

PROGRAMIRANJE U INTEGRISANIM TEHNOLOGIJAMA

Oznaka predmeta: PIT

Predavanje broj: 08

Nastavna jedinica: PYTHON,

Nastavne teme:

Crtanje grafika funkcija (biblioteka matplotlib). Integracija (trapezno pravilo, Simpsonova metoda). Linearne jednačine (Gaus-Jordanov metod). Interpolacija: algebarski polinomi, Lagrange-ova, Newton-ova.

Predavač: prof. dr Perica S. Štrbac, dipl. ing.

Literatura:

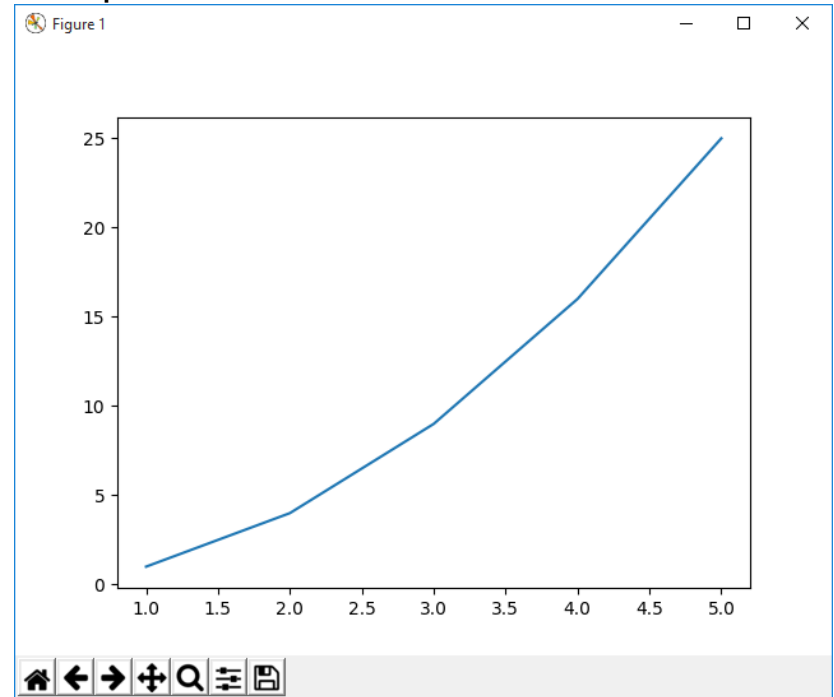
Steven Lott: "Functional Python Programming", Packt Publishing, 2015.

Crtanje grafika

- Ažurirajte pip sa: `python -m pip install --upgrade pip`
- Instalirati: `pip install matplotlib`
- Provera modula (standardno) :

```
>>> import pylab  
>>> import matplotlib  
>>> matplotlib.__version__
```

pylab je modul matplotlib-a
- Crtanje tačaka povezanih linijama



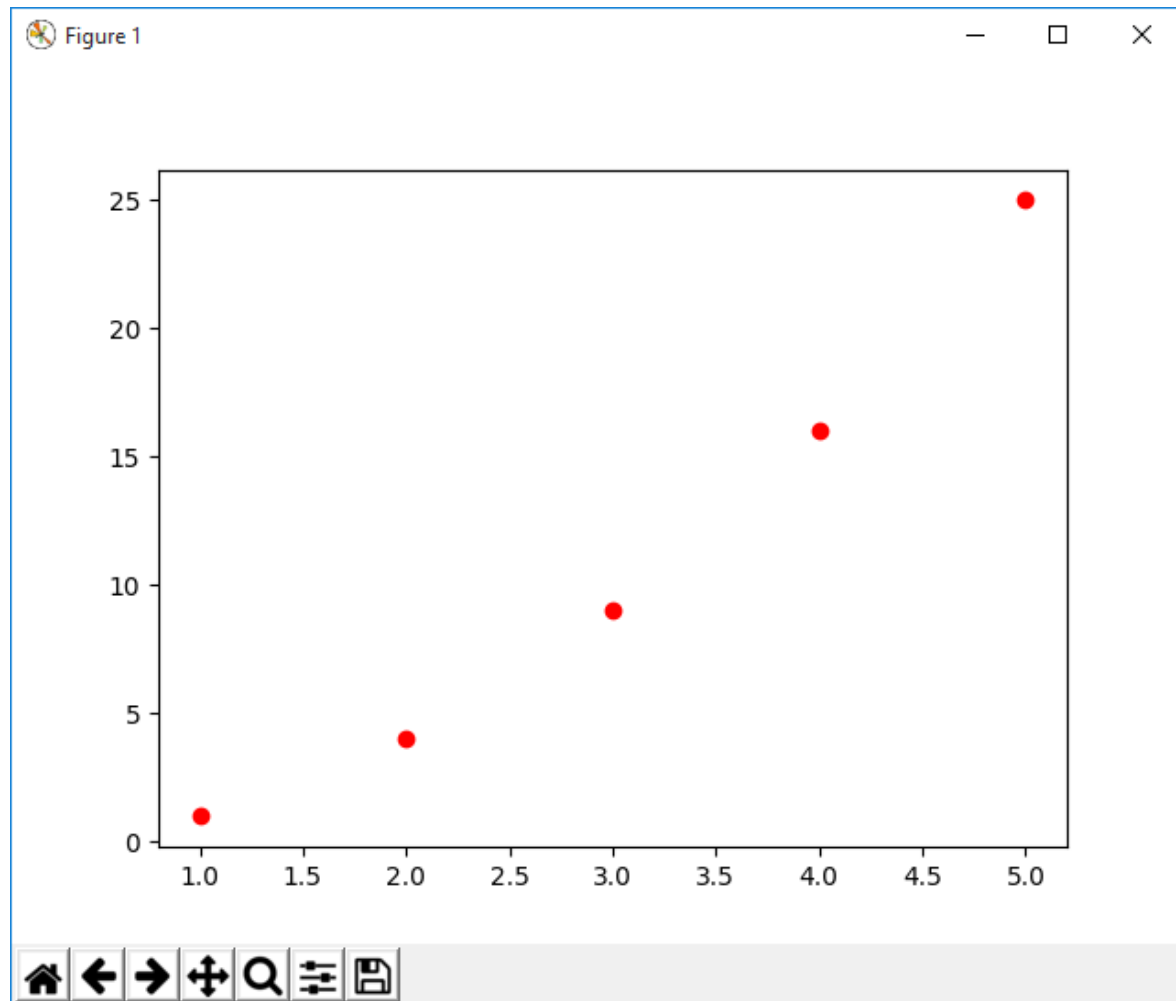
```
import pylab  
x = [1, 2, 3, 4, 5] # lista x vrednosti  
y = [1, 4, 9, 16, 25] # lista y=f(x)=x**2  
pylab.plot(x, y) # generisanje slike za  
# date x i y vrednosti  
pylab.show() # prikaz na ekranu
```

Crtanje grafika - tačkasto

- Prikaz tačkastih vrednosti

```
import pylab
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]
# generisanje slike za
# date x i y vrednosti
# r = crvena boja
# o = marker
# probajte r-*
pylab.plot(x, y, 'ro')
pylab.show()
```

```
# ako se samo navede Y
# lista
# pripadna x lista
# ide redom 1,2,...
```



Crtanje grafika – boje i parametri prikaza

- Menjanje boje crtanja

```
pylab.plot(x, y, 'r')
```

character	color
b	blue
g	green
r	red
c	cyan
m	magenta
y	yellow
k	black
w	white

- Menjanje stila linije

```
plot(x,y, '--') # crta crtkanu liniju
```

```
plot(x,y, 'b*') # tackasto crtanje znakom plavim znakom *
```

sledi popis nekih markera:

's' square marker

'p' pentagon marker

'*' star marker

'h' hexagon1 marker

'H' hexagon2 marker

'+' plus marker

'x' x marker

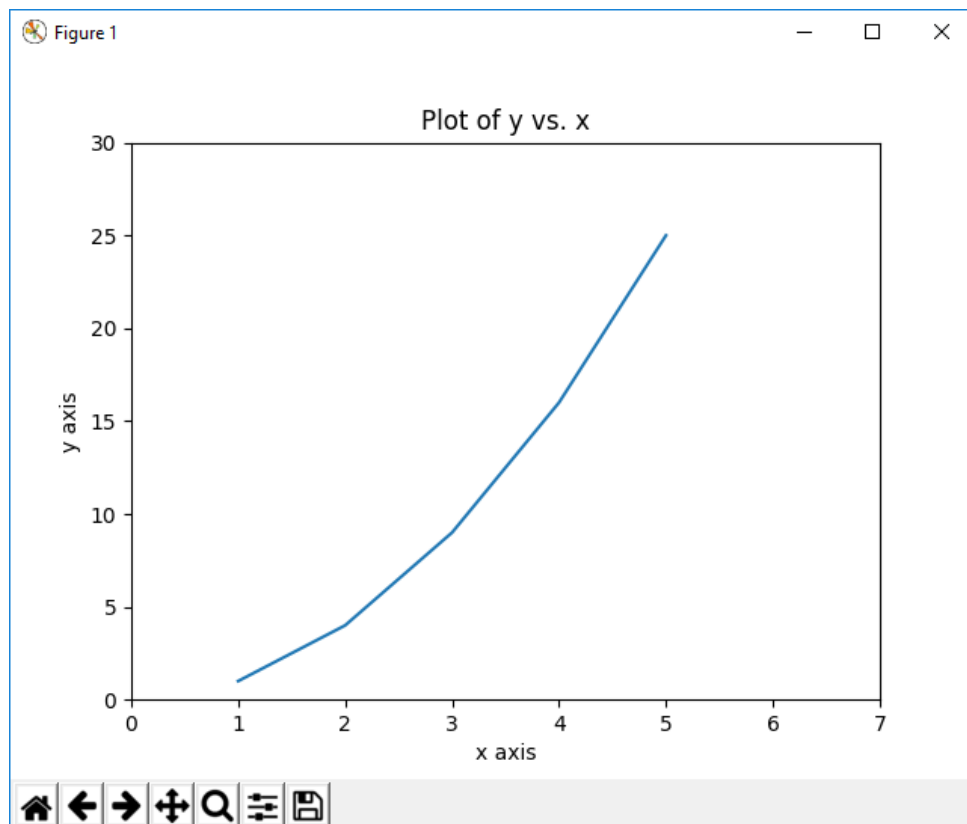
'D' diamond marker

'd' thin diamond marker

Crtanje grafika – labele, naslov, opsezi

- Postavljanje labele x ose i y ose:
`pylab.xlabel('X osa')`
`pylab.ylabel('Y osa')`
- Postavljanje naziva slike:
`pylab.title('NASLOV SLIKE')`
- Postavljanje opsega za prikaz po x i y osi:
`pylab.xlim(x_low, x_high)`
`pylab.ylim(y_low, y_high)`

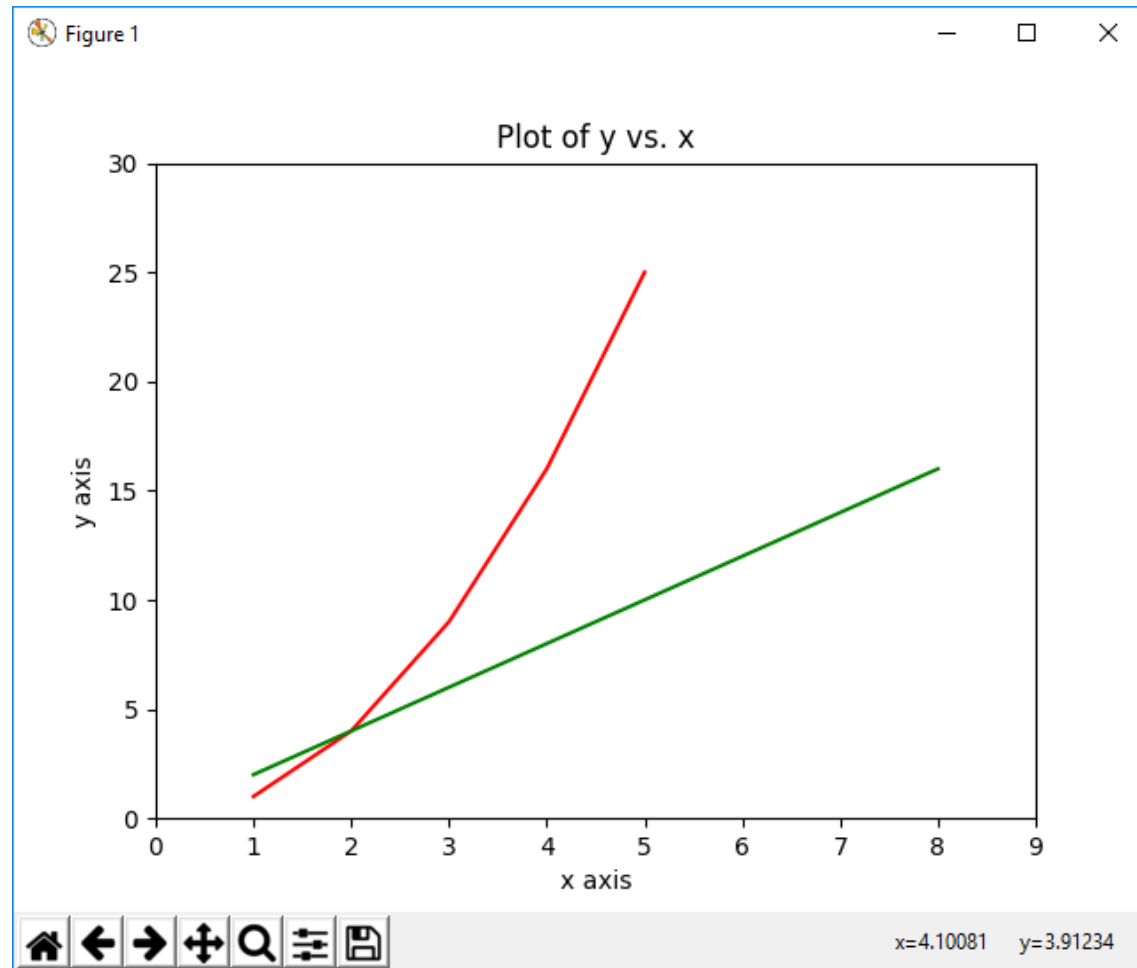
```
import pylab as pl
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]
pl.plot(x, y)
pl.title('Plot of y vs. x')
pl.xlabel('x axis')
pl.ylabel('y axis')
pl.xlim(0.0, 7.0)
pl.ylim(0.0, 30.)
pl.show()
```



Crtanje grafika – dva grafika

- Štampanje npr. dva grafika:

```
import pylab as pl
x1 = [1, 2, 3, 4, 5]
y1 = [1, 4, 9, 16, 25]
x2 = [1, 2, 4, 6, 8]
y2 = [2, 4, 8, 12, 16]
pl.plot(x1, y1, 'r')
pl.plot(x2, y2, 'g')
pl.title('Plot of y vs. x')
pl.xlabel('x axis')
pl.ylabel('y axis')
pl.xlim(0.0, 9.0)
pl.ylim(0.0, 30.0)
pl.show()
```



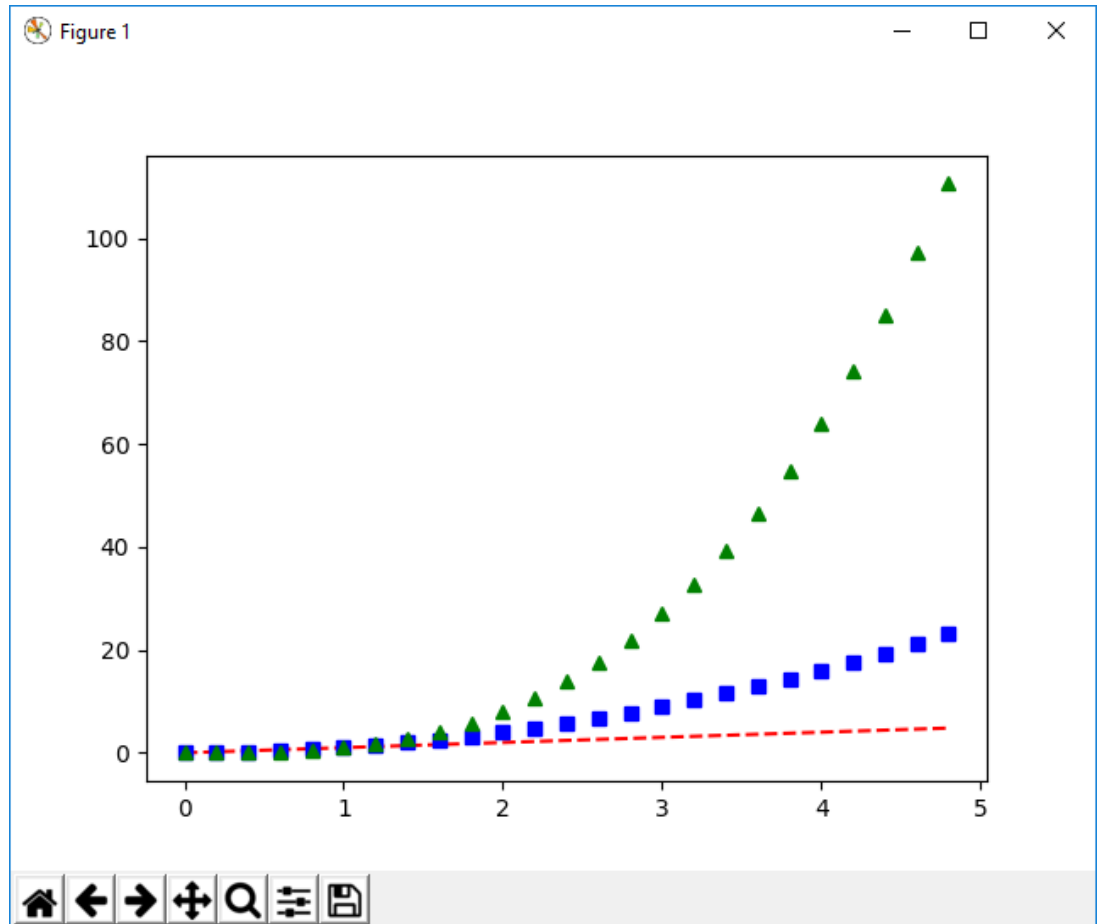
Crtanje grafika – više grafika različitog prikaza

- Štampanje tri grafika sa različitim formatom prikaza:

```
import pylab as pl
import numpy as np
```

```
t = np.arange(0., 5., 0.2)
```

```
# red dashes,
# blue squares
# green triangles
```

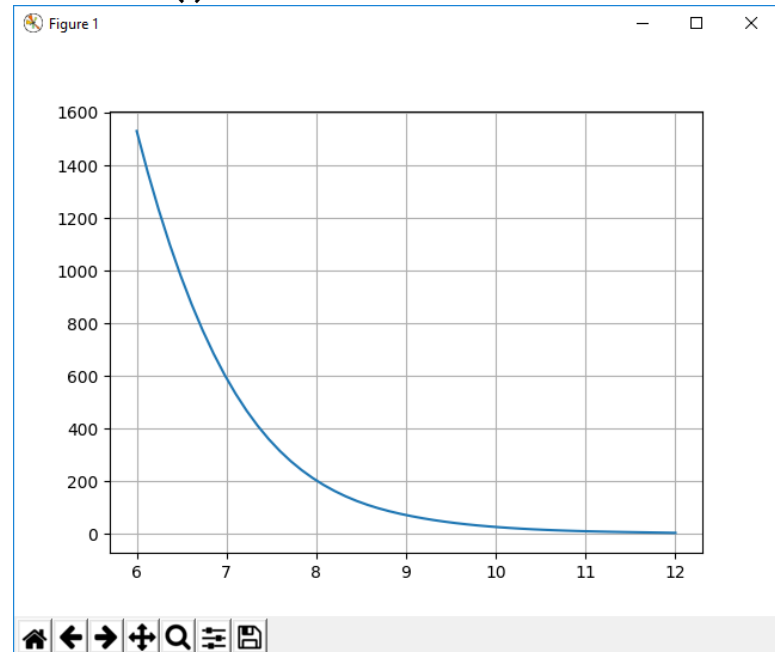
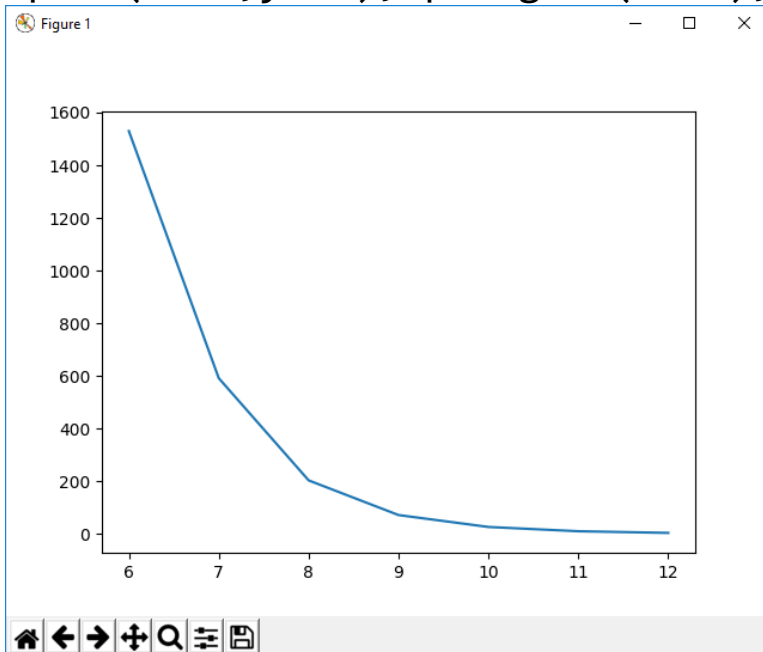


```
pl.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
pl.show()
```

Crtanje grafika – glatki prikaz

- Štampanje glatkog grafika:

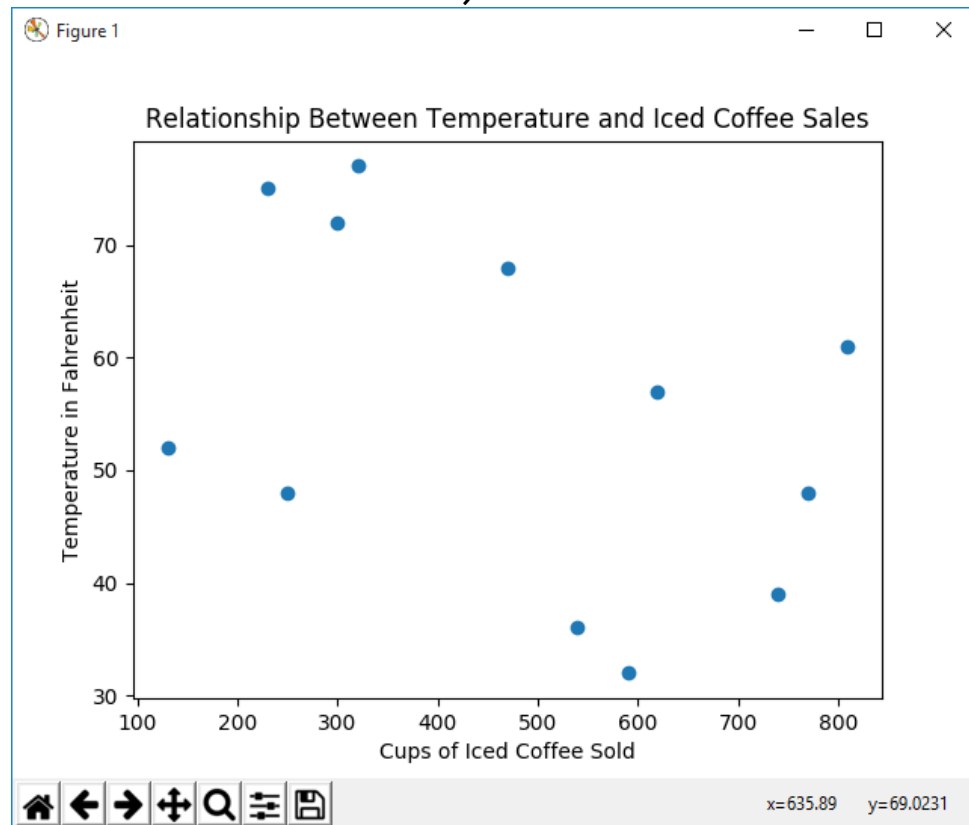
```
from scipy.interpolate import spline
import numpy as np
import matplotlib.pyplot as plt
x = np.array([6, 7, 8, 9, 10, 11, 12])
y = np.array([1.53E+03, 5.92E+02, 2.04E+02, 7.24E+01, 2.72E+01,
1.10E+01, 4.70E+00])
xnew = np.linspace(x.min(),x.max(),50)
ynew = spline(x,y,xnew)
plt.plot(xnew,ynew); plt.grid(True);plt.show()
```



Tačkasti prikaz: scatter

- Crtanje tačaka čije su vrednosti koordinata date kao liste x i y vrednosti

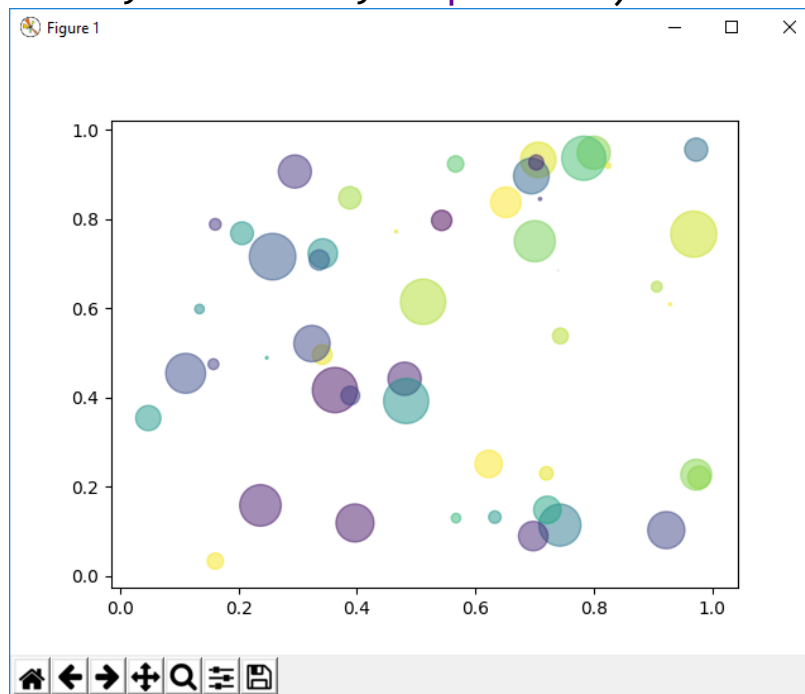
```
import matplotlib.pyplot as plt
X = [590,540,740,130,810,300,320,230,470,620,770,250]
Y = [ 32, 36, 39, 52, 61, 72, 77, 75, 68, 57, 48, 48]
plt.title('Relationship Between Temperature and Iced Coffee Sales')
plt.xlabel('Cups of Iced Coffee Sold')
plt.ylabel('Temperature in Fahrenheit')
plt.scatter(X,Y)
plt.show()
```



Primer tačkastog prikaza

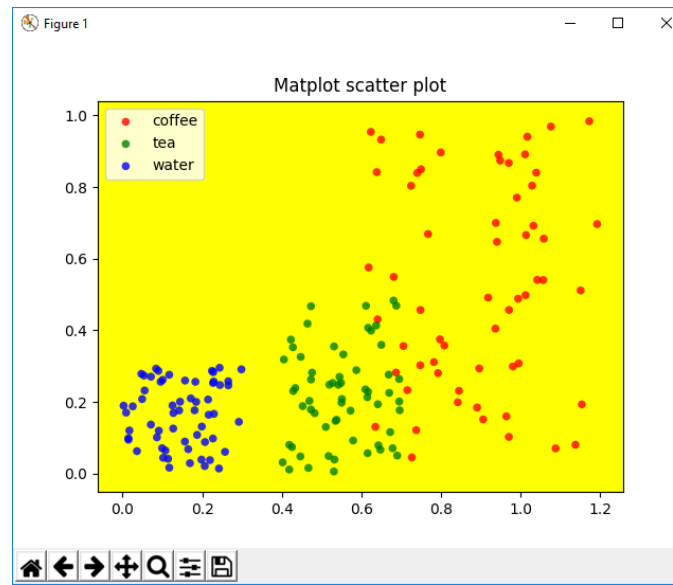
- Crtanje tačaka različite boje, veličine i neprovidnosti.

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(19680801)
N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area = np.pi * (15 * np.random.rand(N))**2 # r=[0,15)
plt.scatter(x, y, s=area, c=colors, alpha=0.5)
plt.show()
```



Primer tačkastog prikaza po grupama

```
import numpy as np
import matplotlib.pyplot as plt
N = 60
g1 = (0.6 + 0.6 * np.random.rand(N), np.random.rand(N))
g2 = (0.4 + 0.3 * np.random.rand(N), 0.5 * np.random.rand(N))
g3 = (0.3 * np.random.rand(N), 0.3 * np.random.rand(N))
data = (g1, g2, g3)
colors = ("red", "green", "blue")
groups = ("coffee", "tea", "water")
fig = plt.figure()
ax = fig.add_subplot(1,1,1,axisbg="yellow")
for data,color,group in zip(data, colors, groups):
    x, y = data
    ax.scatter(x,y,alpha=0.8, c=color, edgecolors='none', s=30,
label=group)
plt.title('Matplot scatter plot')
plt.legend(loc=2)
plt.show()
```



Dva plot-a na istoj slici

- Sledi primer crtanja dva odvojena grafikona:

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
def f(t):
```

```
    return np.exp(-t) * np.cos(2*np.pi*t)
```

```
t1 = np.arange(0.0, 5.0, 0.1)
```

```
t2 = np.arange(0.0, 5.0, 0.02)
```

```
plt.figure(1)
```

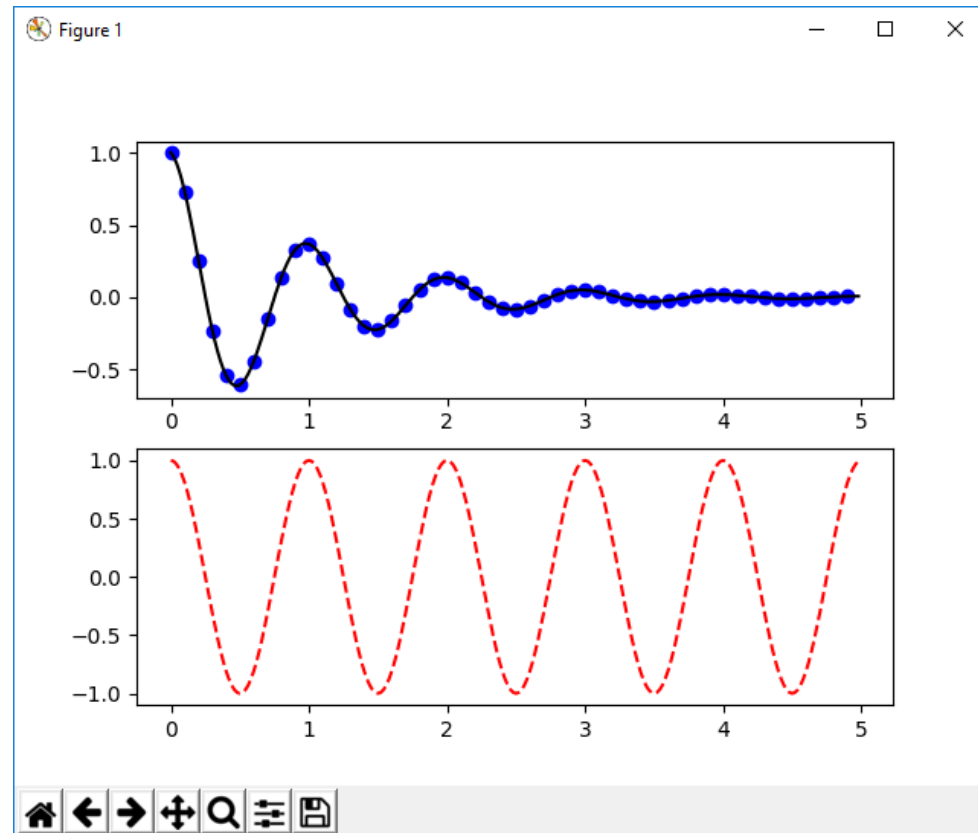
```
plt.subplot(211)
```

```
plt.plot(t1,f(t1),'bo',t2,f(t2),'k')
```

```
plt.subplot(212)
```

```
plt.plot(t2,np.cos(2*np.pi*t2),'r--')
```

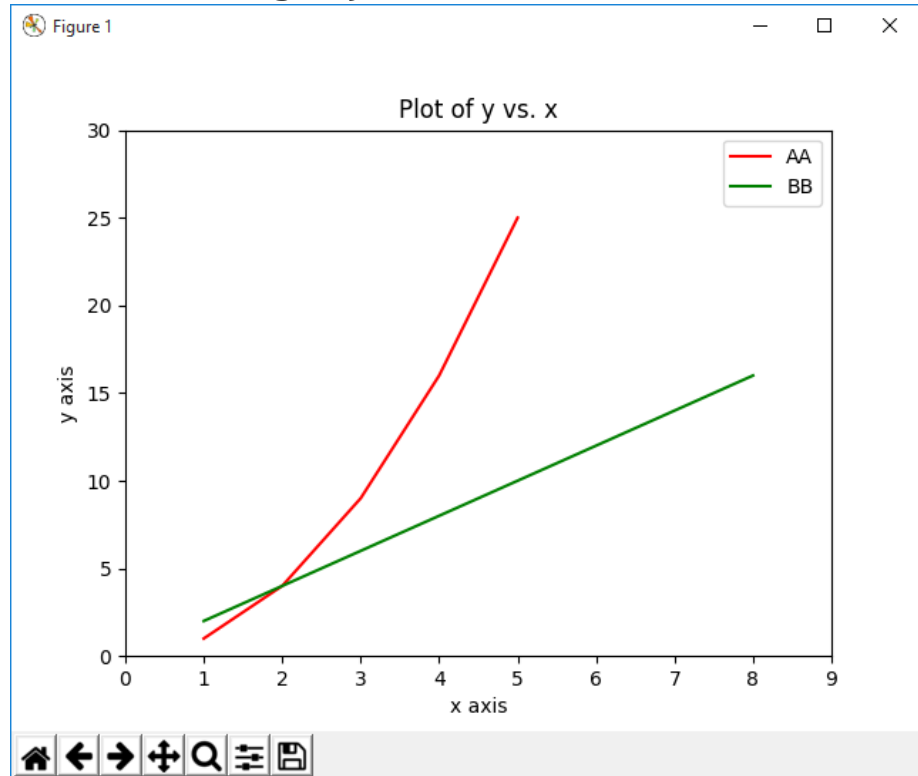
```
plt.show()
```



Legenda

- Neki parametri legende:
 - **handles**: lista linija, **labels**: lista labela, **loc**: pozicija labela ('best', 'upper right', 'upper left', 'center', 'lower left', 'lower right').

```
import pylab as pl
x1 = [1, 2, 3, 4, 5]
y1 = [1, 4, 9, 16, 25]
x2 = [1, 2, 4, 6, 8]
y2 = [2, 4, 8, 12, 16]
pl.title('Plot of y vs. x')
pl.xlabel('x axis')
pl.ylabel('y axis')
pl.xlim(0.0, 9.0)
pl.ylim(0.0, 30.)
```



```
linered, = pl.plot(x1,y1,'r', label='Red line')
linegreen, = pl.plot(x2,y2,'g', label='Green line')
pl.legend(handles=[linered, linegreen], labels=['AA', 'BB'], loc='upper
right'); pl.show()
```

Legenda: Patch, Line2D

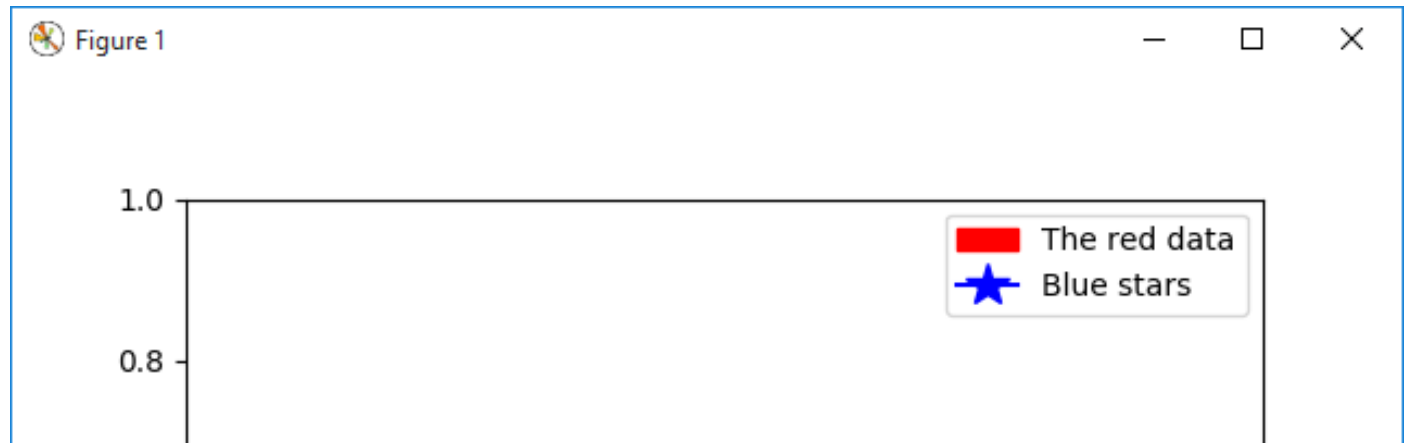
- Kreiranje legende pomoću Patch i Line2D

```
import matplotlib.patches as mpatches
import matplotlib.lines as mlines
import matplotlib.pyplot as plt
```

```
red_patch = mpatches.Patch(color='red', label='The red data')
```

```
blue_line = mlines.Line2D([], [], color='blue', marker='*',
                           markersize=15, label='Blue stars')
```

```
plt.legend(handles=[red_patch, blue_line])
plt.show()
```



Prikaz legende na datoj poziciji

- Kreiranje legende na datoj poziciji i sa definisanim izgledom

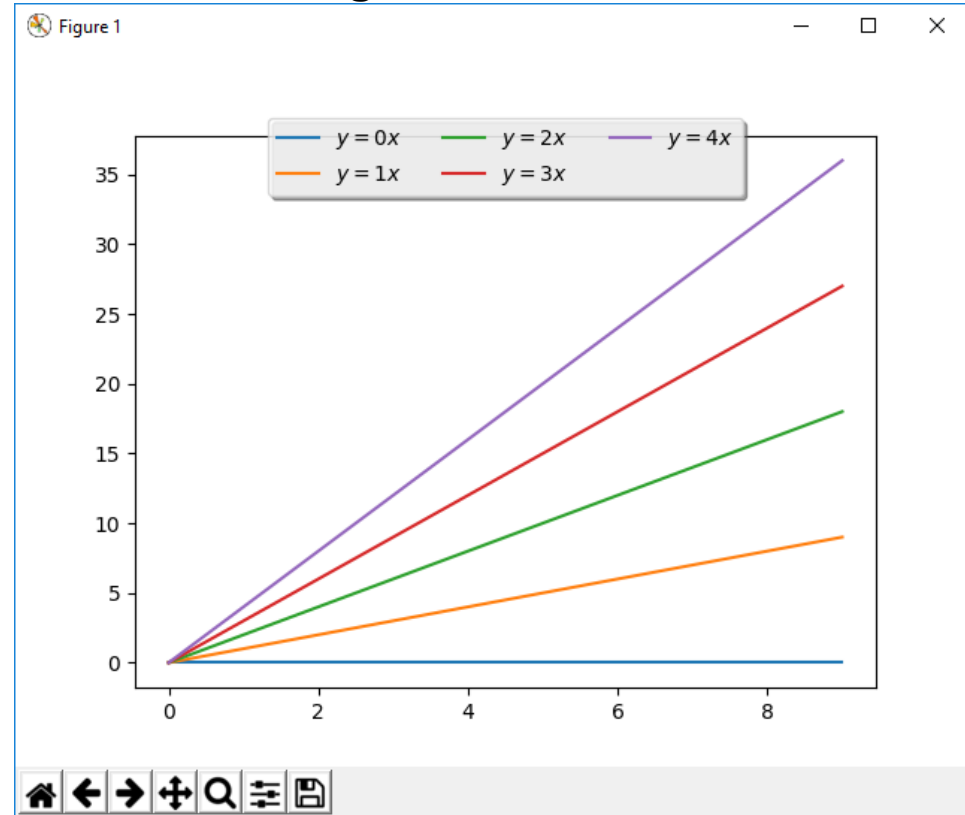
```
import matplotlib.pyplot as plt
import numpy as np
```

```
x = np.arange(10)
```

```
fig = plt.figure()
ax = plt.subplot(111)
```

```
for i in range(5):
    line, = ax.plot(x, i * x, label='$y = %ix$'%i)

ax.legend(loc='upper center', bbox_to_anchor=(0.5, 1.05),
          ncol=3, fancybox=True, shadow=True)
plt.show()
```



Prikaz dve legende

```
import matplotlib.pyplot as plt
```

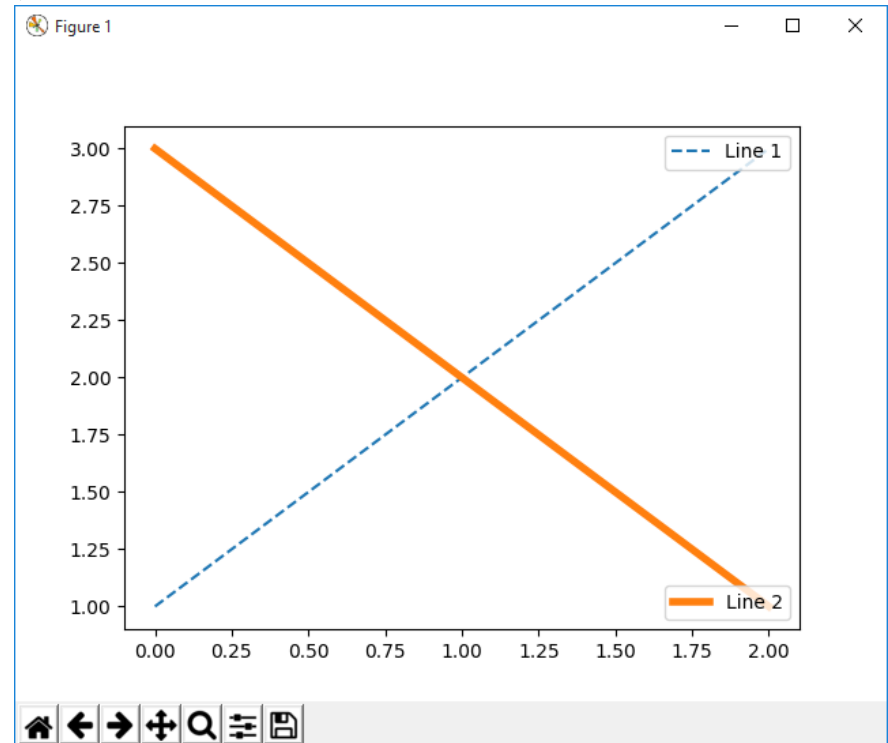
```
line1, = plt.plot([1,2,3], label="Line 1", linestyle='--')  
line2, = plt.plot([3,2,1], label="Line 2", linewidth=4)
```

```
first_legend = plt.legend(handles=[line1], loc=1)
```

```
ax = plt.gca().add_artist(first_legend)
```

```
plt.legend(handles=[line2], loc=4)
```

```
plt.show()
```



Prikaz dve slike grafika

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

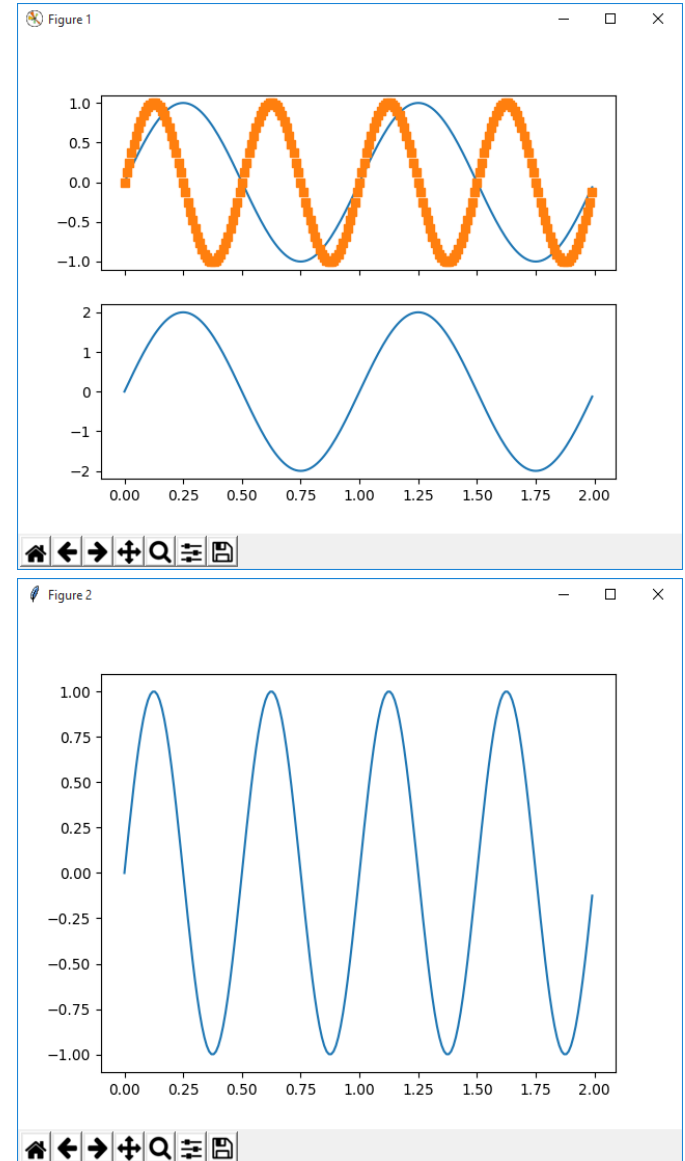
t = np.arange(0.0, 2.0, 0.01)
s1 = np.sin(2*np.pi*t)
s2 = np.sin(4*np.pi*t)

plt.figure(1)
plt.subplot(211); plt.plot(t, s1)
plt.subplot(212); plt.plot(t, 2*s1)

plt.figure(2)
plt.plot(t, s2)

plt.figure(1) # opet malo promene na fig.1
plt.subplot(211)
plt.plot(t, s2, 's')
ax = plt.gca()
ax.set_xticklabels([])

plt.show()
```



3D prikaz

```
import matplotlib as mpl
import numpy as np
import matplotlib.pyplot as plt
```

```
mpl.rcParams['legend.fontsize'] = 10 # tekst → parametric curve
```

```
fig = plt.figure()
ax = fig.gca(projection='3d')
```

```
theta = np.linspace(-4*np.pi, 4*np.pi, 100)
```

```
z = np.linspace(-2, 2, 100)
```

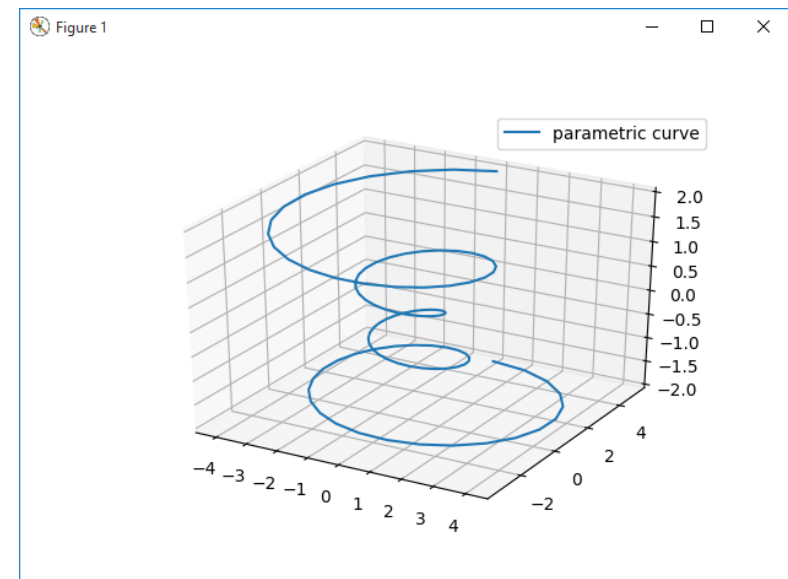
```
r = z**2 + 1
```

```
x = r * np.sin(theta)
```

```
y = r * np.cos(theta)
```

```
ax.plot(x,y,z,label='parametric curve')
ax.legend()
```

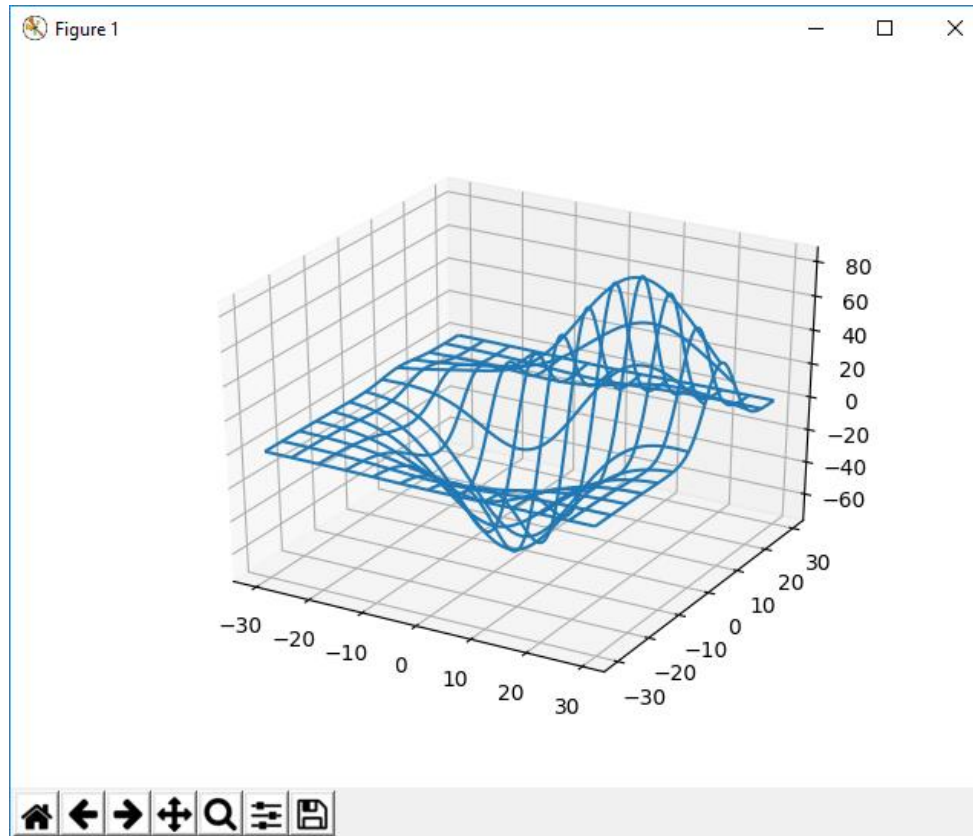
```
plt.show()
```



3D prikaz

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
```

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
X, Y, Z = axes3d.get_test_data(0.05)
ax.plot_wireframe(X, Y, Z, rstride=10, cstride=10)
plt.show()
```



3D prikaz

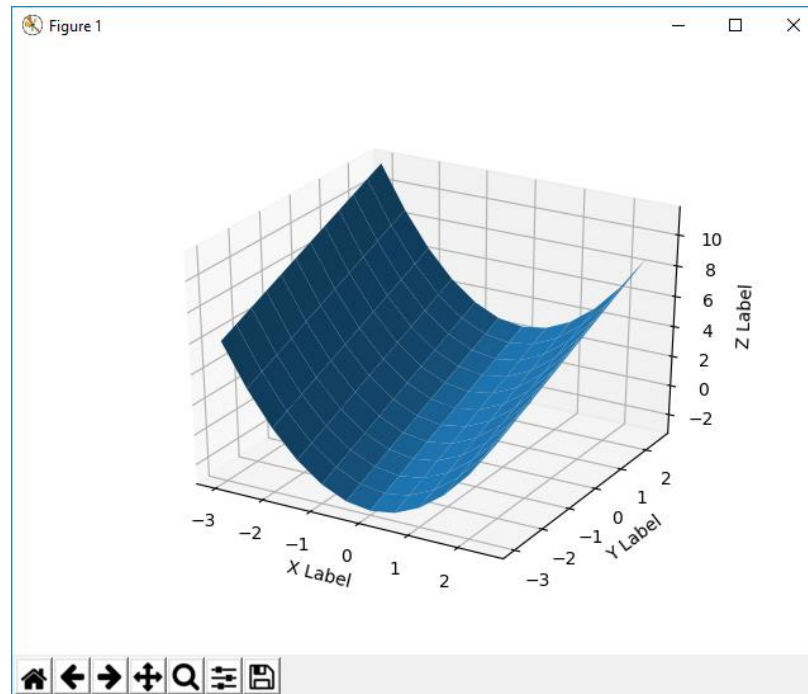
```
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

def fun(x, y): return x**2 + y
fig = plt.figure()
# fig.suptitle('Naslov')
ax = fig.add_subplot(111, projection='3d')
x = y = np.arange(-3.0, 3.0, 0.5)
X, Y = np.meshgrid(x, y)
zs = np.array([fun(x,y) for x,y in zip(np.ravel(X), np.ravel(Y))])
Z = zs.reshape(X.shape)

ax.plot_surface(X, Y, Z)

ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')

plt.show()
```



Uvod u numeričku integraciju

- Postoje dva različita tipa integrala: neodređeni i određeni integral

- Neodređeni integral

$$\int x^2 dx = \frac{x^3}{3}$$

Rezultat integracije je funkcija

- Određeni integral
$$\int_0^1 x^2 dx = \frac{1}{3}$$

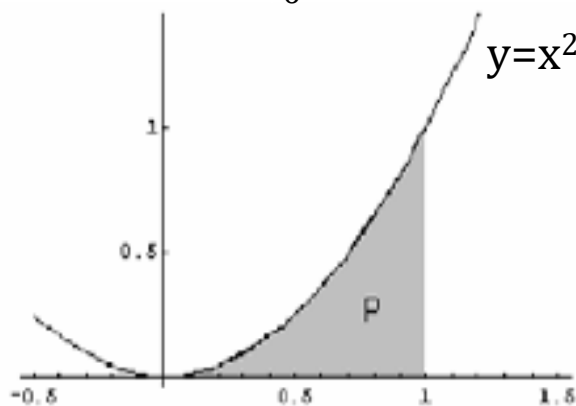
Rezultat integracije je broj

- Samo za rešavanje određenih integrala se mogu upotrebiti numeričke metode.

Numerička integracija

- Određeni integral se geometrijski interpretira kao površina ispod krive. Na primer za parabolu $y=x^2$ u granicama od $x=0$ do $x=1$ to je

$$\int_0^1 x^2 dx = \frac{x^3}{3} \Big|_0^1 = \left(\frac{1^3}{3} \right) - \left(\frac{0^3}{3} \right) = \frac{1}{3}$$

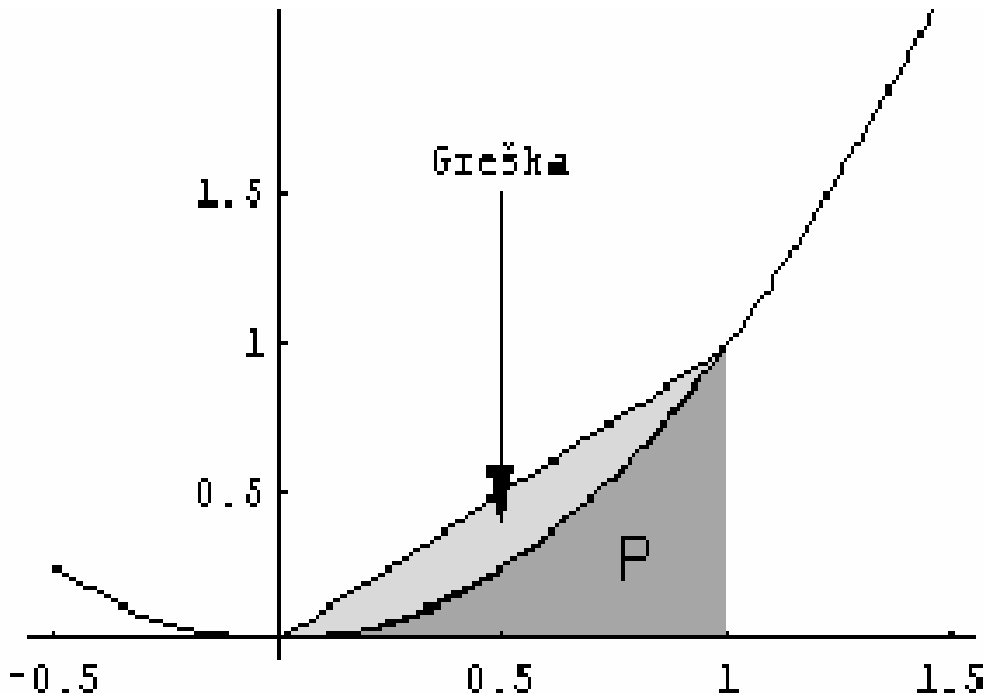


- Površina ispod krive $y=x^2$ u granicama od $x=0$ do $x=1$ iznosi $P=1/3$
- Za ovaj proces postoje i numeričke aproksimacije nazvane numeričko integriranje (kvadratura).

Trapezno pravilo

- početak postupka (jedan trapez)

- Najjednostavnija, ali ne i najbolja metoda, se sastoji u tome da se površina ispod krive aproksimira nizom trapeza



- Površina trougla na slici iznosi $P\Delta=1/2$
- Greška tako izračunate površine iznosi:

$$G = P\Delta - P[f(x)] = \frac{1}{2} - \frac{1}{3} = \frac{1}{6}$$

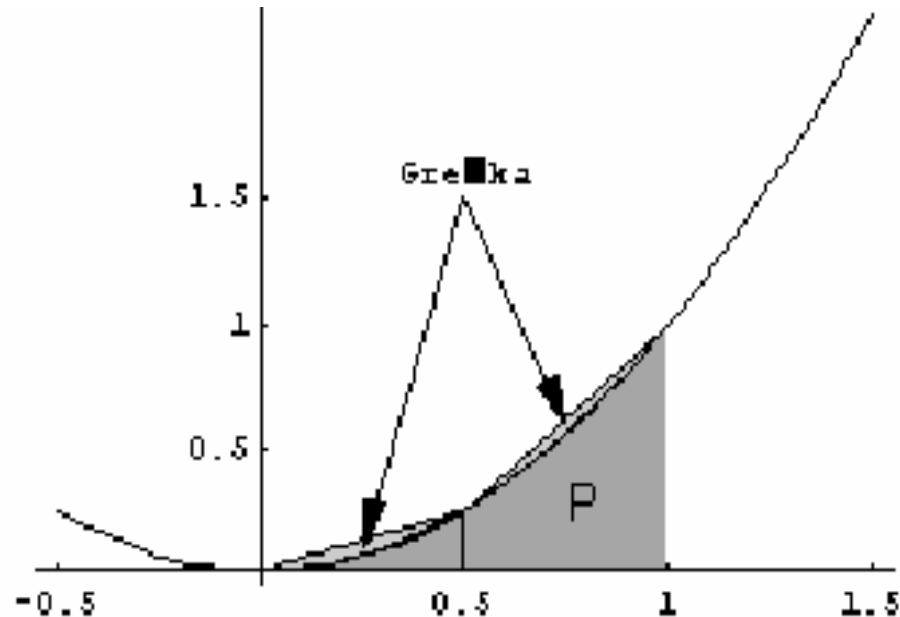
Trapezno pravilo

- nastavak postupka (dva trapeza)

- Ako se interval integracije podeli na dva jednaka dela dobija se sledeći rezultat

$$P_{trougao} = \frac{\frac{1}{2} \bullet \frac{1}{4}}{2} = \frac{1}{16}$$

$$P_{trapez} = \frac{1}{2} \left(\frac{1 + \frac{1}{4}}{2} \right) = \frac{5}{16}$$



$$P_{uk} = P_{trougao} + P_{trapez} = \frac{1}{16} + \frac{5}{16} = \frac{6}{16} = \frac{3}{8}$$

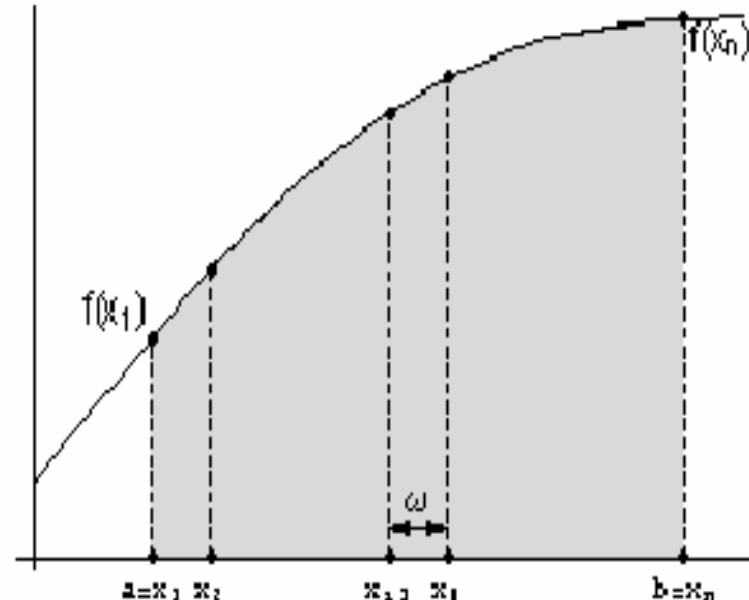
$$Greška = P_{uk} - P[f(x)] = \frac{3}{8} - \frac{1}{3} = \frac{1}{24}$$

Trapezno pravilo

- Funkcija $f(x)$ se integrira u granicama od a do b

$$I = \int_a^b f(x) dx$$

Zadati interval se deli na n jednakih delova



- Površina jednog segmenta ispod krive računa se kao:

$$A_i = \frac{\omega}{2} [f(x_{i-1}) + f(x_i)]; \quad \omega = x_i - x_{i-1} = \frac{b-a}{n}$$

gde je n broj podintervala zadatog intervala.

Trapezno pravilo

- ukupna površina trapeza

- Ukupna površina ispod krive u intervalu od a do b , gde je $a=x_0$, $b=x_n$ iznosi:

$$\begin{aligned}T_n &= \sum_{i=1}^n A_i = A_1 + A_2 + \dots + A_{n-1} + A_n = \\&= \frac{\omega}{2}[f(x_0) + f(x_1)] + \frac{\omega}{2}[f(x_1) + f(x_2)] + \dots + \frac{\omega}{2}[f(x_{n-2}) + f(x_{n-1})] + \frac{\omega}{2}[f(x_{n-1}) + f(x_n)] = \\&= \frac{\omega}{2}[f(x_0) + 2f(x_1) + 2f(x_2) + \dots + 2f(x_{n-1}) + f(x_n)]\end{aligned}$$

- Tačnost ove metode zavisi od broja trapeza (n) koji je odabran.
- Teoretski bi trebalo računati sa što više trapeza, ali iz praktičnih razloga kao što su brzina i greška zaokruživanja ne bi trebalo preterati.

Trapezno pravilo 1/2

```
# implementirati numericko resenje integrala funkcije Trapeznim pravilom
#  $y(x) = 6 - 6 * x^{**5}$ 
def f(x):
    return 6-6*x**5

def egz(a, b ):
    return (6*b-b**6-6*a+a**6)

def Simpson():
    # pretpostavka je da je ulaz korektan
    print("\tTRAPEZ\n")
    a = float(input("donja granica integracije = "))
    b = float(input("gornja granica integracije = "))
    nodiv = int(input("broj intervala = "))

    omega = (b - a) / nodiv;
    area=0;

    for j in range(1,nodiv+1):
        x1 = a + (j - 1) * omega # leva granica podintervala
        xr = a + j * omega        # desna granica podintervala
        area += (omega / 2.0) * (f(x1) + f(xr)); # azuriranje površine
```

Trapezno pravilo 2/2

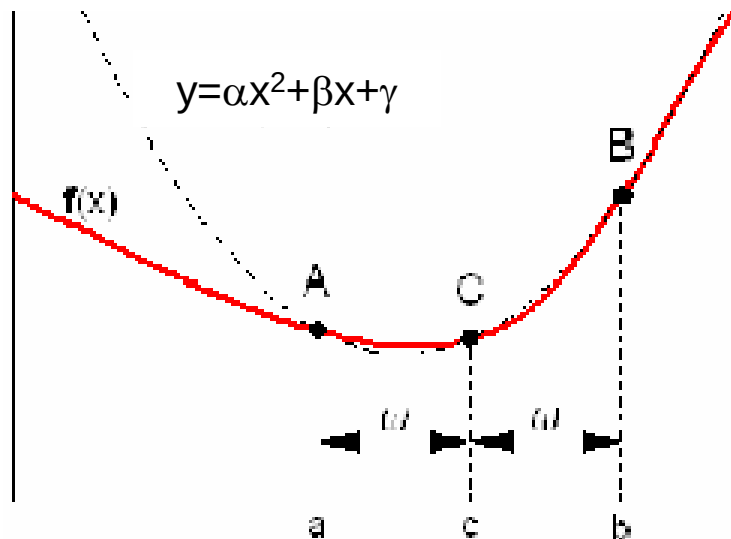
```
print('\nPovrsina: f(x)=6-pow(x,5) u intervalu od {} do  
{}`.format(a,b))  
print("Trapezno:\t\tEgzaktna:")  
print('{:<12.6f}\t{:<12.6f}'.format(area,egz(a,b)))  
  
if __name__=='__main__':Simpson()
```

TRAPEZ

```
donja granica integracije = 0  
gornja granica integracije = 1  
broj intervala             = 7
```

```
Povrsina: f(x)=6-pow(x,5) u intervalu od 0.0 do 1.0  
Trapezno:      Egzaktna:  
4.949188      5.000000
```

Simpsonova metoda integracije



- Za integraciju funkcije $f(x)$ u intervalu $[a, b]$, gde je $c = (a+b)/2$ polovište intervala $[a, b]$ dobiju se dva intervala određena sa tri tačke.

Tačkama:

$A(a, f(a));$

$B(b, f(b));$

$C(c, f(c));$

jednoznačno je određena parabola $y = \alpha x^2 + \beta x + \gamma$. Pošto je površinu ispod parabole lakše odrediti od one ispod krive $f(x)$, uzima se ta površina kao aproksimacija datog integrala.

Primena metode neodređenih koeficijenata

- Trapezno pravilo

$$Area = \frac{\omega}{2} [f(a) + f(c)]$$

odnosno, opšti oblik

$$Area = P \cdot f(a) + Q \cdot f(c)$$

gde je $P=Q=\omega/2$

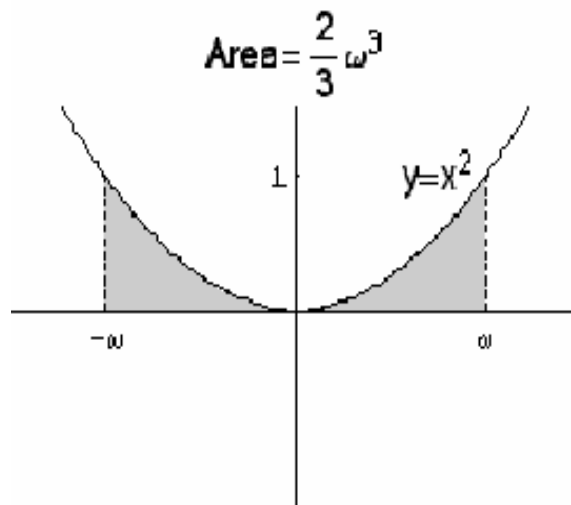
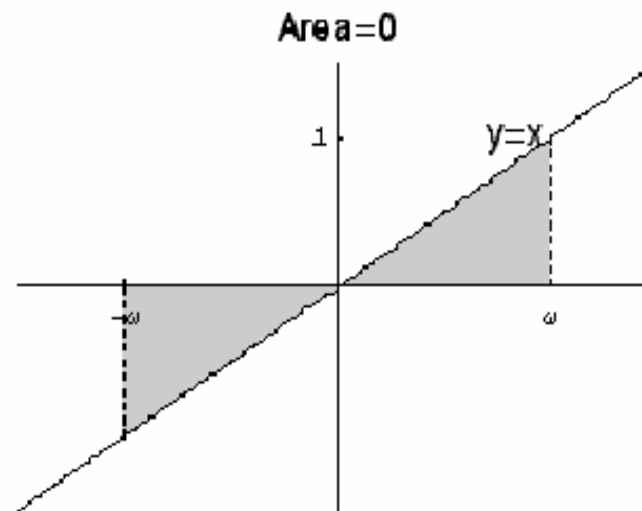
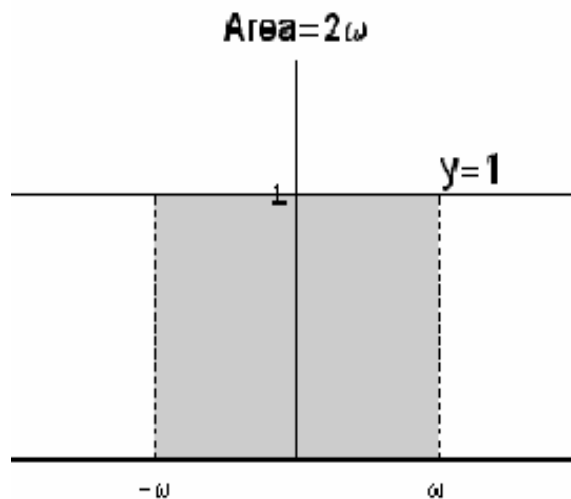
- Za slučaj sa 3 tačke formira se sledeća relacija:

$$Area = P \cdot f(a) + Q \cdot f(c) + R \cdot f(b)$$

- Da bi odredili P, Q i R upotrebljava se metoda neodređenih koeficijenata. Odaberu se tri specijalna integrala:

$$I_1 = \int_{-\omega}^{+\omega} dx = 2\omega \quad I_2 = \int_{-\omega}^{+\omega} x dx = 0 \quad I_3 = \int_{-\omega}^{+\omega} x^2 dx = \frac{2\omega^3}{3}$$

Grafička prezentacija za tri specijalna integrala



Supstitucija izraza u tri jednačine

- U svakom od prethodna tri slučaja uzima se $a=-\omega$, $c=0$, $b=\omega$.
Supstitucijom u relaciju:

$$Area = P \cdot f(a) + Q \cdot f(c) + R \cdot f(b)$$

dobija se

$$P \cdot 1 + Q \cdot 1 + R \cdot 1 = 2\omega$$

$$P \cdot (-\omega) + Q \cdot 0 + R \cdot \omega = 0$$

$$P \cdot (-\omega)^2 + Q \cdot 0^2 + R \cdot \omega^2 = \frac{2\omega^3}{3}$$

Sređivanjem dobijenih jednačina

$$P + Q + R = 2\omega \quad (1)$$

$$P \cdot (-\omega) + R \cdot \omega = 0 \quad (2)$$

$$P \cdot \omega^2 + R \cdot \omega^2 = \frac{2\omega^3}{3} \quad (3)$$

Iz relacije (2) sledi:

$$P = R$$

Kada se uvrsti u (3) dobija se:

$$P \cdot \omega^2 + P \cdot \omega^2 = \frac{2\omega^3}{3} \quad 2P \cdot \omega^2 = \frac{2\omega^3}{3}$$

$$P = \frac{\omega}{3}; R = \frac{\omega}{3}$$

Simpsonovo pravilo

- Iz (1) sledi:

$$P + Q + R = 2\omega$$

$$Q = 2\omega - P - R = 2\omega - \frac{\omega}{3} - \frac{\omega}{3}$$

$$Q = \frac{4}{3}\omega$$

dobija se Simpsonovo pravilo za izračunavanje površine ispod krive

$$Area = \frac{\omega}{3} f(a) + \frac{4\omega}{3} f(c) + \frac{\omega}{3} f(b) = \frac{\omega}{3} [f(a) + 4f(c) + f(b)]$$

- U opštem slučaju treba dati integral podeliti na paran broj podintervala. Izraz za površinu i -tog podintervala glasi:

$$A_i = \frac{\omega}{3} [f(x_{i-1}) + 4f(x_i) + f(x_{i+1})]$$

Simpsonov postupak integracije funkcije

Primenom Simpsonovog postupka izračunati vrednost integrala funkcije:

$$y(x) = 6 - 6x^5$$

u zadatom intervalu.

Ulazni parametri programa su:

- broj podela intervala
- donja granica integracije
- gornja granica integracije

Program treba da prikaže:

stvarnu vrednost integrala

i

vrednost dobijenu Simpsonovim postupkom
za date ulazne parametre.

```
# implementirati numericko resenje integrala funkcije Simpsonovom  
# metodom za date granice integracije  
#  $y(x) = 6 - 6 * x^{**5}$ 
```

```
def f(x):  
    return 6-6*x**5
```

```
def egz(a, b ):  
    return (6*b-b**6-6*a+a**6)
```

```
def Simpson():  
    # pretpostavka je da je ulaz korektan  
    print("\tSIMPSON\n")  
    a = float(input("donja granica integracije = "))  
    b = float(input("gornja granica integracije = "))  
    n = int(input("broj intervala (paran broj)= "))  
  
    area=0;  
    x=a;  
  
    dx = (b-a)/n;  
    dx2 = dx*2.0;
```

```
for i in range( n//2)):
    area += f(x)+4.0*f(x+dx)+f(x+dx2)
    x=x+dx2
```

```
area *= dx/3.0
print('\nPovrsina: f(x)=6-x**5 u intervalu od {} do {}'.format(a,b))
print("Simpson:\t\tEgzaktna:")
print('{:<12.6f}\t{:<12.6f}'.format(area,egz(a,b)))
```

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

```
# Simpson i trapezno pravilo pomocu biblioteke scipy
from scipy.integrate import simps
import numpy as np
def f(x): return 6-6*x**5
x = np.linspace(0,1,11)
y = f(x)
si = simps(y,x)
print(si)          #4.9998
tr = np.trapz(y,x)
print(tr)          #4.97505
```

Izlaz:

SIMPSON

```
donja granica integracije = 0
gornja granica integracije = 1
broj intervala (paran broj)= 10
```

Povrsina: $f(x)=6-x^5$ u intervalu od 0.0 do 1.0

```
Simpson:      Egzaktna:
4.999800      5.000000
```

SIMPSON

```
donja granica integracije = 0
gornja granica integracije = 1
broj intervala (paran broj)= 20
```

Povrsina: $f(x)=6-x^5$ u intervalu od 0.0 do 1.0

```
Simpson:      Egzaktna:
4.999988      5.000000
```

Rešavanje sistema linearnih jednačina : Gauss-Jordanov metod

- Gauss-Jordanova metoda je direktna metoda rešavanja sistema linearnih jednačina. Kao takva pogodna je za rešavanje sistema sa manjim brojem jednačina.
- Sistem n linearnih jednačina sa n nepoznatih može se napisati kao:

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1,$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2,$$

.

.

.

$$a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n.$$

Rešenje ovog sistema je n -torka $(x_1, x_2, \dots, x_n)$ koja zadovoljava sve jednačine.

Rešavanje sistema linearnih jednačina

Gaus-Jordanov metod

Sistem jednačina (1) može se još napisati i kao:

$$\mathbf{Ax} = \mathbf{b}, (2)$$

gdje je $\mathbf{A}$ matrica koficijenata sistema, $\mathbf{b}$ vektor desne strane sistema, a $\mathbf{x}$ vektor rešenja sistema.

Gauss-Jordanova metoda koristi elementarne transformacije matrice, a to su:

- Množenje svih elemenata jednog reda skalarom,
- Zamena dva reda,
- Dodavanje reda pomnoženog sa skalarom drugom redu.

Radi jednostavnosti prikazuje se rešavanje sistema linearnih jednačina na primeru:

$$3x_1 + 2x_2 + x_3 = 10$$

$$x_1 - x_2 + 2x_3 = 5$$

$$2x_1 - 3x_2 - 2x_3 = -10$$

Rešavanje sistema linearnih jednačina :

Gaus-Jordanov metod

- Matrica preko koje rešavamo sistem izgleda ovako :

$$\left| \begin{array}{ccc|c|ccc} 3 & 2 & 1 & 10 & 1 & 0 & 0 \\ 1 & -1 & 2 & 5 & 0 & 1 & 0 \\ 2 & -3 & -2 & -10 & 0 & 0 & 1 \end{array} \right|$$

- U prvom koraku normalizuje se prvi red deljenjem istog sa 3, pa se ostali elementi prve kolone reduciraju na nulu tako da se doda novi prvi red pomnožen sa -1 drugom redu, a pomnožen sa -2 trećem redu.

- Rezultat ove operacije je:
$$\left| \begin{array}{ccc|c|ccc} 1 & \frac{2}{3} & \frac{1}{3} & \frac{10}{3} & \frac{1}{3} & 0 & 0 \\ 0 & -\frac{5}{3} & \frac{5}{3} & \frac{5}{3} & -\frac{1}{3} & 1 & 0 \\ 0 & -\frac{13}{3} & -\frac{8}{3} & -\frac{50}{3} & -\frac{2}{3} & 0 & 1 \end{array} \right|$$

Rešavanje sistema linearnih jednačina :

Gaus-Jordanov metod

- Nakon toga normalizuje se drugi red deljenjem istog sa $-5/3$, i reduciraju se ostali elementi druge kolone na nulu dodavanjem novog drugog reda pomnoženog sa $-2/3$ prvom redu, a pomnoženog sa $13/3$ trećem redu.
- Rezultat drugog koraka je:

$$\left| \begin{array}{ccc|c|ccc} 1 & 0 & 1 & 4 & \frac{7}{15} & -\frac{10}{9} & 0 \\ 0 & 1 & -1 & -1 & \frac{1}{5} & -\frac{5}{3} & 0 \\ 0 & 0 & -7 & -\frac{63}{3} & \frac{1}{5} & -\frac{65}{9} & 1 \end{array} \right|$$

- Na kraju normalizujemo treći red deleći ga sa $-1/7$, i reduciramo ostale elemente treće kolone tako da dodamo novi treći red pomnožen sa -1 prvom, a pomnožen sa 1 drugom redu.

$$\left| \begin{array}{ccc|c|ccc} 1 & 0 & 0 & 1 & \frac{52}{105} & -\frac{135}{63} & \frac{1}{7} \\ 0 & 1 & 0 & 2 & \frac{6}{35} & -\frac{40}{63} & -\frac{1}{7} \\ 0 & 0 & 1 & 3 & -\frac{1}{35} & \frac{65}{63} & -\frac{1}{7} \end{array} \right|$$

Rešavanje sistema linearnih jednačina :

Gaus-Jordanov metod

- Iz gornje matrice možemo pročitati rešenje odnosno n-torku (1,2,3), kao i inverznu matricu

$$A^{-1} = \begin{vmatrix} \frac{52}{105} & -\frac{135}{63} & \frac{1}{7} \\ \frac{6}{35} & -\frac{40}{63} & -\frac{1}{7} \\ -\frac{1}{35} & \frac{65}{63} & -\frac{1}{7} \end{vmatrix}$$

Primenom ove metode može doći do sledećih grešaka:

- Ukoliko je neki pivot element sa kojim se deli jednak nuli, tada će metoda javiti grešku da je sistem singularan.
 - Sistem nije singularan i jednostavnom zamenom redova se može doći do rešenja korišćenjem iste metode.
- Ukoliko je sistem skoro singularan, akumuliranje greške zaokruživanja može dovesti sistem u singularan. Tada metoda javlja grešku.
- Metoda usled greške zaokruživanja može kao rešenje vratiti potuno pogrešne vrednosti. Verovatnoća takve greške raste sa brojem jednačina u sistemu, kao i sa približavanjem sistema singularnom.

Rešavanje sistema linearnih jednačina: Gaus-Jordanov metod

- Primer prve greške:

$$-7x_2 + 4x_3 = 1$$

$$-3x_1 + 8x_2 + 5x_3 = 6$$

$$x_1 + 9x_2 - 6x_3 = 1$$

- Ukoliko se zamene prva dva reda sistem je moguće rešiti ovom metodom:

$$-3x_1 + 8x_2 + 5x_3 = 6$$

$$-7x_2 + 4x_3 = 1$$

$$x_1 + 9x_2 - 6x_3 = 1$$

- Gauss-Jordanova metoda nije pogodna za korišćenje u komercijalne svrhe, kako zbog gore navedenih problema tako i zbog sporosti ove metode.
 - Metoda je direktna, stabilna i razumljiva te se može koristiti kao kontrola neke druge metode. Takođe je dobra za pedagoške namene.

Gaus-Jordanova metoda rešavanja sistema jednačina

Primenom Gaus-Jordanove metode rešiti sistem jednačina:

$$3x_1 + 2x_2 + x_3 = 10$$

$$x_1 - x_2 + 2x_3 = 5$$

$$2x_1 - 3x_2 - 2x_3 = -10$$

Ulazi u program:
unos sistema jednačina

Izlazi iz programa:
rešenje unesenog sistema jednačina

```
a = [  
    [3.0, 2.0, 1.0, 10.0, 1.0, 0.0, 0.0],  
    [1.0, -1.0, 2.0, 5.0, 0.0, 1.0, 0.0],  
    [2.0, -3.0, -2.0, -10.0, 0.0, 0.0, 1.0]  
]  
  
def ispisi(n, m):  
    for i in range(n):  
        for j in range(n+m):  
            print('{:8.2f}'.format(a[i][j]), end=' ')  
        print("")  
    print("");  
  
def main():  
    m = 1+3;  
    n = 3;  
    ispisi(n,m); # mogli bi dodati da se matrica a unosi, samo 3x3  
    eps = 0.00001;  
    #GausJordan  
    for k in range(n):  
        #da li je pivot element premali  
        if (abs(a[k][k])<= eps):  
            print("Mala vrednost pivot-a!")  
            exit(0)
```

```
for j in range (k+1,n+m):#normalizacija pivot reda
    a[k][j] = a[k][j] / a[k][k]
a[k][k] = 1;
print("a[k][k]=a[%d][%d]=1, normalizacija reda %d" % (k,k,k))
ispisi(n,m);
#eliminacija k-tog elementa osim pivota
for i in range(n):
    if (i==k or a[i][k]==0):
        print("(i==k || a[i][k]==0), a[i][k]=a[%d][%d]=%6.2lf,"
continue" % (i,k,a[i][k]))
        ispisi(n,m)
        continue
    for j in range(k+1,n+m):
        a[i][j] = a[i][j] - a[i][k] * a[k][j];
        print("a[i][j]=a[%d][%d]=%6.2lf racunanje a[i][j]-
=a[i][k]*a[k][j],k=%d" % (i,j,a[i][j],k))
        ispisi(n,m)
    a[i][k] = 0;
    print("a[i][k]=a[%d][%d]=0, postavljanje nule" % (i,k))
    ispisi(n,m)
    print("sledeca petlja, k=%d\n" % (k+1))
    ispisi(n,m)
if __name__ == '__main__': main()
```

- Print funkcije u prethodnom kodu ostavite nekomentarisane ako želite da vidite sve korake algoritma za dati ulaz.
- Poslednji korak (ispisi(n,m)) daje sledeći rezultat:

1.00	0.00	0.00	1.00	0.23	0.03	0.14
0.00	1.00	0.00	2.00	0.17	-0.23	-0.14
0.00	0.00	1.00	3.00	-0.03	0.37	-0.14
- Deo matrice rezultata od 0,0 do 2,2 predstavlja jediničnu matricu
- Deo matrice rezultata od 0,3 do 2,3 predstavlja rešenje sistema jednačina
- Deo matrice rezultata od 0,4 do 2,6 predstavlja inverznu matricu, početne matrice na pozicijama od 0,0 do 2,2

Interpolacija

- Interpolacija je proces koji ima brojne primene među kojima su najznačajnije računarska grafika i numerička analiza.
- Neki primeri kada je potrebna interpolacija mogu biti:
 - Poznat je samo određen broj vrednosti neke funkcije i potrebno je pronaći vrednosti u drugim tačkama.
 - Ponekad je vremenski zahtevno izračunavanje kompleksne (ali glatke) funkcije više puta. Tada se može koristiti polinom da se aproksimira funkcija što će smanjiti vreme računanja.
- Problem koji se rešava je:
za zadate vrednosti $(t_i, y_i), \quad i = 1, \dots, m, \quad t_1 < t_2 < \dots < t_m$
odrediti funkciju $f(t_i) = y_i, \quad i = 1, \dots, m$
tako da važi: f je *interpolaciona* funkcija ili *interpolant*
- Pored vrednosti funkcije mogu se zadati i dodatni podaci (vrednosti izvoda) oskulatorne interpolacije
- Takođe se mogu postaviti i dodatna ograničenja (monotonost, glatkost, konveksnost,...) za interpolacionu funkciju

APROKSIMACIJA ALGEBARSKIM POLINOMIMA

Najčešća aproksimacija

Baza: $\phi_j(t) = t^{j-1}, \quad j = 1, \dots, n,$

Interpolacioni polinom: $p_{n-1}(t) = x_1 + x_2 t + \dots + x_n t^{n-1}$

Sistem za određivanje parametara (A – Vandermonodova matrica)

$$Ax = \begin{bmatrix} 1 & t_1 & \cdots & t_1^{n-1} \\ 1 & t_2 & \cdots & t_2^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & t_n & \cdots & t_n^{n-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = y$$

APROKSIMACIJA ALGEBARSKIM POLINOMIMA

PRIMER

Ulazni podaci: $(t_1, y_1), (t_2, y_2), (t_3, y_3)$
 $(-2, -27), (0, -1), (1, 0)$

$$Ax = \begin{bmatrix} 1 & t_1 & t_1^2 \\ 1 & t_2 & t_2^2 \\ 1 & t_3 & t_3^2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = y$$

$$\begin{bmatrix} 1 & -2 & 4 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -27 \\ -1 \\ 0 \end{bmatrix}$$

$$x = [-1 \quad 5 \quad -4]^T \longrightarrow p_2(t) = -1 + 5t - 4t^2$$

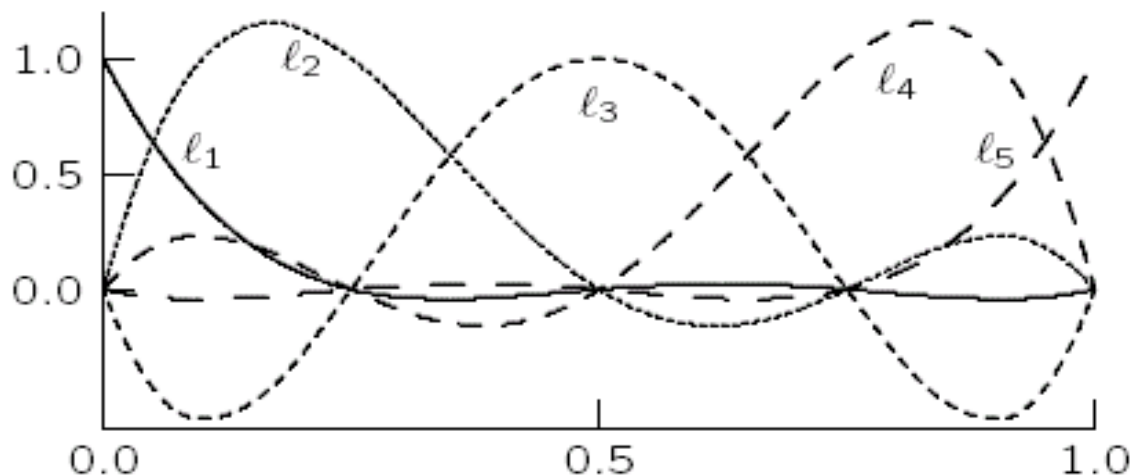
LAGRANŽOVA INTERPOLACIJA

Za skup tačaka (t_i, y_i) , $i = 1, \dots, n$ Lagranžova baza je:

$$\ell_j(t) = \prod_{k=1, k \neq j}^n (t - t_k) / \prod_{k=1, k \neq j}^n (t_j - t_k), \quad j = 1, \dots, n$$

Važi:

$$\ell_j(t_i) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}, \quad i, j = 1, \dots, n$$



Oblik interpolacionog polinoma je:

$$p_{n-1}(t) = y_1 \ell_1(t) + y_2 \ell_2(t) + \dots + y_n \ell_n(t)$$

LAGRANŽOVA INTERPOLACIJA - PRIMER

$$(t_1, y_1), (t_2, y_2), (t_3, y_3)$$

$$(-2, -27), (0, -1), (1, 0)$$

$$p_2(t) = y_1 \frac{(t - t_2)(t - t_3)}{(t_1 - t_2)(t_1 - t_3)} + y_2 \frac{(t - t_1)(t - t_3)}{(t_2 - t_1)(t_2 - t_3)} \\ + y_3 \frac{(t - t_1)(t - t_2)}{(t_3 - t_1)(t_3 - t_2)}$$

$$p_2(t) = -27 \frac{t(t - 1)}{(-2)(-2 - 1)} + (-1) \frac{(t + 2)(t - 1)}{(2)(-1)}$$

LAGRANŽOVA INTERPOLACIJA

```
def main():
    points = int(input("Unesite broj ulaznih tacaka = "))
    x = []
    y = []
    print("Unesite koordinate ulaznih tacaka !");
    for i in range(points):
        x.append(float(input('x{:02d} = '.format(i))))
        y.append(float(input('y{:02d} = '.format(i))))
        print("----->(x,y)=(",x[i],",",y[i],")")
    t = float(input("\nUnesite za koji x zelite interpolaciju f(x), x= "))

    total = 0.0
    for j in range(points):
        temp = 1.0
        for k in range(points):
            if (k != j ):
                temp = temp * (t - x[k]) / (x[j] - x[k])
        temp *= y[j]
        total += temp
    print('\nf(x) = f({}) = {}'.format(t,total))

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

LAGRANŽOVA INTERPOLACIJA

Izlaz:

```
Unesite broj ulaznih tacaka = 3
Unesite koordinate ulaznih tacaka !
x00 =  -2
y00 = -27
----->(x,y)=( -2.0 , -27.0 )
x01 =  0
y01 = -1
----->(x,y)=( 0.0 , -1.0 )
x02 =  1
y02 =  0
----->(x,y)=( 1.0 , 0.0 )
```

Unesite za koji x zelite interpolaciju $f(x)$, $x=$ -1

$f(x) = f(-1.0) = -10.0$

NJUTNOVA INTERPOLACIJA

Za skup parova (t_i, y_i) , $i = 1, \dots, n$ Njutnova baza je

$$\pi_j(t) = \prod_{k=1}^{j-1} (t - t_k), \quad j = 1, \dots, n$$

Polinom je:

$$\begin{aligned} p_{n-1}(t) = & x_1 + x_2(t - t_1) + x_3(t - t_1)(t - t_2) + \dots \\ & + x_n(t - t_1)(t - t_2) \dots (t - t_{n-1}) \end{aligned}$$

NJUTNOVA INTERPOLACIJA - PRIMER

$$(t_1, y_1), (t_2, y_2), (t_3, y_3) \quad (-2, -27), (0, -1), (1, 0)$$

Sa Njutnovom bazom sistem je:

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & t_2 - t_1 & 0 \\ 1 & t_3 - t_1 & (t_3 - t_1)(t_3 - t_2) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

Za zadate tačke je:

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 2 & 0 \\ 1 & 3 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -27 \\ -1 \\ 0 \end{bmatrix} \longrightarrow x = [-27 \quad 13 \quad -4]^T$$

$$p(t) = -27 + 13(t + 2) - 4(t + 2)t$$