Projekt Optymalizacja nieliniowa

Cz 1 Optymalizacja jednowymiarowa

Krawiec Piotr

07/11/2021

Zadanie

Dystans Ziemia-Mars zależy od ich pozycji na orbitach i zmienia się w czasie. Zadaniem jest obliczenie najmniejszego dystansu na jaki planety te zbliżą się. Dla ułatwienia orbity obu planet zostaną zamodelowane jako elipsy.

Pozycja planety w dowolnym punkcie czasu

Pozycje planet mogą być modelowane z pomocą liczb zespolonych[1]. Oto równanie pozwalające na symulację ruchu planety. Zostanie ono odpowienio przeskalowane poprzez dostosowanie parametru r. Model zakłada, że początkowy kąt między planetami wynosi 0 rad.

$$planet(t) = r * exp\left(2 \cdot \pi i r^{\frac{-3}{2}} t\right)$$

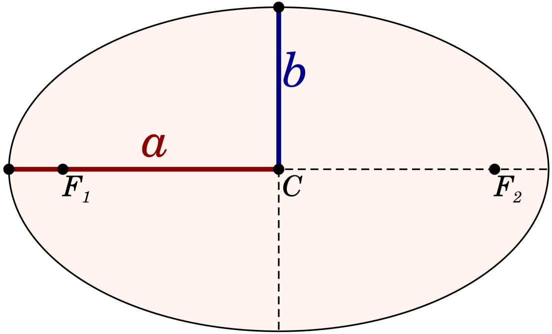
- r - półoś wielka orbity planety (elipsy)
- AU - jednostka astronomiczna 149 597 870 700 m
- t - czas[2]

[1] https://www.johndcook.com/blog/2015/10/24/distance-to-mars/
[2] jednostka nie ma znaczenia, ponieważ szukamy najmniejszej odległości

Elipsa

Oto model orbity. Dostosowując parametr r (na rys a), możemy modelować dowolną z planet.

- a - półoś wielka elipsy
- b - półoś mała elipsy



Równanie dla ziemi i marsa

Ponieważ półoś wielka orbity Ziemi wokół słońca to $1AU$ przyjmiemy parametr $r = 1AU$.

$$earth(t) = exp\left(2 \cdot \pi \cdot i \cdot t\right)$$

Ponieważ półoś wielka orbity Marsa wokół słońca to $1.524AU$ przyjmiemy parametr $r = 1.524AU$.

$$mars(t) = 1.524 * exp\left(2 \cdot \pi \cdot i \cdot (1.524)^{\frac{-3}{2}} \cdot t\right)$$

Równanie dla ziemi i marsa - kod w R

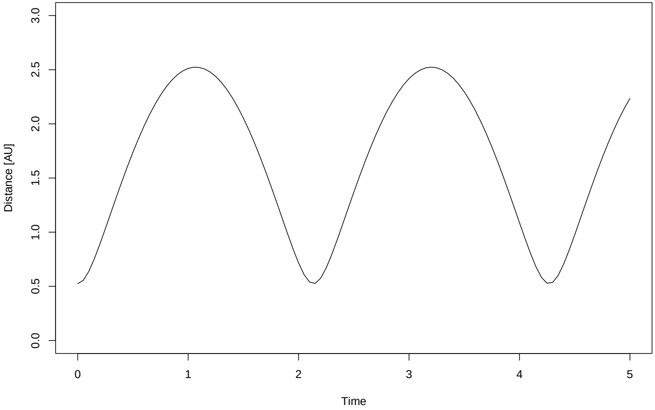
```
r = 1.524 # półoś wielka orbity Marsa w AU

earth <- function(t){ exp(2*pi*1i*t) }

mars <- function(t) { r*exp(2*pi*1i*(r**-1.5*t)) }
```

Odległość między planetami można wyznaczyć jako wartość bezwzględą z różnicy w ich pozycjach w czasie t.

$$f(t) = abs\left(mars(t) - earth(t)\right)$$

```
f <- function(x) { abs(mars(x) - earth(x)) }
```

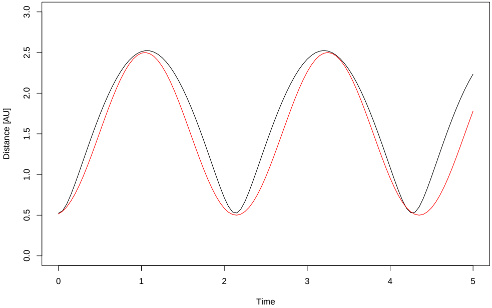
Wykres funkcji odległości planet



Rysunek 1: Wykres funkcji distance(t)

Wykres funkcji odległości planet - porównanie z sin

Przeskalowana funkcja $sin(x)$ na czerwono. Funkcja dystansu przypomina funkcję sinus, ale jak widać na wykresie poniżej nie są identyczne.



Rysunek 2: Wykres funkcji distance(t)

Metody bezgradientowe

Zadanie zostanie roziwązane korzystając z metody Fibonacciego. Do jej implementacji będzie potrzebny *Ciąg Fibonacciego*, który zdefiniowany jest jako:

$$F(0) = 0, F(1) = 1, F_n = F_{n-1} + F_{n-2}$$

Obliczenie Ciągu Fibonacciego

```r
# Obliczenie pierwszych 101 wyrazów ciągu fib
phil <- c(rep(0, 100))
phil[1:3] <- c(1,1,1)
for(i in c(3:length(phil))) {
  phil[i] = phil[i-1] + phil[i-2]}

phi <- function(i) {
  if(i <= 0) {return(0)}; # F(0) = 0
  # Obliczenie nowych elementów jeżeli wyjdziemy poza zakres
  if (i > length(phil)) {
    len <- length(phil)
    phil <- c(phil, rep(0, i - len))
    for(j in c(len:length(phil))) {phil[j] = phil[j-1]
      + phil[j-2]}
  }
  return(phil[i])
}
```

Metoda Fibonacciego

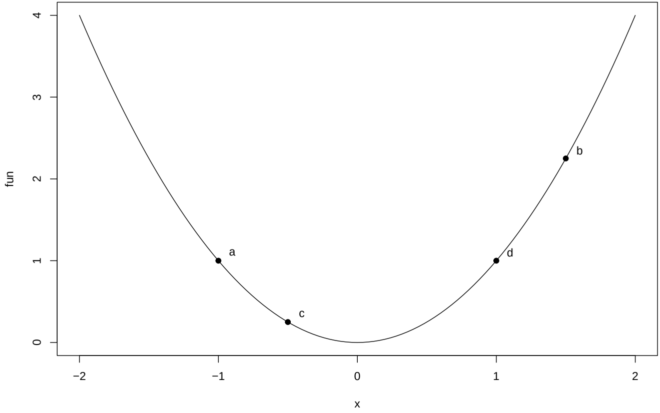
Metoda ta opiera się na metodzie zawężania początkowego przedziału poszukiwania. Zaczynamy od wybrania przedziału $[a, b]$. Nastepnie w każdej iteracji obliczamy punkty $c^{(i)}$ oraz $d^{(i)}$, tak aby spełniały:

$$b^{(i)} - d^{(i)} = c^{(i)} - a^{(i)}$$

oznacza to, że są równo oddalone od aktualnego przedziału przeszukiwań.

Jak działa zawężanie przedziału poszukiwań

- Gdy: $f(c) < f(d)$, wtedy $a^{(i+1)} = a^{(i)}$, $b^{(i+1)} = d^{(i)}$
- Gdy: $f(d) < f(c)$, wtedy $a^{(i+1)} = c^{(i)}$, $b^{(i+1)} = b^{(i)}$



Metoda Fibonacciego

Wyznaczanie punktów $c^{(i)}$ oraz $d^{(i)}$.

$$c^{(i)} = b^{(i)} - \alpha^{(i)} \cdot \left(b^{(i)} - a^{(i)}\right)$$

$$d^{(i)} = a^{(i)} + b^{(i)} - c^{(i)}$$

$$\alpha^{(i)} = \frac{\phi_{k-i-1}}{\phi_{k-i}}$$

$$\phi_k = min\{F(k) : F(k) > \frac{L}{\epsilon}\}$$

Gdzie: $F(k)$ to k-ty wyraz *Ciągu Fibonacciego*, $L = |$ a - b $|$, ϵ - zadana dokładność

Obliczenie ilości iteracji

Z kryterium obliczającego *phi*$_k$ wiemy, że należy wykonać k-2 iteracji. Gdyż pierwsze wartości w ciągu fibonacciego są 1 i 0, więc wartość parametru *alpha* wyniesie 1 i 0 (dla iteracji $k-1$ i k), a więc nie dokona się już Ponadto ze względu na błędy zaokrągleń, zwiększam liczbę iteracji o 1.

```
phi_k <- function(a, b, tol) {
  i <- 1
  L <- b - a
  while(phi(i) < L / tol) {i <- i+1}
  return(i + 1)
}
```

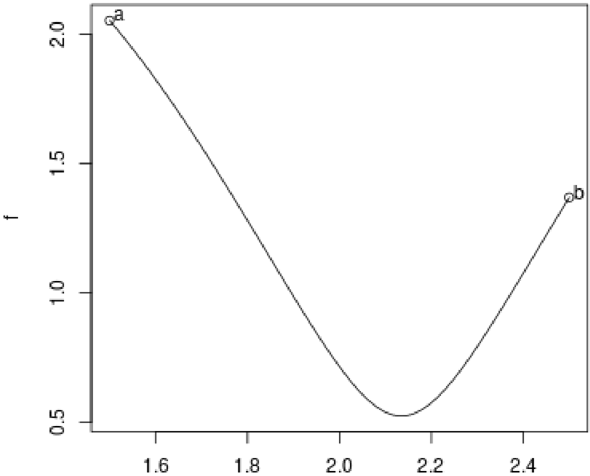
Metoda Fibonacciego implementacja

```
fib <- function(f, a, b, tol) {
  k <- phi_k(a, b, tol)
  for(i in c(0:(k-3))) {
      alpha <- phi(k-i-1)/phi(k-i)
      c <- b - alpha*(b - a)
      d <- a + b - c
      cat("iteracja=", i+1, "a=", a, "c=",c,"d=",d,"b=",
          b , "alpha=", alpha,"\n",sep=" ")
      if( f(c) < f(d) ) {
        b <- d
      } else {
        a <- c
      }
  }
  return((a+b)/2)
}
```

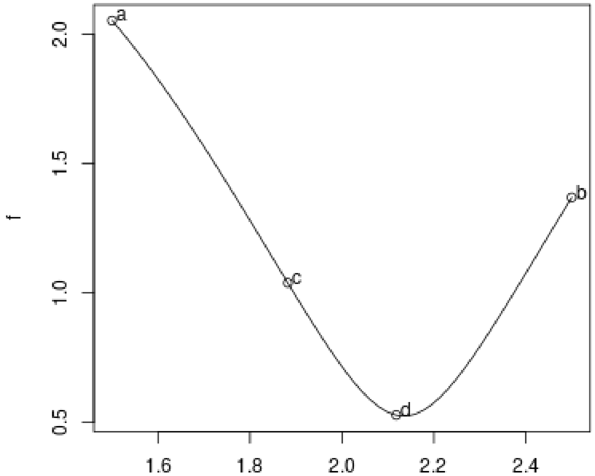
Wizualizacja

```
fib_anim <- function(f, a, b, tol, max_iter) {
  k <- phi_k(a, b, tol)
  lower <- a
  upper <- b
  for(i in c(0: min((k-3), max_iter))) {
      alpha <- phi(k-i-1)/phi(k-i)
      # FITST PLOT
      plot(f, xlim=c(lower, upper))
      points(xp <- c(a,b), yp <- f(xp))
      text(xp+0.02, yp+0.02, c("a", "b"))
      dev.print(png, paste("img/", 3*i + 1, ".png", sep=""), width = 400, height = 400)
      c <- b - alpha*(b - a)
      d <- a + b - c
      # SECOND PLOT
      plot(f, xlim=c(lower, upper))
      points(xp <- c(a,b,c, d), yp <- f(xp))
      text(xp+0.02, yp+0.02, c("a", "b", "c", "d"))
      dev.print(png, paste("img/", 3*i + 2, ".png", sep=""), width = 400, height = 400)
      # THIRD PLOT
      plot(f, xlim=c(lower, upper))
      points(xp <- c(a,b,c, d), yp <- f(xp))
      text(xp+0.02, yp+0.02, c("a", "b", "c", "d"))
      if( f(c) < f(d) ) {
        arrows(b, f(b), d, f(d), col="red")
        b <- d
      } else {
        arrows(a, f(a), c, f(c), col="red")
        a <- c
      }
      dev.print(png, paste("img/", 3*i + 3, ".png", sep=""), width = 400, height = 400) }
  return((a+b)/2)
```

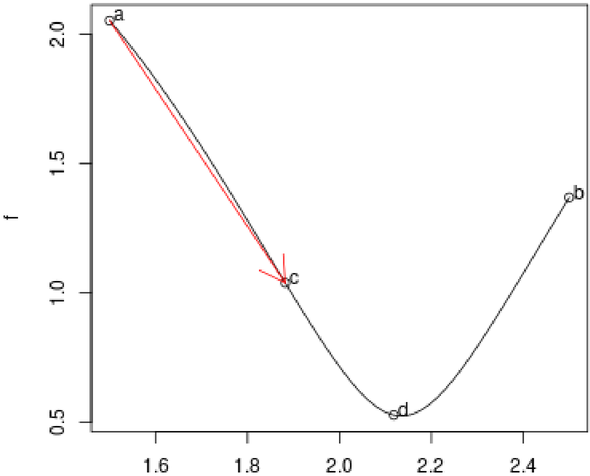
Wizualizacja - animacja



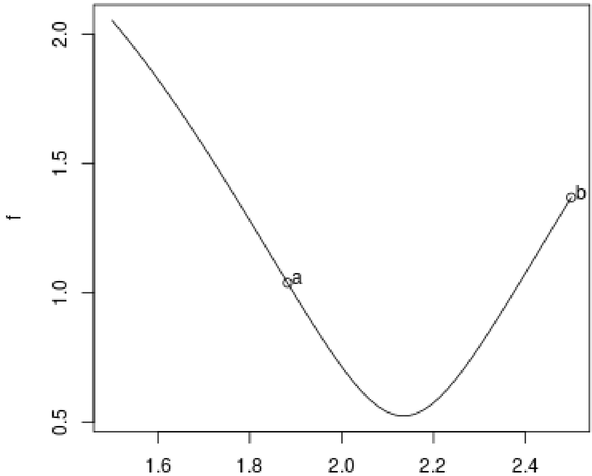
Wizualizacja - animacja



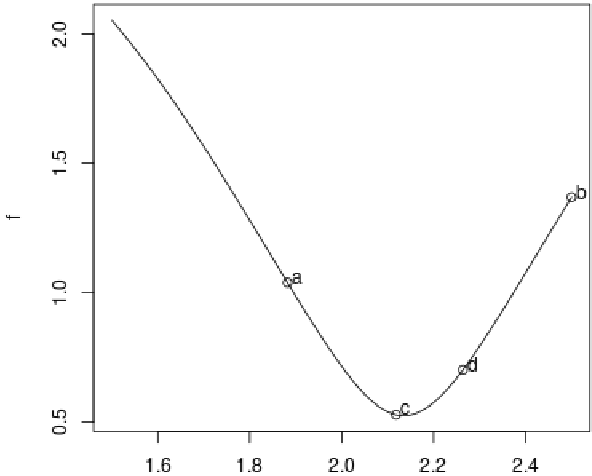
Wizualizacja - animacja



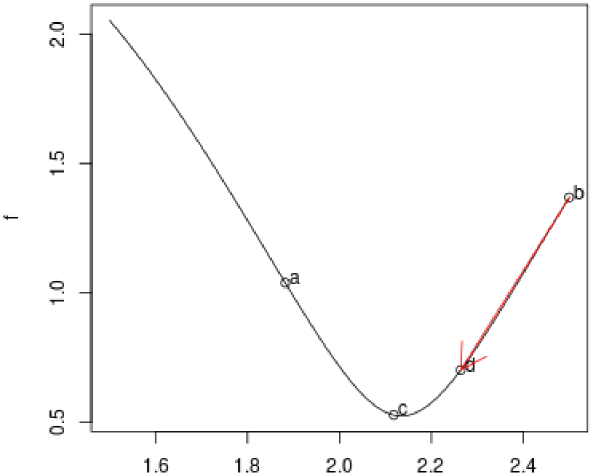
Wizualizacja - animacja



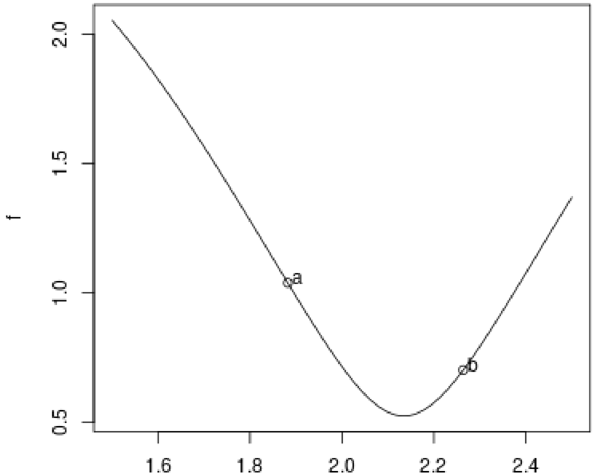
Wizualizacja - animacja



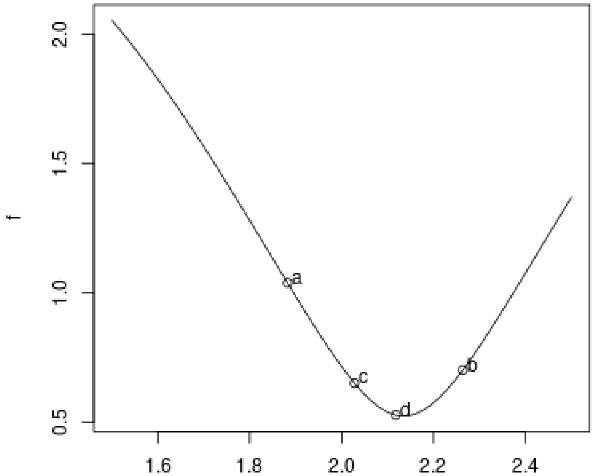
Wizualizacja - animacja



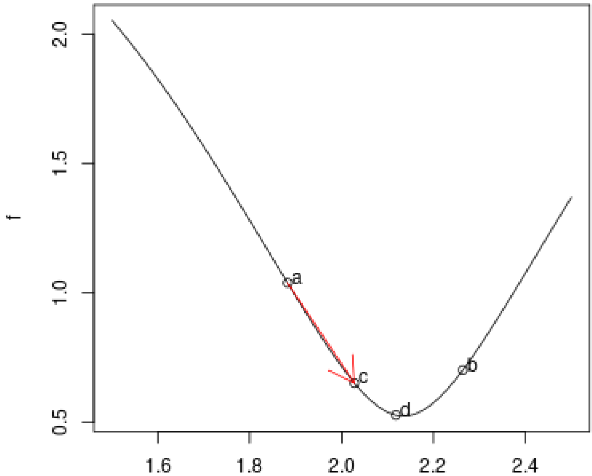
Wizualizacja - animacja



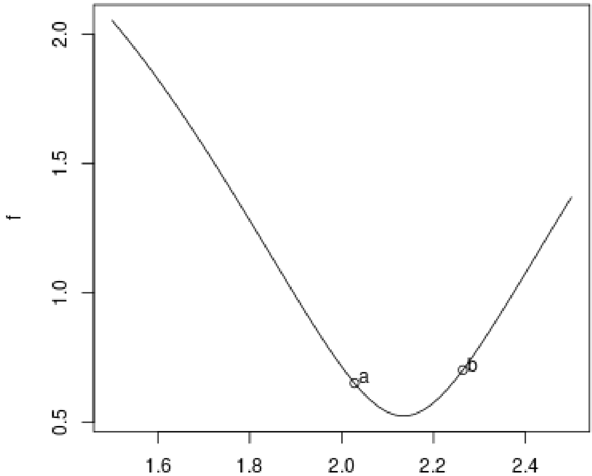
Wizualizacja - animacja



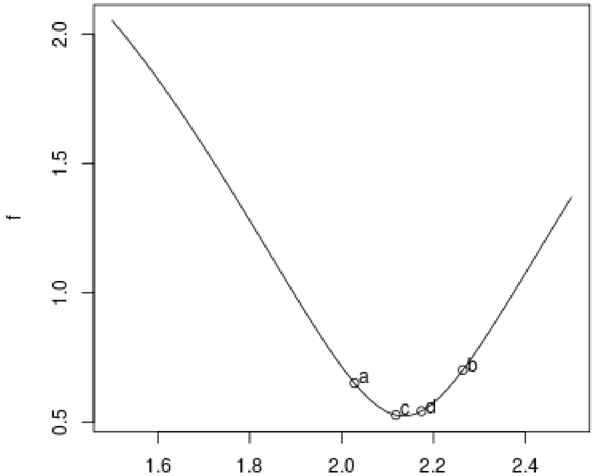
Wizualizacja - animacja



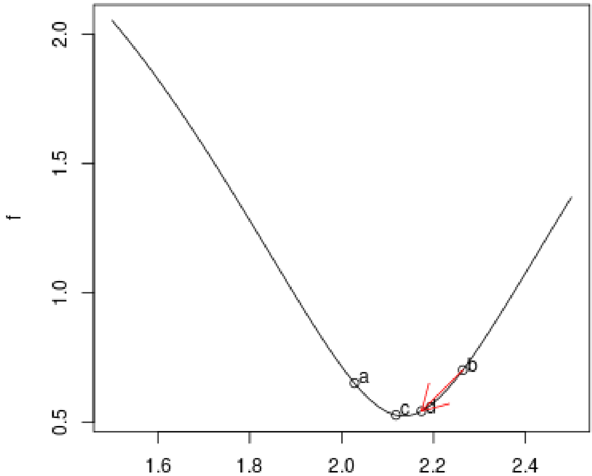
Wizualizacja - animacja



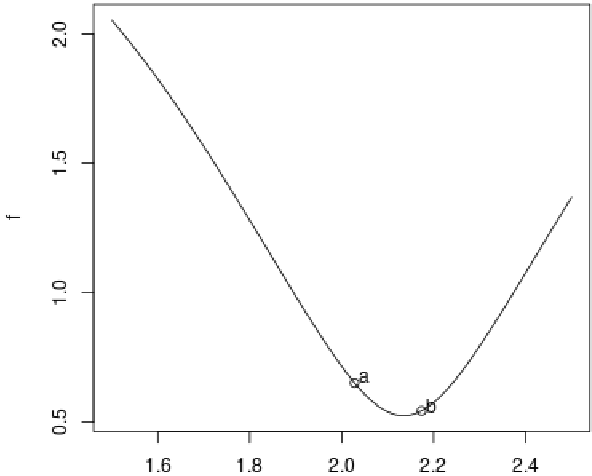
Wizualizacja - animacja



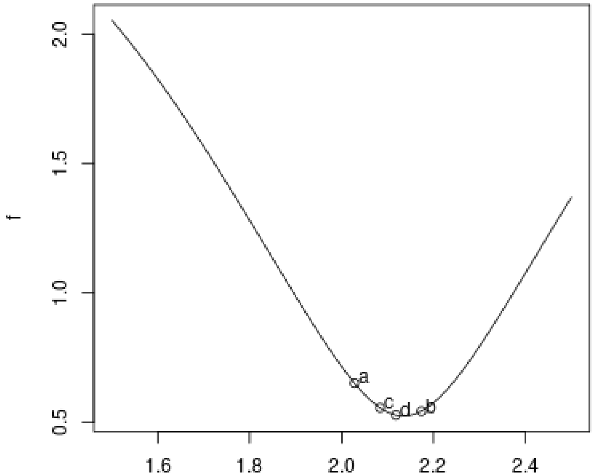
Wizualizacja - animacja



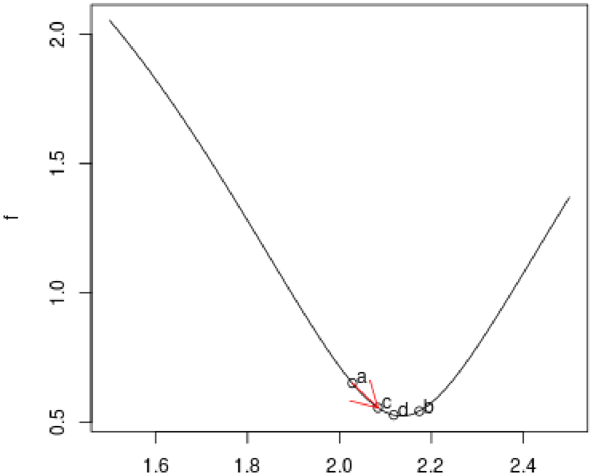
Wizualizacja - animacja



Wizualizacja - animacja



Wizualizacja - animacja



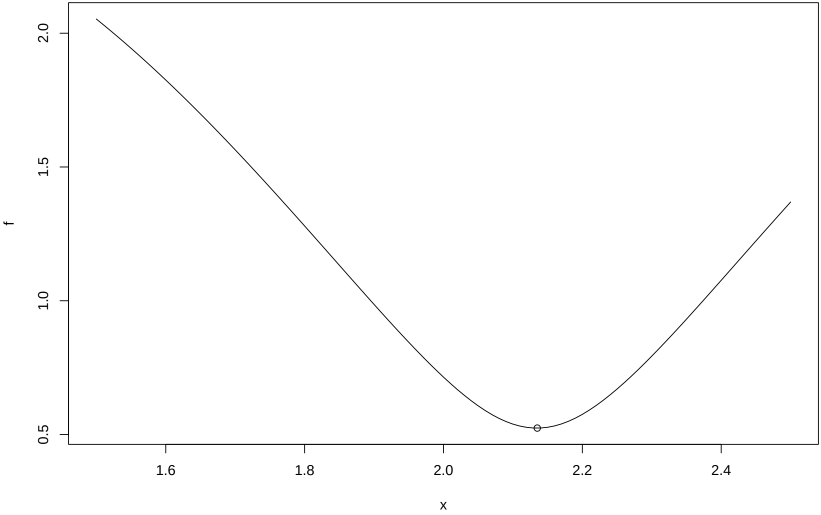
Rozwiązanie

```
fib(f, 1.5, 2.5, 1e-3)
```

```
## iteracja= 1 a= 1.5 c= 1.881966 d= 2.118034 b= 2.5 alpha= 0.6180341
## iteracja= 2 a= 1.881966 c= 2.118034 d= 2.263932 b= 2.5 alpha= 0.6180338
## iteracja= 3 a= 1.881966 c= 2.027864 d= 2.118034 b= 2.263932 alpha= 0.6180344
## iteracja= 4 a= 2.027864 c= 2.118034 d= 2.173762 b= 2.263932 alpha= 0.6180328
## iteracja= 5 a= 2.027864 c= 2.083591 d= 2.118034 b= 2.173762 alpha= 0.6180371
## iteracja= 6 a= 2.083591 c= 2.118034 d= 2.139319 b= 2.173762 alpha= 0.6180258
## iteracja= 7 a= 2.118034 c= 2.139319 d= 2.152477 b= 2.173762 alpha= 0.6180556
## iteracja= 8 a= 2.118034 c= 2.131192 d= 2.139319 b= 2.152477 alpha= 0.6179775
## iteracja= 9 a= 2.118034 c= 2.126161 d= 2.131192 b= 2.139319 alpha= 0.6181818
## iteracja= 10 a= 2.126161 c= 2.131192 d= 2.134288 b= 2.139319 alpha= 0.6176471
## iteracja= 11 a= 2.131192 c= 2.134288 d= 2.136223 b= 2.139319 alpha= 0.6190476
## iteracja= 12 a= 2.131192 c= 2.133127 d= 2.134288 b= 2.136223 alpha= 0.6153846
## iteracja= 13 a= 2.133127 c= 2.134288 d= 2.135062 b= 2.136223 alpha= 0.625
## iteracja= 14 a= 2.133127 c= 2.133901 d= 2.134288 b= 2.135062 alpha= 0.6
## iteracja= 15 a= 2.133901 c= 2.134288 d= 2.134675 b= 2.135062 alpha= 0.6666667
## iteracja= 16 a= 2.134288 c= 2.134675 d= 2.134675 b= 2.135062 alpha= 0.5
## iteracja= 17 a= 2.134675 c= 2.134675 d= 2.135062 b= 2.135062 alpha= 1
## iteracja= 18 a= 2.134675 c= 2.135062 d= 2.134675 b= 2.135062 alpha= 0
```

```
## [1] 2.135062
```

Rozwiązanie - wykres



Metoda bisekcji

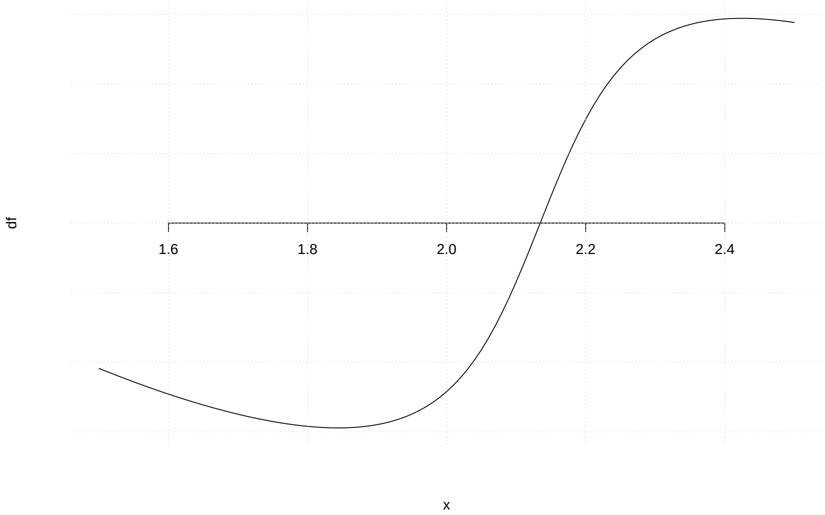
Metoda ta do znalezienia minimum/maximum wykorzystuje **alborytm bisekcji**. Algorytm bisekcji znajduje miejsce zerowe dowolnej ciągłej funkcji. Wykorzystuje on fakt, że funkcja zmienia znak po przejściu przez miejsce zerowe. Mając dane punkty początkowe $[a, b]$ oraz $f(a)$ i $f(b)$. W każdej iteracji **algorytmu bisekcji** wybieramy punkt $c = \frac{a+b}{2}$, oraz obliczamy $f_c = f(c)$. Następnie wybieramy ten z przedziałów $[a, m]$, $[m, b]$, dla których iloczyn $f(a) * f(m)$ lub $f(b) * f(m)$ jest *ujemny* (co oznacza, że minimum jest w wybranym przedziale). Możemy wykorzystać ten fakt, aby znajdować minima/maxima funkcji tj. będziemy poszukiwać miejsc zerowych pochodnej.

Poszukiwanie pochodnej funkcji

Ponieważ funkcja ta działa na liczbach zespolonych i oblicza *abs*, R nie pozwala na analityczne obliczenie jej pochodnej. Należy więc to zrobić numerycznie:

```
library(numDeriv)
df <- function(x) {grad(f, x)}
```

Wykres pochodnej funkcji



Metoda bisekcji

```
bisect <- function(df, a, b, tol) {
  iter <- ceiling(log2((b-a)/tol))
  for(i in c(1:iter)) {
    m <- (a+b)/2
    df.m <- df(m)
    cat("iteracja=", i, "a=", a, "b=",b, "m=",m,"\n",sep="
    if(df.m * df(a) < 0) {
      b <- m
    } else {
      a <- m
    }
  }
  (a+b)/2
}
```

Rozwiązanie - metoda bisecji

```
xr <- bisect(df, 0.5, 1.5, 1e-3)
```

```
## iteracja= 1 a= 0.5 b= 1.5 m= 1
## iteracja= 2 a= 1 b= 1.5 m= 1.25
## iteracja= 3 a= 1 b= 1.25 m= 1.125
## iteracja= 4 a= 1 b= 1.125 m= 1.0625
## iteracja= 5 a= 1.0625 b= 1.125 m= 1.09375
## iteracja= 6 a= 1.0625 b= 1.09375 m= 1.078125
## iteracja= 7 a= 1.0625 b= 1.078125 m= 1.070312
## iteracja= 8 a= 1.0625 b= 1.070312 m= 1.066406
## iteracja= 9 a= 1.066406 b= 1.070312 m= 1.068359
## iteracja= 10 a= 1.066406 b= 1.068359 m= 1.067383
```

```
c(xr, f(xr))
```

```
## [1] 1.066895 2.524000
```

Rozwiązanie - metoda bisekcji - wykres