1DE2116:A - Matematyka - Metody optymalizacji EP
Programowanie liniowe (LP)

Maciej Twardy

Materiał przygotowany w ramach projektu „NERW 2 PW. Nauka – Edukacja – Rozwój - Współpraca". Projekt współfinansowany przez Unie Europejska w ramach Europejskiego Funduszu Społeczego. Program Operacyjny Wiedza Edukacja Rozwój 2014-2020, Os priorytetowa III Szkolnictwo Wyzsze dla gospodarki i rozwoju, Działanie 3.5 Kompleksowe programy szkół wyższych.

1 Wstęp

Programowanie liniowe (ang. *linear programming*, w skrócie LP) stanowi niezwykle ważny obszar optymalizacji matematycznej zarówno z teoretycznego jak i praktycznego punktu widzenia [1, 2]. LP jest również ważnym zagadnieniem algorytmicznym [2]. Poniższe ćwiczenia mają na calu wprowadzenie do zadań LP, stąd niektóre z nich mogą się wydawać nieco akademickie lub wręcz „szkolne". Nie należy jednak z tego wyciągać mylnych wniosków, jakoby sam problem miałby być czysto akademicki. Przeciwnie, ilość zastosowań optymalizacji liniowej jest ogromna i uwaga ta dotyczy również bardzo praktycznych problemów. Przykład praktycznej aplikacji zadań LP można znaleźć np. w [3], dotyczy on planowania pracy systemu elektroenergetycznego. Jest to oczywiście przysłowiowa kropla w morzu, która jednak, jak mam nadzieję, pomoże przekonać Państwa, że chcąc zrozumieć główne idee i założenia metody (i nie ugrzęznąć w technicznych szczegółach), czasami lepiej posłużyć się przykładami „szkolnymi".

Przykład 1. Rozwiązać zadanie programowania liniowego

$$\begin{aligned} \text{minimize} \quad & x_1 + \frac{1}{2}x_2 \\ \text{subject to} \quad & x_1 + x_2 \leqslant 2 \\ & x_1 + \frac{1}{4}x_2 \leqslant 1 \\ & x_1 - x_2 \leqslant 2 \\ & \frac{1}{4}x_1 + x_2 \geqslant -1 \\ & x_1 + x_2 \geqslant 1 \\ & -x_1 + x_2 \leqslant 2 \\ & x_1 + \frac{1}{4}x_2 = \frac{1}{2} \\ & -1 \leqslant x_1 \leqslant \frac{3}{2} \\ & -\frac{1}{2} \leqslant x_2 \leqslant \frac{5}{4} \end{aligned} \tag{1}$$

Zadanie (1) jest stosunkowo proste i można znaleźć rozwiązanie „ręcznie", $x_1^\star = 1/3$, $x_2^\star = 2/3$, jednak w ogólnym przypadku korzysta się z odpowiedniego oprogramowania. Omówimy krótko dwa podejścia, jedno związane ze środowiskiem Matlab, drugie z językiem Python. Zanim przejdziemy do konkretnych rozwiązań, zauważmy, że zadanie (1) jest równoważne zadaniu

$$\begin{aligned} \text{minimize} \quad & c^{\mathrm{T}}x \\ \text{subject to} \quad & Ax \leqslant b \\ & A_{\mathrm{eq}}x = b_{\mathrm{eq}} \\ & x_{\mathrm{LB}} \leqslant x \leqslant x_{\mathrm{UB}} \end{aligned} \tag{2}$$

gdzie

$$A = \begin{bmatrix} 1 & 1 \\ 1 & \frac{1}{4} \\ 1 & -1 \\ -\frac{1}{4} & -1 \\ -1 & -1 \\ -1 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 2 \\ 1 \\ 2 \\ 1 \\ -1 \\ 2 \end{bmatrix}, \quad c = \begin{bmatrix} 1 \\ \frac{1}{2} \end{bmatrix} \tag{3}$$

$$A_{\mathrm{eq}} = \begin{bmatrix} 1 & \frac{1}{4} \end{bmatrix}, \quad b_{\mathrm{eq}} = \begin{bmatrix} \frac{1}{2} \end{bmatrix} \tag{4}$$

$$x_{\mathrm{LB}} = \begin{bmatrix} -1 \\ -\frac{1}{2} \end{bmatrix}, \quad x_{\mathrm{UB}} = \begin{bmatrix} \frac{3}{2} \\ \frac{5}{4} \end{bmatrix} \tag{5}$$

Zadanie (2) jest w oczywisty sposób równoważne zadaniu

$$\begin{aligned} \text{minimize} \quad & c^{\mathrm{T}}x \\ \text{subject to} \quad & \begin{bmatrix} A \\ I \\ -I \end{bmatrix} x \leqslant \begin{bmatrix} b \\ x_{\mathrm{UB}} \\ -x_{\mathrm{LB}} \end{bmatrix} \\ & A_{\mathrm{eq}}x = b_{\mathrm{eq}} \end{aligned} \tag{6}$$

Zatem możemy zdefiniować

$$\tilde{A} = \begin{bmatrix} A \\ I \\ -I \end{bmatrix}, \quad \tilde{b} = \begin{bmatrix} b \\ x_{\mathrm{UB}} \\ -x_{\mathrm{LB}} \end{bmatrix}, \tag{7}$$

i rozwiązać zadanie

$$\begin{aligned} \text{maximize} \quad & c^{\mathrm{T}}x \\ \text{subject to} \quad & \tilde{A}x \leqslant \tilde{b} \\ & A_{\mathrm{eq}}x = b_{\mathrm{eq}} \end{aligned} \tag{8}$$

Korzystając ze środowiska Matlab mamy do dyspozycji bibliotekę Optimization Toolbox, oraz dodatkowy, bardzo wygodny pakiet CVX. Jeśli chodzi o język Python, to możliwości jest wiele, dla naszych potrzeb skorzystamy z modułów CVXPY i CVXOPT.

1.0.1 Matlab

```
clear all
close all
clc
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x1 = optimvar('x1','LowerBound',-1,'UpperBound',1.5);
x2 = optimvar('x2','LowerBound',-1/2,'UpperBound',1.25);
p = optimproblem('Objective',x1 + x2/2,'ObjectiveSense','min');
p.Constraints.c1 = x1 + x2 <= 2;
```

```
p.Constraints.c2 = x1 + x2/4 <= 1;
p.Constraints.c3 = x1 - x2 <= 2;
p.Constraints.c4 = x1/4 + x2 >= -1;
p.Constraints.c5 = x1 + x2 >= 1;
p.Constraints.c6 = -x1 + x2 <= 2;
p.Constraints.c7 = x1 + x2/4 == 1/2;
options = optimoptions('linprog','Algorithm','dual-simplex','OptimalityTolerance',1e-10);
% options = optimoptions('linprog','Algorithm','interior-point','OptimalityTolerance',1e-10);
sol = solve(p,'Options',options);
x1 = sol.x1;
x2 = sol.x2;
disp('-------------------------------')
disp('optimal solution - first method')
disp([x1,x2])
disp('-------------------------------')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

A = [1.0, 1.0;
    1.0, 0.25;
    1.0, -1.0;
    -0.25, -1.0;
    -1.0 -1.0;
    -1.0 1.0];
b = [2; 1; 2; 1; -1; 2];
c = [1; 1.5];
Aeq = [1 0.25];
beq = 0.5;
LB = [-1; -0.5];
UB = [1.5; 1.25];
x = optimvar('x',2,1,'LowerBound',LB,'UpperBound',UB);
p = optimproblem('Objective',c'*x,'ObjectiveSense','min');
p.Constraints.c1 = A*x <= b;
p.Constraints.c2 = Aeq*x == beq;
options = optimoptions('linprog','Algorithm','dual-simplex','OptimalityTolerance',1e-10);
% options = optimoptions('linprog','Algorithm','interior-point','OptimalityTolerance',1e-10);
sol = solve(p,'Options',options);
x = sol.x;
disp('-------------------------------')
disp('optimal solution - second method')
disp(x)
disp('-------------------------------')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

A = [1.0, 1.0;
    1.0, 0.25;
    1.0, -1.0;
    -0.25, -1.0;
    -1.0 -1.0;
    -1.0 1.0];
b = [2; 1; 2; 1; -1; 2];
c = [1; 1.5];
Aeq = [1 0.25];
beq = 0.5;
LB = [-1; -0.5];
UB = [1.5; 1.25];
xOpt = linprog(c,A,b,Aeq,beq,LB,UB);
disp('-------------------------------')
disp('optimal solution - third method')
disp(xOpt)
disp('-------------------------------')


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
cvx_begin
```

```
    variables x1 x2
    x1 + x2 <= 2
    x1 + x2/4 <= 1
    x1 - x2 <= 2
    x1/4 + x2 >= -1
    x1 + x2 >= 1
    -x1 + x2 <= 2
    x1 + x2/4 == 1/2
    -1 <= x1 <= 1.5
    -1/2 <= x2 <= 1.25
    minimize ( x1 + 0.5*x2 )
cvx_end
disp('-------------------------------')
disp('optimal solution - fourth method')
disp([x1,x2])
disp('-------------------------------')
```

1.0.2 Python - moduł CVXPY

```
import numpy as np
import cvxpy as cp

################################
## method 1
################################
x1 = cp.Variable()
x2 = cp.Variable()

objective = cp.Minimize(x1 + 0.5*x2)
constraints = [ x1 + x2 <= 2,
                x1 + x2/4 <= 1,
                x1 - x2 <= 2,
                x1/4 + x2 >= -1,
                x1 + x2 >= 1,
                -x1 + x2 <= 2,
                x1 + x2/4 == 1/2,
                -1 <= x1,
                x1 <= 1.5,
                -1/2 <= x2,
                x2 <= 1.25 ]
p1 = cp.Problem(objective, constraints)
p1.solve()
print("--------------")
print(x1.value)
print(x2.value)
print("--------------")

################################
## method 2
################################
n = 2
x = cp.Variable(n)
A = np.array([[1.0,1.0],[1.0,0.25],[1.0,-1.0],[-0.25,-1.0],[-1.0,-1.0],[-1.0,1.0]])
b = np.array([2.0, 1.0, 2.0, 1.0, -1.0, 2.0])
c = np.array([1, 1.5])
Aeq = np.array([1.0, 0.25])
beq = np.array([0.5])
LB = np.array([-1.0, -0.5])
UB = np.array([1.5, 1.25])
objective = cp.Minimize(c.T @ x)
constraints = [ A @ x <= b, Aeq @ x == beq, x <= UB, x >= LB ]
p2 = cp.Problem(objective, constraints)
p2.solve()
print("\n")
print("--------------")
```

```
print(x.value)
print("--------------")

################################
## method 3
################################
n = 2
x = cp.Variable(n)
A = np.array([[1.0,1.0],[1.0,0.25],[1.0,-1.0],[-0.25,-1.0],[-1.0,-1.0],[-1.0,1.0]])
b = np.array([2.0, 1.0, 2.0, 1.0, -1.0, 2.0])
c = np.array([1, 1.5])
Aeq = np.array([1.0, 0.25])
beq = np.array([0.5])
LB = np.array([-1.0, -0.5])
UB = np.array([1.5, 1.25])
AA = np.vstack((A,np.eye(n),-np.eye(n)))
bb = np.hstack((b,UB,-LB))

objective = cp.Minimize(c.T @ x)
constraints = [ AA @ x <= bb, Aeq @ x == beq ]
p3 = cp.Problem(objective, constraints)
p3.solve()
print("\n")
print("--------------")
print(x.value)
print("--------------")
```

1.0.3 Python - moduł CVXOPT

```
import numpy as np
import cvxopt as co
from cvxopt.modeling import  variable, op, dot, matrix

################################
## method 1
################################
x1 = variable()
x2 = variable()

c1 = ( x1 + x2 <= 2 )
c2 = ( x1 + x2/4 <= 1 )
c3 = ( x1 - x2 <= 2 )
c4 = ( x1/4 + x2 >= -1 )
c5 = ( x1 + x2 >= 1 )
c6 = ( -x1 + x2 <= 2 )
c7 = ( x1 + x2/4 == 1/2 )
c8 = ( -1 <= x1 <= 1.5 )
c9 = ( -1/2 <= x2 <= 1.25 )

p1 = op(x1 + 0.5*x2, [c1,c2,c3,c4,c5,c6,c7,c8,c9])
p1.solve()
##print("\n", p1.objective.value())
print("--------------")
print(x1.value)
print(x2.value)
print("--------------")
##print("\n", c1.multiplier.value)

################################
## method 2
################################
n = 2
x = variable(n)
A = np.array([[1.0,1.0],[1.0,0.25],[1.0,-1.0],[-0.25,-1.0],[-1.0,-1.0],[-1.0,1.0]])
A = matrix(A)
```

```
b = matrix([2.0, 1.0, 2.0, 1.0, -1.0, 2.0])
c = matrix([1, 1.5])
Aeq = matrix([[1.0], [0.25]])
beq = matrix([0.5])
LB = matrix([-1.0, -0.5])
UB = matrix([1.5, 1.25])

c1 = ( A*x <= b )
c2 = ( Aeq*x == beq)
c3 = ( x <= UB )
c4 = ( x >= LB )
p2 = op(dot(c,x), [c1,c2,c3,c4])
p2.solve()
##print("\n", p2.objective.value())
print("\n")
print("--------------")
print(x.value)
print("--------------")

##############################
## method 3
##############################
n = 2
A = matrix([[1.0, 1.0, 1.0, -0.25, -1.0, -1.0], [1.0, 0.25, -1.0, -1.0, -1.0, 1.0]])
b = matrix([2.0, 1.0, 2.0, 1.0, -1.0, 2.0]);
c = matrix([1, 1.5])
Aeq = matrix([[1.0], [0.25]])
beq = matrix([0.5])
LB = matrix([-1.0, -0.5])
UB = matrix([1.5, 1.25])

AA = matrix(np.vstack((A,np.eye(n),-np.eye(n))))
bb = matrix(np.vstack((b,UB,-LB)))
sol = co.solvers.lp(c,AA,bb,Aeq,beq)
print("\n", sol['x'][0], sol['x'][1])
```

---

2 Zadania

Zadanie 1. Rozwiązać poniższe zadanie [4] korzystając z języka Python lub w środowisku Matlab korzystając z funkcji a) `linprog`, b) `solve` (porównać wyniki dla `'dual-simplex'` i `'interior-poin'`), c) pakietu CVX [5].

Tabela 1

	węglo–wodany	białko	sole min.	cena [PLN/tona]
pszenica	0.8	0.01	0.15	300
soja	0.3	0.4	0.1	500
mączka	0.1	0.7	0.2	800
zapotrzebowanie	0.3	0.7	0.1	

Jak wymieszać pszenicę, soję i mączkę rybną aby uzyskać najtańszą mieszankę paszową zapewniającą wystarczającą zawartość węglowodanów, białka i soli mineralnych dla kurcząt.
Rozpoczynamy od zdefiniowania zmiennych. Niech x_i oznacza masę i-tego składnika w mieszance. Funkcją celu jest koszt mieszanki

$$f_0 = 300x_1 + 500x_2 + 800x_3 \tag{9}$$

Ograniczenia są dwojakiego typu.

(a) Mieszanka musi zawierać wystarczającą ilość węglowodanów, białka, soli mineralnych, tzn.

$$0.8x_1 + 0.3x_2 + 0.1x_3 \geqslant 0.3 \tag{9a}$$
$$0.01x_1 + 0.4x_2 + 0.7x_3 \geqslant 0.7 \tag{9b}$$
$$0.15x_1 + 0.1x_2 + 0.2x_3 \geqslant 0.1 \tag{9c}$$

(b) Masa używanych składników musi być nieujemna, tzn.

$$x_1 \geqslant 0, \quad x_2 \geqslant 0, \quad x_3 \geqslant 0. \tag{10}$$

Rozwiązanie: $x_1^\star = 0$, $x_2^\star = 0.8235$, $x_3^\star = 0.5294$.

Zadanie 2. Optymalne śniadanie ([6], Exercise 9.5). Dane są $n = 3$ typy potraw, charakterystykę każdego typu opisuje Tabela 2. Wyznacz, korzystając z języka Python lub w środowisku Matlab korzystając z funkcji a) `linprog`, b) `solve` (porównać wyniki dla `'dual-simplex'` i `'interior-poin'`), c) pakietu CVX [5] skład (liczbę porcji każdego typu potrawy) najtańszego śniadania, o zawartości kalorii pomiędzy 2000 i 2250, zawartości witamin pomiędzy 5000 i 10000, i poziomie cukru nie wyższym niż 1000, zakładając że maksymalna liczba porcji każdego typu nie może przekraczać 10.

Tabela 2

potrawa	koszt	witaminy	cukier	kalorie
płatki	0.15	107	45	70
mleko	0.25	500	40	121
chleb	0.05	0	60	65

Rozwiązanie: oznaczając x_1 - ilość płatków, x_2 - ilość mleka, x_3 - ilość chleba, otrzymujemy wartości optymalne $x_1^\star = 6.5882$, $x_2^\star = 10.0000$, $x_3^\star = 5.0588$.

Zadanie 3. [7, 8] Przedsiębiorstwo produkuje dwa rodzaje leków, *Lek I* oraz *Lek II*, które zawierają czynnik aktywny A, który pozyskuje się z surowców dostępnych na rynku. Dostępne są dwa surowce, *Surowiec I* oraz *Surowiec II*, z których można pozyskiwać czynnik aktywny A.
Dane dotyczące produkcji, kosztów, zasobów produkcyjnych umieszczono w Tabelach 3–5. Należy wyznaczyć plan produkcji, który zmaksymalizuje zyski przedsiębiorstwa.

Tabela 3: Dane produkcyjne

parametr [na 1000 opakowań]	Lek I	Lek II
cena sprzdaży [USD]	6500	7100
zawartość czynnika aktywnego A [gram]	0.500	0.600
zasoby ludzkie [h]	90.0	100.0
zasoby sprzętowe [h]	40.0	50.0
koszty operacyjne [USD]	700	800

Tabela 4: Zawartość czynnika aktywnego A w surowcach

surowiec	cena zakupu [USD/kg]	zawartość czynnika aktywnego A [gram/kg]
Surowiec I	100.00	0.01
Surowiec II	199.90	0.02

Tabela 5: Zasoby

budżet [USD]	100000
zasoby ludzkie [h]	2000
zasoby sprzętowe [h]	800
zasoby magazynowe [kg]	1000

Przyjmijmy następujące oznaczenia. Niech x_{LekI} oznacza ilość *Leku I*, zaś x_{LekII} oznacza ilość *Leku II*, na każde 1000 wyprodukowanych opakowań. Niech x_{SurI} oznacza ilość (w [kg]) zakupionego *Surowca I*, zaś x_{SurII}, odpowiednio, *Surowca II*.

Funkcja celu, którą należy *zminimalizować* jest postaci

$$f_0(x) = f_{\text{costs}}(x) - f_{\text{income}}(x), \tag{11}$$

gdzie

$$x = \begin{bmatrix} x_{\text{LekI}} & x_{\text{LekII}} & x_{\text{SurI}} & x_{\text{SurII}} \end{bmatrix}^{\mathrm{T}}, \tag{12}$$

jest wektorem zmniennych decyzyjnych (optymalizacyjnych),

$$f_{\text{costs}}(x) = 100.00x_{\text{SurI}} + 199.90x_{\text{SurII}} + \\ + 700.00x_{\text{LekI}} + 800.00x_{\text{LekII}} \tag{13}$$

reprezentuje koszty zakupów oraz produkcji, zaś

$$f_{\text{income}}(x) = 6500.00x_{\text{LekI}} + 7100.00x_{\text{LekII}} \tag{14}$$

przedstawia zysk ze sprzedaży leków. Ponadto, w rozpatrywanym zadaniu występują następujące ograniczenia.

1. Bilans czynnika aktywnego

$$0.01x_{\text{SurI}} + 0.02x_{\text{SurII}} - 0.50x_{\text{LekI}} - 0.60x_{\text{LekII}} \geqslant 0. \tag{15}$$

2. Ograniczenia zasobów magazynowych (magazynowanie zakupionych surowców)

$$x_{\text{SurI}} + x_{\text{SurII}} \leqslant 1000. \tag{16}$$

3. Ograniczenia zasobów ludzkich

$$90.00x_{\text{LekI}} + 100.00x_{\text{LekII}} \leqslant 2000. \tag{17}$$

4. Ograniczenia zasobów sprzętowych

$$40.00x_{\text{LekI}} + 50.00x_{\text{LekII}} \leqslant 800. \tag{18}$$

5. Ograniczenia budżetowe

$$100.00x_{\text{SurI}} + 199.90x_{\text{SurII}} + \\ + 700.00x_{\text{LekI}} + 800.00x_{\text{LekII}} \leqslant 100000. \tag{19}$$

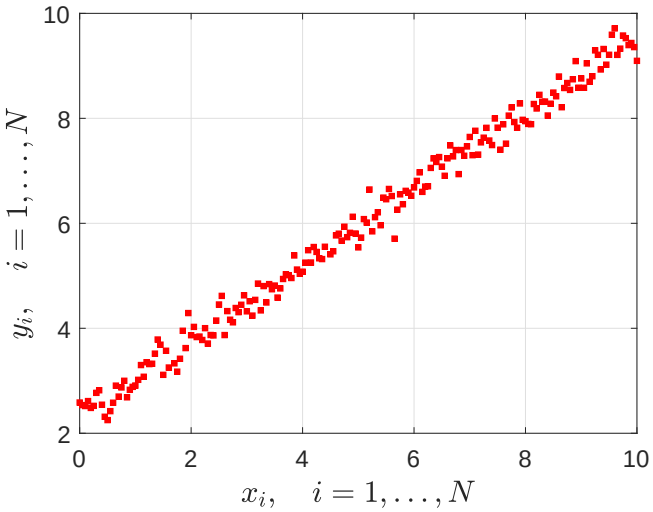
6. Ograniczenia zakresu zmiennych

$$x_{\text{SurI}} \geqslant 0, \quad x_{\text{SurII}} \geqslant 0, \quad x_{\text{LekI}} \geqslant 0, \quad x_{\text{LekII}} \geqslant 0. \tag{20}$$

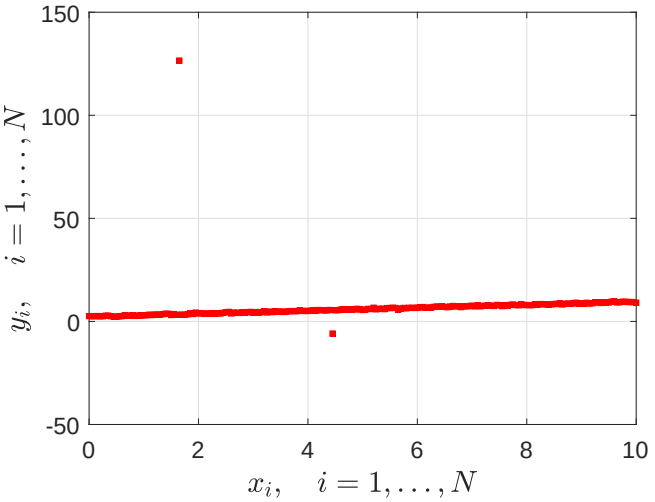
Zadanie należy rozwiązać korzystając z języka Python lub w środowisku Matlab korzystając z a) `linprog`, b) `solve` (porównać wyniki dla `'dual-simplex'` i `'interior-poin'`), c) pakietu CVX.
Rozwiązanie: $x_{\text{LekI}}^\star = 17.552$, $x_{\text{LekII}}^\star = 0$, $x_{\text{SurI}}^\star = 0$, $x_{\text{SurII}}^\star = 438.789$.

Zadanie 4. Chcemy wyznaczyć możliwie najlepsze dopasowanie prostej $y = ax + b$ do zadanego zbioru punktów (Rys. 1).

$$\left\{ \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}, \ldots, \begin{bmatrix} x_N \\ y_N \end{bmatrix} \right\} \tag{21}$$



Rysunek 1: Punkty tworzące trend.



Rysunek 2: Dane do Zadania 4. Dwa punkty nie pasują do danych (ang. *outliers*).

Dla i-tego punktu przyjęty model zwraca wartość ax_i+b, chcemy żeby była ona możliwie blisko wartości y_i. Możemy zatem wprowadzić różene miary dopasowania. W dalszym ciągu rozpatrzymy dwa przypadki.

1. Suma modułów (wartości bezwzględnych) różnic

$$\phi(a,b) = \sum_{i=1}^{N} |ax_i + b - y_i| \,. \tag{22}$$

2. Suma kwadratów różnic

$$\psi(a,b) = \sum_{i=1}^{N} (ax_i + b - y_i)^2 \,. \tag{23}$$

Wprowadzając oznaczenia

$$\boldsymbol{\theta} = \begin{bmatrix} a \\ b \end{bmatrix}, \quad \boldsymbol{\Phi} = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_N & 1 \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \tag{24}$$

i odrobinę nadużywając notacji możemy napisać

$$\phi(\boldsymbol{\theta}) = \|\boldsymbol{\Phi\theta} - \boldsymbol{y}\|_1 \,, \quad \psi(\boldsymbol{\theta}) = \|\boldsymbol{\Phi\theta} - \boldsymbol{y}\|_2^2 \tag{25}$$

Postawione zadanie dopasowania prostej (modelu) sprowadza się zatem do minimalizacji funkcji $\phi(\boldsymbol{\theta})$ lub $\psi(\boldsymbol{\theta})$

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad \|\boldsymbol{\Phi\theta} - \boldsymbol{y}\|_1 \tag{26}$$

lub

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad \|\boldsymbol{\Phi\theta} - \boldsymbol{y}\|_2^2 \tag{27}$$

Rozwiązanie zadania optymalizacji (27) dane jest wzorem

$$\hat{\boldsymbol{\theta}} = \boldsymbol{\Phi}^{\dagger}\boldsymbol{y}, \tag{28}$$

gdzie $\boldsymbol{\Phi}^{\dagger}$ oznacza pseodoodwrotność Moore'a-Penrose'a (w środowisku Matlab można ją wyznaczyć za pomocą polecenia `pinv`), natomiast (26) można sprawadzić do zadania LP (programowania liniowego) wprowadzając dodatkowe zmienne. W szczególności można pokazać, że zadanie (26) jest równoważne zadaniu LP

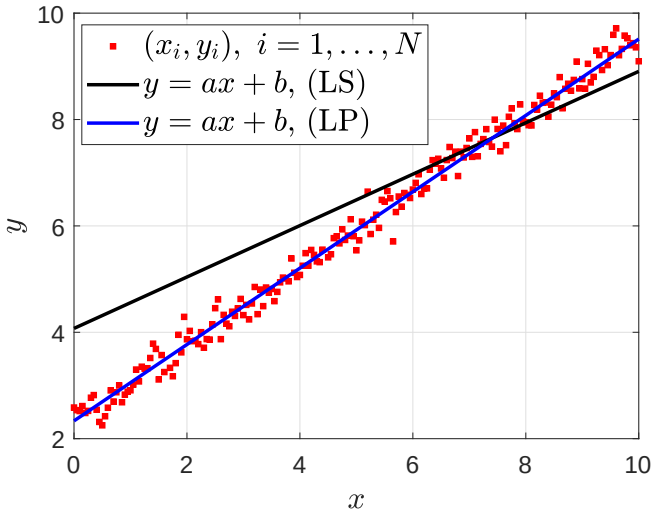
$$\begin{aligned} \underset{\boldsymbol{\theta},\boldsymbol{\tau}}{\text{minimize}} \quad & \begin{bmatrix} \mathbf{0} \\ \mathbf{1} \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} \boldsymbol{\theta} \\ \boldsymbol{\tau} \end{bmatrix} \\ \text{subject to} \quad & \begin{bmatrix} \boldsymbol{\Phi} & -\boldsymbol{I} \\ -\boldsymbol{\Phi} & -\boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{\theta} \\ \boldsymbol{\tau} \end{bmatrix} \leqslant \begin{bmatrix} \boldsymbol{y} \\ -\boldsymbol{y} \end{bmatrix} \end{aligned} \tag{29}$$

gdzie $\boldsymbol{\tau} \in \mathbb{R}^N$, $\mathbf{1}$ oznacza wektor złożony z samych jedynek, zaś $\boldsymbol{I}$ oznacza macierz jednostkową odpowiednich wymiarów.

Polecenie

Pobrać plik danych `Data01.mat` Korzystając z języka Python lub w środowisku Matlab wygenerować wykres danych (Rys. 2), wyznaczyć odpowiednie proste (podobnie jak przedstawiono na Rys. 3) korzystając z funkcji `linprog` do wyznaczenia rozwiązania zadania (29). Wygenerować wykresy przedstawione na Rys. 3.

Rozwiązanie: $a^{\star}_{\mathrm{LP}} = 0.7178$, $b^{\star}_{\mathrm{LP}} = 2.3346$, $a^{\star}_{\mathrm{LS}} = 0.4832$, $b^{\star}_{\mathrm{LS}} = 4.0729$



Rysunek 3: Wykresy prostych $y = ax + b$ dopasowanych do danych. Parametry prostej czarnej wyznaczono metodą LS, parametry prostej niebieskiej wyznaczono metodą LP. Jak można zauważyć, metoda LP jest bardziej odporna na obecność danych obarczonych błędem grubym (ang. *outliers*).

Odporność metody wyznaczania modelu na błędy grube w danych pomiarowych jest ważnym zagadnieniem, jednak stwierdzenie czy dany wynik jest obarczony błędem grubym nie zawsze jest proste.

Dodatek

Zadanie

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad \|\boldsymbol{\Phi\theta} - \boldsymbol{y}\|_1 \tag{30}$$

czyli

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \sum_{i=1}^{N} |\boldsymbol{\varphi}_i^{\mathrm{T}}\boldsymbol{\theta} - y_i| \,, \tag{31}$$

gdzie

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\varphi}_1^{\mathrm{T}} \\ \vdots \\ \boldsymbol{\varphi}_N^{\mathrm{T}} \end{bmatrix}, \tag{32}$$

jest równoważne zadaniu

$$\begin{aligned} \underset{\boldsymbol{\theta},\boldsymbol{\tau}}{\text{minimize}} \quad & \sum_{i=1}^{N} \tau_i \\ \text{subject to} \quad & |\boldsymbol{\varphi}_i^{\mathrm{T}}\boldsymbol{\theta} - y_i| \leqslant \tau_i, \quad i = 1, \ldots, N \end{aligned} \tag{33}$$

czyli

$$\begin{aligned} \underset{\boldsymbol{\theta},\boldsymbol{\tau}}{\text{minimize}} \quad & \sum_{i=1}^{N} \tau_i \\ \text{subject to} \quad & -\tau_i \leqslant \boldsymbol{\varphi}_i^{\mathrm{T}}\boldsymbol{\theta} - y_i \leqslant \tau_i, \quad i = 1, \ldots, N \end{aligned} \tag{34}$$

czyli

$$\begin{aligned} \underset{\boldsymbol{\theta},\boldsymbol{\tau}}{\text{minimize}} \quad & \mathbf{1}^{\mathrm{T}}\boldsymbol{\tau} \\ \text{subject to} \quad & -\boldsymbol{\tau} \leqslant \boldsymbol{\Phi}\boldsymbol{\theta} - \boldsymbol{y} \leqslant \boldsymbol{\tau}, \end{aligned} \tag{35}$$

gdzie

$$\boldsymbol{\tau} = \begin{bmatrix} \tau_1 \\ \vdots \\ \tau_N \end{bmatrix}, \tag{36}$$

czyli

$$\begin{aligned} \underset{\boldsymbol{\theta},\boldsymbol{\tau}}{\text{minimize}} \quad & \mathbf{1}^{\mathrm{T}}\boldsymbol{\tau} \\ \text{subject to} \quad & \boldsymbol{\Phi}\boldsymbol{\theta} - \boldsymbol{y} \leqslant \boldsymbol{\tau} \\ & \boldsymbol{\Phi}\boldsymbol{\theta} - \boldsymbol{y} \geqslant -\boldsymbol{\tau} \end{aligned} \tag{37}$$

czyli

$$\begin{aligned} \underset{\boldsymbol{\theta},\boldsymbol{\tau}}{\text{minimize}} \quad & \mathbf{1}^{\mathrm{T}}\boldsymbol{\tau} \\ \text{subject to} \quad & \boldsymbol{\Phi}\boldsymbol{\theta} - \boldsymbol{\tau} \leqslant \boldsymbol{y} \\ & -\boldsymbol{\Phi}\boldsymbol{\theta} - \boldsymbol{\tau} \leqslant -\boldsymbol{y} \end{aligned} \tag{38}$$

czyli

$$\begin{aligned} \underset{\boldsymbol{\theta},\boldsymbol{\tau}}{\text{minimize}} \quad & \begin{bmatrix} \mathbf{0} \\ \mathbf{1} \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} \boldsymbol{\theta} \\ \boldsymbol{\tau} \end{bmatrix} \\ \text{subject to} \quad & \begin{bmatrix} \boldsymbol{\Phi} & -\boldsymbol{I} \\ -\boldsymbol{\Phi} & -\boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{\theta} \\ \boldsymbol{\tau} \end{bmatrix} \leqslant \begin{bmatrix} \boldsymbol{y} \\ -\boldsymbol{y} \end{bmatrix} \end{aligned} \tag{39}$$

przyjmując oznaczenia

$$\boldsymbol{c} = \begin{bmatrix} \mathbf{0} \\ \mathbf{1} \end{bmatrix}, \qquad \boldsymbol{z} = \begin{bmatrix} \boldsymbol{\theta} \\ \boldsymbol{\tau} \end{bmatrix} \tag{40}$$

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{\Phi} & -\boldsymbol{I} \\ -\boldsymbol{\Phi} & -\boldsymbol{I} \end{bmatrix} \qquad \boldsymbol{b} = \begin{bmatrix} \boldsymbol{y} \\ -\boldsymbol{y} \end{bmatrix} \tag{41}$$

otrzymujemy

$$\begin{aligned} \underset{\boldsymbol{z}}{\text{minimize}} \quad & \boldsymbol{c}^{\mathrm{T}}\boldsymbol{z} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{z} \leqslant \boldsymbol{b} \end{aligned} \tag{42}$$

Pierwsze dwa elementy rozwiązania $z^\star$ zadania (42) są poszukiwanymi współczynnikami a i b prostej trendu.

3 Metody sympleksowe versus metdy puntu wewnętrznego - rys historyczny

Zacytowany poniżej w charakterze ciekawostki fragment pochodzi z [2], gdzie można znaleźć opis metody sympleks. Wprowadzenie do metod punktu wewnętrznego można znaleźć w [1].

Linear programming in polynomial time

Simplex is not a polynomial time algorithm. Certain rare kinds of linear programs cause it to go from one corner of the feasible region to a better corner and then to a still better one, and so on for an exponential number of steps. For a long time, linear programming was considered a paradox, a problem that can be solved in practice, but not in theory!

Then, in 1979, a young Soviet mathematician called Leonid Khachiyan came up with the ellipsoid algorithm*, one that is very different from simplex, extremely simple in its conception (but sophisticated in its proof) and yet one that solves any linear program in polynomial time. Instead of chasing the solution from one corner of the polyhedron to the next, Khachiyan's algorithm confines it to smaller and smaller ellipsoids (skewed high-dimensional balls). When this algorithm was announced, it became a kind of mathematical Sputnik, a splashy achievement that had the U.S. establishment worried, in the height of the Cold War, about the possible scientific superiority of the Soviet Union. The ellipsoid algorithm turned out to be an important theoretical advance, but did not compete well with simplex in practice. The paradox of linear programming deepened: A problem with two algorithms, one that is efficient in theory, and one that is efficient in practice!*

A few years later Narendra Karmarkar, a graduate student at UC Berkeley, came up with a completely different idea, which led to another provably polynomial algorithm for linear programming. Karmarkar's algorithm is known as the interior point method*, because it does just that: it dashes to the optimum corner not by hopping from corner to corner on the surface of the polyhedron like simplex does, but by cutting a clever path in the interior of the polyhedron. And it does perform well in practice.*

But perhaps the greatest advance in linear programming algorithms was not Khachiyan's theoretical breakthrough or Karmarkar's novel approach, but an unexpected consequence of the latter: the fierce competition between the two approaches, simplex and interior point, resulted in the development of very fast code for linear programming.

Literatura

[1] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004. `http://web.stanford.edu/~boyd/cvxbook/`.

[2] Sanjoy Dasgupta, Christos Papadimitriou, and Umesh Vazirani. *Algorytmy*. Wydawnictwo Naukowe PWN, Warszawa, 2012.

[3] Ulrich Münz, Amer Mešanović, Michael Metzger, and Philipp Wolfrum. Robust optimal dispatch, secondary, and primary reserve allocation for power systems with uncertain load and generation. *IEEE Transactions on Control Systems Technology*, 26(2):475–485, 2018.

[4] Andrzej Strojnowski. *Optymalizacja I, skrypt do wykładu*. `http://www.mimuw.edu.pl/~stroa`, 2012.

[5] Inc. CVX Research. CVX: Matlab software for disciplined convex programming, version 2.0. `http://cvxr.com/cvx`, August 2012.

[6] E.K.P. Chong and S.H. Zak. *An Introduction to Optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimi. Wiley, 2004.

[7] A. Ben-Tal and A. Nemirovski. *Lectures on Modern Convex Optimization*. SIAM, 2001.

[8] G.C. Calafiore and L. El Ghaoui. *Optimization Models*. Control systems and optimization series. Cambridge University Press, 2014.