8 — Python grafika

8.1 Tkinter

Python ima više programskih paketa koji nude programsku pomoć za stvaranje korisničkog grafičkog sučelje - GUI (*graphical user interfaces*. Osnovni, koji dolazi zajedno s Python distribucijom je **Tkinter**. On je jednostavniji od *wxPython*-a i *JPython*-a koji nude veće mogućnosti (prvi preko Pythona, a drugi preko Jave). Ima još puno takvih alata, ali ako se shvate načela jednostavnog, lako se prelazi na složenije.

Tkinter ima standardnu GUI knjižnicu za Python. Stvaranje GUI aplikacije je jednostavna zadaća, koja se izvodi u par koraka:

1. učitati Tkinter modul
2. načiniti glavni prozor za GUI aplikaciju
3. dodati jedan ili više grafičkih modula, tzv widgets-a u glavni prozor
4. načiniti glavnu petlju koja reagira na sve korisnikove akcije (preko miša ili tipkovnice)

Program 8.1 Osnovni Tkinter program

```
1 import Tkinter
2 top = Tkinter.Tk()
3 # Ovdje se dodaje kod widgets-a...
4 top.mainloop()
```

To će na zaslonu računala otvoriti prazan okvir:

Tkinter nudi različite upravljačke module, kao što su gumbovi (eng. *buttons*), oznake (eng. *labels*), tekstne okvire (eng. *text boxes*) i slično, koje zovemo *widgets*-ima. Do sad je razvijeno 15 tipova widgets-a u Tkinter-u, koji su ukratko opisani u sljedećoj tablici:

Svi Tkinter wingets-i imaju pristup specifičnim geometrijskim metodama, s kojima se organiziraju widgets-i s obzirom na prvotno (roditeljsko) spremište. Postoje 3 metode koje se mogu koristiti:

1. *Pack metoda* - ovo geometrijsko upravljanje organizira widgets-e u blokove prije nego ih stavi u roditeljski widget
2. *Grid metoda* - ovo geometrijsko upravljanje organizira widgets-e u strukturi sličnoj tablici u roditeljskom widget-u

Slika 8.1: Tkinter prozor

Ime widgets-a	**Namjena**
`Button`	prikazuje gumb koji se spaja s nekom akcijom
`Canvas`	prostor za crtanje grafičkih oblika
`Checkbutton`	služi za prikaz više opcija izmedju kojih korisnik može izabrati jednu ili više
`Entry`	prikazuje 'kućicu' - obrazac za unos korisničkog teksta
`Frame`	okvir (eng. container) za prihvat drugih widgets-a
`Label`	tekstni opis ili slika u jednom retku za opis drugih widgets-a
`Listbox`	lista korisničkih ponuda
`Menubutton`	služi za prikaz izbornih ponuda u aplikaciji
`Menu`	izborne ponude s različitim naredbama koje su sadržane unutar Menu gumba.
`Message`	ispis višelinijskih poruka korisniku
`Radiobutton`	koristi se za prikaz više opcija od kojih korisnik može izabrati samo jednu
`Scale`	upravlja analognom vrpcom (eng. *slider*-om)
`Scrollbar`	upravlja vertikalnim pomakom (eng. scroll), npr. kod lista
`Text`	ispis teksta u više linija
`Toplevel`	služi za vizualno odvajanje više spremišta
`Spinbox`	jedna varijanta Entry-a, kojom se izabire između fiksnog broja vrijednosti
`PanedWindow`	spremište koje može imati bilo koji broj hor. ili vert. podspremišta
`LabelFrame`	jednostavno spremište unutar složenih
`tkMessageBox`	služi za ispis poruke u aplikaciji

Tablica 8.1: Funkcije u *Tkinter* paketu

3. *Mjesna (eng. place) metoda* - ovo geometrijsko upravljanje postavlja widgets-e na specifičnu poziciju u roditeljski widget

Svaka opisana metoda ima jedan ili više atributa i bilo bi prezahtjevno (i nepotrebno) opisivati svakog pojedinačno. Zato je u sljedećoj tablici načinjen pokušaj da se osnovni atributi ujedine, s obzirom na kategorije koje ih ujedinjuju. Svaki argument po imenu, već govori o svojoj ulozi. Detalje treba potražiti u *Tkinter* priručniku.

Atributi se navode na standardan način, u obliku parova '*atribut=vrijednost*', na primjer: *text ="plus"*, *relief=RAISED*, *cursor="plus"* i slično.

Kategorija	Atributi
`Dimensions`	borderwidth, highlightthickness, padX, padY, height, width, ...
`Colors`	background, foreground, highlightcolor,...
`Fonts`	family, size, weight, slant, underline
`Anchors`	NW, N, NE, W, CENTER, E, SW, S, SE
`Relief styles`	FLAT, RAISED, SUNKEN, GROOVE, RIDGE
`Bitmaps`	"error", "gray75", "hourglass", "info", "question", "warning", ...
`Cursors`	"arrow", "circle", "clock", "cross", "dotbox", ...

Tablica 8.2: Standardni atributi *Tkinter* funkcija

8.1.1 Tkinter objekti

Ne ulazeći u detalje, u nastavku će bit pokazano nekoliko Tkinter funkcija i pripadnh kratkih programskih odsječaka s kojima se one lako razumiju.

Frame widget je vrlo važan za proces povezivanja i organiziranja drugih widgets-a na lak način. On radi kao spremište (eng. *container*) i prima na sebe druge widgets-e.

Njegova sintaksa je:

```
w = Frame (glavni, opcije, ...  )
```

gdje argument *glavni*, odgovara roditeljskom widgets-u, a *opcije* nekom od parametara dimenzija, boje i slično.

Program 8.2 Tkinter program koji opisuje *Frame* funkciju

```
1  from Tkinter import *
2
3  root = Tk()
4  frame = Frame(root)
5  frame.pack()
6
7  bottomframe = Frame(root)
8  bottomframe.pack( side = BOTTOM )
9
10 redbutton = Button(frame, text="Red", fg="red")
11 redbutton.pack( side = LEFT)
12
13 greenbutton = Button(frame, text="Brown", fg="brown")
14 greenbutton.pack( side = LEFT )
15
16 bluebutton = Button(frame, text="Blue", fg="blue")
17 bluebutton.pack( side = LEFT )
18
19 blackbutton = Button(bottomframe, text="Black", fg="black")
20 blackbutton.pack( side = BOTTOM)
21
22 root.mainloop()
```

Primjetite stvaranja Button (gumb) objekta i njegovo pozicioniranje na Frame spremište. Također je vidljiva 'pack' metoda kojom se svi gumbi uglavljuju u okvir. Izvodjenjem programa

dobit ćemo sljedeći prozor s obojenim gumbima u definiranom nizu:

Slika 8.2: Tkinter prozor

U idućem programu pokazuje se *tkMessageBox* objekt koji ima *showinfo* metodu za prikaz poruke. Objekt je korišten u *hello()* korisničkoj funkciji koja se poziva iz *Button* objekta.

Program 8.3 Tkinter program koji opisuje *tkMessageBox* funkciju

```
1  import Tkinter
2  import tkMessageBox
3
4  top = Tkinter.Tk()
5  def hello():
6     tkMessageBox.showinfo("Say␣Hello", "Hello␣World")
7
8  B1 = Tkinter.Button(top, text = "Say␣Hello", command = hello)
9  B1.pack()
10
11 top.mainloop()
```

Posebno važno je vidjeti povezivanje nekog Tkinter objekta (u ovom slučaju *Button*-a) s nekom korisničkom funkcijom. Za to nam služi atribut command, pa u našem slučaju *command=hello* čini da se klikom na gumb izvrši funkcija *hello()*, kako je prikazano slikom:

Slika 8.3: Izvodjenje korisničke funkcije unutar Tkinter objekta

8.1.2 Primjeri

U nastavku su pokazana tri primjera, za druge Tkinter objekte. Oni ne donose novu informaciju, nego proširuju i potvrđuju postojeću.

U prvom primjeru uključen je Radiobutton objekt, koji omogućuje korisniku jedan izbor izmedju više mogućnosti. Osluškivanje događaja lijevog klika miša od strane korisnika, obavlja se u objektu bez potrebe pisanja dodatnih funkcija, što olakšava upotrebu.

Program 8.4 Tkinter program koji opisuje *Radiobutton* funkciju

```
from Tkinter import *

def sel():
    selection = "You selected the option " + str(var.get())
    label.config(text = selection)

root = Tk()
var = IntVar()
R1 = Radiobutton(root, text="Option 1", variable=var, value=1,
                 command=sel)
R1.pack( anchor = W )

R2 = Radiobutton(root, text="Option 2", variable=var, value=2,
                 command=sel)
R2.pack( anchor = W )

R3 = Radiobutton(root, text="Option 3", variable=var, value=3,
                 command=sel)
R3.pack( anchor = W)

label = Label(root)
label.pack()
root.mainloop()
```

Primjetite upotrebu cjelobrojne varijable kojom se odabrana mogućnost pridružuje broju, s kojim se onda lako izvode grananja (ako → onda) u programu.

Rezultat pokretanja programa bit će:

Slika 8.4: Radiobutton

Na sličan način moguće je ugraditi složene izborne ponude (menu-e) u vizualno sučelje korisničkog programa:

Program 8.5 Tkinter program koji opisuje *Menu* funkciju

```
from Tkinter import *
def donothing():
    filewin = Toplevel(root)
    button = Button(filewin, text="Do nothing button")
    button.pack()
```

```
6
7  root = Tk()
8  menubar = Menu(root)
9  filemenu = Menu(menubar, tearoff=0)
10 filemenu.add_command(label="New", command=donothing)
11 filemenu.add_command(label="Open", command=donothing)
12 filemenu.add_command(label="Save", command=donothing)
13 filemenu.add_command(label="Save as...", command=donothing)
14 filemenu.add_command(label="Close", command=donothing)
15
16 filemenu.add_separator()
17
18 filemenu.add_command(label="Exit", command=root.quit)
19 menubar.add_cascade(label="File", menu=filemenu)
20 editmenu = Menu(menubar, tearoff=0)
21 editmenu.add_command(label="Undo", command=donothing)
22
23 editmenu.add_separator()
24
25 editmenu.add_command(label="Cut", command=donothing)
26 editmenu.add_command(label="Copy", command=donothing)
27 editmenu.add_command(label="Paste", command=donothing)
28 editmenu.add_command(label="Delete", command=donothing)
29 editmenu.add_command(label="Select All", command=donothing)
30
31 menubar.add_cascade(label="Edit", menu=editmenu)
32 helpmenu = Menu(menubar, tearoff=0)
33 helpmenu.add_command(label="Help Index", command=donothing)
34 helpmenu.add_command(label="About...", command=donothing)
35 menubar.add_cascade(label="Help", menu=helpmenu)
36
37 root.config(menu=menubar)
38 root.mainloop()
```

U ovom slučaju izvodi se bezlična korisnička funkcija (*donothing*)koja daje samo gumb na kojim se pritiskom ne događa ništa. Dakako, svakoj mogućoj izbornoj ponudi u uobičajenom radu se pridružuje adekvatna funkcija, koja željeno izvodi.

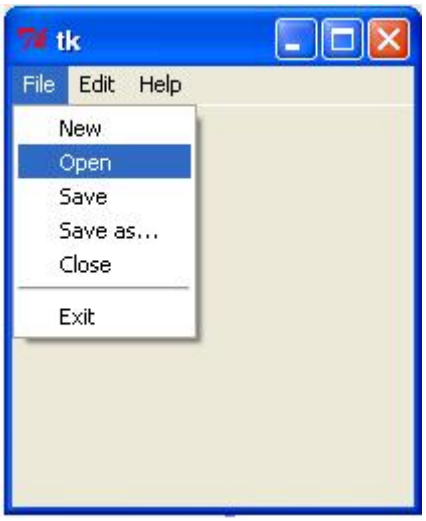
Primjer grafičkog sučelja gornjeg programskog odsječka bio bi:

8.2 Pygame

Pygame paket sadrži više modula/podpaketa (vidi tablicu 8.1.) koji su dostupni u korisničkom programu nakon što se paket učita (*import pygame*).

Svaki od podpaketa ima niz ugradjenih metoda i podatčanih atributa što je spremljeno i dokumentirano u referencijskom priručniku paketa (. U nastavku pokazat će se samo temeljne metoda i atributi nekoliko paketa uz pomoć kojih će biti prikazano par jednostavnijih primjera, ali dostatnih za razumijevanje načina pisanja grafičkih i animacijskih programa, pa i računalnih igrica, šireg dometa.

Tako se iz podpaketa 'display' može koristiti funkcija set_mode((width, height))koja kao



Slika 8.5: Tkinter menu

argument ima cjelobrojni par za širinu i visinu prozora u kojem će se nešto prikazati. Da bi se to ostvarilo potrebno je pozvati funkciju flip().

Program 8.6 Program pokazuje inicijalnu strukturu

```
1  import pygame
2  (width, height) = (400, 250)
3  screen = pygame.display.set_mode((width, height))
4  pygame.display.flip()
```

Pozivom programa 8.1 pojavio bi na na zaslonu računala jedan prozor dimenzija 400x250 točkica i automatski nestao. Da se to ne bi dogodilo program mora ući u beskonačnu petlju, npr.

Zaslon 8.1

```
while True:
  pass
```

koja neće nikad završiti (čak ni klikom na 'x' oznaku u okviru prozora), što je sad novi problem, jer program moramo nasilno prekidati.

Želi li se to izbjeći, potrebno je uključiti modul za osluškivanje događaja (eng. *event*) koji prati sve akcije korisnika programa, npr. pritisak tipke na mišu ili na tipkovnici.

Program 8.7 Najjednostavniji *pygame* program

```
1  running = True
2  while running:
3  for event in pygame.event.get():
4    if event.type == pygame.QUIT:
5      pygame.quit()
```

Prikazanom prozoru moguće je mijenjati svojstva, npr. dodati naslov:
`pygame.display.set_caption('Tutorial 1')`

Ime modula	Namjena
pygame.cdrom	pristupa i upravlja s CD napravom
pygame.cursors	puni slikom kazalo (cursor)
pygame.display	pristupa zaslonu
pygame.draw	crta oblike, crte i točke
pygame.event	upravlja vanjskim događajima
pygame.font	koristi sistemske fontove
pygame.image	radi sa slikama
pygame.joystick	radi s igraćom palicom i sličnim napravama
pygame.key	čita kodove pritisnutih tipki na tipkovnici
pygame.mixer	dohvaća, izvodi i upravlja zvukom
pygame.mouse	upravlja radom miša
pygame.movie	upravlja video datotekama
pygame.mixer	dohvaća, izvodi i upravlja zvukom
pygame.music	radi s glazbom i audio tijekom (radio)
pygame.overlay	dohvaća napredna video preklapanja
pygame.rect	upravlja pravokutnim površinama
pygame.sndarray	upravlja zvukovnim podacima
pygame.sprite	upravlja kretajućim slikama
pygame.surface	upravlja slikama i zaslonom
pygame.surfarray	upravlja točkama slikovnih podataka
pygame.time	upravlja vremenom i brzinom izmjene okvira
pygame.transform	mijenja veličinu i pomak slika

Tablica 8.3: Moduli u **Pygame** paketu

ili boju pozadine:

```
background_colour = (255,255,255) ## bijela boja
screen.fill(background_colour)
```

mijenjajući bilo koji broj u RGB trojcu od 1-255. Promjena se dakako mora načiniti prije primjene flip() funkcije koja prozor prikazuje.

8.2.1 Crtanje s Pygame modulom

U draw Pygame podmodulu ima više različitih funkcija za crtanje jednostavnih geometrijskih oblika, kao što su linije, elipse, trokuti, mnogokuti i slično (pogledajte tablicu . . .). Redovito te funkcije imaju argumente koji opisuju željeni geometrijski oblik, mjesto (koordinate) na kojem će se u prozoru nacrtati, boju i debljinu linije u kojoj će bit orisani i specifičnosti (npr. polumjer kružnice, ako se crta kružnica, ili dužina i širina, ako se crta pravokutnik i slično).

Koordinate ishodišta (x=0 i y=0) nalaze se u lijevom gornjem kutu prikazanog prozora, a x-os u točkama (eng. pixel) raste udesno, a y-os prema dnu prozora. Tako će koordinata (150, 100) bit na udaljnosti 150 točkica od lijevog ruba i 100 točkica od vrha prema dnu prikazanog, prethodno stvorenog pygame prozora. Na toj će se koordinati prozora okvir nacrtati kružnica polumjera 20 točica i debljine linije od 3 točkice u plavoj (0,0,255) boji, uz pomoć ove naredbe:

```
pygame.draw.circle(okvir, (0,0,255), (150, 100), 20, 3)
```

Crtanje objekata, npr. fizikalnih čestica ili različitih figurica u računalnim igrama, ostvaruje se definiranjem klase, pa potom niza objekata te klase. Objekti se najčešće postavljaju u listu, nad kojom se onda u koracima izvode zadani algoritmi nad svakim elementom liste. Tako će npr.

klasa za neku česticu imati inicijalizaciju (__init__) s osnovnim paramterima za prikaz i funkciju (Prikazi) kojom se to ostvaruje.

Program 8.8 Razredi čestica

```
1  class Cestica:
2    def __init__(self, (x, y), size, (r,g,b), debljina):
3      self.x = x
4      self.y = y
5      self.size = size
6      self.colour = (r,g,b)
7      self.thickness = debljina
8      def Prikazi(self):
9        pygame.draw.circle(screen, self.colour, (self.x, self.y),
10       self.size, self.thickness)
```

Od ovako definirane klase lako se stvaraju objekti, pojedinačni:

Zaslon 8.2

```
jedna_cestica = Cestica ((250, 100), 15, (0,0,255), 3)
jedna_cestica.Prikazi()
```

ili mnogobrojni:

Program 8.9 Stvaranje čestica

```
1  broj_cestica = 100
2  skup_cestica = []
3
4  for n in range(broj_cestica):
5      size = random.randint(10, 20)
6      x = random.randint(size, sirina - size)
7      y = random.randint(size, visina - size)
8      deb = random.randint(1,5)
9      skup_cestica.append(Cestica((x, y), size, (255,0,0), deb))
```

U ovom primjeru, spremit će se 100 čestica kružnog oblika, debljine linije od 1 do 5 točkica, polumjera od 10 do 20 točkica, sa slučajno odabranim koordinatama u prozoru dimenzija (širina, visina). Slučajni izbor ostvaren je pozivom funkcije randint() koja se nalazi u modulu random. Osim *randint()* ovaj modul ima i druge korisne funkcije za generiranje slučajnih brojeva (kako pokazuje tablica ....):

- *random()* - returns a random number between 0.0 and 1.0
- *uniform(a, b)* - returns a random number between a and b
- *randint(a, b)* - returns a random integer between a and b
- *choice(list)* - returns a random element from the list
- *shuffle(list)* - rearranges the order of the list

Nakon što su objekti spremljeni u listu, potrebno je samo pokrenuti petlju za njihov prikaz:

Program 8.10 Pokretanje programa - prikaz čestica

```
1 for cestice in skup_cestica:
2     cestice.Prikaz()
```

Osim prikaza, dakako, na svakoj čestici bi se mogao očitovati i bilo koji drugi algoritam, npr. interakcija čestica koje su u blizini, ili djelovanje svake čestice na svaku. Takve akcije se takodjer umeću u gornju petlju, zamišljeno se izvrši, novi raspored čestica spremi u listu (npr. promijeni im se pozicija (x,y) ili veličina (size) ili neko drugo svojstvo zbog međudjelovanja) i onda se ponovo nacrta novonastala situacija, pa opet izračuna iz novih pozicija, pa prikaže i tako u nedogled ili se osluškuje neki događaj za prekid iteracija.

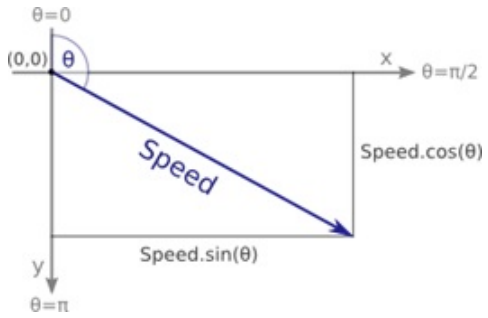
8.2.2 Predstavljanje pomaka

Intuitivni i najjednostavniji način pomaka neke točke po ekranu je dodavanje stanovitog broja točaka (dx i dy) na x i y koordinatu i novog prikaza nakon promjene koordinate. Drugi, način je nešto složeniji, ali djelotvorniji: dodavanje svojstava brzine i kuta objektu koji se pomiče. Ti atributi predstavljaju vektor s promjenljivom brzinom i smjerom i to pomaka neke točke ili skupa točaka nekog objekta koji se istodobno ili pojedinačno mogu mijenjati.

Zaslon 8.3

```
self.speed = 0.01
self.angle = 0
```

Promjena pomaka vektora definira se uporabom jednostavnih trigonometrijskih funkcija (kao na slici .... 8.6) koje se nalaze u modulu math.



Slika 8.6: Trigonometrija

Funkcija *pomak()* mijenja *(x,y)* koordinatu na temelju kuta i brzine, koji su promjenljiva svojstva objekta:

Program 8.11 Funkcija pomak

```
1 def pomak(self):
2     self.x += math.sin(self.angle) * self.speed
3     self.y -= math.cos(self.angle) * self.speed
```

U ovoj funkciji kut je zadan u radijanima. Ako se želi raditi sa stunjevima, onda treba koristiti poznatu formulu da je 1 radijan = $180^o/\pi$. Prema tomu, ako želimo da se čestica ili objekt kreće s lijeva nadesno, kut bi trebao biti $\pi/2$ (90^o):

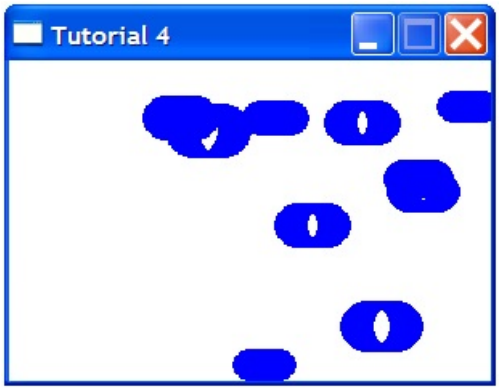
```
self.angle = math.pi/2
```

Pomak se, poput prikaza, ubacuje u petlju po svim česticama:

Zaslon 8.4

```
for particle in my_particles:
    particle.move()
    particle.display()
```

pa će se sljedeća slika pojaviti na zaslonu: Pomak i crtanje u svakom sljedećem koraku, ostavlja



Slika 8.7: Čestice

prethodni trag na zaslonu. Ako se želi brisati prethodni trag, onda se prije novog prikaza čestice ili objekta, izbrisati prozor, što je najlakše postići pozivom naredbe: `screen.fill(background_colour)` u navedenoj petlji. S obzirom da se izračunom koordinata preko *sin()* i *cos()* funkcija redovito dobivaju realni brojevi, potrebno ih je pretvoriti u cjelobrojne (jer su koordinate prozora izražene u točkicama, cjelobrojne):
`pygame.draw.circle(screen, self.colour, (int(self.x), int(self.y)), self.size, self.thickness)`

Često je za neku simulaciju s česticama potrebno staviti ograničenja na okvir u kojem se simulacija događa, odnosno prikazuje. To se obično izvodi postavljanjem virtualnih zidova od kojih se pojedina čestica ili objekt odbija i vraća u okvir ili se pojavljuje s druge strane okvira, kao da se radi o zakrivljenom, kuglastom obliku. Za takve programske realizacije potrebno je samo malo trigonometrijskog znanja (vidi primjer . . .).

Ograničenja gibanja čestice ili objektna na bilo koju od četiri strane okvira/prozora su:

- x=*0* (lijevi zid)
- x=*width* (desni zid)
- y=*0* (vrh)
- y=*height* (dno)

Budući da se simulacija događa u diskretnim/pojedinačnim vremenskim koracima, nije nužno ispitivanje je li čestica dosegla okvir, nego je li njena pozicija izvan okvira, prije nego se nacrta. Ako jest, poduzmu se pripadne akcije prema zamišljenom algoritmu odbijanja ili pojavljivanja (vidi primjer. . . .). Slika (. . . .) pomaže u razmijevanju formula koje se pojavljuju u primjeru.

Verikalno ograničenje ima kut 0^o, dok horizontalno ima kut od 180^o. Refleksija čestice tj. objekta dobiva se dakle u slučaju vertikalnog ograničenja oduzimanjem trenutačnog kuta gibanja od 0^o ili od 180^o u slučaju horizontalnog ograničenja.

Program 8.12 Funkcija odbijanja

```
1 def bounce(self):
2     if self.x > width - self.size:
```

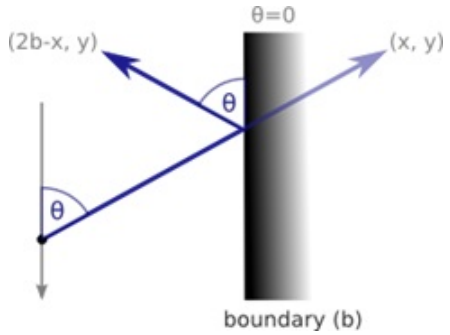
```
3           self.x = 2*(width - self.size) - self.x
4           self.angle = - self.angle
5
6       elif self.x < self.size:
7           self.x = 2*self.size - self.x
8           self.angle = - self.angle
9
10          if self.y > height - self.size:
11              self.y = 2*(height - self.size) - self.y
12              self.angle = math.pi - self.angle
13
14          elif self.y < self.size:
15              self.y = 2*self.size - self.y
16                  self.angle = math.pi - self.angle
```

Na koncu, potrebno je funkciju koja radi s ograničenjima simulacije na prozoru, ugraditi u već poznatu petlju:

Zaslon 8.5

```
for particle in my_particles:
    particle.move()
    particle.bounce()
    particle.display()
```

Prepreku gibanja čestice/objekta mogu predstavljati i druge čestice/objekti, što obično nazvamo sudarima. Koja će se operacija ostvariti u sudaru čestica/objekata, dakako ovisi o zamišljenom algoritmu. Moguć je sraz ili odbijanje čestica, kao i stapanje u jednu zajedničku, poništavanje neke od njih i slično. Često se uz sile koje djeluju u gibanju čestica dodaje i gravitacijska, pa se u svakom trenutku računa rezultanta dvaju ili više vektora.



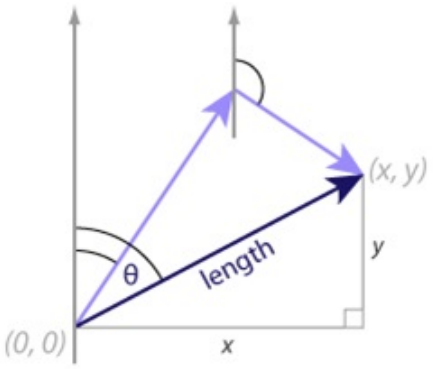
Slika 8.8: Odbijanje

Slika (8.8) pomaže u razumijevanju trigonometrijskih formula ugradjenih u algoritam zbrajanja dvaju vektora:

Program 8.13 Funkcija zbrajanja vektora

```
1
2   def addVectors((angle1, length1), (angle2, length2)):
3     x  = math.sin(angle1)*length1 + math.sin(angle2)*length2
4     y  = math.cos(angle1)*length1 + math.cos(angle2)*length2
```

```
5     length = math.hypot(x, y)
6     angle = 0.5*math.pi-math.atan2(y, x)
7     return (angle, length)
```



Slika 8.9: zbrajanje vektora

Na sličan način u algoritam se može uključiti i trenje, ako se čestica/objekt povlači po nekoj podlozi, kao i elastičnost same podloge.

8.3 Matplotlib

Matplotlib je izvrsna grafička knjižnica za stvaranje 2D i 3D grafike u inženjerskim i znanstvenim primjenama. Prednosti ovog grafičkog Python modula su:

- brza i jednostavna uporaba
- ugrađena potpora LaTeXoznaka i tekstova
- upravljanje svakog elementa u slici
- gotovo identične naredbe kao u Matlab alatu
- visoka kvaliteta izlaznih formata, uključujući PNG, PDF, SVG, EPS i PGF formate
- GUI za interaktivnu uporabu i naknadnu obradbu

Dodatna informacija objavljena je na Matplotlib mrežnoj stranici: http://matplotlib.org/. Uključivanje Matplotlib modula u korisnički Python program provodi se obično na dva načina:

```
from pylab import *
```

ili s:

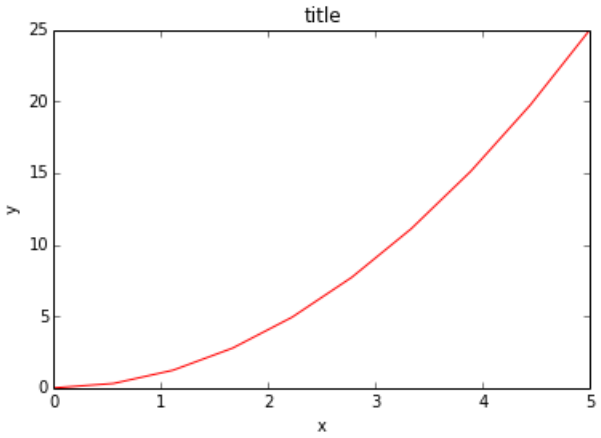
```
import matplotlib.pyplot as plt
```

Nakon učitanja poziva se objekt slike (*figure()*) na kojem se provode željene grafičke obradbe:

Zaslon 8.6

```
x = linspace(0, 5, 10)
y = x ** 2
figure()
plot(x, y, 'r')
xlabel('x')
ylabel('y')
title('title')
show()
```

što će dati:



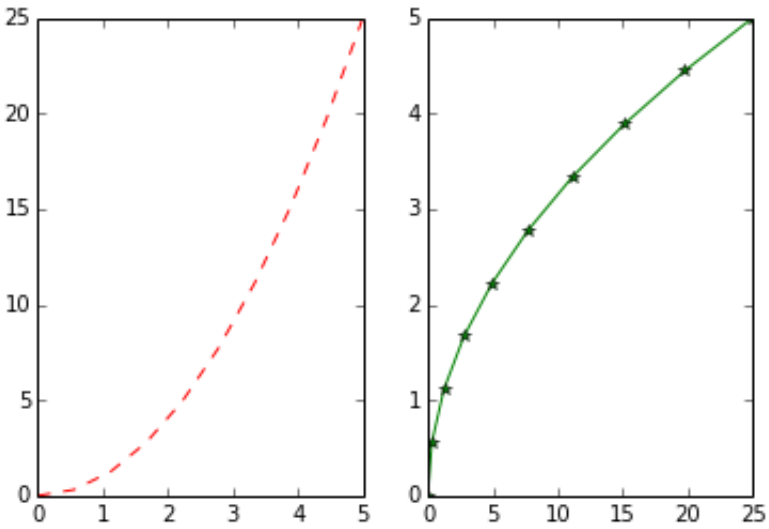
Slika 8.10: Matplotlib grafika

U istom grafičkom prozoru moguće je nacrtati više podcrteža, za što se koristi funkcija *subplot()*:

Zaslon 8.7

```
subplot(1,2,1)
plot(x, y, 'r--')
subplot(1,2,2)
plot(y, x, 'g*-');
```

Izvođenje ovog programskog odsječka dat će:



Slika 8.11: Višestruki grafovi istog crteža

Primjetite, kako se s prva dva argumenta u funkciji *plot()* prenose *x* i *y* odatčani vektori, dok se s trećim parametrom određuje boja i oznaka linije s kojom se graf crta.

8.3.1 Temeljne funkcije

Matplotlib funkcija ima nekoliko stotina, a svaka ima više argumenata. Ovdje će, uz popratne slike, biti predočene samo najosnovnije. Za upoznavanje ostalih funkcija, preporučuje se proučiti programski manual, prema potrebi i uporabi.
Funkcijom *figure()* stvara se slika određene veličine i rezoluciji (*dpi* - eng. *dot per inch*, broj točkica po mjerilu dužine).

Zaslon 8.8
```
figure(figsize=(10,6), dpi=80)\\
plot(X, Y, color="blue", linewidth=2.5, linestyle="-")}
```

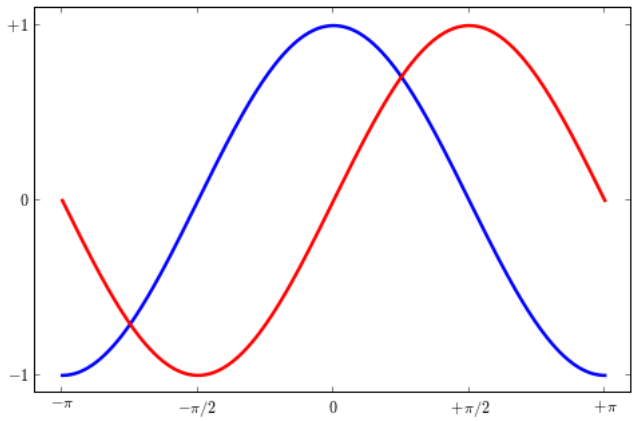
U funkciji *plot()* osim vektora *X* i *Y*, koriste se argumenti za boje, debljine i vrste linija i sl. Ako se želi nacrtati graf koji je npr. za 10% udaljen od obje osi, iskoristit će se *xlim()* i *ylim()* funkcije koje će proširiti okvir ako najveću i najmanju vrijedost pomnožimo s 1.1.

Zaslon 8.9
```
xlim(X.min()*1.1, X.max()*1.1)
ylim(Y.min()*1.1, Y.max()*1.1)
```

Oznaka na osima, mogu se postavljati s pomoću funkcija *xticks()* i *yticks()* i to s dva argumenta - listom mjesta na kojima će se oznake, podioke, pojavljivati i listom s tekstom ili formulom koja će se ispisivati poviše njih. Za formule služi LaTeXzapis.

Zaslon 8.10
```
xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
       [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])

yticks([-1, 0, +1],
       [r'$-1$', r'$0$', r'$+1$'])
```

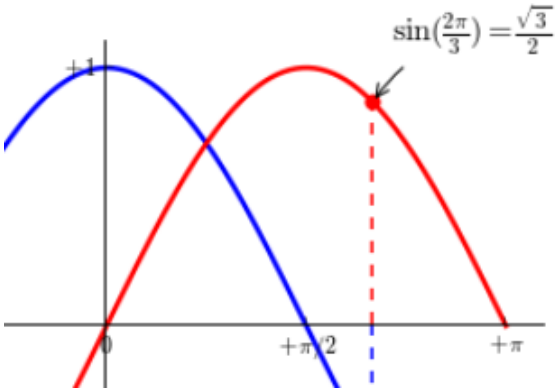


Slika 8.12: Oznake na x,y osima

Ako se želi označiti neka posebna točna na grafu, onda se to može načiniti s pomoću *annotate()* funkcije:

Zaslon 8.11

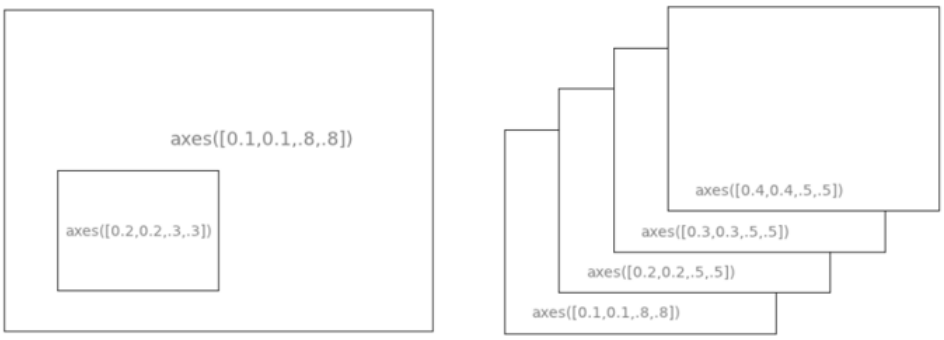
```
annotate(r'$\sin(\frac{2\pi}{3})=\frac{\sqrt{3}}{2}$',
         xy=(t, np.sin(t)), xycoords='data',
         xytext=(+10, +30), textcoords='offset points', fontsize=16,
         arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))
```



Slika 8.13: Označivanje točaka

Legenda se u grafu može smjestiti na neki od kuteva, npr. *legend(loc='upper left')*, a natpisi u legendi se dobiju preko argumenta *label* u plot naredbi, npr. *label="cosinus"*. U tom slučaju će uz liniju (određene vrste i boje, pisati i zadana oznaka, npr. *cosinus* u ovom slučaju).

Funkcija axes() je slična funkciji subplots() s još većom slobodom postavljanja grafova na određeno mjesto. Ako se na primjer želi staviti slika unutar slike, onda je to najlakše izvesti sa axes() funkcijom.



Slika 8.14: Pozicioniranje crteža

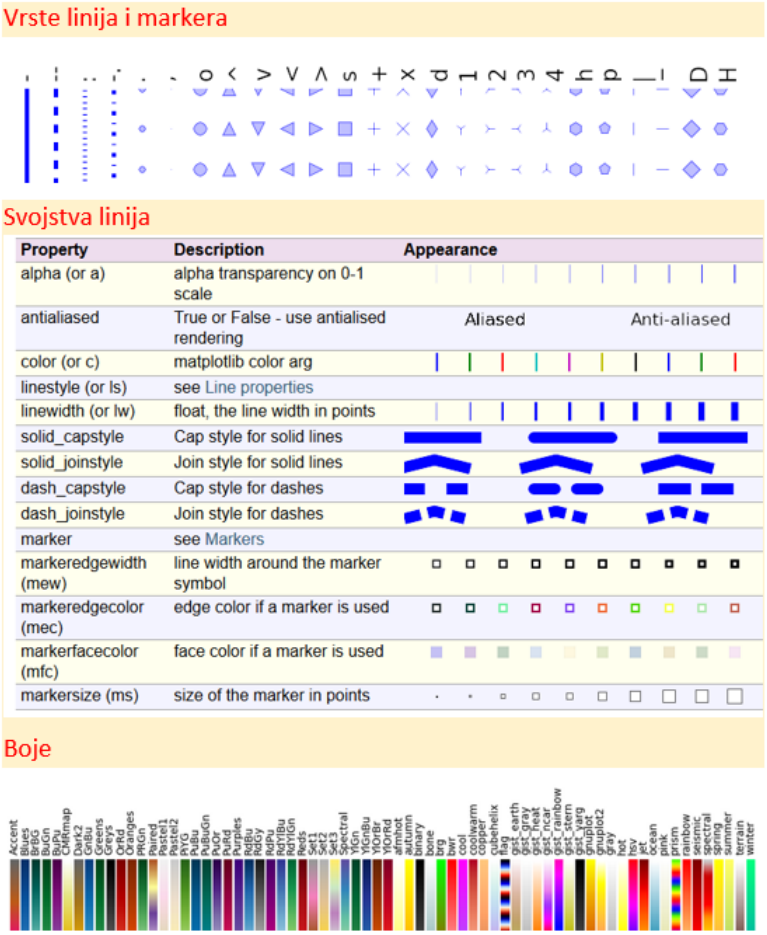
Postoji mnogo vrsta linija, markera i boja, kao što prikazuje slika:

8.3.2 Matplotlib - widgets

Widgets-i su kombinirana interaktivna grafika s ugradjenim upravljačkim elementima.

Program 8.14 Interaktivni upravljački vizualni elementi

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from matplotlib.widgets import Slider, Button, RadioButtons
```

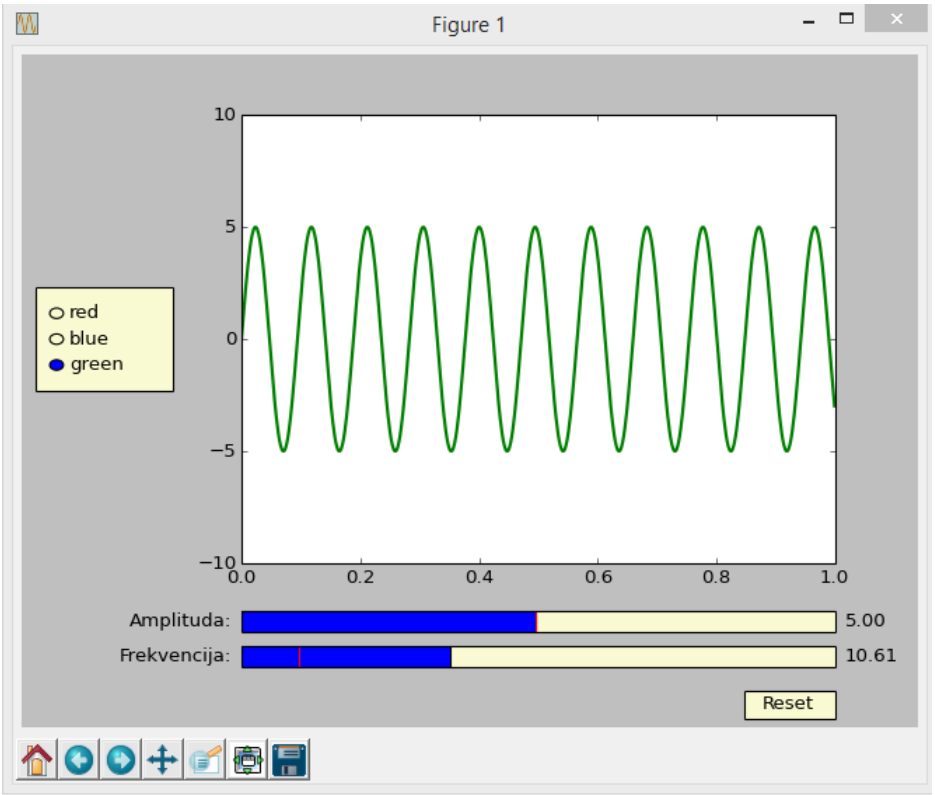


Slika 8.15: vrste linija, markara, boja

```
4
5   fig, ax = plt.subplots()
6   plt.subplots_adjust(left=0.25, bottom=0.25)
7   t = np.arange(0.0, 1.0, 0.001)
8   a0 = 5
9   f0 = 3
10  s = a0*np.sin(2*np.pi*f0*t)
11  l, = plt.plot(t,s, lw=2, color='red')
12  plt.axis([0, 1, -10, 10])
13
14  axcolor = 'lightgoldenrodyellow'
15  axfreq = plt.axes([0.25, 0.1, 0.65, 0.03], axisbg=axcolor)
16  axamp  = plt.axes([0.25, 0.15, 0.65, 0.03], axisbg=axcolor)
17
18  sfreq = Slider(axfreq, 'Frekvencija:', 0.1, 30.0, valinit=f0)
19  samp = Slider(axamp, 'Amplituda:', 0.1, 10.0, valinit=a0)
20
21  def update(val):
22      amp = samp.val
23      freq = sfreq.val
```

```
24      l.set_ydata(amp*np.sin(2*np.pi*freq*t))
25      fig.canvas.draw_idle()
26  sfreq.on_changed(update)
27  samp.on_changed(update)
28
29  resetax = plt.axes([0.8, 0.025, 0.1, 0.04])
30  button = Button(resetax, 'Reset', color=axcolor, hovercolor='
        0.975')
31  def reset(event):
32      sfreq.reset()
33      samp.reset()
34  button.on_clicked(reset)
35
36  rax = plt.axes([0.025, 0.5, 0.15, 0.15], axisbg=axcolor)
37  radio = RadioButtons(rax, ('red', 'blue', 'green'), active=0)
38  def colorfunc(label):
39      l.set_color(label)
40      fig.canvas.draw_idle()
41  radio.on_clicked(colorfunc)
42
43  plt.show()
```

Izvođenje ovog programskog odsječka dat će:



Slika 8.16: Kombinacija grafa i upravljačkih elemenata

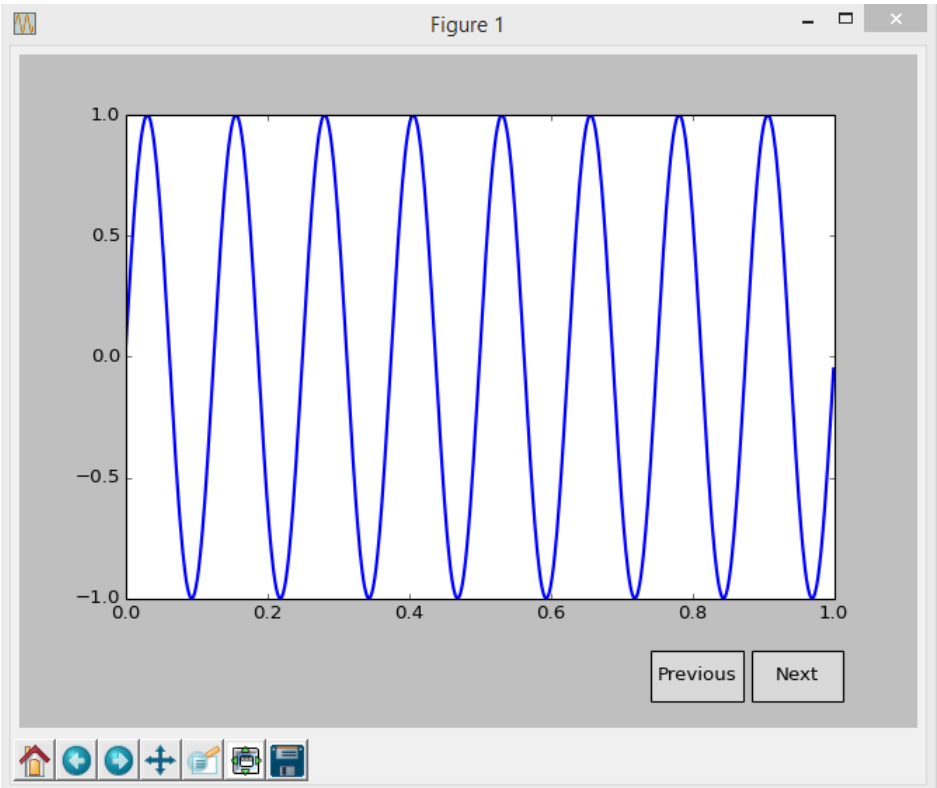
Program 8.15 Funkcije vizualnih elemenata

```
1
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from matplotlib.widgets import Button
5
6  freqs = np.arange(2, 20, 3)
7
8  fig, ax = plt.subplots()
9  plt.subplots_adjust(bottom=0.2)
10 t = np.arange(0.0, 1.0, 0.001)
11 s = np.sin(2*np.pi*freqs[0]*t)
12 l, = plt.plot(t, s, lw=2)
13
14
15 class Index:
16     ind = 0
17     def next(self, event):
18         self.ind += 1
19         i = self.ind % len(freqs)
20         ydata = np.sin(2*np.pi*freqs[i]*t)
21         l.set_ydata(ydata)
22         plt.draw()
23
24     def prev(self, event):
25         self.ind -= 1
26         i = self.ind % len(freqs)
27         ydata = np.sin(2*np.pi*freqs[i]*t)
28         l.set_ydata(ydata)
29         plt.draw()
30
31 callback = Index()
32 axprev = plt.axes([0.7, 0.05, 0.1, 0.075])
33 axnext = plt.axes([0.81, 0.05, 0.1, 0.075])
34 bnext = Button(axnext, 'Next')
35 bnext.on_clicked(callback.next)
36 bprev = Button(axprev, 'Previous')
37 bprev.on_clicked(callback.prev)
38
39 plt.show()
```

Izvođenje ovog programskog odsječka dat će:

Program 8.16 Funkcija 'SpanSelector' s min/max rasponom i promatranim detaljima:

```
1  #!/usr/bin/env python
2  """
3  The␣SpanSelector␣is␣a␣mouse␣widget␣to␣select␣a␣xmin/xmax␣range␣
       and␣plot␣the
4  detail␣view␣of␣the␣selected␣region␣in␣the␣lower␣axes
```

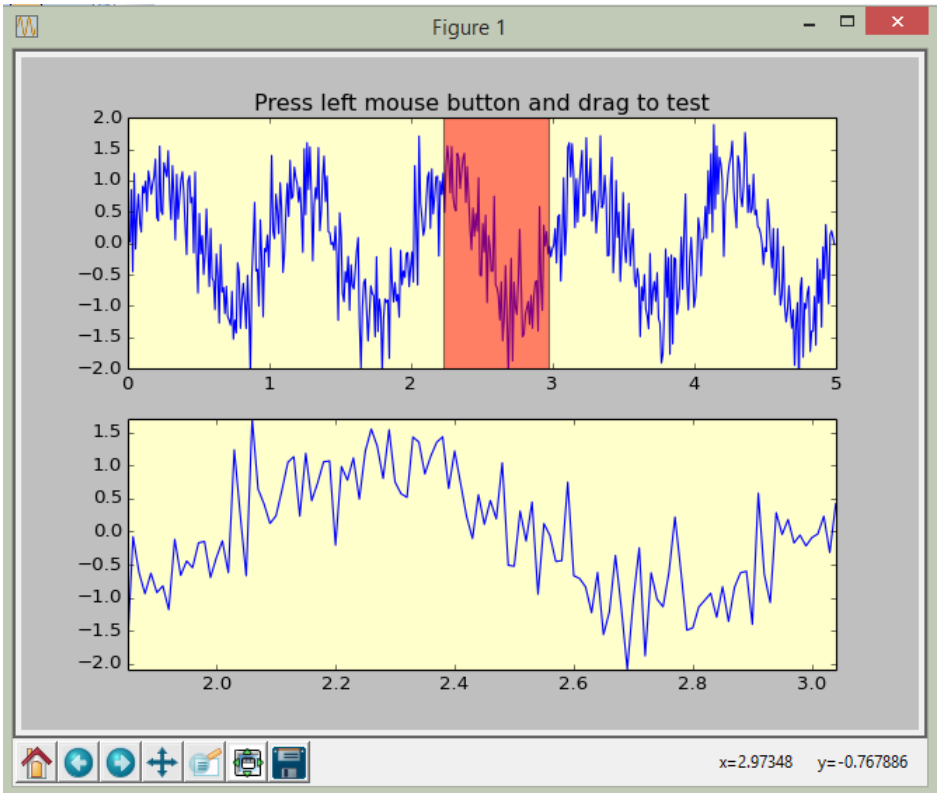


Slika 8.17: Grafika i upravljački elementi

```
5  """
6  import numpy as np
7  import matplotlib.pyplot as plt
8  from matplotlib.widgets import SpanSelector
9
10 fig = plt.figure(figsize=(8,6))
11 ax = fig.add_subplot(211, axisbg='#FFFFCC')
12
13 x = np.arange(0.0, 5.0, 0.01)
14 y = np.sin(2*np.pi*x) + 0.5*np.random.randn(len(x))
15
16 ax.plot(x, y, '-')
17 ax.set_ylim(-2,2)
18 ax.set_title('Press left mouse button and drag to test')
19
20 ax2 = fig.add_subplot(212, axisbg='#FFFFCC')
21 line2, = ax2.plot(x, y, '-')
22
23
24 def onselect(xmin, xmax):
25     indmin, indmax = np.searchsorted(x, (xmin, xmax))
26     indmax = min(len(x)-1, indmax)
27
28     thisx = x[indmin:indmax]
```

```
29      thisy = y[indmin:indmax]
30      line2.set_data(thisx, thisy)
31      ax2.set_xlim(thisx[0], thisx[-1])
32      ax2.set_ylim(thisy.min(), thisy.max())
33      fig.canvas.draw()
34
35  # set useblit True on gtkagg for enhanced performance
36  span = SpanSelector(ax, onselect, 'horizontal', useblit=True,
        rectprops=dict(alpha=0.5, facecolor='red') )
37
38  plt.show()
```

Izvođenje ovog programskog odsječka dat će:



Slika 8.18: Zoom rezultata

Program 8.17 Funkcije numpy i površinske grafike

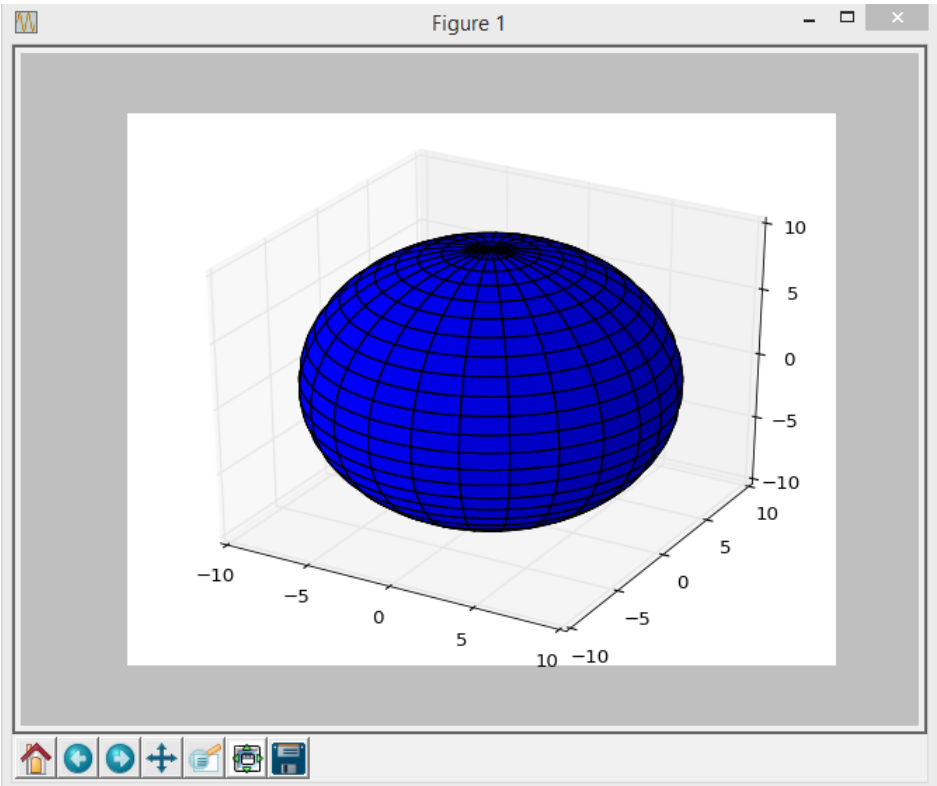
```
1
2  from mpl_toolkits.mplot3d import Axes3D
3  import matplotlib.pyplot as plt
4  import numpy as np
5
6  fig = plt.figure()
7  ax = fig.add_subplot(111, projection='3d')
8
9  u = np.linspace(0, 2 * np.pi, 100)
```

```
v = np.linspace(0, np.pi, 100)

x = 10 * np.outer(np.cos(u), np.sin(v))
y = 10 * np.outer(np.sin(u), np.sin(v))
z = 10 * np.outer(np.ones(np.size(u)), np.cos(v))
ax.plot_surface(x, y, z,  rstride=4, cstride=4, color='b')

plt.show()
```

Izvođenje ovog programskog odsječka dat će 3D crtež:



Slika 8.19: 3D kugla

Program 8.18 Funkcije axes3d i surface.

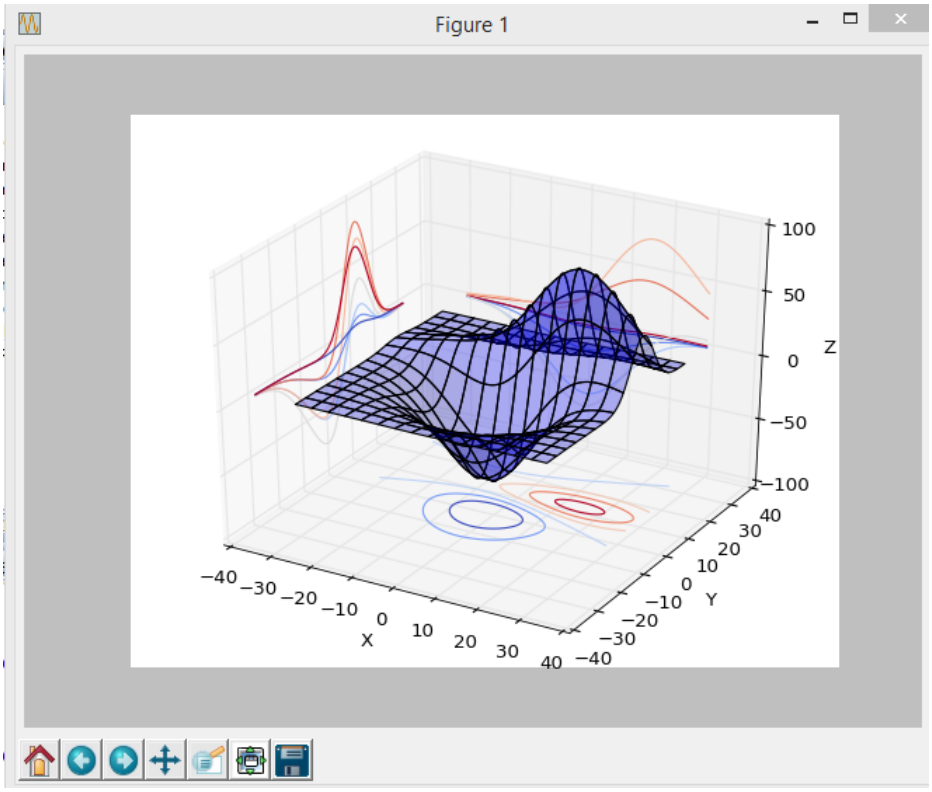
```

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

fig = plt.figure()
ax = fig.gca(projection='3d')
X, Y, Z = axes3d.get_test_data(0.05)
ax.plot_surface(X, Y, Z, rstride=8, cstride=8, alpha=0.3)
cset = ax.contour(X, Y, Z, zdir='z', offset=-100, cmap=cm.
    coolwarm)
cset = ax.contour(X, Y, Z, zdir='x', offset=-40, cmap=cm.
```

```
    coolwarm)
12  cset = ax.contour(X, Y, Z, zdir='y', offset=40, cmap=cm.
    coolwarm)
13
14  ax.set_xlabel('X')
15  ax.set_xlim(-40, 40)
16  ax.set_ylabel('Y')
17  ax.set_ylim(-40, 40)
18  ax.set_zlabel('Z')
19  ax.set_zlim(-100, 100)
20
21  plt.show()
```

Izvođenje ovog programskog odsječka dat će 3D crtež:



Slika 8.20: 3D konturni crtež s projekcijama