Obliczenia inżynierskie

Projekt 2

Rozwiązywanie nieliniowych równań algebraicznych

Autor

Jakub Strzelczyk **325325**

Prowadzący

dr inż. Jakub Wagner

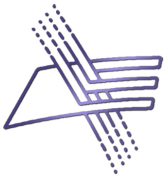
---

8 czerwca 2024

Oświadczam, że niniejsza praca, stanowiąca podstawę do uznania osiągnięcia efektów uczenia się z przedmiotu Obliczenia inżynierskie została wykonana przeze mnie samodzielnie.

Jakub Strzelczyk

Spis treści



Wydział Elektroniki
i Technik Informacyjnych

POLITECHNIKA WARSZAWSKA

1. Lista symboli matematycznych

— x - zmienna skalarne
— $\boldsymbol{x}$ - wektor liczb rzeczywistych
— $f(x)$ - funkcja matematyczna
— $f'(x)$ - pochodna funkcji
— $f''(x)$ - druga pochodna funkcji
— Δ_i - błąd bezwzględny w i-tej iteracji algorytmu iteracyjnego

2. Wprowadzenie

Ważnym problemem obliczeniowym, który często występuje w pracy inżyniera jest numeryczne poszukiwanie rozwiązań równań nieliniowych, które przybierają następującą postać:

$$f(x) = 0 \tag{1}$$

Zazwyczaj nie istnieje bezpośredni algorytm rozwiązania równania nieliniowego, gdyż charakteryzuje się ono tym, że może mieć wiele rozwiązań lub nie posiadać żadnego rozwiązania. Do rozwiązywania takich równań używa się metod iteracyjnych. Algorytm iteracyjny rozwiązywania równania $f(x) = 0$ ma następującą postać [1]:

$$x_{i+1} = \phi(x_i, x_{i-1}, \ldots) \quad \text{dla} \quad i = 0, 1, 2, \ldots \tag{2}$$

gdzie x_{i+1} oznacza wartość przybliżenia rozwiązania w $i+1$-szej iteracji.

Szczególnym przypadkiem jest jednoargumentowy skalarny algorytm iteracyjny, który zdefiniowany jest następującym wzorem:

$$x_{i+1} = \phi(x_i) \quad \text{dla} \quad i = 0, 1, 2, \ldots \tag{3}$$

W celu oceny dokładności powyższego algorytmu można obliczyć błąd bezwzględny i-tego przybliżenia, który wynosi:

$$\Delta_i = x_i - x \tag{4}$$

gdzie x oznacza dokładną wartość rozwiązania równania.

Istnieje także możliwość określenia zbieżności lokalnej na podstawie parametrów C - współczynnika lokalnej zbieżności oraz ρ - wykładnika lokalnej zbieżności, która jest zapewniona, jeśli dla każdego przybliżenia początkowego należącego do najbliższego otoczenia punktu x istnieje co najmniej jedno przybliżenie początkowe $x^{(0)}$, dla którego:

$$|\mathbf{x}_i - \mathbf{x}| \xrightarrow{i \to \infty} 0 \tag{5}$$

Parametry C oraz ρ zdefiniowane są następująco:

$$|\mathbf{x}_i - \mathbf{x}| \cong C\,|\mathbf{x}_i - \mathbf{x}|^{\rho} \tag{6}$$

natomiast skalarny jednoargumentowy algorytm iteracyjny jest lokalnie zbieżny, jeśli $C < 1$ albo $C < \infty$ dla $\rho = 2, 3, \ldots$.

Jednymi z najbardziej popularnych metod iteracyjnych są:

— **Metoda bisekcji**

Dla $f(x)$ ciągłej w $[a_0, b_0]$ spełniającej warunek $f(a_0) \cdot f(b_0) < 0$ metoda bisekcji zdefiniowana jest następującymi wzorami:

$$x_0 = \frac{1}{2}(a_0 + b_0) \tag{7}$$

oraz:

$$x_{i+1} = \frac{1}{2}(a_{i+1} + b_{i+1}), \quad \text{przy czym} \begin{cases} a_{i+1} = a_i, b_{i+1} = x_i & \text{gdy} \quad f(a_i) \cdot f(x_i) < 0 \\ a_{i+1} = x_i, b_{i+1} = b_i & \text{gdy} \quad f(a_i) \cdot f(x_i) > 0 \end{cases} \tag{8}$$

dla $i = 0, 1, 2, \ldots$

Przykładowo, lokalną zbieżność metody bisekcji charakteryzują parametry: $\rho = 1$ oraz $C = \frac{1}{2}$.

— **Metoda regula falsi** - określona następującym wzorem:

$$x_{i+1} = x_i - \frac{x_i - x_0}{f(x_i) - f(x_0)} f(x_i) \quad \text{dla} \quad i = 1, 2, 3, \ldots \tag{9}$$

— **Metoda Newtona** - określona następującym wzorem:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \quad \text{dla} \quad i = 0, 1, 2, \ldots \tag{10}$$

— **Metoda Mullera** - określona następującym wzorem:

$$x_{i+1} = x_i - \frac{2c_i}{b_i + \text{sgn}(b_i)\sqrt{b_i^2 - 4a_i c_i}} \quad \text{dla} \quad i = 1, 2, 3, \ldots \tag{11}$$

gdzie:

$$\begin{bmatrix} a_i \\ b_i \end{bmatrix} = \begin{bmatrix} -(x_i - x_{i-1})^2 & (x_i - x_{i-1}) \\ -(x_i - x_{i-2})^2 & (x_i - x_{i-2}) \end{bmatrix}^{-1} \begin{bmatrix} f(x_i) - f(x_{i-1}) \\ f(x_i) - f(x_{i-2}) \end{bmatrix}$$

oraz $c_i = f(x_i)$.

Celem projektu jest wyznaczanie rozwiązań równań nieliniowych za pomocą przedstawionych metod, a także analiza błędów bezwzględnych, jakimi obarczone są przybliżenia rozwiązania uzyskanego w kolejnych iteracjach.

3. Zadanie 1

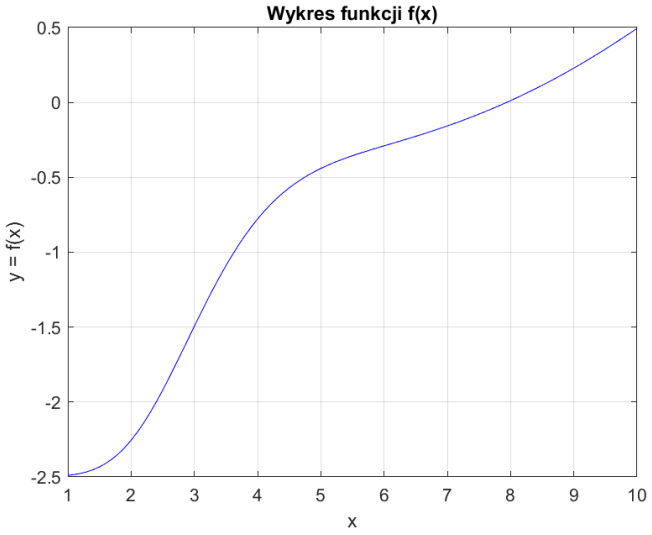
Obliczyłem następujące równanie w MATLABie, korzystając z funkcji `fzero`:

$$f(x) = 0 \tag{12}$$

gdzie:

$$f(x) \equiv 2\left[\exp\left(-\left(\frac{x}{8} - 1\right)^6\right)\right]^{12} + 0.001x^3 - 2.5, \quad x \in [1, 10] \tag{13}$$

W tym celu zdefiniowałem wektor $\boldsymbol{x}$ z wartościami od 1 do 10 z krokiem co 0.1, a następnie napisałem funkcję, która wyznacza wartość $f(x)$ dla zadanej wartości zmiennej x. Korzystając z wbudowanej funkcji `plot`, narysowałem wykres prezentujący zmienność funkcji $f(x)$ w zakresie $x \in [1, 10]$. Odczytałem z wykresu przybliżoną wartość rozwiązania. Na koniec użyłem funkcji `fzero` do znalezienia miejsca zerowego w pobliżu odczytanej z wykresu wartości. Odczytane z okna poleceń w MATLABIE rozwiązanie równania wynosi: **7.937**. Na rysunku 1. przedstawiam wykres funkcji $f(x)$ dla $x \in [1, 10]$.



Rysunek 1. Wykres funkcji $f(x)$

Wyznaczona wartość miejsca zerowego jest zgodna z przewidywaniami na podstawie wykresu - zbliżona do wartości 8.

4. Zadanie 2

4.1. Metodyka i wyniki doświadczeń

Napisałem funkcję, służącą do rozwiązywania nieliniowych równań algebraicznych za pomocą metody bisekcji. Wykonuje się ona dopóki błąd bezwzględny przybliżenia w i-tej iteracji osiągnie wartość mniejszą niż Δ, która jest podawana jako trzeci argument funkcji. Wartość przybliżeń jest obliczana na podstawie wzorów (7) oraz (8), a następnie zwrócony zostaje wektor, zawierający przybliżenia rozwiązania uzyskane w kolejnych iteracjach. W tym zadaniu użyłem formatu long do wyświetlenia zwracanego przez funkcję wektora w celu podkreślenia różnic pomiędzy wynikami uzyskanymi w poszczególnych powtórzeniach. Po napisaniu kodu, przetestowałem opracowaną funkcję, wykorzystując ją do rozwiązania równania (12) dla $\Delta = 10^{-14}$.

Początkowe przybliżenie, uzyskane w wyniku zastosowania metody bisekcji, wynosi:

$$x_0 = \frac{1}{2}(x_{min} + x_{max}) = \frac{1}{2}(1 + 10) = 5.5 \tag{14}$$

Jest ono pierwszym elementem zwracanego wektora. Wraz z wykonywaniem kolejnych iteracji, wartość przybliżeń coraz bardziej zbiega do wartości wyznaczonej za pomocą funkcji `plot`. Ostatnie przybliżenie zostało uzyskane dla 49-tej z kolei iteracji (nie licząc przybliżenia początkowego). Korzystając z następującego wzoru:

$$\Delta_i = x_i - x \tag{15}$$

wyznaczyłem wartość błędu bezwzględnego dla ostatniej iteracji - jako x przyjąłem wartość wyznaczoną w poprzednim zadaniu. Otrzymałem następujący wynik: 9.77e-15.

4.2. Dyskusja wyników eksperymentów numerycznych

Otrzymany wynik jest mniejszy niż wartość $\Delta = 10^{-14}$, co potwierdza uzyskanie oczekiwanego stopnia dokładności. Metoda bisekcji gwarantuje zbieżność dla funkcji ciągłych zmieniających znak w wybranym przedziale, co zostało potwierdzone wynikami eksperymentów.

4.3. Wnioski

Metoda bisekcji jest przydatna w rozwiązywaniu nieliniowych równań algebraicznych. Po zastosowaniu odpowiedniej ilośći iteracji możliwe jest uzyskanie przybliżenia bliskiego rzeczywistej wartości rozwiązania. Za pomocą metody bisekcji łatwo można kontrolować błąd bezwzględny przybliżenia wyniku i nie zależy on od badanej funkcji, natomiast główną wadą tej metody jest to, że ciąg przybliżeń zbiega do dokładnego wyniku bardzo powoli. Uzyskanie dokładnego przybliżenia może okazać się czasochłonne.

5. Zadanie 3

5.1. Metodyka i wyniki doświadczeń

Napisałem w środowisku MATLAB trzy funkcje do rozwiązywania nieliniowych równań algebraicznych, wykorzystując metody: *regula falsi*, Newtona oraz Mullera. Do obliczania wartości w kolejnych iteracjach korzystałem odpowiednio z wzorów (9), (10), (11).

Przed zaimplementowaniem metody Newtona obliczyłem pochodną funkcji $f(x)$ w następujący sposób:

$$f'(x) = 2 \cdot 12 \cdot \left[\exp\left(-\left(\frac{x}{8} - 1\right)^6\right)\right]^{11} \cdot \exp\left(-\left(\frac{x}{8} - 1\right)^6\right) \cdot (-6) \cdot \left(\frac{x}{8} - 1\right)^5 \cdot \left(-\frac{x^2}{16}\right) + 0.003x^2 \tag{16}$$

$$f'(x) = 9\left[\exp\left(-\left(\frac{x}{8} - 1\right)^6\right)\right]^{12} x^2 \left(\frac{x}{8} - 1\right)^5 + 0.003x^2 \tag{17}$$

Składnia funkcji napisanych do rozwiązania tego zadania jest zgodna z funkcją opracowaną w Zadaniu 2. Jedynie w przypadku metody Newtona należy podać dodatkowy argument - zmienną typu `function_handle` - pochodną funkcji $f(x)$. Każda z funkcji zwraca wektor przybliżeń uzyskanych w kolejnych iteracjach, zaczynając

od początkowego przybliżenia x_0.

Warunkiem zakończenia iteracji funkcji jest spełnienie następującego kryterium:

$$\Delta_i = |x_i - x| < \Delta \tag{18}$$

gdzie Δ to trzeci argument funkcji, który zgodnie z treścią zadania przyjąłem jako 10^{-14}. Użyłem formatu long do wyświetlania elementów wektora, aby dokładnie zobaczyć różnice między wynikami przybliżeń w kolejnych iteracjach. Ponadto, dla każdej metody obliczyłem i wypisałem błędy bezwzględne uzyskane w ostatnich iteracjach, co pozwoliło na ocenę dokładności przybliżeń.

5.2. Dyskusja wyników eksperymentów numerycznych

Dla każdej z metod otrzymałem różną liczbę iteracji potrzebnych do spełnienia założonego warunku:
— Metoda Regula Falsi: 45 iteracji
— Metoda Newtona: 8 iteracji
— Metoda Mullera: 9 iteracji
Początkowego przybliżenia nie traktuję jako iterację, jednak jest ono umieszczone jako pierwszy element zwracanego wektora. W przypadku każdej z metod uzyskałem satysfakcjonujące wyniki dotyczące różnicy między ostatecznym przybliżeniem a wynikiem wyznaczonym w Zadaniu 1. Wynoszą one odpowiednio: 5.3291e-15, 8.8818e-16, 0. Każda z tych wartości jest mniejsza niż $\Delta = 10^{-14}$, co świadczy o uzyskaniu odpowiedniego stopnia przybliżenia wyniku. Ponadto dla metody Mullera MATLAB obliczył, iż przybliżony wynik i wynik uzyskany za pomocą funkcji `fzero` są identyczne.

5.3. Wnioski

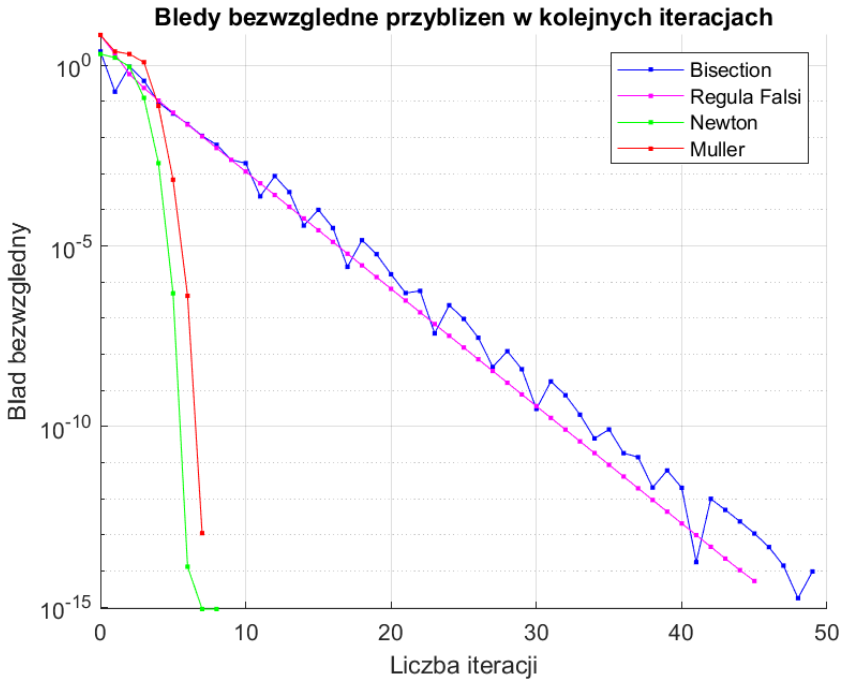
Metoda Newtona okazała się najefektywniejsza pod względem liczby iteracji, co świadczy o jej szybkiej zbieżności. Metoda Mullera również szybko zbiegła do poprawnego przybliżenia, osiągając bardzo dokładny wynik. MATLAB wykazał, że różnica między ostatecznym przybliżeniem a rozwiązaniem uzyskanym w Zadaniu 1 wynosiła 0. Metoda Regula Falsi, choć wymagała największej liczby iteracji, również dostarczyła poprawne wyniki, spełniając założone kryterium dokładności. Wszystkie trzy metody są użyteczne w rozwiązywaniu nieliniowych równań algebraicznych, jednak różnią się efektywnością i liczbą iteracji potrzebnych do osiągnięcia odpowiedniego przybliżenia.

6. Zadanie 4

6.1. Metodyka i wyniki doświadczeń

W celu wykreślenia błędów bezwzględnych, jakimi obarczone są przybliżenia rozwiązania uzyskane w kolejnych iteracjach skorzystałem z wcześniej wyznaczonych wektorów przybliżeń. Od każdego elementu wszystkich wektorów odjąłem wartość wyznaczoną w Zadaniu 1. Następnie za pomocą funkcji `semilogy` narysowałem wykresy zależności błędów bezwzględnych od liczby iteracji. Na osi rzędnych zastosowałem skalę logarytmiczną, aby móc lepiej obserwować występujące różnice, zwłaszcza dla późniejszych iteracji.

Na rysunku 2. przedstawiony jest wykres wartości błędów bezwzględnych w zależności od liczby iteracji dla wszystkich czterech metod.



Rysunek 2. Wykres błędów bezwzględnych przybliżeń w kolejnych iteracjach

6.2. Dyskusja wyników eksperymentów numerycznych

Metoda bisekcji charakteryzuje się powolnym zbiegiem do odpowiedniego przybliżenia. Wykres błędu bezwzględnego dla tej metody pokazuje stopniowe zmniejszanie się błędu w miarę kolejnych iteracji. Wykładnik lokalnej zbieżności dla tej metody wynosi 1, co oznacza, że zbieżność ma charakter liniowy. Współczynnik lokalnej zbieżności dla tej metody wynosi $C = \frac{1}{2}$.

Wykres dla metody regula falsi także pokazuje liniowy spadek błędu, jest nieco bardziej stromy niż w przypadku bisekcji, szybciej osiąga odpowiednią dokładność przybliżenia. Współczynnik lokalnej zbieżności dla metody regula falsi także wynosi 1.

Metoda Newtona charakteryzuje się szybkim tempem zbieżności do odpowiedniego rozwiązania. Wykres błędu bezwzględnego dla tej metody pokazuje gwałtownie opadającą krzywą na osi logarytmicznej, co wskazuje na kwadratową zbieżność. Wykładnik lokalnej zbieżności dla metody Newtona wynosi $\rho = 2$, co oznacza zbieżność kwadratową. Współczynnik lokalnej zbieżności C jest równy $\frac{1}{2}\left|\frac{f''(x)}{f'(x)}\right|$. Metoda Newtona szybko prowadzi do dokładnego przybliżenia rozwiązania.

Wykres dla metody Mullera jest podobny do wykresu dla metody Newtona, następuje szybki spadek wartości błędu. Spadek jest szybszy niż w przypadku metod regula falsi oraz metody bisekcji, natomiast wolniejszy niż w przypadku metody Newtona. W tym przypadku mamy do czynienia z metodą superliniową, gdyż wykładnik lokalnej zbieżności jest większy niż 1, lecz mniejszy niż 2.

6.3. Wnioski

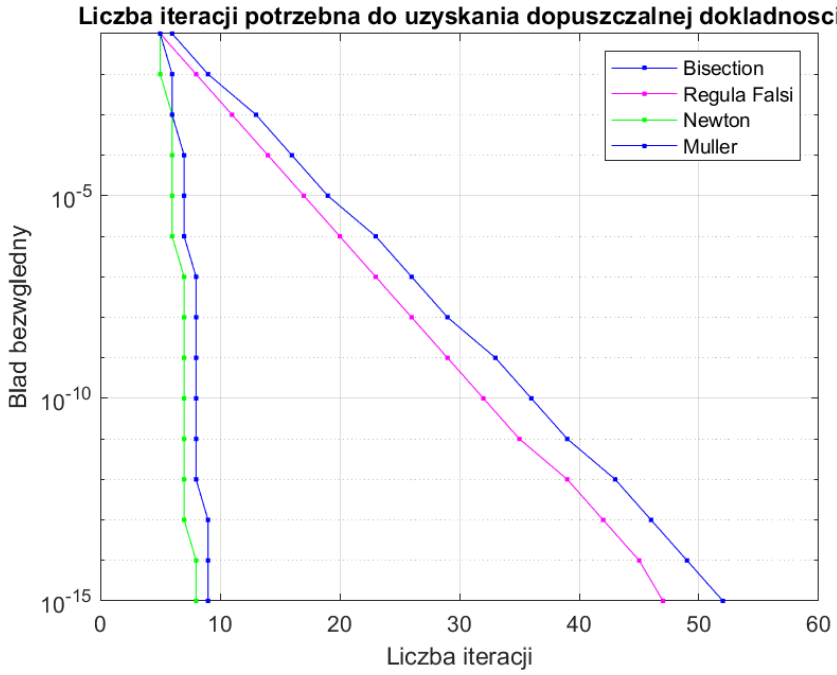
Metody numeryczne do rozwiązywania nieliniowych równań algebraicznych różnią się efektywnością i złożonością. Metody liniowe, takie jak bisekcja, są stabilne i proste, ale wymagają wielu iteracji. Metody superliniowe i kwadratowe, jak Newtona i Mullera, szybko osiągają dokładne wyniki, ale są bardziej skomplikowane obliczeniowo.

7. Zadanie 5

7.1. Metodyka i wyniki doświadczeń

Dla każdej z czterech metod zaimplementowanych w ramach Zadań 2 i 3 wyznaczyłem zależność liczby iteracji potrzebnych do uzyskania dopuszczalnej dokładności rozwiązania od wartości Δ w przedziale $[10^{-15}, 10^{-1}]$. Wykonałem to, wywołując funkcje do wyznaczania wektorów przybliżeń dla wektora wartości Δ składającego się z 15 wartości równomiernie rozmieszczonych w skali logarytmicznej błędów bezwzględnych. Po otrzymaniu wektora przybliżeń, obliczałem jego długość i dodawałem ją jako element do nowo utworzonych wektorów, które następnie wykorzystałem do narysowania wykresu. Na osi rzędnych przyjąłem skalę logarytmiczną.

Na rysunku 3. przedstawiam wykres prezentujący liczbę potrzebnych iteracji potrzebnych do uzyskania wartości określonego błędu bezwzględnego dla wszystkich czterech metod.



Rysunek 3. Zależność liczby iteracji od dopuszczalnej wartości błędu bezwzględnego dla czterech różnych metod

7.2. Dyskusja wyników eksperymentów numerycznych

Metoda bisekcji wymaga znacznie większej liczby iteracji wraz ze zmniejszaniem wartości dopuszczalnego błędu bezwzględnego.

Metoda regula falsi również wymaga dużej liczby iteracji dla małych wartości Δ, choć jest nieco bardziej efektywna niż metoda bisekcji. Dla $\Delta = 10^{-15}$ liczba iteracji jest nadal znaczna.

Metoda Newtona potrzebuje znacznie mniej iteracji, aby osiągnąć dopuszczalną dokładność. Dla $\Delta = 10^{-15}$ metoda ta wymagała tylko kilku iteracji, co świadczy o jej wysokiej efektywności w szybkim osiąganiu dokładnych wyników.

Metoda Mullera również wykazuje szybkie tempo zbieżności, choć wolniejsze niż metoda Newtona. Mimo to, osiąga bardzo dokładne wyniki przy relatywnie małej liczbie iteracji.

7.3. Wnioski

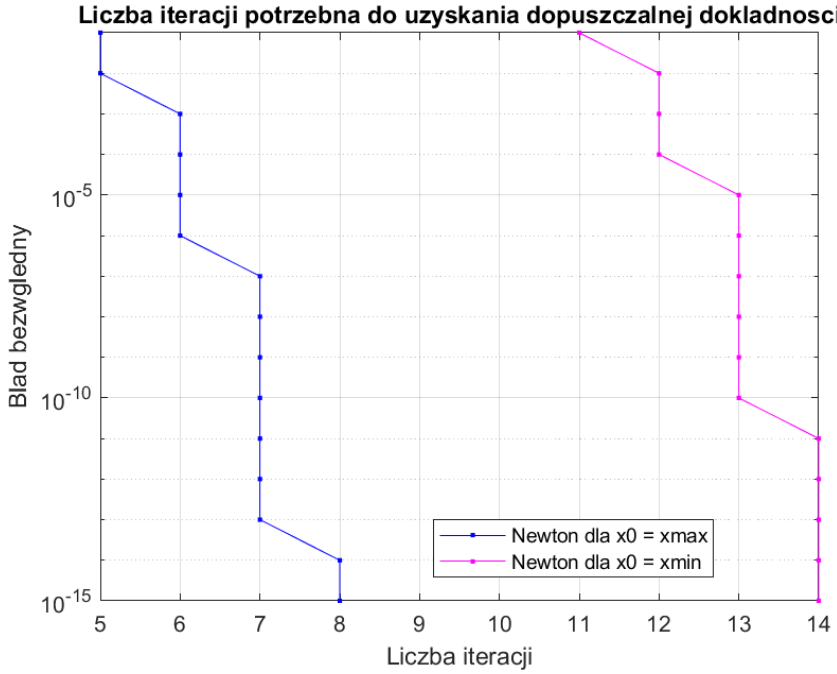
Przeanalizowane metody numeryczne różnią się pod względem efektywności i liczby iteracji potrzebnych do osiągnięcia dopuszczalnej dokładności. Metody liniowe, takie jak bisekcja i regula falsi, wymagają większej liczby iteracji, co może być czasochłonne. Metody bardziej zaawansowane, jak Newtona i Mullera, szybko osiągają dokładne wyniki przy mniejszej liczbie iteracji.

8. Zadanie 6

8.1. Metodyka i wyniki doświadczeń

Wykreślenie i porównanie zależności liczby iteracji, potrzebnej by uzyskać dopuszczalną dokładność rozwiązania za pomocą metody Newtona, dla dwóch różnych punktów startowych: $x_0 = x_{min}$ i $x_0 = x_{max}$ wykonałem analogicznie do Zadania 5. Napisałem tylko dodatkową funkcję Newton2, która jako x_0 przyjmuje x_{max}.

Na rysunku 4. przedstawiam wykres prezentujący liczbę potrzebnych iteracji potrzebnych do uzyskania wartości określonego błędu bezwzględnego dla metody Newton w przypadku dwóch różnych punktów startowych.



Rysunek 4. Zależność liczby iteracji od dopuszczalnej wartości błędu bezwzględnego dla metody Newtona w przypadku dwóch różnych punktów startowych

8.2. Dyskusja wyników eksperymentów numerycznych

Zmiana punktu startowego znacznie wpłynęła na liczbę iteracji potrzebnych do osiągnięcia dopuszczalnej dokładności. W tym przypadku zmiana na mniejszą wartość spowodowała znacznie spowodowała znacznie wolniejszą zbieżność metody Newtona, co skutkowało większą liczbą iteracji - dla niektórych wartości nawet o 7 iteracji.

8.3. Wnioski

Punkt startowy ma kluczowe znaczenie dla efektywności metody Newtona w kontekście liczby iteracji potrzebnych do osiągnięcia dopuszczalnej dokładności rozwiązania. Wybór punktu startowego bliżej rzeczywistego rozwiązania funkcji znacząco przyspieszył proces zbieżności. Zatem odpowiedni dobór punktu początkowego w praktycznych zastosowaniach tej metody ma wpływ, jeśli chodzi o efektywność.

9. Listing programów

Listing 1. Zadanie1.m

```
% wektor liczb od 1 do 10
x = 1:0.1:10;

% funkcja f(x)
function out = f(x)
```

```
    out = 2 .* (exp(-(x ./ 8 - 1) .^ 6)) .^ 12 + 0.001 .* x .^ 3 - 2.5;
end

% wykres f(x)
y = f(x);
figure;
plot(x, y, 'b')
xlabel('x')
ylabel('y = f(x)')
title('Wykres funkcji f(x)')
grid on

% wyznaczenie rozwiazania i wypisanie go na okno polecen
X = fzero(@f, 8);
disp(['Rozwiazanie rownania f(x) = 0 w przedziale [1, 10] to ', num2str(X)]);
```

Listing 2. Zadanie2.m

```
format long;

% Definicja funkcji f(x)
f = @(x) 2 * (exp(-(x / 8 - 1) ^ 6)) ^ 12 + 0.001 * x ^ 3 - 2.5;

% Zakres przedzialu i dokladnosc
range_limits = [1, 10];
delta = 1e-14;

% Obliczenie przyblizen w poszczegolnych iteracjach
approximations = Bisection(f, range_limits, delta);
disp('Przyblizenia w poszczegolnych iteracjach dla metody bisekcji:');
disp(approximations);

% Obliczenie bledu bezwzglednego uzyskanego w ostatniej iteracji
solution_fzero = fzero(f, 8);
final_approximation = approximations(end);

% Wyswietlenie bledu bezwglednego
format shortG;
absolute_error = abs(final_approximation - solution_fzero);
disp('Blad bezwzgledny uzyskany w ostatniej iteracji:');
disp(absolute_error);
```

Listing 3. Zadanie3.m

```
% Zmiana formatu na long
format long;

% Definicja funkcji f oraz wartosci poczatkowych i dokladnosci delta
f = @(x) 2 * (exp(-(x / 8 - 1) ^ 6)) ^ 12 + 0.001 * x ^ 3 - 2.5;
range_limits = [1, 10];
delta = 1e-14;

% Obliczenie przyblizen w poszczegolnych iteracjach dla metody regula falsi
approximations_rf = RegulaFalsi(f, range_limits, delta);
disp('Przyblizenia w poszczegolnych iteracjach dla metody Regula Falsi:');
disp(approximations_rf);

% Obliczenie bledu bezwzglednego uzyskanego w ostatniej iteracji dla
% metody regula falsi
solution_fzero = fzero(f, 8);
final_approximation_rf = approximations_rf(end);
format shortG;
absolute_error_rf = abs(final_approximation_rf - solution_fzero);
disp('Blad bezwzgledny uzyskany w ostatniej iteracji dla metody Regula Falsi:');
disp(absolute_error_rf);
```

```
22
23 % Definicja pochodnej f_der dla metody Newtona
24 f_der = @(x) 9 * (exp(-(x / 8 - 1) ^ 6)) ^ 12 * x ^ 2 * (x / 8 - 1) ^ 5 +
25 0.003 * x ^ 2;
26
27 % Zmiana formatu na long
28 format long;
29
30 % Obliczenie przyblizen w poszczegolnych iteracjach dla metody Newtona
31 approximations_newton = Newton(f, f_der, range_limits, delta);
32 disp('Przyblizenia w poszczegolnych iteracjach dla metody Newtona:');
33 disp(approximations_newton);
34
35 % Obliczenie bledu bezwzglednego uzyskanego w ostatniej iteracji dla metody Newtona
36 final_approximation_newton = approximations_newton(end);
37 format shortG;
38 absolute_error_newton = abs(final_approximation_newton - solution_fzero);
39 disp('Blad bezwzgledny uzyskany w ostatniej iteracji dla metody Newtona:');
40 disp(absolute_error_newton);
41
42 % Zmiana formatu na long
43 format long;
44
45 % Obliczenie przyblizen w poszczegolnych iteracjach dla metody Mullera
46 approximations_muller = Muller(f, range_limits, delta);
47 disp('Przyblizenia w poszczegolnych iteracjach dla metody Mullera:');
48 disp(approximations_muller);
49
50 % Obliczenie bledu bezwzglednego uzyskanego w ostatniej iteracji dla metody Mullera
51 final_approximation_muller = approximations_muller(end);
52 format shortG;
53 absolute_error_muller = abs(final_approximation_muller - solution_fzero);
54 disp('Blad bezwzgledny uzyskany w ostatniej iteracji dla metody Mullera:');
55 disp(absolute_error_muller);
```

Listing 4. Zadanie4.m

```
1 % Zdefiniowanie funkcji i jej pochodnej
2 f = @(x) 2 * (exp(-(x / 8 - 1) ^ 6)) ^ 12 + 0.001 * x ^ 3 - 2.5;
3 f_der = @(x) 9 * (exp(-(x / 8 - 1) ^ 6)) ^ 12 * x ^ 2 * (x / 8 - 1) ^ 5 ...
4 + 0.003 * x ^ 2;
5
6 % Rozwiazanie korzystajac z fzero
7 sol_fzero = fzero(f, 8);
8 delta = 1e-14;
9 range_limits = [1, 10];
10
11 % Wyznaczenie przyblizen
12 appr_bis = Bisection(f, range_limits, delta);
13 appr_rf = RegulaFalsi(f, range_limits, delta);
14 appr_newton = Newton(f, f_der, range_limits, delta);
15 appr_muller = Muller(f, range_limits, delta);
16
17 % Obliczenie bledow bezwzglednych
18 err_bis = abs(appr_bis - sol_fzero);
19 err_rf = abs(appr_rf - sol_fzero);
20 err_newton = abs(appr_newton - sol_fzero);
21 err_muller = abs(appr_muller - sol_fzero);
22
23 x_bis = 0:(length(err_bis)-1);
24 x_rf = 0:(length(err_rf)-1);
25 x_newton = 0:(length(err_newton)-1);
26 x_muller = 0:(length(err_muller)-1);
27
28 % Wykres
```

```
figure;
semilogy(x_bis, err_bis, '.-', 'Color', 'b');
hold on;
semilogy(x_rf, err_rf, '.-', 'Color', 'm');
semilogy(x_newton, err_newton, '.-', 'Color', 'g');
semilogy(x_muller, err_muller, '.-', 'Color', 'b');
grid on;
xlabel('Liczba iteracji');
ylabel('Blad bezwzgledny');
title('Bledy bezwzgledne przyblizen w kolejnych iteracjach');
legend('Bisection', 'Regula Falsi', 'Newton', 'Muller');
legend show;
grid on;
hold off;
```

Listing 5. Zadanie5.m

```
% Definicja funkcji i pochodnej
f = @(x) 2 * (exp(-(x/8 - 1)^6))^12 + 0.001 * x^3 - 2.5;
f_der = @(x) 9 * (exp(-(x / 8 - 1) ^ 6)) ^ 12 * x ^ 2 * (x / 8 - 1) ^ 5 ...
+ 0.003 * x ^ 2;

% Zakres wartosci delty
deltas = logspace(-15, -1, 15);

% Zakres wartosci dla f(x)
range_limits = [1, 10];

% Inicjalizacja tablic do przechowywania wynikow
iterations_bisection = zeros(1, length(deltas));
iterations_regulafalsi = zeros(1, length(deltas));
iterations_newton = zeros(1, length(deltas));
iterations_muller = zeros(1, length(deltas));

for i = 1 : length(deltas)
    delta_val = deltas(i);

    % odejmuje 1 gdyz pierwszy element to przyblizenie poczatkowe
    % ktorego nie licze jako iteracje

    iter_bisection = length(Bisection(f, range_limits, delta_val)) - 1;
    iterations_bisection(i) = iter_bisection;

    iter_regulafalsi = length(RegulaFalsi(f, range_limits, delta_val)) - 1;
    iterations_regulafalsi(i) = iter_regulafalsi;

    iter_newton = length(Newton(f, f_der, range_limits, delta_val)) - 1;
    iterations_newton(i) = iter_newton;

    iter_muller = length(Muller(f, range_limits, delta_val)) - 1;
    iterations_muller(i) = iter_muller;
end

figure;
semilogy(iterations_bisection, deltas, '.-', 'Color', 'b');
hold on;
semilogy(iterations_regulafalsi, deltas, '.-', 'Color', 'm');
semilogy(iterations_newton, deltas, '.-', 'Color', 'g');
semilogy(iterations_muller, deltas, '.-', 'Color', 'b');
grid on;
xlabel('Liczba iteracji');
ylabel('Blad bezwgledny');
title('Liczba iteracji potrzebna do uzyskania dopuszczalnej dokladnosci');
legend('Bisection', 'Regula Falsi', 'Newton', 'Muller');
legend show;
```

```
grid on;
ylim([1e-15 1e-1]);
hold off;
```

Listing 6. Zadanie6.m

```
% Definicja funkcji i pochodnej
f = @(x) 2 * (exp(-(x/8 - 1)^6))^12 + 0.001 * x^3 - 2.5;
f_der = @(x) 9 * (exp(-(x / 8 - 1) ^ 6)) ^ 12 * x ^ 2 * (x / 8 - 1) ^ 5 ...
+ 0.003 * x ^ 2;

% Zakres wartosci delty
deltas = logspace(-15, -1, 15);

% Zakres wartosci dla f(x)
range_limits = [1, 10];

% Inicjalizacja tablic do przechowywania wynikow
iterations_newton = zeros(1, length(deltas));
iterations_newton2 = zeros(1, length(deltas));

for i = 1 : length(deltas)
    delta_val = deltas(i);

    % odejmuje 1 gdyz pierwszy element to przyblizenie poczatkowe
    % ktorego nie licze jako iteracje

    iter_newton = length(Newton(f, f_der, range_limits, delta_val)) - 1;
    iterations_newton(i) = iter_newton;

    iter_newton2 = length(Newton2(f, f_der, range_limits, delta_val)) - 1;
    iterations_newton2(i) = iter_newton2;

end

figure;
semilogy(iterations_newton, deltas, '.-', 'Color', 'b');
hold on;
semilogy(iterations_newton2, deltas, '.-', 'Color', 'm');
grid on;
xlabel('Liczba iteracji');
ylabel('Blad bezwgledny');
title('Liczba iteracji potrzebna do uzyskania dopuszczalnej dokladnosci');
legend('Newton dla x0 = xmax', 'Newton dla x0 = xmin');
legend show;
grid on;
ylim([1e-15 1e-1]);
hold off;
```

Listing 7. Newton.m

```
function out = Newton(f, f_der, range_limits, delta)
    x0 = range_limits(2);
    xi = x0 - f(x0)/ f_der(x0);

    % Inicjalizacja wektora przyblizen
    out = [x0, xi];

    % Petla iteracyjna
    while(abs(out(end) - out(end-1)) > delta)
        xi = xi - f(xi)/ f_der(xi);
        % Zapis nowego przyblizenia
        out = [out, xi];
    end
end
```

Listing 8. Newton2.m

```
function out = Newton(f, f_der, range_limits, delta)
    x0 = range_limits(1);
    xi = x0 - f(x0)/ f_der(x0);

    % Inicjalizacja wektora przyblizen
    out = [x0, xi];

    % Petla iteracyjna
    while(abs(out(end) - out(end-1)) > delta)
        xi = xi - f(xi)/ f_der(xi);
        % Zapis nowego przyblizenia
        out = [out, xi];
    end
end
```

Listing 9. RegulaFalsi.m

```
function out = RegulaFalsi(f, range_limits, delta)
    x0 = range_limits(1);
    xi = range_limits(2);

    % Inicjalizacja wektora przyblizen
    out = [x0, xi];

    %Petla iteracyjna
    while(abs(out(end) - out(end-1)) > delta)
        xi = xi - (xi - x0) * f(xi) / (f(xi) - f(x0));
        out = [out, xi];
    end
end
```

Listing 10. Bisection.m

```
function out = Bisection(f, range_limits, delta)
    xmin = range_limits(1);
    xmax = range_limits(2);

    % sprawdzenie warunku poczatkowego
    if f(xmin) * f(xmax) > 0
        error('f(xmin) * f(xmax) > 0');
    end

    % Inicjalizacja
    delta_i = (xmax - xmin);
    x_i = (xmin + xmax) / 2;

    % Poczatkowe przyblizenie
    out = x_i;

    % Petla iteracyjna
    while (abs(delta_i/2) > delta)
        if(f(xmin) * f(x_i) < 0)
            xmax = x_i;
        else
            xmin = x_i;
        end
        x_i = (xmin + xmax) / 2;
        delta_i = delta_i / 2;
        out = [out, x_i];
    end
end
```

Listing 11. Muller.m

```
function appr = Muller(f, range_limits, delta)
    xmin = range_limits(1);
    xmax = range_limits(2);

    % Poczatkowe przyblizenia
    x0 = xmin;
    x1 = (xmin + xmax) / 2;
    x2 = xmax;

    % Wartosci funkcji w punktach poczatkowych
    f0 = f(x0);
    f1 = f(x1);
    f2 = f(x2);

    % Inicjalizacja wektora przyblizen
    appr = [x0, x1, x2];

    % Petla iteracyjna
    while abs(appr(end) - appr(end-1)) > delta
        % Obliczanie wspolczynnikow ai i bi
        A = [-(x2 - x1)^2, (x2 - x1);
             -(x2 - x0)^2, (x2 - x0)];

        B = [f2 - f1; f2 - f0];

        % Pomnozenie macierzy odwrotnej A przez B
        result = A\B;

        % Przypisanie elementow wynikowej macierzy do zmiennych ai i bi
        ai = result(1);
        bi = result(2);

        % Obliczenie nowego przyblizenia x3
        x3 = x2 - (2 * f2 / (bi + sign(bi)*sqrt(bi^2 - 4 * ai * f2)));

        % Dodanie nowego przyblizenia do wektora
        appr = [appr, x3];

        % Aktualizacja poprzednich punktow
        x0 = x1;
        x1 = x2;
        x2 = x3;

        % Aktualizacja wartosci funkcji w nowych punktach
        f0 = f1;
        f1 = f2;
        f2 = f(x3);
    end
end
```

Literatura

[1] R. Z. Morawski, materiały do przedmiotu Obliczenia inżynierskie, Politechnika Warszawska, Wydział Elektroniki i Technik Informacyjnych, semestr zimowy 2023/24.