131890-S Metody Optymalizacji **Letni 2023/2024**

Zajęcia 7: Algorytm sympleksowy

Daniel Kaszyński

7.1 Programowanie liniowe

Programowanie liniowe jest dziedziną optymalizacji umożliwiającą rozwiązywanie najprostszych problemów optymalizacyjnych - problemów liniowych. Jest to metoda uzyskująca optymalne wyniki dla modeli liniowych:

- Liniowych funkcji celu,
- Liniowych ograniczeń równości oraz nierówności.

Jest szeroko stosowany w różnych dziedzinach, takich jak badania operacyjne, ekonomia, inżynieria i nauki o zarządzaniu, do skutecznego rozwiązywania problemów z alokacją, planowaniem i przydziałem zasobów.

Zbiór możliwych rozwiązań jest zbiorem wypukłym (określonym przez skończony zbiór przecinających się półprzestrzeni - każda określona przez nierówność liniową).

Funkcja celu jest funkcją afiniczną zmiennej rzeczywistej określonej na tym wielościanie. Programowanie liniowe umożliwia znalezienie punktu na tym wielościanie, który daje najmniejszą (lub największą) wartość funkcji celu. W programowaniu liniowym wszystkie funkcje celu i ograniczenia są liniowe. Funkcja celu reprezentuje ilość, którą należy zoptymalizować, natomiast ograniczenia reprezentują istniejące ograniczenia lub warunki, które muszą zostać spełnione. Celem jest znalezienie wartości zmiennych decyzyjnych, które optymalizują funkcję celu, spełniając jednocześnie wszystkie ograniczenia.

Ogólna postać problemu programowania liniowego jest następująca:

- Znajdź wektor x
- który maksymalizuje $c^T x$
- pod warunkiem że $Ax \leqslant b$
- oraz $x \geqslant 0$

gdzie wektor x jest wektorem którego składowe mają zostać określone, c jest podanym wektorem wartości, które chcemy zoptymalizować, b jest danym wektorem elementów, do których się ograniczamy oraz A to macierz zmiennych obok ograniczeń. Funkcję $c^T x$, której wartość ma być maksymalizowana, nazywamy funkcją celu. Wypukły wielobok, nad którym ma zostać zoptymalizowana funkcja celu, jest konstruowany przy uwzględnieniu ograniczeń $Ax \leqslant b$ oraz $x \geqslant 0$.

7.2 Algorytm sympleksowy

7.2.1 Podstawy algorytmu

Algorytm sympleksowy jest szeroko stosowaną metodą rozwiązywania problemów programowania liniowego. Jest często uważany za jeden z najskuteczniejszych algorytmów rozwiązywania problemów liniowych

w praktyce.

Algorytm simpleks działa poprzez iteracyjne przechodzenie od jednego wierzchołka (punktu narożnego) dopuszczalnego obszaru do drugiego wzdłuż krawędzi wieloboku określonego przez ograniczenia, aż do osiągnięcia optymalnego rozwiązania. W każdym kroku algorytm wybiera element kluczowy, w celu poprawy wartość funkcji celu i przechodzi do sąsiedniego wierzchołka odpowiadającego elementowi kluczowemu.

Algorytm sympleksowy wykorzystuje standardową formę do reprezentowania nierówności. Nierówności są zatem konwertowane na równości zawierające dodatkowe **zmienne swobodne**.

Przykład 1. Niech $f(x, y, z) = -x + 3y + 2z$ którą chcemy zmaksymalizować pod następującymi warunkami:

- $x + y + z \leqslant 6$
- $x + z \leqslant 4$
- $y + z \leqslant 3$
- $x + y \leqslant 2$

zakładając, że $x, y, z \geqslant 0$. Ograniczające nierówności reprezentowane w postaci standardowej przy użyciu zmiennych swobodnych wyglądałyby następująco:

- $x + y + z + r + s + t + u = 6$
- $x + z + r + s + t + u = 4$
- $y + z + r + s + t + u = 3$
- $x + y + r + s + t + u = 2$

natomiast funkcja f równała by się $-x + 3y + 2z + r + s + t + u$.

7.2.2 Tabela sympleksowa

Algorytm sympleksowy często wykorzystuje reprezentację analizowanego problemu w postaci **tabeli sympleksowej**.

Zakładając, że funkcja f, którą chcemy maksymalizować, jest równa $-x+3y+2z$ i maksymalizujemy ją pod następującymi warunkami:

- $x + y + z \leqslant 6$
- $x + z \leqslant 4$
- $y + z \leqslant 3$
- $x + y \leqslant 2$
- $x, y, z \geqslant 0$

przykładowa tabela sympleksowa dla tego przypadku wyglądałaby tak:

Tabela 7.1: Przykładowa tabela sympleksowa

x	y	z	r	s	t	u	
1	1	1	1	0	0	0	6
1	0	1	0	1	0	0	4
0	1	1	0	0	1	0	3
1	1	0	0	0	0	1	2
-1	3	2	0	0	0	0	0

W utworzonej tabeli obszar zaznaczony na żółto to macierz A - macierz wartości obok ograniczeń nierówności. W dolnym wierszu widzimy współczynniki funkcji celu. Kolumna po prawej stronie tabeli zawiera wartości, do których ograniczają nierówności - b (wartości po prawej stronie nierówności). Na koniec mamy także 4 zmienne swobodne, po jednej dla każdego ograniczenia. Macierz jednostkowa dla tych zmiennych oznaczona jest w utworzonej tabeli kolorem różowym.

Jeżeli kolumna tabeli odpowiadająca danej zmiennej zawiera same zera i jedną jedynkę, to dana zmienna będzie niezerowa. W przeciwnym razie zmienna ta będzie wynosić zero.

Początkowa tabela simpleksowa jest skonstruowana w taki sposób, że zmienne x, y, z są równe zeru, a zmienne luzu są niezerowe i należą do bazy.

7.2.3 Metoda eliminacji Gaussa

Kolejnym krokiem w algorytmie simpleksowym jest zastosowanie algorytmu eliminacji Gaussa.

Metoda eliminacji Gaussa jest algorytmem stosowanym do rozwiązywania układów równań liniowych poprzez transformację rozszerzonej macierzy reprezentującej układ do postaci schodkowej poprzez serię elementarnych operacji na wierszach. Jest to podstawowa technika algebry liniowej i posiada wiele zastosowań, takich jak rozwiązywanie układów równań liniowych, obliczanie odwrotności macierzy oraz znajdowanie wartości własnych i wektorów własnych. Jest szeroko stosowaną metodą rozwiązywania problemów algebraicznych liniowych, ponieważ jest wydajna i stabilna.

Pierwszym krokiem eliminacji Gaussa jest wybranie kolumny (zmiennej) o największej wartości w dolnym wierszu tabeli. Jeśli kilka zmiennych posiada największą wartość, możemy wybrać dowolną z nich. Wybrana zmienna będzie nową zmienną podstawową wchodzącą do bazy i stanie się *kolumną kluczową*. Kontynuując przykład z podrozdziału o tabeli symplexowej, największy wpływ posiada zmienna y (ponieważ jest mnożona aż przez 3).

Następnie musimy określić, która zmienna swobodna zostanie zastąpiona wybraną zmienną. W tym celu należy obliczyć stosunek kolumny b (kolumny znajdującej się najbardziej na prawo) do wybranej zmiennej. Na szczęście wszystkie wartości kolumny y w naszym przypadku wynoszą 1 lub 0. Powinniśmy wybrać wiersz z **minimalnym stosunkiem**, czyli w tym przypadku ostatnie z ograniczeń. Wiersz z minimalnym stosunkiem nazywa się *wierszem kluczowym*.

Tabela 7.2: Wiersz i kolumna kluczowa w przykładowej tabeli sympleksowej

x	y	z	r	s	t	u	
1	1	1	1	0	0	0	6
1	0	1	0	1	0	0	4
0	1	1	0	0	1	0	3
1	1	0	0	0	0	1	2
-1	3	2	0	0	0	0	0

Następnym krokiem jest wykonanie szeregu elementarnych operacji na wierszach, tak aby kolumna kluczowa stała się kolumną jednostkową z 1 na przecięciu z wierszem kluczowym (na *elemencie kluczowym*). Aby to osiągnąć, musimy najpierw podzielić wszystkie elementy wiersza kluczowego przez element kluczowy (w naszym przykładzie element kluczowy jest równy 1, więc wiersz kluczowy pozostaje taki sam). Następnie dzielimy pozostałe wiersze przez wiersz kluczowy, aby uzyskać zera w kolumnie kluczowej.

Tabela 7.3: Przykładowa tabela sympleksowa po pierwszej iteracji algorytmu

x	y	z	r	s	t	u	
0	0	1	1	0	0	-1	4
1	0	1	0	1	0	0	4
-1	0	1	0	0	1	-1	1
1	1	0	0	0	0	1	2
-4	0	2	0	0	0	-3	-6

Te kroki są powtarzane, aż wszystkie wartości w dolnym wierszu będą równe lub mniejsze od zera. Zapewnia to o tym, iż nie ma możliwości dalszej poprawy wartości analizowanego rozwiązania.

Tabela 7.4: Przykładowa tabela sympleksowa po wszystkich iteracjach algorytmu

x	y	z	r	s	t	u	
1	0	0	1	0	-1	0	3
2	0	0	0	1	-1	1	3
-1	0	1	0	0	1	-1	1
1	1	0	0	0	0	1	2
-2	0	0	0	0	-2	-1	-8

Finalne rozwiązanie w tym przykładzie to $x = 0$, $y = 2$ i $z = 1$. Zmienna x wynosi zero, ponieważ nie jest kolumną jednostkową, natomiast wartości innych zmiennych można odczytać z kolumny znajdującej się najbardziej na prawo.

7.2.4 Implementacja algorytmu sympleksowego

Aby zaimplementować algorytm simpleksowy, musimy najpierw przygotować funkcję odpowiedzialną za tworzenie tabeli sympleksowej. Odczyta ona najpierw liczbę podanych zmiennych, a następnie przygotuje macierz wartości dla początkowej tabeli sympleksowej. Przykładowa implementacja tej funkcji przy użyciu języka programowania **R** mogłaby wyglądać następująco:

```
create_simplex_table <- function(A_mat, b_vec, c_vec){
  # Odczytaj liczbe zmiennych/ograniczen
  n_var <- length(c_vec)
  n_con <- length(b_vec)

  # Skonstruuj tabele Simplex
  st <- matrix(0,
               nrow = n_con+1,
               ncol = n_var+n_con+1)

  st[1:n_con, 1 : n_var] <- A_mat
  st[nrow(st), 1:n_var] <- c_vec
  st[1:n_con, ncol(st)] <- b_vec
  st[1:n_con, (n_var+1): (n_var+n_con)] <- diag(n_con)
  return(st)
```

```
}
```

Listing 1: Implementacja funkcji odpowiedzialnej za utworzenie tabeli sympleksowej

Algorytm simpleksowy wymaga również implementacji metody eliminacji Gaussa. Pojedynczy krok tej metody będzie polegał na wybraniu kolumny kluczowej i wiersza kluczowego oraz wykorzystaniu ich do przekształcenia wybranej kolumny w kolumnę jednostkową. Warto zaznaczyć, że w tym celu możemy wykorzystać wbudowaną metodę języka **R** - *outer()*. Umożliwia ona utworzenie nowej macierzy lub tablicy poprzez zastosowanie podanej jako parametr funkcji do każdej możliwej kombinacji elementów z dwóch wektorów wejściowych (gdzie domyślną funkcją jest mnożenie). Implementację algorytmu eliminacji Gaussa można stworzyć w następujący sposób:

```
gaussian_elimination <- function(st, b_vec, c_vec){
  # Odczytaj liczbe zmiennych/ograniczen
  n_var <- length(c_vec)
  n_con <- length(b_vec)

  # Wybierz id kolumny
  i_col <- which.max(st[nrow(st), 1 : n_var])
  if(st[nrow(st), i_col] <= 0){
    return(TRUE)
  }

  # Wybierz id wiersza
  temp <- st[1:n_con,i_col]
  temp <- st[1:n_con,ncol(st)] / temp
  i_row <- which.min(temp)

  # Eliminacja Gaussa
  temp <- st[i_row, ] / st[i_row, i_col]
  st <- st - outer(st[, i_col], st[i_row, ])
  st[i_row, ] <- temp
  return(st)
}
```

Listing 2: Implementacja metody eliminacji Gaussa

Oczywiście na końcu algorytmu chcielibyśmy także móc odczytać wyniki naszych obliczeń. Pomocną może okazać się metoda odczytu wyników w oparciu o uzyskaną tabelę sympleksową. Przykładowa metoda tego typu może wyglądać następująco:

```
read_results <- function(st, b_vec, c_vec){
  # Odczytaj wyniki
  n_var <- length(c_vec)
  n_con <- length(b_vec)

  x_opt <- rep(NA, n_var)
  for(i in 1 : n_var){
    if((sum(st[,i])==1) & (max(st[,i])==1) & (min(st[,i])==0)){
      id <- which(st[,i]==1)
      x_opt[i] <- st[id, ncol(st)]
    }else{
      x_opt[i] <- 0
    }
  }
  out <- list(x_opt = x_opt,
              f_opt = sum(x_opt * c_vec))
  return(out)
}
```

Listing 3: Implementacja funkcji odczytującej wynik algorytmu

Mając pod ręką wszystkie niezbędne implementacje funkcji, możemy przystąpić do implementacji samego algorytmu. Algorytm simpleksowy powinien składać się z następujących kroków:

- utworzenie początkowej tabeli sympleksowej,
- powtarzanie iteracji metody eliminacji Gaussa, aż wszystkie wartości w dolnym wierszu będą równe lub mniejsze od zera,
- alternatywnie algorytm eliminacji Gaussa powinien zakończyć się po określonej z góry maksymalnej liczbie kroków k,
- odczytanie wyników.

Ostateczna implementacja algorytmu wygląda następująco:

```
simplex_algorithm <- function(A_mat, b_vec, c_vec){
  # Utworz tabele Simplex
  st <- create_simplex_table(A_mat, b_vec, c_vec)

  # Eliminacja Gaussa
  k <- 1
  while(TRUE){
    temp <- gaussian_elimination(st, b_vec, c_vec)
    if((length(temp) == 1) || (k >= 100)){
      break
    }
    st <- temp
    k <- k +1
  }

  # Odczytaj wyniki
  out <- read_results(st, b_vec, c_vec)
  return(out)
}
```

Listing 4: Implementacja algorytmu sympleksowego