

POP - dokumentacja końcowa

Paweł Kochański, Jakub Proboszcz

1 Opis problemu (zadanie 1)

"Zbadaj działanie dowolnego wariantu ewolucji różnicowej oraz dowolnej strategii ewolucyjnej w warunkach niepewności. W tym celu opracuj zestaw funkcji testowych, które będą zawierały co najmniej dwa różne źródła niepewności."

2 Sposób rozwiązania problemu

Przygotowaliśmy implementacje wybranego wariantu ewolucji różnicowej i strategii ewolucyjnej $\mu + \lambda$. Przygotowaliśmy również implementacje funkcji testowych zawierających po 2 źródła losowości - zaburzenie argumentu oraz wyniku funkcji. Wykonaliśmy eksperymenty porównujące działanie obydwu algorytmów, dla różnych parametrów wywołania i dla różnych poziomów niepewności.

3 Szczegółowy opis algorytmów

3.1 Algorytm ewolucji różnicowej

Wybraliśmy ewolucję różnicową z selekcją **b**est, 1 parą różnicowanych punktów i krzyżowaniem dwumianowym (DE/best/1/bin).

Pseudokod algorytmu:

Dane:

P^0 - populacja początkowa (elementy: $P_1^0, P_2^0, \dots, P_\mu^0$)

μ - rozmiar populacji

F - waga różnicowa - parametr mutacji różnicowej

c_r - prawdopodobieństwo krzyżowania

t_{max} - maksymalna liczba iteracji

Algorytm 1 Ewolucja różnicowa DE/best/1/bin

```
t ← 0
while  $t < t_{max}$  do
    for  $i \in 1 : \mu$  do
         $A \leftarrow$  najlepszy( $P^t$ )
         $B \leftarrow$  losuj( $P^t$ )
         $C \leftarrow$  losuj( $P^t$ )
         $M \leftarrow A + F(B - C)$ 
         $O \leftarrow$  krzyżowanie( $P_i^t, M, c_r$ )
         $P_i^{t+1} \leftarrow$  lepszy( $P_i^t, O$ )
    end for
     $t \leftarrow t + 1$ 
end while
```

Krzyżowanie jest opisane poniżej.

Dane:

x, y - dwa osobniki (rodzice)

c_r - prawdopodobieństwo krzyżowania

n - liczba wymiarów osobnika w populacji

Algorytm 2 Krzyżowanie dwumianowe w ewolucji różnicowej

```
procedure krzyżowanie
arguments:  $x, y, c_r$ 
for  $i \in 1 : n$  do
    losuj  $a \sim U(0, 1)$ 
    if  $a < c_r$  then
         $z_i \leftarrow y_i$ 
    else
         $z_i \leftarrow x_i$ 
    end if
end for
return  $z$ 
```

Ewaluacja funkcji celu jest wykonywana przy wyliczaniu ocen populacji początkowej oraz dla wyniku krzyżowania, przy jego porównaniu z elementem populacji. Dodatkowo, ze względu na losowość obecną w funkcjach celu, oceny wszystkich punktów populacji są ponownie przeliczane na końcu każdej iteracji. W sumie, algorytm wykonuje $\mu + 2 \cdot \mu \cdot t_{max}$ wywołań funkcji celu.

Zwracany jest element o najlepszej ocenie z końcowej populacji.

3.2 Algorytm strategii ewolucyjnej

Wybraliśmy strategię ewolucyjną $\mu + \lambda$ z krzyżowaniem uśredniającym wartość osobnika i jego siłę mutacji, z losową wagą.

Pseudokod algorytmu:

Dane:

P^0 - populacja początkowa (elementy: $P_1^0, P_2^0, \dots, P_\mu^0$)

μ - rozmiar populacji bazowej

λ - rozmiar populacji potomnej

σ_{init} - początkowa siła mutacji

t_{max} - maksymalna liczba iteracji

Algorytm 3 Strategia ewolucyjna $\mu + \lambda$

```
 $t \leftarrow 0$ 
 $\sigma^0 \leftarrow \text{inicjuj(tablica takiego samego kształtu jak } P^0\text{, gdzie każdy element ma wartość } \sigma_{init}\text{)}$ 
while  $t < t_{max}$  do
     $R, \sigma^R \leftarrow \text{losuj } \lambda \text{ elementów z powtórzeniami z } P^t, \sigma^t \text{ (osobnik i jego siła mutacji razem)}$ 
     $C, \sigma^C \leftarrow \text{krzyżowanie}(R, \sigma^R)$ 
     $M, \sigma^M \leftarrow \text{mutacja}(C, \sigma^C)$ 
     $P^{t+1}, \sigma^{t+1} \leftarrow \text{wybierz } \mu \text{ najlepszych z } P^t \cup M, \text{ razem z ich siłami mutacji z } \sigma^t \cup \sigma^M$ 
     $t \leftarrow t + 1$ 
end while
```

Krzyżowanie jest opisane poniżej.

Dane:

R - populacja po reprodukcji; elementy: $R^1, R^2, \dots, R^\lambda$

σ^R - siły mutacji populacji po reprodukcji; elementy: $\sigma^{R,1}, \sigma^{R,2}, \dots, \sigma^{R,\lambda}$

n - liczba wymiarów osobnika w populacji

Algorytm 4 Krzyżowanie uśredniające w strategii ewolucyjnej $\mu + \lambda$

procedure **krzyżowanie**

arguments: R, σ^R

losowo zmień kolejność elementów w R , i w taki sam sposób w σ^R (tak, aby elementy i ich siły mutacji pozostały sparowane)

for $i \in 1 : \lambda$ **do**

 wybierz rodziców dla i -tego elementu:

$$p^1 \leftarrow R^{2 \cdot \lfloor \frac{i-1}{2} \rfloor + 1}$$

$$\sigma^{p,1} \leftarrow \sigma^{R,2 \cdot \lfloor \frac{i-1}{2} \rfloor + 1}$$

$$p^2 \leftarrow R^{2 \cdot \lfloor \frac{i-1}{2} \rfloor + 2}$$

$$\sigma^{p,2} \leftarrow \sigma^{R,2 \cdot \lfloor \frac{i-1}{2} \rfloor + 2}$$

for $j \in 1 : n$ **do**

 losuj $a \sim U(0, 1)$

$$C_j^i \leftarrow a \cdot p_j^1 + (1 - a) \cdot p_j^2$$

$$\sigma_j^{C,i} \leftarrow a \cdot \sigma_j^{p,1} + (1 - a) \cdot \sigma_j^{p,2}$$

end for

end for

return C, σ^C

Ze względu na to, że populacja rodziców jest dobierana w pary, rozmiar populacji potomnej λ powinien być parzysty.

Mutacja jest opisana poniżej.

Dane:

C - populacja po krzyżowaniu; elementy: $C^1, C^2, \dots, C^\lambda$

σ^C - siły mutacji populacji po krzyżowaniu; elementy: $\sigma^{C,1}, \sigma^{C,2}, \dots, \sigma^{C,\lambda}$

n - liczba wymiarów osobnika w populacji

Algorytm 5 Mutacja w strategii ewolucyjnej $\mu + \lambda$

procedure **mutacja**

arguments: C, σ^C

for $i \in 1 : \lambda$ **do**

 losuj $a \sim \mathcal{N}(0, 1)$

for $j \in 1 : n$ **do**

 losuj $b, c \sim \mathcal{N}(0, 1)$

$$\sigma_j^{M,i} \leftarrow \sigma_j^{C,i} \cdot \exp\left(a \cdot \frac{1}{\sqrt{2\sqrt{n}}} + b \cdot \frac{1}{\sqrt{2n}}\right)$$

$$M_j^i \leftarrow C_j^i + c \cdot \sigma_j^{M,i}$$

end for

end for

return M, σ^M

Ewaluacja funkcji celu jest wykonywana przy wyliczaniu ocen populacji początkowej oraz dla wyników krzyżowania, przed sukcesją. Dodatkowo, ze względu na losowość obecną w funkcjach celu, oceny wszystkich punktów populacji bazowej są ponownie przeliczane przed sukcesją. W sumie, algorytm wykonuje $\mu + (\mu + \lambda) \cdot t_{max}$ wywołań funkcji celu.

4 Optymalizowane funkcje

4.1 Źródła niepewności

Każde wywołanie funkcji testowej $f(\mathbf{x})$ jest zaburzone - rzeczywisty otrzymany wynik wynosi $f(\mathbf{x} + \boldsymbol{\zeta}_1) + \zeta_2$, gdzie $\boldsymbol{\zeta}_1 = [z_1 z_2 \dots z_n]^T$ (n to liczba wymiarów wektora \mathbf{x}) oraz $z_i \sim \mathcal{N}(0, \sigma_1^2)$, a ζ_2 analogicznie. σ_1, σ_2 to parametry ustalone na poziomie danego eksperymentu.

Oznacza to, że zaburzane są niezależnie argumenty i wartości funkcji.

4.2 Zestaw funkcji

Wszystkie te funkcje mają być minimalizowane. Wszystkie poza funkcjami Himmelblau i Eggholder są określone dla dowolnej liczby wymiarów. Podczas eksperymentów wykorzystamy warianty 10, 30 i 50-wymiarowe tych funkcji.

- Wielowymiarowa funkcja kwadratowa

$$f(\mathbf{x}) = \sum_{i=1}^n x_i^2$$

Podstawowa funkcja z 1 optimum lokalnym w wektorze zerowym, o wartości 0.

- Suma wielomianów

$$f(\mathbf{x}) = \sum_{i=1}^n x_i^4 + 3x_i^3 + 2x_i^2$$

Stosunkowo prosta funkcja z 2^n optimów lokalnych. Optimum globalne jest w punkcie o wszystkich współrzędnych równych około $-1,6404$, o wartości około $-0,6197 \cdot n$.

- Zaburzona funkcja kwadratowa

$$f(\mathbf{x}) = \sum_{i=1}^n \frac{x_i^2}{100} + \sin(x_i) + \sin(2x_i)$$

Funkcja z nieskończoną liczbą minimów lokalnych, z minimum globalnym w punkcie o wszystkich współrzędnych równych około $-0,9319$, o wartości około $-1,7515 \cdot n$.

- Funkcja Ackley'a

$$f(\mathbf{x}) = -20 \exp \left(-0.2 \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2} \right) - \exp \left(\frac{1}{n} \sum_{i=1}^n \cos(2\pi x_i) \right) + e + 20$$

Funkcja ta posiada płaski region zewnętrzny i znacznie niższe wartości w okolicach zera. Równocześnie występuje nieskończenie wiele minimów lokalnych w całej dziedzinie funkcji. Minimum globalne jest w wektorze zerowym, o wartości 0.

- Funkcja Himmelblau

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$

Funkcja ta wyróżnia się posiadaniem 4 identycznych minimów globalnych w punktach (około) $(3; 2)$, $(-2,8051; 3,1313)$, $(-3,7793; -3,2832)$ i $(3,5844; -1,8481)$, o wartości 0. Określona tylko dla dwóch wymiarów.

- Funkcja Eggholder

$$f(x, y) = -(y + 47) \sin \sqrt{\left| \frac{x}{2} + (y + 47) \right|} - x \sin \sqrt{|x - (y + 47)|}$$

Funkcja ta posiada nieskończenie wiele minimów lokalnych; minimum globalne w punkcie $(512; 404,2319)$ o wartości około $-959,6407$ nie wyróżnia się spośród pozostałych minimów lokalnych. Określona tylko dla dwóch wymiarów.

Dla tej funkcji optymalizacja zachodzi tylko w przedziale $[-512, 512] \times [-512, 512]$. Ograniczenie to realizujemy w sposób niewidoczny dla algorytmu, metodą odbijania. Argumenty funkcji Eggholder, przed wyliczeniem wartości funkcji, są zawężane do przedziału $[-512, 512]$ funkcją

$$g(x) = |x + 512 - 2048 \lfloor \frac{x + 1536}{2048} \rfloor| - 512$$

Ostatecznie optymalizowana jest funkcja $f(g(x), g(y))$.

5 Implementacja

Kod algorytmów, eksperymentów oraz analizy wyników powstał w języku Python. Wykorzystuje biblioteki: NumPy (<https://numpy.org/>), Matplotlib (<https://matplotlib.org/>), Seaborn (<https://seaborn.pydata.org/>), Pandas (<https://pandas.pydata.org/>), Scipy (<https://scipy.org/>).

Implementacje obu algorytmów zawarte są w plikach `es_mu_plus_lambda.py` oraz `differential_evolution.py`.

Eksperymenty przeszukiwania po hipersiatce parametrów są wykonywane przez skrypty `experiments_es.py` oraz `experiments_de.py`; testowana funkcja celu jest wybierana na podstawie argumentu (liczby od 0 do 5), wyniki są zapisywane do plików w katalogu `experiment_results`, o nazwach takich jak `es_mu_plus_lambda_ackley.csv`.

Wykresy typu `stripplot`, wykresy pudełkowe oraz najlepsze zestawy parametrów wylicza skrypt `get_best_params.py`.

Eksperymenty porównujące oba algorytmy dla ich najlepszych zestawów parametrów wykonuje skrypt `comparison_experiments.py`; uśrednione wyniki (również dla poszczególnych iteracji, w celu wygenerowania krzywych) są zapisywane do pliku binarnego `plot_data.pkl`.

Na podstawie tego pliku, wykresy krzywych zbieżności i krzywych ECDF wykonuje skrypt `generate_plots.py`. Tabele dla porównania obu algorytmów wykonuje skrypt `generate_comparison_tables.py`.

Program był testowany z użyciem Pythona 3.11. Zależności zewnętrzne wraz z ich wersjami są opisane w pliku `requirements.txt`. Kod był testowany na platformach Windows 10, Ubuntu 20.04 oraz Ubuntu 22.04. Załączony jest również Dockerfile umożliwiający budowę obrazu, na którym uruchomienie skryptów daje oczekiwane wyniki. Nie załączamy zbudowanego obrazu ze względu na jego rozmiar.

6 Eksperymenty

6.1 Przeszukiwanie po hipersiatce parametrów

Dla obu algorytmów wyznaczyliśmy zbiory sensownych wartości każdego z ich parametrów. Przetestowaliśmy algorytmy dla wszystkich kombinacji parametrów z tych zbiorów, dla wszystkich funkcji, dla sił zaburzeń losowych $\sigma_1, \sigma_2 \in \{1; 10\}$.

Populacja początkowa P^0 , dla obu algorytmów i dla wszystkich funkcji, była losowana ze zbioru $[-512, 512]^n$ ($[-32, 32]^n$ dla funkcji Ackley'a - przy losowaniu z większego przedziału żaden z algorytmów nie zbliżał się

nawet do okolic minimum globalnego), z rozkładem jednostajnym, gdzie n to zadana liczba wymiarów.

W każdym uruchomieniu liczba iteracji algorytmu t_{max} została ustalona na maksymalną taką, dla której liczba obliczeń funkcji celu nie przekracza 150000.

Sprawdzane wartości parametrów dla algorytmu ewolucji różnicowej:

- rozmiar populacji μ - 50; 70; 90
- waga różnicowa F - 0,3; 0,5; 0,7
- prawdopodobieństwo krzyżowania c_r - 0,2; 0,4; 0,6

Liczba iteracji algorytmu wynosiła $t_{max} = \lfloor \frac{150000-\mu}{2\mu} \rfloor$.

Sprawdzane wartości parametrów dla algorytmu strategii ewolucyjnej:

- rozmiar populacji bazowej μ - 10; 20; 30
- rozmiar populacji potomnej λ - 5μ ; 7μ ; 9μ
- początkowa siła mutacji σ_{init} - 0,02r; 0,1r; 0,5r, gdy populacja początkowa była losowana ze zbioru $[-\frac{r}{2}, \frac{r}{2}]^n$ ($r = 64$ dla funkcji Ackley'a i $r = 1024$ dla pozostałych funkcji)

Liczba iteracji algorytmu wynosiła $t_{max} = \lfloor \frac{150000-\mu}{\mu+\lambda} \rfloor$.

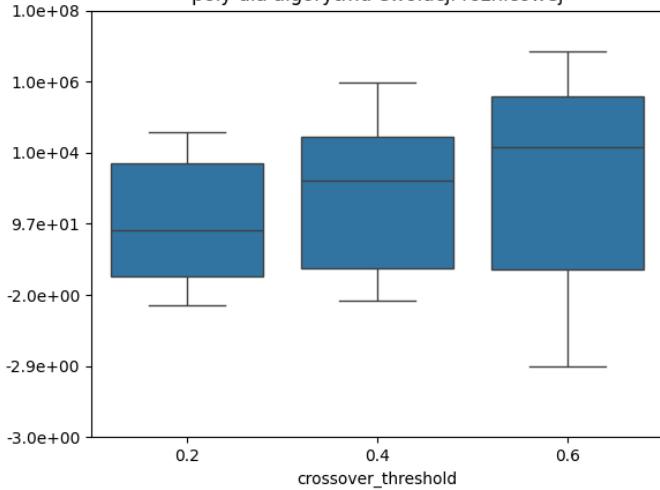
Dla każdej kombinacji parametrów wykonaliśmy po 25 uruchomień algorytmu (ziarna generatorów liczb losowych: od 0 do 24 włącznie); na poniższych wykresach przedstawione są średnie wyniki algorytmów.

Najlepsze kombinacje parametrów dla obu algorytmów wraz z ich wynikami są zawarte w załączonych plikach `best_hyper_de.csv` i `best_hyper_mu_plus_lambda.csv`.

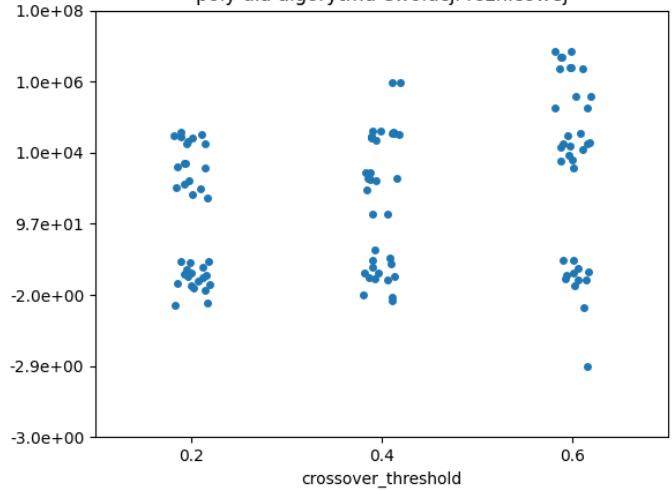
Poniżej przedstawione są wykresy typu `stripplot` oraz wykresy pudełkowe zagregowanych wyników dla różnych kombinacji parametrów prezentujące wpływ parametrów na wyniki obu algorytmów, dla 10-wymiarowej sumy wielomianów oraz dla funkcji Eggholder. Oś y wykresów dla funkcji wielomianowej jest w skali logarytmicznej. Dane zostały zagregowane ze wszystkich wariantów losowości.

6.1.1 Parametry algorytmu ewolucji różnicowej

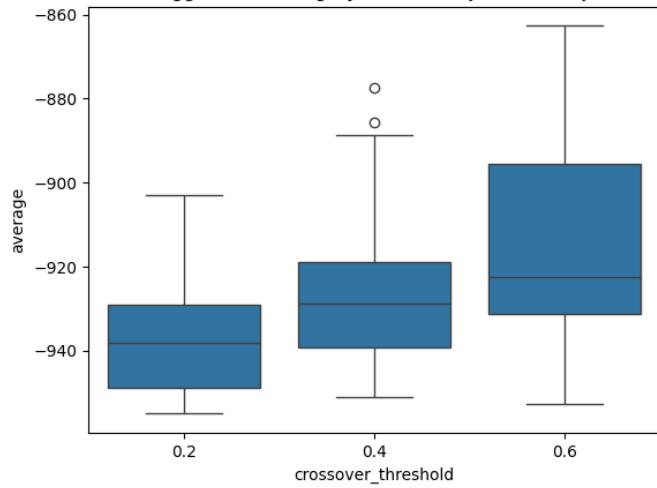
Wpływ crossover_threshold na średnią wartość 10-wymiarowej funkcji poly dla algorytmu ewolucji różnicowej



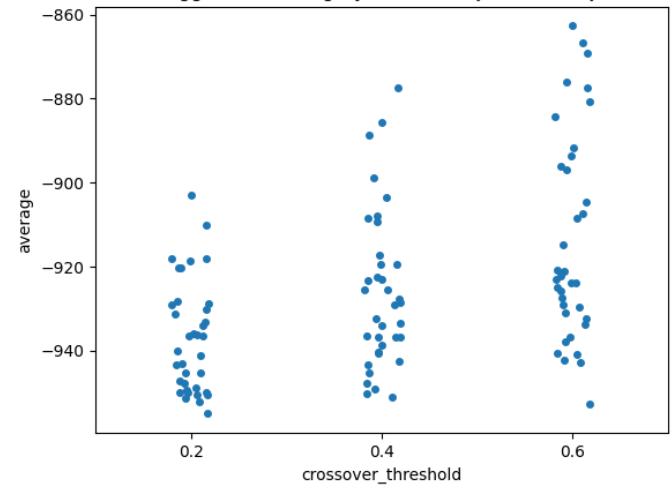
Wpływ crossover_threshold na średnią wartość 10-wymiarowej funkcji poly dla algorytmu ewolucji różnicowej



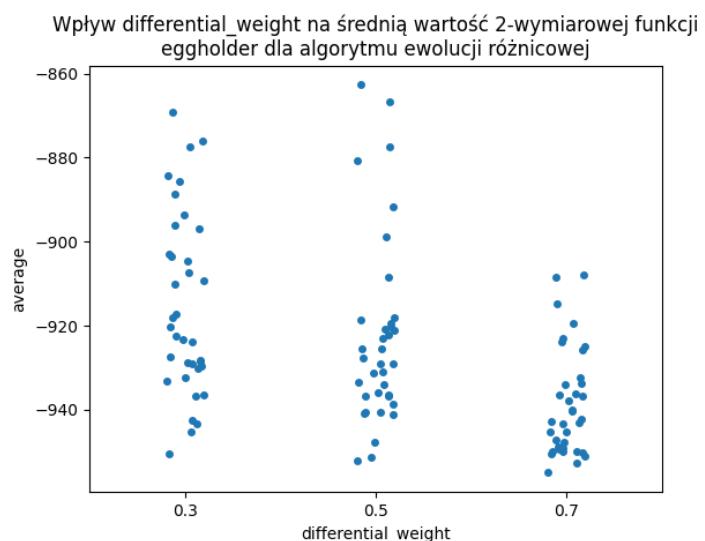
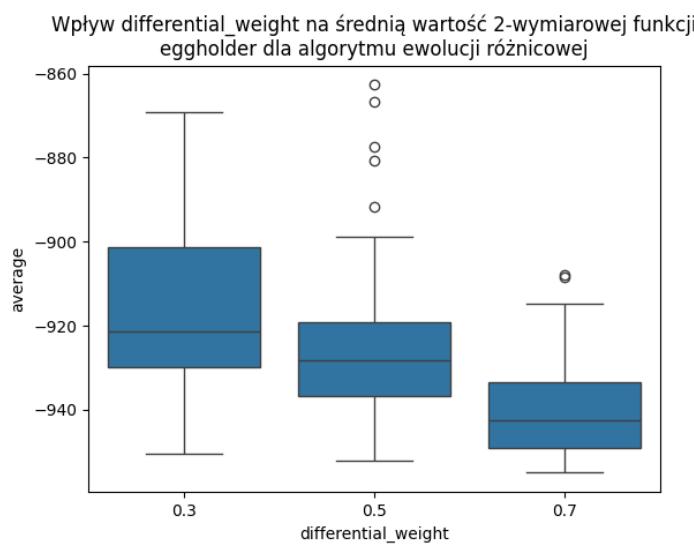
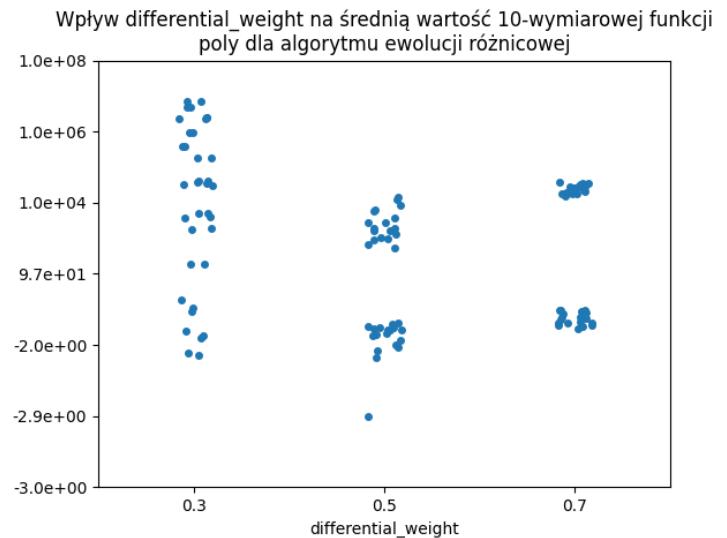
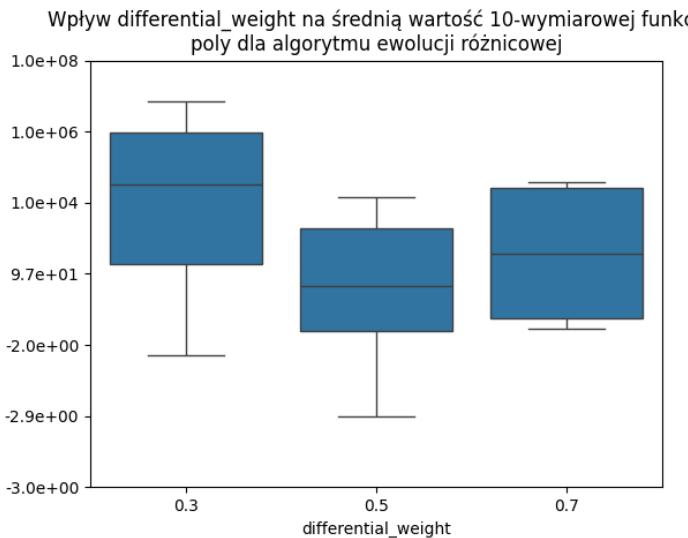
Wpływ crossover_threshold na średnią wartość 2-wymiarowej funkcji eggholder dla algorytmu ewolucji różnicowej



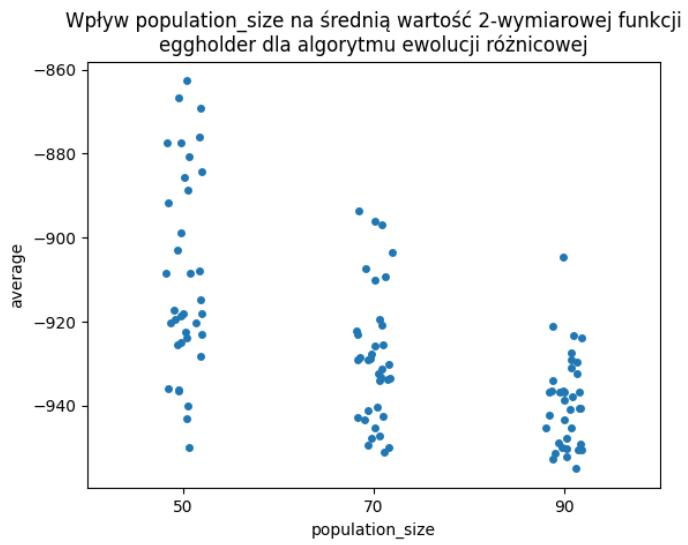
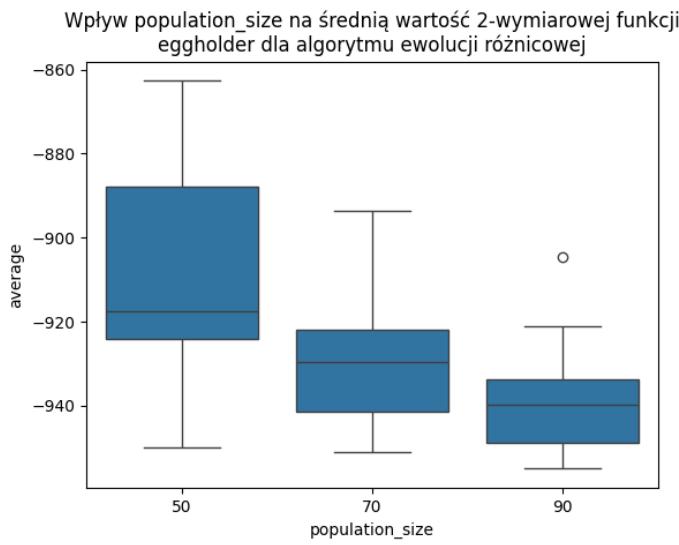
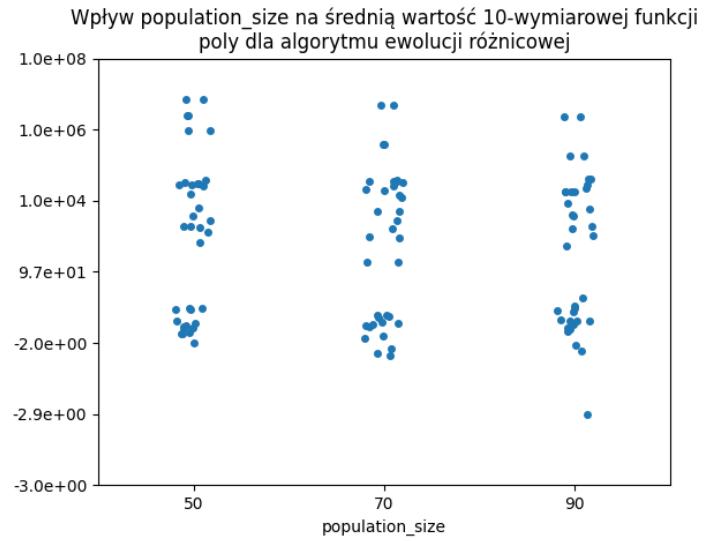
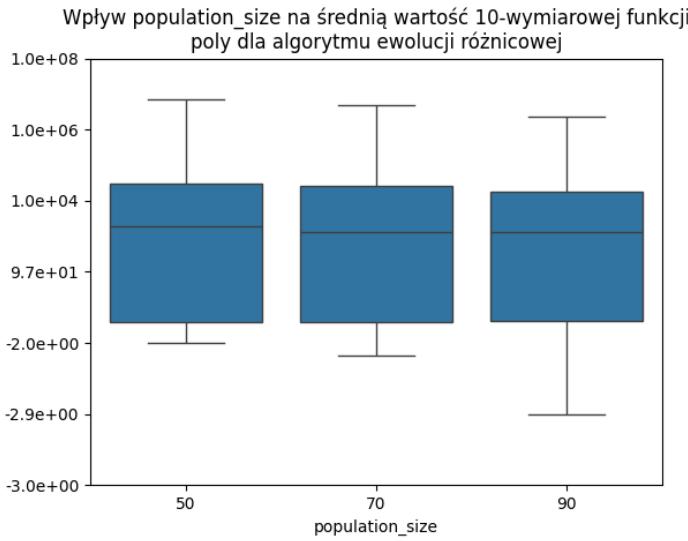
Wpływ crossover_threshold na średnią wartość 2-wymiarowej funkcji eggholder dla algorytmu ewolucji różnicowej



Można zauważyć, że niższe prawdopodobieństwo krzyżowania średnio wpływa pozytywnie na wyniki algorytmu ewolucji różnicowej. Należy jednak zaznaczyć, że wyższe wartości tego parametru dla najlepszych kombinacji parametrów przy funkcji wielomianowej są w stanie osiągnąć lepsze wyniki.

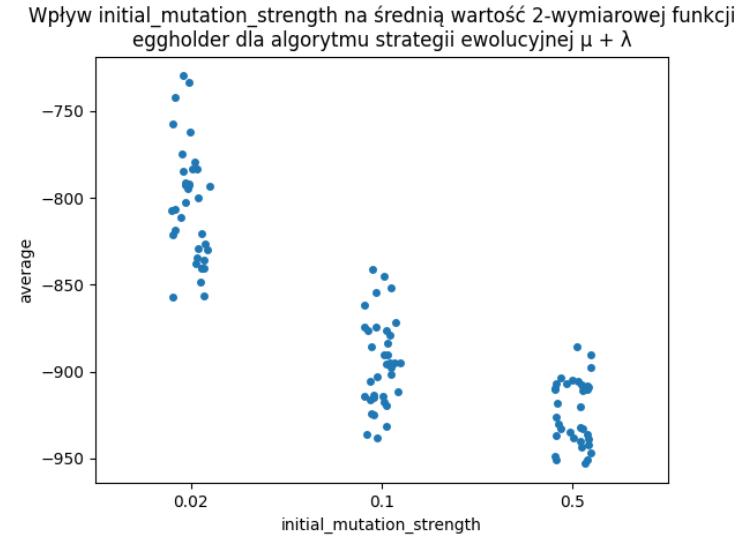
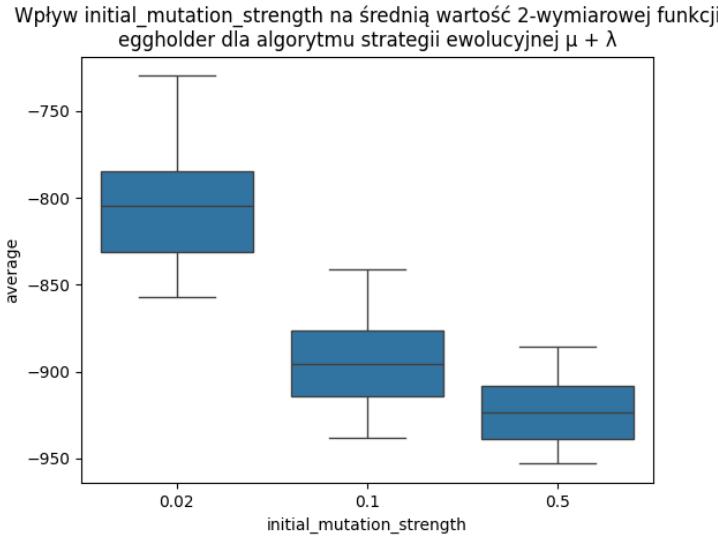
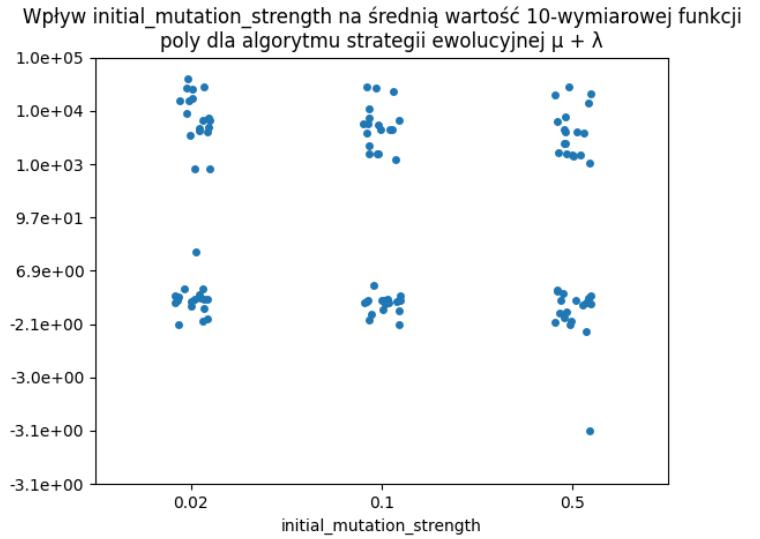
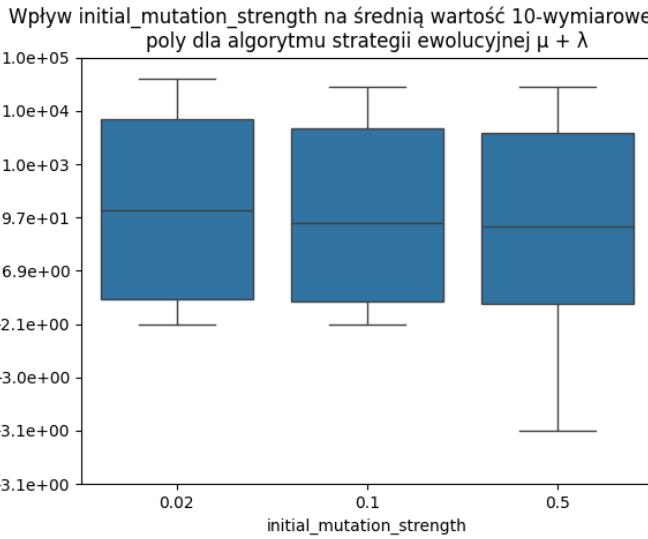


Dla współczynnika wagi różnicowej najlepsze wyniki są osiągane przy wartości 0.5 dla 10-wymiarowej funkcji wielomianowej. Dla 2-wymiarowej funkcji Eggholder z kolei zdaje się on być lepszy dla wyższych wartości. Wygląda na to, że funkcja Eggholder, której minima lokalne są rozsiane po całej dziedzinie funkcji, wymaga większej siły mutacji niż funkcja wielomianowa, której interesujący obszar jest położony stosunkowo blisko zera.



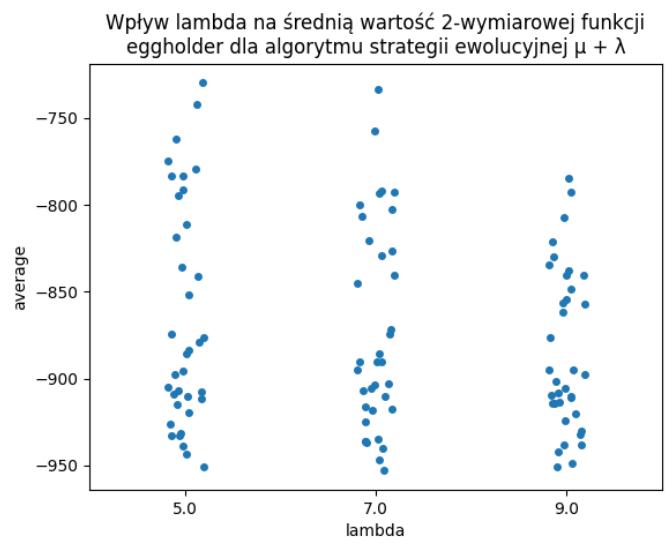
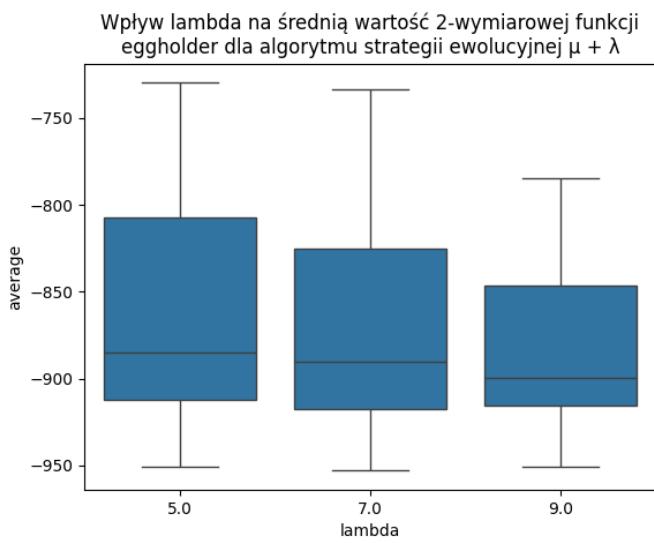
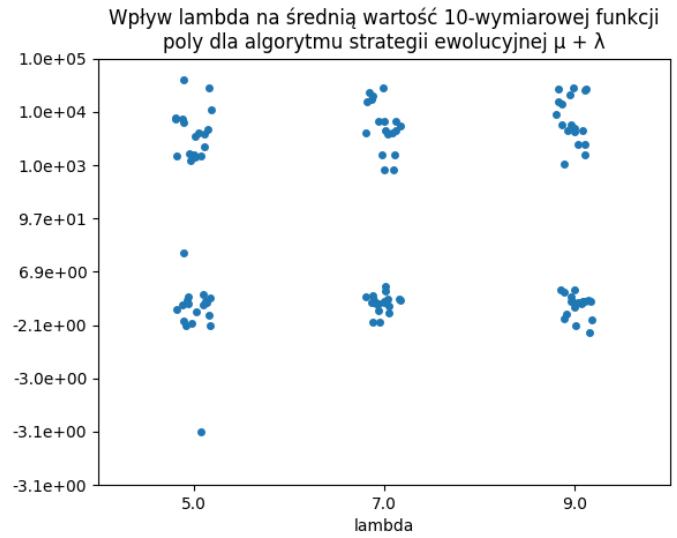
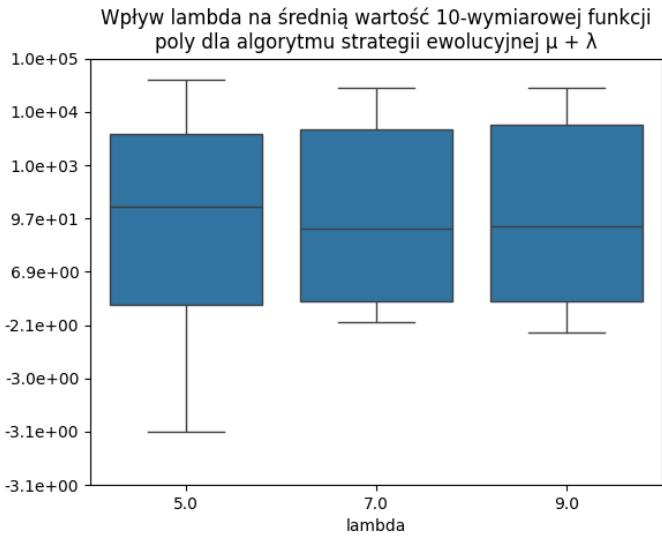
Rozmiar populacji zdaje się nie mieć znaczącego wpływu na działanie algorytmu ewolucji różnicowej dla funkcji wielomianowej, chociaż większe rozmiary populacji osiągają delikatnie lepsze wyniki. Z kolei dla funkcji Eggholder widać faktyczną poprawę wraz ze wzrostem rozmiaru populacji. Warto wspomnieć, że większa populacja oznacza więcej obliczeń funkcji celu na iterację, a zatem mniej iteracji (bo liczba wywołań funkcji jest ograniczona stałą wartością).

6.1.2 Parametry algorytmu strategii ewolucyjnej

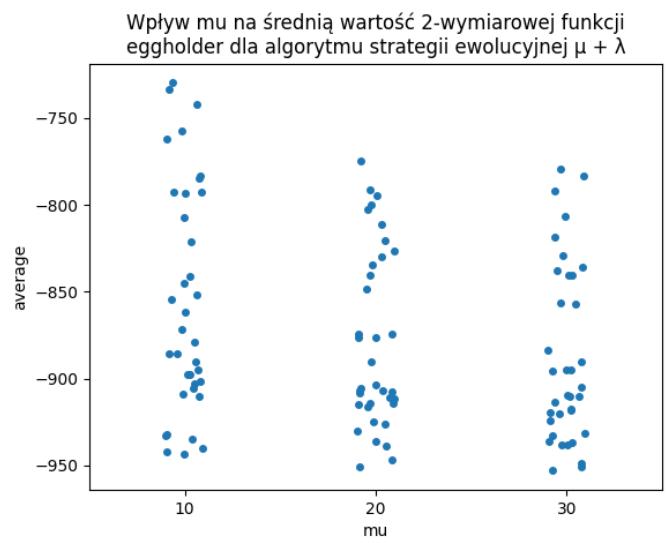
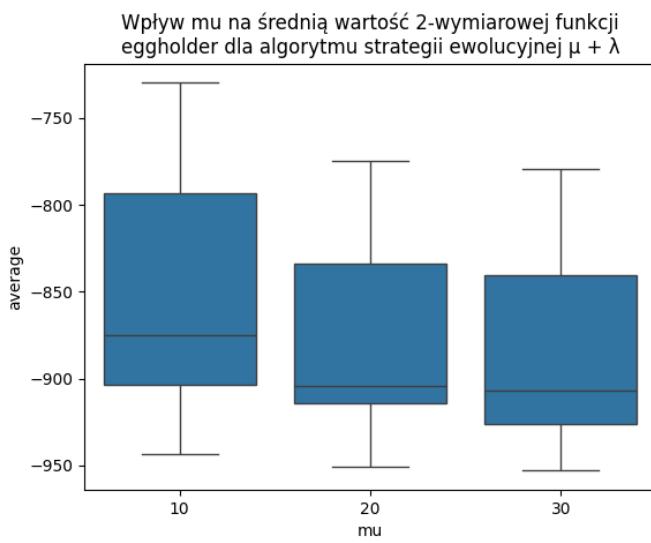
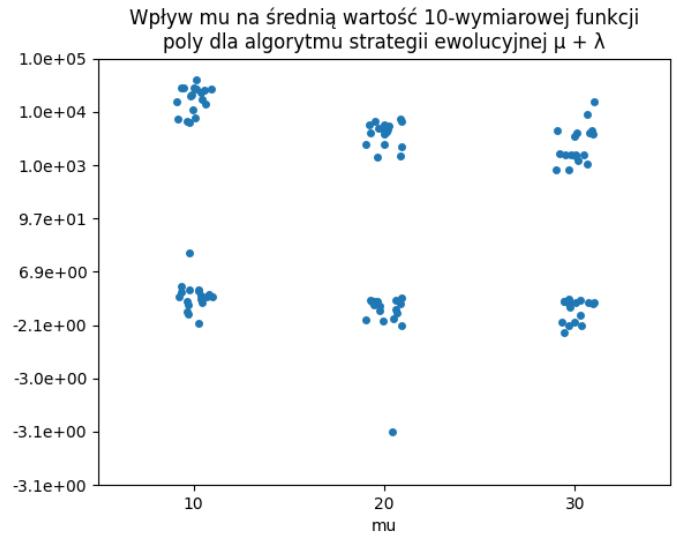
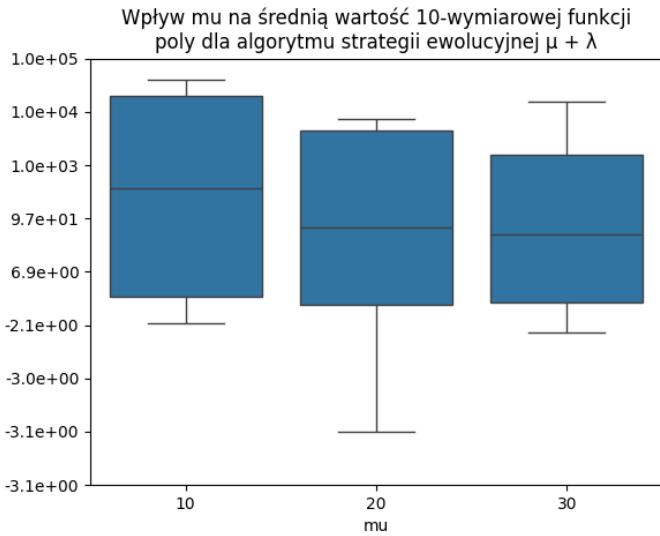


Początkowa siła mutacji na osi x wykresów jest wyrażona jako ułamek szerokości hipersześciianu, z którego były losowane punkty populacji początkowej.

Wybór początkowej siły mutacji spośród sprawdzanych wartości wydaje się nie mieć żadnego wpływu na wyniki algorytmu strategii ewolucyjnej $\mu + \lambda$ dla funkcji wielomianowej, chociaż dla dużej siły mutacji wystąpiły kombinacje hiperparametrów z lepszym wynikiem. Dla funkcji Eggholder widać znaczącą poprawę dla większych wartości początkowej siły mutacji. Wynik jest analogiczny jak dla parametru wagi różnicowej algorytmu DE - funkcja Eggholder, której minima lokalne są rozsiane po całej dziedzinie funkcji, wymaga większej siły mutacji niż funkcja wielomianowa.



Wybór rozmiaru populacji potomnej (wyrażany jako wielokrotność rozmiaru populacji początkowej) nie wpływa w znaczącym stopniu na wyniki algorytmu. Dla większych populacji potomnych liczba iteracji algorytmu jest mniejsza, a jednak osiągnięto podobne wyniki. Oznacza to, że przy stałej liczbie iteracji zwiększenie rozmiaru populacji potomnej zapewne poprawiłoby wyniki algorytmu.



Większy rozmiar początkowej populacji wydaje się nie mieć znaczącego wpływu na działanie algorytmu. Większa populacja oznacza więcej obliczeń funkcji celu na iterację, a zatem mniej iteracji, jednakże wynik algorytmu pozostaje podobny.

Nie opisujemy szerzej optymalizacji parametrów dla pozostałych funkcji celu, żeby uniknąć zamieszczania zbyt dużej liczby wykresów, z których wynika niewiele ponad to, co zostało już zaprezentowane.

6.2 Porównanie obu algorytmów

Dla wyznaczonych w poprzednim kroku najlepszych kombinacji parametrów dla obu algorytmów przeprowadziliśmy porównanie jakości obu algorytmów.

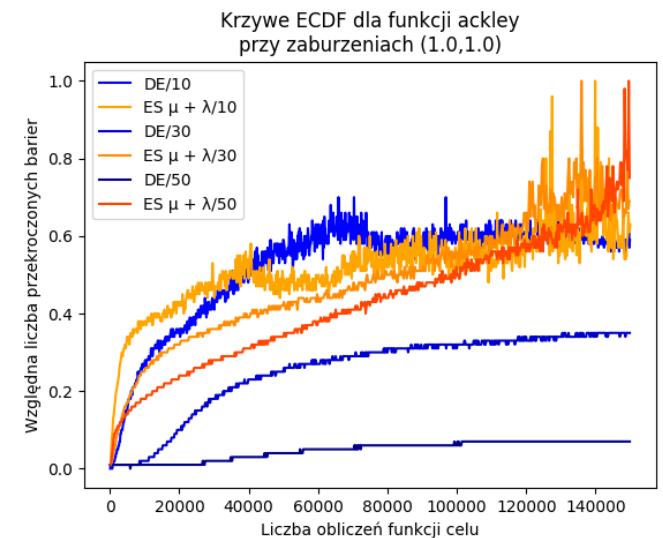
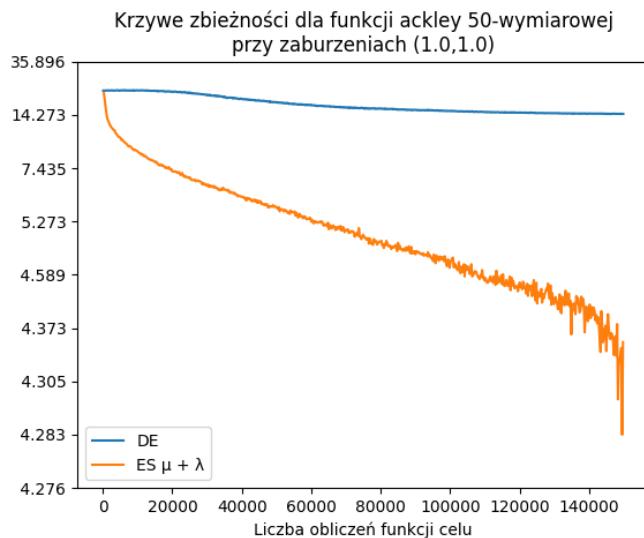
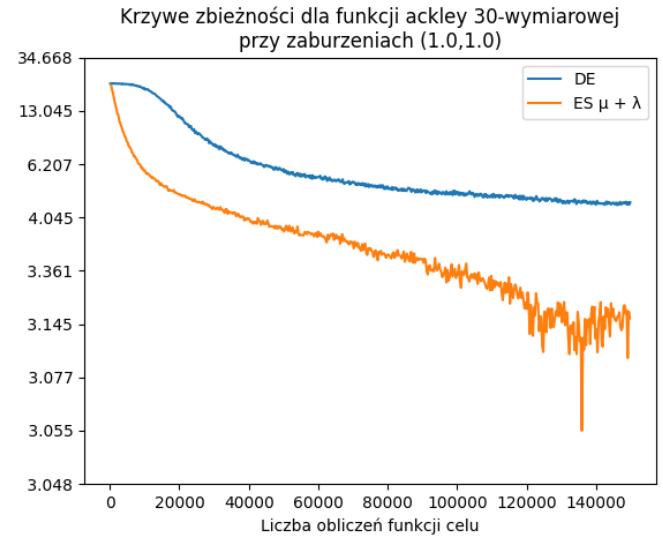
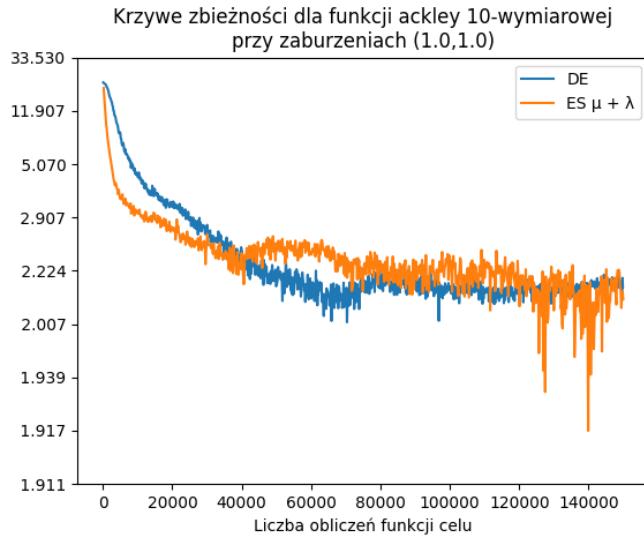
Wyniki algorytmów są przedstawione na krzywych zbieżności oraz krzywych ECDF. W przypadku deterministycznych funkcji celu, nie może wystąpić pogorszenie wyniku algorytmu z iteracji na iterację, jednakże testowane przez nas funkcje zawierają losowość, zatem wykresy nie są monotoniczne. Wyświetlany na wykresie jest wynik najlepszego (wybranego zgodnie z zaburzonymi ocenami wewnątrz algorytmu) osobnika w populacji w danej iteracji, liczony niezaburzoną, deterministyczną funkcją. Algorytm nie ma dostępu do niezaburzonej wersji funkcji celu - ta ewaluacja jest wykonywana na zewnątrz algorytmu.

Krzywe zbieżności są uśredniane z 25 uruchomień algorytmu (ziarna od 0 do 24 włącznie). Krzywe ECDF są liczone na podstawie uśrednionych krzywych zbieżności.

Oś y na wykresach krzywych zbieżności jest w skali logarytmicznej. Bariery dla krzywych ECDF również są ustalane w skali logarytmicznej, opierając się na minimalnym i maksymalnym wyniku spośród obu algorytmów dla danego zadania.

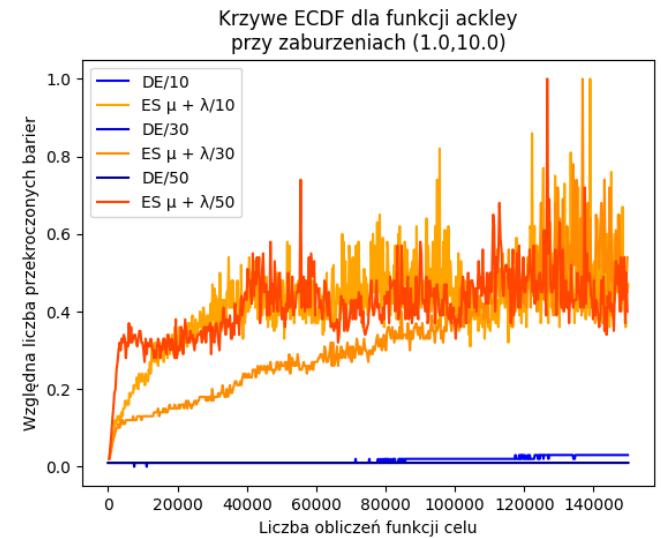
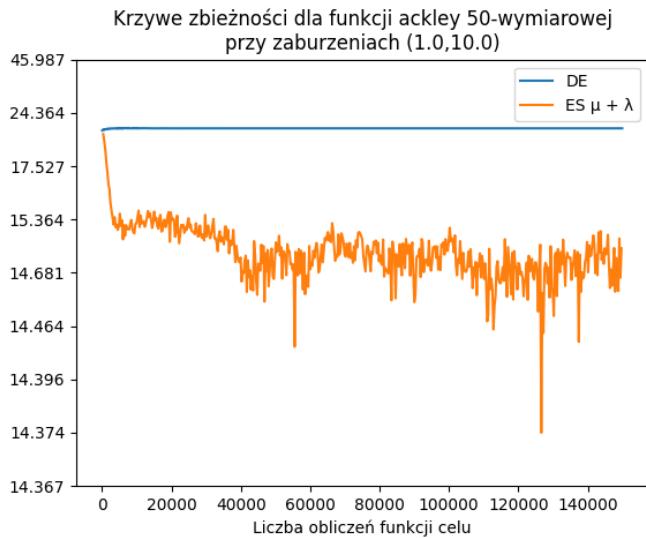
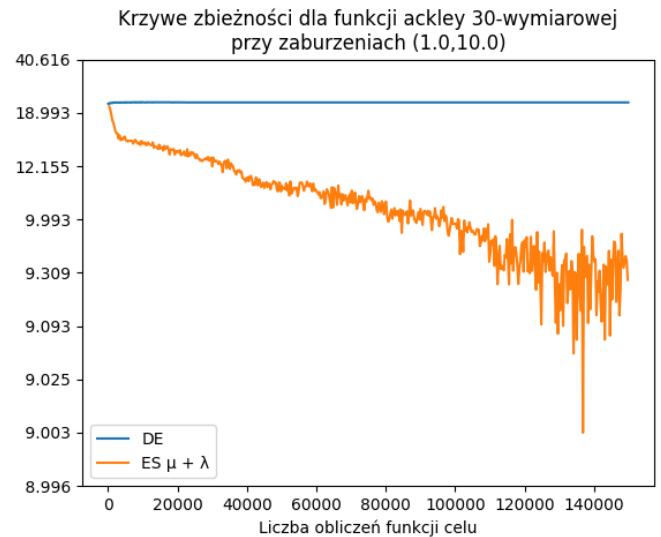
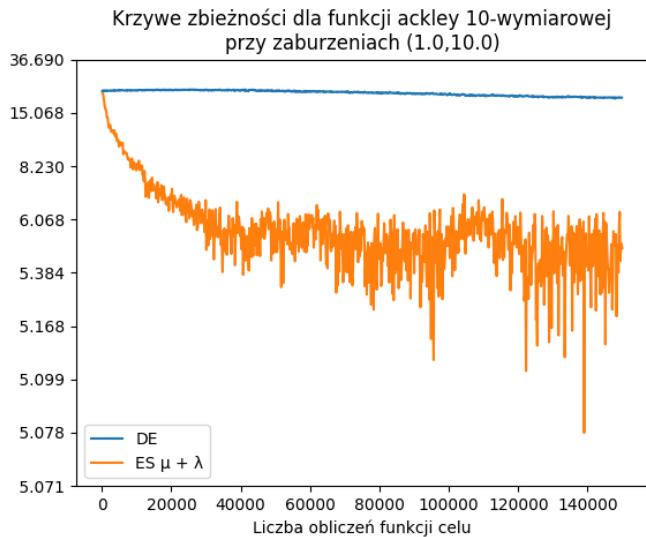
6.2.1 Funkcja Ackley'a

Testy dla małych zaburzeń losowych ($\sigma_1 = 1$ oraz $\sigma_2 = 1$)



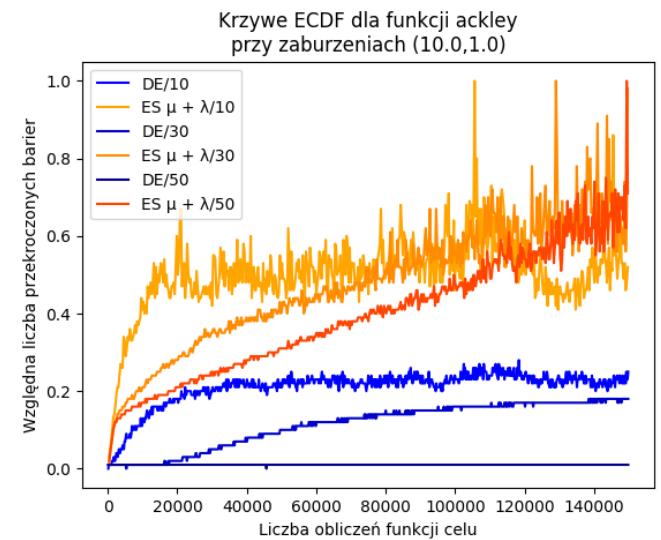
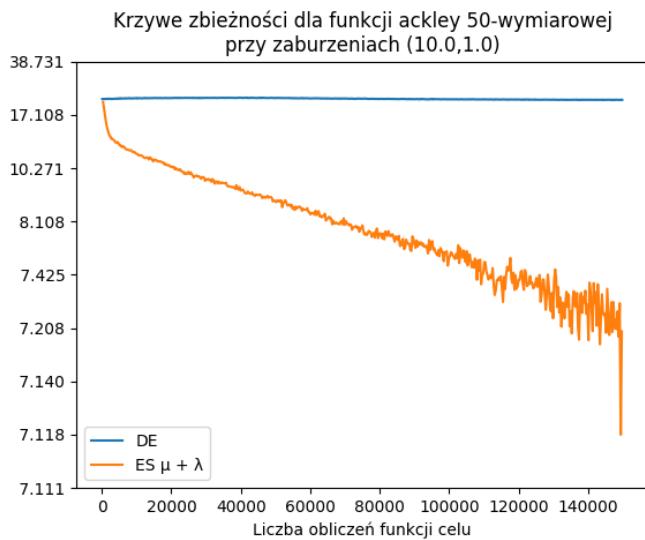
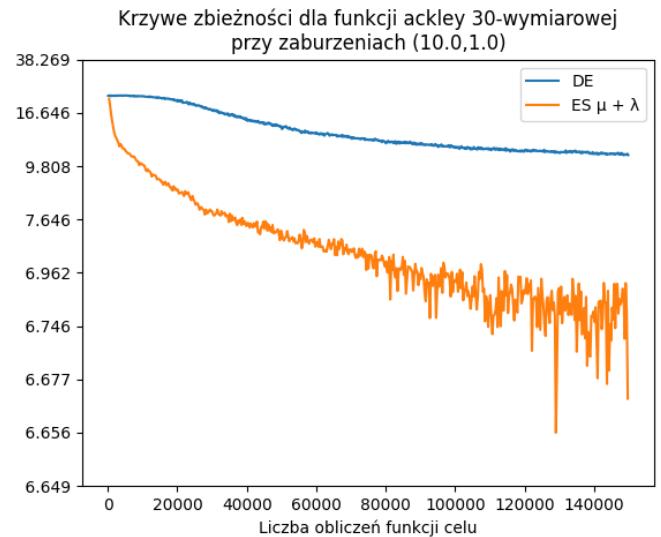
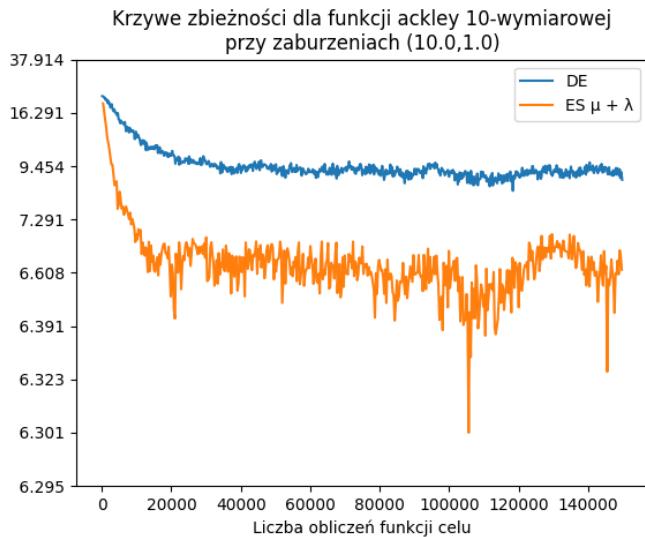
Im większa wymiarowość, tym lepsze okazały się być wyniki algorytmu $ES \mu + \lambda$ w porównaniu do DE. Dla 30 i 50-wymiarowych funkcji algorytm ES okazał się zdecydowanie lepszy. Dla wariantu 10-wymiarowego algorytm ES zbiegał szybciej, lecz ostatecznie oba uzyskały podobny wynik.

Testy dla dużych zaburzeń losowych wartości ($\sigma_1 = 1$ oraz $\sigma_2 = 10$)



Duże zaburzenia wartości uniemożliwiły algorytmowi ES zbliżenie się do minimum globalnego funkcji Ackley'a, ale jak widać algorytm strategii ewolucyjnej $\mu + \lambda$ poradził sobie z nimi znacznie lepiej. Widoczny na osi y wykresów jest fakt, że wraz z rosnącą wymiarowością i siłą losowości wyniki algorytmu ES znacząco się pogarszają.

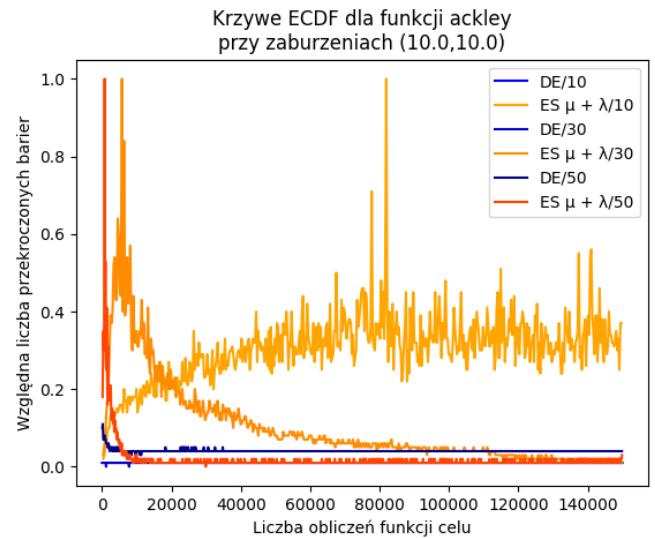
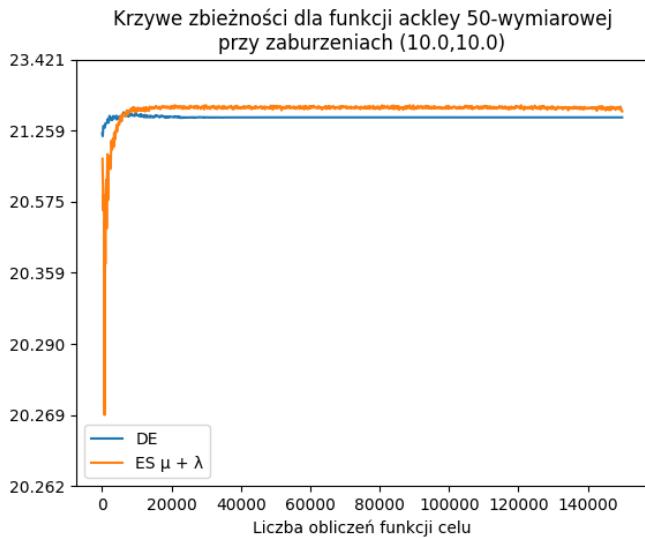
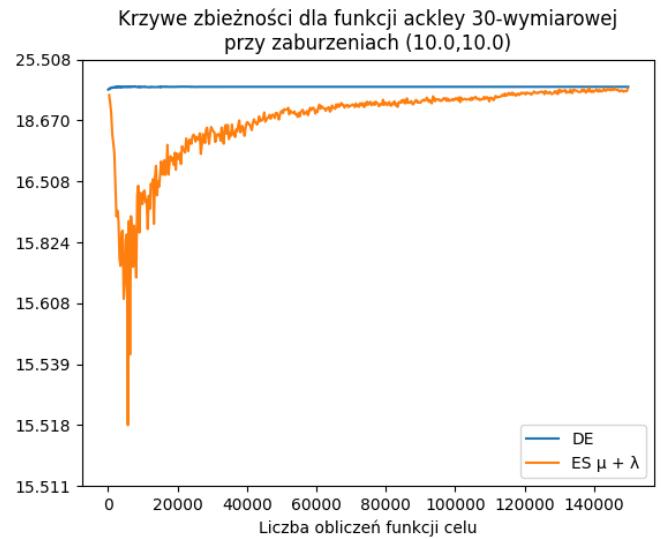
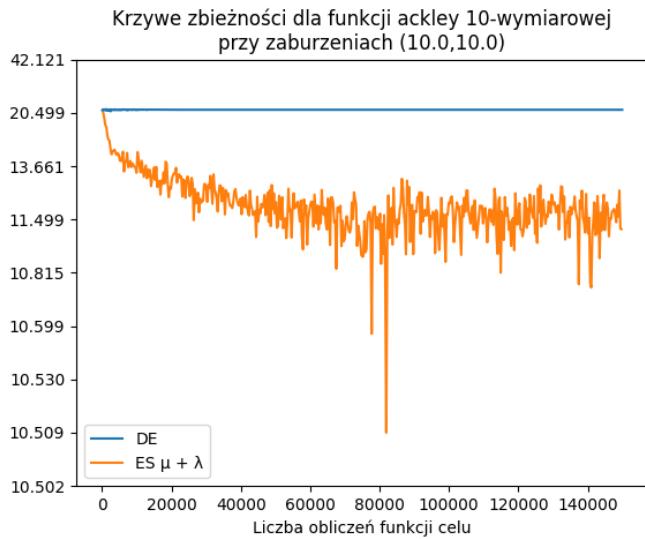
Testy dla dużych zaburzeń losowych argumentu ($\sigma_1 = 10$ oraz $\sigma_2 = 1$)



Przy dużych zaburzeniach losowych argumentu sytuacja jest podobna jak przy silniejszych zaburzeniach losowych wartości, jednakże wyniki obu algorytmów okazały się być nieco lepsze w tym przypadku.

Widoczne jest też zjawisko, że algorytmowi ES zbiegnięcie do minimum zajmuje więcej wywołań funkcji celu dla większych wymiarowości i silniejszej losowości.

Testy dla dużych zaburzeń wartości i argumentu ($\sigma_1 = 10$ oraz $\sigma_2 = 10$)



Gdy silnym zaburzeniom podlegają zarówno argument, jak i wartość, dalej widoczna jest przewaga algorytmu ES $\mu + \lambda$ w wariancie 10-wymiarowym, jednakże przy wyższych wymiarowościach oba algorytmy straciły możliwość lokalizowania optimum globalnego. W obu przypadkach algorytm ES znalazł punkty nieco bliżej optimum globalnego, jednakże losowe zaburzenia spowodowały utratę tych dobrych osobników.

Wyniki algorytmu ewolucji różnicowej dla funkcji Ackley'a w formie tabelarycznej:

		średnia			odch. standardowe			min			max			
		wym.	10	30	50	10	30	50	10	30	50	10	30	50
σ_1	σ_2													
1	1		2,13	4,43	14,48	1,13	1,80	6,86	0,35	2,65	5,59	4,25	9,48	21,91
	10		18,95	21,50	21,52	5,96	0,21	0,18	4,44	21,09	21,12	22,09	21,80	21,83
10	1		8,65	10,66	20,92	2,77	2,67	2,35	4,40	6,05	13,09	13,47	16,27	21,93
	10		21,18	21,47	21,50	0,65	0,21	0,20	18,92	21,01	21,03	21,88	21,84	21,85

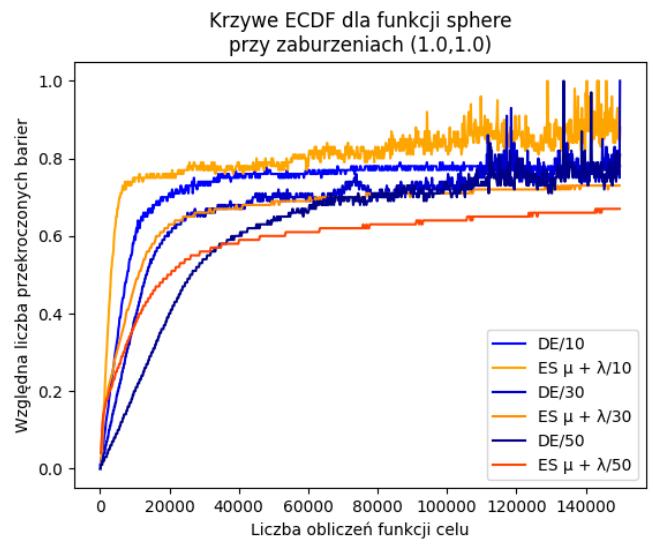
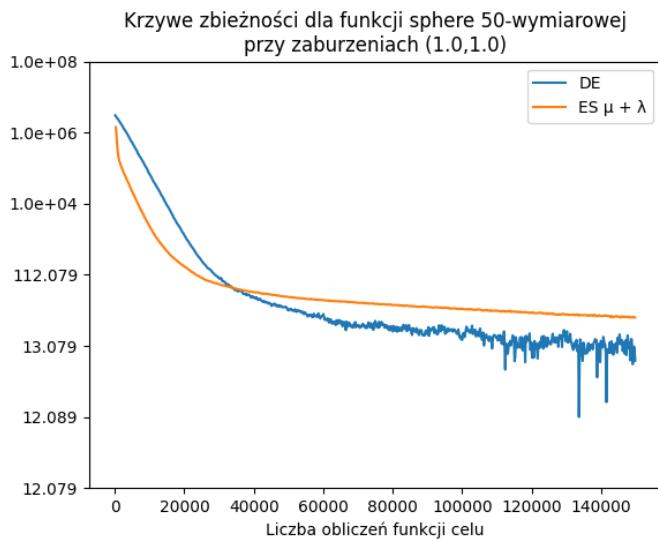
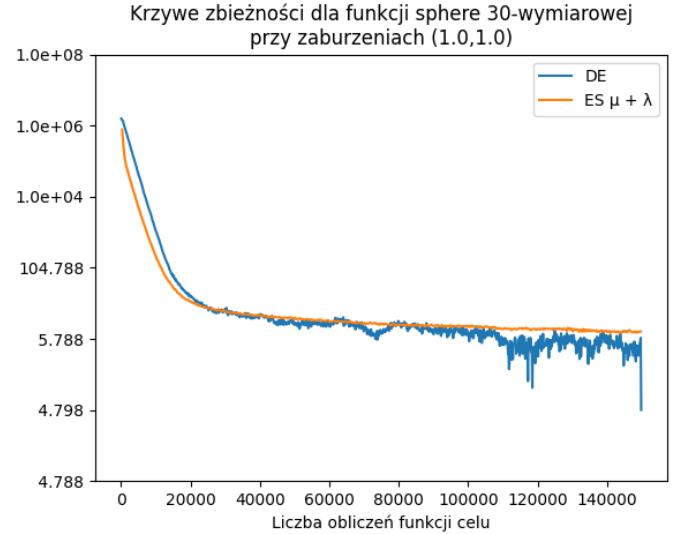
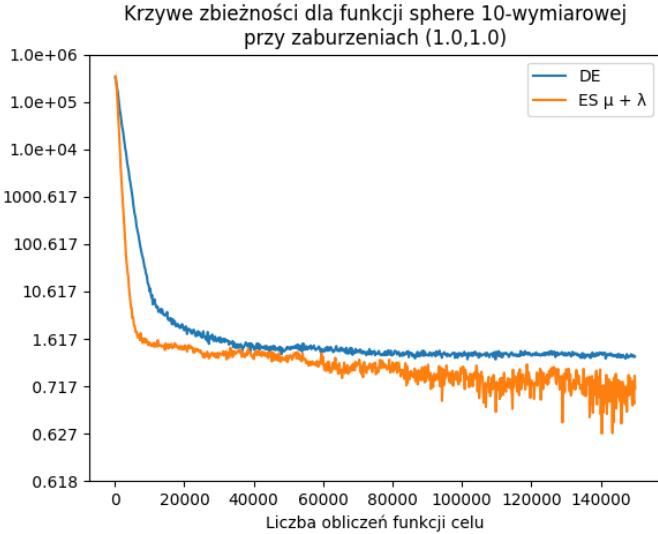
Wyniki algorytmu strategii ewolucyjnej dla funkcji Ackley'a w formie tabelarycznej:

		średnia			odch. standardowe			min			max			
		wym.	10	30	50	10	30	50	10	30	50	10	30	50
σ_1	σ_2													
1	1		2,08	3,16	4,35	0,93	0,73	0,70	0,72	1,39	3,43	4,20	4,35	6,53
	10		5,60	9,26	14,90	1,15	1,05	2,94	3,80	7,37	10,75	7,86	11,53	21,74
10	1		6,63	6,67	7,20	2,68	1,41	0,68	3,40	4,76	6,02	12,67	9,81	8,52
	10		11,31	21,27	21,62	3,10	1,44	0,17	6,59	14,77	21,27	21,64	21,97	21,88

Jak widać, wyniki pogarszają się wraz ze wzrostem wymiarowości oraz siły losowości. W większości przypadków zaburzenie wartości pogorszyło wyniki bardziej niż zaburzenie argumentu - dla funkcji Ackley'a, która przyjmuje wartości z przedziału $[0, 25]$, zaburzenie argumentu zazwyczaj nie zmieni wyniku bardziej niż analogiczne zaburzenie wartości.

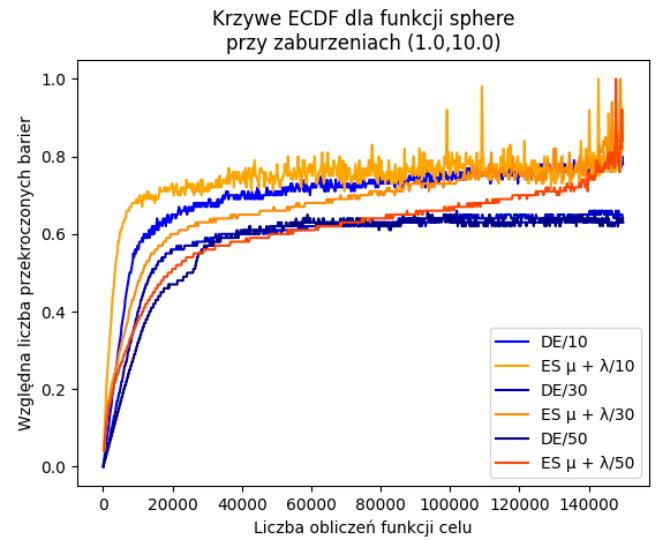
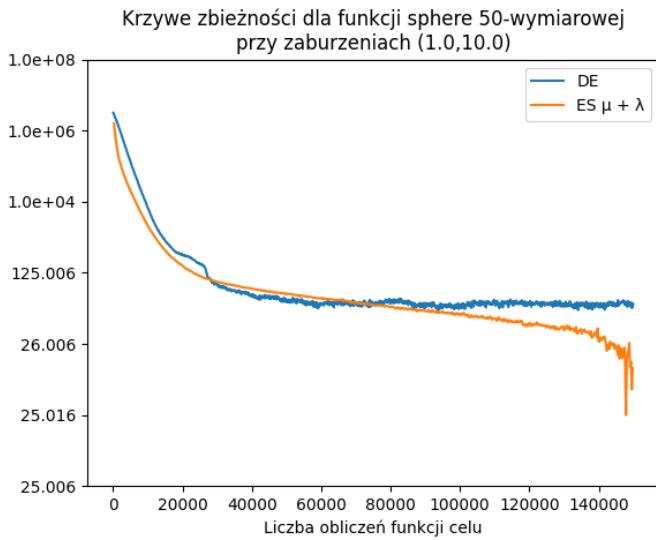
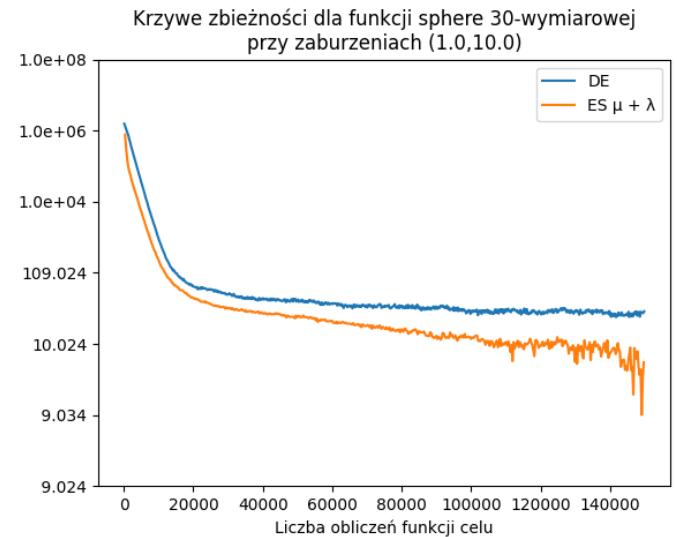
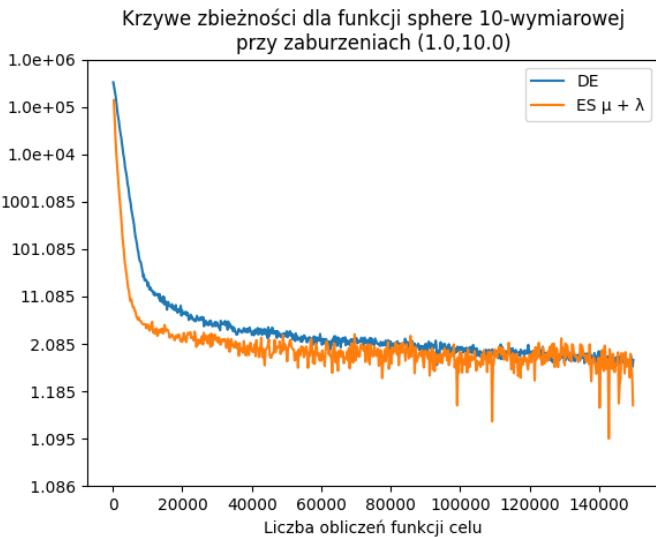
6.2.2 Wielowymiarowa funkcja kwadratowa

Testy dla małych zaburzeń losowych ($\sigma_1 = 1$ oraz $\sigma_2 = 1$)



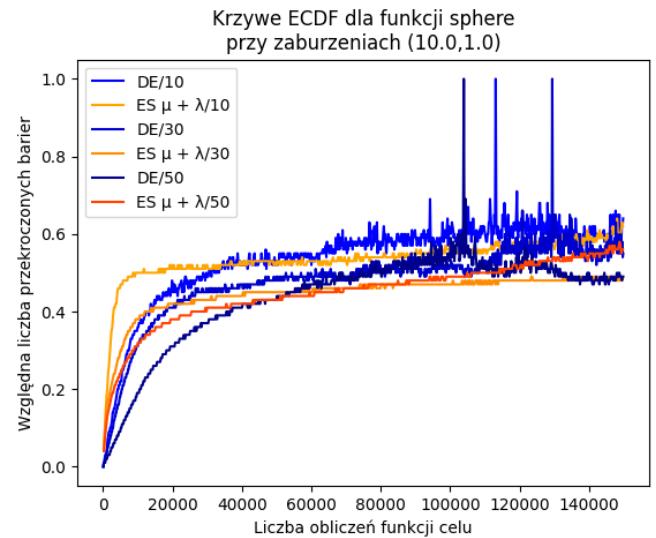
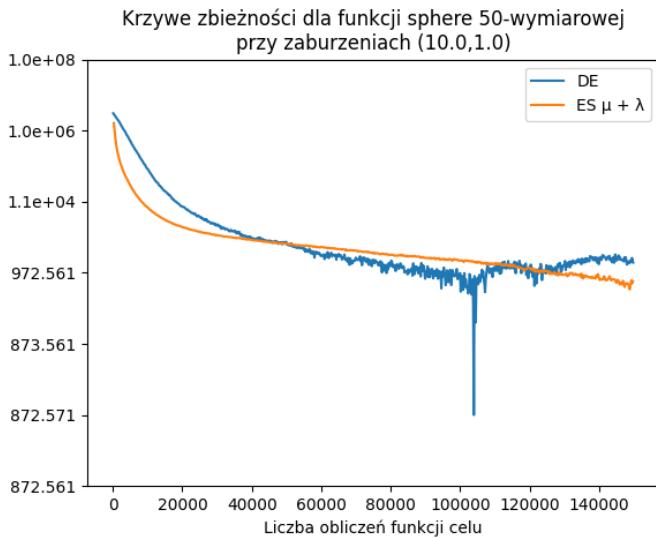
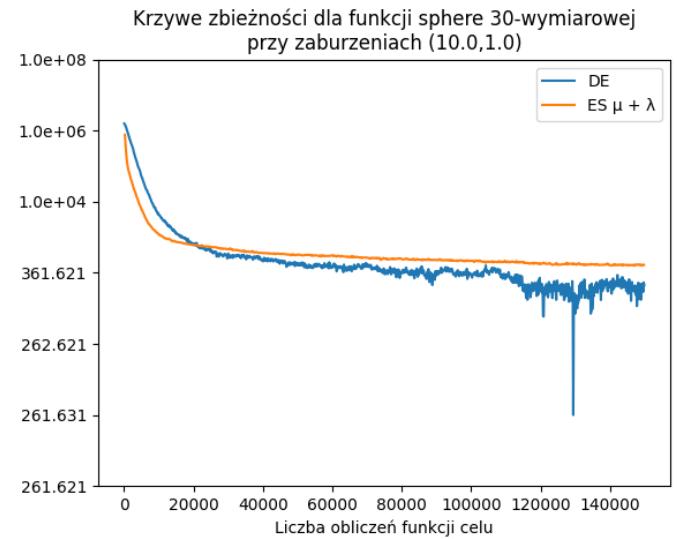
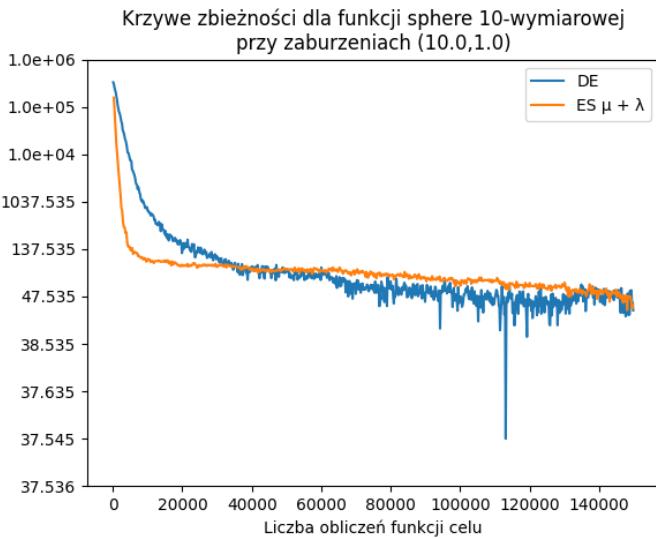
Jak widać na wykresach, algorytm ES początkowo zbiegał nieco szybciej, jednak dla większych wymiarów to algorytm DE osiągnął nieco lepsze przybliżenie minimum.

Testy dla dużych zaburzeń losowych wartości ($\sigma_1 = 1$ oraz $\sigma_2 = 10$)



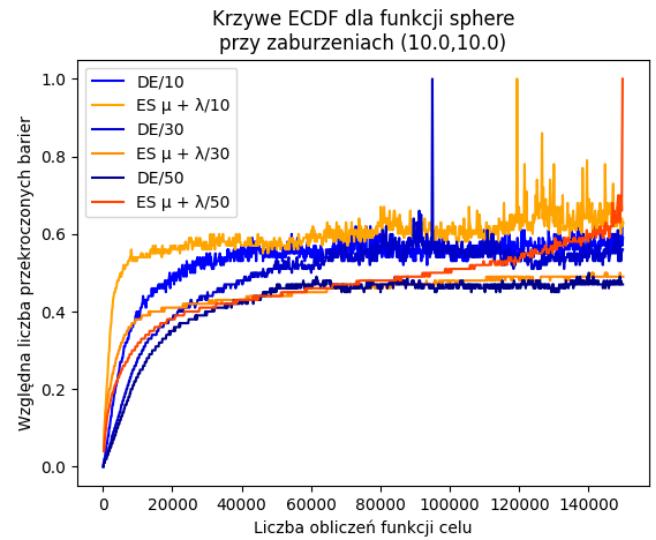
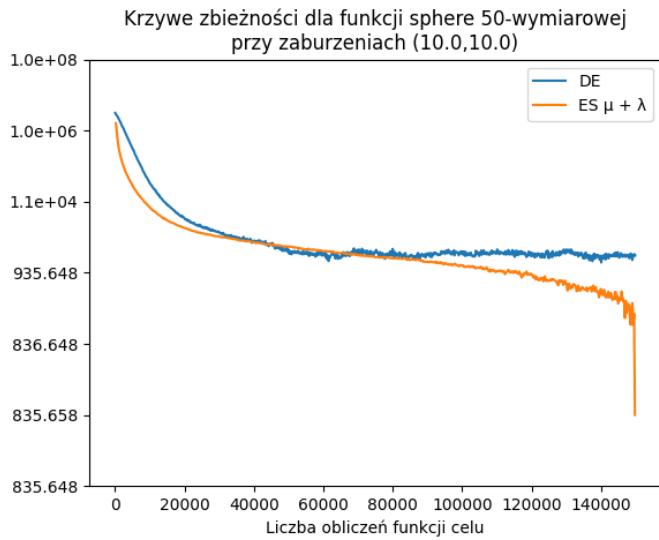
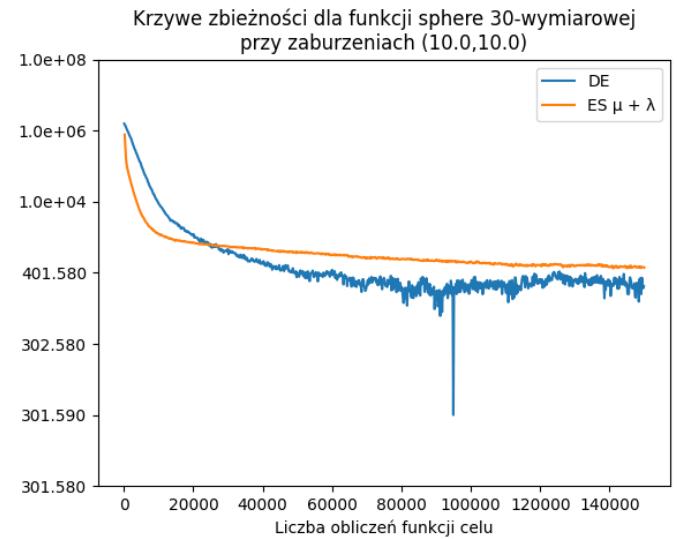
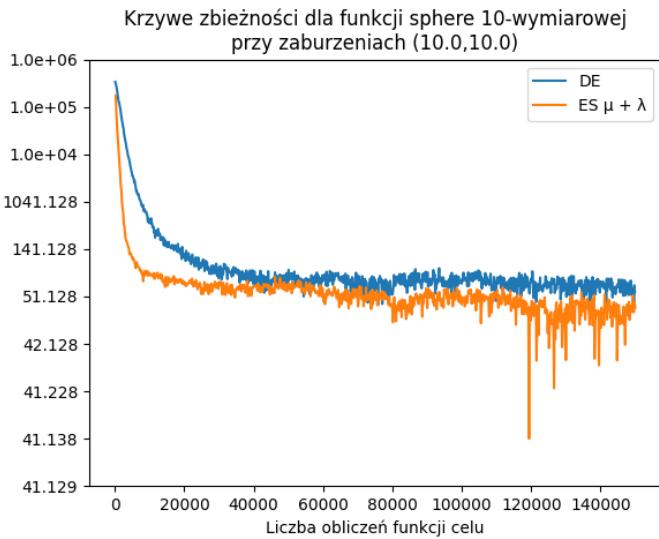
Wygląda na to, że zaburzenia wartości przeszkodziły bardziej algorytmowi ewolucji różnicowej - dla małych zaburzeń losowych uzyskiwał on nieco lepsze wyniki, tutaj dla wariantów o wyższych wymiarowościach uzyskuje on wyniki znacznie gorsze.

Testy dla dużych zaburzeń losowych argumentu ($\sigma_1 = 10$ oraz $\sigma_2 = 1$)



W przypadku zaburzeń losowych argumentu oba algorytmy zachowują się podobnie. Największą różnicą jest drastyczne pogorszenie wyników - jest to spowodowane naturą funkcji kwadratowej, dla której duże zaburzenie argumentu powoduje bardzo duże zaburzenie wartości.

Testy dla dużych zaburzeń wartości i argumentu ($\sigma_1 = 10$ oraz $\sigma_2 = 10$)



W tym przypadku nie da się wyróżnić jednego algorytmu, który okazałby się lepszy - oba zatrzymywały się na podobnych wartościach funkcji celu; silna losowość uniemożliwiła dalszą zbieżność.

Wyniki algorytmu ewolucji różnicowej dla funkcji kwadratowej w formie tabelarycznej:

		średnia			odch. standardowe			min			max			
		wym.	10	30	50	10	30	50	10	30	50	10	30	50
σ_1	σ_2													
1	1		1,04	4,80	12,46	0,89	2,01	3,52	0,08	2,33	7,02	3,70	10,94	19,43
	10		1,54	17,23	37,90	1,00	7,23	10,05	0,42	9,16	23,94	4,21	35,52	62,18
10	1		42,62	306,44	1e+03	40,52	203,00	623,58	4,08	69,51	491,21	142,15	805,44	3e+03
	10		53,47	343,84	1e+03	39,28	236,02	750,47	10,65	118,57	497,66	172,52	1e+03	3e+03

Wyniki algorytmu strategii ewolucyjnej dla funkcji kwadratowej w formie tabelarycznej:

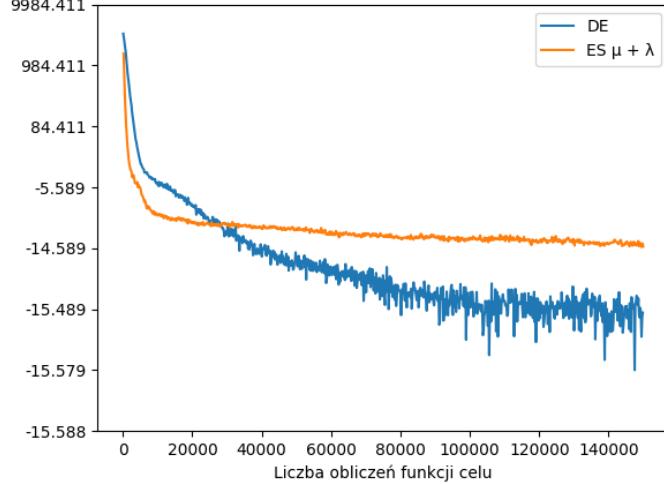
		średnia			odch. standardowe			min			max			
		wym.	10	30	50	10	30	50	10	30	50	10	30	50
σ_1	σ_2													
1	1		0,71	6,40	18,34	0,93	5,62	17,91	0,07	0,44	3,96	4,40	21,82	89,44
	10		1,14	9,33	25,21	0,41	6,07	18,22	0,37	2,42	7,70	2,28	22,85	97,32
10	1		44,22	428,13	929,97	46,81	563,93	807,49	2,61	44,97	161,25	201,08	3e+03	3e+03
	10		46,92	441,77	835,66	65,29	366,61	530,13	4,95	102,29	209,63	302,79	2e+03	2e+03

Jak można się spodziewać, wyniki pogarszają się wraz ze wzrostem wymiarowości oraz siły losowości. W przypadku funkcji kwadratowej zaburzenie argumentu miało znacznie większy wpływ na wynik niż zaburzenie wartości - dla funkcji kwadratowej, zaburzenie argumentu jest zazwyczaj równoważne znacznie większemu zaburzeniu wartości.

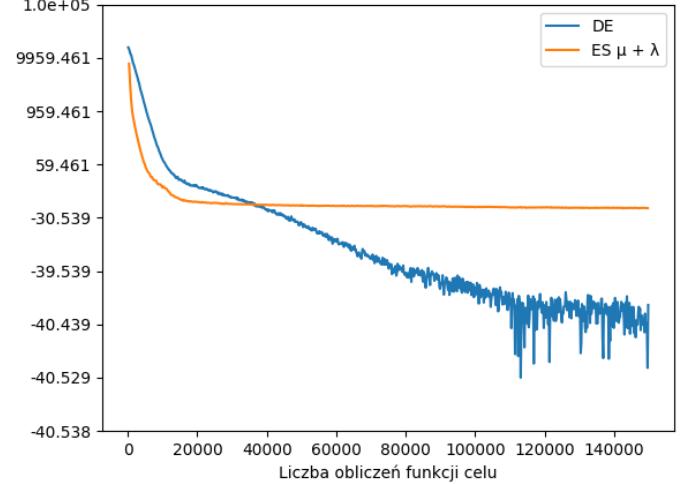
6.2.3 Zaburzona funkcja kwadratowa

Testy dla małych zaburzeń losowych ($\sigma_1 = 1$ oraz $\sigma_2 = 1$)

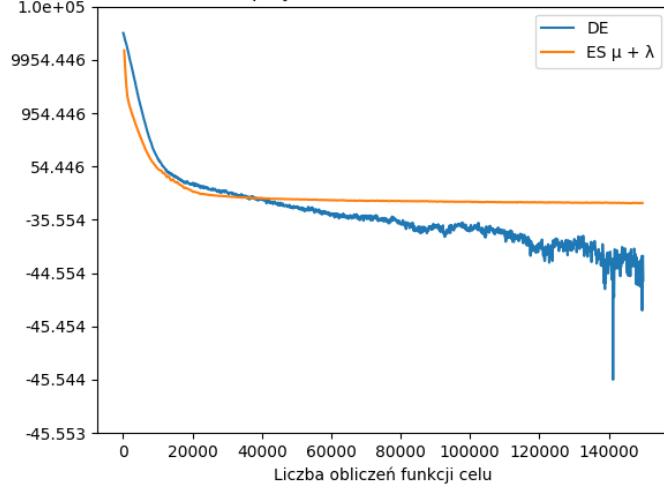
Krzywe zbieżności dla funkcji disturbed_square 10-wymiarowej
przy zaburzeniach (1.0,1.0)



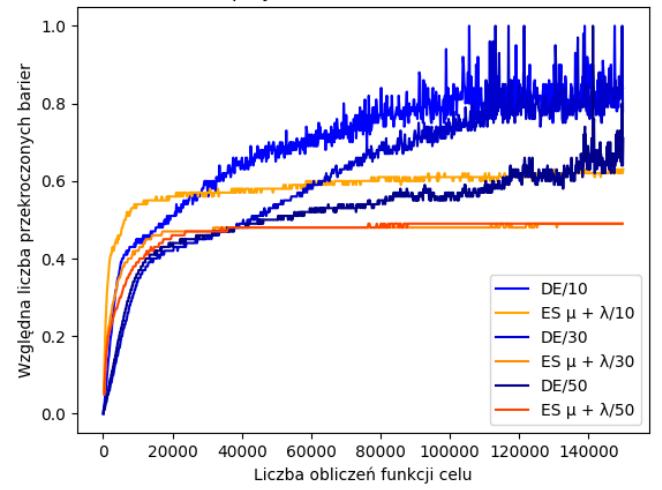
Krzywe zbieżności dla funkcji disturbed_square 30-wymiarowej
przy zaburzeniach (1.0,1.0)



Krzywe zbieżności dla funkcji disturbed_square 50-wymiarowej
przy zaburzeniach (1.0,1.0)

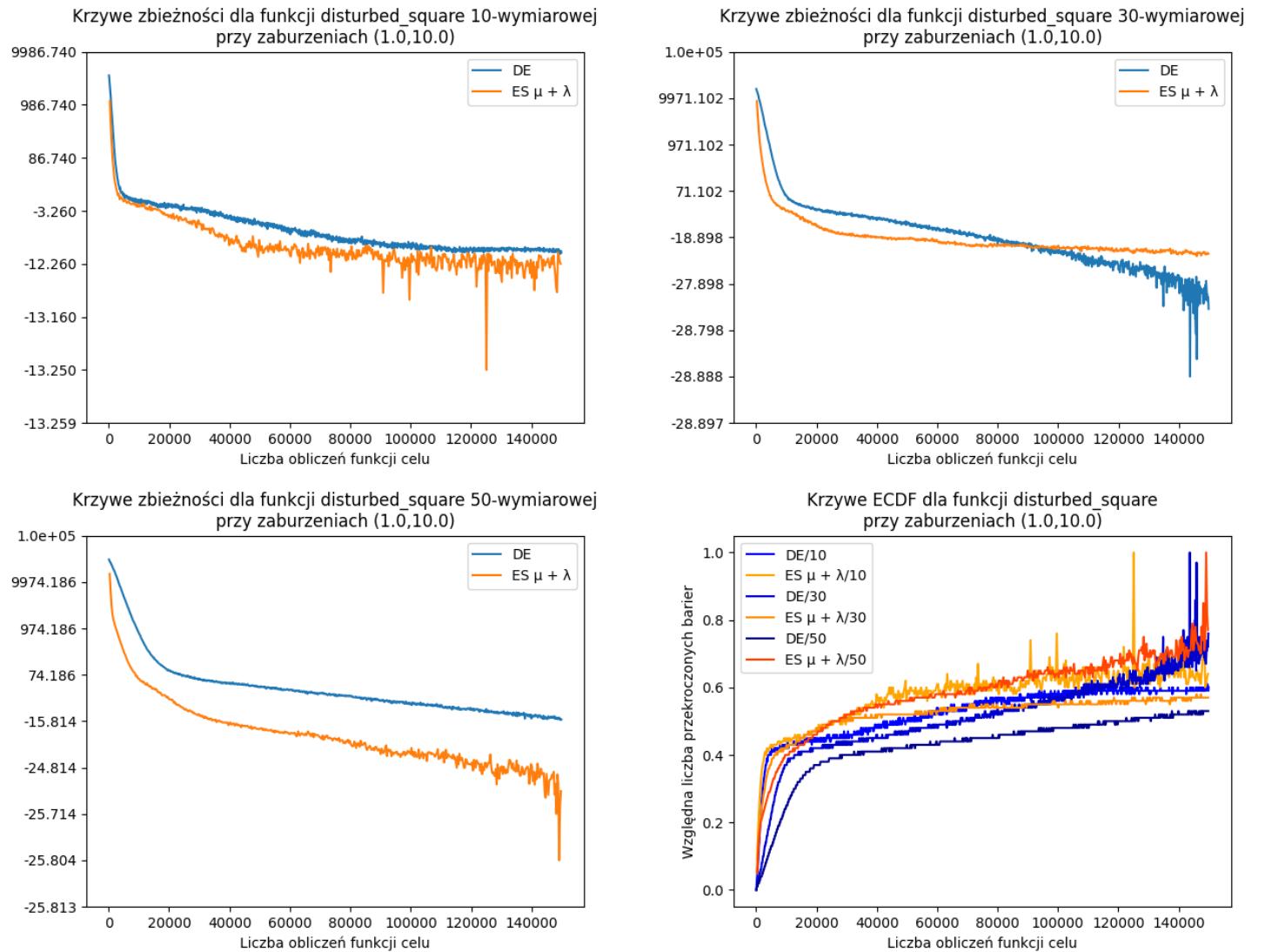


Krzywe ECDF dla funkcji disturbed_square
przy zaburzeniach (1.0,1.0)



Jak widać, dla tej funkcji celu algorytm ewolucji różnicowej poradził sobie zdecydowanie lepiej. Pomimo tego, że początkowo algorytm $ES \mu + \lambda$ zbiegał szybciej, to jednak wykazał większą tendencję do utykania w licznych minimach lokalnych funkcji.

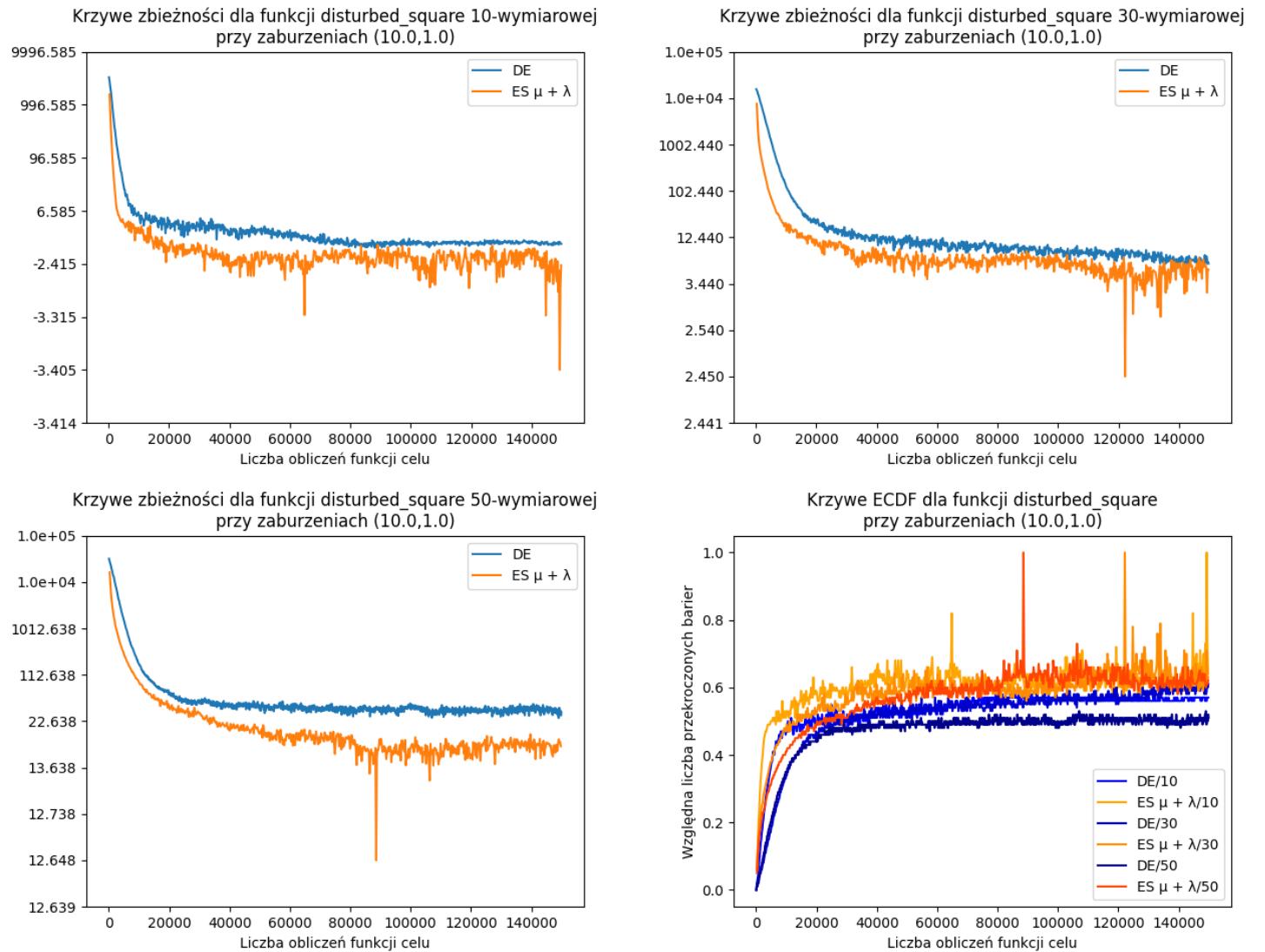
Testy dla dużych zaburzeń losowych wartości ($\sigma_1 = 1$ oraz $\sigma_2 = 10$)



Wyniki dla dużych zaburzeń wartości są niejasne - dla wariantu 30-wymiarowego lepsza jest ewolucja różnicowa, a dla wariantu 50-wymiarowego - strategia ewolucyjna. Warto zauważyć, że nawet dla wariantu 30-wymiarowego algorytm ES początkowo zbiegał szybciej i utrzymywał lepszy wynik przez ponad połowę czasu uruchomienia algorytmu.

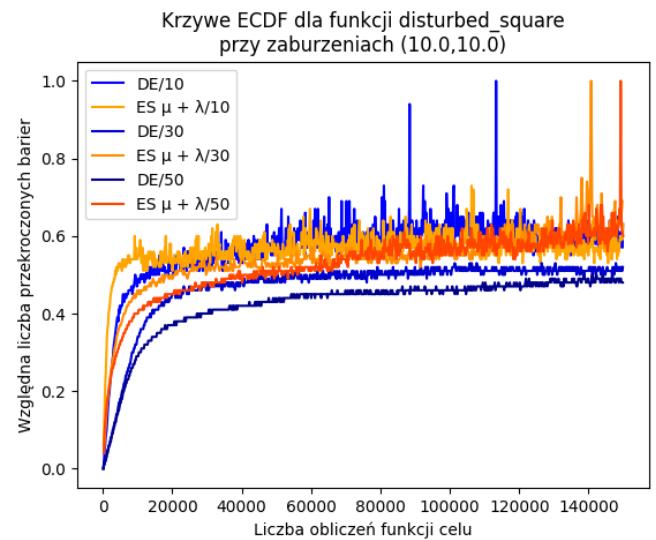
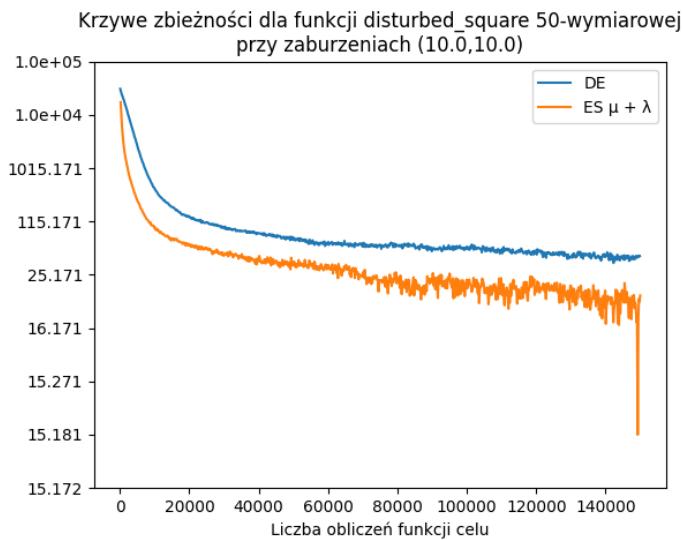
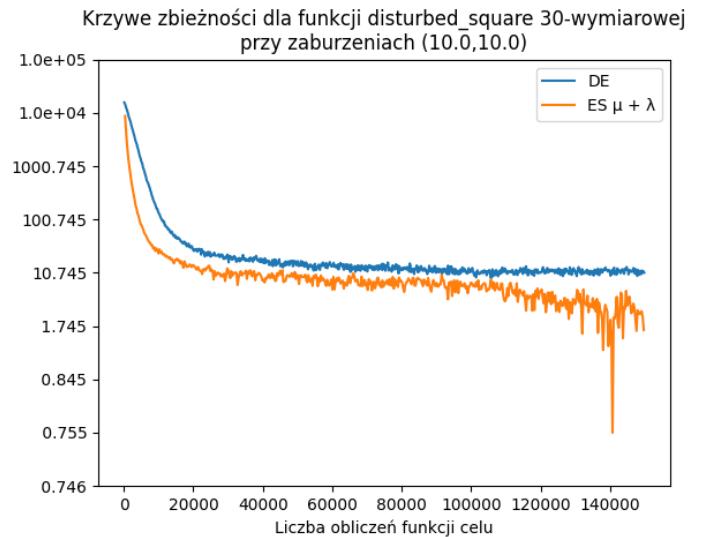
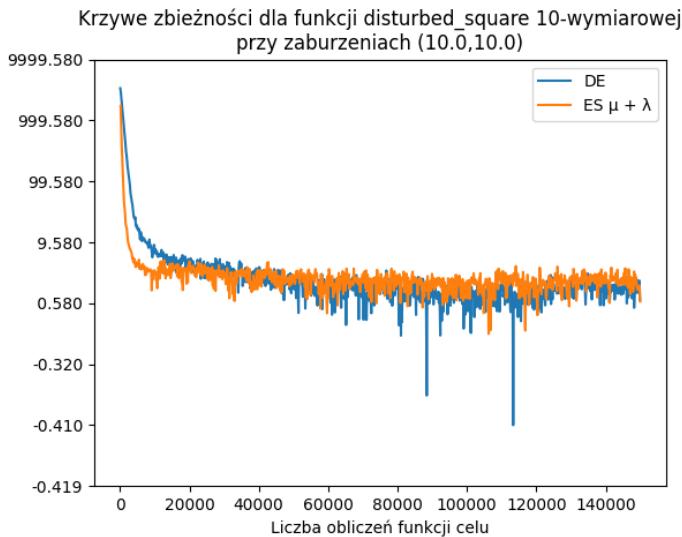
Zaburzenie wartości zmieniło relację wyników algorytmów - tym razem strategia ewolucyjna nie okazała się być bezdyskusyjnie gorsza. Najprawdopodobniej wynika to z faktu, że dość płytkie minima lokalne funkcji zanikają przy silniejszej losowości.

Testy dla dużych zaburzeń losowych argumentu ($\sigma_1 = 10$ oraz $\sigma_2 = 1$)



Dla dużych zaburzeń argumentu, dla wszystkich wymiarowości algorytmu strategia ewolucyjna uzyskała lepsze wyniki. Wygląda na to, że strategia ewolucyjna wykazuje większą odporność na losowe zaburzenia obecne w funkcji celu.

Testy dla dużych zaburzeń wartości i argumentu ($\sigma_1 = 10$ oraz $\sigma_2 = 10$)



Gdy silne zaburzenia losowe dotyczą zarówno argumentu, jak i wartości, strategia ewolucyjna utrzymała swoją przewagę. Wyjątkiem jest wariant 10-wymiarowy, gdzie losowość spowodowała podobne wyniki dla obu algorytmów.

Wyniki algorytmu ewolucji różnicowej dla funkcji zaburzonej kwadratowej w formie tabelarycznej:

		średnia			odch. standardowe			min			max		
wym.		10	30	50	10	30	50	10	30	50	10	30	50
σ_1	σ_2												
1	1	-15,50	-40,31	-44,83	1,11	7,27	6,42	-17,23	-48,15	-54,78	-13,24	-14,80	-24,51
	10	-11,62	-28,61	-14,92	2,15	8,20	33,92	-14,90	-40,44	-59,84	-6,16	-8,55	69,96
10	1	-1,03	5,17	28,27	9,22	26,09	9,02	-16,99	-34,43	15,77	14,97	48,10	43,77
	10	0,71	10,78	37,52	4,44	13,75	13,20	-11,24	-25,08	15,24	8,99	36,25	69,89

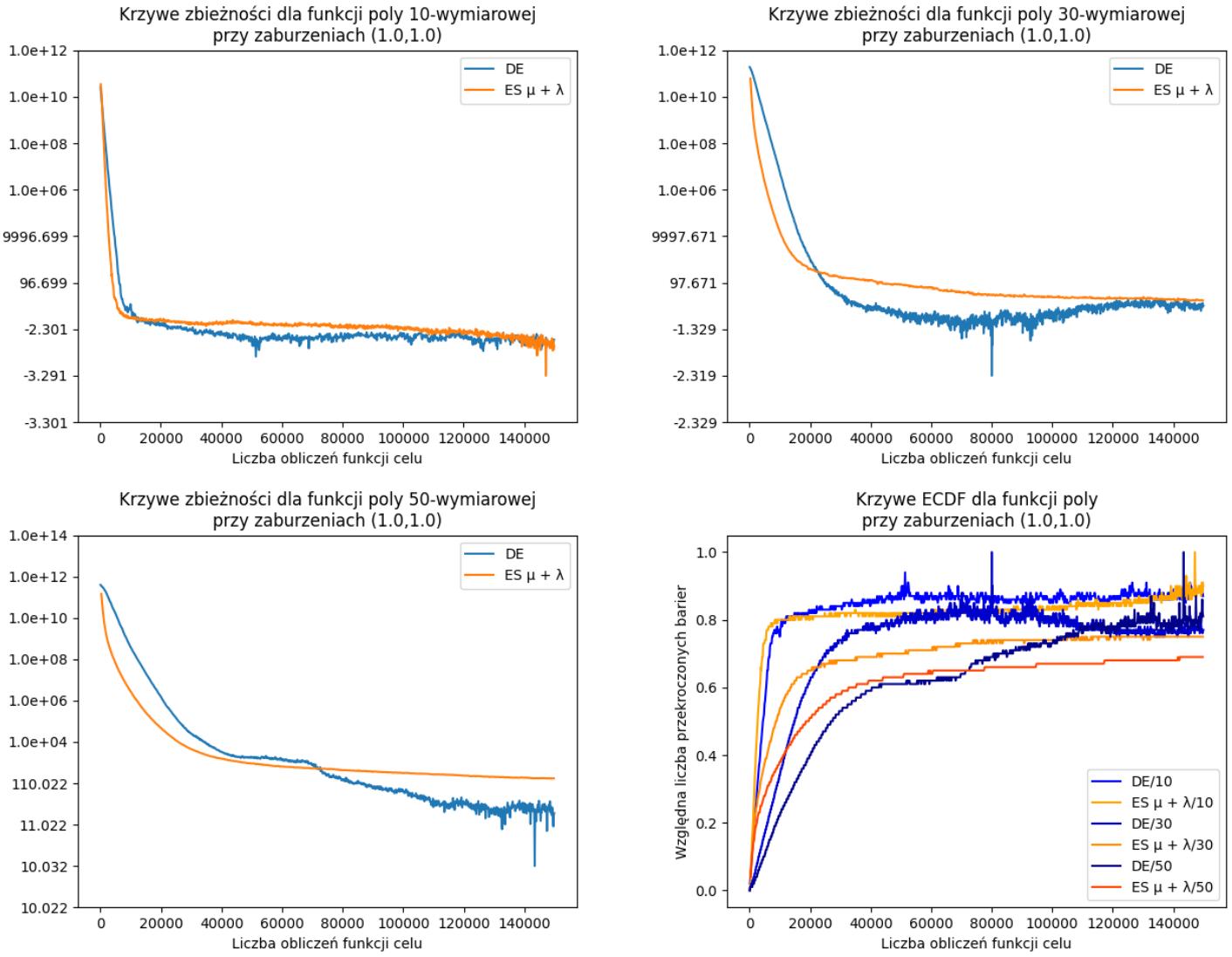
Wyniki algorytmu strategii ewolucyjnej dla funkcji zaburzonej kwadratowej w formie tabelarycznej:

		średnia			odch. standardowe			min			max		
wym.		10	30	50	10	30	50	10	30	50	10	30	50
σ_1	σ_2												
1	1	-14,53	-25,40	-24,94	1,84	8,74	16,22	-17,02	-38,13	-49,37	-9,32	-1,29	18,44
	10	-12,25	-24,45	-25,50	2,08	8,19	13,39	-15,39	-37,44	-40,35	-6,19	-2,45	7,74
10	1	-2,47	4,45	15,53	7,97	12,76	25,36	-14,47	-23,97	-17,50	12,91	30,08	67,47
	10	0,65	1,59	19,25	3,62	14,19	17,07	-10,05	-22,36	-0,44	7,38	33,65	79,53

W tym przypadku, zwiększoną wymiarowość nie zawsze pogarszała wyniki - funkcja zaburzona kwadratowa przyjmuje wartości ujemne, a jej ostateczna wartość jest sumą wartości po wszystkich wymiarach, zatem funkcja ta przyjmuje niższe ujemne wartości dla większych wymiarowości. Ponownie, zaburzenie argumentu miało znacznie większy wpływ na wynik niż zaburzenie wartości, ponieważ funkcja ta jest oparta na funkcji kwadratowej.

6.2.4 Wielowymiarowa funkcja wielomianowa

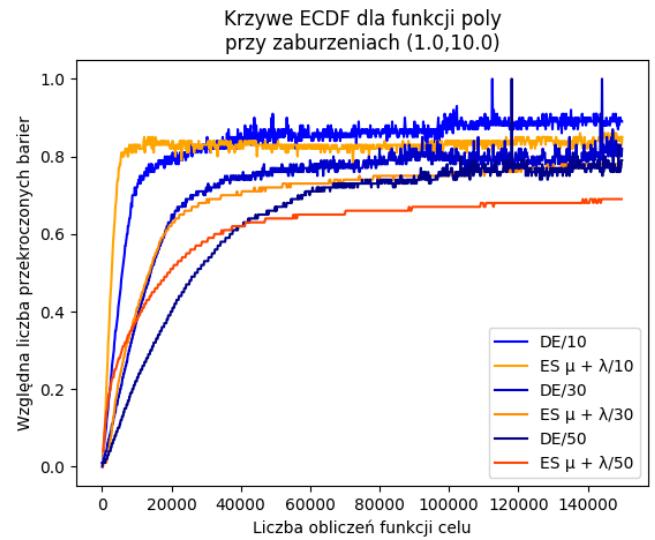
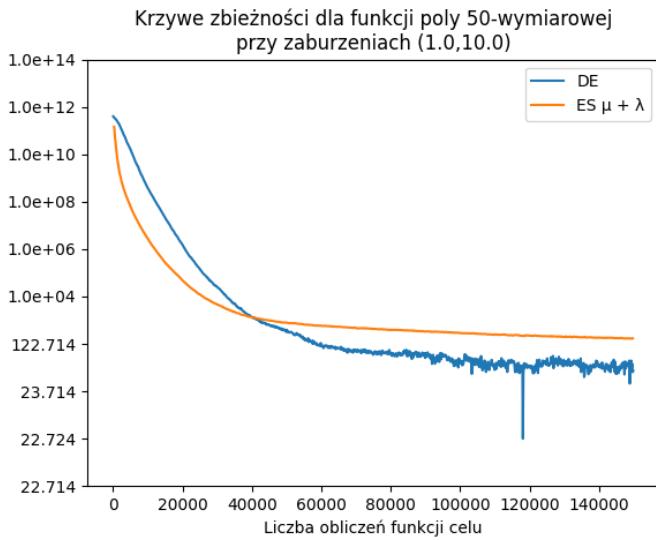
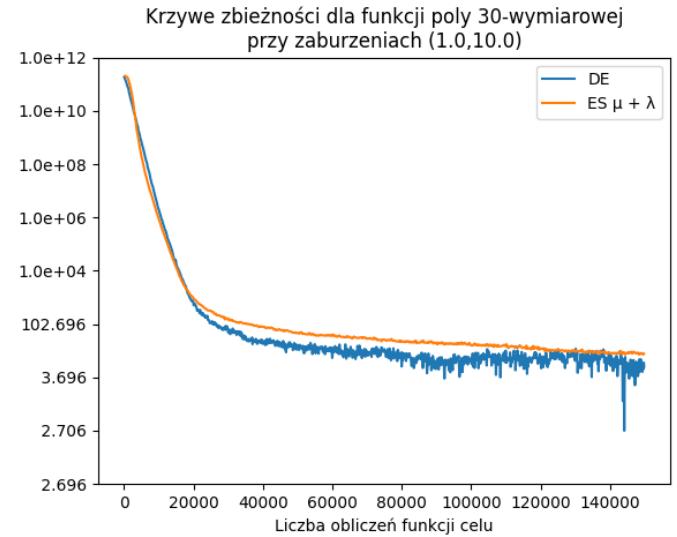
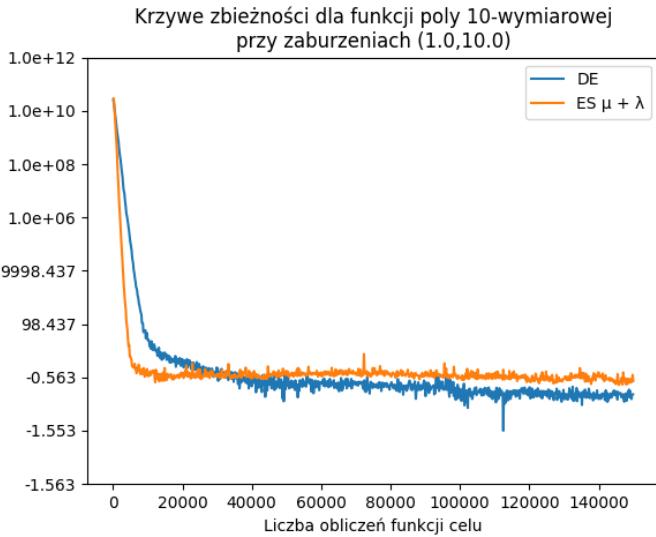
Testy dla małych zaburzeń losowych ($\sigma_1 = 1$ oraz $\sigma_2 = 1$)



Dla funkcji wielomianowej początkowo algorytm ES zbiega nieco szybciej, jednakże utyka w minimum lokalnym i uzyskuje podobny lub gorszy wynik niż algorytm DE.

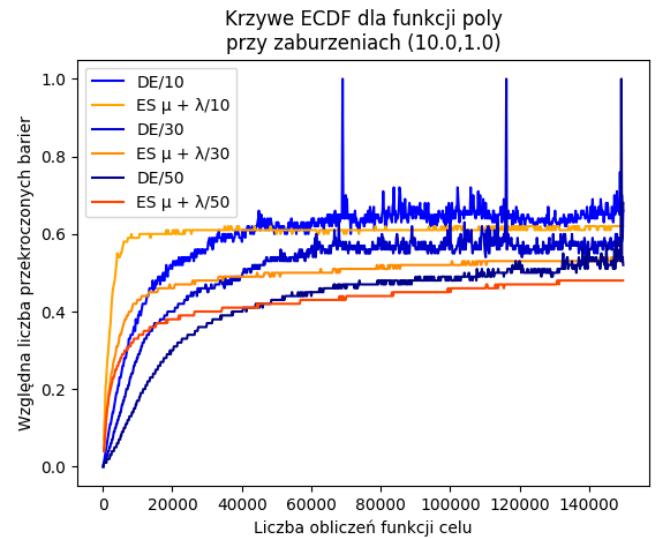
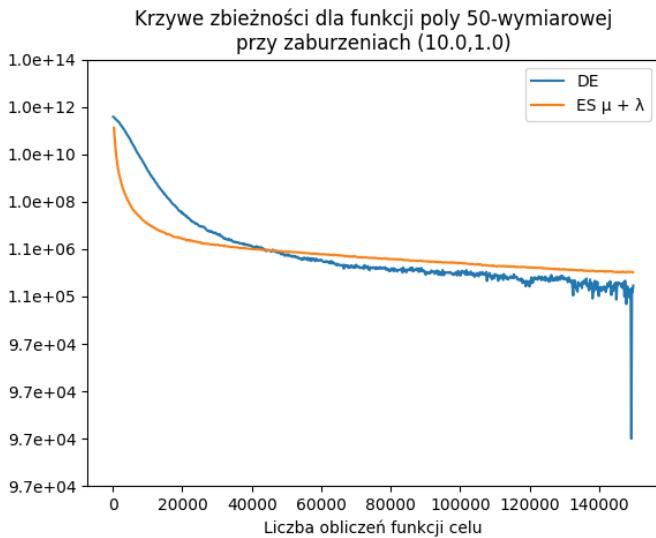
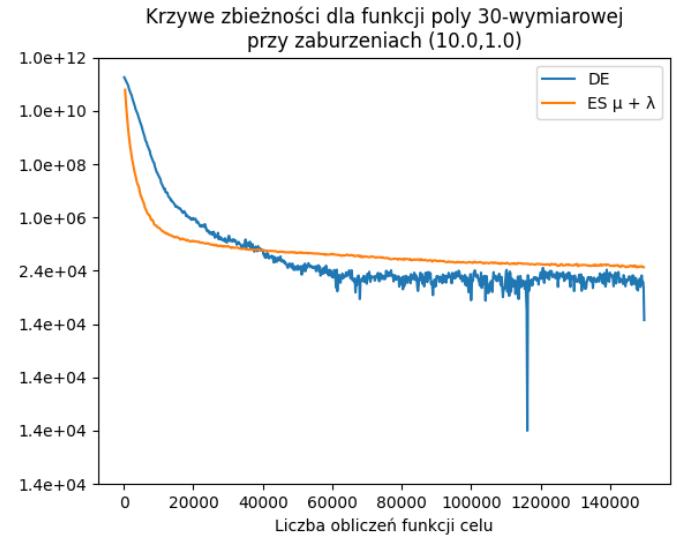
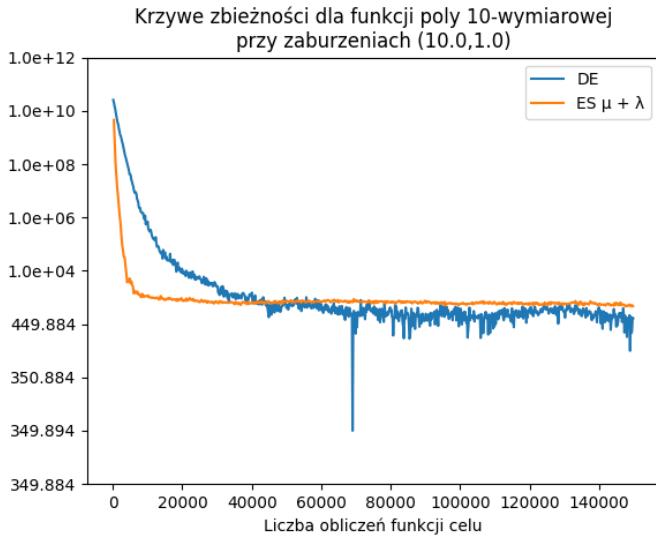
Ciekawy jest przypadek algorytmu DE dla wariantu 30-wymiarowego, gdzie algorytm wyszedł z lepszego minimum lokalnego. Krzywe te są uśrednione dla 25 uruchomień, zatem nie był to pojedynczy przypadek. Może to być przykład efektu survival-of-the flattest, gdzie algorytm przechodzi do szerszego, choć gorszego obszaru. Taki efekt może występować przez obecność losowego zaburzenia argumentu funkcji, które wpływa bardziej na okolice węższego optimum i powoduje utratę osobników z lepszego minimum.

Testy dla dużych zaburzeń losowych wartości ($\sigma_1 = 1$ oraz $\sigma_2 = 10$)



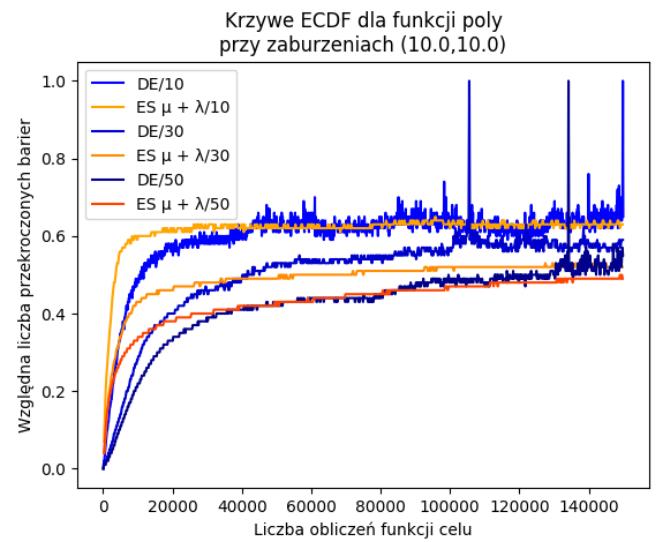
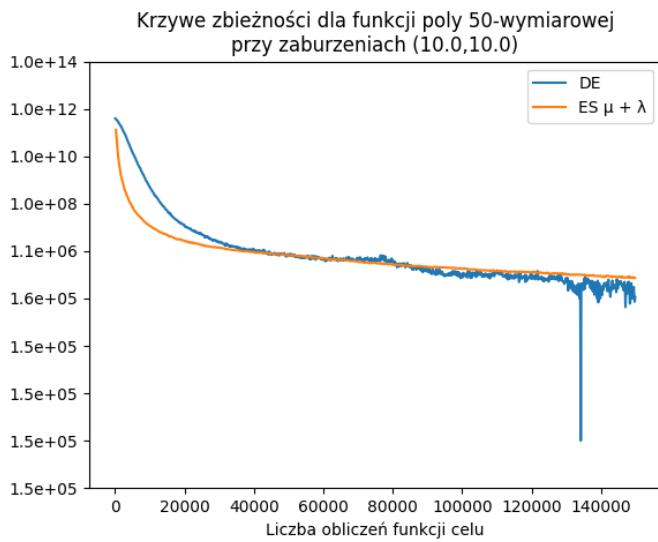
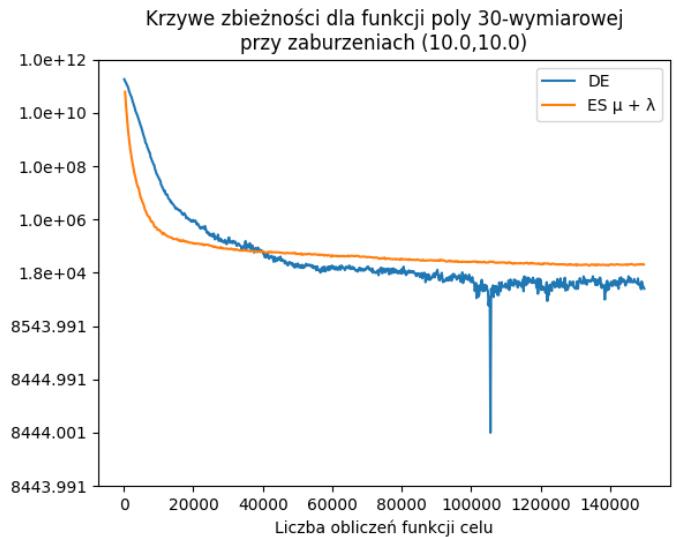
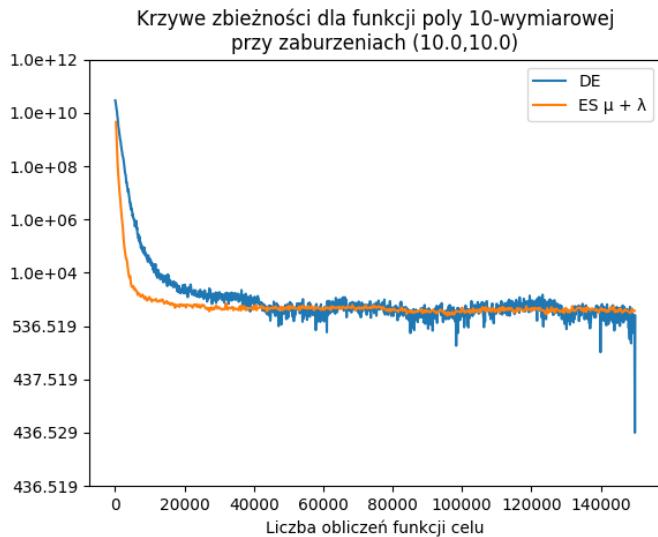
W tym przypadku algorytm DE utrzymał niewielką przewagę nad algorytmem ES, obecną również dla małych zaburzeń. Jednakże, różnica staje się znacząca dopiero dla funkcji 50-wymiarowej.

Testy dla dużych zaburzeń losowych argumentu ($\sigma_1 = 10$ oraz $\sigma_2 = 1$)



Wyniki pozostają takie, jak poprzednio - algorytm DE utrzymał niewielką przewagę nad algorytmem ES.

Testy dla dużych zaburzeń wartości i argumentu ($\sigma_1 = 10$ oraz $\sigma_2 = 10$)



W przeciwieństwie do zaburzonej funkcji kwadratowej, gdzie algorytm ES wykazał się większą odpornością na losowość, dla funkcji wielomianowej dla wszystkich wariantów siły losowości algorytm DE utrzymał niewielką przewagę nad algorytmem ES.

Może to być spowodowane tym, że minima lokalne funkcji zaburzonej kwadratowej są stosunkowo płytkie i przy silniejszej losowości przestają być widoczne dla algorytmu. Takie zjawisko nie zachodzi dla funkcji wielomianowej - stanowi to znaczącą różnicę między tymi funkcjami.

Wyniki algorytmu ewolucji różnicowej dla funkcji wielomianowej w formie tabelarycznej:

		średnia			odch. standardowe			min			max		
wym.		10	30	50	10	30	50	10	30	50	10	30	50
σ_1	σ_2												
1	1	-2,95	7,84	13,62	1,89	12,85	28,05	-5,55	-5,72	-15,81	1,09	35,43	97,61
	10	-1,33	5,26	30,13	2,22	7,45	27,10	-4,75	-5,34	-1,98	4,06	26,40	90,73
10	1	513,09	1e+04	1e+05	890,21	2e+04	1e+05	2,58	576,52	1e+04	4e+03	1e+05	6e+05
	10	682,83	1e+04	2e+05	1e+03	1e+04	2e+05	1,01	1e+03	2e+04	5e+03	6e+04	9e+05

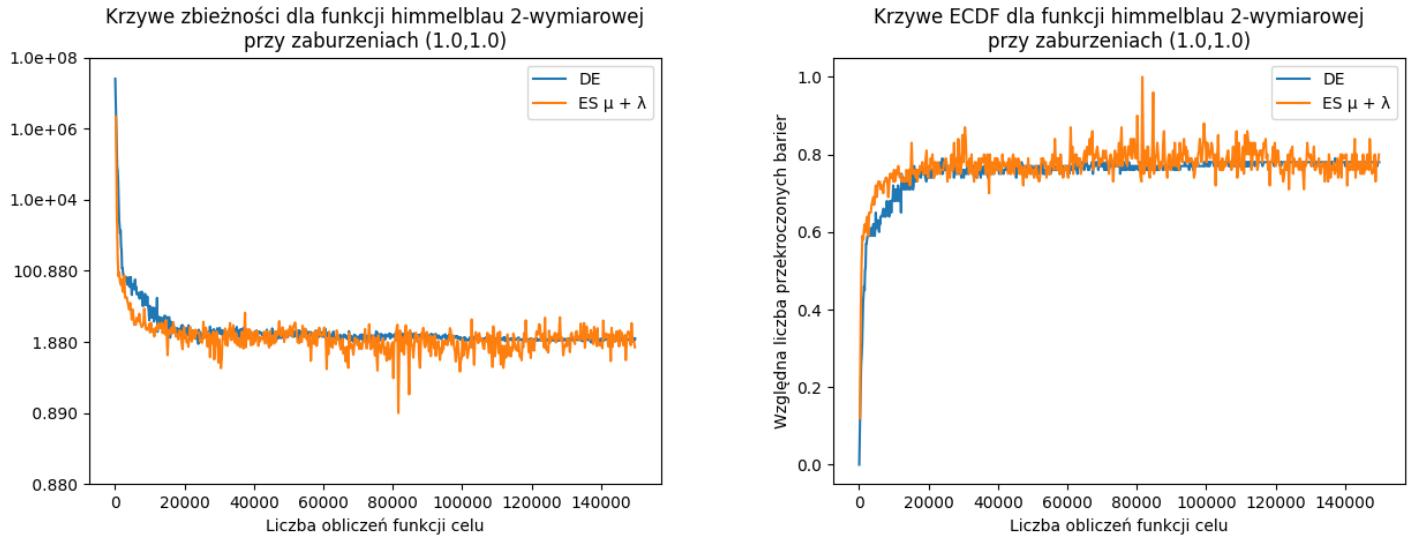
Wyniki algorytmu strategii ewolucyjnej dla funkcji wielomianowej w formie tabelarycznej:

		średnia			odch. standardowe			min			max		
wym.		10	30	50	10	30	50	10	30	50	10	30	50
σ_1	σ_2												
1	1	-3,11	14,94	182,05	1,61	47,35	269,30	-5,74	-10,19	-6,28	0,56	170,52	965,92
	10	-0,83	10,40	192,79	1,65	29,25	335,03	-4,37	-9,25	-5,26	1,39	123,38	1e+03
10	1	819,70	3e+04	2e+05	1e+03	6e+04	2e+05	-1,48	186,87	2e+04	4e+03	2e+05	9e+05
	10	806,32	3e+04	2e+05	1e+03	6e+04	3e+05	-2,47	172,10	3e+04	6e+03	2e+05	2e+06

W tym przypadku, zwiększoną wymiarowość pogorszyła wyniki - pomimo tego, że minima tej funkcji również przyjmują wartości ujemne, coraz głębsze dla rosnącej wymiarowości, zwiększoną trudność zadania ich znalezienia spowodowała pogorszenie wyników. Ponownie, zaburzenie argumentu miało znacznie większy wpływ na wynik niż zaburzenie wartości - funkcja ta zawiera składnik w czwartej potędze, więc efekt jest bardziej drastyczny niż dla funkcji kwadratowej.

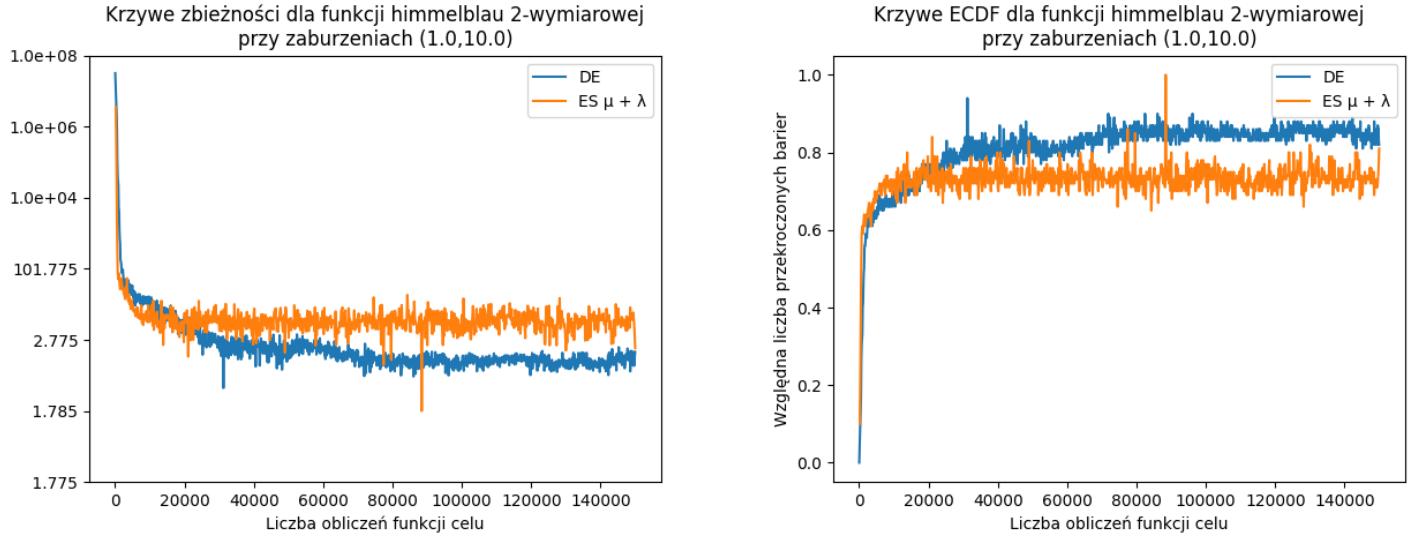
6.2.5 Funkcja Himmelblau

Testy dla małych zaburzeń losowych ($\sigma_1 = 1$ oraz $\sigma_2 = 1$)



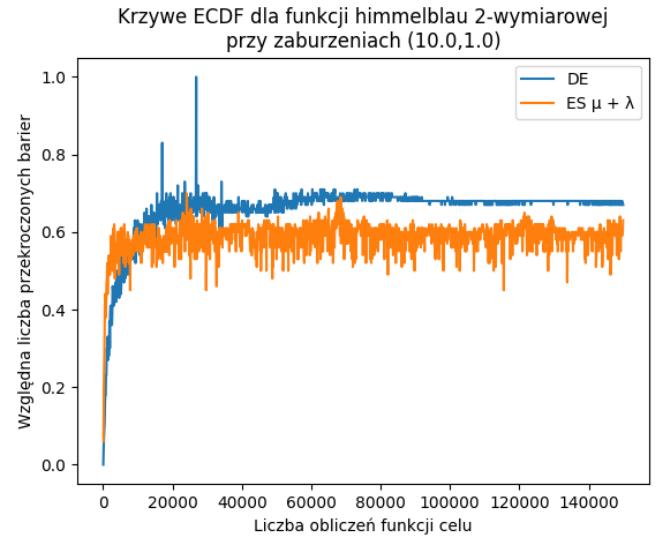
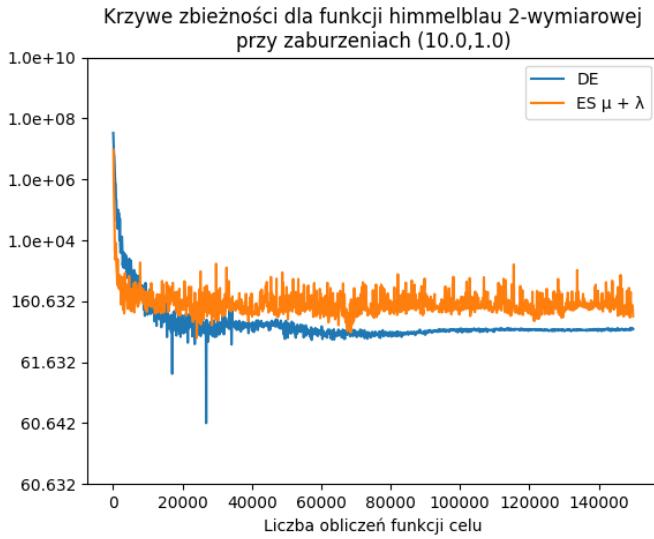
W przypadku tej funkcji nie występuje problem minimów lokalnych - istnieją 4 minima, ale wszystkie są sobie równe. Oba algorytmy okazały się praktycznie tak samo skuteczne w przybliżaniu jednego z minimów funkcji.

Testy dla dużych zaburzeń losowych wartości ($\sigma_1 = 1$ oraz $\sigma_2 = 10$)



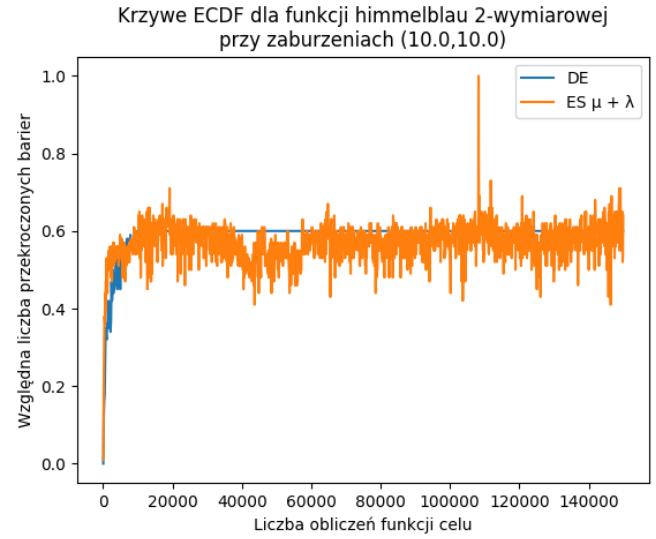
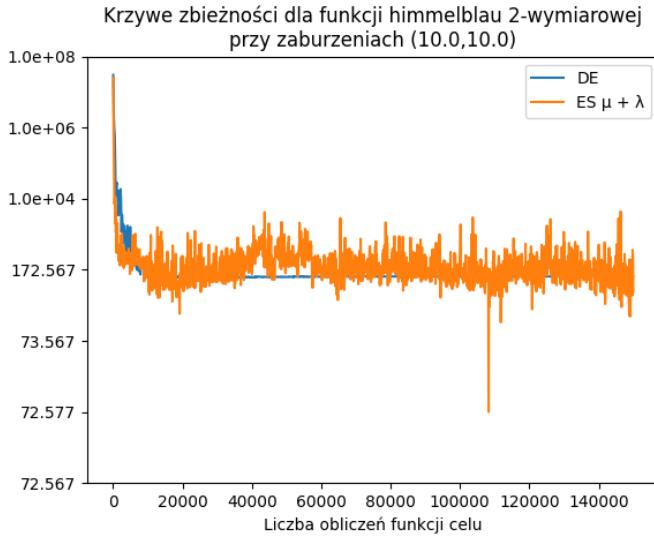
Dla większych zaburzeń losowych wartości algorytm DE okazał się być bardziej wrażliwy na zmiany rozkładu funkcji z niepewnością spowodowane wartością oryginalnej, niezaburzonej funkcji.

Testy dla dużych zaburzeń losowych argumentu ($\sigma_1 = 10$ oraz $\sigma_2 = 1$)



Dla większych zaburzeń losowych argumentu wyniki są analogiczne jak w poprzednim przypadku. Jednakże, wyniki algorytmu ES są bardzo zróżnicowane z iteracji na iterację - ten efekt jest jeszcze bardziej widoczny poniżej.

Testy dla dużych zaburzeń wartości i argumentu ($\sigma_1 = 10$ oraz $\sigma_2 = 10$)



Podczas gdy algorytm ewolucji różnicowej zbiegał do w miarę stałej wartości, wyniki algorytmu ES okazały się być bardzo zróżnicowane, szczególnie przy dużych zaburzeniach losowych. Oznacza to, że najlepsze, wybierane do kolejnych populacji osobniki miały cały czas duże siły mutacji - zazwyczaj siły mutacji maleją, gdy przybliżane jest minimum lokalne. Wygląda na to, że populacja była podzielona między 4 optima globalne - przez to, najlepsze okazywały się być osobniki, które pochodząc z jednego optimum, trafiły do innego. Miały one dużą siłę mutacji i nie doszło do spadku siły mutacji w populacji.

Wyniki algorytmu ewolucji różnicowej dla funkcji Himmelblau w formie tabelarycznej:

σ_1	σ_2	średnia	odch. std.	min	max
wym.		2	2	2	2
1	1	2,11	2,84	0,01	11,24
	10	2,23	2,46	0,01	9,34
10	1	72,97	57,50	2,90	191,24
	10	137,39	145,58	0,06	745,62

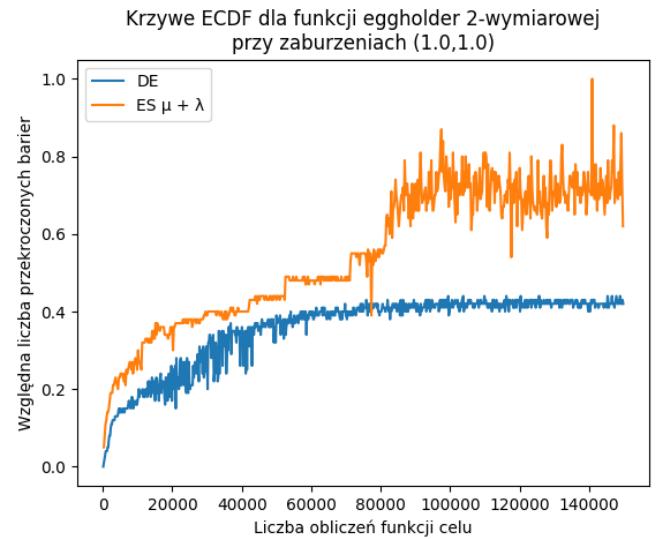
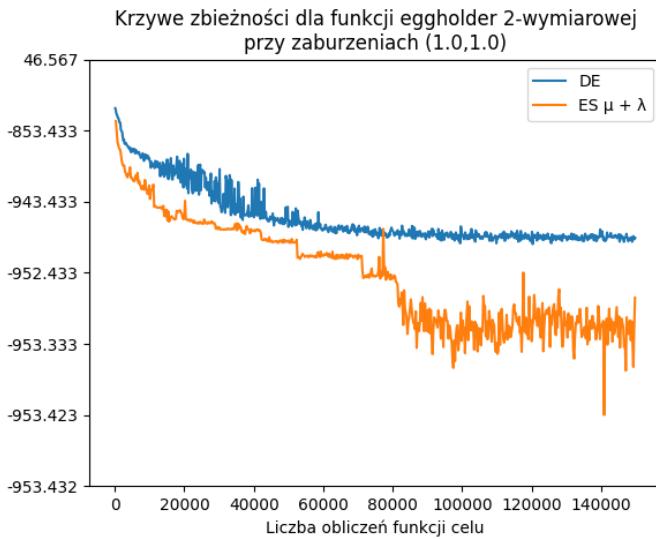
Wyniki algorytmu strategii ewolucyjnej dla funkcji Himmelblau w formie tabelarycznej:

σ_1	σ_2	średnia	odch. std.	min	max
wym.		2	2	2	2
1	1	1,58	1,95	0,01	7,81
	10	2,36	2,45	0,11	10,56
10	1	92,59	61,56	5,05	180,82
	10	96,31	55,56	0,00	170,12

Ponownie, zaburzenie argumentu miało znacznie większy wpływ na wynik niż zaburzenie wartości, ponieważ funkcja Himmelblau zawiera składniki stopnia drugiego.

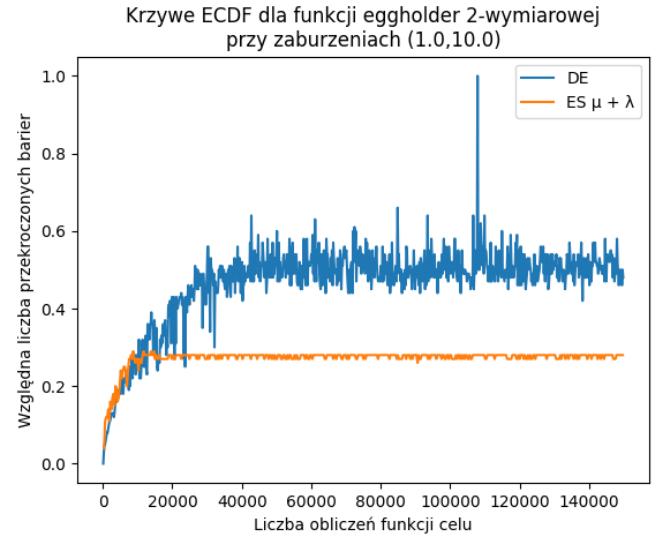
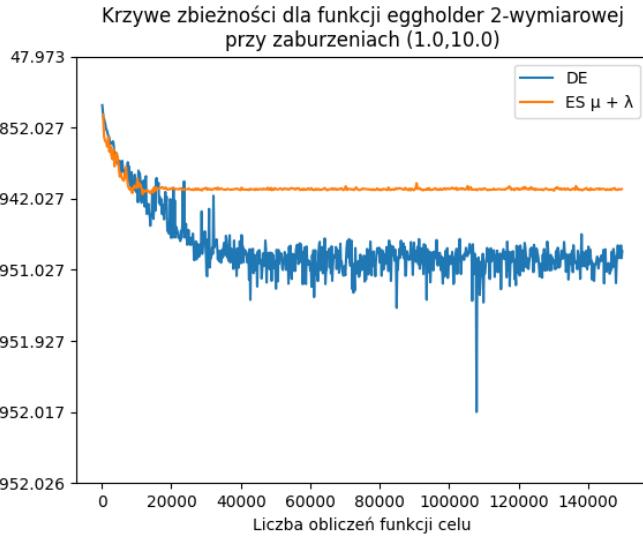
6.2.6 Funkcja Eggholder

Testy dla małych zaburzeń losowych ($\sigma_1 = 1$ oraz $\sigma_2 = 1$)



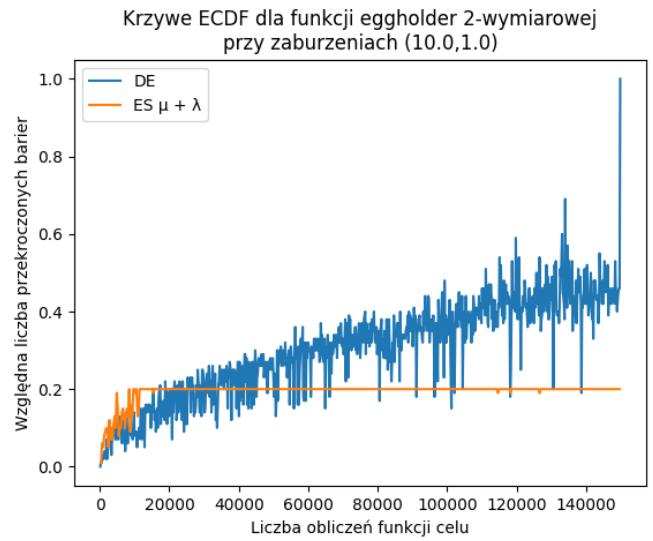
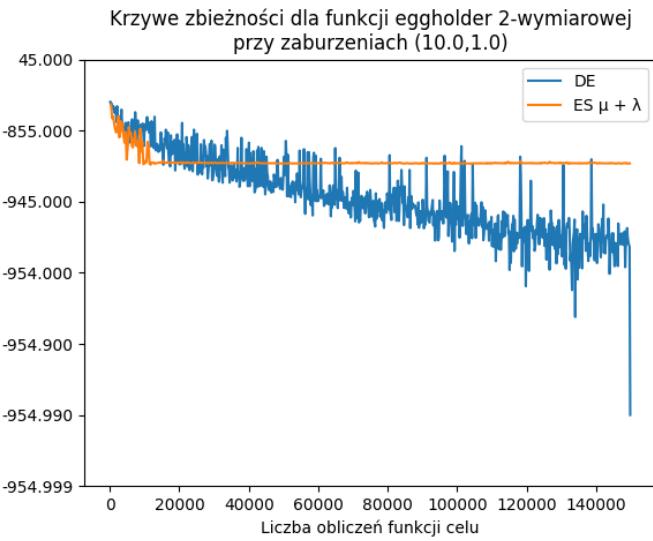
Wyniki obu algorytmów dla funkcji Eggholder są podobne (oba są niższe niż -940 , gdy minimum globalne ma wartość około $-959,64$), jednakże algorytm ES uzyskał wynik nieco bliższy minimum globalnego.

Testy dla dużych zaburzeń losowych wartości ($\sigma_1 = 1$ oraz $\sigma_2 = 10$)



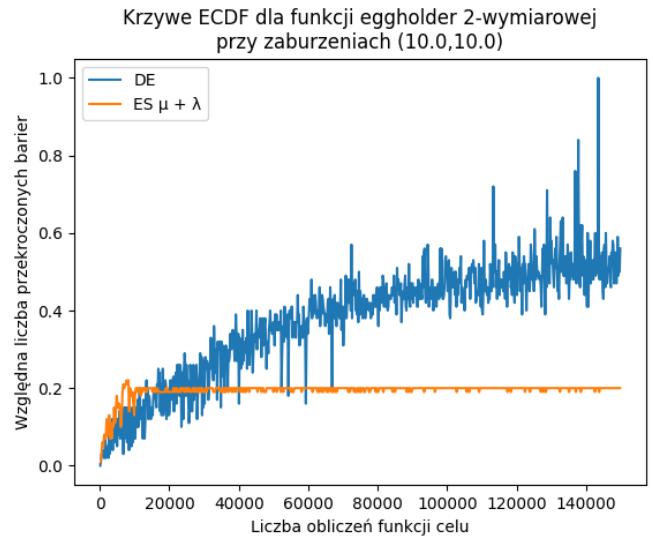
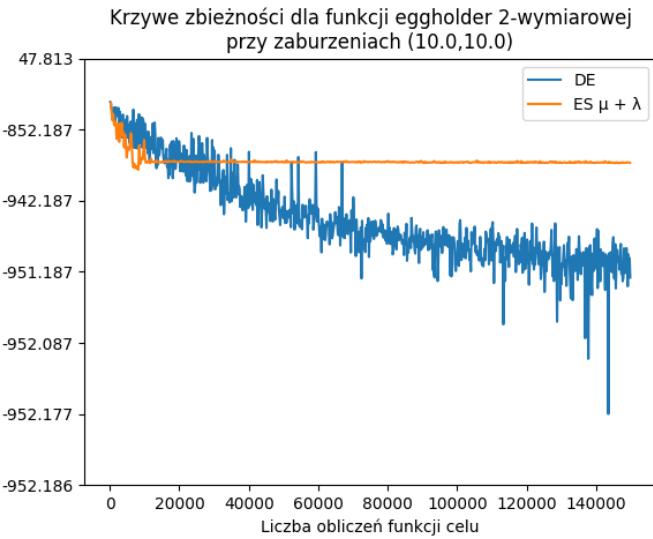
Przy silniejszej losowości, sytuacja się odwróciła - tym razem to algorytm DE uzyskiwał średnio lepsze wyniki, choć różnica wciąż jest mało znacząca.

Testy dla dużych zaburzeń losowych argumentu ($\sigma_1 = 10$ oraz $\sigma_2 = 1$)



W tym przypadku na uwagę zasługuje fakt, że algorytm ewolucji różnicowej nie zbiegł przez praktycznie cały czas uruchomienia algorytmu - dalej stopniowo przybliżał minimum globalne lub jedno z minimów bliskich globalnemu.

Testy dla dużych zaburzeń wartości i argumentu ($\sigma_1 = 10$ oraz $\sigma_2 = 10$)



Przypadek z dużymi zaburzeniami losowymi zarówno wartości, jak i argumentu zachowuje się w dużej mierze podobnie do poprzedniego - algorytm ES szybko zatrzymał się w minimum lokalnym, a ewolucja różnicowa stopniowo przybliżała jedno z lepszych minimów. Ponownie widoczne jest zjawisko, gdzie większa losowość zwiększa liczbę obliczeń funkcji celu potrzebną do zbiegnięcia algorytmu do względnie stałej wartości.

Wyniki algorytmu ewolucji różnicowej dla funkcji Eggholder w formie tabelarycznej:

σ_1	σ_2	wym.	średnia	odch. std.	min	max
		2	2	2	2	2
1	1	-950,35	21,19	-959,44	-893,85	
	10	-950,20	17,37	-958,50	-893,16	
10	1	-954,99	1,92	-956,91	-950,30	
	10	-951,36	12,89	-956,77	-893,47	

Wyniki algorytmu strategii ewolucyjnej dla funkcji Eggholder w formie tabelarycznej:

σ_1	σ_2	wym.	średnia	odch. std.	min	max
		2	2	2	2	2
1	1	-952,99	14,03	-959,15	-894,51	
	10	-938,37	19,61	-958,75	-894,22	
10	1	-920,44	22,82	-955,06	-885,95	
	10	-918,09	24,82	-958,94	-887,32	

Jak widać, zwiększoną losowość pogorszyła wyniki algorytmu strategii ewolucyjnej $\mu + \lambda$, ale wyniki algorytmu ewolucji różnicowej pozostały praktycznie bez zmian. Dla wyższych zaburzeń argumentu wyniki się nawet poprawiły - wydaje się to być przypadkiem odstającym bez daleko idących konsekwencji. Jednak jeśli spojrzeć na najlepsze osiągnięte wartości (min), to minimum globalne zostało przybliżone lepiej dla mniejszych sił losowości.

6.2.7 Porównanie tabelaryczne

Poniżej przedstawione jest porównanie obu algorytmów w tabelach - w komórce tabeli DE oznacza, że algorytm ewolucji różnicowej okazał się lepszy, ES, że lepsza jest strategia ewolucyjna $\mu + \lambda$, –, że wyniki nie są statystycznie różne. Zbadaliśmy, czy wyniki są statystycznie różne za pomocą testu t-Welcha (uogólnienia testu t-Studenta bez założenia o równości wariancji populacji), przyjmując próg p-wartości, poniżej którego odrzucamy hipotezę zerową równy $\alpha = 0,1$.

Tabele dla $\sigma_1 = 1$ i $\sigma_2 = 1$:

Nazwa funkcji \ Wymiarowość	10	30	50
ackley	–	ES	ES
sphere	–	–	–
poly	–	–	DE
disturbed square	DE	DE	DE

Nazwa funkcji \ Wymiarowość	2
himmelblau	–
eggholder	–

Tabele dla $\sigma_1 = 1$ i $\sigma_2 = 10$:

Nazwa funkcji \ Wymiarowość	10	30	50
ackley	ES	ES	ES
sphere	ES	ES	ES
poly	–	–	DE
disturbed square	–	DE	–

Nazwa funkcji \ Wymiarowość	2
himmelblau	–
eggholder	DE

Tabele dla $\sigma_1 = 10$ i $\sigma_2 = 1$:

Nazwa funkcji \ Wymiarowość	10	30	50
ackley	ES	ES	ES
sphere	–	–	–
poly	–	–	–
disturbed square	–	–	ES

Nazwa funkcji \ Wymiarowość	2
himmelblau	–
eggholder	DE

Tabele dla $\sigma_1 = 10$ i $\sigma_2 = 10$:

Nazwa funkcji \ Wymiarowość	10	30	50
ackley	ES	–	DE
sphere	–	–	–
poly	–	–	–
disturbed square	–	ES	ES

Nazwa funkcji \ Wymiarowość	2
himmelblau	–
eggholder	DE

Sumarycznie, dla 15 zadań (konkretniej funkcji, dla konkretnej wymiarowości, dla danych sił losowości σ_1 i σ_2) algorytm ES $\mu + \lambda$ okazał się być lepszy, a algorytm DE okazał się być lepszy dla 10 zadań.

7 Końcowy wniosek

W warunkach niepewności skuteczność obu algorytmów okazała się być zależna od zadania. Nie można zatem stwierdzić, że jeden z algorytmów jest ogólnie lepszy.