1DI2235:A - Metody optymalizacji, lab.
Programowanie liniowe (LP)

(opracował: M.T., ostatnia modyfikacja: 21 lutego 2023)

1 Wstęp

1.1 Postać standardowa zadania optymalizacji

[1] Rozpatrujemy zadanie optymalizacji w standardowej postaci

$$\begin{array}{lll} \text{minimize} & f_0(\boldsymbol{x}) & \\ \text{subject to} & f_i(\boldsymbol{x}) \leqslant 0, & i = 1, \ldots, m \\ & h_i(\boldsymbol{x}) = 0, & i = 1, \ldots, p \end{array} \tag{1}$$

gdzie $\boldsymbol{x} \in \mathbb{R}^n$ oznacza wektor zmniennych optymalizacyjnych (decyzyjnych), funkcję $f_0 : \mathbb{R}^n \to \mathbb{R}$ nazywamy funkcją celu (ang. *objective function*), funkcje $f_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, \ldots, m$ nazywamy funkcjami ograniczeń (więzów) nierównościowych, zaś funkcje $h_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, \ldots, p$ nazywamy funkcjami ograniczeń (więzów) równościowych. Zadanie (1) oznacza poszukiwanie wektora, który minimalizuje funkcję celu, spośród wszystkich $\boldsymbol{x}$ spełniających warunki $f_i(\boldsymbol{x}) \leqslant 0$, $i = 1, \ldots, m$ oraz $h_i(\boldsymbol{x}) = 0$, $i = 1, \ldots, p$. Zbiór punktów $\boldsymbol{x}$, dla których funkcja celu oraz funkcje ograniczeń są określone, nazywamy dziedziną $\mathcal{D}$ zadania (1)

$$\mathcal{D} \equiv \bigcap_{i=0}^{m} \mathbf{dom} f_i \cap \bigcap_{i=1}^{p} \mathbf{dom} h_i.$$

Będziemy zakładać, że $\mathcal{D} \neq \emptyset$ Punkt $\boldsymbol{x} \in \mathcal{D}$ nazywamy dopuszczalnym (ang. *feasible*), jeśli spełnia ograniczenia $f_i(\boldsymbol{x}) = 0$, $i = 1, \ldots, m$ oraz $g_i(\boldsymbol{x}) \leqslant 0$, $i = 1, \ldots, p$. Zadanie (1) będziemy nazywać *dopuszczalnym* (ang. *feasible*)jeśli istnieje co najmniej jeden dopuszczalny punkt $\boldsymbol{x}$. W innym przypadku, zadanie (1) będziemy nazywać *niedopuszczalnym* (ang. *infeasible*) Wartość optymalną dla (1) oznaczaną p^* definiujemy

$$p^* \equiv \inf_{\boldsymbol{x} \in \mathcal{D}} \left\{ f_0(\boldsymbol{x}) \;\middle|\; f_i(\boldsymbol{x}) = 0 \;\text{ dla }\; i = 1, \ldots, m, \;\; h_i(\boldsymbol{x}) \leqslant 0 \;\text{ dla }\; i = 1, \ldots, p \right\}.$$

Należy podkreślić, że wartość optymalna p^* nie ma związku z liczbą więzów równościowych p. Będziemy dopuszczać przyjmowanie przez p^* wartości $\pm\infty$. Zgodnie z konwencją $\inf \emptyset = \infty$, w związku z czym dla zadań niedopuszczalnych (ang. *infeasible*) piszemy $p^* = \infty$. Jeśli natomiast dla danego zadania istnieje ciąg punktów dopuszczalnych $\boldsymbol{x}_k$ taki, że $f_0(\boldsymbol{x}_k) \overset{k \to \infty}{\to} -\infty$, to piszemy $p^* = -\infty$ i mówimy, że zadanie jest nieograniczone z dołu (ang. *unbounded below*). Punkt $\boldsymbol{x}^*$ nazywamy *punktem optymalnym* (ang. *optimal point*), lub rozwiązaniem zadania (1) jeśli $\boldsymbol{x}^*$ jest dopuszczalny oraz $f_0(\boldsymbol{x}^*) = p^*$. Zbiór wszystkich punktów optymalnych nazywamy *zbiorem optymalnym* (ang. *optimal set*)

$$\Omega_{\text{optimal}} = \{ \boldsymbol{x} \,|\, f_i(\boldsymbol{x}) \leqslant 0, \;\; i = 1, \ldots, m, \;\; h_i(\boldsymbol{x}) = 0, \;\; i = 1, \ldots, p, \;\; f_0(\boldsymbol{x}) = p^* \}$$

Jeżeli dla zadania (1) zbiór optymalny $\Omega_{\text{optimal}} \neq \emptyset$ to mówimy, że wartość optymalna (funkcji celu) jest osiągana (ang. *attained* lub *achieved*), zaś zadanie optymalizacji jest rozwiązywalne (ang. *solvable*). Jeśli $\Omega_{\text{optimal}} = \emptyset$ to mówimy, że wartośc optymalna nie jest osiągana, sytuacja taka zawsze ma miejsce, jeśli zadanie jest nieograniczone z dołu.

1.2 Zadanie optymalizacji wypukłej w postaci standardowej

Zadanie optymalizacji wypukłej jest postaci

$$\begin{array}{lll} \text{minimize} & f_0(\boldsymbol{x}) & \\ \text{subject to} & \boldsymbol{a}_i^{\mathrm{T}} \boldsymbol{x} = b_i, & i = 1, \ldots, m \\ & h_i(\boldsymbol{x}) \leqslant 0, & i = 1, \ldots, p \end{array} \tag{2}$$

gdzie f_0 oraz $h_0, \ldots, h_p$ są funkcjami wypukłymi. Standardowa postać zadania optymalizacji wypukłej charakteryzuje się tym, że funkcja celu oraz funkcje ograniczeń nierównościowych są wypukłe, zaś funkcje ograniczeń równościowych są afiniczne. Zbiór dopuszczalny dla zadania optymalizacji wypukłej jest zbiorem wypukłym, co wynika stąd, że jest on przecięciem wypukłej dziedziny zadania

$$\mathcal{D} = \bigcap_{i=0}^{m} \mathbf{dom} h_i,$$

z p wypukłymi zbiorami subwarstwicowymi (ang. *sublevel sets*) $\{\boldsymbol{x} \mid h_i(\boldsymbol{x}) \leqslant 0\}$ i m hiperpłaszczyznami $\{\boldsymbol{x} \mid \boldsymbol{a}_i^{\mathrm{T}} \boldsymbol{x} = b_i\}$. Bez straty ogólności rozważań można założyć, że $\boldsymbol{a}_i \neq 0$ dla $i = 1, \ldots, m$

Uwaga 1. Zadanie

$$\begin{array}{lll} \text{maximize} & f_0(\boldsymbol{x}) & \\ \text{subject to} & \boldsymbol{a}_i^{\mathrm{T}} \boldsymbol{x} = b_i, & i = 1, \ldots, m \\ & h_i(\boldsymbol{x}) \leqslant 0, & i = 1, \ldots, p \end{array} \tag{3}$$

dla *wklęsłej* (ang. *concave*) funckji celu f_0, będziemy traktować jako zadanie optymalizacji wypukłej, ponieważ jest ono w oczywisty sposób równoważne zadaniu optymalizacji wypukłej

$$\begin{array}{lll} \text{minimize} & -f_0(\boldsymbol{x}) & \\ \text{subject to} & \boldsymbol{a}_i^{\mathrm{T}} \boldsymbol{x} = b_i, & i = 1, \ldots, m \\ & h_i(\boldsymbol{x}) \leqslant 0, & i = 1, \ldots, p \end{array} \tag{4}$$

Uwaga 2. Będziemy przyjmmować, że zadaniem optymalizacji wypukłej (lub krócej zadaniem wypukłym) jest zadanie optymalizacji wypukłej funkcji celu z wypukłymi funkcjami ograniczeń nierównościowych i *afinicznymi* funkcjami ograniczeń równościowych. Innymi słowy, aby nazwać zadanie wypukłym nie wystarczy aby polegało ono na minimalizacji funkcji wypukłej przy wypukłym zbiorze dopuszczalnym, lecz muszą dodatkowo być spełnione warunnki dotyczące postaci funkcji ograniczeń.

1.3 Programowanie liniowe

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". Omnia rerum principia parva sunt.

Przykład 1. Rozwiązać zadanie programowania liniowego

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

Zadanie (5) 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 (5) jest równoważne zadaniu

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

gdzie

$$
A = \begin{bmatrix} 1 & 1 \\ 1 & \frac{1}{4} \\ 1 & -1 \\ -\frac{1}{4} & -1 \\ -1 & -1 \\ -1 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 2 \\ 1 \\ 2 \\ 1 \\ -1 \\ 2 \end{bmatrix}, \quad c = \begin{bmatrix} 1 \\ \frac{1}{2} \end{bmatrix}
\tag{7}
$$

$$
A_{\mathrm{eq}} = \begin{bmatrix} 1 & \frac{1}{4} \end{bmatrix}, \quad b_{\mathrm{eq}} = \begin{bmatrix} \frac{1}{2} \end{bmatrix}
\tag{8}
$$

$$
x_{\mathrm{LB}} = \begin{bmatrix} -1 \\ -\frac{1}{2} \end{bmatrix}, \quad x_{\mathrm{UB}} = \begin{bmatrix} \frac{3}{2} \\ \frac{5}{4} \end{bmatrix}
\tag{9}
$$

Zadanie (6) jest w oczywisty sposób równoważne zadaniu

$$
\begin{aligned}
\text{minimize} \quad & c^{\mathrm{T}}x \\
\text{subject to} \quad & \begin{bmatrix} A \\ I \\ -I \end{bmatrix} x \leqslant \begin{bmatrix} b \\ x_{\mathrm{UB}} \\ -x_{\mathrm{LB}} \end{bmatrix} \\
& A_{\mathrm{eq}}x = b_{\mathrm{eq}}
\end{aligned}
\tag{10}
$$

Zatem możemy zdefiniować

$$
\tilde{A} = \begin{bmatrix} A \\ I \\ -I \end{bmatrix}, \quad \tilde{b} = \begin{bmatrix} b \\ x_{\mathrm{UB}} \\ -x_{\mathrm{LB}} \end{bmatrix},
\tag{11}
$$

i rozwiązać zadanie

$$
\begin{aligned}
\text{maximize} \quad & c^{\mathrm{T}}x \\
\text{subject to} \quad & \tilde{A}x \leqslant \tilde{b} \\
& A_{\mathrm{eq}}x = b_{\mathrm{eq}}
\end{aligned}
\tag{12}
$$

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.

1.3.1 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('-------------------------------')
```

1.3.2 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("--------------")
```

1.3.3 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

$$f_0 = 300x_1 + 500x_2 + 800x_3 \tag{13}$$

Ograniczenia są dwojakiego typu.

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

$$0.8x_1 + 0.3x_2 + 0.1x_3 \geqslant 0.3 \tag{13a}$$
$$0.01x_1 + 0.4x_2 + 0.7x_3 \geqslant 0.7 \tag{13b}$$
$$0.15x_1 + 0.1x_2 + 0.2x_3 \geqslant 0.1 \tag{13c}$$

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

$$x_1 \geqslant 0, \quad x_2 \geqslant 0, \quad x_3 \geqslant 0. \tag{14}$$

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

$$f_0(x) = f_{\text{costs}}(x) - f_{\text{income}}(x), \tag{15}$$

gdzie

$$x = \begin{bmatrix} x_{\text{LekI}} & x_{\text{LekII}} & x_{\text{SurI}} & x_{\text{SurII}} \end{bmatrix}^{\text{T}}, \tag{16}$$

jest wektorem zmniennych decyzyjnych (optymalizacyjnych),

$$\begin{aligned} f_{\text{costs}}(x) = 100.00x_{\text{SurI}} + 199.90x_{\text{SurII}} + \\ + 700.00x_{\text{LekI}} + 800.00x_{\text{LekII}} \end{aligned} \tag{17}$$

reprezentuje koszty zakupów oraz produkcji, zaś

$$f_{\text{income}}(x) = 6500.00x_{\text{LekI}} + 7100.00x_{\text{LekII}} \tag{18}$$

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

1. Bilans czynnika aktywnego

$$0.01x_{\text{SurI}} + 0.02x_{\text{SurII}} - 0.50x_{\text{LekI}} - 0.60x_{\text{LekII}} \geqslant 0. \tag{19}$$

2. Ograniczenia zasobów magazynowych (magazynowanie zakupionych surowców)

$$x_{\text{SurI}} + x_{\text{SurII}} \leqslant 1000. \tag{20}$$

3. Ograniczenia zasobów ludzkich

$$90.00x_{\text{LekI}} + 100.00x_{\text{LekII}} \leqslant 2000. \tag{21}$$

4. Ograniczenia zasobów sprzętowych

$$40.00x_{\text{LekI}} + 50.00x_{\text{LekII}} \leqslant 800. \tag{22}$$

5. Ograniczenia budżetowe

$$\begin{aligned} 100.00x_{\text{SurI}} + 199.90x_{\text{SurII}} + \\ + 700.00x_{\text{LekI}} + 800.00x_{\text{LekII}} \leqslant 100000. \end{aligned} \tag{23}$$

6. Ograniczenia zakresu zmiennych

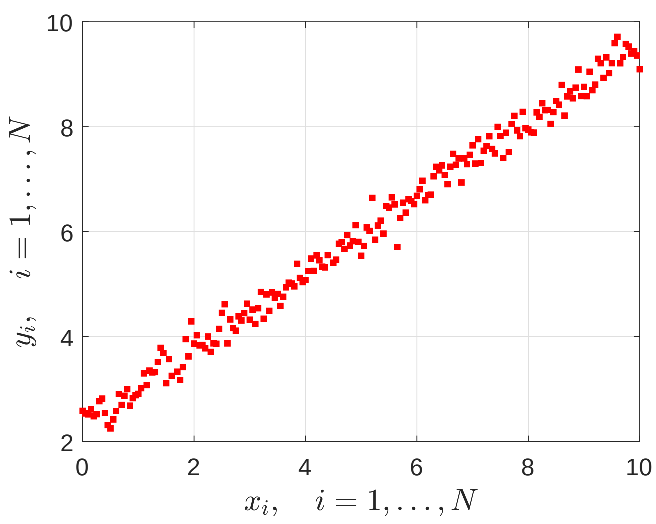
$$x_{\text{SurI}} \geqslant 0, \quad x_{\text{SurII}} \geqslant 0, \quad x_{\text{LekI}} \geqslant 0, \quad x_{\text{LekII}} \geqslant 0. \tag{24}$$

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.

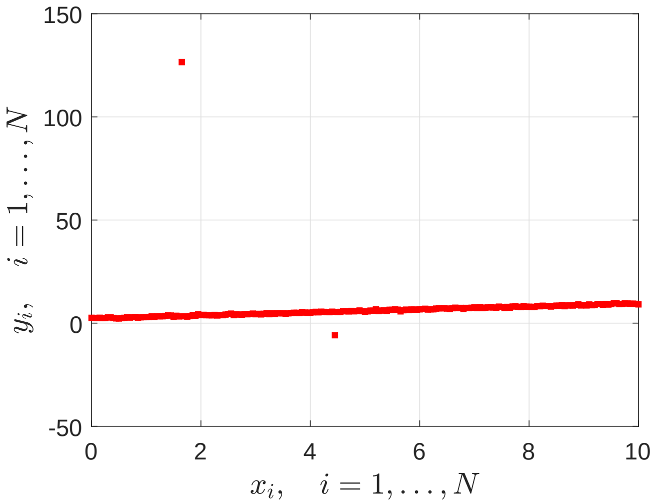
Rozwiązanie: $x^\star_{\text{LekI}} = 17.552$, $x^\star_{\text{LekII}} = 0$, $x^\star_{\text{SurI}} = 0$, $x^\star_{\text{SurII}} = 438.789$.

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

$$\left\{ \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}, \dots, \begin{bmatrix} x_N \\ y_N \end{bmatrix} \right\} \tag{25}$$



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

$$\phi(a, b) = \sum_{i=1}^{N} |ax_i + b - y_i| \, . \tag{26}$$

2. Suma kwadratów różnic

$$\psi(a, b) = \sum_{i=1}^{N} (ax_i + b - y_i)^2 \, . \tag{27}$$

Wprowadzając oznaczenia

$$\boldsymbol{\theta} = \begin{bmatrix} a \\ b \end{bmatrix}, \quad \boldsymbol{\Phi} = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_N & 1 \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \tag{28}$$

i odrobinę nadużywając notacji możemy napisać

$$\phi(\boldsymbol{\theta}) = \|\boldsymbol{\Phi\theta} - \boldsymbol{y}\|_1 \, , \quad \psi(\boldsymbol{\theta}) = \|\boldsymbol{\Phi\theta} - \boldsymbol{y}\|_2^2 \tag{29}$$

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

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

lub

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad \|\boldsymbol{\Phi\theta} - \boldsymbol{y}\|_2^2 \tag{31}$$

Rozwiązanie zadania optymalizacji (31) dane jest wzorem

$$\hat{\boldsymbol{\theta}} = \boldsymbol{\Phi}^{\dagger} \boldsymbol{y}, \tag{32}$$

gdzie $\boldsymbol{\Phi}^{\dagger}$ oznacza pseodoodwrotność Moore'a-Penrose'a (w środowisku Matlab© można ją wyznaczyć za pomocą polecenia `pinv`), natomiast (30) można sprawadzić do zadania LP (programowania liniowego) wprowadzając dodatkowe zmienne. W szczególności można pokazać, że zadanie (30) jest równoważne zadaniu LP

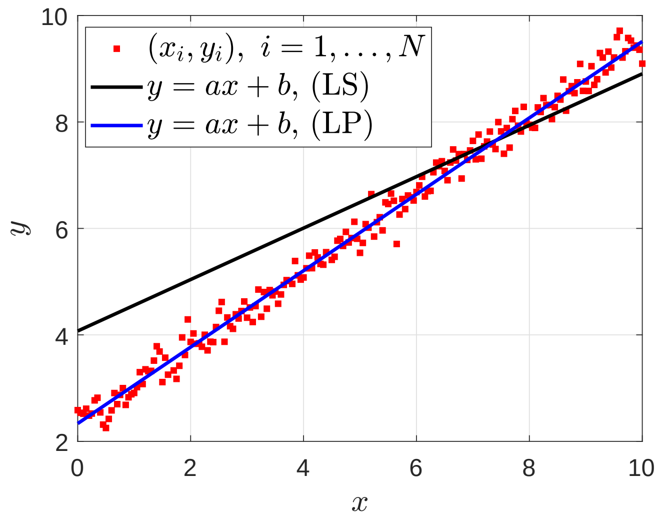
$$\begin{aligned} &\underset{\boldsymbol{\theta},\boldsymbol{\tau}}{\text{minimize}} && \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{1} \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} \boldsymbol{\theta} \\ \boldsymbol{\tau} \end{bmatrix} \\ &\text{subject to} && \begin{bmatrix} \boldsymbol{\Phi} & -\boldsymbol{I} \\ -\boldsymbol{\Phi} & -\boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{\theta} \\ \boldsymbol{\tau} \end{bmatrix} \leqslant \begin{bmatrix} \boldsymbol{y} \\ -\boldsymbol{y} \end{bmatrix} \end{aligned} \tag{33}$$

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

Polecenie

Pobrać plik danych `Data01.mat` (lub data01.csv) (ISOD). 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 (33). 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

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad \|\boldsymbol{\Phi\theta} - \boldsymbol{y}\|_1 \tag{34}$$

czyli

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \sum_{i=1}^{N} |\boldsymbol{\varphi}_i^{\mathrm{T}} \boldsymbol{\theta} - y_i| \, , \tag{35}$$

gdzie

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\varphi}_1^{\mathrm{T}} \\ \vdots \\ \boldsymbol{\varphi}_N^{\mathrm{T}} \end{bmatrix}, \tag{36}$$

jest równoważne zadaniu

$$\begin{aligned} &\underset{\boldsymbol{\theta},\boldsymbol{\tau}}{\text{minimize}} && \sum_{i=1}^{N} \tau_i \\ &\text{subject to} && |\boldsymbol{\varphi}_i^{\mathrm{T}} \boldsymbol{\theta} - y_i| \leqslant \tau_i, \quad i = 1, \ldots, N \end{aligned} \tag{37}$$

czyli

$$
\begin{aligned}
\underset{\boldsymbol{\theta},\boldsymbol{\tau}}{\text{minimize}} \quad & \sum_{i=1}^{N} \tau_i && (38)\\
\text{subject to} \quad & -\tau_i \leqslant \boldsymbol{\varphi}_i^{\mathrm{T}}\boldsymbol{\theta} - y_i \leqslant \tau_i, \quad i = 1, \ldots, N
\end{aligned}
$$

czyli

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

gdzie

$$
\boldsymbol{\tau} = \begin{bmatrix} \tau_1 \\ \vdots \\ \tau_N \end{bmatrix}, \tag{40}
$$

czyli

$$
\begin{aligned}
\underset{\boldsymbol{\theta},\boldsymbol{\tau}}{\text{minimize}} \quad & \mathbf{1}^{\mathrm{T}}\boldsymbol{\tau} && (41)\\
\text{subject to} \quad & \boldsymbol{\Phi}\boldsymbol{\theta} - \boldsymbol{y} \leqslant \boldsymbol{\tau}\\
& \boldsymbol{\Phi}\boldsymbol{\theta} - \boldsymbol{y} \geqslant -\boldsymbol{\tau}
\end{aligned}
$$

czyli

$$
\begin{aligned}
\underset{\boldsymbol{\theta},\boldsymbol{\tau}}{\text{minimize}} \quad & \mathbf{1}^{\mathrm{T}}\boldsymbol{\tau} && (42)\\
\text{subject to} \quad & \boldsymbol{\Phi}\boldsymbol{\theta} - \boldsymbol{\tau} \leqslant \boldsymbol{y}\\
& -\boldsymbol{\Phi}\boldsymbol{\theta} - \boldsymbol{\tau} \leqslant -\boldsymbol{y}
\end{aligned}
$$

czyli

$$
\begin{aligned}
\underset{\boldsymbol{\theta},\boldsymbol{\tau}}{\text{minimize}} \quad & \begin{bmatrix} \mathbf{0} \\ \mathbf{1} \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} \boldsymbol{\theta} \\ \boldsymbol{\tau} \end{bmatrix} && (43)\\
\text{subject to} \quad & \begin{bmatrix} \boldsymbol{\Phi} & -\boldsymbol{I} \\ -\boldsymbol{\Phi} & -\boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{\theta} \\ \boldsymbol{\tau} \end{bmatrix} \leqslant \begin{bmatrix} \boldsymbol{y} \\ -\boldsymbol{y} \end{bmatrix}
\end{aligned}
$$

przyjmując oznaczenia

$$
\boldsymbol{c} = \begin{bmatrix} \mathbf{0} \\ \mathbf{1} \end{bmatrix}, \qquad \boldsymbol{z} = \begin{bmatrix} \boldsymbol{\theta} \\ \boldsymbol{\tau} \end{bmatrix} \tag{44}
$$

$$
\boldsymbol{A} = \begin{bmatrix} \boldsymbol{\Phi} & -\boldsymbol{I} \\ -\boldsymbol{\Phi} & -\boldsymbol{I} \end{bmatrix} \qquad \boldsymbol{b} = \begin{bmatrix} \boldsymbol{y} \\ -\boldsymbol{y} \end{bmatrix} \tag{45}
$$

otrzymujemy

$$
\begin{aligned}
\underset{\boldsymbol{z}}{\text{minimize}} \quad & \boldsymbol{c}^{\mathrm{T}}\boldsymbol{z} && (46)\\
\text{subject to} \quad & \boldsymbol{A}\boldsymbol{z} \leqslant \boldsymbol{b}
\end{aligned}
$$

Pierwsze dwa elementy rozwiązania $z^{\star}$ zadania (46) są poszukiwanymi współczynnikami a i b prostej trendu.

Rysunek 4: Błędów grubych należy unikać – nie tylko utrudniają wyznaczenie modelu na podstawie pomiarów, ale mogą być również powodem nieprzychylnych komentarzy. [`https://demotywatory.pl`]

3 Forma sprawozdania

Wszystkie pliki związane z pojedynczym zadaniem należy umieścić w folderze o nazwie `zadanie<nr zadania>`. Następnie wszystkie foldery umieszczamy w jednym folderze nadrzędnym, o nazwie `cwiczenie<nr ćwiczenia>`, kompresujemy do pliku `.zip`, który następnie umieszczamy w ISOD, za pomocą odpowiedniej bramki. Każdy folder musi być „autonomiczny”, tzn. po uruchomieniu skryptu wszystko musi się wykonać, nie mogą pojawiać się jakieś komunikaty, że brakuje danych czy tego typu. Wystąpienie tego typu problemów będzie powodować obniżenie oceny z zadania lub nawet brak zaliczenia danego zadania. Oddanie zadań z danego ćwiczenia po wyznaczonym terminie również będzie skutkować obniżeniem punktacji.
Bardzo proszę o przestrzeganie podanych zasad, szczególnie dotyczących nazw folderów i plików. Proszę w nazwach tych nie używać spacji, czy polskich liter w rodzaju „ą” lub „ł”, proszę również nie zaczynać ich cyfrą.
Generalnie, o ile nie zostanie wyraźnie powiedziane inaczej, nie ma potrzeby przygotowania formalnych sprawozdań. Wystarczą skrypty, jednak kod musi być czytelny, dobrze skomentowany (ale nie na siłę). Muszą się generować odpowiednie rysunki (również z opisem, tzn. opisem osi, legendą, tytułem). Oczywiście opis zależy od ryunku i nie zawsze potrzebna jest, przykładowo, legenda. Należy podchodzić do tego ze zdrowym rozsądkiem.

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 uncer-

tain 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.

---

4 Metody sympleksowe versus metdy puntu wewnętrznego - rys historyczny

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

PROGRAMOWANIE LINIOWE W CZASIE WIELOMIANOWYM

Sympleks nie jest algorytmem wielomianowym. Pewne rzadkie rodzaje zadań programowania liniowego powodują, że przechodzi z jednego rogu obszaru dopuszczalnego do lepszego rogu, potem do jeszcze lepszego i tak dalej, wykonując wykładniczą liczbę kroków. Przez długi czas programowanie liniowe uważano za paradoks – problem, który można rozwiązać w praktyce, ale nie w teorii!

Wtedy, w 1979 roku, młody radziecki matematyk Leonid Chaczijan opracował *metodę elipsoidalną* (ang. *ellipsoid method*), która jest całkowicie różna od sympleksu, jest ekstremalnie prosta koncepcyjnie (lecz trudna do udowodnienia) i w dodatku umożliwia rozwiązanie dowolnego zadania programowania liniowego w czasie wielomianowym. Zamiast szukać rozwiązania, przechodząc od jednego wierzchołka wielościanu do następnego, algorytm Chaczijana ogranicza je do coraz mniejszych elipsoid (spłaszczonych wielowymiarowych kul). Kiedy ogłoszono wynalezienie tego algorytmu, stał się on czymś w rodzaju „matematycznego Sputnika", osiągięciem, które w dobie zimnej wojny przestraszyło władze USA możliwością wyższości naukowej Związku Radzieckiego. Algorytm elipsoidalny oznaczał ważny postęp teoretyczny, ale w praktycznych zastosowaniach nie konkurował z metodą sympleks. Paradoks programowania liniowego się pogłębił: problem z dwoma algorytmami, jednym efektywnym w teorii, a drugim w praktyce!

Kilka lat później Narendra Karmarkar, doktorant w UC Berkeley, wpadł na zupełnie inny pomysł, który doprowadził do innego wielomianowego algorytmu dla programowania liniowego. Algorytm Karmarkara znany jest jako *metoda punktu wewnętrznego* (ang. *interior-point method*), ponieważ znajduje optymalny wierzchołek nie tak jak sympleks, przeskakując z wierzchołka do wierzchołka po powierzchni wielościanu, lecz przebijając się sprytną ścieżką przez wnętrze wielościanu. I w praktyce działa on szybko.

Być może jednak największy postęp w programowaniu liniowym spowodowany był nie przełomem teoretycznym Chaczijana, ani nie nowym podejściem Karmarkara, lecz niespodziewaną konsekwencją tego drugiego: zaciekła konkurencja między dwoma podejściami, metodą sympleks i metodą punktu wewnętrznego, doprowadziła do opracowania bardzo szybkiego kodu dla programowania liniowego.