Metody optymalizacji nieliniowej w R
Podstawy teoretyczne i zastosowania ekonomiczne

Bogumił Kamiński, Grzegorz Koloch, Michał Lewandowski

29 lipca 2014

Spis treści

Oznaczenia matematyczne

$\mathbb{R}$	zbiór liczb rzeczywistych;
$\mathbb{R}^n$	n-wymiarowa przestrzeń rzeczywista;
$[a, b]$	przedział domknięty o końcach a i b;
(a, b)	przedział otwarty o końcach a i b;
$\mathbf{x}$	wektor kolumnowy w $\mathbb{R}^n$;
x_i	i-ty element wektora $\mathbf{x}$;
$\mathbf{x}(k)$	wartość wektora $\mathbf{x}$ w k-tej iteracji algorytmu optymalizacyjnego;
$\mathbf{x}^s$	dla ze zbioru wektorów indeksowanych zbiorem S, wektor o indeksie $s \in S$;
$\mathbb{R}^n_+$	n-wymiarowa przestrzeń rzeczywista taka, że $x_i \geq 0$;
$\mathbb{R}^n_{++}$	n-wymiarowa przestrzeń rzeczywista taka, że $x_i > 0$;
$\mathbf{0}, \mathbf{1}$	odpowiednio wektory kolumnowe założone z 0 i 1 w $\mathbb{R}^n$;
$\|\|\mathbf{x}\|\|$	norma wektora $\mathbf{x}$ równa $\sqrt{\sum_{i=1}^n x_i^2}$;
$\mathcal{K}(\mathbf{x}, r)$	kula otwarta o środku w punkcie $\mathbf{x}$ i promieniu r: $\{\mathbf{y} : \|\|\mathbf{x} - \mathbf{y}\|\| < r\}$;
A	macierz;
$A_{i,j}$	element macierzy A w i-tym wierszu i j-tej kolumnie;
A^T	transpozycja macierzy A;
A^{-1}	odwrotność macierzy nieosobliwej A;
$\det A$	wyznacznik macierzy kwadratowej A;
f', f''	odpowiednio pierwsza i druga pochodna funkcji $f\colon \mathbb{R} \to \mathbb{R}$;
$f^{(n)}$	n-ta pochodna funkcji $f\colon \mathbb{R} \to \mathbb{R}$;
$\nabla f(\mathbf{x})$	gradient funkcji $f\colon \mathbb{R}^n \to \mathbb{R}$ w punkcie $\mathbf{x}$;
$\nabla^2 f(\mathbf{x})$	hesjan (macierz drugich pochodnych cząstkowych) funkcji $f\colon \mathbb{R}^n \to \mathbb{R}$ w punkcie $\mathbf{x}$;
$f(x) = \mathcal{o}(g(x))$	notacja oznaczająca, że $f(x)$ zbiega do 0 szybciej niż $g(x)$; por. definicja 1.5;
$[\ell]$	jeżeli ℓ jest zdaniem logicznym to: $[\ell] = \begin{cases} 1 & \text{jeżeli } \ell \text{ jest prawdą,} \\ 0 & \text{jeżeli } \ell \text{ jest fałszem;} \end{cases}$

Wprowadzenie

Książka jest o metodach optymalizacji. Ponieważ są one w praktyce numeryczne, więc należało dokonać wyboru referencyjnej implementacji stosowanych procedur. Wykorzystujemy R i w nim będą wszystkie przykłady podawane. To nie jest podręcznik od podstaw do R. Raczej używamy R jako wygodnego, zwięzłego języka do pisania odpowiednika pseudokodu w tradycyjnych podręcznikach + dokładnie omawiamy jak działają metody optymalizacji dostępne R.

W kodzie R w komentarzach podajemy wynik działania podawanych komend.

Zawiera ona omówienie procedur optymalizacyjnych dostępnych w podstawowej (ang. *core*) dystrybucji R:

1) Optymalizacji funkcji jednej zmiennej `optimize` w `stats` + gradientowe;
2) Optymalizacji funkcji wielu zmiennych bez ograniczeń: `optim` w `stats`;
3) Optymalizacji funkcji wielu zmiennych z ograniczeniami: pakiet `Rsolnp`? - to jest w core R;
4) Metod heurystycznych: `stats` (`optim` z opcją `SANN`) + algorytmy genetyczne samodzielnie kodowane;

. Na początku jest wprowadzenie do teorii optymalizacji i podstawy zagadnień numerycznych.

Każdy rozdział ma następujący układ:

1) Wstęp omawiający teorię, referencyjną implementację w R metody (żeby student dokładnie wiedział jak działa) + powiedzenie, gdzie w R to jest zaimplementowane *na poważnie* i proste przykłady;
2) Bardziej zaawansowane przykłady natury ekonomicznej (optymalizacja: tras, załadunku, portfela inwestycyjnego, jakieś zagadnienia typu MNK czy wyznaczanie best response functions) - użycie metod z R na większych zadaniach — tu zakładam, że zadania używane na metodach optymalizacji wejdą;
3) Zadania do samodzielnego rozwiązania przez studentów. **TODO:** Nie są to zadania utrwalające tylko poszerzające wiedzę i mająca za zadanie stymulować czytelnika do własnej analizy poruszanych problemów.

.

W referencyjnej implementacji metod pozostawiane są tylko kroki podstawowego algorytmu, a w celu zwiększenia jakości kodu pomijane elementy odpowiadające za obsługę błędów. **TODO:** Jest ona w pełni sformatowana, żeby pokazać jak to robić, ale jest pozbawiona komentarzy, bo jest skomentowana w tekście. Implementacja i opisy są zgodne z wersją R numer 2.15.0, 2012-03-30.

TODO: Kody nie posiadają kontroli błędów. Więc nie są produkcyjne

1. Teoria optymalizacji

Niniejszy rozdział przedstawia teorię optymalizacji. Nie jest to jednak systematyczny i kompletny wykład. Szczególną uwagę poświęcono prostemu i intuicyjnemu przedstawieniu teorii optymalizacji. W celu zapoznania się z ogólną teorią, zalecane jest korzystanie w pozycji takich jak **TODO:** zacytować. Ważną pomocą przy pisaniu niniejszego rozdziału były następujące pozycje:

- D. Ray, *Optimization for Engineering Design: Algorithms and Examples*, Prentice-Hall of India, 2005;
- M. J. Osborne, *Mathematical methods for economic theory*, skrypt interaktywny, dostępny w: http://www.economics.utoronto.ca/osborne/MathTutorial/index.html;
- A. K. Dixit, *Optimization in Economic Theory*, Oxford University Press, 1990;
- J. Levin, A. Rangel, *Useful Math for Microeconomics*, skrypt, Stanford University, 2001;

Rozdział podzielony jest na następujące części.

Część 1.1 wprowadza podstawowe definicje, pojęcia i twierdzenia, których znajomość niezbędna jest w dalszych częściach niniejszej książki.

Część 1.2 omawia problem optymalizacyjny funkcji wielu zmiennych bez ograniczeń. W części 1.2.1 zaprezentowane jest twierdzenie dotyczące istnienia rozwiązania oraz warunki konieczne i wystarczające dla optimum lokalnego i globalnego funkcji wielu zmiennych. Natomiast w części 1.2.2 omówione są przykłady zastosowania teorii.

Część 1.3 omawia problem optymalizacyjny funkcji wielu zmiennych z ograniczeniami. W części 1.3.1 wprowadzony jest ogólny problem, podstawowe pojęcia oraz najważniejsze pytania, które należy zadać analizując problem. Część 1.3.2 omawia własności rozwiązań: Istnienie, ciągłość, jednoznaczność oraz wypukłość. Część 1.3.3 oraz 1.3.4 dokonuje charakterystyki rozwiązań w dwóch ważnych szczególnych przypadkach ogólnego problemu optymalizacji z ograniczeniami, mianowicie kiedy ograniczenia można zapisać w postaci nierówności oraz w postaci równań. Część 1.3.3 przedstawia intuicję rozwiązań w tych przypadkach, natomiast część 1.3.4 przedstawia główne rezultaty teoretyczne, które identyfikują w sposób formalny warunki konieczne i wystarczające optymalności w omawianych przypadkach. Część 1.3.5 zawiera przykłady oraz zastosowania prezentowanej teorii. Część dodatkowa 1.4 zawiera dyskusję na temat słabej dualności i zawiera przykład jej zastosowania.

1.1. Definicje i pojęcia wstępne

W niniejszej części wprowadzimy pojęcia, definicje i twierdzenia, do których będziemy odwoływać się w dalszej części książki. Należy zaznaczyć, że wybór omówionych pojęć jest selektywny i nie stanowi pełnego wykładu matematyki na potrzeby optymalizacji. Systematyczny wykład czytelnik znajdzie na przykład w **TODO:** zacytuj podręczniki.

Będziemy zajmowali się funkcjami $f\colon \mathbf{R}^n \to \mathbf{R}$. Niech $D \subset \mathbf{R}^n$ będzie dziedziną na której chcemy badać funkcję f.

Definicja 1.1. *Punkt $\mathbf{x_0} \in D$ taki, że $\forall \mathbf{x} \in D\colon f(\mathbf{x}) \leq f(\mathbf{x_0})$ nazywamy maksimum globalnym funkcji f na zbiorze D.*

Definicja 1.2. *Punkt $\mathbf{x_0} \in D$ taki, że $\exists r > 0\colon \forall \mathbf{x} \in \mathcal{K}(\mathbf{x_0}, r) \cap D\colon f(\mathbf{x}) \leq f(\mathbf{x_0})$ nazywamy maksimum lokalnym funkcji f na zbiorze D.*

Analogicznie funkcja $f(\mathbf{x})$ ma w punkcie $\mathbf{x_0}$ minimum lokalne (globalne) wtedy i tylko wtedy gdy funkcja $-f(\mathbf{x})$ ma w tym punkcie maksimum lokalne (globalne). Rysunek 1.1 przedstawia przykłady ekstremów lokalnych i globalnych funckji $f(x) = 3x^4 - 4x^3 - 36x^2$ na przedziale $[-3, 4]$.

Definicja 1.3. *Zbiór D jest ograniczony, jeśli istnieje liczba $M \in \mathbf{R}$, że $D \subset \mathcal{K}(\mathbf{0}, M)$. Jeśli taka liczba nie istnieje, to zbió jest nieograniczony.*

Na przykład zbiór $\{(x_1, x_2) : x_1 x_2 \leq 1\}$ jest nieograniczony, ponieważ dla każdej liczby m, punkt $(2m, 0)$ należy do zbioru a odległość tego punktu od początku układu współrzędnych wynosi $2m$ i jest większa niż m.

Definicja 1.4. *Zbiór $D \subset \mathbf{R}^n$ nazywamy zwartym wtedy i tylko wtedy gdy jest ograniczony oraz granica każdego ciągu zbieżnego elementów z D należy do D.*

Definicja 1.5. *Niech D zawiera w swoim wnętrzu $\mathbf{0}$. Weźmy funkcje $g\colon D \to \mathbf{R}$ oraz $f\colon D \to \mathbf{R}$, przy czym $\mathbf{x} \neq \mathbf{0} \Rightarrow g(\mathbf{x}) \neq 0$. Będziemy pisali $f(\mathbf{x}) = \mathcal{o}(g(\mathbf{x}))$ wtedy i tylko wtedy gdy:*

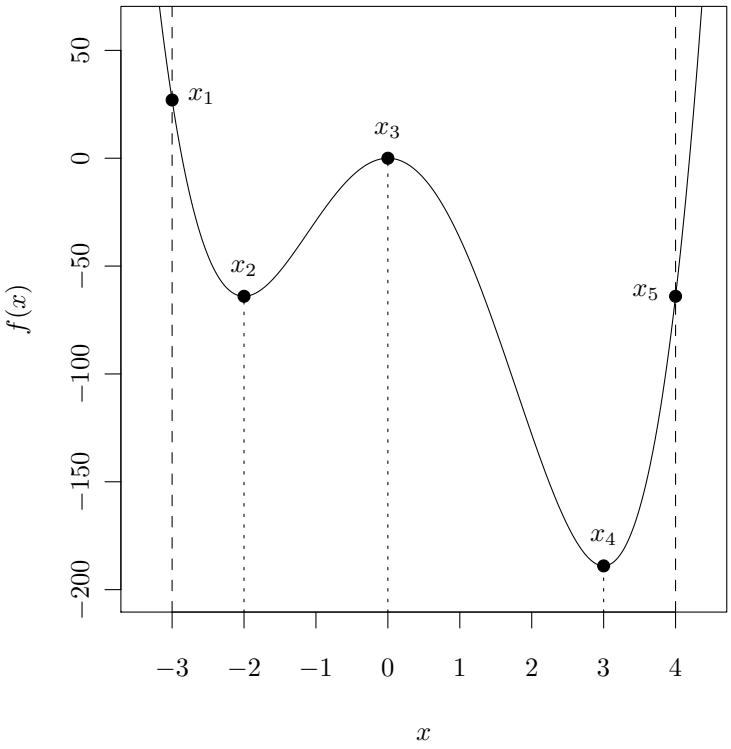
$$\lim_{D \ni \mathbf{x} \to \mathbf{0}} \frac{f(\mathbf{x})}{g(\mathbf{x})} = 0. \tag{1.1}$$

Twierdzenie 1.1. *Weźmy dowolne $x, h \in \mathbf{R}$. Jeżeli funkcja $f\colon \mathbf{R} \to \mathbf{R}$ jest n-razy różniczkowalna w sposób ciągły w pewnym przedziale otwartym zawierającym punkty x i $x + h$ to:*

$$f(x + h) = f(x) + \sum_{k=1}^{n} \frac{f^{(k)}(x)}{k!} h^k + \mathcal{o}(h^n). \tag{1.2}$$

Twierdzenie 1.2. *Weźmy dowolne $\mathbf{x}, \mathbf{h} \in \mathbf{R}^n$. Jeżeli funkcja $f\colon \mathbf{R}^n \to \mathbf{R}$ jest dwukrotnie różniczkowalna w sposób ciągły w pewnym przedziale otwartym zawierającym punkty $\mathbf{x}$ i $\mathbf{x} + \mathbf{h}$ to:*

$$f(\mathbf{x} + \mathbf{h}) = f(\mathbf{x}) + \nabla f(\mathbf{x})\mathbf{h} + \frac{1}{2}\mathbf{h}^T \nabla^2 f(\mathbf{x})\mathbf{h} + \mathcal{o}(||\mathbf{h}||^2). \tag{1.3}$$



Badana jest funkcja $f(x) = 3x^4 - 4x^3 - 36x^2$ na przedziale $[-3, 4]$. Punkty x_1, x_3 i x_5 są maksimami lokalnymi, a x_2 i x_4 minimami lokalnymi. Punkt x_1 jest maksimum globalnym a x_4 minimum globalnym.

Rysunek 1.1: Ekstrema lokalne i globalne funkcji

Definicja 1.6. *Macierz kwadratową A nazywamy półdodatnio określoną wtedy i tylko wtedy gdy $\forall \mathbf{x} \in \mathbf{R}^n \colon \mathbf{x}^T A \mathbf{x} \geq 0$. Macierz A nazywamy dodatnio określoną jeśli jest półdodatnio okreslona i $\mathbf{x}^T A \mathbf{x} = 0 \Rightarrow \mathbf{x} = \mathbf{0}$.*

Macierz A nazywamy ujemnie (półujemnie) określoną, jeżeli macierz $-A$ jest dodatnio (półdodatnio) określona.

Twierdzenie 1.3. *Niech $f \colon \mathbf{R}^n \to \mathbf{R}$ będzie dwukrotnie różniczkowalna w sposób ciągły w punkcie $\mathbf{x}$. W takiej sytuacji hesjan $\nabla^2 f(\mathbf{x})$ jest macierzą symetryczną, a jego wszytkie wartości własne są rzeczywiste. Jeśli wszystkie wartości własne $\nabla^2 f(\mathbf{x})$ są nieujemne/dodatnie/niedodatnie/ujemne, wówczas hesjan jest odpowiednio dodatnio półokreślony/dodatnio określony/ujemnie półokreślony/ujemnie określony. Z kolei jeśli niektóre wartości własne są dodatnie a inne ujemne, to jest on nieokreślony.*

Definicja 1.7. *Zbiór $D \subset \mathbf{R}^n$ nazywamy wypukłym wtedy i tylko wtedy gdy*

$$\forall \mathbf{x}_1, \mathbf{x}_2 \in D, \lambda \in [0,1] \colon \lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2 \in D. \tag{1.4}$$

Definicja 1.8. *Funkcję $f \colon \mathbf{R}^n \to \mathbf{R}$ nazywamy wypukłą na wypukłym zbiorze D wtedy i tylko wtedy gdy:*

$$\forall \mathbf{x}_1, \mathbf{x}_2 \in D, \lambda \in [0,1] \colon f(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2). \tag{1.5}$$

Twierdzenie 1.4. *Funkcja $f \colon \mathbf{R}^n \to \mathbf{R}$ zadana na wypukłym zbiorze D i dwukrotnie różniczkowalna w sposób ciągły jest wypukła wtedy i tylko wtedy gdy dla każdego $\mathbf{x} \in D$ hesjan $\nabla^2 f(\mathbf{x})$ jest półdodatnio określony.*

Twierdzenie 1.5. *Funkcja $f \colon \mathbf{R} \to \mathbf{R}$ ma tylko jedno maksimum lokalne wtedy i tylko wtedy gdy:*

$$\exists x_0 \in \mathbf{R} \, \forall \delta_1 > \delta_2 \geq 0, s \in \{-1, 1\} \colon f(x_0 + s\delta_1) \geq f(x_0 + s\delta_2). \tag{1.6}$$

Analogicznie funkcja $f \colon \mathbf{R} \to \mathbf{R}$ ma tylko jedno minimum lokalne wtedy i tylko wtedy gdy funkcja $-f$ ma jedno maksimum lokalne.

Jeśli minimalizowana funkcja ma tylko jedno minimum lokalne lub maksymalizowana funkcja ma tylko jedno maksimum lokalne, wówczas mówimy, że optymalizacja dotyczy funkcji unimodalnej.

1.1.1. Korespondencje i ich ciągłość

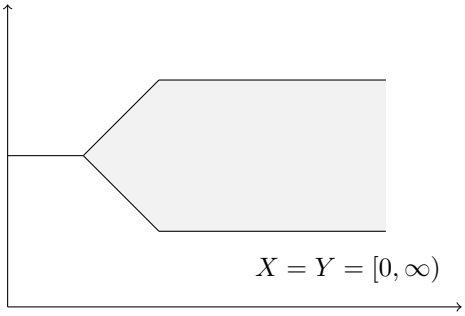
Definicja 1.9. *Dane są zbiory X, Y. W ogólności zbiory te mogą być jakiekolwiek, ale w konkretnych zastosowaniach będziemy zakładać, że $X, Y \in \mathbf{R}^n$. Niemniej dopóki nie zostanie explicite stwierdzone, że $X, Y \in \mathbf{R}^n$, elementy X, Y oznaczać będziemy poprzez x, y. Korespondencja (lub inaczej multifunkcja) $D : X \Rightarrow Y$ jest odwzorowaniem, które przyporządkowuje zbiór $D(x) \subset Y$ dla każdego $x \in X$. Graf korespondencji $D : X \Rightarrow Y$ to zbiór $\{(x, y) \in X \times Y : y \in D(x)\}$*

Korespondencja D ma takie samo oznaczenie jak dziedzina funkcji celu, ponieważ korespondencję będziemy stosować w niniejszym rozdziale na oznaczenie dziedziny funkcji celu, która może się zmieniać w zależności od parametru. Jeśli dziedzina jest stała, wówczas dziedzina D oznaczać będzie stałą korespondencję, czyli jeden zbiór.

Zdefiniowanie ciągłości funkcji rzeczywistej, nazwijmy ją h, wymaga zbadania jak zmienia się ta funkcja przy niewielkiej zmianie jej argumentu (z x do x'). Wystarczy więc porównać wartości funkcji $h(x)$ oraz $h(x')$. W przypadku badania ciągłości korespondencji D, należy porównać dwa zbiory $D(x)$ oraz $D(x')$. Wykres 1.2 przedstawia korespondencję, która jest ciągła w każdym punkcie swojej dziedziny. Intuicyjnie, zbiory będące wartościami korespondencji zmieniają się w sposób ciągły, w miarę jak nieznacznie zmieniają się argumenty tej korespondencji.

TODO: brak oznaczenia osi na wykresie

Aby w sposób formalny porównać dwa zbiory, trzeba zdefiniować dwa pojęcia - dolnej i górnej hemiciągłości.



Wykres 1.2: Ciągła korespondencja.

Definicja 1.10. *Korespondencja $D : X \Rightarrow Y$ jest dolnie hemiciągła (DHC) w punkcie x jeśli dla każdego otwartego zbioru G: $G \cap D(x) \neq \emptyset$, istnieje zbiór otwarty $H(x)$ zawierający x taki, że jeśli $x' \in H(x)$, to $D(x') \cap G \neq \emptyset$. Korespondencja jest DHC, jeśli jest DHC w każdym punkcie $x \in X$.*

Dolna hemiciągłość intuicyjnie oznacza, że do każdego elementu w $D(x)$ można dojść z każdego kierunku.

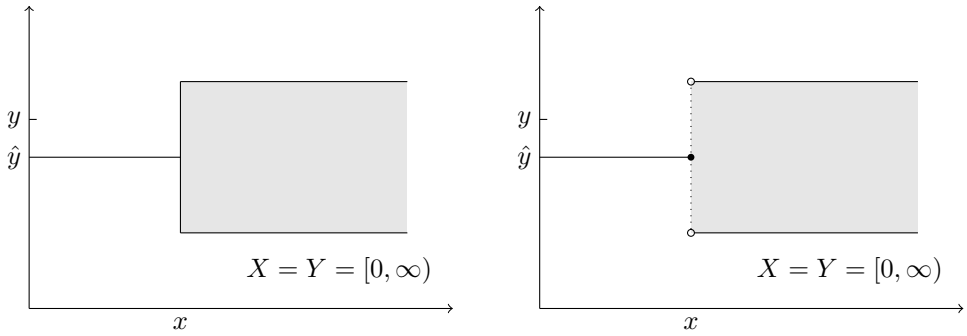
Wykres 1.3a przedstawia korespondencję, która nie jest dolnie hemiciągła w punkcie x. Rozważmy punkt $y \in D(x)$. Niech zbiór G będzie małym przedziałem zawierającym y, ale nie zawierającym $\hat{y}$. Każdy otwarty zbiór zawierający x będzie zawierał jakiś punkt x' na lewo od punktu x. Ale wówczas $D(x') = \{\hat{y}\}$ nie będzie miał punktów wspólnych ze zbiorem G.

Z drugiej strony wykres 1.3b przedstawia korespondencję, która jest dolnie hemiciągła w punkcie x (i również w każdym innym punkcie). Rozważmy zbiór $D(x) = \{\hat{y}\}$. Dla każdego punktu x' na lewo od punktu x, $D(x') = \{\hat{y}\}$, a zatem każdy zbiór przecinający $D(x)$ będzie również przecinał $D(x')$. Z kolei dla punktu x' nieznacznie na prawo od punktu x, $D(x) \subset D(x')$. A zatem każdy zbiór przecinający $D(x)$ będzie również przecinał $D(x')$. Zauważmy, że dla każdego punktu x' większego od x, istnieje przedział otwarty nie zawierający punktu x, a zatem dla każdego punktu na prawo od x korespondencja jest dolnie hemiciągła. Dla punktów na prawo od x korespondencja też oczywiście jest dolnie hemiciągła.

Definicja 1.11. *Korespondencja $D : X \Rightarrow Y$ jest górnie hemiciągła (GHC) w punkcie x jeśli dla każdego otwartego zbioru $G \supset D(x)$, istnieje zbiór otwarty $H(x)$ zawierający x taki, że jeśli $x' \in H(x)$, to $D(x') \subset G$. Korespondencja jest GHC, jeśli jest GHC w każdym punkcie $x \in X$ oraz jej obraz jest zbiorem zwartym.*

Górna hemiciągłość intuicyjnie oznacza, że $D(x)$ nie zacznie nagle zawierać nowych punktów w miarę jak odchodzimy z punktu x.

Wykres 1.3a przedstawia korespondencję, która jest górnie hemiciągła w punkcie x (i również w każdym innym punkcie). Weźmy jakikolwiek otwarty przedział G zawierający $D(x)$ i rozważmy punkt x' nieznacznie na lewo od punktu x. Wówczas zbiór



(a) GHC, ale nie DHC

(b) DHC, ale nie GHC

Wykres 1.3: Korespondencje nieciągłe

$D(x') = \{\hat{y}\}$ należy do G. Teraz rozważmy punkt x' nieznacznie na prawo od punktu x. Również zbiór $D(x')$ należy do G, pod warunkiem, że x' jest dostatecznie blisko punktu x.

Z drugiej strony wykres 1.3b przedstawia korespondencję, która nie jest górnie hemiciągła w punkcie x. Zauważmy, że $D(x) = \{\hat{y}\}$. Rozważmy punkt x' nieznacznie na prawo od punktu x. Jest oczywiste, że zbiór $D(x')$ zawiera dużo punktów, które nie są blisko $\hat{y}$.

Definicja 1.12. *Korespondencja $D : X \Rightarrow Y$ jest ciągła w punkcie x, jeżeli jest dolnie i górnie hemiciągła w tym punkcie. Korespondencja jest ciągła, jeśli jest dolnie i górnie hemiciągła.*

1.2. Optymalizacja bez ograniczeń

Niniejszy rozdział dotyczy optymalizacji funkcji wielu zmiennych bez ograniczeń. Wykres 1.4 przedstawia różne kształty optymalizowanych funkcji.

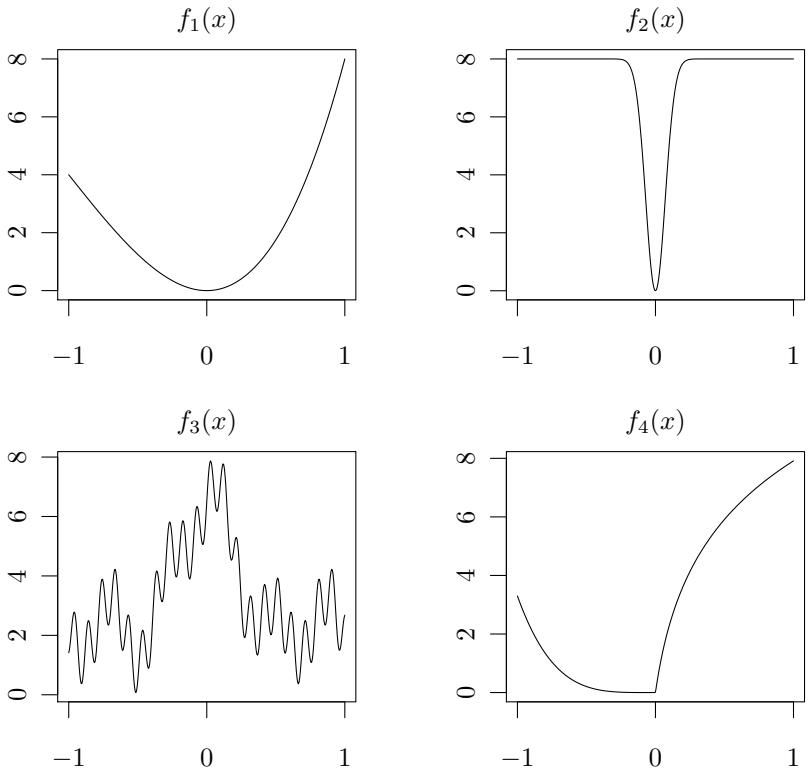
1.2.1. Teoria

Niech $f : D \to \mathbf{R}$ będzie funkcją n zmiennych zdefiniowaną na zbiorze $D \subset \mathbf{R}^n$. Rozważmy następujący problem:

$$\max_{\mathbf{x} \in D} f(\mathbf{x})$$

Kwestie, którymi będziemy się zajmowali to:

1. Czy rozwiązanie tego zadania istnieje?
2. Jeśli tak, to czy rozwiązanie jest jedno czy jest ich wiele?
3. Jak znaleźć rozwiązanie?



Badane są funkcje postaci:
$f_1(x) = 6x^2 + 2x^3$,
$f_2(x) = 8\big(1 - \exp(-100x^2)\big)$,
$f_3(x) = 2\cos(4x) + \cos(8x) + \sin(16x) + \sin(64x) + 3{,}5$,
$f_4(x) = 3{,}3\big([x < 0]x^4 + [x \geq 0]\ln(10x + 1)\big)$.

Wykres 1.4: Przykłady różnych kształtów optymalizowanej funkcji

Na początek przyjrzyjmy się następującym przykładom w przestrzeni $\mathbf{R}$:

- $f(x) = x$, $D = [0, \infty)$;
- $f(x) = 1 - \frac{1}{x}$, $D = [1, \infty)$;
- $f(x) = x$, $D = (0, 1)$;
- $f(x) = x$, jeśli $x < 0.5$ oraz $f(x) = x - 1$, jeśli $x \geq 0.5$, $D = [0, 1]$;

W pierwszych dwóch przypadkach funkcja f nie osiąga wartości maksymalnej, ponieważ zbiór D jest nieograniczony. W trzecim przypadku funkcja f nie osiąga

maksimum na zbiorze D, ponieważ D jest otwarty. W ostatnim przypadku funkcja f nie osiąga maksimum na zbiorze D, ponieważ funkcja f jest nieciągła.

Twierdzenie 1.6 (Twierdzenie Weierstrassa)**.** *Ciągła funkcja na zbiorze zwartym osiąga maksimum i minimum.*

Zauważmy, że jeśli funkcja jest ograniczona, to maksimum może nie istnieć (patrz drugi przykład powyżej).

Powyższe twierdzenie identyfikuje warunki wystarczające na istnienie ekstremum globalnego funkcji. Twierdzenie to daje nam częściową odpowiedź na pytanie dotyczące istnienia rozwiązania. Poniżej identyfikujemy warunki konieczne istnienia ekstremum lokalnego funkcji w przypadku kiedy funkcja jest różniczkowalna. Warunki te nazywamy warunkami pierwszego rzędu.

Twierdzenie 1.7. *Niech $f : D \to \mathbf{R}$ będzie różniczkowalną funkcją, gdzie $D \subset \mathbf{R}^n$. Jeśli punkt $\mathbf{x}$ znajdujący się we wnętrzu zbioru D jest lokalnym lub globalnym maximum lub minimum funkcji f, to:*

$$\nabla f(\mathbf{x}) = 0$$

Mówimy, że punkt $\mathbf{x}$ jest punktem stacjonarnym funkcji f.

Rozważmy trzy przykłady w przestrzeni $\mathbf{R}$.

- $\max\limits_{x\in[-1,2]} x^2$. Funkcja jest ciągła, więc na mocy Twierdzenia Weierstrassa, problem ma rozwiązanie. Jedyny punkt stacjonarny to $x = 0$ i należy do przedziału $[-1, 2]$. Wartości funkcji w punkcie stacjonarnym oraz na brzegach wynoszą: $f(-1) = 1$, $f(0) = 0$, $f(2) = 4$, a zatem punkt $x = 2$ jest globalnym maksimum a punkt $x = 0$ jest globalnym minimum.

- $\max\limits_{x\in(-\infty,\infty)} -x^2$. Problem nie spełnia warunków Twierdzenia Weierstrassa, a zatem nie wiadomo, czy problem ma rozwiązanie. Jedyny punkt stacjonarny to $x = 0$, przedział $(-\infty, \infty)$ nie ma brzegów. Zatem jeśli problem ma rozwiązanie, to jest to punkt $x = 0$. W tym przypadku rzeczywiście problem ma rozwiązanie w punkcie $x = 0$.

- $\max\limits_{x\in(-\infty,\infty)} x^2$. Problem nie spełnia warunków Twierdzenia Weierstrassa, a zatem nie wiadomo, czy problem ma rozwiązanie. Jedyny punkt stacjonarny to $x = 0$, przedział $(-\infty, \infty)$ nie ma brzegów. Zatem jeśli problem ma rozwiązanie, to jest to punkt $x = 0$. W tym przypadku problem nie ma rozwiązania - wartość funkcji rośnie w nieskończoność.

Warunki drugiego rzędu to warunki wystarczające na istnienie ekstermum lokalnego funkcji w przypadku, kiedy funkcja jest różniczkowalna.

Twierdzenie 1.8. *Niech $f : D \to \mathbf{R}$, $D \subset \mathbf{R}^n$ będzie funkcją podwójnie różniczkowalną w sposób ciągły. Niech $\mathbf{x}^*$ będzie punktem stacjonarnym funkcji f we wnętrzu zbioru D. Wówczas:*

- *Jeśli Hesjan $\nabla^2 f(\mathbf{x}^*)$ jest ujemnie (dodatnio) określony, wtedy $\mathbf{x}^*$ jest lokalnym maximum (minimum).*
- *Jeśli $\mathbf{x}^*$ jest lokalnym maksimum (minimum), wtedy Hesjan $\nabla^2 f(\mathbf{x}^*)$ jest określony półujemnie (półdodatnio).*

Punkt stacjonarny, który nie jest ani lokalnym minimum ani lokalnym maksimum nazywany jest punktem siodłowym. Na przykład punkt $(0,0)$ dla funkcji $f(x,y) = x^2 - y^2$.

Rozważmy następujący przykład: $f(x,y) = 8x^3 + 2xy - 3x^2 + y^2 + 1$. Warunki pierwszego rzędu to:

$$\begin{aligned} \frac{\partial f(x,y)}{\partial x} &= 24x^2 + 2y - 6x = 0 \\ \frac{\partial f(x,y)}{\partial y} &= 2x + 2y \end{aligned}$$

Są dwa rozwiązania tego układu równań: $(x^*, y^*) = (0,0)$ oraz $(x^{**}, y^{**}) = (1/3, -1/3)$. Sprawdźmy warunki drugiego rzędu. Macierz Hesjanu przyjmuje postać:

$$\nabla^2 f(x,y) = \begin{pmatrix} 48x - 6 & 2 \\ 2 & 2 \end{pmatrix}$$

Dla punktu (x^*, y^*) wartości własne Hesjanu wynoszą odpowiednio $\lambda_1^* = -2 - 2\sqrt{5} < 0$ oraz $\lambda_2^* = -2 + 2\sqrt{5} > 0$, czyli macierz jest nieokreślona a zatem punkt (x^*, y^*) jest punktem siodłowym. Z kolei dla punktu (x^{**}, y^{**}) wartości własne Hesjanu wynoszą odpowiednio $\lambda_1^{**} = 6 - 2\sqrt{5} > 0$ oraz $\lambda_2^{**} = 6 + 2\sqrt{5} > 0$, czyli macierz jest dodatnio określona a zatem punkt (x^{**}, y^{**}) jest minimum lokalnym funkcji.

Poniżej identyfikujemy warunki wystarczające dla ekstremum globalnego funkcji.

Twierdzenie 1.9. *Niech $f : D \to \mathbf{R}$, będzie funkcją różniczkowalną w sposób ciągły na wypukłym zbiorze $D \subset \mathbf{R}^n$ i niech $\mathbf{x}$ będzie we wnętrzu zbioru D. Wówczas:*

- *Jeśli f jest wklęsła (wypukła), wtedy $\mathbf{x}$ jest maksimum (minimum) globalnym funkcji f na zbiorze S wtedy i tylko wtedy gdy jest punktem stacjonarnym funkcji f.*

Rozważmy nastepujący przykład: $f(x,y) = x^4 + 2y^2$, zdefiniowanej na dziedzinie $(-\infty, \infty)$. Warunki pierwszego rzędu to:

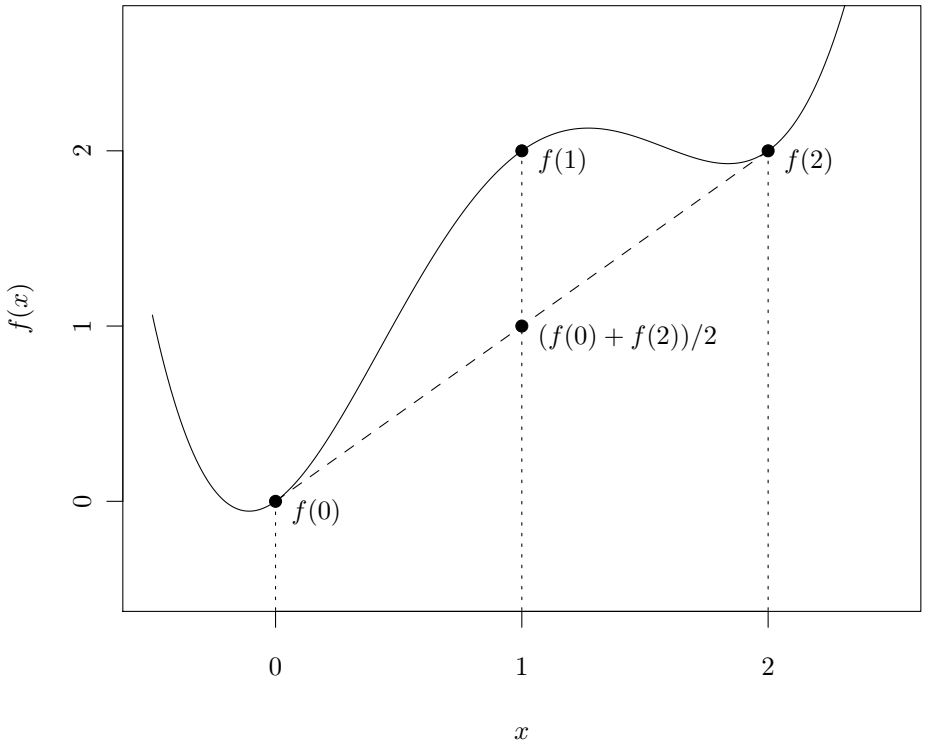
$$\begin{aligned} \frac{\partial f(x,y)}{\partial x} &= 4x^3 \\ \frac{\partial f(x,y)}{\partial y} &= 4y \end{aligned}$$

Jest jedno rozwiązanie tego układu równań: $(x^*, y^*) = (0,0)$. Sprawdźmy warunki drugiego rzędu. Macierz Hesjanu przyjmuje postać:

$$\nabla^2 f(x,y) = \begin{pmatrix} 12x^2 & 0 \\ 0 & 4 \end{pmatrix}$$

Wartości własne tej macierzy to $\lambda_1 = 12x^2$ oraz $\lambda_2 = 4$, a zatem dla wszystkich punktów (x, y) macierz ta jest półdodatnio określona, czyli funkcja f jest wypukła a punkt $(0, 0)$ jest globalnym minimum tej funkcji.

Inny przykład przedstawia Wykres 1.5. Ponieważ funkcja nie jest wypukła na całej swojej dziedzinie, punkt stacjonarny nie musi być minimum globalnym funkcji; w tym wypadku tylko jeden z punktów satcjonarnych jest minimum globalnym. I odwrotnie, ponieważ funkcja nie jest wklęsła na całej swojej dziedzinie, punkt stacjonarny nie musi być maksimum globalnym funkcji; w tym przypadku żaden z punktów stacjonarnych nie jest maksimum globalnym funkcji.



Badana jest funkcja $f(x) = x^2(x-2)^2 + x$.

Wykres 1.5: Przykład istotności założeń o wypukłości funkcji w twierdzeniu 1.9

Zauważmy różnicę pomiędzy Twierdzeniem 1.8 i Twierdzeniem 1.9:

- Warunki wystarczające na maksimum lokalne: jeśli $\mathbf{x}^*$ jest punktem stacjonarnym f a Hesjan funkcji f jest ujemnie określony w punkcie $\mathbf{x}^*$, wówczas $\mathbf{x}^*$ jest maksimum lokalnym funkcji f.

- Warunki wystarczające na maksimum globalne: jeśli $\mathbf{x}^*$ jest punktem stacjonarnym f a Hesjan funkcji f jest półujemnie określony dla wszystkich $\mathbf{x}$, wówczas $\mathbf{x}^*$ jest maksimum globalnym funkcji f.

1.2.2. Przykłady zastosowań

W niniejszej części omówione zostaną dwa przykłady zastosowań teorii optymalizacji bez ograniczeń. Pierwszy przykład dotyczy estymacji parametrów klasycznego modelu regresji liniowej za pomocą minimalizacji odchyleń kwadratowych, natomiast drugi dotyczy estymacji parametrów modelu logitowego za pomocą metody największej wiarygodności.

Minimalizacja odchyleń kwadratowych

Rozważmy liniowy model ekonometryczny $y = X\beta + \epsilon$, gdzie $y_{N\times 1}$ jest wektorem obserwacji zmiennej zależnej, $X_{N\times K}$ jest macierzą obserwacji K zmiennych niezależnych, której rząd wynosi K, natomiast $\epsilon_{N\times 1}$ jest wektorem błędów. Chcemy znaleźć taką wartość parametrów β dla której minimalizowana jest suma kwadratów odchyleń obserwacji y od prognoz $X\beta$:

$$\begin{aligned}\min_{\beta} S(\beta) &= \epsilon^T\epsilon = (y - X\beta)^T(y - X\beta)\\ &= (y^T - \beta^T X^T)(y - X\beta)\\ &= y^T y - \beta^T X^T y - y^T X\beta + \beta^T X^T X\beta\\ &= y^T y - 2y^T X\beta + \beta^T X^T X\beta\end{aligned}$$

gdzie ostatnia równość wynika z faktu, że transpozycja skalara jest równa skalarowi: $\beta^T X^T y = (\beta^T X^T y)^T = y^T X\beta$. Warunek pierwszego rzędu optymalizacji względem β to[1]:

$$\nabla S(\beta) = 2X^T y + 2X^T X\beta = 0$$

Ponieważ X ma rząd K, więc macierz $X^T X$ jest odwracalna i $\hat{\beta} = (X^T X)^{-1} X^T y$ jest kandydatem do optimum. Pozostało sprawdzić warunki drugiego rzędu na minimum, czyli czy Hesjan jest dodatnio określony:

$$\nabla^2 S(\beta) = 2(X^T X)^T = 2X^T X$$

Wystarczy więc pokazać, że dla każdego $\mathbf{x} \neq \mathbf{0}$ mamy $\mathbf{x}^T \nabla^2 S(\beta)\mathbf{x}$. Zauważmy jednak, że to wyrażenie jest równe $(X\mathbf{x})^T(X\mathbf{x})$. Ponieważ X ma rząd K więc $X\mathbf{x} \neq 0$, więc badane wyrażenie jest zawsze dodatnie dla $\mathbf{x} \neq \mathbf{0}$. Oznacza to, że badany Hesjan jest dodatnio okreslony, a więc $\hat{\beta}$ jest minimum funkcji $S(\beta)$.

[1] Wykorzystaliśmy dwie reguły $\frac{\partial a^T x}{\partial x} = a$ oraz $\frac{\partial x^T A x}{\partial x} = 2ax$, gdzie a, x to wektory o tych samych wymiarach $K \times 1$, a A to macierz kwadratowa o wymiarach $K \times K$.

Największa wiarygodność

Dana jest próba obserwacji binarnych $y_1, y_2, ..., y_n$, $y_i \in \{0, 1\}$. Wiadomo, że zostały wylosowane niezależnie, ale nie wiadomo, z jakiego rozkładu. Załóżmy, że prawdopodobieństwo, że dana obserwacja przyjmuje wartość 1 wynosi $P = F(X\beta)$ - nakładamy warunek, że rozkładem prawdopodobieństwa skąd pochodzą obserwacje jest F. Zakładając, że założenie jest prawdziwe, chcemy znaleźć parametry rozkładu F, który jest najbardziej prawdopodobny, pod warunkiem, że obserwacje są dane przez $y_1, y_2, ..., y_n$.

Funkcja wiarygodności to łączne prawdopodobieństwo wystąpienia obserwacji w próbie:

$$\begin{aligned} L &= Pr(Y_1 = y_1, Y_2 = y_2, ..., Y_n = y_n) \\ &= \prod_{y_i=0} [1 - F(X_i\beta)] \prod_{y_i=1} F(X_i\beta) \\ &= \prod_{i=1}^{n} [1 - F(X_i\beta)]^{1-y_i} F(X_i\beta)_i^y \end{aligned}$$

gdzie $X_{i1\times K}$ jest wektorem wartości zmiennych niezależnych dla obserwacji i.

Po zlogarytmowaniu otrzymujemy funkcję łog-likelihood":

$$\ln L = \sum_{i=1}^{n} [(1 - y_i) \ln(1 - F(X_i\beta)) + y_i \ln(F(X_i\beta))]$$

Przyjmijmy model logitowy, czyli funkcja F ma następującą postać:

$$Pr(Y = 1) = F(X\beta) = \frac{e^{X\beta}}{1 + e^{X\beta}} = l(X\beta)$$

Wówczas:

$$f(X\beta) = \frac{e^{X\beta}}{(1 + e^{X\beta})^2} = l(X\beta)[1 - l(X\beta)]$$

Funkcja log-wiarygodności:

$$\ln L = \sum_{i=1}^{n} [(1 - y_i) \ln(1 - l(X_i\beta)) + y_i \ln(l(X_i\beta))]$$

Warunki pierwszego rządu przyjmują zatem postać:

$$\begin{aligned} \frac{\partial \ln L}{\partial \beta^T} &= \sum_{i=1}^{n} \left((1 - y_i) \frac{-l'(X_i\beta)}{1 - l(X_i\beta)} + y_i \frac{l'(X_i\beta)}{l(X_i\beta)} \right) X_i \\ &= \sum_{i=1}^{n} (-(1 - y_i) l(X_i\beta) + y_i (1 - l(X_i\beta))) X_i \\ &= \sum_{i=1}^{n} (y_i - l(X_i\beta) X_i = 0 \end{aligned} \tag{1.7}$$

Warunki drugiego rzędu zaś to:

$$\nabla^2 \ln L = \frac{\partial^2 \ln L}{\partial \beta \partial \beta^T} = -\sum_{i=1}^{n} l(X_i\beta)(1 - l(X_i\beta))X_i^T X_i$$

Macierz kwadratowa $-X_i^T X_i$ jest ujemnie określona dla każdej obserwacji, dla której $X_i \neq \mathbf{0}$. Dla $X_i = \mathbf{0}$ macierz $-X_i^T X_i$ jest macierzą zer. Suma takich macierzy jest również macierzą ujemnie określoną. A zatem warunek drugiego rzędu na istnienie maksimum jest spełniony.

1.3. Optymalizacja z ograniczeniami.

Nieniejsza część dotyczy problemów optymalizacji funkcji wielu zmiennych z ograniczeniami.

1.3.1. Wprowadzenie

Będziemy rozpatrywać parametryczne problemy optymalizacji z ograniczeniami w następującej formie:

$$\max_{x \in D(\theta)} f(x, \theta).$$

gdzie f to funkcja celu (zyski, użyteczność), x to zmienna/e decyzyjna/e (jak dużo gadżetów wyprodukować, ile butelek oranżady kupić), $D(\theta)$ to zbiór dostępnych wyborów, a θ to egzogeniczny parametr, który może wpływać zarówno na funkcję celu, jak i na zbiór dostępnych wyborów (cena gadżetu lub butelki oranżady albo ilość złotówek w portfelu kupującego). Każdy parametr θ definiuje specyficzny problem. Zbiór wszystkich możliwych wartości parametrów θ oznaczmy jako Θ.

W problemach optymalizacyjnych interesują nas głównie dwa elementy:

1. Zbiór rozwiązań optymalnych dla każdego parametru $\theta \in \Theta$:

$$x^*(\theta) \equiv \arg \max_{x \in D(\theta)} f(x, \theta),$$

2. Optymalna wartość funkcji celu dla każdego parametru $\theta \in \Theta$ (funkcja wartości):

$$V(\theta) \equiv \max_{x \in D(\theta)} f(x, \theta)$$

Następujące pytania są kluczowe:

1. Czy rozwiązanie istnieje dla każdego parametru θ?
2. Czy zbiór rozwiązań optymalnych oraz funkcja wartości zmieniają się w sposób ciągły wraz ze zmianami parametrów?
3. Jak można znaleźć rozwiązanie problemu?

4. W jaki sposób zbiór rozwiązań optymalnych oraz funkcja wartości zmieniają się wraz ze zmianami parametrów?

Zauważmy, że każdy rezultat, który zachodzi dla problemu maksymalizacji, zachodzi również dla problemu minimalizacji, ponieważ:

$$x^*(\theta) = \arg\min_{x \in D(\theta)} f(x,\theta) \iff x^*(\theta) = \arg\max_{x \in D(\theta)} -f(x,\theta)$$

oraz

$$V(\theta) = \min_{x \in D(\theta)} f(x,\theta) \iff V(\theta) = -\max_{x \in D(\theta)} -f(x,\theta)$$

1.3.2. Własności rozwiązań: Istnienie, ciągłość, jednoznaczność oraz wypukłość.

Twierdzenie maksimum daje odpowiedź na pytanie 1 i 2 postawione powyżej, tj. Czy rozwiązanie istnieje dla każdego parametru θ oraz czy zbiór rozwiązań optymalnych oraz funkcja wartości zmieniają się w sposób ciągły wraz ze zmianami parametrów.

Twierdzenie 1.10 (Twierdzenie maksimum)**.** *Dana jest następująca parametryczna rodzina problemów optymalizacji z ograniczeniami:*

$$\max_{x \in D(\theta)} f(x,\theta).$$

zdefiniowana na zbiorze parametrów Θ. Niech:

(i) Korespondencja $D : \Theta \Rightarrow X$ jest ciągła (dolna hemiciągłość oraz górna hemiciągłość) oraz jej wartości (zbiory) są zwarte

(ii) Funkcja $f : X \times \Theta \to \mathbf{R}$ jest ciągła

Wówczas:

1. *Zbiór $x^*(\theta)$ jest niepusty dla każdego θ;*
2. *Korespondencja x^* jest górnie hemiciągła (a zatem ciągła, jeśli x^* jest jednoznaczne);*
3. *Funkcja V jest ciągła.*

Ażeby zrozumieć znaczenie powyższego twierdzenia, rozważmy parę kontrprzykładów:

1. Wartości korespondencji D nie są zwarte: wtedy rozwiązanie może nie istnieć da pewnych wartości parametrów θ. Na przykład $\Theta = [0,1]$, $D(\theta) = (0,1)$ oraz $f(x,\theta) = x$. Wówczas $x^*(\theta) = \emptyset$ for all θ.

2. Korespondencja D jest dolnie hemiciągła, ale nie jest górnie hemiciągła. Rozważmy nastepujący przykład: $\Theta = [0,1]$, $f(x,\theta) = x$, oraz

$$D(\theta) = \begin{cases} \{0\} & \text{jeżeli } \theta \leq 0,5; \\ [-1,1] & \text{jeżeli } \theta > 0,5 \end{cases} \tag{1.8}$$

Zbiór rozwiązań to wówczas:

$$x^*(\theta) = \begin{cases} \{0\} & \text{jeżeli } \theta \leq 0,5; \\ \{1\} & \text{jeżeli } \theta > 0,5 \end{cases} \tag{1.9}$$

Jest to funkcja, ale nie jest ciągła. Funkcja wartości V również nie jest ciągła.

3. Korespondencja D jest górnie hemiciągła, ale nie jest dolnie hemiciągła: Rozważmy nastepujący przykład: $\Theta = [0,1]$, $f(x,\theta) = x$, oraz

$$D(\theta) = \begin{cases} \{0\} & \text{jeżeli } \theta < 0,5; \\ [-1,1] & \text{jeżeli } \theta \geq 0,5 \end{cases} \tag{1.10}$$

Zbiór rozwiązań to wówczas:

$$x^*(\theta) = \begin{cases} \{0\} & \text{jeżeli } \theta < 0,5 \\ \{1\} & \text{jeżeli } \theta \geq 0,5 \end{cases} \tag{1.11}$$

Jest to również funkcja, która nie jest ciągła.

4. Funkcja f jest nieciągła. Rozważmy następujący przykład: $\Theta = [0,1]$, $D(\theta) = [\theta; \theta + 0,1]$ oraz

$$f(x,\theta) = \begin{cases} 0 & \text{jeżeli } x < 0,5 \\ 1 & \text{jeżeli } x \geq 0,5 \end{cases} \tag{1.12}$$

Zbiór rozwiązań to wówczas:

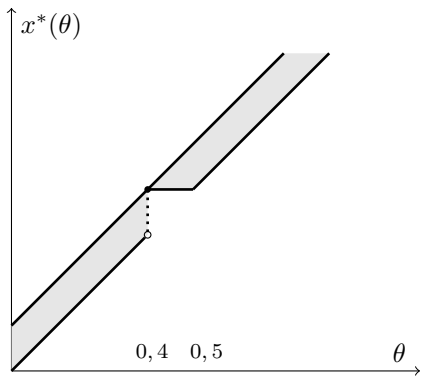
$$x^*(\theta) = \begin{cases} [\theta; \theta + 0,1] & \text{jeżeli } \theta < 0,4 \\ [0,5; \theta + 0,1] & \text{jeżeli } 0,4 \leq \theta < 0,5 \\ [\theta; \theta + 0,1] & \text{jeżeli } 0,5 \leq \theta \end{cases} \tag{1.13}$$

A funkcja wartości to:

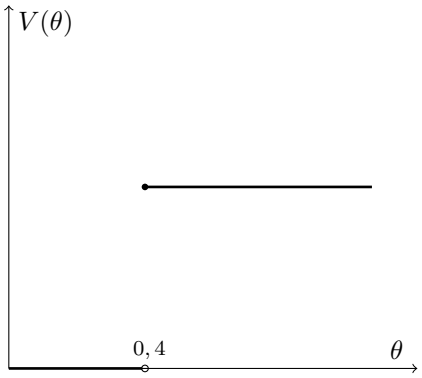
$$V(\theta) = \begin{cases} 0 & \text{jeżeli } \theta < 0,4 \\ 1 & \text{jeżeli } 0,4 \leq \theta < 0,5 \\ 1 & \text{jeżeli } 0,5 \leq \theta \end{cases} \tag{1.14}$$

Zatem korespondencja x^* nie jest górnie hemiciągła a funkcja wartości V nie jest ciągła.

Czy warunki twierdzenia maksimum mogą być wzmocnione tak, aby korespondencja x^* była ciągła a nie tylko górnie hemiciągła? Poniższy przykład pokazuje, że w ogólności się nie da:



(a) $x^*(\theta)$ nie jest GHC dla $\theta = 0,4$



(b) $V(\theta)$ nie jest ciągła dla $\theta = 0,4$

Wykres 1.6: Konsekwencje nieciągłości funkcji f.

Przykład 1.1. Rozważmy problem konsumenta z liniową funkcją użyteczności, który musi wybrać spośród dwóch dóbr. Funkcja celu przyjmuje postać $U(x) = x_1 + x_2$ a problem optymalizacji to:

$$\max_{x_1,x_2} U(x_1, x_2)$$
$$p.w. \mathbf{p}^T\mathbf{x} \leq 10$$

Niech $P \equiv \{\mathbf{p} \in \mathbf{R}^2_{++} | p_1 = 1,\ p_2 \in (0, \infty)\}$ oznacza zbiór możliwych cen (zbiór parametrów). Rozwiązanie problemu to popyt konsumenta jako funkcja cen i wynosi:

$$x^*(\mathbf{p}) = (x_1^*(p), x_2^*(p)) = \begin{cases} \{(0, 10/p_2)\} & \text{jeżeli } p_2 < 1 \\ \{\mathbf{x} | \mathbf{p}^T\mathbf{x} = 10\} & \text{jeżeli } p_2 = 1 \\ \{(10, 0)\} & \text{jeżeli } p_2 > 1 \end{cases} \tag{1.15}$$

Jak łatwo sprawdzić, korespondencja x^* nie jest ciągła, ponieważ x_1^* jako funkcja p_2 nie jest ciągła w punkcie $p_2 = 1$. Jednak, zgodnie z twierdzeniem x^* jest górnie hemiciągła.

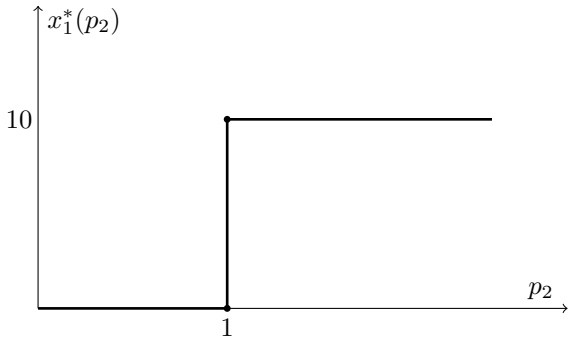
Twierdzenie maksimum identyfikuje warunki, przy których problemy optymalizacyjne mają rozwiązanie oraz stwierdza coś na temat ciągłości rozwiązania. Możemy powiedzieć więcej, jeśli znana jest krzywizna funkcji celu.

Twierdzenie 1.11. *Dana jest parametryczna rodzina problemów optymalizacji z ograniczeniami postaci:*

$$\max_{x \in D(\theta)} f(x, \theta)$$

zdefiniowane na wypukłym zbiorze parametrów Θ. Przyjmijmy, że:

- *korespondencja $D : \Theta \to X$ jest ciągła oraz jej wartości są zbiorami zwartymi oraz*



Wykres 1.7: Popyt nie jest ciągły

- *funkcja* $f : X \times \Theta \to \mathbf{R}$ *jest ciągła.*

Wówczas:

1. *Jeżeli* $f(\cdot, \theta)$ *jest funkcją (ściśle) quasi-wklęsłą*[2] *względem* x *dla każego* θ*, a wartości* D *są zbiorami wypukłymi, wówczas wartości* $x^*(\theta)$ *są zbiorami (jednoelementowymi*[3]*) wypukłymi.*

2. *Jeżeli* $f(\cdot, \theta)$ *jest funkcją (ściśle) wklęsłą względem* (x, θ)*, a wartości* D *są zbiorami wypukłymi, wówczas funkcja wartości* V *jest funkcją (ściśle) wklęsłą a wartości* $x^*(\theta)$ *są zbiorami (jednoelementowymi) wypukłymi.*

1.3.3. Charakterystyka rozwiązań - intuicja

Niniejszy rozdział ma służyć intuicyjnemu przedstawieniu idei twierdzenia Lagrange'a dla ograniczeń w postaci równości oraz twierdzenia Kuhn-Tuckera dla ograniczeń w postaci nierówności.

Dane jest następujące zadanie optymalizacyjne:

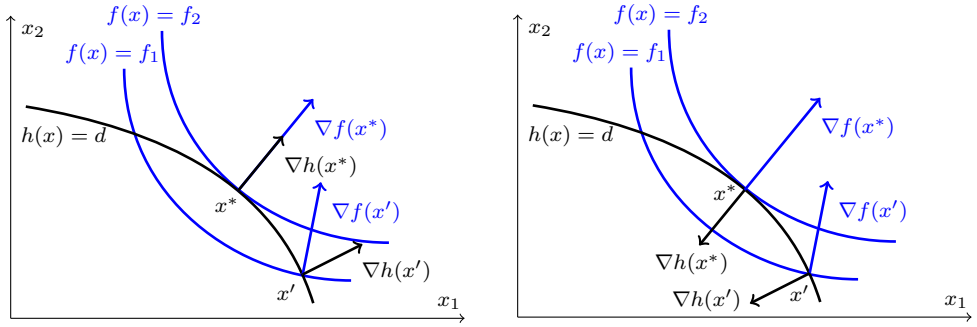
$$\max_{x \in D \subset \mathbf{R}^N} f(x)$$

Zbiór D jest zbiorem punktów dopuszczalnych. Zbiór ten będziemy przedstawiać za pomocą:

- ograniczeń w postaci równości $h(x) = d$ oraz
- ograniczeń w postaci nierówności $g(x) \leq c$.

[2] Funkcja $h : X \to \mathbf{R}$ jest quasiwklęsła, jeśli zbiory ponad krzywymi obojętności $x \in X : h(x) \geq k$ są wypukłe.

[3] Oznacza to, że x^* jest funckją.



(a) Dodatni mnożnik (b) Ujemny mnożnik

Wykres 1.8: W przypadku optymalizacji z ograniczeniami w postaci równań w punkcie optymalnym spełniony jest następujący warunek: $\nabla f(x^*) = \lambda \nabla h(x^*)$. Mnożnik Lagrange'a może być zarówno dodatni, jak i ujemny.

Powyższy problem należy do zawężonej wersji problemów parametrycznych rozważanych w poprzednim podrozdziale. Niemniej klasa problemów, które mogą być przedstawione w powyższy sposób jest bardzo szeroka. Na przykład ograniczenie w postaci $g(x) \geq c$ można przedstawić alternatywnie jako $-g(x) \leq -c$. Zauważmy, że ograniczenie $h(x) = d$ można przedstawić za pomocą dwóch ograniczeń typu "$\leq$"($h(x) \leq d$ oraz $-h(x) \leq -d$), jednak okazuje się, że ograniczenia w postaci równości lepiej analizować osobno.

W przypadku ograniczeń w postaci nierówności, mówimy, że dane ograniczenie jest aktywne bądź napięte w punkcie dopuszczalnym x jeśli zachodzi $g(x) = c$. W przeciwnym przypadku, tj. gdy $g(x) < c$ mówimy, że ograniczenie jest nieaktywne lub luźne w punkcie dopuszczalnym x. Oczywiście dla ograniczeń w postaci równości dla punktu dopuszczalnego ograniczenie z definicji musi być napięte. Intuicję dotyczącą zagadnień optymalizacyjnych najlepiej wyrobić sobie graficznie dla przypadku, gdy $x \in \mathbf{R}^2$.

Rozważmy problem maksymalizacji z jednym ograniczeniem. Wykresy 1.8a oraz 1.8b pokazują, że w punkcie optymalnym nachylenie funkcji celu oraz ograniczenia powinny być równe (punkt x^*). Jest to punkt, w którym nie da się zwiększyć wartości funkcji celu pozostając jednocześnie na krzywej, na której spełnione jest ograniczenie. W innych punktach, takich jak na przykład punkt x' wypadkowa gradientu funkcji celu wzdłuż krzywej ograniczenia jest dodatnia, a zatem można powiększyć wartość funkcji celu jednocześnie spełniając ograniczenie.

Możemy to zapisać w następujący sposób:

$$\nabla f(x^*) = \lambda \nabla h(x^*) \tag{1.16}$$

gdzie symbol λ oznacza mnożnik Lagrange'a.

Tak więc problem optymalizacji z ograniczeniem $\max_x f(x)$, p.w. $h(x) = d$ można sprowadzić do problemu optymalizacji bez ograniczeń funkcji $L(x, \lambda) = f(x) -$

$\lambda(h(x) - d)$, zwanej funkcją Lagrange'a, bowiem warunki pierwszego rzędu optymalizacji tej funkcji będą dokładnie równe warunkom (1.16) oraz ograniczeniu $h(x^*) = d$:

$$\nabla f(x^*) - \lambda \nabla h(x^*) = 0 \tag{1.17}$$
$$h(x^*) - d = 0 \tag{1.18}$$

Z powyższych rysunków (patrz wykresy 1.8a i 1.8b) wynika również, że w przypadku ograniczeń w postaci równości mnożnik Lagrange'a może być zarówno dodatni, jak i ujemny. Jedyna wymagana rzecz jest taka, że musi on być zdefiniowany. Aby mnożnik można było zdefiniować musi być spełniony następujący warunek zwany warunkiem ćonstraint qualification":

$$\nabla h(x^*) \neq 0 \tag{1.19}$$

Gradient ograniczenia w punkcie optymalnym musi być różny od wektora zerowego. W naszym przykładzie z dwoma zmiennymi oznacza to, że:

$$\begin{bmatrix} \frac{\partial h(x^*)}{\partial x_1} \\ \frac{\partial h(x^*)}{\partial x_2} \end{bmatrix} \neq \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Dla problemu optymalizacji z ograniczeniami w postaci nierówności $g(x) \leq c$ problem jest podobny, lecz nieco bardziej złożony. Jeśli optimum jest w punkcie, w którym dane ograniczenie nie jest aktywne, to warunek pierwszego rzędu jest taki sam jak, gdyby ograniczenia nie było tj.

$$\nabla f(x^*) = 0 \tag{1.20}$$

Warunek pierwszego rzędu funkcji Lagrange'a wynosi zaś:

$$\nabla f(x^*) - \lambda \nabla g(x^*) = 0$$

Aby te warunki się zgadzały, mnożnik Lagrange'a w tym przypadku musi wynosić zero.

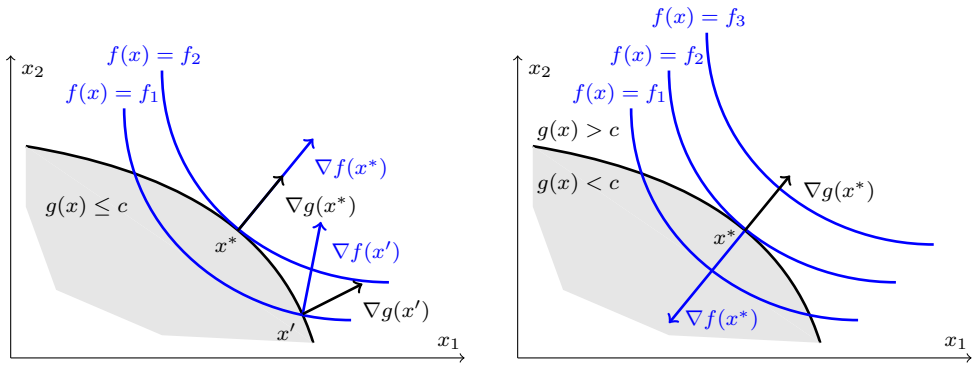
Jeśli zaś optimum jest w punkcie, w którym dane ograniczenie jest aktywne, wówczas nachylenie funkcji celu w punkcie optymalnym jest równe nachyleniu ograniczenia. Wykres 1.9a przedstawia punkt x^*, w którym ograniczenie jest aktywne, zatem nachylenie funkcji celu i ograniczenia musi być równe.

W przypadku ograniczeń w postaci nierówności mnożnik Lagrange'a musi być dodatni, ponieważ w przeciwnym razie moglibyśmy przesunąć punkt x^* do wewnątrz zbioru dopuszczalnego podnosząc wartość funkcji celu f - co przeczyłoby optymalności punktu x^* (patrz wykres 1.9b): Zatem mamy dwie sytuacje możliwe w punkcie optymalnym:

- ograniczenie aktywne, mnożnik dodatni: $g(x) = c$ i $\lambda > 0$
- ograniczenie nieaktywne, mnożnik zerowy: $g(x) < 0$ i $\lambda = 0$

Te dwa przypadki można zapisać jednym warunkiem zwanym ćomplementary slackness":

$$\lambda[g(x^*) - c] = 0$$



(a) Dodatni mnożnik

(b) Ujemny mnożnik

Wykres 1.9: Optymalizacja z ograniczeniami w postaci nierówności: w punkcie optymalnym, gdzie ogranicznie jest aktywne spełniony jest następujący warunek: $\nabla f(x^*) = \lambda \nabla g(x^*)$. Mnożnik Lagrange'a musi być dodatni.

Zastanówmy się nad interpretacją mnożnika Lagrange'a. Załóżmy, że rozwiązaliśmy problem optymalizacji $\max_x f(x)$ przy warunku ograniczającym $h(x) = d$ dla różnych wartości d[4]. Niech rozwiązania tego problemu będą przedstawione jako $\hat{x}(d) = (\hat{x}_1(d), \ldots, \hat{x}_n(d))$ oraz $\hat{\mu}(d)$. Niech dla wszystkich wartości $\hat{x}(d)$ będzie spełniony warunek ćonstraint qualification". Niech $\hat{f}(d) = f(\hat{x}(d))$. Wówczas:

$$\hat{f}'(d) = \nabla f(\hat{x}(d))\hat{x}'(d) = \hat{\mu}(d)\nabla g(\hat{x}(d))\hat{x}'(d)$$

wykorzystując warunek (1.17). Z kolei na mocy warunku (1.18) zachodzi:

$$\nabla g(\hat{x}(d))\hat{x}'(d) = 1, \quad \forall d$$

Stąd otrzymujemy:

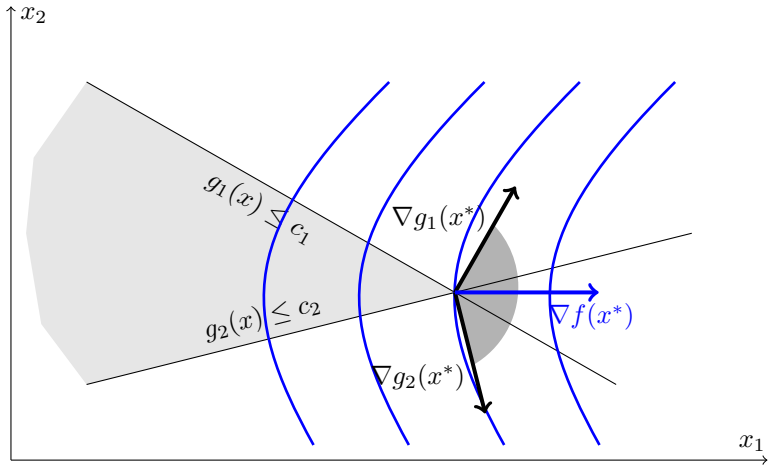
$$\hat{f}'(d) = \hat{\mu}(d)$$

A zatem wartość mnożnika Lagrange'a w punkcie optymalnym informuje o ile zmieni się optymalna wartość funkcji celu, jeśli prawa strona ograniczenia zmieni się nieznacznie.

Powróćmy do analizy warunków koniecznych optymalności. W przypadku więcej niż jednego ograniczenia, zarówno jeśli chodzi o ograniczenia w postaci równości, które w punkcie optymalnym muszą być aktywne z definicji[5], jak i ograniczenia w postaci nierówności, ale które w punkcie optymalnym są aktywne, gradient funkcji celu musi być kombinacją liniową gradientów ograniczeń aktywnych. Dla przypadku ograniczeń w postaci nierówności przedstawione to zostało na wykresie 1.10.

[4]Podobna analiza może zostać przeprowadzona dla problemów optymalizacji z ograniczeniami. Interpretacja jest tam taka sama.

[5]Punkt optymalny musi być punktem dopuszczalnym.



Wykres 1.10: Gradient funkcji celu w punkcie optymalnym jest równy kombinacji liniowej gradientów ograniczeń aktywnych.

W punkcie x^* oba ograniczenia $g_1(x) \leq c_1$ oraz $g_2(x) \leq c_2$ są aktywne, a zatem w tym punkcie gradient funkcji celu jest kombinacją liniową gradientów obu ograniczeń. Mnożniki Lagrange'a odpowiadające poszczególnym ograniczeniom muszą być dodatnie i występują tutaj w roli wag kombinacji liniowej.

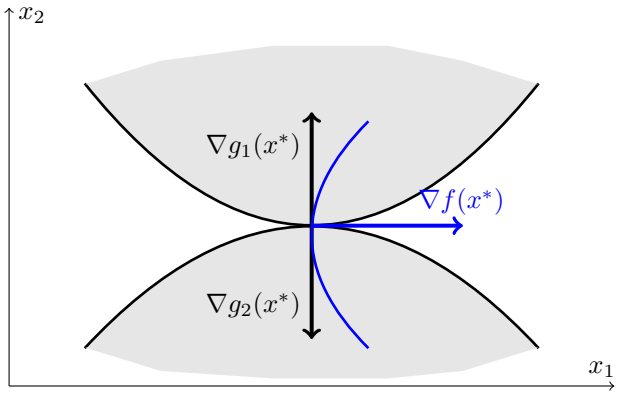
Oczywiście, aby gradient funckji celu dało się zapisać jako kombinację liniową gradientów ograniczeń aktywnych, te gradienty ograniczeń aktywnych powinny być liniowo niezależne. Inaczej możemy mieć do czynienia z sytuacją, przedstawioną na wykresie 1.11.

Jedyny punkt dopuszczalny to punkt x^*, zatem jest on zarazem punktem optymalnym. Jednakże nie da się zapisać $\nabla f(x^*)$ jako liniową kombinację gradientów $\nabla g_1(x^*)$ i $\nabla g_2(x^*)$. Zauważmy, że gradienty $\nabla g_1(x^*)$ i $\nabla g_2(x^*)$ nie są liniowo niezależne. Liniowa niezależność gradientów wszystkich ograniczeń w postaci równości oraz aktywnych ograniczeń w postaci nierówności jest zwana warunkiem ćonstraint qualification".

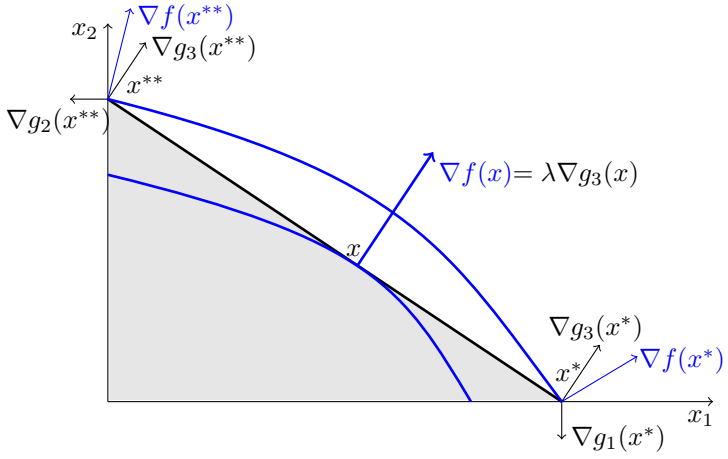
Warunek przedstawiony na wykresie 1.10 jest warunkiem koniecznym optymalności, ale nie jest warunkiem wystarczającym. Rozważmy sytuację przedstawioną na wykresie 1.12.

W punkcie x nachylenie funkcji celu jest równe nachyleniu ograniczenia aktywnego, jednak nie jest to punkt optymalny - optymalne są punkty x^{**} oraz x^*[6]. Aby wykluczyć taką możliwość należy sprawdzić warunki drugiego rzędu, lub warunki wystarczające optymalności. W ogólności weryfikacja warunków drugiego rzędu jest bardzo żmudna. Jednak istnieją pewne przypadki, w których nie trzeba sprawdzać warunków

[6]Zauważmy, że w punktach x^* oraz x^{**} gradient funkcji celu jest liniową kombinacją ograniczeń aktywnych.



Wykres 1.11: Gradienty ograniczeń aktywnych nie są niezależne. Gradient funkcji celu w punkcie optymalnym nie może być zapisany jako kombinacja liniowa gradientów ograniczeń aktywnych.



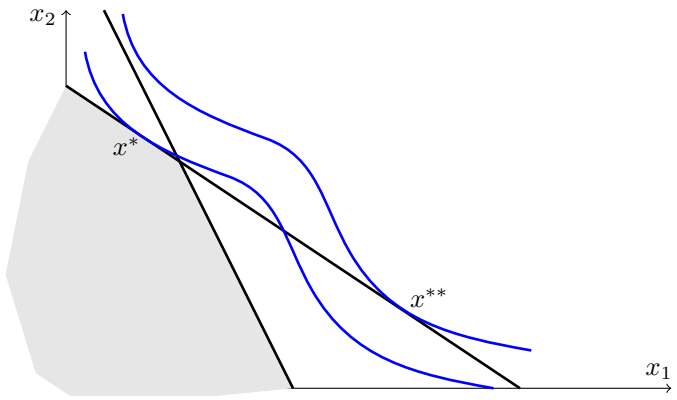
Wykres 1.12: Warunki Kuhn Tuckera nie są wystarczające dla punktu optymalnego.

drugiego rzędu, ponieważ warunki konieczne są również warunkami wystarczającymi optymalności.

W wielu praktycznych problemach optymalizacji wiadomo z wyprzedzeniem, które ograniczenia są aktywne w optimum, a które nie są. Czy możemy wówczas zignorować ograniczenia, które wiemy, że w optimum będą nieaktywne i rozwiązywać problem wyłącznie z aktywnymi ograniczeniami w postaci równości[7]? Okazuje się, że niestety

[7]Problemy z ograniczeniami w postaci równości zazwyczaj łatwiej jest rozwiązać.

nie można tak zrobić. Obrazuje to wykres 1.13.



Wykres 1.13: Nieaktywne ograniczenie nie może zostać zignorowane.

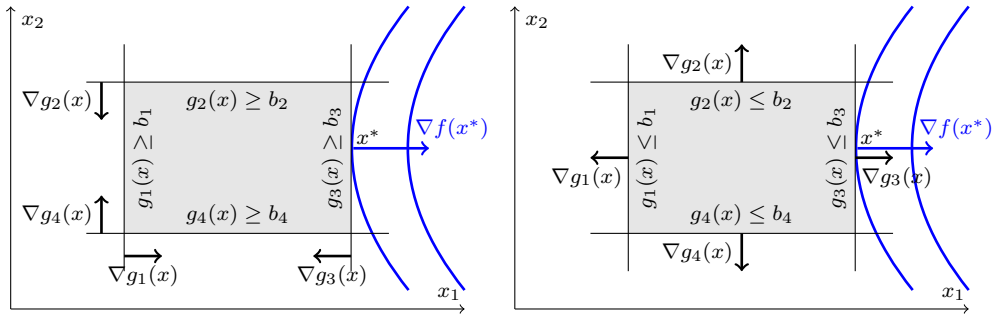
Punkt x^* jest punktem optymalnym, w którym jedno z ograniczeń jest aktywne a pozostałe jest nieaktywne. Jednak, gdybyśmy usunęli ograniczenie nieaktywne, punktem optymalnym stałby się punkt x^{**}, który poprzednio był wykluczony właśnie poprze to nieaktywne ograniczenie.

Wcześniej wspomnieliśmy, że problemy z ograniczeniami w postaci równości można przedstawić jako problemy z ograniczeniami w postaci nierówności. Na przykład ograniczenie $h(x) = d$ można przedstawić jako $h(x) \leq d$ i $-h(x) \leq -d$. Wspomnieliśmy również, że lepiej jednak te problemy w postaci równości analizować osobno. Dlaczego? Otóż zauważmy, że w punkcie spełniającym ogtraniczenie $h(x) = d$, obydwa ograniczenia $h(x) \leq d$ oraz $-h(x) \leq -d$ muszą być aktywne. Powyżej omówiliśmy warunek ćonstraint qualification", który zakłada, że gradienty ograniczeń aktywny w punkcie optymalnym muszą być niezależne. Jednak gradienty $\nabla h(x)$ oraz $-\nabla h(x)$ nie są liniowo niezależne, zatem warunek ćonstraint qualificationńie może być spełniony dla punktów spełniających jednocześnie $h(x) \leq d$ oraz $-h(x) \leq -d$. To jest powód, dla którego mamy oddzielną procedurę znajdowania optimum dla problemów z ograniczeniami w postaci równości i oddzielna procedurę dla problemów z ograczniczeniami w postaci nierówności.

Analiza problemów maksymalizacji funkcji celu z ograniczeniami w postaci "$\leq$"byłaby identyczna w przypadku problemów minimalizacji funkcji celu z ograniczeniami w postaci "$\geq$". W obu przypadkach mnożniki Lagrange'a w punkcie optymalnym muszą być nieujemne, przy czym dla ograniczeń aktywnych zazwyczaj są dodatnie[8]. Z kolei w przypadku analizy problemów maksymalizacji funkcji celu z ograniczeniami w postaci "$\geq$óraz minimalizacji funkcji celu z ograniczeniami w postaci "$\leq$"mielibyśmy do

[8]Istnieje rzadki przypadek, w którym ograniczenie w punkcie optymalnym jest aktywne, jednak punkt ten byłby optymalny również bez tego ograniczenia. Wówczas mimo, iż ograniczenie jest aktywne jego mnożnik Lagrange'a może przyjmować wartość 0.

czynienia z jedną modyfikacją: w tych przypadkach mnożniki Lagrange'a w punkcie optymalnym musiałyby być niedodatnie a nie nieujmne, przy czym dla ograniczeń aktywnych zazwyczaj byłyby ujemne. W sposób intuicyjny jest to przedstawione na wykresie 1.14.



(a) Ograniczenia postaci "$\geq$"

(b) Ograniczenia postaci "$\leq$"

Wykres 1.14: Dwa sposoby przedstawienia tego samego zbioru ograniczeń w postaci nierówności.

Rozważmy problem maksymalizacji z ograniczeniami w postaci "$\leq$"(patrz wykres 1.14b) oraz problem maksymalizacji z ograniczeniami w postaci "$\geq$"(patrz wykres 1.14a). Gradienty ograniczeń w przypadku ograniczeń w postaci "$\geq$"muszą być skierowane do wewnątrz zbioru dopuszczalnego, ponieważ w tym kierunku wartość funkcji $g_i(x)$ rośnie najszybciej i wobec tego ograniczenie jest w tym kierunku spełnione. Gradienty ograniczeń w przypadku ograniczeń w postaci "$\leq$"muszą być skierowane na zewnątrz zbioru dopuszczalnego, ponieważ w tym kierunku wartość funkcji $g_i(x)$ rośnie najszybciej i wobec tego ograniczenie jest spełnione w przeciwnym kierunku. Na wykresach widać, że w punkcie optymalnym x^* gradient funkcji celu musi być skierowany w tym samym kierunku co gradient ograniczenia aktywnego, jeśli mamy do czynienia z maksymalizacją przy ograniczeniach postaci "$\leq$"i w odwrotnym kierunku, jeśli mamy do czynienia z maksymalizacją przy ograniczeniach postaci "$\geq$".

Podobny argument można przeprowadzić dla problemów minimalizacji. Tabela 1.1 przedstawia porównanie wszystkich czterech przypadków.

1.3.4. Twierdzenia

W niniejszym podrozdziale zostaną zaprezentowane następujące rezultaty:

- Dla problemów optymalizacji z ograniczeniami w postaci równań:
  - Warunki konieczne optimum (Twierdzenie Lagrange'a)
  - Warunki wystarczające dla ekstremów lokalnych
  - Warunki wystarczające dla optimum globalnego

Problem	Kierunek gradientu ograniczeń	Mnożniki Lagrange'a
Max z ograniczeniami "$\geq$"	do wewnątrz	niedodatnie
Min z ograniczeniami "$\geq$"	do wewnątrz	nieujemne
Max z ograniczeniami "$\leq$"	na zewnątrz	nieujemne
Min z ograniczeniami "$\leq$"	na zewnątrz	niedodatnie

Tabela 1.1: Porównanie problemów maksymalizacji oraz problemów maksymalizacji przy ograniczeniach w postaci "$\geq$óraz "$\leq$".

- Dla problemów optymalizacji z ograniczeniami w postaci nierówności:
  - Warunki konieczne optimum (Twierdzenie Kuhn-Tuckera)
  - Warunki wystarczające dla optimum globalnego

Dodatkowo na koniec tego podrozdziału przedstawimy twierdzenie, które identyfikuje warunki konieczne optimum dla problemów w ograniczeniami mieszanymi (część w postaci równań, część w postaci nierówności).

Zaczniemy od twierdzenia Lagrange'a, które identyfikuje warunki konieczne, które muszą być spełnione przez każde rozwiązanie problemu optymalizacyjnego z ograniczeniami w postaci równości.

Dane są funkcje $f : \mathbf{R}^N \to \mathbf{R}$, $h_k : \mathbf{R}^N \to \mathbf{R}$, gdzie $k = 1, ..., K$. Rozważmy problem nastepującej postaci [9]:

$$\begin{aligned} & \max_{\mathbf{x}} f(\mathbf{x}) \\ \text{p.w.} \quad h_k(\mathbf{x}) &= d_k, k = 1, ..., K; \end{aligned}$$

Zdefiniujmy funkcję Lagrange'a tego problemu jako

$$L(\mathbf{x}, \mu) = f(\mathbf{x}) - \sum_{k=1}^{K} \mu_{\mathbf{k}}[h_k(\mathbf{x}) - d_k]$$

gdzie $\mu = (\mu_1, \ldots, \mu_K)$.

Twierdzenie 1.12 (Twierdzenie Lagrange'a)**.** *Rozważmy problem optymalizacyjny z $K \leq N$ ograniczeniami w postaci równości. Niech spełnione będą następujące warunki:*

1. *Funkcje $f(\cdot)$ oraz $h_k(\cdot)$ są różniczkowalne w sposób ciągły*
2. *Punkt $\mathbf{x}^*$ jest rozwiązaniem problemu optymalizacyjnego*
3. *Warunek ćonstraint qualification"jest spełniony w punkcie $\mathbf{x}^*$, czyli wektory $\nabla h_k(\mathbf{x}^*)$, dla $k = 1, 2, \ldots K$ muszą być liniowo niezależne.*

[9] Twierdzenie jest ważne również w przypadku, gdy operator max zamienimy na operator min

Wówczas istnieją liczby $\mu_1, \mu_2, \ldots, \mu_K$, takie, że spełniony jest następujący warunek:

$$\nabla f(\mathbf{x}^*) = \sum_{k=1}^{K} \mu_k \nabla h_k(\mathbf{x}^*)$$

Powyższe twierdzenie identyfikuje warunki konieczne, ale nie wystarczające dla punktu optymalnego. Poniżej przedstawione są warunki wystarczające dla minimum i maksimum lokalnego.

Twierdzenie 1.13. *Niech będą spełnione następujące warunki:*

1. *Funkcje $f(\cdot)$ oraz $h_k(\cdot)$, gdzie $k = 1, 2, \ldots, K$ są podójnie różniczkowalne w sposób ciągły.*
2. *Punkt $\mathbf{x}^*$ spełnia warunki konieczne optymalizacji*[10].

Zdefiniujmy Hesjan ograniczony jako:

$$|H_r| = \det \begin{pmatrix} 0 & \ldots & 0 & \frac{\partial h_1}{\partial x_1} & \ldots & \frac{\partial h_1}{\partial x_r} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \ldots & 0 & \frac{\partial h_K}{\partial x_1} & \ldots & \frac{\partial h_K}{\partial x_r} \\ \frac{\partial h_1}{\partial x_1} & \ldots & \frac{\partial h_K}{\partial x_1} & \frac{\partial^2 L}{\partial x_1 \partial x_1} & \ldots & \frac{\partial^2 L}{\partial x_1 \partial x_r} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h_1}{\partial x_r} & \ldots & \frac{\partial h_K}{\partial x_r} & \frac{\partial^2 L}{\partial x_r \partial x_1} & \ldots & \frac{\partial^2 L}{\partial x_r \partial x_r} \end{pmatrix}, \quad r = 1, 2, ..., N$$

gdzie $L(\mathbf{x}, \mu)$ jest funkcją Lagrange'a ($\mathbf{x} = (x_1, \ldots, x_N)$, $\mu = (\mu_1, \ldots, \mu_K)$). Niech $|H_r(\mathbf{x}^*)|$ będzie Hesjanem ograniczonym dla punktu $\mathbf{x}^*$. Wówczas:

1. *Jeśli $(-1)^r |H_r(\mathbf{x}^*)| > 0$, dla $r = K+1, \ldots, N$, wówczas $\mathbf{x}^*$ jest maksimum lokalnym.*
2. *Jeśli $(-1)^K |H_r(\mathbf{x}^*)| > 0$, dla $r = K+1, \ldots, N$, wówczas $\mathbf{x}^*$ jest minimum lokalnym.*

Jest jeszcze jeden rezultat, który identyfikuje warunki wystarczające dla minimum i maksimum globalnego.

Twierdzenie 1.14. *Niech będą spełnione następujące warunki:*

1. *Funkcje $f(\cdot)$ oraz $h_k(\cdot)$, gdzie $k = 1, \ldots, K$ będą zdefiniowane na otwartym i wypukłym zbiorze $S \subset \mathbf{R}^N$.*
2. *Punkt $\mathbf{x}^*$ spełnia warunki konieczne optymalizacji*[11].

Wówczas:

[10] Są to warunki z twierdzenia Lagrange'a.
[11] Są to warunki z twierdzenia Lagrange'a.

1. *Jeśli funkcja Lagrange'a jest wklęsła, punkt* $\mathbf{x}^*$ *jest globalnym maksimum.*

2. *Jeślu funkcja Lagrange'a jest wypukła, punkt* $\mathbf{x}^*$ *jest globalnym minimum.*

Przykład 1.2. Rozważmy przykład optymalizacji z ograniczeniem w postaci równania:

$$\begin{aligned} &\max_{x,y} x^2 y \\ &p.w.\ \ 2x^2 + y^2 = 3 \end{aligned}$$

Zauważmy, że funkcja celu jest ciągła a zbiór dopuszczalny jest zwarty. Zatem rozwiązanie istnieje. Funkcja Lagrange'a to $L(x, y, \mu) = x^2 y - \mu(2x^2 + y^2 - 3)$. Warunki pierwszego rzędu są następujące:

$$\begin{aligned} 2xy - 4\mu x = 2x(y - 2\mu) &= 0 \\ x^2 - 2\mu y &= 0 \end{aligned}$$

Oraz ograniczenie $2x^2 + y^2 - 3 = 0$. Jest sześć rozwiązań tego układu równań:

1. $(x, y, \mu) = (0, \sqrt{3}, 0)$, $f(x, y) = 0$
2. $(x, y, \mu) = (0, -\sqrt{3}, 0)$, $f(x, y) = 0$
3. $(x, y, \mu) = (1, 1, 1/2)$, $f(x, y) = 1$
4. $(x, y, \mu) = (1, -1, -1/2)$, $f(x, y) = -1$
5. $(x, y, \mu) = (-1, 1, 1/2)$, $f(x, y) = 1$
6. $(x, y, \mu) = (-1, -1, -1/2)$, $f(x, y) = -1$

Sprawdzamy warunek constraint qualification $\nabla g(x, y) = (4x, 2y)$. Jest to wektor zerowy tylko wówczas, gdy $x = y = 0$, a zatem jedynymi możliwymi kandydatami na rozwiązanie problemu jest sześć przypadków wyszczególnionych powyżej. Czyli minima globalne to przypadek 4. i 6. a maksima globalne to przypadek 3. i 5. A co z przypadkami 1. i 2.? Dla tych przypadków należy sprawdzić wyznacznik ograniczonego Hesjana dla $r = 2$:

$$|H_r(x, y, \mu)| = \begin{vmatrix} 0 & 4x & 2y \\ 4x & 2y - 4\mu & 2x \\ 2y & 2x & -2\mu \end{vmatrix}$$

Ten wyznacznik wynosi

$$\begin{aligned} |H_r(x, y, \mu)| &= 32x^2 y - 4y^2(2y - 4\mu) + 32x^2\mu \\ &= 8[2\mu(2x^2 + y^2) + y(4x^2 - y^2)] \\ &= 8[6\mu + y(4x^2 - y^2)] \end{aligned}$$

gdzie ostatnia równość jest na mocy warunku $2x^2 + y^2 = 3$. Wartość wyznacznika jest zatem:

1. $|H_r(0, \sqrt{3}, 0)| = -8\sqrt{3}$, czyli punkt $(0, \sqrt{3})$ jest lokalnym minimum;

2. $|H_r(0, \sqrt{3}, 0)| = 8\sqrt{3}$, czyli punkt $(0, -\sqrt{3})$ jest lokalnym maksimum;

Teraz z kolei przedstawimy twierdzenie Kuhn-Tuckera, które identyfikuje warunki konieczne, które muszą być spełnione przez każde rozwiązanie problemu optymalizacyjnego z ograniczeniami w postaci nierówności.

Dane są funkcje $f : \mathbf{R}^N \to \mathbf{R}$, $g_j : \mathbf{R}^N \to \mathbf{R}$, gdzie $j = 1, ..., J$. Rozważmy problem nastepującej postaci [12]:

$$\begin{aligned} & \max_{\mathbf{x}} f(\mathbf{x}) \\ \text{p.w.} \quad g_j(\mathbf{x}) \quad & \leq \quad c_j, j = 1, ..., J \end{aligned}$$

Zdefiniujmy funkcję Lagrange'a tego problemu jako

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) - \sum_{j=1}^{J} \lambda_j [g_j(\mathbf{x}) - c_j]$$

gdzie $\lambda = (\lambda_1, \ldots, \lambda_J)$.

Twierdzenie 1.15 (Twierdzenie Kuhn-Tuckera)**.** *Rozważmy problem optymalizacyjny z $J \leq N$ ograniczeniami w postaci nierówności. Niech spełnione będą następujące warunki:*

1. *Funkcje $f(\cdot)$ oraz $g_j(\cdot)$ są różniczkowalne w sposób ciągły*

2. *Punkt $\mathbf{x}^*$ jest rozwiązaniem problemu optymalizacyjnego*

3. *Warunek ćonstraint qualification"jest spełniony w punkcie $\mathbf{x}^*$, czyli: niech $I(\mathbf{x}^*) = \{j : \ g_j(\mathbf{x}^*) = c_j\}$ oznacza zbiór aktywnych ograniczeń w postaci nierówności. Wówczas wektory $\nabla g_j(\mathbf{x}^*)$ dla $j \in I$ muszą być liniowo niezależne.*

Wówczas:

1. *Istnieją nieujemne liczby $\lambda_1, \lambda_2, \ldots, \lambda_J$, takie, że spełniony jest następujący warunek:*

$$\nabla f(\mathbf{x}^*) = \sum_{j=1}^{J} \lambda_j \nabla g_j(\mathbf{x}^*)$$

2. *Dla $j = 1, 2, \ldots J$, spełniony jest warunek:*

$$\lambda_j [c_j - g_j(\mathbf{x}^*)] = 0$$

[12] Twierdzenie jest ważne również w przypadku, gdy operator max zamienimy na operator min oraz ograniczenia w postaci "≤źamienimy na ograniczenia w postaci "≥".

Twierdzenie Kuhn-Tuckera identyfikuje tylko warunki konieczne ekstremum. Warunki wystarczające na istnienie rozwiązania problemu wymagają sprawdzenia tzw. warunków drugiego rzędu, które w ogólności są bardzo żmudne do sprawdzenia. Istnieją jednak szczególne przypadki, które na szczęście zdarzają się często w praktycznych zastosowaniach, kiedy nie trzeba sprawdzać warunków drugiego rzędu. Następujące twierdzenie identyfikuje warunki wystarczające na istnienie optimum problemu z ograniczeniami w postaci nierówności:

Twierdzenie 1.16 (Warunki wystarczające dla twierdzenia KT)**.** *Niech będą spełnione następujące warunki:*

1. *Funkcja $f(\cdot)$ jest quasi-wklęsła oraz*
2. *Funkcje $g_j(\cdot)$, gdzie $j = 1, 2, \ldots, J$ są quasi-wypukłe.*

Wówczas każdy punkt $\mathbf{x}^*$, który spełnia warunki Kuhn-Tuckera[13] *jest rozwiązaniem problemu optymalizacyjnego.*

Twierdzenie Lagrange'a oraz twierdzenie Kuhn-Tuckera są do siebie podobne, jednak różnią się w dwóch istotnych aspektach. Mnożniki Lagrange'a w twierdzeniu Lagrange'a mogą być dodatnie, ujemne bądź zerowe, podczas gdy w twierdzeniu Kuhn-Tuckera muszą być nieujemne. Twierdzenie Kuhn-Tuckera zawiera dodatkowo warunki zwane ćomplementary slackness", które mówią, że mnożnik Lagrange'a może być dodatni tylko wówczas, gdy dane ograniczenie jest aktywne, w przeciwnym przypadku wynosi zero.

Twierdzenie Lagrange'a dotyczy ograniczeń w postaci równości podczas gdy twierdzenie Kuhn-Tuckera dotyczy ograniczeń w postaci nierówności. Oba twierdzenia można połączyć w jedno, które dotyczy ograniczeń zarówno jednego, jak i drugiego typu.

Dane są funkcje $f : \mathbf{R}^N \to \mathbf{R}$, $g_j : \mathbf{R}^N \to \mathbf{R}$, gdzie $j = 1, ..., J.$ oraz $h_k : \mathbf{R}^N \to \mathbf{R}$, gdzie $k = 1, ..., K$. Ogólny problem optymalizacyjny jest następujący [14]:

$$\begin{array}{rrl} & & \max\limits_{\mathbf{x}} f(\mathbf{x}) \\ \text{p.w.} & g_j(\mathbf{x}) & \leq \quad c_j, j = 1, ..., J; \\ & h_k(\mathbf{x}) & = \quad d_k, k = 1, ..., K; \end{array}$$

Zdefiniujmy funkcję Lagrange'a tego problemu jako

$$L(\mathbf{x}, \mu, \lambda) = f(\mathbf{x}) - \sum_{k=1}^{K} \mu_k [h_k(\mathbf{x}) - d_k] - \sum_{j=1}^{J} \lambda_j [g_j(\mathbf{x}) - c_j]$$

gdzie $\mu = (\mu_1, \ldots, \mu_K)$ oraz $\lambda = (\lambda_1, \ldots, \lambda_J)$.

[13] Patrz warunek 1 i 2 w twierdzeniu Kuhn-Tuckera.

[14] Znowu problem może być zmodyfikowany na problem minimalizacji z ograniczeniami w postaci nierówności zmienionymi z "≤ńa "≥"

Twierdzenie 1.17. *Rozważmy problem optymalizacyjny z J ograniczeniami w postaci nierówności oraz K ograniczeniami w postaci równości, gdzie $J + K \leq N$. Niech spełnione będą następujące warunki:*

1. *Funkcje $f(\cdot)$, $g_j(\cdot)$ oraz $h_k(\cdot)$ są różniczkowalne w sposób ciągły*
2. *Punkt $\mathbf{x}^*$ jest rozwiązaniem problemu optymalizacyjnego*
3. *Warunek ćonstraint qualification"jest spełniony w punkcie $\mathbf{x}^*$, czyli: niech $I(\mathbf{x}^*) = \{j : g_j(\mathbf{x}^*) = c_j\}$ oznacza zbiór aktywnych ograniczeń w postaci nierówności. Wówczas wektory $\nabla g_j(\mathbf{x}^*)$ dla $j \in I$ oraz $\nabla h_k(\mathbf{x}^*)$, dla $k = 1, ..., K$ muszą być liniowo niezależne.*

Wówczas:

1. *Istnieją liczby rzeczywiste $\mu_1, \mu_2, \ldots, \mu_k$ oraz nieujemne liczby $\lambda_1, \lambda_2, \ldots, \lambda_J$, takie, że spełniony jest następujący warunek:*

$$\nabla f(\mathbf{x}^*) = \sum_{k=1}^{K} \mu_k \nabla h_k(\mathbf{x}^*) + \sum_{j=1}^{J} \lambda_j \nabla g_j(\mathbf{x}^*)$$

2. *Dla $j = 1, 2, \ldots J$, spełniony jest warunek:*

$$\lambda_j [c_j - g_j(\mathbf{x}^*)] = 0$$

1.3.5. Przykłady i zastosowania

W niniejszej części przedstawione są przykłady zastosowań teorii optymalizacji funkcji wielu zmiennych z ograniczeniami. Pierwsze dwa przykłady są zastosowaniami twierdzenia Lagrange'a dla problemów optymalizacji z ograniczeniami w postaci równań: jeden dotyczy optymalnej alokacji mocy w generatorach prądu, drugi jest problemem maksymalizacji entropii przy zadanych warunkach. Następne dwa problemy są zastosowaniami twierdzenia Kuhn-Tuckera dla problemów optymalizacji z ograniczeniami w postaci nierówności: jeden dotyczy maksymalizacji użyteczności konsumenta o preferencjach quasilinearnych przy zadanych ograniczeniach budżetowych, drugi natomiast wyznacza optymalny poziom cen usług telefonicznych w szczycie i poza nim.

Minimalizacja kosztów

Trzy generatory obsługują łączne obciążenie 952megawatów. Koszty wytworzenia x_i mocy wyjściowej przez generator i to:

$$f_1 : x_1 + 0.0625x_1^2 \text{PLN/h}$$
$$f_2 : x_2 + 0.0125x_2^2 \text{PLN/h}$$
$$f_3 : x_3 + 0.0250x_3^2 \text{PLN/h}$$

Jednostka mocy wyjściowej to MW. Ponieważ $1W = 1J/1s$ a koszty wytworzenia $1J$ są mierzone w PLN, to koszt wytworzenia mocy wyjściowej f_i można mierzyć w jednostkach PLN/h. Chcemy wyznaczyć przydział mocy wyjściowej trzech generatorów, który minimalizuje koszt łączny:

$$\begin{aligned} &\min f = f_1 + f_2 + f_3 = x_1 + 0.0625x_1^2 + x_2 + 0.0125x_2^2 + x_3 + 0.0250x_3^2 \\ &p.w. \;\; x_1 + x_2 + x_3 = 952 \end{aligned}$$

Funkcja Lagrange'a dla tego problemu to:

$$L(\mathbf{x}, \lambda) = x_1 + 0.0625x_1^2 + x_2 + 0.0125x_2^2 + x_3 + 0.0250x_3^2 - \lambda(x_1 + x_2 + x_3 - 952)$$

Warunki Lagrange'a przyjmują zatem postać:

$$\begin{pmatrix} 0.125 & 0 & 0 & -1 \\ 0 & 0.025 & 0 & -1 \\ 0 & 0 & 0.05 & -1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \lambda \end{pmatrix} = \begin{pmatrix} -1 \\ -1 \\ -1 \\ 952 \end{pmatrix}$$

Rozwiązaniem tego układu równań jest:

$$\begin{aligned} x_1 &= 112 \; MW \\ x_2 &= 560 \; MW \\ x_3 &= 280 \; MW \\ \lambda &= 15 \text{ PLN}/MWh \end{aligned}$$

A funkcja celu wynosi w tym punkcie $f = 7616$ PLN/h. Wartość λ informuje o tym, o ile wzrośnie koszt całkowity, gdybyśmy chcieli wyprodukować o jedną jednostkę więcej łącznego obciążenia (na przykład 953 zamiast 952 megawatów). Wartość ta to maksymalna cena, jaką firma będzie chciała zapłacić za wytworzenie dodatkowej jednostki mocy. Zauważmy, że w punkcie optymalnym zachodzi:

$$\lambda = 1 + 0.125x_1 = 1 + 0.025x_2 + 1 + 0.05x_3$$

Maksymalizacja entropii

Niech $\mathbf{g}$ będzie zmienną losową przyjmującą następujące wartości $g_i,\ i = 1, ..., n$ z odpowiednimi prawdopodobieństwami $p_i,\ i = 1, ..., n\ \sum_{i=1}^{n} p_i = 1$. Entropia informacyjna rozkładu prawdopodobieństw zdefiniowana jest następująco:

$$H(p_1, ..., p_n) = \sum_{i=1}^{n} p_i \log(1/p_i)$$

gdzie $0 \log 0 = 0$.

Podstawa logarytmu w powyższym wyrażeniu może być dobrana dowolnie i odzwierciedla ona wybór jednostki pomiaru entropii. Na przykład jeśli podstawa logarytmu wynosi 2, to wówczas entropia mierzona jest w bitach. Entropia mierzy niepewność rozkładu prawdopodobieństwa i osiąga maksimum, gdy $p_1 = p_2 = ... = p_n = 1/n$,

czyli gdy rozkład jest jednostajny. Zauważmy, że rozkład jednostajny jest najmniej informatywny, w sensie, że najmniej da się przewidzieć, która wartość zmiennej losowej **g** zostanie wylosowana.

Maksymalizacja entropii może być zastosowana, aby wybrać nieznany rozkład prawdopodobieństwa, spośród rozkładów zgodnych z daną informacją, która jest znana (np. średnia rozkładu). Idea jest taka, aby zakładać jak najmniej informacji poza to, co zostało zaobserwowane.

Rozważmy następujący problem. Dana jest średnia zmiennej losowej **g**, która wynosi G. Zgodnie z zasadą maksymalnej entropii możemy skonstruować gęstość rozkładu prawdopodobieństwa tej zmiennej poprzez maksymalizację entropii:

$$\begin{aligned} \max_{p_i \geq 0} \quad & \sum_{i=1}^{n} p_i \ln(1/p_i) \\ p.w. \quad & \sum_{i=1}^{n} p_i = 1 \\ & \sum_{i=1}^{n} p_i g_i = G \end{aligned}$$

Funkcja Lagrange'a przyjmuje postać:

$$L(\mathbf{p}, \lambda) = \sum_{i=1}^{n} p_i \ln(1/p_i) + \lambda_1 (1 - \sum_{i=1}^{n} p_i) + \lambda_2 (G - \sum_{i=1}^{n} p_i g_i)$$

A zatem warunki pierwszego rzędu [15] to:

$$\frac{\partial L}{\partial p_i} = \ln(1/p_i) - 1 - \lambda_1 - \lambda_2 g_i = 0, \quad \forall i \in \{1, 2, ..., n\} \tag{1.21}$$

$$\frac{\partial L}{\partial \lambda_1} = 1 - \sum_{i=1}^{n} p_i = 0 \tag{1.22}$$

$$\frac{\partial L}{\partial \lambda_2} = G - \sum_{i=1}^{n} p_i g_i = 0 \tag{1.23}$$

Warunek (1.21) można zapisać jako $p_i = e^{-1-\lambda_1-\lambda_2 g_i}$, dla każdego $i \in \{1, 2, ..., n\}$. Warunek (1.22) będzie spełniony, jeśli $e^{\lambda_1 - 1} = \sum_{i=1}^{n} e^{-\lambda_2 g_i}$. Wówczas:

$$p_i = \frac{e^{-\lambda_2 g_i}}{\sum_{i=1}^{n} e^{-\lambda_2 g_i}}$$

Warunek (1.23) przyjmuje zatem postać:

$$\frac{\sum_{i=1}^{n} g_i e^{-\lambda_2 g_i}}{\sum_{i=1}^{n} e^{-\lambda_2 g_i}}$$

[15] W tej klasie problemów, warunek nieujemności dla zmiennych p_i jest automatycznie spełniony i nie trzeba się nim przejmować.

A zatem wartość λ_2 jest rozwiązaniem następującego równania:

$$\sum_{i=1}^{n}(g_i - G)e^{-\lambda_2 g_i} = 0$$

które często rozwiązuje się numerycznie.

Przykładem zastosowania powyższego problemu jest wyznaczanie prawdopodobieństw, kiedy nie możemy ich wyznaczyć dokładnie. Rozważmy sytuację, w której są trzy możliwe zdarzenia: $g_1 = 1$, $g_2 = 2$ oraz $g_3 = 3$ a średnia wynosi $2,5$. Lagrangian problemu maksymalizacji entropii dla tego problemu może być zapisany następująco:

$$L(\mathbf{p}, \lambda) = \max_{\mathbf{p}} \left[-\sum_{i=1}^{3} p_i \log p_i + \lambda_1 \left(1 - \sum_{i=1}^{3} p_i \right) + \lambda_2 \left(2,5 - \sum_{i=1}^{3} p_i g_i \right) \right]$$

Warunki pierwszego rzędu dają:

$$p_i = e^{1-\lambda_1-\lambda_2 g_i}, \quad \text{dla } i = 1, 2, 3$$

oraz

$$\lambda_1 = 1,987$$
$$\lambda_2 = -0,834$$

A zatem prawdopodobieństwa maksymalizujące entropię to:

$$p_1 = 0,116$$
$$p_2 = 0,268$$
$$p_3 = 0,616$$

Powyższy przykład sugeruje ogólny sposób przydzielania prawdopodobieństw maksymalizujących entropię:

$$p(x_i) = \frac{\exp\left[-\sum_{j=1}^{m} \lambda_j f_j(x_i)\right]}{\sum_{i=1}^{n} \exp\left[-\sum_{j=1}^{m} \lambda_j f_j(x_i)\right]} = \frac{k(x_i)}{Z(\lambda_1, \cdots, \lambda_m)}$$

gdzie $f_j(x_i)$ jest funkcją zmiennej losowej x_i, która zawiera informację o tym, co wiemy na temat rozkładu $\mathbf{x}$, funkcja $k(x_i)$ nazywana jest "kernelą czynnik normalizujący $Z(\lambda_1, \cdots, \lambda_m)$ nazywany jest "partition function".

Preferencje quasiliniowe

Dane są dwa dobra x i y, których ilości są nieujemne a ceny, odpowiednio p i q, są dodatnie. Rozważmy konsumenta, którego dochód wynosi w a preferencje reprezentowane są przez następującą funkcję użyteczności:

$$U(x, y) = y + a \ln(x),$$

gdzie a jest dodatnim parametrem. Takie preferencje są nazywane quasiliniowymi, ponieważ funkcja użyteczności może być dobrana tak, aby była liniowa względem ilości jednego z dóbr. Spróbujmy znaleźć fukcję popytu tego konsumenta.

W ty celu zastosujemy twierdzenie Kuhn-Tuckera. Lagrangian dla tego problemu to:

$$L(x, y, \lambda) = y + a\ln(x) + \lambda[w - px - qy]$$

Warunki pierwszego rzędu to:

$$\begin{aligned}
a/x - \lambda p &\leq 0,\ x \geq 0, && (1.24)\\
1 - \lambda q &\leq 0,\ y \geq 0, && (1.25)\\
w - px - qy &\geq 0,\ \lambda \geq 0, && (1.26)\\
x[a/x - \lambda p] &= 0, && (1.27)\\
y[1 - \lambda q] &= 0, && (1.28)\\
\lambda[w - px - qy] &= 0, && (1.29)
\end{aligned}$$

Mamy dwie zmienne nieujemne i jedno ograniczenie w postaci nierówności. Daje to $2^3 = 8$ możliwości równań i nierówności. Które z nich powinny być wzięte pod uwagę?

Zauważmy, że ograniczenie budżetowe nie może być luźne. Pieniądze muszą zostać wydane, ponieważ dobra mają dodatnią krańcową użyteczność. Formalnie, gdyby ograniczenie budżetowe było luźne, wówczas $\lambda = 0$. Wtedy warunki 1.24 oraz 1.25 miałyby postać, odpowiednio: $a/x \leq 0$ oraz $1 \leq 0$, co jest niemożliwe.

To redukuje liczbę możliwych przypadków do czterech. Jednak x i y nie mogą być jednocześnie równe zero z tego samego powodu, co powyżej. Zatem mamy do rozważenia tylko trzy przypadki:

Jeśli $x = 0$ i $y = w/q$, wówczas na mocy warunku 1.25 otrzymujemy $\lambda = 1/q$, co daje warunek 1.24 postaci: $p/q \geq \infty$, co jest nieprawdziwe. Inuicyjnie, ponieważ pierwsza mała jednostka dobra x ma nieskończoną użyteczność krańcową, nie może być tak, że w optimum konument nie będzie konumował tego dobra.

Jeśli $y = 0$ i $x > 0$, wówczas na mocy ograniczenia budżetowego otrzymujemy $x = w/p$, natomiast na mocy warunku 1.24 otrzymujemy $\lambda = a/w$. Wstawiając do warunku 1.25, otrzymujemy $w \leq aq$. Ponieważ warunek ten nie jest niespójny z innymi warunkami, otrzymujemy kandydata na optimum.

Ostatecznie, rozważmy przypadek, gdzie x i y są jednocześnie dodatnie. Na mocy warunków 1.24 i 1.25 otrzymujemy $x = aq/p$. Wówczas z ograniczenia budżetowego: $y = w/q - a$. To jest logicznie spójne, gdy $w > aq$. Jest to drugi kandydat na optimum.

To kończy omawianie przypadków. Wnioski są następujące. Po pierwsze, sześć spośród ośmiu przypadków można było wykluczyć za pomocą inuicji ekonomicznej.

Następnie, zauważmy, że zbiór możliwych wartości p, q i w jest podzielony na dwa wyczerpujące oraz wyłączające się zbiory poprzez warunki nałożone na parametry modelu, które czynią dwa analizowane przypadki logicznie niesprzecznymi. Jeden sprowadza się do $w \leq aq$ a drugi do $w > aq$. Taka klasyfikacja jest typowa dla wielu zastosowań.

Po trzecie, jeśli $w = aq$, mamy $\lambda = 1/q$. Wówczas $1 - \lambda q = 0$ i $y = 0$, obie nierówności w 1.25 są spełnione jako równości. To jest przypadek szczególny, który zachodzi tylko w takiej konfiguracji parametrów, gdzie jeden przypadek nierówności i równości przechodzi w drugi przypadek.

W końcu, spróbujmy podsumować powyższą analizę. Optymalna reguła wyboru konsumenta to:

- Jeżeli $w \leq aq$, wtedy $x = w/p$ i $y = 0$,
- Jeżeli $w > aq$, wtedy $x = aq/p$ i $y = w/q - a$.

Wyobraźmy sobie, że konsument startuje z małym dochodem w i powoli ten dochód się powiększa. Na początku cały majątek przeznaczany jest na dobro x. W końcu, po osiągnięciu pewnego pułapu dochodów, wydatek na dobro x pozostaje na stałym poziomie a cała nadwyżka dochodu ponad ten poziom przeznaczana jest na dobro y. Dobro x jest przykładem dobra koniecznego. Ma bezwzględne pierwszeństwo w wydatkach, jednak po zaspokojeniu popytu na to dobro, cały dodatkowy dochód może być wydany na inne dobra.

Kuhn-Tucker: Ceny w szczycie i poza nim

Dana jest firma telefoniczna. W czasie 24-godzinnego dnia ustala ona ceny obowiązujące w każdej godzinie dnia: $p_1, p_2, ..., p_{24}$. Zapotrzebowanie w każdej godzinie dnia oznaczone jest jako: $x_1, x_2, ..., x_{24}$ i mierzone jest w minutach rozmów. Załóżmy, że $x_i > 0$, tj. w każdej godzinie sprzedaje się dodatnią liczbę minut. Maksymalna łączna liczba minut w jednej godzinie wynosi y. Funkcja $C(x_1, x_2, ..., x_{24})$ opisuje łączny dzienny koszt operacyjny, natomiast $g(y)$ - dzienny koszt kapitału (wydolność systemu). Załóżmy, że krańcowe koszty operacyjne, $\frac{\partial C}{\partial x_i}$, oraz krańcowe koszty kapitału $\frac{dg}{dy}$, są dodatnie. Na rynku panuje doskonała konkurecja, tj. liczba zapotrzebowania nie ma wpływu na ceny $\frac{\partial p_i}{\partial x_i} = 0$. Zatem ceny $p_1, p_2, ..., p_{24}$ są stałe i ustalone.

Firma maksymalizuje łączny zysk dzienny:

$$\Pi = \sum_{i=1}^{24} p_i x_i - C(x_1, x_2, ..., x_{24}) - g(y)$$

przy zadanych warunkach ograniczających:

$$\begin{aligned} & x_i \leq y, \quad \forall i \in \{1, 2, ..., 24\}, \\ & x_i \geq 0, \quad \forall i \in \{1, 2, ..., 24\}, \\ & y \geq 0 \end{aligned} \tag{1.30}$$

Funkcja Lagrange'a przyjmuje następującą postać:

$$L(\mathbf{x}, y, \lambda) = \sum_{i=1}^{24} p_i x_i - C(x_1, x_2, ..., x_{24}) - g(y) + \sum_{i=1}^{24} \lambda_i (y - x_i)$$

Przyjmując założenie doskonałej konkurecji warunki Kuhn-Tuckera są nastepujące:

$$\frac{\partial L}{\partial x_i} = p_i - \frac{\partial C}{\partial x_i} - \lambda_i \leq 0, \quad \forall i \in \{1, 2, ..., 24\}, \tag{1.31}$$

$$x_i \frac{\partial L}{\partial x_i} = x_i \left(p_i - \frac{\partial C}{\partial x_i} - \lambda_i \right) = 0, \quad \forall i \in \{1, 2, ..., 24\}, \tag{1.32}$$

$$\frac{\partial L}{\partial y} = -\frac{dg}{dy} + \sum_{i=1}^{24} \lambda_i \leq 0, \tag{1.33}$$

$$y \frac{\partial L}{\partial y} = y \left(-\frac{dg}{dy} + \sum_{i=1}^{24} \lambda_i \right) = 0, \tag{1.34}$$

$$\frac{\partial L}{\partial \lambda_i} = y - x_i \geq 0, \quad \forall i \in \{1, 2, ..., 24\}, \tag{1.35}$$

$$\lambda_i \frac{\partial L}{\partial \lambda_i} = \lambda_i (y - x_i) = 0, \quad \forall i \in \{1, 2, ..., 24\}, \tag{1.36}$$

$$\lambda_i \geq 0, \quad i \in \{1, 2, ..., 24\}. \tag{1.37}$$

Ponieważ założyliśmy, że $x_i > 0$, dla każdego $i \in \{1, 2, ..., 24\}$, zatem warunek (1.35) implikuje, że $y > 0$. Ponieważ interesują nas rozwiązania, gdzie wszystkie x_i oraz y są dodatnie, warunki (1.31) i (1.33) przyjmują postać:

$$p_i - \frac{\partial C}{\partial x_i} - \lambda_i = 0, \quad \forall i \in \{1, 2, ..., 24\}, \tag{1.38}$$

$$-\frac{dg}{dy} + \sum_{i=1}^{24} \lambda_i = 0, \tag{1.39}$$

W każdej godzinie poza szczytem t nie jest wykorzystana cała wydolność: $y > x_t$. A zatem warunek (1.36) implikuje, że $\lambda_t = 0$ dla godzin poza szczytem. Z warunku (1.38) otrzymujemy:

$$p_t = \frac{\partial C}{\partial x_t}, \quad \text{dla kadej godziny poza szczytem } t$$

Ponieważ jest nadwyżka mocy dla godzin poza szczytem, popyt powinien być stymulowany poprzez nałożenie najniższej możliwej ceny bez ponoszenia strat dla krańcowej sprzedanej jednostki.

Dla każdej godziny w szczycie s, zdolności są w pełni wykorzystane: $x_s = y$. Ponieważ założylismy, że $\frac{dg}{dy} > 0$ (zwiększenie wydolności wymaga poniesienia kosztów kapitałowych), stąd wynika, że:

$$\frac{dg}{dy} = \sum_s \lambda_s > 0,$$

A więc przynjamniej dla niektórych godzin w szczycie, mnożniki Lagrange'a muszą być dodatnie. Wówczas otrzymujemy na mocy warunku (1.38):

$$p_s = \frac{\partial C}{\partial x_s} + \lambda_s, \quad \text{dla kadej godziny w szczycie } s$$

Cena przewyższa krańcowy koszt operacyjny o dodatkową wartość równą wartości mnożnika Lagrange'a λ_s. Dodatkowo na mocy warunku (1.39), suma tych dodatkowych narzutów dla wszystkich godzin w szczycie równa się łącznie krańcowemu kosztowi kapitału $\frac{dg}{dy}$. Ponieważ popyt w godzinach szczytu "wywiera presjęńa wydolność systemu, każdy wzrost popytu wymaga dodatkowego kapitału, którego krańcowy koszt musi zatem zostać pokryty.

1.4. Część dodatkowa: Słaba dualność

W niniejszej części omawiana jest słaba dualność. Dla przejrzystości argumentacji zdecydowano się na prezentację problemu z tylko jednym ograniczeniem w postaci nierówności oraz jednym ograniczeniem w postaci równania. Zaprezentowany argument łatwo jest uogólnić na przypadek z wieloma ograniczeniami obydwu typów.

Rozważmy następujący problem:

$$\max f(x), \quad \text{p.w.} \quad g(x) \leq c, \quad h(x) = d. \tag{1.40}$$

Zbiór wszystkich punktów spełniających ograniczenia problemu oznaczmy jako D. Zdefiniujmy Lagrangian jako

$$L(x, \lambda, \mu) = f(x) - \lambda[g(x) - c] - \mu[h(x) - d]$$

Zauważmy, że zachodzi następująca zależność:

$$\forall x \in D, \ \forall \lambda \geq 0, \ \forall \mu \in \mathbf{R}: \quad f(x) \leq L(x, \lambda, \mu) \tag{1.41}$$

Zauważmy również, że problem (1.40) jest równoważny następującemu probelmowi:

$$\max_x \min_{\lambda \geq 0, \ \mu} L(x, \lambda, \mu) = p^*$$

Ponieważ:

$$\min_{\lambda \geq 0, \ \mu} -\lambda[g(x) - c] - \mu[h(x) - d] = \begin{cases} 0, & \text{jeli } g(x) \leq c, \text{ oraz } h(x) = d, \\ -\infty, & \text{w pozostaych przypadkach.} \end{cases}$$

Zdefiniujmy $\phi(\lambda, \mu) = \max x L(x, \lambda, \mu)$. Funkcja ta jest wypukła, ponieważ jest maksymalizacją punkt po punkcie funkcji afinicznych ($L(x, \cdot, \cdot)$ jest funkcją afiniczną dla każdego punktu x). Jeśli zmaksymalizujemy prawą stronę warunku (1.41), otrzymamy:

$$\forall x \in D, \ \forall \lambda \geq 0, \ \forall \mu \in \mathbf{R}: \quad f(x) \leq \max_{x'} L(x', \lambda, \mu) = \phi(\lambda, \mu)$$

Jeśli teraz zmaksymalizujemy lewą stronę tej nierówności względem zmiennych należących do D, to otrzymamy:

$$\forall \lambda \geq 0,\ \forall \mu \in \mathbf{R}: \quad p^* = \max_{x \in D} f(x) \leq \phi(\lambda, \mu)$$

Ostatecznie możemy zminimalizaować prawą stronę tego warunku, skąd otrzymujemy górne ograniczenie wartości maksymalizowanej funkcji f:

$$p^* \leq d^* = \min_{\lambda \geq 0,\ \mu} \phi(\lambda, \mu)$$

Problem, którego rozwiązaniem jest wartość d^* jest nazywany problemem dualnym. Ograniczeniem tego problemu jest nie podane *explicité* ograniczenie, że $(\lambda, \mu) \in \mathbf{dom}\phi$. Problem dualny polega na minimalizacji wypukłej funkcji, przy wklęsłych ograniczeniach. Rezultat, który właśnie udowodniliśmy nosi nazwę słabej dualności[16].

Rozważmy przykład minimalizacji odległości Euklidesowej przy zadanym ograniczeniu afinicznym:

$$\begin{aligned} &\min x^T x \\ &\text{p.w.} \quad Ax = b \end{aligned}$$

Lagrangian przyjmuje następującą postać:

$$L(x, \mu) = x^T x - \mu^T (Ax - b)$$

Aby zminimalizować L względem zmiennej x, przyrównajmy gradient do zera:

$$\nabla_x L(x, \mu) = 2x - A^T \mu = 0 \quad \Rightarrow \quad x = (1/2) A^T \mu$$

Wstawmy rozwiązanie do funkcji Lagrange'a, aby otrzymać funkcję ϕ:

$$\phi(\mu) = L(1/2 A^T \mu, \mu) = -\frac{1}{4} \mu^T A A^T \mu + b^T \mu$$

Dziedziną tej funkcji jest cała przestrzeń $\mathbf{R}^n$. Problem dualny jest zatem postaci:

$$d^* = \max_{\mu} \phi(\mu) = \max_{\mu} -\frac{1}{4} \mu^T A A^T \mu + b^T \mu$$

Problem ten można obliczyć analitycznie, ponieważ jest to problem maksymalizacji bez ograniczeń. Maksymalizowana funkcja jest funkcją wklęsłą, zatem warunki pierwszego rzędu są warunkami wystarczającymi optymalności:

$$-\frac{1}{2} A A^T \mu + b = 0 \quad \Rightarrow \quad \mu^* = \frac{1}{2} (A A^T)^{-1} b$$

Otrzymaliśmy zatem ograniczenie dolne wartości optymalnej:

$$p^* \geq d^* = \frac{1}{4} b^T (A A^T)^{-1} b$$

[16] Niniejszy argument został przedstawiony dla maksymalizacji funkcji przy ograniczeniu postaci "$\leq$". Podobny rezultat zachodzi dla problemu minimalizacji funkcji przy ograniczeniu "$\geq$". Wówczas otrzymujemy problem dualny w postaci maksymalizacji i jego rozwiązanie jest ograniczeniem dolnym wartości minimalizowanej funkcji.

Zadania

1) Czy poniższa korespondencja jest ciągła? Odpowiedź uzasadnij formalnie.

$$D(x) = \begin{cases} \{1, 1/x\} & \text{jeeli } x > 0 \\ \{0\} & \text{jeeli } x = 0 \end{cases}$$

2) Czy poniższa korespondencja budżetu Walrasa jest ciągła? Odpowiedź uzasadnij formalnie.

$$B(\mathbf{p}, w) = \{\mathbf{x} \in \mathbf{R}^n | \mathbf{x} \geq 0, \quad \mathbf{p}^T\mathbf{x} \leq w\},$$

dla $\mathbf{p} \in \mathbf{R}^n$, $\mathbf{p} >> 0$ oraz $w \in \mathbf{R}$.

3) Rozważmy następującą funkcję $f : [0, 1] \to \mathbf{R}$:

$$f(x) = \begin{cases} x & \text{jeeli } 0 \leq x < 1 \\ 0 & \text{jeeli } x = 1 \end{cases}$$

Czy problem $\max_{x \in [0,1]} f(x)$ ma rozwiązanie? Dlaczego nie stosuje się tutaj Twierdzenie maksimum?

4) Znajdź ekstrema następującej funkcji $f(x, y) = x^3 - 8y^3 + 6xy + 1$. Który z punktów stacjonarnych jest lokalnym minimum, lokalnym maksimum, a który jest punktem siodłowym? [Wskazówka: punkt stacjonarny jest punktem siodłowym, jeśli nie jest ani lokalnym minimum, ani lokalnym maksimum]

5) Rozwiąż następujący problem optymalizacyjny z pomocą twierdzenia Kuhn-Tuckera:

$$\min_{\mathbf{x} \in \mathbf{R}^2_+} w_1 x_1 + w_2 x_2$$
$$\text{p.w. } y \geq (x_1^\rho + x_2^\rho)^{1/\rho}$$

zakładając, że $y, w_1, w_2, \rho > 0$.

6) Rozwiąż następujący problem optymalizacyjny

$$\max_{\mathbf{R}^n_+} \prod_{i=1}^{n} x_i^{\alpha_i}$$
$$\text{p.w. } \mathbf{p}^T\mathbf{x} = w$$

gdzie $\alpha_i, p_i, w > 0$. Wskaż rozwiązanie oraz funkcję wartości dla tego problemu dla dowolnych parametrów $(\mathbf{p}, w)$ przy pomocy twierdzenia Lagrange'a.

2. Numeryczne aspekty optymalizacji

TODO: Dodać przykłady od Grześka

W niniejszym rozdziale omawiane są zagadnienia praktyczne związane z przybliżoną reprezentacją liczb rzeczywistych w implementacji komputerowej. W podrozdziale omawiającym podstawy teoretyczne tego zagadnienia przedstawiamy sposób reprezentacji tych liczb oraz jego konsekwencje dla dokładności wykonywanych obliczeń. Z kolei w podrozdziale przedstawiającym przykłady zastosowań prezentujemy typowe sytuacje, w których występowanie błędów zaokrągleń może prowadzić do niedokładności lub błędów we wnioskowaniu w zagadnieniach programistycznych i optymalizacyjnych wykorzystujących obliczenia przybliżone.

Podstawy teoretyczne

W zagadnieniach optymalizacyjnych omawianych w książce zajmujemy się optymalizacją funkcji, których dziedziną jest zbiór $\mathbb{R}^n$, a przyjmują one wartości w zbiorze $\mathbb{R}$. W związku z tym w niniejszym rozdziale omówiony jest sposób reprezentacji komputerowej liczb rzeczywistych oraz jego konsekwencje dla rozwiązywania zadań optymalizacyjnych.

Liczb rzeczywistych jest nieprzeliczalnie wiele. Jednak w reprezentacji komputerowej do ich przechowywania wykorzystywana jest skończona pamięć. W R do ich przechowywania wykorzystywany jest typ zmiennoprzecinkowy `double` (tzw. *storage mode*) wykorzystujący format *binary64* zgodny ze standardem IEEE 754. W formacie tym liczba jest reprezentowana za pomocą 64 bitów. W związku z tym liczba różnych liczb rzeczywistych możliwych do dokładnego przedstawienia jest skończona i nie przekracza 2^{64}. Oznacza to, że w ogólności w reprezentacji komputerowej liczby rzeczywiste będą reprezentowane jedynie w przybliżeniu.

Format *binary64* reprezentuje liczbę rzeczywistą, w uproszczeniu, w postaci zmiennoprzecinkowej danej wzorem:

$$c(-1)^s 2^q, \tag{2.1}$$

gdzie $s \in \{0, 1\}$, c jest liczbą z przedziału $[1, 2)$ posiadającą 52 cyfry po przecinku w zapisie dwójkowym[17], a q jest liczbą całkowitą większą niż -1023 i mniejszą niż

[17] Odpowiada to niecałym 16 cyfrom po przecinku w zapisie dziesiętnym.

1024. Dodatkowo występują specjalne konwencje reprezentacji 0, nieskończoności (`Inf`, `-Inf` w R) oraz oznaczenia wartości nie będącej liczbą (`NaN`) i braku danych (`NA`). R udostępnia funkcje umożliwiające wykrywanie tych specjalnych sytuacji[18]. Zwracane przez nie wyniki w zależności od argumentu zaprezentowane są w kodzie 2.1. Dalej przez $\mathcal{F} \in \mathbb{R}$ będziemy oznaczali podzbiór zbioru liczb rzeczywistych możliwy do reprezentacji dokładnej za pomocą formatu *binary64*.

Kod 2.1. Dostępne w R funkcje umożliwiające testowanie niestandardowych wartości zmiennych

```
x <- c(0, Inf, -Inf, NA, NaN)
is.na(x)       # FALSE FALSE FALSE TRUE  TRUE
is.nan(x)      # FALSE FALSE FALSE FALSE TRUE
is.finite(x)   # TRUE  FALSE FALSE FALSE FALSE
is.infinite(x) # FALSE TRUE  TRUE  FALSE FALSE
```

Zanim przejdziemy dalej waro zwrócić uwagę na fakt, że w R możliwe jest też przechowywanie liczb w formacie całkowitoliczbowym. Ten typ zmiennej, tzw. `integer`, pozwala w obecnej implementacji R na przechowywanie liczb całkowitych o wartościach od $-2^{31}+1$ do $2^{31}-1$. Jednak w praktyce ten typ zmiennej nie jest często wykorzystywany[19]. Przyczyną takiej sytuacji jest fakt, że zmienne w formacie *binary64* pozwalają na dokładne przechowywanie liczb o znacznie większej wartości. Wszystkie liczby całkowite z przedziału od -2^{53} do 2^{53} mogą być zapisane w tym formacie bez utraty precyzji.

Przez $fl\colon \mathbb{R} \to \mathcal{F}$ będziemy oznaczali funkcję, która przypisuje liczbie rzeczywistej jej reprezentację zmiennoprzecinkową. Niektóre liczby rzeczywiste posiadają dokładną reprezentację w zbiorze $\mathcal{F}$, ale większość z nich reprezentowana jest w sposób przybliżony. Na przykład liczby 100 czy 0,25 mają skończoną reprezentację w układzie dwójkowym, która ma mniej niż 53 cyfry oraz są dostatecznie małe co do wartości bezwzględnej. W związku z tym są reprezentowane dokładnie. Innymi słowy mamy $fl(100) = 100$ i $fl(0{,}25) = 0{,}25$. Z drugiej strony iczba 0,3 w zapisie dwójkowym ma reprezentację nieskończoną $0,01(0011)$, która zostanie ucięta w reprezentacji zmiennoprzecinkowej, a więc $fl(0{,}3) \neq 0{,}3$. Z kolei liczba 10^{400} jest zbyt duża, aby mogła być reprezentowana w formacie *binary64*. Własności te można zweryfikować wykonując kod 2.2, gdzie stwierdzamy, że liczba 0,3 została zaokrąglona w dół, a liczba 10^{400} jest oznaczona jako plus nieskończoność.

Kod 2.2. Weryfikacja odwzorowania liczb rzeczywistych do zbioru $\mathcal{F}$

```
format(100, digits = 22)     # 100
format(0.25, digits = 22)    # 0.25
format(0.3, digits = 22)     # 0.2999999999999999888978
format(10^400, digits = 22)  # Inf
```

[18] Wbudowanie w jądro R pełnej obsługi tych przypadków jest jedną z jego istotnych zalet. Wiele alternatywnych środowisk analitycznych nie pozwala np. na rozróżnienie wartości brakującej (`NA`) od błędnego wyniku obliczeń (`NaN`).

[19] Jest on przewidziany głównie w celu umożliwienia komunikacji z programami napisanymi w językach C i Fortran.

Zwróćmy uwagę, że podobnie jak w zbiorze $\mathbb{R}$ zdefiniowane są działania dodawania, mnożenia itd., podobnie można je zdefiniować w zbiorze $\mathcal{F}$. Działania zdefiniowane w zbiorze $\mathcal{F}$ są wykonywane przez R podczas dokonywania obliczeń. Dalej w niniejszym rozdziale będziemy stosowali konwencję, w której rozważania dotyczą zbioru $\mathbb{R}$ będziemy wykorzystywali krój matematyczny, np. $1{,}5 - 0{,}7 = 0{,}8$, natomiast w przypadku rozważań dotyczących zbioru $\mathcal{F}$ krój maszyny do pisania, np. `1.5-0.7=0.8`.

Skończona reprezentacja liczb rzeczywistych oznacza, że można za ich pomocą przedstawiać tylko liczby ze skończonego przedziału, który w przybliżeniu wynosi $(-2 \cdot 10^{308}, 2 \cdot 10^{308})$. Ze względu na to, że w zapisie binarnym liczba zgodna z formatem *binary64* posiada 52 cyfry po przecinku to $fl(x) = (1+e)x$, gdzie $|e| \leq 2^{-53}$, jeżeli x należy do powyższego przedziału. Oznacza, to, że liczby w reprezentacji zmiennoprzecinkowej mają gwarantowaną względną precyzję reprezentacji, tj. dla $x \neq 0$:

$$\left| \frac{x - fl(x)}{x} \right| \leq 2^{-53}. \tag{2.2}$$

Z tej zależności wynika również, że odstępy pomiędzy kolejnymi dostępnymi liczbami nie są równe. Liczby bardzo małe co do wartości bezwzględnej są rozłożone w zbiorze $\mathcal{F}$ znacznie gęściej niż liczby bardzo duże. Na przykład najmniejszą liczbą `e` taką że `1≠1+e` jest w przybliżeniu $fl(2{,}220446 \cdot 10^{-16})$. Jej wartość w R zapisana jest w zmiennej `.Machine$double.eps` i wynosi 2^{-52}. Z kolei `2+e` jest w reprezentacji zmiennopozycyjnej jest równe `2`. Z drugiej strony zachodzi `0.5≠0.5+e/2`. Zachodzenie tych zależności można zweryfikować za pomocą kodu 2.3.

Kod 2.3. Porównanie bezwzględnej precyzji reprezentacji liczb zmiennoprzecinkowych w zależności od ich skali

```
test <- function(x, y) {
    x == x + y
}

test(1, .Machine$double.eps)       # FALSE
test(2, .Machine$double.eps)       # TRUE
test(1, .Machine$double.eps / 2)   # TRUE
test(0.5, .Machine$double.eps / 2) # FALSE
```

Podobnie czytelnik może sprawdzić, że wyrażenie R postaci `10 ^ k + 1 == 10 ^ k` przyjmuje wartość `TRUE` dla naturalnych `k` większych od 15, a wartość `FALSE` dla mniejszych.

Dokładna specyfikacja liczb zmiennopozycyjnych wykorzystywanych w R przechowywana jest w zmiennej `.Machine`. W tym miejscu zwróćmy jeszcze uwagę na normalizację liczb w reprezentacji zmiennoprzecinkowej. Zmienna `.Machine$double.xmin` zawiera najmniejszą dodatnią znormalizowaną liczbę zmiennoprzecinkową i przyjmuje wartość około $2{,}225074 \cdot 10^{-308}$. Poprzez liczbę znormalizowaną rozumiemy taką, która jest zgodna ze specyfikacją zadaną równaniem (2.1). Jest ona równa dokładnie $1 \cdot (-1)^0 \cdot 2^{-1022}$. Zauważmy jednak, że akceptując utratę precyzji możemy przyjąć,

że w równaniu (2.1) stała c nie należy do przedziału $[1, 2)$, ale jest mniejsza niż 1. Ponieważ dysponujemy 52 bitami po przecinku to najmniejszą dodatnią zdenormalizowaną liczbą możliwą do przedstawienia w formacie *binary64* jest $2^{-1022}/2^{52}$, czyli 2^{-1074}, która ma wartość około $4{,}940656 \cdot 10^{-324}$.

Przedstawiona w kodzie 2.3 własność reprezentacji zmiennoprzecinkowej jest przykładem powstawania błędów zaokrągleń. W ogólności występowanie takiego błędu oznacza, że wynik wykonania działania na reprezentacji liczb zmiennoprzecinkowych nie musi być dokładnie równy reprezentacji wyniku tego działania wykonanego w liczbach rzeczywistych. Na przykład w R wartość wyrażenia `0.2 + 0.4` nie jest równa `0.6`. Można to sprawdzić wykonując polecenie `0.2 + 0.4 - 0.6`, które daje wynik w przybliżeniu równy $1{,}110223 \cdot 10^{-16}$, a nie 0. Przyczyna takiej zależności jest wyjaśniona w kodzie 2.4. Zauważamy, że liczby 0,2 i 0,4 w reprezentacji zmiennoprzecinkowej są zaokrąglane w górę, a liczba 0,6 jest zaokrąglana w dół. To jest źródło zaobserwowanej niezgodności.

Kod 2.4. Reprezentacja zmiennoprzecinkowa liczb 0,2, 0,4 i 0,6

```
format(0.2, digits = 22)       # 0.20000000000000000111022
format(0.4, digits = 22)       # 0.40000000000000000222045
format(0.2 + 0.4, digits = 22) # 0.60000000000000000888178
format(0.6, digits = 22)       # 0.59999999999999999777955
```

Rozważmy następujący problem ukazujący powszechność występowania błędów zaokrągleń w obliczeniach. Chcemy od liczby mającej zapis dziesiętny $u.d$ odjąć najpierw część całkowitą u a następnie mnożymy przez 10 i odejmujemy część dziesiętną d, gdzie $u, d \in \{1, \ldots, 9\}$. Okazuje się, że w 67 przypadkach na 81 wynik tego działania nie daje dokładnie 0. Jest to przedstawione w kodzie 2.5.

Kod 2.5. Wynik działania $10(u.d - u) - d$, gdzie $u, d \in \{1, \ldots, 9\}$

```
data.set <- expand.grid(u = 1:9, d = 1:9)
data.set$u.d <- as.numeric(with(data.set, paste(u, d, sep = ".")))
data.set$result <- with(data.set, 10 * (u.d - u) - d)
table(data.set$result ==  0) # FALSE: 67, TRUE: 14
```

W związku z występowaniem błędów zaokrągleń nie są w ogólności spełnione dla liczb w reprezentacji zmiennoprzecinkowej własności działań matematycznych, które są prawdziwe dla liczb rzeczywistych. Na przykład nie jest spełnione prawo przemienności dodawania, co jest zobrazowane w kodzie 2.6.

Kod 2.6. Weryfikacja braku przemienności dodawania w arytmetyce liczb zmiennoprzecinkowych

```
0.3 + 0.3 + 0.4 - 1   # 0
-1 + 0.3 + 0.3 + 0.4  # 5.551115e-17
```

Należy jednak zauważyć, że liczby całkowite nieprzekraczające 2^{53} co do wartości bezwzględnej mają dokładną reprezentację zmiennoprzecinkową. Mówiąc precyzyjnie

jeżeli x jest liczbą całkowitą taką, że $|x| \leq 2^{53}$ to $fl(x) = x$. Oznacza to, że jeżeli wyniki działań (w tym pośrednie) na reprezentacji zmiennoprzecinkowej liczb całkowitych są całkowite i nie przekraczają w wartości bezwzględnej tej wartości będą spełnione wszystkie własności teoretyczne działań matematycznych. Znaczenie konieczności zapewnienia, że wyniki pośrednie działań nie przekraczają co do wartości bezwzględniej 2^{53} prezentuje kod 2.7.

Kod 2.7. Badanie dokładności obliczeń wykonywanych na liczbach całkowitych w reprezentacji zmiennoprzecinkowej

```
2^52 - 2^52 - 1    # -1
2^52 - (2^52 + 1)  # -1
2^53 - 2^53 - 1    # -1
2^53 - (2^53 + 1)  # 0
```

Poprawną wartością wyrażenia jest -1. Jednak w ostatnim przypadku liczba $2^{53} + 1$ przekracza próg dokładnej reprezentacji i otrzymujemy błędny wynik wynikający z zaokrąglenia.

Powyższy przykład jest przypadkiem szczególnym problemu utraty względnej precyzji liczb zmiennoprzecinkowych w wyniku wykonywania na nich działań arytmetycznych. W celu jego zobrazowania rozważmy hipotetycznie najbardziej pesymistyczny przypadek poddający się jednocześnie prostej analizie. Weźmy dwie liczby dodatnie $x = 1 + y$ takie, że $fl(x) = x(1 + e_1)$, $fl(y) = y(1 + e_2)$ oraz $e_1 = -e_2 = 2^{-53}$. Oszacujmy względny błąd zaokrąglenia wyniku odejmowania tych liczb w arytmetyce zmiennoprzecinkowej:

$$\frac{(x - y) - fl(fl(x) - fl(y))}{x - y} \approx \frac{(x - y) - (x(1 - e_1) - y(1 - e_2))}{x - y} = \frac{xe_1 - ye_2}{x - y}. \quad (2.3)$$

Podstawiając założone wartości do tego wyrażenia otrzymujemy:

$$\frac{(1 + y)2^{-53} + y2^{-53}}{y + 1 - y} = (1 + 2y)2^{-53}. \quad (2.4)$$

Zauważmy, że jeżeli y będzie znacznie większe od 0 otrzymamy względny błąd wyniku obliczeń znacznie większy niż gwarantowana precyzja reprezentacji argumentów działania równa 2^{-53}. Otrzymany wynik pokazuje jedynie, że możliwa jest utrata względnej precyzji. Kod 2.8 prezentuje przykład, w którym sytuacja ta występuje.

Kod 2.8. Przykład utraty względnej precyzji w obliczeniach wykonywanych w reprezentacji zmiennoprzecinkowej

```
x <- 10 ^ 15 + 0.44
format(x, digits = 22) # 1000000000000000.500000
y <- 10 ^ 15 - 0.44
format(y, digits = 22) # 999999999999999.500000
x - y                  # 1
```

Najpierw zauważmy, że `x` oraz `y` są reprezentowane ze względnym błędem równym około $0{,}06/10^{15}$. Jest to precyzja większa niż 2^{-53}. Niestety w wyniku ich odejmowania mamy do czynienia ze znaczną utratą precyzji, ponieważ różnica `x-y` wynosi 1, a powinna wynieść 0,88, więc względny błąd wyniku obliczeń wynosi aż 3/22.

W niniejszym podrozdziale omówiliśmy podstawowe własności reprezentacji komputerowej liczb zmiennopozycyjnych oraz jej konsekwencje dla wykonania obliczeń na tego typu liczbach. W kolejnym podrozdziale omówimy przykłady potencjalnie pojawiające się w zagadnieniach optymalizacyjnych, w których mogą pojawić się problemy z precyzją obliczeń przy wykorzystaniu reprezentacji zmiennoprzecinkowej.

Przykłady zastosowań

W niniejszym podrozdziale prezentujemy dziewięć przykładów, w których zagadnienie nieprecyzyjnej reprezentacji liczb rzeczywistych za pomocą formatu zmiennoprzecinkowego jest istotne. Rozpoczynamy od przedstawienia problemów związanych z wykorzystaniem liczb zmiennoprzecinkowych przy weryfikacji warunków logicznych i indeksowaniu pętli. Następnie przedstawiamy wpływ błędów zaokrągleń na sekwencje liczb zmiennoprzecinkowych i jego potencjalne konsekwencje w połączeniu z błędami zaokrągleń w algorytmach optymalizacyjnych wykorzystujących listę tabu. W kolejnym przykładzie przechodzimy do omówienia zagadnienia kumulowania się błędów zaokrągleń w przypadku wykonywania wielokrotnych obliczeń zmiennoprzecinkowych na przykładzie algorytmu sumowania liczb. Następnie wskazujemy znaczenie problemu utraty precyzji obliczeń dla określania wartości funkcji celu przy optymalizacji. W konsekwencji przechodzimy do omówienia zagadnień numerycznego wyznaczania pochodnej funkcji jednej zmiennej oraz hesjanu funkcji wielu zmiennych. Następnie badamy wpływ zmiany punktu w którym jest obliczana wartość funkcji na otrzymywany wynik. Prowadzi to do zagadnienia numerycznej weryfikacji warunku zatrzymania optymalizacji. Kolejny przykład pokazuje możliwość zwiększenia precyzji obliczeń w R przy pomocy pakietu `Rmpfr`. Ostatni przykład porusza typowe problemy związane z błędami zaokrągleń pojawiające się w przypadku wyznaczania estymatorów za pomocą metody największej wiarygodności.

Przykład 2.1. Rozważmy znaczenie problemu dokładności obliczeń na zmiennych w formacie *binary64* na wykonywanie pętli. Jest to częsty problem pojawiający się w sytuacjach praktycznych i związany jest z weryfikacją warunku zakończenia obliczeń w pętli. Kod 2.9 prezentuje ten problem w przypadku równościowego warunku zakończenia pętli. Zaprezentowane są w nim dwa przypadki. Pętla pierwsza indeksowana liczbami całkowitymi. Mimo, że są one reprezentowane za pomocą liczb zmiennopozycyjnych obliczenia wykonywane są z pełną dokładnością i pętla przerywa działanie w momencie gdy `x` osiąga wartość dokładnie równą 0. Druga pętla jest odpowiednikiem pierwszej, ale zmienna `x` jest w niej podzielona przez 10. Ponieważ nie operuje ona na liczbach całkowitych, ale ułamkowych w 11 iteracji zmienna `x` przyjmuje wartość bliską zeru (ale nie dokładnie równą zero) i pętla nie kończy działania, ale je kontynuuje wykonując obliczenia na liczbach ujemnych.

Kod 2.9. Porównanie wpływu niedokładności reprezentacji liczb zmiennopozycyjnych na weryfikację równościowego warunku zakończenia pętli

```
# obliczenia dokładne
x <- 10
while (x != 0) {
    print(x)
    x <- x - 1
}

# obliczenia przybliżone, pętla przekracza 0
x <- 1
while (x != 0) {
    print(x)
    x <- x - 0.1
}
```

Częściowym rozwiązaniem tego problemu jest zastąpienie warunku równościowego nierównością. Przykład takiego działania podany jest w kodzie 2.10. Podane są w nim dwie pętle, które są identyczne z dokładnością do przeskalowania wartości zmiennej `x` o 3 razy. W związku z tym można by oczekiwać, że powinny dawać dokładnie te same wartości w wyniku swojego działania. Niestety taka sytuacja nie zachodzi. Pętla pierwsza nie drukuje wartości `0.3` mimo, że należałoby oczekiwać, że powinna. Pętla druga wartość tą drukuje.

Kod 2.10. Porównanie wpływu niedokładności reprezentacji liczb zmiennopozycyjnych na weryfikację nierównościowego warunku zakończenia pętli

```
# Wynik: 0  0.1  0.2
x <- 0
while (x <= 0.3) {
    cat(x, "␣")
    x <- x + 0.1
}

# Wynik: 0  0.1  0.2  0.3
x <- 0
while (x / 3 <= 0.3) {
    cat(x / 3, "␣")
    x <- x + 0.3
}
```

Przyczynę tej sytuacji wyjaśnić można badając reprezentację zmiennoprzecinkową liczb 0,1 i 0,3; por. kod 2.11. Widzimy, że liczba 0,1 jest zaokrąglana w górę, a liczba 0,3 w dół. W konsekwencji po trzecim wykonaniu pętli pierwszej wartość zmiennej `x` jest nieco większa niż `0.3`. Z kolei w pętli drugiej dodawana jest liczba `0.3`, która

jest zaokrągleniem w dół liczby 0,3 i w trzecim kroku pętli warunek `x / 3 <= 0.3` jest spełniony.

Kod 2.11. Reprezentacja zmiennoprzecinkowa liczb 0,1 i 0,3

```
format(0.1, digits = 22) # 0.10000000000000000055511
format(0.3, digits = 22) # 0.29999999999999999888978
```

Metodą, która pozwala na rozwiązanie problemu pojawiającego się w kodzie 2.10 jest dołożenie korekty do w warunku zakończenia pętli gwarantującej, że błędy zaokrągleń nie spowodują nieoczekiwanego działania pętli. Kod 2.12 prezentuje wprowadzenie takiej korekty o wartości `0.0001`. Liczba ta jest na tyle duża, że jest większa niż błąd zaokrągleń, a jednocześnie na tyle mała, że nie powoduje wykonania kolejnego kroku pętli. Wadą tego podejścia jest konieczność każdorazowego ustalenia odpowiedniej wielkości korekty, co nie zawsze musi być proste.

Kod 2.12. Przykład nierównościowego warunku zakończenia pętli z korektą

```
# 0  0.1  0.2  0.3
x <- 0
while (x <= 0.3 + 0.0001) {
    cat(x, "␣")
    x <- x + 0.1
}
```

Podsumowując powyższy przykład należy stwierdzić, że podstawową rekomendacją w przypadku projektowania pętli jest unikanie konstrukcji programistycznych, w których pojawiają się błędy zaokrągleń. Jeśli nie jest to możliwe to kluczowe jest takie ich konstruowanie, które bierze pod uwagę występowanie zaokrągleń w obliczeniach. □

Przykład 2.2. Typowym zagadnieniem pojawiającym się przy wykonywaniu obliczeń numerycznych jest generowanie sekwencji liczb, a w szczególności sekwencji o równych odstępach. Rozważmy więc zagadnienie wygenerowania 11 liczb rozłożonych równomiernie w przedziale $[0, 1]$. Kod 2.13 prezentuje dwa sposoby wygenerowania takiej sekwencji. Okazuje się, że nawet w tak prostym zadaniu 3 z 11 liczb nie są równe. Dodatkowo zarówno dla sekwencji `x` jak i dla sekwencji `y` odstępy między kolejnymi liczbami nie są równe ale przyjmują 4 różne wartości.

Kod 2.13. Przykład nierównościowego warunku zakończenia pętli z korektą

```
x <- (0:10) / 10
y <- seq(0, 1, len = 11)
table(x == y)                 # FALSE: 3, TRUE: 8
length(unique(diff(x) - 0.1)) # 4
max(abs(diff(x) - 0.1))       # 8.326673e-17
length(unique(diff(y) - 0.1)) # 4
max(abs(diff(y) - 0.1))       # 8.326673e-17
```

Różnice w ostępach w przypadku obu sekwencji `x` i `y` są mniejsze niż 10^{-16}, a więc nie są duże. Jednak, jeśli kod wykorzystujący te sekwencje zakładałby, że mają one *dokładnie* równe odstępy to działałby błędnie.

Problem ten w zagadnieniach optymalizacyjnych pojawia się np. algorytmie przeszukiwania z listą tabu. Idea listy tabu polega na tym aby nie wykonywać oszacowania wartości funkcji dwukrotnie w tym samym punkcie dziedziny funkcji. Zagadnienie to jest szczególnie istotne, gdy wyznaczanie wartości funkcji celu jest bardzo czasochłonne. Kod 2.14 prezentuje przykład prostego randomizowanego algorytmu z listą tabu, który ma na celu poszukiwanie minimum funkcji $f(x) = x^2$ z dokładnością 0,1.

Kod 2.14. Przykład prostego algorytmu z listą tabu

```
interval <- seq(-0.3, 0.3, len = 7)

f <- function(x) {
    tabu <<- c(tabu, x)
    x ^ 2
}

tabu.opt <- function(x0, n) {
    tabu <<- NULL
    f.min <- f(x0)

    for (i in 1:n) {
        x.new <- x0 + sample(interval, 1)
        if (!(x.new %in% tabu)) {
            f.new <- f(x.new)
            if (f.new < f.min) {
                f.min <- f.new
                x0 <- x.new
            }
        }
    }
    x0
}

set.seed(1)
tabu.opt(1, 20) # Znaleziona wartość optymalna: 0
tabu # [1] 1.0 0.8 0.7 0.5 0.6 0.2 0.0 -0.2 0.1 -0.1 0.2 0.3
format(tabu[c(6, 11)], digits = 22)
# 0.20000000000000000111022 0.19999999999999999555911
```

Procedura startuje z punktu 1 i przez 20 iteracji próbuje przesunąć aktualne rozwiązanie do nowego o niższej wartości funkcji celu.

W każdej iteracji losowane jest nowe rozwiązanie próbne jako odchylenie od aktualnie znalezionego rozwiązania optymalnego. Odchylenie to jest liczbą wylosowaną ze

zbioru $\{-0{,}3, -0{,}2, -0{,}1, 0, 0{,}1, 0{,}2, 0{,}3\}$. Algorytm stara się unikać dwukrotnego odwiedzania tego samego punktu za pomocą warunku `!(x.new %in% tabu)`. Z kolei funkcja `f` każdy nowo odwiedzony punkt dodaje do listy tabu przechowywanej w zmiennej `tabu`. Niestety okazuje się, że przyjęta w kodzie strategia nie zapewnia osiągnięcia pożądanego efektu i punkt 0,2 odwiedzany jest dwukrotnie przy 6 i 11 oszacowaniu funkcji `f` co możemy stwierdzić badając zawartość zmiennej `tabu`. Przy tych dwóch oszacowaniach wartość zmiennej `x0` jest różna ze względu na występowanie błędów zaokrągleń, co stwierdzamy wykonując polecenie `format(tabu[c(6, 11)], digits = 22)`.

W przypadku niniejszej procedury metodą zapewniającą uniknięcie tego problemu jest zaokrąglenie zmiennej `x.new` do jednego miejsca po przecinku za pomocą komendy `x.new <- round(x0 + sample(interval, 1), 1)`. Funkcja `round` zapewnia, że zaokrąglenie ma zawsze tą samą reprezentację. □

Przykład 2.3. Problemy zaokrągleń w obliczeniach pojawiają się najczęściej w przypadku wielokrotnej aktualizacji przetwarzanych wartości. Aby zobrazować to zjawisko przedstawimy zagadnienie sumowania liczb. Kod 2.15 prezentuje problem sumowania 10^7 liczb o wartości $1/3$.

Kod 2.15. Sumowanie liczb metodą bezpośrednią oraz zgodnie z algorytmem Kahana

```
simple.sum <- function(x) {
    sum <- 0.0
    for (i in seq_along(x)) {
        sum <- sum + x[i]
    }
    return(sum)
}

kahan.sum <- function(x) {
    sum <- c <- 0.0
    for (i in seq_along(x)) {
        y <- x[i] - c
        t <- sum + y
        c <- (t - sum) - y
        sum <- t
    }
    return(sum)
}

direct <- 10 ^ 7 / 3
data <- rep(1 / 3, 10 ^ 7)
simple.sum(data) - direct # 0.0003803289
kahan.sum(data) - direct  # -4.656613e-10
```

Metoda bezpośrednia (`simple.sum`) daje błąd w stosunku do wartości sumy wyliczonej dokładnie (`direct`) równy około 0,00038, który występuje na 11 pozycji dzie-

siętnej mimo nominalnej dokładności reprezentacji liczb równej ponad 15 pozycji dziesiętnych.

W przypadku algorytmu sumowania problem ten rozwiązuje tzw. algorytm Kahana [2], który dokonuje korekty błędu. Przedstawiony jest on w funkcji `kahan.sum`. Wykorzystuje on dodatkową zmienną `c` do przechowywania oszacowania błędu zaokrąglenia powstającego w procesie sumowania, która ma wartość równą różnicy pomiędzy oczekiwanym przyrostem wartości sumy równym `y`, a jego zaokrągloną wartością `t - sum`. Zauważmy, że jeśli obliczenia byłyby wykonywane z pełną precyzją to zmienna `c` stale przyjmowałaby wartość 0. Algorytm Kahana zastosowany do tego samego problemu daje błąd równy około $-4{,}656613 \cdot 10^{-10}$, który jest znacznie mniejszy.

W zagadnieniach optymalizacyjnych podobne problemy pojawiają się w algorytmach wielokrotnie przetwarzających warunki optymalizacji, np. w algorytmach rozwiązujących zadania programowania liniowego za pomocą metody sympleks. □

Przykład 2.4. Zagadnienie skończonej reprezentacji liczb zmiennoprzecinkowych powoduje, że podczas optymalizacji należy zwracać uwagę na błędy zaokrągleń. Dla przykładu rozważmy funkcję:

$$f(x) = (x + 100)^2 + 10000 - 200(x + 100). \tag{2.5}$$

Jest to po uproszczeniu funkcja x^2, jednak świadomie stosujemy bardziej złożony wzór, aby pokazać problemy związane z zaokrąglaniem. Wyliczenie jej wartości jest pokazane w kodzie 2.16.

Kod 2.16. Implementacja wyznaczania wartości funkcji $f(x)$ zadanej równaniem (2.5) na przedziale $[-2^{-18}, 2^{-18}]$

```
x <- seq(-2^-18, 2^-18, len = 64)
f.x <- (x + 100) ^ 2 + 10000 - 200 * (x + 100)
```

Wynik działania kodu znajduje się na rysunku 2.1. Widzimy że w wyniku niedokładności wykonywania obliczeń, możliwe są wartości ujemne funkcji, która teoretycznie ma minimum równe 0. Dodatkowo w 0 funkcja nie osiąga minimum i nie jest wypukła. Są to typowe problemy związane z utratą precyzji w przypadku wykonywania obliczeń na liczbach w reprezentacji zmiennoprzecinkowej. □

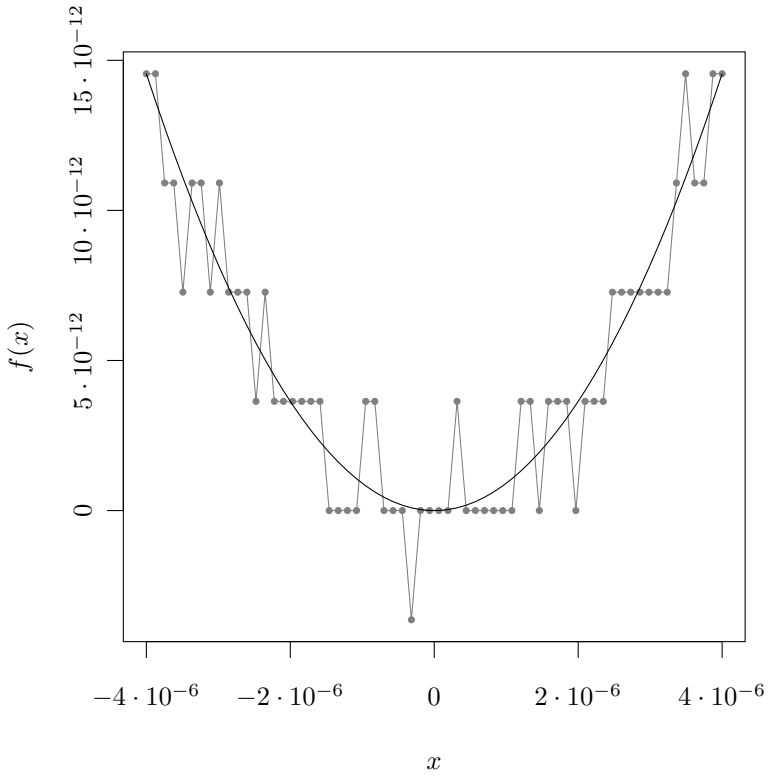
Przykład 2.5. Przejdźmy teraz do omówienia ważnego zagadnienia numerycznego wyznaczania pochodnej funkcji, które jest jednym z fundamentalnych w zagadnieniach optymalizacyjnych.

Dla dowolnej funkcji $f\colon \mathbb{R} \to \mathbb{R}$ zdefiniujmy jej *iloraz różnicowy centralny* w punkcie x dla parametru $\varepsilon \neq 0$ zgodnie ze wzorem:

$$\frac{\Delta f}{\Delta \varepsilon}(x) = \frac{f(x + \varepsilon) - f(x - \varepsilon)}{2\varepsilon}. \tag{2.6}$$

Zauważmy, że jeżeli funkcja f jest w punkcie x różniczkowalna, to:

$$\lim_{\varepsilon \to 0} \frac{\Delta f}{\Delta \varepsilon}(x) = f'(x). \tag{2.7}$$



Badana jest funkcja $f(x)$ zadana równaniem (2.5) na przedziale $[-2^{-18}, 2^{-18}]$. Linią czarną oznaczono wykres funkcji $f(x)$. Kolorem szarym zaznaczono jej przybliżenie numeryczne wyznaczone zgodnie z kodem 2.16.

Wykres 2.1: Wartość dokładna a przybliżenie numeryczne funkcji

Można zadać pytanie dlaczego obliczamy iloraz różnicowy biorąc pod uwagę odchylenie o ε do przodu i do tyłu. Aby to uzasadnić załóżmy, że funkcja f jest dwukrotnie różniczkowalna w sposób ciągły w otoczeniu punktu x_0 o promieniu większym niż ε. Wtedy mamy na podstawie wzoru Taylora:

$$f(x_0 + \varepsilon) = f(x_0) + f'(x_0)\varepsilon + \frac{f''(x_0)}{2}\varepsilon^2 + \mathcal{O}(\varepsilon^2). \tag{2.8}$$

Standardowy wzór na pochodną prowadzi do ilorazu różnicowego wprzód lub wstecz (w zależności od znaku ε):

$$\frac{f(x_0 + \varepsilon) - f(x_0)}{\varepsilon} = f'(x_0) + \frac{f''(x_0)}{2}\varepsilon + \mathcal{O}(\varepsilon). \tag{2.9}$$

Z kolei dla ilorazu różnicowego centralnego mamy:

$$\frac{f(x_0+\varepsilon)-f(x_0-\varepsilon)}{2\varepsilon} = f'(x_0)+\mathcal{O}(\varepsilon). \tag{2.10}$$

W związku z tym zauważamy, że teoretyczny błąd ilorazu różnicowego centralnego o rząd wielkości szybciej zbiega do 0 niż w przypadku metody wprzód/wstecz.

Niestety zależność teoretyczna opisana równaniami (2.7) i (2.10) w praktyce nie zachodzi ze względu na ograniczoną precyzję reprezentacji zmiennoprzecinkowych, co pokażemy na przykładzie. Rozważmy funkcję:

$$f(x) = 100\sin(x). \tag{2.11}$$

Wyznaczmy jej pochodną w punkcie $x = 1$. Wiemy, że $f'(x) = 100\cos(1)$. Kod 2.17 prezentuje implementację metody `num.deriv` wyliczającej iloraz różnicowy centralny oraz jego zastosowanie do funkcji $f(x)$ opisanej równaniem (2.11).

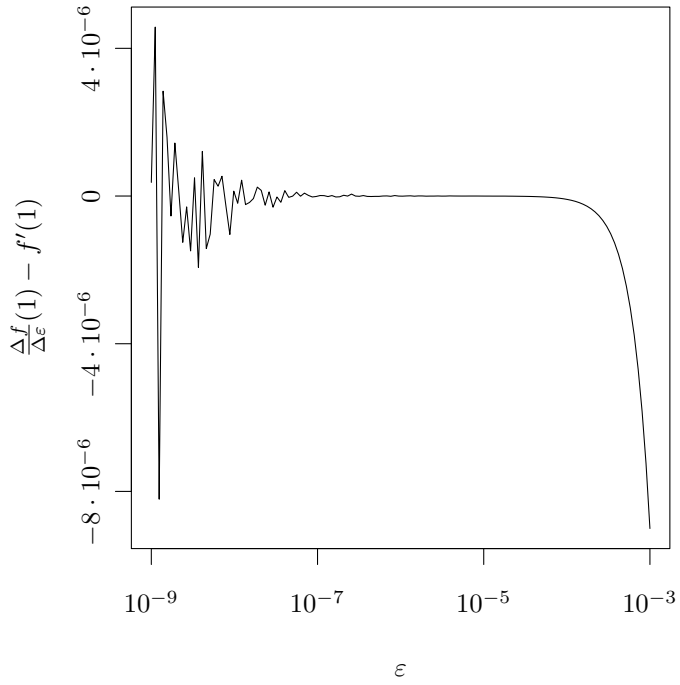
Kod 2.17. Implementacja procedury wyznaczającej wartość ilorazu różnicowego centralnego oraz jej zastosowania dla funkcji $100\sin(x)$

```
num.deriv <- function(f, x, eps) {
    (f(x + eps)-f(x - eps)) / (2 * eps)
}

sin100 <- function(x) {
    100 * sin(x)
}

y <- 10 ^ (seq(-3, -9, len = 128))
num.deriv(sin100, 1, y) - 100 * cos(1)
```

Wyliczana wartość jest odchyleniem ilorazu różnicowego centralnego od wartości pochodnej $100\cos(1)$ wyznaczonej analitycznie. Na rysunku 2.2 przedstawiono wykres prezentujący wynik obliczeń. Zauważmy, że gdy ε jest zbyt duże (większe niż 10^{-4}) i rośnie, to dokładność oszacowania pochodnej spada, ponieważ różniczka zbyt słabo przybliża pochodną. Z kolei gdy ε jest zbyt małe (mniejsze niż 10^{-7}) wartość różniczki staje się niestabilna ze względu na niedokładności numeryczne procedury.

R wykorzystuje w funkcji `optim` do przybliżania wartości pochodnej funkcji iloraz różnicowy centralny[20]. Domyślną wartością parametru ε jest 10^{-3}. Można ją zmienić modyfikując wartość parametrów kontrolnych `ndeps` lub `parscale`. Wartość ε jest określana jako `ndeps`$\cdot$`parscale`. Ponieważ parametr `parscale` jest ustawiany oddzielnie dla każdej zmiennej możliwe jest ustalenie dla nich różnych wartości ε wykorzystywanych do obliczenia ilorazu różnicowego centralnego w jednej procedurze obliczeniowej. Jest to przydatne w przypadkach, gdy różne zmienne w zadaniu optymalizacyjnym są podawane w znacznie różniących się skalach. Dodatkowo, gdy dokonywana jest optymalizacja z ograniczeniami w metodzie `L-BFGS-B` to przy obliczaniu różniczki algorytm

[20] Omówienie procedury `optim` przedstawione jest w rozdziale 4.



Badana jest funkcja $f(x)$ zadana równaniem (2.11). Wyznaczono odchylenie ilorazu różnicowego centralnego od pochodnej funkcji w punkcie $x = 1$ w zależności od parametru $\varepsilon \in [10^{-9}, 10^{-3}]$. Dla osi odciętych zastosowano skalę logarytmiczną.

Wykres 2.2: Odchylenie ilorazu różnicowego centralnego od wartości pochodnej funkcji

kontroluje aby nie przekroczyć wartości ograniczeń przy obliczaniu wartości funkcji w punkcie i gdy zachodzi taka potrzeba ogranicza wartość odchylenia do wartości granicznej. □

Przykład 2.6. Ważnym praktycznie rozszerzeniem poprzedniego przykładu jest problem numerycznego wyznaczenia hesjanu funkcji wielu zmiennych.

Dla dowolnej funkcji $f\colon \mathbb{R}^n \to \mathbb{R}$ zdefiniujmy jej *ilorazy różnicowe drugiego rzędu* w punkcie $\mathbf{x}$ dla parametru $\varepsilon \neq 0$ zgodnie ze wzorem:

$$\frac{\Delta^2 f}{\Delta^2 \varepsilon_i}(\mathbf{x}) = \frac{f(\mathbf{x} + \varepsilon_i) - 2f(\mathbf{x}) + f(\mathbf{x} - \varepsilon_i)}{\varepsilon^2}, \tag{2.12}$$

oraz dla $i \neq j$:

$$\frac{\Delta^2 f}{\Delta \varepsilon_i \Delta \varepsilon_j}(\mathbf{x}) = \frac{f(\mathbf{x} + \varepsilon_i + \varepsilon_j) - f(\mathbf{x} + \varepsilon_i - \varepsilon_j) - f(\mathbf{x} - \varepsilon_i + \varepsilon_j) + f(\mathbf{x} - \varepsilon_i - \varepsilon_j)}{4\varepsilon^2}, \tag{2.13}$$

gdzie ε_i oznacza wektor w $\mathbb{R}^n$, który przyjmuje wartość $\varepsilon \in \mathbb{R}$ na i-tej pozycji, a 0 w pozostałych.

Przejdźmy do uzasadnienia podanych wzorów. Dla $\frac{\Delta^2 f}{\Delta^2 \varepsilon_i}(\mathbf{x})$ zauważmy, że:

$$\frac{f(\mathbf{x}+\varepsilon_i) - 2f(\mathbf{x}) + f(\mathbf{x}-\varepsilon_i)}{\varepsilon^2} = \frac{\frac{f(\mathbf{x}+\varepsilon_i)-f(\mathbf{x})}{2(\varepsilon/2)} - \frac{f(\mathbf{x})-f(\mathbf{x}-\varepsilon_i)}{2(\varepsilon/2)}}{2(\varepsilon/2)}. \tag{2.14}$$

W związku z tym widzimy, że otrzymane wyrażenie najpierw przybliża $\frac{\Delta^2 f}{\Delta^2 \varepsilon_i}(\mathbf{x})$ dla parametru $\varepsilon/2$ za pomocą ilorazu różnicowego centralnego pochodnej cząstkowej pierwszego rzędu, a następnie ta pochodna cząstkowa w punktach $\mathbf{x} + \varepsilon_i/2$ i $\mathbf{x} - \varepsilon_i/2$ jest znowu przybliżana ilorazem różnicowym centralnym z parametrem $\varepsilon/2$.

Z kolei wzór na $\frac{\Delta^2 f}{\Delta \varepsilon_i \Delta \varepsilon_j}(\mathbf{x})$ możemy przekształcić do następującej postaci:

$$\frac{\frac{f(\mathbf{x}+\varepsilon_i+\varepsilon_j)-f(\mathbf{x}+\varepsilon_i-\varepsilon_j)}{2\varepsilon} - \frac{f(\mathbf{x}-\varepsilon_i+\varepsilon_j)-f(\mathbf{x}-\varepsilon_i-\varepsilon_j)}{2\varepsilon}}{2\varepsilon}. \tag{2.15}$$

I znowu otrzymany wzór wynika z dwókrotnego zastosowania przybliżenia pochodnej ilorazem różnicowym centralnym, jednak w tym wypadku wykorzystujemy parametr równy ε.

Wykorzystując wzory na ilorazy różnicowe drugiego rzędu możemy numerycznie wyznaczyć przybliżenie hesjanu funkcji dwóch zmiennych:

$$\nabla^2 f(\mathbf{x}) \approx \begin{bmatrix} \frac{\Delta^2 f}{\Delta^2 \varepsilon_1}(\mathbf{x}) & \cdots & \frac{\Delta^2 f}{\Delta \varepsilon_1 \Delta \varepsilon_n}(\mathbf{x}) \\ \vdots & \ddots & \vdots \\ \frac{\Delta^2 f}{\Delta \varepsilon_n \Delta \varepsilon_1}(\mathbf{x}) & \cdots & \frac{\Delta^2 f}{\Delta^2 \varepsilon_n}(\mathbf{x}) \end{bmatrix}. \tag{2.16}$$

Warto zwrócić uwagę, że tak wyznaczona aproksymacja jest macierzą symetryczną ponieważ (pomijając błędy zaokrągleń) $\frac{\Delta^2 f}{\Delta \varepsilon_i \Delta \varepsilon_j}(\mathbf{x}) = \frac{\Delta^2 f}{\Delta \varepsilon_j \Delta \varepsilon_i}(\mathbf{x})$.

Kod 2.18 prezentuje implementację metody `num.hess` wyznaczającej hesjan numerycznie. Wynik obliczeń jest porównany z hesjanem wyznaczonym w procedurze `sym.hess` metodą symboliczną (analityczną) za pomocą funkcji `deriv3`. W rozważanym przypadku przybliżenie hesjanu należy uznać za dokładne.

Kod 2.18. Implementacja procedury wyznaczającej przybliżenie hesjanu oraz jej zastosowania dla funkcji $e^{x_1^2+x_2^2}$

```
num.hess <- function(f, x, eps) {
    deriv2 <- function(i, j) {
        if (i == j) {
            return((f(x + epsm[i,]) - 2 * f(x) +
                      + f(x - epsm[i,]) ) / eps ^ 2)
        }
        (f(x + epsm[i,] + epsm[j,]) -
         f(x + epsm[i,] - epsm[j,]) -
```

```
        f(x - epsm[i,] + epsm[j,]) +
        f(x - epsm[i,] - epsm[j,])) /
       (4 * eps ^2)
  }

  n <- length(x)
  epsm <- diag(n) * eps
  idx <- expand.grid(i = 1:n, j = 1:n)
  dnames <- paste("x", 1:n, sep="")
  matrix(mapply(deriv2, idx$i, idx$j), nrow = n,
         dimnames = list(dnames, dnames))
}

test <- function(x) {
  exp(x[1] ^ 2 + x[2] ^ 2)
}

num.hess(test, c(1, 2), 1e-4)
#          x1       x2
# x1  890.479 1187.305
# x2 1187.305 2671.437

sym.hess <- function(x) {
  x1 <- x[1]
  x2 <- x[2]
  attr(eval(deriv3(expression(exp(x1 ^ 2 + x2 ^ 2)),
       c("x1", "x2"))), "hessian")[1, , ]
}

sym.hess(c(1, 2))
#          x1       x2
# x1  890.479 1187.305
# x2 1187.305 2671.437

num.hess(test, c(1, 2), 1e-4) - sym.hess(c(1, 2))
#              x1           x2
# x1 2.056412e-05 6.034064e-05
# x2 6.034064e-05 7.021886e-05
```

W R dostępny jest pakiet `numDeriv`, który umożliwia numeryczne wyliczanie pierwszych i drugich pochodnych funkcji wielu zmiennych. □

Przykład 2.7. W implementacji procedury `optim` w R często wykonywaną procedurą jest zmiana wartości punktu, w którym badana jest wartość funkcji. Oznaczmy stary punkt przez `x` i badane jego przesunięcie o `d`. W celu sprawdzenia, czy przesunięcie

jest dostatecznie duże aby nie było pomijalne w wyniku zaokrąglenia wykonywany jest test `10 + x == 10 + (x + d)` a nie `x == x + d`. W kodzie 2.19 pokazane jest w jakich przypadkach te dwa podejścia powodują powstawanie różnicy w wynikach porównania.

Kod 2.19. Badanie sposobu porównywania równości liczb

```
rel <- 10
rng <- -1000:1000
base <- expand.grid(s = c(-1, 1), px = rng, pd = rng)
x <- 2 ^ base$px
d <- base$s * 2 ^ base$pd
eq <- x == (x + d)
eq.rel <- (rel + x) == (rel + x + d)
rule <- (abs(d) < 1e-15) & (abs(x / d) < 5e15)
table(eq, eq.rel, rule)
# , , rule = FALSE
#
#        eq.rel
# eq        FALSE    TRUE
#   FALSE 3208039    1003
#   TRUE        3 3794701
#
# , , rule = TRUE
#
#        eq.rel
# eq        FALSE    TRUE
#   FALSE       2 1004254
#   TRUE        0       0
```

Porównujemy sytuacje, w których `x` i `d` są potęgami 2 w zakresie od 2^{-1000} do 2^{1000}. W zmiennej `eq` przechowujemy wynik porównania bezpośredniego, a `eq.rel` z dodaniem 10 po obu stronach równości. Okazuje się, że, w przybliżeniu, wyniki tych porównań nie są zgodne, gdy `abs(d)` jest mniejsze niż 10^{-15} oraz jednocześnie `abs(x / d)` jest mniejsze niż $5 \cdot 10^{15}$ — co jest opisane za pomocą zmiennej `rule`. W takich wypadkach reguła `x == x + d` wskazywałaby, że liczby są różne, a dodanie 10 po obu stronach porównanie prowadzi do stwierdzenia, że są one równe. Takie podejście podyktowane jest założeniem, że zbyt małe (jednocześnie bezwzględnie i relatywnie) zmiany wartości badanego punktu w procesie optymalizacji powinny być uznawane jako nieistotne. □

Przykład 2.8. W implementacji algorytmu BFGS[21] w procedurze `optim` w R jako kryterium zatrzymania optymalizacji wykorzystywane jest kryterium *relatywnej* tolerancji. Procedura zatrzymuje się, jeśli nowy punkt, w którym dokonywany jest pomiar

[21] Jest on omówiony w rozdziale 4

funkcji celu nie zapewnia dostatecznie dużej poprawy w stosunku do aktualnego minimum.

Oznaczmy przez `m` wartość funkcji w dotychczasowym minimum, przez `f` jej wartość w nowo testowanym punkcie, a przez `r` dodatnią wartość kryterium relatywnej tolerancji. Kryterium zatrzymania optymalizacji jest: `abs(f - m) <= r * (abs(f) + r)` i łączy ono ze sobą w jednej formule warunek relatywnej i bezwzględnej tolerancji.

Przeanalizujmy własności tego kryterium dla różnych wartości funkcji `f` i standardowej wartości parametru `r` równej `sqrt(.Machine$double.eps)`. Wartość ta jest równa dokładnie 2^{-26}. Oznacza, to, że jeśli wartość `abs(f)` jest duża (większa niż 2^{26}) to w wyniku zaokrąglenia `abs(f) + r` jest równe `abs(f)` i kryterium to testuje wyłącznie relatywną tolerancję. Z kolei jeżeli wartość `abs(f)` jest bardzo mała (nie większa niż 2^{-79}) to wyrażenie `abs(f) + r` jest równe `r` i testowany jest wyłącznie warunek kryterium bezwzględnej tolerancji. Dla pośrednich wartości `abs(f)` testowana jest suma obu warunków.

W tym miejscu warto wymienić standardowe typy kryteriów zatrzymania obliczeń w algorytmach optymalizacyjnych. Dalej będziemy oznaczali kolejne generowane przez aglorytm minimalizujący funkcję f punkty będące kandydatami na rozwiązanie optymalne przez x_k. Do typowych kryteriów zatrzymania zaliczamy:

1) warunek bezwzględnej odległości od punktu optymalnego (x^* jest nieznanym rozwiązaniem optymalnym):
$$|x_{k+1} - x^*| < \varepsilon;$$

2) warunek względnej odległości od punktu optymalnego (x^* jest nieznanym rozwiązaniem optymalnym):
$$|(x_{k+1} - x^*)/x^*| < \varepsilon;$$

3) warunek bezwzględnej zmiany wartości punktu optymalnego:
$$|x_{k+1} - x_k| < \varepsilon;$$

4) warunek względnej zmiany wartości punktu optymalnego:
$$|(x_{k+1} - x_k)/x_k| < \varepsilon;$$

5) warunek bezwzględnej zmiany wartości rozwiązania optymalnego:
$$|f(x_{k+1}) - f(x_k)| < \varepsilon;$$

6) warunek względnej zmiany wartości rozwiązania optymalnego:
$$|(f(x_{k+1}) - f(x_k))/f(x_k)| < \varepsilon;$$

7) warunek bezwzględnego odchylenia od znanej wartości, najczęściej stosowany przy testowaniu odchylenia pochodnej funkcji od 0:
$$|f'(x_k)| < \varepsilon;$$

8) warunek maksymalnej dopuszczalnej liczby iteracji algorytmu:

$$k > k_{\max}.$$

Oczywiście — jak pokazano w niniejszym przykładzie — w praktyce mogą być stosowane również kombinacje tych kryteriów. □

Przykład 2.9. W przypadkach gdy standardowa precyzja obliczeń zmiennoprzecinkowych oferowana przez R jest niewystarczająca możliwe jest wykorzystanie pakietu `Rmpfr`. Za jego pomocą można dokonywać obliczeń przy wykorzystaniu reprezentacji zmiennoprzecinkowej o określonej przez użytkownika precyzji. Kod 2.20 prezentuje prosty przykład wykorzystania tego pakietu.

Kod 2.20. Przykład wykorzystania pakietu `Rmpfr` do wykonywania obliczeń zmiennoprzecinkowych o wysokiej precyzji

```
1  library(Rmpfr)
2
3  format(1 / 3, digits=22) # 0.333333333333333148296
4  mpfr(1, 100) / 3         # 0.33333333333333333333333333333346
5
6  f <- function(x) {
7      1 + x^2 / 10 ^ 20
8  }
9
10 f(1) == f(0)                        # TRUE
11 f(mpfr(1, 100)) == f(mpfr(0, 100)) # FALSE
12
13 optimize(f, lower = -1, upper = 1, tol = 1e-20)$minimum
14 # 1
15
16 optimizeR(f, lower = -1, upper = 1, tol = 1e-20)$minimum
17 # 1.19867284861731718865298225312629987081 21e-20
```

W pierwszej części kodu w linijkach 3–4 porównujemy standardowe przybliżenie liczby 1/3 oraz przybliżenie obliczone z dokładnością do 100 binarnych cyfr znaczących. Zauważmy, że przybliżenie dokładnej wartości jest znacznie lepsze.

W dalszej części kodu w linijkach 6–8 definiujemy funkcję, która posiada minimum w 0, ale jest wokół niego bardzo *płaska*. Przy wykorzystaniu obliczeń na standardowych liczbach zmiennoprzecinkowych wartość funkcji w 0 i w 1 jest taka sama. Z kolei przy wykorzystaniu reprezentacji zmiennoprzecnkowej posiadającej 100 znaczących cyfr binarnych te wartości są różne. Sytuacja ta ma istotne znaczenie dla procesu optymalizacji funkcji `f`. Próba jej przeprowadzenia za pomocą standardowej funkcji `optimize`[22] z tolerancją równą 10^{-20} na przedziale $[-1, 1]$ prowadzi do wyniku równego 1, który jest istotnie nieprawidłowy. Zwiększenie precyzji obliczeń przy wykorzystaniu

[22] Jest ona szczegółowo omówiona w rozdziale 3.

funkcji `optimizeR` z pakietu `Rmpfr` pozwala uzyskać wynik odległy od rzeczywistego minimum o wartość równą w przybliżeniu oczekiwanej tolerancji. □

Przykład 2.10. Na zakończenie rozdziału przedstawmy typowy problem optymalizacyjny pojawiający się przy estymacji parametrów metodą największej wiarygodności. Załóżmy, że dysponujemy 20000 obserwacjami pochodzącymi z rozkładu normalnego o nieznanej średniej i odchyleniu standardowym. Oznaczmy je przez $x_1, x_2, \ldots, x_{20000}$. Zgodnie z teorią estymacji asymptotycznie nieobciążony i efektywny estymator średniej m i odchylenia standardowego s otrzymamy znajdując maksimum funkcji wiarygodności:

$$\prod_{i=1}^{20000} f_{m,s}(x_i), \tag{2.17}$$

gdzie $f_{m,s}$ jest funkcją opisującą gęstość rozkładu normalnego o średniej m i odchyleniu standardowym s. Kod 2.21 przedstawia różne możliwe podejścia do numerycznej minimalizacji tej funkcji.

Kod 2.21. Przykład estymacji przy wykorzystaniu metody największej wiarygodności

```
library(Rmpfr)

set.seed(1)
x <- rnorm(20000)

lik <- function(x, m, s) {
    prod(dnorm(x, m, s))
}

optim(c(1, 1), function(arg) { -lik(x, arg[1], arg[2]) })$par
# 1 1
lik(x, 1, 1)
# 0
prod(mpfr(dnorm(x, 1, 1), 53))
# 7.9735147080153480e-16729

llik <- function(x, m, s) {
    sum(log(dnorm(x, m, s)))
}

optim(c(1, 0.1), function(arg) { -llik(x, arg[1], arg[2]) })
# Error in optim(c(1, 0.1), function(arg) { :
#  function cannot be evaluated at initial parameters
llik(x, 1, 0.1)
# -Inf
range(dnorm(x, 1, 0.1))
```

```
27 # 0.000000 3.989423
28
29 llik.safe <- function(x, m, s) {
30     loglik <- log(dnorm(x, m, s))
31     loglik[is.infinite(loglik)] <- floor(log(2^-1074))
32     sum(loglik)
33 }
34
35 optim(c(1,0.1), function(arg) { -llik.safe(x, arg[1], arg[2]) })$par
36 # -0.005444574  1.002203655
37 c(mean(x), sd(x))
38 # -0.005363553  1.001600824
```

W kodzie wykorzystujemy funkcję `optim`, która poszukuje minimum funkcji wielu zmiennych[23]. W liniach 3–4 generujemy 20000–elementową próbę losową. Najprostsze podejście do optymalizacji wykorzystuje bezpośrenio funkcję zadaną równaniem (2.17) i podejmuje próbę znalezienia optimum startując z punktu $m = 1$ i $s = 1$. W linii 10 definiujemy pomocniczą funkcję zwracającą `-lik` ponieważ funkcja wiarygodności powinna być maksymalizowana, a funkcję `optim` poszukuje domyślnie minimum.

Niestety w wyniku optymalizacji rozwiązanie startowe nie jest poprawione. W 12 linii kodu stwierdzamy, że funkcja wiarygodności w tym punkcie przyjmuje wartość 0. Jest to wynik będący skutkiem zaokrąglenia — funkcja wiarygodności przyjmuje wartość mniejszą niż najmniejsza nieujemna zdenormalizowana liczba zmiennoprzecinkowa możliwa do przedstawienia w formacie *binary64*, która zgodnie z rozważaniami przedstawionymi w poprzednim podrozdziale jest równa 2^{-1074}. Dodatkowo potwierdzamy to wykorzystując obliczenia o podniesionej precyzji za pomocą pakietu `Rmpfr`. W linijce 14 stwierdzamy, że rzeczywiście wartość tej funkcji jest znacznie mniejsza niż ta liczba. Wartość ta jest tak mała, że w otoczeniu punktu $m = 1$ i $s = 1$ obliczona numerycznie wartość funkcji wiarygodności wszędzie przyjmuje wartość 0, a więc jest stała i procedura optymalizacyjna przerywa działanie. Czytelnik może sprawdzić, że zmiana startowych m i s nie pozwala na rozwiązanie tego problemu.

Standardowym rozwiązaniem tego problemu jest maksymalizacja logarytmu funkcji wiarygodności. Funkcja taka jest zaimplementowana w liniach 17–19 kodu. Niestety i w tym wypadku optymalizacja nie jest skuteczna. Problemem jest fakt, że funkcja `llik` dla $m = 1$ i $s = 0{,}1$ przyjmuje wartość `-Inf`. W linijce 26 sprawdzamy, że przyczyną tego zjawiska jest fakt, że w wyniku zaokrąglenia dla niektórych obserwacji oszacowanie funkcji gęstości jest tak małe, że w zaokrągleniu przyjmuje wartość równą dokładnie 0.

Rozwiązanie tego problemu jest przedsawione w funkcji `llik.safe`. Zastępujemy w niej wartości `-Inf` liczbami mniejszymi niż najmniejsza możliwa do otrzymania wartość logarytmu. Ta najmniejsza wartość jest osiągana dla argumentu logarytmu równego 2^{-1074} będącego najmniejszą dodatnią zdenormalizowaną liczbą możliwą do reprezentacji w formacie *binary64*. Ponieważ `log(2^-1074)` wynosi około $-744{,}4401$,

[23] Jest ona szczegółowo omówiona w rozdziale 4.

więc zaokrąglając ją w dół otrzymujemy wartość -750. Stosując taką procedurę jednocześnie zapewniamy, że: (i) nie będziemy otrzymali wartości `-Inf` w wyniku działania procedury oraz (ii) mamy pewność, że przypadki w których oszacowana wartość funkcji gęstości wynosi 0 są najmniej pożądane w optymalizacji. Optymalizacja wykorzystująca tak zmodyfikowaną funkcję celu daje zadowalające wyniki, co stwierdzamy porównując je do średniej i odchylenia standardowego z próby.

Problemy numeryczne przedstawione w niniejszym przykładzie są typowe i często pojawia się potrzeba wprowadzania modyfikacji do oryginalnego sformułowania funkcji celu ze względu na ograniczenia numeryczne.

Na zakończenie przykładu podkreślmy, że przedstawiona procedura nie jest jeszcze doskonała, ponieważ nie będzie sobie poprawnie radziła z przypadkami w których odchylenie standardowe s byłoby ujemne, na przykład w wyniku niepoprawnego podania punktu startowego optymalizacji. Sposoby radzenia sobie z takimi problemami przedstawione są w rozdziale 5 omawiającym zagadnienia optymalizacji, w których występują ograniczenia na zakres zmienności argumentów funkcji celu. □

Zadania

1) Wyjaśnij dlaczego wyrażenie `0.7 / 0.1 == 7` przyjmuje wartość `FALSE`.
2) Zbadaj czy zawsze wyrażenie `1 / (1 / x) == x` przyjmuje wartość `TRUE` gdy `x` jest liczbą zapisaną w formacie *binary64*. Oszacuj jaka jest szansa, że liczba wylosowana za pomocą komendy `x = runif(1)` nie spełnia tego warunku.
3) Niech $x_1, x_2 \in \{1, 2, \ldots, 99\}$ i $n \in \{1, 2, \ldots, 10\}$. Oblicz w `R` w jakim procencie przypadków w zależności od n zachodzi $(x_1/n) + (x_2/n) \neq (x_1 + x_2)/n$ ze względu na przybliżone wykonywanie obliczeń. Jakie wyciągasz wnioski?
4) Porównaj w otoczeniu 0 wartości wyrażeń $1 - \cos(x)$ oraz $\sin^2(x)/(1 + \cos(x))$. Które wyrażenie daje lepsze przybliżenie w przypadku obliczeń numerycznych i dlaczego?
5) Niech $a \geq b \geq c > 0$ będą długościami boków trójkąta. Niech $s = (a + b + c)/2$. Standardowy sposób wyliczenia pola trójkąta to:
$$P = \sqrt{s(s-a)(s-b)(s-c)}. \tag{2.18}$$
Kahan [3] zaproponował następujący zmodyfikowany sposób obliczenia tej wartości (z zachowaniem nawiasowania):
$$P = \frac{\sqrt{(a+(b+c))(c-(a-b))(c+(a-b))(a+(b-c))}}{4}. \tag{2.19}$$
Porównaj dokładność obu wzorów dla $a = b = 10^8$ oraz, $c = 10^{-8}$ (zwróć uwagę, że jest to trójkąt równoramienny o polu 1/2).
6) Wykorzysując pakiet `Rmpfr` porównaj dokładność wyznaczenia wartości wyrażenia `(1 + 1 / n) ^ n` w zależności od `n`, gdzie `n` jest liczbą w formacie *binary64*.
7) Wzory służące do wyznaczania wariancji z próby $\sum\left(x_i - \sum x_i/n\right)^2/(n-1)$ oraz $\sum x_i^2/(n-1) - \left(\sum x_i\right)^2/(n^2 - n)$ są matematycznie równoważne. Porównaj ich

dokładność numeryczną wyznaczenia wariancji biorąc próbę $(x_1, x_2) = (10^{10} + 1, 10^{10} + 2)$.

8) Zaproponuj modyfikację procedury `num.deriv` z kodu 2.17 tak, aby można było za jej pomocą wyliczać iloraz różnicowy centralny dla funkcji wielu zmiennych $f\colon \mathbb{R}^n \to \mathbb{R}$. Zastosuj procedurę dla funkcji $x_1 + x_2$ w punkcie $(10^{-8}, 10^8)$ i wartości ε należącej do przedziału $[10^{-8}, 10^{-4}]$. Oceń precyzję otrzymywanych wyników w stosunku do wartości analitycznych. Badanie powtórz dla funkcji $x_1 x_2$ badanej w tym samym punkcie. Dlaczego precyzja otrzymanych wyników w tym wypadku jest inna?
9) Rozważmy funckję $f(\mathbf{x}) = x_1 x_2 (x_1^2 - x_2^2)/(x_1^2 + x_2^2)$ dla $\mathbf{x} \neq (0, 0)$ oraz $f(0, 0) = 0$. Wyznacz analitycznie jej pochodne rzędu pierwszego i drugiego w punkcie $(0, 0)$. Następnie wyznacz te pochodne numerycznie. Kiedy i dlaczego otrzymane wyniki nie są zgodne?
10) Niech $x = 4.7$. Oblicz w R wartość następujących wyrażeń: $x - 4.7$, $x - 4 - 0.7$, $(10x - 47)/10$, $(10(x - 4) - 7)/10$.
11) Wyjaśnij z jakiego powodu w R zostały udostępnione funkcje `expm1` oraz `log1p`. Porównaj dokładność obliczeń wykorzystujących te funkcje oraz podstawowe funkcje `exp` oraz `log`. Otrzymane wyniki zweryfikuj wykonując obliczenia z precyzją 1000 znaczących cyfr binarnych wykorzystując pakiet `Rmpfr`.
12) Należy wygenerować 10^6+1 liczb rozłożonych równomiernie na przedziale $[10^6, 10^6+1]$. Zaproponowano dwa sposoby na wykonanie tego zadania. Przedstawione są one w kodzie 2.22 i zapisują swój wynik odpowiednio w zmiennych `x1` i `x2`. Porównaj otrzymane wyniki zwracając szczególną uwagę na to, czy: (i) otrzymane liczby są rozłożone równomiernie oraz (ii) jaka jest dokładność wyniku dla ostatniej wyznaczonej wartości w otrzymanych wektorach. Zmodyfikuj zaproponowany kod tak, aby przeprowadzić obliczenia dla $2^20 + 1$ liczb rozłożonych równomiernie na przedziale $[10^6, 10^6 + 1]$. Dlaczego tym razem otrzymane wnioski są różne?

Kod 2.22. Dwie metody wyznaczenia sekwencji $10^6 + 1$ liczb rozłożonych równomiernie na przedziale $[10^6, 10^6 + 1]$

```
x1 <- seq(10 ^ 6, 10 ^ 6 + 1, len = 10 ^ 6 + 1)
x2 <- numeric(10 ^ 6 + 1)
x2[1] <- 10 ^ 6
for (i in 1:10 ^ 6) {
    x2[i + 1] <- x2[i] + 10^-6
}
```

13) Wyjaśnij dlaczego wyrażenie `is.integer(3)` zwraca wartość `FALSE`. Zaproponuj alternatywne metody pozwalające na sprawdzenie czy dana liczba jest całkowita i przetestuj je na liczbach 12345678901 i 12345678901,1.
14) Wyjaśnij dlaczego funkcja `floor` zwraca wynik swojego działania w formacie zmiennoprzecinkowym?

3. Optymalizacja funkcji jednej zmiennej

W niniejszym rozdziale omawiamy procedury numeryczne przeznaczone do poszukiwania minimum lokalnego funkcji jednej zmiennej. W podrozdziale prezentującym podstawy teoretyczne dokonujemy przeglądu zarówno bezgradientowych jak i gradientowych metod wyznaczania minimum funkcji. Metody te same w sobie często są wykorzystywane w problemach praktycznych. Proste sytuacje tego typu przedstawione w podrozdziale poświęconemu zastosowaniom. Dodatkowo metody te stanowią często składnik bardziej ogólnych metod optymalizacyjnych wykorzystywanych w przypadku funkcji wielu zmiennych.

Podstawy teoretyczne

Podstawową techniką wykorzystywaną w numerycznych algorytmach optymalizacyjnych jest podejście iteracyjne.W podejściu tym generowana jest sekwencja $x(k)$ przybliżeń poszukiwanego minimum x^*. W niniejszym rozdziale będziemy przyjmowali, że $x(k), x^* \in \mathbb{R}$, jednak rozważania w naturalny sposób przenoszą się na wielowymiarowe przestrzenie euklidesowe.

Oznaczmy $e_k = |x(k) - x^*|$. Ważną własnością algorytmów optymalizacyjnych jest ich tempo zbieżności. W celu jego określenia rozważany jest iloraz:

$$\limsup_{k \to +\infty} \frac{e_{k+1}}{e_k^\alpha} = \beta_\alpha, \tag{3.1}$$

gdzie $\alpha \geq 1$. Rozróżniamy następujące przypadki:

1) jeżeli wartość β_1 jest mniejsza niż 1 to mówimy, że że ciąg $x(k)$ ma *liniowe* tempo zbieżności;
2) jeżeli wartość β_1 jest równa 1 to mówimy, że że ciąg $x(k)$ ma *subliniowe* tempo zbieżności;
3) jeżeli wartość β_1 jest równa 0 to mówimy, że że ciąg $x(k)$ ma *superliniowe* tempo zbieżności;
4) jeżeli dla $\alpha > 1$ wartość β_α jest skończona, to mówimy, że ciąg $x(k)$ ma tempo zbieżności rzędu α.

W szczególności tempo zbieżności rzędu 2 nazywane jest kwadratowym.

Dla niektórych sekwencji $x(k)$ powyższe warunki nie muszą być spełnione, ponieważ mimo, że w ogólności zbiegają one szybko, to jednak ich tempo zbieżności jest

zmienne. W takich sytuacjach wprowadzamy pojęcie ε-zbieżności. Mianowicie, jeśli istnieje taki ciąg ε_k zbieżny do 0 w tempie liniowym, że $e_k \leq \varepsilon_k$ to mówimy, że ciąg $x(k)$ jest ε-liniowe tempo zbieżności. Analogicznie definiujemy pozostałe przypadki.

Przejdźmy teraz do przedstawienia podstawowych algorytmów optymalizacyjnych. Rozważania zaczniemy od metod nie wykorzystujących informacji o pochodnej funkcji celu.

Rozważmy funkcję ciągłą $f\colon \mathbb{R} \to \mathbb{R}$, dla której szukamy minimum lokalnego w przedziale $[x_l, x_u]$. Wybierzmy dwa punkty $x_1, x_2 \in (x_l, x_u)$, takie, że $x_1 < x_2$. Zauważmy, że jeżeli $f(x_1) \leq f(x_2)$ to w przedziale $[x_l, x_2]$ będzie leżało przynajmniej jedno minimum lokalne funkcji f w dziedzinie zadanej przedziałem $[x_l, x_u]$. Aby to stwierdzić zauważmy, że funkcja f na przedziale $[x_l, x_2]$ osiąga swój kres dolny. Oznaczmy zbiór punktów, w którym jest on osiągany przez X. Ale ponieważ $f(x_1) \leq f(x_2)$, więc $X \setminus x_2 \neq \emptyset$. W takim razie w przedziale $[x_l, x_2[$ istnieje minimum lokalne funkcji f, co oznacza, że jest to również minimum lokalne na przedziale $[x_l, x_h]$. Podobnie jeżeli $f(x_1) \geq f(x_2)$ to takie minimum lokalne będzie leżało w przedziale $[x_1, x_u]$.

Powyższe spostrzeżenie oznacza, że poszukując minimum lokalnego funkcji f i dysponują jej oszacowaniami w dwóch różnych punktach x_1 i x_2 przedział $[x_l, x_u]$ możemy zawęzić zastępując x_l lub x_u odpowiednio przez x_1 lub x_2 w zależności od tego jaka jest relacja wartości $f(x_1)$ i $f(x_2)$. W przypadku gdy $f(x_1) = f(x_2)$ ograniczenie przedziału można wybrać w dowolny z dwóch powyższych sposobów. Obserwacja ta jest podstawą procedur mających na celu znalezienie minimum funkcji jednej zmiennej niewykorzystujących pochodnej funkcji.

Załóżmy dalej, że oczekujemy, że minimum lokalne funkcji f będzie oszacowane z dokładnością ε. Mówiąc precyzyjnie, jeżeli x_0 jest minimum lokalnym f a x_s jest jego oszacowaniem, to wymagamy aby $|x_0 - x_s| < \varepsilon$. W związku z tym proces opisany w poprzednim paragrafie trzeba powtarzać wielokrotnie tak aby krańce przedziału $[x_l, x_u]$ spełniały warunek $x_u - x_l < 2\varepsilon$. Wtedy punkt $x_s = (x_u + x_l)/2$ spełnia żądany warunek.

Przyjrzyjmy się teraz prostemu algorytmowi wykorzystującemu tą koncepcję. Nazywany jest on *przeszukiwaniem potrójnym* (ang. *ternary search*). W podejściu tym punkty x_1 i x_2 są dobrane w taki sposób, aby dzieliły przedział $[x_l, x_u]$ na trzy równe części, a więc:

$$\begin{array}{rcl} x_1 & = & (2x_l + x_u)/3 \\ x_2 & = & (x_l + 2x_u)/3 \end{array} . \tag{3.2}$$

Kod 3.1 przedstawia implementację tej procedury.

Kod 3.1. Implementacja procedury przeszukiwania potrójnego

```
1 ternary <- function(f, lower, upper, tol) {
2     f.lower <- f(lower)
3     f.upper <- f(upper)
4     while (abs(upper - lower) > 2 * tol) {
5         x1 <- (2 * lower + upper) / 3
6         f.x1 <- f(x1)
```

```
        x2 <- (lower + 2 * upper) / 3
        f.x2 <- f(x2)
        if (f.x1 < f.x2) {
            upper <- x2
            f.upper <- f.x2
        } else {
            lower <- x1
            f.lower <- f.x1
        }
    }
    return((upper + lower) / 2)
}
```

W jednym kroku algorytmu badany przedział jest skracany o 1/3. W takim razie po n krokach procedury oryginalna długość przedziału zmniejszona jest $1{,}5^n$ razy. Zauważmy, że w takiej sytuacji oszacowanie wartości funkcji f jest wywołane $2n$ razy. Można więc powiedzieć, że jedno oszacowanie wartości funkcji pozwala średnio na zmniejszenie długości przedziału $\sqrt{3/2} \approx 1{,}22$ razy.

Przyjmijmy, że po k-tej iteracji przybliżeniem rozwiązania optymalnego $x(k)$ jest środek przedziału $[x_l, x_u]$, a przez x^* oznaczymy poszukiwane minimum funkcji. Widzimy, że $|x(k) - x^*|$ możemy oszacować z góry przez $\varepsilon_k = (x_u - x_l)/2$. Ponieważ jednym kroku algorytmu badany przedział jest skracany o 1/3, więc $\varepsilon_{k+1}/\varepsilon_k = 2/3$. W związku z tym procedura przeszukiwania potrójnego ma tempo zbieżności ε-liniowe.

Bardziej efektywnym sposobem doboru punków x_1 i x_2 jest tak zwane przeszukiwanie według złotego podziału. W tym wypadku oznaczając złotą proporcję przez $\varphi = (1 + \sqrt{5})/2$ mamy:

$$\begin{array}{rcl} x_1 & = & (\varphi - 1)x_l + (2 - \varphi)x_u \\ x_2 & = & (2 - \varphi)x_l + (\varphi - 1)x_u \end{array} . \tag{3.3}$$

W takim razie w każdym kroku procedury długość przedziału zmniejszona jest $\varphi \approx 1.62$ razy. Jednak prawdziwa korzyść z zastosowania złotego podziału wynika z faktu, że w jednym kroku wymagane jest tylko *jednokrotne* oszacowanie wartości funkcji. Aby to zauważyć rozważmy najpierw przypadek $f(x_1) \leq f(x_2)$. W tej sytuacji x_2 zastąpi wartość x_u. Ale w takim razie w kolejnym kroku procedury wartość x_2 będzie wynosiła (wykorzystujemy oznaczenia z poprzedniego kroku procedury):

$$\begin{aligned} (2 - \varphi)x_l + (\varphi - 1)x_2 &= (2 - \varphi)x_l + (\varphi - 1)((2 - \varphi)x_l + (\varphi - 1)x_u) = \\ &= \varphi(2 - \varphi)x_l + (\varphi - 1)^2 x_u = (\varphi - 1)x_l + (2 - \varphi)x_u. \end{aligned} \tag{3.4}$$

Zauważmy, że zgodnie z równaniem (3.3) nowo wyznaczone x_2 jest dokładnie równe x_1 wyznaczonemu w poprzednim kroku procedury, więc dysponujemy już wartością funkcji w tym punkcie. Rozumowanie w przypadku gdy $f(x_2) \leq f(x_1)$ jest analogiczne tylko w tym wypadku nowe x_1 jest równe x_2 z poprzedniego kroku procedury.

Kod 3.2 przedstawia implementację procedury przeszukiwania według złotego podziału. Zawarte jest w nim jedno usprawnienie implementacyjne w celu uniknięcia zamiany roli punktów x_1 i x_2 opisanej w poprzednim akapicie. Za każdym razem wyznaczamy punkt x_2. Jest to problem gdy $f(x_1) < f(x_2)$ ponieważ w tym wypadku nowo wyznaczone x_2 pokryłoby się ze starym x_1 aby temu zapobiec zamieniane są role zmiennych x_1 i x_2, co jest możliwe ponieważ wzór (3.3) jest ze względu na ich zamianę symetryczny. Oznacza to, że jest możliwe, że w trakcie działania procedury `lower` > `upper`, co przeczy intuicyjnemu rozumieniu oznaczeń tych zmiennych. Zmienną `lower` należy rozumieć jako *bliższą* do tego z punktów x_1 i x_2, który nie stał się nowym końcem przeszukiwanego przedziału.

Kod 3.2. Implementacja procedury przeszukiwania według złotego podziału

```
golden <- function(f, lower, upper, tol) {
    ratio <- 2 / (3 + sqrt(5))
    x1 <- (1 - ratio) * lower + ratio * upper
    f.x1 <- f(x1)
    while (abs(upper - lower) > 2 * tol) {
        x2 <- (1 - ratio) * x1 + ratio * upper
        f.x2 <- f(x2)
        if (f.x1 < f.x2) {
            upper <- lower
            lower <- x2
        } else {
            lower <- x1
            x1 <- x2
            f.x1 <- f.x2
        }
    }
    return((upper + lower) / 2)
}
```

Porównajmy szybkość przeszukiwania potrójnego z przeszukiwaniem według złotego podziału. Zauważmy, że $\varphi > 1{,}5$. W takim razie przeszukiwanie potrójne jest ponad dwa razy wolniejsze, ponieważ wymaga dwóch oszacowań wartości funkcji w każdym kroku. Na przykład w celu osiągnięcia skrócenie inicjalnego przedziału $1{,}5^{19} \approx 2217$ razy przeszukiwanie potrójne wymaga $19 \cdot 2 = 38$ oszacowań wartości funkcji. Z kolei przeszukiwanie według złotego podziału dla osiągnięcia skrócenia przedziału $\varphi^{16} \approx 2207$ razy wymaga 16 oszacowań.

Zauważmy, że metoda złotego podziału, podobnie jak metoda przeszukiwania potrójnego, ma tempo zbieżności ε-liniowe. Wynika to z faktu, że w każdej iteracji algorytmu przedział jest skracany dokładnie φ razy.

Przedstawione powyżej metody zakładały wyłącznie ciągłość minimalizowanej funkcji. Okazuje się, że w przypadku przyjęcia mocniejszych założeń możliwe jest dalsze przyspieszanie tempa zbieżności algorytmu. W tym miejscu omówimy metodę Brenta,

która jest zaimplementowana w procedurze `optimize` w pakiecie R. Metoda ta, w przypadku gdy minimalizowana funkcja ma ciągłą i dodatnią drugą pochodną w pewnym otoczeniu wokół wewnętrznego minimum lokalnego charakteryzuje się większym tempem zbieżności niż metoda złotego podziału. Wykorzystuje ona twierdzenie Taylora, na podstawie którego dwukrotnie różniczkowalną funkcję można lokalnie przybliżać wielomianem drugiego stopnia.

Aby wyjaśnić działanie metody Brenta rozważmy najpierw punkty $x_1, x_2, x_3 \in \mathbb{R}$ takie, że $f(x_1) \leq f(x_2) \leq f(x_3)$. Będziemy chcieli przez punkty $(x_i, f(x_i))$ przeprowadzić parabolę $a_1(x - x_1)^2 + a_2(x - x_1) + a_3$. Jeżeli $a_1 \neq 0$ to jej ekstremum jest odchylone o $\delta = -a_2/(2a_1)$ od punktu dotychczas będącego minimalnym (x_1).

Współczynniki a_i spełniają równanie:

$$\begin{bmatrix} 0 & 0 & 1 \\ (x_2 - x_1)^2 & x_2 - x_1 & 1 \\ (x_3 - x_1)^2 & x_3 - x_1 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \end{bmatrix}. \tag{3.5}$$

Dalej załóżmy, że punkty $(x_i, f(x_i)$ nie są współliniowe, ponieważ tylko wtedy ekstremum istnieje. Wtedy oznaczając $\Delta = (x_2 - x_1)(x_3 - x_1)(x_2 - x_3)$ rozwiązanie układu ma postać:

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} \Delta^{-1}\big((x_1 - x_3)(f(x_1) - f(x_2)) - (x_1 - x_2)(f(x_1) - f(x_3))\big) \\ \Delta^{-1}\big((x_1 - x_3)^2(f(x_1) - f(x_2)) - (x_1 - x_2)^2(f(x_1) - f(x_3))\big) \\ f(x_1) \end{bmatrix}. \tag{3.6}$$

Oznaczmy:

$$\begin{aligned} t_1 &= (x_1 - x_3)(f(x_1) - f(x_2)), \\ t_2 &= (x_1 - x_2)(f(x_1) - f(x_3)), \\ p &= (x_1 - x_3)t_1 - (x_1 - x_2)t_2, \\ q &= -2(t_1 - t_2). \end{aligned}$$

Przy tych oznaczeniach otrzymujemy $\delta = p/q$. Punktu eksremalnego δ nie da się wyznaczyć gdy $q = 0$, czyli we wspomnianym już przypadku, gdy trzy rozważane punkty są współliniowe.

Spostrzeżenia te są podstawą do działania metody Brenta. Procedura ta przedstawiona jest w kodzie 3.3.

Kod 3.3. Implementacja procedury przeszukiwania Brenta

```
brent <- function(f, lower, upper, tol) {
    ratio <- 2 / ((3 + sqrt(5)))
    d <- d2 <- 0
    x3 <- x2 <- x1 <- lower + ratio * (upper - lower)
    f.x3 <- f.x2 <- f.x1 <- f(x1)
    while ( abs(upper - lower) > 2 * tol) {
        xm <- (lower + upper) / 2
```

```
        t1 <- (x1 - x3) * (f.x1 - f.x2)
        t2 <- (x1 - x2) * (f.x1 - f.x3)
        p <- (x1 - x3) * t1 - (x1 - x2) * t2
        q <- 2 * (t1 - t2)
        if (q > 0) {
            p <- -p
        } else {
            q <- -q
        }
        if ((abs(d2) < tol) || (abs(p) >= abs(0.5 * q * d2)) ||
            (p <= q * (lower - x1)) || (p >= q * (upper - x1))) {
            d2 <- ifelse (x1 >= xm, lower, upper) - x1
            d <- ratio * d2
        } else {
            d2 <- d
            d <- p / q
            u <- x1 + d
            if (min(u - lower, upper - u) < 2 * tol) {
                d <- tol
                if (x1 >= xm) {
                    d <- -tol
                }
            }
        }
        u <- x1 + ifelse(d > 0, max(d, tol), min(d, -tol))
        f.u <- f(u)
        if (f.u <= f.x1) {
            if (u >= x1) {
                lower <- x1
            } else {
                upper <- x1
            }
            x3 <- x2
            f.x3 <- f.x2
            x2 <- x1
            f.x2 <- f.x1
            x1 <- u
            f.x1 <- f.u
        } else {
            if (u < x1) {
                lower <- u
            } else {
                upper <- u
            }
```

```
52              if ((f.u <= f.x2) || (x2 == x1)) {
53                  x3 <- x2
54                  f.x3 <- f.x2
55                  x2 <- u
56                  f.x2 <- f.u
57              } else if ((f.u <= f.x3) || (x3 == x1) || (x2 == x3)) {
58                  x3 <- u
59                  f.x3 <- f.u
60              }
61          }
62      }
63      return((upper + lower) / 2)
64  }
```

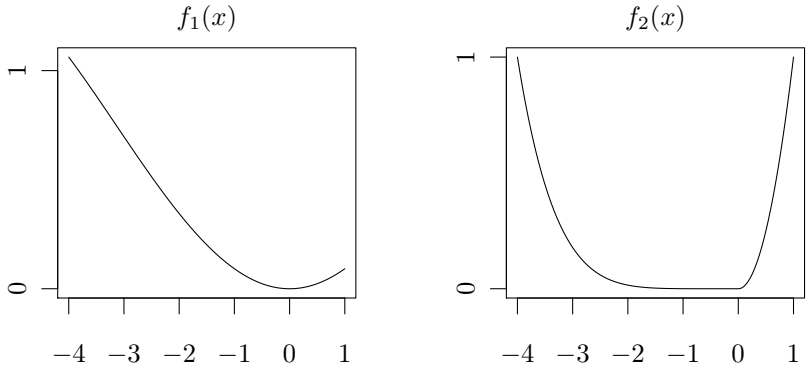
Podobnie jak wcześniej opisane metody dokonuje ona iteracyjnego zawężenia przedziału przeszukiwania o końcach `lower` i `upper`. Zapamiętuje ona trzy najlepsze znalezione wartości funkcji f w zmiennych `x1`, `x2`, `x3`. Na ich podstawie wyznaczane jest rekomendowane przesunięcie do kolejnego punktu próbnego o `d` zgodne z wyprowadzonym wzorem na wyliczenie wartość δ. Przesunięcie to nie jest akceptowane gdy występuje zagrożenie braku spełnienia założenia mówiącego, że druga pochodna jest ciągła i dodatnia. Brent zidentyfikował cztery warunki, w których może występować taka sytuacja: (i) rekomendowany punkt nie należy do przedziału [`lower`, `upper`], (ii) punkty `x1`, `x2` i `x3` są współliniowe, (iii) przesunięcie (δ) w ostatniej iteracji było bardzo małe lub (iv) aktualne przesunięcie jest większe niż połowa ostatniego przesunięcia. Jeżeli przynajmniej jeden z tych warunków jest spełniony to przesunięcie jest wyznaczane zgodnie z metodą złotego podziału. Dodatkowo, jeśli rekomendowany punkt znajduje się zbyt blisko krańca przedziału lub dotychczasowego optimum `x1` to jest odpowiednio od nich odsuwany w celu uniknięcia niepotrzebnego wykonywania pomiarów badanej funkcji w punktach za mało od siebie oddalonych. Następnie poprzez porównanie wartości funkcji w punktach `x1` i `x1+d` odpowiednio korygowane są krańce przedziału i lista trzech najlepszych znalezionych wartości badanej funkcji.

Podsumowując — algorytm Brenta poszukuje minimum funkcji f za pomocą aproksymacji parabolicznej. Jednak gdy stwierdzi, że nie zachowuje się ona zgodnie z przewidywaniami stosuje metodę złotego podziału. W ten sposób dla funkcji spełniających założenia metody zbieżność jest bardzo szybka, a jednocześnie algorytm działa prawidłowo również w przypadku funkcji mocno nieregularnych. W szczególności Brent (1973) dowodzi, że w przypadku gdy badana funkcja spełnia założenie o dodatniości i ciągłości swojej drugiej pochodnej w otoczeniu minimum to metoda ma zbieżność superliniową.

Zaprezentowana w kodzie 3.3 implementacja metody Brenta jest algorytmicznie zgodna z procedurą `optimize` z pakietu R. Jedyną różnicą jest określenie warunku zatrzymania obliczeń. W zaprezentowanym kodzie w procedurze podawany jest parametr `tol` i zwrócone rozwiązanie znajduje się w odległości nie większej niż ta wartość od minimum lokalnego, a więc jest to kryterium dokładności bezwzględnej. Kryte-

rium przerwania optymalizacji w procedurze `optimize` jest bardziej złożone. Również pobiera ona parametr `tol`, jednak gwarantowany błąd procedury zadany jest w przybliżeniu wzorem $2^{-26}x_0 + \texttt{tol}$, gdzie x_0 jest znalezionym przybliżeniem optimum. Kryterium to łączy więc warunek dokładności bezwzględnej i względnej.

Porównajmy działanie metod Brenta i złotego podziału na dwóch przykładowych funkcjach na przedziale $[-4, 1]$ i tolerancji 0,001. Pierwsza z nich zadana jest wzorem $f_1(x) = 3(1 - \cos(x/2))/4$, a druga ma postać $f_2(x) = [x < 0](x/4)^6 + [x \geq 0]x^2$. Obie funkcje maja jedyne minimum lokalne na badanym przedziale w punkcie 0. Zwróćmy jednak uwagę, że funkcja f_1 spełnia założenia metody Brenta o własnościach drugiej pochodnej w otoczeniu minimum. Z kolei mimo, że funkcja f_2 na badanym przedziale różniczkowalna w sposób ciągły, to niestety jej druga pochodna w 0 nie istnieje. Funkcje te zobrazowano na rysunku 3.1.



Badane są funkcje postaci: $f_1(x) = 3(1 - \cos(x/2))/4$ i $f_2(x) = [x < 0](x/4)^6 + [x \geq 0]x^2$.

Wykres 3.1: Wykresy funkcji f_1 i f_2 minimalizowanych metodami złotego podziału i Brenta

Kod 3.4 prezentuje wynik porównania zastosowania metod złotego podziału i Brenta do minimalizacji funkcji f_1 i f_2.

Kod 3.4. Porównanie liczby wywołań funkcji f_1 i f_2 przez metody złotego podziału i Brenta przy poszukiwaniu ich minimum na przedziale $[-4, 1]$ z bezwzględną dokładnością równą 0,001

```
f1 <- function(x) {
    calls <<- calls + 1
    3 * (1 - cos(x / 2)) / 4
}

f2 <- function(x) {
    calls <<- calls + 1
    ifelse(x < 0, x ^ 6 / 4 ^ 6, x ^ 2)
```

```
}

calls <- 0
golden(f1, -4, 1, 0.001) # -9.916044e-05
calls                    # 18
calls <- 0
brent(f1, -4, 1, 0.001)  # -0.0001221645
calls                    # 7

calls <- 0
golden(f2, -4, 1, 0.001) # -0.000633977
calls                    # 18
calls <- 0
brent(f2, -4, 1, 0.001)  # -0.0003152834
calls                    # 35
```

Zauważmy, że metoda złotego podziału — zgodnie z przewidywaniami — wykonuje tyle samo porównań niezależnie od postaci badanej funkcji. Z kolei algorytm Brenta okazuje się szybszy w przypadku funkcji f_1, ale jest wolniejszy dla funkcji f_2.

Omówione powyżej metody poszukują minimum lokalnego na zadanym przedziale. Minimum takie może znajdować się wewnątrz przedziału lub na jego brzegu. Częstą sytuacją praktyczną jest przypadek, w którym wiemy, że badana funkcja posiada minimum na zbiorze $\mathbb{R}$ i chcemy je znaleźć. Zły dobór przedziału początkowego w procedurze optymalizacyjnej może spowodować, że nie będzie on zawierał poszukiwanego punktu. W takich sytuacjach przed zastosowaniem omówionych wcześniej metod optymalizacji niezbędne jest wyznaczenie odpowiedniego przedziału startowego. Przykład procedury realizującej tego typu zadanie podany jest w kodzie 3.5. Funkcja `band.search` poszukuje przedziału zawierającego minimum dla funkcji unimodalnej w minimum.

Kod 3.5. Implementacja procedury wyszukiwania przedziału zawierającego minimum dla funkcji unimodalnej w minimum

```
band.search <- function(f, x0, delta) {
    f.x0 <- f(x0)
    f.pd <- f(x0 + delta)
    f.md <- f(x0 - delta)
    if (!all(sapply(c(f.x0, f.pd, f.md), is.finite))) {
        stop("Function value is not finite")
    }

    if (f.x0 <= min(f.pd, f.md)) {
        return(range(x0 + delta, x0 - delta))
    } else if (f.x0 > max(f.pd, f.md)) {
        stop("Function is not unimodal in minimum")
    } else if (f.pd > f.md) {
```

```
14          search <- -delta
15          f.x2 <- f.md
16      } else {
17          search <- delta
18          f.x2 <- f.pd
19      }
20      x1 <- x0
21      x2 <- x0 + search
22      repeat {
23          x3 <- x0 + 2 * search
24          if (!is.finite(x3)) {
25              stop("Proper␣interval␣not␣found")
26          }
27          f.x3 <- f(x3)
28          if (!is.finite(f.x3)) {
29              stop("Function␣value␣is␣not␣finite")
30          }
31          if (f.x3 > f.x2) {
32              return(range(x1, x3))
33          }
34          search <- 2 * search
35          x1 <- x2
36          x2 <- x3
37          f.x2 <- f.x3
38      }
39  }
```

Procedura rozpoczyna działanie od porównania wartości funkcji `f` w punktach `x0`, `x0 + delta` i `x0 - delta`. Możliwe są trzy przypadki. Jeżeli w punkcie `x0` wartość funkcji jest nie większa niż pozostałych dwóch punktach, to oznacza, że w przedziale od `x0 - delta` do `x0 + delta` musi znajdować się poszukiwane minimum. Z kolei, jeżeli wartość funkcji w punkcie `x0` jest większa niż w pozostałych dwóch punktach oznacza to, że badana funkcja nie jest unimodalna w minimum (ponieważ musi posiadać maksimum lokalne) i funkcja przerywa działanie. Ostatnim przypadkiem jest sytuacja, w której wartość funkcji w punkcie `x0` leży pomiędzy pozostałymi wartościami.

Ponieważ poszukujemy minimum więc w takim razie będziemy chcieli się poruszać w kierunku spadku wartości funkcji `f`. Kierunek ten jest zapisany do zmiennej `search`. Poszukiwana jest taka sekwencja punktów `x1`, `x2`, `x3`, żeby zachodziło `f(x1)>f(x2)` oraz `f(x2)<f(x3)`. Wtedy będziemy wiedzieli, że poszukiwany przedział ma końce w punktach `x1` i `x3`. Kolejnych kandydatów na punkt `x3` poszukujemy dwukrotnie wdłużając krok poszukiwania `search` w każdej iteracji. Jeżeli okazuje się, że nowo znaleziony punkt `x3` spełnia warunek `f(x2)<f(x3)` to procedura kończy działanie z sukcesem. Z kolei jeżeli go nie spełnia to aktualizowane są wartości zmiennych `x1` i `x2` i rozpoczyna się kolejna iteracja algorytmu.

Warto zwrócić uwagę na przypadek, w którym wartość `x3` w pewnym momencie przyjmuje wartość nieskończoną (`Inf` lub `-Inf`). Sytuacja taka zdaża się, gdy poszukiwania przekroczą dopuszczalny zakres liczb możliwych do zapisania w formacie *binary64*. Nie musi jednak oznaczać to, że badana funkcja nie jest unimodalna w minimum. Może tak się zdażyć wówczas gdy poszukiwane minimum znajduje się bardzo blisko granicy dopuszczalnego zakresu zmienności liczb zmiennoprzecinkowych i podczas jego *przeskakiwania* napotykamy nieskończoność[24]. Dodatkowo funkcja kontroluje, czy otrzymywane wartości funkcji są skończone, ponieważ w przeciwnym wypadku ich porównanie może prowadzić do nieprawidłowych wniosków.

W kodzie 3.6 przedstawiono możliwe przypadki działania funkcji `band.test`. Funkcja `f1` jest unimodalna w minimum. Przy inicjacji funkcji `band.test` parametrami `x0` równym 2 i `delta` równym 4 funkcja od razu znajduje odpowiedni przedział. Zmniejszenie parametru `delta` do 1 powoduje, że rozpoczyna się procedura przeszukiwania w kierunku ujemnym w stosunku do `x0` ponieważ w tym kierunku spadają wartości funkcji `f1`. Drugim przypadkiem jest funkcja `f2`, która jest unimodalna ale w maksimum. Dla parametrów `x0` równego 0 i `delta` równego 1 procedura stwierdza, że funkcja ta musi mieć maksimum lokalne i przerywa działanie. Zmiana wartości `x0` na 2 powoduje, że rozpoczyna się przeszukiwanie przestrzeni argumentów funkcji w kierunku dodatnim. Jednak w pewnym momencie — ze względu na ograniczenia numeryczne— wartość argumentu funkcji staje się na tyle duża, że wyliczona wartość funkcji jest nieskończona i procedura `band.test` przerywa działanie. Ostatnim przypadkiem jest przypadek funkcji `f3`. W tym wypadku mamy do czynienia z funkcją rosnącą. Więc startując z punktu `x0` równego 2 przeszukiwany jest kierunek ujemny. Niestety ponieważ funkcja nie posiada minimum lokalnego w pewnym momencie wartość `x0 + 2 * search` przyjmuje wartość `-Inf` i również procedura przerywa działanie. Zwróćmy uwagę, że w tym wypadku wartość funkcji jest przez R wyliczana bez informacji o błędzie, ponieważ wyrażenie `atan(-Inf)` przyjmuje wartość równą w przybliżeniu $-1{,}570796$.

Kod 3.6. Możliwe wyniki przy zastosowaniu funkcji `band.search`

```
f1 <- function(x) {
    x ^ 2
}

f2 <- function(x) {
    -x ^ 2
}

f3 <- function(x) {
    atan(x)
}
```

[24] Należy jednak podkreślić, że w takich przypadkach najprawdopodobniej obliczenia wartości funkcji `f` mogą jednocześnie napotykać na duże problemy związane z zaokrąglaniem i taka sytuacja wymaga szczegółowego zbadania pod kątem numerycznym.

```
12
13 band.search(f1, 2, 4) # -2  6
14 band.search(f1, 2, 1) # -2  1
15 band.search(f2, 0, 1) # Function is not unimodal in minimum
16 band.search(f2, 2, 1) # Function value is not finite
17 band.search(f3, 2, 1) # Proper interval not found
```

Powyżej przedstawione metody wykorzystywały jedynie wartości funkcji do poszukiwania jej minimum lokalnego. Przejdźmy teraz do omówienia tzw. algorytmów gradientowych, które wykorzystują informację o pochodnych minimalizowanej funkcji f. W oczywisty sposób — ograniczeniem tych metod jest konieczność spełnienia warunku różniczkowalności przez badaną funkcję.

Najprostszym algorytmem gradientowym jest metoda bisekcji. Poszukuje ona minimum lokalnego funkcji f na zadanym przedziale $[x_l, x_u]$. Rozważmy punkt $m = (x_l + x_u)/2$. Możliwe są trzy przypadki wartości pochodnej $f'(m)$. Jeżeli $f'(m) < 0$ oznacza to, że w przedziale $[m, x_u]$ musi się znajdować przynajmniej jedno minimum lokalne funkcji f. Jeżeli $f'(m) > 0$, to analogicznie możemy ograniczyć przedział poszukiwań do $[x_l, m]$. Najbardziej subtelna jest sytuacja gdy $f'(m) = 0$. Jeżeli wiemy, że badana funkcja jest wypukła to możemy zatrzymać wykonanie algorytmu i stwierdzić, że w m znajduje się minimum lokalne funkcji f. Jednak w ogólności warunek ten nie gwarantuje występowania minimum, por. np. funkcję x^3 w punkcie 0. W takiej sytuacji możliwe są różne metody praktycznego radzenia sobie z taką sytuacją. Najprostszą z nich jest niewielkie zaburzenie wartości m i próba kontynuacji obliczeń z wykorzystaniem tak skorygowanego punktu. Implementację tego algorytmu przedstawiono w kodzie 3.7.

Kod 3.7. Implementacja algorytmu bisekcji

```
1  bisection <- function(df, lower, upper, tol) {
2      while (upper - lower > 2 * tol) {
3          m <- (lower + upper) / 2
4          df.m <- df(m)
5          while (df.m == 0) {
6              m <- (lower + upper) / 2 + runif(1, -tol, tol)
7              df.m <- df(m)
8          }
9          if (df.m < 0) {
10             lower <- m
11         } else {
12             upper <- m
13         }
14     }
15     return((upper + lower) / 2)
16 }
```

Warto zwrócić uwagę na fakt, że algorytm ten w każdym kroku dwukrotnie zmniejsza długość przeszukiwanego przedziału. W związku z tym ma on ε-liniowe tempo zbieżności i wydaje się być szybszy niż algorytm złotego podziału. Jednak jego praktyczna szybkość zależy od kosztu wyznaczenia wartości pochodnej $f'(x)$ w punkcie w stosunku do kosztu wyliczenia wartości $f(x)$. Jeśli jest on porównywalny, to metoda bisekcji będzie bardziej efektywna. Z kolei jeśli np. pochodna jest wyznaczana numerycznie, co wymaga przynajmniej dwóch oszacowań funkcji f to metoda złotego podziału jest bardziej efektywna.

Zauważmy, że algorytm bisekcji w rzeczywistości poszukuje miejsca zerowego pochodnej badanej funkcji. Spostrzeżenie to pokazuje, że możemy go wykorzystać do poszukiwania miejsc zerowych dowolnej funkcji. Jest to problem również często występujący w praktyce. W R dostępna jest procedura `unitroot`, za pomocą której można poszukiwać miejsc zerowych funkcji jednej zmiennej. Jest ona rozszerzeniem metody bisekcji wykorzystującym lokalną liniową lub kwadratową aproksymację badanej funkcji. Jest to rozszerzenie analogiczne z jakim mieliśmy do czynienia w przypadku metody Brenta, która udoskonalała metodę złotego podziału.

W przypadku gdy dysponujemy informacją o pierwszej i drugiej pochodnej minimalizowanej funkcji f możliwe jest zastosowanie procedury jeszczce efektywniejszej niż metoda bisekcji. Mianowicie biorąc pod uwagę punkt $x(k)$ rozważany w k-tym kroku procedury optymalizacyjnej możemy funkcję f aproksymować stosując wzór Taylora:

$$\hat{f}(x(k)+h) = f(x(k)) + f'(x(k))h + \frac{1}{2}f''(x(k))h^2. \tag{3.7}$$

Następnie w celu przybliżenia minimum funkcji f znajdujemy minimum funkcji $\hat{f}$ i otrzymjemy kolejny punkt zadany wzorem

$$x(k+1) = x(k) - f'(x(k))/f''(x(k)). \tag{3.8}$$

Procedura ta nazywana jest algorytmem Newtona. Przykład jej implementacji przedstawiony jest w kodzie 3.8. Zwróćmy uwagę na fakt, że w tym wypadku warunek zatrzymania procedury nie gwarantuje dokładności przybliżenia rozwiązania optymalnego. Mówi on jedynie, że korekta otrzymanego rozwiązania w stosunku do poprzedniego jest mała.

Kod 3.8. Implementacja algorytmu Newtona i przykład jego zastosowania

```
newton <- function(df, d2f, x, tol) {
    repeat {
        new.x <- x - df(x) / d2f(x)
        if (abs(new.x - x) < tol) {
            return(new.x)
        }
        x <- new.x
    }
}
```

```
10
11 f <- function(x) {
12     - exp(-x ^ 2)
13 }
14 df <- function(x) {
15     2 * x * exp(-x ^ 2)
16 }
17 d2f <- function(x) {
18     2 * exp(-x ^ 2) - 4 * x ^ 2 * exp(-x ^ 2)
19 }
20 newton(df, d2f, 1, 0.001)   # error
21 newton(df, d2f, 0.5, 0.001) # infinite loop
22 newton(df, d2f, 0.4, 0.001) # converges
```

Warto zwrócić uwagę na fakt, że metoda ta — podobnie jak algorytm bisekcji — nie korzysta z informacji o wartości badanej funkcji. Punkt, którego poszukujemy to miejsce zerowe pochodnej funkcji f. W szczególności oznacza to, że zwrócona aproksymacja może przybliżać maksimum (jeśli druga pochodna jest ujemna) lub punkt przegięcia (druga pochodna równa 0) badanej funkcji. Z kolei zaletą algorytmu Newtona jest jego szybkość zbieżności. Jeżeli funkcja $f''(x)$ w okolicy minimum jest dodatnia oraz spełnia warunek Lipshitza, a dodatkowo punkt startowy optymalizacji jest położony w stosunku do niego dostatecznie blisko to tempo jego zbieżności jest kwadratowe (Nocedal i Wright, 1999).

Podajmy tutaj dowód tej zależności w nieco słabszej postaci. Niech x^* będzie minimum funckji f. Załóżmy, że funkcja ta w pewnym otoczeniu $(x^* - \varepsilon, x^* + \varepsilon)$ punktu x^* jest trzykrotnie różniczkowalna w sposób ciągły oraz w tym otoczeniu $f''(x) > L$, gdzie ε oraz L są stałymi dodatnimi. Oznaczmy $x_i = x^* + h_i$. Przyjmijmy, że dla pewnego k zachodzi $|h_k| < \varepsilon$, czyli że punkt $x(k)$ leży *blisko* minimum. Przy tych oznaczeniach równanie (3.8) możemy przekształcić do postaci:

$$x^* + h_{k+1} = x^* + h_k - f'(x^* + h_k)/f''(x^* + h_k). \tag{3.9}$$

Przekształcając ten wzór otrzymujemy równość:

$$\frac{h_{k+1}}{h_k^2} = \frac{f'(x + h_k) - f''(x + h_k)h_k}{f''(x + h_k)h_k^2}. \tag{3.10}$$

W takim razie aby dowieść kwadratowego tempa zbieżności metody Newtona wystarczy pokazać, że wyrażenie po prawej stronie równania (3.10) jest ograniczone. Zauważmy, że z założenia o trzykrotnej różniczkowalności funkcji f w otoczeniu x^* wynika, że: $f'(x^*) = f'(x^* + h_k) - f''(x^* + h_k)h_k + f'''(x^* + h_k)h_k^2 + \mathcal{O}(h_k^2)$. Ponieważ wiemy, że $f'(x^*) = 0$, więc $f'(x^* + h_k) = f''(x^* + h_k)h_k - f'''(x^* + h_k)h_k^2 + \mathcal{O}(h_k^2)$. Podstawiając to do równania (3.10) otrzymujemy:

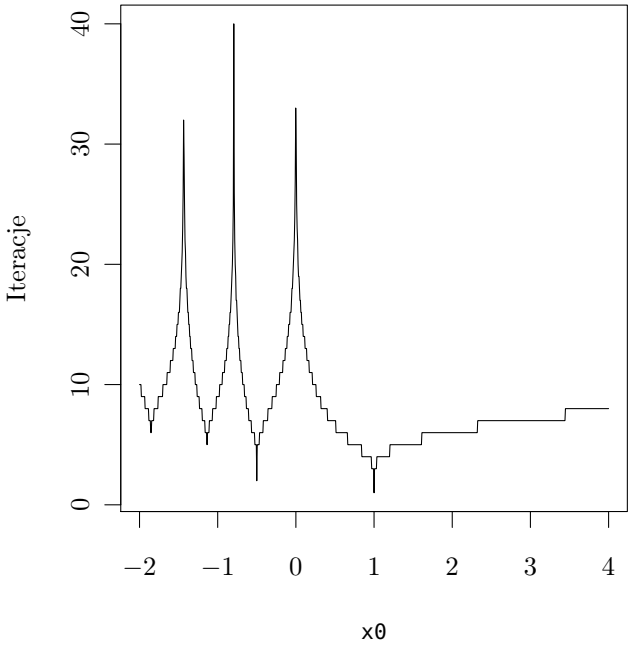
$$\frac{h_{k+1}}{h_k^2} = \frac{-f'''(x^* + h_k)h_k^2 + \mathcal{O}(h_k^2)}{f''(x + h_k)h_k^2} = \frac{-f'''(x^* + h_k)}{f''(x + h_k)} + \mathcal{O}(1). \tag{3.11}$$

Ponieważ $|h_k| < \varepsilon$ oraz z założenia funkcja f''' jest ciągła w przedziale $(x^* - \varepsilon, x^* - \varepsilon)$ to znaczy, że istnieje takie $L' > 0$, że $|f'''(x+t)| < L'$ dla $t \leq |h_k|$. W takim razie otrzymujemy:

$$\left| \frac{h_{k+1}}{h_k^2} \right| < \left| \frac{L'}{L} + \mathcal{O}(1) \right|. \tag{3.12}$$

To oznacza, że wyrażenie h_{k+1}/h_k^2 jest w pewnym otoczeniu x^* ograniczone, czyli metoda Newtona przy przyjętych założeniach zbiega w tempie kwadratowym, jeżeli przyjęty punkt startowy jest dostatecznie bliski minimum.

Przyspieszona zbieżność metody Newtona okupiona jest jej wrażliwością na kształt optymalizowanej funkcji, co zobrazowane jest w kodzie 3.8. Optymalizowana jest funkcja $-e^{-x^2}$. Posiada ona jedno minimum w 0. Niestety tylko w lokalnym otoczeniu tego punktu procedura Newtona jest do niego zbieżna. Punktami granicznymi są $-0{,}5$ i $0{,}5$. Jeśli punkt startowy jest co do wartości bezwzględnej mniejszy niż $0{,}5$ to procedura jest zbieżna. Dla punktów granicznych obliczenia wpadają w cykl. Dla pozostałych punktów startowych procedura rozbiega do nieskończoności.



Minimalizowana jest funkcja $x^4 - 4x$ w zależności od punktu startowego `x0`. Przyjęta wartość parametru `tol` to 10^{-6}.

Wykres 3.2: Zależność liczby iteracji algorytmu Newtona od punktu startowego

Niestabilność zachowania algorytmu Newtona zobrazowano również na rysunku

3.2. Minimalizowana jest funkcja $x^4 - 4x$, która ma minimum w punkcie 1. Zauważmy, że dla wartości punktu startowego większych niż 1 liczba wymaganych iteracji rośnie bardzo wolno wraz z oddalaniem się od minimum. Dla punktów startowych mniejszych od minimum sytuacja wyląda inaczej. Ponieważ $f''(0)$ jest równe 0 więc dla tego punktu startowego procedura kończy się błędem. Podobnie jest dla punktu $-1/\sqrt[3]{2}$. Ponieważ w kolejnym kroku algorytmu trafi on w 0. Rozszerzając indukcyjnie to rozumowanie otrzymamy nieskończenie wiele punktów, w których procedura nie jest zbieżna. Dodatkowo — jak pokazano na rysunku 3.2 w pobliżu punktów, w których następuje rozbieżność algorytmu szybkość jego zbieżności znacznie spada.

Mimo pokazanej niestabilności algorytmu Newtona jest on ważną teoretycznie metodą optymalizacji. W szczególności okazuje się, że można go w sposób naturalny uogólnić na przypadek wielowymiarowej dziedziny optymalizowanej funkcji i zmodyfikować w taki sposób, który podnosi jego stabilność. Zagadnienia te przedstawione są w rozdziale 4. W dalszej części tego podrozdziału zaprezentujemy dwa podstawowe elementy koncepcyjne wykorzystywane w tych uogólnieniach. Pierwszym z nich jest tzw. metoda siecznych pozwalająca na uniknięcie konieczności wyznaczania drugiej pochodnej funkcji. Drugim uogólnieniem jest zapewnienie, że w wyniku wykonania pojedynczego kroku algorytmu wartość funkcji celu spada. Jest ono zaprezentowane przy wykorzystaniu tzw. reguły Armijo.

Metoda siecznej ma na celu wyeliminowanie w równaniu $x(k+1) = x(k) - f'(x(k))/f''(x(k))$ konieczności wyznaczania wartości $f''(x(k))$. Jeśli założymy, że został wykonany już przynajmniej jeden krok algorytmu to dysponujemy dwoma punktami $x(k)$ oraz $x(k-1)$ oraz wartościami pochodnej minimalizowanej funkcji w tych punktach. W takim razie $f''(x(k))$ można przybliżyć za pomocą współczynnika kierunkowego siecznej przechodzącej przez punkty $(x(k-1), f'(x(k-1)))$ oraz $(x(k), f'(x(k)))$.

Mamy zatem $f''(x(k)) \approx (f'(x(k)) - f'(x(k-1)))/(x(k) - x(k-1))$ i oryginalny wzór w algorytmie Newtona ulega modyfikacji do postaci:

$$x(k+1) = x(k) - f'(x(k)) \frac{x(k) - x(k-1)}{f'(x(k)) - f'(x(k-1))}. \tag{3.13}$$

Zwróćmy uwagę, że krok ten odpowiada minimalizacji funkcji kwadratowej postaci:

$$\tilde{f}(x(k) + h) = f(x(k)) + f'(x(k))h + \frac{1}{2} \frac{f'(x(k)) - f'(x(k-1))}{x(k) - x(k-1)} h^2. \tag{3.14}$$

Funkcja ta ma pochodne w punktach $x(k)$ i $x(k-1)$ równe dokładnie $f'(x(k))$ oraz $f'(x(k-1))$. Dodatkowo zauważmy, że funkcja $\tilde{f}$ jest jedyną funkcją kwadratową, która spełnia ten warunek.

Kod 3.9 prezentuje przykładową implementację metody siecznych. Zwróćmy uwagę, że wykorzystuje on tylko pierwszą pochodną funkcji w procesie optymalizacji. Z drugiej strony w celu rozpoczęcia optymalizacji niezbędne jest podanie dwóch punktów startowych. Zwróćmy uwagę, że metoda siecznych charakteryzuje się podobnymi problemami ze zbieżnością jak metoda Newtona.

Kod 3.9. Implementacja metody siecznych

```r
secant <- function(df, x1, x2, tol) {
    df.x2 <- df(x2)
    repeat {
        df.x1 <- df(x1)
        new.x <- x1 - df.x1 * (x1 - x2) / (df.x1 - df.x2)
        if (abs(new.x - x1) < tol) {
            return(new.x)
        }
        x2 <- x1
        df.x2 <- df.x1
        x1 <- new.x
    }
}

f <- function(x) {
    - exp(-x ^ 2)
}
df <- function(x) {
    2 * x * exp(-x ^ 2)
}
secant(df, 1, 1.1, 0.001)   # diverges
secant(df, 0.5, 0.6, 0.001) # converges
```

Algorytm Newtona stara się w maksymalnie szybki sposób przejść do optimum badanej funkcji, co powoduje jego niestabilność. Przejdźmy teraz do zasady wyznaczania długości kroku optymalizacji, która jest nazywana regułą Armijo. Podobnie jak metoda Newtona korzysta ona z informacji o pochodnej minimalizowanej funkcji, jednak jej zaletą jest to, że gwarantuje zmniejszenie wartości funkcji celu w wyniku wykonania kroku optymalizacji.

Rozważmy różniczkowalną w sposób ciągły funkcję $f\colon \mathbb{R} \to \mathbb{R}$ w punkcie x_0 takim, że $f'(x_0) \neq 0$. Dla ustalenia uwagi przyjmijmy $f'(x_0) < 0$. Niech $\alpha \in (0, 1)$. Zauważmy, że zachodzi tzw. warunek Armijo:

$$\exists d\colon f(x_0 + d) \leq f(x_0) + d\alpha f'(x_0). \tag{3.15}$$

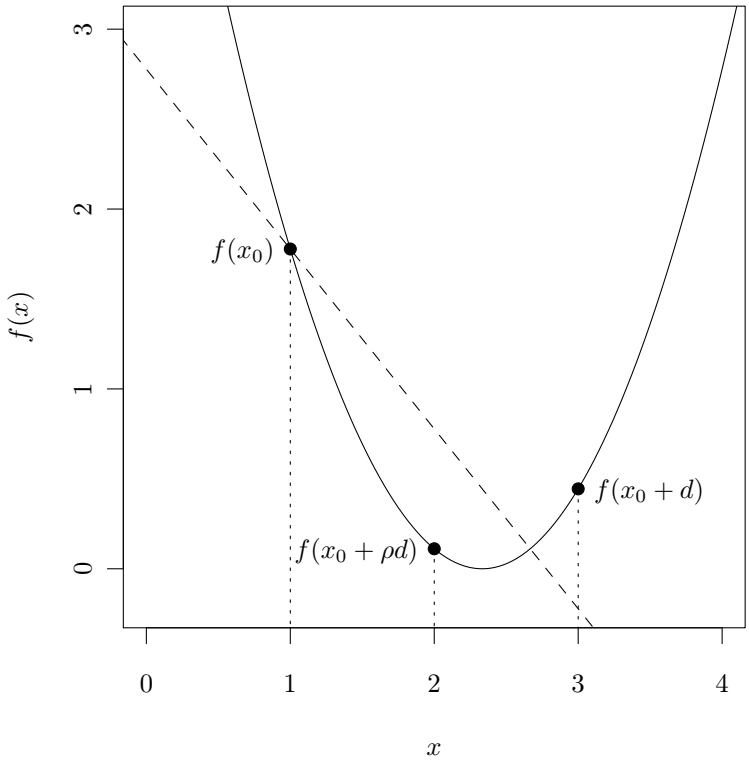
Aby to wykazać obliczmy korzystając z wzoru Taylora:

$$f(x_0+d) = f(x_0)+df'(x_0)+\mathcal{O}(d) = f(x_0)+d\alpha f'(x_0)+d(1-\alpha)(f'(x_0)+\mathcal{O}(1)). \tag{3.16}$$

Jeśli więc będziemy rozpatrywali takie d, że $df'(x_0) > 0$ to dla dostatecznie małych jego wartości wyrażenie $d(1 - \alpha)(f'(x_0) + \mathcal{O}(1))$ również będzie dodatnie. W takim razie zachodzi warunek (3.15).

W praktyce startujemy od wybranej wielkości kroku d i dokonujemy sprawdzenia, czy warunek (3.15) dla niego zachodzi, a jeśli nie to zmniejszamy wartość kroku do ρd, gdzie $\rho \in (0, 1)$. Taką procedurę wykonujemy iteracyjnie, aż warunek zostanie

spełniony (3.15). Jak pokazaliśmy procedura zawsze się zatrzyma dla dostatecznie małej wartości kroku (pomijając problemy z dokładnością obliczeń numerycznych). Zasada ta zobrazowana jest na rysunku 3.3. Wartość $f(x_0+d)$ jest większa niż $f(x_0)+d\alpha f'(x_0)$, więc przeszukiwanie jest kontynuowane. W kolejnym kroku wartość $f(x_0+\rho d)$ spełnia regułę Armijo i algorytm kończy działanie.



Reguła Armijo dla $f(x) = (x - 7/3)^2$, $x_0 = 1$, $d = 2$, $\alpha = 0{,}375$ oraz $\rho = 0{,}5$. Linią ciągłą oznaczono wykres funkcji $f(x)$, a linią przerywaną prostą przechodzącą przez punkt $(x_0, f(x_0))$ o współczynniku kierunkowym $\alpha f'(x_0)$.

Wykres 3.3: Ilustracja reguł Armijo opisanej warunkiem (3.15)

Implementacja reguły Armijo przedstawiona jest w kodzie 3.10. Zauważmy, że w przypadku praktycznym pętla `while` może zakończyć wykonanie dopiero gdy `d` będzie tak małe, że `x` i `x + d` będą sobie równe ze względu na zaokrąglenia numeryczne.

Kod 3.10. Implementacja reguły Armijo

```
armijo.rule <- function(f, df, x, alpha, rho, d) {
    d <- -sign(df(x)) * abs(d)
    while(f(x + d) > f(x) + alpha * d * df(x)) {
```

```
        d <- rho * d
    }
    return(x + d)
}
```

Kod 3.11 prezentuje prostą modyfikację algorytmu Newtona wykorzystującą regułę Armijio. Modyfikacja wprowadza dwie zmiany. Pierwsza dotyczy długości startowego kroku procedury, a druga zasad akceptacji znalezionego punktu. Mianowicie jeżeli druga pochodna minimalizowanej funkcji jest dodatnia, to startowa wartość zmiennej d z równania (3.15) wyznaczana jest zgodnie z algorytmem Newtona. Jednak gdy nie jest ona dodatnia przyjmujemy długość kroku równą 1 w kierunku spadku wartości badanej funkcji. Druga zmiana związana jest z samą regułą Armijo. Procedura `newton.armijo` w przeciwieństwie do procedury `newton` wykorzystuje informację o wartościach badanej funkcji. W związku z w każdym punkcie będącym kandydatem na rozwiązanie optymalne badana jest wartość funkcji celu i jest on przyjmowany jedynie gdy spełnia warunek zadany regułą Armijo.

Przykład zastosowania procedury `newton.armijo` w kodzie 3.11 jest taki sam jak w kodzie 3.8. W tym przypadku za każdym razem procedura zbiega do poszukiwanego minimum.

Kod 3.11. Implementacja procedury optymalizacyjnej wykorzystującej regułę Armijo i przykład jej zastosowania

```
newton.armijo <- function(f, df, d2f, x, alpha, rho, tol) {
    f.x <- f(x)
    repeat {
        df.x <- df(x)
        d2f.x <- d2f(x)
        d <- -sign(df.x)
        if (d2f(x) > 0) {
            d <- - df(x) / d2f(x)
        }
        repeat {
            new.x <- x + d
            f.new.x <- f(new.x)
            if (f.new.x <= f(x) + alpha * d * df.x) {
                break
            }
            d <- rho * d
        }
        if (abs(new.x - x) < tol) {
            return(new.x)
        }
        x <- new.x
        f.x <- f.new.x
    }
```

```
}


f <- function(x) {
    - exp(-x ^ 2)
}
df <- function(x) {
    2 * x * exp(-x ^ 2)
}
d2f <- function(x) {
    2 * exp(-x ^ 2) - 4 * x ^ 2 * exp(-x ^ 2)
}
newton.armijo(f, df, d2f, 1, 0.1, 0.5, 0.001)
newton.armijo(f, df, d2f, 0.5, 0.1, 0.5, 0.001)
newton.armijo(f, df, d2f, 0.4, 0.1, 0.5, 0.001)
```

Warto podkreślić, że reguła Armijo jest wykorzystywana jako komponent w omówionych w rodziale 4 algorytmach minimalizujących funkcje wielu zmiennych. W szczególności w R procedura `optim` stosuje ją w algorytmach gradientu sprzężonego i BFGS.

Przykłady zastosowań

Niniejszy podrozdział zawiera sześć przykładów zastosowań omawianych metod. Rozpoczyna się od przedstawienia sposobu wyznaczenia prostego estymatora metodą największej wiarygodności. Kolejne dwa przykłady dotyczą zagadnień matematyki finansowej: wyznaczenia wewnętrznej stopy zwrotu inwestycji oraz wyznaczania wartości równej raty kredytu. Następne dwa przykłady poruszają zagadnienia ekonomii matematycznej: wyznaczanie optymalnej ceny przez monopolistę oraz poszukiwanie równowagi w duopolu Bertranda z dobrami nie będącymi doskonałymi substytutami. Ostatni przykład zaczerpnięty jest z mikroekonometrii i prezentuje problem ustalenia optymalnego progu odcięcia w modelach klasyfikacyjnych.

Przykład 3.1. Dysponujemy n elementową próbą x_i pochodzącą z rozkładu wykładniczego zadanego funkcją gęstości $f(x) = \lambda e^{-\lambda x}$ dla $x \geq 0$ i $\lambda > 0$. Mamy za zadanie oszacować parametr λ metodą największej wiarygodności.

Tak zadany problem daje się rozwiązać analitycznie. Logarytm funkcji wiarygodności ma postać $\mathcal{L}(\lambda) = n \ln(\lambda) - \lambda \sum_{i=1}^{n} x_i$. Obliczając pochodną $\mathcal{L}$ po λ otrzymujemy $\mathcal{L}'(\lambda) = n/\lambda - \sum_{i=1}^{n} x_i$ oraz $\mathcal{L}''(\lambda) = -n/\lambda^2$. W związku z tym w punkcie $\lambda = n/\sum_{i=1}^{n} x_i$ badana funkcja wiarygodności osiąga maksimum.

W niniejszym przykładzie porównamy wartość otrzymanego rozwiązania analitycznego z wynikiem optymalizacji numerycznej. Procedura wykonująca to zadanie przedstawiona jest w kodzie 3.12.

Kod 3.12. Minimalizacja logarytmu wiarygodności z wykorzystaniem procedury `optimize`

```
1  llik <- function(a) {
2      length(x) * log(a) - a * sum(x)
3  }
4
5  set.seed(1)
6  x <- rexp(100, 0.1)
7
8  a.x <- 1 / mean(x)
9  # 0.09702366
10 a.o <- optimize(llik, range(1 / x), maximum = T)$maximum
11 # 0.09703363
12
13 .Machine$double.eps * a.o + (.Machine$double.eps) ^ 0.25
14 # 0.0001220703
15 a.x - a.o
16 # -9.966001e-06
```

Procedura `optimize` wykorzystuje algorytm Brenta. Ponieważ logarytm wiarygodności powinien być maksymalizowany więc, wybrano opcję `maximum = T`. W algorytmie tym kluczowy jest prawidłowy wybór startowego przedziału startowego. W tym przypadku wiemy, że optimum badanej funkcji musi się znajdować w przedziale $[1/\max\{x_i\}, 1/\min\{x_i\}]$. Otrzymany wynik optymalizacji dobrze przybliża rozwiązanie analityczne. W linijkach 13 i 15 sprawdzamy, że ich różnica jest co do wartości bezwzględnej mniejsza niż gwarantowana dokładność przybliżenia rozwiązania optymalnego. □

Przykład 3.2. Rozważmy inwestycję, która w pierwszym roku wymaga poniesienia nakładu początkowego w wysokości 100 jednostek, a następnie przez sześć kolejnych lat generuje 20 jednostek gotówki dla inwestora. Należy wyznaczyć wewnętrzną stopę zwrotu inwestycji. Kod 3.13 prezentuje implementację procedury wyznaczającej poszukiwaną wartość.

Kod 3.13. Wyznaczanie wewnętrznej stopy zwrotu inwestycji

```
1  flows <- c(-100, 20, 20, 20, 20, 20, 20)
2  npv <- function(r) {
3      sum(flows / (1 + r) ^ (seq_along(flows)-1))
4  }
5
6  npv(0)                       # 20
7  npv(0.1)                     # -12.89479
8  uniroot(npv, c(0, 0.1))$root # 0.05471951
```

Poszukujemy takiej stopy dyskonta `r` przy której wartość bieżąca netto inwestycji wynosi 0. W związku z tym wykorzystujemy funkcję `uniroot` w celu znalezienia miejsca zerowego funkcji `npv`. Ponieważ funkcja `uniroot` wymaga podania przedziału w którym poszukiwane jest zero w linijkach 6 i 7 sprawdzamy, że przedział $[0,0{,}1]$ zawiera poszukiwane miejsce zerowe. □

Przykład 3.3. Rozważmy bank, który udziela kredytu w wysokości 100 jednostek, który jest spłacany przez kredytobiorcę w ratach rocznych przez pięć okresów. Przyjmując, że oprocentowanie kredytu wynosi 10% należy wyznaczyć schemat spłaty kredytu przy założeniu że spłacane raty powinny być równe. Kod 3.14 prezentuje sposób wyznaczenia poszukiwanej wartości.

Kod 3.14. Określenie wielkości równej raty kredytu

```
1  r <- 0.1
2  s <- 100
3  t <- 5
4  final <- function(p) {
5      balance <- 100
6      for (i in 1:t) {
7          balance <- balance * (1 + r) - p
8      }
9      return(balance)
10 }
11
12 uniroot(final, c(20, 110))$root # 26.37975
```

Procedurze `final` podaje się argument `p` będący założoną stałą ratą kredytu, a zwraca bilans rozliczeń klienta z bankiem po pięciu latach spłacania kredytu. Podobnie jak w przykładzie 3.2 właściwą ratę kredytu znajdujemy za pomocą funkcji `uniroot`. Zwróćmy uwagę, że jako startowy przedział wybrano $[20, 100]$ ponieważ wiemy, że rata musi być większa niż 20 (taka rata obowiązywałaby przy zerowym oprocentowaniu) i mniejsza niż 110 (taka rata prowadzi do spłacenia kredytu już w pierwszym roku). Otrzymana w wyniku działania procedury wartość jest bardzo dokładnie przybliża wynik obliczeń otrzymanych na podstawie wzoru jawnego na równą ratę kredytu: $100 \cdot 1{,}1^5 \cdot 0{,}1/(1{,}1^5 - 1)$. □

Przykład 3.4. Rozważmy problem monopolisty, który stara się wyznaczyć cenę p maksymalizującą jego zyski. Monopolista wie, że popyt jest związany z ceną zależnością $q(p) = e^{-p}$ dla $p \geq 0$. Z drugiej strony jednostkowe koszty zmienne monopolisty rosną wraz ze wzrostem popytu i wynoszą $c(q) = 1 + q$. W związku z tym funkcję zysku monopolisty można zapisać równaniem:

$$\pi(p) = e^{-p}\left(p - (1 + e^{-p})\right). \tag{3.17}$$

Chcąc rozwiązać postawione zadanie wyliczamy pierwszą i drugą pochodną funkcji zysku $\pi'(p) = -pe^{-p} + 2e^{-p} + 2e^{-2p}$ oraz $\pi''(p) = pe^{-p} - 3e^{-p} - 4e^{-2p}$. Warunek

konieczny istnienia optimum to $\pi'(p) = 0$ i po przekształceniu przyjmuje on postać $2 + 2e^{-p} = p$. Niestety tego równania nie da się rozwiązać analitycznie. W związku z tym skorzystamy z procedury `newton.armijo` w celu znalezienia optymalnej ceny. Kod 3.15 prezentuje wyniki optymalizacji.

Kod 3.15. Wyznaczenie optymalnej ceny monopolisty z wykorzystaniem procedury `newton.armijo`

```
1  nprofit <- function(p) {
2      -exp(-p) * (p - (1 + exp(-p)))
3  }
4
5  dnprofit <- function(p) {
6      p * exp(-p) - 2 * exp(-p) - 2 * exp(-2 * p)
7  }
8
9  d2nprofit <- function(p) {
10     -p * exp(-p) + 3 * exp(-p) + 4 * exp(-2 * p)
11 }
12
13 opt <- newton.armijo(nprofit, dnprofit, d2nprofit,
14                       5, 0.001, 0.5, 1e-4)
15 # 2.217715
16
17 dnprofit(opt+0.5e-7) # 5.369694e-09
18 dnprofit(opt-0.5e-7) # -7.886055e-09
19 d2nprofit(opt)       # 0.1325575
```

Ponieważ funkcja π jest maksymalizowana to procedura `nprofit` oraz wzory na pochodne odnoszą się do funkcji $-\pi$, która jest następnie minimalizowana. Dodatkowo wierszach 17 i 18 procedury stwierdzamy, że pochodna w tych punktach ma różny znak. Oznacza to, że wyznaczony punkt jest oddalone od minimum funkcji $-\pi$ o nie więcej niż 10^{-7}. W wierszu 19 sprawdzamy, że druga pochodna w wyznaczonym punkcie jest dodatnia, co oznacza, że przybliżamy minimum funkcji $-\pi$.

Warto podkreślić, że oba te sprawdzenia są istotne. Bez informacji o znaku drugiej pochodnej występowało ryzyko, że otrzymany punkt jest punktem przegięcia lub maksimum badanej funkcji. Z kolei bez badania zakresu zmiany znaku pierwszej pochodnej nie znalibyśmy oszacowania dokładności otrzymanego przybliżenia. Parametr `tol` procedury `newton.armijo` kontroluje *zmianę* położenia punktu kandydata do optimum. □

Przykład 3.5. Rozważmy model duopolu Bertranda, w którym dobra sprzedawane przez obie firmy nie są doskonałymi substytutami. Oznaczmy przez p_1 i p_2 odpowiednio oferowane ceny. Zakładamy, że zmienny koszt jednostkowy produkcji dobra przez każdą z firm wynosi 1. Z kolei całkowity popyt na rynku wynosi 1 i jest nieelastyczny

ze względu na ceny. Dzieli się on pomiędzy firmy zgodnie z tzw. regułą softmax. Sprzedaż firmy i wynosi $q_i(p_1, p_2) = e^{-p_i}/(e^{-p_1} + e^{-p_2})$. Będziemy się starali wyznaczyć ceny równowagowe na badanym rynku.

Oznaczmy odpowiednio przez $r_1(p_2)$ oraz $r_2(p_1)$ funkcje reakcji firm 1 i 2 na cenę konkurenta. Funkcja reakcji mówi jaka cena maksymalizuje zyski firmy przy ustalonej cenie konkurenta. W związku z tym równowaga Nasha w tej grze występuje jeśli łącznie spełnione są warunki $r_1(p_2) = p_1$ oraz $r_2(p_1) = p_2$.

Wykażmy teraz, że funkcje r_1 i r_2 są identyczne. Wynika to z tego, że firmy mają dokładnie takie same koszty, a funkcja softmax jest niezmiennicza ze względu na permutację argumentów *oraz* że istnieje dokładnie jedno minimum lokalne funkcji zysku każdej firm przy zadanej cenie konkurenta. Aby pokazać tą ostatnią własność zauważmy, że zysk firmy 1 wynosi:

$$\pi_1(p_1, p_2) = \frac{e^{-p_i}}{e^{-p_1} + e^{-p_2}}(p_1 - 1). \tag{3.18}$$

Zauważmy, że pochodna $\pi_1(p_1, p_2)$ po p_1 wynosi:

$$\frac{\partial \pi_1}{\partial p_1} = \frac{(2 - p_1)\, e^{p_1+p_2} + e^{2p_2}}{2e^{p_1+p_2} + e^{2p_2} + e^{2p_1}}, \tag{3.19}$$

i warunek $\partial\pi_1/\partial p_1 = 0$ sprowadza się do równania $(2 - p_1)e^{p_1} + e^{p_2} = 0$. Zauważmy, że może mieć ono rozwiązania jedynie dla $p_1 > 2$, ale w przedziale $[2, +\infty]$ pochodna wyrażenia $(2-p_1)e^{p_1}$ wynosi $(1-p_1)e^{p_1}$ i jest stale ujemna i rośnie. W związku z tym równanie to ma dokładnie jedno rozwiązanie dla każdego p_2.

Zauważmy dodatkowo, że warunek $(2 - p_1)e^{p_1} + e^{p_2} = 0$ nie jest możliwy do rozwiązania analitycznie ze względu na p_1 w związku z tym maksymalizacja funkcji $\pi_1(p_1, p_2)$ ze względu na p_1 przy ustalonym p_2 musi być wykonywana numerycznie.

Wykazaliśmy więc, że funkcje reakcji firm 1 i 2 na ceny konkurenta istnieją i są identyczne. W takim razie ceny równowagi obu firm są sobie równe i muszą spełniać warunek $r_1(p_1) = p_1$. Wykorzystując to spostrzeżenie w warunku $(2-p_1)e^{p_1}+e^{p_2} = 0$ otrzymujemy analityczne ceny równowagi $p_1 = p_2 = 3$. Fakt ten wykorzystamy do weryfikacji wyniku numerycznego.

Pokażmy teraz jak numerycznie można wyznaczyć poszukiwane ceny równowagi. Zauważmy, że minimum funkcji $(r_1(p_1) - p_1)^2$ jest 0, a więc tak wyznaczony punkt jest rozwiązaniem równania $r_1(p_1) = p_1$. Wykorzystując to spostrzeżenie kod 3.16 prezentuje rozwiązanie zadania.

Kod 3.16. Wyznaczenie ceny równowagi w duopolu Bertranda z niedoskonałymi substytutami

```
profit <- function(t, p) {
    (t - 1) * exp(-t) / (exp(-t) + exp(-p))
}

r <- function(p) {
```

```
6     lo <- 2
7     hi <- 2
8     repeat {
9         hi <- hi + 1
10        if (profit(hi, p) < profit(hi - 1, p)) {
11            break
12        }
13    }
14    optimize (profit, c(lo, hi), maximum = T, p = p)$maximum
15 }
16
17 obj <- function(p) {
18    (r(p) - p) ^ 2
19 }
20
21 obj(2) # 0.3216571
22 obj(4) # 0.1961206
23 obj(6) # 1.152904
24
25 optimize(obj, c(2, 6))$minimum # 2.999997
```

W rozwiązaniu numerycznym optymalizacja dokonywana jest dwukrotnie. Funkcja `r` wyznacza optymalną reakcję gracza na cenę konkurenta równą `p`. W linijkach od 6 do 13 wyznaczany jest przedział, który będzie następnie przekazany do procedury optymalizacyjnej. Ponieważ wiemy z rozważań analitycznych, że maksimum funkcji zysku jest większe niż 2, więc dolna granica przedziału `lo` ustalona jest na tą wartość. Z kolei górna granica przeszukiwania jest zwiększana tak długo aż nie napotkamy na spadek wartości funkcji celu. Mamy wtedy pewność, że zmienna `hi` jest większa niż poszukiwane maksimum. Następnie znajdowana jest optymalna reakcja firmy przy wykorzystaniu procedury `optimize`. Posiadając metodę wyznaczania funkcji reakcji firmy w funkcji `obj` zapisano opisaną wcześniej funkcję $(r_1(p_1) - p_1)^2$, która osiąga minimum w cenie równowagi. W linijkach od 21 do 23 stwierdzamy, że minimum tej funkcji musi leżeć w przedziale $[2, 6]$, w takim razie wykorzystujemy jako punkt startowy optymalizacji. Otrzymane rozwiązanie numeryczne 2,999997 dokładnie przybliża znany analitycznie punkt optymalny równy 3.

W niniejszym przykładzie zaprezentowano podejścia wykorzystujące funkcję `optimize` do poszukiwania miejsc zerowych funkcji jednej poprzez transformację tego zadania do zadania optymalizacyjnego. Standardowym podejściem to tego problemu jest zastosowanie omówionej wcześniej funkcji `uniroot`. □

Przykład 3.6. Załóżmy, że dysponujemy prostym modelem regresji logistycznej, w którm objaśniamy zmienną binarną y za pomocą dwóch zmiennych objaśniających x_1 i x_2. Standardowo otrzymujemy z niego prognozy będące oszacowaniami prawdopodobieństwa przyjęcia przez zmienną y wartości 1. W praktyce często pojawia się zagadnienie wymagające podjęcia decyzji o klasyfikacji nowej obserwacji do odpo-

wiedniej klasy (0 lub 1) na podstawie oszacowanego prawdopodobieństwa. Dokonuje się tego ustalając tzw. próg odcięcia t i klasyfikując wszystkie prognozy mniejsze od tego progu jako 0, a większe jako 1. Takie postępowanie może prowadzić do dwóch rodzajów błędów. Pierwszy z nich pojawia się gdy obserwację klasyfikujemy jako 0, a naprawdę jest 1. Drugi gdy obserwacja zaklasyfikowana jako 1 okaże się być 0. Każdy z typów błędów może nieść ze sobą różny koszt. W niniejszym przykładzie zaprezentujemy sposób wyznaczania optymalnego progu odcięcia, gdzie kryterium optymalizacji jest minimalizacja łącznego kosztu błędów obu typów.

Kod 3.17 prezentuje przykładowy model, dla którego wyznaczono optymalny próg odcięcia w klasyfikacji.

Kod 3.17. Procedura wyznaczająca optymalny próg odcięcia w klasyfikacji

```
library(R2Cuba)

gen.data <- function(n) {
    x1 <- runif(n)
    x2 <- runif(n)
    y <- (2 * x1 - x2 - 1 + rlogis(n))>0
    data.frame(x1, x2, y)
}

set.seed(1)
est <- gen.data(1000)
opt <- gen.data(1000)
model <- glm(y ~ ., data = est, family = "binomial")
pred <- predict(model, newdata = opt, type = "response")

cost.s <- function(t) {
  (2 * sum((pred < t) & opt$y) +
   sum((pred >= t) & (!opt$y))) / length(opt$y)
}

cost.t <- function(t) {
    point.cost <- function(arg) {
        x1 <- arg[1]
        x2 <- arg[2]
        p.true <- plogis(2 * x1 - x2 - 1)
        if (sum(model$coef * c(1, x1, x2)) >= log(t / (1 - t))) {
          return(1 - p.true)
        }
        2 * p.true
    }
    cuhre(2, 1,  point.cost,
          lower = c(0, 0), upper = c(1, 1),
```

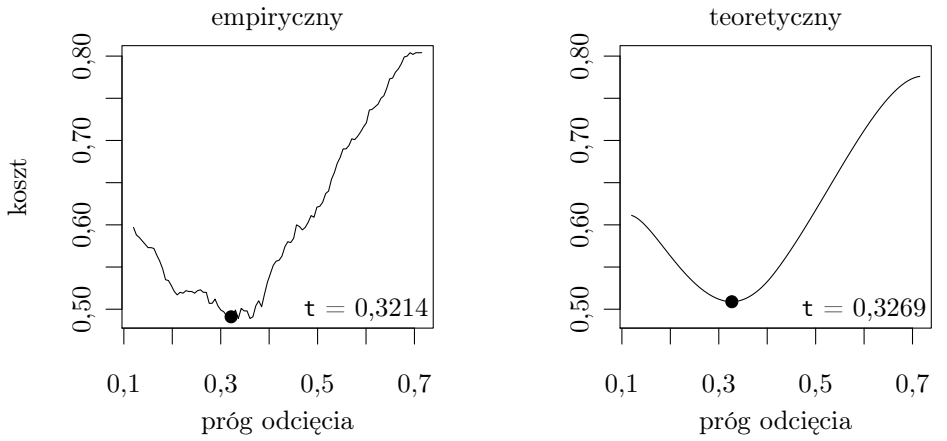
```
33          flags = list(verbose = 0, final = 0),
34          rel.tol = 0.0001, max.eval = 10 ^ 6)$value
35 }
36
37 optimize(cost.s, range(pred))$minimum # 0.3214393
38 optimize(cost.t, range(pred))$minimum # 0.3268846
```

W linijkach do 3 do 8 zawarta jest funkcja generująca dane wykorzystane w modelowaniu. Ponieważ znamy prawdziwy proces generowania danych możemy odnieść wyniki obliczeń na próbie do teoretycznych wynikających z prawdziwego modelu opisującego dane. Zauważmy, że proces generujący dane jest dobrany tak, że regresja logistyczna jest prawidłowo wyspecyfikowaną postacią modelu.

Standardowy proces wykorzystywany w modelowaniu predykcyjnym zawarto w linijkach od 11 do 14. Na podstawie zbioru danych `est` szacowany jest model regresji logistycznej, a zbiór `opt` będziemy wykorzystywali do wyznaczenia optymalnego progu odcięcia. Zakładamy, że koszt błędu w przypadku gdy prawdziwa wartość zmiennej objaśnianej wynosi 1 (wartość `TRUE` zmiennej `y`) jest dwa razy większy niż koszt błędu gdy zmienna ta przyjmuje przeciwną wartość (`y` równe `FALSE`).

Funkcja `cost.s` oszacowuje koszt błędnej klasyfikacji na podstaiw próby `opt` przy zadanym progu `t`. W linicje 37 koszt ten jest minimalizowany przy wykorzystaniu procedury `optimize`. Wynik optymalizacji graficznie przedstawiony jest w lewej części rysunku 3.4. Zwróćmy uwagę na fakt, że minimalizowana funkcja nie jest unimodalna w minimum. Wynika to z faktu, że zbiór danych `opt` jest tylko skończoną próbą. Dodatkowo warto pamiętać, że funkcja `cost.s` jest przedziałami stała i skokowo zmienia swoją wartość pomiędzy odcinkami stałej wartości. Pomiędzy dwoma kolejnymi oszacowaniami prawdopodobieństwa w zmiennej `pred` wartość funkcji `cost.s` się nie zmienia. Dodatkowo widzimy, że im mniejsza byłaby próba `opt`, tym dłuższe byłyby odcinki o stałej wartości tej funkcji. Z tych względów funkcja `optimize` nie gwarantuje znalezienia globalnego minimum badanej funkcji. Jednak widzimy, że w dobry sposób ona to minimum przybliża. To co jest gwarantowane to to, że zwrócony będzie minimalny punkt spośród tych, w których funkcjia `cost.s` była oszacowana. W takich sytuacjach rekomendowane jest wykorzystanie metod poszukujących globalnego minimum funkcji, które opisane są w rozdziale 6.

Natomiast funkcją, która może być bezpiecznie optymalizowana za pomocą `optimize` jest `cost.t`. Funkcja ta korzysta z faktu, że znamy teoretyczną specyfikację zależności między zmienną `y` a zmiennymi `x1` i `x2`. Przy zadanym progu odcięcia `t` wylicza ona numerycznie całkę za pomocy funkcji `cuhre` mówiącą jaki byłby koszt błędnej klasyfikacji gdybyśmy dysponowali nieskończoną próbą `opt`. Zwróćmy uwagę, że funkcja ta korzysta również z faktu, że zmienne `x1` i `x2` mają niezależny rozkład jednostajny na przedziale $[0, 1]$. Na prawej części rysunku 3.4 zobrazowano funkcję kosztu oszacowaną za pomocą tej metody. Widzimy, że w tym wypadku jest ona gładka oraz unimodalna w minimum. Warto zwrócić uwagę, że optymalna progu odcięcia wyznaczona na podstawie oszacowania teoretycznego jest bardzo zbliżona do wartości optymalnej progu odcięcia wyznaczonej na podstawie próby. □



Wykres 3.4: Porównanie optymalnych progów odcięcia wyznaczonych na podstawie próby empirycznej i wyliczeń teoretycznych

Zadania

1) Za pomocą algorytmu `golden` podjęto próbę minimalizacji funkcji $x^2 + 1$ na przedziale $[-1, 1]$ z tolerancją `tol` równą 10^{-12}. Procedura zatrzymuje się i zwraca wynik w przybliżeniu równy $7{,}070263/10^9$. Nie spełnia on jednak założonej tolerancji. Wyjaśnij dlaczego.
2) Rozważmy funkcję ciągłą $f\colon \mathbb{R} \to \mathbb{R}$, dla której poszukujemy minimum na przedziale $[a, b]$. Masz do dyspozycji dwie możliwości oszacowania wartości funkcji w punkcie. Jakie punkty wybierzesz aby zminimalizować długość przedziału, w którym na pewno znajduje się przynajmniej jedno minimum lokalne? Jaką procedurę zastosujesz jeśli masz do dyspozycji: (a) trzy porównania, (b) n porównań?
3) Podaj przykład takiej funkcji unimodalnej w minimum, która nie spełnia założeń metody Brenta, ale dla której jednak metoda ta zbiega do optimum szybciej niż metoda złotego podziału.
4) Podaj przykład takiego przedziału, poziomu tolerancji oraz funkcji unimodalnej w minimum, która spełnia założenia metody Brenta, ale dla której jednak metoda ta zbiega do optimum wolniej niż metoda złotego podziału.
5) W procedurze `band.search` w 31 linijce ze względów numerycznych testowany jest warunek `f(x2)<f(x3)` a nie `f(x2)<=f(x3)`, który teoretycznie (w przypadku nieskończonej precyzji obliczeń) powinien być wystarczający. Wyjaśnij dlaczego.
6) Skonstruuj przykład zastosowania funkcji `band.search` w którym badana funkcja jest unimodalna w minimum, a jednak nie jest ono znajdowane i zwracany jest komunikat: (a) `"Proper interval not found"`, (b) `"Function value is not finite"`,

(c) `"Function is not unimodal in minimum"`. Wszystkie te trzy sytuacje mogą wystąpić w wyniku problemów wynikami obliczeń numerycznych.

7) Procedurę `bisection` zastosowano do optymalizacji funkcji $f(x) = [|x| > 1](x^2-1)^2$ na przedziale $[-10, 10]$ przyjmując wartość `tol` równą 0,01. Sprawdź, że funkcja f jest różniczkowalna w sposób ciągły na $\mathbb{R}$ i wyznacz tą pochodną. Wyjaśnij jaki problem zostanie napotkany w trakcie wykonywania obliczeń. Jakie widzisz możliwości jego rozwiązania?
8) Wyjaśnij ilu iteracji potrzebuje algorytm Newtona do znalezienia optimum jeśli minimalizowana jest funkcja kwadratowa.
9) Niech $x_1 > a > 0$. Znajdź granicę ciągu $x_{n+1} = (x_n + a/x_n)/2$. Wyjaśnij związek tego ciągu z algorytmem Newtona. Jakie jest tempo jego zbieżności?
10) Jeżeli reguła Armijo przyjmuje proponowany punkt już w pierwszej iteracji to istnieje ryzyko, że inicjalny krok procedury (d) jest zbyt mały. Może to powodować wolną zbieżność procedury optymalizacyjnej. Zaproponuj modyfikację procedury `armijo.rule` uwzględniającą ten przypadek.
11) Zbadaj dla jakich wartości parametru `alpha` procedura `newton.armijo` zbiega do optimum w jednym kroku w przypadku gdy minimalizowana jest funkcja kwadratowa.
12) Skonstruuj przykłady w których procedura `newton.armijo` kończy działanie nie w minimum lokalnym funkcji, ale: (a) w maksimum lokalnym, (b) w punkcie przegięcia. Oceń, czy ryzyko takiej sytuacji jest duże.
13) Dokonaj modyfikacji procedury `newton.armijo` tak aby inicjalna długość kroku wyznaczana była za pomocą metody siecznych zamiast metody Newtona.
14) Sprawdź skutki zastąpienia w przykładzie 3.4 metody optymalizacji `newton.armijo` na `newton`.
15) W przykładach 3.3 oraz 3.2 zmień implementację rozwiązania tak aby wykorzystać funkcję `optimize` zamiast `uniroot`.
16) Zbadaj jaki skutek zmiany przykładzie 3.6 wielkości próby w zbiorach `est` i `opt`.
17) W przykładzie 3.6 wyznacz teoretyczny próg odcięcia w scenariuszu gdybyśmy do prognozowania wykorzystywali prawdziwy model generujący dane, a nie oszacowany na podstawie próby `est`.
18) W przykładzie 3.6 dokonaj modyfikacji funkcji `cost.s` tak aby zwracała ona wygładzone oszacowanie kosztu w punkcie odcięcia `t` wykorzystując informacje o oszacowaniu tego kosztu w lokalnym otoczeniu rozważanego punktu. Do wyznaczenia wygładzonych oszacowań możesz wykorzystać np. procedury `loess` lub `smooth.spline`.

4. Optymalizacja nieliniowa bez ograniczeń w $\mathbf{R}^n$

TODO: Dodać porównanie algorytmów -> gdy mamy i gdy nie mamy pochodnej.

Podstawy teoretyczne

Problem minimalizacji funkcji wielu zmiennych bez ograniczeń można przedstawić następująco:

$$\min_{\mathbf{x}} f(\mathbf{x})$$

gdzie $f : \mathbf{R}^n \to \mathbf{R}^n$. Dla wygody zakładamy, że funkcja f jest ciągła, chociaż niektóre algorytmy nie wymagają tego założenia.

Metody optymalizacji funkcji można podzielić w dwojaki sposób:

- Czy wykorzystywana jest informacja o krzywiźnie funkcji?
  - Metody bezgradientowe
  - Metody gradientowe
- Czy wybierany jest kierunek poszukiwań?
  - Metody poszukiwania wzdłuż kierunku
  - Inne metody

W niniejszej części skryptu omówione będą następujące algorytmy minimalizacji funkcji wielu zmiennych:

- Metody bezpośrednich poszukiwań:
  - Metoda hipersześcienna (ang. *evolutionary optimization*)
  - Metoda sympleksowa Neldera-Meada
  - Metoda kierunków sprzężonych (ang. *conjugate direction*) Powella
- Metody gradientowe (ang. *descent methods*)

- Metoda najszybszego spadku (ang. *steepest descent method*)
- Metoda Newtona
- Metoda Marquardta
- Metoda sprzężonego gradientu
- Metody quasinewtonowskie:
  * Metoda Broyden-Fletcher-Goldfarb-Shannona (BFGS)
  * Metoda Davidon-Fletcher-Powella (DFP)

4.0.1. Metody bezpośrednich poszukiwań

Tak jak w przypadku optymalizacji funkcji jednej zmiennej metody bezpośrednich poszukiwań korzystają wyłącznie z wartości funkcji w punktach, w przeciwieństwie do metod gradientowych, które dodatkowo wykorzystują pochodne funkcji.

Metoda hipersześcienna

Algorytm potrzebuje w pojedynczej iteracji $2^n + 1$ punktów, z czego 2^n punktów to są wierzchołki hipersześcianu scentrowanego na pozostałym punkcie. Porównuje się wartości funkcji we wszystkich tych punktach i wskazuje się najlepszy (z najmniejszą wartością). W następnej iteracji tworzy się sześcian wokół tego najlepszego punktu. Jeśli najlepszym punktem okaże się punkt, który był środkiem danego hipersześcianu, wówczas zmniejsza się rozmiar sześcianu. Proces ten kontynuuje się aż hipersześcian stanie się dostatecznie mały.

Algorytm

1) Wybierz punkt początkowy $\mathbf{x}(1)$ oraz parametry redukcji Δ_i dla każdej ze zmiennych, $i = 1, 2, ..., n$. Wybierz parametr zakończenia ϵ. Ustal $\bar{\mathbf{x}} = \mathbf{x}(1)$

2) Jeśli $||\boldsymbol{\Delta}|| < \epsilon$, **Zakończ**;

   W przeciwnym przypadku stwórz 2^n punktów poprzez dodanie i odjęcie $\Delta_i/2$ do/od każdej zmiennej w punkcie $\bar{\mathbf{x}}$.

3) Oblicz wartość funkcji dla wszystkich $(2^n + 1)$ punktów. Znajdź punkt z najmniejszą wartością. Ustal ten punkt jako $\bar{\mathbf{x}}$.

4) Jeśli $\bar{\mathbf{x}} = \mathbf{x}(1)$, zredukuj parametry redukcji $\Delta_i = \Delta_i/2$ i przejdź do kroku 2);

   W przeciwnym przypadku ustal $\mathbf{x}(1) = \bar{\mathbf{x}}$ i przejdź do kroku 2)

W powyższym algorytmie w każdej iteracji trzeba policzyć maksymalnie 2^n wartości fukcji. Czyli ilość potrzebnych ewaluacji funkcji wzrasta wykładniczo wraz z n.

Jeśli algorytm znajdzie dokładne minimum funkcji w którejś iteracji, nie zatrzymuje się automatycznie. Wartość $||\boldsymbol{\Delta}||$ musi spaść poniżej ϵ, aby algorytm zakończył działanie.

Ustalenie dużego parametru redukcji Δ_i jest dobre, ale wtedy algorytm może potrzebować wielu iteracji a zatem i wielu ewaluacji funkcji. Z drugiej strony ustalenie małego parametru redukcji może prowadzić do zbyt wczesnej zbieżności algorytmu do suboptymalnego punktu, szczególnie w przypadku funkcji bardzo nieliniowych. Redukcja parametru redukcji nie musi być dwukrotna tak jak w algorytmie podanym powyżej (patrz krok 4)). Dla lepszej zbieżności algorytmu rekomenduje się redukcję mniejszą niż dwukrotną[25].

Metoda sympleksu Neldera-Meada

Rozważmy funkcję $f\colon \mathbf{R}^n \to \mathbf{R}$, dla której szukamy minimum lokalnego. Algorytm Neldera–Meada jest metodą bezgradientową, która w procesie optymalizacji wykorzystuje jednocześnie kilka punktów referencyjnych. Punkty referencyjne są wybrane jako punkty sympleksu. Sympleks wymaga dużo mniejszej ilości punktów niż hipersześcian, co staje się szczególnie widoczne dla wielu wymiarów. Sympleks w n-wymiarowej przestrzeni ma $n+1$ wierzchołków. Jest to minimalna liczba wierzchołków umożliwiająca poszukiwanie we wszystkich możliwych kierunkach n-wymiarowej przestrzeni. Ważne jest jednak, aby sympleks, przy czym chodzi tutaj o sympleks początkowy, gdyż o następne nie trzeba się martwić, nie rozpinał figury o zerowej objętości w n-wymiarowej przestrzeni - czyli na przykład dla funkcji dwuwymiarowej trzy punkty sympleksu nie mogą leżeć na jednej linii, a w przypadku funkcji trójwymiarowej cztery punkty sympleksu nie mogą leżeć na jednej płaszczyźnie. Dokładnie rzecz ujmując: dokonując optymalizacji w przestrzeni $\mathbf{R}^n$ rozważamy $n+1$ punktów $\mathbf{x}^1, \mathbf{x}^2, \ldots \mathbf{x}^{n+1}$. Algorytm działa iteracyjnie i w każdym kroku stara się zmniejszyć wartość funkcji celu w przynajmniej jednym z rozważanych punktów zmieniając go. Bez utraty ogólności będziemy zakładali, że w konkretnej iteracji punkty $\mathbf{x}^i$ są tak uszeregowane, że $f(\mathbf{x}^i) \leq f(\mathbf{x}^{i+1})$.

Procedura działania wykorzystuje trzy *dodatnie* parametry α, β i γ spełniające dodatkowo warunki $\gamma > 1$ oraz $\beta < 1$. Standardowo i dla zapewnienia dobrej zbieżności algorytmu przyjmuje się $\alpha = 1, \beta = 0{,}5, \gamma = 2$. Algorytm będzie starał się poprawić lokalizację najgorszego punktu $\mathbf{x}^{n+1}$ tak aby zmniejszyć wartość funkcji. W tym celu wyznaczany jest punkt $\mathbf{x}^b = \sum_{i=1}^{n} \mathbf{x}^i/n$. Nowe położenie punktu będzie poszukiwane na prostej łączącej punkty $\mathbf{x}^b$ i $\mathbf{x}^{n+1}$. Procedura realizowana jest w następujących krokach:

1) *odbicie*: wyznaczany jest punkt $\mathbf{x}^r = (1+\alpha)\mathbf{x}^b + \mathbf{x}^{n+1}$ będący odbiciem punktu $\mathbf{x}^{n+1}$ względem punktu $\mathbf{x}^b$;
   jeżeli $f(\mathbf{x}^r) < f(\mathbf{x}^1)$ to przechodzi do kroku 2), w przeciwnym wypadku do kroku 3);
2) *rozszerzenie*: ponieważ kierunek przeszukiwań okazał się obiecujący wyznaczany jest punkt $\mathbf{x}^e = \gamma\mathbf{x}^r + (1-\gamma)\mathbf{x}^b$ oddalony γ razy dalej od punktu $\mathbf{x}^b$ niż punkt $\mathbf{x}^r$;
   jeżeli $f(\mathbf{x}^e) < f(\mathbf{x}^r)$ to nowa wartość $\mathbf{x}^{n+1}$ to $\mathbf{x}^e$, a w przeciwnym wypadku $\mathbf{x}^r$;
   obliczenia w tej iteracji procedury optymalizacyjnej kończą się;
3) *testowanie poprawy*: badane jest, czy $f(\mathbf{x}^r) < f(\mathbf{x}^{n+1})$;

[25] W kroku 4): $\Delta_i = \Delta_i/p$, gdzie $p \in (1, 2)$.

jeżeli tak to procedura przechodzi do kroku 4), a w przeciwnym razie do 5);

4) *zawężenie odbicia*: wyznaczany jest punkt $\mathbf{x}^{rc} = (1-\beta)\mathbf{x}^r + \beta\mathbf{x}^b$;
jeżeli $f(\mathbf{x}^{rc}) < f(\mathbf{x}^r)$ to nowa wartość $\mathbf{x}^{n+1}$ to $\mathbf{x}^{rc}$, a w przeciwnym wypadku $\mathbf{x}^r$;
obliczenia w tej iteracji procedury optymalizacyjnej kończą się;

5) *zawężenie*: wyznaczany jest punkt $\mathbf{x}^c = (1-\beta)\mathbf{x}^{n+1} + \beta\mathbf{x}^b$;
jeżeli $f(\mathbf{x}^c) < f(\mathbf{x}^{n+1})$ to nowa wartość $\mathbf{x}^{n+1}$ to $\mathbf{x}^c$, a w przeciwnym wypadku procedura przechodzi do kroku 6);

6) *pomniejszenie*: krok ten wykonywany jest w przypadku gdy nie powiodła się próba znalezienia punktu o mniejszej wartości funkcji niż $f(\mathbf{x}^{n+1})$;
w takiej sytuacji wszystkie punkty są przybliżane do aktualnie najlepszego zgodnie z regułą $\mathbf{x}^{si} = \beta\mathbf{x}^i + (1-\beta)\mathbf{x}^i$ dla $i = 2, 3, \ldots, n+1$.

Powyższą procedurę można podsumować za pomocą następującego zestawu reguł opisujących sposób aktualizacji listy punktów:

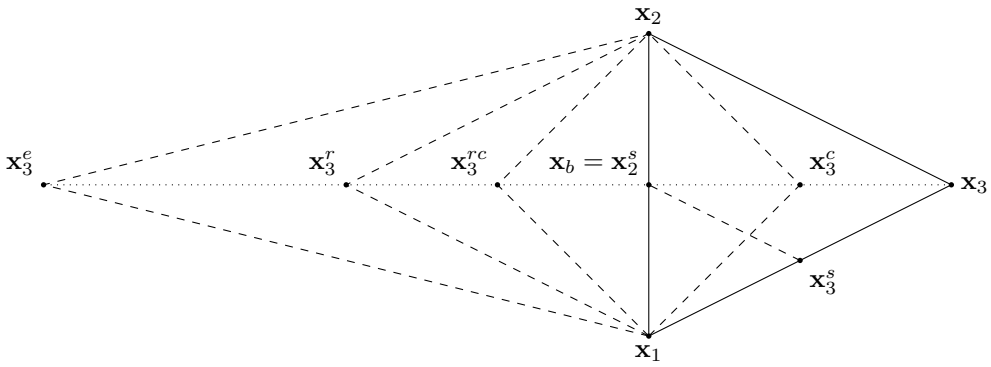
$$\begin{cases} f(\mathbf{x}^e) < f(\mathbf{x}^r) < f(\mathbf{x}^1) & \Rightarrow \quad \mathbf{x}_{n+1} := \mathbf{x}^e, \\ f(\mathbf{x}^{rc}) < f(\mathbf{x}^r) < f(\mathbf{x}^1) & \Rightarrow \quad \mathbf{x}^{n+1} := \mathbf{x}^{rc}, \\ f(\mathbf{x}^r) < \min\{f(\mathbf{x}^{n+1}), f(\mathbf{x}^e), f(\mathbf{x}^{rc})\} & \Rightarrow \quad \mathbf{x}^{n+1} := \mathbf{x}^r, \\ f(\mathbf{x}^c) < f(\mathbf{x}^{n+1}) < f(\mathbf{x}^r) & \Rightarrow \quad \mathbf{x}^{n+1} := \mathbf{x}^c, \\ f(\mathbf{x}^{n+1}) < \min\{f(\mathbf{x}^r), f(\mathbf{x}^c)\} & \Rightarrow \quad \mathbf{x}^i := \mathbf{x}^{si}. \end{cases} \tag{4.1}$$

W tym miejscu warto zwrócić uwagę, że powyższe zestawienie, w celu zapewnienia zwartego przedstawienia warunków, nie obejmuje przypadków, w których wartości funkcji w porównywanych punktach są równe. Klasyfikacja tych sytuacji do odpowiednich reguł w praktycznych implementacjach bywa różna.

Rysunek 4.1 prezentuje punkty wyznaczone zgodnie z powyższymi regułami w przypadku gdy dziedziną minimalizowanej funkcji jest $\mathbf{R}^2$. W tym wypadku wykorzystujemy trzy punkty $\mathbf{x}^1$, $\mathbf{x}^2$ oraz $\mathbf{x}^3$. Zgodnie z założeniem przyjmujemy, że $f(\mathbf{x}^1) \leq f(\mathbf{x}^2) \leq f(\mathbf{x}^3)$. Linią ciągłą połączono początkowo rozważane punkty. Linia przerywana oznacza prostą na której poszukiwana jest mniejsza niż w punkcie $\mathbf{x}^3$ wartość funkcji celu. Liniami przerywanymi połączono możliwe konfiguracje punktów otrzymanych w wyniku działania jednego kroku procedury.

Kod 4.1 prezentuje implementację omówionej metody. Procedura działa iteracyjnie i kończy działanie jeżeli różnica między największą i najmniejszą wartością funkcji w punktach $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^{n+1}$ różni się nie więcej niż o `tol`. W implementacji R dodatkowymi warunkami zakończenia działania procedury jest osiągnięcie maksymalnej liczby iteracji lub nie osiągnięcie zmniejszenia odległości pomiędzy badanymi punktami (najczęściej taka sytuacja zdaża się gdy badane punkty są w pozycji zdegenerowanej i z dokładnością do zaokrągleń numerycznych leżą na wspólnej $n-1$-wymiarowej hiperpłaszczyźnie).

Poza kryterium stopu procedura `neldermead` jest zgodna z jej implementacją w procedurze `optim`. W szczególności dla punktu startowego `x0` inicjalny zestaw punktów $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^{n+1}$ jest tworzony w ten sposób, że pierwszy z nich to `x0`. A pozostałe n jest tworzone poprzez kolejno dodanie do zmiennej `x0` na każdym z n wymiarów wartości równej 10% wartości bezwzględnej największej składowej wektora `x0`, jednak nie mniej niż `0.1`.



Przyjęto standardową parametryzację $\alpha = 1, \beta = 0{,}5, \gamma = 2$.

Wykres 4.1: Możliwe konfiguracje punktów otrzymywanych w metodzie Neldera–Meada

Kod 4.1. Implementacja algorytmu Neldera–Meada

```
neldermead <- function(f, x0, alpha, beta, gamma, tol) {
    n <- length(x0)
    step <- 0.1 * max(c(abs(x0), 1))
    x <- matrix(x0, nrow = n + 1, ncol = n, byrow = T) +
         step * rbind(0, diag(n))
    f.x <- apply(x, 1, f)
    while(diff(range(f.x)) > tol) {
        l <- which.min(f.x)
        f.l <- min(f.x)
        h <- which.max(f.x)
        f.h <- max(f.x)
        xb <- colMeans(x[-h,,drop = F])
        xr <- (1 + alpha) * xb -
              alpha * x[h,]
              f.xr <- f(xr)
        if (f.xr < f.l) {
            xe <- (1 - gamma) * xb +
                  gamma * xr
                  f.xe <- f(xe)
            if (f.xe < f.xr) {
                x[h,] <- xe
                f.x[h] <- f.xe
            } else {
                x[h,] <- xr
                f.x[h] <- f.xr
```

```
            }
        } else {
            shrink <- T
            if (f.xr < f.x[h]) {
                x[h,] <- xr
                f.x[h] <- f.xr
                shrink <- F
            }
            xc <- beta * xb + (1 - beta) * x[h,]
            f.xc <- f(xc)
            if (f.xc < f.x[h]) {
                x[h,] <- xc
                f.x[h] <- f.xc
            } else if ((f.xr >= f.x[h]) && shrink) {
                x <- matrix(x[l,], nrow = n + 1, ncol = n, byrow = T) +
                     beta * sweep(x, 2, x[l,])
                f.x <- apply(x, 1, f)
            }
        }
    }
    return(x[which.min(f.x),])
}
```

Tak jak wcześniej wspomniano, ważne jest aby początkowy sympleks nie był zdegenerowany. Jednym ze sposobów stworzenia sympleksu początkowego do pierwszego kroku algorytmu jest wybranie punktu bazowego $\mathbf{x}^1$ oraz liczby C. Wówczas $n+1$ punktów sympleksu to $\mathbf{x}^1$ oraz dla $i, j = 1, 2, ..., n$:

$$x_j^i = \begin{cases} x_j^1 + C & \text{jeeli } j = i \\ x_j^1 + C\Delta & \text{jeeli } j \neq i, \end{cases} \quad \text{gdzie } \Delta = \begin{cases} 0.25 & \text{jeeli } n = 3 \\ \frac{\sqrt{n+1}-2}{n-3} & \text{jeeli } n \geq 4. \end{cases}$$

Metoda kierunków sprzężonych Powella.

Metoda kierunków sprzężonych Powella jest chyba najbardziej popularną metodą bezpośrednich poszukiwań. Wykorzystuje ona historię poprzednich rozwiązań, aby stworzyć nowe kierunki poszukiwań. Idea jest prosta: trzeba utworzyć n liniowo niezależnych kierunków poszukiwań i dokonać sekwencyjnie serię poszukiwań wzdłuż tych kierunków, startując za każdym razem z poprzednio znalezionego punktu. Ta procedura gwarantuje znalezienie minimum funkcji kwadratowej w jednej serii poszukiwań wzdłuż każdego z n kierunków. W przypadku innych funkcji konieczne jest przeprowadzenie więcej niż jednej serii poszukiwań.

Algorytm bazuje na własności funkcji kwadratowych nazywanej własnością równoległych podprzestrzeni (ang. *paralel subspace property*). Własność ta jest zobrazowana poniżej.

Dana jest funkcja kwadratowa w dwóch wymiarach postaci $f(\mathbf{x}) = a + \mathbf{b}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T C\mathbf{x}$, gdzie $a \in \mathbf{R}$, $b \in \mathbf{R}^2$ i $C \in \mathbf{R^{2\times 2}}$. Rozważmy dwa punkty początkowe $\mathbf{x}^1$, $\mathbf{x}^2$ oraz kierunek $\mathbf{d}$. Niech $\mathbf{y}^1$ oraz $\mathbf{y}^2$ będą rozwiązaniami problemów poszukiwania wzdłuż kierunku $\mathbf{d}$ poczynając od punktów, kolejno $\mathbf{x}^1$ i $\mathbf{x}^2$. Można to zapisać nastepująco:

$$\mathbf{y}^i = \arg\min_{\lambda} f(\mathbf{x}^i + \lambda\mathbf{d})$$

Wówczas mówimy, że kierunek $\mathbf{y}^2 - \mathbf{y}^1$ jest C-sprzężony z kierunkiem $\mathbf{d}$, a więc zachodzi następujący warunek:

$$(\mathbf{y}^2 - \mathbf{y}^1)^T C\mathbf{d} = 0$$

Dla funkcji kwadratowych wiemy, że minimum leży na linii łączącej punkty $\mathbf{y}^1$ oraz $\mathbf{y}^2$.

W celu stworzenia pary kierunków C-sprzężonych, zamiast dwóch punktów $\mathbf{x}^1$, $\mathbf{x}^2$ oraz jednego kierunku poszukiwań $\mathbf{d}$, możemy alternatywnie wykorzystać jeden punkt $\mathbf{x}^1$ oraz dwa liniowo niezależne kierunki, np. wektory z bazy kanonicznej $\mathbf{d}^1 = (1, 0)^T$ oraz $\mathbf{d}^2 = (0, 1)^T$. Wówczas startując z punktu $\mathbf{x}^1$ dokonujemy poszukiwań wzdłuż kierunku $\mathbf{d}^1$ uzyskując jako rozwiązanie punkt $\mathbf{y}^1$. Następnie startując z tego punktu dokonujemy poszukiwań wzdłuż kierunku $\mathbf{d}^2$ i z nowego punktu dokonujemy jeszcze raz poszukiwań wzdłuż kierunku $\mathbf{d}^1$. Rozwiązaniem będzie punkt $\mathbf{y}^2$.

Własność równoległych podprzestrzeni może być uogólniona na przypadek n-wymiarowy, który będziemy stosować w algorytmie kierunków sprzężonych. Rozważmy funkcję kwadratową postaci $f(\mathbf{x}) = a + \mathbf{b}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T C\mathbf{x}$, gdzie $a \in \mathbf{R}$, $b \in \mathbf{R}^n$ i $C \in \mathbf{R^{n\times n}}$. Startujemy z punktu $\mathbf{x}^1$ i dokonujemy poszukiwań wzdłuż kierunku określonego przez wektor bazy kanonicznej, np. $\mathbf{e}^1 = (1, 0, \ldots, 0)$. Wynik oznaczamy jako $\mathbf{y}^1$. Następnie startując z punktu $\mathbf{y}^1$ dokonujemy serii n poszukiwań wzdłuż kierunków określonych przez kolejne wektory bazy kanonicznej $\mathbf{e}^i$, gdzie $i = 2, 3, \ldots, n, 1$, gdzie końcowy kierunek to $\mathbf{e}^1$. Wynik oznaczamy jako $\mathbf{y}^2$. Kierunek $\mathbf{e}^n$ może być teraz zastąpiony przez kierunek C-sprzężony z kierunkiem $\mathbf{e}^1$, czyli $\mathbf{y}^2 - \mathbf{y}^1$. Ta procedura może być teraz powtórzona startując od kierunku $\mathbf{e}^2$.

Algorytm

1) Wybierz punkt początkowy $\mathbf{x}(1)$ oraz zbiór n liniowo niezależnych kierunków; najlepiej $\mathbf{s}^i = \mathbf{e}^i$, dla $i = 1, 2, ..., n$, gdzie $\mathbf{e}^i$ oznacza i-ty wektor bazowy bazy kanonicznej.

2) Szukaj minimum startując z punktu początkowego wzdłuż kierunku $\mathbf{s}^1$. Startując z nowo znalezionego punktu (oznacz go jako $\mathbf{y}^1$), szukaj minimum wzdłuż kierunku $\mathbf{s}^2$. Kontynuuj szukanie wzdłuż kolejnych kierunków aż do kierunku $\mathbf{s}^n$. Następnie ponownie szukaj minimum wdłuż kierunku $\mathbf{s}^1$. Punkt, który został ostatecznie znaleziony oznacz jako $\mathbf{y}^2$.

3) Oblicz $\mathbf{d} = \mathbf{y}^2 - \mathbf{y}^1$. Jest to kierunek sprzężony do $\mathbf{s}^1$.

4) Jeśli $||\mathbf{d}||$ jest małe lub kierunki $\mathbf{s}^1, \mathbf{s}^2, ..., \mathbf{s}^{n-1}, \mathbf{d}$ są liniowo zależne, **Zakończ**; W przeciwnym przypadku ustal $\mathbf{s}^j = \mathbf{s}^{j-1}$ dla wszystkich $j = n, n-1, ..., 2$. Ustal $\mathbf{s}^1 = \mathbf{d}/||\mathbf{d}||$ i idź do kroku 2).

Aby znaleźć jeden kierunek C-sprzężony koniecznych jest dokładnie $n+1$ poszukiwań wzdłuż kierunku. Musimy znaleźć $n-1$ kierunków sprzężonych, ponieważ ostatni n-ty kierunek jest tym ostatnim, który sie ostał z wektorów bazy kanonicznej. A zatem potrzebnych jest $(n-1)(n+1) = n^2 - 1$ poszukiwań wzdłuż kierunku, aby otrzymać n kierunków C-sprzężonych. W przypadku funkcji kwadratowej po znalezieniu n kierunków C-sprzężonych potrzebne jest jeszcze jedno ostatnie poszukiwanie wzdłuż kierunku i otrzymujemy minimum.

4.0.2. Metody gradientowe

Gradient w punkcie $\mathbf{x}(t)$ możemy przybliżyć numerycznie za pomocą następującej formuły:

$$\nabla f(\mathbf{x}(t)) = \begin{bmatrix} \frac{\partial f(\mathbf{x}(t))}{\partial x_1} \\ \frac{\partial f(\mathbf{x}(t))}{\partial x_2} \\ ... \\ \frac{\partial f(\mathbf{x}(t))}{\partial x_n} \end{bmatrix}, \quad \text{gdzie } \frac{\partial f(\mathbf{x}(t))}{\partial x_i} = \frac{f(x_i(t) + \Delta x_i(t)) - f(x_i(t) - \Delta x_i(t))}{2\Delta x_i(t)}$$

Hesjan w punkcie $\mathbf{x}(t)$ natomiast liczymy następująco:

$$\nabla^2 f(\mathbf{x}(t) = \begin{bmatrix} \frac{\partial^2 f(\mathbf{x}(t))}{\partial x_1^2} & \frac{\partial^2 f(\mathbf{x}(t))}{\partial x_1 \partial x_2} & ... & \frac{\partial^2 f(\mathbf{x}(t))}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(\mathbf{x}(t))}{\partial x_2 \partial x_1} & \frac{\partial^2 f(\mathbf{x}(t))}{\partial x_2^2} & ... & \frac{\partial^2 f(\mathbf{x}(t))}{\partial x_2 \partial x_n} \\ ... & ... & ... & ... \\ \frac{\partial^2 f(\mathbf{x}(t))}{\partial x_n \partial x_1} & \frac{\partial^2 f(\mathbf{x}(t))}{\partial x_n \partial x_2} & ... & \frac{\partial^2 f(\mathbf{x}(t))}{\partial x_n^2} \end{bmatrix}$$

gdzie:

$$\begin{aligned} \frac{\partial^2 f(\mathbf{x}(t))}{\partial x_i^2} &= \frac{f(x_i(t) + \Delta x_i(t)) - 2f(x_i(t)) + f(x_i(t) - \Delta x_i(t))}{(\Delta x_i(t))^2} \\ \frac{\partial^2 f(\mathbf{x}(t))}{\partial x_i \partial x_j} &= \frac{\frac{\partial f(x_i(t) + \Delta x_i(t))}{\partial x_j} - \frac{\partial f(x_i(t) - \Delta x_i(t))}{\partial x_j}}{2\Delta x_i(t)} \end{aligned}$$

Pochodne cząstkowe w ostatnim wyrażeniu powyżej są z kolei liczone tak, jak składowe gradientu, tylko że w innym punkcie. Wyrażenie $x_i(t) + \Delta x_i(t)$ reprezentuje wektor $(x_1(t), ..., x_i(t) + \Delta x_i(t), ..., x_n(t))^T$

Żeby policzyć gradient potrzebnych jest $2n$ różnych wartości funkcji. Ażeby policzyć Hesjan potrzebnych jest $3n + 4 \begin{pmatrix} n \\ 2 \end{pmatrix} = 2n^2 + n$.

Ponieważ gradient jest kierunkiem najszybszego wzrostu, minus gradient jest kierunkiem najszybszego spadku funkcji. Kierunek poszukiwań (ang. *search direction*)

$\mathbf{d}(t)$ jest kierunkiem spadku w punkcie $\mathbf{x}(t)$, jeśli w otoczeniu tego punktu spełniony jest następujący warunek:

$$\nabla f(\mathbf{x}(t)) \cdot \mathbf{d}(t) \leq 0$$

Oznacza to, że cosinus kąta między gradientem i kierunkiem poszukiwań jest większy niż 90^0. Kierunek $\mathbf{d}(t)$ jest kierunkiem spadku, ponieważ w wyniku rozwinięcia f wokół $\mathbf{x}(t)$ otrzymujemy:

$$f(\mathbf{x}(t+1)) = f(\mathbf{x}(t) + \alpha \mathbf{d}(t)) = f(\mathbf{x}(t)) + \alpha \nabla f(\mathbf{x}(t)) \cdot \mathbf{d}(t).$$

Im niższa ujemna wartość $\nabla f(\mathbf{x}(t)) \cdot \mathbf{d}(t)$ tym większy spadek funkcji w kierunku $\mathbf{d}(t)$.

W metodach gradientowych często w ramach pojedynczej iteracji dokonuje się poszukiwań w danym kierunku. Jest to optymalizacja jednowymiarowa. Najpierw zapisujemy reprezentatywny punkt wzdłuż kierunku $\mathbf{s}(k)$ jako:

$$\mathbf{x}(k+1) = \mathbf{x}(k) + \alpha(k)\mathbf{s}(k),$$

gdzie $\alpha(k)$ jest długością kroku. Ponieważ $\mathbf{x}(k)$ i $\mathbf{s}(k)$ są znane, punkt $\mathbf{x}(k+1)$ można zapisać tylko jedną zmienną. Można więc wykonać minimalizację jednowymiarową, aby otrzymać nowy punkt $\mathbf{x}(k+1)$. Następnie poszukiwania kontynuuje się wzdłuż nowego kierunku $\mathbf{s}(k+1)$ i tak dalej aż do momentu znalezienia lokalnego minimum. Jeśli metoda gradientowa jest użyta do poszukiwań jednowymiarowych wzdłuż kierunku, minimum znajdujemy poprzez zróżniczkowanie wyrażenia $f(\mathbf{x}(t+1)) = f(\mathbf{x}(t) + \alpha \mathbf{s}(k))$ względem α i przyrównaniem do zera:

$$\nabla f(\mathbf{x}(k+1)) \cdot \mathbf{s}(k) = 0.$$

W ten sposób znajdujemy nowy punkt $\mathbf{x}(k+1)$. Okazuje się że kąt pomiędzy kierunkiem poszukiwań w k-tej iteracji i kierunkiem najszybszego spadku w nowym punkcie $-\nabla f(\mathbf{x}(k+1))$ jest równy 90^0.

Metoda Cauchy'ego najszybszego spadku

Kierunek poszukiwań w metodzie Cauchy'ego jest kierunkiem najszybszego spadku:

$$\mathbf{s}(k) = -\nabla f(\mathbf{x}(k)).$$

Algorytm ten gwarantuje poprawę, to jest spadek wartości funkcji, w każdej iteracji. Metoda najszybszego spadku działa dobrze, gdy punkt początkowy $\mathbf{x}(1)$ jest daleko od minimum $\mathbf{x}^*$. Jeśli bieżący punkt jest blisko minimum, zmiana gradientu jest mała, wobec następny punkt powstały w wyniku poszukiwania w jednym kierunku jest blisko punktu bieżącego.

Poniżej prezentujemy psuedokod algorytmu, w którym są wykorzystane dwa przykładowe warunki stopu. Warunki stopu mogą różnić się w zależności od implementacji.

Algorytm

1) Wybierz maksymalną liczbę iteracji M, punkt początkowy oraz dwa parametry zakończenia ϵ_1 i ϵ_2 i ustal $k = 1$.

2) Oblicz $\nabla f(\mathbf{x}(k))$

3) Jeśli $||\nabla f(\mathbf{x}(k))|| \leq \epsilon_1$, **Zakończ;**

   Jeśli $k \geq M$; **Zakończ;**

   W przeciwnym razie idź do kroku 4).

4) Wykonaj poszukiwanie wzdłuż kierunku, żeby znaleźć $\alpha(k)$ tak, aby $f(\mathbf{x}(k+1)) = f(\mathbf{x}(k)+\alpha(k)\mathbf{s}(k))$ było minimalne. Jednym z kryteriów zakończenia jest $|\nabla f(\mathbf{x}(k+1)) \cdot \nabla f(\mathbf{x}(k))| \leq \epsilon_2$.

5) Jeśli $\frac{||\mathbf{x}(k+1)-\mathbf{x}(k)||}{||\mathbf{x}(k)||} \leq \epsilon_1$, **Zakończ;**

   W przeciwnym przypadku ustal $k = k + 1$ i idź do kroku 2).

Metoda Newtona

Rozwinięcie w szereg Taylora drugiego rzędu funkcji f wokół punktu $\mathbf{x}(t)$ przyjmuje następującą formę:

$$f(\mathbf{x}(k+1)) = f(\mathbf{x}(k)) + \nabla f(\mathbf{x}(k))^T(\mathbf{x} - \mathbf{x}(k)) + \frac{1}{2}(\mathbf{x} - \mathbf{x}(k))^T \nabla^2 f(\mathbf{x}(k))(\mathbf{x} - \mathbf{x}(k))$$

Jeśli policzymy warunek pierwszego rzędu na minimum lokalne tej funkcji, otrzymujemy:

$$\nabla f(\mathbf{x}(k)) + \nabla^2 f(\mathbf{x}(k))(\mathbf{x} - \mathbf{x}(k)) = 0$$

Podstawiając $\mathbf{x}(k+1) = \mathbf{x}$, otrzymujemy:

$$\mathbf{x}(k+1) = \mathbf{x}(k) - \left[\nabla^2 f(\mathbf{x}(k))\right]^{-1} \nabla f(\mathbf{x}(k))$$

Kierunek poszukiwań w metodzie Newtona jest zatem dany wyrażeniem:

$$s(k) = -\left[\nabla^2 f(\mathbf{x}(k))\right]^{-1} \nabla f(\mathbf{x}(k))$$

Jeśli macierz $\left[\nabla^2 f(\mathbf{x}(k))\right]^{-1}$ jest półdodatnio określona, kierunek $s(k)$ jest kierunkiem spadku. Warunek drugiego rzędu optymalizacji mówi, że macierz $\nabla^2 f(\mathbf{x}^*)$ jest dodatnio określona dla minimum lokalnego. Można zatem założyć, że macierz $\nabla^2 f(\mathbf{x})$ jest dodatnio określona w otoczeniu minimum. Metoda Newtona jest więc dobra, kiedy punkt początkowy jest blisko minimum.

Algorytm jest bardzo podobny do metody najszybszego spadku. Poszukiwania prowadzone są jednak w innym kierunku $s(k) = -\left[\nabla^2 f(\mathbf{x}(k))\right]^{-1} \nabla f(\mathbf{x}(k))$. Możliwy warunek zakończenia optymalizacji wzdłuż kierunku wygląda następująco:

$$\left|\nabla f(\mathbf{x}(k+1)) \cdot \left[\nabla^2 f(\mathbf{x}(k))\right]^{-1} \nabla f(\mathbf{x}(k))\right| \leq \epsilon_2.$$

Kod 4.2. Implementacja algorytmu Newtona

```
library(numDeriv)

newton <- function(df, d2f, x, tol) {
    repeat {
        new.x <- x - solve(d2f(x), df(x))
        if (dist(rbind(new.x, x)) < tol) {
            return(new.x)
        }
        x <- new.x
    }
}

f <- function(x) {
    calls <<- calls + 1
    (1 - x[1]) ^ 2 + 100 * (x[2] - x[1] ^ 2) ^ 2 +
    (1 - x[2]) ^ 2 + 100 * (x[3] - x[2] ^ 2) ^ 2
}

calls <- 0
newton(function(x) grad(f, x),
       function(x) hessian(f, x),
       c(-3, 2, 4), 1e-6)         # 1 1 1
calls                             # 675
```

Metoda Marquardta

Metoda Cauchy'ego działa dobrze, gdy punkt początkowy jest daleko od minimum, podczas, gdy metoda Newtona działa dobrze, gdy punkt początkowy jest blisko minimum. W metodzie Marquardta, metodę Cauchy'ego stosuje się na początku by następnie zaadoptować metodę Newtona.

Algorytm

1) Wybierz maksymalną liczbę iteracji M, punkt początkowy oraz parametr zakończenia ϵ. Ustal $k = 1$ oraz $\lambda(1) = 10^4$ (duża liczba).

2) Oblicz $\nabla f(\mathbf{x}(k))$

3) Jeśli $||\nabla f(\mathbf{x}(k))|| \leq \epsilon$, **Zakończ;**

   Jeśli $k \geq M$; **Zakończ;**

   W przeciwnym razie idź do kroku 4).

4) Oblicz $\mathbf{x}(k+1) = \mathbf{x}(k) - \left[\nabla^2 f(\mathbf{x}(k)) + \lambda(k) I\right]^{-1} \nabla f(\mathbf{x}(k))$

5) Jeśli $f(\mathbf{x}(k+1)) < f(\mathbf{x}(k))$, idź do kroku 6);
   W przeciwnym przypadku idź do kroku 7).

6) Ustal $\lambda(k+1) = \frac{1}{2}\lambda(k)$, $k = k+1$ i idź do kroku 2).

7) Ustal $\lambda(k+1) = 2\lambda(k)$ i idź do kroku 4).

Algorytm może być szybszy, jeśli dodatkowo będziemy dokonywać optymalizacji wzdłuż kierunku w każdej iteracji.

Metoda sprzężonego gradientu (Conjugate gradient

Metoda sprzężonego gradientu bazuje na tej samej idei, co metoda kierunków sprzężonych Powella. W odróżnieniu od tamtej jest jednak metodą gradientową, wykorzystując informacje o krzywiźnie optymalizaowanej funkcji. Zarówno metoda kierunków sprzężonych, jak i sprzężonego gradientu narodziła się jako antidotum na podstawową wadę algorytmów Newtona oraz najszybszego spadku. Obie te metody, minimalizują funkcję f w danym kieruku $\mathbf{u}$ uzyskując punkt optymalny w tym kierunku. Następnie optymalizują funkcję f w innym kierunku $\mathbf{v}$ nie zachowując jednak optymalności względem poprzedniego kierunku $\mathbf{u}$. Dobry algorytm optymalizując w kierunku $\mathbf{v}$, powinien jednocześnie zachować optymalność względem kierunku $\mathbf{u}$.

Metody newtonowskie i quasinewtonowskie z iteracji na iterację przechowują oszacowanie macierzy Hesjana. Dla problemów z wieloma wymiarami może to stanowić problem. Na przykład jeśli gradient jest wektorem 1000-elementowym, wówczas Hesjan jest macierzą 1000x1000. Aby rozwiązywać takie problemy, potrzebne są metody, które mają mniejsze wymagania dotyczące przechowywania danych. Metody kierunków sprzężonych są metodami, które przechowują tylko gradient. Dodatkowa informacja dotycząca zakrzywienia funkcji jest również przekazywana, jednak bez konieczności przechowywania macierzy Hesjanu i historii przeszłych kierunków poszukiwań.

Aby sformalizować tą koncepcję, rozważmy funkcję kwadratową

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} + \mathbf{b}^T\mathbf{x} + c.$$

Gradient tej funkcji to $\nabla f = (A\mathbf{x} + \mathbf{b})^T$, a zatem zmiana gradientu w wyniku przesunięcia punktu $\mathbf{x}$ do punktu $\mathbf{x} + d\mathbf{x}$ to $Ad\mathbf{x}$. W wyniku minimalizacji funkcji f w kierunku $\mathbf{u}$, znajdujemy się w punkcie $\mathbf{x}$, dla którego zachodzi:

$$(A\mathbf{x} + \mathbf{b})^T\mathbf{u} = 0 \tag{4.2}$$

Jeśli teraz będziemy poruszali się z punktu $\mathbf{x}$ w kierunku $\mathbf{v}$, zmiana (4.2) wynosi $\mathbf{v}^T A\mathbf{u}$. Jeśli chcemy zagwarantować, aby warunek pierwszego rzędu względem kierunku $\mathbf{u}$ pozostał spełniony w miarę poruszania się względem nowego kierunku $\mathbf{v}$, musimy wybrać nowy kierunek $\mathbf{v}$ tak, aby spełniony był warunek

$$0 = \mathbf{v}^T A\mathbf{u}.$$

Jeśli u i $\mathbf{v}$ spełniają taki warunek, mówimy, że są kierunkami A-sprzężonymi.

Metod sprzężonego gradientu jest wiele, a różnią się one sposobem wyznaczania współczynnika korekty kierunku β. Oznaczmy przez $\mathbf{x}(i)$ ciąg punktów, w których jest funkcja,$\mathbf{t}_i$ jest kierunkiem przeszukiwania, wtedy.:

$$\begin{aligned}
&\text{Fletchera–Reevsa} \quad \beta_i^{\mathrm{FR}} = \frac{\nabla f(\mathbf{x}_i)^T \nabla f(\mathbf{x}_{i-1})}{||\nabla f(\mathbf{x}_{i-1})||^2}, \\
&\text{Polaka–Ribèrie} \quad \beta_i^{\mathrm{PR}} = \frac{\nabla f(\mathbf{x}_i)^T \big(\nabla f(\mathbf{x}_i) - \nabla f(\mathbf{x}_{i-1})\big)}{||\nabla f(\mathbf{x}_{i-1})||^2}, \\
&\text{Hestensa–Steifela} \quad \beta_i^{\mathrm{HS}} = \frac{\nabla f(\mathbf{x}_i)^T \big(\nabla f(\mathbf{x}_i) - \nabla f(\mathbf{x}_{i-1})\big)}{\big(\nabla f(\mathbf{x}_i) - \nabla f(\mathbf{x}_{i-1})\big)^T \mathbf{t}_{i-1}}.
\end{aligned} \tag{4.3}$$

Jeśli w którymś z tych wyrażeń mianownik jest zerowy to przyjmujemy $\beta_i = 1$. Dalej reguła aktualizacji $\mathbf{t}_i$ jest następująca:

$$\mathbf{t}_i = \beta_i \mathbf{t}_{i-1} - \nabla f(\mathbf{x}_i). \tag{4.4}$$

Kod 4.3. Implementacja algorytmu gradientu sprzężonego

```
cg <- function(f, grad.f, x, tol, type) {
    n <- length(x)
    f.x <- f(x)
    repeat {
        t <- g2 <- rep(0, n)
        step <- 1
        for (cycle in 1:n) {
            g <- grad.f(x)
            if (type == "FR") {
                G1 <- sum(g^2)
                G2 <- sum(g2^2)
            } else if (type == "PR") {
                G1 <- sum(g*(g-g2))
                G2 <- sum(g2^2)
            } else if (type == "BS") {
                G1 <- sum(g*(g-g2))
                G2 <- sum(t*(g-g2))
            } else {
                stop("Only FR, PR or BS type supported")
            }
            g2 <- g
            failed <- TRUE
            if (G1 <= tol) {
                break
            } else {
                if (G2 > 0) {
```

```
                    beta <- G1 / G2
                } else {
                    beta <- 1.0
                }
                t <- t * beta - g
                old.x <- x
                while (failed) {
                    x <- old.x + step * t
                    if (all(old.x == x)) {
                        break
                    } else {
                        f.x.new <- f(x)
                        if (f.x.new <= f.x + sum(t * g) * step / 10000) {
                            f.x <- f.x.new
                            failed <- FALSE
                        } else {
                            step <- step * 0.2
                        }
                    }
                }
                if (any(old.x != x)) {
                    x2 <- old.x + step * t / 2
                    f.x2 <- f(x2)
                    if (f.x2 < f.x) {
                        x <- x2
                        f.x <- f.x2
                    }
                }
            }
            step <- min(1, 1.7 * step)
            if (failed) {
                break
            }
        }
        if (cycle == 1 && (failed || (G1 <= tol))) {
            return(x)
        }
    }
}
```

Metody quasinewtonowskie - BFGS

TODO: BFGS

Algorytm Newtona posiada podobne problemy ze stabilnością jakie były omówione dla jego wersji jednowymiarowej. Z tego względu w praktycznych zastosowaniach wykorzystuje się jego modyfikację — metode BFGS (Broyden, Fletcher, Goldfarb, Shanno **TODO:** : cytacje). Jest to rozszerzenie idei metody siecznych. Poszukujemy aproksymacji kwadratowej $\tilde{f}$ funkcji f, która spełniać będzie warunki $\nabla f(\mathbf{x}_k) = \nabla \tilde{f}(\mathbf{x}_k)$ oraz $\nabla f(\mathbf{x}_{k-1}) = \nabla \tilde{f}(\mathbf{x}_{k-1})$. Oznaczmy $s_k = \mathbf{x}_k - \mathbf{x}_{k-1}$ oraz $y_k = \nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}_{k-1})$. Funkcja kwadratowa spełniająca te warunki zadana jest wzorem:

$$\tilde{f}(\mathbf{x}_k + \mathbf{h}) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)\mathbf{h} + \frac{1}{2}\mathbf{h}^T B_k \mathbf{h}. \tag{4.5}$$

gdzie B_k jest macierzą spełniającą warunek $B_k s_k = y_k$. Dodatkowo wymagamy, aby macież B_k była symetryczna i dodatnio określona. Wtedy funkcja $\tilde{f}$ posiada minimum w punkcie:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - B_k^{-1} \nabla f(\mathbf{x}_k). \tag{4.6}$$

Zwróćmy uwagę na fakt, że w przeciwieństwie do sytuacji jednowymiarowej macierz B_k nie jest wyznaczona w sposób jednoznaczny. Po pierwsze istnieje ona tylko w przypadku gdy:

$$s_k^T y_k > 0. \tag{4.7}$$

Warunek ten musi być spełniony aby możliwe było wyznaczenie macierzy B_k. W metodzie BFGS dokonywane jest to w sposób iteracyjny, przy czym operacje wykonywane są na macierzy H_k zdefiniowanej jako odwrotność B_k. W pierwszym kroku algorytmu przyjmowana jest macierz jednostkowa $H_1 = I$. Następnie w kroku k algorytm stara się wyznaczyć taką macierz H_k aby spełniała ona założone ograniczenia oraz jak najmniej różniła się od macierzy H_{k-1}. W związku z tym H_k jest rozwiązaniem zadania optymalizacyjnego:

$$\begin{aligned} &\min_H d(H, H_{k-1}) \\ &\text{pod warunkiem:} \\ &H = H^T, \\ &Hy_k = s_k. \end{aligned} \tag{4.8}$$

Definicja wykorzystywanej w algorytmie BFGS metryki d jest skomplikowana, podają ją np. Nocedal i Wright (1999). Rozwiązanie tego zadania optymalizacyjnego zadane jest wzorem:

$$H_k = \left(I - \frac{s_k y_k^T}{y_k^T s_k}\right) H_{k-1} \left(I - \frac{y_k s_k^T}{y_k^T s_k}\right) + \frac{s_k s_k^T}{y_k^T s_k}. \tag{4.9}$$

Dodatkowo zwróćmy uwagę na fakt, że iteracyjne obliczanie H_k zamiast B_k powoduje brak konieczności wyznaczania odwrotności macierzy przy dokonywaniu iteracji algorytmu.

Kod 4.4. Implementacja algorytmu BFGS

```
bfgs <- function(f, grad.f, x, tol) {
    n <- length(x)
    x <- matrix(x)
    f.x <- f(x)
    steps.left <- -1
    g <- matrix(grad.f(x), n)
    repeat {
        if (steps.left < 0) {
            H <- diag(n)
            steps.left <- 2 * n
        }
        old.x <- x
        g2 <- g
        dx <- -H %*% g
        if (t(g) %*% dx < 0) {
            repeat {
                x <- old.x + dx
                f.x.new <- f(x)
                if (f.x.new <= f.x + t(g) %*% dx / 10000) {
                    break
                } else {
                    dx <- dx * 0.2
                }
            }
            if (f.x-f.x.new < tol) {
                f.x <- f.x.new
                if (steps.left == 2 * n) {
                    break
                }
                steps.left <- -1
            } else {
                f.x <- f.x.new
                g <- matrix(grad.f(x), n)
                steps.left <- steps.left - 1
                dg <- g - g2
                D1 <- (t(dg) %*% dx)[1, 1]
                if (D1 > 0) {
                    D2 <- (t(dg) %*% H %*% dg)[1, 1]
                    D2 <- 1 + D2 / D1
                    U <- (H %*% dg) %*% t(dx)
                    H <- H + (D2 * dx %*% t(dx) - U - t(U)) / D1
                } else {
                    steps.left <- -1
                }
```

```
            }
        } else {
            if (steps.left == 2 * n) {
                break
            }
            steps.left <- -1
        }
    }
    dim(x) <- NULL
    return(x)
}
```

Przykłady zastosowań

Aby zilustrować działanie algorytmów optymalizacji przedsatwiony zostanie przykład monopolu. Rozważmy firmę, która produkuje dwa produkty y_1 oraz y_2. Załóżmy, że popyt na te produkty można wyprowadzić z następującej funkcji użyteczności

$$\begin{aligned} U(y_1, y_2) &= (y_1^\alpha + y_2^\alpha)^{\eta/\alpha} + M \\ &= u(y_1, y_2) + M, \end{aligned}$$

gdzie M to łączne wydatki na pozostałe dobra. Niech $\alpha = 0,98$ a $\eta = 0,85$. Jeśli firma produkuje ilość dóbr równą y_1 i y_2, to ceny tych dóbr wynoszą u_{y_1} oraz u_{y_2} (pochodne cząstkowe u względem zmiennych y_1 i y_2). Zatem problem monopolu można zapisać następująco:

$$\max_{y_1, y_2} \Pi(y_1, y_2) \equiv y_1 u_{y_1}(y_1, y_2) + y_2 u_{y_2}(y_1, y_2) - C_{y_1}(y_1) + C_{y_2}(y_2), \tag{4.10}$$

gdzie $C_{y_1}(y_1)$ oraz $C_{y_2}(y_2)$ to koszty produkcji y_1 i y_2. Niech $C_{y_1}(y_1) = 0,62y_1$ oraz $C_{y_2}(y_2) = 0,60y_2$.

Zauważmy, że próba wyliczenia wartości funkcji celu w punkcie, w którym y_1 lub y_2 jest ujemne, spowoduje przerwanie algorytmu, ponieważ ułamkowe potęgi ujemnych wartości są wartościami zespolonymi. Aby uniknąć tego typu problemów, przekształcimy problem w następujący sposób $x_1 \equiv \ln y_1$ oraz $x_2 \equiv \ln y_2$. Funkcję celu można zatem zapisać jako:

$$\max_{x_1, x_2} \pi(x_1, x_2), \tag{4.11}$$

gdzie $\pi(x_1, x_2) \equiv \Pi(e^{y_1}, e^{y_2})$. Teraz funkcja Π nigdy nie będzie wyliczana dla ujemnych argumentów. Powyższy system 4.11 będzie służył jako problem testowy. Rozwiązanie tego problemu to $x_1^* = -0,58$ oraz $x_2^* = 1,08$. Porównamy działanie kilku algorytmów optymalizacji nieliniowej bez ograniczeń. Punkt startowy wynosi $\mathbf{x}^0 = (x_1^0, x_2^0) = (1,5; 2,0)$. Funkcję celu oznaczać będziemy poprzez $f(\mathbf{x}) = \pi(x_1, x_2)$. Poniższe tabele przedstawiają informację na temat funkcjonowania wybranych algorytmów, tj. algorytm Newtona, kierunków sprzężonych, Newtona z poszukiwaniem wzdłuż kierunku, BFGS:

k	$\mathbf{x}(k) - \mathbf{x}^*$	$\mathbf{x}(k) - \mathbf{x}(k-1)$	$\nabla f(\mathbf{x}(k))/(1 + \lvert f(\mathbf{x}(k))\rvert)$

Zadania

1) Omów w jaki sposób algorytm Neldera–Meada poszukuje minimum funkcji $f : \mathbf{R} \to \mathbf{R}$. Porównaj tą procedurę do metod bezgradientowych omawianych w rozdziale 3.

5. Optymalizacja nieliniowa z ograniczeniami w $\mathbf{R}^n$

Podstawy teoretyczne

TODO: Metoda kar / bariery

TODO: L-BFGS-B?

TODO: Rsolnp?

Przykłady zastosowań

Zadania

1) .

6. Algorytmy heurystyczne

Podstawy teoretyczne

TODO: Symulowane wyżarzanie

W R jest: `maxit = 10000, tmax = 10, temp.ini = 10`

Kod 6.1. Implementacja procedury symulowanego wyżarzania

```
sann <- function(f, x0, maxit, tmax, temp.ini) {
    xb <- x <- x0
    f.xb <- f.x <- f(x0)
    for (i in 1:(maxit-1)) {
        if ((i %% tmax) == 1) {
            temp <- temp.ini / log(i + exp(1) - 1)
        }
            if (i >= maxit) { break }
            xtry <- x + rnorm(length(x)) / temp.ini
            f.xtry <- f(xtry)
            df <- f.xtry - f.x
            if ((df <= 0) || (runif(1) < exp(-df / temp))) {
                x <- xtry
                f.x <- f.xtry
                if (f.x < f.xb) {
                    xb <- x
                    f.xb <- f.x
                }
            }
    }
    return(xb)
}
```

Taki dokładnie algorytm jest w opcji `SANN` w funkcji `optim` w R.

TODO: Algorytmy genetyczne

Kod 6.2. Implementacja procedury algorytmu genetycznego

```
ga <- function(f, pop.size, maxit,
  init, cross, p.cross, mutate, p.mutate, prob) {
    pop.size2 <- round(pop.size / 2)
    pop <- replicate(2 * pop.size2, init(), simplify = FALSE)
    f.best <- +Inf
    for (i in 1:maxit) {
        f.pop <- sapply(pop, f)
        if (min(f.pop) < f.best) {
            best <- pop[[which.min(f.pop)]]
            f.best <- f(best)
        }
        pop <- sample(pop, replace = T, prob = prob(f.pop))
        for (i in 1:pop.size2) {
            if (runif(1) < p.cross) {
                pop[2 * i -1:0] <- cross(pop[2 * i -1:0])
            }
        }
        mut.sel <- runif(2 * pop.size2) < p.mutate
        pop[mut.sel] <- lapply(pop[mut.sel], mutate)
    }
    return(best)
}
```

Kod 6.3. Przykład zastosowania algorytmu genetycznego dla funkcji zmiennych rzeczywistych

```
set.seed(1)

f <- function (p) {
    -cos(p[1] * 4) + sum(p^2)/4
}

init <- function() {
    runif(2, min=-10, max=10)
}

cross <- function(p) {
    w <- runif(1)
    list(p[[1]] * w + p[[2]] * (1 - w),
         p[[2]] * w + p[[1]] * (1 - w))
}

mutate <- function(p) {
    p + rnorm(2)
```

```
}

prob <- function(f.p, scale = 4, tol = .Machine$double.eps^0.25) {
    if (max(f.p) - min(f.p) > tol) {
        return(1 + (scale - 1) * (max(f.p) - f.p) / (max(f.p) - min(f.p)))
    }
    return(rep(1, length(f.p)))
}

ga(f, 100, 10000, init, cross, 0.05, mutate, 0.01, prob)
```

Kod 6.4. Przykład zastosowania algorytmu genetycznego dla problemu pakowania plecaka

```
set.seed(1)
size <- 10
max.weight <- 2
weights <- runif(size)
f <- function (p) {
    total.weight <- sum(p*weights)
    if (total.weight > max.weight) {
        return(max.weight)
    }
    return(max.weight - total.weight)
}
init <- function() { sample(0:1, size, replace = T) }
cross <- function(p) {
    cut <- sample.int(size - 1, 1)
    return(list(c(p[[1]][1:cut], p[[2]][(cut+1):size]),
                c(p[[2]][1:cut], p[[1]][(cut+1):size])))
}
mutate <- function(p) {
    selector <- runif(size) < 1 / size
    p[selector] <- 1 - p[selector]
    return(p)
}
prob <- function(f.p) { order(f.p, decreasing = T) }
ga(f, 100, 10000, init, cross, 0.05, mutate, 0.01, prob)
```

Przykłady zastosowań

Zadania

1) .

7. Podsumowanie

Tutaj co dalej:
1) Gdzie więcej szukać o optymalizacji;
2) Co jest poza R;
3) Jakich zagadnień nie poruszyliśmy.

.

Bibliografia

[1] Brent R., Algorithms for Minimization without Derivatives, Prentice–Hall, 1973.

[2] Kahan W., Pracniques: further remarks on reducing truncation errors, *Communications of the ACM*, 8, s. 40, 1965.

[3] Kahan W., Miscalculating Area and Angles of a Needle–like Triangle, `http://http.cs.berkeley.edu/~wkahan/Triangle.pdf`, 2000.

[4] Nocedal J., Wright S.J., Numerical Optimization, Springer, 1999.