Metody obliczeniowe
Laboratorium 2
Rozwiązywanie równań nieliniowych

Uwaga: Instrukcje dotyczące przebiegu zajęć laboratoryjnych zawarte są w części 2 tego dokumentu.

1 Część teoretyczna

Metody rozwiązywania równań nieliniowych:

- metoda bisekcji (połowienia)
- regula falsi
- metoda siecznych
- metoda stycznych (Newtona)
- metoda iteracji prostych (iteracje proste Banacha)
- metoda Pegaza
- metoda Mullera
- metoda Brenta (Matlab, Octave → `fzero`)
- i in.

Tabela 1: Rzędy zbieżności metod

Metoda	Rząd
bisekcji	1
regula falsi	1
siecznych	$\frac{1}{2}(1+\sqrt{5}) \approx 1.62$
Brenta	1.8
Newtona	2
Eulera	3

1.1 Metoda bisekcji

Metoda ta polega na iteracyjnym wyznaczaniu coraz mniejszych przedziałów, co do których mamy pewność, że zawierają zero funkcji. Załóżmy, że dana jest funkcja $f(x)$, która jest ciągła w przedziale $[a, b]$ i przyjmuje na jego brzegach wartości o przeciwnych znakach ($f(a)f(b) < 0$). W takim przypadku przedział $[a, b]$ musi zawierać pierwiastek funkcji. Za przybliżone rozwiązanie możemy uznać punkt leżący w środku przedziału $[a, b]$, czyli:

$$c = \frac{a+b}{2} \tag{1}$$

W przypadku gdy $f(c) = 0$ miejsce zerowe zostało odnalezione. Z reguły jednak $f(c) \neq 0$ i poszukiwania trzeba kontynuować. Zero będzie znajdować się albo w przedziale $[a, c]$, albo w przedziale $[c, b]$. Aby wybrać właściwy przedział wystarczy sprawdzić wartość funkcji w punkcie c. Jeśli $f(a)f(c) < 0$ kontynuujemy proces w przedziale $[a, c]$, jeśli $f(c)f(b) < 0$ oznacza to, że w następnej iteracji wartość c będzie trzeba podstawić w miejsce a. Iteracje można przerywać jeśli odnaleziona wartość $f(c) < tol$, gdzie tol jest zadaną przez nas wartością.

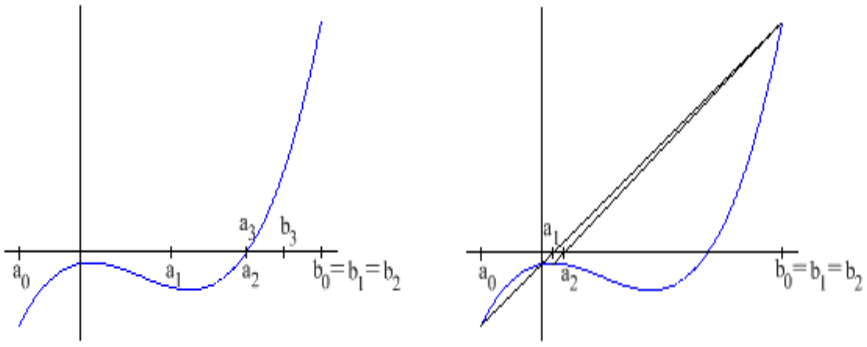
Metoda bisekcji jest wolno zbieżna (zbieżność jest liniowa), gdyż w ogóle nie korzysta z informacji jakie daje kształt funkcji $f(x)$. Jej niewątpliwą zaletą jest natomiast to, że jest pewna. Bisekcja sprawuje się dobrze, tam gdzie inne (szybsze) metody mają problemy. W obliczeniach numerycznych jest rzadko używana osobno. Za to często stosuje się ją do początkowego przybliżenia przedziału zawierającego miejsce zerowe funkcji, aby potem skorzystać z szybszych metod.

```
Dane: a, b, M, δ, ϵ
Wyniki: n,x̃, f(x̃)
   fa ← f(a); fb ← f(b);
   e ← b − a;
   if sgn(fa) = sgn(fb) then
      return error;
   end if
   for n ← 1 to M do
      e ← e/2;
      c ← a + e; {c = (a + b)/2}
      fc ← f(c);
      if |e| < δ or |fc| < ϵ then
         return n, c, fc
      end if
      if sgn(fc) ≠ sgn(fa) then
         b ← c; fb ← fc;
      else
         a ← c; fa ← fc;
      end if
   end for
```

Algorytm metody bisekcji zapisany w pseudokodzie.

1.2 Regula falsi

Pierwszym sposobem na to, aby wykorzystać informacje o kształcie funkcji do przyspieszenia zbieżności poszukiwania zera jest aproksymacja funkcji $f(x)$ za



(a) (b)

Rysunek 1: (a) metoda bisekcji; (b) - regula falsi

pomocą linii prostej. W metodzie regula falsi, podobnie jak metodzie połowienia zaczynamy od przedziału $[a, b]$, na którym spełniony jest warunek $f(a)f(b) < 0$. Przybliżoną wartością pierwiastka w każdym następnym kroku będzie miejsce przecięcia osi OX i linii przechodzącej przez punkty $(a, f(a))$ i $(b, f(b))$, tzn:

$$c = b - \frac{b-a}{f(b) - f(a)} f(b) \tag{2}$$

Jeśli zero znajduje się w przedziale $[a, c]$ zmienną a pozostawiamy bez zmian i podstawiamy $b = c$; w przeciwnym przypadku przypisujemy $a = c$, z kolei b pozostaje takie samo.

```
Dane: a, b, M, δ, ε
Wyniki: n,x̃, f(x̃)
   fa ← f(a); fb ← f(b);
   e ← b − a;
   if sgn(fa) = sgn(fb) then
      return error;
   end if
   for n ← 1 to M do
      e ← e/2;
      c ← b − (b − a)/(fb − fa) · fb
      fc ← f(c);
      if |e| < δ or |fc| < ε then
         return n, c, fc
      end if
      if sgn(fc) ≠ sgn(fa) then
         b ← c; fb ← fc;
      else
         a ← c; fa ← fc;
      end if
   end for
```

Algorytm regula falsi zapisany w pseudokodzie.

1.3 Metoda siecznych

Podobnie jak w metodzie regula falsi, tak i w tej funkcja $f(x)$ jest aproksymowana linią prostą. Jednak metoda siecznych nie wymaga, aby w każdej iteracji sprawdzać przedział, w którym znajduje się pierwiastek. Aby rozpocząć algorytm wymagane są dwa przybliżenia pierwiastka (x_0, x_1) i nie musi być spełniony warunek $f(x_0)f(x_1) < 0$. Kolejne aproksymowane wartości pierwiastka znajdujemy (podobnie jak w powyższej metodzie) jako miejsce przecięcia się osi OX i linii przechodzącej przez dwie ostatnie aproksymacje pierwiastka $(x_{k-1}, f(x_{k-1}))$ i $(x_k, f(x_k))$.

$$x_{k+1} = x_k - \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} f(x_k) \tag{3}$$

```
Dane: x_{n-1}, x_n, M, δ, ε
Wyniki: n,x̃, f(x̃)
  f_{n-1} ← f(x_{n-1}); f_n ← f(x_n);
  for n ← 1 to M do
    x_{n+1} = x_n − (x_n − x_{n-1})/(f_n − f_{n-1}) · f_n ;
    f_{n+1} = f(x_{n+1}) ;
    x_{n-1} ← x_n; f_{n-1} ← f_n;
    x_n ← x_{n+1}; f_n ← f_{n+1};
    if |x_n − x_{n-1}| < δ or |f_n| < ε then
      return n, x_n, f_n;
    end if
  end for
```

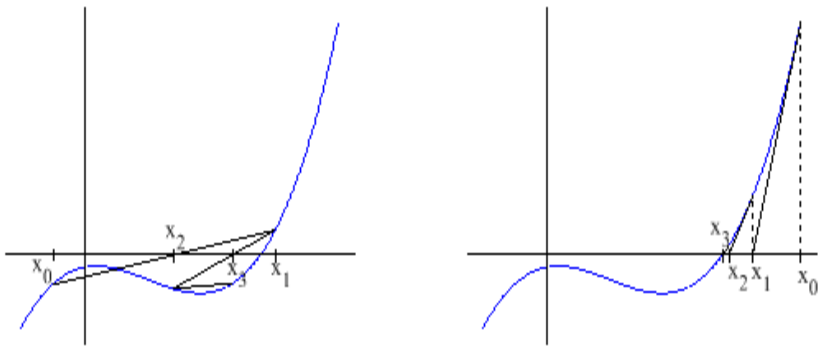
Algorytm metody siecznych zapisany w pseudokodzie.

1.4 Metoda stycznych/Newtona

W metodzie Newtona jako aproksymację funkcji $f(x)$ przyjmujemy styczną do funkcji w x_k-tym miejscu. Następnym przybliżenie miejsca zerowego jest przecięcie stycznej z osią OX. Minusem takiego podejścia jest to, że dodatkowo jest wymagana znajomość pochodnej zadanej funkcji. Metoda ta również nie gwarantuje zbieżności procesu. Jest ona jednak najszybszą z podstawowych metod i stąd często sięga się po nią jako pierwszą. Aby obliczyć $k+1$-sze przybliżenie pierwiastka funkcji $f(x)$ należy zastosować wzór:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \tag{4}$$

```
Dane: x_0, M, δ, ε
Wyniki: n,x̃, f(x̃)
  x_n ← f(x_0); f_n ← f(x_n);
  if |f_n| < ε then
    return 0, x_n, f_n;
  end if
  for n ← 1 to M do
    x_{n+1} ← x_n − f_n/f'(x_n);
    f_{n+1} ← f(x_{n+1});
    if |x_{n+1} − x_n| < δ or |f_n + 1| < ε then
      return n, x_{n+1}, f_{n+1};
    end if
    x_n ← x_{n+1};
    f_n ← f_{n+1};
  end for
```



(a) (b)

Rysunek 2: (a) metoda siecznych; (b) - metoda stycznych

Algorytm metody stycznych zapisany w pseudokodzie.

W przeciwieństwie do dwóch pierwszych metod, metody siecznych oraz stycznych nie gwarantują zbieżności.

1.5 Warunki stopu

1. $|x_n - x_{n-1}| < tol_1 \wedge |x_{n+1} - x_n| \geqslant |x_n - x_{n-1}|$
2. grube stopowanie: $|x_n - x_{n-1}| < tol_1$
3. grube stopowanie: $|f(x_n)| < tol_2$

1.6 Funkcje biblioteczne Octave: fzero, roots

W programach Octave/Matlab istnieją dwie funkcje służące do znajdywania miejsc zerowych. Pierwsza z nich to `fzero`. Implementuje ona metodę Brenta-Dekkera opublikowaną w roku 1973, która jest połączeniem metod bisekcji, siecznych odwrotnej interpolacji kwadratowej. Najprostszym sposobem jej wywołania jest:

```
>> fzero(fun, x0);
```

fun jest zadaną funkcją, której pierwiastków poszukujemy. Parametr x0 może być wartością skalarną (pierwsze przybliżenie) lub też wektorem dwuelementowym (wtedy elementy wektora **muszą** zawierać pierwiastek). W tym drugim przypadku $fun(x0(1)) \cdot fun(x0(2))$ musi być mniejsze od zera.

Funkcja

```
>> roots(c);
```

oblicza pierwiastki wielomianu, którego współczynnikami są elementy wektora c. Przykładowo jeśli interesuje nas znalezienie zer funkcji $x^5 - 3x^4 + 2x^2 - 9x + 5$ wtedy parametr c jest wektorem postaci: [1 -3 0 2 -9 5].

1.7 Zadanie

W programie Octave zapoznaj się z działaniem i sposobami wywoływania poniżej podanych funkcji:

- eval
- sign
- error
- disp

2 Ćwiczenia

Wykonaj następujące zadania:

1. Pobierz pliki zgodnie z instrukcjami prowadzącego i zapisz je katalogu o nazwie `Lab2` utworzonym na swoim dysku. Plik `main.m` będzie skryptem, z wnętrza którego będziemy wywoływać wszystkie pisane przez nas funkcje i polecenia.

   UWAGA: Nie umieszczamy tych plików w katalogu głównym Octave'a! Aby móc wywołać pliki zmieniamy katalog bieżacy ($\rightarrow$ Laboratorium 0 $\rightarrow$ polecenia `cd, pwd, dir`).

2. Napisz funkcję `wielomian.m` będącą implementacją wzoru

$$f(x) = x^3 + x^2 - 3x - 3$$

   Przy pisaniu funkcji należy użyć operatorów **tablicowego** mnożenia czy potęgowania. W przeciwnym przypadku próba jej wywołania dla wektorów zwróci błąd.

   Wywołanie tej funkcji jest umieszczone wewnątrz skryptu `main.m`.

3. Jeśli funkcja `wieloman.m` zostanie napisana poprawnie skrypt `main.m` powinien wygenerować wykres funkcji $f(x)$ w przedziale $< -3, 3 >$.

   Uwaga: Zwróć uwagę na sposób wywołania funkcji wielomian.m. Nie jest ona wywoływana w skrypcie `main.m` bezpośrednio. Utworzona jest dodatkowo pomocnicza zmienna `fun` będąca funkcją anonimową. Funkcja anonimowa `fun` przyjmuje jeden argument, który jest następnie przekazywany do funkcji `wielomian`. Ten (wydawałoby się nieco okrężny) sposób obliczania wartości funkcji `wielomian` ma tę zaletę, że w kolejnych liniach kodu pozwala na przekazanie funkcji anonimowej `fun` jako parametr do funkcji `f_bisect` (i kolejnych) wyłącznie poprzez podanie jej nazwy (a więc identycznie jak przekazywane są do funkcji "zwykłe" parametry - liczby/macierze). Z kolei wewnątrz funkcji `f_bisect` możemy korzystać z przekazanego parametru `fun` jakby to była normalna funkcja (patrz linie 20, 21 w `f_bisect.m`).

4. Na podstawie wykresu można określić przedział izolacji pierwiastka dodatniego funkcji $f(x)$. Będziemy go poszukiwać przy pomocy metod omówionych na wykładzie. Przedział ten jest już zdefiniowany poprzez zmienne `a`, `b` wewnątrz skryptu `main.m`.

5. Uzupełnij kod funkcji implementującej metodę bisekcji. Znajduje się on w pliku `f_bisect.m`. Jej wywołanie wygląda następująco:
   `[y, yc] = f_bisect(fun, a, b, k_max, tol)`

   gdzie

fun	funkcja, dla której szukamy miejsca zerowego
a,b	granice przedziału w którym znajduje się pierwiastek
tol	żądana dokładność
k_max	maksymalna ilość iteracji
c	znalezione przybliżenie pierwiastka
yc	wartość funkcji dla znalezionego przybliżenia pierwiastka

Miejsca, które wymagają uzupełnienia są zaznaczone w kodzie tak jak poniżej:

```
%%% TODO >>>
% Obliczenie kolejnego przyblizenia pierwiastka
% c = ..
%%%<<<<
```

co oznacza, że linię rozpoczynającą się od

```
% c=..
```

należy odkomentować i dokonać w niej implementacji odpowiedniego wzoru, zgodnego z opisem w linii powyżej.

Uwaga: Znacznik `%%%TODO >>>` występuje w pliku `f_bisect.m` w **dwóch** miejscach, czyli edytujemy dwie linie funkcji `f_bisect.m`.

W celu uproszczenia algorytmu, jako kryterium stopu zastosowano ostatni warunek z części 1.5.

Zwróć uwagę na sposób obliczania wartości funkcji podawanej jako argument do funkcji `f_bisect` przy pomocy funkcji bibliotecznej Octave'a `eval`. Metoda ta będzie wykorzystywana podczas pisania kolejnych funkcji.

6. Porównaj stworzoną funkcję z pseudokodem algorytmu podanym na wykładzie.

7. Skopiuj plik `f_bisect.m` i zmień nazwę kopii na `f_rfalsi.m`. Wzorując się na kodzie metody bisekcji zaimplementuj metodę regula falsi.

8. Skopiuj plik `f_rfalsi.m` i zmień nazwę kopii na `f_secant.m`. Wzorując się na kodzie regula falsi zaimplementuj metodę siecznych.

9. Skopiuj plik `f_secant.m` i zmień nazwę kopii na `f_newton.m`. Wzorując się na kodzie metody siecznych zaimplementuj metodę stycznych. Funkcję należy zdefiniować, tak, aby była zgodna z jej wywołaniem znajdującym się w ostatniej linii skryptu `main`.

   W przypadku tej metody wymagane będzie dodatkowo stworzenie funkcji obliczającej pochodną funkcji. W `main.m` do funkcji tej odnosi się zmienna `dfun`. Deklarując `dfun` wzoruj się na anonimowej funkcji `fun`.

10. Porównaj zbieżność poszczególnych metod szukając miejsca zerowego dla tych samych parametrów.

11. Skorzystaj z funkcji omówionych w punkcie 1.6 (`fzero` i `roots`), aby obliczyć pierwiastki równania

$$x^3 + x^2 - 3x - 3 = 0 \tag{5}$$

12. Zaproponuj takie dane wejściowe dla metody stycznych/siecznych, aby dla funkcji posiadającej miejsce zerowe w pobliżu pierwszych/-ego przybliżenia/-żeń metoda nie była zbieżna. Jakie cechy funkcji spowodowały rozbieżność?

    Zadanie to można wykonać modyfikując przybliżenie początkowe dla funkcji `fun1` lub definiując inną/ne funkcję/-je w nowym/-wych pliku.

13. Równanie (5) można przekształcić do postaci, która umożliwia wykorzystanie metody iteracji prostej $x = \Phi(x)$ na kilka sposobów, m.in.:

$$x = \frac{x^3+x^2-3}{3}\ ; \qquad x = \sqrt{-x^3 + 3x + 3};$$
$$x = \frac{-x^3+3}{x-3}; \qquad x = (-x^2 + 3x + 3)^{\frac{1}{3}}$$

    Zbadaj zbieżność metody iteracji prostej dla tych wzorów przyjmując różne przybliżenia początkowe pierwiastka, np: $x_0 = 1$ lub $x_0 = 0.1$.