8. Numerička integracija

Data je funkcija:

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

1. Naći vrednost određenog integrala funkcije $f(x)$ na intervalu $x \in \left[0, \frac{3\pi}{2}\right]$:

```
import numpy as np
from matplotlib import pyplot as plt
import scipy.integrate as integrate

f = lambda x: np.sin(x)
a = 0
b = 3*np.pi/2

I = integrate.quad(f, a, b)
```

Rezultat:

```
I =
    (1.0, 3.334366076909447e-14)
```

1. Trapezna metoda

Zadatak 1. Napisati trapeznu metodu za izračunavanje vrednosti određenog integrala proizvoljne funkcije nad proizvoljnim intervalom.

Pokušati prvo ručno jednu iteraciju trapezne metode nad funkcijom $f(x) = \sin x$ na intervalu $x \in \left[0, \frac{3\pi}{2}\right]$:

```
f = lambda x: np.sin(x)
a = 0
b = 3*np.pi/2
```

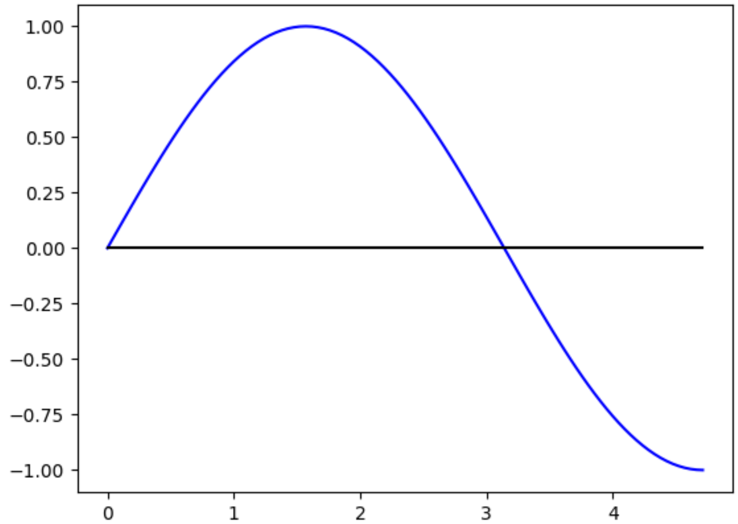
1. Definisati broj podintervala:

```
intervals = 4
```

2. Nacrtati funkciju i x-osu nad intervalom i zadržati grafik:

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

Rezultat:



Slika 1. Grafik funkcije

3. Izračunati širinu podintervala, na osnovu nje naći vrednosti nezavisno promenljive x u krajevima podintervala, a na osnovu njih naći vrednosti funkcije $f(x)$ u krajevima intervala:

```
width = (b - a)/intervals
x = np.linspace(a, b, intervals + 1)
fX = f(x)
```

4. Postaviti vrednost integralne sume na 0:

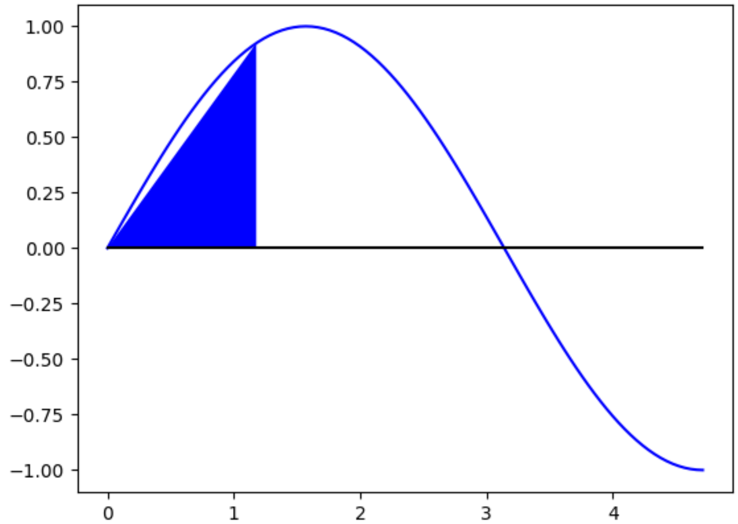
```
I = 0
```

Rezultat:
```
I =
    0
```

5. Na osnovu vrednosti funkcije u krajevima 1. podintervala i nacrtati trapeznu površ između x-ose i funkcije u 1. podintervalu:

```
x1 = x[0]
x2 = x[1]
fX1 = fX[0]
fX2 = fX[1]
plt.stackplot([x1, x2], [fX1, fX2], colors=['b'])
```

Rezultat:



Slika 2. 1. podinterval

6. Naći vrednost nacrtane površine i dodati je na integralnu sumu:

```
I = I + (fX1 + fX2)*width/2
```

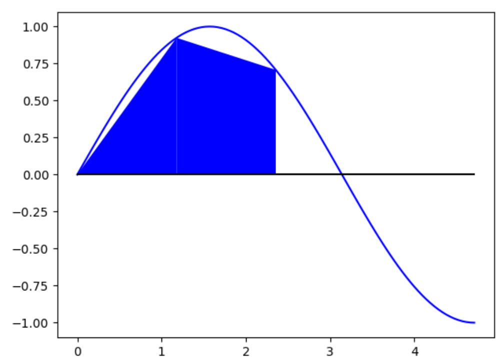
Rezultat:

```
I =
    0.5442
```

7. Ponoviti korake 5 i 6 za 2, 3 i 4. podinterval:

2. podinterval

```
x1 = x[1]
x2 = x[2]
fX1 = fX[1]
fX2 = fX[2]
.
.
.
```

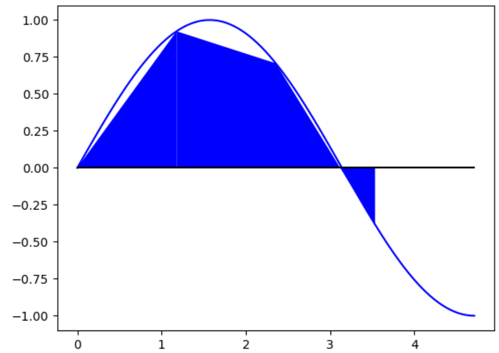


Rezultat:

```
I =
    1.5049
```

3. podinterval

```
x1 = x[2]
x2 = x[3]
fX1 = fX[2]
fX2 = fX[3]
.
.
.
```

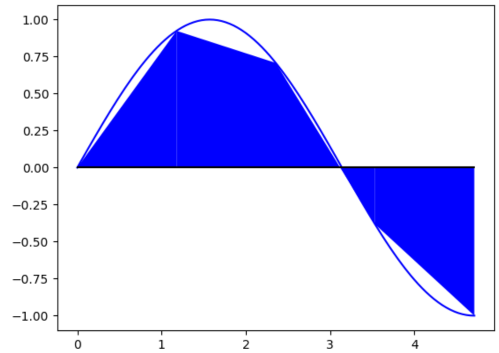


Rezultat:

```
I =
    1.6960
```

4. podinterval

```
x1 = x[3]
x2 = x[4]
fX1 = fX[3]
fX2 = fX[4]
.
.
.
```



Rezultat:

```
I =
    0.8816
```

Uporediti korake:

```
x1 = x[0]
x2 = x[1]
fX1 = fX[0]
fX2 = fX[1]
.
.
.

x1 = x[1]
x2 = x[2]
fX1 = fX[1]
fX2 = fX[2]
.
.
.

x1 = x[2]
x2 = x[3]
fX1 = fX[2]
fX2 = fX[3]
.
.
.

x1 = x[3]
x2 = x[4]
fX1 = fX[3]
fX2 = fX[4]
.
.
.
```

Proširiti indekse:

```
x1 = x[0]
x2 = x[0 + 1]
fX1 = fX[0]
fX2 = fX[0 + 1]
.
.
.

x1 = x[1]
x2 = x[1 + 1]
fX1 = fX[1]
fX2 = fX[1 + 1]
.
.
.

x1 = x[2]
x2 = x[2 + 1]
fX1 = fX[2]
fX2 = fX[2 + 1]
.
.
.

x1 = x[3]
x2 = x[3 + 1]
fX1 = fX[3]
fX2 = fX[3 + 1]
.
.
.
```

8. Primetiti šta je promenljivo. Koraci 5 i 6 se mogu zapisati jednom *for* petljom, pri čemu promenljivi indeksi zavise od indeksa *for* petlje:

```
I = 0
for it in range(intervals):
    x1 = x[it]
    x2 = x[it + 1]
    fX1 = fX[it]
    fX2 = fX[it + 1]
    plt.stackplot([x1, x2], [fX1, fX2], colors=['b']

    I = I + (fX1 + fX2) * width/2
```

9. Sada je moguće definisati funkciju koja sadrži prethodni algoritam.

```
def integrate_trapezoid(f, a, b, intervals, plotSpeed):
    .
    .
    .
    plt.show()
    return I
```

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

```
    .
    .
    .

    for it in range(intervals):
        .
        .
        .

        plt.pause(1/plotSpeed)
    plt.show()
    return I
```

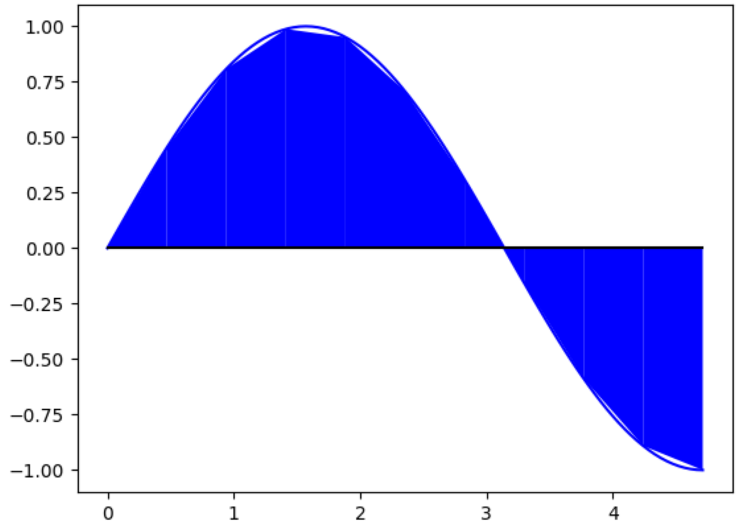
11. Testirati funkciju `integrate_trapezoid` na primeru:

```
f = lambda x: np.sin(x)
a = 0
b = 3*np.pi/2

I = integrate_trapezoid(f, a, b, 10, 10.0)
```

Rezultat:

```
I =
    0.9814
```



Slika 6. Trapezna metoda

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

```
def integrate_trapezoid(f, a, b, intervals, plotSpeed):
    if plotSpeed <= 0:
        return integrate_trapezoid_no_plot(f, a, b, intervals)

    return integrate_trapezoid_plot(f, a, b, intervals, plotSpeed)
```

13. Testirati funkciju `integrate_trapezoid` na primeru:

```
f = lambda x: np.sin(x)
a = 0
b = 3*np.pi/2

I = integrate_trapezoid(f, a, b, 1000, 0.0)
```

Rezultat:

```
I =
    1.0000
```

2. Simpsonova metoda

Zadatak 2. Napisati Simpsonovu 1/3 metodu za izračunavanje vrednosti određenog integrala proizvoljne funkcije nad proizvoljnim intervalom.
Pokušati prvo ručno jednu iteraciju Simpsonove 1/3 metode nad funkcijom $f(x) = \sin x$ na intervalu $x \in \left[0, \frac{3\pi}{2}\right]$:

```
f = lambda x: np.sin(x)
a = 0
b = 3*np.pi/2
```

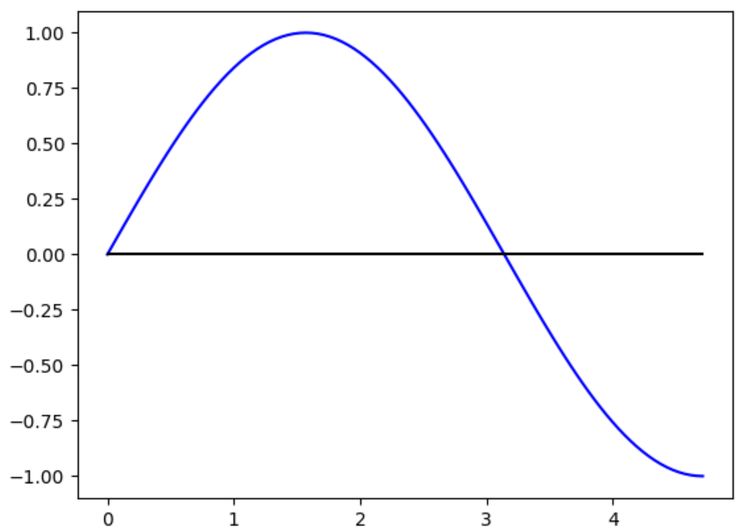
1. Definisati broj podintervala:

```
intervals = 4
```

2. Nacrtati funkciju i x-osu nad intervalom i zadržati grafik:

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

Rezultat:



Slika 7. Grafik funkcije

3. Izračunati širinu podintervala, na osnovu nje naći vrednosti nezavisno promenljive x u krajevima podintervala, a na osnovu njih naći vrednosti funkcije $f(x)$ u krajevima intervala:

```
width = (b - a)/intervals
x = np.linspace(a, b, intervals + 1)
fX = f(x)
```

4. Postaviti vrednost integralne sume na 0:

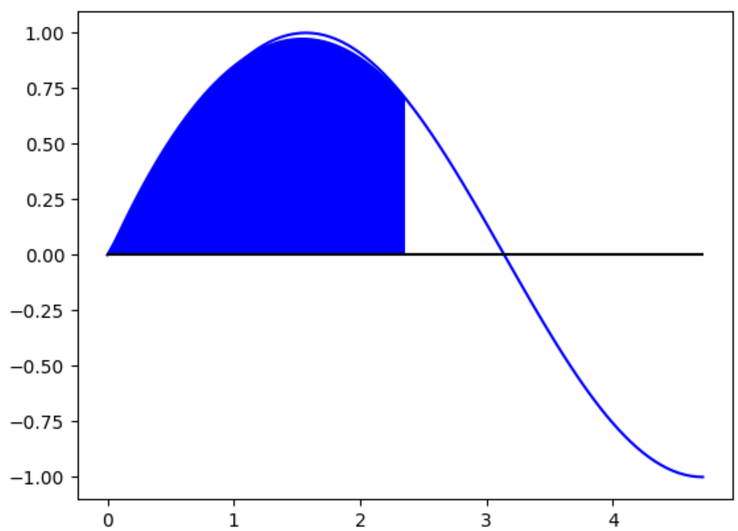
```
I = 0
```

Rezultat:
```
I =
     0
```

5. Na osnovu vrednosti funkcije u krajevima i između prva 2 podintervala i nacrtati površ između x-ose i aproksimirane funkcije polinomom 2. stepena na prva 2 podintervala:

```
x1 = x[0]
x2 = x[1]
x3 = x[2]
fX1 = fX[0]
fX2 = fX[1]
fX3 = fX[2]
p = np.polyfit([x1, x2, x3], [fX1, fX2, fX3], 2)
xP = np.linspace(x1, x3, 100)
fXP = np.polyval(p, xP)
plt.stackplot(xP, fXP, colors=['b'])
```

Rezultat:



Slika 8. Prva 2 podintervala

6. Naći vrednost nacrtane površine i dodati je na integralnu sumu:

```
I = I + (fX1 + 4*fX2 + fX3)*width/3
```

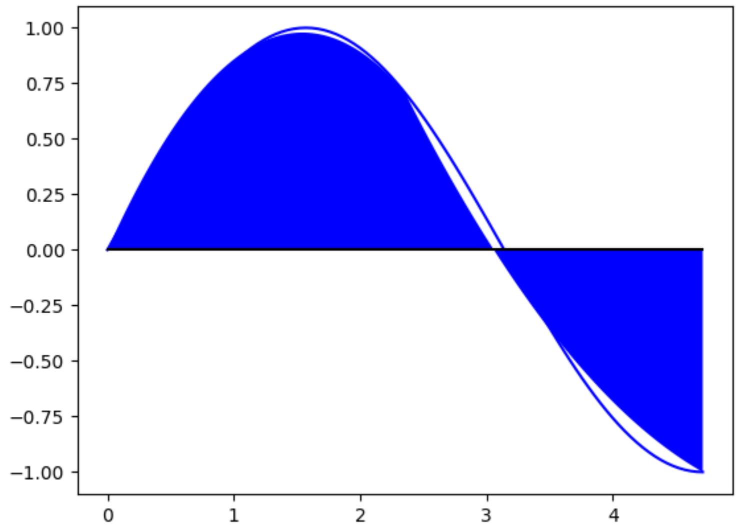
Rezultat:
```
I =
    1.7289
```

7. Ponoviti korake 5 i 6 za druga 2 podintervala:

```
x1 = x[2]
x2 = x[3]
x3 = x[4]
fX1 = fX[2]
fX2 = fX[3]
fX3 = fX[4]
.
.
.
```

Rezultat:

```
I =
    1.0128
```



Slika 9. Druga 2 podintervala

Uporediti korake:

```
x1 = x[0]
x2 = x[1]
x3 = x[2]
fX1 = fX[0]
fX2 = fX[1]
fX3 = fX[2]
.
.
.

x1 = x[2]
x2 = x[3]
x3 = x[4]
fX1 = fX[2]
fX2 = fX[3]
fX3 = fX[4]
.
.
.
```

Proširiti indekse:

```
x1 = x[0]
x2 = x[0 + 1]
x3 = x[0 + 2]
fX1 = fX[0]
fX2 = fX[0 + 1]
fX3 = fX[0 + 2]
.
.
.

x1 = x[2]
x2 = x[2 + 1]
x3 = x[2 + 2]
fX1 = fX[2]
fX2 = fX[2 + 1]
fX3 = fX[2 + 2]
.
.
.
```

10. Primetiti šta je promenljivo. Koraci 5 i 6 se mogu zapisati jednom *for* petljom, pri čemu promenljivi indeksi zavise od indeksa *for* petlje:

```
I = 0
for it in range(0, intervals, 2):
    x1 = x[it]
    x2 = x[it + 1]
    x3 = x[it + 2]
    fX1 = fX[it]
    fX2 = fX[it + 1]
    fX3 = fX[it + 2]
    p = np.polyfit([x1, x2, x3], [fX1, fX2, fX3], 2)
    xP = np.linspace(x1, x3, 100)
    fXP = np.polyval(p, xP)
    plt.stackplot(xP, fXP, colors=['b'])

    I = I + (fX1 + 4*fX2 + fX3) * width/3
```

11. Sada je moguće definisati funkciju koja sadrži prethodni algoritam.

```
def integrate_simpson(f, a, b, intervals, plotSpeed)
    .
    .
    .
    plt.show()
    return I
```

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

```
    .
    .
    .

    for it in range(0, intervals, 2):
        .
        .
        .

        plt.pause(1/plotSpeed)
    plt.show()
    return I
```

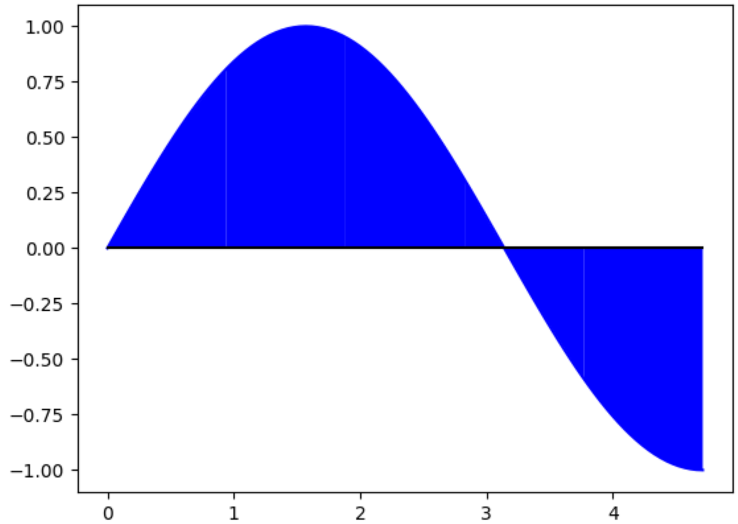
15. Testirati funkciju `integrate_trapezoid` na primeru:

```
f = lambda x: np.sin(x)
a = 0
b = 3*np.pi/2

I = integrate_simpson(f, a, b, 10, 10.0)
```

Rezultat:

```
I =
    1.0003
```



Slika 10. Simpsonova metoda

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

```
def integrate_simpson(f, a, b, intervals, plotSpeed):
    if plotSpeed <= 0:
        return integrate_simpson_no_plot(f, a, b, intervals)

    return integrate_simpson_plot(f, a, b, intervals, plotSpeed)
```

17. Testirati funkciju `integrate_simpson` na primeru:

```
f = lambda x: np.sin(x)
a = 0
b = 3*np.pi/2

I = integrate_simpson(f, a, b, 100, 0.0)
```

Rezultat:

```
I =
    1.0000
```