Одеський національний університет імені І. І. Мечникова
Факультет математики, фізики та інформаційних технологій
Кафедра оптимального керування і економічної кібернетики

Кваліфікаційна робота

на здобуття ступеня вищої освіти «бакалавр»

«**Числові методи розв'язання багатокрокових оптимізаційних задач**»

«**Numerical methods for multistep optimization problems**»

Виконав: здобувач денної форми навчання
спеціальності 113 Прикладна математика
Освітня програма «Прикладна математика»
Радчин Євгеній Михайлович
Керівник: доктор фіз.-мат. наук, доц. Кічмаренко О.Д. ____
Рецензент: канд. фіз.-мат. наук, доц. Стехун. А.О.

Рекомендовано до захисту:
Протокол засідання кафедри
№____від _____2023 р.

Завідувач кафедри

_____ Ольга Кічмаренко

Захищено на засіданні ЕК №_____
протокол №____від_____2023 р.
Оцінка_____ /_____ /_____

Голова ЕК

_____ _____

Одеса – 2023

ЗМІСТ

ВСТУП

Динамічне програмування – це особливий метод оптимізації, найбільш ефективний при розв’язанні деякого класу задач математичного програмування. Цей клас характеризується можливістю природної (а іноді й штучної) розбивки всієї операції на ряд взаємозалежних етапів. Наприклад, планування виробництва і інвестицій - на ряд часових інтервалів (років, кварталів, місяців), послідовність тестових випробувань при контролі апаратури, пошук оптимальної траєкторії руху тощо. Однак етапами можуть бути елементи операції, ніяк не зв'язані з показником часу. Проте, метод розв’язання подібних багатоетапних задач застосовується той самий. В будь-якому випадку, йдеться про процеси, в яких остаточне рішення (план, проект, послідовність керованих впливів, що забезпечує потрібну траєкторію і т. д.) виробляється послідовно (по кроках), причому на кожному кроці доводиться розв’язувати однотипні задачі, що є суттєво простішим, ніж розв’язувати початкову задачу загалом. У цьому і полягає основна ідея методу: звести розв’язання однієї складної задачі до розв'язання множини однотипних, іноді зовсім простих задач, наприклад вибірка чисел з масиву, підсумовування і порівняння результатів.

Мета кваліфікаційної роботи: дослідити методи розв’язання багатокрокових оптимізаційних задач зі станом векторного типу. Застосувати ці методи до розв’язання транспортної задачі Хічкока-Купманса.

Об’єкт дослідження: транспортна задача Хічкока-Купманса.

Предмет дослідження: розв’язки транспортної задачі Хічкока-Купманса.

Методи дослідження: метод динамічного програмування, метод множників Лагранжа, метод послідовних наближень.

В даній кваліфікаційній роботі розглядаються задачі динамічного програмування з багатовимірним станом, описано підходи до зменшення вимірності стану.

1. ПОСТАНОВКА ЗАДАЧІ, ОСНОВНІ ПОНЯТТЯ ТА ОЗНАЧЕННЯ МЕТОДУ ДИНАМІЧНОГО ПРОГРАМУВАННЯ

Метод динамічного програмування був запропонований наприкінці 50-х років американським математиком Р. Беллманом та є одним із найбільш потужних і широко відомих математичних методів сучасної теорії керування.

Ідеологія методу побудована на засадах системного аналізу, а загальне і чітке формулювання методу динамічного програмування зробило можливим його широке застосування до розв'язання різноманітних задач теорії прийняття рішень, економіки, екології та інших галузей знань.

Тому метод динамічного програмування є одним з найважливіших інструментів теорії керованих процесів.

Словосполучення «динамічне програмування» вперше було використане в 1940-х роках Р. Беллманом для опису процесу знаходження розв'язку задачі, де відповідь на одну задачу може бути отримана тільки після розв'язання задачі, що «передує» їй. Слово «програмування» у словосполученні «динамічне програмування», як і у словосполученні «математичне програмування», означає оптимальну послідовність дій отримання розв'язку задачі. Слово «динамічне» вказує на суттєве значення часу та порядку виконання операцій в процесах і методах, що розглядаються.

В основі методу динамічного програмування покладено два основних принципа.

Принцип декомпозиції (інваріантного занурення) передбачає заміну однієї задачі багатокрокової оптимізації на клас однотипних багатокрокових задач, взаємопов'язаних між собою так, що розв'язок однієї задачі може бути отриманий як наслідок розв'язання іншої задачі (ланцюжок задач), при цьому цей клас обов'язково містить хоча б одну «звичайну» оптимізаційну задачу. Розв'язок кожної задачі з цього класу прив'язують до одного етапу (кроку). Розв'язання останньої задачі в побудованому ланцюжку і є розв'язком початкової багатокрокової оптимізаційної задачі.

Другим фундаментальним принципом методу динамічного програмування є принцип оптимальності Беллмана: яким би не був стан системи в результаті якоїсь кількості кроків, на найближчому кроці потрібно вибирати керування так, щоб воно в сукупності з оптимальним керуванням на всіх наступних кроках приводило до оптимального виграшу на всіх кроках, що залишилися, включаючи даний.

Використання цього принципу гарантує, що керування, обране на будь-якому кроці, є не локально кращим, а кращим з точки зору процесу в цілому.

Беллманом чітко були сформульовані й умови, при яких принцип вірний. Основна вимога - процес керування повинен бути без зворотного зв'язку, тобто керування на даному кроці не повинне мати вплив на попередні кроки.

Принцип оптимальності стверджує, що для будь-якого процесу без зворотного зв'язку оптимальне керування є таким, що воно є оптимальним для будь-якого підпроцесу стосовно вихідного стану цього підпроцесу. Тому розв’язок на кожному кроці виявляється найкращим з погляду керування в цілому.

В методі динамічного програмування використовуються такі основні означення:

Стадія (або етап): задачу, яку ми розв'язуємо за допомогою динамічного програмування, можна розбити на декілька стадій або етапів. Кожна стадія представляє собою певну частину задачі або підзадачу. При вирішенні проблеми цим методом ми прогресивно рухаємося від однієї стадії до наступної, розв'язуючи підзадачі на кожному кроці.

Стан: відображає інформацію, яка пов'язана з певною стадією проблеми. Це може бути значенням змінних, положенням у просторі, кількістю ресурсів і т.д. Кожен стан відповідає певному набору умов або характеристик, які описують поточний етап розв'язання проблеми. Кількість можливих станів зазвичай залежить від розміру проблеми.

Стратегія (варіант рішення): Стратегія в динамічному програмуванні визначає вибір дії або варіанту рішення на кожному етапі. Це може бути,

наприклад, вибір певного руху, операції або вибір набору параметрів. Стратегія визначає, яким чином ми обираємо оптимальний варіант рішення на кожному кроці, враховуючи вже обчислені результати попередніх стадій.

Для застосування методу динамічного програмування для розв'язання конкретної оптимізаційної задачі необхідно побудувати математичну модель задачі. Нехай це буде задача нелінійного програмування.

$$\left.\begin{array}{l} \text{Знайти} \quad \max z = \sum_{j=1}^{N} g_j\left(x_j\right) \\ \text{при обмеженні} \\ \quad \sum_{j=1}^{N} a_j x_j \le b, \;\; a_j > 0, \;\; j = 1, 2, \ldots, N \\ \quad x_j \ge 0, \;\; j = 1, 2, \ldots, N, \;\; \text{всі } x_j \text{ - цілі.} \end{array}\right\} \qquad (1.1)$$

Це цілочисельна задача математичного програмування з сепарабельною цільовою функцією та тільки з одним обмеженням. Задачі такого типу досить часто зустрічаються на практиці. Наприклад, задача про забезпечення автономно процюючої техніки запасними частинами. Якщо б на x_j не накладалася умова цілочисельності, а функції $g_j\left(x_j\right)$ були б вогнутими, то для отримання наближеного розв'язку такої задачі можна було б використовувати методи, в яких нелінійні функції $g_j\left(x_j\right)$ апроксимуються кусково-лінійними функціями, в результаті чого можна застосувати симплекс-метод [5]. Якщо до того ж функції $g_j\left(x_j\right)$ всюди диференційовані, то розв'язок можна знайти за допомогою метода множників Лагранжа [4]. Але, якщо z має багато локальних максимумів, тоді згадані методи дають лише локальний максимум.

Якщо ж вимагається умова цілочисельності невідомих x_j, то жоден з класичних чи числових оптимізаційних методів не може бути застосований.

Далі будемо вважати, що $a_j\ (j=1,2,...,N)$ і b – цілі (цілозначність коефіцієнтів завжди можна забезпечити, обравши більш дрібні одиниці вимірювання).

На основі математичної моделі оптимізаційної задачі будуємо модель динамічного програмування, використовуючи фундаментальні принципи методу.

Елементи моделі динамічного програмування:

1. визначаємо **Етап** $j\ (j=1,2,...,N)$, який співвідноситься з номером функції $g_j(x_j)$. На кожному етапі буде розв'язана одна задача із класу оптимізаційних задач, сформованого за принципом декомпозиції.

2. **стан** співвідноситься з основним обмеженням задачі (1.1). Стан y_j на етапі j – це виконання основного обмеження на етапах *1*, *2*, …, *j*. При цьому $y_j = 0,1,...,$ при $j=1,2,...,N,\ y_N = b$.

3. **Варіанти розв'язків** x_j на етапі j – це допустимі значення розв'язку задачі на етапі j, або змінні керування. Вони ж є невідомими величинами в математичній моделі (1.1).

4. **Функціональні рівняння** (рівняння Беллмана) будуються на функціях Беллмана $f_j(y_j)$, які визначаються із цільової функції задачі (1.1) та є цільовими функціями оптимізаційних задач кожного етапу і залежать виключно від стану y_j на етапі j.

Функціональні рівняння рекурсивно пов'язують функції Беллмана двох сусідніх етапів та реалізують взаємозв'язок між задачами з класу оптимізаційних задач, сформованих за принципом декомпозицій.

На кожному етапі оптимальний розв'язок задачі $x_j^* = x_j^*(y_j)$ залежать від стану системи на цьому етапі, та цей стан може приймати різні значення (діапазонні або дискретні). Тому, знаючи кінцевий стан $y_N = b$, відбувається

поетапне визначення оптимального розв'язку $x_j^*(y_j)$ із перерахунком значення стану y_{j-1} етапу $j-1$ через значення стану y_j. Знайдене значення $x_1^*(y_1)$ – це розв'язок задачі (1.1).

Отже, обчислення проводяться в послідовності $f_1(y_1) \to f_2(y_2) \to \ldots \to f_N(y_N)$, а після цього за допомогою перерахунку станів $y_N \to \ldots \to y_2 \to y_1$ будуть отримані оптимальні розв'язки $x_N, \ldots, x_2, x_1$ на кожному етапі.

Така схема обчислень має назву **прямої прогонки методу динамічного програмування**.

Якщо **стан** y_j на етапі j – це виконання основного обмеження на етапах $j,\ j+1,\ \ldots,\ N-1,\ N$ та при цьому $y_j = 0,1,\ldots,$ при $j = 1,2,\ldots,N$ і $y_1 = b$, тоді обчислення проводяться в послідовності $f_N(y_N) \to \ldots \to f_2(y_2) \to f_1(y_1)$, $y_1 \to y_2 \to \ldots \to y_N$ та $x_1, x_2, \ldots, x_N$, то така схема обчислень має назву **оберненої прогонки методу динамічного програмування**.

Зауважимо, що модель динамічного програмування для кожної задачі нелінійного математичного програмування будується персоналізовано.

Якщо в задачі (1.1) буде 2 основних обмеження, то стан в моделі ДП стає двовимірним. В цьому випадку модель динамічного програмування будується за тією ж схемою, але при розв'язанні задачі збільшується вимірність матриць, в яких реалізується прямий та обернений хід методу динамічного програмування.

2. ЗАДАЧІ РОЗПОДІЛУ ЗА НАЯВНОСТІ ДВОХ ТИПІВ РЕСУРСІВ

2.1 Процес розподілу двох типів ресурсів

Процес поділу ресурсів на частини двох різних типів, які призначені для використання в незалежних технологічних процесах, є прямим узагальненням простого процесу розподілу.

Постановка задачі

Нехай два види ресурсів є відповідно у кількостях *x* та *y*, а x_i та y_i – кількості цих ресурсів, що відокремили для *i*–го процесу. Нехай задана функція корисності

$g_i(x_i, y_i)$ - дохід від *i*–го процесу, якщо на нього відпущено x_i ресурсів 1-го типу та y_i – 2-го типу. (2.1)

Наше математичне завдання полягає у визначенні найбільш ефективного способу використання ресурсів, що можна досягнути шляхом максимізації функції з $2N$ змінними, покладеній в основу задачі

$$R(x_1, x_2, ..., x_N; y_1, y_2, ..., y_N) = \sum_{i=1}^{N} g_i(x_i, y_i) \tag{2.2}$$

при обмеженнях

$$\left.\begin{aligned} (a) \quad & \sum_{i=1}^{N} x_i = x, \quad x_i > 0, \\ (b) \quad & \sum_{i=1}^{M} y_i = y, \quad y_i > 0. \end{aligned}\right\} \tag{2.3}$$

У випадку наявності двох різних видів ресурсів, ми можемо говорити про двовимірний процес розподілу, незалежно від величини *N*. Розв'язання цієї задачі виводиться на чисельних послідовностях функцій з двох змінних, як буде роз'яснено далі.

Рекурентні співвідношення

Введемо послідовність функцій Беллмана $\{f_N(x, y)\}$:

$$f_N(x,y) = \max_{\{x,y\}} R_N(x_1, x_2, ..., x_N;\ y_1, y_2, ..., y_N) \tag{2.4}$$

де максимум береться по області значень x_i , y_i , що визначається умовами (2.3). Тут N приймає значення 1, 2, …; x та y – будь-які невід'ємні числа.

Для $N = 1$ маємо рівність

$$f_1(x,y) = g_1(x,y), \tag{2.5}$$

а для $N \geq 2$ – рекурентне співвідношення

$$f_N(x,y) = \max_{0 \leq x_N \leq x} \max_{0 \leq y_N \leq y} \left[g_N(x_N, y_N) + f_{N\text{-}1}(x - x_N, y - y_N) \right], \tag{2.6}$$

яке є прямим наслідком принципу оптимальності.

2.2 Процес розподілу одного ресурсу з двома типами обмежень

Часто виникає ситуація, коли потрібно розподілити один тип ресурсів між двома видами обмежень. Наприклад, при вирішенні задачі щодо завантаження судна обмеженнями можуть бути вага та обсяг. Цей вид задач може бути розглянутий як окремий випадок задачі, де зв'язок між змінними x_i та y_i представлений як $y_i = h_i(x_i)$.

Постановка задачі

Аналітична задача в цьому випадку полягає у знаходженні максимуму функції

$$R(x_1, x_2, ..., x_N) = g_1(x_1) + g_2(x_2) + ... + g_N(x_N) \tag{2.7}$$

при обмеженнях

$$\left.\begin{array}{ll} (a) & x_i > 0, \\ (b) & \sum_{i=1}^{N} a_i(x_i) \le x, \\ (c) & \sum_{i=1}^{N} b_i(x_i) \le y. \end{array}\right\} \tag{2.8}$$

Якщо припустити, що $a_i(x_i)$ та $b_i(x_i)$ є монотонно зростаючими функціями x_i, які прямують до ∞ при $x_i \to \infty$, можливо провести заміну змінних, яка перетворює $a_i(x_i)$ у x_i при умові, що x_i змінюється безперервно. Однак, якщо x_i має дискретний набір значень (наприклад, $0, 1, 2, \ldots$), така заміна буде неможливою.

Важливо зазначити, що нерівності було використано для заміни обмежень, які були виражені рівностями (2.3). Ця несуттєва зміна дає можливість уникнути деяких математичних тонкощів, що є корисним під час процесу та числового розв'язання.

Рекурентні співвідношення

Дотримуючись підходу, наміченого в **2.1**, покладемо:

$$f_N(x,y) = \max_{\{x\}} \left[g_1(x_1) + g_2(x_2) + \ldots + g_N(x_N) \right], \tag{2.9}$$

де максимум береться по усім x_i, які задовольняють умовам (2.8). Таким чином маємо рівність

$$f_1(x,y) = \max_{\substack{a_1(x_1) \le x \\ b_1(x_1) \le y}} g_1(x_1) \tag{2.10}$$

та загальне рекурентне співвідношення вигляду

$$f_N(x,y) = \max_{\substack{a_N(x_N) \le x \\ b_N(x_N) \le y}} \left[g_N(x_N) + f_{N-1}\left(x - a_N(x_N),\ y - b_N(x_N)\right) \right]. \tag{2.11}$$

Аналіз обчислювальних аспектів

Отримані рекурентні співвідношення можуть бути використані для знаходження чисельного розв'язку на підставі методу, подібного до одновимірного процесу розподілу. Хоча *якість* методу абсолютно не змінюється, ми маємо суттєві *кількісні* зміни в обчислювальному процесі, коли розглядаємо його з погляду вимог до пам'яті - зберігання даних та часу обчислень. Цей приклад показує класичне правило, що велика кількісна різниця може породити значну якісну відмінність. Побудуємо сітку для (2.11). Для того, щоб визначити функцію $f_N(x,y)$, наприклад, в області $0 \le x \le M$, $0 \le y \le M$, домовимося обчислювати її значення тільки у вузлах сітки, наприклад, при $x = k\Delta$ та $y = l\Delta$, $k,l = 0,1,...,M$.

Якщо при одновимірному процесі розподілу для задавання функції $f_{N-1}(x)$ потрібно зберігати $M+1$ значень, то для двох типів ресурсів задля запам’ятовування інформації про функцію $f_{N-1}(x,y)$ треба зберігати вже $(M+1)^2$ значень. Так як $0 \le x \le 1$, $0 \le y \le 1$ – основна область, а для x та y обрано крок у $0,01$, то це означає, що замість 10^2 точок потрібно вже 10^4 точок. Та через те, що одночасно зі зберіганням значень $f_{N-1}(x,y)$ ми маємо обчислювати нову функцію $f_N(x,y)$ та політику $x_N = x_N(x,y)$, збільшення цільності точок буде принаймні втричі більшим.

3. ТРАНСПОРТНА ЗАДАЧА ХІЧКОКА-КУПМАНСА

Задача оптимізації мережі є одним із ключових питань математичної економіки. У контексті цієї задачі важливим аспектом є розподіл ресурсів між вузлами мережі. Одним із найбільш популярних методів оптимізації мережі є транспортна модель Хічкока-Купманса, яка незалежно вивчалась Канторовичем та Ароншайном. Вона застосовується для знаходження найкращого способу транспортування ресурсів між кожною парою вузлів. Така модель, як розв'язання задачі оптимізації мережі, застосовується у різних галузях сфери діяльності людини: економіці, промисловості, обчислювальних системах, електроніці, зв'язку та організації.

Існують достатньо потужні методи аналізу транспортної задачі Хічкока-Купманса, котрі були розроблені під час проведення багатьох досліджень такими вченими, як Данциг, Флад, Форд та Фалкерсон [6]. Але ми розглянемо метод, заснований на використанні функціональних рівнянь. Цей метод дозволяє враховувати обмеження, які наближають до реальності аналізовані процеси та не можуть бути враховані з допомогою методів, розроблених для лінійних моделей. Крім того, разом з функціональними рівняннями може використовуватися метод множників Лагранжа, що дає змогу отримувати додаткову інформацію про оптимальні розв’язки.

Вивчення транспортної задачі Хічкока-Купманса показує, що вона складніша порівняно з іншими мережними задачами. Отже, поєднання різних методів аналізу дозволяє отримувати надійніші результати. Зокрема, використання методу послідовних наближень та функціональних рівнянь дає змогу вирішувати досить складні задачі у межах даної галузі.

Таким чином, рішення транспортної задачі Хічкока-Купманса є одним з найважливіших аспектів математичної економіки, і для досягнення оптимального рішення потрібно використовувати комбінацію різних методів аналізу.

3.1 Математична модель транспортного процесу

Назвемо пункти, в яких зосереджені ресурси, складами і пункти, в яких існує попит на ці ресурси, пунктами споживання. Ми можемо вважати їх розташованими таким чином:

Склади	Пункти споживання
$i = 1 : D_1$	$j = 1 : P_1$
$2 : D_2$	$2 : P_2$
.	.
.	.
.	.
$M : D_M$	$N : P_N$.

Будемо розглядати тільки той випадок, коли або число складів, або число пунктів споживання мале, хоча ми й формулюватимемо процес у загальних термінах. Якщо одне з цих чисел мале, інше може бути як завгодно великим - це не вплине на можливість розв'язання задачі. Надалі припускатимемо, що є один вид ресурсів, і введемо такі величини:

$$\left.\begin{array}{l} x_i - \text{ запаси цього ресурсу на } i\text{-му складі, } i = 1,\ldots,M, \\ r_j - \text{ попит на цей ресурс у } j\text{-му пункті споживання,} \\ \quad j = 1,\ldots,N. \end{array}\right\} \tag{3.1}$$

Припускаючи, що загальний запас дорівнює загальному попиту, тобто що задача полягає тільки в розподілі, матимемо:

$$\sum_i x_i = \sum_j r_j \tag{3.2}$$

Нехай

$$x_{ij} - \text{ кількість ресурсів, що відправляють з } i\text{-го складу в } j\text{-й пункт споживання,} \tag{3.3}$$

та

$$g_{ij}\left(x_{ij}\right) - \text{ вартість здійснення цієї операції.} \tag{3.4}$$

Величини x_{ij} підпорядковані трьом обмеженням:

a) невід'ємність;

b) обмеження на запаси: загальна кількість ресурсів, відправлена з будь-якого складу, має дорівнювати запасам на цьому складі;

c) обмеження на попит: загальна кількість ресурсів, відправлена в будь-який пункт споживання, має дорівнювати попиту в цьому пункті. (3.5)

Зазначимо, що ми виключаємо транспортування через перевалочні пункти, тобто ми не можемо переміщати ресурси з одного складу до іншого і потім до пункту споживання або зі складу до пункту споживання і потім до іншого пункту споживання.

Отже, маємо такі співвідношення:

$$\left.\begin{array}{ll} (a) & x_{ij} \geq 0, \\ (b) & \sum_{j=1}^{N} x_{ij} = x_i, \qquad i = 1,2,...,M, \\ (c) & \sum_{i=1}^{M} x_{ij} = r_j, \qquad j = 1,2,...,N. \end{array}\right\} \tag{3.6}$$

Задача полягає у визначенні величин x_{ij} , що задовольняють умовам (3.6), так, щоб мінімізувати загальну вартість переміщення ресурсів

$$C_{MN} = \sum_{i,j} g_{ij}\left(x_{ij}\right). \tag{3.7}$$

Якщо не встановити додаткових обмежень на функції $g_{ij}\left(x\right)$, то очевидно, що ця задача становить високу складність. Однак, навіть якщо такі обмеження буде встановлено, задача залишиться складною, особливо якщо значення M і N будуть великими. Існує дуже цікавий випадок, коли вартість доставки товарів з будь-якого складу в будь-який пункт споживання залежить пропорційно від обсягу товарів, що перевозяться, тобто

$$g_{ij} = d_{ij} x_{ij} \tag{3.8}$$

Коефіцієнт d_{ij} можна інтерпретувати як *відстань*.

Коли функції вартості є лінійними, задача мінімізації стає класичною транспортною задачею.

Застосуємо метод динамічного програмування

Почнемо з розглядання випадку, коли є два склади та довільна кількість пунктів споживання.

Склади	Пункти споживання
$D_1 : x_1$	$P_1 : r_1$
	$P_2 : r_2$
$D_2 : x_2$	.
	.
	.
	$P_{N-1} : r_{N-1}$
	$P_N : r_N$

Попит можна задовольняти по черзі, починаючи з пункту P_N, потім переходячи до P_{N-1}, і т. д. Так статичний процес перетворюється на динамічний.

Введемо функції $f_N(x_1, x_2)$, що визначені для $N = 1,\ 2,\ ...,\ x_1,\ x_2 \geq 0$ наступним чином:

$f_N(x_1, x_2)$ - величина витрат при використанні оптимальної політики, коли починають відповідно з кількостей $x_1,\ x_2$ на складах D_1 та D_2, при фіксованих потребах $r_1,\ r_2,\ ...,\ r_N$ відповідно в пунктах споживання $P_1,\ P_2,\ ...,\ P_N$. (3.9)

Задовольняючи першим попит у N-му пункті споживання, ми понесемо витрати

$$g_{1N}(x_{1N}) + g_{2N}(x_{2N}) \tag{3.10}$$

та зменшимо запаси ресурсів на складах D_1 та D_2 до $x_1 - x_{1N}$ та $x_2 - x_{2N}$. Використовуючи принцип оптимальності, отримаємо для $N \geq 2$ рекурентне співвідношення

$$f_N(x_1, x_2) = \min_{\{R_N\}} \left[g_{1N}(x_{1N}) + g_{2N}(x_{2N}) + f_{N-1}(x_1 - x_{1N}, x_2 - x_{2N}) \right], \qquad (3.11)$$

де R_N - двовимірна область, що визначається умовами

$$\left.\begin{array}{ll} (a) & x_{1N} + x_{2N} = r_N, \\ (b) & 0 \leq x_{1N} \leq x_1, \\ (c) & 0 \leq x_{2N} \leq x_2. \end{array}\right\} \qquad (3.12)$$

Для $N = 1$ маємо:

$$f_1(x_1, x_2) = g_{11}(x_1) + g_{21}(x_2) \qquad (3.13)$$

Очевидно, що цей метод для розв'язання задачі з двома складами, можна використати для розв'язання аналогічної задачі з більш ніж двома складами. Однак, якщо задача вимагає чисельного розв'язання, то при значенні $M \geq 3$, ми стикаємося зі труднощами, що пов'язані з великою кількістю варіантів і розмірністю задачі.

3.2 Зниження розмірності

До сіх ми не використовували додаткову інформацію про те, що запаси дорівнюють попиту, тобто що

$$x_1 + x_2 = \sum_{i=1}^{N} r_i. \qquad (3.14)$$

Звідси випливає, що при фіксованому попиті величину x_2 можна визначити щоразу, коли відома x_1. Отже, можна виключати параметр стану x_2:

$$f_N(x_1, x_2) \equiv f_N(x_1). \qquad (3.15)$$

Співвідношення (3.11) перейде у

$$f_N(x_1) = \min_{x_{1N}} \left[g_{1N}(x_{1N}) + g_{2N}(r_N - x_{1N}) + f_{N-1}(x_1 - x_{1N}) \right], \tag{3.16}$$

де x_{1N} задовольняє умовам

$$\left.\begin{array}{ll} (a) & 0 \le x_1 - x_{1N}, \\ (b) & 0 \le r_N - x_{1N} \le \sum_{i=1}^{N} r_i - x_1. \end{array}\right\} \tag{3.17}$$

Областю зміни змінної x_1 у $f_N(x_1)$ слугує $\left[0,\ \sum_{i=1}^{N} r_i\right]$.

Таким чином, задача, в якій М складів, може бути зведена до послідовності функцій М-1 змінних, використовуючи той же самий підхід. Отже, задачі з двома складами є досить простими, задачі з трьома складами є доступними для прямого розв’язання, тоді як задачі з чотирма складами і більше вимагають використання спеціальних методів.

3.3 Збільшення кроку сітки

Одним із способів обійти труднощі з розмірністю є збільшення кроку сітки. У випадку з задачами, що мають два параметри стану, ми можемо дозволити кожній змінній набувати 100 різних значень, адже для цього достатньо лише 10^4 клітинок. Для розв’язання методом динамічного програмування потребується певна кількість пам'яті. Це пов'язано з тим, що ми займаємося одночасно декількома функціями, а саме: «старою» функцією доходу $f_{N-1}(p)$, «новою» функцією доходу $f_N(p)$ та новою функцією політики $x_N(p)$.

Якщо ми зменшимо кількість різних значень для кожної змінної до 20, а кількість параметрів стану збільшимо до трьох то нам знадобиться 8×10^3 значень. Тобто, можна використовувати функції багатьох змінних, "розріджуючи" область зміни кожної змінної. Але ця стратегія має певний недолік – меншу точність.

3.4 Застосування множника Лагранжа до задачі з трьома складами

Якщо є три склади, то з обчислювальної точки зору задачу можна розглядати в термінах послідовностей функцій двох змінних. Але якщо ввести множник Лагранжа, задача зведеться до обчислення послідовності функцій однієї змінної.

Розглянемо задачу з трьома складами, у якій є кількість x_1 на першому складі та необмежені кількості на другому і третьому складах:

$$\begin{array}{ll} D_1 : x_1 & P_1 : r_1 \\ D_2 : \infty & P_2 : r_2 \\ D_2 : \infty & . \\ & . \\ & . \\ & P_{N-1} : r_{N-1} \\ & P_N : r_N \end{array}$$

Припустимо, що вартість перевезення така сама, як і вище, і що за кожну одиницю, перевезену з D_2, ми платимо додатково λ, і за кожну одиницю, перевезену з D_3, додатково платимо 1.

У цьому випадку загальні витрати задаються виразом

$$\sum_{i=1}^{3}\sum_{j=1}^{N} g_{ij}\left(x_{ij}\right) + \lambda\sum_{j=1}^{N} x_{2j} + \sum_{j=1}^{N} x_{3j}. \qquad (3.18)$$

По-перше на думку приходить те, що слід було б використовувати два множники Лагранжа. Однак у цьому немає необхідності, що видно з наступних міркувань. Припустімо, що λ змінюється доти, доки для значень x_{ij}, що мінімізують, ми не матимемо рівності $\sum_{j=1}^{N} x_{2j} = x_2$. Тоді автоматично отримаємо:

$$\sum_{j=1}^{N} x_{3j} = x_1 + x_2 - \sum_{j=1}^{N} r_j, \tag{3.19}$$

тобто $\sum_{j=1}^{N} x_{3j}$ є фіксованою величиною, так як запаси дорівнюють попиту. Таким чином, можна було б зовсім виключити третій член у (3.18).

Покладемо $f_N(x_1)$ таким, що буде дорівнювати мінімальному значенню функції (3.18), де мінімум береться по області:

$$\left.\begin{aligned} &(a) \quad \sum_{j=1}^{N} x_{1j} = x_1, \\ &(b) \quad x_{1j},\ x_{2j},\ x_{3j} \geq 0. \end{aligned}\right\} \tag{3.20}$$

Щоб отримати рекурентне співвідношення для f_N, як і раніше, буде задовольнятися спочатку попит у N-му пункті споживання.

Ми прийшли до рівняння

$$\begin{aligned} f_N(x_1) = \min_{R_N} \big[& g_{1N}(x_{1N}) + g_{2N}(x_{2N}) + g_{3N}(x_{3N}) + \\ & + \lambda x_{2N} + x_{3N} + f_{N-1}(x_1 - x_{1N}) \big], \end{aligned} \tag{3.21}$$

де R_N - область, що задається умовами

$$\left.\begin{aligned} &(a) \quad x_{1N} + x_{2N} + x_{3N} = r_N, \\ &(b) \quad 0 \leq x_{1N} \leq x_1, \\ &(c) \quad x_{2N},\ x_{3N} \geq 0. \end{aligned}\right\} \tag{3.22}$$

Мінімізація по змінним x_{2N} та x_{3N} може бути виконана заздалегідь у явному вигляді або чисельно. Нехай

$$g_N(x_{1N}; \lambda) = \min_{S_N} \big[g_{2N}(x_{2N}) + g_{3N}(x_{3N}) + \lambda x_{2N} + x_{3N} \big], \tag{3.23}$$

де S_N - нова область, що задається умовами

$$\left.\begin{array}{ll}(a) & x_{2N}+x_{3N}=r_N-x_{1N},\\ (b) & x_{2N},\ x_{3N}\geq 0.\end{array}\right\} \tag{3.24}$$

Тоді рекурентна формула (3.21) приймає вигляд

$$f_N(x_1)=\min_{0\leq x_{1N}\leq r_N}\left[g_{1N}(x_{1N};\lambda)+g_N(x_{1N};\lambda)+f_{N-1}(x_1-x_{1N})\right]. \tag{3.25}$$

Параметр λ тепер змінюється між $-\infty$ та $+\infty$, доки загальна кількість, що перевозиться з D_2, не буде дорівнювати x_2 - початковому рівню ресурсів у D_2.

Кількість, що перевозиться з D_3 тоді автоматично буде дорівнювати $x_3=\sum_{j=1}^{N}r_j-x_1-x_2$.

Приклад I

У першому чисельному прикладі - перевезення з двох складів у десять пунктів споживання - ми вважатимемо, що вартість перевезення являє собою квадратичну функцію від перевезеної кількості плюс "організаційні" витрати. Під "організаційними" витратами ми розуміємо витрати, які не залежать від кількості, що перевозиться, і дорівнюють нулю, коли перевезення не здійснюється. Вартості наведено у таблиці 3.1.

Цією таблицею треба користуватись наступним чином. Кожна функція $g_{ij}(x)$ має вигляд

$$g_{ij}(x)=a_{ij}x+b_{ij}x^2+c_{ij}(x), \tag{3.26}$$

де $c_{ij}(x)$ - те, що звичайно називають «організаційними» витратами, або «постійними витратами»; $c_{ij}(x)$ дорівнює 0, якщо $x=0$, та дорівнює постійній c_{ij}, якщо $x>0$.

Таким чином, вартість перевезень кількості x зі складу 1 до споживача 3 дорівнює $3x+0,01x^2$; зі складу 1 до споживача 2 дорівнює $1+2x$, якщо $x>0$, та 0, якщо $x=0$.

Таблиця 3.1

Вартості перевезень у прикладі I

Споживачі	Зі складу 1			Зі складу 2			Попит
	організаційні витрати (c_{ij})	a_{ij}	b_{ij}	організаційні витрати (c_{ij})	a_{ij}	b_{ij}	
1		1,0		2	3,1		10
2	1	2,0			4,1		25
3		3,0	0,01		2,1		45
4		1,5			1,1	0,1	15
5		2,5			2,6		5
6	10	5,0	-0,01		3,0		15
7		3,0		5	1,0	0,2	20
8		6,0			2,0		15
9	8	6,0	-0,05		2,0		10
10		6,0			5,0	0,01	20

Припустимо, що треба перевезти 100 одиниць зі складу 1 та 80 одиниць зі складу 2. Оптимальний розв’язок наведено у таблиці 3.2.

Код, за допомогою якого було обчислено цей приклад знаходиться в [Додатку А].

Загальний час роботи програми дорівнює в середньому 18,7 мілісекунд за 20 прогонів. Ці розрахунки були проведені на ноутбуці Lenovo IdeaPad L340 Gaming з процесором Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz та і розміром оперативної пам'яті (RAM) 16,0 ГБ.

Таблиця 3.2

Оптимальний розв’язок прикладу I

Споживачі	Зі складу 1	Зі складу 2	Витрати	Накопичені витрати
1	10	0	10,00	10,00
2	25	0	51,00	61,00
3	5	40	99,25	160,25
4	15	0	22,50	182,75
5	5	0	12,50	195,25
6	0	15	45,00	240,25
7	20	0	60,00	300,25
8	0	15	30,00	330,25
9	0	10	20,00	350,25
10	20	0	120,00	470,25

Приклад II

Спробуємо додати до прикладу 1 третій склад. Показники вартості його перевезень наведено у таблиці 3.3.

На складі 3 знаходяться 155 додаткових одиниць та, як показано вище, на таку ж величину збільшився попит. При використанні методу множників Лагранжа вибір $\lambda = 2,0$ дає потрібний результат. Оптимальний розв’язок наведено в таблиці 3.4.

Для розрахунку використовувався код на мові програмування Java, що знаходиться в [Додатку А].

Загальний час роботи програми дорівнює в середньому 20,65 мілісекунд за 20 прогонів. Ці розрахунки були проведені на ноутбуці Lenovo IdeaPad L340 Gaming з процесором Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz та і розміром оперативної пам'яті (RAM) 16,0 ГБ.

Таблиця 3.3

Вартості перевезень стосовно 3-го складу

Споживачі	Організаційні витрати (c_{ij})	Зі складу 3		Попит
		a_{ij}	b_{ij}	
1		7		25
2		3		40
3		9		60
4		1		30
5		1		20
6	5	2		30
7		4		35
8	6	3		30
9		5		25
10		6		40

Таблиця 3.4

Оптимальний розв’язок після додавання 3-го складу

Споживачі	Склад 1	Склад 2	Склад 3	Витрати	Накопичені витрати
1	25	0	0	25,00	25,00
2	40	0	0	81,00	106,00
3	5	55	0	130,75	236,75
4	0	0	30	30,00	266,75
5	0	0	20	20,00	286,75
6	0	0	30	65,00	351,75
7	30	0	5	110,00	461,75
8	0	0	30	96,00	557,75
9	0	25	0	50,00	607,75
10	0	0	40	240,00	847,75

3.5 Послідовні наближення

Для подолання труднощів, пов'язаних з різницями, можна використовувати найбільш потужний і універсальний метод аналізу - метод послідовних наближень.

Загальна схема методу послідовних наближень

Основні етапи методу можна описати наступним чином. Для заданого функціонального рівняння ми вгадуємо рішення. Якщо передбачуваний розв'язок не є істинним, то ми робимо поправку, яку визначає саме функціональне рівняння, сподіваючись отримати в такий спосіб краще наближення до розв'язку. Процес триває доти, доки ми або не отримаємо розв'язок, або не наблизимось до нього з необхідним ступенем точності.

Нехай задане рівняння, розв’язок якого ми шукаємо, має вигляд

$$T(u)=0 \tag{3.27}$$

та нехай рівняння $S(u)=0$ легше розв’язати. Запишемо початкове рівняння у вигляді

$$S(u)=S(u)-T(u), \tag{3.28}$$

та нехай за перше наближення u_0 взято розв’язок рівняння $S(u)=0$. Нехай наступне наближення u_1 визначається з рівняння

$$S(u_1)=S(u_0)-T(u_0), \tag{3.29}$$

і взагалі $(n+1)$-е наближення визначається з n-го наближення за допомогою рівняння

$$S(u_{n+1})=S(u_n)-T(u_n), \tag{3.30}$$

Якщо $S(u)$ обрано відповідним чином та $T(u)$ має відповідні властивості, то послідовність $\{u_n\}$ буде сходитися до розв’язку рівняння $T(u)=0$.

Було проведено багато досліджень з використанням цього методу при вивчені диференціальних рівнянь - більш конкретно, як звичайних, так і в частинних похідних [7].

Наближення в просторі політик

Прототипом рівняння динамічного програмування у загальному вигляді є рівняння

$$f(p) = \max_q \left[g(p,q) + f\left(T(p,q)\right) \right]. \qquad (3.31)$$

Тут p – параметр стану, а q – розв’язок. Класичний підхід до цього рівняння у звичайній ситуації, тобто коли неможна знайти точний аналітичний розв’язок, полягає у вгадуванні початкової функції $f_0(p)$, а потім у визначенні послідовностей функцій за допомогою рекурентного співвідношення

$$f_{n+1}(p) = \max_q \left[g(p,q) + f_n\left(T(p,q)\right) \right], \quad n = 0,\ 1,\ \ldots. \qquad (3.32)$$

В цілому ця послідовність збігається до розв’язку рівняння (3.31), крім того, умови, що забезпечують збіжність, зазвичай гарантують також і єдиність розв’язку.

Зауважимо, що у рівняння (3.31) входять насправді дві невідомі функції: *функція доходу* $f(p)$ та *функція політики* $q(p)$. Вони, звісно, не є незалежними, оскільки одна визначає іншу. За заданої функції доходу $f(p)$ функція політики визначається за допомогою операції максимізації правої частини рівняння (3.31), за заданої функції політики $q(p)$ функція $f(p)$ визначається як розв'язок рівняння

$$f(p) = g(p,q) + f\left(T(p,q)\right), \qquad (3.33)$$

де $q = q(p)$. Це рівняння зазвичай розв'язується прямим ітеруванням. Визнання рівноцінності двох функцій $f(p)$ і $q(p)$ дає нам можливість сильно збільшити сферу дії методу послідовних наближень. На додаток до запропонованого вище

способу наближення за допомогою співвідношення (3.32), ми можемо розглянути новий тип наближення, що становить особливий інтерес для теорії багатокрокових процесів розв'язання, *наближення в просторі політик*.

Замість того, щоб починати з вгадування вигляду функції $f(p)$, ми будемо починати з вгадування вигляду функції $q(p)$.

Відповідна функція доходу $f_0(p)$ визначається як розв'язок рівняння

$$f_0(p) = g(p, q_0) + f_0\left(T(p, q_0)\right), \tag{3.34}$$

де $q_0 = q_0(p)$. Щоб отримати краще наближення, визначаємо $q_1 = q_1(p)$ як функцію, що максимізує вираз

$$g(p, q) + f_0\left(T(p, q)\right), \tag{3.35}$$

а потім визначаємо $f_1(p)$ за допомогою співвідношення

$$f_1(p) = g(p, q_1) + f_1\left(T(p, q_1)\right). \tag{3.36}$$

Продовжуючи таким чином, отримаємо дві послідовності: $\{q_n(p)\}$ та $\{f_n(p)\}$. Послідовність

$$f_0(p) \le f_1(p) \le \ldots \tag{3.37}$$

монотонно збігається.

Загалом, наближення в просторі політик тим чи іншим способом буде монотонним наближенням. Цікавим у цьому понятті наближення є те, що воно становить основу методу *квазілінеаризації*.

3.6 Послідовні наближення – II

Нехай треба максимізувати функцію процесу розподілу при наявності двох типів ресурсів, як у процесі, описаному в [Розділі 2]

$$f_N(x,y)=\sum_{i=1}^{N} g_i(x_i,y_i) \tag{3.38}$$

за умовами

$$\left.\begin{array}{ll}(a) & \sum_{i=1}^{N} x_i = x, \quad x_i \ge 0, \\ (b) & \sum_{i=1}^{N} y_i = y, \quad y_i \ge 0. \end{array}\right\} \tag{3.39}$$

Нехай $x^0=\{x_i^0\}$ - початкове наближення для множини значень x_i. Це є наближення у «просторі політик». Потім визначаємо максимум функції

$$R_N(x,y)=\sum_{i=1}^{N} g_i(x_i^0,y_i) \tag{3.40}$$

по усім y_i, що задовольняють умову (3.39, b), використовуючи звичайне одновимірне рекурентне співвідношення

$$f_N(y)=\max_{0\le y_N\le y}\left[g_N(x_N^0,y_N)+f_{N-1}(y-y_N)\right], \tag{3.41}$$

де $N=2,\ 3,\ \ldots$ та

$$f_1(y)=g_1(x_1^0,y). \tag{3.42}$$

Цей процес дає для кожного значення y множину значень y_i, $y^0=\{y_i^0\}$. Використовуючи ці значення y_i, котрі називаються y_i^0, розглянемо задачу максимізації функції

$$\sum_{i=1}^{N} g_i(x_i,y_i^0) \tag{3.43}$$

по всім значенням x_i, що задовольняють умові (3.39, a). Ця задача вирішується за допомогою рекурентного співвідношення, аналогічно до (3.41).

Таким чином ми отримуємо множину значень x_i, $x^1=\{x_i^1\}$. Повторюючи тепер цей процес, отримаємо пару послідовностей $\{x^n\}$, $\{y^n\}$, $n=1,\ 2,\ \ldots$.

Зрозуміло, що послідовність значень $\left\{R_N\left(x^n, y^n\right)\right\}$ - монотонно зростаюча. Однак ця послідовність зовсім не зобов'язана сходитися до абсолютного максимуму. Щоб переконатися в цьому, розглянемо, наприклад, функцію двох змінних $z = g\left(x, y\right)$, зображувану поверхнею виду, показаного на рис. 3.1.

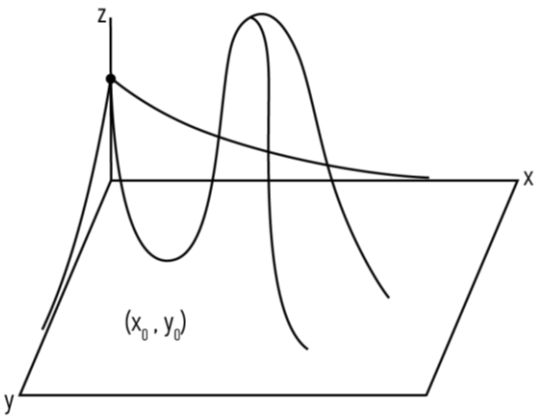

Рис. 3.1

Хоча абсолютний максимум знаходиться у точці $\left(x_0, y_0\right)$, якщо ми візьмемо за початкове наближення $x = 0$, то в описаному вище процесі максимізації спочатку по *x*, потім по *y*, потім знову по *x* і т.д. ми застрягнемо у точці $\left(0,0\right)$ - точці відносного максимуму, як показано у прикладі відносного мінімуму.

З іншого боку, якщо початкове наближення було взяте достатньо близьким до точки $\left(x_0, y_0\right)$, то приведений вище метод сходитиметься до необхідної точки $\left(x_0, y_0\right)$.

У будь-якому разі цей метод може бути використаний для перевірки того, чи дає конкретний вибір значень x_i та y_i *відносний максимум*, і якщо ні, то чи має місце збіжність до найближчого відносного максимуму. Якщо виходити з початкових векторів $\left\{x^0, y^0\right\}$, досить віддалених них один від одного, то ми можемо очікувати, що таким шляхом визначиться ціла низка відносних максимумів та, як можна сподіватися, серед них буде абсолютний максимум.

3.7 Послідовні наближення – III

Розглянемо тепер інше застосування послідовних наближень, в якому використовується неперервність зміни значення, що доставляє максимум.

Розглянемо процес розподілу двох типів ресурсів, що був описаний в [Розділі 2], та відмітимо залежність положення точки максимуму від *x* та *y*, які є відповідно кількостями двох початкових ресурсів. Ця залежність може не бути рівномірно неперервною. Для ілюстрації цього, розглянемо одновимірний випадок. Припустимо, що ми маємо функцію $g(x_1, x_2)$ від x_1 та x_2, максимум якої треба знайти в області $x_1 + x_2 = x$, $x_1, x_2 \geq 0$. Позначимо $f(x_1) = g(x_1, x - x_1)$ та розглянемо графік цієї функції (рис. 3.2) при $0 \leq x_1 \leq x$.

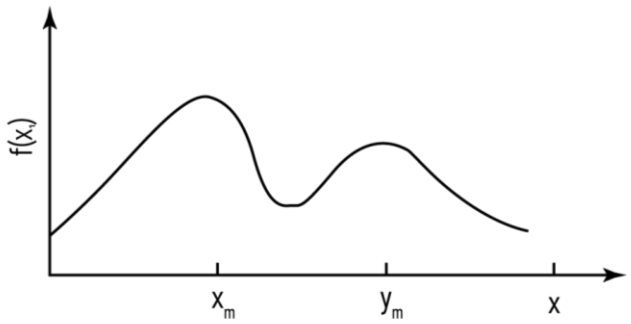

Рис. 3.2

В цій області функція має два відносних максимуми, один з котрих є абсолютним максимумом. Якщо g неперервна по x_1 та x_2 то при зміні *x* положення точки x_m, яка доставляє абсолютний максимум, буде змінюватися разом із *x* неперервно до тих пір, доки не досягнеться значення *x*, для якого графік має вигляд, зображений на рис. 3.3.

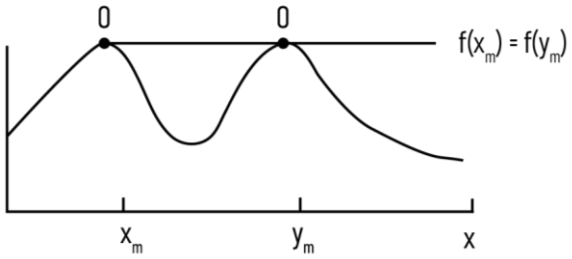

Рис. 3.3

Для цього значення *x* відносні максимуми рівні.

Припустимо тепер, що задля близького значення *x* значення $f\left(y_m\right)$ більше, ніж $f\left(x_m\right)$. В результаті положення абсолютного максимуму різко переміщується з околиці x_m в околицю y_m. Отже, x_m, що розглядається як функція від *x*, може мати точки розриву.

Цікавий приклад такого роду з'являється у зв'язку з рівнянням

$$f\left(x\right)=\max_{0\le y\le x}\left[g\left(y\right)+h\left(x-y\right)+f\left(ay+b\left(x-y\right)\right)\right]. \tag{3.44}$$

Якщо

$$\left.\begin{aligned}\mathrm{g}\left(y\right)&=e^{-10/y},\\ \mathrm{h}\left(y\right)&=e^{-15/y},\end{aligned}\right\} \tag{3.45}$$

то функція $f\left(x\right)$ є гладкою (рис. 3.4), в той час як функція політики $y\left(x\right)$ має вигляд, зображений на рис. 3.5.

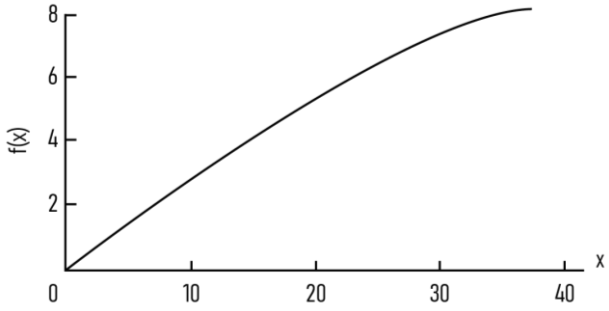

Рис. 3.4

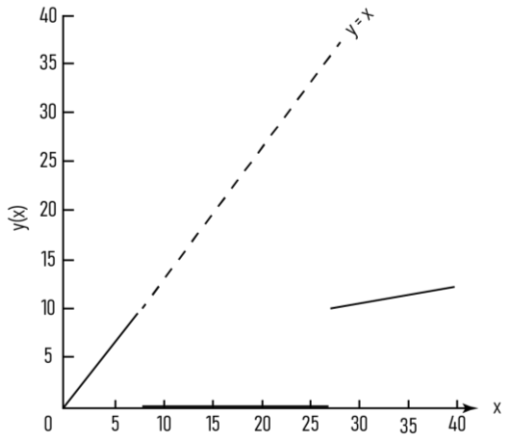

Рис. 3.5

При деяких значеннях x відбувається різка зміна в характері оптимальної політики.

Розглянемо поєднання методу послідовних наближень з *методом безперервності*, що є ще одним інструментом аналітики.

Для $x=0$ єдиним можливим вибором значень x_i є $x_i=0,\ i=1,2,...,N$ (задачі (3.38), (3.39)). Таким чином, задача максимізації по y_i є задача визначення максимуму функції

$$R_N(0,y)=\sum_{i=-1}^{N} g_i(0,y_i), \tag{3.46}$$

задача, яку ми розв'язуємо за допомогою послідовностей функції однієї змінної. Припустимо, що вона має єдиний розв'язок $y^0=\{y_i^0\}$. Для того, щоб розв'язати задачу з обмеженнями виду

$$\left.\begin{aligned}\sum_{i=1}^{N} x_i=\Delta,\quad x_i\ge 0,\\ \sum_{i=1}^{N} y_i=y,\end{aligned}\right\} \tag{3.47}$$

де Δ - «мале», ми виходимо з початкового наближення $y^0=\{y_i^0\}$ та приступаємо до максимізації по x_i. Отримавши таким шляхом розв'язок, ми повторюємо міркування для задачі з обмеженнями

$$\left.\begin{aligned}\sum_{i=1}^{N} x_i &= 2\Delta,\\ \sum_{i=1}^{N} y_i &= y,\end{aligned}\right\} \tag{3.48}$$

де в якості початкового наближення по y береться y – розв'язок задачі (3.47).

Разом з цим можна використати прямий метод для розв'язку задачі

$$\left.\begin{aligned}\sum_{i=1}^{N} x_i &= x_0,\\ \sum_{i=1}^{N} y_i &= y,\end{aligned}\right\} \tag{3.49}$$

для деякого більшого значення x_0 та області значень y, $0 \le y \le y_0$, та почати процес від цієї точки.

Коефіцієнти зв'язку

Іноді можна використовувати метод послідовних наближень іншим способом. Припустимо, що треба максимізувати функцію виду

$$\sum_{i=1}^{N} g_i(x_i) + \sum_{i=1}^{N} h_i(x_i) + t\sum_{i=1}^{N} k_i(x_i, y_i) \tag{3.50}$$

при обмеженнях

$$\sum_{i=1}^{N} x_i = x, \qquad \sum_{i=1}^{N} y_i = y, \qquad x_i,\ y_i \ \ge\ 0. \tag{3.51}$$

Тут t – параметр, який за припущенням приймає усі невід'ємні значення. Якщо $t = 0$, задача легко розв'язується мовою двох послідовностей функцій від однієї змінної.

Тому, задля того щоб розв'язати задачу для малого t, наприклад $t = \Delta$, треба використати в якості початкового наближення розв'язок $\{x_i^0, y_i^0\}$, який був отриманий при $t = 0$. Припустивши, що $x^0 = \{x_i^0\}$, можна спростити задачу пошуку, обмеживши галузь пошуку околицею точки $y^0 = \{y_i^0\}$. Отримаємо

розв’язок, що відповідає $t = \Delta$. Його можна використати в якості початкового наближення для випадку $t = 2\Delta$, і так далі.

3.8 Груба сітка

Інший шлях наближення до розв’язку задачі максимізації полягає в первісному використанні грубої сітки. Припустимо, що треба максимізувати функцію

$$g_1(x_1) + g_2(x_2) + \ldots + g_N(x_N) \tag{3.52}$$

при обмеженні

$$x_1 + x_2 + \ldots + x_N = x. \tag{3.53}$$

Нехай для початку x набуває тільки значення $0,\ \delta,\ 2\delta,\ \ldots$, де δ велике порівняно зі звичайним кроком сітки. Аналогічно, змінним x_i ми дозволяємо приймати ту саму множину значень. У результаті знаходження розв'язку прискорюється у двох напрямках. Тут підлягає табулюванню менше значень $f_N(x)$, і процес пошуку x_N, котрий максимізує, на кожному кроці коротший.

Таким чином, отримали множину максимізуючих значень $\{x_i(x)\},\ i = 1,\ 2,\ \ldots,\ N$, затабульованих у точках $x = 0,\ \delta,\ 2\delta,\ \ldots$.

Використання грубої сітки може призвести до пропуску дуже гострого абсолютного максимуму та помилково обрати за нього плоский відносний максимум. Наприклад, розглянемо функцію, що зображена на рис. 3.6.

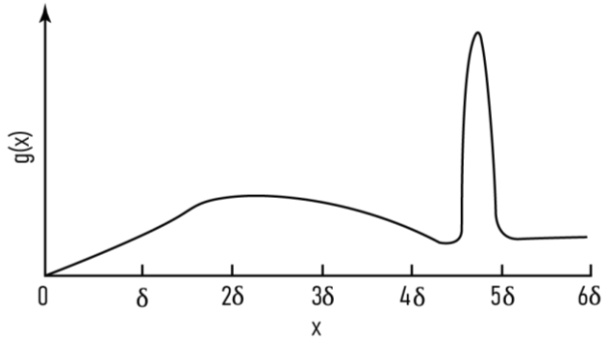

Рис. 3.6

Пошук при значеннях x, що дорівнюють $0,\ \delta,\ 2\delta,\ \ldots$, дає відносний максимум у точці 2δ, але не абсолютний максимум, що лежить між 4δ та 5δ.

Припустимо, що ми насправді визначили положення максимуму $x_i = x_i\left(x\right) = x_i\left(x;\delta\right)$ у межах від положення істинного максимуму. Тоді, використовуючи дрібнішу сітку, наприклад $0,\ \Delta,\ 2\Delta,\ \ldots$, де $\delta = 10\Delta$ або 100Δ, ми приєднуємо обмеження

$$x_i\left(x;\delta\right) - \delta \le x_i\left(x\right) \le x_i\left(x;\delta\right) + \delta, \quad i = 1,\ 2,\ \ldots,\ N, \tag{3.54}$$

де $x_i\left(x;\delta\right)$ - функції, що визначені початково при більш грубій сітці.

Обмеження такого типу зменшують вимоги до об’єму пам’яті та суттєво скорочують час на визначення максимуму процесом пошуку.

3.9 Послідовні наближення в задачі Хічкока-Купманса

Повернемося до задачі Хічкока-Купманса [Розділ 3]. Процес з M складами можна розглядати безпосередньо в термінах функцій від $M-1$ змінних та за допомогою множників Лагранжа – в термінах функцій від $M-k$ змінних.

Використаємо метод послідовних наближень для аналізу цієї задачі за допомогою послідовностей функцій від однієї змінної.

Нехай запаси на складах $D_3,\ D_4,\ \ldots,\ D_M$ розподілені так, щоб будь-яким способом задовольнити частину вимог у точках попиту. Тоді запаси, що залишилися на D_1 та D_2 розподіляються таким чином, щоб задовольнити за мінімальних витрат ті вимоги, що залишилися.

Задача може бути вирішена за допомогою обчислення послідовностей функцій від однієї змінної. Цей процес можна розглядати як стандартну операцію - перший крок у методі послідовних наближень. На складах $D_4,\ \ldots,\ D_M$ ми використовуємо ті ж розподіли, що і на першому складі D_1, визначені досягненням мінімуму, а на складах D_2 і D_3 ми вибираємо розподіли

так, щоб задовольняти залишковим вимогам при застосуванні процедури мінімізації. Це другий крок.

Далі фіксується розподіл на складах D_5, ..., D_M, D_1, використовуємо на складі D_2 розподіл, що визначається попередньою мінімізацією, а потім шукаємо розв'язок нової задачі мінімізації вартості перевезення зі складів D_3 та D_4. Потім цей процес продовжується таким же чином.

Послідовність функцій мінімальних витрат повинна сходитися, оскільки витрати на кожному кроці обчислень залишаються мінімальними. Проте процес її збіжності не є наочно явним, і можна знайти приклади, в яких мінімум не досягається таким чином.

В приведених далі прикладах метод полягає у тому, що спочатку фіксується стовпець x_{3j} при єдиній умові, що $\sum_j x_{3j} = x_3$, але не $x_{3j} > r_j$. У першому прикладі запас зі складу 3 було розподілено рівномірно між точками попиту. У другому прикладі ми задовольняємо попит кожного зі споживачів послідовно, витрачаючи запас на складі 3 до кінця. Потім ми віднімаємо від попиту кількості товару, отримані кожним з клієнтів, та розв'язуємо залишкову задачу з двома джерелами. Отриманий розв'язок використовується для визначення величини x_{1j} в другій ітерації. Цей розв'язок визначає x_{2j} для третьої ітерації і т.д. Цикл повторюється при фіксованому x_{3j}, поки всі три типи підзадач не досягнуть стійкого розподілу.

Розглянемо приклади з трьома складами та десятьма точками попиту.

Приклад I (відносний мінімум)

Розглянемо випадок, коли кожна з функцій g_{ij} має вигляд

$$g_{ij}(x) = a_{ij}x + b_{ij}x^2 + c_{ij}(x), \tag{3.55}$$

де $c_{ij}(x)$ - «організаційні» витрати, або фіксовані затрати, що дорівнюють нулеві, якщо $x = 0$, та постійній c_{ij}, якщо $x > 0$.

Коефіцієнти a_{ij} наведені у стовпці (x) таблиці 3.5, а b_{ij} - в стовпці (x^2).

Таблиця 3.5

Вартості перевезень у прикладі відносного мінімуму

Споживач	Джерело 1			Джерело 2			Джерело 3			Попит
	Організаційні витрати (c_{ij})	a_{ij}	b_{ij}	Організаційні витрати (c_{ij})	a_{ij}	b_{ij}	Організаційні витрати (c_{ij})	a_{ij}	b_{ij}	
1		+1,0		+2,0	+3,1			+7,0		25
2	+1,0	+2,0			+4,1			+3,0		40
3		+3,0	+0,01		+2,1			+9,0		60
4		+1,5			+1,1	+0,10		+1,0		30
5		+2,5			+2,6			+1,0		20
6	+10,0	+5,0	-0,01		+3,0		+5,0	+2,0		30
7		+3,0		+5,0	+1,0	+0,20		+4,0		35
8		+6,0			+2,0		+6,0	+3,0		30
9	+8,0	+6,0	-0,05		+2,0			+5,0		25
10		+6,0			+5,0	+0,01		+6,0		40
										335
x_1 - запас на складі 1 = 100, x_2 - запас на складі 2 = 80, x_3 - запас на складі 3 = 155.										

Для розрахунку використовувався код на мові програмування Java, що знаходиться в [Додатку В].

Загальний час роботи програми дорівнює в середньому 36,65 мілісекунд за 20 прогонів. Ці розрахунки були проведені на ноутбуці Lenovo IdeaPad L340 Gaming з процесором Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz та і розміром оперативної пам'яті (RAM) 16,0 ГБ.

Результати, отримані методом послідовних наближень, наведені в таблиці 3.6 [Додаток С].

Приклад II (абсолютний мінімум)

Розглянемо тепер випадок, коли усі функції $g_{ij}(x)$ опуклі, тобто будь-який відносний мінімум є абсолютним, та мають вигляд $a_{ij}x + b_{ij}x^2$. Чисельні дані у прикладі наведені в таблиці 3.7.

Таблиця 3.7

Вартості перевезень у прикладі абсолютного мінімуму

Споживач	Джерело 1			Джерело 2			Джерело 3			Попит
	Організаційні витрати (c_{ij})	a_{ij}	b_{ij}	Організаційні витрати (c_{ij})	a_{ij}	b_{ij}	Організаційні витрати (c_{ij})	a_{ij}	b_{ij}	
1		1,0	0,20		3,1	0,10		7,0		25
2		2,0	0,06		4,1			3,0	0,04	40
3		3,0	0,01		2,1			9,0		60
4		1,5			1,1	0,10		1,0		30
5		2,5	0,05		2,6			1,0	0,20	20
6		5,0	0,01		3,0			2,0		30
7		3,0			1,0	0,20		4,0		35
8		6,0			2,0			3,0	0,10	30
9		6,0	0,05		2,0			5,0		25
10		6,0			5,0	0,01		6,0		40
x_1 - запас на складі 1 = 100, x_2 - запас на складі 2 = 97, x_3 - запас на складі 3 = 138.										

Для розрахунку використовувався код на мові програмування Java, що знаходиться в [Додатку В].

Загальний час роботи програми дорівнює в середньому 41,13 мілісекунд за 20 прогонів. Ці розрахунки були проведені на ноутбуці Lenovo IdeaPad L340 Gaming з процесором Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz та і розміром оперативної пам'яті (RAM) 16,0 ГБ.

Результати ітерацій наведені в таблиці 2.8 [Додаток D].

Цікаво зазначити, що точність на рівні 4% досягається на третій ітерації, абсолютний мінімум було досягнуто лише після двадцяти восьми ітерацій. Звернувши увагу на двадцять другу та двадцять третю ітерації, було помічено, що під час фіксування розподілу на першому складі не було покращення, проте зміна розподілу на другому складі привела до значного покращення результатів.

ВИСНОВКИ

В кваліфікаційній роботі були розглянуті методи розв’язання транспортної задачі Хічкока-Купманса, яка зводиться до розв’язання задачі багатокрокової оптимізації з нелінійними цільовими функціями на всіх етапах, окремо було розглянуто випадки, коли розподіл здійснюється з одного, двох або трьох складів, тобто стан в моделі динамічного програмування для цієї задачі є одновимірним, двовимірним та тривимірним.

Реалізував методи розв’язання транспортної задачі, а саме метод динамічного програмування, поєднаний з методом множників Лагранжа, або поєднаний з методом послідовних наближень. Програмні коди були написані на мові програмування Java.

Було проаналізовано результати обчислювальних експериментів за різними методами. Було з’ясовано, що комбінування методу динамічного програмування з методом множників Лагранжа або з методом послідовних наближень дозволяє зменшити вимірність стану, а тим самим відповідно зменшити обсяги пам’яті, необхідної для зберігання проміжних обчислень, та зменшити час отримання оптимального розв’язку задачі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Introduction to Algorithms 4th Edition / Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, The MIT Press, Cambridge, Massachusetts London, England, 2022. 1312 p.

2. Григорків В.С., Григорків М.В., Ярошенко О.І. Оптимізаційні методи та моделі: підручник / Чернівецький національний університет. Чернівці : Черн. ун-т, 2022. 440 с.

3. Соколов С.В. Оптимальні та адаптивні системи: навчальний посібник / Сумський державний університет. Суми : Сум. ун-т, 2018. 221 с.

4. Математичні методи дослідження операцій : підручник / Є. А. Лавров [та ін.]. Сумський державний університет. Суми : Сум. ун-т, 2017. 212 с.

5. G. Hadley. Nonlinear and dynamic programming / Addison-Wesley Publishing Company, Massachusetts, 1964. 496 p.

6. Ravindra K. Ahuja, Thomas L. Magnanti, James B. Orlin. Network Flows: Theory, Algorithms, and Applications / Prentice-Hall, Inc. Upper Saddle River, New Jersey, 1993. 863 p.

7. Richard L. Burden, J. Douglas Faires. Numerical Analysis, Ninth Edition / Brooks/Cole, 20 Channel Center Street, Boston, Massachusetts, 2010. 895 p.

8. Richard E. Bellman, Stuart E. Dreyfus. Applied Dynamic Programming / London: Oxford University Press, 1962. 468 p.

9. Richard E. Bellman. Dynamic Programming / Princeton University Press, Princeton, New Jersey, 1957. 365 p.

10. Dimitri P. Bertsekas. Dynamic Programming and Optimal Control / Athena Scientific, 2012. 840 p.

11. Nocedal Jorge, Wright, Stephen J. Numerical optimization / New York: Springer, 2nd edition, 2006. 683 p.

12. Sniedovich M. Dynamic Programming: Foundations and Principles / Taylor & Francis, 2nd edition, 2010. 624 p. (ISBN 978-0-8247-4099-3)

Додаток А

Код алгоритму методу ДП з методом множників Лагранжа

```
import java.util.Scanner;

public class TransportProblemSolverDynProg {

    private static double[][] c_ij = {
            {0, 1, 0, 0, 0, 10, 0, 0, 8, 0},
            {2, 0, 0, 0, 0, 0, 5, 0, 0, 0},
            {0, 0, 0, 0, 0, 5, 0, 6, 0, 0}
    };

    private static double[][] a_ij = {
            {1, 2, 3, 1.5, 2.5, 5, 3, 6, 6, 6},
            {3.1, 4.1, 2.1, 1.1, 2.6, 3.0, 1.0, 2.0, 2.0, 5.0},
            {7, 3, 9, 1, 1, 2, 4, 3, 5, 6}
    };

    private static double[][] b_ij = {
            {0, 0, 0.01, 0, 0, -0.01, 0, 0, -0.05, 0},
            {0, 0, 0, 0.1, 0, 0, 0.2, 0, 0, 0.01},
            {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
    };

    private static int numConsumers = 10;
    private static int numWarehouses = 3;


    private static double[] supply = {100, 80, 155}; // кількість ресурсів на
складах
    private static double[] demand = {25, 40, 60, 30, 20, 30, 35, 30, 25, 40};
// попит на ресурси в пунктах споживання

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.println("Введіть кількість складів: ");
        int inputNumberOfWarehouse = scanner.nextInt();
        double[] inputSupply = new double[inputNumberOfWarehouse];
        System.out.println("Введіть кількість ресурсів на кожному складі: ");
        for (int i = 0; i < inputNumberOfWarehouse; i++) {
            System.out.println("Склад " + (i + 1) + ":");
            inputSupply[i] = scanner.nextDouble();
        }

        System.out.println("Введіть кількість пунктів споживання: ");
        int inputNumberOfConsumer = scanner.nextInt();
        double[] inputDemand = new double[inputNumberOfConsumer];
        System.out.println("Введіть попит в кожному пункті споживання: ");
        for (int i = 0; i < inputNumberOfConsumer; i++) {
            System.out.println("Пункт споживання " + (i + 1) + ":");
            inputDemand[i] = scanner.nextDouble();
        }

        double[][] inputA_ij = new
double[inputNumberOfWarehouse][inputNumberOfConsumer];
        System.out.println("Введіть коефіцієнти a_ij для кожного складу та
пункту споживання: ");
        for (int i = 0; i < inputNumberOfWarehouse; i++) {
```

```
            System.out.println("Для " + (i + 1) + "-ого складу:");
            for (int j = 0; j < inputNumberOfConsumer; j++) {
                System.out.println("Для " + (j + 1) + "-ого пункту
споживання:");
                inputA_ij[i][j] = scanner.nextDouble();
            }
        }

        double[][] inputB_ij = new
double[inputNumberOfWarehouse][inputNumberOfConsumer];
        System.out.println("Введіть коефіцієнти b_ij для кожного складу та
пункту споживання: ");
        for (int i = 0; i < inputNumberOfWarehouse; i++) {
            System.out.println("Для " + (i + 1) + "-ого складу:");
            for (int j = 0; j < inputNumberOfConsumer; j++) {
                System.out.println("Для " + (j + 1) + "-ого пункту
споживання:");
                inputB_ij[i][j] = scanner.nextDouble();
            }
        }

        double[][] inputC_ij = new
double[inputNumberOfWarehouse][inputNumberOfConsumer];
        System.out.println("Введіть коефіцієнти c_ij (або організаційні витрати)
для кожного складу та пункту споживання: ");
        for (int i = 0; i < inputNumberOfWarehouse; i++) {
            System.out.println("Для " + (i + 1) + "-ого складу:");
            for (int j = 0; j < inputNumberOfConsumer; j++) {
                System.out.println("Для " + (j + 1) + "-ого пункту
споживання:");
                inputC_ij[i][j] = scanner.nextDouble();
            }
        }

        numConsumers = inputNumberOfConsumer;
        numWarehouses = inputNumberOfWarehouse;
        supply = inputSupply;
        demand = inputDemand;
        a_ij = inputA_ij;
        b_ij = inputB_ij;
        c_ij = inputC_ij;

        long time = System.currentTimeMillis();
        double[][] result = solveTransportProblem(numWarehouses, numConsumers,
supply, demand);

        // Виводимо отриманий оптимальний розподіл ресурсів
        System.out.println("Оптимальний розподіл ресурсів:");
        double totalCost = 0.0;
        for (int i = 0; i < numWarehouses; i++) {
            for (int j = 0; j < numConsumers; j++) {
                System.out.println("Зі складу " + (i + 1) + " у пункт споживання
" + (j + 1) + ": " + result[i][j]);
                totalCost += calculateCost(i, j, result[i][j]);
            }
        }
        System.out.println("Загальна вартість операцій: " + totalCost);
        System.out.println("Час рахування: " + (System.currentTimeMillis() -
time) + " ms");
    }

    public static double[][] solveTransportProblem(int numWarehouses, int
numConsumers, double[] supply, double[] demand) {
```

```
        double[][] result = new double[numWarehouses][numConsumers];

        double[] remainingSupply = supply.clone();
        double[] remainingDemand = demand.clone();

        while (true) {
            // Знаходимо найменшу вартість перевезення
            double minCost = Double.MAX_VALUE;
            int minWarehouseIndex = -1;
            int minConsumerIndex = -1;

            for (int i = 0; i < numWarehouses; i++) {
                for (int j = 0; j < numConsumers; j++) {
                    if (remainingSupply[i] > 0 && remainingDemand[j] > 0) {
                        double optimalAmount = Math.min(remainingSupply[i],
remainingDemand[j]);
                        double cost = calculateCost(i, j, optimalAmount);
                        if (cost < minCost) {
                            minCost = cost;
                            minWarehouseIndex = i;
                            minConsumerIndex = j;
                        }
                    }
                }
            }

            if (minCost == Double.MAX_VALUE) {
                break; // Усі ресурси розподілені
            }

            // Виконуємо перевезення з мінімальною вартістю
            double amountToAllocate =
Math.min(remainingSupply[minWarehouseIndex], remainingDemand[minConsumerIndex]);
            result[minWarehouseIndex][minConsumerIndex] = amountToAllocate;
            remainingSupply[minWarehouseIndex] -= amountToAllocate;
            remainingDemand[minConsumerIndex] -= amountToAllocate;
        }

        return result;
    }

    private static double calculateCost(int warehouseIndex, int consumerIndex,
double amount) {
        double cost;
        cost = amount * (
                a_ij[warehouseIndex][consumerIndex]
              + b_ij[warehouseIndex][consumerIndex] *
b_ij[warehouseIndex][consumerIndex]
            ) + c_ij[warehouseIndex][consumerIndex];

        return cost;
    }
}
```

Додаток В

Код алгоритму ДП з методом послідовних наближень

```
import java.util.Arrays;
import java.util.Scanner;

public class TransportationProblemSolver {
    private static double[][] c_ij = {
            {0, 1, 0, 0, 0, 10, 0, 0, 8, 0},
            {2, 0, 0, 0, 0, 0, 5, 0, 0, 0},
            {0, 0, 0, 0, 0, 5, 0, 6, 0, 0}
    };

    private static double[][] a_ij = {
            {1, 2, 3, 1.5, 2.5, 5, 3, 6, 6, 6},
            {3.1, 4.1, 2.1, 1.1, 2.6, 3.0, 1.0, 2.0, 2.0, 5.0},
            {7, 3, 9, 1, 1, 2, 4, 3, 5, 6}
    };

    private static double[][] b_ij = {
            {0, 0, 0.01, 0, 0, -0.01, 0, 0, -0.05, 0},
            {0, 0, 0, 0.1, 0, 0, 0.2, 0, 0, 0.01},
            {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
    };

    private static int numConsumers = 10;
    private static int numWarehouses = 3;

    private static double[] supply = {100, 80, 155};
    private static double[] demand = {25, 40, 60, 30, 20, 30, 35, 30, 25, 40};

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.println("Введіть кількість складів: ");
        int inputNumberOfWarehouse = scanner.nextInt();
        double[] inputSupply = new double[inputNumberOfWarehouse];
        System.out.println("Введіть кількість ресурсів на кожному складі: ");
        for (int i = 0; i < inputNumberOfWarehouse; i++) {
            System.out.println("Склад " + (i + 1) + ":");
            inputSupply[i] = scanner.nextDouble();
        }

        System.out.println("Введіть кількість пунктів споживання: ");
        int inputNumberOfConsumer = scanner.nextInt();
        double[] inputDemand = new double[inputNumberOfConsumer];
        System.out.println("Введіть попит в кожному пункті споживання: ");
        for (int i = 0; i < inputNumberOfConsumer; i++) {
            System.out.println("Пункт споживання " + (i + 1) + ":");
            inputDemand[i] = scanner.nextDouble();
        }

        double[][] inputA_ij = new
double[inputNumberOfWarehouse][inputNumberOfConsumer];
        System.out.println("Введіть коефіцієнти a_ij для кожного складу та
пункту споживання: ");
        for (int i = 0; i < inputNumberOfWarehouse; i++) {
            System.out.println("Для " + (i + 1) + "-ого складу:");
            for (int j = 0; j < inputNumberOfConsumer; j++) {
```

```
                System.out.println("Для " + (j + 1) + "-ого пункту
споживання:");
                inputA_ij[i][j] = scanner.nextDouble();
            }
        }

        double[][] inputB_ij = new
double[inputNumberOfWarehouse][inputNumberOfConsumer];
        System.out.println("Введіть коефіцієнти b_ij для кожного складу та
пункту споживання: ");
        for (int i = 0; i < inputNumberOfWarehouse; i++) {
            System.out.println("Для " + (i + 1) + "-ого складу:");
            for (int j = 0; j < inputNumberOfConsumer; j++) {
                System.out.println("Для " + (j + 1) + "-ого пункту
споживання:");
                inputB_ij[i][j] = scanner.nextDouble();
            }
        }

        double[][] inputC_ij = new
double[inputNumberOfWarehouse][inputNumberOfConsumer];
        System.out.println("Введіть коефіцієнти c_ij (або організаційні витрати)
для кожного складу та пункту споживання: ");
        for (int i = 0; i < inputNumberOfWarehouse; i++) {
            System.out.println("Для " + (i + 1) + "-ого складу:");
            for (int j = 0; j < inputNumberOfConsumer; j++) {
                System.out.println("Для " + (j + 1) + "-ого пункту
споживання:");
                inputC_ij[i][j] = scanner.nextDouble();
            }
        }

        numConsumers = inputNumberOfConsumer;
        numWarehouses = inputNumberOfWarehouse;
        supply = inputSupply;
        demand = inputDemand;
        a_ij = inputA_ij;
        b_ij = inputB_ij;
        c_ij = inputC_ij;

        long time = System.currentTimeMillis();
        int[] consumers = new int[numWarehouses];

        for (int i = 0; i < numWarehouses; i++) {
            consumers[i] = numConsumers / numWarehouses;
        }

        double[][] transportation = new double[numWarehouses][numConsumers];
        double totalCost = 0;

        // Ітеративне обчислення оптимального транспорту
        while (true) {
            double[][] previousTransportation = transportation.clone();
            transportation = findOptimalTransportation(transportation);

            // Перевірка на оптимальність
            double maxDifference = 0;
            for (int i = 0; i < numWarehouses; i++) {
                for (int j = 0; j < consumers[i]; j++) {
                    double difference = Math.abs(transportation[i][j] -
previousTransportation[i][j]);
                    if (difference > maxDifference) {
                        maxDifference = difference;
```

```
                    }
                }
            }

            if (maxDifference == 0) {
                break; // Досягнення оптимального рішення
            }
        }

        // Розрахунок загальних витрат на транспортування
        for (int i = 0; i < numWarehouses; i++) {
            for (int j = 0; j < consumers[i]; j++) {
                totalCost += transportation[i][j] * (a_ij[i][j] + b_ij[i][j] *
b_ij[i][j]) + c_ij[i][j];
            }
        }

        // Виведення результатів
        System.out.println("Оптимальний розподіл ресурсів:");
        for (int i = 0; i < numWarehouses; i++) {
            System.out.println("Склад " + (i + 1) + ": " +
Arrays.toString(transportation[i]));
        }
        System.out.println("Загальні витрати на транспортування: " + totalCost);
        System.out.println("Час рахування: " + (System.currentTimeMillis() -
time) + " ms");
    }

    private static double[][] findOptimalTransportation(double[][]
transportation) {
        int[] consumers = new int[numWarehouses];

        for (int i = 0; i < numWarehouses; i++) {
            consumers[i] = numConsumers / numWarehouses;
        }

        double[][] costs = new double[numWarehouses][];
        double[][] adjustedCosts = new double[numWarehouses][];
        double[][] normalizedCosts = new double[numWarehouses][];
        double[][] normalizedTransportation = new double[numWarehouses][];

        double[] supplyCopy = Arrays.copyOf(supply, numWarehouses);

        for (int i = 0; i < numWarehouses; i++) {
            costs[i] = new double[consumers[i]];
            adjustedCosts[i] = new double[consumers[i]];
            normalizedCosts[i] = new double[consumers[i]];
            normalizedTransportation[i] = new double[consumers[i]];
        }

        // Розрахунок вартостей і коригування
        for (int i = 0; i < numWarehouses; i++) {
            for (int j = 0; j < consumers[i]; j++) {
                costs[i][j] = a_ij[i][j] + b_ij[i][j] * b_ij[i][j] + c_ij[i][j];
                adjustedCosts[i][j] = costs[i][j] - b_ij[i][j] * b_ij[i][j];
            }
        }

        // Нормалізація вартостей і розрахунок нормалізованого транспорту
        while (true) {
            double maxAdjustedCost = 0;

            for (int i = 0; i < numWarehouses; i++) {
```

```
                for (int j = 0; j < consumers[i]; j++) {
                    if (adjustedCosts[i][j] > maxAdjustedCost) {
                        maxAdjustedCost = adjustedCosts[i][j];
                    }
                }
            }

            if (maxAdjustedCost == 0) {
                break; // Нормалізацію завершено
            }

            for (int i = 0; i < numWarehouses; i++) {
                for (int j = 0; j < consumers[i]; j++) {
                    normalizedCosts[i][j] = adjustedCosts[i][j] /
maxAdjustedCost;
                    normalizedTransportation[i][j] = Math.min(supplyCopy[i],
demand[j]);
                    transportation[i][j] += normalizedTransportation[i][j];
                    supplyCopy[i] -= normalizedTransportation[i][j];
                    demand[j] -= normalizedTransportation[i][j];
                }
            }

            for (int i = 0; i < numWarehouses; i++) {
                for (int j = 0; j < consumers[i]; j++) {
                    adjustedCosts[i][j] = costs[i][j] - b_ij[i][j] * b_ij[i][j]
- normalizedCosts[i][j] * maxAdjustedCost;
                }
            }

            // Додатковий цикл для розподілу ресурсів, що залишилися, і
задоволення незадоволеного попиту
            for (int i = 0; i < numWarehouses; i++) {
                for (int j = consumers[i]; j < numConsumers; j++) {
                    if (supplyCopy[i] > 0 && demand[j] > 0) {
                        double additionalTransport = Math.min(supplyCopy[i],
demand[j]);
                        transportation[i][j] += additionalTransport;
                        supplyCopy[i] -= additionalTransport;
                        demand[j] -= additionalTransport;
                    }
                }
            }

            // Перевірка на завершення циклу
            if (maxAdjustedCost == 0) {
                break; // Нормалізацію завершено
            }
        }

        return transportation;
    }
}
```

Додаток С

Таблиця 3.6

Результати, отримані методом послідовних наближень

	Політика 1			Політика 2			Політика 3			Політика 4		
1	10	0	15	10	15	0	10	15	0	25	0	0
2	25	0	15	25	0	15	40	0	0	40	0	0
3	5	40	15	5	55	0	5	55	0	0	60	0
4	15	0	15	15	0	15	0	0	30	0	0	30
5	5	0	15	5	0	15	0	0	20	0	0	20
6	0	15	15	0	0	30	0	0	30	0	0	30
7	20	0	15	20	0	15	35	0	0	35	0	0
8	0	15	15	0	0	30	0	0	30	0	0	30
9	0	10	15	0	10	15	0	10	15	0	10	15
10	20	0	20	20	0	20	10	0	30	0	10	30
	Витрати = = 1126,25			Витрати = = 966,00			Витрати = = 921,25			Витрати = 874,00		
	Політика 5			Політика 6			Політика 7			Точний розв’язок		
1	25	0	0	25	0	0	25	0	0	25	0	0
2	40	0	0	40	0	0	40	0	0	40	0	0
3	0	60	0	0	60	0	0	60	0	5	55	0
4	0	0	30	0	0	30	0	0	30	0	0	30
5	0	0	20	0	0	20	0	0	20	0	0	20
6	0	0	30	0	0	30	0	0	30	0	0	30
7	35	0	0	35	0	0	35	0	0	30	0	5
8	0	0	30	0	0	30	0	0	30	0	0	30
9	0	20	5	0	20	5	0	20	5	0	25	0
10	0	0	40	0	0	40	0	0	40	0	0	40
	Витрати = = 853,00			Витрати = = 853,00			Витрати = = 853,00			Витрати = = 847,75		

Додаток D

Таблиця 3.8

Результати, отримані методом послідовних наближень

	Політика 1			Політика 2			Політика 3			Політика 4		
1	0	0	25	0	4	21	15	4	6	12	7	6
2	0	0	40	0	0	40	21	0	19	21	0	19
3	0	0	60	0	60	0	0	60	0	7	53	0
4	17	0	13	17	0	13	0	0	30	0	0	30
5	11	9	0	11	0	9	13	0	7	11	2	7
6	0	30	0	0	0	30	0	0	30	0	0	30
7	32	3	0	32	0	3	35	0	0	33	2	0
8	0	30	0	0	20	10	0	20	10	0	20	10
9	0	25	0	0	13	12	0	13	12	0	13	12
10	40	0	0	40	0	0	16	0	24	16	0	24
	Витрати = = 1535,249			Витрати = = 1141,25			Витрати = = 1040,55			Витрати = = 1035,439		
	Політика 5			Політика 6			Політика 7			Політика 8		
1	12	4	9	15	4	6	12	7	6	12	4	9
2	21	0	19	21	0	19	21	0	19	21	0	19
3	7	53	0	7	53	0	12	48	0	12	48	0
4	0	0	30	0	0	30	0	0	30	0	0	30
5	11	0	9	13	0	7	13	0	7	13	0	7
6	0	0	30	0	0	30	0	0	30	0	0	30
7	33	0	2	35	0	0	33	2	0	33	0	2
8	0	20	10	0	20	10	0	20	10	0	20	10
9	0	20	5	0	20	5	0	20	5	0	25	0
10	16	0	24	9	0	31	9	0	31	9	0	31
	Витрати = = 1031,24			Витрати = = 1026,44			Витрати = = 1022,089			Витрати = = 1020,69		

Продовження таблиці 3.8

	Політика 9			Політика 10			Політика 11			Політика 12		
1	15	4	6	12	7	6	12	6	7	15	6	4
2	21	0	19	21	0	19	21	0	19	21	0	19
3	12	48	0	17	43	0	17	43	0	17	43	0
4	0	0	30	0	0	30	0	0	30	0	0	30
5	13	0	7	13	0	7	13	0	7	13	0	7
6	0	0	30	0	0	30	0	0	30	0	0	30
7	35	0	0	33	2	0	33	1	1	34	1	0
8	0	20	10	0	20	10	0	22	8	0	22	8
9	0	25	5	0	25	0	0	25	0	0	25	0
10	4	0	36	4	0	36	4	0	36	0	0	40
	Витрати = = 1016,89			Витрати = = 1013,04			Витрати = = 1012,44			Витрати = = 1009,64		
	Політика 13			Політика 14			Політика 15			Політика 16		
1	13	8	4	13	7	5	14	7	4	13	8	4
2	21	0	19	21	0	19	20	0	20	20	0	20
3	20	40	0	20	40	0	20	40	0	21	39	0
4	0	0	30	0	0	30	0	0	30	0	0	30
5	13	0	7	13	0	7	13	0	7	13	0	7
6	0	0	30	0	0	30	0	0	30	0	0	30
7	33	20	0	33	2	0	33	2	0	33	2	0
8	0	22	8	0	23	7	0	23	7	0	23	7
9	0	25	0	0	25	0	0	25	0	0	25	0
10	0	0	40	0	0	40	0	0	40	0	0	40
	Витрати = = 1007,85			Витрати = = 1007,75			Витрати = = 1007,25			Витрати = = 1006,76		

Продовження таблиці 3.8

	Політика 17			Політика 18			Політика 19			Політика 20		
1	13	8	4	14	8	3	13	9	3	13	8	4
2	20	0	20	20	0	20	20	0	20	20	0	20
3	21	39	0	21	39	0	22	38	0	22	38	0
4	0	0	30	0	0	30	0	0	30	0	0	30
5	13	0	7	12	0	8	12	0	8	12	0	8
6	0	0	30	0	0	30	0	0	30	0	0	30
7	33	2	0	33	2	0	33	2	0	33	2	0
8	0	23	7	0	23	7	0	23	7	0	24	6
9	0	25	0	0	25	0	0	25	0	0	25	0
10	0	0	40	0	0	40	0	0	40	0	0	40
	Витрати = = 1006,76			Витрати = = 1006,41			Витрати = = 1006,14			Витрати = = 1006,04		
	Політика 21			Політика 22			Політика 23			Політика 24		
1	14	8	3	13	9	3	13	9	3	14	9	2
2	19	0	21	19	0	21	19	0	21	18	0	22
3	22	38	0	23	37	0	23	37	0	23	37	0
4	0	0	30	0	0	30	0	0	30	0	0	30
5	12	0	8	12	0	8	12	0	8	12	0	8
6	0	0	30	0	0	30	0	0	30	0	0	30
7	33	2	0	33	2	0	33	2	0	33	2	0
8	0	24	6	0	24	6	0	24	6	0	24	6
9	0	25	0	0	25	0	0	25	0	0	25	0
10	0	0	40	0	0	40	0	0	40	0	0	40
	Витрати = = 1005,74			Витрати = = 1005,49			Витрати = = 1005,49			Витрати = = 1005,39		

Продовження таблиці 3.8

	Політика 25			Політика 26			Політика 27			Політика 28		
1	13	10	2	13	9	3	13	9	3	13	9	3
2	18	0	22	18	0	22	18	0	22	18	0	22
3	24	36	0	24	36	0	24	36	0	24	36	0
4	0	0	30	0	0	30	0	0	30	0	0	30
5	12	0	8	12	0	8	12	0	8	12	0	8
6	0	0	30	0	0	30	0	0	30	0	0	30
7	33	2	0	33	2	0	33	2	0	33	2	0
8	0	24	6	0	25	5	0	25	5	0	25	5
9	0	25	0	0	25	0	0	25	0	0	25	0
10	0	0	40	0	0	40	0	0	40	0	0	40
	Витрати = = 1005,36			Витрати = = 1005,26			Витрати = = 1005,26			Витрати = = 1005,26		