Analiza działania algorytmów minimalizacji stochastycznej

Patryk Klatka

Szymon Twardosz

Wstęp

Tematem projektu była analiza działania algorytmów minimalizacji stochastycznej dwóch algorytmów: PRS (Pure Random Search) oraz GA (Genetic Algorithm). Oba algorytmy zostały przetestowane na dwóch funkcjach z pakietu `smoof` z wymiarami: 2, 10, 20. Każdy algorytm został uruchomiony 50 razy, a każde uruchomienie (budżet obliczeniowy) miało 1000 wywołań. Każda funkcja została przetestowana na dwóch algorytmach, a wyniki zostały porównane pod kątem średnich znalezionych minimów, położenia rozkładu wyników oraz istotności statystycznej różnicy między średnim wynikiem obu algorytmów.

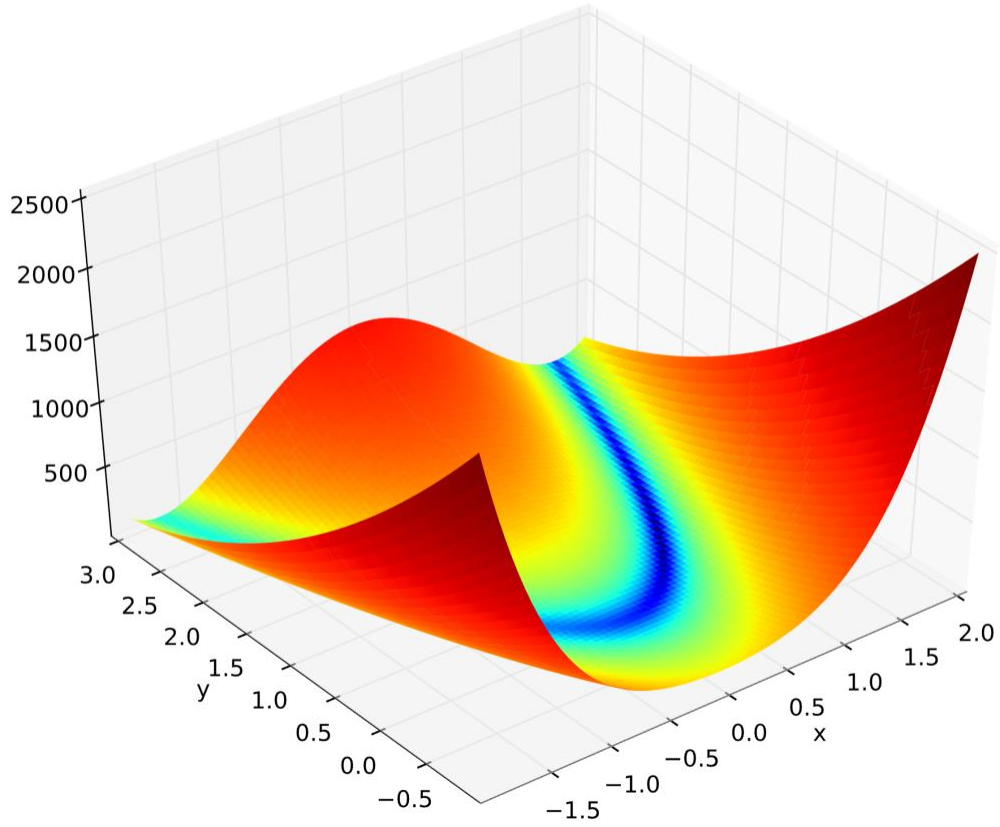
Testowane funkcje

W celu przetestowania algorytmów zostały wybrane dwie funkcje z pakietu `smoof`: funkcja Rosenbrocka oraz Rastrigina. Każda funkcja została wygenerowana na trzech różnych wymiarach: 2, 10, 20. Obie badane funkcje osiągają globalne minimum równe 0. Dziedziny funkcji, zgodnie z dokumentacją pakietu, wynoszą:

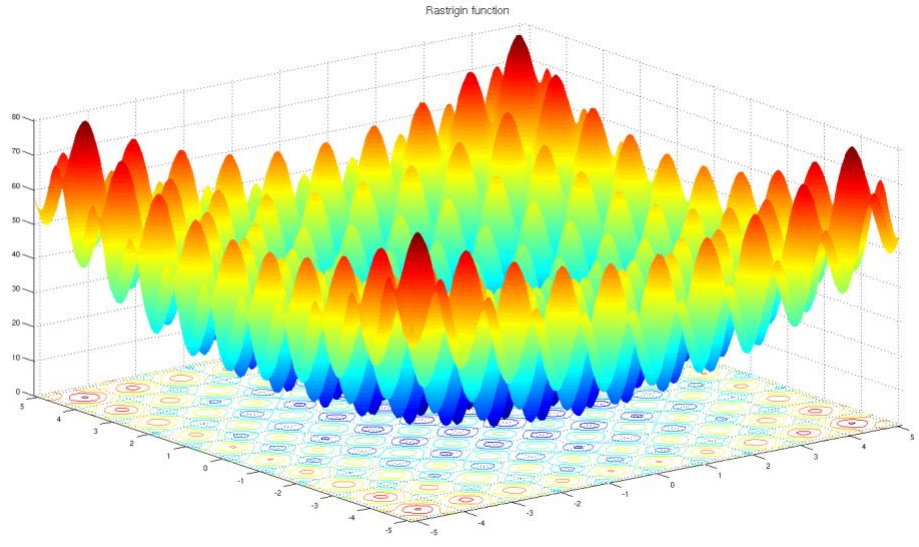
- Funkcja Rosenbrocka: $x_i \in [-5,10]$ dla $i = 1,2,\dots,n$
- Funkcja Rastrigina: $x_i \in [-5.12,5.12]$ dla $i = 1,2,\dots,n$

gdzie n oznacza wymiar funkcji.

```
# Rosenbrock function
rosenbrock_2D <- makeRosenbrockFunction(dimensions = 2L)
rosenbrock_10D <- makeRosenbrockFunction(dimensions = 10L)
rosenbrock_20D <- makeRosenbrockFunction(dimensions = 20L)

# Rastrigin function
rastrigin_2D <- makeRastriginFunction(dimensions = 2L)
rastrigin_10D <- makeRastriginFunction(dimensions = 10L)
rastrigin_20D <- makeRastriginFunction(dimensions = 20L)
```



Funkcja Rosenbrocka



Funkcja Rastrigina

Stałe i funkcje pomocnicze

W celu ułatwienia analizy oraz łatwej zmiany parametrów zostały zdefiniowane stałe, oraz funkcje pomocnicze.

```
# Define number of calculations for each algorithm
number_of_calculations <- 50L

loadData <- function(filename) {
  #' Loads data from file
  #'
  #' @param filename Name of the file
  #' @return Returns data from file, otherwise NULL if file does not exist

  if (file.exists(filename)) {
    return(readRDS(filename))
  }else {
    return(NULL)
  }
}

saveData <- function(data, filename) {
  #' Saves data to file
  #'
  #' @param data Data to save
  #' @param filename Name of the file
  #' @return Returns data parameter

  saveRDS(data, filename)
  return(data)
}
```

Algorytm PRS (Pure Random Search)

Algorytm PRS (Pure Random Search - przeszukiwanie przypadkowe) oparty jest na losowaniu punktów z zadanej dziedziny. Każda współrzędna jest losowana z rozkładem jednostajnym. Po wylosowaniu zadanej liczby punktów, wyszukujemy minimum funkcji wyliczając jej wartość w wylosowanych punktach. Wynikiem algorytmu jest najmniejsza wartość funkcji.

Implementacja

```
prs_algorithm <- function(f, cost = 1000L) {
  #' Pure Random Search algorithm
  #'
  #' @param f Function to minimize (from spoof package)
  #' @param cost Number of calculations
  #' @return Returns function value

  if (!isSmoofFunction(f)) {
```

```
    stop("Function is not from smoof package.")
  }

  # Generate points
  domain_min <- getLowerBoxConstraints(f)
  domain_max <- getUpperBoxConstraints(f)

  points <- matrix(0, nrow = cost, ncol = length(domain_min))
  for (i in seq_along(domain_min)) {
    uniform_distribution_points <- runif(cost, domain_min[i], domain_max[i])
    for (j in seq_along(uniform_distribution_points)) {
      points[j, i] <- uniform_distribution_points[j]
    }
  }

  # Return minimum function value
  return(min(apply(points, 1, f)))
}
```

Generacja danych oraz obliczenie średnich wyników algorytmu

W celu przyspieszenia działania programu wszystkie wygenerowane dane zostały zapisane do plików o rozszerzeniu rds.

```
prs_rosenbrock_2D <- loadData("./data/prs_rosenbrock_2D.rds")
if (is.null(prs_rosenbrock_2D)) prs_rosenbrock_2D <-
saveData(replicate(number_of_calculations, prs_algorithm(rosenbrock_2D)),
"./data/prs_rosenbrock_2D.rds")
mean_prs_rosenbrock_2D <- mean(prs_rosenbrock_2D)

prs_rosenbrock_10D <- loadData("./data/prs_rosenbrock_10D.rds")
if (is.null(prs_rosenbrock_10D)) prs_rosenbrock_10D <-
saveData(replicate(number_of_calculations, prs_algorithm(rosenbrock_10D)),
"./data/prs_rosenbrock_10D.rds")
mean_prs_rosenbrock_10D <- mean(prs_rosenbrock_10D)

prs_rosenbrock_20D <- loadData("./data/prs_rosenbrock_20D.rds")
if (is.null(prs_rosenbrock_20D)) prs_rosenbrock_20D <-
saveData(replicate(number_of_calculations, prs_algorithm(rosenbrock_20D)),
"./data/prs_rosenbrock_20D.rds")
mean_prs_rosenbrock_20D <- mean(prs_rosenbrock_20D)

prs_rastrigin_2D <- loadData("./data/prs_rastrigin_2D.rds")
if (is.null(prs_rastrigin_2D)) prs_rastrigin_2D <-
saveData(replicate(number_of_calculations, prs_algorithm(rastrigin_2D)),
"./data/prs_rastrigin_2D.rds")
mean_prs_rastrigin_2D <- mean(prs_rastrigin_2D)

prs_rastrigin_10D <- loadData("./data/prs_rastrigin_10D.rds")
if (is.null(prs_rastrigin_10D)) prs_rastrigin_10D <-
```

```
saveData(replicate(number_of_calculations, prs_algorithm(rastrigin_10D)),
"./data/prs_rastrigin_10D.rds")
mean_prs_rastrigin_10D <- mean(prs_rastrigin_10D)

prs_rastrigin_20D <- loadData("./data/prs_rastrigin_20D.rds")
if (is.null(prs_rastrigin_20D)) prs_rastrigin_20D <-
saveData(replicate(number_of_calculations, prs_algorithm(rastrigin_20D)),
"./data/prs_rastrigin_20D.rds")
mean_prs_rastrigin_20D <- mean(prs_rastrigin_20D)
```

Algorytm GA (Genetic Algorithm)

Algorytm genetyczny oparty jest na zjawisku ewolucji biologicznej. Na początku losowana jest pewna populacja początkowa, która zostaje poddana selekcji (ocenie). Po wyborze najlepszych osobników do reprodukcji następuje krzyżowanie genotypów, a następnie przeprowadzana jest mutacja. Po odrzuceniu najsłabszych osobników utworzone jest nowe pokolenie. Algorytm działa, dopóki nie zostanie spełniony warunek zatrzymania np. liczba pokoleń. W projekcie została wykorzystana biblioteka `GA`, dostępna w `CRAN`, gdzie zaimplementowany jest algorytm. Przy korzystaniu z funkcji z tej biblioteki należy funkcję, która ma być minimalizowana, przekształcić do postaci `f(x) = -f(x)`, ponieważ domyślnie funkcja biblioteczna szuka maksimum.

Implementacja

```
ga_algorithm <- function(f, cost = 1000L) {
  #' Genetic Algorithm
  #'
  #' @param f Function to minimize (from spoof package)
  #' @param cost Number of calculations
  #' @return Returns function value

  if (!isSmoofFunction(f)) {
    stop("Function is not from smoof package.")
  }

  result <- ga(type = "real-valued",
               fitness = function(x) -f(x),
               lower = getLowerBoxConstraints(f),
               upper = getUpperBoxConstraints(f),
               maxiter = cost,
               run = cost)

  return(f(result@solution))
}
```

Generacja danych oraz obliczenie średnich wyników algorytmu

W celu przyspieszenia działania programu wszystkie wygenerowane dane zostały zapisane do plików o rozszerzeniu rds.

```
ga_rosenbrock_2D <- loadData("./data/ga_rosenbrock_2D.rds")
if (is.null(ga_rosenbrock_2D))ga_rosenbrock_2D <-
saveData(replicate(number_of_calculations, ga_algorithm(rosenbrock_2D)),
"./data/ga_rosenbrock_2D.rds")
mean_ga_rosenbrock_2D <- mean(ga_rosenbrock_2D)

ga_rosenbrock_10D <- loadData("./data/ga_rosenbrock_10D.rds")
if (is.null(ga_rosenbrock_10D)) ga_rosenbrock_10D <-
saveData(replicate(number_of_calculations, ga_algorithm(rosenbrock_10D)),
"./data/ga_rosenbrock_10D.rds")
mean_ga_rosenbrock_10D <- mean(ga_rosenbrock_10D)

ga_rosenbrock_20D <- loadData("./data/ga_rosenbrock_20D.rds")
if (is.null(ga_rosenbrock_20D)) ga_rosenbrock_20D <-
saveData(replicate(number_of_calculations, ga_algorithm(rosenbrock_20D)),
"./data/ga_rosenbrock_20D.rds")
mean_ga_rosenbrock_20D <- mean(ga_rosenbrock_20D)

ga_rastrigin_2D <- loadData("./data/ga_rastrigin_2D.rds")
if (is.null(ga_rastrigin_2D)) ga_rastrigin_2D <-
saveData(replicate(number_of_calculations, ga_algorithm(rastrigin_2D)),
"./data/ga_rastrigin_2D.rds")
mean_ga_rastrigin_2D <- mean(ga_rastrigin_2D)

ga_rastrigin_10D <- loadData("./data/ga_rastrigin_10D.rds")
if (is.null(ga_rastrigin_10D)) ga_rastrigin_10D <-
saveData(replicate(number_of_calculations, ga_algorithm(rastrigin_10D)),
"./data/ga_rastrigin_10D.rds")
mean_ga_rastrigin_10D <- mean(ga_rastrigin_10D)

ga_rastrigin_20D <- loadData("./data/ga_rastrigin_20D.rds")
if (is.null(ga_rastrigin_20D)) ga_rastrigin_20D <-
saveData(replicate(number_of_calculations, ga_algorithm(rastrigin_20D)),
"./data/ga_rastrigin_20D.rds")
mean_ga_rastrigin_20D <- mean(ga_rastrigin_20D)
```

Porównanie średnich wyników algorytmów oraz ich analiza pod kątem położenia i rozproszenia rozkładu

Dla każdej funkcji (oraz jej wymiaru) został przedstawiony histogram oraz wykres pudełkowy wyników algorytmów, w celu analizy ich położenia i rozproszenia. Dodatkowo została zdefiniowana funkcja pomocnicza `draw_plots`.

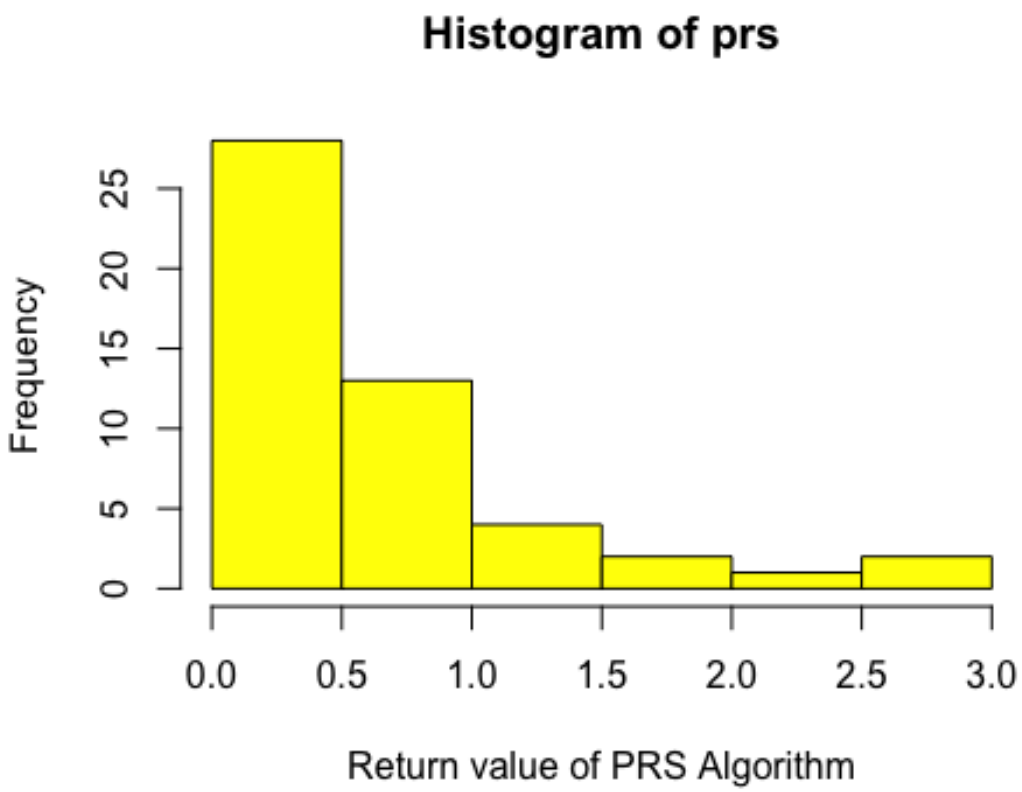
```
draw_plots <- function(prs, ga) {
  #' Draw algorithm results plots
  #'
  #' @param prs PRS algorithm results
  #' @param ga GA algorithm results
  #' @return Returns nothing, just draws plots
```

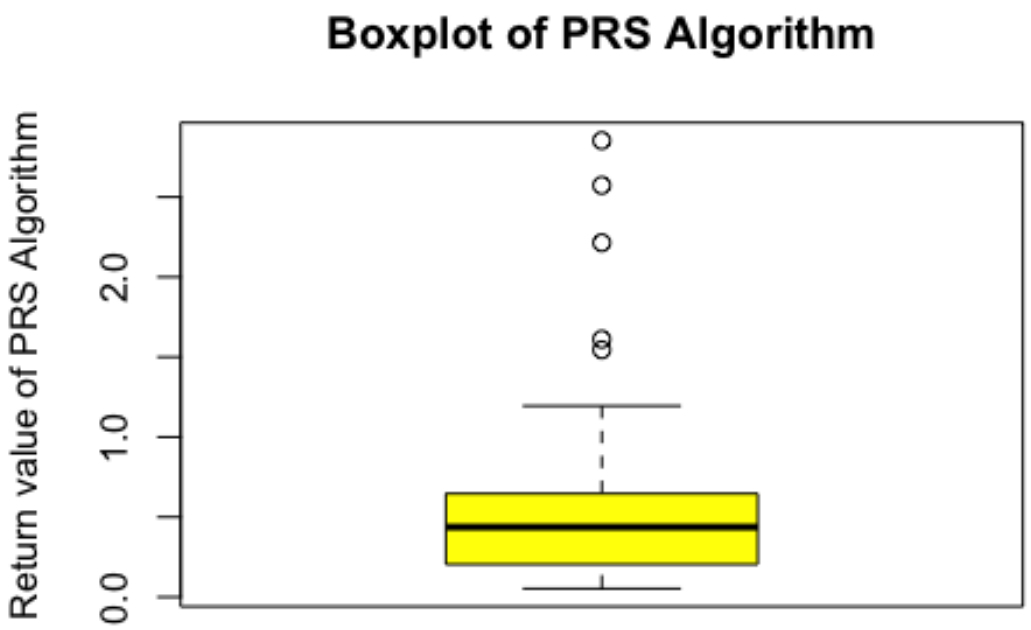
```
  # PRS Algorithm
  hist(prs, col = "yellow", xlab = "Return value of PRS Algorithm")
  boxplot(prs, col = 'yellow', ylab = "Return value of PRS Algorithm", main =
"Boxplot of PRS Algorithm")

  # GA Algorithm
  hist(ga, col = "green", xlab = "Return value of GA Algorithm")
  boxplot(ga, col = 'green', ylab = "Return value of GA Algorithm", main =
"Boxplot of GA Algorithm")

  # Comparison
  data <- data.frame(prs, ga)

  func <- c(rep("PRS", length(prs)), rep("GA", length(ga)))
  value <- c(prs, ga)
  data <- data.frame(func, value)
  boxplot(data$value ~ data$func, col = terrain.colors(4))
}
```

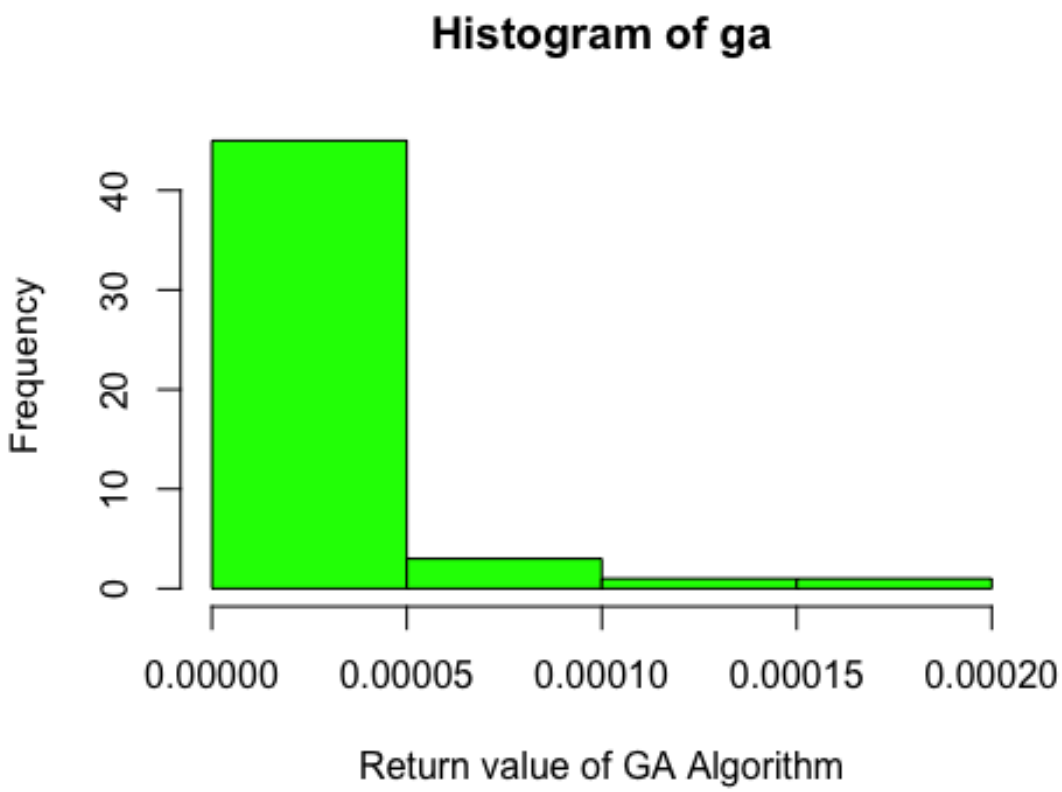
Funkcja Rosenbrocka

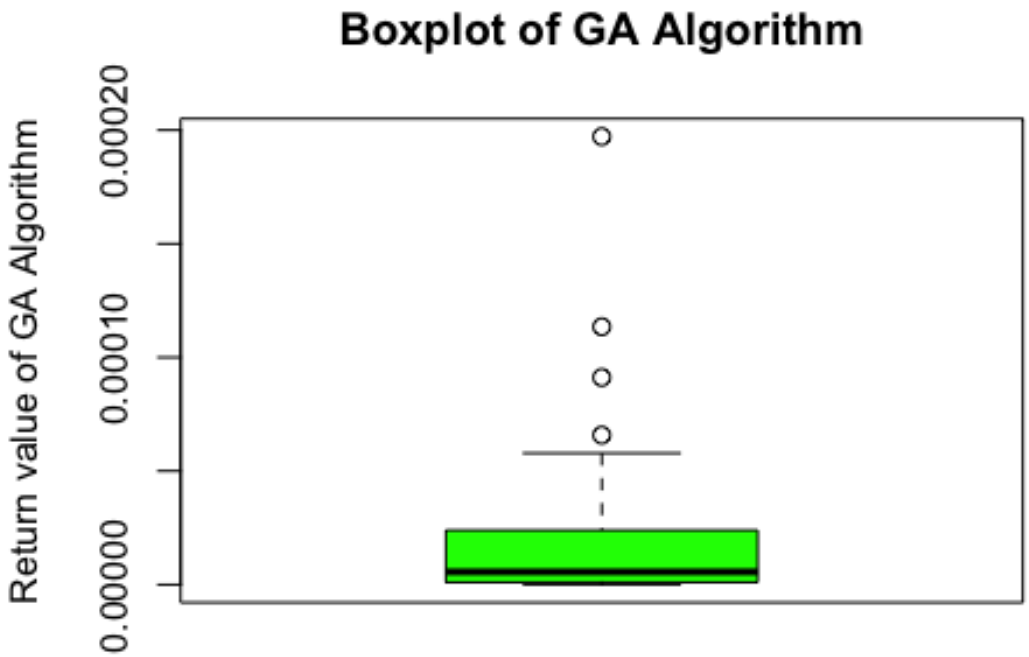
Wymiar 2D

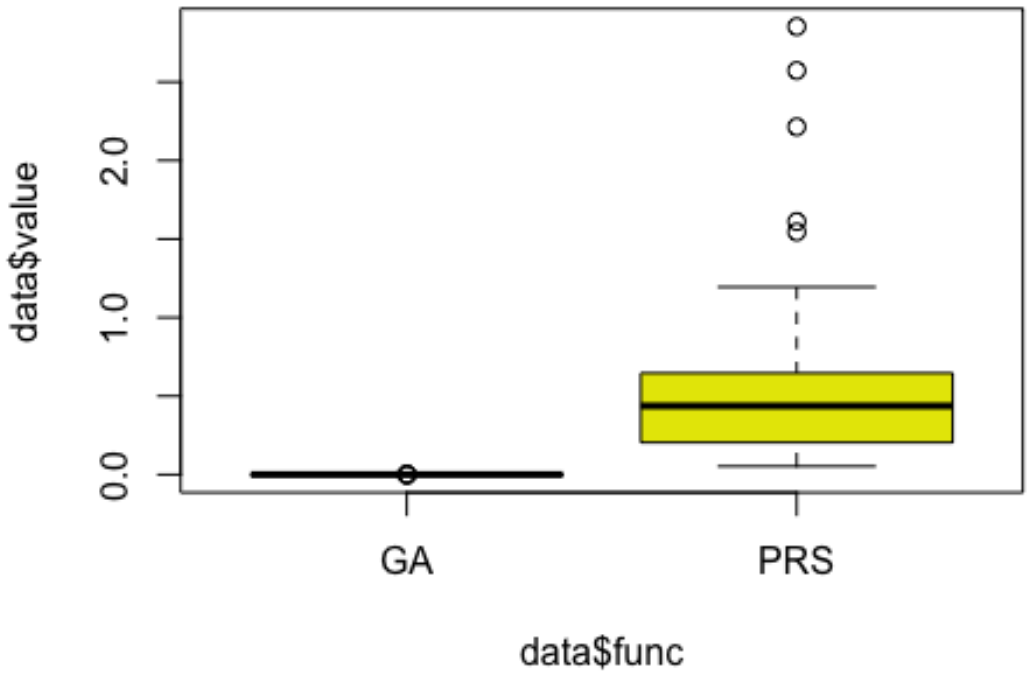
```
draw_plots(prs_rosenbrock_2D, ga_rosenbrock_2D)
```


Histogram of prs
Frequency
0
5
10
15
20
25
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Return value of PRS Algorithm


Boxplot of PRS Algorithm
Return value of PRS Algorithm
0.0
1.0
2.0


Histogram of ga
Frequency
0
10
20
30
40
0.00000
0.00005
0.00010
0.00015
0.00020
Return value of GA Algorithm

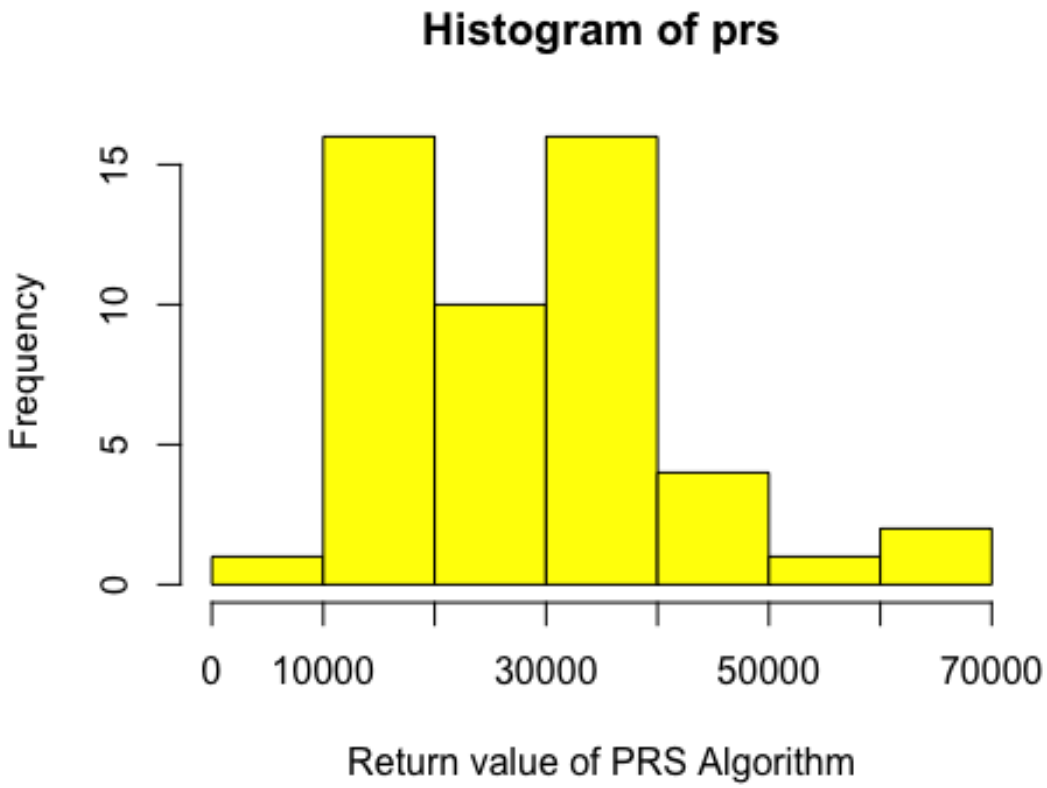

Boxplot of GA Algorithm
Return value of GA Algorithm
0.00000
0.00010
0.00020

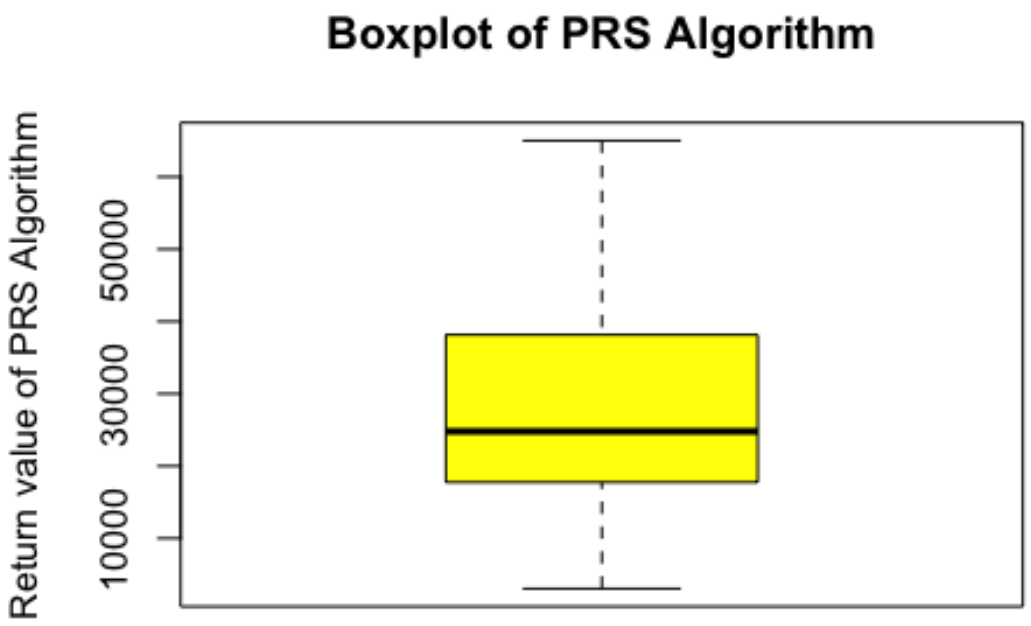


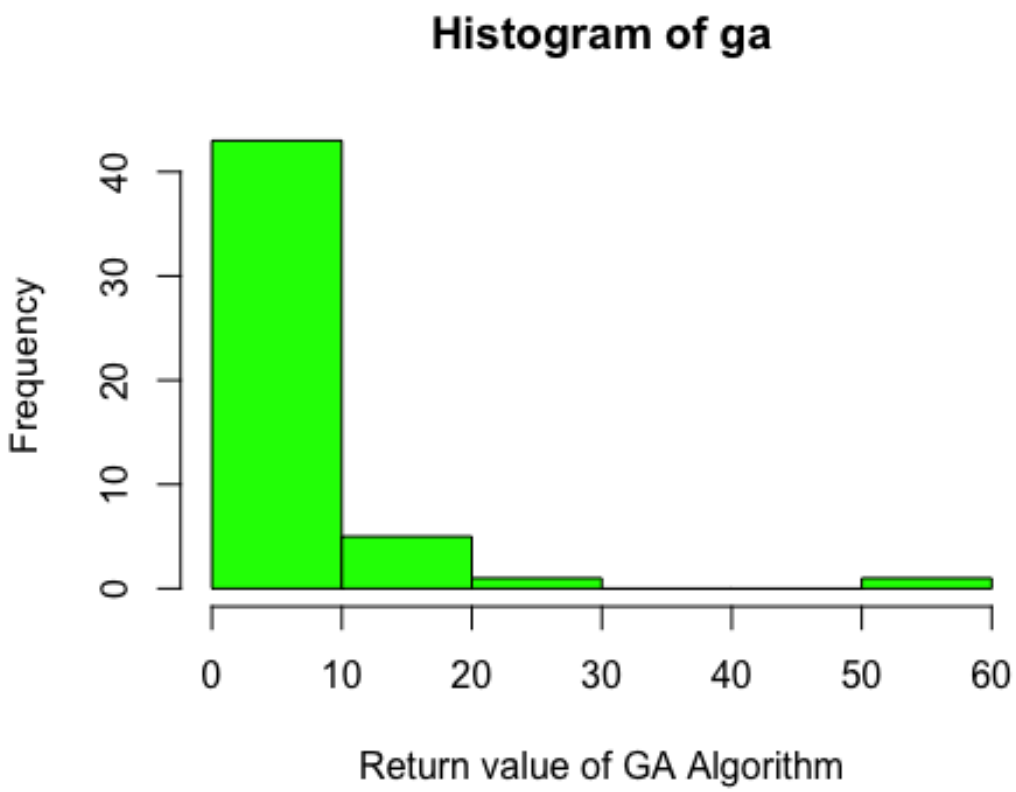
Powyższe wykresy przedstawiają rozkład wyników, jakie uzyskały algorytmy PRS oraz GA dla 50 powtórzeń. Z histogramów wywnioskować można, że znaczna część wyników, jakie zwróciły algorytmy skupia się blisko wartości 0. Różnicę pomiędzy algorytmami zaobserwować można zwracając uwagę na skalę tych wykresów (oś OX). Analizując wykresy pudełkowe wyników algorytmów zauważyć można występowanie wielu tzw. outlierów (wartości odstających), znajdujących się nad "górnym wąsem". Ta sytuacja powtarza się dla dwóch algorytmów. Szczególnie dobrze widać to na wykresie pudełkowym - pudełko dla Algorytmu GA sprowadza się do prostej.

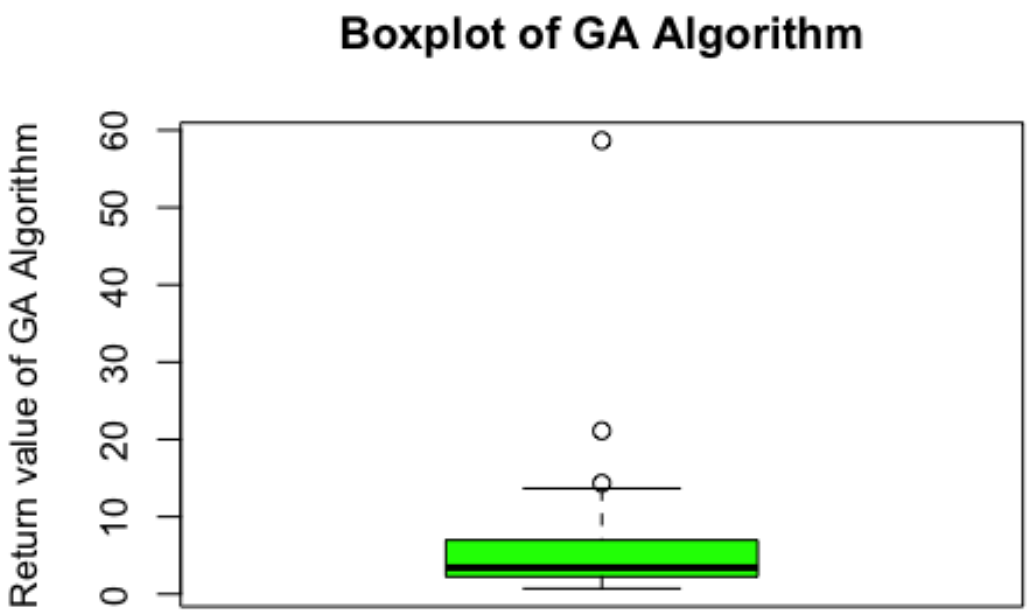
Wymiar 10D

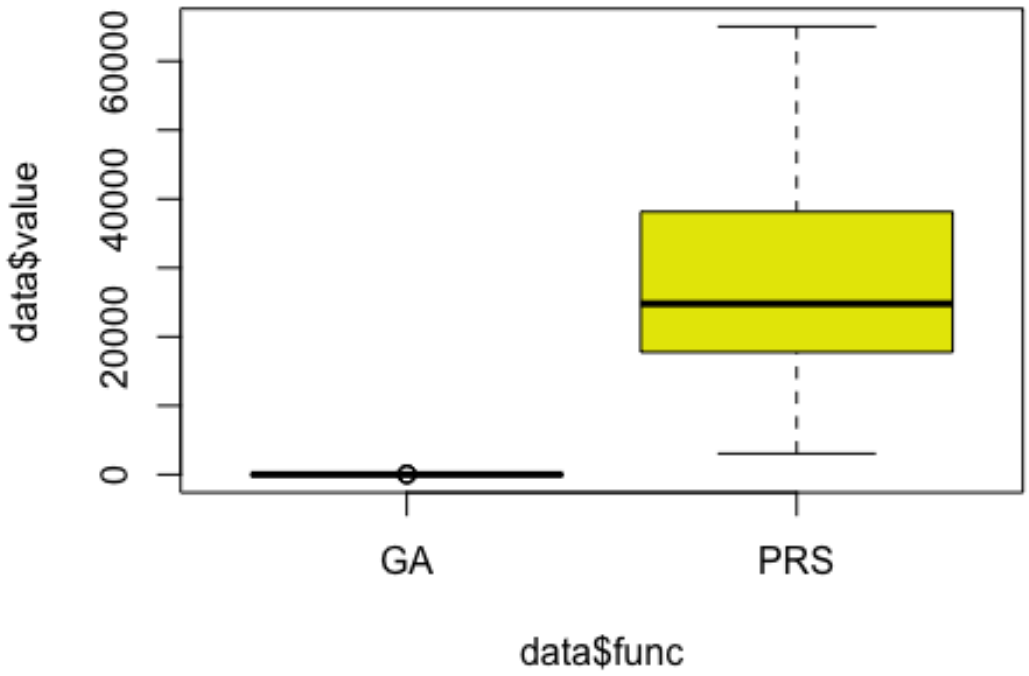
```
draw_plots(prs_rosenbrock_10D, ga_rosenbrock_10D)
```


Histogram of prs
Frequency
0
5
10
15
0
10000
30000
50000
70000
Return value of PRS Algorithm


Boxplot of PRS Algorithm
Return value of PRS Algorithm
10000
30000
50000


Histogram of ga
Frequency
0
10
20
30
40
0
10
20
30
40
50
60
Return value of GA Algorithm

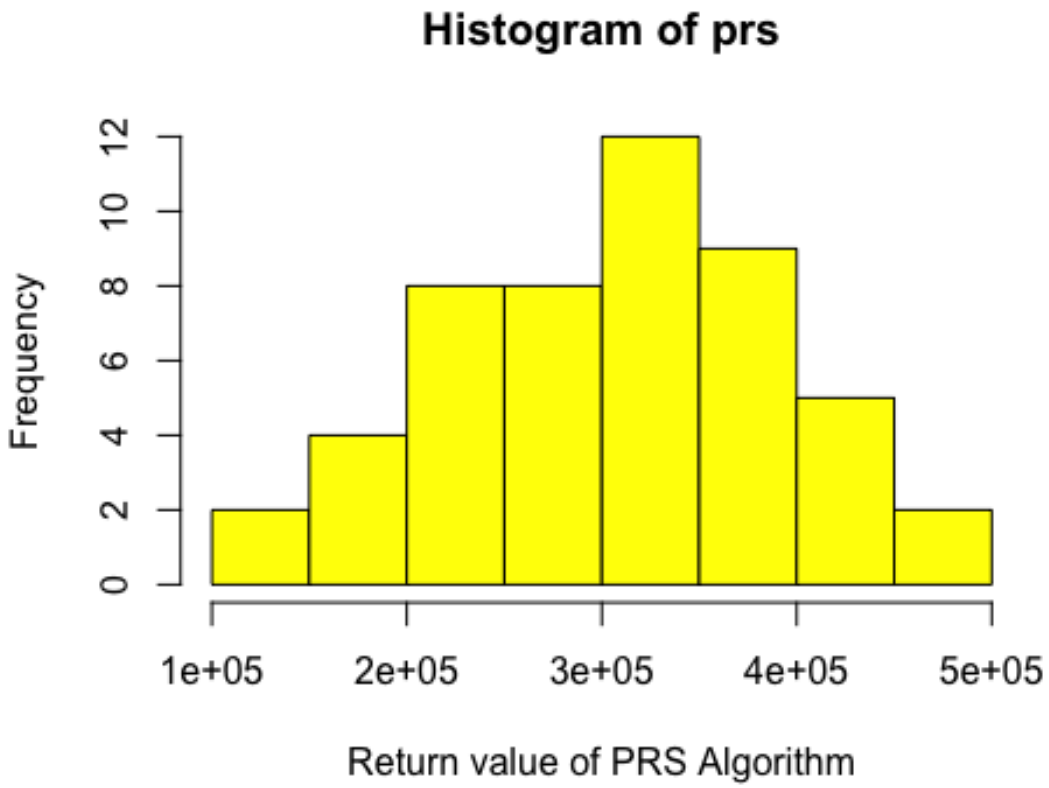

Boxplot of GA Algorithm
Return value of GA Algorithm
0
10
20
30
40
50
60

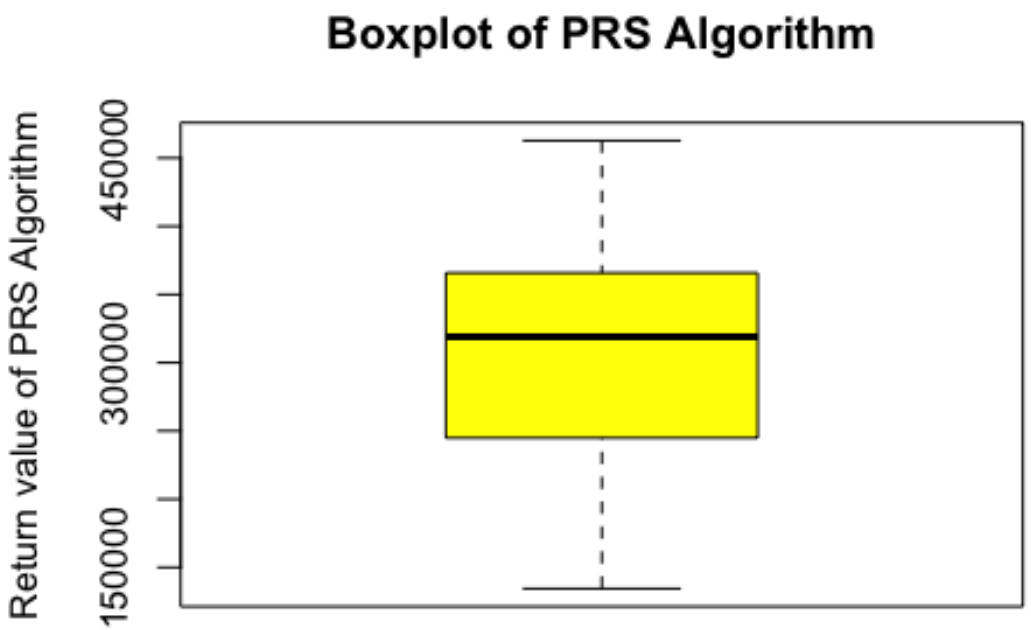


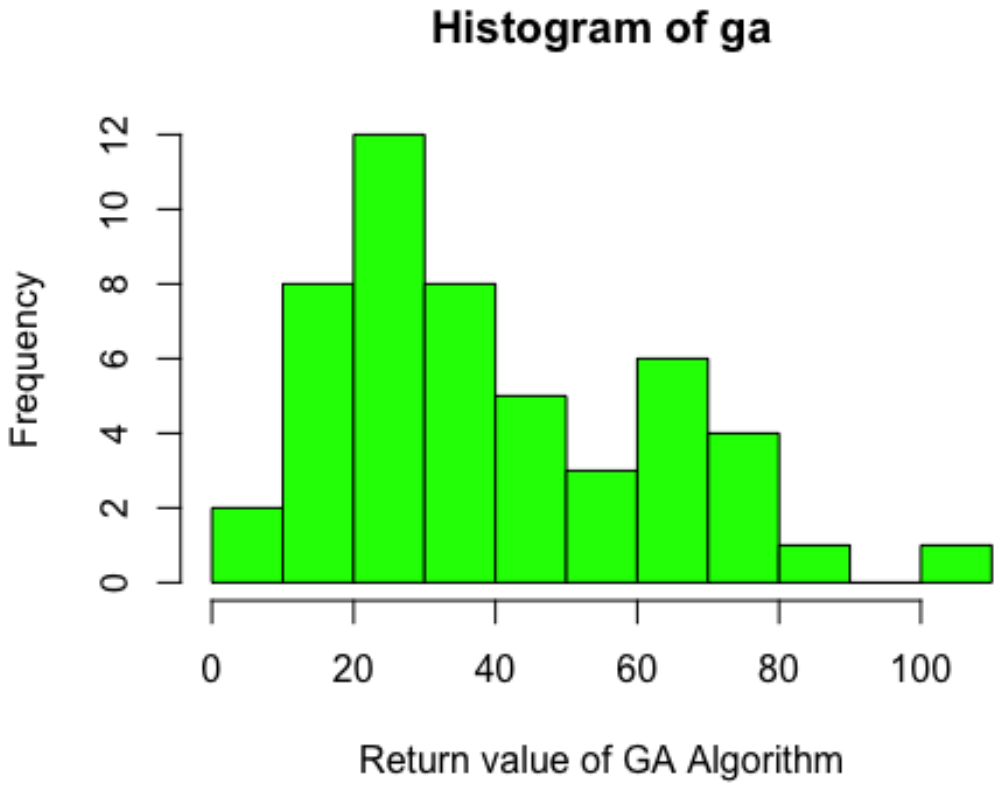
W przypadku funkcji 10 - wymiarowych, histogramy wyników działania algorytmów zaczynają się od siebie różnić (w porównaniu z 2 - wymiarowymi funkcjami). Przede wszystkim wykres wyników algorytmu PRS "przesunął się" w dodatnią stronę osi OX. Wyniki algorytmu nie są skupione wokół zera, ale przedziału [10000,40000]. Również wykres pudełkowy uległ zmianie. W tym przykładzie nie zaobserwowano żadnych tzw. outlierów (wartości odstających), za to "wąsy" wykresu stały się znacznie większe i symetryczne. Wykresy wyników algorytmu GA zachowały swój kształt (z poprzedniego przykładu).

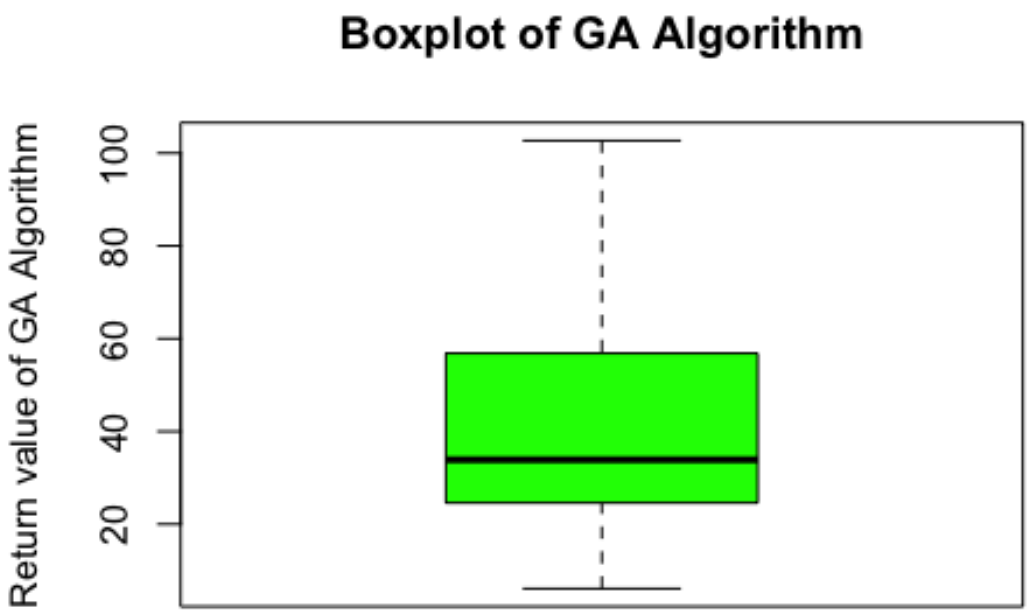
Wymiar 20D

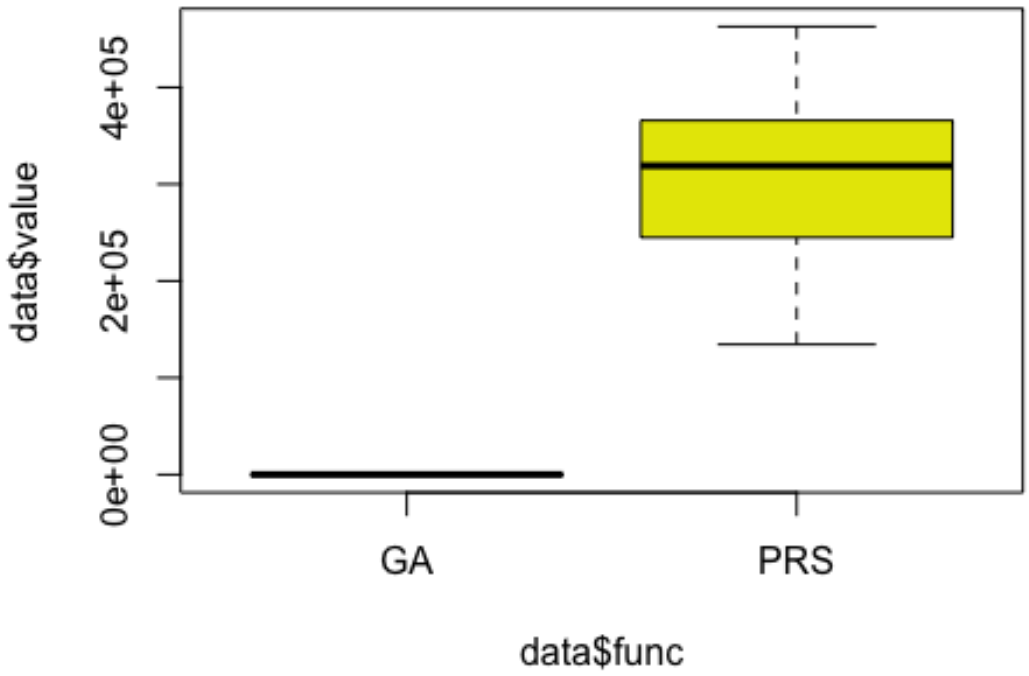
```
draw_plots(prs_rosenbrock_20D, ga_rosenbrock_20D)
```


Histogram of prs
Frequency
0
2
4
6
8
10
12
1e+05
2e+05
3e+05
4e+05
5e+05
Return value of PRS Algorithm


Boxplot of PRS Algorithm
Return value of PRS Algorithm
150000
300000
450000


Histogram of ga
Frequency
0
2
4
6
8
10
12
0
20
40
60
80
100
Return value of GA Algorithm


Boxplot of GA Algorithm
Return value of GA Algorithm
20
40
60
80
100

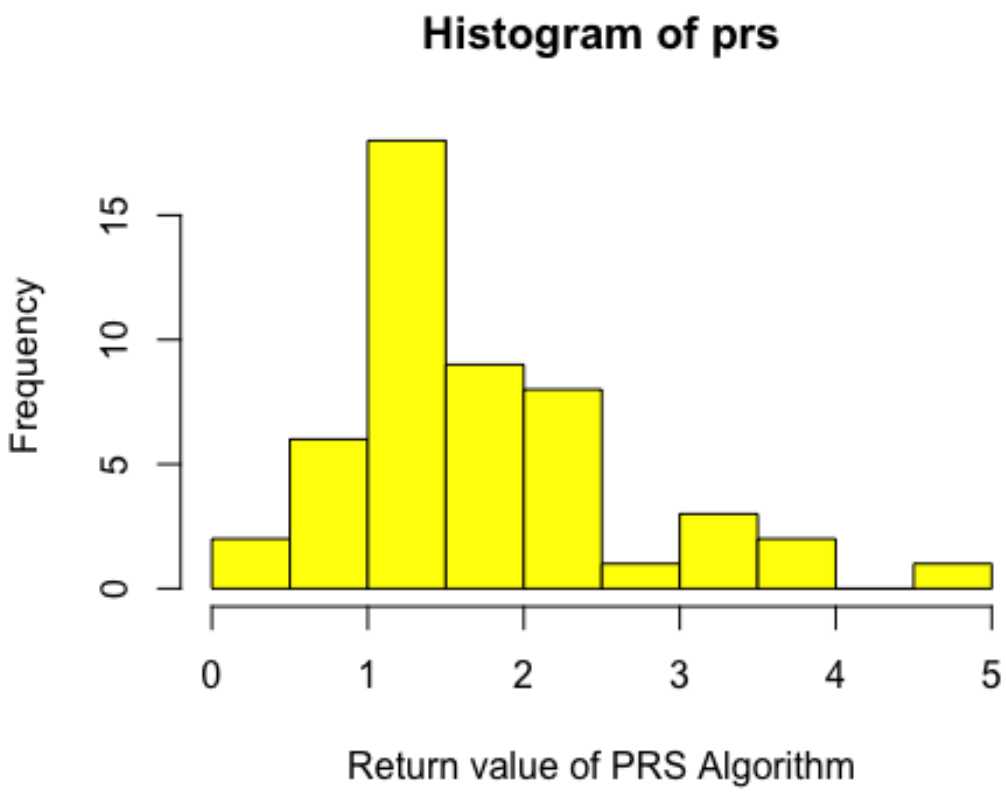


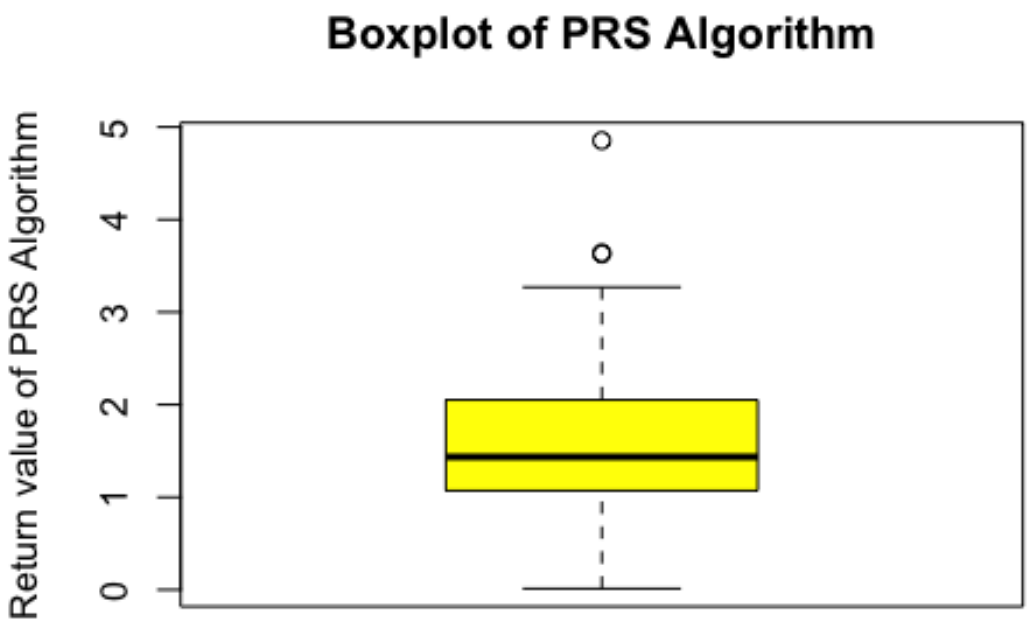
Powyższe wykresy, przedstawiające wyniki działania algorytmów, różnią się w zależności od wyboru metody stochastycznej optymalizacji. Tak jak poprzednio, rozkład wyników algorytmu PRS nie skupia się blisko zera, ale wokół wartości znacznie większych tj. $[2 \times 10^5, 4 \times 10^5]$. Wykres pudełkowy zachował swój kształt z poprzedniego przykładu. W tej sytuacji również wykres drugiego algorytmu uległ przesunięciu. Jeszcze w poprzednim przykładzie większość wartości znajdowała się blisko zera. Teraz skupiają się one wokół przedziału [10,50]. Wykres pudełkowy nadal obrazuję podobną sytuację jedynie z przeskalowaną osią OY.

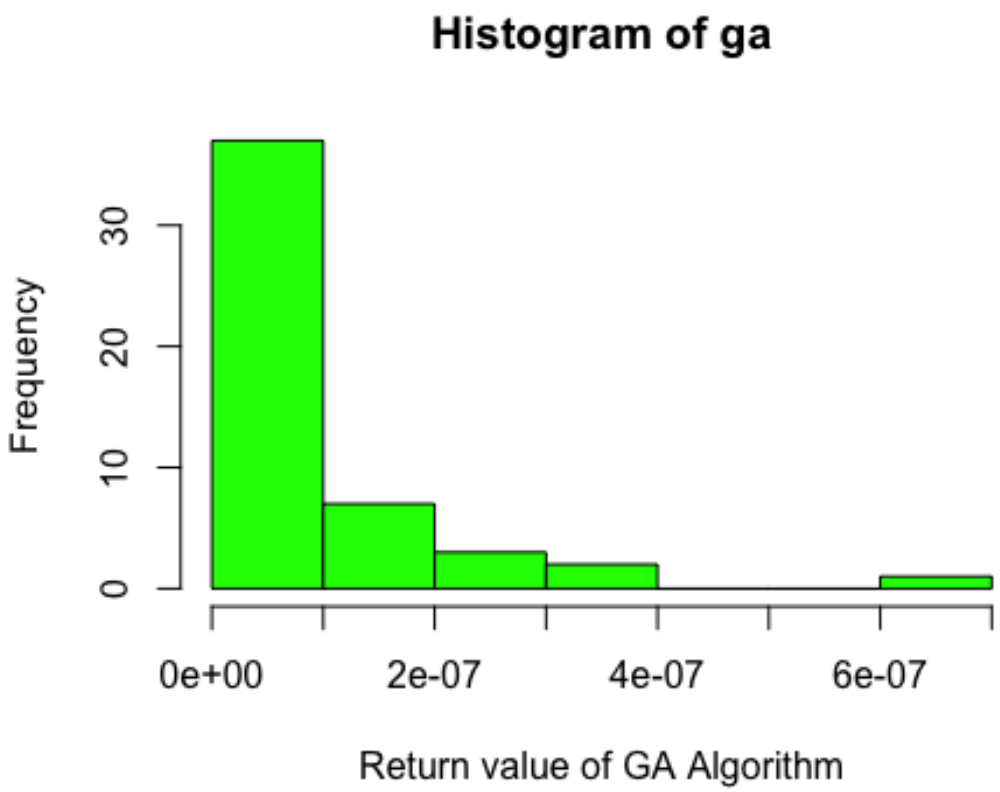
Funkcja Rastrigina

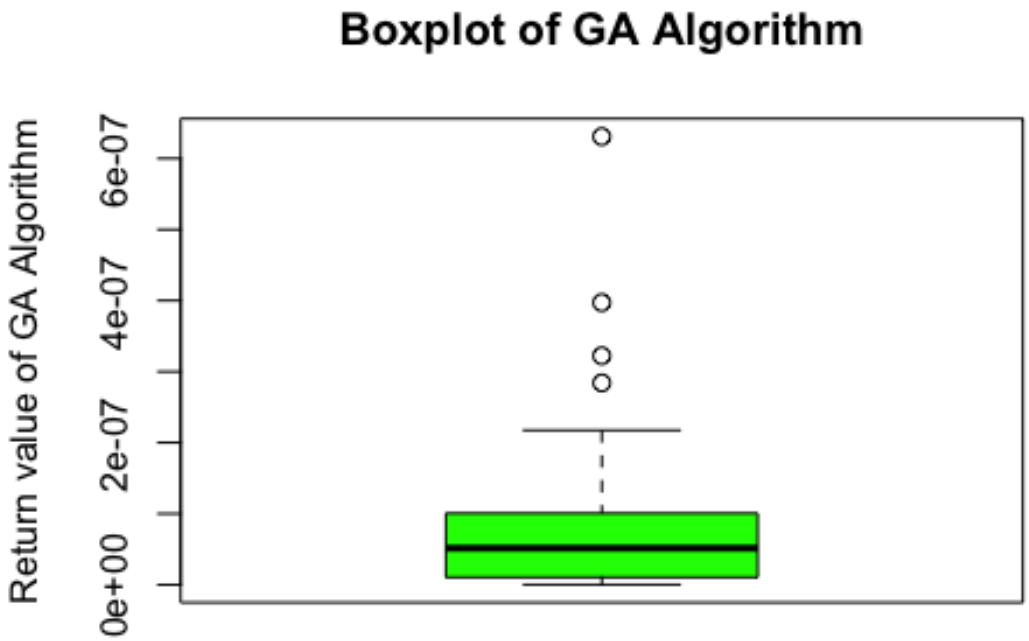
Wymiar 2D

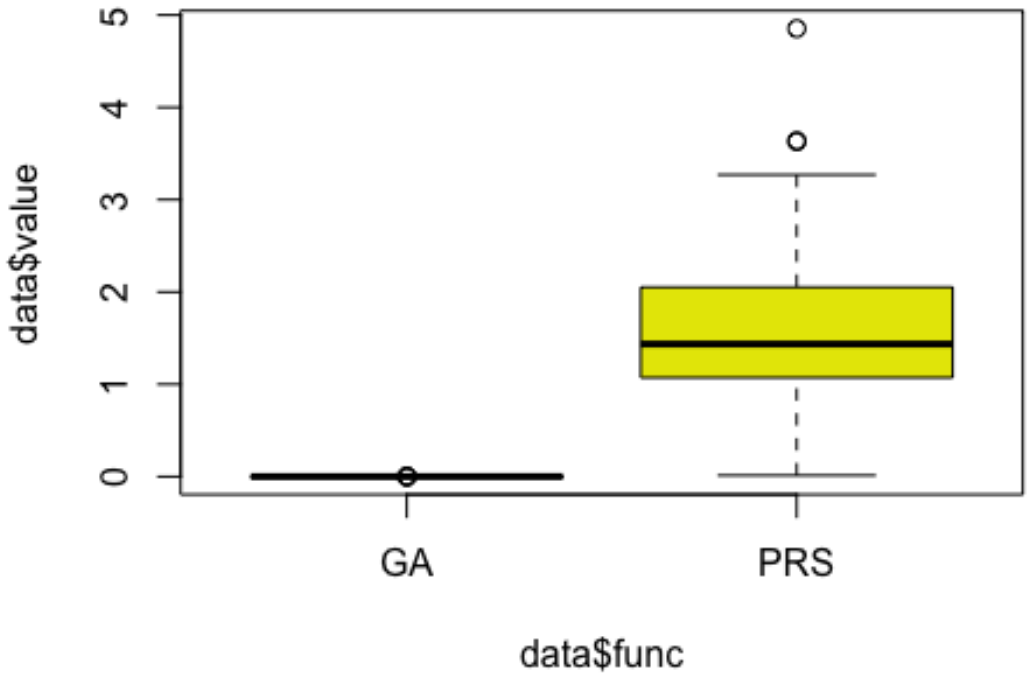
```
draw_plots(prs_rastrigin_2D, ga_rastrigin_2D)
```


Histogram of prs
Frequency
0
5
10
15
0
1
2
3
4
5
Return value of PRS Algorithm


Boxplot of PRS Algorithm
Return value of PRS Algorithm
0
1
2
3
4
5


Histogram of ga
Frequency
0
10
20
30
0e+00
2e-07
4e-07
6e-07
Return value of GA Algorithm

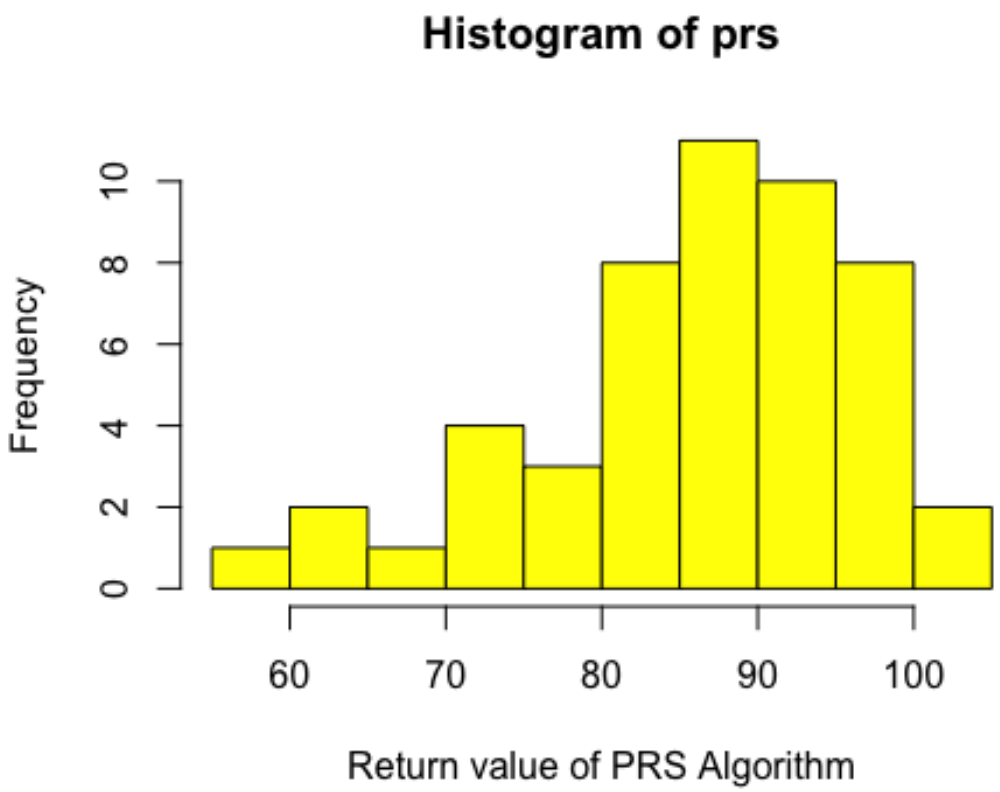

Boxplot of GA Algorithm
Return value of GA Algorithm
0e+00
2e-07
4e-07
6e-07

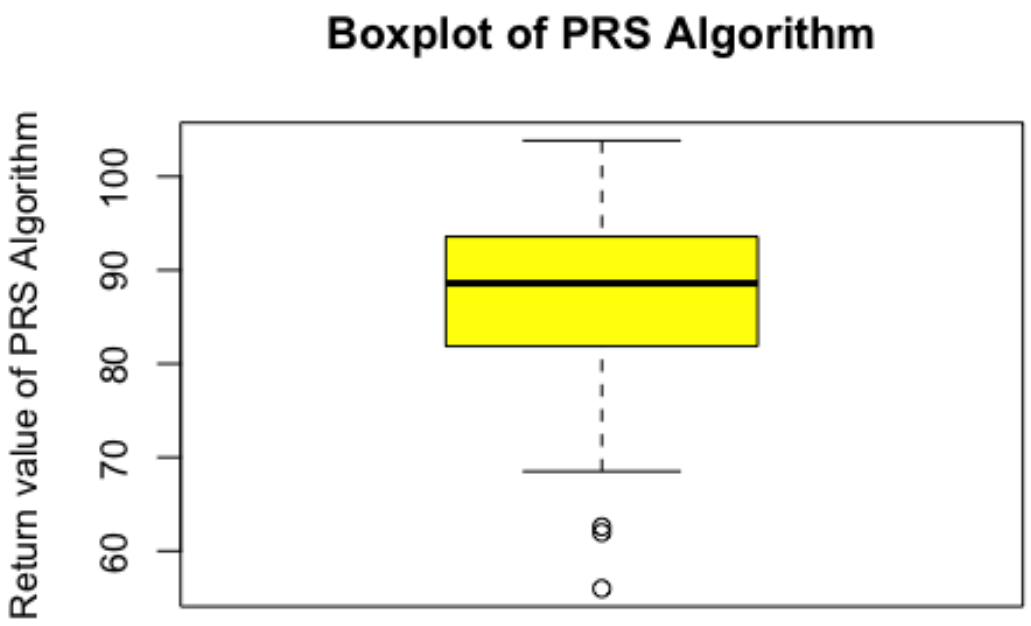


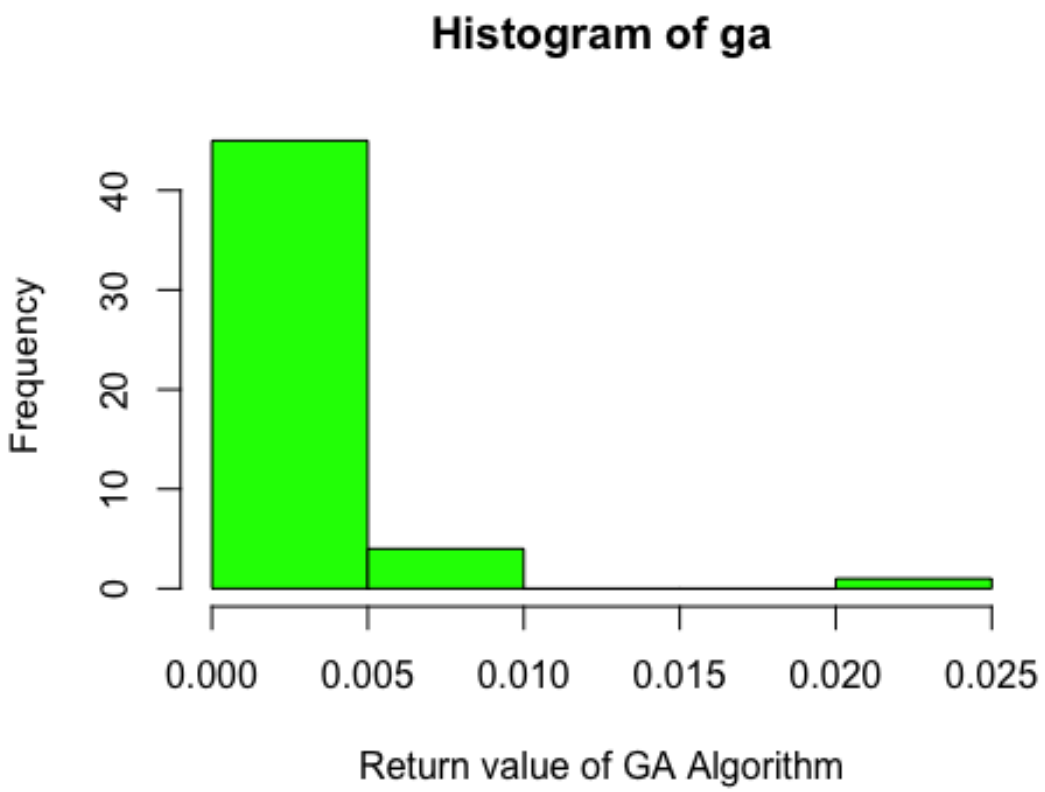
Powyższe wykresy przedstawiające wyniki działania algorytmów PRS i GA, różnią się w zależności od wybranej metody optymalizacji stochastycznej. Wartości na wykresie dla algorytmu PRS skupiają się wokół wartości 1, podczas gdy wartości na histogramie algorytmu GA znajdują się bardzo blisko zera. Szczególnie dobrze tę zależność pokazują wykresu pudełkowe, w których "pudełko" algorytmu GA redukuje się do prostej. Zarówno w wykresie pudełkowym dla algorytmu GA, jak i algorytmu PRS zabserwować można górnych outlierów. Różnicą jest położenie "wąsów", które dla wyników metody PRS są symetryczne, a dla metody GA, górny "wąs" jest znacznie dłuższy.

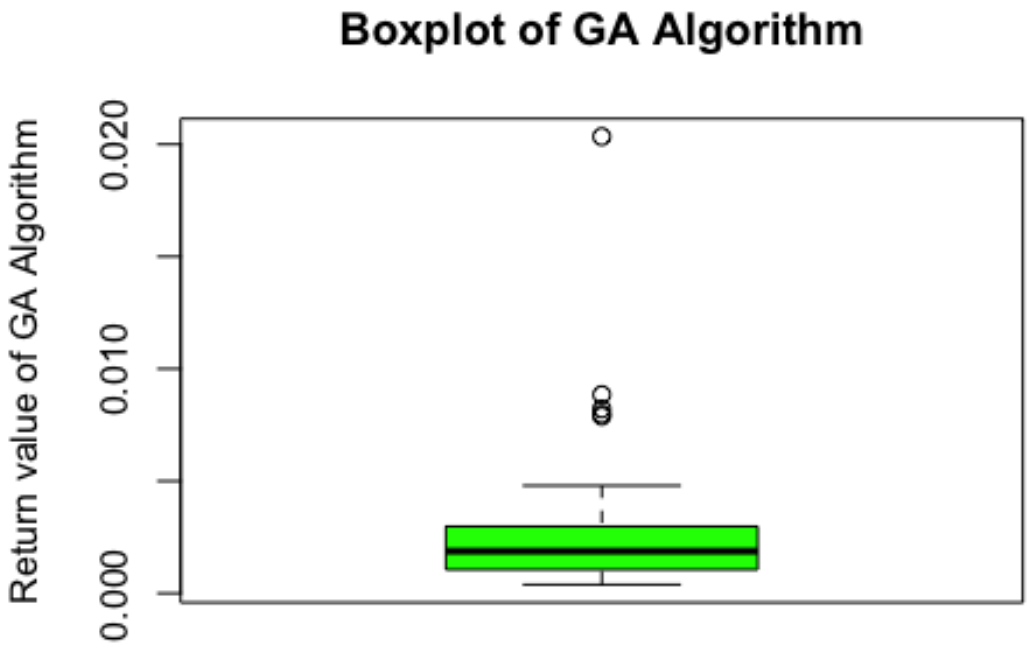
Wymiar 10D

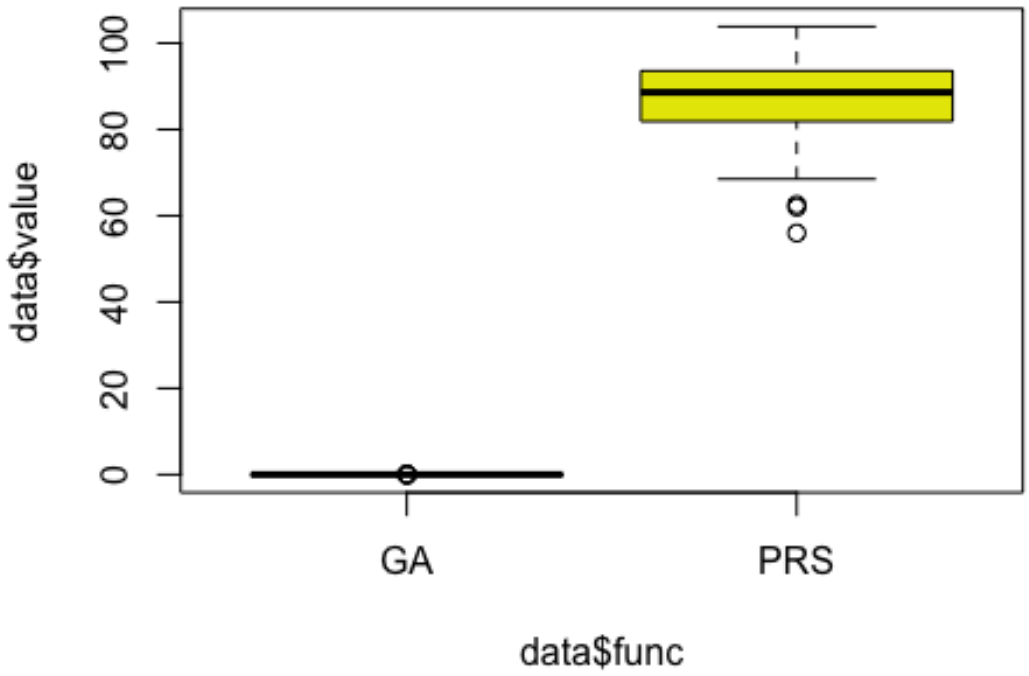
```
draw_plots(prs_rastrigin_10D, ga_rastrigin_10D)
```


Histogram of prs
Frequency
0
2
4
6
8
10
60
70
80
90
100
Return value of PRS Algorithm


Boxplot of PRS Algorithm
Return value of PRS Algorithm
60
70
80
90
100


Histogram of ga
Frequency
0
10
20
30
40
0.000
0.005
0.010
0.015
0.020
0.025
Return value of GA Algorithm

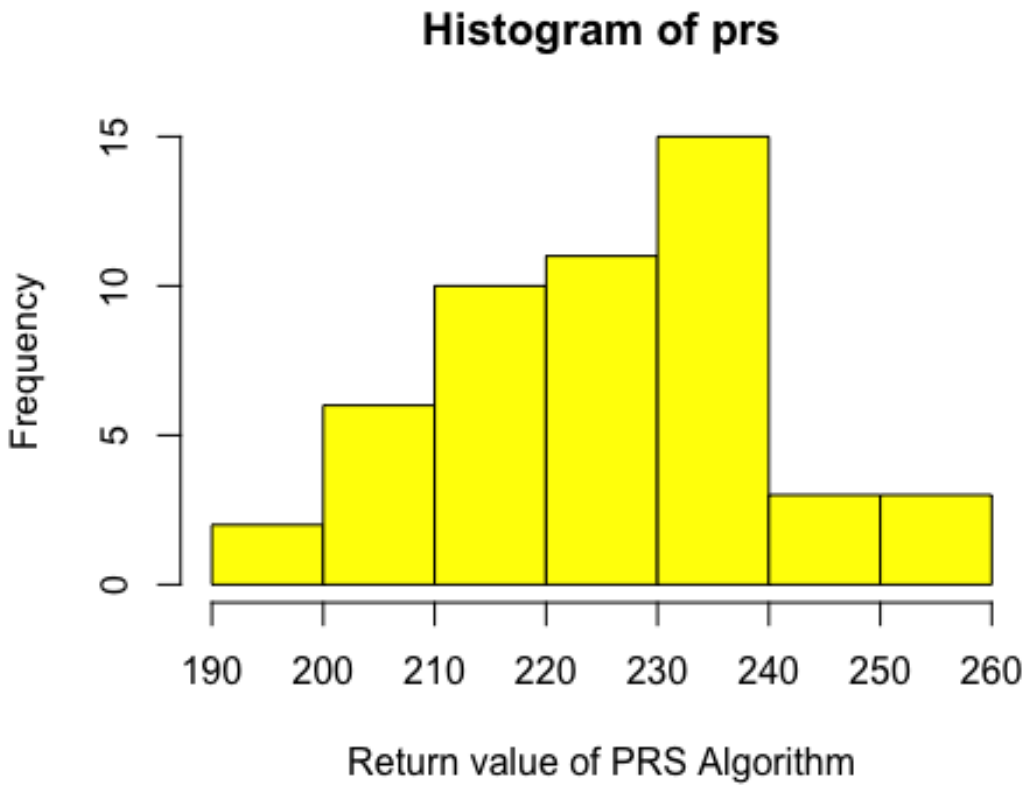

Boxplot of GA Algorithm
Return value of GA Algorithm
0.000
0.010
0.020

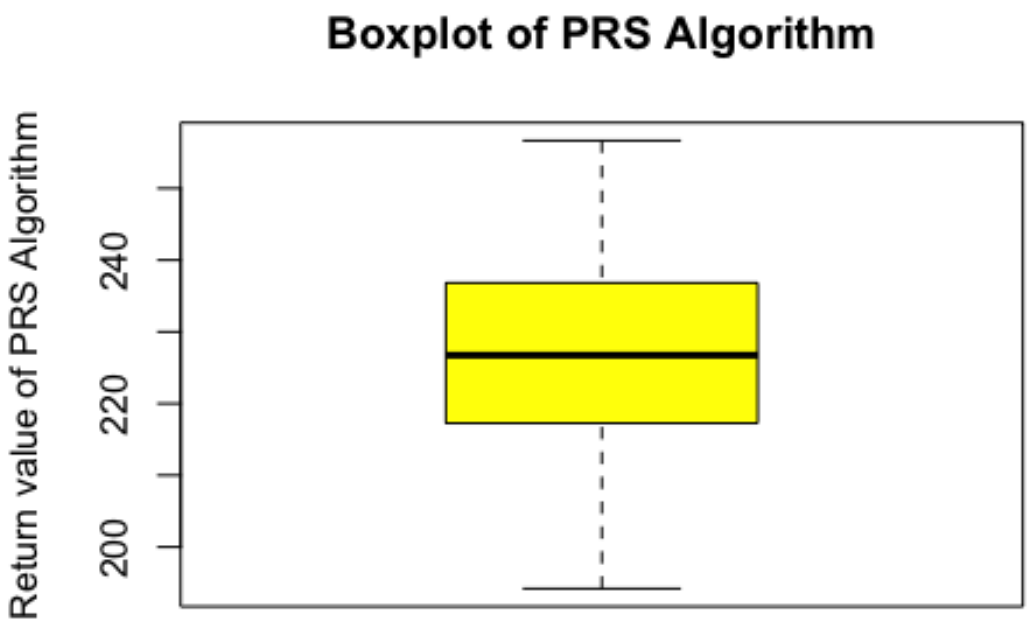


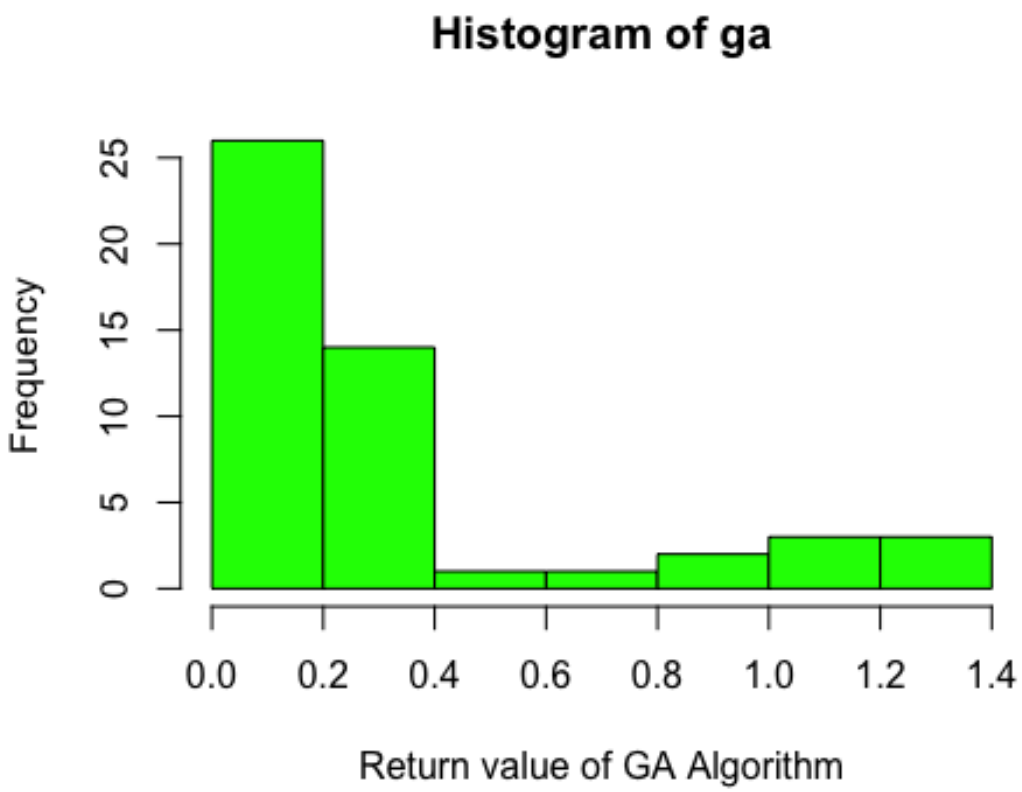
Powyższe wykresy różnią się w zależności od wyboru algorytmu optymalizacji stochastycznej. Dla PRS zaobserwować można znaczną przesunięcie danych w dodatnią stronę osi OX. Najwięcej ich znajduje się w przedziale [80,100]. Dla porównania dane w histogramie algorytmu GA nadal skupiają blisko 0. Wykres pudełkowy dla algorytmu PRS różni się od tych wygenerowanych we wcześniejszych przykładach. Po raz pierwszy bowiem zaobserwowano występowanie outlierów dolnych (wcześniej było to tylko outliery górne). Sytuacja ta nie powtarza się dla algorytmu GA, którego wykres pudełkowy przypomina te z wcześniejszych przykładów).

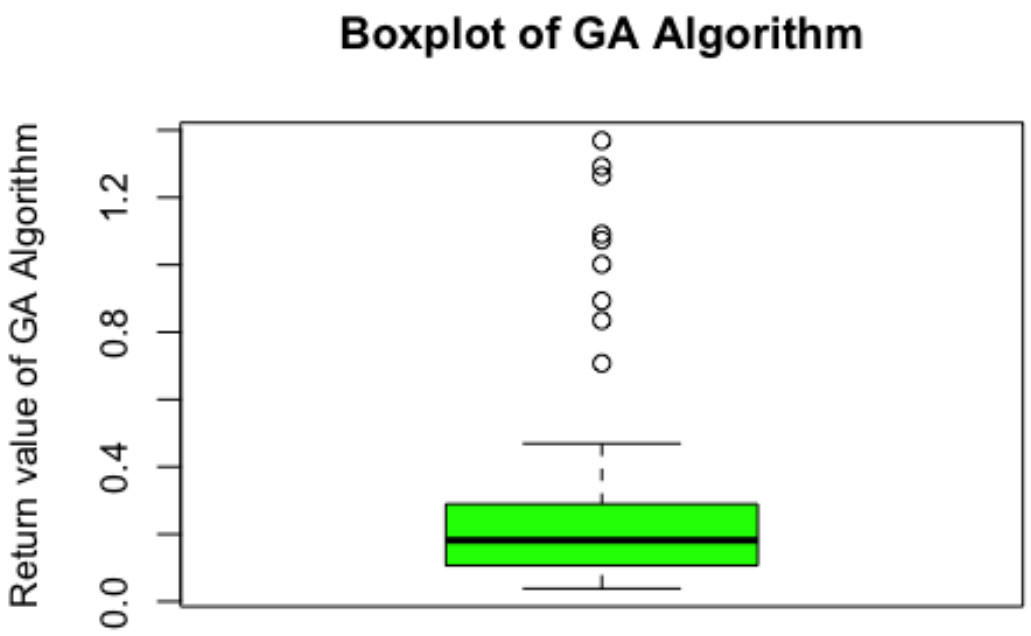
Wymiar 20D

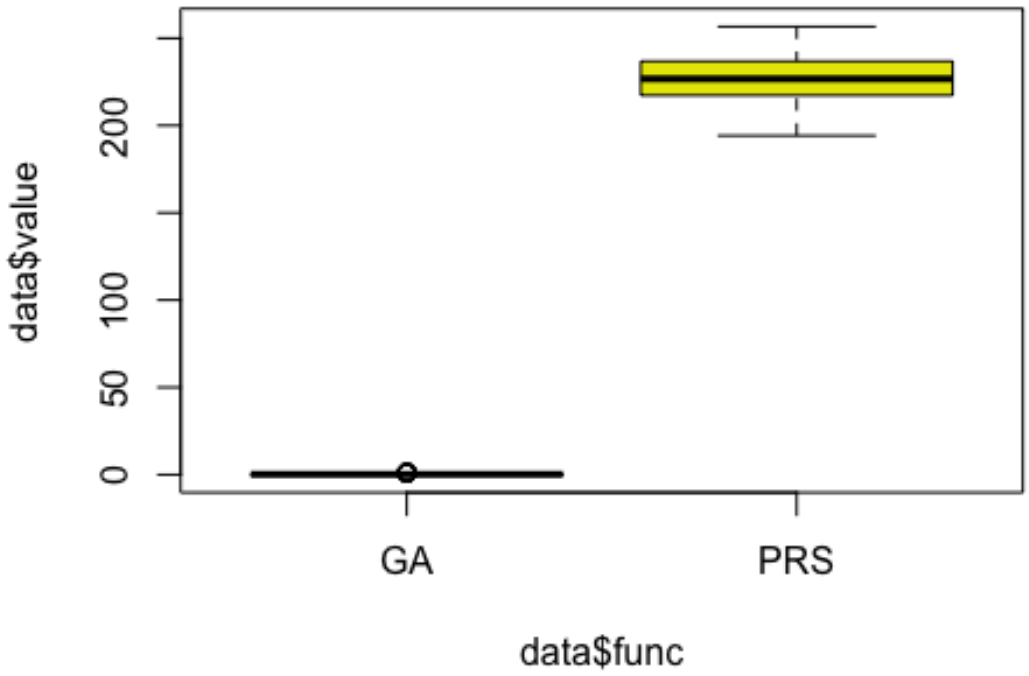
```
draw_plots(prs_rastrigin_20D, ga_rastrigin_20D)
```


Histogram of prs
Frequency
0
5
10
15
190
200
210
220
230
240
250
260
Return value of PRS Algorithm


Boxplot of PRS Algorithm
Return value of PRS Algorithm
200
220
240


Histogram of ga
Frequency
0
5
10
15
20
25
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
Return value of GA Algorithm


Boxplot of GA Algorithm
Return value of GA Algorithm
0.0
0.4
0.8
1.2



Powyższe wykresy różnią się w zależności od wyboru metody optymalizacji stochastycznej. Wykres algorytmu PRS jest przesunięty w prawą stronę (w dodatnią stronę osi OX). Wartości na wykresie skupiają się wokół przedziału [210,240]. Dodatkowo z analizy wykresu pudełkowego zaobserwować można, iż dane równomiernie rozkładają się na wynikowym przedziale. Nie zaobserwowano również żadnych outlierów. W przypadku algorytmu GA dane skupione są bliżej zera. Z analizy wykresu pudełkowego zaobserwowano występowanie wielu outlierów, znajdujących się powyżej górnego "wąsa". Porównując dwa algorytmy na jednym wykresie pudełkowym, zabserwować można znaczne różnice pomiędzy wynikami. Przedział, na którym skupiają się wyniki algorytmu GA jest tak mały w porównaniu do algorytmu PRS, że redukuje się do prostej.

Analiza istotności statystycznej różnicy między wynikami algorytmów

W tej sekcji została przeprowadzona analiza istotności statystycznej różnicy między wynikami algorytmów. Ze względu na liczbę powtórzeń (w naszym przypadku wykonano algorytm 50 razy) założono, że rozkład średniej może być sensownie przybliżony przez rozkład normalny $N(m, \sigma)$. W związku z powyższą obserwacją, został wykorzystany test t-Studenta dla dwóch niezależnych próbek, który miał na celu ocenienie hipotezy $H_0: \overline{X_1} = \overline{X_2}$ (średnie wyniki algorytmów są równe) wobec hipotezy alternatywnej $H_1: \overline{X_1} \neq \overline{X_2}$ dla poziomu istotności $\alpha = 0.05$.

W tym celu wykorzystano funkcję `run_t.test` opartej na funkcji `t.test`, która jako argumenty przyjmuje ona dwie próbki (wyniki algorytmu PRS i GA), etykiety zestawów danych oraz parametr `mu`, który jest równy różnicy między średnimi (w naszym przypadku równy 0).

```
run_t.test <- function (data_x, data_y, mu=0, alpha=0.05, label_1 = "x",
label_2 = "y") {
  #' Runs t-test on two data sets
  #'
  #' @param data_x First data set
  #' @param data_y Second data set
  #' @param mu Mean value
  #' @param alpha Significance level
  #' @param label_1 Label for first data set
  #' @param label_2 Label for second data set
  #' @return Returns t-test result

  test <- t.test(data_x, data_y, mu=mu, alternative="two.sided",
conf.level=1-alpha)

  p_value <- test$p.value
  if (p_value <= alpha) {
    hypothesis_analysis_result <- "Conclusion: hypothesis is false, because
p_value <= alpha"
  }else{
    hypothesis_analysis_result <- "Conclusion: not enough information if
hypothesis is false, because p_value > alpha"
  }

  cat(sep = "",
      "t = ", test$statistic, '\n',
      "df = ", test$parameter, '\n',
      "p-value = ", test$p.value, '\n',
      "hypothesis: true difference in means is equal to ", mu, '\n',
      "alternative hypothesis: true difference in means is not equal to ",
mu, '\n',
      attr(test$conf.int, "conf.level")*100, " percent confidence interval:
[", test$conf.int[1],", ", test$conf.int[2], "]\n",
      "sample estimates:", '\n',
      "   mean of ", label_1, " = ", test$estimate[1], '\n',
      "   mean of ", label_2, " = ", test$estimate[2], '\n',
      '\n',
      hypothesis_analysis_result
  )
}
```

Funkcja Rosenbrocka

Wymiar 2D

```
run_t.test(prs_rosenbrock_2D, ga_rosenbrock_2D, label_1 = "PRS algorithm
result", label_2 = "GA algorithm result")
```

```
## t = 6.836186
## df = 49
## p-value = 1.190042e-08
## hypothesis: true difference in means is equal to 0
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval: [0.4259411, 0.7806251]
## sample estimates:
##    mean of PRS algorithm result = 0.6033034
##    mean of GA algorithm result = 2.026331e-05
##
## Conclusion: hypothesis is false, because p_value <= alpha
```

Wymiar 10D

```
run_t.test(prs_rosenbrock_10D, ga_rosenbrock_10D, label_1 = "PRS algorithm
result", label_2 = "GA algorithm result")
```

```
## t = 15.47865
## df = 49.00004
## p-value = 1.683683e-20
## hypothesis: true difference in means is equal to 0
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval: [24843.09, 32256.23]
## sample estimates:
##    mean of PRS algorithm result = 28555.61
##    mean of GA algorithm result = 5.952174
##
## Conclusion: hypothesis is false, because p_value <= alpha
```

Wymiar 20D

```
run_t.test(prs_rosenbrock_20D, ga_rosenbrock_20D, label_1 = "PRS algorithm
result", label_2 = "GA algorithm result")
```

```
## t = 25.8674
## df = 49.00001
## p-value = 3.178981e-30
## hypothesis: true difference in means is equal to 0
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval: [282029.8, 329541.2]
## sample estimates:
##    mean of PRS algorithm result = 305825.3
##    mean of GA algorithm result = 39.86294
##
## Conclusion: hypothesis is false, because p_value <= alpha
```

Funkcja Rastrigina

Wymiar 2D

```
run_t.test(prs_rastrigin_2D, ga_rastrigin_2D, label_1 = "PRS algorithm
result", label_2 = "GA algorithm result")
```

```
## t = 12.9952
## df = 49
## p-value = 1.70302e-17
## hypothesis: true difference in means is equal to 0
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval: [1.437971, 1.964061]
## sample estimates:
##    mean of PRS algorithm result = 1.701016
##    mean of GA algorithm result = 8.741563e-08
##
## Conclusion: hypothesis is false, because p_value <= alpha
```

Wymiar 10D

```
run_t.test(prs_rastrigin_10D, ga_rastrigin_10D, label_1 = "PRS algorithm
result", label_2 = "GA algorithm result")
```

```
## t = 57.81111
## df = 49.00001
## p-value = 9.41969e-47
## hypothesis: true difference in means is equal to 0
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval: [83.11309, 89.09938]
## sample estimates:
##    mean of PRS algorithm result = 86.10899
##    mean of GA algorithm result = 0.002755626
##
## Conclusion: hypothesis is false, because p_value <= alpha
```

Wymiar 20D

```
run_t.test(prs_rastrigin_20D, ga_rastrigin_20D, label_1 = "PRS algorithm
result", label_2 = "GA algorithm result")
```

```
## t = 111.4376
## df = 49.06511
## p-value = 1.133125e-60
## hypothesis: true difference in means is equal to 0
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval: [221.6304, 229.7703]
## sample estimates:
##    mean of PRS algorithm result = 226.0245
##    mean of GA algorithm result = 0.324142
##
## Conclusion: hypothesis is false, because p_value <= alpha
```

Analiza wyników

Algorytm PRS (Pure Random Search) i algorytm genetyczny (GA) działają poprawnie na wygenerowanych zbiorach danych. Obydwa algorytmy dążą do uzyskania minimalnej wartości optymalizowanej funkcji. Natomiast lepiej z tym zadaniem radzi sobie algorytm genetyczny.

Dla funkcji 2D (przy budżecie 1000 wywołań) średnia osiągana wartość jest rzędu 10^{-8} (funkcja Rastrigina) lub 10^{-5} (funkcja Rosenbrocka). Podczas gdy średnia drugiego algorytmu wynosi aż 1.7 (funkcja Rastrigina) oraz 0.6 (funkcja Rosenbrocka).

Dla większej ilości wymiarów (tj. 10, 20) nadal obserwujemy przewagę algorytmu GA nad PRS. Tutaj również algorytmy uzyskały lepsze wyniki dla funkcji Rastrigina (różniły się one o rząd/rzędy wielkości) niż dla funkcji Rosenbrocka.

Szczegółowe informacje dotyczące średnich dla poszczególnych wywołań funkcji oraz minimalnej uzyskanej wartości znajdują się poniżej.

Algorytm (GA) - średnie wyniki

Wymiar	Rastrigin	Rosenbrock
2	8.741563e-08	2.026331e-05
10	0.002755626	5.952174
20	0.324142	39.86294

Algorytm (PRS) - średnie wyniki

Wymiar	Rastrigin	Rosenbrock
2	1.701016	0.6033034
10	86.10899	28555.61
20	226.0245	305825.3

Dla każdego przykładu (każdej funkcji i wymiaru) zostały wykonane testy statystycznej różnicy pomiędzy wynikami algorytmów. Hipotezą zerową, jaką przyjeliśmy było zdarzenie: Różnica średnich jest równa 0. Hipotezą alternatywną natomiast był zdarzenie przeciwne, czyli: Różnica średnich jest różną od 0. Wszystkie wykonane test na podstawie 95% poziomu ufności, obliczajac p-wartość, dały podstawy do odrzucenia hipotezy zerowej. Biorąc to pod uwagę, można stwierdzić, że algorytmy PRS i GA daję statystycznie różne wyniki na poziomie ufności 95%.