

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 . . . . .	5
<b>3</b>	<b>Przebieg badań</b>	<b>6</b>
3.1	Branin (2 parametry) . . . . .	6
3.2	Gulf (3 parametry) . . . . .	11
3.3	CosMix4 (4 parametry) . . . . .	16
3.4	EMichalewicz (5 parametrów) . . . . .	19
3.5	Hartman6 (6 parametrów) . . . . .	23
3.6	PriceTransistor (9 parametrów) . . . . .	26
3.7	Schwefel (10 parametrów) . . . . .	29
3.8	Zeldasine20 (20 parametrów) . . . . .	32
<b>4</b>	<b>Podsumowanie</b>	<b>35</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 dev.off(dev.list()["RStudioGD"])
4
5 require("GA")
6 require("globalOptTests")
7 require("rgl")
8
9 # Params ----
10
11 nOfRuns <- 10 # minimum 5
12
13 colors <- c("red", "blue", "orange")
14 series <- c("Seria 1", "Seria 2", "Seria 3")
15
16 baseParams = matrix(
17   c(0, 0, 50, 100, 1,
18     0.1, 0.8, 50, 100, 2,
19     0.1, 0.8, 25, 50, 3),
20   # [mutations,crossovers,populations,iterations,color]*
21   nrow=3, ncol=5, byrow = TRUE)
22
23 graphs <- TRUE
24 quality <- 100 #graph resolutions
25
26 mutationTests <- seq(0, 1, 0.1)
27 crossoverTests <- seq(0, 1, 0.1)
28 populationTests <- seq(10, 100, 5)
29 iterationTests <- seq(10, 200, 10)
30 elitismTests <- seq(0, 1, 0.1)
31
32 # Functions ----
33
34 functions <- c("Branin", "Gulf", "CosMix4", "EMichalewicz", "Hartman6",
35               "PriceTransistor", "Schwefel", "Zeldasine20")
36 for (abc in functions)
37 {
38   funcName <- abc
```

```

38
39 # for manual launch
40 #funcName <- "Branin" #2d
41 #funcName <- "Gulf" #3d
42 #funcName <- "CosMix4" #4d
43 #funcName <- "EMichalewicz" #5d
44 #funcName <- "Hartman6" #6d
45 #funcName <- "PriceTransistor" #9d
46 #funcName <- "Schwefel" #10d
47 #funcName <- "Zeldasine20" #20d
48
49 # Processing ----
50
51 customMeasure <- function(fileName, graphName, values, mType, xlab, main) {
52
53   gMin <- .Machine$integer.max
54   gBest <- NA
55
56   temp <- c()
57   for (defRow in 1:nrow(baseParams)) {
58     averages <- c()
59     for (value in values) {
60       sum <- 0
61       for (i in 1:nOfRuns) {
62         GAmin <- ga(type = "real-valued",
63           fitness = function(xx) -f(xx),
64           min = c(B[1,]), max = c(B[2,]),
65           popSize = if (mType == "pop") value else baseParams[defRow,3],
66           maxiter = if (mType == "itr") value else baseParams[defRow,4],
67           pmutation = if (mType == "mut") value else baseParams[defRow,1],
68           pcrossover = if (mType == "crs") value else baseParams[defRow,2],
69           elitism = if (mType == "elt") value else max(1,
             round(baseParams[defRow,3] * 0.05)))
70         solution <- matrix(unlist(GAmin@solution),ncol=dim,byrow=TRUE)
71         eval <- f(solution[1,])
72         if (eval < gMin) {
73           gMin <- eval
74           gBest <- GAmin
75         }
76         sum <- sum + eval
77       }
78       averages <- c(averages, (sum / nOfRuns))
79     }
80     temp <- c(temp, averages)
81   }
82   result <- matrix(c(temp),nrow = nrow(baseParams),ncol = length(values))
83   write.table(result, file = fileName, row.names=FALSE,
84     na="", col.names=FALSE, sep=";")
85
86   if (graphs) {
87
88     png(file = paste(funcName, graphName, ".png", sep=""), width=600,
89       height=400, units="px")
90
91     plot(0, 0, main=main,
92       ylim=c(min(c(temp,globalOpt)),max(c(temp,globalOpt))),

```

```

92     xlim=c(min(values),max(values)),
93     type="n", xlab=xlab, ylab="wartosc")
94 abline(globalOpt,0, col="green")
95 colorNames <- c()
96 seriesNames <- c()
97 for (i in 1:nrow(baseParams)) {
98     color <- colors[baseParams[i,5]]
99     colorNames <- c(colorNames, color)
100     seriesNames <- c(seriesNames, series[baseParams[i,5]])
101     lines(values, result[i,], col = color, type = 'l')
102 }
103 legend("topright", seriesNames, lty=rep(1,nrow(baseParams)), col=colorNames)
104
105 dev.off()
106
107 summary(gBest)
108
109 png(file = paste(funcName, graphName, mType, ".png", sep=""), width=600,
110     height=400, units="px")
111 filled.contour(x, y, z, color.palette = jet.colors, nlevels = 24,
112     plot.axes = { axis(1); axis(2);
113     points(solution[1,1], solution[1,2],
114         pch = 3, cex = 5, col = "black", lwd = 2)
115     }
116 )
117 dev.off()
118
119 png(file = paste(funcName, graphName, mType, "fitness", ".png", sep=""),
120     width=600, height=400, units="px")
121 plot(gBest)
122 dev.off()
123 }
124
125 dim <- getProblemDimen(funcName)
126 B <- matrix(unlist(getDefaultBounds(funcName)),ncol=dim,byrow=TRUE)
127 f <- function(xx) goTest(par=c(xx, rep(0, dim-length(xx))),
128     fnName=funcName, checkDim = TRUE)
129 globalOpt <- getGlobalOpt(funcName)
130
131 if (graphs) {
132
133     xprobes <- abs(B[2,1] - B[1,1]) / quality
134     yprobes <- abs(B[2,2] - B[1,2]) / quality
135     x <- seq(B[1,1], B[2,1], by = xprobes)
136     y <- seq(B[1,2], B[2,2], by = yprobes)
137     z <- outer(x, y, Vectorize(function(x,y) f(c(x,y))))
138     nbcol = 100
139     color = rev(rainbow(nbcol, start = 0/6, end = 4/6))
140     zcol = cut(z, nbcol)
141     persp3d(x, y, z, theta=50, phi=25, expand=0.75, col=color[zcol],
142         ticktype="detailed",axes=TRUE)
143
144     png(file = paste(funcName, "1.png", sep=""), width=600, height=400,
145         units="px")

```

```

145 | persp3D(x, y, z, theta = -45, phi = 20, color.palette = jet.colors)
146 | dev.off()
147 |
148 | }
149 |
150 | customMeasure("resultsMutations.csv", "2", mutationTests, "mut", "p. mutacji",
151 |               "Znalezienie minimum dla roznych prawdopodobienstw mutacji")
152 |
153 | customMeasure("resultsCrossover.csv", "3", crossoverTests, "crs", "p.
154 |               krzyzowania",
155 |               "Znalezienie minimum dla roznych prawdopodobienstw krzyzowania")
156 | customMeasure("resultsPopulation.csv", "4", populationTests, "pop", "rozmiar
157 |               populacji",
158 |               "Znalezienie minimum dla roznych rozmiarow populacji")
159 | customMeasure("resultsIterations.csv", "5", iterationTests, "itr", "ilosc
160 |               iteracji",
161 |               "Znalezienie minimum dla roznych ilosci iteracji")
162 | customMeasure("resultsElitism.csv", "6", elitismTests, "elt", "elityzm",
163 |               "Znalezienie minimum dla roznych wartosci elityzmu")
164 |
165 | }

```

## 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”. [todo: dopisać o pętli przechodzącej po wszystkich funkcjach oraz po wszystkich parametrach domyślnych]

### 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:

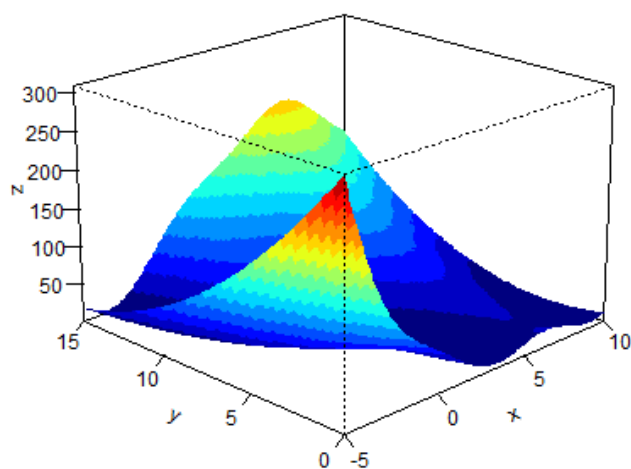
-	Parametry domyślne 1	Parametry domyślne 2	Parametry domyślne 3
Rozmiar populacji	50	50	25
Rozmiar iteracji	100	100	50
Prawdopodobieństwo mutacji	0	0.1	0.1
Prawdopodobieństwo krzyżowania	0	0.8	0.8

- 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). [todo: opisać dziedzinę]

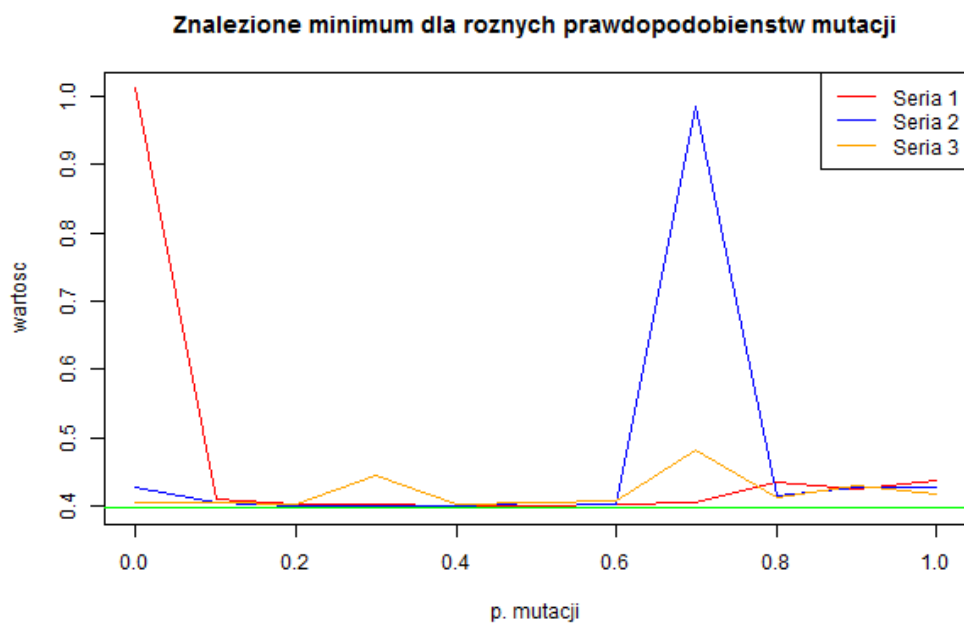
$$f(\mathbf{x}) = a(x_2 - bx_1^2 + cx_1 - r)^2 + s(1 - t) \cos(x_1) + s \quad (1)$$



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

Powyższy wykres pokazuje trójwymiarowy obraz funkcji Branin. [todo: z którego wynika ...]

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



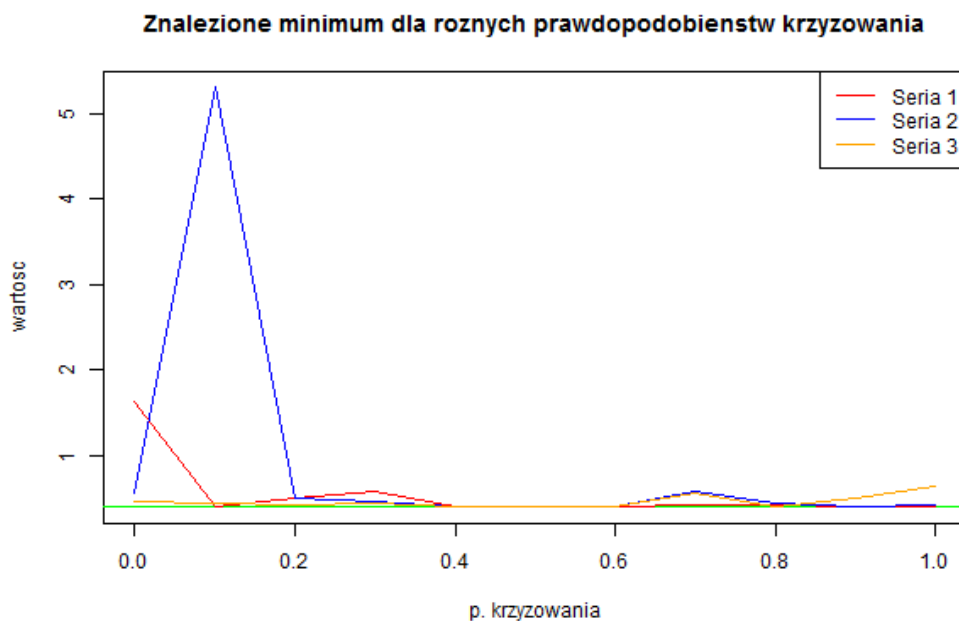
Rysunek 2: Wartość znalezionej optimum w zależności od prawdopodobieństwa mutacji

Na powyższym wykresie można zauważyć niski wpływ mutacji na znalezione rozwiąza-



nia. Przy wszystkich parametrach domyślnych funkcja znajduje się w pobliżu optymalnej wartości.

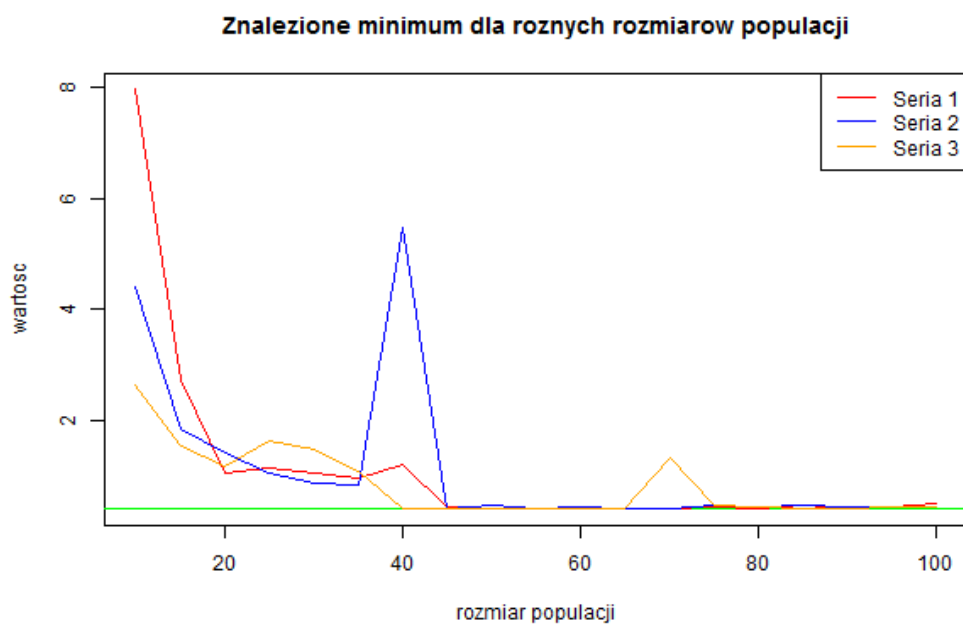
Wyjątkiem stanowi tutaj "seria 2" reprezentująca drugi zestaw wartości domyślnych. Przy mutacji wynoszącej 0.7 wynik funkcji znacząco się pogorszył.



Rysunek 3: Wartość znalezionego optimum w zależności od prawdopodobieństwa krzyżowania

Wykres przyjmuje wartości zbliżone do oczekiwanych, gdy prawdopodobieństwo krzyżowania wynosi 0.2 lub więcej.

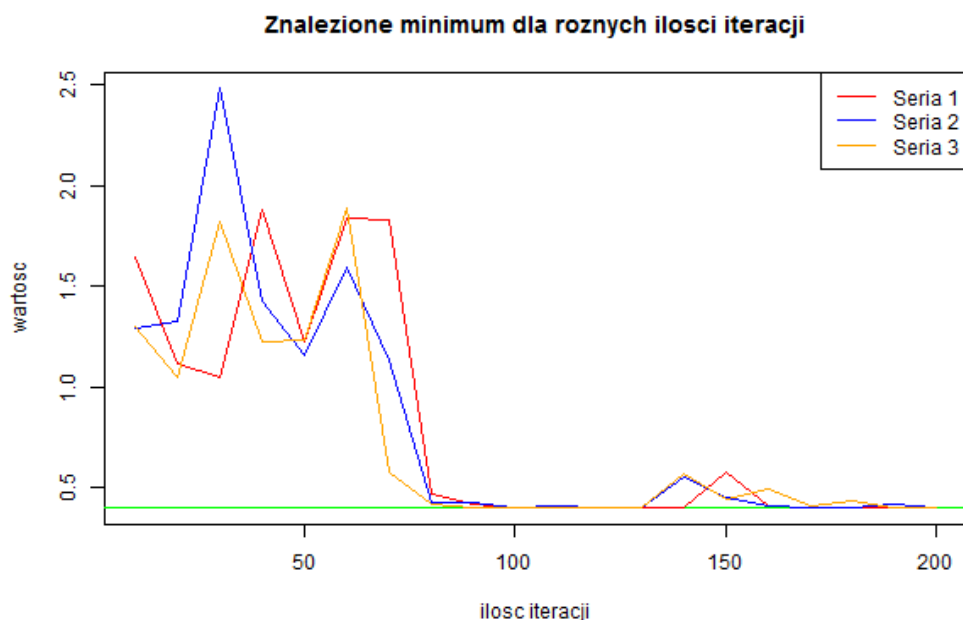
Najlepszy rezultat został otrzymany w przedziale prawdopodobieństwa krzyżowania między 0.4 a 0.6.



Rysunek 4: Wartość znalezionego optimum w zależności od rozmiarów populacji

Z wykresu można odczytać podatność funkcji w zależności od populacji. Wyniki zbliżone do oczekiwanych zostały uzyskane dla populacji wynoszącej 45 jednostek.

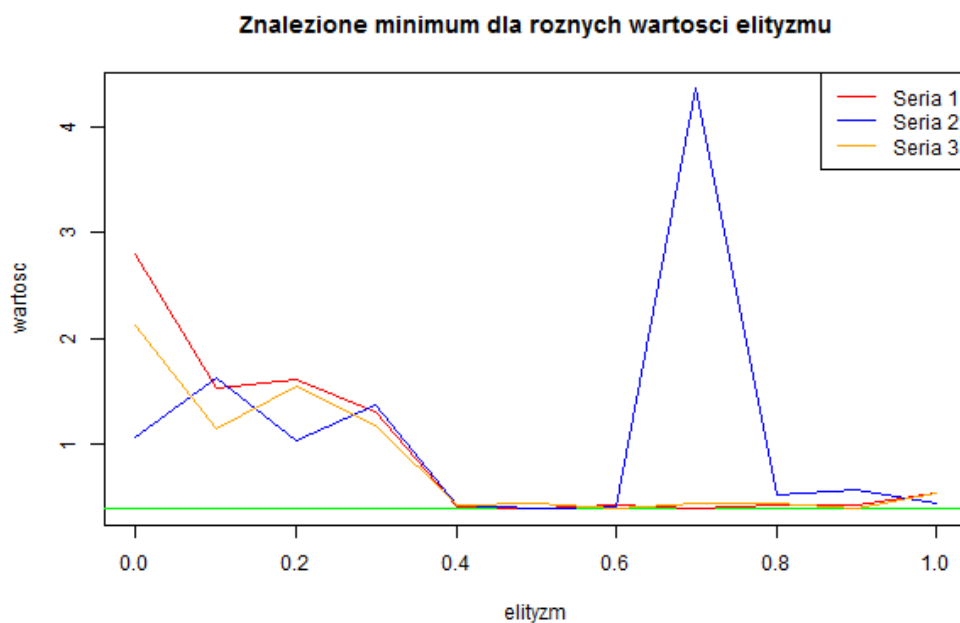
Zauważalny jest wzrost jakości rozwiązania wraz ze wzrostem ilości jednostek populacji.



Rysunek 5: Wartość znalezionego optimum w zależności od ilości iteracji

Wykres wskazuje wyraźną zmianę jakości rozwiązań dla 75 i więcej iteracji. Poniżej tej wartości uzyskiwane wyniki są niestabilne, powyżej osiągają wartość zbliżoną do

oczekiwanej.



Rysunek 6: Wartość znalezione optimum w zależności od przyjętego elityzmu

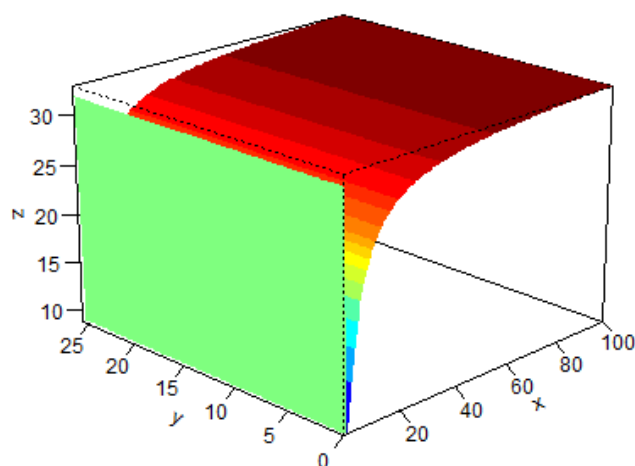
Z wykonanych badań wynika, że do uzyskania optymalnego rozwiązania należy zastosować wartość elityzmu na poziomie przynajmniej 0.4. Jego ustawienie poniżej tej wartości powoduje znaczące obniżenie się jakości rozwiązania.

Warto tutaj zauważyć ponowne (jak w przypadku mutacji) obniżenie się jakości wyniku dla przedziału 0.6-0.8.

### 3.2 Gulf (3 parametry)

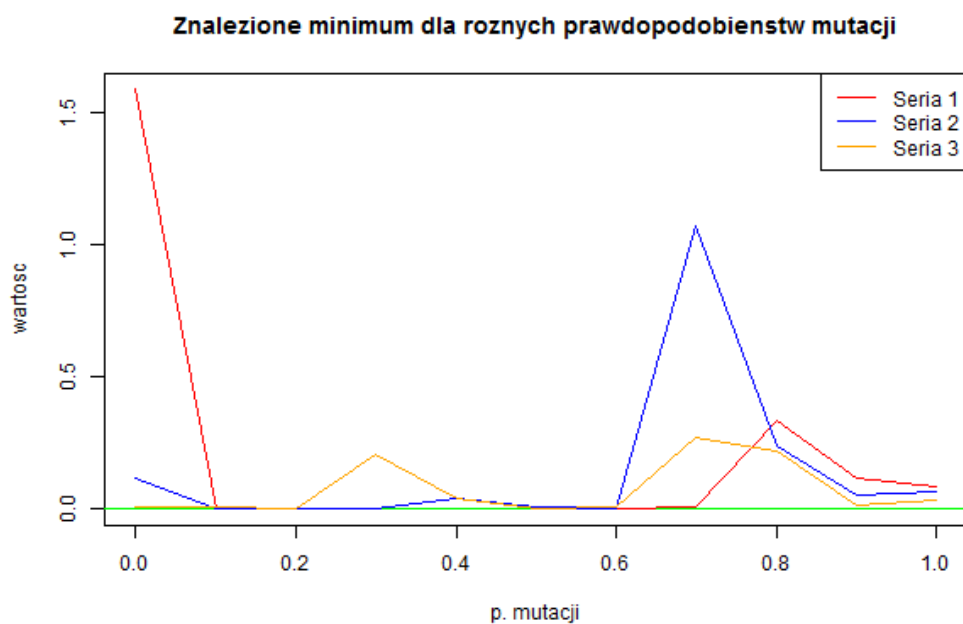
Gulf jest funkcją przyjmującą trzy parametry. Na ilustracji (rys. 7) przedstawiono jej wykres dla pierwszych dwóch wymiarów.

[todo: dodać wzór, dziedzinę funkcji]



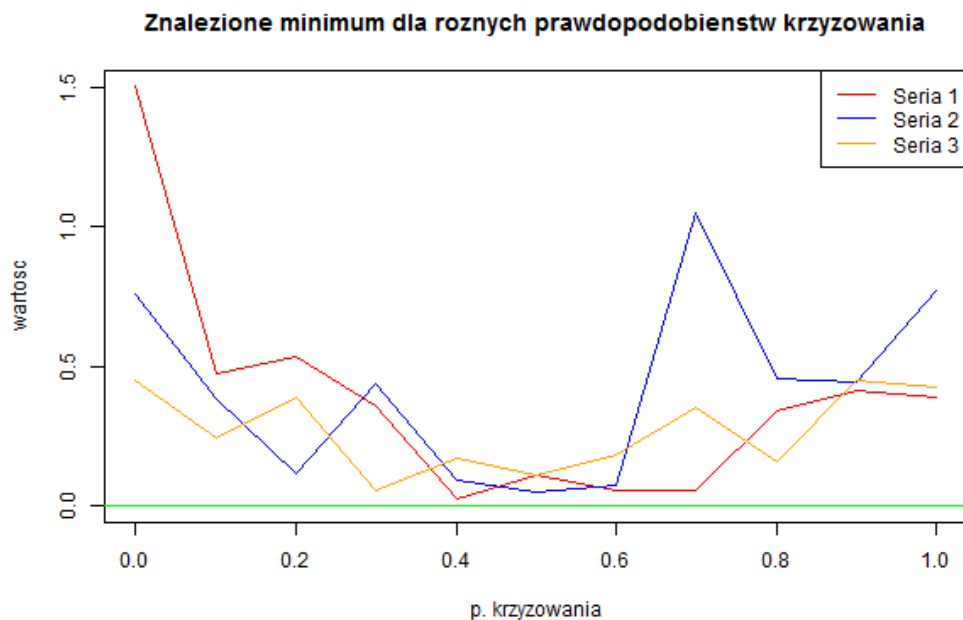
Rysunek 7: Wykres funkcji Gulf ( $d=3$ )

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



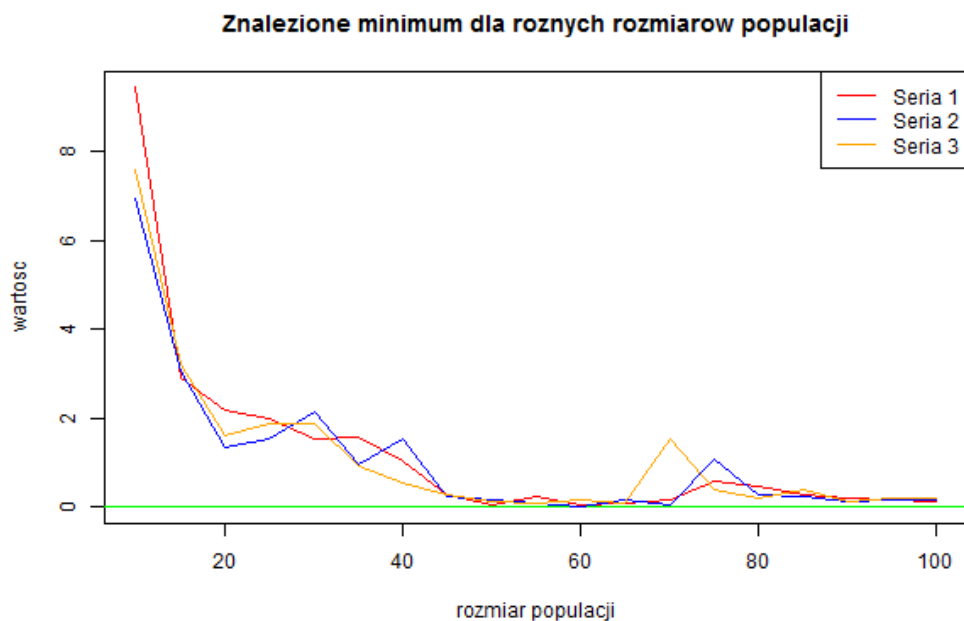
Rysunek 8: Wartość znalezione optimum w zależności od prawdopodobieństwa mutacji

Wartości funkcji Gulf dla zadanego prawdopodobieństwa mutacji są zbliżone do wartości oczekiwanej w przedziale 0.1-0.6. Powyżej tego przedziału mutacja wywiera negatywny wpływ na otrzymywane wyniki.



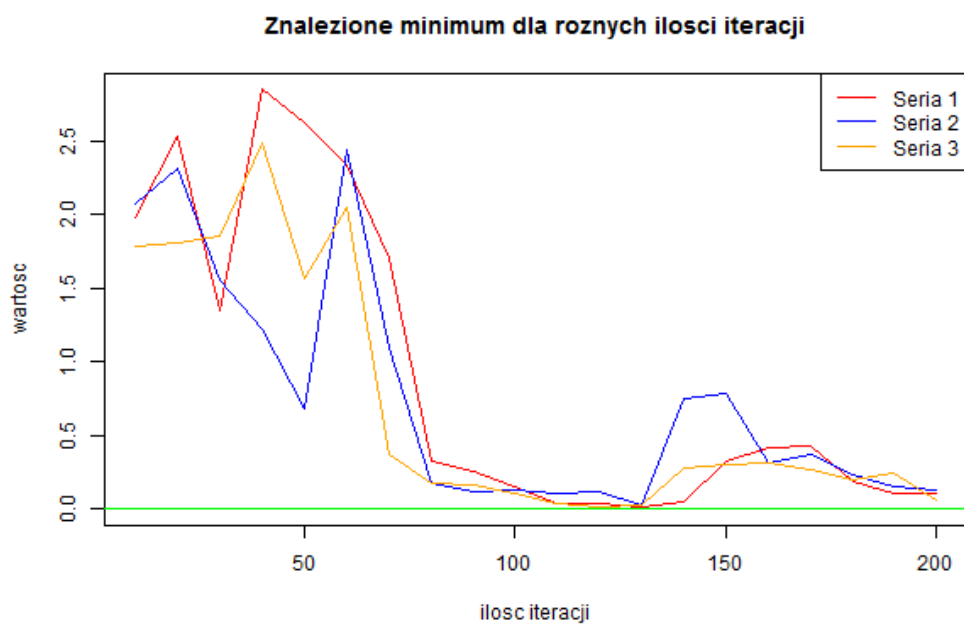
Rysunek 9: Wartość znalezione optimum w zależności od prawdopodobieństwa krzyżowania

Prawdopodobieństwo krzyżowania ma niski oraz niestabilny wpływ na otrzymane wyniki. Wspólnie (dla wszystkich ustawień domyślnych) najlepsze wyniki uzyskane zostały w przedziale 0.4-0.6. Przyjęcie wartości krzyżowania wykraczających poza wskazany przedział znacząco obniża jakość uzyskanych wyników.



Rysunek 10: Wartość znalezione optimum w zależności od rozmiarów populacji

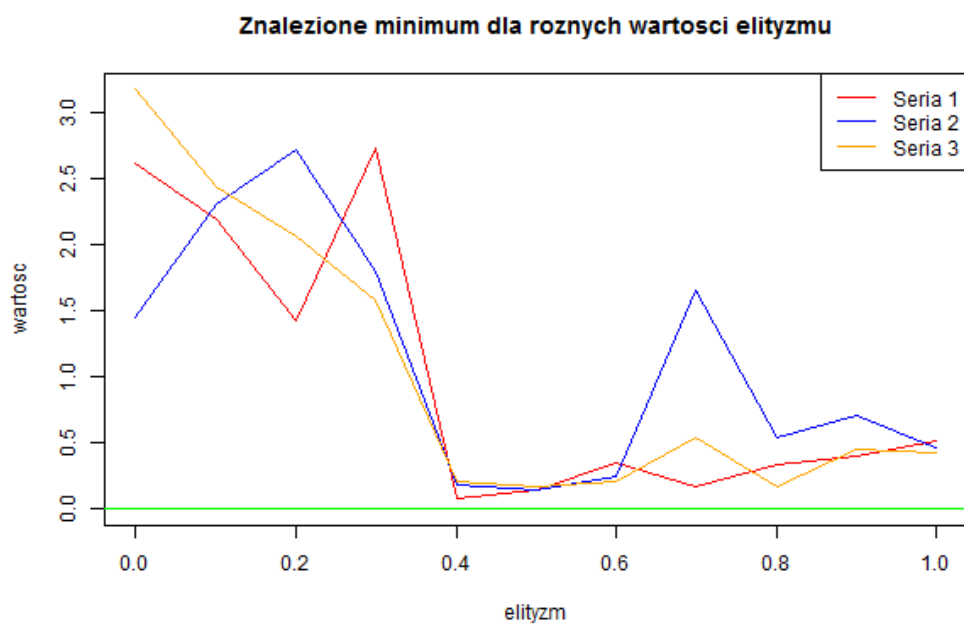
Wykres ten wyraźnie obrazuje pozytywny wpływ zwiększenia populacji na jakość wyników. Najlepsze wyniki uzyskano dla populacji wynoszącej przynajmniej 50 jednostek. Zauważalne jest również pogorszenie wyników w przedziale 65-80[*todo: dlaczego*].



Rysunek 11: Wartość znalezionej optimum w zależności od ilości iteracji

Na wykresie można zauważyć znaczące poprawienie się rezultatów, gdy ilość iteracji wynosi przynajmniej 80. Poniżej tej wartości uzyskane wyniki są znacząco gorsze od optymalnego rozwiązania.

W przedziale 130-200 [todo: co się dzieje]



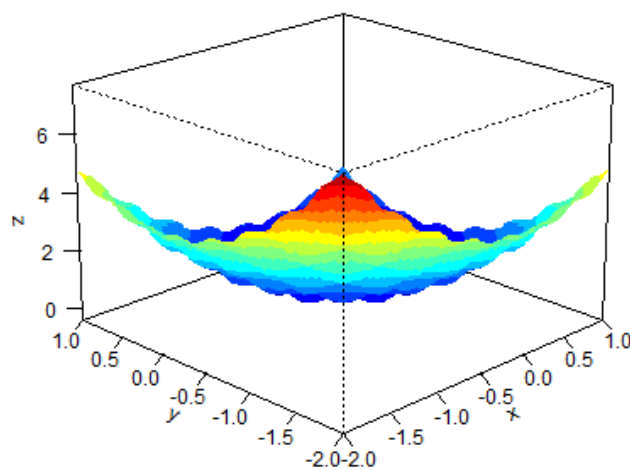
Rysunek 12: Wartość znalezionej optimum w zależności od przyjętego elityzmu

W przypadku funkcji Gulf elitzym ma znaczący wpływ na otrzymywane wyniki. W celu ich optymalizacji wymanaja jest wartość elityzmu na poziomie przynajmniej 0.4.

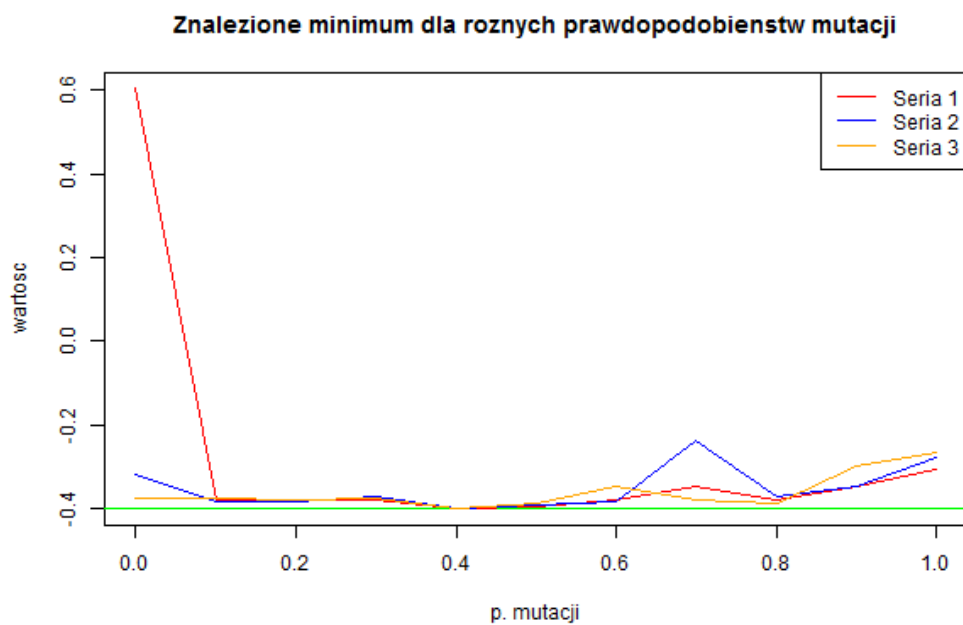
Dla wartości powyżej 0.6 wyniki zaczynają się pogarszać. [todo: dlaczego]  
[todo: zmienić] Jak możemy zauważyć na ilustracji poniżej (rys. ??) przedstawiona lokalizacja optimum nie jest poprawna, gdyż optymalizacji poddano wersję z 3 parametrami. Ogólnie rzecz biorąc gdyby 3 wymiar przedstawić w postaci gradientu kolorystycznego wtedy byłaby to poprawna lokalizacja niemniej trudna dla intuicyjnego sprawdzenia.



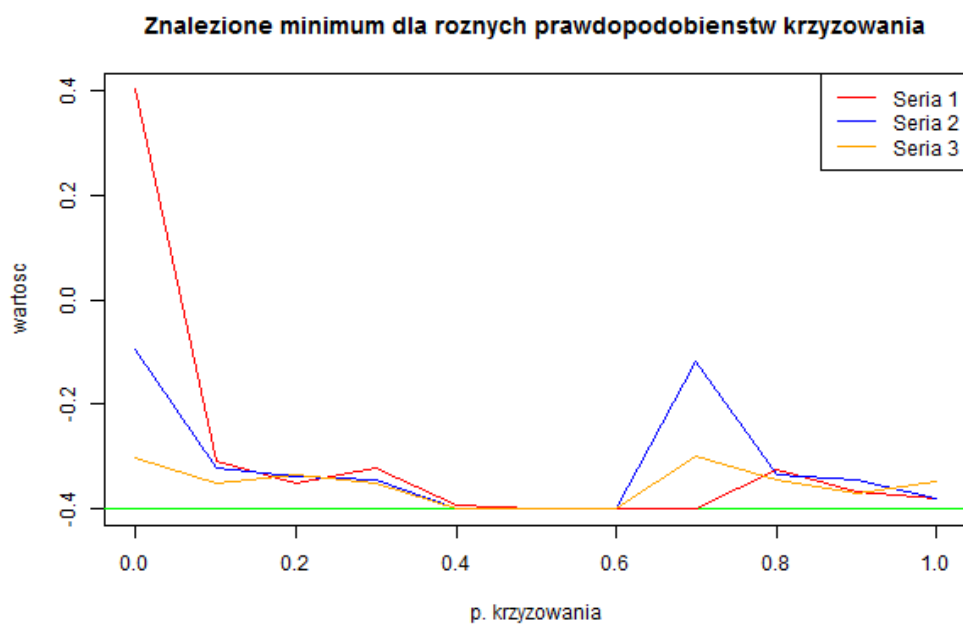
### 3.3 CosMix4 (4 parametry)



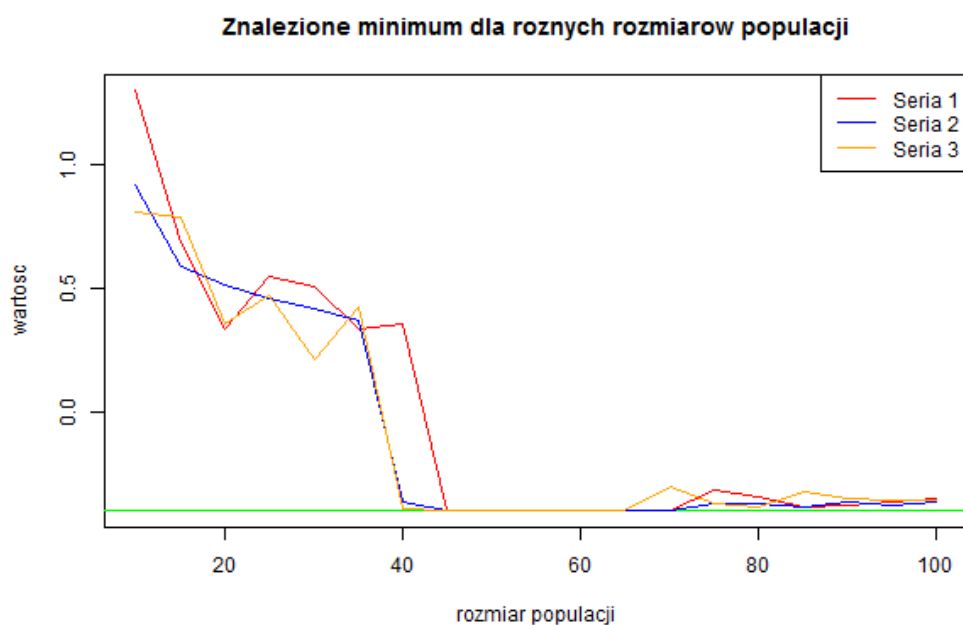
Rysunek 13: Wykres funkcji CosMix4 ( $d=4$ )



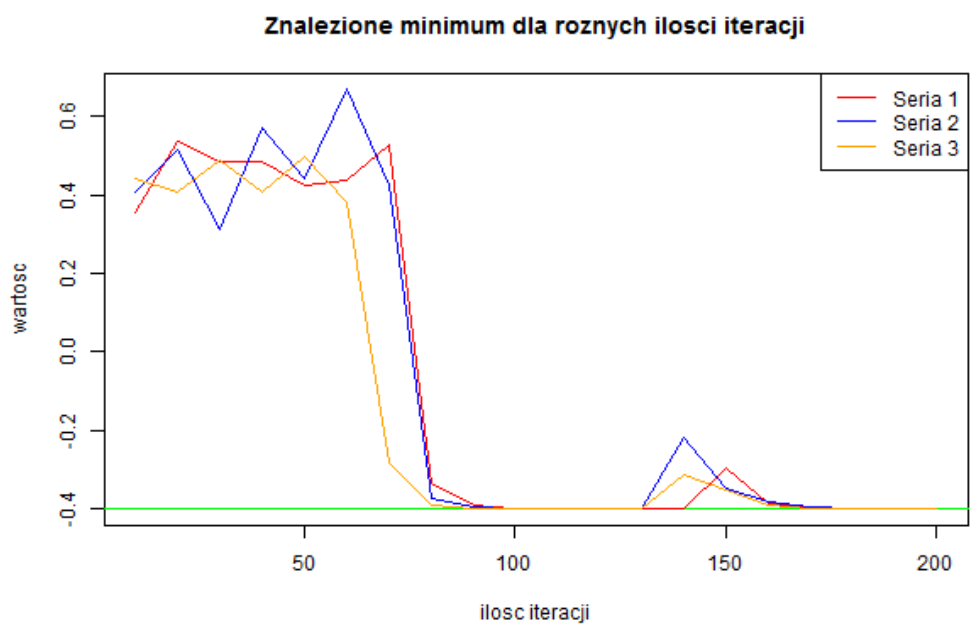
Rysunek 14: Wartość znalezionej optimum w zależności od prawdopodobieństwa mutacji



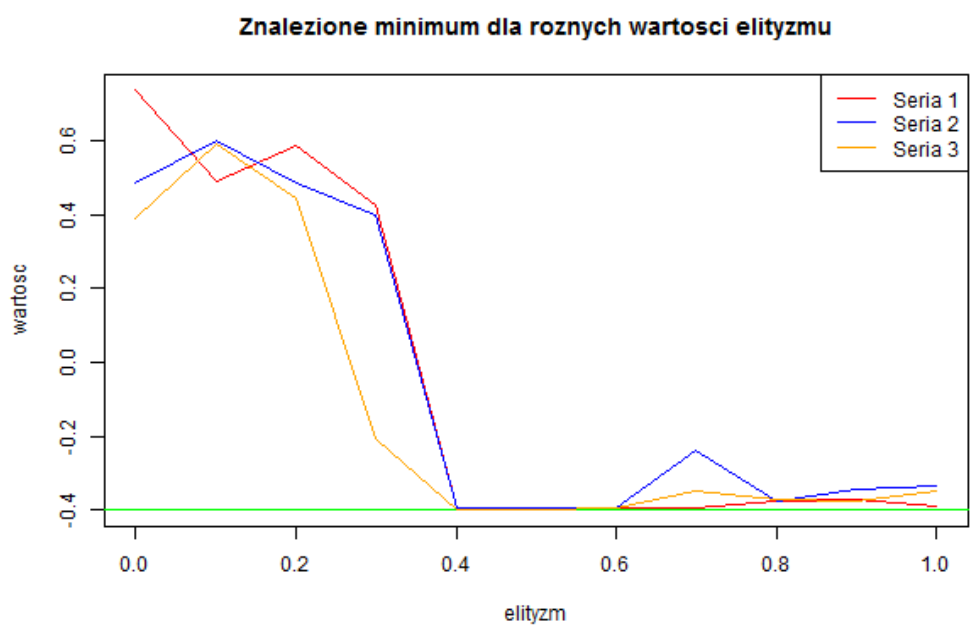
Rysunek 15: Wartość znalezionej optimum w zależności od prawdopodobieństwa krzyżowania



Rysunek 16: Wartość znalezionej optimum w zależności od rozmiarów populacji

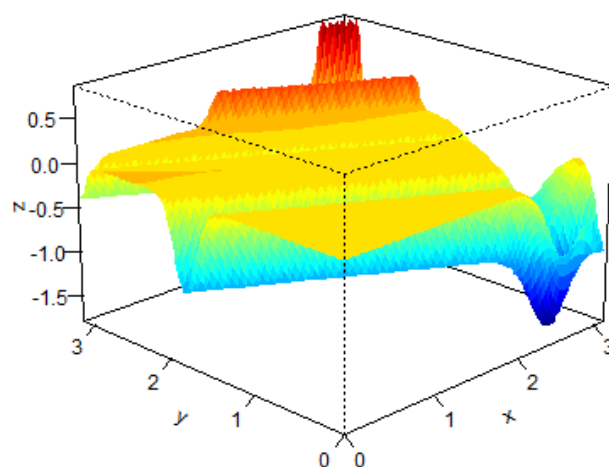


Rysunek 17: Wartość znalezionej optimum w zależności od ilości iteracji

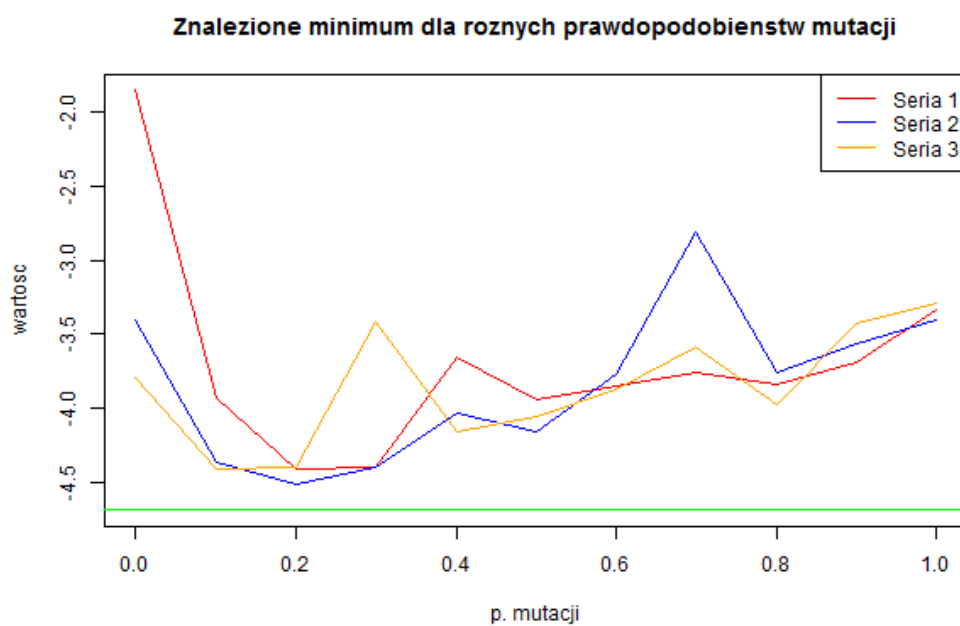


Rysunek 18: Wartość znalezionej optimum w zależności od przyjętego elityzmu

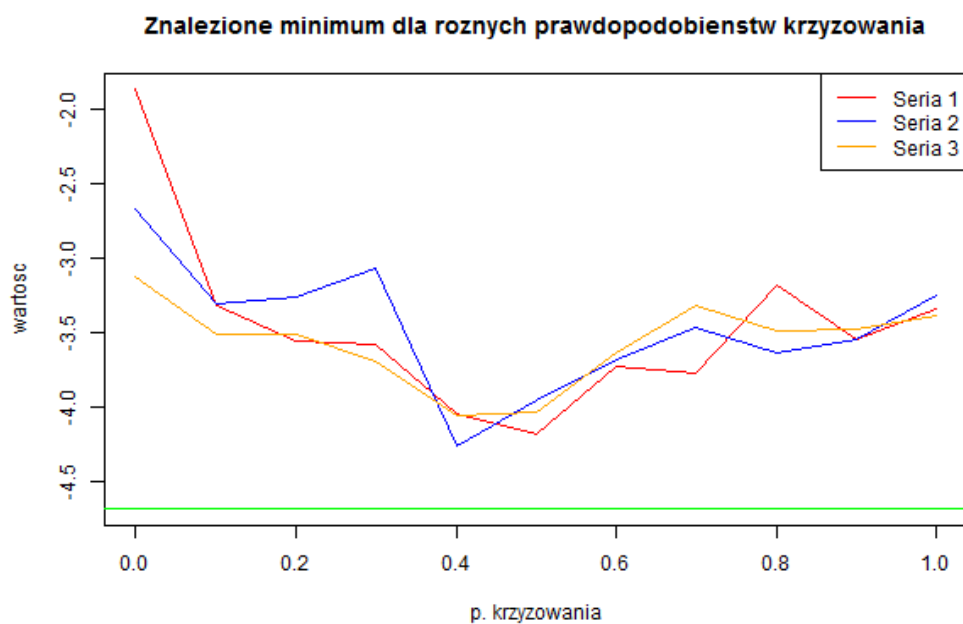
### 3.4 EMichalewicz (5 parametrów)



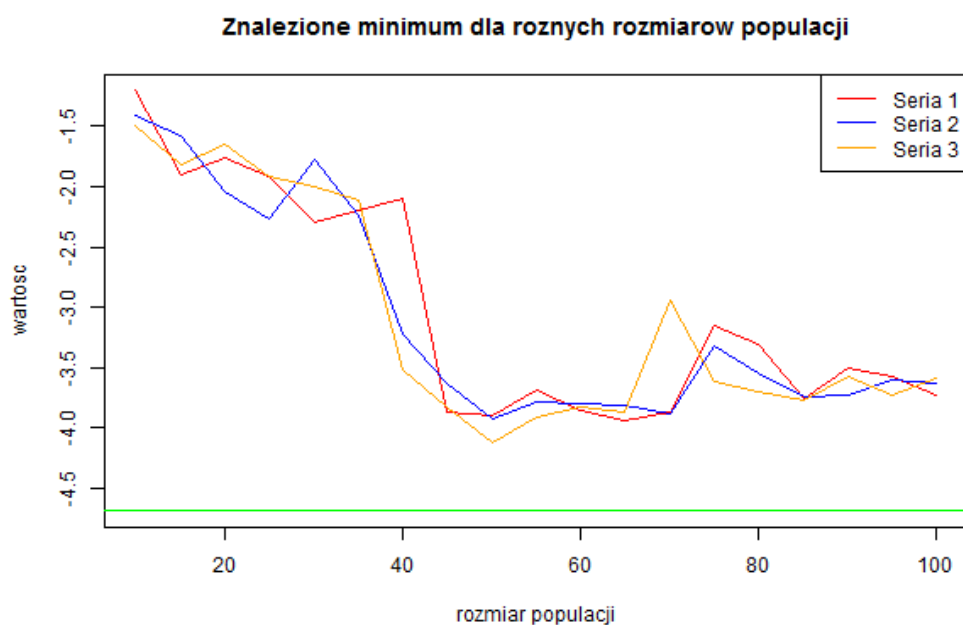
Rysunek 19: Wykres funkcji EMichalewicz ( $d=5$ )



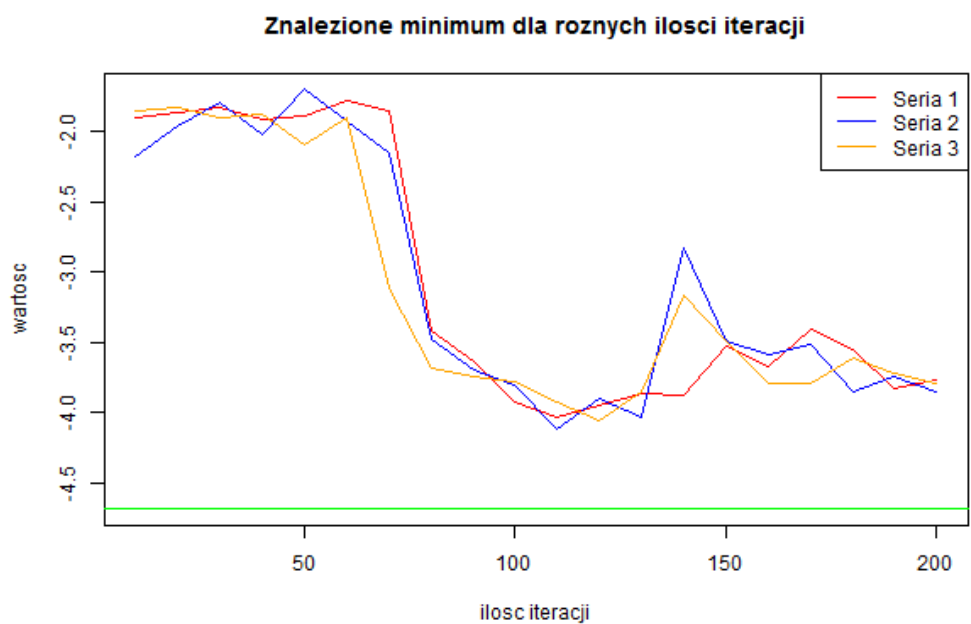
Rysunek 20: Wartość znalezionej optimum w zależności od prawdopodobieństwa mutacji



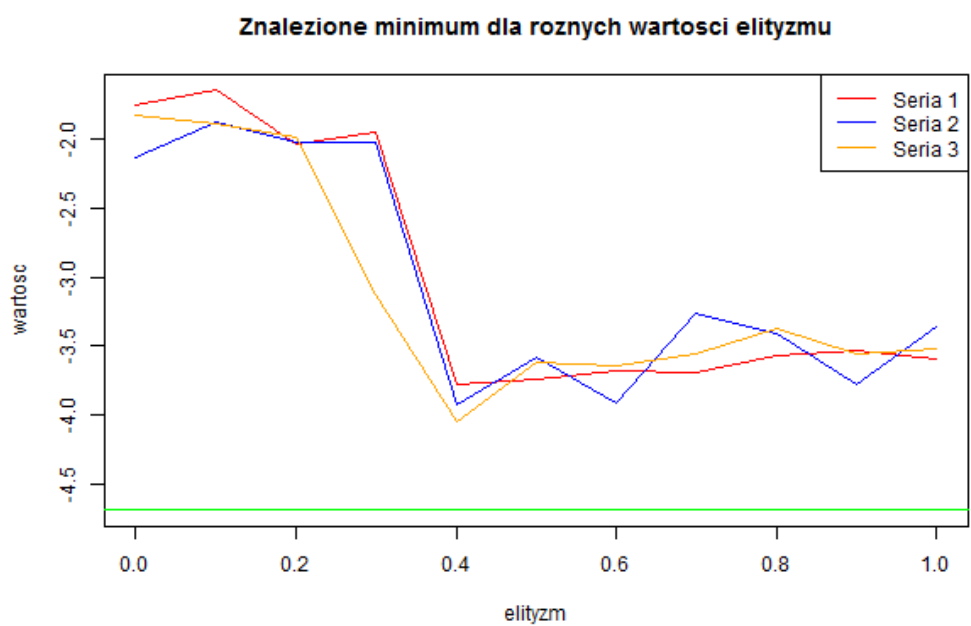
Rysunek 21: Wartość znalezionej optimum w zależności od prawdopodobieństwa krzyżowania



Rysunek 22: Wartość znalezionej optimum w zależności od rozmiarów populacji

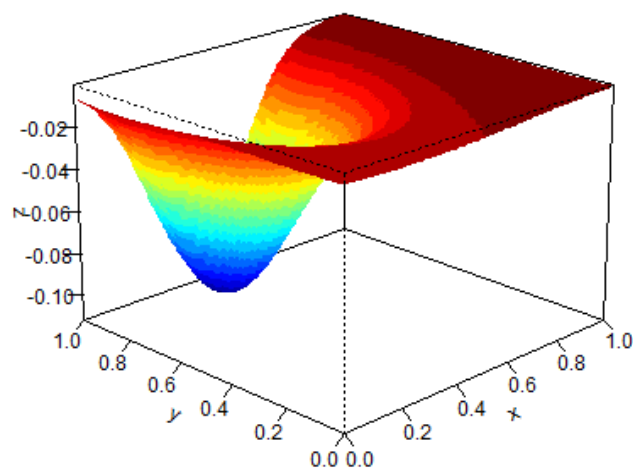


Rysunek 23: Wartość znalezione optimum w zależności od ilości iteracji

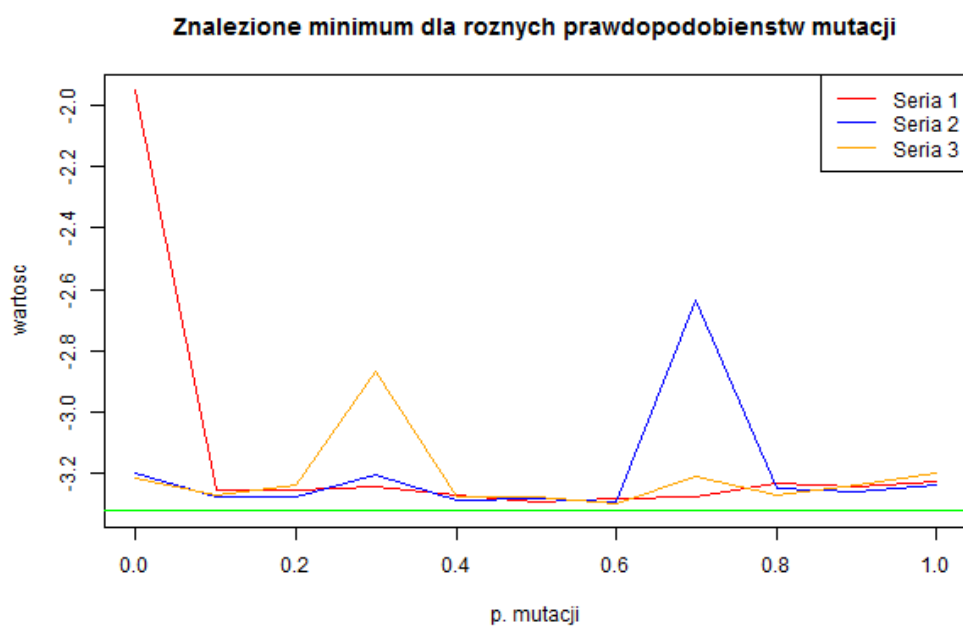


Rysunek 24: Wartość znalezione optimum w zależności od przyjętego elityzmu

### 3.5 Hartman6 (6 parametrów)

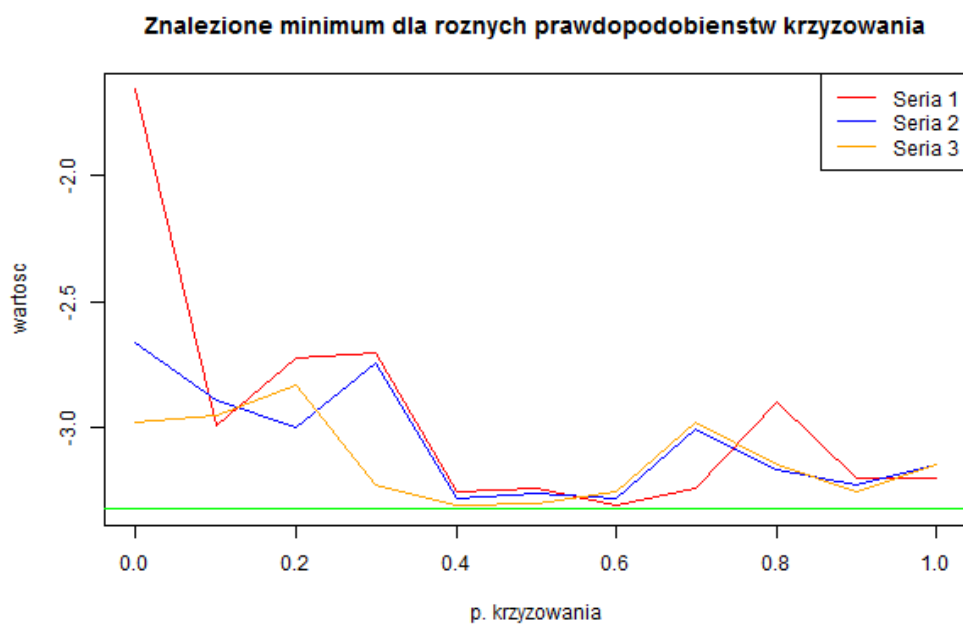


Rysunek 25: Wykres funkcji Hartman6 ( $d=6$ )

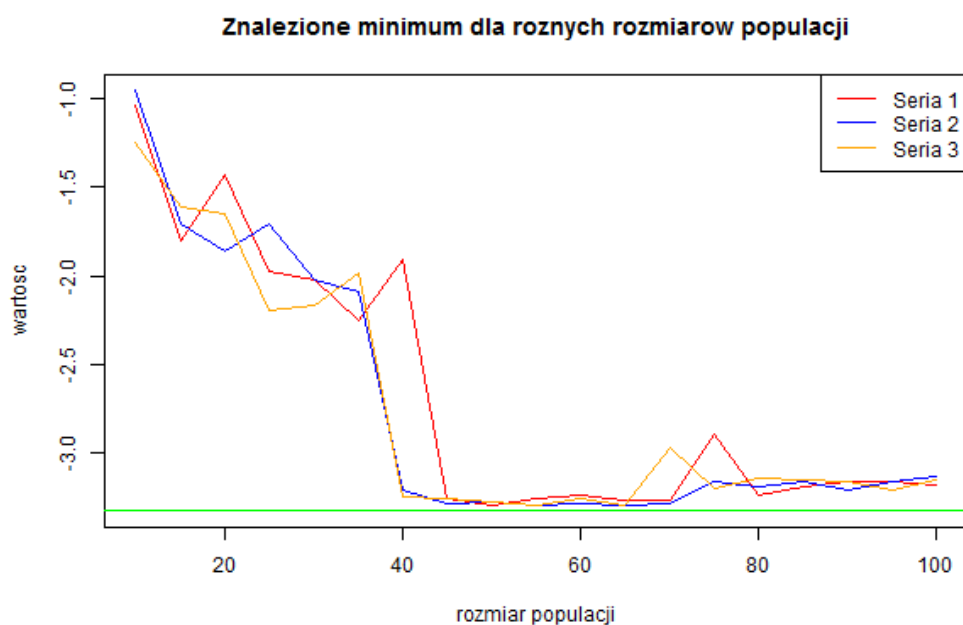


Rysunek 26: Wartość znalezionej optimum w zależności od prawdopodobieństwa mutacji

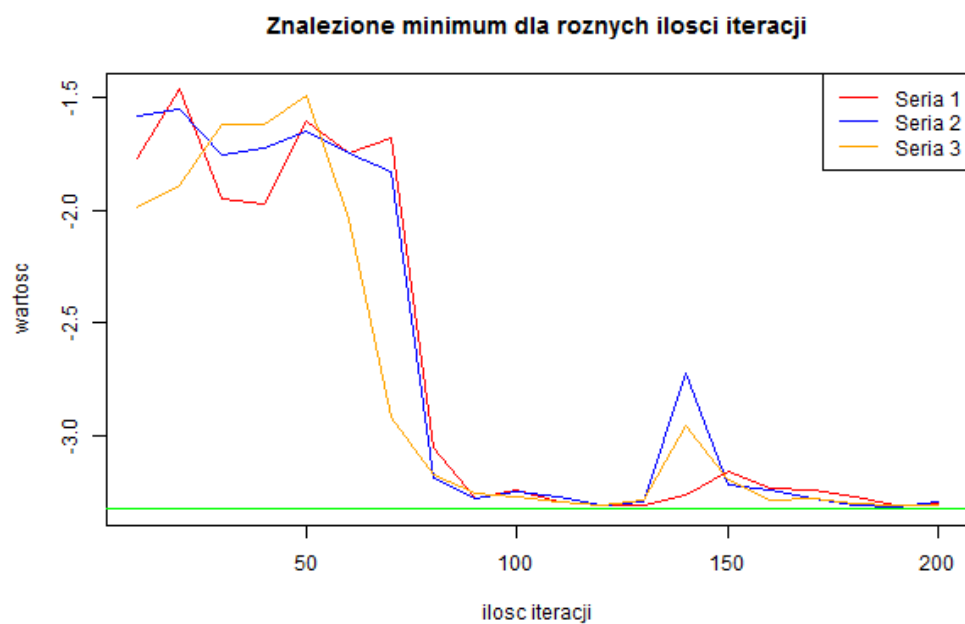




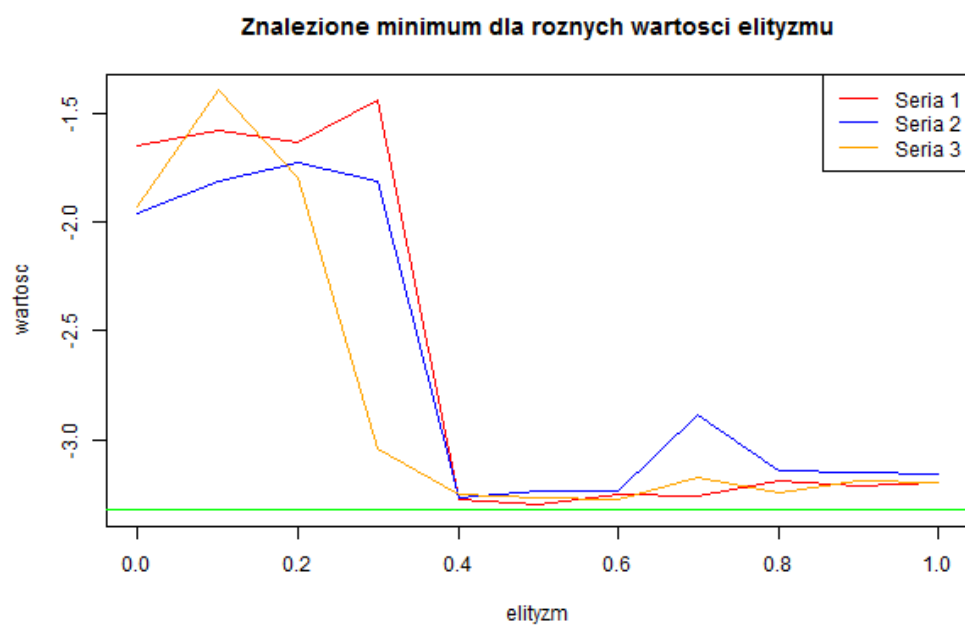
Rysunek 27: Wartość znalezionej optimum w zależności od prawdopodobieństwa krzyżowania



Rysunek 28: Wartość znalezionej optimum w zależności od rozmiarów populacji

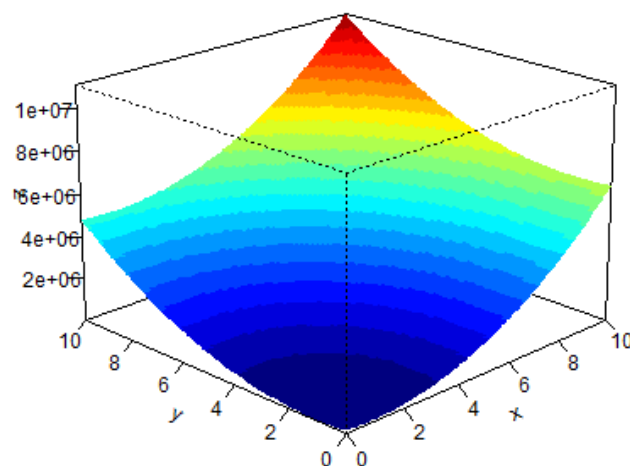


Rysunek 29: Wartość znalezione optimum w zależności od ilości iteracji

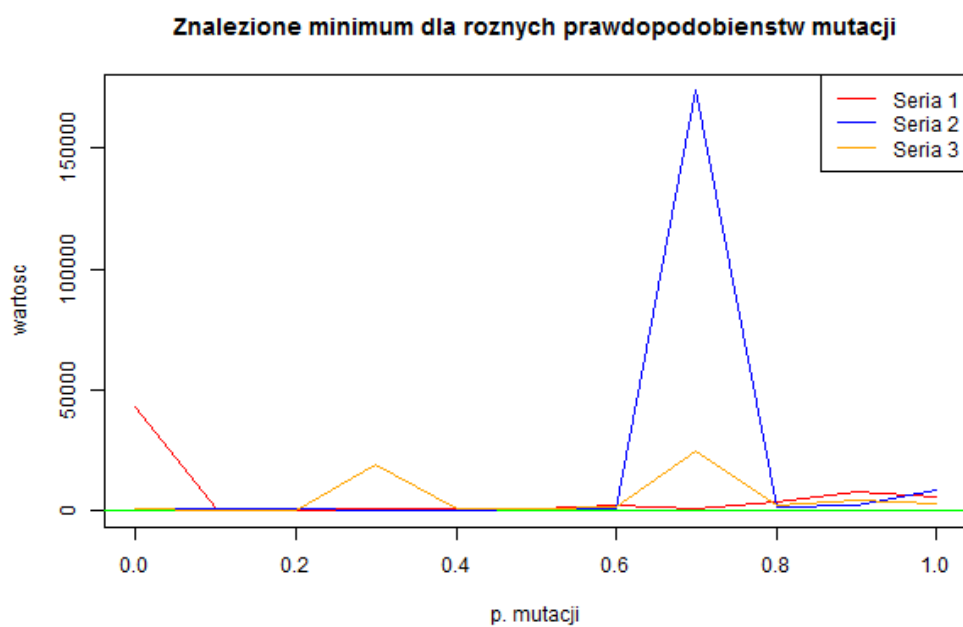


Rysunek 30: Wartość znalezione optimum w zależności od przyjętego elityzmu

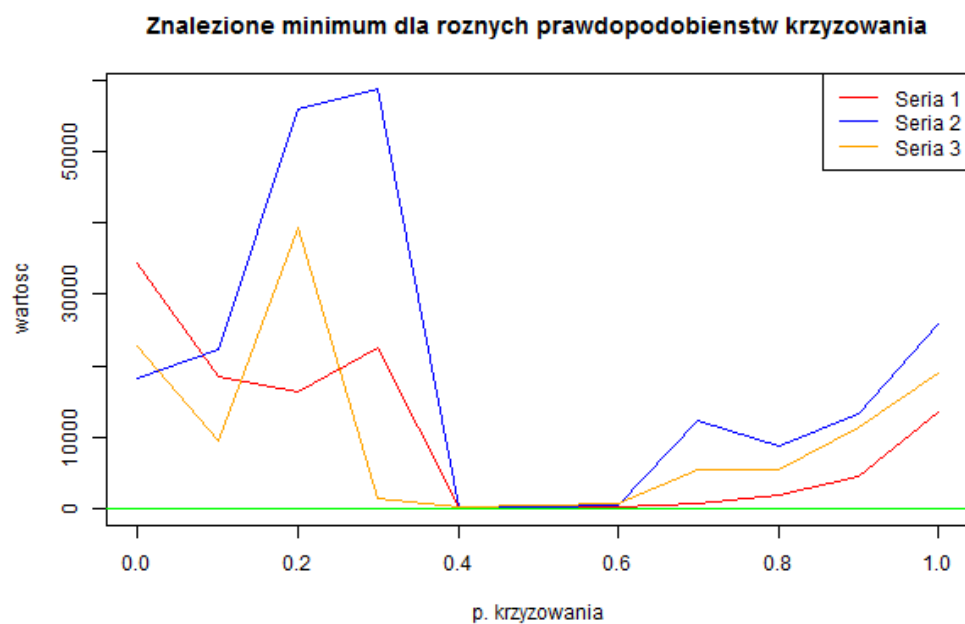
### 3.6 PriceTransistor (9 parametrów)



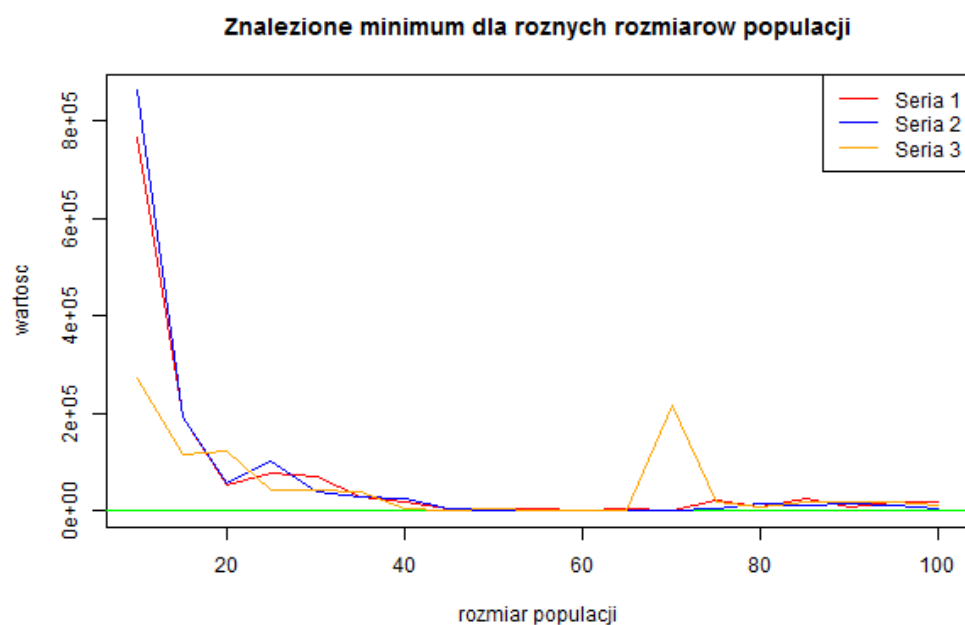
Rysunek 31: Wykres funkcji PriceTransistor ( $d=9$ )



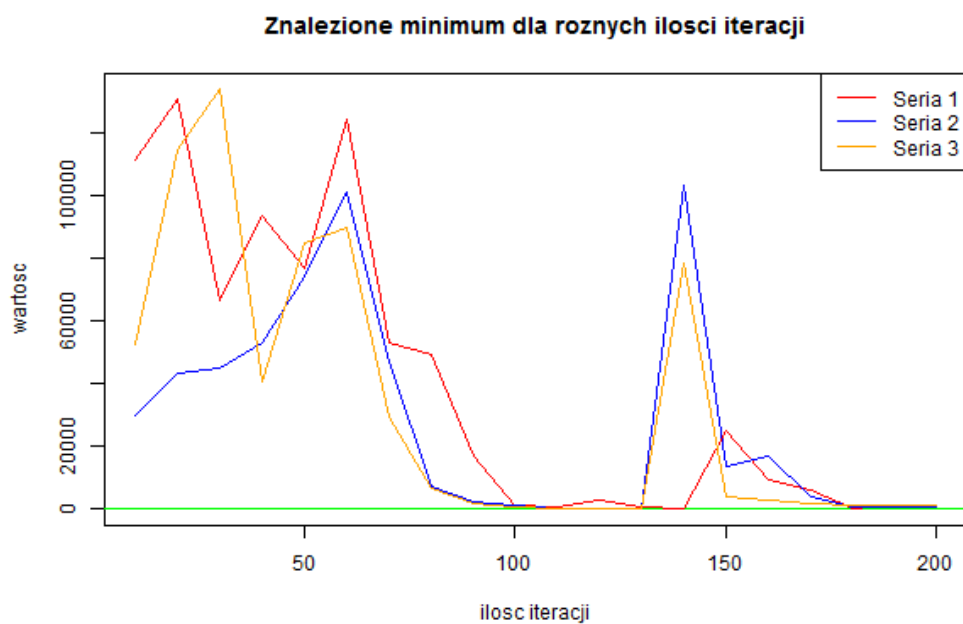
Rysunek 32: Wartość znalezionej optimum w zależności od prawdopodobieństwa mutacji



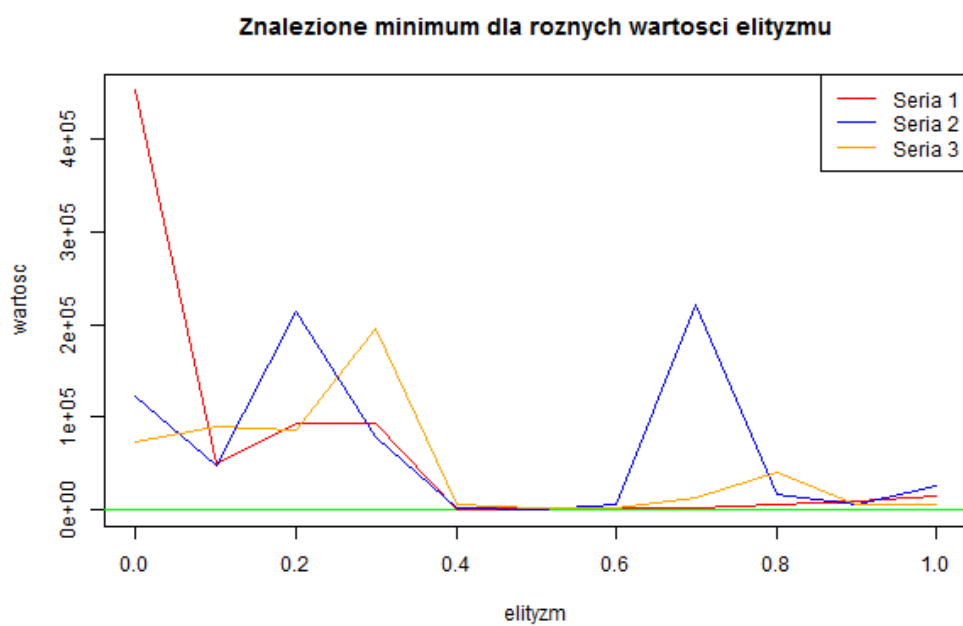
Rysunek 33: Wartość znalezionej optimum w zależności od prawdopodobieństwa krzyżowania



Rysunek 34: Wartość znalezionej optimum w zależności od rozmiarów populacji

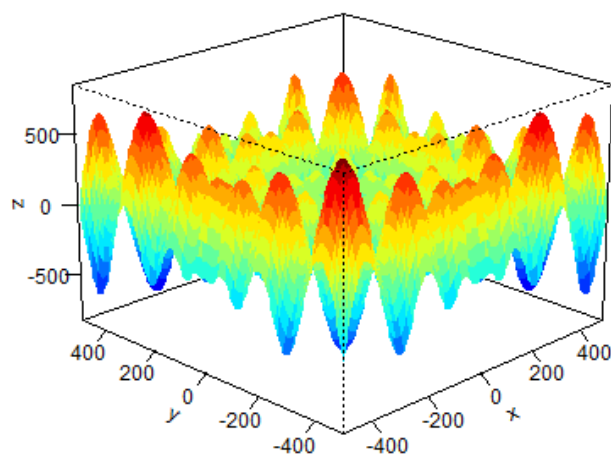


Rysunek 35: Wartość znalezione optimum w zależności od ilości iteracji

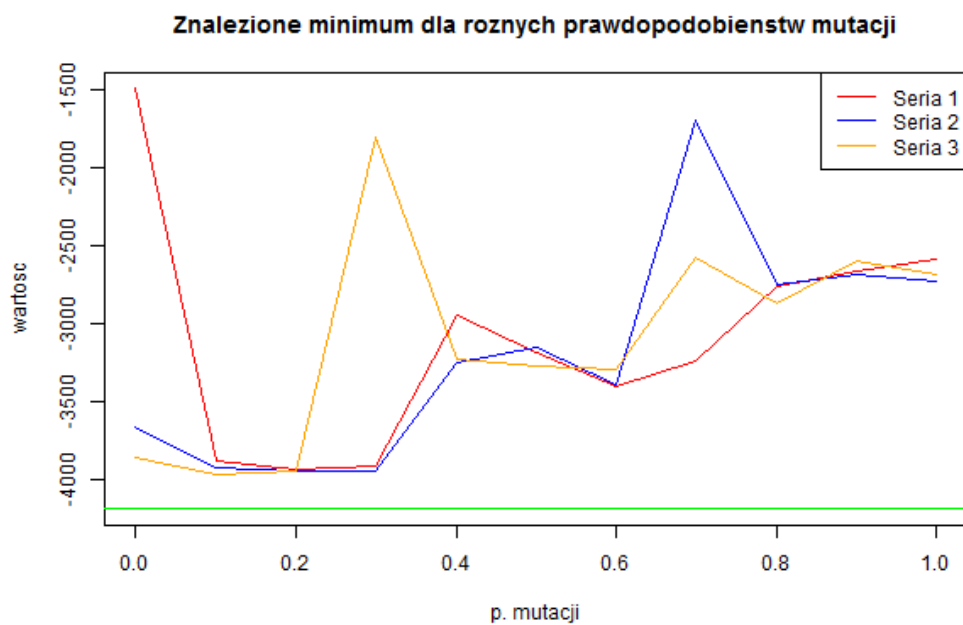


Rysunek 36: Wartość znalezione optimum w zależności od przyjętego elityzmu

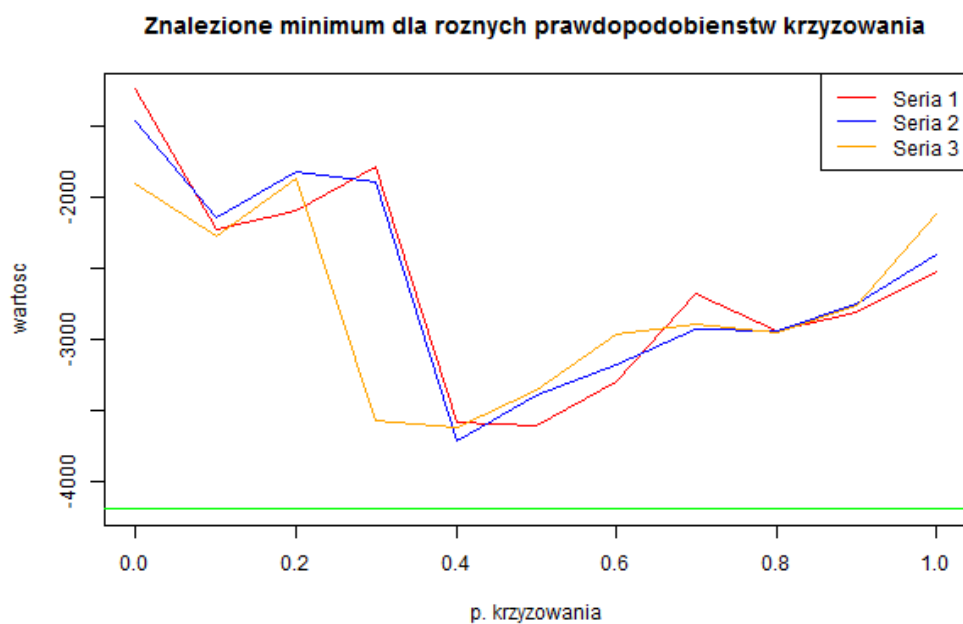
### 3.7 Schwefel (10 parametrów)



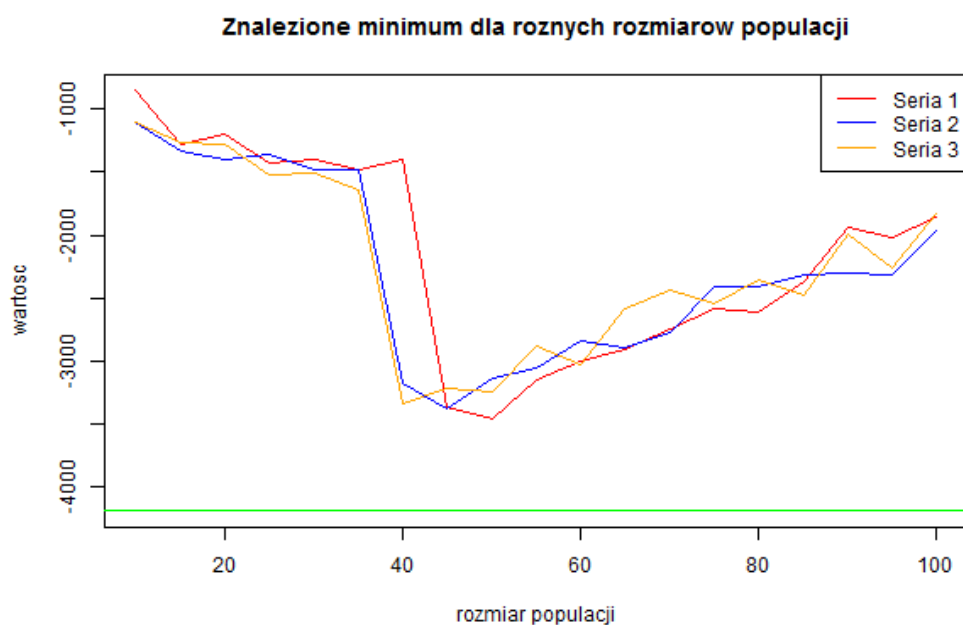
Rysunek 37: Wykres funkcji Schwefel ( $d=10$ )



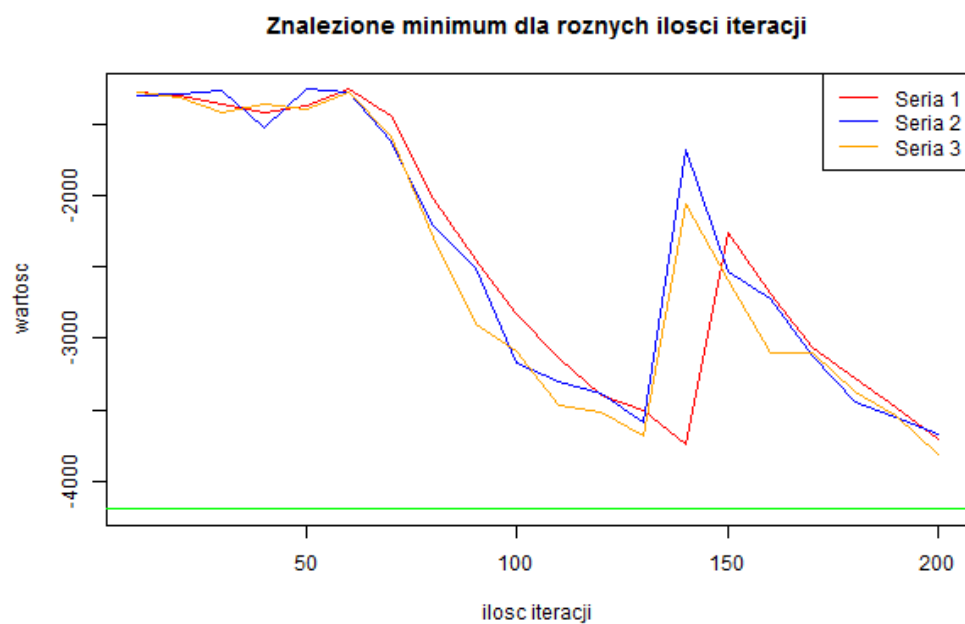
Rysunek 38: Wartość znalezionej optimum w zależności od prawdopodobieństwa mutacji



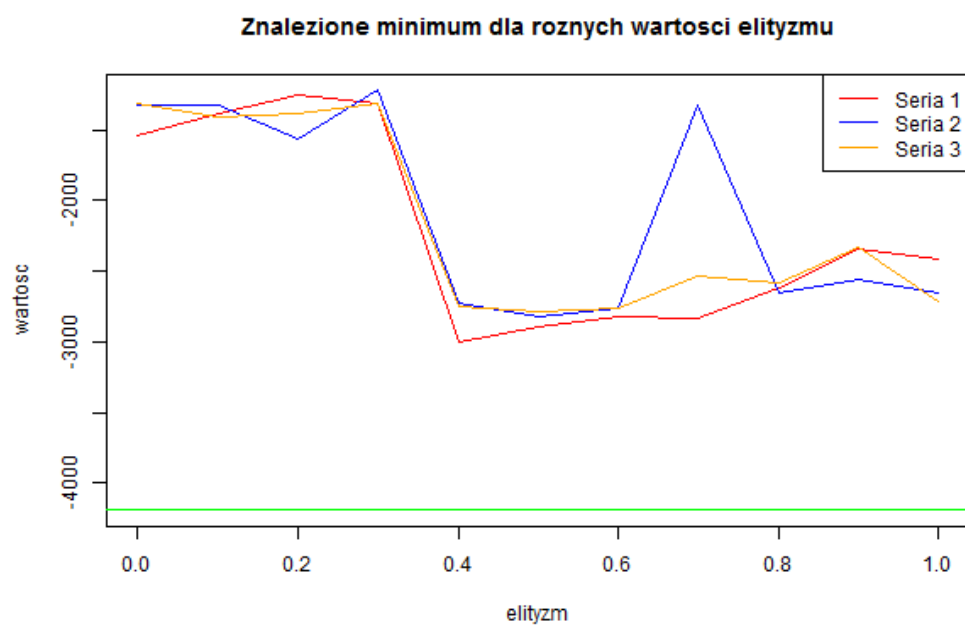
Rysunek 39: Wartość znalezionej optimum w zależności od prawdopodobieństwa krzyżowania



Rysunek 40: Wartość znalezionej optimum w zależności od rozmiarów populacji



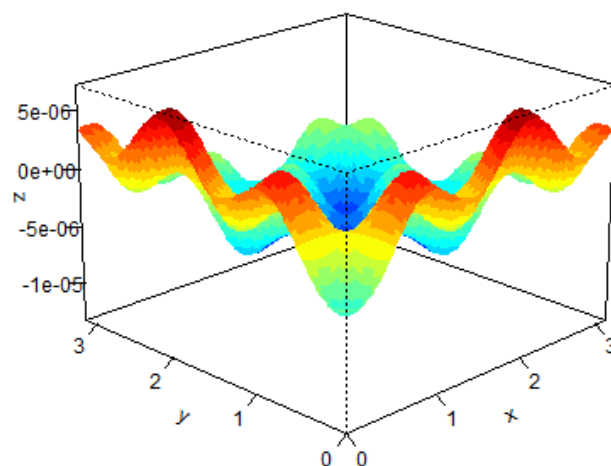
Rysunek 41: Wartość znalezione optimum w zależności od ilości iteracji



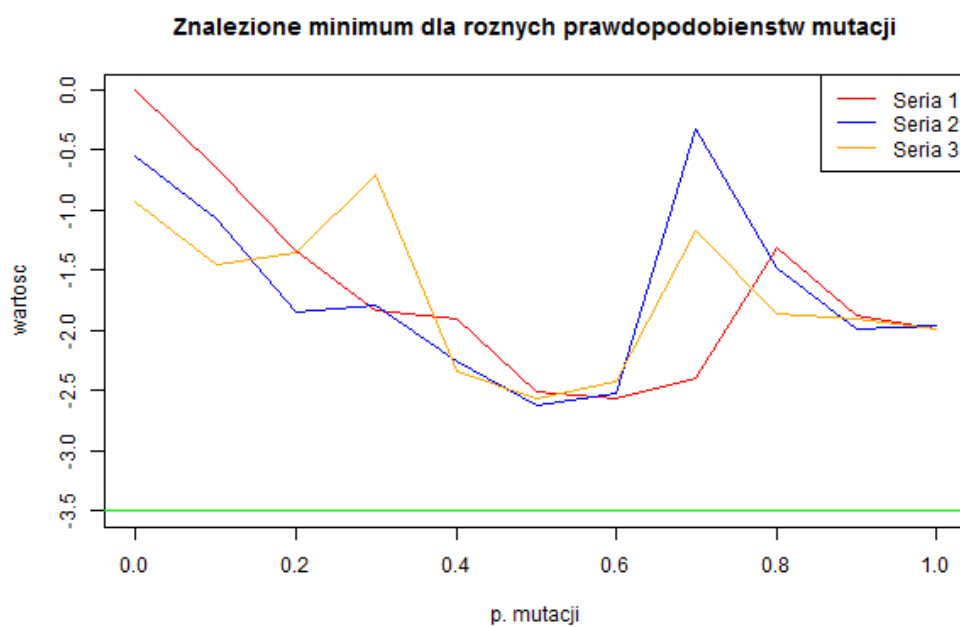
Rysunek 42: Wartość znalezione optimum w zależności od przyjętego elityzmu



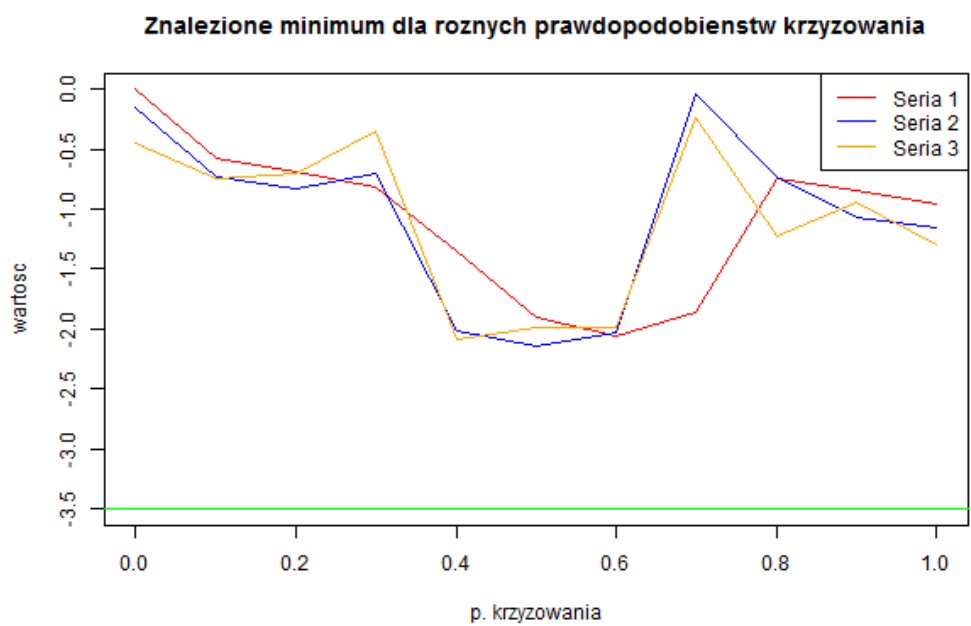
### 3.8 Zeldasine20 (20 parametrów)



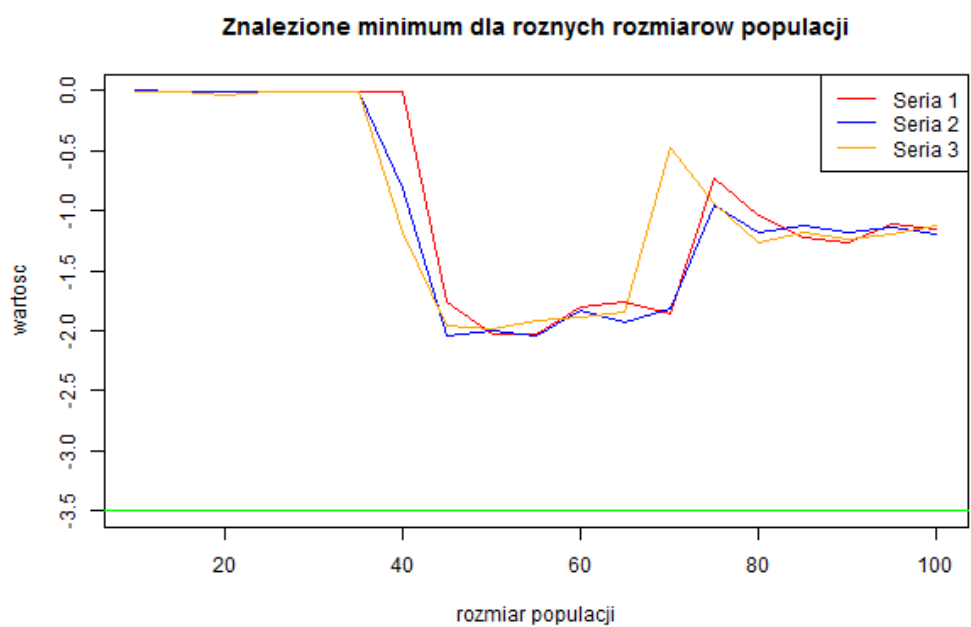
Rysunek 43: Wykres funkcji Zeldasine ( $d=20$ )



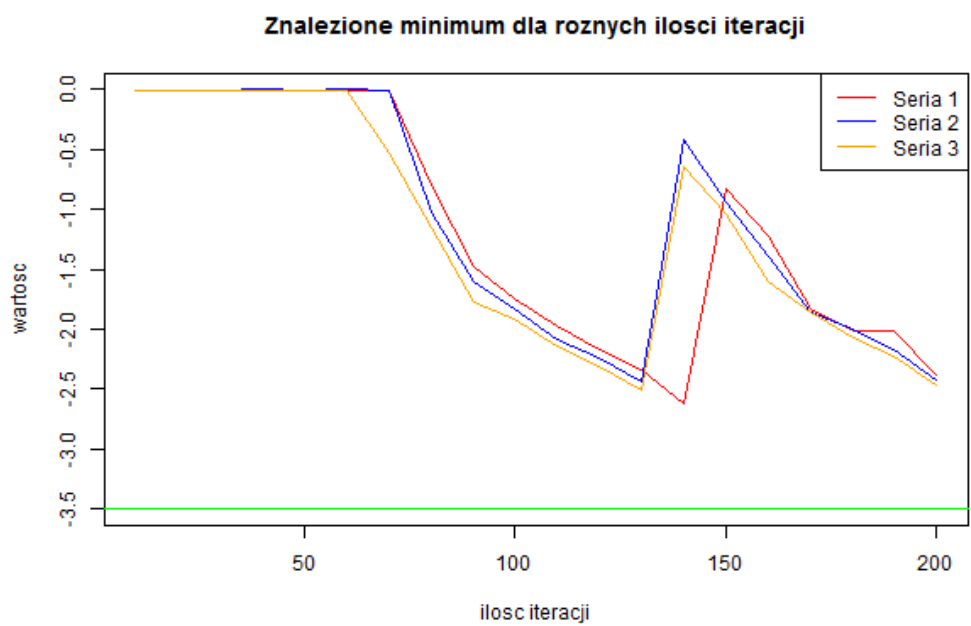
Rysunek 44: Wartość znalezionej optimum w zależności od prawdopodobieństwa mutacji



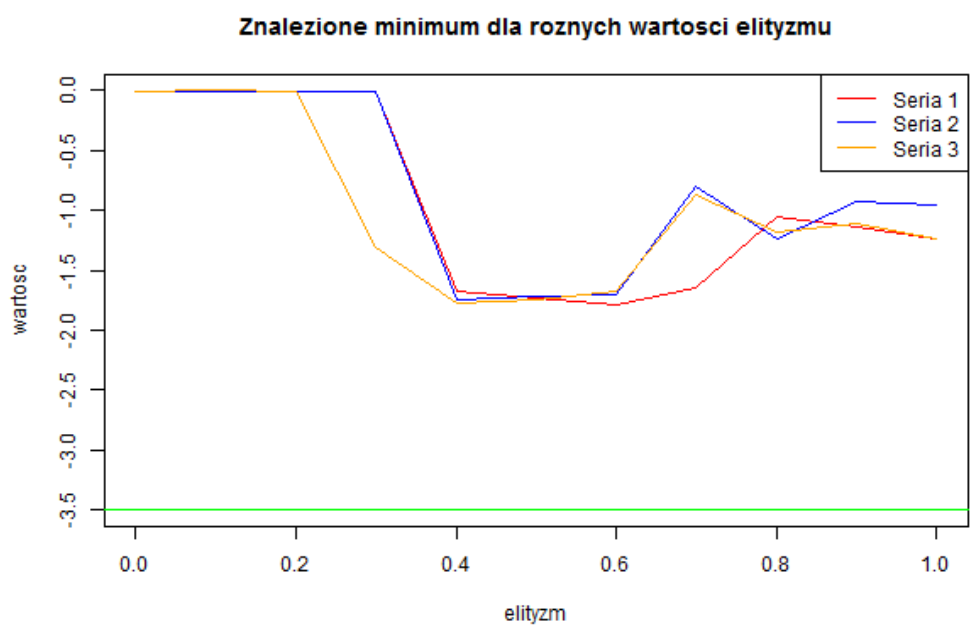
Rysunek 45: Wartość znalezione optimum w zależności od prawdopodobieństwa krzyżowania



Rysunek 46: Wartość znalezione optimum w zależności od rozmiarów populacji



Rysunek 47: Wartość znalezione optimum w zależności od ilości iteracji



Rysunek 48: Wartość znalezione optimum w zależności od przyjętego elityzmu

## 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>