

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

1	Wprowadzenie	2
2	Implementacja	2
2.1	Parametryzacja skryptu	7
3	Przebieg badań	7
3.1	Branin (2 parametry)	8
3.2	Gulf (3 parametry)	11
3.3	CosMix4 (4 parametry)	15
3.4	EMichalewicz (5 parametrów)	18
3.5	Hartman6 (6 parametrów)	22
3.6	PriceTransistor (9 parametrów)	25
3.7	Schwefel (10 parametrów)	29
3.8	Zeldasine20 (20 parametrów)	32
4	Podsumowanie	36

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

```

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

```

```

93     gMin <- eval
94     gBest <- GAmin
95   }
96   sum <- sum + eval
97   vector[i] <- eval
98 }
99 temp <- c(temp, vector)
100 averages <- c(averages, (sum / n))
101 }
102 result <- matrix(c(temp),nrow = n,ncol = length(values))
103 write.table(result, file = "resultsMutations.csv", row.names=FALSE, na="",
104             col.names=FALSE, sep=";")
105
106 if (graphs) {
107   plot(values, averages,
108        main="Function values for different mutation probabilities",
109        ylim=c(min(c(averages,globalOpt)),max(c(averages,globalOpt))),
110        type="l", col="red", xlab="params", ylab="value")
111   abline(globalOpt,0, col="green")
112 }
113
114 temp <- c()
115 values <- crossoverTests
116 averages <- c()
117 for (crossover in values) {
118   sum <- 0
119   vector <- rep(NA,n)
120   for (i in 1:n) {
121     GAmin <- ga(type = "real-valued",
122                fitness = function(xx) -f(xx),
123                min = c(B[1,]), max = c(B[2,]),
124                popSize = GAPopulation, maxiter = GAIterations,
125                pmutation = GAMutations, pcrossover = crossover)
126     solution <- matrix(unlist(GAmin@solution),ncol=dim,byrow=TRUE)
127     eval <- f(solution[i,])
128     if (eval < gMin) {
129       gMin <- eval
130       gBest <- GAmin
131     }
132     sum <- sum + eval
133     vector[i] <- eval
134   }
135   temp <- c(temp, vector)
136   averages <- c(averages, (sum / n))
137 }
138 result <- matrix(c(temp),nrow = n,ncol = length(values))
139 write.table(result, file = "resultsCrossover.csv", row.names=FALSE, na="",
140             col.names=FALSE, sep=";")
141
142 if (graphs) {
143   plot(values, averages,
144        main="Function values for different crossover probabilities",
145        ylim=c(min(c(averages,globalOpt)),max(c(averages,globalOpt))),
146        type="l", col="red", xlab="params", ylab="value")
147   abline(globalOpt,0, col="green")
148 }

```

```

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

```

```

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

```

```

255 if (graphs) {
256   summary(GAmin)
257   filled.contour(x, y, z, color.palette = jet.colors, nlevels = 24,
258     plot.axes = {
259       axis(1);
260       axis(2);
261       points(solution[1,1], solution[1,2], pch = 3, cex = 5, col = "black", lwd
         = 2)
262     }
263   )
264   plot(GAmin)
265 }

```

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)

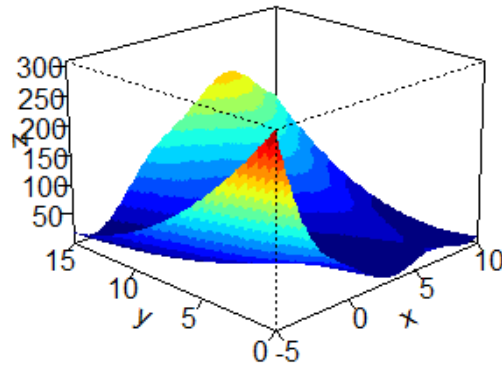
Każdy pomiar przeprowadzano 10-krotnie wyniki uśredniając. Domyślne parametry wynosiły kolejno:

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

3.1 Branin (2 parametry)

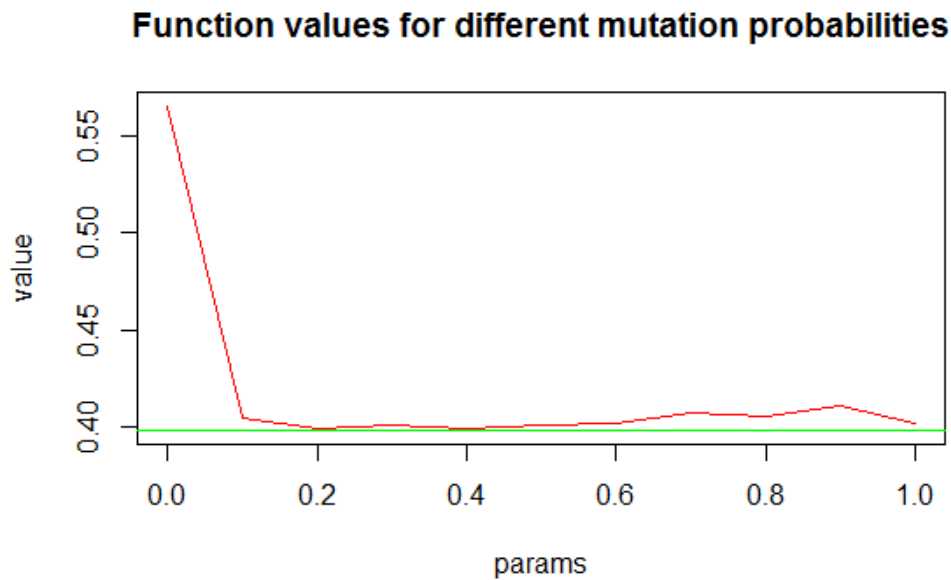
Branin jest funkcją z dwoma parametrami. Na ilustracji (rys. 1) przedstawiono jej wykres. Wzór funkcji zamieszczono poniżej (1).

$$f(\mathbf{x}) = a(x_2 - bx_1^2 + cx_1 - r)^2 + s(1 - t) \cos(x_1) + s \quad (1)$$

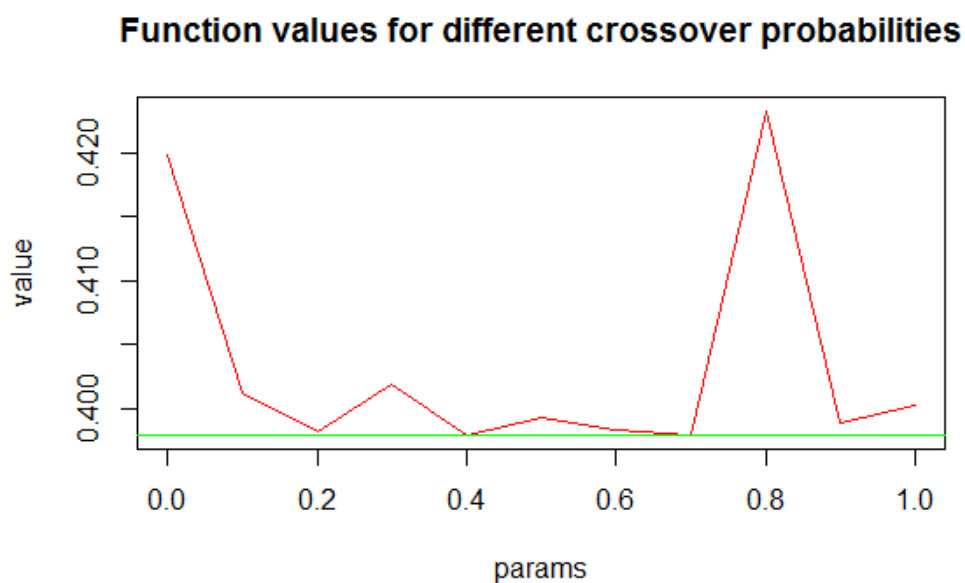


Rysunek 1: Wykres funkcji Branin (d=2)

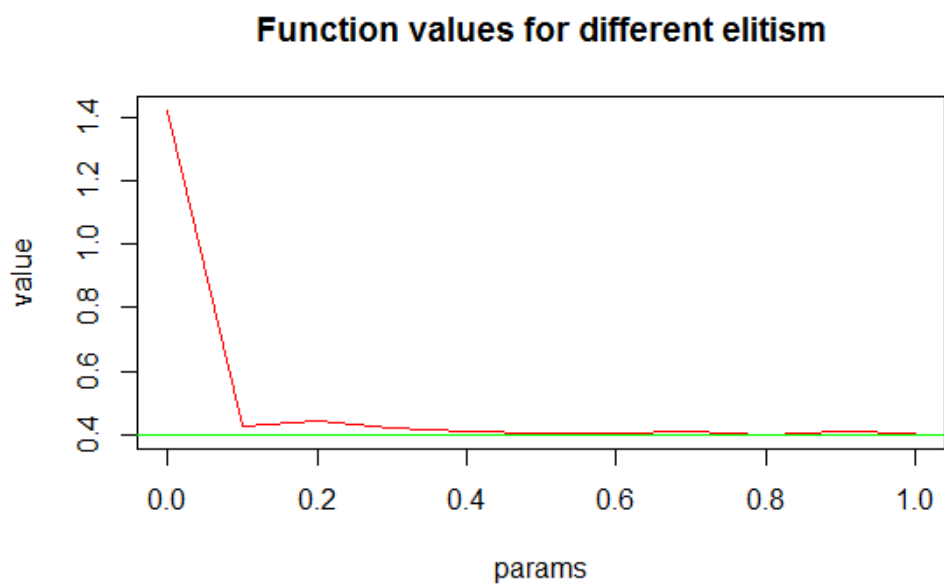
Na kolejnych stronach zamieszczono wyniki pomiarów dla różnych wartości parametrów algorytmu genetycznego.



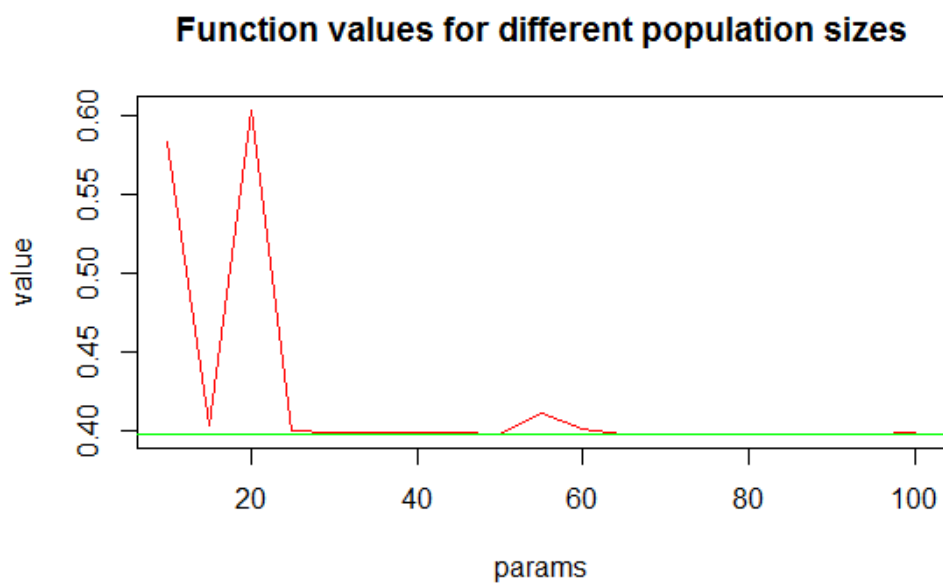
Rysunek 2: Wartość znalezionej optimum w zależności od prawdopodobieństwa mutacji



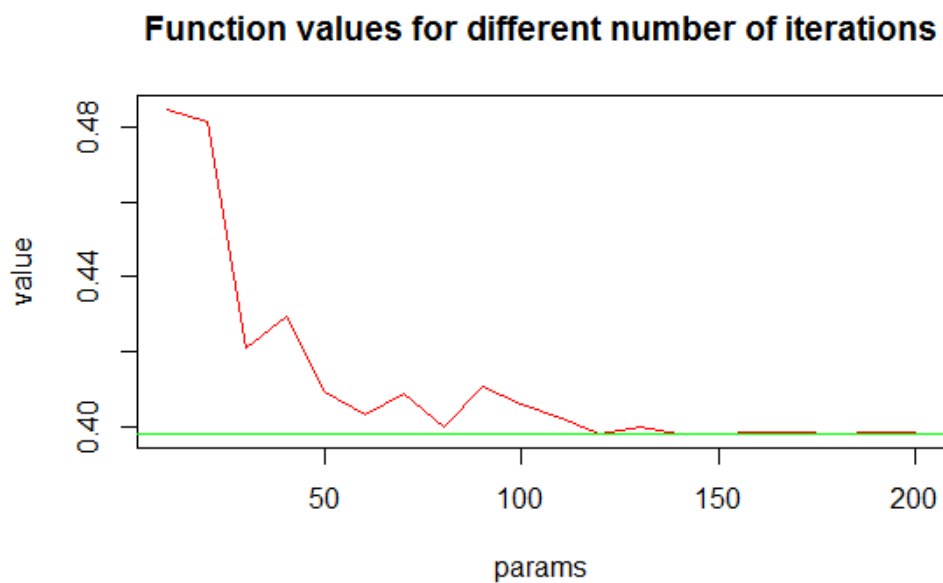
Rysunek 3: Wartość znalezione optimum w zależności od prawdopodobieństwa krzyżowania



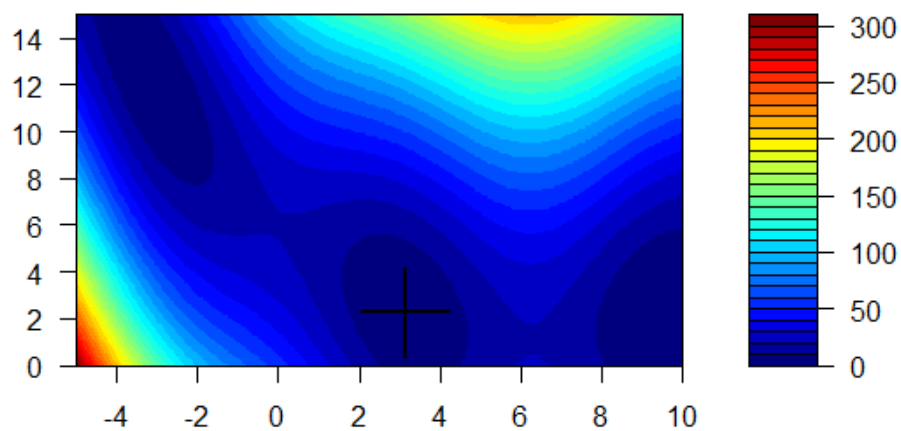
Rysunek 4: Wartość znalezione optimum w zależności od przyjętego elitizmu



Rysunek 5: Wartość znalezionej optimum w zależności od rozmiarów populacji



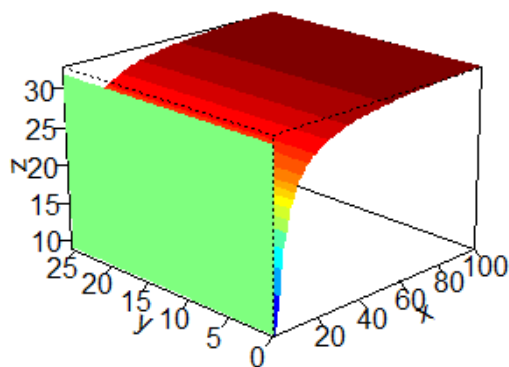
Rysunek 6: Wartość znalezionej optimum w zależności od ilości iteracji



Rysunek 7: Poglądowa lokalizacja najlepszego znalezionej optimum

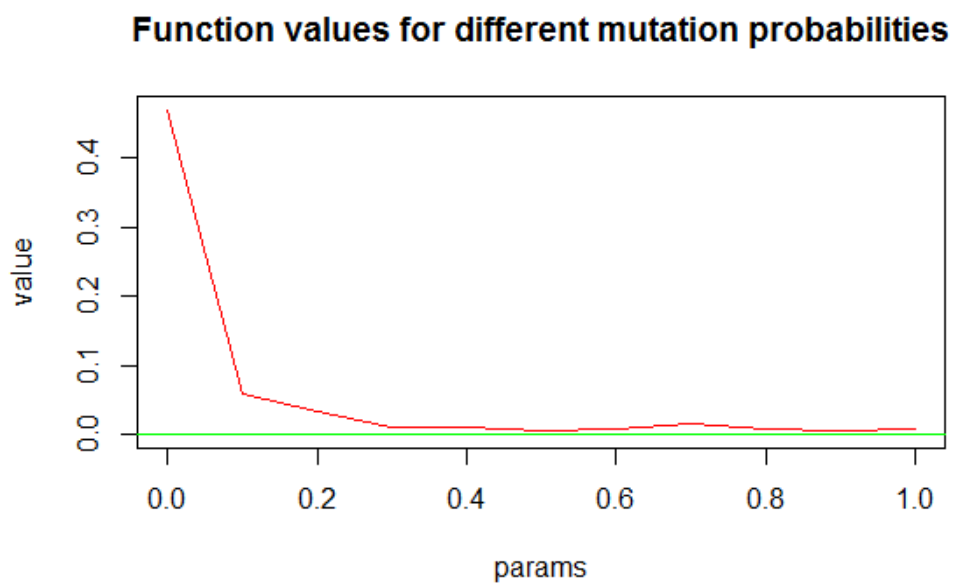
3.2 Gulf (3 parametry)

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

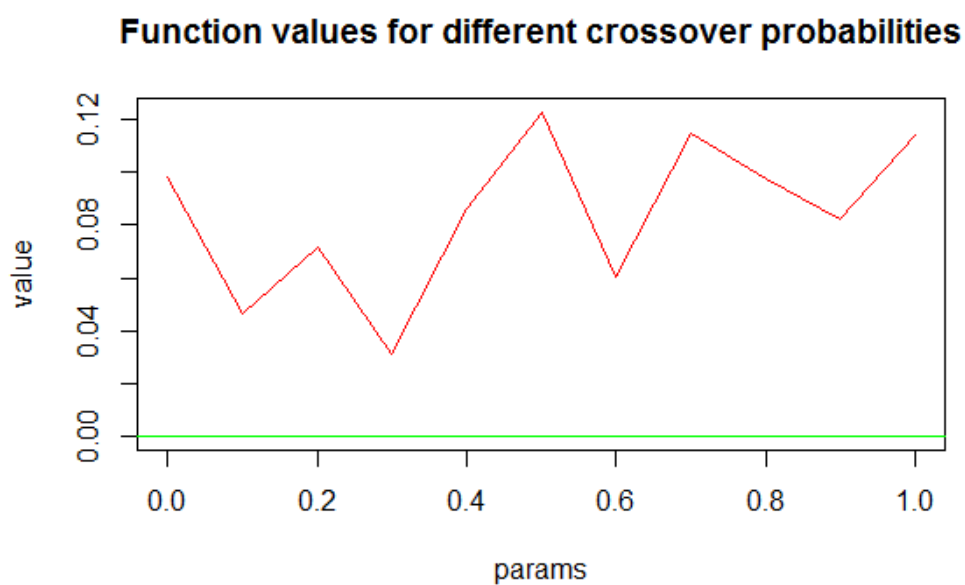


Rysunek 8: Wykres funkcji Gulf ($d=3$)

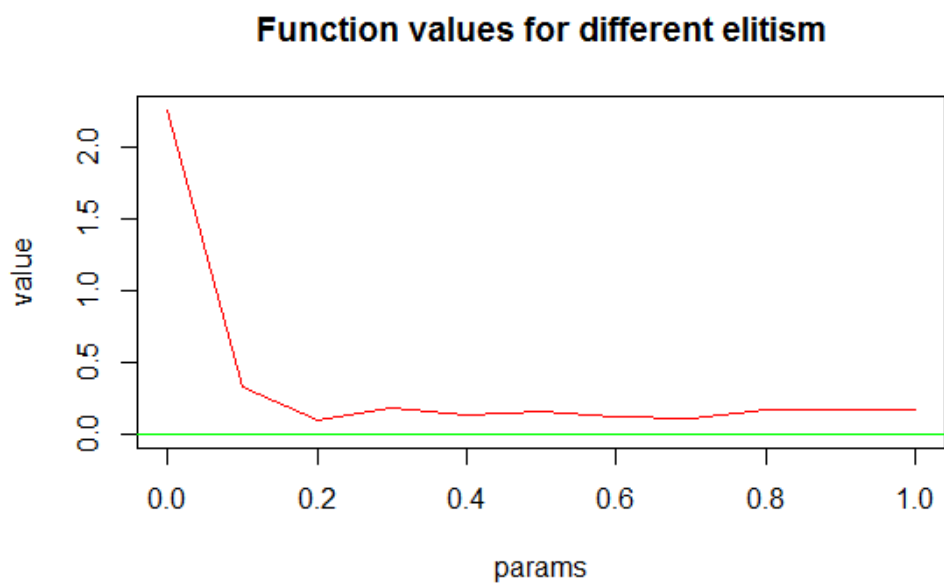
Na kolejnych stronach zamieszczono wyniki pomiarów dla różnych wartości parametrów algorytmu genetycznego.



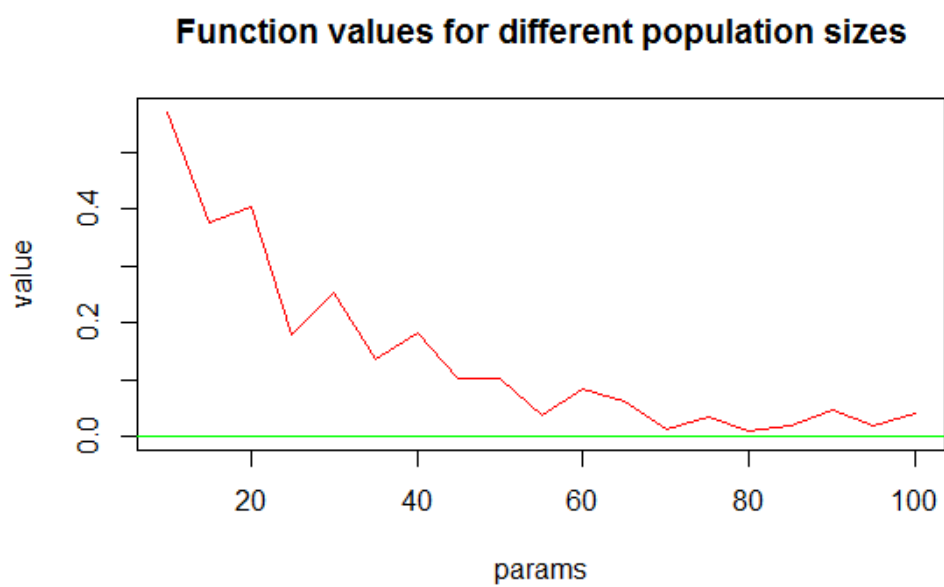
Rysunek 9: Wartość znalezionej optimum w zależności od prawdopodobieństwa mutacji



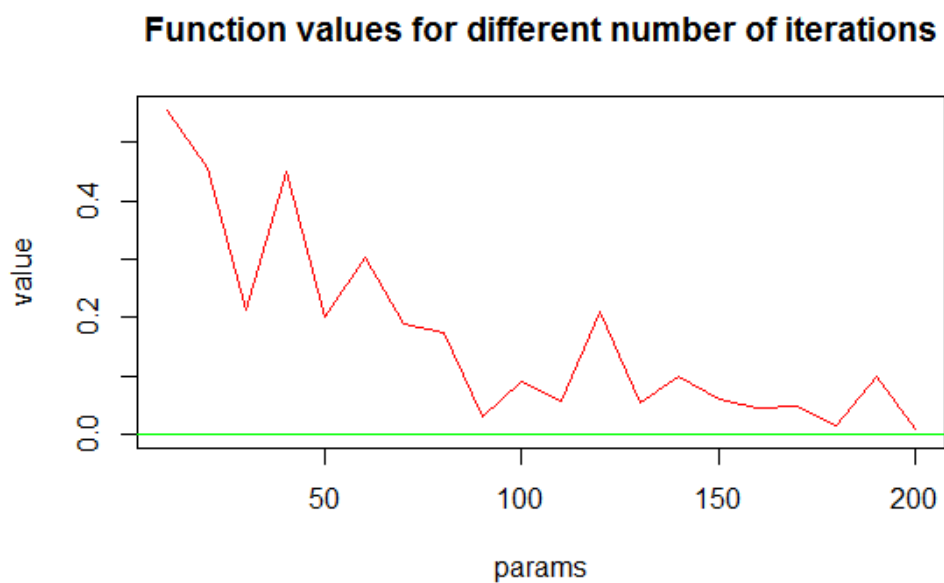
Rysunek 10: Wartość znalezionej optimum w zależności od prawdopodobieństwa krzyżowania



Rysunek 11: Wartość znalezionej optimum w zależności od przyjętego elityzmu

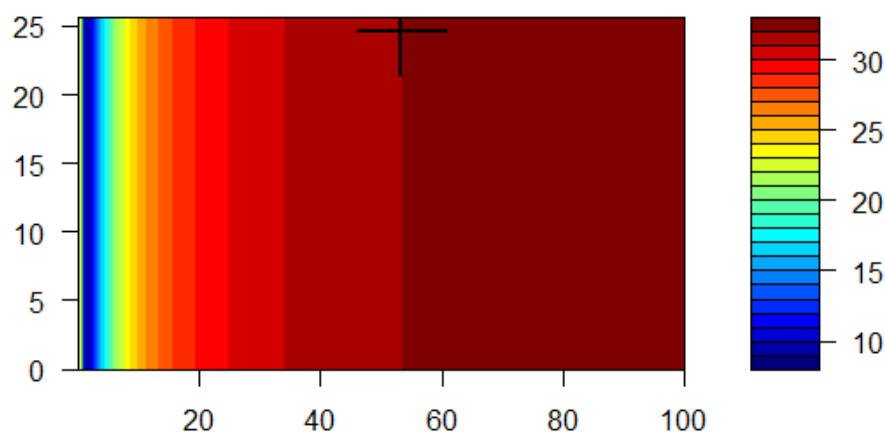


Rysunek 12: Wartość znalezionej optimum w zależności od rozmiarów populacji



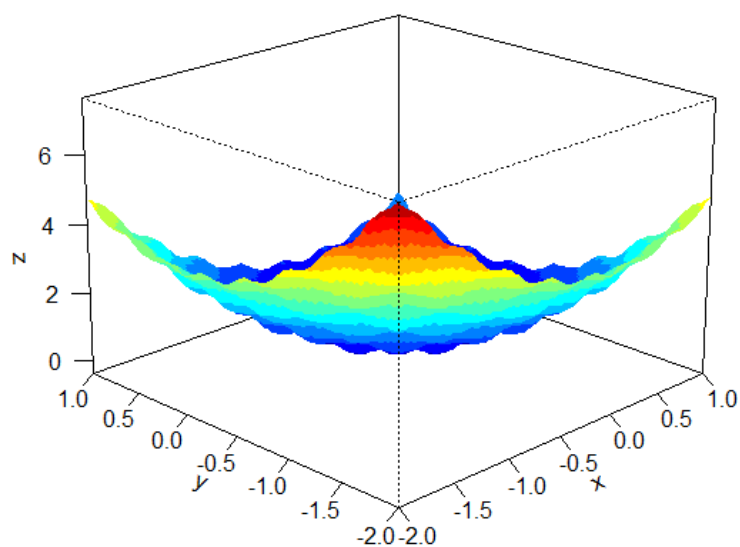
Rysunek 13: Wartość znalezionej optimum w zależności od ilości iteracji

Jak możemy zauważyć na ilustracji poniżej (rys. 14) 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.

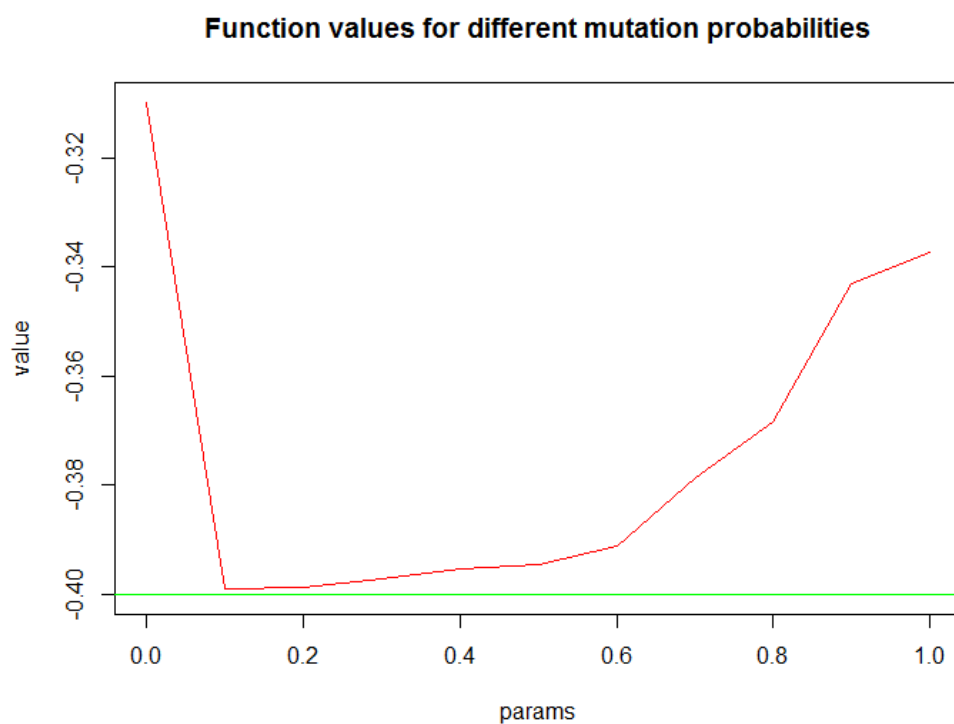


Rysunek 14: Poglądowa lokalizacja najlepszego znalezionej optimum

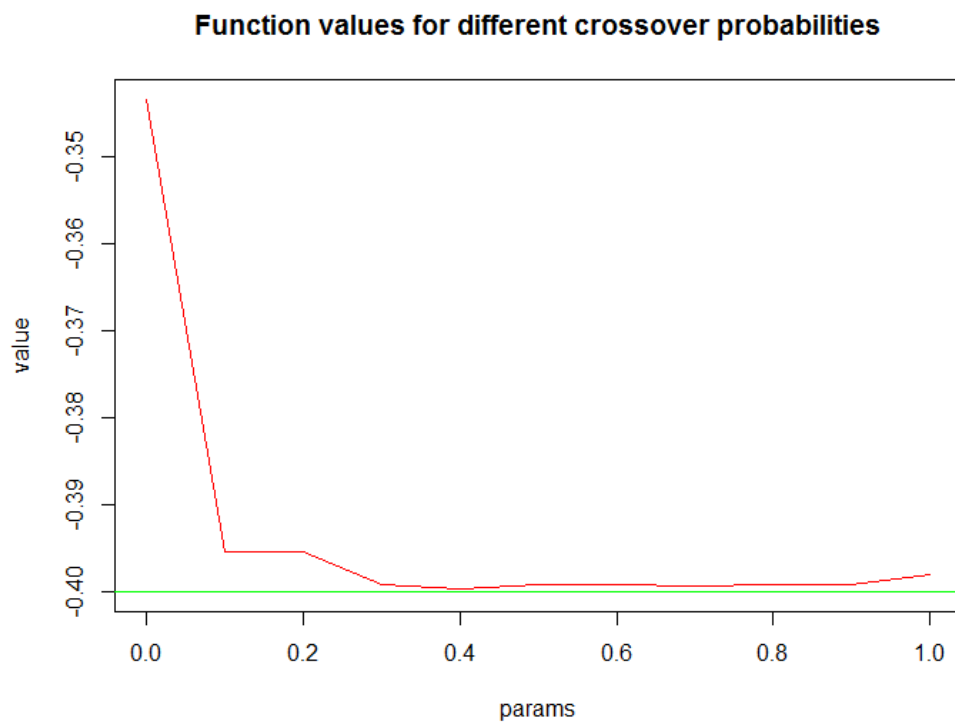
3.3 CosMix4 (4 parametry)



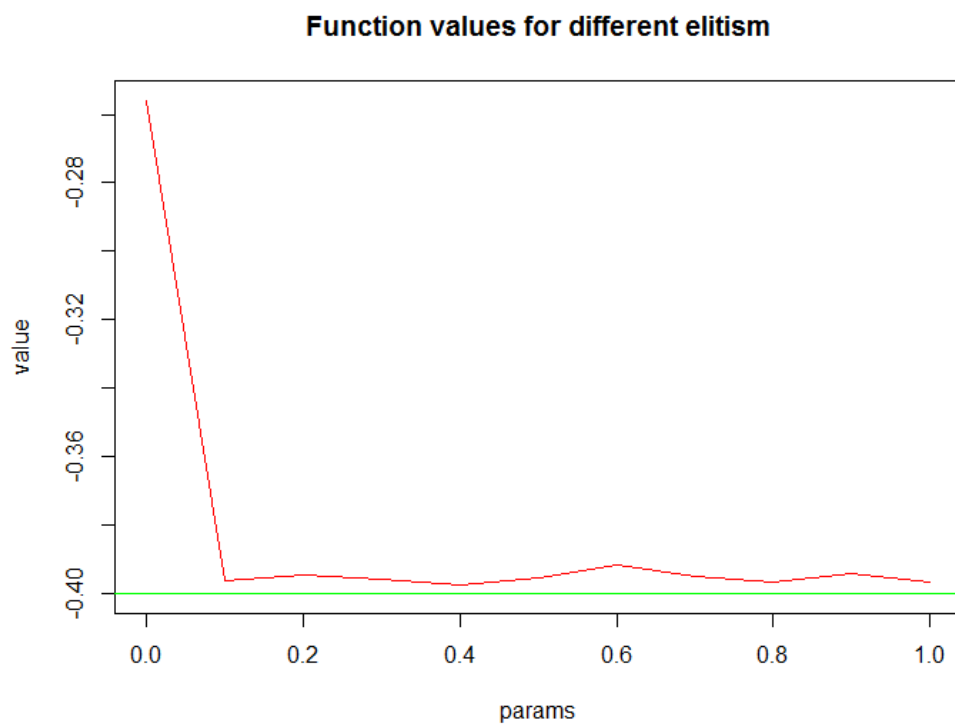
Rysunek 15: Wykres funkcji CosMix4 ($d=4$)



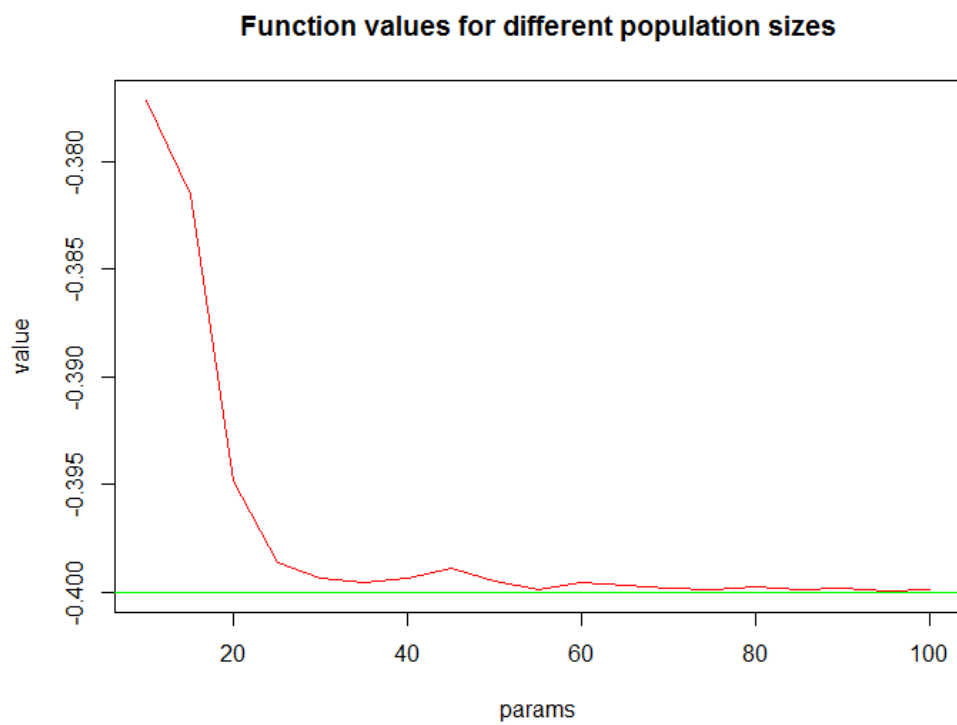
Rysunek 16: Wartość znalezionej optimum w zależności od prawdopodobieństwa mutacji



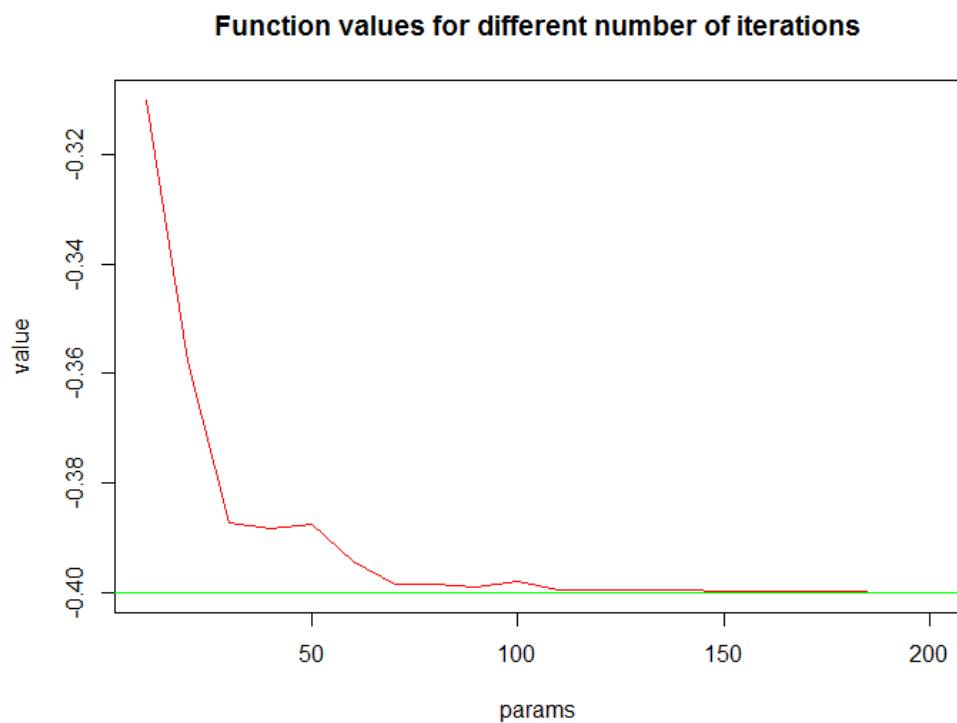
Rysunek 17: Wartość znalezionej optimum w zależności od prawdopodobieństwa krzyżowania



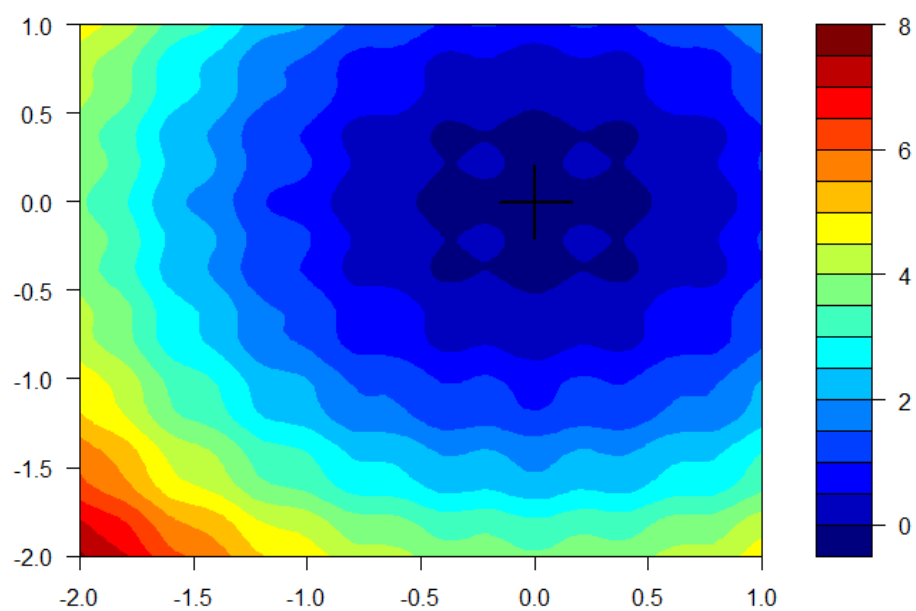
Rysunek 18: Wartość znalezionej optimum w zależności od przyjętego elitizmu



Rysunek 19: Wartość znalezione optimum w zależności od rozmiarów populacji

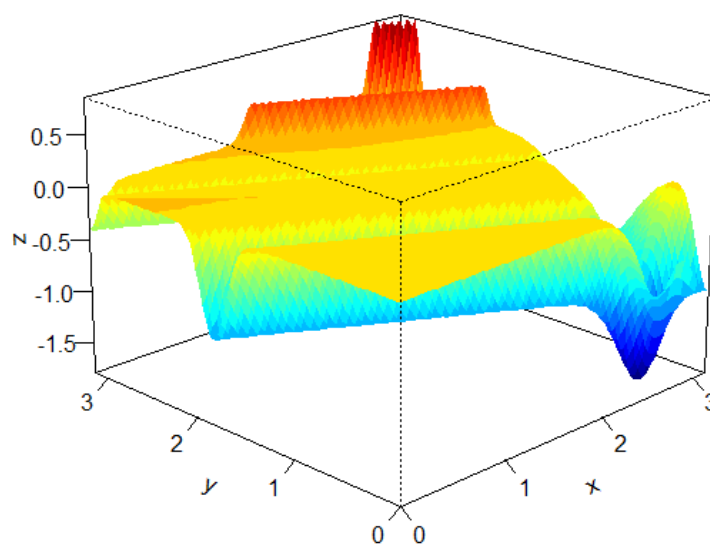


Rysunek 20: Wartość znalezione optimum w zależności od ilości iteracji

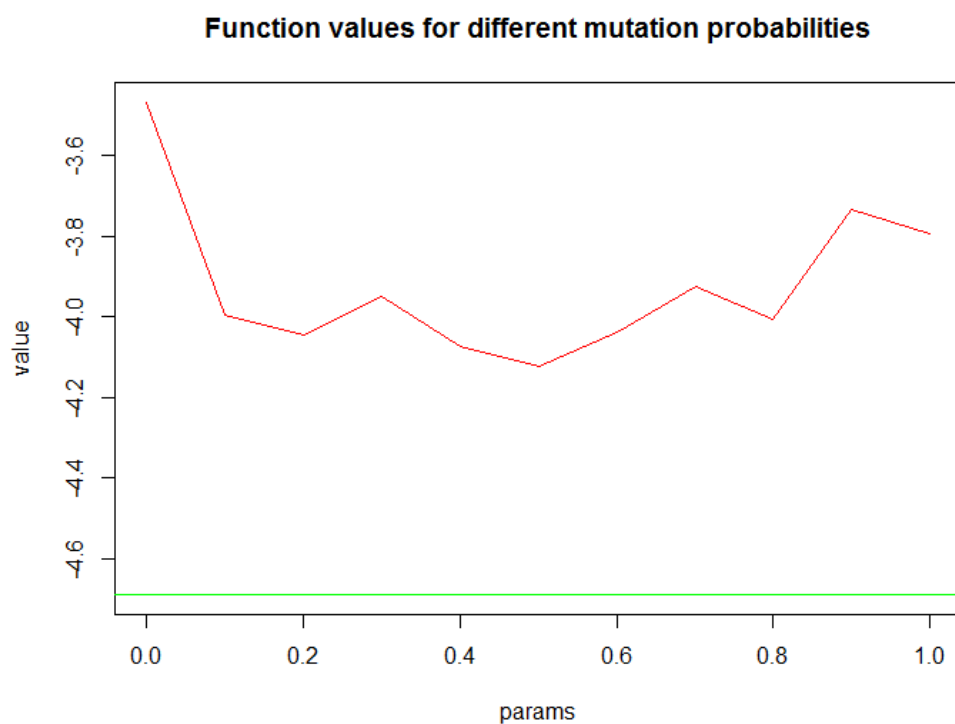


Rysunek 21: Poglądowa lokalizacja najlepszego znalezionej optimum

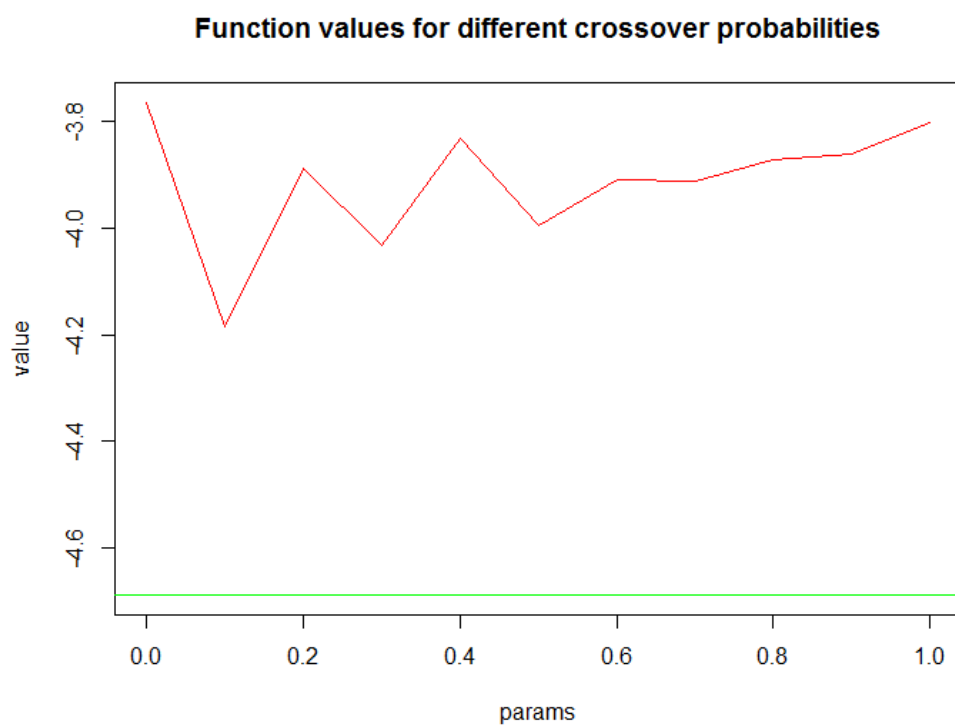
3.4 EMichalewicz (5 parametrów)



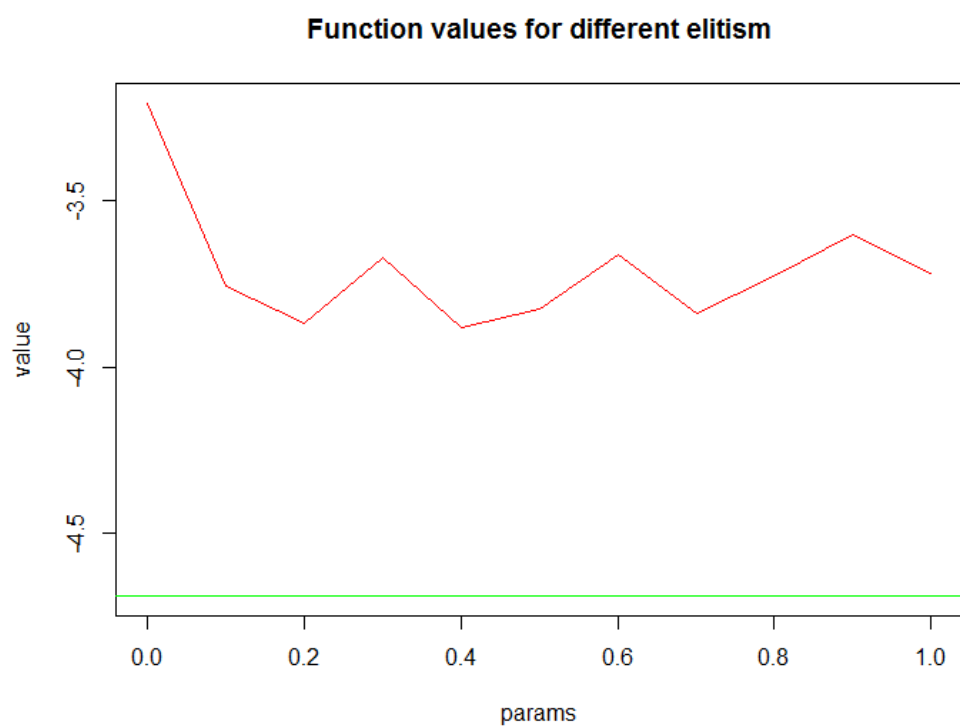
Rysunek 22: Wykres funkcji EMichalewicz ($d=5$)



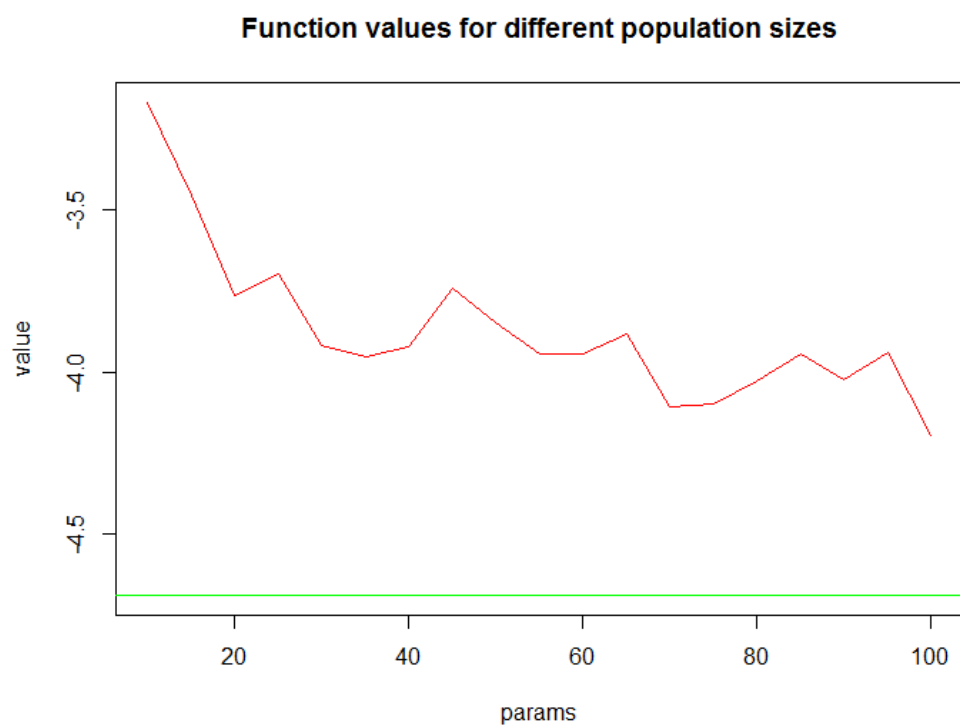
Rysunek 23: Wartość znalezionej optimum w zależności od prawdopodobieństwa mutacji



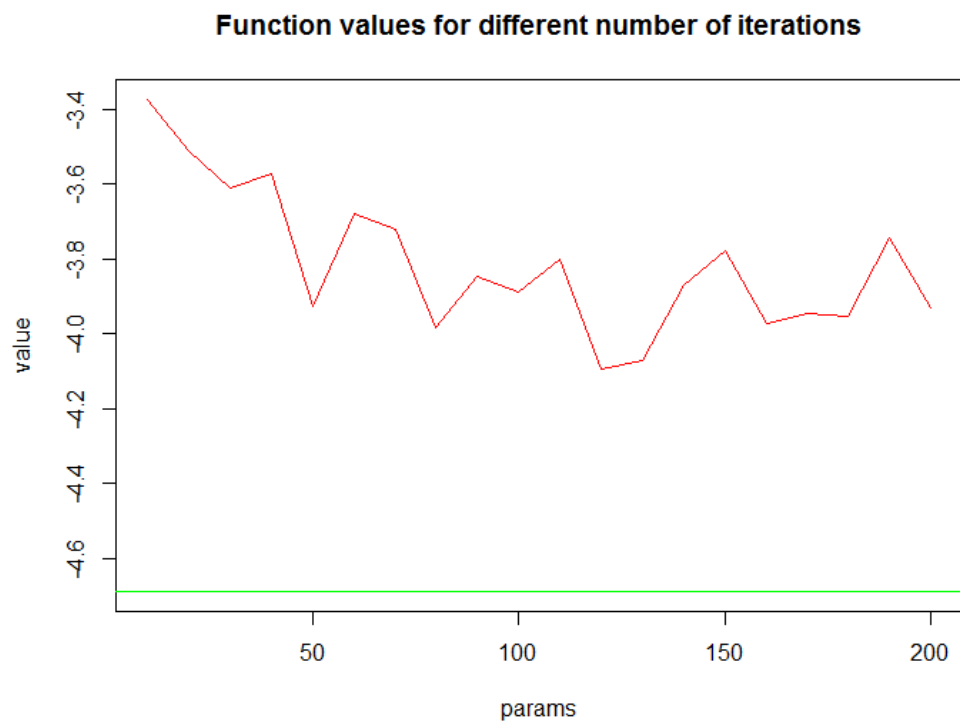
Rysunek 24: Wartość znalezionej optimum w zależności od prawdopodobieństwa krzyżowania



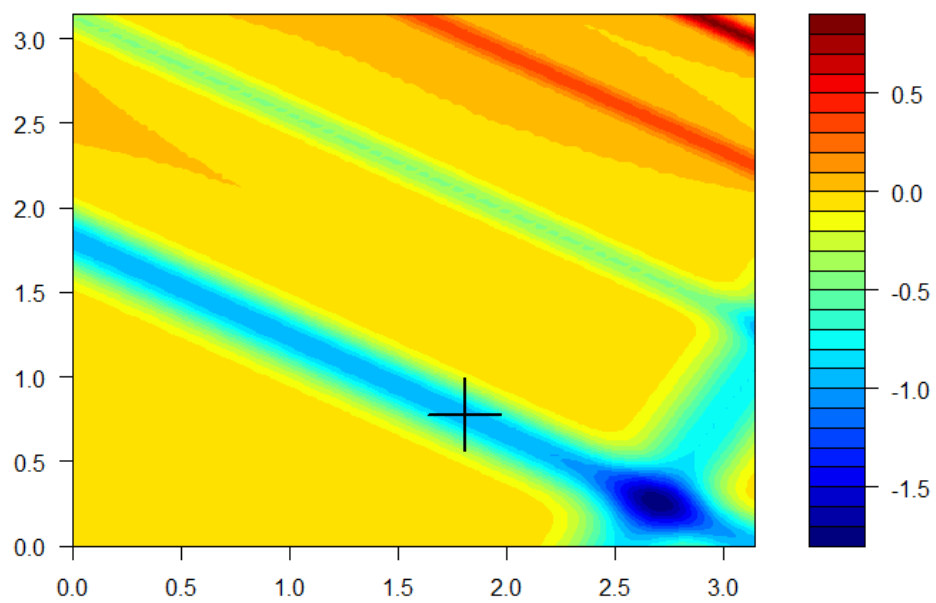
Rysunek 25: Wartość znalezione optimum w zależności od przyjętego elityzmu



Rysunek 26: Wartość znalezione optimum w zależności od rozmiarów populacji

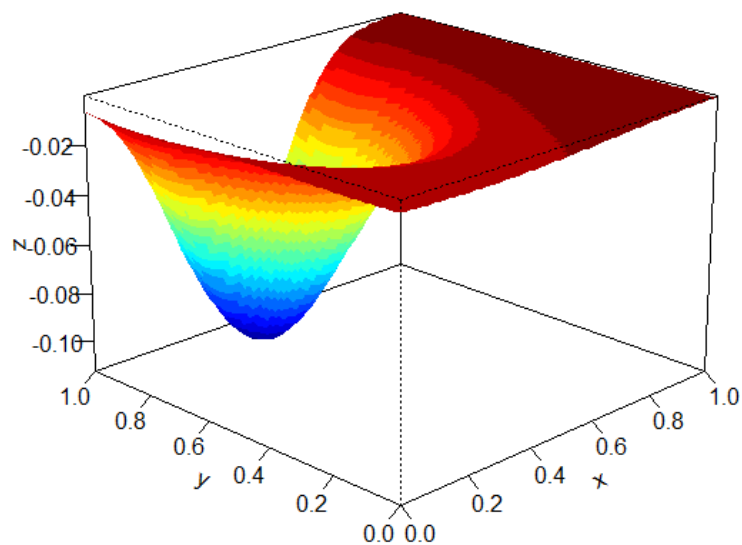


Rysunek 27: Wartość znalezionej optimum w zależności od ilości iteracji

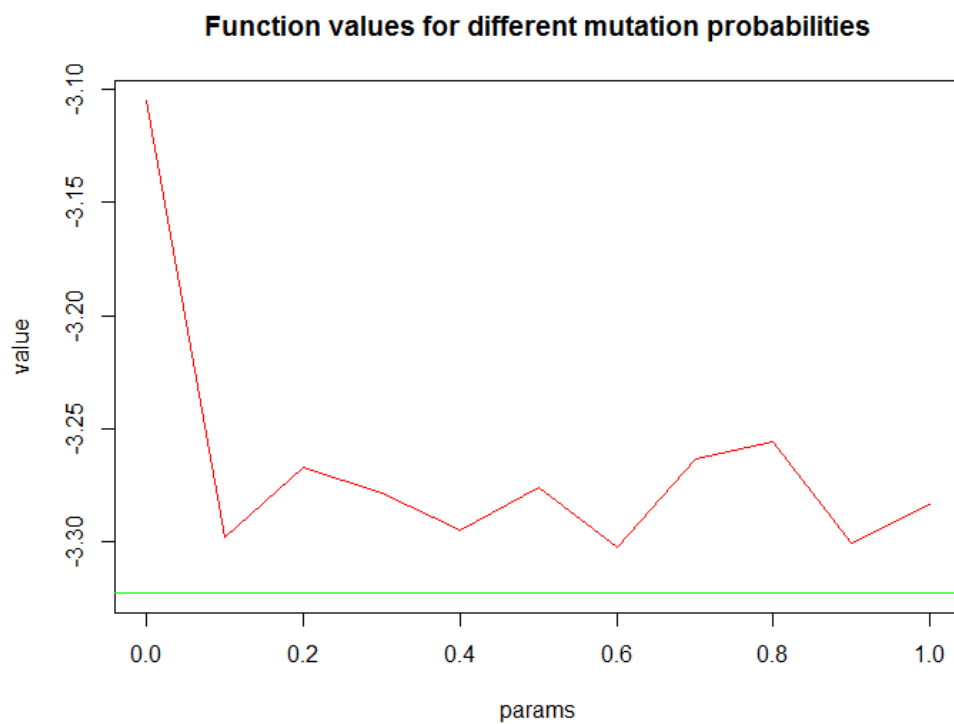


Rysunek 28: Poglądowa lokalizacja najlepszego znalezionej optimum

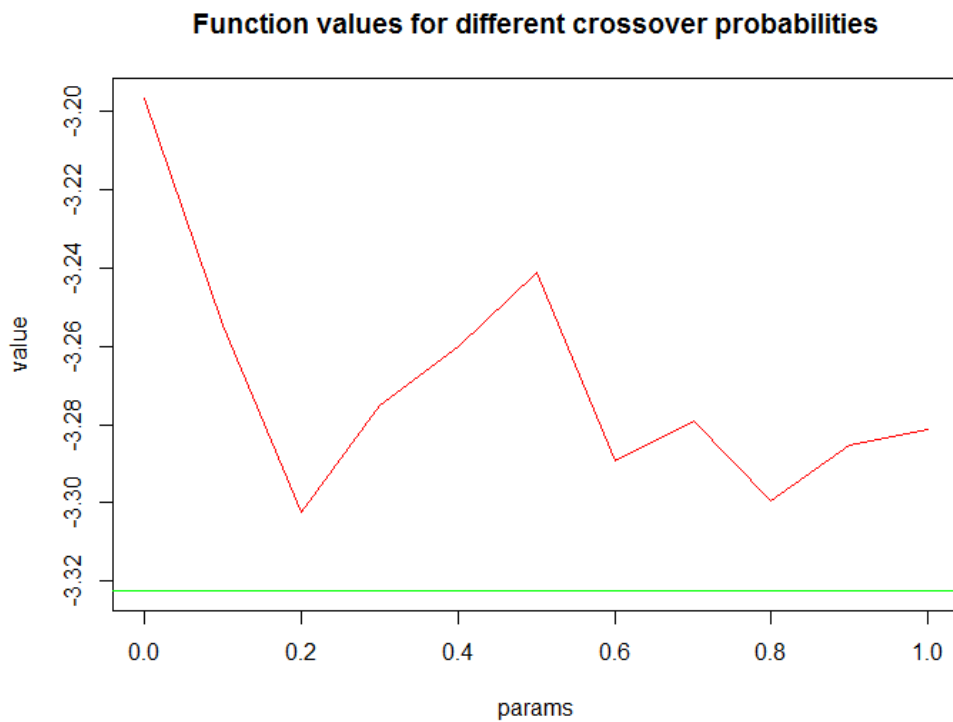
3.5 Hartman6 (6 parametrów)



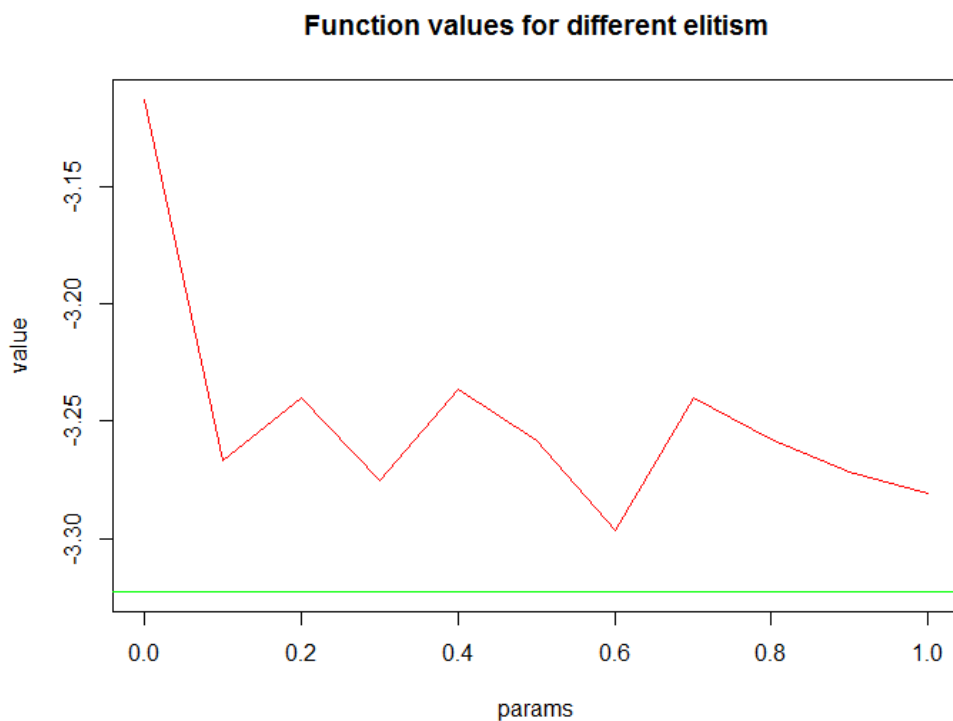
Rysunek 29: Wykres funkcji Hartman6 (d=6)



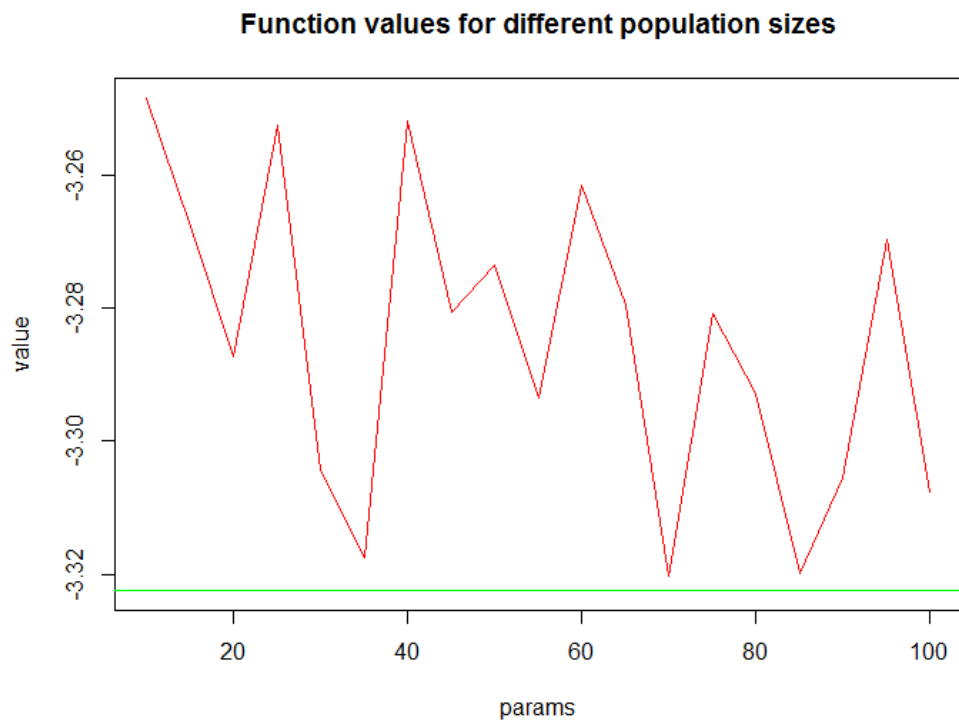
Rysunek 30: Wartość znalezionej optimum w zależności od prawdopodobieństwa mutacji



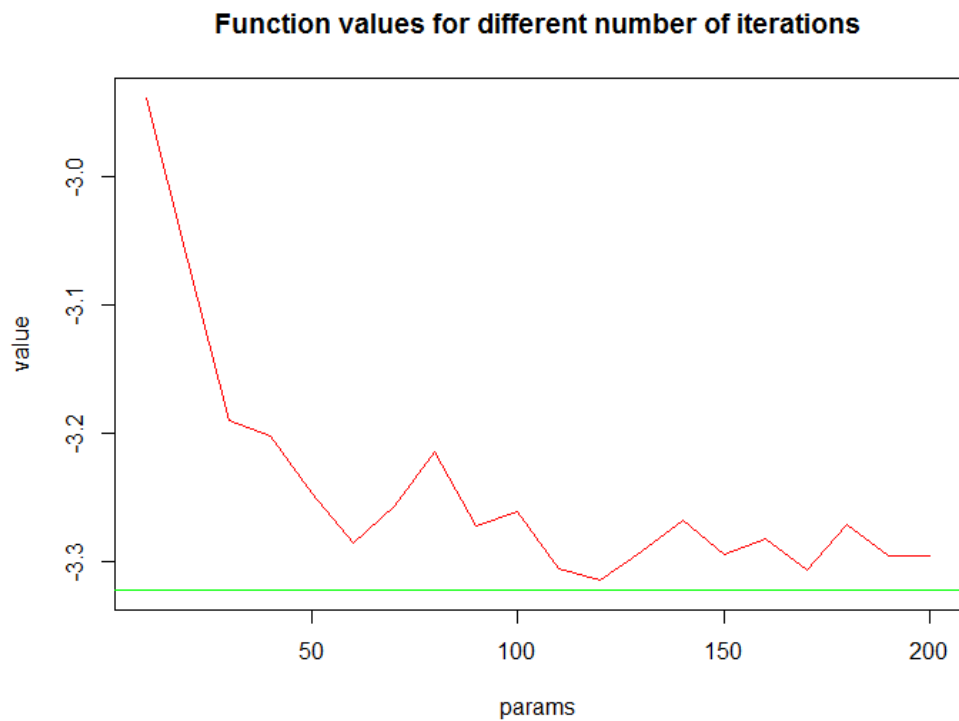
Rysunek 31: Wartość znalezionej optimum w zależności od prawdopodobieństwa krzyżowania



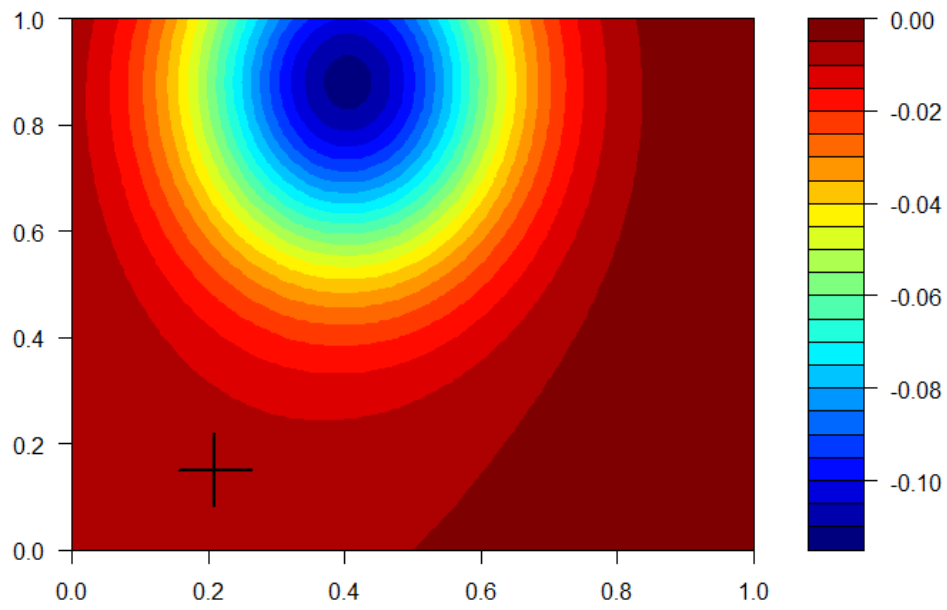
Rysunek 32: Wartość znalezionej optimum w zależności od przyjętego elityzmu



Rysunek 33: Wartość znalezione optimum w zależności od rozmiarów populacji

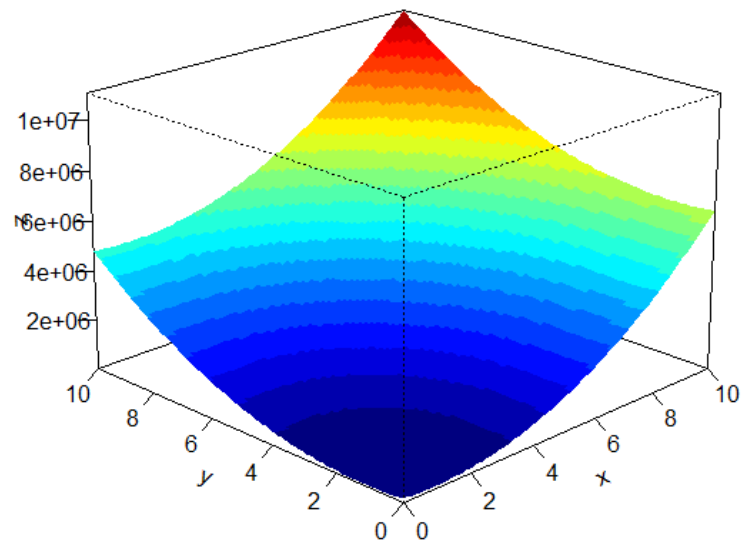


Rysunek 34: Wartość znalezione optimum w zależności od ilości iteracji

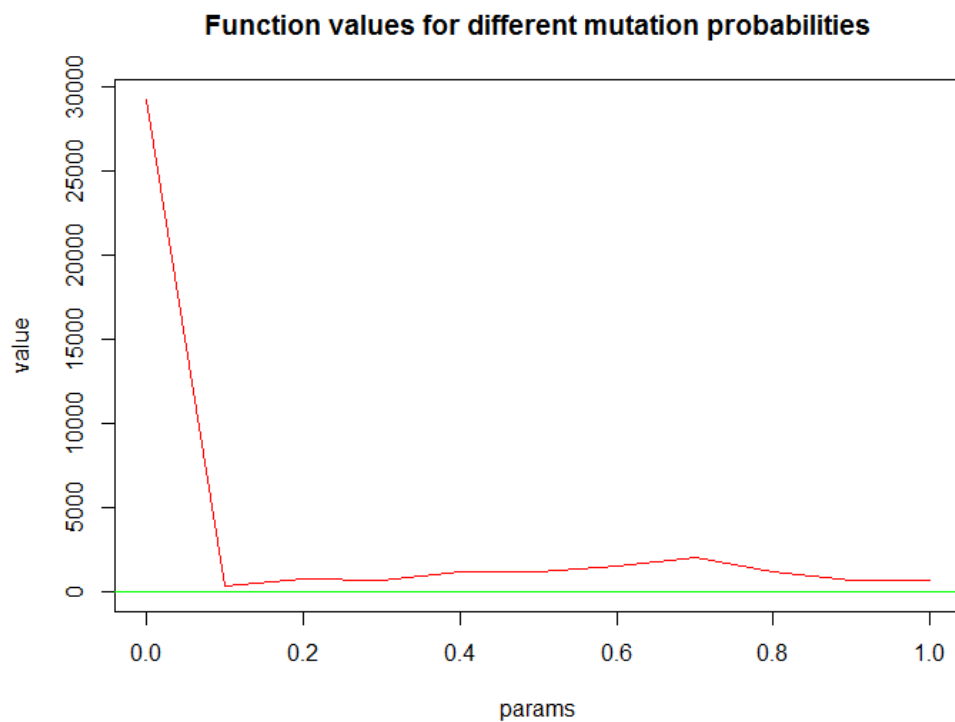


Rysunek 35: Poglądowa lokalizacja najlepszego znalezionej optimum

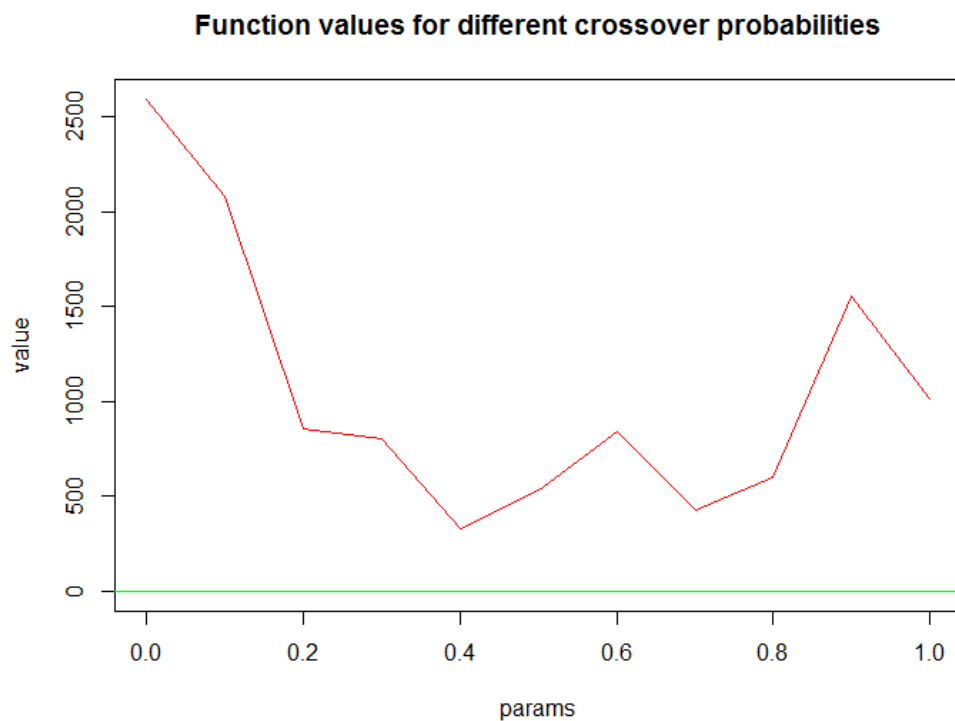
3.6 PriceTransistor (9 parametrów)



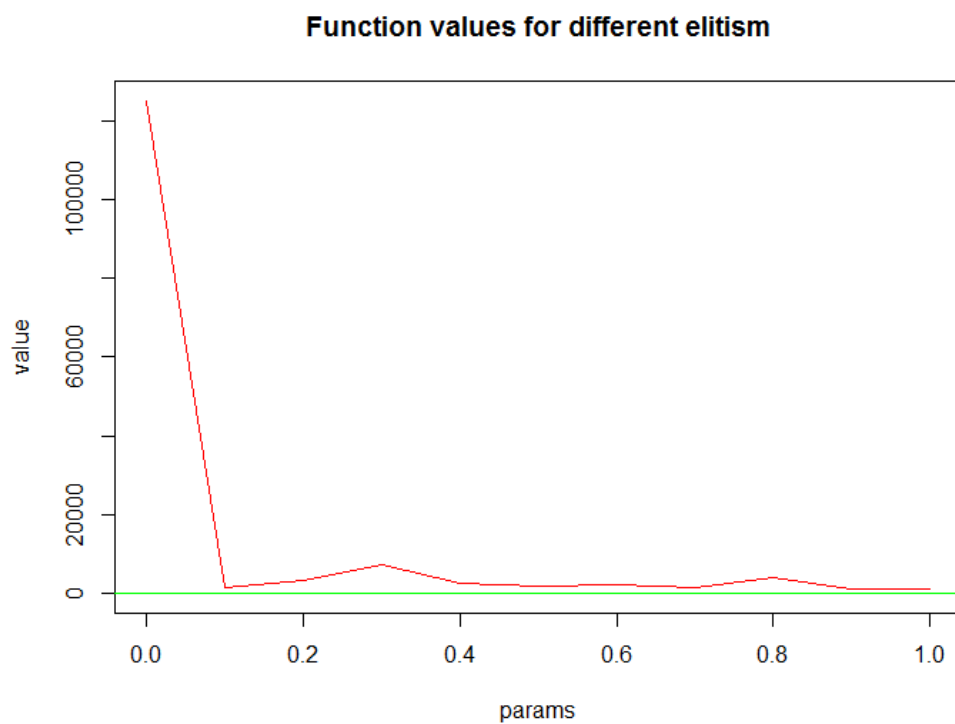
Rysunek 36: Wykres funkcji PriceTransistor ($d=9$)



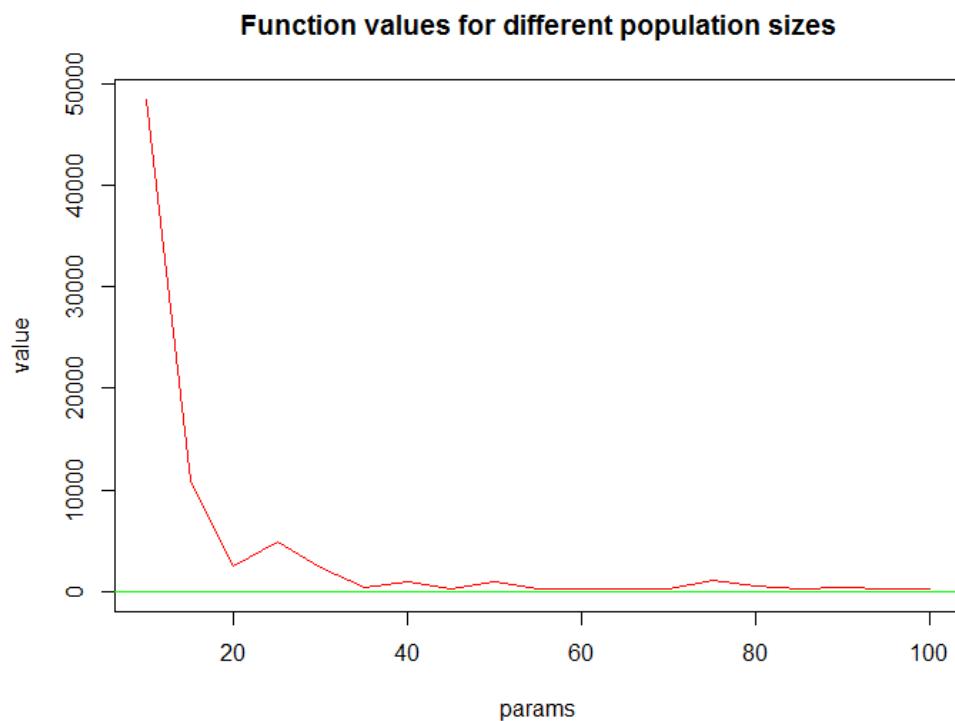
Rysunek 37: Wartość znalezionej optimum w zależności od prawdopodobieństwa mutacji



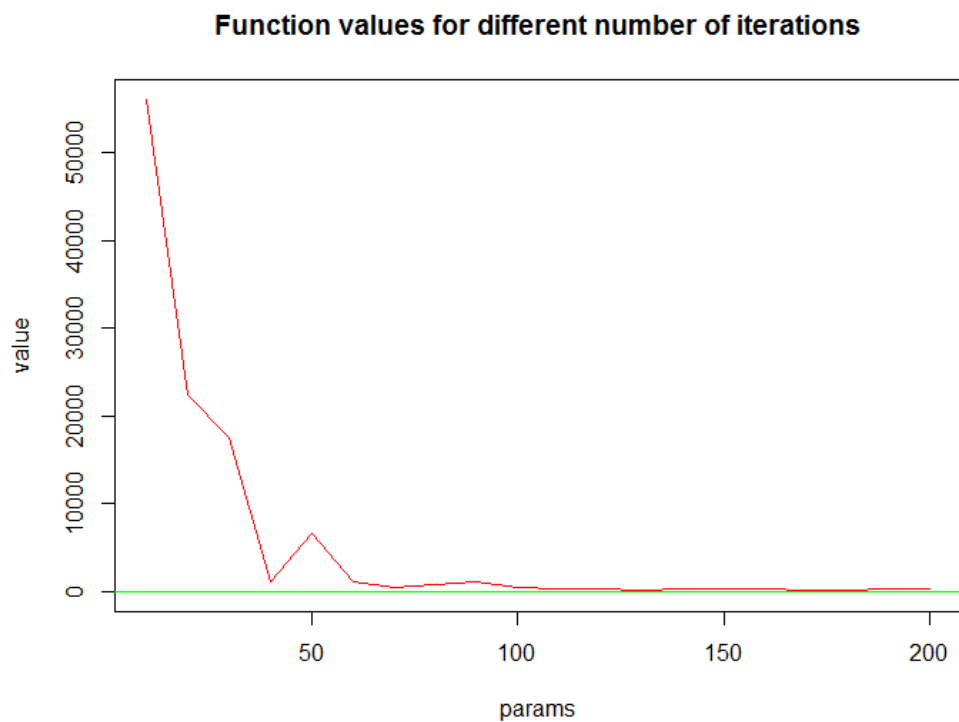
Rysunek 38: Wartość znalezionej optimum w zależności od prawdopodobieństwa krzyżowania



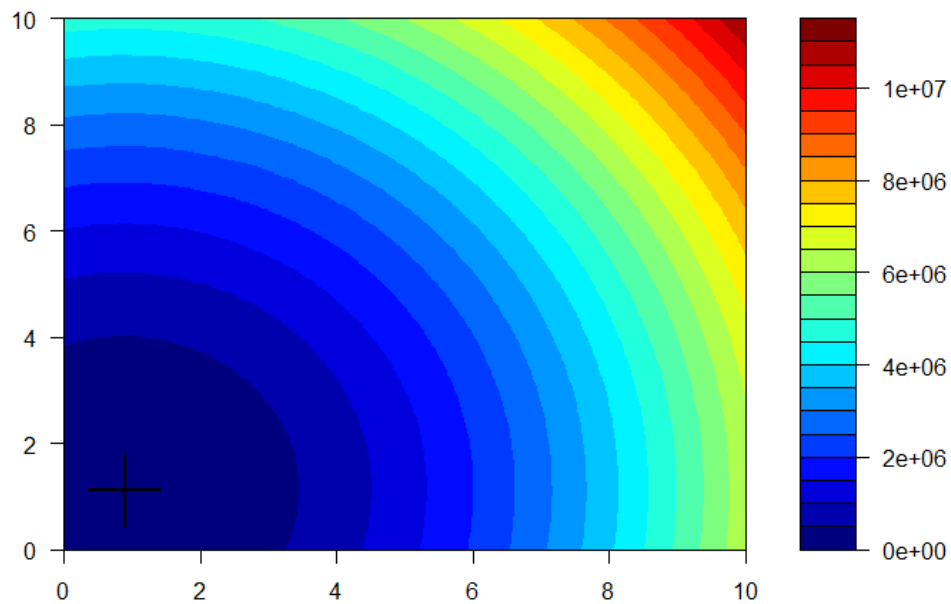
Rysunek 39: Wartość znalezionej optimum w zależności od przyjętego elityzmu



Rysunek 40: Wartość znalezionej optimum w zależności od rozmiarów populacji

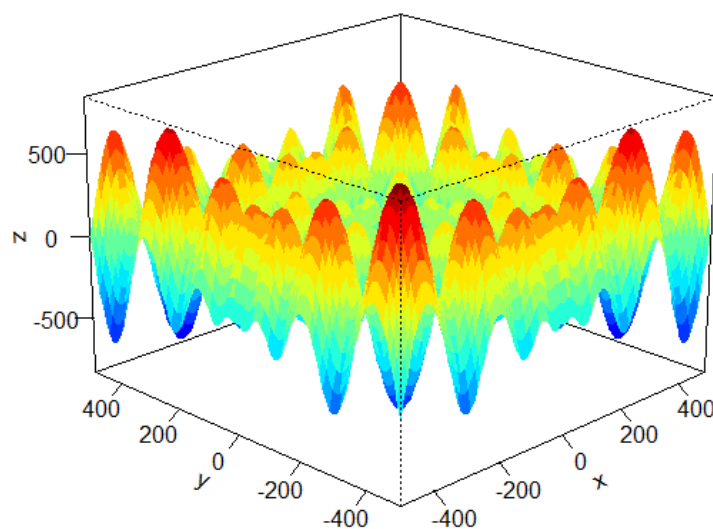


Rysunek 41: Wartość znalezionej optimum w zależności od ilości iteracji

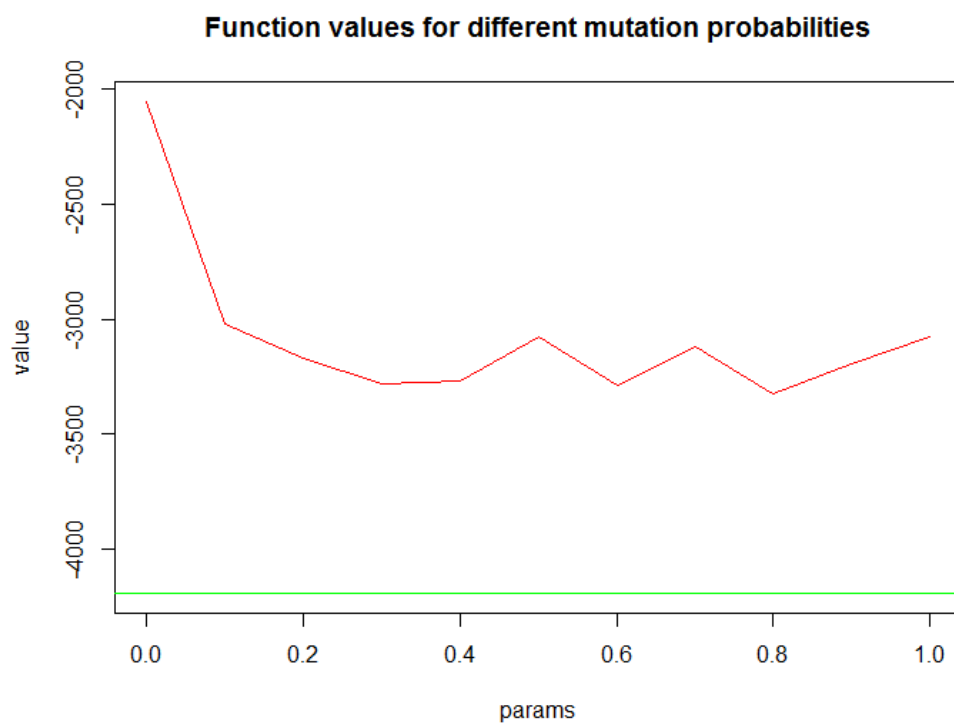


Rysunek 42: Poglądowa lokalizacja najlepszego znalezionej optimum

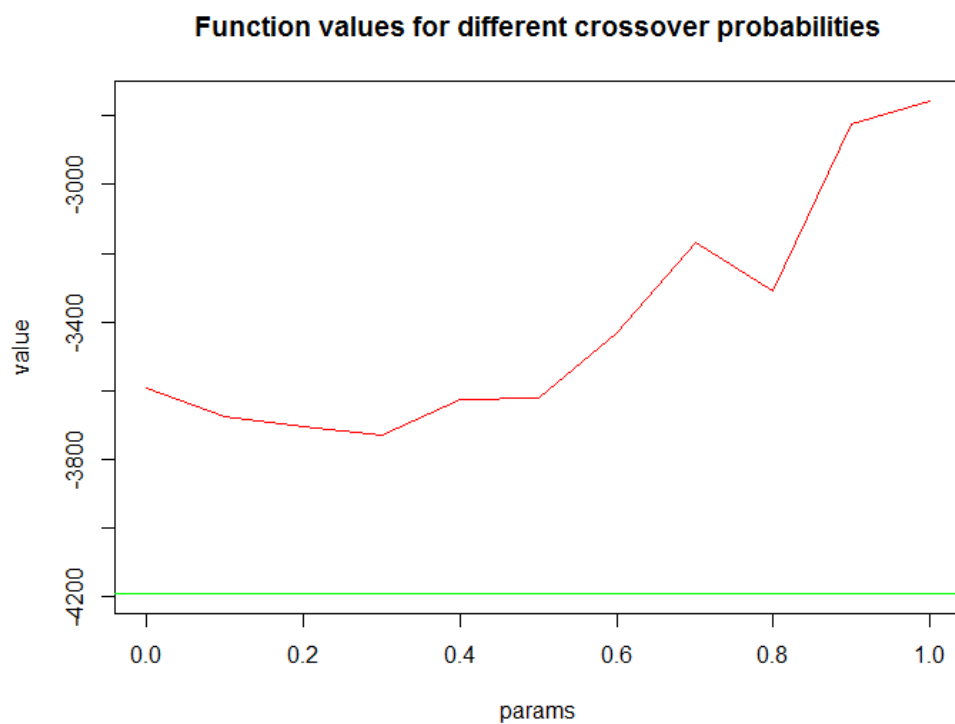
3.7 Schwefel (10 parametrów)



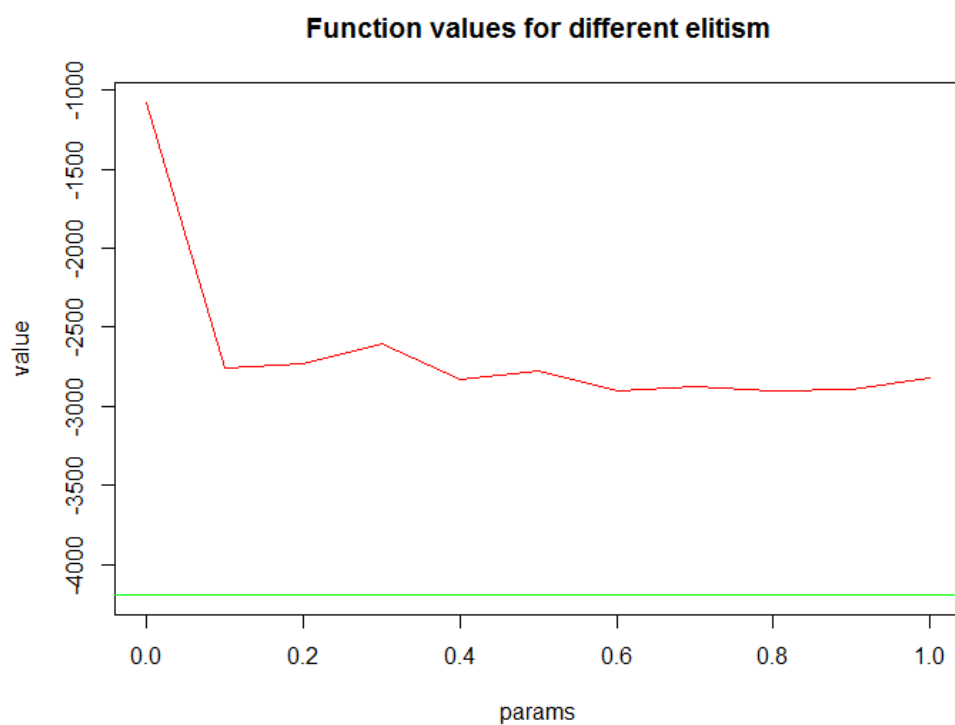
Rysunek 43: Wykres funkcji Schwefel ($d=10$)



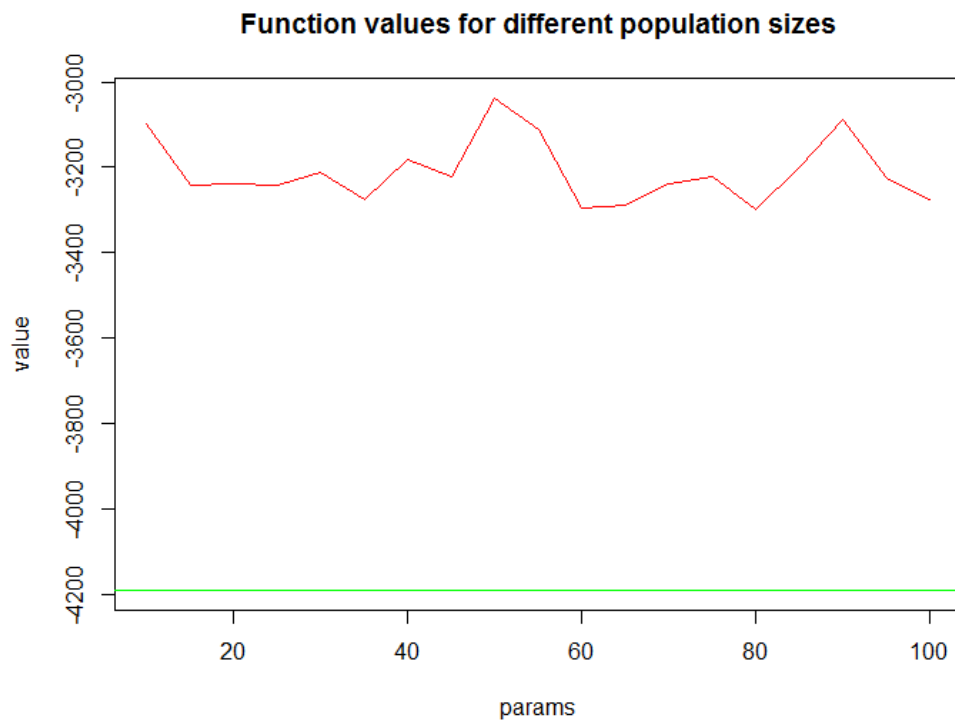
Rysunek 44: Wartość znalezionej optimum w zależności od prawdopodobieństwa mutacji



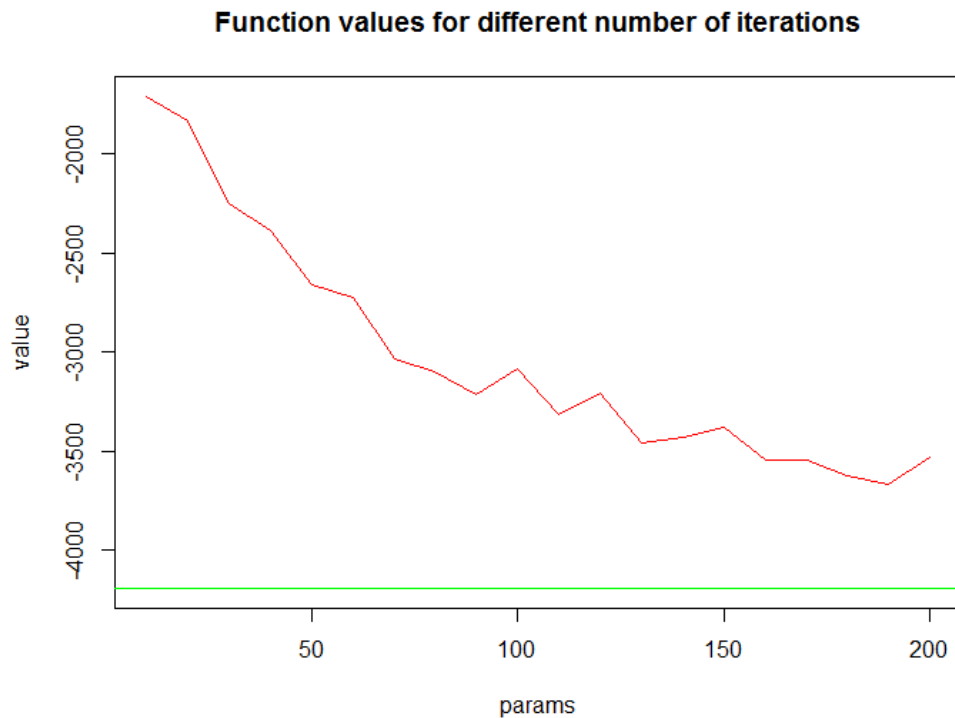
Rysunek 45: Wartość znalezionej optimum w zależności od prawdopodobieństwa krzyżowania



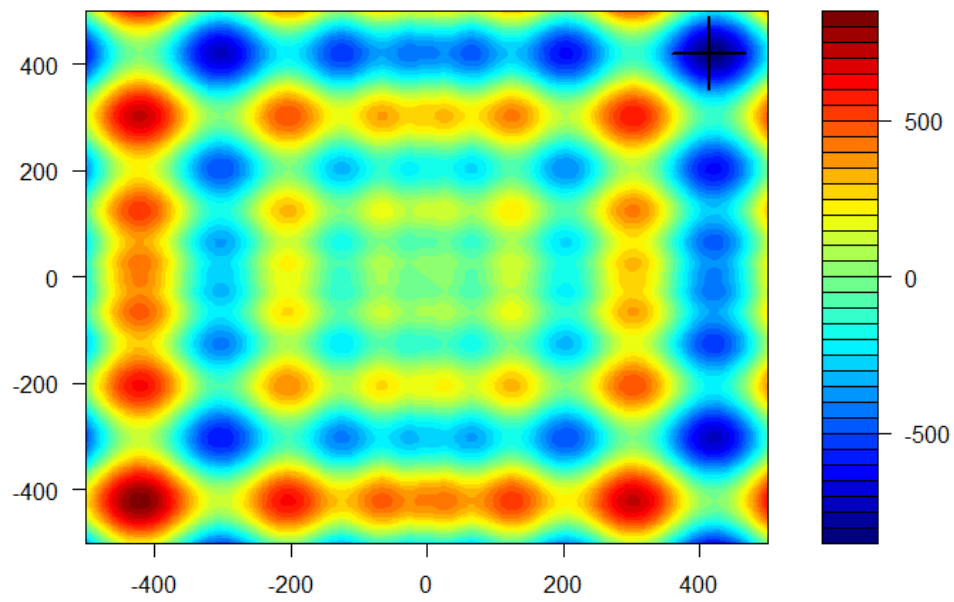
Rysunek 46: Wartość znalezionej optimum w zależności od przyjętego elityzmu



Rysunek 47: Wartość znalezione optimum w zależności od rozmiarów populacji

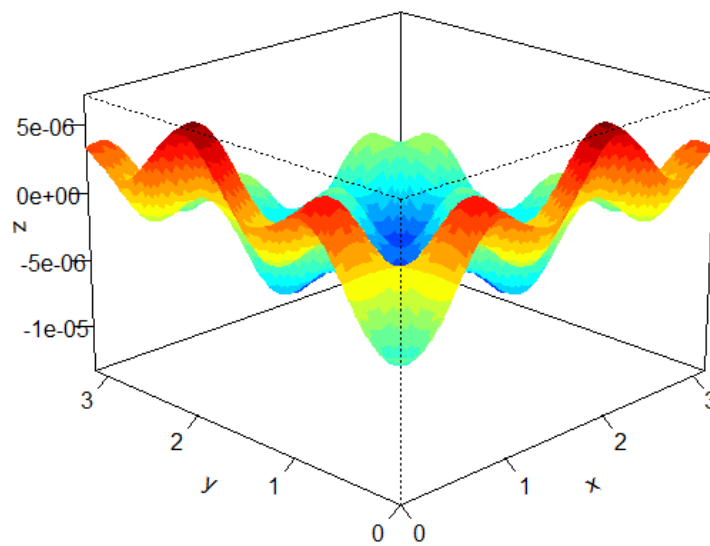


Rysunek 48: Wartość znalezione optimum w zależności od ilości iteracji

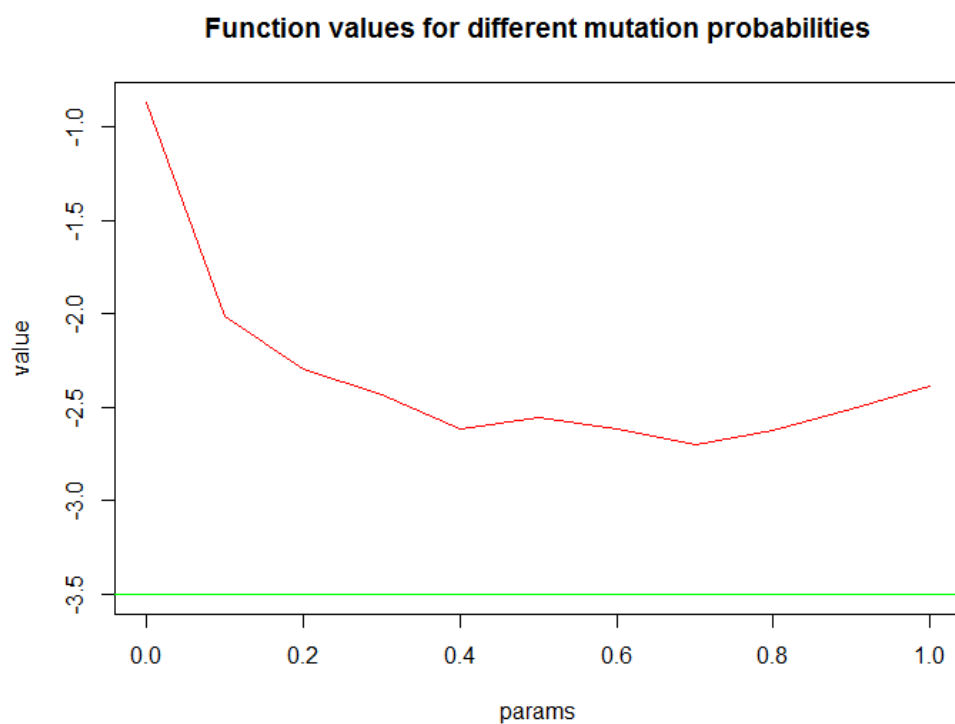


Rysunek 49: Poglądowa lokalizacja najlepszego znalezione optimum

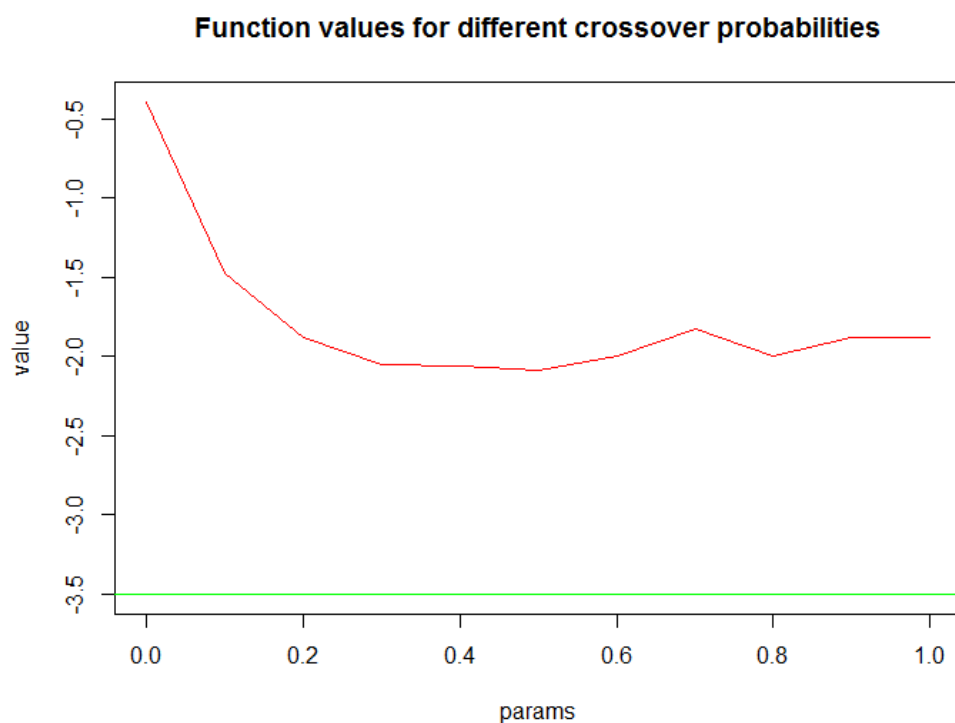
3.8 Zeldasine20 (20 parametrów)



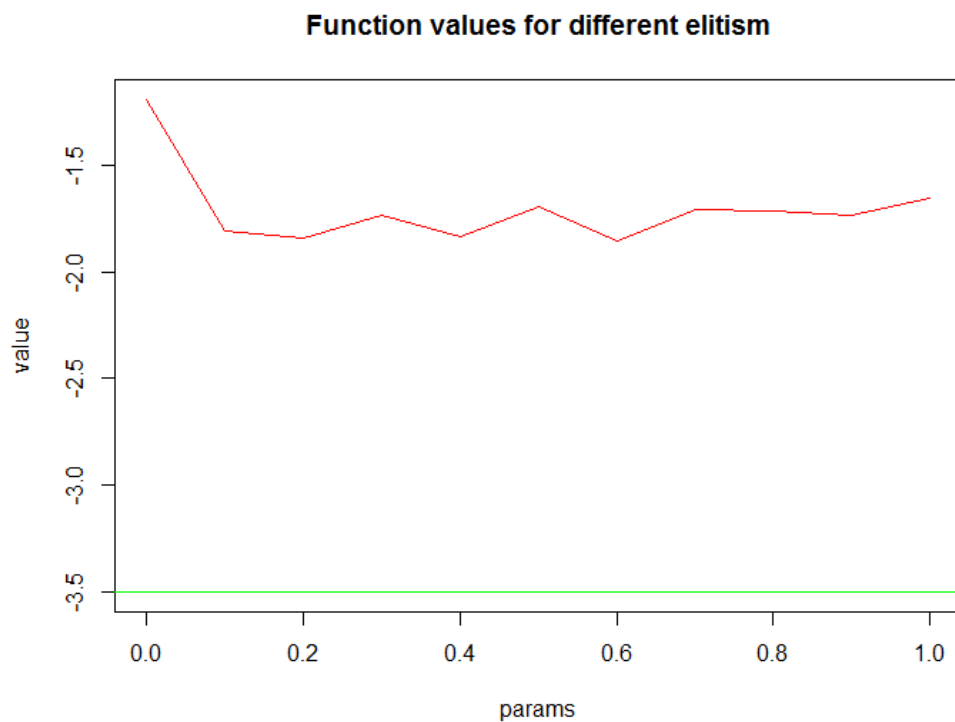
Rysunek 50: Wykres funkcji Zeldasine ($d=20$)



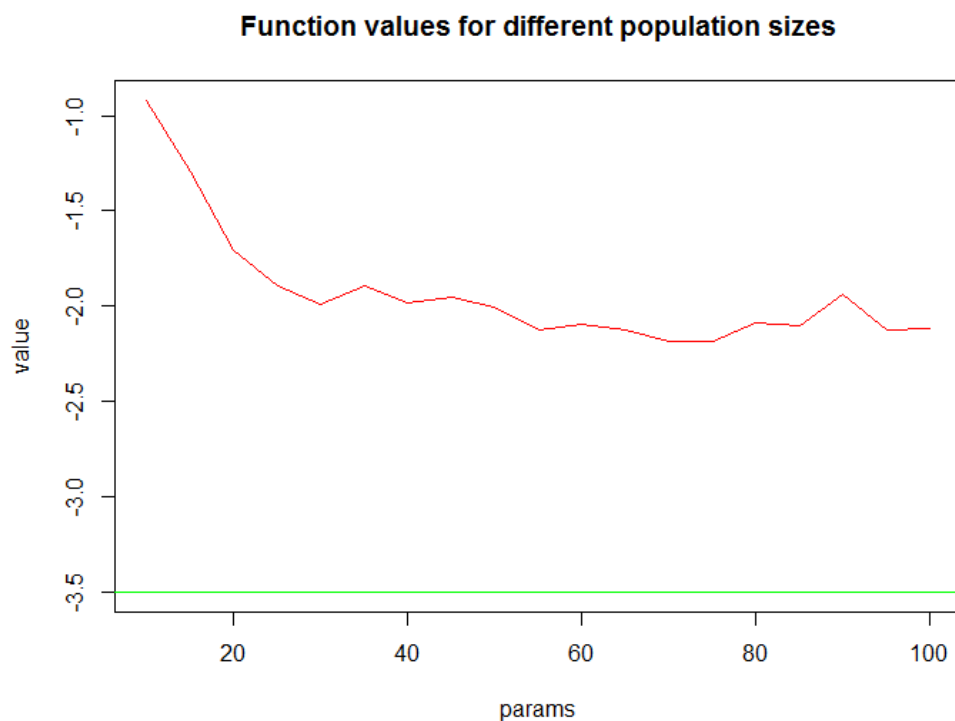
Rysunek 51: Wartość znalezionej optimum w zależności od prawdopodobieństwa mutacji



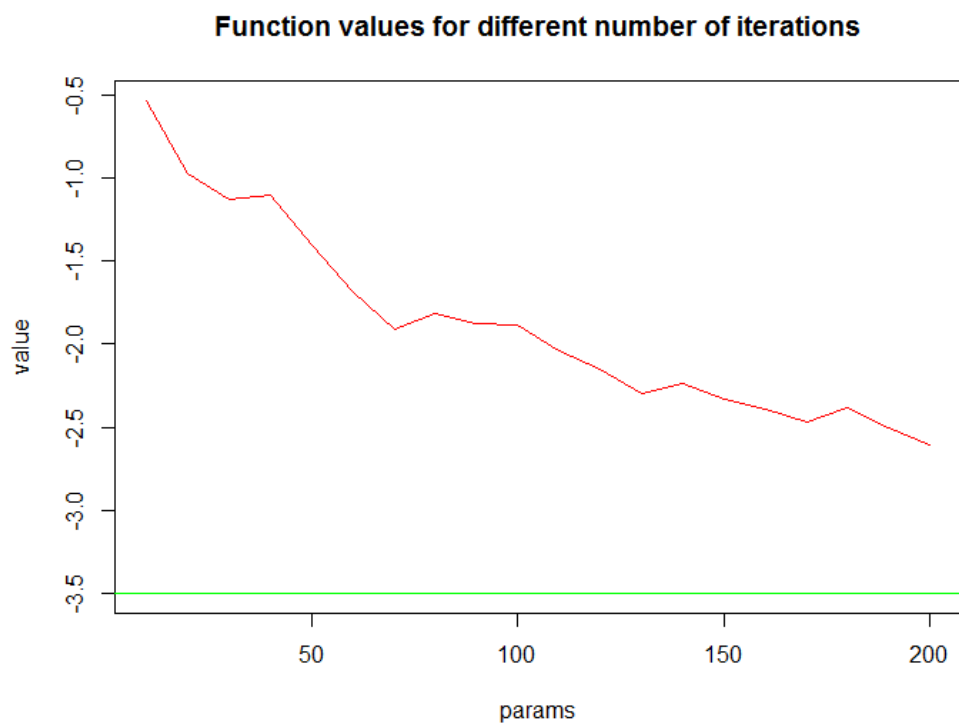
Rysunek 52: Wartość znalezionej optimum w zależności od prawdopodobieństwa krzyżowania



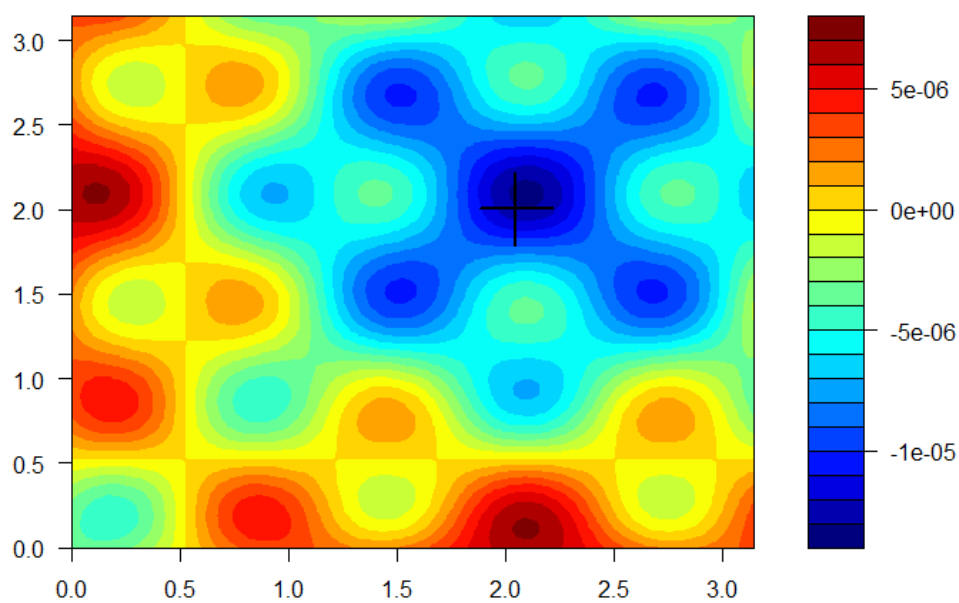
Rysunek 53: Wartość znalezionej optimum w zależności od przyjętego elityzmu



Rysunek 54: Wartość znalezionej optimum w zależności od rozmiarów populacji



Rysunek 55: Wartość znalezionej optimum w zależności od ilości iteracji



Rysunek 56: Poglądowa lokalizacja najlepszego znalezionej 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>