Projekt:

Teoria i metody optymalizacji

Temat:

Ulepszony algorytm poszukiwania harmonii

Albert Lis 235534

Adrian Czupak 237118

1 WSTĘP

Algorytm wyszukiwania harmonii (HS – Harmony Search) został wprowadzony w 2001 roku. Jest on inspirowany sposobem improwizacji harmonii przez muzyków jazzowych. Algorytm HS narzuca mniej wymagań matematycznych w porównaniu do innych algorytmów optymalizacji, gdyż wykorzystuje on wyszukiwanie losowe. Należy on do grupy algorytmów metaheurystycznych i jego rozwiązania są przybliżone. HS generuje nowy wektor rozwiązania, po rozważeniu wszystkich istniejących wektorów, gdzie algorytm generyczny rozważa tylko dwa wektory. Algorytm ma problem z wyszukiwaniem lokalnym. W celu rozwiązania tego problemu wykorzystuje się ulepszone algorytmy wyszukiwania harmonii (IHS – Improved Harmony Search).

2 ZADANIE OPTYMALIZACJI

Znaleźć min funkcji $f(x)$ nieliniowej, ciągłej bez ograniczeń za pomocą Ulepszonego Algorytmu Poszukiwania Harmonii. (Improved Harmony Search Algorithm).

Funkcja celu:

$$\begin{gathered} min\, f(x) \\ x \in R^n \end{gathered}$$

Wymiar zadania:

$$n \leq 5$$

Kryterium stopu:

L – liczba iteracji

Rozpoznawane funkcje:

funkcje wielomianowe, exp., log., potęgowe i trygonometryczne

3 ULEPSZONY ALGORYTM WYSZUKIWANIA HARMONII

Algorytm IHS pozwala na wyszukiwanie harmonii w o wiele wydajniejszy sposób w porównaniu ze zwykłym HS. Dzieje się to dzięki zastosowaniu parametrów algorytmu ewoluujących wraz z kolejnymi generacjami wektorów. Algorytmy HS składają się z poniższych kroków:

- Krok 1. Inicjalizacja parametrów problemu i algorytmu.
- Krok 2. Inicjalizacja pamięci harmonii.
- Krok 3. Improwizacja nowej harmonii.
- Krok 4. Aktualizacja pamięci harmonii.
- Krok 5. Sprawdzenie kryterium stopu.

Kroki zostały opisane poniżej.

3.1 KROK 1. INICJALIZACJA PARAMETRÓW PROBLEMU I ALGORYTMU

W Kroku 1 należy wprowadzić następujące parametry:

Parametry problemu (zmienne X, X_{min}, X_{max} są wektorami):

- funkcja problemu – $f(X)$,
- zakresy dla każdego elementu wektora wejściowego funkcji – X_{min} i X_{max},

Parametry algorytmu:

- L – liczba iteracji,
- HMS – rozmiar pamięci harmonii (Harmony Memory Size),
- $HMCR_{min}$ i $HMCR_{max}$ – zakres dla wskaźnika uwzględniania pamięci (Harmony Memory Consideration Rate)
- PAR_{min} i PAR_{max} – zakres dla wskaźnika regulacji „wysokości tonu” (Pitch Adjustment Rate)
- bw_{min} i bw_{max} – zakres dla szerokości pasma wyszukiwania harmonii (bandwidth)

Dane te są przekazywane do programu.

3.2 KROK 2. INICJALIZACJA PAMIĘCI HARMONII

Na podstawie danych wprowadzonych w Kroku 1 przez program tablica HM o rozmiarach $N\ x\ HMS$, gdzie N jest ilością elementów wektora X przekazanego w funkcji $f(X)$, wypełniana jest losowymi zmiennymi w zakresach odpowiadających danym elementom wektora.

$$HM = \begin{bmatrix} x_1^1 & \cdots & x_N^1 \\ \vdots & \ddots & \vdots \\ x_1^{HMS} & \cdots & x_N^{HMS} \end{bmatrix} \tag{3.1}$$

3.3 KROK 3. IMPROWIZACJA NOWEJ HARMONII

W tym kroku generowany jest nowy wektor $x' = (x'_1, \ldots x'_N)$, bazując na trzech zasadach:

- Rozważanie pamięci,
- Dostosowywanie ”wysokości tonu”,
- Losowy wybór

Dla każdej generacji wyliczane są wskaźniki $HMCR(gn)$, $PAR(gn)$ i $bw(gn)$ według zaproponowanych funkcji:

$$HMCR(gn) = HMCR_{max} - \frac{HMCR_{max} - HMCR_{min}}{L} \cdot gn \tag{3.2}$$

$$PAR(gn) = PAR_{min} + \frac{PAR_{max} - PAR_{min}}{L} \cdot gn \tag{3.3}$$

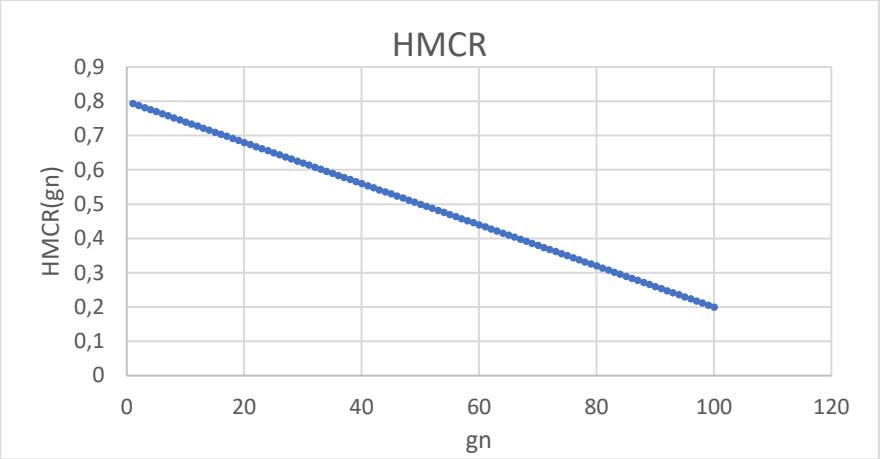
$$bw(gn) = bw_{max} \cdot exp\left(\frac{ln\left(\frac{bw_{min}}{bw_{max}}\right)}{L} \cdot gn \right) \tag{3.4}$$

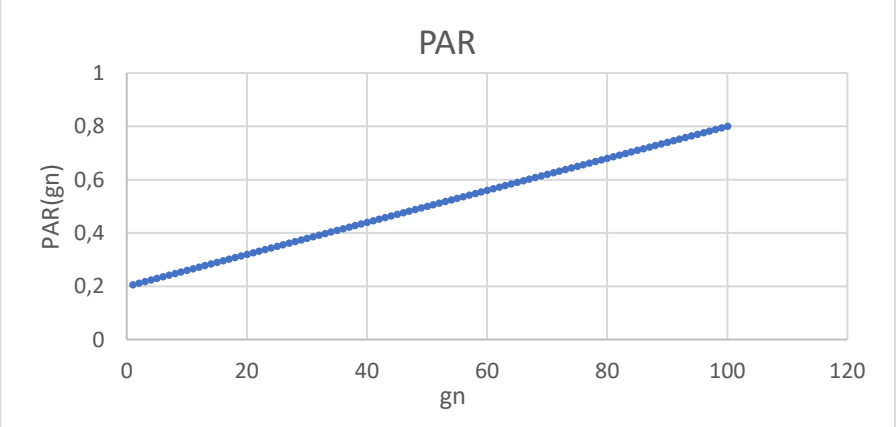
gdzie

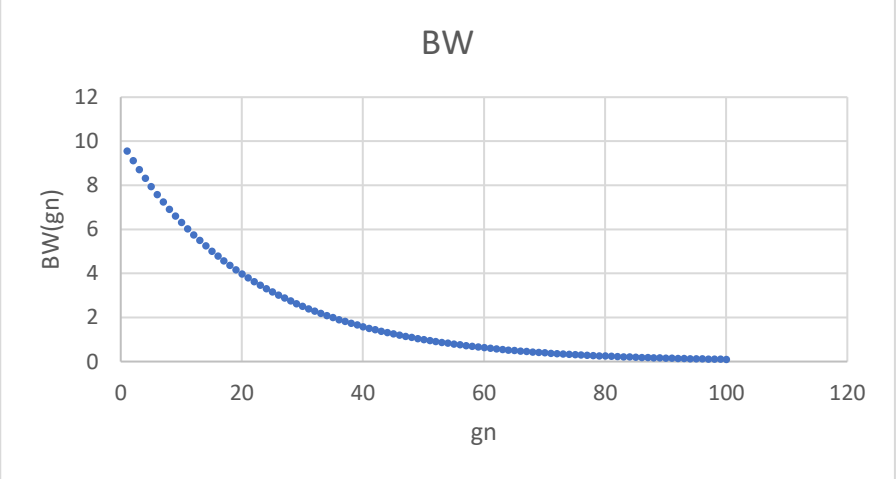
gn – generacja.

Poniżej przedstawiono wykresy zależności tych wskaźników od generacji algorytmu, dla parametrów:

	$HMCR$	PAR	bw	L
min	0.2	0.2	0.1	100
max	0.8	0.8	10	

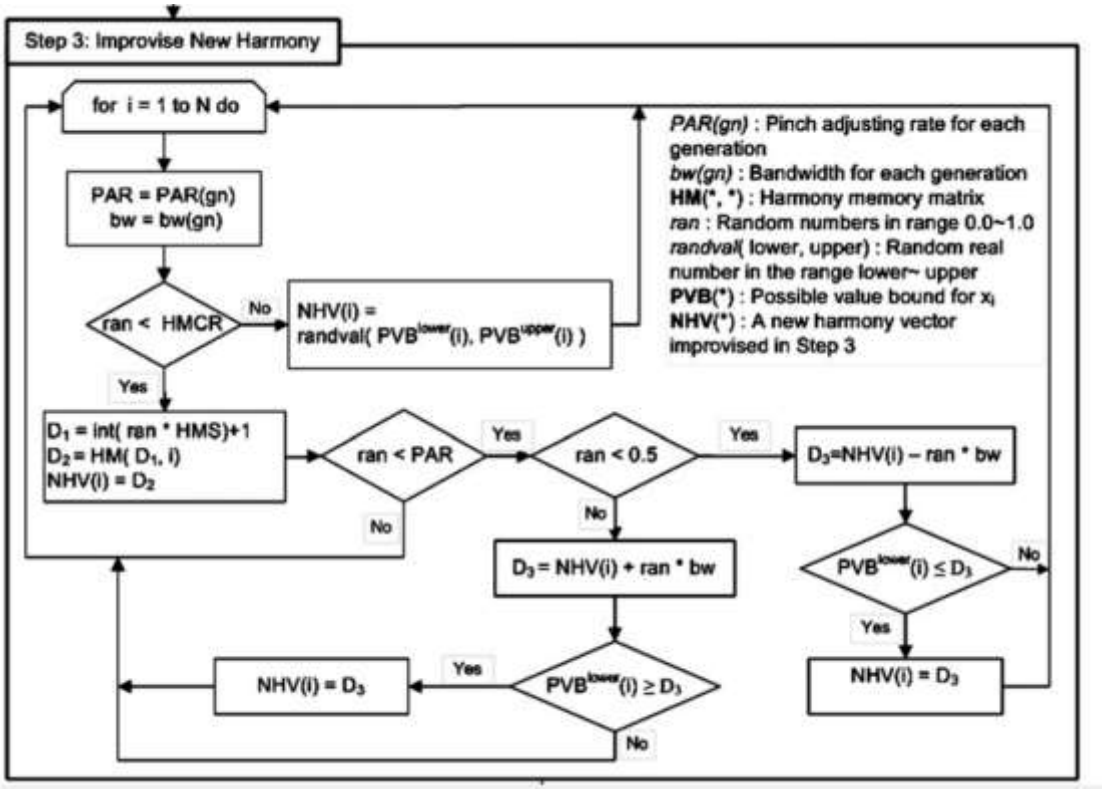






W przypadku, kiedy zakres któregokolwiek parametru algorytmu jest większy od zera algorytm jest ulepszonym algorytmem wyszukiwania harmonii, a jeżeli zakresy są zerowej szerokości algorytm zachowa się jak zwykły algorytm wyszukiwania harmonii.

Nowy wektor jest generowany według algorytmu przedstawionego na rysunku 3.1 z tą różnicą, że na początku każdej pętli poza uaktualnianiem parametrów PAR oraz bw uaktualniany jest również parametr $HMCR$. Streszczając – dla każdego elementu nowego wektora rozważa się przypisanie mu losowej wartości z dziedziny z prawdopodobieństwem $1 - HMCR$. W przeciwnym razie przypisywana jest mu wartość losowego elementu z pamięci. W przypadku wykorzystania elementu z pamięci rozważa się dostosowanie wartości tego elementu z prawdopodobieństwem PAR. Dostosowanie wartości polega na zmianie wartości po przepisaniu o wartość w zakresie pasma bw, pod warunkiem, że nowa wartość znajduje się w dziedzinie.



Rysunek 3.1) Algorytm improwizacji nowej harmonii bez ewoluującego parametru HMCR. Źródło: [1]

3.4 KROK 4. AKTUALIZACJA PAMIĘCI HARMONII

W przypadku, kiedy $f(X')$, gdzie X' jest nowo wygenerowanym wektorem, jest rozwiązaniem lepszym od najgorszego rozwiązania w pamięci, najgorszy wektor zostaje zastąpiony nowym.

3.5 KROK 5. SPRAWDZENIE KRYTERIUM STOPU

Jeśli osiągnięto maksymalną liczbę iteracji algorytm jest zakańczany, a jeśli nie algorytm powraca do kroku 3 zwiększając numer generacji o 1.

4 ŚRODOWISKO PROGRAMISTYCZNE

Język programowania: *Python*

Środowisko interfejsu graficznego aplikacji: *PyQt5*

Biblioteka prezentacji graficznej warstwic funkcji: *Matplotlib*

Pozostałe:

- *NumPy* – biblioteka dla obliczeń naukowych
- *github.com* – system kontroli wersji
- *PyCharm, Spyder* – środowiska programistyczne

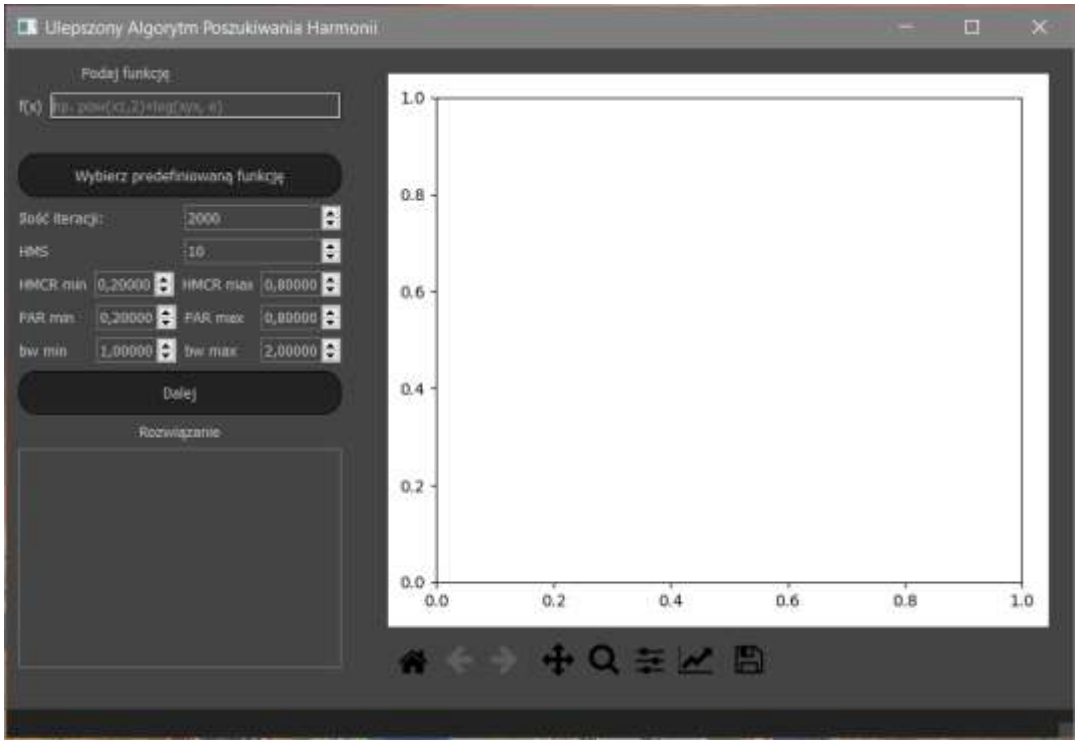
5 ZASADY WPROWADZANIA DANYCH POCZĄTKOWYCH

- Ilość iteracji: $L \in \mathbb{N}$,
- $HMS \in \mathbb{N}$,
- $HMCR_{min}, HMCR_{max} \in \langle 0,1 \rangle, HMCR_{min} < HMCR_{max}$,
- $PAR_{min}, PAR_{max} \in \langle 0,1 \rangle, PAR_{min} < PAR_{max}$,

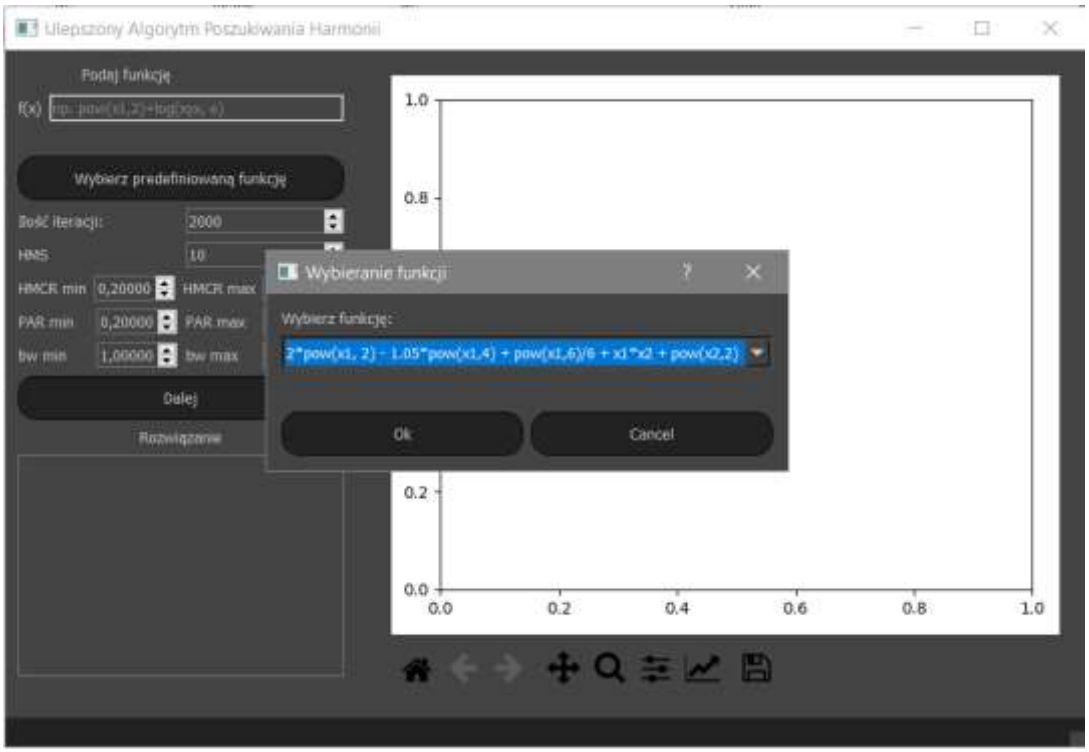
- $bw_{min}, bw_{max} \in \mathbb{R}, bw_{min} < bw_{max}$
- Zakresy zmiennych: $x_{n\,max}, x_{n\,min} \in \mathbb{R}, x_{n\,min} < x_{n\,max}$

6 PRZEZENTACJA PROGRAMU

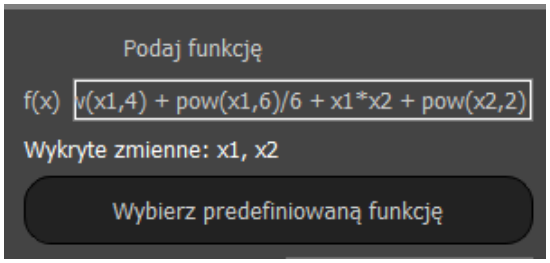
Okno po otworzeniu programu wygląda następująco:



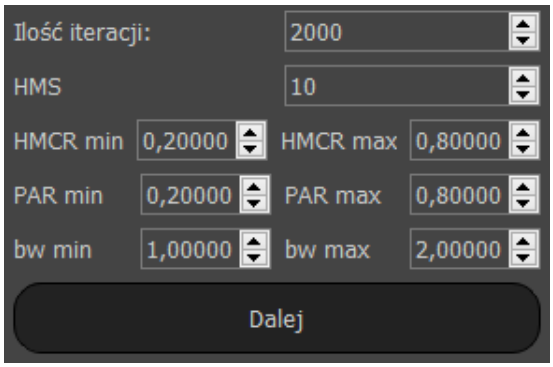
- Pole $f(x)$ służy do wpisywania zadanej funkcji.
- Przycisk „Wybierz predefiniowaną funkcję” otwiera okno z możliwym wyborem wstępnie przygotowanych funkcji:



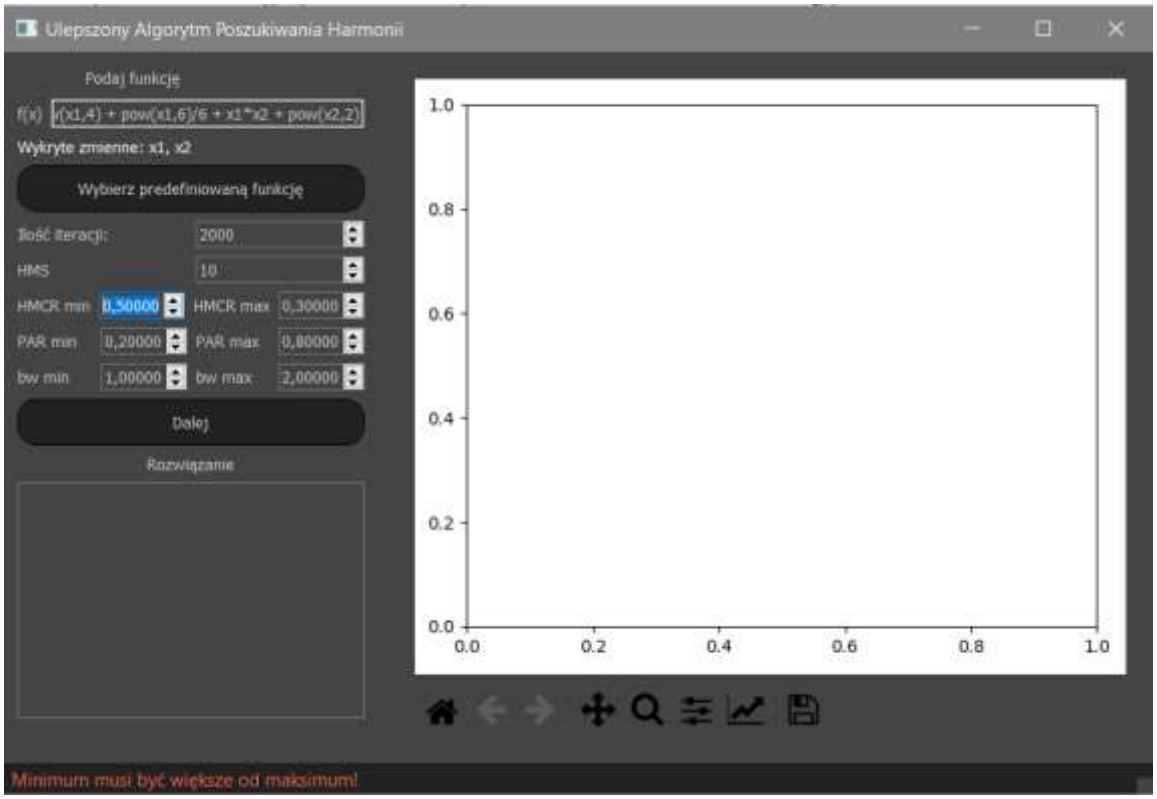
- Po wpisaniu funkcji pojawia się komunikat o wykrytych zmiennych:



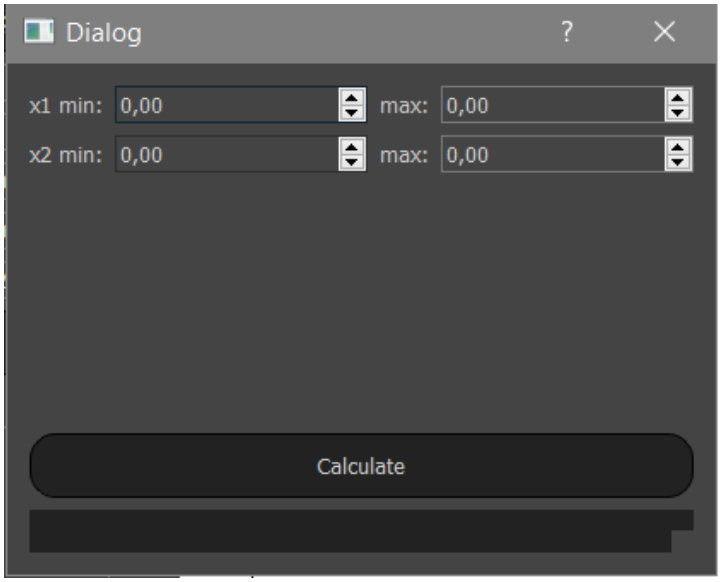
- Następne są pola wyboru parametrów funkcji:



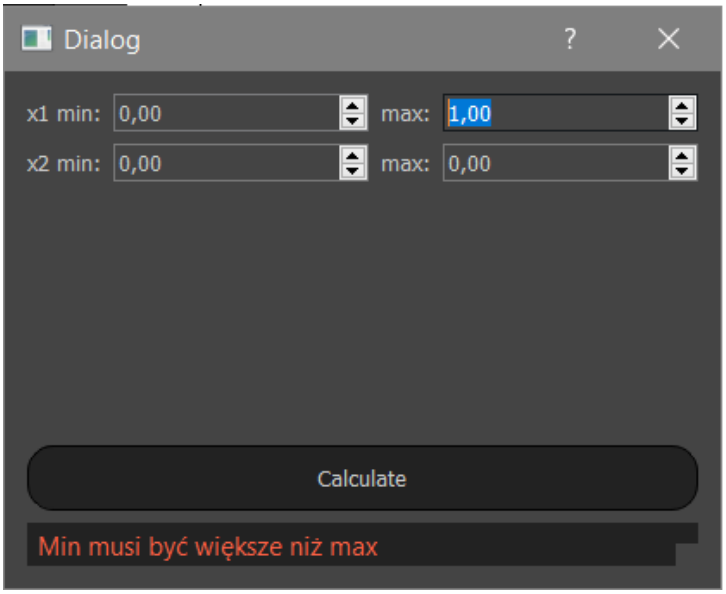
  - Ilość iteracji – określa liczbę wykonanych iteracji algorytmu.
  - HMS – określa rozmiar pamięci harmonii.
  - $HMCR_{min}$ oraz $HMCR_{max}$ – odpowiadają za przedział ewolucji wskaźnika uwzględniania pamięci.
  - PAR_{min} oraz PAR_{max} – odpowiadają za przedział ewolucji wskaźnika regulacji „wysokości tonu”.
  - bw_{min} oraz bw_{max} – odpowiadają za przedział szerokości pasma wyszukiwania harmonii.
- Jeśli jakiś parametr został wprowadzony niepoprawnie pojawi się stosowny komunikat, a możliwość przejścia dalej zostanie zablokowana.



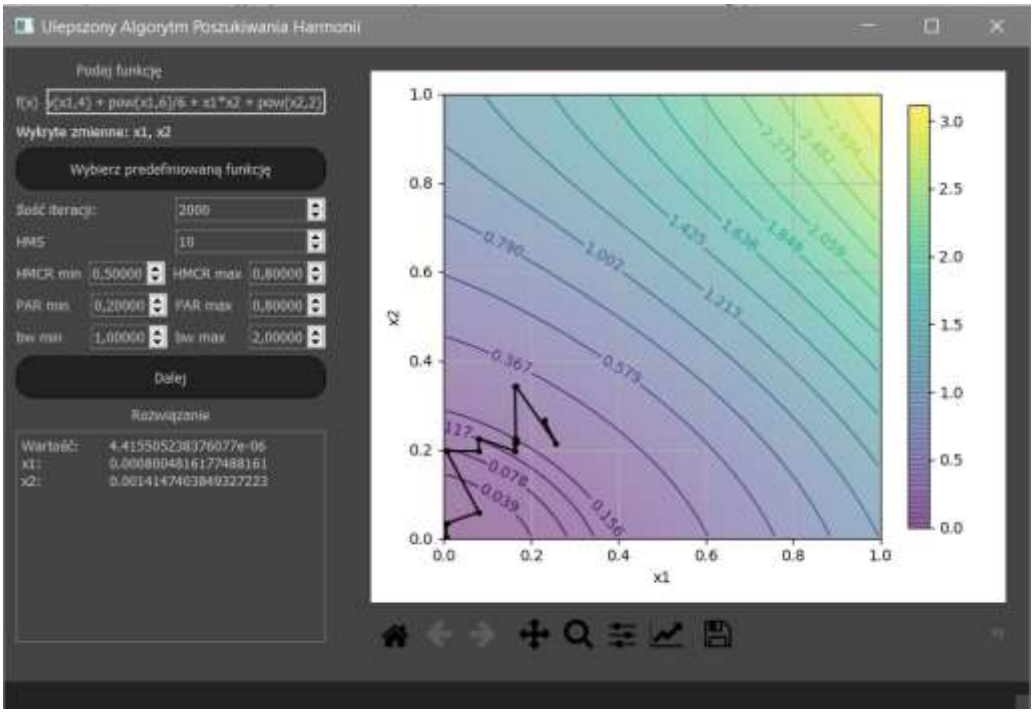
- Po naciśnięciu przycisku dalej pojawi się okno odpowiedzialne za pobieranie od użytkownika przedziałów wartości dla zmiennych wykrytych w równaniu:



- Jeśli jakiś parametr został błędnie wprowadzony, pojawi się stosowny komunikat:



- Po poprawnym wprowadzeniu danych i obliczeniu rozwiązania w oknie głównym pojawi się wykres warstwic funkcji oraz kolejne najlepsze punkty znalezione przez algorytm. Natomiast w polu rozwiązanie pojawią się wartości zmiennych oraz wartość funkcji w znalezionym punkcie.



- Wykres posiada Toolbar z pakietu matplotlib, który pozwala na podstawowe operacje na wykresie np. przybliżanie i zapis do pliku. Szczegółowy opis możliwośći znajduje się w: [2]

7 PORÓWNANIE WYNIKÓW DLA RÓŻNYCH PARAMETRÓW

Do rozważań wykorzystana została funkcja

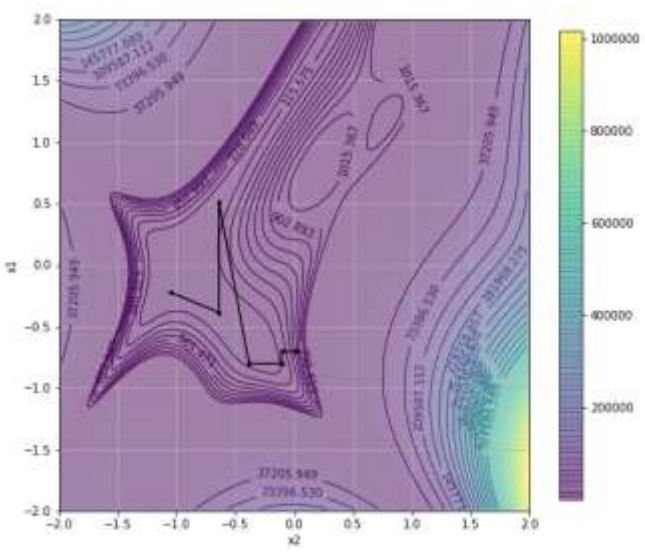
$$f(\vec{x}) = \{1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\} \times \\ \times \{3 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)\} \quad (7.1)$$

$$\min f(\vec{x}) = f(0,1) = 3 \quad (7.2)$$

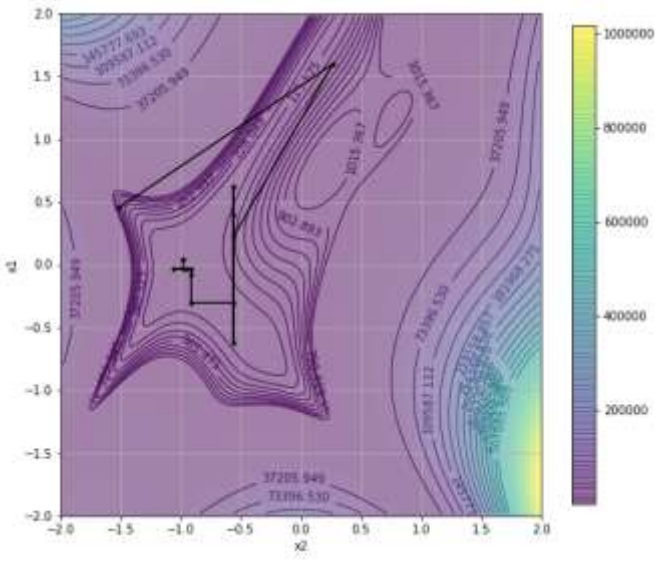
Dla tej funkcji uruchomiono algorytm z różnymi kombinacjami wartości parametrów. Wartości te dla każdego przypadku testowego przedstawiono w poniższej tabeli. Zakresy zmiennych to:

$$\begin{cases} -2 < x_1 < 2, \\ -2 < x_2 < 2. \end{cases} \quad (7.3)$$

ID	*L*	*HMS*	*HMCR*	*PAR*	*bw*
1	100	5	0.2-0.8	0.2-0.8	0-0.1
2	1000	5	0.2-0.8	0.2-0.8	0-0.1
3	1000	100	0.2-0.8	0.2-0.8	0-0.1
4	1000	100	0-0	0.2-0.8	0-0.1
5	1000	5	0-0	0.2-0.8	0-0.1
6	1000	5	1-1	0.2-0.8	0-0.1
7	1000	100	1-1	0.2-0.8	0-0.1
8	1000	10	1-1	0-0	0-0.1
9	1000	10	1-1	1-1	0-0.1
10	1000	10	0.2-0.8	0-0	0-0.1
11	1000	10	0.2-0.8	1-1	0-0.1
12	10000	10	0.2-0.8	0.2-0.8	0-0.1
13	10000	10	0.2-0.8	0.2-0.8	1-2



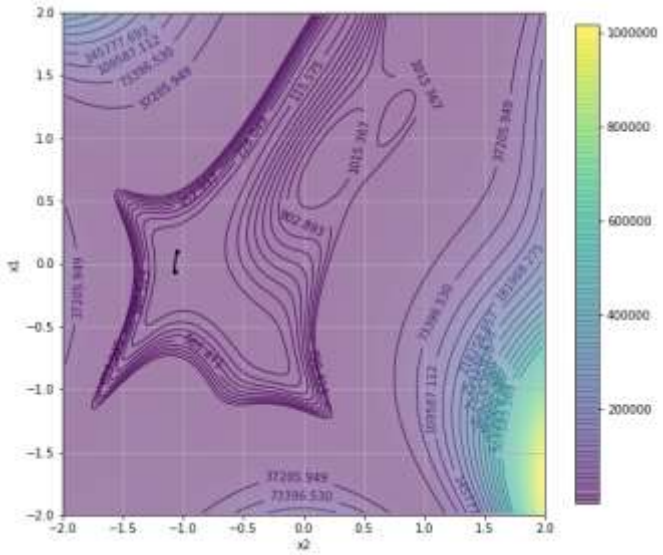
Rysunek 7.1) Przypadek 1 (f = 27.2765)*



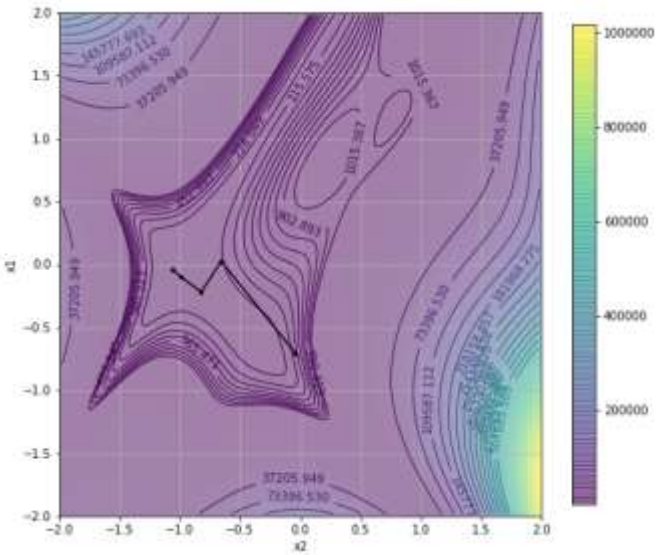
Rysunek 7.2) Przypadek 2 (f = 3,4156)*

Dla każdego z wielu wywołań przypadku pierwszego wynik był inny, więc 100 iteracji to za mało do dalszych rozważań (Rys. 7.1). W przypadku ustawienia ilości iteracji na 1000 program zwraca w większości przypadków wartość minimum zbliżoną do minimum globalnego (Rys. 7.2).

Po zwiększeniu parametru HMS w przypadku 3 zauważono, że algorytm znajduje mniej nowych wektorów, a wyniki są mniej dokładne (Rys. 7.3).



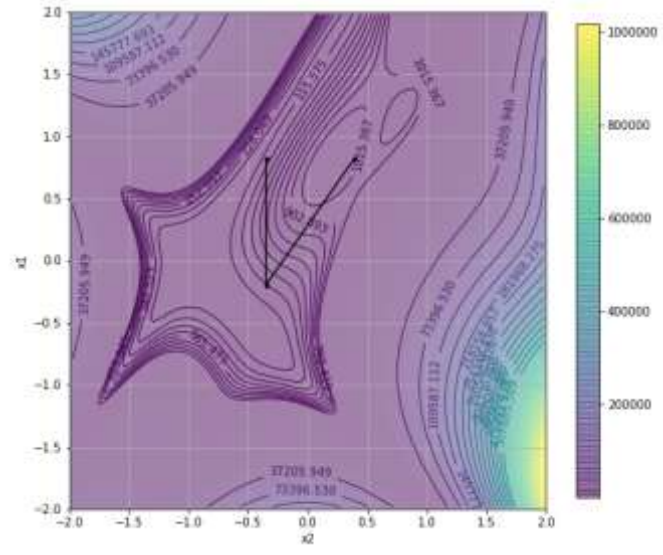
Rysunek 7.3) Przypadek 3 (f = 5.9362)*



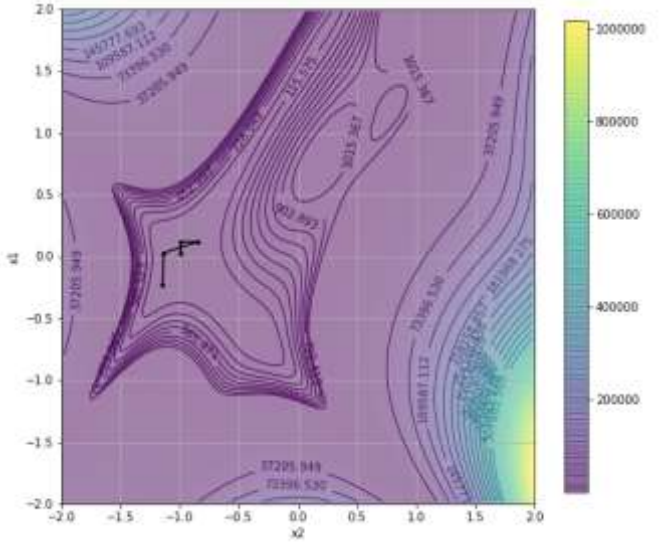
Rysunek 7.4) Przypadek 5 (f = 4.8874)*

Dla parametru $HMCR = 0$ zmiana żadnego innego parametru nie wywołuje zauważalnych zmian w działaniu algorytmu. Wyniki w tym przypadku są mało dokładne, ale zbiegają do optimum (Rys. 7.4). Jest to spowodowane tym, że dla parametru $HMCR = 0$, parametry PAR oraz bw nie są brane pod uwagę, gdyż algorytm nie dociera do momentu ich wykorzystania. parametr HMS również nie ma znaczenia, bo niezależnie od niego nowe punkty są wybierane w pełni losowo z całego zakresu wyszukiwania.

W przypadku ustawienia parametru $HMCR = 1$ można zauważyć, że przy niskiej wartości parametru HMS algorytm nie znajduje żadnego rozwiązania (Rys. 7.5). Ale jeśli zwiększy się wartość HMS algorytm zaczyna pracować lepiej, gdyż znajduje wartości zbliżone do optymalnych (Rys. 7.6). Wyjaśnić to można w następujący sposób: Przy $HMCR = 1$ wszystkie nowe punkty są generowane z uwzględnieniem punktów pamiętanych. Algorytm w każdej generacji może wyszukiwać nowe współrzędne w odległości bw od każdej ze współrzędnych pamiętanych. W takim razie większa ilość pamiętanych punktów HMS oznacza, że w większej części dziedziny algorytm może wyszukiwać nowego rozwiązania.

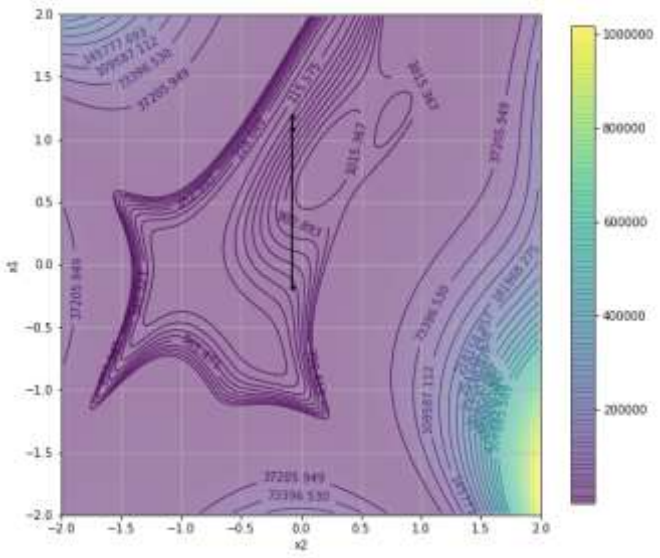


Rysunek 7.5) Przypadek 6 (f = 168.451)*

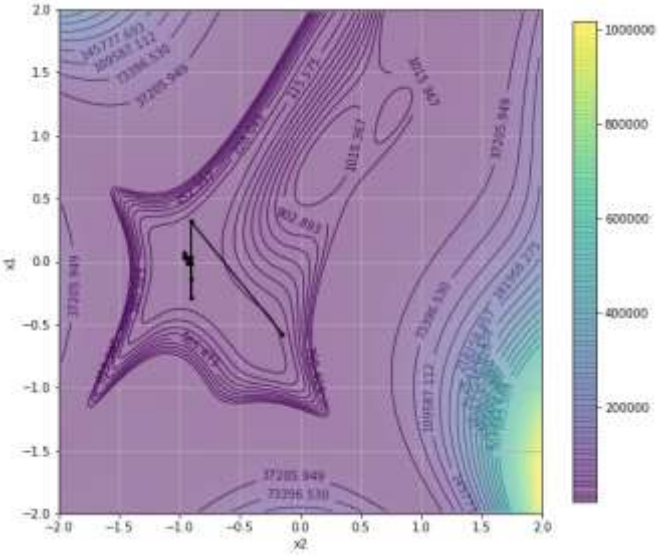


Rysunek 7.6) Przypadek 7 (f = 3.1735)*

W przypadku 8, gdzie parametr $HMCR = 1$, a $PAR = 0$ można zauważyć, że dla każdego wywołania znajdywane są jedynie pojedyńcze punkty (Rys. 7.7). Ma to związek, że algorytm jest zmuszony do wyszukiwania najoptymalniejszego rozwiązania jedynie z początkowo wylosowanych HMS punktów. Ustawienie parametru $PAR = 1$ w przypadku 9 nie wniosło istotnej zmiany w wynikach.



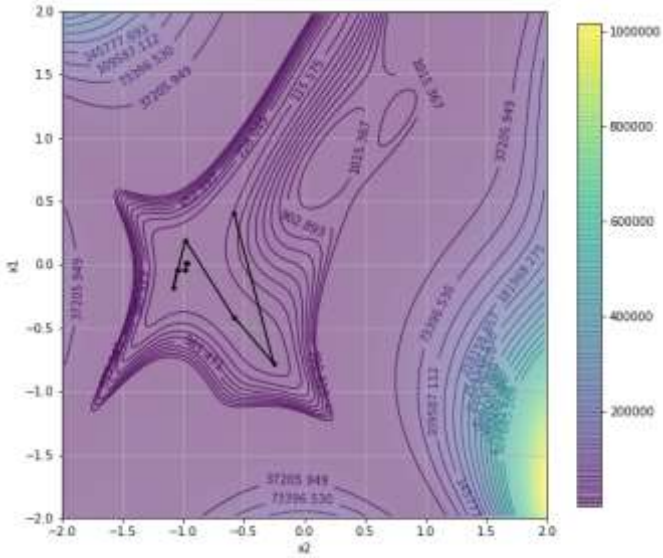
Rysunek 7.7) Przypadek 8 (f = 263.993)*



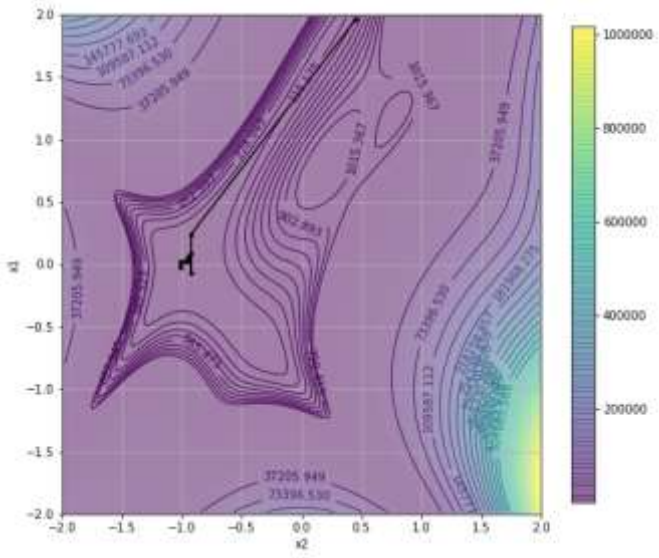
Rysunek 7.8) Przypadek 10 (f = 3,679)*

W przypadku 10 (Rys. 7.8), kiedy parametr $HMCR$ ewoluuje między 0.2, a 0.8 można zauważyć poprawę w porównaniu do $HMCR = 0$. Dzieje się to dzięki temu, że nowe współrzędne nie są losowane jedynie z całej dziedziny, ale również uwzględniane są najlepsze pamiętane rozwiązania.

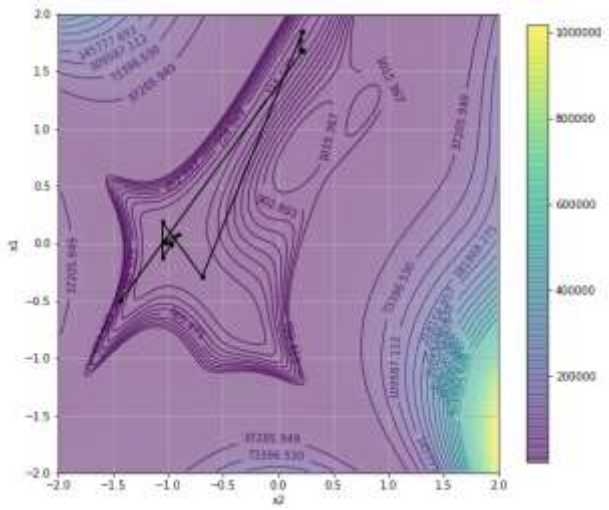
Przypadek 11 (Rys. 7.9) pokazuje, że dzięki dostosowywaniu wysokości tonu (gdy $PAR > 0$) otrzymywane są dokładniejsze wyniki niż bez, a jeszcze lepsze wyniki otrzymuje się po ustawieniu parametru PAR na zakres (0.2 – 0.8), jak w przypadku 12 (Rys. 7.10).



Rysunek 7.9) Przypadek 11 (f = 3.1178)*



Rysunek 7.10) Przypadek 12 (f = 3.052)*



Rysunek 7.11) Przypadek 13 (f = 3.1479)*

Po dokładnym przeanalizowaniu przypadku 13 (Rys. 7.11) można zauważyć, że po zwiększeniu parametru bw algorytm przeszedł przez wszystkie minima lokalne, ale ostatecznie odnalazł minimum globalne, choć z mniejszą dokładnością niż dla parametru bw o niższej wartości.

8 WNIOSKI

Na podstawie obserwacji wyników testów przedstawionych w poprzednim rozdziale oraz założeń projektowych wysnuto następujące wnioski:

- Paramaetr L – ilość iteracji – wpływa na dokładność wyniku w największym stopniu pod warunkiem, że pozostałe parametry zostaną dobrane tak, aby nie blokować funkcjonalności algorytmu. Duża ilość iteracji powoduje zwiększenie czasu obliczeń, ale też dokładności otrzymywanego wyniku oraz daje pewność, że zostanie znalezione rzeczywiste optimum dla danego zakresu.
- Parametr HMS – ilość wektorów składowanych w pamięci. Duża jego wartość zwiększa czas obliczeń. Dla wartości $HMCR$ bliskich 1 parametr HMS ma duży wpływ na jakość, a przy wartościach $HMCR$ bliskich 0 HMS nie ma dużego wpływu na jakość wyniku.
- Parametr $HMCR$ – wskaźnik rozważania pamięci – wpływa na rzeczywisty determinizm otrzymywania optymalnego wyniku. Im wyższy tym bardziej algorytm skupia się na wyszukiwaniu w pobliżu odnalezionych minimów, ale może to spowodować ugrzęźnięcie w minimum lokalnym. Determinuje to konieczność zwiększania parametru HMS wraz ze zwiększaniem parametru $HMCR$, co w pewnym stopniu może uchronić algorytm przed zatrzymaniem w minimum lokalnym.
- Parametr PAR – wskaźnik dostosowywania wartości elementu wektora. Ma istotne znaczenie przy wysokich wartościach $HMCR$, gdyż umożliwia on wyszukiwanie nowych wartości elemetów wektora w paśmie zamiast w linii, co pozwala zbliżać się do optimum. Ma on też duży związek z parametrem bw, gdyż dla wysokich wartości bw osłabia on skłonność algorytmu o wysokim wskaźniku $HMCR$ do utknięcia w optimach lokalnych, a obniżanie wartości bw powoduje zwiększenie dokładności wyniku.
- Zakresy wartości zmiennych – Im większe zakresy tym mniej dokładny wynik. Dodatkowo przez zmianę zakresów w pobliżu optimów można wymusić odnajdywanie optimów lokalnych.

9 PRZYKŁADY TESTOWE

9.1 ZADANIE TESTOWE NR 1: ZADANIE Z LITERATURY [1]

$$f(\vec{x}) = \{1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\} \times \\ \times \{3 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)\} \quad (9.1)$$

Funkcja ta posiada 4 minima lokalne, jedno z nich jest optimum globalnym

- $f(1.2, 0.8) = 840.0$
- $f(1.8, 0.2) = 84.0$
- $f(0.6, 0.4) = 30.0$
- $f(0.0, 1.0) = 3.0$ – optimum globalne

Parametry algorytmu:

HMCR		***PAR***		***HMS***	***bw***		***L***
min	max	min	max		min	max	
0,94999*	0,95*	0,35	0,99	7	1E-6	4	6000

*Zaimplementowany algorytm posiada ewolucyjny parametr $HMCR$, natomiast wersja z literatury posiada ten parametr stały o wartości 0,95. Dlatego zdecydowano się użyć jak najmniejszego przedziału wokół tej wartości.

Zakres zmiennych $x_1, x_2 \in \langle -50{,}50 \rangle$.

Według literatury [1] optymalne rozwiązanie znaleziono po 2340 iteracjach a otrzymane wartości wynosiły:

- $x_1 = 0.0$,
- $x_2 = -1{,}000001$,
- $f_g^*(\vec{x}) = 3.0$

gdzie

$f_g^*(\vec{x})$ – znaleziona wartość optymalna dla danego przypadku

Autorzy zauważyli tutaj nieścisłość w literaturze, gdyż wedle wcześniejszych informacji optimum globalne znajduje się w puncie $(0.0, 1.0)$, natomiast rozwiązanie i wykres sugeruje, że optimum znajduje się w puncie $(0.0, -1.0)$.

W przypadku zastosowanej tutaj implementacji dla przedstawionych powyżej parametrów nie udało się osiągnąć nawet porównywalnego wyniku. Przykładowy wynik:

- $x_1 = 5.333802627044513$
- $x_2 = 2.6051521013121786$
- $f_g^*(\vec{x}) = 28846.53430192294$
- Rozwiązanie znalezione po 2879 iteracjach

Próbę kilkukrotnie powtarzano i wyniki były powtarzalne. Autorzy nie wiedzą, dlaczego zachodzi aż taka rozbieżność w wynikach. Sytuację udało się znacząco poprawić rozszerzając zakres ewolucji Parametru $HMCR$ w zakresie $\langle 0.3, 0.95 \rangle$. Zmiana ta poprawiła w dużej mierze jakość generowanego rozwiązania. Przykładowy wynik:

- $x_1 = 1.8710131259234402$
- $x_2 = 0.18954301912516058$
- $f_g^*(\vec{x}) = 117.59417582875817$
- Rozwiązanie znalezione po 5896 iteracjach

Kolejną dość istotną poprawę wprowadziła zmiana ilości iteracji. Rozszerzenie jej 10-krotnie, czyli do 60000, pozwoliło na osiągnięcie wyniku przybliżonego do rzeczywistego globalnego optimum. Przykładowy wynik:

- $x_1 = -0.0071832624629522$
- $x_2 = -1.018756389556522$
- $f_g^*(\vec{x}) = 3.140001134341988$
- Rozwiązanie znalezione po 51239 iteracjach

9.2 ZADANIE TESTOWE NR 2: ZMODYFIKOWANA FUNKCJA HIMMELBLAU'A

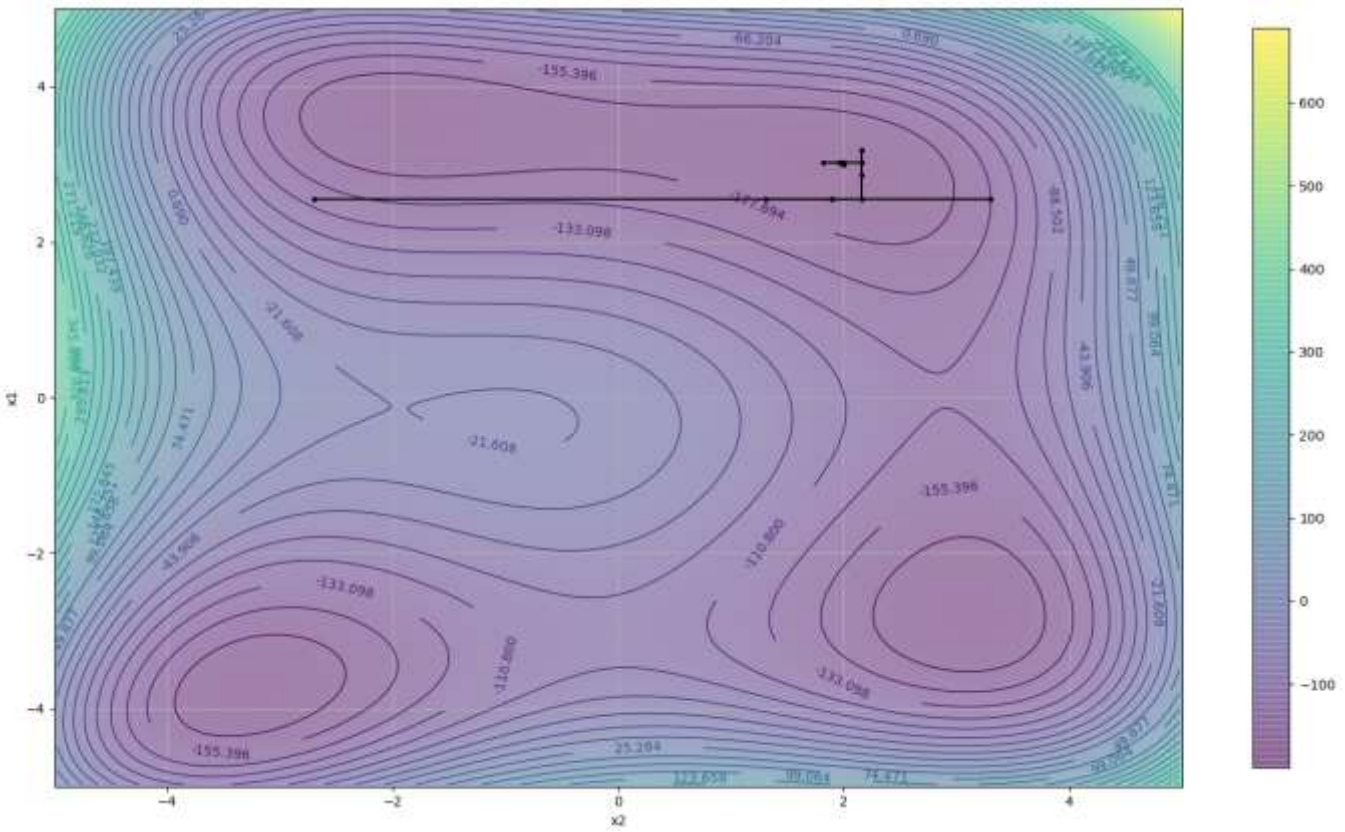
$$\min f(\vec{x}) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 - 200 \tag{9.2}$$

Funkcja posiada cztery minima lokalne o tej samej wartości funkcji celu.

Parametry algorytmu:

HMCR		***PAR***		***HMS***	***bw***		***L***	x_1	x_2
min	max	min	max		min	max			
0,3	0,95	0,35	0,99	7	1E-6	4	20000	$\langle -5{,}5 \rangle$.	$\langle -5, 5 \rangle$.

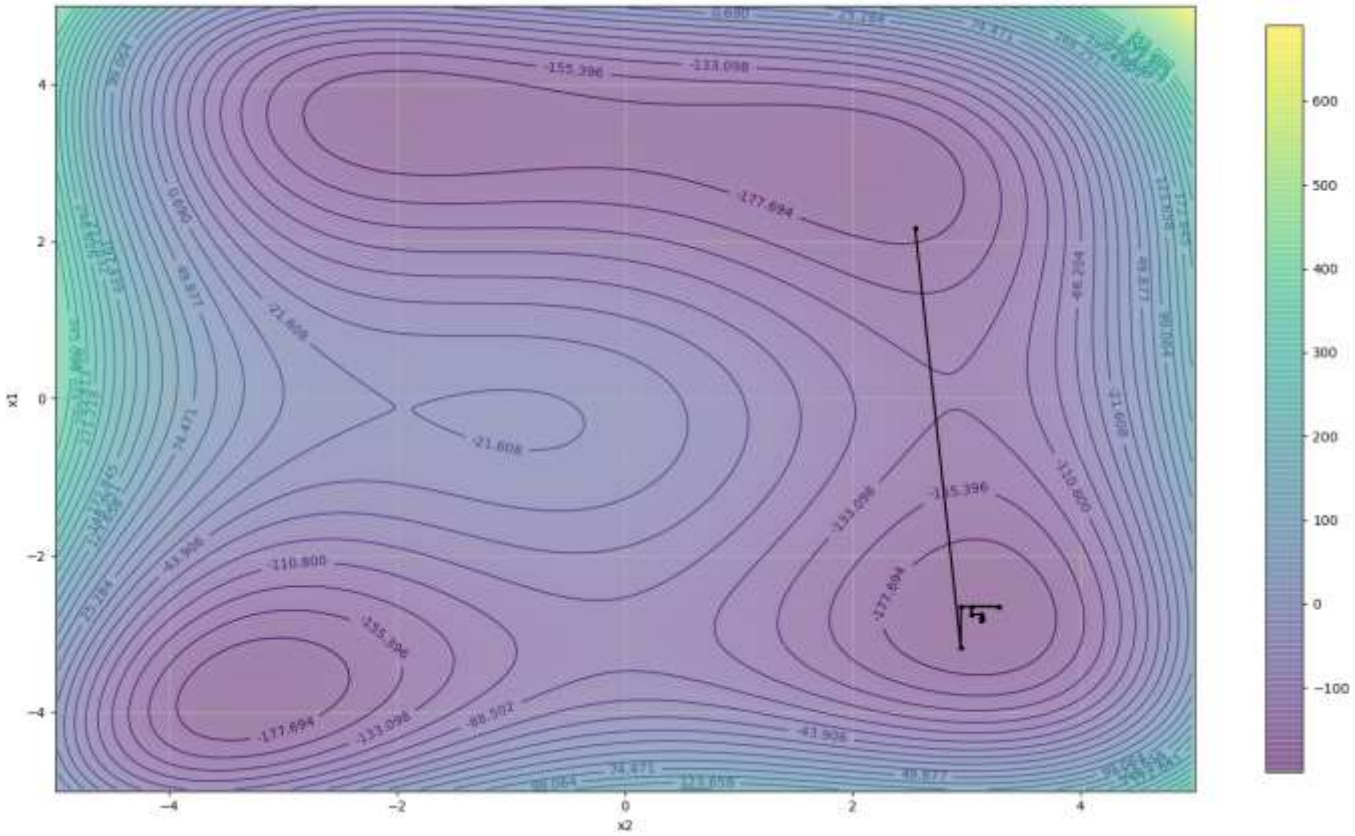
Wykresy warstwic funkcji:



Rysunek 9.1) Próba pierwsza.

Wynik dla próby pierwszej:

- $f_g^*(\vec{x}) = -199.99900168823498$
- $x_1 = 3.0051367992550855$
- $x_2 = 2.0001921266098845$



Rysunek 9.2) Próba czwarta

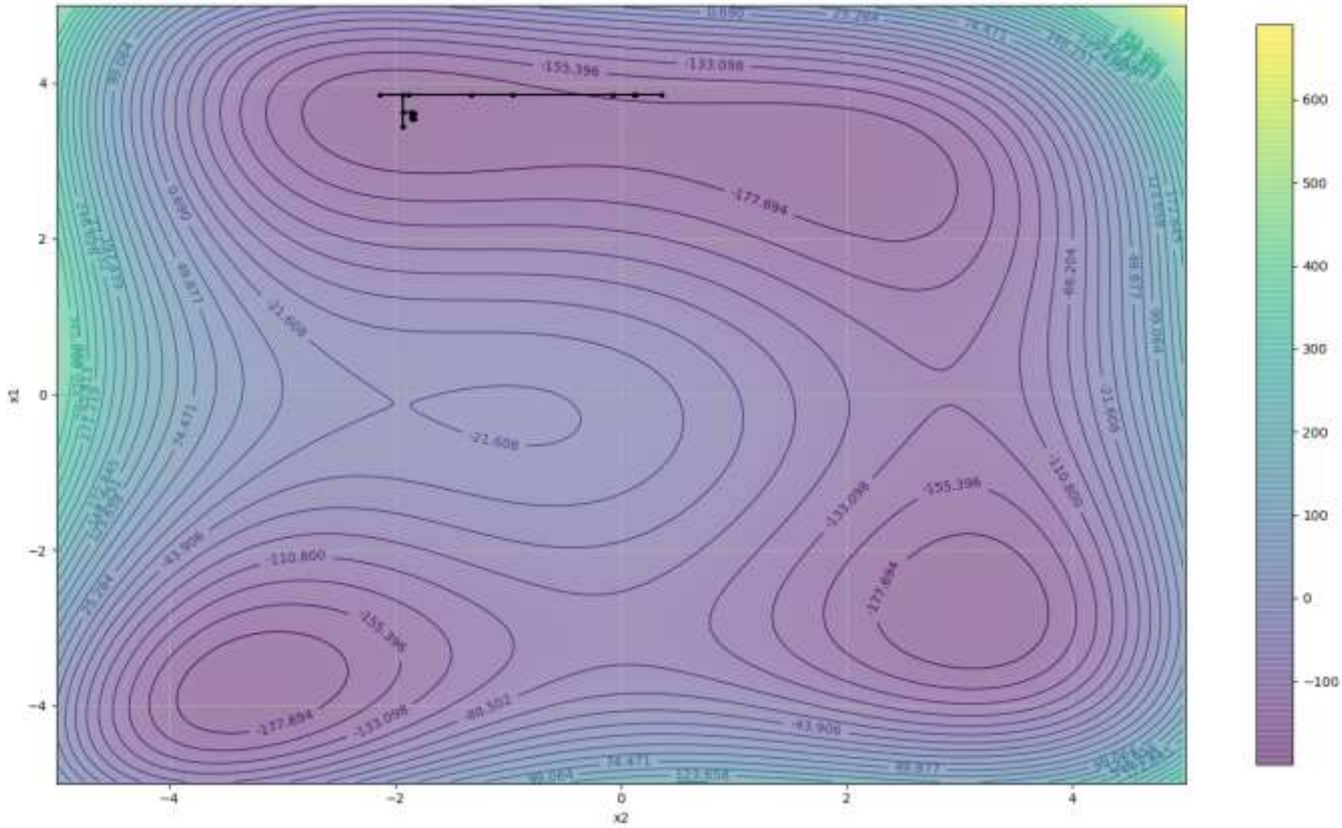
Wynik dla próby czwartej:

- $f_g^*(\vec{x}) = -199.9999181374237$
- $x_1 = -2.80392037157191$
- $x_2 = 3.1303559519995616$

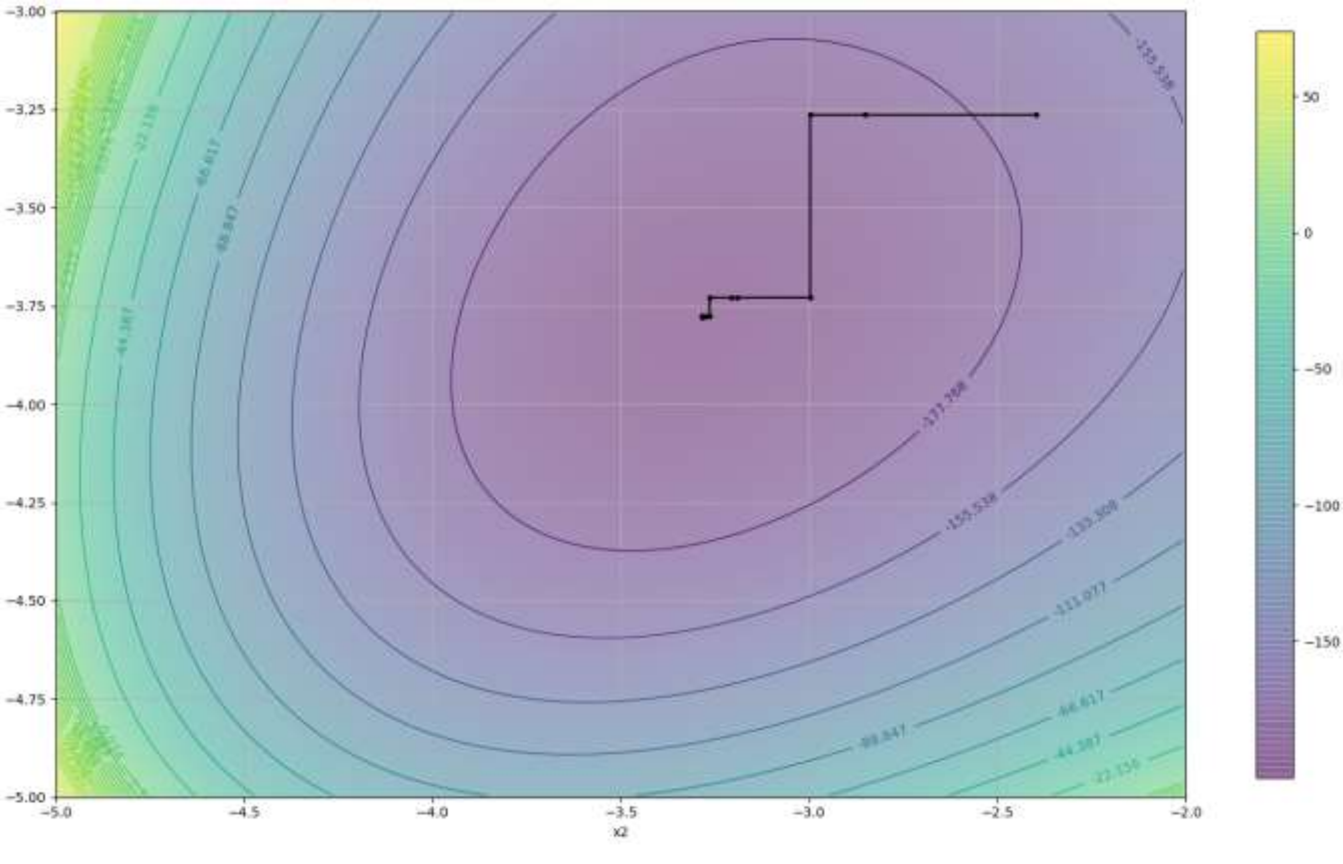
Wynik dla próby piątej:

- $f_g^*(\vec{x}) = -199.99997602144197$
- $x_1 = 3.5838129057307295$
- $x_2 = -1.8485316299568488$

Na Rysunkach 9.1-9.3 widać 4 obszary, w których może znajdować się minimum. Wykonując wielokrotnie algorytm nie udało się znaleźć minimum w obszarze $x_1 \in \langle -5,-3 \rangle$, $x_2 \in \langle -5,-2 \rangle$, dlatego zdecydowano się sprawdzić ręcznie, zmieniając zakres wektora $\vec{x}$ do wspomnianego wyżej zakresu. Dla tych rozwiązanych przez algorytm wartości minimów były porównywalne i były bliskie wartości -200. Wykres po pwrowadzonej zmianie przedstawiono na Rysunku 9.4.



Rysunek 9.3) Próba piąta



Rysunek 9.4) Próba po zmianie zakresu.

Natomiast otrzymane wyniki to:

- $f_g^*(\vec{x}) = -199.99998078896024$
- $x_1 - 3.778728051111024$
- $x_2 - 3.282845871321974$

Zatem przybliżone pozycje lokalnych minimów i ich wartości to:

- $f(3.01, 2.00) = -200{,}00$
- $f(-2.80, 3.13) = -200{,}00$
- $f(3.58, -1.85) = -200{,}00$
- $f(-3.78, -3.28) = -200{,}00$

9.3 ZADANIE TESTOWE NR 3

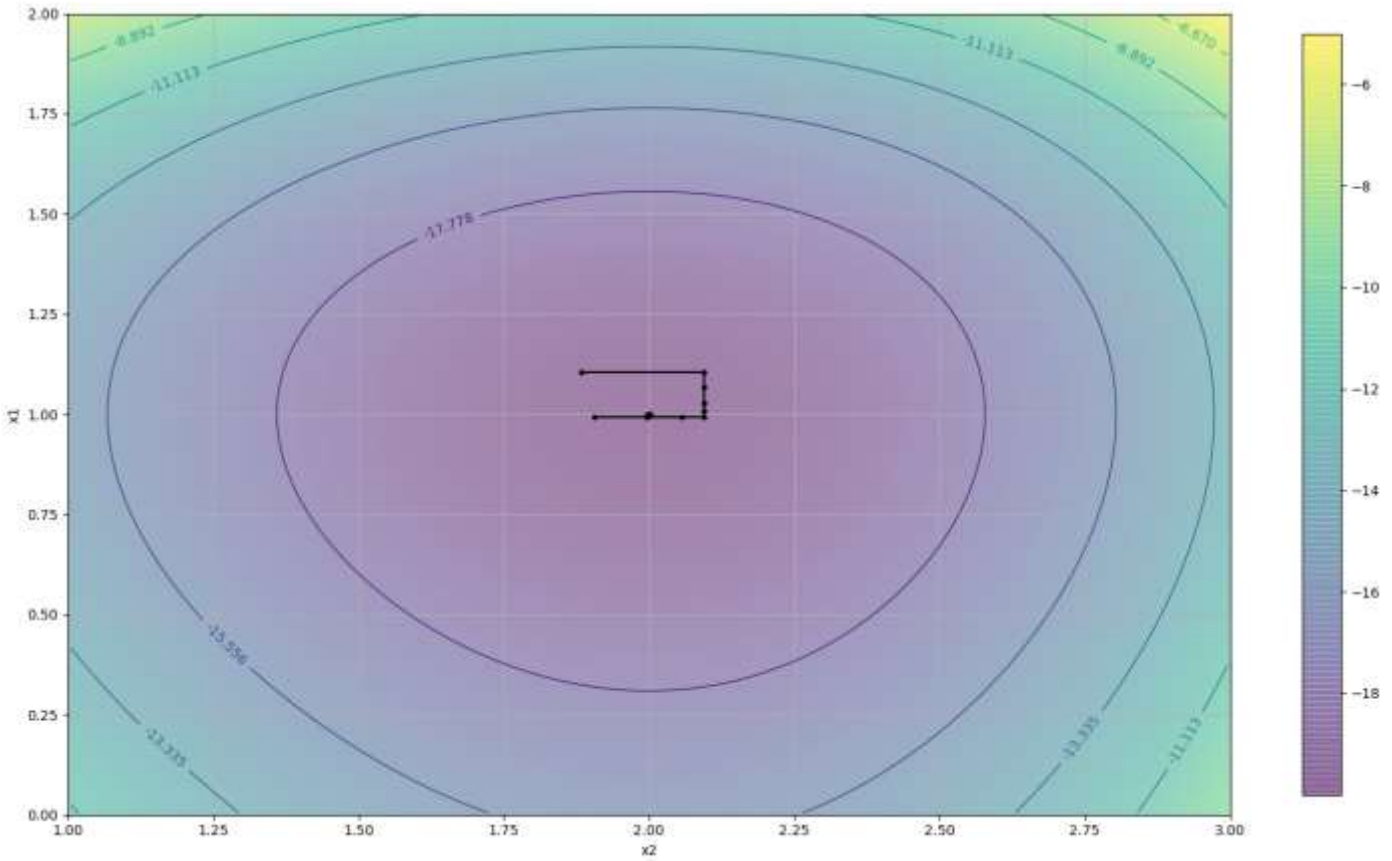
$$f(\vec{x}) = 2x_1^3 + x_2^3 - 6x_1 - 12x_2 \tag{9.3}$$

MINIMUM

HMCR		***PAR***		***HMS***	***bw***		***L***	x_1	x_2
min	max	min	max		min	max			
0,2	0,95	0,2	0,8	7	1E-4	2	50000	⟨0,2⟩.	⟨1, 3⟩.

Dla parametrów przedstawionych w tabeli po 20044 iteracjach znaleziono minimum lokalne funkcji:

$f_g^*(\vec{x}) = f(1.00,\ 2.00) = -20{,}00$



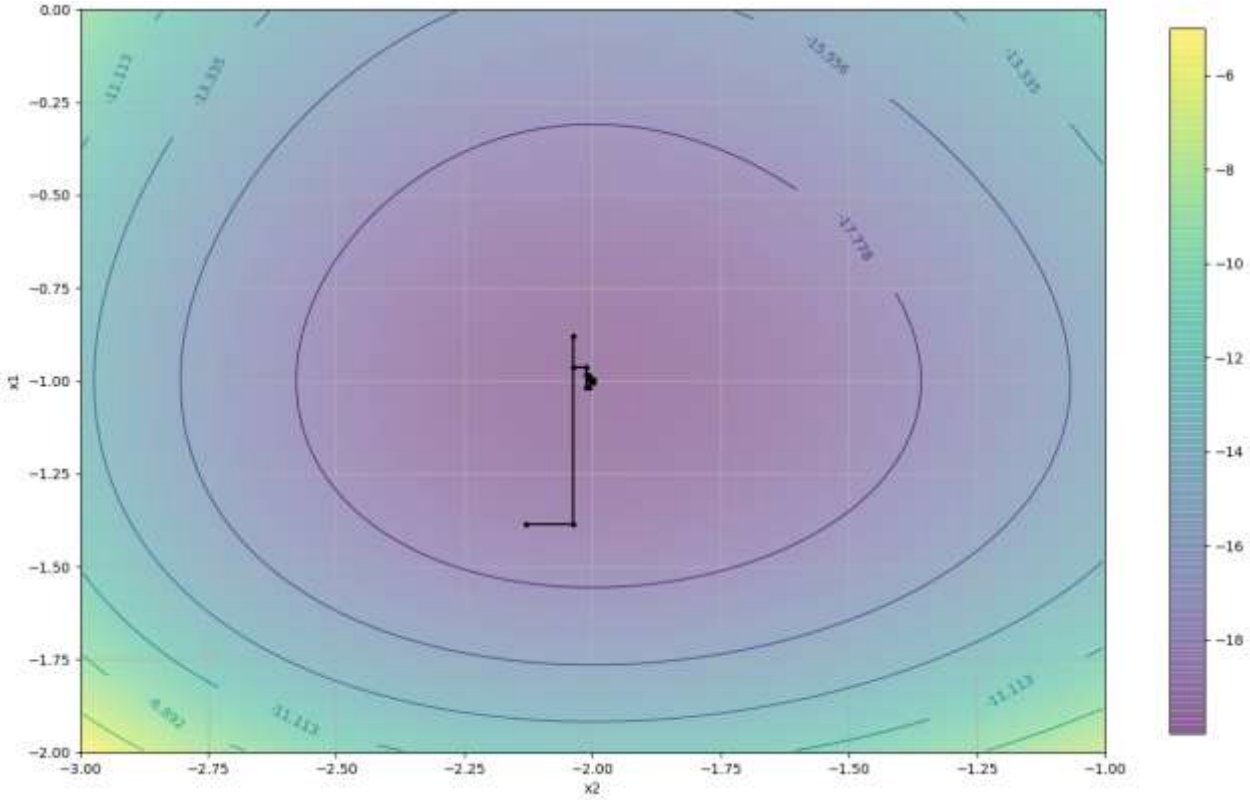
Rysunek 9.5) Próba wyszukania minimum: f(x_1, x_2)

MAXIMUM

HMCR		PAR		HMS	bw		L	x_1	x_2
min	max	min	max		min	max			
0,2	0,95	0,2	0,8	7	1E-4	2	50000	$\langle -2,0\rangle$.	$\langle -3,-1\rangle$.

Aby znaleźć maksimum przemnożono funkcję przez (-1). Po 28557 iteracjach znaleziono maksimum lokalne funkcji:

$f(-1.00, -2.00) = 20{,}00$



Rysunek 9.6) Próba wyszukiwania maksimum: -f(x_1, x_2)

9.4 ZADANIE TESTOWE NR 4: TRÓJGARBNY WIELBŁĄD

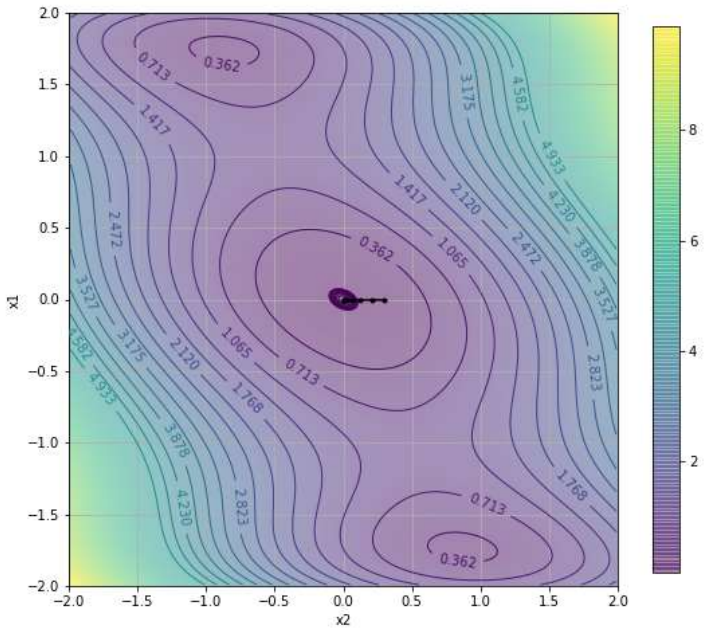
$$f(x_1, x_2) = 2x_1^2 - 1.05x_1^4 + \frac{x_1^6}{6} + x_1x_2 + x_2^2 \tag{9.4}$$

MINIMUM GLOBALNE

$HMCR$		PAR		HMS	bw		L	x_1	x_2
min	max	min	max		min	max			
0,2	0,8	0,2	0,8	10	1E-6	2	2000	$\langle -2,2\rangle$.	$\langle -2, 2\rangle$.

Dla parametrów przedstawionych w tabeli znaleziono minimum globalne f_g^* po 1722 iteracjach. Przedstawia się ono następująco:

$$f_g^* = f(0,0) = 0$$

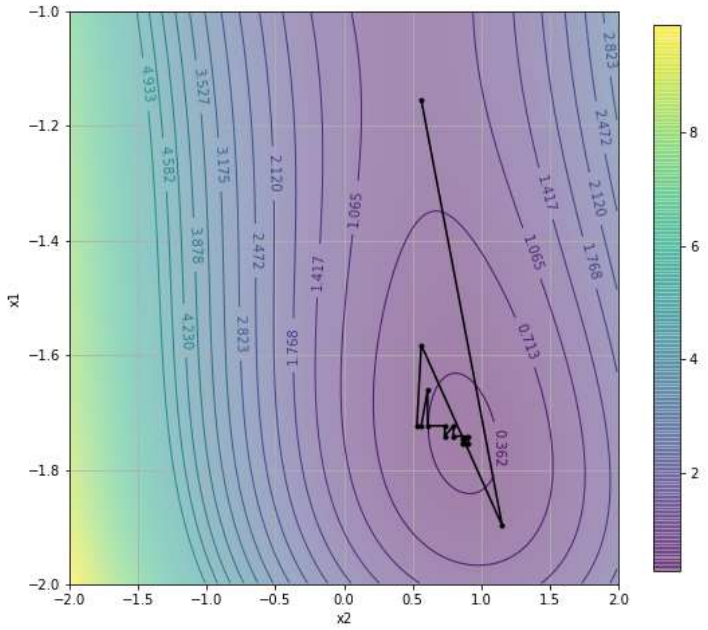


Rysunek 9.7) Wyszukiwanie minimum globalnego

Na wykresie można zaobserwować istnienie trzech minimów lokalnych, których wartości można obliczyć zmieniając zakresy zmiennej x_1.

Dla $x_1 \in \langle -2,-1 \rangle$ po 462 iteracjach znaleziono minimum lokalne f_1^*.

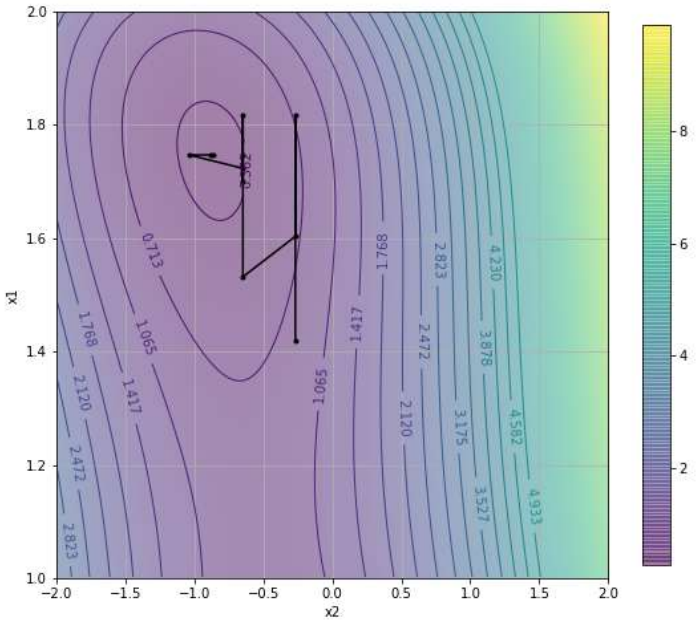
$$f_1^* = f(-1.7475,0.8738) = 0.2986$$



Rysunek 9.8) Wyszukiwanie pierwszego minimum lokalnego

Dla $x_1 \in \langle 1,2 \rangle$ po 1753 iteracjach znaleziono minimum lokalne f_2^*.

$$f_2^* = f(1.7467, -0.8643) = 0.2987$$



Rysunek 9.9) Wyszukiwanie drugiego minimum lokalnego

Odpowiednio dobierając parametry algorytmu wyznaczono wartości minimów dla każdego z trzech wielbłądzich garbów:

$$f_g^* = f(0,0) = 0$$

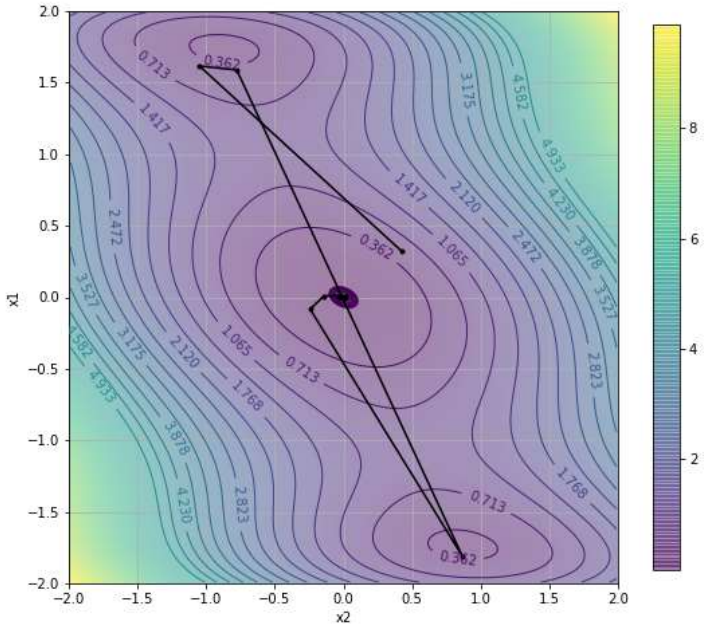
$$f_1^* = f(-1.7475, 0.8738) = 0.2986$$

$$f_2^* = f(1.7467, -0.8643) = 0.2987$$

WYKRYCIE WSZYSTKICH MINIMÓW

$HMCR$		PAR		HMS	bw		L	x_1	x_2
min	max	min	max		min	max			
0,2	0,3	1	1	2	1	2	2000	$\langle -2,2 \rangle$.	$\langle -2, 2 \rangle$.

Wykorzystując wnioski z rozdziału 8 wysnute na podstawie porównywania pracy algorytmu dla różnych parametrów dobrano parametry algorytmu tak, aby zwiększyć prawdopodobieństwo wejścia algorytmu we wszystkie minima lokalne. Co przedstawiono na rysunku 9.10. Udało się tego dokonać dla parametrów podanych w powyższej tabeli.



Rysunek 9.10) Przejście algorytmu przez wszystkie minima lokalne

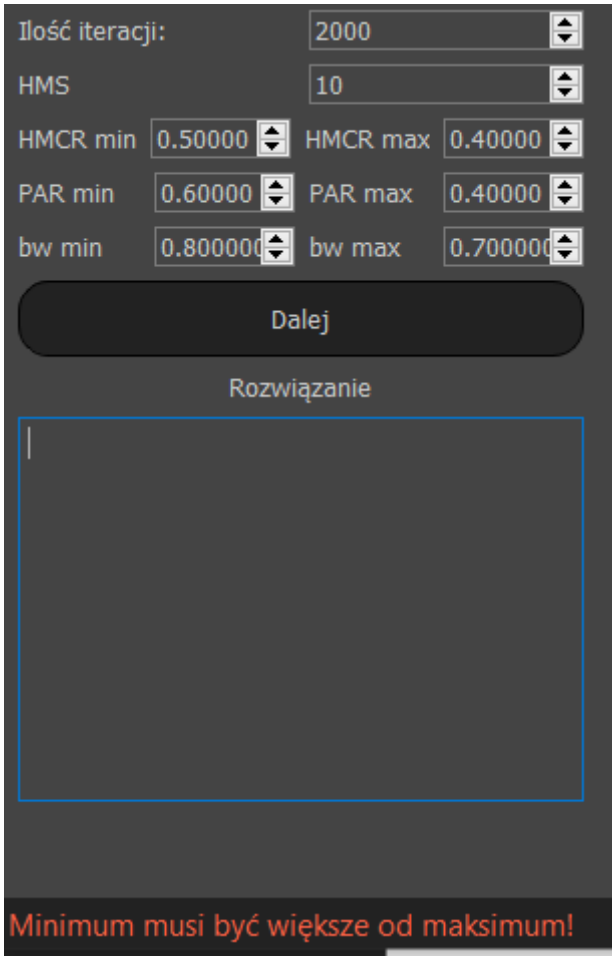
10 OPIS ZABEZPIECZEŃ PROGRAMU PRZED BŁĘDAMI

Program został zabezpieczony przed podstawowymi błędami. Program posiada podstawowe zabezpieczenia, które mogłyby wystąpić. Poniżej przedstawiono błędy przed którymi program jest chroniony:

- Nieodpowiednie wpisanie funkcji – W przypadku błędnego wprowadzenia funkcji w polu $f(x)$ poniżej wyświetlany jest na czerwono komunikat o tym, jaki błąd się pojawił, bądź na biało komunikat o znalezionych we wzorze zmiennych i w przypadku błędu blokowany jest przycisk uruchomienia algorytmu. Przykłady komunikatów wraz z przykładami ich wywołań:

- Dzielenie przez zero – Jeśli funkcja zostanie wpisana tak, że występuje w niej dzielenie przez 0 (nawet utajone) niezależnie od zmiennych, czyli w całej ich dziedzinie program również nie dopuści do uruchomienia algorytmu.

- Wpisanie minimum większego od maksimum – W przypadku wpisania w polach zakresów parametrów $HMCR, PAR, bw$ liczby minimum większej niż maksimum w pasku u dołu wyświetlany jest stosowny komunikat oraz blokowany jest przycisk „Dalej”. To zabezpieczenie jest zaimplementowane podwójnie – w klasie interfejsu algorytmu (I_IHS) oraz w części front-end:



- Wpisanie wartości parametru poza zakresem – Pola wartości parametrów, które powinny się znajdować w określonych zakresach nie pozwalają na wpisanie wartości spoza zakresu. To zabezpieczenie również zostało zaimplementowane dwukrotnie.

11 PROBLEMY NAPOTKANE PODCZAS PISANIA PROGRAMU ORAZ ICH ROZWIĄZANIA

Podczas pisania programu napotkano szereg problemów i wszystkie z nich zostały rozwiązane. Opis tych problemów oraz sposobów ich rozwiązania opisano poniżej.

- Problem: Po każdym odrysowaniu wykresu pojawiał się nowy colorbar natomiast stary pozostawał. Rozwiązanie: Obiekt początkowo był zdefiniowany lokalnie. Przeniesiono go to konstruktora klasy i teraz jest jej atrybutem. To pozwala sprawdzić, czy obiekt został wcześniej stworzony. Jeśli nie jest tworzony nowy colorbar. Jeśli istniał wcześniej to jest usuwany i tworzony od początku.
- Problem: Błąd informujący o różnych rozmiarach wektorów podczas rysowania wykresu.
  Rozwiązanie: Mimo iż wektory miały taki sam rozmiar, były różnych typów. Zmienne x były generowane przez pakiet NumPy, a więc były typu NumPy.array. Natomiast wartości funkcji przechowywaliśmy w

postaci listy. Niejawne konwersje nie rozwiązywały tego problemu, dlatego zdecydowano się ujednolicić strukturę danych i przechowywać wartości zmiennych x, oraz wartości funkcji w obiekcie typu: NumPy.array.

- Problem: Wykres był rysowany tylko gdy zakresy zmiennych x_1 i x_2 były takie same. W innym przypadku był rzucany wyjątek o niepoprawnym rozmiarze wektorów.
  Rozwiązanie: Początkowo do generowania wartości zmiennych x wykorzystano funkcję, która generowała kolejne wartości o określonym skoku. Zdecydowano się ją zmienić na NumPy.linspace(). Funkcja ta pozwala generować wektory o określonym rozmiarze z wartościami równo rozmieszczonymi w danym przedziale.
- Problem: Wektor z trajektorią rozwiązań posiadał dużo powtórek.
  Rozwiązanie: Początkowo dodawano nowe rozwiązanie za każdym razem, gdy zostało ono umieszczane w HM. Natomiast nie każde umieszczane tam rozwiązanie było najlepsze w danym momencie. Dlatego zdecydowano się je dodawać tylko w przypadku, gdy było lepsze od najlepszego w HM.
- Problem: Gdy nie wprowadzono danych do okna z zakresem zmiennych i je wyłączono następowało rzucenie wyjątku.
  Rozwiązanie: Po zamknięciu okna należy sprawdzać status jaki zwraca okno. Dalsze kroki zdecydowano się wykonywać tylko w przypadku, gdy status jest „Accepted".
- Problem: Brak dostępu do obiektów okien BandwidthDialog i FunctionDialog poza funkcją setupUI().
  Rozwiązanie: Problem był w sposobie tworzenia nowego okna. Do ich projektowania wykorzystano program QtDesigner. Generował on kod, do którego należało przesłać osobny obiekt dziedziczący po QObject. Powodowało to brak możliwości napisania własnych metod operujących na danym obiekcie. Aby rozwiązać ten problem należało stworzyć osobną klasę dziedziczącą klasę wygenerowaną przez QtDesigner oraz klasę QObject (lub pochodną w zależności od potrzebnych metod). To pozwalało uzyskać dostęp do metod pozwalających na manipulację oknem.
- Problem: Długie wykonywanie obliczeń.
  Rozwiązanie: Problem wynikał z samej specyfikacji języka (interpretowany) oraz sposobu dostępu do elementów listy, wektora. W zależności od wykorzystywanej struktury danych, dostęp do elementu n wymaga pewnego narzutu obliczeniowego. Aby go zmniejszyć wszędzie tam gdzie istniała potrzeba skorzystania z danego elementu podczas iterowania struktury danych zdecydowano się skorzystać z funkcji „enumerate()" zamiast konstrukcji „in range(len())". Spowodowało to istotne skrócenie czasu obliczeń.
- Problem: Używanie parsera przy każdym wywołaniu funkcji powodowało zbyt duży narzut obliczeniowy.
  Rozwiązanie: Parser wykorzystano, aby upewnić się, że funkcja jest wpisana bezbłędnie oraz aby znaleźć zmienne we wpisanej funkcji, a sama funkcja jest przekazywana do dynamicznie generowanego wyrażenia lambda objective_function przy wykorzystaniu funkcji exec(), a następnie to wyrażenie jest wywoływane przez kolejną dynamicznie definiowaną funkcję lambda compute() z odpowiednią liczbą parametrów.
- Problem: Podczas implementacji algorytmu IHS wykorzystywano informacje zawarte rozmaitych artykułach w internecie. Były one niespójne i ciężko było złożyć je w jeden działający algorytm.
  Rozwiązanie: W artykule [1] znaleziono kompletny algorytm wraz ze schematem blokowym, którego implementacja przebiegła bezbłędnie.
- Problem: Z powodu braku dokumentacji korzystanie z pamięci HM będącej macierzą wymagało odrobinę czasu, aby upewnić się który wymiar jest wielkością wektora, a który jest rozmiarem HMS.
  Rozwiązanie: Zmieniono budowę pamięci HM z macierzy na jednowymiarową tablicę wektorów, gdzie wektor jest listą swoich elementów, dzięki czemu elementy pamięci HM można iterować oddzielnie po numerach wektorów i po nazwach elementów wektora w czytelniejszy sposób, a po poprawnym wygenerowaniu nowego wektora łatwiej jest go wstawić do pamięci HM. Pociągnęło to za sobą

konieczność refaktoryzacji dotychczas napisanego kodu. Dodatkowo dzięki temu rozwiązaniu ułatwiona została integracja parsera zwracającego listę zmiennych we wpisanej funkcji z samym algorytmem.

12 WYKAZ LITERATURY

[1] M. Mahdavi, M. Fesanghary i E. Demangir, „An improved harmony search algorithm for solving optimization problems,” Science Direct, Tehran, 2007.

[2] „MatPlotLib: Navigation toolbar,” 1 Maj 2020. [Online]. Available: https://matplotlib.org/3.2.1/users/navigation_toolbar.html.