WSI - Gradient prosty

Jan Szymczak

Marzec 2024

1 Cel i opis eksperymentów

Celem eksperymentów było wykorzystanie algorytmu gradientu prostego w celu znajdowania minimum podanych funkcji. Badane były dwie funkcje:

$$f(x) = 2x^2 + 3x - 1, \text{ oraz}$$
$$g(x) = 1 - 0.6exp\{-x_1^2 - x_2^2\} - 0.4exp\{-(x_1 + 1.75)^2 - (x_2 - 1)^2\}$$

Tak przedstawiają się ich wykresy:

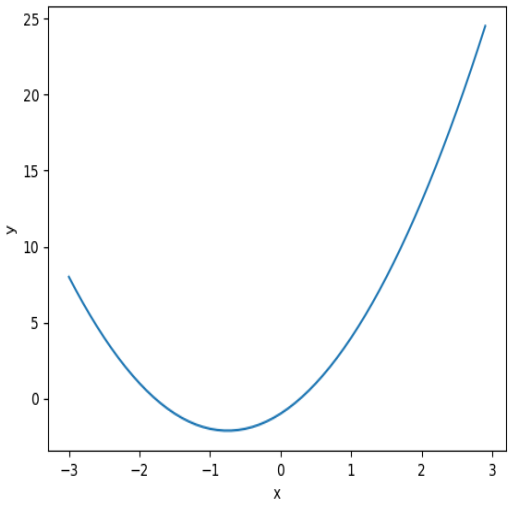

Figure 1: Wykres funkcji f.

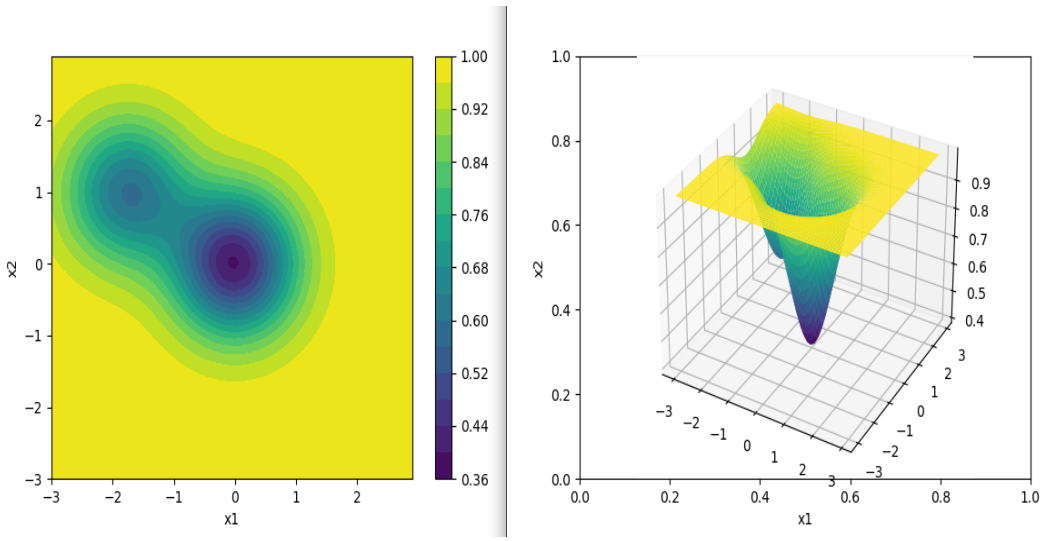

Figure 2: Wykresy funkcji g w 2d i 3d.

W poniższych testach skupiłem się przede wszystkim na wpływie wielkości kroku, hiperparametru *beta*, oraz wybranego punktu początkowego. Głównym założeniem projektu jest fakt, że gradient funkcji zeruje

się tylko w jej minimach. W praktyce gradient ten nigdy nie wynosił dokładnie zero - zbliżając się do faktycznego minimum stale malał do wartości coraz niższych, dążących do zera. Stąd jednym z argumentów algorytmu podawanym przy jego wywoływaniu jest parametr *epsilon*, który definiuje de facto dokładność obliczeń, w praktyce określa wielkość, poniżej której liczby można traktować już jako zero. Innymi słowy algorytm działa i szuka minimum tak długo, jak gradient badanej funkcji nie osiągnie wartości mniejszej od podanego parametru *epsilon*. Przy testach przyjąłem wartość tego parametru równą $1e-6$. W teorii, czym bliżej punkt początkowy jest szukanego minimum, tym szybciej algorytm powinien je znaleźć. Zależne to też jest od długości kroku - większy parametr beta umożliwia szybsze znalezienie minimum, jednak gdy przekroczy on pewną wartość (odwrotność modułu największej wartości własnej hesjanu funkcji) algorytm wpada w oscylacje wokół minimum, przeskakując szukany punkt przez zbyt duży krok.

2 Wyniki

2.1 Funkcja f

Algorytm dla funkcji jednowymiarowych jest bardzo efektywny. Tym bardziej dla funkcji posiadających tylko jedno minimum lokalne, które jednocześnie stanowi minimum globalne, a takimi są funkcje kwadratowe, a więc także funkcja f. Przedział wartości parametru beta, dla którego można mówić o efektywnym działaniu to $\langle 0.02; 0.48\rangle$. Dla tego przedziału przeprowadziłem symulację, w której sprawdzałem działanie algorytmu dla każdej wartości parametru z tego przedziału z krokiem 0.01 dla losowego punktu startowego z przedziału $\langle -100; 100\rangle$. Dla zarówno wyższych jak i niższych wartości parametru algorytm potrzebował znacznie większej ilości iteracji lub w ogóle nie znajdował minimum. W tym przedziale jednak, każdy parametr beta umożliwił znalezienie minimum, które znajdowało się w punkcie $x = -0.75$ i osiągało wartość $f(x) = -2.125$. Tak przedstawia się wykres ilości iteracji w zależności od parametru beta:

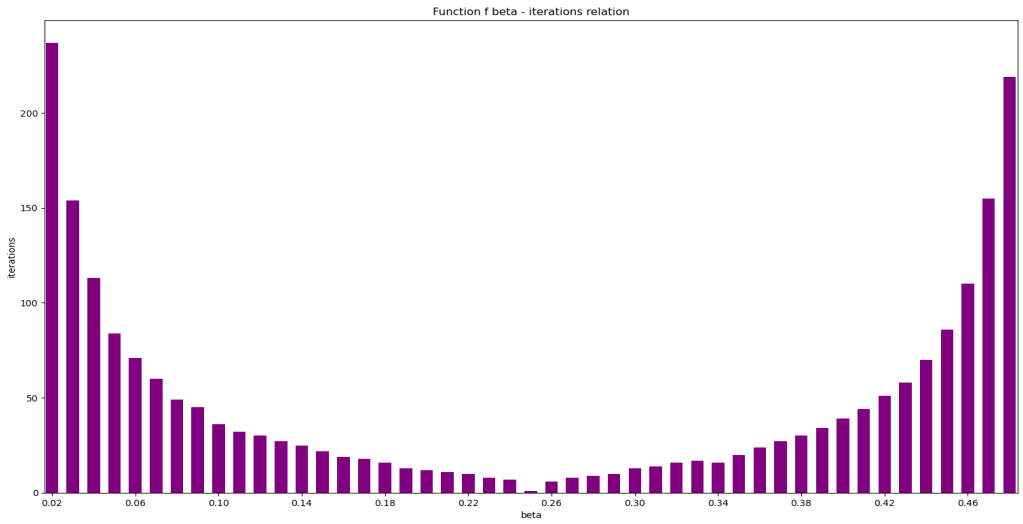

Figure 3: Zależność iteracji od parametru beta dla funkcji f.

Jak widać, najbardziej optymalną wartością jest 0.25. Dla tej wartości przeprowadziłem testy, w których sprawdziłem ilość iteracji działania algorytmu dla 200 losowych punktów z przedziału $\langle -1e6; 1e6\rangle$:

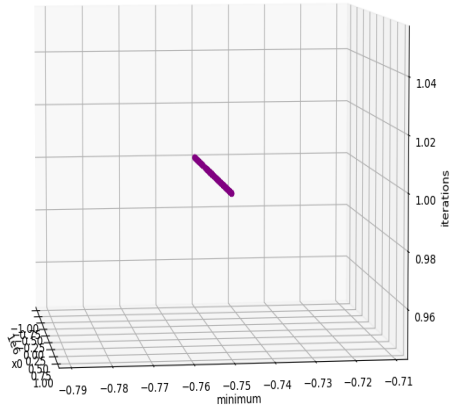

Figure 4: Działanie algorytmu dla beta = 0.25 w funkcji f.

Algorytm, niezależnie od odległości punktu startowego od minimum, znajdował je w tylko 1 iteracji. Jednak każda wartość parametru nawet z przedziału $\langle -0.06; 0.42 \rangle$ potrzebowała maksymalnie 50 iteracji do znalezienia minimum, co również jest dobrym wynikiem. Wpływ parametru beta na ilość iteracji dobrze przedstawia poniższa heatmapa, gdzie każda wartość parametru z podanego wyżej przedziału była testowana dla zestawu tych samych punktów z przedziału $\langle -100; 100 \rangle$:

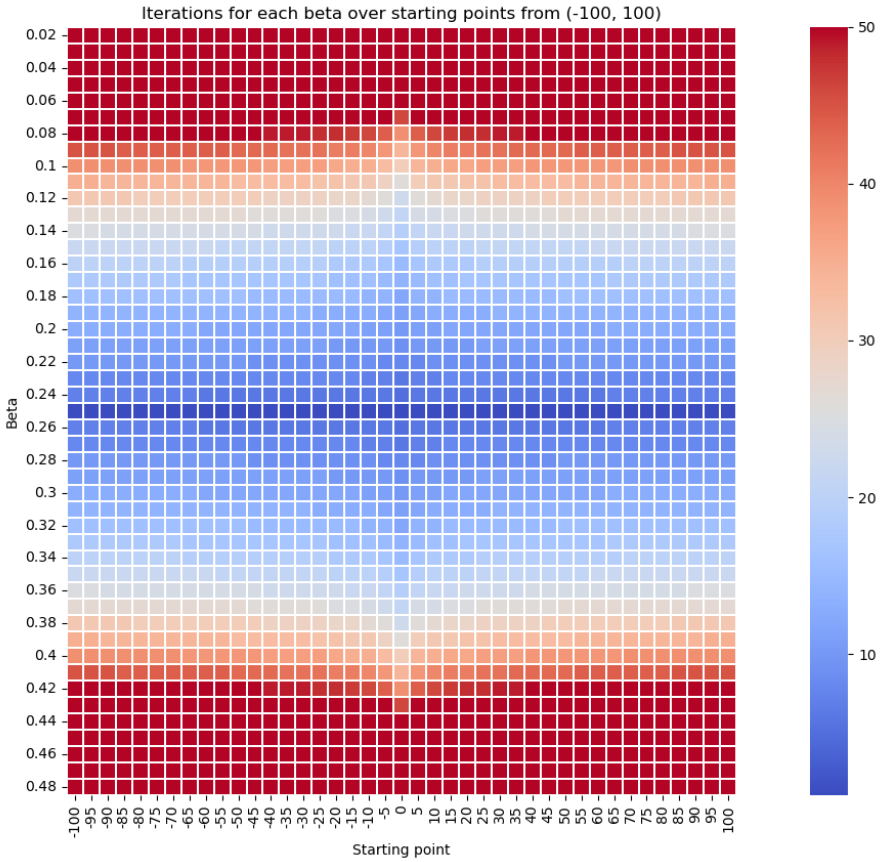

Figure 5: Heatmapa ukazująca wpływ parametru beta.

Punkt startowy również ma znaczenie, ale tym mniejsze im większy jest parametr beta. Widać, że dla $x_0 = 0$ liczba iteracji maleje dla każdej wartości parametru. Co ciekawe jednak, heatmapa jest w przybliżeniu symetryczna względem osi przechodzącej własnie przez punkt 0, w miare oddalania się wartości parametru od wartości optymalnej 0.25 ilość iteracji wzrasta, niezależnie od tego czy parametr maleje czy wzrasta w podanym przedziale. Dokładne wartości poszczególnych prób znajdują się w repo w pliku data_f.csv

2.2 Funkcja g

Funkcja ta jest już bardziej złożona. Jest dwuwymiarowa, posiada dwa minimum lokalne, położone dodatkowo blisko siebie oraz "wypłaszcza" się szybko poza sąsiedztwem minimum, co wykazało wadę algorytmu. W obszarach, gdzie nachylenie funkcji jest w przybliżeniu równe zero, gradient także jest w przybliżeniu równy zero, co powoduje, że algorytm traktuje te punkty jako minimum lokalne. Oczywiście, w teorii w najbliższym otoczeniu tego punktu to rzeczywiście jest minimum lokalne, bo funkcja przyjmuje tam tą samą wartość, jednak w tym problemie zależy nam na minimum globalnym. Zatem w tym przypadku skuteczność znalezienia minimum globalnego zależy tak naprawdę od wybranego punktu początkowego. Z tego powodu, żeby przetestować wpływ parametru beta, dla każdej wartości parametru przeprowadziłem symulację dla 100 losowych punktów z przedziału $\langle -2.5; 2.5 \rangle$ i medianę z wyników przedstawiłem na poniższym wykresie:

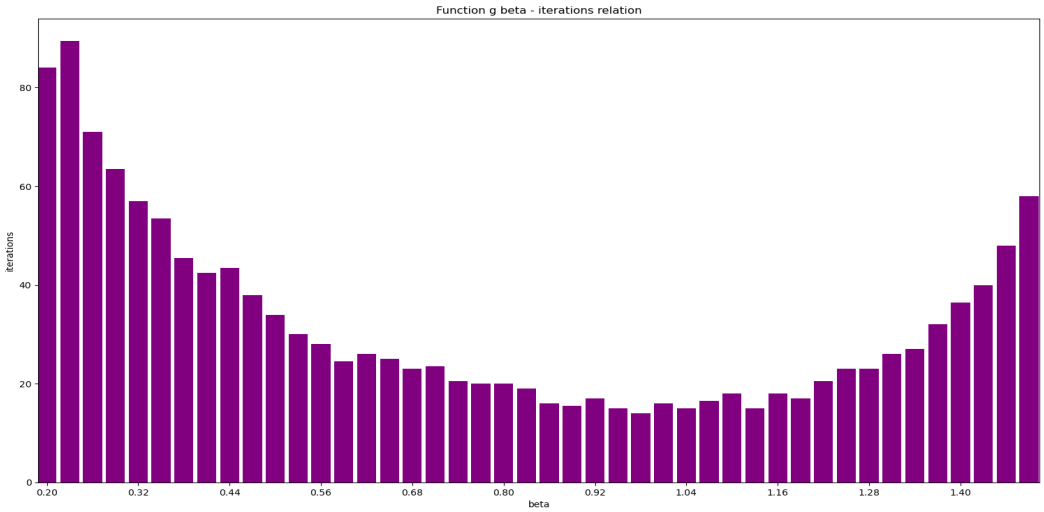

Figure 6: Zależność iteracji od parametru beta dla funkcji g.

Użycie mediany było tu nieuniknione, ponieważ dla punktów skrajnych algorytm czasem potrzebował znacznie większej ilości iteracji niż zwykle, przez co wykres był nieczytelny i niemiarodajny. Dla tej funkcji trudniej też wskazać jeden optymalny parametr, ilość iteracji nieznacznie zmienia się przy każdym teście, jednak kształt wykresu zawsze pozostaje ten sam. Najbardziej optymalny wydaje się zatem parametr z przedziału $\langle 0.86; 1.04 \rangle$. Następnie, dla parametru o najlepszym wyniku z danej próby, przeprowadziłem symulację, w której algorytm starał się znaleźć minimum dla 100 losowych punktów z przedziału $\langle -5; 5 \rangle$. Na poniższym wykresie czerwone krzyżyki oznaczają, że algorytm uznał dany punkt za minimum, niebieskie kropki oznaczają punkty startowe, ich wielkość określa ilość iteracji, a przerywane linie łączą wylosowany punkt startowy z odnalezionym minimum:

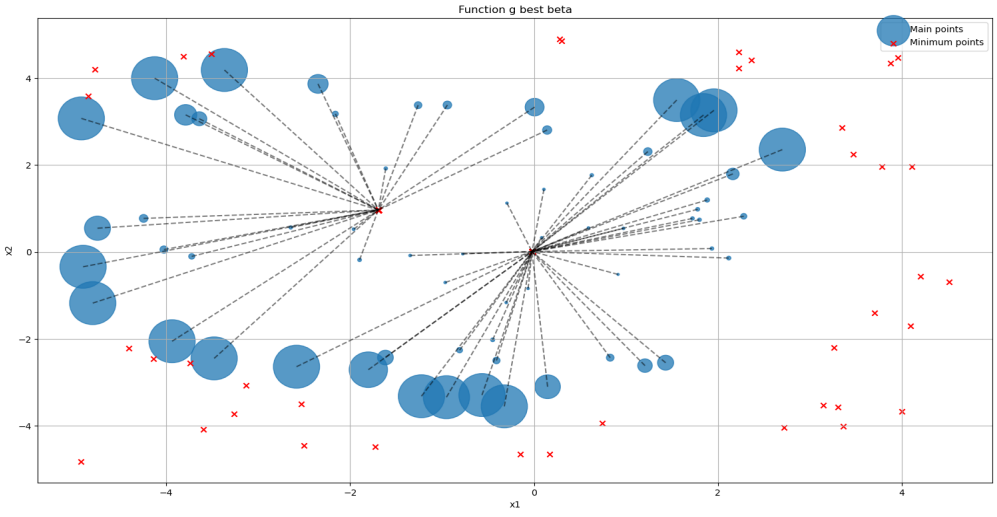

Figure 7: Test najlepszego parametru beta dla funkcji g.

Z większości punktów algorytm rzeczywiście odnajduje minimum, częściej globalne w okolicach punktu $(0, 0)$ o wartości ok. 0.39 niż drugie, lokalne w $(-1.69, 0.97)$ o wartości 0.59, jednak widać, że algorytm jest w stanie je odnaleźć tylko w najbliższym ich otoczeniu. Ponadto, na granicach tych obszarów liczba iteracji

znacznie wzrasta, przekraczając często wartości kilkunastu tysięcy iteracji. Co więcej, na granicach podanego przedziału na wykresie widać wiele czerwonych krzyżyków, co oznacza, że w tych punktach nachylenie było tak małe, że algorytm uznał te miejsca od razu jako minimum lokalne.

3 Wnioski

Użycie gradientu prostego pozwala na bardzo szybkie i efektywne znajdowanie minimum dla funkcji niezłożonych, jednak algorytm ten napotyka problemy z funkcjami wielowymiarowymi posiadającymi więcej niż jedno minimum lub funkcjami o obszarach o małym nachyleniu. Implementacja algorytmu jednak jest bardzo prosta, a jego złożoność obliczeniowa niska i zależna tak naprawdę od skomplikowania gradientu funkcji, więc nie można oczekiwać świetnych efektów w przypadku wymagających problemów.