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 5

3 Przebieg badań **6**
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) 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  # Settings ----
10
11 nOfRuns <- 20 # number of runs to calc average
12
13 colors <- c("red", "blue", "purple", "black")
14 series <- c("Seria 1", "Seria 2", "Seria 3", "Seria 4")
15
16 # [mutations,crossovers,populations,iterations,color]
17 params = matrix(
18   c(0, 0, 50, 100, 1,
19     0, 0.8, 50, 100, 2,
20     0.1, 0, 50, 100, 3,
21     0.1, 0.8, 50, 100, 4),
22   nrow=4, ncol=5, byrow = TRUE)
23
24 functions <- c("Branin", "Gulf", "CosMix4", "EMichalewicz",
25    "Hartman6", "PriceTransistor", "Schwefel", "Zeldasine20")
26
27 graphs <- TRUE
28 quality <- 100 #graph resolutions
29
30 mutationTests <- seq(0, 1, 0.1)
31 crossoverTests <- seq(0, 1, 0.1)
32 populationTests <- seq(10, 100, 5)
33 iterationTests <- seq(10, 200, 10)
34 elitismTests <- seq(0, 1, 0.1)
35
36 # Processing ----
37
38 customMeasure <- function(fileName, graphName, values, mType, xlab, main) {
```

```
39
40  gMin <- .Machine$integer.max
41  gBest <- NA
42
43  temp <- c()
44  for (defRow in 1:nrow(params)) {
45    averages <- c()
46    for (value in values) {
47      sum <- 0
48      for (i in 1:nOfRuns) {
49        GAmin <- ga(type = "real-valued",
50            fitness = function(xx) -f(xx),
51            min = c(B[1,]), max = c(B[2,]),
52            popSize = if (mType == "pop") value else params[defRow,3],
53            maxiter = if (mType == "itr") value else params[defRow,4],
54            pmutation = if (mType == "mut") value else params[defRow,1],
55            pcrossover = if (mType == "crs") value else params[defRow,2],
56            elitism = if (mType == "elt") value else max(1,
                  round(params[defRow,3] * 0.05)))
57        solution <- matrix(unlist(GAmin@solution),ncol=dim,byrow=TRUE)
58        eval <- f(solution[1,])
59        if (eval < gMin) {
60          gMin <- eval
61          gBest <- GAmin
62        }
63        sum <- sum + eval
64      }
65      averages <- c(averages, (sum / nOfRuns))
66    }
67    temp <- c(temp, averages)
68  }
69  result <- matrix(c(temp),nrow = nrow(params),ncol = length(values))
70  write.table(result, file = paste(funcName, fileName, sep=""), row.names=FALSE,
              na="", col.names=FALSE, sep=";")
71
72
73  if (graphs) {
74    png(file = paste(funcName, graphName, ".png", sep=""), width=600,
        height=400, units="px")
75    plot(0, 0, main=main,
76         ylim=c(min(c(temp,globalOpt)),max(c(temp,globalOpt))),
77         xlim=c(min(values),max(values)),
78         type="n", xlab=xlab, ylab="wartosc")
79    abline(globalOpt,0, col="green")
80    colorNames <- c()
81    seriesNames <- c()
82    for (i in 1:nrow(params)) {
83      color <- colors[params[i,5]]
84      colorNames <- c(colorNames, color)
85      seriesNames <- c(seriesNames, series[params[i,5]])
86      lines(values, result[i,], col = color, type = 'l')
87    }
88    legend("topright", seriesNames, lwd=rep(2,nrow(params)),
        lty=rep(1,nrow(params)), col=colorNames)
89    dev.off()
90    summary(gBest)
91    png(file = paste(funcName, graphName, mType, ".png", sep=""), width=600,
```

```
        height=400, units="px")
    filled.contour(x, y, z, color.palette = jet.colors, nlevels = 24,
        plot.axes = { axis(1); axis(2);
          points(solution[1,1], solution[1,2],
                pch = 3, cex = 5, col = "black", lwd = 2)
        }
    )
    dev.off()
    png(file = paste(funcName, graphName, mType, "fitness", ".png", sep=""),
        width=600, height=400, units="px")
    plot(gBest)
    dev.off()
  }

}


for (funcName in functions) {

   dim <- getProblemDimen(funcName)
   B <- matrix(unlist(getDefaultBounds(funcName)),ncol=dim,byrow=TRUE)
   f <- function(xx) goTest(par=c(xx, rep(0, dim-length(xx))),
                    fnName=funcName, checkDim = TRUE)
   globalOpt <- getGlobalOpt(funcName)

   if (graphs) {

     xprobes <- abs(B[2,1] - B[1,1]) / quality
     yprobes <- abs(B[2,2] - B[1,2]) / quality
     x <- seq(B[1,1], B[2,1], by = xprobes)
     y <- seq(B[1,2], B[2,2], by = yprobes)
     z <- outer(x, y, Vectorize(function(x,y) f(c(x,y))))
     nbcol = 100
     color = rev(rainbow(nbcol, start = 0/6, end = 4/6))
     zcol  = cut(z, nbcol)
     persp3d(x, y, z, theta=50, phi=25, expand=0.75, col=color[zcol],
          ticktype="detailed",axes=TRUE)

     png(file = paste(funcName, "1.png", sep=""), width=600, height=400,
         units="px")
     persp3D(x, y, z, theta = -45, phi = 20, color.palette = jet.colors)
     dev.off()

   }

   customMeasure("resultsMutations.csv", "2", mutationTests, "mut",
      "p. mutacji", "Znalezione minimum dla roznych prawdopodobienstw mutacji")

   customMeasure("resultsCrossover.csv", "3", crossoverTests, "crs",
      "p. krzyzowania", "Znalezione minimum dla roznych prawdopodobienstw
          krzyzowania")

   customMeasure("resultsPopulation.csv", "4", populationTests, "pop",
      "rozmiar populacji", "Znalezione minimum dla roznych rozmiarow populacji")

   customMeasure("resultsIterations.csv", "5", iterationTests, "itr",
```

```
144       "ilosc iteracji", "Znalezione minimum dla roznych ilosci iteracji")
145
146     customMeasure("resultsElitism.csv", "6", elitismTests, "elt",
147       "elityzm", "Znalezione minimum dla roznych wartosci elityzmu")
148
149 }
```

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 „globalOptTests". [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 przedstawiono poniżej (tab. 1).

Tabela 1: Parametry domyślne poszczególnych serii pomiarowych

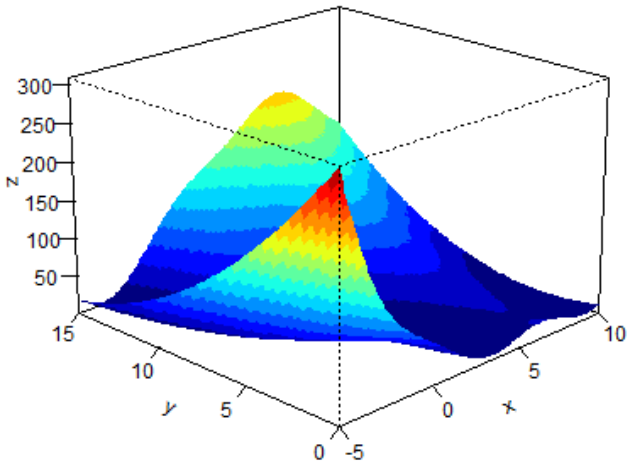
-	Seria 1	Seria 2	Seria 3	Seria 4
Rozmiar populacji	50	50	50	50
Rozmiar iteracji	100	100	100	100
Prawdopodobieństwo mutacji	0	0	0.1	0.1
Prawdopodobieństwo krzyżowania	0	0.8	0	0.8

Zielone linie na wykresach oznaczają optima zwracane w pakiecie „globalOptTests" dla danej funkcji przy domyślnych ograniczeniach (tych samych dla których wykonywana jest optymalizacja podczas omawianych badań) .

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]

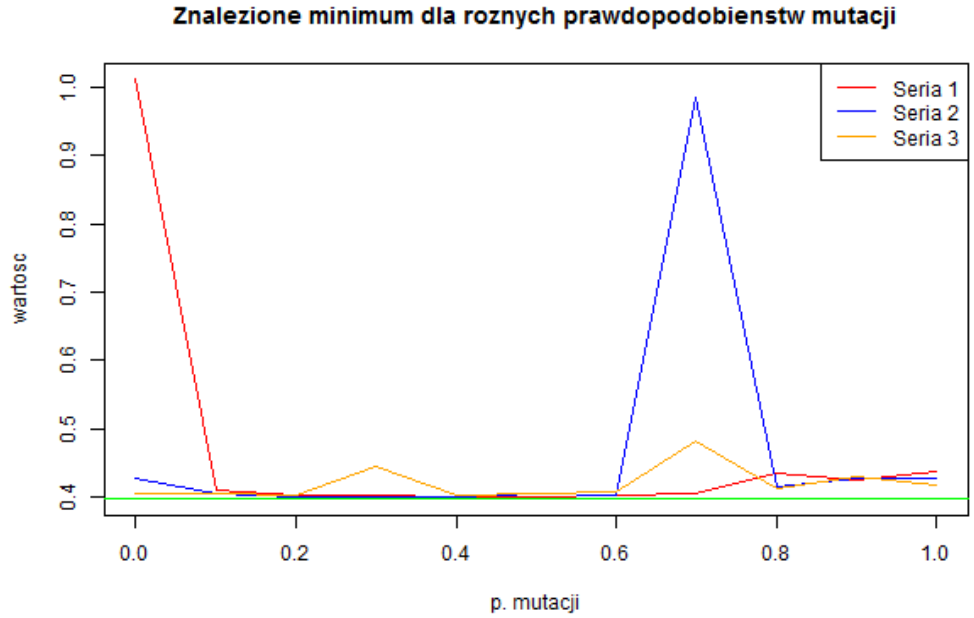
$$f(\boldsymbol{x}) = a(x_2 - bx_1^2 + cx_1 - r)^2 + s(1-t)\cos(x_1) + s \tag{1}$$



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.

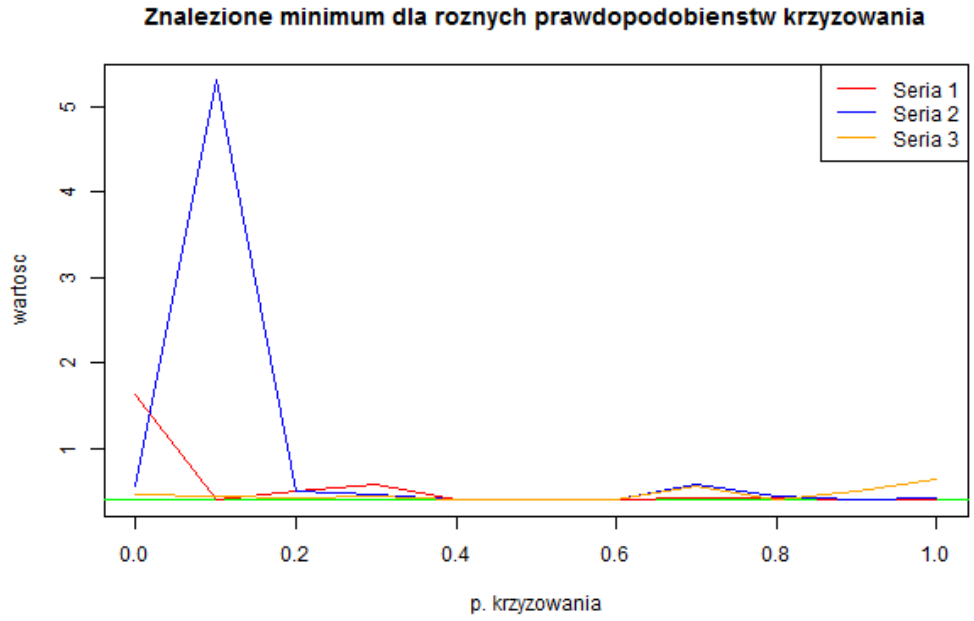


Rysunek 2: Wartość znalezionego optimum w zależności od prawdopodobieństwa mutacji

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

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

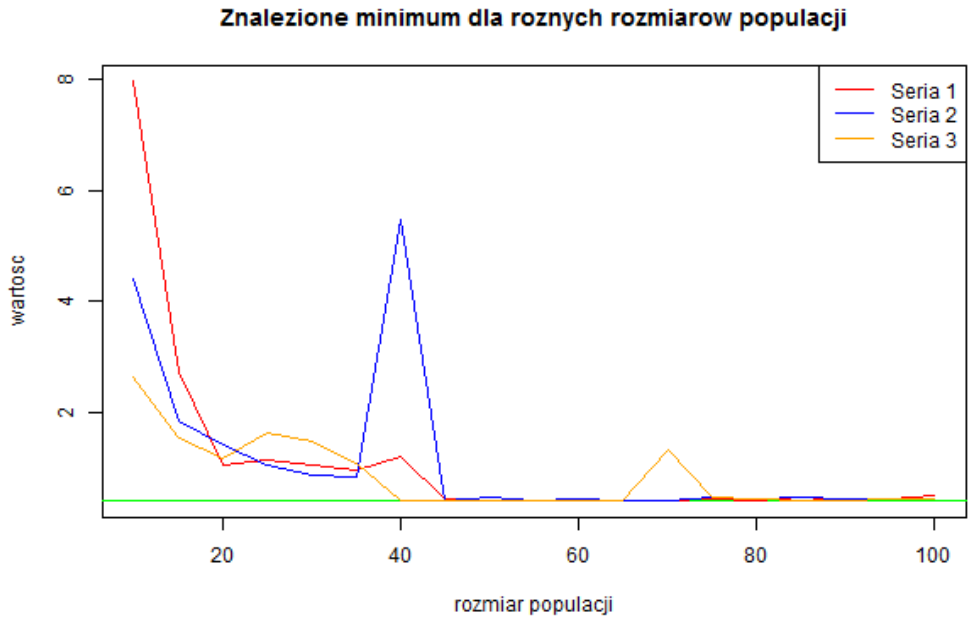
Wyjątkiem stanowi tutaj "seria 2" reprezentująca drugi zestaw wartości domyślnych. Przy mutacji wynoszącej 0.7 wynik funkcji znacząco się pogorszył.



Rysunek 3: Wartość znalezionego optimum w zależności od prawdopodobieństwa krzyżowania

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

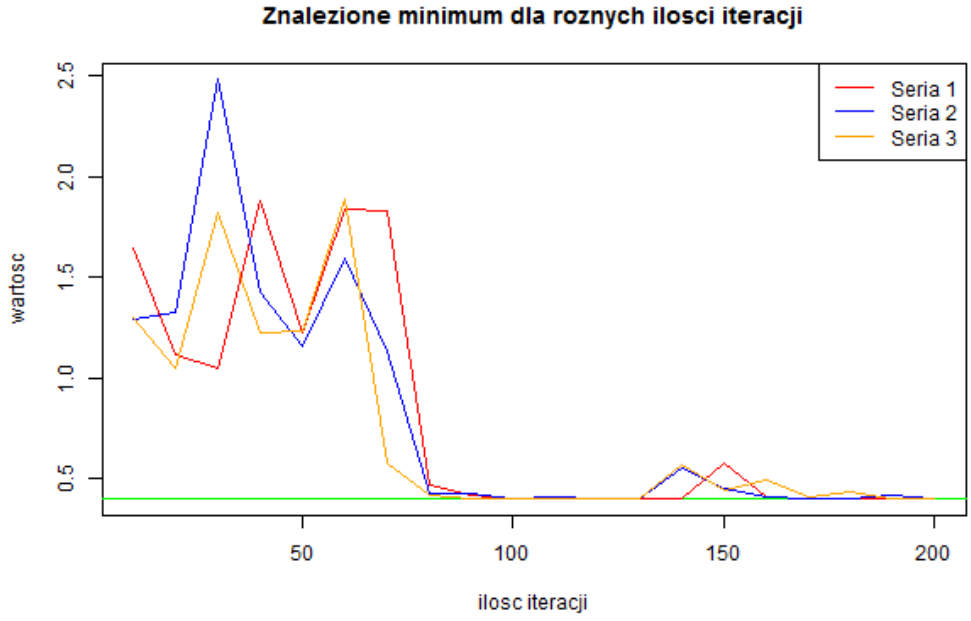
Najlepszy rezultat został otrzymany w przedziale prawdopodobieństwa krzyżowania między 0.4 a 0.6.



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.

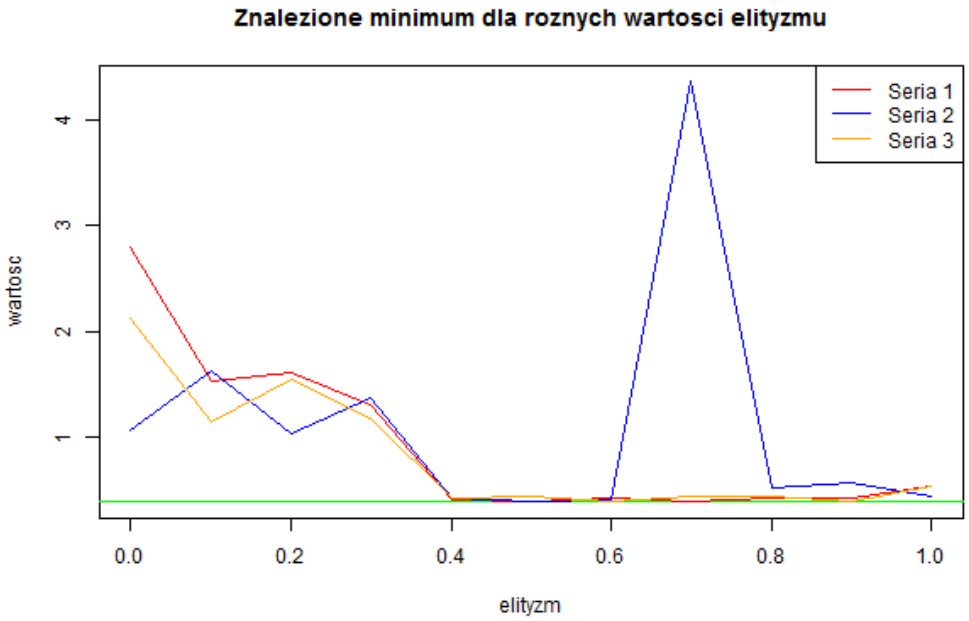
Zauważalny jest wzrost jakości rozwiązania wraz ze wzrostem ilości jednostek populacji.



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 oczeki-

wanej.



Rysunek 6: Wartość znalezionego 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.

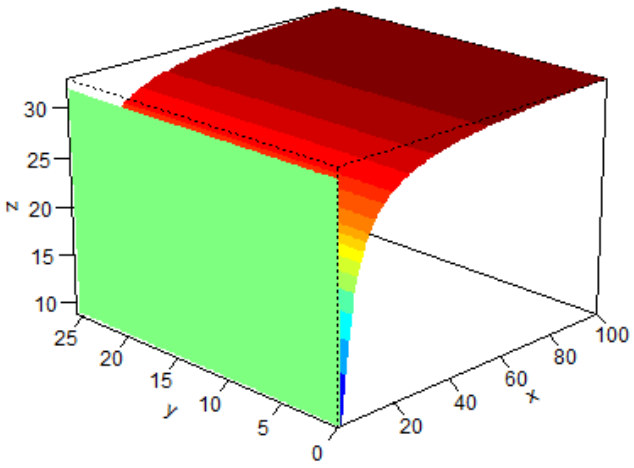
Warto tutaj zauważyć ponowne (jak w przypadku mutacji) obniżenie się jakości wyniku dla przedziału 0.6-0.8.

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]

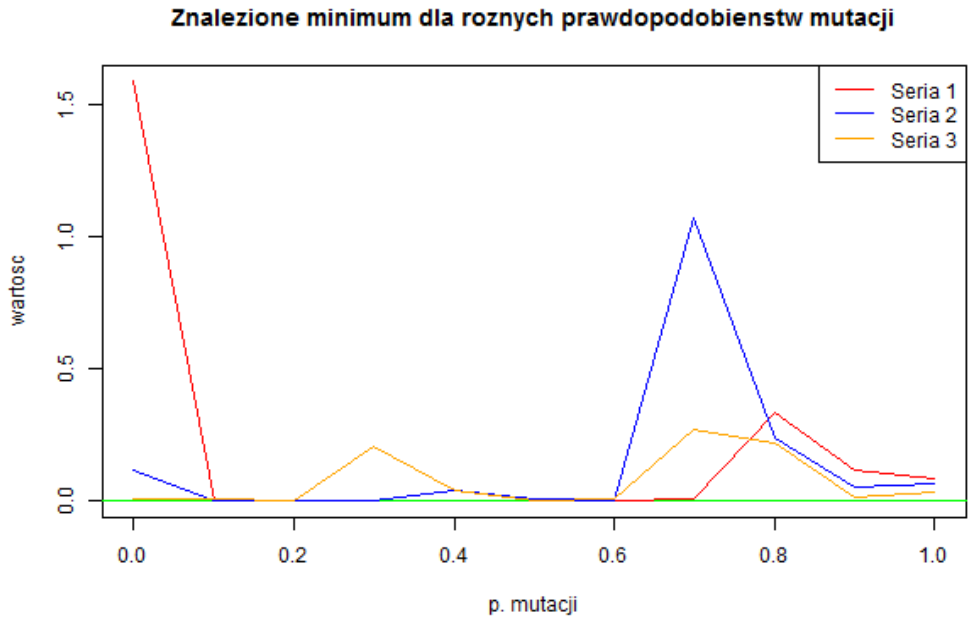
$$f(\boldsymbol{x}) = \tag{2}$$

http://www.gamsworld.org/performance/selconglobal/htm/selconglobal/Gulf.htm



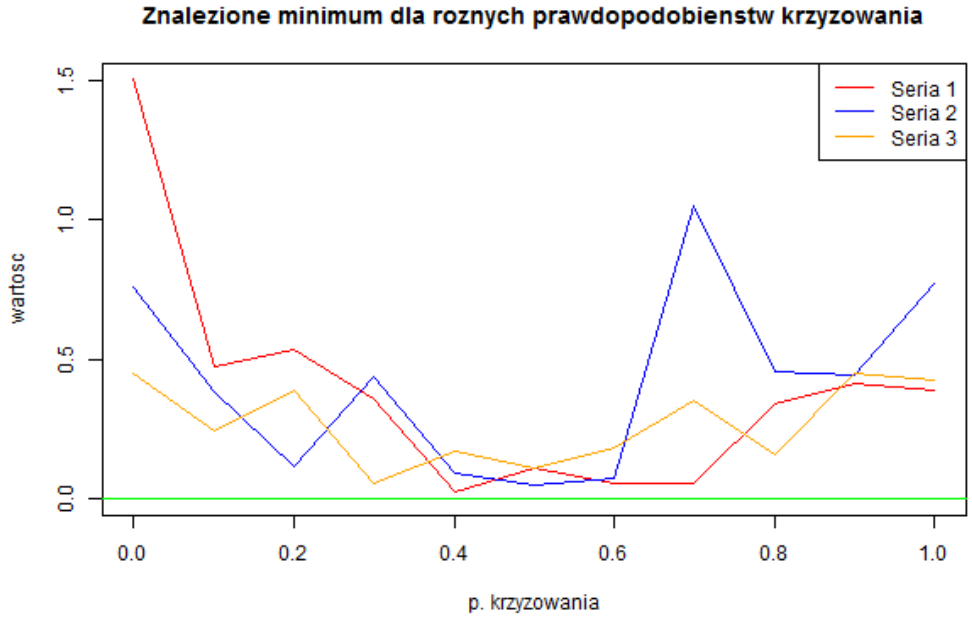
Rysunek 7: Wykres funkcji Gulf (d=3)

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



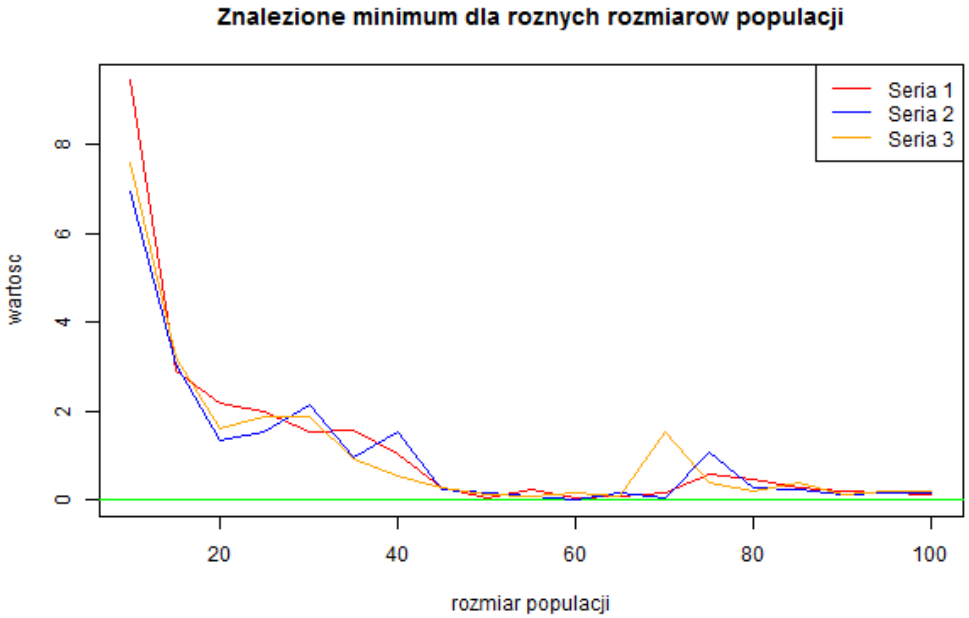
Rysunek 8: Wartość znalezionego optimum w zależności od prawdopodobieństwa mutacji

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.



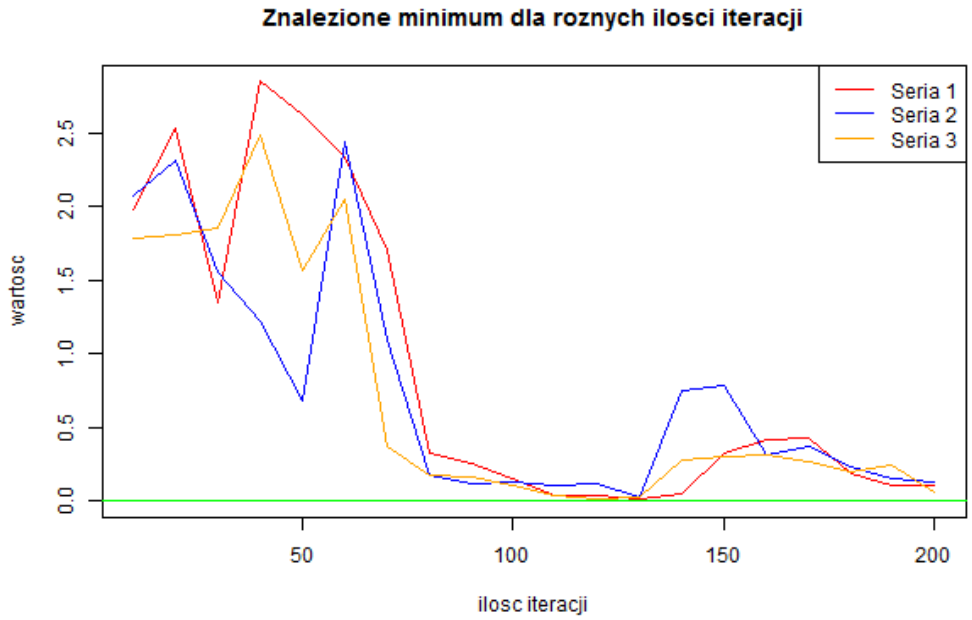
Rysunek 9: Wartość znalezionego optimum w zależności od prawdopodobieństwa krzyżowania

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 jakoś uzyskanych wyników.



Rysunek 10: Wartość znalezionego optimum w zależności od rozmiarów populacji

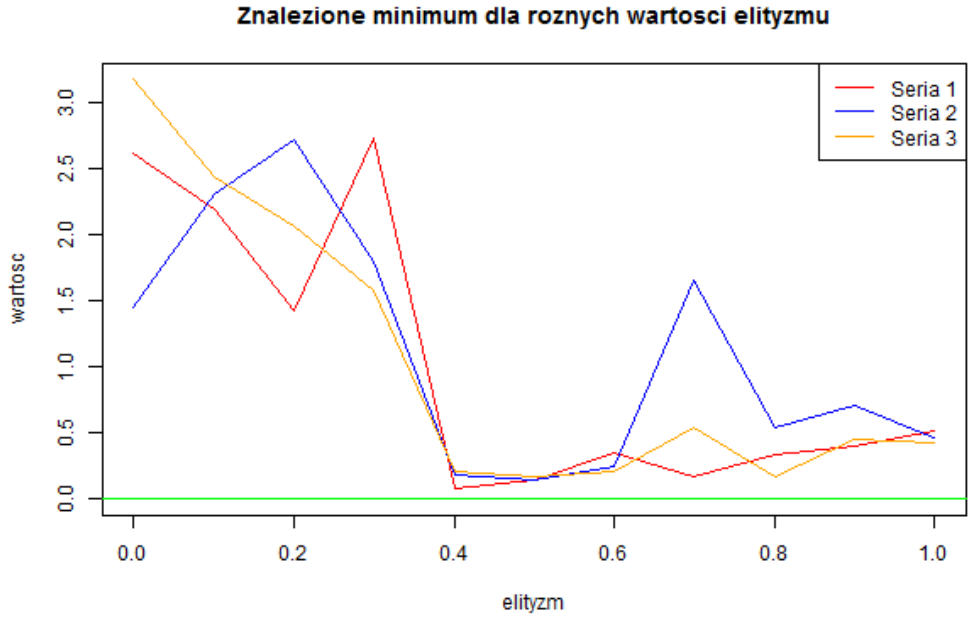
Wykres ten wyraznie obrazuje pozytywny wpływ zwiększenia populacji na jakość wyników. Najlepsze wyniki uzyskano dla populacji wynoszącej przynajmniej 50 jednostek.
Zauważalne jest również pogorszenie wyników w przedziale 65-80[todo: dlaczego].



Rysunek 11: Wartość znalezionego optimum w zależności od ilości iteracji

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

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



Rysunek 12: Wartość znalezionego optimum w zależności od przyjętego elityzmu

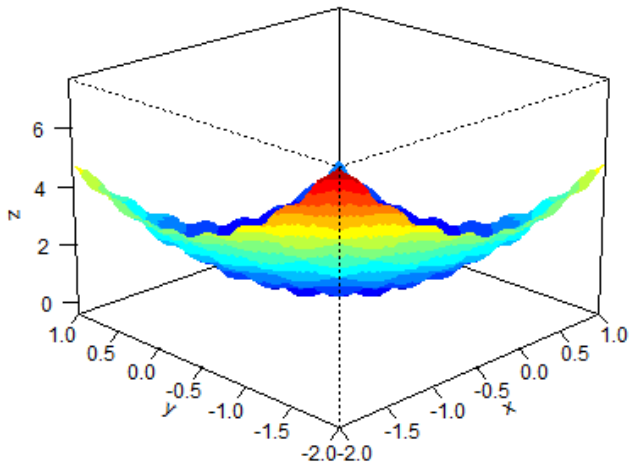
W przypadku funkcji Gulf elitzym ma znaczący wpływ na otrzymywane wyniki. W celu ich optymalizacji wymanaja jest wartość elityzmu na poziomie przynajmniej 0.4.

Dla wartości powyżej 0.6 wyniki zaczynają się pogarszać. [todo: dlaczego]

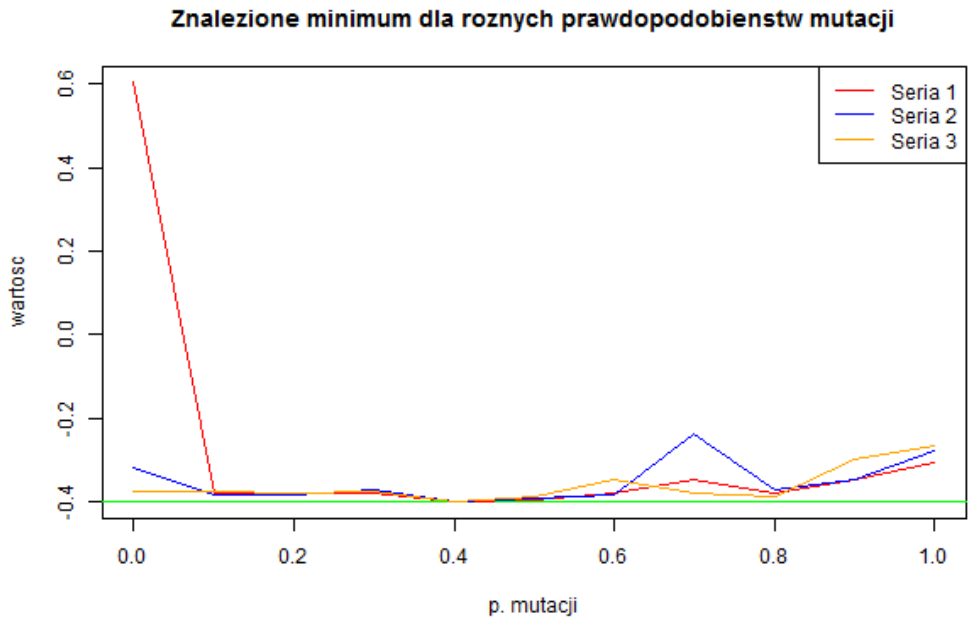
[todo: zmienic] Jak możemy zauważyć na ilustracji poniżej (rys. **??**) przedstawiona lokalizacja optimum nie jest poprawna, gdyż optymalizacji poddano wersję z 3 parametrami. Ogólnie rzecz biorąc gdyby 3 wymiar przedstawić w postaci gradientu kolorystycznego wtedy byłaby to poprawna lokalizacja niemniej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

3.3 CosMix4 (4 parametry)

http://www.gamsworld.org/performance/selconglobal/htm/selconglobal/CosMix4.htm

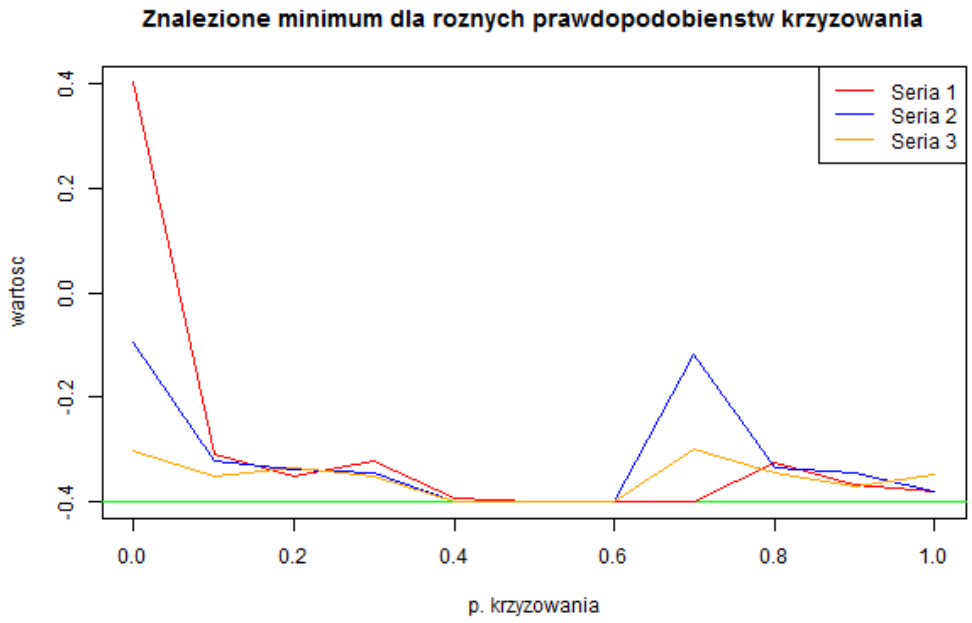


Rysunek 13: Wykres funkcji CosMix4 (d=4)



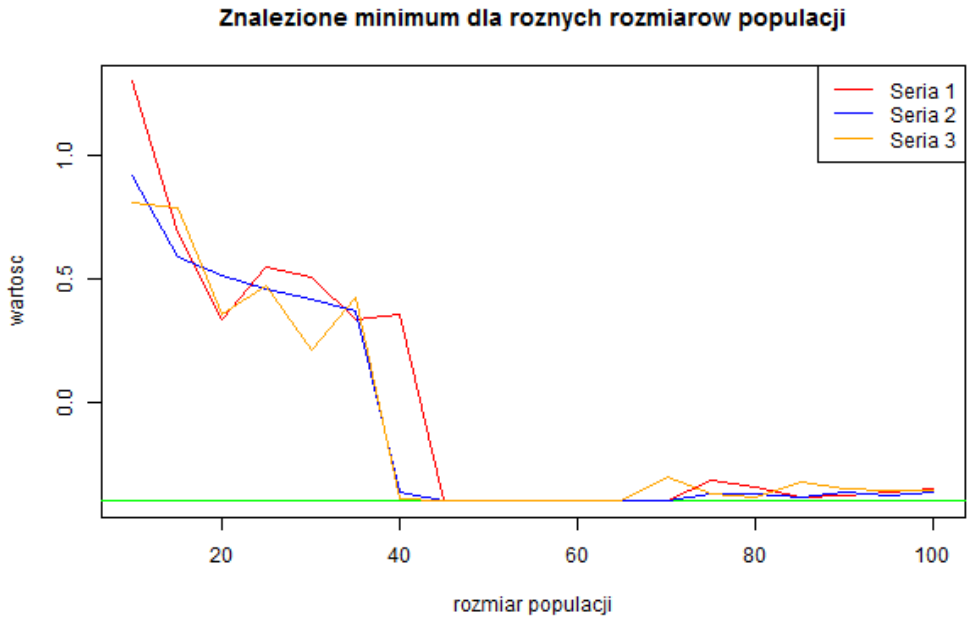
Rysunek 14: Wartość znalezionego optimum w zależności od prawdopodobieństwa mutacji

Ogólnie rzecz biorąc, niezależnie od pozostałych parametrów, jedyną niekorzystną sytuacją jest tu jednoczesne wyłączenie mutacji i krzyżowania co możemy zaobserwować na przykładzie serii 1-szej.

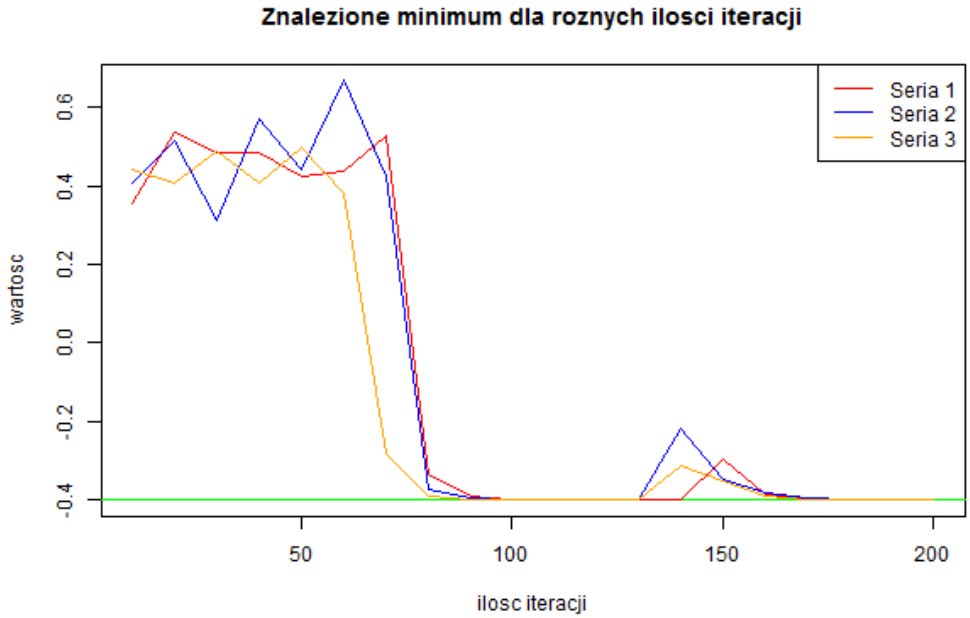


Rysunek 15: Wartość znalezionego optimum w zależności od prawdopodobieństwa krzyżowania

Podobnie jak na poprzednim wykresie (rys. 14) tak i na powyższym (rys. 15) ujawnia się niekorzystny wpływ wyłączenia mutacji i krzyżowania.

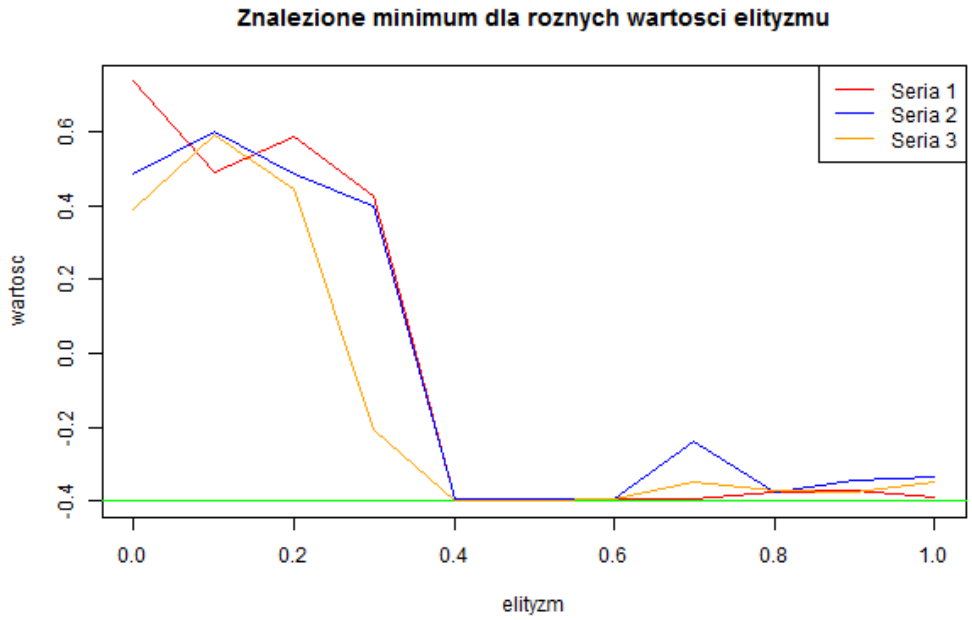


Rysunek 16: Wartość znalezionego optimum w zależności od rozmiarów populacji



Rysunek 17: Wartość znalezionego optimum w zależności od ilości iteracji

Na dwóch poprzedzających wykresach możemy zaobserwować, że domyślne wartości w postaci wielkości populacji w liczbie 50 i ilości iteracji równej 100 są wzajemnie optymalne.

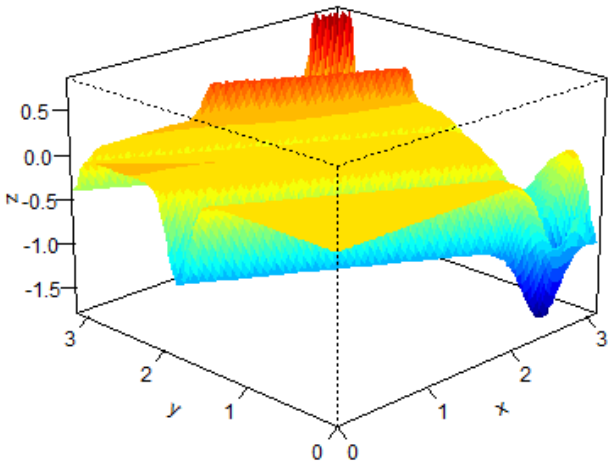


Rysunek 18: Wartość znalezionego optimum w zależności od przyjętego elityzmu

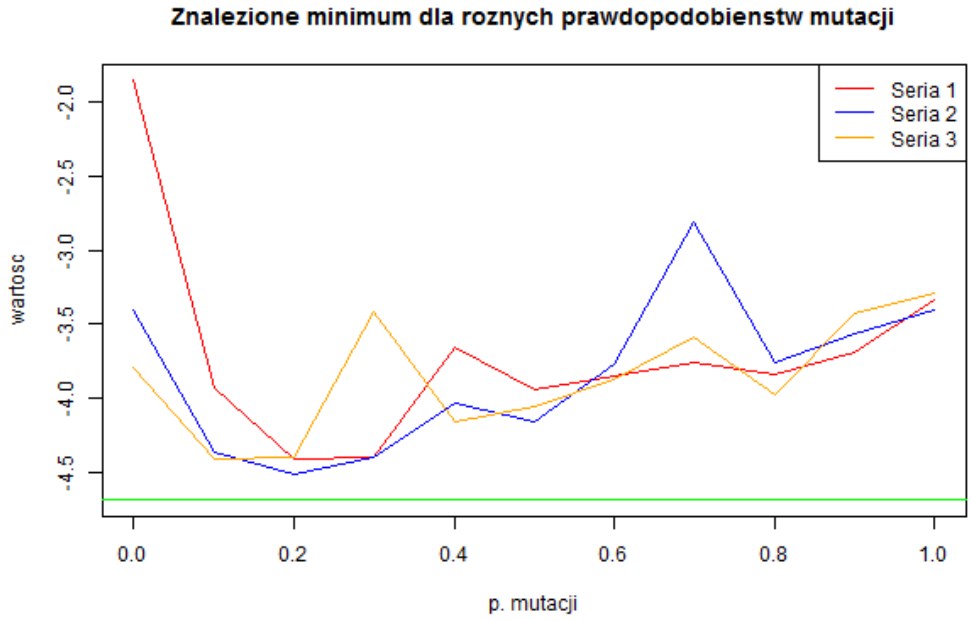
3.4 EMichalewicz (5 parametrów)

Poniżej zamieszczono wzór rozpatrywanej funkcji.

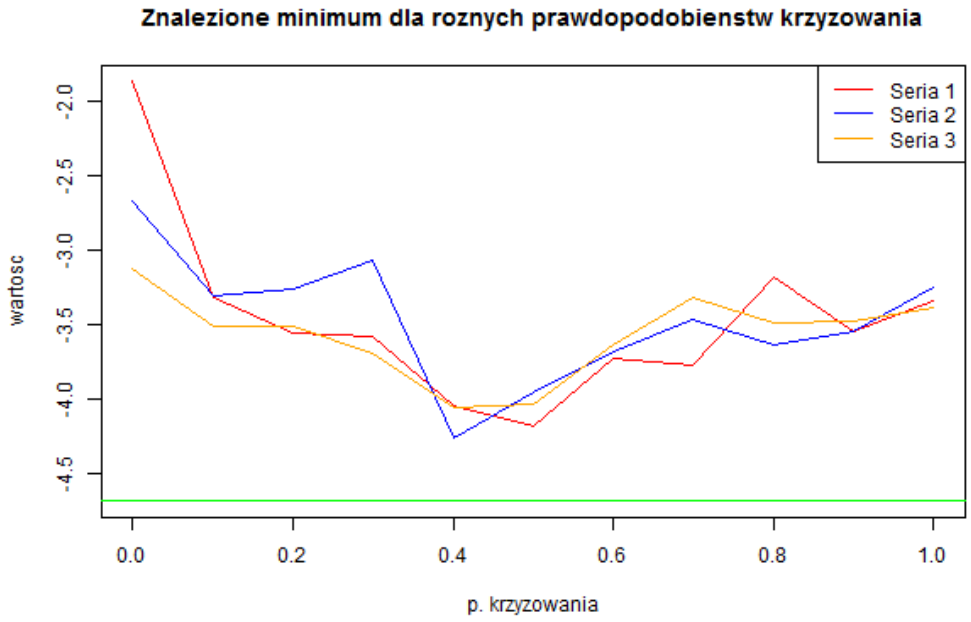
$$f(\boldsymbol{x}) = -\sum_{i=1}^{d} \sin(x_i) \sin^{2m}(\frac{ix_i^2}{\pi}) \tag{3}$$



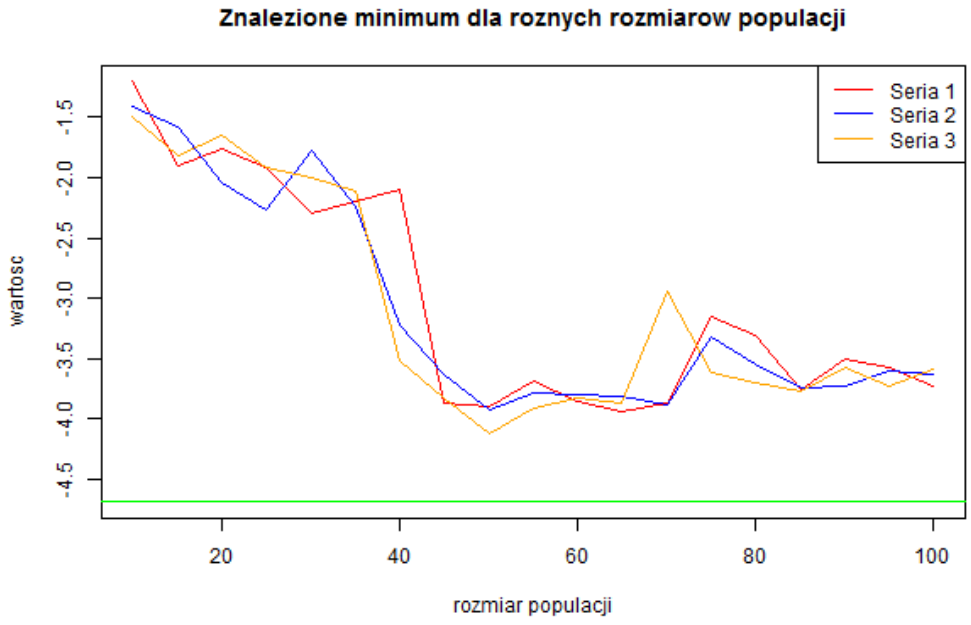
Rysunek 19: Wykres funkcji EMIchalewicz (d=5)



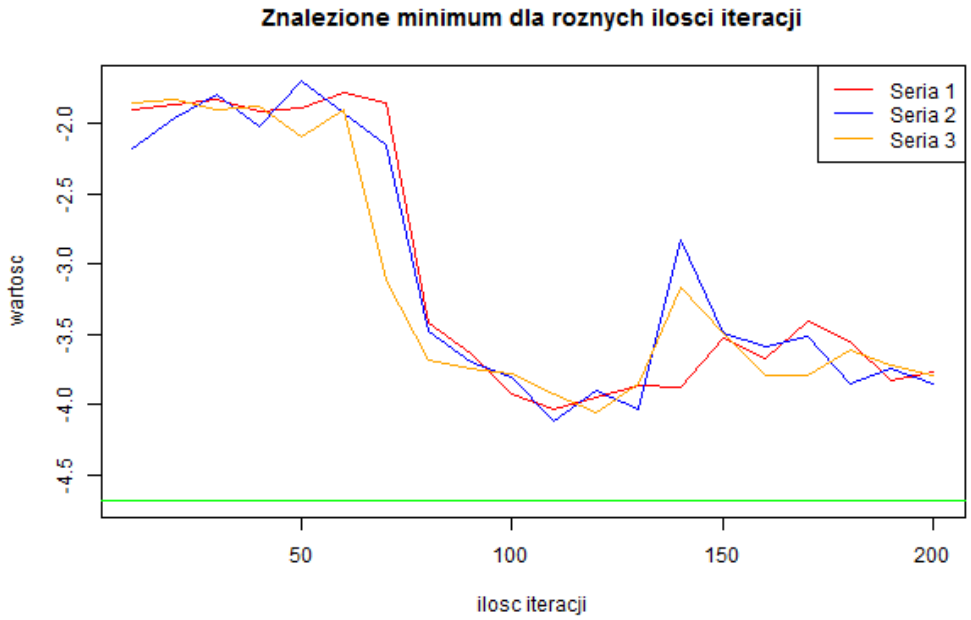
Rysunek 20: Wartość znalezionego optimum w zależności od prawdopodobieństwa mutacji



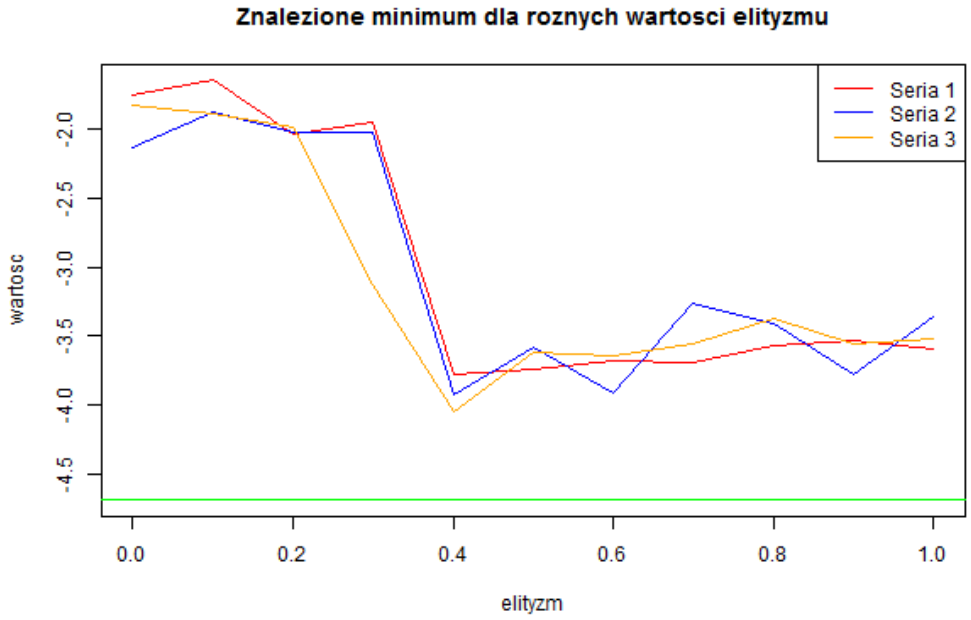
Rysunek 21: Wartość znalezionego optimum w zależności od prawdopodobieństwa krzyżowania



Rysunek 22: Wartość znalezionego optimum w zależności od rozmiarów populacji



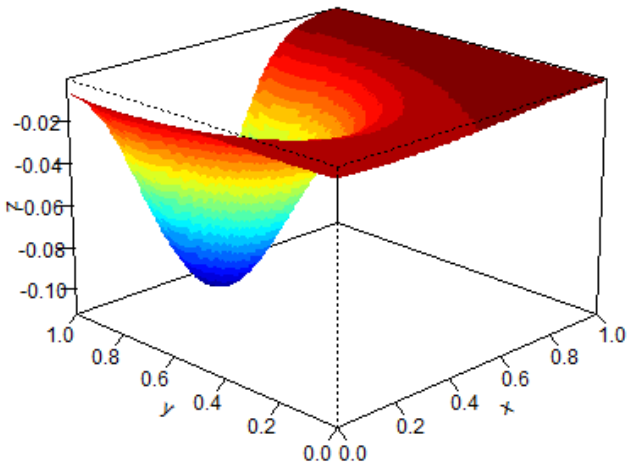
Rysunek 23: Wartość znalezionego optimum w zależności od ilości iteracji



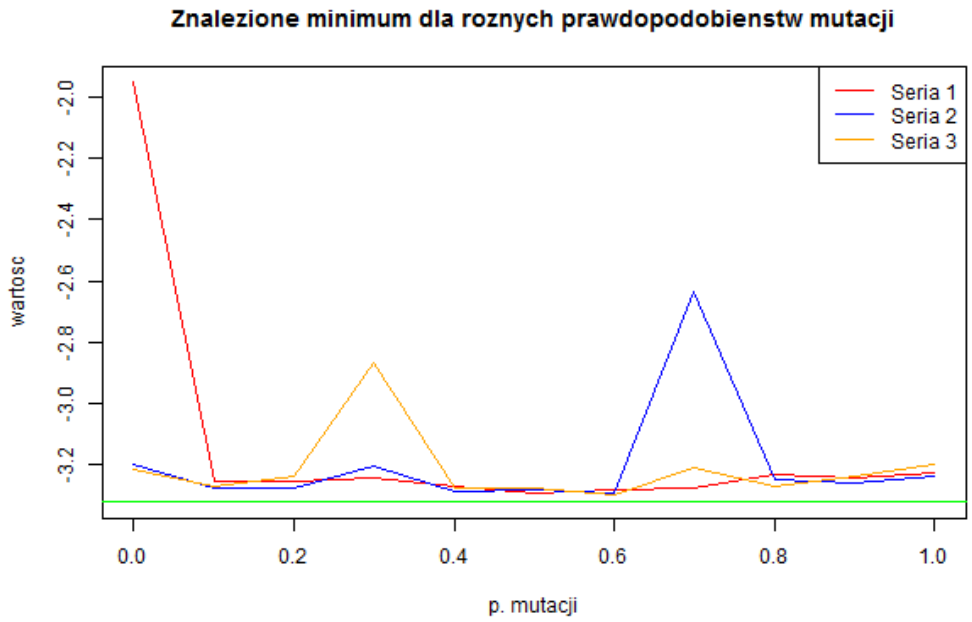
Rysunek 24: Wartość znalezionego optimum w zależności od przyjętego elityzmu

3.5 Hartman6 (6 parametrów)

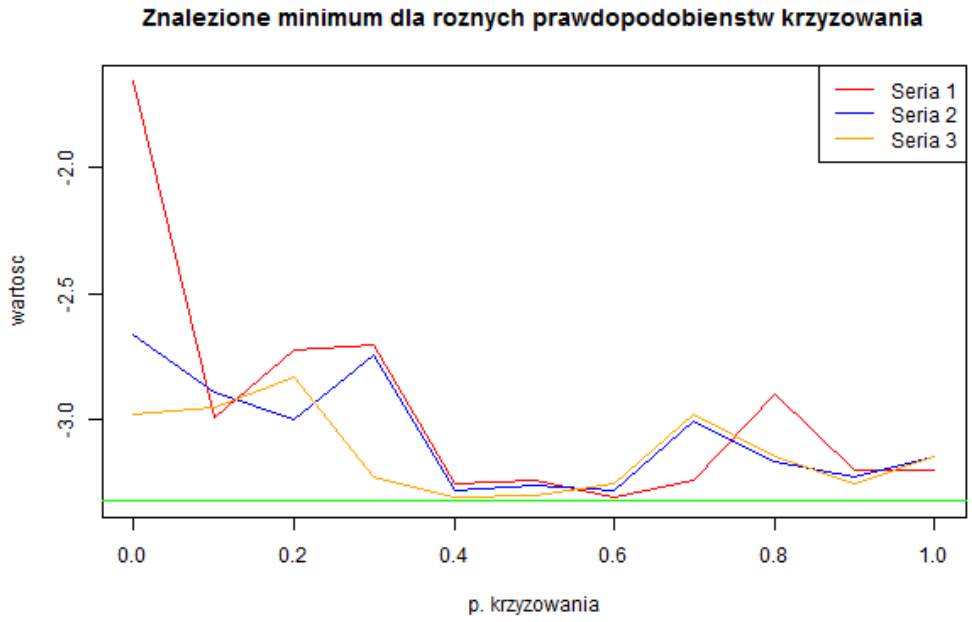
http://www.gamsworld.org/performance/selconglobal/htm/selconglobal/Hartman6.htm



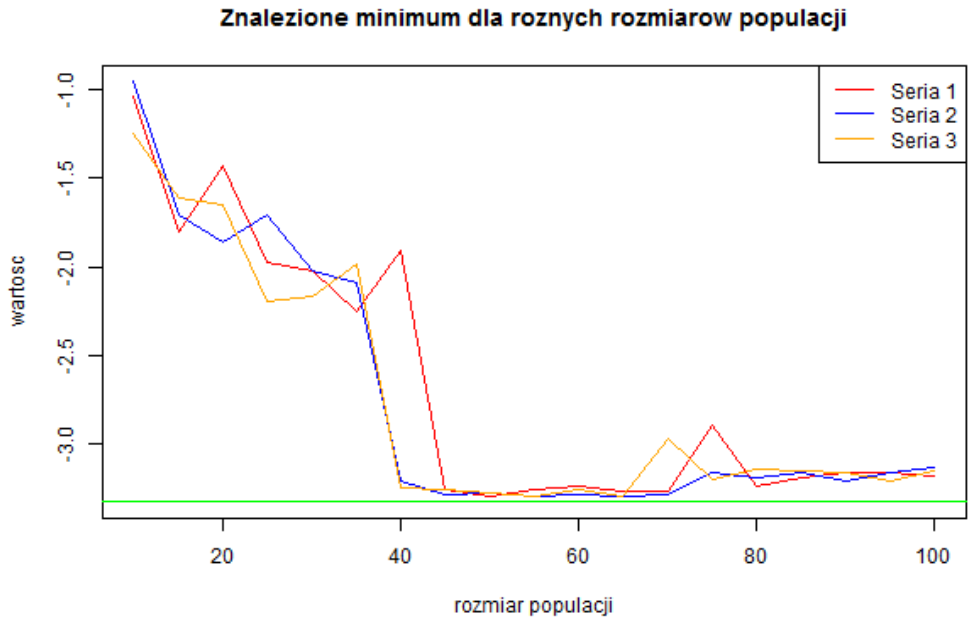
Rysunek 25: Wykres funkcji Hartman6 (d=6)



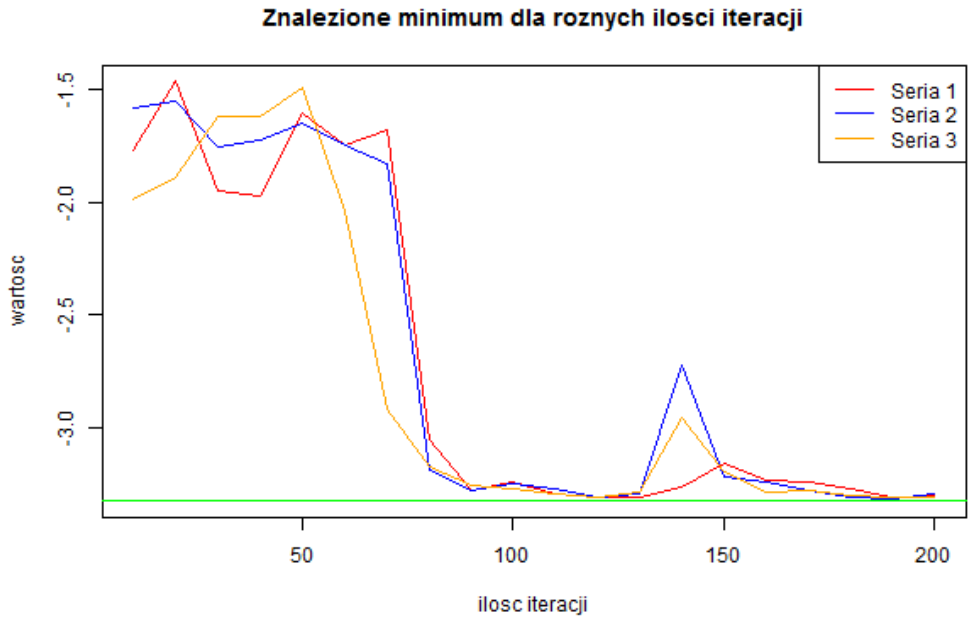
Rysunek 26: Wartość znalezionego optimum w zależności od prawdopodobieństwa mutacji



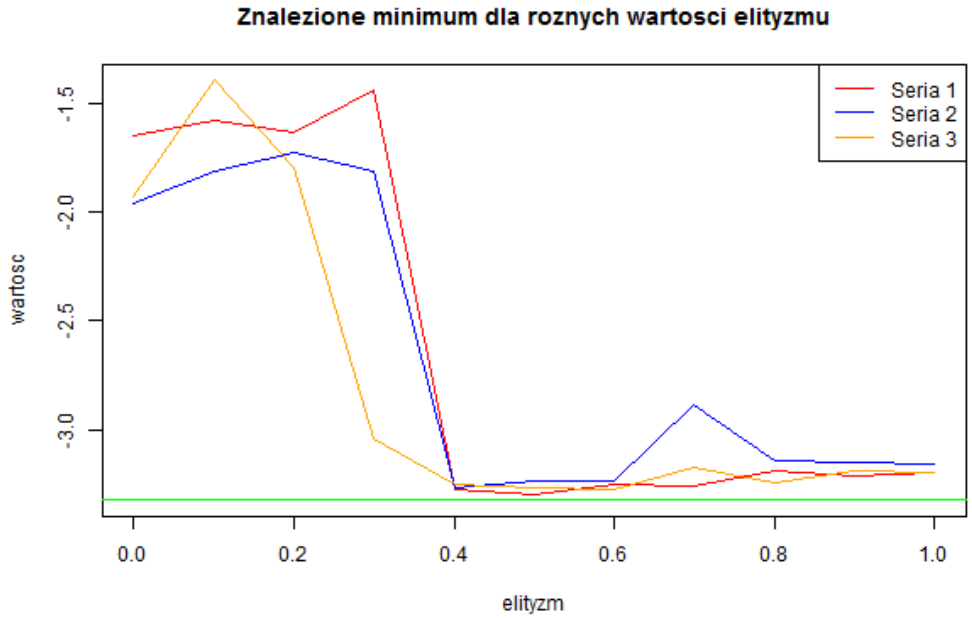
Rysunek 27: Wartość znalezionego optimum w zależności od prawdopodobieństwa krzyżowania



Rysunek 28: Wartość znalezionego optimum w zależności od rozmiarów populacji



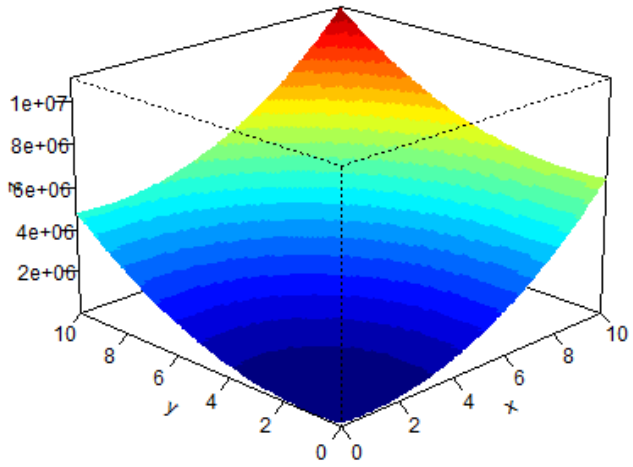
Rysunek 29: Wartość znalezionego optimum w zależności od ilości iteracji



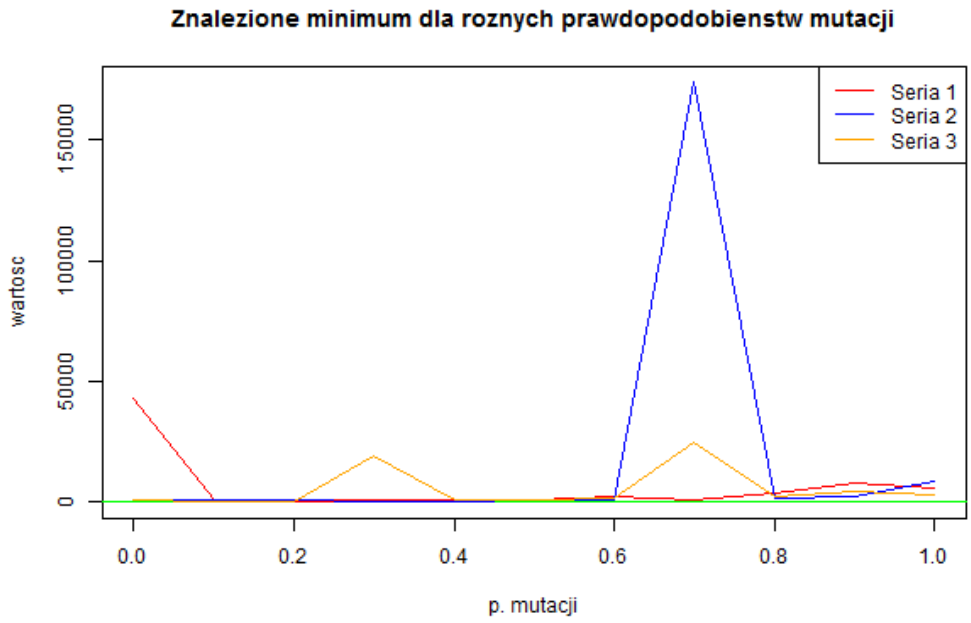
Rysunek 30: Wartość znalezionego optimum w zależności od przyjętego elityzmu

3.6 PriceTransistor (9 parametrów)

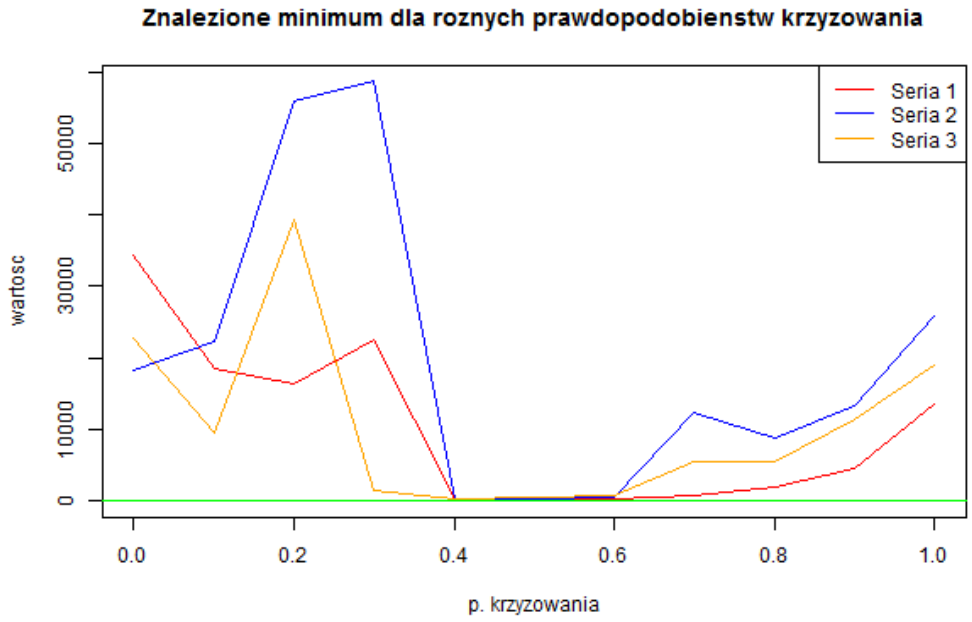
http://www.gamsworld.org/performance/selconglobal/htm/selconglobal/PriceTransistor.htm



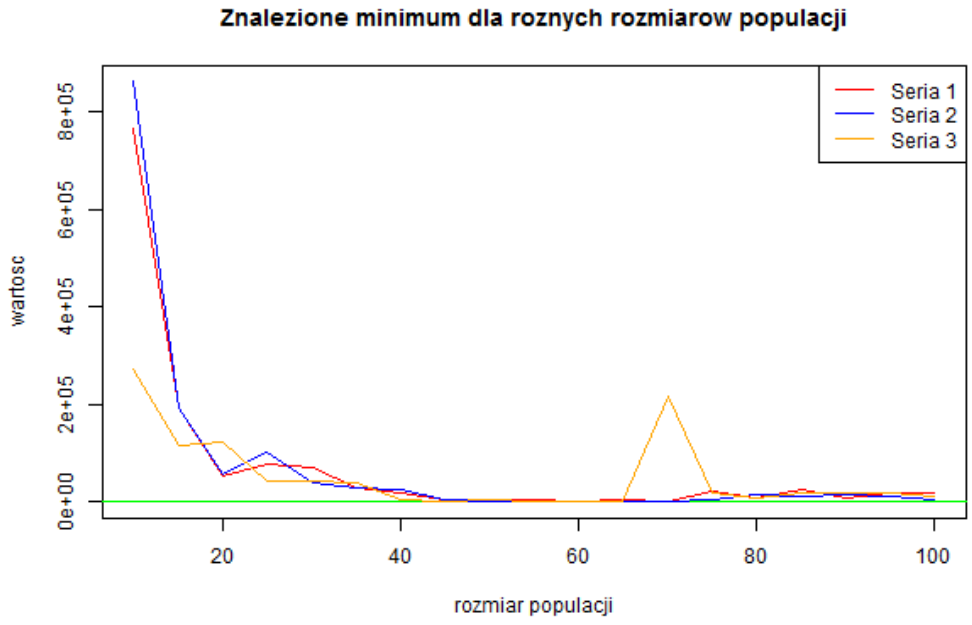
Rysunek 31: Wykres funkcji PriceTransistor (d=9)



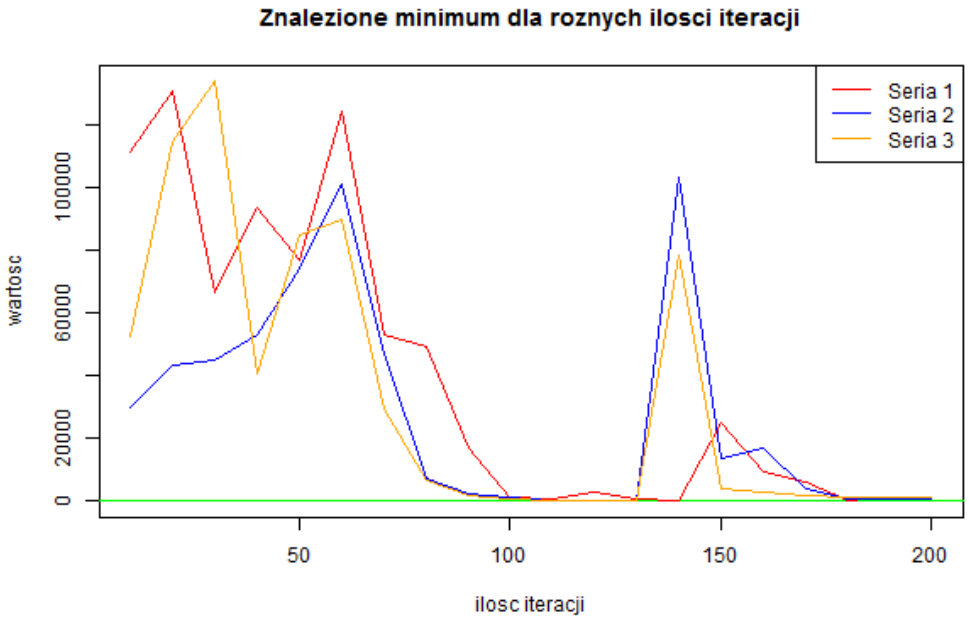
Rysunek 32: Wartość znalezionego optimum w zależności od prawdopodobieństwa mutacji



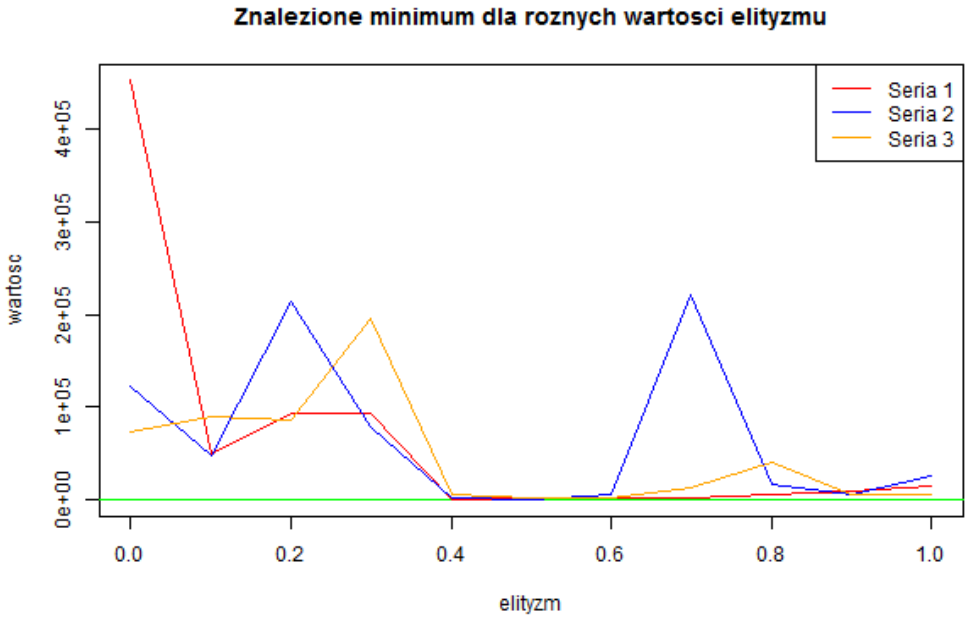
Rysunek 33: Wartość znalezionego optimum w zależności od prawdopodobieństwa krzyżowania



Rysunek 34: Wartość znalezionego optimum w zależności od rozmiarów populacji



Rysunek 35: Wartość znalezionego optimum w zależności od ilości iteracji

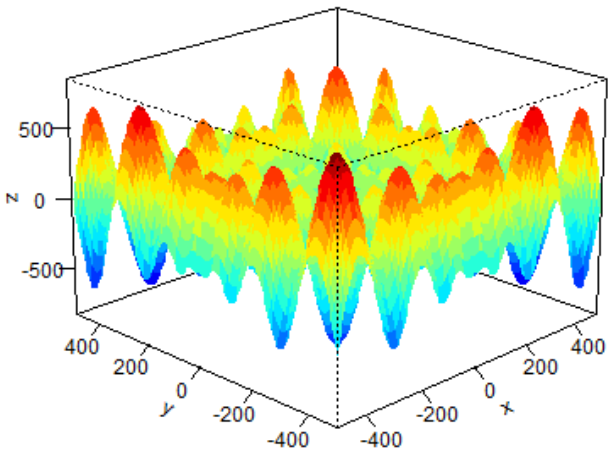


Rysunek 36: Wartość znalezionego optimum w zależności od przyjętego elityzmu

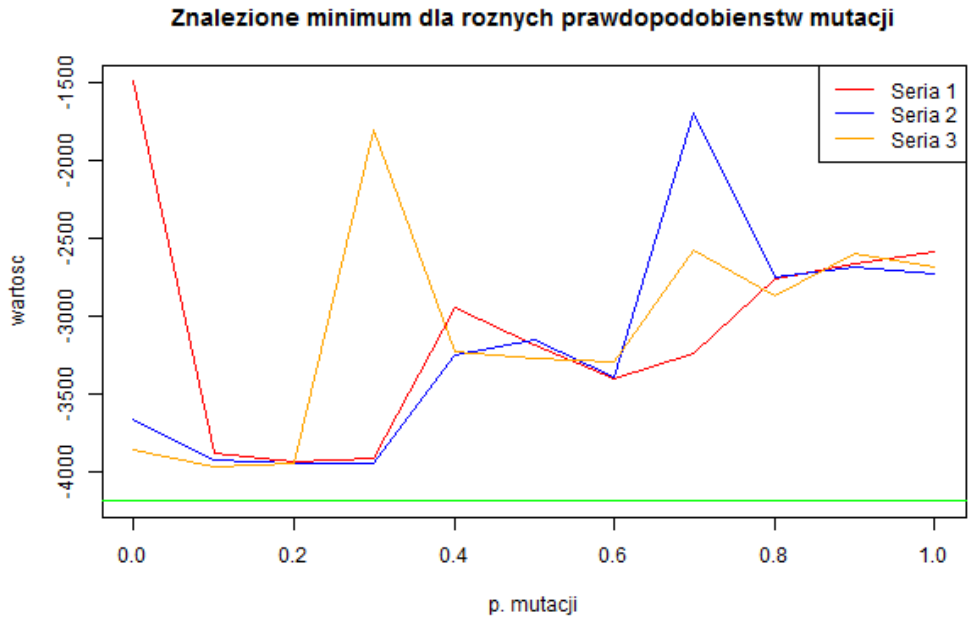
3.7 Schwefel (10 parametrów)

Poniżej zamieszczono wzór rozpatrywanej funkcji.

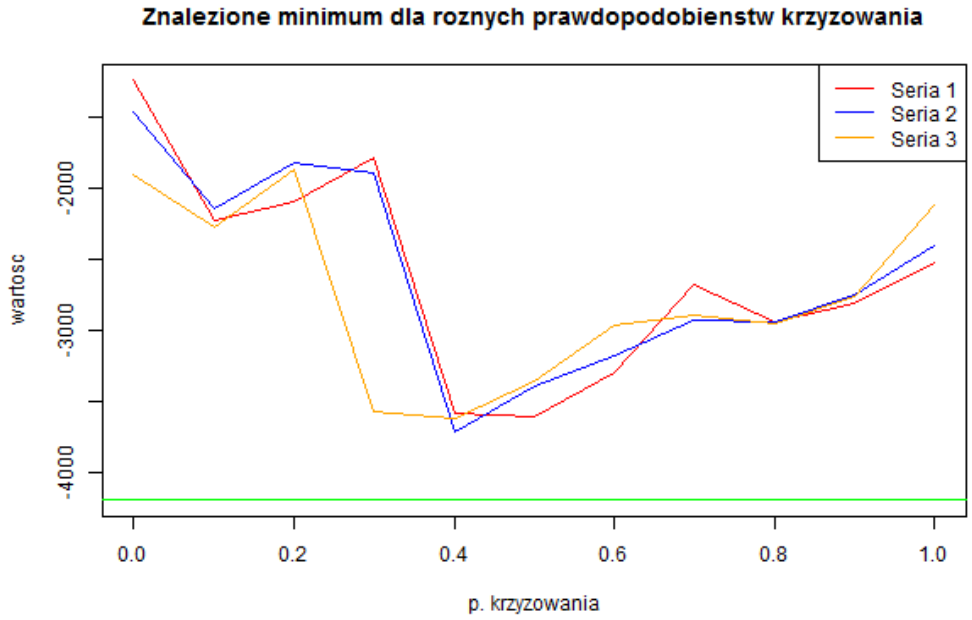
$$f(\boldsymbol{x}) = 418.9829d - \sum_{i=1}^{d} x_i \sin(\sqrt{|x_i|}) \tag{4}$$



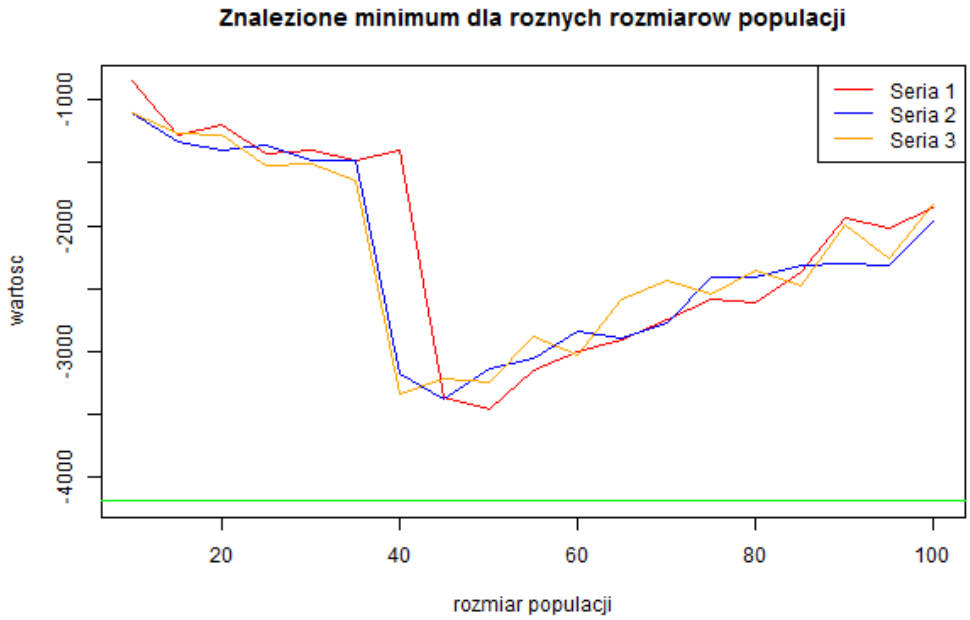
Rysunek 37: Wykres funkcji Schwefel (d=10) dla dwóch pierwszych wymiarów



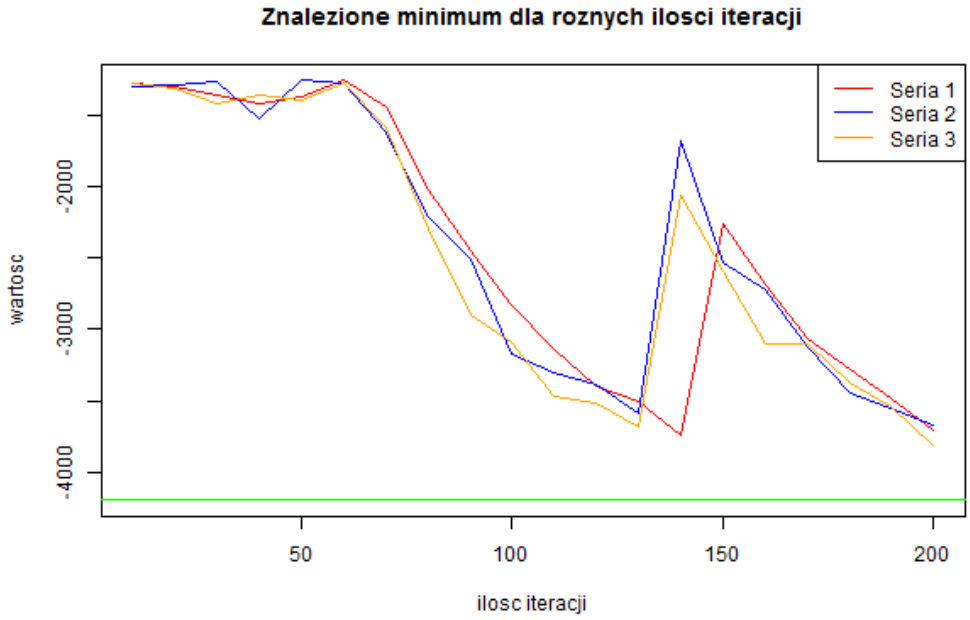
Rysunek 38: Wartość znalezionego optimum w zależności od prawdopodobieństwa mutacji



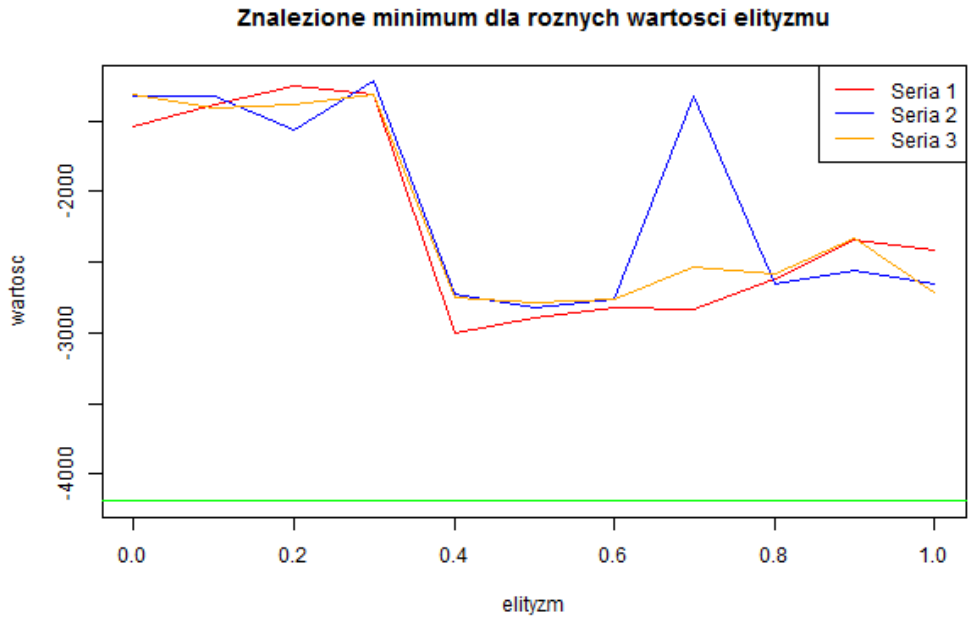
Rysunek 39: Wartość znalezionego optimum w zależności od prawdopodobieństwa krzyżowania



Rysunek 40: Wartość znalezionego optimum w zależności od rozmiarów populacji



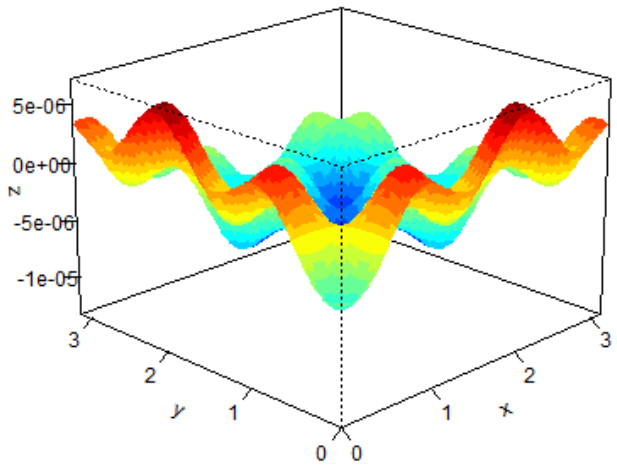
Rysunek 41: Wartość znalezionego optimum w zależności od ilości iteracji



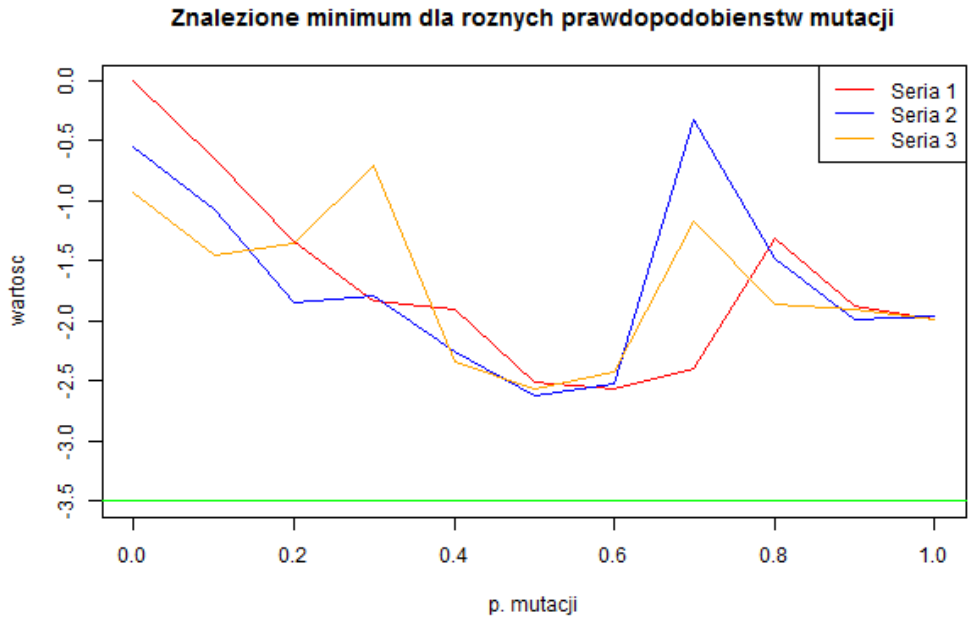
Rysunek 42: Wartość znalezionego optimum w zależności od przyjętego elityzmu

3.8 Zeldasine20 (20 parametrów)

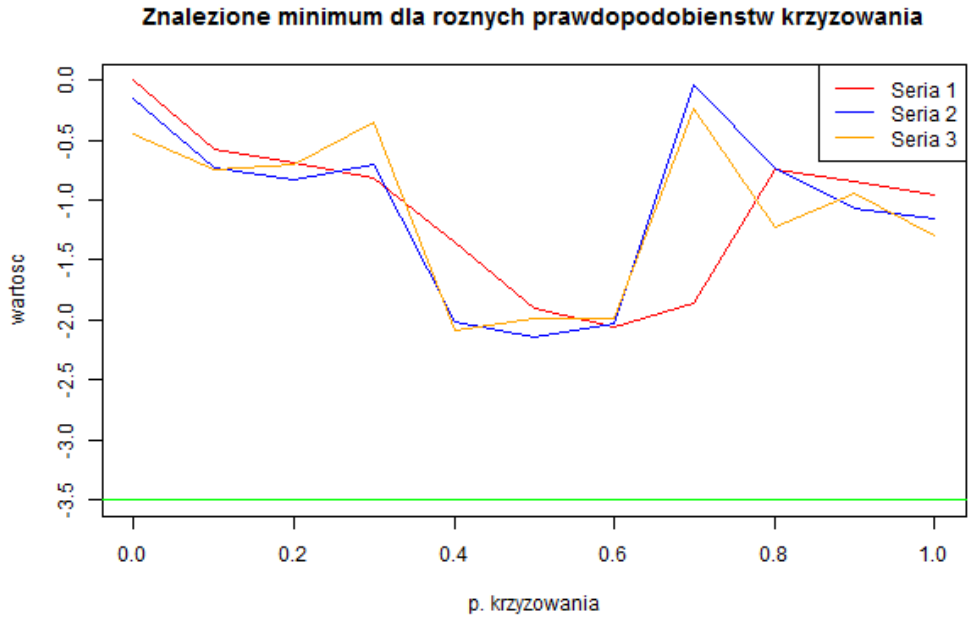
http://www.gamsworld.org/performance/selconglobal/htm/selconglobal/Zeldasine20.htm



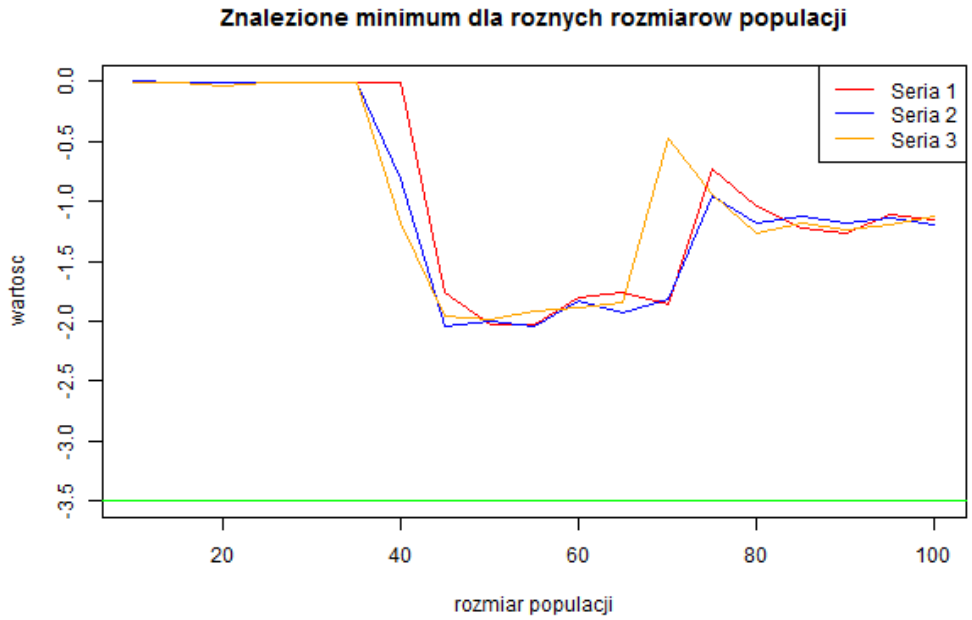
Rysunek 43: Wykres funkcji Zeldasine (d=20)



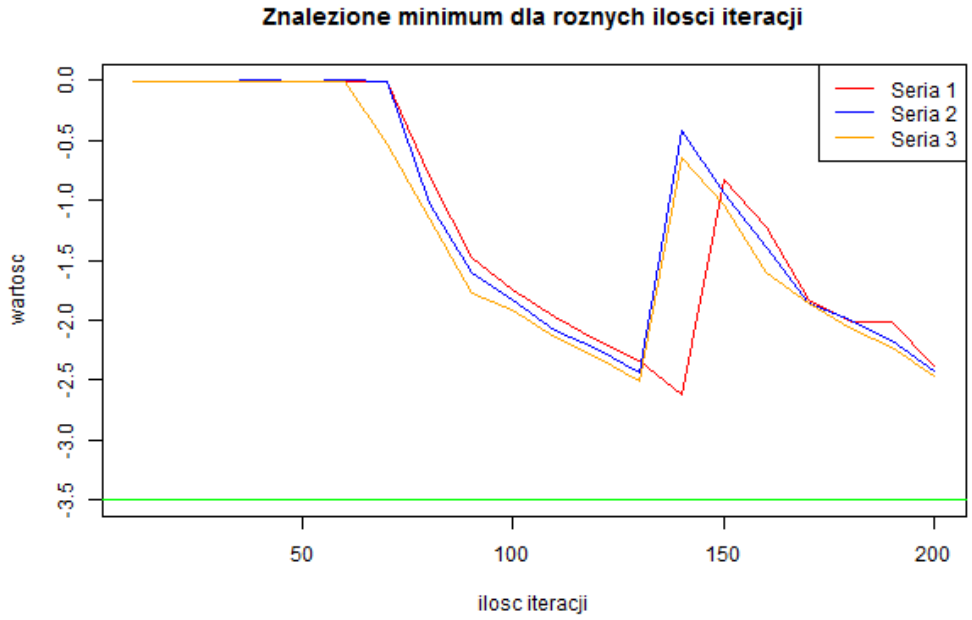
Rysunek 44: Wartość znalezionego optimum w zależności od prawdopodobieństwa mutacji



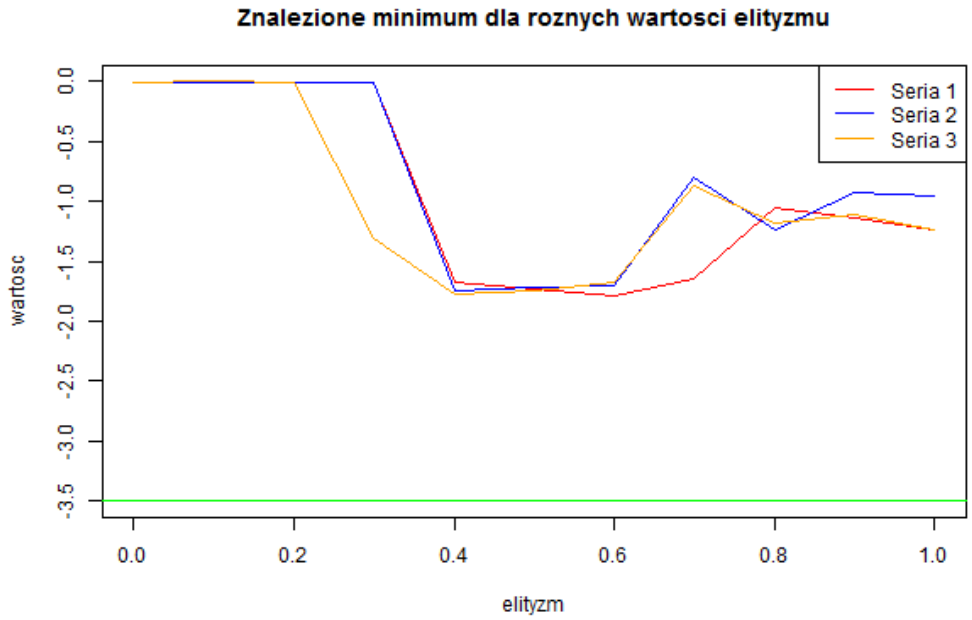
Rysunek 45: Wartość znalezionego optimum w zależności od prawdopodobieństwa krzyżowania



Rysunek 46: Wartość znalezionego optimum w zależności od rozmiarów populacji



Rysunek 47: Wartość znalezionego optimum w zależności od ilości iteracji



Rysunek 48: Wartość znalezionego optimum w zależności od przyjętego elityzmu

Analizując otrzymane wyniki możemy stwierdzić, że nie udało się otrzymać wartości bliskiej szukanemu optimum. Jest to związane z dużą przestrzenią poszukiwań. Musimy pamiętać, że rozpatrujemy tu funkcję o 20 parametrach.

4 Podsumowanie

Wartość prawdopodobieństwa mutacji i krzyżowania zdaje się odgrywać drugorzędną rolę. Istotne jednak by chociaż jedna z nich była włączona z prawdopodobieństwem większym niż 0.

Najlepszym ustawieniem dla elityzmu jest prawdopodobieństwo rzędu 0,5.

Literatura

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

[2] Luca Scrucca „A quick tour of GA” https://cran.r-project.org/web/packages/GA/vignettes/GA.html

[3] Surjanovic, S. & Bingham, D. (2013). Virtual Library of Simulation Experiments: Test Functions and Datasets. Retrieved April 3, 2017, from http://www.sfu.ca/ ssurjano.