WSI – ćwiczenie 1.

Zagadnienie przeszukiwania

I podstawowe podejścia do niego

Denys Fokashchuk

Funkcje do badania:

$$f(x) = \sin(\pi x) + x^2$$
$$g(x) = 5e^2 - 4ex_1 + x_1^2 + 2ex_2 + x_2^2$$

Ich gradienty:

$$\nabla f(x) = \pi\cos(\pi x) + 2x$$
$$\nabla g(x) = \begin{bmatrix} 2(x_1 - 2e) \\ 2(x_2 + e) \end{bmatrix}$$

Treść ćwiczenia

1. Narysować funkcje f(x) i g(x). Która z funkcji będzie „łatwiejsza“ do optymalizacji?
2. Zaimplementować algorytm najszybszego spadku oraz zastosować go do znalezienia minimum funkcji f i g.
3. Zbadać wpływ rozmiaru kroku dla różnych (losowych) punktów początkowych. Jak często udaje się zlokalizować minimum globalne?

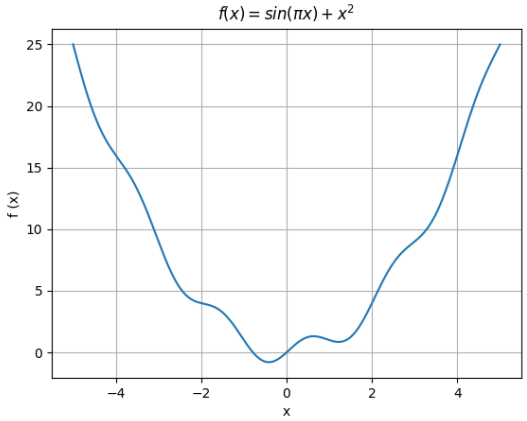
Uwagi ogólne

Postarałem się podzielić program w taki sposób, żeby była spójność i jasność. Jednak, moim zdaniem, niektóre momenty należy wytłumaczyć:

- jeśli nazwa funckji lub pliku kończy się na *_fx, to oznacza, że dotyczy to wyłącznie funkcji f(x), jeśli *_gx, to anologicznie g(x).
- Plik *test_data.py* mieści dwie listy z danymi, które zostały wybrane arbitralnie w celu łatwego analizowania różnych parametrów dla algorytmu najszybszego spadku.
- Jeśli użytkownik zamierza badać jak zachowuje się algorytm najszybszego spadku dla funkcji f(x), to należy uruchomić program *main_fx.py* i odpowiednio dodać lub usunąć komentarz na niektórych linijkach. Analogicznie dla funkcji g(x) - skrypt *main_gx.py*.

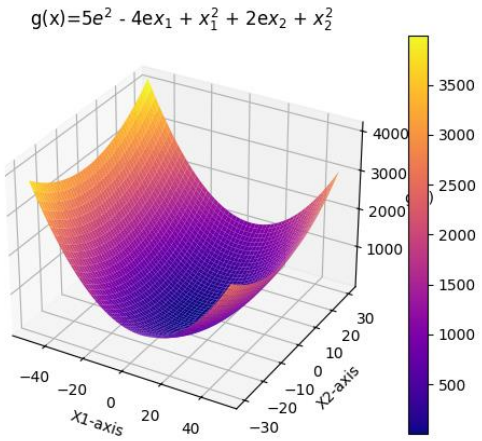
Zadanie 1.1 Narysować funkcje f(x) i g(x)

Ponieważ wartość funkcji f(x) bardzo szybko wzrasta wraz z zwiększaniem wartości bezwzględnej parametru x, to wybrałem przedział [-5, 5] w celu rysowania funkcji.

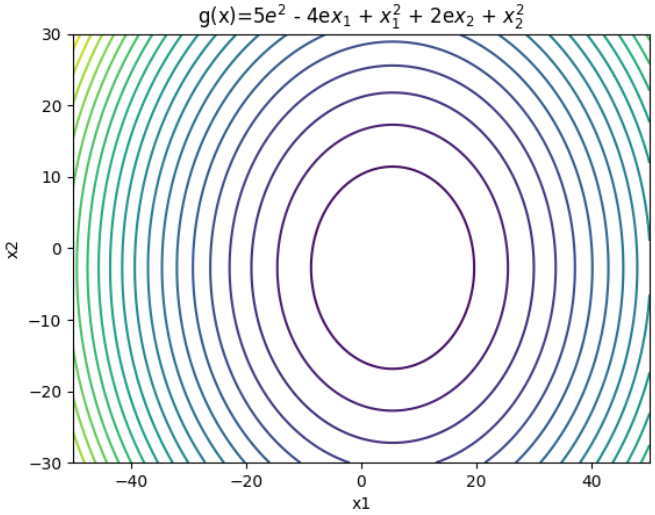


Rysunek 1. Wykres funkcji f(x)

Dlatego że funkcja g(x) ma minimum globalne w punkcie $x_1 \approx 5.44$ $x_2 \approx -2.72$ (w dalszej części sprawozdania zostanie to udowodnione), to rysowałem wykres tak, aby ten punkt został widoczny, więc wybrałem przedział dla x_1=[-40, 40], a dla x_2=[-30, 30].



Rysunek 2. Wykres powierzchni funkcji g(x)



Rysunek 3. Wykres konturowy funkcji g(x)

Zadanie 1.2. Która z funkcji będzie „łatwiejsza“ do optymalizacji?

Najpierw mi się wydało, iż funkcja f(x) będzie łatwiejsza do optymalizacji, ponieważ jest jednowymiarowa, a g(x) – dwuwymiarowa. Jednak to jest błędne założenie, ponieważ większa liczba wymiarów może spowodować:

1. Trudności obliczeniowe: liczba elementów w gradiencie, które trzeba policzyć wzrośnie, ale niekoniecznie trudność w znalezieniu optimum globalnego funkcji.
2. Trudności wizualizacji problemu.

Jednak w danym przypadku funkcja g(x) ma tylko jedno optimum, co łatwo udowodnić matematycznie: należy przyrównać każdą wartość w wektorze gradientu do zera i rozwiązać układ równań:

$$\begin{cases} 2(x_1 - 2e) = 0 \\ 2(x_2 + e) = 0 \end{cases}$$

A więc $x_1 = 2e \approx 5.44; x_2 = -e \approx -2.72$.

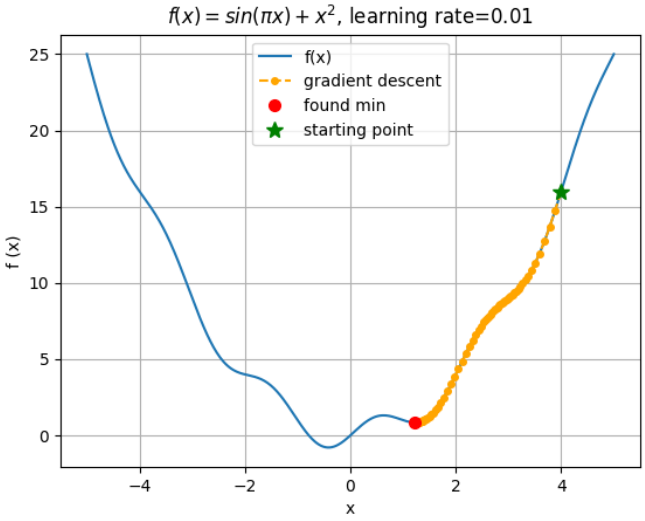
Natomiast z wykresu na rysunku 1. Widać od razu, że funkcja f(x) ma co najmniej dwa minima w punktach:

$x \approx -0.41$ oraz $x \approx 1.22$,

Zatem, dla funkcji f(x) metoda najszybszego spadku może znaleźć minimum lokalne ($x \approx 1.22$), a nie globalne ($x \approx -0.41$), jednak dla funkcji g(x) takiego problemu nie będzie, czyli funkcja g(x) jest łatwiejsza do optymalizacji.

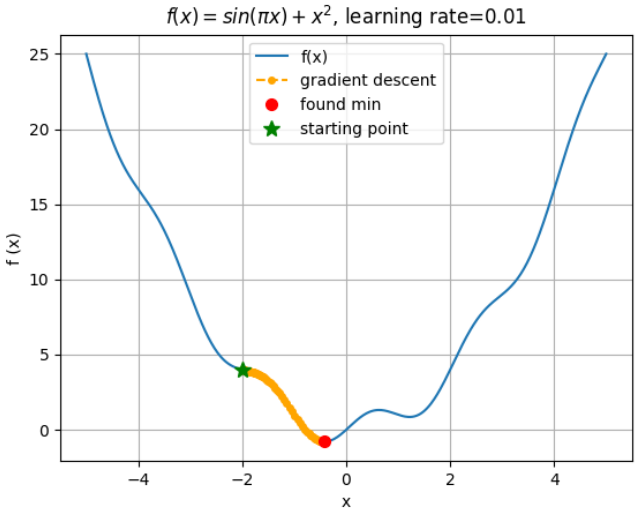
Zadanie 2. Zaimplementować algorytm najszybszego spadku oraz zastosować go do znalezienia minimum funkcji f i g.

Algorytm najszybszego spadku został zaimplementowany w pliku *gradient_descent.py*. Na ten algorytm mają wpływ kilka rzeczy: punkt początkowy, krok, liczba iteracji algorytmu. Przykładowe wyniki tego algorytmu zilustrowano na rysunkach 4, 5, 6, 7, 8.



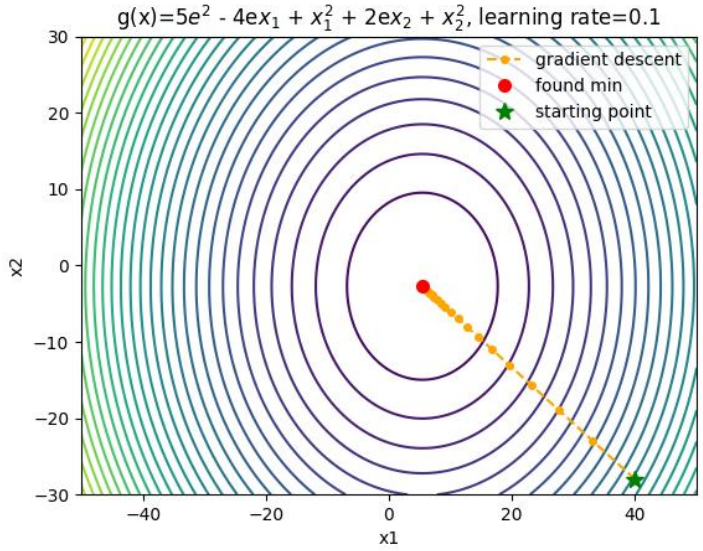
Rysunek 4. Wynik działania algorytmu najszybszego spadku

dla f(x): krok=0.01, x_{start}=4

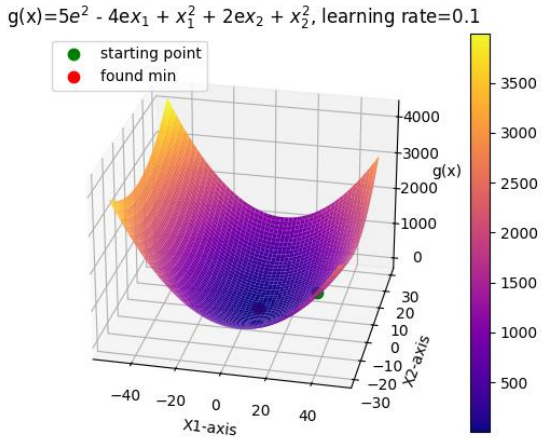


Rysunek 5. Wynik działania algorytmu najszybszego spadku

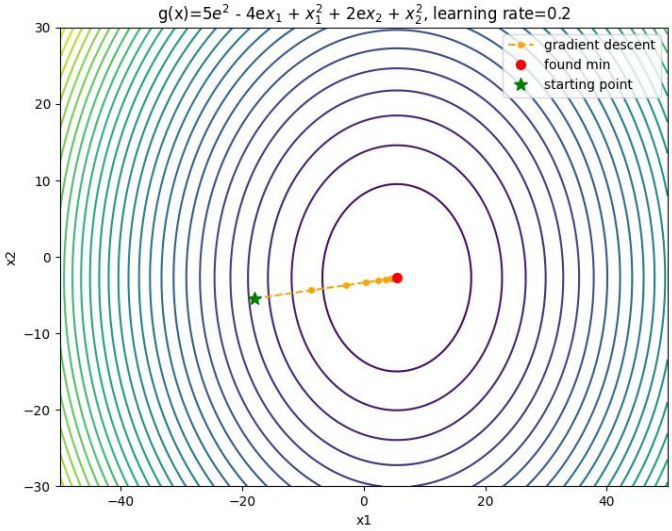
dla f(x): krok=0.01, x_{start}=-2



Rysunek 6. Wynik działania algorytmu najszybszego spadku dla g(x): krok=0.1, x_{1start}=20, x_{2start}=-20



Rysunek 7. Wynik działania algorytmu najszybszego spadku dla g(x): krok=0.1, x_{1start}=40, x_{2start}=-28



Rysunek 8. Wynik działania algorytmu najszybszego spadku

dla g(x): krok=0.2, x_{1start}=-18, x_{2start}=-2e

Na rysunkach 4 oraz 5 można zauważyć, że znaleziony minimum przy różnych punktach początkowych znajduje się w różnych miejscach. Spowodowano to tym, że algorytm najszybszego spadku, w wersji bez modyfikacji, szuka minimum lokalnego, a nie globalnego. Rysunek 6 i 7 reprezentują tą samą sytuację, jednak w dalszej części sprawozdania będę używał reprezentacji funkcji za pomocą wykresu konturowego, bo jest ona bardziej czytelna.

Zadanie 3. Zbadać wpływ rozmiaru kroku dla różnych (losowych) punktów początkowych. Jak często udaje się zlokalizować minimum globalne?

Za pomocą własnoręcznie stworzonych danych testowych w pliku *test_data.py* znaleziono minima, które zwrócił algorytm najszybszego spadku. W tabelach 1. oraz 2. można zobaczyć wartości znalezionych minimów przy użyciu algorytmu gradientowego spadku przy różnych wartościach początkowych oraz krokach dla funkcji f(x) i g(x) odpowiednio.

Wartość początkowa	krok	Znalezione minimum x	f(x)
-6	0.0000002	-5.99	35.92
-6	0.0001	-2.28	4.42
-6	0.03892	-0.41	-0.79
-6	0.0938	-0.41	-0.79
-6	0.999	-81.9	6708.33
-4	0.0023211	-0.41	-0.79
-4	0.01939	-0.41	-0.79
-4	0.02232	-0.41	-0.79
-4	0.292	0.23	0.72
-4	0.608	-2.34	4.61
-2	0.0089	-0.41	-0.79
-2	0.09	-0.41	-0.79
-2	0.423	1.28	0.87
-2	0.849	1.79	2.61
-2	0.9232	2.44	6.92
0	0.02838	-0.41	-0.79
0	0.084949	-0.41	-0.79
0	0.093929	-0.41	-0.79
0	0.129923	-0.41	-0.79
0	0.43937	0.45	1.18
0.77	0.02929	1.22	0.85
0.77	0.059302	1.22	0.85
0.77	0.6398	1.56	1.46
0.77	0.9292	-3.89	15.48
0.77	1.1	nan	nan
2	0.0000085	1.73	2.22
2	0.01	1.22	0.85
2	0.232	-0.75	-0.15
2	0.76	-3.43	12.76
2	1.12	nan	nan

Tabela 1. Wartości znalezionych minimów przy różnych krokach oraz punktów początkowych dla f(x), liczba iteracji=5000

punkt początkowy x1	punkt początkowy x2	krok	znalezione minimum x1	znalezione minimum x2	g(x1, x2)
-6	-8	0.01	5.44	-2.72	0.0
-6	-8	0.02	5.44	-2.72	-0.0
-6	-8	0.1	5.44	-2.72	-0.0
-6	-8	0.5	5.44	-2.72	-0.0
-6	-8	1	16.87	2.56	158.69
-10	8	0.00000009	-10	8	353.04
-10	8	0.09	5.44	-2.72	-0.0
-10	8	0.01	5.44	-2.72	0.0
-10	8	0.2	5.44	-2.72	-0.0
-10	8	0.5	5.44	-2.72	-0.0
-10	8	0.6	5.44	-2.72	-0.0
10	0	1E-12	10	0	28.21
10	0	0.0000005	10	0	28.16
10	0	0.02	5.44	-2.72	0.0
10	0	0.6	5.44	-2.72	-0.0
10	0	1.2	-4.3772E+146	-2.6074E+146	2.595852791426908e+293
400	-100	0.000006	395.3	-98.84	161231.61
400	-100	0.003	6.4	-2.96	0.99
400	-100	0.02	5.44	-2.72	0.0
400	-100	0.07	5.44	-2.72	-0.0
400	-100	0.999	-47.96	10.45	3024.71
-200.2	-10.32	0.0002	-132.45	-7.82	19038.65
-200.2	-10.32	0.001	-22.39	-3.75	775.56
-200.2	-10.32	0.0786	5.44	-2.72	-0.0
-200.2	-10.32	0.849	5.44	-2.72	0.0
-200.2	-10.32	1.27	4.4289E+189	1.6372E+188	inf

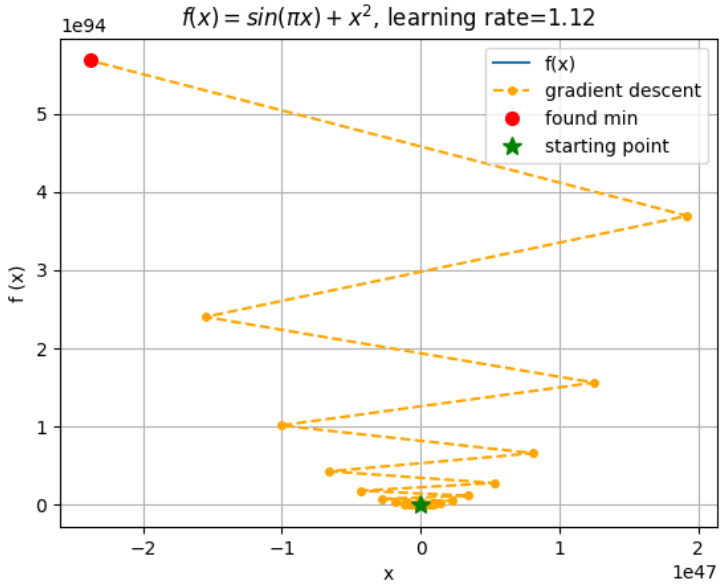
Tabela 2. Wartości znalezionych minimów przy różnych krokach oraz punktów początkowych dla g(x)

W powyższych tabelach można zauważyć, iż niektóre znalezione wartości mają wartość nan lub bardzo wielkie. Spowodowano to tym, że przy wielkich wartościach kroku, wartość punktu w aktualnej chwili „przeskakiwała” minimum funkcji i zaczynała „oscylować” co powodowało coraz większe oddalenie się od minimum. Kiedy po raz pierwszy zobaczyłem to zjawisko, to „zdebugowałem” kod i zobaczyłem, że wartości

aktualne przy każdej iteracji liczenia algorytmu najszybszego spadku rzeczywiście zaczynały „oscylować”. To zjawisko pokazano na dwóch rysunkach niżej.

```
000: 2
001: -5.998583772020569
002: 3.9196949312587455
003: -8.267622506091275
004: 7.90532234322621
005: -13.166683487417728
006: 19.37377748412804
007: -22.664505353539077
008: 29.84254786185675
009: -40.10158698231462
010: 46.385058311217556
011: -58.76060017531692
012: 75.43261833726251
013: -92.79716306437058
014: 117.89653088428958
015: -149.52602180381052
016: 185.1249434207761
017: -226.30394302320133
```

Rysunek 9. Wartości policzone iteracyjnie w pierwszych chwilach algorytmu najszybszego spadku dla f(x): krok=1.12, x_{start}=2



Rysunek 10. Wynik działania algorytmu najszybszego spadku dla f(x): krok=1.12, x_{start}=2

Jednak czasami (jak np. dla wartości pierwszej w tabeli 1.) można dotrzeć, że krok jest zbyt mały i algorytm nie zdąża dojść do minimum mając tą liczbę iteracji.

Odpowiadając na pytanie dotyczące częstotliwości znalezienia minimum globalnego: mogę powiedzieć, że przy dobraniu rozsądnych parametrów kroku, liczby iteracji zazwyczaj udaje się dojść do minimum globalnego dla funkcji g(x). Jeśli odpowiedzieć na to pytania, ale dla funkcji f(x), to sytuacja jest bardziej skomplikowana, bo istnieje więcej niż jedno minimum lokalne, a to znaczy, że nie zawsze uda się dojść do minimum globalnego (jeśli punkt początkowy jest trochę dalej od wartości x=0.5).