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

Zajęcia 5: Metody wyższego rzędu

Daniel Kaszyński

5.1 Metoda Newtona

5.1.1 Teoria stojąca za metodą Newtona

Metoda Newtona (ang. *Newton Descent*) to kolejny algorytm iteracyjny, często używany do minimalizacji funkcji, podobnie jak metoda gradientu prostego i metoda najszybszego spadku omawiane w poprzednich wykładach. Wykorzystuje aproksymacje drugiego rzędu do znalezienia punktów krytycznych danej funkcji f. Podstawową ideą metody Newtona w optymalizacji jest iteracyjne aktualizowanie wstępnego przypuszczenia dotyczącego optymalnego rozwiązania w oparciu o pierwszą i drugą pochodną funkcji. Reguła aktualizacji wywodzi się z rozwinięcia funkcji w szereg Taylora wokół bieżącego przypuszczenia.

Aby lepiej zrozumieć działanie tego algorytmu, zacznijmy od przybliżenia szeregu Taylora wokół punktu x_k aż do pochodnej drugiego stopnia (przybliżenie Taylora drugiego rzędu f):

$$f(x_k + h) \approx f(x_k) + f'(x_k)h + \frac{1}{2}f''(x_k)h^2 \tag{5.1}$$

Celem tej metodologii jest znalezienie h, dla którego funkcja wokół danego punktu x_k zmienia się najszybciej. Zakładamy, że w każdym kroku podane jest x_k, a zatem jest to wartość stała. Mając to na uwadze możemy dokonać obserwacji, iż szereg Taylora jest wielomianowym wzorem na aproksymację wartości ($f(x_k)$, $f'(x_k)$ oraz $\frac{1}{2}f''(x_k)$ są stałe, pozostawiając nam do dyspozycji tylko h).

Następnie możemy zastosować warunki pierwszego rzędu i przyrównać pochodną naszego przybliżenia do zera:

$$0 = \frac{d}{dh}\left(f(x_k) + f'(x_k)h + \frac{1}{2}f''(x_k)h^2\right) = f'(x_k) + f''(x_k)h \tag{5.2}$$

Następnie, stosując proste przekształcenie wzoru, możemy otrzymać wzór na optymalną wartość h:

$$h = -\frac{f'(x_k)}{f''(x_k)} \tag{5.3}$$

Ten wzór na h można wykorzystać do obliczenia kroków podejmowanych w kolejnych iteracjach metody Newtona dla funkcji 1D.

Definicja 1: Metoda Newtona (ang. *Newton Descent*) dla funkcji 1D

Mając na uwadze fakt, iż metoda Newtona jest algorytmem iteracyjnym, punktem obliczonym w $(k-1)$-tej iteracji tego algorytmu dla funkcji $f : \mathbb{D} \subset \mathbb{R}^n \to \mathbb{R}$, $x_k \in \mathbb{D}$ nazywamy:

$$x_{k+1} = x_k + h = x_k - \frac{f'(x_k)}{f''(x_k)} \tag{5.4}$$

gdzie $k \in \mathbb{N}$ jest numerem iteracji, $f'(x_k)$ jest pochodną pierwszego stopnia funkcji f w punkcie x_k oraz $f''(x_k)$ jest pochodną drugiego stopnia funkcji f w punkcie x_k.

Algorytm Newton descent przedstawiony we wzorze powyżej (5.4) można uogólnić do większej ilości wymiarów, zastępując pochodną gradientem, a odwrotność drugiej pochodnej odwrotnością Hesjanu (ponieważ jak wiemy z poprzednich wykładów, Hesjan jest uogólnieniem drugiej pochodnej).

Definicja 2: Metoda Newtona (ang. *Newton Descent*)

Mając na uwadze fakt, iż metoda Newtona jest algorytmem iteracyjnym, punktem obliczonym w $(k-1)$-tej iteracji tego algorytmu dla funkcji $f : \mathbb{D} \subset \mathbb{R}^n \to \mathbb{R}$, $x_k \in \mathbb{D}$ nazywamy:

$$x_{k+1} = x_k - H_f(x_k)^{-1} \nabla_f(x_k) \tag{5.5}$$

gdzie $k \in \mathbb{N}$ jest numerem iteracji, $H_f(x_k)^{-1}$ jest odwrotnością Hesjanu a $\nabla_f(x_k)$ jest gradientem funkcji f obliczoną w punkcie x_k.

Uwaga! Używając metody Newtona, domyślnie nie wiemy, czy optymalizujemy funkcje w kierunku minimum, maksimum czy innych punktów krytycznych danej funkcji. Newton Descent kieruje się w stronę punktu stacjonarności funkcji. Dlatego dobrą praktyką jest wykorzystanie metody Newtona jako ostatniego kroku optymalizacji, aby szybciej dojść do ekstremum funkcji, do której doszliśmy wcześniej innym algorytmem.

Przykład 1. Niech $f(x) = ax^2 + bx + c$ oraz niech x_0 będzie punktem startowym metody Newtona.

Korzystając ze wzoru na metodę Newtona dla podanej funkcji 1D f, otrzymujemy:

$$x_1 = x_0 - \frac{2ax_0 + b}{2a} \tag{5.6}$$

Możemy wówczas podzielić ułamek uzyskany po prawej stronie równania na dwie części:

$$x_1 = x_0 - \frac{2ax_0}{2a} - \frac{b}{2a} \tag{5.7}$$

Dzięki tej transformacji możemy wyeliminować wyrażenia zawierające x_0:

$$x_1 = -\frac{b}{2a} \tag{5.8}$$

Otrzymany wzór na x_1 jest wzorem na wierzchołek funkcji wielomianowej drugiego rzędu (w tym przypadku minimum). Zatem w jednym kroku jesteśmy w stanie znaleźć ekstremum funkcji. Jak widzimy, algorytm ten jest bardzo skuteczny, gdy mamy do czynienia z problemami związanymi z formami kwadratowymi.

Jak widać we wzorze na Newton descent (5.5), aby obliczyć kolejne kroki wykonywane w każdej iteracji algorytmu, musimy znać Hesjan funkcji. Aby móc obliczyć Hesjan funkcji, należy wyprowadzić pochodną

cząstkową po x_i oraz x_j.

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

Aby obliczyć Hesjan numerycznie, musimy zastosować przybliżenia pochodnych cząstkowych - różnic skończonych. Oznacza się je za pomocą znaku Δ. Dlatego macierz Hessego wykorzystująca wspomniane przybliżenia wyglądałaby następująco:

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

Mając na uwadze wzór na różnicę centralną, możemy zapisać pojedynczą skończoną różnicę po x_i jako:

$$\frac{\partial f}{\partial x_i}(x) \approx \frac{\Delta f}{\Delta x_i}(x) = \frac{f(x + e_i h) - f(x - e_i h)}{2h} \tag{5.9}$$

Podobnie, każdy element macierzy Hessego, będący skończoną różnicą pomiędzy x_i i x_j, będzie wyglądał następująco:

$$\frac{\Delta^2 f}{\Delta x_i \Delta x_j}(x) = \frac{f(x + e_i h + e_j h) - f(x + e_i h - e_j h) - f(x - e_i h + e_j h) + f(x - e_i h - e_j h)}{4h^2} \tag{5.10}$$

Warto zaznaczyć, że kolejność stosowania przybliżeń pochodnych cząstkowych nie ma znaczenia. Jeśli funkcja $f \in C^2$ (jest co najmniej dwukrotnie różniczkowalna), korzystając z twierdzenia Schwartza, możemy udowodnić, co następuje:

$$\frac{\Delta^2 f}{\Delta x_i \Delta x_j} = \frac{\Delta^2 f}{\Delta x_j \Delta x_i} \tag{5.11}$$

co również świadczy o:

$$H_f(x) = H_f^T(x) \tag{5.12}$$

Kolejną rzeczą wartą uwagi jest fakt, iż nie z każdej funkcji można łatwo uzyskać Hesjan (na przykład funkcje, które nie mają krzywizny). Aby zapewnić prawidłową pracę algorytmu, należy zastosować pewien trik. Ten trik sprowadza się do korekcji Hesjanu funkcji poprzez dodanie do niego macierzy diagonalnej pomnożonej przez małą wartość λ. Zmianę tę należy zastosować zawsze, gdy wyznacznik $abs(H_f(x))$ jest mniejszy niż odgórnie ustalony próg t. Opisana metoda jest często nazywana **regresją grzbietu** lub **korektą Levenberga-Marquardta**.

5.1.2 Metoda Newtona-Raphsona

There is also another algorithm, similar to Newton descent, that is worth mentioning. It is called the Newton-Raphson method (and sometimes just Newton's method). This algorithm is a root-finding algorithm which produces successively better approximations to the roots (or zeroes) of a given function. The formula used by this algorithm is as follows:

Warto również wspomnieć o innym algorytmie, podobnym do metody Newtona. Nazywa się on metodą Newtona-Raphsona (a czasami po prostu metodą Newtona). Algorytm ten jest algorytmem znajdowania

miejsc zerowych, który generuje sukcesywnie lepsze przybliżenia tych miejsc dla danej funkcji. Wzór używany przez ten algorytm wygląda następująco:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \tag{5.13}$$

Uwaga! Jest to wzór bardzo podobny do wzoru metody Newtona, różni się jednak stopniem pochodnych funkcji f. Dwa wspomniane algorytmy próbują rozwiązać dwa różne problemy. Łączy ich jednak wspólna relacja. Metoda Newtona-Raphsona znajduje miejsca zerowe podanej jej funkcji. Zakładając, że dana funkcja jest już pochodną innej funkcji (branej pod uwagę przez algorytm Newton descent), znalezienie tych miejsc zerowych w istocie daje nam wzór na Newton descent i rozwiązuje adresowane przez niego problemy.

5.1.3 Implementacja algorytmu Newton descent

Używając języka programowania **R** możemy napisać implementację metody Newtona. Po pierwsze potrzebujemy funkcji do obliczenia numerycznej macierzy Hessego:

```
num_hessian <- function(f, x, h = 10^-3){
  #' Hesjan Numeryczny
  #'
  #' @description Funkcja odpowiedzialna za wyznaczenie numerycznego Hesjanu
  #' poprzez obliczenie macierzy pochodnych czastkowych.
  #'
  #' @param f funkcja. Funkcja, ktorej Hesjan wyznaczamy
  #' @param x wektor numeryczny. Punkt, w ktorym wyznaczany jest
  #' numeryczny Hesjan
  #' @param h skalar. Wartosc roznicy skonczonej
  #'
  #' @usage num_hessian(f, x)
  #'
  #' @return Hesjan pochodnych czastkowych funkcji f.

  n <- length(x)
  H <- matrix(NA, nrow = n, ncol = n)
  E <- diag(n)

  for(i in 1 : n) { # Wiersze
      for(j in 1 : n) { # Kolumny
          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 1: Implementacja numerycznej macierzy Hessego

Wzór stosowany w kolejnych iteracjach algorytmu Newton descent wymaga obliczenia odwrotności macierzy Hessego. Na szczęście język **R** zapewnia gotowe rozwiązanie tego problemu w postaci metody *solve()*.

```
my_fun <- function(x) 1/8*x[1]^2+x[2]^2
x0 <- c(3, 4)
solve(num_hessian(my_fun, x0)) # Zwraca odwrotnosc macierzy Hessego
```

Listing 2: Przykład metody solve() dla Hesjanu funkcji

Having the ability to calculate the inverse of the matrix, we can finally create the code necessary for the Newton descent algorithm. Example of it's implementation can look like this:

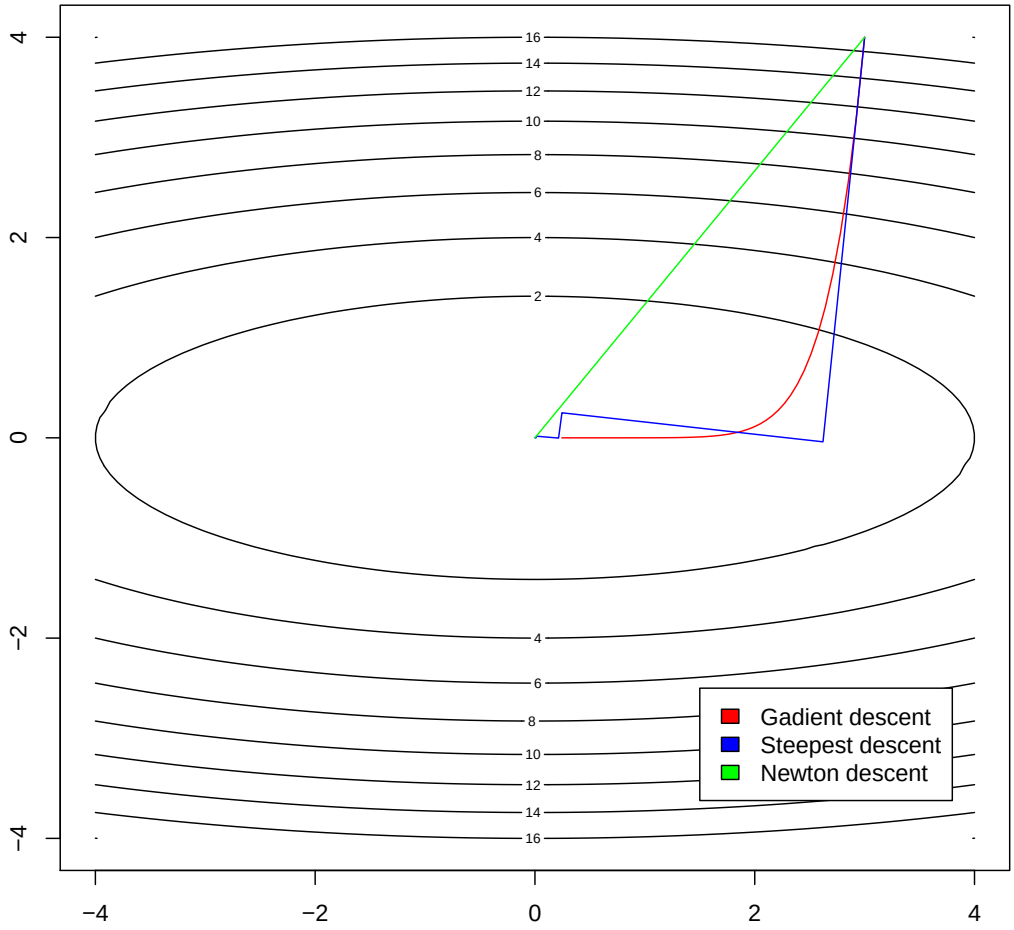
Mając możliwość obliczania odwrotności macierzy, możemy teraz stworzyć kod algorytmu metody Newtona. Przykład implementacji może wyglądać następująco:

```
newton_descent <- function(f, x, K = 100){
  #' METODA NEWTONA
  #'
  #' WEJSCIE:
  #'       - f: funkcja celu
  #'       - x: punkt startowy
  #'       - K: maksymalny limit iteracji
  #'
  #' WYJSCIE:
  #'       - x_opt: znalezione rozwiazanie
  #'       - f_opt: wartosc funkcji celu w znalezionym rozwiazaniu
  #'       - x_hist: historia eksplorowanych rozwiazan
  #'       - f_hist: historia wartosci funkcji celu
  #'       - t_eval: czas 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){
    # obliczanie gradientu i hesjanu funkcji f
    G <- grad(f, x)
    H <- num_hessian(f, x)

    # uzycie regresji grzbietowej w celu rozwiazania
    # sytuacji, w ktorych nie mozna obliczyc hesjanu
    if(abs(det(H)) < 10^(-3)) H <- H + diag(n)*10^(-3)

    # opis przejscia z punktu x_k do x_k+1
    x_new <- x - solve(H) %*% G

    # sprawdzenie czy nowe rozwiazanie
    # jest najlepszym dotychczas
    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 czasu pomiedzy koncem i startem algorytmu
  results$t_eval <- Sys.time() - start_time
  return(results)
}
```

Listing 3: Przykładowa implementacja Newton descent

Przykład 2. Niech $f(x_1, x_2) = \frac{1}{8} \cdot (x_1)^2 + (x_2)^2$ oraz $x_{start} = (3, 4)$.

Zbieganie metody Newtona można zobaczyć na poniższym wykresie (wraz ze ścieżkami utworzonymi przez inne podobne algorytmy):

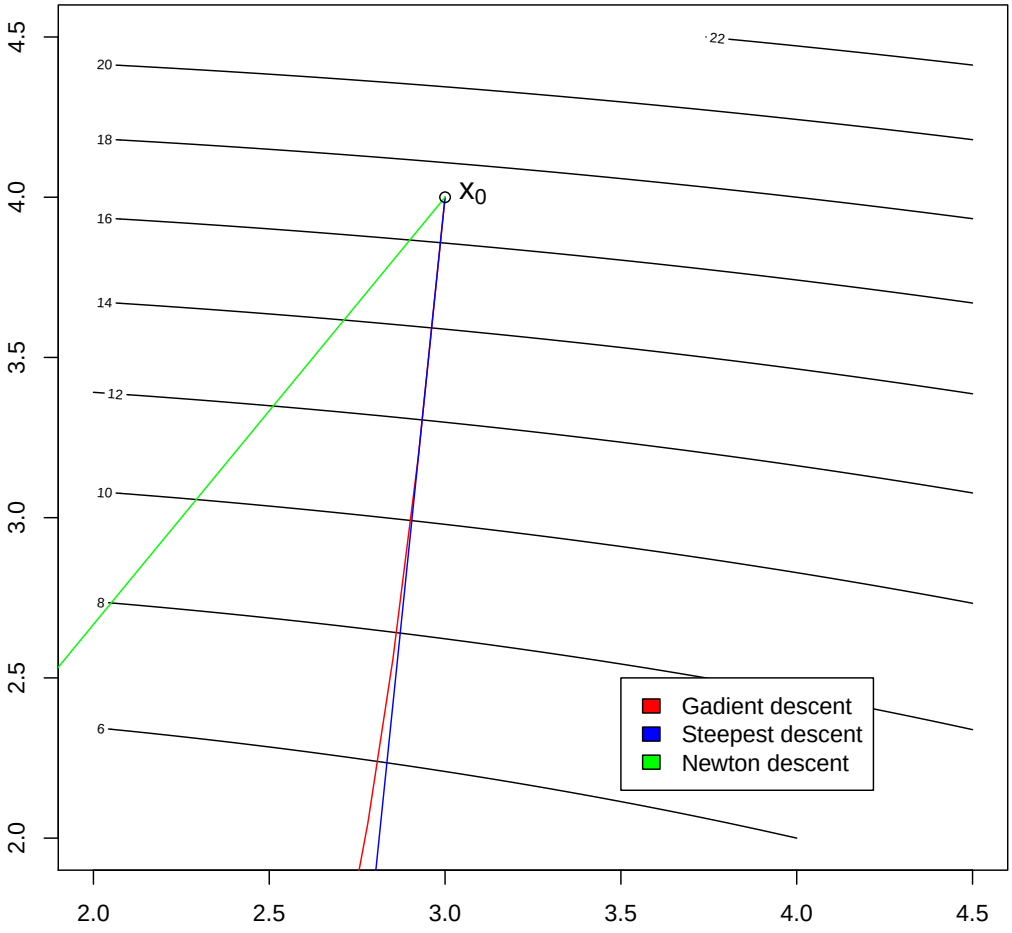


Rysunek 5.1: Porównanie ścieżek uzyskanych przez metodę Newtona i inne podobne algorytmy na funkcji f w przestrzeni 2D

We can see that Newton descent has a path that is significantly different from the other algorithms. Steepest descent and gradient descent algorithms are moving orthogonal with regards to counterplot. Newton descent takes a much more direct route towards the global minimum of the function. It is no longer going towards locally best direction. This behavior is caused by the fact that the function f is quadratic.

Możemy zaobserwować, że metoda Newtona ma ścieżkę znacznie różniącą się od pozostałych algo-

rytmów. Metoda najszybszego spadku i metoda gradientu prostego poruszają się ortogonalnie w odniesieniu do wykresu przeciwnego. Newton descent zmierza znacznie bardziej bezpośrednio w kierunku globalnego minimum funkcji. Nie kieruję się już w stronę lokalnie najlepszego rozwiązania. To zachowanie jest spowodowane faktem, że funkcja f jest funkcją kwadratową.



Rysunek 5.2: Zbliżenie ścieżek uzyskanych metodą Newtona i innych podobnych algorytmów