МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Івано-Франківський національний технічний університет нафти і газу

Кафедра комп'ютерних систем та мереж

М. І. Горбійчук

АЛГОРИТМИ ТА МЕТОДИ ОБЧИСЛЕНЬ

КОНСПЕКТ ЛЕКЦІЙ

Івано-Франківськ
2013

УДК 681.3
ББК 519.85
Г 67

Рецензент:
Юрчишин В. М. доктор техн. наук, професор кафедри програмного забезпечення автоматизованих систем

Рекомендовано методичною радою університету
(протокол № 5 від 28 травня 2013 р.)

Горбійчук М. І.
Г 67 Алгоритми і методи обчислень: конспект лекцій / М.І.Горбійчук. – Івано-Франківськ: ІФНТУНГ, 2013. – 270 с.

МВ -02070855 - 5005 - 2013

Конспект лекцій складено відповідно до робочої програми дисципліни «Алгоритми та методи обчислень». Конспект лекцій складається із двох частин. У першій частині висвітлені такі питання як побудова та властивості алгоритмів, зокрема, розглянуті стратегії «розділяй та володарюй», «жадібні алгоритми» та динамічне програмування. У другій частині, яка присвячена числовим методам, знайшли своє відображення задачі лінійної та нелінійної алгебри, методи розв'язку диференціальних та інтегральних рівнянь, задачі математичної фізики, методи наближення функцій та методи оптимізації. Кожний розділ конспекту лекцій закінчується контрольними питаннями та завданнями, які служать для самоперевірки і самостійної роботи студентів.

Призначено для підготовки бакалаврів за напрямом 6.050102 «Комп'ютерна інженерія».

МВ -02070855 - 5005 - 2013

УДК 681.3
ББК 519.85

ЗМІСТ

ВСТУП

За останні кілька десятиріч досягнуті вражаючі успіхи у розвитку електроніки. Це дало змогу значно збільшити об'єм пам'яті комп'ютерів та їх швидкодію.

На початку 80-х років архітектура комп'ютера у значній мірі обмежувала об'єм пам'яті і швидкість виконання машинних операцій. Переважно об'єм пам'яті тогочасних комп'ютерів не перевищував 64 К. У сучасних комп'ютерів ця величина збільшилась у 1000 раз.

Зі збільшенням ємності кристалів (чипів) підвищується і тактова частота роботи процесора. Якщо у 1980-х роках вона не перевищувала 50 МГц, то сучасні комп'ютери працюють на частоті до 3 ГГц. Щороку тактова частота зростає приблизно на 30 %.

Узагальнюючою характеристикою комп'ютера є його продуктивність, яка враховує як ріст тактової частоти, так і вдосконалення його архітектури. Починаючи з середини 80-х років продуктивність комп'ютера щороку зростає приблизно на 50 %.

Незважаючи на такі вражаючі успіхи у розвитку як апаратних засобів комп'ютерів, так і відповідного програмного забезпечення, актуальною проблемою залишається розробка ефективних алгоритмів для вирішення як наукових, так і інженерних задач.

У основі багатьох мов програмування лежать теорія алгоритмів, теорія формальних систем, логіка предикатів. Крім того синтез логіки і комп'ютерів привів до виникнення баз даних і експертних систем, що є важливим етапом на шляху до створення штучного інтелекту — машинної моделі людського розуму.

Кожен алгоритм передбачає наявність деяких початкових або апріорних даних, у результаті застосування яких отримують певний шуканий результат. Далі,

застосування кожного алгоритму здійснюється шляхом виконання дискретної послідовності деяких елементарних дій. Ці дії називають кроками, а процес їх виконання називають алгоритмічним процесом.

Таким чином, однією із основних властивостей алгоритму є його дискретність. Іншою істотною рисою алгоритму є його масовий характер, тобто можливість застосування його до широкого класу початкових даних з можливістю варіювання такими початковими даними. Іншими словами, кожен алгоритм покликаний вирішити ту або іншу масову проблему, тобто вирішувати клас однотипних завдань. Неодмінною умовою, якій задовольняє алгоритм є його детермінованість, або визначеність. Це означає, що настанови алгоритму з рівним успіхом можуть бути виконані будь-якою іншою людиною і в будь-який інший час причому результат вийде той же самий. Тобто настанови алгоритму настільки точні і однозначні, що не допускають жодних двозначних тлумачень і жодного свавілля з боку виконавця. Вони єдиним і цілком визначеним шляхом кожний раз приводять до шуканого результату. Це наводить на думку, що виконання тих або інших алгоритмів може бути доручено машині, що широко і робиться на практиці. Кажучи про початкові дані для алгоритму, мають на увазі так звані допустимі початкові дані, тобто такі початкові дані, які сформульовані в термінах даного алгоритму. Серед допустимих початкових даних алгоритму можуть бути такі, до яких він пристосований, тобто відштовхуючих від яких можна отримати шуканий результат, а можуть бути і такі, до яких даний алгоритм непридатний, тобто використання яких не дає шуканого результату. Непридатність алгоритму до допустимих початкових даних може полягати або в тому, що алгоритмічний процес ніколи не закінчиться (у цьому випадку говорять, що він безконечний), або в тому, що його виконання під час одного з кроків потрапляє на перешкоду, заходить у безвихідь (в цьому випадку кажуть, що він безрезультатно обривається).

В основі значного числа алгоритмів, які орієнтовані на розв'язок прикладних проблем, лежать числові методи.

Числові методи застосовують у тих випадках, коли певну математичну задачу неможливо або важко розв'язати аналітичним способом. Як приклади таких задач можна навести розв'язок лінійних і нелінійних алгебраїчних рівнянь, звичайних диференціальних рівнянь і рівнянь з частковими похідними, задачі оптимізації та ін.

У цілому ряду випадків предметом вивчення числових методів є процес заміни початкової задачі іншою задачею, яка легко алгоритмізується. Такий процес заміни носить назву апроксимації початкової задачі. Зрозуміло, що у процесі розв'язку задачі числовим методом повинна бути оцінена точність такої апроксимації.

Навчальний посібник «Алгоритми і методи обчислень» написано відповідно до програми однойменного курсу, і він складається з двох частин.

У першій частині (розділи 1 - 3) знайшли своє висвітлення такі питання як аналіз алгоритмів (θ-означення; O-, Ω-означення; o- і ω- означення. Стандартні функції і означення: монотонність, цілі наближення знизу і зверху. Оцінка ефективності алгоритму: аналіз часу виконання в середньому) та алгоритмічні стратегії (метод «розділяй і володарюй»: загальна характеристика, приклад застосування; «жадібні» алгоритми: доцільність використання «жадібних» алгоритмів, приклади застосування; динамічне програмування: доцільність використання динамічного програмування, приклади застосування; алгоритми на графах: означення, алгоритм пошуку у ширину. пошук найкоротшого шляху).

Друга частина (розділи 4 – 10) присвячена висвітленню питань розв'язку числовими методами таких задач як задачі лінійної алгебри (задачі лінійної та нелінійної алгебри; методи розв'язку звичайних диференціальних рівнянь і диференціальних рівнянь з частковими похідними; методи оптимізації).

Матеріал книги проілюстровано численними прикладами, що посприяє кращому засвоєнню основних понять з теорії алгоритмів та числових методів розв'язку типових інженерних задач.

АЛГОРИТМИ

1 АНАЛІЗ АЛГОРИТМІВ

1.1 Поняття алгоритму

Процес створення комп'ютерної програми для розв'язку певної інженерної задачі складається із таких етапів:

- формалізація сформованої задачі;
- розроблення алгоритму розв'язку задачі;
- написання налагоджування і тестування програми;
- отримання результату розв'язку задачі на ЕОМ.

Центральне місце у цьому процесі займає розроблення алгоритму розв'язку технічної задачі.

Алгоритм – це формальний опис обчислювальної процедури, яка отримує початкові дані, які ще називають *входом* алгоритму, і видає результати обчислень на *виході*.

Алгоритм вважають *правильним*, якщо за кінцевий проміжок часу він закінчує роботу і видає очікуваний результат. У такому випадку кажуть, що алгоритм розв'язує дану обчислювальну задачу.

Таким чином, програма, яка написана на основі розробленого алгоритму, при будь-яких початкових даних не повинна здійснювати нескінченні циклічні обчислення.

1.2 Задача сортування даних

У багатьох алгоритмах сортування використовується як проміжний крок.

Задача сортування (сортування вставками) – це упорядкування послідовності записів таким чином, щоб значення виходу алгоритму складали незгасаючу послідовність. Вона записується наступним чином:

Вхід: послідовність n чисел $(a_1, a_2, ..., a_n)$.

Вихід: послідовність упорядкованих чисел $(a'_1, a'_2, ..., a'_n)$, для якої $a'_1 \le a'_2 \le ... \le a'_n$.

Для запису алгоритму будемо використовувати псевдокод, у якому оператори наближені до алгоритмічної мови. Процедуру сортування даних назвемо – sort.

sort

```
1 n=length(A)
2 for j=2 to n
3     key=A(j)
4     i=j-1
5 while i>0 and A(i)>key
6     A(i+1)=A(i)
7     i=i-1
8     if i=0 then break
9 end while
10    A(i+1)=key
12 end for
```

У процедурі sort прийняті такі позначення:

- оператор `length` визначає розмірність масиву `A`;

- оператори, які розміщені між `for` і `end for` утворюють тіло циклу;

- `while` оператор циклу, виконання якого визначається певною умовою; у процедурі sort це умови `i>0` та `A(i)>key`, які зв'язані логічним оператором `and`;

- у рядку 8 використаний умовний оператор `if-then-else`: якщо змінна `i` приймає значення нуль, то відбувається вихід із циклу `while-end while` за допомогою оператора `break` (відмітимо, що рядок 8 не є обов'язковим у тому випадку, коли певна алгоритмічна мова допускає нульовий індекс масиву);

- індекс масиву пишеться у круглих дужках. Наприклад, `A(i)` є `i`-тий елемент у масиві `A`.

Результатом роботи алгоритму sort буде масив, елементи якого будуть розміщені у зростаючому порядку. Наприклад, якщо початковий масив - $A = [12,8,2,9,13,17,6,15,3]$, то в результаті роботи алгоритму отримаємо упорядковану послідовність $A = [2,3,6,8,9,12,13,15,17]$.

1.3 Приклад аналізу алгоритму

Аналіз алгоритму проводиться з метою визначення певних оцінок, які характеризують ефективність розробленого алгоритму. Оцінки алгоритмів включають у себе такі суб'єктивні поняття як простота, зрозумілість або відповідність результату роботи очікуваним даним. Об'єктивнішою (можливо не найважливішою) оцінкою алгоритму може служити *час роботи алгоритму*.

Одним із способів визначення часу роботи алгоритму полягає у тому, що на основі розробленого алгоритму можна написати програму і виміряти час її виконання на певній ЕОМ для вибраної множини вхідних даних. Хоча такий спосіб є простим і зрозумілим він породжує певні проблеми. Визначений у такий спосіб час роботи алгоритму залежить не тільки від певного алгоритму, але й від архітектури і набору внутрішніх інструкцій даного комп'ютера, від властивостей компілятора, а також від рівня кваліфікації програміста, який реалізує тестовий алгоритм. Час роботи алгоритму може також залежати і від вибраної множини тестових даних. Ця залежність стає очевидною за реалізації одного і того ж алгоритму з використанням різних комп'ютерів, різних компіляторів або із залученням програмістів різного рівня кваліфікації і при використанні різних тестових даних. Тому для вироблення об'єктивної оцінки часу роботи алгоритму треба домовитися про те, яку модель обчислень використовуються. Ми будемо розглядати однопроцесорну

ЕОМ з довільним доступом, яка не передбачає паралельного виконання обчислювальних операцій.

Таким чином, *час роботи алгоритму* будемо визначати числом елементарних кроків, які необхідно виконати, щоб отримати бажаний результат. Питання лише у тому, що вважати елементарними кроками. Будемо вважати, що на виконання одного рядка псевдокоду затрачується певне число фіксованих операцій. Слід розрізняти *виклик* процедури, на яке витрачається певне фіксоване число операцій і її *виконання*, яке може бути досить довгим.

Як приклад визначення часу роботи алгоритму розглянемо процедуру sort.

Біля кожного рядка будемо записувати його вартість і число повторних виконань цього рядка. Для кожного j від 2 до n підрахуємо скільки разів буде виконуватись рядок 5 і позначимо це число через t_j. Відмітимо, що рядки всередині циклу `while-end while` виконуються на один раз менше, ніж сама умова `while i>0 and A(i)>key`, оскільки остання перевірка виводить із циклу.

sort		*вартість*	*число раз*
1	`n=length(A)`	c_1	1
2	`for j=2 to n`	c_2	n
3	`key=A(j)`	c_3	$n-1$
4	`i=j-1`	c_4	$n-1$
5	`while i>0 and A(i)>key`	c_5	$\sum_{j=2}^{n} t_j$
6	`A(i+1)=A(i)`	c_6	$\sum_{j=2}^{n}(t_j-1)$
7	`i=i-1`	c_7	$\sum_{j=2}^{n}(t_j-1)$
8	`if i=0 then break`	c_8	$\sum_{j=2}^{n}(t_j-1)$
9	`end while`		
10	`A(i+1)=key`	c_9	$n-1$
11	`end for`		

Якщо вартість рядка c і він повторяється m раз, то вклад цього рядка у загальне число операцій – cm. Додавши, вклади всіх лінійок отримаємо:

$$T(n)=c_1+c_2n+c_3(n-1)+c_4(n-1)+c_5\sum_{j=2}^{n}t_j+c_6\sum_{j=2}^{n}(t_j-1)+c_7\sum_{j=2}^{n}(t_j-1)+c_8\sum_{j=2}^{n}(t_j-1)+c_9(n-1)$$

Час роботи алгоритму **sort** залежить не тільки від n, але й від того який саме масив розміру n поданий на вхід алгоритму. Для алгоритму sort найбільш прийнятним є випадок коли початковий масив A уже відсортований. Тоді цикл у рядку 5 завершиться зразу ж після першої перевірки, так що всі значення t_j дорівнюють 1, і загальний час роботи алгоритму буде таким:

$$T(n)=(c_2+c_3+c_4+c_5+c_9)n-(c_3+c_4+c_7+c_9-c_1).$$

Таким чином в найсприятливішому випадку час роботи алгоритму $T(n)$, яке необхідне для упорядкування масиву A, є *лінійною функцією* від n, тобто має вигляд

$$T(n)=an+b,$$

де коефіцієнти a і b визначаються вибраними значеннями $c_1, c_2, \ldots, c_9$.

У тому випадку, коли елементи масиву A розміщені в протилежному напрямку (у спадаючому), то час роботи

алгоритму буде максимальним: кожний елемент $A(j)$ необхідно порівнювати із всіма елементами $A(1)$, $A(2)$, …, $A(j-1)$. При цьому $t_j = j$. Обчислимо суму $\sum_{j=2}^{n} j$, яка є арифметичною прогресією. Сума N членів арифметичної прогресії обчислюється за відомою формулою:

$$s_N = \frac{(a_1 + a_N)N}{2}.$$

У нашому випадку – $a_1 = 2$, $a_N = n$, $N = n-1$ і $\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$. Аналогічно знаходимо, що $\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$.

Таким чином, у найгіршому випадку час роботи алгоритму sort визначимо за формулою:

$$T(n) = c_1 + c_2 n + c_3(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + c_6\frac{n(n-1)}{2} + c_7\frac{n(n-1)}{2} + c_8\frac{n(n-1)}{2} + c_9(n-1)$$

Після розкриття дужок і нескладних алгебраїчних перетворень, отримаємо:

$$T(n) = \frac{n^2}{2}(c_5 + c_6 + c_7 + c_8) + \left(c_2 + c_3 + c_4 + \frac{1}{2}(c_5 - c_6 - c_7 - c_8)\right)n - (c_3 + c_4 + c_5 + c_9 - c_1)$$

.

Тепер час роботи алгоритму $T(n)$ має вигляд *квадратичної функції*:

$$T(n) = an^2 + bn + c, \qquad (1.1)$$

де $a = \frac{1}{2}(c_5 + c_6 + c_7 + c_8)$; $b = c_2 + c_3 + c_4 + \frac{1}{2}(c_5 - c_6 - c_7 - c_8)$; $c = -(c_3 + c_4 + c_5 + c_9 - c_1)$.

Бачимо, що в найкращому і найгіршому випадках час роботи алгоритму відрізняється досить значно. Як правило, визначають час роботи алгоритму у найгіршому випадку. Це зумовлено тим, що знаючи час роботи алгоритму у найгіршому випадку, можна гарантувати, що обчислення закінчиться за певний фіксований час. Як показує практика, середній час алгоритму часто є досить близьким до часу $T(n)$ у найгіршому випадку.

1.4 Основні параметри, що характеризують роботу алгоритму

Для оцінки ефективності роботи алгоритму необхідно мати певні критерії. Одним із таких критеріїв – час роботи алгоритму, але він не єдиний. Іншою важливою характеристикою алгоритму є *асимптотична складність алгоритму*. Як раз асимптотична складність алгоритму визначає у підсумку розмір задачі, яку можна розв'язати за допомогою такого алгоритму. Якщо алгоритм обробляє вхідні дані розміром n за час cn^2, де c - деяка постійна величина, то *складність* такого алгоритму буде $O(n^2)$ (читається «порядку n^2).

Можна подумати, що ріст швидкості обчислень знімає проблему розробки ефективних алгоритмів. Насправді маємо протилежну ситуацію: саме складність алгоритму визначає то збільшення розміру задачі, яке можна досягти зі збільшенням швидкодії ЕОМ.

Допустимо, що у нашому розпорядженні є п'ять алгоритмів $A_1 - A_5$ з наступними складностями:

Алгоритм	*Складність алгоритму*
A_1	n
A_2	$n\log_2 n$
A_3	n^2
A_4	n^3
A_5	2^n

Складність алгоритму – це кількість кроків, які необхідно виконати, щоб досягти заданого результату. Часова складність алгоритму - це затрати часу T_a на обробку входу розміром n. Допустимо, що затрати часу на виконання однієї операції - $c = 10^{-3}$ c. Тоді асимптотична складність алгоритму - $T(n) = \frac{T_a}{c}$. При часовій складності T_a алгоритм A_1 може обробити масив розміру, який визначиться за формулою:

$$n = \frac{T_a}{c}.$$

Для $T_a = 1$ c і $c = 10^{-3} c$ розмір вхідного масиву $n = 1000$, а для алгоритму A_5 $n = \log_2\left(\frac{T_a}{c}\right)$ і відповідно $n = 9.966$, тобто алгоритм A_5 за одну секунду може обробляти масив розмірністю не більше десяти.

Допустимо, що швидкодія ЕОМ збільшилась у десять раз. Тоді $c = 10^{-4}$. У табл. 1.1 показано як виростуть розміри задач, які можна розв'язати за одну секунду одну хвилину і за одну годину.

Таблиця 1.1 – Границі розмірів задач, що визначаються швидкістю росту складності

Алгоритм	*Часова складність*	*Максимальний розмір задачі*		
		1 *с*	1 *хв.*	1 *год.*
A_1	n	10^4	$6\cdot10^5$	$3{,}6\cdot10^7$
A_2	$n\log_2 n$	$1{,}003\cdot10^3$	$3{,}9312\cdot10^4$	$1{,}7368\cdot10^6$
A_3	n^2	100	775	$6\cdot10^3$
A_4	n^3	22	85	330
A_5	2^n	14	20	26

Замість ефекту збільшення швидкості обчислень розглянемо ефект від вибору дієвішого алгоритму.

Якщо за базу порівняння вибрати 1 *хв.*, то, замінивши A_4 на A_2 можна розв'язати задачу у 46 раз швидше. У той же час десятикратне збільшення швидкості обчислень дає виграш у часі для алгоритму A_4 лише у два рази.

Таким чином, ефективні алгоритми дають значно більші виграші у часі, ніж збільшення швидкодії обчислювальних машин.

1.5 Асимптотика росту часу роботи алгоритму

Аналіз алгоритму **sort** дав можливість встановити, що час роботи алгоритмі при постійних значеннях c_i, $i = \overline{1,9}$ може бути обчислений за формулою (1.1). Якщо у виразі (1.1) знехтувати членами bn і c, то отримаємо оцінку

$$T(n)=\Theta(n^2), \qquad (1.2)$$

яка носить назву *порядок росту* часу роботи алгоритму.

Точний зміст формули (1.2) такий: знайдуться такі константи α_1, $\alpha_2>0$ і таке число n_0, що $\alpha_1 n^2 \le T(n) \le \alpha_2 n^2$ при всіх $n \ge n_0$ (рис. 1.1).

У загальному випадку, якщо $g(n)$ деяка функція, то запис $f(n)=\Theta(g(n))$ означає, що знайдуться такі α_1, $\alpha_2>0$ і таке n_0, що $\alpha_1 g(n) \le f(n) \le \alpha_2 g(n)$ для всіх $n \ge n_0$.

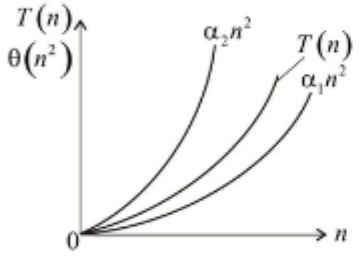


Рисунок 1.1 – Асимптотика росту алгоритму

Визначення $\Theta(g(n))$ допускає, що функції $f(n)$ і $g(n)$ асимптотичні та невід'ємні, $f(n)$, $g(n) \ge 0$ для досить великих n.

Якщо $f(n)=\Theta(g(n))$, то кажуть, що $g(n)$ є асимптотичною точною оцінкою для $f(n)$.

Шукаючи асимптотичну точну оцінку для суми, можна відкидати доданки меншого порядку, які при великих n стають малими у порівнянні з основним доданком. Взагалі, для будь-якого полінома $p(n)$ степені m з додатним коефіцієнтом при старшому члені маємо $p(n)=\Theta(n^m)$.

Відмітимо також, що значення коефіцієнту при старшому члені не відіграє ніякої ролі, а лише може впливати на вибір значень α_1 і α_2.

Приклад 1.1

Знайти додатні константи α_1 і α_2, якщо $T(n)=\frac{1}{2}n^2-3n$.

Згідно визначення необхідно знайти такі константи α_1 і α_2, щоб нерівність

$$\alpha_1 n^2 \le \frac{1}{2}n^2-3n \le \alpha_2 n^2$$

виконувалась для всіх $n \ge n_0$. Розділимо останню нерівність на n^2:

$$\alpha_1 \le \frac{1}{2}-\frac{3}{n} \le \alpha_2 .$$

Остання умова розпадається на дві нерівності $\alpha_1 \le \frac{1}{2}-\frac{3}{n}$ і $\alpha_2 \ge \frac{1}{2}-\frac{3}{n}$. Для виконання другої нерівності достатньо взяти $\alpha_2=\frac{1}{2}$. Перша нерівність буде виконана, якщо $n_0=7$ (рис. 1.2). Тоді $\alpha_1=\frac{1}{14}$.

Запис $f(n)=\Theta(g(n))$ включає у собі дві оцінки: верхню і нижню. Їх можна розділити. Говорять, що $f(n)=O(g(n))$, якщо знайдеться така константа $\alpha>0$ і таке

число n_0, що $0 \le f(n) \le \alpha g(n)$ для всіх $n \ge n_0$ і $f(n) = \Omega(g(n))$, якщо знайдеться така константа $\alpha > 0$ і таке число n_0, що $0 \le \alpha g(n) \le f(n)$ для всіх $n \ge n_0$.

Функція $g(n)$ утворює верхню межу для класу функцій $f(n) = O(g(n))$. Цей клас складається із функцій, які ростуть не швидше ніж $g(n)$.

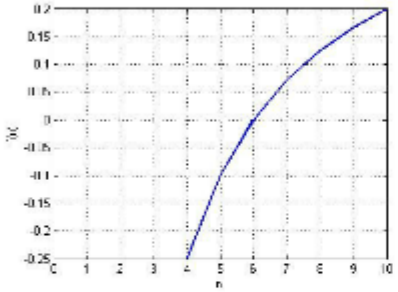


Рисунок 1.2 – Залежність $f(n) = \frac{1}{2} - \frac{3}{n}$

Можна вважати, що клас функцій $\Omega(g(n))$ задає нижню межу: всі функції із неї ростуть по крайній мірі так же швидко як і $g(n)$.

Вираз $f(n) = O(g(n))$ має той зміст, що з ростом n відношення $\frac{f(n)}{g(n)}$ залишається обмеженим. Якщо до того ж

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0,$$

то пишуть $f(n) = o(g(n))$.

Кажучи формально, $f(n) = o(g(n))$, якщо для будь-якого додатного ε знайдеться таке n_0, що $0 \le f(n) \le \varepsilon g(n)$ при всіх $n \ge n_0$.

Аналогічним чином вводиться ω- позначення: говорять, що $f(n)$ є $f(n) = \omega(g(n))$, якщо для будь-якого додатного ε існує таке n_0, що $0 \le \varepsilon g(n) \le f(n)$ для всіх $n \ge n_0$. Якщо має місце $f(n) = \omega(g(n))$, то $g(n) = o(f(n))$.

Контрольні питання та завдання

1. Дайте визначення алгоритму.
2. У чому полягає універсальність алгоритму?
3. Що означає детермінованість алгоритму?
4. Який алгоритм вважають правильним?
5. У чому різниця між задачами сортування вставками і сортування злиттям?
6. Який із двох алгоритмів сортування вставками і сортування злиттям є ефективнішим і чому?
7. Чому час роботи алгоритму визначають у найгіршому випадку?
8. Чому дорівнює порядок складності алгоритму сортування вставками і сортування злиттям?
9. Знайти додатні константи α_1 і α_2, якщо $T(n) = \frac{1}{2} n^2 - 2n$.
10. Що означає вираз $f(n) = \Theta(g(n))$?
11. Яку властивість алгоритму характеризує його асимптотика?
12. Дайте тлумачення таким виразам: $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, $f(n) = o(g(n))$ і $f(n) = \omega(g(n))$.

2 АЛГОРИТМІЧНІ СТРАТЕГІЇ

2.1 Принцип «розділяй і володарюй»

Багато алгоритмів за своєю природою *рекурсивні*, розв'язуючи деяку задачу вони визивають самих себе для розв'язку її підзадач. Ідея методу «розділюй і володарюй» як раз і полягає у цьому. Спочатку задача розбивається на декілька задач меншого розміру. Потім ці задачі розв'язуються за допомогою рекурсивного виклику – або безпосередньо, якщо розмір задачі невеликий. Нарешті, розв'язки часткових задач комбінуються і отримують розв'язок задачі у цілому.

Як приклад, розглянемо задачу знаходження найбільшого (найменшого) елементу множини S, яка вміщує n елементів. Очевидний шлях пошуку найбільшого елементу полягає у тому, щоб шукати його як результат порівняння певного елементу з іншими і запам'ятовувати кожний раз більший із них. Наступна процедура `Max` находить найбільший елемент множини S, зробивши $n-1$ порівнянь:

```
Max
1 n=length(S)
2 ▷Вибираємо довільний елемент із множини S
3 max=S(1)
4 for i=2 to n
5   x=S(i)
6   if x>max
7      then max=x
8   end if
9 end for
```

Аналогічно можна знайти найменший елемент із $n-1$ елементів, що залишитись, здійснивши $n-2$ порівнянь.

Таким чином, для знаходження найбільшого і найменшого елементів із множини S необхідно виконати $2n-3$ порівнянь ($n-1+n-2=2n-3$).

Застосуємо до задачі знаходження найбільшого і найменшого елементів із множини S принцип «розділюй і володарюй». Для цього множину S розіб'ємо на дві підмножини S_1 і S_2. Для простоти допустимо, що у кожній із підмножин є по $\frac{n}{2}$ елементи. Тоді описаний вище алгоритм знайшов би найменший і найбільший елементи у кожній із підмножин S_1 і S_2. Найбільший і найменший елемент множини S можна визначити провівши ще два порівняння найбільших і найменших елементів із множин S_1 і S_2.

Отже, матимемо таку процедуру:

```
procedure Max_Min(S):
1. n=length(S)
2. if n=2 then
      ▷нехай S={a,b}
3.     return (Max(a,b), Min(a,b))
     else
6.     розбити S на дві рівні підмножини S1 і S2
      ▷виклик процедури MaxMin
5.    (max1,min1)= MaxMin(S1)
6.    (max2,min2)= MaxMin(S2)
7.     return (Max(max1,max2), Min(min1,min2))
8    end if
```

Аналізуючи процедуру `Max_Min`, можна зробити висновок, що порівняння елементів множити S відбувається тільки на кроці 3, де порівнюються два елементи множини S, із яких вона складається, і на кроці 7, де порівнюються `max1,max2` та `min1,min2`. Очевидно, що $T(2)=1$ (одне

порівняння). Якщо $n > 2$, то $T(n)$ - загальне число порівнянь отриманих у двох викликах процедури `MaxMin` (рядки 5 і 6), які працюють на множинах розміром $\frac{n}{2}$, і ще два порівняння у рядку 7. Таким чином,

$$T(n) = \begin{cases} 1 \text{ при } n = 2, \\ 2T(n/2) + 2 \text{ при } n > 2. \end{cases} \qquad (2.1)$$

Знайдемо розв'язок рекурентного співвідношення (2.1) Нехай у (2.1) j приймає довільне значення. Тоді

$T(2) = 1$,

$T(j) = 2T\left(\frac{j}{2}\right) + 2$, $j > 2$.

Допустимо, що $j \in \left[\frac{n}{2}, \frac{n}{4}, \frac{n}{8}, \ldots\right]$. Тоді

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + 2,$$

$$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + 2,$$

$$T\left(\frac{n}{8}\right) = 2T\left(\frac{n}{16}\right) + 2,$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots .$$

Шляхом послідовного підставлення знаходимо, що

$$T(n) = 2T\left(\frac{n}{2}\right) + 2 = 2\left(2T\left(\frac{n}{4}\right) + 2\right) + 2 = 4T\left(\frac{n}{4}\right) + 4 + 2,$$

$$T(n) = 4T\left(\frac{n}{4}\right) + 4 + 2 = 4\left(2T\left(\frac{n}{8}\right) + 2\right) + 4 + 2 = 8T\left(\frac{n}{8}\right) + 8 + 4 + 2$$

$$T(n) = 8T\left(\frac{n}{8}\right) + 8 + 4 + 2 = 8\left(2T\left(\frac{n}{16}\right) + 2\right) + 8 + 4 + 2 = 16T\left(\frac{n}{16}\right) + 16 + 8 + 4 + 2,$$

Останній вираз подамо у такому вигляді:

$$T(n) = 2^4 T\left(\frac{n}{2^4}\right) + 2^4 + 2^3 + 2^2 + 2.$$

Узагальнюючи отриманий результат для довільного $k \in$, де - множина натуральних чисел, будемо мати

$$T(n,k) = 2^k T\left(\frac{n}{2^k}\right) + 2^k + 2^{k-1} + \cdots + 2 = 2^k T\left(\frac{n}{2^k}\right) + \sum_{i=1}^{k} 2^i .$$

Допустимо, що $2^k = \frac{n}{2}$. Тоді $k = \log_2 n - 1$. Оскільки $T(2) = 1$, то при $k = \log_2 n - 1$

$$T(n) = 2^{\log_2 n - 1} + \sum_{i=1}^{\log_2 n - 1} 2^i .$$

Перший доданок суми, яка є правою частиною останнього співвідношення - $2^{\log_2 n - 1} = \frac{2^{\log_2 n}}{2} = \frac{n}{2}$, а другий доданок – це геометрична прогресія, в якої перший член $a_1 = 2$; знаменник прогресії - $q = 2$, а кількість її членів дорівнює k. Тому

$$\sum_{i=1}^{k} 2^i = \frac{a_1\left(q^k - 1\right)}{q - 1} = 2^{k+1} - 2.$$

Враховуючи значення k, маємо

$$\sum_{i=1}^{\log_2 n-1} 2^i = 2^{\log_2 n} - 2 = n - 2 .$$

Таким чином, розв'язком рекурентних співвідношень (2.1) є функція

$$T(n) = \frac{3}{2}n - 2$$

за умови, що $n = 2^k$, де k - ціле додатне число.

Рисунок 2.1 дає наочне уявлення про те, що алгоритм, який побудований за стратегією «розділяй і володарюй», потребує меншого числа кроків ніж алгоритм, який здійснює пошук максимального і мінімального елементів шляхом перебору елементів масиву.

Ще одним прикладом застосування стратегії «володарюй і розділяй» може служити алгоритм сортування злиттям. Суть алгоритму у тому, що початкова множина елементів розбивається на підмножини S_1, S_2, …, S_k. Кожна із множин S_i, $i = \overline{1,k}$, у принципі, може вміщувати лише один елемент. У такому разі кожен із списків уже відсортований. Тепер задача полягає у тому, щоб певним чином списки S_1, S_2, …, S_k злити.

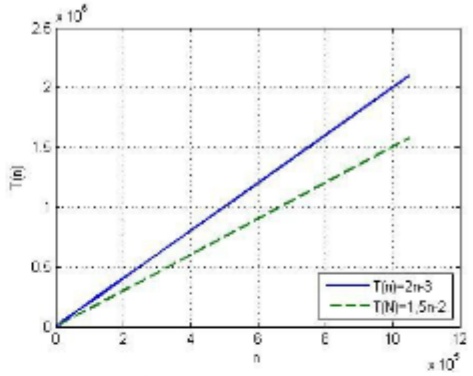


Рисунок 2.1 – Порівняння ефективності двох алгоритмів

Нижче наведена процедура `MergeSort`, яка виконує сортування елементів множини S злиттям.

```
1 procedure MergeSort
▷list список елементів, що підлягає сортуванню
▷first номер першого елементу у списку, що підлягає сортуванню
▷last номер останнього елементу у списку, що підлягає сортуванню
2 while first<last then
3   middle=(first+last)/2
4   MergeSort(list,first,middle)
5   MergeSort(list,middle+1,last)
6   MergeLists(list,first,middle,middle+1,last)
7 end while
```

Наведений алгоритм розбиває список S на дві половини до того часу, поки номер першого елемента куска менше останнього у ньому номера. Якщо у черговому куску указана умова не виконується, то це означає, що отримали список із одного елементу, який уже відсортований. Після двох

викликів процедури `MergeSort` викликається процедура `MergeLists`, яка зливає два куски, у результаті отримаємо злитий список із двох елементів. На наступному кроці два списки, що вміщують по два елементи, зливаються в один довжиною чотири. Цей процес продовжується до того часу, поки дві відсортовані половини загального списку зіллються в один. Таким чином, процедура `MergeSort` розбиває список на половину при русі по рекурсії вниз, а потім на зворотному шляху зливає відсортовані половинки списку.

Операцію зі злиття списків в один виконує процедура `MergeLists`, яку розглянемо детальніше.

Нехай S_1 і S_2 два списки, які вже відсортовані у порядку зростання елементів. При злитті двох списків в один найменший елемент у загальному списку може бути першим або у списку S_1 або – у S_2, а найбільший елемент в об'єднаному списку повинен бути останнім в одному із списків S_1 чи S_2. Створимо новий список P, який буде вміщувати елементи як списку S_1, так і елементи - S_2. Створювати список P почнемо з того, що в нього перенесемо менший із двох елементів $S_1(1)$ і $S_2(1)$. Якщо $S_1(1) < S_2(1)$, наступним елементом $P(2)$ може бути $S_2(1)$ або $S_1(2)$. І перший і другий варіанти можливі оскільки нам невідомі співвідношення між величинами списку S_1 і списку S_2. Для спрощення процесу злиття слід обзавестись двома індикаторами списків S_1 та S_2 і збільшувати той індикатор того списку, черговий елемент у якому виявиться меншим. Процедура `MergeLists` продовжує порівнювати елементи ще не переглянутих частин списків S_1 і S_2 та переміщувати менший із них у список P. У певний момент елементи в одному із списків закінчаться. У другому списку залишаться елементи, які більші останнього елементу у списку P. На закінчення, необхідно перемістити елементи, що залишились у кінець списку P.

```
procedure MergeLists
▷list список елементів, що підлягає упорядкуванню
▷start1 початок списку S1
▷end1 кінець списку S1
▷start2 початок списку S2
▷end2 кінець списку S2

finalStart=start1
finalEnd=end2
indexP=1
while (start1<=end1) and (start2<=end2)
    if list(start1)<list(start2)
       result(indexP)=list(start1)
       start1=start1+1
    else
       result(indexP)=list(start2)
       start2=start2+1
    end if
       indexP= indexP=1
    end while

    ▷перенос частини списку, що залишилась
if (start1<=end1) then
  for i=start1 to end1
     result(indexP)=list(i)
     indexP=indexP+1
else
  for i=start2 to end2
     result(indexP)=list(i)
     indexP=indexP+1
  end for
end if

▷повернення результату у список
indexP=1
for i=finalStart1 to finalEnd
```

```
    list(i)=result(indexP)
    indexP=indexP+1
end for
```

Проаналізуємо отриманий алгоритм. Почнемо з процедури `MergeLists`. Допустимо, що всі елементи списку S_1 менші всіх елементів списку S_2. Оскільки $S_1(i) < S_2(i)$, $\forall i$ (допускається, що кількість елементів у обох списках однакова), то процедура `MergeLists` розмістить всі елементи списку S_1 у список P. Це означає, що процедура `MergeLists` виконає N_{s1} (N_{s1} - кількість елементів списку S_1) операцій порівняння. У випадку протилежної ситуації, коли $S_1(i) > S_2(i)$, $\forall i$ процедура `MergeLists` перемістить всі елементи із списку S_2 у список P, виконавши при цьому N_{s2} операцій порівняння.

Приклад 2.1. Розглянемо два списки $S_1 = \{9;12;14;15\}$ і $S_2 = \{1;3;5;7\}$. Необхідно списки S_1 і S_2 злити в один список P.

У випадку, що розглядається, відбувається порівняння елемента $S_1(1) = 9$ з елементом $S_2(1) = 1$. Оскільки $S_1(1) > S_2(1)$, то у список P переміщається елемент $S_2(1)$. Знову порівнюємо елемент $S_1(1)$ з елементом $S_2(2)$. Має місце співвідношення $S_1(1) > S_2(2)$ і елемент $S_2(2)$ переміщаємо у список P і т. д. Бачимо, для переміщення всіх елементів списку S_2 у список P необхідно виконати чотири порівняння ($N_{s1} = 4$).

Розглянемо тепер випадок, коли $S_1(1) > S_2(1)$, але всі елементи списку S_1 менші, ніж другий елемент списку S_2. ($\forall S_1(i) < S_2(2)$) У такій ситуації процедура `MergeLists` перенесе елемент $S_2(1)$ у список P, а потім відбудуться порівняння всіх елементів списку S_1 з другим елементом списку S_2. Тому повне число порівнянь буде $N_{s1} + 1$.

Приклад 2.2. Допустимо, що програмою **sort** сформовано два масиви $S_1 = \{5;9;12;14\}$ і $S_2 = \{3;16;19;28\}$. Маємо випадок , коли $S_1(1) > S_2(1)$, але всі елементи списку S_1 менші за $S_2(2)$ ($S_2(2) = 16$). Порівнюємо $S_1(1)$ і $S_2(1)$. Оскільки $S_1(1) > S_2(1)$, то елемент $S_2(1)$ переміщаємо у список P. Потім порівнюємо $S_1(1)$ і $S_2(2)$. Маємо $S_1(1) < S_2(2)$. Елемент $S_1(1)$ переміщуємо у список P і т. д. Неважко підрахувати, що після п'яти порівнянь всі елементи із списку S_1 будуть перемішені у список P.

Тепер допустимо, що $S_1(1)$ знаходиться між $S_2(1)$ і $S_2(2)$, а $S_1(2)$ знаходиться між $S_2(2)$ і $S_2(3)$ і т. д. У такому випадку перенос елементів із списків S_1 і S_2 у список P відбувається почергово: спочатку із S_2, потім із S_1, потім знову із S_2 і знову із S_1 і т. д. до того часу, поки не вичерпаються всі елементи за винятком останнього у списку S_1. Цей випадок є найгіршим: загальне число порівнянь буде - $N_{s1} + N_{s2} - 1$.

Приклад 2.3. Нехай процедура **sort** сформувала два списки $S_1 = \{5;8;10;13\}$ і $S_2 = \{2;6;9;11\}$. Аналізуючи списки S_1 і S_2, бачимо, що $S_1(1)$ знаходиться між $S_2(1)$ і $S_2(2)$, а

$S_1(2)$ - між $S_2(2)$ і $S_2(3)$; $S_1(3)$ знаходиться між $S_2(3)$ і $S_2(4)$. Порівнюємо елемент $S_1(1)$ і $S_2(1)$. Оскільки має місце співвідношення $S_1(1) > S_2(1)$, то елемент $S_2(1)$ поміщаємо у список P. Потім порівнюємо $S_1(1)$ з $S_2(2)$. Має місце нерівність $S_1(1) < S_2(2)$. Це означає, що $S_1(1)$ попаде у список P. Тепер порівнюємо $S_1(2)$ і $S_2(2)$. Бачимо, що $S_1(2) > S_2(2)$ і $S_2(2)$ потрапляє до списку P. В подальшому процес переносу елементів із списків S_1 і S_2 у список P відбувається аналогічно (табл. 2.1).

Таблиця 2.1 – Розміщення елементів масивів S_1 і S_2 у списку P

Елементи списку P	$S_2(1)$	$S_1(1)$	$S_2(2)$	$S_1(2)$	$S_2(3)$	$S_1(3)$	$S_2(4)$	$S_1(4)$
Значення елементів списку P	2	5	6	8	9	10	11	13

Проаналізуємо тепер процедуру `MergeSort`, яка викликається до тих пір поки `first<last`. Виконання останньої умови приводить до розбиття загального списку S на дві половини, тобто кожний із списків S_1 і S_2 буде мати по $\frac{n}{2}$ елементи. Це означає, що у найгіршому випадку процедура `MergeLists` здійснить $N_{s1} + N_{s2} - 1$ порівнянь. Оскільки ми допустили, що $N_{s1} = N_{s2} = \frac{n}{2}$, де n - загальне число елементів у початковому списку, який необхідно відсортувати, то будемо мати $\frac{n}{2} + \frac{n}{2} - 1$ порівнянь. На виконання операцій сортування у списках S_1 і S_2 процедура `MergeSort` затратить $T\left(\frac{n}{2}\right)$ операцій. Це означає, що у найгіршому випадку загальне число операцій (складність алгоритму) буде таким:

$$T(n) = 2T\left(\frac{n}{2}\right) + n - 1. \qquad (2.2)$$

Якщо список вміщує тільки один елемент, то він уже відсортований і процедура `MergeLists` не здійснює порівняння. Тому $T(1) = 0$.

Розв'яжемо рекурентне співвідношення (2.2). Нехай у виразі (2.2) аргумент приймає будь-яке ціле значення k, яке $k \geq 0$, тобто

$$T(k) = 2T\left(\frac{k}{2}\right) + k - 1 \qquad (2.3)$$

Тоді відповіднодо з (2.3) матимемо при $k \in \left\{\frac{n}{2}, \frac{n}{4}, \frac{n}{8}, \frac{n}{16}, \ldots\right\}$

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + \frac{n}{2} - 1,$$

$$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + \frac{n}{4} - 1,$$

$$T\left(\frac{n}{8}\right)=2T\left(\frac{n}{16}\right)+\frac{n}{8}-1,$$

$$T\left(\frac{n}{16}\right)=2T\left(\frac{n}{32}\right)+\frac{n}{16}-1,$$

. . . . . .

Здійснюючи послідовну підстановку отриманих значень у вираз (2.2) отримаємо:

$$T(n)=2\left(2T\left(\frac{n}{4}\right)+\frac{n}{2}-1\right)+n-1=4T\left(\frac{n}{4}\right)+2n-2-1,$$

$$T(n)=4\left(2T\left(\frac{n}{8}\right)+\frac{n}{4}-1\right)+2n-2-1=8T\left(\frac{n}{8}\right)+3n-4-2-1,$$

$$T(n)=8\left(2T\left(\frac{n}{16}\right)+\frac{n}{8}-1\right)+3n-4-2-1=16T\left(\frac{n}{16}\right)+4n-8-4-2-1$$

$$T(n)=16\left(2T\left(\frac{n}{32}\right)+\frac{n}{16}-1\right)+4n-8-4-2-1=32T\left(\frac{n}{32}\right)+5n-16-8-4-2-1$$

. . . . . (2.4)

Процес підставлення можна продовжувати до досягання у правій частині отриманих послідовностей (2.4) $T(1)=T\left(\frac{n}{n}\right)$.

Аналіз послідовності (2.4) показує, що $32=2^5$, тобто коефіцієнт при n співпадає з показником степеня числа, яке є знаменником аргумента $\frac{n}{32}$. Крім того $-16-8-4-2-1=-\sum_{i=0}^{5-1}2^i$.

У загальному випадку $T(n,r)=2^rT\left(\frac{n}{2^r}\right)+rn-\sum_{i=0}^{r-1}2^i$. Нехай $n=2^r$. Тоді

$$T(n,r)=nT\left(\frac{n}{n}\right)+rn-\sum_{i=0}^{r-1}2^i=nT(1)+rn-\sum_{i=0}^{r-1}2^i.$$

Проведений аналіз показав, що при поділі списку S кожний раз на половину $n=2^r$. Звідси випливає, що $r=log_2 n$. Оскільки $T(1)=0$, то

$$T(n)=n\,log_2 n-\sum_{i=0}^{log_2 n-1}2^i. \qquad (2.5)$$

Сума $\sum_{i=0}^{log_2 n-1}2^i$ у формулі (2.5) є геометричною прогресією, знаменник якої $q=2$. Тому

$$\sum_{i=0}^{log_2 n-1}2^i=\frac{a_{r-1}q-a_0}{q-1}.$$

Оскільки $a_0=1$ і $a_{r-1}=2^{log_2 n-1}$, то

$$\sum_{i=0}^{log_2 n-1}2^i=2^{log_2 n}-1=n-1.$$

Таким чином,

$$T(n)=n\,log_2 n-n+1.$$

Це означає, що $T(n) = O(n log_2 n)$ і сортування злиттям є значно ефективнішою за процедуру сортування вставками, де $T(n) = O(n^2)$.

2.2 «Жадібні» алгоритми

«Жадібний» алгоритм на кожному кроці забезпечує локально найкращий вибір – з надією, що кінцеве рішення буде оптимальним.

Суть «жадібного» алгоритму пояснимо на простій задачі. Допустимо, що у продавця є монети вартістю в 20, 10, 5 копійок та 1 копійка. Йому необхідно вернути решту 56 копійок. Алгоритм, який розв'язує цю нескладну задачу полягає у тому, що вибирається монета найбільшої вартості (20 копійок), але не більше 56 копійок. Потім знаходимо різницю 56-20=36. Оскільки 20<36, то наступною монетою буде монета найбільшої вартості (20 копійок). Цю монету продавець добавляє до у список решти (20+20=40). Так як наступна різниця 56-40=16<20, то наступною монетою найбільшої вартості буде 10 копійок. Цю монету добавляємо у список решти: 20+20+10=50. Очевидно, що наступними монетами у списку решти будуть монети вартістю 5 копійок та 1 копійка, що у підсумку дає розв'язок поставленої задачі (20+20+20+10+5++1=56), яка є оптимальною у сенсі мінімальної кількості монет із даного набору.

У даному випадку «жадібність» алгоритму полягає у тому, що на кожному кроці вибирається монета найбільшої вартості, виходячи із умови $B_M = \max(M_1, M_2, ..., M_h) < P_M$, де - 1, 2, …, h - номера наборів монет вартостей $M_1, M_2, ..., M_h$; P_M - список решти. У загальному випадку на кожному кроці «жадібний» алгоритм вибирає той варіант, який є оптимально локальним в тому чи іншому сенсі.

Тепер розглянемо нетривіальний приклад – задачу, яка називається *задачею комівояжера*. Суть її у наступному. Задане певне число населених пунктів, які сполучені між собою дорогами. Відомі віддалі між такими населеними пунктами. Необхідно обійти всі населені пункти таким чином, щоб сумарний пройдений шлях був найкоротшим і при цьому комівояжер повернувся у початковий пункт.

Допустимо, що є шість населених пунктів, віддалі між якими задані у вигляді табл. 2.1

Знайдемо довжину кожного із чотирьох маршрутів. Для маршруту, який показаний на рис. 2.2,а, маємо

$$L_1 = L_{ab} + L_{bc} + L_{cd} + L_{de} + L_{ef} + L_{fa}.$$

Із табл. 2.1 знаходимо, що $L_{ab} = 4,7$; $L_{bc} = 3,4$; $L_{cd} = 15,0$; $L_{de} = 2,4$; $L_{ef} = 4,1$; $L_{fa} = 18,0$ і відповідно $L_1 = 47,6$.

Таблиця 2.1 – Віддалі між населеними пунктами

		Віддаль між населеними пунктами, км					
		a	*b*	*c*	*d*	*e*	*f*
Населені пункти	*a*	0	4,7	1,7	15,7	15,4	18,0
	b	4,7	0	3,4	11,8	10,7	14,5
	c	1,7	3,4	0	15,0	14,1	17,6
	d	15,7	11,8	15,0	0	2,4	5,1
	e	15,4	10,7	14,1	2,4	0	4,1
	f	18,0	14,5	17,6	5,1	4,1	0

Аналогічно знаходимо, що

$$L_2 = L_{ac} + L_{cd} + L_{df} + L_{fe} + L_{eb} + L_{ba} = 1,7 + 15,0 + 5,1 + 4,1 + 10,7 + 4,7 = 41,3,$$

$$L_3 = L_{ac} + L_{cd} + L_{de} + L_{ef} + L_{fb} + L_{ba} = 1,7 + 15,0 + 2,4 + 4,1 + 14,5 + 4,7 = 42,4$$

$$L_4 = L_{ac} + L_{cb} + L_{bd} + L_{de} + L_{ef} + L_{fa} = 1,7 + 3,4 + 11,8 + 2,4 + 4,1 + 18,0 = 41,4$$

.

На рис. 2.2 показано можливі варіанти маршруту комівояжера шістьма населеними пунктами.

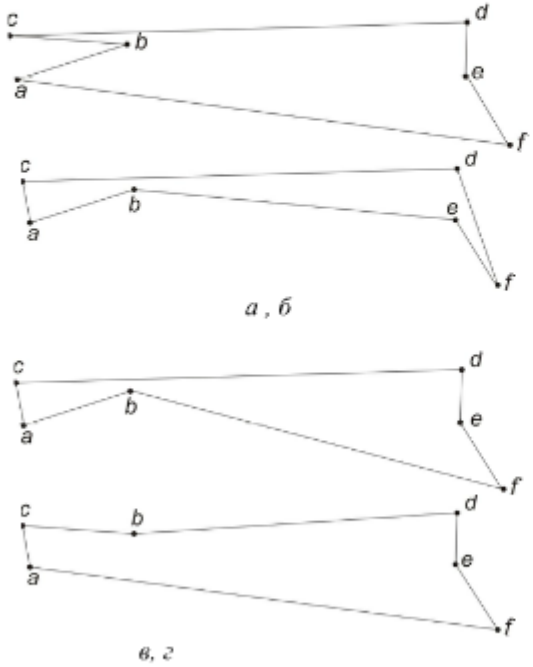


Рисунок 2.2 – Можливі маршрути комівояжера шістьма населеними пунктами

Аналіз отриманих результатів показує, що маршрут L_2, який показаний на рис 2.2,б, є найкоротшим із чотирьох можливих.

Відмітимо, маршрути, які показані на рис. 2.2 можуть розглядатися як графи, у яких вершини – населені пункти, а ребра – шляхи, які з'єднують відповідні населені пункти. Кожне ребро графа має вагу, яка дорівнює віддалі між двома населеними пунктами, що з'єднані відповідним ребром.

Робота «жадібного» алгоритму для задачі комівояжера полягає у тому, що спочатку розглядаються найкоротші ребра. Нове ребро добавляється до уже наявних ребер у тому випадку, коли це не приводить до появи вершини зі степеню три і більше та не утворює цикл (замкнутий маршрут) за винятком випадку, коли кількість включених ребер дорівнює кількості вершин графа.

Множина ребер, що вибрані у відповідності зі зазначеними умовами, утворюють сукупність не з'єднаних шляхів; така ситуація зберігається до виконання останнього кроку, тоді єдиний шлях, що залишився, замикається, утворюючи замкнутий маршрут.

Застосуємо даний алгоритм до сформульованої задачі комівояжера (табл. 2.1). Спочатку вибираємо ребро (a,c), оскільки воно є найкоротшим і має довжину 1,7. Наступним найкоротшим ребром є ребро (d,e) довжиною 2,4. Після перших двох кроків маємо множину ребер - $P_r = \{(a,c),(d,e)\}$. До множини P_r включаємо наступне найкоротше ребро (c,b) довжиною 3,4. Тепер $P_r = \{(a,c),(d,e),(c,b)\}$ і до отриманої множини P_r слід було б включити ребро (e,f), довжина якого 4,1, але воно може з ребром (d,f) утворити цикл. Тому ребро

(e,f) не включаємо до множини P_r. Після четвертого кроку множина P_r залишається незмінною. Із ребер, що залишились, найкоротшим є ребро (d,f), довжина якого – 5,1; ребро (d,f) долучаємо до множини P_r, тобто . $P_r = \{(a,c),(d,e),(c,b),(d,f)\}$. Наступним найкоротшим ребром є ребро (b,e), довжини якого – 10,7, яке включаємо до множини P_r. Отже, $P_r = \{(a,c),(d,e),(c,b),(d,f),(b,e)\}$ Наступним претендентом до включення у множину P_r є ребро (b,d) довжиною 11,8. Оскільки на поточному кроці алгоритму умови включення ребра (b,d) у множину P_r не виконуються, то ребро (b,d) не включаємо у множину P_r. Після семи кроків залишилось сім ребер - (c,e), (b,f), (c,d), (a,e), (a,d), (c,f) і (a,f), довжини яких відповідно 14,1, 14,5, 15,0, 15,4, 15,7 17,6 та 18,0. Неважко переконатись, що умовам включення до множини P_r із семи ребер відповідає тільки ребро (a,f), яке і замикає шлях комівояжера. Таким чином, отримали шлях $a \to c \to b \to e \to d \to f \to a$, довжина якого $L = 41{,}3$. Отриманий маршрут співпадає з найкоротшим маршрутом L_2, тобто можна стверджувати, що отриманий маршрут є оптимальним.

2.3 Динамічне програмування

Динамічне програмування дає змогу розв'язувати задачі, в яких шукана відповідь складається із частин кожна із яких у свою чергу оптимальний розв'язок деякої підзадачі. Динамічне програмування використовують тоді, якщо на різних етапах розв'язку задачі багаторазово зустрічається одні і ті ж підзадачі.

Ідея алгоритму динамічного програмування полягає у тому, що створюють таблицю розв'язків всіх підзадач, які можливо необхідно буде розв'язувати. Таблиця заповнюється незалежно від того чи потрібна та чи інша підзадача для отримання загального розв'язку. Процес заповнення таблиці для отримання розв'язку певної задачі дістало назву *динамічного програмування.*

Форми алгоритму динамічного програмування можуть бути різні, але загальною їх рисою є заповнення таблиці у певному порядку.

Як приклад динамічного програмування розглянемо задачу *тріангуляції.*

Тріангуляція багатокутника - це набір хорд, що ділять багатокутники на трикутники; сторони цих трикутників є сторонами початкового багатокутника і хорд тріангуляції. На рис. 2.3 показано приклад тріангуляції семикутника.

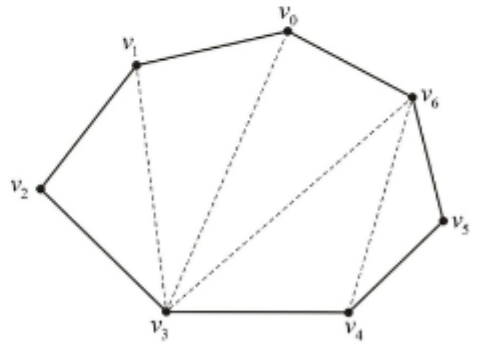


Рисунок 2.3 – Приклад тріангуляції семикутника

Суть задачі у наступному. Заданий випуклий багатокутник[1] $P=\langle v_0, v_1, ..., v_n\rangle$ і вагова функція, що визначена на множині трикутників з вершинами v_0, v_1, …, v_n. Потрібно знайти тріангуляцію, для якої сума ваг трикутників буде мінімальною.

Приклад вагової функції трикутника

$$w(\Delta v_i v_j v_k) = |v_i v_j| + |v_j v_k| + |v_k v_i|,$$

де $|v_i v_j|$ - евклідова віддаль між вершинами v_i і v_j.

Розв'язок сформульованої задачі полягає у тому, що вона розбивається на окремі підзадачі і оптимальне рішення початкової задачі вміщує у собі оптимальний розв'язок підзадач.

На основі сформульованого підходу складемо алгоритм розв'язку задачі тріангуляції багатокутника.

Нехай T - оптимальна тріангуляція $(n+1)$ - кутника $P=\langle v_0, v_1, ..., v_n\rangle$. Сторона $v_0 v_n$ входить в один із трикутників тріангуляції. Припустимо, що це трикутник $\Delta v_0 v_k v_n$, де $1 \le k \le n-1$. Тоді вага тріангуляції T дорівнює вазі трикутника $\Delta v_0 v_k v_n$ плюс сума ваг тріангуляцій багатокутників $\langle v_0, v_1, ..., v_k\rangle$ і $\langle v_k, v_{k+1}, ..., v_n\rangle$.

Розглянемо багатокутник, який показаний на рис. 2.3. Допустимо, що $k=3$. Тоді одержимо дві задачі, які виникають після вибору вершини v_3 (рис. 2.4).

[1] Багатокутник називається *випуклим*, якщо для будь-яких двох точок, що лежать всередині або на границі багатокутника, відрізок який з'єднує ці точки повністю лежить всередині чи на границі багатокутника.

Для багатокутника, який показаний на рис. 2.3 вага тріангуляції (рис. 2.4) буде обчислюватись за такою формулою:

$$w(\textit{тріангуляції}) = w(\Delta v_0 v_3 v_6) + w(\langle v_0, v_1, v_2, v_3\rangle) + w(\langle v_3, v_4, v_5, v_6\rangle)$$

.

Якщо тріангуляції багатокутників $\langle v_0, v_1, ..., v_k\rangle$ і $\langle v_k, v_{k+1}, ..., v_n\rangle$ оптимальні при різних значеннях k, то для тріангуляції початкового багатокутника залишається вибрати оптимальні значення k.

Формалізуємо процес тріангуляції багатокутника. Нехай $m(i,j)$ - вага оптимальної тріангуляції багатокутника $\langle v_{i-1}, v_i, ..., v_j\rangle$, де $1 \le i < j \le n$. Вага оптимальної тріангуляцій всього багатокутника - $m(1,n)$. Для двох суміжних вершин $\langle v_{i-1}, v_i\rangle$ вага дорівнює нулю. Крім того $m(i,i)=0$, $i=\overline{1,n}$.

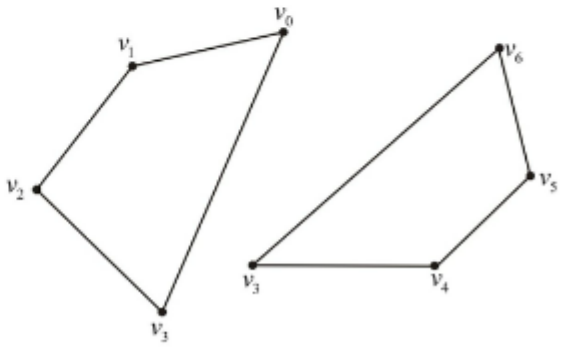


Рисунок 2.4 – Дві підзадачі, які виникли після вибору вершини v_3

Якщо $j-i \geq 1$, то багатокутник $\langle v_{i-1}, v_i, ..., v_j \rangle$ має не менше трьох вершин. У такому випадку необхідно знайти мінімум за всіма значеннями k із множини значень $i \leq k \leq j-1$ такої суми: вага $\Delta v_{i-1} v_k v_j$ плюс вага оптимальної тріангуляції $\langle v_{i-1}, v_i, ..., v_k \rangle$ плюс вага оптимальної тріангуляції $\langle v_k, v_{k+1}, ..., v_j \rangle$. Тому

$$m(i,j) = \begin{cases} 0 & при\ i = j, \\ \min\limits_{i \leq k < j} \left(m(i,k) + m(k+1,j) + w\left(\Delta v_{i-1} v_k v_j\right) \right) & при\ i < j. \end{cases} \quad (2.6)$$

Замість рекурсивного розв'язання співвідношення (2.6) обчислимо оптимальну вагу тріангуляції багатокутника шляхом побудови таблиці у висхідному напрямку. Вхідні дані це послідовність вершин багатокутника, які задані своїми координатами; довжина такої послідовності - $length(v) = n+1$. У процедурі використовується допоміжна таблиця $m(1..n, 1..n)$ для зберігання ваг оптимальної тріангуляції багатокутника. і допоміжна таблиця $s(1..n, 1..n)$, в яку заносяться індекси k. Таблиця s буде використовуватись при побудові оптимального рішення задачі. Нижче наведено алгоритм розв'язку задачі у псевдокоді.

```
Triangulation_Polygon(v)
1  n=length(v)-1
2  for i=1 to n
3      m(i,i)=0
4  end for
5  for l=2 to n
6     for i=1 to n-l+1
7        j=i+l-1
8        m(i,j)=∞
9        for k=I to j-1
10          q=m(i,k)+m(k+1,j)+w(Δv_{i-1}, v_k, v_j)
11            if q<m(I,j)
12               then m(I,j)=q
13                    s(I,j)=k
14            end if
15        end for
16  end for
17 end for
18 return m, s
```

У наведеному алгоритмі (рядки 2 – 3) обчислюються величини, $m(i,i) = 0$ $(i = \overline{1,n})$, які є мінімальними вартостями тріангуляції. Потім у першій ітерації циклу у рядках 5 – 17 обчислюються величини при $m(i,i+1)$ при $i = \overline{1,n-1}$ (мінімальні вартості для послідовностей довжиною $l = 2$). При другому проході цього циклу обчислюються величини $m(i,i+2)$ при $i = \overline{1,n-2}$ (мінімальні вартості для послідовностей довжиною $l = 3$) і т. д. На кожному етапі обчислювальні у рядках 10 – 13 величини $m(i,j)$ залежать тільки від уже обчислених і занесених у таблицю значень $m(i,k)$ і $m(k+1,j)$.

Алгоритм `Triangulation_Polygon(v)` визначає вагу оптимальної тріангуляції; час роботи $\theta(n^3)$, обсяг пам'яті, що використовується, $\theta(n^2)$.

2.4 Алгоритми на графах

Графи є зручними моделями при аналізі роботи і проектуванні комп'ютерних мереж.

Граф є кінцева множина V, яка називається множиною вершин, і множина E двоелементних підмножин множини V.

Множина E носить назву множини *ребер*. Елемент множини Е називають *ребром*. Граф позначають $G(V,E)$.

Зазвичай кінцевий граф, зображають у вигляді діаграми, на якій вершини це точки, а лінії – ребра.

Якщо $\{a,b\}$ – ребра, тоді вершини a і b називають їх кінцями. Ребро (a,b) називають *інцедентним* вершинам a і b. І, навпаки, вершини a і b *інцедентні* ребру (a,b). Дві вершини називають суміжними, якщо вони є кінцями ребра (інцедентні ребру). Два ребра називають суміжними, якщо вони інцедентні одній вершині (рис. 2.5)

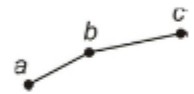


Рисунок 2.5 – Суміжні ребра (*a,b*) і (*b,c*)

Приклад. Граф з множиною вершин V={a,b,c,d,e} і множиною ребер E={(a,b), (a,e), (b,e), (b,c), (b,d), (c,d)} показаний на рис. 2.6

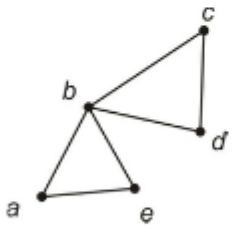


Рисунок 2.6 – Граф з множиною вершин *V* і множиною ребер *E*.

Визначені нами графи носять назву – *простих*. Обмеження на існування тільки одного ребра між двома вершинами дає можливість подати будь–яке ребро не як елемент множини E.

Наше визначення виключає також ребра, які називають петлями (рис. 2.7). Ребро яке з'єднує вершину саму з собою називається петлею. Граф, в якому є петлі, називають *графом з петлями*.

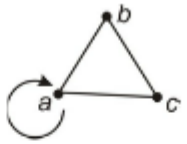


Рисунок 2.7 – Граф з петлею

Якщо в графі є більше одного ребра між двома вершинами, то він носить назву *мультиграфа* (рис. 2.8)

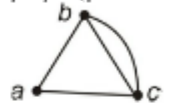


Рисунок 2.8 – Мультиграф.

Якщо граф має петлі і вершини, які з'єднані більше ніж одним ребром, то такий граф називають *псевдографом*

Степеню вершини v, позначають deg(v), називають кількість ребер, які інцидентні даній вершині V. Вершина степені 0 називається *ізольованою*.

Граф, який складається із множини вершин V і множини E упорядкованих пар елементів із V називається *орієнтованим графом* або *орграфом*. Ребра такого графа називають *дугами*. Якщо $(a,b)\in E$, то a – *початкова*, а b – *кінцева* вершини. Відмітимо, що поняття орграфа допускає наявність петель, чого не було у випадку простих графів. Це пояснюються тим, що орграф допускає відношення між елементами, а простий граф визначений як множина вершин і ребер, а в множині два однакових елементи вважають одним елементом.

Якщо a – початкова, а b – кінцева вершини орграфа $G(V,E)$, то вершини a і b *інцидентні* ребру (a,b); вершина a *суміжна* до вершини b, і, навпаки, вершина b також *суміжна* до вершини a.

Степенем виходу вершини v називається кількість ребер, для яких v є початковою вершиною, позначається $outdeg(v)$. Степеню входу вершини v називається кількість ребер, для яких v є кінцевою вершиною і позначається $in\,deg(v)$. Якщо

$in\,deg(v)$=0, то v називається *джерелом*. Якщо *outdeg*(v)=0, то вершина v називається *стоком* (рис. 2.9).

Існує два способи подання граф $G(V,E)$ як набір списків суміжних вершин або як матрицю суміжності.

Подання графа $G(V,E)$ у вигляді *списку суміжних вершин* використовує масив Adj із $|V|$ списків – по одному на кожну вершину ($|V|$ - кількість вершин графа). Для кожної вершини $v \in V$ список суміжних вершин $Adj(v)$ вміщує у довільному порядку всі суміжні з нею вершини, для яких $(v,w) \in E$. На рис. 2.10,б показано подання неорієнтованого графа (рис.2.10,а) за допомогою списку суміжних вершин, а на рис 2.10,в той самий граф поданий за допомогою матриці суміжності.

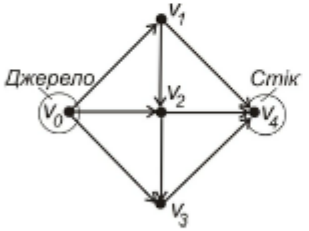


Рисунок 2.9 - Орграф, в якому є джерело і стік

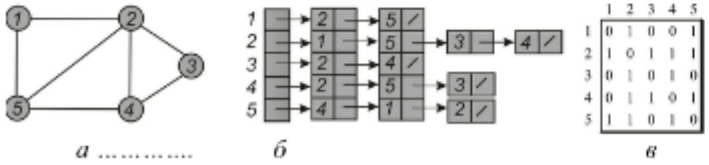


Рисунок 2.10 – Два подання неорієнтованого графа

Аналогічним чином можна подати і орграф: у вигляді списку суміжних вершин (рис. 2.11,б) та у вигляді матриці суміжності (рис. 2.11,в).

Для орієнтованого графа сума всіх елементів матриці суміжності дорівнює числу його дуг (орієнтованих ребер), а для неорієнтованого графа – подвоєній сумі елементів матриці суміжності, так як ребро (v,w) асоційовано в таблиці суміжності з двома елементами: як з вершиною v, так і з вершиною w. В обох випадках кількість пам'яті є $O(V+E)$. Списки суміжних вершин зручні для зберігання *графів з вагами*, в якому кожному ребру назначена певна вага (*вагова функція*). Недолік такого способу наступному: якщо необхідно дізнатись чи є у графа ребро із u у w, то необхідно проглянути увесь список $Adj(v)$ у пошуку w. Цього можна уникнути, подавши граф у вигляді матриці суміжності – але тоді буде потрібно більше пам'яті.

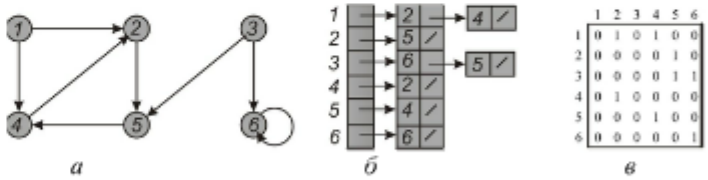


Рисунок 2.11 – Два подання орієнтованого графа

При використанні *матриці суміжності* будемо нумерувати вершини графа числами 1, 2, …, $|V|$ і розглядатимемо матрицю A розміром $|V| \times |V|$, елементи якої

$$a_{ij} = \begin{cases} 1, & якщо\ (i,j) \in E, \\ 0\ у\ протилежному\ випадку. \end{cases}$$

На рис. 2.10,б і 2.11,б наведені матриці суміжності для неорієнтованого і орієнтованого графів відповідною. Матриця суміжності вимагає $\theta(V^2)$ пам'яті незалежно від кількості ребер.

Оскільки матриця суміжності неорієнтованого графа симетрично, то вона співпадає зі своєю транспонованою матрицею, тобто $A = A^T$. Завдяки симетрії матриці A у пам'яті машини можна зберігати лише числа на головній діагоналі і числа, які розміщені вище головної діагоналі. Зберігання ваг ребер графа у пам'яті машини здійснюється у матриці на пересіченні v лінійки і w стовпця.

2.5 Алгоритм пошуку у ширину

Алгоритм пошуку у ширину – один із базових алгоритмів, який є основою для багатьох інших алгоритмів на графах (наприклад алгоритм Дейкстри).

Допустимо, що заданий граф $G(V,E)$.і фіксована початкова вершина s. Алгоритм пошуку у ширину знаходить всі доступні із s вершини у порядку збільшення віддалі від s. Віддалю між будь-якими двома вершинами графа є довжина (число ребер) найкоротшого шляху. У процесі пошуку із графа виокремлюється його частина, яка називається «» з коренем s. Для кожної із вершин, які належать дереву пошуку у ширину, шлях із кореня у дереві буде найкоротшим. Алгоритм може застосовуватись як до неорієнтованих, так і до орієнтованих графів.

Для наочності вважатимемо, що у процесі роботи алгоритму вершини графа можуть бути білими, сірими і чорними. Спочатку всі вершини білі, але в процесі роботи алгоритму біла вершина може стати сірою, а сіра – чорною (але не навпаки). Зустрівши нову вершину (білу або сіру), алгоритм її зафарбовує. Це будуть ті вершини, які уже виявлені алгоритмом. Різниця між сірими і чорними вершинами використовується алгоритмом для керування порядком обходу: сірі вершини утворюють лінію «фронту», а чорні – «тил». Точніше, якщо $(v,w) \in E$, то і v чорна, то w - сіра або чорна.

Спочатку дерево пошуку вміщує тільки один корінь – початкову вершину s. Як тільки алгоритм виявляє нову білу вершину w, яка є суміжною з раніше знайденою вершиною v, вершина w разом з ребром (v,w) долучається до дерева пошуку, стаючи *дитиною* вершини v, а v буде *родичем* вершини w. Кожна вершина виявляється тільки один раз, так що двох родичів у неї не може бути. Поняття предка і нащадка визначається як завжди (потомки – це діти, діти дітей і т. д.). Рухаючись від кореня до вершина дерева пошуку, знаходимо всіх предків.

Наведена нижче процедура BFS (breadth-first search - пошук у ширину) використовує подання графа $G(V,E)$ списком суміжних вершин. Для кожної вершини графа v додатково зберігається її колір $color(v)$ і її попередник $\pi(v)$. Якщо попередника немає (наприклад, $v = s$ або вершина v ще не виявлена), то $\pi(v) = NIL$. Віддаль від s до v записується в поле $d(v)$ Процедура використовує також чергу Q для зберігання множини сірих вершин.

```
BFS(G,s)
1 for (для кожної вершини v ∈ V(G) – {s})
2       color(v)=білий
3       d(v) = ∞
4       π(v) = NIL
5 end for
6 color(s)=сірий
```

```
7  d(s) = 0
8  π(s) = NIL
9  Q={s}
10 while Q ≠ ∅
11       v=head(Q)
12         for (для) всіх w ∈ Adj(v)
13               if color(w)=білий
14                   then color(w)=сірий
15                   d(w)=d(w)+1
16                   π(w) = v
17                   Enqueue(Q,w)
18               end if
19           end for
20         Dequeue(Q,w)
21         color(v)=чорний
22 end while
```

Процедура `BFS` використовує операцію добавляння вершини до черги і вона позначається як `Enqueue` та операцію видалення вершини із черги - `Dequeue`. Тут використано наступне провило: перший прийшов – першим обслужений. Черга має *голову* (head) і *хвіст* (tail). Вершина, що додається до черги стає у хвіст, а вершина, що вилучається із черги, знаходиться у її голові. На рис. 2.12 показано як можна реалізувати чергу, яка вміщує не більше ніж $n-1$ елемент, на базі масиву $Q(1..n)$. Зберігається індекс $head(Q)$ - голова черги та $tail(Q)$ - індекс вільної комірки, в яку буде поміщений наступний елемент, що долучається до черги. Черга складається із елементів масиву з номерами $head(Q)$, $head(Q)+1$, …, $tail(Q)-1$. Якщо $head(Q)=tail(Q)$, то черга пуста. На початку роботи процедури маємо $head(Q)=tail(Q)=1$.

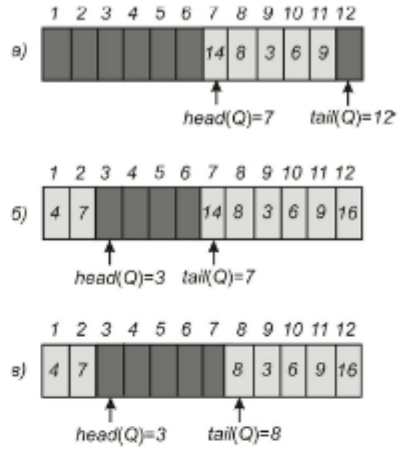


Рисунок 2.12 – Черга, що реалізована на базі масиву Q:

а) світло-сірі клітинки заняті елементами черги. У черзі перебувається 5 елементів (позиції $Q(7..11)$); б) черга після виконання процедур `Enqueue(Q,16)`, `Enqueue(Q,4)`і `Enqueue(Q,7)`; в) черга після виконання процедури `Dequeue(Q)`, яка повертає значення 16. Новою головою черги стає число 8

Нижче наведені процедури додавання елемента (вершини) до черги `Enqueue(Q,w)` та вилучення елемента (вершини) вершини із черги `Dequeue`.

```
Enqueue(Q,w)
1 Q(tail(Q))=w
2 if tail(Q)=length(Q)
3    then tail(Q)=1
4    else tail (Q)= tail (Q)+1
5 end if
```

```
Dequeue(Q,w)
1 w= Q(tail(Q))
2 if head(Q)=length(Q)
3    then head(Q)=1
4    else head(Q)=head(Q)+1
5 end if
6 return w
```

Повернемось тепер до основної процедури BFS. У рядках 1 – 4 вершини стають білими, а всі значення d приймають значення ∞ і родичі всіх вершин об'являються NIL. Рядки 6 – 9 фарбують вершини s у білий колір і виконують пов'язані з цим дії: у рядку 7 віддаль $d(s)$ об'являється рівною нулю, а в рядку 8 говориться, що родичів у s немає. І нарешті у рядку 9 вершина s стає у чергу Q і з цього моменту черга Q буде вміщувати сірі вершини і тільки їх.

Основний цикл програми (рядки 10 – 22) виконується поки черга не пуста, тобто існують сірі вершини. У лінійці 11 така вершина поміщається у v. Цикл for у рядках 12 – 19 проглядає всі суміжні з нею вершини. Виявивши серед них білу вершину , процедура BFS робить її сірою (рядок 14), об'являє її родичів (рядок 16) і встановлюємо віддаль рівною $d(v)+1$ (рядок15). І нарешті, ця вершина добавляється у хвіст черги (рядок 17). Після цього вже можна видалити вершину v із черги Q, перефарбувавши цю вершину у чорний колір (рядки 20 – 21).

Рис. 2.13 наочно демонструє виконання процедури BFS для неорієнтованого графа $G(V,E)$, у якого $|V|=8$, $|E|=9$.

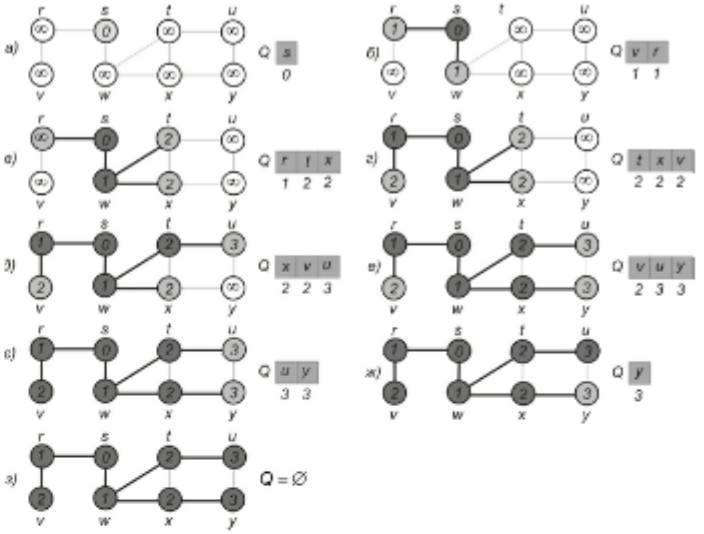


Рисунок 2.13 – Використання процедури BFS для неорієнтованого графа

Вершини дерева, що формуються, показані сірим. Всередині кожної вершини v вказано значення $d(v)$. Показано стан черги перед кожним повторенням циклу у рядках 10 – 22. Поряд з елементами черги вказані віддалі від кореня дерева.

2.6 Алгоритм найкоротшого шляху

Процедура BFS пошуку у ширину знаходить віддаль від початкової вершини s до кожної із вершин графа $G(V,E)$, для яких існує шлях із s. Під віддалю розуміють довжину

найкоротшого шляху: $\delta(s,w)$ визначається як довжина мінімального шляху, що веде із s у w. У нашому випадку довжина шляху – це кількість ребер із s у w. Якщо шляхів немає, то віддаль дорівнює нескінченності. Найкоротших шляхів у графі $G(V,E)$ може бути декілька.

Працюючи на графі орієнтованому чи неорієнтованому з початковою вершиною s, процедура `BFS` виявить (зробить чорними) всі вершини, що є досягнутими із s, і для всіх $w \in V$ буде виконана рівність $d(w) = \delta(s,w)$. Окрім того, для будь-якої вершини $w \neq s$, що є досягнутою із s один із найкоротших шляхів із s у w можна отримати додаючи ребро $(\pi(w), w)$ до (будь-якого) найкоротшого шляху із s у $\pi(w)$. Для вершини, що є не досягнутою, із s значення $\pi(w)$ дорівнює `NIL`.

Контрольні питання та завдання

1. Дайте визначення рекурсивного алгоритму та наведіть приклади такого алгоритму.
2. Доведіть перевагу алгоритму «розділяй і володарюй» над алгоритмом, який здійснює пошук максимального і мінімального елементів шляхом перебору елементів масиву
3. Визначте асимптотичну складність алгоритму сортування злиттям.
4. Деяка ЕОМ виконує одну операцію за 10^{-6} *с*. Який розмір масиву можна обробити на такій машині за 1 *хв.*, якщо використовувати алгоритм сортування вставками; якщо використовувати алгоритм сортування злиттям? Порівняйте отримані результати і зробіть висновок.
5. Дайте визначення «жадібного» алгоритму та наведіть приклади такого алгоритму.
6. Сформулюйте задачу комівояжера та наведіть алгоритм її розв'язку.
7. На який клас задач орієнтоване динамічне програмування?
8. Сформулюйте та наведіть алгоритм розв'язку задачі тріангуляції багатокутників.
9. Основою якого алгоритму може бути алгоритм пошуку у ширину?
10. Яку задачу вирішує алгоритм пошуку в ширину?

3 Побудова алгоритмів

3.1 Алгоритми для роботи з множинами

У цілого ряду додатків використовується така структура даних: n елементів розбиті на непусті множини, що не перетинаються.

Система множин, що не перетинаються це набір непустих множин, що не пересікаються, і у кожному із яких зафіксований один із елементів – *представник*. Точніше, елементи множини є об'єктами і робота з ними здійснюється за допомогою покажчика. При цьому повинні підтримуватись такі процедури.

«*Створити множину*» (`MakeSet`) Процедура створює нову множину, у якій елемент задається за допомогою вказівника x. Оскільки множини не повинні перетинаються, вимагається, щоб x вказував на новий елемент, що не лежить ні в одному із уже створених множин.

«*Знайти множину*» (`FindSet`). Повертає вказівник на представника множини, що вміщує елемент, на який вказує x.

«*Об'єднання*» (`Union`). Процедура тоді може бути застосована, якщо елементи x і y належать до двох різних множин S_x і S_y, і замінюють ці дві множини на одну $S_x \cup S_y$; при цьому вибирається деякий представник для елементу множини $S_x \cup S_y$. Самі множини S_x і S_y вилучаються.

У певних випадках вибір представника повинен підкорятись деяким правилам, наприклад, елементи можуть бути упорядковані і представником є найменший елемент множини. У будь-якому випадку суттєвим є те, що вказівник залишається незмінним до тиго часу, поки сама множина не змінюється.

Приклад 3.1. Алгоритм `ConnectedComponents` (зв'язані компоненти), що наведений нижче, розбиває множину вершин графа на множини, що не пересікаються, і які відповідають зв'язаним компонентам; після цього за допомогою процедури `SameComponent` можна з'ясувати чи містяться дві дані вершини в одній компоненті.

Позначимо через $V[G]$ і $E[G]$ множину вершин і ребер графа G.

```
ConnectedComponents
1 for (для) кожної вершини w ∈ V[G]
2      MakeSet(w)
3 end for
4 for (для) кожного ребра (v,w) ∈ E[G]
5      if FindSet(v) ≠ FindSet(w)
6         then Union(v,w)
7       end if
8 end for
SameComponent
1 if FindSet(v) =FindSet(w)
2    then return true
3    else return false
4 end if
```

Алгоритм `ConnectedComponents` працює таким чином (рис. 3.1) Спочатку кожна вершина розглядається як одноелементна підмножина. Потім для кожного ребра графа об'єднуємо підмножини, у які попали кінці цього ребра. Коли всі ребра опрацьовані, множина вершин розбита на зв'язані компоненти.

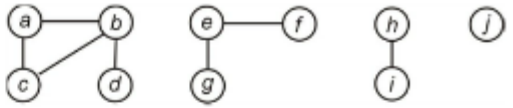
На рис 3.1,а показаний граф, що складається із чотирьох зв'язаних компонент: $\{a,b,c,d\}$, $\{e,g,f\}$, $\{h,i\}$ та $\{j\}$. У процесі роботи алгоритму стан системи множин, що не перетинаються послідовно змінюється і на останньому кроці алгоритм формує чотири зв'язані компоненти (рис. 3.1,б).

Найпростіший варіант реалізації системи множин S, що не перетинаються, зберігати кожну множину у вигляді окремого списку. Для ідентифікації кожного списку із

множини S можна використати *функцію належності*, $h(a)$, $a \in S$, яка відображає елементи множини ідентифікаторів у множину цілих чисел від 0 до $m-1$ (для деяких алгоритмічних мов такі числа набувають значенння від 1 до m). Компонентами масиву A розміром m є вказівники на списки елементів множини S. Список, на який вказує $A(i)$, складається із всіх тих елементів $a \in S$, для яких $h(a)=i$ (рис. 3.2).

Ребро, що опрацьовується	Множини, що не перетинаються									
початок	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(b,d)	$\{a\}$	$\{b,d\}$	$\{c\}$		$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(e,g)	$\{a\}$	$\{b,d\}$	$\{c\}$		$\{e,g\}$	$\{f\}$		$\{h\}$	$\{i\}$	$\{j\}$
(a,c)	$\{a,c\}$	$\{b,d\}$			$\{e,g\}$	$\{f\}$		$\{h\}$	$\{i\}$	$\{j\}$
(h,i)	$\{a,c\}$	$\{b,d\}$			$\{e,g\}$	$\{f\}$		$\{h,i\}$		$\{j\}$
(a,b)	$\{a,b,c,d\}$				$\{e,g\}$	$\{f\}$		$\{h,i\}$		$\{j\}$
(e,f)	$\{a,b,c,d\}$				$\{e,g,f\}$			$\{h,i\}$		$\{j\}$
(b,c)	$\{a,b,c,d\}$				$\{e,g,f\}$			$\{h,i\}$		$\{j\}$

б)



а)

Рисунок 3.1 – Процес опрацювання множин вершин графа

Виконуючи операцію об'єднання (`Union(x,y)`) додаємо список, що вміщує x до кінця списку, що вміщує y. Тепер представником нового списку буде його початок, тобто представник списку, що вміщує y. При цьому виникає необхідність у правильній установці вказівників на початок списку для всіх колишніх елементів множини, що вміщувала x. Час на виконання цієї операції лінійно залежить від розміру вказаної множини.

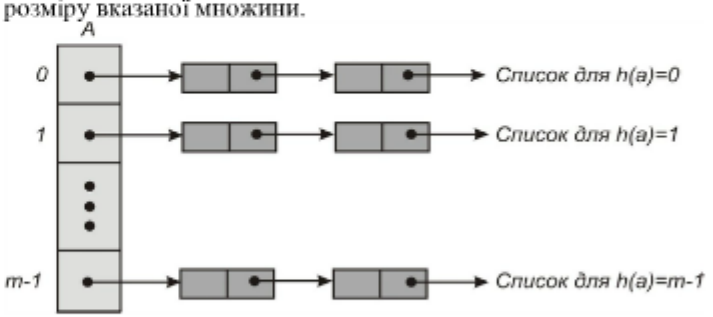


Рисунок 3.2 – Схема ідентифікації списків за допомогою функції розстановки

Припустимо, що список складається із n елементів x_1, x_2, ..., x_n. Виконаємо операції `MakeSet(x1)` для всіх $i=\overline{1,n}$, а потім $n-1$ операцій `Union(x1,x2)`, `Union(x2,x3)`, ...,

`Union(x_{n-1}, x_n)`. марна вартість операції `Union(x_1, x_{i+1})` пропорційна i, то сумарна вартість виконання $2n-1$ операцій буде

$$T(n) = n + \sum_{i=1}^{n-1} i .$$

Використовуючи формулу суми m членів арифметичної прогресії $s_m = \frac{2a_1 + d(m-1)}{2} m$, де a_1 - перший член прогресії; d - різниця прогресії, і враховуючи те, що $a_1 = 1$, $d = 1$ і $m = n-1$, знайдемо

$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2},$$

тобто

$$T(n) = \frac{n(n+1)}{2} = \theta(n^2).$$

Описаний вище спосіб реалізації процедури `Union` вимагає порядку n^2 операцій. Алгоритм `Union` можна покращити, якщо до довшого списку долучати коротший. Такий прийом називається *ваговою евристикою*. Якщо списки, що об'єднуються вміщують приблизно рівну кількість елементів, то великого виграшу не буде, але у протилежному випадку економія може бути відчутною.

Якщо довжина більшого списку m, а довжина меншого n, то загальне число операцій з об'єднання двох списків буде $O(m) + O(n \log_2 n)$.

Приклад 3.2. Нехай задано два списки A і B, які вміщують певні числові величини. Необхідно їх об'єднати в один, використавши прийом вагової евристики. Алгоритм `Union` у псевдокоді буде мати такий вигляд:

```
Union(A,B)
1 n1=length(A)  ▷Кількість елементів у списку A
2 n2=length(B)  ▷Кількість елементів у списку B
▷Першим у загальному списку буде довший список
3 n=max(n1,n2)
4 if n1>n2
5    then C=A
6         D=B
7    else C=B
8         D=A
9 end if
10 k=1
11 for i=n+1 to n1+n2
12     C(i)=D(k)
13     k=k+1
14 end for
```

3.2 Задача знаходження найкоротших шляхів

Задачі подібного типу виникають при синтезі алгоритмів багатошляхової маршрутизації, коли повідомлення, що надходять у комп'ютерну мережу, розбиваються на окремі пакети.

У цьому випадку приходимо до *загальної задачі знаходження найкоротших шляхів*, тобто знаходження найкоротших шляхів між всіма парами вершин орієнтованого графа (орграфа). Більш строге формулювання цієї задачі наступне: заданий орієнтований граф $G(V, E)$, кожному ребру (v, w) цього графа співставлена невід'ємна вартість (ваги ребер) $C(v, w)$. Тоді під довжиною шляху будемо розуміти суму всіх ваг ребер орієнтованого графа, які ведуть із вершини v до вершини w.

Загальна задача знаходження найкоротшого шляху полягає у знаходженні для кожної упорядкованої пари вершин (v, w) будь-якого шляху від вершини v до вершини w,

довжина якого мінімальна серед всіх можливих шляхів від v до w. Алгоритм розв'язку сформованої задачі носить назву *алгоритму Флойда* (R. W. Floyd). Для визначеності допустимо, що всі вершини графа послідовно пронумеровані від 1 до n. Алгоритм Флойда використовує матрицю A розміром $n \times n$, в яку заносять довжини найкоротших шляхів. На початку роботи алгоритму $A(i,j) = C(i,j)$ для всіх $i \neq j$. Якщо вершини i та j не з'єднані між собою, то $C(i,j) = \infty$. Кожний діагональний елемент матриці A дорівнює нулю, тобто $C(i,i) = 0$.

Над матрицею A виконується n ітерацій. Після k-ої ітерації елемент $A(i,j)$ матриці A набуває значення найменшої довжини шляху із вершини i до вершини j, який не проходить через вершини з номером більшим за k. Іншими словами, між кінцевими вершинами i та j можуть бутиь тільки вершини, номери яких не більше k.

На k-ій ітерації для обчислення значень елементів матриці A використовують таку формулу:

$$A_k(i,j) = min\left(A_{k-1}(i,j), A_{k-1}(i,k) + A_{k-1}(k,j)\right), \qquad (3.1)$$

де нижній індекс k позначає значення елемента матриці після k-ої ітерації. Графічну інтерпретацію формули (3.1) показано на рисунку 3.3.

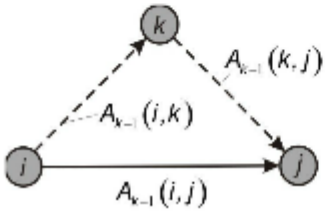


Рисунок 3.3 – Включення вершини k у шлях від i до j

Процедура, яка реалізує алгоритм Флойда буде такою:

```
FloydPath
⊳n – розмір матриці A
1 for i=1 to n
2     for j=1 to n
3        A(i,j)=C(i,j)
4     end for
5 end for
6 for i=1 to n
7     A(i,j)-0
8 end for
9 for k=1 to n
10    for i=1 to n
11        for j=1 to n
12           if A(i,k)+ A(k,j)< A(i,j)
13              then A(i,j)= A(i,k)+ A(k,j)
14            end if
15        end for
16    end for
17 end for
```

Час виконання алгоритму `FloydPath` має порядок $O(n^3)$, оскільки вона вміщує вкладені один в одного три цикли.

У ситуації, коли необхідно визначити найдешевший шлях із одної вершини до іншої, можна в процедуру `FloydPath` ввести ще одну матрицю P, в якій елемент $P(i,j)$ вміщує вершину k, отриману при знаходженні найменшого значення $A(i,j)$. Якщо $P(i,j) = 0$, то найкоротший шлях із вершини i до вершини j утворює тільки одна дуга (i,j).

Модифікована версія алгоритму Флойда, яка дає змогу визначати найкоротший шлях наведена нижче.

```
ModifiedFloydPath
```

```
▷ n - розмір матриці A
1 for i=1 to n
2     for j=1 to n
3        A(i,j)=C(i,j)
4        P(i,j)=0
5     end for
6 end for
7 for i=1 to n
8     A(i,j)=0
9 end for
10 for k=1 to n
11     for i=1 to n
12         for j=1 to n
13            if A(i,k)+ A(k,j)< A(i,j)
14               then A(i,j)= A(i,k)+ A(k,j)
15                    P(i,j)=k
16             end if
17         end for
18     end for
19 end for
```

Для індикації послідовності вершин, які визначають найкоротший шлях від вершини i до вершини j, викликається процедура `Path(i,j)`, псевдокод якої має такий вигляд:

```
Path(i,j)
1 k=P(i,j)
2 if k=0
3     return
4    path(i,k)
5    writeln(k)
6    path(i,k)
7 end if
```

3.3 Множення матриць

Матриця - математичний об'єкт, який задається таблицею чисел a_{ij}, $i=\overline{1,m}$, $j=\overline{1,n}$. Числа m і n визначають розмір матриці A, тобто матриця A має m рядків і n стовпців; на перетині i-го рядка та j-го стовпця знаходиться елемент матриці a_{ij}. Отже, матриця розміром $m\times n$ задається таблицею чисел

$$A=\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}.$$

Дві матриці A і B однакового розміру можна додавати чи віднімати у результаті отримаємо матрицю $C=A+B$ чи $C=A-B$, у якій $c_{ij}=a_{ij}+b_{ij}$ або $c_{ij}=a_{ij}-b_{ij}$, $i=\overline{1,m}$, $j=\overline{1,n}$.

Якщо число стовпців матриці A співпадає з числом рядків матриці B, то такі матриці можна перемножувати, тобто $C=AB$. У тому випадку, коли матриці A і B мають розміри $m\times r$ і $r\times n$ відповідно, то елементи матриці C обчислюють за такою формулою:

$$c_{ij}=\sum_{k=1}^{r} a_{ik}b_{kj},\ i=\overline{1,m},\ j=\overline{1,n}. \qquad (3.2)$$

У результаті виконання операції (3.2) отримаємо матрицю C розміром $m\times n$.

У загальному випадку $AB\neq BA$, тобто операція множення матриць не комутативна.

Стандартний алгоритм множення двох матриць A і B розмірами $m\times r$ і $r\times n$ має такий вигляд:

```
MatrixMultiply
1 Визначити розміри матриць A і B – m,r,n
2 for i=1 to m
3    for j=1 to n
4       C(i,j)=0
5       for k=1 to r
6          C(i,j)=C(i,j)+A(i,k)*B(k,j)
7       end for
8     end for
9 end for
```

Алгоритм `MatrixMultiply` множення матриць A і B розмірами $m\times r$ і $r\times n$ виконує mrn операцій множення і $m(r-1)n$ операцій додавання. Загальна кількість операцій, що виконує алгоритм `MatrixMultiply` при перемноженні двох матриць, дорівнюватиме $mrn+m(r-1)n=mn(2r-1)$.

Крім стандартного алгоритму множення матриць відомі ще такі алгоритми як Винограда та Штрассена, які є ефективнішими за стандартний алгоритм. Якщо, наприклад, перемножуються дві квадратні матриці розмірами n, то наочне уявлення про ефективність кожного із трьох алгоритмів дає таблиця 3.1

Таблиця 3.1 – Порівняння ефективності трьох Алгоритмів

Алгоритм	*Операції*	
	множення	*додавання*
Стандартний	n^3	n^3-n^2
Винограда	$\frac{n^3+2n^2}{2}$	$\frac{3n^3+4n^2-4n}{2}$
Штрассена	$n^{2.81}$	$6n^{2.81}-6n^2$

Економія обчислювальних ресурсів при використанні алгоритмів Винограда чи Штрассена проявляється при множенні матриць великої розмірності, що досягається шляхом ускладнення відповідних алгоритмів. Алгоритм Штрассена на практиці застосовується рідко. Застосування його виправдано для щільних матриць (таких, що вміщують мало нулів) і, розмір яких досить великий (приблизно 45×45). Матриці невеликого розміру рекомендовано перемножувати за допомогою стандартного алгоритму.

Ми розглянули алгоритм перемноження двох матриць A і B. На практиці зустрічаються задачі, коли необхідно знайти добуток ланцюжка матриць

$$A=A_1A_2\cdots A_n, \tag{3.3}$$

де матриця A_i має розмір $p_{i-1}\times p_i$, $i=\overline{1,n}$.

Ланцюжок матриць (3.3) можна перемножувати природнім способом – зліва направо. При множенні матриці A_1 на A_2 розмірами $p_0\times p_1$ і $p_1\times p_2$ отримаємо матрицю $A_{12}=A_1A_2$ розміром $p_0\times p_2$. Потім до матриць A_{12} та A_3 застосовуємо операцію множення, що дає змогу отримати матрицю $A_{123}=A_{12}A_3$ розміром $p_0\times p_3$ і т. д.

Виявляється, що кількість операцій, які необхідно затратити для пошуку добутку ланцюжка матриць (3.3) можна значно скоротити, якщо порядок множення матриць визначити певною розстановкою дужок. Нагадаємо, що операція множення матриць володіє властивістю асоціативності, тому результат не залежить від способу розстановки дужок. Наприклад, якщо задана послідовність матриць $\langle A_1,A_2,A_3,A_4\rangle$, то спосіб їх обчислення повністю визначається за допомогою розстановки дужок п'ятьма різними варіантами

$$\left(A_1\left(A_2\left(A_3A_4\right)\right)\right),$$
$$\left(A_1\left(\left(A_2A_3\right)A_4\right)\right),$$
$$\left(\left(A_1A_2\right)\left(A_3A_4\right)\right),$$
$$\left(\left(A_1\left(A_2A_3\right)\right)A_4\right),$$
$$\left(\left(\left(A_1A_2\right)A_3\right)A_4\right).$$

Час обчислення добутку матриць $C = AB$ визначається кількістю скалярних добутків, які обчислюються у рядку 6 процедури `MatrixMultiply`. Ця кількість дорівнює mrn. Якщо i-ту лінійку матриці A та j-тий стовпець матриці B розглядати як вектори з компонентами $\overline{a}_i = \left(a_{i1}, a_{i2}, ..., a_{ik}\right)^T$ і $\overline{b}_j = \left(b_{1j}, b_{2j}, ..., b_{kj}\right)^T$, то під скалярним добутком будемо розуміти операцію знаходження елемента c_{ij} матриці C як скалярний добуток двох векторів $c_{ij} = \overline{a}_i^T \overline{b}_j$.

Щоб проілюструвати як спосіб розстановки дужок при перемноженні декількох матриць впливає на кількість операцій, що виконуються, розглянемо приклад, у якому перемножуються три матриці $\left\langle A_1, A_2, A_3\right\rangle$. Допустимо, що розміри цих матриць – 10×100, 100×5 та 5×50. Перемножуючи ланцюжок матриць $A_1A_2A_3$ у природному порядку – зліва направо; спочатку отримаємо матрицю $A_{12} = A_1A_2$, а потім матрицю $A_{123} = A_{12}A_3$. Розміри матриць A_1 і A_2 відповідно дорівнюють $p_0 \times p_1$ і $p_1 \times p_2$ відповідно, де $p_0 = 10$, $p_1 = 100$ і $p_2 = 5$. Матриця A_{12} буде мати розмір $p_0 \times p_2 = 10\times5$. Для обчислення всіх елементів матриці A_{12} необхідно обчислити $p_0p_1p_2 = 10\cdot100\cdot5 = 5000$ скалярних добутків. Знаходження елементів матриці A_{123}, яка матиме розмір $p_0 \times p_3$, потребує $p_0p_2p_3 = 10\cdot5\cdot50 = 2500$. Разом будемо мати 5000+2500=7500 скалярних множень. Якщо обчислити результат, який задається виразом $\left(A_1\left(A_2A_3\right)\right)$, то неважко підрахувати, що отримаємо 75000 скалярних множень. Таким чином, для обчислення першим способом добутку ланцюжка матриць $A_1A_2A_3$ затрати часу будуть у 10 раз менші, ніж у другому випадку.

Задачу про перемноження послідовності матриць сформулюємо наступним чином: для заданої послідовності матриць $\left\langle A_1A_2\cdots A_n\right\rangle$, в якій матриця A_i має розмір $p_{i-1} \times p_i$, за допомогою дужок повністю визначити порядок множення у матричному добутку $A_1A_2\cdots A_n$, при якому кількість скалярних множень буде мінімальною.

Звернемо увагу на те, що операція перемноження матриць у задачу не входить. Мета розв'язку поставленої задачі – визначити оптимальний порядок перемноження матриць.

Сформульована задача, у принципі, може бути розв'язана шляхом перебору всіх можливих варіантів розстановки дужок у матричному добутку $A_1A_2\cdots A_n$. Позначимо через $P(n)$ кількість альтернативних способів розстановки дужок у послідовності, що складається із n матриць. Якщо $n = 1, 2$, то порядок розстановки дужок повністю визначений і - $P(n) = 1$. При $n > 2$ величину $P(n)$ можна визначити із рекурентного співвідношення

$$P(n)=\begin{cases}1 & при\ n=1,2,\\ \sum_{k=1}^{n-1}P(k)P(n-k) & при\ n>2.\end{cases} \qquad (3.4)$$

Розв'язком рекурентної співвідношення (3.4) є так звана послідовність чисел *Каталана*, яка зростає як $\Omega\left(\frac{4^n}{n^{3/2}}\right)$. Таким чином, кількість варіантів розстановки дужок має експоненціальне збільшення зі зростанням n, і метод прямого перебору не є ефективним для визначення оптимальної стратегії розстановки дужок.

Для розв'язку поставленої задачі застосуємо алгоритмічну стратегію динамічного програмування.

Позначимо результат перемноження матриць $A_iA_{i+1}\cdots A_j$ через $A_{i..j}$, де $i\le j$. Якщо задача нетривіальна, то будь-який спосіб розстановки дужок розбиває добуток $A_iA_{i+1}\cdots A_j$ між матрицями A_k і A_{k+1}, де k - ціле число, що задовольняє умові $i\le k<j$. Таким чином, при деякому k спочатку обчислюються матриці $A_{i..k}$ і $A_{k+1..j}$, а потім вони перемножуються між собою і у результаті отримуємо добуток $A_{i..j}$. Вартість, яка визначається вибраним способом розстановки дужок дорівнює сумі вартості обчислення матриці $A_{i..k}$, вартості обчислення матриці $A_{k+1..j}$ та вартості обчислення добутку матриць $A_{i..k}$ і $A_{k+1..j}$.

Нехай $m(i,j)$ мінімальна кількість скалярних добутків необхідних для обчислення матриці $A_{i..j}$. Тоді повна вартість мінімальної кількості скалярних добутків для обчислення $A_{1..n}$ буде - $m(1,n)$.

Оскільки початкову задачу ми розбили на дві підзадачі, то величина $m(i,j)$ дорівнює сумі мінімальних вартостей обчислення часткових добутків $A_{i..k}$ і $A_{k+1..j}$ плюс вартість множення матриць $A_{i..k}$ і $A_{k+1..j}$ одну на одну. Якщо врахувати, що кожна A_i матриця має розмір $p_{i-1}\times p_i$, то неважко зрозуміти, що для обчислення добутку матриць $A_{i..k}$ і $A_{k+1..j}$ знадобиться $p_{i-1}p_kp_j$ скалярних добутків, тобто

$$m(i,j)=m(i,k)+m(k+1,j)+p_{i-1}p_kp_j. \qquad (3.5)$$

У рекурсивному співвідношенні невідомою є величина k. Для знаходження значення k є $j-i$ можливостей, а саме - $k=\overline{i,j-1}$, тобто

$$m(i,j)=\begin{cases}0 & при\ i=j,\\ \min\limits_{i\le k<j}\{m(i,k)+m(k+1,j)+p_{i-1}p_kp_j\} & при\ i<j.\end{cases} \qquad (3.6)$$

В описаній нижче процедурі `MatrixChainOrder` допускається, що розміри матриці A_i - $p_{i-1}\times p_i$, $i=\overline{1,n}$. Вхідні дані – це послідовність $p=\langle p_0,p_1,\ldots,p_n\rangle$, розмір якої $length(p)=n+1$. у процедурі використовується допоміжна таблиця $m(1..n,1..n)$ для зберігання вартостей $m(i,j)$. Таблиця s використовується для побудови оптимального рішення. У ході виконання алгоритму таблиця m заповнюється у порядку збільшення довжини послідовності матриць.

```
MatrixChainOrder
1 n=length(p)
2 for i=1 to n
3    m(i,i)=0
4 end for
5 for l=2 to n
6    for i=1 to n-l+1
7       j=i+l-1
8       m(i,j)=∞
9       for k=i to j-1
10         q=m(i,k)+m(k+1,j)+p_{i-1}p_kp_j
11         if q< m(i,j)
12            then m(i,j)=q
13                 s(i,j)=k
14         end if
15      end for
16   end for
17 end for
```

Спочатку в процедурі `MatrixChainOrder` (рядки 2 – 4) величинам $m(i,i)$ присвоюють нульове значення, що відповідає послідовності нульової довжини. У першій ітерації циклу (рядки 5 – 16) за допомогою рекурентного співвідношення (3.6) обчислюються величини $m(i,i+1)$ при $i=\overline{1,n-1}$ - мінімальні вартості для послідовностей довжини $l=2$. При другому проході цього ж циклу обчислюються величини $m(i,i+2)$ при $i=\overline{1,n-2}$ - мінімальні вартості для послідовностей довжини $l=3$. і т. д. На кожному етапі уже вичислені у рядках 9 – 17 величина $m(i,j)$ залежать тільки від попередньо обчислених і занесених у таблицю значень $m(i,k)$ і $m(k+1,j)$.

Оптимальне значення k нескладно знайти за допомогою інформації, що зберігається у таблиці s. Кожний елемент таблиці $s(i,j)$ вміщує індекс k, де при оптимальному розміщенні дужок у послідовності виконано розбиття. $A_iA_{i+1}\cdots A_j$. Таким чином, нам відомо, що оптимальне обчислення добутку матриць виглядає наступним чином: $A_{1\cdots s(1,n)}A_{1\cdots s(1,n)+1\cdots n}$. Ці часткові добутки матриць можна обчислити рекурсивно, оскільки елемент $s(1,s(1,n))$ визначає матричне множення, що виконується останнім при обчисленні $A_{1\cdots s(1,s)}$, а $s(s(1,n)+1,n)$ - останнє множення при обчисленні $A_{1\cdots s(1,s)+1\cdots n}$.

Приклад 3.3 Для матриць, розміри яких наведені у таблиці 3.2 визначити оптимальну розстановку дужок з використанням процедури `MatrixChainOrder`.

Оскільки використовується тільки частина таблиці m у якій елементи визначені тільки при $i \le j$, то використовується тільки та її частина, що розміщена над верхньою діагоналлю (рис. 3.4). На рисунку 3.4 таблиця m повернута таким чином, щоб її головна діагональ розмістилась горизонтально. Із даної таблиці легко знайти мінімальну вартість $m(i,j)$ часткового добутку матриць $A_{i\cdots j}=A_iA_{i+1}\cdot A_j$. Вона міститься на перетині ліній, що йдуть від матриці A_i вправо і вгору, і від матриці A_j - вліво і вгору. У кожному горизонтальному рядку матриці є вартості перемноження часткових послідовностей, які складаються з однакової кількості матриць. У процедурі рядки обчислюються знизу вверх, а елементи у кожному рядку - зліва направо. із таблиці m видно, що мінімальна вартість скалярних добутків, яка необхідна для перемноження шести матриць із таблиці 3.2, дорівнює 15125. Таблиця s дає змогу визначити порядок розстановки дужок у добутку матриць $A_1A_2A_3A_4A_5A_6$. У верхній клітинці таблиці s є значення k, яке розбиває добуток матриць на два співмножники $A_1A_2\cdots A_k$ і

$A_{k+1}A_{k+2}\cdots A_6$. Оскільки $k=3$, то оптимальна розстановка дужок у добутку матриць буде такою: $\left(A_1\left(A_2A_3\right)\right)\left(\left(A_4A_5\right)A_6\right)$.

Таблиця 3.2 – Матриці та їх розміри

Матриця	*Розмір*
A_1	30×35
A_2	35×15
A_3	15×5
A_4	5×10
A_5	10×20
A_6	20×25

Для порівняння підрахуємо кількість скалярних добутків шести матриць, коли множення здійснюється зліва направо (природнім чином) (табл. 3.3).

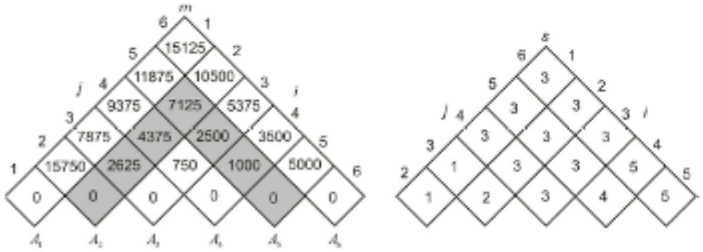


Рисунок 3.4 – Таблиці m та s, що обчислюються процедурою `MatrixChainOrder` для $n=6$

Порівняльний аналіз отриманих результатів показує, що при використанні процедури `MatrixChainOrder` кількість скалярних добутків при перемноженні шести матриць є значно меншою ніж при природному їх множенні.

Аналіз структури вкладених циклів у процедурі `MatrixChainOrder` показує, що час її роботи складає $O\left(n^3\right)$. Для збереження таблиць m і s необхідний обсяг пам'яті складає $\theta\left(n^2\right)$.

Таким чином, процедура `MatrixChainOrder` значно ефективніша, ніж метод перебору і повірки всіх можливих варіантів розстановки дужок, час роботи якого експоненціально залежить від кількості матриць, що перемножуються.

Таблиця 3.3 – Скалярні добутки матриць при природному їх множенні

Часткові добутки матриць	*Розмір матриці* $A_{1..j}$	*Скалярні добутки*
$A_{1..2}=A_1A_2$	30×15	$30\cdot35\cdot15=15750$
$A_{1..3}=A_{12}A_3$	30×5	$30\cdot15\cdot5=2250$
$A_{1..4}=A_{1..3}A_4$	30×10	$30\cdot5\cdot10=1500$
$A_{1..5}=A_{1..4}A_5$	30×20	$30\cdot10\cdot20=6000$
$A_{1..6}=A_{1..5}A_6$	30×25	$30\cdot20\cdot25=15000$
Загальна кількість скалярних добутків		40500

3.4 Швидке перетворення Фур'є

Перетворення Фур'є широко використовується в інженерній практиці для аналізу як періодичних, так і неперіодичних сигналів.

Перетворення Фур'є дійсної функції $a(t)$, яка визначена на нескінченому інтервалі, обчислюється за формулою

$$A(f)=\int_{-\infty}^{\infty} a(t)e^{-j2\pi ft}dt,$$

де f - частота;

j - уявна одиниця.

Якщо область інтегрування не обмежена, то у загальному випадку перетворення Фур'є не існує. Для кінцевого інтервалу часу $[0;T]$ можна отримати фінітне перетворення Фур'є

$$A(f,T)=\int_{0}^{T} a(t)e^{-j2\pi ft}dt.$$

Допустимо, що величина $a(t)$ спостерігається у дискретні моменти часу $a_i=a(t_i)$, де $t_i=i\Delta t$, $i=\overline{0,n-1}$; Δt - крок дискретності. Тоді інтеграл в останньому співвідношенні можна замінити сумою

$$A(f)=\Delta t\sum_{i=0}^{n-1} a_i e^{-j2\pi fi\Delta t}.$$

Для обчислення $A(f,n)$ вибирають, як правило, дискретні значення частот

$$f_k=\frac{k}{T}=\frac{k}{n\Delta t},\ k=\overline{0,n-1}.$$

Тоді

$$y_k=\frac{A(f,k)}{\Delta t}=\sum_{i=0}^{n-1} a_i e^{-j2\pi f_k i\Delta t}.$$

Підставляючи в останню формулу замість f_k його значення, отримуємо

$$y_k=\sum_{i=0}^{n-1} a_i e^{-j\frac{2\pi ki}{n}}.$$

Якщо ввести позначення $\omega_n=e^{-j\frac{2\pi}{n}}$, то

$$y_k=\sum_{i=0}^{n-1} a_i\left(\omega_n^k\right)^i, \qquad (3.6)$$

Величина $\omega_n^k=e^{-j\frac{2\pi k}{n}}$ носить назву *множника повороту* (рис. 3.5).

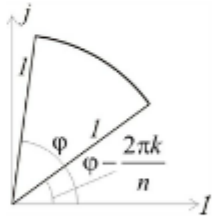


Рисунок 3.5 – Геометрична інтерпретація множника повороту

Вектор $\overline{y}=(y_0,y_1,\ldots,y_{n-1})^T$, компоненти якого визначаються формулою (3.6), називається *дискретним перетворенням Фур'є* (Discrete Fourier Transform, DFT) вектора $\overline{a}=(a_0,a_1,\ldots,a_{n-1})^T$ і позначається таким чином: $\overline{y}=DFT_n(\overline{a})$.

В алгоритмі дискретного перетворення Фур'є використовуються властивості комплексних коренів із одиниці.

Комплексним коренем степеня n із одиниці називають таке комплексне число ω, що

$$\omega^n = 1. \tag{3.7}$$

Рівняння (3.7) має рівно n комплексних коренів

$$\omega_n^k = e^{-j\frac{2\pi k}{n}}, \tag{3.8}$$

де $k = \overline{0, n-1}$; $j = \sqrt{-1}$ - уявна одиниця.

Комплексні корені із одиниці рівномірно розподілені на колі одиничного радіуса з центром у нулі (рис. 3.6). Значення

$$\omega_n = e^{-j\frac{2\pi}{n}} \tag{3.9}$$

називається *головним значенням кореня степені n із одиниці*. Інші корені із одиниці є його степенями.

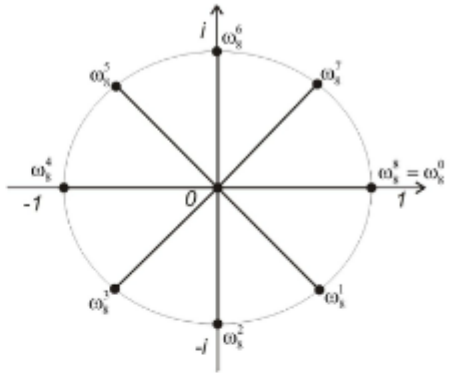


Рисунок 3.6 – Значення $\omega_n^{(k)}$ на комплексній площині ($n = 8$)

Наведемо основні властивості коренів із одиниці.

Властивість 1. для будь-яких цілих $n \geq 0$, $k \geq 0$ і $r \geq 0$

$$\omega_{rn}^{rk} = \omega_n^k. \tag{3.10}$$

Відповідно до (3.8)

$$\omega_{rn}^{rk} = e^{-j\frac{2\pi rk}{rn}} = \omega_n^k. \tag{3.11}$$

Властивість 2. Для будь-якого парного $n > 0$

$$\omega_n^{n/2} = \omega_2 = -1. \tag{3.12}$$

Використовуючи співвідношення (3.8), можемо записати $\omega_n = e^{-j\frac{2\pi n/2}{n}} = e^{-\pi j}$. Згідно з формулою Ейлера

$e^{-\pi j} = cos\,\pi - j\,sin\,\pi = -1$. Із рівності (3.9) випливає, що $\omega_2 = e^{-\pi j} = -1$, тобто має місце формула (3.11).

Властивість 3 (ділення наполовину). Якщо $n > 0$ парне, то, піднісши до квадрату всі n комплексних коренів степеня n із одиниці, отримаємо всі $n/2$ комплексні корені степені $n/2$ із одиниці.

Із властивості 1, коли $r = 2$ випливає, що $\omega_{2n/2}^{2k} = \omega_{n/2}^{k}$, тобто

$$\omega_n^{2k} = \omega_{n/2}^{k}.$$

Властивість 4 (додавання). Для будь-якого $n \geq 1$ і невід'ємного числа k, яке не є кратним до n, має місце рівність

$$\sum_{i=1}^{n-1}\left(\omega_n^k\right)^i = 0. \qquad (3.13)$$

Ряд $\sum_{i=0}^{n-1}\left(\omega_n^k\right)^i = 1 + \omega_n^k + \omega_n^{2k} + \cdots + \omega_n^{k(n-1)}$ представляє собою геометричну прогресію, у якої перший член - $a_1 = 1$, знаменник - $q = \omega_n^k$ і останній член прогресії $a_n = \omega_n^{k(n-1)}$.

Тому $\sum_{i=1}^{n-1}\left(\omega_n^k\right)^i = \frac{\omega_n^k\omega_n^{k(n-1)} - 1}{\omega_n^k - 1} = \frac{\omega_n^k\omega_n^{k(n-1)} - 1}{\omega_n^k - 1}$. Оскільки $\omega_n^k\omega_n^{k(n-1)} = \omega_n^k\omega_n^{kn-k} = \omega_n^k\omega_n^{kn}\omega_n^{-k} = \omega_n^k\left(\omega_n^n\right)^k\omega_n^{-k}$. Оскільки $\omega_n^n = e^{-2\pi jn/n} = e^{-2\pi j} = 1$, то $\omega_n^k\omega_n^{k(n-1)} = 1$. Отже, маємо

$$\sum_{i=1}^{n-1}\left(\omega_n^k\right)^i = \frac{(1)^k - 1}{\omega_n^k - 1} = 0.$$

Знаменник не перетворюється у нуль, так як k не кратне n.

Властивість 5. Величина ω_n^{ki} періодична як по k, так і по i з періодом n.

Нехай r і m будь-які цілі числа. Тоді повинно мати місце таке співвідношення:

$$\omega_n^{(k+rn)(i+mn)} = \omega_n^{ki}. \qquad (3.14)$$

Дійсно $\omega_n^{(k+rn)(i+mn)} = e^{-j\frac{2\pi}{n}\left(ki+kmn+rni+rmn^2\right)} = e^{-j\frac{2\pi ki}{n}}e^{-j2\pi(km+ri+rmn)}$. Величина $q = km + ri + rmn$ є цілим числом, оскільки всі множники і всі доданки цілі числа. Знайдемо $e^{-j2\pi q} = cos\left(2\pi q\right) - j\,sin\left(2\pi q\right) = 1$. Таким чином має місце рівність (3.14).

Властивість 6. Для ω_n^k справедлива формула

$$\omega_n^k = -\omega_n^{k-n/2}. \qquad (3.15)$$

Очевидно, що $-\omega_n^{k-n/2} = -\omega_n^k\omega_n^{-n/2}$. Оскільки $\omega_n^{-n/2} = e^{j\frac{2\pi n/2}{n}} = e^{j\pi} = -1$, то має місце співвідношення (3.15).

Для розв'язку FFT-задачі використаємо метод «розділяй і володарюй». Допускаємо, що степінь полінома (3.14) є степенем числа 2, тобто $n = 2^z$, де z - ціле додатне число. Якщо це не так, то поліном (3.14) доповнюється до полінома степені $n = 2^z$, з нульовими коефіцієнтами при відповідних

степенях. Суму у правій частині рівняння (3.6) розіб'ємо на дві частини – одна із них буде вміщувати тільки парні індекси, а інша лише немарні індекси при коефіцієнтах a_i, тобто

$$a_i^{[0]} = a_{2i},\ a_i^{[1]} = a_{2i+1},\ i = \overline{0, \frac{n}{2}-1}$$

і відповідно

$$y_k = \sum_{i=0}^{n/2-1} a_{2i}\left(\omega_n^k\right)^{2i} + \sum_{i=0}^{n/2-1} a_{2i+1}\left(\omega_n^k\right)^{2i+1}.$$

Із останньої формули випливає, що

$$y_k = y_k^{[0]} + y_k^{[1]},$$

де $y_k^{[0]} = \sum_{i=0}^{n/2-1} a_i^{[0]}\left(\omega_n^k\right)^{2i}$; $y_k^{[1]} = \sum_{i=0}^{n/2-1} a_i^{[1]}\left(\omega_n^k\right)^{2i+1}$; $k = \overline{0, n-1}$.

Враховуючи те, що $\left(\omega_n^k\right)^{2i} = \left(\omega_n^{2k}\right)^i$ і $\left(\omega_n^k\right)^{2i+1} = \omega_n^k\left(\omega_n^k\right)^{2i}$.

Тоді відповідно до властивості 3 $\left(\omega_n^k\right)^{2i} = \left(\omega_{n/2}^k\right)^i$ можемо записати, що

$$y_k = y_k^{[0]}\left(\omega_{n/2}^k\right) + \omega_n^k y_k^{[1]}\left(\omega_{n/2}^k\right), \qquad (3.16)$$

де $y_k^{[0]}\left(\omega_{n/2}^k\right) = \sum_{i=0}^{n/2-1} a_i^{[0]}\left(\omega_{n/2}^k\right)^i$; $y_k^{[1]}\left(\omega_{n/2}^k\right) = \sum_{i=0}^{n/2-1} a_i^{[1]}\left(\omega_{n/2}^k\right)^i$, $k = \overline{0, n/2-1}$.

Формула (3.16) дає змогу обчислити ординати перетворення Фур'є для значень k від 0 до $n/2-1$.

У виразах для $y_k^{[0]}\left(\omega_{n/2}^k\right)$ і $y_k^{[1]}\left(\omega_{n/2}^k\right)$ замінимо k на $k-n/2$. В силу того, що n парне число і відповідно до властивості 5, будемо мати $\omega_{n/2}^{k-n/2} = \omega_{n/2}^k$. Крім того, якщо взяти до уваги властивість 6, то отримаємо формулу

$$y_k = y_{k-n/2}^{[0]}\left(\omega_{n/2}^{k-n/2}\right) - \omega_n^{k-n/2} y_{k-n/2}^{[1]}\left(\omega_{n/2}^{k-n/2}\right), \qquad (3.17)$$

де $y_{k-n/2}^{[0]}\left(\omega_{n/2}^{k-n/2}\right) = \sum_{i=0}^{n/2-1} a_i^{[0]}\left(\omega_{n/2}^{k-n/2}\right)^i$;

$y_{k-n/2}^{[1]}\left(\omega_{n/2}^{k-n/2}\right) = \sum_{i=0}^{n/2-1} a_i^{[1]}\left(\omega_{n/2}^{k-n/2}\right)^i$; $k = \overline{n/2, n-1}$.

У формулі (3.17) замінимо k на $k+n/2$. Тоді

$$y_{k+n/2} = y_k^{[0]}\left(\omega_{n/2}^k\right) - \omega_n^k y_k^{[1]}\left(\omega_{n/2}^k\right),\ k = \overline{0, n/2-1}. \qquad (3.18)$$

Формули (3.16) і (3.18) дають змогу скоротити число множень вдвічі, якщо обчислювати y_k не послідовно від 0 до $n-1$, а попарно - y_k і $y_{k+n/2}$, де $k = \overline{0, n/2-1}$.

Контрольні питання та завдання

1 Які процедури необхідні для роботи змножинами?

2 З якою метою використовують функцію належності при роботі з множинами?

3 Сформулюйте задачу знаходження найкоротших шляхів.

4 Область застосування алгоритму знаходження найкоротших шляхів.

5 Використовуючи процедуру алгоритму Флойда, створіть відповідну комп'ютерну програму на одній із алгоритмічних мов.

6 Задані дві матриці матриця розміром 12x12 кожна. Визначте кількість операцій, які необхідно виконати при їх перемноженні, якщо використовувати стандартний алгоритм,алгоритм Винограда або алгоритм Штрассена.

7 Чи залежить кількість операцій множення декількох матриць (більше двох) від порядку їх перемноження?

8 Використовуючи процедуру MatrixChainord, складіть програму оптимальної розстановки дужок при множенні декількох матриць (більше двох).

9 Область застосування швидкого перетворення Фур'є.

10 Сформулюйте основні властивості перетворення Фур'є.

11 Який метод використовують для розв'язку FFT-задачі?

12 Який ступінь складності має алгоритм швидкого перетворення Фур'є,якщо використовується метод «розділяй і володарюй»?

МЕТОДИ ОБЧИСЛЕНЬ

4 Задачі лінійної алгебри

4.1 Розв'язок систем лінійних рівнянь

Будемо розглядати системи лінійних рівнянь, в яких число невідомих $x_1, x_2, \ldots, x_n$ співпадає з числом рівнянь у системі при цьому допускаємо, що існує єдиний розв'язок такої системи.

Отже, вивчається питання про чисельний розв'язок системи такого виду:

$$\begin{aligned} &a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1, \\ &a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2, \\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ &a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n, \end{aligned} \qquad (4.1)$$

де a_{ij}, $i, j = \overline{1,n}$ - дійсні коефіцієнти системи;

b_i – вільні члени системи.

Систему рівнянь (4.1) можна подати компактніше, якщо ввести квадратну $n \times n$ матрицю, елементами якої є коефіцієнти a_{ij}

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}.$$

Тоді система (4.1) буде еквівалентною такому матричному рівнянню:

$$A\bar{x} = \bar{b}, \tag{4.2}$$

де $\bar{x}^T = (x_1, x_2, \ldots, x_n)$ – вектор невідомих;

$\bar{b}^T = (b_1, b_2, \ldots, b_n)$ – вектор вільних членів системи.

Ефективність способів розв'язку системи (4.1) багато залежить від структури і властивостей матриці A: розміру, обумовленості, симетричності, наповненості (тобто від співвідношення між нульовими і ненульовими елементами), місця і специфіки розміщення ненульових елементів та ін.

Якщо система рівнянь (4.1) має невелике число змінних ($n \le 3$), то для її розв'язку застосовують *формулу Крамера*

$$x_i = \frac{det(A_i)}{det(A)}, i = \overline{1,n}, \tag{4.3}$$

де $det(A_i)$ - детермінант матриці A_i, в якій i – тий стовпець замінений вільними членами;

det(A) - детермінант матриці A.

Інший спосіб розв'язку системи рівнянь (4.1), який по суті еквівалентний (4.3), це знаходження $\bar{x}$ безпосередньо із рівнянням (4.2) шляхом множення зліва лівої і правої частин рівнянь (4.2) на матрицю A^{-1}, що є оберненою до матриці A

$$\bar{x} = A^{-1}\bar{b}. \tag{4.4}$$

Незважаючи на простоту формул (4.3) і (4.4), воно є непридатним для систем з великим числом невідомих, оскільки число машинних операцій множення зв'язаних з обчисленням детермінанта матриці або її оберненням пропорційно факторіалу числа невідомих $O(n!)$. Наприклад, при $n = 100$ $n! \approx 10^{158}$, і при швидкодії ЕОМ 10^9 (операцій множення)/с на розв'язок такої системи необхідно приблизно затратити 10^{141} років, що робить нереальним застосування формул (4.3), (4.4) для систем великої розмірності .

Метод Гауса з вибором головного елемента. Ідея методу Гауса полягає в приведенні матриці A до верхньої, (нижньої) трикутної матриці у результаті чого система рівнянь (4.1) набуде також вигляду:

$$\begin{aligned}
&a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1,\\
&a_{22}^{(1)}x_2 + \ldots + a_{1n}^{(1)}x_n = b_1^{(1)},\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&a_{n-1,n-1}^{(n-2)}x_{n-1} + a_{n-1,n}^{(n-2)}x_n = b_{n-1}^{(n-2)},\\
&a_{nn}^{(n-1)}x_n = b_n^{(n-1)}.
\end{aligned} \tag{4.5}$$

Останнє рівняння системи (4.5) має тільки одне невідоме і його легко розв'язати відносно x_n

$$x_n = \frac{b_n^{(n-1)}}{a_{nn}^{(n-1)}}.$$

Оскільки тепер x_n відомо, то його можна підставити в передостаннє рівняння і знайти

$$x_{n-1} = \frac{b_n^{(n-2)} - a_{n-1,n}^{(n-2)}x_n}{a_{n-1,n-1}^{(n-2)}}.$$

Продовжуючи цей процес, на k – тому етапі так званого *зворотного ходу Гауса,* отримаємо

$$x_k = \frac{1}{a_{kk}^{(k-1)}}\left(b_k^{(k-1)} - \sum_{j=k+1}^{n} a_{kj}^{(k-1)}x_j\right), \quad k = n, n-1, \ldots, 1. \tag{4.6}$$

Сума у формулі (4.6) за визначенням дорівнює нулеві, якщо нижній індекс суми більший за верхній.

Тепер задача полягає у приведені матриці A до діагонального вигляду. Це робиться послідовним виключенням спочатку x_1 із другого, третього ,..., $n-$го рівнянь, потім x_2 із третього, ..., $n-$го рівнянь, x_k із $k+1$, ..., $n-$го рівнянь і т. д.

На першому етапі перше рівняння помножимо на коефіцієнт $-\frac{a_{21}}{a_{11}}$ і додамо його до другого рівняння; потім перше рівняння множимо на коефіцієнт $-\frac{a_{31}}{a_{11}}$ і додаємо до третього рівняння і т.д.; на $k-1-$му кроці перше рівняння множимо на $-\frac{a_{k1}}{a_{11}}$ і додаємо до $k-$того рівняння.

Отже, після $n-1$ кроків отримаємо систему рівнянь:

$$\begin{aligned}
&a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n = b_1,\\
&a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + \ldots + a_{2n}^{(1)}x_n = b_2^{(1)},\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&a_{n2}^{(1)}x_2 + a_{n3}^{(1)}x_3 + \ldots + a_{nn}^{(1)}x_n = b_n^{(1)},
\end{aligned} \tag{4.7}$$

яка буде еквівалентною (4.1).

Нові коефіцієнти системи (4.7) (з верхнім індексом «1») перераховуються за формулами:

$$a_{ij}^{(1)} = a_{ij} - \frac{a_{ij}}{a_{11}}a_{1j},\ b_i^{(1)} = b_i - \frac{a_{i1}}{a_{11}},\ \text{де } i,j = 2,3,\ldots$$

При цьому ми вважаємо, що $a_{11} \neq 0$, оскільки система має єдиний розв'язок і не всі коефіцієнти $a_{i1}, i = \overline{1,n}$ дорівнюють нулю. Це значить, що завжди рівняння з ненульовим коефіцієнтом при змінній x_1 можна поставити першим.

На другому етапі виключають змінну x_2 із третього і наступних рівнянь, виконуючи аналогічні операції, що і на першому етапі. У результаті отримують

$$\begin{aligned}
&a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n = b_1,\\
&a_{12}^{(1)}x_2 + a_{23}^{(1)}x_3 + \ldots + a_{2n}^{(1)}x_n = b_2^{(1)},\\
&a_{33}^{(2)}x_3 + \ldots + a_{3n}^{(2)}x_n = b_3^{(2)},\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&a_{n3}^{(2)}x_3 + \ldots + a_{nn}^{(2)}x_n = b_n^{(2)},
\end{aligned}$$

де $a_{ij}^{(2)} = a_{ij}^{(1)} - \frac{a_{i2}^{(1)}}{a_{22}^{(1)}}a_{2j}, b_i^{(2)} = b_i^{(1)} - \frac{a_{i2}^{(1)}}{a_{22}^{(1)}}b_2^{(1)}$.

Процес послідовного виключення змінних можна продовжити і на $n-1$ кроці отримаємо систему рівнянь (4.5), в якій

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}a_{kj}^{(k-1)},\ b_i^{(k)} = b_i^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}b_k^{(k-1)}, \tag{4.8}$$

$k = \overline{1,n-1}$, $a_{ij}^{(0)} = a_{ij}$, $b_i^{(0)} = b_i$. Нижні індекси i, j змінюються від $k+1$ до n.

Для полегшення запам'ятовування формул можна скористатись правилом прямокутника (рис. 3.9), на основній діагоналі якого розміщені коефіцієнти $a_{ij}^{(k-1)}$ і $a_{kk}^{(k-1)}$, а на допоміжній коефіцієнти $a_{ik}^{(k-1)}$ і $a_{kj}^{(k-1)}$. Тоді коефіцієнти для наступної ітерації обчислюються як різниці добутків коефіцієнтів основної та допоміжної діагоналей, що діляться

на коефіцієнт $a_{kk}^{(k-1)}$. За таким же правилом перераховуються і вільні коефіцієнти системи. Різниця лише у тому, що на схемі, яка наведена на рис. 4.1, коефіцієнт $a_{ij}^{(k-1)}$ необхідно замінити на $b_i^{(k-1)}$, а $a_{kj}^{(k-1)}$ на $b_k^{(k-1)}$.

Степінь складності алгоритму $O\left(n^2\right)$.

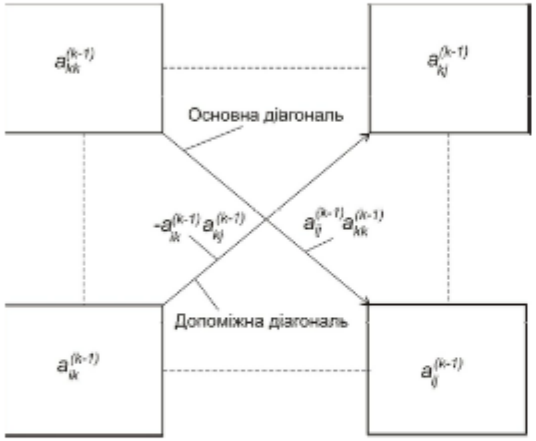


Рисунок 4.1 – Правило прямокутника

Приклад 4.1. Систему лінійних рівнянь

$$\begin{aligned} x_1 + 2x_2 + x_3 + 4x_4 &= 13, \\ 2x_1 + 4x_3 + 3x_4 &= 28, \\ 4x_1 + 2x_2 + 2x_3 + x_4 &= 20, \\ -3x_1 + x_2 + 3x_3 + 2x_4 &= 6 \end{aligned}$$

розв'язати методом Гауса.

Перший етап

Перше рівняння системи множимо на (-2) і додаємо його до другого рівняння

$$\begin{array}{r} -2x_1 - 4x_2 - 2x_3 - 8x_4 = -26 \\ + \quad 2x_1 + 0x_2 + 4x_3 + 3x_4 = 28 \\ \hline -4x_2 + 2x_3 - 5x_4 = 2 \end{array}$$

На наступному кроці перше рівняння множимо на (-4) і додаємо його до третього рівняння

$$\begin{array}{r} -4x_1 - 8x_2 - 4x_3 - 16x_4 = -52 \\ + \quad 4x_1 + 2x_2 + 2x_3 + x_4 = 20 \\ \hline -6x_2 - 2x_3 - 15x_4 = -32 \end{array}$$

І накінець перше рівняння множимо на 3 і додаємо його до четвертого рівняння

$$\begin{array}{r} 3x_1 + 6x_2 + 3x_3 + 12x_4 = 39 \\ + \quad -3x_1 + x_2 + 3x_3 + 2x_4 = 6 \\ \hline 7x_2 + 6x_3 + 14x_4 = 45 \end{array}.$$

Отже, після першого етапу отримали таку систему рівнянь:

$$\begin{aligned} x_1 + 2x_2 + x_3 + 4x_4 &= 13, \\ -4x_2 + 2x_3 - 5x_4 &= 2, \\ 6x_2 + 2x_3 + 15x_4 &= 32, \\ 7x_2 + 6x_3 + 14x_4 &= 45. \end{aligned}$$

Другий етап

На другому етапі виконуються кроки аналогічні крокам, що виконуються на першому етапі. Тільки вихідною системою рівнянь є система, що отримана після першого етапу обчислень.

Тепер «базовим» рівнянням системи буде її друге рівняння. Це рівняння множимо на $\frac{6}{4}$ і додаємо його до третього рівняння

$$\begin{array}{r} -6x_2+3x_3-7{,}5x_4=3 \\ +\ 6x_2+2x_3+15x_4=32 \\ \hline 5x_3+7{,}5x_4=35 \end{array}.$$

На другому кроці друге рівняння множимо на $\frac{7}{4}$ і додаємо його до четвертого рівняння. у результаті отримаємо

$$\begin{array}{r} -7x_2+3{,}5x_3-8{,}75x_4=3{,}5 \\ +\ 7x_2+6x_3+14x_4=45 \\ \hline 9{,}5x_3+5{,}25x_4=48{,}5 \end{array}$$

Після закінчення другого етапу обчислень, отримали таку систему рівнянь:

$$\begin{aligned} x_1+2x_2+x_3+4x_4&=13,\\ -4x_2+2x_3-5x_4&=2,\\ 5x_3+7{,}5x_4&=35,\\ 9{,}5x_3+5{,}25x_4&=48{,}5. \end{aligned}$$

Третій етап

На третьому етапі «базовим» буде третє рівняння системи, яку ми отримали на другому етапі. Це рівняння множимо на $\left(-\frac{9{,}5}{5}\right)$ і додаємо до четвертого рівняння

$$\begin{array}{r} -9{,}5x_3-14{,}25x_4=-66{,}5 \\ +\ 9{,}5x_3+5{,}25x_4=48{,}5 \\ \hline -9x_4=-18 \end{array}.$$

Після трьох етапів обчислень отримали верхню трикутну систему лінійних алгебраїчних рівнянь

$$\begin{aligned} x_1+2x_2+x_3+4x_4&=13,\\ -4x_2+2x_3-5x_4&=2,\\ 5x_3+7{,}5x_4&=35,\\ 9x_4&=18, \end{aligned}$$

розв'язок якої починаємо із останнього рівняння. Оскільки воно вміщує тільки одну невідому x_4, то його розв'язком буде $x_4=\frac{18}{9}=2$. Тепер знайдене значення x_4 підставляємо у третє рівняння, яке також вміщує тільки одне невідоме x_3. У результаті розв'язку такого рівняння, матимемо $x_3=4$. Аналогічно, підставивши значення x_3 і x_4 у друге рівняння системи отримаємо значення x_2 як її розв'язок - $x_2=-1$. І на кінець, після підстановки значень x_3, x_4 і x_2 у перше рівняння системи знаходимо, що $x_1=3$.

Наведена нижче процедура `SystemEqualizationAgebra`[1] знаходить розв'язок системи лінійних алгебраїчних рівнянь, яка подана у матрично-векторній формі (4.2)

```
SystemEqualizationAgebra*
⊳A- матриця коефіцієнтів системи розміром nxn
⊳b- вектор вільних членів системи розміром nx1
⊳X - вектор розв'язків системи розміром nx1
1 n=length(b)
2 Ab=[a b]  ⊳об'єднання матриць Ai b
⊳Вибір головного елементу для стовпця k
3 for k=1 to n-1
4     for r=k to n
```

[1] Процедури, що помічено, * реалізовані у програмному середовищі MatLab. Тексти програм наведено у [ ]

```
5          ab(r)=Ab(r,k)
6       end for
7       [Y,j]=Max(abs(ab))
▷Міняємо місцями лінійки k та j
▷Створюємо нульову матрицю C розміром 1xn+1
8       C = Ab_k
9       Ab_k  = Ab_{j+k-1}
10       Ab_k  = C
11      if AB(k,k)=0
12           then error ('Матриця A особлива. Система
немає розв'язку')
13      end
▷Процес виключення для стовпця k
14      for p=k to n+1
15          M=Ab(p,k)/Ab(k,k)
16          for r=k to n+1
17             Ab(p,r)=Ab(p,r)-M*Ab(k,r)
18          end for
19       end for
20 end for
▷Створюємо нульову матрицю As розміром nxn
▷Створюємо нульовий вектор bs розміром nx1
21 As=Ab
22 bs=b
▷Обчислення розв'язку методом зворотного ходу
▷Створюємо нульовий вектор X розміром nx1
23 X(n)=bs(n)/As(n,n)
24 for k=n-1 step -1 to 1
25      s=0
26      for r=k+1 to n
27         s=s+As(k,r)*X(r)
28      end for
29      p(k)=s
30      X(k)=(bs(k)-p(k))/A(k,k)
31 end for
32 return X
```

У рядках 8 – 9 процедури через Ab_k і Ab_{j+k-1} позначені k-ий і $j+k-1$ рядки відповідних матриць.

Процедура `SystemEqualizationAgebra` побудована так, що на кожному кроці обчислень рівняння у системі (4.1) переставляються таким чином, що у k-ому рядку вибирають найбільший за модулем елемент $a_{kk}^{(k-1)}$. Це запобігає діленню на нуль і дає змогу зменшити похибки від закруглень.

4.2 Обчислення визначників матриць

Метод Гауса рішення лінійних алгебраїчних рівнянь можна застосувати для знаходження визначників квадратних матриць розміром n.

У методі Гауса матриця A перетворюється у верхню трикутну матрицю. Ці перетворення такі, що вони не змінюють абсолютне значення визначника матриці A.

У подальшому при розробленні алгоритму ми будемо спиратись на такі властивості визначників:

- величина визначника не зміниться якщо до елементів одного рядка (або стовпця) додати елементи іншого рядка (стовпця), помножені на одне і й теж число;
- якщо у визначнику поміняти місцями два рядки (стовпці), то знак визначника зміниться на протилежний.

При перетворенні матриці A до верхньої трикутної ми користувались першою і другою властивостями визначників. У результаті отримали

$$\tilde{A}=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & a_{nn}^{(n-1)} \end{bmatrix}.$$

Оскільки матриця A трансформувалась у матрицю $\tilde{A}$, то відповідно до наведених властивостей

$$det(A)=det(\tilde{A}).$$

Враховуючи те, що матриця $\tilde{A}$ - верхня трикутна, будемо мати

$$det(A) = (-1)^p \prod_{i=1}^{n} a_{ii}^{(i-1)}. \qquad (4.9)$$

В останній формулі враховано ту обставину, що при перестановці двох рядків визначник матриці змінює знак на протилежний, тобто у алгоритмі SystemEqualizationAgebra, додатково необхідно врахувати число перестановок p рядків матриці. Крім того, у формулі (4.9) за визначенням $a_{11}^{(0)} = a_{11}$.

Нижче наведена процедура DeterminantMatrix, яка обчислює визначник квадратної матриці A розміром n

```
DeterminantMatrix
▷A- матриця коефіцієнтів системи розміром nxn
1 Визначити розмір квадратної матриці A
2 Ab=A
▷Вибір головного елементу для стовпця k
3 r=0 ▷Лічильник перестановок рядків матриці A
4 for k=1 to n-1
5     for r=k to n
6         ab(r)=Ab(r,k)
7     end for
8     [Y,j]=Max(abs(ab))
▷Міняємо місцями лінійки k та j
9     C = Abk
10     Abk  = Abj+k-1
11     Abk  = C
12     if AB(k,k)=0
13           then error ('Матриця A особлива. Система
немає розв'язку')
14     end
15     if k≠j+k-1
16       then r=r+1
17     end if
▷Процес виключення для стовпця k
18     for p=k to n+1
19         M=Ab(p,k)/Ab(k,k)
20         for r=k to n+1
21            Ab(p,r)=Ab(p,r)-M*Ab(k,r)
22         end for
23     end for
24 end for
25 p=1
26 for i=1 to n
27    p=p*Ab(I,i)
28 end
29 d=(-1)^r*p
30 return d
```

Неважко переконатись, що процедура DeterminantMatrix є частиною процедури SystemEqualizationAgebra у тій частині, де матриця A трансформується у діагональну, різниця лише у тому, що у рядках 15 – 17 процедури DeterminantMatrix Уведений лічильник числа перестановок рядків, які здійснюються у процесі трансформації матриці A до верхньої трикутної.

У рядках 25 -29 процедури здійснюється обчислення визначника матриці A за формулою (4.9).

4.3 Обчислення обернених матриць

Матрицю A^{-1} обернену до матриці A можна знайти як розв'язок матричного рівняння

$$AX = E, \qquad (4.10)$$

де E одинична матриця.

Із останнього рівняння знаходимо $X = A^{-1}E$. Оскільки $A^{-1}E = A^{-1}$, то $X = A^{-1}$. Отже, розв'язавши рівняння (4.10) відносно X, визначимо матрицю A^{-1}.

Подамо шукану матрицю X як набір стовпчикових матриць

$$X = \begin{bmatrix} \overline{x}_1 & \overline{x}_2 & \cdots & \overline{x}_n \end{bmatrix},$$

а одиничну матрицю E як набір векторів

$$E = \begin{bmatrix} \overline{e}_1 & \overline{e}_2 & \cdots & \overline{e}_n \end{bmatrix},$$

де

$$\overline{x}_1 = \begin{bmatrix} x_{11} \\ x_{21} \\ \cdots \\ x_{n1} \end{bmatrix}, \ \overline{x}_1 = \begin{bmatrix} x_{12} \\ x_{22} \\ \cdots \\ x_{n2} \end{bmatrix}, \ \ldots, \ \overline{x}_n = \begin{bmatrix} x_{1n} \\ x_{2n} \\ \cdots \\ x_{nn} \end{bmatrix}; \ \overline{e}_1 = \begin{bmatrix} 1 \\ 0 \\ \cdots \\ 0 \end{bmatrix}, \ \overline{e}_2 = \begin{bmatrix} 0 \\ 1 \\ \cdots \\ 0 \end{bmatrix}, \ \ldots,$$

$$\overline{e}_n = \begin{bmatrix} 0 \\ 0 \\ \cdots \\ 1 \end{bmatrix}.$$

Матричне рівняння (4.10) у відповідності з правилами множення матриць запишемо у такому вигляді

$$\begin{bmatrix} A\overline{x}_1 & A\overline{x}_2 & \cdots & A\overline{x}_n \end{bmatrix} = \begin{bmatrix} \overline{e}_1 & \overline{e}_2 & \cdots & \overline{e}_n \end{bmatrix}.$$

Із останнього співвідношення отримуємо систему матрично-векторних не зв'язаних між собою рівнянь

$$A\overline{x}_1 = \overline{e}_1, \ A\overline{x}_2 = \overline{e}_2, \ \ldots, \ A\overline{x}_n = \overline{e}_n \qquad . (4.11)$$

Кожне із рівнянь (4.11) можна розв'язати за допомогою методу Гауса, послідовно замінивши у (4.2) вектор коефіцієнтів $\overline{b}$ на $\overline{e}_i$, $i = \overline{1,n}$. Для цього можна використати процедуру `SystemEqualizationAgebra`, передбачивши у ній формування векторів $\overline{e}_i$ і організацію додаткового циклу, який дає змогу послідовно обчислювати компоненти вектора $\overline{x}_i$, $i = \overline{1,n}$, із яких стовпець за стовпцем синтезується матриця $X = A^{-1}$.

Контрольні питання та завдання

1. У яких випадках можна застосовувати формулу Крамера для розв'язку систем лінійних алгебраїчних рівнянь?

2. Використавши процедуру `SystemEqualizationAgebra` складіть на одній із алгоритмічних мов програму розв'язку систем лінійних алгебраїчних рівнянь і знайдіть розв'язок системи рівнянь $x_1 + 2x_2 + x_3 = 7$; $4x_1 + x_2 + x_3 = 12$; $x_1 + 2x_2 + 4x_3 = 16$.

3. Чому у процедурі `SystemEqualizationAgebra` рядки переставляються таким чином, щоб у черговій ітерації елемент a_{ik} відповідного стовпця тої частини матриці A, над якою здійснюються перетворення, за модулем був би найбільшим?

4. Які властивості визначників матриць лежать в основі їх обчислення з використанням методу Гауса?

5. На основі процедури `DeterminantMatrix` на одній із алгоритмічних мов складіть програму обчислення визначників матриць і знайдіть визначник матриці

$$A = \begin{bmatrix} 1 & 2 & 1 & 4 \\ 2 & 0 & 4 & 3 \\ 4 & 2 & 2 & 1 \\ -3 & 1 & 3 & 2 \end{bmatrix}.$$

6. Модифікуйте процедуру `SystemEqualizationAgebra` таким чином, щоб вона була придатною для знаходження обернених матриць і на основі отриманої процедури `InverseMatrix` і на одній із алгоритмічних мов складіть програму та знайдіть матрицю обернену матриці A із п. 5.

5 ЗАДАЧІ НЕЛІНІЙНОЇ АЛГЕБРИ

5.1 Розв'язок нелінійних рівнянь

Будемо розглядати задачу знаходження коренів рівняння

$$f(x)=0, \qquad (5.1)$$

де $f(x)$ - алгебраїчна або трансцендентна функція.

Розв'язати рівняння (5.1) означає знайти такі значення аргумента $x=c_j$, за яких виконується рівність

$$f(c_j)=0.$$

У загальному випадку аналітичним способом рівняння (5.1) не розв'язується і тоді звертаються до числових методів, які дають можливість лише *наближено* обчислювати корені рівняння (5.1). Нехай c_j точне значення *j*-го кореня, а x_j - його наближення. Тоді під близькістю наближеного значення x_j до його істинного c_j розуміють виконання нерівності

$$\delta_x=\left|c_j-x_j\right|<\varepsilon$$

за малих $\varepsilon>0$. Іноді важливо контролювати не абсолютну похибку δ_x, а відносну

$$\Delta_x=\frac{\delta_x}{\left|c_j\right|},$$

особливо, коли величина c_j близькою до нуля.

Нелінійна функція $f(x)$ у своїй області визначення може мати кінцеве чи нескінченне число коренів, або не мати їх взагалі.

Переважна більшість числових методів знаходження коренів(нулів) функції $f(x)$ вимагає знання інтервалів $[a_j;b_j]$, де знаходиться *єдиний* корінь. Для функцій загального вигляду немає універсальних способів знаходження інтервалів $[a;b]$. Найпростіший з них є спосіб побудови графіка (якщо це можливо) функції $f(x)$ (рис.5.1).

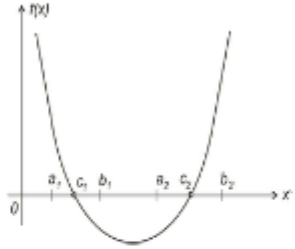


Рисунок 5.1 – Знаходження інтервалів $c_j\in[a_j;b_j]$ за допомогою графіка функції $f(x)$.

Алгоритмічний спосіб (за допомогою ЕОМ) визначення інтервалів $[a_j;b_j]$ грунтується на порівнянні знаків функції $f(x)$ на кінцях інтервалу $[a_j;b_j]$. У тому випадку, коли

$$f(a_j)\cdot f(b_j)<0, \qquad (5.2)$$

то на інтервалі $[a_j;b_j]$ функція принаймні один раз набуває значення нуль. Такий спосіб має один суттєвий недолік: він не дає відповіді на питання про кількість коренів на інтервалі $[a_j;b_j]$.

У випадку виконання умови (5.2) їх може бути більше одного і їх кількість є непарне число (рис. 5.2, *а*);

На рисунку 5.2,б показано випадок, коли умова (5.2) не виконується, хоча відрізок вміщує два корені (кількість коренів парне число).

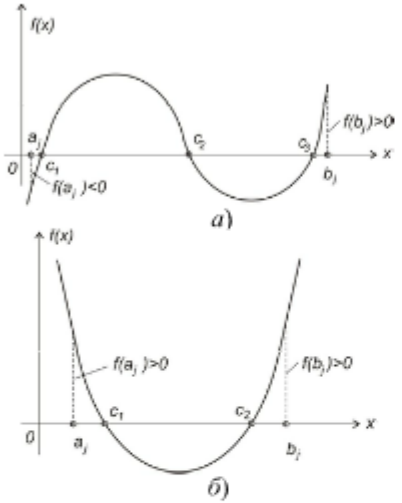


Рисунок 5.2 – Приклади непарної (*а*) і парної (*б*) кількості коренів функції $f(x)$ на інтервалі $\left[a_j;b_j\right]$.

Для того щоб умова (5.2), однозначно давала відповідь про єдиний корінь на інтервалі $\left[a_j;b_j\right]$, необхідно і достатньо, щоб функція на цьому інтервалі була б неперервною і строго монотонною.

Для неперервних функцій $f\left(x\right)$ властивість монотонності на інтервалі $\left[a_j;b_j\right]$ означає, що похідна $f'\left(x\right)$ не змінює свого знака на цьому інтервалі.

Знаходження інтервалу $\left[a_j;b_j\right]$, який вміщує єдиний нуль функції $f(x)$, носить назву *способу локалізації коренів.*

Метод дихотомії. Вказаний спосіб локалізації коренів є основою розв'язку нелінійного рівняння (5.1) і носить назву *методу дихотомії (бісекції).*

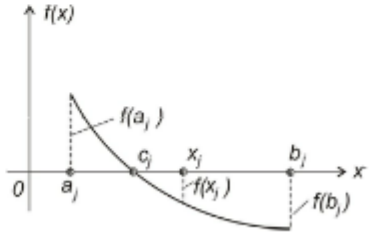
Припустимо, що на відрізку $\left[a_j;b_j\right]$ функція $f\left(x\right)$ неперервна і монотонна, крім того $f\left(a_j\right)$ і $f\left(b_j\right)$ мають протилежні знаки. Це означає, що інтервал $\left[a_j;b_j\right]$ вміщує єдиний корінь. Визначальною операцією у процесі поділу інтервалу наполовину є вибір середньої точки x_j і аналіз трьох можливостей, які при цьому можуть виникнути:

I) якщо $f\left(a_j\right)$ і $f\left(x_j\right)$ мають різні знаки, то нуль належить інтервалу $\left[a_j;x_j\right]$;

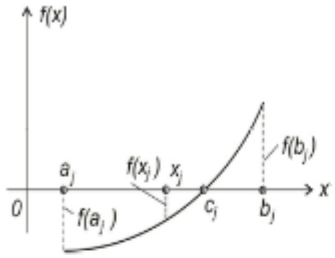
II) якщо $f\left(x_j\right)$ і $f\left(b_j\right)$ мають різні знаки, то нуль належить інтервалу $\left[x_j;b_j\right]$;

III) якщо $f\left(x_j\right)$=0, то $x_j=c_j$.

Випадки i) та ii) ілюструє рис. 5.3.



а) укорочення інтервалу зліва;



б) укорочення інтервалу справа

Рисунок 5.3 – Розв'язок рівняння $f(x)=0$ методом дихотомії:

Подальші обчислення відбуваються за таким алгоритмом:

К1. Ділимо відрізок $[a_j;b_j]$ наполовину і знаходимо середню точку:

$$x_j = \frac{a_j + b_j}{2}.$$

К2. Якщо $f(x_j) = 0$; кінець обчислень.

К3. Якщо $f(a_j)\, f(x_j) < 0$, то $b_j = x_j$ і перейти до *К1*.

К6. Якщо $f(x_j)\, f(b_j)) < 0$, то $a_j = x_j$ і перейти до *К1*.

Умова на другому кроці (*К2*), як правило, не виконується, оскільки, ітераційний процес дає можливість знайти лише наближене значення кореня. Тому її слід замінити іншою умовою $|f(x_j)| \le \varepsilon$, де $\varepsilon > 0$ – досить мати число, яке визначає точність розв'язку задачі.

Оскільки на кожному кроці ітерації вихідний відрізок $[a_j;x_j]$ ділиться наполовину, то він породжує таку послідовність точок, що

$$|c_j - x_k| \le \frac{b-a}{2^{k+1}},\ k = 0, 1, 2, \ldots. \qquad (5.3)$$

Це означає, що послідовність значень $x_k \to c_j$ при $k \to \infty$.

Наведена нижче процедура `MethodDichotomy`, яка подана у псевдокоді, розв'язує нелінійне рівняння $f(x) = 0$ методом дихотомії із заданою точністю $\varepsilon > 0$.

```
MethodDichotomy'

▷ Вхід процедури
▷     Функція f(x)
▷     a та b – початок і кінець інтервалу [a;b],
який вміщує корінь рівняння f(x)=0
▷     epsilon – точність розв'язку задачі f(x)=0
▷     N – число ітерацій
▷  Вихід процедури
▷     c – корінь рівняння f(x)=0
▷     Yc – значення функції f(x) при x=c
▷ Функці f(x) задається процедурою fun_Solve
▷ Побудова графіка функції f(x)
▷  Задати початковий і кінцевий інтервали x0 s
xk, на якому функція f(x) унімодальна, а також крок
delta
    1 k=0
    2 for x=x0 step delta to xk
    3     k=k+1
    4     X(k)=x
    5     Y(k)=fun_Solve(x)
    6 end for
    8 plot(X,Y)
▷ Із графіка функції f(x) знаходимо інтервал
локального нуля [a;b]
    9 Ya=fun_Solve(a)
    10 Yb=fun_Solve(b)
    11 if Ya*Yb>0
```

```
    12   then error('Неправильний інтервал локального
нуля')
    13 end if
    ▷ Обчислення кількості ітерацій
    14 N=1+round((ln((b-a)/epsilon))/ln(2))
    ▷ Реалізація процесу дихотомії
    15 for k=1 to N
    16     c=(a+b)/2
    17     Yc=fun_Solve©
    18     if Yc=0
    19        then a=c
    20        b=c
    21      end if
    22      if Yb*Yc>0
    23        then b=c
    24        Yb=Yc
    25       else a=c
    26         Ya=Yc
    27       end if
    28       if b-a<epsilon
    29          then вихід із циклу
    30       end if
    31 end for
    32 c=(a+b)/2
    33 Yc=fun_Solve(c)
```

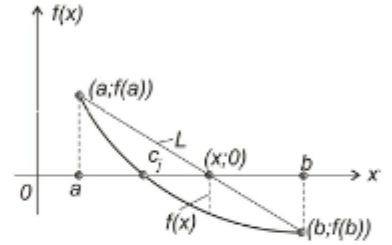
Для знаходження інтервалу $[a_j;b_j]$, який вміщує локальний корінь функції $f(x)$, процедура `MethodDichotomy` будує графік (рядки 1 – 8), із якого знаходимо значення a і b.

Якщо задана точність розв'язку задачі $f(x)=0$ - $|c_j - x_k| \le \varepsilon$, то із (5.3) визначимо максимальну кількість ітерацій N, яка необхідна для досягнення заданої точності ε.

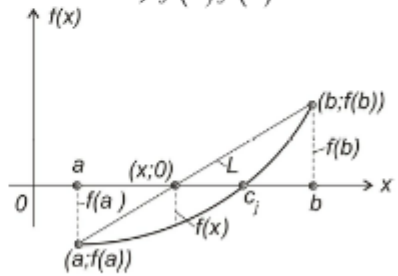
$$N = \max : k = \left[\left(log \frac{b-a}{\varepsilon} \right) / log(2) \right] - 1,$$

де [...]- ціла частина числа.

Метод хорд. Метод дихотомії має невелику збіжність. Збіжність алгоритму розв'язку задачі $f(x)=0$ можна збільшити, якщо відрізок $[a;b]$ ділити точкою x на частини не навпіл, а пропорційно величинам ординат $f(a)$ та функції $f(x)$. Як і раніше допускаємо, що $f(a)$ і $f(b)$ мають протилежні знаки. Знайдемо точку $(x;0)$, в якій хорда L, що з'єднує точки $(a;f(a))$ і $(b;f(b))$ перетинає вісь x (рис. 5.4)



а) $f(a)f(x)<0$



б) $f(x)f(b)<0$

Рисунок 5.4 – Процес розв'язку задачі $f(x)=0$ методом дихотомії

Знайдемо значення x. Для цього спочатку визначимо кут нахилу прямої L до осі абсцис з використанням точок $(a; f(a))$ і $(b; f(b))$ (рис 5.5).

$$m = \operatorname{tg}\alpha = \frac{f(b)-f(a)}{b-a}. \qquad (5.5)$$

Цей самий кут знайдемо, використавши точки $(x;0)$ і $(b; f(b))$

$$m = \frac{f(b)}{b-x} \qquad (5.6)$$

Прирівнявши між собою праві частини рівнянь (5.5) і (5.6), знаходимо

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

Тут, як і в методі дихотомії, існують три можливості:

I) якщо $f(a)$ і $f(x)$ мають різні знаки, то корінь $c = x$ рівняння $f(x)=0$ належить інтервалу $[a; x]$;

II) якщо $f(x)$ і $f(b)$ мають різні знаки, то $c \in [x; b]$ (рис.5.5,б);

III) якщо $f(x)=0$, то коренем функції $f(x)$ є значення $c = x$.

Остання формула разом з і) та іі) використовується для побудови ітераційної процедури

$$x_k = b_k - \frac{f(b_k)(b_k - a_k)}{f(b_k)-f(a_k)}.$$

Метод хорд збігається значно швидше, ніж метод дихотомії в тому випадку, коли $f(x)$ близька до лінійної функції. Але у загальному випадку може статися, що метод хорд буде програвати у швидкості методу дихотомії тоді, коли $f(x)$ значно відрізняється від прямої лінії.

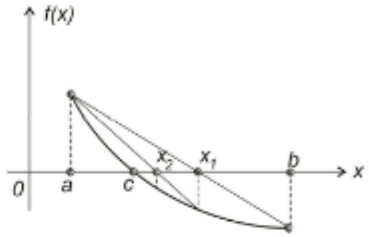


Рисунок 5.5 – Наближення до кореня функції $f(x)$ за методом хорд.

Наступна процедура `MethodChords` знаходить розв'язок задачі $f(x)=0$ методом хорд.

```
    MethodChords'
    ▷ Вхід процедури
    ▷     Функція f(x)
    ▷     a та b - початок і кінець інтервалу [a;b],
який вміщує корінь рівняння f(x)=0
    ▷     epsilon - точність розв'язку задачі f(x)=0
    ▷     eps - граничне відхилення f(x) у нулі
    ▷     N - число ітерацій
    ▷  Вихід процедури
    ▷     c - корінь рівняння f(x)=0
    ▷     Yc - значення функції f(x) при x=c
    ▷ Функці f(x) задається процедурою fun_Solve
    1 Побудувати графік функції f(x) і знайти
інтервал [a; b], який вміщує локальний корінь
    2 Ya=fun_Solve(a)
    3 Yb=fun_Solve(b)
    4 if Ya*Yb>0
```

```
    5    then error('Неправильний інтервал локального
нуля')
    6 end for
    ▷Реалізація процедури методу хорд
    7 for k=1 to N
    8     Dx=Yb*(b-a)/(Yb-Ta)
    9     c=b-Dx
    10    ac=c-a
    11    Yc= fun_Solve(c)
    12    if Yc=0
    13       then вихід із циклу
    14    end if
    15    if Yb*Yc>0
    16       then b=c
    17       Yb=Yc
    18     else
    19       a=c
    20       Ya=Yc
    21    end if
    22    Dx=Min(abs(Dx),ac)
    23    if abs(Dx)<epsilon
    24       then вихід із циклу
    25    end if
    26    if abs(Dx)<eps
    27       then вихід із циклу
    28    end if
    29 end for
    ▷Корінь рівняння f(x)=0
    30 c=(a+b)/2
    31 Yc= fun_Solve(c)
```

Метод Ньютона-Рафсона. Цей метод має вищу збіжність у порівнянні з методами дихотомії і хорд. Будемо вважати, що функція *f(x)* має першу $f'(x)$ та другу $f''(x)$ похідні і нехай $[a;b]$ – інтервал локального кореня. На інтервалі $[a;b]$ функцію *f(x)* розкладемо у ряд Тейлора, обмежившись лише лінійним членом.

$$f(x) \approx f(x_k) + f'(x_k)\Delta x + \frac{1}{2} f''(\theta_k)\Delta^2 x, \qquad (5.7)$$

де $x_k \in [a;b]$; $\theta_k \in [a;b]$; $x = x_k + \Delta x$;

$R_2(x) = \frac{1}{2} f''(\theta_k)\Delta^2 x$ - залишковий член ряду;

Δx - приріст аргументу.

Оскільки $\Delta x = x - x_k$, то рівняння (5.7) набуде такого вигляду:

$$f(x) \approx f(x_k) + f'(x_k)(x - x_k) + \frac{1}{2} f''(\theta_k)(x - x_k)^2 \qquad (5.8).$$

Рівняння (5.8) справедливе для будь-якого x із інтервалу $[a;b]$. Справедливо і для $x = c$

$$f(c) = f(x_k) + f'(x_k)(c - x_k) + \frac{1}{2} f''(\theta_k)(c - x_k)^2.$$

Але $f(c) = 0$. Тому

$$f(x_k) + f'(x_k)(c - x_k) + \frac{1}{2} f''(\theta_k)(c - x_k)^2 = 0. \qquad (5.9)$$

У тому випадку, коли c близьке до x_k, тоді у (5.9) можна знехтувати квадратним членом, але у такому випадку будемо мати не корінь c, а деяку іншу точку x_{k+1}. Отже,

$$f(x_k) + f'(x_k)(x_{k+1} - x_k) = 0. \qquad (5.10)$$

Із останнього рівняння знаходимо вираз

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \qquad (5.11)$$

який і визначає ітераційний процес Ньютона-Рафсона.

Із рівняння (5.10) можна утворити деяку функцію шляхом заміни x_{k+1} на x

$$y = f(x_k) + f'(x_k)(x - x_k).$$

Це рівняння прямої лінії, яка є дотичною до функції $f(x)$ у точці $(x_k; f(x_k))$. Звідси випливає і геометричний зміст методу Ньютона-Рафсона: наближення до кореня c функції $f(x)$ здійснюється за абсцисами точок перетинів дотичних до $f(x)$, які проведені у точках попередніх наближень (рис 5.6).

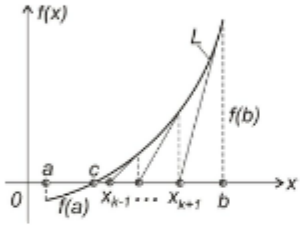


Рисунок 5.6 – Наближення до кореня функції $f(x)$ методом Ньютона-Рафсона

Геометричне тлумачення методу Ньютона-Рафсона дало йому другу назву – *метод дотичних.*

У тому випадку, коли перша похідна обмежена знизу $|f'(x)| \geq \alpha > 0$, а друга похідна - $|f''(x)| \leq \beta < \infty$, можна знайти апостеріорну оцінку похибки методу Ньютона-Рафсона

$$|c - x_{k+1}| \leq \frac{\beta}{2\alpha} |x_{k+1} - x_k|^2, \text{ де } c, x_k \in [a;b]. \qquad (5.12)$$

При реалізації ітераційної процедури (5.11) виникає питання про вибір початкової точки x_0, яка за вимогою методу, повинна бути близькою до кореня c функції $f(c)$. У тому випадку, коли похідні $f'(x)$ і $f''(x)$ на інтервалі $[a;b]$ мають постійні знаки, $f(a) \cdot f(b) < 0$ і стартова точка $x_0 \in [a;b]$ вибрана так, що $f(x_0) f''(x_0) > 0$, то починаючи з неї послідовність $\{x_k\}$, яка визначена процедурою (5.11), монотонно збігається до кореня c функції $f(x)$.

Метод січних. У методі Ньютона-Рафсона для реалізації ітераційної процедури необхідно обчислювати дві функції $f(x_{k-1})$ і $f'(x_{k-1})$ на кожному k-му кроці. Для простих функцій обчислення $f'(x)$ не викликає особливих труднощів, У тих випадках, коли $f(x)$ мають форму інтегралів, сум і т.д. бажано мати метод, який збігається майже так швидко, як і метод Ньютона-Рафсона, але не вимагає інформації про значення похідної $f'(x)$.

У методі *січних* похідна обчислюється тільки один раз в точці $x = x_0$. У подальшому нахил прямої L (рис 5.8) до осі абсцис залишається незмінним, тобто

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_0)}. \qquad (5.13)$$

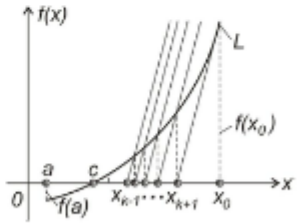


Рисунок 5.7 – Наближення до кореня *c* методом січних

Ітераційна процедура (5.13) втрачає високу швидкість збіжності і замість квадратичної (5.12) має лише швидкість

збіжності геометричної прогресії. Зменшення швидкості збіжності процедури (5.13) пояснюється тим, що процес не реагує на зміну нахилу кривої при наближенні x_k до c. Збільшити швидкість збіжності можна, якщо на кожному кроці процедури (5.11) похідну замінити її наближенням

$$f'(x_k) \approx \frac{f(x_{k+1}) - f(x_k)}{x_{k+1} - x_k}.$$

Оскільки $x_{k+1} = x_k + h_k$, то

$$f'(x_k) \approx \frac{f(x_k + h_k) - f(x_k)}{h_k}, \qquad (5.14)$$

Підставляючи (5.14) у (5.11), отримують процедуру

$$x_{k+1} = x_k - \frac{f(x_k)h_k}{f(x_k + h_k) - f(x_k)}, \qquad (5.15)$$

яку називають *різницевим методом Ньютона.*

Якщо у (5.15) h_k покласти рівним: $h_k = x_{k-1} - x_k$, то отримаємо

$$x_{k+1} = x_k - \frac{f(x_k)(x_{k-1} - x_k)}{f(x_{k-1}) - f(x_k)}. \qquad (5.16)$$

Ітераційна процедура (5.16) носить назву *модифікованого методу січних.* Цей метод на k-ому кроці наближення дає похибку, яка обчислюється за такою формулою:

$$|c - x_k| < c_1 \nu^{s^k},$$

де $c_1 = \frac{2\alpha}{3\beta}$, $\nu = \frac{3\beta}{2\alpha}|c - x_0|$, $s = \frac{1+\sqrt{5}}{2} \approx 1.618$.

Високий порядок швидкості збіжності ітераційної процедури (5.16) у поєднані з невеликими обчислювальними затратами виводять модифікований методом січних за ефективністю вирішення задачі $f(x) = 0$ на перше місце, що підтверджується як теоретичними, так і практичними висновками.

5.2 Методи розв'язку систем нелінійних рівнянь

Припустимо, що необхідно розв'язати систему нелінійних рівнянь.

$$f_i(x^{(1)}, x^{(2)}, \ldots, x^{(n)}) = 0, \quad i = \overline{1,n}, \qquad (5.17)$$

де $f_i(x^{(1)}, x^{(2)}, \ldots, x^{(n)})$, відомі дійсні нелінійні функції своїх змінних $x^{(i)}$, $i = \overline{1,n}$, які є також дійсними.

Рівняння (5.17) подано у матрично векторній формі:

$$\bar{F}(\bar{x}) = \bar{0}, \qquad (5.18)$$

де $\bar{F}(\bar{x}) = \begin{bmatrix} f_1(x^{(1)}, x^{(2)}, \ldots, x^{(n)}) \\ f_2(x^{(1)}, x^{(2)}, \ldots, x^{(n)}) \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ f_n(x^{(1)}, x^{(2)}, \ldots, x^{(n)}) \end{bmatrix} = \begin{bmatrix} f_1(\bar{x}) \\ f_2(\bar{x}) \\ \ldots\ldots \\ f_n(\bar{x}) \end{bmatrix}$ - n - вимірна вектор функція; $\bar{x}^T = \left(x^{(1)}, x^{(2)}, \ldots, x^{(n)}\right)$ - n - вимірний вектор змінних $x^{(i)}, i = \overline{1,n}$ $\bar{0}$ - n - вимірний нуль-вектор.

Припустимо, що рівняння (5.18) можна записати у такому вигляді:

$$\bar{x} = \bar{\Phi}(\bar{x}), \qquad (5.19)$$

де $\bar{\Phi}(\bar{x})$ - деяка вектор-функція.

На основі (5.19) можна отримати формальну рекурентну рівність

$$\bar{x}_{k+1} = \bar{\Phi}(\bar{x}_k), \qquad (5.20)$$

яка визначає *метод простих ітерацій (МПІ).*

За певних умов* (див. с. 283-285) ітераційний процес буде збіжним, тобто

$$\lim_{k\to\infty} \bar{x}_k = \bar{c}, \qquad (5.21)$$

де $\bar{c}$ -вектор, який є розв'язком рівняння (5.18).

Щоб метод простих ітерацій застосувати до розв'язку будь-якого нелінійного рівняння, яке приведено до (5.18), необхідно здійснити наступне формальне перетворення. Помножимо рівняння (5.18) на деяку невласну $n \times n$ - матрицю A, взяту зі знаком мінус: $-A\bar{F}(\bar{x}) = \bar{0}$.

Тепер до лівої і правої частини додамо вектор невідомих $\bar{x}$. У результаті отримаємо

$$\bar{x} = \bar{x} - A\bar{F}(\bar{x}). \qquad (5.22)$$

Проблема полягає в підборі елементів матриці A таких, щоб ітераційний процес

$$\bar{x}_{k+1} = \bar{x}_k - A_k \bar{F}(\bar{x}_k) \qquad (5.23)$$

* Вержбицкий В. М. Основы численных методов. Учебник. – М.: Высш. шк., 2002. – 840 с.

був збіжним, тобто виконувалася умова (5.21). Залежно від вибору елементів матриці A_k отримаємо різні ітераційні процедури пошуку коренів системи нелінійних рівнянь (5.17)

Метод Ньютона. Якщо $A_k = \left[\bar{F}'(\bar{x}_k)\right]^{-1}$

де $\bar{F}'(\bar{x}_k) = \begin{bmatrix} \frac{\partial f_1(\bar{x})}{\partial x^{(1)}} & \frac{\partial f_1(\bar{x})}{\partial x^{(2)}} & \dots & \frac{\partial f_1(\bar{x})}{\partial x^{(n)}} \\ \frac{\partial f_2(\bar{x})}{\partial x^{(1)}} & \frac{\partial f_2(\bar{x})}{\partial x^{(2)}} & \dots & \frac{\partial f_2(\bar{x})}{\partial x^{(n)}} \\ \dots & \dots & \dots & \dots \\ \frac{\partial f_n(\bar{x})}{\partial x^{(1)}} & \frac{\partial f_n(\bar{x})}{\partial x^{(2)}} & \dots & \frac{\partial f_n(\bar{x})}{\partial x^{(n)}} \end{bmatrix}_{\bar{x}=\bar{x}_k}$ - матриця

Якобі, то

$$\bar{x}_{k+1} = \bar{x}_k - \left[\bar{F}'(\bar{x}_k)\right]^{-1} \bar{F}(\bar{x}_k). \qquad (5.24)$$

Формула (5.24) є прямим аналогом метода Ньютона для скалярної функції $f(x)$, де похідна $f'(x)$ замінена на матрицю обернену до матриці Якобі.

Модифікований метод Ньютона. Метод Ньютона вимагає на кожній ітерації знаходження оберненої матриці $\left[\bar{F}'(\bar{x}_k)\right]^{-1}$, що значно збільшує обчислювальні затрати. Ці затрати можна зменшити, якщо матрицю $\left[\bar{F}'(\bar{x})\right]^{-1}$ обчислювати лише у початковій точці $\bar{x}_0$. У результаті отримаємо модифікований метод Ньютона

$$\bar{x}_{k+1} = \bar{x}_k - \left[\bar{F}'(\bar{x}_0)\right]^{-1} \bar{F}(\bar{x}_k). \qquad (5.25)$$

З іншої сторони реалізація модифікованого методу Ньютона може привести до збільшення кількості ітерацій для досягнення заданої точності розв'язку задачі (5.18).

Дискретний метод Ньютона. На базі метода Ньютона можна отримати близький до нього ітераційний процес, який не вимагає обчислення похідних. Досягти цього можна, замінивши в матриці Якобі часткові похідні різницевим відношенням

$$\frac{\partial f_i(\bar{x}_k)}{\partial x^{(j)}} = \frac{f_i(x_k^{(1)}, x_k^{(2)}, \ldots, x_k^{(j)} + h_j, \ldots, x_k^{(n)}) - f_i(x_k^{(1)}, x_k^{(2)}, \ldots, x_k^{(j)}, \ldots x_k^{(n)})}{h_j},$$

де $i, j = \overline{1,n}$.

Тоді $\bar{F}'(\bar{x}_k) \approx J(\bar{x}_k, \bar{h}_k)$, де $J(\bar{x}_k, \bar{h}_k)$- матриця Якобі, в якій часткові похідні замінені кінцевими відношеннями, що приводить до *дискретного методу Ньютона*

$$\bar{x}_{k+1} = \bar{x}_k - \left[J(\bar{x}_k, \bar{h}_k)\right]^{-1} F(\bar{x}_k). \qquad (5.26)$$

Якщо взяти $h_k^{(j)} = x_{k-1}^{(j)} - x_k^{(j)}$, то отримаємо *метод січних*, який узагальнений на багатовимірну задачу (5.13).

Контрольні питання та завдання

1 Які методи застосовують для розв'язку нелінійних алгебраїчних рівнянь?

2 Як знайти інтервал, на якому функція $f(x)$ дорівнює нулю?

3 Чи означає умова $f(a)f(b) > 0$, що рівняння $f(x) = 0$ не має жодного кореня на інтервалі $I = [a;b]$?

4 Чи означає умова $f(a)f(b) < 0$, що рівняння $f(x) = 0$ має єдиний корінь на інтервалі $I = [a;b]$?

5 Які переваги і недоліки методу дихотомії над іншими числовими методами розв'язку рівняння $f(x) = 0$?

6 У якому випадку метод хорд збігається швидше у порівнянні з методом дихотомії?

7 Завдяки чому забезпечується вища збіжність методу Ньютона-Рафсона у порівнянні з методами дихотомії і хорд?

8. Яким чином оцінити похибку розв'язку рівняння $f(x) = 0$ методом Ньютона-Рафсона?

9 Чим пояснити появу модифікацій методу Ньютона-Рафсона?

10 Використовуючи програми процедуру `MethodChords`, знайти локальний корінь рівняння $f(x) = xe^{-0.5} - 0.8\sin x$.

11 Охарактеризуйте методи розв'язку систем нелінійних алгебраїчних рівнянь.

6 Розв'язання диференціальних рівнянь

6.1 Методи розв'язку диференціальних рівнянь

В інженерній практиці часто доводиться досліджувати динаміку тих чи інших технічних об'єктові. Для цього створюють математичні моделі, які у більшості випадків мають вигляд диференціальних рівнянь. Для отримання відомостей про властивості технічних об'єктів необхідно знати розв'язки їх моделей. Отримати аналітичний розв'язок таких моделей можна тільки в окремих випадках. У переважній більшості для розв'язку диференціальних рівнянь застосовують числові методи.

Метод Ейлера. Він має обмежене застосування із-за великої похибки, яка накопичується в процесі обчислень.

Отже, будемо розглядати звичайне диференціальне рівняння першого порядку:

$$\frac{dy}{dt} = f(t, y). \qquad (6.1)$$

З початковою умовою

$$y(t_0) = y_0, \qquad (6.2)$$

де $f(t, y)$ - деяка відома у загальному випадку, нелінійна функція двох аргументів.

Будемо вважати, що для даної задачі (6.1), (6.2), яка носить назву задачі Коші, виконуються вимоги, які забезпечують існування і єдиність на відрізку $[t_0, t_f]$ її розв'язку $y = y(t)$. Допустимо, що $y(t)$, $y'(t)$ і $y''(t)$ неперервні. Використовуючи теорему Тейлора, розкладемо функцію $y(t)$ у ряд Тейлора в околі точки $t = t_k$. Для кожного значення t існує таке ξ, яке лежить між t_k і t, що $y(t) = y(t_k) + y'(t_k)(t - t_k) + y''(\xi)\frac{(t - t_k)^2}{2}$. У відповідності з (6.1) $y'(t_k) = f(t_k, y(t_k))$. Тому

$$y(t) = y(t_k) + f(t_k, y(t_k))(t - t_k) + y''(\xi_k)\frac{(t - t_k)^2}{2}.$$

Нехай $t = t_{k+1}$ і $t_{k+1} - t_k = h = const$, а ξ_k - точка, яка лежить між t_k і t_{k+1}. Тоді

$$y(t_{k+1}) = y(t_k) + hf(t_k, y(t_k)) + y''(\xi_k)\frac{h^2}{2}.$$

Якщо довжина кроку h вибрана досить малою, то членом другого порядку можна знехтувати і отримати

$$y_{k+1} = y_k + hf(t_k, y_k), \qquad (6.3)$$

де $y_{k+1} = y(t_{k+1})$; $y_k = y(t_k)$.

Ітераційна процедура (6.3) і є наближенням Ейлера для задачі (6.1), (6.2) при цьому $y_0 = y(t_0)$.

Як випливає із формули (6.3) кожна наступна ордината, починаючи із y_0, обчислюється шляхом додавання до неї площі прямокутника $hf(t_k, y_k)$ (рис. 6.1).

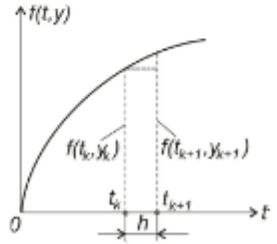


Рисунок 6.1 – Процес утворення прямокутників в ітераційній процедурі (6.3)

Виводячи ітераційну формулу Ейлера для кожного кроку обчислень ми нехтували величиною $y''(\xi_k)\frac{h^2}{2}$, що приводить до накопичування похибки і після N кроків вона складе величину

$$\sum_{k=1}^{N} y''(\xi_k)\frac{h^2}{2} \approx Ny''(\xi)\frac{h^2}{2} \le (t_f - t_0)y''(\xi)\frac{h}{2} = O(h^1),$$

де $N = \frac{t_f - t_0}{h}$; $[t_0; t_f]$ - інтервал, на якому шукають розв'язок задачі Коші;

$O(h^1)$ - похибка першого порядку.

Для розв'язку диференціального рівняння (6.1) з початковою умовою (6.2) методом Ейлера може бути використана процедура `MethodEuler`.

```
MethodEuler*
▷Числовий розв'язок диференціального рівняння методом
ейлера
▷Вхід:n-кількість вузлів дискретності
▷      t0-початкове значення часу
▷      tk-кінцеве значення часу
▷      y0- початкова умова
▷Вихід:y(k)-значення ординат функції у вузлах
дискретизації
▷       t-вектор абсцис
▷ Значення функції f(t,y) рівняння (6.1) обчислюється
підпроцедурою - fun_Eiler.
▷Обчислення кроку дискретизації
1 h=(tk-t0)/n
▷Числовий розв'язок задачі
2 СФормувати нульовий вектор у розміром n+1
3 y(1)=y0
4 for k=1 to n
5     y_k=fun_Eiler(y(k))
6     y(k+1)=y(k)+h*y_k
7 end for
8 t(1)=t0
9 for i=2 to n
10    t(i)= y0+(i-1)*h
11 end for
12 return t,y
```

Метод Гюна. Проінтегруємо ліву і праву частини рівняння (6.1) в межах від t_0 до t_1:

$$\int_{t_0}^{t_1} f(x, y(x))dx = y(t_1) - y(t_0). \qquad (6.4)$$

Із рівняння (6.4) знайдемо

$$y(t_1) = y(t_0) + \int_{t_0}^{t_1} f(x, y(x))dx. \qquad (6.5)$$

Інтеграл, який знаходиться у правій частині рівняння (6.5), можна наближено обчислити за формулою трапецій (рис. 6.1).

$$y(t_1) \approx y(t_0) + \frac{h}{2}(f(t_0, y(t_0)) + f(t_1, y(t_1))). \qquad (6.6)$$

Права частина рівняння (6.6) вимагає значення $y(t_1)$, яке знайдемо, скориставшись методом Ейлера. Із формули (6.3) для $k = 0$ отримаємо

$$y(t_1) = y(t_0) + hf(t_0, y(t_0)).$$

З врахуванням значення $y(t_1)$ формула (6.1) набуде такого вигляду:

$$y(t_1) = y(t_0) + \frac{h}{2}\left(f\left(t_0, y(t_0)\right) + f\left(t_1, y(t_0) + hf\left(t_0, y(t_0)\right)\right)\right).$$

Тепер, зробивши заміну $t_2 \leftarrow t_1$, $t_1 \leftarrow t_0$, генеруємо процес розв'язку задачі Коші (6.1).

Для k-го кроку обчислень будемо мати

$$p_{k+1} = y_k + hf(t_k, y_k),\ t_{k+1} = t_k + h, \tag{6.7}$$

$$y_{k+1} = y_k + \frac{h}{2}\left(f\left(t_k, y_k\right) + f\left(t_{k+1}, p_{k+1}\right)\right), \tag{6.8}$$

$k = 0, 1, 2, \ldots$.

Залишковий член у формулі трапецій, який використовують для наближення інтервалу в (6.6), дорівнює

$$-y''(\xi_k)\frac{h^3}{12}. \tag{6.9}$$

У процесі обчислень на кожному кроці накопичується похибка (6.9) і після N кроків накопичень похибка методу Гюна буде такою:

$$-\sum_{k=1}^{N} y''(\xi_k)\frac{h^3}{12} \le \frac{t_f - t_0}{12} y''(\xi)h^2 = O\left(h^2\right),$$

тобто метод Гюна має другий порядок точності $0(h^2)$. Нижче наведена процедура `MethodHeun`, за допомогою якої можна розв'язати диференціальне рівняння (6.1) з початковою умовою (6.2) методом Гюна.

```
MethodHeun*
⊳Метод Гюна розв'язку задачі Коші y'=f(t,y)
⊳Вхід:f(t,y)-функція,    значення    якої    задається
підпроцедурою fun_Heun
⊳      to,tk-початкова і кінцева точки інтегрування
⊳      y0-початкова умова
⊳      n-число ітерацій
⊳Вихід:t-вектор абсцис
⊳       y-вектор ординат
⊳Обсилюємо крок h та ініціалізуємо вектори t,y
1 h=(tk-t0)/n
2 СФормувати нульовий вектор y розміром n+1
⊳Ітераційна процедура методу Гюна
3 y(1)=y0
4 t(1)=t0
5 for i=2 to n
6    t(i)=t0+(i-1)*h
7 end for
8 for k=1 to n
9     k1=fun_Heun(t(k),y(k))
10     k2=fun_Heun(t(k+1),y(k)+h*k1)
11     y(k+1)=y(k)+(h/2)*(k1+k2)
12 end for
13 return t,y
```

Метод Рунге-Кутта.

Ідея побудови методів Рунге-Кутта p-го порядку полягає в отриманні наближеного числового розв'язку задачі Коші за формулою

$$y_{k+1} = y_k + h\varphi\left(t_k, y_k, h\right), \tag{6.10}$$

де $\varphi(t, y, h)$ - деяка функція, яка наближує частину ряду Тейлора до p-го порядку і не вміщує часткових похідних функції $f(t, y)$.

Якщо в (6.10) замінити $\varphi(t, y, h)$ на $f(t, y)$, то отримаємо метод Ейлера, тобто метод Ейлера можна вважати частковим випадком методів Рунге-Кутта, коли $p = 1$.

Для побудови методів Рунге-Кутта вище першого порядку функцію $\varphi(t, y, h)$ вибирають як таку, що залежить від певних параметрів, які вибирають шляхом порівняння виразу (6.10) з многочленом Тейлора для $y(t)$ з бажаним порядком степеня.

Розглянемо випадок $p = 2$ і візьмемо функцію $\varphi(t, y, h)$ з наступною структурою:

$$\varphi(t, y, h) = c_1 f(t, y) + c_2 f(t + ah, y + bhf(t, y)) \qquad (6.11)$$

Розкладемо функцію двох змінних $f(x + ha, y + bhf(t, y))$ в ряд Тейлора, обмежившись лише лінійними членами

$$f(t+ah, y+bhf(t,y)) = f(t,y) + \frac{\partial f(t,y)}{\partial t} ah + \frac{\partial f(t,y)}{\partial y} bhf(t,y) + O(h^2) \qquad (6.12)$$

У (6.11) функцію $f(x + ha, y + bhf(t, y))$ замінимо її наближеним значенням (6.12). У результаті будемо мати

$$\varphi(t, y, h) = c_1 f(t, y) + c_2 \left(f(t, y) + f_t'(t, y) ah + f_y'(t, y) bhf(t, y) \right) + O(h^2),$$

де $f_t'(t, y) = \dfrac{\partial f(t, y)}{\partial t}$, $f_y' = \dfrac{\partial f(t, y)}{\partial y}$.

Підставивши останній вираз в (6.10), отримаємо

$$y_{k+1} = y_k + h\left((c_1 + c_2) f(t_k, y_k) + h\left(c_2 a f_t'(t_k, y_k) + c_2 b f_y'(t_k, y_k) f(t_k, y_k) \right) \right) + O(h^3). \qquad (6.13)$$

Розкладемо тепер функцію $y(t_k + h)$ в ряд Тейлора, враховуючи члени першого і другого порядків

$$y(t_k + h) = y(t_k) + hy'(t_k) + \frac{h^2}{2} y''(t_k) + O(h^3). \qquad (6.14)$$

Оскільки $\dfrac{dy}{dt} = y'(t) = f(t, y)$, то $y'(t_k) = f(t_k, y_k)$.

Знайдемо другу похідну $y''(t) = (f(t, y))'$. За формулою повної похідної будемо мати

$$(f(t, y))' = \frac{\partial f(t, y)}{\partial t} + \frac{\partial f(t, y)}{\partial y} \cdot \frac{dy}{dt}.$$

Враховуючи значення $\dfrac{dy}{dt}$, отримаємо

$$(f(t, y))' = f_t'(t, y) + f_y'(t, y) f(t, y).$$

Отже,

$$y''(t_k) = f_t'(t_k, y_k) + f_y'(t_k, y_k) f(t, y).$$

Тепер значення $y'(t_k)$ і $y''(t_k)$ можемо підставити у вираз (6.14). У результаті отримаємо

$$y_{k+1} = y_k + hf(t_k, y_k) + \frac{h^2}{2} \left(f_t'(t_k, y_k) + f_y'(t_k, y_k) f(t_k, y_k) \right) + O(h^3). \qquad (6.15)$$

Порівнюючи між собою вирази (6.13) і (6.15), приходимо до висновку, що

$$c_1 + c_2 = 1,\ c_2 a = \frac{1}{2},\ c_2 b = \frac{1}{2}.$$

Отримана система із трьох рівнянь, яка вміщує чотири невідомих. Це означає, що один із параметрів вільний і його можна вибрати довільним. Візьмемо $c_2 = \alpha$, $\alpha \neq 0$. Тоді

$$c_1 = 1 - \alpha,\ a = b = \frac{1}{2\alpha}.$$

У результаті підставлення значень c_1, c_2, a і b у формулу (6.11) отримаємо:

$$\varphi(t_k, y_k, h) = (1-\alpha) f(t_k, y_k) + \alpha f\left(t_k + \frac{h}{2\alpha}, y_k + \frac{h}{2\alpha} f(t_k, y_k)\right).$$

Отриманий результат дає підставу ітераційну процедуру (6.10) записати у такому вигляді:

$$y_{k+1} = y_k + h\left[(1-\alpha) f(t_k, y_k) + \alpha f\left(t_k + \frac{h}{2\alpha}, y_k + \frac{h}{2\alpha} f(t_k, y_k)\right)\right]. \quad (6.16)$$

Якщо в (6.16) $\alpha = \frac{1}{2}$, то

$$y_{k+1} = y_k + \frac{h}{2}\left(f(t_k, y_k) + f(t_k + h, y_k + hf(t_k y_k))\right), \quad (6.17)$$

а при $\alpha = 1$ маємо

$$y_{k+1} = y_k + hf\left(t_k + \frac{h}{2}, y_k + \frac{h}{2} f(t_k, y_k)\right). \quad (6.18)$$

Ітераційна процедура (6.17) носить назву *метода Хойна*, а (6.18) – породжує метод *середньої точки*.

Аналіз методів Рунге-Кутта другого порядку дає уявлення, в якій формі слід шукати метод Рунге-Кутта довільного порядку.

За аналогією з (6.18) можемо записати

$$K_1^{(k)} = f(t_k, y_k),$$

$$K_j^{(k)} = f\left(t_k + a_j h, y_k + h\sum_{r=1}^{j-1} b_{jr} K_r^{(k)}\right),\ j = 2,3,\ldots,p, \quad (6.19)$$

$$y_{k+1} = y_k + h\sum_{j=1}^{p} c_j K_j^{(k)}.$$

Найпоширенішим із сімейства методів (6.17) є метод четвертого порядку (p = 4) або просто метод *Рунге-Кутта*, який породжує таку ітераційну процедуру:

$$f_1^{(k)} = f(t_k, y_k),$$

$$f_2^{(k)} = f\left(t_k + \frac{h}{2}, y_k + \frac{h}{2} f_1^{(k)}\right),$$

$$f_3^{(k)} = f\left(t_k + \frac{h}{2}, y_k + \frac{h}{2} f_2^{(k)}\right), \quad (6.20)$$

$$f_4^{(k)} = f\left(t_k + h, y_k + hf_3^{(k)}\right),$$

$$y_{k+1} = y_k + \frac{h}{6}\left(f_1^{(k)} + 2f_2^{(k)} + 2f_3^{(k)} + f_4^{(k)}\right).$$

Метод Рунге-Кутта дає похибку накопичення четвертого порядку - $O(h^4)$.

Бажання підвищити точність методу Рунге-Кутта призвело до появи різних його версій. Одна із них – метод *Кутта-Мерсона* з вибором кроку на кожній із ітерацій.

На k-му кроці розв'язку задачі Коші послідовно обчислюють:

$$f_1^{(k)} = f(t_k, y_k),$$

$$f_2^{(k)} = f\left(t_k + \frac{h}{3}, y_k + \frac{h}{3} f_1^{(k)}\right),$$

$$f_3^{(k)} = f\left(t_k + \frac{h}{3}, y_k + \frac{h}{6} f_1^{(k)} + \frac{h}{6} f_2^{(k)}\right), \qquad (6.21)$$

$$f_4^{(k)} = f\left(t_k + \frac{h}{2}, y_k + \frac{h}{8} f_1^{(k)} + \frac{3}{8} h f_2^{(k)}\right),$$

$$f_5^{(k)} = f\left(t_k + h, y_k + \frac{h}{2} f_1^{(k)} - \frac{3}{2} h f_3^{(k)} + 2h f_4^{(k)}\right),$$

$$\tilde{y}_{k+1} = y_k + \frac{h}{2}\left(f_1^{(k)} - 3 f_3^{(k)} + 4 f_4^{(k)}\right),$$

$$y_{k+1} = y_k + \frac{h}{6}\left(f_1^{(k)} + 4 f_4^{(k)} + f_5^{(k)}\right).$$

Після цього підраховують всличину $R = 0{,}2\left|y_{k+1} - \tilde{y}_{k+1}\right|$, якщо $R \le \varepsilon$, то вважають, що в точці $t = t_k$ отриманий розв'язок задачі Коші з точністю ε. У тому випадку, коли $R > \varepsilon$ крок h зменшують вдвічі $\left(h \leftarrow \frac{h}{2}\right)$ і знову обчислюють значення $f_2^{(k)}, f_3^{(k)}, f_4^{(k)}, f_5^{(k)}$.

При переході до наступного кроку $k+1$ здійснюється перевірка на можливість збільшення кроку. Якщо $R \le \frac{\varepsilon}{64}$, то $h \leftarrow 2h$.

Процедура `MethodRunge-Kutt` дає змогу розв'язати диференціальне рівняння (6.1) з початковою умовою (6.2) методом Рунге-Кутта.

```
MethodRunge-Kutt*
▷ %Метод Рунге-Кутта розв'язку y'=f(t,y) задачі Коші
на
   інтервалі [t0tk] з початковою умовою y(0)=y0
▷ Вхід:f(t,y)-функція,   значення   якої   задати   у
підпроцедурі fun_RK
▷      to,tk-початкова і кінцева точки інтегрування
▷      y0-початкова умова
▷      n-число ітерацій
▷ Вихід:t-вектор абсцис
▷       y-вектор ординат
▷ Обсилаємо крок h та ініціалізуємо вектор y
1 h=(tk-t0)/n
2 СФормувати нульовий вектор y розміром n+1
▷ Ітераційна процедура методу Рунге-Кутта
3 y(1)=y0
4 t(1)=t0
5 for i=2 to n
6    t(i)=t0+(i-1)*h
7 end for
8 for k=1 to n
9    f1=fun_RK(t(k),y(k))
10    f2=fun_RK(t(k)+h/2,y(k)+h*f1/2)
11    f3=fun_RK(t(k)+h/2,y(k)+h*f2/2)
12    f4=fun_RK(t(k)+h,y(k)+h*f3)
13    y(k+1)=y(k)+(f1+2*f2+2*f3+f4)*h/6
14 end for
15 return t,y
```

6.2 Системи диференціальних рівнянь

Нехай необхідно знайти розв'язок задачі Коші для системи звичайних диференціальних рівнянь першого порядку, які записані у стандартній формі

$$\begin{aligned}
y_1' &= f_1(t, y_1, y_2, \ldots, y_n),\ y_1(t_0) = y_1^{(0)},\\
y_2' &= f_2(t, y_1, y_2, \ldots, y_n),\ y_2(t_0) = y_2^{(0)}, \qquad (6.25)\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
y_n' &= f_n(t, y_1, y_2, \ldots, y_n),\ y_n(t_0) = y_n^{(0)}.
\end{aligned}$$

Якщо ввести векторні позначення

$$\overline{Y} = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix}, \quad \overline{Y}' = \begin{pmatrix} y_1' \\ y_2' \\ \dots \\ y_n' \end{pmatrix}, \quad \overline{F}(t,\overline{Y}) = \begin{pmatrix} f_1(t,\overline{Y}) \\ f_2(t,\overline{Y}) \\ \dots \\ f_n(t,\overline{Y}) \end{pmatrix}, \quad \overline{Y}^{(0)} = \begin{pmatrix} y_1^{(0)} \\ y_2^{(0)} \\ \dots \\ y_n^{(0)} \end{pmatrix},$$

то задачу Коші (6.25) можна записати у такому компактному вигляді:

$$\overline{Y}' = \overline{F}(t,\overline{Y}), \ \overline{Y}(t_0) = \overline{Y}^{(0)}. \qquad (6.26)$$

Векторне рівняння (6.26) за своєю структурою аналогічне скалярному рівнянню (6.1). Це означає, що для задачі (6.26), яка подана у вигляді векторного рівняння, у принципі, можна застосувати будь-який числовий метод розв'язку звичайних диференціальних рівнянь, який розглядався раніше. При цьому скалярним величинам у формулах, які визначені відповідними методами, скалярними величинами є тільки змінна t та розрахунковий крок h; всім іншим величинам відповідають вектори розмірності n. Необхідно лише врахувати, що при контролі за точністю розв'язку замість умови (6.21) потрібно використовувати аналогічну умову, в якій замість модуля треба взяти норму відповідного вектора, наприклад, норму-максимум.

Для вектора $\overline{a}$ з компонентами a_1, a_2, …, a_n норма-вектор буде такою:

$$|\overline{a}|_{\max} = \max\{|a_1|, |a_2|, \dots, |a_n|\}. \qquad (6.27)$$

6.3 Розв'язування диференціальних рівнянь вищих порядків

У загальному вигляді необхідно розв'язати диференціальне рівняння n - го порядку

$$y^{(n)} = f\left(t, y, y', \dots, y^{(n-1)}\right) \qquad (6.28)$$

з такими початковим умовами:

$y = y_0$, $y' = y_0'$, …, $y^{(n-1)} = y_0^{(n-1)}$ при $t = t_0$.

Позначимо $x_1 = y$, $\frac{dx_1}{dt} = x_2$, $\frac{dx_2}{dt} = x_3$, …, $\frac{dx_{n-1}}{dt} = x_n$.

Звідси випливає, що $y' = x_2$, $y'' = x_3$, …, $y^{(n-1)} = x_n$ і $\frac{dx_n}{dx} = f\left(t, x_1, x_2, \dots, x_n\right)$, і рівняння (6.28) n - го порядку перетворюється в систему із n рівнянь кожне із яких є рівнянням першого порядку

$$\begin{aligned} \frac{dx_1}{dt} &= x_2, \\ \frac{dx_2}{dt} &= x_3, \\ &\dots \\ \frac{dx_{n-1}}{dt} &= x_n, \end{aligned} \qquad (6.29)$$

$$\frac{dx_n}{dt} = f\left(t, x_1, x_2, \dots, x_n\right), \qquad y = x_1$$

з початковими умовами $x_1 = y_0$, $x_2 = y_0'$, …, $x_n = y_0^{(n-1)}$ при $t = t_0$.

Таким чином, шляхом введення допоміжних змінних $x_1, x_2, \ldots, x_n$ отримали систему рівнянь (6.29), яка аналогічна системі рівнянь (6.25).

Якщо тепер систему рівнянь (6.29) записати у векторному вигляді

$$\frac{d\overline{x}}{dt} = \overline{f}\left(t, \overline{x}\right)$$

з початковими умовами $\overline{x}\left(t_0\right) = \overline{x}^{(0)}$,

де $\overline{x} = \begin{bmatrix} x_1 \\ x_2 \\ \ldots \\ x_n \end{bmatrix}$; $\overline{f}\left(t, \overline{x}\right) = \begin{bmatrix} x_2 \\ x_3 \\ \ldots \\ f\left(t, \overline{x}\right) \end{bmatrix}$; $\overline{x}\left(t_0\right) = \begin{bmatrix} y_0 \\ y_0' \\ \ldots \\ y_0^{(n-1)} \end{bmatrix}$, то задача

знаходження розв'язку диференціального рівняння (6.28) є аналогічною задачі (6.26).

Приклад 6.1. Знайти напругу U_2 на виході електричної ланки (рис 6.2) після подачі на її вхід напруги U_1, якщо $R = 10\,Ом$, $L = 0{,}1\ Г$, $C = 1000\ мкФ$, $u_2 = 100\ В$. Припускаємо, що у початковий момент часу $t_0 = 0$ конденсатор C не заряджений.

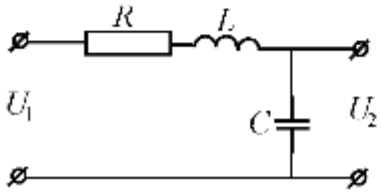


Рисунок 6.2 – Схема електричної ланки

Відповідно до другого закону Кірхгофа

$$U_1 = iR + L\frac{di}{dt} + U_2\,.$$

Враховуючи те, що $i = C\frac{dU_2}{dt}$, отримуємо

$$LC\frac{d^2U_2}{dt^2} + CR\frac{dU_2}{dt} + U_2 = U_1\,. \qquad (6.30)$$

За умовою задачі $U_2\left(0\right) = 0$, $U_2'\left(0\right) = 0$.

Введемо такі позначення: $y = U_2$, $u = U_1$, $a_2 = LC$, $a_1 = CR$. Тоді рівняння (6.30) набуде такого вигляду:

$$a_2\frac{d^2y}{dt^2} + a_1\frac{dy}{dt} + y = u\,. \qquad (6.31)$$

Відповідно до прийнятих позначень початкові умови будуть такими: $y\left(0\right) = 0$, $y'\left(0\right) = 0$.

Отримане рівняння (6.31) подамо у формі (6.28)

$$\frac{d^2y}{dt^2} = \frac{1}{a_2}\left(-a_1\frac{dy}{dt} - y + u\right).$$

Вводимо допоміжні змінні - $x_1 = y$, $x_2 = \frac{dx_1}{dt}$. Тоді

$$\frac{dx_1}{dt} = x_2,$$

$$\frac{dx_2}{dt} = \frac{1}{a_2}\left(-x_1 - a_1 \cdot x_2 + u\right), \qquad (6.32)$$

$$y = x_1.$$

Враховуючи прийняті позначення початкові умови приймуть такі значення: $x_1(0) = y(0)$ і $x_2(0) = y'(0)$

Таким чином, диференціальне рівняння (6.31) другого порядку ми замінили на систему диференціальних рівнянь, кожне із яких є диференціальним рівнянням першого порядку. Звернемо увагу на те, що кількість рівнянь у системі визначається порядком початкового диференціального рівняння (у нашому випадку це рівняння (6.31)).

Рівняння (6.32) подамо у векторній формі

$$\frac{d\bar{x}}{dt} = \bar{f}\left(\bar{x}, u\right), \qquad (6.33)$$

з початковими умовами $\bar{x}(0) = \bar{x}^{(0)}$,

де $\bar{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$; $\bar{f}\left(\bar{x}, u\right) = \begin{bmatrix} x_2 \\ \frac{1}{a_2}\left(-x_1 - a_1 \cdot x_2 + u\right) \end{bmatrix}$; $\bar{x}(0) = \begin{bmatrix} y(0) \\ y'(0) \end{bmatrix}$.

Тепер отримане векторне рівняння (6.33) можна розв'язати одним із числових методів, маючи на увазі, що у ньому t і h скалярні величини, а всі інші – векторні.

Розв'язок рівняння (6.33) отриманий із застосування методу Рунге-Кутта (рис. 6.3)

Розглянемо таке диференціальне рівняння:

$$a_n\frac{d^n y}{dt^n} + a_{n-1}\frac{d^{n-1} y}{dt^{n-1}} + \ldots + a_1\frac{dy}{dt} + a_0 y = b_m\frac{d^m u}{dt^m} + b_{m-1}\frac{d^{m-1} u}{dt^{m-1}} + \ldots + b_1\frac{du}{dt} + b_0 u, \qquad (6.34)$$

де a_i, $i = \overline{1,n}$; b_j, $j = \overline{1,m}$ - постійні величини;

u - величина, яка характеризує дію зовнішнього середовища на деяку систему, що описується рівнянням (6.34).

Для фізичних систем завжди має місце співвідношення $m \le n$, яке носить назву умови *фізичної реалізації системи.*

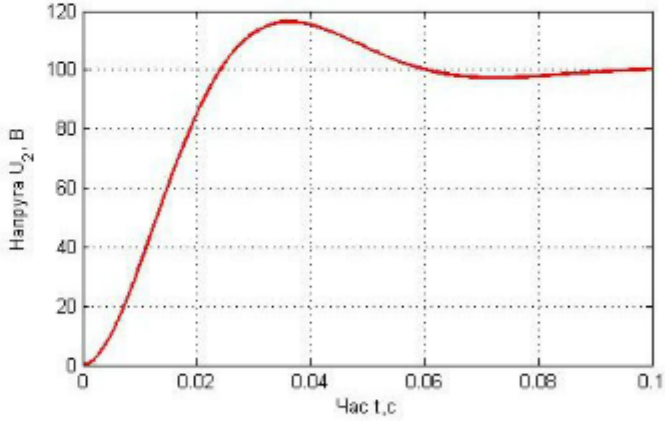


Рисунок 6.3 – Зміна напруги на виході у часі на виході електричної ланки

Розглянемо диференціальне рівняння (6.34) в якому $m = n$. Якщо це не так, то, прирівнюючи до нуля коефіцієнти

$b_{m+1}, b_{m+2}, \ldots, b_n$, завжди зможемо записати рівняння (6.34) у вигляді

$$a_n\frac{d^n y}{dt^n}+a_{n-1}\frac{d^{n-1}y}{dt^{n-1}}+\ldots+a_1\frac{dy}{dt}+a_0 y=b_n\frac{d^n u}{dt^n}+b_{n-1}\frac{d^{n-1}u}{dt^{n-1}}+\ldots+b_1\frac{d u}{dt}+b_0 u. \quad (6.35)$$

Рівняння (6.35) розпадається на систему диференціальних рівнянь першого порядку

$$\begin{aligned}
\frac{dx_1}{dt} &= x_2+\beta_1 u,\\
\frac{dx_2}{dt} &= x_3+\beta_2 u,\\
&\cdots\cdots\\
\frac{dx_{i-1}}{dt} &= x_i+\beta_{i-1}u,\\
&\cdots\cdots\\
\frac{dx_n}{dt} &= -\frac{1}{a_n}\sum_{j=1}^{n}a_{j-1}x_j+\beta_n u,\\
y &= x_1+\beta_0 u.
\end{aligned} \quad (6.36)$$

Величини $\beta_i, i=\overline{0,n}$ визначаються як розв'язок системи лінійних алгебраїчних рівнянь

$$\sum_{j=i}^{n}a_j\beta_{j-i}=b_i,\ i=\overline{0,n}. \quad (6.37)$$

Таким чином, диференціальне рівняння (6.35) n - го порядку подано у вигляді системи диференціальних рівнянь, кожне із яких є рівнянням першого порядку і кількість таких рівнянь у системі (6.36) визначається порядком n.

Відмітимо, що для диференціального рівняння (6.34) початкові умови нульові, а функція $u=u(t)$ повинна бути заданою.

Як і раніше, систему диференціальних рівнянь (6.36) можна подати у векторній формі

$$\frac{d\bar{x}}{dt}=\bar{f}(\bar{x},u), \quad (6.38)$$

де $\bar{x}=\begin{bmatrix}x_1\\x_2\\\cdots\\x_n\end{bmatrix}$; $\bar{f}(\bar{x},u)=\begin{bmatrix}x_2+\beta_1 u\\x_3+\beta_2 u\\\cdots\\-\frac{1}{a_n}\sum_{j=1}^{n}a_{j-1}x_j+\beta_n u\end{bmatrix}$.

Це означає, що до рівняння (6.38) можна застосувати раніше розглянуті методи числового розв'язку диференціальних рівнянь, маючи на увазі, що тепер $\bar{x}$ і $\bar{f}(\bar{x},u)$ векторні величини, t і h - скалярні.

Приклад 6.2. Знайти зміну напруги u_2 на виході пасивного фільтру (рис. 6.4) після подачі на його вхід напруги u_1, якщо $R_1=R_2=20\ Ом$, $L=0{,}1\ Г$, $C=1000\ мкФ$, $u_2=100\ В$. Допускаємо, що у початковий момент часу $t_0=0$ конденсатор C не заряджений.

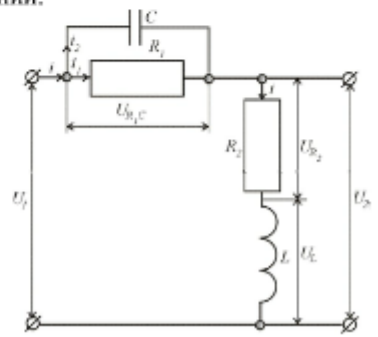


Рисунок 6.4 – Схема електричного фільтра

Запишемо перший і другий закони Кірхгофа для нашого випадку

$$i = i_1 + i_2, \tag{6.39}$$

$$U_1 = U_{R_1C} + U_L + U_{R_2}, \tag{6.40}$$

де U_{R_1C} і U_L - відповідно падіння напруг на опорі на вітці R_1 - C і індуктивності L.

З іншої сторони $U_2 = U_L + U_{R_2}$.

Оскільки в початковий момент часу $t_0 = 0$, $i(0) = 0$ і $U_C(0) = 0$, то $i_2 = C\dfrac{dU_{R_1C}}{dt}$, $i_1 = \dfrac{U_{R_1C}}{R_1}$. З врахуванням рівняння (6.39), будемо мати $i = C\dfrac{dU_{R_1C}}{dt} + \dfrac{U_{R_1C}}{R_1}$. Якщо взяти до уваги значення $U_2 = U_L + U_{R_2}$, то рівняння (6.40) можна записати в такій формі: $U_1 = U_{R_1C} + U_2$. Звідси знаходимо $U_{R_1C} = U_1 - U_2$. Отже, $i = C\dfrac{d(U_1 - U_2)}{dt} + \dfrac{U_1 - U_2}{R_1}$. Знайдемо тепер U_{R_2} і U_L. - $U_L = L\dfrac{di}{dt}$, $U_{R_2} = iR_2$. Враховуючи значення i будемо мати

$$U_{R_2} = R_2C\frac{d(U_1 - U_2)}{dt} + \frac{R_2}{R_1}(U_1 - U_2).$$

Значення U_L знайдемо, використавши формулу $U_L = L\dfrac{di}{dt}$.

Диференціюючи вираз $i = C\dfrac{d(U_1 - U_2)}{dt} + \dfrac{U_1 - U_2}{R_1}$ за змінною t, отримаємо $\dfrac{di}{dt} = C\dfrac{d^2(U_1 - U_2)}{dt^2} + \dfrac{1}{R_1}\dfrac{d(U_1 - U_2)}{dt}$. Тепер

$$U_L = LC\frac{d^2(U_1 - U_2)}{dt^2} + \frac{L}{R_1}\frac{d(U_1 - U_2)}{dt}.$$

Знаючи U_{R_2} і U_L, знайдемо

$$U_2 = LC\frac{d^2(U_1 - U_2)}{dt^2} + \frac{L}{R_1}\frac{d(U_1 - U_2)}{dt} + R_2C\frac{d(U_1 - U_2)}{dt} + \frac{R_2}{R_1}(U_1 - U_2)$$

Останнє рівняння запишемо у такому вигляді:

$$LC\frac{d^2U_2}{dt^2} + \left(R_2C + \frac{L}{R_1}\right)\frac{dU_2}{dt} + \left(1 + \frac{R_2}{R_1}\right)U_2 = LC\frac{d^2U_1}{dt^2} + \left(R_2C + \frac{L}{R_1}\right)\frac{dU_1}{dt} + \frac{R_2}{R_1}U_1. \tag{6.41}$$

Введемо такі позначення: $a_2 = LC$, $a_1 = R_2C + \dfrac{L}{R_1}$, $a_0 = 1 + \dfrac{R_2}{R_1}$; $b_2 = LC$, $b_1 = R_2C + \dfrac{L}{R_1}$, $b_0 = \dfrac{R_2}{R_1}$; $y = U_2$, $u = U_1$.

Тепер рівняння (6.41) набуде такого вигляду:

$$a_2\frac{d^2y}{dt^2} + a_1\frac{dy}{dt} + a_0y = b_2\frac{d^2u}{dt^2} + b_1\frac{du}{dt} + b_0u.$$

Тоді

$$\frac{dx_1}{dt} = x_2 + \beta_1 u,$$
$$\frac{dx_2}{dt} = -\frac{a_0}{a_2} x_1 - \frac{a_1}{a_2} x_2 + \beta_2 u, \qquad (6.42)$$
$$y = x_1 + \beta_0 u.$$

з початковими умовами

$$x_1(0) = 0;\ x_2(0) = 0.$$

Величини $\beta_i, \overline{i = 0,n}$ визначаються як розв'язок системи лінійних алгебраїчних рівнянь (6.37), де $n = 2$.
Для $n = 2$ отримаємо

$$a_0\beta_0 + a_1\beta_1 + a_2\beta_2 = b_0,$$
$$a_1\beta_0 + a_2\beta_1 = b_1,$$
$$a_2\beta_0 = b_2,$$

із якої визначаємо -

$$\beta_0 = \frac{b_2}{a_2};\beta_1 = \frac{a_2 b_1 - a_1 b_2}{a_2^2};\beta_2 = \frac{a_2^2 b_0 - a_0 a_2 b_2 - a_1 a_2 b_1 + a_1^2 b_2}{a_2^2}.$$

Запишемо рівняння (6.42) у формі векторного рівняння

(6.38), в якому $\overline{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$; $\overline{f}(\overline{x}, u) = \begin{bmatrix} x_2 + \beta_1 u \\ -\frac{a_0}{a_2} x_1 - \frac{a_1}{a_2} x_2 + \beta_2 u \end{bmatrix}$.

Тепер векторне диференціальне рівняння (6.38) можна розв'язати одним із числових методів. Вибираємо метод Рунге-Кутта. Результат такого розв'язку ілюструє рис. 6.5 , із якого видно, що у початковий момент часу, конденсатор C заряджається до напруги джерела. Потім з плином часу він розряджається до напруги, яка зумовлена співвідношеннями між опорами R_1 і R_2. У нашому випадку $U_2 = \frac{R_2}{R_1 + R_2} U_1$. Якщо врахувати значення R_1, R_2 і U_1, то $U_2 = 50$ B при $t \to \infty$.

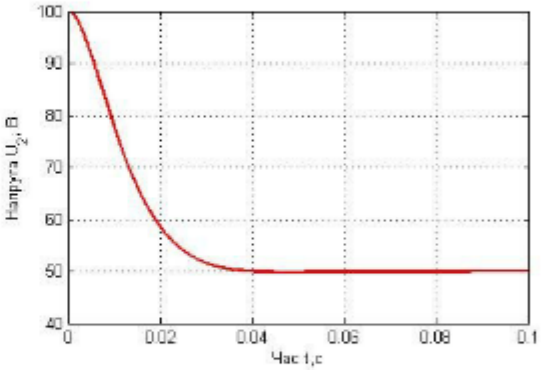


Рисунок 6.5 – Зміна напруги на виході електричного фільтра

Контрольні питання та завдання

1 Які недоліки притаманні методу Ейлера розв'язку диференціальних рівнянь?
2 Який порядок похибки методу Ейлера?
3 Порівняйте між собою методи Ейлера і Гюна.
4 Яка особливість методів сімейства Рунге-Кутта?

5 Який вигляд має загальна ітераційна процедура у методі Рунге-Кутта?

6 Як із загальної процедури Рунге-Кутта отримати метод Хойна і метод середньої точки?

7 Який порядок точності розв'язку диференціальних рівнянь забезпечує метод Рунге-Кутта?

8 Яка відмінність методу Кутта-Мерсона від методу Рунге-Кутта?

9 Яка особливість розв'язку систем диференціальних рівнянь у порівняння з розв'язками скалярних диференціальних рівнянь?

10. Яка особливість розв'язку диференціальних рівнянь вищих порядків?

7 Розв'язання інтегральних рівнянь

7.1 Квадратурні методи розв'язування інтегральних рівнянь Фредгольма і Вольтеррі

Інтегральним рівнянням називають рівняння відносно невідомої функції, яка знаходиться під знаком інтеграла.

Відомі закони збереження маси, імпульсу, енергії у своїй формалізованій постановці приводять до інтегральних рівнянь.

У загальному випадку інтегральне рівняння має такий вигляд:

$$x(t)=\int_D K\left(t,s,x(s)\right)ds+f(t), \qquad (7.1),$$

де D - деяка область n- вимірного простору; x - невідома, а f - відома векторні функції; K - у загальному випадку невідома відносно x функція.

Будемо розглядати лише скалярні рівняння, тобто такі, в яких шуканою невідомою є скалярна функція однієї змінної, а область інтегрування D - відрізок $[a;b]$. Крім того будемо вважати, що підінтегральна функція $K\left(t,s,x(s)\right)$ у (7.1) має такий вигляд:

$$K\left(t,s,x(s)\right)=Q(t,s)x(s). \qquad (7.2)$$

Інтегральні рівняння з підінтегральною функцією (7.2) називають *лінійними інтегральними рівняннями.*

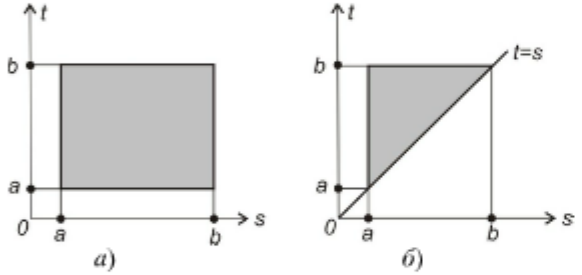
Залежно від того чи постійні обидві межі інтегрування, чи одна із них може бути змінною, розрізняють *лінійні інтегральні рівняння другого роду Фредгольма* –

$$x(t)=\lambda\int_a^b Q(t,s)x(s)ds+f(t) \qquad (7.3)$$

і *Вольтеррі*

$$x(t)=\lambda\int_a^t Q(t,s)x(s)ds+f(t), \qquad (7.4)$$

У рівняннях (7.3) і (7.4) задана і шукана функції $f(t)$ $x(t)$ залежать від змінної t, область визначення якої відрізок $[a;b]$. Функція двох змінних $Q(t,s)$, яка називається *ядром* інтегрального рівняння, визначена на множині точок квадрата $[a;b]\times[a;b]$ у випадку рівняння (7.3) (рис. 7.1,а) і трикутника $a\le s\le t\le b$, якщо маємо рівняння Вольтеррі (рис. 7.1,б).



а) *б)*

Рисунок 7.1 Області визначення ядер $Q(t,s)$ інтегральних рівнянь Фредгольма (а) і Вольтеррі (б)

Наявність параметру λ у рівнянь (7.3) і (7.4) надає цим рівнянням більш загальний вигляд і дає змогу виявити існування їх розв'язків при тих чи інших значенням λ. Якщо допустити, що $\lambda=1$, то розв'язків рівнянь (7.3) і (7.4) може і не бути.

Нехай для обчислення визначеного інтегралу використовується деяка квадратурна формула

$$\lambda\int_a^b Q(t,s)x(s)ds\approx\lambda\sum_{j=1}^{n}A_jQ(t,s_j)x(s_j),$$

з n вузлами і з відповідними їм ваговими коефіцієнтами A_j. Тоді отримаємо наближене подання розв'язку рівняння (7.3) через n його значень $x(s_1)$, $x(s_2)$, …, $x(s_n)$

$$x(t_i)=\lambda\sum_{j=1}^{n}A_jQ(t_i,s_j)x(s_j)+f(t_i), \qquad (7.5)$$

де значення t_i, $i=\overline{1,n}$, співпадають з вузлами s_j, $j=\overline{1,n}$.

Введемо такі позначення $Q_{ij}=Q(t_i,s_j)$, $f_i=f(t_i)$ $x_i=x(t_i)$ і $x_j=x(s_j)$. Тоді, змінюючи i від 1 до n в (7.5), отримуємо систему лінійних алгебраїчних рівнянь

$$a_{i1}x_1+a_{i2}x_2+\cdots+a_{ij}x_j+\cdots+a_{in}x_n=b_i,\ i=\overline{1,n}, \qquad (7.6)$$

де $a_{ij}=-\lambda A_jQ_{ij}$, $i\ j=\overline{1,n}$, $i\ne j$; $a_{ii}=1-\lambda A_iQ_{ii}$, $b_i=f_i$, $i=\overline{1,n}$.

Систему рівнянь (7.6) можна розв'язати за допомогою процедури `SystemEqualizationAgebra`. Результатом такого розв'язку будуть значення x_1, x_2, …, x_n, які утворюють *каркас наближеного розв'язку* інтегрального рівняння на сітці t_i, $i=\overline{1,n}$.

При фіксованих значеннях A_j і Q_{ij} системи рівнянь (7.6) зміною параметра λ (зменшення його модуля) можна покращити обумовленість[1] матриці коефіцієнтів a_{ij}.

---

[1] *Числом обумовленості* матриці A розміром $m\times n$ називають величину $cond(A)=\|A\|\cdot\|A^{-1}\|$, де через $\|\|$ позначено норму відповідної матриці. Норма матриці A це деяка скалярна величина, яка може бути визначена одним із трьох способів:

Для визначення параметрів A_j можна використати методи числового обчислення визначених інтегралів, наприклад метод трапецій чи метод Сімпсона.

Використання метода трапецій приводить до такої *складеної формули:*

$$\int_a^b Q(t,s)x(s)ds \approx \sum_{j=1}^{n} A_j Q(t,s_j)x(s_j) = \frac{h}{2}\left(Q_{t1}x_1 + 2Q_{t2}x_2 + 2Q_{t3}x_3 + \cdots + 2Q_{tM}x_M + Q_{t,M+1}x_{M+1}\right),$$

де $h = \dfrac{b-a}{M}$; M - кількість під інтервалів, на які розбитий інтервал $[a;b]$.

Таким чином, $A_1 = A_n = \dfrac{h}{2}$, $A_j = h$, $j = \overline{2,n-1}$, де $n = M+1$.

---

1. $\|A\|_1 = \max\limits_{1\le j\le n}\left(\sum\limits_{i=1}^{m}|a_{ij}|\right)$ - максимум суми модулів елементів матриці у стовпці.

2. $\|A\|_2 = \left(\lambda_{max}\left[A^T A\right]\right)^{1/2}$ - квадратний корінь максимального власного значення симетричної матриці $A^T A$.

3. $\|A\|_\infty = \max\limits_{1\le i\le m}\left(\sum\limits_{j=1}^{n}|a_{ij}|\right)$ - максимум суми модулів елементів матриці у рядку.

При великих $cond(A)$ точний розв'язок системи лінійних рівнянь може суттєво змінюватись навіть при малих змінах вхідних даних. Матриця A з великим $cond(A)$ називається *погано обумовленою* матрицею.

У випадку використання формули Сімпсона будемо мати

$$\int_a^b Q(t,s)x(s)ds \approx \sum_{j=1}^{n} A_j Q(t,s_j)x(s_j) = \frac{h}{3}\left(Q_{t1}x_1 + 4Q_{t2}x_2 + 2Q_{t3}x_3 + 4Q_{t4}x_4 + \cdots\right.$$

$$\left. \cdots + 2Q_{t,2M-1}x_{2M-1} + 4Q_{t,2M}x_{2M} + Q_{t,2M+1}x_{2M+1}\right),$$

де $h = \dfrac{b-a}{2M}$; $n = 2M+1$; $2M$ - кількість підінтервалів, на які розбитий інтервал $[a;b]$.

Із останнього співвідношення випливає, що $A_1 = A_n = \dfrac{h}{3}$, $A_{2i} = \dfrac{4}{3}h$, $A_{2i+1} = \dfrac{2}{3}h$, $i = \overline{1,\dfrac{n}{2}-1}$.

Точність числового розв'язку інтегрального рівняння квадратурним методом залежить від ряду взаємозв'язаних факторів: застосованої квадратурної формули, кількістю вузлів, які використовуються, властивостей функцій, що входять до інтегрального рівняння.

Параметр λ у лінійних інтегральних рівняннях Вольтерра не має ніякого смислового значення (на відміну від рівнянь Фредгольма). Тому, поклавши у (7.4) $\lambda = 1$, приходимо до рівняння

$$x(t) = \int_a^t Q(t,s)x(s)ds + f(t), \qquad (7.7)$$

яке будемо розв'язувати числовим методом.

Позначимо $K(t,s) = Q(t,s)$. Тоді можемо записати, що

$$K(t,s)=\begin{cases} Q(t,s) \text{ при } a\le s\le t\le b, \\ 0 \text{ при } a\le t\le s\le b. \end{cases}$$

Рівняння (7.7) замінимо наближеним його аналогом

$$x(t_i)=\sum_{j=1}^{i} A_j Q(t_i,s_j)x(s_j)+f(t_i),\ i=\overline{1,n}. \qquad (7.8)$$

Змінюючи i від 1 до n та, враховуючи прийняті раніше позначення, отримуємо систему лінійних алгебраїчних рівнянь

$$\begin{aligned} &a_{11}x_1=b_1, \\ &a_{21}x_1+a_{22}x_2=b_2, \qquad (7.9), \\ &a_{n1}x_1+a_{n2}x_2+\cdots+a_{nn}x_n=b_n, \end{aligned}$$

де $a_{ij}=-A_jQ_{ij}$, $i\,j=\overline{1,n}$, $i\ne j$; $a_{ii}=1-A_iQ_{ii}$, $b_i=f_i$, $i=\overline{1,n}$, із якої просто отримати шукані значення x_1, x_2, …, x_n.

Конкретне значення величини A_j, $i=\overline{1,n}$ визначається квадратурною формулою для заміни визначених інтегралів у рівнянні (7.7) кінцевими сумами (7.8). Як приклад, розглянемо квадратурну формулу трапецій. У цьому випадку при $i=1$ $t_1=a$ і $x_1=f(a)=f_1$. Якщо $i=2$, то $t=t_2$ і

$$\int_a^{t_2} Q(t,s)x(s)ds\approx\frac{h}{2}(Q_{21}x_1+Q_{22}x_2),$$

де $h=t_2-a$. З врахуванням (7.8) матимемо

$$x_2=\frac{h}{2}(Q_{21}x_1+Q_{22}x_2)+f_2.$$

Нехай $i=3$. Тоді $t=t_3$ і відповідно

$$\int_a^{t_3} Q(t,s)x(s)ds\approx\frac{h}{2}Q_{31}x_1+hQ_{32}x_2+\frac{h}{2}Q_{33}x_3.$$

Отриманий результат дає можливість записати

$$x_3=\frac{h}{2}Q_{31}x_1+hQ_{32}x_2+\frac{h}{2}Q_{33}x_3+f_3.$$

Таким чином, для першого рівняння системи (7.9) $A_1=\frac{h}{2}$; для другого рівняння - $A_1=A_2=\frac{h}{2}$; для третього рівняння - $A_1=A_3=\frac{h}{2}$, $A_2=h$ і т. д.

У загальному випадку будемо мати систему рівнянь (7.9), у якій $a_{11}=1$, $a_{12}=-\frac{h}{2}Q_{21}$, $a_{22}=1-\frac{h}{2}Q_{22}$, $a_{31}=-\frac{h}{2}Q_{31}$, $a_{32}=-hQ_{32}$, $a_{33}=1-\frac{h}{2}Q_{33}$, …, $a_{n1}=-\frac{h}{2}Q_{n1}$ $a_{n2}=-hQ_{n2}$, …, $a_{nn}=1-\frac{h}{2}Q_{nn}$.

Система лінійних алгебраїчних рівнянь (7.9) є верхньою трикутною і її розв'язок легко знайти, визначивши x_1 з першого рівняння. Потім, підставивши знайдене значення x_1 у друге рівняння, знову матимемо рівняння з одним невідомим, яке розв'яжемо відносно x_2 і т. д. У загальному випадку приходимо до такої формули:

$$x_i=\frac{1}{a_{ii}}\left(b_i-\sum_{j=1}^{i-1}a_{ij}x_j\right), \qquad (7.10)$$

яка дає змогу визначити *каркасний наближений розв'язок інтегрального рівняння* на сітці $t=t_i$, $i=\overline{1,n}$.

Приклад 7.1. Знайти каркасний наближений розв'язок інтегрального рівняння Вольтері

$$x(t)=\int_0^t t\cos^2\left(ts^3\right)x(s)ds+t^2-\frac{1}{3}tg\left(t^4\right). \qquad (7.11)$$

У нашому випадку $Q(t,s)=t\cos^2\left(ts^3\right)$, $f(t)=t^2-\frac{1}{3}tg\left(t^4\right)$. До рівняння (7.11) застосуємо квадратурну формулу трапецій з постійним кроком h. Задамо число вузлів n і кінцевий час t_f. Тоді $h=\frac{t_f-t_0}{n}$, де t_0 - початковий час. На рис. 7.2 показана сітка для числового розв'язку інтегрального рівняння Вольтеррі.

Для подальших обчислень створимо процедуру, яку назвемо `EqualizationVolterri`

```
    EqualizationVolterri
    ▷Задати числові значення n, t0, tf та обчислити
крок h

    ▷Сформувати вузли ti si i=1, 2, …, n
    ▷Обчислення коефіцієнтів aij
1 for i=1 to n
2     if i=1
3        then a(i,i)=1
4     end if
5     for j=1 to i
6         Обчислити Q=Qij
7         if j=i
8            then a(i,j)=1-h*Q/2
9         elseif j=1
10           then a(i,j)=-h*Q/2
11        else
12           a(i,j)=-h*Q
13        end if
14     end for
15 end for
    ▷Обчислити bi=f(ti), i=1, 2, …, n
```

```
    ▷Розв'язок верхньої трикутної системи лінійних
алгебраїчних рівнянь
    16 x(1)=f(1)
    17 for i=2 to n
    18     s=0
    19     for j=1 to i-1
    20         s=s+a(i,j)*x(j)
    21     end
    22     x(i)=(b(i)-s)/a(i,i)
    23 end for
    ▷Побудувати графік у координатах t, x
```

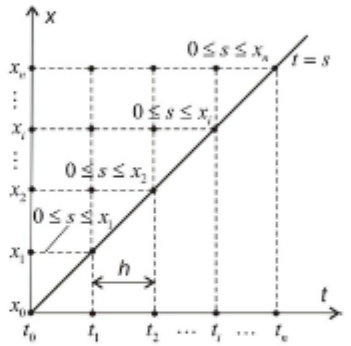


Рисунок 7.2 – Сітка числового розв'язку інтегрального рівняння Вольтеррі

Процедуру `EqualizationVolterri` реалізовано за таких значень параметрів: $n=10$, $t_0=0$, $t_f=0{,}8$. На рис. 7.3 показаний графік (суцільна лінія) каркасного розв'язку $x_i=x(t_i)$, $i=\overline{1,10}$ інтегрального рівняння Вольтеррі (7.11). Той самий рисунок вміщує графік (пунктирна лінія) точного розв'язку рівняння (7.11) - $x(t)=t^2$.

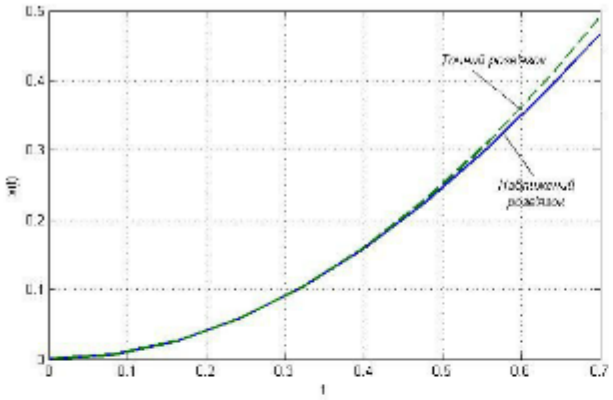


Рисунок 7.3 – Графіки функцій $x(t)$ - розв'язку інтегрального рівняння Вольтеррі (7.11)

Із рисунка видно, що у міру віддалення t_i від свого початкового значення t_0 похибка обчислення $x_i = x(t_i)$ зростає. Підвищити точність розв'язку задачі (7.11) можна збільшити, якщо використовувати точніші квадратурні формули для обчислення визначених інтегралів.

Контрольні питання та завдання

1. Дайте визначення інтегрального рівняння.
2. Яку назву мають інтегральні рівняння з постійною і змінною верхніми межами інтегрування?
3. Як знайти каркас наближеного розв'язку інтегрального рівняння?
4. Як підвищити точність розв'язку інтегрального рівняння при використанні числових методів?
5. Задано рівняння $x(t) = \int_a^b \left(\frac{t}{s^2} - 1\right) x(s)\,ds$. Знайти його наближений розв'язок, використавши формулу трапецій, якщо $a = 1$, $b = 3$. Порівняйте отриманий результат з точним його розв'язком $x(t) = t^2 + \frac{t}{6} - \frac{7}{3}$.

8 Задачі математичної фізики

8.1 Класифікація рівнянь математичної фізики

При вивченні більшості фізичних явищ та процесів приходиться мати справу з тим, що їх властивості описуються диференціальними рівняннями у часткових похідних, в яких невідомими величини є функції не однієї, а декількох змінних. Аргументам таких рівнянь надають змісту просторових змінних або просторових змінних і часу.

В загальному випадку диференціальне рівняння з частковими похідними можна записати у такій формі:

$$au_{xx} + bu_{yx} + cu_{yy} = f(x, y, u, u_x, u_y), \qquad (8.1)$$

де a, b, c - постійні коефіцієнти,

$u = u(x, y)$ - невідома функція, аргументи якої змінні x і y, одна із яких може бути часом t, а інша, допустимо x, просторовою змінною;

$u_{xx} = \frac{\partial^2 u(x, y)}{\partial x^2}$; $u_{xy} = \frac{\partial^2 u(x, y)}{\partial x\, \partial y}$; $u_{yy} = \frac{\partial^2 u(x, y)}{\partial y^2}$ - часткові похідні другого порядку;

$u_x = \frac{\partial u(x, y)}{\partial x}$; $u_y = \frac{\partial u(x, y)}{\partial y}$ - часткові похідні першого порядку;

$f\left(x, y, u, u_x, u_y\right)$ - відома функція, яка враховує взаємодію фізичної системи з навколишнім середовищем.

Існує три типи рівнянь (8.1).

Якщо $b^2 - 4ac < 0$, то рівняння (8.1) називається *еліптичним*;

Якщо $b^2 - 4ac = 0$, то рівняння (8.1) буде *параболічним* і якщо $b^2 - 4ac > 0$, то матимемо *гіперболічне* рівняння.

У випадку постійних коефіцієнтів при похідних другого порядку шляхом перетворення змінних рівняння (8.1) можна привести до виду, яке не вміщує змішаних похідних ($b = 0$). Тоді тип рівняння визначається за таким правилом: якщо у рівняння (8.1) $b = 0$ і постійні коефіцієнти при похідних другого порядку мають одинакові знаки в одній частині рівняння, то таке рівняння – *еліптичне*; якщо різні – *гіперболічне*; якщо похідна другого порядку за однією із змінних відсутня ($b = 0$; $ac = 0$ при $a^2 + c^2 \neq 0$) – *параболічне*.

8.2 Еліптичні рівняння

Як приклади еліптичних рівнянь у часткових похідних можна навести рівняння Лапласа, Пуассона і Гельмгольца.

Якщо ввести оператор Лапласа $\nabla^2 u = \frac{\partial^2 u(x, y)}{\partial x^2} + \frac{\partial^2 u(x, y)}{\partial y^2}$ функції $u(x, y)$, то в загальному вигляді можна записати

$\nabla^2 u = 0$ - рівняння Лапласа;

$\nabla^2 u = g\left(x, y\right)$ - рівняння Пуассона;

$\nabla^2 u + f\left(x, y\right) = g\left(x, y\right)$ - рівняння Гельмгольца,

де $g\left(x, y\right)$, $f\left(x, y\right)$ - задані функції, зміст яких визначається задачею, що розв'язується.

У рівняннях Лапласа, Пуасона, Гельмгольца одна із змінних може бути як просторовою координатою, так і часом t.

Рівняння Лапласа використовують для математичного опису електромагнітних полів, магнітних полів постійного струму, стаціонарних теплових полів.

Області застосування рівнянь Пуассона – задачі електростатики, електронної оптики, теорії пружності та ін.

Рівняння Гельмгольца є математичною моделлю коливних процесів, наприклад, в акустиці.

Допустимо, що розв'язок еліптичного рівняння належить області Ω з границею Г. Розглянемо найпростіший випадок, коли Ω - прямокутник $[a, b] \times [c, d]$.

Двовимірну область Ω накриємо сіткою вузлів з координатами $u_{ij} = u(x_i, y_j)$ (рис 8.1), де $x_i = a + ih_1$ $y_j = c + jh_2$, $h_1 = \dfrac{b-a}{n}$; $h_2 = \dfrac{d-c}{m}$, $i = \overline{0,n}$, $j = \overline{0,m}$, $(n+1)(m+1)$ - кількість вузлів розбиття ; h_1, h_2 - кроки та сітці.

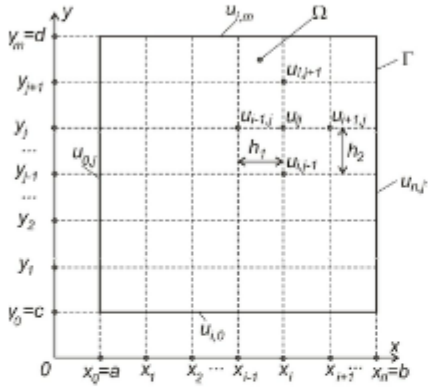


Рисунок 8.1 –Сітка для дискретизації задачі

Вузли u_{ij} називають *внутрішніми*, коли $i \in \{1,2,...,n-1\}$, $j \in \{1,2,...,m-1\}$ і *граничними*, коли $i = 0$ або $i = n$ та $j = 0$ або $j = m$, тобто $u(x_0, y_j) = u_{0,j}$, $j \in \{1,2,...,m-1\}$,

$$u(x_n, y_j) = u_{n,j}, \tag{8.2}$$

$u(x_i, y_0) = u_{i,0}$, $u(x_i, y_m) = u_{i,m}$, $i \in \{1,2,...,n-1\}$.

Виразимо оператор Лапласа у дискретному вигляді, скориставшись формулою для обчислення наближення до другої похідної

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}. \tag{8.3}$$

Відповідно до формули (8.3) будемо мати

$$\frac{\partial^2 u(x,y)}{\partial x^2} = \frac{u(x+h,y) - 2u(x,y) + u(x-h,y)}{h^2} \tag{8.4}$$

і відповідно

$$\frac{\partial^2 u(x,y)}{\partial y^2} \approx \frac{u(x,y+h) - 2u(x,y) + u(x,y-h)}{h^2}. \tag{8.5}$$

Якщо u_{ij} - розрахункова точка, то $u_{ij} = u(x_i, y_j)$ і відповідно $u_{i-1,j} = u(x_{i-1}, y_j)$, $u_{i+1,j} = u(x_{i+1}, y_j)$, $u_{i,j-1} = u(x_i, y_{j-1})$, $u_{i,j+1} = u\left(x_i, y_{j+1}\right)$, де $x_{i-1} = a + (i-1)h_1 = a + ih_1 - h_1 = x_i - h_1$.

Аналогічно $x_{i+1} = a + (i+1)h_1 = x_i + h_1$, $y_{j-1} = c + (j-1)h_2 = y_j - h_2$, $y_{j+1} = c + (j+1)h_2 = y_j + h_2$

Отже, в розрахунковій точці u_{ij} часткові похідні u_{xx} і u_{yy} можна замінити такими різницевими рівняннями :

$$u_{xx}\Big|_{\substack{x=x_i \\ y=y_j}} \approx \frac{u(x_i + h_1, y_j) - 2u(x_i, y_j) + u(x_i - h_1, y_j)}{h_1^2},$$

$$u_{yy}\Big|_{\substack{x=x_i \\ y=y_j}} \approx \frac{u(x_i, y_j + h_2) - 2u(x_i, y_j) + u(x_i, y_j - h_2)}{h_2^2}.$$

Переходячи до прийнятих позначень, отримаємо

$$u_{xx}\Big|_{\substack{x=x_i \\ y=y_j}} \approx \frac{u_{i+1,j} - 2u_{ij} + u_{i-1,j}}{h_1^2}, \tag{8.6}$$

$$u_{yy}\Big|_{\substack{x=x_i\\y=y_j}} \approx \frac{u_{i,j+1}-2u_{ij}+u_{i,j-1}}{h_2^2}. \qquad (8.7)$$

Таким чином,

$$\nabla^2 u(x,y)\Big|_{\substack{x=x_i\\y=y_j}} \approx \frac{1}{h_1^2}\left(u_{i+1,j}-2u_{ij}+u_{i-1,j}\right)+\frac{1}{h_2^2}\left(u_{i,j+1}-2u_{ij}+u_{i,j-1}\right). \qquad (8.8)$$

Різницеве рівняння (8.8) можна спростити, якщо крок сітки зробити постійним як за змінною x_1, так за змінною y_1 тобто $h_1 = h_2 = h$.

Тоді

$$\nabla^2 u(x,y)\Big|_{\substack{x=x_i\\y=y_j}} \approx \frac{u_{i-1,j}+u_{i+1,j}+u_{i,j-1}+u_{i,j+1}-4u_{i,j}}{h^2}. \qquad (8.9)$$

Допустимо, що числовим методом необхідно розв'язати диференціальне рівняння з частковими похідними другого порядку, яке є еліптичним рівнянням

$$\frac{\partial^2 u(x,y)}{\partial x^2}+\frac{\partial^2 u(x,y)}{\partial y^2}=f(x,y), \qquad (8.10)$$

де $f(x,y)$- задана функція, а Ω - прямокутник. Тоді у відповідності з (8.8) запишемо дискретний аналог еліптичного рівняння

$$\frac{1}{h_1^2}\left(u_{i+1,j}-2u_{ij}+u_{i-1,j}\right)+\frac{1}{h_2^2}\left(u_{i,j+1}-2u_{ij}+u_{i,j-1}\right)=f\left(x_i,y_j\right). \qquad (8.11)$$

Рівняння (8.10 необхідно доповнити сукупністю граничних умов (8.2) на Г.

Розглянемо техніку обчислення значень функції $u(x,y)$ у вузлах решітки Ω Розглянемо перших три шари решітки (рис 8.2). Зафіксуємо значення j. Візьмемо $j = 1$.

Тоді при $i=1$ із (8.11) отримаємо рівняння

$$-2(h_1^2+h_2^2)u_{11}+h_1^2u_{12}+h_2^2u_{21}=h_1^2h_2^2f_{11}-h_1^2u_{10}-h_2^2u_{01}.$$

Якщо тепер i змінювати від 2 до $n-2$, то

$$h_2^2u_{i-1,1}-2(h_1^2+h_2^2)u_{i1}+h_2^2u_{i+1,1}+h_2^2u_{i2}=f_{i1}h_1^2h_2^2-h_1^2u_{i0},\ i=\overline{2,n-2}.$$

І, нарешті, при $i=n-1$ будемо мати

$$h_2^2u_{n-2,1}-2(h_1^2+h_2^2)u_{n-1,1}+h_2^2u_{n1}+h_1^2u_{n-1,2}=h_1^2h_2^2f_{n-1,1}-h_1^2u_{n-1,0},$$

де $f_{ij}=f(x_i,y_j)$.

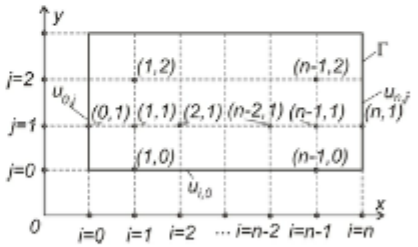


Рисунок 8.2-Перші три шари решітки Ω

Тепер можна взяти j = 2 і знову скласти систему із n-1 рівнянь. Повторюючи процес складання рівнянь для $j=3,\ldots,m-1$, отримаємо систему із $(n-2)(m-2)$ рівнянь в якій $(n-2)(m-2)$ невідомих.

Сказане проілюструємо конкретним прикладом.

Приклад 8.1. Знайдемо наближений розв'язок еліптичного рівняння $\nabla^2 u(x,y)=0$ у прямокутнику

$\Omega = \{(x,y): 0 \le x \le 6,\ 0 \le y \le 6\}$ де $u(x,y)$ - температура у точці (x,y), а граничні умови наступні :

$u(x,0) = 20;\quad u(x,6) = 160$ для $0 < x < 6$,

$u(0,y) = 80;\quad u(6,y) = 0$ для $0 < y < 6$.

Виберемо крок сітки $h_1 = h_2 = h = 1.5$ знайдемо кількість вузлів сітки: в напрямку x $n-1 = \dfrac{6}{1.5} = 4$; у напрямку y $n-1 = m-1 = \dfrac{6}{1.5} = 4$.

Сітка Ω показана на рис 8.3

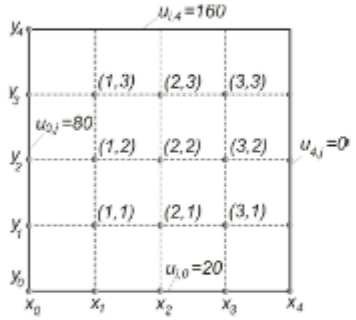


Рисунок 8.3 – Сітка Ω для *n*=*m*=5

Складемо систему алгебраїчних рівнянь використавши формулу (8.11) в якій $h_1 = h_2 = h$, а $f(x_i, y_j) \equiv 0$. Маємо

$$u_{i+1,j} - 4u_{ij} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} = 0. \qquad (8.12)$$

Змінюючи індекси i та j від 1 до 3 отримаємо систему із дев'яти рівнянь

а) $j = 1$

$i = 1:\ -4u_{11} + u_{21} + u_{12} = -100$,

$i = 2:\ u_{11} - 4u_{21} + u_{31} + u_{22} = -20$,

$i = 3:\ u_{21} - 4u_{31} + u_{32} = -20$;

б) $j = 2$,

$i = 1:\ u_{11} - 4u_{12} + u_{22} + u_{13} = -80$,

$i = 2:\ u_{21} + u_{12} - 4u_{22} + u_{32} + u_{23} = 0$,

$i = 3:\ u_{31} + u_{22} - 4u_{32} + u_{33} = 0$;

в) $j = 3$

$i = 1:\ u_{12} - 4u_{13} + u_{23} = -240$,

$i = 2:\ u_{22} + u_{13} - 4u_{23} + u_{33} = -160$,

$i = 3:\ u_{32} + u_{23} - 4u_{33} = -160$.

Отриману систему рівнянь можна записати у матрично векторній формі

$$A\bar{u} = \bar{b}, \qquad (8.13)$$

де

$$A = \begin{bmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{bmatrix}, \quad \bar{u} = \begin{bmatrix} u_{11} \\ u_{21} \\ u_{31} \\ u_{12} \\ u_{22} \\ u_{32} \\ u_{13} \\ u_{23} \\ u_{33} \end{bmatrix}, \quad \bar{b} = \begin{bmatrix} -100 \\ -20 \\ -20 \\ -80 \\ 0 \\ 0 \\ -240 \\ -160 \\ -160 \end{bmatrix}$$

Відмітимо, що при складанні системи рівнянь зручно використовувати шаблон типу "хрест" (рис 8.4) . Першою у

рівнянні (8.11) записується змінна $u_{i,j-1}$ другою $u_{i-1,j}$, третьою $u_{i,j}$ з коефіцієнтом мінус чотири, четвертю $u_{i+1,j}$ і п'ятою $u_{i,j+1}$.

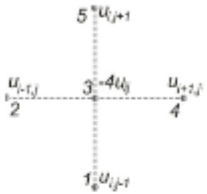


Рисунок 8.4-Шаблон типу "хрест"

У тому випадку, коли у процесі складання рівнянь шаблон вміщує граничні точки, то значення відповідних змінних u_{i0}, u_{iw}, u_{0j} та u_{xy}, які відповідають таким точкам, переносяться у праву частину рівнянь (8.11).

Для розв'язування отриманої системи рівнянь (8.13) скористаємося процедурою `SystemEqualizationAgebra` (роділ 6). У результаті розв'язку отримані такі значення компонентів вектора:

$\vec{u}^T = (54.286;\ 41.250;\ 25.714;\ 75.893;\ 65.000;\ 41.607;\ 104.290;\ 101;\ 250;\ 75.714)$

Графік зміни температурного поля в координатах (x, y) показаний на рис 8.5.

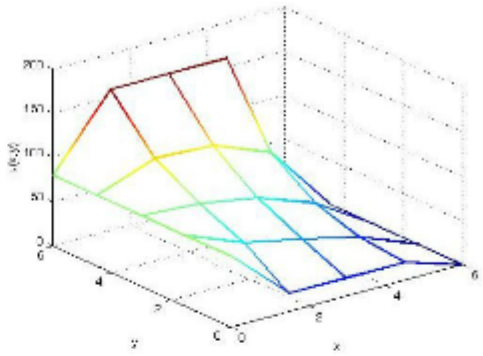


Рисунок 8.5 –Графік зміни температури у вузлових точках функції $u(x, y)$

Таким чином, задачу розв'язку еліптичних рівнянь ми апроксимували дискретною задачею, яка має другий порядок точності і яка має однозначне рішення. Це рішення збігається до рішення з початкової задачі при нескінченному згущенні вузлів сітки за двома змінними одночасно.

8.3 Параболічні рівняння

Як приклад параболічного рівняння розглянемо рівняння теплопровідності

$$\frac{\partial u}{\partial t} = a^2 \frac{\partial^2 u}{\partial x^2} + g(x,t)\ ,\quad x \in [0,l], t \in [0, t_f] \qquad (8.14)$$

з початковими умовами за змінною t

$$u(x,0) = \varphi(x) \text{ при } x \in [0,l] \qquad (8.15)$$

і крайовими умовами за просторовою координатою x

$$u(0,t)=\alpha(t),\ u(l,t)=\beta(t),\ \text{при } t\in\left[0,t_f\right] \qquad (8.16)$$

Рівняння (8.14) можна інтерпретувати як задачу визначення температури $u(x,t)$ тонкого однорідного стержня (бруса) довжиною l, якщо відомий розподіл температури в стержні у початковий момент часу $t = 0$ (початкова умова (8.15)) і відома зміна температури на кінцях стержня $x = 0$ і $x=l$ у будь-який момент часу t (граничні умови (8.16)) Коефіцієнт a^2 в (8.14) визначається теплофізичними властивостями матеріалу, а функція $g(t,x)$ виражає дію зовнішнього теплового джерела на стержень і вважається заданою.

Із умов (8.15) і (8.16) видно, що область Ω - прямокутник зі сторонами $\left[0;l\right]\times\left[0;t_f\right]$ у системі координат $x0t$, а її границя Γ утворена відрізками прямих $x=0$, $x=l$, $t=0$ і $t=t_f$. Як і раніше розіб'ємо прямокутник на $(n-1)\times(m-1)$ прямокутників зі сторонами h і τ, тобто

$$x_i=ih,\ \ h=\frac{l}{n},\ i=\overline{0,n}\ , \qquad (8.17)$$

$$t_j=j\tau,\ \ \tau=\frac{t_f}{m},\ \ j=\overline{0,m}\,. \qquad (8.18)$$

Значення функції $u(x,t)$ у вузлах (i,j) позначимо через $u(x_i,t_j)=u_{ij}$ (рис 8.6) Аналогічне позначення введемо і для функції $g(x,t)-g(x_i,t_j)=g_{ij}$.

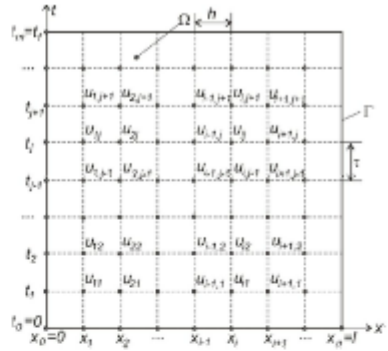


Рисунок 8.6 – Сітка області Ω

Часткові похідні $\frac{\partial^2 u}{\partial x^2}$ і $\frac{\partial u}{\partial t}$ замінимо різницевими апроксимаціями першого порядку

$$\left.\frac{\partial u(x,t)}{\partial t}\right|_{\substack{x=x_i\\t=t_j}}\approx\frac{u(x_i,t_j+\tau)-u(x_i,t_j)}{\tau},$$

$$\left.\frac{\partial^2 u(x,t)}{\partial x^2}\right|_{\substack{x=x_i\\t=t_j}}\approx\frac{u(x_i+h,t_j)-2u(x_i,t_j)+u(x_i-h,t_j)}{h^2}.$$

Оскільки (див. рис. 8.6) $t_{j+1}=t_j+\tau$, $x_{i-1}=x_i-h$, $x_{i+1}=x_i+h$, то

$$\left.\frac{\partial u(x,t)}{\partial x}\right|_{\substack{x=x_i\\t=t_j}}\approx\frac{u(x_i,t_{j+1})-u(x_i,t_j)}{\tau}=\frac{u_{i,j+1}-u_{ij}}{\tau}, \qquad (8.19)$$

$$\left.\frac{\partial^2 u(x,t)}{\partial x^2}\right|_{\substack{x=x_i\\ t=t_j}} \approx \frac{u(x_{i-1},t_j)-2u(x_i,t_j)+u(x_{i+1},t_j)}{h^2} = \frac{u_{i-1,j}-2u_{ij}+u_{i+1,j}}{h^2}. \quad (8.20)$$

Замінюючи в рівнянні (8.14) часткові похідні $\frac{\partial u(x,t)}{\partial t}$ і $\frac{\partial^2 u(x,t)}{\partial x^2}$ їх апроксимаціями (8.16) і (8.17).

Отримаємо наступне рівняння

$$u_{i,j+1} = \gamma u_{i-1,j} + (1-2\gamma)u_{ij} + \gamma u_{i+1,j} + \tau g_{ij},$$

$$i = \overline{1,n-1}, \quad j = \overline{1,m-1}, \quad (8.21)$$

де $\gamma = \tau\frac{a^2}{h^2}$.

До рівнянь (8.21) слід додати початкові та граничні умови, які отримаємо шляхом апроксимації функції умов (8.15) і (8.16) . Маємо

$$u(x_i,0) = u_{i0} = \varphi(x_i) = \varphi_i, \ i = \overline{0,n}, \quad (8.22)$$

$$\begin{cases} u(0,t_j) = u_{0j} = \alpha(t_j) = \alpha_j, \\ u(l,t_j) = u_{nj} = \beta(t_j) = \beta_j, \ j = \overline{0,m}. \end{cases} \quad (8.23)$$

Якщо індекси i та j в (8.21) будуть набувати значень від 1 до n-1 і від 1 до m-1 відповідно, то в результаті отримаємо систему $(n-2)(m-2)$ алгебраїчних лінійних рівнянь з $(n-2)(m-2)$ невідомими. Складання такої системи рівнянь опирається на шаблон різницевої схеми (8.18) (рис 8.7).

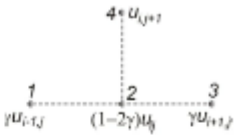


Рисунок 8.7 – Шаблон різницевої схеми (8.21)

Апроксимація рівняння (8.14) різницевою схемою (8.21) може привести до того, що обчислювальний процес стає нестійкий. Це означає, що помилки, які виникають на одному шарі можуть збільшуватись на наступних парах для деякою $k > j$. Для того щоб отримати стійку схему обчислень розмір, кроку τ повинен підкорятись нерівності $\tau \le \frac{h^2}{2a^2}$.

Інший спосіб отримання різницевої схеми, яка апроксимує параболічне рівняння (8.14) полягає в апроксимації першої похідної лівою різницею

$$\left.\frac{\partial u}{\partial t}\right|_{\substack{x=x_i\\ t=t_j}} \approx \frac{u(x_i,t_j)-u(x_i,t_{j-1})}{\tau} = \frac{u_{ij}-u_{i,j-1}}{\tau}. \quad (8.24)$$

Після заміни другої і першої похідних в (8.14) їх апроксимаціями (8.20) і (8.24) отримаємо

$$\frac{u_{ij}-u_{i,i-1}}{\tau} = a^2\frac{u_{i-1,j}-2u_{ij}+u_{i+1,j}}{h^2} + g_{ij}.$$

Звідси знаходимо, що

$$\gamma u_{i-1,j} - (1+2\gamma)u_{ij} + \gamma u_{i+1,j} + u_{i,j-1} = -\tau g_{ij}. \quad (8.25)$$

Зміні, які входять у різницеву схему (8.25), утворюють шаблон, який показано на рис. 8.8.

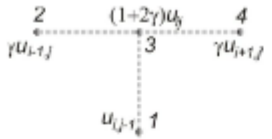


Рисунок 8.8 – Шаблон для різницевої схеми (8.25)

Структура системи рівнянь (8.25) така, що значення невідомих $u_{ij} = u(x_i, t_j)$ можна обчислювати для кожного шару окремо. Враховуючи, що на нульовому шарі значення u_{io} відомі, формула (8.25) дає можливість безпосередньо обчислювати всі значення $u_{i1}, i = \overline{1, n-1}$ для першого шару. Знаючи значення u_{i1} першого шару, можна обчислити значення u_{i2} для другого шару і т.д.

Отже, для знаходження значень u_{i1} у вузлах першого шару необхідно розв'язати систему лінійних алгебраїчних рівнянь

$$\gamma u_{i-1,1} - (1+2\gamma)u_{i1} + \gamma u_{i+1,1} = -\varphi_1 - \tau g_{i1},$$

де $i = \overline{1, n-1}$, $u_{01} = \alpha_1$, $u_{n1} = \beta_1$.

Значення величин u_{i2} у вузлових точках другого шару отримаємо як розв'язок системи

$$\gamma u_{i-1,2} - (1+2\gamma)u_{i2} + \gamma u_{i+1,2} = -u_{i1} - \tau g_{i2},$$

де $i = \overline{1, n-1}$, $u_{02} = \alpha_2$, $u_{n2} = \beta_2$, і т. д.

Для передостаннього шару, коли $j = m-1$, будемо мати

$$\gamma u_{i-1,m-1} - (1+2\gamma)u_{i,m-1} + \gamma u_{i+1,m-1} = -u_{i,m-2} - \tau g_{i,m-1}.$$

У кожній із двох розглянутих різницевих схем (8.21) і (8.25) для обчислення значень u_{ij} на i-ому шарі використовуються значення функції u_{ij}, на двох сусідніх шарах. Звідси і їх назва – *двошарові схеми.*

При цьому різницева схема (8.21), яка є формулою для безпосереднього обчислення шуканих значень u_{ij} носить назву – *явної схеми*, а схема (8.25), яка для своєї реалізації вимагає розв'язку систем алгебраїчних рівнянь (при переході від шару до шару) називається *неявною схемою.*

Приклад 8.2. Розв'язати числовим методом параболічне рівняння $\dfrac{\partial u(x,t)}{\partial t} = a^2 \dfrac{\partial^2 u(x,t)}{\partial x^2} + g(x,t)$, коли відсутня дія зовнішнього джерела тепла, тобто $g(x,t) \equiv 0$.

Шляхом вибору безрозмірних величин: $\theta = \dfrac{t}{T}$ і $\xi = \dfrac{x}{l}$ рівняння (8.14) можна привести до вигляду, у якому коефіцієнт при частковій похідні $\dfrac{\partial^2 u(x,t)}{\partial x^2}$ буде дорівнювати одиниці.

Оскільки $\partial\theta = \dfrac{1}{t_f}\partial t$ і $\partial\xi^2 = \dfrac{1}{l^2}\partial x^2$, то $\partial t = t_f \partial\theta$ і $\partial x^2 = l^2 \partial\xi^2$.

Підставляючи значення ∂x^2 і ∂t у рівняння (8.14), отримаємо

$$\frac{\partial u(\xi,\theta)}{\partial\theta} = t_f \frac{a^2}{l^2} \cdot \frac{\partial^2(\xi,\theta)}{\partial\xi^2}. \qquad (8.26)$$

Виберемо масштаб часу t_f таким, щоб коефіцієнт $t_f\left(\dfrac{a}{l}\right)^2$ у рівнянні (8.26) був рівний одиниці, тобто

$$t_f = \left(\frac{l}{a}\right)^2.$$

При такому виборі t_f рівняння (8.26) набуде такого вигляду:

$$\frac{\partial u(\xi,\theta)}{\partial \theta} = \frac{\partial^2 (\xi,\theta)}{\partial \xi^2}. \qquad (8.27)$$

з початковою умовою

$$u(\xi,\theta) = 4\xi(1-\xi) \text{ для } \theta = 0 \text{ і } 0 \le \xi \le 1 \qquad (8.28)$$

і граничними умовами

$$u(0,\theta) = \alpha(\theta) \equiv 0, \text{ для } x = 0 \text{ і } 0 \le \theta \le \theta_f,$$

$$u(1,\theta) = \beta(\theta) \equiv 0, \text{ для } x = 1 \text{ і } 0 \le 0 \le \theta_f, \qquad (8.29)$$

де $\theta_f = 0,5$.

Графік розподілу температури вздовж стержня в початковий момент часу показано на рис.8.9.

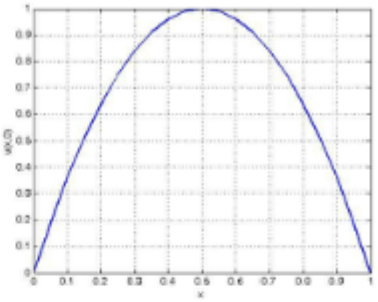


Рисунок 8.9 – Розподіл температури вздовж стержня у початковий момент часу

Крок дискретності τ виберемо таким, щоб виконувалась умова $\tau \le \frac{h^2}{2a^2} = \frac{h^2}{2}$. Візьмемо $\tau = \frac{h^2}{4}$.

Інтервал зміни ξ розіб'ємо на $m = 20$ частин. Тоді $h = \frac{1}{20} = 0,05$, а $\tau = \frac{1}{400 \cdot 4} = \frac{1}{1600}$ і відповідно $n = \frac{\theta_f}{\tau} = 0,5 \cdot 1600 = 800$.

Для розв'язку задачі (8.27) з початковою умовою (8.28) і граничними умовами (8.29) використаємо явну різницеву схему (8.21). Відповідне алгоритмічне забезпечення має назву `EqualizationHeat-conducting`, де індексація починається не з нуля, а з одиниці тому в (8.21) індекс j замінимо на $j-1$. У результаті отримаємо

$$u_{ij} = \gamma u_{i-1,j-1} + (1-2\gamma) u_{i,j-1} + \gamma u_{i+1,j-1}, \qquad (8.30)$$

де $\gamma = \tau \frac{a^2}{h^2} = \frac{h^2}{4} \cdot \frac{1}{h^2} = 0,25$.

```
EqualizationHeat-conducting
▷Процедура розв'язку рівняння теплопровідності
▷Вхід:fi(x)-заноситься у підпроцедуру run_fiX
▷      alfa(t)=u(0,t)i beta(t)=u(l,t)
▷      l i tk - праві точки інтервалів [0;l] і [0;tk]
▷      a-постійна у рівнянні теплопровідності
▷      n і m-число точок решітки [0;l] і [0;tk]
▷Вихід:U-матриця розв'язків
▷Вхідні дані
1 l=1
2 tk=0.5
3 c1=0
4 c2=0
5 a=1
6 m=80;
7 n=21
▷Кроки tau і h на решітці
8 h=1/(n-1)
9 tau=tk/(m-1)
▷Параметри рівняння та ініціалізація U
9 gama=tau*a^2/h^2;
10 gama_1=1-2*gama;
11 Створити нульову матрицю U розміром nxm
12 Створити нульовий вектор fi розміром n-2
▷Граничні умови
13 U(1,1:m)=c1
14 U(n,1:m)=c2;
▷Генерування значень функції u(x,t) другого рядка
15 x=0
16 for i=2 to n-1
17      x=x+h
18     fi=fun_fiX(x)
19     U(i,1)=fi
20 end for
21 for i=1 to n
22     X(i)=(i-1)*h
23     F(i)=fun_fiX(X(i))
24 end for
25 Побудувати графік розподілу температури вздовж
стержня у
  початковий момент часу в координатах F,X
▷Генерування значень функції u(x,t) наступних рядків
26 for j=2 to m
27     for i=2 to n-1
28         U(i,j)=gama_1*U(i,j-1)+gama*(U(i-1,j-
1)+U(i+1,j-1))
29     end for
30 end for
31 Побудувати графік функції у координатах x,y,U

Підпроцедура fun_fiX
fi=fun_fiX(x)
▷Функція, що задає початкові умови fi(x)
fi=4*x*(1-x)
```

Результати роботи програми відображені на рис. 8.10.

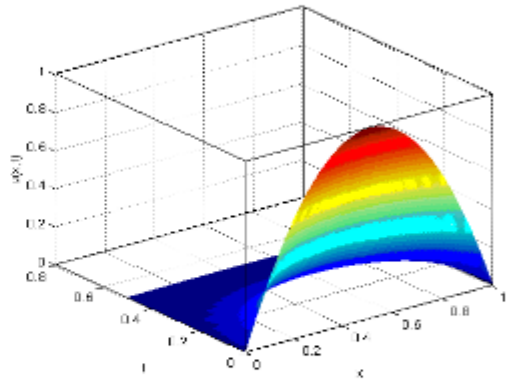


Рисунок 8.10 - Графік функції $u(x,t)$

8.4 Гіперболічні рівняння

Як приклад гіперболічного рівняння розглянемо математичну модель вільних коливань струни

$$\frac{\partial^2 u(x,t)}{\partial t^2} = a^2 \frac{\partial^2 u(x,t)}{\partial x^2}, \qquad (8.31)$$

кінці якої зафіксовані у точках $x = 0$ і $x = l$. У початковий момент часу кожній точці струни придали початкове

зміщення $\varphi(x)$ і швидкість $\psi(x)$. Постійна a^2 у рівняння (8.31) визначається вагою і натягом струни.

Отже, процес коливань струни визначається такими початковими

$$u(x,0)=\varphi(x) \text{ при } x\in[0;l], \tag{8.32}$$

$$\frac{\partial u(x,0)}{\partial t}=\psi(x), \text{ при } x\in[0;l] \tag{8.33}$$

і граничними

$$u(0,t)=0,\ u(l,t)=0 \text{ при } t\in\left[0;t_f\right] \tag{8.34}$$

умовами.

Задача (8.32) – (8.33) визначена на прямокутній області $\Omega=[0;l]\times\left[0;t_f\right]$, яка є ідентичною для задачі про теплопровідність і тому для побудови різницевої схеми доцільно використати ті ж вузли, що і раніше (див. п. 8.3).

Часткові похідні другого порядку будемо апроксимувати за відомими нам формулами:

$$\left.\frac{\partial^2 u(x,t)}{\partial t^2}\right|_{\substack{x=x_i\\t=t_j}}\approx\frac{u(x_i,t_{j-1})-2u(x_i,t_j)+u(x_i,t_{j+1})}{\tau^2}, \tag{8.35}$$

$$\left.\frac{\partial^2 u(x,t)}{\partial x^2}\right|_{\substack{x=x_i\\t=t_j}}\approx\frac{u(x_{i-1},t_j)-2u(x_i,t_j)+u(x_{i+1},t_j)}{h^2}. \tag{8.36}$$

Замінивши у (8.31) часткові похідні $\frac{\partial^2 u(x,t)}{\partial t^2}$ і $\frac{\partial^2 u(x,t)}{\partial x^2}$ їх наближеннями (8.35) і (8.36), отримаємо різницеву схему

$$\frac{u_{i,j-1}-2u_{ij}+u_{i,j+1}}{\tau^2}=\frac{u_{i-1,j}-2u_{ij}+u_{i+1,j}}{h^2},$$

яка апроксимує хвильове рівняння (8.31) на шаблоні типу «хрест» (рис. 8.4).

Якщо ввести позначення $\gamma=a^2\frac{\tau^2}{h^2}$, то

$$u_{i,j+1}=-u_{i,j-1}+\gamma u_{i-1,j}+2(1-\gamma)u_{ij}+\gamma u_{i+1,j}, \tag{8.37}$$

де $i=\overline{1,n-1}$, $j=\overline{1,m-1}$.

Відповідним чином видозмінюється початкова (8.32)

$$u(x_i,0)=u_{i0}=\varphi(x_i)=\varphi_i, i=\overline{1,n-1} \tag{8.38}$$

і гранична (8.34)

$$u(0,t_j)=u_{0i}=0,\ u(l,t_j)=u_{nj}=0,\ j=\overline{1,m} \tag{8.39}$$

умови.

Оскільки кінці струни зафіксовані у точках $x=0$ і $x=l$, то $u(0,0)=u_{00}=0$ і $u(l,0)=u_{n0}=0$.

Початкова умова (8.33) задана у диференціальній формі, що вимагає, застосування до неї форм апроксимації. Найпростіша із них

$$\left.\frac{\partial(x,0)}{\partial t}\right|_{x=x_i}\approx\frac{u(x_i,t_1)-u(x_i,0)}{\tau}=\frac{u_{i1}-u_{i0}}{\tau}.$$

Тоді відповідно з (8.33) $\frac{u_{i1}-u_{i0}}{\tau}=\psi_i$ або

$$u_{i1}=u_{i0}+\tau\psi_i. \tag{8.40}$$

У тому випадку, коли функція $\varphi(x)$ має другу похідну, можна скористатися формулою Тейлора порядку два, щоб обчислити значення u_{ij} для другого ряду.

Оскільки $\left.\frac{\partial^2 u(x,t)}{\partial x^2}\right|_{t=0} = \varphi''(x)$. Відповідно до рівняння (8.31) $\left.\frac{\partial^2 u(x,t)}{\partial x^2}\right|_{t=0} = a^2 \left.\frac{\partial^2 u(x,t)}{\partial x^2}\right|_{t=0}$

.

Отже,

$$\left.\frac{\partial^2 u(x,t)}{\partial x^2}\right|_{t=0} = a^2 \cdot \varphi''(x) \qquad (8.41)$$

Розкладемо функцію $u(x,\tau)$ у ряд Тейлора порядку два. Маємо

$$u(x,\tau) = u(x,0) + u_t(x,0)\tau + \frac{u_{tt}(x,0)\tau^2}{2} + O(\tau^3),$$

де $u_t(x,0) = \left.\frac{\partial u(x,t)}{\partial t}\right|_{t=0}$; $u_{tt}(x,0) = \left.\frac{\partial^2 u(x,t)}{\partial t^2}\right|_{t=0}$;

$O(\tau^3)$ - залишковий член ряду Тейлора.

Оскільки, відповідно до формули (8.33) $u_t(x,0) = \psi(x)$, а згідно з (8.40) $u_{tt}(x,0) = a^2\varphi''(x)$, то

$$u(x,\tau) \approx u(x,0) + \psi(x)\tau + \frac{a^2}{2}\varphi''(x)\tau^2. \qquad (8.42)$$

Формулу (8.42) можна застосувати для обчислення значень $u(x,t)$ для першого ряду. Для цього візьмемо $x = x_i$, $i = \overline{1, n-1}$, а функції $\varphi''(x)$ апроксимуємо за формулою

$$\varphi''(x)\Big|_{x=x_i} = \frac{\varphi_{i-1} - 2\varphi_i + \varphi_{i+1}}{h^2} + O(h^2), \qquad (8.43)$$

де $\varphi_i = \varphi(x_i)$.

Оскільки $u(x_i,\tau) = u(x_i,t_1) = u_{i1}$, то формули (8.42) і (8.43) дають можливість записати наступну формулу для дискретних значень x_i: $u_{i1} = u_{i0} + \tau\psi_i + \frac{a^2\tau^2}{2h^2}(\varphi_{i-1} - 2\varphi_i + \varphi_{i+1})$.

Оскільки $u(x_i,0) = u_{i0} = \varphi_i$,

то $u_{i1} = \varphi_i + \tau\psi_i + \frac{a^2\tau^2}{2h^2}(\varphi_{i-1} - 2\varphi_i + \varphi_{i+1})$.

Якщо врахувати, що $\gamma = \frac{a^2\tau^2}{h^2}$, то

$$u_{i1} = (1-\gamma)\varphi_i + \tau\psi_i + \frac{\gamma}{2}(\varphi_{i-1} + \varphi_{i+1}). \qquad (8.44)$$

На відміну від формули (8.37), яка забезпечує перший порядок точності $O(\tau)$, формула (8.40) має другий порядок точності $O(\tau^2 + h^2)$.

Доведено*, що умова $\tau \le \frac{h}{a}$ забезпечує стійкість обчислювального процесу за різницевою схемою (8.37).

Приклад 8.3. Задано хвильове рівняння

$$\frac{\partial^2 u(x,t)}{\partial t^2} = a^2 \frac{\partial^2 u(x,t)}{\partial t^2},\ 0 < x < 1 \text{ і } 0 < t < 0{,}5,$$

де $a = 2$, з граничними умовами

$$u(0,t) = 0,\ u(1,t) = 0 \text{ для } 0 \le t \le 0{,}5,$$
$$u(x,0) = 0.5 - x, \text{ для } 0 \le t \le 0{,}5,$$
$$u_t(x,0) = \psi(x) = 0 \text{ для } 0 < x < 1.$$

---

*Крылов В. П., Бобнов В. В., Монастырный П. Н. Начала теории вычислительных методов. Уравнения в частных производных. – Минск: Наука и техника, 1986.

Знайти його розв'язок числовим методом. Виберемо $h = 0.05$. Для стійкого обчислювального процесу повинна виконуватись умова $\tau \le \frac{h}{a}$. Візьмемо $\tau = \frac{h}{a}$. Оскільки $a = 2$, то $\tau = \frac{h}{a} = 0.025$. Отже, кількість точок для дискретних значень x - $n = \frac{1}{h} = 20$, а для дискретних значень $t - m = \frac{0.5}{\tau} = 20$.

Обчислення значень функції $u(x,t)$ у вузлах (i,j) будемо здійснювати за формулою (8.37). Значення $u(x,t)$ для нульового ряду обчислюється за формулою (8.38), а для першого ряду – за (8.44).

Процедура HyperbolicEqualization знаходить наближений розв'язок хвильового рівняння (8.31) за різницевою схемою (8.37).

```
HyperbolicEqualization
▷Розв'язок хвильового рівняння (гіперболічного)
▷Вхід:fi(x)-визначається підпроцедурою run_fiG
▷      psi(x)- визначається підпроцедурою run_psiG
▷      l і tk - праві точки інтервалів [0;l] і [0;tk]
▷      a-постійна у хвильовому рівнянні
▷      n і m-число точок решітки [0;l] і [0;tk]
▷Вихід:U-матриця розв'язків
▷Вхідні дані
1 l=1
2 tk=0.5
3 a=2
4 m=21
5 n=21
▷Кроки tau і h на решітці
6 h=1/(n-1)
7 tau=tk/(m-1)
▷Параметри рівняння та ініціалізація U
8 gama=tau^2*a^2/h^2
9 Створити нульову матрицю U розміром nxm
▷Генерування значень функції u(x,t) першого і другого рядів
10 for i=2 to n-1
11      fi_1=fun_fiG(h*(i-1))
12      fi=fun_fiG(h*i)
13      fi_2=fun_fiG(h*(i-2))
14      psi_1=fun_psiG(h*(i-1))
15      U(i,1)=fi_1
16                                          U(i,2)=(1-
gama)*fi_1+tau*psi_1+gama*(fi+fi_2)/2
17 end for
▷Генерування значень функції u(x,t) наступних рядків
18 for j=3 to m
19    for i=2 to n-1
20          U(i,j)=2*(1-gama)*U(i,j-1)+gama*(U(i-1,j-
1)+...
21               U(i+1,j-1))-U(i,j-2)
22    end for
23 end for
24 Побудувати графік функції у координатах x,y,U
```

М-файли до програми 8.3

```
Підпроцедура fun_fiG(x)
▷Функція, що задає початкові умови fi(x)
1 fi=fun_fiG(x)
2 if x<=0.5
3     then fi=x
4 else
5     fi=0.5-x
6 end if

Підпроцедура fun_psiG
▷Функція, що задає початкові умови fi(x)
1 psi=fun_psiG(x)
2 psi=0;
```

Тривимірне подання результатів обчислень показано на рис. 8.11.

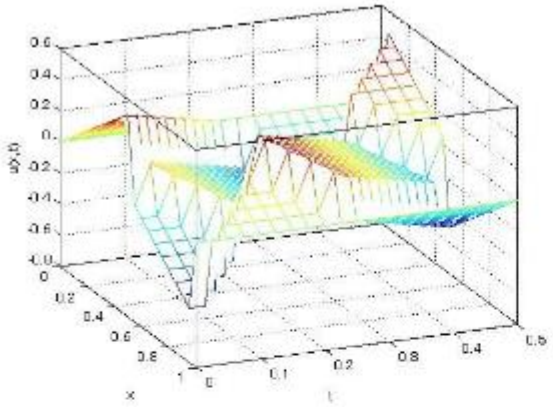


Рисунок 8.11 - Графік функції $u(x,t)$

Контрольні питання та завдання

1 Дайте класифікацію рівнянь математичної фізики.

2 Як здійснити класифікацію рівнянь математичної фізики у випадку, коли коефіцієнт при змішаних часткових похідних дорівнює нулю?

3 До якого класу рівнянь слід віднести рівняння Лапласа?

4 Назвіть області застосування рівнянь Пуассона.

5 Яким чином отримати сітку для дискретизації еліптичного рівняння?

6 Виразіть оператор Лапласа у дискретному вигляді.

7 Отримайте дискретний аналог еліптичного рівняння.

8 Розв'яжіть задачу із прикладу 8.1, коли $0 < x < 8$, $0 < y < 8$ і $h = 1$.

9. До якого класу рівнянь з частковими похідними відноситься рівняння теплопровідності?

10 Отримайте дискретний аналог параболічного рівняння.

11 Розв'яжіть задачу із прикладу 8.2, якщо $g(x,t) = 0{,}1x^2e^{-t}$.

12 Сформуйте граничні і початкові умови для гіперболічного рівняння.

13 Отримайте дискретний аналог гіперболічного рівняння.

14 Яка умова забезпечує стійкість обчислювального процесу за різницевою схемою для гіперболічного рівняння?

15 Розв'яжіть задачу із прикладу 8.3, якщо $\psi(x) = x$.

9 Методи наближення функцій

9.1 Поняття про наближення функцій

Часто при наукових дослідженнях виникають ситуації коли деяка функція відома тільки в певних точках, така функція може бути отримана експериментально або вона задана у вигляді таблиці. У таких випадках виникає задача наближення функції $f(x)$ іншою функцією $\varphi(x)$, яка є близькою до $f(x)$ за певним *критерієм узгодженості* (*наближення*). Таку заміну однієї функції іншою називають *апроксимацією* або *наближенням функцій*. Аналітичний вираз функції $f(x)$, як правило, - невідомий.

Таким чином, задача наближення функції $f(x)$ функцією $\varphi(x)$ полягає у побудові для заданої функції $f(x)$ такої функції $\varphi(x)$, що

$$f(x) \approx \varphi(x). \qquad (9.1)$$

Як функцію наближення у (9.1) будемо використовувати многочлени або функції, які є лінійними по відношенню до своїх параметрів. У першому випадку говорять про *поліноміальну апроксимацію* або *кусково-поліноміальну апроксимацію*. У другому випадку лінійність параметрів функції дає змогу в явному вигляді отримати формулу, що визначає параметри функції, які залежать як від структури $\varphi(x)$, так і від табличних значень функції $f(x)$. У залежності від вибору критерію наближення і від точок узгодження $f(x)$ з $\varphi(x)$ (будемо називати їх *вузлами*), тобто точок, в яких відома інформація про функцію $f(x)$ і, можливо, про її похідні, можна отримати різні способи апроксимації.

9.2 Наближення Лагранжа

Допускається, що в точках $N+1$ відомі точні значення як аргументу x_k, $k=\overline{0,N}$ так і самої функції $f(x_k)$. Нехай $y=\varphi(x)=P(x)$ деяка крива, яка проходить через точки $f(x_k)$.

Якщо $x_0 \le x \le x_N$, наближення $P(x)$ називають значенням *інтерполяції*[*]. Якщо $x < x_0$ або $x > x_N$, то $P(x)$ називають значенням *екстраполяції*.

Інтерполяція – це процес побудови інтерполяційної функції. $P(x)$ з метою знаходження проміжних значень табличної функції.

Нехай на відрізку $[a;b]$ в точках $x_0, x_1, ..., x_N$ відомі значення функції $y_k = f(x_k), k=\overline{0,N}$.

Поставимо задачу знайти інтерполяційну функцію $P(x)$ для $f(x)$, тобто таку функцію, значення якої $P(x_0), P(x_1), ..., P(x_N)$ в заданих точках $x_0, x_1, ..., x_N$ (у *вузлах інтерполяції*) співпадають з табличними значеннями функції: $y_0 = f(x_0); y_1 = f(x_1); ...; y_N = f(x_N)$;

Будемо вважати, що функція $P(x)$ є многочленом степеня N.

Таким чином, нам необхідно розв'язати таку задачу: для табличної функції $f(x)$ знайти многочлен $P(x)$ степеня N такий, що виконується сукупність умов інтерполяції

---

[*] Латинське слово interpolatio перекладається як оновлення, зміна, переробка. Цей термін увів в 1656 р. англійський математик Д. Уоллесом (1616-1703 рр.)

$$P_N(x_k) = y_k \ \forall k \in \{0,1,...,N\}. \qquad (9.2)$$

Многочлен $P_N(x)$ будемо шукати у такій формі:

$$P_N(x) = a_0 + a_1 x + a_2 x_2 + ... + a_N x^N. \qquad (9.3)$$

Оскільки степінь многочлена відомий, то розв'язок задачі інтерполяції зводиться до знаходження його коефіцієнтів: $a_0, a_1, .., a_N$.

Найпростіший шлях розв'язку такої задачі – це скласти систему лінійних відносно коефіцієнтів $a_0, a_1, ..., a_N$ рівнянь. Для цього як раз є $N+1$ умова (9.2). Таким чином, для того щоб многочлен (9.3) був інтерполяційним необхідно, щоб його коефіцієнти були розв'язком системи рівнянь

$$a_0 + a_1 x_k + a_2 x^2{}_k + ... + a_N x_k{}^N = y_k, k = \overline{0, N}.$$

Практичне застосування цього методу є малоефективним. Тому для побудови інтерполяційного поліному $P_N(x)$ користуються методом, який носить назву *наближення Лагранжа.*

Многочлен $P_N(x)$ будемо шукати у такій формі:

$$P_N(x) = \sum_{k=0}^{N} y_k \varphi_k(x).$$

Для того, щоб виконувались умови (9.2), функція $\varphi_k(x)$ в точці $x = x_k$ повинна дорівнювати одиниці і нулю в інших точках $x = x_i, i = \overline{0, N}, i \neq k$, тобто:

$$\varphi_k(x_i) = \delta_{ki} = \begin{cases} 0, \text{ якщо } i \neq k \\ 1, \text{ якщо } i = k \end{cases} \forall k, i \in \{0,1,...,N\}.$$

Рівність нулю $k-го$ многочлена в усіх вузлах інтерполяції, крім $k = i$, означає, що $\varphi_k(x)$ можна записати у такому вигляді:

$$\varphi_k(x) = A_k (x - x_0)...(x - x_{k-1})(x - x_{k+1})...(x - x_N) = A_k \prod_{\substack{i=0 \\ i \neq k}}^{N} (x - x_i) \qquad (9.5)$$

Коефіцієнт A_k вводиться для того, щоб у точці $x = x_k$ виконувалась умова $\varphi_k(x_k) = 1$. Отже, для того щоб знайти значення A_k, необхідно в (9.5) замість x підставити його значення у вузлі інтерполяції $(x = x_k)$ і отриманий результат прирівняти до одиниці. У результаті отримаємо:

$$A_k \prod_{\substack{i=0 \\ i \neq k}}^{N} (x_k - x_i) = 1.$$

Звідси знаходимо:

$$A_k = \frac{1}{\prod_{\substack{i=0 \\ i \neq k}}^{N} (x_k - x_i)}.$$

Знаючи значення A_k, знайдемо функцію $\varphi_k(x)$. Для цього необхідно знайдене значення A_k підставити у вираз (9.5)

$$\varphi_k(x)=\frac{\prod\limits_{\substack{i=0\\ i\neq k}}^{N}(x-x_i)}{\prod\limits_{\substack{i=0\\ i\neq k}}^{N}(x_k-x_i)}.$$

Тепер можемо знайти *інтерполяційний поліном Лагранжа* шляхом підставлення функції $\varphi_k(x)$ у вираз (9.4)

$$P_N(x)=\sum_{k=0}^{N}y_k\frac{\prod\limits_{\substack{i=0\\ i\neq k}}^{N}(x-x_i)}{\prod\limits_{\substack{i=0\\ i\neq k}}^{N}(x_k-x_i)}. \qquad (9.6)$$

Оскільки поліноми Лагранжа використовуються для апроксимації (наближення) функції *f(x),* то поза вузлами інтерполяції $P(x)\neq f(x)$. Тому важливо оцінити похибки, які при цьому виникають. Це можна зробити, використавши залишковий член ряду Тейлора, в якому множник $(x-a)^{N+1}$ слід замінити добутком

$$\prod_{N+1}(x)=(x-x_0)(x-x_1)...(x-x_N)=\prod_{i=0}^{N}(x-x_i).$$

Це природно, оскільки інтерполяція здійснюється у кожному із $N+1$ вузлів x_k, де $E_{N+1}(x_k)=f(x_k)-P(x_k)=0$. Отже, залишковий член ряду Тейлора для функції $f(x)$, який дає оцінку точності апроксимації, буде мати такий вигляд:

$$E_{N+1}(x)=\frac{\prod_{N+1}(x)f^{(N+1)}(z)}{(N+1)!} \qquad (9.7)$$

де z - деяке значення x із інтервалу $[a;b]$. Так як точка z невідома, то у формулі (9.7) значення $f^{(N+1)}(z)$ замінюють його верхньою оцінкою

$$M_{N+1}=\max_{x\in[a;b]}\left|f^{(N+1)}(z)\right|.$$

Тоді

$$\left|E_{N+1}(x)\right|\le\frac{M_{N+1}}{(n+1)!}\bar{\Pi}_{N+1}, \qquad (9.8)$$

де $\bar{\Pi}_{N+1}=\max\limits_{x\in[a;b]}\left|\Pi_{N+1}(x)\right|$.

Формула (9.8) має практичне значення лише в тому випадку, коли відома функція $f(x)$, що в інженерній практиці трапляється рідко.

Приклад 9.1. Обчислимо ординати функції $y=f(x)=x^2e^{-x}-0.5\sin x$ у рівновіддалених точках на відрізку $x\in[0;5]$. Візьмемо *N*=10 і відповідно $x_k=x_0+k\Delta x$, $k=\overline{0,N}$, а $\Delta x=\frac{x_0-x_N}{N}$, де x_0 - початкова, x_N - кінцева точки інтервалу $[a;b]$ $(x_0=a=0;x_N=b=5)$. Для значень аргументів $x_k, k=\overline{0,N}$ функції $f(x)$ обчислимо її ординати y_k. Результати обчислень зведемо у таблицю 9.1.

Таблиця 9.1 - Ординати функції $f(x)$ при вибраних x_k

x_k	0	0,5	1,0	1,5	2,0	2,5
y_k	0	-0,08808	-0,052856	0,003954	0,086692	0,2138
x_k	3,0		3,5	4,0	4,5	5,0
y_k	0,3775		0,5453	0,6714	0,7137	0,6479

Отримані результати будемо вважати табличною функцією, для якої побудуємо інтерполяційний поліном Лагранжа. Для цього використаємо процедуру, яку назвемо PolynomialLagrange.

```
   PolynomialLagrange
   ▷Наближення Лагранжа
   ▷Формування вхідних даних
   ▷Вхід:
   ▷      N - кількість вузлів інтерполяції
   ▷      x0 - початок інтервалу інтерполяції
   ▷      xk - кінець інтервалу інтерполяції
   ▷Вихід:
   ▷     X - значення аргументу
   ▷     Y - значення функції f(x)
   1 N=10
   2 x0=0
   3 xk=5
   ▷Обчислити значення функції f(x) у вузлах
інтерполяції
   ▷Підпроцедура: Визначення коефіцієнтів Лагранжа
   ▷a - коефіцієнти полінома Лагранжа
   ▷Вхід: X - вектор абсцис
   ▷       Y - вектор ординат
   ▷Вихід: a - матриця коефіцієнтів інтерполяції
   ▷            полінома Лагранжа
   ▷       d - матриця коефіцієнтів полінома Лагранжа
розміром nxn
   4 n=length(X) ▷Кількість вузлів інтерполяції
   5 N=n-1
   6 Сформувати нульову матрицю a розміром nxn
   ▷Формування коефіцієнтів полінома Лагранжа
   7 for k=1 to N+1
   8     V=1;
   9     for j=1 to N+1
   10        if k~=j
   11                then V=conv(V,poly(X(j)))/(X(k)-
X(j))
   12        end if
   13    end for
   14     d_k =V
   15 end for
   ▷Визначення коефіцієнтів інтерполяційного полінома
Лагранжа
   16 a=Y*d
   ▷Обчислення значень функції P(x) у вузлах
інтерполяції
   16 N1=length(a)
   17 for i=1 to Nn+1
   18    S=0
   19    for j=1 to N1
   20       r=a(N1-j+1)*X1(i)^(j-1);
   21       S=S+r
   22    end for
   23    P(i)=S
   24 end for
   ▷Побудувати графіки функцій f(x) і P(x)
```

У процедурі PolynomialLagrange прийнято такі позначення: d_i (рядок 14) - i - тий рядок матриці d; poly – оператор, який створює вектор, компоненти якого коефіцієнти полінома з точно визначеними коренями x_k; conv – оператор, який утворює вектор компоненти якого – коефіцієнти полінома, який дорівнює добутку двох поліномів.

Після проведених розрахунків отримали такі значення коефіцієнтів інтерполяційного полінома Лагранжа: a_0=0; a_1= - 0,42814; a_2= 0,99119; a_3= - 0,89861; a_4= 0,47878; a_5 = - 0,1548; a_6= 0,033396; a_7= -0,0052074; a_8=0,58463·10^{-3} a_9= - 0,41956·10^{-6} a_{10}= 0,14143·10^{-3}

На рис. 9.1 показані графіки функцій $f(x)$ і $P(x)$, а також „експериментальні” точки.

Видно, що значення їх ординат співпадають у вузлових точках з ”експериментальними” даними.

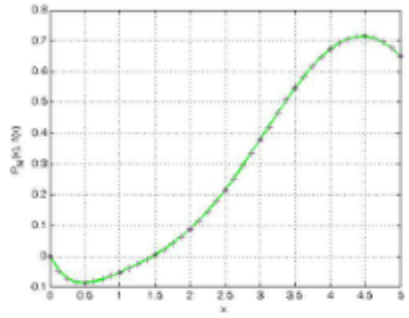


Рисунок 9.1 – Інтерполяція функції $f(x)$ поліномом Лагранжа

Оскільки інтерполяція здійснюються для того, щоб знайти значення функції $f(x)$ поза вузловими точками на проміжку [a;b], то, природно, поставити запитання про точність наближення Лагранжа. В даному випадку це можна зробити, так як відомо функція $f(x)$.

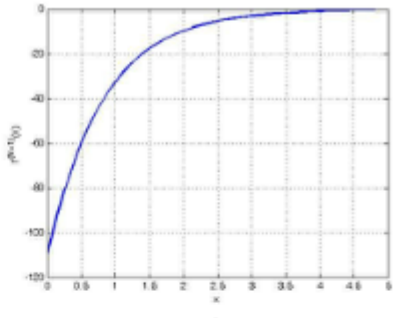
Обчислимо спочатку *k-му* похідну від функції $f(x)$. Методом індукції* можна показати, що

$$f^{(k)}(x)=(-1)^k\left(x^2-2kx+(k-1)k\right)e^{-x}-0.5sin\left(x+k\frac{\pi}{2}\right),$$

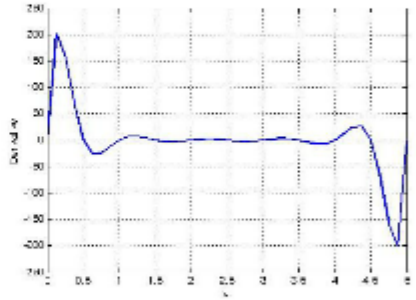
k=0,1,2,…

Далі знайдемо функцію $\prod_{N+1}(x)=\prod_{k=0}^{N}(x-x_k)$, де $x_k, k=\overline{0,N}$ - вузли інтерполяції і побудуємо графіки функцій $f^{(N+1)}(x)$ і $\prod_{N+1}(x)$ (рис. 9.2).

* Індукція (від латинського Induction-наведення) висновок від фактів до деякої гіпотези (загальному твердженню)



а)



б)

Рисунок 9.2 – Графік функцій $f^{(N+1)}(x)$ (*а*) і $\prod_{N+1}(x)$ (*б*).

Із графіків функцій $f^{(N+1)}(x)$ і $\prod_{N+1}(x)$ знаходимо, що на відрізку [a=0;b=5] $M_{N+1}=\max_{x\in[a;b]}\left|f^{(N+1)}(x)\right|=109.5$ і $\bar{\Pi}_{N+1}=\max_{x[a;b]}\prod_{N+1}(x)=201.4$. Підставляючи отримані значення M_{N+1} і $\bar{\Pi}_{N+1}$ у формулу (9.8), знаходимо, що

$|E_{N+1}(x)| \le 0.55248 \cdot 10^{-3}$,тобто в проміжних точках (поза вузлами інтерполяції) похибка наближена не перевершить величину $0{,}55248 \cdot 10^{-3}$.

9.3 Застосування інтерполяційного полінома Лагранжа для обчислення власного інтегралу

Нехай необхідно обчислити власний інтеграл

$$I = \int_a^b f(x)dx, \qquad (9.9)$$

де функція $f(x)$ - неперервна на відрізку $[a;b]$.

Розіб'ємо відрізок $[a;b]$ на N рівних частин, так що $x_k = x_0 + kh$, $k = \overline{1,N}$, де $x_0 = a$, $x_N = b$ і $h = \dfrac{b-a}{N}$. Тепер функцію $f(x)$ можна апроксимувати поліномом Лагранжа степені N. У відповідності з (9.6)

$$f(x) \approx P_N(x) = \sum_{k=0}^{N} f(x_k) L_{N,k}(x), \qquad (9.10)$$

де $L_{N,k}(x) = \dfrac{\prod_{i=0, i\neq k}^{N} (x - x_i)}{\prod_{i=0, j\neq k}^{N} (x_k - x_i)}$.

Наближене значення власного інтегралу (9.9) отримаємо шляхом заміни підінтегральної функції $f(x)$ її наближенням $P_N(x)$. У результаті отримаємо формулу числового інтегрування

$$I \approx \int_{x_0}^{x_N} \sum_{k=0}^{N} f(x_k) L_{N,k}(x)dx = \sum_{k=0}^{N} f(x_k) \int_{x_0}^{x_N} L_{N,k}(x)dx, \qquad (9.11)$$

яка носить назву формули *Ньютона-Котеса.*

У формулі (9.11) зробимо такі заміни: $x = x_0 + hz$. Тоді $x_k - x_i = x_0 + kh - x_0 - ih = (k-i)h$, $x - x_j = x_0 + zh - x_0 - ih = (z-i)h$ і $dx = hdz$. Відповідним чином зміняться і межі інтегрування: при $x = x_0$ змінна інтегрування $z = 0$, а при $x = x_N$ будемо мати - $x_N = x_0 + hz$, де $x_N = x_0 + Nh$. Звідси випливає, що $z = N$. Отже,

$$I \approx h \sum_{k=0}^{N} f(x_k) \int_0^N L_{N,k}(z)dz. \qquad (9.12)$$

З врахуванням зроблених замін у формулі (9.12) підінтегральний вираз набуде такого значення

$$L_{N,k}(z) = \frac{\prod_{i=0, i\neq k}^{N} (z - i)}{\prod_{i=0, j\neq k}^{N} (k - i)}. \qquad (9.13)$$

Надаючи величині N різні значення, можна отримати цілий ряд формул, які з різною точністю дають наближення власного інтегралу (9.9).

Для $N=1$ у відповідності з формулами (9.12) і (9.13) отримаємо $L_{N,0}(z)=-z+1$, $L_{N,1}(z)=z$.

Знаходимо

$$\int_0^1 (1-z)\,dz = z\Big|_0^1 - \frac{z^2}{2}\Big|_0^1 = \frac{1}{2}.$$

Аналогічно знаходимо, що

$$\int_0^1 z\,dz = \frac{1}{2}.$$

Відповідно до формули (9.12) маємо

$$I \approx \frac{h}{2}\left(f(x_0)+f(x_1)\right).$$

Остання формула носить назву *формули трапецій* обчислення наближеного значення інтегралу (9.9) на відрізку $[a;b]$ ($x_0=a$, $x_1=b$).

Для $N=2$ будемо мати $L_{N,0}(x)=\dfrac{(z-1)(z-2)}{2}$, $L_{N,1}(z)=-z(z-2)$ і $L_{N,2}(z)=\dfrac{z(z-1)}{2}$.

Тепер $\int_0^2 L_{N,0}(z)\,dz = \frac{1}{2}\int_0^2 (z-1)(z-2)\,dz = \frac{1}{3}$.

Аналогічно знаходимо, що $\int_0^2 L_{N,1}(z)\,dz = -\int_0^2 z(z-2)\,dz = \frac{4}{3}$, і нарешті, $\int_0^2 L_{N,1}(z)\,dz = \frac{1}{2}\int_0^2 z(z-1)\,dz = \frac{1}{3}$.

Таким чином,

$$I \approx \frac{h}{3}\left(f(x_0)+4f(x_1)+f(x_2)\right).$$

Отримана формула наближеного обчислення власного інтегралу носить назву *формули Сімпсона*.

Якщо взяти $N=3$, то отримаємо так звану *формулу Сімпсона 3/8*, при $N=4$ будемо мати *формулу Буля*.

Для підвищення точності числового інтегрування відрізок $[a;b]$ розбивають на N інтервалів $[x_0;x_1]$, $[x_1;x_2]$, $[x_2;x_3]$, …, $[x_{N-1};x_N]$, де $x_0=a$, $x_N=b$, і для кожного інтервалу $[x_{k-1};x_k]$, $k=\overline{1,N}$ обчислюють наближене значення інтегралу за формулою трапецій $I_k=\dfrac{h}{2}\left(f(x_{k-1})+f(x_k)\right)$. Тоді $I=\int_a^b f(x)\,dx \approx \sum_{k=1}^{N} I_k$.

Якщо врахувати значення I_k, то

$$I \approx \frac{h}{2}\left(f(a)+f(b)\right)+h\sum_{k=1}^{N-1} f(x_k). \qquad (9.14)$$

Остання формула носить назву *складена формула трапецій*.

Складену формулу Сімпсона для наближеного обчислення визначеного інтегралу (9.9) можна отримати,

якщо відрізок $[a;b]$ розбити на парне число інтервалів - N. У такому випадку:

$$I=\int_a^b f(x)dx \approx \sum_{k=1}^{N/2} I_k, \qquad (9.15)$$

де $I_k=\frac{h}{3}\left(f(x_{k-1})+4f(x_k)+f(x_{k+1})\right)$, $k=2i-1$, $i=\overline{1,\frac{N}{2}}$.

Підставлення значень I_k у вираз (9.15) приводить до наступного результату:

$$I \approx \frac{h}{3}\left(f(a)+f(b)+4\sum_{k=1}^{N/2} f(x_{2k-1})+2\sum_{k=1}^{N/2-1} f(x_{2k})\right). \qquad (9.16)$$

При використанні формул (9.14) і (9.15) виникають похибки

$$E_T(f,h)=\int_a^b f(x)dx-\frac{h}{2}\left(f(a)+f(b)\right)+h\sum_{k=1}^{N-1} f(x_k),$$

$$E_S(f,h)=\int_a^b f(x)dx-\frac{h}{3}\left(f(a)+f(b)+4\sum_{k=1}^{N/2} f(x_{2k-1})+2\sum_{k=1}^{N/2-1} f(x_{2k})\right),$$

які мають порядок $O(h^2)$ і $O(h^4)$ відповідно. Це означає, що похибка формули Сімпсона прямують до нуля швидше, ніж похибка формули трапецій при $h\to 0$.

У тому випадку, коли відомі похідні другого $f^{(2)}(x)$ і четвертого $f^{(4)}(x)$ порядків функції $f(x)$, формули

$$E_T(f,h)=-\frac{(b-a)f^{(2)}(\xi)h^2}{12} \text{ і } E_S(f,h)=-\frac{(b-a)f^{(4)}(\xi)h^4}{180},$$

де $\xi\in[a;b]$, можна використовувати для підрахунку числа інтервалів, які необхідні для досягнення заданої точності.

9.4 Кусково-поліномна апроксимація

У тих випадках, коли відрізок $[a;b]$, на якому виконується заміна функції $f(x)$ інтерполяційним поліномом $P_N(x)$, є досить великим і відсутні підстави вважати функцію $f(x)$ (яка, як правило, невідома) достатньо гладкою при $x\in[a;b]$, немає сенсу підвищувати точність інтерполяції за рахунок збільшення степені полінома $P_N(x)$. У таких випадках доцільно застосовувати *кусково-поліномну апроксимацію*, допускаючи, що функція апроксимації $\varphi(x)$ складається із окремих поліномів, як правило, невисокої степені, кожний із яких визначений на своїй частині відрізку $[a;b]$.

9.4.1 Кусково-лінійна і кусково-квадратична інтерполяція

Найпростішим із поліномів, які використовують для кусково-поліноміальної апроксимації це поліноми першої степені, які утворюють ломану лінію, що проходить через задані точки.

Припустимо, що на системі вузлів x_k відомі значення функції $y_k = f(x_k)$. Абсциси x_k упорядковані наступним чином:

$$a = x_0 < x_1 < x_2 < \cdots < x_N = b .$$

Необхідно апроксимувати $f(x)$ кусково-лінійною функцією $\varphi(x)$, виходячи із умов інтерполяції

$$\varphi(x_k) = y_k ,\ k = \overline{1,N} .$$

Якщо функцію $\varphi(x)$ взяти у вигляді $\varphi(x) = ax + b$, виходячи із умов інтерполяції, отримуємо систему із $2N$ лінійних рівнянь

$$\begin{array}{l} a_1 x_0 + b_1 = y_0; \\ a_1 x_1 + b_1 = y_1; \\ a_2 x_1 + b_2 = y_1; \\ a_2 x_2 + b_2 = y_2; \\ a_n x_{n-1} + b_n = y_{n-1}; \\ a_n x_n + b_n = y_n. \end{array} \qquad (9.17)$$

Кожна пара сусідніх рівнянь системи (9.17), яка має коефіцієнти з однаковими індексами є незалежною від інших пар і може розв'язуватись самостійно.

При кусково-квадратичній інтерполяції функція $\varphi(x)$ вибирається у вигляді квадратного трьох члена - $\varphi(x) = ax^2 + bx + c$. При $N = 2M$ кожна ланка *кусково-квадратичної функції* визначається трійкою коефіцієнтів a_k, b_k і c_k, послідовним розв'язком трьохвимірних лінійних систем

$$\begin{array}{c} a_k x_{2k-2}^2 + b_k x_{2k-2} + c_k = y_{2k-2}; \\ a_k x_{2k-1}^2 + b_k x_{2k-1} + c_k = y_{2k-1}; \\ a_k x_{2k}^2 + b_k x_{2k} + c_k = y_{2k}, \end{array}$$

де $k = \overline{1,M}$.

Недоліком кусково-квадратичної інтерполяції є те, що кривизна у парних вузлах x_{2k} різко змінюється, а це може викликати небажаний вигин або викривлення графіка функції $\varphi(x)$.

9.4.2 Інтерполяційний сплакни

Побудова поліноміальної лінії за заданою сукупністю точок здійснюється у системах комп'ютерної графіки. Для цієї мети застосовують кубічний сплайни.

Сплайном називають функцію $S_m(x)$, яка визначена на відрізку $[a;b]$ l раз неперервне диференційована і така, що на кожному проміжку $[x_{k-1};x_k]$ - це многочлен m-ої степені. Різниця $d = m - l$ називається *дефектом* сплайна.

Якщо сплайни апроксимує деяку функцію $f(x)$, яка задана своїми ординатами $y_k = f(x_k)$, і при цьому виконується умова $S_m(x_k) = y_k$, то такий сплайни називається *інтерполяційним сплайном* для функції $f(x)$.

Найбільш відомий і такий, що знайшов широке застосування, є кубічний сплайни дефекту 1. При цьому допускають, що вузли сплайна $a = x_0 < x_1 < x_2 < \cdots < x_N = b$

одночасно є і вузлами інтерполяції, тобто $y_k = f(x_k)$, $k = \overline{0,N}$.

Отже, кубічним сплайном дефекту 1, який на відрізку $[a;b]$, інтерполює задану функцію $f(x)$ називається функція

$$q_k(x) = a_k + b_k(x - x_k) + c_k(x - x_k)^2 + d_k(x - x_k)^3;\ x \in [x_{k-1}; x_k];$$

$$k = \overline{1,N}, \qquad (9.18)$$

яка задовольняє таку сукупність умов:

- умову інтерполяції у вузлах сплайна - $q(x_k) = f(x_k)$;
- подвійну неперервну диференційованість на відрізку $[a;b]$;
- крайові умови - $q''(a) = q''(b) = 0$.

Визначений у такий спосіб сплайни називають ще *креслярським сплайном*[1]. Щоб побудувати кубічний сплакни, (9.18) необхідно визначити $4N$ його коефіцієнти, виходячи із умов інтерполяції

$$q_1(x_0) = f_0,\ q_k(x_k) = f_k \text{ при } k = \overline{1,N}$$

гладкого спряження ланок сплайну

$$q_{k-1}(x_{k-1}) = q_k(x_{k-1}),$$
$$q'_{k-1}(x_{k-1}) = q'_k(x_{k-1})$$
$$q''_{k-1}(x_{k-1}) = q''_k(x_{k-1}),\ k = \overline{2,N}$$

та крайових умов

$$q''_1(x_0) = 0,\ q''_N(x_N) = 0.$$

[1] Англійське слово spline як планка, рейка, брусок.

Із (9.18) знаходимо, що

$$q'_k(x) = b_k + 2c_k(x - x_k) + 3d_k(x - x_k)^2. \qquad (9.19)$$

$$q''_k(x) = 2c_k + 6d_k(x - x_k). \qquad (9.20)$$

Рис. 9.3 дає наочне уявлення про розташування вузлів і ланок кубічного сплайну.

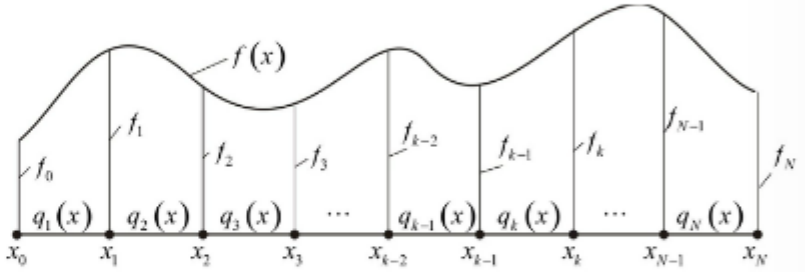


Рисунок 9.3 – Розташування вузлів і ланок кубічного сплайну

Враховуючи значення функції $q_k(x)$, яке визначається формулою (9.18), та умови інтерполяції, знаходимо

$$q_1(x_0) = f_0 = a_1 + b_1(x_0 - x_1) + c_1(x_0 - x_1)^2 + d_1(x_0 - x_1)^3,$$
$$q_k(x_k) = a_k + b_k(x_k - x_k) + c_k(x_k - x_k)^2 + d_k(x_k - x_k)^3 = a_k = f_k,$$

де $f_k = f(x_k)$, $k = \overline{1,N}$.

Введемо позначення $h_k = x_k - x_{k-1}$, $k = \overline{1,N}$. Тоді

$$a_1 - b_1 h_1 + c_1 h_1^2 - d_1 h_1^3 = f_0,$$

$$a_k = f_k,\ k = \overline{1,N}\ /$$

Виходячи із умов гладкого спряження ланок сплайну, отримаємо

$$q_{k-1}(x_{k-1}) = a_{k-1} + b_{k-1}(x_{k-1} - x_{k-1}) + c_{k-1}(x_{k-1} - x_{k-1})^2 + d_{k-1}(x_{k-1} - x_{k-1})^3,$$
$$q_k(x_{k-1}) = a_k + b_k(x_{k-1} - x_k) + c_k(x_{k-1} - x_k)^2 + d_k(x_{k-1} - x_k)^3,$$
$$q'_{k-1}(x_{k-1}) = b_{k-1} + 2c_{k-1}(x_{k-1} - x_{k-1}) + 3d_{k-1}(x_{k-1} - x_{k-1})^2,$$
$$q'_k(x_{k-1}) = b_k + 2c_k(x_{k-1} - x_k) + 3d_k(x_{k-1} - x_k)^2,$$
$$q''_{k-1}(x_{k-1}) = 2c_{k-1} + 6d_{k-1}(x_{k-1} - x_{k-1}),$$
$$q''_k(x_{k-1}) = 2c_k + 6d_k(x_{k-1} - x_k).$$

Враховуючи співвідношення (9.18) – (9.20), маємо

$$a_{k-1} = a_k - b_k h_k + c_k h_k^2 - d_k h_k^3,$$
$$b_{k-1} = b_k - 2c_k h_k + 3d_k h_k^2,$$
$$c_{k-1} = c_k - 3d_k h_k,\ k = \overline{2,N}.$$

Крайові умови дають змогу знайти

$$q''_1(x_0) = 2c_1 + 6d_1(x_0 - x_1)^2 = 0,$$
$$q''_N(x_N) = 2c_N + 6d_N(x_N - x_N) = 0.$$

або $c_1 - 3d_1 h_1^2 = 0$,.

Таким чином, отримали таку систему лінійних алгебраїчних рівнянь:

$$a_1 - b_1 h_1 + c_1 h_1^2 - d_1 h_1^3 = f_0, \qquad (9.21)$$

$$a_k = f_k,\ k = \overline{1,N}; \qquad (9.22)$$

$$a_{k-1} = a_k - b_k h_k + c_k h_k^2 - d_k h_k^3, \qquad (9.23)$$

$$b_{k-1} = b_k - 2c_k h_k + 3d_k h_k^2, \qquad (9.24)$$

$$c_{k-1} = c_k - 3d_k h_k,\ k = \overline{2,N}; \qquad (9.25)$$

$$c_1 - 3d_1 h_1^2 = 0, \qquad (9.26)$$

$$c_N = 0. \qquad (9.27)$$

В отриманій системі рівнянь коефіцієнти a_k відомі і дорівнюють f_k для всіх значень $k = \overline{1,N}$. Підставляючи їх значення в рівності (9.21) і (9.23), приходимо до таких співвідношень:

$$-b_1 h_1 + c_1 h_1^2 - d_1 h_1^3 = f_0 - a_1 = f_0 - f_1, \qquad (9.28)$$

$$-b_k h_k + c_k h_k^2 - d_k h_k^3 = a_{k-1} - a_k = f_{k-1} - f_k. \qquad (9.29)$$

Для спородження запису введемо таке позначення:

$$\Delta(f_{k-1}, f_k) = \frac{f_k - f_{k-1}}{h_k}.$$

Величина $\Delta(f_{k-1}, f_k)$ носить назву роздільної *різниці першого порядку* або скорочено – *роздільної різниці*. Враховуючи зроблене позначення рівняння (9.28) і (9.29) набудуть такого вигляду:

$$b_1 - c_1 h_1 + d_1 h_1^2 = \Delta(f_0, f_1),$$

$$b_k - c_k h_k + d_k h_k^2 = \Delta(f_{k-1}, f_k),\ k = \overline{2,N}.$$

Із отриманих рівнянь визначимо b_k і об'єднаємо їх в одне

$$b_k = \Delta(f_{k-1}, f_k) + c_k h_k - d_k h_k^2,\ k = \overline{1,N}. \qquad (9.30)$$

Тепер із (9.25) та (9.26) знайдемо

$$d_k = \frac{c_k - c_{k-1}}{3h_k},\ k = \overline{1,N}, \qquad (9.31)$$

де $c_0 = 0$ - фіктивний коефіцієнт.

Знайдене значення d_k дає змогу вилучити його із співвідношення (9.30)

$$b_k = \Delta(f_{k-1}, f_k) + \frac{2}{3} c_k h_k + \frac{1}{3} c_{k-1} h_k,\ k = \overline{1,N}. \qquad (9.32)$$

В останньому виразі замінимо k на $k-1$

$$b_{k-1} = \Delta(f_{k-2}, f_{k-1}) + \frac{2}{3} c_{k-1} h_{k-1} + \frac{1}{3} c_{k-2} h_{k-1}.$$

Отримані значення d_k, b_k і b_{k-1} підставимо у рівняння (9.24). У результаті отримаємо

$$\Delta(f_{k-2}, f_{k-1}) + \frac{2}{3} c_{k-1} h_{k-1} + \frac{1}{3} c_{k-2} h_{k-1} = \Delta(f_{k-1}, f_k) + \frac{2}{3} c_k h_k + \frac{1}{3} c_{k-1} h_k - 2c_k h_k + (c_k - c_{k-1}) h_k$$

.

Після нескладних алгебраїчних перетворень приходимо до такого різницевого рівняння:

$$h_{k-1} c_{k-2} + 2(h_{k-1} + h_k) c_{k-1} + h_k c_k = v_{k-1}, \qquad (9.33)$$

де $k = \overline{2,N}$; $v_{k-1} = 3\left(\Delta(f_{k-1}, f_k) - \Delta(f_{k-2}, f_{k-1})\right)$; $c_0 = c_N = 0$.

Змінюючи k від 2 до N, отримаємо систему лінійних алгебраїчних рівнянь

$$\begin{aligned}
&h_1 c_0 + 2(h_1 + h_2) c_1 + h_2 c_2 = v_1,\\
&h_2 c_1 + 2(h_2 + h_3) c_2 + h_3 c_3 = v_2,\\
&h_3 c_2 + 2(h_3 + h_4) c_3 + h_4 c_4 = v_3, \qquad (9.34)\\
&h_{N-2} c_{N-3} + 2(h_{N-2} + h_{N-1}) c_{N-2} + h_{N-1} c_{N-1} = v_{N-2},\\
&h_{N-1} c_{N-2} + 2(h_{N-1} + h_N) c_{N-1} + h_N c_N = v_{N-1}.
\end{aligned}$$

Маємо систему із $N-1$ рівнянь, у якій невідомими є коефіцієнти сплайна c_1, c_2, …, c_{N-1} ($c_0 = 0$ і $c_N = 0$).

Систему рівнянь (9.34) подамо у векторно-матричній формі $H\overline{c} = \overline{v}$,
де

$$H = \begin{bmatrix}
2(h_1 + h_2) & h_2 & 0 & 0 & \cdots & 0 & 0 & 0\\
h_1 & 2(h_2 + h_3) & h_3 & 0 & \cdots & 0 & 0 & 0\\
0 & h_2 & 2(h_3 + h_4) & h_4 & \cdots & 0 & 0 & 0\\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots\\
0 & 0 & 0 & 0 & \cdots & h_{N-2} & 2(h_{N-2} + h_{N-1}) & h_{N-1}\\
0 & 0 & 0 & 0 & \cdots & 0 & h_{N-1} & 2(h_{N-1} + h_N)
\end{bmatrix};$$

$$\overline{c} = \begin{bmatrix} c_1\\ c_2\\ c_3\\ \cdots\\ c_{N-2}\\ c_{N-1} \end{bmatrix};\ \overline{v} = \begin{bmatrix} v_1\\ v_2\\ v_3\\ \cdots\\ v_{N-2}\\ v_{N-1} \end{bmatrix}.$$

Як видно із аналізу матриці H рівняння $H\overline{c} = \overline{v}$ має тридіагональну структуру. Такі рівняння носять назву *стрічкових систем* і для їх розв'язку метод Гауса трансформується у більш ефективний метод, який називається *методом прогонки.*

Припустимо, що існують такі набори чисел δ_k і λ_k, за яких

$$c_{k-1} = \delta_{k-1} c_k + \lambda_{k-1}. \qquad (9.35)$$

В останній рекурентній формулі індекс k зменшимо на одиницю

$$c_{k-2} = \delta_{k-2} c_{k-1} + \lambda_{k-2}$$

і отримане значення c_{k-2} підставимо в рівняння (9.33). У результаті отримаємо

$$c_{k-1}\left(h_{k-1}\delta_{k-2} + 2\left(h_{k-1} + h_k\right)\right) = v_{k-1} - h_{k-1}\lambda_{k-2} - h_k c_k.$$

Із останнього рівняння знаходимо

$$c_{k-1} = -\frac{h_k}{h_{k-1}\delta_{k-2} + 2\left(h_{k-1} + h_k\right)} c_k + \frac{v_{k-1} - h_{k-1}\lambda_{k-2}}{h_{k-1}\delta_{k-2} + 2\left(h_{k-1} + h_k\right)}. \qquad (9.36)$$

Порівнюючи між собою вирази (9.35) і (9.36) приходимо до висновку, що

$$\delta_{k-1} = -\frac{h_k}{h_{k-1}\delta_{k-2} + 2\left(h_{k-1} + h_k\right)},$$

$$\lambda_{k-1} = \frac{v_{k-1} - h_{k-1}\lambda_{k-2}}{h_{k-1}\delta_{k-2} + 2\left(h_{k-1} + h_k\right)},\ k = \overline{2,N}, \qquad (9.37)$$

де $\delta_0 = 0$.

Таким чином, застосування методу прогонки до розв'язку системи рівнянь (9.34) зводиться до обчислення коефіцієнтів *прямої прогонки* δ_{k-1} і λ_{k-1} за формулами (9.37) при $k = \overline{2,N}$, а потім до одержання значень c_k *зворотної прогонки* за формулою (9.35), вважаючи, що $k = N, N-1, \ldots, 2$. Після підставленняи знайдених значень c_k у рівняння (9.31), (9.32) отримуємо числові значення коефіцієнтів кубічного сплайну b_k і d_k. Коефіцієнти a_k визначаються із умови

$$a_k = f_k,\ k = \overline{1,N}.$$

Всі обчислювальні затрати на побудову природного сплайну, який складається із N ланок складуть $O(N)$ арифметичних операцій.

Всі розрахункові формули значно спрощуються, якщо кубічний сплайн $q(x)$ будується на системі рівновіддалених вузлів - $h = x_k - x_{k-1} = const$ для всіх значень $k = \overline{1,N}$. У такому випадку замість *роздільної різниці* $\Delta(f_{k-1}, f_k)$ будемо мати *кінцеві різниці* функції $f(x)$ - $\Delta f_{k-1} = f_k - f_{k-1}$, $k = \overline{1,N}$.

Таким чином, рекурентні формули прямої і зворотної прогонки для обчислення коефіцієнтів кубічного сплайну при $h = const$ набудуть такого вигляду:

$$\delta_{k-1} = -\frac{1}{\delta_{k-2} + 4},\ \lambda_{k-1} = \frac{v_{k-1}/h - \lambda_{k-2}}{\delta_{k-2} + 4};\ \delta_0 = 0,\ \lambda_0 = 0,\ k = \overline{2,N},$$

де $v_{k-1}/h = \frac{3}{h^2}\Delta^2 f_{k-2}$;

$$c_{k-1} = \delta_{k-1}c_k + \lambda_{k-1},\ c_N = 0,\ k = N, N-1, \ldots, 2;$$

$$a_k = f_k;\ b_k = \frac{\Delta f_{k-1}}{h} + \frac{h}{3}\left(2c_k + c_{k-1}\right);\ d_k = \frac{c_k - c_{k-1}}{3h};\ k = \overline{1,N}.$$

Оскільки $\Delta^2 f_{k-2} = \Delta f_{k-1} - \Delta f_{k-2}$, а $\Delta f_{k-1} = f_k - f_{k-1}$; $\Delta f_{k-2} = f_{k-1} - f_{k-2}$, то $\Delta^2 f_{k-2} = f_k - 2f_{k-1} + f_{k-2}$.

У результаті функція $f(x)$ на відрізку $[x_{k-1}; x_k]$ апроксимується кубічним сплайном (9.18) з знайденими коефіцієнтами a_k, b_k, d_k і c_k з похибкою $O\left(h^4\right)$.

9.4.3 Наближення функцій методом найменших квадратів

При вивченні деяких закономірностей на основі спостережень або експериментів отримують початковий матеріал, який є наближеним через похибки вимірювань, неповторюваності умов спостережень, випадкових помилок і. т. п. Звідси природне прагнення дослідника до накопичення якомога більше експериментального матеріалу, для того щоб певним чином його усереднити і отримати необхідні значення параметрів закономірностей, що вивчаються.

Припустимо, що між величинами x_1, x_2, …, x_m і y існує певний функціональний зв'язок - $y = \varphi(x_1, x_2, \ldots, x_m)$, який нам невідомий. У таких випадках проводиться серія дослідів, які дають змогу отримати початкові емпіричні дані, які можна подати у вигляді двох матриць

$$X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_m^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_m^{(2)} \\ \cdots & \cdots & \cdots & \cdots \\ x_1^{(N)} & x_2^{(N)} & \cdots & x_m^{(N)} \end{bmatrix},\ Y = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_N \end{bmatrix},$$

де X - матриця спостережень;

Y - вектор експериментальних даних величини y;

N - кількість спостережень $(N \geq m)$.

$x_i^{(j)}$, y_j - значення величин x_i. $i = \overline{1,m}$ та y в j-тому спостереженні ($j = \overline{1,N}$).

Величини x_1, x_2, …, x_m носять назву *вхідних величин* досліджуваного об'єкта, а y - *вихідних величин*. Залежність $y = \varphi(x_1, x_2, \ldots, x_m)$ називають *математичною моделлю* об'єкта. Ця залежність об'єктивно існує, але вона, як правило, невідома досліднику. Тому її апроксимують вибраною певним чином іншою функцією. Найчастіше такою функцією, яка апроксимує залежність $y = \varphi(x_1, x_2, \ldots, x_m)$, виступає функція, яка є лінійною відносно своїх параметрів

$$\eta(\overline{x}, \overline{a}) = a_0 f_0(\overline{x}) + a_1 f_1(\overline{x}) + \cdots + a_n f_n(\overline{x}), \qquad (9.38)$$

де $\overline{x} = (x_1, x_2, \ldots, x_m)$ - вектор вхідних величин;

$\overline{a} = (a_0, a_1, \ldots a_n)^T$ - вектор параметрів моделі;

$f_0(\overline{x})$, $f_1(\overline{x})$, …, $f_n(\overline{x})$ - відомі функції.

Залежність $\eta(\bar{x},\bar{a})$, яка носить назву *регресійної моделі*, подамо у більш компактній формі

$$\eta(\bar{x},\bar{a}) = \sum_{i=0}^{n} a_i f_i(\bar{x}). \qquad (9.39)$$

Допустимо, що вектор параметрів моделі (9.39), вибраний певним чином. Тоді на точках матриці експерименту X можна обчислити значення $\hat{y}_k = \eta(\bar{x}^{(k)},\bar{a})$, $k = \overline{1,N}$. У результаті отримуємо систему рівнянь,

$$\hat{y}_1 = a_0 f_0(\bar{x}^{(1)}) + a_1 f_1(\bar{x}^{(1)}) + \cdots + a_n f_n(\bar{x}^{(1)}),$$
$$\hat{y}_2 = a_0 f_0(\bar{x}^{(2)}) + a_1 f_1(\bar{x}^{(2)}) + \cdots + a_n f_n(\bar{x}^{(2)}),$$
$$\hat{y}_N = a_0 f_0(\bar{x}^{(N)}) + a_1 f_1(\bar{x}^{(N)}) + \cdots + a_n f_n(\bar{x}^{(N)}),$$

яку подамо у матрично-векторній формі

$$\hat{Y} = F\bar{a}, \qquad (9.40)$$

де

$$F = \begin{bmatrix} f_0(\bar{x}^{(1)}) & f_1(\bar{x}^{(1)}) & \cdots & f_n(\bar{x}^{(1)}) \\ f_0(\bar{x}^{(2)}) & f_1(\bar{x}^{(2)}) & \cdots & f_n(\bar{x}^{(2)}) \\ \cdots & \cdots & \cdots & \cdots \\ f_0(\bar{x}^{(N)}) & f_1(\bar{x}^{(N)}) & \cdots & f_n(\bar{x}^{(N)}) \end{bmatrix}; \quad \hat{Y} = \begin{bmatrix} \eta(\bar{x}^{(1)},\bar{a}) \\ \eta(\bar{x}^{(2)},\bar{a}) \\ \cdots \\ \eta(\bar{x}^{(N)},\bar{a}) \end{bmatrix}.$$

Таким чином, при відомій матриці F вихід моделі $\hat{Y}$ залежить тільки від вибору параметрів $\bar{a}$ регресійної моделі (9.39). Параметри моделі (9.39) будемо вибирати такими, щоб сума квадратів відхилень значень виходу $\hat{Y}$ моделі, які обчислені на точках матриці X, від відповідних даних y_i, $i = \overline{1,N}$ експериментального дослідження була б мінімальною. Іншими словами, критерієм вибору параметрів моделі (9.39) буде вираз

$$J(\bar{a}) = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2, \qquad (9.41)$$

який необхідно мінімізувати за вектор-параметром $\bar{a}$.

Оскільки мінімізується сума квадратів відхилень $\varepsilon_i = y_i - \hat{y}_i$ за вектор-параметром $\bar{a}$, то відповідна процедура носить назву *методу найменших квадратів* (МНК).

Знайдемо оцінку $\hat{a}$, яка мінімізує (9.41) за умови, що $m \le N$. Для цього критерій (9.41) подамо як квадрат евклідової норми вектора $Y - \hat{Y}$, тобто

$$J(\bar{a}) = (Y - \hat{Y})^T (Y - \hat{Y}).$$

Якщо врахувати значення $\hat{Y}$, яке визначається формулою (9.40), то отримаємо

$$J(\bar{a}) = (Y - F\bar{a})^T (Y - F\bar{a}).$$

В останньому виразі розкриємо дужки і врахуємо те, що $Y^T(F\bar{a}) = (F\bar{a})^T Y$ і $(F\bar{a})^T = \bar{a}^T F^T$. У результаті приходимо до такого рівняння

$$J(\bar{a}) = Y^T Y - 2Y^T F\bar{a} + \bar{a}^T M\bar{a}, \qquad (9.42)$$

де $M = F^T F$ - так звана матриця Фішера.

Знайдемо мінімум функції (9.42) із умови $\frac{\partial J(\bar{a})}{\partial \bar{a}} = 0$. Виконавши формально операцію диференціювання скалярної функції за її вектор-аргументом, отримаємо матрично-векторне рівняння

$$M\bar{a} = F^T Y. \qquad (9.43)$$

Якщо матриця M не вироджена, то рівняння (9.43) має єдиний розв'язок

$$\bar{a} = M^{-1} F^T Y. \qquad (9.44)$$

Приклад 9.2. Припустимо, необхідно знайти лінійну залежність між величинами y і x - $y = a_0 + a_1 x$. Параметри a_1 і a_2 моделі необхідно визначити, спираючись на результати експерименту, який отриманий у вигляді пар значень (x_i, y_i), $i = \overline{1, N}$.

У даному випадку матриця F і вектор спостережень за вихідною величиною y матимуть такий вигляд:

$$F = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \cdots & \cdots \\ 1 & x_N \end{bmatrix}, \quad Y = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_3 \end{bmatrix}.$$

Знайдемо матрицю M і обернену до неї матрицю

$$M = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_N \end{bmatrix} \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \cdots & \cdots \\ 1 & x_N \end{bmatrix} = \begin{bmatrix} N & \sum_{i=1}^{N} x_i \\ \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 \end{bmatrix} \text{ і}$$

$$M^{-1} = \frac{1}{N\sum_{i=1}^{N} x_i^2 - \left(\sum_{i=1}^{N} x_i\right)^2} \begin{bmatrix} \sum_{i=1}^{N} x_i^2 & -\sum_{i=1}^{N} x_i \\ -\sum_{i=1}^{N} x_i & N \end{bmatrix}.$$

Підставивши значення F Y і M^{-1} у формулу (9.44), знайдемо МНК-оцінки $\hat{a}_1$ і $\hat{a}_2$ параметрів a_1 та a_2 лінійної моделі $y = a_0 + a_1 x$ (лінійної регресії). Маємо

$$\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \frac{1}{N\sum_{i=1}^{N} x_i^2 - \left(\sum_{i=1}^{N} x_i\right)^2} \begin{bmatrix} \sum_{i=1}^{N} x_i^2 & -\sum_{i=1}^{N} x_i \\ -\sum_{i=1}^{N} x_i & N \end{bmatrix} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_N \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_3 \end{bmatrix}.$$

Після множення матриць у правій частині останньої рівності отримаємо

$$\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \frac{1}{N\sum_{i=1}^{N} x_i^2 - \left(\sum_{i=1}^{N} x_i\right)^2} \begin{bmatrix} \sum_{i=1}^{N} x_i^2 \sum_{j=1}^{N} y_j - \sum_{i=1}^{N} x_i \sum_{j=1}^{N} x_j y_j \\ N\sum_{j=1}^{N} x_j y_j - \sum_{i=1}^{N} x_i \sum_{j=1}^{N} y_j \end{bmatrix}.$$

Із останнього матричного рівняння знаходимо, що

$$a_0 = \frac{\sum_{i=1}^{N} x_i^2 \sum_{j=1}^{N} y_j - \sum_{i=1}^{N} x_i \sum_{j=1}^{N} x_j y_j}{N\sum_{i=1}^{N} x_i^2 - \left(\sum_{i=1}^{N} x_i\right)^2}, \quad a_1 = \frac{N\sum_{j=1}^{N} x_j y_j - \sum_{i=1}^{N} x_i \sum_{j=1}^{N} y_j}{N\sum_{i=1}^{N} x_i^2 - \left(\sum_{i=1}^{N} x_i\right)^2}. \qquad (9.45)$$

Приклад 9.4. Для даних, які наведені у табл. 9.2, знайти МНК-оцінки параметрів лінійної моделі $y = a_0 + a_1 x$.

Введемо нову змінну $z = x - x_0$. Тоді $x = z + x_0$ і $y = \alpha_0 + a_1 z$, де $\alpha_0 = a_0 + a_1 x_0$. Візьмемо $x_0 = 2$ і обчислимо значення нової змінної z. Результати таких обчислень заносимо до табл. 9.2.

Таблиця 9.2 – Вихідні дані до прикладу 9.3

x	0	1	2	3	4
z	-2	-1	0	1	2
y	0,5888	2,3853	4,3931	6,4191	8,2056

Для знаходження МНК-оцінок у формулах (9.45) x замінимо на z. Тоді

$$\alpha_0 = \frac{\sum_{i=1}^{N} z_i^2 \sum_{j=1}^{N} y_j - \sum_{i=1}^{N} z_i \sum_{j=1}^{N} z_j y_j}{N\sum_{i=1}^{N} z_i^2 - \left(\sum_{i=1}^{N} z_i\right)^2}, \quad a_1 = \frac{N\sum_{j=1}^{N} z_j y_j - \sum_{i=1}^{N} z_i \sum_{j=1}^{N} y_j}{N\sum_{i=1}^{N} z_i^2 - \left(\sum_{i=1}^{N} z_i\right)^2},$$

де $N = 5$.

Аналізуючи дані табл. 9.2, неважко помітити, що $\sum_{i=1}^{N} z_i = 0$. Остання умова значно спрощує формули для обчислення МНК-оцінок параметрів α_0 і a_1. Отже,

$$\alpha_0 = \frac{\sum_{j=1}^{N} y_j}{N}, \quad a_1 = \frac{\sum_{j=1}^{N} z_j y_j}{\sum_{i=1}^{N} z_i^2}.$$

Дві останні формули дають можливість знайти - $\alpha_0 = 4{,}3984$, $a_1 = 1{,}9267$. Оскільки $\alpha_0 = a_0 + a_1 x_0$, то $a_0 = \alpha_0 - a_1 x_0$ і $a_0 = 0{,}5449$.

Графік залежності $y = a_0 + a_1 x$ показаний на рис. 9.4, де «о» відмічені табличні значення y, а «+» - розрахункові значення.

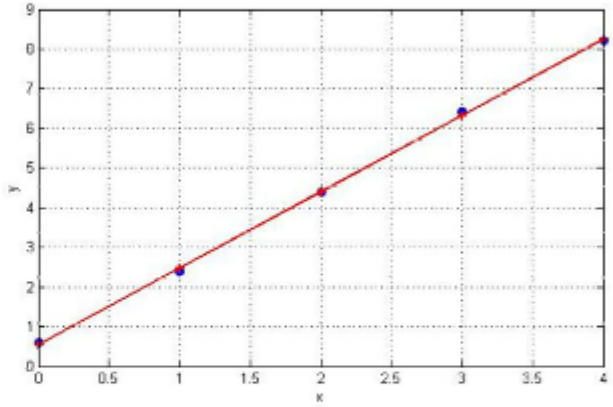


Рисунок 9.4 – Залежність $y = a_0 + a_1 x$

Знайдемо тепер суму квадратів відхилень розрахункових даних від табличних - $\sum_{i=1}^{N} (y_i - \hat{y}_i)^2 = 0{,}0204$. Отриманий результат дає змогу стверджувати про задовільну збіжність розрахункових і табличних значень y_i і $\hat{y}_i$.

Контрольні питання та завдання

1 У яких випадках виникає задача інтерполяції?

2 Яка різниця між інтерполяцією та екстраполяцією при знаходженні проміжних значень деякої функції, яка задана у вигляді таблиці?

3 З якою метою будують інтерполяційний поліном?

4 Який критерій наближення використовують при побудові інтерполяційного полінома Лагранжа?

5 Для табличної функції

x	0	1	2
y	2,0	3,5	6,0

знайти інтерполяційний поліном Лагранжа.

6 Яким чином можна застосувати наближення Лагранжа для обчислення власного інтегралу?

7 Як за допомогою наближення Лагранжа можна отримати складені формули трапецій і Сімпсрна?

8 У яких випадках застосовують кусково-поліномну апроксимацію?

9 Для даних п. 5 здійсніть кусково-лінійну та кусково-квадратичну інтерполяцію.

10 Дайте визначення сплайна дефекту d.

11 Сформуйте сукупність умов, за яких визначають коефіцієнти кубічного сплайна.

12 Наведіть алгоритм обчислення коефіцієнтів кубічного сплайну за умови, що крок h на сітці вузлів є постійним.

13 У яких випадках застосовують наближення за методом найменших квадратів?

14 Використовуючи формулу (9.44), визначити коефіцієнти рівняння лінійної регресії $y = a_0 + a_1 x$, спираючись на результати експерименту, який оформлений у вигляді пар значень x_i та y_i:

x	0	1	2	3
y	2,0049	2,5103	3,0073	3,4970

10 Методи оптимізації

10.1 Знаходження екстремальних значень функції однієї змінної

Розглянемо таку задачу. Необхідно виготовити закриту ємність, яка має форму круглого конуса (рис. 10.1). Визначити такі геометричні розміри ємності h і R (h - висота, R - радіус), щоб її об'єм був максимальним при заданій повній площі S.

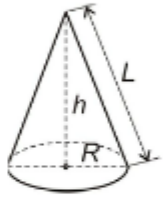


Рисунок 10.1 – Ємність у формі конуса

Об'єм круглого конуса $V = \frac{1}{3} S_{oc} h$, де $S_{oc} = \pi R^2$ - площа основи.

Повна площа конуса $S = S_{oc} + S_{біч}$, де $S_{біч} = \pi RL$ - бічна площа конуса; L - твірна конуса.

Із рис. 10.1 видно, що $h^2 = L^2 - R^2$. Отже,

$$V = \frac{1}{3} \pi R^2 \sqrt{L^2 - R^2}. \qquad (10.1)$$

Оскільки площа S відома, то $L = \frac{S - \pi R^2}{\pi R}$. Підставляючи значення L в (10.1), отримаємо

$V = \frac{R}{3}\sqrt{S^2 - 2S\pi R^2}$. Якщо увести позначення $x = R$ і $f(x) = V$, то $f(x) = \frac{x}{3}\sqrt{S^2 - 2S\pi x^2}$.

Таким чином, ми прийшли до задачі, в якій необхідно знайти максимум функції $f(x)$ відносно змінної x. Формалізований запис цієї задачі буде таким:

$$\max_x : f(x) = \frac{x}{3}\sqrt{S^2 - 2S\pi x^2}. \qquad (10.2)$$

Задача (10.2) носить назву *задачі безумовної максимізації* функції однієї змінної. У загальному вигляді будемо мати

$$\min_x : f(x). \qquad (10.3)$$

У подальшому будемо розглядати тільки *задачі безумовної мінімізації* однієї змінної, оскільки задачу максимізації завжди можна звести до задачі мінімізації (і навпаки)

$$\max_x : f(x) = -\min_x : F(x), \qquad (10.4)$$

де $F(x) = -f(x)$.

Функцію $f(x)$, яка підлягає мінімізації, називають *цільовою функцією* або *критерієм оптимальності*, а саму задачу (10.3) *задачею безумовної оптимізації.*

Необхідні і достатні умови у задачі безумовної мінімізації. Допустимо, що функція $f(x)$ неперервна на інтервалі $[a;b]$ і має першу $f'(x)$ і другу $f''(x)$ похідні. Тоді, якщо мають місце умови

$$f'(x) = 0, \qquad (10.5)$$

$$f''(x) > 0, \qquad (10.6)$$

то у точці $x = x^*$ функція $f(x)$ має *локальний мінімум.*

Кажуть, що функція $f(x)$ має локальний мінімум у точці $x = x^*$, якщо існує такий відкритий інтервал $I = (a;b)$, який вміщує x^*, що $f(x) \le f(x^*)$ для всіх $x \in I$.

Із умови (10.5) випливає, що задача безумовної мінімізації зводиться до задачі пошуку нуля функції $\varphi(x) = f'(x)$. Говорячи точніше, мова йде про задачу пошуку на кінцевому інтервалі I такої точки x^*, в якій нелінійна функція $\varphi(x)$ набуває нульового значення і змінює свій знак при переході через точку x^*.

Таку задачу ми розглядали раніше і для знаходження нуля функції $\varphi(x)$ тут можна застосувати один із методів розв'язку нелінійних рівнянь.

В окремих випадках задачу (10.3) можна розв'язати аналітично.

Приклад 10.1. Знайти розв'язок задачі (10.2).

Для цього скористаємося необхідною умовою (10.5) існування мінімума функції (10.2). Будемо мати:

$$f'(x) = \frac{S}{3} \cdot \frac{S - 4\pi x^2}{\sqrt{S^2 - 2\pi S x^2}}.$$

Прирівнюючи значення похідної до нуля, знаходимо, що

$$x = R = \frac{1}{2}\sqrt{\frac{S}{\pi}}.$$

Знаючи R, знайдемо $L = \frac{3}{2}\sqrt{\frac{S}{\pi}}$ або $L = 3R$. Отже, $h = \sqrt{\frac{2S}{\pi}}$.

Таким чином, об'єм ємності, яка має форму кругового конуса, буде максимальним, якщо $R = \frac{1}{2}\sqrt{\frac{S}{\pi}}$ і $h = \sqrt{\frac{2S}{\pi}}$.

Числові методи розв'язку задачі безумовної мінімізації. Методи безумовної мінімізації функції однієї змінної багато в чому аналогічні розглянутим раніше процедурам пошуку нуля функції $f'(x)$. Недоліком такого підходу є те, що доводиться шукати похідну від функції $f(x)$. У багатьох випадках це може бути досить громіздкою процедурою. Тому доцільно для розв'язку задачі (10.3) звертатись до методів, які спеціально призначенні для задач оптимізації.

Такі методи не вимагають знання похідної *f'(x)* і працюють за принципом послідовного скорочення початкового інтегралу $I\ (x^* \in I)$ і ґрунтується на деякій умові, яка дає можливість визничити відрізок локалізації мінімума функції, якщо вона *унімодальна.*

Функція $f(x)$ унімодальна на відрізку $I = [a;b]$, якщо існує єдина точка $x^* \in I$ така, що для будь-яких $x_1,\ x_2 \in I$ і $x_1 < x_2$ (рис. 10.2)

$f(x_1) > f(x_2)$, якщо $x_2 < x^*$;

$f(x_1) < f(x_2)$, якщо $x_1 > x^*$.

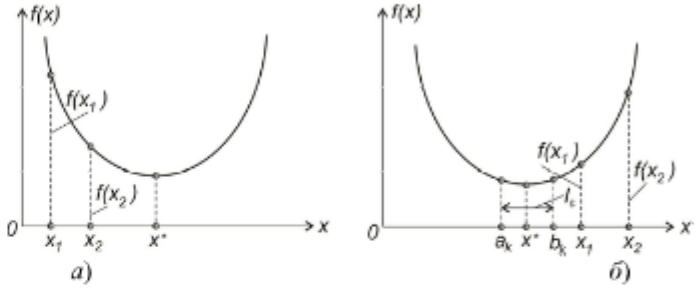


Рисунок 10.2 – До визначення унімодальної функції
а) $x_2 < x^*$; **б)** $x_1 > x^*$

Якщо відомо, що функція $f(x)$ унімодальна на відрізку I, то можна послідовним зменшенням довжини початкового інтервалу I знайти інтервал I_ε (рис. 10.2,б), що вміщує точку x^*, для якого виконується умова $f(a_k) - f(x^*) < \varepsilon$ або $f(b_k) - f(x^*) < \varepsilon$, де $I_\varepsilon = [a_k; b_k]$, $x^* \in I$, $\varepsilon > 0$ - додатне число, яке визначає точність розв'язку задачі (10.2).

Пошук Фінабоначчі. Ця стратегія базується на числах Фібоначчі F_i, які обчислюються за допомогою рекурентного співвідношення

$$F_i = F_{i-1} + F_{i-2},\ F_0 = 1,\ F_1 = 2,\ i = 2, 3, \ldots .$$

Початковими числами послідовності будуть 1, 2, 3, 5, 8, 13, ... Нехай задана похибка $\varepsilon > 0$ у визначенні мінімуму

функції на відрізку $[a;b]$. Визначимо n-те число Фібоначчі за правилом

$$F_\varepsilon = (b-a)\varepsilon^{-1},\ F_{n-1} < F_\varepsilon < F_n,$$

де F_ε – проміжне число.

Тоді для пошуку положення мінімуму функції $f(x)$ на відрізку $[a;b]$ з абсолютною похибкою, яка не перевищує ε, досить обчислити n значень функції $f(x)$.

Алгоритм пошуку, в якому використовуються числа Фібоначчі, складається з таких кроків.

К 1. Визначення оптимального кроку пошуку:

$$\varepsilon_n = (b-a)F_n^{-1}.$$

К 2. Обчислення значення функції $f(x)$ на початку інтервалу - $f(a)$.

К 3. Знаходження наступної точки $x_1 = a + \varepsilon_n F_{n-2}$, в якій обчислюється нове значення функції $f(x_1)$.

К 4. При вдалому кроці, коли $f(x_1) < f(a)$, наступна точка визначається як

$$x_2 = x_1 + \varepsilon_n F_{n-3}.$$

У протилежному ж випадку (крок невдалий)

$$x_2 = x_1 - \varepsilon_n F_{n-3}.$$

К 5. Подальші кроки виконуються аналогічно, і на k-тій ітерації при вдалому кроці

$$x_{k+1} = x_k + \varepsilon_n F_{n-k-2}$$

при невдалому

$$x_{k+1} = x_k - \varepsilon_n F_{n-k-2}.$$

Указаний процес обчислення продовжується доти, поки не вичерпаються всі числа Фібоначчі з послідовності

$$F_{n-k-2} = F_{n-k} - F_{n-k-1},\ k=0,1,2,\ \ldots\ .$$

Приклад 10.2. Знайти мінімум функції $f(x) = e^{-x} + 2x^2$ з використанням пошуку Фібоначчі.

Для розв'язку задачі $\min\limits_x : f(x)$ скористаємося процедурою `minFibonachy`.

```
minFibonachy*
▷Пошук мінімуму функції f(x) методом Фібначчі на
інтервалі
▷ [a;b], де f(x)- унімодальна функція
▷Вхід:[a;b]-початковий інтервал (вибирається за
графіком)
▷        eps-число, яке визначає точність розв'язку
задачі
▷      min:f(x)
▷Вихід:x_opt-значення x, при якому функція f(x)
набуває
▷             мінімального значення
▷       f_opt-мінімальне значення функції f(x)
▷Фунуція, яка підлягає мінімізації задати у під
процедурі F_min
▷Побудувати графік функції f(x) та визначити інтервал
[a;b], на якому функція
 f(x) унімодальна
1 Fm=abs((b-a)/eps);
2 F(1)=1
3 F(2)=1
4 i=2
▷Обчислення n-го числа Фібоначчі
5 while Fm>F(i)
6      i=i+1
7    F(i)=F(i-1)+F(i-2)
8 end while
```

```
▷ Реалізація пошуку Фібоначчі
9 n=length(F)
10 em=abs((b-a)/F(n))
11 fa=fun_minF(a)
12 x=a
13 x=x+em*F(n-2)
14 fx=fun_minF(x)
15 k=3
16 while (k<n-2)&(abs(fa-fx))>eps
17    if then fa>fx
18         fa=fx
19         c=x
20         x=x+em*F(n-k)
21         fx=fun_minF(x)
22         k=k+1
23    else
24         fa=fx
25         x=x-em*F(n-k)
26         fx=fun_minF(x)
27         em=-em;
28    end for
29 end while
30 return x,fx
```

Були вибрані такі параметри програми: $a=0$, $b=0,5$, $\varepsilon=10^{-10}$. Результат пошуку мінімуму функції - $x^*=0,2039$, $f\left(x^*\right)=0,8987$.

Метод золотого січення. Недоліком методу Фібоначчі є те, що він оперує лише з додатніми інтервалами. Крім того пошук Фібоначчі важко налаштувати на критерій зупину, який вимагає, щоб значення функції на кінцевому інтервалі I_ε відрізнялось би від I_ε не більше, ніж на задану величину ε.

Через наведені недоліки пошуку Фібоначчі замість нього використовують метод, який носить назву *золотого січення*.

Відрізок довжиною l (рис 10.3) поділений точкою r у пропорції *золотого січення*, якщо відношення більшої частини відрізка l_1 до його меншої частини l_2 дорівнює відношенню довжини відрізка l до його більшої частини l_1.

$$\frac{l_1}{l_2}=\frac{l}{l_1}.$$

Оскільки $l_2=l-l_1$, то

$$\frac{l}{l-l_1}=\frac{l}{l_1}.$$

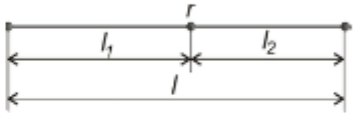


Рисунок 10.3 - Поділ відрізка у пропорції золотого січення

Отже значення l_1 є розв'язком рівняння

$$l_1^2+ll_1-l^2=0,$$

тобто

$$l_1=l\frac{\sqrt{5}-1}{2}.$$

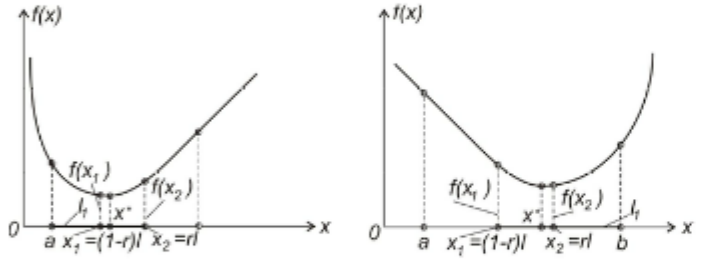
Знаючи l_1 знайдемо $l_2=l-l_1=l\frac{3-\sqrt{5}}{2}$.

Таким чином, точка r ділить відрізок довжиною l у такій пропорції: $r=\frac{l_1}{l}=\frac{\sqrt{5}-1}{2}=0,61803$ і

$1-r=\frac{l_2}{l}=\frac{3-\sqrt{5}}{2}=0,38197.$

Для пошуку методом золотого січення необхідно, щоб першими пробними внутрішніми точками були $x_1 = (1-r)l$ і $x_2 = rl$.

Нехай початковий інтервал $I = [a;b]$. Тоді у випадку, коли $f(x_1) \le f(x_2)$, то стискування інтервалу I буде справа (рис. 10.4, а) і наступним інтервалом буде $I_1 = [a;x_2]$. Якщо $f(x_1) > f(x_2)$, то інтервал I стискуємо зліва і $I_1 = [x_1;b]$ (рис. 10.4, б).



а) *б*)

Рисунок 10.4 – Процес вибору нового інтервала

Отже, коли $f(x_1) \le f(x_2)$ вибираємо інтервал $[a;x_2]$ і змінюємо x_2 на b. Якщо $f(x_1) > f(x_2)$, то мінімум знаходиться на інтервалі $[x_1;b]$ і x_1 необхідно замінити на a і продовжити пошук.

Таким чином, ліва (рис. 10.4, а) або права (рис. 10.4, б) частина інтервалу I буде новим інтервалом I_1 для наступної ітерації.

Закінченням пошуку є виконання умови $|x_2 - x_1| \le \varepsilon$, де ε – задана похибка пошуку x^*. При виконанні останньої умови обчислюють $x^* \approx \frac{x_1 + x_2}{2}$ і $f(x^*)$.

Приклад 10.3 За допомогою методу золотого січення знайти мінімум функції $f(x) = e^{-x} + x^2$. Для розв'язку задачі скористаємося процедурою `GoldCut`.

```
GoldCut*
▷Пошук мінімуму унімодальної функції однієї змінної
методом
▷золотого січення. Функція f(x) повинна бути
визначена у
▷підпроцедурі fun_minF
▷Вхід:del-похибка пошуку x
▷      epselon-допустиме відхилення для ординат
▷      x0-стартова точка для x
▷       delta_x-крок у підпрограмі пошуку початкового
інтервалу
%Вихід:xs-точка мінімуму функції
▷       fs-мінімальне значення функції
▷Визначити початковий інтервал [a;b], на якому
функція f(x) унімодальна
▷Пошук мінімуму функції методом золотого січення
1 r=(sqrt(5)-1)/2
2 r1=r^2
3 if x0>x
4    then xii=x
5    x=x0
6    x0=xii
7 end if
8 a=x0
9 b=x
10 L=b-a
12 ll x1=a+r1*L
13 x2=a+r*L
14 f1=fun_minF(x1)
15 f2=fun_minF(x2)
```

```
fa=fun_minF(a)
fb=fun_minF(b)
while (abs(fb-fa)>epselon)|(L>del)
    if f1<f2
       then b=x2
       fb=f2
       x2=x1
       f2=f1
       L=b-a
       x1=a+r1*L
       f1=fun_minF(x1)
   else
       a=x1
       fa=f1
       x1=x2
       f1=f2
       L=b-a
       x2=a+r*L
       f2=fun_minF(x2)
   end if
end for
xs=(x1+x2)/2
```

10.2 Знаходження екстремальних значень функції багатьох змінних

Методи знаходження екстремальних значень функції багатьох змінних

$$min\left(max\right): f\left(\overline{x}\right),\ \overline{x} \in E^n, \tag{10.7}$$

де E^n - n- вимірний евклідовий простір, можна розбити на дві групи.

Перша з них – це методи, в яких не використовуються похідні. До другої належать методи, які використовують похідні для пошуку екстремальних значень функції $f\left(\overline{x}\right)$, де $\overline{x}^T = (x_1, x_2, ..., x_n)$- вектор змінних, який належить n-вимірному евклідовому простору E^n.

10.2.1 Безградієнтні методи. Особливістю цих методів є те, що для їх реалізації не потрібно знати похідні функції $f\left(\overline{x}\right)$. Ця перевага особливо відчутна у задачах з досить великим числом змінних, коли одержати похідні у вигляді аналітичних функцій досить складно. Крім того, методи цієї групи рекомендовано застосовувати тоді, коли функції в деяких точках не мають похідних або мають розриви. Недолік цих методів – більш уповільнена швидкість збіжності порівняно із методами, які оперують з першими та другими похідними функції $f\left(\overline{x}\right)$.

Симплексний метод. В основі цього методу лежить поняття симплексу. Нагадаємо, що симплексами є регулярні багатокутники в E^n. Наприклад, для n = 2 регулярний симплекс – це рівносторонній трикутник (три точки), а для n= 3 регулярний симплекс – це тетраедр (чотири точки) і т.д.

Координати вершин регулярного симплексу визначаються матрицею D,

$$\begin{bmatrix} 0 & d_1 & d_2 & d_2 & \dots & d_2 \\ 0 & d_2 & d_1 & d_2 & \dots & d_2 \\ & & \dots\dots\dots\dots & & & \\ 0 & d_2 & d_2 & d_2 & \dots & d_1 \end{bmatrix}. \tag{10.8}$$

в якої стовпці – вершини, пронумеровані від 1 до $n+1$, а лінійки – їх координати. Загальна кількість лінійок дорівнює числу незалежних змінних x_i, $i = \overline{1, n}$.

Елементи матриці D визначаються за формулами

$$d_1 = l_c (n\sqrt{2})^{-1}(\sqrt{n+1}+n-1),\ d_2 = l_c (n\sqrt{2})^{-1}(\sqrt{n+1}-1), \qquad (10.9)$$

де l_c відстань між вершинами симплексу.

Матриця D_c визначає такий сиплекс, одна з вершин якого перебуває на початку координат.

Щоб побудувати симплекс з вершиною в точці $\overline{x}^{(i)}$, досить координати всіх вершин симплекса змістити на величину $\overline{x}^{(i)}$, $i = \overline{1,n}$.

Слід відзначити, що навпроти будь-якої вершини симплексу розміщена тільки одна грань, на якій можна побудувати новий симплекс, що відрізнятиметься від попереднього тільки новою вершиною, тоді як решта вершин старого і нового симплексів збігаються. Саме ця властивість симплексу зумовила його широке застосування при розв'язуванні задач мінімізації. Ідею алгоритму у розв'язуванні задач симплексним методом розглянемо на прикладі мінімізації функції двох змінних, лінії постійного рівня* якої зображені на рис. 10.5.

Обчислимо цільову функцію в кожній з вершин симплексу (рис. 10.5). Будуємо новий симплекс, який включає одну нову вершину (точка 4), розміщену на проектуючій прямій (на рис. 10.5 - пунктирна пряма) симетрично відносно вершини 3, у якій цільова функція набуває максимального значення. Ця операція називається відображенням, а новий симплекс – відображеним. Продовження цієї процедури, в якій кожний раз викреслюється вершина, де цільова функція максимальна, дає змогу визначити окіл мінімуму функції. З рис. 10.5 випливає, що поблизу від точки мінімуму може виникнути зациклення (точки 10, 13). Для усунення цього явища змінюють розміри початкового симплексу до його зменшення. Критерієм закінченням пошуку можуть бути розміри симплексу: пошук припиняють, якщо всі ребра симплексу стануть меншими наперед заданої достатньо малої достатньої величини.

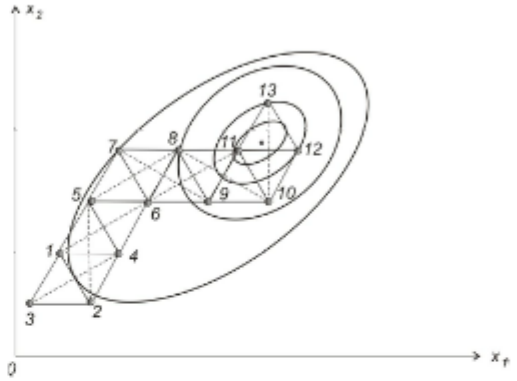


Рисунок 10.5 – Пошук мінімуму функції $f(\overline{x})$ методом многокутників

Пошук за деформованим многокутником. Цей метод дає змогу уникнути зациклення поблизу точки мінімуму функції і прискорити процес обчислень. Початковий многокутник, як правило, вибирається у вигляді регулярного симплексу, координати вершин якого визначаються матрицею D. Потім вершина (точка) в E^n, в якій значення функції $f(\overline{x})$ максимальне, проектується через центр грані. Більш низькі значення цільової функції знаходять послідовною заміною точки з максимальним значенням $f(\overline{x})$ на більш «привабливі» точки. При цьому початковий симплекс належним чином деформується для того, щоб дальше від

* Лінія постійного рівня – геометричне місце точок таких, що $f(\overline{x}) = \beta_p$.

точки мінімуму прискорити рух до цієї точки, а ближче – сповільнити рух і тим самим запобігти зацикленню.

Нехай на якомусь кроці обчислень одержаний многокутник з вершинами $\overline{x}_i^{(k)}$, $i=\overline{1,n+1}$. Визначимо вершину $\overline{x}_h^{(k)}$ з найбільшим значенням цільової функції

$$f\left(\overline{x}_h^{(k)}\right)=max:\left\{f\left(\overline{x}_1^{(k)}\right),f\left(\overline{x}_2^{(k)}\right),\dots,f\left(\overline{x}_{n+1}^{(k)}\right)\right\}$$

і вершину з найменшим значенням $f(\overline{x})$

$$f\left(\overline{x}_l^{(k)}\right)=min:\left\{f\left(\overline{x}_1^{(k)}\right),f\left(\overline{x}_2^{(k)}\right),\dots,f\left(\overline{x}_{n+1}^{(k)}\right)\right\}.$$

Далі пошук вершин в E^n, в якій $f\left(\overline{x}\right)$ має мінімальне значення скадається з наступних кроків:

К1. Відображення (рис. 10.6) – проектування вершини $\overline{x}_h^{(k)}$, в якій функція $f\left(\overline{x}\right)$ має найбільше значення, через центр протилежної грані симплексу:

$$\overline{x}_{n+3}^{(k)}=\overline{x}_{n+2}^{(k)}+\alpha\left(\overline{x}_{n+2}^{(k)}-\overline{x}_h^{(k)}\right), \qquad (10.10)$$

де $\alpha>0$ - коефіцієнт відображення. Координати центру протилежної грані визначають за формулою

$$x_{n+2,j}^{(k)}=\frac{1}{n}\left[\left(\sum_{i=1}^{n+1}x_{ij}^{(k)}-x_{hj}^{(k)}\right)\right],\quad j=\overline{1,n}. \qquad (10.11)$$

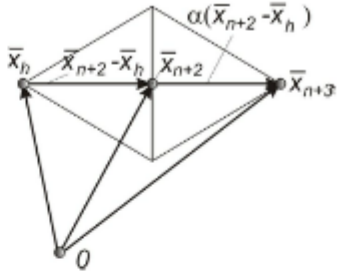


Рисунок 10.6 – Відображення

К2. Розтяг (рис. 10.7). Коли $f\left(\overline{x}_{n+3}^{(k)}\right)<f\left(\overline{x}_l^{(k)}\right)$, то вектор $\overline{x}_{n+3}^{(k)}-\overline{x}_{n+2}^{(k)}$ розтягується в γ раз відповідно зі співвідношенням

$$\overline{x}_{n+4}^{(k)}=\overline{x}_{n+2}^{(k)}+\gamma\left(\overline{x}_{n+3}^{(k)}-\overline{x}_{n+2}^{(k)}\right), \qquad (10.12)$$

де $\gamma>1$ - коефіцієнт розтягу.

Якщо $f\left(\overline{x}_{n+4}^{(k)}\right)<f\left(\overline{x}_l^{(k)}\right)$, то $\overline{x}_h^{(k)}$ замінюють на $\overline{x}_{n+4}^{(k)}$ і процедура продовжується, починаючи з *К1*. В іншому випадку $\overline{x}_h^{(k)}$ замінюють на $\overline{x}_{n+3}^{(k)}$ і також здійснюється перехід до *К1*.

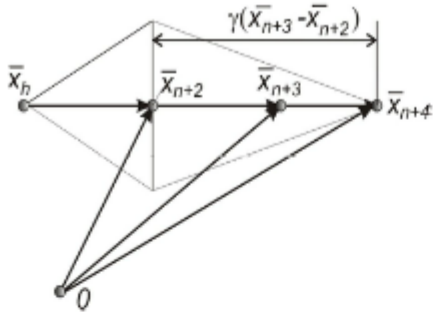


Рисунок 10.7 – Розтяг

Sp3. Стискання (рис. 10.8) Якщо $f\left(\overline{x}_{n+3}^{(k)}\right) > f\left(\overline{x}_i^{(k)}\right)$ для всіх $i = \overline{1, n+1}$, $i \neq h$, то вектор $\overline{x}_h^{(k)} - \overline{x}_{n+2}^{(k)}$ стискається у β раз відповідно з формулою

$$\overline{x}_{n+5}^{(k)} = \overline{x}_{n+2}^{(k)} + \beta\left(\overline{x}_h^{(k)} - \overline{x}_{n+2}^{(k)}\right), \qquad (10.13)$$

де $0 < \beta < 1$ - коефіцієнт стиску.

Потім $\overline{x}_h^{(k)}$ замінюють на $\overline{x}_{n+5}^{(k)}$ і повертаються до *K1* для продовження пошуку на $k+1$ кроці.

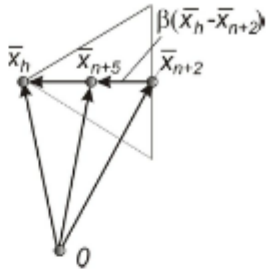


Рисунок 10.8 – Стиснення

K4. Редукція (рис. 10.9) Якщо у результаті виконання операції 1 значення функції $f\left(\overline{x}\right)$ не зменшилось $f\left(\overline{x}_{n+3}^{(k)}\right) > f\left(\overline{x}_h^{(k)}\right)$, тоді всі вектори $\overline{x}_i^{(k)} - \overline{x}_l^{(k)}$, $i = \overline{1, n+1}$ $(i \neq l)$ зменшують у два рази у відповідності за формулою:

$$\overline{x}_i^{(k)} = \overline{x}_l^{(k)} + \frac{1}{2}\left(\overline{x}_i^{(k)} - \overline{x}_l^{(k)}\right), \quad i \neq l, i = \overline{1, n+1}. \qquad (10.14)$$

Потім повертаються до операції 1 для продовження пошуку на $k+1$ кроці.

Критерій закінчення ітераційного процесу полягає у перевірці умови:

$$\left(\frac{1}{n+1}\sum_{i=1}^{n+1}\left(f\left(\overline{x}_i^{(k)}\right) - f\left(\overline{x}_{n+2}^{(k)}\right)\right)^2\right)^{\frac{1}{2}} \leq \varepsilon, \qquad (10.15)$$

де ε - додатнє, що визначає точність розв'язку задачі (10.7). Рекомендоване значення коефіцієнта відображення $\alpha = 1$, а

значення коефіцієнта розтягу γ і стиснення β лежать у межах:

$$2{,}8 \le \gamma \le 3, \quad 0{,}4 \le \beta \le 0{,}6.$$

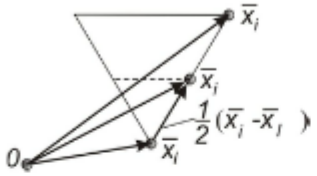


Рисунок 10.9 – Редукція

Приклад 10.4. Мінімізувати функцію двох змінних

$$f(\overline{x}) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1^2\right),$$

яка відома як функція Розенброка (рис. 10.10) і визначає у тривимірному просторі поверхню, яка має вигляд серпоподібної ущелини (рис. 10.10) з точкою мінімума $x^{*T} = (1;1)$. Для розв'язку задачі скористаємося програмою процедурою `MinNelderMead`.

```
MinNelderMead*
▷ Мінімізація функції f(x) багатьох змінних методом
Нелдера
▷  Міда. Функція f(x), яка підлягає мінімізації,
визначається
▷ підпроцедурою fun_NelderMead
▷ Вхід: N-розмірність вектора змінних
▷       t-довжина ребра симплекса
▷       alfa, beta, gama-параметри алгоритму
▷       N_max-максимальне число ітерацій
▷       epselon-точність розв'язку задачі мінімізації
▷          show-показує число обчислень функції, яка
підлягає
▷             мінімізації (show=1)
▷  Вихід:x_opt-значення x, в якому функція f(x)
набуває
▷       мінімального значення
▷       f_min-мінімальне значення функції при x=x_opt
▷ Будуємо початковий симплекс, одна із вершин якого
   знаходиться на початку координат
1 sq=sqrt(N+1)
2 sq_2=sqrt(2)
3 d1=t*(sq+N-1)/(N*sq_2)
4 d2=t*(sq-1)/(N*sq_2)
5 for i=1 to N
6    for j=1 to N
7       if then i==j
8          V1(i,j)=d1
9       else
10          V1(i,j)=d2
11       end if
12     end for
13 end for
14 Сформувати нульовий вектор L розміром Nx1
15 Vd=[L V1]
16 V=Vd'
▷ Початок алгоритму Нелдера-Міда
17 ix=0
18 while (sig>epselon)&(ix<N_max)
19     ix=ix+1
▷ Обчислення xh і xl
20 for j=1 to N+1
21    for k=1 to N
22         x(j)=V(j,k)
23     end for
24    [z,tev]=fun_NelderMead(x,tev)
25    f(j)=z
26 end for
27 [fh h]=max(f)
28 [fl l]=min(f)
▷Формування вектора як l і h рядка матриці V
29 x_l=V(l,:)
30 x_h=V(h,:)
▷ Обчислення центру ваги симплексу
31 Сформувати нульовий вектор S розміром 1xN
```

```
for j=1 to N+1
    S=S+V(j,1:N)
end for
in=in+1
x_n2=(S-V(h,1:N))/N
Sn(in,:)=x_n2
▷Відображення
x_n3=x_n2+alfa*(x_n2-x_h)
 [f_xn3,tev]=fun_NelderMead(x_n3,tev)
if f_xn3<fl
  ▷Розтяг
   then x_n4=x_n2+gama*(x_n3-x_n2)
    [f_xn4,tev]=fun_NelderMead(x_n4,tev)
   if then f_xn4<fl
       x_h=x_n4
▷У матриці h-тий рядок замінюємо на вектор x_h
       V(h,:)=x_h
      [sig,tev]=fun_stop(V,x_n2,tev)
       continue
   else
       x_h=x_n3
       V(h,:)=x_h
   end if
   [sig,tev]=fun_stop(V,x_n2,tev)
   continue
else
   for j=1:N+1
       [z,tev]=fun_NelderMead(V(j,:),tev)
      f_i(j)=z
      f(j)=f_xn3
   end for
   for j=1 to N+1
       if j==h
▷У веторів f i f_i вилучається j -тий компонент
▷У рядках 62, 63 використано таке позначення: emptily
- пусто
         f(j)=emptily
      f_i(j)=emptily
   end for
end if
   if f>f_i
       if f_xn3>=fh
           x_h=x_n3
           V(h,:)=x_h
       end if
     ▷Стиснення
       x_n5=x_n2+beta*(x_h-x_n2)
       [f_xn5,tev]=fun_NelderMead(x_n5,tev)
       if f_xn5>fh
           for j=1 to N+1
             ▷Редукція
▷Обчислюємо j-тий рядок матриці S
               s(j,:)=x_1+0.5*(V(j,:)-x_1)
           end for
           V=s
           [sig,tev]=fun_stop(V,x_n2,tev)
           continue
       else
           x_h=x_n5
           V(h,:)=x_h
       end if
       [sig,tev]=fun_stop(V,x_n2,tev)
       continue
   else
       x_h=x_n3
▷h-тий рядок матриці V замінюємо вектором x_h
       V(h,:)=x_h
       [sig,tev]=fun_stop(V,x_n2,tev)
   end if
  end if
end while
▷Результат розв'язку задачі мінімізації функції f(x)
x_opt=min(V)
[f_min,tev]=fun_NelderMead(x_opt,tev)
```

Підпроцедура fun_stop

```
[sig,tev]=fun_stop(V,x_n2,tev)
N=length(x_n2)
[z,tev]=fun_NelderMead(x_n2,tev)
fx=z
s=0
for j=1 to N+1
```

```
▷ У 10 рядку V(j,:) j-тий рядок матриці V
7       [z,tev]=fun_NelderMead(V(j,:),tev)
8       s=s+(z-fx)^2
9  end
10 sig=sqrt(s/(N+1))
11 return sig, tev
```

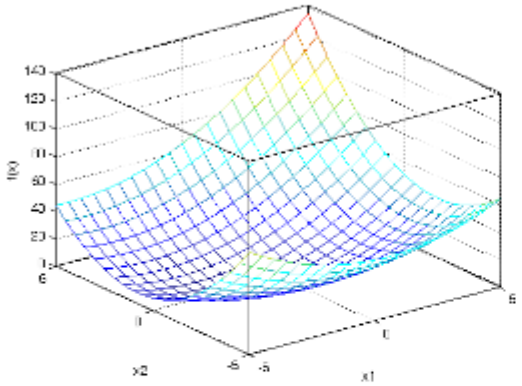


Рисунок 10.10 – Функція Розенброка

Початковим симплексом був вибраний трикутник з вершиною в точці (0;0) і довжиною ребра l_c=1.

На рис. 10.11 показана траєкторія руху точки $\bar{x}_{n+3}$ (центру симплекса) на фоні ліній рівня. Для переходу в окіл точки $\bar{x}^*$ було здійснено 1026 обчислень функції $f(\bar{x})$ при $\varepsilon = 10^{-6}$. Мінімум функції досягнутий у точці $\bar{x}^T = (0.9992;\ 0.9984)$.

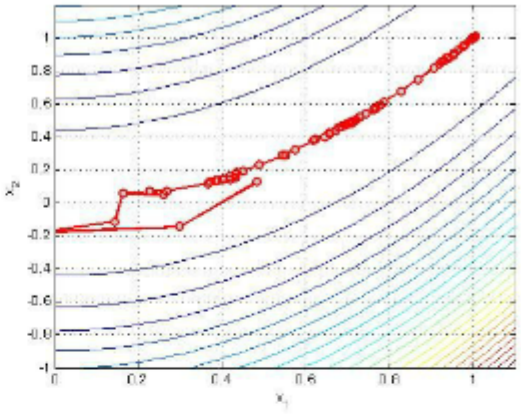


Рисунок 10.11 – Траєкторія пошуку мінімуму функції Розенброка методом Нелдера-Міда

10.2.2 Градієнтні методи. В основі градієнтних методів лежать необхідні і достатні умови існування мінімуму функції $f(\bar{x})$ багатьох змінних.

Нехай необхідно розв'язати задачу

$$min: f(\bar{x}),\quad \bar{x} \in E^n. \qquad (10.16)$$

Необхідними умовами того, що $\bar{x}^*$ - точка локального мінімуму є:

1) функція $f(\bar{x})$ має часткові похідні у точці $\bar{x}^*$;

2) градієнт функції $f(\bar{x}) = \left(\frac{\partial f(\bar{x})}{\partial x_1}, \frac{\partial f(\bar{x})}{\partial x_2}, \ldots, \frac{\partial f(\bar{x})}{\partial x_n} \right)^T$ у точці $\bar{x}^*$ є нульовим вектором.

Достатніми умовами існування розв'язкузадачі (10.16) є вимога додатної визначеності матриці Гессен

$$\nabla^2 f(\bar{x}) = \begin{bmatrix} \frac{\partial^2 f_1(\bar{x})}{\partial x_1^2} & \frac{\partial^2 f_1(\bar{x})}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 f_1(\bar{x})}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f_2(\bar{x})}{\partial x_2 \partial x_1} & \frac{\partial^2 f_2(\bar{x})}{\partial x_2^2} & \dots & \frac{\partial^2 f_2(\bar{x})}{\partial x_2 \partial x_2} \\ \dots & \dots & \dots & \dots \\ \frac{\partial^2 f_n(\bar{x})}{\partial x_n \partial x_1} & \frac{\partial^2 f_n(\bar{x})}{\partial x_n \partial x_2} & \dots & \frac{\partial^2 f_n(\bar{x})}{\partial x_n^2} \end{bmatrix}_{\bar{x}=\bar{x}^*}.$$

Визначення. Будь-яка квадратна матриця додатно визначена, якщо всі її діагональні мінори більші нуля.

Коли обчислення похідних функції $f(\bar{x})$ не становить принципових труднощів, то рекомендуються градієнтні методи мінімізації $f(\bar{x})$, які порівняно з методами прямого пошуку мають вищу швидкість збіжності. Існує велика кількість методів мінімізації, в яких для пошуку напрямку руху до мінімуму функції $f(\bar{x})$ використовують градієнт $\nabla f(\bar{x})$.

Узагальнена схема розв'язування задачі оптимізації градієнтним методом. В усіх градієнтних методах будується ітераційний процес так, що в напрямку градієнта функції $f(\bar{x})$ критерій оптимальності зменшує своє значення при переході від k до $k+1$ ітерації. Методи, які задовольняють ці вимоги, називають *методами спуску*. Усі ці методи мають таку узагальнюючу схему.

*К*1. Вибрати початкову (стартову) точку $\bar{x}^{(0)}$.

*К*2. Перевірити умови зупинок, і якщо вони виконуються, обчислення припинити і прийняти $\bar{x}^{(k)}$ (на першому кроці обчислень $k=0$) як розв'язок задачі, в іншому ж випадку перейти до наступного кроку.

*К*3. Розрахувати ненульовий n-мірний вектор $\bar{p}^{(k)}$, названий напрямком пошуку.

*К*4. Вирахувати додатне число λ_k (довжину) кроку, яке забезпечить виконання нерівності $f(\bar{x}^{(k)}) > f(\bar{x}^{(k)} + \lambda_k \bar{p}^{(k)})$. Обчислити $\bar{x}^{(k+1)} = \bar{x}^{(k)} + \lambda_k \bar{p}^{(k)}$.

*К*5. Замінити $\bar{x}^{(k+1)}$ на $\bar{x}^{(k)}$ і $k+1$ на k і перейти до кроку 2. Хоча узагальнююча схема розв'язку задачі оптимізації і гарантує одержання монотонно спадної послідовності $f(\bar{x}^{(0)}) > f(\bar{x}^1) > \dots > f(\bar{x}^{(k)}) > \dots$, це не означає, що така послідовність завжди сходиться до екстремальної точки $\bar{x}^*$.

По-перше, може бути невдало вибраною довжина кроку λ_k, по-друге, вектори $\bar{p}^{(k)}$ розраховані так, що їх напрямок збігається з дотичними до лінії рівня функції $f(\bar{x})$. Останнє означало б, що напрямок і градієнт функції $\nabla f(\bar{x})$ ортогональні: величина $\left[\bar{p}^{(k)}\right]^T \nabla f\left(\bar{x}^{(k)}\right) = 0$.

Отже, якщо ми захочемо одержати збіжну обчислювальну процедуру, потрібно, щоб вибір λ_k забезпечував суттєве зменшення $f(\bar{x})$ на кожній ітерації та кут між $\bar{p}^{(k)}$ і $\nabla f(\bar{x}^{(k)})$ був менший від прямого. Відповідно до першої вимоги довжину кроку λ_k вибирають, виходячи із умови $\min_{\lambda_k} f(\bar{x}^{(k)} + \lambda_k \bar{p}^{(k)})$. Якщо множина ліній рівня*

---

* Множина ліній рівня – це сукупність усіх точок $\bar{x}$, що задовольняють нерівність $f(\bar{x})$ J b_p,

де β_p – деяке число.

замкнута й обмежена, то узагальнена схема розв'язку задачі оптимізації генерує послідовність точок, для якої $\lim \left| \nabla f(\bar{x}^{(k)}) \right| = 0$

$k \to \infty$

Залежно від способу вибору напрямку пошуку $\bar{p}^{(k)}$, можна одержати різні алгоритми мінімізації функції $f(\bar{x})$.

Метод найшвидшого спуску (метод Коші). Застосування цього методу для розв'язування задач мінімізації розглянуто ще французьким математиком Коші. Оскільки функція $f(\bar{x})$ зменшується в напрямку, протилежному до напрямку градієнта, то $\bar{p}^{(k)} = -\nabla f(\bar{x}^{(k)})$.

Іноді замість градієнта функції $f(\bar{x})$ вибирають нормований напрямок $\bar{p}^{(k)} = -\nabla f(\bar{x}^{(k)}) / \left| \nabla f(\bar{x}^{(k)}) \right|$. Таким чином, у методі найшвидшого спуску обчислювальна процедура утворюється за таким рекурентним співвідношенням:

$$\bar{x}^{(k+1)} = \bar{x}^{(k)} - \lambda_k \nabla f(\bar{x}^{(k)}), \qquad (10.17)$$

або

$$\bar{x}^{(k+1)} = \bar{x}^{(k)} - \lambda_k \nabla f(\bar{x}^{(k)}) / \left| \nabla f(\bar{x}^{(k)}) \right|.$$

Метод найшвидшого спуску лише тоді забезпечує високу швидкість збіжності, коли найбільше і найменше власне значення матриці Гессе значно відрізняються одне від одного.

Приклад 10.5. Мінімізувати функцію $f(\bar{x}) = x_1^2 + 2x_2^2 + x_1x_2 + 4x_1 + 2x_2 + 4$ методом найшвидшого спуску (градієнтним методом), використавши процедуру `MinimizationGradient`. Мінімум функції $f(\bar{x})$ знаходиться у точці $\bar{x}^T = (-2;0)$. Графік функції показаний на рис. 10.12, із якого видно, що функція є унімодальною і має мінімум на множині точок $D = \{\bar{x} : x_1 \in [-5;5], x_2 \in [-5;5]\}$.

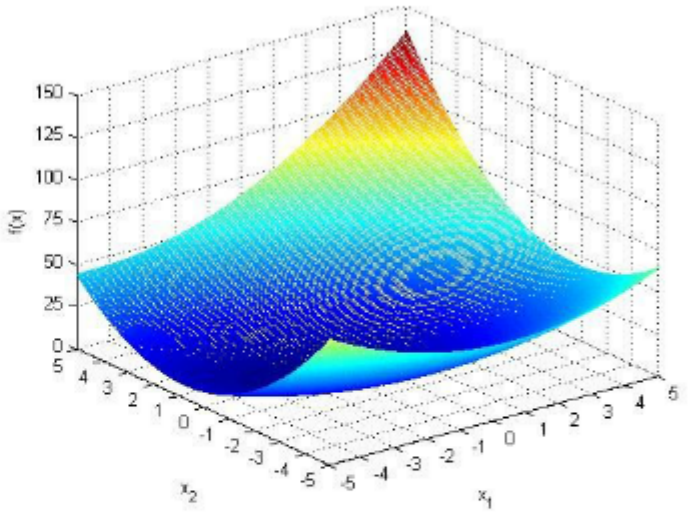


Рисунок 10.12 – Графік функції $f(\bar{x})$.

```
MinimizationGradient'
▷Пошук мінімуму функції багатьох змінних градієнтним
методом
▷  (найшвидшого спуску)
▷Підпроцедури:fun_GradStep-пошук довжини кроку lamda
методом
                золотого січення
▷               fun_gradF-обчислення градієнта функції
▷                  fun_GradNt-обчислення функції f(x) у
точці x
```

```
▷Вхід:max_N-максимальна кількість кроків пошуку
▷      epselon-точність розв'язку задачі
▷      x0-стартова точка
▷Вихід:x-точка мінімуму
▷       f_min-мінімальне значення функції
gradF=fun_gradF(x0)
1 p=-gradF/norm(gradF)
2 x=x0
3 k=1
4 while (abs(norm(p))>epselon)&(k<max_N)
▷Пошук довжини кроку lamda
5       lamda=fun_GradStep(x,p)
▷k-ому рядку матриці V присвоїти значення x
6       V(k,:)=x
7       x=x+lamda*p
8       gradF=fun_gradF(x)
9       p=-gradF/norm(gradF)
10       k=k+1
11 end while
12 return x
```

У процесі розв'язку задачі, як стартова, була взята точка $\bar{x}^T = (0{,}5; -4)$. На рис. 10.13 показана траєкторія руху точки $\bar{x}$ до значення $\bar{x}^*$. Як бачимо, ітераційний процес збігається значно швидше, ніж у безградієнтному методі (див. рис. 10.11). Результат розв'язку задачі: $\bar{x}^* = (-2{,}0; 0)$.

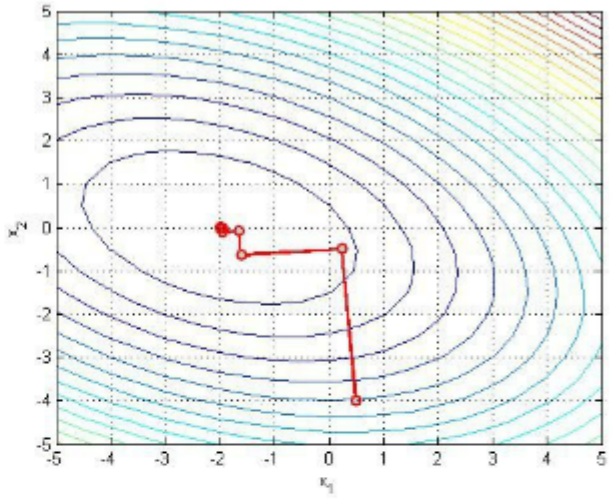


Рисунок 10.14 – Траєкторія руху точки у процесі пошуку мінімуму функції $f(\bar{x})$

Метод Ньютона. За цим методом для визначення напрямку пошуку застосовують другі похідні, які функція $f(\bar{x})$ повинна мати.

Нехай на k –му кроці пошуку незалежні змінні набувають значення $\bar{x}^{(k)}$; при переході до наступної ітерації $\bar{x}$ зміниться на величину $\Delta\bar{x}^{(k)}$. Розкладемо функцію $f(\bar{x}^{(k)} + \mathrm{D}\bar{x}^{(k)})$ у ряд Тейлора, обмежившись лінійними і квадратичними членами розкладу:

$$f(\bar{x}^{(k)} + \mathrm{D}\bar{x}^{(k)}) = f(\bar{x}^{(k)}) + \mathrm{C}^T f(\bar{x}^{(k)})\mathrm{D}\bar{x}^{(k)} + \frac{1}{2}(\mathrm{D}\bar{x}^{(k)})^T \mathrm{C}^2 f(\bar{x}^{(k)})\mathrm{D}\bar{x}^{(k)}. \qquad (10.18)$$

Визначимо мінімум функції $f(\bar{x})$ відносно приросту $\Delta(\bar{x})$. Для цього знаходимо градієнт функції (10.18) $\nabla f\left(\bar{x}^{(k)} + \Delta\bar{x}^{(k)}\right) = \nabla f(\bar{x}^{(k)}) + \nabla^2 f(\bar{x}^{(k)})\Delta\bar{x}^{(k)}$ і, прирівнюючи його до нульового вектора, одержимо:

$$f(\bar{x}^{(k)}) + C^2 f(\bar{x}^{(k)})D\bar{x}^{(k)} = \bar{0} \qquad (10.19)$$

Якщо матриця Гессе не вироджена, то розв'язком (10.19) буде

$$\Delta\bar{x}^{(k)} = -[\nabla^2 f(\bar{x}^{(k)})]^{-1}\nabla f(\bar{x}^{(k)}).$$

Звідси випливає рекурентне правило, яке лежить в основі методу Ньютона:

$$\bar{x}^{(k+1)} = \bar{x}^{(k)} - \lambda_k[\nabla^2 f(\bar{x}^{(k)})]^{-1}\nabla f(\bar{x}^{(k)}). \qquad (10.20)$$

Метод Ньютона має квадратичну швидкість збіжності, а метод Коші тільки лінійну.

Приклад 10.6. За допомогою метода Ньютона розв'язати задачу
min: $f(\bar{x})$, де $f(\bar{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$ – функція Розенброка.

Розв'язок поставленої задачі здійснимо числовим методом. Для цього скористаємося процедурою MethodNewton. В ітераційній процедурі (10.20) λ будемо вибирати із умови

$$\min: f(\bar{x}_k + \lambda_k \bar{p}_k),$$

де $p_k = -[\nabla^2 f(\bar{x}_k)]^{-1}\nabla f(\bar{x}_k)$ - напрям пошуку на кожному кроці.

Обчислимо градієнт функції $f(\bar{x})$:

$$\nabla f(\bar{x}) = \begin{bmatrix} \dfrac{\partial f(\bar{x})}{\partial x_1} \\ \dfrac{\partial f(\bar{x})}{\partial x_2} \end{bmatrix},$$

де $\dfrac{\partial f(\bar{x})}{\partial x_1} = -400(x_2 - x_1^2)x_1 - 2(1 - x_1)$; $\dfrac{\partial f(\bar{x})}{\partial x_2} = 200(x_2 - x_1^2)$
і матрицю Гессе:

$$\nabla^2 f(\bar{x}) = \begin{bmatrix} \dfrac{\partial^2 f(\bar{x})}{\partial x_1^2} & \dfrac{\partial^2 f(\bar{x})}{\partial x_1 \partial x_2} \\ \dfrac{\partial^2 f(\bar{x})}{\partial x_2 \partial x_1} & \dfrac{\partial^2 f(\bar{x})}{\partial x_2^2} \end{bmatrix}.$$

Оскільки $\dfrac{\partial^2 f(\bar{x})}{\partial x_i \partial x_j} = \dfrac{\partial^2 f(\bar{x})}{\partial x_j \partial x_i}$, то матриця Гессе є симетричною матрицею.
Знаходимо

$$\frac{\partial^2 f(\bar{x})}{\partial x_1^2} = -400(x_2 - 3x_1^2) + 2,$$

$$\frac{\partial^2 f(\bar{x})}{\partial x_1 \partial x_2} = -400x_1,$$

$$\frac{\partial^2 f(\bar{x})}{\partial x_2^2} = 200.$$

```
MethodNewton*
▷Пошук мінімуму функції багатьох змінних методом
Ньютона
▷Підпроцедури:fun_gradFNewton-обчислення градієнта
функції
▷             fun_MetodNt-обчислення функції f(x) у
точці x
▷             fun_LengthStep-обчислення довжини кроку
пошуку
▷                              lamda
▷Вхід:max_N-максимальна кількість кроків пошуку
▷      epselon-точність розв'язку задачі
▷      x0-стартова точка
▷Вихід:x-точка мінімуму
▷       f_min-мінімальне значення функції
1 gradF=fun_gradFNewton(x0)
2 Hes=fun_Hess(x0)
3 p=-Hes^(-1)*gradF'
4 x=x0
5 k=1
6 nr=sqrt(gradF(1)^2+gradF(2)^2)
7 while (nr>epselon)&(k<max_N)
8       lamda=fun_LengthStep(x,p')
▷У рядку 10 k -тий рядок матриці V замінений на
вектор x
9       V(k,:)=x
10      x=x+lamda*p'
11      Hes=fun_Hess(x)
12      gradF=fun_gradFNewton(x)
13      p=-Hes^(-1)*gradF'
14      nr=sqrt(gradF(1)^2+gradF(2)^2)
15      k=k+1
16 end while
17 f_min=fun_MetodNt(x)
18 return x, f_min
```

Рис. 10.16 ілюструє процес пошуку мінімуму функції $f(\bar{x})$.

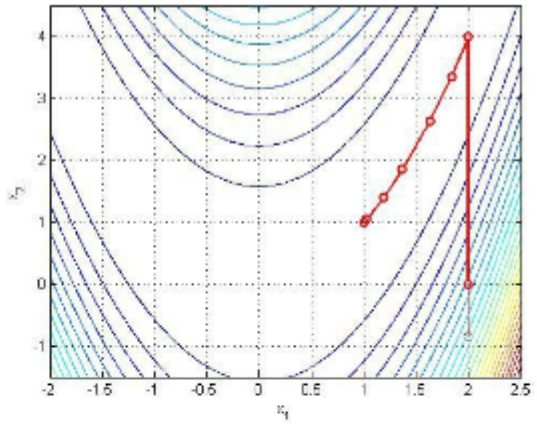


Рисунок 10.16 – Зміна положення точки $\bar{x}$ у процесі мінімізації функції $f(\bar{x})$ (метод Ньютона)

Результат розв'язку задачі – $\bar{x}^* = (1;1)^T$.

Вищу збіжність методу Ньютона обтяжує необхідність знаходження матриці Гессе. Щоб поєднати перевагу методу Коші (похідні лише першого порядку) з перевагою методу Ньютона (висока швидкість збіжності), розроблені методи, які носять назву *квазіньютонівських.*

Квазіньютонівські методи. Вони базуються на апроксимації матриці Гессе $\left[C^2 f\left(\bar{x}^{(k)}\right)\right]^{-1}$ так, що при цьому використовуються лише перші похідні. Ітерація методу Ньютона (10.20) змінюється на

$$\bar{x}^{(k+1)} = \bar{x}^{(k)} - \lambda_k B_k C f\left(\bar{x}^{(k)}\right), \qquad (10.21)$$

де B_k – симетрична матриця розміром n x n. При $B_k = I$ (I – одинична матриця) – маємо градієнтний метод, а при $B_k = \left[C^2 f\left(\bar{x}^{(k)}\right) \right]^{-1}$ – метод Ньютона.

В інших випадках B_k вибирається так, щоб напрямок $B_k C f\left(\bar{x}^{(k)}\right)$ був напрямком спуску при $C f\left(\bar{x}^{(k)}\right) \neq 0$. Ця умова виконується, якщо матриця B_k додатно визначена.

Прийнято, що найефективнішою процедурою апроксимації матриці Гессе є процедура під назвою Брейдена – Флетчера – Гольтфарба – Шанно (BFGS - формула):

$$B_{k+1} = B_k - \frac{\bar{s}^k \left(\bar{s}^k\right)^T B_k \bar{s}^{(k)} \left(\bar{s}^{(k)}\right)^T B_k}{\left(\bar{s}^{(k)}\right)^T B_k \bar{s}^{(k)}} + \frac{\bar{y}^{(k)} \left(\bar{y}^{(k)}\right)^T}{\left(\bar{s}^{(k)}\right)^T \bar{y}^{(k)}} \left(\bar{y}^{(k)}\right)^T, \qquad (10.22)$$

де $\bar{s}^{(k)} = \bar{x}^{(k+1)} - \bar{x}^{(k)}$ - зміна вектора-змінної $\bar{x}^k$ на k-тій ітерації:

$\bar{y}^{(k)} = \nabla f(\bar{x}^{(k+1)}) - \nabla f(\bar{x}^{(k)})$ - відповідна зміна градієнта. Початкове наближення B_k, як правило, дорівнює одиничній матриці. При такому виборі B_k перша ітерація квазіньютонівського пошуку еквівалента методу найшвидшого спуску.

Приклад 10.7. Знайти мінімум функції Розенброка (див. приклад 10.6), використавши квазіньтоновський метод.

Для розв'язку поставленої задачі скористаємося процедурою `KvasiNewton`.

```
KvasiNewton`
▷Пошук мінімуму функції багатьох змінних
квазіньютоновським
  методом
▷Підпроцедури:fun_gradFNewton-обчислення градієнта
функції
▷            fun_MetodNt-обчислення функції f(x) у
точці x
▷            fun_KvasiNewtonStep-пошук довжини кроку
▷Вхід:max_N-максимальна кількість кроків пошуку
▷      epselon-точність розв'язку задачі
▷      x0-стартова точка
▷Вихід:x-точка мінімуму
▷       f_min-мінімальне значення функції
1 gradF0=fun_gradFNewton(x0);
2 n=length(gradF0);
3 ▷Створити одиничну матрицю B розміром nxn;
4 p=-B*gradF0';
5 x=x0;
6 k=1;
▷У рядках 7 та 29 k -тий рядок матриці V замінений на
вектор x
7 V(k,:)=x;
8 nr=sqrt(gradF0(1)^2+gradF0(2)^2);
9 while (nr>epselon)&(k<max_N)
10     if k=1
11        then lamda=fun_KvasiNewtonStep(x,p');
12        x=x+lamda*p';
13        gradF=fun_gradFNewton(x);
14     end if
15       s=x-x0;
16       if s=0
17          then break
18       end if
19       y=gradF-gradF0;
20       B=B-(s*B*s')^(-1)*B*s'*s*B+(y'*y)/(y*s');
21       p=-B^(-1)*gradF';
22       lamda=fun_KvasiNewtonStep(x,p');
23       gradF0=gradF;
24       x0=x;
25       x=x+lamda*p';
26       gradF=fun_gradFNewton(x);
27       nr=sqrt(gradF(1)^2+gradF(2)^2);
28       k=k+1;
29       V(k,:)=x;
30 end while
31 f_min=fun_MetodNt(x %Результати розв'язку задачі
```

```
32 return x, f_min
```

Візуалізація розв'язку задачі min : $f(\bar{x})$ наведене на рис. 10.17.

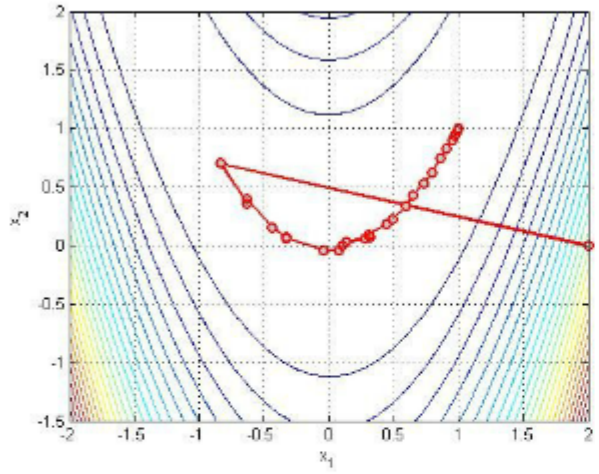


Рисунок 10.17 – Візуалізація пошуку мінімуму функції Розенброка (квазіньютоновський метод)

Результат розв'язку задачі : $\bar{x}^* = (1;1)^T$ її вибір довжини кроку повинен грунтуватись на критерії - min : $f(\bar{x}^{(k)} + 1\,\bar{p}^{(k)})$.

Для того щоб BFGS-формула зберігала додатну визначеність, необхідно і достатньо виконання нерівності:

$$\bar{y}_k^T \bar{s}_k > 0 .$$

Метод спряжених градієнтів. В основі методу лежать властивості квадратичних функцій, і при побудові алгоритму обчислень теж використовують градієнти цільових функцій. Квадратична цільова функція n-незалежних.

$$f(\bar{x}) = \bar{a}^T \bar{x} + \frac{1}{2} \bar{x}^T D \bar{x} , \qquad (10.23)$$

де D – симетрична додатньо-визначена матриця; $\bar{a}$ - вектор, компоненти якого постійні величини, може бути мінімізована за n кроків (або менше), якщо кроки вибрані в так званих спряжених

Порівняльний аналіз рис.10.16 і 10.17 показує, що метод Ньютона має вищу збіжність ніж квазіньютоновський метод.

Глобальна збіжність квазіньютонівського методу до точки мінімуму $\bar{x}^*$ має місце при виконанні таких умов:

і) матриця B_k додатньо визначена і її числа обумовленості* обмежені у кожній із k-ітерацій.

---

* Величина $norm(A) \cdot norm(A^{-1})$ носить назву числа обумовленості матриці, де $norm(A)$ і $norm(A^{-1})$ – норми матриць A і A^{-1}. Існує три норми матриці A розміром n x n.

і) $norm(A)_1 = \max_{1 \le j \le n} \sum_{i=1}^{m} | a_{ij} |$ - максимум суми модулів елементів у стовпці.

іі) $norm(A)_2 = (1_{max} | A^T A |)^{\frac{1}{2}}$, де λ_{max} - максимальне власне значення матриці $A^T A$, $|A^T A|$ - визначник матриці $A^T A$.

ііі) $norm(A)_3 = \max_{1 \le i \le m} \sum_{j=1}^{n} | a_{ij} |$ - максимум суми модулів у лінійці.

напрямках. Вектори d_i і d_j називаються спряженими відносно симетричної матриці D, якщо $\overline{d}_i^T D\overline{d}_j = 0$ для всіх $i \neq j$.

У реальних задачах функція $f(\overline{x})$, яка підлягає мінімізації, часто відрізняється від (10.23). Але все ж таки метод спряжених градієнтів застосовують і в загальному випадку, хоч при складних цільових функціях він може погано працювати і алгоритм, застосований на цьому методі не дасть спряжених напрямків пошуку. Нехай початковий напрямок $\overline{p}_0$ збігається з антиградієнтом - $Cf(\overline{x}^{(0)})$ у початковій точці $\overline{x}^{(0)}$ і k кроків спуску за взаємно спряженими напрямками $p_1, p_2, ..., p_{k-1}$ уже виконані. Тоді за основу вектора $\overline{p}$ слід брати:

$$\overline{p}_k = -\nabla f(\overline{x}^{(k)}) + \beta_{k-1}\overline{p}_{k-1}. \qquad (10.24)$$

Величину β_{k-1} вибирають так, щоб напрямки $\overline{p}_k$ і $\overline{p}_{k-1}$ були спряжені:

$$b_{k-1} = \frac{\left(\overline{y}^{(k-1)}\right)^T Cf(\overline{x}^{(k)})}{|Cf(\overline{x}^{(k-1)})|^2}, \qquad (10.25)$$

де $\overline{y}^{(k-1)} = \nabla f(\overline{x}^{(k)}) - \nabla f(\overline{x}^{(k-1)})$.

Отже, відповідно до методу спряжених графіків, перехід з точки $\overline{x}^k$ в точку $\overline{x}^{(k+1)}$ здійснюється за рекурентною процедурою.

$$x^{(k+1)} = \overline{x}^{(k)} + 1_k\overline{p}_k, \qquad (10.26)$$

де $\overline{p}_k$ - вибирають згідно з (10.24).

Приклад 10.8. Знайти мінімум функції $f(\overline{x}) = e^{-(x_1+x_2)} + x_1^2 + x_1x_2 + x_2^2$, використавши метод спряжених градієнтів.

Графік функції $f(\overline{x})$ показано на рис. 10.19.

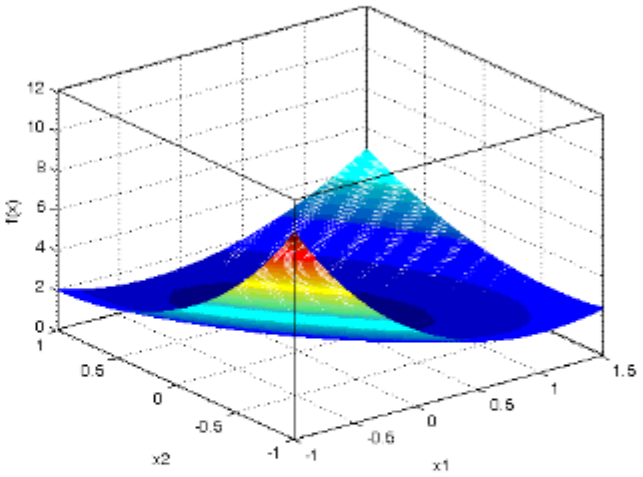


Рисунок 10.19 – Графік функції $f(\overline{x})$, яка підлягає мінімізації.

Для пошуку точки $\overline{x}^*$, яка є мінімумом функції $f(\overline{x})$ скористаємося процедурою MethodConjugatingGradients.

Градієнт функції

$$Cf(\overline{x}) = \begin{pmatrix} -e^{-(x_1+x_2)} + 2x_1 + x_2 \\ -e^{-(x_1+x_2)} + 2x_2 + x_1 \end{pmatrix}$$

запишемо помістимо у підпроцедуру fun_gradFNewton.

```
MethodConjugatingGradients*
▷Пошук мінімуму функції багатьох змінних методом спряжених
  градієнтів
▷Підпрограми:fun_gradFNewton-обчислення градієнта функції
▷             fun_MetodNt-обчислення функції f(x) у точці x
▷             fun_LengthStep-пошук довжини кроку методом
▷                            золотого січення
▷Вхід:max_N-максимальна кількість кроків пошуку
▷     epselon-точність розв'язку задачі
▷     x0-стартова точка
▷Вихід:x-точка мінімуму
▷      f_min-мінімальне значення функції
1 gradF0=fun_gradFNewton(x0);
2 p0=-gradF0;
3 x=x0;
4 k=1;
▷У рядках 5 та 28 k -тий рядок матриці V замінений на вектор x
5 V(k,:)=x;
6 nr=sqrt(gradF0(1)^2+gradF0(2)^2);
7 while (nr>epselon)&(k<max_N)
8    if k=1
9       then lamda=fun_LengthStep(x,p0);
10       x=x+lamda*p0;
11       gradF=fun_gradFNewton(x);
12    end if
13      s=x-x0;
14      if s=0
15         then break
16      end if
17      y=gradF-gradF0;
18      beta=y*gradF'/norm(gradF0);
19      p=-gradF+beta*p0;
20      lamda=fun_LengthStep(x,p);
21      gradF0=gradF;
22      x0=x;
23      p0=p;
24      x=x+lamda*p;
25      gradF=fun_gradFNewton(x);
26      nr=sqrt(gradF(1)^2+gradF(2)^2);
27      k=k+1;
28      V(k,:)=x;
29 end while
30 f_min=fun_MetodNt(x);
30 return f_min, x
```

Як стартова була вибрана точка $\bar{x}^{(0)} = (2;0)^T$. Подальший процес пошуку мінімуму функції $f(\bar{x})$ показаний на рис. 10.20.

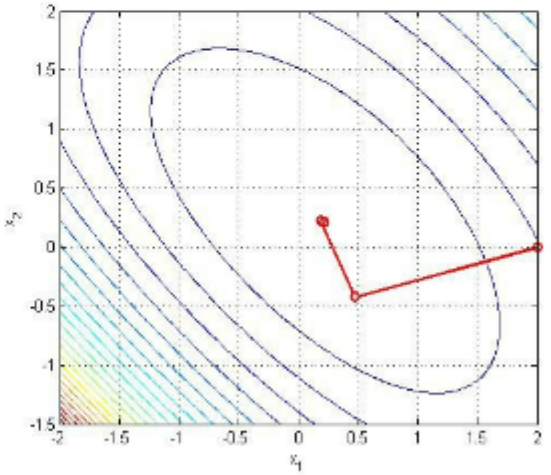


Рисунок 10.20 – Процес пошуку мінімуму функції $f(\bar{x})$ з використанням методу спряжених градієнтів

Вибір довжини кроку здійснювався шляхом розв'язку проміжної задачі мінімізації: $f\left(\overline{x}^k + 1_k \Psi^k\right)$.

Результат розв'язку задачі : $\overline{x}^* = (0,2163; 0,2163)^T$, $f(\overline{x}^*) = 0,7892$.

Вибір методу безумовної мінімізації. Методи, які застосовують для розв'язування задач безумовної мінімізації, можна ранжувати наступним чином:

а) ньютонівські, з урахуванням других похідних;

б) квазіньютонівські з врахуванням градієнтів;

в) методи спряжених градієнтів;

г) методи прямого пошуку.

Чим вища позиція методу в цьому списку, тим більше число задач вказаного класу він розв'язує. Першість належить ньютонівським методам. Використання інформації про другі похідні забезпечує їм високі швидкості збіжності і дає змогу робити якісні висновки відносно знайденого числового розв'язку. Наприклад, за матрицею Гессе можна проаналізувати чутливість знайденого розв'язку задачі поблизу оптимальної точки. Крім того тільки ньютонівські методи запобігають попаданню у сідлові точки.

При переході від ньютонівських до квазіньютоновських методів ефективність останніх переважно практично не змінюється.

У поданому списку найневибагливішим щодо необхідної інформації відносно $f(\overline{x})$ є методи прямого пошуку. Вони мають найменшу швидкість збіжності, тому їх слід використовувати тільки тоді, коли немає альтернативи.

Контрольні питання та завдання

1. Сформулюйте задачу безумовної мінімізації функції однієї змінної.

2. Чи існує принципіальна різниця між задачами максимізації і мінімізації?

3. Для прикладу 3.1 знайдіть такі геометричні розміри кругового конуса, щоб витрати матеріалу на його виготовлення були мінімальними за умови, що заданий його об'єм.

4. Використовуючи необхідні і достатні умови існування мінімуму функції однієї змінної, знайти відповідні умови існування максимуму функції однієї змінної.

5. Яку властивість повинна мати функція, яка гарантує існування єдиного мінімуму?

6. Які недоліки і переваги методу Фібоначчі над іншими методами пошуку мінімуму функції однієї змінної?

7. Як визначити число кроків пошуку мінімуму функції однієї змінної для методу Фібоначчі?

8. Як знайти точку поділу відрізка у пропорції золотого січення?

9. Складіть блок-схему алгоритму, який реалізує метод золотого січення.

10. Використавши програми 10.1 і 10.2 знайти мінімум функції $y = e^{-x^2} + x^2 - x$. Порівняти отримані результати.

11. Дайте характеристику симплексному методу пошуку мінімуму функції багатьох змінних.

12. Які переваги методу Нелдера-Міда над симплексним методом?

14. З якою метою деформують початковий симплекс у методі Нелдера-Міда?

15. За допомогою програми 10.3 знайдіть мінімум функції $f\left(\overline{x}\right) = 4\left(x_1 - 5\right)^2 + \left(x_2 - 6\right)^2$. Цю задачу розв'яжіть аналітичним способом і оцініть похибку числового методу.

16. Коли доцільно застосовувати градієнтні методи для пошуку мінімуму функції багатьох змінних?

17. Методом найшвидшого спуску мінімізуйте функцію $f(\bar{x}) = 4(x_1 - 5)^2 + (x_2 - 6)^2$. Ефективність методу порівняйте з методом Нелдера-Міда.

18. Які переваги і недоліки методу Ньютона над іншими градієнтними методами?

19. Знайдіть точку на поверхні $z = x^2 + (y-1)^2$, яка є найближчою до точки з координатами (2; 5; 7). Обґрунтуйте вибір методу розв'язку задачі.

20. Використовуючи метод спряжених градієнтів та метод Ньютона, знайдіть мінімум функції $f(x, y, z) = 2x^2 + 2y^2 + z^2 - 2xy + yz - 7y - 4z$. Порівняйте ефективність цих методів.

21 Задачу із п. 10 розв'яжіть за допомогою квазіньютонівського методу.

22. Як здійснюється вибір довжини кроку в градієнтних методах?

23. Проаранжуйте числові методи пошуку мінімуму функції багатьох змінних.

ПЕРЕЛІК РЕКОМЕНДОВАНИХ ДЖЕРЕЛ

1 Ахо А. Построение и анализ вычислительных алгоритмов / А. Ахо, Дж. Хопкрофт, Дж. Ульман; пер. с англ. А. О. Слисенко под ред. Ю. В. Матиясевича. – М.: Мир, 1979. 536 с.

2 Ахо А. Структура данных и алгоритмы / А. Ахо, Дж. Хопкрофт, Дж. Ульман; пер. с англ. А. А. Минько. – М.: Издательский дом «Вильямс», 2000. – 384 с.

3 Автоматизація процесів переробки нафти і газу: начальний посібник. / Г. Н. Семенцов, М. І. Горбійчук, Л. І. Жуган, С. А. Чеховський. – Львів: Світ, 1992. – 350 с.

4 Ануфриев И. Е. Самоучитель MatLab 5.3/6.x. / И. Е. Ануфриев. – СПб: БХВ-Петербург, 2002. – 736 с.

4 Бокс Дж. Анализ временных рядов. Прогноз и управление. В 2-ух выпусках / Дж. Бокс, Г. А Дженикс: пер. с англ. – М.: Мир, 1974. – Вып. 1, 321 с.; вып 2, 197 с.

5 Вержбицкий В. М. Основы численных методов: учебник / В. М. Вержбицкий– М.: Высшая школа, 2002. – 840 с.

6 Вирт Н. Алгоритмы + структура данных = программы / Н. Вирт; пер. с англ. Л. Ю. Иоффе под ред. Д. Б. Подшивалова. – М.: Мир, 1985. – 406 с.

7 Гилл Ф., Мюррей У., Райт М. Практическая оптимизация. Пер. с англ. – М.: Мир, 1985. 509 с.

8 Горбійчук М. І. Математичне моделювання на ЕОМ технологічних об'єктів: навчальний посібник / М. І. Горбійчук - Івано-Франківськ: Факел, 2001. – 240 с.

9 Горбійчук М. І. Числові методи і моделювання на ЕОМ / М. І. Горбійчук, Є. П. Пістун. - Івано-Франківськ: Факел, 2010. – 408 с.

10 Калиткин Н. Н. Численные методы: учебное пособие / Н. Н. Калиткин. – СПб: БХВ-Петербург, 2011. – 592 с.

11. Кормен Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн; пер с англ. И. В. Красикова, Н. А. Ореховой, В. Н. Романова. – 2-е изд. – М.: Издательский дом «Вильямс», 2005. – 1296 с.

12 Макконнелл Дж. Основы современных алгоритмов / Дж. Макконнелл; пер с англ. под ред. С. К. Лано. – 2-е изд. дополненное. – М.: Издательский дом «Вильямс», 2004. – 366 с.

13 Мэтьюз Джон Г. Численные методы. Использование MatLab / Джон Г. Мэтьюз, Куртис Д. Финк; пер. с англ. – 3-е изд. – М.: Издательский дом "Вильямс", 2001. – 720 с.

14 Самарский А. А. Численные методы: учебное пособие / А. А.Самарский, А. В. Гулин. – М.: Наука, 1989. – 432 с.

15 Соммервилл И. Инженерия программного обеспечения. 6-е изд. Пер с англ. – М.: Издательский дом "Вильямс", 2002. – 624 с.

16 Фаддеев Д. К. Вычислительные методы линейной алгебры / Д. К.Фаддеев, В. Н. Фаддеева. – М.: Наука, 1963. – 655 с.

17 Химмельблау Д. Прикладное нелинейное программирование. Пер с англ. – М.: Мир, 1975. – 534 с.