НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра прикладної математики

Звіт

до лабораторної роботи

із дисципліни «АЛГОРИТМИ І СИСТЕМИ КОМП'ЮТЕРНОЇ МАТЕМАТИКИ 1.МАТЕМАТИЧНІ АЛГОРИТМИ»

на тему:

ЛІНІЙНЕ ПРОГРАМУВАННЯ

Виконав:
студент групи КМ-13
Онищенко В.С.

Перевірила:
асистент кафедри ПМА
Ковальчук-Химюк Л.О.

Київ — 2024

ЗМІСТ

1 ВСТУП **3**

2 ПОРЯДОК ВИКОНАННЯ РОБОТИ **4**

3 ОСНОВНА ЧАСТИНА **5**

3.1 Вхідні дані 5

3.2 Типовий перелік вимог до ПЗ 6

3.3 Опис методів 8

3.3.1 Задача про суміші 8

3.4 Блок-схеми алгоритмів методів 10

3.5 Перевірка розробленої програми 11

3.5.1 Python 11

3.5.2 Octave 11

3.6 Приклад роботи програми 12

3.6.1 Python 12

3.6.2 Octave 12

4 ВИСНОВКИ **13**

5 СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ **14**

6 ДОДАТОК А (код програм) **15**

1. ВСТУП

Мета роботи: побудова математичних моделей лінійного програмування, практичне розв’язання задач лінійного програмування з використанням СКМ.

Для розробки використати мови Python [1] та Octave [2].

2. ПОРЯДОК ВИКОНАННЯ РОБОТИ

1. Ознайомитися з теоретичним матеріалом і методичними вказівками до роботи.

2. Одержати варіант індивідуального завдання /вихідні дані для задачі, метод розв’язання задачі. Формалізувати задачу лінійного програмування: скласти математичну модель у загальному вигляді; використовуючи конкретні числові дані перетворити вихідну математичну модель до вигляду, що допускає застосування підпрограм СКМ.

3. Розробити блок-схему алгоритму розв’язання задачі.

4. Написати, налагодити і виконати програму. У програмі передбачити опис усіх використовуваних масивів і введення значень їх елементів, вивід вихідних даних, результатів розрахунків і значення ознаки результату.

5. Розв’язати задачу лінійного програмування.

6. Оформити звіт по роботі.

3. ОСНОВНА ЧАСТИНА

3.1. Вхідні дані

Варіант №6 (табл. 3.1.1)

Таблиця 3.1.1 – Початкові дані за варіантом №6

Задача про суміші

Варіант	Види ресурсів	Кількість одиниць поживних речовин на одиницю продукції				Максимальна норма поживних речовин	Вартість одиниці продукції			
		P_1	P_2	P_3	P_4		C_{P1}	C_{P2}	C_{P3}	C_{P4}
6	S_1	3	1	1	-	9				
	S_2	1	2	-	-	8	5	6	-	-
	S_3	1	6	-	-	12				

3.2. Типовий перелік вимог до ПЗ

Програмне забезпечення, розроблюване в рамках виконання кожної лабораторної роботи, повинно задовольняти низку вимог, які можна розділити на *обов'язкові* (які ПЗ повинно задовольняти незалежно від лабораторної роботи) та *варіативні* (які для кожної лабораторної роботи унікальні). До обов'язкових вимог належать:

- У програмі повинно бути передбачено перевірки на некоректне введення для всіх полів введення, зокрема:

    1. порожнє введення;
    2. синтаксично некоректне введення (наприклад, літери в полі для числових коефіцієнтів);
    3. уведення спеціальних символів;
    4. уведення чисел, які перевищують максимальний розмір для чисел відповідного типу даних (для перевірки на переповнення розрядної сітки).

    У випадку некоректного введення повинно з'являтися діалогове вікно з відповідним повідомленням.

- Для всіх полів введення повинно бути визначено гранично допустиму кількість символів (для числових полів — гранично допустимі значення).

- У програмі повинно відслідковуватися переповнення розрядної сітки під час виконання обчислень. У випадку переповнення повинно з'являтися діалогове вікно з відповідним повідомленням.

- У графічному інтерфейсі користувача повинно бути передбачено можливість гнучкого налаштування розмірності розв’язуваної задачі (наприклад, можливість зміни розмірності матриці чи кількості складів у транспортній задачі).

До варіативних вимог належать вимоги щодо перевірки на коректне опрацювання виключних ситуацій, які можуть виникати під час застосування заданого методу до розв’язання поставленої задачі (наприклад, коли сума заявок не збігається з сумою ресурсів у транспортній задачі, нижня межа інтегрування перевищує верхню тощо).

3.3. Опис методів

3.3.1. Задача про суміші

Розглядаються чотири типи продукції: P_1, P_2, P_3, P_4, для виготовлення яких використовуються три види ресурсів: S_1, S_2, S_3. Кожен вид продукції вимагає певну кількість відповідних ресурсів. Завдання полягає в мінімізації загальних витрат на виробництво, з одночасним дотриманням обмежень щодо максимально доступної кількості кожного ресурсу. [3]

Позначення

- x_j — кількість одиниць продукції P_j, де $j = 1, 2, 3, 4$.
- C_{P_j} — ціна одиниці продукції P_j, де $j = 1, 2, 3, 4$.
- a_{ij} — кількість одиниць ресурсу S_i, які необхідні для виробництва продукції P_j.
- b_i — максимальна кількість ресурсу S_i, яку можна використати.

Побудова математичної моделі

Оптимізація Поживних Речовин

Математична модель:

Мінімізувати:

$$Z = 5x_1 + 6x_2$$

За умов:

$$\begin{aligned} 3x_1 + x_2 + x_3 &\geq 9, && \text{(Потреба у Поживній Речовині 1)} \\ x_1 + 2x_2 &\geq 8, && \text{(Потреба у Поживній Речовині 2)} \\ x_1 + 6x_2 &\geq 12, && \text{(Потреба у Поживній Речовині 3)} \end{aligned}$$

Змінні:

$$x_1, x_2, x_3 \geq 0, \quad x_1, x_2, x_3 \in \mathbb{R} \text{ (неперервні).}$$

3.4. Блок-схеми алгоритмів методів

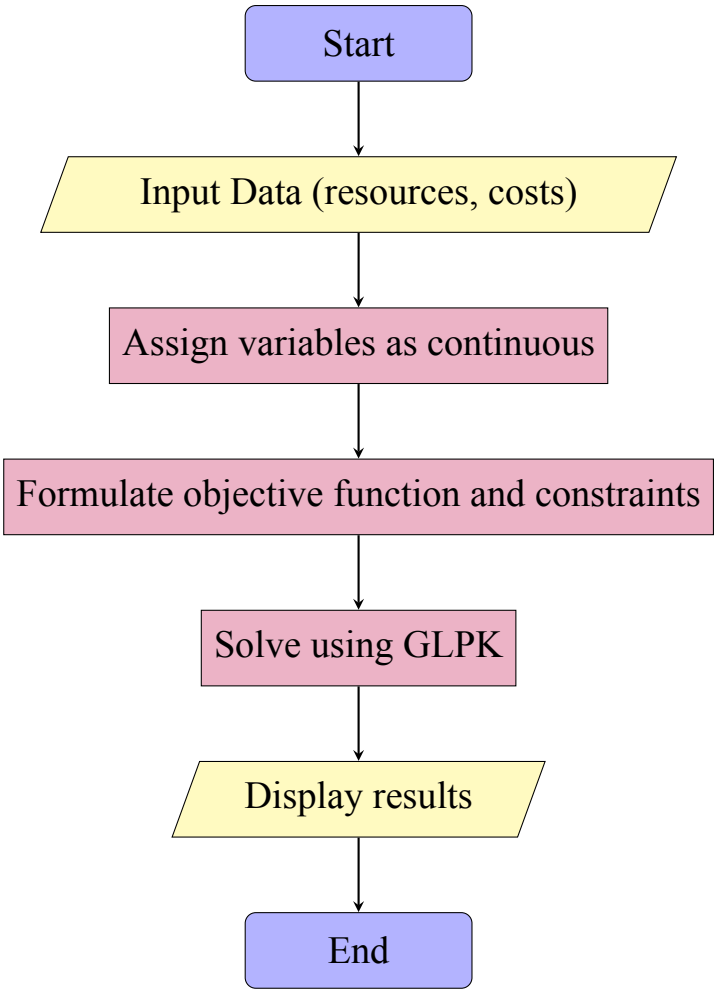

Рисунок 3.4.1 – Блок-схема алгоритму оптимізаційної задачі

3.5. Перевірка розробленої програми

3.5.1. Python

- реалізація перевірки введених даних на відповідність числовому формату
- контроль за відповідністю розмірів матриць у процесі обчислень
- додаткові перевірки для запобігання наявності від'ємних значень у таблицях вартостей та максимальних норм

3.5.2. Octave

Відмітимо, що для програм, які написані мовою Octave немає вбудованої функціональності для автоматичної перевірки коректності введених даних.

3.6. Приклад роботи програми

3.6.1. Python

Для вирішення цієї оптимізаційної задачі розробимо програмне забезпечення, використовуючи бібліотеку PuLP [4]. Виконання програми дасть нам кількість одиниць кожного виду продукції та мінімальну вартість (рис. 3.6.1.1). Програма буде розраховувати результат для дійсних чисел, щоб уникнути повторних розрахунків для цілочисельного лінійного програмування, якщо результат виявиться цілим числом. Як можна побачити, отримані значення виявились цілими. Повний код і більш детальні результати можна знайти у *додатку до роботи*.

```
Optimal solution determined:
Variable Product_1: 0.00
Variable Product_2: 4.00
Variable Product_3: 5.00
Objective Function Value (Total Cost): 24.00
```

Рисунок 3.6.1.1 – Результат даної задачі мінімізації про суміші (Python)

3.6.2. Octave

Програма, яка реалізована за допомогою мови Octave має такі ж результати, як і на Python (рис. 3.6.2.1)

```
Optimal solution determined:
Variable Product_1: 0.00
Variable Product_2: 4.00
Variable Product_3: 5.00
Objective Function Value (Total Cost): 24.00
```

Рисунок 3.6.2.1 – Результат даної задачі мінімізації про суміші (Octave)

4. ВИСНОВКИ

Під час виконання цієї лабораторної роботи було отримано практичні навички створення математичних моделей лінійного програмування та їх реалізації в рамках комп'ютерного моделювання. Основним завданням було розв'язання оптимізаційної задачі типу «Суміші», де також була розроблена функція перевірки коректності введених даних, що дозволило підвищити точність програми і спростити процес інтерпретації отриманих результатів.

Для перевірки точності результатів було застосовано два різних підходи програмування. Результати, які отримано в кожному випадку, збігаються: $x_1 = 0$, $x_2 = 4$, $x_3 = 5$, а мінімальна вартість дорівнює 24 умовні одиниці. Це свідчить про правильність обчислень і дає змогу зробити висновок, що для оптимального результату потрібно використовувати 0 одиниць першого продукту, 4 одиниці другого та 5 одиниць третього продукту, що забезпечить мінімальну витрату у 24 умовні одиниці. Це може свідчити про те, що перший продукт є нерентабельним або невигідним у даному контексті, тоді як другий і третій продукти є основними для досягнення оптимальної витрати ресурсів. Також можна зазначити, що з точки зору виробництва, такий результат може вказувати на необхідність перегляду виробничих планів, зменшення або повне усунення виробництва певного продукту, а також оптимізацію використання інших ресурсів для зниження загальних витрат.

5. СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Foundation, P. S. *Python Language Reference* (2024). https://docs.python.org/3/reference/.
2. Eaton, J. W., Bateman, D., Hauberg, S. & Wehbring, R. *GNU Octave 4.0 Reference Manual: Free Your Numbers* (Free Software Foundation, 2015). ISBN: 978-9888381050. https://docs.octave.org/octave-4.0.0.pdf.
3. Ладогубець, Т. С., Фіногенов, О. Д. & Губський, А. М. *Чисельні методи оптимізації. Практикум [Електронний ресурс]* 2-ге вид., переробл. та допов. Навч. посіб. за освітньою програмою «Наука про дані та математичне моделювання» спец. 113 Прикладна математика. Електронні текстові дані (1 файл: 3,2 Мбайт). https://ela.kpi.ua/items/f9de754f-2a88-4fb7-85f5-bfa5cf0bcd2d (КПІ ім. Ігоря Сікорського, Київ, 2024).
4. Dunning, I., Mitchell, S. & O'Sullivan, M. *PuLP: A Linear Programming Toolkit for Python* https://optimization-online.org/2011/09/3178/. Accessed: 2024-10-18. 2011.

6. ДОДАТОК А (код програм)

Python

```python
import numpy as np
from pulp import LpProblem, LpMinimize, LpVariable, lpSum, LpStatus, value


nutrient_contributions = np.array([
    [3, 1, 1],  # Product 1
    [1, 2, 0],  # Product 2
    [1, 6, 0],  # Product 3
])
nutrient_requirements = np.array([9, 8, 12])
product_costs = np.array([5, 6, 0])


def validate_inputs(contributions, requirements, costs):
    if not all(isinstance(arr, np.ndarray) for arr in [contributions, requ
        raise TypeError("All inputs must be numpy arrays.")
    if contributions.ndim != 2 or requirements.ndim != 1 or costs.ndim !=
        raise ValueError("Ensure contributions is 2D and requirements/cost
    if contributions.shape[0] != requirements.size or contributions.shape[
        raise ValueError("Mismatch between input dimensions.")
    if any(arr.min() < 0 for arr in [requirements, costs, contributions.fl
        raise ValueError("All input values must be non-negative.")
    if not all(np.isfinite(arr).all() for arr in [contributions, requireme
        raise ValueError("All input values must be finite.")

validate_inputs(nutrient_contributions, nutrient_requirements, product_cos


```

```
problem = LpProblem("Nutrient_Optimization", LpMinimize)


product_quantities = [
    LpVariable(f"Product_{i+1}", lowBound=0, cat="Continuous")
    for i in range(len(product_costs))
]


problem += lpSum(product_costs[i] * product_quantities[i] for i in range(l


for idx, requirement in enumerate(nutrient_requirements):
    problem += (
        lpSum(nutrient_contributions[idx, i] * product_quantities[i] for i
        f"Nutrient_{idx+1}_Requirement"
    )


problem.solve()


if LpStatus[problem.status] == "Optimal":
    print("Optimal solution determined:")
    for var in product_quantities:
        print(f"Variable {var.name}: {var.varValue:.2f}")
    print(f"Objective Function Value (Total Cost): {value(problem.objectiv
else:
    print("Optimal solution not attained. Further analysis is required.")
```

Octave

```
nutrient_contributions = [
    3, 1, 1;  % Product 1
    1, 2, 0;  % Product 2
    1, 6, 0;  % Product 3
];
nutrient_requirements = [9; 8; 12];
product_costs = [5; 6; 0];

[num_nutrients, num_products] = size(nutrient_contributions);

c = product_costs;
A = nutrient_contributions;
b = nutrient_requirements;

lb = zeros(num_products, 1);
ub = [];

sense = 1;

ctype = repmat('L', num_nutrients, 1);

vartype = repmat('C', num_products, 1);

[x, fval, status] = glpk(c, A, b, lb, ub, ctype, vartype, sense);

if status == 0
    fprintf('Optimal solution determined:\n');
    for i = 1:num_products
        fprintf('Variable Product_%d: %.2f\n', i, x(i));
    end
```

```
    fprintf('Objective Function Value (Total Cost): %.2f\n', fval);
else
    fprintf('Optimal solution not attained. Further analysis is required.\
end
```