Metody Obliczeniowe w Nauce i Technice
Natalia Bratek, 24.04.2024
Laboratorium 07

# Pierwiastki równań nieliniowych

## 1. Treści zadań

1.1 Napisz iteracje wg metody Newtona do rozwiązywania każdego z następujących równań nieliniowych:

(a) x cos(x) =1;
(b) $x^3$ - 5x - 6 = 0;
(c) $e^{-x} = x^2-1$.

1.2 **(**a) Pokaż, że iteracyjna metoda $$x_{k+1} = \frac{x_{k-1}f(x_k) - x_k f(x_{k-1})}{f(x_k) - f(x_{k-1})}$$ matematycznie jest równoważna z metodą siecznych przy rozwiązywaniu skalarnego nieliniowego równania f (x) = 0.
(b) Jeśli zrealizujemy obliczenia w arytmetyce zmiennoprzecinkowej o skończonej precyzji, jakie zalety i wady ma wzór podany w podpunkcie (a), w porównaniu ze wzorem dla metody siecznych podanym poniżej?

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

1.3 Zapisz iteracje Newtona do rozwiązywania następującego układu równań nieliniowych.

$x_1^2 + x_1x_2^3 = 9$
$3x_1^2 x_2 - x_2^3 = 4$

# 2. Rozwiązania

## 2.1 Zadanie pierwsze

Założenia dla metody Newtona:

1) W przedziale [a, b] znajduje się dokładnie jeden pierwiastek funkcji f.

2) Funkcja ma różne znaki na krańcach przedziału, tj. f(a) * f(b) < 0
3) Pierwsza i druga pochodna funkcji mają stały znak w tym przedziale.

Schemat iteracyjny wg metody Newtona dla równań nieliniowych:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, k = 1, 2, \ldots.$$

### a) x cos(x) = 1

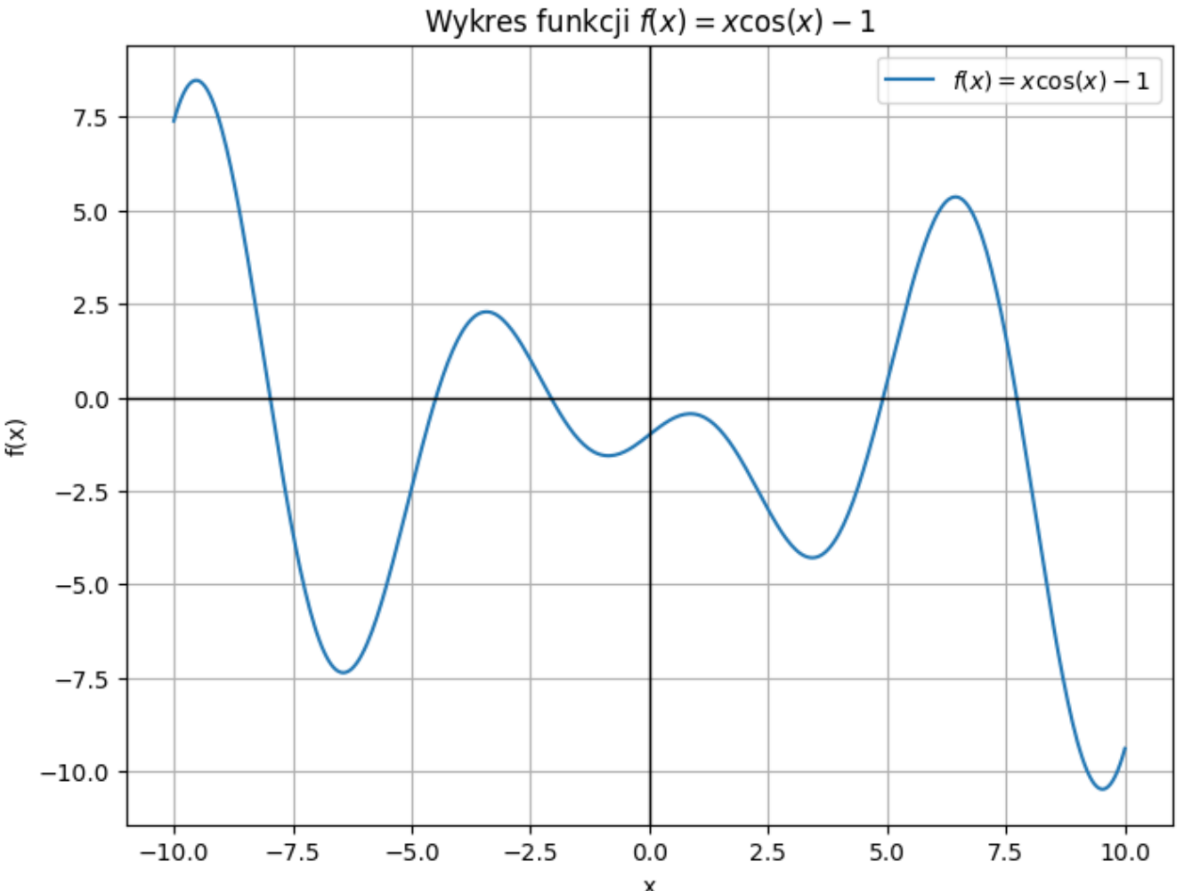
To równanie można przekształcić do postaci

x cos(x) – 1 = 0

```
def f(x):
    return x * np.cos(x) - 1
x_values = np.linspace(-10, 10, 400)
y_values = f(x_values)
plt.figure(figsize=(8, 6))
plt.plot(x_values, y_values, label=r'$f(x) = x \cos(x) - 1$')
plt.axhline(0, color='black', lw=1)
plt.axvline(0, color='black', lw=1)
plt.title(r'Wykres funkcji $f(x) = x \cos(x) - 1$')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.legend()
plt.grid(True)
plt.show()
```

Rys. 1 Implementacja wykresu

Kolejne wykresy mają analogiczną implementacje, różnią się tylko f(x).



Rys.2 Wykres funkcji

Funkcja zawiera więcej niż 1 miejsce zerowe. Dlatego biorę pod uwagę przedział [4.5, 5.5], zawierający jeden pierwiastek dodatni (zał. 1).

Funkcja na przedziale [4, 6] spełnia założenie 2:

$$f(4.5) * f(5.5) < 0$$

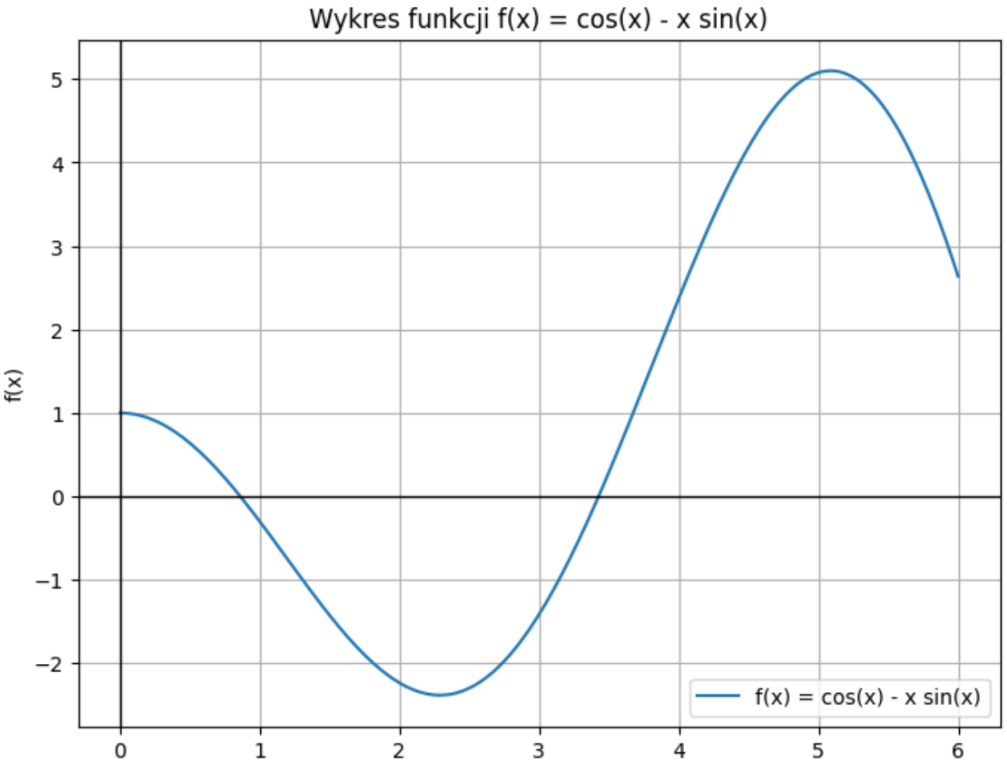
ponieważ

$f(4.5) < 0$ i $f(5.5) > 0$

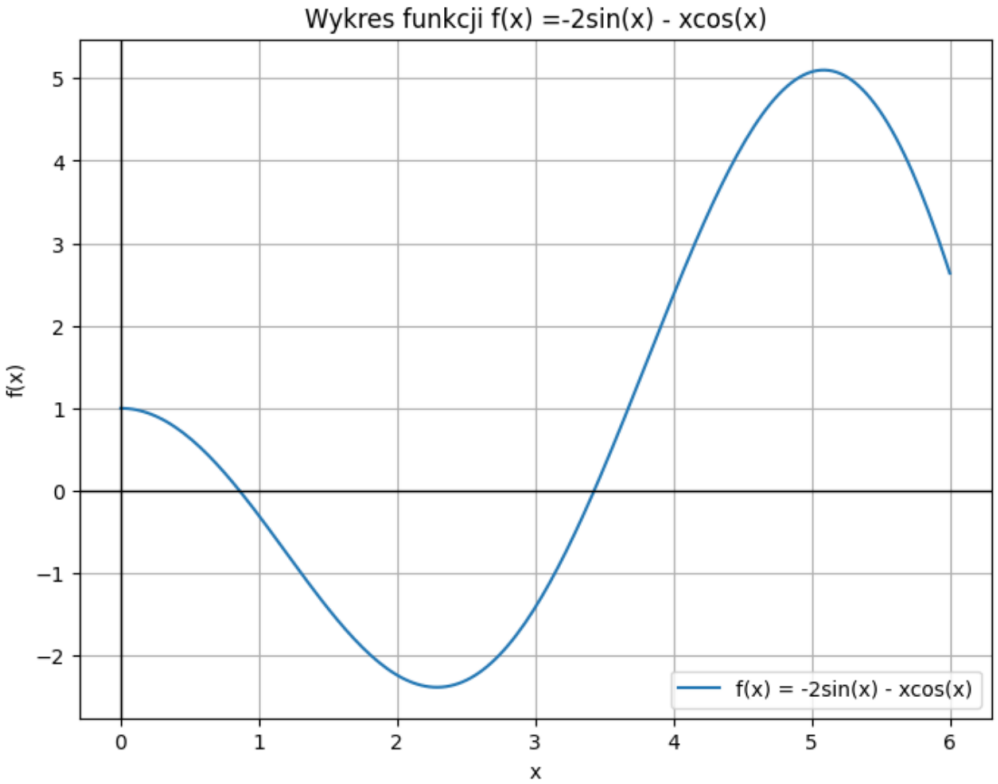
Pochodne funkcji:

$$f'(x) = \cos(x) - xsin(x)$$

$$f''(x) = -2\sin(x) - xcos(x)$$




Rys.3 Wykres pierwszej pochodnej




Rys. 4 Wykres drugiej pochodnej

Z rysunków 3 i 4 widać, że pierwsza i druga pochodna w przedziale [4, 6] maja stały znak. (założenie 3)

Funkcja spełnia założenia.
Iteracje algorytmu Newtona:

$$x_{k+1} = x_k - \frac{x_k \cos(x_k) - 1}{\cos(x_k) - x_k sin(x_k)}$$

$$x_0 = 4.5$$

Według WolframAlpha rozwiązaniem równania xcos(x) – 1 = 0 w przedziale [4.5, 5,5] jest:

$$x \approx 4.917185925287 13\ldots$$

```
def f(x):
    return x * np.cos(x) - 1
def df(x):
    return np.cos(x) - x * np.sin(x)
x0 = 4.5
def newtons_method(f, df, x0, tol=1e-10, max_iter=10):
    xn = x0
    for n in range(0, max_iter):
        fxn = f(xn)
        print("iteracja: ", n, ": ", xn)
        if abs(fxn) < tol:
            print('Znaleziono rozwiązanie po', n, 'iteracjach.')
            return xn
        dfxn = df(xn)
        if dfxn == 0:
            print('Pochodna wynosi zero. Brak rozwiązań.')
            return None
        xn = xn - fxn/dfxn
    print('Przekroczono maksymalną liczbę iteracji. Brak rozwiązań.')
    return None
newton = newtons_method(f, df, x0)
newton
```

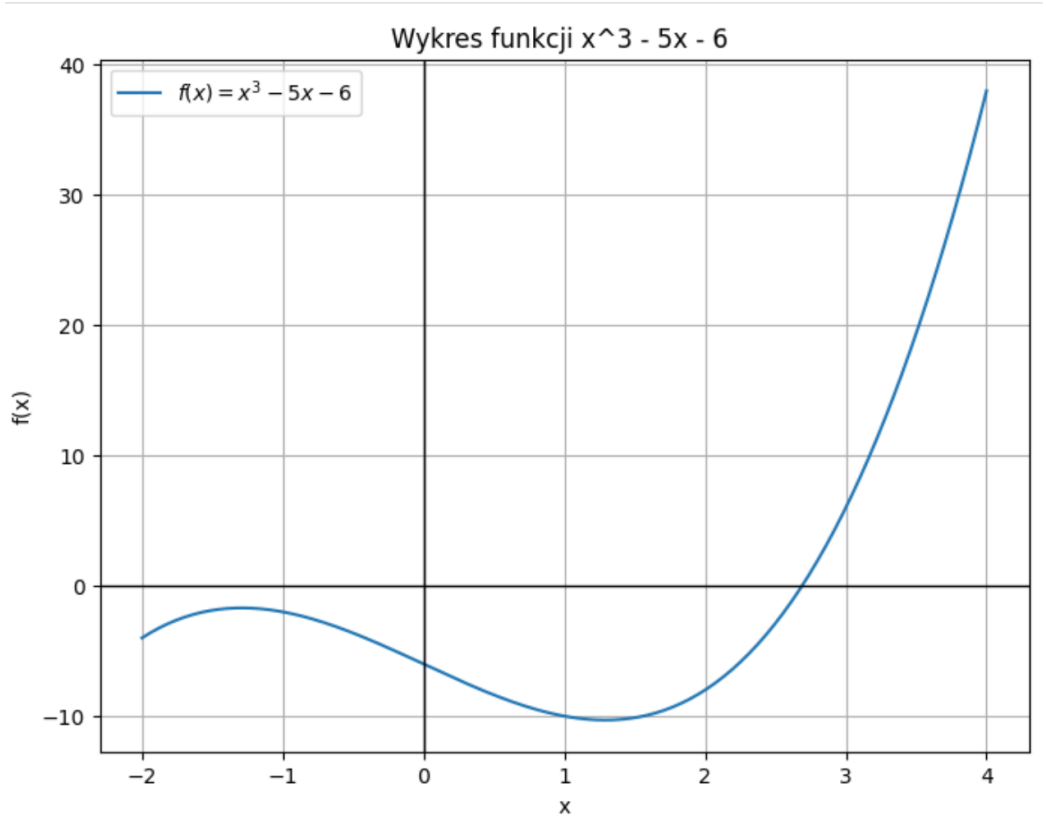
```
iteracja:  0 :  4.5
iteracja:  1 :  4.965267275304913
iteracja:  2 :  4.917364808523858
iteracja:  3 :  4.91718592834027
iteracja:  4 :  4.917185925287132
Znaleziono rozwiązanie po 4 iteracjach.
[29]:

4.917185925287132
```

Rys. 5 Iteracje

Można zauważyć, że wynik zgadza się z wynikiem WolframAlfa.

b) $x^3 - 5x - 6 = 0$



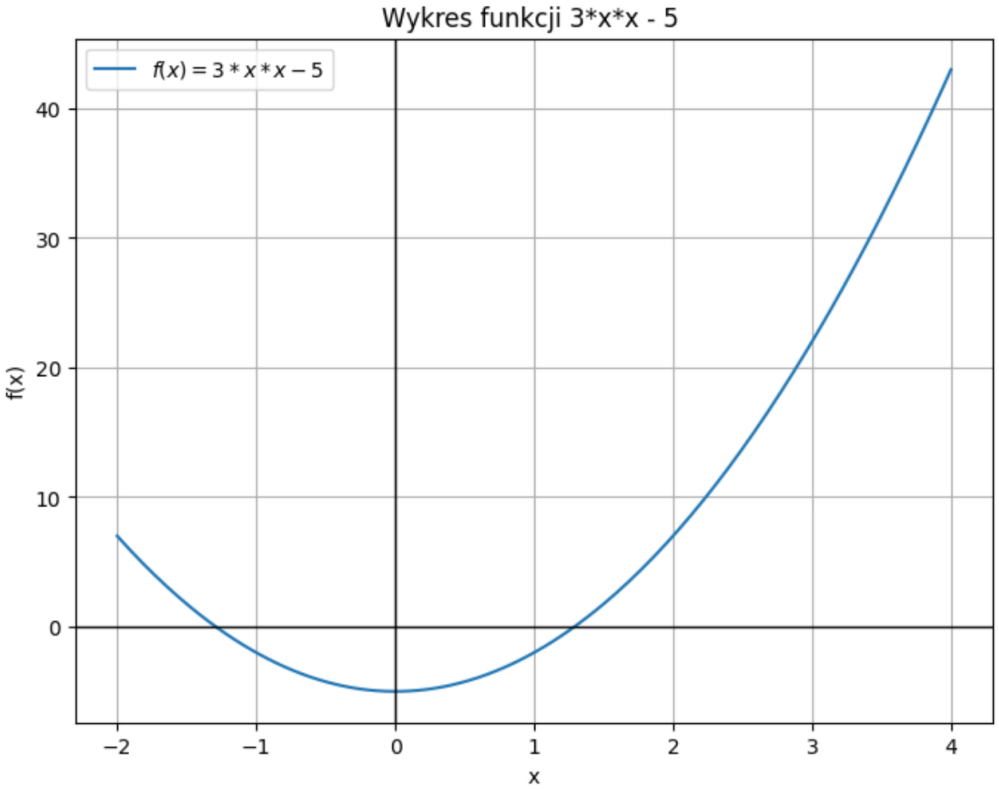

Rys. 6 Wykres funkcji

Miejsce zerowe funkcji znajduje się pomiędzy 2 i 3 i jest to jedyne miejsce zerowe w tym przedziale (spełnione jest więc założenie 1).

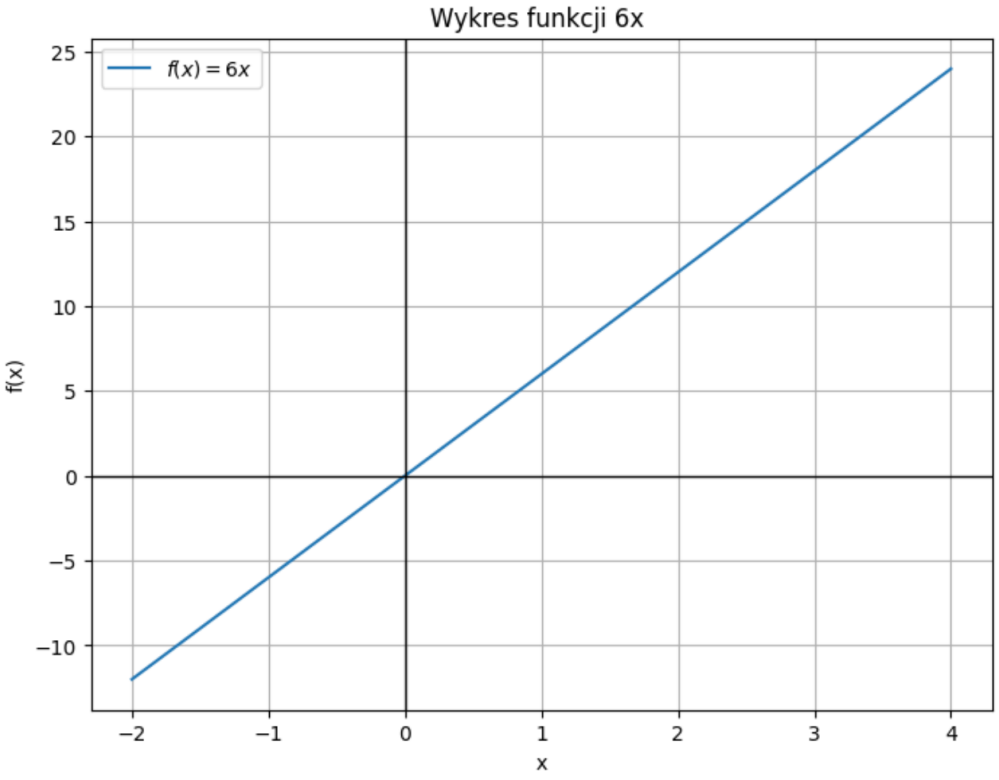
f(2) < 0. Oraz f(3) > 0
Z tego wynika, że spełnione jest założenie 2

Pochodne funkcji:

$$f'(x) = 3x^2 - 5$$
$$f''(x) = 6x$$




Rys. 7 Wykres pochodnej funkcji




Rys. 8 Wykres drugiej pochodnej

W przedziale [2, 3] pierwsza i druga pochodna ma taki sam znak, więc spełnione jest założenie 3.

Funkcja spełnia założenia.
Iteracje algorytmu Newtona:

$$x_{k+1} = x_k - \frac{\mathrm{x} * \mathrm{x} * \mathrm{x} - 5\mathrm{x} - 6}{3x^2 - 5}$$

$$x_0 = 2$$

Według WolframAlpha rozwiązaniem równania xcos(x) – 1 = 0 w przedziale [4.5, 5,5] jest:

$$x \approx 2.6891$$

```
def f(x):
    return x*x*x - 5*x - 6
def df(x):
    return 3*x*x - 5
x0 = 4.5
def newtons_method(f, df, x0, tol=1e-10, max_iter=10):
    xn = x0
    for n in range(0, max_iter):
        fxn = f(xn)
        print("iteracja: ", n, ": ", xn)
        if abs(fxn) < tol:
            print('Znaleziono rozwiązanie po', n, 'iteracjach.')
            return xn
        dfxn = df(xn)
        if dfxn == 0:
            print('Pochodna wynosi zero. Brak rozwiązań.')
            return None
        xn = xn - fxn/dfxn
    print('Przekroczono maksymalną liczbę iteracji. Brak rozwiązań.')
    return None
newton = newtons_method(f, df, x0)
newton
```

```
iteracja:  0 :  4.5
iteracja:  1 :  3.376681614349776
iteracja:  2 :  2.841946354982163
iteracja:  3 :  2.69926807539913
iteracja:  4 :  2.6891449701194072
iteracja:  5 :  2.6890953248287266
iteracja:  6 :  2.6890953236376594
Znaleziono rozwiązanie po 6 iteracjach.
2.6890953236376594
```

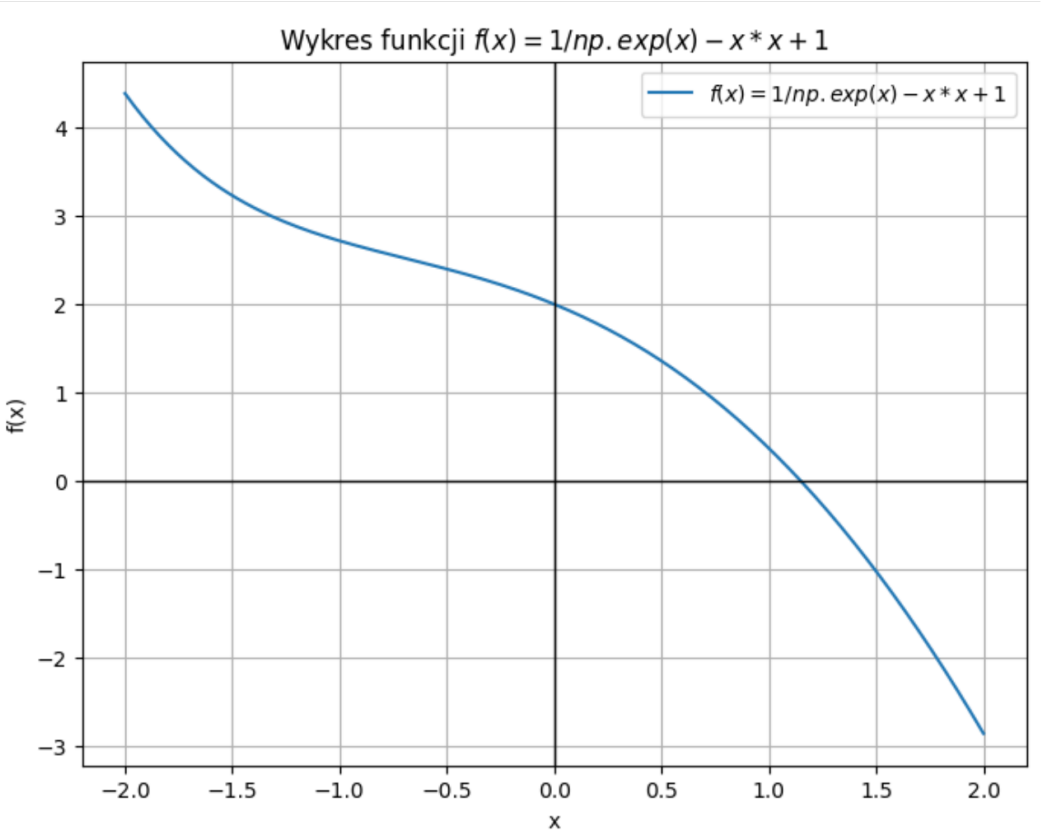
Rys. 9 Iteracje wg metody Newtona

Można zauważyć, że wynik zgadza się z wynikiem WolframAlfa.

c) $e^{-x} = x^2-1$.

Równanie można przekształcić do:

$e^{-x} - x^2 + 1 = 0$



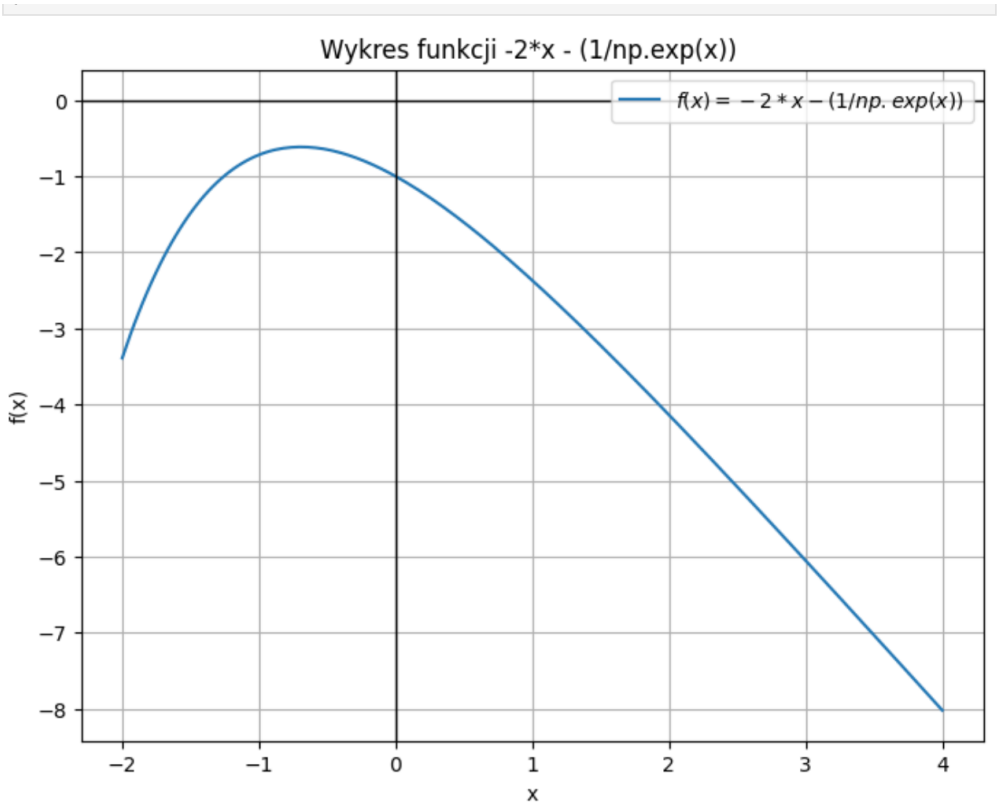

Rys.10 Wykres funkcji

Miejsce zerowe funkcji znajduje się pomiędzy 1 i 1.5 i jest to jedyne miejsce zerowe w tym przedziale (spełnione jest więc założenie 1).

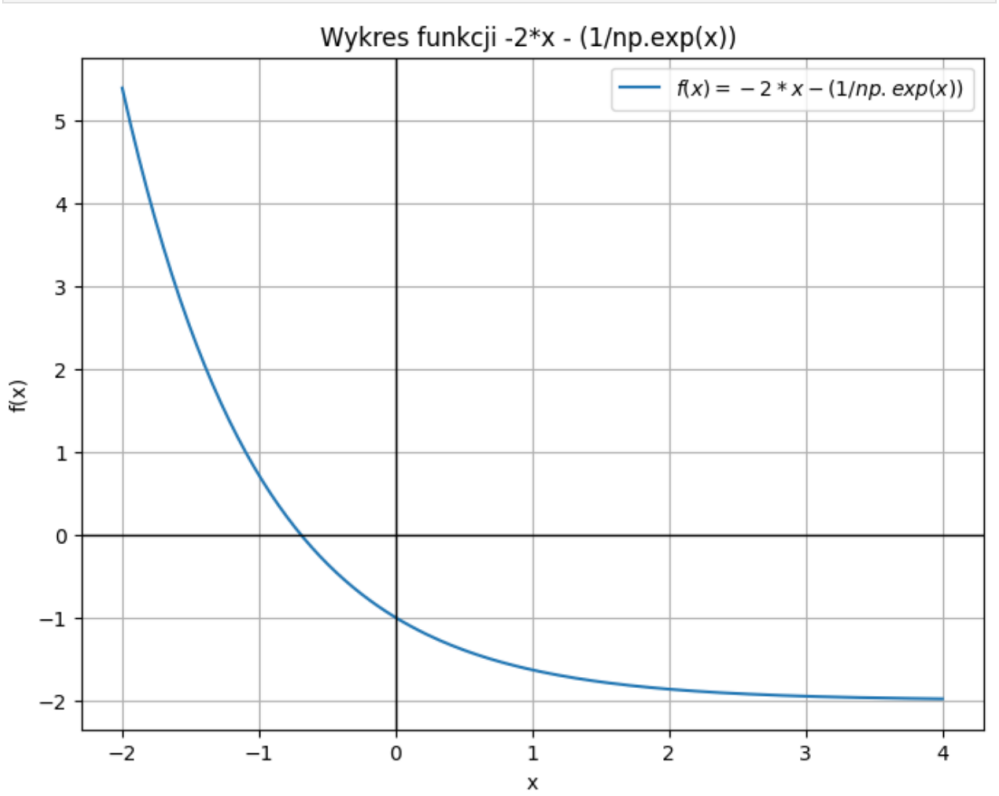
f(1) > 0. Oraz f(1.5) < 0
Z tego wynika, że spełnione jest założenie 2

Pochodne funkcji:

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



Rys. 11 Wykres pochodnej



Rys.12 Wykres drugiej pochodnej

Pierwsza i druga pochodna w przedziale [1, 1.5] mają stały znak. Spełniony jest trzeci warunek.

Funkcja spełnia założenia.
Iteracje algorytmu Newtona:

$$x_{k+1} = x_k - \frac{e^{-x} - x^2 + 1}{-2x - e^{-x}}$$

$$x_0 = 1$$

Według WolframAlpha rozwiązaniem równania w przedziale [1, 1.5] jest:

$$x \approx 1.14776$$

```
def f(x):
    return 1/ np.exp(x) - x*x + 1
def df(x):
    return -2*x - (1/np.exp(x))
x0 = 4.5
def newtons_method(f, df, x0, tol=1e-10, max_iter=10):
    xn = x0
    for n in range(0, max_iter):
        fxn = f(xn)
        print("iteracja: ", n, ": ", xn)
        if abs(fxn) < tol:
            print('Znaleziono rozwiązanie po', n, 'iteracjach.')
            return xn
        dfxn = df(xn)
        if dfxn == 0:
            print('Pochodna wynosi zero. Brak rozwiązań.')
            return None
        xn = xn - fxn/dfxn
    print('Przekroczono maksymalną liczbę iteracji. Brak rozwiązań.')
    return None
newton = newtons_method(f, df, x0)
newton
```

```
iteracja:  0 :  4.5
iteracja:  1 :  2.364980768643163
iteracja:  2 :  1.4322974344880617
iteracja:  3 :  1.1704143169887098
iteracja:  4 :  1.147920998054195
iteracja:  5 :  1.1477576407375276
iteracja:  6 :  1.1477576321447436
Znaleziono rozwiązanie po 6 iteracjach.
1.1477576321447436
```

Rys. 13 Iteracje dla funkcji

Można zauważyć, że wynik zgadza się z wynikiem WolframAlfa.

## 2.2 Zadanie drugie

a) Iteracyjna metoda:

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

Metoda siecznych jest podobna do metody Newtona, ale nie wymaga obliczania pochodnej funkcji. Zamiast tego używa dwóch ostatnich przybliżonych wartości pierwiastka do estymacji nachylenia siecznej, która jest wykorzystywana do znalezienia następnego przybliżenia.

Wzór można przekształcić do postaci:

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

Pierwsza formuła:

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

Druga formuła:

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

Po przekształceniu drugiej formuły otrzymujemy:

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

Co dowodzi, że obie formuły sa równoważne.

Rys. 14 Dowód zrobiony w LaTeXie

Wyrażenie

$$\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}$$

jest przybliżeniem odwrotności pochodnej funkcji f(x) obliczonej w punkcie $\boldsymbol{x_k}$. Jest to średnia zmiana funkcji między dwoma punktami $x_{k-1}$ *i* $x_k$.

W metodzie siecznych zamiast używać rzeczywistej pochodnej funkcji f($x_k$) w punkcie $x_k$ stosujemy różnicę ilorazową, aby oszacować nachylenie siecznej przechodzącej przez punkty $(x_{k-1}, f((x_{k-1}))$ i $(x_k, f((x_k))$. Następnie korzystając z przybliżenia możemy oszacować kolejne przybliżenie $x_{k+1}$.

W metodzie Newtona używamy rzeczywistej wartości pochodnej w danym punkcie, więc w przypadku gdy pochodna jest dostępna i łatwa do obliczenia, metoda Newtona

może być szybsza. Metoda siecznych jest jednak bardziej praktyczna, gdy pochodna jest trudna do obliczenia lub chcemy uniknąć jej obliczania.
Podsumowując, podane równanie jest formułą iteracyjną dla metody siecznych, która jest metoda przybliżenia pierwiastka równania nieliniowego $f(x)=0$ bez konieczności wykorzystania pochodnej.

b) W arytmetyce zmiennoprzecinkowej o skończonej precyzji, zarówno metoda siecznych, jak i metoda przedstawiona w podpunkcie a), mogą wykazywać różne zachowania związane z precyzją i stabilnością numeryczną.

Wariant z podpunktu a:
**Zalety:**
- Unikanie odejmowania podobnych wartości: Wzór jest przekształcony tak, aby uniknąć odejmowania dwóch bliskich sobie wartości co może zmniejszyć ryzyko błędu zaokrąglenia.
- Lepsza stabilność numeryczna: Jeśli wartości $\boldsymbol{x_{k-1}}$i $\boldsymbol{x_k}$są bliskie, a funkcja $f$ zmienia się powoli, wzór może zapewnić lepszą stabilność numeryczną.

**Wady:**
- Potencjalne ryzyko dzielenia przez zero: Jeśli wartości funkcji $f$ dla dwóch kolejnych przybliżeń są identyczne lub bardzo bliskie sobie, może dojść do dzielenia przez bardzo małą wartość, podobnie jak w metodzie siecznych.
- Złożoność: Wzór może być mniej intuicyjny i nieco bardziej skomplikowany do implementacji niż standardowa metoda siecznych.

Wariant z podpunktu b:
**Zalety:**
- Mniej obliczeń: W porównaniu z metodą Newtona, metoda siecznych nie wymaga obliczania pochodnej funkcji.
- Potencjalnie szybsza konwergencja: Dla funkcji, gdzie obliczenie pochodnej jest kosztowne, metoda siecznych może konwergować szybciej niż metoda Newtona.

**Wady:**
- Ryzyko dzielenia przez małe liczby: W przypadku, gdy $\boldsymbol{f(x_k)}$ jest bardzo bliska $\boldsymbol{f(x_{k-1})}$ istnieje ryzyko dzielenia przez małą liczbę, co może spowodować niestabilność numeryczną i błędy zaokrągleń.
- Brak gwarancji zbieżności: Metoda siecznych może być rozbieżna, jeżeli nie jest stosowana z odpowiednią ostrożnością.

### 2.3 Zadanie trzecie

$$\begin{cases} \boldsymbol{x_1^2 + x_1 x_2^3 = 9} \\ \boldsymbol{3x_1^2 x_2 - x_2^3 = 4} \end{cases}$$

$$\begin{cases} \boldsymbol{x_1^2 + x_1 x_2^3 - 9 = 0} \\ \boldsymbol{3x_1^2 x_2 - x_2^3 - 4 = 0} \end{cases}$$

Gdzie

f$(x_1, x_2) = x_1^2 + x_1 x_2^3 - 9$

g$(x_1, x_2)$ = $3x_1^2 x_2 - x_2^3 - 4$

Aby zastosować metodę Newtona do rozwiązania tego układu równań, będziemy potrzebować zastosować ogólny wzór na iterację Newtona dla układów równań nieliniowych:

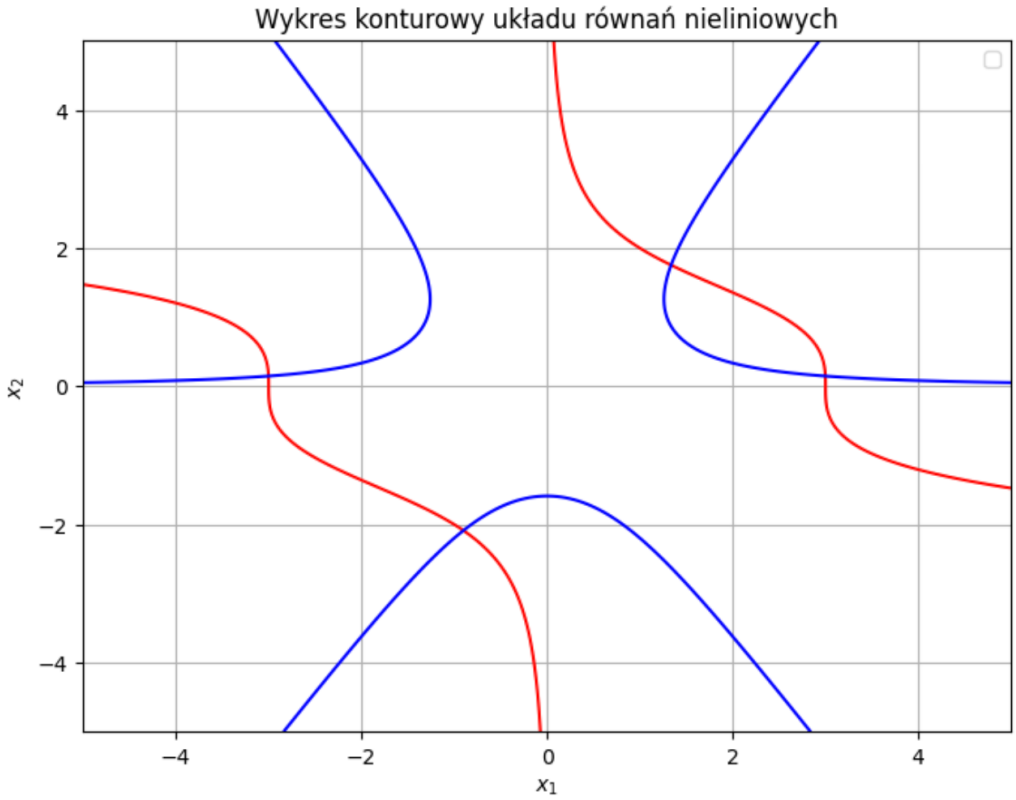
$$\boldsymbol{x_{k+1} = x_k - [J(x_k)]^{-1} f(x_k)}$$

$x_k$ – wektor przybliżeń $[x_{1k}, x_{2k}]^T$ w k-tej iteracji, f$(x_k)$ jest wektorem funkcji, a $J(x_k)$ jest macierzą Jacobiego obliczoną dla wektora $x_k$.

$$\text{J}(x_1, x_2) = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \\ \frac{\partial g}{\partial x_1} & \frac{\partial g}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2x_1 + {x_2}^3 & 3x_1{x_2}^2 \\ 6x_1 x_2 & 3{x_1}^2 - 3{x_2}^2 \end{bmatrix}$$

$$\text{f}(x_k) = \begin{bmatrix} x_1^2 + x_1 x_2^3 - 9 \\ 3x_1^2 x_2 - x_2^3 - 4 \end{bmatrix}$$

$$\boldsymbol{x_{k+1} = x_k - \begin{bmatrix} 2x_1 + {x_2}^3 & 3x_1{x_2}^2 \\ 6x_1 x_2 & 3{x_1}^2 - 3{x_2}^2 \end{bmatrix}^{-1} \begin{bmatrix} x_1^2 + x_1 x_2^3 - 9 \\ 3x_1^2 x_2 - x_2^3 - 4 \end{bmatrix}}$$




Rys. 15 Wykresy funkcji

Na początku przyjmuję

- $(x_1, x_2)$ = (1, 1)
- $(x_1, x_2)$ = (-1, -1)
- $(x_1, x_2)$ = (-4, -1)
- $(x_1, x_2)$ = (4, 1)

```
import numpy as np
from numpy.linalg import inv
def newton(x0, y0, n):
    F = lambda x, y: np.array([[x*x+x*y*y*y-9],[3*x*x*y - y*y*y -4]], dtype=float)
    DF = lambda x, y: np.array([[2*x + y*y*y , 3*x*y*y],
    [6*x*y, 3*x*x - 3*y*y]], dtype=float)
    X = np.array([[x0],[y0]], dtype=float)
    for i in range(1, n):
        x = X[0][0]
        y = X[1][0]
        X = X - inv(DF(x, y)).dot(F(x, y))
        print("iteracja", i, ":", X.transpose())
```

Rys.16 Implementacja

```
newton(1, 1, 10)
```

```
iteracja 1 : [[1.33333333 3.        ]]
iteracja 2 : [[1.27790478 2.24629452]]
iteracja 3 : [[1.30079564 1.8619263 ]]
iteracja 4 : [[1.3325684  1.76019543]]
iteracja 5 : [[1.3363389  1.75425044]]
iteracja 6 : [[1.33635538 1.7542352 ]]
iteracja 7 : [[1.33635538 1.7542352 ]]
iteracja 8 : [[1.33635538 1.7542352 ]]
iteracja 9 : [[1.33635538 1.7542352 ]]
```

Rys.17 Wynik 1

```
newton(-1, -1, 10)
```

```
iteracja 1 : [[ 0.         -4.33333333]]
iteracja 2 : [[-0.11060537 -2.95989481]]
iteracja 3 : [[-0.43432904 -2.15265294]]
iteracja 4 : [[-0.92990055 -2.00441057]]
iteracja 5 : [[-0.89916412 -2.08929941]]
iteracja 6 : [[-0.90126264 -2.08658845]]
iteracja 7 : [[-0.90126619 -2.08658759]]
iteracja 8 : [[-0.90126619 -2.08658759]]
iteracja 9 : [[-0.90126619 -2.08658759]]
```

Rys.18 Wynik 2

```
newton(-4, -1, 10)
```

```
iteracja 1 : [[-5.          0.66666667]]
iteracja 2 : [[-3.37978516  0.48613173]]
iteracja 3 : [[-2.98102324  0.2294787 ]]
iteracja 4 : [[-2.99963637  0.14714002]]
iteracja 5 : [[-3.00162536  0.14810928]]
iteracja 6 : [[-3.00162489  0.14810799]]
iteracja 7 : [[-3.00162489  0.14810799]]
iteracja 8 : [[-3.00162489  0.14810799]]
iteracja 9 : [[-3.00162489  0.14810799]]
```

Rys.19 Wynik 3

```
newton(4, 1, 10)
```

```
iteracja 1 : [[ 4.17948718 -0.05128205]]
iteracja 2 : [[3.16607644 0.05147348]]
iteracja 3 : [[3.00394595 0.13831283]]
iteracja 4 : [[2.99839288 0.14838967]]
iteracja 5 : [[2.99836535 0.14843098]]
iteracja 6 : [[2.99836535 0.14843098]]
iteracja 7 : [[2.99836535 0.14843098]]
iteracja 8 : [[2.99836535 0.14843098]]
iteracja 9 : [[2.99836535 0.14843098]]
```

Rys.20 Wynik 4

Można zauważyć, że wyniki dążą do prawdziwego miejsca zerowego, czyli:

- $(x_1, x_2)$ = (1.33635538, 1.7542352)
- $(x_1, x_2)$ = (-0.90126619, -2.08658759)
- $(x_1, x_2)$ = (-3.00162489, 0.14810799)
- $(x_1, x_2)$ = (2.99836535, 0.14843098)

## 3. Bibliografia


- https://pl.wikipedia.org/wiki/Metoda_Newtona
- https://home.agh.edu.pl/~funika/mownit/lab7/RF.pdf