

Metody Numeryczne

Laboratorium 8: Rozwiązywanie równań nieliniowych

Wykonaj: <imię i nazwisko>

Metoda zaliczenia:

Podczas kąjuć należy wykonać poniższe polecenia oraz udzielić odpowiedzi na pytania zamieszczone w treści zadania.

Wszystkie funkcje wymagane w ramach ćwiczenia należy zaimplementować w pliku `main.py`. Poprawność ich działania należy zweryfikować za pomocą testów jednostkowych dostępnego w pliku `test_main.py`.

Cel zajęć:

Celem zajęć jest poznawanie numerycznych metod rozwiązywania układów równań nieliniowych. W ramach laboratorium przedstawione zostaną metody: *bisekcji*, *siecznych* oraz *Newtona*.

Tematem wiodącym podczas tych zajęć będzie poszukiwanie miejsc zerowych funkcji zdefiniowanej w następujący sposób:

$$f(x) = e^{-2x} + x^2 - 1$$

Funkcja $f(x)$ oraz jej pierwsza i druga pochodna ($f'(x)$, $f''(x)$) zostały zaimplementowane w pliku `main.py`, odpowiednio jako `func()`, `dfunc()` i `ddfunc()`.

Uwagi wstępne:

• Funkcje wymagające implementacji (także zaimplementowane na poprzednich laboratoriach) oznaczone są pochyłą czcionką maszynową (np. `my_func()`).

• W skrypcie stosuje się następujące skróty:

- `np` - `numpy`,
- `sp` - `scipy`.

```
In [34]: #!python -m pip install numpy scipy matplotlib
import main
import numpy as np
import scipy as sp
import matplotlib
import matplotlib.pyplot as plt
from typing import Callable
```

Zadanie 1.

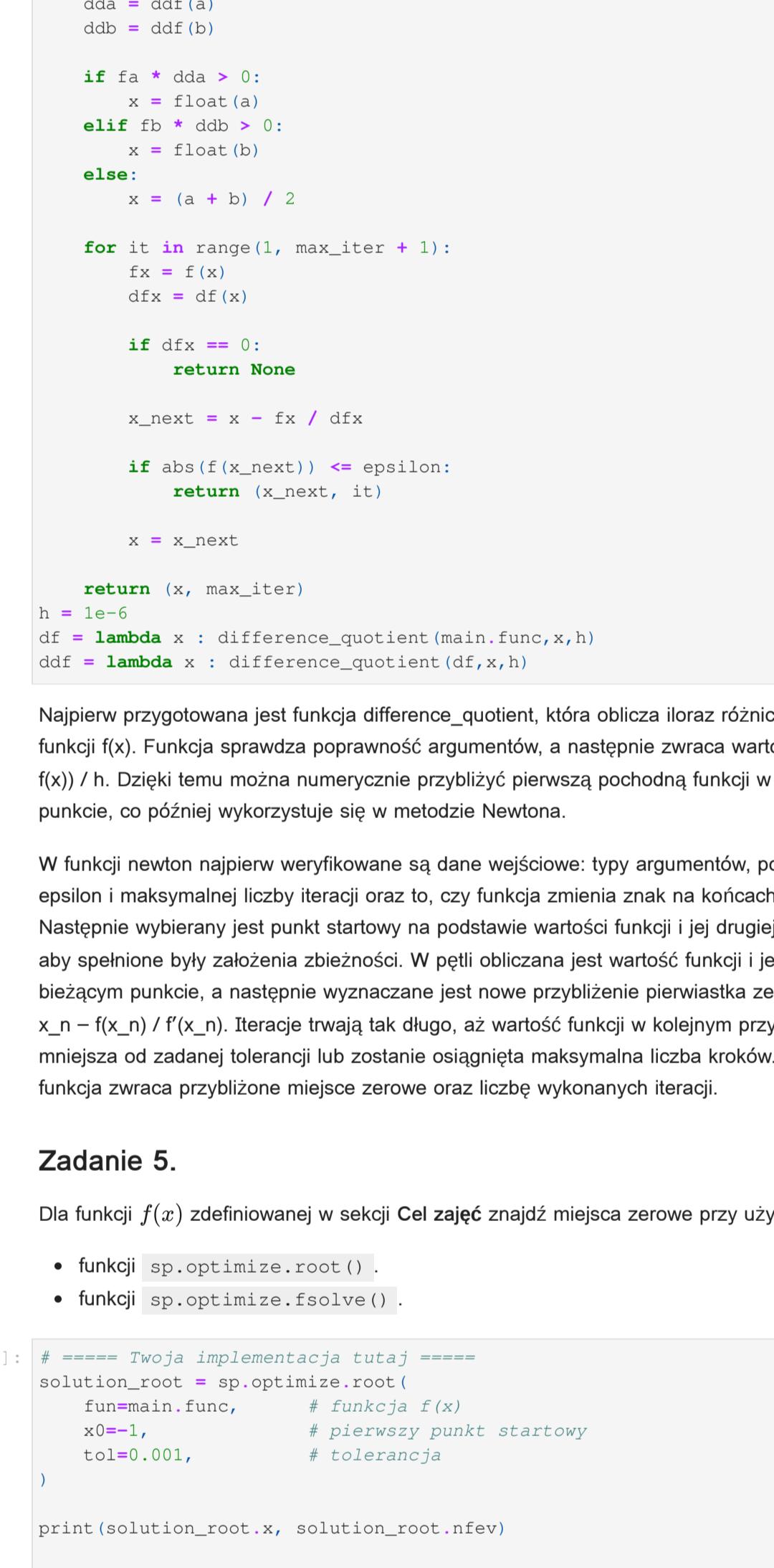
1. Zaproś się na jednym wykresie przebieg funkcji $f(x)$, $f'(x)$ oraz $f''(x)$, w taki sposób, aby na podstawie rysunku można było zgrubnie określić położenie miejsc zerowych funkcji.

2. Na podstawie analizy wykresu określ przedział, w których znajdują się miejsca zerowe funkcji $f(x)$.

3. Odpowiedz w jaki sposób (wykorzystując pierwszą i drugą pochodną) można znaleźć miejsca zerowe funkcji.

```
In [35]: #===== Twoja implementacja tutaj =====
x = np.linspace(-4,4,1000)
y = main.func(x)
yd = main.dfunc(x)
ydd = main.ddfunc(x)

plt.plot(x,y,label="Pierwotna")
plt.plot(x,yd,label="Pochodna")
plt.plot(x,ydd,label="Druga pochodna")
plt.xlabel("Argument x")
plt.ylabel("Wartości")
plt.legend()
plt.title("f(x)=e^-2x + x^2 - 1")
plt.ylim((-1,5))
plt.grid()
```



Na wykresie przedstawiono funkcję $f(x)$ oraz jej pierwszą i drugą pochodną. Na podstawie przecięcia wykresu funkcji z osią OX widać dwa miejsca zerowe. Pierwsze znajduje się w pobliżu zera, a drugie po prawej stronie wykresu. Analizując zmiany znaku funkcji, a także kształtu pochodnych, można wygrubnie określić przedziały, w których znajdują się pierwiastki.

Wyznaczono przedziały miejsc zerowych: pierwszy pierwiastek: $[-0.5, 0.3]$ drugi pierwiastek: $(0.5, 1.0)$

Pierwsza pochodna pomaga określić monotoniczność funkcji, a druga pochodna informuje o wypukłości i może wskazać lepszy punkt startowy dla metod numerycznych.

Zadanie 2.

Najprostszą metodą do wyznaczenia miejsc zerowych funkcji nieliniowej jest *metoda bisekcji*.

Zaimplementuj funkcję `bisection()` pamiętając, że gwarancją zbieżności metody bisekcji dla poszukiwania miejsca zerowego funkcji $f(x)$ na odcinku $[a, b]$ są następujące założenia:

1. Funkcja $f(x)$ jest ciągła w przedziale domkniętym $[a, b]$.
2. Funkcja $f(b)$ przyjmuje różne znaki na końcach przedziału: $f(a)f(b) < 0$.

```
In [44]: def bisection(
    a: int | float,
    b: int | float,
    f: Callable[[float], float],
    epsilon: float,
    max_iter: int,
) -> tuple[float, int] | None:
    if not isinstance(a, (int, float)) or not isinstance(b, (int, float)):
        return None
    if not isinstance(epsilon, float):
        return None
    if not isinstance(max_iter, int):
        return None
    if not callable(f):
        return None
    if a >= b:
        return None
    if epsilon < 0:
        return None
    if f(a) * f(b) >= 0:
        return None

    counter = 0
    x0 = (a + b) / 2

    while abs(f(x0)) > epsilon and counter < max_iter:
        counter += 1

        if f(a) * f(x0) < 0:
            b = x0
        else:
            a = x0

        x0 = (a + b) / 2

    return x0, counter + 1
```

Na początku sprawdzane są wszystkie warunki poprawności danych wejściowych: typy argumentów, poprawność epsilon i max_iter, kolejność przedziału oraz to, czy funkcja zmienia znak na krańcach przedziału. Następnie licznik iteracji i wyznaczany pierwszy punkt jako środek przedziału. W pętli obliczana jest wartość funkcji w aktualnym punkcie i sprawdzane są warunki stopu: osiągnięcie wymaganej dokładności lub przekroczenie maksymalnej liczby iteracji. W każdej iteracji zatrzymywany jest przedział do tej części, w której funkcja zmienia znak.

Nowy środek przedziału staje się kolejnym przybliżeniem rozwiązania. Po zakończeniu algorytmu zwraca znaleziony pierwiastek i liczbę wykonanych iteracji.

Zadanie 3.

Bardziej zaawansowaną metodą wyznaczania miejsc zerowych jest *metoda siecznych*, która stanowi rozwinięcie *metody Regula Falsi*.

Zaimplementuj funkcję `secant()` pamiętając, że gwarancją zbieżności metody siecznych dla poszukiwania miejsca zerowego funkcji $f(x)$ na odcinku $[a, b]$ są następujące założenia:

1. Funkcja $f(x)$ jest ciągła w przedziale domkniętym $[a, b]$.
2. Funkcja $f(b)$ przyjmuje różne znaki na końcach przedziału: $f(a)f(b) < 0$.
3. Pierwsza i druga pochodna funkcji $f(x)$ są ciągłe w przedziale domkniętym $[a, b]$.
4. Pierwsza i druga pochodna funkcji $f(x)$ w przedziale domkniętym $[a, b]$ mają stały znak i są różne od zera.

```
In [45]: def secant(
    a: int | float,
    b: int | float,
    f: Callable[[float], float],
    epsilon: float,
    max_iters: int,
) -> tuple[float, int] | None:
    if not isinstance(a, (float, int)) or not isinstance(b, (float, int)):
        return None
    if not callable(f):
        return None
    if a == b:
        return None
    if max_iters < 0:
        return None

    fa = f(a)
    fb = f(b)
    if fa * fb >= 0:
        return None

    x = (a * fb - b * fa) / (fb - fa)
    fx = f(x)
    counter = 1

    while abs(fx) > epsilon and counter < max_iters:
        if fa * fx < 0:
            b = x
            fb = fx
        else:
            a = x
            fa = fx

        x = (a * fb - b * fa) / (fb - fa)
        fx = f(x)
        counter += 1

    return x, counter
```

Na początku sprawdzane są wszystkie warunki poprawności danych wejściowych: typy argumentów, poprawność epsilon i max_iters, kolejność przedziału oraz to, czy funkcja zmienia znak na krańcach przedziału. Następnie licznik iteracji i wyznaczany pierwszy punkt jako środek przedziału. W pętli obliczana jest wartość funkcji w aktualnym punkcie i sprawdzane są warunki stopu: osiągnięcie wymaganej dokładności lub przekroczenie maksymalnej liczby iteracji. W każdej iteracji zatrzymywany jest przedział do tej części, w której funkcja zmienia znak.

Nowy środek przedziału staje się kolejnym przybliżeniem rozwiązania. Po zakończeniu algorytmu zwraca znaleziony pierwiastek i liczbę wykonanych iteracji.

Zadanie 4.

Inną metodą, wykorzystywaną do poszukiwania miejsc zerowych funkcji, jest *metoda Newtona*, nazywana również metodą stycznych.

Algorytm metody Newtona wykorzystuje wartości pierwsiowej pochodnej, dlatego przed przystąpieniem do jej implementacji przygotuj pomocniczą funkcję `difference_quotient()`, służącą do wyznaczania wartości ilorazu różnicowego dla zadanej funkcji $f(x)$.

1. Podpunkt 1.

```
In [46]: def difference_quotient(
    f: Callable[[float], float], x0: float, h: float
) -> float:
    if not isinstance(x0, (float, int)) or not isinstance(h, (float, int)):
        return None
    if not callable(f):
        return None
    if h == 0:
        return None
    if max_iters < 0:
        return None

    f_x0 = f(x0)
    if f_x0 is None:
        return None
    if h < 0:
        return None
    if not isinstance(h, float):
        return None
    if not isinstance(max_iters, int):
        return None
    if max_iters < 0:
        return None

    f_x1 = f(x0 + h)
    if f_x1 is None:
        return None
    if h == 0:
        return None
    if not callable(f):
        return None
    if not isinstance(h, float):
        return None
    if not isinstance(max_iters, int):
        return None
    if max_iters < 0:
        return None

    return (f_x1 - f_x0) / h
```

Na początku sprawdzane są wszystkie warunki poprawności danych wejściowych: typy argumentów, poprawność epsilon i max_iters, kolejność przedziału oraz to, czy funkcja zmienia znak na krańcach przedziału. Następnie obliczane są wartości funkcji w punktach a i b. Na tej podstawie wyznaczane jest pierwsiowa przybliżenie dla zadanej funkcji $f(x)$.

W poniższej funkcji `newton()` zatrzymywany jest przedział do tej części, w której funkcja zmienia znak na krańcach przedziału. Następnie obliczana jest wartość funkcji w aktualnym punkcie i sprawdzane są warunki stopu: osiągnięcie wymaganej dokładności lub przekroczenie maksymalnej liczby iteracji. W każdej iteracji zatrzymywany jest przedział do tej części, w której funkcja zmienia znak.

Nowy środek przedziału staje się kolejnym przybliżeniem rozwiązania. Po zakończeniu algorytmu zwraca znaleziony pierwiastek i liczbę wykonanych iteracji.

Zadanie 5.

Dla funkcji $f(x)$ zdefiniowanej w sekcji Cel zajęć znajdź miejsca zerowe przy użyciu:

- funkcji `sp.optimize.root()`.
- funkcji `sp.optimize.fsolve()`.

```
In [39]: print(solution_root.x, solution_root.nfev)
```

Na początku sprawdzane są wszystkie warunki poprawności danych wejściowych: typy argumentów, poprawność epsilon i max_iters, kolejność przedziału oraz to, czy funkcja zmienia znak na krańcach przedziału. Następnie licznik iteracji i wyznaczany pierwszy punkt jako środek przedziału. W pętli obliczana jest wartość funkcji w aktualnym punkcie i sprawdzane są warunki stopu: osiągnięcie wymaganej dokładności lub przekroczenie maksymalnej liczby iteracji. W każdej iteracji zatrzymywany jest przedział do tej części, w której funkcja zmienia znak.

W poniższej funkcji `secant()` zatrzymywany jest przedział do tej części, w której funkcja zmienia znak na krańcach przedziału. Następnie licznik iteracji i wyznaczany pierwszy punkt jako środek przedziału. W pętli obliczana jest wartość funkcji w aktualnym punkcie i sprawdzane są warunki stopu: osiągnięcie wymaganej dokładności lub przekroczenie maksymalnej liczby iteracji. W każdej iteracji zatrzymywany jest przedział do tej części, w której funkcja zmienia znak.

Nowy środek przedziału staje się kolejnym przybliżeniem rozwiązania. Po zakończeniu algorytmu zwraca znaleziony pierwiastek i liczbę wykonanych iteracji.

Zadanie 6.

Korzystając z przedziałów wyznaczonych w Zadaniu 1. znajdź miejsca zerowe funkcji $f(x)$ zdefiniowanej w sekcji Cel zajęć, przy użyciu:

- metody bisekcji,
- metody siecznych,
- metody Newtona,

z tolerancją równą $1e-10$.

Zbadaj dokładność (względem rozwiązania z Zadania 5.) i czas obliczeń metod w zależności od liczby iteracji. Wyniki przedstaw na wykresach.

```
In [ ]: fun = main.func
x1_range = [-0.5, 0.3]
x2_range = [0.5, 1.0]
epsilon = 1e-10
max_iters = 1000
tol = 0.001
full_output=True
```

Na początku sprawdzane są wszystkie warunki poprawności danych: typy argumentów, poprawność epsilon i max_iters, kolejność przedziału oraz to, czy funkcja zmienia znak na krańcach przedziału. Następnie licznik iteracji i wyznaczany pierwszy punkt jako środek przedziału. W pętli obliczana jest wartość funkcji w aktualnym punkcie i sprawdzane są warunki stopu: osiągnięcie wymaganej dokładności lub przekroczenie maksymalnej liczby iteracji. W każdej iteracji zatrzymywany jest przedział do tej części, w której funkcja zmienia znak.

Nowy środek przedziału staje się kolejnym przybliżeniem rozwiązania. Po zakończeniu algorytmu zwraca znaleziony pierwiastek i liczbę wykonanych iteracji.

Zadanie 7.

Bardziej zaawansowaną metodą wyznaczania miejsc zerowych jest *metoda Regula Falsi*.

Zaimplementuj funkcję `regula_falsi()` pamiętając, że gwarancją zbieżności metody regula falsi dla poszukiwania miejsca zerowego funkcji $f(x)$ na odcinku $[a, b]$ są następujące założenia:

1. Funkcja $f(x)$ jest ciągła w przedziale domkniętym $[a, b]$.
2. Funkcja $f(b)$ przyjmuje różne znaki na końcach przedziału: $f(a)f(b) < 0$.
3. Pierwsza i druga pochodna funkcji $f(x)$ są ciągłe w przedziale domkniętym $[a, b]$.
4. Pierwsza i druga pochodna funkcji $f(x)$ w przedziale domkniętym $[a, b]$ mają stały znak i są różne od zera.

```
In [47]: def regula_falsi(
    a: int | float,
    b: int | float,
    f: Callable[[float], float],
    epsilon: float,
    max_iters: int,
) -> tuple[float, int] | None:
    if not isinstance(a, (float, int)) or not isinstance(b, (float, int)):
        return None
    if not callable(f):
        return None
    if a == b:
        return None
    if max_iters < 0:
        return None
    if not isinstance(epsilon, float):
        return None
    if not isinstance(max_iters, int):
        return None
    if max_iters < 0:
        return None

    fa = f(a)
    fb = f(b)
    if fa * fb >= 0:
        return None

    x = (a * fb - b * fa) / (fb - fa)
    fx = f(x)
    counter = 1

    while abs(fx) > epsilon and counter < max_iters:
        if fa * fx < 0:
            b = x
            fb = fx
        else:
            a = x
            fa = fx

        x = (a * fb - b * fa) / (fb - fa)
        fx = f(x)
        counter += 1

    return x, counter
```

Na początku sprawdzane są wszystkie warunki poprawności danych wejściowych: typy argumentów, poprawność epsilon i max_iters, kolejność przedziału oraz to, czy funkcja zmienia znak na krańcach przedziału. Następnie licznik iteracji i wyznaczany pierwszy punkt jako środek przedziału. W pętli obliczana jest wartość funkcji w aktualnym punkcie i sprawdzane są warunki stopu: osiągnięcie wymaganej dokładności lub przekroczenie maksymalnej liczby iteracji. W każdej iteracji zatrzymywany jest przedział do tej części, w której funkcja zmienia znak.

Nowy środek przedziału staje się kolejnym przybliżeniem rozwiązania. Po zakończeniu algorytmu zwraca znaleziony pierwiastek i liczbę wykonanych iteracji.

Zadanie 8.

Wyniki przedstawionego wykresu dotyczą czasu obliczeń metod w zależności od liczby iteracji.

W poniższej funkcji `difference_quotient()` zatrzymywany jest przedział do tej części, w której funkcja zmienia znak na krańcach przedziału. Następnie licznik iteracji i wyznaczany pierwszy punkt jako środek przedziału. W pętli obliczana jest wartość funkcji w aktualnym punkcie i sprawdzane są warunki stopu: osiągnięcie wymaganej dokładności lub przekroczenie maksymalnej liczby iteracji. W każdej iteracji zatrzymywany jest przedział do tej części, w której funkcja zmienia znak.

Nowy środek przedziału staje się kolejnym przybliżeniem rozwiązania. Po zakończeniu algory