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

Wykład 1: *Klasyczne* Metody Optymalizacji

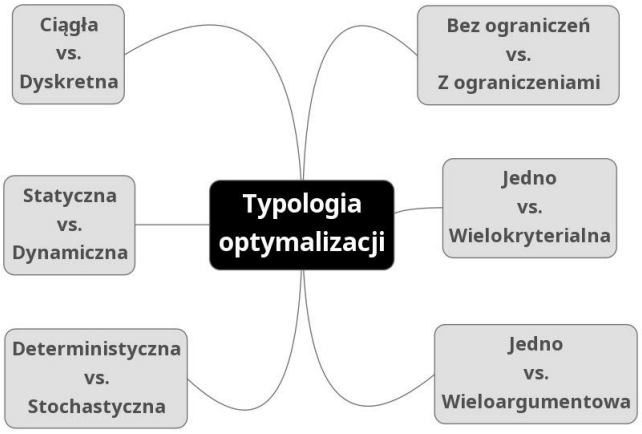
Daniel Kaszyński *26 Lutego 2023 r.*

Kwestie organizacyjne

- **Prowadzący**: mgr. Daniel Kaszyński, dkaszy[@]sgh.waw.pl
- **Konsultacje**: wtorki 14:00-15:00, G-213.
- **Ocena z przedmiotu**:
  - 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 Czemu ustrukturyzowane podejście do optymalizacji?

- Problem Komiwojażera dla 100 miast.
- Funkcja $\text{argmin} f(x, y)$.
- Kształtowanie polityki połowów ryb w danym obszarze.
- Kalibracja głębokiej sieci neuronowej, sterującej autonomicznym samochodem.



Rysunek 1.1: Przykład: Typologia metod optymalizacji

1.2 Komentarze wstępne

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

1.2.1 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:

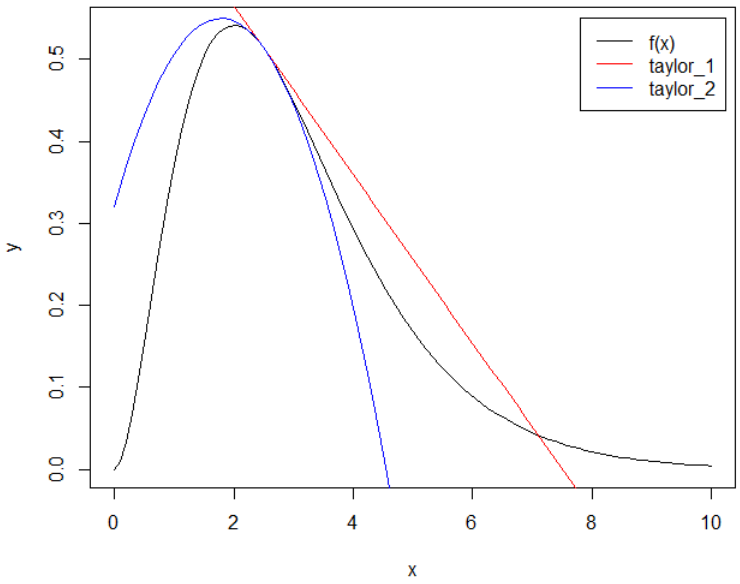
$$f(x+h) \approx f(x) + f'(x)h$$

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

Przykład: Przyjmijmy za przykład funkcję $f(x) = \frac{x^2}{\exp(x)}$. Rozwińmy funkcję f w szereg Taylora 2 stopnia.

```
# 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:

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

1.2.2 Własności Gradientu funkcji

Z uwagi na fakt, że obiekt gradientu jest często wykorzystywany w optymalizacji, przyjrzyjmy się bliżej jego dwóm najważniejszym własnościom.

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.

1.2.3 Optymalizacja analityczna – bez ograniczeń

Podstawowymi sposobami pośrednimi – *implicit* – przeszukiwania przestrzeni są twierdzenia analityczne. Szczególnym przypadkiem optymalizacji analitycznej nieliniowej jest optymalizacja bez warunków ograniczających.

Warunki Pierwszego Rzędu $f : \mathrm{R} \to \mathrm{R}$

Niech $f : \mathbb{D} \subset \mathbb{R} \to \mathbb{R}$, $f \in C^1$. Jeżeli funkcja f posiada ekstremum w punkcie $x \in \mathbb{D}$, to $f'(x) = 0$.

Dowód: Ponieważ funkcja f ,a w punkcie x minumim, wiemy, że istnieje $d > 0$, takie, że dla każdego $0 < |\delta| < d$, mamy $f(x + \delta) > f(x)$, czyli $f(x + \delta) - f(x) > 0$. Dzieląc obie strony przez δ, otrzymujemy:

$$\frac{f(x+\delta) - f(x)}{\delta} > 0 \qquad oraz \qquad \frac{f(x+\delta) - f(x)}{\delta} < 0$$

odpowiednio dla $\delta > 0$ i $\delta < 0$. Odpowiednio przy przejściu granicznym $\delta \to 0^+$ i $\delta \to 0^-$ mamy:

$$\lim_{\delta\to 0^+} \frac{f(x+\delta) - f(x)}{\delta} = f'_+(x) \geqslant 0 \qquad oraz \qquad \lim_{\delta\to 0^-} \frac{f(x+\delta) - f(x)}{\delta} = f'_-(x) \leqslant 0$$

Ponieważ jednak funkcja f jest różniczkowalna, mamy $f'(x) = f'_+(x) = f'_-(x) = 0$. ■

Warunki Drugiego Rzędu $f : \mathrm{R} \to \mathrm{R}$

Niech $f : \mathbb{D} \subset \mathbb{R} \to \mathbb{R}$, $f \in C^n$. Jeżeli dla pewnego $x \in \mathbb{D}$ zachodzi: $f'(x) = 0, f''(x) = 0, \ldots, f^{(n-1)}(x) = 0$, ale $f^{(n)}(x) \neq 0$, to:

- jeśli n jest parzyste, to funkcja f ma w punkcie x ekstremum; jeśli $f^{(n)}(x) > 0$ to jest to minumum, $f^{(n)}(x) < 0$ to jest to maksimum
- jeśli n jest nieparzyste, to funkcja f ma w punkcie x nie ma ekstremum.

Dowód: Ze wzoru Taylora dla pewnego $0 < \theta < 1$ mamy:

$$f(x+h) = \sum_{k=0}^{n-1} \frac{1}{k!} f^{(k)}(x) h^k + \frac{1}{(n)!} f^{(n)}(x+\theta h) h^{(n)}$$

a ponieważ $f'(x) = 0, f''(x) = 0, \ldots, f^{(n-1)}(x) = 0$ to:

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

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

Kiedy n jest parzyste, i $f^{(n)}(x) > 0$, ze względu na parzystość n, mamy h^n ¿ 0.Ze względu na ciągłość funkcji $f^{(n)}$ w punkcie n wiemy, że istnieje $\delta > 0$ taka, że dla każdego $h : 0 < |h| < \delta$ mamy $f^{(n)}(x+h) > 0$, a więc także $f^{(n)}(x+\theta h) > 0$. Oznacza to, że funkcja f ma w tym miejscu minimum. ■

Warunki Pierwszego Rzędu $f : \mathrm{R}^n \to \mathrm{R}$

Niech $f : \mathbb{D} \subset \mathbb{R}^n \to \mathbb{R}$, $f \in C^1$. Jeżeli funkcja f posiada ekstremum w punkcie x, to $\nabla_f(x) = \mathbf{0}$

Dowód: Rozważmy funkcję $g_x(t) = f(x+th)$ oraz $h \in \mathbb{R}^n : x + h \in \mathbb{D}$. Ponieważ f ma ekstremum w x, to g ma ekstremum w $t = 0$, a więc $g'(t) = 0$ dla $t = 0$, co oznacza $g'(t)|_{t=0} = 0$. W konsekwencji:

$$g'(t)|_{t=0} = \lim_{\Delta \to 0} \frac{g(t+\Delta) - g(t)}{\Delta} = \lim_{\Delta \to 0} \frac{f(x+\Delta h) - f(x)}{\Delta} = \frac{df}{dh}(x) = \nabla_f(x) h = \mathbf{0}$$

■

Przykład:
$f(x) = x_1^2 + x_2^2$ $\nabla_f(x) = \left[\frac{\partial f}{\partial x_1}(x), \frac{\partial f}{\partial x_2}(x)\right] = [2x_1, 2x_2] = \mathbf{0} \implies [x_1, x_2] = [0, 0]$

Warunki Drugiego Rzędu $f : \mathrm{R}^n \to \mathrm{R}$

Niech $f : \mathbb{D} \subset \mathbb{R}^n \to \mathbb{R}$, $f \in C^2$. Jeżeli w x^* zachodzi: * $\nabla_f(x^*) = \mathbf{0}$, oraz * $H_f(x^*) > 0$ Wtedy w x^* jest minimum lokalne f.

Dowód: Z twierdzenia Taylora dla $\mathbb{R}^n$ mamy:

$$f(x+h) = f(x) + \nabla_f(x) h + \frac{1}{2} h^T H_f(x) h + o(|h|^2) = f(x) + \frac{1}{2} h^T H_f(x) h + o(|h|^2)$$

czyli $f(x+h) - f(x) = \frac{1}{2} h^T H_f(x) h + o(|h|^2)$. Z twierdzenia Rayleigh'a, wartość formy kwadratowej $h^T H_f(x) h$ możemy ograniczyć z dołu przez $\lambda_{min}|h|^2$:

$$f(x+h) - f(x) = \frac{1}{2} h^T H_f(x) h + o(|h|^2) \geqslant \frac{1}{2} \lambda_{min} |h|^2 + o(|h|^2)$$

Dla dostatecznie małych h mamy więc $f(x+h) - f(x) > 0$. ■

Zwróćmy uwagę, że pierwsze kryterium odnosi się do warunków FOC.

W powyższym zapisie, traktujemy, że $H_f(x)$ to macierz Hessego, taka, że:

$$H_f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Uwaga! Twierdzenie Schwarza $\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial^2 f}{\partial x_j \partial x_i}$

W jaki sposób sprawdzić, czy $H_f(x^*) > 0$?

Twierdzenie Sylwestera.

- Forma jest określona dodatnio wtedy i tylko wtedy, gdy wszystkie minory główne h_i macierzy H_f są dodatnie.
- Forma jest określona ujemnie wtedy i tylko wtedy, gdy wszystkie minory główne parzyste h_{2i} są dodatnie, a wszystkie minory główne nieparzyste h_{2i-1} są ujemne, $i = 1, 2, \ldots$.

Przykład:
$f(x) = x_1^2 + x_2^2$

FOC:

$$\nabla_f(x) = \left[\frac{\partial f}{\partial x_1}(x), \frac{\partial f}{\partial x_2}(x)\right] = [2x_1, 2x_2] = \mathbf{0} \implies [x_1, x_2] = [0, 0]$$

SOC:

$$H_f([0,0]) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} > 0$$

1.2.4 Optymalizacja analityczna – ograniczenia równościowe

Warunki Pierwszego Rzędu $f : \mathrm{R}^n \to \mathrm{R}$, tw. Lagrange

Niech $f : \mathbb{D} \subset \mathbb{R}^n \to \mathbb{R}$, $f \in C^1$. Jeżeli funkcja f posiada w x ekstremum związane $h(x) = 0$, gdzie $h : \mathbb{R}^n \to \mathbb{R}^m$, $h \in C^1$, w punkcie x, to

$$\nabla_f(x) + \lambda^T \mathbf{D}h(x) = \mathbf{0}$$

Dla $h : \mathbb{R}^n \to \mathbb{R}$.

$$\nabla_f(x) + \lambda \nabla_h(x) = \mathbf{0}$$
$$\nabla_f(x) = -\lambda \nabla_h(x)$$

Przykład $f(x) = x_1^2 + x_2^2$ ograniczona przez $[x_1, x_2] : h(x) = x_1^2 + 2x_2^2 - 1 = 0$

Z FOC mamy:

$$[2x_1, 2x_2] + \lambda[2x_1, 4x_2] = 0$$

$$\begin{cases} 2x_1 + \lambda 2x_1 = 0 \Rightarrow x_1(1+\lambda) = 0 \\ 2x_2 + \lambda 4x_2 = 0 \Rightarrow x_2(1+2\lambda) = 0 \end{cases}$$

Co daje nam 4 rozwiązania:

1. $[x_1, x_2] = \left[0, \frac{1}{\sqrt{2}}\right]$, $\lambda = -\frac{1}{2}$
2. $[x_1, x_2] = \left[0, -\frac{1}{\sqrt{2}}\right]$, $\lambda = -\frac{1}{2}$
3. $[x_1, x_2] = [1, 0]$, $\lambda = -1$
4. $[x_1, x_2] = [-1, 0]$, $\lambda = -1$

Warunki Drugiego Rzędu $f : \mathrm{R}^n \to \mathrm{R}$, tw. Lagrange

Niech $f : \mathbb{D} \subset \mathbb{R}^n \to \mathbb{R}$, $h : \mathbb{R}^n \to \mathbb{R}^m$, $h \in C^2$. Niech istnieją takie x oraz λ, że:

1. $\nabla_f(x) + \lambda^T \mathbf{D}h(x) = 0$, oraz
2. $\forall_{z \in T(x), z \neq 0}$ mamy $z^T H_{L(x)} z > 0$

Wtedy punkt x nazywamy minimum funkcji f, związane $h(x) = 0$.

$T(x)$ nazywamy przestrzenią styczną, tj: $T(x) = \{z \in \mathbb{R}^n : z^T \mathbf{D}h(x) = 0\}$. W przypadku, gdy $m = 1$, mamy $T(x) = \{z \in \mathbb{R}^n : z^T \nabla_h(x) = 0\}$

Przykład Rozważmy wcześniejsze 4 punkty.

1. $[x_1, x_2] = \left[0, \frac{1}{\sqrt{2}}\right]$, $\lambda = -\frac{1}{2}$

$$z : z^T \nabla_h(x) = [z_1, z_2][2x_1, 4x_2]^T = [z_1, z_2] \left[0, \frac{4}{\sqrt{2}}\right] = 0$$

$$z_1 0 + z_2 \frac{4}{\sqrt{2}} = 0 \Rightarrow \mathbf{z} = [\alpha, 0]$$

$$[\alpha, 0]^T H_f([x_1, x_2])[\alpha, 0] = [\alpha, 0]^T \begin{bmatrix} 2+2\lambda & 0 \\ 0 & 2+4\lambda \end{bmatrix} [\alpha, 0] = [\alpha, 0]^T \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} [\alpha, 0] = \alpha^2 > 0$$

2. $[x_1, x_2] = \left[0, -\frac{1}{\sqrt{2}}\right]$, $\lambda = -\frac{1}{2}$
3. $[x_1, x_2] = [1, 0]$, $\lambda = -1$

$$z : z^T \nabla_h(x) = [z_1, z_2][2x_1, 4x_2]^T = [z_1, z_2]\,[1, 0] = 0$$

$$z_1 1 + z_2 0 = 0 \Rightarrow \mathbf{z} = [0, \alpha]$$

$$[0, \alpha]^T H_f([x_1, x_2])[0, \alpha] = [0, \alpha]^T \begin{bmatrix} 2+2\lambda & 0 \\ 0 & 4+\lambda \end{bmatrix} [0, \alpha] = [0, \alpha]^T \begin{bmatrix} 0 & 0 \\ 0 & -2 \end{bmatrix} [0, \alpha] = -2\alpha^2 < 0$$

4. $[x_1, x_2] = [-1, 0]$, $\lambda = -1$

1.2.5 Optymalizacja analityczna – ograniczenia nierównościowe

Warunki Pierwszego Rzędu $f : \mathrm{R}^n \to \mathrm{R}$, tw. Karusha-Kuhna-Tuckera

Niech $f : \mathbb{D} \subset \mathbb{R}^n \to \mathbb{R}$, $f \in C^1$ będzie funkcją celu, oraz $h : \mathbb{R}^n \to \mathbb{R}^m$, $h \in C^1$ oraz $g : \mathbb{R}^n \to \mathbb{R}^p$, $g \in C^1$ będą ograniczeniami. Jeżeli x jest ekstremum, to istnieje taki zestaw $\lambda = (\lambda_1, \ldots, \lambda_n)$ oraz $\mu = (\mu_1, \ldots, \mu_n)$ że:

1. **Stationarity condition**: $\nabla_f(x) + \sum_{i=1}^m \lambda_i \nabla_{h_i(x)} + \sum_{i=1}^p \mu_i \nabla_{g_i(x)} = 0$
2. **Primal feasibility**: $\forall_{i=1,\ldots,m} h_i(x) = 0$ and $\forall_{i=1,\ldots,p} g_i(x) \leqslant 0$
3. **Dual feasibility**: $\forall_{i=1,\ldots,p} \mu_i \geqslant 0$ ¡- for min
4. **Complementary slackness**: $\forall_{i=1,\ldots,p} \mu_i g_i(x) = 0$

Przykład

$f(x) = x_1^2 + x_2^2$

ograniczona przez $[x_1, x_2] : g(x) = x_1^2 + 2x_2^2 - 1 \leqslant 0$

Z FOC mamy:

$$[2x_1, 2x_2] + \mu[2x_1, 4x_2] = 0$$

$$\begin{cases} 2x_1 + \mu 2x_1 = 0 \Rightarrow x_1(1+\lambda) = 0 \\ 2x_2 + \mu 4x_2 = 0 \Rightarrow x_2(1+2\lambda) = 0 \end{cases}$$

Co daje nam 4 rozwiązania:

1. $[x_1, x_2] = \left[0, \frac{1}{\sqrt{2}}\right]$, $\mu = -\frac{1}{2}$ **Dual feasibility**
2. $[x_1, x_2] = \left[0, -\frac{1}{\sqrt{2}}\right]$, $\mu = -\frac{1}{2}$ **Dual feasibility**
3. $[x_1, x_2] = [1, 0]$, $\mu = -1$ **Dual feasibility**
4. $[x_1, x_2] = [-1, 0]$, $\mu = -1$ **Dual feasibility**
5. $[x_1, x_2] = [0, 0]$, $\mu = 0$
6.

Warunki Drugiego Rzędu $f : \mathrm{R}^n \to \mathrm{R}$, tw. Karusha-Kuhna-Tuckera

Niech $f : \mathbb{D} \subset \mathbb{R}^n \to \mathbb{R}$, $h : \mathbb{R}^n \to \mathbb{R}^m$, $h \in C^2$, $g : \mathbb{R}^n \to \mathbb{R}^p$, $g \in C^2$. Niech istnieją takie x, λ oraz μ, że: 1. $\nabla_f(x) + \lambda^T \mathbf{D}h(x) + \mu^T \mathbf{D}g(x) = 0$, oraz 2. $\forall_{z \in T(x), z \neq 0}$ mamy $z^T H_{L(x)} z > 0$

Wtedy punkt x nazywamy minimum funkcji f, związane $h(x) = 0$.

$T(x)$ nazywamy przestrzenią styczną, tj: $T(x) = \{z \in \mathbb{R}^n : z^T \mathbf{D}h(x) = 0\}$. W przypadku, gdy $m = 1$, mamy $T(x) = \{z \in \mathbb{R}^n : z^T \nabla_h(x) = 0\}$

1.2.6 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:

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

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

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

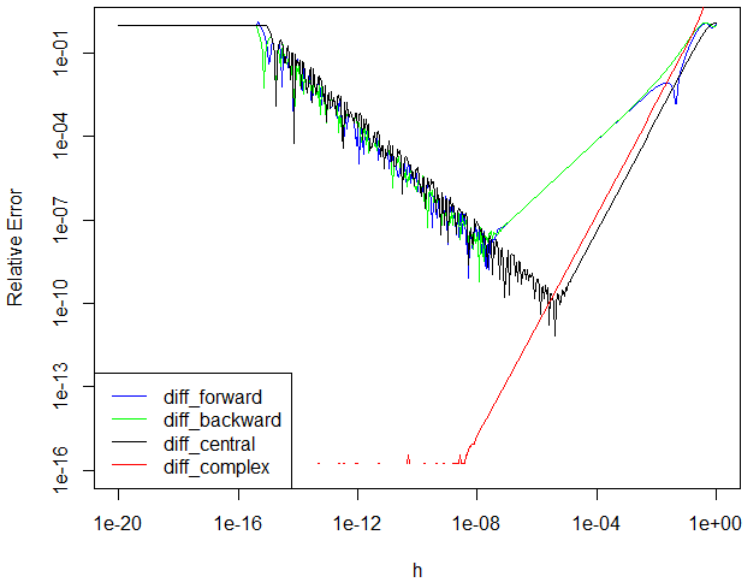
$$f'(x) = Im\left(\frac{f(x+ih) - f(x)}{h}\right) + \underbrace{\frac{f'''(x)}{3!}h^2}_{O(h^2)} + \dots$$

1.2.7 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



Rysunek 1.2: Przykład względnego błędu aproksymacji wartości pochodnej

1.3 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.
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. $||\mathbf{x}^{(k)} - \mathbf{x}^{(k+1)}|| < \epsilon_g$

1.4 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).

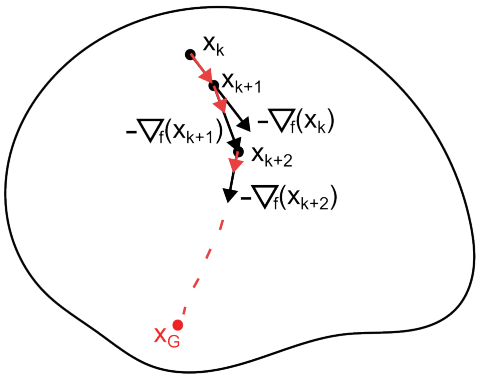
1.5 Metoda spadku gradientowego (Gradient Descent)

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.

```r
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)
}

?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
  #'
```

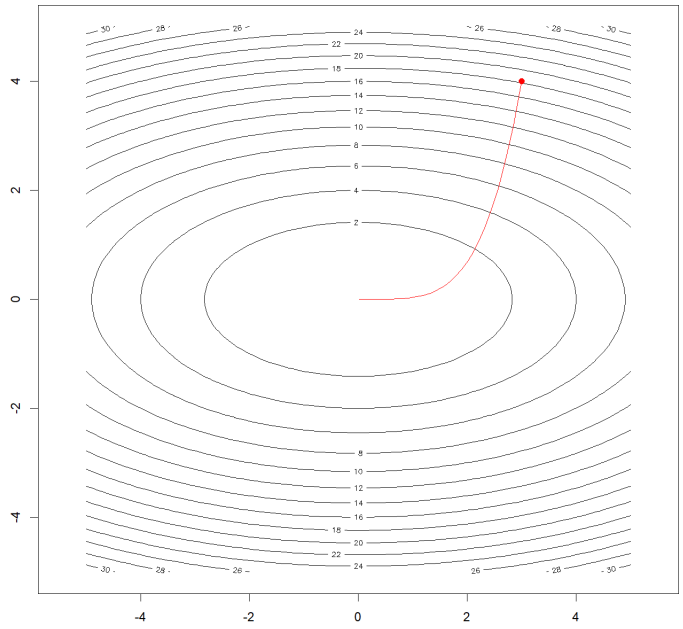
```
  #' OUTPUT:
  #'       - x_opt: znalezione rozwiazanie
  #'       - f_opt: wartosc funkcji celu w rozwiazaniu
  #'       - x_hist: historia przeszukiwanych rozwiazan
  #'       - f_hist: historia wartosci funkcji celu
  #'       - t_eval: czas trwania dzialania algorytmu

  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")
```

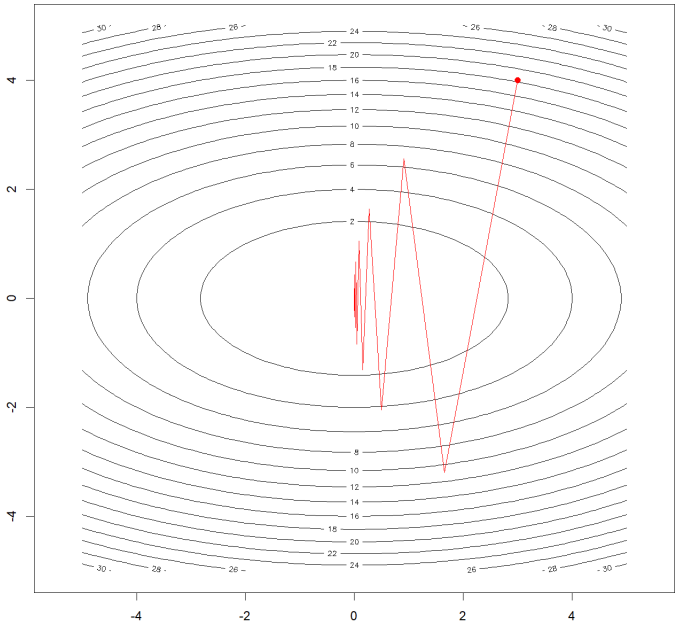
Listing 5: Kod wykorzystany do porównania działania algorytmu GD

Działanie algorytmu pozwoliło na utworzenie na podstawie zgromadzonych danych o wartości funkcji celu i wartościach wektora x potrzebnych wykresów. Rysunek 1.3 przedstawia trajektorię budowaną przez algorytm w kolejnych iteracjach z punktu startowego do minimum.



Rysunek 1.2: Trajektoria wektora x w kolejnych iteracjach algorytmu GD

Z racji tego, że parametr α dobieramy ręcznie, i jest on stały przez cały cykl optymalizacji – możemy mieć problem ze złym jego doborem (zbyt mały, lub zbyt duży). Poniżej pokazano przykład, sytuacji, w której parametr α został dobrany na zbyt wysokim poziomie (mamy problem oscylacji w okół ekstremum).



Rysunek 1.3: Trajektoria wektora x w kolejnych iteracjach algorytmu GD

1.6 Metoda najszybszego spadku (Steepest Descent)

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

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

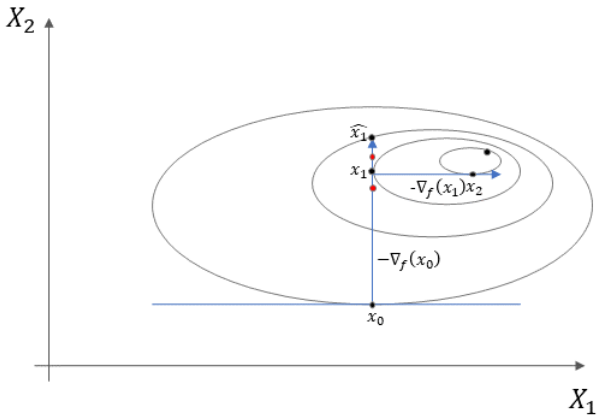
Przykład Rozważmy funkcję $f(\mathbf{x}) = x_1^2 + \frac{1}{2}x_2^2$, oraz punkt startowy $\mathbf{x}^{(1)} = [1, 2]$. Wtedy:

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

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

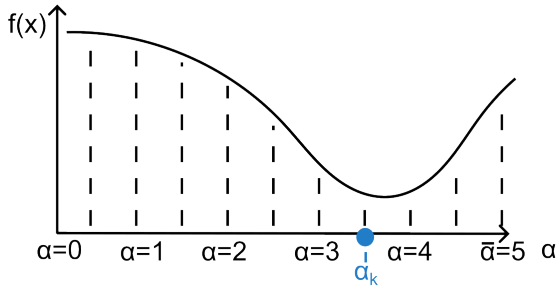
Rozwiązywanie wewnętrznego problemu optymalizacyjnego wiąże się z rozwiązaniem równania:

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

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)
}
```

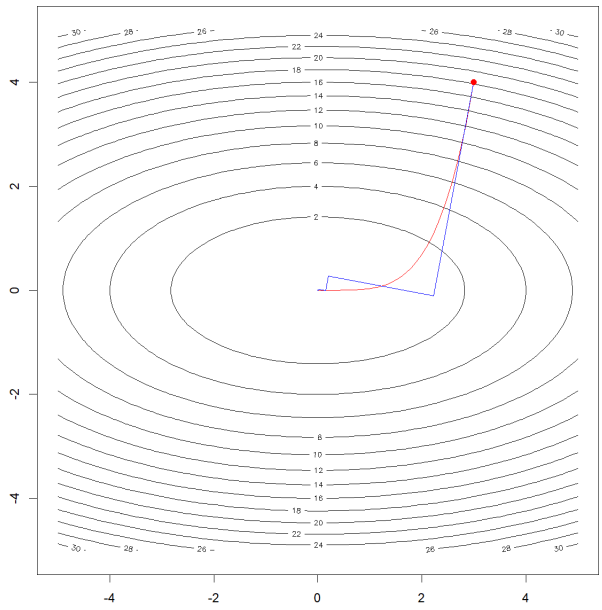
Listing 6: Implementacja algorytmu największego spadku w R

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")
```

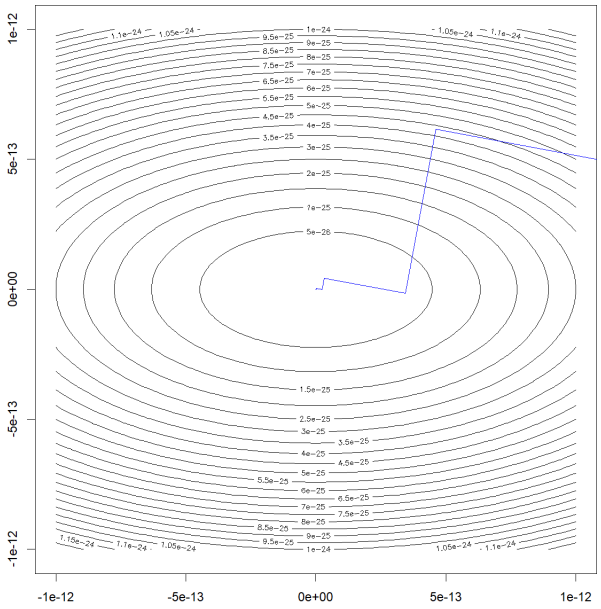
Listing 7: Kod wykorzystany do porównania działania algorytmu GD i SD

Rysunek 1.2 przedstawia wartości funkcji celu w kolejnych iteracjach algorytmu. Widać, że algorytm ten znacznie szybciej zbiega do minimum globalnego funkcji. Natomiast na rysunku 1.1 przedstawiono różnicę w sposobie budowania trajektorii w przestrzeni rozwiązań przez algorytmy GD (czerwony) i SD (niebieski).



Rysunek 1.3: Trajektoria wektora x w kolejnych iteracjach algorytmów GD i SD

Zauważmy, że ortogonalność wektora gradientu i całej trajektorii przeszukiwania jest widoczna również przy dużym powiększeniu (zwróć uwagę na osie)!



Rysunek 1.4: Trajektoria wektora x w kolejnych iteracjach algorytmu SD – duże powiększenie!

1.7 Spadek Newtona (Newton Descent)

Wzór iteracyjny metody Spadku Newtona jest następujący:

$$x_{k+1} = x_k - \mathrm{H}_f^{-1}(x_k)\nabla_f(x_k)$$

Skąd to wiemy? Rozważmy wzór aproksymacyjny Taylora:

$$f(x+h) \approx f(x) + \nabla_f(x)h + \frac{1}{2}h^T H_f(x)h$$

oznacza to, że dla FOC ze względu na parametr h, tj. $\frac{df}{dh}(x+h)$ mamy

$$\frac{df}{dh}(x+h) = \nabla_f(x) + H_f(x)h = 0$$

$$h = -H_f(x)^{-1}\nabla_f(x)$$

Oznacza to, że optymalne przesunięcie z punktu x_k to wektor $-\mathrm{H}_f^{-1}(x_k)\nabla_f(x_k)$. Zwróćmy uwagę, że h minimalizuje wyrażenie przybliżenia wielomianu drugiego stopnia.

Zauważmy, że dla wielomianów drugiego stopnia (przyjmijmy na chwilę przypadek jednowymiarowy $f : \mathrm{R} \to \mathrm{R}$, metoda Newtona zbiega w jednej iteracji!

$$f(x) = ax^2 + bx + c?$$

$$x^{(2)} = x^{(1)} - \frac{2x^{(1)}a + b}{2a} = x^{(1)} - x^{(1)} - \frac{b}{2a} = -\frac{b}{2a}$$

Metoda Newtona przybliża wykres funkcji f wielomianem drugiego stopnia. Dla zadanego przybliżenia znajdowane jest ekstremum, do którego przechodzimy – przyjmujemy podejście iteracyjne.

Do wykorzystania metody Newtona potrzebna jest znajomość macierzy Hesianu w dowolnym punkcie f. W tym celu stosujemy przybliżenie numeryczne, którego osiągane jest przy pomocy poniższej funkcji:

```
num_hessian <- function(f, x, h=10^-3){
  #' Numerical hessian
  #'
  #' @description Numerical (central) hessian calculation
  #'
  #' @param f function that is subject to be calculate hessian
  #' @param x point at with the gradient is calculated (must be compliant with
  #' f function).
  #' @param h finite difference (numerical perturbation scalar).
  #'
  #' @return hessian matrix of f.

  n <-length(x)
  H <- matrix(NA, nrow=n, ncol=n)
  e <- diag(n)
  for(i in 1:n){
    for(j in 1:n){
      if(i>j){
        H[i,j] <- H[j,i]
      }else{
        H[i,j] <- (f(x+e[i,]*h+e[,j]*h)-f(x-e[i,]*h+e[,j]*h)
                   -f(x+e[i,]*h-e[,j]*h)+f(x-e[i,]*h-e[,j]*h)) / (4*h^2)
      }
    }
  }
  return(H)
}
```

Listing 8: Implementacja algorytmu spadku gradientowego

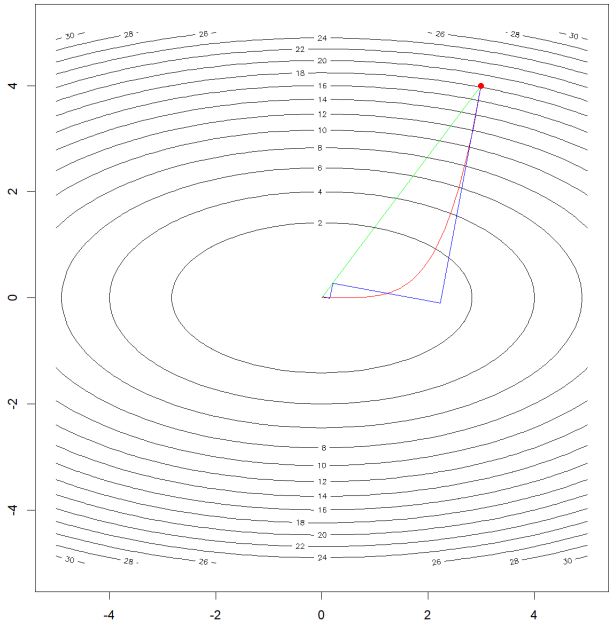
Poniżej przedstawiono przykładową implementację metody Newtona.

```
source('num_grad.R')
source('num_hessian.R')

newton_descent <- function(f, x, max_iter=100){
  #' Newton Descent solver
  #'
  #' @description Newton Descent Solver (for R^n -> R functions)
  #'
  #' @param f objective function that is subject to be optimize.
  #' @param x initial solution that is scalar/vector (must be compliant with
  #' f function).
  #' @param max_iter maximum number of iterations used in the solver.
  #'
  #' @return list with the optimization results.

  start_t <- Sys.time()
  n<-length(x)
  out <- list(x_opt=x,
              f_opt=f(x),
```

```
                x_hist=matrix(NA, nrow=max_iter, ncol=n),
                f_hist=rep(NA, max_iter),
                a=rep(NA, max_iter),
                t_eval=NA)

  # Assign first solution as a x0 imputed to the function
  out$x_hist[1,] <- x
  out$f_hist[1] <- f(x)
  out$a[1] <- NA

  xk <- x
  for(k in 2:max_iter){
    G <- num_grad(f, xk)
    H <- num_hessian(f, xk)
    if(abs(det(H)) < 10^-3) H <- H+diag(n)*10^-3

    xk <- xk - solve(H)%*%G
    if(f(xk) < f(x)){
      x <- xk
    }
    out$x_hist[k,] <- xk
    out$f_hist[k] <- f(xk)
  }

  # Assign optimal solution to output
  out$x_opt <- x
  out$f_opt <- f(x)
  out$t_eval <- Sys.time() - start_t
  return(out)
}
```

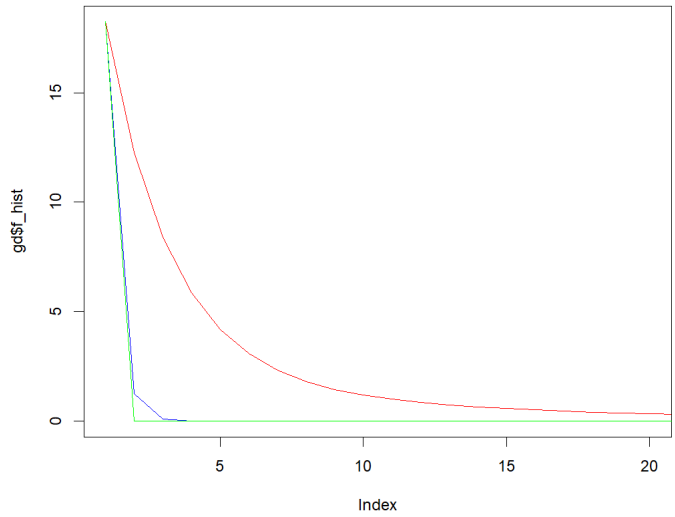
Listing 9: Kod wykorzystany do porównania działania algorytmu GD

Poniżej przedstawiono porównanie metod: Spadek Gradientowy, Najszybszy Spadek, oraz Spadek Newtona.



Rysunek 1.1: Trajektoria wektora x w kolejnych iteracjach algorytmów GD, SD, ND

Poniżej porównano wartości funkcji celu, dla poszczególnych solwerów optymalizacyjnych.



Rysunek 1.2: Wartość funkcji celu w kolejnych iteracjach algorytmów GD, SD i ND SD

Bibliografia

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