Ćwiczenie 1

Zagadnienie przeszukiwania i podstawowe podejścia do niego

Adam Nowakowski (300428)

1. Zadanie

Zadaniem była implementacja *metody Newtona* i przebadanie jej pod kątem *rozmiaru kroku* β oraz *wartości punktu początkowego*. Algorytm był badany na dwóch funkcjach celu:

$$f_1(x) = -x^T x$$

$$f_2(x) = -x^T x + 1.1cos(x^T x)$$

2. Wprowadzenie teoretyczne

Metoda Newtona służy znalezieniu ekstremum lokalnego funkcji celu przy pomocy gradientu oraz hesjanu tejże funkcji. Algorytm polega na obliczeniu wektora d, wskazującego kierunek, w jaki należy się przesunąć aby dojść do rozwiązania. Oblicza się go ze wzoru:

$$\mathrm{d} = H_q^{-1}(x) * \nabla q(x)$$

gdzie:

$H_q^{-1}(x)$ – macierz Hessego funkcji (hesjan)

$\nabla q(x)$ – gradient funkcji

Punkt w każdej kolejnej iteracji algorytmu obliczany jest ze wzoru:

$$x(k+1) = x(k) + \beta d$$

gdzie:

$x(k)$ – punkt w k-tej iteracji

β – rozmiar kroku

3. Implementacja

W katalogu /implementacja znajdują się dwa skrypty w kodzie Pythona:

- *functions.py* – zawiera klasy reprezentujące kolejno obie funkcje, w których za pomocą biblioteki numpy obliczane są atrybuty zależne od punktu x.
- *NewtonMethod.py* – zawiera klasę, w której zaimplementowany jest badany algorytm.

Aby rozwiązać problem metodą Newtona należy utworzyć obiekt klasy NewtonMethod, wpisując jako argument klasę badanej funkcji, oraz wywołać metodę solve. Metoda solve jest właściwą implementacją algorytmu i przyjmuje jako argumenty punkt startowy oraz rozmiar kroku β.

W etapie inicjalizacji do zmiennej x przypisany jest punkt startowy oraz do zmiennej u przypisana jest duża liczba (w tym przypadku 10). Przy pomocy u definiowany jest warunek stopu.

W pętli głównej dzieją się następujące rzeczy:

1. Do listy punktów dodawany jest x, aby później można było odtworzyć drogę optymalizacji.
2. Obliczany jest wektor d
3. Obliczana jest nowa wartość punktu x
4. Obliczane jest u, które jest odległością między starym a nowym punktem

Algorytm działa, dopóki u nie osiągnie małej wartości (w ty przypadku 0.00001).

4. Badanie metody Newtona

W katalogu /implementacja znajduje się skrypt tests.py, w którym przedstawiona jest metoda badania algorytmu na kilku przykładach.

Na początku, dla lepszego wyobrażenia sobie problemu, pod uwagę brana była funkcja jednej zmiennej. Problem wtedy sprowadzał się do następujących funkcji celu:

$$f_{11}(x) = -x^2$$

$$f_{12}(x) = -x^2 + 1.1cos(x^2)$$

Intuicyjnie można stwierdzić iż funkcja $f_{11}(x)$ ma tylko jedno ekstremum, które jest maksimum globalnym: $f_{11}(0) = 0$. Funkcja $f_{12}(x)$ ma za to wiele ekstremów lokalnych, gdyż jest tam człon z $cos(x^2)$. Maksimum globalne osiąga zaś w $f_{11}(0) = 1.1$.

Testy dla funkcji $f_{11}(x)$ wyglądały następująco:

x	β	Liczba iteracji	Wynik
-10	0.2	∞	∞
	0.01	∞	∞
	-0.1	112	0.00
	-0.01	918	0.00
	-0.7	13	0.00
	-2	∞	$\sim$
	-2.1	∞	∞
0.1	0.2	∞	∞
	0.01	∞	∞
	-0.1	67	0.00
	-0.01	460	0.00
	-0.7	7	0.00
	-0.9	5	0.00
	-1.6	20	0.00
200	0.2	∞	∞
	0.01	∞	∞
	-0.1	139	0.00
	-0.01	1216	0.00
	-0.7	15	0.00
	-1	2	0.00
	-1.1	9	0.00

Z testów powyższej funkcji można wysnuć następujące wnioski dotyczące tej implementacji metody Newtona:

- Im dalej oddalony punkt od ekstremum, tym więcej potrzeba iteracji na jego znalezienie.
- Liczba iteracji maleje, gdy β zmienia się od 0 do -1 (im mniejszy krok, tym więcej trzeba ich poczynić), potem rośnie, gdy zmienia się od -1 do -2 (wartość coraz bardziej zaczyna oscylować wokół zera)
- Niezależnie od x, dla $\beta = 0$, algorytm wskaże jedną iterację i jako wynik da punkt z którego startował. Dla $\beta = -1$ algorytm w przypadku tej funkcji od razu wskaże poprawny wynik. Dla $\beta = -2$ wynik będzie oscylował w nieskończoność między punktem startowym a punktem do niego przeciwnym.
- Algorytm w przypadku tej funkcji rozbiega się przy $\beta > 0$ i $\beta < -2$.
- Algorytm z dużą dokładnością wyznaczyło wartość, której się spodziewaliśmy

Testy dla funkcji $f_{12}(x)$ wyglądały następująco:

x	β	Liczba iteracji	Wynik
-10	0.2	1885	-10.54
	0.01	1462	-10.83
	-0.1	61	-9.97
	-0.01	384	-9.97
	-0.7	9	-9.97
	-2	∞	$\sim$
	-2.1	∞	$\sim$
0.1	0.2	743	12.70
	0.01	7568	48.19
	-0.1	67	0.00
	-0.01	461	0.00
	-0.7	9	0.00
	-0.9	6	0.00
	-1.6	20	0.00
200	0.2	48	199.88
	0.01	468	199.99
	-0.1	45	199.99
	-0.01	175	200.00
	-0.7	8	199.99
	-1	6	200.01
	-1.1	7	200.05

Patrząc na powyższe wyniki, możemy stwierdzić:

- Algorytm w przypadku tej funkcji jest bardziej dokładny bliżej $\mathrm{x} = 0$, ponieważ im dalej się od niego oddalimy tym bliżej siebie położone są ekstrema. Wynikiem tego może być „przeskoczenie” do innego ekstremum – dalszego.
- Przy $\beta > 0$ i $\beta < -2$. wyniki, tak jak w poprzednim przypadku, są niemiarodajne.
- W przypadku tej funkcji, liczba iteracji nie zależy od x, ponieważ algorytm dąży do pobliskiego ekstremum, a nie do maksimum globalnego.
- Tak jak w poprzednim przypadku liczba iteracji maleje, gdy β zmienia się od 0 do -1, potem rośnie, gdy zmienia się od -1 do -2

Algorytm działa podobnie dla x rzędu > 1, np. dla $f_{21}(\mathrm{x},\mathrm{y})$:

x,y	β	Liczba iteracji	Wynik
$[-10, 0.1]$	0.2	∞	$[\infty, \infty]$
	0.01	∞	$[\infty, \infty]$
	-0.1	111	$[0.00, 0.00]$
	-0.01	918	$[0.00, 0.00]$
	-0.7	13	$[0.00, 0.00]$
	-2	∞	$\sim$
	-2.1	∞	$[\infty, \infty]$

Oraz dla $f_{22}(\mathrm{x},\mathrm{y})$:

x,y	β	Liczba iteracji	Wynik
$[-10, 0.1]$	0.2	10765	$[28.34, -0.28]$
	0.01	4223	$[-14.14, 0.14]$
	-0.1	61	$[-9.97, 0.10]$
	-0.01	386	$[-9.97, 0.10]$
	-0.7	9	$[-9.97, 0.10]$
	-0.9	6	$[-9.97, 0.10]$
	-1.6	17	$[-9.97, 0.10]$

Można wysunąć konkluzję, że w ogólnym przypadku, od współczynnika β zależy ilość iteracji, a także dokładność wyniku końcowego. Od punktu startowego zależy jaki wynik końcowy dostaniemy (w którym ekstremum lokalnym się znajdziemy).