в змінній y, якщо необхідність в обмінах закінчена, то ми виходимо з циклу й записуємо «просіяну» змінну на призначене їй місце.

Саме сортування буде складатися зі створення купи з масиву й N переносів елементів з вершини купи з наступним відновленням властивості купи:

```
void heap_sort(int a[], int n) {
  int i, temp;
  for(i=n/2-1; i >= 0; i--) down_heap(a, i, n);
  for(i=n-1; i > 0; i--) {
   temp=a[i]; a[i]=a[0]; a[0]=temp;
   down_heap(a, 0, i);
   }
}
```

Усього в процесі роботи алгоритму буде виконано $3 \cdot \frac{N}{2} - 2$ викликів функції down_heap, кожний з яких має обчислювальну складність $O(\log N)$. Таким чином, ми й одержуємо шукану складність в $O(N\log N)$, не використовуючи при цьому додаткової пам'яті. Кількість присвоювань також становить $O(N\log N)$.

Пірамідальне сортування слід здійснювати, якщо з умови задачі зрозуміло, що єдиною дозволеною операцією є «проштовхування» елемента у купі, або у випадку відсутності додаткової пам'яті.

2.1.2.10. Сортування підрахунком

Таке сортування можна використовувати тільки для сортування чисел та за умови, що кожне число в масиві зустрічається у великій кількості екземплярів. Припустимо, що нам необхідно відсортувати масив чисел, які знаходяться у діапазоні від 0 до 99. Заведемо масив розміром у 100 елементів у якому будемо запам’ятовувати скільки разів зустрічається кожне число (тобто при знаходженні числа будемо збільшувати елемент масиву з індексом, що дорівнює цьому числу, на 1). Після цього пройдемо по числах від 0 до 99 і виведемо кожне число стільки разів, скільки воно зустрічалося. Сортування реалізують таким чином:

```
for (1=0; i<MAXV; i++)
 c[i] = 0;
for (i=0; i<n; i++)
```

```
 c[a[1] ]++;
k=0;
for (1=0; i<MAXV; i++)
for (j =0; j<c [i]; j++)
a[k++] =i;
```

Тут $MAXV$ – максимальне значення, яке може зустрічатися (тобто всі числа масиву повинні лежати в межах від 0 до $MAXV-1$).

Алгоритм використовує $O(MAXV)$ додаткової пам'яті та має складність $O(N+MAXV)$. Його застосування дає відмінний результат, якщо $MAXV$ менший, ніж кількість елементів у масиві.

2.1.2.11. Порозрядне сортування

Алгоритм сортування підрахунком надзвичайно привабливий своєю високою продуктивністю але вона погіршується при зростанні $MAXV$, також різко зростають вимоги до додаткової пам'яті. Фактично, неможливо здійснити сортування підрахунком для змінних типу `unsigned int` *(*$MAXV$ при цьому дорівнює 2^{32}).

Варіант сортування для unsined int

```
void radix_int(unsigned* begin, int size, unsigned bit =
0x80000000)
{
    if (!bit)
        return;

    if (size < 2)
        return;

    int last = 0;
    for (int i = 0; i < size; i++)
    {
        if ((begin[i] & bit) == 0)
            swap(begin[last++], begin[i]);
    }

    radix_int(begin,      last,      bit >> 1);
    radix_int(begin+last, size-last, bit >> 1);
}
```

2.2. Вибір варіанту завдання на лабораторну роботу

Відповідно до номеру у списку групи вибрати з таблиці варіант завдання для виконання лабораторної роботи.

Таблиця 2.3

Варіанти виконання лабораторної роботи № 2

Вар.	Назва алгоритму	Алгоритмічна складність
1	Бульбашкове сортування з початку до кінця	$O\left(^{2}\right)$
2	Алгоритм 2.1. Сортування простими вставками	$O\left(n^{2}\right)$
3	Сортування вибором з пошуком максимального елемента	$O\left(n^{2}\right)$
4	Сортування Боуза-Нельсона	$O\left(n\log n\right)$
5	Сортування Хоара	$O\left(n\log n\right)$
6	Порозрядне сортування.	$O\left(nk\right)$
7	Швидке сортування	$O\left(n\log n\right)$
8	Пірамідальне сортування	$O\left(n\log n\right)$
9	Сортування шейкером	$O\left(n^{2}\right)$
10	Прискорене бульбашкове сортування з початку до кінця	$O\left(n^{2}\right)$
11	Сортування вибором з пошуком мінімального елемента	$O\left(n^{2}\right)$
12	Сортування злиттям	$O\left(n\log n\right)$
13	Сортування Хоара	$O\left(n\log n\right)$
14	Швидке сортування	$O\left(n\log n\right)$
15	Сортування за Шейкером	$O\left(n^{2}\right)$
16	Сортування бінарними вставками	$O\left(n\log n\right)$
17	Сортування вибором з пошуком максимального елемента	$O\left(n^{2}\right)$
18	Сортування підрахунком	$O\left(n\right)$
19	Сортування Шелла	$O\left(n\log n\right)$
20	Прискорене бульбашкове сортування з кінця до початку	$O\left(n^{2}\right)$
21	Порозрядне сортування	$O\left(nk\right)$

22	Сортування Хоара	$O(n\log n)$
23	Сортування злиттям	$O(n\log n)$
24	Сортування простими вставками	$O(n^2)$
25	Сортування Боуза-Нельсона	$O(n\log n)$
26	Пірамідальне сортування	$O(n\log n)$
27	Швидке сортування.	$O(n\log n)$
28	Сортування за Шейкером	$O(n^2)$
29	Сортування бінарними вставками	$O(n\log n)$
30	Бульбашкове сортування з кінця до початку	$O(n^2)$

2.3. Вимоги до програмного забезпечення:

- Введення даних з клавіатури і з зовнішнього файлу;
- Перевірка коректності введених даних;
- Інтерфейс у стилі GUI.
- Вимірювання виконувати з кроком 1000 елементів масиву
- Порівняти графіки залежності у одиницях часу та у кількості виконаних операцій

2.4. Зміст звіту:

- Титульний лист;
- Тема завдання;
- Завдання;
- Блок-схеми алгоритмів;
- Роздруківка тексту програми;
- Роздруківка результатів виконання програми;
- Аналіз результатів.

2.5. Контрольні запитання

- Які алгоритми сортування вам відомі?

- Дати визначення обчислювальної складності алгоритму?
- Дайте визначення асимптотичної оцінки складності Ω.
- Дайте визначення асимптотичних оцінок складності Ω та O.
- Як на практиці визначити обчислювальну складність алгоритму сортування?

2.6. Інформаційні ресурси

1. Про алгоритми сортування: https://foxminded.ua/alhorytmy-sortuvannia/
2. Алгоритми сортування в теорії та на практиці: https://javarush.com/ua/groups/posts/uk.1997.algoritmi-sortuvannja-v-teor-ta-na-praktic
3. Алгоритми сортування в C/C++/C# - що треба знати початківцю: https://buki.com.ua/blogs/algoritmi-sortuvannia-v-ssc-shho-treba-znati-pocatkivciu/
4. Розбираємо швидке сортування: https://campus.epam.ua/ua/blog/483

Розділ 3. ЛАБОРАТОРНА РОБОТА № 3

Тема: «Інтерполяція функцій. Інтерполяційні многочлени».

Мета: Ознайомлення з інтерполяційними формулами Лагранжа, Ньютона, рекурентним співвідношенням Ейткена, методами оцінки похибки інтерполяції.

Завдання: Закріплення, поглиблення і розширення знань студентів при вирішенні практичних обчислювальних завдань. Оволодіння обчислювальними методами і практичними методами оцінки похибки обчислень. Придбання умінь і навичок при програмуванні та налагодженні обчислювальних завдань на комп'ютері.

3.1. Теоретичні відомості до лабораторної роботи №3

Інтерполяція функцій є одним із фундаментальних розділів обчислювальної математики. До появи комп'ютерів для багатьох практичних обчислень застосовувалися таблиці елементарних функцій (синусів, логарифмів і т. ін.). Для отримання досить точних результатів при значеннях аргументів, розташованих між вузловими точками, для яких дані табличні значення функції, вирішувалося завдання інтерполяції (у перекладі - «між полюсами»). У найбільш простому випадку сусідні точки графіка цієї функції з'єднувалися відрізком прямої (лінійна інтерполяція). Власне, густота точок таблиці і вибиралася з огляду на інтерполяцію. Наприклад, вираз «чотиризначні таблиці» означає, що для будь-якого значення аргументу, а не тільки для зазначених в таблиці в якості вузлових, шляхом інтерполяції, (як правило, лінійної) можна отримати табличне значення функції з точністю до чотирьох значущих цифр. Основоположне значення задачі інтерполяції пояснюється також тим, що багато методів розв'язання задач чисельного диференціювання, інтегрування, розв’язування диференціальних та інтегральних рівнянь зводяться до диференціювання і інтегрування інтерполяційного многочлена. Після появи комп'ютерів значення задачі інтерполяції функцій, заданих таблицею, не втратило актуальності, оскільки в результаті чисельного вирішення складних задач отримують ряд значень шуканої функції при різних значеннях вхідного параметра. Одержання значної кількості таких значень пов'язане з великими витратами машинного часу. Застосування

інтерполяції в цьому випадку дозволяє істотно зменшити ці витрати. Однак, на відміну від задачі інтерполяції відомої функції, в цьому випадку інформація про шукану функцію обмежується таблицею її значень. Ця задача є некоректною, оскільки існує нескінченна множина функцій, що мають задане кінцеве число відомих значень. З подібними ж проблемами доводиться стикатися і при розв’язуванні диференціальних та інтегральних рівнянь. Тому можна сформулювати таку тезу: в обчислювальній математиці не існує коректних задач. Існують тільки коректно поставлені завдання, тобто штучно придумані умови, які на практиці, як правило, не виконуються у зв'язку з браком інформації про те, що є шуканим. У зв'язку з цим задача інтерполяції в реальних умовах є найважливішою проблемою обчислювальної математики, розв’язок якої дозволяє знайти ключ до вирішення багатьох інших завдань, необхідних для практики.

Постановка завдання

Нехай деяка функція $f(x)$ задана своїми значеннями $y_j = f(x_j)$ на дискретній множині точок x_j , $j = 0,...,m$. Потрібно наближено визначити аналітичний вигляд цієї функції і тим самим отримати можливість обчислити її значення в проміжних точках $x \in (x_j, x_{j+1})$. Графік, що ілюструє дану задачу, зображений на рис. 3.1.

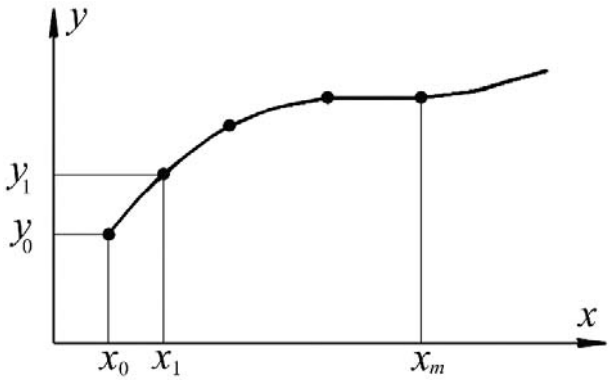


Рис. 3.1. Інтерполяція функцій

Інтерполюючу функцію будемо шукати у вигляді алгебраїчного многочлена.

$$P_n(x) = \sum_{i=0}^{n} a_i x^i . \tag{3.1}$$

Оскільки многочлен $P_n(x)$ у вузлових точках повинен збігатися з заданими значеннями функції, то завдання зводиться до вирішення системи лінійних алгебраїчних рівнянь

$$\sum_{i=0}^{n} a_i x_j^i = y_j, \quad j = k,\ldots,k+n \tag{3.2}$$

щодо невідомих a_1 (k - номер початкової вузлової точки, використовуваної в даному розрахунку). Ця система рівнянь має єдине рішення (якщо *mn*+*k*, і всі x_j різні), оскільки визначник цієї системи – визначник Вандермонда – не дорівнює нулю.

3.1.1. Інтерполяційний многочлен Лагранжа

Розв'язування системи рівнянь (3.2) можна представити у формі інтерполяційного многочлена Лагранжа:

$$P_n\left(x\right) = L_n\left(x\right) = \sum_{j=k}^{k+n} y_j \prod_{\substack{i=k \\ i\neq j}}^{k+n} \frac{x - x_i}{x_j - x_i}. \tag{3.3}$$

В окремому випадку $n = 1$ (лінійна інтерполяція)

$$L_1\left(x\right) = \frac{x - x_{k+1}}{x_k - x_{k+1}} y_k + \frac{x - x_k}{x_{k+1} - x_k} y_{k+1}$$

а при $n = 2$

$$L_2\left(x\right) = \frac{\left(x - x_{k+1}\right)\left(x - x_{k+2}\right)}{\left(x_k - x_{k+1}\right)\left(x_k - x_{k+2}\right)} y_k + \frac{\left(x - x_k\right)\left(x - x_{k+2}\right)}{\left(x_{k+1} - x_k\right)\left(x_{k+1} - x_{k+2}\right)} y_{k+1} +$$

$$+ \frac{\left(x - x_k\right)\left(x - x_{k+1}\right)}{\left(x_{k+2} - x_k\right)\left(x_{k+2} - x_{k+1}\right)} y_{k+2}$$

Неважко помітити, що структура цих формул така, що для кожної вузлової точки $x = x_j$ з вхідного в набору використовуваних формулою вузлових точок, тільки один доданок відмінний від нуля і саме той, в який входить y_j. Крім того, дріб, що входить в цей відмінний від нуля доданок, при $x = x_j$ дорівнює одиниці. Тому $L_n\left(x_j\right) = y_j$.

3.1.2. Інтерполяційний многочлен Ньютона

Спочатку необхідно дати кілька визначень. Для спрощення запису введемо позначення: $f_k = f\left(x_k\right)$. Скінченною різницею першого порядку функції f в точці x_k називається величина

$$\Delta f_k = f_{k+1} - f_k$$

а скінченною різницею n-го порядку $\left(n-1\right)$ величина

$$\Delta^n f_k = \Delta^{n-1} f_{k+1} - \Delta^{n-1} f_k \tag{3.4}$$

Звідси, зокрема, випливає, що різниця другого порядку

$$\Delta^2 f_k = \Delta f_{k+1} - \Delta f_k = (f_{k+2} - f_{k+1}) - (f_{k+1} - f_k) = f_k - 2f_{k+1} + f_{k+2}$$

Розділеними різницями нульового порядку називають значення функції f_k. Розділеною різницею першого порядку називають величину

$$f\left(x_k, x_{k+1}\right) = f\left(x_{k+1}, x_k\right) = \frac{f_{k+1} - f_k}{x_{k+1} - x_k} = \frac{f_k}{x_k - x_{k+1}} + \frac{f_{k+1}}{x_{k+1} - x_k}$$

Розділену різницю n-го порядку визначають через розділені різниці $\left(n-1\right)$-го порядку за рекурентною формулою

$$f\left(x_k, x_{k+1}, x_{k+n}\right) = \frac{f\left(x_{k+1}, x_{k+2}, x_{k+n}\right) - f\left(x_k, x_{k+1}, x_{k+n-1}\right)}{x_{k+n} - x_k} \tag{3.5}$$

Інший вираз розділеної різниці n-го порядку

$$f\left(x_k, x_{k+1}, x_{k+n}\right) = \sum_{j=k}^{k+n} f_j \left(\prod_{\substack{i=k \\ i \neq j}}^{k+n} \left(x_j - x_i\right) \right)^{-1} \tag{3.6}$$

Інтерполяційним многочленом Ньютона називають алгебраїчний многочлен

$$l_n\left(x\right) = f\left(x_k\right) + \left(x - x_k\right) f\left(x_k, x_{k+1}\right) + \left(x - x_k\right)\left(x - x_{k+1}\right) f\left(x_k, x_{k+1}, x_{k+2}\right) + \ldots$$
$$\ldots + \left(x - x_k\right)\left(x - x_{k+1}\right)\ldots\left(x - x_{k+n-1}\right) f\left(x_k, x_{k+1}, \ldots, x_{k+n}\right) \tag{3.7}$$

Цей многочлен тотожно дорівнює многочлену n-го ступеня, записаному в формі Лагранжа або в будь-якій іншій формі в силу єдиності інтерполяційного

многочлена. Проте така форма запису дозволяє при необхідності збільшення ступеня многочлена не перебудовувати весь многочлен заново, а тільки додавати додаткові доданки.

3.1.3. Схема Ейткена

Нехай функція $f(x)$ і розташування вузлів $x_0, x_1, ..., x_n$ на відрізку інтерполяції $[a;b]$ такі, що інтерполяційний процес має збіжність. І нехай потрібно знайти не загальний вираз $L_n(x)$, а лише його значення при конкретних x, тобто вирішується задача обчислення окремих наближених значень функції $f(x)$ за допомогою обчислення відповідних їм значень інтерполяційного многочлена Лагранжа $L_n(x)$. Розглянемо даний процес більш детально і побудуємо обчислювальну схему для отримання наближеного значення таблично заданої функції $f(x)$ в заданій точці $x = \tilde{x}$, в основу якої буде покладена інтерполяція Лагранжа на сітці вузлів $x_0, x_1, ..., x_n$. Організація обчислень за цією схемою матиме ітераційний характер, кожен крок якої полягає в обчисленні деякого визначника другого порядку.

Нехай дано дві точки на кривій $y = f(x)$: (x_0, y_0) і (x_1, y_1). Побудуємо функцію $M_{0,1}(x)$:

$$M_{0,1} = \frac{1}{x_1 - x_0}\begin{vmatrix} x - x_0 & y_0 \\ x - x_1 & y_1 \end{vmatrix} = \frac{x - x_2}{x_0 - x_1} y_0 + \frac{x - x_0}{x_1 - x_0} y_1, \text{ тобто } M_{0,1}(x) = L_1(x),$$

де $L_1(x)$ - це многочлен Лагранжа першого ступеня, побудований на вузлах x_0 та x_1.

Побудуємо з використанням такого ж визначника функцію $M_{1,2}(x)$ для точок (x_1, y_1) та (x_2, y_2):

$$M_{1,2} = \frac{1}{x_2 - x_1}\begin{vmatrix} x - x_1 & y_1 \\ x - x_2 & y_2 \end{vmatrix} = \frac{x - x_2}{x_1 - x_2} y_1 + \frac{x - x_1}{x_2 - x_1} y_2$$

Функція $M_{1,2}(x)$ - це також многочлен Лагранжа першого степеню який побудований на двох точках (x_1, y_1) та (x_2, y_2).

Аналогічно можливо побудувати функції $M_{2,3}(x)$, $M_{3,4}(x)$ і так далі.

Якщо для функції $y = f(x)$ задані три точки (x_0, y_0), (x_1, y_1) та (x_2, y_2), то, використовуючи введені лінійні функції $M_{0,1}(x)$ і $M_{1,2}(x)$, можемо створити нову функцію:

$$M_{0,1,2} = \frac{1}{x_2 - x_0}\begin{vmatrix} x - x_0 & M_{0,1} \\ x - x_2 & M_{1,2} \end{vmatrix}$$

Функція $M_{0,1,2}$ — це многочлен другого степеню, який вирішує задачу параболічної інтерполяції по трьох точках (x_0, y_0), (x_1, y_1) та (x_2, y_2). Оскільки такий многочлен є єдиним, то $M_{0,1,2}(x) = L_2(x)$, де $L_2(x)$ — многочлен Лагранжа. Якщо продовжити процес побудови далі, то можемо отримати послідовність інтерполяційних многочленів Лагранжа, яка лежить в основі інтерполяційної схеми Ейткена. Член такої послідовності, що охоплює точки $x_0, x_1, ..., x_i$ має вигляд:

$$f(x) \approx M_{0,1,...,i}(x) = \frac{1}{x_i - x_0}\begin{vmatrix} x - x_0 & M_{0,1,...,i-1}(x) \\ x - x_i & M_{0,1,...,i}(x) \end{vmatrix} \tag{3.8}$$

Схема Ейткена легко реалізується на ПК. Організація обчислень за формулою (3.8) повинна бути така, що якщо заздалегідь невідомий степінь інтерполяційного многочлена, який потрібно використовувати для обчислення $y = f(x)$, то має відбуватися поступове підвищення степені інтерполяційних многочленів за рахунок підключення нових вузлів. Обчислення проводяться поки йде уточнення наближеного значення $y = f(x)$, тобто поки величина $\left|M_{0,1,...,i-1}(x) - M_{0,1,...,i}(x)\right| \to 0$. Також відмітимо, що обчислювальний процес інтерполяційної схеми Ейткена зазвичай оформляється у вигляді наступної таблиці:

Таблиця 3.1

Схема Ейткена

i	x_i	y_i	$M_{i,i+1}(x)$	$M_{i,i+1,i+2}(x)$	$M_{i,i+1,i+2,i+3}(x)$	$M_{i,i+1,i+2,i+3,i+4}(x)$
0	x_0	y_0	$M_{0,1}(x)$	$M_{0,1,2}(x)$	$M_{0,1,2,3}(x)$	$M_{0,1,2,3,4}(x)$
1	x_1	y_1	$M_{1,2}(x)$	$M_{1,2,3}(x)$	$M_{1,2,3,4}(x)$	
2	x_2	y_2	$M_{2,3}(x)$	$M_{2,3,4}(x)$		
3	x_3	y_3	$M_{3,4}(x)$			
4	x_4	y_4				

Приклад побудови інтерполяційної функції за схемою Ейткена

Нехай деяка функція $y = f(x)$ задана таблицею своїх значень:

Таблиця 3.2

Таблиця функції $y = f(x)$

x_i	-1	0	2	4	6	9
$f(x_i)$	-2	3	1	7	-1	5

Розглянемо процес обчислення наближеного значення даної функції, за **схемою Ейткена**, в точці $x = -0.5$. Для цього, скориставшись формулою (3.8) побудуємо таблицю наступного виду:

Таблиця 3.3

Інтерполяційна схема Ейткена

i	x_i	y_i	$M_{i,i+1}(-0.5)$	$M_{i,i+1,i+2}(-0.5)$	$M_{i,i+1,i+2,i+3}(-0.5)$	$M_{i,i+1,i+2,i+3,i+4}(-0.5)$	$M_{i,i+1,i+2,i+3,i+4,i+5}(-0.5)$
0	-1	-2	$M_{0,1} = 0.5$	$M_{0,1,2} = 1$	$M_{0,1,2,3} = 1.375$	$M_{0,1,2,3,4} = 1.8$	$M_{0,1,2,3,4,5} = 2.255$
1	0	3	$M_{1,2} = 3.5$	$M_{1,2,3} = 4.75$	$M_{1,2,3,4} = 7.328$	$M_{1,2,3,4,5} = 10.902$	
2	2	1	$M_{2,3} = -6.5$	$M_{2,3,4} = -26.187$	$M_{2,3,4,5} = -57.004$		
3	4	7	$M_{3,4} = 25$	$M_{3,4,5} = 60.1$			
4	6	-1	$M_{4,5} = -14$				
5	9	5					

Виходячи з того, що точка в якій ми обчислюємо наближене значення розташована між вузлами $x_0 = -1$ та $x_1 = 0$, то доцільно в якості основної послідовності значень многочленів Лагранжа брати рядок таблиці, який відповідає значенню $i = 0$, тобто в якості наближених значень будемо брати числа

$M_{0,1} = 0.5$	$M_{0,1,2} = 1$	$M_{0,1,2,3} = 1.375$	$M_{0,1,2,3,4} = 1.8$	$M_{0,1,2,3,4,5} = 2.255$

Обчисливши модуль різниці між попередніми і наступними числами цього рядка, а саме:

$$0.5;\ 0.375;\ 0.425;\ 0.425$$

бачимо, що після обчислення третього наближення, а саме $M_{0,1,2,3} = 1.375$ різниця перестала зменшуватися. Тобто подальший рахунок не має сенсу і отримати більш точне значення заданої функції $f(x)$ в точці $x = 0.5$, ніж $M_{0,1,2,3} = 1.375$ не вдасться. Тому приймаємо $f(0.5) \approx 1.375$.

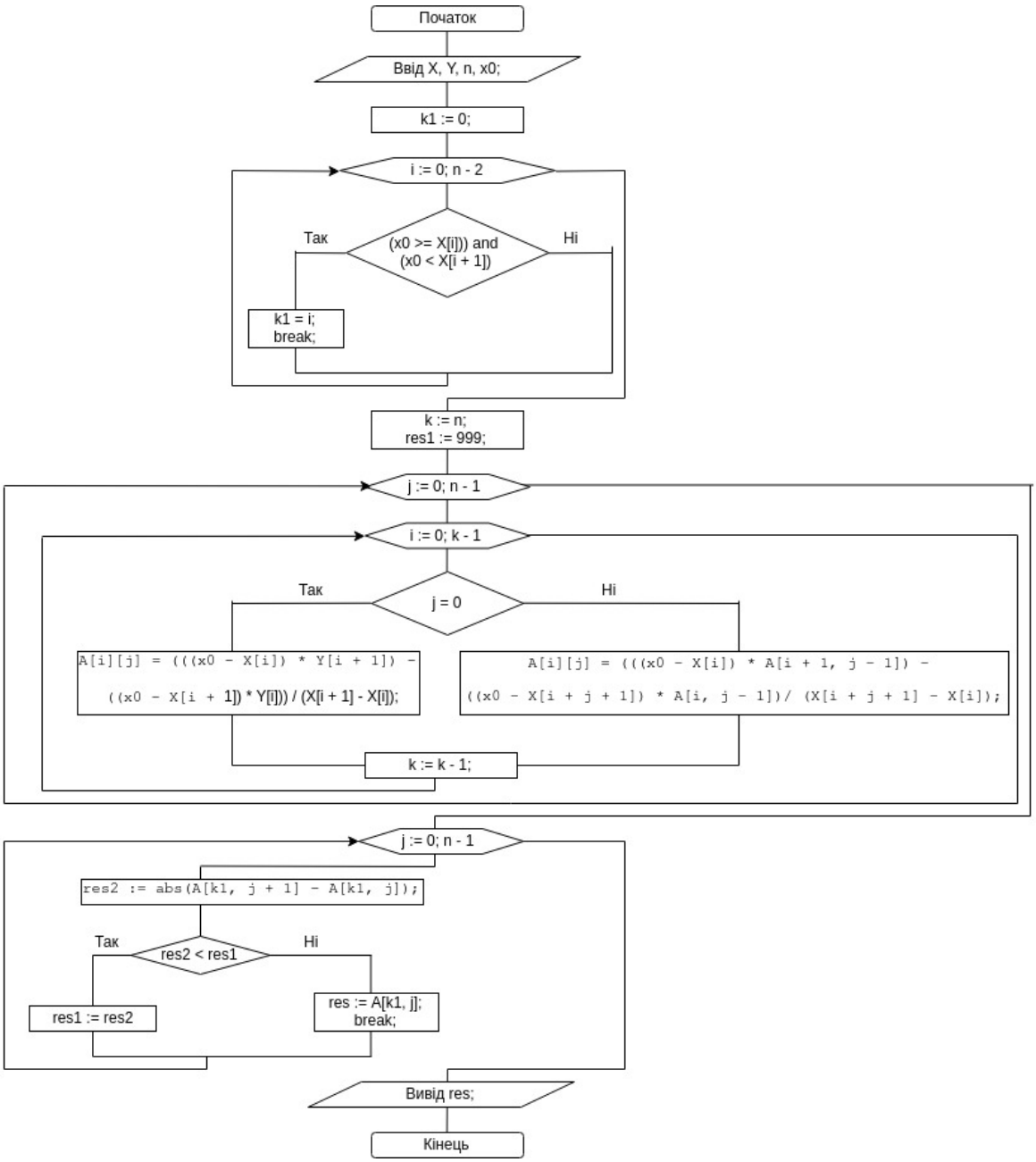


Рис.3.2. Блок-схема алгоритму інтерполяції табличної функції використовуючи схему Ейткена

3.2. Методи оцінки похибки інтерполяції . Оцінка похибки методу

Теоретична оцінка похибки інтерполяції.

Справедлива наступна оцінка похибки інтерполяції

$$f\left(x\right) = P_n\left(x\right) + \frac{\prod_{j=k}^{k+n}\left(x - x_j\right)}{\left(n+1\right)!} f^{\left(n+1\right)}\left(\xi\right) \tag{3.9}$$

де x_j – вузли сітки $\xi \in \left[x_k, x_{n+k}\right]$, x – значення аргументу, де оцінюється похибка інтерполяції. Для безпосереднього застосування цієї формули необхідно мати верхню оцінку модуля $\left(n+1\right)$ - ї похідної функції $f\left(x\right)$. Якщо йдеться про інтерполяцію відомої функції по її табличних значеннях, то така оцінка може бути отримана аналітично. Наприклад, похідна будь-якого порядку від функцій $\sin x$ і $\cos x$ по модулю не перевищує одиниці.

Необхідно відзначити, що значення $\omega_n\left(x\right)$ між вузлами x_j поблизу кінців інтервалу інтерполяції істотно більше (по модулю), ніж у середині. Крім того, при збільшенні n значення $\omega_n\left(x\right)$ швидко зростають. Звідси випливає, що підвищення степеня многочлена може призвести до збільшення похибки інтерполяції, якщо зі збільшенням порядку похідної досить швидко збільшується її величина.

Практична оцінка похибки інтерполяції за результатами чисельного експерименту. У разі, коли інтерпольована функція є результатом чисельного розв'язку деякої задачі, вся інформація про шукану функцію вичерпується її значеннями у вузлових точках. Задача інтерполяції при цьому є некоректною, оскільки може існувати скільки завгодно функцій, графіки яких проходять через дані точки (рис. 3.2а). Тобто, рішення задачі може бути отримано тільки з точністю до довільної адитивної складової, що має нульові значення у всіх заданих вузлових точках.

Однак, якщо сітка вибирається довільно, то існування функції, рівної нулю саме в вузлових точках цієї сітки, малоймовірно. Можна також вказати різні способи використання декількох сіток для підвищення надійності одержуваних результатів. У випадку, розглянутому на рис. 3.2б, функція має різкий сплеск на одному з часткових відрізків. При цьому інтерполяційна формула може просто «не помітити» цього сплеску, оскільки у вузлових точках його вплив може бути дуже малим. «Відчути» такий сплеск можна тільки при уточненні результату (наприклад, шляхом підвищення степеню інтерполяційного многочлена, згущенням сітки).

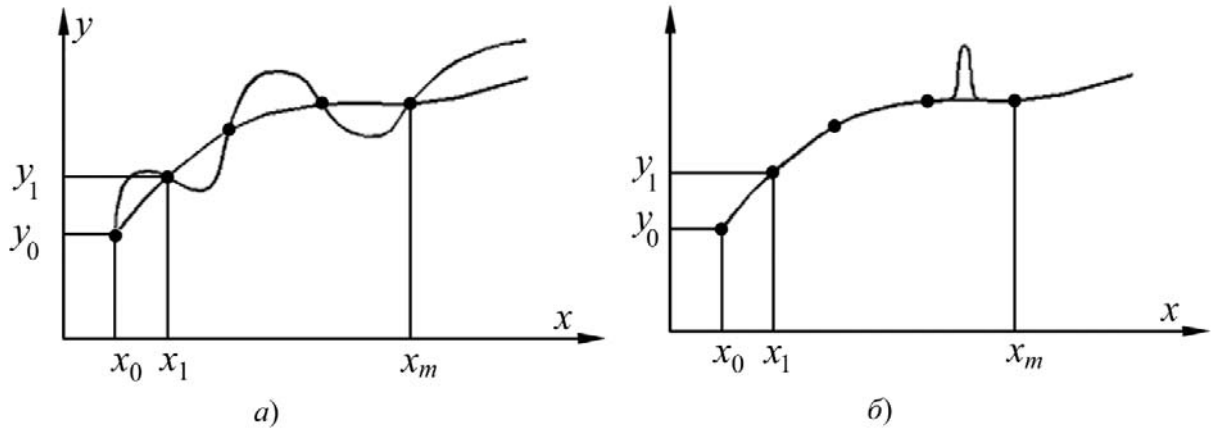


Рис. 3.2. Некоректність задачі інтерполяції

Таким чином, хоча повністю виключити можливість помилки (неправильної оцінки похибки результату), пов'язаної з некоректністю завдання, не можна, але є шляхи зменшення такої можливості.

Розглянемо спосіб оцінки похибки інтерполяції, що не потребує використання будь-якої іншої інформації, крім значень функції у вузлових точках. Для цього на підставі (3.9) представимо математичну модель похибки інтерполяції у наступному вигляді

$$P_n^1\left(x\right) - f\left(x\right) = c\prod_{j=k_1}^{k_1+n}\left(x - x_j^1\right) + \delta_1\left(x\right) \tag{3.10}$$

Тут x_j^1 – вузли деякої сітки; $j = 0,...,N_1$, c – величина, що незалежна від положення вузлів; k_1 – номер початкового вузла, використовуваного інтерполяційною формулою; $\delta_1\left(x\right)$ – додаткова частина похибки, яку вважають малою величиною в порівнянні з першим доданком.

Тепер змінимо сітку, використовуючи нові вузли x_j^2, $j = 0,...,N_2$. Тоді отримаємо друге рівняння для знаходження невідомих c і $f\left(x\right)$.

$$P_n^2\left(x\right) - f\left(x\right) = c\prod_{j=k_2}^{k_2+n}\left(x - x_j^2\right) + \delta_2\left(x\right) \tag{3.11}$$

Віднімаючи (1.10) з (1.11) і нехтуючи малими, знайдемо c

$$c = \frac{P_n^2\left(x\right) - P_n^1\left(x\right)}{\Pi_2 - \Pi_1},\ \Pi_i = \prod_{j=k_i}^{k_i+m}\left(x - x_j^i\right) \tag{3.12}$$

оцінку похибки інтерполяції

$$P_n^1\left(x\right)-f\left(x\right)=\frac{\left(P_n^2\left(x\right)-P_n^1\left(x\right)\right)\Pi_1}{\Pi_2-\Pi_1} \tag{3.13}$$

і більш точне значення функції

$$f\left(x\right)\approx\frac{P_n^1\left(x\right)\Pi_2-P_n^2\left(x\right)\Pi_1}{\Pi_2-\Pi_1}. \tag{3.14}$$

Формувати різні сітки можна різними способами (наприклад, зменшенням кроку в 2 рази, вибором закону розподілу вузлів). У тому числі для оцінки інтерполяції можна використовувати значення функції в інших вузлах тієї ж самої сітки. Останнє може виявитися більш зручним з практичної точки зору. Спосіб вибору вузлів також може бути різним.

Розглянемо випадок, коли другий набір x_j^2 складається з вузлів x_j^1 з номерами від $k+1$ до $n+k+1$ (тобто $k_1=k$, $k_2=k+1$). У цьому випадку згідно (3.13) похибка оцінюється за формулою

$$P_n^1\left(x\right)-f\left(x\right)\approx\frac{\left[P_n^2\left(x\right)-P_n^1\left(x\right)\right]\prod\limits_{j=k}^{k+n}\left(x-x_j\right)}{\prod\limits_{j=k+1}^{k+n+1}\left(x-x_j\right)-\prod\limits_{j=k}^{k+n}\left(x-x_j\right)}=$$

$$=-\left[P_n^2\left(x\right)-P_n^1\left(x\right)\right]\frac{x-x_k}{x_{k+n+1}-x_k}, \tag{3.15}$$

а (3.14) має вигляд

$$f\left(x\right)\approx\frac{x_{k+n+1}-x}{x_{k+n+1}-x_k}P_n^1\left(x\right)+\frac{x-x_k}{x_{k+n+1}-x_k}P_n^2\left(x\right)=P_{n+1}\left(x\right) \tag{3.16}$$

Функція (3.15) в дійсності представляє собою інтерполяційний многочлен степеню $n+1$, оскільки:

- $P_{n+1}\left(x\right)$ є алгебраїчним многочленом степеню $n+1$;

- у вузлах з номерами від $i=k+1$ до $i=k+n$ обидва многочлени $P_n^1\left(x_i\right)$ і, $P_n^2\left(x_i\right)$, а, отже, і $P_{n+1}\left(x_i\right)$, збігаються з $f\left(x_i\right)$;

$$P_{n+1}\left(x_k\right)=P_n^1\left(x_k\right)=f\left(x_k\right);$$

$$P_{n+1}\left(x_{n+k+1}\right)=P_n^2\left(x_{n+k+1}\right)=f\left(x_{n+k+1}\right)$$

Рекурентна формула (3.15) використовується при інтерполяції за схемою Ейткена.

Таким чином, даний спосіб оцінки похибки інтерполяції зводиться до побудови інтерполяційного многочлена $P_{n+1}\left(x\right)$ і порівнянні $P_n\left(x\right)$ значень з $P_{n+1}\left(x\right)$ як з більш точними.

Оцінка похибок початкових даних та округлення

Крім похибки інтерполяції необхідно враховувати похибку, яка обумовлена помилками самих використовуваних значень функції. Цю похибку, згідно (3.3), можна оцінити за формулою

$$\Delta_n\left(x\right) = \sum_{j=k}^{k+n} \sigma_j A_j, \quad A_j = \left| \prod_{\substack{i=k \\ i \neq j}}^{k+n} \frac{x - x_i}{x_j - x_i} \right| \tag{3.17}$$

або за рекурентним співвідношенням,

$$\Delta_{n+1}\left(x\right) = \left| \frac{x_{k+n+1} - x}{x_{k+n+1} - x_k} \Delta_n^1\left(x\right) \right| + \left| \frac{x - x_k}{x_{k+n+1} - x_k} \Delta_n^2\left(x\right) \right| \tag{3.18}$$

$$\Delta_0^1\left(x\right) = \sigma_k, \qquad \Delta_0^2\left(x\right) = \sigma_{k+1}$$

де σ_j – відомі оцінки похибки значень y_j. Якщо $y_j = f\left(x_j\right)$ – обчислені значення відомої функції, то

$$\sigma_j \leq \left| y_j \right| \cdot 10^{-M+1}, \tag{3.19}$$

(M – число десяткових розрядів мантиси машинного слова).

Помилку округлення при застосуванні інтерполяційної формули Лагранжа можна оцінити таким способом. Якщо при програмній реалізації цього способу інтерполяції проводиться додавання методом накопичення, то величина похибки округлення приблизно в n разів більше, ніж та, яка випливає з (3.17) з урахуванням (3.19).

При застосуванні рекурентної формули (3.16) відбувається попарне додавання та накопичення часткових сум за схемою бінарного дерева. Якщо при кожному додаванні доданки приблизно дорівнюють один одному, то накопичення похибки округлення, пов'язаної з вирівнюванням порядків доданків, що суттєво відрізняються між собою, не відбувається. Загальну похибку, пов'язану з машинним представленням чисел, можна тоді оцінити за формулою (3.18).

Відзначимо, що в практичних розрахунках степінь інтерполяційного многочлена, як правило, не перевищує 10, тому похибка округлення не перевищує набагато похибку вхідних даних. Однак існує можливість того, що доданки суми мають великі за модулем величини і різні знаки, так що сума має істотно менше значення. Тоді відносна похибка округлення може виявитися дуже великою.

Критерій якості оцінки похибки

Оскільки оцінки (3.13) – (3.16) виведені з припущенням, що величини $\delta_i(x)$ малі, то необхідна перевірка справедливості цього припущення. Це можна зробити наступним чином. Оцінка похибки за формулою (3.16) зводиться до порівняння значення $P_n(x)$ із значенням, отриманим при інтерполяції многочленом $(n+1)$-го степеню $P_{n+1}(x)$. Тому процес збільшення степеню можна продовжити і отримати значення $P_{n+2}(x)$. $\Delta_n = P_n(x) - P_{n+1}(x)$ є оцінкою похибки інтерполяції значення $P_n(x)$. Різниця $\Delta_{\Delta n} = P_{n+1}(x) - P_{n+2}(x)$ є оцінкою похибки оцінки похибки (рис. 3.3). Відношення $\delta_n = \left|\Delta_{\Delta n}/\Delta_n\right|$ змістовно визначає відносну розмитість оцінки похибки. Якщо $\delta_n \ll 1$, то це означає, що відносна розмитість оцінки мала, і такій оцінці можна довіряти. Якщо ж $\delta_n > 0.3 - 0.4$, то ширина області розмитості порівняна з Δ_n і таку оцінку слід відкинути.

3.3. Чисельний експеримент

Застосуємо цей спосіб оцінки до конкретної задачі інтерполяції. Нехай

$$f(x) = \sin x, \;\; x_j = \frac{j}{m}\frac{\pi}{2}, \quad y_j = f(x_j), \quad j = 0,\ldots,m.$$

Результати інтерполяції і оцінки похибки зручно представляти на графіку у вигляді залежності $-\lg\left|P_n - P_{n+1}\right|$ (десяткового логарифма правої частини (3.14)) від $\overline{x} = (x - x_j)/(x_{j+1} - x_j)$. На рис. 3.3 різні криві відповідають різним n (при $j = 2$).

Відзначимо, що збільшення ординати кривої на одиницю при збільшенні степеню інтерполяційного многочлена означає зменшення похибки в 10 разів. Зближення кривих означає те, що за рахунок подальшого збільшення степеню точність підвищити не вдається.

Попарне зближення кривих пояснюється тим, що функція $\sin x$ – непарна, і в її розкладанні за степенями x присутні тільки непарні члени.

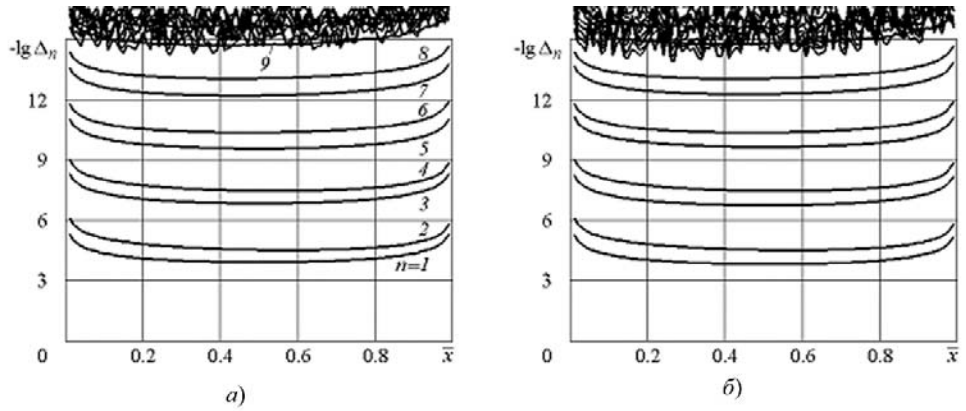


Рис. 3.3. Результати інтерполяції

З рис. 3.3а видно, що в результаті інтерполяції даних цього прикладу при m = 20 можуть бути отримані значення з похибкою порядку 10-13 з відносною розмитістю близько 0.01. На рис. 3.3б зображені криві, аналогічні наведеним на рис. 3.3а, тільки для оцінки похибки використані точні значення функції $\sin x$. Видно, що відмінність графіків на обох малюнках незначна, що говорить про високу точність оцінки похибки за цим методом.

На рис. 3.4а наведені оцінки похибки (3.16), яка викликана помилками використовуваних значень функції $\sigma_j = \sin x_j \cdot 10^{-15}$ (тут використана подвійна точність). Ця похибка, як неважко помітити, істотно перевищує значення σ_j. При інтерполяції на відрізках, близьких до середини таблиці, ця похибка значно менша (рис. 3.4б).

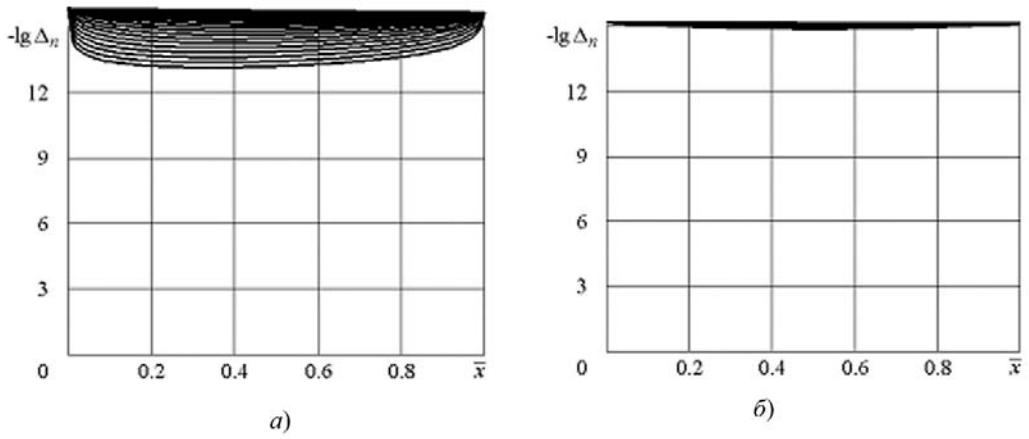


Рис. 3.4. Вплив похибки початкових даних при інтерполяції

У табл. 3.1 подано результати розрахунків для цього ж прикладу для точки, розташованої посередині між вузлами. Величина $\Delta_n = P_n\left(x\right) - P_{n+1}\left(x\right)$ оцінена за формулою (3.15) похибка інтерполяції; Δ_n^{exact} – різниця між інтерпольованим і точним значеннями; $k_\Delta = 1 - \Delta_n^{exact} \big/ \Delta_n$ – є коефіцієнтом уточнення інтерпольованого значення і він же рівний частковій оцінці похибки оцінки похибки (3.15), тобто фактичної розмитості оцінки (3.15).

Таблиця 3.4

Похибки

n	$_n$	Δ_n^{exact}	k
1	-1.2 x10^{-4}	-1.5x10^{-4}	0.25
2	-3.0x10^{-5}	-3.0x10^{-5}	0.01
3	-1.4x10^{-7}	-1.7x10^{-7}	0.25
4	-3.4x10^{-8}	-3.4x10^{-8}	0.01
5	-2.7x10^{-10}	-2.2x10^{-10}	-0.16
6	-4.3x10^{-11}	4.4x10-^{11}	0.01
7	6.1x10^{-13}	5.2x10^{-13}	-0.15

n	$_n$	Δ_n^{exact}	k
8	-9.0x10^{-14}	-9.1x10^{-14}	0.02
9	-1.8x10^{-15}	-1.6x10^{-15}	-0.13
10	1.9x10^{-16}	2.4x10^{-16}	0.22
11	5.6x10^{-17}	4.3x10^{-17}	-0.22
12	2.8x10^{-17}	-1.2x10^{-17}	-1.44
13	8.3x10^{-17}	-4.0x10^{-17}	-1.48

З таблиці видно, що при $k_\Delta = 0.01$ значення Δ_n і Δ_n^{exact} практично збігаються. При $0.2 < k_\Delta < 0.3$ значення Δ_n і Δ_n^{exact} помітно різняться, але при оцінці похибки такі відмінності можуть вважатися допустимими. При $k_\Delta > 0.3$ значення Δ_n і Δ_n^{exact} розрізняються суттєво, і оцінку похибки за таких умов не можна вважати задовільною.

Таким чином, застосування розглянутого способу оцінки похибки інтерполяції дозволяє не лише з високою точністю оцінити цю похибку (користуючись тільки інформацією, закладеною в табличних даних), але і наближено визначити частку похибки, що міститься в цій оцінці. Це дозволяє судити про якість оцінки, і в разі незадовільного результату відкинути таку оцінку.

Порядок розв'язування задачі на комп'ютері

1) Застосувати метод інтерполяції (многочлени Лагранжа (3.3), Ньютона (3.6) або рекурентне співвідношення Ейткена (3.8)) відповідно до варіанту .

2) Скласти програму, що обчислює значення заданої функції $y_i = f\left(x_i\right)$ у вузлах інтерполяції $x_i = a + hi$, де $h = \dfrac{\left(b-a\right)}{10}$, $i = 0,1,...,10$, на відрізку $\left[a,b\right]$.

3) Передбачити в програмі оцінку похибки на основі порівняння значень, отриманих за допомогою інтерполяційних многочленів різного степеню.

4) Оцінити розмитість оцінки похибки.

5) Налагодити програму шляхом інтерполяції функції $\sin x$ (див. «Чисельний експеримент»).

6) Застосувати програму для інтерполяції функції, з таблиці 2 за номером у списку.

7) Результат оцінки похибки представити у вигляді графіка (рис. 3.3, 3.4) і для одного з значень x у вигляді таблиці 3.2.

Таблиця 3.2

Варіанти завдань до лабораторної роботи №3

№ вар.	*f*(*x*)	[*a, b*]	формула	№ вар.	*f*(*x*)	[*a, b*]	формула
1	$\sin x^2$	[0, 2]	1.3	2	$x \cdot \cos\left(x + \ln\left(1 + x\right)\right)$	[1, 5]	1.3
3	$\cos x^2$	[0, 2]	1.6	4	$10 \cdot \ln 2x \big/ \left(1 + x\right)$	[1, 5]	1.6
5	$e^{\sin x}$	[0, 5]	1.8	6	$\sin x^2 \cdot e^{-\left(x/2\right)^2}$	[0, 3]	1.8
7	$1 \big/ \left(0.5 + x^2\right)$	[0, 2]	1.3	8	$\cos\left(x + \cos^3 x\right)$	[0, 2]	1.3

9	$e^{-(x+\sin x)}$	[2, 5]	1.6	10	$\cos(x + e^{\cos x})$	[3, 6]	1.6
11	$1/(1+e^{-x})$	[0, 4]	1.8	12	$\cos(2x + x^2)$	[0, 1]	1.8
13	$\sin(x + e^{\sin x})$	[0, 3]	1.3	14	$e^{\cos x}\cos x^2$	[0, 2]	1.3
15	$e^{-(x+1/x)}$	[1, 3]	1.6	16	$\sin^2 x$	[0,1]	1.6
17	$\cos^2 x$	[0,1]	1.8	18	$e^{\cos x}$	[0,5]	1.8
19	$x^2 + 2e^x$	[0,1]	1.3	20	$e^x \sin x$	[0,1	1.3
21	$\sin x - 2\cos x$	[0,4]	1.6	22	$3\cos^2(x) - \sqrt{x}$	[0,3]	1.6
23	$\sin^3 x + 3\cos^2(x)$	[0,4]	1.8	24	$e^x - 2\cdot\sin x$	[0,2]	1.8
25	$x^2 \cdot \cos x$	[0,3]	1.3	26	$2^x - \ln x$	[1,3]	1.3
27	$2\sin x - 3\cos x$	[0,4]	1.6	28	$x^3 - 3^x$	[0,3]	1.6
29	$4^x - 8x$	[0,2]	1.8	30	$3^x + e^{(x-1)}$	[0,1]	1.8

3.4. Вимоги до звіту з лабораторної роботи

Звіт по лабораторній роботі повинен містити:

1) файл вхідного тексту програми;

2) файли результатів для тестового прикладу і для інтерполяції заданої функції;

3) опис алгоритму розрахунку (в текстовій формі та у вигляді блок-схеми) в електронному та роздрукованому вигляді;

4) роздруківку файлів з коментарями;

5) загальні висновки за результатами роботи, що включають результати тестування, отримані оцінки похибки результатів і обґрунтування цих оцінок.

3.5. Контрольні запитання

1) Переваги та недоліки різних методів інтерполяції.
2) Оцінка ефективності різних способів оцінки похибки інтерполяції з точки зору їх надійності та практичної застосовності.

3) Вплив похибки початкових даних та округлення на результат інтерполяції.

4) Способи зменшення похибок при інтерполяції.

5) Способи підвищення надійності оцінки похибки інтерполяції.

3.6. Інформаційні ресурси

1. Інтерполяція і наближення функцій: http://amc.ptngu.com/rozdil6.html
2. Інтерполяційні формули Лагранжа для нерівновіддалених вузлів інтерполяції: https://www.mathros.net.ua/interpoljacijni-formuly-lagranzha-dlja-nerivnoviddalenyh-vuzliv-interpoljacii.html
3. Інтерполяційний поліном в формі Лагранжа: https://stud.com.ua/140762/prirodoznavstvo/interpolyatsiyniy_polinom_formi_lagranzha
4. Перша інтерполяційна формула Ньютона для рівновіддалених вузлів інтерполяції: https://www.mathros.net.ua/persha-interpoljacijna-formula-njutona-dlja-rivnoviddalenyh-vuzliv-interpoljacii.html
5. Інтерполяційний многочлен у формі Ньютона: https://stud.com.ua/180927/prirodoznavstvo/interpolyatsiyniy_mnogochlen_formi_nyutona
6. Інтерполяційна схема Ейткена: https://www.mathros.net.ua/interpoljacijna-shema-ejtkena.html
7. Оцінка похибки інтерполяції: https://stud.com.ua/140763/prirodoznavstvo/otsinka_pohibki_interpolyatsiyi

Розділ 4. ЛАБОРАТОРНА РОБОТА № 4

Тема: «Розв'язання нелінійних рівнянь на комп'ютері»

Мета: Метою даного заняття є ознайомлення з методиками та вивчення різних алгоритмів розв'язання нелінійних рівнянь на комп'ютері.

Завдання: Закріплення знань студентів при вирішенні практичних завдань з розв'язування нелінійних рівнянь. Оволодіння методами і практичними навичками розв'язування нелінійних рівнянь на комп'ютері. Набуття умінь і навичок при програмуванні та налагодженні програм для розв'язування нелінійних рівнянь на комп'ютері.

4.1. Теоретичні відомості до лабораторної роботи №4

4.1.1. Загальні поняття та визначення

При вирішенні практичних інженерних задач часто доводиться зустрічатися з розв'язанням рівнянь виду

$$\phi(x) = g(x), \tag{4.1}$$

$$\text{або } f(x) = 0 \tag{4.2}$$

де $\phi(x)$, $g(x)$ та $f(x) = 0$ – нелінійні функції, визначені на деякій числовій множині X, яка називається *областю допустимих значень рівняння.*

Рівняння виду (4.1) або (4.2) називаються *нелінійними рівняннями*. Всі нелінійні рівняння можна поділити на алгебраїчні та трансцендентні (рис. 4.1)

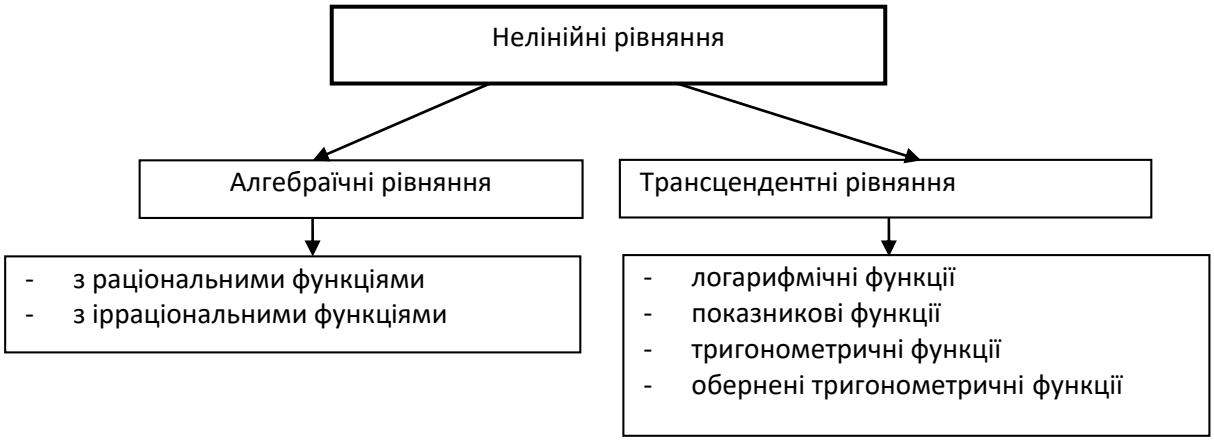


Рис. 4.1. Класифікація нелінійних рівнянь

Функція називається алгебраїчною, якщо для отримання значення функції на заданій множині X потрібно здійснити арифметичні операції та піднесення до степеню з раціональним або ірраціональним показником. Рівняння, які містять алгебраїчні функції, називають нелінійними алгебраїчними рівняннями.

До трансцендентних функцій відносять всі неалгебраїчні функції: показникові, логарифмічні, тригонометричні, обернені тригонометричні та інші. Нелінійні рівняння, які містять трансцендентні функції, називають ***нелінійними трансцендентними рівняннями.***

Розв'язком нелінійного рівняння на ПК називають вектор

$$\overline{X} = \left\{ x_1, x_2, ..., x_n \right\},$$

координати якого при підстановці в початкове рівняння перетворюють його в тотожність.

В нелінійному рівнянні виду

$$a_0 x^n + a_1 x^{n-1} + a_2 x^{n-2} + ... + a_{n-1} x + a_n = 0 \qquad (4.3)$$

i-ту координату вектора $\overline{X =} \left\{ x_1, x_2, ..., x_n \right\}$ називають i-тим коренем рівняння, а $a_1, a_2, ..., a_m$ – коефіцієнтами рівняння (4.3).

4.1.2. Метод половинного ділення

Нехай потрібно уточнити єдиний корінь рівняння $f\left(x\right) = 0$, що належить відрізку $\left[a;b\right]$ (відрізок невизначеності).

Точка $c = \dfrac{a+b}{2}$ – середина відрізка $\left[a;b\right]$. Якщо $f(c) = 0$, то корінь знайдений. В іншому випадку для подальшого розгляду залишаємо ту з половин $\left[a;c\right]$ або $\left[c;b\right]$, на кінцях якої знаки функції $f\left(x\right)$ різні. При цьому виходить послідовність вкладених відрізків, що містять шуканий корінь. На кожному кроці довжина відрізка невизначеності зменшується вдвічі. Метод сходиться завжди. Умовою закінчення пошуку кореня може бути, наприклад,

$$\left|f\left(x\right)\right| < E \text{ або } \frac{\left|b-a\right|}{2^n} < E,$$

де E – точність, $\left[a;b\right]$ – початковий відрізок, n – число ітерацій.

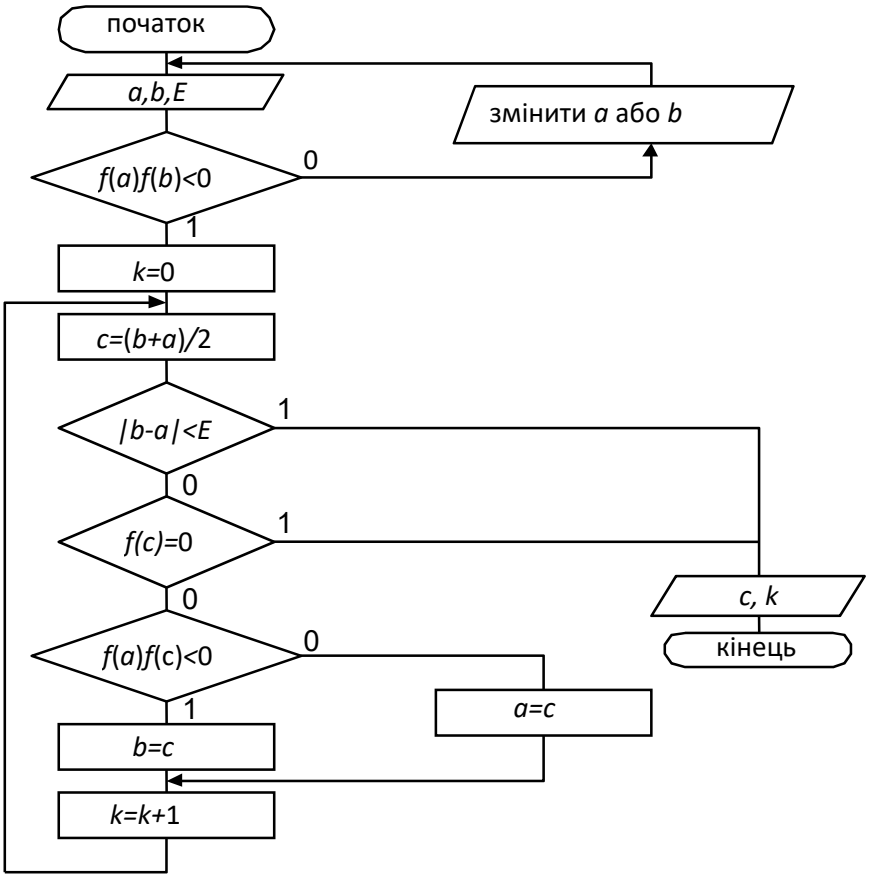


Рис. 4.2. Схема алгоритму розв'язання нелінійного рівняння методом половинного ділення

4.1.3. Метод хорд

Метод хорд є одним з найбільш поширених методів розв’язання алгебраїчних і трансцендентних рівнянь. В літературі він також зустрічається під назвою "метод лінійного інтерполювання" і "метод пропорційних частин".

Постановка задачі

Розглянемо рівняння $f(x)=0$, де $f(x)$ неперервна нелінійна функція, яка на відрізку $\left[a,b\right]$ монотонна, диференційована і має єдиний корінь ξ (тобто

$f(a) \cdot f(b) < 0$). Потрібно знайти наближене значення кореня ξ з заданою похибкою ε .

Суть метода хорд полягає в тому, що на достатньо малому відрізку $[a,b]$ дуга функції $f(x)$ замінюється хордою ab , яка її стягує. За наближене значення кореня приймається точка x_1 перетину хорди з віссю Ox (рис.4.3а).

Рівняння хорди, яка проходить через точки, має вигляд

$$\frac{y - f(x)}{f(b) - f(a)} = \frac{x - a}{b - a} \tag{4.4}$$

Знайдемо значення x_1, для якого $y = 0$, тобто для нерухомого кінця:

$$x_1 = a - \frac{f(a) \cdot (b - a)}{f(b) - f(a)} \tag{4.5}$$

Ця формула називається *формулою методу хорд*. Тепер корінь ξ знаходиться всередині відрізка $[x_1, b]$. Значення кореня x_1 можна уточнити за допомогою метода хорд на відрізку $[x_1, b]$, тоді нове наближене значення кореня x_2 знаходиться за формулою

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

Аналогічно для всякого $(i+1)$-го наближення до точного значення кореня ξ даного рівняння використовується формула:

$$x_{i+1} = x_i - \frac{f(x_i) \cdot (b - x_i)}{f(b) - f(x_i)} \tag{4.6}$$

Процес стягування хордою продовжується багаторазово доти, поки не одержано наближений корінь із заданим ступенем точності

$$\left|x_{i+1} - x_i\right| < \varepsilon \tag{4.7}$$

де x_{i+1}, x_i – наближені значення коренів рівняння $f(x) = 0$, відповідно на $(i+1)$ і i -му ітераційному кроці; ε – задана точність обчислень. Слід відмітити, що розглянутий випадок (рис. 4.3а) перетину функцією $f(x)$ відрізка $\left[a, b\right]$ не є єдиним. Існує ще три варіанти перетину функції, кожний з яких відрізняється напрямком побудови хорд і, відповідно, рухомими кінцями відрізка. Наприклад, на рис. 4.3а,б рухомий кінець відрізка *a,* а на рис. 4.3в,г рухомий кінець – b і, відповідно, формула (4.5) для нього має вигляд:

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

Для автоматизації цього алгоритму необхідно розробити правило для автоматичного вибору рухомого кінця хорди і, відповідно, формули для обчислення наближеного значення кореня. Існує два правила визначення рухомого кінця хорди.

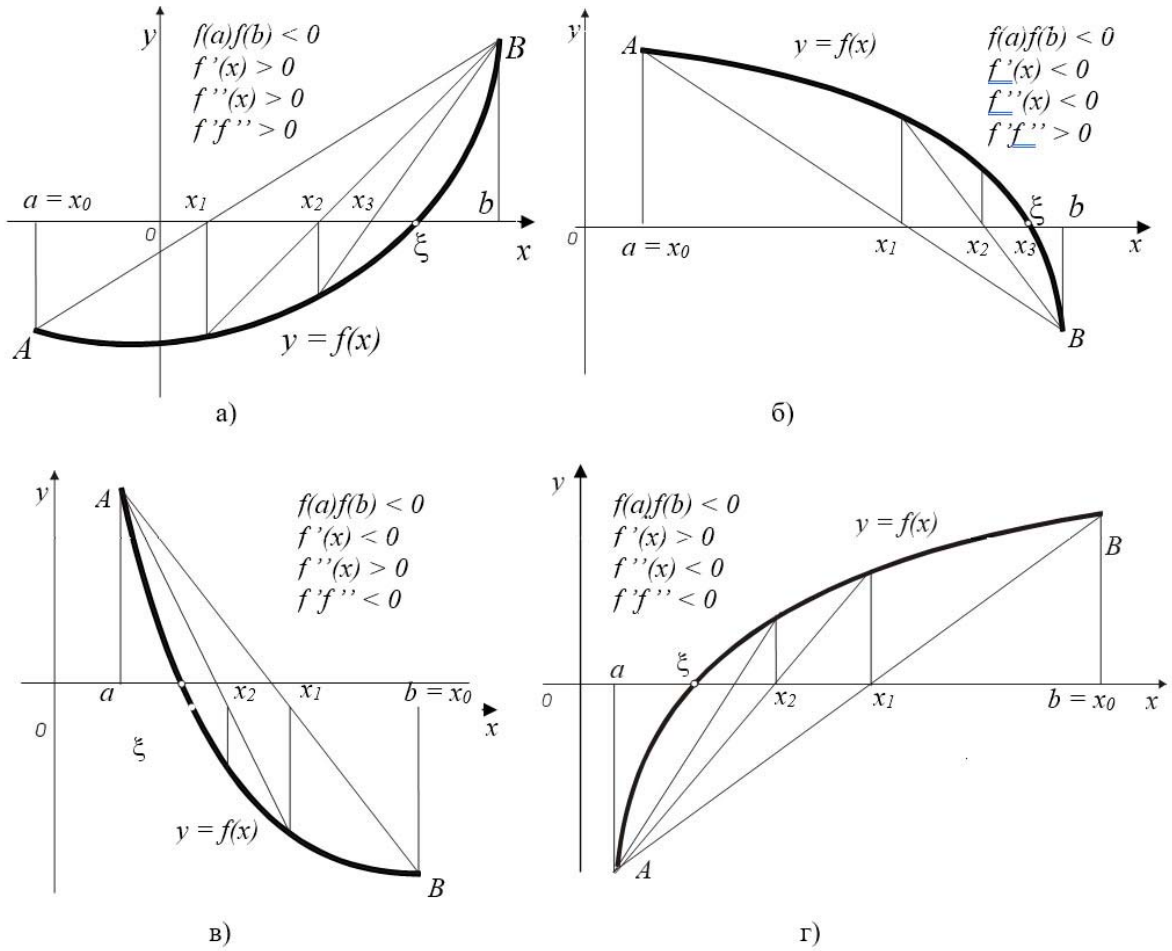


Рис. 4.3. Графічна інтерпретація методу хорд і процедури визначення рухомого кінця хорди

***Правило* 1.** Нерухомим кінцем відрізка є той, для якого знак функції співпадає із знаком другої похідної. Якщо $f(b) \cdot f''(x) > 0$, то нерухомим є кінець b, а всі наближення до кореня ξ лежать зі сторони кінця a. Якщо $f(a) \cdot f''(x) > 0$, то нерухомим є кінець a, а всі наближення до кореня ξ лежать зі сторони кінця b (рис. 4.3а,б,в,г).

Правило 2. Якщо добуток першої на другу похідну функції $f(x)$ більший за нуль: $f'f'' > 0$, то рухомий кінець a; якщо добуток першої на другу похідну менший за нуль: $f'f'' < 0$, то рухомий кінець b.

Схема алгоритму розв'язання нелінійного рівняння методом хорд представлена на рис. 4.4.

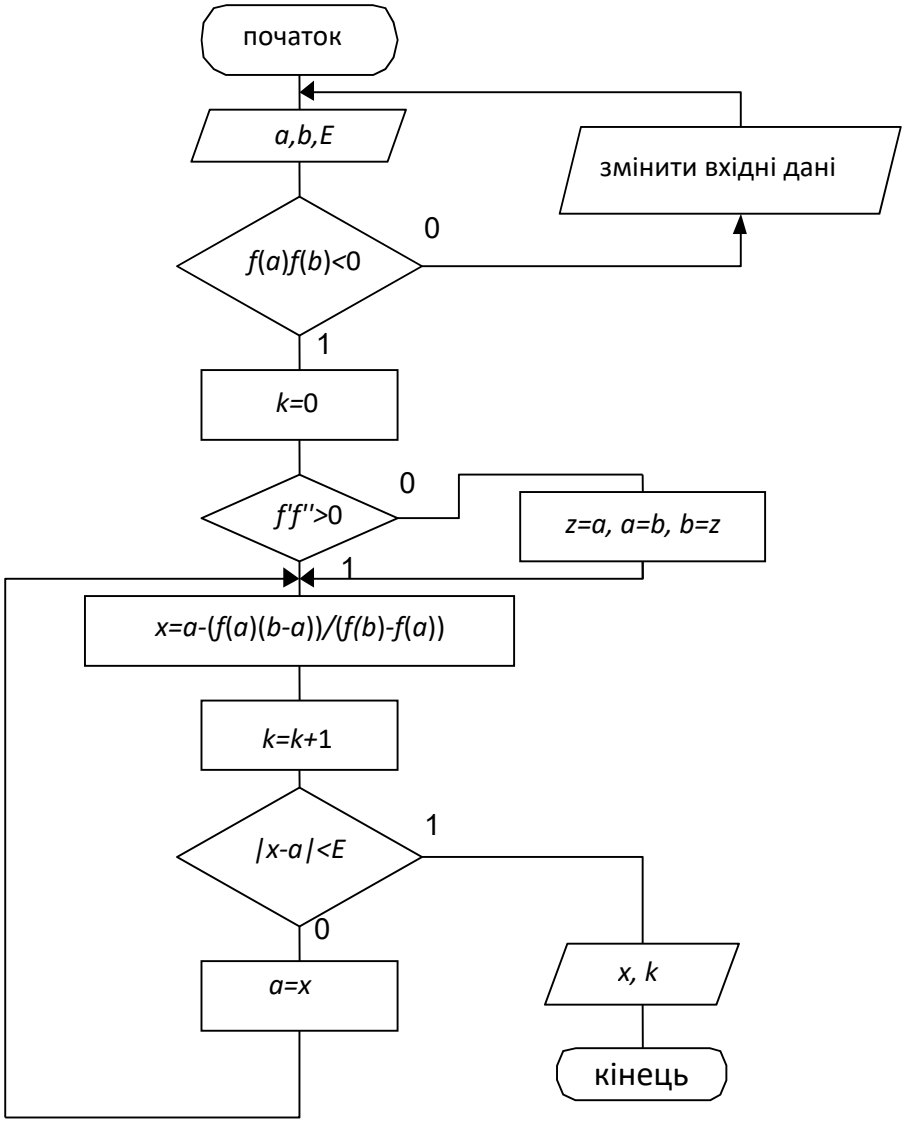


Рис. 4.4. Схема алгоритму розв'язання нелінійного рівняння методом хорд

Особливості розробки функцій, які реалізують алгоритм методу

1. Метод половинного ділення та метод хорд розробляються як незалежні підпрограми-функції з вхідними параметрами: *a, b,* та вихідними: *x, k*, де *x* – наближене значення кореня, *k*– кількість ітерацій.
2. В цих підпрограмах-функціях необхідно передбачити перевірку вхідних даних, наприклад, чи дійсно відрізок вибраний так, що функція на його кінцях має різні знаки.
3. Обчислення перших та других похідних здійснюється за допомогою спеціальних функцій, в яких заданий математичний вигляд похідної.
4. Перед викликом підпрограми-функції, яка реалізує метод, необхідно аналітично визначити кількість коренів. Аналітично чи програмно відокремити корені і в

циклі по кількості коренів викликати функцію, яка реалізує метод уточнення коренів так, щоб на екран були виведені всі відрізки, корені на кожному з цих відрізків та кількість ітерацій, за яку був отриманий кожен корінь.

4.1.4. Метод Ньютона (метод дотичних)

Метод послідовних наближень, розроблений Ньютоном, дуже широко використовується при побудові ітераційних алгоритмів. Його популярність обумовлена тим, що на відміну від двох попередніх методів замість інтерполяції по двох значеннях функції в методі Ньютона здійснюється екстраполяція за допомогою дотичної до кривої в одній точці.

Постановка задачі

Нехай корінь рівняння $f(x) = 0$ відокремлений на відрізку $[a,b]$, на якому нелінійна функція $f(x)$ монотонна і має різні знаки на кінцях відрізка, причому похідні $f'(x)$ та $f''(x)$ неперервні та зберігають постійні знаки на всьому відрізку $[a,b]$. Потрібно знайти наближене значення кореня ξ з заданою похибкою ε.

Геометричний зміст метода Ньютона полягає в тому, що дуга кривої $y = f(x)$ на відрізку $[a,b]$ замінюється дотичною до цієї кривої, а наближене значення кореня визначається як точка перетину дотичної з віссю Ox, проведеної з одного з кінців досліджуваного відрізка. Рівняння дотичної має вигляд:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}.$$

Перший випадок. Нехай $f(a) < 0,\ f(b) > 0,\ f'(x) > 0,\ f''(x) > 0$ (рис. 5а) або $f(a) > 0, f(b) < 0,\ f'(x) < 0,\ f''(x) < 0$ (рис. 5б). Проведемо дотичну до кривої $y = f(x)$ в точці $B_0(b; f(b))$ і знайдемо абсцису точки перетину дотичної з віссю Ox. Відомо, що рівняння дотичної в точці $B_0(b; f(b))$ має вид: $y - f(b) = f'(b) \cdot (x - b)$.

Припускаючи $y = 0, x = x_1$, отримаємо

$$x_1 = b - \frac{f(b)}{f'(b)} \tag{4.8}$$

Тепер корінь рівняння знаходиться на відрізку $\left[a, x_1\right]$. Застосовуючи знову метод Ньютона, проведемо дотичну до кривої в точці $B_1\left(x_1; f\left(x_1\right)\right)$ і отримаємо

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)},$$

і так далі (рис. 4.5).

Даний процес ітераційний, тому формула для будь-якого *n-го* кроку ітерації має вигляд:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{4.9}$$

В результаті отримана послідовність наближених значень $x_1, x_2, ..., x_n, ...$, кожний наступний член якої ближчий до кореня , ніж попередній. Однак всі x_n залишаються більше істинного кореня , тобто x_n – наближене значення кореня з надлишком. Процес визначення кореня продовжується багаторазово доти, поки не одержано наближений корінь із заданим ступенем точності

$$\left|x_{i+1} - x_i\right| < \varepsilon,$$

де x_{i+1}, x_i – наближені значення коренів рівняння $f(x) = 0$ відповідно на $(i+1)$ і i-му ітераційному кроці; ε – задана похибка обчислень.

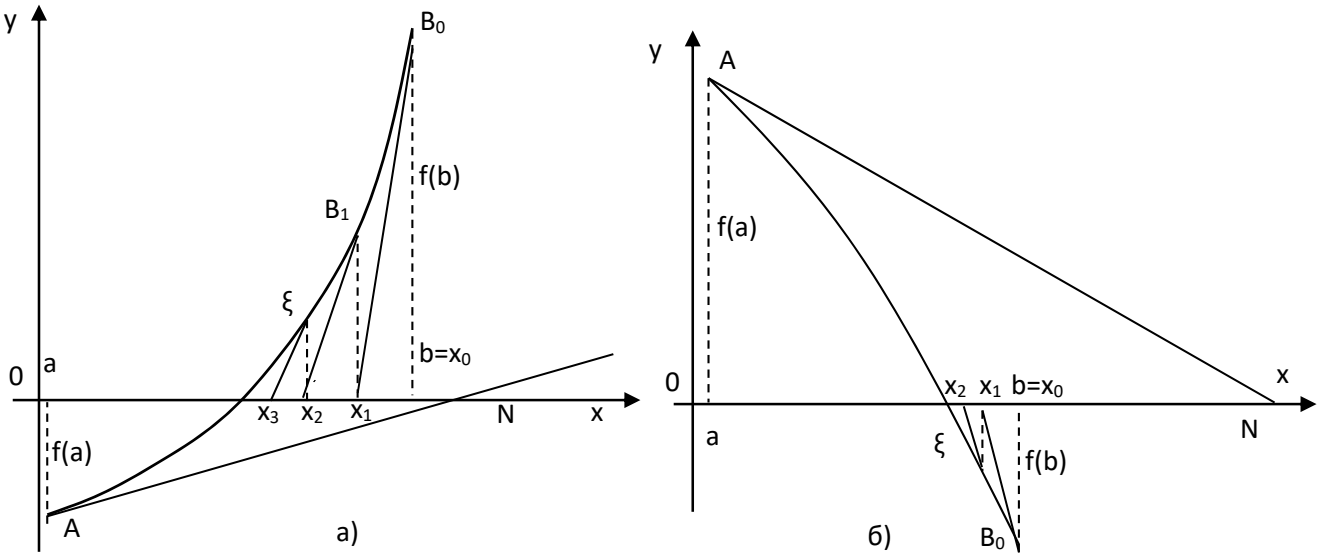


Рис. 4.5. Геометричний зміст метода Ньютона для випадків, коли

а) функція, яка досліджується, увігнута $\left(f'\left(x\right)>0, f''\left(x\right)>0\right)$

б) функція, яка досліджується, опукла $\left(f'\left(x\right)<0, f''\left(x\right)<0\right)$

Другий випадок. Нехай $f\left(a\right)<0,\ f\left(b\right)>0,\ f'\left(x\right)>0,\ f''\left(x\right)<0$ (рис. 6а) або $f\left(a\right)>0$, $f\left(b\right)<0$, $f'\left(x\right)<0$, $f''\left(x\right)>0$ (рис. 4.6б). Якщо провести дотичну до кривої $y=f(x)$ в точці *B,* то вона перетне вісь абсцис в точці, яка не належить відрізку $\left[a,b\right]$. Тому проведемо дотичну в точці $A_0\left(a;f\left(a\right)\right)$ і запишемо її рівняння для даного випадку: $y-f\left(a\right)=f'\left(a\right)\left(x-a\right)$.

Припускаючи, що $y=0, x=x_1$, отримаємо

$$x_1 = a - \frac{f(a)}{f'(a)} \tag{4.10}$$

Корінь знаходиться тепер на відрізку $\left[x_1;b\right]$. Застосовуючи знову метод Ньютона, проведемо дотичну до кривої в точці $A_1\left(x_1;f\left(x_1\right)\right)$ і отримаємо

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}, \text{ і загалом } x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{4.11}$$

В результаті отримаємо послідовність наближених значень $x_1, x_2, ..., x_n, ...$ кожний наступний член якої ближчій до істинного кореня , ніж попередній, тобто, x_n – наближене значення кореня з недостачею.

Порівнюючи формули (4.10), (4.11) з раніше виведеними, а також враховуючи випадки, які показано на рис. 4.6а,б помічаємо, що вони відрізняються одна від одної тільки вибором початкового наближення: в першому випадку за x_0 приймався кінець b відрізка, в другому – кінець a .

При виборі початкового наближення кореня необхідно використовувати наступне правило:

за початкову точку слід вибирати той кінець відрізка $[a;b]$*, в якому знак функції співпадає зі знаком другої похідної.*

В першому випадку $f(b) \cdot f''(x) > 0$ і початкова точка $b = x_0$, в другому $f(a) \cdot f''(x) > 0$ і в якості початкового наближення беремо $a = x_0$.

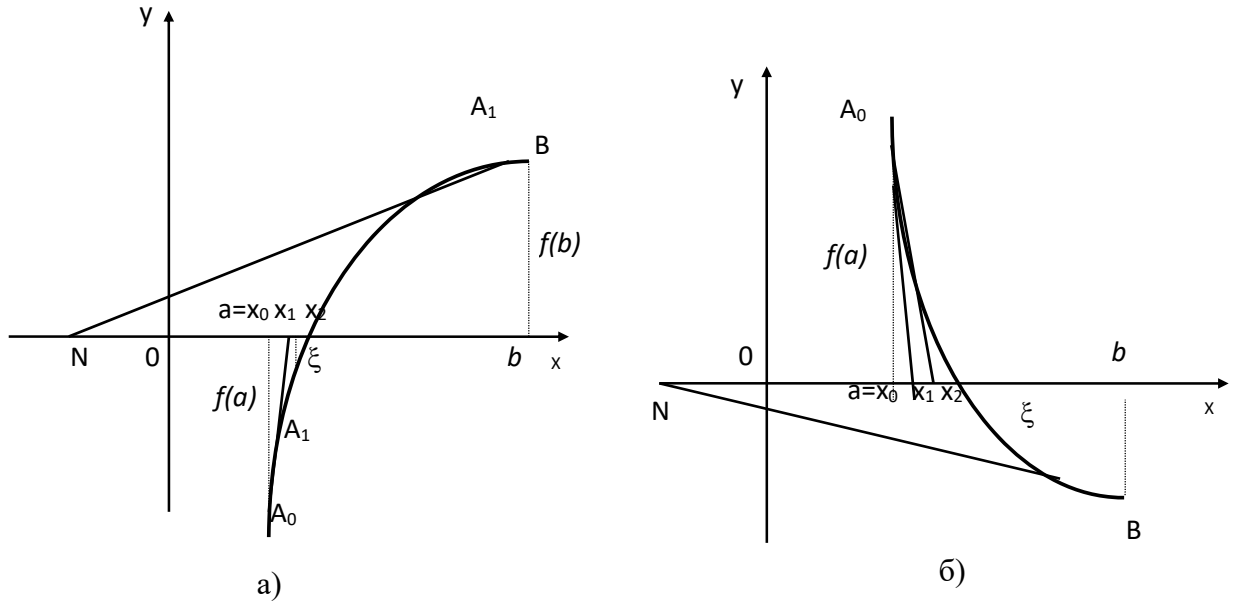


Рис. 4.6. Геометричний зміст методу Ньютона для випадків, коли

а) функція, яка досліджується, опукла $\left(f'(x) > 0, f''(x) < 0\right)$,

б) функція, яка досліджується, увігнута $\left(f'(x) < 0, f''(x) > 0\right)$.

Для оцінки похибки можна користуватися загальною формулою

$$\left|\xi - x_n\right| \leq \frac{\left|f(x_n)\right|}{m}, \tag{12}$$

де $m = \min\limits_{[a,\ b]}\left| f'(x) \right|$ (цю формулу можна використовувати і в методі хорд).

В тому випадку, коли відрізок $\left[a,b\right]$ настільки малий, що на ньому виконується умова $M_2 < 2m_1$, де $M_2 = \min\limits_{[a,\ b]}\left| f''(x) \right|$, а $m_1 = \min\limits_{[a,\ b]}\left| f'(x) \right|$, точність наближення на n -му кроці інтерполяційного процесу оцінюється наступним чином: якщо

$$\left|x_n - x_{n-1}\right| < \varepsilon \quad , \qquad \left|\xi - x_n\right| < \varepsilon^{\ 2}.$$

Якщо похідна $f'\left(x\right)$ мало змінюється на відрізку $\left[a,b\right]$, то для спрощення обчислень можна користуватися формулою

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_0)}, \tag{4.13}$$

тобто значення похідної в початковій точці достатньо обчислити тільки один раз.

Процес побудови дотичної продовжується багаторазово доти, поки $\left|x_{i+} - x_i\right| < \varepsilon$, де ε – задана точність обчислень; x_{i+1}, x_i – наближені значення кореня рівняння $f(x) = 0$, відповідно на $(i+1)$ та i - тому ітераційному кроці.

Схема алгоритму розв'язання нелінійного рівняння методом дотичних представлена на рис. 4.7.

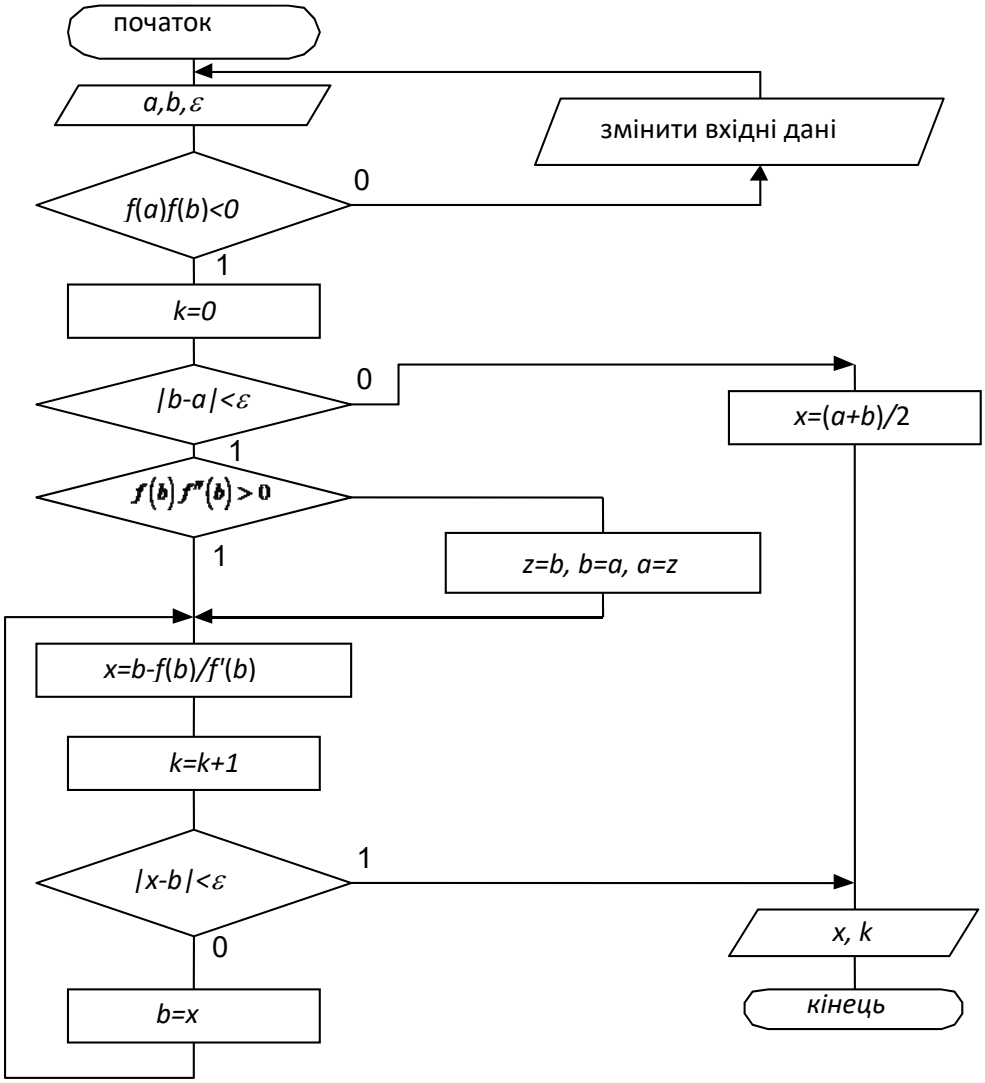


Рис. 4.7. Схема алгоритму розв'язання нелінійного рівняння методом дотичних

4.1.5. Комбінований метод

Методи хорд і дотичних дають наближення кореня з різних сторін відрізка $[a,b]$. Тому їх часто використовують в поєднанні один з одним, і процес уточнення кореня ξ нелінійного рівняння (4.1) проходить швидше .

Постановка задачі

Нехай дано рівняння $f(x)=0$, де $f(x)$ – неперервна нелінійна функція, яка на відрізку $[a,b]$ монотонна, диференційована і має єдиний корінь ξ (тобто $f(a)\cdot f(b)<0$). Потрібно знайти наближене значення кореня ξ з заданою похибкою ε .

Використаємо комбінований метод хорд і дотичних з урахуванням поведінки функції на відрізку $\left[a,b\right]$. Якщо $f'\left(x\right)f''\left(x\right)>0$, то метод хорд дає наближення кореня з недостачею, а метод дотичних – з надлишком (рис. 4.8.а,б).

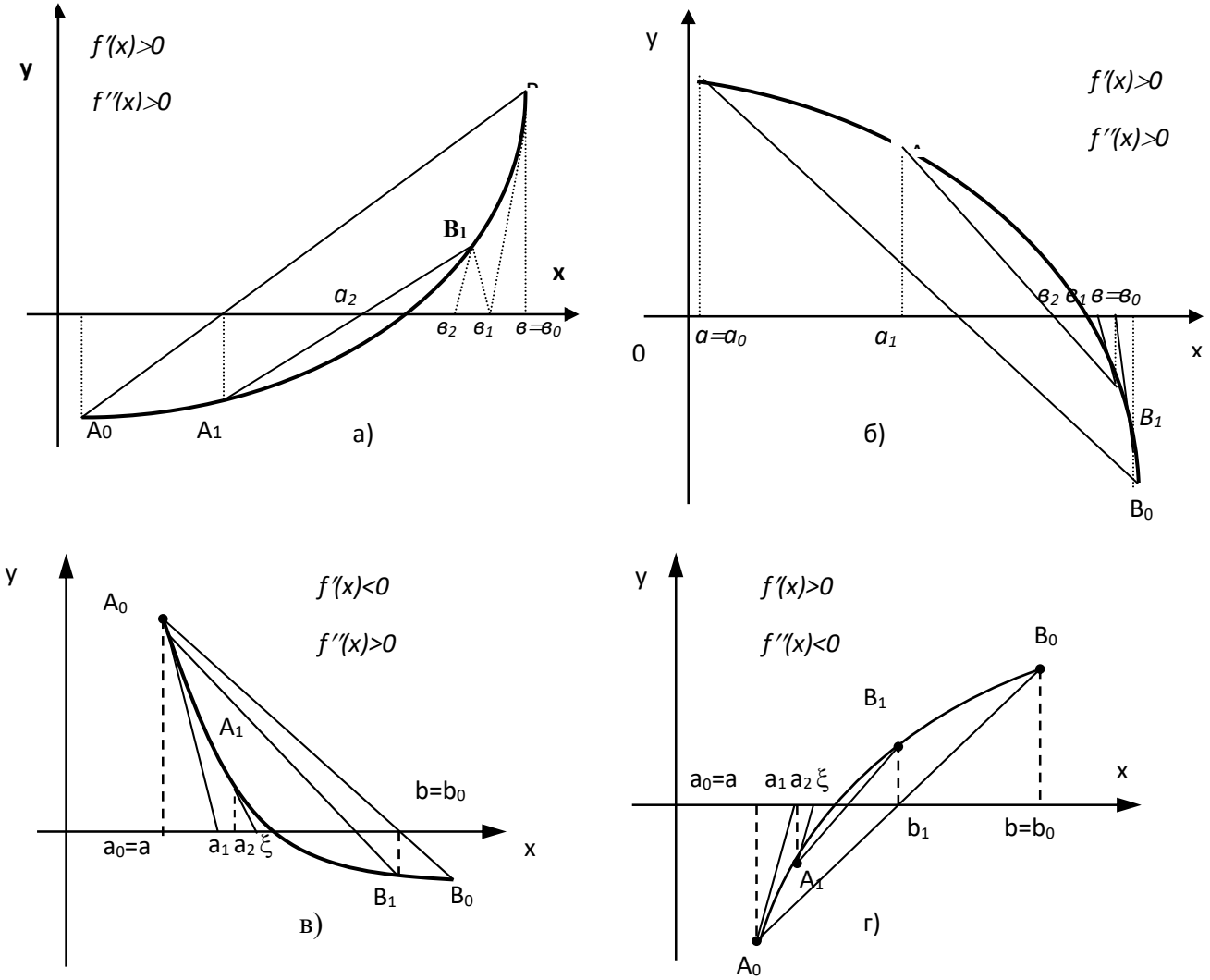


Рис. 4.8. Геометричний зміст комбінованого методу

Якщо ж $f'\left(x\right)f''\left(x\right)<0$, то методом хорд отримуємо значення кореня з надлишком, а методом дотичних – з недостачею (рис.4.8.в,г). Однак в усіх випадках справжній корінь ξ знаходиться між наближеними коренями, які отримані за методом хорд і методом дотичних, тобто виконується нерівність $a<x_n<\xi<\overline{x}_n<b$, де x_n – наближене значення кореня з недостачею, x_n – з надлишком.

Суть методу полягає в тому, що на досить малому відрізку $\left[a,b\right]$ (отриманому при відокремлені коренів) дуга функції $f(x)$ з одного кінця відрізка стягується хордою, а з другого – дотичною. Тобто, якщо сумістити обидва методи, то після знаходження коренів відрізок $\left[a,b\right]$ на кожному кроці ітерації звужується шляхом переносу кінців відрізка $\left[a,b\right]$ в точки перетину хорди та дотичної з віссю Ox.

Наближене значення кореня нелінійного рівняння визначається відповідно до таких правил:

Правило 1. Якщо добуток першої на другу похідну функції $f(x)$ більший за нуль: $f'(x)f''(x) > 0$, (рис. 4.8 а,б) то рухомим для методу хорд є кінець a, і наближене значення кореня з боку кінця a обчислюється за формулою хорд:

$$\overline{x}_{n+1} = a_n - \frac{f(a_n)\cdot(b_n - a_n)}{f(b_n) - f(a_n)}. \tag{4.14}$$

Для методу дотичних рухомим є кінець b, і наближене значення кореня обчислюється за формулою дотичних:

$$\overset{=}{x}_{n+1} = b_n - \frac{f(b_n)}{f'(b_n)}. \tag{4.15}$$

Правило 2. Якщо добуток першої на другу похідну функції $f(x)$ менший за нуль: $f'(x)f''(x) < 0$ (рис. 4.7 в, г), то рухомим для методу хорд є кінець b, і наближене значення кореня з боку кінця b обчислюється за формулою хорд:

$$\overset{=}{x}_{n+1} = b_n - \frac{f(b_n)\cdot(b_n - a_n)}{f(b_n) - f(a_n)}. \tag{4.16}$$

Для методу дотичних рухомим є кінець a, і наближене значення кореня обчислюється за формулою дотичних:

$$\overline{x}_{n+1} = a_n - \frac{f(a_n)}{f'(a_n)}. \tag{4.17}$$

Комбінований метод дуже зручний при оцінці похибки обчислень.

Ітераційний процес продовжується доти, поки не стане виконуватися нерівність $\left|\overline{\overline{x_n}} - \overline{x_n}\right| < \varepsilon$. За наближене значення кореня приймають $\xi = \frac{1}{2}\left(\overline{x_n} + \overline{\overline{x_n}}\right)$, де $\overline{x_n}$ і $\overline{\overline{x_n}}$ – наближені значення кореня відповідно з недостачею та з надлишком.

Схема алгоритму методу представлена на рис. 4.9.

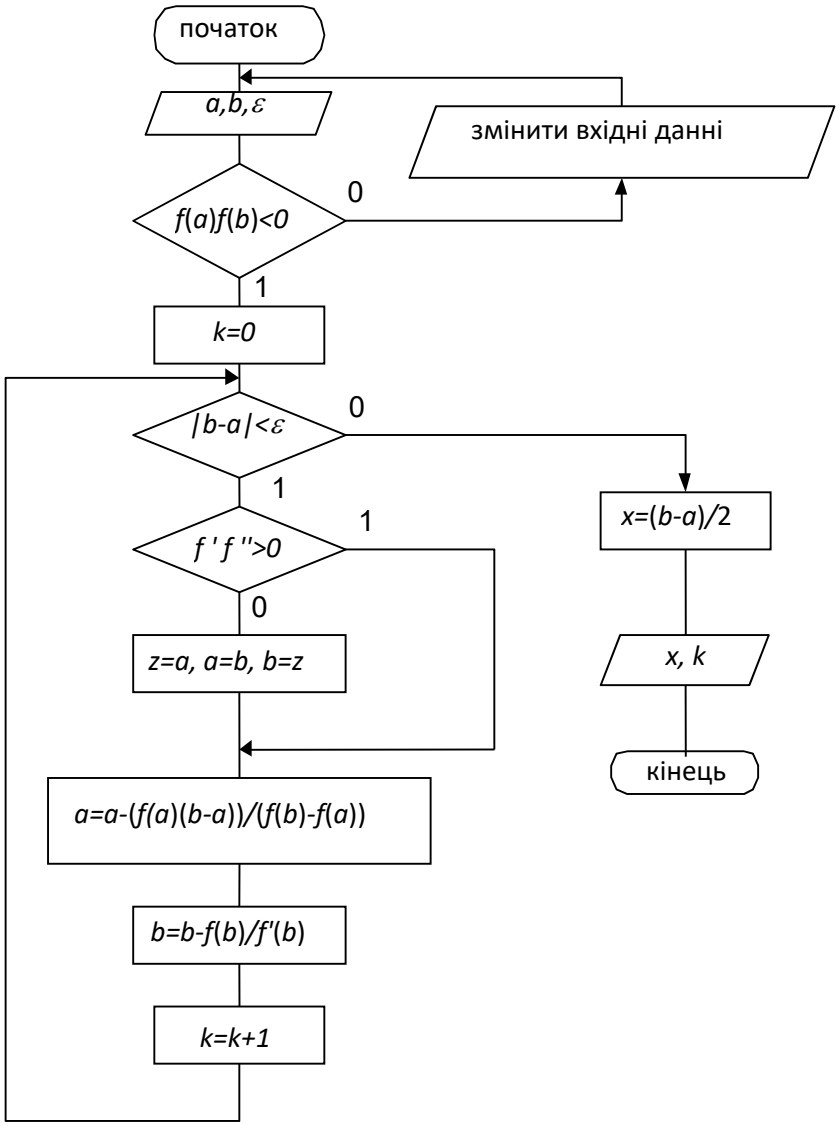


Рис.4.9. Схема алгоритму розв'язання нелінійного рівняння комбінованим методом

4.1.6. Метод ітерацій (метод послідовних наближень)

Суть методу полягає у заміні початкового рівняння

$$f(x) = 0 \tag{4.18}$$

еквівалентним йому рівнянням

$$x = \phi(x), \tag{4.19}$$

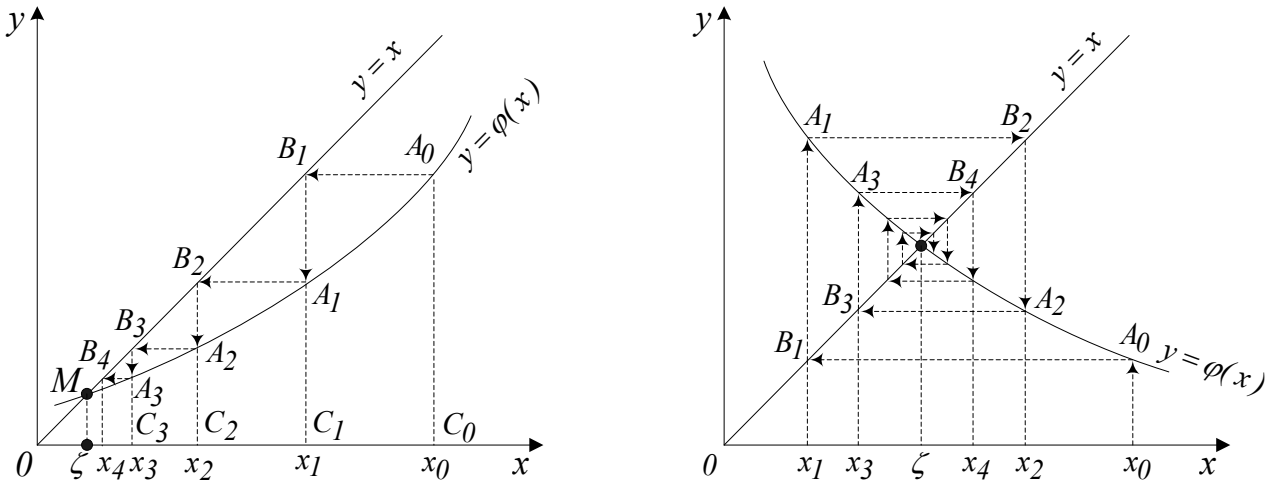
Постановка задачі

Нехай задано рівняння $f(x) = 0$, де $f(x)$ – неперервна нелінійна функція. Потрібно визначити корінь ξ цього рівняння, який знаходиться на відрізку $\left[a, b\right]$ з заданою похибкою ε.

Виберемо довільним способом $x_0 \in \left[a, b\right]$ і підставимо його в праву частину рівняння (4.10); тоді отримаємо $x_1 = \phi(x_0)$. Потім це значення x_1 підставимо знову в праву частину рівняння (4.9) і отримаємо $x_2 = \phi(x_1)$ (рис. 4.10 а,б). Повторюючи цей процес, отримаємо послідовність чисел $x_n = \phi(x_{n-1})$. При цьому можливі два випадки: послідовність *збігається,* тобто має границю, і тоді ця границя буде коренем рівняння (4.10).

послідовність *розбігається,* тобто не має границі.

Приведемо без доказу теорему, яка виражає умову, за якої ітераційний процес розв'язку нелінійного рівняння методом ітерацій на комп'ютері збігається.



а б

Рис. 4.10. Геометрична інтерпретація методу ітерацій

Теорема. Нехай на відрізку $\left[a, b\right]$ знаходиться єдиний корінь рівняння

$$x = \phi(x)$$

та у всіх точках цього відрізка похідна $f'(x)$ задовольняє нерівності $\left|\phi'(x)\right| \le q < 1$. Якщо при цьому виконується і умова $a \le \phi(x) \le b$, то ітераційний процес збігається, а за нульове наближення x_0 можна взяти число з відрізка $\left[a,b\right]$.

Розв'яжемо один етап ітерацій. Виходячи із заданого на попередньому кроці значення x_{n-1}, обчислюємо $y = \phi(x_{n-1})$. Якщо $\left|y - x_{n-1}\right| > \varepsilon$, покладемо $x_n = y$ і виконаємо наступну ітерацію. Якщо ж $\left|y - x_{n-1}\right| < \varepsilon$, то обчислення закінчують, за наближене значення кореня приймають величину $x_n = y$.

При використанні методу простих ітерацій основною операцією є вибір функції $\phi(x)$ в рівнянні $x = \phi(x)$, яку слід підібрати так, щоб $\left|\phi'(x)\right| \le q < 1$ і швидкість збіжності послідовності $\left\{x_n\right\}$ до кореня ξ тим вища, чим менше число q. Схема алгоритму методу ітерацій представлена на рис. 4.11.

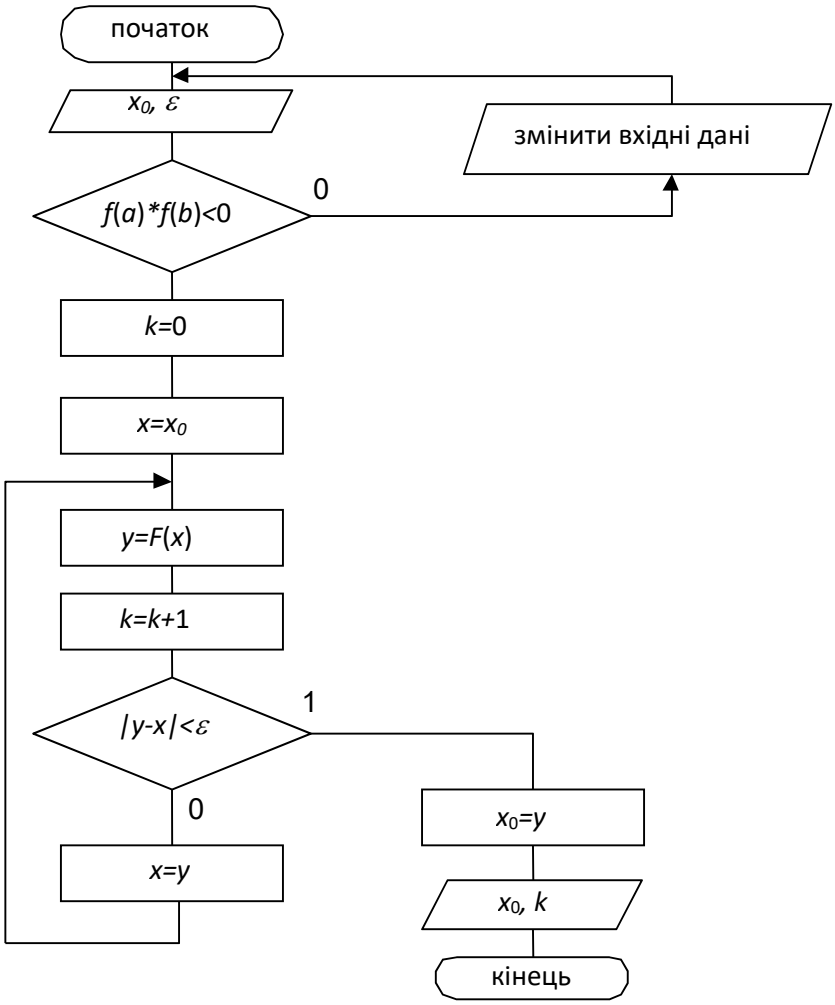


Рис. 4.11. Схема алгоритму розв'язання нелінійного рівняння методом ітерацій

4.2. Порядок виконання роботи

- Ознайомитись з методикою розв’язання нелінійних рівнянь на ПК.
- Побудувати графік функції лівої частини рівняння.
- Конкретний метод розв’язання рівняння визначає викладач.
- Скласти схему алгоритму розв’язання нелінійного рівняння, вибраного з таблиці 4.1.
- Скласти програму розв’язання нелінійного рівняння, користуючись схемою алгоритму.
- Відокремити корені в рівнянні, що досліджується. (Відокремлення коренів виконати на ПК, розробивши для цього програму).
- Вибрати значення точності обчислень ε.
- На кожному проміжку уточнити корені, користуючись розробленою програмою. Початкові дані (проміжок, точність обчислень ε) і результати роздрукувати.
- Використовуючи блок-схему алгоритму, написати програму розв’язування нелінійного рівняння мовою програмування, узгодженою з викладачем.
- Оформити звіт, що відповідає таким вимогам.

4.3. Зміст звіту

1) файл тексту програми;

2) опис алгоритму розрахунку (в текстовій формі та у вигляді блок-схеми) в електронному та роздрукованому вигляді;

3) роздруківку файлів з коментарями;

4) загальні висновки за результатами роботи, що включають результати тестування, отримані оцінки похибки результатів і обґрунтування цих оцінок.

5) інтерфейс у стилі GUI.

Таблиця 4.1

Варіанти завдань до лабораторної роботи №4

Метод	***Номер варіанту***	***Рівняння***	***Примітка***
Метод половинного ділення	*1*	$x^3 - x + 1 = 0$	*-1.325*
	2	$x^3 + 2x - 4 = 0$	*1.180*
	3	$x^4 + 5x - 3 = 0$	*-1.876; 0.578*
	4	$2.2x - 2^x = 0$	*0.781; 2.401*
	5	$2^x - 2x^2 - 1 = 0$	*0.0;0.399;6.352*
	6	$2^x - 4x = 0$	*0.310; 4.0*
Метод дотичних	*7*	$10 \cdot arctg(5-x) - 1 = 0$	*4.907*
	8	$\sin^2\left(x + \frac{\pi}{2}\right) - \frac{x^2}{4}$	*1.352*
	9	$x^3 - x - 3 = 0$	*1.213*
	10	$x^3 + 8x - 6 = 0$	*0.703*
	11	$x^3 + 10x - 9 = 0$	*0.841*
	12	$x^2 - \cos \pi x = 0$	*-0.438; 0.438*
Метод хорд	*13*	$x^2 - \sin \pi x = 0$	*0.0; 0.787*
	14	$\lg x - \frac{1}{x^2} = 0$	*1.897*
	15	$x^3 - 6x^2 + 9x - 3 = 0$	*-0.4679; 1.6527; 3.8794*
	16	$x^3 - 12x - 8 = 0$	*-0.695; -3.067;3.757*
	17	$2\lg x - \frac{x}{2} + 1 = 0$	*0.398; 4.682*
	18	$x^2 - 20\sin x = 0$	*0.0; 2.753*

	19	$x - \cos x = 0$	*0.739*
Комбінований метод	*20*	$x^3 + 6x - 5 = 0$	*0.760*
	21	$x^3 - 2x + 7 = 0$	*-2.258*
	22	$x^3 - 2x^2 + x + 1 = 0$	*-0.465*
	23	$1.8x^2 - \sin 10x = 0$	*-0.567;-0.335; 0.0*
	24	$\lg x - \dfrac{7}{(2x+6)} = 0$	*3.473*
	25	$2x \ln x - 1 = 0$	*1.422*
Метод ітерацій	*26*	$\ln x + (x+1)^3 = 0$	*0.187*
	27	$x + \lg x = 0.5$	*0.672*
	28	$tg1.5x - 2.3x = 0$	0
	29	$5 \sin 5x - x = 0$	0
	30	$0.83e^{-0.54x} - x = 0$	0

4.4. Контрольні запитання

- Дати визначення таких термінів: алгебраїчне та трансцендентне рівняння, розв'язок рівняння, корінь рівняння.
- Класифікація рівнянь.
- Суть відокремлення коренів нелінійного рівняння.
- Умова єдиності кореня рівняння *f(x)*=0 на деякому відрізку $[a,b]$.
- Умова відокремлення коренів.
- Суть методів уточнення коренів.

Порівняйте алгоритми методів хорд та комбінованого методу. Який з алгоритмів швидший? Ефективніший?

- Геометричний зміст комбінованого методу.
- Геометричний зміст методу Ньютона.
- Правила визначення рухомого та нерухомого кінця відрізка за допомогою методу хорд.

- Постановка задачі методу хорд.
- Постановка задачі методу послідовних наближень.
- Схема алгоритму розв'язання нелінійного рівняння методом ітерацій.
- Схема алгоритму розв'язання нелінійного рівняння методом половинного ділення.
- Схема алгоритму розв'язання нелінійного рівняння комбінованим методом.
- Відокремити корені рівнянь:

а) $x^3 + 3x^2 - 3 = 0$;

б) $x^3 + 3x^2 - 24x + 1 = 0$;

в) $x^3 - 6x^2 + 9x - 3 = 0$;

г) $x^3 - x - 3 = 0$

д) $x - \cos x = 0$

4.5. Інформаційні ресурси

1. Метод хорд: https://www.mathros.net.ua/znahodzhennja-nablyzhenogo-rozvjazku-nelinijnogo-algebraichnogo-rivnjannja-metodom-hord.html
2. Метод хорд: https://studfile.net/preview/1081239/page:6/
3. Метод хорд: https://vseosvita.ua/library/embed/001hjv-89d6.doc.html
4. Метод дотичних: https://www.mathros.net.ua/znahodzhennja-nablyzhenogo-rozvjazku-nelinijnogo-algebraichnogo-rivnjannja-metodom-dotychnyh.html
5. Метод Ньютона: https://studfile.net/preview/5581807/page:3/
6. Метод дотичних: https://vseosvita.ua/library/embed/001hvv-bac9.doc.html
7. Комбінований метод: https://studfile.net/preview/9982537/page:10/
8. Чисельні методи розв'язування нелінійних рівнянь: https://pns.hneu.edu.ua/pluginfile.php/293300/mod_resource/content/1/%D0%A2%D0%B5%D0%BC%D0%B0%204.pdf

Розділ 5. ЛАБОРАТОРНА РОБОТА № 5

Тема: «Розв’язання систем лінійних алгебраїчних рівнянь».

Мета: Вивчити алгоритми методів розв'язання систем лінійних алгебраїчних рівнянь на ПК.

Завдання: Відповідно до варіанту завдання скласти схему алгоритму розв’язання систем лінійних алгебраїчних рівнянь зазначеним у варіанті методом. Відповідно до блок-схеми скласти програму розв'язання систем лінійних алгебраїчних рівнянь алгоритмічною мовою, узгодженою з викладачем. Розв’язати СЛАР на комп’ютері відповідно до варіанту.

5.1. Теоретичні відомості до лабораторної роботи №5

5.1.1. Основні поняття та визначення

Системою лінійних алгебраїчних рівнянь (*СЛАР*) називають систему виду:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + ... + a_{1m}x_m = b_1 \\ a_{21}x_1 + a_{22}x_2 + ... + a_{2m}x_m = b_2 \\ a_{31}x_1 + a_{32}x_2 + ... + a_{3m}x_m = b_3 \\ ..... \\ a_{n1}x_1 + a_{n2}x_2 + ... + a_{nm}x_m = b_n \end{cases} \tag{5.1}$$

де x_i, $(i = \overline{1,n})$ – невідомі; b_i, $(i = \overline{1,n})$ – вільні члени системи;

a_{ij}, $(i, j = \overline{1,n})$ – коефіцієнти системи.

В матричному вигляді рівняння (5.1) має вигляд:

$$\mathbf{A} \times \mathbf{X} = \mathbf{B},$$

де $\mathbf{X} = \left(x_1, x_2, ..., x_n\right)^T$ – вектор невідомих; $\mathbf{B} = \left(b_1, b_2, ..., b_n\right)^T$ – вектор вільних членів;

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1m} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2m} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nm} \end{pmatrix}$$ – матриця коефіцієнтів СЛАР.

Розв'язком системи лінійних алгебраїчних рівнянь (5.1) називають вектор $\mathbf{X}$, координати якого $x_1, x_2, ..., x_n$ при підстановці у систему, що розв'язують, перетворюють кожне рівняння системи в тотожність.

Кількість невідомих m в системі називають *порядком* СЛАР. Систему лінійних алгебраїчних рівнянь називають *сумісною*, якщо вона має хоча б один ненульовий розв'язок. В протилежному випадку СЛАР називають *несумісною.* СЛАР називається *визначеною,* якщо вона має тільки один розв'язок (випадок, коли $m = n$). Систему називають *невизначеною,* якщо вона має безліч розв'язків $\left(m \neq n\right)$. Система називається *виродженою,* якщо головний визначник системи дорівнює нулю. Система називається *невиродженою*, якщо головний визначник системи не дорівнює нулю.

Дві системи називаються *еквівалентними,* якщо ці системи сумісні, визначені і мають однаковий розв'язок.

СЛАР можна розв'язати на ПК чисельними методами, якщо вона сумісна, визначена, невироджена.

До наближених методів відносяться методи, які дозволяють розв'язок системи отримати як границю послідовних k розв'язків системи (5.1) при $k \to \infty$ виду: $\overline{x} = \lim\{\overline{x}^0, \overline{x}^1, \overline{x}^2 ... \overline{x}^k\}$, де $\overline{x}^0$ – вектор розв'язку 0-го наближення, $\overline{x}^1$ – вектор розв'язку 1-го наближення і т. д., $\overline{x}^k$ – вектор розв'язку k -го наближення.

Для розв'язання СЛАР наближеними методами найбільший інтерес представляють такі методи:

- метод послідовних наближень;
- метод Гауса-Зейделя;
- метод верхньої релаксації.

Розглянемо особливості загального підходу до розв'язання СЛАР наближеними методами.

5.1.2. Класифікація методів розв'язання СЛАР на комп'ютері

Для розв'язання СЛАР на комп'ютері традиційно використовують дві групи чисельних методів, що представлені на рис.5.1:

-точні (метод Гауса, метод Гауса з вибором головного елементу, метод Гауса з одиничною матрицею, метод Гауса з перетвореною матрицею, метод Гауса-Халецького, метод Гауса-Жордана, метод Крамера);

-наближені (метод послідовних ітерацій, метод Гауса-Зейделя, метод векторів зміщень).

До **точних методів** відносять методи, які дозволяють отримати точний розв'язок системи (5.1) за відповідну кількість операцій перетворення без урахування похибок заокруглення.

До **наближених методів** відносять методи, які дозволяють отримати розв'язок системи (5.1) у вигляді границі послідовності векторів $\lim\limits_{k\to\infty}\left\{X^0, X^1, X^2, ..., X^k\right\}$, яка збігається до точного розв'язку системи.

Рис. 5.1. Класифікація чисельних методів

5.1.3. Особливості методів Гауса

Найбільш відомими з точних методів розв'язання системи лінійних алгебраїчних рівнянь (5.1) є методи Гауса, суть яких полягає в тому, що система рівнянь, яка розв'язується, зводиться до еквівалентної системи з верхньою трикутною матрицею. Невідомі знаходяться послідовними підстановками, починаючи з останнього рівняння перетвореної системи. Алгоритми Гауса складаються із виконання однотипних операцій, які легко формалізуються. Однак, точність результату й витрачений на його отримання час у більшості випадків залежить від алгоритму формування трикутної матриці системи. У загальному випадку алгоритми Гауса складаються з двох етапів:

Прямий хід, в результаті якого СЛАР (5.1), що розв'язується, перетворюється в еквівалентну систему з верхньою трикутною матрицею коефіцієнтів виду:

$$\begin{cases} a_{11} \cdot x_1 & +a_{12} \cdot x_2 & +... + & a_{1n} \cdot x_n & = b_1 \\ 0 \cdot x_1 & +a_{22} \cdot x_2 & +... + & a_{2n} \cdot x_n & = b_2 \\ \hline 0 \cdot x_1 & +0 \cdot x_2 & +... + & a_{nn} \cdot x_n & = b_n \end{cases} \tag{5.2}$$

Зворотний хід дозволяє визначити вектор розв'язку, починаючи з останнього рівняння системи (5.2) шляхом підстановки координат вектору невідомих, отриманих на попередньому кроці.

Відомо декілька різних алгоритмів отримання еквівалентної системи з верхньою трикутною матрицею. Розглянемо найбільш відомі з них.

5.1.3.1. Метод Гауса з послідовним виключенням невідомих

Метод Гауса з послідовним виключенням невідомих (базовий метод) засновано на алгоритмі, в основі якого лежить послідовне виключення невідомих вектора $\overline{x}$ з усіх рівнянь, починаючи з $\left(i+1\right)$ го, шляхом елементарних перетворень: перемноження обох частин рівняння на будь-яке число, крім нуля; додавання (віднімання) до обох частин одного рівняння відповідних частин другого рівняння, помножених на будь-яке число, крім нуля.

Суть алгоритму розглянемо на прикладі системи, яка складається з трьох лінійних алгебраїчних рівнянь з трьома невідомими:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 \end{cases} \tag{5.3}$$

1) Перевіримо, щоб принаймні один із коефіцієнтів a_{11} , a_{21} , a_{31} не дорівнював нулю. Якщо, наприклад, $a_{11} = 0$, тоді необхідно переставити рівняння так, щоб коефіцієнт при x_1 у першому рівнянні не дорівнював нулю.

2) Обчислюється множник:

$$M_2 = \frac{a_{21}}{a_{11}}. \tag{5.4}$$

3) Перше рівняння системи (5.3) множиться на M_2 і віднімається від другого рівняння системи, отриманої після перестановки рівнянь, якщо вона була необхідною. Результат обчислення має вигляд:

$$(a_{21} - M_2 a_{11})x_1 + (a_{22} - M_2 a_{12})x_2 + (a_{23} - M_2 a_{13})x_3 = b_2 - M_2 b_1, \tag{5.5}$$

але

$$a_{21} - M_2 a_{11} = a_{21} - \left(\frac{a_{21}}{a_{11}}\right) a_{11} = 0 \tag{5.6}$$

Тоді x_1 виключається із другого рівняння.

Позначимо нові коефіцієнти:

$$\begin{aligned} &a'_{22} = a_{22} - M_2 a_{12}; \quad a'_{23} = a_{23} - M_2 a_{13}; \\ &b'_2 = b_2 - M_2 b_1 \end{aligned} \tag{5.7}$$

Тоді друге рівняння системи (5.3) набуває вигляду:

$$a'_{22}\, x_2 + a'_{23}\, x_3 = b_2. \tag{5.8}$$

Далі необхідно звільнитися від коефіцієнта a_{31} при x_1 в третьому рівнянні системи (5.3) за аналогічним алгоритмом

4) Обчислюється множник для третього рівняння:

$$M_3 = \frac{a_{31}}{a_{11}}. \tag{5.9}$$

5) Перше рівняння системи (5.3) множиться на M_3 і віднімається від третього рівняння. Коефіцієнт при x_1 стає нулем, і третє рівняння набуває вигляду:

$$a'_{32}\, x_2 + a'_{33}\, x_3 = b_3, \tag{5.10}$$

де

$$a'_{32} = a_{32} - M_3 a_{12}, \tag{5.11}$$

$$a'_{33} = a_{33} - M_3 a_{13}, \tag{5.12}$$

$$b'_3 = b_3 - M_3 b_1. \tag{5.13}$$

Перетворена таким чином система рівнянь (5.3) набуває вигляду:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \\ 0 * x_1 + a'_{22}\, x_2 + a'_{23}\, x_3 = b'_2 \\ 0 * x_1 + a''_{32}\, x_2 + a''_{33}\, x_3 = b''_3 \end{cases} \tag{5.14}$$

Ця система рівнянь еквівалентна початковій і має певні переваги, оскільки x_1 входить тільки до першого рівняння. Спробуємо тепер виключити x_2 з останнього рівняння. Якщо $a_{22} = 0$, а $a_{32} \neq 0$, тоді переставимо друге й третє рівняння так, щоб $a_{22} \neq 0$. Інакше система вироджена і має безліч розв'язків.

7) Обчислюємо множник

$$M_3'' = \frac{a_{32}}{a_{22}}. \tag{5.15}$$

8) Друге рівняння системи (5.11) помножується на M_3 і віднімається від 3го рівняння:

$$(a_{32} - M_3 a_{22})x_2 + (a_{33} - M_3 a_{23})x_3 = b_3 - b_2 M_2. \tag{5.16}$$

При цьому коефіцієнт біля x_2 дорівнює нулю:

$$a'_{32} - M_3 a'_{22} = 0, \tag{5.17}$$

$$a''_{33} = a'_{33} - M_3 a'_{23}, \tag{5.18}$$

$$b''_3 = b'_3 - M_3 b'_2, \tag{5.19}$$

Отримаємо
$$a''_{33} x_3 = b''_3. \tag{5.20}$$

Замінивши в системі (5.14) третє рівняння на (5.20), отримаємо систему рівнянь виду:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \\ 0 * x_1 \ + a'_{22} x_2 + a'_{23} x_3 = b'_2 \\ 0 * x_1 \ + 0 * x_2 \quad + a''_{33} x_3 = b''_3 \end{cases} \tag{5.21}$$

Таку систему називають ***системою з трикутною матрицею коефіцієнтів,*** що еквівалентна СЛАР (5.3). Процес знаходження такої системи називається ***прямим ходом Гауса***. Знайти розв'язок такої системи просто: із 3го рівняння знайти x_3, підставити результат у друге і знайти x_2, підставити x_2 і x_3 в 1е рівняння системи (5.21) і знайти x_1 за формулами:

$$x_3 = \frac{b''_3}{a''_{33}}, \tag{5.22}$$

$$x_2 = \frac{b'_2 - a'_{23} x_3}{a'_{22}}, \tag{5.23}$$

$$x_1 = \frac{b_1 - a_{12}x_2 - a_{13}x_3}{a_{11}}. \tag{5.24}$$

Процес знаходження вектору розв'язку системи (5.3) називають ***зворотним ходом метода Гауса***. На рис. 5.2 показана схема алгоритму метода Гауса з послідовним виключенням для розв'язання системи із *N* рівнянь з *N* невідомими. Ця схема відповідає розглянутому алгоритму і може бути використана при розробці програми. Блок "Перестановка рівнянь так, щоб $a_{nn} \neq 0$ " означає деякий алгоритм, який дає змогу не допустити помилки "ділення на 0". Якщо прямувати до можливого зменшення помилок округлення, то можна використати алгоритм з вибором головного елемента.

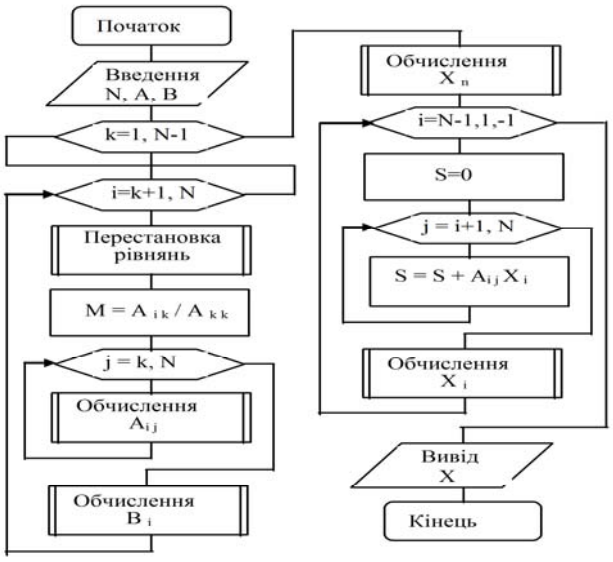


Рис. 5.2. Схема алгоритму розв’язання системи лінійних алгебраїчних рівнянь методом Гауса.

Призначення індексів в схемі алгоритму (рис. 5.2):

k – номер рівняння, яке віднімається від інших, а також номер невідомого, яке виключається із залишених *к*-рівнянь;

i – номер рівняння, із якого в даний момент виключається невідоме;

j – номер стовпця.

5.1.3.2 Метод Гауса з вибором головного елемента

Ідея цього методу виникла у зв’язку з тим, що коефіцієнти СЛАР є параметрами реальних інженерних систем та в більшості є наближеними значеннями, тому що отримані звичайно в результаті вимірювання або як статистичні дані. Для таких систем рівнянь при обчисленні масштабного множника

$$M = \frac{a_{ik}}{a_{kk}} \tag{5.25}$$

можлива ситуація при визначені a_{kk} , що ділення наближеного числа a_{ik} на достатньо мале число a_{kk} призводить до різкого збільшення похибки методу. Тому для того, щоб не збільшувати похибку результату, необхідно виконувати такі дії:

1) в системі (5.1) необхідно знайти з k -го стовпця найбільший за абсолютним значенням коефіцієнт a_{kj} ;

2) переставити k -те рівняння з рівнянням, у якому знаходиться цей максимальний коефіцієнт;

3) масштабний множник буде обчислюватись за формулою (5.25), де a_{kk} – максимальний коефіцієнт, а тому похибка розв'язання СЛАР у результаті арифметичних операцій не збільшується.

Схема алгоритму метода Гауса з вибором головного елемента (прямий та обернений хід) показана на рис. 5.3.

5.1.3.3. Метод Гауса з одиничними коефіцієнтами

В цьому методі зроблена спроба зменшити недоліки перших двох методів, пов'язаних з багаторазовим діленням одного наближеного числа на інше. Для цього перед введенням масштабного множника k -те рівняння системи ділиться один раз на діагональний елемент a_{kk} так, щоб коефіцієнт при $x_k = 1$, а масштабний множник M_i буде дорівнювати a_{kj} . Результатом прямого ходу є система, еквівалентна СЛАР (1), з одиничними коефіцієнтами на головній діагоналі виду:

$$\begin{cases} 1 * x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1 \\ 0 * x_1 + 1 * x_2 + ... + a_{2n}x_n = b_2 \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ 0 * x_1 + 0 * x_2 + ... + 1 * x_n = b_n \end{cases} \tag{5.26}$$

Дана система схожа на систему (5.2), яка отримується в результаті прямого ходу базового методу Гауса з послідовним вилученням невідомих і відрізняється від неї тільки діагональними коефіцієнтами. Для отримання такої системи необхідно використовувати алгоритм, який включає в себе наступні етапи:

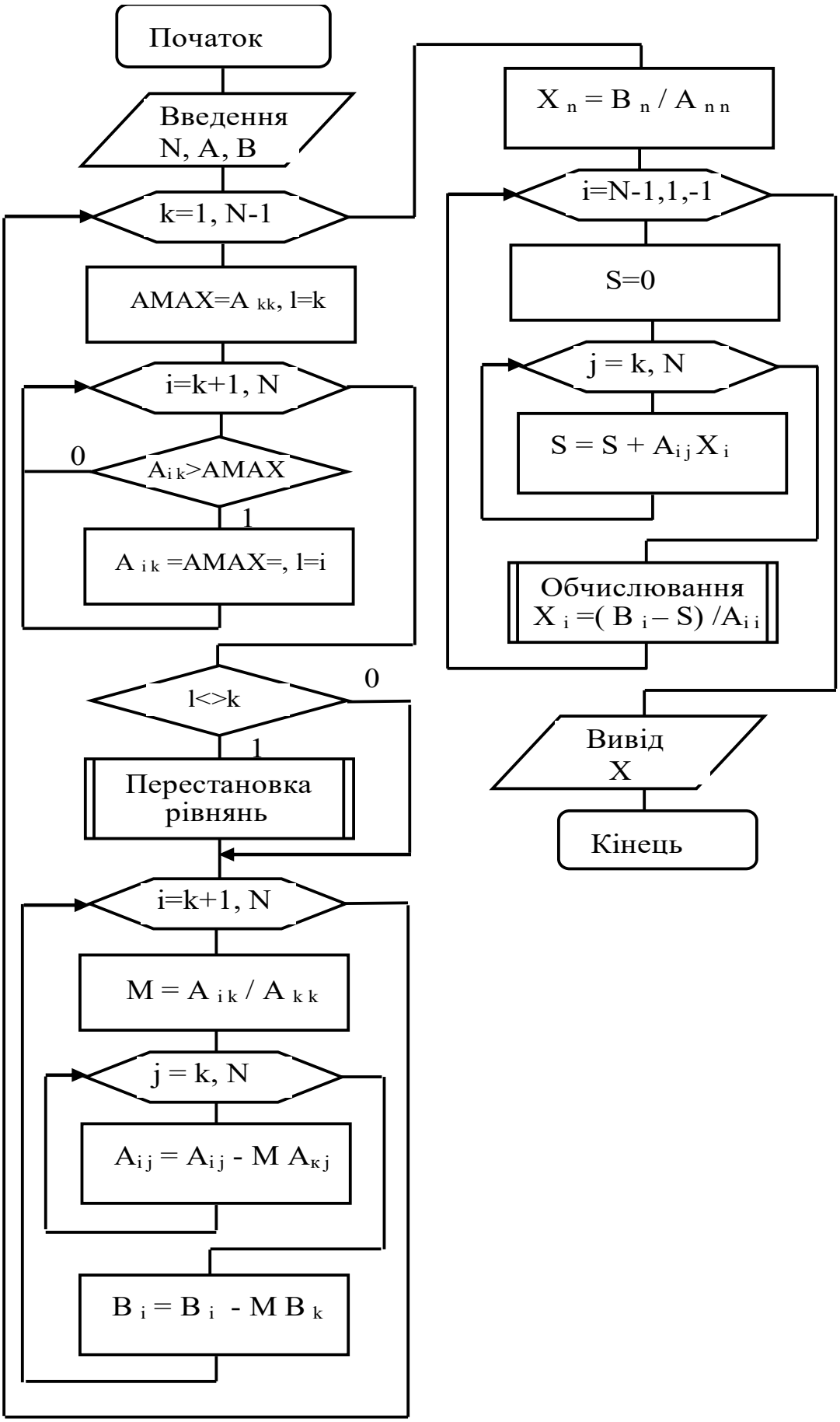


Рис. 5.3. Схема алгоритму метода Гауса з вибором головного елемента

- Організація циклу по всіх рівняннях від 1 до N-1 (k = 1, 2, …, N-1).
- В кожному k -му стовпці визначається номер l-го рівняння з головним елементом (тобто номер l-го рівняння, в якому знаходиться коефіцієнт при x_n зі всіх рівнянь, починаючи з k -го до N-го).
- Якщо номер цього рівняння l не дорівнює k $\left(l <> k\right)$, тоді необхідно переставити місцями l-е рівняння з k -м.
- Нормування k -го рівняння, тобто ділення всіх коефіцієнтів k -го рівняння на a_{kk} (головний елемент при x_n), включаючи b_n.
- Перетворення всіх $i-$ х рівнянь, починаючи з $\left(k+1\right)$ до N у відповідності з базовим алгоритмом Гауса з метою отримання еквівалентної системи з верхньою трикутною матрицею коефіцієнтів.
- Кінець циклу по k.

Формула зворотного ходу для систем виду (5.26) спрощується і має вигляд:

$$\begin{aligned} & x_n = b_n \\ & \dots\dots\dots\dots\dots\dots\dots\dots \\ & x_2 = b_2 - a_{2n}x_n \\ & x_1 = b_1 - a_{1n}x_n - a_{12}x_2 \end{aligned} \tag{5.27}$$

Схема алгоритму методу Гауса з одиничними діагональними коефіцієнтами наведена на рис. 5.4.

5.1.4. Загальний підхід до розв'язання СЛАР наближеними методами

Розглянемо систему лінійних алгебраїчних рівнянь виду:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ a_{31}x_1 + a_{32}x_2 + \dots + a_{3n}x_n = b_3 \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n \end{cases} \tag{5.28}$$

Для розв'язання системи (5.28) методами послідовних наближень необхідно виконати наступні кроки:

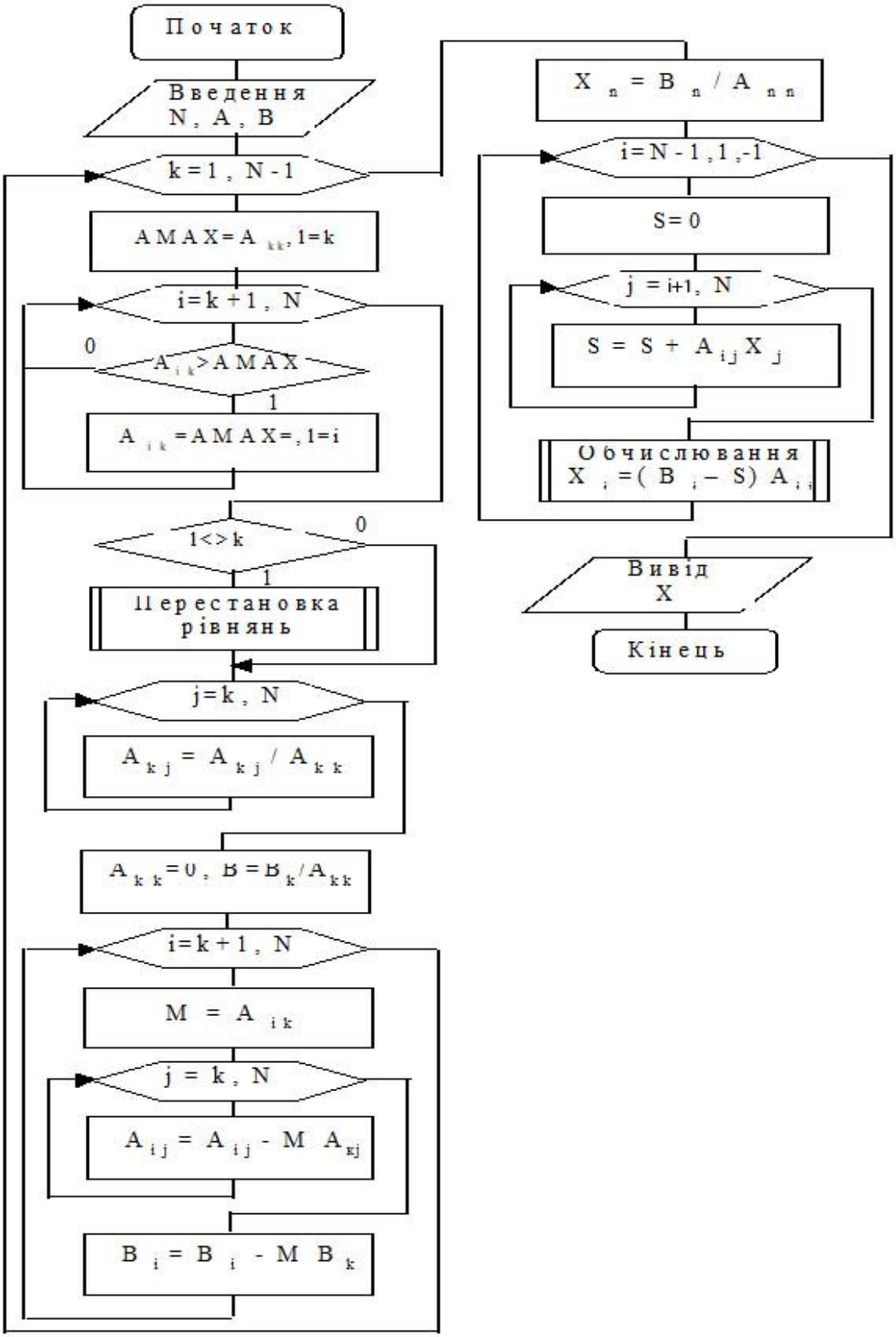


Рис. 5.4. Схема алгоритму метода Гауса з одиничними коефіцієнтами

1) Кожне рівняння системи розділити на діагональний елемент a_{kk}, де $k = 1,2,...,n$, n – кількість рівнянь в системі, і перетворити кожне рівняння системи відносно координат вектора, індекс якого співпадає з номером рівняння:

$$\begin{cases} x_1 = \dfrac{b_1}{a_{11}} - (\dfrac{a_{12}}{a_{11}} x_2 + \dfrac{a_{13}}{a_{11}} x_3 + ... + \dfrac{a_{1n}}{a_{11}} x_n) \\ x_2 = \dfrac{b_2}{a_{22}} - (\dfrac{a_{21}}{a_{22}} x_1 + \dfrac{a_{23}}{a_{22}} x_3 + ... + \dfrac{a_{2n}}{a_{22}} x_n) \\ x_3 = \dfrac{b_3}{a_{33}} - (\dfrac{a_{31}}{a_{33}} x_1 + \dfrac{a_{32}}{a_{33}} x_2 + ... + \dfrac{a_{1n}}{a_{33}} x_n) \\ ..... \\ x_n = \dfrac{b_n}{a_{nn}} - (\dfrac{a_{n1}}{a_{nn}} x_1 + \dfrac{a_{n2}}{a_{nn}} x_2 + ... + \dfrac{a_{nn-1}}{a_{nn}} x_{n-1}) \end{cases} \tag{5.29}$$

2) Нехай $\dfrac{b_k}{a_{kk}} = \beta_k$, а $-\dfrac{a_{ki}}{a_{kk}} = \alpha_{ki}$, де $k = 1,2,...,n$; $i = 1,2,...,n$. Тоді система матиме вигляд:

$$\begin{cases} x_1 = \beta_1 + \alpha_{11} x_2 + \alpha_{12} x_2 + ... + \alpha_{1n} x_n \\ x_2 = \beta_2 + \alpha_{11} x_1 + \alpha_{22} x_2 + ... + \alpha_{2n} x_n \\ x_3 = \beta_3 + \alpha_{11} x_1 + \alpha_{22} x_2 + ... + \alpha_{3n} x_n \\ ..... \\ x_n = \beta_n + \alpha_{11} x_1 + \alpha_{22} x_2 + ... + \alpha_{nn} x_n \end{cases} \tag{5.30}$$

Така система називається *зведеною до нормального вигляду*.

3) Представимо систему (30) в матричному вигляді:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ ... \\ x_n \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ ... \\ \beta_n \end{bmatrix} + \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & ... & \alpha_{1n} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & ... & \alpha_{n2} \\ \alpha_{311} & \alpha_{32} & \alpha_{33} & ... & \alpha_{3n} \\ ..... & ..... & ..... & ..... & ..... \\ \alpha_{n11} & \alpha_{n2} & \alpha_{n3} & ... & \alpha_{nn} \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ ... \\ x_n \end{bmatrix}, \tag{5.31}$$

або векторному $\bar{x} = \bar{\beta} + \bar{\alpha} * \bar{x}$. (5.32)

Якщо деяким чином визначити так званий вектор початкових значень $\overline{x}^{(0)}$, який знаходиться в правій частині (5.32), то можна отримати певні значення вектора $\overline{x}$.

В якості вектора початкових наближень $\overline{x}^{(0)}$ вибирають:

- вектор, в якого всі координати x_i дорівнюють 0;
- вектор, в якого всі координати x_i дорівнюють 1;
- вектор, координати x_i якого дорівнюють координатам вектора вільних членів β_i;
- координати вектора $\overline{x}$ вибирають в результаті аналізу особливостей об'єкта дослідження та задачі, яка розв'язується.

4) Якщо вектор початкових наближень $\overline{x}^{(0)}$ підставити в праву частину системи (5.31) або (5.32), то вона набуває вигляду:

$$\begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \\ \dots \\ x_n^{(1)} \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \dots \\ \beta_n \end{bmatrix} + \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \dots & \alpha_{1n} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \dots & \alpha_{n2} \\ \alpha_{311} & \alpha_{32} & \alpha_{33} & \dots & \alpha_{3n} \\ \dots\dots & \dots\dots & \dots\dots & \dots\dots & \dots\dots \\ \alpha_{n11} & \alpha_{n2} & \alpha_{n3} & \dots & \alpha_{nn} \end{bmatrix} * \begin{bmatrix} x_1^{(0)} \\ x_2^{(0)} \\ x_3^{(0)} \\ \dots \\ x_n^{(0)} \end{bmatrix}$$

або $\overline{x}^{(1)} = \overline{\beta} + \overline{\alpha} * \overline{x}^{(0)}$,

легко розв'язується, тому що в правій частині містить всі визначені елементи, і дозволяє отримати розв'язок системи, який називається *вектором першого наближення* $\overline{x}^{(1)}$.

- Перевіряється виконання умови закінчення ітераційного процесу пошуку розв'язку системи (5.28) виду:

$$| \overline{x}^{(1)} - \overline{x}^{(0)} | \leq \varepsilon, \qquad (5.33),$$

де ε - задана похибка результатів розв'язання задачі.

Якщо умова (5.33) не виконується, то $x^{(1)}$ підставляється в праву частину (5.31) або (5.32) і знаходиться $x^{(2)}$ з системи виду:

$$\begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \\ \dots \\ x_n^{(2)} \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \dots \\ \beta_n \end{bmatrix} + \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \dots & \alpha_{1n} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \dots & \alpha_{n2} \\ \alpha_{311} & \alpha_{32} & \alpha_{33} & \dots & \alpha_{3n} \\ \dots\dots & \dots\dots & \dots\dots & \dots\dots & \dots\dots \\ \alpha_{n11} & \alpha_{n2} & \alpha_{n3} & \dots & \alpha_{nn} \end{bmatrix} * \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \\ \dots \\ x_n^{(1)} \end{bmatrix}$$

або $\overline{x}^{(2)} = \overline{\beta} + \overline{\alpha} * \overline{x}^{(1)}$.

- Знову перевіряється виконання умови закінчення ітераційного процесу пошуку розв'язку системи (5.28)

$$| \overline{x}^{(2)} - \overline{x}^{(1)} | \leq \varepsilon .$$

Якщо умова не виконується, то $x^{(2)}$ підставляється в праву частину (5.31) і знаходиться $x^{(3)}$ з системи виду:

$$\begin{bmatrix} x_1^{(3)} \\ x_2^{(3)} \\ x_3^{(3)} \\ \dots \\ x_n^{(3)} \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \dots \\ \beta_n \end{bmatrix} + \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \dots & \alpha_{1n} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \dots & \alpha_{n2} \\ \alpha_{311} & \alpha_{32} & \alpha_{33} & \dots & \alpha_{3n} \\ \dots\dots & \dots\dots & \dots\dots & \dots\dots & \dots\dots \\ \alpha_{n11} & \alpha_{n2} & \alpha_{n3} & \dots & \alpha_{nn} \end{bmatrix} * \begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \\ \dots \\ x_n^{(2)} \end{bmatrix}.$$

і т. д.

7) Етапи 4 та 5 повторюються доти, поки на якому-небудь k –ому кроці не виконається умова

$$| \overline{x}^{(k)} - \overline{x}^{(k-1)} | \leq \varepsilon . \qquad (5.34)$$

Таким чином, процес пошуку розв'язку системи (5.28) наближеними методами з заданою похибкою ε є ***ітераційним***, а умовою виходу з цього процесу є умова (5.34).

5.1.5. Умови збіжності ітераційного процесу

Описаний вище алгоритм дозволяє отримати розв'язок системи (28), близький до точного (з заданою похибкою ε) тільки в тому випадку, коли ітераційний процес пошуку розв'язку СЛАР збігається.

Теорема про збіжність. Ітераційний процес пошуку розв'язку системи лінійних алгебраїчних рівнянь виду (5.31) наближеними методами збігається, якщо будь-яка канонічна норма матриці $\|\alpha\| < 1$.

Канонічною нормою матриці називається будь-яке дійсне додатне число, яке визначається за такими умовами:

перша канонічна норма – це максимальна з сум модулів елементів матриці коефіцієнтів α по рядках:

$$\|\alpha\|_1 = \max_i \sum_{j=1}^{n} |\alpha_{ij}|, \tag{5.34}$$

друга канонічна норма – це максимальна з сум модулів елементів матриці коефіцієнтів α по стовпцях:

$$\|\alpha\|_2 = \max_j \sum_{i=1}^{n} |\alpha_{ij}|, \tag{5.35}$$

третя канонічна норма – це корінь квадратний з сум квадратів модулів всіх елементів матриці коефіцієнтів α:

$$\|\alpha\|_3 = \sqrt{\sum_i \sum_j |\alpha_{ij}|^2}.$$

Наслідок 1: Ітераційний процес розв'язання системи (5.30) збігається, якщо сума модулів елементів рядків матриці коефіцієнтів α або сума модулів елементів її стовпців менше за одиницю, тобто виконується умова

$$\max_i \sum_{j=1}^{n} |\alpha_{ij}| < 1 \quad \text{або} \quad \text{умова} \quad \max_j \sum_{i=1}^{n} |\alpha_{ij}| < 1. \tag{5.36}$$

Наслідок 2: Ітераційний процес розв'язання системи (5.30) збігається, якщо елементи головної діагоналі більше суми модулів елементів відповідного рядка, крім діагонального елемента цього рядка, тобто виконується умова:

$$\left|\alpha_{ii}\right| > \max_{i}\sum_{j=1}^{n}\left|\alpha_{ij}\right| \quad \text{або умова} \quad \left|\alpha_{jj}\right| > \max_{j}\sum_{i=1}^{n}\left|\alpha_{ij}\right|. \tag{5.37}$$

Приклад.

Визначити, чи збігається ітераційний процес для системи рівнянь

$$\begin{cases} 8x_1 + x_2 + x_3 = 20 \\ x_1 + 5x_2 - x_3 = 7 \\ x_1 - x_2 + 5x_3 = 7 \end{cases} \quad \text{або} \quad \begin{cases} x_1 = 3{,}25 - 0{,}125x_2 - 0{,}125x_3 \\ x_2 = 1{,}4 - 0{,}2x_1 + 0{,}2x_3 \\ x_3 = 1{,}4 - 0{,}2x_1 + 0{,}2x_2 \end{cases}$$

Матриця $\overline{\alpha}$ даної системи має вигляд:

$$\alpha = \begin{bmatrix} 0 & -0{,}125 & -0{,}125 \\ -0{,}2 & 0 & 0{,}2 \\ -0{,}2 & 0{,}2 & 0 \end{bmatrix}$$

Визначаємо норми:

$$\left\|\alpha\right\|_1 = \max\begin{Bmatrix} \left|\alpha_{11}\right| + \left|\alpha_{12}\right| + \left|\alpha_{13}\right| \\ \left|\alpha_{21}\right| + \left|\alpha_{22}\right| + \left|\alpha_{23}\right| \\ \left|\alpha_{31}\right| + \left|\alpha_{32}\right| + \left|\alpha_{33}\right| \end{Bmatrix} =$$

$$= \max\begin{Bmatrix} 0 + 0{,}125 + 0{,}125 \\ 0{,}2 + 0 + 0{,}2 \\ 0{,}2 + 0{,}2 + 0 \end{Bmatrix} = \max\begin{Bmatrix} 0{,}25 \\ 0{,}4 \\ 0{,}4 \end{Bmatrix} = 0{,}4 < 1$$

$$\|\alpha\|_2 = \max\begin{Bmatrix} |\alpha_{11}| + |\alpha_{21}| + |\alpha_{31}| \\ |\alpha_{12}| + |\alpha_{22}| + |\alpha_{23}| \\ |\alpha_{31}| + |\alpha_{32}| + |\alpha_{33}| \end{Bmatrix} = \max\begin{Bmatrix} 0 + 0,2 + 0,2 \\ 0,125 + 0 + 0,2 \\ 0,125 + 0,2 + 0 \end{Bmatrix} =$$

$$= \max\begin{Bmatrix} 0,4 \\ 0,325 \\ 0,325 \end{Bmatrix} = 0,4 < 1 .$$

Таким чином, перша та друга канонічні норми менші за одиницю, тобто ітераційний процес для даної системи збігається.

Розглянемо особливості алгоритмів наближених методів.

5.1.6. Метод послідовних наближень (метод Якобі)

хай задана система лінійних алгебраїчних рівнянь виду (5.28). Метод послідовних наближень (метод Якобі) відноситься до ітераційних методів, тому потребує перетворення даної системи до нормального вигляду (5.31) та пошуку канонічних норм матриці $\overline{\alpha}$, для того, щоб визначити умови збіжності ітераційного процесу (5.34)–(5.36) пошуку розв'язку системи з заданою похибкою ε відповідно до теореми про збіжність. Якщо жодна з умов (5.34) – (5.36) не виконується, то дану систему необхідно перетворити за певними правилами та знову перевірити умови збіжності ітераційного процесу (5.34) – (5.36). Якщо жодна з умов знову не виконується, то метод послідовних наближень не має сенсу використовувати. Якщо хоча б одна з умов (5.34)–(5.36) виконується, то ітераційний процес пошуку розв'язку системи з заданою похибкою ε збігається, і метод послідовних наближень можна використовувати.

По-перше, вибирається певне значення вектора початкових наближень $\overline{x}^{(0)}$, яке підставляється в праву частину системи рівнянь виду:

$$\begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \\ \dots \\ x_n^{(1)} \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \dots \\ \beta_n \end{bmatrix} + \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \dots & \alpha_{1n} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \dots & \alpha_{n2} \\ \alpha_{311} & \alpha_{32} & \alpha_{33} & \dots & \alpha_{3n} \\ \dots\dots & \dots\dots & \dots\dots & \dots\dots & \dots\dots \\ \alpha_{n11} & \alpha_{n2} & \alpha_{n3} & \dots & \alpha_{nn} \end{bmatrix} * \begin{bmatrix} x_1^{(0)} \\ x_2^{(0)} \\ x_3^{(0)} \\ \dots \\ x_n^{(0)} \end{bmatrix}, \qquad (5.38)$$

що легко розв'язується для знаходження вектора розв'язку першого наближення $\overline{x}^{(1)}$, тому що в правій частині містить всі визначені елементи.

По-друге, перевіряється виконання умови закінчення ітераційного процесу виду: $| \overline{x}^{(1)} - \overline{x}^{(0)} | \leq \varepsilon$,

де ε - задана похибка результатів розв'язання задачі. Якщо умова не виконується, то $x^{(1)}$ підставляється в праву частину системи (5.31) і знаходиться $x^{(2)}$:

$$\begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \\ \dots \\ x_n^{(2)} \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \dots \\ \beta_n \end{bmatrix} + \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \dots & \alpha_{1n} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \dots & \alpha_{n2} \\ \alpha_{311} & \alpha_{32} & \alpha_{33} & \dots & \alpha_{3n} \\ \dots\dots & \dots\dots & \dots\dots & \dots\dots & \dots\dots \\ \alpha_{n11} & \alpha_{n2} & \alpha_{n3} & \dots & \alpha_{nn} \end{bmatrix} * \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \\ \dots \\ x_n^{(1)} \end{bmatrix}$$

та знову перевіряється виконання умови закінчення ітераційного процесу виду: $| \overline{x}^{(2)} - \overline{x}^{(1)} | \leq \varepsilon$.

За аналогією будь-яке $(k+1)$-е наближення можна обчислити за формулою:

$$x^{(\overline{k}+1)} = \overline{\beta} + \overline{\alpha} * x^{(\overline{k})}, \quad \text{де } k = 1,2,... \tag{5.39}$$

Якщо послідовність $x^{(0)}, x^{(1)}, x^{(2)}, ..., x^{(k)}$, отримана в результаті ітераційного процесу, має границю $x = \lim\limits_{k \to \infty} x^k$, то ця границя є розв'язком системи. Умова закінчення ітераційного процесу має вигляд:

$$| x^{(\overline{k}+1)} - x^{(k)} | < \varepsilon, \tag{5.40}$$

де ε - задана похибка результатів розв'язання системи.

Алгоритмічно перевірка умови (5.40) представляє собою алгоритм пошуку максимального відхилення між координатами вектора $\overline{x}^{(k)}$ і $\overline{x}^{(k-1)}$ і порівняння його з заданою похибкою ε.

Алгоритм методу послідовних наближень зображено на рис. 5.5.

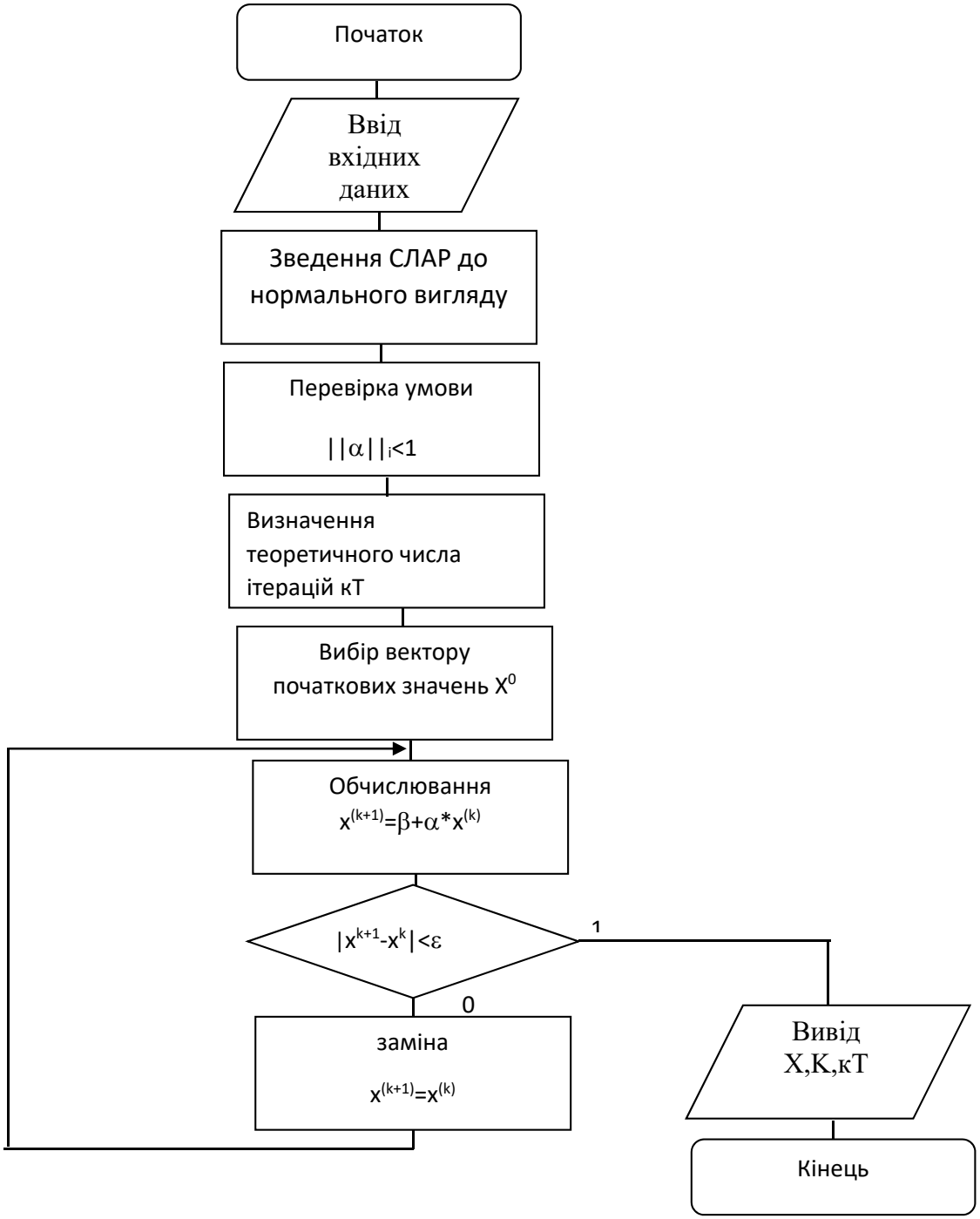


Рис. 5.5. Схема алгоритму методу послідовних наближень

Оцінка похибки методу Якобі

Якщо задана допустима похибка обчислень ε і x – вектор точного розв'язку системи лінійних рівнянь, а $x_j^{(k)}$ – k-те наближення до вектору точного розв'язку, то для оцінки похибки метода послідовних наближень використовується формула:

$$\left\| x_j - x_j^k \right\| \le \frac{\left\| \alpha \right\|^{(k+1)}}{1 - \left\| \alpha \right\|} \left\| \beta \right\|, \qquad (5.41)$$

де $\|\alpha\|$ - одна з 3 норм матриці α; $\|\beta\|$ - аналогічна норма вектора β; k - кількість ітерацій, необхідна для досягнення потрібної точності ε.

5.1.7. Метод Гауса-Зейделя

Метод Зейделя є модифікацію методу послідовних наближень, причому у методі Зейделя при обчисленні i -ої координати вектора розв'язку $(k+1)$ -го наближення використовуються значення всіх $(i-1)$ координат вектора $(k+1)$-е наближення, обчислені раніше. Розглянемо метод більш детально.

Нехай початкова система лінійних алгебраїчних рівнянь приведена до нормального вигляду:

$$\begin{cases} x_1 = \beta_1 + \alpha_{11}x_1 + \alpha_{12}x_2 + ... + \alpha_{1n}x_n; \\ x_2 = \beta_2 + \alpha_{21}x_1 + \alpha_{22}x_2 + ... + \alpha_{2n}x_n; \\ ..... \\ x_n = \beta_n + \alpha_{n1}x_1 + \alpha_{n2}x_2 + ... + \alpha_{nn}x_n. \end{cases} \tag{5.42}$$

Алгоритм Гауса-Зейделя

- Вибрати значення координат вектора початкових наближень

$$\overline{x} = \left\{ x_1^{(0)}, ..., x_n^{(0)} \right\}$$

- Визначити значення першої координати $x_1^{(1)}$ вектора першого наближення з першого рівняння системи:

$$x_1^{(1)} = \beta_1 + \alpha_{11}x_1^{(0)} + \alpha_{12}x_2^{(0)} + ... + \alpha_{1n}x_n^{(0)}.$$

- Підставити в друге рівняння системи значення першої координати $x_1^{(1)}$, яке обчислено на попередньому кроці

$$x_2^{(1)} = \beta_1 + \alpha_{21}x_1^{(1)} + \alpha_{22}x_2^{(0)} + ... + \alpha_{2n}x_n^{(0)}.$$

- Отримані значення координат першого наближення $x_1^{(1)}$, $x_2^{(1)}$ підставляємо у третє рівняння системи (5.39)

$$x_3^{(1)} = \beta_1 + \alpha_{21}x_1^{(1)} + \alpha_{22}x_2^{(1)} + \alpha_{23}x_3^{(0)} + ... + \alpha_{2n}x_n^{(0)}$$

для знаходження третьої координати і т. д.

- Для знаходження останньої координати вектора першого наближення $x_n^{(1)}$ в останнє рівняння системи треба підставити значення всіх $\left(n-1\right)$ координат ($x_1^{(1)}, x_2^{(1)}, x_{3,\ldots,}^{(1)} x_{n-1}^{(1)}$), які отримані на попередніх кроках, та значення координати $x_n^{(0)}$

$$x_n^{(1)} = \beta_n + \alpha_{n1} x_1^{(1)} + \alpha_{n2} x_2^{(1)} + \ldots + \alpha_{n,n-1} x_{n-1}^{(1)} + \alpha_{nn} x_n^{(0)}.$$

6. Аналогічно будують друге, третє та інші наближення. Так, для вектора $\left(k+1\right)$-го наближення за методом Зейделя використовують наступні формули:

$$\begin{cases} x_1^{(k+1)} = \beta_1 + \sum_{j=1}^{n} \alpha_{1j} x_j^{(k)}; \\ x_2^{(k+1)} = \beta_2 + \sum_{j=2}^{n} \alpha_{2j} x_j^{(k)} + \alpha_{21} x_j^{(k+1)}; \\ x_3^{(k+1)} = \beta_3 + \alpha_{31} x_1^{(k+1)} + \alpha_{32} x_2^{(k+1)} + \sum_{j=3}^{n} \alpha_{3j} x_j^{(k)}; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ x_n^{(k+1)} = \beta_n + \sum_{j=1}^{n-1} \alpha_{nj} x_j^{(k+1)} + \alpha_{nn} x_n^{(k)}. \end{cases} \tag{5.43}$$

Умови збіжності ітераційного процесу Зейделя

Даний процес розв'язання СЛАР – ітераційний, тому важливим є аналіз умов збіжності ітераційного процесу. Процес Зейделя для системи лінійних рівнянь $\overline{x} = \overline{\beta} + \overline{\alpha} x$ збігається до точного розв'язку з заданою похибкою ε при будь-якому виборі вектора початкових наближень, якщо будь-яка норма матриці $\overline{\alpha}$ менша 1, тобто якщо:

$$\left\|\alpha\right\|_1 = \max \sum_{j=1}^{n} \left|\alpha_{ij}\right| < 1 \tag{5.44}$$

$$\left\|\alpha\right\|_2 = \max \sum_{i=1}^{n} \left|\alpha_{ij}\right| < 1 \tag{5.45}$$

$$\|\alpha\|_3 = \sqrt{\sum_i \sum_j |\alpha_{ij}|^2} < 1 \qquad (5.46)$$

Відомо, що процес Зейделя сходиться до точного розв'язку СЛАР швидше, ніж метод послідовних наближень.

Приклад. Знайти першу та другу норми та проаналізувати умови збіжності ітераційного процесу для матриці α, яка має вигляд :

$$\overline{\alpha} = \begin{bmatrix} 0{,}24 & -0{,}05 & -0{,}24 \\ -022 & 0{,}09 & -0{,}44 \\ 0{,}13 & -0{,}02 & 0{,}42 \end{bmatrix}$$

$$\|\alpha\|_1 = \max_j \sum_{j=1}^{n} |\alpha_{ij}| = \max\{0{,}53;\ 0{,}75;\ 0{,}57\} = 0{,}75 < 1.$$

Очевидно, що процес ітерації для даної системи сходиться до точного розв'язку, незважаючи на те, що

$$\|\alpha\|_2 = \max_i \sum_{j=1}^{n} |\alpha_{ij}| = \max\{0{,}59;\ 0{,}16;\ 1{,}1\} = 1{,}1 > 1.$$

Оцінка похибки методу Гауса-Зейделя

Якщо $\overline{x}$ – точне значення вектора розв'язку системи лінійних рівнянь; а $\overline{x}^{(k)}$ - к-е наближення, обчислене за методом Гауса-Зейделя, то для оцінки похибки цього методу використовується формула:

$$\left\|\overline{x} - \overline{x}^{(k)}\right\|_1 \le \frac{\|\alpha\|_1^{(k)}}{1 - \|\alpha\|_1} \left\|\overline{x}^{(1)} - \overline{x}^{(0)}\right\|_1 \qquad (5.47)$$

5.1.8. Метод верхньої релаксації

В основі методу верхньої релаксації використовується алгоритм та обчислювальна схема метода Гауса-Зейделя, але на відміну від нього нові значення координат вектора k -го наближення визначаються за формулами:

$$x_i^{(k+1)} = x_i^{(k)} + \varpi(\overline{x}_i^{(k+1)} - x_i^{(k)}), \qquad (5.48)$$

де $\overline{x}_i^{(k+1)}$ - уточнене значення змінної по методу Гауса-Зейделя, ϖ - параметр релаксації, значення якого визначається з інтервалу $1 \le \varpi \le 2$. При ϖ=1 метод тотожний методу Гауса-Зейделя. Швидкість збіжності ітераційного процесу залежить від значення ϖ. Приклад блок-схеми алгоритму за методом релаксації наведено на рис. 5.6.

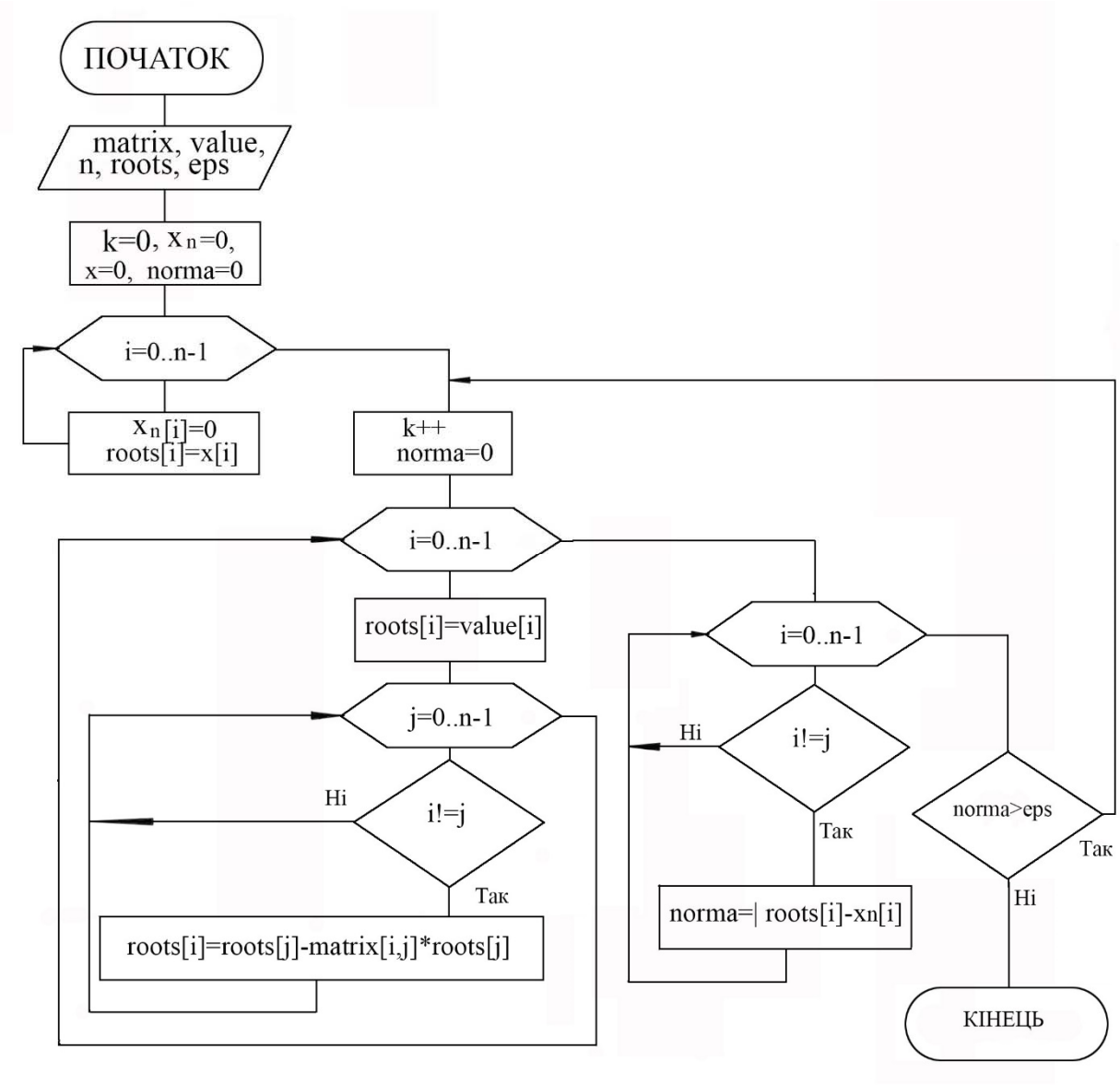


Рис. 5.6. Алгоритм методу релаксації

5.2. Варіанти виконання лабораторної роботи

- Ознайомитись з відомими точними чисельними методами розв'язання систем лінійних алгебраїчних рівнянь (СЛАР).

Скласти схему алгоритму розв’язання систем лінійних алгебраїчних рівнянь методом відповідно до варіанту.

- Скласти програму розв'язання систем лінійних алгебраїчних рівнянь алгоритмічною мовою, узгодженою з викладачем.
- Користуючись даними відповідного варіанту (табл. 5.1), розв’язати СЛАР на ПК (початкові дані та результати роздрукувати).
- Оформити звіт.

Звіт по лабораторній роботі повинен містити:

1) файл тексту програми;

2) опис алгоритму розрахунку (в текстовій формі та у вигляді блок-схеми) в електронному та роздрукованому вигляді;

3) роздруківку файлів з коментарями;

4) загальні висновки за результатами роботи, що включають результати тестування, отримані оцінки похибки результатів і обґрунтування цих оцінок.

Варіанти лабораторної роботи приведені в таблиці 5.1.

Таблиця 5.1

Варіанти завдань до лабораторної роботи №5

Номер варіанту	Матриця коефіцієнтів системи	Стовпець вільних членів	Примітка
1 Метод Гауса з послідовним виключенням невідомих	$1 \quad 1 \quad 2$ $2 \quad -1 \quad 2$ $4 \quad 1 \quad 4$	-1 -4 -2	$x_1 = 1$ $x_2 = 2$ $x_3 = -2$
2 Метод Гауса з одиничними коефіцієнтами	$1 \quad -3 \quad 2$ $3 \quad -4 \quad 0$ $2 \quad -5 \quad 3$	1 2 2	$x_1 = 2$ $x_2 = 1$ $x_3 = 1$
3 Метод послідовних наближень (метод Якобі)	$1 \quad -3 \quad 2$ $3 \quad -4 \quad 0$ $2 \quad -5 \quad 3$	5 7 9	$x_1 = 5$ $x_2 = 2$ $x_3 = 3$

4 Метод Гауса-Зейделя	$\begin{matrix} 1 & 1 & 1 \\ 2 & 3 & 1 \\ 1 & -1 & -1 \end{matrix}$	4 9 -2	$x_1 = 1$ $x_2 = 2$ $x_3 = 1$
5 Метод верхньої релаксації	$\begin{matrix} 1 & -3 & 2 \\ 3 & -4 & 0 \\ 2 & -5 & 3 \end{matrix}$	-5 -2 -7	$x_1 = 6$ $x_2 = 5$ $x_3 = 2$
6 Метод Гауса з вибором головного елемента	$\begin{matrix} 0,63 & 1,00 & 0,71 & 0,34 \\ 1,17 & 0,18 & -0,65 & 0,71 \\ 2,71 & -0,75 & 1,17 & -2,35 \\ 3,58 & 0,21 & -3,45 & -1,18 \end{matrix}$	2,08 0,17 1,28 0,05	$x_1 = 0.4026$ $x_2 = 1.5016$ $x_3 = 0.5862$ $x_4 = -0.2678$
7 Метод Гауса з послідовним виключенням невідомих	$\begin{matrix} 7,09 & 1,17 & -2,23 \\ 0,43 & 1,40 & -0,62 \\ 3,21 & -4,25 & 2,13 \end{matrix}$	-4,75 -1,05 -5,06	$x_1 = 0.2386$ $x_2 = 0.5945$ $x_3 = 3.2019$
8 Метод Гауса з одиничними коефіцієнтами	$\begin{matrix} 1,84 & 2,25 & 2,58 \\ 2,32 & 2,00 & 2,82 \\ 1,83 & 2,06 & 2,24 \end{matrix}$	-6,09 -6,96 -5,52	$x_1 = -0.996$ $x_2 = -0.009$ $x_3 = -1.642$
9 Метод Якобі	$\begin{matrix} 2,36 & 2,37 & 2,13 \\ 2,51 & 2,40 & 2,10 \\ 2,59 & 2,41 & 2,06 \end{matrix}$	1,48 1,92 2,16	$x_1 = 3.864$ $x_2 = -3.899$ $x_3 = 0.753$
10 Метод Гауса-Зейделя	$\begin{matrix} 6,1 & 0,7 & -0,05 \\ -1,3 & -2,05 & 0,87 \\ 2,5 & -3,12 & -5,03 \end{matrix}$	6,97 0,10 2,04	$x_1 = 1.22$ $x_2 = -0.67$ $x_3 = 0.35$
11 Метод Гауса з послідовним виключенням невідомих	$\begin{matrix} 8,7 & -3,1 & 1,8 & 2,2 \\ 2,1 & 6,7 & -2,2 & 0 \\ 3,2 & -1,8 & -9,5 & -1,9 \\ 1,2 & 2,8 & -1,4 & -9,9 \end{matrix}$	-9,7 13,1 6,9 25,1	$x_1 = -0.72$ $x_2 = 1.88$ $x_3 = -0.92$ $x_4 = -1.94$
12 Метод Гауса з вибором головного елемента	$\begin{matrix} 2,58 & 2,98 & 3,13 \\ 1,32 & 1,55 & 1,58 \\ 2,09 & 2,25 & 2,84 \end{matrix}$	-6,66 -3,58 -5,01	$x_1 = 14.923$ $x_2 = -10.526$ $x_3 = -4.406$

13 Метод верхньої релаксації	$1,54 \quad 1,70 \quad 1,62$ $3,69 \quad 3,73 \quad 3,59$ $2,45 \quad 2,43 \quad 2,25$	-1,97 -3,69 -5,98	$x_1 = -2.423$ $x_2 = -36.144$ $x_3 = 39.017$
14 Метод Гауса з одиничними коефіцієнтами	$7,6 \quad 0,5 \quad 2,4$ $2,2 \quad 9,1 \quad 4,4$ $-1,3 \quad 0,2 \quad 5,8$	1,9 9,7 -1,4	$x_1 = 0.248$ $x_2 = 1.114$ $x_3 = -0.224$
15 Метод Якобі	$8 \quad 1 \quad 1$ $1 \quad 5 \quad -1$ $1 \quad -1 \quad 5$	26 7 7	$x_1 = 3$ $x_2 = 1$ $x_3 = 1$
16 Метод Гауса-Зейделя	$2 \quad 1 \quad 4$ $2 \quad -1 \quad -3$ $3 \quad 4 \quad 5$	7 -5 -14	$x_1 = 0$ $x_2 = -1$ $x_3 = 2$
17 Метод верхньої релаксації	$11 \quad 3 \quad -1$ $2 \quad 5 \quad -5$ $1 \quad 1 \quad 1$	15 -11 1	$x_1 = 2$ $x_2 = -2$ $x_3 = 1$
18 Метод Гауса з вибором головного елемента	$1 \quad -4 \quad 0 \quad -1$ $1 \quad 1 \quad 2 \quad 3$ $2 \quad 3 \quad -1 \quad -1$ $1 \quad 2 \quad 3 \quad -1$	6 -1 -1 3	$x_1 = 1$ $x_2 = -1$ $x_3 = 1$ $x_4 = -1$
19 Метод Гауса з послідовним виключенням невідомих	$2 \quad 0 \quad -1 \quad -2$ $0 \quad 1 \quad 2 \quad -1$ $1 \quad -1 \quad 0 \quad -1$ $-1 \quad 3 \quad 2 \quad 0$	-8 -1 -6 7	$x_1 = -1$ $x_2 = 2$ $x_3 = 0$ $x_4 = 3$
20 Метод Гауса з одиничними коефіцієнтами	$1,14 \quad -2,15 \quad -5,11$ $0,42 \quad -1,13 \quad 7,05$ $-0,71 \quad 0,81 \quad -0,02$	2,05 0,80 -1,07	$x_1 = 1.12$ $x_2 = -0.341$ $x_3 = -0.008$
21 Метод Якобі	$0,61 \quad 0,71 \quad -0,05$ $-1,03 \quad -2,05 \quad 0,87$ $2,5 \quad -3,12 \quad 5,03$	-0,16 0,50 0,95	$x_1 = 0.008$ $x_2 = -0.231$ $x_3 = 0.042$
22 Метод Гауса-Зейделя	$3 \quad 2 \quad -5$ $2 \quad -1 \quad 3$ $1 \quad 2 \quad -1$	-1 13 9	$x_1 = 3,$ $x_2 = 5,$ $x_3 = 4$

23 Метод верхньої релаксації	$\begin{matrix} 4 & 2 & -1 \\ 5 & 3 & -2 \\ 3 & 2 & -3 \end{matrix}$	1 2 0	$x_1 = -1,$ $x_2 = 3,$ $x_3 = 1$
24 Метод Гауса з вибором головного елемента	$\begin{matrix} 8 & 7 & 3 \\ -7 & -4 & -4 \\ 6 & 5 & -4 \end{matrix}$	18 11 15	$x_1 = -8.083$ $x_2 = 12.112$ $x_3 = -0.725$
25 Метод Гауса з послідовним виключенням невідомих	$\begin{matrix} 2 & 5 & 4 & 1 \\ 1 & 3 & 2 & 1 \\ 2 & 10 & 9 & 7 \\ 3 & 8 & 9 & 2 \end{matrix}$	20 11 40 37	$x_1 = 1$ $x_2 = 2$ $x_3 = 2$ $x_4 = 0$
26 Метод Гауса з одиничними коефіцієнтами	$\begin{matrix} 1 & 3 & -5 \\ -3 & -12 & 13 \\ 2 & 10 & -3 \end{matrix}$	27 -82 38	$x_1 = -2,$ $x_2 = 3,$ $x_3 = -4$
27 Метод Якобі	$\begin{matrix} 2 & 1 & 4 \\ 2 & -1 & -3 \\ 3 & 4 & -5 \end{matrix}$	20 3 -8	$x_1 = 5,$ $x_2 = -2,$ $x_3 = 3$
28 Метод Гауса-Зейделя	$\begin{matrix} 3 & 3 & 1 & 1 \\ 1 & -1 & 4 & -1 \\ -2 & -2 & -3 & 1 \\ 1 & 5 & -1 & 2 \end{matrix}$	-2 -1 9 4	$x_1 = -3,$ $x_2 = -1,$ $x_3 = 2,$ $x_4 = 7$
29 Метод верхньої релаксації	$\begin{matrix} 0.12 & 0.18 & -0.17 \\ 0.06 & 0.09 & 0.15 \\ 0.22 & 0.1 & 0.06 \end{matrix}$	5.5 -1.95 0.5	$x_1 = 10,$ $x_2 = 5,$ $x_3 = -20$
30 Метод Гауса з вибором головного елемента	$\begin{matrix} 1 & 1 & 1 \\ 0.6 & -0.3 & -0.1 \\ 0 & 0.3 & -0.1 \end{matrix}$	150 28.4 6.2	$x_1 = 72,$ $x_2 = 35,$ $x_3 = 43$

5.3. Контрольні запитання

- Дати визначення термінів: розв’язання СЛАР, визначена і невизначена, вироджена, сумісна і несумісна СЛАР.
- Класифікація методів розв’язання СЛАР.

- Суть точних методів розв’язання СЛАР.
- У яких випадках краще використовувати точні методи?
- Суть алгоритму методу Гауса розв’язання СЛАР.
- У чому різниця між алгоритмами методів Гауса з послідовним виключенням невідомих і з вибиранням головного елементу; що спільного?
- У чому суть методу Гауса за схемою Халецького?
- Порівняйте алгоритми методів Гауса з вибиранням головного елементу і за схемою Халецького. Обчислення за яким з цих алгоритмів швидше? Який із алгоритмів ефективніший?
- В чому суть методу Гауса за схемою Халецького?
- Яку систему отримано в результаті прямого ходу методу Гауса-Жордана?
- Дати визначення термінів: розв’язання СЛАР, норма матриці, норма вектора, достатня умова збіжності.
- Привести СЛАР:

$$\begin{aligned} 9{,}9x_1 - 1{,}5x_2 + 2{,}6x_3 &= 0; \\ 0{,}4x_1 + 13{,}6x_2 - 4{,}2x_3 &= 8{,}2; \\ 0{,}7x_1 + 0{,}4x_2 + 7{,}1x_3 &= -13; \end{aligned}$$

  до нормального вигляду.

5.4. Інформаційні ресурси

1. Метод Гауса для Розв’язання Систем Лінійних Рівнянь: Повний Огляд : https://www.mathros.net.ua/metod-gaussa-rozvjazok-systemy-linijnyh-rivnjan-metodom-gaussa.html
2. Чисельні методи розв’язання систем лінійних алгебраїчних рівнянь: https://web.posibnyky.vntu.edu.ua/fksa/14moskvina__komp_metod_dosl_analiz_danykh/lek2.htm
3. Методи розв’язування алгебраїчних рівнянь і систем алгебраїчних рівнянь: http://amc.ptngu.com/rozdil3.html
4. Чисельне розв’язання систем лінійних алгебраїчних рівнянь: https://ee.kpi.ua/~vk/calcmat/lab/slau_ua.pdf

Перелік використаних джерел

1. Новотарський М.А. Алгоритми та методи обчислень: навч. посіб. – Електронні текстові дані (1 файл) . – Київ: КПІ ім. Ігоря Сікорського, 2019. – 407 с.
2. Креневич А.П. Алгоритми і структури даних. Підручник. – ВПЦ «Київський Університет», 2021. – 200 с.
3. Коробань О.В. Чисельні методи в інформатиці: навч. посіб. / укладач Корбань О.В. – Умань: Візаві, 2016. – 129 с.
4. Волонтир Л.О., Зелінська Л. В., Потапова Н. А. Чіков І. А. Чисельні методи: навч. посіб. – Вінницький національний аграрний університет. – Вінниця: ВНАУ, 2020 – 322 с.
5. Олійник О. П. Системи лінійних алгебраїчних рівнянь: навч.-метод. посіб. – К: НАУ, 2021. – 84 с.
6. Числові моделі та алгоритми. Лабораторний практикум [Електронний ресурс]: навч. посіб. / КПІ ім. Ігоря Сікорського; уклад. : І. В. Стьопочкіна, І.М. Терещенко, К.І. Ільїн. – Електронні тестові дані (1 файл) . – Київ: КПІ ім. Ігоря Сікорського, 2022 . – 46 с.
7. Костюшко І.А. Методи обчислень: / І.А. Костюшко, Н.Д. Любашенко, В.В. Третиник. – Київ: КПІ ім. Ігоря Сікорського, Вид-во «Політехніка», 2021. – 243 с.
8. Методи обчислень: Комп'ютерний практикум [Електронний ресурс]: навч. посіб. / КПІ ім. Ігоря Сікорського; уклад. І.В. Стьопочкіна. – Електронні текстові дані (1 файл) . – Київ: КПІ ім. Ігоря Сікорського, 2022. – 30 с.
9. Гаврилюк І.П., Макаров В.Л. Методи обчислень: Підручник: у 2 ч. – К.: Вища школа, 1995. – Ч. 1. – 367 с.
10. Гаврилюк І.П., Макаров В.Л. Методи обчислень: Підручник: у 2 ч. – К.: Вища школа, 1995. – Ч. 2. – 431 с.
11. Ляшенко Б.М., Кривонос О.М., Вакалюк Т.А. Методи обчислень: навч.-метод. посіб. – Житомир: Вид-во ЖДУ, 2014. – 228 с.

12. Методи та алгоритми комп’ютерних обчислень. Теорія і практика: підручник / Р. Н. Квєтний, Я. В. Іванчук, І. В. Богач, О. Ю. Софина, М. В. Барабан. – Вінниця : ВНТУ, 2023. – 280 с.
13. Самборська О.М., Шелестовський Б.Г. Чисельні методи. Навчальний посібник для студентів вищих технічних навчальних закладів. – Тернопіль:ТНТУ імені Івана Пулюя, 2010. – 164с.
14. Задачин В. М. Чисельні методи : навчальний посібник / В. М. Задачин, І. Г. Конюшенко. – Х. : Вид. ХНЕУ ім. С. Кузнеця, 2014. – 180 с.
15. Матюшенко М. В. Методи наближених обчислень [Електронний ресурс] : навч. посіб. / М. В. Матюшенко, Є. М. Сівак ; Нац. техн. ун-т "Харків. політехн. ін-т". – Електрон. текст. дані. – Харків, 2022. – 92 с.