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 `SOFunction`, 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 przerwanie algorytmu dojdzie też, gdy szerokość przedziału będzie mniejsza niż zadana tolerancja ϵ 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 miesce 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

```
interval Bisection(interval a, interval b, Function *func,
 ↪  long double tolerance, int iterations, bool &reached) {
    check_interval(a, b, func);

    if (iterations < 1) {
        throw NOT_ENOUGH_ITERATIONS;
    }

    interval x, y, right, left, mid;
    x = hull(a, b);
    long double midpoint;
    reached = false;

    for (int i = 0; i < iterations; i++) {
        midpoint = median(x);
        left = interval(lower(x), midpoint);
        mid = interval(midpoint);
        right = interval(midpoint, upper(x));
```

```

        y = func->evaluate(mid);

        if (singleton(y) && cereq(y, 0.0l)) {
            reached = true;
            return mid;
        }

        if (width(x) < tolerance) {
            reached = true;
            break;
        }

        if (zero_in(func->evaluate(left))) {
            x = left;
        } else {
            x = right;
        }
    }

    return x;
}
```

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 miesce 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 nielokalne

Brak, nie licząc wymienionych w punkcie 1.3.

3.9 Kod źródłowy

```
interval RegulaFalsi(interval a, interval b, Function *func) {
    check_interval(a, b, func);

    interval fa, fb, fx, x;
    int sign_fa, sign_fx;

    fa = func->evaluate(a);
    fb = func->evaluate(b);
    sign_fa = sgn(fa);

    x = b - fb * (b - a) / (fb - fa);
    while (upper(a) < lower(x) && upper(x) < lower(b)) {
        fx = func->evaluate(x);
        sign_fx = sgn(fx);

        if (sign_fa == sign_fx) {
            a = x;
            fa = fx;
        } else if (sign_fa == -sign_fx) {
            b = x;
            fb = fx;
        } else {
            break;
        }

        if (zero_in(fb - fa)) {
            break;
        }
```

```

        x = b - fb * (b - a) / (fb - fa);
    }

    return hull(a, b);
}
```

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]	[2.9999999999999999e-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 miesce 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

```
interval Secant(interval a, interval b, Function *func) {
    check_interval(a, b, func);

    interval fa, fb, h, x, fx;
    h = (b - a) * 0.179372l;
    a += h;
    b -= h;
    fa = func->evaluate(a);
    fb = func->evaluate(b);

    if (cerlt(abs(fa), abs(fb))) {
```

```
        std::swap(a, b);
        std::swap(fa, fb);
    }

    while (true) {
        if (zero_in(fa - fb)) {
            break;
        }

        x = b + fb * (b - a) / (fa - fb);
        fx = func->evaluate(x);
        if (overlap(a, x) || overlap(b, x) || (singleton(fx)
         ↪  && zero_in(fx))) {
            break;
        }

        fa = fb;
        fb = fx;
        a = b;
        b = x;
    }

    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.