POLITECHNIKA RZESZOWSKA
im. Ignacego Łukasiewicza
WYDZIAŁ MATEMATYKI I FIZYKI STOSOWANEJ

Optymalizacja nieliniowa

Projekt 2, zastosowanie metody gradientu sprzężonego Hestenesa-Stiefela do poszukiwania minimum funkcji

Maciej Żak FS0-DI

Rzeszów , 17.01.2023

Spis treści

Wstęp Teoretyczny

Metody gradientów sprzężonych - narodziły się jako antidotum na podstawową wadę metod Cauchy'ego i Newtona. Mianowicie obie te metody w i-tej iteracji, minimalizuji funkcję f w kierunku d (i) a w kolejnej iteracji w kierunku d (i+1), nie zachowując jednak optymalności względem poprzedniego kierunku. Metody gradientów sprzężonych, optymalizując funkcję celu w kierunku d (i+1) biorą pod uwagę optymalność względem poprzedniego kierunku.

Metoda Hestenesa-Stiefela, znana również jako metoda gradientu sprzężonego z restartami, jest algorytmem optymalizacji, który jest używany do minimalizowania funkcji wielu zmiennych. Jest rozszerzeniem metody Fletchera-Reeves i jest używana do znajdowania minimum funkcji, która jest zdefiniowana na przestrzeni Hilberta.

Metoda opiera się na idei kierunków sprzężonych, które są ortogonalne wobec siebie w każdym kroku. Algorytm rozpoczyna się od początkowego przypuszczenia rozwiązania, a następnie iteracyjnie przesuwa się w kierunku ujemnego gradientu funkcji, dostosowując kierunek w każdym kroku do bycia sprzężonym z poprzednim kierunkiem.

Jedną z ważniejszych zalet metody Hestenesa-Stiefela w porównaniu z metodą Fletchera-Reeves jest to, że jest mniej podatna na zatrzymanie się w lokalnym minimum. Dodatkowo, Metoda Hestenesa-Stiefela ma lepsze właściwości globalnego zbieżności, co oznacza, że może szybciej zbiegać do prawdziwego minimum funkcji.

Kierynek poszukiwania wyznaczamy według zasady:

$$\begin{aligned} \mathbf{d}^{(0)} &= -\nabla f(\mathbf{x}^{(0)}), \\ \mathbf{d}^{(i+1)} &= -\nabla f(\mathbf{x}^{(i+1)}) + \beta^{(i+1)} \mathbf{d}^{(i)}, \text{ gdzie} \end{aligned}$$

$$\beta^{(i+1)} = \frac{\nabla f(\mathbf{x}^{(i+1)})^T \left(\nabla f(\mathbf{x}^{(i+1)}) - \nabla f(\mathbf{x}^{(i)}) \right)}{\left(\mathbf{d}^{(i)} \right)^T \left(\nabla f(\mathbf{x}^{(i+1)}) - \nabla f(\mathbf{x}^{(i)}) \right)}.$$

Opis Funkcji

Funkcja Beale'a jest testową funkcją optymalizacji wielu zmiennych. Jest ona zdefiniowana jako:

*f(x,y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + x*y^3)^2*

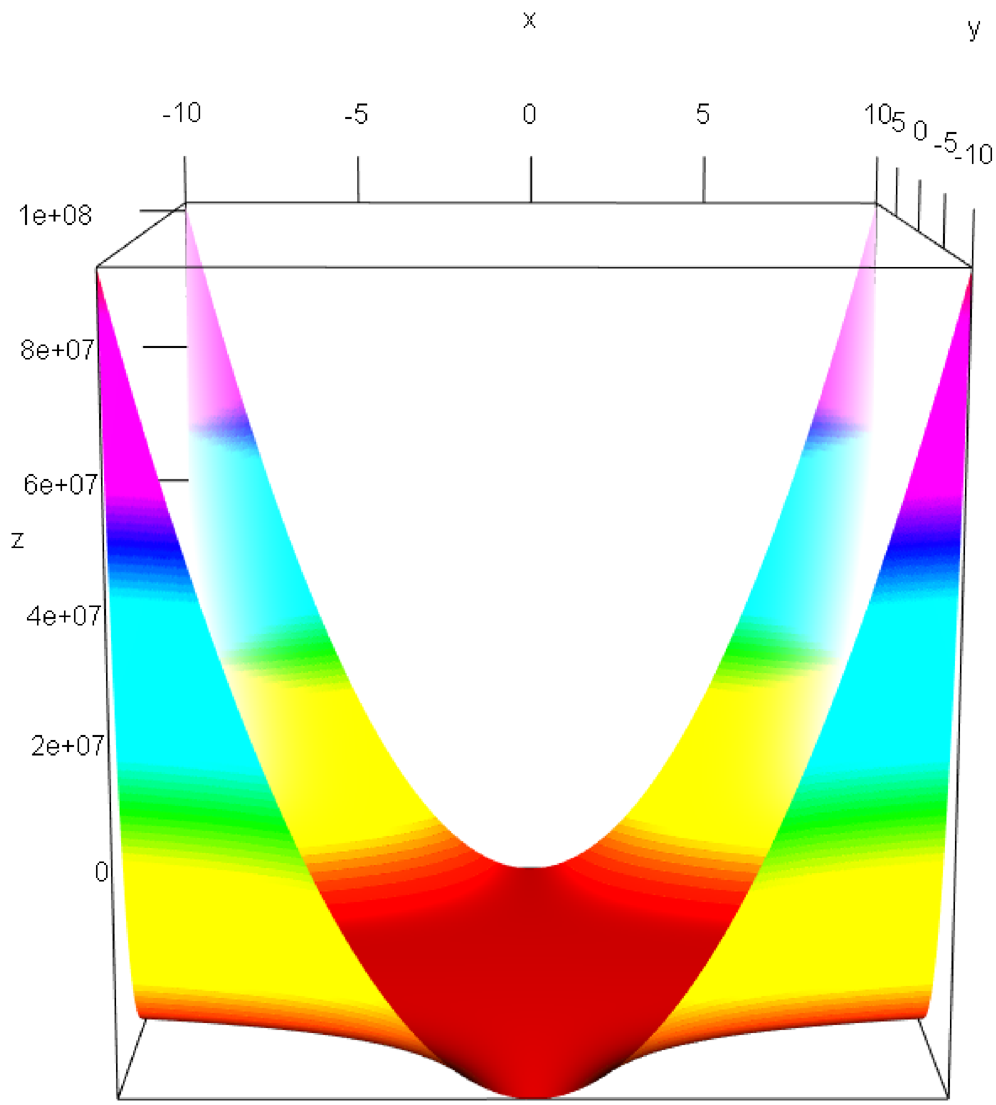
Funkcja ta ma globalne minimum w punkcie (3,0.5), gdzie wartość jest równa 0. Funkcja Beale'a została zaproponowana przez Howarda H. Beale'a w latach 60-tych XX wieku. Jest ona często stosowana jako przykład trudnego problemu optymalizacji, ponieważ jest łatwa do oceny, ale trudna do optymalizacji. Jest to często używana jako benchmark dla różnych metod optymalizacji, aby ocenić ich skuteczność i zbieżność.

Wizualizacja

Środowisko, w którym implementuję kod to RSTUDIO-2022.12.0-353 ,natomiast wersja języka R to R-4.2.2.

Do wizualizacji użyłem biblioteki „rgl" która służy do tworzenia wykresów w 3D

```
# Podaję Funkcję
Beale <- function(x, y) {
  (1.5 - x + x*y)^2 + (2.25 - x + x*y^2)^2 + (2.625 - x + x*y^3)^2
}


#Podaję zakres
x <- seq(-10, 10, by = 0.01)
y <- seq(-10, 10, by = 0.01)


# Obliczam wartosci
z <- outer(x, y,Beale )

# W zaleznosci od wartosci z dziele na przedzialy by uzyskac ladniejszy kolor
intervals <- cut(z, breaks = 100)

colors <- rainbow(length(levels(intervals)))[as.numeric(intervals)]


# Tworze wykres
persp3d(x, y, z, col = colors)
#dodaje zrodlo swiatla zeby byl jasniejszy
rgl.light(x=2, y=2, z=2, ambient="white", diffuse="white")
```



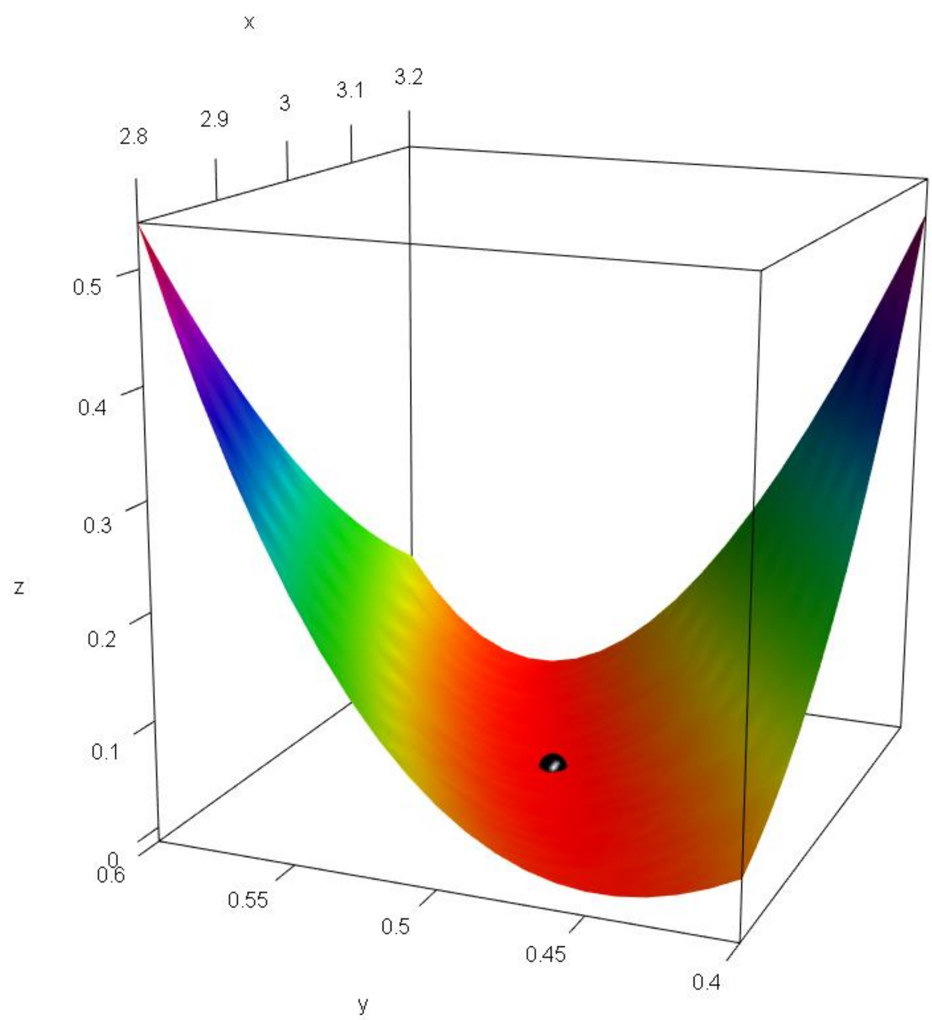
Na pierwszy rzut oka funkcja wygląda na niemal symetryczną. Wydaję się, że minimum powinno się znajdować w środku.

Jako że wiem jakie funkcja ma minimum zmniejszę zakres wykresu, aby widoczne było tylko miejsce, w którym znajduję się minimum.

```
# #Podaję zakres
x <- seq(2.8, 3.2, by = 0.01)
y <- seq(0.4, 0.6, by = 0.01)
```

Oraz rysuję czarny punkt w miejscu minimum:

```
minimum.point <- c(3,0.5,0)
rgl.spheres(minimum.point, radius=0.01,color="black")
```



Nawet po narysowaniu wykresu w bardzo bliskim otoczeniu minimum oraz zaznaczeniu czarną kropką, gdzie jest ,wciąż nie wydaje oczywiste, że jest dokładnie jedno minimum.

Implementacja kodu

Zaczynam od wczytania odpowiedniej biblioteki „numDeriv” która będzie niezbędna do obliczania gradientów. Kolejnym krokiem będzie zaimplementowanie funkcji pobranych ze strony domowej prowadzącego. Pobieram implementację metody ternary która posłuży do wyznaczania kroku w metodzie gradientowej. Pobieram również implementację metody Fletcher’a Reeves’a która po wprowadzeniu kilku zmian w obliczaniu bety zmieni się w metodę Hestenesa-Stiefela. Do metody gradientowej dodaję również komentarze funkcją „cat” które wyświetlą mi jak wraz z każdą iteracją zmieniały się wartości poszczególnych zmiennych

```
ternary <- function(f, lower, upper, tol) {
  f.lower <- f(lower)
  f.upper <- f(upper)
  while (abs(upper - lower) > 2 * tol) {
    x1 <- (2 * lower + upper) / 3
    f.x1 <- f(x1)
    x2 <- (lower + 2 * upper) / 3
    f.x2 <- f(x2)
    if (f.x1 < f.x2) {
      upper <- x2
      f.upper <- f.x2
    } else {
      lower <- x1
      f.lower <- f.x1
    }
  }
  return((upper + lower) / 2)
}
```

.

```
hestenes.stiefel <- function(f, x, tol) {
  beta <- 1
  d <- -grad(f,x)
  iter <- 0
  repeat {
    g <- function(a) {f(x + a * d)}
    step <- ternary(g,0,5,tol)
    new.x <- x + step * d
    if (dist(rbind(new.x,x)) < tol) {
    cat("Wektor Kierunku: ",d, "\nWielkość kroku: ", step,
        "\nBeta: ",beta,"\nIteracja: ", iter, "\nPunkt: ",new.x,"\n\n")
      return(new.x)
    }
    beta <- (grad(f,new.x) %*% (grad(f,new.x) - grad(f,x))) / (d %*% (grad(f,new.x) - grad(f,x)))
    d <- -grad(f,new.x) + as.vector(beta) * d
    x <- new.x
    iter <- iter + 1
    cat("Wektor Kierunku: ",d, "\nWielkość kroku: ", step,
        "\nBeta: ",beta,"\nIteracja: ", iter, "\nPunkt: ",new.x,"\n\n")
  }
}
```

Analiza wyników

Zapisanie funkcji, punktu początkowego oraz tolerancji. Jako pierwszy punkt przyjmujemy współrzędną(5,5).

```
funkcja<- function(x) {
  return((1.5 - x[1] + x[1]*x[2])^2 + (2.25 - x[1] + x[1]*x[2]^2)^2 + (2.625 - x[1] + x[1]*x[2]^3)^2)
}
tolerancja <- 1e-14
```

```
punk_poczatkowy <- c(5,5)
```

Dla współrzędnych początkowych (5 , 5) algorytm potrzebował 26 iteracji by znaleźć minimum znajdujące się w punkcie (3 , 0.5). Teraz Będę stopniowo zwiększał wartości współrzędnych początkowych by zobaczyć, kiedy program przestaje być w stanie odnaleźć minimum.

Warto zauważyć ,że jako tolerancję biorę dość małą liczbę. Kiedy tolerancja była większą liczbą algorytm ternary często zwracał błędy i miał problemy ze znajdowaniem minimum.

```
Wektor Kierunku:  0.02864188 0.005554417
Wielkość kroku:  0.03564591
Beta:  0.08366616
Iteracja:  22
Punkt:  2.998185 0.4996488

Wektor Kierunku:  -0.0001477514 -5.569275e-05
Wielkość kroku:  0.0635225
Beta:  -0.005335025
Iteracja:  23
Punkt:  3.000004 0.5000017

Wektor Kierunku:  -2.00342e-07 -3.889699e-08
Wielkość kroku:  0.0296859
Beta:  0.001274145
Iteracja:  24
Punkt:  3 0.5

Wektor Kierunku:  1.95974e-11 7.386941e-12
Wielkość kroku:  0.06447013
Beta:  -0.0001011547
Iteracja:  25
Punkt:  3 0.5

Wektor Kierunku:  5.069046e-14 9.838231e-15
Wielkość kroku:  0.02970212
Beta:  0.002410263
Iteracja:  26
Punkt:  3 0.5

Wektor Kierunku:  5.069046e-14 9.838231e-15
Wielkość kroku:  0.08322764
Beta:  0.002410263
Iteracja:  26
Punkt:  3 0.5

[1] 3.0 0.5
```

Ostatnimi współrzędnymi dla których jesteśmy w stanie policzyć minimum są współrzędne (30.1 , 10.8) ,
a policzenie minimum zajmuje 185 iteracji.

```
punk_poczatkowy <- c(30.1,10.8)
```

```
Wektor Kierunku:  2.440502 -4.709161
Wielkość kroku:  0.2241615
Beta:  0.1496101
Iteracja:  178
Punkt:  2.773139 0.5518794

Wektor Kierunku:  0.0748303 0.016937
Wielkość kroku:  0.02113384
Beta:  0.003656461
Iteracja:  179
Punkt:  2.824716 0.4523568

Wektor Kierunku:  1.248699 0.6120044
Wielkość kroku:  1.933205
Beta:  17.63479
Iteracja:  180
Punkt:  2.969378 0.4850995

Wektor Kierunku:  0.002478789 0.0005621427
Wielkość kroku:  0.0242046
Beta:  0.001778518
Iteracja:  181
Punkt:  2.999602 0.4999128

Wektor Kierunku:  -0.0003885053 -0.0002235737
Wielkość kroku:  0.1638929
Beta:  -0.1685194
Iteracja:  182
Punkt:  3.000009 0.5000049

Wektor Kierunku:  1.124972e-08 2.551134e-09
Wielkość kroku:  0.02195268
Beta:  -2.458927e-05
Iteracja:  183
Punkt:  3 0.5

Wektor Kierunku:  1.070822e-11 6.161792e-12
Wielkość kroku:  0.2682328
Beta:  0.001023636
Iteracja:  184
Punkt:  3 0.5

Wektor Kierunku:  -3.45977e-14 -7.844905e-15
Wielkość kroku:  0.02187638
Beta:  -0.002809103
Iteracja:  185
Punkt:  3 0.5

Wektor Kierunku:  -3.45977e-14 -7.844905e-15
Wielkość kroku:  0.2016682
Beta:  -0.002809103
Iteracja:  185
Punkt:  3 0.5

[1] 3.0 0.5
```

Jeżeli zwiększymy wartość którejkolwiek ze współrzędnych punktu początkowego o $\frac{1}{10}$, to algorytm nie będzie w stanie obliczyć minimum.

```
punk_poczatkowy <- c(30.2,10.8)
```

```
Wektor Kierunku:  1907115 6.188294e+14
Wielkość kroku:  9.087758e-15
Beta:  1.239882
Iteracja:  21
Punkt:  -2.320637e-07 224.5692

Wektor Kierunku:  2296400 8.680227e+14
Wielkość kroku:  9.087758e-15
Beta:  1.402685
Iteracja:  22
Punkt:  -2.147323e-07 230.193

Wektor Kierunku:  539551.4 3.898987e+14
Wielkość kroku:  9.087758e-15
Beta:  0.4491803
Iteracja:  23
Punkt:  -1.938631e-07 238.0814

Wektor Kierunku:  -485117.1 1.051971e+14
Wielkość kroku:  2.120004e-13
Beta:  0.2698062
Iteracja:  24
Punkt:  -7.947803e-08 320.7401

Wektor Kierunku:  1549745 1.985026e+15
Wielkość kroku:  9.087758e-15
Beta:  18.8696
Iteracja:  25
Punkt:  -8.388666e-08 321.6961

Wektor Kierunku:  38487035 5.862955e+16
Wielkość kroku:  1.817552e-14
Beta:  29.5359
Iteracja:  26
Punkt:  -5.571925e-08 357.7749

Wektor Kierunku:  -178704445522 1.804129e+20
Wielkość kroku:  9.087758e-15
Beta:  3077.167
Iteracja:  27
Punkt:  2.940416e-07 890.5861

Wektor Kierunku:  4.745558e+34 1.598132e+43
Wielkość kroku:  9.087758e-15
Beta:  8.858194e+22
Iteracja:  28
Punkt:  -0.001623729 1640439

Error in grad.default(f, new.x) :
 function returns NA at 2.37277908519043e+317.99066213449681e+39 distance from x.
```

Sprawdzamy również jakie najmniejsze wartośći punktu początkowego możemy wybrać, aby wciąż być w stanie obliczyć minimum.

Najmniejsze wartości jakie możemy wpisać aby obliczyć minimum to:

```
punk_poczatkowy <- c(-1.5,-1)
```

Jeżeli zwiększymy wartość którejkolwiek ze współrzędnych punktu początkowego o $\frac{1}{10}$, to algorytm nie będzie w stanie obliczyć minimum.

```
Wektor Kierunku:  0.5051973 0.04357577
Wielkość kroku:  0.03317733
Beta:  -0.01995173
Iteracja:  9
Punkt:  3.007449 0.5030687

Wektor Kierunku:  -0.1183744 -0.03403872
Wielkość kroku:  0.008965507
Beta:  -0.2383515
Iteracja:  10
Punkt:  3.011978 0.5034594

Wektor Kierunku:  0.002970057 0.0002415315
Wielkość kroku:  0.1017793
Beta:  -0.0237161
Iteracja:  11
Punkt:  2.99993 0.4999949

Wektor Kierunku:  -2.723105e-05 -7.886831e-06
Wielkość kroku:  0.02453473
Beta:  -0.009322818
Iteracja:  12
Punkt:  3.000003 0.5000008

Wektor Kierunku:  -6.994871e-09 -5.691802e-10
Wielkość kroku:  0.1077183
Beta:  0.0002425793
Iteracja:  13
Punkt:  3 0.5

Wektor Kierunku:  1.774217e-12 5.138621e-13
Wielkość kroku:  0.02499771
Beta:  -0.000257791
Iteracja:  14
Punkt:  3 0.5

Wektor Kierunku:  -1.563045e-15 -1.247158e-16
Wielkość kroku:  0.1082974
Beta:  -0.001029595
Iteracja:  15
Punkt:  3 0.5

Wektor Kierunku:  -1.563045e-15 -1.247158e-16
Wielkość kroku:  0.4261769
Beta:  -0.001029595
Iteracja:  15
Punkt:  3 0.5

[1] 3.0 0.5
```

Wnioski

W ramach tego projektu zbadano możliwości optymalizacji nieliniowej z wykorzystaniem metody Hestenesa-Stiefela. W tym celu zaimplementowano skrypt, który pozwala na znalezienie minimum dla różnych funkcji nieliniowych. Metoda Hestenesa-Stiefela uwzględnia poprzedni kierunek poszukiwania, co pozwala na bardziej precyzyjne znalezienie minimum. Wartość kierunku poszukiwania jest aktualizowana za każdym razem, co pozwala na uniknięcie zbędnego przeskakiwania.

Do wizualizacji wyników zastosowano trójwymiarowy wykres, który pozwala na lepsze zrozumienie zmian wartości funkcji w zależności od zmiennych. Dodatkowo, wykorzystano przezroczystość oraz źródło światła, co ułatwiło interpretację wyników.

W trakcie przeprowadzania obliczeń dla funkcji Beale'a, zauważono, że ważne jest odpowiednie dobranie zakresu dla zmiennych oraz zwiększenie precyzji kroku. Dzięki temu można zwiększyć dokładność znalezionego minimum.

Rezultaty projektu wskazują, że metoda Hestenesa-Stiefela jest skuteczna w optymalizacji nieliniowej i pozwala na precyzyjne znalezienie minimum dla różnych funkcji nieliniowych. Dodatkowo, wykorzystanie wizualizacji 3D pozwala na lepsze zrozumienie zmian wartości funkcji w zależności od zmiennych, co jest szczególnie ważne w przypadku funkcji wielu zmiennych.

Źródła

1. https://kpupka.v.prz.edu.pl/materialy-do-pobrania/optymalizacja-nieliniowa-6.html
2. https://pl.wikipedia.org/wiki/Metoda_gradientu_sprz%C4%99%C5%BConego
3. https://zeszyty-naukowe.wwsi.edu.pl/zeszyty/zeszyt13/Zastosowanie_algorytmu_optymalizacji_rojem_czastek_do_znajdowania_ekstremow_globalnych_wybranych_funkcji_%20testowych.pdf