

4. Rešavanje nelinearnih jednačina

Data je funkcija:

$$f(x) = \sin x$$

1. Nacrtati funkciju na intervalu $x \in \left[\frac{\pi}{3}, \frac{4\pi}{3}\right]$. Prikazivanje grafika (plt.show()) uvek postaviti nakon crtanja celog grafika:

```
import numpy as np
from matplotlib import pyplot as plt

f = lambda x: np.sin(x)
x = np.linspace(np.pi/3, 4*np.pi/3, 100)
fX = f(x)
plt.plot(x, fX, 'b')

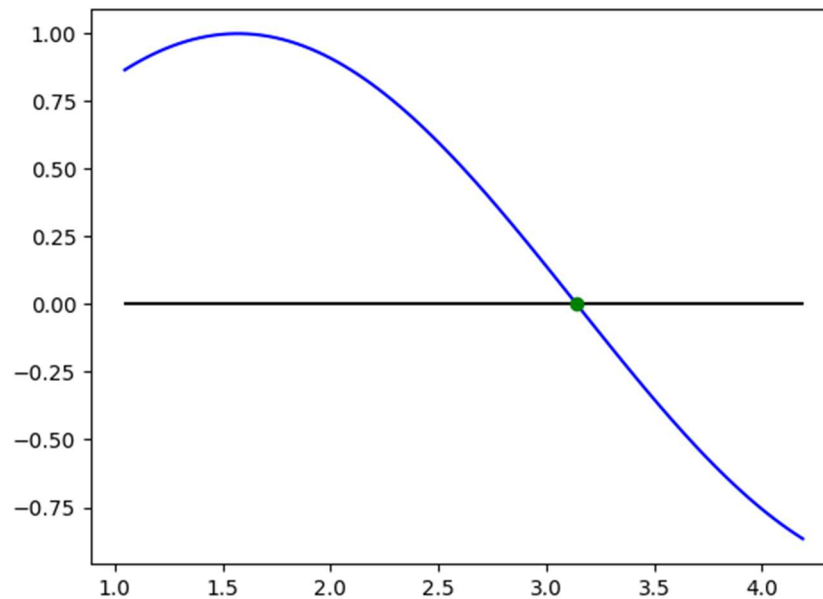
plt.show()
```

2. Nacrtati x -osu na intervalu $x \in \left[\frac{\pi}{3}, \frac{4\pi}{3}\right]$:

```
plt.plot([np.pi/3, 4*np.pi/3], [0, 0], 'k')
```

3. Nacrtati nulu funkcije (isključiti zadržavanje grafika):

```
zero = np.pi
fZero = f(zero)
plt.plot(zero, fZero, 'go')
```



Slika 1. Nula funkcije

1. Metoda polovljenja

Zadatak 1. Napisati metodu polovljenja za traženje nule nelinearnih funkcija.

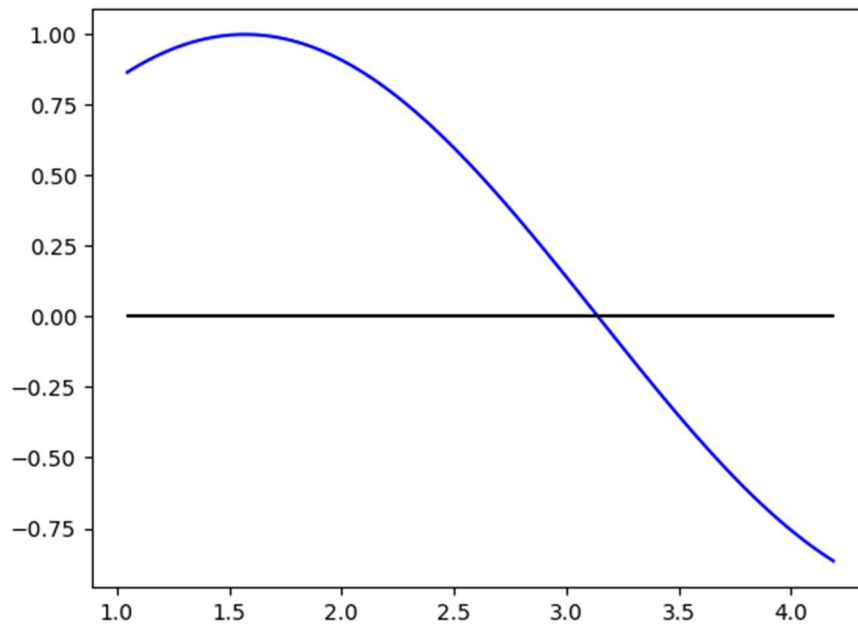
Pokušati prvo ručno jednu iteraciju metode polovljenja nad funkcijom $f(x) = \sin x$ na intervalu $x \in \left[\frac{\pi}{3}, \frac{4\pi}{3}\right]$:

```
f = lambda x: np.sin(x)
a = np.pi/3
b = 4*np.pi/3
```

1. Nacrtati funkciju nad intervalom i naći njen minimum i maksimum i zadržati grafik:

```
x = np.linspace(np.pi/3, 4*np.pi/3, 100)
fX = f(x)
fMin = np.min(fX)
fMax = np.max(fX)
plt.plot(x, fX, 'b', [a, b], [0, 0], 'k')
```

Rezultat:

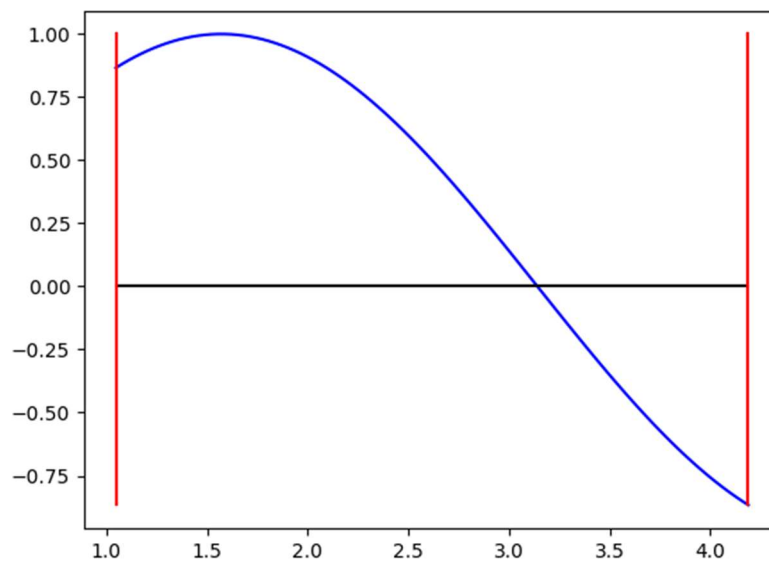


Slika 2. Grafik funkcije

2. Nacrtati 2 vertikalne ose na početku i na kraju intervala crvenom bojom:

```
plt.plot([a, a], [fMin, fMax], 'r', [b, b], [fMin, fMax], 'r')
```

Rezultat:



Slika 3. Interval

3. Pretpostaviti nulu funkcije na polovini intervala i izračunati vrednost funkcije u pretpostavljenoj nuli:

```
zero = (a + b) / 2;  
fZero = f(zero)
```

Rezultat:

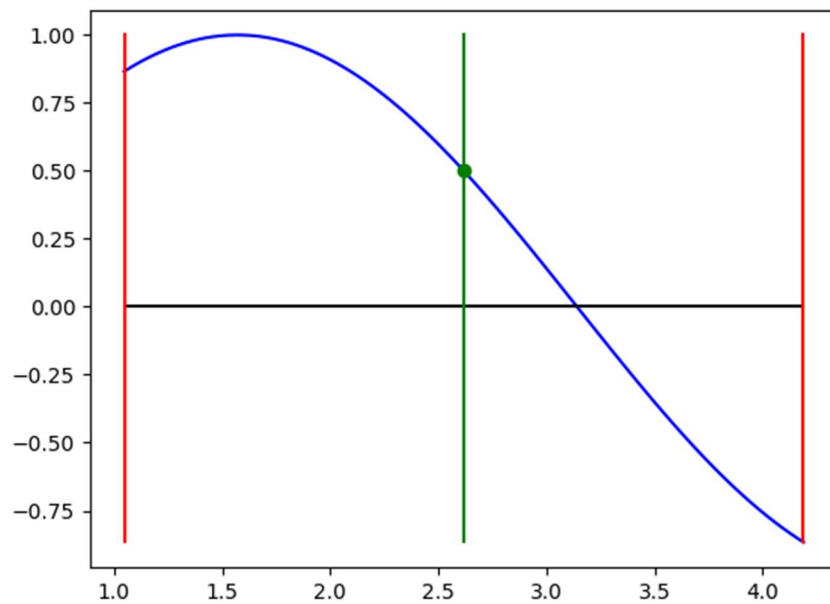
fZero = 0.5000

Izračunata vrednost je različita od 0.

4. Nacrtati vertikalnu osu i tačku određenu izračunatom vrednošću u pretpostavljenoj nuli funkcije zelenom bojom:

```
plt.plot([zero, zero], [fMin, fMax], 'g', zero, fZero, 'go')
```

Rezultat:



Slika 4. Pretpostavljena nula

5. Na osnovu izračunate vrednosti funkcije u jednom od krajeva intervala, pripremiti podinterval za narednu iteraciju. Odabrati onaj od dva podintervala($[a, zero]$ ili $[zero, b]$) na čijim krajevima vrednost funkcije ima različit znak:

```
if f(a)*fZero < 0:  
    b = zero  
else:  
    a = zero
```

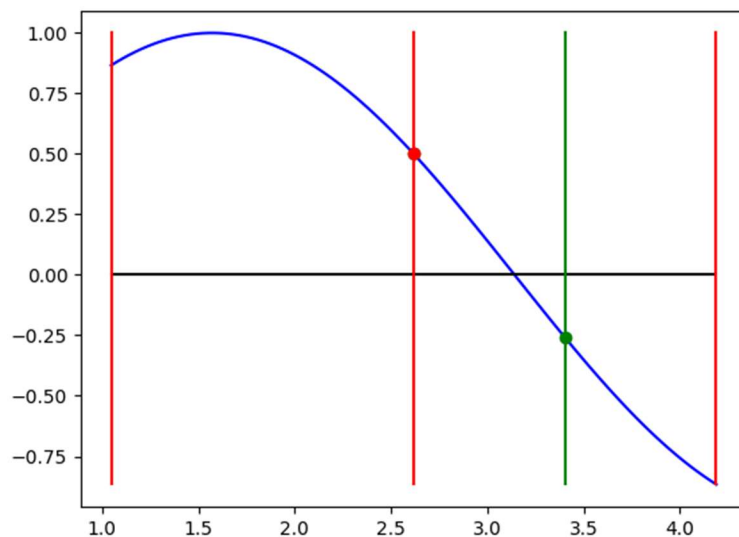
6. Prekriti zelenu vertikalnu osu i tačku iz prethodne iteracije crvenom bojom:

```
plt.plot([zero, zero], [fMin, fMax], 'r', zero, fZero, 'ro')
```

Ako se prethodni postupak (od tačke 3) ponavlja:

nakon 1. ponavljanja:

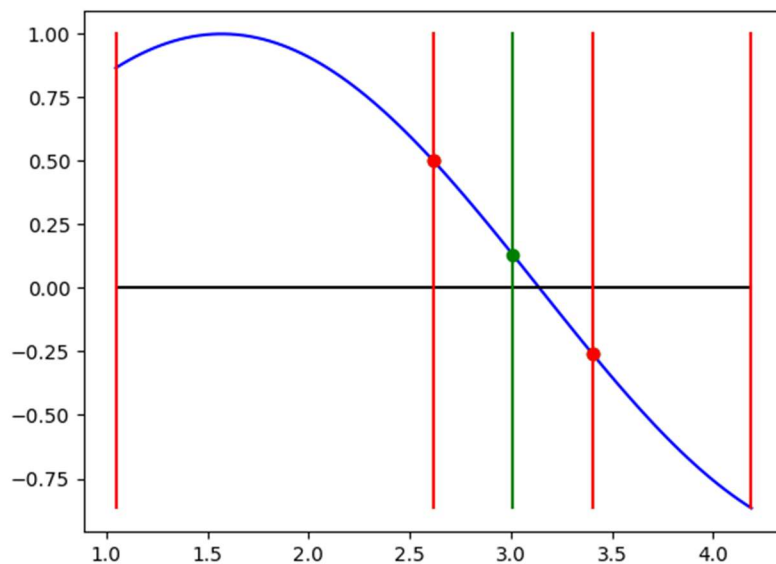
fZero = -0.2588



Slika 5. Nakon 1. ponavljanja

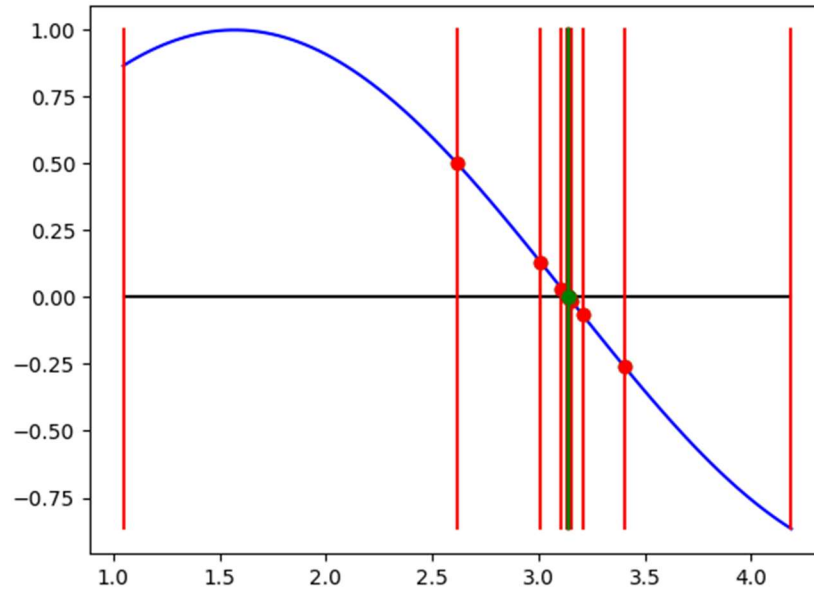
nakon 2. ponavljanja:

fZero = .1305



Slika 6. Nakon 2. ponavljanja

nakon 16. ponavljanja:
fZero = 7.9895e-06



Slika 7. Nakon 16. ponavljanja

7. Prethodni postupak (od tačke 3) se može ugnjeziditi u beskonačnu *while* petlju. Usput je poželjno **brojati iteracije**:

```
it = 0
while True:
    zero = (a + b)/2
    fZero = f(zero)

    plt.plot([zero, zero], [fMin, fMax], 'g', zero, fZero, 'go')

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

    it += 1
    plt.plot([zero, zero], [fMin, fMax], 'r', zero, fZero, 'ro')
```

8. Potrebno je definisati uslov za prekid iteracije u slučaju da **vrednost funkcije u pretpostavljenoj nuli padne ispod tražene preciznosti** ili u slučaju da **dužina podintervala padne ispod tražene preciznosti**:

```
if np.abs(fZero) < errMax or np.abs(b - a) < errMax:
    break
```

Sada je moguće definisati funkciju koja sadrži prethodni algoritam. Na početku funkcije zatvoriti eventualno postojeći prethodni grafik:

```
def zeroBisection(f, a, b, errMax=0.0001, plotSpeed=1):  
    .  
    .  
    .  
    plt.show()  
    return zero, it
```

9. Pauzirati algoritam u zavisnosti od tražene brzine iscrtavana postupka:

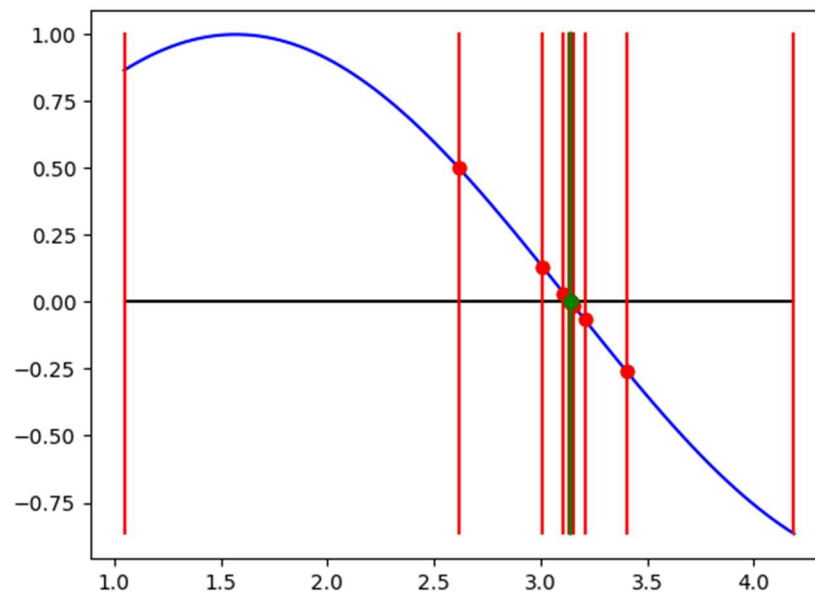
```
.  
. .  
. .  
if np.abs(fZero) < errMax or np.abs(b - a) < errMax:  
    break  
plt.pause(1/plotSpeed)  
.  
. .  
. .
```

10. Testirati funkciju zeroBisection na primeru:

```
f = lambda x: np.sin(x)  
a = np.pi/3  
b = 4*np.pi/3  
  
zero, it = zeroBisection(f, a, b, 0.0001, 10)  
print(zero, it)
```

Rezultat:

```
zero = 3.1416  
it = 13
```



Slika 8. Nula funkcije

11. Napraviti 2 podfunkcije u funkciji zeroBisection. Ako je prosleđena brzina iscrtavanja postupka 0 ili manja, pozvati varijantu funkcije bez naredbi za iscrtavanje i pauziranje algoritma:

```
def zeroBisection(f, a, b, errMax=0.001, itMax=100, plotSpeed=-1):
    if f(a)*f(b) > 0:
        raise Exception('Invalid parameters: f(a)*f(b) > 0!')

    if plotSpeed <= 0:
        return zeroBisectionNoPlot(f, a, b, errMax, itMax)

    return zeroBisectionPlot(f, a, b, errMax, itMax, plotSpeed)
```

12. Testirati funkciju zeroBisection na primeru:

```
f = lambda x: np.sin(x)
a = np.pi/3
b = 4*np.pi/3

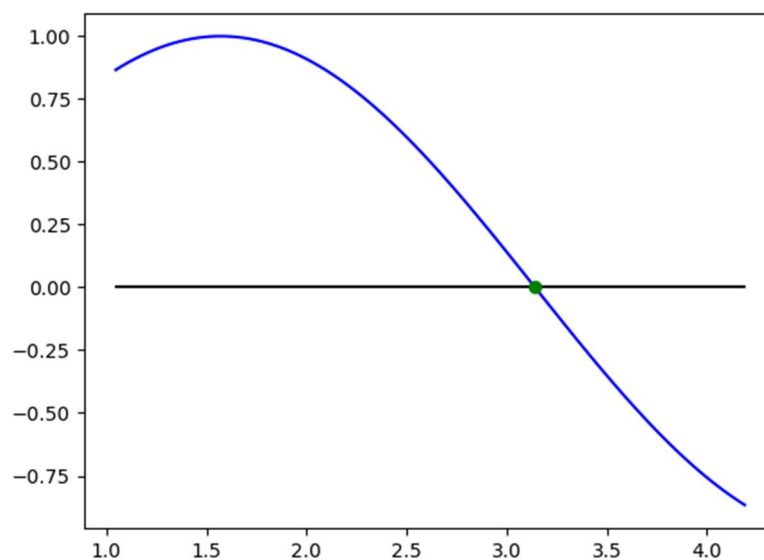
x = np.linspace(a, b, 100)
fX = f(x)

plt.plot(x, fX, 'b')
plt.plot([a, b], [0, 0], 'k')

zero, it = zeroBisection(f, a, b, 0.0001, 100, -1)
fZero = f(zero)

plt.plot(zero, fZero, 'go')
plt.show()
print(zero, it, fZero)

Rezultat:
zero = 3.1416
it = 14
fZero = -6.391586611786031e-05
```



Slika 9. Nula funkcije

2. Metoda sečice

Zadatak 1. Napisati metodu sečice za traženje nule nelinearnih funkcija.

Pokušati prvo ručno jednu iteraciju metode sečice nad funkcijom $f(x) = \sin x$ na intervalu $x \in \left[\frac{\pi}{3}, \frac{4\pi}{3}\right]$:

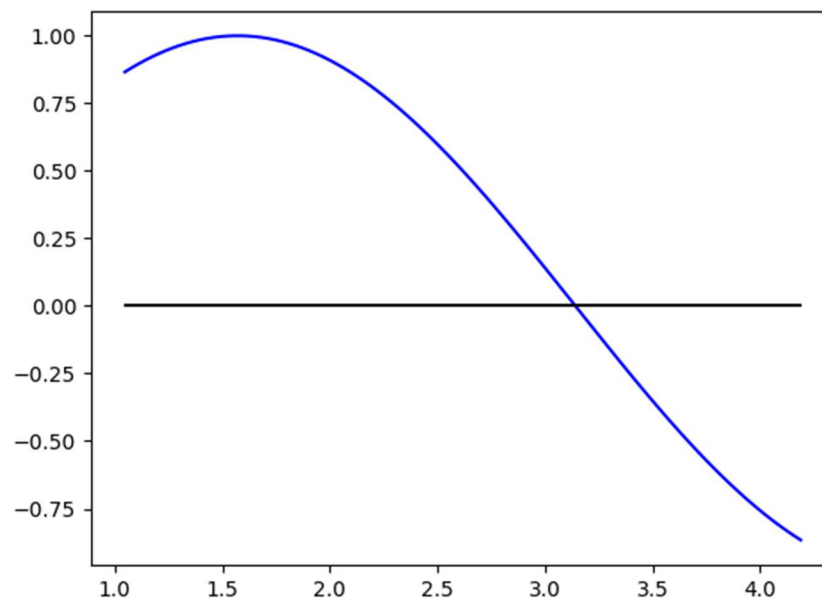
```
f = lambda x: np.sin(x)
a = np.pi/3
b = 4*np.pi/3

plotA = np.pi/3
plotB = 4*np.pi/3
```

1. Nacrtati funkciju nad intervalom i naći njen minimum i maksimum i zadržati grafik:

```
x = np.linspace(plotA, plotB, 100)
fX = f(x)
fMin = min(fX)
fMax = max(fX)
plt.plot(x, fX, 'b', [plotA, plotB], [0, 0], 'k')
plt.show()
```

Rezultat:

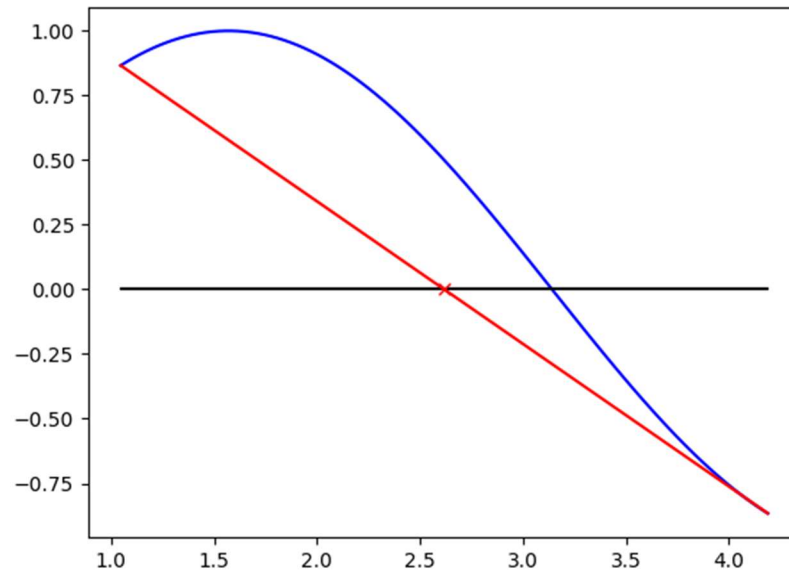


Slika 10. Grafik funkcije

2. Naći nulu sečice. Nacrtati sečicu i nacrtati znak 'x' u njenoj nuli crvenom bojom:

```
fA = f(a)
fB = f(b)
zero = b - fB*(b - a)/(fB - fA)
plt.plot([a, b], [fA, fB], 'r', zero, 0, 'rx')
```

Rezultat:



Slika 11. Sečica

3. Pretpostaviti nulu funkcije u nuli sečice i izračunati vrednost funkcije u pretpostavljenoj nuli:

```
fZero = f(zero)
```

Rezultat:

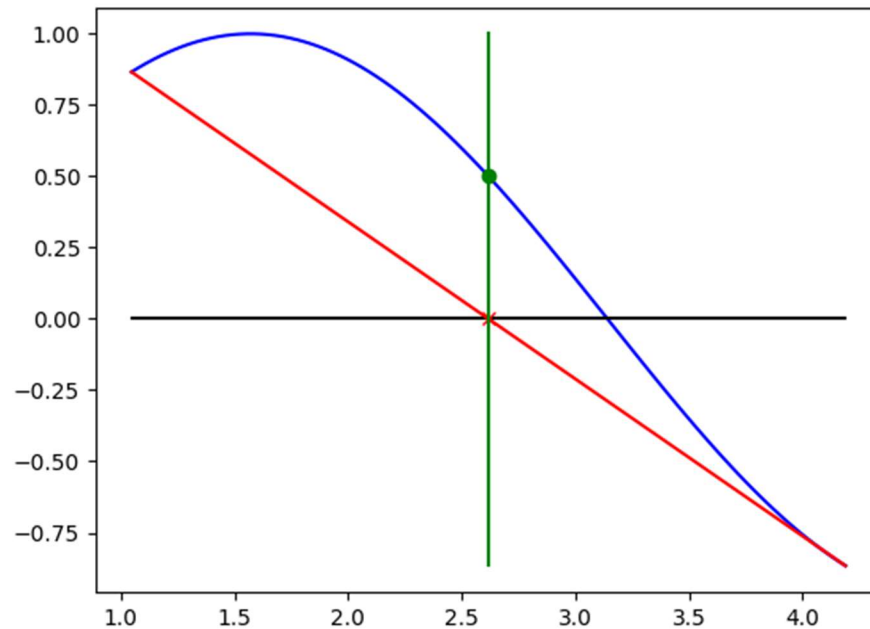
```
fZero = 0.5000
```

Izračunata vrednost je **različita od 0**.

4. Nacrtati vertikalnu osu i tačku određenu izračunatom vrednošću u pretpostavljenoj nuli funkcije zelenom bojom:

```
plt.plot([zero, zero], [fMin, fMax], 'g', zero, fZero, 'go')
```

Rezultat:



Slika 12. Pretpostavljena nula

5. Pripremiti podinterval za narednu iteraciju. Proglasiti kraj intervala za novi početak, a pretpostavljenu nulu za novi kraj intervala:

```
a = b  
b = zero
```

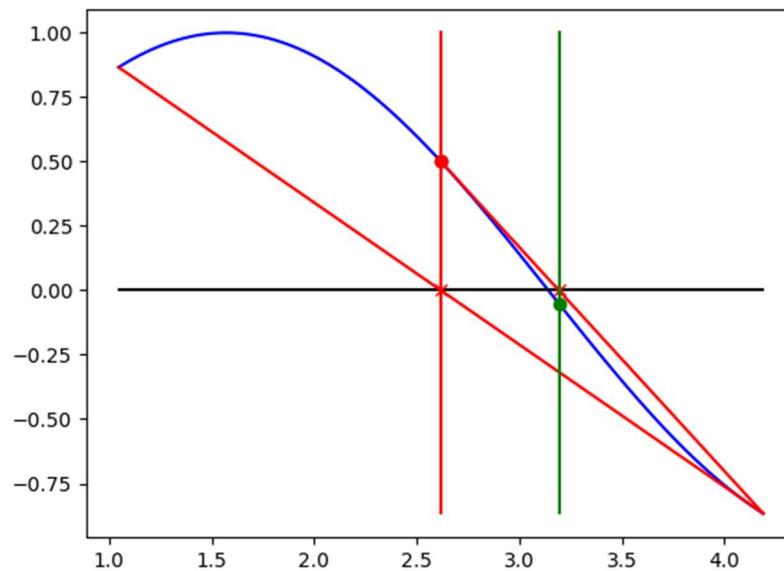
6. Prekriti zelenu vertikalnu osu i tačku iz prethodne iteracije crvenom bojom:

```
plt.plot([zero, zero], [fMin, fMax], 'r', zero, fZero, 'ro')
```

Ako se prethodni postupak (od tačke 2) ponavlja:

nakon 1. ponavljanja:

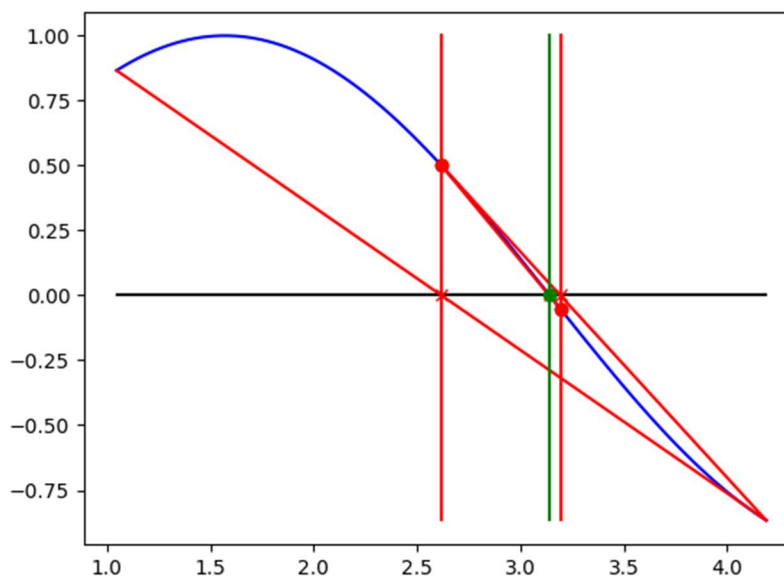
$fZero = -0.0513$



Slika 13. Nakon 1. ponavljanja

nakon 2. ponavljanja:

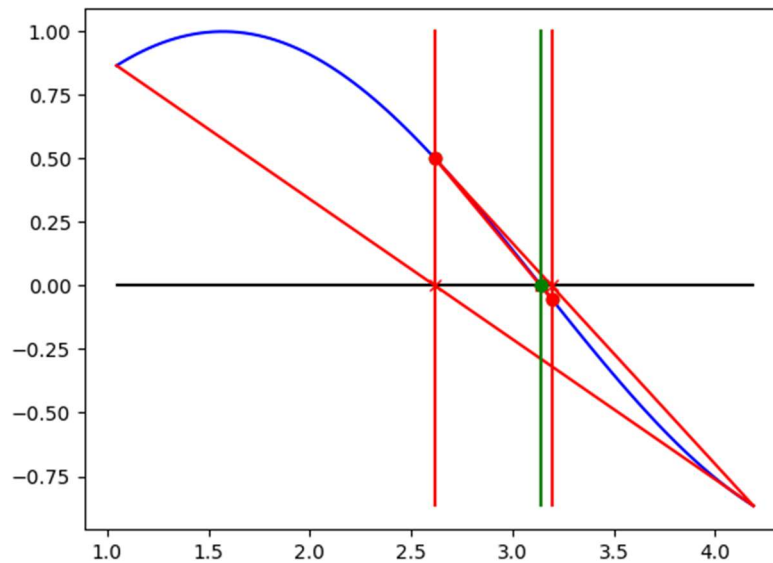
$fZero =$
0.0022



Slika 14. Nakon 2. ponavljanja

nakon 3. ponavljanja:

```
fZero =  
-9.1638e-07
```



Slika 15. Nakon 3. ponavljanja

7. Metoda sečice nema zagarantovanu konvergenciju. Prethodni postupak (od tačke 2) se može ugnjeziditi u *for* petlju sa **ograničenim brojem iteracija**:

```
for i in range(itMax):  
    fA = f(a)  
    fB = f(b)  
    zero = b - fB*(b - a)/(fB - fA)  
    plt.plot([a, b], [fA, fB], 'r', zero, 0, 'rx')  
  
    fZero = f(zero)  
    plt.plot([zero, zero], [fMin, fMax], 'g', zero, fZero, 'go')  
  
    a = b  
    b = zero  
  
    plt.plot([zero, zero], [fMin, fMax], 'r', zero, fZero, 'ro')
```

8. Potrebno je definisati uslov za prekid iteracije u slučaju da **vrednost funkcije u pretpostavljenoj nuli padne ispod tražene preciznosti**:

```
if abs(fZero) < errMax:  
    break
```

Sada je moguće definisati funkciju koja sadrži prethodni algoritam. Na kraju funkcije prikazati grafik:

```
def zeroSecant(f, a, b, errMax, itMax, plotSpeed):  
    plt.show()  
    return zero, i
```

9. Pauzirati algoritam u zavisnosti od tražene brzine iscrtavanja postupka:

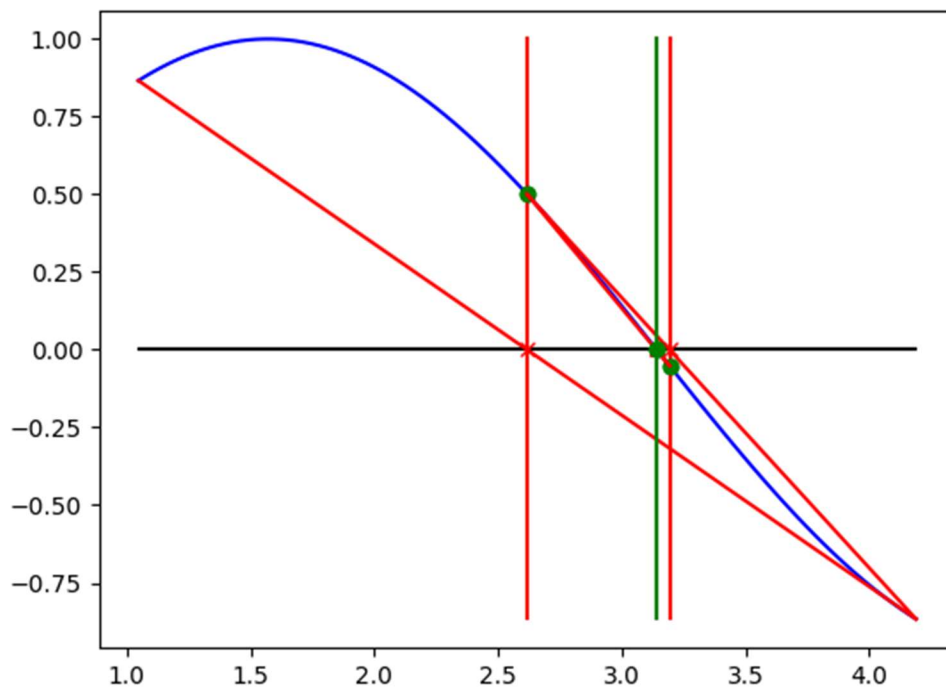
```
.  
.   
.   
if abs(fZero) < errMax:  
    break  
plt.pause(1/plotSpeed)  
.   
.   
.
```

10. Testirati funkciju zeroSecant na primeru:

```
f = lambda x: np.sin(x)  
a = np.pi/3  
b = 4*np.pi/3  
  
zero, it = zeroSecant(f, a, b, 0.0001, 100, 2)
```

Rezultat:

```
zero = 3.1416  
it = 4
```



Slika 16. Nula funkcije

11. Napraviti 2 podfunkcije u funkciji zeroSecant. Ako je prosleđena brzina iscrtavanja postupka 0 ili manja, pozvati varijantu funkcije bez naredbi za iscrtavanje i pauziranje algoritma:

```
def zeroSecant(f, a, b, errMax, itMax, plotSpeed):
    if f(a) == f(b):
        raise Exception('Invalid parameters: f(a) == f(b)!')

    if plotSpeed <= 0:
        return zeroSecantNoPlot(f, a, b, errMax, itMax)

    return zeroSecantPlot(f, a, b, errMax, itMax, plotSpeed)
```

12. Testirati funkciju zeroSecant na primeru:

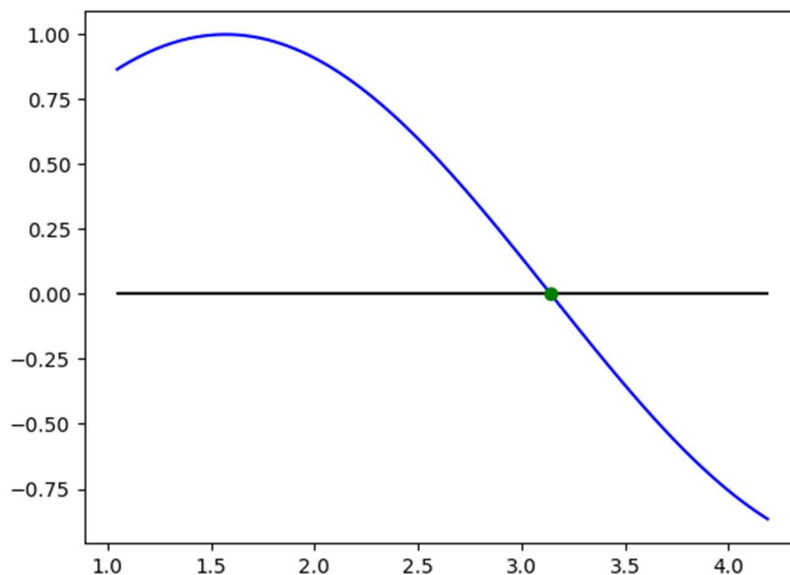
```
f = lambda x: np.sin(x)
a = np.pi/3
b = 4*np.pi/3

x = np.linspace(a, b, 100)
fX = f(x)
plt.plot(x, fX, 'b')
plt.plot([a, b], [0, 0], 'k')

zero, it = zeroSecant(f, a, b, 0.0001, 100, 0)
fZero = f(zero)

plt.plot(zero, fZero, 'go')
print(zero, it, fZero)
plt.show()

Rezultat:
zero = 3.1416
it = 4
fZero = -9.1638e-07
```



Slika 17. Nula funkcije

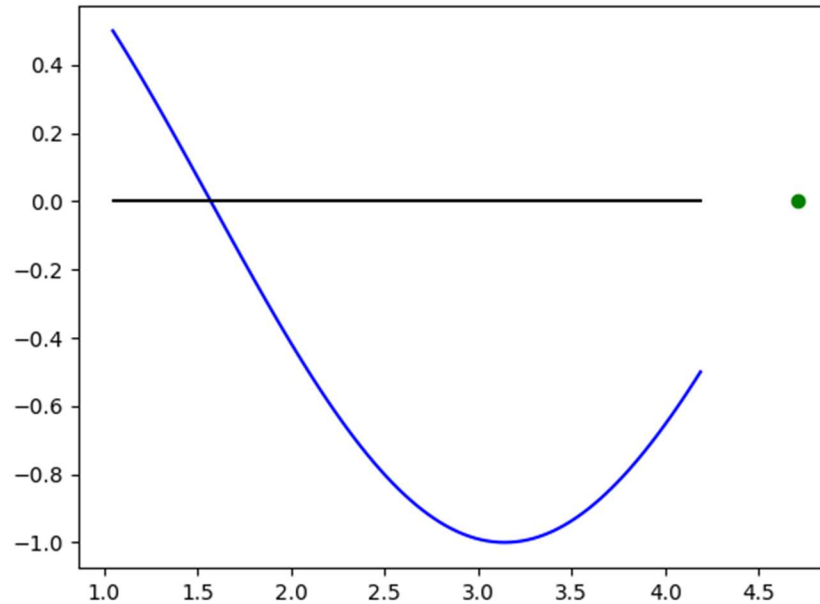
13. Testirati funkciju `zeroSecant` za funkciju $f(x) = \cos x$ na intervalu $x \in \left[\frac{\pi}{3}, \frac{4\pi}{3}\right]$.

Rezultat:

`zero = 4.7124`

`it = 6`

`fZero = 8.6152e-06`



Slika 18. Nula funkcije van intervala

Metoda sečice je **otvorena metoda**. Pronađena **nula funkcije je van traženog intervala**. Problem se može rešiti odabrom drugačijeg početnog intervala ili upotrebom zatvorene metode.

* **Zadatak 3.** Napisati metodu *False Position* za traženje nule nelinearnih funkcija.

3. Njutnova metoda

Zadatak 4. Napisati Njutnovu metodu za traženje nule nelinearnih funkcija.

Pokušati prvo ručno jednu iteraciju Njutnove metode nad funkcijom $f(x) = \sin x$. Analitički izvod funkcije je $f'(x) = \cos x$. Početi sa kraja intervala $x_0 = \frac{4\pi}{3}$:

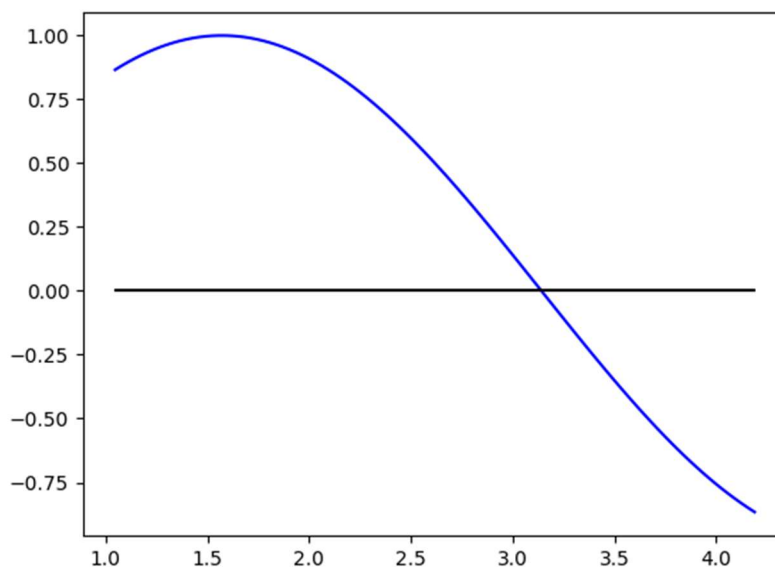
```
f = lambda x: np.sin(x)
df = lambda x: np.cos(x)
x0 = 4*np.pi/3

plotA = np.pi/3
plotB = 4*np.pi/3;
```

1. Nacrtati funkciju nad intervalom i naći njen minimum i maksimum i zadržati grafik:

```
x = np.linspace(plotA, plotB, 100)
fX = f(x)
fMin = np.min(fX)
fMax = np.max(fX)
plt.plot(x, fX, 'b', [plotA, plotB], [0, 0], 'k')
```

Rezultat:

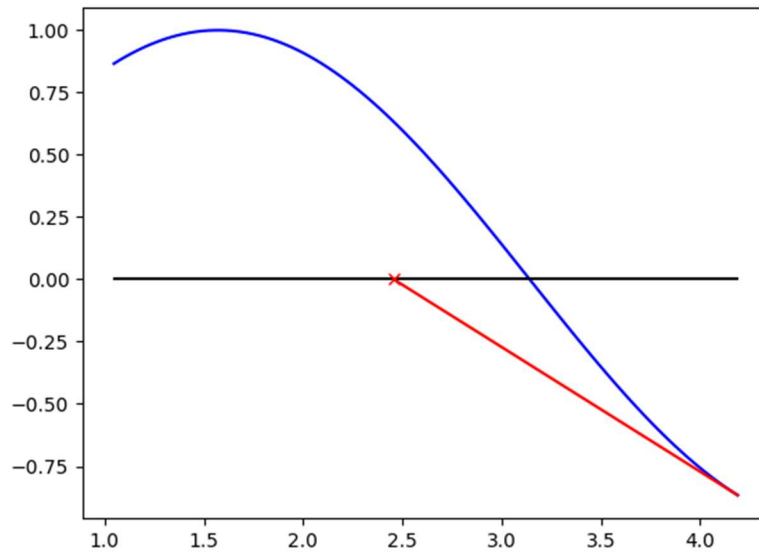


Slika 19. Grafik funkcije

2. Naći nulu tangente. Nacrtati tangentu i nacrtati znak 'x' u njenoj nuli crvenom bojom:

```
zero = x0 - f(x0)/df(x0)
plt.plot([x0, zero], [f(x0), 0], 'r', zero, 0, 'rx')
```

Rezultat:



Slika 20. Tangenta

3. Pretpostaviti nulu funkcije u nuli tangente i izračunati vrednost funkcije u pretpostavljenoj nuli:

```
fZero = f(zero)
```

Rezultat:

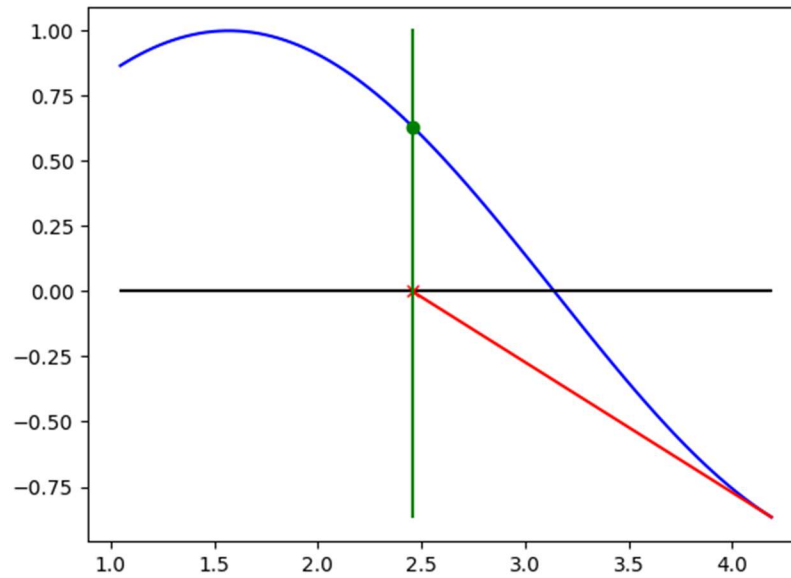
```
fZero = 0.6326
```

Izračunata vrednost je **različita od 0.**

4. Nacrtati vertikalnu osu i tačku određenu izračunatom vrednošću u pretpostavljenoj nuli funkcije zelenom bojom:

```
plt.plot([zero, zero], [fMin, fMax], 'g', zero, fZero, 'go')
```

Rezultat:



Slika 21. Pretpostavljena nula

5. Pripremiti podinterval za narednu iteraciju. Proglasiti pretpostavljenu nulu za novu početnu tačku:

```
x0 = zero
```

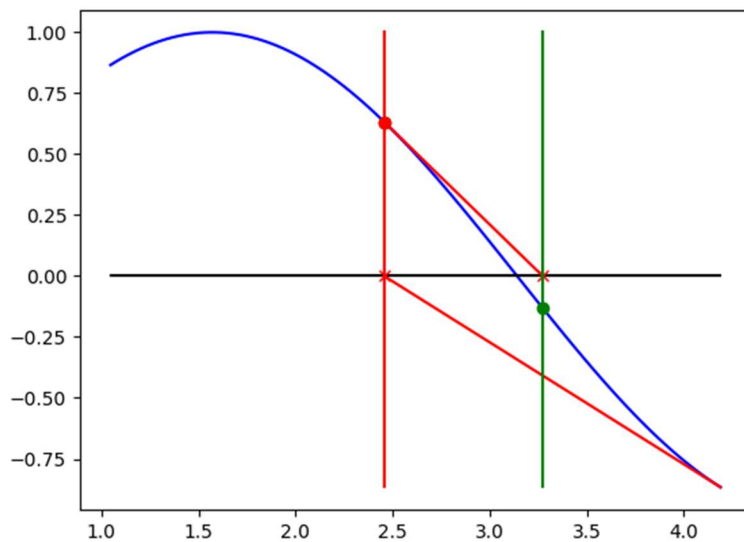
6. Prekriti zelenu vertikalnu osu i tačku iz prethodne iteracije crvenom bojom:

```
plt.plot([zero, zero], [fMin, fMax], 'r', zero, fZero, 'ro')
```

Ako se prethodni postupak (od tačke 2) ponavlja:

nakon 1. ponavljanja:

$fZero = -0.1315$

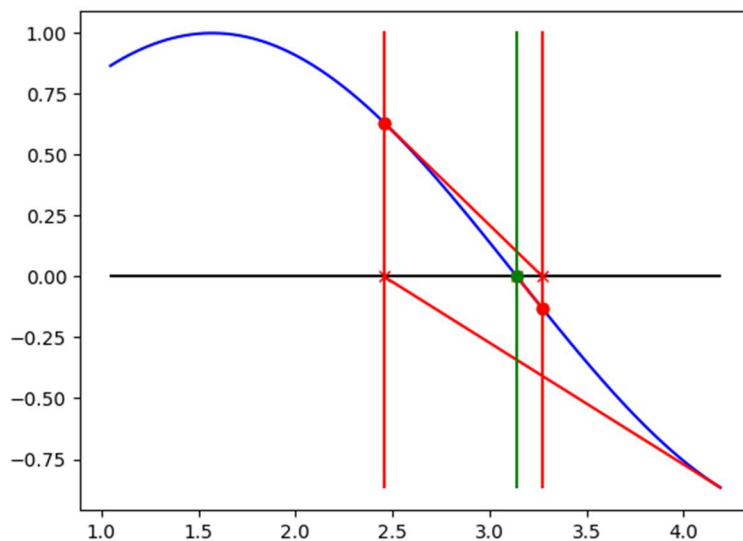


Slika 22. Nakon 1. ponavljanja

nakon 2. ponavljanja:

$fZero =$

$7.6969e-04$



Slika 23. Nakon 2. ponavljanja

nakon 3. ponavljanja:

fZero = -1.5199e-10

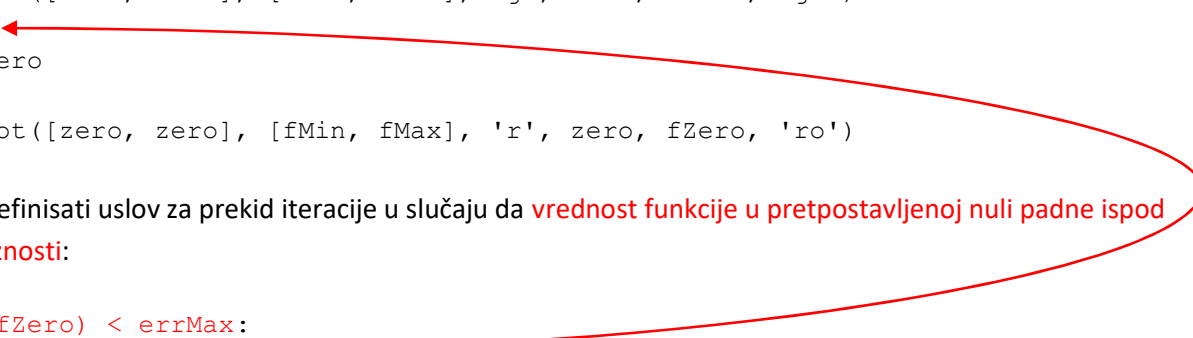
7. Njutnova metoda nema zagaranovanu konvergenciju. Prethodni postupak (od tačke 2) se može ugnjeziditi u *for* petlju sa **ograničenim brojem iteracija**:

```
for i in range(itMax):
    zero = x0 - f(x0)/df(x0)
    plt.plot([x0, zero], [f(x0), 0], 'r', zero, 0, 'rx')

    fZero = f(zero)
    plt.plot([zero, zero], [fMin, fMax], 'g', zero, fZero, 'go')

    x0 = zero

    plt.plot([zero, zero], [fMin, fMax], 'r', zero, fZero, 'ro')
```



8. Potrebno je definisati uslov za prekid iteracije u slučaju da **vrednost funkcije u pretpostavljenoj nuli padne ispod tražene preciznosti**:

```
if np.abs(fZero) < errMax:
    break
```

9. Sada je moguće definisati funkciju koja sadrži prethodni algoritam. Na početku funkcije zatvoriti eventualno postojeći prethodni grafik:

```
def zeroNewton(f, df, x0, errMax=0.0001, itMax=100, plotSpeed=-1, plotA=-5, plotB=5):
    .
    .
    .
    plt.show()
    return zero, i
```

10. Pauzirati algoritam u zavisnosti od tražene **brzine iscrtavanja postupka**:

```
.
.
.

if np.abs(fZero) < errMax:
    break
plt.pause(1/plotSpeed)
.
.
.
```

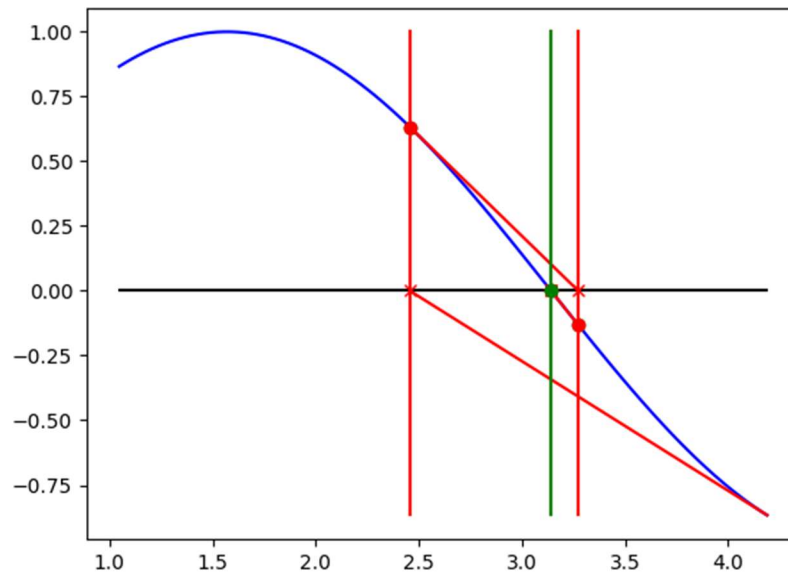
11. Testirati funkciju zeroNewton na primeru:

```
f = lambda x: np.sin(x)
df = lambda x: np.cos(x)
x0 = 4*np.pi/3

zero, it = zeroNewton(f, df, x0, 10**-5, 100, 2, np.pi/3, 4*np.pi/3)
print(zero, it)
```

Rezultat:

```
zero = 3.1416
it = 4
```



Slika 24. Nula funkcije

12. Napraviti 2 podfunkcije u funkciji zeroNewton. Ako je prosleđena brzina iscrtavanja postupka 0 ili manja, pozvati varijantu funkcije bez naredbi za iscrtavanje i pauziranje algoritma:

```
def zeroNewton(f, df, x0, errMax=0.0001, itMax=100, plotSpeed=-1, plotA=-5, plotB=5):
    if df(x0) == 0:
        raise('Invalid input: df(x0) == 0!')

    if plotSpeed <= 0:
        return zeroNewtonNoPlot(f, df, x0, errMax, itMax)

    return zeroNewtonPlot(f, df, x0, errMax, itMax, plotSpeed, plotA, plotB)
```

13. Testirati funkciju zeroNewton na primeru:

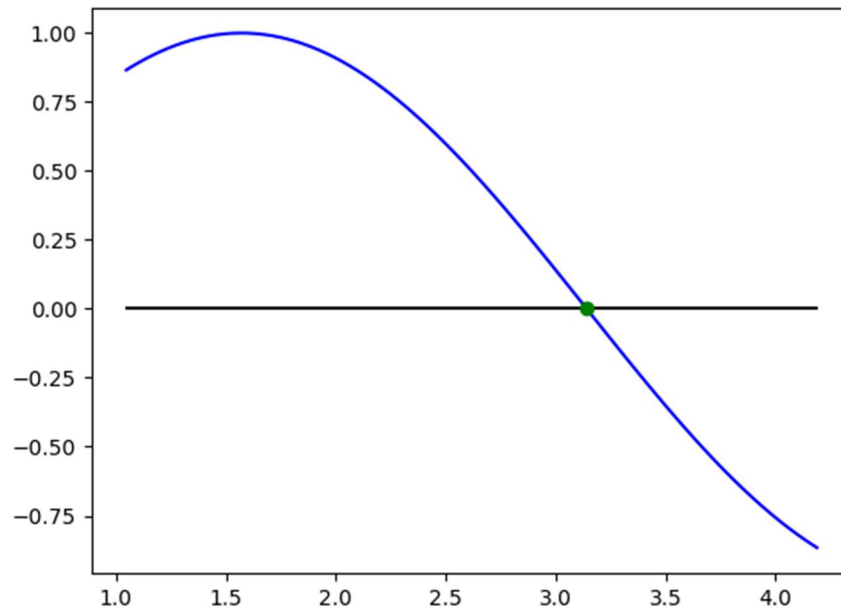
```
f = lambda x: np.sin(x)
df = lambda x: np.cos(x)
x0 = 4*np.pi/3

plotA = np.pi/3
plotB = 4*np.pi/3

x = np.linspace(plotA, plotB, 100)
fX = f(x)
plt.plot(x, fX, 'b')
plt.plot([plotA, plotB], [0, 0], 'k')

zero, it = zeroNewton(f, df, x0, 10**-5, 100, 0, np.pi/3, 4*np.pi/3)
fZero = f(zero)

plt.plot(zero, fZero, 'go')
print(zero, it, fZero)
plt.show()
Rezultat:
zero = 3.1416
it = 4
fZero = 1.5199e-10
```



Slika 25. Nula funkcije