

# Podstawy Sztucznej Inteligencji

Projekt: Algorytm ewolucyjny  
Problem: Maksimum funkcji

Prowadzący:  
Kazimierz Krosman

Autorzy:  
Tomasz Bajura  
Przemysław Jakielaszek  
Kamil Kaczmarek

# Spis Treści:

<b>Temat projektu</b>	<b>2</b>
<b>Opis algorytmu</b>	<b>2</b>
Funkcja celu	2
Osobnik	2
Populacja	2
Przebieg algorytmu	2
Krzyżowanie	3
Mutacja	3
Selekcja	4
Inicjalizacja algorytmu	5
Warunek stopu	5
<b>Implementacja</b>	<b>5</b>
Opis problemu	5
Osobnik	5
Algorytm	6
<b>Testowanie</b>	<b>6</b>
$f(x)=x^5 - 90000x$	6
$f(x)=\log(\text{abs}(\sin(x)))$	8
$f(x)=\sin(x)$	10
$f(x,y,z)=0.01 \cdot x^2 + 0.02 \cdot y^2 + 0.03 \cdot z^2 - \cos(x) \cdot \cos(y) \cdot \cos(z)$	10
$f(x,y)=\sin(x) / \arccos(y)$	12
$f(x,y)=\text{ctg}(x/100)$	13
<b>Wnioski</b>	<b>14</b>

# 1. Temat projektu

KK.AE4. Maksimum funkcji. Dla zadanej funkcji wieloparametrowej (z funkcjami trygonometrycznymi) znaleźć minimum lub maksimum z zadaną dokładnością.

## 2. Opis algorytmu

### 2.1. Funkcja celu

W analizowanym przez nas problemie znajdowania maksimum/minimum funkcji funkcją celu jest bezpośrednio podana funkcja. Z tym rozróżnieniem, że w przypadku szukania maksimum za lepsze osobniki uznawane są te, dla których funkcja celu przyjmuje większe wartości, a w przypadku szukania minimum – mniejsze.

### 2.2. Osobnik

Na genotyp osobnika populacji składają się dwa wektory  $n$ -elementowe, gdzie  $n$  to liczba argumentów zadanej funkcji. Pierwszy z nich określa wartości tych argumentów, natomiast w drugim zapisane są odchylenia standardowe rozkładów, które są wykorzystywane przy mutacjach dla kolejnych argumentów. Oprócz tego osobnik ma w sobie zapisaną funkcję celu, która odróżnia go od osobników o innej funkcji celu i nie pozwala się im krzyżować.

### 2.3. Populacja

Wykorzystana została metoda  $(\mu + \lambda)$ . W każdej iteracji populacja rodzicielska składa się z  $\mu$  osobników, a generowana populacja potomna z  $\lambda$  osobników. Do następnej populacji przeżywa  $\mu$  spośród połączonej populacji rodzicielskiej i potomnej.

### 2.4. Przebieg algorytmu

W każdej iteracji następują po sobie kolejno etapy: krzyżowanie populacji rodzicielskiej, mutacja powstałej populacji potomnej oraz selekcja osobników do populacji rodzicielskiej w następnej iteracji. Każdy osobnik ma 10 000 szans

na stworzenie się. Jest to spowodowane tym, że każda funkcja ma inną dziedzinę. Losowanie wartości z zadanego przedziału może powodować, że nowy osobnik zwraca jakiś błąd matematyczny. Gdy 10 000 prób stworzenia osobnika kończy się fiaskiem kończymy algorytm z informacją, że nie istnieje szansa stworzenia osobnika - jest to obrona przed nieskończoną pętlą, w której to użytkownik podaje funkcję, która za każdym razem zwraca błąd dzielenia przez zero. Teraz dokładnie zostaną omówione wszystkie te etapy.

#### 2.4.1. Krzyżowanie

Osobnik populacji potomnej powstaje poprzez krzyżowanie dwóch osobników z populacji rodzicielskiej. Są oni losowani ze zbioru  $\mu$  osobników ze zwracaniem. Początkowo genotyp potomka był wyznaczany poprzez średnią arytmetyczną genotypów rodziców, ale szybko okazało się, że istnieje prosta metoda poprawiająca działanie algorytmu. Zamiast średniej wykorzystana została metoda interpolacji genotypów rodziców ze współczynnikiem  $\alpha$  losowanym z rozkładu jednostajnego ciągłego  $U(0,1)$ . Należy zauważyć, że potomek znajduje się gdzieś na prostej łączącej rodziców.

#### 2.4.2. Mutacja

Mutacja służy do wprowadzania zróżnicowania genetycznego w populacji. Poddawane są jej wszystkie osobniki potomne powstałe w wyniku krzyżowania. W pierwszym kroku aktualizowany jest wektor odchyleń standardowych. Nowa wartość wyraża się wzorem:

$$\begin{aligned}\sigma'_i &= \sigma_i \cdot \exp(\tau' \cdot \varepsilon + \tau \cdot \varepsilon_i), \quad 0 < i < n, \text{ gdzie} \\ \varepsilon &\sim N(0, 1) \text{ i jest takie samo dla każdego wymiaru,} \\ \varepsilon' &\sim N(0, 1) \text{ i jest losowane dla każdego wymiaru } i, \\ \tau \text{ i } \tau' &\text{ to współczynniki o wartościach równych} \\ \tau' &= \frac{1}{\sqrt{2n}}, \quad \tau = \frac{1}{\sqrt{2 \cdot \sqrt{n}}}\end{aligned}$$

Następnie wyznaczany jest nowy wektor wartości argumentów, który jest obliczany danym wzorem:

$$x'_i = x_i + \sigma'_i \cdot v_i, \quad 0 < i < n, \text{ gdzie}$$

$$v_i \sim N(0, 1) \text{ i jest losowane dla każdego wymiaru } i;$$

Wynika z tego, że dzięki mutacji w każdym wymiarze (dla wszystkich argumentów), wykonywany jest krok losowany z rozkładu  $N(0, \sigma'_i)$ . Duże wartości odchylen sprzyjają eksploracji tzn. nasz nowy punkt będzie znajdował się daleko od pierwotnego. Daje to szansę na znalezienie nowych maksimów lokalnych, które mogą okazać się globalnym. Natomiast małe wartości odchylen standardowych sprzyjają eksploatacji tzn. punkt poszukuje najbliższego maksimum lokalnego. Oczekiwane działanie algorytmu sprowadza się do tego, że najpierw odchylenia są duże, żeby przeszukać jak największą przestrzeń. Następnie odchylenia powinny zacząć maleć, aby zbliżyć się do maksimów lokalnych. Oczywiście cały czas powinna być urzymywana przy życiu grupa osobników z dużymi odchyleniami, która prowadzi eksplorację.

#### 2.4.3. Selekcja

Spośród  $(\mu + \lambda)$  osobników z połączonych populacji rodzicielskiej i potomnej należy wybrać te, które zostaną wykorzystane jako populacja rodzicielska w następnej iteracji. Wybór opiera się na wartości funkcji celu osobników. Początkowo użyta została metoda  $\mu$  najlepszych, ale okazała się ona nieskuteczna. Odrzucała ona obiecujące osobniki, które w danym momencie miały niską wartość funkcji celu, ale dawały szansę na znalezienie maksimum globalnego. W obecnej postaci algorytm korzysta z metody turniejowej. Do turnieju losowane jest  $k$  osobników bez zwracania. Najlepszy z nich przechodzi do populacji rodzicielskiej w następnej iteracji i jest usuwany z obecnej populacji. Kolejne turnieje są organizowane, aż uzyskamy pełną populację  $\mu$  osobników w do następnej iteracji. Metoda ta zdecydowanie faworyzuje osobniki o wysokiej wartości funkcji celu, gdyż jeśli zostaną wylosowane to wygrają turniej i przeżyją. Jednak daje ona też szansę słabszym osobnikom na przetrwanie w sytuacji, gdy do turnieju nie dostanie się wartościowy osobnik. Takie słabsze osobniki mogą okazać się w przyszłości niezwykle istotne, gdyż pozwolą na odkrycie nowego maksimum lokalnego. Dodatkowym ulepszeniem tej metody jest zastosowanie strategii elitarnej, która gwarantuje przeżycie najlepszemu osobnikowi automatycznie bez obowiązku brania udziału w turnieju. Zapewnia to gwarancję, że nie stracimy najlepszego dotychczas znalezionej rozwiązania.

#### 2.4.4. Inicjalizacja algorytmu

Zostaje wylosowana początkowa populacja rodzicielska o wartościach obu wektorów z zadanych zakresów oraz określona funkcja celu - w naszym wypadku zadana funkcja matematyczna.

#### 2.4.5. Warunek stopu

Algorytm działa zadaną liczbę iteracji. Zwraca najlepszy uzyskany dotychczas wynik.

### 3. Implementacja

Program został napisany w języku Python w wersji 3.5.2. Zakłada posiadanie takich bibliotek jak: copy, math, matplotlib, operator, parser, random, sys. Większość tych bibliotek jest dołączonych w standardzie Pythona 3.5.2. Jednak gdyby program zwracał błąd o nieistnieniu jakiejś biblioteki wystarczy ściągnąć ją za pomocą funkcji terminala. Najprawdopodobniej w standardzie nie będzie dołączona jedynie biblioteka matplotlib, która pomogła nam przy rysowaniu wykresów. Żeby ją pobrać należy w terminalu wpisać `pip install matplotlib`. Program składa się z 3 modułów – main, Individual oraz Formula, które zostaną opisane w dalszej części sprawozdania.

#### 3.1. Opis problemu

Klasa Formula odpowiada za reprezentację badanego problemu funkcją celu. W naszym projekcie jej zadaniem jest wyznaczenie argumentów oraz zapisanie samej funkcji matematycznej. W tym celu wykorzystana została biblioteka parser. Do przechowywania nazw argumentów oraz ich wartości służy struktura danych - słownik. W celu wykorzystania algorytmu do innego problemu należy przeciążyć zawartość klasy Formula.

#### 3.2. Osobnik

Klasa Individual odpowiada za reprezentację osobnika populacji. Opisuje ona strukturę genotypu oraz operacje wykorzystywane w czasie trwania algorytmu (krzyżowanie, mutacja).

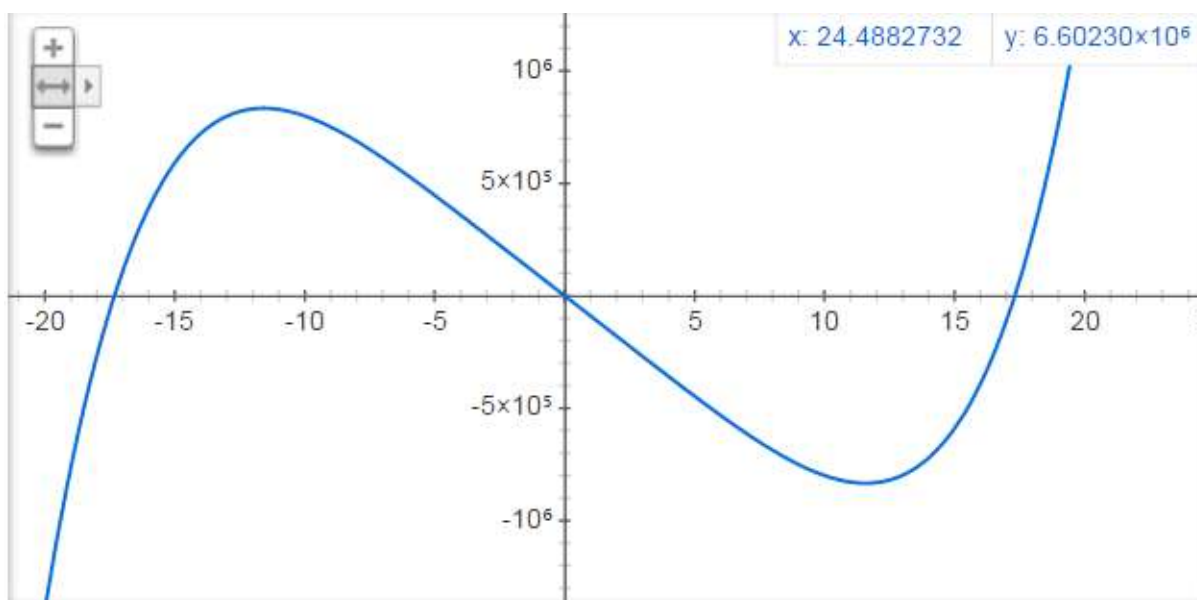
### 3.3. Algorytm

Inicjalizacja oraz główna pętla algorytmu zawarte są w pliku main.py.

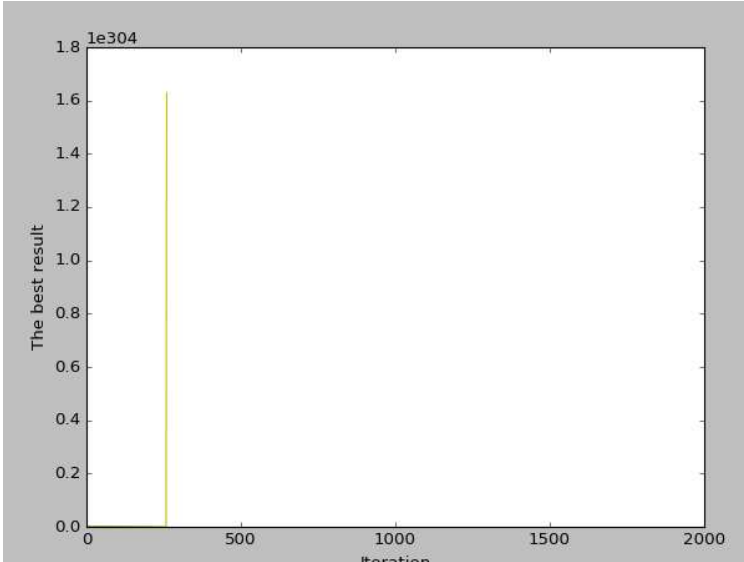
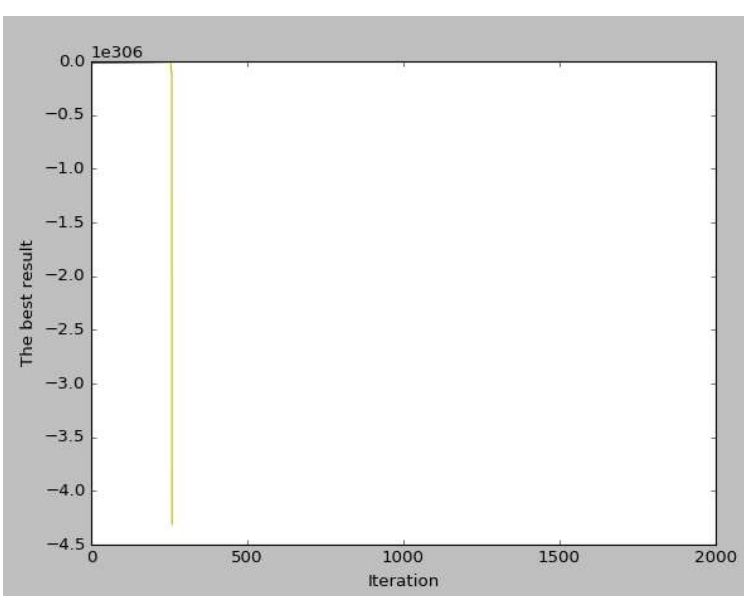
## 4. Testowanie

Podczas testowania badany był czas zbiegania algorytmu do minimum/maksimum. Badane były również funkcje, z którymi algorytm może mieć problem bądź w ogóle sobie nie radzić.

4.1.  $f(x) = x^5 - 90000x$



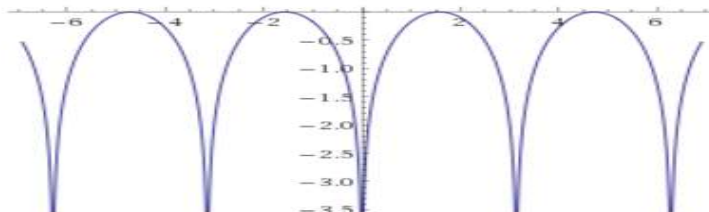
max/min	maxiterations	iterations	final value
max	2000	262	inf
min	2000	261	-4.3690968475437303e+307
max	2000	247	2.6304322159379683e+307
min	2000	252	-inf
średnia	2000	255,5	

	<p><b>Maximize</b></p> <p>Output:</p> <p><i>number: 262</i></p> <p><i>arguments: {'x': 4.838197815248702e+61}</i></p> <p><i>distributions: {'x': 4.2164627713633684e+61}</i></p> <p><i>value: inf</i></p>
	<p><b>Minimize:</b></p> <p>Output:</p> <p><i>number: 261</i></p> <p><i>arguments: {'x': -3.373481511426606e+61}</i></p> <p><i>distributions: {'x': 1.3653403708042627e+61}</i></p> <p><i>value: -4.3690968475437303e+307</i></p>

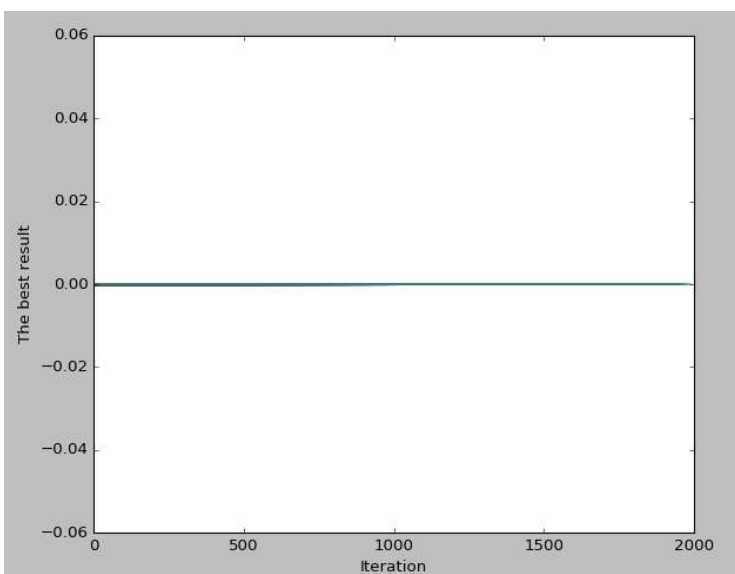
Jak widać, algorytm dość dobrze poradził sobie z zadaną funkcją, znajdując rozwiązanie w średnio 255,5 iteracji. Podczas przebiegu można zobaczyć, że na początku najlepszy osobnik znajduje się po stronie ujemnej osi OX w maksimum lokalnym. Pomimo tego, że po stronie dodatniej osi OX znajduje się duża studnia pewne osobniki przeskakują na drugą stronę i szybko znajdują maksimum globalne.



4.2.  $f(x) = \log(\text{abs}(\sin(x)))$



max/min	maxiterations	iterations	final value
max	2000	2000	-2.656763697931529e-12
max	2000	2000	-6.390998841275136e-12
min	2000	2000	-12.362175748463981
min	10000	10000	-12.499548847867146



#### Maximize:

Output:

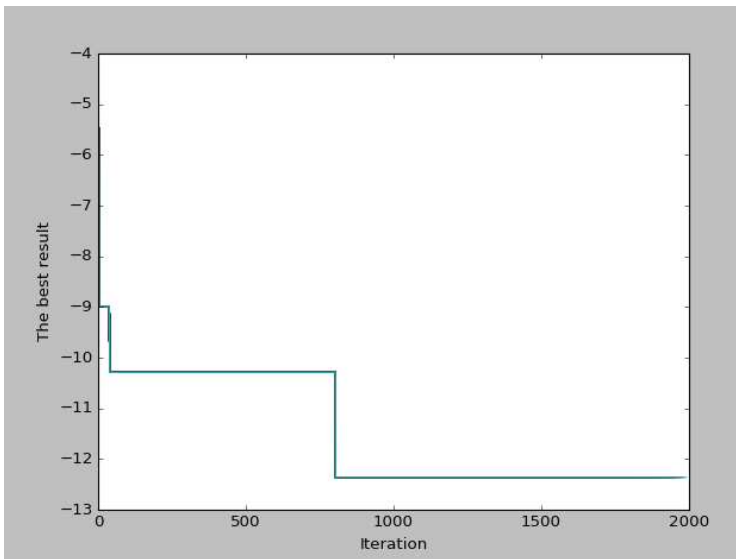
*number: 2000*

*arguments: {'x': -8823.16296530181}*

*distributions: {'x': 6886.7104861998905}*

*value: -2.656763697931529e-12*

Zauważmy prawie natychmiastowe znalezienie maksimum w początkowych iteracjach (brak zmian w kolejnych).



**Minimize:**

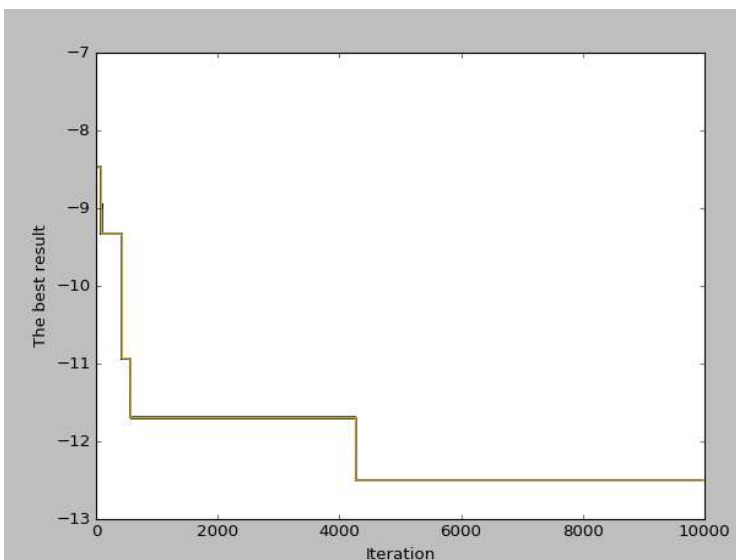
Output:

*number: 2000*

*arguments: {'x': 3238846.937362695}*

*distributions: {'x': 584571.0900219205}*

*value: -12.362175748463981*



**Minimize:**

Output:

*number: 10000*

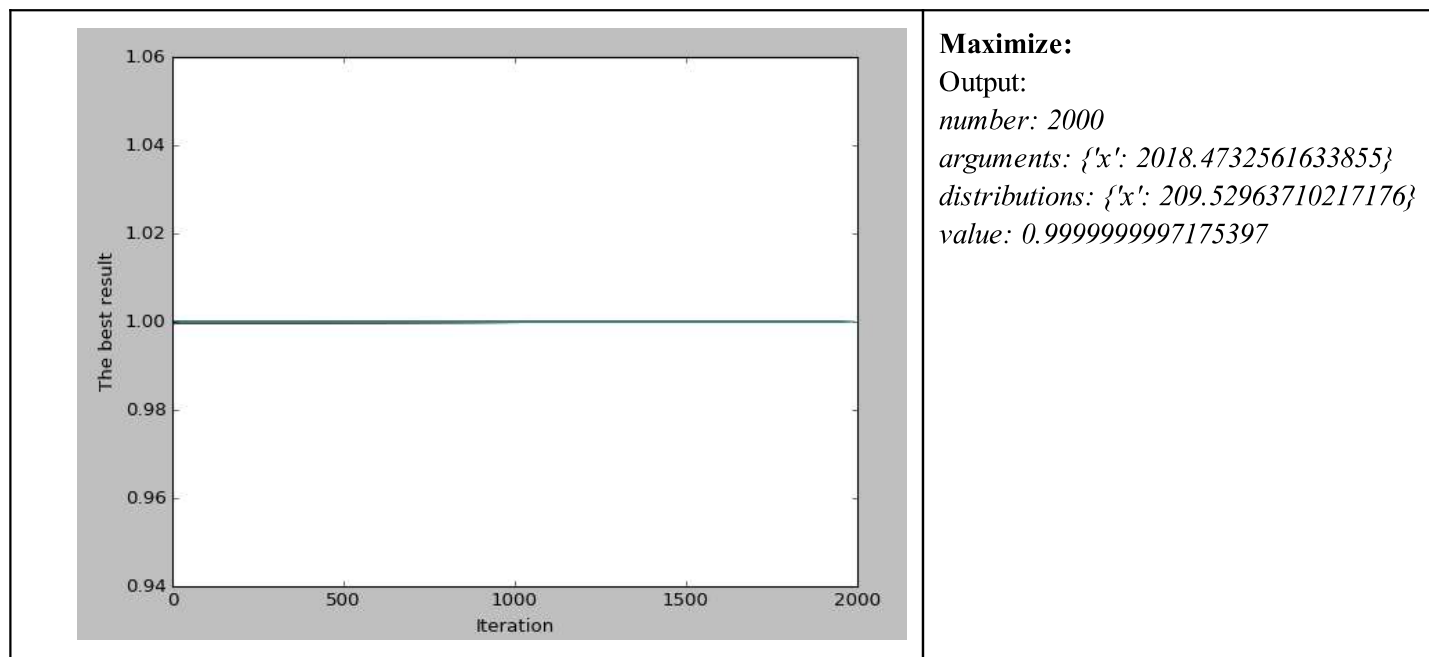
*arguments: {'x': -186944243.90463388}*

*distributions: {'x': 100341006.29468535}*

*value: -12.499548847867146*

Jak widzimy, algorytm ma problem z ustaleniem minimum powyższej funkcji – nie wykrywane jest minimum leżące w minus nieskończoności. Spowodowane jest to tym, że funkcja niemalże w miejscu zbiega do minimum i z powrotem rośnie. Można zauważyć, że osobniki, które uzyskały najlepsze wartości dla minimum mają dużą wartość dystrybucji - jest to długość skoku, który wykonuje osobnik. Przy próbach małych skoków osobniki przeskakiwały minimum.

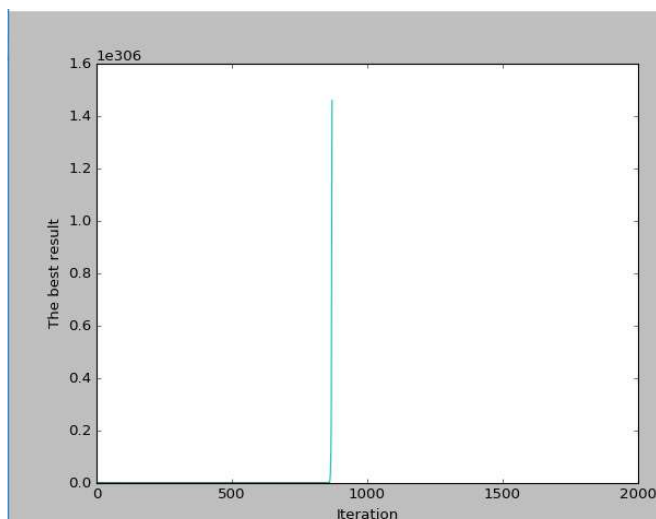
### 4.3. $f(x) = \sin(x)$



Prosta funkcja, dla której nasz algorytm niemal od razu znajduje z dość dobrym przybliżeniem maksimum. Szukanie minimum przebiega w sposób symetryczny, stąd nie zostało tu ukazane.

### 4.4. $f(x, y, z) = 0.01 \cdot x^2 + 0.02 \cdot y^2 + 0.03 \cdot z^2 - \cos(x) \cdot \cos(y) \cdot \cos(z)$

max/min	maxiterations	iterations	final value
max	2000	871	1.6962906994707576e+307
max	2000	840	1.1324918531617298e+307
średnia max	2000	855,5	~inf
min	2000	152	-1.0



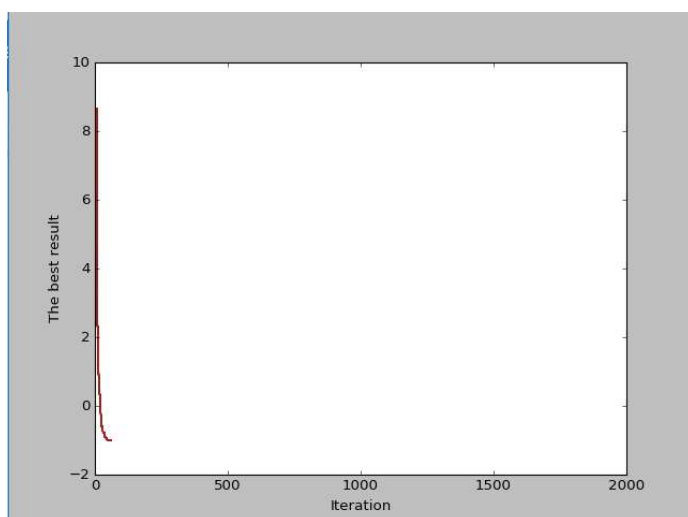
### Maximize:

number: 871

arguments: {'z': -2.3778776948298228e+154, 'x': -4.698839477942388e+86, 'y': 3.324537288354564e+85}

distributions: {'z': 9.332473514071735e+153, 'x': 3.734741484911327e+86, 'y': 1.7765323837169553e+85}

value: 1.6962906994707576e+307



### Minimize:

Output:

number: 152

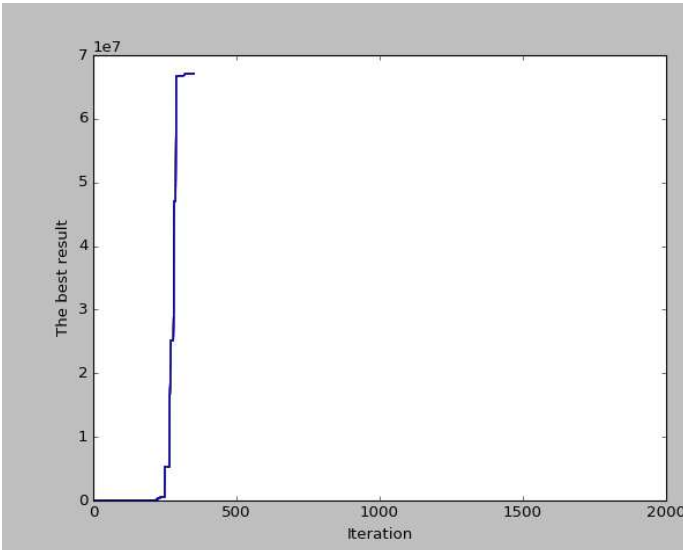
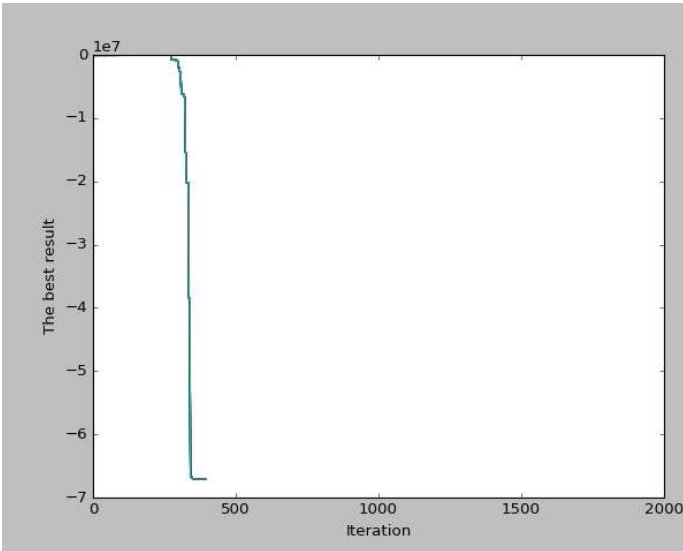
arguments: {'z': -8.42549652285797e-09, 'x': 6.5747166925515215e-09, 'y': 8.702753677357047e-09}

distributions: {'z': 4.2844367421922e-08, 'x': 4.4145774599622946e-08, 'y': 1.1197385866506161e-07}

value: -1.0

Algorytm dobrze radzi sobie z powyższą funkcją. Zarówno minimum, znajdujące się w -1, jak i maksimum jest wyszukiwane dość sprawnie.

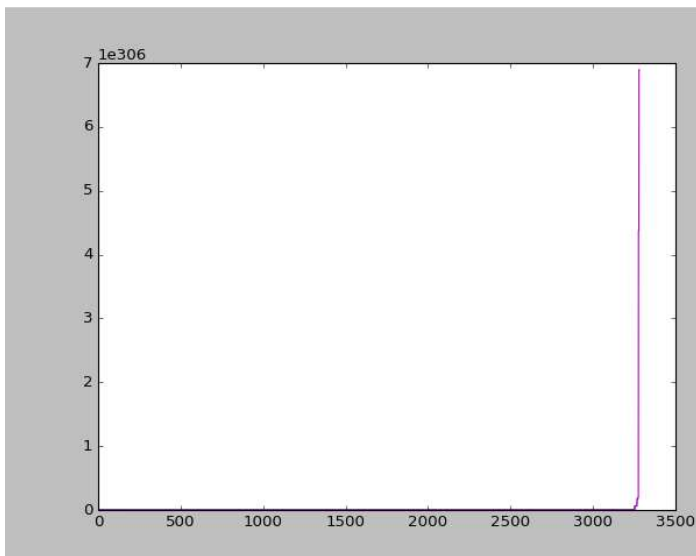
#### 4.5. $f(x,y) = \sin(x) / \cos(y)$

	<p><b>Maximize:</b>  number: 396  arguments: {'x': 45.553093482592814, 'y': 0.9999999999999999}  distributions: {'x': 1.4418528526928786e-06, 'y': 8.80754233276591e-19}  value: 67108864.0</p>
	<p><b>Minimize:</b>  number: 437  arguments: {'x': 23.56194489820077, 'y': 0.9999999999999999}  distributions: {'x': 5.40295590293583e-06, 'y': 4.7444013389483686e-18}  value: -67108864.0</p>

Algorytm dobrze radzi sobie z powyższą funkcją. Zarówno minimum, jak i maksimum jest wyszukiwane dość sprawnie. Uważny czytelnik zauważy, że wynik funkcji powinien zbiegać do +/-inf, niestety dokładność pythona pozwala tylko na dojście do wyniku równego +/- 67108864.0.

4.6.  $f(x, y) = \text{ctg}(x/100)$

max/min	maxiterations	iterations	final value
max	5000	3269	1.3435811164772888e+307
max	5000	3285	4.0889837030934424e+307
średnia max	5000	3277	~inf
min	5000	3305	-1.3420664111696783e+307
min	5000	3295	-1.0237353001237658e+307
średnia min	5000	3300	~ -inf



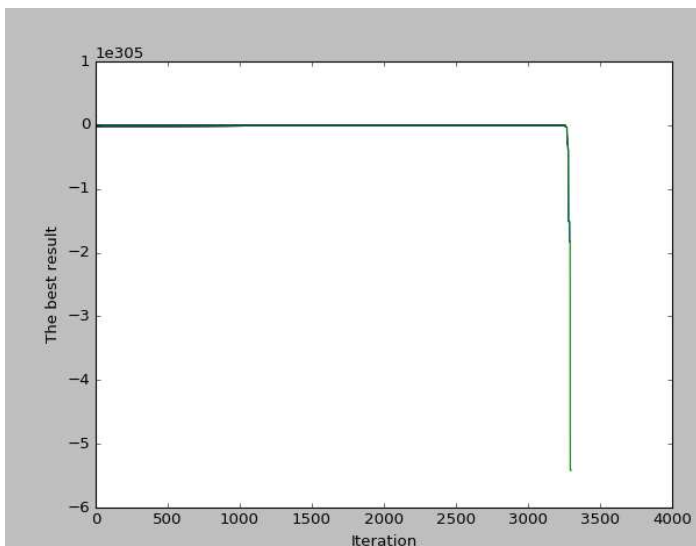
**Maximize:**

number: 3285

arguments: {'x': 2.445595464817992e-306}

distributions: {'x': 4.630724509647625e-304}

value: 4.0889837030934424e+307



**Minimize:**

Output:

number: 3305

arguments: {'x': -7.451196093406806e-306}

distributions: {'x': 8.356243300744425e-304}

value: -1.3420664111696783e+307

Pomimo tego, że funkcja jest nieciągła i szybko przeskakuje z wartości -inf do inf, to algorytm dobrze sobie z nią radzi. Zarówno minimum, jak i maksimum

jest wyszukiwane poprawnie. Osobniki muszą wykonywać drobne skoki, żeby znaleźć oczekiwaną wartość, dlatego algorytm potrzebuje dużej ilości iteracji.

## 5. Wnioski

Nasz algorytm dobrze i szybko radzi sobie z funkcjami ciągłymi oraz okresowymi, co widać na podstawie przykładów m.in. 4.1 czy 4.4. Ewentualne “fałszywe” maksima lub minima lokalne są sprawnie zastępowane przez kolejne iteracje. Większa liczba zmiennych spowalnia szybkość zbiegania algorytmu do rozwiązania, co wydaje się intuicyjne. Nieciągłość badanej funkcji, a szczególnie szybki przeskok z bardzo dużych liczb do bardzo małych liczb potrafi utrudniać lub uniemożliwiać znalezienie poprawnego rozwiązania. Przykładowo algorytm słabo sobie radzi ze znalezieniem maksimum funkcji  $\text{ctg}(x)$ , natomiast jak pokazano w przykładzie 4.6 dla funkcji  $\text{ctg}(x/100)$  poradził sobie znakomicie. Ponadto dla funkcji przedstawianych przez dyskretny zbiór oddalonych od siebie punktów, problematyczne staje się już na samym początku znalezienie osobnika, który w ogóle należy do dziedziny rozpatrywanego problemu.