Metoda Newtona-Rapsona

Michał Michniak

Styczeń 2023

1 Założenia wstępne

Dla uproszczenia zadania optymalizacji będę rozpatrywał problem gdzie funkcja celu jest opisana jako: $F(x): \mathbb{R}^n \to \mathbb{R}$ gdzie: $n \in \mathbb{N}$
Oznaczenia:

- $F(x): \mathbb{R}^n \to \mathbb{R}$ - funkcja celu
- $x_j \in \mathbb{R}^n$ - rozwiązanie w j-tej iteracji
- $\nabla F(x) \in \left[C^1(\mathbb{R}^n)\right]^n$ - gradient funkcji celu
- $\nabla^2 F(x) \in [f : \mathbb{R}^n \to \mathbb{R}]^{n \times n}$ - macierz Hessego funkcji celu (oznaczana też jako H aby nie mylić z laplasjanem)

Do przeprowadzenia algorytmu Newtona Rapsona funkcja celu musi posiadaść pewne własności:

- funkcja celu musi być co najmniej klasy C^2 tzn. musi posiadać pochodną 2-ego rzędu (a przynajmniej posiadać ją na pewnym badanym zbiorze rozwiązań)
- macierz Hessego funkcji celu musi być ściśle dodatnio określona na badanym obszarze.

2 Algorytm Newtona-Rapsona

Podstawowym celem metody jest iteracyjne rozwiązanie problemu:

$$f(\vec{x}) = \vec{0} \qquad gdzie \qquad f(x) : \mathbb{R}^n \to \mathbb{R}^n$$

podstawą algorytmu Newtona-Rapsona jest aproksymacja funkcji f za pomocą funkcji afinicznej dzięki rozwinięciu jej w szereg taylora:

$$\widetilde{f}(x_0 + p) = f(x_0) + \nabla f(x_0) p$$

gdzie $\nabla f(x_0)$ jest macierzą Jakobiego w punkcie x_0
dzięki tej aproksymacji możemy wyliczyć krok p przy którym funkcja aproksymująca osiąga wartość $\vec{0}$ pod warunkiem że macierz Jakobiego jest nieosobliwa:

$$p = -\left[\nabla f(x_0)\right]^{-1} f(x_0)$$

z uwagi na fakt że funkcja $\widetilde{f}$ jest tylko aproksymacją funkcji f operację należy powtarzać iteracyjnie aż do zadowalającej aproksymacji miejsca zerowego (wektora zerowego).

Warunkiem zbieżności algorytmu jest wymóg aby macierz Jakobiego była odwzorowaniem nieosobliwym oraz zwężającym czyli takim dla którego stała Lipschitza $L < 1$.
Warunek Lipschitza:

$$\|\nabla f(x)x - \nabla f(x)x'\| \leqslant L\|x - x'\| \quad gdzie: \quad L \in \mathbb{R}$$

3 Algorytm Newtona-Rapsona znajdywanie ekstremów lokalnych

Idea metody jest dość prosta. Korzystamy z lokalnej aproksymacji funkcji celu szeregiem Taylora wokół badanego punktu :

$$\widetilde{F}(x_0) = F(x_0) + \nabla^T F(x_0)(x - x_0) + \frac{1}{2}(x - x_0)^T \nabla^2 F(x_0)(x - x_0)$$

możemy zapisać gradient funkcji aproksymującej jako:

$$\nabla\widetilde{F}(x_0) = \nabla F(x_0) + \nabla^2 F(x_0)(x - x_0)$$

Jak można zauważyć problem sprowadza się do znalezienia:

$$\nabla F(\vec{x}) = \vec{0} \qquad gdzie \qquad \nabla F(x) : \mathbb{R}^n \to \mathbb{R}^n$$

czyli do typowego problemu rozwiązywalnego za pomocą metody Newtona-Rapsona.
Z założenia funkcja celu jest dobrze aproksymowana przez $\widetilde{F}(x)$ w pewnym bliskim otoczeniu dlatego warunek na zerowanie się gradientu funkcji celu można przybliżyć warunkiem na zerowanie się gradientu funkcji aproksymowanej w pewnym punkcie x_1:

$$0 = \nabla F(x_0) + \nabla^2 F(x_0)(x_1 - x_0)$$

jednak takie przybliżenie zwraca dokładny wynik tylko w przypadku kiedy funkcja aproksymowana jest formą kwadratową. Niestety większość badanych funkcji nie jest formami kwadratowymi dlatego aproksymacja nie będzie dokładna dla nie lokalnego otoczenia punktu x_0. Dlatego w następnym kroku należy powtórzyć aproksymacje oraz znaleźć nowy punkt zerowania się gradientu funkcji aproksymującej $\widetilde{F}(x)$ tym samym przybliżając się do pewnego punktu zbierzności algorytmu. Można wyprowadzić ogólny wzór na kolejne przyblirzenia rozwiązania algorytmu:

$$0 = \nabla F(x_k) + \nabla^2 F(x_k)x_{k+1} - \nabla^2 F(x_k)x_k$$
$$\nabla^2 F(x_k)x_{k+1} = -\nabla F(x_k) + \nabla^2 F(x_k)x_k$$
$$x_{k+1} = x_k - \left[\nabla^2 F(x_k)\right]^{-1} \nabla F(x_k)$$

odwracalność macierzy Hessego jest zapewniona z warunku ścisłej dodatniej określoności.
Dodatkowo aby poprawić zbieżność metody ale i wykluczyć maksima lokalne z punktów zbieżności metody (wyjątkem są przypadki kiedy punkt początkowy jest maksimum lokalnym) można dodać przeszukiwanie na kierunku:

$$x_{k+1} = x_k - \mu\left[\nabla^2 F(x_k)\right]^{-1} \nabla F(x_k)$$

gdzie $\mu = \min_{\mu}\left(F(x_k - \mu\left[\nabla^2 F(x_k)\right]^{-1} \nabla F(x_k))\right)$

UWAGA!
Metoda Newtona-Rapsona nie gwarantuje znalezienia lokalnego minimum a tylko aproksymacji punktu zerowania się gradientu funkcji celu (np. punktu siodłowego).

4 Zbieżność

Zbieżność metody jest zagwarantowana dla macierzy Hessego ściśle dodatnio określonej. Oraz jeżeli macierz Hessego lokalnie spełnia warunek Lipschitza:

$$\left\|\nabla^2 F(x) - \nabla^2 F(x')\right\| \leqslant L \left\|x - x'\right\| \quad gdzie: \quad L \in \mathbb{R}$$

to rząd zbierzności metody jest równy 2.

5 Warunki Stop-u

- $\|x_{k+1} - x_k\| \leqslant \varepsilon$
- $det(\nabla^2 F(x_k)) = 0$ - nie da się wykonać kroku z uwagi na brak odwrotności macierzy Hessego
- $\nabla F(x_k)$ - osiągnięto miejsce zerowe gradientu
- osiągnięcie maksymalnej liczby iteracji (N_{max})

6 Implementacja algorytmu

```
function [x_lst] = newton(f,x_,N_max,epsilon)
%newton - metoda newtona dla problemu optymalizacji
% dane wejściowe:
% f - funkcja celu dla sympolicznych wartości wektora syms
% x_ - punkt początkowy
% N_max - maksymalna liczba iteracji
% epsilon - warunek stopu

i = 1;
% liczenie gradientu:
f_diff = gradient(f);
% zamiana gradientu funkcji symbolicnej na funkcje zwykłą
f_diff_func = matlabFunction(f_diff);
% zamiana funkcji symbolicnej na funkcje zwykłą
f_func = matlabFunction(f);
% rozpakowanie argumentów funkcji:
f_diff_func_vect = @(x)f_diff_func(x(1),x(2));
f_func_vect = @(x)f_func(x(1),x(2));
% symboliczne policzenie macierzy Hessego
H = hessian(f);
% zamiana macierzy Hessego z formy symbolicnej na funkcje zwykłą
H_func = matlabFunction(H);
H_func_vect = @(x)H_func(x(1),x(2));
%inicjalizacja wartości początkowej:
x = [];
x(:,end+1) = x_;
%warunek stopu na liczbę iteracji
while i<N_max
    % warunek stopu na wyznacznik macierzy Hessego
    if det(H_func_vect(x(:,i))) == 0
        x_lst = x;
        return
    end
```

Rysunek 1: Implementacja metody cz. 1

```
        % warunek stopu na gradient funkcji celu
        if f_diff_func_vect(x(:,i)) == [0;0]
            x_lst = x;
            return
        end
        % policzenie kierunku do optymalizacji na kierunku
        d = inv(H_func_vect(x(:,i)))*f_diff_func_vect(x(:,i));
        % funkcja do optymalizacji jednowymiarowej
        f_func_direction = @(step)(f_func_vect(x(:,end)-step*d));
        % optymalizacja na kierunku (dobranie suboptymalnego kroku)
        h = fminsearch(f_func_direction,0);
        i=i+1;
        x(:,end+1) = x(:,end) - h*d;
        % warunek stopu na przyrost argumentów
        if sqrt(sum((x(:,end)-x(:,end-1)).^2)) < epsilon
            x_lst = x;
            return
        end
    end
    x_lst = "error";
    end
```

Rysunek 2: Implementacja metody cz. 2

7 Przykładowe przebiegi algorytmu

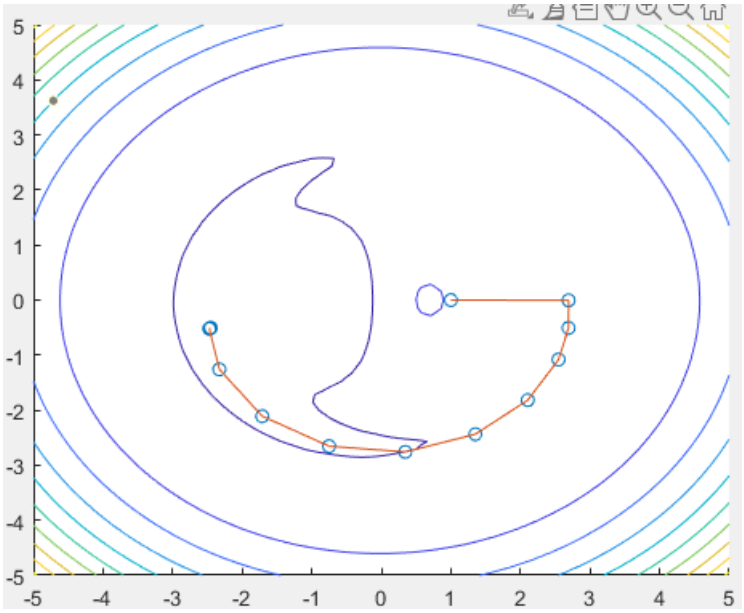
7.1 Funkcja 1

```
syms x1 x2
f = 0.3*x1+0.1*x2+(-3.5+0.5*x1.^2+0.5*x2.^2).^2+100*x1 .* exp(-x1.^2 - x2.^2)
```

Rysunek 3: wzór funkcji

```
wynik1 = newton(f,[1;0],300000,1e-6)
```

Rysunek 4: pierwsze parametry algorytmu



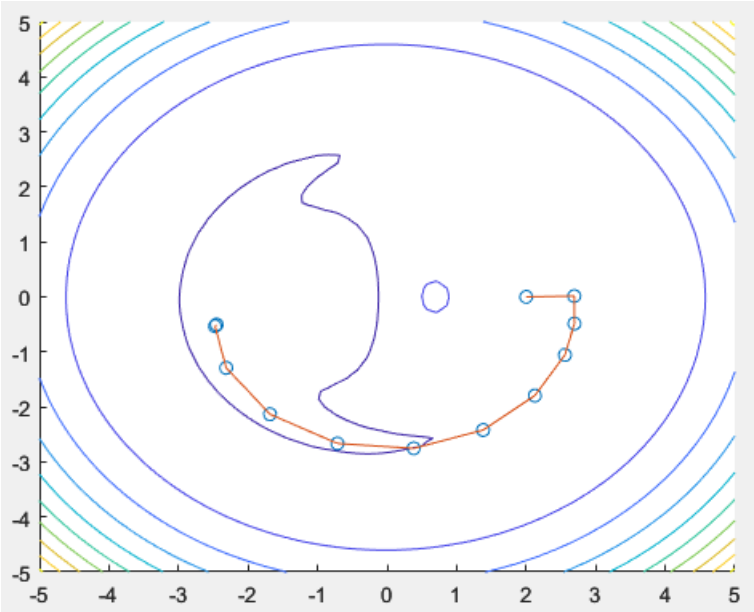
Rysunek 5: pierwszy przebieg algorytmu

```
ans = 2×1
   -2.453968371375165
   -0.502464774893187
```

Rysunek 6: wynik algorytmu dla pierwszego przebiegu

```
wynik2 = newton(f,[2;0],300000,1e-6)
```

Rysunek 7: drugie parametry algorytmu



Rysunek 8: drugi przebieg algorytmu

```
ans = 2×1
   -2.453968371375165
   -0.502464774893187
```

Rysunek 9: wynik algorytmu dla drugiego przebiegu

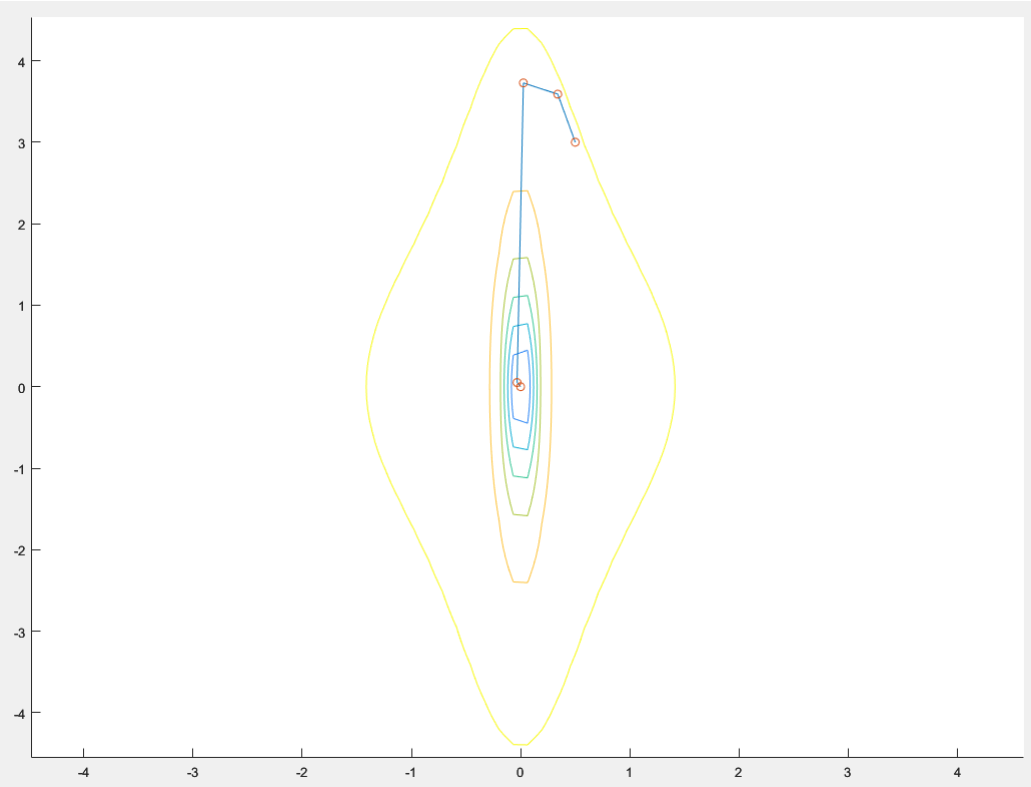
7.2 Funkcja 2

```
syms x1 x2
f = (x1.^2+x2.^2)-(400)/(100*x1.^2+x2.^2+1)
```

Rysunek 10: wzór funkcji

```
wynik1 = newton(f,[0.5;3],300000,1e-6)
wynik1(:,end)
```

Rysunek 11: pierwsze parametry algorytmu



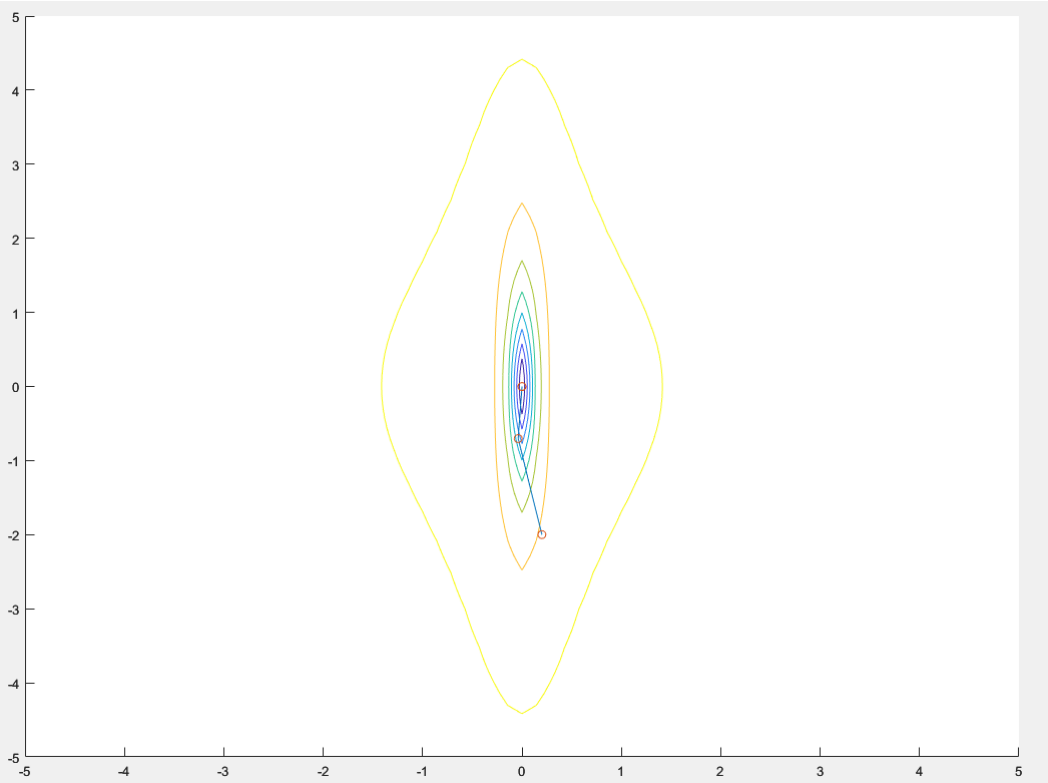
Rysunek 12: pierwszy przebieg algorytmu

```
ans = 2×1
10^-12 ×
    -0.007584567743721
    -0.912528285899016
```

Rysunek 13: wynik algorytmu dla pierwszego przebiegu

```
wynik2 = newton(f,[0.2;-2],300000,1e-6)
wynik1(:,end)
```

Rysunek 14: drugie parametry algorytmu



Rysunek 15: drugi przebieg algorytmu

```
ans = 2×1
10^-12 ×
   -0.007584567743721
   -0.912528285899016
```

Rysunek 16: wynik algorytmu dla drugiego przebiegu

8 Wnioski

Metoda Newtona-Rapsona z pewnością jest wartą uwagi metodą optymalizacji z uwagi na korzystanie z 2-giej pochodnej funkcji celu. Zaletą stosowania tej metody jest jej niesamowita zbieżność 2-giego rzędu do punktów w których zeruje się hesjan lub gradient. Niestety metoda wymaga silnych założeń co do gładkości i różniczkowalności funkcji celu. Jeżeli chodzi o podejście implementacyjne to metoda wymaga policzenia macierzy Hessego na szczęście biblioteka symboliczna w matlabie świetnie się w tym przypadku sprawdza lecz wymaga dużego nakładu obliczeniowego. Niestety inne o wiele szybsze języki nie posiadają jeszcze popularnych bibliotek do obliczeń symbolicznych więc metoda jest trudna w implementacji dla przypadku ogólnego. Inną wadą algorytmu jest możliwość wystąpienia źle uwarunkowanych macierzy Hessego co może doprowadzić do pogorszenia zbieżności algorytmu.