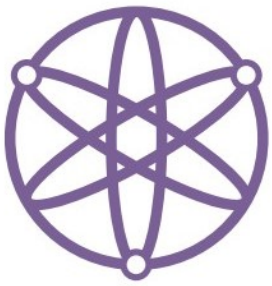


WYDZIAŁ
MATEMATYKI
I FIZYKI STOSOWANEJ
POLITECHNIKI RZESZOWSKIEJ

OPTYMALIZACJA NIELINIOWA - projekt

Autorzy:

Mateusz Rzeźnikiewicz oraz Patryk Ryba

Opiekun pracy:

dr Krzysztof Pupka

Rzeszów, 2024

Spis treści

Rozdział 1

Wstęp

1.1 Założenia projektu

Celem ćwiczenia jest zapoznanie się z metodami optymalizacji jednowymiarowej, zarówno gradientowymi, jak i bezgradientowymi, poprzez ich implementację oraz zastosowanie do wyznaczenia ekstremów dla podanej w dalszej części funkcji f.

Przedział poszukiwań wyznaczony zostanie przy użyciu dwupunktowej metody ekspansji Boxa-Davies-Swanna. Następnie do dokładnego określenia ekstremów funkcji zastosowane będą:

- metoda oparta na interpolacji Lagrange'a,
- metoda Newtona z warunkiem Armijo.

1.2 Zadania do wykonania

a) Metoda oparta na interpolacji Lagrange'a

Zadanie obejmuje przeprowadzenie wstępnego zawężenia przedziału poszukiwań dla 100 losowych punktów startowych z przedziału $[-80, 100]$, osobno dla każdej z trzech różnych długości kroku δ. Następnie należy przeprowadzić optymalizację dla wyznaczonego przedziału. Wymagane jest również przeprowadzenie optymalizacji bez wcześniejszego zawężania przedziału poszukiwań.

- Wyniki, oddzielnie dla minimalizacji i maksymalizacji, należy zapisać w pliku `xlsx` w arkuszu `_wyniki`.
- Średnie wartości należy przedstawić w arkuszu `_wartości_średnie`.
- Dla przypadków bez wstępnego zawężenia przedziału poszukiwań należy stworzyć wykresy przedstawiające długość przedziału $[a, b]$ jako funkcję numeru iteracji. Wykresy należy umieścić w arkuszu `_wykres`.

b) Metoda Newtona z warunkiem Armijo

Zadanie polega na przeprowadzeniu wstępnego zawężenia przedziału poszukiwań dla 100 losowych punktów startowych z przedziału $[-100, 100]$, osobno dla każdej z trzech różnych długości kroku δ. Następnie należy przeprowadzić optymalizację dla punktu startowego będącego środkiem wyznaczonego przedziału.

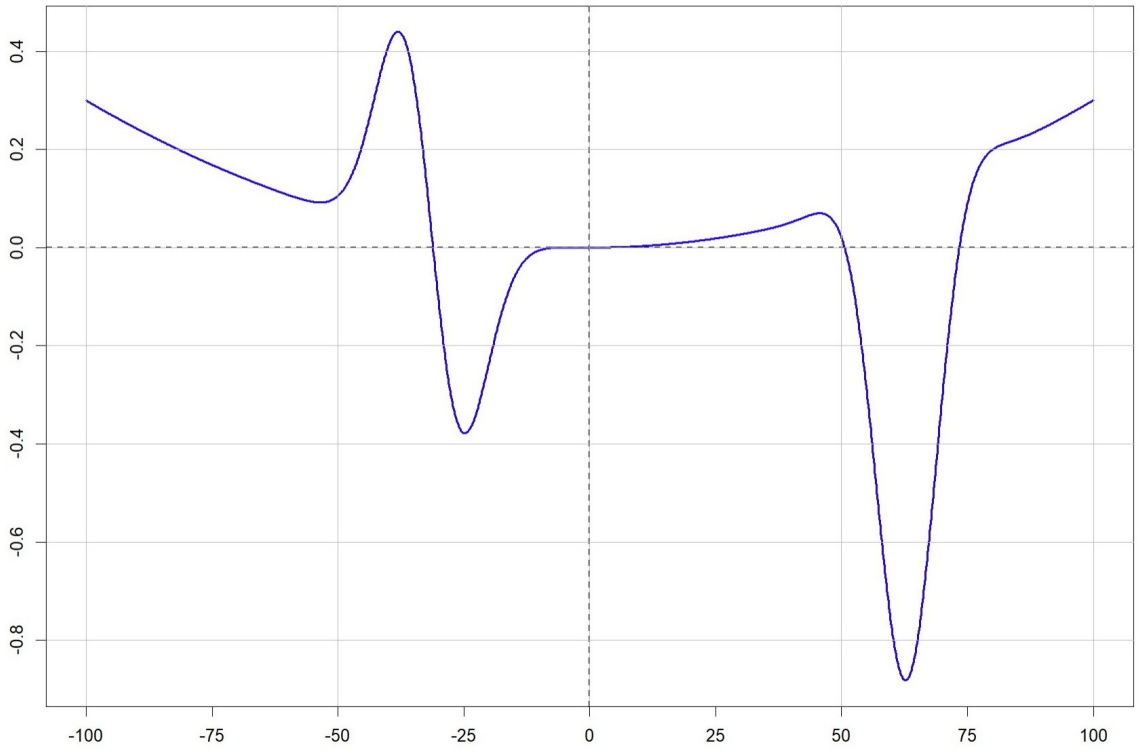
- Wyniki, oddzielnie dla minimalizacji i maksymalizacji, należy zapisać w pliku `xlsx` w arkuszu `_wyniki`.
- Średnie wartości należy przedstawić w arkuszu `_wartości_średnie`.

1.3 Funkcja celu i jej wykres

Funkcja celu to matematyczna formuła, która opisuje wielkość, którą chcemy maksymalizować lub minimalizować. W odróżnieniu od funkcji celu w programowaniu liniowym, funkcja ta może mieć postać nieliniową, co sprawia, że problem jest bardziej złożony i wymaga zaawansowanych metod rozwiązania, takich jak metody gradientowe czy metody bezgradientowe. Funkcja celu dla naszego problemu dana jest wzorem:

$$f(x) = \sin\left(\frac{x}{10}\right) e^{-\left(\frac{x}{10}+\pi\right)^2} - \cos\left(\frac{x}{10}\right) e^{-\left(\frac{x}{10}-2\pi\right)^2} + 0.003\left(\frac{x}{10}\right)^2 \tag{1.1}$$

Wykres funkcji f dla $x \in [-100, 100]$, czyli na całej jej dziedzinie przedstawiono poniżej.

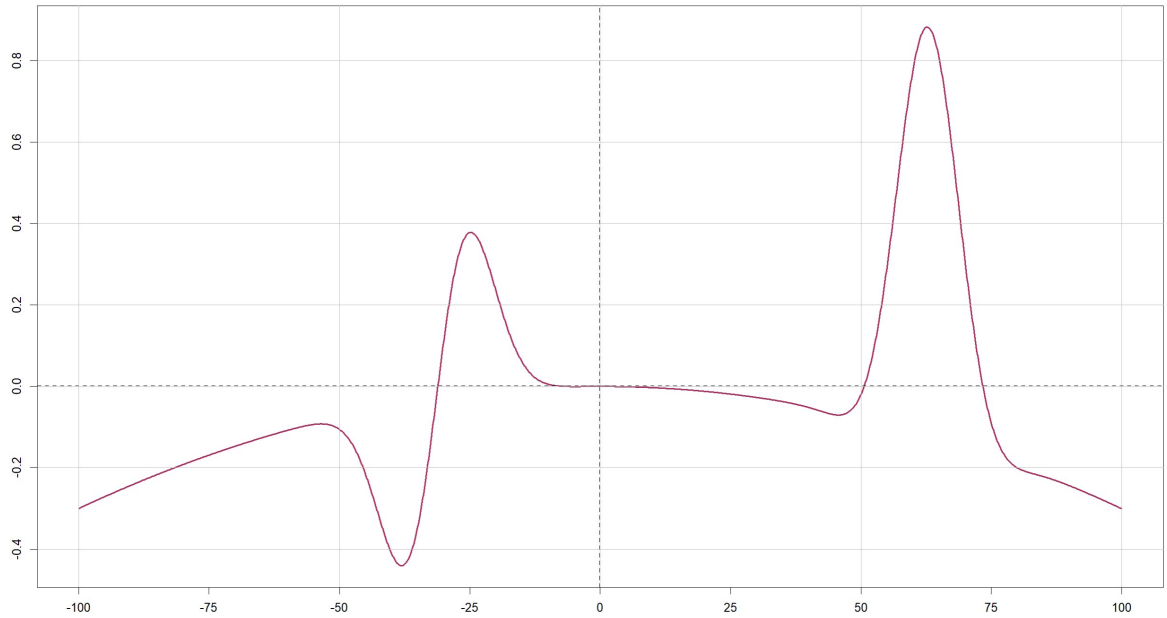


Rysunek 1.1: Wykres funkcji celu f dla D_f

Na podstawie wykresu łatwo wywnioskować, że funkcja f jest unimodalna na zadanym przedziale, a co za tym idzie również ciągła. Znalezienie optimum funkcji nieciągłej jest niemałym wyzwaniem i wymaga stosowania złożonych algorytmów.

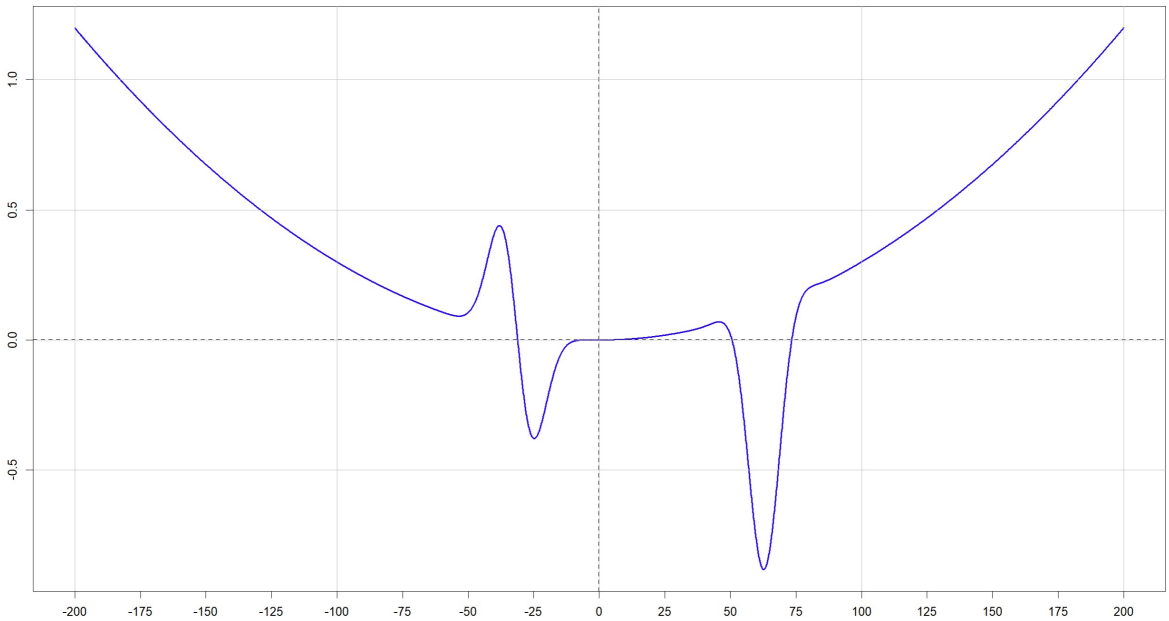
Przed przystąpieniem do opisu oraz implementacji poszczególnych algorytmów przedstawimy ogólny sposób postępowania. Każda metoda optymalizacyjna będzie opisywała proces znajdowania minimum funkcji. W przypadku poszukiwania maksimum zdefiniujemy funkcję pomocniczą jako $y = -f(x)$ i zastosujemy te same algorytmy.

Kody źródłowe będą prezentowane wyłącznie dla przypadku znajdowania minimum, ponieważ dla znajdowania maksimum są one analogiczne. Aby ułatwić sobie analizę, poniżej zaprezentujemy wykres funkcji $-f$.



Rysunek 1.2: Wykres funkcji pomocniczej $-f$ dla D_f

Warto także sprawdzić jak zachowuje się funkcja f poza swoją dziedziną i jaki przybiera kształt. Posłuży nam do tego poniższy wykres, który przedstawia $x \in [-200, 200]$.



Rysunek 1.3: Wykres funkcji f na poszerzonej dziedzinie

Rozdział 2

Algorytmy optymalizacyjne

2.1 Metoda ekspansji Boxa-Daviesa-Swanna

W sytuacjach w których wiemy, że funkcja celu posiada minimum na zbiorze $\mathbb{R}$ i chcemy je znaleźć, potrzebujemy odpowiedniego przedziału startowego. Zły (przypadkowy) wybór przedziału początkowego w procedurze optymalizacyjnej, może spowodować, że nie będzie ona zbieżna. Metody ekspansji służą do wstępnego określenia przedziału $[a, b]$, w którym znajduje się minimum funkcji unimodalnej.

Dwupiunktowa metoda ekspansji służy do znajdowania przedziału, w którym funkcja osiąga swoje minimum, poprzez iteracyjne poszerzanie zakresu wokół punktu początkowego x_0, w celu wykrycia wartości funkcji, które są większe od wartości w tym punkcie. Procedurę rozpoczynamy od sprawdzenia wartości funkcji w prawym sąsiedztwie dowolnego punktu x_0, co polega na porównaniu wartości funkcji w punktach:

$$x_{i+1} = x_0 + 2^i \delta, \tag{2.1}$$

gdzie δ to ustalona z góry długość kroku, a $i \in \mathbb{N}_0$. Poszukiwanie kontynuujemy, dopóki nie znajdziemy punktu, w którym spełniony zostanie warunek:

$$f(x_{i+1}) \geq f(x_i). \tag{2.2}$$

Punkt x_{i+1} staje się wówczas górną granicą przedziału poszukiwań, czyli $b = x_{i+1}$. Dolną granicę określamy w analogiczny sposób, jednak stosujemy następujący wzór iteracyjny:

$$x_{i+1} = x_0 - 2^i \delta, \quad i \in \mathbb{N}_0. \tag{2.3}$$

Proces wyznaczania dolnej granicy przedziału poszukiwań trwa, dopóki nie zostanie znaleziony punkt spełniający warunek

$$f(x_{i+1}) \geq f(x_i). \tag{2.4}$$

Wówczas punkt ten wyznacza dolną granicę przedziału poszukiwań, czyli $a = x_{i+1}$.

W kontekście dalszych rozważań nad algorytmami optymalizacyjnymi warto wspomnieć o liczniku wywołań funkcji celu, który użyjemy w każdym z naszych kodów. Jest on miarą tego, jak często w trakcie iteracji procesu optymalizacji obliczane są wartości funkcji celu. Stosuje się go do monitorowania efektywności algorytmu.

2.1.1 Implementacja kodu w języku R

```
expansionBDS <- function(f, x0, delta) {
  # Inicjalizacja licznika wywołań funkcji celu
  function_calls <- 0

  # Obliczenie wartości funkcji w x0
  f_x_prev <- f(x0)
  function_calls <- function_calls + 1
  cat("Wartość funkcji w punkcie początkowym wynosi", f_x_prev, "\n")

  # Inicjalizacja zmiennej i dla iteracji górnej granicy
  i_up <- 0

  # Inicjalizacja dla górnej granicy
  x_upper <- x0 + 2^i_up * delta
  f_x_upper <- f(x_upper)
  function_calls <- function_calls + 1

  # Iteracyjne zwiększanie x0 w celu wyznaczenia górnej granicy
  cat("\nIteracja dla górnej granicy:\n")
  cat("Dla i =", i_up, ", b =", x_upper, ", f(b) =", f_x_upper, "\n")

  while (f_x_upper < f_x_prev) {
    i_up <- i_up + 1
    f_x_prev <- f_x_upper
    x_upper <- x0 + 2^i_up * delta
    f_x_upper <- f(x_upper)
    function_calls <- function_calls + 1

    # Sprawdzamy, czy wartość funkcji jest skończona
    if (!is.finite(f_x_upper)) {
      stop("Wartość funkcji nie jest skończona w górnej granicy")
    }

    cat("i =", i_up, ", b =", x_upper, ", f(b) =", f_x_upper, "\n")
  }

  cat("Po liczbie iteracji:", i_up, "\n")
  cat("Górna granica: b =", x_upper, ", f(b) =", f_x_upper, "\n")
```

```
  # Inicjalizacja zmiennej i dla iteracji dolnej granicy
  i_low <- 0

  # Inicjalizacja dla dolnej granicy
  f_x_prev <- f(x0)  # Reset wartości odniesienia dla dolnej granicy
  x_lower <- x0 - 2^i_low * delta
  f_x_lower <- f(x_lower)
  function_calls <- function_calls + 1

  # Iteracyjne zmniejszanie x0 w celu wyznaczenia dolnej granicy
  cat("\nIteracja dla dolnej granicy:\n")
  cat("Dla i =", i_low, ", a =", x_lower, ", f(a) =", f_x_lower, "\n")

  while (f_x_lower < f_x_prev) {
    i_low <- i_low + 1
    f_x_prev <- f_x_lower
    x_lower <- x0 - 2^i_low * delta
    f_x_lower <- f(x_lower)
    function_calls <- function_calls + 1

    # Sprawdzamy, czy wartość funkcji jest skończona
    if (!is.finite(f_x_lower)) {
      stop("Wartość funkcji nie jest skończona w dolnej granicy")
    }

    cat("i =", i_low, ", a =", x_lower, ", f(a) =", f_x_lower, "\n")
  }

  cat("Po liczbie iteracji:", i_low, "\n")
  cat("Dolna granica: a =", x_lower, ", f(a) =", f_x_lower, "\n")

  # Wyświetlenie liczby wywołań funkcji celu
  cat("\nŁączna liczba wywołań funkcji celu:", function_calls, "\n")

  return(range(x_lower, x_upper))
}
```

2.2 Metoda oparta na interpolacji Lagrange'a

Metoda interpolacji Lagrange'a opiera się na założeniu, że każdą funkcję można przybliżyć wielomianem kwadratowym z różnym stopniem dokładności. Jest to szczególnie skuteczne w przypadku funkcji, których wykres jest zbliżony wyglądem do paraboli. Dla innych funkcji metoda ta będzie nieskuteczna, a nawet rozbieżna.

Algorytm polega na obliczeniu wartości funkcji w trzech punktach (najczęściej są to początek, środek i koniec przedziału poszukiwań), a następnie użyciu tych wartości do skonstruowania aproksymacji parabolicznej. Przez te punkty prowadzony jest wielomian drugiego stopnia, który wyznacza się przy użyciu wzoru interpolacyjnego Lagrange'a. W każdej iteracji wybierany jest czwarty punkt, który pozwala na zawężenie przedziału poszukiwań $[a, b]$, koncentrując się na obszarze, gdzie funkcja może osiągnąć minimum lokalne.

Dla trzech punktów $a < c < b$ z wartościami funkcji $f(a)$, $f(c)$ i $f(b)$, wielomian drugiego stopnia $w(x)$, który interpoluje te punkty, wyrażony jest wzorem:

$$w(x) = \sum_{k=1}^{3} f(x_k) \prod_{\substack{j=1 \\ j \neq k}}^{3} \frac{(x - x_j)}{(x_k - x_j)}, \tag{2.5}$$

gdzie $x_1 = a$, $x_2 = c$, $x_3 = b$.

Ten wielomian osiąga minimum w wierzchołku paraboli, danym wzorem:

$$d = \frac{1}{2} \frac{f(a)(c^2 - b^2) + f(c)(b^2 - a^2) + f(b)(a^2 - c^2)}{f(a)(c - b) + f(c)(b - a) + f(b)(a - c)}. \tag{2.6}$$

Należy zauważyć, że jeśli punkt d leży poza zbiorem $(a, c) \cup (c, b)$, wtedy algorytm nie jest zbieżny. Dla $d = c$ istnieją tylko trzy punktu podziału, co implikuje brak możliwości zawężenia, zaś dla pozostałych przypadków wypadamy poza przedział poszukiwań.

Załóżmy, że $d \in (a, c) \cup (c, b)$. Jeśli znamy wartość $f(d)$, możemy zawęzić obszar analizy i powtórzyć procedurę. Zawężanie przedziału odbywa się w następujący sposób:

- Jeżeli $a^{(i)} < d^{(i)} < c^{(i)}$ oraz $f(d^{(i)}) < f(c^{(i)})$, wówczas

$$a^{(i+1)} = a^{(i)}, \quad c^{(i+1)} = d^{(i)}, \quad b^{(i+1)} = c^{(i)}.$$

- Jeżeli $a^{(i)} < d^{(i)} < c^{(i)}$ oraz $f(d^{(i)}) \geq f(c^{(i)})$, wówczas

$$a^{(i+1)} = d^{(i)}, \quad c^{(i+1)} = c^{(i)}, \quad b^{(i+1)} = b^{(i)}.$$

- Jeżeli $c^{(i)} < d^{(i)} < b^{(i)}$ oraz $f(d^{(i)}) < f(c^{(i)})$, wówczas

$$a^{(i+1)} = c^{(i)}, \quad c^{(i+1)} = d^{(i)}, \quad b^{(i+1)} = b^{(i)}.$$

- Jeżeli $c^{(i)} < d^{(i)} < b^{(i)}$ oraz $f(d^{(i)}) \geq f(c^{(i)})$, wówczas

$$a^{(i+1)} = a^{(i)}, \quad c^{(i+1)} = c^{(i)}, \quad b^{(i+1)} = d^{(i)}.$$

Jeżeli długość przedziału $[a^{(i)}, b^{(i)}]$ maleje w kolejnych iteracjach, to metoda jest zbieżna do minimum funkcji celu w przedziale $[a^{(0)}, b^{(0)}]$.

Zawężanie przedziału poszukiwań kontynuujemy, aż długość tego przedziału będzie mniejsza od zadanej dokładności ε, co przedstawia poniższy wzór:

$$|b - a| < \varepsilon. \tag{2.7}$$

Aby zapobiec stagnacji przedziału w niektórych przypadkach dodajemy kolejny warunek stopu. Kryterium zatrzymania obejmuje wówczas dodatkowy składnik:

$$|d^{(i)} - d^{(i-1)}| < \gamma. \tag{2.8}$$

Mała wartość parametru γ wskazuje na wysoką precyzję w obliczeniach, co oznacza, że dalsze iteracje nie będą miały znaczącego wpływu na dokładność rozwiązania. Współczynnik dokładności γ powinien być znacząco mniejszy niż ε.

2.2.1 Implementacja kodu w języku R

```
lagrange <- function(f, a, b, epsilon, gamma) {
  # Licznik wywołań funkcji celu
  function_calls <- 0

  # Inicjalizacja punktu środkowego c
  c <- (a + b) / 2
  # Obliczenie wartości funkcji w punktach a, c, b
  fa <- f(a)
  fc <- f(c)
  fb <- f(b)
  function_calls <- function_calls + 3

  # Inicjalizacja dla porównania d(i-1)
  d_old <- Inf
  i <- 0
  max_iter <- 1000

  repeat {
    licznik <- fa * (c^2 - b^2) + fc * (b^2 - a^2) + fb * (a^2 - c^2)
```

```
mianownik <- fa * (c - b) + fc * (b - a) + fb * (a - c)

# Sprawdzenie dzielenia przez zero
if (abs(mianownik) < .Machine$double.eps) {
  cat("Mianownik bliski zeru - dzielenie przez zero.\n")
  break
}

d <- 0.5 * licznik / mianownik

# Sprawdzenie, czy punkt d należy do przedziału [a, b]
if (d <= a || d >= b) {
  cat("Brak minimum lokalnego w obrębie przedziału.\n")
  break
}

# Sprawdzenie, czy d jest zbyt bliskie c
if (abs(d - c) < .Machine$double.eps) {
  cat("Brak możliwości zawężenia przedziału, d zbyt bliskie c.\n")
  break
}

fd <- f(d)
function_calls <- function_calls + 1

if (a < d && d < c) {
  if (fd < fc) {
    b <- c
    c <- d
    fb <- fc
    fc <- fd
  } else {
    a <- d
    fa <- fd
  }
} else if (c < d && d < b) {
  if (fd < fc) {
    a <- c
    c <- d
    fa <- fc
```

```
      fc <- fd
    } else {
      b <- d
      fb <- fd
    }
  }

  # Wyświetlanie informacji diagnostycznych
  cat("Iteracja:", i, "\n")
  cat("a:", a, "f(a):", fa, "\n")
  cat("c:", c, "f(c):", fc, "\n")
  cat("b:", b, "f(b):", fb, "\n")
  cat("d:", d, "f(d):", fd, "\n")
  cat("Długość przedziału:", abs(b - a), "\n")
  cat("------------------------------------------------------------\n")

  # Sprawdzenie głównego warunku stopu
  if (abs(b - a) < epsilon) {
    cat("Długość przedziału jest mniejsza niż epsilon.\n")
    break
  }
  # Sprawdzenie dodatkowego warunku stopu
  if (abs(d - d_old) < gamma) {
    cat("Zmiana w punkcie d jest mniejsza niż gamma.\n")
    break
  }

  d_old <- d

  if (i >= max_iter) {
    cat("Osiągnięto maksymalną liczbę iteracji.\n")
    break
  }

  i <- i + 1
}

cat("\nŁączna liczba wywołań funkcji celu:", function_calls, "\n")
return(list(min_x = d, fmin_x = f(d)))
}
```

2.3 Metoda Newtona z warunkiem Armijo

Metoda Newtona-Armijo to technika optymalizacyjna, która modyfikuje standardową metodę Newtona poprzez wprowadzenie warunku Armijo do kontroli długości kroku. Ta modyfikacja pomaga w zapewnieniu stabilności algorytmu, szczególnie gdy sama metoda Newtona może prowadzić do zbyt dużych kroków, przez co minimum funkcji może być pominięte. Warunek Armijo zapewnia, że każdy krok prowadzi do zmniejszenia wartości funkcji w kontrolowany sposób.

2.3.1 Metoda Newtona

Metoda Newtona jest jedną z najważniejszych metod optymalizacyjnych używanych do rozwiązywania problemów minimalizacji funkcji. Wykorzystuje ona zarówno pierwszą, jak i drugą pochodną funkcji celu, co pozwala na szybkie zbliżenie do minimum. Wyznaczamy ciąg kolejnych przybliżeń, w którym każdy kolejny wyraz jest bliższy szukanego minimum. Podstawą metody jest aproksymacja optymalizowanej funkcji wielomianem stopnia drugiego, wyznaczanym poprzez rozwinięcie funkcji f w szereg Taylora wokół punktu c, co pozwala na przybliżenie wartości funkcji w otoczeniu tego punktu. Jeśli funkcja f jest klasy C^2 (czyli posiada ciągłe pochodne do drugiego rzędu), to możemy zastosować następujące przybliżenie Taylora dla dowolnych x oraz c:

$$f(x) \approx f(c) + f'(c)(x - c) + \frac{1}{2}f''(c)(x - c)^2. \tag{2.9}$$

To przybliżenie pozwala na szybkie oszacowanie punktów, w których funkcja osiąga minimum lub maksimum, poprzez iteracyjne uaktualnianie punktu x w oparciu o wartości pochodnych.

Podstawiając $x = x + h$, gdzie h jest małym krokiem oraz $c = x$, możemy zapisać rozwinięcie Taylora w następujący sposób:

$$f(x + h) \approx f(x) + f'(x)h + \frac{1}{2}f''(x)h^2. \tag{2.10}$$

Krok h powinien być dobrany tak, aby $x + h$ należało do przedziału $[a, b]$.

2.3.2 Iteracyjna procedura metody Newtona

Niech $x^{(i)}$ będzie przybliżeniem rzeczywistego minimum, uzyskanym w i-tej iteracji metody Newtona. Zgodnie ze wzorem (2.10) możemy zapisać przybliżenie funkcji f w punkcie $x^{(i)} + h$ w następujący sposób:

$$f(x^{(i)} + h) \approx f(x^{(i)}) + f'(x^{(i)})h + \frac{1}{2}f''(x^{(i)})h^2 \tag{2.11}$$

Iteracyjna procedura rozpoczyna się od wyboru punktu początkowego $x^{(0)}$, który jest pewnym przybliżeniem rozwiązania. W każdej iteracji korygujemy długość kroku h, aby uzyskać lepsze przybliżenie minimum funkcji. Łatwo zauważyć, że prawa strona zależności (2.11) jest funkcją kwadratową zmiennej h. Przy założeniu, że $f''(x^{(i)}) > 0$, to funkcja kwadratowa posiada minimum. Możemy zatem przybliżyć minimum naszej funkcji celu przez minimum tej funkcji kwadratowej. Wykorzystując własności funkcji kwadratowej, możemy obliczyć wartość h jako współrzędną wierzchołka paraboli.
Wzór na h jest dany przez:

$$h = -\frac{a}{2b}, \tag{2.12}$$

gdzie a jest współczynnikiem przy h^2, a b przy h w zależności (2.11). Zatem mamy:

$$x^{(i+1)} = x^{(i)} + h = x^{(i)} - \frac{f'(x^{(i)})}{2 \cdot \frac{1}{2} f''(x^{(i)})} = x^{(i)} - \frac{f'(x^{(i)})}{f''(x^{(i)})}. \tag{2.13}$$

Proces iteracyjny kontynuujemy do momentu, gdy różnica między kolejnymi przybliżeniami spełnia założoną dokładność ε, czyli:

$$\left| x^{(i+1)} - x^{(i)} \right| < \varepsilon. \tag{2.14}$$

Powyższy warunek stopu nie gwarantuje dokładności przybliżenia rozwiązania optymalnego. Mówi on jedynie, że korekta otrzymanego rozwiązania w stosunku do poprzedniego jest mała. Metoda Newtona nie korzysta z informacji o wartości badanej funkcji. Punkt, do którego dąży ciąg kolejnych przybliżeń $x^{(i)}$, to miejsce zerowe pochodnej funkcji f. W szczególności oznacza to, że algorytm Newtona może przybliżać maksimum lub punkt przegięcia badanej funkcji. Zaletą algorytmu Newtona jest jego szybka zbieżność. W metodzie Newtona kluczową rolę odgrywa dobry wybór punktu początkowego. Dla źle wybranego punktu algorytm może nie być zbieżny.

2.3.3 Reguła Armijo

Algorytm Newtona dąży do jak najszybszego osiągnięcia optymalnego punktu funkcji, co jednak może prowadzić do problemów ze stabilnością. Dzieje się tak, ponieważ w niektórych przypadkach krok do kolejnego przybliżenia jest zbyt duży, co skutkuje rozbieżnością algorytmu.

Reguła Armijo podaje zasadę, że długość kroku iteracyjnego d optymalizacji jest dostosowywana tak, aby wartość funkcji celu f zmniejszała się w sposób kontrolowany. Niech dana będzie unimodalna i różniczkowalna funkcja $f : \mathbb{R} \to \mathbb{R}$ minimalizowana na przedziale $[a, b]$. Jeżeli $x_0 \in [a, b]$ jest punktem, w którym $f'(x_0) \neq 0$ i $\alpha \in (0, 1)$, wtedy warunek Armijo definiuje krok d, który spełnia nierówność:

$$f(x_0 + d) \leq f(x_0) + \alpha d f'(x_0), \tag{2.15}$$

gdzie:

- $f(x_0)$ jest wartością funkcji celu f w aktualnym punktem iteracji,
- $f'(x_0)$ jest wartością pochodnej funkcji celu f w punkcie x_0,
- α jest stałą kontrolującą stopień zmniejszania wartości funkcji.

Warto zauważyć, że warunek Armijo zapewnia istnienie właściwego kroku d lecz nie wskazuje metody jego wyznaczenia. Reguła Armijo wymusza, aby krok d nie był zbyt duży, co pozwala na stabilną zbieżność algorytmu Newtona. Jeżeli warunek Armijo nie jest spełniony dla początkowej wartości d, zmniejsza się ją w ustalonym stosunku, np. przez pomnożenie przez pewną stałą $\beta \in (0, 1)$, aż do momentu spełnienia warunku.

2.3.4 Szczegóły algorytmu Newtona-Armijo

Metoda Newtona-Armijo wprowadza dwa główne dostosowania do standardowej metody Newtona:

1. **Początkowa długość kroku w każdej iteracji**:
   Jeżeli druga pochodna funkcji f w bieżącym punkcie przybliżenia $x^{(i)}$ jest dodatnia, to początkowa wartość kroku d jest wyznaczana na podstawie warunku Armijo, zgodnie z algorytmem Newtona. Jeśli druga pochodna nie jest dodatnia (co może wskazywać na maksimum lub punkt przegięcia), wówczas algorytm przyjmuje długość kroku d równą 1, w kierunku spadku wartości funkcji, aby uniknąć rozbieżności. W przypadku funkcji jednej zmiennej kierunek wzrostu lub spadku wartości funkcji określany jest przez znak pochodnej w punkcie x.

2. **Kryterium akceptacji długości kroku**:
   Procedura Newtona-Armijo, oprócz informacji o pochodnej badanej funkcji, wykorzystuje również jej wartości. W każdej iteracji wartość funkcji w bieżącym punkcie jest sprawdzana pod kątem spełnienia warunku Armijo. Mianowicie, algorytm akceptuje krok d tylko wtedy, gdy spełnione jest kryterium:

   $$f(x^{(i)} + d) \leq f(x^{(i)}) + \alpha d f'(x^{(i)}), \tag{2.16}$$

   Jeżeli warunek Armijo nie jest spełniony, długość kroku d jest iteracyjnie zmniejszana, najczęściej poprzez pomnożenie d przez stałą $\beta \in (0, 1)$, aż warunek zostanie spełniony. Wprowadzone modyfikacje algorytmu Newtona-Armijo sprawiają, że jest on bardziej stabilny niż standardowy algorytm Newtona i zbiega do poszukiwanego minimum z każdego punktu początkowego.

2.3.5 Przebieg algorytmu Newtona-Armijo

1. **Inicjalizacja**: Ustal początkowe przybliżenie $x^{(0)}$ oraz wybierz parametry α i β do warunku Armijo. Zdefiniuj tolerancję zbieżności ε.

2. **Krok iteracyjny**: W każdej iteracji i, gdy $f(x^{(i)}) > 0$:

   - Oblicz krok Newtona przy użyciu:

   $$h = -\frac{f'(x^{(i)})}{f''(x^{(i)})}.$$

3. **Ustawienie początkowej długości kroku**:

   - Dla $d = h$ ustal punkt próbny $x^{(i+1)} = x^{(i)} + d$.

4. **Sprawdzenie warunku Armijo**:

   - Sprawdź, czy długość kroku d spełnia warunek Armijo:

   $$f(x^{(i)} + d) \leq f(x^{(i)}) + \alpha d f'(x^{(i)}).$$

   - Jeśli warunek jest spełniony, zaakceptuj $x^{(i+1)} = x^{(i)} + d$ jako kolejne przybliżenie.
   - Jeśli nie, zmniejsz d przez pomnożenie przez β i ponownie sprawdź warunek. Powtarzaj, aż warunek zostanie spełniony.

5. **Sprawdzenie zbieżności**: Iteracja jest kontynuowana, aż różnica między kolejnymi przybliżeniami będzie mniejsza niż ε, czyli:

   $$|x^{(i+1)} - x^{(i)}| < \varepsilon.$$

6. **Zakończenie**: Algorytm kończy działanie, gdy spełnione zostanie kryterium zbieżności, zwracając przybliżone minimum funkcji.

2.3.6 Implementacja kodu w języku R

```
# Zaimportowanie biblioteki do liczenia pochodnych
install.packages("Deriv")
library(Deriv)

# Pierwsza pochodna funkcji f
f_prime = Deriv(f)

# Druga pochodna funkcji f
f_double_prime = Deriv(f_prime)

newton_armijo <- function(
    f, f_prime, f_double_prime, x0, alpha, beta, epsilon, max_iter
) {

  # Inicjalizacja
  x_i <- x0
  i <- 0
  function_calls <- 0

  while (i < max_iter) {
    # Obliczenie wartośći pierwszej i drugiej pochodnej
    f_prime_i <- f_prime(x_i)
    f_double_prime_i <- f_double_prime(x_i)

    # Krok Newtona, ale z dodatkowym sprawdzeniem drugiej pochodnej
    if (f_double_prime_i > 0) {
      # Klasyczny krok Newtona dla dodatniej drugiej pochodnej
      h <- -f_prime_i / f_double_prime_i
    } else {
      # Krok początkowy na 1 dla niedodatniej drugiej pochodnej
      h <- -sign(f_prime_i) * 1 # Ruch w kierunku spadku funkcji
    }

    # Ustawienie początkowej długości kroku
    d <- h
    x_trial <- x_i + d

    # Obliczenie wartości funkcji celu w punkcie początkowym
    f_x_i <- f(x_i)
```

```
    # Liczenie wywołań funkcji celu
    function_calls <- function_calls + 1

    # Sprawdzenie warunku Armijo
    while (f(x_trial) > f_x_i + alpha * d * f_prime_i) {
      # Jeśli warunek Armijo nie jest spełniony, zmniejsz krok
      d <- d * beta
      x_trial <- x_i + d
      function_calls <- function_calls + 1
    }

    # Aktualizacja punktu
    x_next <- x_i + d

    cat("Iteracja:", i, ", x_i:", x_i, ", h:", h, ", d:", d,
    ", x_next:", x_next, ", Dokładność:", abs(x_next - x_i),
    ", Wywołania funkcji celu:", function_calls, "\n")

    # Sprawdzenie kryterium zbieżności
    if (abs(x_next - x_i) < epsilon) {
      return(list(min_x = x_next, f_min_x = f(x_next)))
    }

    # Przejdź do kolejnej iteracji
    x_i <- x_next
    i <- i + 1
  }

  cat("Osiągnięto maksymalną liczbę iteracji.\n")
  return(x_i)
}
```

2.4 Parametry poszczególnych algorytmów

W tej sekcji opiszemy argumenty jakie przyjmują zaimplementowane przez nas funkcje, realizujące algorytmy zadane w problematyce projektu.

Metoda ekspansji dwupunktowej Boxa-Daviesa-Swanna

- $\boldsymbol{f}$ - funkcja celu, której ekstremum chcemy znaleźć.
- $\boldsymbol{x_0}$ - losowy punkt początkowy z przedziału $[-80, 100]$, wokół którego wyznaczać będziemy wstępny przedział poszukiwań.
- $\boldsymbol{\delta}$ - ustalona z góry długość kroku.

Metoda oparta na interpolacji Lagrange'a

- $\boldsymbol{f}$ - funkcja celu, której ekstremum chcemy znaleźć.
- $\boldsymbol{a}$ - początek przedziału, w obrębie którego algorytm poszukuje ekstremum.
- $\boldsymbol{b}$ - koniec przedziału, w obrębie którego algorytm poszukuje ekstremum.
- $\boldsymbol{\varepsilon}$ - parametr dokładności rozwiązania. Określa minimalną długość przedziału $[a, b]$, poniżej której algorytm kończy działanie. Gdy długość przedziału staje się mniejsza niż ε, algorytm zatrzymuje poszukiwanie ekstremum.
- $\boldsymbol{\gamma}$ - dodatkowy parametr stopu. Jeśli zmiana w punkcie d między iteracjami jest mniejsza niż γ, algorytm kończy swoje działanie.

Metoda Newtona z warunkiem Armijo

- $\boldsymbol{f}$ - funkcja celu, której ekstremum chcemy znaleźć.
- $\boldsymbol{f'}$ - pierwsza pochodna funkcji celu f. Służy do obliczenia kierunku, w którym algorytm poszukuje ekstremum.
- $\boldsymbol{f''}$ - druga pochodna funkcji celu f, która jest używana do obliczenia kroków w algorytmie Newtona oraz do oceny wklęsłości funkcji w danym punkcie.
- $\boldsymbol{x_0}$ - punkt początkowy, od którego algorytm rozpoczyna poszukiwanie ekstremum. Jest to średnia arytmetyczna współrzędnych początku i końca przedziału, które zostały wyznaczone przez metodę ekspansji Boxa-Swana-Daviesa.
- $\boldsymbol{\alpha}$ - stała parametryzująca warunek Armijo. Określa minimalny współczynnik spadku funkcji celu, poniżej którego algorytm będzie zmniejszał krok w każdej iteracji.

- $\boldsymbol{\beta}$ - stała parametryzująca warunek Armijo. Określa współczynnik zmniejszania długości kroku w przypadku niespełnienia warunku Armijo.
- $\boldsymbol{\varepsilon}$ - parametr dokładności rozwiązania. Określa minimalną zmianę między dwoma kolejnymi iteracjami, poniżej której algorytm kończy poszukiwania ekstremum.
- `max_iter` - maksymalna liczba iteracji, po której algorytm zatrzymuje działanie, jeśli nie osiągnie zbieżności. Jest to parametr zabezpieczający przed nieskończoną ilością iteracji.

Przyjęte wartości poszczególnych parametrów

Podczas wywoływania algorytmów korzystaliśmy z wyżej opisanych parametrów. Wartości kilku z nich ustaliliśmy odgórnie, co prezentujemy w poniższej tabeli.

Parametr	**Wartość**
δ	$0.3; 0.6; 0.9$
ε	10^{-6}
γ	10^{-8}
α	0.1
β	0.5
`max_iter`	1000

Pozwoli nam to ujednolicić badanie zachowań poszczególnych algorytmów na zmiany w długościach kroku.

Rozdział 3

Wyniki i wnioski

3.1 Omówienie wyników

- Gdy korzystając z metody ekspansji Boxa-Daviesa-Swana dla minimum, losowo wybrany punkt znajdował się akurat w pobliżu któregoś z nich, to liczba wywołań funkcji celu zaczynała spadać. Analogiczna zależność występowała w przypadku maksimum.
- Im lepiej dobrany punkt początkowy, tym metoda Newtona z warunkiem Armijo szybciej trafiała na ekstremum.
- Wyraźne zwiększenie długości kroku δ powodowało zmniejszenie średniej długości przedziału $[a, b]$ oraz średniej liczby wywołań funkcji celu f.
- Oba algorytmy trafiały częściej na ekstrema lokalne niż na ekstremum globalne.

3.2 Wnioski

- Bezgradientowa metoda oparta na interpolacji Lagrange'a jest wolniej zbieżna niż gradientowa metoda Newtona z warunkiem Armijo, gdyż charakteryzuje się prostotą i korzysta jedynie ze znajomości funkcji celu.
- Minusem algorytmów bezgradientowych dla skomplikowanych funkcji celu z kilkoma ekstremami jest zatrzymywanie się często na tych lokalnych, zamiast na globalnym.
- Jeśli druga pochodna $f''(x)$ jest bliska zeru, wyrażenie $h = -\frac{f'(x)}{f''(x)}$ staje się bardzo wrażliwe na zmiany, co może prowadzić do dużych, nieoptymalnych kroków, które prowadzą do rozbieżności algorytmu Newtona nawet mimo zastowania reguły Armijo dla $\alpha = 0.1$ oraz $\beta = 0.5$.
- Zwiększenie albo zmniejszenie wartości parametrów ε oraz γ w metodzie opartej na interpolacji Lagrange'a nawet o jeden rząd wielkości może znacząco wpłynąć na wynik końcowy.
- Rygorystyczne dobieranie wartości ε oraz γ może prowadzić do nieznalezienia minimum w obrębie danego przedziału.

Rozdział 4

Spis wykorzystanych funkcji i bibliotek

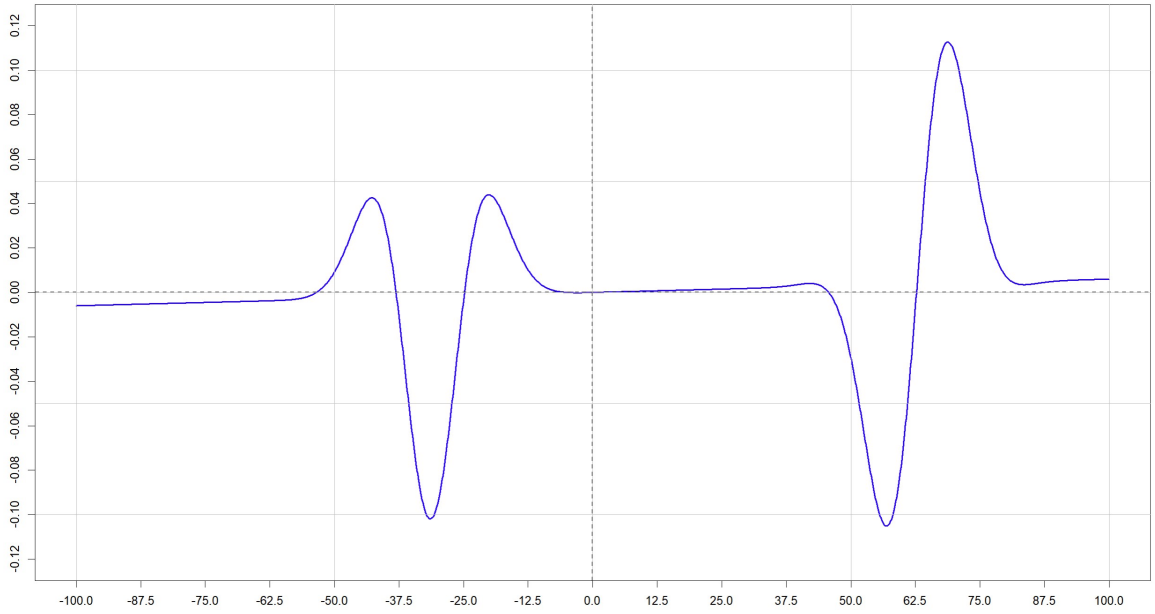
Nazwa funkcji	**Opis działania**	**Biblioteka**
`seq()`	Generuje sekwencję liczb w zadanym przedziale z określonym krokiem.	`base`
`plot()`	Tworzy wykres danych na podstawie dostarczonych argumentów.	`graphics`
`abline()`	Dodaje prostą linię do istniejącego wykresu na podstawie współczynnika nachylenia lub współrzędnych.	`graphics`
`grid()`	Dodaje siatkę na istniejącym wykresie.	`graphics`
`axis()`	Rysuje osie z dostosowanymi etykietami i skalą.	`graphics`
`cat()`	Wyświetla komunikaty na konsoli w trakcie wykonywania programu.	`base`
`abs()`	Oblicza wartość bezwzględną liczby.	`base`
`stop()`	Zatrzymuje wykonywanie programu i wyświetla komunikat błędu.	`base`
`sign()`	Zwraca znak liczby, tzn. -1 dla liczb ujemnych, 0 dla zera i 1 dla liczb dodatnich.	`base`
`is.finite()`	Sprawdza, czy wartość jest liczbą skończoną (nie NaN, nie nieskończoność).	`base`
`list()`	Służy do tworzenia listy.	`base`
`range()`	Zwraca wektor zawierający wartość minimalną i maksymalną wszystkich podanych argumentów.	`base`
`Machine()`	Zwraca informacje o charakterystykach liczbowych maszyny, na której działa R np. o precyzji maszyny.	`base`

Tabela 4.1: Tabela funkcji wbudowanych w R

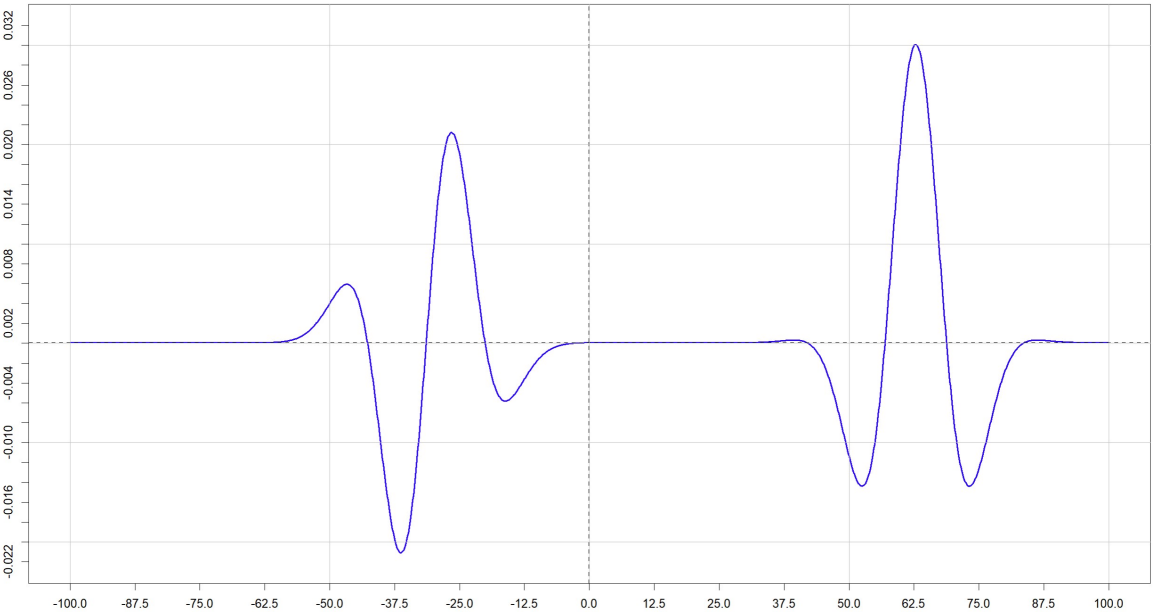
Rozdział 5

Dodatkowe wykresy

Poniżej przedstawiamy wykresy pierwszej i drugiej pochodnej funkcji celu. Pierwszy wykres wskazuje punkty, w których znajdują się ekstrema. Drugi dostarcza informacji o wypukłości, wklęsłości oraz punktach przegięcia.



Rysunek 5.1: Wykres f' dla D_f



Rysunek 5.2: Wykres f'' dla D_f

Bibliografia

[1] Krzysztof Pupka. *Optymalizacja nieliniowa w R*. Materiały pomocnicze dla studentów, Rzeszów, 2023.