Semestr zimowy 2021

# Pierwsze laboratorium z przedmiotu Wprowadzenie do sztucznej inteligencji


***Autor***
Łukasz Jaremek
310710

***Data***
październik 2021

Praca wykonana samodzielnie.

# Użyte technologie

Język:
- Python 3.8.8

Biblioteki ponad standardowe:
- matplotlib 3.4.2
- scipy 1.6.3

# Metoda najszybszego wzrostu

## Problem:

Znajdowanie minimum funkcji przy pomocy gradientu.
Gradient wymaga znajomości pierwszych pochodnych funkcji, więc zostaną one policzone.
Dodatkowo, aby mieć pewność poprawności wyznaczania minimum funkcji, policzymy lokalizację tego punktu.

Wzór funkcji:

$f(x, y) = (x + 2y - 7)^2 + (2x+y - 5)^2$ , $x \in [-5, 5]$ $y \in [-5, 5]$

$f(x, y) = (x^2 + 4y^2 + 49 + 4xy - 14x - 28y) + (4x^2 + y^2 + 25 + 4xy - 20x - 10y) =$
$= 5x^2 + 5y^2 + 8xy - 34x - 38y + 74$

## Liczenie minimum funkcji:

Liczenie pochodnych:

| | |
|---|---|
| Pochodna po x: | Pochodna po y: |
| $fx(x, y) = 10x + 8y - 34$ | $fy(x, y) = 10y + 8x - 38$ |
| Pochodna po x po x: | Pochodna po y po y: |
| $fxx(x, y) = 10$ | $fyy(x, y) = 10$ |
| Pochodna po x po y: | Pochodna po y po x: |
| $fxy(x, y) = 8$ | $fyx(x, y) = 8$ |

Liczenie punktów podejrzanych o bycie ekstremum:

$$\begin{cases} fx(x, y) = 0 \\ fy(x, y) = 0 \end{cases}$$

$$\begin{cases} 10x + 8y - 34 = 0 \\ 10y + 8x - 38 = 0 \end{cases}$$

$$\begin{cases} x = 1 \\ y = 3 \end{cases}$$

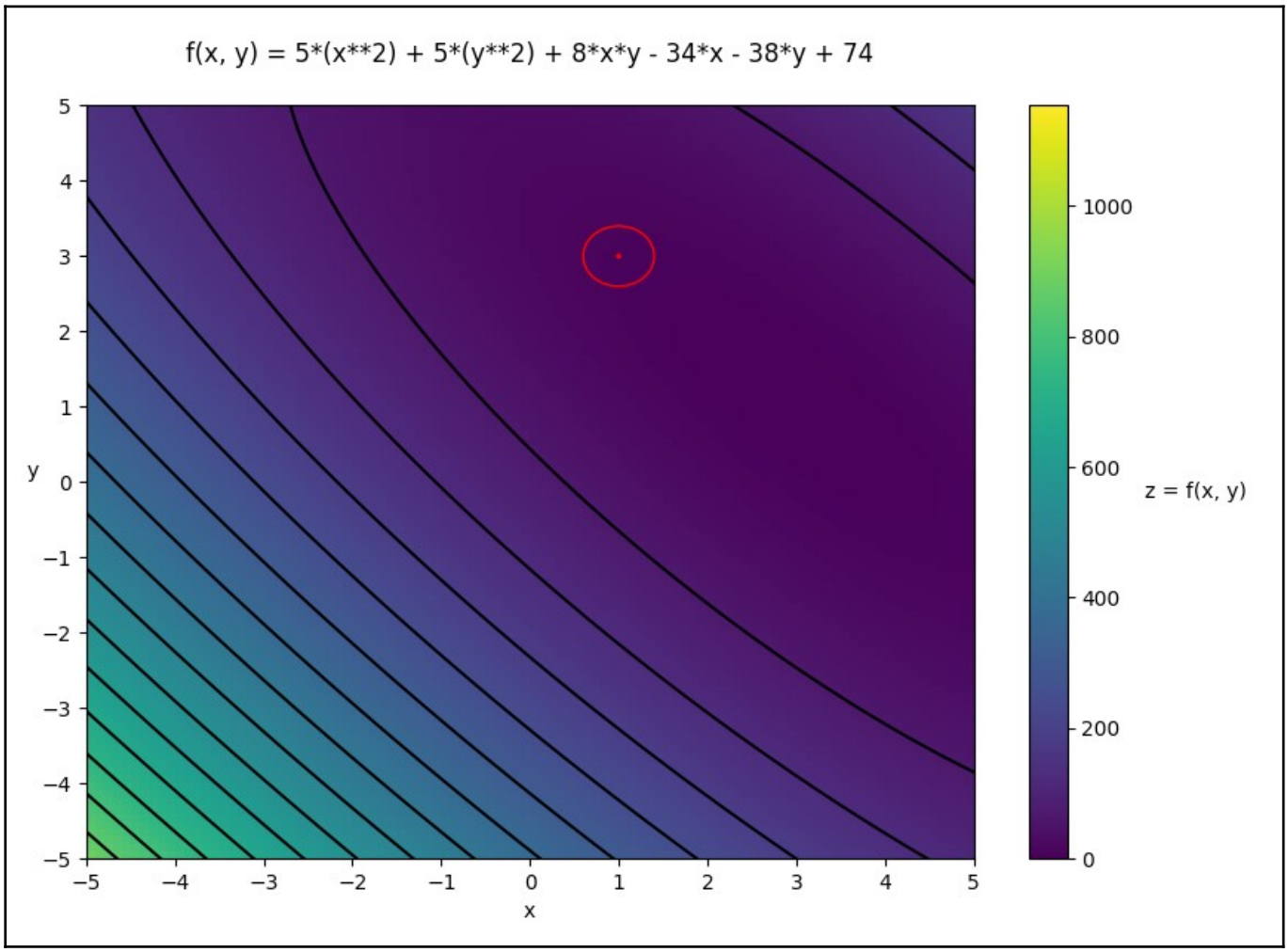
Punkt podejrzany o bycie ekstremum: (x, y) = (1, 3)

Sprawdzanie czy punkt jest ekstremum, jeśli tak, to jakim:

$$W = \begin{vmatrix} 10 & 8 \\ 8 & 10 \end{vmatrix} = 100 - 64 = 36 > 0 \Rightarrow minimum\,lokalne$$

Znając już zoptymalizowany wzór funkcji oraz minimum lokalne, sprawdźmy jak wygląda wykres funkcji.

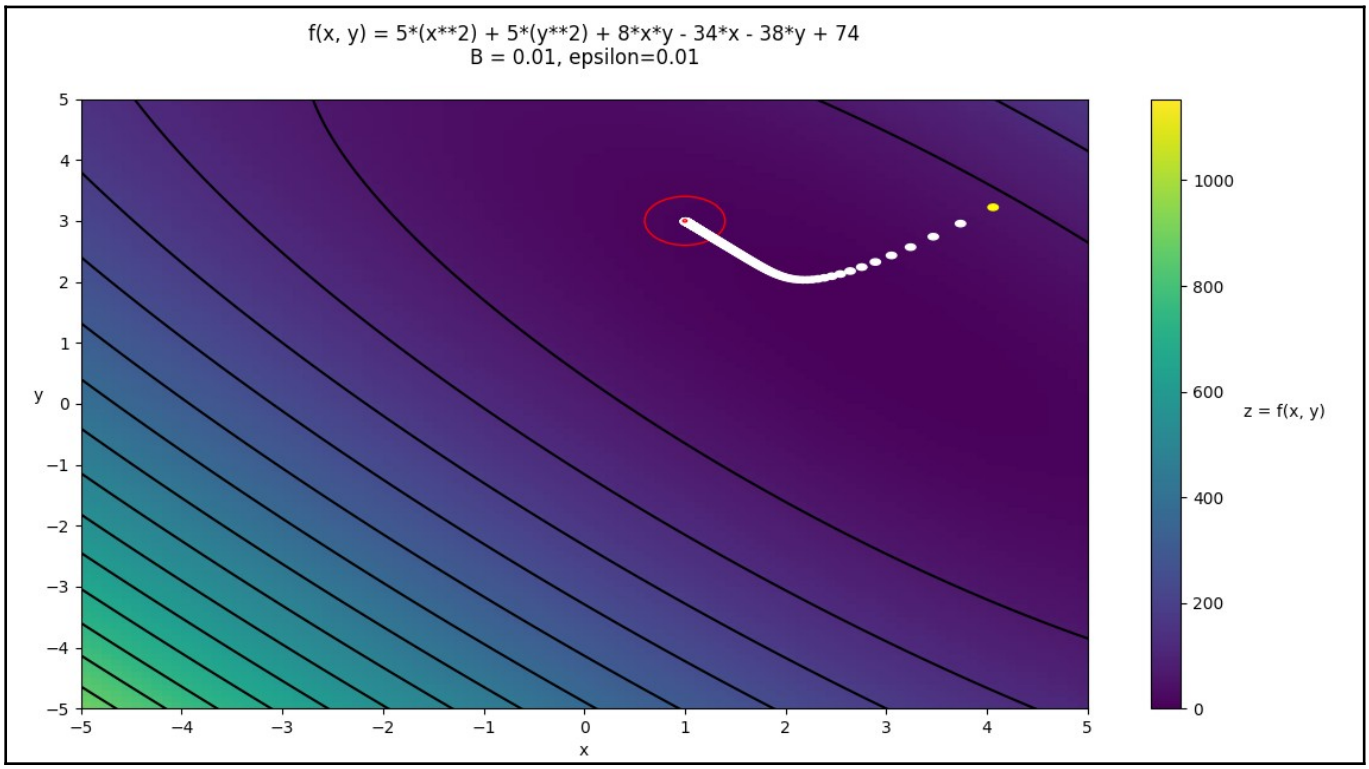
*czerwone kółko z kropką to punkt (x, y) = (1, 3)*




Następnie przy pomocy gradientu obliczamy punkty coraz bliższe minimum zgodnie ze wzorem:

```
end_x = 1
end_y = 3
x = random_float(-5, 5)
y = random_float(-5, 5)
while not stop:
   dx, dy = gradient(x, y)
   x = x - dx*B
   y = y - dy*B
   stop =  √((x − end_x)² + (y − end_y)²) > epsilon
```

Zaznaczając na żółto pseudolosowy punkt startowy oraz na biało wszystkie kolejne kroki otrzymujemy następujący wynik:

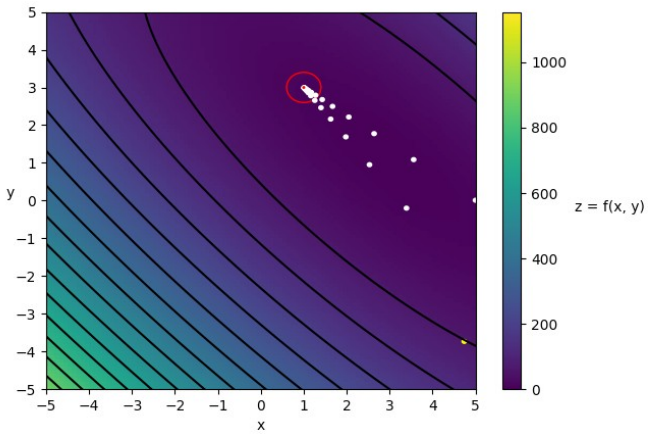



Przy takich parametrach B oraz epsilon obliczam średni czas znalezienia minimum funkcji oraz średnią liczba iteracji, potrzebnych do jego odnalezienia  - 10 000 prób.
Średni czas: 0.00065 sekundy
Średnia liczba iteracji: 256

Sprawdzam, jak wygląda średnia liczba iteracji przy zmianie parametru B.
W badaniu nie uwzględniam czasu ponieważ zawsze był on zbyt mały, aby dało się go zbadać.

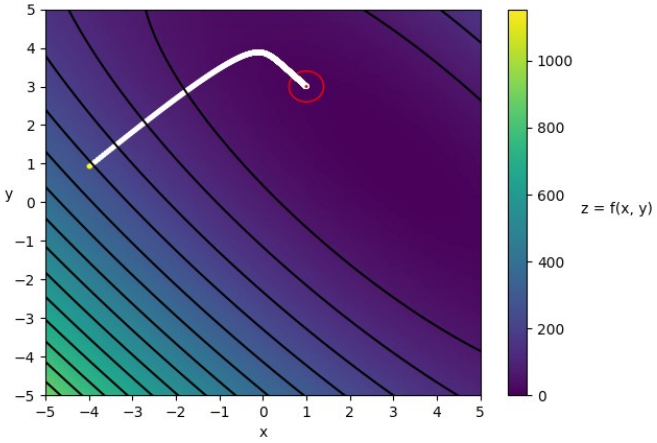
| B = 0.1 | Epsilon = 0.01 |
|---|---|
| Iteracje: 26 | |

Przykładowy wykres:




| B = 0.001 | Epsilon = 0.1 |
|---|---|
| Iteracje: 2593 | |

Przykładowy wykres:




Zmiana B jest bardzo zauważalna, im większy tym większy jest „rozrzut” kolejnych kroków, natomiast im mniejszy tym są one coraz bliżej siebie oraz wyraźnie rośnie ich liczba.

# Metoda Newtona-Raphsona

## Problem:

Znajdowanie minimum funkcji przy pomocy gradientu oraz auto-kalibracji parametru B. Wcześniejsza metoda posiadała stanowczą wadę w postaci parametru, który trzeba było samodzielnie kalibrować. Problem ten rozwiązuje Metoda Stycznych, która parametr B zastępuje odwróconym Hesjanem funkcji f.

## Obliczanie Hesjanu:

Hesjan:

$$H=\begin{vmatrix} fxx & fxy \\ fxy & fyy \end{vmatrix}$$

Żeby wyznaczyć odwrotność macierzy 2x2 należy sprawdzić czy jej wyznacznik jest niezerowy,

$$H=\begin{vmatrix} 10 & 8 \\ 8 & 10 \end{vmatrix}=100-64=36>0$$

Teraz korzystając ze wzoru na odwracanie macierzy 2x2 obliczam:

$$H^{-1}=\frac{1}{det(H)}\cdot\begin{vmatrix} d & -b \\ -c & a \end{vmatrix}$$

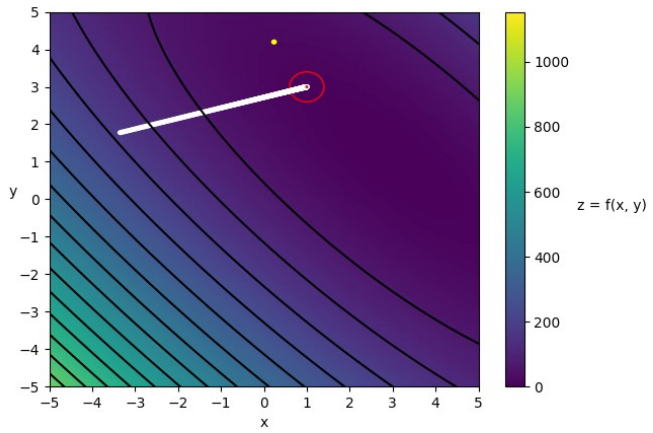
$$H^{-1}=\frac{1}{36}\cdot\begin{vmatrix} 10 & -8 \\ -8 & 10 \end{vmatrix}=\begin{vmatrix} \frac{5}{18} & \frac{-2}{9} \\ \frac{-2}{9} & \frac{5}{18} \end{vmatrix}$$

Posiadając już tą daną korzystam z algorytmu:

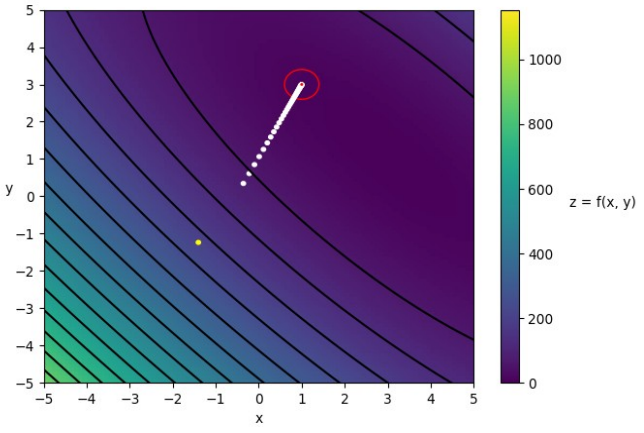
```
end_x = 1
end_y = 3
x = random_float(-5, 5)
y = random_float(-5, 5)
while not stop:
   grad = gradient(x, y)
   x, y +=  H^-1*grad
   stop =  sqrt((x-end_x)^2+(y-end_y)^2) > epsilon
```

Sprawdzam, jak zachowuje się algorytm dla różnych parametrów B:

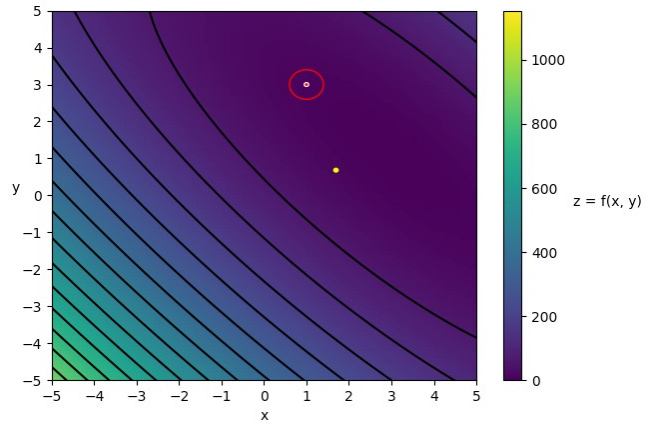
B = 0.01
Średnia liczba iteracji = 600



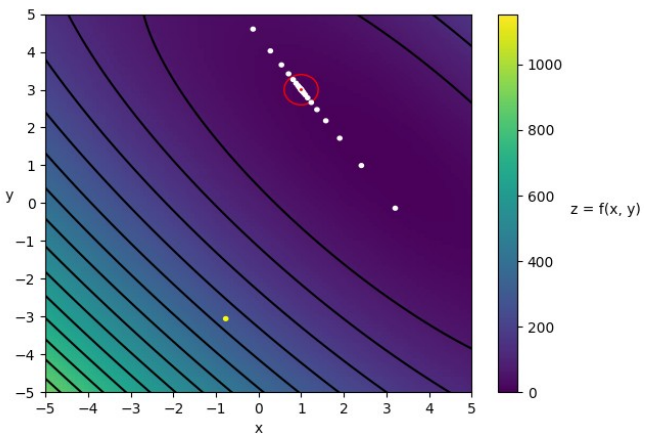

B = 0.1
Średnia liczba iteracji = 57



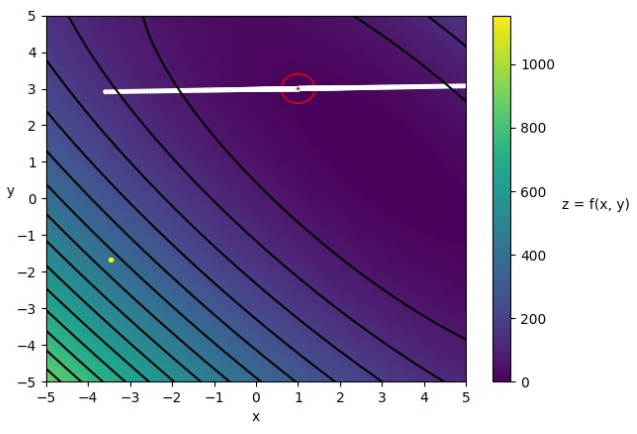

B = 1.0
Średnia liczba iteracji = 1



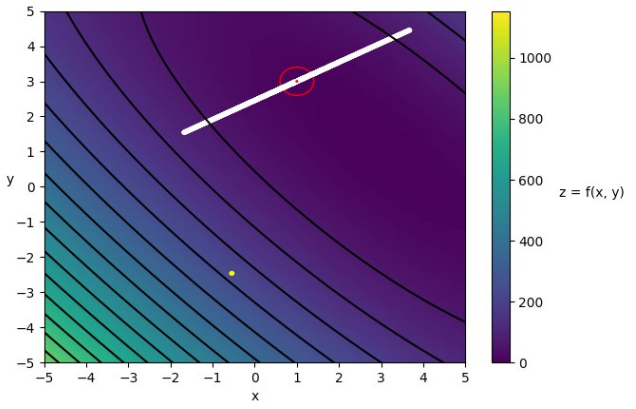

B = 1.8
Średnia lizba iteracji = 30




B = 1.99
Średnia liczba iteracji = 580




B = 1.999
Średnia liczba iteracji = 6650



Na podstawie powyższych wykresów można stwierdzić, że im mniejsze B, tym mniejszymi krokami zbliżamy się do minimum. Przy B = 1 osiągamy idealne trafienie (liczba kroków zawsze = 1), wynika to z faktu, że hesjan jest w rzeczywistości auto-korektującym się parametrem B. Natomiast gdy B > 1 i coraz większe, tym robią się coraz większe „przeskoki” nad punktem minimum. Dla B = 2 komputer nie był w stanie znaleźć punkt minimalny.

# Wnioski z laboratorium

Każda z metod jest w czymś lepsza lub gorsza od drugiej, w poniższej tabeli zamieściłem porównanie obu:

| Metoda najszybszego wzrostu | Metoda Newtona |
|---|---|
| Łatwa matematyka | Trudniejsza matematyka |
| Okrężna droga do punktu minimum | Prosta droga do punktu minimum |
| Nawet setki skoków do punktu minimum | Jeden skok do punktu minimum |
| Parametr B wymagający kalibracji | Hesjan, który zastępuje parametr B |

Jak widać, jedyną a zarazem największą zaletą pierwszej metody jest prosta matematyka, co za tym idzie, implementacja jest bardzo prosta i szybka, niestety przez swoją prostotę jest nieoptymalny więc będzie złym rozwiązaniem dla osób, które potrzebują wysokiej wydajności. Trzeba jednak spojrzeć na te wnioski z dystansem, ponieważ badania zostały przeprowadzone na jednej, a do tego dość prostej funkcji, co nie jest odpowiednim zbiorem danych aby wysunąć ostateczną opinię.