Badania operacyjne: Programowanie Liniowe ćwiczenia

Programowanie liniowe (eng. LP) to jedna z najważniejszych dziedzin optymalizacji matematycznej, której celem jest maksymalizacja lub minimalizacja funkcji liniowej przy spełnieniu określonych ograniczeń również w postaci równań lub nierówności liniowych. Metody programowania liniowego znajdują zastosowanie w wielu dziedzinach, takich jak logistyka, ekonomia, inżynieria, zarządzanie produkcją, analiza finansowa czy planowanie strategiczne.

Krótka historia

Początki programowania liniowego sięgają lat 30. XX wieku, jednak jego dynamiczny rozwój rozpoczął się w czasie II wojny światowej. W 1947 roku amerykański matematyk George Dantzig opracował metodę simpleks, która zrewolucjonizowała sposób rozwiązywania problemów optymalizacyjnych i do dziś pozostaje jedną z najpopularniejszych metod rozwiązywania problemów PL.

Od tamtej pory programowanie liniowe stało się integralną częścią badań operacyjnych (ang. operations research) i zyskało szerokie uznanie w praktyce gospodarczej i naukowej.

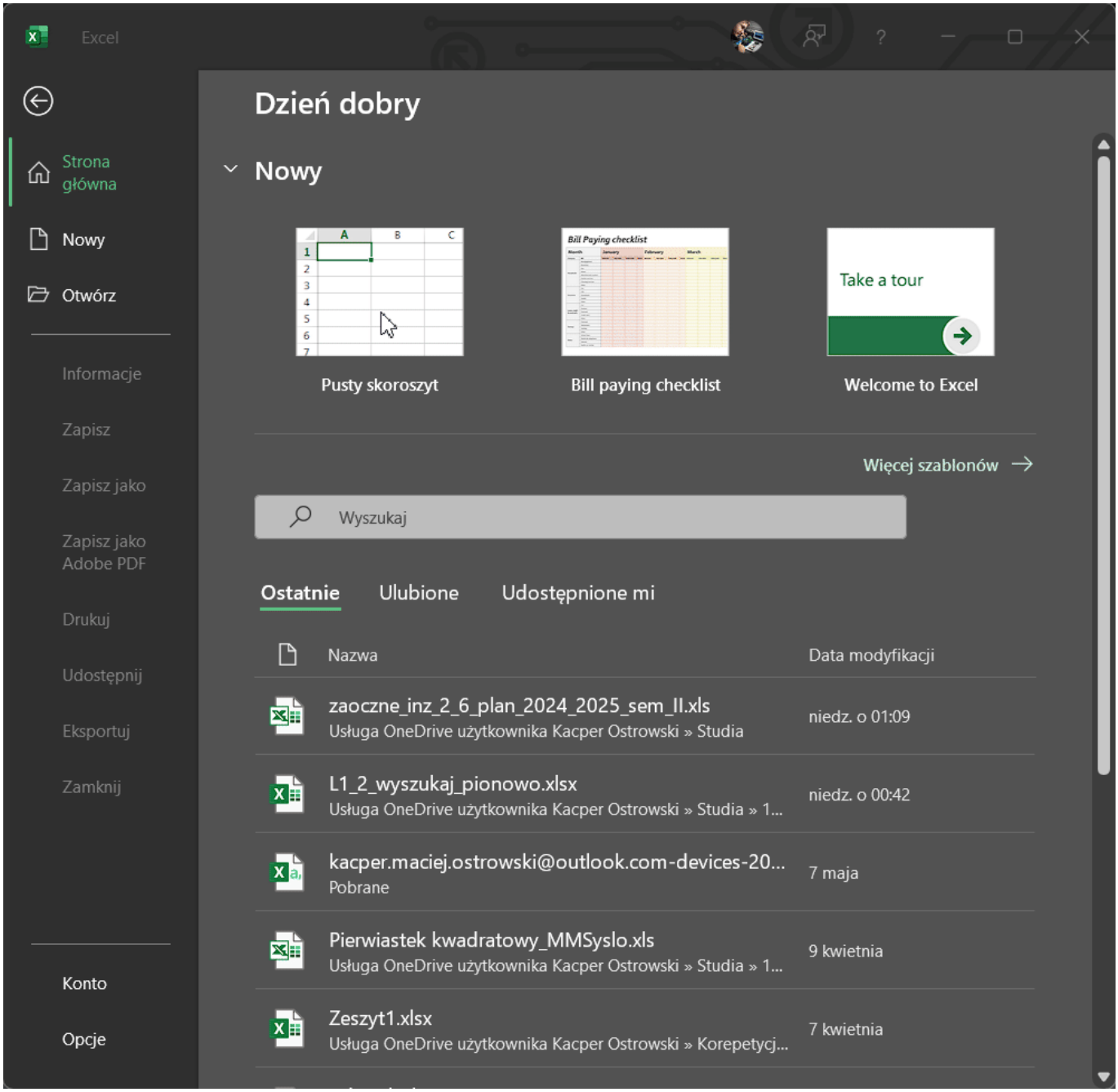
Podstawowe pojęcia

Podstawowe elementy programowania liniowego to:

- Funkcja celu – funkcja liniowa, którą chcemy maksymalizować lub minimalizować (np. zysk, koszt, czas).
- Zmienna decyzyjna – wielkości, których wartości są poszukiwane i które mają wpływ na wartość funkcji celu.
- Ograniczenia (restrykcje) – warunki, jakie muszą być spełnione przez zmienne decyzyjne, zwykle w formie równań lub nierówności liniowych.
- Obszar dopuszczalny – zbiór wszystkich możliwych rozwiązań, które spełniają ograniczenia.
- Rozwiązanie optymalne – punkt w obszarze dopuszczalnym, dla którego funkcja celu osiąga wartość największą (maksimum) lub najmniejszą (minimum), zgodnie z założeniem problemu.

W kolejnych sekcjach tego artykułu przedstawione zostaną metody rozwiązywania problemów programowania liniowego oraz przykłady praktycznych zastosowań.

Jak zainstalować Solver



Przykład problemu produkcyjnego – "Lampy"

Rozważmy typowy problem produkcyjny, w którym celem jest maksymalizacja zysku przy ograniczonych zasobach czasu i materiałów.

Rzemieślnik produkuje dwa rodzaje lamp:

- **Lampy stojące (duże)** – wymagają 6 godzin pracy i 200 zł na materiały, zysk z jednej sztuki wynosi 240 zł.
- **Lampy biurkowe (małe)** – wymagają 5 godzin pracy i 100 zł na materiały, zysk z jednej sztuki wynosi 160 zł.

Rzemieślnik dysponuje tygodniowo:

- **maksymalnie 40 godzinami pracy**,

- **maksymalnie 1000 zł na materiały**.

Zadanie polega na określeniu, ile lamp każdego typu powinien produkować tygodniowo, aby **zmaksymalizować swój zysk**, nie przekraczając dostępnych zasobów.

Zmienna decyzyjna

Niech:

- x – liczba lamp stojących produkowanych tygodniowo,
- y – liczba lamp biurkowych produkowanych tygodniowo.

Funkcja celu

Chcemy **zmaksymalizować zysk**:

$$ Z = 240x + 160y $$

Ograniczenia

1. **Czas pracy**:

$$ 6x + 5y \leq 40 $$

2. **Koszt materiałów**:

$$ 200x + 100y \leq 1000 $$

3. **Nieujemność zmiennych** (nie można wyprodukować ujemnej liczby lamp):

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

Model matematyczny

$$ \begin{cases} \text{Maksymalizuj } Z = 240x + 160y \\ \text{przy warunkach:} \\ 6x + 5y \leq 40 \\ 200x + 100y \leq 1000 \\ x \geq 0,\ y \geq 0 \end{cases} $$

W kolejnych krokach można ten model rozwiązać metodą graficzną (ponieważ są tylko dwie zmienne) lub zastosować metodę simpleks. Wynik da nam optymalną liczbę lamp każdego typu, które powinien produkować rzemieślnik, by osiągnąć maksymalny zysk.

Rozwiązanie problemu lamp – Excel i Python

Rozwiązanie w Excelu (Dodatek Solver)

Aby rozwiązać problem programowania liniowego w Excelu, można skorzystać z wbudowanego narzędzia **Solver**:

Krok po kroku:

1. Wprowadź dane do arkusza:

	A	B	C
1		Lampy stojące (x)	Lampy biurkowe (y)
2	Zysk jednostkowy	240	160
3	Czas pracy	6	5
4	Koszt materiału	200	100
5	Ilość	=x	=y

2. Wprowadź zmienne decyzji (np. komórki B5 i C5) – to będą liczby lamp.

3. Oblicz łączny zysk: W komórce D1 wpisz formułę: `=240*B5 + 160*C5`

4. Oblicz zużycie zasobów:

- Łączny czas pracy: `=6*B5 + 5*C5`
- Łączny koszt materiałów: `=200*B5 + 100*C5`

5. Otwórz **Solver**:

- `Dane > Solver`
- Ustaw „Maksymalizuj”: komórka z całkowitym zyskiem
- Zmieniane komórki: `B5:C5`

Ograniczenia:

- czas pracy ≤ 40
- koszt materiałów ≤ 1000
- B5 ≥ 0, C5 ≥ 0

6. Wybierz **Simplex LP** jako metodę rozwiązywania.

7. Kliknij **Rozwiąż**.

Rozwiązanie w Pythonie (biblioteka `PuLP`)

Python oferuje bibliotekę `PuLP`, która umożliwia tworzenie i rozwiązywanie problemów programowania liniowego.

Przykładowy kod:

```
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")
```

Wynik

```
Lampy stojące: 2.5
Lampy biurkowe: 5.0
Maksymalny zysk: 1400.0 zł
```

Przykład problemu produkcyjnego – Rajstopy

Rozważmy problem optymalizacji produkcji w zakładzie wytwarzającym dwa rodzaje rajstop: cienkie i grube. Celem jest **maksymalizacja utargu**, przy ograniczonych zasobach surowców.

Treść zadania

Zakład produkuje:

- **Rajstopy cienkie** – wymagają: 10 g przędzy elastycznej i 10 g bawełnianej,
- **Rajstopy grube** – wymagają: 20 g przędzy elastycznej i 25 g bawełnianej.

Stan magazynowy:

- **200 kg przędzy elastycznej** = 200 000 g,
- **500 kg przędzy bawełnianej** = 500 000 g.

Ceny sprzedaży:

- Rajstopy cienkie: **1,50 zł / para**,
- Rajstopy grube: **4,00 zł / para**.

Zmienna decyzyjna

Niech:

- x – liczba par rajstop cienkich,
- y – liczba par rajstop grubych.

Funkcja celu

Zakład chce **zmaksymalizować utarg** (czyli całkowity przychód):

$$ Z = 1.5x + 4y $$

Ograniczenia

1. **Zużycie przędzy elastycznej**:

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

2. **Zużycie przędzy bawełnianej**:

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

3. **Nieujemność zmiennych**:

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

Model matematyczny

$$ \begin{cases} \text{Maksymalizuj } Z = 1.5x + 4y \\ \text{przy warunkach:} \\ 10x + 20y \leq 200\,000 \\ 10x + 25y \leq 500\,000 \\ x \geq 0,\ y \geq 0 \end{cases} $$

Komentarz

Model ten można rozwiązać w Excelu (przy pomocy dodatku Solver) lub w Pythonie, np. za pomocą biblioteki `PuLP`. Wynikiem będzie liczba par rajstop cienkich i grubych, które zakład powinien wyprodukować, aby osiągnąć jak największy przychód, nie przekraczając dostępnych zapasów przędzy.

W kolejnych sekcjach można przedstawić konkretne rozwiązanie (metodą graficzną, w Excelu lub Pythonie) i analizę wyniku.

Rozwiązanie w Pythonie (biblioteka `PuLP`)

Python oferuje bibliotekę `PuLP`, która umożliwia tworzenie i rozwiązywanie problemów programowania liniowego.

Przykładowy kod:

```
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")
```

Nagłówek

```
Plan produkcji maksymalizujący utarg:
Rajstopy cienkie: 0.0 par
Rajstopy grube: 10000.0 par
Maksymalny utarg: 40000.0 zł
```

Przykład problemu produkcyjnego – "Lampy"

Zakład produkcyjny wytwarza trzy rodzaje tkanin:

- **Pościelowe**
- **Sukienkowe**
- **Dekoracyjne**

Proces produkcji każdej tkaniny przebiega w trzech wydziałach:

- **Przędzalnia**
- **Tkalnia**
- **Wykończalnia**

Dla każdego rodzaju tkaniny określono:

- **Jednostkowe zużycie czasu pracy maszyn** (w minutach na 1 mb – metr bieżący),
- **Jednostkowy zysk** (w złotówkach na 1 mb).

Maksymalny **czas pracy maszyn** w każdym z wydziałów jest ograniczony – wartości te podano w godzinach, co należy przeliczyć na minuty (1 godzina = 60 minut).

Dane wejściowe

Tkaniny	**Przędzalnia [min]**	**Tkalnia [min]**	**Wykończalnia [min]**	**Zysk jednostkowy [zł]**
Pościelowe	2	1	2	5
Sukienkowe	1	2	2	4.5
Dekoracyjne	2	2	1	6
Maksymalny czas pracy maszyn (w godz.)	**2400**	**3000**	**2600**	
Po przeliczeniu na minuty	**144000**	**180000**	**156000**	

Zmienna decyzyjna

Niech:

- x – liczba metrów bieżących tkanin **pościelowych**,
- y – liczba metrów bieżących tkanin **sukienkowych**,
- z – liczba metrów bieżących tkanin **dekoracyjnych**.

Funkcja celu

Celem jest **maksymalizacja zysku**:

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

Ograniczenia czasowe (przeliczone na minuty)

1. **Przędzalnia**:

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

2. **Tkalnia**:

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

3. **Wykończalnia**:

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

4. **Nieujemność zmiennych**:

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

Model matematyczny

$$ \begin{cases} \text{Maksymalizuj } Z = 5x + 4.5y + 6z \\ \text{przy warunkach:} \\ 2x + 1y + 2z \leq 144000 \\ 1x + 2y + 2z \leq 180000 \\ 2x + 2y + 1z \leq 156000 \\ x, y, z \geq 0 \end{cases} $$

Cel zadania

Wyznaczyć optymalny plan produkcji tkanin, czyli wartości x, y, z, które maksymalizują zysk zakładu, nie przekraczając dostępnych limitów czasu pracy maszyn w każdym dziale produkcyjnym.

Rozwiązanie w Pythonie (biblioteka `PuLP`)

Python oferuje bibliotekę `PuLP`, która umożliwia tworzenie i rozwiązywanie problemów programowania liniowego.

Przykładowy kod:

```
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")
```

Wynik

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

Przykład problemu produkcyjnego – kapelusze

Pracownia specjalizująca się w produkcji męskich kapeluszy i beretów dysponuje ograniczonymi zasobami surowca oraz stoi wobec ograniczonego popytu rynkowego. Celem jest opracowanie takiego planu produkcji, który **zmaksymalizuje całkowity zysk** w miesiącach jesienno-zimowych (I i IV kwartał).

Treść zadania

- **Surowiec**: podstawowym materiałem do produkcji obu wyrobów jest **filc**.
- Na jeden **kapelusz** potrzeba: **0,6 m²** filcu.
- Na jeden **beret** potrzeba: **0,4 m²** filcu.
- Łączny miesięczny **limit zakupu filcu**: **960 m²**.
- **Popyt rynkowy**: nie więcej niż **1000 sztuk każdego produktu miesięcznie**.

Zysk jednostkowy:

- Kapelusz: **3 zł/szt.**
- Beret: **2 zł/szt.**

Zmienna decyzyjna

Niech:

- x – liczba produkowanych kapeluszy miesięcznie,
- y – liczba produkowanych beretów miesięcznie.

Funkcja celu

Celem jest **maksymalizacja zysku**:

$$ Z = 3x + 2y $$

Ograniczenia

1. **Zużycie filcu**:

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

2. **Ograniczenia popytowe (sprzedażowe)**:

$$ x \leq 1000 \\ y \leq 1000 $$

3. **Nieujemność zmiennych**:

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

Model matematyczny

$$ \begin{cases} \text{Maksymalizuj } Z = 3x + 2y \\ \text{przy warunkach:} \\ 0.6x + 0.4y \leq 960 \\ x \leq 1000 \\ y \leq 1000 \\ x, y \geq 0 \end{cases} $$

Cel zadania

Wyznaczyć liczbę kapeluszy i beretów, jaką pracownia powinna produkować w miesiącach jesienno-zimowych, aby osiągnąć **maksymalny zysk**, przy uwzględnieniu ograniczeń materiałowych oraz popytowych.

Rozwiązanie w Pythonie (biblioteka `PuLP`)

Python oferuje bibliotekę `PuLP`, która umożliwia tworzenie i rozwiązywanie problemów programowania liniowego.

Przykładowy kod:

```
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")
```

Wynik:

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

Przykład problemu produkcyjnego – Pani Zuzia

Pani Zuzia prowadzi małą firmę gastronomiczną, w której przygotowuje dwa rodzaje dań: **Maxi** i **Mini**. Oba dania różnią się wielkością porcji, zużyciem składników oraz ceną sprzedaży. Celem jest ustalenie takiej liczby dań każdego typu, aby **zmaksymalizować dzienny przychód** z ich sprzedaży – zakładamy, że wszystko, co zostanie przygotowane, zostanie również sprzedane.

Treść zadania

Dania oferowane:

- **Maxi** – cena: 19 zł
- **Mini** – cena: 11 zł

Składniki potrzebne do przygotowania:

Mięso:

- Maxi: 7 dag (0,07 kg)
- Mini: 4 dag (0,04 kg)

Kapusta:

- Maxi: 10 dag (0,10 kg)
- Mini: 7 dag (0,07 kg)

Dzienne zapasy:

- **Mięso**: 1,3 kg
- **Kapusta**: 2,0 kg

Zmienna decyzyjna

Niech:

- x – liczba dań typu **Maxi**,
- y – liczba dań typu **Mini**.

Funkcja celu

Celem jest **maksymalizacja przychodu**:

$$ Z = 19x + 11y $$

Ograniczenia zasobów

1. **Mięso**:

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

2. **Kapusta**:

$$ 0.10x + 0.07y \leq 2.0 $$

3. **Nieujemność zmiennych**:

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

Model matematyczny

$$ \begin{cases} \text{Maksymalizuj } Z = 19x + 11y \\ \text{przy warunkach:} \\ 0.07x + 0.04y \leq 1.3 \\ 0.10x + 0.07y \leq 2.0 \\ x, y \geq 0 \end{cases} $$

Cel zadania

Wyznaczyć ile dań typu **Maxi** oraz **Mini** powinna przygotowywać Pani Zuzia dziennie, aby osiągnąć **maksymalny możliwy przychód**, przy ograniczonych zasobach mięsa i kapusty.

Rozwiązanie w Pythonie (biblioteka `PuLP`)

Python oferuje bibliotekę `PuLP`, która umożliwia tworzenie i rozwiązywanie problemów programowania liniowego.

Przykładowy kod:

```
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")
```

Wynik:

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

Przykład Minimalizacyjny Ogrzewanie pomieszczeń

Do ogrzania dwóch pomieszczeń o początkowej temperaturze 0°C można używać dwóch źródeł ciepła: węgla i koksu. Każde z tych paliw wpływa inaczej na temperaturę w pomieszczeniach, a ich ceny również się różnią.

Dane wejściowe

Początkowa temperatura w obu pomieszczeniach: 0°C Minimalne wymagane temperatury:

- Pomieszczenie 1: co najmniej 180°C
- Pomieszczenie 2: co najmniej 200°C

fekt spalania: **1 kg węgla**:

- Pomieszczenie 1: +30°C
- Pomieszczenie 2: +20°C

1 kg koksu:

- Pomieszczenie 1: +10°C
- Pomieszczenie 2: +20°C

Koszty:

- 1 tona węgla: 500 zł → 1 kg = 0,50 zł
- 1 tona koksu: 300 zł → 1 kg = 0,30 zł

Zmienne decyzyjne

- $ x $ – liczba **kg węgla** do spalenia
- $ y $ – liczba **kg koksu** do spalenia

Funkcja celu

Minimalizacja łącznego kosztu ogrzewania: $$ \min Z = 0.50x + 0.30y $$

Ograniczenia

Temperatury muszą być osiągnięte lub przekroczone:

- Dla pierwszego pomieszczenia:

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

- Dla drugiego pomieszczenia:

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

- Dodatkowo, nie możemy spalać ujemnych ilości paliwa:

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

Cel

Wyznaczyć minimalne ilości węgla $ x $ i koksu $ y $, które pozwolą uzyskać wymaganą temperaturę w obu pomieszczeniach przy **najniższym koszcie ogrzewania**.

Rozwiązanie

Rozwiązanie można uzyskać za pomocą:

- arkusza kalkulacyjnego (Excel: dodając Solver),
- programowania liniowego w Pythonie (np. z użyciem biblioteki PuLP),
- metody graficznej (jeśli szukamy wizualnego zrozumienia dla 2 zmiennych).

Rozwiązanie w Pythonie (biblioteka `PuLP`)

Python oferuje bibliotekę `PuLP`, która umożliwia tworzenie i rozwiązywanie problemów programowania liniowego.

Przykładowy kod:

```
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))
```

Wynik:

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

Problem transportowy - Transport mieszanki soli i piasku

Miasto musi w okresie zimowym dostarczyć mieszankę soli i piasku z dwóch składnic do czterech dzielnic. Celem jest ustalenie takiego planu transportu, który **zaspokoi zapotrzebowanie wszystkich dzielnic** przy **najniższym możliwym koszcie transportu**.

Dane wejściowe

- Miasto dysponuje **dwiema składnicami** materiału.
- Materiał musi być rozwieziony do **czterech dzielnic**.
- Każda dzielnica ma określone **zapotrzebowanie na mieszankę (w tonach)**.
- Każda składnica ma ograniczoną **maksymalną ilość materiału, jaką może dostarczyć**.
- Koszty transportu (w zł/t) różnią się w zależności od trasy (składnica–dzielnica).

Tabela kosztów transportu i zapotrzebowań

Dzielnica	Składnica 1 (zł/t)	Składnica 2 (zł/t)	Zapotrzebowanie (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

Pojemności składnic

- Składnica 1: **900 ton**
- Składnica 2: **750 ton**

Zmienne decyzyjne

Niech: x_{ij} – ilość ton mieszanki przetransportowanej ze składnicy i do dzielnicy j, gdzie $i \in \{1, 2\}$ i $j \in \{1, 2, 3, 4\}$.

Funkcja celu

Minimalizacja całkowitego kosztu transportu:

$$ Z = \min \sum_{i=1}^{2} \sum_{j=1}^{4} c_{ij} \cdot x_{ij} $$

Gdzie c_{ij} to koszt transportu 1 tony ze składnicy i do dzielnicy j.

Ograniczenia

Pojemności składnic:

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

Zapotrzebowanie dzielnic:

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

Nieujemność zmiennych:

$$ x_{ij} \geq 0 \quad \text{dla każdego } i, j $$

Cel

Wyznaczyć optymalne wartości x_{ij}, aby **całkowity koszt transportu był jak najmniejszy**, a

jednocześnie spełnione zostały ograniczenia zapotrzebowania i pojemności.

Uwagi

To klasyczny przykład tzw. **problemu transportowego** w programowaniu liniowym. Rozwiązanie można znaleźć m.in. za pomocą:

- metody potencjałów (ręcznie lub w arkuszu kalkulacyjnym),
- programowania liniowego w Pythonie (np. `PuLP`, `scipy.optimize`),
- narzędzi takich jak Excel Solver.

Rozwiązanie w Pythonie (biblioteka `PuLP`)

Python oferuje bibliotekę `PuLP`, która umożliwia tworzenie i rozwiązywanie problemów programowania liniowego.

Przykładowy kod:

```
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")
```

Wynik

```
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
```

Kod bez dynamicznego generowania ograniczeń i transportów

```
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 truskawek

W sezonie letnim truskawki zbierane przez plantatorów muszą być dostarczone do punktów skupu. Każdy z plantatorów dysponuje określoną ilością truskawek (w tonach), a każdy punkt skupu ma sprecyzowane zapotrzebowanie. Celem jest tak zaplanować transport, aby **pokryć zapotrzebowanie punktów skupu przy minimalnych kosztach transportu**.

Dane wejściowe

Dostępność truskawek u plantatorów:

- Plantator I: 12 tony
- Plantator II: 30 ton
- Plantator III: 6 ton

Zapotrzebowanie punktów skupu:

- Punkt A: 18 ton
- Punkt B: 12 ton
- Punkt C: 18 ton

Tabela kosztów transportu (w zł za 1 tonę)

Plantator ↓ / Punkt →	A	B	C
I	80	160	160
II	240	320	80
III	32	160	32

Zmienne decyzyjne

Niech:

- x_{ij} – ilość ton truskawek transportowana od plantatora i do punktu skupu j

gdzie $i \in \{1,2,3\}$ (plantatorzy) i $j \in \{A,B,C\}$ (punkty skupu)

Funkcja celu

Minimalizacja całkowitego kosztu transportu:

$$ Z = \min \sum_{i=1}^{3} \sum_{j \in \{A,B,C\}} c_{ij} \cdot x_{ij} $$

Gdzie c_{ij} to koszt transportu 1 tony z plantatora i do punktu skupu j.

Ograniczenia

Dostępność u plantatorów: $$ x_{1A} + x_{1B} + x_{1C} \leq 24 \\ x_{2A} + x_{2B} + x_{2C} \leq 30 \\ x_{3A} + x_{3B} + x_{3C} \leq 6 $$

Zapotrzebowanie punktów skupu: $$ x_{1A} + x_{2A} + x_{3A} = 18 \\ x_{1B} + x_{2B} + x_{3B} = 12 \\ x_{1C} + x_{2C} + x_{3C} = 18 $$

Nieujemność zmiennych: $$ x_{ij} \geq 0 \quad \text{dla każdego } i,j $$

Cel

Wyznaczyć optymalne wartości zmiennych x_{ij}, aby:

- Całkowity koszt transportu był **najniższy**
- Spełnione zostały ograniczenia dostępności i zapotrzebowania

Uwagi

To klasyczny **problem transportowy** możliwy do rozwiązania z użyciem:

- **Python + PuLP**
- **Excel Solver**
- **Metody północno-zachodniego narożnika + metoda potencjałów** (dla zadań ręcznych)

Kod

```
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")
```

Wynik:

```
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
```