222800-1358 Nieklasyczne Metody Optymalizacji **Letni 2022/2023**

Ćwiczenia 1: Metoda Najszybszego Spadku

Daniel Kaszyński *21-28 Lutego 2023 r.*

Kwestie organizacyjne

- **Prowadzący ćwiczenia**: mgr. Daniel Kaszyński, dkaszy[@]sgh.waw.pl
- **Konsultacje**: wtorki 14:00-15:00, G-213.
- **Ocena z ćwiczeń** z NMO jest 0/1 (zal., nie zal.).
- **Zaliczenie ćwiczeń**:
  - w oparciu o przygotowany raport z terminem oddania: 7 czerwca 2023 r.)
  - raport przygotowywany w grupach max 4 osobowych
  - przed rozpoczęciem prac nad raportem, proszę o wiadomość z: 1) tytułem/tematyką raportu, 2) zespołem który będzie opracowywał raport; tematy podlegają akceptacji!
  - tematyka raportu: dla zadanego problemy optymalizacyjnego (np. problem transportowy, plecakowy, optymalizacja strategii inwestycyjnej, propagacja wsteczna dla sieci neuronowych), wykorzystujemy jedną z metaheurystyk/metod optymalizacyjnych.
  - załącznikiem do raportu jest kod źródłowy w R (metaheurystyka opracowana samodzielnie – nie korzystamy z gotowych solwerów)
  - raport z kodem źródłowym wysyłacie Państwo na email – wiadomości te później są archiwizowane
- **Literatura**:
  - Chong, E.K. and Zak, S.H., 2004. *An introduction to optimization*. John Wiley & Sons.
  - Kochenderfer, M.J. and Wheeler, T.A., 2019. *Algorithms for optimization*. Mit Press.
  - Dréo, J., Pétrowski, A., Siarry, P. and Taillard, E., 2006. *Metaheuristics for hard optimization: methods and case studies*. Springer Science & Business Media.
  - Cortez, P., 2014. *Modern optimization with R*. New York: Springer.
  - Sydsæter, K., Hammond, P., Seierstad, A. and Strom, A., 2008. *Further mathematics for economic analysis*. Pearson education.

1.1 Komentarze wstępne

Poniżej przedstawiono podstawowe informacje, które posłużą nam w dalszej części ćwiczeń.

1.1.1 Podstawowe pojęcia

Pochodna funkcji

Pochodną funkcji $f : \mathbb{R} \to \mathbb{R}$ nazwiemy funkcję:

$$f'(x) = \frac{df}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} = \lim_{h \to 0} \frac{f(x) - f(x-h)}{h}$$

Ciągłość funkcji f jest warunkiem koniecznym różniczkowalności!

W analogiczny sposób można wyprowadzić wzór na pochodną dowolnego rzędy:

$$f''(x) = \frac{d^2 f}{dx^2} = (f'(x))' = \lim_{h \to 0} \frac{f'(x+h) - f'(x)}{h} = \lim_{h \to 0} \frac{f(x+2h) - 2f(x+h) + f(x)}{h^2}$$

Twierdzenie Taylora

Niech $f : \mathbb{D} \subset \mathbb{R} \to \mathbb{R}$ oraz $f \in C^{n+1}$ w każdym punkcie odcinka $[x, x+h]$. Wówczas dla pewnego θ mamy:

$$f(x+h) = f(x) + \sum_{k=1}^{n} \frac{1}{k!} f^{(k)}(x) h^k + \frac{1}{(n+1)!} f^{(n+1)}(x + \theta h) h^{(n+1)}$$

$$f(x+h) \approx f(x) + \sum_{k=1}^{n} \frac{1}{k!} f^{(k)}(x) h^k$$

Rozwinięcie funkcji w szereg Taylora 1 i 2 stopnia:

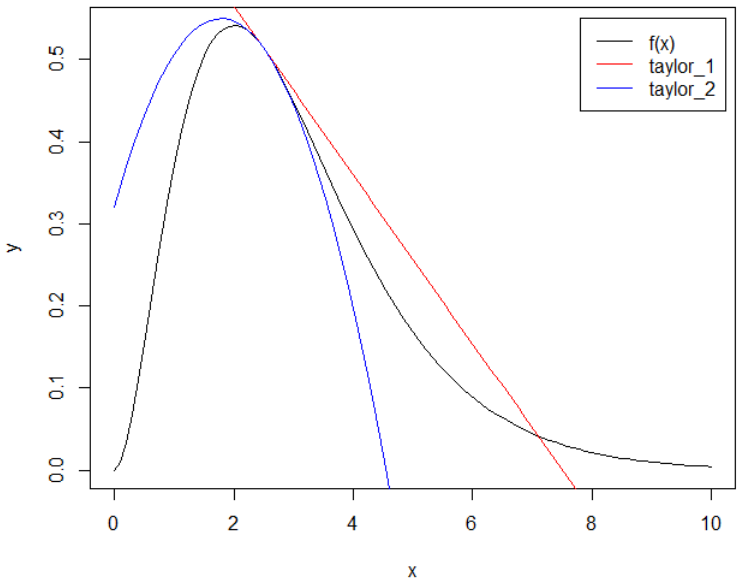
$$f(x+h) \approx f(x) + f'(x)h$$

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

Przykładowa funkcja

```
# Dane wejsciowe
f  <- function(x) x^2/exp(x)
x0 <- 2.5
h_seq <- seq(0, 10, length = 100)

# Pochodne numeryczne
d1f <- function(f, x, h = 10^-6) (f(x+h)-f(x))/h
d2f <- function(f, x, h = 10^-6) (f(x+2*h)-2*f(x+h)+f(x))/h^2

# Aproksymacja Taylora funkcji f wokol x0
taylor_1 <- function(f, x, h) f(x)+d1f(f, x)*(h-x)
taylor_2 <- function(f, x, h) f(x)+d1f(f, x)*(h-x)+1/2*d2f(f, x)*(h-x)^2

# Wykresy
plot(h_seq, f(h_seq), type='l', col='black', xlab = 'x', ylab = 'y')
lines(h_seq, taylor_1(f, x0, h_seq), col='red')
lines(h_seq, taylor_2(f, x0, h_seq), col='blue')
legend(7.8, 0.55, legend=c('f(x)', 'taylor_1', 'taylor_2'),
       col=c('black', 'red', 'blue'), lty=1, cex=1)
```

Listing 1: Przykład rozwiniecia funkcji w szereg Taylora



Rysunek 1.1: Przykład: rozwinięcie funkcji w szereg Taylora

Pochodna kierunkowa

Niech $f : \mathbb{D} \subset \mathbb{R}^n \to \mathbb{R}$, $x \in \mathbb{D}$ oraz $h \in \mathbb{R}^n : x + h \in \mathbb{D}$. Pochodną kierunkową funkcji f w punkcie x w kierunku h nazywamy funkcję:

$$\frac{df}{dh}(x) = \lim_{t \to 0} \frac{f(x + th) - f(x)}{t}$$

Pochodna cząstkowa

Niech $f : \mathbb{D} \subset \mathbb{R}^n \to \mathbb{R}$, $x \in \mathbb{D}$ oraz $h \in \mathbb{R}^n : x + h \in \mathbb{D}$. Pochodną cząstkową funkcji f w punkcie x względem zmiennej x_i, $i = 1, 2, ...n$ nazywamy funkcję:

$$\frac{\partial f}{\partial x_i}(x) = \frac{df}{de_i}(x)$$

gdzie e_i jest $i$tym wersorem przestrzeni $\mathbb{R}^n$. Pochodna cząstkowa f względem x_i jest więc pochodną kierunkową f w kierunku $i$tego wersowa, tj. przy $h = e_i$

Gradient funkcji

Niech $f : \mathbb{D} \subset \mathbb{R}^n \to \mathbb{R}$, $x \in \mathbb{D}$. Gradientem funkcji f, jest funkcja $\nabla_f(x) : \mathbb{R}^n \to \mathbb{R}^n$ w punkcie x nazywamy funkcję:

$$\nabla_f(x) = \left[\frac{\partial f}{\partial x_1}(x), \frac{\partial f}{\partial x_2}(x), \ldots, \frac{\partial f}{\partial x_n}(x) \right]$$

Związek między Pochodną Kierunkową a Gradientem?

Jeżeli istnieje gradient funkcji $\nabla_f(\mathbf{x})$ w punkcie $\mathbf{x}$ (co oznacza, że f jest różniczkowalna w $\mathbf{x}$)

$$\nabla_f(\mathbf{x}) = \left[\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right]$$

to pochodna kierunkowa funkcji f w kierunku wektora $\mathbf{h}$ jest równa iloczynowi skalarnemu gradientu funkcji f i wektora $\mathbf{h}$

$$\frac{df}{dh} = \nabla_f(\mathbf{x}|\mathbf{h}) = \nabla_f(\mathbf{x}) \times \mathbf{h}$$

Gradient jako kierunek najszybszego wzrostu wartości funkcji

Niech $|h| = 1$, tj. będzie znormalizowanym wektorem. Wtedy stopa wzrostu wartości funkcji f w punkcie x w kierunku h jest dana przez pochodną kierunkową $\frac{df}{dh}(x)$. Wyznaczmy w takim razie kierunek h, który maksymalizuje stopę wzrostu wartości funkcji f, tj. kierunek maksymalizujący pochodną kierunkową:

$$\frac{df}{dh}(x) = \nabla_f(x)h = |\nabla_f(x)||h|cos(\nabla_f(x), h) = |\nabla_f(x)|cos(\nabla_f(x), h)$$

dla $|\nabla_f(x)| \geqslant 0$ oraz $cos(\nabla_f(x), h) \in [-1, 1]$ stopa wzrostu wartości funkcji f jest największa, gdy $cos(\nabla_f(x), h) = 1$, co implikuje, że h wskazuje ten sam kierunek co $\nabla_f(x)$. Oznacza to, że $h = \frac{\nabla_f(x)}{|\nabla_f(x)|}$.

Gradient jako wektor ortogonalny względem warstwicy funkcji

Niech $f : \mathbb{R}^n \to \mathbb{R}$, $x^* = (x_1^*, \ldots, x_n^*)$ oraz $\nabla_f(x^*) \neq 0$. Niech $r : \mathbb{R} \to \mathbb{R}^n$ taki, że $r(t_0) = x^*$. Wartość funkcji jest stała dla wszystkich punktów z zadanej warstwicy (zgodnie z definicją warstwicy) oraz: $\forall_{t\in\mathbb{R}} f(r(t)) = c$. Wtedy $\frac{d}{dt}(f(r(t)) = \nabla_f(r(t))\frac{dr}{dt}(t) = 0$. W szczególności $\nabla_f(r(t_0))\frac{dr}{dt}(t_0) = 0$. Ponieważ $\frac{dr}{dt}(t_0)$ jest przestrzenią styczną do warstwicy funkcji f w x^*, oznacza to wtedy, że $\nabla_f(x^*)$ jest prostopadła do warstwicy funkcji.

1.1.2 Numeryczne przybliżenie pochodnych funkcji

Z analitycznego punktu widzenia, symboliczne różniczkowanie jest *agnostyczne* ze względu na kierunek perturbacji, t.j.:

$$f'(x) = \underbrace{\lim_{h\to 0} \frac{f(x+h) - f(x)}{h}}_{\text{forward derivative}} = \underbrace{\lim_{h\to 0} \frac{f(x) - f(x-h)}{h}}_{\text{backward derivative}} = \underbrace{\lim_{h\to 0} \frac{f(x+\frac{h}{2}) - f(x-\frac{h}{2})}{h}}_{\text{central derivative}}$$

Jednak z punktu widzenia różnic skończonych, powyższe równanie nie jest prawdziwe:

$$\underbrace{\frac{f(x+h) - f(x)}{h}}_{\text{forward difference}} \neq \underbrace{\frac{f(x) - f(x-h)}{h}}_{\text{backward difference}} \neq \underbrace{\frac{f(x+\frac{h}{2}) - f(x-\frac{h}{2})}{h}}_{\text{central difference}}$$

Forward-* i *backward-difference

Z Twierdzenia Taylora wiemy, że:

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2}h^2 + \frac{f'''(x)}{3!}h^3 + \dots$$

Jesteśmy w stanie przekształcić wyrażenie na:

$$f'(x)h = f(x+h) - f(x) - \frac{f''(x)}{2}h^2 - \frac{f'''(x)}{3!}h^3 - \dots$$

oraz dzieląc przez h uzyskujemy:

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \underbrace{\frac{f''(x)}{2}h}_{O(h)} - \frac{f'''(x)}{3!}h^2 - \dots$$

Mamy więc wskazanie, że błąd w *forward-difference* jest rzędu $O(h)$

Dla *backward-difference* mamy:

$$f(x-h) = f(x) - f'(x)h + \frac{f''(x)}{2}h^2 - \frac{f'''(x)}{3!}h^3 + \dots$$

$$f'(x) = \frac{f(x) - f(x-h)}{h} + \underbrace{\frac{f''(x)}{2}h}_{O(h)} - \frac{f'''(x)}{3!}h^2 - \dots$$

a więc również błąd aproksymacji dla *backward-difference* jest rzędu $O(h)$.

Central-difference

W przypadku *Central difference*, błąd jest rzędu $O(h^2)$:

$$f(x+\frac{h}{2}) = f(x) + f'(x)\frac{h}{2} + \frac{f''(x)}{2}\frac{h^2}{4} + \frac{f'''(x)}{3!}\frac{h^3}{8} + \dots$$

$$f(x-\frac{h}{2}) = f(x) - f'(x)\frac{h}{2} + \frac{f''(x)}{2}\frac{h^2}{4} - \frac{f'''(x)}{3!}\frac{h^3}{8} + \dots$$

Odejmując dwie powyższe formuły uzyskujemy:

$$f(x+\frac{h}{2}) - f(x-\frac{h}{2}) = 2f'(x)\frac{h}{2} + \frac{2f'''(x)}{3!}\frac{h^3}{8} + \dots$$

$$f'(x) = \frac{f(x+\frac{h}{2}) - f(x-\frac{h}{2})}{h} - \underbrace{f'''(x)\frac{h^2}{24}}_{O(h^2)} + \dots$$

a więc błąd dla *central-differnece* jest rzędu $O(h^2)$.

Complex-step-difference

Twierdzenie Taylora z przejściem przez dziedzinę zespoloną jest następujące:

$$f(x+ih) = f(x) + f'(x)ih - \frac{f''(x)}{2}h^2 - \frac{f'''(x)}{3!}ih^3 + \dots$$

$$f'(x)ih = f(x+ih) - f(x) + \frac{f''(x)}{2}h^2 + \frac{f'''(x)}{3!}ih^3 + \dots$$

$$f'(x)i = \frac{f(x+ih) - f(x)}{h} + \frac{f''(x)}{2}h + \frac{f'''(x)}{3!}ih^2 + \dots$$

$$f'(x) = Im\left(\frac{f(x+ih) - f(x)}{h}\right) + \underbrace{\frac{f'''(x)}{3!}h^2}_{O(h^2)} + \dots$$

1.1.3 Zagadnienia obliczeń numerycznych

Poniżej przedstawiono wykres wielkość **błędu względnego** dla poszczególnych metod numerycznych przybliżania wartości pochodnej (przykład za [KW19]):

```
if(!require(Deriv)) install.packages(Deriv); #Biblioteka do obl. symbolicznych
f  <- function(x) sin(x^2);
df <- Deriv(f)

# Implementacja funkcji pochodnych numerycznych
diff_forward  <- function(f, x, h=10^-6) (f(x+h)-f(x))/(h);
diff_backward <- function(f, x, h=10^-6) (f(x)-f(x-h))/(h);
diff_central  <- function(f, x, h=10^-6) (f(x+h/2)-f(x-h/2))/(h);
diff_complex  <- function(f, x, h=10^-6) Im(f(x+h*1i))/h;

# Wykresy
h <- exp(log(10)*seq(-20, 0, length.out=500))
plot(h, abs((df(x0) - diff_forward(f, x0, h)) / df(x0)), col = 'blue',
     log = 'xy', type = 'l', ylab = 'Relative Error', ylim = c(10^-16, 1))
lines(h, abs((df(x0) - diff_backward(f, x0, h))/ df(x0)), col = 'green')
lines(h, abs((df(x0) - diff_central(f, x0, h)) / df(x0)), col = 'black')
lines(h, abs((df(x0) - diff_complex(f, x0, h)) / df(x0)), col = 'red')
legend('bottomleft', c("diff_forward", "diff_backward", "diff_central",  "diff_complex"),
       col=c("blue", "green","black", "red"), lty=1)
```

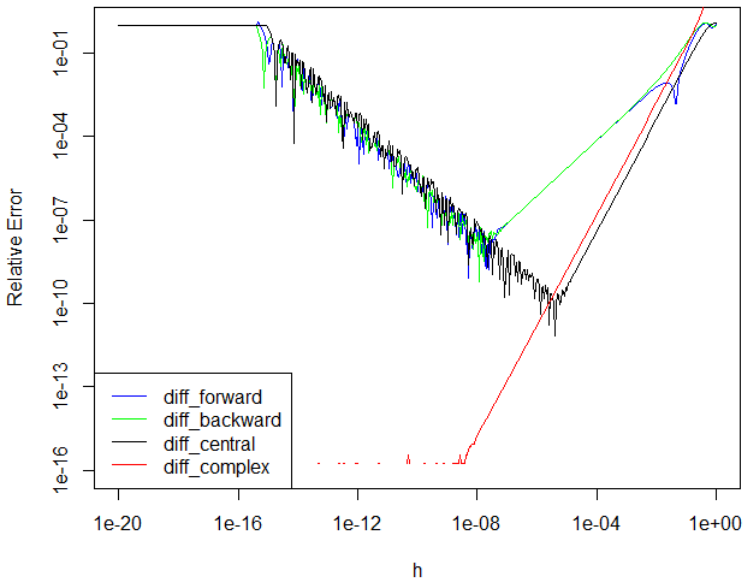
Listing 2: Przykład względnego błędu aproksymacji wartości pochodnej

1.2 Metody lokalne

Częstym podejściem do optymalizacji jest inkrementalne (iteracyjne) poprawianie rozwiązania $\mathbf{x}$, wykonując *kroki*, korzystając z lokalnej informacji, w kierunku minimalizacji funkcji celu. W celu wyboru odpowiedniego kierunku, można wykorzystać twierdzenie Taylora (1 lub 2 rzędu). Metody stosujące takie podejście zbiorczo nazywane są *metodami lokalnymi*. Działanie algorytmu rozpoczyna się od wyboru *rozwiązania początkowego* $\mathbf{x}^{(1)}$.

Schemat działania:

1. Sprawdzenie, czy rozwiązania $\mathbf{x}$ spełnia warunki *terminacji algorytmu* (kryteria terminacji, kryterium stop). Jeżeli te warunki **nie** są spełnione, wykonujemy kolejny krok.



Rysunek 1.2: Przykład względnego błędu aproksymacji wartości pochodnej

2. Określenie kierunku zmiany, spadku (descent direction*$\mathbf{d}^{(k)}$), wykorzystując lokalną informację (np. gradient lub Hesjan funkcji). W niektórych wariantach zakłada się normalizację wektora kroku, tj. $||\mathbf{d}^{(k)}|| = 1$.
3. Ustalenie długości kroku (tzw. learning rate $\alpha^{(k)}$).
4. Wyznaczenie nowego rozwiązania, tj. $\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{d}^{(k)}$

Kryteria terminacji algorytmu: Najczęściej wykorzystuje się 5 głównych kryteriów terminacji działania algorytmów:

- Maksymalna liczba iteracji. Numer iteracji k przekracza limit na iteracje k_{max}, tj. $k > k_{max}$.
- Bezwzględna poprawa wartości funkcji celu. $f(\mathbf{x}^{(k)}) - f(\mathbf{x}^{(k+1)}) < \epsilon_a$
- Względna poprawa wartości funkcji celu. $f(\mathbf{x}^{(k)}) - f(\mathbf{x}^{(k+1)}) < \epsilon_r |f(\mathbf{x}^{(k)})|$
- Wielkość gradientu. $||\nabla_f(\mathbf{x}^{(k+1)})|| < \epsilon_g$
- Długość kroku. $||\mathbf{x}^{(k)} - \mathbf{x}^{(k+1)}|| < \epsilon_g$

1.3 Metody Gradientowe

Intuicyjnym wyborem kierunku $\mathbf{d}^{(k)}$ jest kierunek najszybszego wzrostu / spadku wartości funkcji celu. Podążanie za tym kierunkiem gwarantuje poprawianie wartości funkcji celu (przy założeniu, że funkcja celu jest funkcja ciągłą, oraz długość kroku odpowiednio małe). Kierunkiem najszybszego spadku wartości funkcji jest kierunek przeciwny do gradientu - tzw. antygradient (stąd metody gradientowe).

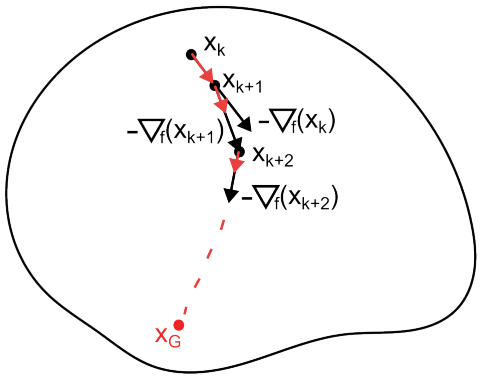
1.4 Metoda spadku gradientowego (Gradient Descent)

Metoda zakłada przemieszczanie się w kierunku antygradientu, przy założeniu stałego parametru skalowania kroku - α.

$$x^{(k+1)} = x^{(k)} - \alpha \nabla_f(x^{(k)})$$

lub w przypadku normalizacji długości kroku:

$$x^{(k+1)} = x^{(k)} - \alpha \frac{\nabla_f(x^{(k)})}{||\nabla_f(x^{(k)})||}$$



Rysunek 1.1: Budowa trajketorii w kierunku anty gradientu

Kod zamieszczony w listingu 3 jest przykładem implementacji funkcji do obliczenia numerycznego gradientu z wykorzystaniem definicji pochodnej centralnej. Alternatywnie na końcu ukazano również załadowanie biblioteki numDeriv, w której istnieją już zaimplementowane funkcje do obliczenia gradientu oraz Hesjanu, które działają szybciej.

```
if(!require(docstring)) library(docstring) # Biblioteka do dokumentacji kodu!

num_grad <- function(f, x, h = 10^-6){
  #' Centralny Gradient Numeryczny
  #'
  #' @description Funkcja do wyznaczania gradientu numerycznego poprzez
  #' wyznaczenie wektora centralnych pochodnych czastkowych.
  #'
  #' @param f funkcja. Funkcja, ktorej gradient wyznaczamy
  #' @param x wektor numeryczny. Punkt, w ktorym wyznaczany jest gradient
  #' numeryczny
  #' @param h skalar. Wartosc skonczonej roznicy
  #'
  #' @usage num_grad(f, x)
  #' @return Wektor pochodnych czastkowych funkcji f.

  n <- length(x)
  g <- rep(NA, n) # prealokacja pamieci pod gradient
  e <- diag(n)

  # obliczenie pochodznych czastkowych
  for(i in 1 : n) g[i] = (f(x+h*e[i,])-f(x-h*e[i,]))/(2*h)
  return(g)
}

```

```r
?num_grad # Mamy dostep do dokumentacji

# Parametry wejsciowe
my_fun <- function(x) 2*x[1]^2 + x[2]^2
c(3,4) -> x0  # Uwaga! Strzalki nie tylko w lewo!

# Wyniki
my_fun(x) # 2*3^2+4^2 = 17 # Sprbuj: sin(x^2);
x0 <- c(-3, -2)
my_fun(x0) # 2*(-3)^2+(-2)^2 = 22
num_grad(my_fun, x0, 10^-6) # zwraca wektor [-12, -4]

# Benchmark 1
if(!require(numDeriv)) library(numDeriv)   # Biblioteka do obliczen numerycznych
grad(my_fun, x0)    # gradient
hessian(my_fun, x0) # hessian

# Benchmark 2
if(!require(Deriv)) install.packages(Deriv); #Biblioteka do obl. symbolicznych
my_fun <- function(x, y) 2*x^2 + y^2
df <- Deriv(my_fun)
cat('f =  ',  deparse(my_fun)[2], '\n')
cat('df = ', deparse(df)[2])
```

Listing 3: Implementacja gradientu w R

Listing 4 ukazuje implementację algorytmu spadku gradientowego, natomiast Listing 5 ukazuje kod do wygenerowania funkcji celu, rozwiązania początkowego oraz wywołanie optymalizacji metodą spadku gradientowego.

```r
gradient_descent <- function(f, x, a = 0.1, K = 100){
  #' GRADIENT DESCENT
  #'
  #' INPUT:
  #'       - f: funkcja celu
  #'       - x: punkt poczatkowy
  #'       - a: learning rate
  #'       - K: maksimum iteracji
  #'
  #' OUTPUT:
  #'       - x_opt: znalezione rozwiazanie
  #'       - f_opt: wartosc funkcji celu w rozwiazaniu
  #'       - x_hist: historia przeszukiwanych rozwiazan
  #'       - f_hist: historia wartosci funkcji celu
  #'       - t_eval: czas trwania dzialania algorytmu

  start_time <- Sys.time()
  results <- list(x_opt = x,
                  f_opt = f(x),
                  x_hist = matrix(NA, nrow = K, ncol = length(x)),
                  f_hist = rep(NA, K),
                  t_eval = NA)

  results$x_hist[1,] <- x
  results$f_hist[1] <- f(x)

  for(k in 2:  K){
    # opis przejscia z punktu x_k do x_k+1
    x_new <- x - a * grad(f, x)

    # sprawdzenie czy nowe rozwiazanie
    # jest dotychczasowo najlepsze
    if(f(x_new) < results$f_opt){
      results$x_opt <- x_new
```

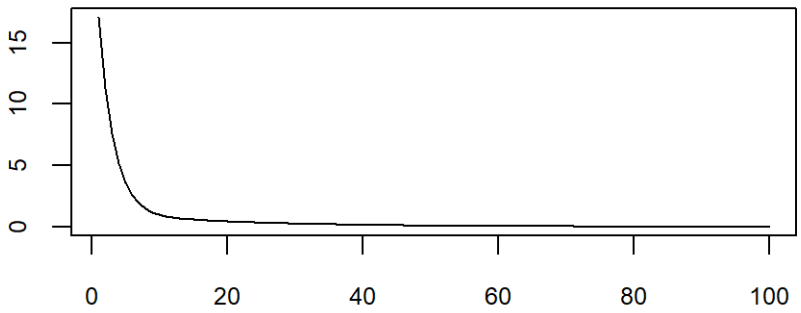
```
      results$f_opt <- f(x_new)
    }

    results$x_hist[k,] <- x_new
    results$f_hist[k] <- f(x_new)

    x <- x_new
  }
  # roznica czasow miedzy zakonczeniem a startem algorytmu
  results$t_eval <- Sys.time() - start_time
  return(results)
}
```

Listing 4: Implementacja algorytmu spadku gradientowego

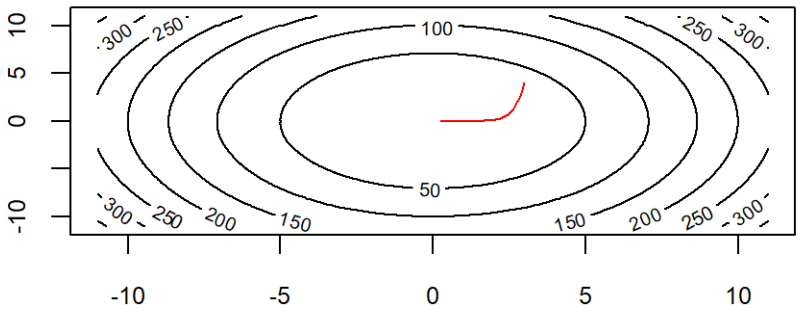
```
# zaladowanie biblioteki i funkcji algorytmu GD
library(numDeriv)
source("grad_descent.R")

# funkcja celu
f = function(x) 1/8*x[1]^2 + x[2]^2
# punkt poczatkowy
x = c(3, 4)
# wywolanie algorytmu
gd = gradient_descent(f, x ,a = 0.1, K = 100)

# wykres wartosci funkcji celu w kolejnych iteracjach
plot(gd$f_hist, type = "l")

# rysunek konturowy przestrzeni rozwiazan i budowana trajektoria
xx = yy = seq(-11, 11, by = 0.1)
ff = function(x1, x2) 2*x1^2 + x2^2
zz = outer(xx,yy, ff)

contour(xx, yy, zz)
lines(gd$x_hist[,1],  gd$x_hist[,2], col = "red")
```

Listing 5: Kod wykorzystany do porównania działania algorytmu GD

Działanie algorytmu pozwoliło na utworzenie na podstawie zgromadzonych danych o wartości funkcji celu i wartościach wektora x potrzebnych wykresów. Rysunek 1.2 przedstawia zmianę wartości funkcji celu w kolejnych iteracjach działania algorytmu. Jak widać po około stu iteracjach znaleźliśmy się w minimum globalnym funkcji. Rysunek 1.3 przedstawia trajektorię budowaną przez algorytm w kolejnych iteracjach z punktu startowego do minimum.



Rysunek 1.2: Wartość funkcji celu w kolejnych iteracjach algorytmu GD



Rysunek 1.3: Trajektoria wektora x w kolejnych iteracjach algorytmu GD

1.5 Metoda najszybszego spadku (Steepest Descent)

Metoda zakłada przemieszczanie się w kierunku antygradientu, przy założeniu *endogenicznego* parametru skalowania kroku – $\alpha^{(k)}$, takiego, że:

$$\alpha^{(k)} = \arg\min_{\alpha} f(x^{(k+1)}) = \arg\min_{\alpha} f(x^{(k)} - \alpha \nabla_f(x^{(k)}))$$

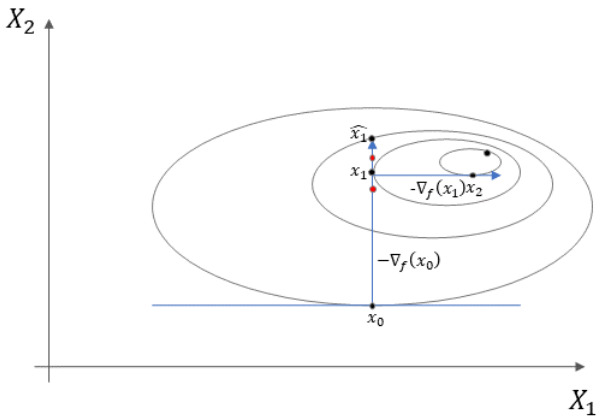
Przykład Rozważmy funkcję $f(\mathbf{x}) = x_1^2 + \frac{1}{2}x_2^2$, oraz punkt startowy $\mathbf{x}^{(1)} = [1, 2]$. Wtedy:

$$\mathbf{x}^{(2)} = [1, 2] - \alpha[2, 2] = [1 - 2\alpha, 2 - 2\alpha]$$

Jakie powinno być α?

$$\frac{df}{d\alpha}(\mathbf{x}^{(2)}) = \frac{d}{d\alpha}\left((1-2\alpha)^2 + \frac{1}{2}(2-2\alpha)^2\right) = \frac{d}{d\alpha}(6\alpha^2 - 8\alpha + 3) = 0$$

Oznacza to, że $\alpha = \frac{3}{4}$.



Rysunek 1.1: Metoda najszybszego spadku

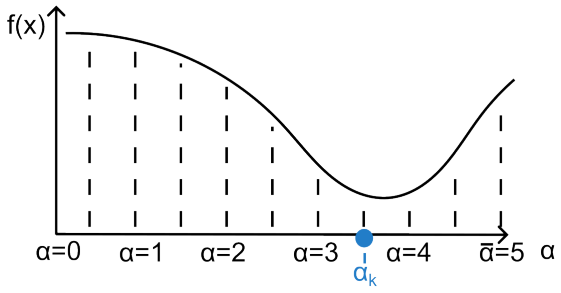
Rozwiązywanie wewnętrznego problemu optymalizacyjnego wiąże się z rozwiązaniem równania:

$$\alpha_k = \underset{\alpha>0}{\operatorname{argmin}} f(x_k - \alpha \nabla_f(x_k)) \tag{1.1}$$

W praktyce dodajemy również parametr $\hat{\alpha}$ będący maksymalną wartością jaką może przyjąć α co zabezpiecza nas przed nieskończenie malejącymi funkcjami oraz przyśpiesza działanie algorytmu. Do rozwiązania problemu wykorzystuje się jedną z metod przeszukiwania liniowego:

1. Grid search
2. Monte Carlo search
3. Golden Section search
4. Fibonacci search
5. Bisection search

Ograniczymy się wyłącznie do omówienia metod Grid i Monte Carlo search ze względu na ich uniwersalność, gdyż nie wymagają żadnych dodatkowych założeń odnośnie kształtu funkcji celu. W przypadku Grid searchu, do wyznaczenia α_k bierzemy długość anty gradientu i mnożymy ją przez wartość ograniczającą $\hat{\alpha}$. Uzyskany odcinek dzielimy na n elementów i w każdym z tych punktów wyznaczamy wartość funkcji celu, wybieramy takie α_k, dla której wartość funkcji celu będzie najmniejsza. Ilustrację tego jak minimalizacja współczynnika α_k odnosi się na funkcję celu w kierunku największego spadku przedstawiono na Rysunku 1.2. Jak w przypadku Grid searchu minimalizujemy funkcję celu na n elementach, to w przypadku metody Monte Carlo z tego zbioru pobieramy wyłącznie p elementów, gdzie $p < n$, i na ich podstawie przeprowadzamy minimalizację funkcji.



Rysunek 1.2: Metoda najszybszego spadku - optymalizacja parametru α

Listing 6 ukazuje implementacje algorytmu największego spadku oraz funkcję do rozwiązania wewnętrznego problemu optymalizacyjnego związanego z przeszukiwaniem parametru α.

```
steepest_descent <- function(f, x, a = 2, grid_size = 100, K = 100){
  #' STEEPEST DESCENT
  #'
  #' INPUT:
  #'       - f: funkcja celu
  #'       - x: punkt poczatkowy
  #'       - a: learning rate
  #'       - grid_size: liczba przeszukiwan dla przeszukiwania alpha
  #'       - K: maksimum iteracji
  #'
  #' OUTPUT:
  #'       - x_opt: znalezione rozwiazanie
```

```
  #'       - f_opt: wartosc funkcji celu w rozwiazaniu
  #'       - x_hist: historia przeszukiwanych rozwiazan
  #'       - f_hist: historia wartosci funkcji celu
  #'       - t_eval: czas trwania dzialania algorytmu

  start_time <- Sys.time()
  results <- list(x_opt = x,
                  f_opt = f(x),
                  x_hist = matrix(NA, nrow = K, ncol = length(x)),
                  f_hist = rep(NA, K),
                  t_eval = NA)

  results$x_hist[1,] <- x
  results$f_hist[1] <- f(x)

  for(k in 2:  K){
    # wywolanie funkcji do znalezienia wartosci parametru alpha
    x_new = line_search(f, x, x-a * grad(f, x), grid_size = grid_size)

    if(f(x_new) < results$f_opt){
      results$x_opt <- x_new
      results$f_opt <- f(x_new)
    }
    results$x_hist[k,] <- x_new
    results$f_hist[k] <- f(x_new)

    x <- x_new
  }
  results$t_eval <- Sys.time() - start_time
  return(results)
}


line_search <- function(f, x0, x1, grid_size = 100){
  xb <- x0
  for(i in 1: grid_size){
    a <- i / grid_size
    xt <- a * x1 + (1 - a)*x0

    if(f(xt) < f(xb)){
      xb <- xt
    }
  }
  return(xb)
}
```

Listing 6: Implementacja algorytmu największego spadku w R

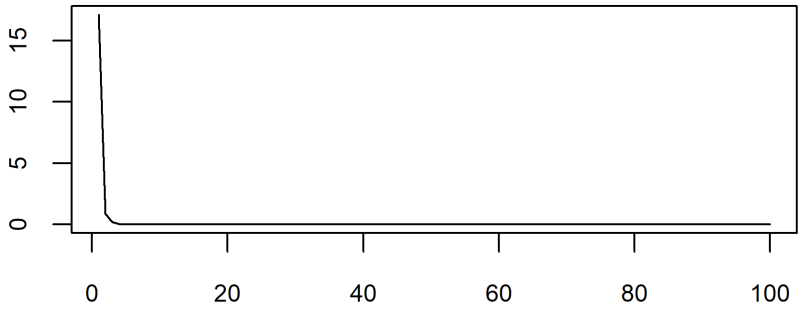
Listing 7 ukazuje fragment kodu wykorzystany do porównania działania algorytmów GD i SD.

```
library(numDeriv)
source("grad_descent.R")
source("steepest_descent.R")

# funkcja celu
f = function(x) 1/8*x[1]^2 + x[2]^2
# wektor poczatkowy
x = c(3, 4)

# wynik dla gd
gd = gradient_descent(f, x ,a = 0.1, K = 100)
# wynik dla sd
sd = steepest_descent(f, x, a = 2, grid_size = 100, K = 100)

```

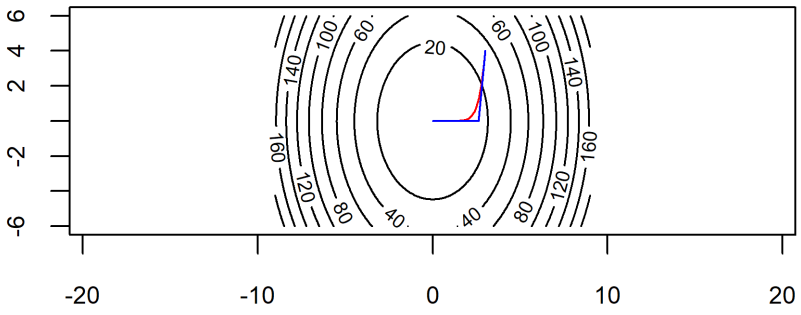
```
yy = seq(-6, 6, length = 400)
xx = seq(-9, 9, length = 400)
ff = function(x1, x2) 2*x1^2 + x2^2
zz = outer(xx,yy, ff)

# spadek wartosci funkcji celu w przypadku SD
plot(sd$f_hist, type = "l")

# porownanie trajektori w gd i sd
contour(xx, yy, zz, asp = F)
lines(gd$x_hist[,1],  gd$x_hist[,2], col = "red")
lines(sd$x_hist[,1],  sd$x_hist[,2], col = "blue")
```

Listing 7: Kod wykorzystany do porównania działania algorytmu GD i SD

Rysunek 1.3 przedstawia wartości funkcji celu w kolejnych iteracjach algorytmu. Widać, że algorytm ten znacznie szybciej zbiega do minimum globalnego funkcji. Natomiast na rysunku 1.4 przedstawiono różnicę w sposobie budowania trajektorii w przestrzeni rozwiązań przez algorytmy GD (czerwony) i SD (niebieski).



Rysunek 1.3: Wartość funkcji celu w kolejnych iteracjach algorytmu SD



Rysunek 1.4: Trajektoria wektora x w kolejnych iteracjach algorytmów GD i SD

Bibliografia

[KW19] M. Kochenderfer and T. Wheeler, "Algorithms for optimization, "*Mit Press*, 2019.