MNUM PROJEKT, zadanie 3.11

Bartosz Cywiński

Spis treści

Znaleźć wszystkie zera funkcji

$f(x) = 0.55xsin(x) - \ln(x+2)$, w przedziale $[2,12]$

Metoda regula falsi

Opis algorytmu

Startujemy z początkowego przedziału $[a,b] = [a_0, b_0]$ izolacji pierwiastka. Na początku sprawdzam warunek konieczny niezawierania pierwiastka - $f(a) * f(b) > 0$. Jeśli warunek ten jest spełniony, to znaczy, że w przedziale wyznaczonym przez punkty a_0 i b_0 nie znajduje się żaden pierwiastek.

W każdej n-tej iteracji przedział dzielony jest punktem c_n na dwa najczęściej nierówne podprzedziały, prostą łączącą na płaszczyźnie (f,x) punkty $(f(a_n), a_n)$ i $(f(b_n), b_n)$, przecinającą oś rzędnych w punkcie oznaczonym c_n.

Punkt c_n wyznaczany jest według wzoru:

$$c_n = b_n - \frac{f(b_n)(b_n - a_n)}{f(b_n) - f(a_n)} = \frac{a_n f(b_n) - b_n f(a_n)}{f(b_n) - f(a_n)}$$

Następnie obliczane są iloczyny $f(a_n) * f(c_n)$ i $f(c_n) * f(b_n)$, nowy przedział zawierający pierwiastek wybierany jest jako ten z dwóch podprzedziałów, któremu odpowiada iloczyn ujemny.

Procedura zostanie zatrzymana, jeśli zostanie spełniony przynajmniej jeden z trzech warunków:

- Zostanie osiągnięta maksymalna liczba iteracji zdefiniowana jako n_{max}
- $|f(c_n)| \leq \delta$, gdzie δ to założona dokładność rozwiązania
- $b_n - a_n \leq \epsilon$, ϵ to założona długość przedziału izolacji

Kod solwera

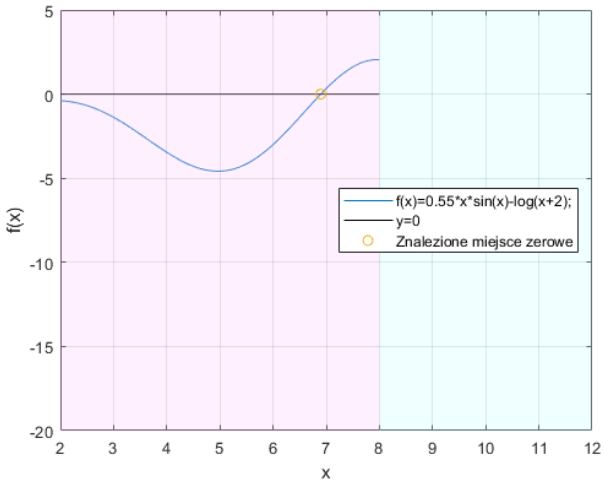
```
function [c, n] = regulaFalsi(a0, b0, delta, epsilon, n_max, f)
%REGULAFALSI Znajdywanie miejsc zerowych funkcji
% za pomocą metody regula falsi (false position)
% [a0,b0] - początkowy przedział izolacji pierwiastka
% delta - założona dokładność rozwiązania
% epsilon - założona długość przedziału izolacji
% n_max - maksymalna liczba iteracji
% f - funkcja, której są szukane pierwiastki


a = a0;
b = b0;

if f(a) * f(b) > 0
    error('Obszar nie zawiera pierwiastka');
end

n = 1;
err = inf;

while n < n_max && err > delta && (b - a) > epsilon
    c = (a*f(b) - b*f(a)) / (f(b) - f(a));

    if f(a)*f(c) < 0
        b = c;
    else
        a = c;
    end

    err = abs(f(c));
    n = n + 1;
end
end
```

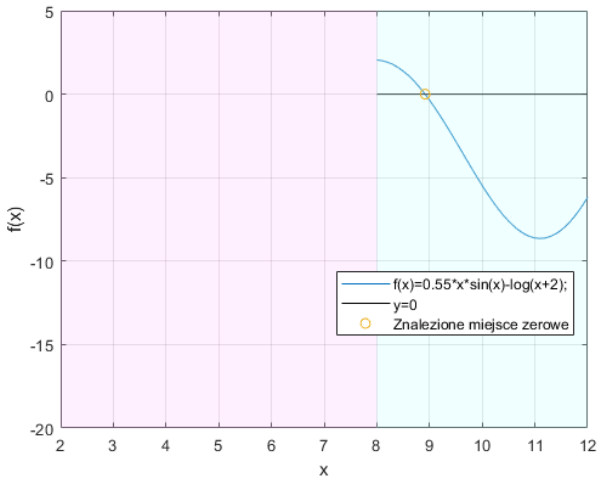
Kod programu testującego

```
function [a0_value, b0_value, c, c_value, n] = findRootRegulaFalsi(a0, b0, delta, epsilon,
n_max)

a0_value = f1(a0);
b0_value = f1(b0);

[c, n] = regulaFalsi(a0, b0, delta, epsilon, n_max, @f1);
```

```
c_value = f1(c);

plot_f(2, 12, 0.1, @f1, "brak", c);
end
```

Wyniki





Metoda regula falsi				
Przedział początkowy	Wartości funkcji na krańcach przedziału	Punkt końcowy	Wartość funkcji w punkcie końcowym	Liczba iteracji
[2,8]	[-0.3861, 2.0506]	6.8989	0.0053	6
[2,8]	[-0.3861, 2.0506]	6.8972	8.8818e-16	13
[8,12]	[2.0506, -6.1804]	8.9136	0.0082	4
[8,12]	[2.0506, -6.1804]	8.9156	-8.8818e-16	13

Metoda Newtona

Opis algorytmu

Metoda Newtona zakłada aproksymacje funkcji jej liniowym przybliżeniem wynikającym z uciętego rozwinięcia w szereg Taylora w aktualnym punkcie x_n

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

Następny punkt x_{n+1} wynika z przyrównania do zera sformułowanej lokalnej liniowej aproksymacji funkcji $f(x)$, tzn. z równania

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

Co prowadzi do zależności iteracyjnej

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

Procedura zostanie, jeśli zostanie spełniony przynajmniej jeden z dwóch warunków:

- Zostanie osiągnięta maksymalna liczba iteracji zdefiniowana jako n_{max}
- $|f(x_{n+1})| \leq \delta$, gdzie δ to założona dokładność rozwiązania

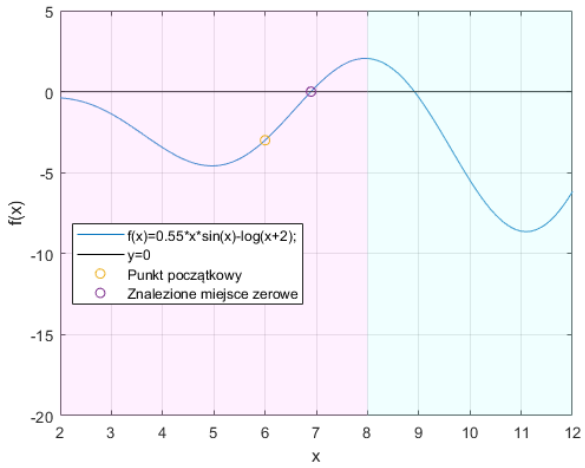
Kod solwera

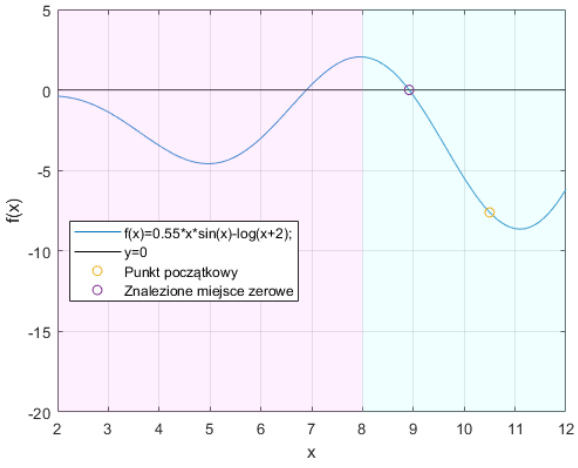
```
function [x_next, n] = newtonMethod(x0, delta, n_max, f, df)
%NEWTONMETHOD Znajdywanie miejsc zerowych funkcji
% za pomocą metody Newtona (stycznych)
% x0 - punkt początkowy
% delta - założona dokładność rozwiązania
% n_max - maksymalna liczba iteracji
% f - funkcja, której są szukane pierwiastki
% df - pochodna funkcji, której są szukane pierwiastki

n = 1;
x_curr = x0;
x_next = x0;
err = inf;

while n < n_max && err > delta
    x_curr = x_next;
    x_next = x_curr - (f(x_curr)/df(x_curr));
    err = abs(f(x_next));
    n = n + 1;
end
end
```

Kod programu testującego

```
function [x0_value, x_end, x_end_value, n] = findRootNewtonMethod(x0, delta, epsilon, n_max)

x0_value = f1(x0);
[x_end, n] = newtonMethod(x0, delta, epsilon, n_max, @f1, @df1);
x_end_value = f1(x_end);

plot_f(2, 12, 0.1, @f1, x0, x_end);

end
```

Wyniki





Metoda Newtona				
Punkt początkowy	Wartości funkcji w punkcie początkowym	Punkt końcowy	Wartość funkcji w punkcie końcowym	Liczba iteracji
6	-3.0015	6.8972	-1.8041e-05	4
6	-3.0015	6.8972	8.8818e-16	6
10.5	-7.6060	8.9156	-4.7438e-08	6
10.5	-7.6060	8.9156	-8.8818e-16	7
2	-0.3861	-3.2376 + 1.0862i	-4.9960e-16 + 8.8818e-16i	10
12	-6.1804	12.9558	2.2204e-15	10

Wnioski

W metodzie regula falsi koniecznym jest, aby zadany przedział początkowy był przedziałem izolacji pierwiastka – w przedziale musi znajdywać się dokładnie jeden pierwiastek. Przedziały izolacji obu pierwiastków są zaznaczone na wykresach dwoma innymi kolorami. Metoda regula falsi jest zawsze zbieżna, jednakże może być ona wolno zbieżna – gdy jeden z końców izolacji pierwiastka pozostaje stały.

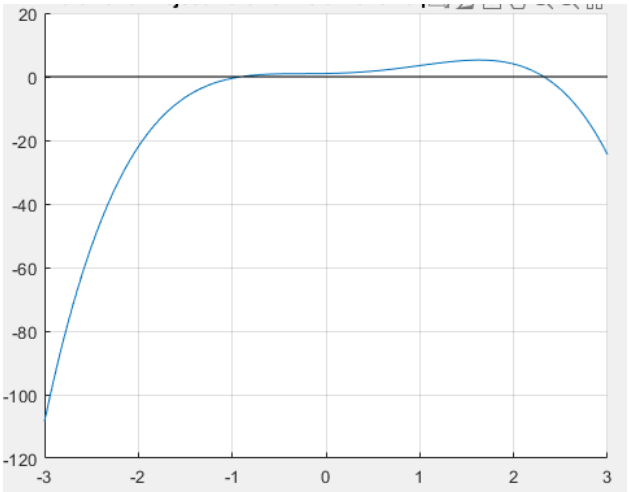
Metoda Newtona natomiast jest zbieżna lokalnie - jeśli zaczynamy ją stosować w punkcie zbytnio oddalonym od rozwiązania (poza obszarem atrakcji pierwiastka), to może być ona rozbieżna. W dwóch ostatnich wynikach tabeli dla metody Newtona pokazuję przykłady rozbieżności metody.

Na podstawie uzyskanych przeze mnie wyników widać, że metoda Newtona szybciej znajduje miejsca zerowe funkcji od metody regula falsi. Jest to wynikiem tego, że metoda Newtona jest szczególnie efektywna, kiedy w otoczeniu pierwiastków funkcja jest stroma, a tak jest w moim przypadku.

Widać też, że im większy przedział tym wolniej pierwiastek jest znajdowany w przypadku metody regula falsi. Gdy przedział miał długość 6 znalezienie dość dokładnego przybliżenia pierwiastka zajęło dłużej niż w przypadku, gdy przedział miał długość 4.

Znaleźć wszystkie pierwiastki wielomianu

$$f(x) = -x^4 + 1.5x^3 + 1.5x^2 + 0.5x + 1$$



Metoda Műllera MM2

Opis algorytmu

Każdy wielomian n-tego stopnia posiada dokładnie n pierwiastków – zespolonych lub rzeczywistych, przy czym mogą być one też pojedyncze lub wielokrotne.

Wersja metody wykorzystująca informację nie o wartości wielomianu w kolejnych trzech punktach, ale o wartości wielomianu i jego pochodnych, pierwszego i drugiego rzędu w aktualnym punkcie (przybliżeniu zera) x_k. Wersja nieco efektywniejsza obliczeniowo z tego powodu, że obliczenie wartości wielomianu w $k+1$ punktach jest nieco kosztowniejsze niż obliczenie wartości wielomianu i jego k kolejnych pochodnych w jednym punkcie.

Z definicji paraboli $y(z) = az^2 + bz + c$, dla $z \stackrel{\text{def}}{=} x - x_k$ w punkcie $z = 0$ bezpośrednio wynika:

$$\begin{aligned} y(0) &= c = f(x_k) \\ y'(0) &= b = f'(x_k) \\ y''(0) &= 2a = f''(x_k) \end{aligned}$$

Co prowadzi do wzoru na pierwiastki:

$$z_{+,-} = \frac{-2f(x_k)}{f'(x_k) \mp \sqrt{\left(f'(x_k)\right)^2 - 2f(x_k)f''(x_k)}}$$

Do przybliżenia zera α bierzemy pierwiastek paraboli o mniejszym module:

$$x_{k+1} = x_k + z_{min}$$

Gdzie

$$z_{min} = \begin{cases} z_+, gdy \ \left|b + \sqrt{b^2 - 4ac}\right| \geq |b - \sqrt{b^2 - 4ac}| \\ z_-, w\ przeciwnym\ przypadku \end{cases}$$

Ponadto po znalezieniu pojedynczego pierwiastka α należy, przed wyznaczaniem następnego, uprościć wielomian dzieląc go przez czynnik $(x - \alpha)$ zawierający znaleziony pierwiastek – deflacja czynnikiem liniowym. Uzyskany wielomian niższego rzędu jest prostszy i nie wyznaczymy ponownie tego samego pierwiastka (chyba że jest wielokrotny).

Do deflacji liniowej wykorzystałem schemat Hornera prosty:

$$\begin{aligned} &q_{n+1} = 0, \\ &q_i = a_i + q_{i+1}\alpha, \qquad i = n, n-1, \dots, 0 \end{aligned}$$

Kod solwera

```
function [x_next, n] = MM2(x0, delta, n_max, f, df, ddf)
%MM2 Funkcja znajdująca pierwiastek wielomianu metodą MM2

x_next = x0;
n = 1;
err = inf;

while n < n_max && err > delta
    x_curr = x_next;
    s_root = sqrt((polynomialValue(df, x_curr)^2) - 2*polynomialValue(f,
x_curr)*polynomialValue(ddf, x_curr));
```

```
    z_plus = -2*polynomialValue(f, x_curr) / (polynomialValue(df, x_curr) + s_root);
    z_minus = -2*polynomialValue(f, x_curr) / (polynomialValue(df, x_curr) - s_root);

    if abs(polynomialValue(df, x_curr) + s_root) >= abs(polynomialValue(df, x_curr) - s_root)
        z_min = z_plus;
    else
        z_min = z_minus;
    end

    x_next = x_curr + z_min;
    err = abs(polynomialValue(f, x_next));
    n = n + 1;
end
end
```

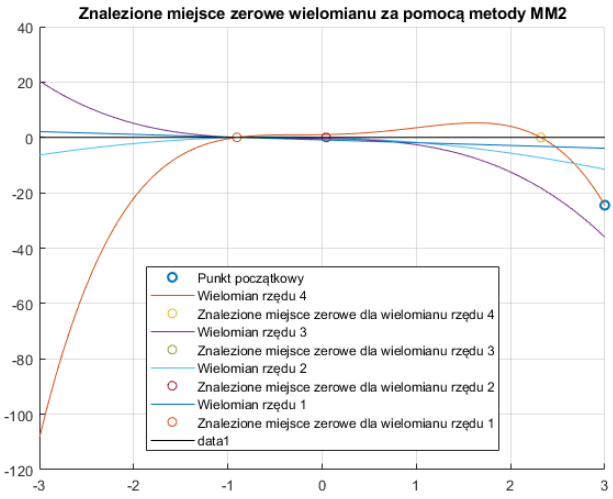
```
function [q] = deflationHorner(a, alpha)
%DEFLATIONHORNER Funkcja przeprowadzająca deflację
% schematem Hornera

n = size(a, 1) - 1;
q = zeros(n+1, 1);

for i = n:-1:1
    q(n-i+2) = a(n-i+1) + q(n-i+1)*alpha;
end

q = q(2:n+1);
end
```

Kod programu testującego

```
function [x0_value, alphas, alphas_values, ns] = findRootsMM2(x0, delta, n_max)
% Początkowe wartości współczynników wielomianu
a = [-1 1.5 1.5 0.5 1]';

alphas = zeros(size(a,1)-1,1);
alphas_values = zeros(size(a,1)-1,1);
ns = zeros(size(a,1)-1,1);
x0_value = polynomialValue(a, x0);

figure(1);
lin = linspace(-3, 3)';

% Wykres punktu początkowego
scatter(x0, polynomialValue(a, x0), "DisplayName","Punkt początkowy", "LineWidth",1.5);

title('Znalezione miejsce zerowe wielomianu za pomocą metody MM2');

hold on;
grid on;

% W każdej iteracji upraszczamy wielomian do wielomianu niższego rzędu
for i = 1:size(a,1)-1
    % Znalezienie pierwiastka wielomianu
```

```
    f = a;
    % Wyznaczenie współczynników pochodnych
    df = polynomialDerivative(f);
    ddf = polynomialDerivative(df);

    % Znalezienie pierwiastka
    [alphas(i), ns(i)] = MM2(x0, delta, n_max, f, df, ddf);
    alphas_values(i) = polynomialValue(f, alphas(i));

    % Wykres wielomianu i znalezionego miejsca zerowego w danej iteracji
    plot(lin, polynomialValue(f, lin), "DisplayName", "Wielomian rzędu " + (size(a,1) - 1));
    scatter(alphas(i), polynomialValue(f, alphas(i)), "DisplayName", "Znalezione miejsce
zerowe dla wielomianu rzędu " + (size(a,1) - 1));
    % Deflacja czynnikiem liniowym
    a = deflationHorner(a, alphas(i));
end
line(xlim(), [0,0], 'LineWidth', 0.5, 'Color', 'k');
legend("Location","best");
hold off;
end
```

Wyniki



Metoda MM2				
Punkt początkowy	Wartości funkcji w punkcie początkowym	Punkt końcowy	Wartość funkcji w punkcie końcowym	Liczba iteracji
3	-24.5000	2.3197 + 0.0000i	1.0e-05 * 0.4043 - 0.2999i	4
		0.0419 - 0.6895i	1.0e-05 * 0.0000 - 0.0000i	6
		0.0419 + 0.6895i	1.0e-05 * -0.0000 + 0.0000i	2
		-0.9035 - 0.0000i	1.0e-05 * 0.0000 + 0.0000i	2
100	-98484949	2.3197 - 0.0000i	1.0e-07 * 0.0036 + 0.0016i	15
		0.0419 + 0.6895i	1.0e-07 * 0.0014 - 0.1134i	12
		0.0419 - 0.6895i	1.0e-07 * 0.0000 + 0.0000i	2
		-0.9035 + 0.0000i	1.0e-07 * 0.0000 + 0.0000i	2
-1000000	-1.0000e+24	-0.9035 - 0.0000i	1.0e-07 * 0.0004 - 0.0003i	43
		0.0419 + 0.6895i	1.0e-07 * -0.2151 + 0.4512i	28
		0.0419 - 0.6895i	1.0e-07 * 0.0000 + 0.0000i	3
		2.3197 - 0.0000i	1.0e-07 * 0.0002 + 0.0000i	2
1000000	-1.0000e+24	2.3197 + 0.0000i	1.0e-04 * -0.1152 - 0.2171i	41
		0.0419 + 0.6895i	1.0e-04 * -0.0000 + 0.0000i	29
		0.0419 - 0.6895i	1.0e-04 * 0.0517 - 0.0000i	2
		-0.9035 - 0.0000i	1.0e-04 * -0.0000 + 0.0000i	2

Wnioski

Algorytm bardzo efektywnie i skutecznie znajduje pierwiastki wielomianu startując z różnych punktów początkowych. Widać na wynikach wyraźną tendencję, że każdy kolejny pierwiastek jest znajdywany szybciej – jest to zasługa deflacji czynnikiem liniowym. Warto zauważyć, że w zależności od punktu początkowego pierwiastki mogą być znajdywane w różnej kolejności.