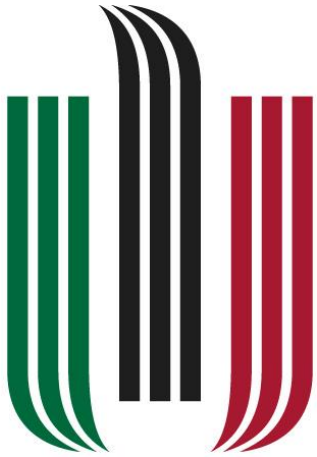
Laboratorium VII

Całkowanie numeryczne

Dominik Marek

23 kwietnia 2024



AGH

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

1.Zadania

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

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?

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

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

Korzystając z metody Newtona przyjmuje się następujące założenia dla funkcji f:

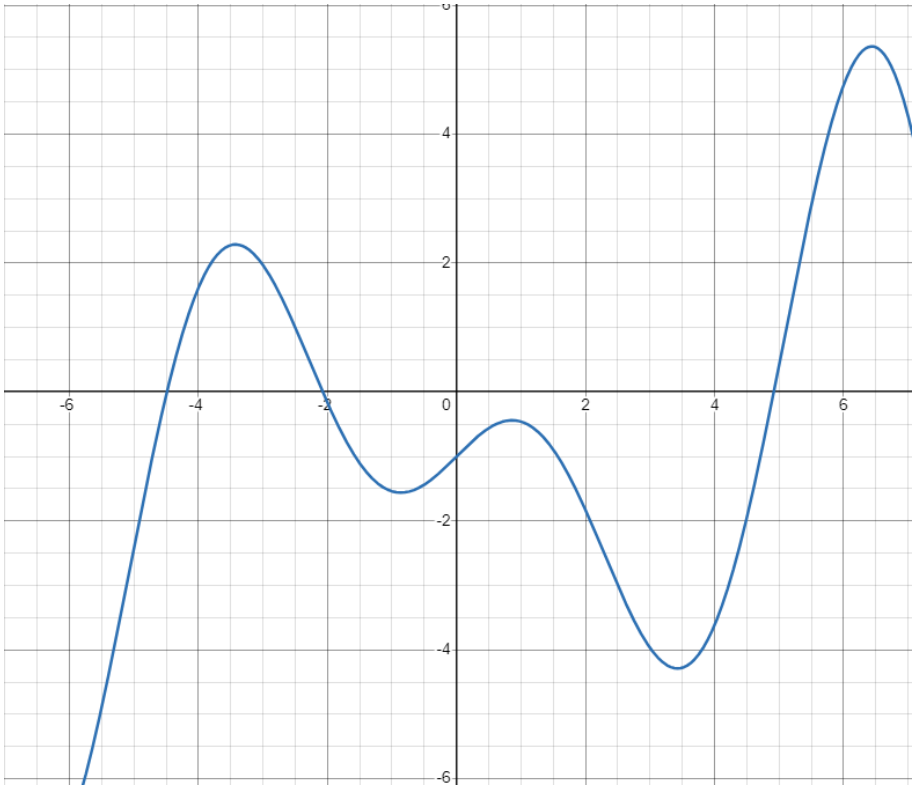
1. W przedziale [a, b] znajduje się dokładnie jeden pierwiastek.
2. Funkcja ma różne znaki na krańcach przedziału, tj. $f(a) \cdot f(b) < 0$.
3. Pierwsza i druga pochodna funkcji mają stały znak w tym przedziale.

Kolejne przybliżenia szukanego miejsca zerowego dane są następującym wzorem rekurencyjnym;

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, k = 1, 2, 3, \dots$$

a) $f(x) = x\,cos(x) - 1$

Wykres funkcji ma poniższą postać:



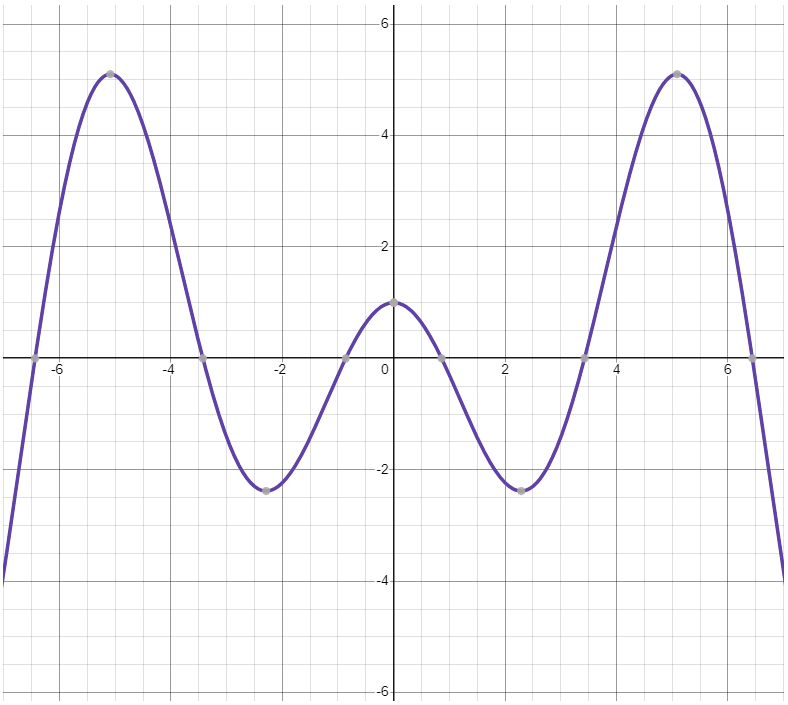
Rysunek 1: Wykres funkcji f(x) = xcos(x)-1

Można zauważyć, że miejsce zerowe znajduje się w przedziale od 4 do 6 i jest to zarazem jedyne miejsce zerowe w tym przedziale, więc pierwsze założenie metody Newton jest spełnione.

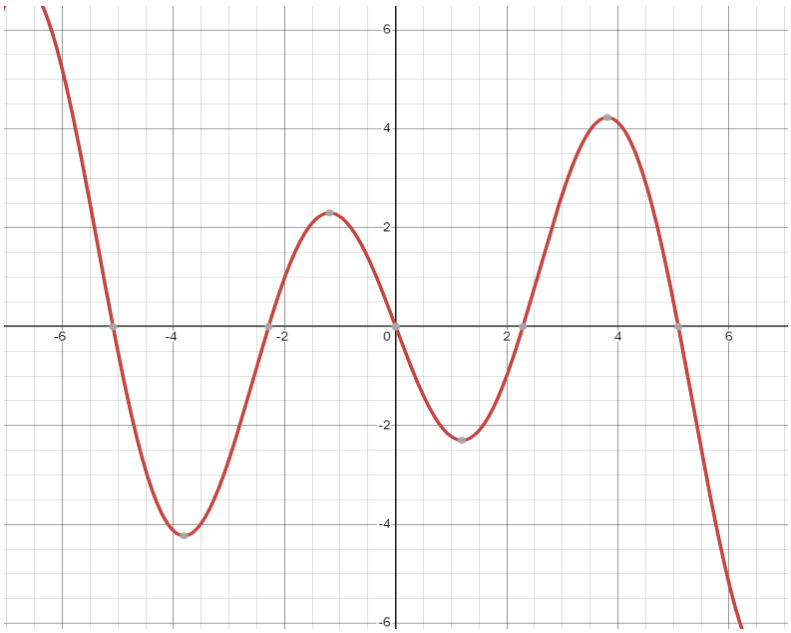
Obliczamy pierwszą i drugą pochodną funkcji;

$$f'(x) = \frac{d}{dx}(x\,cos(x) - 1) = cos\,x - x\,sin\,x$$

$$f''(x) = \frac{d}{dx}(cos\,x - x\,sin\,x) = -sinx - sinx - xcosx = -xcosx - 2sinx$$



Rysunek 2: Wykres pierwszej pochodnej f'(x) = cos(x) -xsin(x)



Rysunek 3:Wykres drugiej pochodnej: f(x) = -xcos(x) - 2sin(x)

Można zauważyć iż, f'' nie ma stałego znaku na przedziale (4, 6), więc tym samym trzecie założenie nie jest spełnione . Jednak po mimo to zapiszemy iterację algorytmu Newtona i obliczmy kilka pierwszych z nich.

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = x_k - \frac{\mathrm{xcos}(x) - 1}{\cos(x) - \sin(x)}$$

Numer iteracji	Wartość aproksymacji miejsca zerowego
1	5.522845345218782
2	4.860294809581786
3	4.917565305315841
4	4.9171859390089105
5	4.917185925287132
6	4.917185925287132
7	4.917185925287132
8	4.917185925287132
9	4.917185925287132
10	4.917185925287132

Analizując wyniki można zauważyć, że już dla 5 iteracji otrzymuje dobre przybliżenie miejsca zerowego zgodne z wykresem funkcji.

b) $f(x) = x^3 - 5x - 6$

Wykres funkcji ma następujacy przebieg:

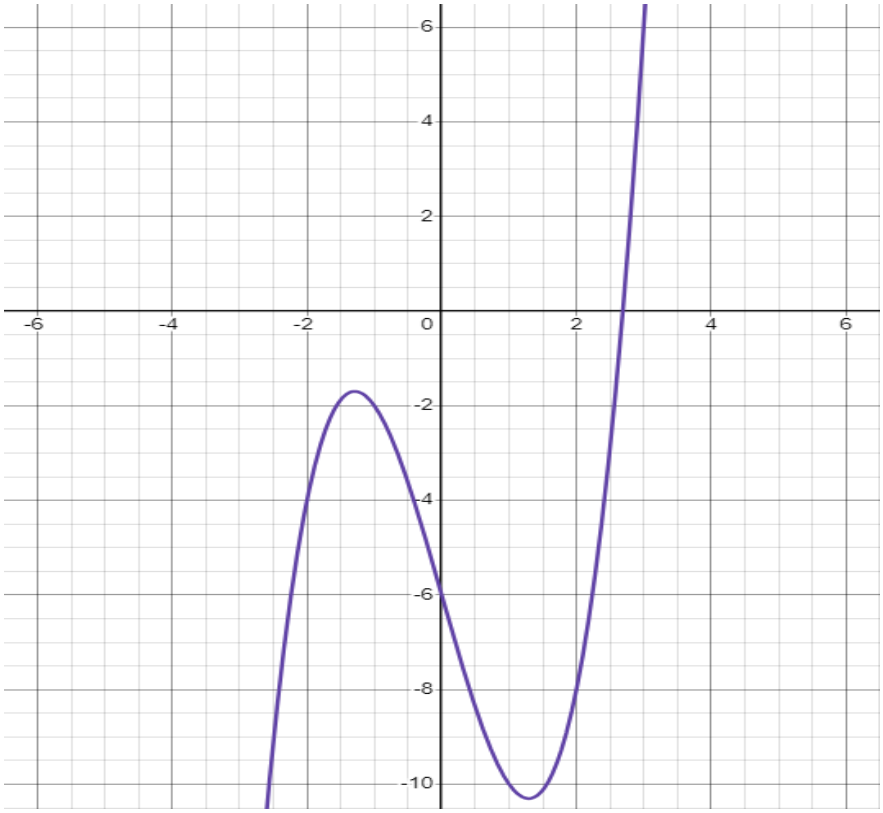


Figura 1:Wykres funkcji f(x)= x^3-5x-6

Przeanalizowawszy powyższy wykres możemy zauważyć, iż analizowana funkcja ma miejsce zerowe w przedziale $(2, 4)$ i jest to jedyne miejsce zerowe w tym przedziale.

Obliczamy pierwszą oraz drugą pochodną funkcji:

$$f'(x) = \frac{d}{dx}(x^3 - 5x - 6) = 3x^2 - 5$$

$$f''(x) = \frac{d}{dx}(3x^2 - 5) = 6x$$

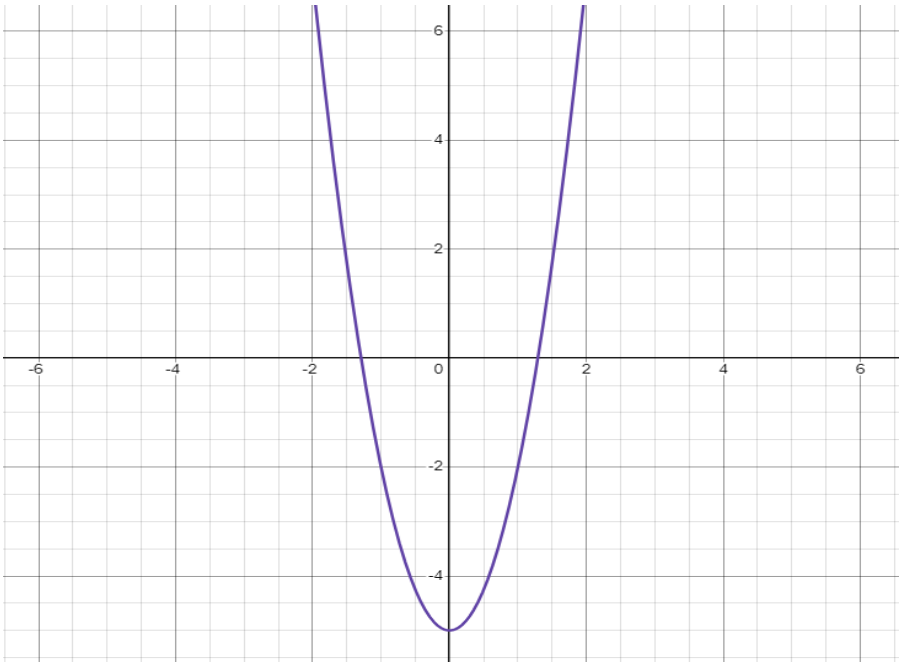


Figura 2:Wykres pierwszej pochodnej f(x) = $3x^2$ -5

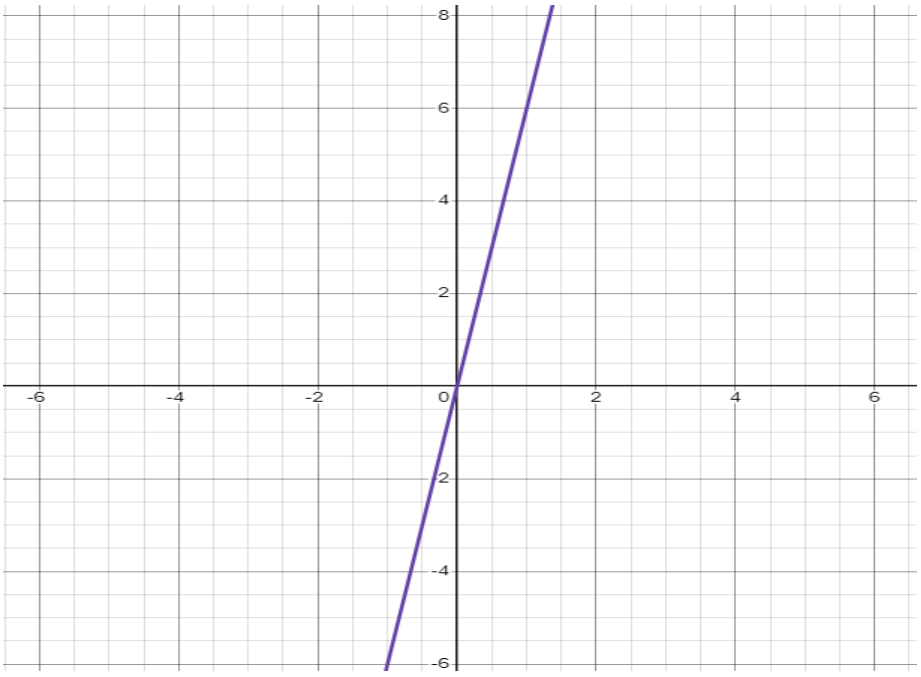


Figura 3:Wykres drugiej pochodnej f(x) = 6x

Jak można zauważyć z wykresów pierwszych dwóch pochodnych badanej funkcji obie funkcje nie zmieniają znaku w intersującym nas przedziale$(4, 6)$. Tym samym trzecie założenie metody iteracyjnej Newtona jest spełnione. Zatem obliczamy postać rekurencyjną wzoru:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = x_k - \frac{x^3 - 5x - 6}{3x^2 - 5}$$

Iteracja z wykorzystaniem powyższego wzoru daj następujące wyniki:

Numer iteracji	Wartość aproksymacji miejsca zerowego
1	3.142857142857143
2	2.7641140963427624
3	2.6916758513953347
4	2.6916758513953347
5	2.6916758513953347
6	2.6890953236376594
7	2.6890953236376594
8	2.6890953236376594
9	2.6890953236376594
10	2.6890953236376594

Podobnie jak we wcześniejszym przykładzie już od 6 iteracji otrzymujemy dobre przybliżenie miejsca zerowego funkcji, pokrywające z przewidywanym miejsce zerowym na podstawie wykresu.

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

Poniższy wykres przedstawia przebieg funkcji:

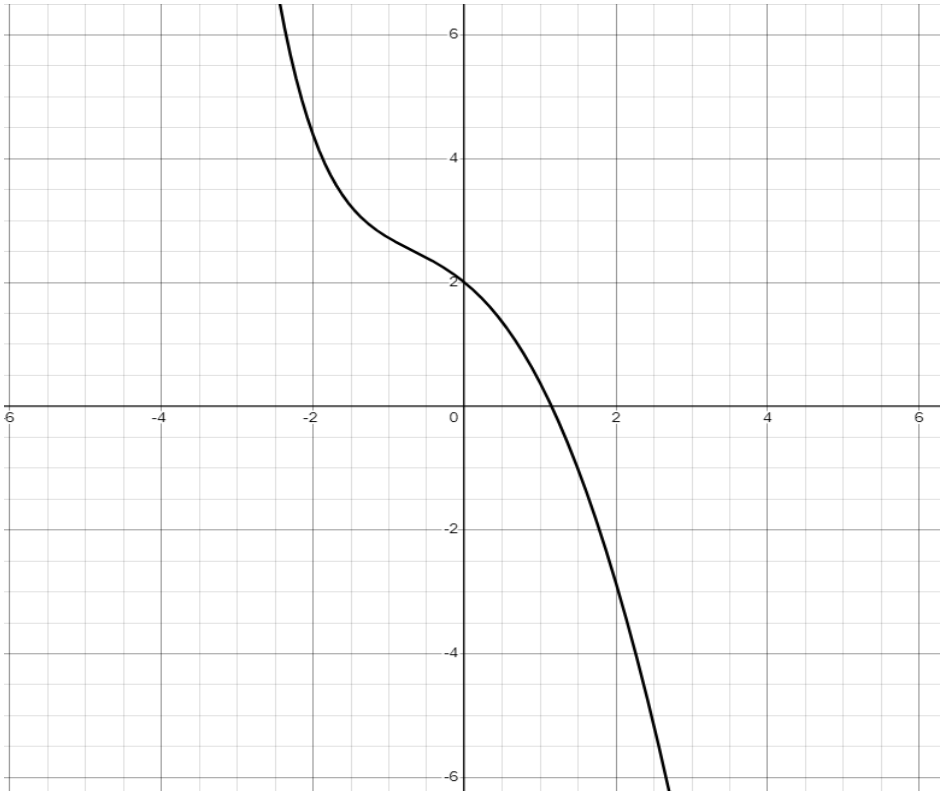


Figura 4: Wykres funkcji $f(x) = e^{-x} - x^2 + 1$

Jedyne miejsce zerowe funkcji znajduje się w przedziale $(0, 2)$.

Dwie pierwsze pochodne mają poniższą postać:

$$f'(x) = \frac{d}{dx}(e^{-x} - x^2 + 1) = -e^{-x} - 2x$$

$$f''(x) = \frac{d}{dx}(-e^{-x} - 2x) = e^{-x} - 2$$

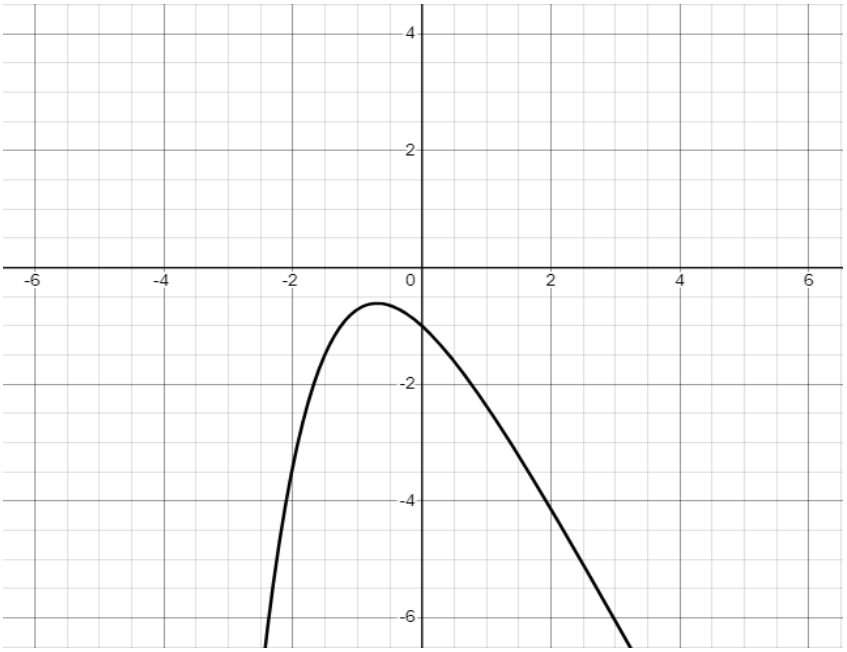


Figura 5: Wykres pierwszej pochodnej f(x) = -e^{-x} -2x

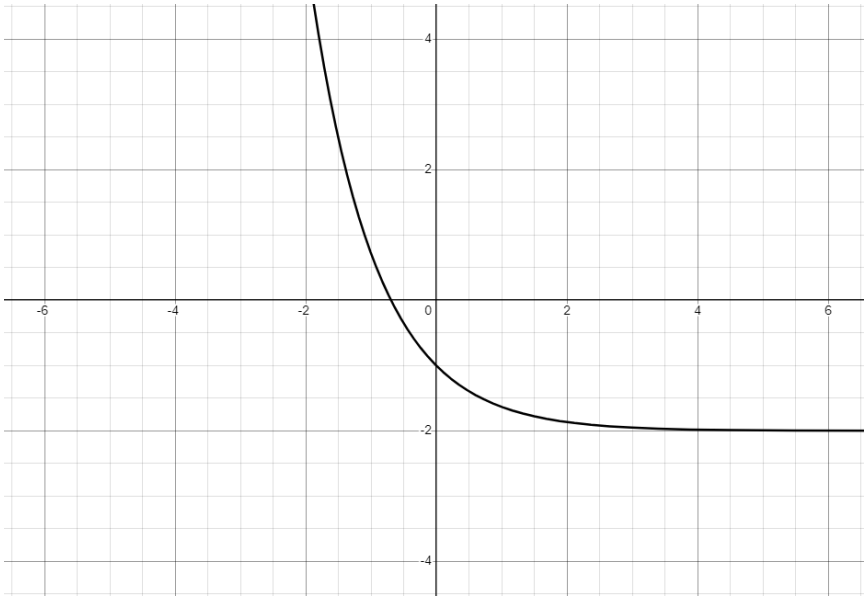


Figura 6: Wykres drugiej pochodnej f(x) = e^{-x} -2

Obie pochodne w interesującym nas przedziale są stałego znaku, tym samym spełniając trzecie założenie metody iteracyjnej Newtona. Przystępujemy zatem do wyliczenia postaci rekurencyjnej:

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

Wykonawszy dziesięć pierwszych iteracji dostajemy poniższe wyniki:

Numer iteracji	Wartość aproksymacji miejsca zerowego
1	3.142857142857143
2	2.7641140963427624
3	2.6916758513953347
4	2.6916758513953347
5	2.6916758513953347
6	2.6890953236376594
7	2.6890953236376594
8	2.6890953236376594
9	2.6890953236376594
10	2.6890953236376594

Również tym razem już dla szóstej iteracji dostajemy dobre przybliżenie miejsca zerowego badanej funkcji, które pokrywa się z przypuszczalnym miejscem zerowym z wykresu.

Obliczenia dla 3 powyższych podpunktów zostały wykonane z pomocą poniższego kodu napisanego w Pythonie:

```
from numpy import cos, sin, e
from typing import Callable


def f1(x:float) -> float:
    return x*cos(x)-1


def f2(x:float) -> float:
    return x**3 - 5*x - 6


def f3(x:float) -> float:
    return e**(-x) -x**2 + 1


def df1(x:float) -> float:
    return cos(x) - x*sin(x)


def df2(x:float) -> float:
    return 3*x**2 - 5


def df3(x:float) -> float:
    return -e**(-x) - 2*x


def recursive_newton_method(fun: Callable, df: Callable,  x0: int, n: int)
-> None:
    root_approximation = x0
```

```
    for i in range(n):
        root_approximation -=
fun(root_approximation)/df(root_approximation)
        print(f'Approximation number {i}: {root_approximation = }')


def main() -> None:
    print("----------------------------Function_1----------------------------
")
    recursive_newton_method(f1, df1, 4, 10)
    print("----------------------------Function_2----------------------------
")
    recursive_newton_method(f2, df2, 2, 10)
    print("----------------------------Function_3----------------------------
")
    recursive_newton_method(f3, df3, 0, 10)


if __name__ == '__main__':
    main()
```

2.

a)

Przekształcając równoważnie poniższy wzór rekurencyjny dla metody siecznych pokażemy , iż jest on równoważny wzorowi pojawiającemu się w zadaniu.

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

Zatem wychodząc od poniższej postaci:

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

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

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

Finalnie otrzymujemy postać równoważna ze wzorem zadanym w treści zadania.

b)

Rozważmy sytuację gdy x_{k-1} oraz x_k są bliskie rzeczywistej wartości rozwiązania, z zatem są one również bliskie względem siebie, co z kolei implikuje, że ich różnica jest bliska zeru. Podobnie wygląda sytuacja z $f(x_k)$ oraz $f(x_{k-1})$, których różnica również będzie bliska zeru. Zatem może dojść do sytuacji gdy w przypadku metody siecznych otrzymamy niezdefiniowane wyrażenie postaci : $\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}$.

Jednakże, skoro jesteśmy blisko pierwiastka badanej funkcji to $f(x_k)$ przyjmie wartość zbliżona do zera, więc wyzeruje część wyrażenie, w której się znajduje. Efektem tego, będzie , że otrzymamy równanie rekurencyjne postaci $x_{k_{+1}} = x_k$. Tak więc, gdy w razem ze wzrostem nieokreśloności części naszego wyrażenia, w skończonej precyzji otrzymujemy stała wartość naszego przybliżenia równą $x_{k_{+1}}$. Jednakże wspomniana nieokreśloność jest jedynym aspektem wpływającym na dokładność obliczeń w naszym problemie.

3.

Wstępny układ równań przekształcamy do postaci:

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

Używając wzorów przedstawionych na wykładzie napisałem poniższy program w Pythonie obliczający x_1 oraz x_2 :

```
def f1(x:float, y:float) -> float:
    return x**2 + x*y**3 - 9

def f2(x:float, y: float) -> float:
    return 3*x**2*y - y**3 - 4


def df1x(x:float, y:float) -> float:
    return 2*x + y**3


def df1y(x:float, y:float) -> float:
    return 3*x*y**2


def df2x(x:float, y:float) -> float:
    return 6*x*y


def df2y(x:float, y:float) -> float:
    return 3*x**2 - 3*y**2


def Jacob(x:float, y:float) -> float:
    return df1x(x, y) * df2y(x, y) - df1y(x, y) * df2x(x, y)


def Newton(x0:float, y0:float, n: int) -> None:
    x1 = x0
    x2 = y0

    for i in range(1, n+1):
        delta_x1 = (f2(x1, x2)*df1y(x1, x2) - f1(x1, x2)*df2y(x1,
x2))/Jacob(x1, x2)
        delta_x2 = (f1(x1, x2) * df2x(x1, x2) - f2(x1, x2) * df1x(x1, x2))
/ Jacob(x1, x2)

        x1 += delta_x1
```

```
        x2 += delta_x2

        print(f'Iteration number {i}: {x1 = } {x2 = }')


def main() -> None:
    Newton(0, 2, 10)


if __name__ == '__main__':
    main()
```

Dla dziesięciu iteracji oraz przedziału (0, 2) otrzymujemy następujące wyniki:

Numer iteracji	Wartość aproksymacji x_{1_i}	Wartość aproksymacji x_{2_i}
1	1.125	1.0
2	1.0652568195008705	3.0158638034436063
3	1.148297766081981	2.196713429883737
4	1.2800036673414303	1.8243761677605692
5	1.3340307751054068	1.7557450693263783
6	1.3363544653829675	1.7542349021984458
7	1.3363553772176073	1.7542351976517294
8	1.336355377217167	1.7542351976516988
9	1.336355377217167	1.7542351976516988
10	1.336355377217167	1.7542351976516988

3 Bibliografia

- Katarzyna Rycerz: Wykład z przedmiotu Metody Obliczeniowe
- w Nauce i Technice
- https://en.wikipedia.org/wiki/Newton%27s_method
- https://www.wolframalpha.com/
- Piotr Fronczak Metody numeryczne