

# Rozwiązywanie równań nieliniowych metodami połowienia, *regula falsi* i siecznych w arytmetyce przedziałowej

Mikołaj Rozwadowski

29 maja 2017

## 1 Informacje ogólne

### 1.1 Zastosowanie

Każda z niżej opisanych metod rozwiązuje w arytmetyce przedziałowej równanie  $f(x) = 0$ , gdzie  $f$  jest dowolną funkcją rzeczywistą. Do znalezienia rozwiązania potrzebny jest początkowy przedział  $[a, b]$ , który jest potem sukcesywnie zawężany.

### 1.2 Sposób wczytania funkcji

Wymagana przez każdą metodę funkcja  $f$  jest przekazywana jako wskaźnik do obiektu klasy `Function`. Można samemu napisać podklasę realizującą jakąkolwiek funkcję albo posłużyć się klasą `S0Function`, która umożliwia załadowanie jej z biblioteki dynamicznej `.so` pod systemami operacyjnymi z rodziny GNU/Linux. Sposób utworzenia i skompilowania kompatybilnej biblioteki opisany jest na stronie projektu w serwisie GitHub.

### 1.3 Identyfikatory nielokalne

W pliku nagłówkowym `common.h` znajdują się deklaracje pomocniczych funkcji i stałych, z których korzystają wszystkie metody:

### Function

klasa udostępniająca metodę `interval evaluate(interval x)`.

### WRONG\_INTERVAL

stała o wartości liczbowej 1; jest to wyjątek, którym funkcja sygnalizuje, że w podanym przedziale początkowym lewy koniec jest większy lub równy od prawego końca,

### NO\_REAL\_ROOTS

stała o wartości liczbowej 2; jest to wyjątek, którym funkcja sygnalizuje brak pierwiastków rzeczywistych w podanym przedziale początkowym,

### check\_interval

pomocnicza funkcja, która sprawdza warunki dla przedziału początkowego, a niespełnienie ich sygnalizuje zgłoszeniem jednego z powyższych wyjątków,

### sgn

pomocnicza funkcja wyznaczająca znak przedziału, przy czym jeśli porównanie nie jest jednoznaczne, to przyjmowany jest znak 0.

## 2 Metoda połowienia przedziału

### 2.1 Zastosowanie

Funkcja `Bisection` znajduje wartość pierwiastka równania  $f(x) = 0$  metodą połowienia przedziału w arytmetyce przedziałowej.

### 2.2 Opis metody

Metoda połowienia wykorzystuje własność Darboux, która mówi, że jeśli dana jest funkcja ciągła  $f$  i przedział rzeczywisty  $[a, b]$  takie, że  $f(a) = y_a$  i  $f(b) = y_b$ , to funkcja ta przyjmuje w przedziale  $(a, b)$  wszystkie wartości pośrednie między  $y_a$  a  $y_b$ . W szczególności oznacza to, że jeśli taka funkcja ma na końcach przedziału różne znaki, to istnieje tam miejsce zerowe, czyli punkt  $x_0 \in (a, b)$ , dla którego  $f(x_0) = 0$ .

Algorytm rozpoczyna pracę z danym przedziałem  $[a, b]$ , o którym wiadomo, że  $f(a) \cdot f(b) < 0$ . W każdej iteracji następuje wyznaczenie środka przedziału  $m = \frac{a+b}{2}$  i podział go na dwie połowy  $[a, m]$  i  $[m, b]$ . Jeżeli

$f(a) \cdot f(m) < 0$ , to przeszukiwanie jest kontynuowane w przedziale  $[a, m]$ , w przeciwnym razie  $[m, b]$ .

Pętla powtarza się aż do momentu, w którym środek przedziału  $m$  będzie miejscem zerowym funkcji. Ponieważ w praktyce może to jednak nie nastąpić, do przerywania algorytmu dojdzie też, gdy szerokość przedziału będzie mniejsza niż zadana tolerancja  $\epsilon$  lub po wykonaniu określonej liczby iteracji. W takim wypadku wynikiem będzie najwęższy uzyskany przedział zawierający pierwiastek.

## 2.3 Wywołanie funkcji

`Bisection(a, b, func, tolerance, iterations, reached)`

## 2.4 Dane

`a, b`

lewy i prawy koniec przedziału zawierającego pierwiastek,

`func`

funkcja, której miejsce zerowe należy znaleźć,

`tolerance`

największa akceptowalna szerokość przedziału wynikowego,

`iterations`

maksymalna liczba iteracji.

## 2.5 Wyniki

`Bisection(a, b, func, tolerance, iterations, reached)`

przedział zawierający miejsce zerowe

## 2.6 Inne parametry

`reached`

określa, czy w `iterations` iteracjach udało się zmieścić w wymaganej tolerancji.

Jeśli początkowy przedział  $[a, b]$  nie będzie spełniać wymagań, to funkcja wyrzuci wyjątek (zob. 1.3). Również za pomocą wyjątku odrzucone zostanie podanie liczby iteracji mniejszej lub równej 0.

## 2.7 Typy parametrów

interval a, interval b, Function\* func, long double tolerance, int iterations, bool& reached

## 2.8 Identyfikatory nielokalne

Oprócz wymienionych w punkcie 1.3 plik źródłowy definiuje następujące identyfikatory nielokalne:

NOT\_ENOUGH\_ITERATIONS

stała o wartości liczbowej 3; jest to wyjątek, którym funkcja sygnalizuje, że parametr iterations nie jest liczbą dodatnią.

## 2.9 Kod źródłowy

```
3 interval Bisection(interval a, interval b, Function *func,
  ↳ long double tolerance, int iterations, bool &reached) {
4     check_interval(a, b, func);
5
6     if (iterations < 1) {
7         throw NOT_ENOUGH_ITERATIONS;
8     }
9
10    interval x, y, right, left, mid;
11    x = hull(a, b);
12    long double midpoint;
13    reached = false;
14
15    for (int i = 0; i < iterations; i++) {
16        midpoint = median(x);
17        left = interval(lower(x), midpoint);
18        mid = interval(midpoint);
19        right = interval(midpoint, upper(x));
```

```

20
21     y = func->evaluate(mid);
22
23     if (singleton(y) && cereq(y, 0.01)) {
24         reached = true;
25         return mid;
26     }
27
28     if (width(x) < tolerance) {
29         reached = true;
30         break;
31     }
32
33     if (zero_in(func->evaluate(left))) {
34         x = left;
35     } else {
36         x = right;
37     }
38 }
39
40 return x;
41 }

```

## 2.10 Przykłady

Równanie	a	b	$x_0$
$x^2 - 2 = 0$	1	2	[1.41421356237309503, 1.41421356237309509]
$xe^{\sqrt{x+1}} - 1 = 0$	-1	1	[3.17347582146508266, 3.17347582146508323]
$\sin x \cdot (\sin x + \frac{1}{2}) - \frac{1}{2} = 0$	[0.4, 0.5]	1	[5.2359877559829801e-01, 5.2359877559829809e-01]

We wszystkich przykładach przyjęto liczbę iteracji = 60 i  $\epsilon = 10^{-16}$ .

## 3 Metoda *regula falsi*

### 3.1 Zastosowanie

Funkcja `RegulaFalsi` znajduje wartość pierwiastka równania  $f(x) = 0$  metodą *regula falsi* w arytmetyce przedziałowej.

## 3.2 Opis metody

Metoda *regula falsi* opiera się na założeniu, że każdą funkcję można w odpowiednio małym zakresie argumentów traktować jak funkcję liniową. Choć z matematycznego punktu widzenia jest to nieprawda (stąd nazwa – *regula falsi* to po łacinie fałszywa zasada albo fałszywa prosta), to obliczanie kolejnych miejsc zerowych tak, jakby funkcja rzeczywiście była na tym odcinku liniowa, daje coraz lepsze przybliżenia prawdziwego pierwiastka.

W każdej iteracji algorytmu wyznaczany jest punkt przecięcia prostej przechodzącej przez punkty  $(a, f(a))$  i  $(b, f(b))$  z osią  $X$ :

$$x = b - f(b) \cdot \frac{b - a}{f(b) - f(a)},$$

a następnie uzyskany punkt zastępuje lewy lub prawy koniec przedziału początkowego. Pętla trwa dopóki spełniony jest warunek  $a > x > b$ .

## 3.3 Wywołanie funkcji

`RegulaFalsi(a, b, func)`

## 3.4 Dane

`a, b`

lewy i prawy koniec przedziału zawierającego pierwiastek,

`func`

funkcja, której miejsce zerowe należy znaleźć.

## 3.5 Wyniki

`RegulaFalsi(a, b, func)`

przedział zawierający miejsce zerowe

## 3.6 Inne parametry

Brak. Jeśli początkowy przedział  $[a, b]$  nie będzie spełniać wymagań, to funkcja wyrzuci wyjątek (zob. 1.3).

### 3.7 Typy parametrów

interval a, interval b, Function\* func

### 3.8 Identyfikatory nielocalne

Brak, nie licząc wymienionych w punkcie 1.3.

### 3.9 Kod źródłowy

```
3 interval RegulaFalsi(interval a, interval b, Function *func) {
4     check_interval(a, b, func);
5
6     interval fa, fb, fx, x;
7     int sign_fa, sign_fx;
8
9     fa = func->evaluate(a);
10    fb = func->evaluate(b);
11    sign_fa = sgn(fa);
12
13    x = b - fb * (b - a) / (fb - fa);
14    while (upper(a) < lower(x) && upper(x) < lower(b)) {
15        fx = func->evaluate(x);
16        sign_fx = sgn(fx);
17
18        if (sign_fa == sign_fx) {
19            a = x;
20            fa = fx;
21        } else if (sign_fa == -sign_fx) {
22            b = x;
23            fb = fx;
24        } else {
25            break;
26        }
27
28        if (zero_in(fb - fa)) {
29            break;
30        }
    }
```

```

31
32         x = b - fb * (b - a) / (fb - fa);
33     }
34
35     return hull(a, b);
36 }

```

### 3.10 Przykłady

Równanie	a	b	$x_0$
$x^2 - 2 = 0$	1	2	[1.4142135623730950, 1.4142135623734952]
$x^2 - 2 = 0$	0.3	[1.5, 1.6]	[2.999999999999999e-01, 1.6000000000000001]
$\sin x \cdot (\sin x + \frac{1}{2}) - \frac{1}{2} = 0$	0.1	1	[5.2359877559652552e-01, 1]

## 4 Metoda siecznych

### 4.1 Zastosowanie

Funkcja `Secant` znajduje wartość pierwiastka równania  $f(x) = 0$  metodą siecznych w arytmetyce przedziałowej.

### 4.2 Opis metody

Podobnie jak *regula falsi*, metoda siecznych również opiera się na interpolacji liniowej. Algorytm konstruuje zbieżny do dokładnej wartości pierwiastka ciąg przybliżeń  $(x_i)$  według rekurencyjnego wzoru:

$$x_i = x_{i-1} - f(x_{i-1}) \frac{x_{i-1} - x_{i-2}}{f(x_{i-1}) - f(x_{i-2})}.$$

Pierwsze przybliżenia wyznaczane są na podstawie końców przedziału początkowego jako:  $x_1 = a + h$ ,  $x_2 = b - h$ ,  $h = 0,179372 \cdot (b - a)$ , przy czym jeśli nie jest spełniony warunek  $|f(x_1)| \geq |f(x_2)|$ , to wartości te są na początku zamieniane miejscami.

### 4.3 Wywołanie funkcji

`Secant(a, b, func)`



## 4.4 Dane

a, b

lewy i prawy koniec przedziału zawierającego pierwiastek,

func

funkcja, której miejsce zerowe należy znaleźć.

## 4.5 Wyniki

Secant(a, b, func)

przedział zawierający miejsce zerowe

## 4.6 Inne parametry

Brak. Jeśli początkowy przedział  $[a, b]$  nie będzie spełniać wymagań, to funkcja wyrzuci wyjątek (zob. 1.3).

## 4.7 Typy parametrów

interval a, interval b, Function\* func

## 4.8 Identyfikatory nielokalne

Brak, nie licząc wymienionych w punkcie 1.3.

## 4.9 Kod źródłowy

```
6 interval Secant(interval a, interval b, Function *func) {
7     check_interval(a, b, func);
8
9     interval fa, fb, h, x, fx;
10    h = (b - a) * 0.1793721;
11    a += h;
12    b -= h;
13    fa = func->evaluate(a);
14    fb = func->evaluate(b);
15
16    if (cerlt(abs(fa), abs(fb))) {
```

```

17         std::swap(a, b);
18         std::swap(fa, fb);
19     }
20
21     while (true) {
22         if (zero_in(fa - fb)) {
23             break;
24         }
25
26         x = b + fb * (b - a) / (fa - fb);
27         fx = func->evaluate(x);
28         if (overlap(a, x) || overlap(b, x) || (singleton(fx)
29             ↪ && zero_in(fx))) {
30             break;
31         }
32
33         fa = fb;
34         fb = fx;
35         a = b;
36         b = x;
37     }
38
39     return hull(x, hull(a, b));

```

## 4.10 Przykłady

Równanie	a	b	$x_0$
$x^2 - 2 = 0$	1	2	[1.4142135623730950, 1.4142135623734952]
$xe^{\sqrt{x+1}} - 1 = 0$	-1	1	[3.1734757786211827e-01, 3.1734758214652323e-01]
$xe^{\sqrt{x+1}} - 1 = 0$	[-0.5, -0.4]	[0.2, 0.4]	[-3.9237680000000000e-01, 1.2677667075395474]

## 5 Bibliografia

- A. Marciniak, D. Gregulec, J. Kaczmarek: *Podstawowe procedury numeryczne w języku Turbo Pascal*. NAKOM, Poznań, 2000 r.