Krzysztof Trajkowski

Przegląd pakietów do optymalizacji nieliniowej

2012

1. Wprowadzenie

W tym opracowaniu będą przedstawione wybrane algorytmy do optymalizacji nieliniowej które są dostępne w środowisku R. W pierwszej kolejności zostaną omówione funkcje `optim(stats)` oraz `fminsearch(pracma)` służące do szukania ekstremów bezwarunkowych funkcji wielowymiarowych. Następnie zaprezentujemy takie algorytmy jak `solve.QP(quadprog)`, `constrOptim(stats)` oraz `constrOptim.nl(alabama)`, `solnp(Rsolnp)` i `nloptr(nloptr)` które są wykorzystywane do rozwiązywania zadań programowania nieliniowego z ograniczeniami.

Gradient funkcji $f(\mathbf{x})$ w pewnym punkcie $\mathbf{x}_0$:

$$\nabla f(\mathbf{x_0}) = \left[\frac{\partial f(\mathbf{x}_0)}{\partial x_1}, \ldots, \frac{\partial f(\mathbf{x}_0)}{\partial x_n}\right] \tag{1.1}$$

Jeżeli gradient funkcji (1.1) jest zerowym wektorem to funkcja w punkcie $\mathbf{x_0}$ ma ekstremum (minimum, maksimum) lub punkt siodłowy. Jest to warunek konieczny istnienia ekstremum.

Hesjan:

$$\nabla^2 f(\mathbf{x_0}) = \begin{bmatrix} \frac{\partial^2 f(\mathbf{x_0})}{\partial x_1 \partial x_1} & \frac{\partial^2 f(\mathbf{x_0})}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(\mathbf{x_0})}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(\mathbf{x_0})}{\partial x_2 \partial x_1} & \frac{\partial^2 f(\mathbf{x_0})}{\partial x_2 \partial x_2} & \cdots & \frac{\partial^2 f(\mathbf{x_0})}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\mathbf{x_0})}{\partial x_n \partial x_1} & \frac{\partial^2 f(\mathbf{x_0})}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(\mathbf{x_0})}{\partial x_n \partial x_n} \end{bmatrix} \tag{1.2}$$

Jeżeli wyznacznik macierzy Hessa (1.2) jest większy od zera to funkcja osiąga w punkcie $\mathbf{x}_0$ ekstremum. Jest to warunek wystarczający istnienia ekstremum.

2. Optymalizacja bez ograniczeń

Dzięki zastosowaniu funkcji `optim(stats)` możemy skorzystać z następujących metod optymalizacji:

- SANN – simulated-annealing (symulowanego wyżarzania),
- BFGS – Broyden, Fletcher, Goldfarb, Shanno,
- L-BFGS-B – Limited-memory BFGS (BFGS z ograniczoną pamięcią). Metoda L-BFGS-B dobrze sobie radzi w problemach optymalizacyjnych z dużą ilością zmiennych i prostych ograniczeniach,
- CG – conjugate gradients (gradientu sprzężonego). Do obliczania współczynnika kierunku β można zostosować następujące formuły:
  - — Fletcher-Reeves: `control=list(type=1)`,
  - — Polak-Ribiere: `control=list(type=2)`,
  - — Beale-Sorenson: `control=list(type=3)`.
- Nelder-Mead. Jest to domyślna metoda w funkcji `optim`.

Poniżej został przedstawiony przykład wyznaczenia ekstremum z wykorzystaniem funkcji `optim` oraz metody `Nelder-Mead`.

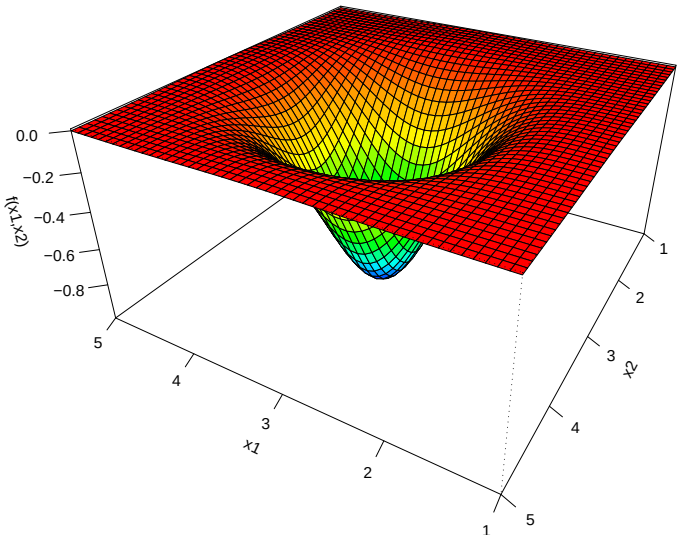
Funkcja Easom:

$$-\cos(x_1) \cdot \cos(x_2) \cdot \exp\Big(-(x_1-\pi)^2 - (x_2-\pi)^2\Big) \to \min$$

Wykres funkcji Easom (Rys. 2.1) został wygenerowany za pomocą kodu który jest widoczny na listingu 2.1.

Listing 2.1. Wykres

```
> x1= seq(1,5, length=50)
> x2= x1
> f= function(x1,x2) -cos(x1)*cos(x2)*exp(-(x1-pi)^2-(x2-pi)^2)
> z= outer(x1,x2,f)
        # kolory:
> library("RColorBrewer")
> jet.colors= colorRampPalette(c("blue","cyan","green", "yellow","orange","↩
    red"))
> nrz= nrow(z), ncz= ncol(z), nbcol= 100
> color= jet.colors(nbcol)
> zfacet= z[-1, -1] + z[-1, -ncz] + z[-nrz, -1] + z[-nrz, -ncz]
> facetcol= cut(zfacet, nbcol)
        # wykres:
> pdf("/.../Pulpit/3D.pdf",width=6,height=5.5,paper="special")
> par(mar=c(2,1,2,0))
> persp(x1, x2, z, expand= 0.5, col= color[facetcol], xlab="x1",ylab="x2",↩
    zlab="f(x1,x2)", theta= 210, phi= 30)
> dev.off()
```



Rys. 2.1. Funkcja Easom.

Listing 2.2. Optymalizacja - metoda Nelder-Mead

```
> f= function(x) -cos(x[1])*cos(x[2])*exp(-(x[1]-pi)^2-(x[2]-pi)^2)
> optim(c(3,3),f)
$par
[1] 3.141564 3.141629

$value
[1] -1

$counts
function gradient
      51       NA

$convergence
[1] 0

$message
NULL
```

Listing 2.3. Optymalizacja - metoda L-BFGS-B

```
> f= function(x) -cos(x[1])*cos(x[2])*exp(-(x[1]-pi)^2-(x[2]-pi)^2)
> optim(c(3,3), f, method="L-BFGS-B")
$par
[1] 3.141593 3.141593

$value
[1] -1

$counts
function gradient
       6        6

$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

Zastosowanie funkcji `fminsearch(pracma)` z wykorzystaniem metody Nelder-Mead.

Listing 2.4. Optymalizacja - metoda Nelder-Mead

```
> library(pracma)
> fminsearch(f, minimize= T, c(3,3))
$x
[1] 3.141588 3.141589

$fval
[1] -1
```

Na koniec sprawdzimy warunki istnienia ekstremum.

Listing 2.5. Warunki istnienia ekstremum

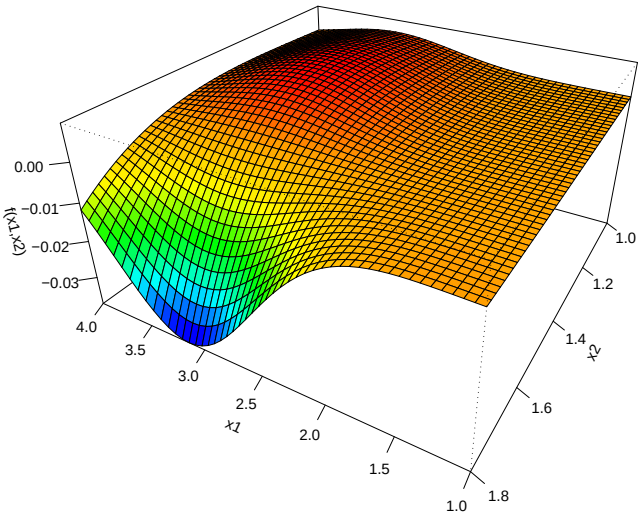
```
> x0= fminsearch(f, minimize= T, c(3,3))$x # punkty stacjonarne
> grad(f, x0) # gradient:
[1] -1.385556e-05 -1.162260e-05

> hessian(f, x0) # macierz Hessego:
     [,1] [,2]
[1,]    3    0
[2,]    0    3

> det(hessian(f, x0)) # wyznacznik macierzy Hessego:
[1] 9
```

Na podstawie funkcji `grad(pracma)`, `hessian(pracma)` oraz `det(stats)` możemy sformułować kilka wniosków:

- gradient funkcji `f` jest zerowy w punkcie `x0` – warunek konieczny istnienia ekstremum został spełniony (twierdzenie Fermata),
- minory główne macierzy Hessa są dodatnie (ich wartości są równe 3 oraz 9) czyli funkcja `f` osiąga w punkcie `x0` minimum (twierdzenie Sylvestera),
- wyznacznik macierzy Hessego jest dodatni czyli funkcja `f` osiąga w punkcie `x0` ekstremum – warunek wystarczający istnienia ekstremum.
  Kolejny przykład to wyznaczanie ekstremów funkcji:

$$f(x_1, x_2) = -\cos(x_1) \cdot \cos(x_2) \cdot \exp\Big(-(x_1-\pi)^2 - (x_2-\pi)^2\Big)$$

z prostymi ograniczeniami: $1 \leqslant x_1 \leqslant 4 \quad \text{oraz} \quad 1 \leqslant x_2 \leqslant 1,8$.



Rys. 2.2. Funkcja Easom na przedziale $x_1 \in \langle 1; 4\rangle$ i $x_2 \in \langle 1; 1, 8\rangle$.

Listing 2.6. Minimalizacja - metoda L-BFGS-B

```
> f= function(x) -cos(x[1])*cos(x[2])*exp(-(x[1]-pi)^2-(x[2]-pi)^2)
> optim(c(3,3), f, method="L-BFGS-B", lower=c(1,1), upper=c(4,1.8))
$par
[1] 3.141593 1.800000

$value
[1] -0.03756110

$counts
function gradient
       5        5

$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

Żeby obliczyć maksimum badanej funkcji wystarczy pomnożyć jej wzór przez -1.

Listing 2.7. Mksymalizacja - metoda L-BFGS-B

```
> f= function(x) cos(x[1])*cos(x[2])*exp(-(x[1]-pi)^2-(x[2]-pi)^2)
> optim(c(3,3), f, method="L-BFGS-B", lower=c(1,1), upper=c(4,1.8))
$par
[1] 3.141597 1.304995

$value
[1] -0.009005678

$counts
function gradient
      11       11

$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

3. Optymalizacja z ograniczeniami liniowymi

Optymalizacja funkcji kwadratowej z ograniczeniami liniowymi (szczególny przypadek programownia nieliniowego) nosi nazwę programowania kwadratowego. W środowisku R są dostępne specjalne algorytmy które rozwiązują tego typu zadania. Poniżej zostanie przedstawiona funkcja `solve.QP(quadprog)` która umożliwia rozwiązanie następującego zadania:

$$-\mathbf{d}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{D}\mathbf{x} \to \min \tag{3.1}$$

$$\mathbf{A}^T\mathbf{x} \geqslant \mathbf{b} \tag{3.2}$$

Załóżmy, że chcemy zminimalizować funkcję kwadratową:

$$3x_1^2 + 4x_2^2 + 2x_1x_2 - 2x_1 + 3x_2 \to \min$$

uwzględniając następujące ograniczenia liniowe:

$$\begin{aligned} x_1 + 2x_2 &= 3 \\ 2x_1 + x_2 &\leqslant 4 \\ x_1 \geqslant 0,\ x_2 &\geqslant 0 \end{aligned}$$

Listing 3.1. Rozwiązanie

```
> library(quadprog)
# funkcja:
> D= matrix(c(6,2, 2,8), 2,2)
> d= c(2,-3)
# ograniczenia:
> A= matrix(c(1,-2,1,0, 2,-1,0,1), 4,2)
> b= c(3,-4,0,0)
# rozwiązanie:
> solve.QP(Dmat=D,dvec=d,Amat=t(A),bvec=b,me=1)
$solution
[1] 1.0833333 0.9583333

$value
[1] 9.979167

$unconstrained.solution
[1]  0.5 -0.5

$iterations
[1] 2 0

$Lagrangian
[1] 6.416667 0.000000 0.000000 0.000000

$iact
[1] 1
```

Warto zaznaczyć, że funkcja `solveQP(quadprog)` jest często wykorzystywana do optymalizacji procesów ekonomicznych np. do budowy optymalnego składu portfela inwestycyjnego.

Powyższe zadanie programowania kwadratowego możemy rozwiązać także z wykorzystaniem funkcji `constrOptim(stats)`. Omawiany algorytm rozwiązuje zadanie które jest sformułowane w następujący sposób:

$$f(\mathbf{x}) \to \min \tag{3.3}$$

$$g_i(\mathbf{x}) \geqslant 0 \tag{3.4}$$

Należy w tym miejscu podkreślić, że funkcja $f(\mathbf{x})$ która jest optymalizowana za pomocą tego algorytmu nie musi być kwadratowa. Poniżej został przedstawiony przykład minimalizacji funkcji nieliniowej z ograniczeniami liniowymi.[1]

Funkcja nieliniowa:

$$\frac{1}{27\sqrt{3}}\Big((x_1-3)^2-9\Big)x_2^3 \to \min$$

Ograniczenia liniowe:

$$\begin{aligned}
&\frac{1}{\sqrt{3}}x_1 - x_2 \geqslant 0\\
&x_1 + \sqrt{3}x_2 \geqslant 0\\
&-x_1 - \sqrt{3}x_2 \geqslant -6\\
&x_1 \geqslant 0,\ x_2 \geqslant 0
\end{aligned}$$

Listing 3.2. Rozwiązanie

```
# funkcja:
> f=function(x){
+ x1=x[1]
+ x2=x[2]
+ (1/(27*sqrt(3)))*( (x[1]-3)^2-9 )*x[2]^3 }
# ograniczenia:
> A=rbind(c(1/sqrt(3),-1),c(1,sqrt(3)),c(-1,-sqrt(3)),c(1,0),c(0,1))
> b=c(0,0,-6,0,0)
# rozwiązanie:
> constrOptim(c(1,0.5),f,NULL,ui=A,ci=b)
$par
[1] 3.000000 1.732051

$value
[1] -1

$counts
function gradient
     141       NA

$convergence
[1] 0

$message
NULL

$outer.iterations
[1] 6

$barrier.value
[1] -0.001026577
```

Wartość `NULL` oznacza, że nie został wyznaczony przez nas gradient funkcji. Jeśli chcielibyśmy zmaksymalizować wartość funkcji to wtedy należy dodać do kodu napis `control=list(fnscale=-1))`.

[1] *Test examples for nonlinear programming codes - all problems from the Hock-Schittkowski-collection.* `www.klaus-schittkowski.de/refercs.htm`.

4. Optymalizacja z ograniczeniami nieliniowymi

Kolejna grupa algorytmów która zostanie przdstawiona służy do wyznaczania minimum (maksimum) funkcji nieliniowej z uwzględnieniem ograniczeń nieliniowych.[2] Należy zaznaczyć, że niektóre omawiane w tym rozdziale algorytmy umożliwiają wyznaczanie ekstremum funkcji $f(\mathbf{x})$ bez ograniczeń.

W pakiecie `alabama` zadanie optymalizacyjne może być sformułowane w następujący sposób:

$$f(\mathbf{x}) \to \min \tag{4.1}$$
$$g_i(\mathbf{x}) > 0 \tag{4.2}$$
$$h_i(\mathbf{x}) = 0 \tag{4.3}$$

Funkcja:

$$x_1 x_4 (x_1 + x_2 + x_3) + x_3 \to \min$$

ograniczenia:

$$x_1^2 + x_2^2 + x_3^2 + x_4^2 = 40$$
$$x_1 x_2 x_3 x_4 \geqslant 25$$
$$1 \leqslant x_{1,2,3,4} \leqslant 5$$

Listing 4.1. Rozwiązanie

```
> library(alabama)
Loading required package: numDeriv
# funkcja:
> fn= function(x) x[1]*x[4]*(x[1]+x[2]+x[3])+x[3]
# równości:
> heq= function(x) {
+ h= rep(NA,1)
+ h[1]= x[1]^2+x[2]^2+x[3]^2+x[4]^2-40
+ return(h)
+ }
# nierówności:
> hin= function(x){
+ h= rep(NA,9)
+ h[1]= x[1]*x[2]*x[3]*x[4]-25
+ h[2]= x[1]-1
+ h[3]= -x[1]+5
+ h[4]= x[2]-1
+ h[5]= -x[2]+5
+ h[6]= x[3]-1
+ h[7]= -x[3]+5
+ h[8]= x[4]-1
+ h[9]= -x[4]+5
+ return(h)
+ }
> p0= c(1,5,5,1)
# rozwiązanie:
> ans= auglag(par=p0, fn=fn, hin=hin, heq=heq)
Outer iteration:  16
Min(hin):  -1.304974e-07 Max(abs(heq)):  2.983057e-09
par:  1 4.743 3.82115 1.37941
fval =   17.01
```

[2] W tym rozdziale będziemy wykorzystywać przykłady które są dostępne w artykule: *Test examples for nonlinear programming codes - all problems from the Hock-Schittkowski-collection*. Jest on dostępny pod adresem: `www.klaus-schittkowski.de/refercs.htm`.

Do rozwiązania powyższego zadania możemy zastosować również inne funkcje dostępne w środowisku R. Z wykorzystaniem biblioteki `Rsolnp` możemy rozwiązywać zadania opisane w następujący sposób:

$$f(\mathbf{x}) \to \min \tag{4.4}$$
$$L_g \leqslant g_i(\mathbf{x}) \leqslant U_g \tag{4.5}$$
$$h_i(\mathbf{x}) = U_h \tag{4.6}$$
$$L_x \leqslant \mathbf{x} \leqslant U_x \tag{4.7}$$

Funkcja:

$$(x_1 - 1)^2 + (x_1 - x_2)^2 + (x_3 - 1)^2 + (x_4 - 1)^4 + (x_5 - 1)^6 \to \min$$

ograniczenia:

$$x_1^2 x_4 + \sin(x_4 - x_5) - 2\sqrt{2} = 0$$
$$x_2 + x_3^4 x_4^2 - 8 - \sqrt{2} = 0$$

Listing 4.2. Rozwiązanie

```
> library(Rsolnp)
# funkcja:
> f=function(x) (x[1]-1)^2+(x[1]-x[2])^2+(x[3]-1)^2+(x[4]-1)^4+(x[5]-1)^6
# równości:
> eqn1=function(x){
+       z1=x[1]^2*x[4]+sin(x[4]-x[5])
+       z2=x[2]+x[3]^4*x[4]^2
+ return(c(z1,z2))
+ }
# punkty startowe:
> x0= c(2,2,2,2,2)
# rozwiązanie:
> o=solnp(x0, fun= f, eqfun= eqn1, eqB= c(2*sqrt(2), 8+sqrt(2)))

Iter: 1 fn: 0.8828       Pars:  1.38369 1.37782 1.73079 1.66398 0.56104
Iter: 2 fn: 0.3767       Pars:  1.16920 1.17421 1.49431 1.56262 0.61092
Iter: 3 fn: 0.2570       Pars:  1.16191 1.17514 1.39684 1.51389 0.61192
Iter: 4 fn: 0.2419       Pars:  1.16603 1.18184 1.38063 1.50627 0.61096
Iter: 5 fn: 0.2415       Pars:  1.16617 1.18211 1.38026 1.50604 0.61092
Iter: 6 fn: 0.2415       Pars:  1.16617 1.18211 1.38026 1.50604 0.61092
Iter: 7 fn: 0.2415       Pars:  1.16617 1.18211 1.38026 1.50604 0.61092
solnp--> Completed in 7 iterations
```

Dla funkcji `solnp(Rsolnp)` parametry dotyczące ograniczeń są opcjonalne (nie musimy ich podawać) a więc za pomocą omawianego algorytmu możemy rozwiązywać zadania optymalizacyjne bez ograniczeń. Poniżej zostanie przedstawiony przykład minimalizacji funkcji Bealea.

$$\Big(1,5 - x_1(1 - x_2)\Big)^2 + \Big(2,25 - x_1(1 - x_2^2)\Big)^2 + \Big(2,625 - x_1(1 - x_2^3)\Big)^2 \to \min$$

Listing 4.3. Rozwiązanie

```
# funkcja:
> f=function(x){
+ (1.5-x[1]*(1-x[2]))^2+(2.25-x[1]*(1-x[2]^2))^2+(2.625-x[1]*(1-x[2]^3))^2
+ }
# punkty startowe:
> x0=c(1,1)
```

```
# rozwiazanie:
> solnp(x0, fun=f)

Iter: 1 fn: 3.81e-13     Pars:  3.00000 0.50000
Iter: 2 fn: 3.81e-13     Pars:  3.00000 0.50000
solnp--> Completed in 2 iterations
$pars
[1] 2.9999987 0.4999997

$convergence
[1] 0

$values
[1] 1.420312e+01 3.810059e-13 3.810059e-13

$lagrange
[1] 0

$hessian
           [,1]      [,2]
[1,]   3.155075 -11.40960
[2,] -11.409599  45.91189

$ineqx0
NULL

$nfuneval
[1] 103

$outer.iter
[1] 2

$elapsed
Time difference of 0.07828093 secs
```

Na zakończenie warto także przedstawić pakiet `nloptr` zawierający bardzo szeroką gamę różnych algorytmów do rozwiązywania zadań optymalizacyjnych o postaci:

$$f(\mathbf{x}) \to \min \tag{4.8}$$
$$g_i(\mathbf{x}) \leqslant 0 \tag{4.9}$$
$$h_i(\mathbf{x}) = 0 \tag{4.10}$$
$$L_b \leqslant \mathbf{x} \leqslant U_b \tag{4.11}$$

Każda nazwa algorytmu jest poprzedzona stałą nazwą `NLOPT_` oraz dwiema literami:
- `LD_` (local, derivative) – lokalna optymalizacja z wykorzystaniem pochodnych,
- `LN_` (local, don't need derivatives) – lokalna optymalizacja bez wykorzystywania pochodnych,
- `GD_` (global, derivative) – globalna optymalizacja z wykorzystaniem pochodnych,
- `GN_` (global, don't need derivatives) – globalna optymalizacja bez wykorzystywania pochodnych.

Ostatni człon to nazwa algorytmu z jakiego chcemy skorzystać np. `NLOPT_LN_COBYLA` (Constrained Optimization BY Linear Approximations), `NLOPT_LD_MMA` (Method of Moving Asymptotes). Wszystkie nazwy dostępnych algorytmów możemy uzyskać wpisując komendę `nloptr.print.options("algorithm")`.

Listing 4.4. Lista dostępnych algorytmów

```
> library(nloptr)
```

```
> nloptr.print.options("algorithm")
algorithm
        possible values: NLOPT_GN_DIRECT, NLOPT_GN_DIRECT_L,
                         NLOPT_GN_DIRECT_L_RAND, NLOPT_GN_DIRECT_NOSCAL,
                         NLOPT_GN_DIRECT_L_NOSCAL,
                         NLOPT_GN_DIRECT_L_RAND_NOSCAL,
                         NLOPT_GN_ORIG_DIRECT, NLOPT_GN_ORIG_DIRECT_L,
                         NLOPT_GD_STOGO, NLOPT_GD_STOGO_RAND,
                         NLOPT_LD_SLSQP, NLOPT_LD_LBFGS_NOCEDAL,
                         NLOPT_LD_LBFGS, NLOPT_LN_PRAXIS, NLOPT_LD_VAR1,
                         NLOPT_LD_VAR2, NLOPT_LD_TNEWTON,
                         NLOPT_LD_TNEWTON_RESTART,
                         NLOPT_LD_TNEWTON_PRECOND,
                         NLOPT_LD_TNEWTON_PRECOND_RESTART,
                         NLOPT_GN_CRS2_LM, NLOPT_GN_MLSL, NLOPT_GD_MLSL,
                         NLOPT_GN_MLSL_LDS, NLOPT_GD_MLSL_LDS,
                         NLOPT_LD_MMA, NLOPT_LN_COBYLA, NLOPT_LN_NEWUOA,
                         NLOPT_LN_NEWUOA_BOUND, NLOPT_LN_NELDERMEAD,
                         NLOPT_LN_SBPLX, NLOPT_LN_AUGLAG, NLOPT_LD_AUGLAG,
                         NLOPT_LN_AUGLAG_EQ, NLOPT_LD_AUGLAG_EQ,
                         NLOPT_LN_BOBYQA, NLOPT_GN_ISRES
```

Funkcja:

$$\sin(x_1 + x_2) + (x_1 - x_2)^2 - 1,5x_1 + 2,5x_2 + 1 \to \min$$

ograniczenia:

$$\begin{bmatrix} -1,5 \\ -3 \end{bmatrix} \leqslant \mathbf{x} \leqslant \begin{bmatrix} 4 \\ 3 \end{bmatrix}$$

Do rozwiązania powyższego problemu zastosujemy algorytm BOBYQA (Bound Optimization BY Quadratic Approximation), który może być wykorzystywany do optymalizacji funkcji z prostymi ograniczeniami.

Listing 4.5. Rozwiązanie

```
# funkcja:
> f= function(x) sin(x[1]+x[2])+(x[1]-x[2])^2-1.5*x[1]+2.5*x[2]+1
# rozwiązanie:
> res= nloptr(x0=c(0,0), eval_f=f, lb=c(-1.5,-3), ub=c(4,3), opts= list("↩
    algorithm"="NLOPT_LN_BOBYQA", "xtol_rel"=1.0e-8))
> res

Call:
nloptr(x0 = c(0, 0), eval_f = f, lb = c(-1.5, -3), ub = c(4,
    3), opts = list(algorithm = "NLOPT_LN_BOBYQA", xtol_rel = 1e-08))


Minimization using NLopt version 2.2.4

NLopt solver status: 4 ( NLOPT_XTOL_REACHED: Optimization stopped because
xtol_rel or xtol_abs (above) was reached. )

Number of Iterations....: 39
Termination conditions:  xtol_rel: 1e-08
Number of inequality constraints:  0
Number of equality constraints:    0
Optimal value of objective function:  -1.91322295498104
Optimal value of controls: -0.5471975 -1.547198
```

Funkcja:

$$(x_1 - 10)^3 + (x_2 - 20)^3 \to \min$$

ograniczenia:

$$(x_1 - 5)^2 + (x_2 - 5)^2 - 100 \geqslant 0$$
$$-(x_2 - 5)^2 - (x_1 - 6)^2 + 82,81 \geqslant 0$$

$$\begin{bmatrix} 13 \\ 0 \end{bmatrix} \leqslant \mathbf{x} \leqslant \begin{bmatrix} 100 \\ 100 \end{bmatrix}$$

Tym razem zastosujemy algorytm COBYLA, który jest wykorzystywany do optymalizacji funkcji z ograniczeniami nierównościowymi. Jak już wcześniej zostało zaznaczone ograniczenia nierównościowe powinny mieć postać jak we wzorze (4.9). Zatem pierwsze dwie nierówności po przekształceniu będą miały postać:

$$-(x_1 - 5)^2 - (x_2 - 5)^2 + 100 \leqslant 0$$
$$(x_2 - 5)^2 + (x_1 - 6)^2 - 82,81 \leqslant 0$$

Listing 4.6. Rozwiązanie

```
# funkcja:
> f= function(x) (x[1]-10)^3+(x[2]-20)^3
# nierówności:
> ineq= function(x) {
+     c( -(x[1]-5)^2-(x[2]-5)^2+100, (x[2]-5)^2+(x[1]-6)^2-82.81 )
+ }
> lb= c(13,0)
> ub= c(100,100)
# punkty startowe:
> x0= c(20.1, 5.84)
# rozwiązanie:
> res= nloptr( x0=x0, eval_f=f, lb=lb, ub=ub, eval_g_ineq=ineq,
+    opts=list("algorithm"="NLOPT_LN_COBYLA", "xtol_rel"=1.0e-8))
> res

Call:
nloptr(x0 = x0, eval_f = f, lb = lb, ub = ub, eval_g_ineq = ineq,
    opts = list(algorithm = "NLOPT_LN_COBYLA", xtol_rel = 1e-08))


Minimization using NLopt version 2.2.4

NLopt solver status: 4 ( NLOPT_XTOL_REACHED: Optimization stopped because
xtol_rel or xtol_abs (above) was reached. )

Number of Iterations....: 63
Termination conditions:  xtol_rel: 1e-08
Number of inequality constraints:  2
Number of equality constraints:    0
Optimal value of objective function:  -6961.81387558015
Optimal value of controls: 14.095 0.8429608
```