UNIWERSYTET TECHNOLOGICZNO-PRZYRODNICZY

WTiE

Informatyka Stosowana

DOKUMENTACJA

Projekt zaliczeniowy z przedmiotu

„Narzędzia programistyczne”

Zastosowanie metod sztucznej inteligencji inspirowanych naturą do optymalizacji funkcji matematycznych

Jonasz Kulpinski

Jarosław Borkowski

Kamil Stenzel

Jakub Koperski

Bydgoszcz 2017

Spis treści

1. Opis wykorzystanych algorytmów

1.1. Algorytmy optymalizacji rojem cząstek

Algorytm optymalizacyjny roju cząstek (PSO-Particle Swarm Optimization) to technika obliczeniowa, która została opracowana na wzór zachowania, które zaobserwowano u ptaków i ryb w ławicach. Zachowania te polegają na tym, że poszczególni członkowie w stadzie starają się tak dostosować prędkość ruchu, aby utrzymywać określony, najkorzystniejszy dystans do sąsiednich osobników. Korzyść z tego modelu zachowania jest taka, że wszyscy członkowie reagują jednocześnie, co zapobiega kolizjom, umożliwia szybkie zmienianie kierunku ruchu całej grupy i sprawniejsze przemieszczanie się, szczególnie wtedy, gdy zachodzi potrzeba wykonania np. zwrotu, który wymaga reorganizacji układu całego "oddziału".

Propozycja algorytmu PSO autorstwa Ebercharta i Kennedy'ego [1], została opracowana na wzór zachowań zwierząt, które mają poprawić bezpieczeństwo stada, ułatwić mu poszukiwanie jedzenia i poprawić jego mobilność. Z poziomu algorytmu, rój znajduje się w przestrzeni posiadającej D wymiarów, poruszając się z losowo określonymi pozycjami i prędkościami, jednak wiadoma jest ich wartość najkorzystniejsza.

Można teraz zastanowić się nad pozycją i-tej cząsteczki $X_{i,m}$, przemieszczającej się wewnątrz D wymiarowej przestrzeni. Zapisywana jest jako$Pbest_{i,m}$ najkorzystniejsza i najlepsza pozycja i-tej cząstki. Najlepsza spośród cząstek w populacji jest określana jako $gbest_{i,m}$ i zapisywana, zaś najlepsza z cząstek występujących w najbliższym sąsiedztwie to $Lbest_{i,m}$. Prędkość poruszania się poszczególnych cząsteczek znajdujących się w przestrzeni rozważań zapisywana jest jako $V_{i,m}$. Prędkości oraz pozycje aktualizowane są zależnie od obliczeń do których wykorzystywane są pozycje i prędkości bieżące [2, 3].

Miejsce, w przestrzeni x_i o D wymiarach, w którym znajduje się cząstka jest opisywane w sposób następujący:

$$x_i = (X_{i,1}, X_{i,2}, ..., X_{i,D}), \quad i = 1, ..., N \tag{1}$$

gdzie: N to ilość cząsteczek w roju

Zapamiętywanie najkorzystniejszej pozycji i-tej cząsteczki $Pbest_i$:

$$Pbest_i = (Pbest_1, Pbest_2, ..., Pbest_D) \tag{2}$$

Najlepsza cząstka w populacji, czyli o najkorzystniejszym wskaźniku $Pbest_i$, po zapisaniu określana jest jako $gbest$. V_i, czyli prędkość cząstki jest zapisywana w postaci:

$$V_i = (V_{i,1}, V_{i,2}, ..., V_{i,D}) \tag{3}$$

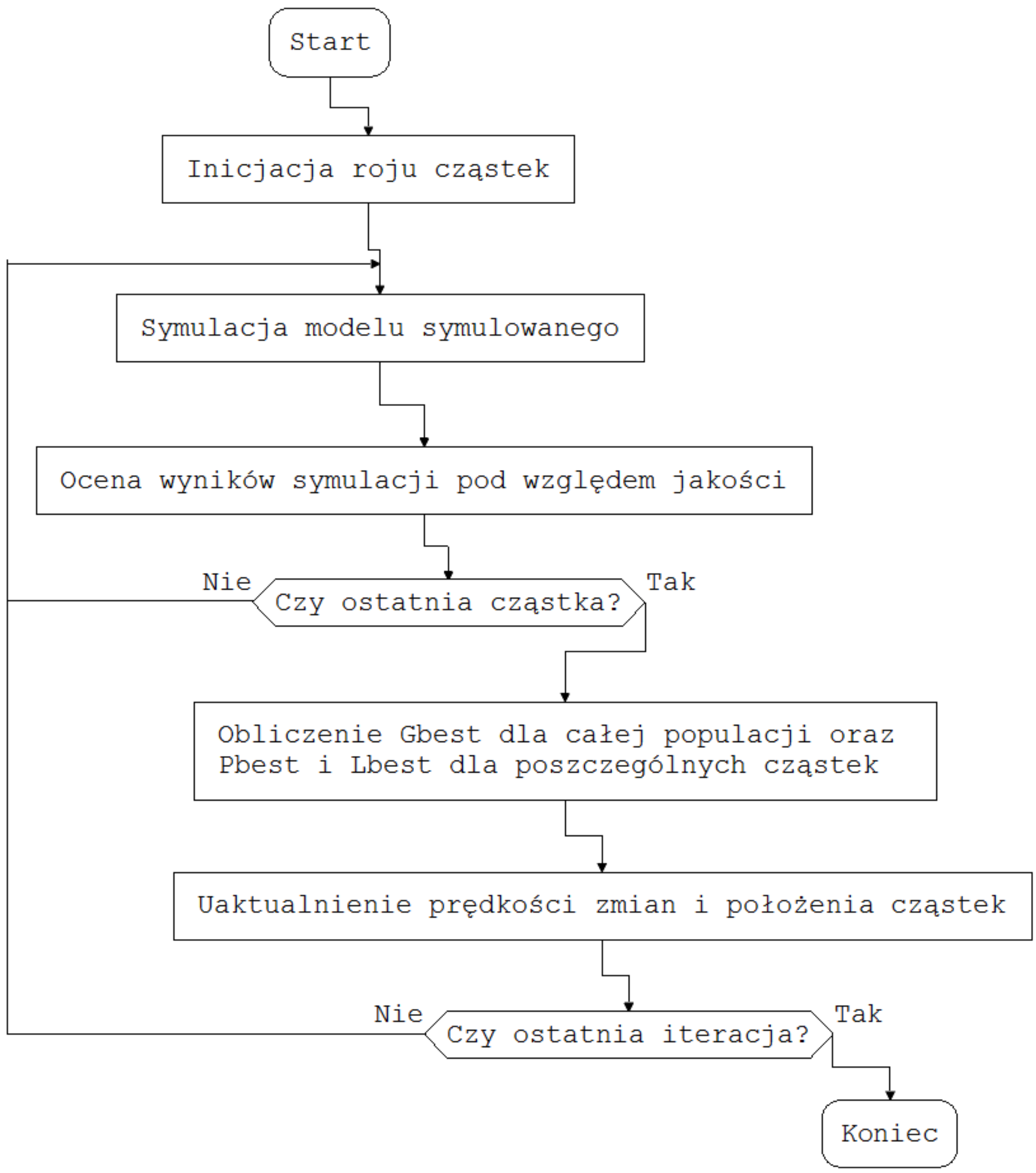
Wykorzystując różnicę odległości pozycji i-tej cząsteczki x_i od współrzędnych $Pbest_i$ oraz $Lbest$, uaktualniana jest pozycja i prędkość każdej kolejnej cząstki, odbywa się to z wykorzystaniem wzorów:

$$V_{i,m}^{(t+1)} = w * V_{i,m}^{(t)} + c_1 rand[0,1] * (Pbest_{i,m} - x_{i,m}^{(t)}) + c_2 rand[0,1] * (Lbest_m - x_{i,m}^{(t)}) \tag{4}$$

$$x_{i,m}^{(t+1)} = x_{i,m}^{(t)} + V_{i,m}^{(t+1)}, \quad i = 1, ..., N; \quad m = 1, ..., D \tag{5}$$

gdzie: w to wagowy współczynnik inercji, c_1, c_2 to stałe przyspieszenia, $rand[0,1]$ to generator liczb losowych z zakresu [0,1].

Proces wyznaczania pozycji cząsteczki przedstawić można na układzie o dwóch wymiarach . Na początku określany jest nowy wektor V^{k+1} (wektor prędkości) dla cząstki x^k. Jest to obliczane z wykorzystaniem bieżącej pozycji tej cząstki i pozycję $Pbest$ oraz $Lbest$. Wyznaczony wektor jest niezbędny do określenia nowych współrzędnych w kolejnej instrukcji danej iteracji algorytmu x^{k+1}. Na rysunku 1.1.1 przedstawiono schemat blokowy omawianego algorytmu optymalizacji rojem cząstek [4].



Rysunek 1.1.1: Algorytm PSO na schemacie blokowym

1.2. Algorytm świetlika

Algorytm GSO (Glowworm Swarm Optimization), czyli algorytm świetlika został wymyślony przez Xin-She Yang'a w Cambridge University w 2007 roku. Algorytm ten jak sama nazwa wskazuje wzoruje się swym działaniem na zachowaniu robaczków świętojańskich. "Świecenie" tych owadów ma za zadanie wabić ofiary, przestrzegać wrogich osobników przed zbliżaniem się oraz jest wykorzystywane w zalotach. Elementem, który wykorzystuje się w omawianych algorytmach są zmiany w natężeniu światła emitowanego przez świetliki, co określa jaki jest cel wysyłania sygnału świetlnego. Jeśli jakiś z owadów świeci jaśniej, reszta, o mniej intensywnym sygnale będzie zbliżała się do niego, a to z kolei umożliwia wydajniejsze zbadanie przez algorytm przestrzeni poszukiwań [5, 6].

W algorytmie GSO obowiązują następujące zasady [5]:

- zarówno osobniki żeńskie jak i męskie są uważane za atrakcyjne,
- to jak bardzo dany osobnik jest atrakcyjny zależy od blasku emitowanego przez niego światła, im odległość między osobnikami większa, tym intensywność świecenia coraz mniejsza, kiedy osobniki są jednakowo atrakcyjne, przemieszczają się w losowych kierunkach,
- funkcja celu determinuje swą wartością, jakie jest natężenie wysyłanego przez świetlika sygnału świetlnego.

Atrakcyjność jest cechą właściwą dla każdego świetlika, istnieją jednak różnice w poziomie atrakcyjności poszczególnych osobników. Atrakcyjność określa funkcja dystansu między wybranymi dwoma owadami:

$$\beta(r) = \beta_0 e^{-\gamma r^m}, \quad m \geqslant 1, \tag{6}$$

gdzie: β_0 to atrakcyjność kiedy $r = 0$, γ to wartość współczynnika absorpcji promieniowania świetlnego.

Wspomniany wcześniej dystans pomiędzy wybraną parą świetlików (i, j), zajmujących pozycje x_i oraz x_j można obliczyć za pomocą wzoru:

$$r_{ij} = \parallel x_i - x_j \parallel = \sqrt{\sum_{k=1}^{d} (x_{i,k} - x_{j,k})^2}, \tag{7}$$

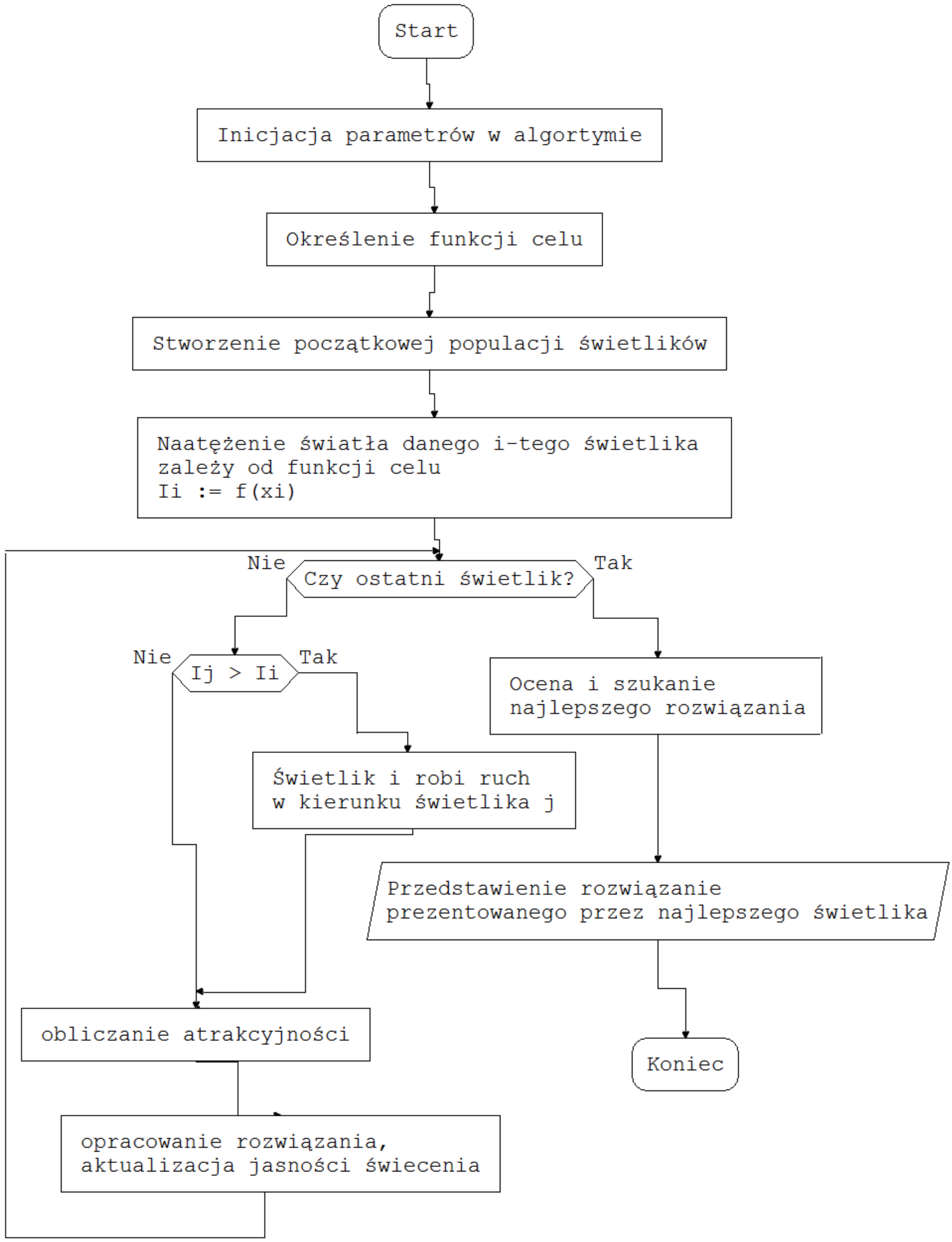
gdzie: d to ilość wymiarów.

Świetlik i porusza się w sposób określony następującym wzorem:

$$x_i = xi + \beta_0 e^{-\gamma r_{ij}^2}(x_j - x_i) + \alpha(rand - 0,5), \tag{8}$$

gdzie: x_i to aktualne współrzędne świetlika i, drugi element sumy stanowi o atrakcyjności, a ostatni wykorzystywany jest, jeżeli występuje losowa zmiana położenia; $rand$ to losowo generowana wartość [0, 1], natomiast $\alpha \in (0, 1)$. Zwykle β_0 i γ przyjmują wartość 1.

Budowa algorytmu GSO została przedstawiona na schemacie blokowym (rys. 1.2.2) [7, 5]:



Rysunek 1.2.2: Algorytm świetlika na schemacie blokowym

1.3. Algorytm nietoperza

Bat algorithm (BA) czyli algorytm nietoperza to metoda metaheurystyczna, zaproponowana przez Yang'a w 2010 roku [5, 8]. Zdolność echolokacji nietoperzy to fascynująca rzecz, ponieważ pomaga ona znaleźć nietoperzom zdobycz oraz rozpoznawać różne rodzaje owadów w zupełnej ciemności [9]. Oparty o echolokację nietoperzy algorytm, prowadzi proces poszukiwań za pomocą sztucznych odpowiedników nietoperzy, które wysyłają impulsy o odpowiedniej częstotliwości i głośności, podobnie jak to ma miejsce w naturze. Kiedy zwierzęta te gonią swą zdobycz, natężenie impulsów jest zmniejszane, a rośnie ich częstotliwość.

Algorytm nietoperza jest wydajny w przypadku optymalizacji danych o małej komplikacji parametrów [10, 11, 12], szeroko się go używa w optymalizacji inżynieryjnej [13] i wieloobiektowej [14]. Jednak ze względu na niską różnorodność populacji, traci na wydajności przez konwergencję w przypadku optymalizacji problemu trudnego [15]. Powstały różne warianty algorytmu nietoperza, starające się zwiększyć różnorodność populacji, żeby uniknąć uwięzienia w optimum lokalnym.

Echolokacja jest istotną cechą charakteryzującą nietoperze. Yang odwzorował ich charakterystykę w swym algorytmie. Nietoperze latają z użyciem echolokacji aby uniknąć przeszkód i zlokalizować pożywienie. W celu przekształcenia zachowania zwierząt na działanie algorytmu, trzeba dokonać pewnych uproszczeń i zastosować wyidealizowane reguły [8].

- Wszystkie nietoperze używają echolokacji do określania dystansu od obiektu i potrafią rozróżnić, czy konkretny obiekt jest przeszkodą, czy potencjalnym pożywieniem.
- Nietoperze latają losowo z prędkością v_i, znajdując się w miejscu x_i, emitują fale o stałej częstotliwości f_{min}, różnej długości λ i głośności A_0, żeby szukać zdobyczy. Mogą automatycznie dostosowywać długość fali lub częstotliwość emitowanych impulsów, a także regulować szybkość emisji sygnałów $r \in [0, 1]$, w zależności od bliskości celu.
- Mimo, iż poziom głośności może być różny pod wieloma względami, zakłada się, że głośność może przyjąć wartości od dużej (dodatniej) A_0 do minimalnej stałej A_{min}.

W algorytmie BA, dla i-tych nietoperzy roju, jest określana pozycja (rozwiązanie)x_i, prędkość v_i i częstotliwość f_i, każdy nietoperz przemieszcza się w kierunku najlepszej aktualnej pozycji(rozwiązania), a jego pozycja, prędkość oraz częstotliwość są aktualizowane podczas kolejnych iteracji następująco:

$$f_i = f_{min} + (f_{max} - f_{min})\beta$$

$$v_i^t = v_i^{t-1} + (x_i^{t-1} - x_g^{t-1})f_i \tag{9}$$

$$x_i^t = x_i^{t-1} + v_i^t$$

gdzie: β jest liczbą losową równomiernego rozmieszczenia o wartości [0,1], a x_g^{t-1} reprezentuje aktualnie najlepsze globalne rozwiązanie (pozycję) po porównaniu wszystkich rozwiązań (pozycji) spośród wszystkich n nietoperzy. Te równania mogą zagwarantować zdolności poszukiwawcze algorytmu BA.

Podczas szukania lokalnego, kiedy rozwiązanie jest wybierane ze zbioru najlepszych, może zostać wygenerowane nowe rozwiązanie kandydujące, wg wzoru:

$$x_{new} = x_{old} + \varepsilon\overline{A}^t \tag{10}$$

gdzie: ε jest liczbą losową z zakresu [0,1] i określa nowe rozwiązanie, które jest odmienne lub zbliżone do aktualnego najlepszego rozwiązania, a $\overline{A}^t$ to średnia wartość głośności sygnałów wszystkich nietoperzy.

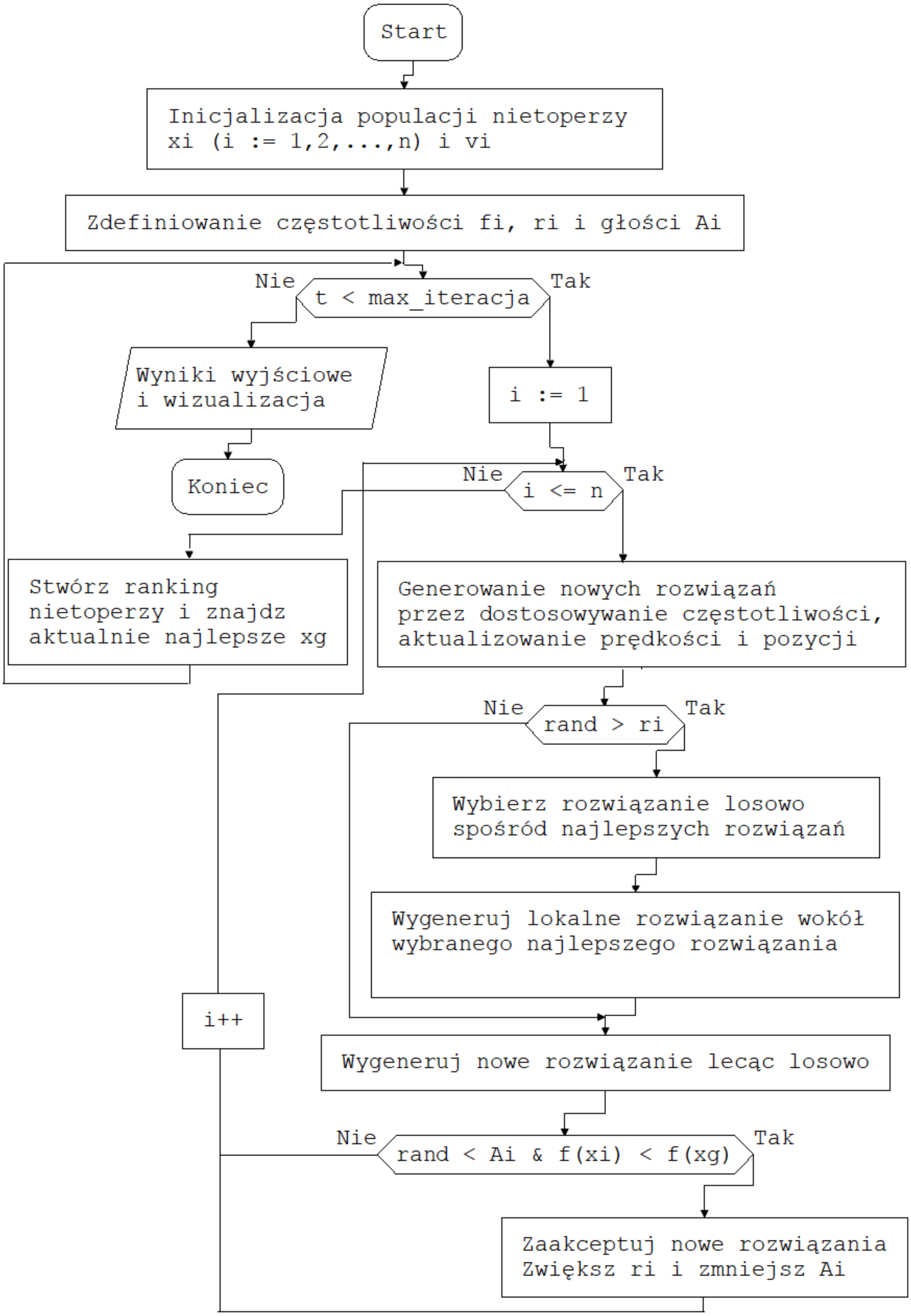
Gdy nietoperz znajdzie cel, stopniowo zmniejsza głośność impulsów i zwiększa szybkość ich emisji, żeby śledzić zdobycz i pochwycić ją. Poziom głośności oraz częstości występowania sygnałów jest aktualizowany w czasie wykonywania procesu, przebiegającego iteracyjnie:

$$A_i^t = \alpha A_i^{t-1}$$

$$r_i^t = r_i^0 + (1 - exp(-\gamma t)) \tag{11}$$

gdzie: α i λ są stałe. Parametr α kontroluje zbieżność algorytmu.

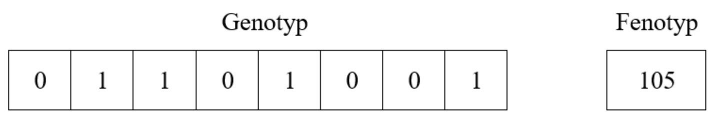
Podstawowe etapy algorytmu BA można przedstawić w postaci schematu blokowego przedstawionego na rys. 1.3.3 [16].



Rysunek 1.3.3: Schemat blokowy Bat algorithm

1.4. Inspiracja naturą: algorytmy ewolucyjne

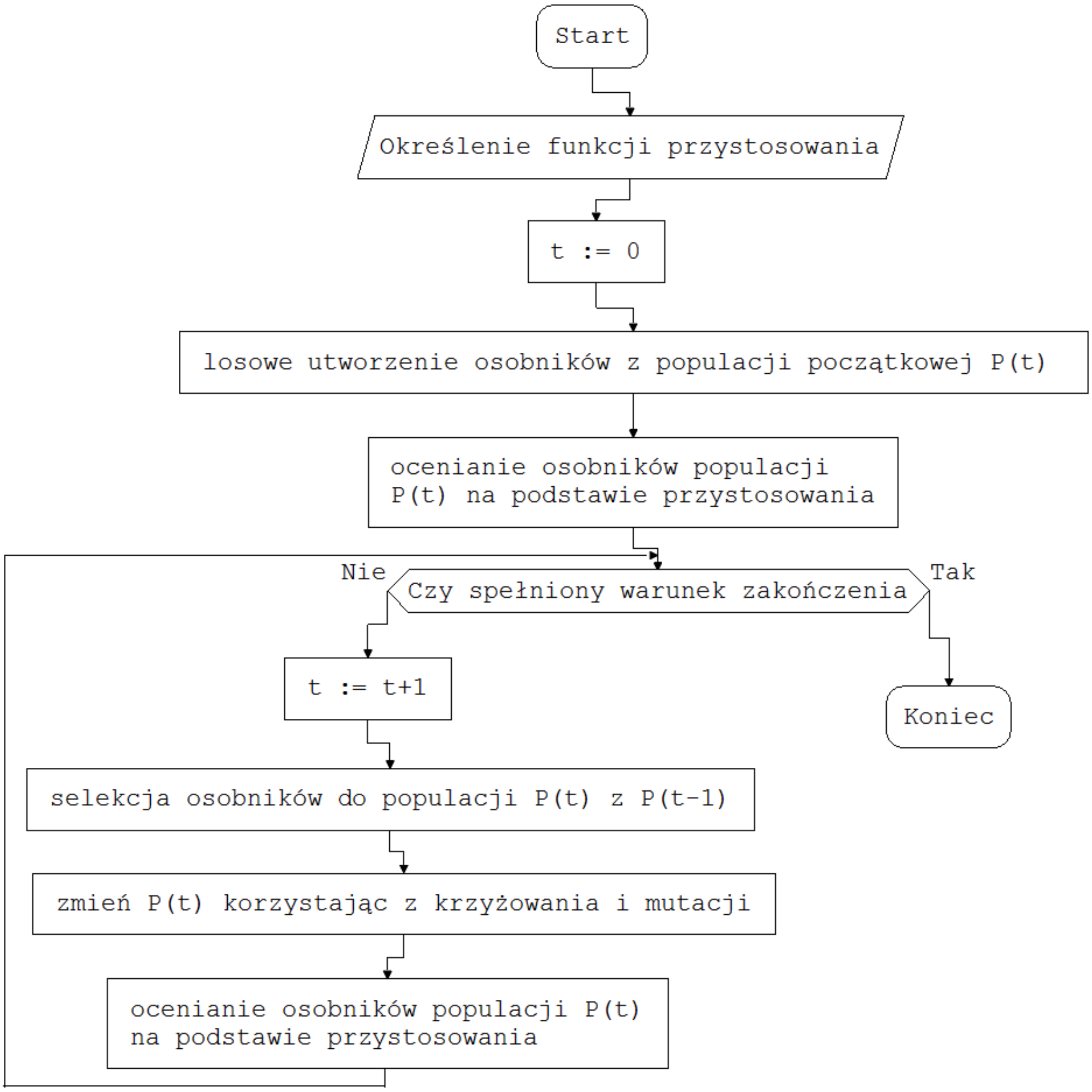
Algorytmy ewolucyjne wywodzą się od procesów zachodzących naturalnie, gdzie procesy szukania rozwiązania zadania są związane z teorią Darwina o selekcji naturalnej. Opis działania algorytmów ewolucyjnych oraz pojęcia związane z tymi algorytmami są ściśle powiązane z ewolucją i genetyką. Uogólniając, algorytm ewolucyjny działa na populacji składającej się z P osobników, z których każdy zawiera chromosom, będący określonym sposobem rozwiązania zadania. Algorytm ewolucyjny działa w przestrzeni, czy środowisku, które jest determinowane przez rodzaj problemu, który ma być przez ten algorytm rozwiązany. Im bardziej określony osobnik jest dostosowany do danego środowiska (ma większe prawdopodobieństwo przeżycia w tym środowisku), tym jakość sposobu rozwiązania problemu, który prezentuje, jest wyższa i otrzymuje lepszą "ocenę". Tak przydzielona ocena jest zwana przystosowaniem osobnika. Wybrany osobnik posiada genotyp, reprezentujący dane w postaci kodu. Z genotypu, dzięki zawartej w nim instrukcji, jest tworzony fenotyp, czyli już odkodowane, możliwe rozwiązanie zadania. Fenotypy również muszą być oceniane w środowisku. Odbywa się zatem kodowanie fenotypu wykonywane przez genotyp (czasem, w niektórych algorytmach ewolucyjnych pojęcie fenotypu jest tożsame z genotypem). Streszczając, fenotyp to punkt znajdujący się w przestrzeni zawierającej rozwiązania problemu, a genotyp to punkt w przestrzeni zawierającej kody. Wzorcowego osobnika o binarnej reprezentacji genotypu i jego fenotyp przedstawiono na rys. 1.4.4.



Rysunek 1.4.4: Osobnik wzorcowy z genotypem i fenotypem

Środowisko może zostać scharakteryzowane funkcją przystosowania, dzięki której mając na uwadze fenotyp osobnika, przypisywane jest mu przystosowanie. Chromosomy osobników zaś, zbudowane są z cząstek nazywanych genami. Wyróżniania się także allele, czyli wartości danego genu. W przedstawionym wyżej przykładzie allelami są 1 i 0.

Na rysunku 1.4.5 przedstawiono schemat blokowy algorytmu ewolucyjnego. Po strukturze tego algorytmu widać, że należy on do algorytmów probabilistycznych, którego działanie polega na utworzeniu populacji osobników $P(t) = x_1^t, ..., x_n^t$ dla każdej iteracji t. Poszczególne osobniki zawierają różne sposoby rozwiązania problemu i zwykle występują jako chromosomy jednowarstwowe, czyli struktury S. Aby ocenić poszczególne rozwiązania x_i^t wprowadza się jakąś skalę przystosowania chromosomu. Wobec tego w tzw. fazie selekcji (iteracja $t + 1$) generowana jest nowa populacja wyselekcjonowana z najlepszych osobników.



Rysunek 1.4.5: Algorytm ewolucyjny na schemacie blokowym

W kolejnej fazie, fazie zmiany, niektóre osobniki poddawane są transformacji przez operatory genetyczne, rezultatem czego pojawiają się nowe rozwiązania. Wyróżnia się transformacje jednoargumentowe m_i, polegające na tworzeniu osobników dzięki niewielkiej zmianie jednego osobnika (mutacja) oraz transformacje o wielu argumentach (wieloargumentowe c_j), polegające na tworzeniu osobników (przyjmujących postać jednowarstwowych chromosomów) dzięki złożeniu fragmentów kilku osobników[17]. Program napisany na podstawie algorytmu wykonuje kilka kroków generacji, ilość sensownych rozwiązań maleje, a rozwiązanie prezentowane przez osobniki najlepsze jest bardzo zbliżone do optymalnego.

1.4.1. Algorytmy ewolucyjne: Algorytmy genetyczne

Genetyczne algorytmy, których autorem jest John Holland są najpopularniejsze spośród algorytmów, które powstały na wzór procesów zachodzących w naturze. Holland poprzez algorytmy genetyczne starał się zrozumieć i rozjaśnić, w jaki sposób organizmy dostosowują się do zmian zachodzących w środowisku.

2. Program do optymalizacji i omówienie funkcji

2.1. Metody i środowisko wykorzystane podczas tworzenia programu

Podczas projektowania programu optymalizującego funkcje wykorzystano metody napisane w języku *R* [24]. W przypadku większości algorytmów skorzystano z pakietów w repozytorium *CRAN*, które uzupełniono o mechanizmy umożliwiające kontrolę pracy algorytmu np. pomiar czasu. Jako środowisko programistyczne wykorzystano IDE *RStudio* [25]. Wybrano następujące metody, poszukujące minimum funkcji, w ograniczonym zakresie wartości zmiennych:

1. Algorytmy rojowe:
   - Particle Swarm Optimization (pakiet *psoptim* [26]),
   - Bat algorithm (pakiet *microbats* [27]).
2. Algorytmy ewolucyjne:
   - Genetic Algorithm (pakiet *GA* [28]),
   - Differential Evolution (pakiet *DEoptim* [29]).

Wybór celów optymalizacji do testów ograniczono do 13 funkcji z wieloma ekstremami lokalnymi i jednym ekstremum globalnym. Dla każdej z tych funkcji i dla każdej liczebności zbioru poszukującego rozwiązania wykonano po 50 prób optymalizacji każdym algorytmem($10x5x50x5$). Obliczono błędy średniokwadratowe znalezionych minimów i ich odchylenie standardowe. Na podstawie analizy prób uzyskano wyniki średnie, które umieszczono w tabelach. Dodano także wykresy procesu poszukiwania rozwiązania dla każdej funkcji oraz wykresy znajdowanych minimów w stosunku do iteracji.

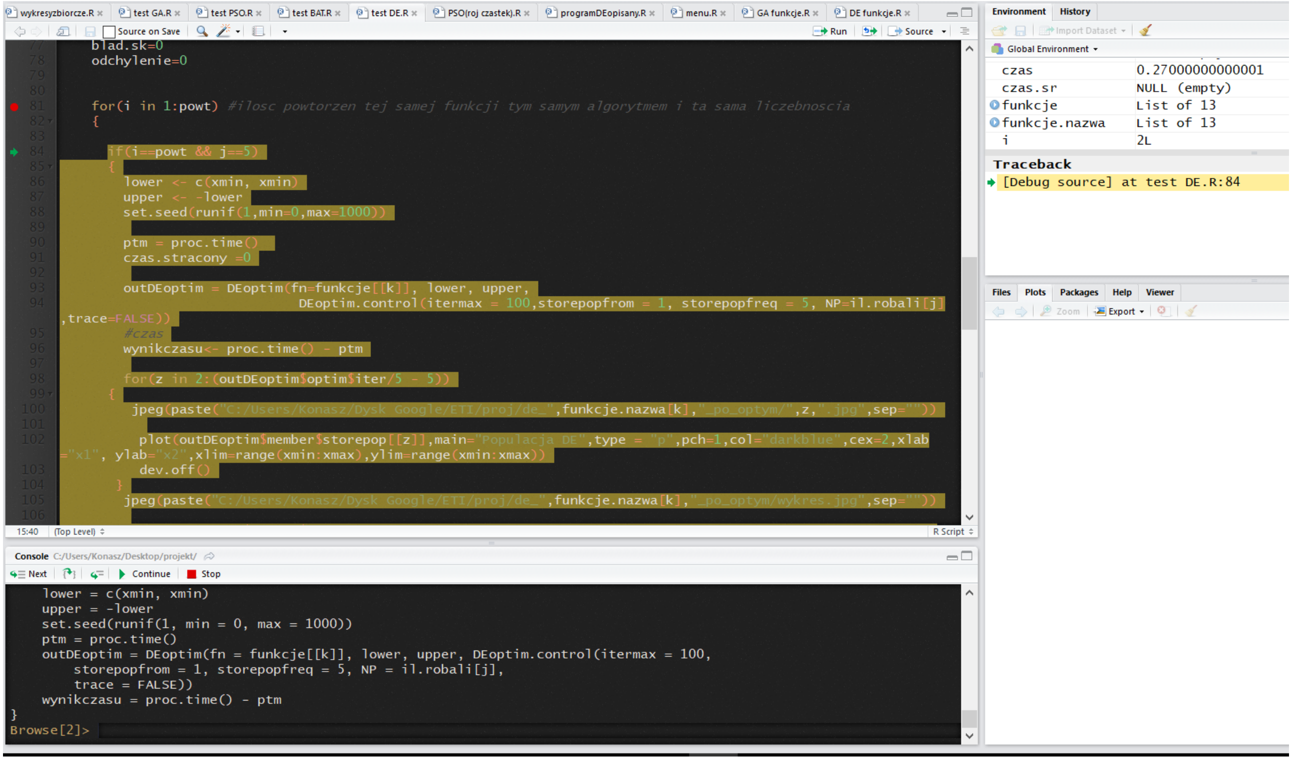
2.2. Narzędzia programistyczne wykorzystane w programie

2.2.1. Walidacja

```
Walidację wprowadzonych wartości zastosowano przy wyborze
   algorytmu i~funkcji optymalizowanej. Poniżej podano przykład
   funkcji sprawdzającej poprawność wyboru algorytmu:
algorytmy <- function() #Poprawiony wybór algorytmu
{
  algor <- readline("Wybrany algorytm: ")
  if(!grepl("^[0-9]+\$",algor))                    #walidacja
     danych wprowadzonych przez użytkownika
  {                                         #wyrażenie regularne
     sprawdza czy wprowadzony cyfrę
    cat("Niewlasciwy wybor")
    return(algorytmy())
  }
```

2.2.2. Debugger

Z debuggera korzystano podczas pisania głównego programu do wielokrotnych testów poszczególnymi algorytmami, sprawdzano za jego pomocą poprawność działania kodu w pętlach. Na 2.2.6 rysunku przedstawiono proces sprawdzania krok po kroku działania pętli, która zadaną ilość razy wykonuje metodę SI.



Rysunek 2.2.6: Debugger Rstudio w działaniu

2.2.3. Testy

Wielokrotne testy jednostkowe w formie eksperymentu dla wszystkich algorytmów i metod zostały przedstawione w rozdziale 3.

2.3. Opis kodów programów wykorzystujących możliwości metod sztucznej inteligencji

Celem programu jest znalezienie minimum danej funkcji używając wybranego przez użytkownika algorytmu.

Na początek uruchamiany jest plik menu.R zawiera on interfejs użytkownika pozwalający na wybór algorytmu, funkcji oraz ilości powtórzeń. Wybory dokonane przez użytkownika odczytywane są i sprawdzane przy użyciu funkcji z pliku /testy/wybory.R. Po dokonaniu wyborów uruchamiany jest odpowiedni algorytm, który wyszukuje minimum, przygotowuje rysunek i zapisuje wynik pracy w strukturze języka R *date.table*. Po zadanej ilości powtórzeń dane eksportowane są do tabel w programie MS Excel.

2.3.1. Opis kodu menu.R

```
source('testy/BAT zrobione funkcje.R')
source('testy/DE funkcje.R')
source('testy/GA funkcje.R')                                    #
    dołączenie niezbędnych funkcji
source('testy/PSO funkcje.R')
source('testy/wybory.R')



print("Wybierz algorytm do optymalizacji")
print("1.BAT - alg.nietoperza")
print("2.DE - alg.ewolucji roznicowej")                 #lista
    dostępnych algorytmów
print("3.GA - alg.genetyczny")
print("4.PSO - alg.roju czastek")

#Wywolanie funkcji wyboru algorytmu
algorytmy()

cat("\n")
print("Lista dostepnych funkcji")                       #lista
    dostępnych funkcji
print("1. Ackley")
print("2. Beale")
print("3. Goldstein")
print("4. Bartels-Conn")
print("5. Leon")
print("6. Eggholder")
print("7. Venter")
print("8. Matyas")
print("9. Zirilli")
print("10. Easom")
print("11. Rastrigin")
```

```
print("12. Levin13")
print("13. Drop Wave")


#Wywolanie funkcji wyboru ilosci powtorzen i funkcji
wybor_funkcji()
powtorzenia()

if (alg==1){                                  #wywołanie wybranego
    algorytmu
  source('testy/test_BAT.R')
} #BAT

if(alg==2){

  source('testy/test_DE.R')
} #DE

if (alg==3){

  source('testy/test_GA.R')
} #GA

if (alg==4){

  source('algorytmy_optymalizacyjne/PSO(roj_czastek).R')
  source('testy/test_PSO.R')
} #PSO
```

2.3.2. Opis kodu wybory.R

```
algorytmy <- function() #Poprawiony wybór algorytmu
```

```
{
  algor <- readline("Wybrany algorytm: ")
  if(!grepl("^[0-9]+\$", algor))                     #walidacja
     danych wprowadzonych przez użytkownika
  {                                             #wyrażenie regularne
     sprawdza czy wprowadzony cyfrę
    cat("Niewlasciwy wybor")
    return(algorytmy())
  }

  alg <<- as.integer(algor)

  if(alg>4 || alg<1)                            #sprawdzenie czy podana
     cyfra należy do odpowiedniego przedziału
  {                                             #z racji istnienie 4
     algorytmów liczby
    cat("Niewlasciwy wybor")                    # większe od 4 nie
       odpowiadają żadnemu możliwemu wyborowi
    return(algorytmy())
  }
}

wybor_funkcji <- function() #Wybor funkcji z listy
{
  wyb <- readline("Podaj numer wybranej funkcji: ")#walidacja
     danych wprowadzonych przez użytkownika
  if(!grepl("^[0-9]+\$",wyb))                        #wyrażenie
     regularne sprawdza czy wprowadzony cyfrę
  {
    cat("Niewlasciwy wybor")
    return(wybor_funkcji())
  }
```

```
  wybor <<- as.integer(wyb)

  if(wybor>13 || wybor<1)                          #sprawdzenie czy
      podana cyfra należy do odpowiedniego przedziału
  {
    cat("Niewlasciwy wybor")
    return(wybor_funkcji())
  }
}

powtorzenia <- function() #Wybor ilosci powtorzen
{
  p <- readline("Podaj ilosc powtorzen: ")          #walidacja
     danych wprowadzonych przez użytkownika
  if(!grepl("^[0-9]+\$",p))                        #wyrażenie
     regularne sprawdza czy wprowadzony cyfrę
  {
    return(powtorzenia())
  }
  powt <<- as.integer(p)                           #minimalna ilość
      powtórzeń wynosi 10
  if(powt<10)
  {
    cat("Ilosc powtorzen musi wynosic co najmniej 10")
    return(powtorzenia())
  }
}
```

2.3.3. Opis kodu programu głównego

Przedstawiony kod zawiera część programu odpowiadającą za przeprowadzenie wiarygodnego eksperymentu na metodach SI na wybranych funkcjach i umożliwia jego kontrolę, a efektem jego działania są gotowe uśrednione wyniki wielokrotnie powtórzonego eksperymentu wraz z wykresami. Poniższy kod dotyczy algorytmu DE, ale w przypadku innych algorytmów metoda jest podobna i różni się głównie funkcją wywołującą algorytm.

```
library(DEoptim)          #uruchomienie pakietu z wybranym
    algorytmem
library(data.table)       #uruchomienie pakietu z tabela
    przechowujaca wyniki w R
library(xlsx)             #pakiet eksportujacy tabele do MS Excel

# funkcje optymalizowane w programie, w nawiasie podane
    ogranicznia szukanych niewiadomych i szukanego minimum
funkcje=list(ackley.pso,#[-32,32] min 0 at(0,0)
             beale.pso,#[-4.5,4.5] min 0 at (3,0.5)
             goldstein.pso,# [-2,2] min 3 at (0,-1)
             bartels.conn.pso,# [-500,500] min 1 at (0,0)
             leon.pso,# [-1.2,1.2] min 0 at (1,1)
             eggholder.pso,# [-512,512] min -959 at (512,404)
             venter.pso,# [-50,50] min -400 at (0,0)
             matyas.pso,# [-10,10] min 0 at (0,0)
             zirilli.pso,#[-10,10] min -0.35 at (-1.04,0)
             easom.pso,# [-100,100] min -1 at (3.14,3.14)
             rastrigin.pso,# x[-5.12,5.12] min 0 at (0,0)
             levin13.pso,# [-10,10] min 0 at (1,1)
             drop_wave.pso# [-5.12,5.12] min -1 at (0,0)
)

funkcje.nazwa=list("ackley.pso",#[-32,32] min 0 at(0,0)
                   "beale.pso",#[-4.5,4.5] min 0 at (3,0.5)
                   "goldstein.pso",# [-2,2] min 3 at (0,-1)
                   "bartels.conn.pso",# [-500,500] min 1 at
                      (0,0)
                   "leon.pso",# [-1.2,1.2] min 0 at (1,1)
                   "eggholder.pso",# [-512,512] min -959 at
```

```
                    (512,404)
                  "venter.pso",# [-50,50] min -400 at (0,0)
                  "matyas.pso",# [-10,10] min 0 at (0,0)
                  "zirilli.pso",#[-10,10] min -0.35 at
                    (-1.04,0)
                  "easom.pso",# [-100,100] min -1 at
                    (3.14,3.14)
                  "rastrigin.pso",# x[-5.12,5.12] min 0 at
                    (0,0)
                  "levin13.pso",# [-10,10] min 0 at (1,1)
                  "drop_wave.pso"# [-5.12,5.12] min -1 at (0,0)
)

zakres=c(32,4.5,2,500,1.2,512,50,10,10,100,5.12,10,5.12)#wektor
    ograniczen szukanych niewiadomych i szukanego minimum
min=c(0,0,3,1,0,-959,-400,0,-0.35,-1,0,0,-1) # szukane minima
    kolejnych funkcji

il.robali <- c(30,60,80,100,150) # liczebnosci rojow

for(k in wybor:wybor)      # optymalizacja funkcji w zaleznosci
    od wyboru
{

     xmin <- -zakres[k]
     xmax <- zakres[k]



  x1=c()        # wektory potrzebne do przechowywania wynikow
  x2=c()
  minimum=c()
  czas=c()
```

```
iter=c()
tabele=list()
blad.sk=c()
odchlenie=c()

x1.sr=c()
x2.sr=c()
minimum.sr=c()
czas.sr=c()
iter.sr=c()
blad.sk.sr=c()
odchylenie.sr=c()

for(j in 1:5) # iteracja dla kazdej z liczebnosci rojow
{
  x1=0 # zerowanie wynikow
  x2=0
  minimum=0
  czas=0
  iter=0
  blad.sk=0
  odchylenie=0


  for(i in 1:powt) # ilosc iteracji uzalezniona od liczby
     powtorzen
    if(i==powt && j==5) # jesli liczebnosc roju wynosi 200 i
       powtarzana jest ostatnia iteracja to generowane sa
       wykresy
    {
      lower <- c(xmin, xmin)
      upper <- -lower
```

```
set.seed(runif(1,min=0,max=1000))

ptm = proc.time()
czas.stracony =0

outDEoptim = DEoptim(fn=funkcje[[k]], lower, upper,
                    DEoptim.control(itermax = 100,
                       storepopfrom = 1, storepopfreq =
                        5, NP=il.robali[j],trace=FALSE)
                       ) #wywolanie algorytmu DE
#czas
wynikczasu<- proc.time() - ptm

for(z in 2:(outDEoptim$optim$iter/5 - 5))
{
  jpeg(paste("C:/Users/Konasz/Dysk Google/ETI/proj/de_",
     funkcje.nazwa[k],"_po_optym/",z,".jpg",sep="")) #
     sciezka pod jaka zapisywany bedzie wykres roju

  plot(outDEoptim$member$storepop[[z]],main="Populacja
     DE",type = "p",pch=1,col="darkblue",cex=2,xlab="x1"
     , ylab="x2",xlim=range(xmin:xmax),ylim=range(xmin:
     xmax))                   #rysowanie wykresu
  dev.off()
}
jpeg(paste("C:/Users/Konasz/Dysk Google/ETI/proj/de_",
   funkcje.nazwa[k],"_po_optym/wykres.jpg",sep=""))#
   sciezka do wykresu zaleznosci znalezionego minimum od
    iteracji

plot(outDEoptim$member$bestvalit, type = 'o', col = '
   black',xlab="Iteracje", ylab="Wartosci minimum")     #
```

```
        rysowanie wykresu

      dev.off()

    }

  else{ # opcja bez tworzenia wykresow

  {

      lower = c(xmin, xmin)
      upper = -lower

      set.seed(runif(1,min=0,max=1000))

      ptm = proc.time() # czas poczatkowy

      outDEoptim = DEoptim(fn=funkcje[[k]], lower, upper, #
        wywolanie algorymu DE
                           DEoptim.control(itermax = 100,
                              storepopfrom = 1, storepopfreq =
                               5, NP=il.robali[j],trace=FALSE)
                              )

      wynikczasu= proc.time() - ptm # czas = czas teraz- czas
        przed wykonaniem DE
  }

    x1[i] = outDEoptim$optim$bestmem[1]# znaleziona
       wspolrzedana x1
    x2[i] = outDEoptim$optim$bestmem[2]# x2
    minimum[i] = outDEoptim$optim$bestval # znalezione minimum
```

```
      funkcji
   czas[i] = wynikczasu[3]          # czas wykonania
      powtorzenia
   iter[i] = outDEoptim$optim$iter   # ilosc iteracji
      algorytmu do wyniku

   blad.sk[i] = (minimum[i] + min[k])^2#blad
      sredniokwadratowy

}

odchylenie[j] = sd(minimum)#odchylenie standardowe

x1[powt+1] = mean(x1) #obliczanie srednich z 50 prob
x2[powt+1] = mean(x2)
minimum[powt+1] = mean(minimum)
czas[powt+1] = mean(czas)
iter[powt+1] = mean(iter)
blad.sk[powt+1] = mean(blad.sk)

x1.sr[j]=x1[powt+1]
x2.sr[j]=x2[powt+1]
minimum.sr[j]=minimum[powt+1]
czas.sr[j]=czas[powt+1]
iter.sr[j]=iter[powt+1]
blad.sk.sr[j]=blad.sk[powt+1]

odchylenie.sr[j]=odchylenie[j]

#wprowadzanie danych z wektorow do tabeli w R data table
tabele[[j]] <- data.table(Lp=c(1:powt,"Srednie"),Wielkosc.
    roju=il.robali[j],Znalezione.x1=x1,Znalezione.x2=x2,
```

```
        Minimum=minimum,MSE.od.minimum=blad.sk,Odchylenie.
        standardowe=odchylenie[j], Iteracje=iter, Czas=czas,"")
    print(tabele[[j]])

    cat("\n\n")
  }
  cat("TABELA SREDNICH Z 1 FUNKCJI", "\n\n")
  tabela.srednich <- data.table(Nr_sredniej=c(1:5),Wielkosc.roju
     =il.robali,Znalezione.x1=x1.sr,Znalezione.x2=x2.sr, Minimum
     =minimum.sr,MSE.od.minimum=blad.sk.sr,Odchylenie.
     standardowe=odchylenie.sr, Iteracje=iter.sr, Czas=czas.sr)
  # tworzenie tabeli srednich wynikow
  print(tabela.srednich)

  write.xlsx(tabele,paste("D:/PSO/DE_",funkcje.nazwa[[k]],".xlsx
     "))
  #eksport do Excela
  write.xlsx(tabela.srednich,paste("D:/PSO/sr_de_",funkcje.nazwa
     [[k]],".xlsx"))

  rm(x1,x2,minimum,czas,iter,tabele,blad.sk,x1.sr,x2.sr,minimum.
     sr,czas.sr,iter.sr,blad.sk.sr,odchylenie.sr)#usuwa
     niepotrzebne wektory
}
```

2.4. Opis wywołań metod SI i funkcji rysujących w języku R użytych w programie

Zostanie przedstawiony opis poszczególnych argumentów funkcji wywołujących metody optymalizacyjne w R oraz przykład wyniku działania tych metod na przykładzie funkcji *Ackley* i algorytmu *PSO*. Wywołanie różnych algorytmów wygląda różnie, jednak ogólny wzorzec jest podobny. Przedstawiono też fragmenty kodu, odpowiedzialne za generowanie wykresów 2D i 3D.

2.4.1. PSO

Optymalizację przeprowadzono z użyciem pakietu *psoptim*. Oto przykład formuły wywołującej tę metodę:

```
library(pso)
ptm <- proc.time()
psoptim(FUN=ackley.pso, n <- 200, max.loop<- 100, w<- 0.95, c1<-
    0.2, c2<- 0.2,
        xmin <- c(-32, -32), xmax<- c(32, 32), vmax<- c(4, 4),
          seed=runif(1,min=0,max=1000),anim=FALSE)
czas<- proc.time() - ptm
print(czas[3])
```

gdzie: FUN-nazwa funkcji, n-liczba cząstek w roju, max.loop-maksymalna liczba iteracji, w-współczynnik bezwładności, c2- współczynnik własnego „zaufania”, c1-współczynnik „zaufania” do roju, xmin-wektor określający dolne ograniczenia wartości zmiennych, xmax-wektor określający górne ograniczenia wartości zmiennych, vmax-wektor ograniczeń prędkości w każdym kierunku, seed-liczba określająca tzw. ziarno dla generatora liczb pseudolosowych, anim-wartość logiczna informująca o tym czy wykresy 2D mają być generowane, ptm,czas-wartości odpowiadające za pomiar czasu wykonania programu

Wynik działania metody na funkcji Ackley:

```
$sol
                x1           x2
[1,] 0.0009598709 0.0006415018
$val
[1] -0.003300919
$loop
[1] 76
elapsed
   0.26
```

gdzie: sol-znalezione wartości niewiadomych x1,x2, val-wartość minimum funkcji, loop-ilość przebiegów, elapsed-czas wykonania

2.4.2. Wykresy 2D z pozycjami osobników

W przypadku PSO wykresy 2D zostały wygenerowane za pomocą funkcji *contour* i *points*. *contour* odpowiada za wyświetlanie tła, przedstawiającego zarys optymalizowanej funkcji rzutowanej z góry, zaś *points* rysuje pozycje osobników:

```
contour(x_image[,1], x_image[,2], z, nlevels=20, xlab="x1", ylab
    ="x2", col="darkgray", main=paste(loop))
points(x, xlim=c(xmin[1], xmax[1]), ylim=c(xmin[2], xmax[2]),
    pch=21, bg="cadetblue")
```

gdzie: *x_image*[, 1], *x_image*[, 2]- macierze zawierające informacje na temat rzutowanego obrazu

W pozostałych metodach wykorzystano funkcję *plot* do rysowania pozycji członków populacji:

```
plot(Sol,main=t,type = "p",pch=19, col="darkred", cex = 2, xlab=
    "x1", ylab="x2",xlim=range(Lower:Upper),ylim=range(Lower:
    Upper))
```

gdzie: Sol(różne nazwy w zależności od metody) - macierz zawierająca współrzędne członków populacji

Funkcje tworzenia wykresów 2D znajdują się wewnątrz metod optymalizacyjnych i były wywoływane poprzez wartość logiczną $anim = TRUE$.

2.4.3. Wykresy 2D zależności znalezionego minimum od iteracji

Wykresy 2D przedstawiające wartość minimum w z biegiem iteracji są generowane dzięki funkcji *plot* przedstawionej w poniższym fragmencie kodu:

```
plot(wynik,type = "o",pch=19, col="darkblue", xlab="Iteracje",
    ylab="Znalezione_minimum_funkcji")
```

gdzie: wynik(różne nazwy w zależności od metody) - wektor zawierający wyniki najlepszego znalezionego minimum w kolejnych iteracjach

2.4.4. Wykresy 3D funkcji

Wykresy 3D wygenerowano za pomocą funkcji *persp* należącej do pakietu *GA*. Poniżej zaprezentowano kod wywołujący tę funkcję rysującą:

```
x1 <- x2 <- seq(xmin, xmax, by = 0.1)
f <- outer(x1, x2, nazwa)
persp3D(x1, x2, f, theta = 50, phi = 20)
```

gdzie: x1,x2 - wektory liczb od minimum przedziału do maksimum, skok co 0.1, xmin,xmax-minimalna wartość ograniczeń wartości zmiennych oraz wartość maksymalna, nazwa-nazwa funkcji

2.5. Opis funkcji użytych w eksperymencie/testach

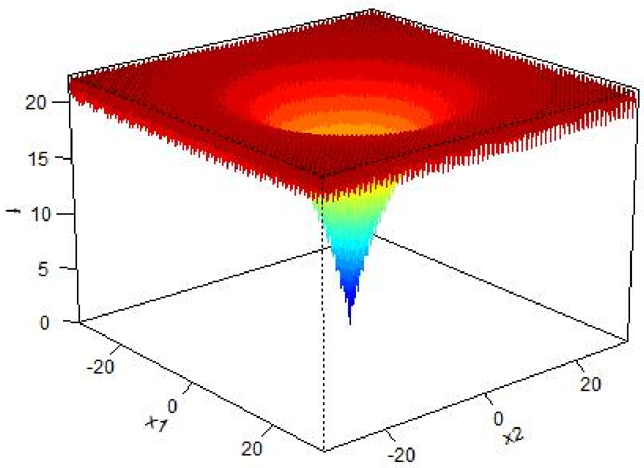
Podczas badań optymalizowano 13 funkcji o dwu niewiadomych [30, 31] szukając ich minimum globalnego w określonym przedziale wartości zmiennych. W tabeli 2.5.1 umieszczono nazwy wykorzystanych funkcji, ograniczenia wartości zmiennych, poszukiwane wartości niewiadomych i poszukiwana wartość minimum globalnego.

Tabela 2.5.1: Lista funkcji użytych do eksperymentu

Lp.	Nazwa funkcji	Zakres wartości x1 i x2	Poszukiwane wartości x1 i x2	Poszukiwana wartość f(x1,x2)
1	Ackley	[-32, 32]	(0, 0)	0
2	Beale	[-4.5, 4.5]	(3, 0.5)	0
3	Goldstein	[-2, 2]	(0, -1)	3
4	Bartels Conn	[-500, 500]	(0, 0)	1
5	Leon	[-1.2, 1.2]	(1, 1)	0
6	Eggholder	[-512, 512]	(512, 404)	-959
7	Venter	[-50, 50]	(0, 0)	-400
8	Matyas	[-10, 10]	(0, 0)	0
9	Zirilli	[-10, 10]	(-1.04 ,0)	-0,35
10	Easom	[-100, 100]	(3.14, 3.14)	-1
11	Rastrigin	[-5.12, 5.12]	(0, 0)	0
12	Levy N.13	[-10, 10]	(1, 1)	0
13	Drop Wave	[-5.12, 5.12]	(0, 0)	-1

W kolejnych podsekcjach przedstawiono poszczególne funkcje, ich wykresy 3D (rysunki 2.5.7 - 2.5.19), postać matematyczną oraz kod źródłowy w języku R.

2.5.1. Ackley



Rysunek 2.5.7: Wykres 3D funkcji Ackley

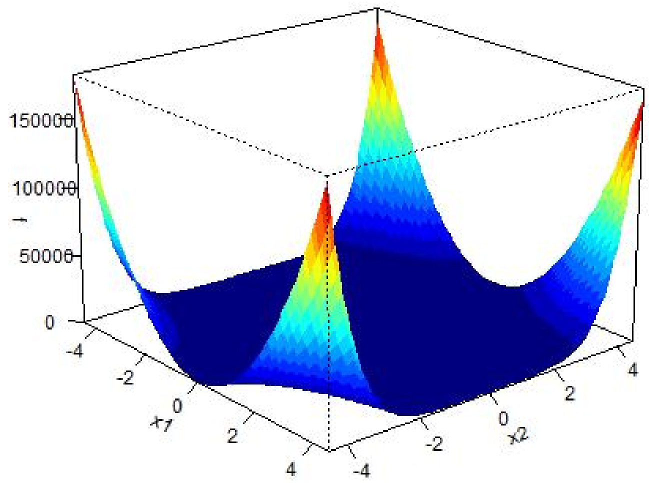
Postać matematyczna funkcji:

$$f(x) = -20exp(-0.2 \cdot \sqrt{0.5 \cdot (x_1^2 + x_2^2)}) - exp(0.5 \cdot (cos(2\pi x_1) + cos(2\pi x_2))) + e + 20 \quad (12)$$

Kod źródłowy:

```
ackley = function(x1,x2)
  -20*exp(-0.2*sqrt(0.5*(x1^2+x2^2)))-exp(0.5*(cos(2*pi*x1)+cos
      (2*pi*x2))) + exp(1) + 20
```

2.5.2. Beale



Rysunek 2.5.8: Wykres 3D funkcji Beale

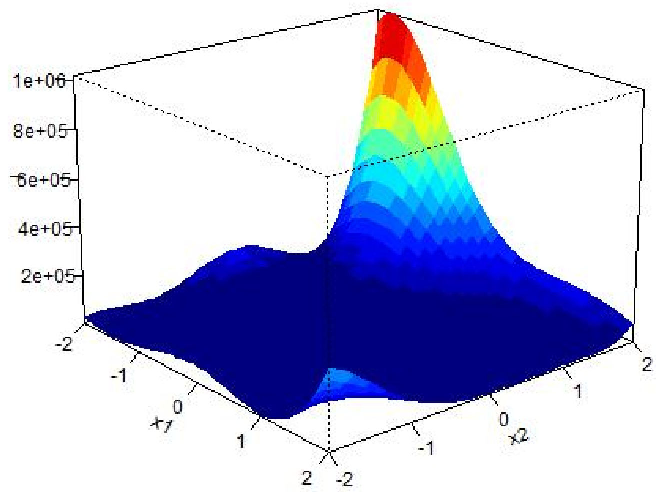
Postać matematyczna funkcji:

$$f(x) = (1.5 - x_1 + x_1x_2)^2 + (2.25 - x_1 + x_1x_2^2)^2 + (2.625 - x_1 + x_1x_2^3)^2 \tag{13}$$

Kod źródłowy:

```
beale = function(x1,x2)
  (1.5-x1+x1*x2)^2 + (2.25-x1+x1*x2^2)^2 +(2.625-x1+x1*x2^3)^2
```

2.5.3. Goldstein



Rysunek 2.5.9: Wykres 3D funkcji Goldstein

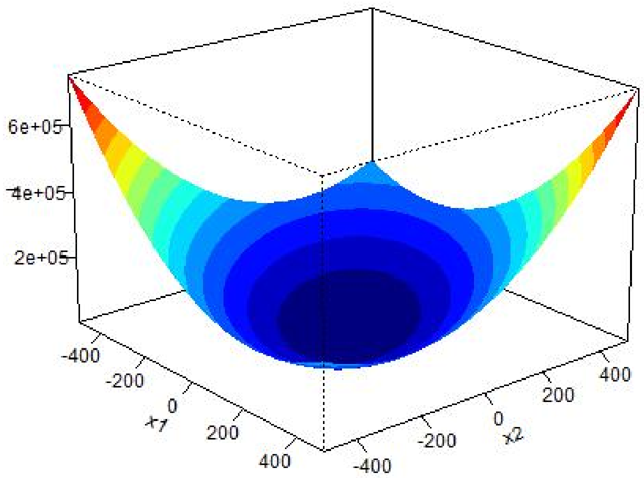
Postać matematyczna funkcji:

$$f(x) = (1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2))$$
$$(30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)) \quad (14)$$

Kod źródłowy:

```
goldstein = function(x1,x2)
  (1+(x1+x2+1)^2 * (19-14*x1+3*x1^2 - 14*x2+6*x1*x2+3*x2^2))*
     (30+(2*x1-3*x2)^2 * (18-32*x1+12*x1^2 + 48*x2-36*x1*x2+27*
     x2^2))
```

2.5.4. Bartels Conn



Rysunek 2.5.10: Wykres 3D funkcji Bartels Conn

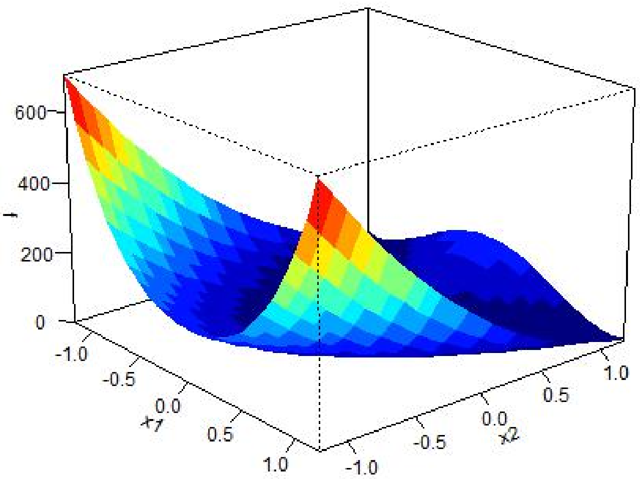
Postać matematyczna funkcji:

$$f(x) = \mid x_1^2 + x_1^2 + x_1 x_2 \mid + \mid sin(x_1) \mid + \mid cos(x_2) \mid \tag{15}$$

Kod źródłowy:

```
bartels.conn = function(x1,x2)
  abs(x1^2 + x2^2 + x1*x2)+abs(sin(x1))+abs(cos(x2))
```

2.5.5. Leon



Rysunek 2.5.11: Wykres 3D funkcji Leon

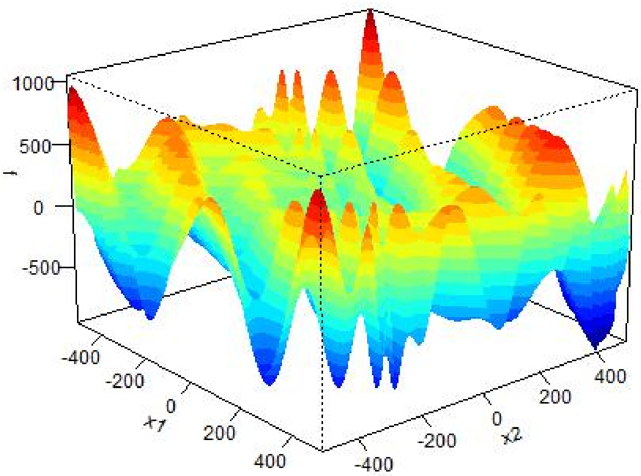
Postać matematyczna funkcji:

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \tag{16}$$

Kod źródłowy:

```
leon = function(x1,x2)
  100*(x2-x1^2)^2 + (1-x1)^2
```

2.5.6. Eggholder



Rysunek 2.5.12: Wykres 3D funkcji Eggholder

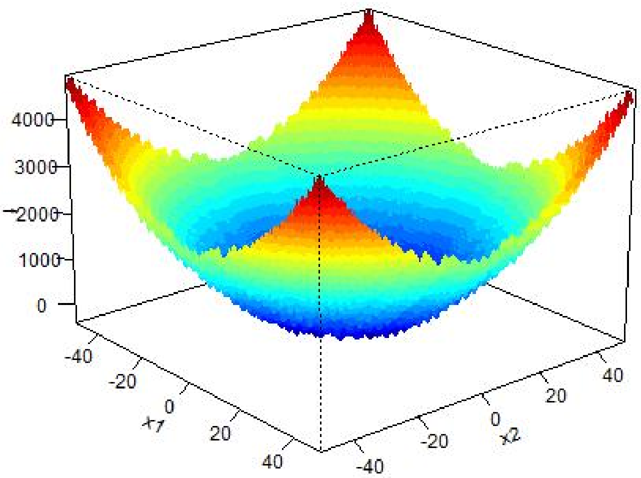
Postać matematyczna funkcji:

$$f(x) = -(x_2 + 47) sin\sqrt{\mid \frac{x_1}{2} + (x_2 + 47) \mid} - x_1 sin\sqrt{\mid x_1 - (x_2 + 47) \mid} \tag{17}$$

Kod źródłowy:

```
eggholder = function(x1, x2)
  -(x2+47)*sin(sqrt(abs(x2+x1/2+47)))-x1*sin(sqrt(abs(x1-(x2+47)
      )))
```

2.5.7. Venter



Rysunek 2.5.13: Wykres 3D funkcji Venter

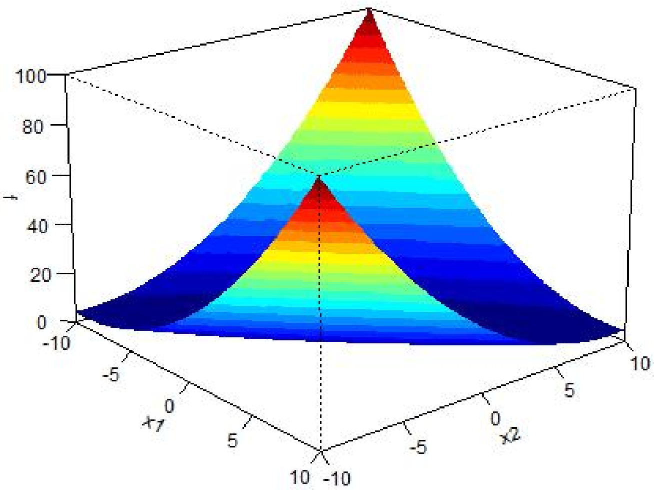
Postać matematyczna funkcji:

$$f(x) = x_1^2 - 100cos(x_1)^2 - 100cos(x_1^2/30) + x_2^2 - 100cos(x_2)^2 - 100cos(x_2^2/30) \quad (18)$$

Kod źródłowy:

```
venter = function(x1,x2)
  x1^2 - 100*cos(x1)^2-100*cos(x1^2/30)+x2^2-100*cos(x2)^2-100*
    cos(x2^2/30)
```

2.5.8. Matyas



Rysunek 2.5.14: Wykres 3D funkcji Matyas

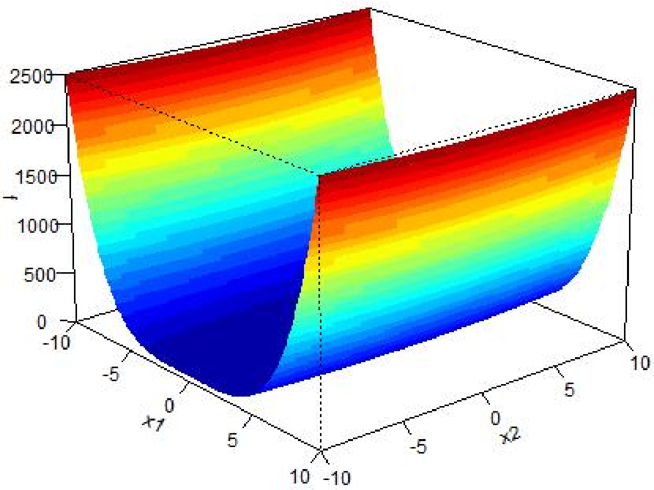
Postać matematyczna funkcji:

$$f(x) = 0.26(x_1^2 + x_2^2) - 0.48x_1x_2 \tag{19}$$

Kod źródłowy:

```
matyas = function(x1,x2)
  0.26*(x1^2+x2^2)-0.48*x1*x2
```

2.5.9. Zirilli



Rysunek 2.5.15: Wykres 3D funkcji Zirilli

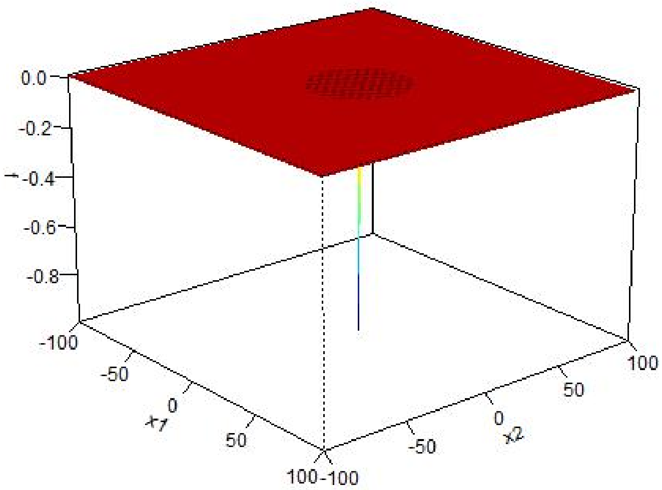
Postać matematyczna funkcji:

$$f(x) = 0.25x_1^4 - 0.5x_1^2 + 0.1x_1 + 0.5x_2^2 \tag{20}$$

Kod źródłowy:

```
zirilli = function(x1,x2)
  0.25*x1^4 - 0.5*x1^2 + 0.1*x1 + 0.5*x2^2
```

2.5.10. Easom



Rysunek 2.5.16: Wykres 3D funkcji Easom

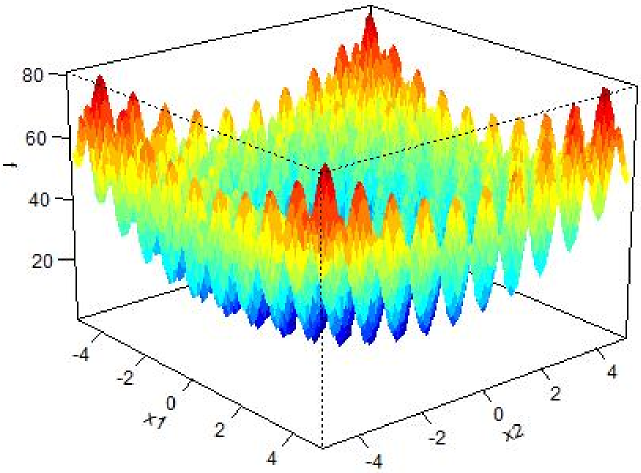
Postać matematyczna funkcji:

$$f(x) = -cos(x_1)cos(x_2)exp(-((x_1 - \pi)^2 + (x_2 - \pi)^2)) \tag{21}$$

Kod źródłowy:

```
easom = function(x1,x2)
  -cos(x1)*cos(x2)*exp(-((x1-pi)^2 + (x2-pi)^2))
```

2.5.11. Rastrigin



Rysunek 2.5.17: Wykres 3D funkcji Rastrigin

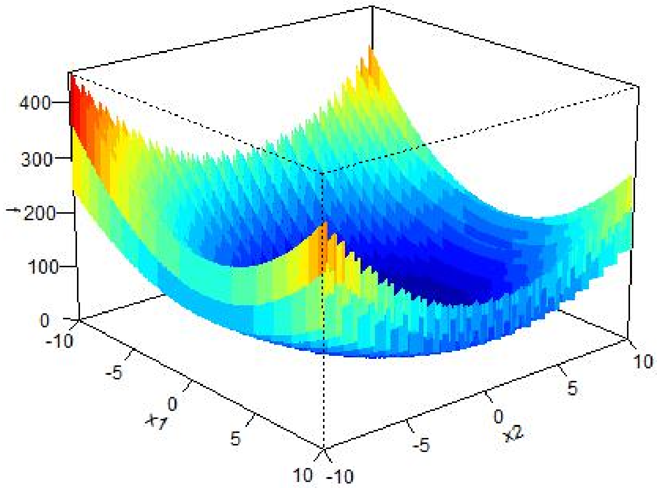
Postać matematyczna funkcji:

$$f(x) = -cos(x_1)cos(x_2)exp(-((x_1 - \pi)^2 + (x_2 - \pi)^2)) \tag{22}$$

Kod źródłowy:

```
rastrigin <- function(x1,x2)
  20 + x1^2 + x2^2 - 10*(cos(2*pi*x1) + cos(2*pi*x2))
```

2.5.12. Levy N.13



Rysunek 2.5.18: Wykres 3D funkcji Levy N.13

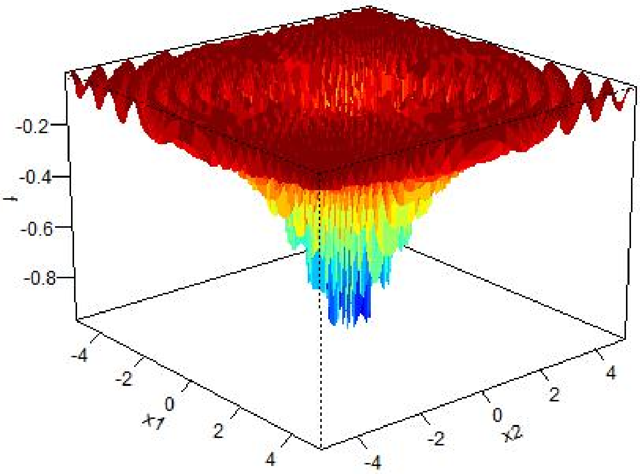
Postać matematyczna funkcji:

$$f(x) = sin^2(3\pi x_1) + (x_1 - 1)^2(1 + sin^2(3\pi x_2)) + (x_2 - 1)^2(1 + sin^2(2\pi x_2)) \qquad (23)$$

Kod źródłowy:

```
levin13 = function(x1,x2)
  sin(3*pi*x1)^2+(x1-1)^2 * (1+sin(3*pi*x2)^2)+(x2-1)^2 * (1+sin
      (2*pi*x2)^2)
```

2.5.13. Drop Wave



Rysunek 2.5.19: Wykres 3D funkcji Drop Wave

Postać matematyczna funkcji:

$$f(x) = -\frac{1 + cos(12\sqrt{x_1^2 + x_2^2})}{0.5(x_1^2 + x_2^2) + 2} \tag{24}$$

Kod źródłowy:

```
drop_wave = function(x1, x2)
  (-(1+cos(12*sqrt(x1^2+x2^2)))/(0.5*(x1^2+x2^2)+2))
```

3. Wyniki eksperymentu będącego testem programu

3.1. Optymalizacja rojem cząstek

Eksperyment przeprowadzono z wykorzystaniem pakietu *psoptim*. Wartości parametrów wywołania metody optymalizacyjnej były następujące:

- nazwa funkcji
- liczba cząstek w roju: [20,40,70,100,200]
- maksymalna liczba iteracji: 100
- liczba powtórzeń o identycznym wyniku zatrzymująca pracę programu: 20
- współczynnik bezwładności: 0,95
- współczynnik własnego „zaufania”: 0,2
- współczynnik „zaufania” do roju: 0,2
- wektor określający dolne ograniczenia wartości zmiennych
- wektor określający górne ograniczenia wartości zmiennych
- wektor ograniczeń prędkości w każdym kierunku: (4,4)
- liczba określająca tzw. ziarno dla generatora liczb pseudolosowych argument stosowany w celu uzyskania powtarzalności otrzymywanych wyników: rand(0:1000)

Na podstawie 50 prób dla każdej funkcji i liczebności zbioru wygenerowano wyniki średnie, które zamieszczono w tabelach 3.1.2 - 3.1.6.

Tabela 3.1.2: Średnie wyniki optymalizacji PSO

Ackley								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	-0,021	0,025	0,454	0,27	0,25	63,16	0,0142
2	40	-0,011	-0,007	0,572	0,72	0,64	52,16	0,0236
3	70	-0,002	-0,011	0,260	0,16	0,31	63,32	0,054
4	100	-0,001	-0,014	0,236	0,10	0,22	52,9	0,0694
5	200	0,001	-0,004	0,108	0,03	0,12	60	0,1716
	Wartości szukane	0	0	0				
Beale								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	3,160	0,524	0,022	0,00	0,03	45,02	0,0132
2	40	3,016	0,503	0,002	0,00	0,00	51,46	0,0202
3	70	3,006	0,500	0,002	0,00	0,00	56,8	0,048
4	100	3,004	0,503	0,001	0,00	0,00	61,68	0,0968
5	200	3,001	0,500	0,000	0,00	0,00	56,16	0,1564
	Wartości szukane	3	0,5	0				
Goldstein								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	0,007	-0,995	3,233	0,22	0,41	59,1	0,018
2	40	0,005	-0,998	3,164	0,25	0,48	54,5	0,0316
3	70	0,000	-0,999	3,095	0,07	0,26	57,78	0,059
4	100	0,001	-1,001	3,034	0,01	0,09	55,78	0,0722
5	200	0,001	-1,000	3,005	0,00	0,01	70,72	0,2138
	Wartości szukane	0	-1	3				

Tabela 3.1.3: cd. Średnie wyniki optymalizacji PSO

Bartels Conn

Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	-7,161	-0,073	849,971	2253397,61	1250,57	98,06	0,0254
2	40	-0,596	0,547	24,818	8567,17	90,35	84,66	0,038
3	70	-0,582	0,723	8,476	868,32	28,79	87,84	0,0936
4	100	-0,060	0,147	1,318	0,64	0,74	78,48	0,1362
5	200	-0,172	0,303	2,602	106,55	10,30	82	0,2252
	Wartości szukane	0	0	1				

Leon

Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	0,951	0,915	0,018	0,00	0,03	52,34	0,0162
2	40	1,014	1,029	0,002	0,00	0,00	50,78	0,0224
3	70	0,999	1,000	0,003	0,00	0,01	54,48	0,0402
4	100	1,000	1,002	0,002	0,00	0,00	56,86	0,072
5	200	0,999	0,999	0,000	0,00	0,00	58,88	0,1712
	Wartości szukane	1	1	0				

Eggholder

Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	353,201	332,005	-896,709	11860,21	90,24	80,96	0,0208
2	40	442,033	406,984	-935,571	2506,96	44,70	75,12	0,0344
3	70	478,452	413,605	-948,901	665,04	23,97	62	0,063
4	100	502,098	411,507	-956,303	61,08	7,41	84,96	0,1382
5	200	504,944	411,175	-958,521	3,07	1,70	67,02	0,1872
	Wartości szukane	512	404	-959				

Tabela 3.1.4: cd. Średnie wyniki optymalizacji PSO

Venter								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	-0,475	-1,651	-387,826	196,69	7,03	50,78	0,0132
2	40	0,105	-0,058	-396,711	47,48	6,12	61,56	0,0286
3	70	-0,061	0,042	-398,554	14,68	3,58	62,18	0,0682
4	100	0,005	-0,003	-399,666	0,49	0,62	63,98	0,0986
5	200	-0,006	-0,002	-399,813	0,21	0,42	63,56	0,1984
	Wartości szukane	0	0	-400				
Matyas								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	-0,066	-0,063	0,001	0,00	0,00	57,98	0,014
2	40	0,003	0,003	0,000	0,00	0,00	51,68	0,0224
3	70	0,016	0,011	0,000	0,00	0,00	43,52	0,0322
4	100	-0,014	-0,008	0,000	0,00	0,00	50,34	0,0584
5	200	0,001	0,001	0,000	0,00	0,00	53,68	0,1466
	Wartości szukane	0	0	0				
Zirilli								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	-1,045	0,022	-0,352	0,00	0,00	69,96	0,0182
2	40	-1,049	0,002	-0,351	0,00	0,00	54,18	0,023
3	70	-1,047	-0,002	-0,351	0,00	0,00	52,58	0,0518
4	100	-1,046	0,004	-0,352	0,00	0,00	58,3	0,0818
5	200	-1,046	-0,003	-0,352	0,00	0,00	53,38	0,1602
	Wartości szukane	-1,04	0	-0,35				

Tabela 3.1.5: cd. Średnie wyniki optymalizacji PSO

Drop Wave

Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	0,015	-0,059	-0,949	0,00	0,02	40,14	0,011
2	40	0,006	-0,035	-0,970	0,00	0,03	55,46	0,0232
3	70	-0,032	0,009	-0,987	0,00	0,02	65,04	0,0582
4	100	0,003	-0,013	-0,991	0,00	0,02	63,6	0,0844
5	200	0,003	0,010	-0,996	0,00	0,01	68,44	0,1842
	Wartości szukane	0	0	-1				

Levy N.13

Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	1,011	0,980	0,059	0,01	0,07	53,06	0,0144
2	40	1,011	1,001	0,052	0,01	0,06	50,38	0,0218
3	70	1,001	1,007	0,013	0,00	0,01	46,22	0,0338
4	100	1,017	1,031	0,015	0,00	0,03	49,76	0,0598
5	200	1,000	0,998	0,001	0,00	0,00	62,46	0,1802
	Wartości szukane	1	1	0				

Rastrigin

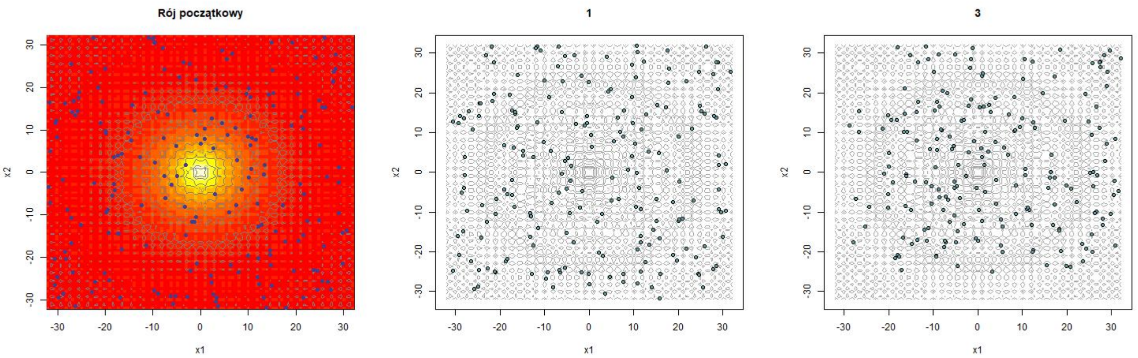
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	-0,018	0,099	0,481	0,62	0,63	63,76	0,0164
2	40	0,023	-0,061	0,427	0,61	0,66	55,44	0,0262
3	70	0,017	-0,021	0,104	0,06	0,23	61,5	0,0486
4	100	0,061	-0,041	0,119	0,10	0,30	60,96	0,0794
5	200	-0,001	-0,021	0,040	0,03	0,16	62,62	0,1698
	Wartości szukane	0	0	0				

Tabela 3.1.6: cd. Średnie wyniki optymalizacji PSO

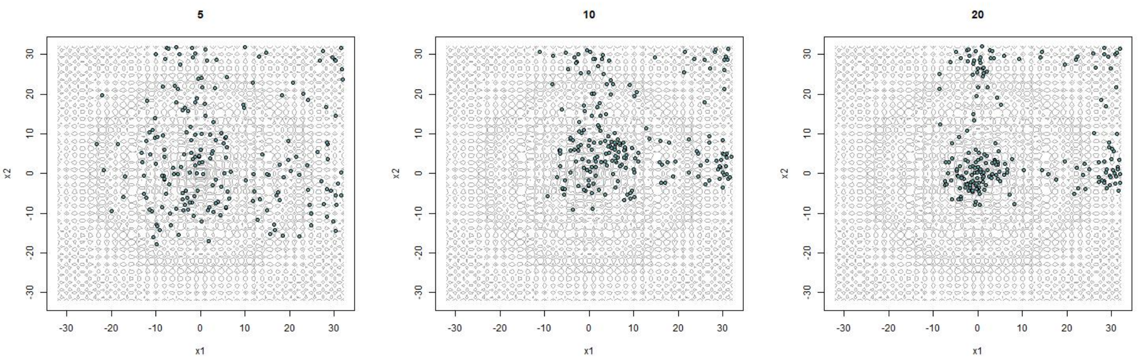
Easom								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	11,196	26,280	-0,620	0,37	0,48	53,2	0,016
2	40	3,135	3,135	-0,991	0,00	0,01	70,28	0,0326
3	70	3,141	3,148	-0,982	0,00	0,06	63,22	0,0648
4	100	3,138	3,134	-0,988	0,00	0,03	66,1	0,1032
5	200	3,139	3,138	-0,995	0,00	0,01	62,64	0,1936
	Wartości szukane	3,14	3,14	-1				

Na rysunkach znajdujących się w następnych podsekcjach przedstawiono wizualizację zainicjowania roju PSO o liczebności 200 i poszukiwania rozwiązań przez ten rój dla funkcji. Na rysunkach 3.1.20 - 3.1.30 przedstawiono początkowe pozycje osobników oraz ich pozycje w dalszych iteracjach.

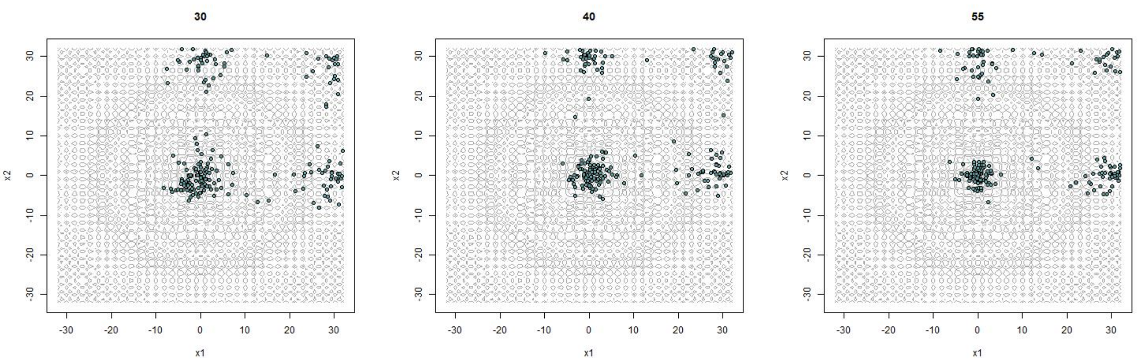
3.1.1. Ackley



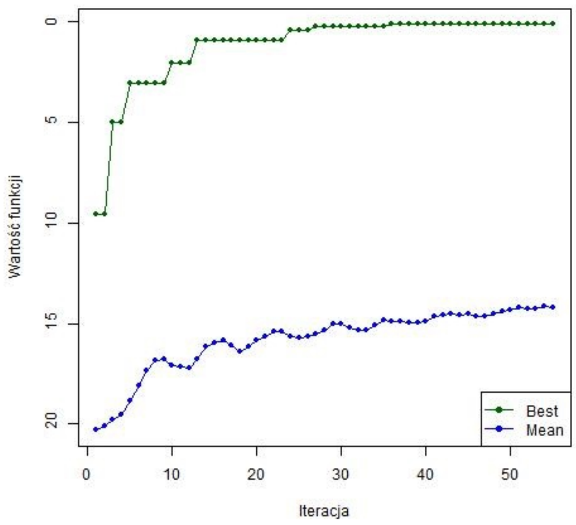
Rysunek 3.1.20: Iteracja 0, 1 i 3



Rysunek 3.1.21: Iteracja 5, 10 i 20

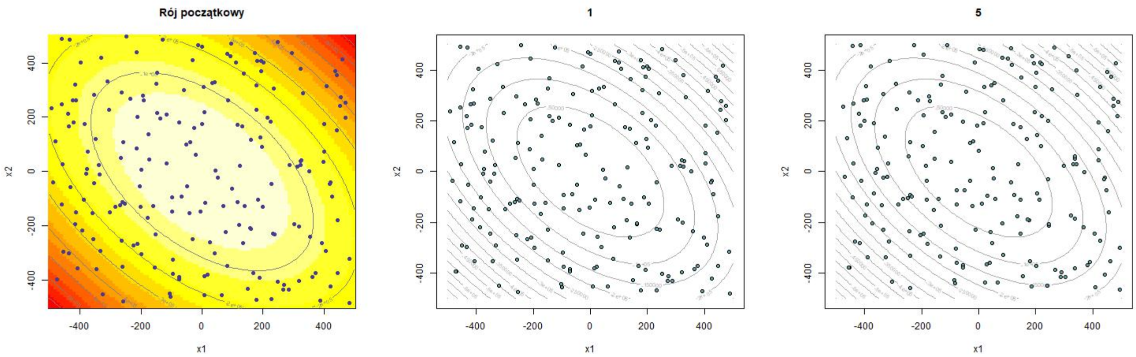


Rysunek 3.1.22: Iteracja 30, 40 i 55

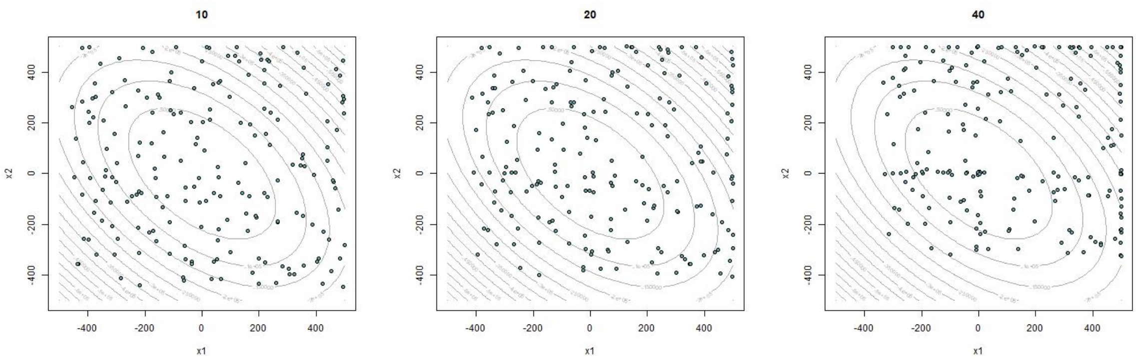


Rysunek 3.1.23: Wartości funkcji z biegiem iteracji

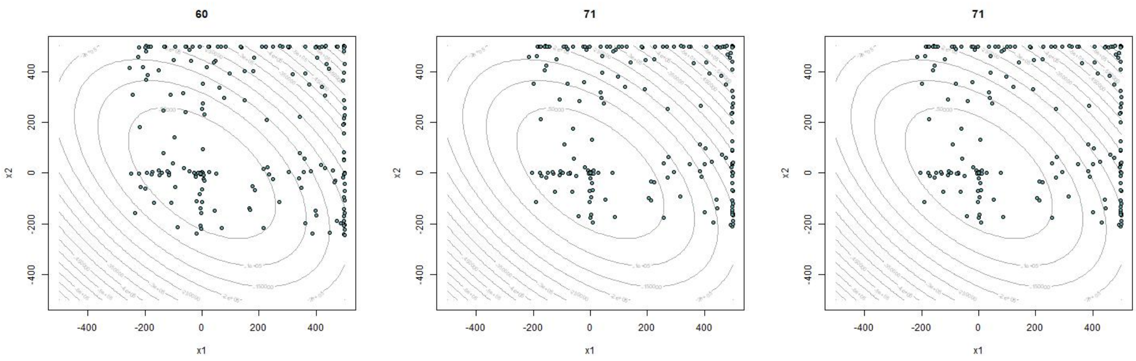
3.1.2. Bartels



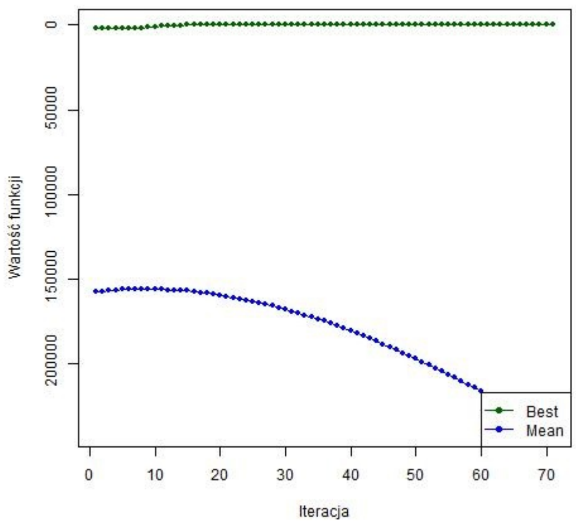
Rysunek 3.1.24: Iteracja 0, 1 i 5



Rysunek 3.1.25: Iteracja 10, 20 i 40

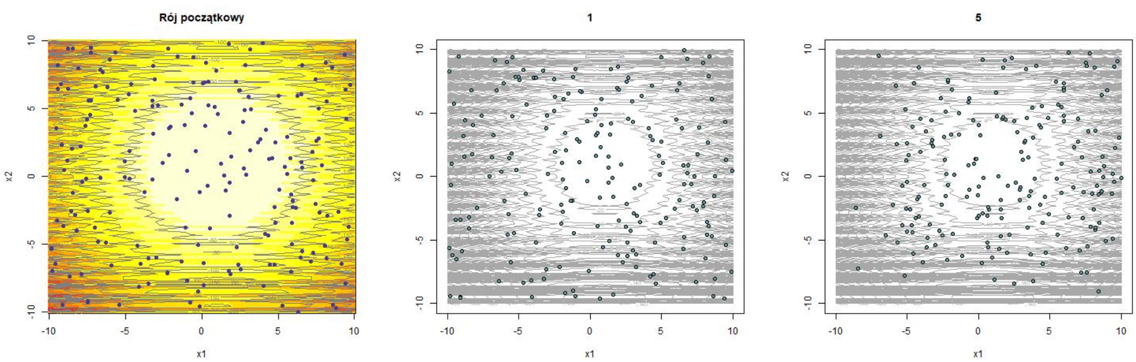


Rysunek 3.1.26: Iteracja 60, 71 i 71

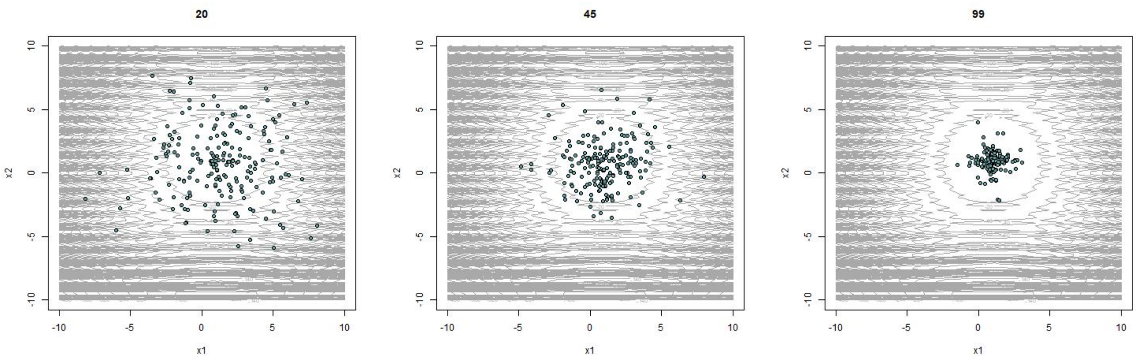


Rysunek 3.1.27: Wartości funkcji z biegiem iteracji

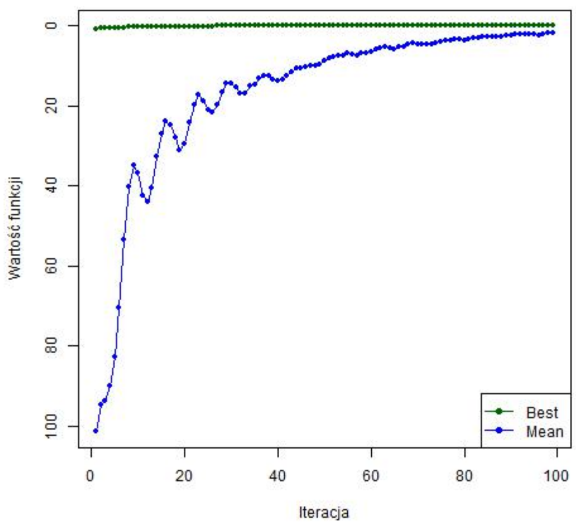
3.1.3. Levy N.13



Rysunek 3.1.28: Iteracja 0, 1 i 5



Rysunek 3.1.29: Iteracja 20, 45 i 99



Rysunek 3.1.30: Wartości funkcji z biegiem iteracji

3.2. Optymalizacja algorytmem nietoperza

Optymalizacja została przeprowadzona algorytmem nietoperza z pakietu *microbats*. Argumenty funkcji programu wprowadzono następujące:

- nazwa funkcji
- liczba nietoperzy: [20,40,70,100,200]
- maksymalna liczba iteracji: 100
- liczba powtórzeń o identycznym wyniku zatrzymująca pracę programu: 20
- „głośność” nietoperzy: 0,5

- szybkość impulsów: 0,5
- minimalna częstotliwość: 0
- maksymalna częstotliwość: 2
- wektor określający dolne ograniczenia wartości zmiennych
- wektor określający górne ograniczenia wartości zmiennych
- liczba określająca tzw. ziarno dla generatora liczb pseudolosowych argument stosowany w celu uzyskania powtarzalności otrzymywanych wyników: rand(0:1000)

Wykonano zestawy 50 prób badania dla każdej funkcji oraz liczebności zbioru i na ich podstawie wygenerowano wyniki średnie, które zamieszczono w tabelach 3.2.7 - 3.2.11.

Tabela 3.2.7: Uśrednione wyniki optymalizacji algorytmem nietoperza

Ackley								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	0,266	0,011	5,321	35,02	2,61	86,6	0,0232
2	40	0,039	-0,176	3,190	15,46	2,32	72	0,0352
3	70	-0,101	0,156	2,181	8,74	2,02	61,1	0,0716
4	100	0,041	0,155	1,652	6,23	1,89	47,5	0,0876
5	200	-0,019	0,001	0,872	2,47	1,32	43,64	0,1672
	Wartości szukane	0	0	0				

Beale								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	1,583	0,766	0,301	0,27	0,43	97,72	0,024
2	40	1,560	0,677	0,183	0,14	0,33	88,86	0,0428
3	70	1,595	0,622	0,134	0,10	0,29	70,4	0,0826
4	100	2,047	0,581	0,088	0,07	0,24	61,18	0,0954
5	200	2,665	0,526	0,029	0,02	0,14	51,92	0,1758
	Wartości szukane	3	0,5	0				

Goldstein								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	-0,008	-0,923	8,200	249,86	15,08	73,86	0,0154
2	40	-0,060	-0,940	5,700	72,90	8,18	47,3	0,0196
3	70	0,000	-1,000	3,000	0,00	0,00	40,54	0,0416
4	100	0,000	-1,000	3,000	0,00	0,00	39,6	0,0542
5	200	0,000	-1,000	3,000	0,00	0,00	34,78	0,1242
	Wartości szukane	0	-1	3				

Tabela 3.2.8: cd. Uśrednione wyniki optymalizacji algorytmem nietoperza

Bartels Conn

Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	-6,679	-2,472	1474,761	6024176,81	1982,63	100	0,023
2	40	-0,466	0,656	625,355	1143986,53	877,25	100	0,0454
3	70	-1,842	-0,897	246,037	174922,97	342,38	100	0,1252
4	100	1,385	-0,423	56,269	9039,46	78,15	100	0,183
5	200	-0,639	0,931	8,054	260,26	14,66	99,86	0,2788
	Wartości szukane	0	0	1				

Leon

Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	0,902	0,908	0,104	0,29	0,54	98,76	0,0232
2	40	1,005	1,011	0,001	0,00	0,00	97,28	0,0544
3	70	1,000	1,001	0,000	0,00	0,00	90,02	0,094
4	100	1,000	1,000	0,000	0,00	0,00	74,76	0,1276
5	200	1,000	1,000	0,000	0,00	0,00	57	0,1884
	Wartości szukane	1	1	0				

Eggholder

Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	55,148	144,065	-755,063	58159,56	130,03	100	0,024
2	40	1,490	79,190	-793,781	36654,19	97,71	100	0,0444
3	70	77,336	226,536	-843,395	22014,37	93,95	100	0,1382
4	100	116,847	225,280	-854,870	18877,59	90,55	100	0,1712
5	200	228,957	363,206	-918,208	5226,82	60,30	97,6	0,286
	Wartości szukane	512	404	-959				

Tabela 3.2.9: cd. Uśrednione wyniki optymalizacji algorytmem nietoperza

Venter								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	1,426	0,721	-299,963	20691,25	104,41	95,86	0,0264
2	40	0,024	-0,479	-365,818	3317,82	46,83	84,86	0,054
3	70	-0,807	0,185	-379,007	1547,07	33,60	71,28	0,0838
4	100	0,123	0,000	-385,770	298,83	9,91	52,92	0,0806
5	200	0,000	0,431	-386,659	276,85	10,04	40,56	0,1394
	Wartości szukane	0	0	-400				

Matyas								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	-0,081	-0,058	0,096	0,10	0,30	95,14	0,0176
2	40	0,016	0,016	0,003	0,00	0,01	86,02	0,0388
3	70	-0,002	-0,002	0,000	0,00	0,00	69,72	0,0748
4	100	0,000	0,000	0,000	0,00	0,00	47,9	0,0722
5	200	0,000	0,000	0,000	0,00	0,00	38,08	0,1288
	Wartości szukane	0	0	0				

Zirilli								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	-0,072	-0,048	-0,128	0,16	0,33	92,28	0,0252
2	40	-0,250	-0,001	-0,272	0,02	0,10	75,46	0,0386
3	70	-0,808	0,000	-0,328	0,00	0,07	54,04	0,0502
4	100	-0,728	0,000	-0,320	0,01	0,07	45,98	0,0728
5	200	-1,007	0,000	-0,348	0,00	0,03	37,54	0,1312
	Wartości szukane	-1,04	0	-0,35				

Tabela 3.2.10: cd. Uśrednione wyniki optymalizacji algorytmem nietoperza

Drop Wave								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	-0,119	0,143	-0,764	0,07	0,14	41,04	0,0108
2	40	-0,010	0,043	-0,885	0,02	0,10	40,16	0,0192
3	70	0,012	0,035	-0,928	0,01	0,06	39	0,033
4	100	0,039	-0,020	-0,930	0,01	0,04	34,58	0,052
5	200	-0,064	-0,046	-0,940	0,00	0,02	36,72	0,1284
	Wartości szukane	0	0	-1				

Levy N.13								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	0,934	1,167	0,989	2,57	1,27	74,48	0,0188
2	40	1,059	0,997	0,339	0,32	0,46	63,84	0,029
3	70	1,013	0,962	0,135	0,04	0,16	51,8	0,056
4	100	0,974	0,988	0,122	0,06	0,21	48,44	0,078
5	200	0,967	1,005	0,050	0,01	0,08	43,04	0,1576
	Wartości szukane	1	1	0				

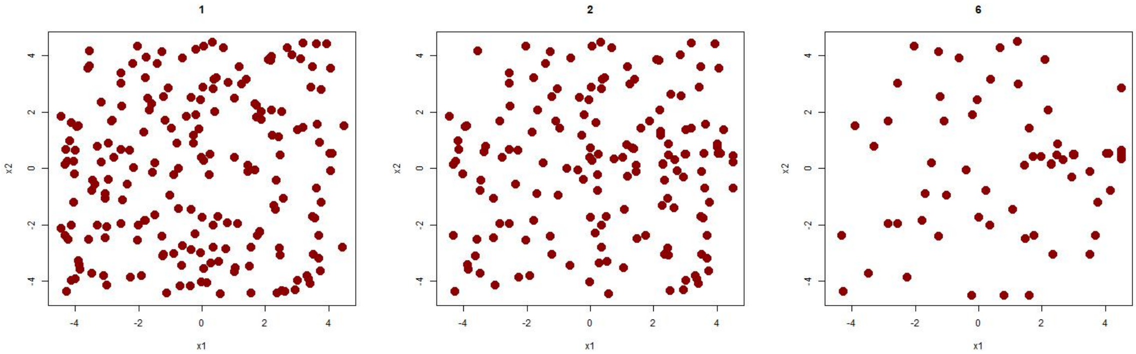
Rastrigin								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	-0,020	0,199	3,681	29,36	4,02	57,12	0,0136
2	40	0,080	-0,040	3,025	14,02	2,23	42,84	0,0204
3	70	-0,159	0,040	1,672	6,14	1,85	37,96	0,0346
4	100	0,020	-0,119	1,333	4,97	1,80	38,26	0,0538
5	200	0,040	-0,080	0,995	1,78	0,90	35,48	0,1156
	Wartości szukane	0	0	0				

Tabela 3.2.11: cd. Uśrednione wyniki optymalizacji algorytmem nietoperza

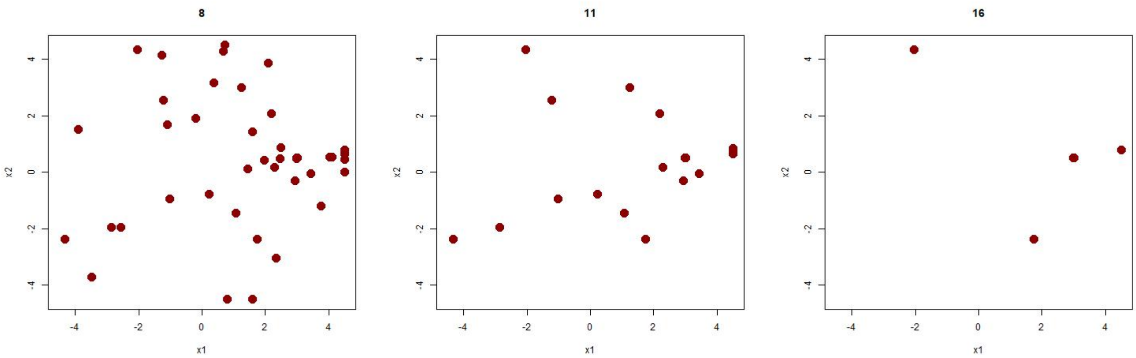
Easom								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	-2,436	-1,023	-0,024	0,97	0,12	78,68	0,0198
2	40	-3,063	1,028	-0,104	0,87	0,26	92,32	0,0538
3	70	3,281	3,087	-0,256	0,73	0,42	93,92	0,112
4	100	2,954	3,453	-0,478	0,52	0,50	85,78	0,1546
5	200	3,031	3,178	-0,860	0,14	0,35	60,38	0,1954
	Wartości szukane	3,14	3,14	-1				

Na rysunkach 3.2.31 - 3.2.40 przedstawiono podobnie jak w przypadku PSO pozycje (populacji 200) osobników w wybranych iteracjach poszukiwania minimów wybranych funkcji przez roje.

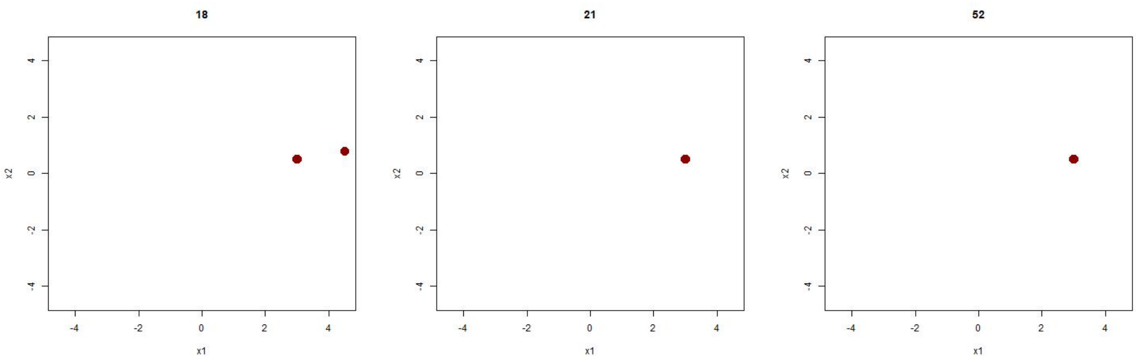
3.2.1. Beale



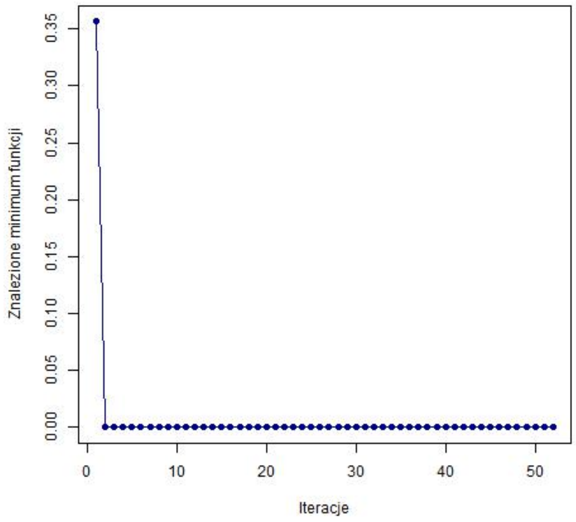
Rysunek 3.2.31: Iteracja 0, 1 i 5



Rysunek 3.2.32: Iteracja 7, 10 i 15

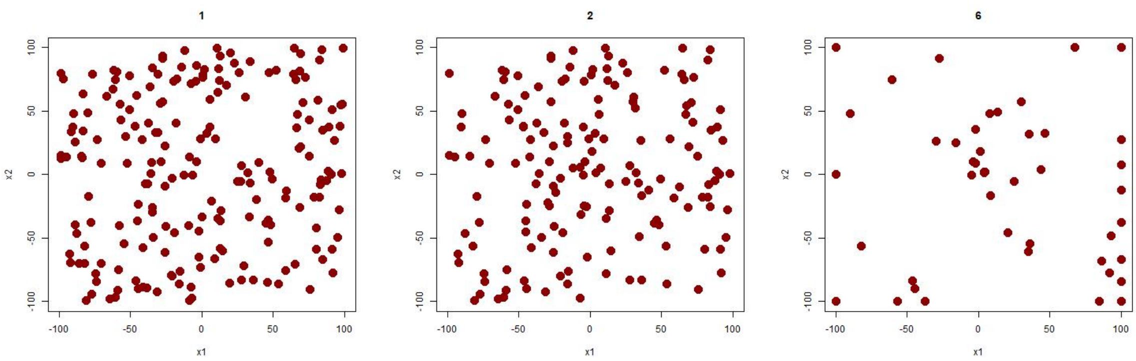


Rysunek 3.2.33: Iteracja 17, 20 i 51

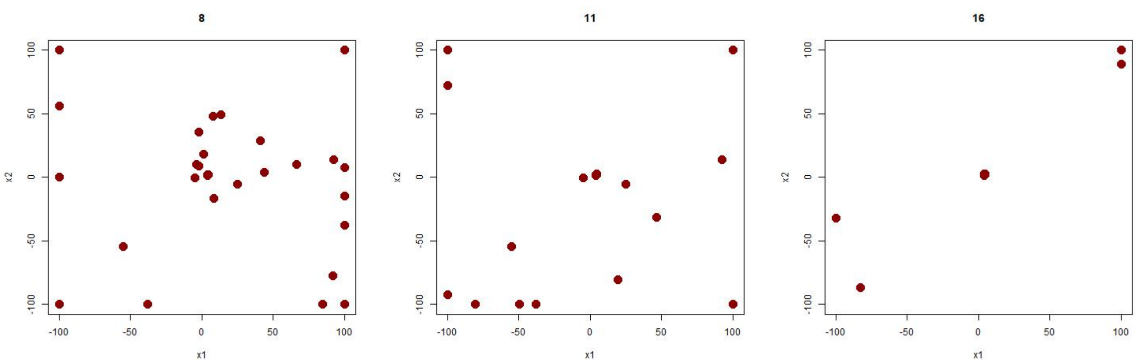


Rysunek 3.2.34: Wartości funkcji z biegiem iteracji

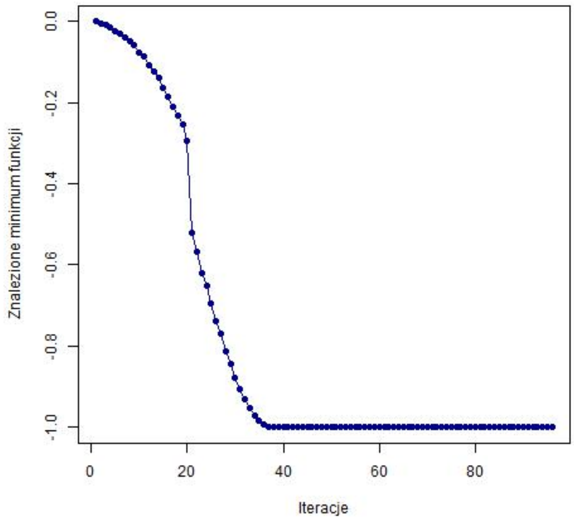
3.2.2. Easom



Rysunek 3.2.35: Iteracja 0, 1 i 5

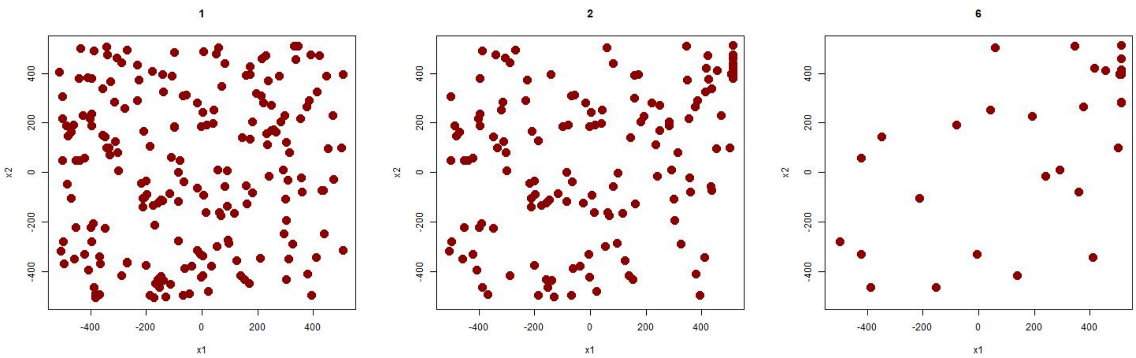


Rysunek 3.2.36: Iteracja 7, 10 i 15

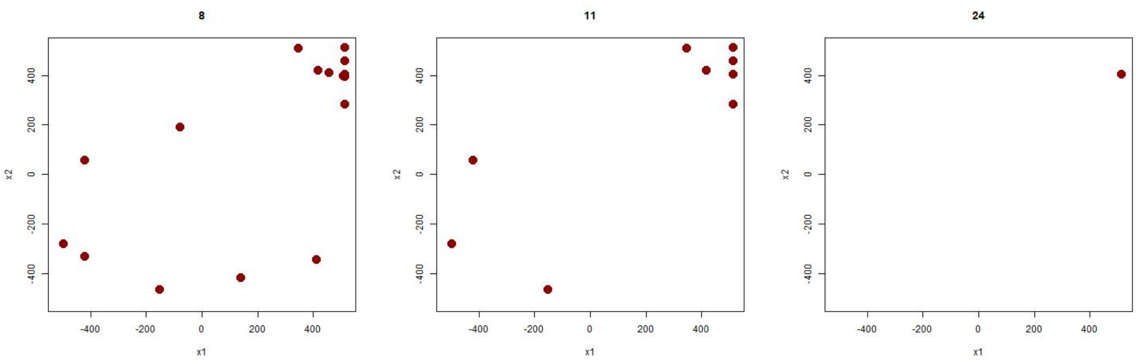


Rysunek 3.2.37: Wartości funkcji z biegiem iteracji

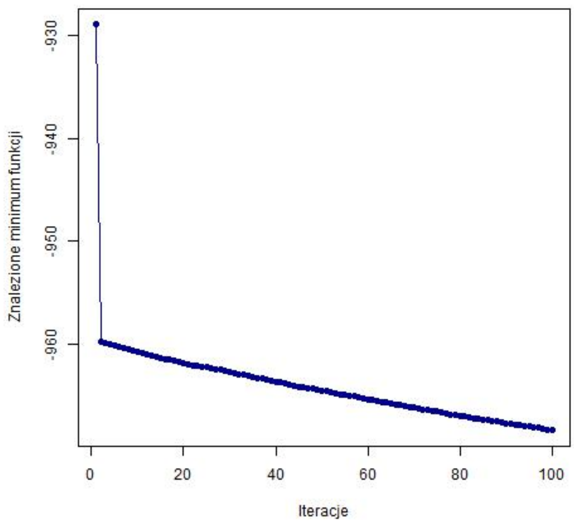
3.2.3. Eggholder



Rysunek 3.2.38: Iteracja 0, 1 i 5



Rysunek 3.2.39: Iteracja 7, 10 i 23



Rysunek 3.2.40: Wartości funkcji z biegiem iteracji

3.3. Optymalizacja algorytmem genetycznym

Optymalizacja została przeprowadzona algorytmem nietoperza z pakietu *GA*. Argumenty funkcji, które zostały wprowadzone:

- nazwa funkcji
- wielkość populacji: [20,40,70,100,200]
- liczba powtórzeń o identycznym wyniku zatrzymująca pracę programu: 20
- maksymalna liczba iteracji: 100
- prawdopodobieństwo krzyżowania między parami chromosomów: 0,8
- prawdopodobieństwo mutacji między parami chromosomów: 0,1
- liczba określająca procent najlepszych osobników populacji przechodzących do następnego pokolenia: 5
- wektor określający dolne ograniczenia wartości zmiennych
- wektor określający górne ograniczenia wartości zmiennych

Średnie wyniki, tak jak w przypadku wcześniejszych algorytmów zamieszczono w tabelach, 3.3.12 - 3.3.16.

Tabela 3.3.12: Uśrednione wyniki optymalizacji algorytmem genetycznym

Ackley

Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	0,004	-0,002	0,170	0,16	0,36	61,56	0,074
2	40	0,001	-0,001	0,024	0,01	0,07	67,3	0,1138
3	70	0,000	0,000	0,001	0,00	0,00	82,86	0,1928
4	100	0,000	0,000	0,001	0,00	0,00	77,08	0,23
5	200	0,000	0,000	0,000	0,00	0,00	83,06	0,4308
	Wartości szukane	0	0	0				

Beale

Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	2,830	0,442	0,026	0,00	0,04	46,1	0,0548
2	40	2,953	0,483	0,011	0,00	0,02	48,48	0,0898
3	70	2,965	0,491	0,003	0,00	0,00	53,14	0,1256
4	100	2,982	0,493	0,003	0,00	0,00	49,04	0,1466
5	200	2,993	0,497	0,001	0,00	0,00	53,76	0,2786
	Wartości szukane	3	0,5	0				

Goldstein

Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	0,005	-0,982	3,652	3,03	1,63	43,82	0,0518
2	40	0,004	-0,995	3,094	0,03	0,14	47,98	0,0858
3	70	0,001	-0,998	3,028	0,00	0,06	59,76	0,1406
4	100	0,001	-1,000	3,018	0,00	0,03	53,94	0,1642
5	200	0,001	-1,000	3,005	0,00	0,01	64,66	0,3376
	Wartości szukane	0	-1	3				

Tabela 3.3.13: cd. Uśrednione wyniki optymalizacji algorytmem genetycznym

Bartels Conn

Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	0,107	-0,285	10,154	509,59	20,84	55,46	0,0688
2	40	0,128	0,004	2,792	18,47	3,95	58,88	0,0988
3	70	-0,005	-0,030	1,349	1,65	1,25	71,68	0,1622
4	100	0,079	-0,046	1,428	4,51	2,10	69,42	0,2084
5	200	-0,001	0,043	1,040	0,01	0,11	79,12	0,4056
	Wartości szukane	0	0	1				

Leon

Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	0,614	0,394	0,175	0,04	0,12	41,18	0,0456
2	40	0,769	0,602	0,076	0,01	0,06	38,06	0,0724
3	70	0,843	0,730	0,051	0,01	0,05	39,24	0,099
4	100	0,854	0,740	0,038	0,00	0,04	32,64	0,1028
5	200	0,921	0,854	0,017	0,00	0,02	29,7	0,1594
	Wartości szukane	1	1	0				

Eggholder

Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	-14,709	211,910	-831,726	24317	91,02	53,3	0,0604
2	40	38,193	321,456	-874,517	11337	65,46	53,86	0,0904
3	70	83,472	415,712	-897,569	5282	39,23	62,62	0,1464
4	100	152,118	411,942	-903,743	4158	33,58	50,2	0,1538
5	200	275,515	428,405	-917,263	2660	30,60	51,96	0,2832
	Wartości szukane	512	404	-959				

Tabela 3.3.14: cd. Uśrednione wyniki optymalizacji algorytmem genetycznym

Venter								
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	0,072	0,105	-396,501	47,03	5,96	63,58	0,0732
2	40	-0,005	0,128	-398,444	18,97	4,11	65,46	0,1126
3	70	-0,001	0,000	-399,940	0,03	0,18	73,18	0,1706
4	100	0,001	-0,001	-399,967	0,01	0,09	73,22	0,2136
5	200	-0,002	0,001	-399,984	0,00	0,06	73,82	0,375
	Wartości szukane	0	0	-400				

Matyas								
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	-0,024	-0,030	0,003	0,00	0,01	51,54	0,0652
2	40	0,009	0,007	0,000	0,00	0,00	58,38	0,1008
3	70	0,000	0,000	0,000	0,00	0,00	65,72	0,1488
4	100	0,000	0,001	0,000	0,00	0,00	62,32	0,1836
5	200	0,000	0,000	0,000	0,00	0,00	69,28	0,3484
	Wartości szukane	0	0	0				

Zirilli								
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	-1,008	0,003	-0,341	0,00	0,02	35,8	0,0394
2	40	-1,030	0,006	-0,348	0,00	0,01	28,38	0,0532
3	70	-1,048	0,001	-0,351	0,00	0,00	18,4	0,0438
4	100	-1,046	-0,007	-0,351	0,00	0,00	19,04	0,0592
5	200	-1,044	-0,002	-0,351	0,00	0,00	15,54	0,0848
	Wartości szukane	-1,04	0	-0,35				

Tabela 3.3.15: cd. Uśrednione wyniki optymalizacji algorytmem genetycznym

				Drop Wave				
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	0,030	0,033	-0,944	0,00	0,02	53,66	0,063
2	40	-0,062	-0,004	-0,971	0,00	0,03	60,7	0,1032
3	70	0,006	-0,035	-0,982	0,00	0,03	67,46	0,1676
4	100	0,011	0,006	-0,988	0,00	0,02	65,06	0,2182
5	200	0,000	0,000	-0,999	0,00	0,00	69,54	0,3882
	Wartości szukane	0	0	-1				

				Levy N.13				
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	1,005	1,004	0,034	0,00	0,06	52,24	0,0666
2	40	1,000	0,996	0,009	0,00	0,03	56,08	0,096
3	70	1,000	1,005	0,001	0,00	0,00	73,98	0,1746
4	100	1,000	1,002	0,001	0,00	0,00	73,38	0,2122
5	200	1,000	0,999	0,000	0,00	0,00	76,68	0,3872
	Wartości szukane	1	1	0				

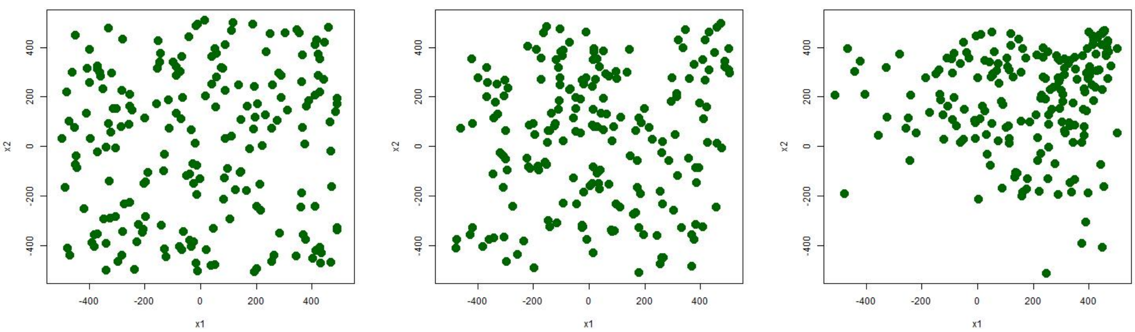
				Rastrigin				
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	0,000	-0,060	0,194	0,22	0,43	63,68	0,0812
2	40	0,000	0,000	0,001	0,00	0,00	76,84	0,1238
3	70	0,000	0,000	0,001	0,00	0,00	74,7	0,1766
4	100	0,000	0,000	0,000	0,00	0,00	76,54	0,2304
5	200	0,000	0,000	0,000	0,00	0,00	82,12	0,4158
	Wartości szukane	0	0	0				

Tabela 3.3.16: cd. Uśrednione wyniki optymalizacji algorytmem genetycznym

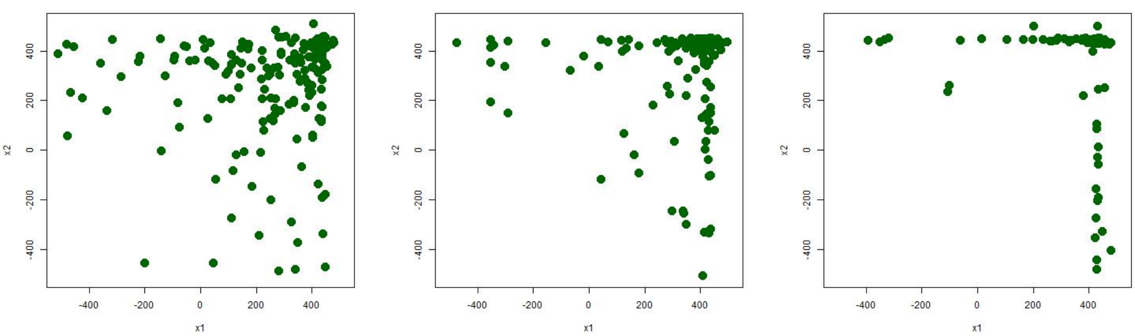
Easom								
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	3,309	2,811	-0,536	0,40	0,43	49,28	0,0568
2	40	2,961	2,945	-0,748	0,21	0,38	62,26	0,1022
3	70	3,138	3,093	-0,944	0,03	0,18	69,1	0,1556
4	100	3,122	3,127	-0,977	0,01	0,12	75,76	0,2184
5	200	3,145	3,136	-0,996	0,00	0,02	78,96	0,3994
	Wartości szukane	3,14	3,14	-1				

Proces optymalizacji GA dla wszystkich funkcji został przedstawiony graficznie na rysunkach 3.3.41 - 3.3.49.

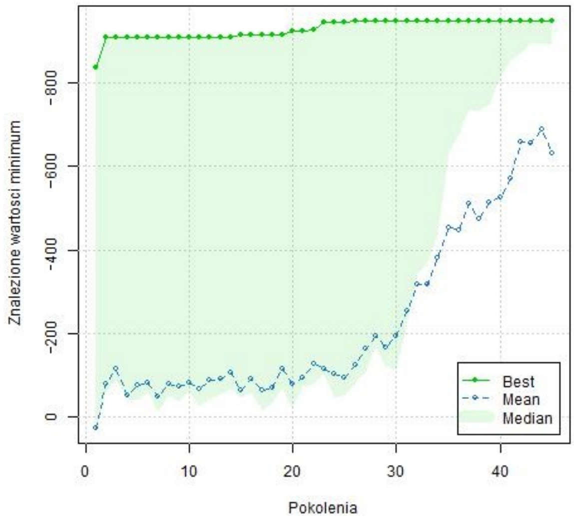
3.3.1. Eggholder



Rysunek 3.3.41: Iteracja 0, 3 i 20

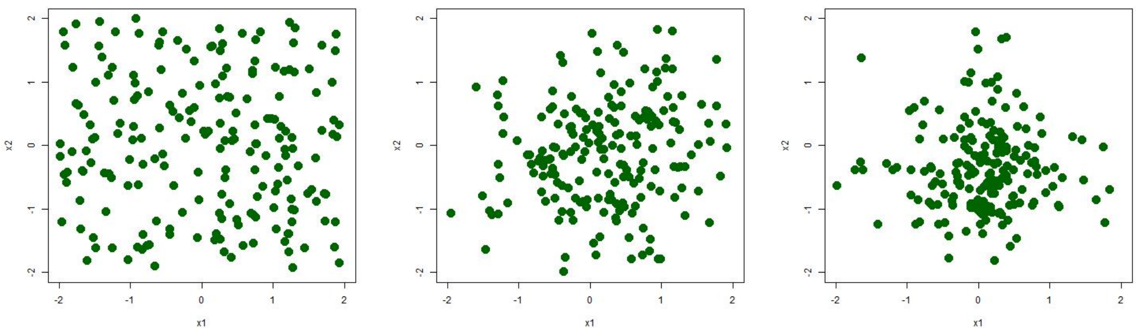


Rysunek 3.3.42: Iteracja 28, 34 i 45

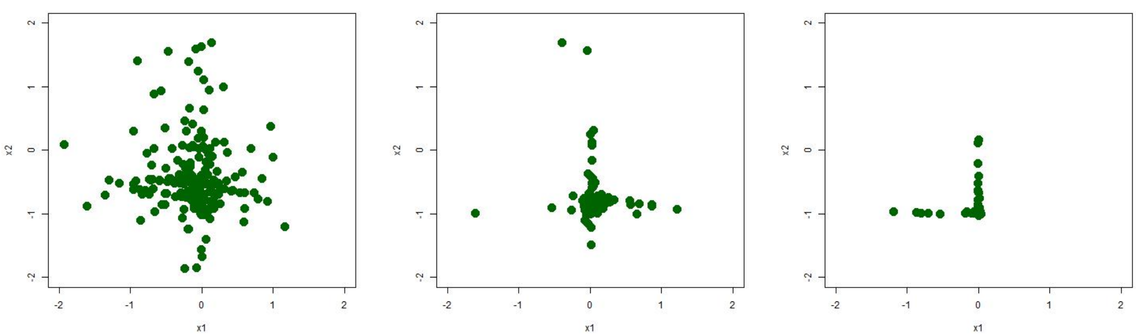


Rysunek 3.3.43: Wartości funkcji z biegiem iteracji

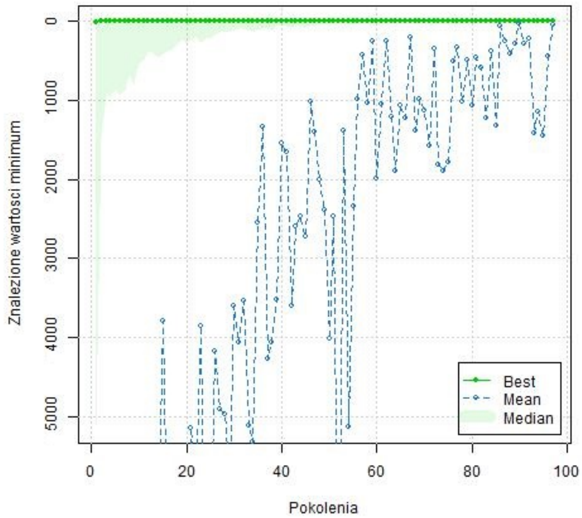
3.3.2. Goldstein



Rysunek 3.3.44: Iteracja 0, 3 i 10

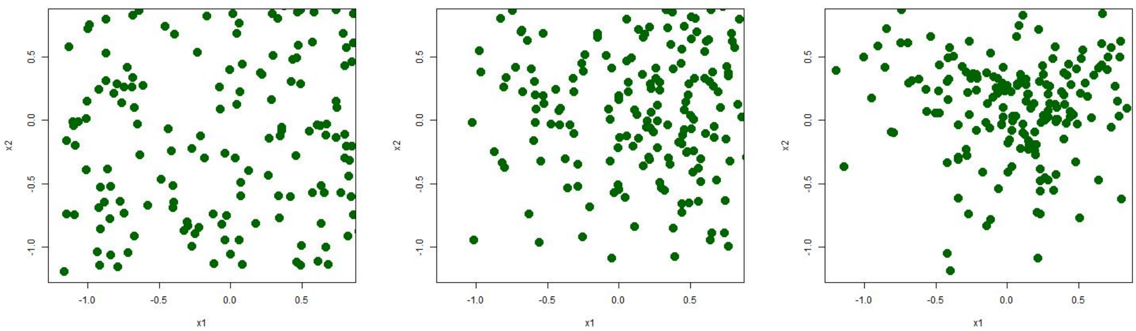


Rysunek 3.3.45: Iteracja 25, 60 i 97

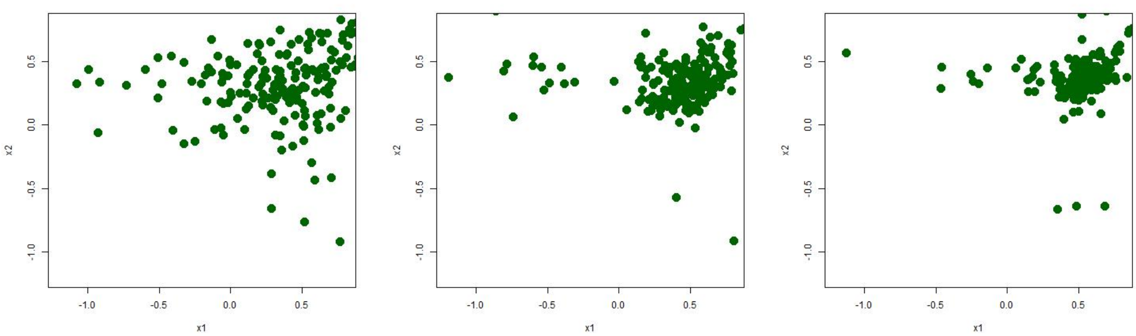


Rysunek 3.3.46: Wartości funkcji z biegiem iteracji

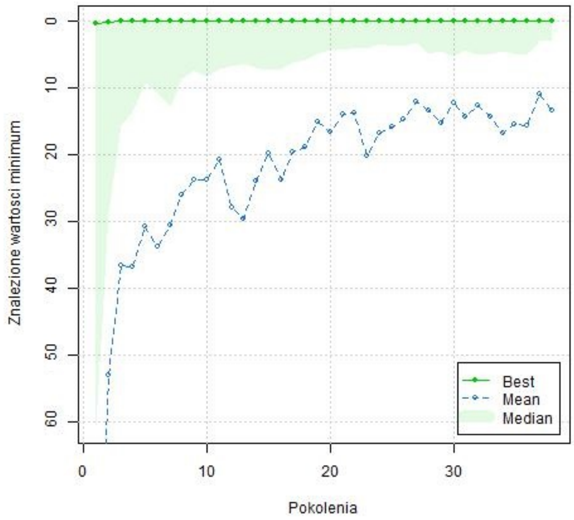
3.3.3. Leon



Rysunek 3.3.47: Iteracja 0, 3 i 10



Rysunek 3.3.48: Iteracja 30, 45 i 49



Rysunek 3.3.49: Wartości funkcji z biegiem iteracji

3.4. Optymalizacja algorytmem ewolucji różnicowej

Argumenty funkcji *DEOptim*, które zostały wprowadzone podczas eksperymentu z wykorzystaniem ewolucji różnicowej:

- nazwa funkcji
- strategia algorytmu: DE / local-to-best / 1 / bin
- wielkość populacji: [20,40,70,100,200]
- maksymalna liczba iteracji: 100
- prawdopodobieństwo krzyżowania: 0,5
- współczynnik wagi różnicowej: 0,8
- prędkość adaptacji po krzyżowaniu: 0
- wektor określający dolne ograniczenia wartości zmiennych
- wektor określający górne ograniczenia wartości zmiennych
- liczba określająca tzw. ziarno dla generatora liczb pseudolosowych argument stosowany w celu uzyskania powtarzalności otrzymywanych wyników: rand(0:1000)

Uśrednione wyniki optymalizacji zawarto w tabelach 3.4.17 - 3.4.21.

Tabela 3.4.17: Uśrednione wyniki optymalizacji algorytmem ewolucji różnicowej

Ackley								
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	0,000	0,000	0,000	0,00	0,00	100	0,01
2	40	0,000	0,000	0,000	0,00	0,00	100	0,02
3	70	0,000	0,000	0,000	0,00	0,00	100	0,03
4	100	0,000	0,000	0,000	0,00	0,00	100	0,04
5	200	0,000	0,000	0,000	0,00	0,00	100	0,07
	Wartości szukane	0	0	0				

Beale								
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	2,850	0,514	0,015	0,01	0,11	100	0,01
2	40	3,000	0,500	0,000	0,00	0,00	100	0,01
3	70	3,000	0,500	0,000	0,00	0,00	100	0,02
4	100	3,000	0,500	0,000	0,00	0,00	100	0,03
5	200	3,000	0,500	0,000	0,00	0,00	100	0,06
	Wartości szukane	3	0,5	0				

Goldstein								
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	0,000	-1,000	3,000	0,00	0,00	77	0,01
2	40	0,000	-1,000	3,000	0,00	0,00	75	0,01
3	70	0,000	-1,000	3,000	0,00	0,00	73	0,02
4	100	0,000	-1,000	3,000	0,00	0,00	74	0,02
5	200	0,000	-1,000	3,000	0,00	0,00	72	0,05
	Wartości szukane	0	-1	3				

Tabela 3.4.18: cd. Uśrednione wyniki optymalizacji algorytmem ewolucji różnicowej

Bartels Conn

Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	0,000	0,000	1,000	0,00	0,00	100	0,01
2	40	0,000	0,000	1,000	0,00	0,00	100	0,02
3	70	0,000	0,000	1,000	0,00	0,00	100	0,02
4	100	0,000	0,000	1,000	0,00	0,00	100	0,03
5	200	0,000	0,000	1,000	0,00	0,00	100	0,06
	Wartości szukane	0	0	1				

Leon

Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	1,000	1,000	0,000	0,00	0,00	100	0,01
2	40	1,000	1,000	0,000	0,00	0,00	100	0,01
3	70	1,000	1,000	0,000	0,00	0,00	100	0,02
4	100	1,000	1,000	0,000	0,00	0,00	100	0,03
5	200	1,000	1,000	0,000	0,00	0,00	100	0,06
	Wartości szukane	1	1	0				

Eggholder

Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	197,306	398,599	-927,745	2015	32,54	100	0,01
2	40	444,374	427,229	-947,299	337	14,27	100	0,02
3	70	496,297	418,435	-955,498	27	3,85	100	0,02
4	100	497,511	417,136	-956,717	8	1,81	96	0,03
5	200	494,656	420,648	-957,273	5	1,27	96	0,06
	Wartości szukane	512	404	-959				

Tabela 3.4.19: cd. Uśrednione wyniki optymalizacji algorytmem ewolucji różnicowej

Venter								
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	0,000	0,000	-400,000	0,00	0,00	99	0,01
2	40	0,000	0,000	-400,000	0,00	0,00	100	0,02
3	70	0,000	0,000	-400,000	0,00	0,00	99	0,02
4	100	0,000	0,000	-400,000	0,00	0,00	100	0,03
5	200	0,000	0,000	-400,000	0,00	0,00	100	0,07
	Wartości szukane	0	0	-400				
Matyas								
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania
1	20	0,000	0,000	0,000	0,00	0,00	100	0,01
2	40	0,000	0,000	0,000	0,00	0,00	100	0,01
3	70	0,000	0,000	0,000	0,00	0,00	100	0,02
4	100	0,000	0,000	0,000	0,00	0,00	100	0,03
5	200	0,000	0,000	0,000	0,00	0,00	100	0,05
	Wartości szukane	0	0	0				
Zirilli								
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	-1,047	0,000	-0,352	0,00	0,00	100	0,01
2	40	-1,047	0,000	-0,352	0,00	0,00	100	0,01
3	70	-1,047	0,000	-0,352	0,00	0,00	100	0,02
4	100	-1,047	0,000	-0,352	0,00	0,00	100	0,03
5	200	-1,047	0,000	-0,352	0,00	0,00	100	0,06
	Wartości szukane	-1,04	0	-0,35				

Tabela 3.4.20: cd. Uśrednione wyniki optymalizacji algorytmem ewolucji różnicowej

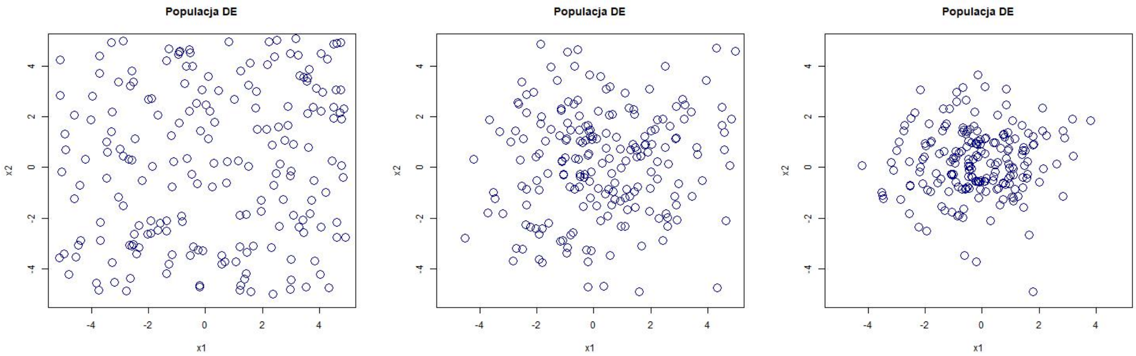
Drop Wave								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	0,002	0,000	-0,997	0,00	0,00	100	0,01
2	40	0,000	0,000	-1,000	0,00	0,00	100	0,02
3	70	0,000	0,000	-1,000	0,00	0,00	100	0,02
4	100	0,000	0,000	-1,000	0,00	0,00	100	0,03
5	200	0,000	0,000	-1,000	0,00	0,00	100	0,06
	Wartości szukane	0	0	-1				
Levy N.13								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	1,000	1,000	0,000	0,00	0,00	100	0,01
2	40	1,000	1,000	0,000	0,00	0,00	100	0,01
3	70	1,000	1,000	0,000	0,00	0,00	100	0,02
4	100	1,000	1,000	0,000	0,00	0,00	100	0,03
5	200	1,000	1,000	0,000	0,00	0,00	100	0,07
	Wartości szukane	1	1	0				
Rastrigin								
Nr_sredniej	Liczebnosc_roju	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	-0,020	0,001	0,034	0,02	0,14	100	0,01
2	40	0,000	0,000	0,000	0,00	0,00	100	0,02
3	70	0,000	0,000	0,000	0,00	0,00	100	0,02
4	100	0,000	0,000	0,000	0,00	0,00	100	0,03
5	200	0,000	0,000	0,000	0,00	0,00	100	0,06
	Wartości szukane	0	0	0				

Tabela 3.4.21: cd. Uśrednione wyniki optymalizacji algorytmem ewolucji różnicowej

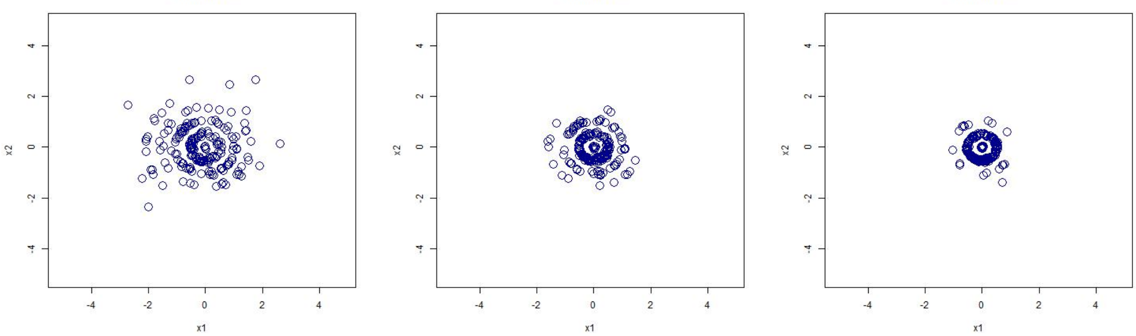
Easom								
Nr_sredniej	Liczebnosc_pop	Wartosc_x1	Wartosc_x2	Znalezione_minimum	MSE_minimum	Odchylenie_standardowe	Iteracje_do_wyniku	Czas_wykonania[s]
1	20	3,142	3,142	-1,000	0,00	0,00	99	0,01
2	40	3,142	3,142	-1,000	0,00	0,00	98	0,01
3	70	3,142	3,142	-1,000	0,00	0,00	99	0,02
4	100	3,142	3,142	-1,000	0,00	0,00	99	0,03
5	200	3,142	3,142	-1,000	0,00	0,00	97	0,06
	Wartości szukane	3,14	3,14	-1				

Optymalizacja DE (populacja 200) została przedstawiona także na rysunkach 3.4.50 - 3.4.58.

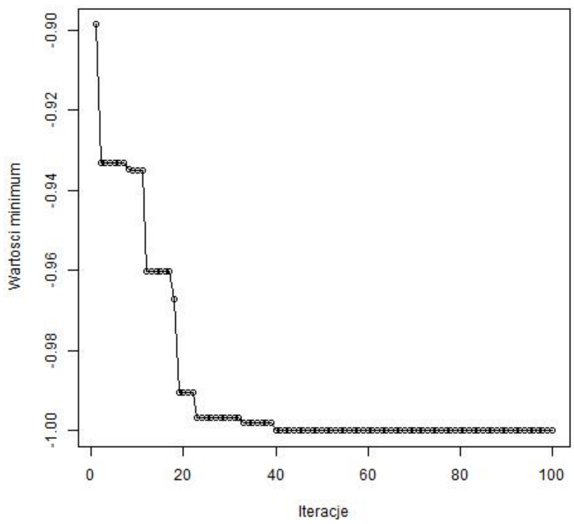
3.4.1. Drop Wave



Rysunek 3.4.50: Iteracja 0, 4 i 8

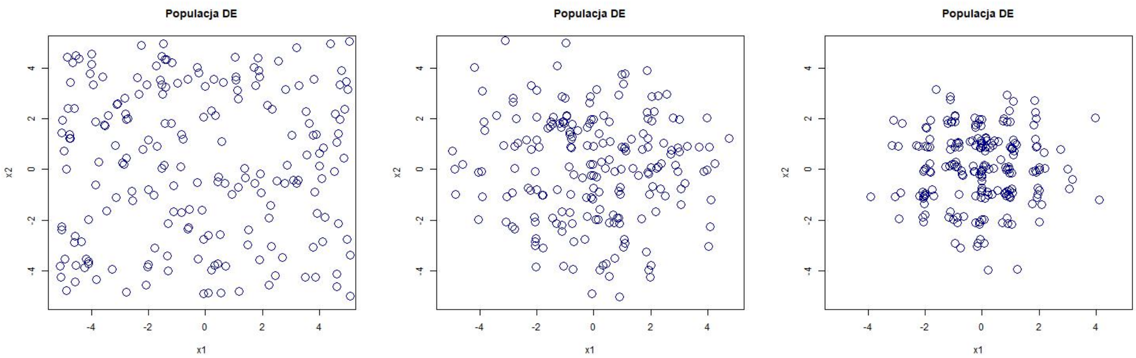


Rysunek 3.4.51: Iteracja 14, 20 i 30

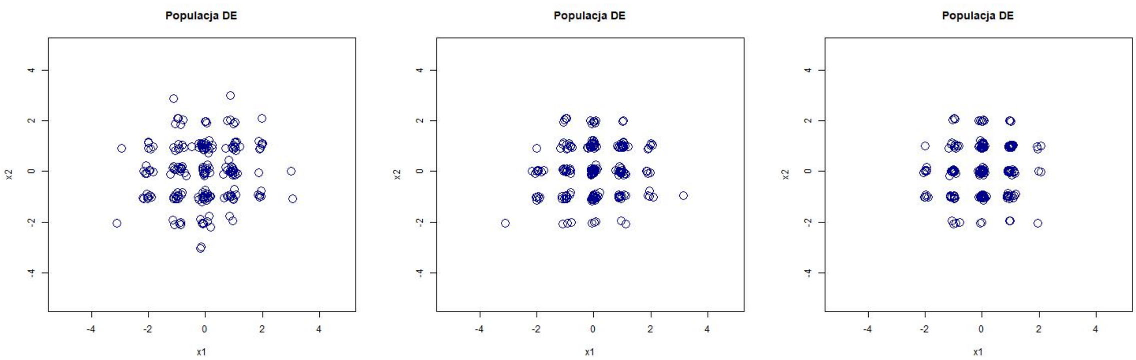


Rysunek 3.4.52: Wartości funkcji z biegiem iteracji

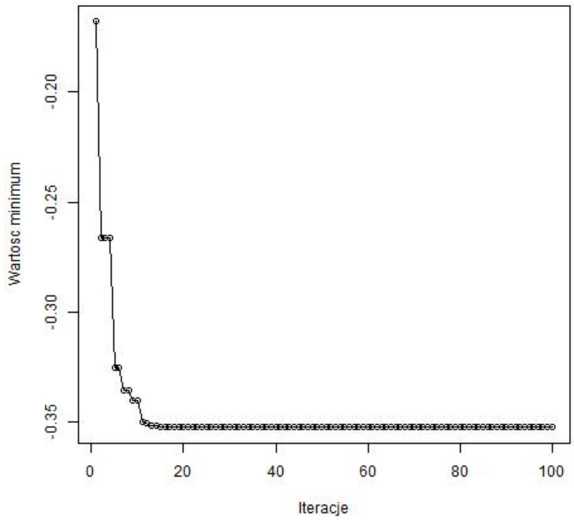
3.4.2. Rastrigin



Rysunek 3.4.53: Iteracja 0, 4 i 8

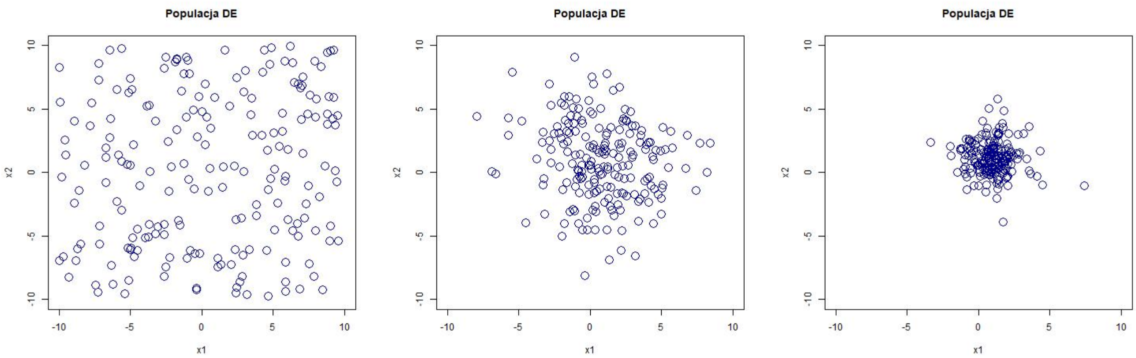


Rysunek 3.4.54: Iteracja 14, 20 i 30

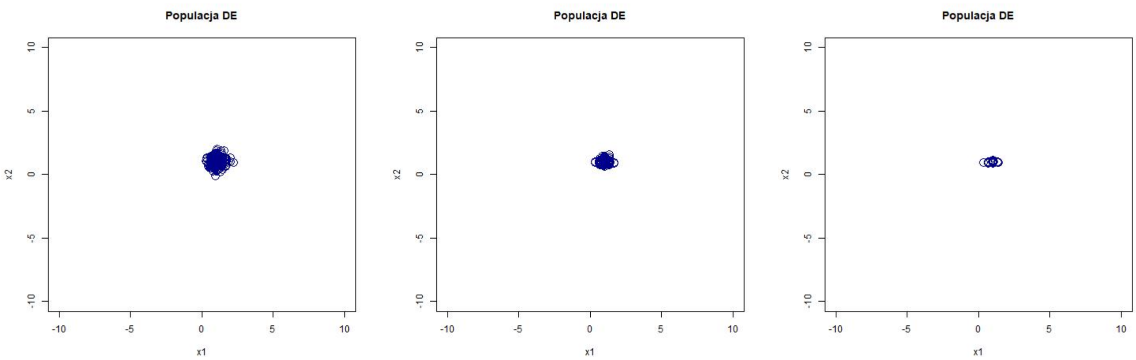


Rysunek 3.4.55: Wartości funkcji z biegiem iteracji

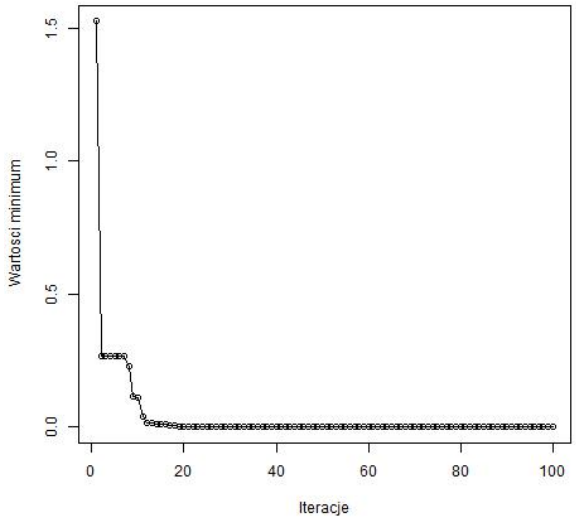
3.4.3. Levy N.13



Rysunek 3.4.56: Iteracja 0, 4 i 8



Rysunek 3.4.57: Iteracja 14, 20 i 30



Rysunek 3.4.58: Wartości funkcji z biegiem iteracji

3.5. Omówienie wyników testów

Otrzymane wyniki z rodziałów 3.4 - 3.7 dotyczące poszukiwania minimów poszczególnych funkcji świadczą o tym, że program prawidłowo optymalizuje funkcje matematyczne o dwóch niewiadomych x1 i x2. Tabele przedstawione w poprzednich rozdziałach oraz rysunki są potwierdzeniem właściwego działania programu. W wynikach można zauważyć wzrost dokładności rozwiązań wraz ze wzrostem liczbności roju szukającego rozwiązania co jest prawidłowe. Także wykresy obrazujące proces optymalizacji potwierdzają właściwe działanie kodu. Wyniki poszczególnych algorytmów dla tych samych funkcji matematycznych są często bardzo różne, za co odpowiadają różnice w strukturze algorytmów.

Bibliografia

[1] Kennedy J., Eberhart R.: *Particle swarm optimization*, Proceedings of the International Conference on Neural Network, pp. 1942-1948, 1995.

[2] Engelbrecht A.: *Particle Swarm Optimization: Velocity Initialization*, in Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2012, Brisbane, Australia, June 10-15, pp. 70-77, 2012.

[3] Helwig S., Branke J., Mostaghim S.: *Experimental analysis of bound handling techniques in particle swarm optimization*, IEEE Transactions on Evolutionary Computation, Vol. 17, No. 2, pp. 259-271, 2013.

[4] Chen S., Montgomery J., Bolufé-Röhler A., Gonzalez-Fernandez Y.: *Standard particle swarm optimization on the CEC2013 real parameter optimization benchmark functions (revised)*, Technical Report, School of Information Technology, York University, Toronto, Ontario, December 2013.

[5] Yang X.S.: *Nature-Inspired Metaheuristic Algorithms*, Luniver Press, 2008.

[6] Yang X.S.: *Firefly algorithms for multimodal optimization*, Stochastic Algorithms: Foundations and Applications, SAGA, Lecture Notes in Computer Sciences, 5792, 169–178, 2009.

[7] Łukasik S., Żak S., *Firefly algorithm for continuous constrained optimization task*, Computational Collective Intelligenc, Semantic Web, Social Networks and Multiagent Systems LNCS, 5796, 97–106, 2009.

[8] Yang X.S., *A new metaheuristic bat-inspired algorithm*, in Nature Inspired Cooperative Strategies for Optimization (NICSO 2010), vol. 284, pp. 65–74, Springer, 2010.

[9] Xie J., Zhou Y., Chen H., *A novel bat algorithm based on differential operator and Lévy flights trajectory*, Computational Intelligence and Neuroscience, vol. 2013, Article ID 453812, 13 pages, 2013.

[10] Xie J., Zhou Y., Zheng H., *A hybrid bat algorithm with path relinking for capacitated vehicle routing problem*, Mathematical Problems in Engineering, vol. 2013, Article ID 392789, 10 pages, 2013.

[11] Xie J., Zhou Y., Zheng H.,*A hybrid metaheuristic for multiple runways aircraft landing problem based on bat algorithm*, Journal of Applied Mathematics, vol. 2013, Article ID 742653, 8 pages, 2013.

[12] Gandomi A.H., Yang X.S., Alavi A.H., Talatahari S., *Bat algorithm for constrained optimization tasks*, Neural Computing and Applications, vol. 22, no. 6, pp. 1239–1255, 2013.

[13] Yang X.S., Hossein Gandomi A., *Bat algorithm: a novel approach for global engineering optimization*, Engineering Computations, vol. 29, no. 5, pp. 464–483, 2012.

[14] Yang X.S., *Bat algorithm for multi-objective optimisation*, International Journal of Bio-Inspired Computation, vol. 3, no. 5, pp. 267–274, 2011.

[15] A. Rezaee Jordehi, *Chaotic bat swarm optimisation (CBSO)*, Applied Soft Computing, vol. 26, pp. 523–530, 2015.

[16] Zhu B., Zhu W., Liu Z., Duan Q., Cao L., *A Novel Quantum-Behaved Bat Algorithm with Mean Best Position Directed for Numerical Optimization*, Computational Intelligence and Neuroscience Volume 2016, Article ID 6097484, 17 pages, 2016.

[17] Michalewicz Z., *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag Berlin Heidelberg, 1992.

[18] Vose M. D., Leipins G.E.: *Schema disruption, In Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 237-243, Morgan Kaufmann (San Mateo), 1991.

[19] Kozieł S.: *Algorytmy ewolucyjne i ich zastosowania do optymalizacji i modelowania analogowych układów elektronicznych*, Rozprawa doktorska, Politechnika Gdańska, Wydział Elektroniki, Telekomunikacji i Informatyki, Gdańsk 1999.

[20] Radcliffe N. J., Surry P. D.: *Fundamental Limitations on Search Algorithms: Evolutionary Computing in Perspective*, Lecture Notes in Computer Science, Vol. 1000, J. Van Leeuwen (ed.), Springer-Verlag, 1995.

[21] Słowik A.: *Właściwości i zastosowania algorytmów ewolucyjnych w optymalizacji*, Metody Informatyki Stosowanej, nr 2/2007 Kwartalnik Komisji Informatyki Polskiej Akademii Nauk Oddział w Gdańsku.

[22] Arabas J.: *Wykłady z algorytmów ewolucyjnych*, WNT, 2004.

[23] Storn R., Price K., *Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces*, Journal of Global Optimization, 11(4), s. 341-359, 1997.

[24] R Core Team (2017). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

[25] RStudio Team (2016). RStudio: Integrated Development for R. RStudio, Inc., Boston, MA URL http://www.rstudio.com/.

[26] Ciupke K.: psoptim: Particle Swarm Optimization, R package version 1.0, 2016, URL: https://CRAN.R-project.org/package=psoptim.

[27] Hwang S. H., Moon R. M., microbats: An Implementation of Bat Algorithm in R, R package version 0.1-1, 2016, URL: https://CRAN.R-project.org/package=microbats, https://github.com/stathwang/microbats.

[28] Scrucca, L. (2013) GA: A Package for Genetic Algorithms in R. Journal of Statistical Software, 53(4), 1-37. https://www.jstatsoft.org/article/view/v053i04.

[29] Ardia, D., Boudt, K., Carl, P., Mullen, K.M., Peterson, B.G. (2011) Differential Evolution with DEoptim. An Application to Non-Convex Portfolio Optimization. The R Journal, 3(1), 27-34. URL: https://journal.r-project.org/archive/2011-1/RJournal_2011-1_Ardia~et~al.pdf. Differential Evolution homepage: URL http://www.icsi.berkeley.edu/~storn/code.html.

[30] Surjanovic, S. & Bingham, D. (2013). Virtual Library of Simulation Experiments: Test Functions and Datasets. Retrieved August 9, 2017, from http://www.sfu.ca/∼ssurjano.

[31] Jamil M., Yang X. S., A literature survey of benchmark functions for global optimization problems, Int. Journal of Mathematical Modelling and Numerical Optimisation, Vol. 4, No. 2, pp. 150–194 (2013).