Programowanie liniowe

Rozwiązywanie problemów za pomocą funkcji linprog

Funkcja linprog w MATLAB pozwala rozwiązywać problemy programowania liniowego w postaci:

$$\begin{aligned} \min_{x} \quad & f^T x \\ \text{subject to} \quad & Ax \leq b \\ & A_{eq}x = b_{eq} \\ & lb \leq x \leq ub \end{aligned}$$

Gdzie:

- **x**: Wektor zmiennych decyzyjnych
- **f**: Wektor współczynników liniowej funkcji kosztu (funkcji celu)
- **A, b**: Macierz i wektor ograniczeń nierównościowych
- **Aeq, beq**: Macierz i wektor ograniczeń równościowych
- **lb, ub**: Górna i dolna granica zmiennych decyzyjnych x

Celem jest znalezienie wektora x, który zapewnia minimalną wartość funkcji kosztu respektując założone ograniczenia.

Rozważmy prosty przykład żeby zilustrować znaczenie każdego z symboli:

$$\begin{aligned} \min_{x} \quad & 2x_1 + 3x_2 \\ \text{subject to} \quad & x_1 + x_2 \leq 10 \\ & 2x_1 + x_2 \leq 15 \\ & x_1, x_2 \geq 0 \end{aligned}$$

W tym przykładzie:

- **x = [x_1, x_2]**: Zmienne decyzyjne (wektor dwuelementowy)

- **f = [2, 3]**: Współczynniki funkcji kosztu ($2x_1 + 3x_2$)
- **A = [1 1; 2 1]**: Macierz ograniczeń nierównościowych
- **b = [10; 15]**: Wektor ograniczeń nierównościowych
- **lb = [0; 0]**: Dolna granica dla x_1 i x_2

Nierówność $Ax \leq b$ może zostać zapisana jako:

$$\begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} 10 \\ 15 \end{bmatrix}$$

Powyższy zapis może zostać rozbity:

$$\begin{aligned} x_1 + x_2 &\leq 10 \\ 2x_1 + x_2 &\leq 15 \end{aligned}$$

Powyższy przykład nie zawiera ograniczeń równościowych (Aeq i beq), ale reprezentuje się je w podobny sposób co A i b.

W celu rozwiązania tego problemu z wykorzystaniem funkcji linprog, można wykorzystać program:

```
% Define the problem
f = [2; 3];  % Objective function coefficients
A = [1 1; 2 1];  % Inequality constraint matrix
b = [10; 15];  % Inequality constraint vector
lb = [0; 0];  % Lower bounds

% Solve the problem
[x, fval] = linprog(f, A, b, [], [], lb);

% Display the results
disp('Optimal solution:');
disp(x);
disp('Optimal objective value:');
disp(fval);
```

Warto zauważyć, że podanie Aeq i beq nie jest konieczne, ze względu na brak ograniczeń równościowych w tym zadaniu. Puste wektory ($[\,]$) reprezentują te argumenty.

Po uruchomieniu programu, MATLAB rozwiąże problem i wypisze optymalne wartości x_1 i x_2, oraz wartość funkcji kosztu.

Problem dualny w programowaniu liniowym

W programowaniu liniowym, każdy problem (nazywany problemem pierwotnym) ma swój odpowiednik w postaci problemu dualnego.

Problem dualny pozwala przyjrzeć się problemowi z innego punktu widzenia, co może prowadzić do cennych spostrzerzeń.

Dla zadania *Stoisko z owocami*, problem dualny wygląda następująco:

$$\begin{aligned} \max_{y} \quad & 100y_1 + 80y_2 - 20y_3 - 20y_4 \\ \text{subject to} \quad & y_1 + 0.5y_2 - y_3 \leq 0.5 \\ & y_1 + 0.75y_2 - y_4 \leq 0.75 \\ & y_1, y_2, y_3, y_4 \geq 0 \end{aligned}$$

Gdzie:

- $\mathbf{y_1}$ odpowiada ograniczeniu ilości owoców
- $\mathbf{y_2}$ odpowiada ograniczeniu budżetu
- $\mathbf{y_3}$ i $\mathbf{y_4}$ odpowiadają ograniczeniu minimalnej ilości zakupionych owoców

Cel tego problemu jest znalezienie najbardziej korzystnych cen które możnaby przypisać do owoców z zadania pierwotnego.

Metody rozwiązywania zadań wykorzystywane przez funkcję linprog

Funkcja linprog w MATLAB wykorzystuje różne algorytmy do rozwiązywania zadań. Wybór algorytmu zależy od rozmiaru i struktury problemu. Głównie wykorzystywane są metody:

- **Dual Simplex Algorithm**: Domyślna metoda. Wydajna dla problemów o średniej i dużej złożoności.
- **Interior Point Algorithm**: Metoda szczególnie wydajna dla problemów o dużej złożoności.

Funkcja linprog jest w stanie automatycznie dobrać metodę rozwiązywania zadania w zależności od charakterystyki danego problemu. Użytkownik może jednak narzucić konkretną metodę wykorzystując argument *optimoptions* funkcji linprog.

Działanie algorytmu *Dual Simplex*

Kroki algorytmu:

1. Rozpocznij od rozwiązania dualnego

Wyobraź sobie, że próbujesz rozwiązać zagadkę, ale zaczynasz rozwiązywać ją od gotowej odpowiedzi! Podobnie działa algorytm *dual simplex*. Zaczyna od rozwiązania, które spełnia ograniczenia problemu dualnego, ale może nie być optymalne dla oryginalnego (pierwotnego) problemu.

2. Sprawdź optymalność

Algorytm analizuje obecne rozwiązanie, i sprawdza czy jest to najlepsze rozwiązanie jakie możemy uzyskać.

3. Jeśli nie jest optymalne, wprowadź poprawki

Jeśli obecne rozwiązanie nie jest najlepsze, algorytm wprowadza zmiany:

- Identyfikuje, która część rozwiązania wymaga zmiany (nazywana „zmienną wychodzącą”).
- Następnie ustala, jak ją zmienić (poprzez wybór „zmiennej wchodzącej”).
- Zmiany te są wprowadzane w sposób, który zawsze utrzymuje problem dualny w stanie wykonalnym.

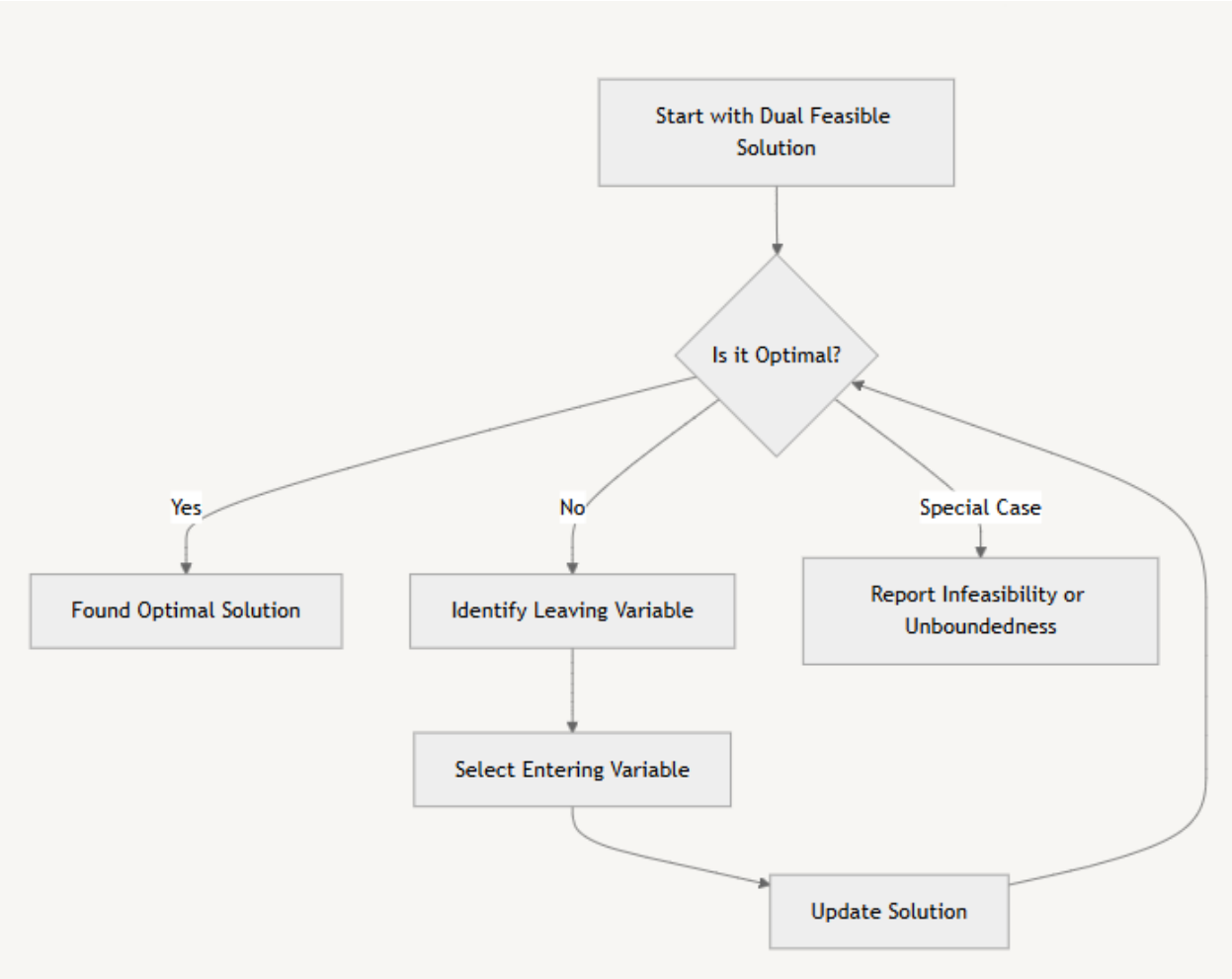
4. Repeat

Algorytm wciąż wprowadza te poprawki, krok po kroku, aż osiągnie najlepsze rozwiązanie.

5. Przypadki szczególne

Czasami problem może nie mieć rozwiązania lub może mieć nieskończoną liczbę rozwiązań. Algorytm jest w stanie rozpoznać takie przypadki szczególne i je zgłosić.

Diagram algorytmu



Ten diagram pokazuje, jak algorytm *dual simplex* przechodzi przez swoje kroki, stale poprawiając rozwiązanie, aż osiągnie punkt optymalny lub zidentyfikuje przypadek szczególny.

- Przykład: Stoisko z owocami

  Pamiętasz nasz problem ze straganem owocowym? Zobaczmy, jak algorytm dualnego simplex mógłby do niego podejść:

  Wykorzystując algorytm *dual simplex* zadanie mogłoby zostać wykonane w następujący sposób

  1. Rozpocznij od założenia, które spełnia wszystkie ograniczenia, ale może nie maksymalizować zysku.
  2. Sprawdź, czy to założenie przynosi najwyższy możliwy zysk.
  3. Jeśli nie, dostosuj liczbę jabłek lub pomarańczy do kupienia.
  4. Kontynuuj dostosowywanie, aż do momentu znalezienia idealnego balansu jabłek i pomarańczy.
  5. Jeśli nie było sposobu na spełnienie wszystkich ograniczeń (np. zbyt mały budżet), algorytm szybko to zidentyfikuje.

Jak działa algorytm punktów wewnętrznych

Kroki algorytmu:

1. Rozpocznij od wnętrza

W przeciwieństwie do metody simplex, która porusza się wzdłuż krawędzi, metoda punktów wewnętrznych zaczyna od punktu wewnątrz obszaru możliwych rozwiązań.

2. Poruszaj się w optymalnym kierunku

Algorytm podejmuje następnie kroki w kierunku rozwiązania optymalnego:

- Oblicza kierunek, który poprawia funkcję celu.
- Porusza się w tym kierunku, ale nigdy nie dociera całkowicie do granicy.

3. Funkcja bariery

Aby pozostać wewnątrz obszaru możliwych rozwiązań, algorytm używa "funkcji bariery". Ta funkcja działa jak ściana, która zapobiega uderzeniu rozwiązania w granice:

- Gdy rozwiązanie zbliża się do ograniczenia, funkcja bariery gwałtownie rośnie.
- To utrzymuje rozwiązanie ściśle wewnątrz obszaru możliwych rozwiązań.

4. Kolejne iteracje

Algorytm powtarza ten proces, z każdym krokiem zbliżając się do rozwiązania optymalnego:

- Stopniowo zmniejsza wpływ funkcji bariery.
- To pozwala rozwiązaniu zbliżyć się do granic, gdzie często leży punkt optymalny.

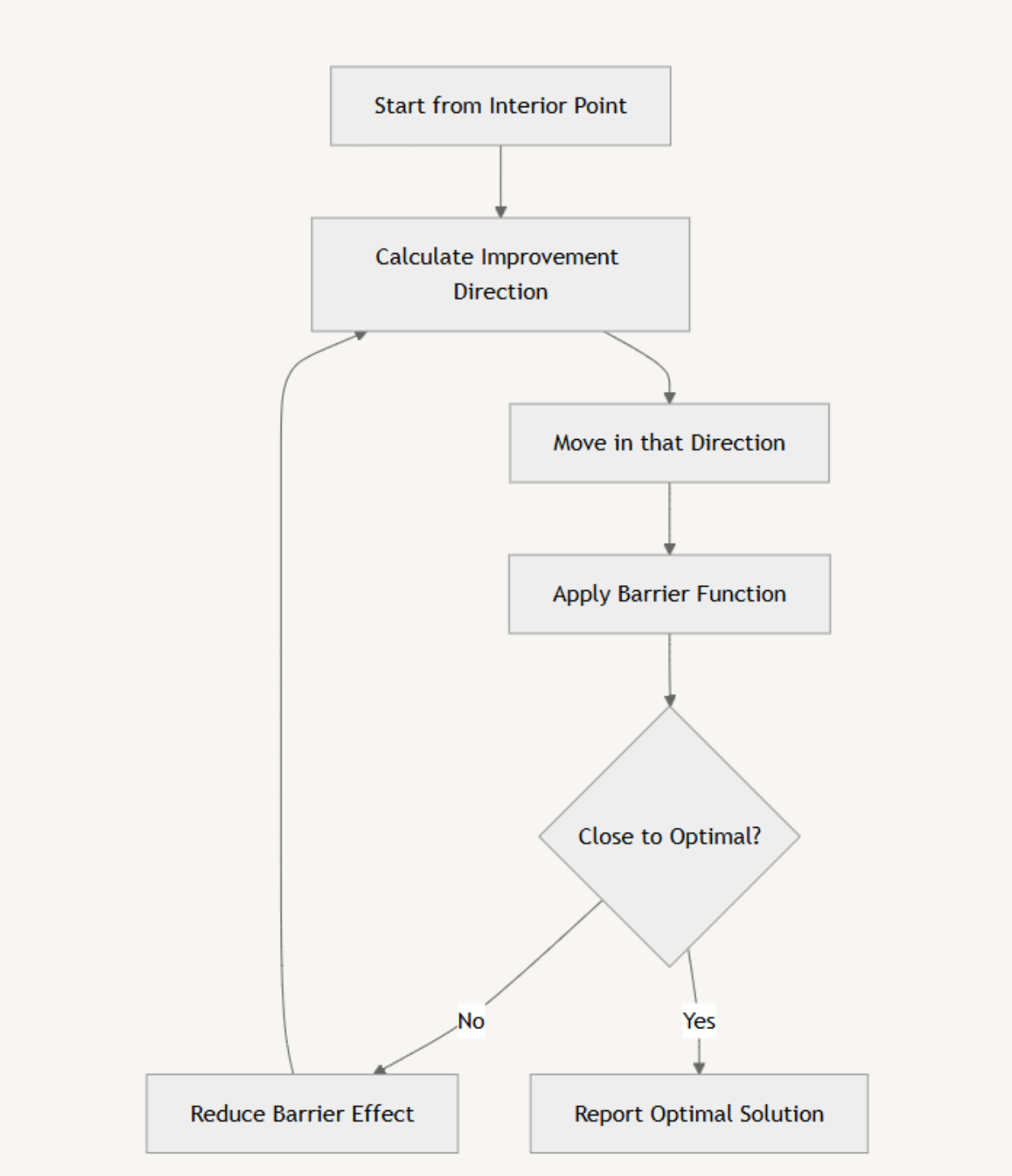
5. Zbieżność

Proces trwa, aż rozwiązanie będzie bardzo bliskie optymalnego:

- Algorytm sprawdza, czy obecne rozwiązanie jest wystarczająco bliskie prawdziwemu optimum.
- Jeśli tak, algorytm zatrzymuje się i raportuje to rozwiązanie.

> 💡 Kluczowa zaleta: Metoda punktów wewnętrznych jest szczególnie wydajna w przypadku problemów o dużej skali. Często wymaga mniej iteracji niż metoda simplex, zwłaszcza w przypadku problemów z wieloma zmiennymi.

Diagram algorytmu



Ten diagram ilustruje, jak algorytm punktów wewnętrznych porusza się po obszarze możliwych rozwiązań, zawsze pozostając wewnątrz, jednocześnie zbliżając się do rozwiązania optymalnego.

- Przykład: Stoisko z owocami w 3D

Rozszerzmy nasz problem ze straganem owocowym, aby uwzględnić trzeci produkt, przykładowo banany:

1. Rozpocznij od punktu wewnątrz obszaru możliwych rozwiązań (np. kupując trochę każdego owocu).
2. Oblicz, w którym kierunku (więcej jabłek? więcej pomarańczy? więcej bananów?) zysk wzrasta najbardziej.
3. Poruszaj się w tym kierunku.
4. Powtarzaj ten proces, z każdym krokiem zbliżając się do optymalnego rozwiązania.
5. Zatrzymaj się, gdy rozwiązanie będzie wystarczająco dobre.