Jakub Firlej,
Michał Kolankiewicz

POP - Dokumentacja końcowa

1. Wstęp i cel projektu (przypomnienie):

Zależnie od wybranej liczby populacji algorytmy ewolucyjne mogą zachowywać się inaczej, np.:

- mała liczba populacji może skutkować mniejszą precyzją typowego algorytmu ewolucyjnego i większą ruchliwością, tzn. większym prawdopodobieństwem na odkrycie lepszego rozwiązania (przekroczenie optimum lokalnego),
- duża liczba populacji analogicznie może skutkować większą precyzją rozwiązania kosztem mniejszego prawdopodobieństwa na znalezienie optimum globalnego.

Celem projektu jest wdrożenie dynamicznej (w trakcie działania) zmienności populacji do algorytmu ewolucyjnego i zbadanie korzyści z takiego zabiegu dla różnych metod tej zmienności.

2. Opis realizacji projektu:

W projekcie zaimplementowano mutacyjny algorytm ewolucyjny przedstawiony na wykładzie prof. Arabasa w celu minimalizacji funkcji celu.

- Jako funkcję selekcji wybrano metodę turniejową ze stałą wielkością szranki, a jako funkcję mutacji - mutację gaussowską ze stałym parametrem sigma.
- Jako budżet każdego eksperymentu, czyli maksymalną dopuszczalną liczbę wywołań funkcji celu, została przyjęta wartość **20000**. Jeżeli w danej iteracji liczba ta miałaby zostać przekroczona, aktualna populacja zostaje odrzucona, a przebieg algorytmu zakończony.
- Eksperymenty zostały przeprowadzone dla funkcji benchmarkowych *f4* i *f7* z zestawu *CEC 2017*, dla których zbadano działanie algorytmu w 2, 10 i 20 wymiarach. Dodatkowo sprawdzono działanie algorytmu dla dwuwymiarowej funkcji Ackley'a.
- W przypadku wykroczenia punktu roboczego poza przestrzeń przeszukiwań, punkty będą rzutowane na tą przestrzeń.

Uwaga! Należy zwrócić uwagę na to, że w poszczególnych iteracjach algorytmu wykorzystanie budżetu będzie różne ze względu na zmienność populacji, podczas gdy funkcja zmienności liczby potomków polega nie tylko na **aktualnej** wartości *t*, ale w niektórych przypadkach również na **maksymalnej liczbie** iteracji *T_MAX*. Jest to problem, ponieważ, w przeciwieństwie do maksymalnego budżetu *Q_MAX,* maksymalna wartość iteracji *T_MAX* nie jest odgórnie znana i należy ją obliczyć, znając zależność:

$$\int_{0}^{T\_MAX?} pop\_f(t,\ T\_MAX?)dt \ = \ Q\_MAX$$

Dlatego zastosowano numeryczne rozwiązanie zadania określenia szukanej wartości *T_MAX* na podstawie danej funkcji zmienności populacji *pop_f* oraz budżetu funkcji celu *Q_MAX*.

Stałe parametry wszystkich eksperymentów:

Przestrzeń przeszukiwań	$[-100, 100]^D$
Minimalna wielkość populacji (*POP_MIN)*	1
Maksymalna wielkość populacji (*POP_MAX)*	40
Wielkość populacji w wersji ze stałą populacją	5
Budżet każdego eksperymentu	20000
Sigma rozkładu Gaussa	1
Wielkość szranki selekcji turniejowej	5
Punkt startowy	$\{50\}^D$

W ramach eksperymentów stworzono 7 funkcji zmiany populacji:

- liniowy wzrost i spadek
- wykładniczy wzrost i spadek,
- okresowa zmienność sinusoidalna,
- okresowa zmienność prostokątna,
- zmiana populacji przy zbyt długiej stałości najlepszego znalezionego punktu.

Wspomniane wcześniej obliczenie maksymalnej liczby iteracji jest potrzebne pod kątem funkcji liniowych i wykładniczych. Chcemy, aby na całej długości eksperymentu populacja zmieniła się z wartości minimalnej na maksymalną lub vice versa, co wymaga odgórnej znajomości iteracji, które wykona algorytm przy założonym budżecie. Znając skrajne wartości populacji i ilość iteracji metody te odpowiednio fitują funkcje i obliczają wielkość populacji dla danej iteracji.

Warto również wspomnieć o zasadzie działania funkcji zmiany przy stagnacji:

- Funkcja skokowo zmienia wielkość populacji pomiędzy ***POP_MIN*** i ***POP_MAX***, jeśli najlepsza wartość funkcji celu ***q_best*** w danej iteracji ***k*** różni się o mniej niż ***q_tol*** niż średnia najlepszych wartości funkcji celu ostatnich ***q_keep*** iteracji algorytmu od ostatniej zmiany populacji (lub od początku algorytmu). To oznacza, że przy zajściu warunku:

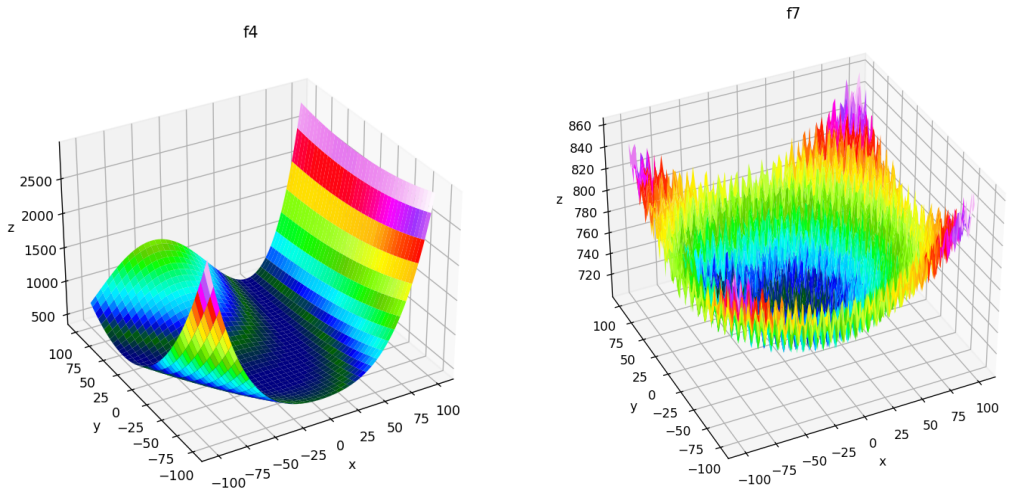
$$\left| q\_best_k - \left(\frac{\sum_{i=1}^{q\_keep} q\_best_{k-i}}{q\_keep} \right) \right| \leq q\_tol$$

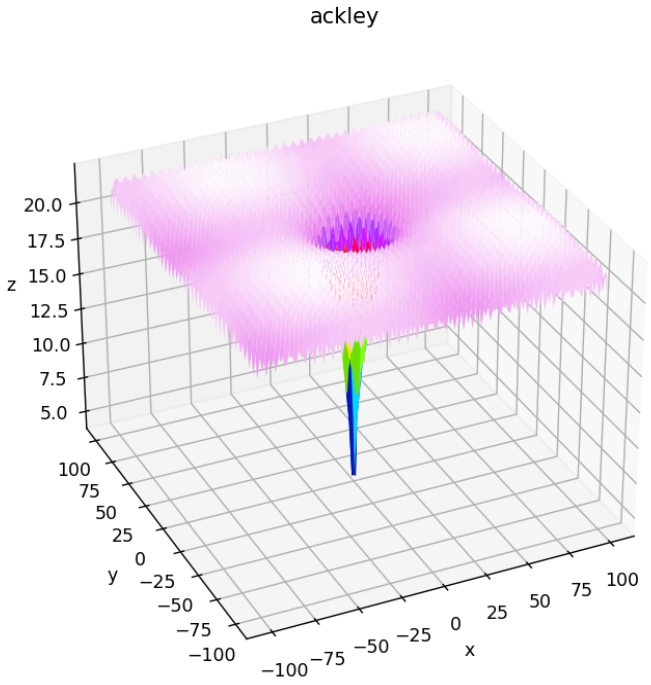
  Następuje zmiana populacji oraz reset loga poprzednich wartości ***q_best*** (musi zostać uzupełniony ponownie).

- Domyślnie parametry wynoszą: (***q_keep***=*50*, ***q_tol***=*0,5*), co oznacza, że po 50 iteracjach od ostatniej skokowej zmiany populacji (lub od początku algorytmu), co każdą iterację sprawdzany jest warunek, czy najlepsza wartość funkcji celu zmieniła się o *0,5* jednostki od średniej najlepszych wartości funkcji celu uzyskanych w ostatnich 50 iteracjach. Jeśli tak, wielkość populacji pozostaje niezmieniona. Jeśli nie, wykryto stagnację i następuje skokowa zmiana wielkości populacji.

- “Najlepsza wartość funkcji celu w iteracji” oznacza wartość funkcji celu dla najlepszego potomka w obecnej populacji.

Funkcje *f4*, *f7* oraz *ackley* brane pod uwagę mają następujące postacie przestrzeni przeszukiwań:





Optimum (zadanie minimalizacji) jest równe kolejno 400, 700 i 0 dla poszczególnych funkcji. W kolejnych wizualizacjach na rzecz lepszej widoczności śladów populacji przestrzenie są w części przezroczyste.

3. Wyniki eksperymentów (tabelaryzacja):

Dla każdej trójki {funkcja celu, funkcja zmiany populacji, wymiarowość} wykonano 25 uruchomień algorytmu, z których obliczono wartość średnią, odchylenie standardowe znalezionej wartości minimalnej i ogólnie znalezioną najmniejszą wartość funkcji. Wyniki zaokrąglone do trzech miejsc po przecinku.

Funkcja F4 CEC2017, 2 wymiary:

Funkcja zmiany populacji	q_{min}	μ	σ
f. stała (wersja bazowa)	400	400	0
f. liniowo rosnąca	400	400	0
f. liniowo malejąca	400	400	0
f. wykładniczo rosnąca	400	400	0
f. wykładniczo malejąca	400	400	0
f. okresowo zmienna (sinus)	400	400	0
f. okresowo zmienna (prostokąt)	400	400	0
f. zmiany przy stagnacji	400	400	0

Obserwacje i wnioski:
Dla dwóch wymiarów niezależnie od wybranego podejścia w każdej iteracji zostało znalezione minimum funkcji.

Funkcja F4 CEC2017, 10 wymiarów:

Funkcja zmiany populacji	q_{min}	μ	σ
f. stała (wersja bazowa)	400.187	400.758	0.28
f. liniowo rosnąca	400,275	400,741	0,325
f. liniowo malejąca	400,270	401,056	0,476
f. wykładniczo rosnąca	400,245	400,782	0,364
f. wykładniczo malejąca	400,452	400,866	0,262
f. okresowo zmienna (sinus)	400,456	400,938	0,4
f. okresowo zmienna (prostokąt)	400,386	402,085	1,311
f. zmiany przy stagnacji	400.294	403.041	1.809

Obserwacje i wnioski:
Dla dziesięciu wymiarów powstają różnice między wykonaniami algorytmu. Szczególnie widoczne jest to dla populacji zmiennej prostokątnie (okresowo oraz przy stagnacji), gdzie wariancja jest większa niż dla innych metod zmian populacji. Najbliżej optimum dotarł algorytm ze stałą liczbą populacji równą 5 potomków.

Funkcja F4 CEC2017, 20 wymiarów:

Funkcja zmiany populacji	q_{min}	μ	σ
f. stała (wersja bazowa)	406.308	463.286	23.09
f. liniowo rosnąca	403,836	465,154	22,894
f. liniowo malejąca	404,187	471,265	14,029
f. wykładniczo rosnąca	469,309	472,092	1,669
f. wykładniczo malejąca	469,917	474,391	1,637
f. okresowo zmienna (sinus)	405,552	465,412	22,482
f. okresowo zmienna (prostokąt)	470,767	474,66	1,866
f. zmiany przy stagnacji	429.841	7631.68	14511.276
f. zmiany przy stagnacji (2)	405.694	470.417	24.038

Uwaga! Dla funkcji zmiany przy stagnacji wiele przebiegów kończyło się 20000 iteracjami z jednym potomkiem (co właściwie uczyniło z algorytmu błądzenie przypadkowe), dlatego zdecydowano się na dodatkowy eksperyment ze zwiększonym parametrem tolerancji stagnacji *q_tol=5*, żeby wymusić większe prawdopodobieństwo jego zmiany. Jak widać, średnia eksperymentów po zmianie znacząco się poprawiła. Do następnych eksperymentów powrócono do poprzedniej wartości parametru *q_tol=0,5*.

Obserwacje i wnioski:
Dla 20 wymiarów obserwujemy najlepszy wynik dla liczby populacji zmiennej w czasie, liniowo rosnącej od jednego do czterdziestu potomków per iteracja. Jednak biorąc pod uwagę średnią, wciąż najlepszą pozostaje niezmienna liczba populacji, natomiast najbardziej przewidywalne wyniki otrzymano dla zmienności wykładniczych oraz okresowo zmiennej (prostokąt), gdzie wariancja jest najmniejsza.

Funkcja F7 CEC2017, 2 wymiary:

Funkcja zmiany populacji	q_{min}	μ	σ
f. stała (wersja bazowa)	702.018	704.004	2.101
f. liniowo rosnąca	707,038	716,829	6,026
f. liniowo malejąca	710,276	716,117	3,989
f. wykładniczo rosnąca	703	712,3	5,78
f. wykładniczo malejąca	702,745	709,58	5,544
f. okresowo zmienna (sinus)	705,512	715,103	6,033
f. okresowo zmienna (prostokąt)	706,518	717,165	6,482
f. zmiany przy stagnacji	702.104	714.848	9.22

Obserwacje i wnioski:
Dla funkcji *f7* najlepszy rezultat uzyskano zdecydowanie dla niezmiennej liczby populacji. W przeciągu 25 iteracji osiągnięto wynik (średni oraz minimalny) najbliższy faktycznemu optimum funkcji. Ponadto wariancja jest najmniejsza, a więc niska rozbieżność wyników między wykonaniami.

Należy zwrócić uwagę, że *f7* to jest już znacznie trudniejsza funkcja do przeszukiwania. Dla funkcji *f4* otrzymano faktyczne minimum (z założoną dokładnością) niezależnie od metody przy wymiarowości 2. Tutaj dla tej samej wymiarowości widoczne są już znaczne statystyczne różnice.

Funkcja F7 CEC2017, 10 wymiarów:

Funkcja zmiany populacji	q_{min}	μ	σ
f. stała (wersja bazowa)	731.507	744.309	6.205
f. liniowo rosnąca	726,467	733,104	3,796
f. liniowo malejąca	726,247	736,305	4,474
f. wykładniczo rosnąca	724,942	732,519	3,803
f. wykładniczo malejąca	724,647	736,277	4,576
f. okresowo zmienna (sinus)	722,598	736,714	4,899
f. okresowo zmienna (prostokąt)	732,098	741,334	4,393
f. zmiany przy stagnacji	730.097	740.108	5.694

Obserwacje i wnioski:
Po raz pierwszy można zaobserwować zdecydowaną porażkę metody ze stałą liczbą populacji. Jest najmniej przewidywalna (największa wariancja). Najsłabszy średni wynik oraz drugi najgorszy znaleziony punkt w logu z minimalną wartością funkcji celu.

Najlepszy rezultat osiągnięto dla funkcji okresowej sinus natomiast najbardziej przewidywalny dla funkcji rosnącej liniowo oraz wykładniczo, z niezłym średnim wynikiem wykonania.

Funkcja F7 CEC2017, 20 wymiarów:

Funkcja zmiany populacji	q_{min}	μ	σ
f. stała (wersja bazowa)	830.505	852.61	12.418
f. liniowo rosnąca	796,305	814,767	9,155
f. liniowo malejąca	803,654	824,925	10,449
f. wykładniczo rosnąca	801,056	817,226	9,882
f. wykładniczo malejąca	811,515	834,633	9,911
f. okresowo zmienna (sinus)	811,707	830,227	9,888
f. okresowo zmienna (prostokąt)	820,976	842,121	11,909
f. zmiany przy stagnacji	810.941	838.733	13.877

Obserwacje i wnioski:
Ponownie najgorsza jest metoda ze stałą liczbą populacji. Najlepsze okazały się te z liczbą populacji rosnącą wykładniczo oraz liniowo, które zdominowały pozostałe pod każdym względem. Można wnioskować, że dla większej wymiarowości funkcji *f7* stopniowe zwiększanie liczby populacji jest korzystne dla zadania optymalizacji.

Funkcja Ackleya, 2 wymiary:

Funkcja zmiany populacji	q_{min}	μ	σ
f. stała (wersja bazowa)	0.009	12.811	9.776
f. liniowo rosnąca	0,014	15,199	8,695
f. liniowo malejąca	0,02	12,813	9,752
f. wykładniczo rosnąca	0,02	12,019	9,971
f. wykładniczo malejąca	0,002	12,795	9,758
f. okresowo zmienna (sinus)	0,012	10,543	10,056
f. okresowo zmienna (prostokąt)	0,065	16,805	7,44
f. zmiany przy stagnacji	0.03	12.825	9.757

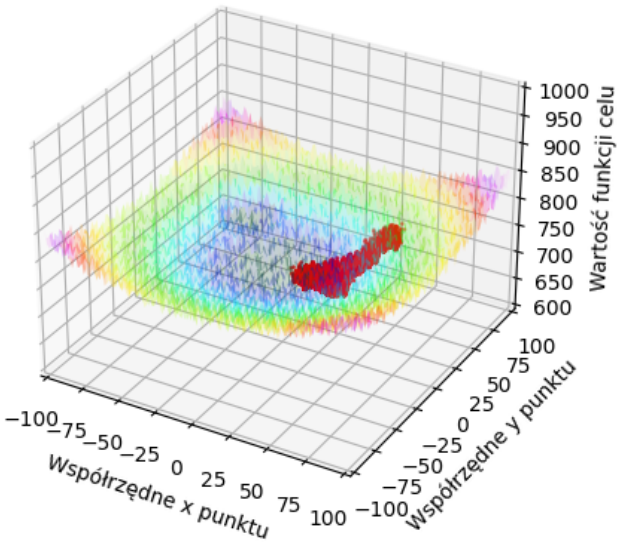
Obserwacje i wnioski:
Tym razem wyniki są zupełnie nieoczywiste. Najlepszą wartość funkcji celu osiągnięto dla metody z populacją malejącą wykładniczo. Średnio najskuteczniejsza jest wersja z populacją rosnącą wykładniczo. Natomiast najbardziej zgodne ze sobą wyniki przyniosły okresowe (prostokąt) zmiany populacji.

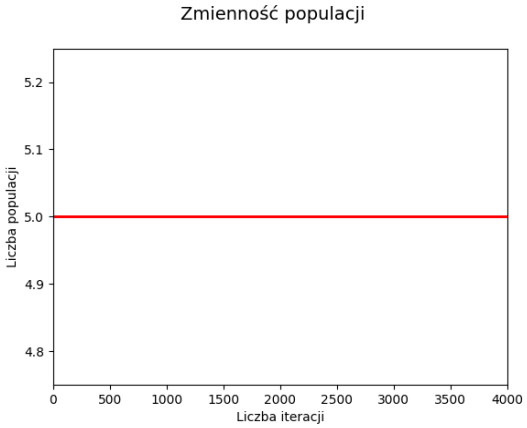
Świadczy to o niezwykłej trudności zadania optymalizacji dla funkcji Ackleya. Jest to niezwykle powtarzalna funkcja z ogromną liczbą minimów lokalnych, gdzie żaden punkt przestrzeni (wartość funkcji celu) nie wskazuje drogi do globalnego optimum.

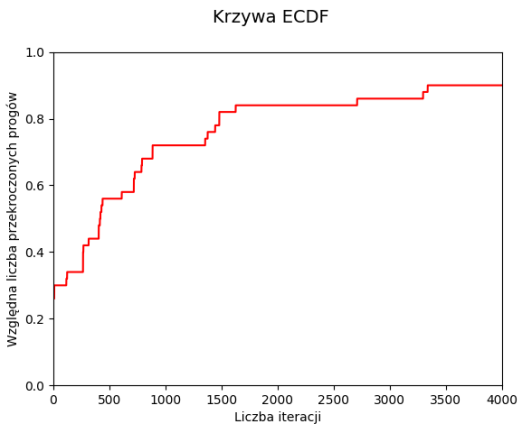
4. Wyniki eksperymentów (grafiki):

Aby zobrazować zachowanie algorytmu w zależności od wybranej metody zmienności populacji, wygenerowano dla każdej z nich przebieg algorytmu (w zadaniu przeszukiwania metody *f7* w dwóch wymiarach) wraz z wykresem przedstawiającym liczbę populacji w zależności od iteracji. Dodatkowo dołączono do każdego eksperymentu krzywą ECDF dla 50 wartości progowych równomiernie rozłożonych w przedziale [700 ; 750].

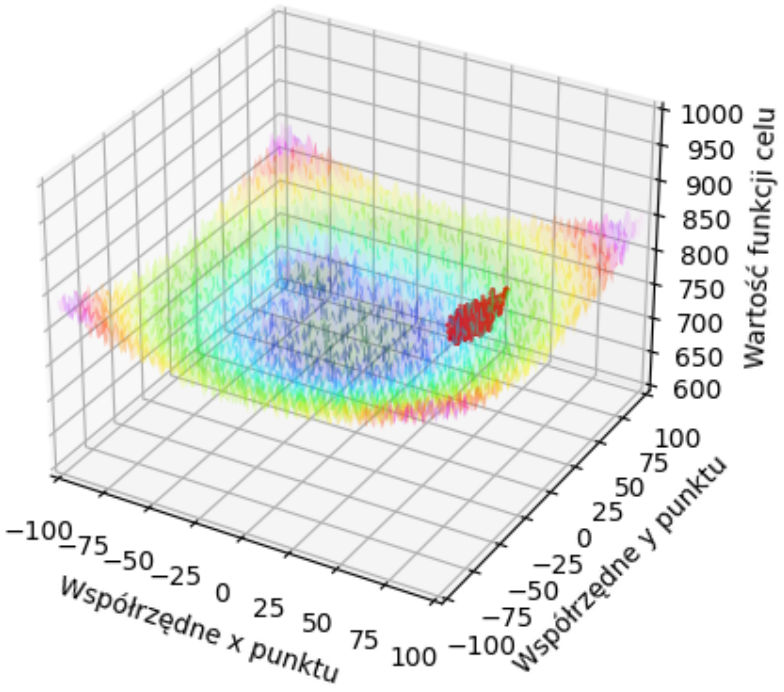
- f. stała (wersja bazowa)

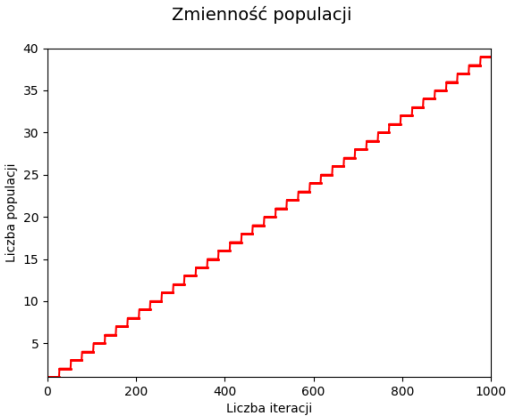


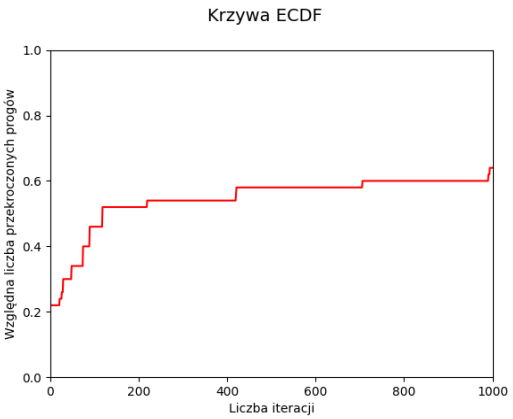




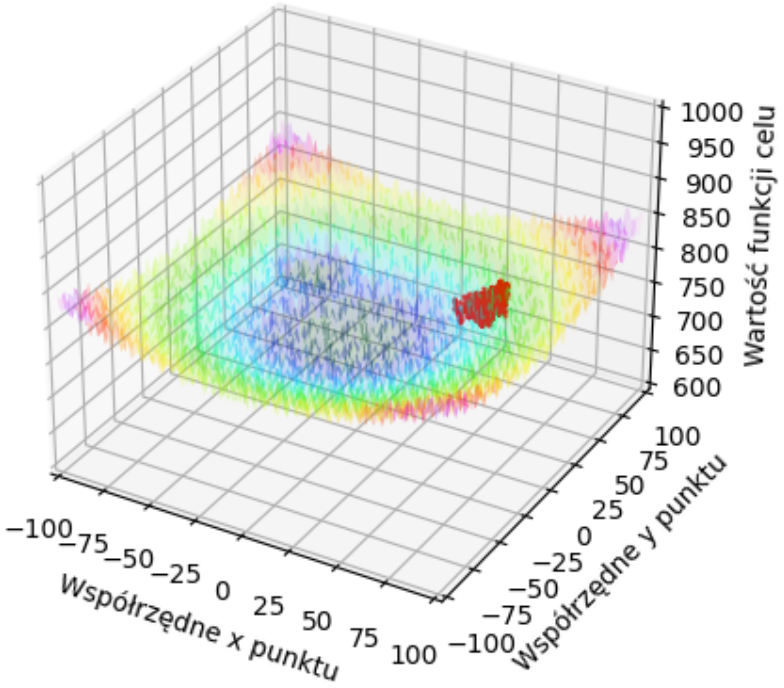
- f. liniowo rosnąca

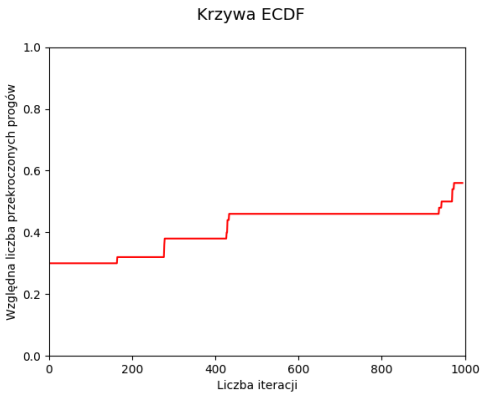




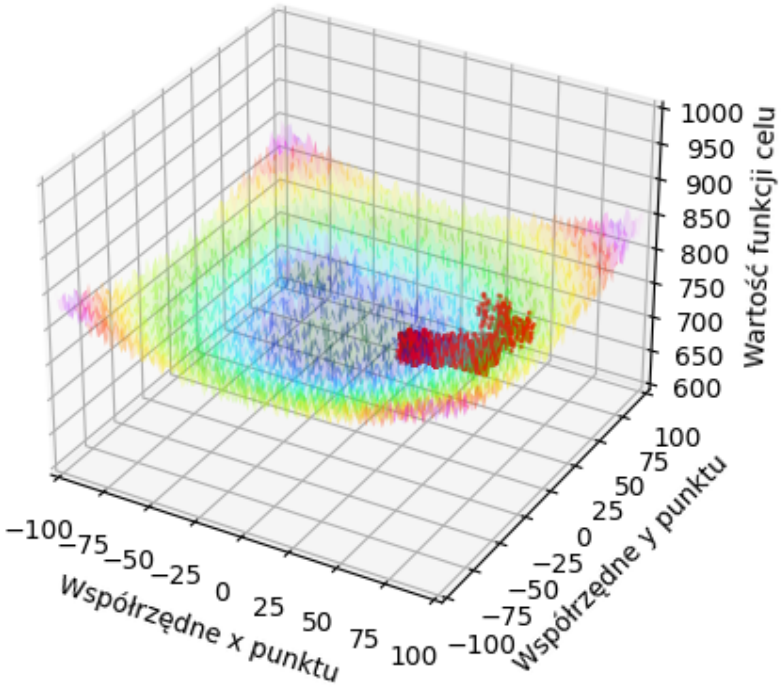


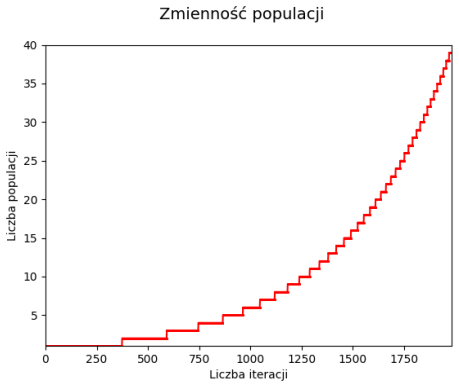
- f. liniowo malejąca

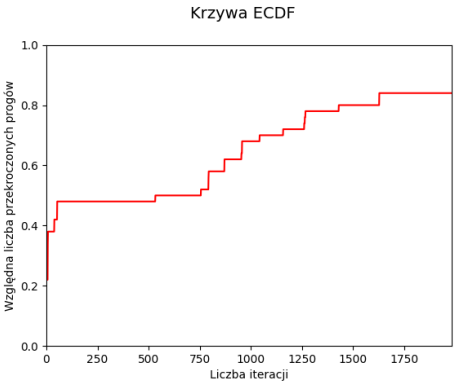




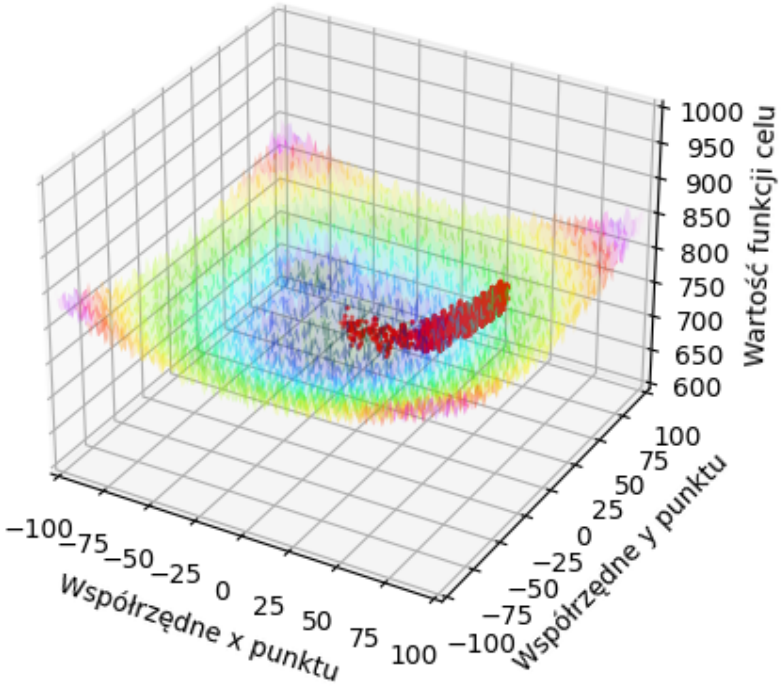
- f. wykładniczo rosnąca

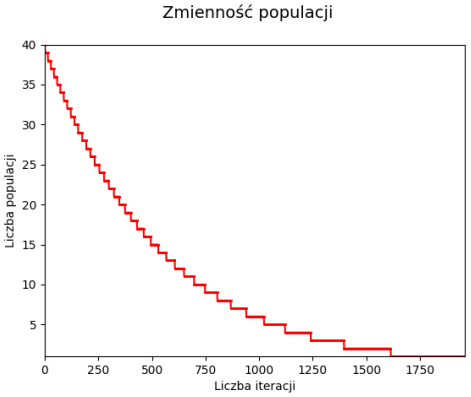


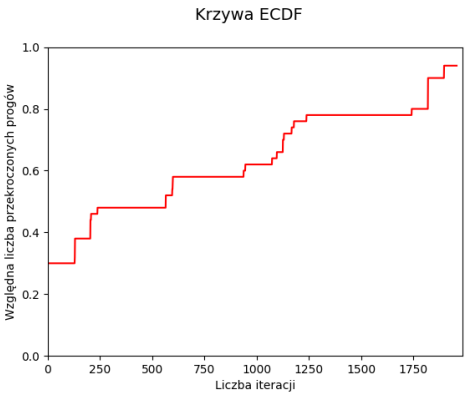




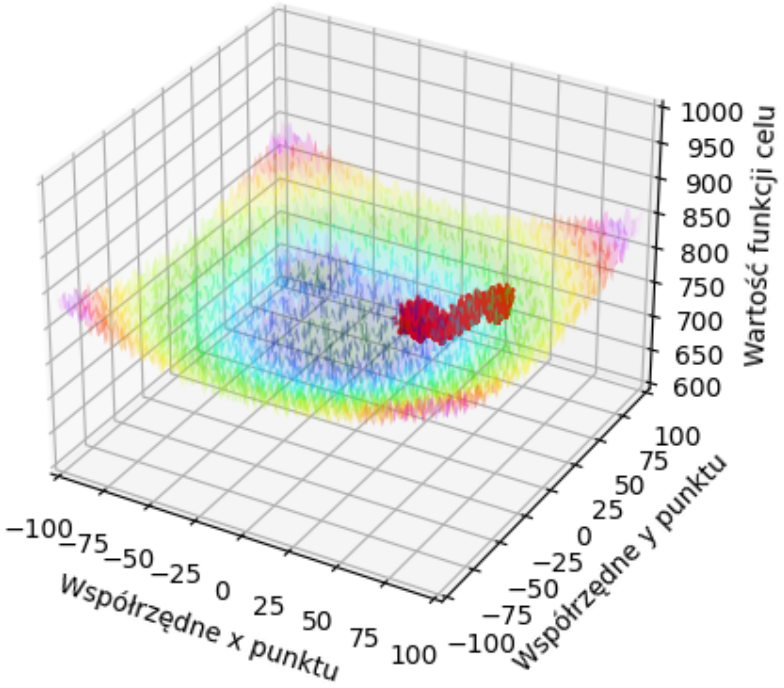
- f. wykładniczo malejąca

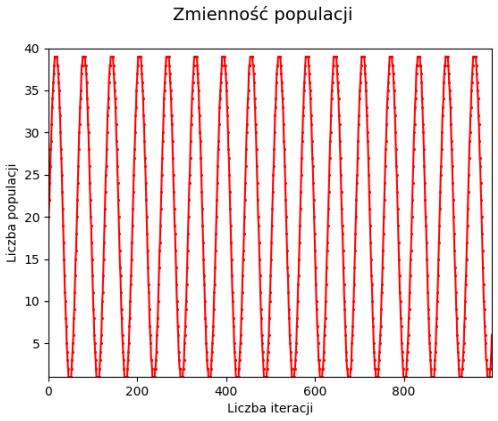


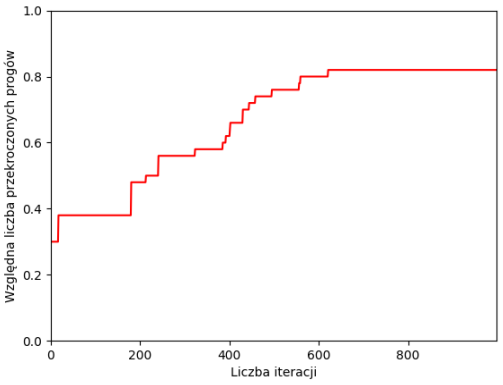




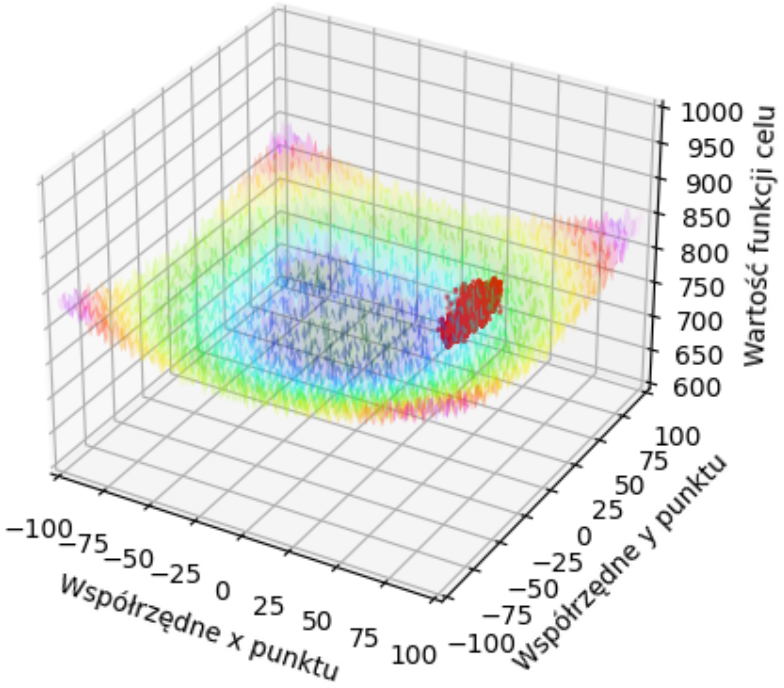
- f. okresowo zmienna (sinus)

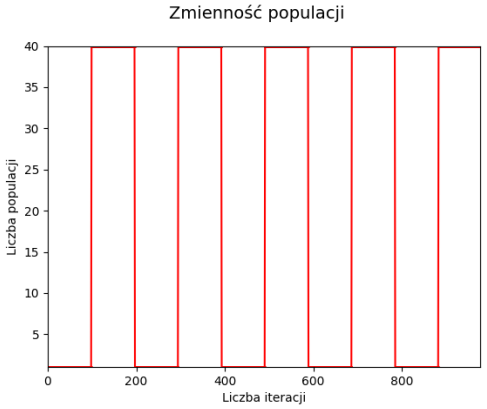


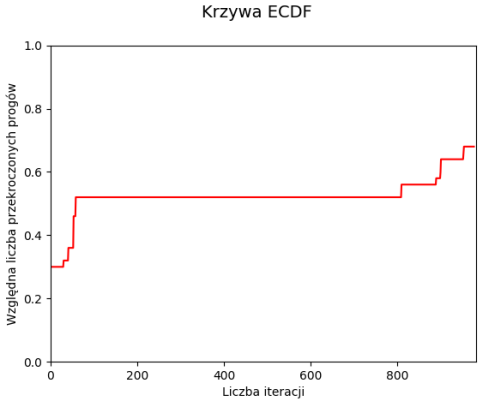




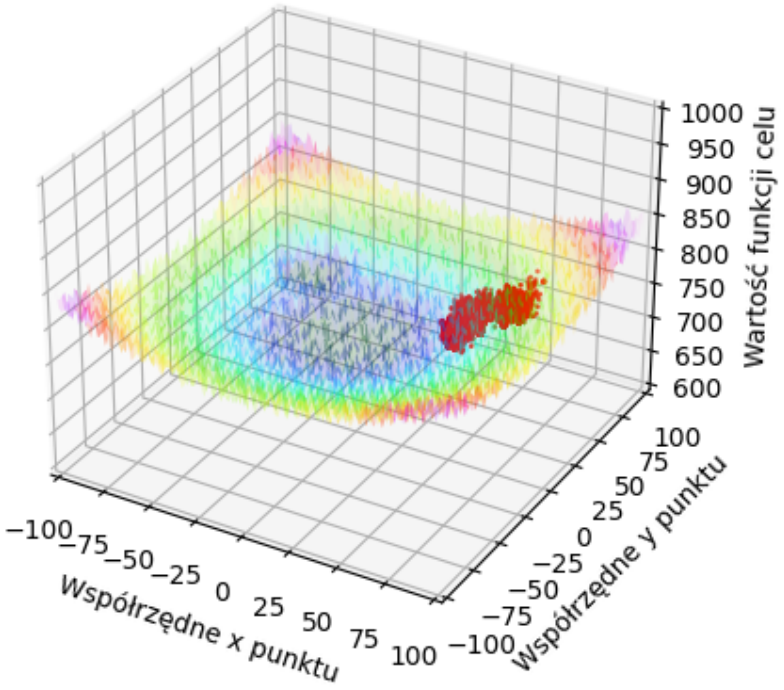
- f. okresowo zmienna (prostokąt)

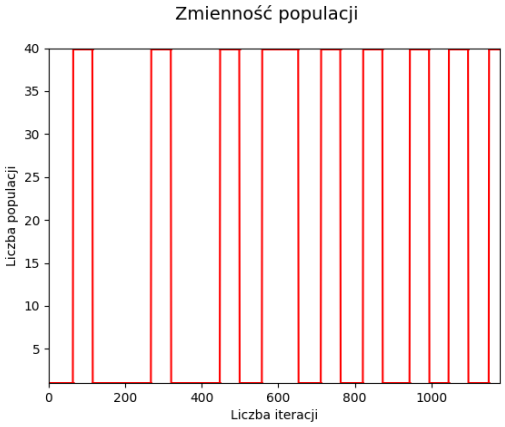


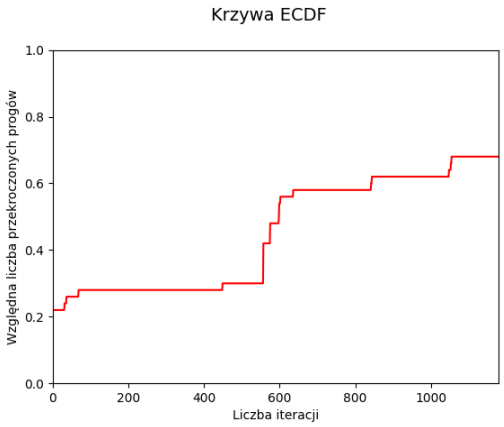




- f. zmiany przy stagnacji

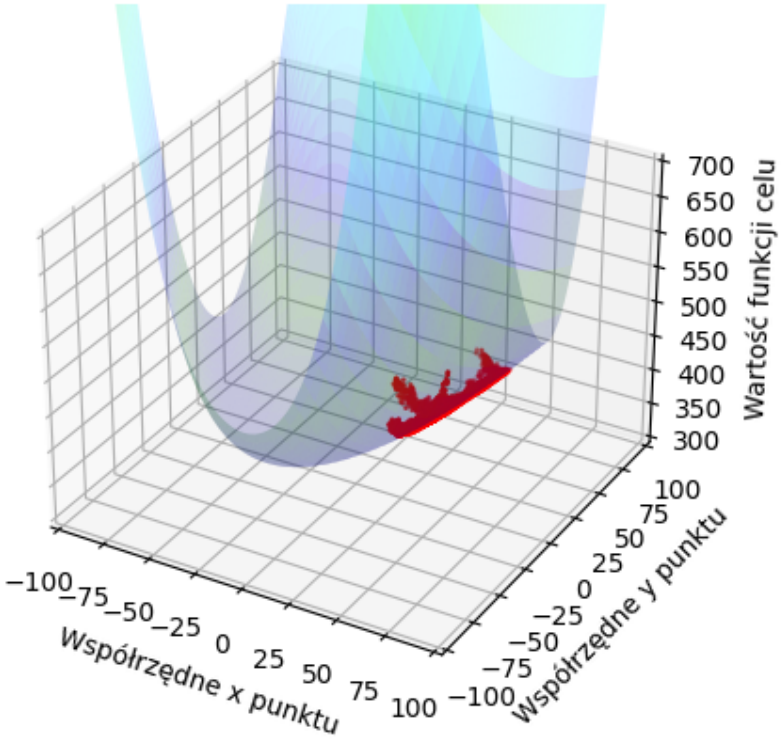


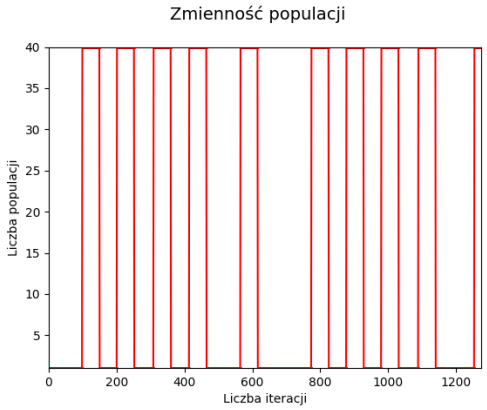


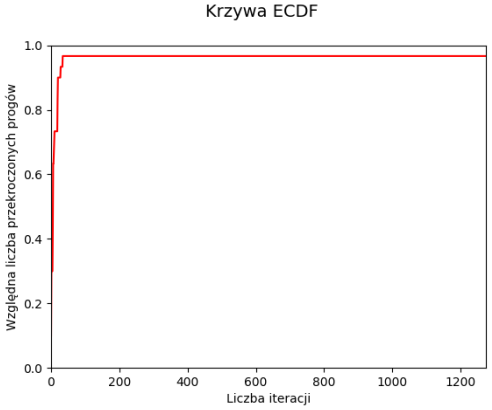


Dodatkowo, zwizualizowano eksperymenty w dwóch wymiarach dla zmiany liczby populacji przy stagnacji. Tutaj krzywa ECDF jest dla 30 równomiernie rozłożonych progów (w przedziale [400 ; 430] dla *f4*, w przedziale [0 ; 30] dla funkcji *ackley*)

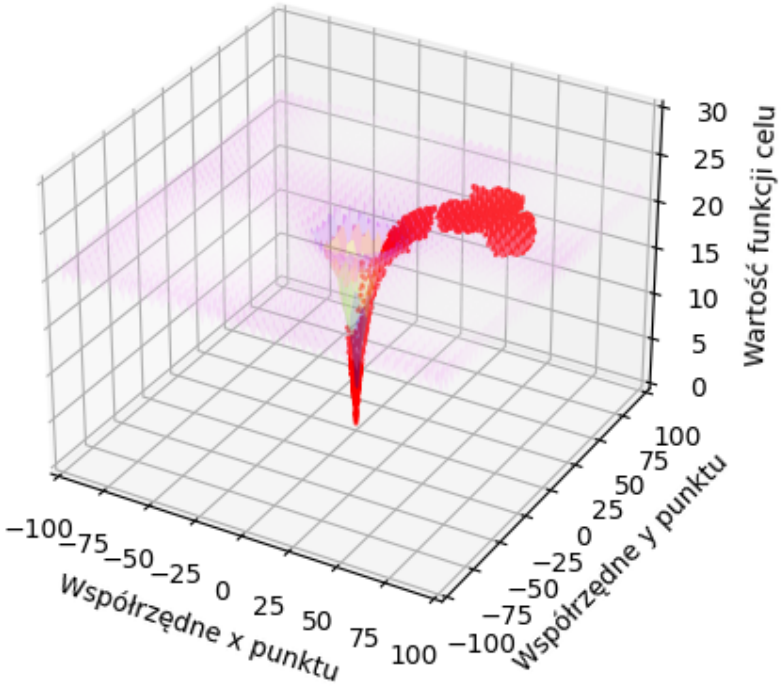
- *f4*

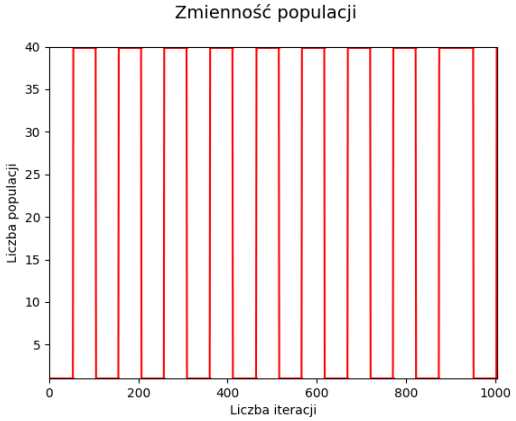


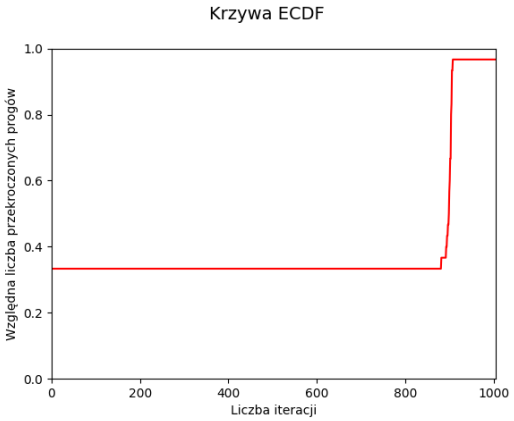




- *ackley*


Wartość funkcji celu
30
25
20
15
10
5
0
Współrzędne x punktu
−100
−75
−50
−25
0
25
50
75
100
Współrzędne y punktu
100
75
50
25
0
−25
−50
−75
−100


Zmienność populacji
Liczba populacji
Liczba iteracji
0
200
400
600
800
1000
5
10
15
20
25
30
35
40


Krzywa ECDF
Względna liczba przekroczonych progów
Liczba iteracji
0.0
0.2
0.4
0.6
0.8
1.0
0
200
400
600
800
1000

5. Ostateczne wnioski i podsumowanie

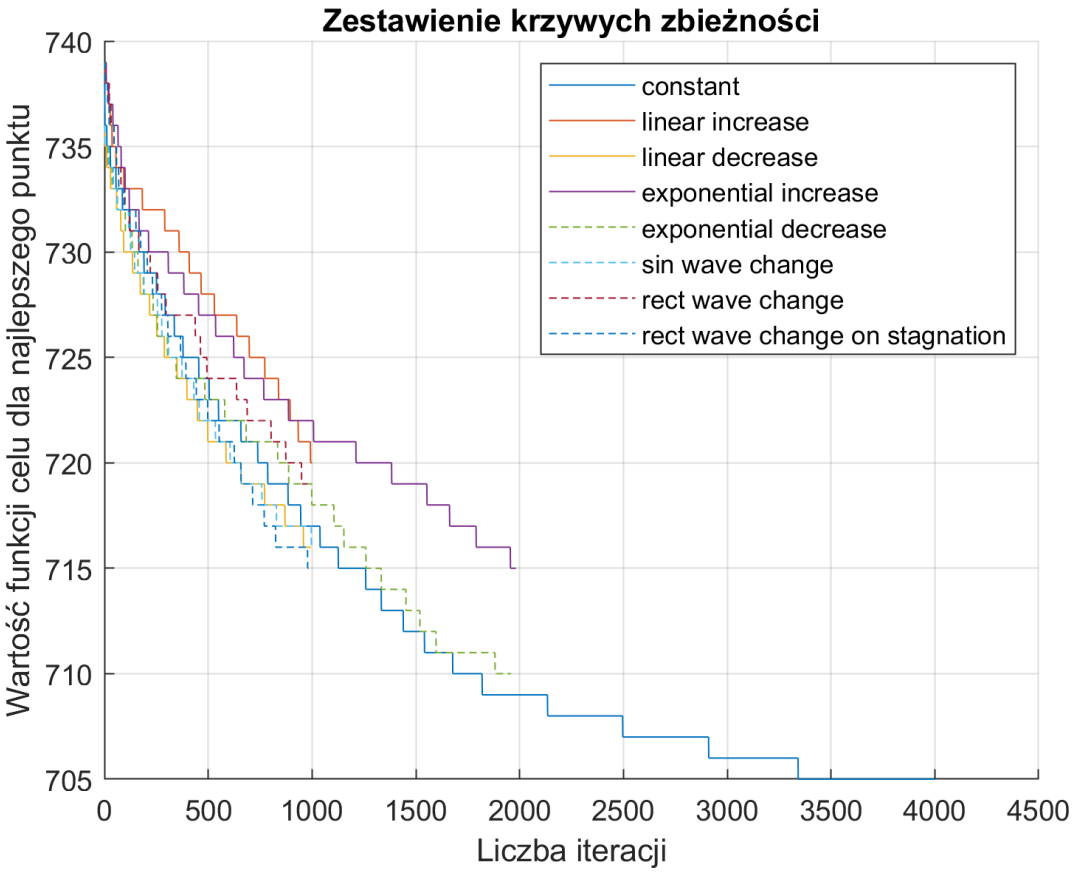
Z wizualizacji zaobserwowano, że algorytmy z metodą ze zmiennością okresową (sinus i prostokąt) populacji są mniej chętne do eksploracji, tzn. chmury punktów logu są bardziej skupione. Do wyciągnięcia większej liczby wniosków na podstawie grafik należałoby również wygenerować ich znacznie więcej, jednak tabelaryczne zestawienie 25 uruchomień algorymtu i przedstawienie statystycznych wyników jest bardziej oczywiste.

Ostatecznie żadna metoda zmiany liczby populacji w trakcie działania algorytmu nie okazała się być znacznie lepsza od podejścia standardowego. Nie zmienia to faktu, że przy konkretnych scenariuszach, takich jak dla wielowymiarowego przeszukiwania funkcji *f7*, wprowadzenie takiej metody okazało się przynosić lepsze rezultaty.

Prawdopodobnie dla scenariuszy, w których dominowała metoda ze stałą liczbą populacji, również istnieje jakaś metoda zmienności populacji konsekwentnie przynosząca korzyści. Należy jednak najpierw ją znaleźć, np. drogą eksperymentów lub analizując typowy przebieg algorytmu i decydując się na inteligentne rozwiązanie w postaci zmienności liczby populacji na podstawie przebiegu (np. stagnacji). Jak pokazały eksperymenty, implementacja zmienności przy stagnacji nie okazała się być korzystna, a wręcz charakteryzowała ją duża nieprzewidywalność (co w przypadku nieprzewidywalnych przestrzeni może być równie dobrze zaletą). Niewykluczone, że dobranie parametrów *q_tol* oraz *q_keep* pod konkretny scenariusz poskutkowałoby znaczną poprawą działania algorytmu. Widać to dla funkcji *f4* w 20 wymiarach, gdzie wymagana była zmiana, by metoda ta była w ogóle sensowna do implementacji. Być może wartoby również rozważyć, by zmienność nie była tak skokowa, a bardziej płynna w zmianie liczebności populacji.

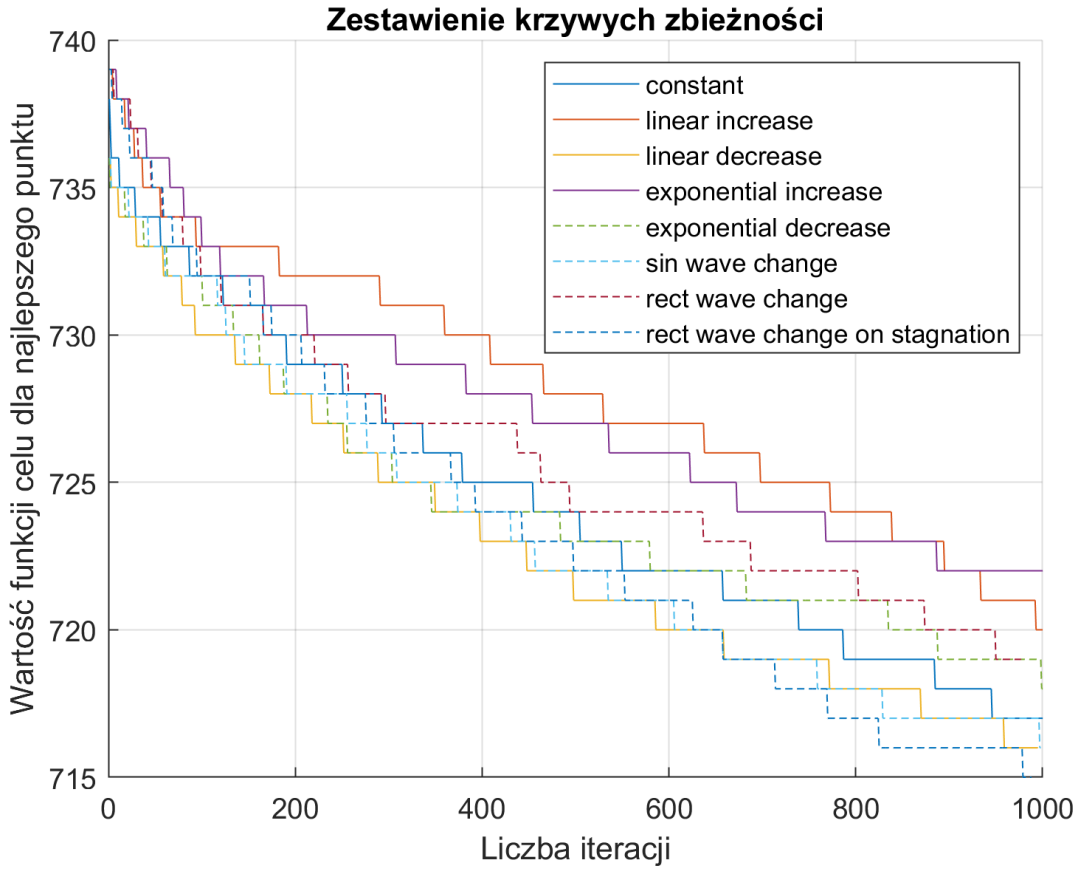
6. Dodatkowe zestawienie wyników

Zestawiono działania algorytmów na wspólnym wykresie krzywych zbieżności (kwanty co jednostkę od 700 do 740) oraz na wykresie dystrybuanty empirycznej (wykres przedstawiający, jak często dany algorytm przekracza dany próg poszukiwanej wartości minimalnej). Dla obu rodzajów wykresów, dla każdej funkcji zmiany populacji wykonano po 25 uruchomień algorytmu. Przedstawione wyniki, podobnie jak w rozdziale 4., dotyczą działania programu dla funkcji f7 w wersji dwuwymiarowej.

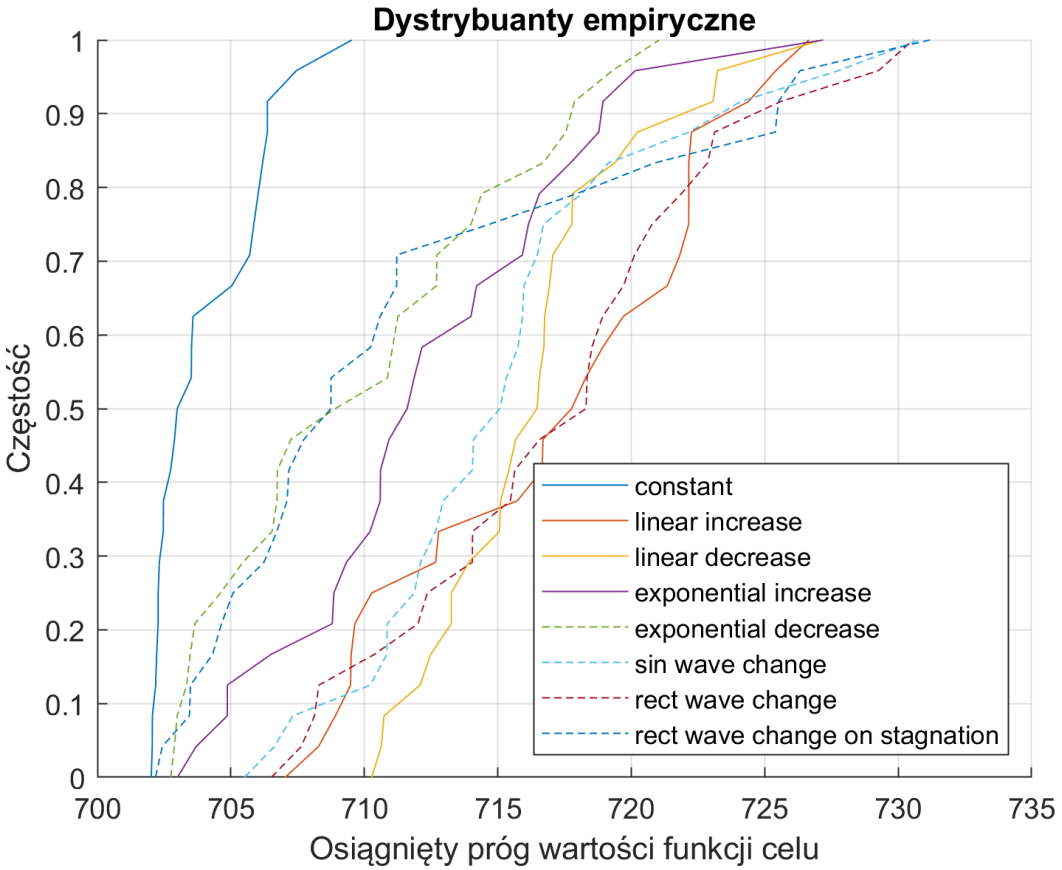


Dla powyższych eksperymentów można wskazać zwycięzcę jeżeli chodzi o jakość osiągniętego rozwiązania. Funkcja o stałej populacji dla stałego budżetu znajduje najlepsze średnio najlepsze rozwiązanie / osiąga najwięcej celów. Warto jednak zwrócić uwagę, że porównujemy przebiegi eksperymentów o stałym budżecie, co oznacza, że nie każda metoda działała przez tyle samo iteracji.

Gdybyśmy mieli np. narzucić ograniczenie do tysiąca pierwszych iteracji (czyli mniej więcej tyle, ile trwały najkrótsze eksperymenty), otrzymalibyśmy wykres, jak poniżej:



Dla pierwszych 1000 iteracji wyniki nie są już oczywiste, ani takie same, jak w poprzednim porównaniu. Algorytm o stałej populacji znajduje się mniej więcej pośrodku, tzn. wypada przeciętnie na tle pozostałych metod. Najlepszymi w tym zestawieniu okazały się algorytmy o liniowym spadku populacji, sinusoidalnej zmianie populacji i funkcja zmiany populacji przy stagnacji. Trudno wskazać najlepszy algorytm z tej trójki, ponieważ ich krzywe się przecinają.



Jeżeli chodzi o konsekwentność algorytmów w osiąganiu minimalnych rozwiązań, to zdecydowanie najlepszym algorytmem jest algorytm o stałej populacji. Jest to zrozumiały wynik, ponieważ funkcja *f7* ma bardzo wiele minimów lokalnych i dotarcie do minimum globalnego jest obarczone pewną losowością, którą dodatkowo zwiększa zmienność populacji. Ze zmiennych populacji niewiele gorsze wyniki osiągają algorytm o wykładniczym spadku i z funkcją zmiany przy stagnacji. Najgorsze wyniki osiągają zmienności liniowe i funkcja prostokątna.

Informacje dotyczące aplikacji

Link do repozytorium z implementacją algorytmu:
https://gitlab-stud.elka.pw.edu.pl/jfirlej/pop_22l

Zachęcamy do uruchomienia skryptu w pliku *main.py*. Zmienne ustawiane są w kodzie pod komentarzem “INIT VARIABLES”:

- funkcje celu oraz populacji wybieramy wpisując nazwę pod odpowiednią zmienną *q_name* lub *pop_f_name*.
    - dostępne nazwy funkcji populacji można podejrzeć w pliku functions/population_functions.py
- Wymiarowość zadania wybieramy zmieniając wartość zmiennej *dimensions*
- Wartości graniczne liczby populacji definiują zmienne *pop_min* oraz *pop_max*

Zaimplementowaliśmy podgląd przebiegu algorytmu na żywo w *matplotlib*! Domyślnie jest to włączone - można wyłączyć zmieniając analogicznie nazwane zmienne pod komentarzem “PLOT SETTINGS”.

Potrzebne do uruchomienia:

- wymagany Python w wersji >= 3.9
- wymagane moduły:
    - *matplotlib*
    - *numpy*
    - *scipy*
    - *cec2017-py**

* moduł cec2017-py należy pobrać z github (https://github.com/tilleyd/cec2017-py) i zainstalować ręcznie zgodnie z instrukcją (tego modułu nie ma w menedżerze pakietów *pip*)