Operational Research: Linear Programming Exercises

Linear programming (eng. LP) is one of the most important fields of mathematical optimisation, which aims to maximise or minimise a linear function given certain constraints also in the form of linear equations or inequalities. Linear programming methods are used in many fields such as logistics, economics, engineering, production management, financial analysis or strategic planning.

Brief history

The origins of linear programming date back to the 1930s, but its rapid development began during the Second World War. In 1947, American mathematician George Dantzig developed the simplex method, which revolutionised the way optimisation problems are solved and remains one of the most popular methods for solving PL problems today.

Since then, linear programming has become an integral part of operations research (OR) and has gained wide recognition in business and scientific practice.

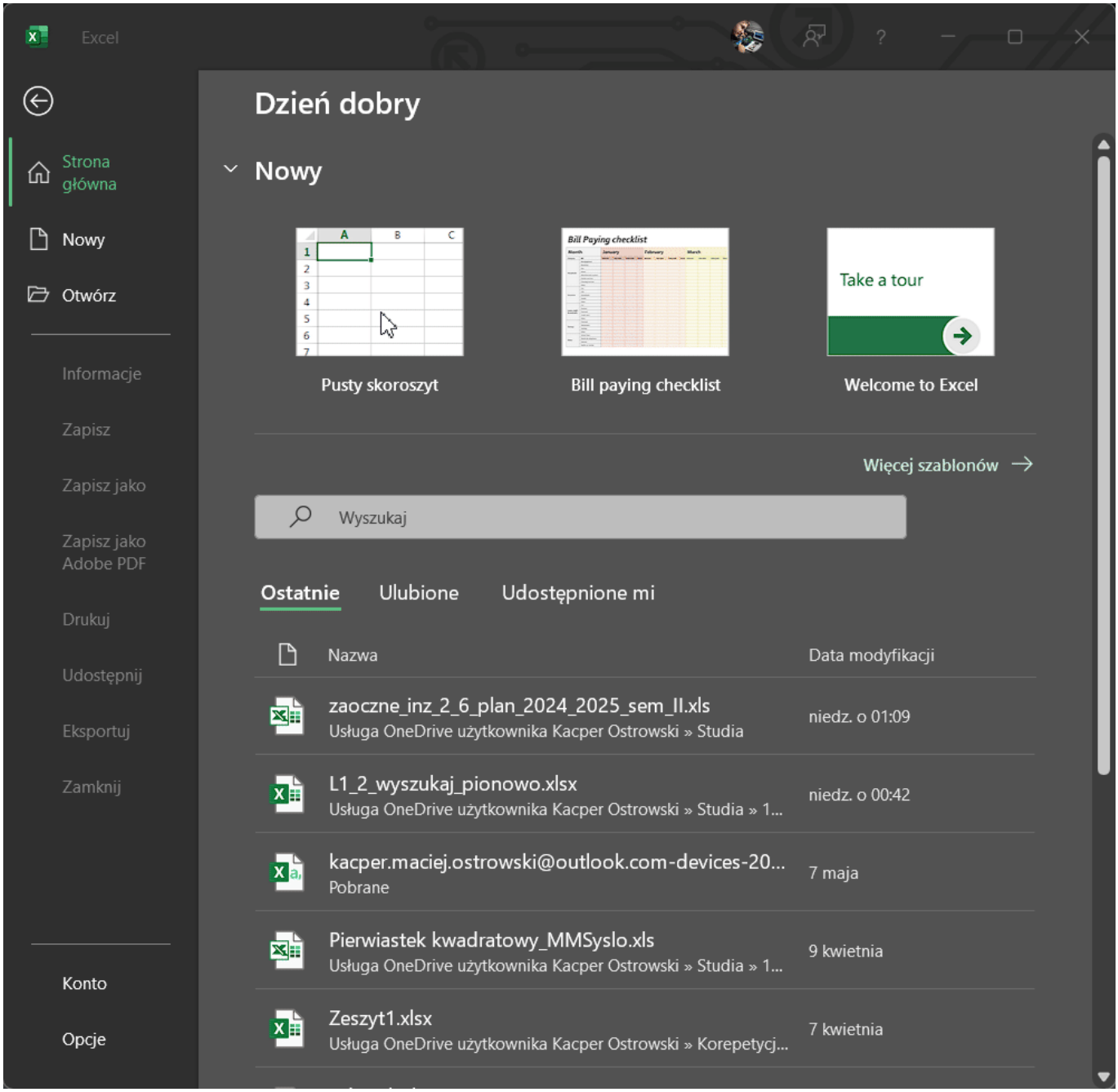
Basic concepts

The basic elements of linear programming are:

- Goal function - the linear function that we want to maximise or minimise (e.g. profit, cost, time).
- Decision variable - the quantities whose values are sought and which affect the value of the objective function.
- Constraints (restrictions) - conditions that must be met by the decision variables, usually in the form of linear equations or inequalities.
- Acceptable area - the set of all possible solutions that satisfy the constraints.
- Optimal solution - the point in the admissible area for which the objective function reaches the largest (maximum) or smallest (minimum) value, according to the problem assumption.

The following sections of this article will present methods for solving linear programming problems and examples of practical applications.

How to install Solver



Example of a production problem - "Lamps"

Consider a typical manufacturing problem in which the aim is to maximise profit with limited time and material resources.

A craftsman produces two types of lamps:

- **Standing lamps (large)** - require 6 hours of work and £200 for materials, profit per piece is £240.
- **Desk lamps (small)** - require 5 hours of work and £100 for materials, profit per unit is £160.

The craftsman has at his disposal weekly:

- **a maximum of 40 working hours**,
- **a maximum of £1,000 for materials**.

The task is to determine how many lamps of each type he should produce per week in order to **maximise its profit**without exceeding the available resources.

Decision variable

Let:

- x - the number of standing lamps produced per week,
- y - the number of desk lamps produced per week.

The objective function

We want to **maximise profit**:

$$ From = 240x + 160y $$

Limitations

1. **Working time**:

$$ 6x + 5y \n40 $$

2. **Cost of materials**:

$$ 200x + 100y \n-1000 $$

3. **Non-negativity of variables** (no negative number of tubes can be produced):

$$ x ¯geq 0,¯quad y ¯geq 0. $$

Mathematical model

$$ \Ùbegin{cases} \{text{ maximise } From = 240x + 160y \\ \} \\ 6x + 5y } 40 \\ 200x + 100y } 1000 \\ x \geq 0, y \geq 0. \$$end{cases} $$ In the following steps, this model can be solved using the graphical method (as there are only two variables) or the simplex method can be used. The result will give us the optimal number of lamps of each type that an artisan should produce to achieve maximum profit. ===== Solving the lamp problem - Excel and Python ===== ==== Solution in Excel (Solver add-in) ==== To solve a linear programming problem in Excel, you can use the built-in tool **Solver**: === Step by Step: === 1. Enter the data into the sheet: | ^ A ^ B ^ C ^ ^ 1 | | Standing lamps (x) | Desk lamps (y) | ^ 2 | Unit yield | 240 | 160 | ^ 3 | Operating time | 6 | 5 | ^ 4 | Material cost | 200 | 100 | ^ 5 | Quantity | =x | =y | 2. enter decision variables (e.g. cells ''B5'' i ''C5'') - these will be lamp numbers. 3. calculate the total gain: In cell ''D1'' enter the formula: '' =240*B5 + 160*C5'' 4 Calculate resource consumption: * Total operating time: `=6*B5 + 5*C5`. * Total cost of materials: `=200*B5 + 100*C5` 5 Open **Solver**: * `Data > Solver`. * Set "Maximise": cell with total profit. * Changed cells: `B5:C5`. Restrictions: * working time ≤ 40 * material cost ≤ 1000 * B5 ≥ 0, C5 ≥ 0 6. select **Simplex LP** as the solution method. 7. click **Solve**. ==== Solution in Python (`PuLP` library) ==== Python offers the `PuLP` library to create and solve linear programming

problems. === Example code: === <code python> from pulp import LpMaximize, LpProblem, LpVariable # Definiowanie problemu model = LpProblem(name="lamp-production", sense=LpMaximize) # Zmienne decyzyjne (rzeczywiste) x = LpVariable(name="lampy_stojace", lowBound=0) y = LpVariable(name="lampy_biurkowe", lowBound=0) # Funkcja celu model += 240 * x + 160 * y, "Zysk" # Ograniczenia model += (6 * x + 5 * y <= 40, "Czas_pracy") model += (200 * x + 100 * y <= 1000, "Koszt_materialow") # Rozwiązanie model.solve() # Wynik print(f"Lampy stojące: {x.value()}") print(f"Lampy biurkowe: {y.value()}") print(f"Maksymalny zysk: {model.objective.value()} zł") </code> === Result === <code> Lampy stojące: 2.5 Lampy biurkowe: 5.0 Maksymalny zysk: 1400.0 zł </code> ====== Example of a production problem - Tights ====== Consider a production optimisation problem in a plant producing two types of tights: thin and thick. The objective is to **maximise turnover**with limited raw material resources. ==== Content of the task ==== The plant produces: * **Thin tights** - require: 10 g elastic yarn and 10 g cotton yarn, * **Thick tights** - require: 20 g of elastic yarn and 25 g of cotton. Stock: * **200 kg of elastic yarn** = 200 000 g, * **500 kg cotton yarn** = 500 000 g. Selling prices: * Thin tights: **PLN 1.50 / pair**, * Thick tights: **4.00 zł / pair**. ==== Decision variable ==== Let: * x - the number of pairs of thin tights,

- y - the number of pairs of thick tights.

The objective function

The plant wants to **maximise the utlage** (i.e. total revenue):

$$ From = 1.5x + 4y $$

Limitations

1. **Consumption of elastic yarn**:

$$ 10x + 20y \n 200,000 $$

2. **Cotton yarn consumption**:

$$ 10x + 25y \leq 500,000 $$

3. **Non-negativity of variables**:

$$ x ¯geq 0,¯quad y ¯geq 0. $$

Mathematical model

$$ \Ùbegin{cases} \{text{maximise } From = 1.5x + 4y \\ \} \\ 10x + 20y \n200,000 \\ 10x + 25y \leq 500,000 \\ x \geq 0, y \geq 0. \$$$end{cases} $$ ==== Comment ==== This model can be solved in Excel (using the Solver add-on) or in Python, e.g. using the `PuLP` library. The result will be the number of pairs of thin and thick pantyhose that the factory should produce to maximise revenue without exceeding the available yarn stock. In the following sections, a concrete solution can be presented (using a graphical method, in Excel or Python) and an analysis of the result. ==== Solution in Python (`PuLP` library) ==== Python offers the `PuLP` library to create and solve linear

programming problems. === Example code: === <code python> from pulp import LpMaximize, LpProblem, LpVariable # Tworzymy problem maksymalizacji model = LpProblem(name="produkcja-rajstop", sense=LpMaximize) # Zmienne decyzyjne (liczba par rajstop) x = LpVariable(name="rajstopy_cienkie", lowBound=0) y = LpVariable(name="rajstopy_grube", lowBound=0) # Funkcja celu – maksymalizacja utargu model += 1.5 * x + 4 * y, "Utarg" # Ograniczenia zużycia przędzy (gramy) model += (10 * x + 20 * y <= 200_000, "Przędza_elastyczna") model += (10 * x + 25 * y <= 500_000, "Przędza_bawełniana") # Rozwiązanie model.solve() # Wyniki print("Plan produkcji maksymalizujący utarg:") print(f"Rajstopy cienkie: {x.value()} par") print(f"Rajstopy grube: {y.value()} par") print(f"Maksymalny utarg: {model.objective.value()} zł") </code> === Headline === <code> Plan produkcji maksymalizujący utarg: Rajstopy cienkie: 0.0 par Rajstopy grube: 10000.0 par Maksymalny utarg: 40000.0 zł </code> ====== Example of a production problem - "Lamps" ====== The production plant manufactures three types of fabric: * **Bed linen** * **Dress** * **Decorative** The production process for each fabric takes place in three departments: * **Spinning mill** * **Weaving mill** * **Finishing** For each type of fabric specified: * **Unit machine time consumption** (in minutes per 1 mb - running metre), * **Unit profit** (in PLN per 1 mb). Maximum **working time of machines** in each department is limited - these values are given in hours, which should be converted into minutes (1 hour = 60 minutes). ==== Input data ==== ^ Textiles ^ Spinning mill [min] ^ Weaving mill [min] ^ Finishing mill [min] ^ Unit profit [PLN] ^ | Bed linen | 2 | 1 | 2 | 5 | | Dress | 1 | 2 | 2 | 4.5 | | Decorative | 2 | 2 | 1 | 6 | ^ Maximum machine uptime (in hours) ^ 2400 ^ 3000 ^ 2600 ^ ^ ^ Converted into minutes ^ 144000 ^ 180000 ^ 156000 ^ ^ ==== Decision variable ==== Let: * x - number of running metres of fabrics **of bedding fabric**,

- y - number of running metres of fabrics **of dress fabric**,
- z - number of running metres of fabrics **decorative fabrics**.

Target function

The objective is to **profit maximisation**:

$$ Z = 5x + 4.5y + 6z $$

Time restrictions (converted into minutes)

1. **Yarn**:

$$ 2x + 1y + 2z \leq 144000 $$

2. **Weaving**:

$$ 1x + 2y + 2z \leq 180000 $$

3. **Finishing**:

$$ 2x + 2y + 1z \leq 156000 $$

4. **Non-negativity of variables**:

$$ x ¯geq 0,¯quad y ¯geq 0,¯quad z ¯geq 0. $$

Mathematical model

$$ \Ùbegin{cases} \{text{maximise } Z = 5x + 4.5y + 6z \\ \} \\ 2x + 1y + 2z } 144000 \\ 1x + 2y + 2z \leq 180000 \\ 2x + 2y + 1z \leq 156000 \\ x, y, z \gq 0 \³{cases} $$

Purpose of the task

Determine the optimum fabric production plan, i.e. the values x, y, z that maximise the plant's profit without exceeding the available machine time limits in each production department.

Solution in Python (`PuLP` library)

Python offers the `PuLP` library to create and solve linear programming problems.

Example code:

```
from pulp import LpMaximize, LpProblem, LpVariable

# Tworzymy model
model = LpProblem(name="produkcja-tkanin", sense=LpMaximize)

# Zmienne decyzyjne: ilość mb tkanin
x = LpVariable(name="tkanina_poscielowa", lowBound=0)
y = LpVariable(name="tkanina_sukienkowa", lowBound=0)
z = LpVariable(name="tkanina_dekoracyjna", lowBound=0)

# Funkcja celu: maksymalizacja zysku
model += 5 * x + 4.5 * y + 6 * z, "Zysk"

# Ograniczenia czasowe – przeliczone na minuty
model += 2 * x + 1 * y + 2 * z <= 144_000, "Przędzalnia"
model += 1 * x + 2 * y + 2 * z <= 180_000, "Tkalnia"
model += 2 * x + 2 * y + 1 * z <= 156_000, "Wykończalnia"

# Rozwiązanie
model.solve()

# Wyniki
print("Optymalny plan produkcji:")
print(f"Tkanina pościelowa:    {x.value():.2f} mb")
print(f"Tkanina sukienkowa:    {y.value():.2f} mb")
print(f"Tkanina dekoracyjna:   {z.value():.2f} mb")
print(f"Maksymalny zysk:       {model.objective.value():.2f} zł")
```

Result

```
Optymalny plan produkcji:
Tkanina pościelowa:    12000.00 mb
Tkanina sukienkowa:    48000.00 mb
Tkanina dekoracyjna:   36000.00 mb
Maksymalny zysk:       492000.00 zł
```

Example of a production problem - hats

A workshop specialising in the production of men's hats and berets has limited raw material resources and faces limited market demand. The objective is to develop a production plan that **maximises overall profit** in the autumn and winter months (Q1 and Q4).

Content of the task

- **Raw material**The basic material for both products is **felt**.
- Per **hat** is needed: **0,6 m²** felt.
- For one **beret** needed: **0,4 m²** felt.
- Total monthly **felt purchase limit**: **960 m²**.
- **Market demand**: no more than **1000 units of each product per month**.

Unit profit:

- Hat: **£3/unit.**
- Beret: **£2/piece.**

Decision variable

Let:

- x - number of hats produced per month,
- y - number of berets produced per month.

The objective function

The objective is **profit maximisation**:

$$ From = 3x + 2y $$

Limitations

1. **Felt consumption**:

$$ 0.6x + 0.4y \n 960 $$

2. **Demand (sales) constraints.**:

$$ x \n-1000 \\ y -leq 1000 $$

3. **Non-negativity of variables**:

$$ x ¯geq 0,¯quad y ¯geq 0. $$

Mathematical model

$$ \Ùbegin{cases} \{text{maximise } From = 3x + 2y \\ \{text{conditions} \\ 0.6x + 0.4y } 960 \\ x \leq 1000 \\ y \leq 1000 \\ x, y \gq 0 \} $$

Purpose of the task

Determine the number of hats and berets that the studio should produce during the autumn and winter months in order to make the **maximum profit**, taking into account material and demand constraints.

Solution in Python (`PuLP` library)

Python offers the `PuLP` library to create and solve linear programming problems.

Example code:

```
from pulp import LpMaximize, LpProblem, LpVariable

# Tworzymy model maksymalizacji zysku
model = LpProblem(name="produkcja-kapeluszy-i-beretow", sense=LpMaximize)

# Zmienne decyzyjne: liczba kapeluszy i beretów
x = LpVariable(name="kapelusze", lowBound=0)
y = LpVariable(name="berety", lowBound=0)

# Funkcja celu: maksymalizacja zysku
model += 3 * x + 2 * y, "Zysk"

# Ograniczenie zużycia filcu
model += 0.6 * x + 0.4 * y <= 960, "Filc"

# Ograniczenia popytowe
model += x <= 1000, "Max_kapelusze"
model += y <= 1000, "Max_berety"
```

```
# Rozwiązanie
model.solve()

# Wyniki
print("Optymalny plan produkcji:")
print(f"Kapelusze: {x.value():.0f} szt.")
print(f"Berety:    {y.value():.0f} szt.")
print(f"Maksymalny zysk: {model.objective.value():.2f} zł")
```

Result:

```
Optymalny plan produkcji:
Kapelusze: 1000 szt.
Berety:    900 szt.
Maksymalny zysk: 4800.00 zł
```

Example of a production problem - Ms Susie

Ms Susie runs a small catering business where she prepares two types of dishes: **Maxi** i **Mini**. The two dishes differ in portion size, ingredient consumption and selling price. The aim is to determine the number of dishes of each type in order to **maximise your daily revenue** from their sale - we assume that everything prepared will also be sold.

Content of the task

Dishes offered:

- **Maxi** - price: 19 zł
- **Mini** - price: 11 zł

Ingredients needed for preparation:

Meat:

- Maxi: 7 dag (0.07 kg)
- Mini: 4 dag (0.04 kg)

Cabbage:

- Maxi: 10 dag (0.10 kg)
- Mini: 7 dag (0.07 kg)

Daily supply:

- **Meat**: 1.3 kg
- **Cabbage**: 2.0 kg

Decision variable

Let:

- x - number of dishes of type **Maxi**,
- y - number of dishes of type **Mini**.

Objective function

The objective is to **to maximise revenue**:

$$ From = 19x + 11y $$

Resource constraints

1. **Meat**:

$$ 0.07x + 0.04y \n 1.3 $$

2. **Cabbage**:

$$ 0.10x + 0.07y \n2.0 $$

3. **Non-negativity of variables**:

$$ x ¯geq 0,¯quad y ¯geq 0. $$

Mathematical model

$$ \Ùbegin{cases} \text{ maximise } From = 19x + 11y \\ \{text{conditions} \\ 0.07x + 0.04y \leq 1.3 \\ 0.10x + 0.07y \leq 2.0 \\ x, y \gq 0 \■end{cases} $$

Purpose of the task

Determine how many dishes of the type **Maxi** and **Mini** Ms Susie should prepare daily in order to achieve **maximum possible income**with limited meat and cabbage resources.

Solution in Python (`PuLP` library)

Python offers the `PuLP` library to create and solve linear programming problems.

Example code:

```
from pulp import LpMaximize, LpProblem, LpVariable, LpInteger
```

```
# Tworzymy model optymalizacji
model = LpProblem(name="produkcja-dan-pani-zuzia", sense=LpMaximize)

# Zmienne decyzyjne jako liczby całkowite
x = LpVariable(name="danie_maxi", lowBound=0, cat=LpInteger)
y = LpVariable(name="danie_mini", lowBound=0, cat=LpInteger)

# Funkcja celu: maksymalizacja przychodu
model += 19 * x + 11 * y, "Przychod"

# Ograniczenia surowców
model += 0.07 * x + 0.04 * y <= 1.3, "Ograniczenie_miesa"
model += 0.10 * x + 0.07 * y <= 2.0, "Ograniczenie_kapusty"

# Rozwiązanie problemu
model.solve()

# Wyniki
print("Optymalny plan produkcji (liczby całkowite):")
print(f"Dania Maxi: {int(x.value())} szt.")
print(f"Dania Mini: {int(y.value())} szt.")
print(f"Maksymalny przychód: {model.objective.value():.2f} zł")
```

Result:

```
Optymalny plan produkcji (liczby całkowite):
Dania Maxi: 14 szt.
Dania Mini: 8 szt.
Maksymalny przychód: 354.00 zł
```

Minimisation example Space heating

Two heat sources can be used to heat two rooms with an initial temperature of 0°C: coal and coke. Each of these fuels affects the room temperature differently and their prices also vary.

Input data

Initial temperature in both rooms: 0°C Minimum temperatures required:

- Room 1: at least 180°C
- Room 2: at least 200°C

combustion effect: **1 kg of carbon**:

- Room 1: +30°C
- Room 2: +20°C

1 kg coke:

- Room 1: +10°C
- Room 2: +20°C

Costs:

- 1 tonne of coal: 500 zł → 1 kg = 0.50 zł
- 1 tonne of coke: 300 zł → 1 kg = 0.30 zł

Decision variables

- $ x $ - number **kg of coal** to be burned
- $ y $ - number **kg of coke** to be burned

Objective function

Minimise the total cost of heating: $$ \Zmin Z = 0.50x + 0.30y $$

Limitations

Temperatures must be reached or exceeded:

- For the first room:

$$ 30x + 10y \n 180 $$

- For the second room:

$$ 20x + 20y \geq 200 $$

- In addition, we cannot burn negative amounts of fuel:

$$ x \geq 0,\quad y \geq 0 $$

Objective

Determine the minimum quantities of coal $ x $ and coke $ y $ that will achieve the required temperature in both rooms at the **the lowest cost of heating**.

Solution

The solution can be obtained using:

- a spreadsheet (Excel: adding Solver),
- linear programming in Python (e.g. using the PuLP library),
- a graphical method (if we are looking for a visual understanding for 2 variables).

Solution in Python (`PuLP` library)

Python offers the `PuLP` library to create and solve linear programming problems.

Example code:

```
from pulp import LpProblem, LpVariable, LpMinimize, LpStatus, value

# Tworzenie modelu
model = LpProblem("ogrzewanie_pomieszczen", LpMinimize)

# Zmienne decyzyjne: kg węgla (x), kg koksu (y)
x = LpVariable("wegiel_kg", lowBound=0)
y = LpVariable("koks_kg", lowBound=0)

# Funkcja celu: minimalizacja kosztów
model += 0.50 * x + 0.30 * y, "Koszt_ogrzewania"

# Ograniczenia temperatury
model += 30 * x + 10 * y >= 180, "Temp_pomieszczenie_1"
model += 20 * x + 20 * y >= 200, "Temp_pomieszczenie_2"

# Rozwiązanie
model.solve()

# Wyniki
print("Status:", LpStatus[model.status])
print("Węgiel (kg):", round(x.value(), 2))
print("Koks (kg):", round(y.value(), 2))
print("Minimalny koszt (zł):", round(value(model.objective), 2))
```

Result:

```
Status: Optimal
Węgiel (kg): 4.0
Koks (kg): 6.0
Minimalny koszt (zł): 3.8
```

Transport problem - Transport of salt and sand mixture

The city has to deliver salt and sand mixture from two depots to four districts during the winter period. The aim is to establish a transport plan that **meets the needs of all districts** at **the lowest possible transport cost**.

Input data

- The city has **two depots** material.
- The material must be distributed to **four districts**.
- Each district has a specific **mix requirements (in tonnes) .**.
- Each depot has a limited **maximum amount of material it can deliver .**.
- Transport costs (in PLN/t) vary depending on the route (depot-district).

Table of transport costs and demands

District	Depot 1 (PLN/t)	Depot 2 (PLN/t)	Demand (t)
1	2.00	4.00	300
2	3.00	3.50	450
3	1.50	2.50	500
4	2.50	3.00	350

Storage capacities

- Depot 1: **900 tonnes**
- Depot 2: **750 tonnes**

Decision variables

Let: x_{ij} the number of tonnes of mixture transported from the depot i to the district j, where $i ¢in ¢{1, 2}$ and $j ¢in ¢{1, 2, 3, 4}$.

Objective function

Minimise the total cost of transport:

$$ From = ■min ■sum_{i=1}^{2} \sum_{j=1}^{4} c_{ij} \dot x_{ij} $$

Where c_{ij} is the cost of transporting 1 tonne from depot i to district j.

Restrictions

Capacity of depots:

$$ x_{11} + x_{12} + x_{13} + x_{14} \$$ 900
x_{21} + x_{22} + x_{23} + x_{24} \$$750 $$

Demand of districts:

$$ x_{11} + x_{21} = 300 \\ x_{12} + x_{22} = 450 \\ x_{13} + x_{23} = 500 \\ x_{14} + x_{24} = 350 $$

Non-negativity of variables:

$$ x_{ij} \$$geq 0 ■quad ■text{for each } i, j $$ ===== Objective ===== Determine the optimal values of x_{ij} so that the **the total cost of transport is as low as possible**, while satisfying demand and capacity constraints.

Notes

This is a classic example of the so-called. **transport problem** in linear programming. The solution can be found by means of, among others:

- the method of potentials (manually or in a spreadsheet),
- linear programming in Python (e.g. `PuLP`, `scipy.optimize`),
- tools such as Excel Solver.

Solution in Python (`PuLP` library)

Python offers the `PuLP` library to create and solve linear programming problems.

Example code:

```
import pulp

# Tworzymy problem minimalizacji kosztów
problem = pulp.LpProblem("Transport_soli_i_piasku", pulp.LpMinimize)

# Składnice i dzielnice
skladnice = [1, 2]
dzielnice = [1, 2, 3, 4]

# Koszty transportu c_ij (słownik)
koszty = {
    (1, 1): 2.00, (1, 2): 3.00, (1, 3): 1.50, (1, 4): 2.50,
    (2, 1): 4.00, (2, 2): 3.50, (2, 3): 2.50, (2, 4): 3.00
```

```
}

# Zapotrzebowania dzielnic
zapotrzebowanie = {1: 300, 2: 450, 3: 500, 4: 350}

# Pojemności składnic
pojemnosc = {1: 900, 2: 750}

# Zmienne decyzyjne x_ij: ilość materiału z i-tej składnicy do j-tej
dzielnicy
x = pulp.LpVariable.dicts("x", [(i, j) for i in skladnice for j in
dzielnice],
                          lowBound=0, cat='Continuous')

# Funkcja celu: minimalizacja kosztu transportu
problem += pulp.lpSum(koszty[i, j] * x[i, j] for i in skladnice for j in
dzielnice), "Koszt_calkowity"

# Ograniczenia pojemności składnic
for i in skladnice:
    problem += pulp.lpSum(x[i, j] for j in dzielnice) <= pojemnosc[i],
f"Pojemnosc_skladnicy_{i}"

# Ograniczenia zapotrzebowania dzielnic
for j in dzielnice:
    problem += pulp.lpSum(x[i, j] for i in skladnice) == zapotrzebowanie[j],
f"Zapotrzebowanie_dzielnicy_{j}"

# Rozwiązanie problemu
problem.solve()

# Wyniki
print("Status:", pulp.LpStatus[problem.status])
print("Minimalny koszt transportu:", pulp.value(problem.objective), "zł")

for i in skladnice:
    for j in dzielnice:
        print(f"x({i},{j}) = {x[i, j].varValue} t")
```

Result

```
Status: Optimal
Minimalny koszt transportu: 3925.0 zł
x(1,1) = 300.0 t
x(1,2) = 0.0 t
x(1,3) = 500.0 t
x(1,4) = 100.0 t
x(2,1) = 0.0 t
x(2,2) = 450.0 t
x(2,3) = 0.0 t
```

```
x(2,4) = 250.0 t
```

Code without dynamic generation of constraints and transports

```
import pulp

# Tworzymy problem minimalizacji kosztów
problem = pulp.LpProblem("Transport_soli_i_piasku", pulp.LpMinimize)

# Koszty transportu
koszty = {
    (1, 1): 2.00, (1, 2): 3.00, (1, 3): 1.50, (1, 4): 2.50,
    (2, 1): 4.00, (2, 2): 3.50, (2, 3): 2.50, (2, 4): 3.00
}

# Zapotrzebowanie i pojemności
zapotrzebowanie_1 = 300
zapotrzebowanie_2 = 450
zapotrzebowanie_3 = 500
zapotrzebowanie_4 = 350

pojemnosc_1 = 900
pojemnosc_2 = 750

# Zmienne decyzyjne
x_11 = pulp.LpVariable("x_11", lowBound=0, cat='Continuous')
x_12 = pulp.LpVariable("x_12", lowBound=0, cat='Continuous')
x_13 = pulp.LpVariable("x_13", lowBound=0, cat='Continuous')
x_14 = pulp.LpVariable("x_14", lowBound=0, cat='Continuous')
x_21 = pulp.LpVariable("x_21", lowBound=0, cat='Continuous')
x_22 = pulp.LpVariable("x_22", lowBound=0, cat='Continuous')
x_23 = pulp.LpVariable("x_23", lowBound=0, cat='Continuous')
x_24 = pulp.LpVariable("x_24", lowBound=0, cat='Continuous')

# Funkcja celu
problem += (
    2.00 * x_11 + 3.00 * x_12 + 1.50 * x_13 + 2.50 * x_14 +
    4.00 * x_21 + 3.50 * x_22 + 2.50 * x_23 + 3.00 * x_24
), "Koszt_calkowity"

# Ograniczenia pojemności składnic
problem += x_11 + x_12 + x_13 + x_14 <= pojemnosc_1, "Pojemnosc_skladnicy_1"
problem += x_21 + x_22 + x_23 + x_24 <= pojemnosc_2, "Pojemnosc_skladnicy_2"

# Ograniczenia zapotrzebowania dzielnic
problem += x_11 + x_21 == zapotrzebowanie_1, "Zapotrzebowanie_dzielnicy_1"
problem += x_12 + x_22 == zapotrzebowanie_2, "Zapotrzebowanie_dzielnicy_2"
problem += x_13 + x_23 == zapotrzebowanie_3, "Zapotrzebowanie_dzielnicy_3"
problem += x_14 + x_24 == zapotrzebowanie_4, "Zapotrzebowanie_dzielnicy_4"
```

```
# Rozwiązanie problemu
problem.solve()

# Wyniki
print("Status:", pulp.LpStatus[problem.status])
print("Minimalny koszt transportu:", pulp.value(problem.objective), "zł")

print("x(1,1) =", x_11.varValue, "t")
print("x(1,2) =", x_12.varValue, "t")
print("x(1,3) =", x_13.varValue, "t")
print("x(1,4) =", x_14.varValue, "t")
print("x(2,1) =", x_21.varValue, "t")
print("x(2,2) =", x_22.varValue, "t")
print("x(2,3) =", x_23.varValue, "t")
print("x(2,4) =", x_24.varValue, "t")
```

Transport of strawberries

During the summer season, the strawberries picked by the growers must be delivered to the collection points. Each grower has a specific quantity of strawberries (in tonnes) and each collection point has a specific demand. The aim is to plan transport in such a way as to **cover the demand of the collection centres with minimum transport costs**.

Input data

Availability of strawberries from growers:

- Grower I: 12 tonnes
- Grower II: 30 tonnes
- Grower III: 6 tonnes

Purchase point demand:

- Point A: 18 tonnes
- Point B: 12 tonnes
- Point C: 18 tonnes

Transport costs table (in PLN per 1 tonne)

Grower ↓ / Point →	A	B	C
I	80	160	160
II	240	320	80
III	32	160	32

Decision variables

Let:

- x_{ij} - the number of tonnes of strawberries transported from grower i to collection point j.

where $i ¢in ¢{1,2,3}$ (growers) and $j ¢in ¢{A,B,C}$ (collection points).

Objective function

Minimise the total cost of transport:

$$ From = \min \sum_{i=1}^{3} \sum_{j } c_{ij}} \cdot x_{ij} $$

Where c_{ij} is the cost of transporting 1 tonne from the grower i to the collection point j.

Limitations

Availability at growers: $$ x_{1A} + x_{1B} + x_{1C} \$$ 24
x_{2A} + x_{2B} + x_{2C} \Ùleq 30
x_{3A} + x_{3B} + x_{3C} \$$ 6 $$

Demand of collection points: $$ x_{1A} + x_{2A} + x_{3A} = 18 \\ x_{1B} + x_{2B} + x_{3B} = 12 \\ x_{1C} + x_{2C} + x_{3C} = 18 $$

Non-negativity of variables: $$ x_{ij} \$$geq 0 ■quad ■text{for each } i,j $$

Objective

Determine the optimal values of the variables x_{ij} so that:

- The total cost of transport is the **lowest**
- Availability and demand constraints are met

Comments

This is a classic **transport problem** solvable with:

- **Python + PuLP**
- **Excel Solver**
- **Northwest corner methods + potential method** (for manual tasks)

Code

```
from pulp import LpProblem, LpVariable, LpMinimize, LpStatus, lpSum, value

# Model
model = LpProblem("transport_truskawek", LpMinimize)

# Plantatorzy i punkty skupu
plantatorzy = ['P1', 'P2', 'P3']
punkty_skupu = ['A', 'B', 'C']

# Koszty transportu (zł/t)
koszty = {
    ('P1', 'A'): 80,   ('P1', 'B'): 160, ('P1', 'C'): 160,
    ('P2', 'A'): 240,  ('P2', 'B'): 320, ('P2', 'C'): 80,
    ('P3', 'A'): 32,   ('P3', 'B'): 160, ('P3', 'C'): 32
}

# Ilość dostępnych truskawek (t)
dostawy = {'P1': 12, 'P2': 30, 'P3': 6}

# Zapotrzebowanie punktów skupu (t)
zapotrzebowanie = {'A': 18, 'B': 12, 'C': 18}

# Zmienne decyzyjne
x = {
    (p, s): LpVariable(f"x_{p}_{s}", lowBound=0)
    for p in plantatorzy for s in punkty_skupu
}

# Funkcja celu: minimalizacja kosztów transportu
model += lpSum(koszty[p, s] * x[p, s] for p in plantatorzy for s in
punkty_skupu), "Koszt_calkowity"

# Ograniczenia dostępności plantatorów
for p in plantatorzy:
    model += lpSum(x[p, s] for s in punkty_skupu) <= dostawy[p],
f"Dostawa_{p}"

# Ograniczenia zapotrzebowania punktów skupu
for s in punkty_skupu:
    model += lpSum(x[p, s] for p in plantatorzy) == zapotrzebowanie[s],
f"Zapotrzebowanie_{s}"

# Rozwiązanie
model.solve()

# Wyniki
print("Status:", LpStatus[model.status])
print("Minimalny koszt transportu (zł):", round(value(model.objective), 2))
```

```
print("\nWielkości dostaw (t):")
for p in plantatorzy:
    for s in punkty_skupu:
        print(f"{p} -> {s}: {round(x[p, s].value(), 2)} t")
```

Result:

```
Status: Optimal
Minimalny koszt transportu (zł): 6432.0

Wielkości dostaw (t):
P1 -> A: 0.0 t
P1 -> B: 12.0 t
P1 -> C: 0.0 t
P2 -> A: 12.0 t
P2 -> B: 0.0 t
P2 -> C: 18.0 t
P3 -> A: 6.0 t
P3 -> B: 0.0 t
P3 -> C: 0.0 t
```