Opracowanie statystyczne działania algorytmów minimalizacji stochastycznej

Krzysztof Solecki, Paweł Jaśkowiec

2023-01-25

Tytuł projektu: Rachunek Prawdopodobieństwa i Statystyka - Projekt Zaliczeniowy 2022/23

Autorzy: Paweł Jaśkowiec, Krzysztof Solecki

Opis:

Projekt polega na prostym opracowaniu statystycznym wyników porównania działania wybranych algorytmów minimalizacji stochastycznej. W naszym przypadku dotyczy to algorytmu poszukiwania przypadkowego PRS (Pure Random Search) oraz Algorytmu Genetycznego (GA).

Teoria:

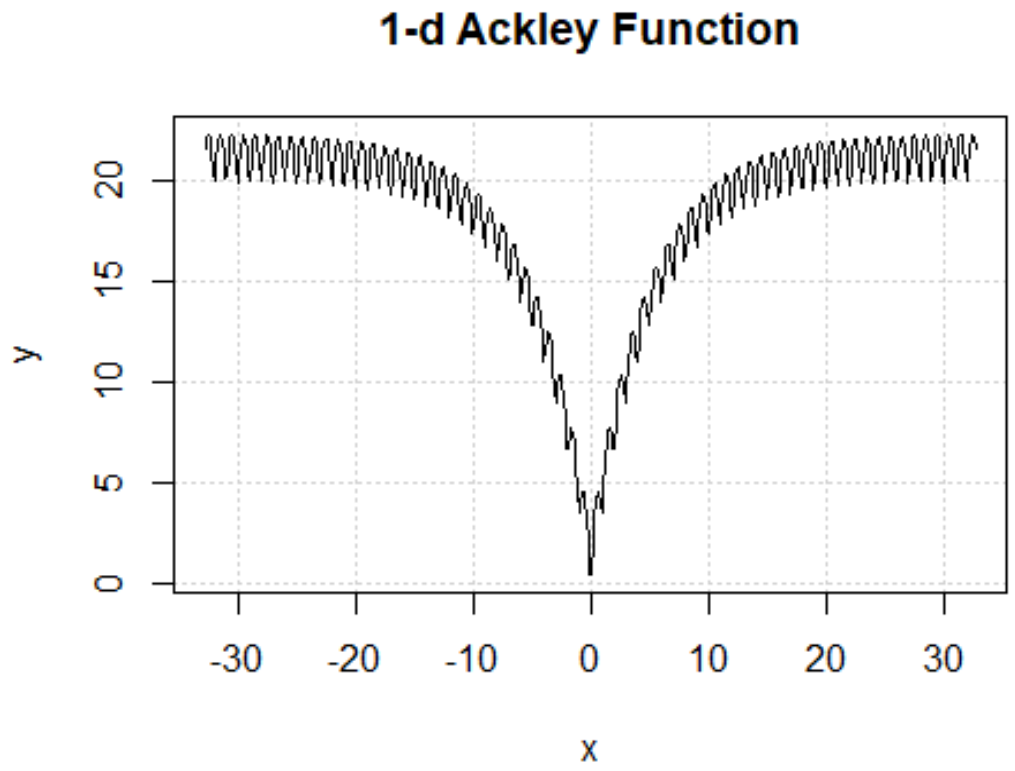
Algorytm genetyczny to rodzaj algorytmu przeszukującego przestrzeń alternatywnych rozwiązań problemu w celu wyszukania rozwiązań najlepszych. Sposób działania algorytmów genetycznych nieprzypadkowo przypomina zjawisko ewolucji biologicznej, ponieważ ich twórca John Henry Holland właśnie z biologii czerpał inspiracje do swoich prac. Algorytmy genetyczne znalazły zastosowanie do rozwiązywania problemów optymalizacji, rozpoznawania obrazów, przewidywaniu ruchów na giełdzie, tworzenia grafiki oraz projektowania budynków i maszyn.

Poszukiwanie przypadkowe (Pure Random Search, PRS) - losujemy po kolei zadaną z góry liczbę punktów z rozkładem jednostajnym w zadanej dziedzinie. Jeżeli dziedzina jest kostką wielowymiarową, to losujemy kolejno współrzędne poszczególnych punktów według odpowiedniego jednowymiarowego rozkładu jednostajnego.

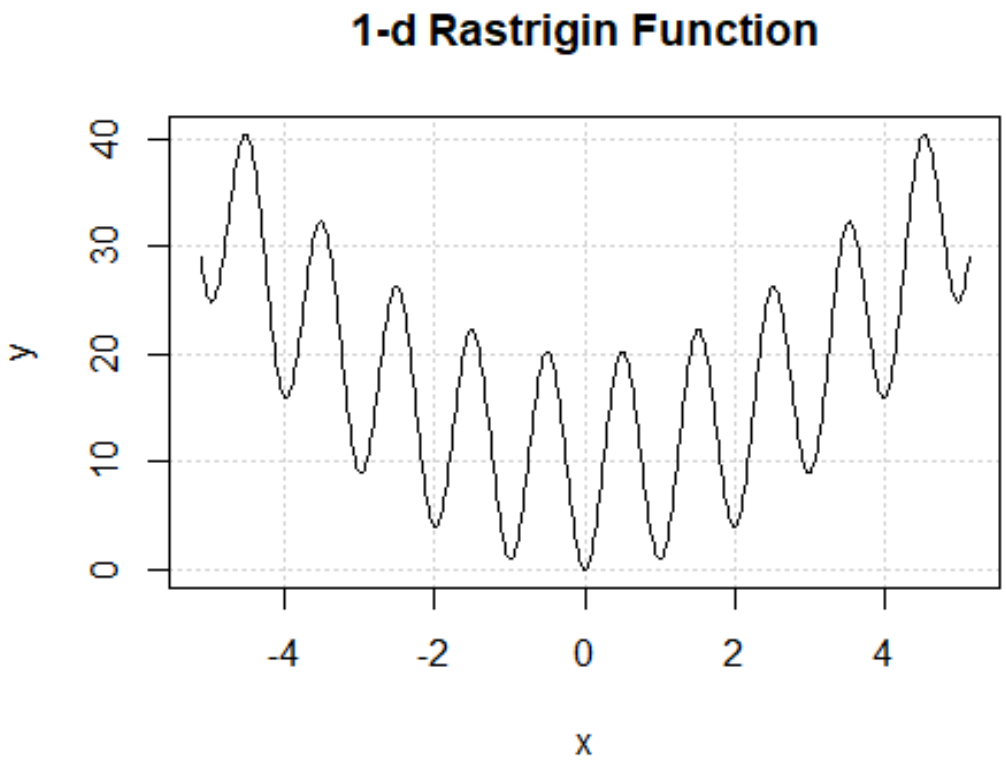
```
library(smoof)

library(GA)

library(vioplot)
```

Do porównania algorytmów PRS i GA użyjemy dwóch funkcji dostępnych w bibliotece smoof: Rastrigina i Ackley'a. Są to funkcje skalarne i wielomodalne oraz akceptują parametr określający ilość wymiarów. My bedziemy analizować dla 2,10 i 20 wymiarów.

```
plot(smoof::makeAckleyFunction(1))
```



```
plot(smoof::makeRastriginFunction(1))
```



Poszukiwanie przypadkowe (PRS)

Poniższa funkcja PRS przyjmuje jako argument funkcję oraz budżet obliczeniowy, czyli ilość wywołań minimalizowanej funkcji. W naszym przypadku ustalamy budżet równy 1000.

```
prs <- function(f,N){
  upper <- smoof::getUpperBoxConstraints(f)
  lower <- smoof::getLowerBoxConstraints(f)
  dim <- length(upper)

  return(min(replicate(N,f(runif(dim,lower[1],upper[1])))))
}
```

Algorytm Genetyczny (GA)

Poniższa funkcja ga_min została napisana z użyciem funkcji ga z biblioteki GA, która wyznacza maksimum przekazanej funkcji. Została ona zmodyfikowana tak, aby wyznaczała minimum funkcji. Parametr maxIter został ustawiony na 20, ponieważ dla każdej iteracji domyślnie algorytm przyjmuje Population Size = 50, co oznacza, że dla jednej iteracji znajduje 50 rozwiązań, a skoro nasz narzucony budżet wynosi 1000 to możemy wykonać dokładnie 20 takich iteracji.

```
ga_min <- function(f){

  res <- GA::ga(type="real-valued",
                fitness = function(x) -1*f(x),
                lower = smoof::getLowerBoxConstraints(f),
                upper = smoof::getUpperBoxConstraints(f),
                maxiter = 20,
                monitor = FALSE)
  return((-1)*res@fitnessValue)
}
```

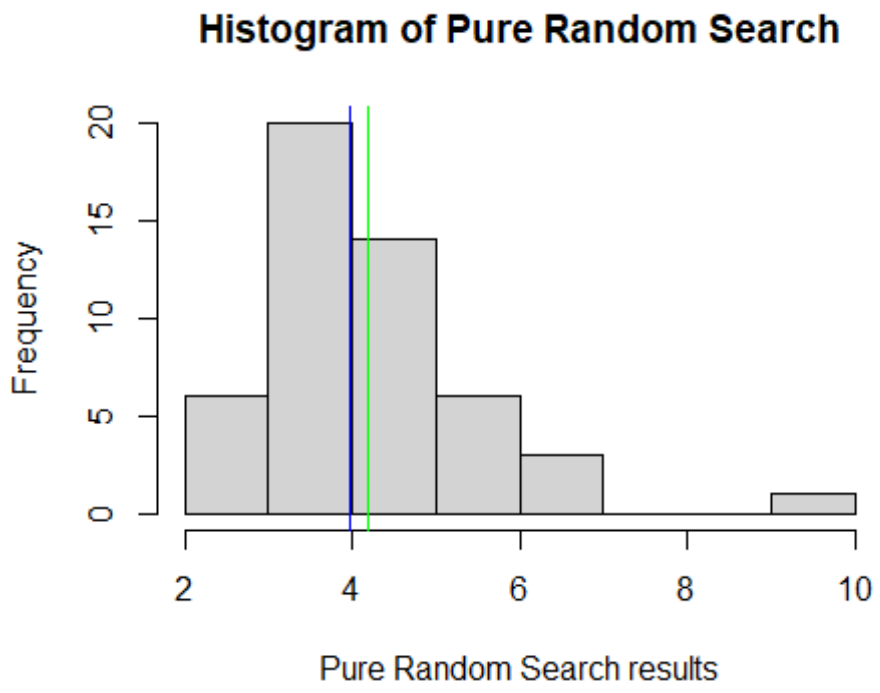
Poniżej dokonano łącznie 6 porównań wybranych algorytmów: 3 różne liczby wymiarów dla każdej z 2 wybranych funkcji. W tym celu użyto funkcji compare, która zapisuje odpowiednio wynik funkcji prs oraz ga_min dla określonej funkcji i wymiarów. Następnie oblicza średnią, medianę, rysuje histogramy i wykresy skrzypcowe z uwzględnieniem rozproszenia wyników dla obu algorytmów.Zieloną linią zaznaczono średnią z wyników, natomiast niebieską - medianę.

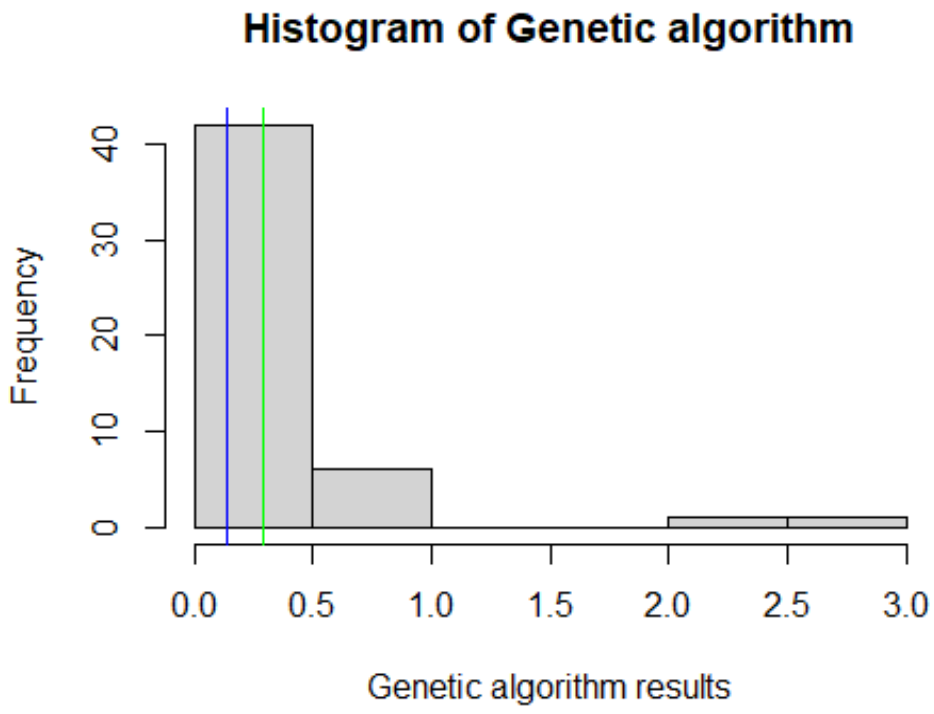
```
compare <- function(f){
  prs_res <- replicate(50,prs(f,1000))
  ga_res <- replicate(50,ga_min(f))

  mean(prs_res)
  mean(ga_res)

  hist(prs_res,
       main="Histogram of Pure Random Search",
       xlab="Pure Random Search results",
       ylab="Frequency")
  abline(v = median(prs_res),col = "blue")
  abline(v = mean(prs_res), col = "green")

  hist(ga_res,
       main="Histogram of Genetic algorithm",
       xlab="Genetic algorithm results",
```

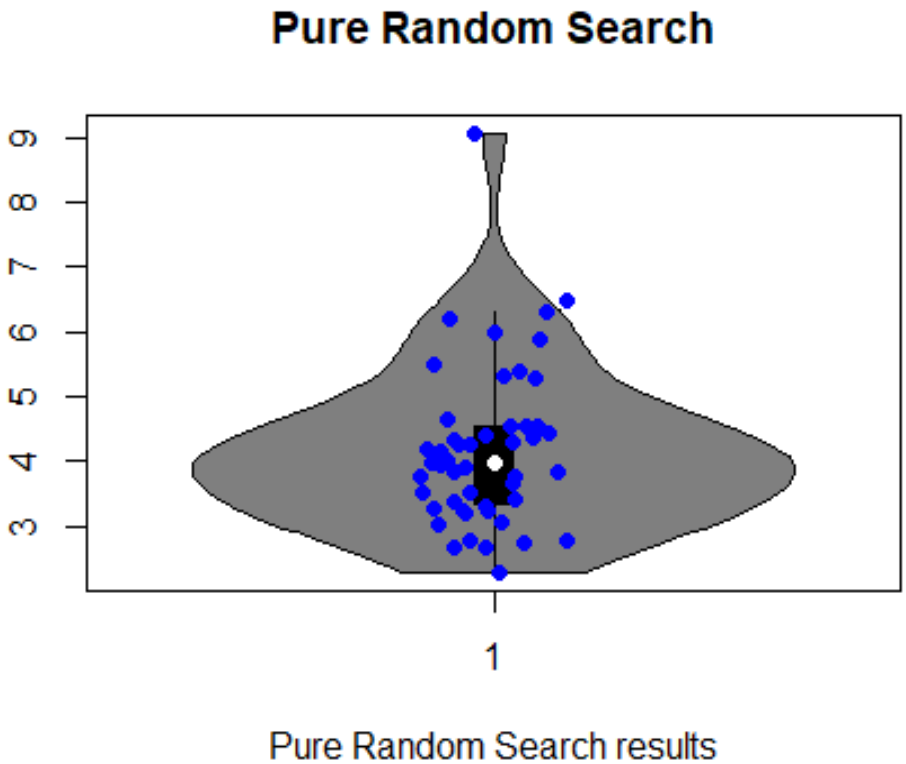
```
       ylab="Frequency")
  abline(v = median(ga_res),col = "blue")
  abline(v = mean(ga_res),col = "green")

  vioplot::vioplot(prs_res,
                   main="Pure Random Search",
                   xlab="Pure Random Search results",
                   ylab="")
  stripchart(prs_res, method = "jitter", col = "blue",
             vertical = TRUE, pch = 19, add = TRUE)

  vioplot::vioplot(ga_res,
                   main="Genetic algorithm",
                   xlab="Genetic algorithm results",
                   ylab="")
  stripchart(ga_res, method = "jitter", col = "blue",
             vertical = TRUE, pch = 19, add = TRUE)

}
```

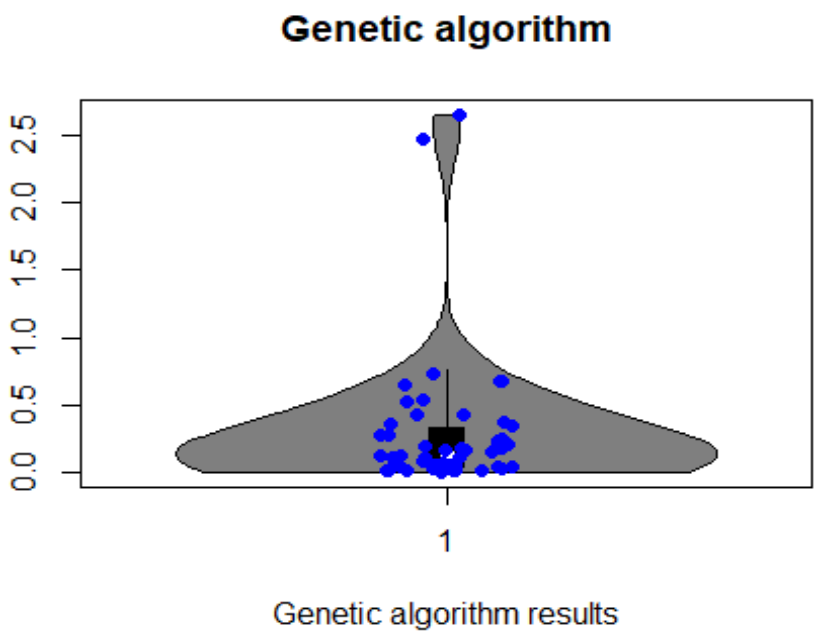
Funkcja Ackleya, liczba wymiarów: 2

```
compare(smoof::makeAckleyFunction(2))
```



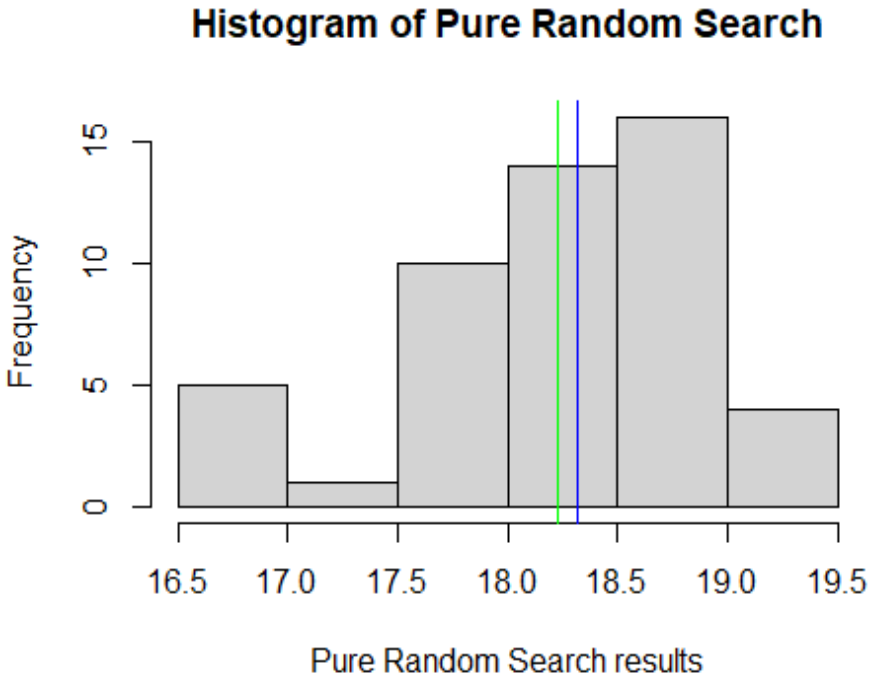

Histogram of Genetic algorithm
Frequency
0
10
20
30
40
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Genetic algorithm results

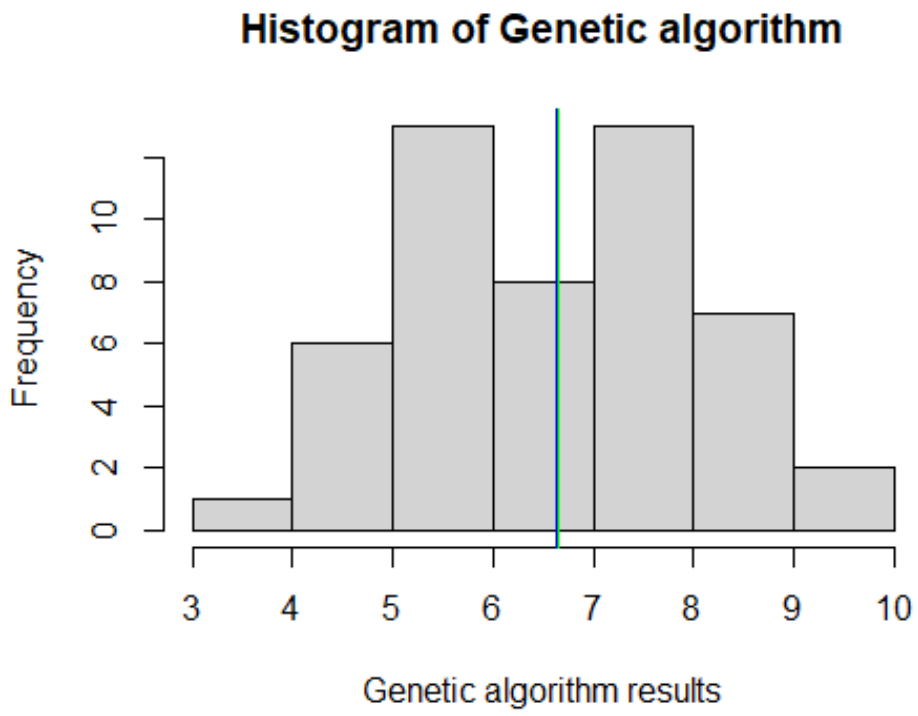

Pure Random Search
3
4
5
6
7
8
9
1
Pure Random Search results

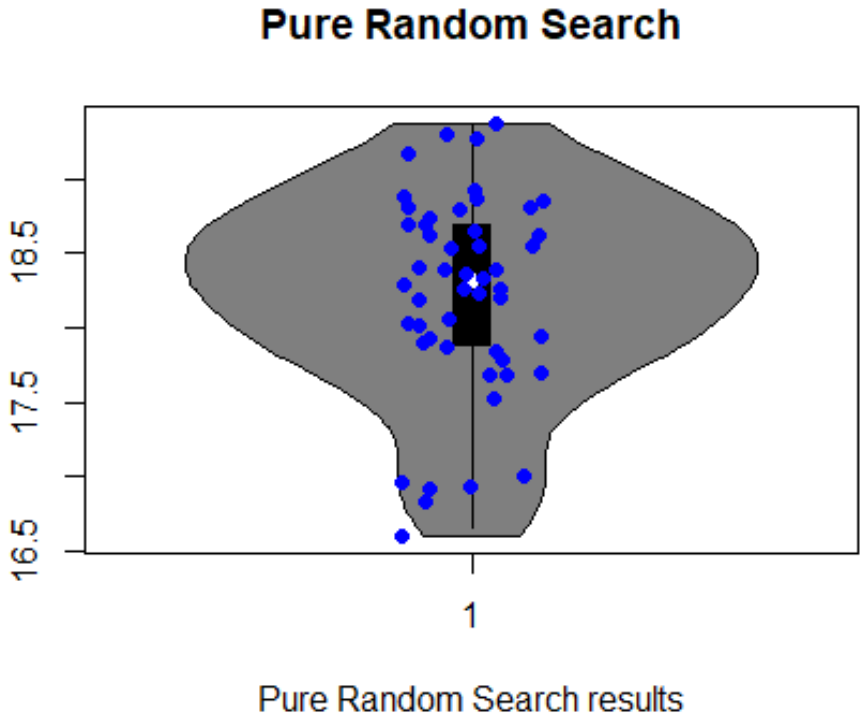


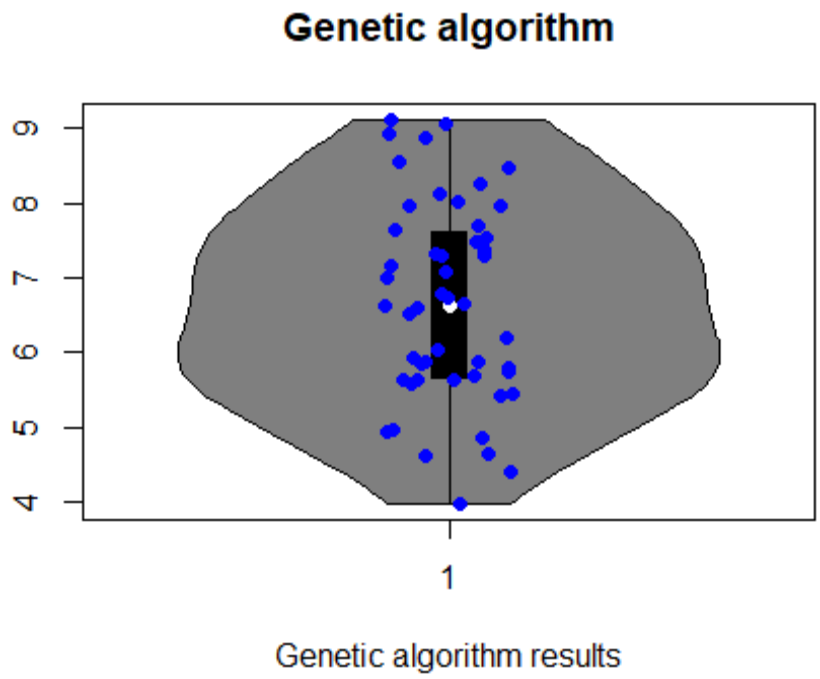
Funkcja Ackleya, liczba wymiarów: 10

```
compare(smoof::makeAckleyFunction(10))
```



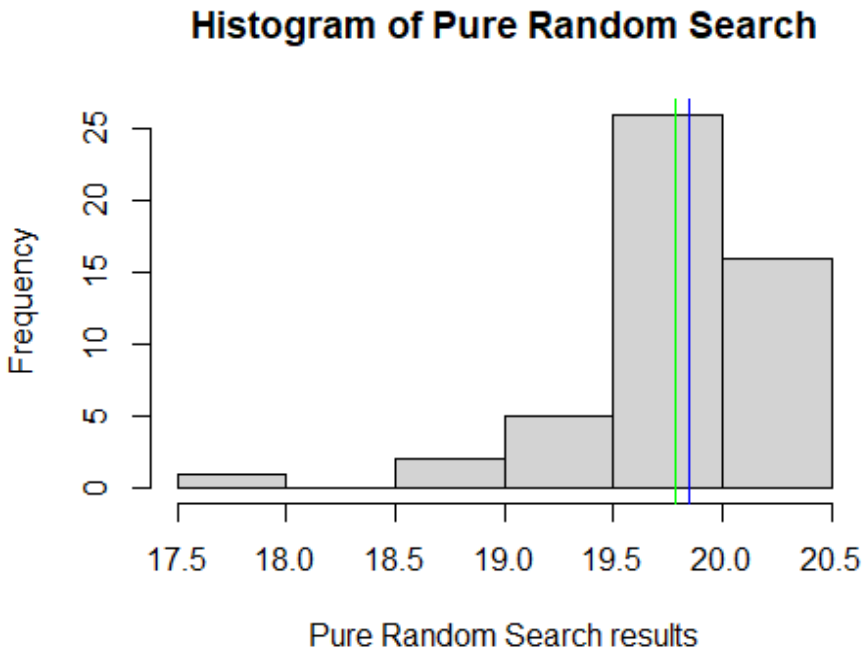

Histogram of Genetic algorithm
Frequency
0
2
4
6
8
10
3
4
5
6
7
8
9
10
Genetic algorithm results

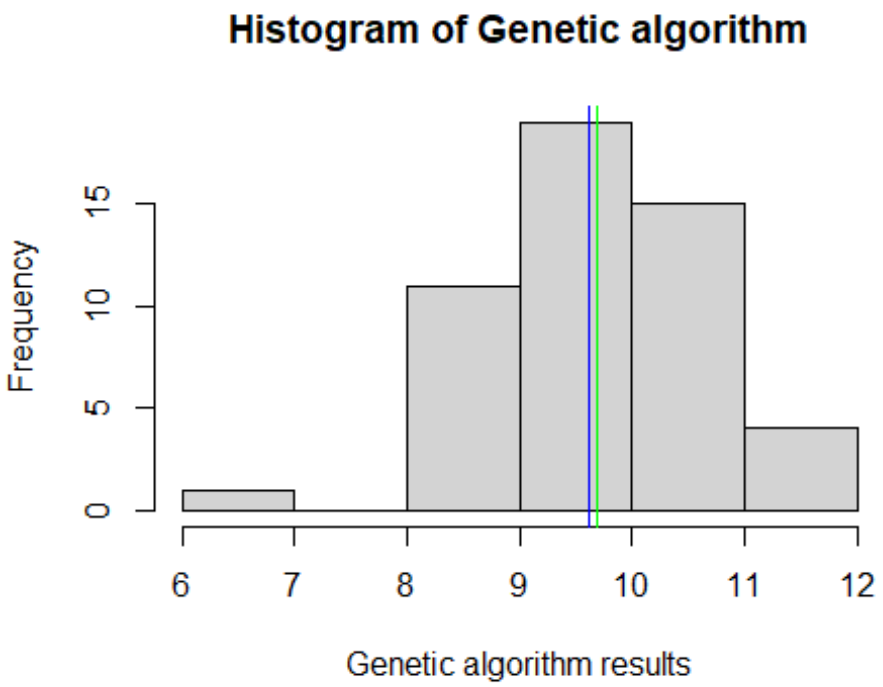

Pure Random Search
16.5
17.5
18.5
1
Pure Random Search results

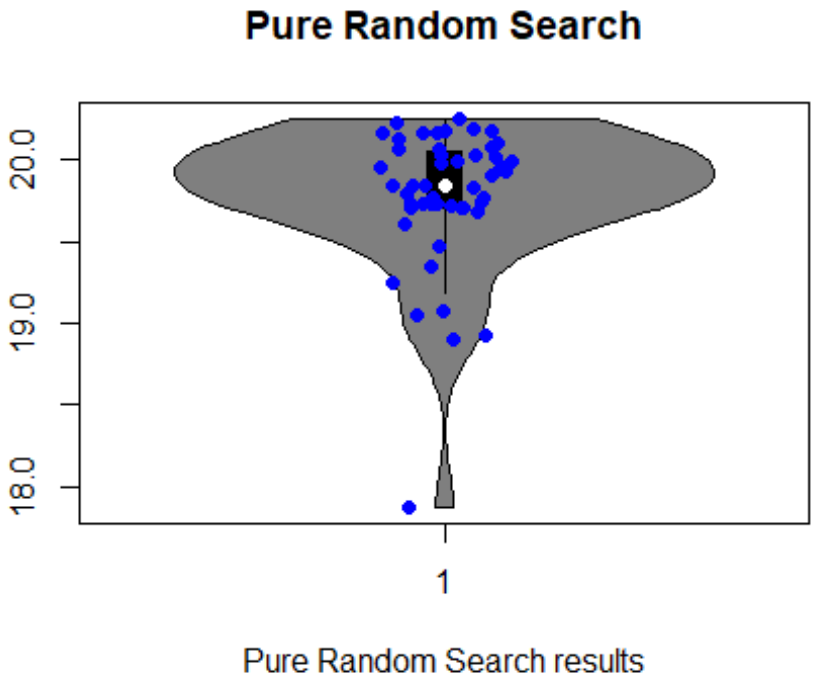


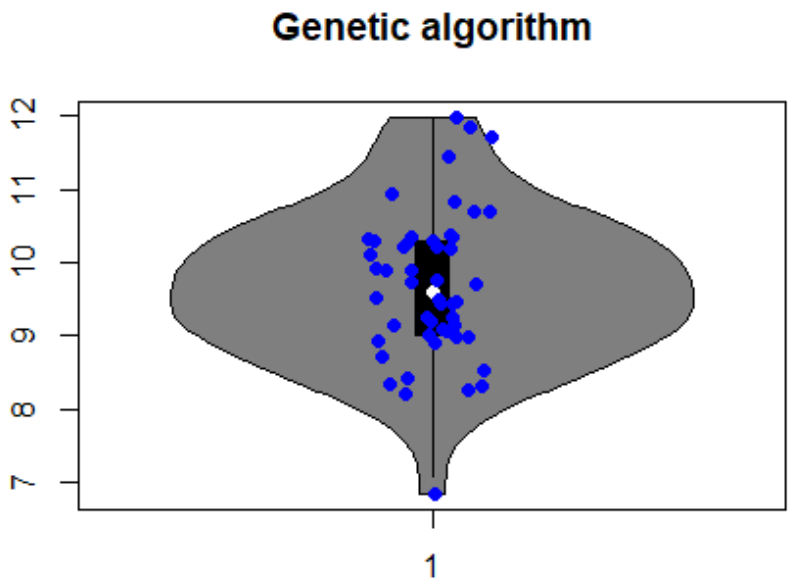
Funkcja Ackleya, liczba wymiarów: 20

```
compare(smoof::makeAckleyFunction(20))
```



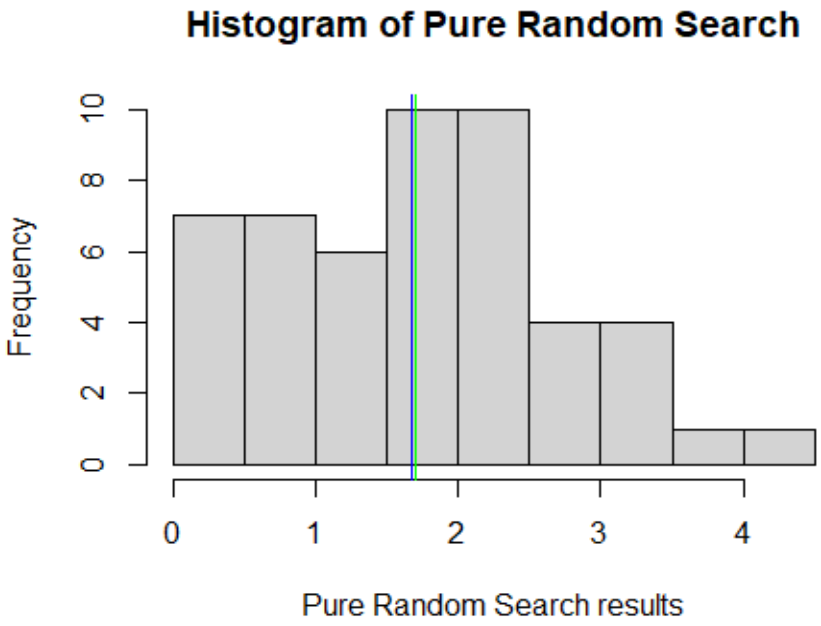

Histogram of Genetic algorithm
Frequency
0
5
10
15
6
7
8
9
10
11
12
Genetic algorithm results

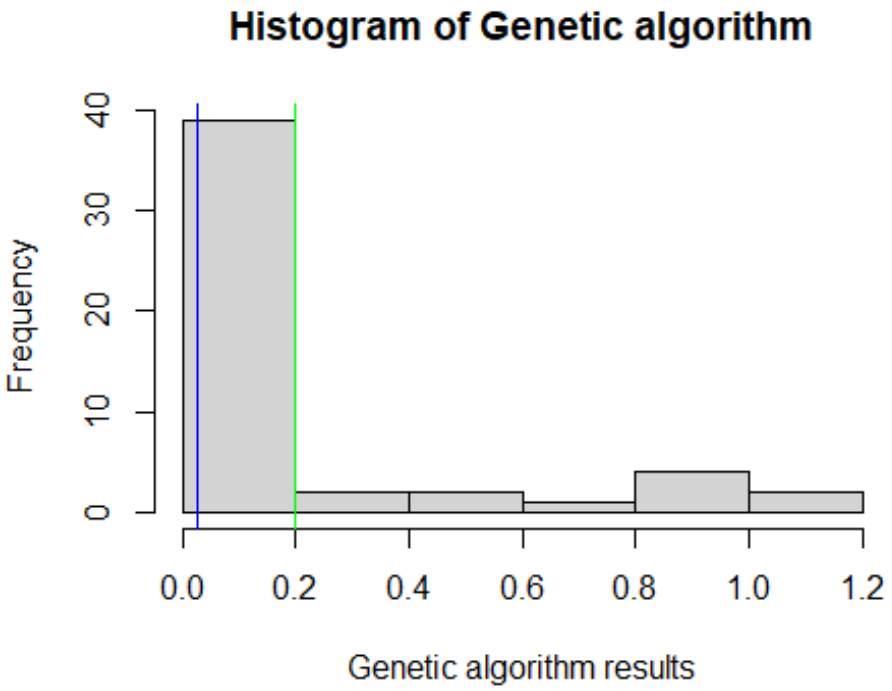

Pure Random Search
18.0
19.0
20.0
1
Pure Random Search results

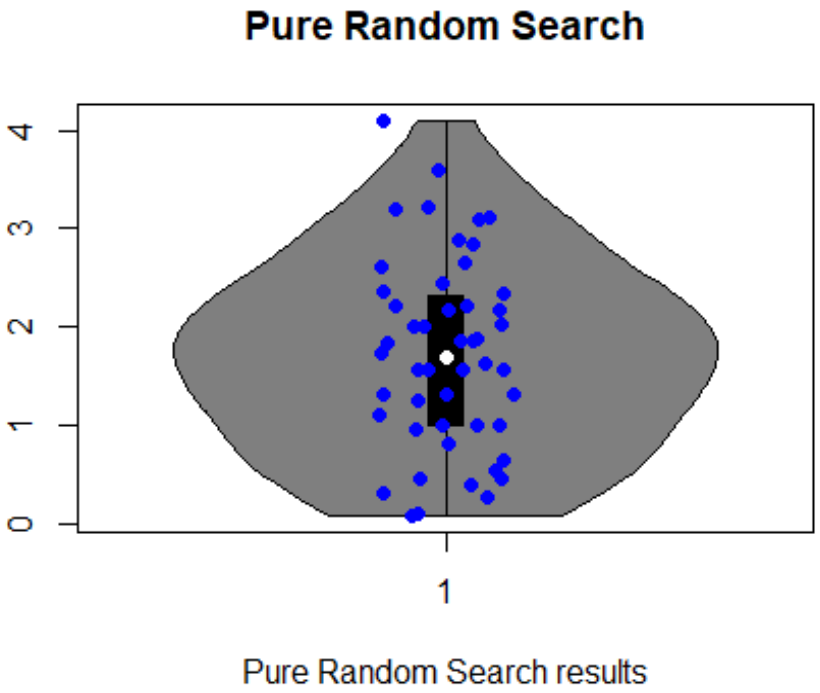


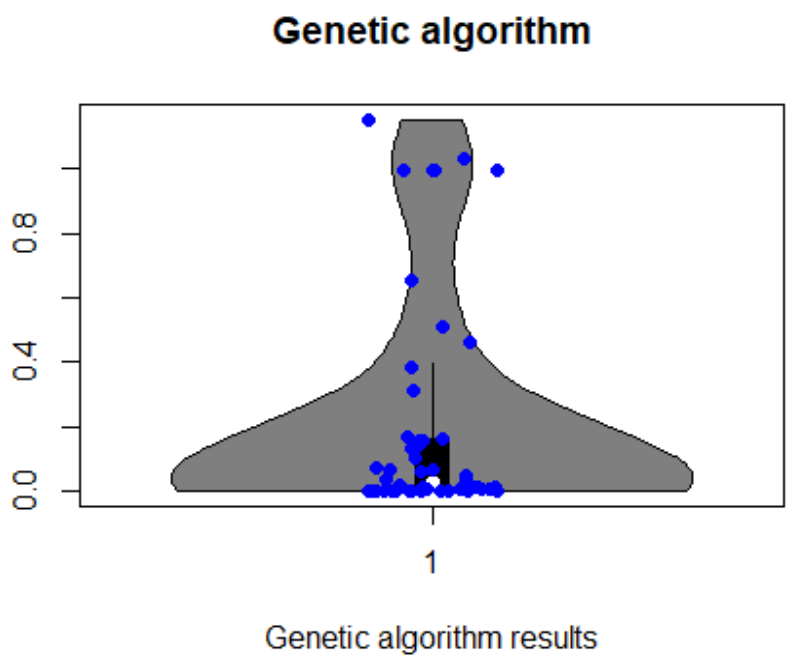
Funkcja Rastrigina, liczba wymiarów: 2

```
compare(smoof::makeRastriginFunction(2))
```



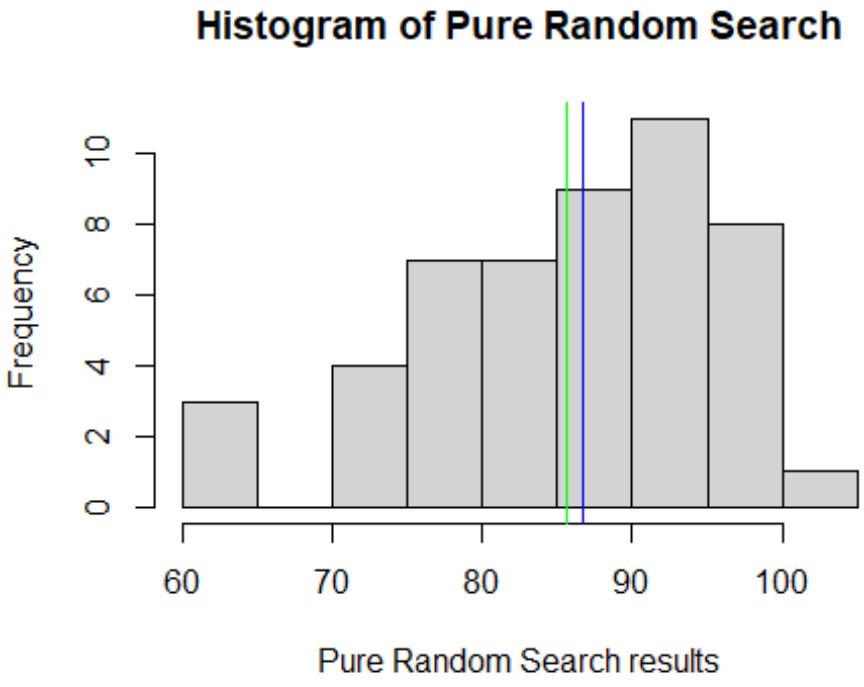

Histogram of Genetic algorithm
Frequency
0
10
20
30
40
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Genetic algorithm results

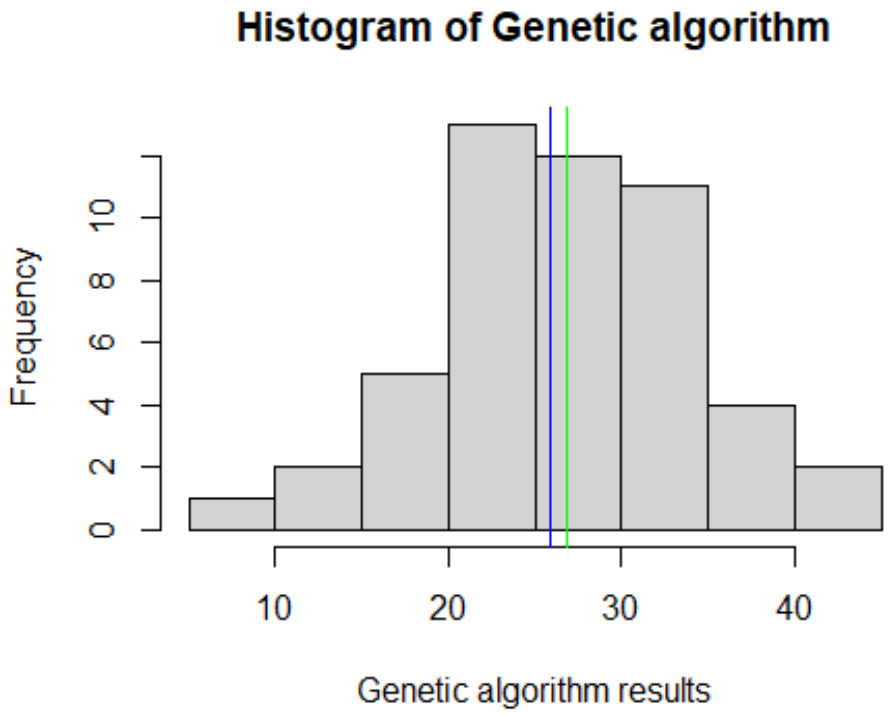

Pure Random Search
0
1
2
3
4
1
Pure Random Search results

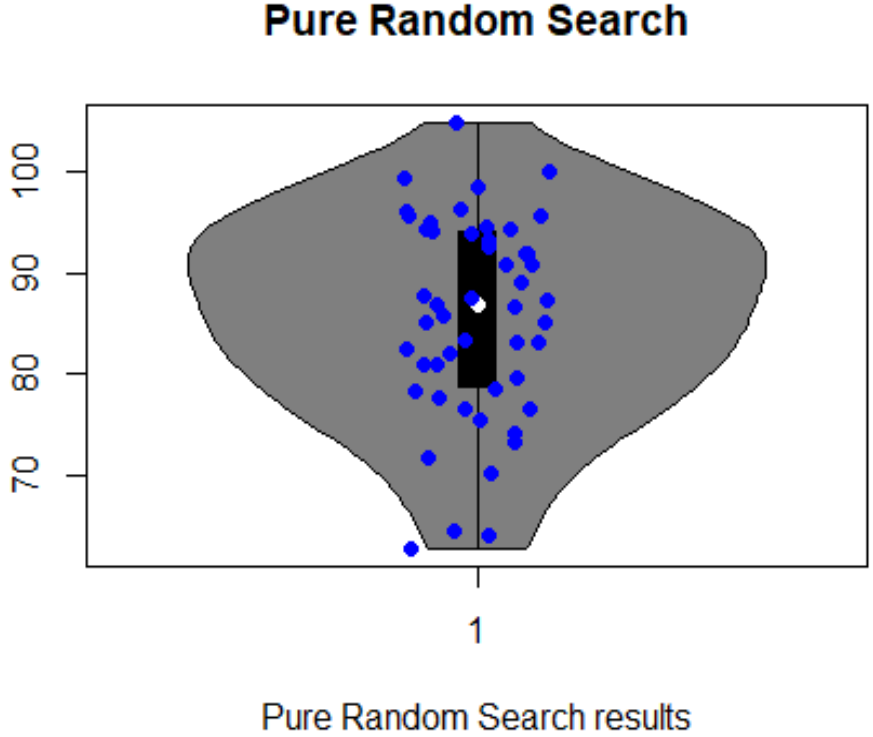


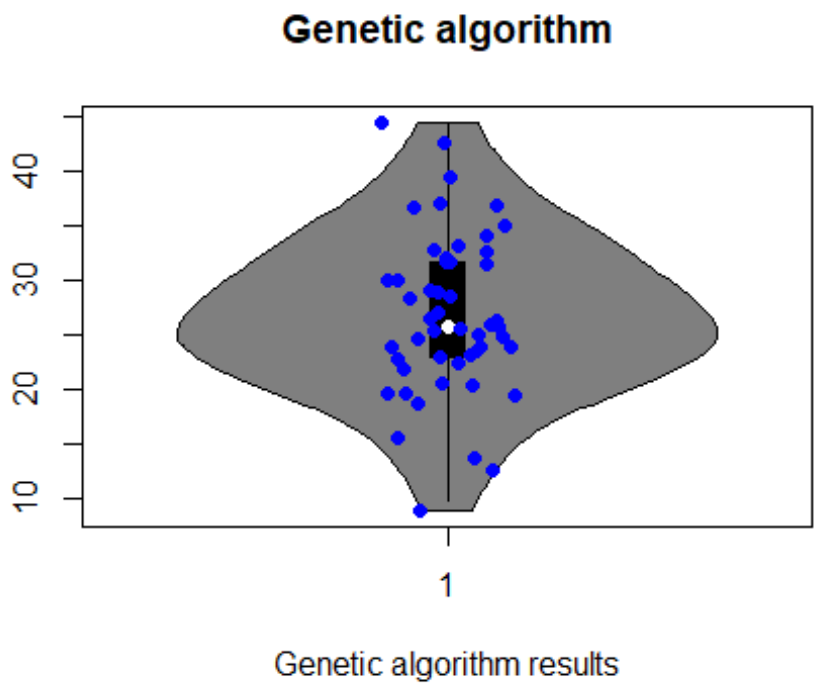
Funkcja Rastrigina, liczba wymiarów: 10

```
compare(smoof::makeRastriginFunction(10))
```



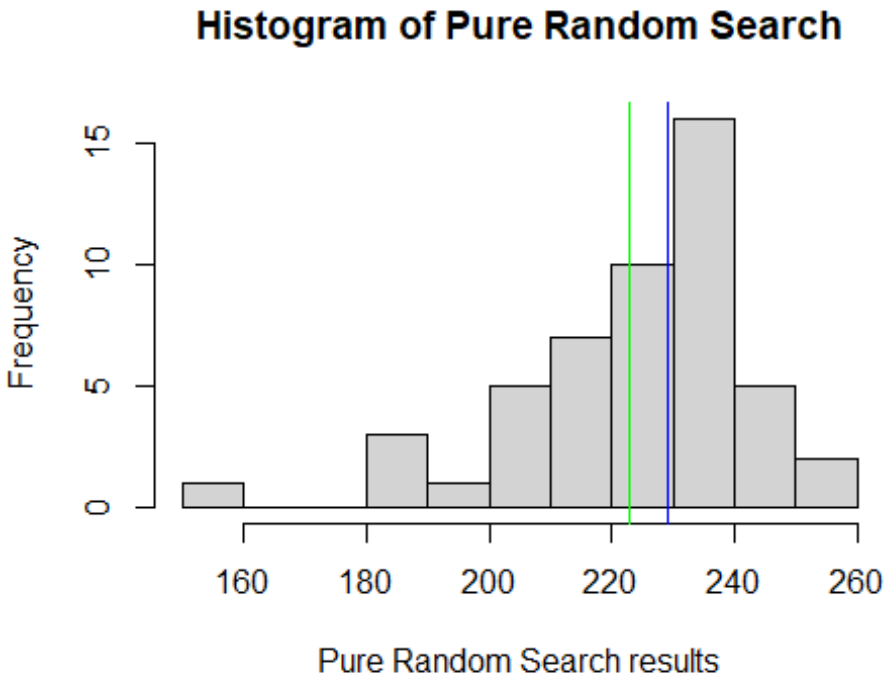

Histogram of Genetic algorithm
Frequency
0
2
4
6
8
10
10
20
30
40
Genetic algorithm results

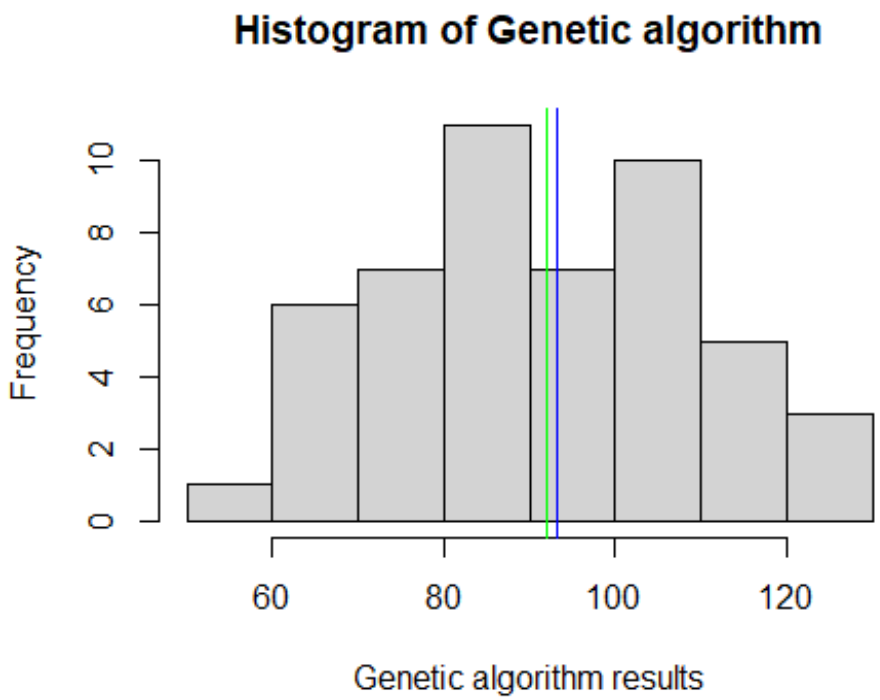

Pure Random Search
70
80
90
100
1
Pure Random Search results

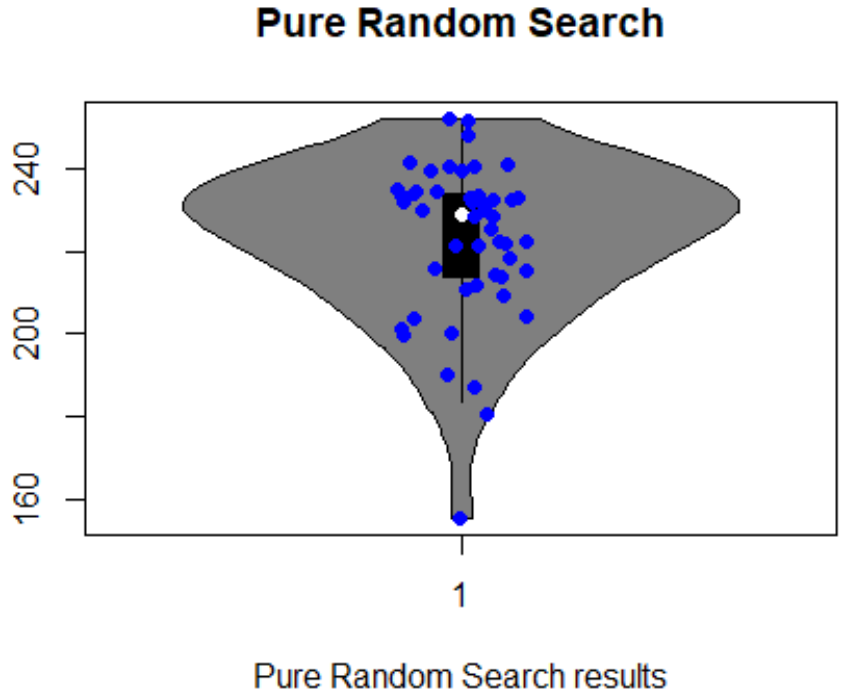


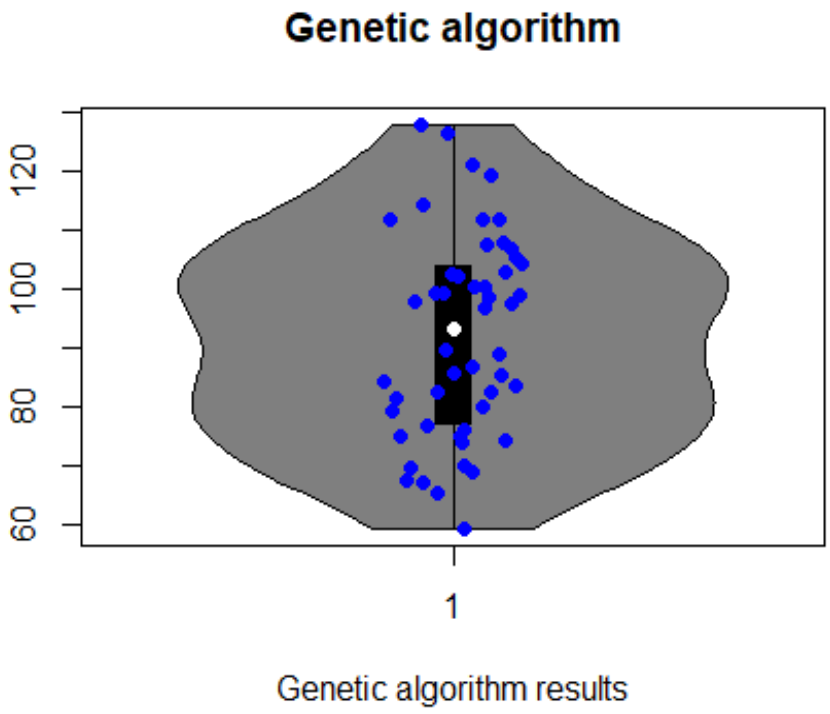
Funkcja Rastrigina, liczba wymiarów: 20

```
compare(smoof::makeRastriginFunction(20))
```




Histogram of Genetic algorithm
Frequency
0
2
4
6
8
10
60
80
100
120
Genetic algorithm results


Pure Random Search
160
200
240
1
Pure Random Search results



Można zauważyć, że algorytm genetyczny wyznacza minimum bliższe wartości rzeczywistej (0) w porównaniu do algorytmu PRS dla jednakowej liczby wywołań minimalizowanej funkcji. Ponadto dla 2 wymiarów wyniki zwracane przez algorytm genetyczny mają niewielki rozrzut i są silnie skoncentrowane wokół wartości średniej niezależnie od testowanej funkcji. W przypadku algorytmu PRS niezależnie od liczby wymiarów wyniki mają duży rozrzut, a ich rozkład jest w przybliżeniu symetryczny względem wartości średniej, zwłaszcza dla liczby wymiarów większej od 2.

Analiza istotności statystycznej różnicy między średnim wynikiem obu algorytmów

```
prs_2_ackley <- replicate(50,prs(smoof::makeAckleyFunction(2),1000))
ga_2_ackley <- replicate(50,ga_min(smoof::makeAckleyFunction(2)))

t.test(prs_2_ackley,ga_2_ackley)

##
##  Welch Two Sample t-test
##
## data:  prs_2_ackley and ga_2_ackley
## t = 21.69, df = 68.475, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  3.570709 4.294175
## sample estimates:
```

```
## mean of x mean of y
##  4.180153  0.247711

prs_10_ackley <- replicate(50,prs(smoof::makeAckleyFunction(10),1000))
ga_10_ackley <- replicate(50,ga_min(smoof::makeAckleyFunction(10)))

t.test(prs_10_ackley,ga_10_ackley)

##
##  Welch Two Sample t-test
##
## data:  prs_10_ackley and ga_10_ackley
## t = 48.378, df = 83.298, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  10.82839 11.75689
## sample estimates:
## mean of x mean of y
## 18.072738  6.780096

prs_20_ackley <- replicate(50,prs(smoof::makeAckleyFunction(20),1000))
ga_20_ackley <- replicate(50,ga_min(smoof::makeAckleyFunction(20)))

t.test(prs_20_ackley,ga_20_ackley)

##
##  Welch Two Sample t-test
##
## data:  prs_20_ackley and ga_20_ackley
## t = 63.167, df = 54.065, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##   9.903492 10.552745
## sample estimates:
## mean of x mean of y
## 19.863537  9.635419

prs_2_rastrigin <- replicate(50,prs(smoof::makeRastriginFunction(2),1000))
ga_2_rastrigin <- replicate(50,ga_min(smoof::makeRastriginFunction(2)))

t.test(prs_2_rastrigin,ga_2_rastrigin)

##
##  Welch Two Sample t-test
##
## data:  prs_2_rastrigin and ga_2_rastrigin
## t = 12.613, df = 60.67, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  1.195253 1.645686
## sample estimates:
```

```
## mean of x mean of y
## 1.5369891 0.1165196

prs_10_rastrigin <- replicate(50,prs(smoof::makeRastriginFunction(10),1000))
ga_10_rastrigin <- replicate(50,ga_min(smoof::makeRastriginFunction(10)))

t.test(prs_10_rastrigin,ga_10_rastrigin)

##
##  Welch Two Sample t-test
##
## data:  prs_10_rastrigin and ga_10_rastrigin
## t = 31.696, df = 88.044, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  54.21617 61.46931
## sample estimates:
## mean of x mean of y
##  85.25995  27.41721

prs_20_rastrigin <- replicate(50,prs(smoof::makeRastriginFunction(20),1000))
ga_20_rastrigin <- replicate(50,ga_min(smoof::makeRastriginFunction(20)))

t.test(prs_20_rastrigin,ga_20_rastrigin)

##
##  Welch Two Sample t-test
##
## data:  prs_20_rastrigin and ga_20_rastrigin
## t = 43.167, df = 86.289, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  132.5070 145.3002
## sample estimates:
## mean of x mean of y
## 229.27902  90.37543
```

Algorytm genetyczny (GA) opiera się na zasadach selekcji naturalnej i genetyki. Rozpoczyna się od populacji rozwiązań, a następnie wykorzystuje selekcję, krzyżowanie i mutację do generowania nowych rozwiązań w każdym pokoleniu. To pozwala GA na eksplorowanie większego przestrzeni rozwiązań i szybsze zbieganie do optymalnego rozwiązania niż PRS. Ponadto, GA jest w stanie utrzymać różnorodność w populacji, co pomaga uniknąć zakleszczenia się w lokalnych optymach.

PRS natomiast generuje nowe rozwiązania losowo. Nie stosuje żadnej specyficznej metody do kierowania przeszukiwaniem i może zająć dużo dłuższy czas, aby znaleźć optymalne rozwiązanie. Nie ma zdolności utrzymywania różnorodności i dlatego ma większe prawdopodobieństwo zakleszczenia się w lokalnym optymum.

Pod względem wydajności GA zwykle jest skuteczniejsze niż PRS dla złożonych i dużych problemów. Jednak PRS może być bardziej efektywny dla prostych problemów z małymi przestrzeniami przeszukiwania.

Podsumowując, GA jest bardziej zaawansowaną i kierowaną techniką optymalizacji, która wykorzystuje zasady z biologii do eksploracji i zbiegania do optymalnych rozwiązań, podczas gdy PRS jest prostszą techniką generującą nowe rozwiązania losowo bez konkretnego kierunku.

Na podstawie naszych testów widać, iż w każdym porównaniu p-wartość jest znacznie mniejsza od 0.05, zatem należy odrzucić hipotezę iż średnie wartości wyników zwracanych przez oba algorytmu są równe. W takim razie istnieje statystycznie istotnia różnica między tymi algorytmami na korzyść algorytmu genetycznego, który zwraca wartości bliższe rzeczywistej wartości minimalnej danych funkcji.