MNUM – Projekt 3.9

KrissKry xd

Spis treści

Zadanie 1

Polecenie:

Proszę znaleźć wszystkie zera funkcji

$$f(x) = 7{,}2 - 2x - 5e^{-x}$$

w przedziale [–5, 10], używając dla każdego zera programu z implementacją

a) metody bisekcji,

b) metody Newtona

Wprowadzenie teoretyczne

Zero

lub zamiennie pierwiastek funkcji f(x), odnosi się do takiego x, dla którego jest spełnione

$$f(x) = 0$$

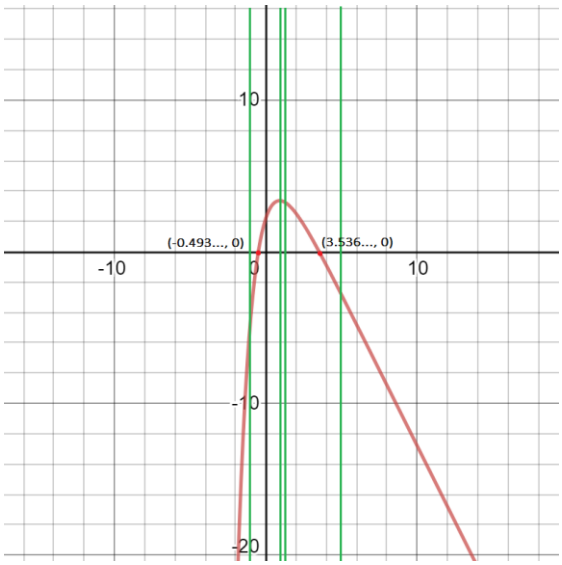
Dla funkcji nieliniowych wyznaczanie zer nie jest tak oczywiste jak dla funkcji liniowych. By obliczyć punkty przecięcia funkcji z osią x należy wyznaczyć pewne przedziały, w których mamy pewność istnienia jednego zera. Następnie można skorzystać z metod iteracyjnych przybliżających ten punkt.

By wyznaczyć takie przedziały można skorzystać z wykresu danej funkcji lub skorzystać z algorytmu wyznaczającego je za nas. Przekazując mu pewien przedział początkowy, algorytm powinien badać zmiany wartości funkcji i przybliżać nas do rozwiązania.

Opis problemu

Problemem jest znalezienia zer zadanej funkcji. Po szybkim jej naszkicowaniu wiemy, że będzie ona przecinać oś X w dokładnie dwóch miejscach. Celem algorytmu będzie wyznaczenie przybliżonych wartości punktów przecięcia.

Wykres funkcji



Metoda bisekcji (przepołowienia)

Schemat rozwiązania

Na podstawie wartości skrajnych przedziału wyznaczam punkt c znajdujący się w równej odległości od nich:

$$c = \frac{a+b}{2}$$

Następnym krokiem jest obliczenie wartości funkcji $f(a), f(b), f(c)$ dla tych punktów

Wyznaczamy i sprawdzamy znak iloczynów $f(a) \cdot f(c)$ oraz $f(b) \cdot f(c)$

W przypadku gdy któryś z iloczynów ma znak ujemny, należy kontynuować badania ograniczając przedział badawczy do punktów dla których iloczyn był sprawdzony, tzn.

$$f(a) \cdot f(c) < 0 \Rightarrow a = a, b = c$$

$$f(b) \cdot f(c) < 0 \Rightarrow a = c, b = b$$

Obliczenia te powtarzamy aż do spełnienia któregoś z warunków przerwania obliczeń, a mogą być to:

- Osiągnięta została maksymalna liczba iteracji algorytmu
- $f(c)$ spełnia założoną dokładność wartości funkcji bliskiej zeru
- $|b-a|$ spełnia założoną dokładność wyznaczenia przedziału dla miejsca zerowego

Dokładność metody

W swoim algorytmie założyłem dokładność przekazywaną funkcji bisekcji o wartości 0.0001, dotyczy ona zarówno wartości funkcji w punkcie c oraz rozmiaru przedziału – może zostać spełniony dowolny warunek z tych dwóch w celu zakończenia obliczeń.

Zbieżność metody

$$\varepsilon_k - odległość\ między\ krańcami\ przedziału\ w\ k-tej\ iteracji$$

$$\varepsilon_{k+1} = \frac{1}{2} \cdot \varepsilon_k$$

Dla n iteracji osiągniemy odległość wynoszącą $\varepsilon_n = \frac{\varepsilon_0}{2^n}$

Liczba iteracji potrzebna do dokładności ε_n wynosi

$$n = \log_n \frac{\varepsilon_0}{\varepsilon_n}$$

Można więc zauważyć, że współczynnik zbieżności wynosi 0.5, co czyni metodę bisekcji niekoniecznie szybko zbieżną przy wybraniu dostatecznie dużego przedziału początkowego.

Kod

W celu przeprowadzenia eksperymentu w ramach laboratorium został zaimplementowany algorytm wyznaczenia przedziałów izolacji pierwiastka zgodnie z treścią podręcznika do Metod Numerycznych prof. Piotra Tatjewskiego. Implementacja metody bisekcji również na nim bazuje.

```
function root = bisection(a, b, tolerance)

    fn_c = 10000;
    while (abs(fn_c) > tolerance) && (abs(b-a) > tolerance)

        c = (a + b)/2;
        fn_a = evaluate(a);
        fn_b = evaluate(b);
        fn_c = evaluate(c);

        %looking for a root on the left
        if fn_a * fn_c < 0
            b = c;

        %looking for a root on the right
        elseif fn_c * fn_b < 0
            a = c;

        end
    end
    root = c;
end
```

Wyniki

Pierwsze miejsce zerowe				
iteracja	a	b	c	fn_c
1	-1.050000e+00	1	-2.500000e-02	2.123424e+00
2	-1.050000e+00	-2.500000e-02	-5.375000e-01	-2.836110e-01
3	-5.375000e-01	-2.500000e-02	-2.812500e-01	1.138576e+00
4	-5.375000e-01	-2.812500e-01	-4.093750e-01	4.893684e-01
5	-5.375000e-01	-4.093750e-01	-4.734375e-01	1.193568e-01
6	-5.375000e-01	-4.734375e-01	-5.054688e-01	-7.787457e-02
7	-5.054688e-01	-4.734375e-01	-4.894531e-01	2.178729e-02
8	-5.054688e-01	-4.894531e-01	-4.974609e-01	-2.778000e-02
9	-4.974609e-01	-4.894531e-01	-4.934570e-01	-2.930704e-03
10	-4.934570e-01	-4.894531e-01	-4.914551e-01	9.444674e-03
11	-4.934570e-01	-4.914551e-01	-4.924561e-01	3.261084e-03
12	-4.934570e-01	-4.924561e-01	-4.929565e-01	1.662150e-04
13	-4.934570e-01	-4.929565e-01	-4.932068e-01	-1.381988e-03
14	-4.932068e-01	-4.929565e-01	-4.930817e-01	-6.078226e-04
15	-4.930817e-01	-4.929565e-01	-4.930191e-01	-2.207878e-04

Drugie miejsce zerowe				
iteracja	a	b	c	fn_c
1	1	5.100000e+00	3.050000e+00	8.632054e-01
2	3.050000e+00	5.100000e+00	4.075000e+00	-1.034961e+00
3	3.050000e+00	4.075000e+00	3.562500e+00	-6.683908e-02
4	3.050000e+00	3.562500e+00	3.306250e+00	4.042332e-01
5	3.306250e+00	3.562500e+00	3.434375e+00	1.700222e-01
6	3.434375e+00	3.562500e+00	3.498437e+00	5.190198e-02
7	3.498437e+00	3.562500e+00	3.530469e+00	-7.393412e-03
8	3.498437e+00	3.530469e+00	3.514453e+00	2.227337e-02
9	3.514453e+00	3.530469e+00	3.522461e+00	7.444713e-03
10	3.522461e+00	3.530469e+00	3.526465e+00	2.682922e-05

BISEKCJA		
a	b	n iteracji
-10	0	18
-20	0	19
-50	0	20
-100	0	21
-200	0	22
-500	0	24
-1000	0	24

BISEKCJA		
a	b	n iteracji
1	10	17
1	20	18
1	50	20
1	100	18
1	200	22
1	500	21
1	1000	24

Metoda Newtona (stycznych)

Schemat rozwiązania

Ten sposób wyznaczania miejsc zerowych zakłada aproksymacje funkcji jej liniowym przybliżeniej wynikającym z uciętego rozwinięcia w szereg Taylora w aktualnym punkcie x_n – przybliżeniu pierwiastka:

$$f(x) \approx f(x_n) + f'(x_n) \cdot (x - x_n)$$

Następny punkt aproksymacji x_{n+1} wynika z przyrównania do zera lokalnej liniowej aproksymacji funkcji $f(x)$:

$$f(x_n) + f'(x_n) \cdot (x_{n+1} - x_n) = 0$$

Z czego można wyznaczyć zależność iteracyjną:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Dokładność metody:

Dokładna aproksymacja w tej metodzie jest bezpośrednio powiązana z jej zbieżnością – jeśli oddalimy się zbytnio od miejsc zerowych, może się zdarzyć, że nie otrzymamy poprawnego wyniku. Nie zmienia to faktu, że przy odpowiednim punkcie startowym, dokładność zależy od ilości iteracji lub od założonej tolerancji. W moim przykładzie ustaliłem akceptowalną tolerancję na wartośc 0.0001

Zbieżność metody:

Jest to metoda zbieżna lokalnie co oznacza, że nie można jej stosować w punktach zbytnio oddalonych od rozwiązania, gdyż może być tam rozbieżna. Lokalnie, jest to bardzo szybka metoda o rzędzie zbieżności kwadratowym.

Kod

W celu wywołania algorytmu, korzystam z przygotowanego wcześniej algorytmu wyznaczania przedziałów, z którego wykorzystuję początek przedziału badawczego.

```
function [root] = newton(x, tolerance)

    xn = x;
    f_xn = 10000;
    % f_xn = evaluate(xn);
    % f_prim = derivative(xn);
    while abs(f_xn) > tolerance

        f_xn = evaluate(xn);

        f_prim = derivative(xn);

        xn = xn - (f_xn/f_prim);
    end
    root = xn;
end
```

Wyniki

Pierwsze miejsce zerowe			
iteracja	xn	f_xn	f_prim
1	-1.050000e+00	-4.988256e+00	1.228826e+01
2	-6.440632e-01	-1.032885e+00	7.521011e+00
3	-5.067299e-01	-8.581232e-02	6.299272e+00
4	-4.931073e-01	-7.665822e-04	6.186981e+00
5	-4.929834e-01	-6.283994e-08	6.185967e+00

Drugie miejsce zerowe			
iteracja	xn	f_xn	f_prim
1	1	3.360603e+00	-1.606028e-01
2	2.192493e+01	-3.664987e+01	-2.000000e+00
3	3.600000e+00	-1.366186e-01	-1.863381e+00
4	3.526682e+00	-3.763352e-04	-1.852989e+00
5	3.526479e+00	-3.032179e-09	-1.852959e+00

NEWTON	
x0	n iteracji
-5	10
-10	15
-20	25
-100	105
-200	205
-500	505
-1000	NaN

NEWTON	
x1	n iteracji
5	4
10	5
20	5
50	5
100	5
500	5
1000	5

Wnioski

Po krótkiej analizie wykresu funkcji, można stwierdzić, że jej miejsca zerowe różnią się nachyleniem, pod którym ta funkcja przez nie przechodzi. Sam ten fakt, mówi już, że różnica w szybkości działania wyznaczania miejsca zerowego może być znacząca.

Potwierdza to porównanie tabeli wyników **metody bisekcji** – w przypadku drugiego przecięcia osi x, a więc, gdy funkcja jest mniej stroma, liczba potrzebnych iteracji zmalała o 33%. Niewątpliwą jej zaletą jest brak wrażliwości na specyficzne zachowania funkcji (np. niespodziewane przegięcia). Mimo słabego współczynnika zbieżności jest to odpowiednia metoda do badania funkcji, których przebiegu nie znamy. Uwagę mogą przykuć porównania liczby iteracji dla punktów startowych znacznie oddalonych od miejsc zerowych – bez względu na nachylenie funkcji, liczba iteracji dla argumentów po obu stronach osi X jest bardzo podobna i często taka sama.

Dla porównania **metoda Newtona** radzi sobie z tą funkcją znacznie lepiej – dla danych miejsc zerowych radzi sobie odpowiednio o **66% i 33% szybciej** z ich wyznaczaniem. Wynika to z jej dobrego uwarunkowania dla niewielkiej odległości od miejsca zerowego oraz wysokiego współczynnika zbieżności. Przy wybraniu punktów oddalonych znacznie bardziej od miejsca zerowego, a w dodatku dla funkcji stromo nachylonej, liczba potrzebnych iteracji jest nawet kilkudziesięciokrotnie większa niż dla analogicznej sytuacji z drugiej strony osi y, gdzie funkcja jest łagodniej nachylona. Metoda ta **zawodzi dla znacznie oddalonych punktów** startowych (jak np. -1000), gdyż zarówno wartość funkcji jak i pochodnej w naszym przypadku dążą do minus nieskończoności. Po stronie argumentów dodatnich, liczba iteracji jest niewielka dzięki nachyleniu funkcji słabszemu niż kwadratowe.

Zadanie 2

Polecenie

Używając metody Laguerre'a proszę znaleźć wszystkie pierwiastki wielomianu

$$f(x) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$$

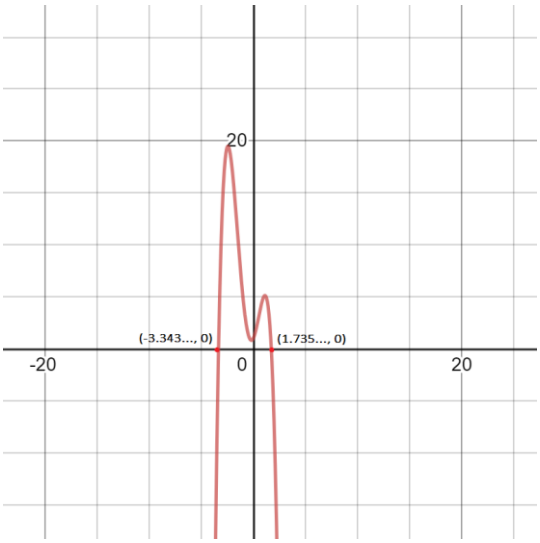
$$[a_4\ a_3\ a_2\ a_1\ a_0\ ] = [-1\ -2\ \ 5\ \ 2\ \ 1]$$

Wprowadzenie teoretyczne

Zasadnicze twierdzenie algebry

Zgodnie z zasadniczym twierdzeniem algebry wielomian n-tego stopnia będzie miał n pierwiastków w przestrzeni zespolonej. Po naszkicowaniu wielomianu, dowiadujemy się o dwóch pierwiastkach rzeczywistych (mających zerową część zespoloną). Z tego wnioskujemy, że pozostałe dwa miejsca zerowe będą istnieć w przestrzeni liczb zespolonych.

Wykres



Metoda Laguerre'a

Metodę Laguerre'a definiuje wzór:

$$x_{k+1} = x_k - \frac{n \cdot f(x_k)}{f'(x_k) \pm \sqrt{(n-1) \cdot [(n-1) \cdot (f'(x_k))^2 - n \cdot f(x_k) \cdot f''(x_k)]}}$$

Gdzie n to stopień wielomianu, a znak w mianowniku jest dobierany tak, by miał on(mianownik) największy moduł.

Metoda ta uchodzi za wyjątkowo sprawną w znajdowaniu zer wielomianów. Dla wielomianu z zerami rzeczywistymi algorytm jest zbieżny z każdego punktu startowego, a w przypadku zer zespolonych bardzo rzadko zdarzają się przypadki niezbieżności.

Schemat rozwiązania

Przy implementacji metody korzystałem z materiałów udostępnionych przez prof. Piotra Tatjewskiego. Mój algorytm kończy przybliżanie miejsca zerowego, gdy odległość funkcji od zera będzie mniejsza od ustalonej tolerancji. Podobnie, jak dla metod poprzednich uznałem wartość 0.0001 za odpowiednią w celach badawczych.

Przy znalezieniu aproksymacji pierwiastka, wykonuję deflację wielomianu czynnikiem liniowym – dzielę aktualny wielomian $f(x)$ przez jednomian $(x - x_k)$.

Kod

```
function [roots] = laguerre(x, coeffs, tolerance)

    F = coeffs;
    n = length(coeffs) - 1;
    roots = zeros(4, 1);
    xk = x;

    while n > 0
        %derivatives
        dF = polyder(F);
        d2F = polyder(dF);

        %max iterations for each polynomial of nth degree
        for i = 1:25
```

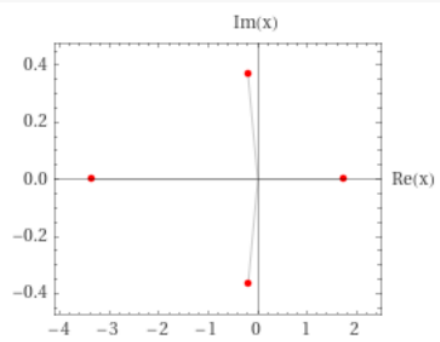
```
            % f(x), f'(x), f''(x)
            Fx = polyval(F, xk);
            dFx = polyval(dF, xk);
            d2Fx = polyval(d2F, xk);

            %tolerance reached
            if abs(Fx) < tolerance
                break;
            end

            %nominator
            nomin =  n * Fx;

            %denominator
            z = (n - 1) * ((n - 1) * dFx^2 - n * Fx * d2Fx);
            zp = dFx + sqrt(z);
            zn = dFx - sqrt(z);

            %choose which denominator to use
            if abs(zp) > abs(zn)
                denomin = zp;
            else
                denomin = zn;
            end

            fprintf('Iteration %d, x = %.5f + %.5fi, |Fx| = %.5f\n', i, real(xk), imag(xk),
               abs(Fx));

            %calculate next x
            xk = xk - nomin/denomin;
        end

        %f(xk) for approximity of root
        y = abs(polyval(F, xk));

        fprintf('Root found at x = %.5f + %.5fi. |Fx| = %.5f\n', real(xk), imag(xk), y);

        %add calculated root to output
        roots(n) = xk;

        % https://uk.mathworks.com/help/matlab/ref/deconv.html
        % deflation
        [ F, R ] = deconv(F, [1, -xk]);

        % polynomial degree decreased by 1
        n = n - 1;
    end
end
```

Wyniki

Obliczone przez algorytm miejsca zerowe wielomianu

	real(x)	imag(x)
x0	-3.3433	0
x1	1.7357	0
x2	-0.1962	0.3658i
x3	-0.1962	- 0.3658i

Zaznaczone punkty w przestrzeni zespolonej



Liczba iteracji dla różnych punktów startowych

x startowy	n iteracji
0.01	8
1	10
10	9
100	9
-0,19 + 0.365i	6
1.73	7

Wnioski

Dla tego wielomianu nie udało mi się znaleźć punktu niezbieżności algorytmu pomimo wykonania wielu testów wykraczających poza zawartość tego sprawozdania. Na podstawie wyników, a także biorąc pod uwagę analizę algorytmu, można przyjąć, że algorytm jest zbieżny dla dowolnych punktów startowych. Jednocześnie metoda zgodnie z teorią odznacza się niezwykle dużą szybkością, gdyż w zaledwie 10 (lub mniej) iteracjach wyznacza wszystkie 4 miejsca zerowe dla tolerancji rzędu jednej-dziesięciotysięcznej.

Dodatki do zadania 1

Izolacja przedziałów startowych dla metod bisekcji i Newtona

```
function [roots] = find_range_and_call(type)
    % type == 0 => Call bisection
    % type != 0 => Call newton
    a = 0;
    b = 1;
    beta = 1.05;
    roots = zeros(2, 1);
    x1 = a;
    x2 = b;

    for k = 1:2
        for iter = 1:25
            if evaluate(x1) * evaluate(x2) < 0


                if type == 0
                    fprintf('Bisection call in range: [%d, %d]\n', x1, x2);
                    roots(k) = bisection(x1, x2, 0.0001);
                else
```

```
                    fprintf('Newton call with argument %d\n', x1);
                    roots(k) = newton(x1, 0.0001);
                end

                x1 = x2;
                x2 = b + 2;
                break;

            elseif abs(evaluate(x1)) < abs(evaluate(x2))
                x1 = x1 + beta * (x1 - x2);

            else
                x2 = x2 + beta * (x2 - x1);
            end
        end
    end
end
```

Wartości funkcji i pochodnej w punkcie x

```
function [value] = evaluate(x)

    value = 7.2 - 2*x - 5*exp(-x);
end

function [value] = derivative(x)

    value = 5 * exp(-x) - 2;
end
```