Zadanie 1

Julia Jodczyk
310714

Polecenie

Zaimplementuj algorytm gradientu prostego oraz algorytm Newtona. Algorytm Newtona powinien móc działać w dwóch trybach:

- ze stałym parametrem kroku

- z adaptacją parametru kroku przy użyciu metody z nawrotami.

Zbadaj zbieżność algorytmów, używając funkcji: $\sum_{i=1}^{n} \alpha^{\frac{i-1}{n-1}} x_i^2 \, , dla\ \alpha \in \{1, 10, 100\}\ i\ n \in \{10, 20\}$

Zbadaj wpływ wartości parametru kroku na zbieżność obu metod. Ponadto porównaj czasy działania obu algorytmów.

Opis algorytmów

Metoda gradientu prostego i Newtona są algorytmami mającymi na celu znalezienie minimum lokalnego funkcji celu. Kroki podstawowej wersji algorytmów są następujące:

1. Wybierz punkt początkowy x0 i załaduj go do x

2. Dopóki warunek stopu nie jest zaspokojony:

   a) oblicz kierunek

   b) do x załaduj wartość $x\ -\ parametr\ kroku\ *\ kierunek$

Różnicą między metodą gradientu prostego i Newtona jest sposób obliczania kierunku. W przypadku pierwszego algorytmu jest nim wartość gradientu funkcji celu. Kierunek w metodzie Newtona wylicza się ze wzoru $d = H(x)^{-1} * grad(x)$, gdzie H(x) oznacza hesjan funkcji celu.

Dzięki gradientowi wiemy, w którym kierunku funkcja zmienia się najszybciej. Dlatego wyliczając na jego podstawie następne wartości x mamy sporą szansę na przybliżanie się do minimum. Metoda gradientu prostego działa dobrze w większości przypadków, jednak może wymagać dużej ilości iteracji, a parametr kroku ma duże znaczenie dla optymalności algorytmu. Za duży krok spowoduje, że algorytm 'przeskoczy' minimum i wpadnie w oscylacje nigdy go nie znajdując. Z kolei przy za małym kroku niepotrzebnie wykonamy wiele powtórzeń obliczania kolejnych wartości x.

Przewagą metody Newtona jest wykorzystywanie hesjanu, który informuje, w którym kierunku zmienia się gradient. Dzięki tej wiedzy algorytm może zastosować poprawkę przy doborze następnej wartości x i potencjalnie znaleźć minimum funkcji w mniejszej ilości iteracji. Dodatkowo, algorytm jest mniej wrażliwy na dobór kroku, ponieważ wyznaczany kierunek jest dokładniejszy i co za tym idzie z mniejszą szansą 'przeskoczenia' minimum bądź wpadnięcia w oscylacje.

Punkt startowy dobieram losowo z dziedziny funkcji celu $(-100, 100)^n$, optymalną wartość parametru kroku określę w eksperymentach. Jako warunek stopu przyjmuję $abs(prev\_x\ -\ x) < epsilon$, gdzie $prev\_x$ oznacza poprzednią wartość parametru x. Domyślną wartością epsilonu jest 0.0000001, może być ona modyfikowana przez użytkownika. Ponadto narzucam maksymalną ilość iteracji (domyślnie

5000), aby zapewnić zakończenie działania algorytmu (który mógłby się nigdy nie zakończyć przy funkcji celu nie mającej minimum, lub przy źle dobranym kroku).

Metoda Newtona z nawrotami zakłada dodatkową korektę parametru kroku: $krok *= betha$. Przy każdej iteracji sprawdzany jest warunek:

$$f(x) > f(prev\_x) + gamma * krok * (grad(prev\_x))@(-d)$$

Jeśli jest on spełniony, wartości parametru kroku i x są aktualizowane. Betha i gamma są parametrami dobieranymi przez użytkownika z przedziału: betha (0, 1), gamma (0, 0.5). Domyślnie betha = 0.9, gamma = 0.5. Zadaniem tej korekty jest poprawienie działania algorytmu. Im bliżej minimum jesteśmy, tym mniejszy krok chcielibyśmy mieć, żeby uniknąć 'przeskoczenia'. Metoda z nawrotami powinna powodować szybsze znalezienie najmniejszej wartości funkcji celu i mniejszą ilość iteracji.

Dodatkowe uwagi implementacyjne:

Do obliczania hesjanu i gradientu używam funkcji ***Gradient*** i ***Hessian*** z biblioteki ***numdifftools***, co może mieć wpływ na pomiary czasowe algorytmów.

Planowane eksperymenty:

Eksperymentalnie należy dobrać parametr kroku. Może się on różnić w zależności od funkcji celu i użytej metody wyznaczania minimum. Dobry krok powoduje zmniejszenie wartości funkcji w małej ilości kroków. Dlatego jakość doboru parametru kroku będę oceniać na podstawie wykresu wartości funkcji od ilości iteracji. Dla każdej wartości α i n porównam trzy różne wartości parametru kroku.

Drugim eksperymentem będzie porównanie czasowe metod z zastosowaniem najbardziej optymalnego kroku.

Dobór parametru kroku

Wartość x0 dla n = 10: [84, -27, 8, 74, 59, -81, 14, -85, 78, -65]

Wartość x0 dla n = 20: [33, 17, 10, -69, 79, 78, -85, 7, 85, -47, -15, 74, -71, -41, 22, -85, 23, -76, 5, -42]

Metoda gradientu prostego

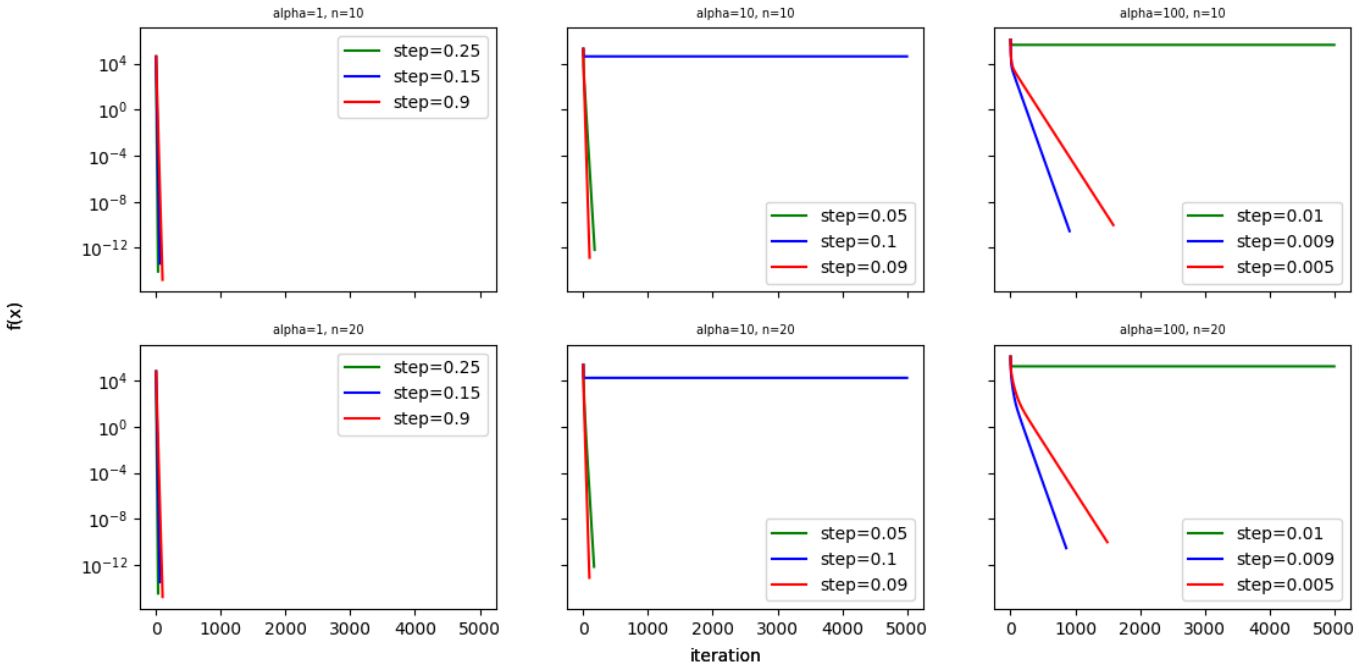
(wykresy pod analizą)

Dla alpha = 1 algorytm szybko znajduje minimum funkcji celu dla każdego z testowanych wartości kroku. Najlepiej radzi sobie z wartością kroku równą 0.25. W przypadku wartości 0.9 parametr kroku jest za duży, więc przy zbliżaniu się do wartości minimalnej algorytm przeskakuje wartość minimalną i musi się do niej wracać, co zajmuje większą ilość iteracji. Parametr kroku równy 0.15 jest za mały, więc żeby zejść do najmniejszej wartości funkcji potrzeba większej ilości powtórzeń (różnice między kolejnymi znajdywanymi wartościami x nie są aż tak duże jak by mogły być).

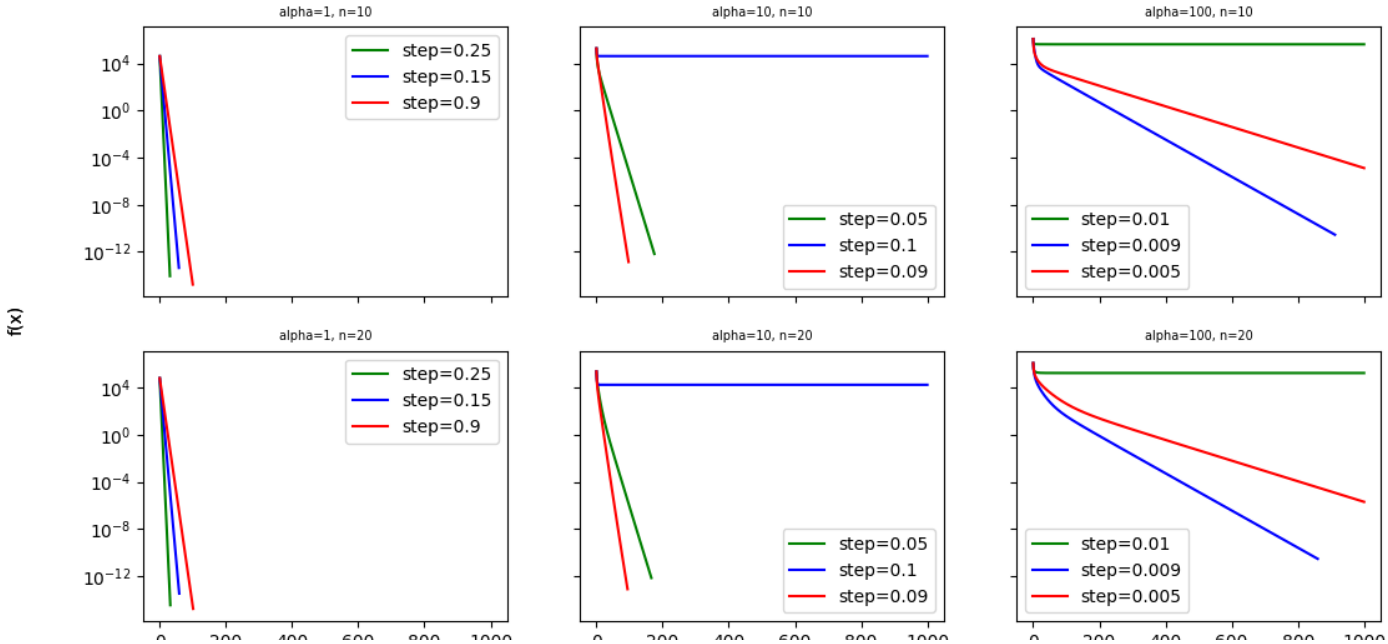
W przypadku alpha = 10 mniejsze wartości parametru kroku działają lepiej, ponieważ wykres funkcji jest bardziej stromy, więc przy większym parametrze kroku łatwiej wpaść w oscylacje. Najlepiej algorytm radzi sobie przy kroku = 0.09. W przypadku parametru kroku 0.1 po kilku iteracjach kierunek przyjmuje wartość bliską zera dla wszystkich współrzędnych x oprócz ostatniej, dla której przyjmuje na zmianę a i -a, gdzie a jest pewną dużą wartością. Dlatego wartość funkcji przestaje się zmieniać i algorytm kończy pracę po maksymalnej ilości iteracji.

Dla alpha = 100, potrzebny jest jeszcze mniejszy krok. Najlepiej działa 0.009.

Im większe jest alpha, tym metoda gradientu prostego jest bardziej wrażliwa na odpowiedni dobór kroku. Patrząc na jednowymiarową funkcję $a * x^2$ widzimy, że im większy parametr a, tym bardziej stromy jest wykres funkcji. Oznacza to, że przy większym parametrze kroku łatwiej przeskoczyć na drugie ramię paraboli, co nie byłoby optymalne. Analogicznie dzieje się z badaną funkcją celu w 10 i 20 wymiarach. Dlatego dla większych wartości alpha musimy używać małego parametru kroku.

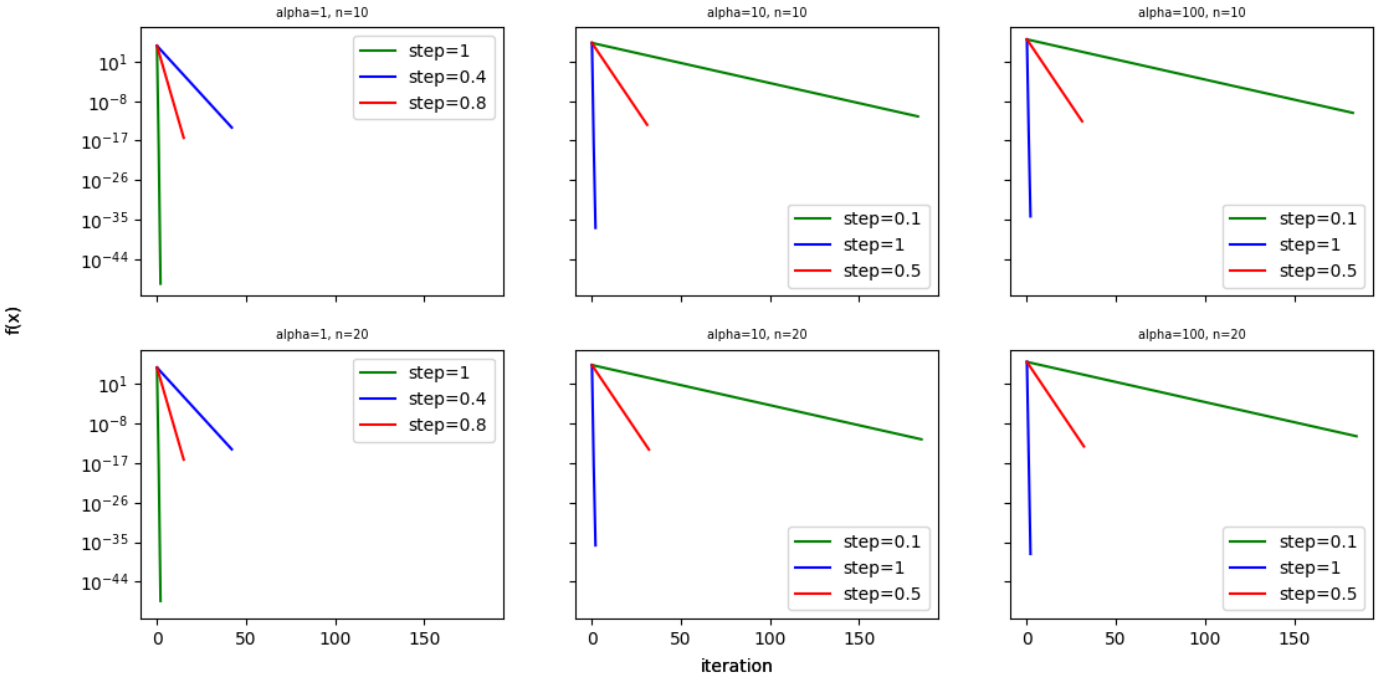


Metoda gradientu prostego



Metoda gradientu prostego przy maksymalnej ilości iteracji równej 1000. Na tych wykresach lepiej widać zachowanie prawidłowych kroków.

Metoda Newtona



Metoda Newtona

Metoda Newtona przy każdej wartości alpha działa najlepiej dla kroku równego 1. Dla alpha = 1 metoda jest bardziej wybaczająca dobór kroku i daje w miarę zadowalające wyniki dla każdej testowanej wartości parametru, przy większych wartościach niewłaściwy krok znacząco pogarsza działanie algorytmu (z tego samego powodu co przy gradiencie prostym).

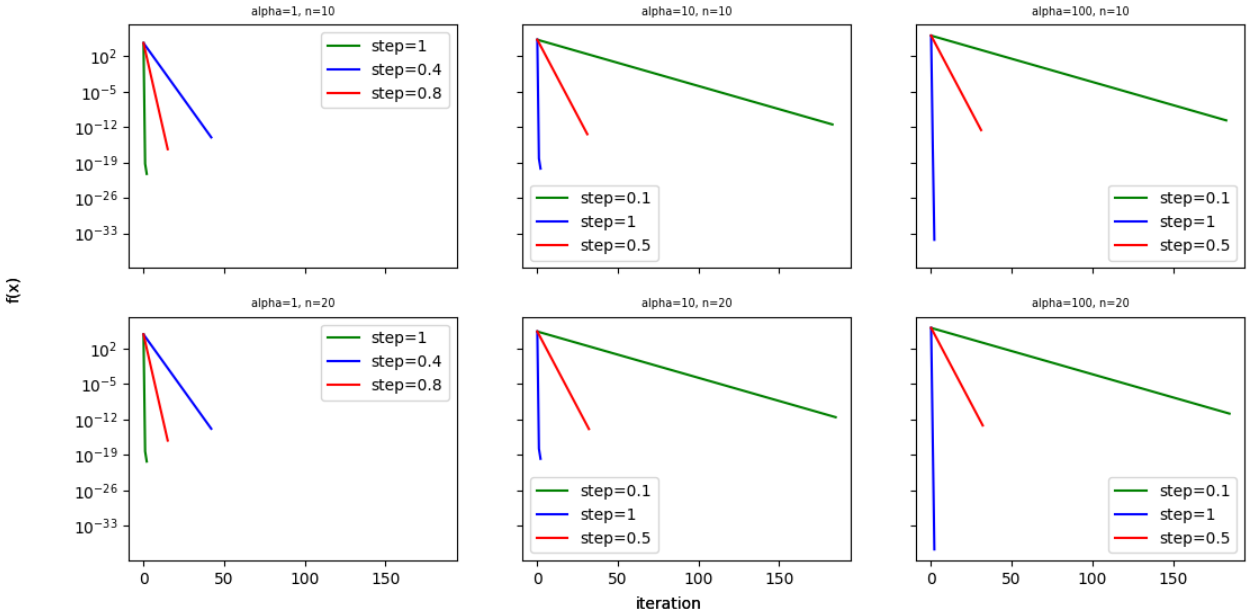
Krok równy 1 działa najlepiej, ponieważ wyliczony kierunek jest prawie równy wektorowi x, przez co już po jednej iteracji jesteśmy bardzo blisko minimum.

Inne wartości kroku również działają, jednak im są mniejsze, tym algorytm działa gorzej – wartości wyliczonego kierunku są mniejsze i nie przybliżają nas aż tak szybko jak z krokiem równym 1. Dla wartości rzędu 0.001 krok jest za mały, żeby znaleźć minimum w limicie iteracji.

Metoda Newtona z nawrotami

(wykresy pod analizą)

Dla zadanej funkcji celu dodanie nawrotów powoduje, że metoda Newtona działa równie dobrze lub gorzej niż w wersji oryginalnej. Dzieje się tak, ponieważ zmniejszenie wartości kroku akurat w tym przypadku nie jest potrzebne, a więc mniejszy krok zwiększa ilość iteracji.



Zmiana wymiarowości z n = 10 na n = 20 nie ma wpływu na ilość iteracji potrzebnych do znalezienia minimum funkcji, czyli rozwiązanie jest skalowalne.

Najlepsze wartości kroków:

Alpha	Metoda gradientu prostego	Metoda Newtona	Metoda Newtona z nawrotami
1	0.25	1	1
10	0.09	1	1
100	0.009	1	1

Czas działania

Pomiar czasu został wykonany dla najlepszych parametrów kroku dobranych przy analizie grafów. Poniższa tabela przedstawia czasu działania każdej z metod w sekundach oraz ilość iteracji przez nie wykonanych.

Losowa wartość początkowa nr 1: [74, 79, 87, -61, 88, 21, 71, -37, -39, 98]

Alpha	Metoda gradientu prostego		Metoda Newtona		Metoda Newtona z nawrotami	
1	0.25979761700000026	32	0.9358859559999999	2	1.4518547620000004	2
10	1.8705365019999993	98	0.9515927280000005	2	9.111538741	11
100	6.507097472000002	905	0.7808620220000009	2	1.5683905409999994	2

Losowa wartość początkowa nr 2: [-56, -84, -14, -46, 67, -64, 53, -7, 59, -76]

Alpha	Metoda gradientu prostego		Metoda Newtona		Metoda Newtona z nawrotami	
1	0.1714348090000002	31	0.7096387020000001	2	1.341722511	2
10	1.8705365019999993	98	0.7125902279999998	2	1.3222528530000002	2

100	5.711809302	889	0.795031024	2	7.873205014	11

Dla alpha = 1 najszybsza jest metoda gradientu prostego. Może być to spowodowane sposobem wyliczania hesjanu, co zajmuje dużo czasu. Jednak, jeśli chodzi o ilość powtórzeń, metoda Newtona potrzebuje ich 15 razy mniej niż metoda gradientu prostego. Zastosowanie nawrotów nie pomaga w zmniejszeniu ilości iteracji (ciężko poprawić wynik równy 2) a nawet zwiększa czas wykonania. Jest to spowodowane większą ilością obliczeń przy sprawdzeniu warunków nawrotów.

Przewagę czasową Newtona możemy zaobserwować przy alpha = 10 i alpha = 100. Wtedy ilość iteracji potrzebna w metodzie gradientu jest odpowiednio prawie 50 / 450 razy większa niż w przypadku Newtona. Z tego powodu, nawet jeśli wyliczanie hesjanu jest kosztowne czasowo, to powtórzenie tej operacji 2 razy jest szybsze od wielokrotnego powtarzania algorytmu gradientu. W eksperymentach możemy również zaobserwować, że czasami metoda Newtona z nawrotami potrzebuje większej ilości powtórzeń niż klasyczna wersja algorytmu. Jest to wynik zjawiska, które wspomniałam wcześniej – przy zadanej funkcji celu najlepszym krokiem jest taki równy 1, więc zmniejszanie go może co najwyżej nie zaszkodzić działaniu algorytmu. W przypadku, kiedy ilości iteracji są równe, metoda z nawrotami zajmie więcej czasu, ponieważ musi przy każdym powtórzeniu obliczyć warunek adaptacji kroku.

Jednak, jeśli wartość kroku ustawimy na większą od 1, to możemy zaobserwować lepsze wyniki przy nawrotach:

krok	Metoda Newtona	Metoda Newtona z nawrotami
1.1	11	6
2	5000 (algorytm osiągną limit iteracji)	8

Podsumowanie

Metoda gradientu prostego i metoda Newtona różnią się metoda wyznaczania kierunku szukania minimum funkcji. Obie używają do tego gradientu, a metoda Newtona dodaje obliczanie hesjanu funkcji celu. Dzięki temu, algorytm jest bardziej skuteczny, bo oprócz obserwowania kierunku zmiany funkcji, obserwuje też kierunek zmiany gradientu. Kolejną zaletą metody Newtona jest jego większa odporność na źle dobrany parametr kroku. Poprzez lepsze przewidywanie następnych wartości x, algorytm nie wpada tak łatwo w oscylacje jak metoda gradientu.

Dla niektórych ustawień parametrów funkcji celu algorytm gradientu prostego potrafi być szybszy, ale jest to spowodowane użyciem funkcji ***Hessian*** z ***numdifftools*** do obliczania hesjanu zamiast użycia aproksymacji. Jednak w ogólności metoda Newtona działa szybciej albo przynajmniej w mniejszej ilości iteracji.

Nawroty w metodzie Newtona pomagają poprawić wartość kroku, co powinno prowadzić do mniejszej ilości iteracji algorytmu. Jeśli krok jest dobrany optymalnie przy niektórych funkcjach celu, to zmniejszanie go może powodować odwrotne efekty.