Politechnika
Wrocławska

Wydział Elektroniki, Fotoniki i Mikrosystemów

Teoria i Metody Optymalizacji

Projekt

Metoda Complex

Metoda nieliniowa z ograniczeniami

Autorzy
Michał Dołharz 248943
Michał Maćkowiak 249464

Prowadzący
dr inż. Ewa Szlachcic

Termin zajęć
wt/N 9.15

Spis treści

1 Sformułowanie zadania

Zadanie polega na znalezieniu minimum nieliniowej, ciągłej funkcji z ograniczeniami za pomocą algorytmu Complex. Wykorzystano metodę minimum w kierunku dla metody Neldera i Meada bez ograniczeń (pełzający simplex). Ustalono górne ograniczenie liczby zmiennych oraz liczby ograniczeń funkcyjnych w ilości pięć każde ($m, n \leq 5$).

Program realizujący to zadanie jest napisany w języku programistycznym Python przy wykorzystaniu specjalistycznych bibliotek. Parametry wejściowe są wprowadzane przy pomocy interfejsu graficznego. Znalezienie rozwiązania zadania odbywa się poprzez kryterium stopu lub osiągnięcie jednego z ograniczeń narzucanych przez program.

2 Algorytm

2.1 Omówienie

Metoda Complex opracowana przez Boxa [3][1] jest lekko zmodyfikowaną metodą Simplex Neldera-Meada, pozwalającą na wprowadzenie do zadania ograniczeń. Istotą tego algorytmu jest utworzenie nieregularnego simplexu, czyli figury wklęsłej, o k wierzchołkach, przy czym $k > n + 1$, gdzie n to wymiar wektora zmiennych. Owa figura wpisana zostaje na powierzchnię, która reprezentuje badaną funkcję celu $f(\underline{x})$. Tak przedstawiona figura nazywana jest complexem. Metoda ta, nazywana również „pełzającym simplexem", poddawana jest przekształceniom w taki sposób, żeby odległości między wierzchołkami (boki figury) malały w trakcie przesuwania się w kierunku minimum.

W metodzie Complex przyjmuje się następującą postać ograniczeń:

$$l_i \leqslant x_i \leqslant u_i, \qquad i = 1, 2, \ldots, n, \tag{1}$$

$$l_j \leqslant g_j(\underline{x}) \leqslant u_j, \qquad j = 1, 2, \ldots m, \tag{2}$$

gdzie l_i i u_i to wartości stałe, a l_j oraz u_j są funkcjami zmiennych $\underline{x}$, przy czym ograniczenia typu (2) zwane są ograniczeniami funkcyjnymi, a ograniczenia zmiennych nazwano ograniczeniami „kostki". Dodatkowo zakładamy, że $m, n \leq 5$.

Algorytm opiera się w głównej mierze na kilku operacjach:

- operacja wyliczenia „centroidu" $\underline{c}$ zdefiniowanego jako

$$\underline{c} = \frac{\sum_{i=1}^{k} \underline{x}_i}{k-1}, \quad \text{dla } i \neq w \ , \tag{3}$$

  gdzie $\underline{x}_i$ oznacza i-ty punkt wierzchołkowy complexu, a w jest indeksem punktu wierzchołkowego $\underline{x}_w$, w którym funkcja celu $f(\underline{x})$ osiąga maksimum spośród k punktów complexu tzn. $f(\underline{x}_w) = \max$ (skrótowo nazywany „najgorszym punktem"),

- modyfikacja powyższej operacji (3), która polega na wyliczeniu środka figury składającej się z wszystkich wierzchołków, z tego powodu została roboczo nazwana „centrum"

$$\underline{c}_k = \frac{\sum_{i=1}^{k} \underline{x}_i}{k}, \tag{4}$$

  gdzie $\underline{x}_i$ oznacza i-ty punkt wierzchołkowy complexu, a w jest indeksem punktu wierzchołkowego $\underline{x}_w$, w którym funkcja celu $f(\underline{x})$ osiąga maksimum spośród k punktów complexu tzn. $f(x_w) = \max$,

- operacja „odbicia” (ang. reflect) punktu $\underline{x}_w$ względem $\underline{c}$ określona przez

$$\underline{x}^* = (1+\alpha)\underline{c} - \alpha\underline{x}_w \tag{5}$$

  gdzie $\alpha > 1$ (przyjęto $\alpha = 1,3$),

- operacja przesunięcia punktu $\underline{x}_i$ o połowę odległości w kierunku innego punktu $\underline{x}_d$, zazwyczaj centroidu lub centrum (ang. contract)

$$\underline{x}_i = \underline{x}_d - \frac{(\underline{x}_d - \underline{x}_i)}{2}, \tag{6}$$

- generowanie punktu $\underline{x}_i$ z wykorzystaniem generatora liczb pseudo-losowych r_i

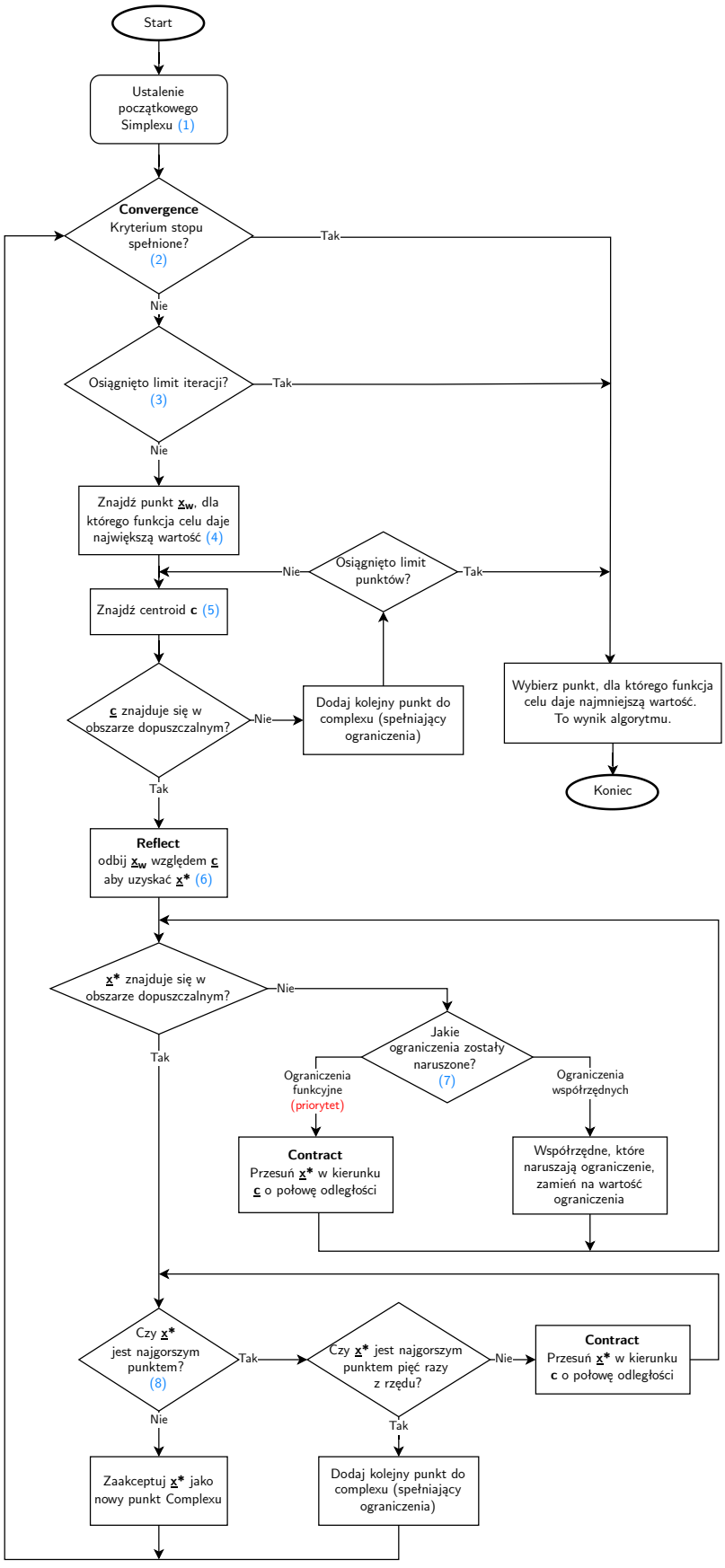
$$\underline{x}_i = u_i + r_i \left| u_i - l_i \right|, \tag{7}$$

  gdzie r_i są liczbami pseudolosowymi wygenerowanymi ze zbioru liczb przypadkowych w przedziale $[0, 1]$ oraz l_i, u_i to stałe ograniczające zmienne (ogr. kostki),

- sprawdzenie, czy kryterium stopu zostało osiągnięte (ang. convergence). Algorytm posiada trzy kryteria stopu, przy czym pierwsze jest główne, a dwa pozostałe wynikają z implementacji algorytmu:

  1. $d \leq \varepsilon$, gdzie d to długość najdłuższego boku wielokąta oraz $\varepsilon \leq 10^{-3}$,
  2. L – maksymalna liczba iteracji,
  3. K – maksymalna liczba punktów complexu.

Schemat blokowy zaprezentowany na rysunku 2.1 przedstawia sposób działania zaimplementowanego algorytmu. Opierając się na przedstawionym schemacie blokowym, opisane zostanie działanie algorytmu krok po kroku wraz z wszelkimi modyfikacjami i założeniami.



Rysunek 2.1: Schemat blokowy algorytmu Complex. Niebieskim kolorem zaznaczono poszczególne kroki z materiału źródłowego [3]

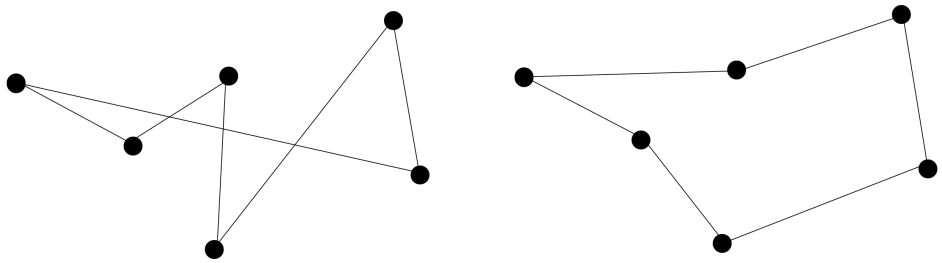
2.2 Zasada działania algorytmu

Zgodnie ze schematem blokowym (rys. 2.1).

(1) W pierwszym kroku algorytmu należy skonstruować początkowy complex. W tym celu należy uzyskać k punktów wierzchołkowych figury, które znajdują się w obszarze dopuszczalnym. Zakładamy, że w momencie uruchamiania algorytmu $k = n + 2$. Pierwszy punkt jest losowany w obszarze ograniczonym (1) i akceptowany, jeżeli spełnia warunki (2). Następne punkty generowane są w myśl zasady przedstawionej wcześniej (7). Jeżeli w ten sposób wygenerowany punkt $\underline{x}_i$ nie spełnia ograniczeń funkcyjnych (2), wówczas przesuwamy ten punkt w kierunku centrum zaakceptowanych już punktów wierzchołkowych o połowę odległości od tego centrum (4). Procedurę powtarzamy, aż warunki (2) zostaną spełnione.

(2) W tym kroku 2 oraz 3 sprawdzamy czy kryteria stopu zostały spełnione. Na początku obliczamy długość najdłuższego boku complexu $d_{\max}$, a następnie sprawdzamy czy $d_{\max} \leq \boldsymbol{\varepsilon}$ (domyślnie $\varepsilon = 0,001$). Jeśli tak, to działanie procedury kończy się i zwracany jest punk, dla którego funkcja celu daje najmniejszą wartość. W przeciwnym razie przechodzimy dalej.

Complex na płaszczyźnie jest figurą niewypukłą. Z tego względu, by móc obliczyć długość najdłuższego boku należało tak uszeregować punkty $\underline{x}_i$, żeby były one odpowiednio połączone, tzn. aby linie łączące punkty się nie krzyżowały. Poglądowy rysunek obrazujący problem przedstawiony został na rysunku 2.2. Odpowiednie uszeregowanie uzyskano poprzez wykorzystanie współrzędnych biegunowych. Każdy punkt complexu został przeliczony na współrzędne biegunowe względem centrum $\underline{c}_k$, który był punktem referencyjnym $(0,\ 0)$. Następnie zostały one posortowane ze względu na współrzędne: najpierw względem kąta, następnie względem promienia. W wyniku otrzymano odpowiednio uszeregowane punkty tworzące wielokąt.



Rysunek 2.2: Przykład niepoprawnego i poprawnego połączenia wierzchołków figury

(3) Jeżeli pierwsze kryterium stopu nie zostało osiągnięte, to należy teraz sprawdzić drugie, czyli czy została przekroczona liczba iteracji L. Wynik tego kroku jest analogiczny do kroku 2.

(4) Jeżeli kryteria stopu nie zakończyły działania algorytmu to zaczyna się esencja algorytmu complex. Wyznaczamy spośród k punktów complexu punkt $\underline{x}_w$, w którym funkcja celu osiąga wartość maksymalną tzn. $f(x_w) = \max$,

(5) następnie obliczamy centroid complexu $\underline{c}$ według (3). W ten sposób otrzymujemy środek figury bez najgorszego punktu, tzn. o największej wartości funkcji celu. Teraz

badamy, czy wyliczony punkt centroidu $\underline{c}$ znajduje się w obszarze dopuszczalnym 2. Jeśli tak, to przechodzimy do następnego kroku.

Może jednak zdarzyć się, że centroid wypada poza obszarem dopuszczalnym. Literatura wskazuje, aby dodać kolejny punkt do complexu, podstawiając $k = k+1$. Nie jest to rozwiązanie niezawodne. W niektórych przypadkach dodatkowe punkty mają za mały wpływ na położenie centroidu, aby „wyciągnąć" go ze strefy niedopuszczalnej. Algorytm tworzy wtedy coraz więcej dodatkowych punktów, a ze względu na ich liczbę, każdy kolejny ma coraz mniejszy wpływ na centroid. Z tego powodu powstało ograniczenia liczby punktów complexu do 100. W momencie przekroczenia tej liczby program kończy wykonywanie algorytmu informując użytkownika o zaistniałej sytuacji nieudanej próby przeniesienia centroidu. Wynik jest zwracany z informacją, że może nie być prawdziwym rozwiązaniem.

(6) W tym kroku wykonywana jest operacja, która pozwala complexowi się przemieszczać (pełzać). Wykonujemy odbicie punktu $\underline{x}_w$ względem $\underline{c}$ w myśl (5), aby uzyskać punkt odbity $\underline{x}^*$. Teraz należy zbadać, czy odbity punkt $\underline{x}^*$ spełnia ograniczenia (1) oraz (2). Jeśli tak, to przechodzimy do wykonania kroku 8, natomiast w przeciwnym razie

(7) należy zbadać które ograniczenia zostały naruszone:

- ograniczenia kostki (1) – wtedy na miejsce $\underline{x}^*$ podstawiamy wartość liczbową naruszanego ograniczenia.
- ograniczenia funkcyjne (2) – wtedy przesuwamy punkt $\underline{x}^*$ w kierunku centrum o połowę odległości zgodnie ze wzorem (6), przy czym czynność tę powtarzamy dotąd, aż ograniczenia (2) zostaną spełnione.
- oba ograniczenia – wtedy priorytet ma rozwiązanie pochodzące od ograniczenia funkcyjne (2).

(8) Ostatnim krokiem jest zbadanie czy w znalezionym punkcie $\underline{x}^*$ funkcja celu osiąga w dalszym ciągu wartość maksymalną spośród pozostałych punktów complexu, czyli czy punkt dalej jest najgorszy. Jeśli nie, to akceptujemy ten punkt i podstawiamy $\underline{x}^*$ w miejsce $\underline{x}_w$ i rozpoczynamy wykonywanie procedury od kroku 2.

Natomiast jeśli tak, to wykonujemy operację *contract* (6) i powtarzamy krok 8. Literatura wskazuje, aby czynić to tak długo, aż punkt przestanie zwracać największą wartość funkcji celu. Rozwiązanie to nie gwarantuje poprawnego działania. W wielu przypadkach trafia się na takie miejsce, gdzie przybliżanie (6) nie minimalizuje wystarczająco wartości funkcji celu dla tego punktu i program się zapętla, ponieważ funkcja celu nadal zwraca największą wartość dla tego punktu. Zaproponowanym rozwiązaniem jest ograniczenie liczby operacji *contract*. Przyjęta liczba pięciu wykonań daje rezultat zbliżenia punktów do 3,125% początkowej odległości. Jeżeli po tych pięciu operacjach punkt $\underline{x}^*$ nadal będzie najgorszy, to dodawany jest następny punkt do complexu, analogicznie jak w kroku 5.

W momencie skończenia działania programu zwraca on punkt $\underline{x}_b$, w którym funkcja celu daje najmniejszą wartość, $f(\underline{x}_b) = \min$.

3 Środowisko programistyczne

Projekt został napisany w całości w języku programowania Python.

Wykorzystywane biblioteki w projekcie to:

- NumPy – biblioteka ta ułatwia i polepsza dokładność obliczeń numerycznych, które były głównym celem projektu. Biblioteka ta udostępnia własne tabele oraz przydatne metody jak generator liczba pseudo-losowych, obliczenie numeryczne pierwiastka oraz funkcje trygonometryczne np. arcus tangens,
- MatPlotLib – biblioteka ta umożliwiła rysowanie wykresów prawie tak wygodnie jak w środowisku Matlab. Metody zawarte w tej bibliotece pozwoliły na rysowanie punktów (funkcja *scatter*), wykresów ciągłych (funkcja *plot*) oraz wyrysowanie warstwicy funkci dwóch zmiennych przy użyciu funkcji *contourf*,
- SymPy – biblioteka ta pozwala wykonywać obliczenia symboliczne w odróżnieniu od biblioteki NumPy. Symboliczne obliczenia okazały się bardzo przydatne w momencie rysowania nierówności funkcji oraz rozwiązywania równań względem zadanej zmiennej,
- PySimpleGui – biblioteka ta jest prosta i szybka w użyciu, ale zarazem kompletna i rozbudowana. Pozwala na tworzenie aplikacji okienkowych, które zawierają takie elementy jak pole tekstowe, przyciski, listy, suwak itp. oraz umożliwia ich edycję i dopasowanie wraz z obsługą zdarzeń. Twórcy udostępniają liczne programy demonstracyjne, które ułatwiły pracę z biblioteką.

Struktura programu podzielona jest łącznie na 5 plików:

- point.py – zawiera definicję klasy Point, która posiada takie elementy jak współrzędne oraz id, które jego liczbę i jest unikatowe dla każdego punktu. Posiada również bardzo podstawowe metody. Współrzędne punktu zapisywane w postaci tablicy wszystkich współrzędnych.
- complex.py – zawiera klasę Complex, która zawiera w sobie wszystkie punkty complexu zapisane są w tablicy klas Point. Posiada również inne elementy, jak kryterium stopu, informacje o complexie w postaci liczby punktów oraz liczby zmiennych. Metody zawarte w tej klasie pozwalają wykonywać operacje i przekształcenia na complexie. Cały algorytm jest wykonywany przy pomocy metod tej klasy.
- interface.py – zawiera całą konstrukcję interfejsu użytkownika, to tam każdy element aplikacji ma swoją definicję.
- funcParser.py – zawiera funkcję dzięki, której możliwe jest parsowanie wpisanych funkcji w aplikacji do programu w postaci wykonywalnej.
- main.py – zawiera całą obsługę zdarzeń związaną z aplikacją okienkową.

Funkcja parsowania funkcji zawarta w pliku funcParser.py polega na sprawdzaniu wprowadzanej funkcji w postaci String znak po znaku i sprawdzaniu czy element tam zawarty jest dopuszczalny. Jeżeli znaleziono niedozwoloną operację program informuje błędem. W momencie prześledzenia całego wprowadzonego tekstu zamieniane są sekwencje znaków na sekwencje znaków, które są rozpoznawalne przez program jako funkcje np.sin -> $np.sin$,

co oznacza metodę biblioteki NumPy. Następnie przy pomocy metody *eval()* sparsowanemu tekstowi można przyporządkować odpowiednie wartości dla każdej zmiennej tam występującej.

Z uwagi na charakter zapisywania punktów oraz całego complexu wykonywanie operacji polegało na wykorzystaniu wielu pętli, by wykonać operacje na poszczególnych operacjach odpowiednich punktów.

Wykreślanie tylu elementów (warstwica, complex, centroid, funkcje ograniczeń) wymagało tworzenia wykresów w odpowiedniej kolejności w odpowiedni sposób. Do rysowania funkcji ograniczeń wykorzystano bibliotekę SymPy, która udostępnia możliwość rysowania wykresów. Niestety nie było możliwe wyrysowanie warstwicy przy pomocy biblioteki SymPy, a bezpośrednie łączenie wykresów dwóch różnych bibliotek nie było możliwe. Wykorzystano dlatego możliwości przeniesienia infrastruktury wykresu SymPy na taką rozumianą przez bibliotekę MatPlotLib. Dzięki temu możliwe było połączenie wszystkich wykresów w jedną całość i wyrysować wykres w aplikacji.

Zaimplementowano również suwak, który umożliwia sprawdzenie pozycji complexu w każdej iteracji programu. Niestety z uwagi na dużą ilość elementów na wykresie program potrzebuje około 1 sekundę na wyświetlenie wykresu.

Poniżej na rysunku 3.1 przedstawiony jest wygląd aplikacji zaraz po uruchomieniu.

Rysunek 3.1: Aplikacja po uruchomieniu

4 Dane początkowe i wprowadzanie danych

Program po uruchomieniu posiada domyślnie wprowadzone wielkości w okna wejściowe. Możliwa jest ich pełna edycja.

Ograniczenia kostki oraz ograniczenia funkcyjne wprowadzamy i zatwierdzamy przyciskiem „Dodaj”, a usuwamy zaznaczone ograniczenie przyciskiem „Usuń”.

Funkcję celu można wprowadzić ręcznie albo wybrać jedną z trzech wpisanych domyślnie do programu (są to funkcje, na których przeprowadzono testy).

Wszystkie parametry oprócz ograniczeń oraz ramki „Ustawienia wykresu" zatwierdzane są w momencie uruchomienia programu. Ich edycja wiąże się z koniecznością ponownego uruchomienia algorytmu.

Wartości w ramce „Ustawienia wykresu" nie są obligatoryjne do wprowadzenia. Jeżeli pozostaną puste w momencie uruchomienia algorytmu, to automatycznie zostaną uzupełnione wartościami ograniczeń kostki. Po ich edytowaniu nie nie jest konieczne ponowne uruchamianie algorytmu. W takim wypadku można skorzystać z przycisku „Odśwież", który przerysowuje wykres aktualnego wyniku na ograniczenia osi wpisane w ramkę.

Domyśle wartości wprowadzone w aplikacje to:

- $\varepsilon = 0,001$,
- Maksymalna liczba iteracji $= 5000$,
- dolne i górne ograniczenie kostki o wartości kolejno -5 i 5,
- lista predefiniowanych funkcji celu.

5 Testy programu

5.1 Funkcja nieliniowa dwóch zmiennych

Przedmiotem testów jest nieliniowa funkcja dwóch zmiennych

$$f(x) = (x_1 - 2)^2 + (x_1 - x_2^2)^2$$

oraz nieliniowe ograniczenia funkcyjne

$$\begin{aligned} x_2^2 + 1 - x_1 \leq & 0 \\ (x_1 - 1)^2 - x_2 \leq & 0 \end{aligned}$$

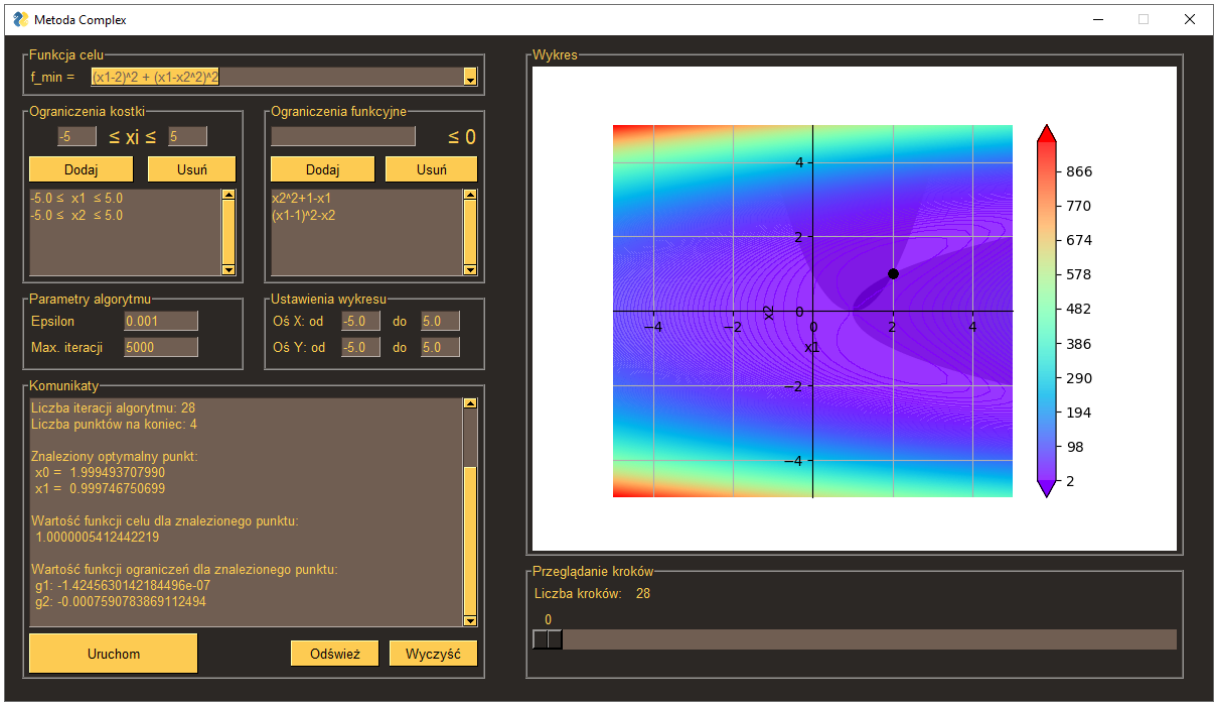
Ponadto każda zmienna jest ograniczona od dołu oraz od góry poprzez tzw. ograniczenia kostki.

Funkcja celu oraz jej fragment wynikający z ograniczeń funkcyjnych są przedstawione w widoku 3D pod adresem https://www.geogebra.org/m/uwvvqxqt. Korzystając z kolorowych oznaczeń po lewej stronie, możliwe jest włączanie i wyłączanie widoku poszczególnych wykresów.

Rysunek 5.1 przedstawia przykładowe rozwiązanie zwrócone przez program. Tabela 1 zawiera porównanie tego samego rozwiązania z oczekiwanym wynikiem. Rezultat jest zadowalający. Znaleziony punkt jest określony z dużą precyzją, a algorytm potrzebował jedynie 28 iteracji, aby go uzyskać i nie potrzebował dodać żadnego kolejnego punktu do complexu.

Tabela 1: Porównanie wyników

	Oczekiwany wynik	**Przykładowy wynik**
x_1	2	1,999493707990
x_2	1	0,999746750699
$f(x)$	1	1,0000005412442219
$g_1(x)$	0,000	-1,4245630142184496e-07
$g_2(x)$	0,000	-0,0007590783869112494



Rysunek 5.1: Wynik algorytmu funkcji dwóch zmiennych

5.2 Funkcja nieliniowa dwóch zmiennych typu butelka

Przedmiotem testów jest nieliniowa funkcja dwóch zmiennych typu butelka

$$f(x) = x_1^4 + x_2^4 - x_1^2 - x_2^2$$

oraz dwa nieliniowe ograniczenia funkcyjne rozważane osobno

$$x_1^2 + x_2^2 - 0,3 \leq 0 \tag{8}$$

$$(x_1 + 2)^2 + (x_2 - 1)^2 - 1 \leq 0 \tag{9}$$

Funkcja celu oraz jej fragmenty wynikające z obu ograniczeń funkcyjnych są przedstawione w widoku 3D pod adresem https://www.geogebra.org/m/tjdaqybm

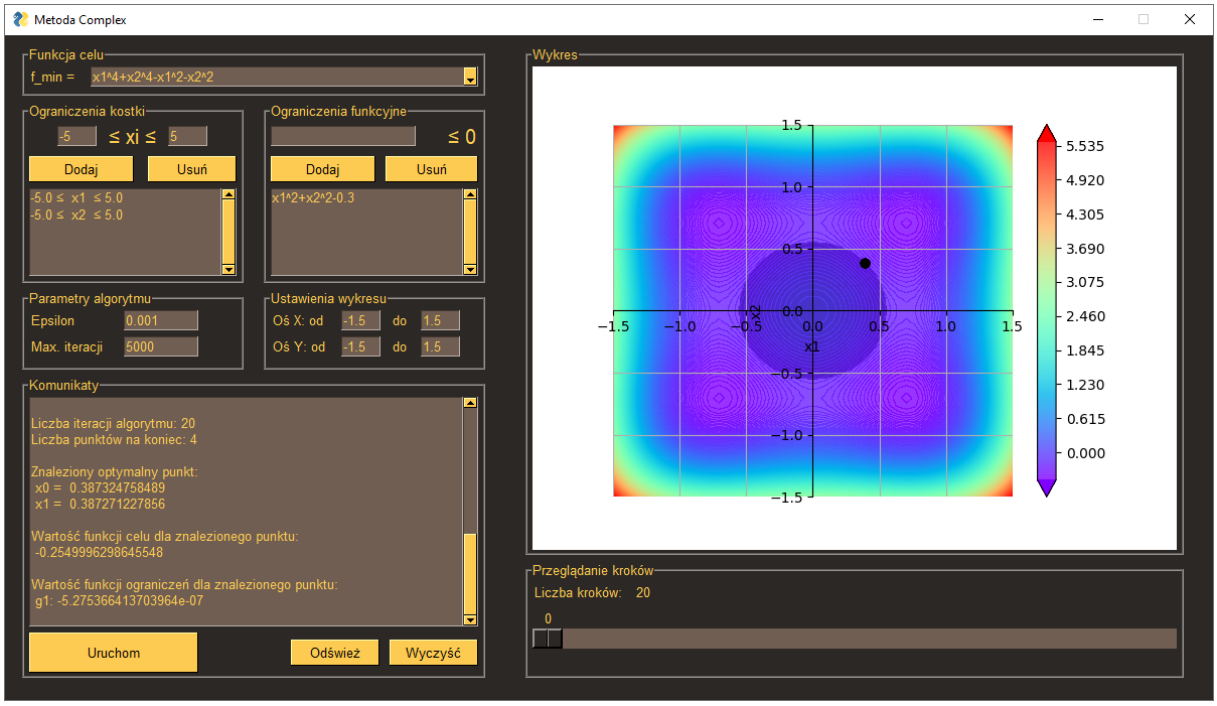
Rysunek 5.2 przedstawia przykładowe rozwiązanie zwrócone przez program dla funkcji z ograniczeniem (8). Tabela 2 zawiera porównanie tego samego rozwiązania z oczekiwanym

Tabela 2: Porównanie wyników przy ograniczeniach (8)

	Oczekiwany wynik	**Przykładowy wynik**
x_1	(-)0,38730	0,387324758489
x_2	(-)0,38730	0,387271227856
$f(x)$	-0,255	-0,2549996298645548
$g_1(x)$	0,000	-5,275366413703964e-07

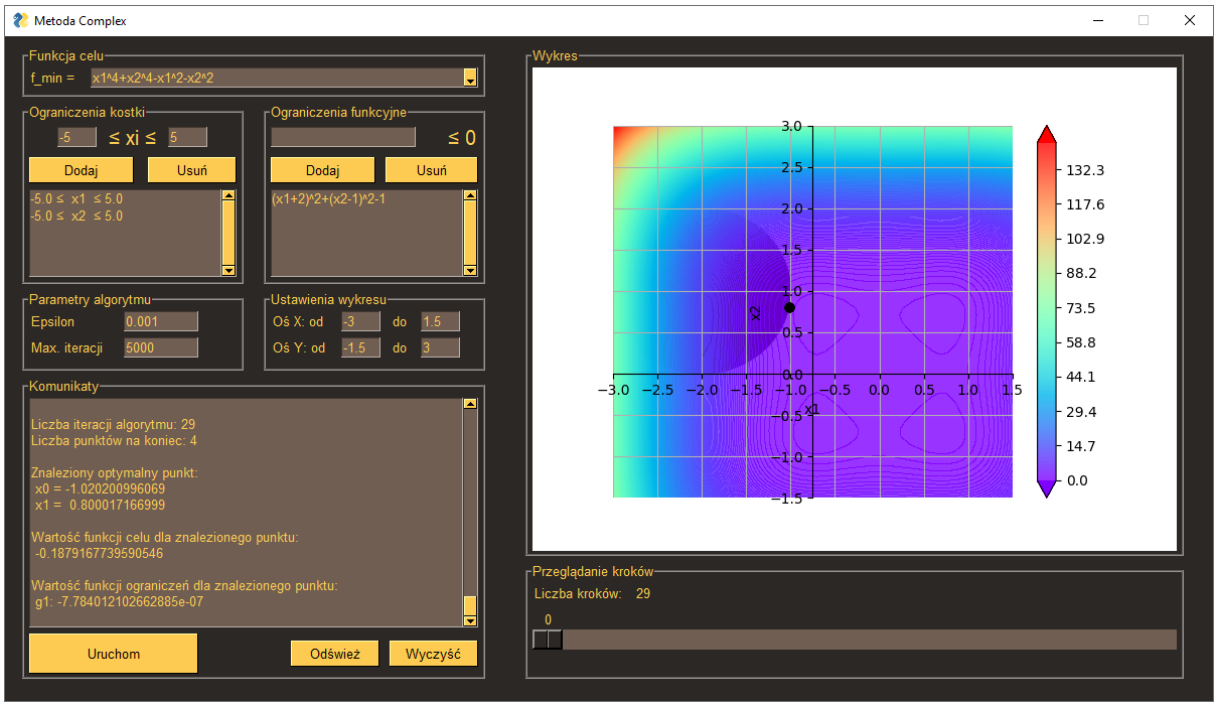
wynikiem. Nieograniczona funkcja celu posiada cztery minima rozłożone symetrycznie względem początku układu współrzędnych. Metoda Complex nie pozwala na uzyskanie wszystkich czterech przy jednym uruchomieniu algorytmu, ale do oceny efektu wystarczy tylko jeden wynik. Również w tym przypadku rezultat jest zadowalający, zarówno pod względem funkcji celu, jak i poszczególnych zmiennych. Algorytm tym razem potrzebował tylko 20 iteracji na uzyskanie takiego wyniku i żadnego dodatkowego punktu complexu.

Powodem, dla którego algorytm wyznaczył akurat to minimum, jest usytuowanie początkowego complexu w taki sposób, że to minimum miało największy wpływ na algorytm.



Rysunek 5.2: Wynik algorytmu funkcji dwóch zmiennych z ograniczeniami (8)

Rysunek 5.3 przedstawia przykładowe rozwiązanie zwrócone przez program dla funkcji z ograniczeniem (9). Tabela 3 zawiera porównanie tego samego rozwiązania z oczekiwanym wynikiem.



Rysunek 5.3: Wynik algorytmu funkcji dwóch zmiennych z ograniczeniami (9)

Tabela 3: Porównanie wyników przy ograniczeniach (9)

	Oczekiwany wynik	**Przykładowy wynik**
x_1	-1,1056	-1,020200996069
x_2	0,5528	0,800017166999
$f(x)$	-0,255	-0,1879167739590546
$g_1(x)$	0,000	-7,784012102662885e-07

5.3 Szukanie wszystkich minimum

Funkcja ta ma wiele minimów, a algorytm Complex może wyznaczyć tylko jedno naraz. W związku z tym, aby odnaleźć wszystkie minima, należy odpowiednio manewrować ograniczeniami kostki. W ten sposób można uzyskać określone przestrzenie, np. $x_1 > 0$, $x_2 < 0$.

Wyznaczone w ten sposób minima są przedstawione w tabeli 4. Wyniki są zbliżone do oczekiwanych i wskazują, że faktycznie punkty minimum różnią się jedynie znakami przy zmiennych. Z tabeli można również wywnioskować, że we wszystkich minimach funkcja celu przyjmuje tę samą wartość.

Tabela 4: Wyniki szukania wszystkich minimum funkcji dwóch zmiennych typu butelka

	Minimum	**Znalezione minimum**
x_1	0,38730	0,387324758489
x_2	0,38730	0,387271227856
$f(x)$	-0,255	-0,2549996298645548
$g_1(x)$	0,000	-5,275366413703964e-07
x_1	0,38730	0,412579824579
x_2	-0,38730	-0,360244793555
$f(x)$	-0,255	-0,2541809646683245
$g_1(x)$	0,000	-1,5770668371772878e-06
x_1	-0,38730	-0,399837366072
x_2	-0,38730	-0,374338985220
$f(x)$	-0,255	-0,25480487800794516
$g_1(x)$	0,000	-4,0483744095931584e-07
x_1	-0,38730	-0,317696253083
x_2	0,38730	0,446168216330
$f(x)$	-0,255	-0,25018263493238074
$g_1(x)$	0,000	-3,013514195460143e-06

5.4 Funkcja nieliniowa czterech zmiennych Rosena-Suzukiego

Przedmiotem testów jest nieliniowa funkcja czterech zmiennych Rosena-Suzukiego [2][4]

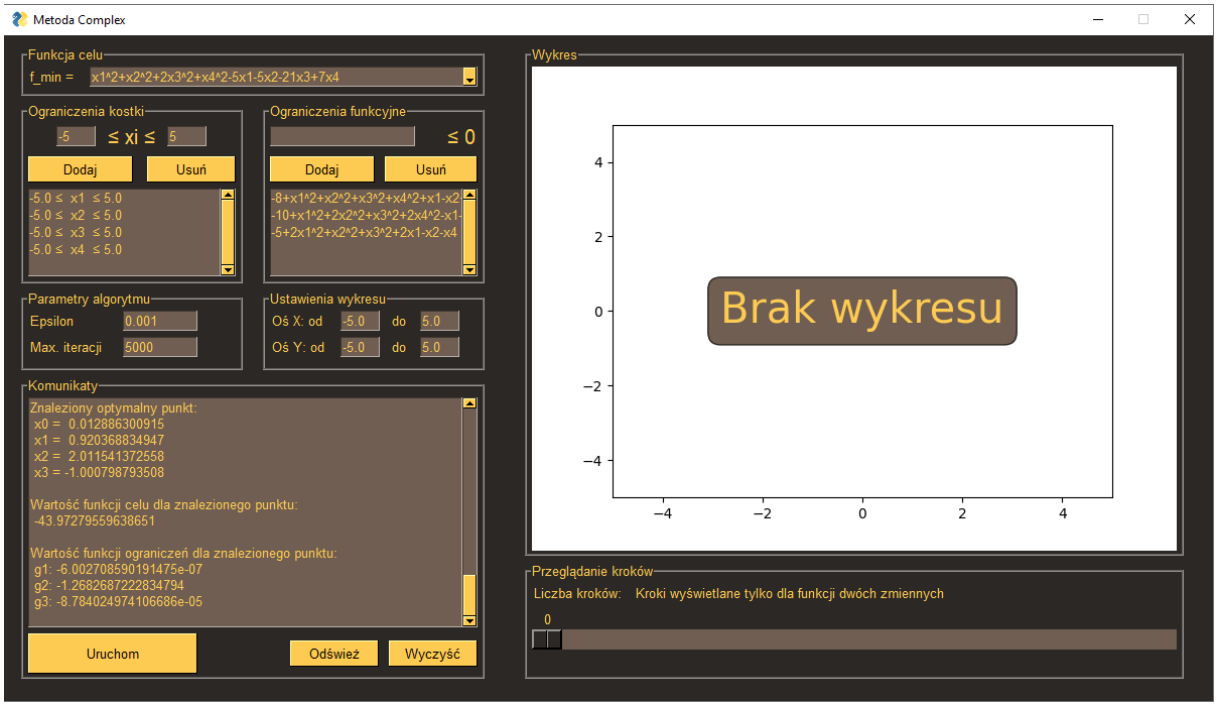
$$f(x) = x_1^2 + x_2^2 + 2x_3^2 + x_4^2 - 5x_1 - 5x_2 - 21x_3 + 7x_4$$

oraz trzy nieliniowe ograniczenia

$$\begin{aligned} -8 + x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_1 - x_2 + x_3 - x_4 &\leq 0 \\ -10 + x_1^2 + 2x_2^2 + x_3^2 + 2x_4^2 - x_1 - x_4 &\leq 0 \\ -5 + 2x_1^2 + x_2^2 + x_3^2 + 2x_1 - x_2 - x_4 &\leq 0 \end{aligned} \tag{10}$$

Rysunek 5.4 przedstawia przykładowe rozwiązanie zwrócone przez program dla funkcji z ograniczeniami (10). Tabela 5 zawiera porównanie tego samego rozwiązania z oczekiwanym wynikiem. Biorąc pod uwagę wymiary funkcji, wyniki wyszły bardzo dobre. Ze względu na rozmiary okna programu nie widać niestety liczby iteracji ani liczby punktów complexu na koniec. Algorytm tym razem skończył z sześcioma punktami, czyli żadnego nie dodawał, oraz potrzebował 243 iteracje, co wydaje się być dużą liczbą, jednak w trakcie pracy nad programem występowały również przypadku dużo większych liczb, sięgających nawet limitów liczby iteracji czy punktów complexu. Czas wykonywania algorytmu był wtedy znacznie dłuższy, a wyniki nie były wiele dokładniejsze. Zapotrzebowanie na dokładność wyników zależy oczywiście od konkretnego przypadku i do tego służy parametr kryterium stopu, ε

Należy zaznaczyć, że w przypadku funkcji Rosena-Suzukiego rozrzut wyników był dużo większe, niż w przypadku dwóch poprzednich funkcji, gdzie wyniki były praktycznie zawsze mocno zbliżone. W tym przypadku zdarzało się, że funkcja celu przyjmowała wartości nawet większe od zera, co przy oczekiwanej warto -44 robi dużą różnicę. Z oczywistych względów nie jesteśmy w stanie wykreślić funkcji, jednak można wysnuć wniosek, że taka rozbieżność wyników jest spowodowana istnieniem innych minimów lokalnych poza tym o wartości -44.



Rysunek 5.4: Wynik algorytmu funkcji Rosena-Suzukiego

Tabela 5: Porównanie wyników

	Oczekiwany wynik	**Przykładowy wynik**
x_1	0	0,012886300915
x_2	1	0,920368834947
x_3	2	2,011541372558
x_4	-1	-1,000798793508
$f(x)$	-44	-43,97279559638651
$g_1(x)$	0,000	-6,002708590191475e-07
$g_2(x)$	-1,000	-1,2682687222834794
$g_3(x)$	0,000	-8,784024974106686e-05

6 Wpływ parametru ε

W tym rozdziale zbadano wpływ parametru ε na wyniki algorytmu. Wartość tego parametry określa pierwsze kryterium stopu metody complex.

6.1 Funkcja nieliniowa dwóch zmiennych

Poniżej w tabelach 6, 7 przedstawiono wyniki dla przykładowej mniejszej i większej wartości ε niż wcześniej rozważanej $\varepsilon = 0,001$ w rozdziale 5.

Tabela 6: Wyniki algorytmu dla $\varepsilon = 0,1$ po 10 iteracjach i dla 4 punktów complexu

	Oczekiwany wynik	**Przykładowy wynik**
x_1	2	1,971504784556
x_2	1	0,984884309938
$f(x)$	1	1,0038296115930172
$g_1(x)$	0,000	-0,0015076805945342109
$g_2(x)$	0,000	-0,041062763521838797

Tabela 7: Wyniki algorytmu dla $\varepsilon = 0,00000001$ po 120 iteracjach i dla 4 punktów complexu

	Oczekiwany wynik	**Przykładowy wynik**
x_1	2	1,999999968501
x_2	1	0,999999984251
$f(x)$	1	1,000000000000013
$g_1(x)$	0,000	-2,220446049250313e-16
$g_2(x)$	0,000	-4,724823876589568e-08

6.2 Funkcja nieliniowa dwóch zmiennych typu butelka

Poniżej w tabelach 8, 9 przedstawiono wyniki dla przykładowej mniejszej i większej wartości ε niż wcześniej rozważanej $\varepsilon = 0,001$ w rozdziale 5.

6.3 Ograniczenia (8)

Tabela 8: Wyniki algorytmu dla $\varepsilon = 0,1$ po 6 iteracjach i dla 4 punktów complexu

	Oczekiwany wynik	**Przykładowy wynik**
x_1	(-)0,38730	0,292743930774
x_2	(-)0,38730	0,456717188835
$f(x)$	-0,255	-0,2434352449605042
$g_1(x)$	0,000	-0,005710400417574812

Tabela 9: Wyniki algorytmu dla $\varepsilon = 0,00000001$ po 77 iteracjach i dla 4 punktów complexu

	Oczekiwany wynik	**Przykładowy wynik**
x_1	(-)0,38730	0,387298339879
x_2	(-)0,38730	0,387298339363
$f(x)$	-0,255	-0,2549999999999995
$g_1(x)$	0,000	0,0

6.4 Ograniczenia (9)

Poniżej w tabelach 10, 11 przedstawiono wyniki dla przykładowej mniejszej i większej wartości ε niż wcześniej rozważanej $\varepsilon = 0,001$ w rozdziale 5.

Tabela 10: Wyniki algorytmu dla $\varepsilon = 0,1$ po 5 iteracjach i dla 4 punktów complexu

	Oczekiwany wynik	**Przykładowy wynik**
x_1	-1,1056	-1,851971187849
x_2	0,5528	1,926918401374
$f(x)$	-0,255	18,407174446973496
$g_1(x)$	0,000	-0,11890974796692699

Tabela 11: Wyniki algorytmu dla $\varepsilon = 0,00000001$ po 92 iteracjach i dla 4 punktów complexu

	Oczekiwany wynik	**Przykładowy wynik**
x_1	-1,1056	-1,020143952385
x_2	0,5528	0.800294902538
$f(x)$	-0,255	-0,1879179771215589
$g_1(x)$	0,000	-1,1102230246251565e-16

6.5 Funkcja nieliniowa czterech zmiennych Rosena-Suzukiego

Poniżej w tabelach 12, 13 przedstawiono wyniki dla przykładowej mniejszej i większej wartości ε niż wcześniej rozważanej $\varepsilon = 0,001$ w rozdziale 5.

Tabela 12: Wyniki algorytmu dla $\varepsilon = 0,1$ po 41 iteracjach i dla 6 punktów complexu

	Oczekiwany wynik	**Przykładowy wynik**
x_1	0	-0,212583245898
x_2	1	1,042990220776
x_3	2	2,293937980594
x_4	-1	-0,021588448394
$f(x)$	-44	-40,81806238981897
$g_1(x)$	0,000	-0,5444092807009355
$g_2(x)$	-1,000	-2,2818958869848327
$g_3(x)$	0,000	-0,006204931860110535

Tabela 13: Wyniki algorytmu dla $\varepsilon = 0,0000001$ po 360 iteracjach i dla 6 punktów complexu

	Oczekiwany wynik	**Przykładowy wynik**
x_1	0	0,014455317524
x_2	1	0,949015451388
x_3	2	2,003351278757
x_4	-1	-1,005640230859
$f(x)$	-44	-43,988228065178
$g_1(x)$	0,000	-7,210502541710184e-07
$g_2(x)$	-1,000	-1,1713045825725903
$g_3(x)$	0,000	-8,881784197001252e-16

Jak można zaobserwować w tabelach 6, 8, 10, 12 zwiększenie wartości parametru ε powoduje zmniejszenie dokładności uzyskiwanych wyników. W niektórych wynikach odstępstwo jest na tyle duże, że punkty są bardzo daleko od szukanych minimum. Dodatkowo znacznie zmniejsza się ilość iteracji. Wynika to z tego, że znacznie szybciej długość najdłuższego boku osiąga wartość $d_{max} \leq \varepsilon$, czyli kryterium stopu.

Zmniejszenie wartości ε kilka rzędów wartości zwiększa dokładność wyników, co zostało przedstawione w tabelach 7, 9, 11, 13. Tak dokładne wyniki wymagają jednak znacząco więcej iteracji, które potrzebuje complex by zmniejszyć się do takiej skali.

7 Podsumowanie

Metoda Complex wydaje się być prosta. Faktycznie, główna idea jest nieskomplikowana, a zasada działania łatwa do przyswojenia. W praktyce okazuje się, że za tą prostotą kryje się niekompletność metody. Implementacja metody ściśle według [3] byłaby bardzo zawodna, program bardzo często by się zapętlał i wymagał ponownego uruchomienia. Implementacja metody wymagała od nas załatania wszystkich „dziur" i szukania własnych rozwiązań. Nasze usprawnienia również nie są absolutnie niezawodne, jednak zdecydowanie usprawniają algorytm.

Należy wspomnieć, że w opisie algorytmu w [3] jest błąd w formule na odbicie punktu względem centroidu. Sprawdzając wzór tradycyjnie na kartce papieru, okazywało się, że wzór był zwyczajnie niepoprawny i w wyniku nie uzyskiwało się odbitego punktu.

Wyniki dowodzą, że metoda działa poprawnie. W zależności od żądanej dokładności, zadawanej poprzez kryterium stopu, można uzyskać większą lub mniejszą dokładność wyników, odpowiednio poświęcając lub zyskując czas. Wadą naszego programu jest mało efektywne wyświetlanie kolejnych kroków algorytmu na wykresie. Inną wadą jest swego rodzaju kompromis wyświetlania jednocześnie półprzezroczystych warstwic i zakresów wyznaczanych przez ograniczenia, ponieważ w niektórych sytuacjach efekt jest bardzo niezadowalający i niewiele pomaga w odczytaniu wykresu. Są to jednak problemy związane z samym kreśleniem funkcji, a nie z działaniem algorytmu, który to był głównym celem niniejszego projektu.

References

[1] M. Chen. *Stats 102A Lesson 8-2 Nelder Mead Method / Algorithm.* Dostępny w Internecie: https://www.youtube.com/watch?v=vOYlVvT3W80&t=793s.

[2] SAS Institute Inc. *SAS/IML User's Guide, Version 8,* Cary, NC: SAS Institute Inc., 1999. 846 pp.

[3] W. Findeisen, J. Szymanowski, A. Wierzbick. *Metody obliczeniowe optymalizacji.* Wydawnictwa Politechniki Warszawskiej. 1972.

[4] W. Hock, K. Schittkowski. *Test Examples for Nonlinear Programming Codes.* Springer, Lecture Notes in Economics and Mathematical Systems, Vol. 187.