Skład zespołu:
Mateusz Winnicki

WSI ĆWICZENIE 1

Cel ćwiczenia

Pierwszym tematem ćwiczeń jest zagadnienie przeszukiwania i związane z tym podejścia. Naszym celem będzie minimalizacja funkcji celu i porównanie wyników dla metody najszybszego spadku gradientu i metody Newtona.

Funkcją celu jest funkcja bananowa Rosenbrocka postaci:

$f(x,y) = (1-x)^2 + 100 * (y-x^2)^2, \qquad dla\ x,y\ \in [-5,5]$

Obie badane metody w podstawowych wersjach mogą okazać się bezskuteczne dla określonych warunków wejściowych, do których należą:

- punkt początkowy (init_x, init_y)
- współczynnik β obu metod
- maksymalna liczba iteracji
- współczynnik ε, kończący przebieg algorytmu, jeżeli MSE (będący kwadratem różnicy znalezionego minimum i jego wartości oczekiwanej) jest od niego niższy

W przypadku niepowodzenia naszym celem będzie znalezienie interpretacji niefortunnego przebiegu metody.

Przebieg ćwiczenia

Do obydwu algorytmów potrzebna jest znajomość gradientu oraz hesjanu funkcji celu.

Gradient funkcji Rosenbrocka

$$\nabla f = \begin{pmatrix} 400x^3 - 400xy + 2x - 2 \\ 200y - 200x^2 \end{pmatrix}$$

Hesjan funkcji Rosenbrocka

$$\nabla^2 f = \begin{pmatrix} 1200x^2 - 400y + 2 & -400x \\ -400x & 200 \end{pmatrix}$$

Gradient oraz hesjan zostały obliczone ‘ręcznie’, natomiast przekształcenia hesjanu potrzebne w metodzie Newtona obliczamy za pomocą biblioteki *numpy*.

Algorytmy zostały zaimplementowane w Pythonie, z użyciem następujących bibliotek:

- *numpy* (do wykonywania operacji na macierzach)
- *argparse* (do parametryzacji obu metod z konsoli)
- *timeit* (do pomiaru czasu wykonywania algorytmów)
- *matplotlib* (do tworzenia wykresu konturowego kolejnych punktów x, y)

Aby uruchomić program należy to zrobić z parametrami określającymi: punkt początkowy, wartość współczynnika β, wartość współczynnika ε, maksymalną liczbę iteracji oraz wybrany algorytm – *gd* dla metody najszybszego spadku gradientu lub *nm* dla metody Newtona.

Dodanie do linii *-p* zakończy program nakreśleniem przebiegu poszczególnych punktów obliczanych przez algorytmy.

Dodanie do linii *-d* wykona program wyświetlając wynik obliczeń kolejnych punktów (x, y) w każdej epoce oraz MSE dla rozwiązania funkcji Rosenbrocka f(x, y).

Przykład:
> *py .\lab1_optimization_algorithms.py -x 4 -y 2 -b 0.02 -e 1e-12 -i 15000 -a nm -d -p*

Wyniki ćwiczenia i wnioski

Zacznijmy od omówienia kilku zestawów danych wejściowych, dla których obydwa algorytmy działają poprawnie i znajdują minimum lub rozwiązanie o MSE mniejszym od ε[1]. Skupimy się na porównaniu liczby iteracji oraz czasu wykonywania poszczególnych algorytmów (mierzonego jako średni czas 10 pomiarów danej metody).

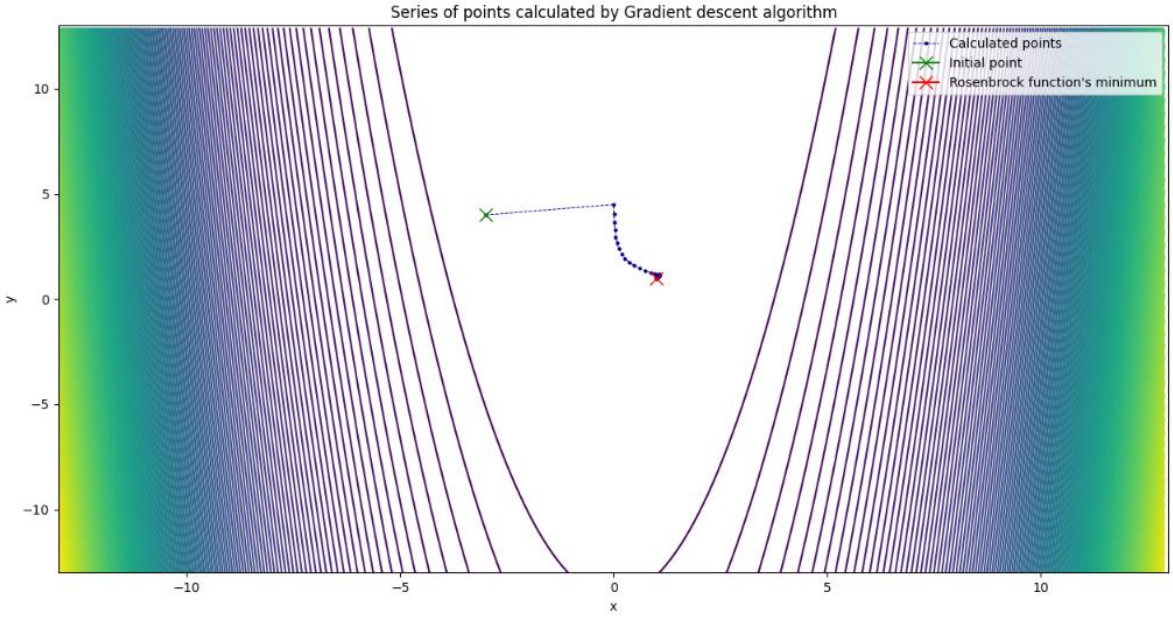
Punkt początkowy	(2; 2)	(2; 2)	(-3; 4)	(1,15; 1,55)	(2,13; -4)	(-1,25; -4,2)	(-1,25; -4,2)
Współczynnik beta	0,0005	0,001	0,0005	0,001	0,0005	0,0005	0,0009
Metoda najszybszego spadku gradientu							
Liczba iteracji	34564	15257	22217	14588	28619	28622	15895
Czas trwania algorytmu [s]	2,64	1,16	1,67	1,12	2.33	2,2	1,25
Metoda Newtona							
Liczba iteracji	25816	12914	30652	9044	28962	29615	16486
Czas trwania algorytmu [s]	20,76	10,53	25,51	7,09	23,51	23,43	13,46

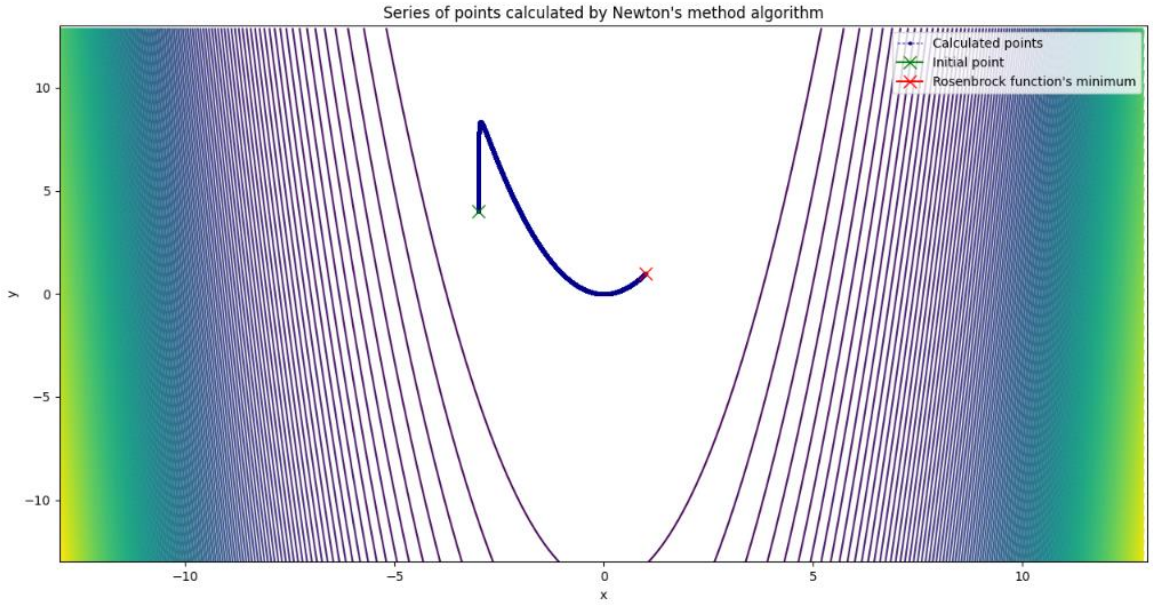
Jak widać w obydwu metodach zwiększenie wartości współczynnika beta dla tego samego punktu skraca czas wykonywania oraz zmniejsza liczbę iteracji. Wynika to z faktu, że rozpoczynamy algorytm robiąc większe kroki między kolejnymi punktami, dzięki czemu jesteśmy w stanie szybciej znaleźć minimum.

[1] W praktyce bardzo trudnym okazuje się wyliczenie idealnego rozwiązania (1, 1). Im bliżej znajdujemy się celu, tym mniejsze otrzymujemy różnice między kolejnymi punktami w obu metodach. Niektóre przebiegi były w stanie znajdować rozwiązania z dokładnością nawet 1e-54, jednak na potrzeby ćwiczenia przyjmujemy ε równy 1e-12 i z reguły oczekujemy zakończenia programu na rozwiązaniu o dokładności tego rzędu.

Metoda Newtona czasem pozwala nam znaleźć rozwiązanie w mniejszej liczbie iteracji, niż metoda najszybszego spadku gradientu, jednak jej czas wykonywania jest zdecydowanie dłuższy[2], w związku z czym może być ona mniej praktycznym wyborem przy minimalizacji funkcji celu. Długi czas wykonywania wynika z faktu przeprowadzania większych obliczeń przez metodę Newtona.

Przykładowe wizualizacje obu metod dla punktu (x, y) = (-3, 4) i β = 0,0005





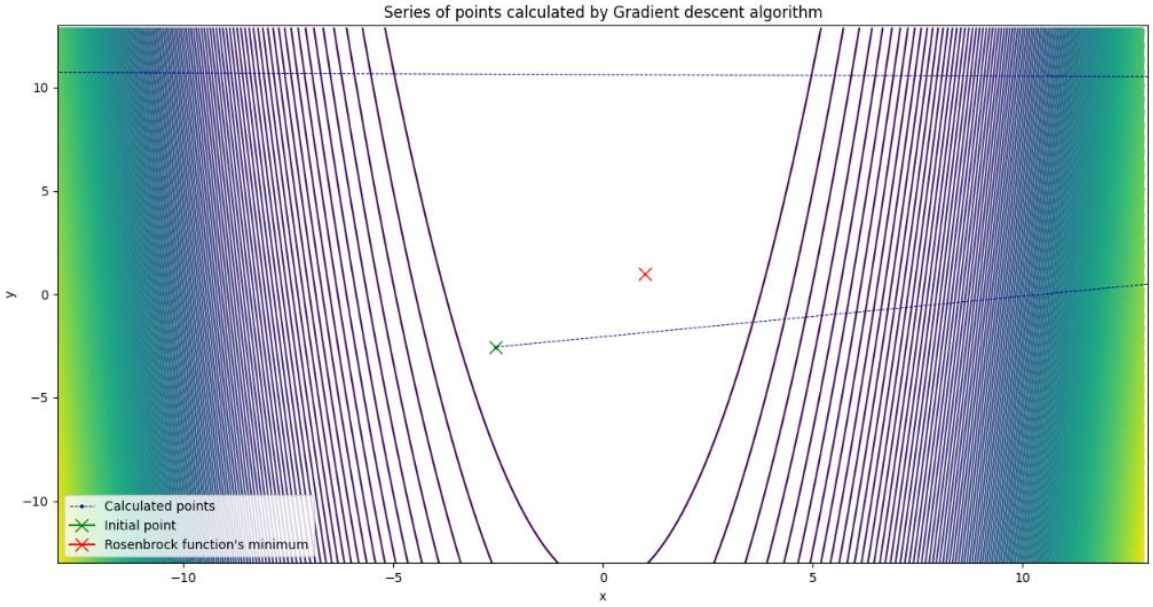
[2] Warto odnotować, że implementacja metody Newtona wykorzystuje dobrodziejstwa biblioteki NumPy przy operacjach na macierzy Hessego. Rozwiązania dostarczane przez ten moduł przyspieszają obliczenia, dlatego gdybyśmy z niego nie korzystali, czas trwania drugiego badanego algorytmu byłby jeszcze dłuższy.

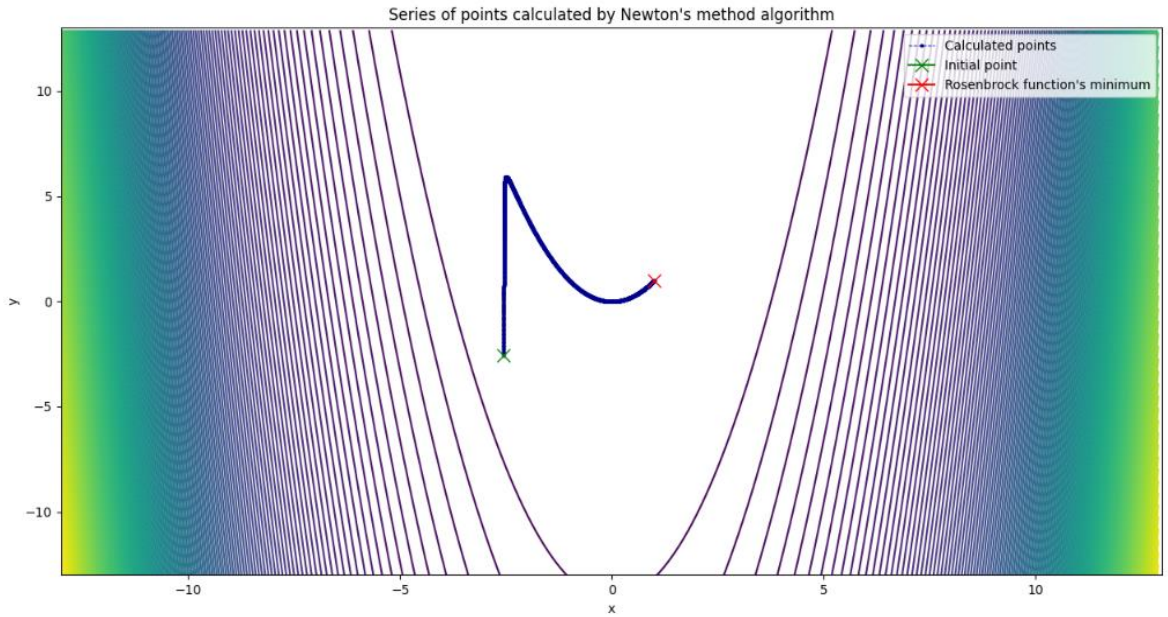
Następnym badanym wariantem będą zestawy parametrów, dla których działa jedynie metoda Newtona.

Punkt początkowy	(-2,55; -2,55)	(-4; -2)	(3,3; 3,3)
Współczynnik beta	0,007	0,01	0,1
Metoda Newtona			
Liczba iteracji	2376	1868	178
Czas trwania algorytmu [s]	2,2	1,52	0,15

Metoda najszybszego spadku gradientu ma trudności ze znalezieniem rozwiązania dla współczynników parametru β rzędu ok. części setnych lub większych. Powodem tego, są za duże kroki wykonywane na początku algorytmu, przez co łatwo jest ‘przeskoczyć’ punkt docelowy.

Przykładowe wizualizacje obu metod dla punktu (x, y) = (-2,55; -2,55) i β = 0,007





Dla punktu początkowego (x, y) = (-5; -5) i β = 2 obydwa algorytmy nie spełniają swojego zadania.

Dla metody najszybszego spadku gradientu powodem jest osiągnięcie ogromnych wartości już w drugiej iteracji.

```
epoch: 0, (x, y): (-80011.0, 8005.0), MSE: 1.67955958139097e+43
epoch: 1, (x, y): (4.0976847084286086e+17, 2560700854405.0), MSE: 7.948923411562181e+144
```

Dla metody Newtona jest zapętlenie się algorytmu pomiędzy dwoma punktami.

```
epoch: 4157, (x, y): (5.0000008311847814, 5.000008308525324), MSE: 1601280257.596014
epoch: 4158, (x, y): (4.998001330644655, 44.98001330644652), MSE: 1601277697.8521345
epoch: 4159, (x, y): (5.000000831584586, 5.000008312521743), MSE: 1601280257.596792
epoch: 4160, (x, y): (4.998001331044264, 44.98001331044263), MSE: 1601277697.8529065
epoch: 4161, (x, y): (5.000000831984389, 5.000008316518198), MSE: 1601280257.597552
epoch: 4163, (x, y): (5.000000832384194, 5.000008320514603), MSE: 1601280257.5983334
epoch: 4164, (x, y): (4.998001331843472, 44.98001331843471), MSE: 1601277697.854441
epoch: 4165, (x, y): (5.000000832783995, 5.000008324511043), MSE: 1601280257.5990922
```

Podsumowanie

Metoda Newtona wygląda na pierwszy rzut oka na bardziej uniwersalny oraz szybszy (co do liczby iteracji) algorytm. Wymaga on jednak znajomości drugich pochodnych funkcji oraz odwracania macierzy, przez co czas trwania jego obliczeń w porównaniu z metodą najszybszego spadku gradientu może okazywać się nieopłacalnie długi. W bardziej zaawansowanych algorytmach sztucznej inteligencji ciężar obliczeń jaki niesie ze sobą metoda Newtona, może deklasować ja w porównaniu z innymi rozwiązaniami.