

POLITECHNIKA WROCŁAWSKA

INTELIGENCJA OBLICZENIOWA I JEJ ZASTOSOWANIA

---

# Badanie algorytmu genetycznego z zakresu optymalizacji globalnej dla wybranych funkcji testowych

---

*Autorzy:*

Paweł ANDZIUL 200648

Marcin SŁOWIŃSKI 200638

*Prowadzący:*

dr hab. inż. Olgierd UNOLD,

prof. nadzw. PWr

29 marca 2017

# Spis treści

<b>1</b>	<b>Wprowadzenie</b>	<b>2</b>
<b>2</b>	<b>Implementacja</b>	<b>2</b>
2.1	Parametryzacja skryptu . . . . .	7
<b>3</b>	<b>Przebieg badań</b>	<b>7</b>
3.1	Branin (2 parametry) . . . . .	8
3.2	Gulf (3 parametry) . . . . .	11
3.3	CosMix4 (4 parametry) . . . . .	15
3.4	EMichalewicz (5 parametrów) . . . . .	15
3.5	Hartman6 (6 parametrów) . . . . .	15
3.6	PriceTransistor (9 parametrów) . . . . .	15
3.7	Schwefel (10 parametrów) . . . . .	15
3.8	Zeldasine20 (20 parametrów) . . . . .	15
<b>4</b>	<b>Podsumowanie</b>	<b>21</b>

# 1 Wprowadzenie

Algorytmy genetyczne to

W ramach laboratorium należało przeprowadzić testy algorytmu genetycznego dla różnych parametrów. Jako benchmark oceny należało użyć pakietu „getGlobalOpts” oraz języka R.

Pomiary wykonywano na 2 różnych jednostkach roboczych. Ich parametry nie są istotne z punktu widzenia analizy i możliwości porównania rezultatów.

## 2 Implementacja

Poniżej (listing 1) zamieszczono kod napisany w języku R przygotowany w celu umożliwienia przeprowadzenia pomiarów.

Listing 1: Skrypt w języku R wykorzystany do badań

```
1
2 rm(list=ls())
3
4 require("GA")
5 require("globalOptTests")
6 require("rgl")
7
8 # Params ----
9
10 n <- 10 # minimum 5
11 GAPopulation <- 50 # default 50
12 GAlterations <- 100 # default 100
13 GAMutations <- 0.1 # default 0.1
14 GACrossovers <- 0.8 # default 0.8
15
16 isSingleTest <- FALSE
17 graphs <- TRUE
18 quality <- 100 #graph resolutions
19
20 mutationTests <- seq(0, 1, 0.1)
21 crossoverTests <- seq(0, 1, 0.1)
22 elitismTests <- seq(0, 1, 0.1)
23 populationTests <- seq(10, 100, 5)
24 iterationTests <- seq(10, 200, 10)
25
26 # Functions ----
27
28 #funcName <- "Branin" #2d
29 funcName <- "Gulf" #3d
30 #funcName <- "CosMix4" #4d
31 #funcName <- "EMichalewicz" #5d
32 #funcName <- "Hartman6" #6d
33 #funcName <- "PriceTransistor" #9d
34 #funcName <- "Schwefel" #10d
35 #funcName <- "Zeldasine20" #20d
36
37 # Processing ----
38
```

```

39 dim <- getProblemDimen(funcName)
40 B <- matrix(unlist(getDefaultBounds(funcName)),ncol=dim,byrow=TRUE)
41 f <- function(xx) goTest(par=c(xx, rep(0, dim-length(xx))), fnName=funcName,
    checkDim = TRUE)
42 globalOpt <- getGlobalOpt(funcName)
43
44 if (graphs) {
45   xprobes <- abs(B[2,1] - B[1,1]) / quality
46   yprobes <- abs(B[2,2] - B[1,2]) / quality
47   x <- seq(B[1,1], B[2,1], by = xprobes)
48   y <- seq(B[1,2], B[2,2], by = yprobes)
49   z <- outer(x, y, Vectorize(function(x,y) f(c(x,y))))
50   nbcol = 100
51   color = rev(rainbow(nbcol, start = 0/6, end = 4/6))
52   zcol = cut(z, nbcol)
53   persp3d(x, y, z, theta=50, phi=25, expand=0.75, col=color[zcol],
54     ticktype="detailed",axes=TRUE)
55   persp3D(x, y, z, theta = -45, phi = 20, color.palette = jet.colors)
56 }
57
58 if (isSingleTest) {
59
60   vector <- rep(NA,n)
61   for (i in 1:n) {
62     GAmin <- ga(type = "real-valued", fitness = function(xx) -f(xx),
63       min = c(B[1,]), max = c(B[2,]),
64       popSize = GAPopulation, maxiter = GAIterations,
65       pmutation = GAMutations, pcrossover = GACrossovers)
66     solution <- matrix(unlist(GAmin@solution),ncol=dim,byrow=TRUE)
67     vector[i] <- f(solution[1,])
68   }
69   result <- matrix(c(vector),nrow = n,ncol = 1)
70   write.table(result, file = "resultsSingle.csv", row.names=FALSE, na="",
71     col.names=FALSE, sep=";")
72 } else {
73
74   gMin <- .Machine$integer.max
75   gBest <- NA
76
77   temp <- c()
78   values <- mutationTests
79   averages <- c()
80   for (mutation in values) {
81     sum <- 0
82     vector <- rep(NA,n)
83     for (i in 1:n) {
84       GAmin <- ga(type = "real-valued",
85         fitness = function(xx) -f(xx),
86         min = c(B[1,]), max = c(B[2,]),
87         popSize = GAPopulation, maxiter = GAIterations,
88         pmutation = mutation, pcrossover = GACrossovers)
89       solution <- matrix(unlist(GAmin@solution),ncol=dim,byrow=TRUE)
90       eval <- f(solution[1,])
91       if (eval < gMin) {
92         gMin <- eval

```

```

93     gBest <- GAmin
94   }
95   sum <- sum + eval
96   vector[i] <- eval
97 }
98 temp <- c(temp, vector)
99 averages <- c(averages, (sum / n))
100 }
101 result <- matrix(c(temp),nrow = n,ncol = length(values))
102 write.table(result, file = "resultsMutations.csv", row.names=FALSE, na="",
103             col.names=FALSE, sep=";")
104
105 if (graphs) {
106   plot(values, averages,
107         main="Function values for different mutation probabilities",
108         ylim=c(min(c(averages,globalOpt)),max(c(averages,globalOpt))),
109         type="l", col="red", xlab="params", ylab="value")
110   abline(globalOpt,0, col="green")
111 }
112
113 temp <- c()
114 values <- crossoverTests
115 averages <- c()
116 for (crossover in values) {
117   sum <- 0
118   vector <- rep(NA,n)
119   for (i in 1:n) {
120     GAmin <- ga(type = "real-valued",
121                 fitness = function(xx) -f(xx),
122                 min = c(B[1,]), max = c(B[2,]),
123                 popSize = GAPopulation, maxiter = GAIterations,
124                 pmutation = GAMutations, pcrossover = crossover)
125     solution <- matrix(unlist(GAmin@solution),ncol=dim,byrow=TRUE)
126     eval <- f(solution[i,])
127     if (eval < gMin) {
128       gMin <- eval
129       gBest <- GAmin
130     }
131     sum <- sum + eval
132     vector[i] <- eval
133   }
134   temp <- c(temp, vector)
135   averages <- c(averages, (sum / n))
136 }
137 result <- matrix(c(temp),nrow = n,ncol = length(values))
138 write.table(result, file = "resultsCrossover.csv", row.names=FALSE, na="",
139             col.names=FALSE, sep=";")
140
141 if (graphs) {
142   plot(values, averages,
143         main="Function values for different crossover probabilities",
144         ylim=c(min(c(averages,globalOpt)),max(c(averages,globalOpt))),
145         type="l", col="red", xlab="params", ylab="value")
146   abline(globalOpt,0, col="green")
147 }

```

```

147 temp <- c()
148 values <- elitismTests
149 averages <- c()
150 for (elitism in values) {
151   sum <- 0
152   vector <- rep(NA,n)
153   for (i in 1:n) {
154     GAmin <- ga(type = "real-valued",
155               fitness = function(xx) -f(xx),
156               min = c(B[1,]), max = c(B[2,]),
157               popSize = GAPopulation, maxiter = GAIterations,
158               pmutation = GAMutations, pcrossover = GACrossovers, elitism =
               elitism)
159     solution <- matrix(unlist(GAmin@solution),ncol=dim,byrow=TRUE)
160     eval <- f(solution[1,])
161     if (eval < gMin) {
162       gMin <- eval
163       gBest <- GAmin
164     }
165     sum <- sum + eval
166     vector[i] <- eval
167   }
168   temp <- c(temp, vector)
169   averages <- c(averages, (sum / n))
170 }
171 result <- matrix(c(temp),nrow = n,ncol = length(values))
172 write.table(result, file = "resultsElitism.csv", row.names=FALSE, na="",
173             col.names=FALSE, sep=";")
174
175 if (graphs) {
176   plot(values, averages,
177         main="Function values for different elitism",
178         ylim=c(min(c(averages,globalOpt)),max(c(averages,globalOpt))),
179         type="l", col="red", xlab="params", ylab="value")
180   abline(globalOpt,0, col="green")
181 }
182
183 temp <- c()
184 values <- populationTests
185 averages <- c()
186 for (population in values) {
187   sum <- 0
188   vector <- rep(NA,n)
189   for (i in 1:n) {
190     GAmin <- ga(type = "real-valued",
191               fitness = function(xx) -f(xx),
192               min = c(B[1,]), max = c(B[2,]),
193               popSize = population, maxiter = GAIterations,
194               pmutation = GAMutations, pcrossover = GACrossovers)
195     solution <- matrix(unlist(GAmin@solution),ncol=dim,byrow=TRUE)
196     eval <- f(solution[1,])
197     if (eval < gMin) {
198       gMin <- eval
199       gBest <- GAmin
200     }
201     sum <- sum + eval

```

```

201     vector[i] <- eval
202   }
203   temp <- c(temp, vector)
204   averages <- c(averages, (sum / n))
205 }
206 result <- matrix(c(temp),nrow = n,ncol = length(values))
207 write.table(result, file = "resultsPopulation.csv", row.names=FALSE, na="",
208             col.names=FALSE, sep=";")
209
210 if (graphs) {
211   plot(values, averages,
212        main="Function values for different population sizes",
213        ylim=c(min(c(averages,globalOpt)),max(c(averages,globalOpt))),
214        type="l", col="red", xlab="params", ylab="value")
215   abline(globalOpt,0, col="green")
216 }
217
218 temp <- c()
219 values <- iterationTests
220 averages <- c()
221 for (iterations in values) {
222   sum <- 0
223   vector <- rep(NA,n)
224   for (i in 1:n) {
225     GAmin <- ga(type = "real-valued",
226                fitness = function(xx) -f(xx),
227                min = c(B[1,]), max = c(B[2,]),
228                popSize = GAPopulation, maxiter = iterations,
229                pmutation = GAMutations, pcrossover = GACrossovers)
230     solution <- matrix(unlist(GAmin@solution),ncol=dim,byrow=TRUE)
231     eval <- f(solution[1,])
232     if (eval < gMin) {
233       gMin <- eval
234       gBest <- GAmin
235     }
236     sum <- sum + eval
237     vector[i] <- eval
238   }
239   temp <- c(temp, vector)
240   averages <- c(averages, (sum / n))
241 }
242 result <- matrix(c(temp),nrow = n,ncol = length(values))
243 write.table(result, file = "resultsIterations.csv", row.names=FALSE, na="",
244             col.names=FALSE, sep=";")
245
246 if (graphs) {
247   plot(values, averages,
248        main="Function values for different number of iterations",
249        ylim=c(min(c(averages,globalOpt)),max(c(averages,globalOpt))),
250        type="l", col="red", xlab="params", ylab="value")
251   abline(globalOpt,0, col="green")
252 }
253
254 if (graphs) {

```

```

255 summary(GAmin)
256 filled.contour(x, y, z, color.palette = jet.colors, nlevels = 24,
257   plot.axes = {
258     axis(1);
259     axis(2);
260     points(solution[1,1], solution[1,2], pch = 3, cex = 5, col = "black", lwd
      = 2)
261   }
262 )
263 plot(GAmin)
264 }

```

## 2.1 Parametryzacja skryptu

Parametryzacji podlega jedynie algorytm genetyczny. Wybór funkcji do optymalizacji odbywa się przez podanie jej nazwy. Pozostałe dane są odczytywane z pakietu „globalOpt-Tests”.

## 3 Przebieg badań

Do badań zostały wybrane funkcje o różnych wymiarach zaczynając na 2 kończąc na 20. Poniżej wymieniono te funkcje wraz z ilością wymiarów podaną w nawiasie.

- Branin (2)
- Gulf (3)
- CosMix4 (4)
- EMichalewicz (5)
- Hartman6 (6)
- PriceTransistor (9)
- Schwefel (10)
- Zeldasine20 (20)

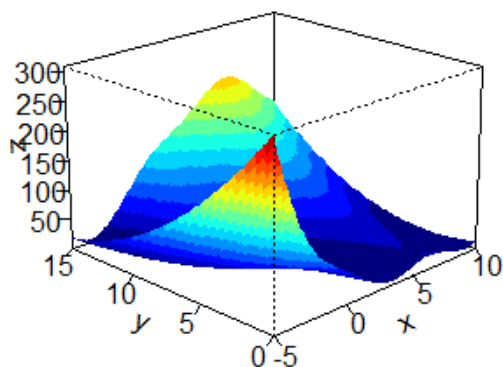
Każdy pomiar przeprowadzano 10-krotnie wyniki uśredniając. Domyślne parametry wynosiły kolejno:

- rozmiar populacji - 50
- liczba iteracji - 100
- p. mutacji - 0,1
- p. krzyżowania - 0,8



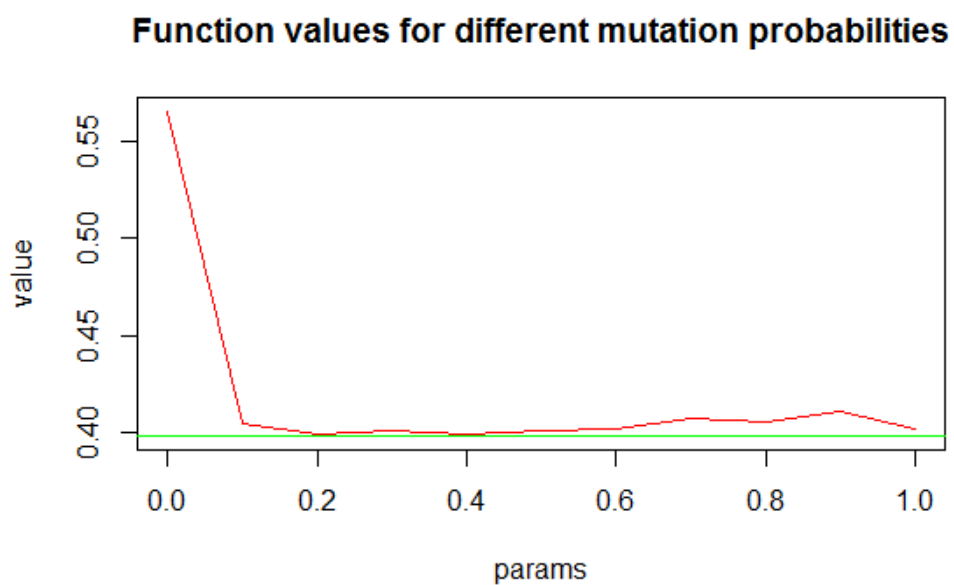
### 3.1 Branin (2 parametry)

Branin jest funkcją z dwoma parametrami. Na ilustracji (rys. 1) przedstawiono jej wykres.

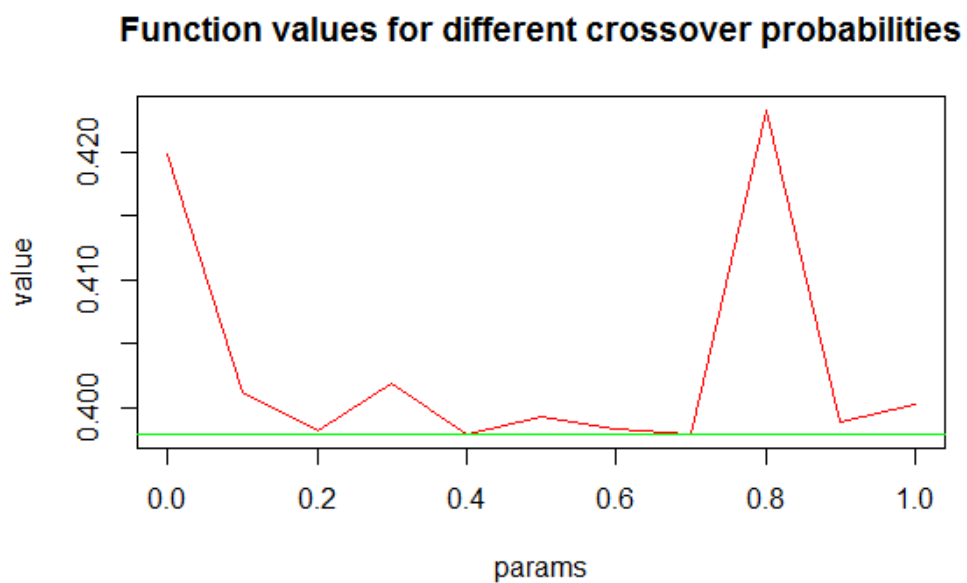


Rysunek 1: Wykres funkcji Branin ( $d=2$ )

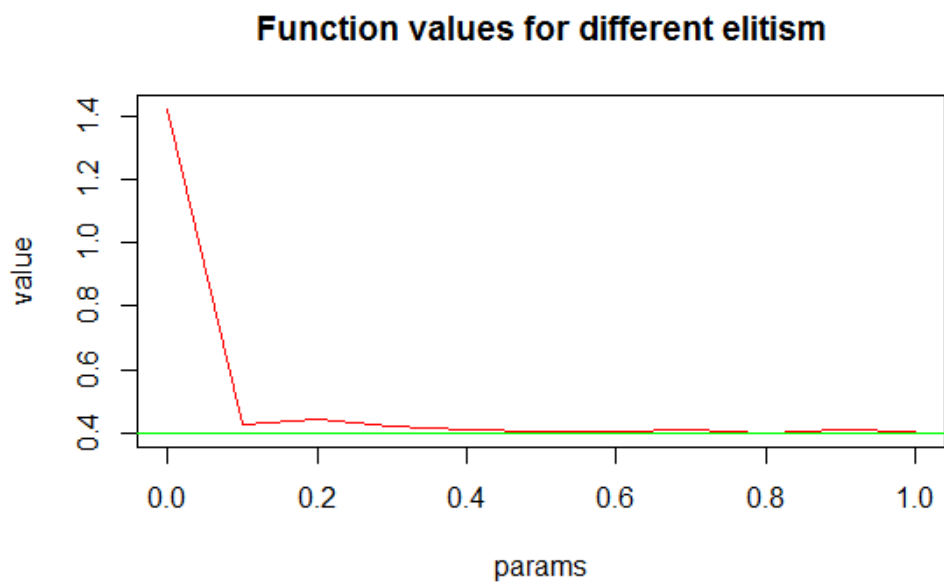
Na kolejnych stronach zamieszczono wyniki pomiarów dla różnych wartości parametrów algorytmu genetycznego.



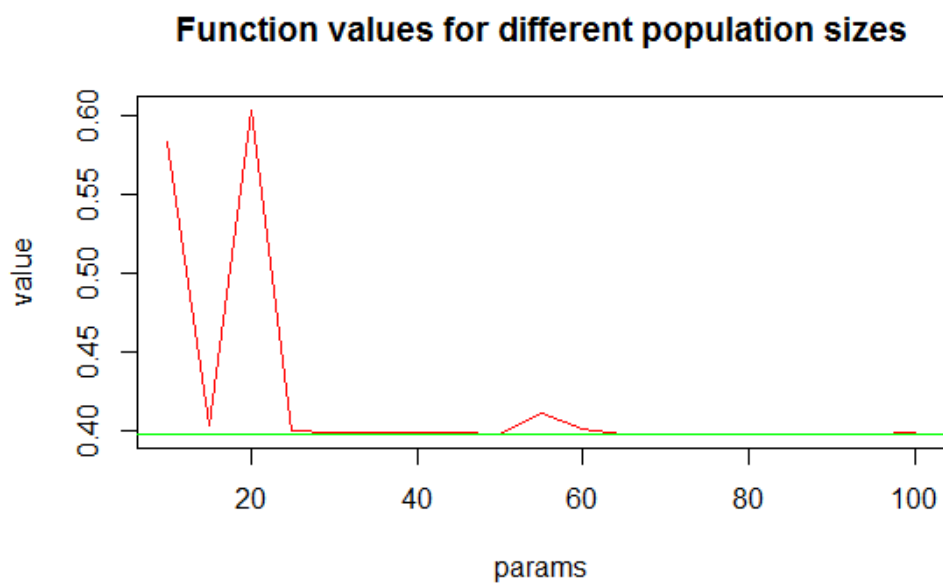
Rysunek 2: Wartość znalezionej optimum w zależności od prawdopodobieństwa mutacji



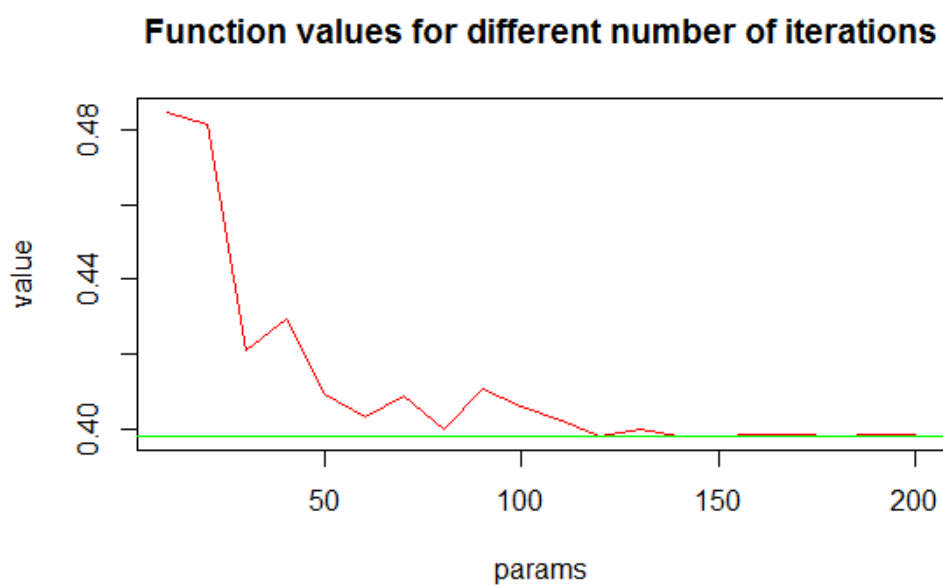
Rysunek 3: Wartość znalezionej optimum w zależności od prawdopodobieństwa krzyżowania



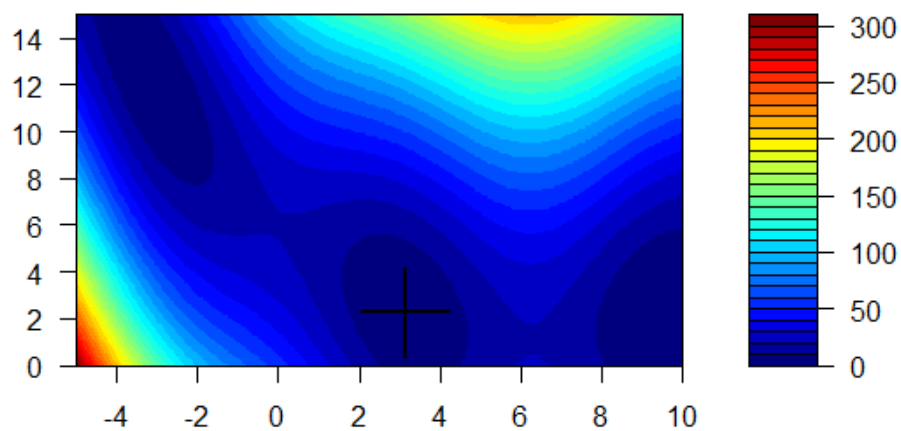
Rysunek 4: Wartość znalezionej optimum w zależności od przyjętego elitizmu



Rysunek 5: Wartość znalezione optimum w zależności od rozmiarów populacji



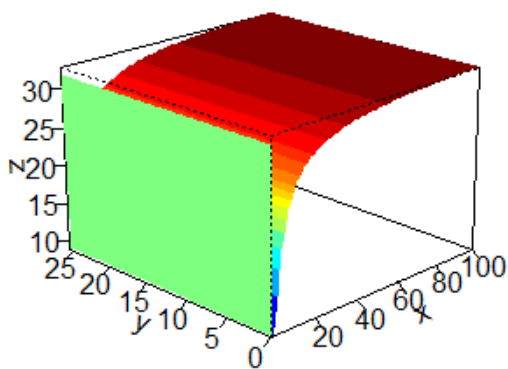
Rysunek 6: Wartość znalezione optimum w zależności od ilości iteracji



Rysunek 7: Poglądowa lokalizacja najlepszego znalezionej optimum

### 3.2 Gulf (3 parametry)

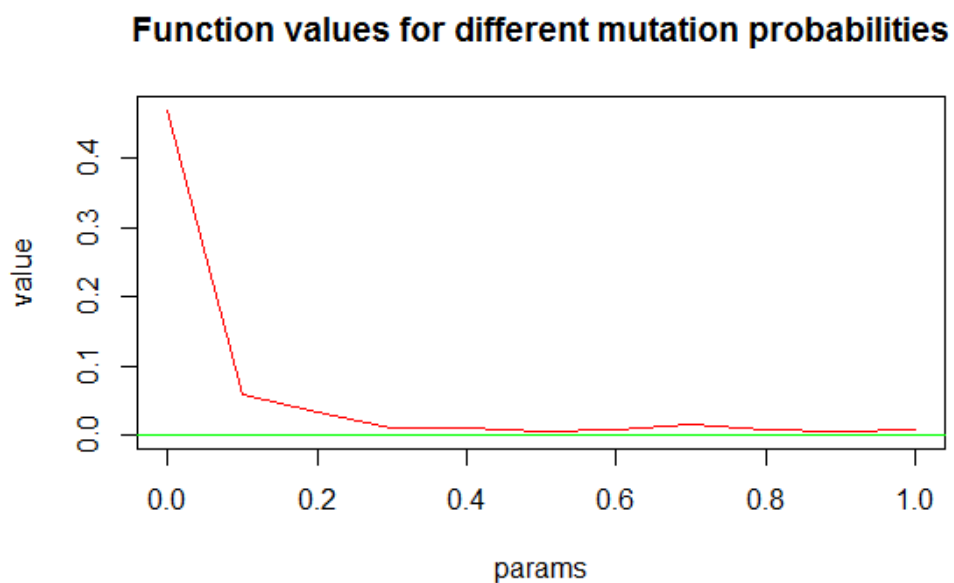
Gulf jest funkcją przyjmującą trzy parametry. Na ilustracji (rys. 8) przedstawiono jej wykres dla pierwszych dwóch wymiarów.



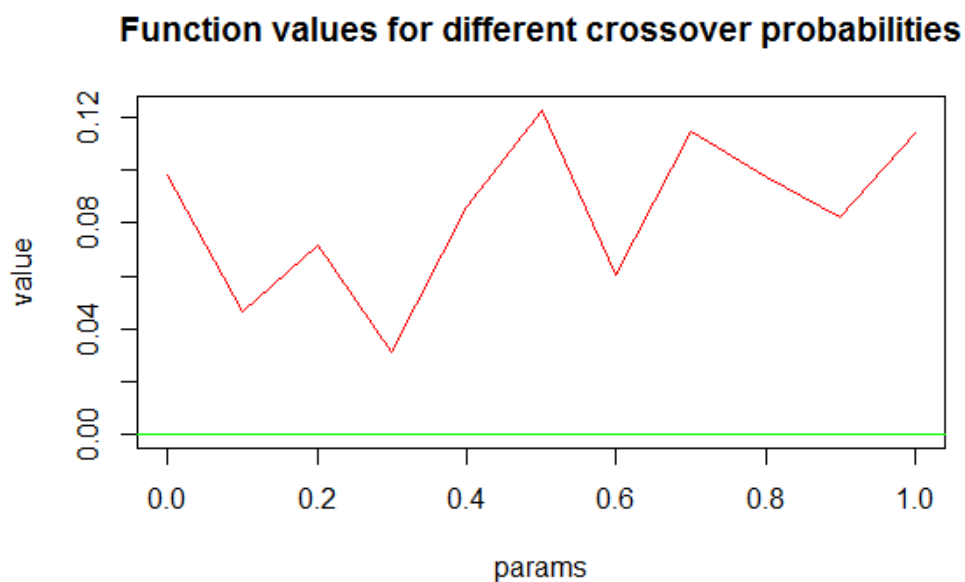
Rysunek 8: Wykres funkcji Gulf ( $d=3$ )

Na kolejnych stronach zamieszczono wyniki pomiarów dla różnych wartości parametrów

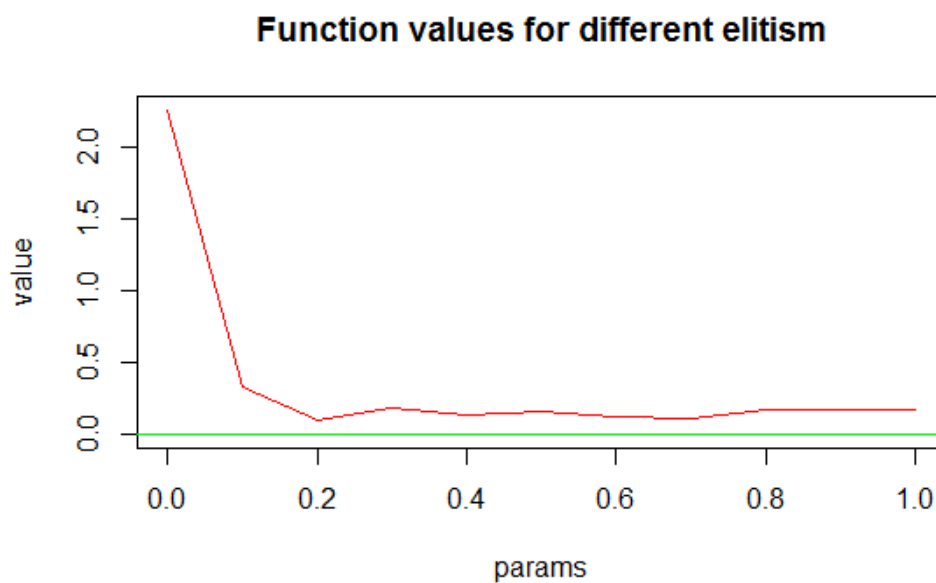
algorytmu genetycznego.



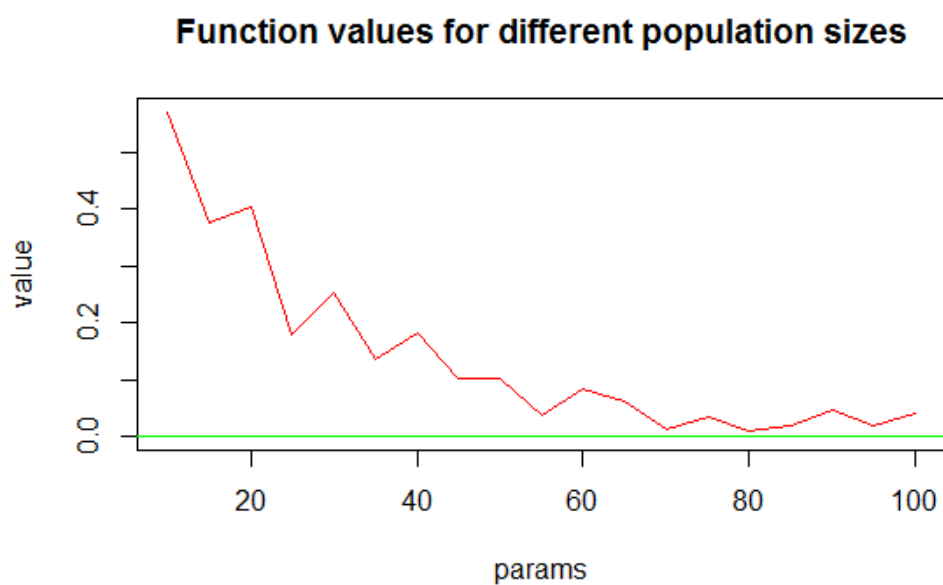
Rysunek 9: Wartość znalezionej optimum w zależności od prawdopodobieństwa mutacji



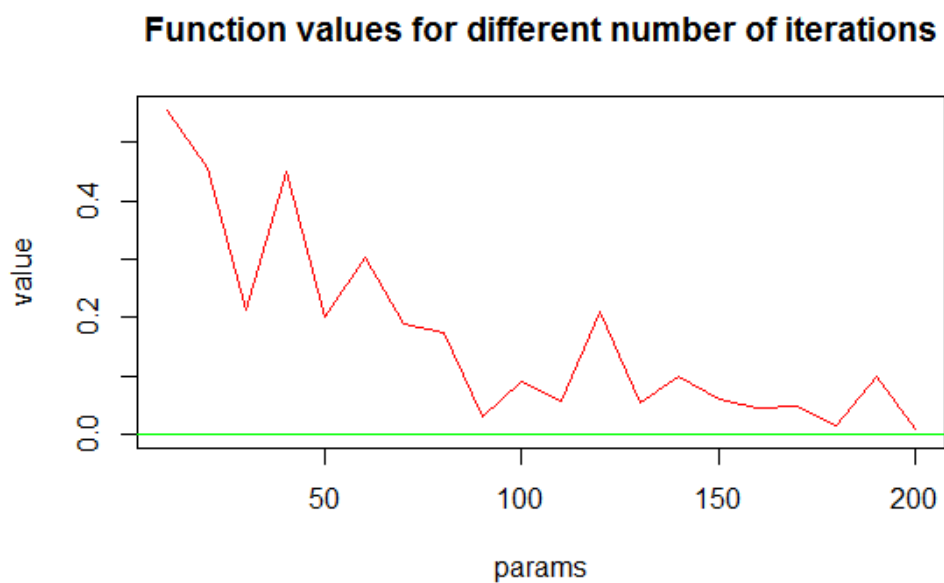
Rysunek 10: Wartość znalezionej optimum w zależności od prawdopodobieństwa krzyżowanie



Rysunek 11: Wartość znalezionej optimum w zależności od przyjętego elityzmu

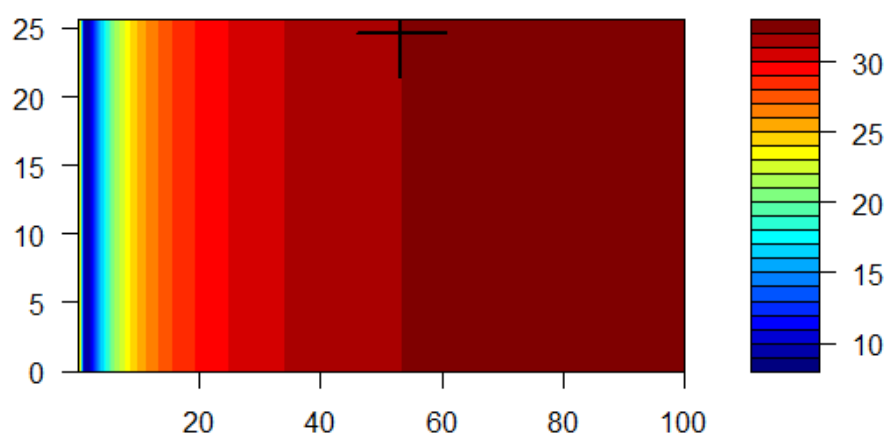


Rysunek 12: Wartość znalezionej optimum w zależności od rozmiarów populacji



Rysunek 13: Wartość znalezionej optimum w zależności od ilości iteracji

Jak możemy zauważyć na ilustracji poniżej (rys. 21) przedstawiona lokalizacja optimum nie jest poprawna, gdyż optymalizacji poddano wersję z 3 parametrami. Ogólnie rzecz biorąc gdyby 3 wymiar przedstawić w postaci gradientu kolorystycznego wtedy byłaby to poprawna lokalizacja niemniej trudna dla intuicyjnego sprawdzenia.



Rysunek 14: Poglądowa lokalizacja najlepszego znalezionej optimum

### 3.3 CosMix4 (4 parametry)

Test

### 3.4 EMichalewicz (5 parametrów)

Test

### 3.5 Hartman6 (6 parametrów)

Test

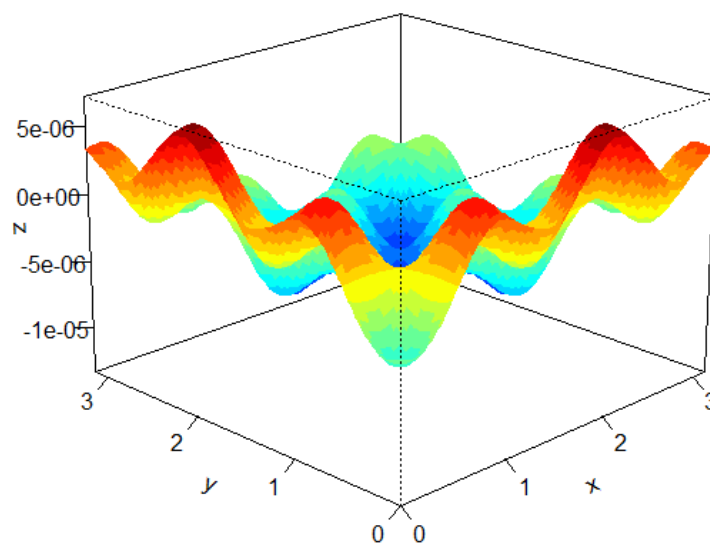
### 3.6 PriceTransistor (9 parametrów)

Test

### 3.7 Schwefel (10 parametrów)

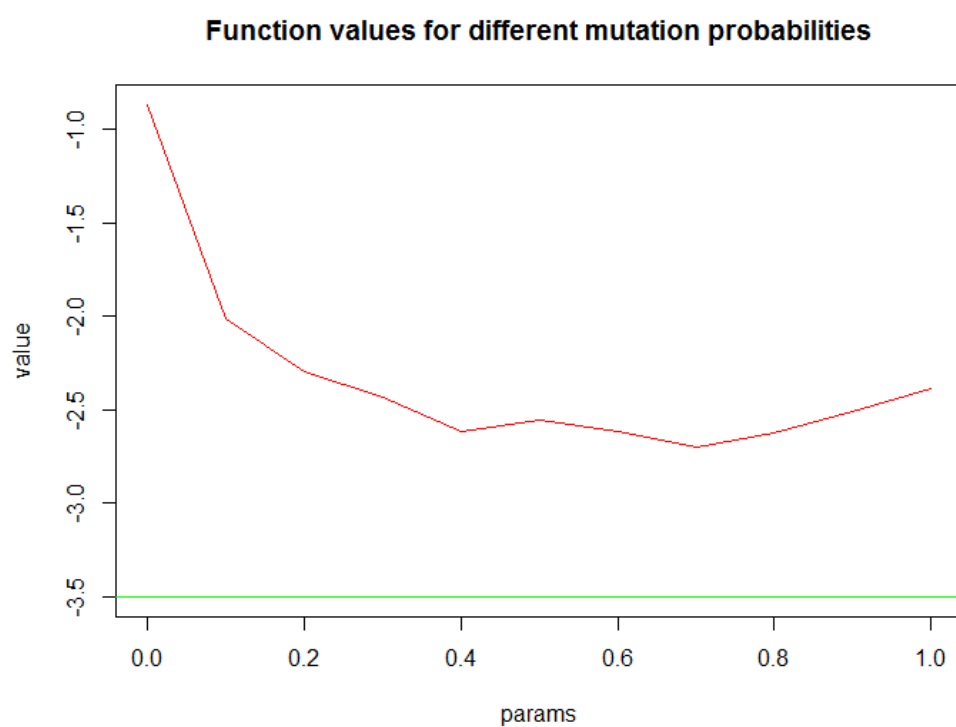
Test

### 3.8 Zeldasine20 (20 parametrów)

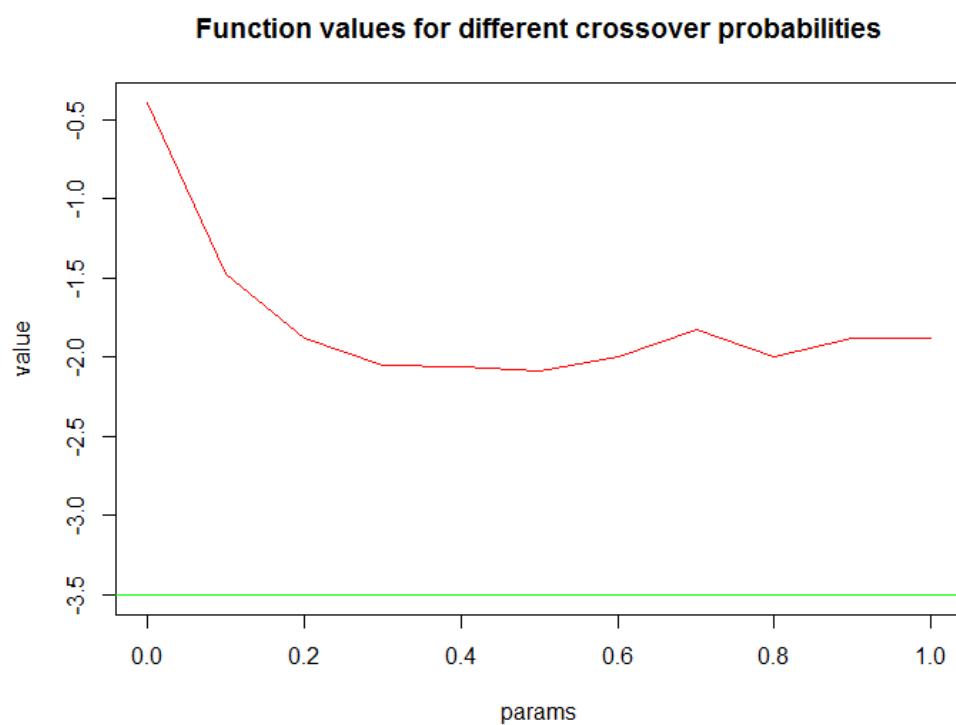


Rysunek 15: Poglądowa lokalizacja najlepszego znalezionej optimum

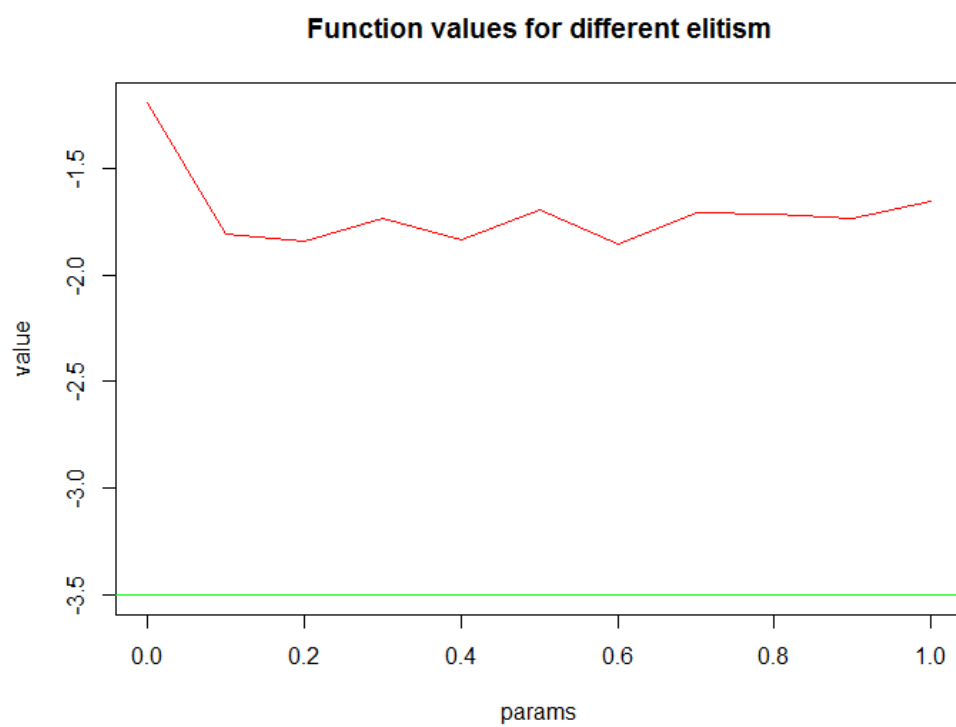




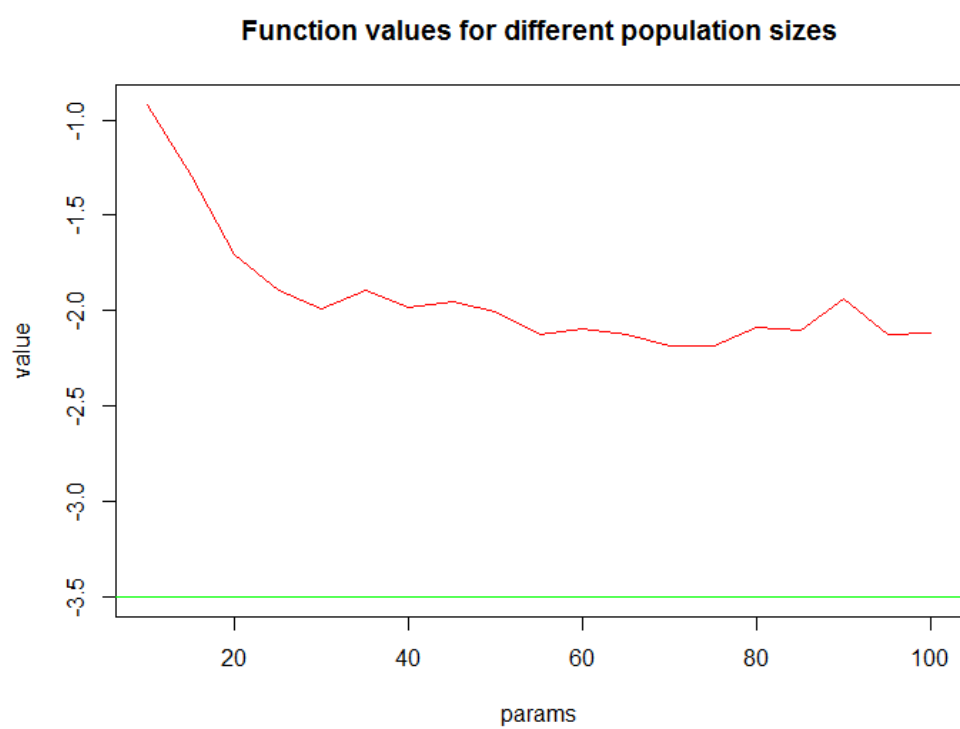
Rysunek 16: Wartość znalezionej optimum w zależności od prawdopodobieństwa mutacji



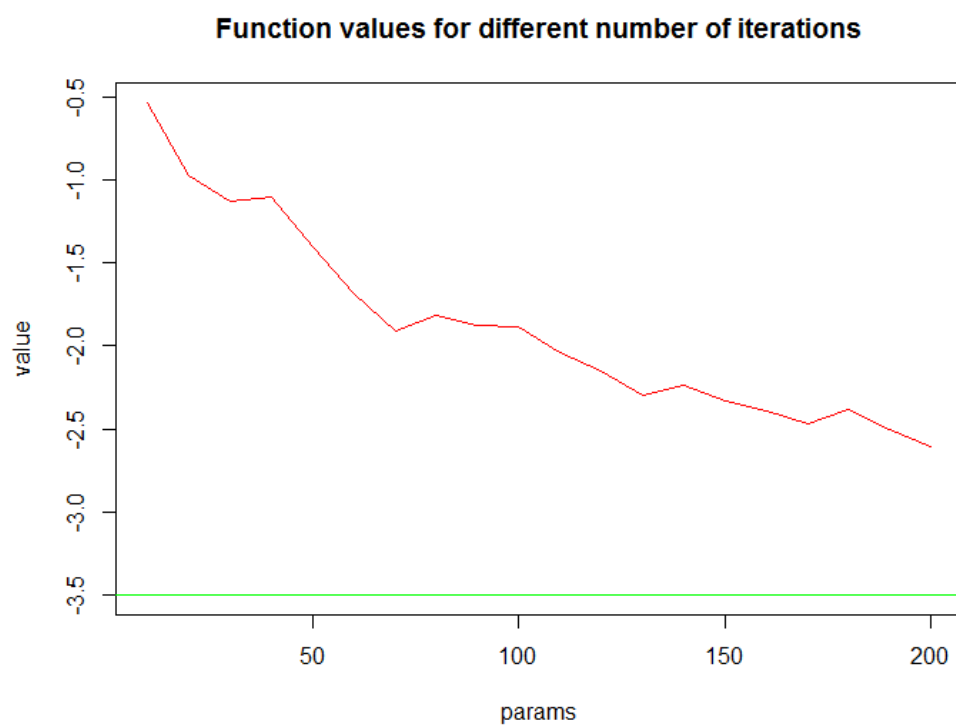
Rysunek 17: Wartość znalezionej optimum w zależności od prawdopodobieństwa krzyżowania



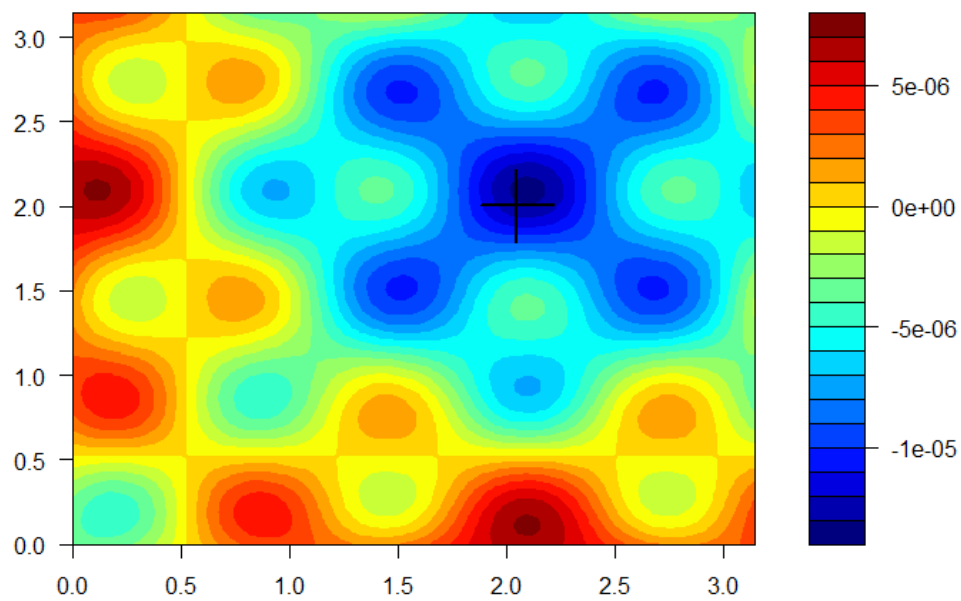
Rysunek 18: Wartość znalezionej optimum w zależności od przyjętego elitizmu



Rysunek 19: Wartość znalezionej optimum w zależności od rozmiarów populacji



Rysunek 20: Wartość znalezionej optimum w zależności od iteracji



Rysunek 21: Poglądowa lokalizacja najlepszego znalezione optimum

## 4 Podsumowanie

Test

Akapit

## Literatura

- [1] Artur Suchwałko “Wprowadzenie do R dla programistów innych języków”  
<https://cran.r-project.org/doc/contrib/R-dla-programistow-innych-jezykow.pdf>