



Prirodoslovno-matematički fakultet
Matematički odsjek
Sveučilište u Zagrebu

Uvod u teoriju upravljanja

Kratki uvod u Python

Sastavio: Zvonimir Bujanović

21. rujna 2021.



- 1 Uvod
- 2 Rad u Jupyter Lab-u
- 3 Osnove programiranja u Pythonu
- 4 NumPy, SciPy, matplotlib

Python i znanstveno računanje

Python je programski jezik opće namjene.

Sve je popularniji u znanstvenoj zajednici, naročito jer je velika većina njegovih proširenja otvorenog koda.

Brojnim paketima moguće je jako proširiti funkcionalnost, čime Python omogućava:

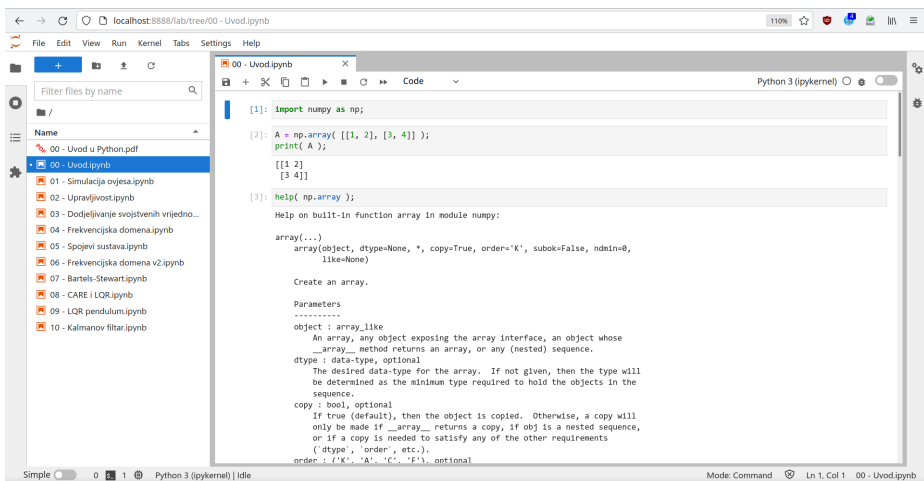
- izvođenje kompleksnih proračuna
- vizualizaciju rezultata
- izvođenje simulacija
- programiranje

Uz to, postoje interaktivne ljske poput Jupyter Lab-a, koje dodatno olakšavaju rad.

Aktualna su dva standarda, Python 2.x i Python 3.x.

Mi ćemo koristiti Python 3.x.

Ovi slajdovi su nastali na temelju materijala za kolegij Matematički softver autora prof. Ivice Nakića.



The screenshot displays the JupyterLab web interface. On the left, a file browser shows a directory structure with files named '00 - Uvod u Python.pdf' through '10 - Kalmanov filter.ipynb'. The main area is a code editor for '00 - Uvod.ipynb' running Python 3 (ipykernel). The code in the editor is as follows:

```
[1]: import numpy as np;
[2]: A = np.array( [[1, 2], [3, 4]] );
    print( A );
[[1 2]
 [3 4]]
[3]: help( np.array );
```

The output of the second cell shows the array `[[1 2] [3 4]]`. The third cell displays the help text for `np.array`:

```
Help on built-in function array in module numpy:

array(...)
array(object, dtype=None, *, copy=True, order='K', subok=False, ndmin=0,
      like=None)

Create an array.

Parameters
-----
object : array_like
    An array, any object exposing the array interface, an object whose
    __array__ method returns an array, or any (nested) sequence.
dtype : data-type, optional
    The desired data-type for the array. If not given, then the type will
    be determined as the minimum type required to hold the objects in the
    sequence.
copy : bool, optional
    If true (default), then the object is copied. Otherwise, a copy will
    only be made if __array__ returns a copy, if obj is a nested sequence,
    or if a copy is needed to satisfy any of the other requirements
    ('dtype', 'order', etc.).
order : {'K', 'A', 'C', 'F'}, optional
```

The status bar at the bottom indicates 'Simple' mode, 'Python 3 (ipykernel) | Idle', and 'Mode: Command Ln 1, Col 1 00 - Uvod.ipynb'.

Jupyter Lab pokrećemo naredbom `jupyter lab` u konzoli, u radnom direktoriju.

U ulazne ćelije unosimo kod u Pythonu u jednoj ili više linija.

- Kod izvršavamo tako da pritisnemo `Control + Enter`.
- Varijable ispisujemo funkcijom `print`.
- Alternativno, ako zadnja linija ulazne ćelije nema `;` na kraju, bit će ispisana njezina vrijednost.

Dokumentaciju za pojedinu naredbu možemo dobiti izvršavanjem naredbe `help`:

```
help( print );
```

Tipovi podataka

Python je *slabo tipiziran* jezik. Nije potrebno deklarirati varijable; pojedine varijable mogu mijenjati svoj tip “u letu”:

```
a = [1, 2, 3]; print(type(a));  
a = 'pero'; print(type(a));  
a = (1, 2); print(type(a));
```

```
<class 'list'>  
<class 'str'>  
<class 'tuple'>
```

Osnovni tipovi podataka

Standardni tipovi podataka: `bool`, `int`, `float`, `str`, `list`, `tuple`, `dict`.

- `True`, `False` – logičke vrijednosti (`bool`)
- `7` – cijeli broj (`int`)
- `0.314` ili `3.14e-1` – realni broj (`float`, 64-bitni)
- `-3+0.5j` – kompleksni broj
- `'iter'` ili `"iter"` – string (tip `str`, 1D polje znakova)
- `'x'` ili `"x"` – također string (tip `str`)
- `[1, 2, 3]` – lista (tip `list`)
- `(1, 2)` – n-torka (tip `tuple`)
- `{'ime': 'Pero', 'ocjena': 5}` – rječnik (tip `dict`)
- `print(type(x))` – ispis tipa varijable `x`
- `del(x)` – brisanje varijable `x`.

Rad sa stringovima

```
s = "Hello world";  
print( len(s) );  
  
s2 = s.replace( "world", "test" );  
print( s2 );
```

```
11  
Hello test
```

```
print( s[:5] );  
print( s[6:] );  
print( s[::2] );
```

```
Hello  
world  
Hlowrd
```

```
s3 = "Hello" + "world";  
s3[3] = "x"; # Ovo NE radi, ne možemo mijenjati znakove u stringu!
```


Rad s listama (1)

```
l = [1, 2, 3, 4];  
print( l[1:3] );
```

```
[2, 3]
```

```
l2 = [1, 'a', 1.0, 1-1j];  
l3 = [1, [2, [3, [4, [5]]]]];
```

```
s = 'Hello';  
s2 = list( s );  
print( s2 );
```

```
['H', 'e', 'l', 'l', 'o', ' ', 't', 'e', 's', 't']
```

Rad s listama (2)

```
L = [];  
L.append( "A" );  
L.append( "d" );  
L.append( "d" );  
print( L );
```

```
['A', 'd', 'd']
```

```
L[1] = 'A';  
print( L );
```

```
['A', 'A', 'd']
```

```
L.remove( "A" );  
print( L );
```

```
['A', 'd']
```

Rad sa n-torkama (tuples)

n-torke su kao liste, ali im se elementi ne smiju mijenjati/dodavati.

```
point = (10, 20);  
point[0] = 20;
```

```
TypeError: 'tuple' object does not support item assignment
```

Rad s rječnicima (dict)

Ključevi su obično int-ovi ili stringovi, vrijednosti mogu biti bilo što.

```
student = { "ime": "Pero", "ocjena": 5, 3: "nesto" };  
print("Student " + student["ime"] + " ima ocjenu " + str(student[ "ocjena"])
```

```
Student Pero ima ocjenu 5
```

Funkcije

Python je JAKO osjetljiv na razmake.

Tijelo funkcije (kao i petlji, if-a, itd.) je odmaknuto jednim tab-om ili 4 razmaka.

```
def zero():  
    return 0;
```

```
zero()
```

```
0
```

Sljedeća funkcija vraća n-torku (tuple).

```
def powers(x):  
    """
```

```
    Potencije od x.  
    """
```

```
    # Može i return (x ** 2, x ** 3, x ** 4);
```

```
    return x ** 2, x ** 3, x ** 4;
```

```
# Može i (x2, x3, x4) = powers( 3 ) ili [x2, x3, x4] = powers( 3 );
```

```
x2, x3, x4 = powers( 3 );
```

```
print( x3 );
```

```
27
```

Kontrola toka

```
izjava1 = False;
izjava2 = False;

if izjava1:
    print( "izjava1 je True" );
elif izjava2:
    print( "izjava2 je True" );
else:
    print( "izjava1 i izjava2 su False" );
```

izjava1 i izjava2 su False

```
if not izjava1:
    if not izjava2:
        print( "izjava1 i izjava2 su False" )
```

izjava1 i izjava2 su False

Petlje (1)

```
for x in range(4): # range počinje od 0
    print(x)
```

```
0
1
2
3
```

```
for broj in [ "jen", "dva", "tri" ]:
    print(word)
```

```
jen
dva
tri
```

```
params = { "prvi": 3, "drugi": "osam", 5: "blabla"}
for key, value in params.items():
    print( str(key) + " -> " + str(value) );
```

```
prvi -> 3
drugi -> osam
5 -> blabla
```

Petlje (2)

```
i = 0;
while( i < 2 ):
    print( i );
    i = i + 1;

print( "gotovo" );
```

```
0
1
gotovo
```



```
class Point:
    """
    Jednostavna klasa koja služi za rad s točkama u Euklidskoj ravnini.
    """
    def __init__( self, x, y ):
        """
        Kreiranje točke s koordinatama x i y.
        """
        self.x = x;
        self.y = y;
    def translate( self, dx, dy ):
        """
        Translacija točke za dx u smjeru x-osi i dy u smjeru y-osi.
        """
        self.x += dx;
        self.y += dy;
    def __str__( self ):
        """
        Prikaz točke (konverzija u str).
        """
        return( "Point at [{:f}, {:f}]".format( self.x, self.y ) );

p1 = Point( 1, 1 );
p1.translate( 2, 3.5 );
print( p1 );
```

Moduli (1)

Veći komad koda možemo spremiti u datoteku i kasnije koristiti kao modul.
Neka je ovo spremljeno u datoteku `mojmodul.py` :

```
"""  
Primjer modula. Sadrži varijablu x, funkciju f te klasu K.  
"""  
x = 0;  
  
def f():  
    """  
    Primjer funkcije  
    """  
    return x + 1;  
  
class K:  
    """  
    Primjer klase  
    """  
    def __init__( self ):  
        self.clan = x + 2;  
    def get_clan( self ):  
        return self.clan;
```

Moduli (2)

Korištenje modula s prethodnog slajda:

```
import mojmodul;

print( mojmodul.x );
print( mojmodul.f() );
k = mojmodul.K(); print( k.get_clan() );

0
1
2
```

Umjesto `modmodul`, možemo koristiti alternativno ime:

```
import mojmodul as mm;

print( mm.x );
print( mm.f() );
k = mm.K(); print( k.get_clan() );
```

Možemo i sve iz modula koristiti direktno, bez ikakvog prefiksa (nije dobra praksa):

```
from mojmodul import *;

print( x );
print( f() );
k = K(); print( k.get_clan() );
```

Zadatak 1

Napravite modul `kvadratna.py` koji sadrži funkciju `kvadratna`. Funkcija prima brojeve a , b , c i vraća uređeni par rješenja kvadratne jednadžbe

$$ax^2 + bx + c = 0.$$

NumPy

- Paket (modul) za efikasno numeričko računanje u Pythonu.
- Naglasak je na efikasnom računanju s nizovima, vektorima i matricama, uključivo višedimenzionalne strukture.

SciPy

- Nadgradnja paketa NumPy, sadrži veliki broj numeričkih algoritama za cijeli niz područja.
- Specijalne funkcije (`scipy.special`), integracija (`scipy.integrate`), optimizacija (`scipy.optimize`), interpolacija (`scipy.interpolate`), fourierova transformacija (`scipy.fftpack`), linearna algebra (`scipy.linalg`), linearna algebra s rijetkim matricama (`scipy.sparse`), statistika (`scipy.stats`), procesiranje slika (`scipy.ndimage`).

matplotlib

- Paket za generiranje 2d-grafova.

NumPy – Vektori i matrice

```
import numpy as np;
```

```
v = np.array( [ 1, 2, 3, 4] );    # vektor (stupac ili redak, svejedno).  
M = np.array( [[1, 2], [3, 4]] ); # 2x2 matrica.
```

```
print( v.shape );  
print( M.shape );
```

```
(4,)  
(2, 2)
```

Možemo koristiti i tip `matrix` kojem nije svejedno je li vektor red ili stupac. Iako je ovako jasnije i sličnije Matlabu, tip `matrix` nije preporučeno koristiti.

```
v = np.matrix( [ 1, 2, 3, 4] );    # vektor-redak.  
M = np.matrix( [[1, 2], [3, 4]] ); # 2x2 matrica.
```

```
print( v.shape );  
print( M.shape );
```

```
(1, 4)  
(2, 2)
```

Koja je razlika između `numpy.ndarray` tipa i standardnih lista u Pythonu?

- liste u Pythonu mogu sadržavati bilo kakve vrste objekata, to nije slučaj s `numpy.ndarray`.
- `numpy.ndarray` nisu dinamički objekti: pri kreiranju im je određen tip.
- za `numpy.ndarray` implementirane su razne efikasne metode važne u numerici.
- de facto sva računanja se odvijaju u C-u i Fortranu pomoću BLAS rutina.

`dtype` (data type) nam daje informaciju o tipu podataka u nizu:

```
M.dtype
```

```
dtype('int32')
```

Tip elementa koji se čuva u matrici možemo zadati kod deklaracije.

```
M = np.array( [[1, 2], [3, 4]], dtype=complex );
```

Generiranje vektora i matrica

```
x = np.linspace( 0, 1, 5 );  
print( x );
```

```
[0.  0.25 0.5  0.75 1.  ]
```

```
y = np.logspace( 0, 10, 10, base=e );  
print( y );
```

```
[1.00000000e+00 3.03773178e+00 9.22781435e+00 2.80316249e+01  
8.51525577e+01 2.58670631e+02 7.85771994e+02 2.38696456e+03  
7.25095809e+03 2.20264658e+04]
```

```
A = np.zeros( (3, 3) );           # 3x3 nul-matrica.  
B = np.eye( 3, 3 );             # 3x3 jedinična matrica (bez zagrada!)  
C = np.ones( (3, 3) );         # 3x3 matrica puna jedinica.  
D = np.diag( [1, 2, 3] );      # 3x3 dijagonalna matrica s 1, 2, 3 na dijag.  
E = np.diag( [1,2,3], k=1 );   # 4x4 matrica sa 1, 2, 3 na 1. sporednoj dij.
```

```
from numpy import random;  
F = random.rand( 5, 3 );      # 5x3 slučajna matrica, uniformo u [0, 1].  
G = random.randn( 4, 7 );    # 4x7 slučajna matrica, normalna distribucija.
```


Pristup elementima i podmatricama

Elementi se indexiraju od 0.

```
v = np.array( [1, 2, 3, 4, 5] );  
  
v[0] = 7;           # sada: v = [7, 2, 3, 4, 5]  
print( v[1:3] );   # indexi od 1 do 3 (bez desnog ruba, indexi 1, 2) -> [2,  
print( v[1:] );    # od indexa 1 do kraja -> [2, 3, 4, 5]  
print( v[:3] );    # od početka do (ne uključujući!) indexa 3 -> [7, 2, 3]  
print( v[-2] );    # drugi element od kraja -> 4  
print( v[-3:] );   # zadnja 3 elementa -> [3, 4, 5]  
v[1:3] = [0, 1];   # radi i pridruživanje -> v = [7, 0, 1, 4, 5]
```

```
[2 3]  
[2 3 4 5]  
[7 2 3]  
4  
[3 4 5]
```

Pristup elementima i podmatricama

```
M = np.array( [[10, 20, 30], [40, 50, 60]] );

print( M );
print( M[1, 1] )      # element (1, 1) -> 50
print( M[1] )        # redak 1 -> [40, 50, 60]
print( M[1, :] )     # redak 1 -> [40, 50, 60]
print( M[:, 1] )     # stupac 1 -> [20, 50]
print( M[1:2, 0:2] ); # redak 1 i stupci 0, 1 -> [[40, 50]]

indeksi_redaka = [-1, 0];
print( M[indeksi_redaka, :] ) # zadnji redak i redak 0.

M[1, :] = 0;        # sve elemente retka 1 postavi na 0.
M[:, 0] = -1;       # sve elemente stupca 0 postavi na -1.
```

```
[[10 20 30]
 [40 50 60]]
50
[40 50 60]
[40 50 60]
[20 50]
[[40 50]]
[[40 50 60]
 [10 20 30]]
```

Funkcije nad vektorima i matricama

```
A = np.array( [[1, 2], [3, 4]] );
v = np.array( [5, 6] );

np.sum( v );           # suma elemenata vektora v
np.prod( v );         # produkt elemenata vektora v

np.diag( A );         # dijagonala matrice A
np.diag( A, -1 );    # prva poddijagonala matrice A

B = 2 * A;           # Svaki element od A pomnoži s 2.
C = 2 + A;          # Svakom elementu od A dodaj 2.

# array i matrix se različito ponašaju!
D = A * A;          # HADAMARDOV produkt! Ili: D = np.multiply( A, A );
E = A @ A;         # MATRIČNI produkt! Ili: E = np.matmul( A, A );
F = A @ v;         # Matrica puta vektor.
G = v @ A;         # Vektor zapisan kao array se sam transponira.
```

Funkcije nad vektorima i matricama

```
A = np.matrix( [[1, 2], [3, 4]] );
v = np.matrix( [5, 6] );

np.diag( A );      # dijagonala matrice A
np.diag( A, -1 ); # donja sporedna dijagonala matrice A

B = 2 * A; # Svaki element od A pomnoži s 2.
C = 2 + A; # Svakom elementu od A dodaj 2.

# array i matrix se različito ponašaju!
D = A * A; # MATRIČNI produkt!
E = A @ A; # MATRIČNI produkt! Ili: E = np.matmul( A, A );
F = A * v.T; # Matrica puta vektor. Treba transponirati!
G = v * A; # v je vektor redak, pa ne treba transponirati.
H = np.multiply( A, A ); # HADAMARDOV produkt.

I = A.H; # Adjungiranje kompleksne matrice: I = A.conj().T
J = np.real( I ); # Realni dio kompleksne matrice.
K = np.imag( J ); # Imaginarni dio kompleksne matrice.

from scipy.linalg import inv, det;
L = inv(A); # Inverz (ne koristiti!)
d = det(A); # Determinanta.
```

Funkcije nad vektorima i matricama

```
A = np.matrix( [[1, 2], [3, 4]] );  
v = np.matrix( [5, 6] );  
  
B = np.vstack( (A, v) ); # B je 3x2: ispod matrice A stavimo vektor v  
C = np.hstack( (A, v.T) ); # C je 2x3: desno od matrice A stavimo vektor v  
# D = np.hstack( (A, v) ); # Krivo: desno od 2x2 matrice ne može 1x2 vektor
```

Ovo radi i sa `array`, ali obje varijable moraju imati isti broj dimenzija:

```
A = np.array( [[1, 2], [3, 4]] );  
v = np.array( [[5, 6]] ); # Uoči: v je sada 1x2 matrica.  
  
B = np.vstack( (A, v) ); # B je 3x2: ispod matrice A stavimo matricu v  
C = np.hstack( (A, v.T) ); # C je 2x3: desno od matrice A stavimo matricu v  
# D = np.hstack( (A, v) ); # Krivo: desno od 2x2 matrice ne može 1x2 matricu  
  
# Alternativno:  
v = np.array( [5, 6] ); # v je prvo vektor (1d array)  
v = v.reshape( (2, 1) ); # preoblikujemo v u 2x1 matricu  
C = np.hstack( (A, v) ); # sada ga možemo staviti desno od matrice A  
  
# Block je općenitija funkcija, kombinacija hstack i vstack:  
v = np.array( [5, 6] );  
C = np.block( [[A, v.reshape(2, 1)], [v, 7]] ); # C = [A v'; v 7];
```

Funkcije nad vektorima i matricama

Pridruživanje matrica/array-a kopira **samo referencu, ne i elemente!**
Za stvaranje kopije treba koristiti `copy`.

```
A = np.array( [1, 2, 3, 4] );  
  
B = A; B[0] = 17;  
print( A );           # Matrica A se promijenila! -> [17, 2, 3, 4].  
  
B = np.copy( A ); B[0] = 32;  
print( A );           # A je ostala ista -> [17, 2, 3, 4].
```

Sažetak, te detaljni popis funkcionalnosti NumPy-a su ovdje:

- <https://docs.scipy.org/doc/numpy/user/numpy-for-matlab-users.html>
- <https://docs.scipy.org/doc/numpy/reference/routines.html>

SciPy: složenije matrične operacije

Paket SciPy sadrži brojne složenije operacije s matricama i vektorima.
Potpuna lista je ovdje: <https://docs.scipy.org/doc/scipy/reference/linalg.html>

Opisat ćemo sam neke najčešće korištene funkcionalnosti.

```
# Rješavanje sustava linearnih jednadžbi.
from scipy import linalg;

A = np.array( [[1, 2, -1], [4, 5, 6], [7, 8, 9]] );
b = np.array( [1, 2, 3] );

x = linalg.solve( A, b ); # Rješava sustav A*x = b; može i s matrix
print( A @ x - b );

# Puno desnih strana.
B = np.array( [[1, 2, 3], [4, 5, 6]] );
X = linalg.solve( A, B.T ); # Rješava sustav A*X = B.T.
print( A @ X - B.T );
```

```
[ 0.00000000e+00 -2.22044605e-16  0.00000000e+00]
```

Problem najmanjih kvadrata možemo riješiti ovako:

```
x = linalg.lstsq( A, b );
```

SciPy: složenije matrične operacije

Svojstvene vrijednosti.

```
from scipy import linalg;
A = np.array( [[1, 2, -1], [4, 5, 6], [7, 8, 9]] );
lam = linalg.eigvals(A);  # Svojstvene vrijednosti od A.
print( lam );
```

```
[14.36189074+0.j  0.31905463+0.85659134j  0.31905463-0.85659134j]
```

```
[lam, X] = linalg.eig(A);  # I svoj. vrijednosti i svoj. vektori od A.
# Svoj. vektori su stupci u X.
print( A @ X[:, 0] - X[:, 0]*lam[0] );
```

```
[ 3.05311332e-15+0.j -1.77635684e-15+0.j -7.10542736e-15+0.j]
```


SciPy: složenije matrične operacije

Matrične norme.

```
linalg.norm(A, 2);           # 2-norma matrice A
linalg.norm(A, np.inf);      # beskonačno-norma matrice A
linalg.norm(A, 'fro');       # Frobeniusova norma matrice A
```

Ostale važne matrične operacije. Imaju brojne opcije (“ekonomične” faktorizacije, sa ili bez pivotiranja, ...)

```
from scipy.linalg import *;

[U, S, Vh] = svd( A ); # SVD matrice A,
B = pinv( A );        # pseudoinverz matrice A.

[P, L, U] = lu( A );  # LU-faktorizacija od A s (parc.) pivotiranjem.
L = cholesky( A );    # Cholesky faktorizacija (poz.def.matrice) A.
[Q, R] = qr( A );     # QR-faktorizacija od A.

B = expm( A );        # matrična eksponencijalna funkcija.
```

Pomoću modula `matplotlib` možemo lako crtati grafove raznih tipova.

```
from matplotlib.pyplot import *;
# Iduća linija omogućava crtanje unutar Jupyter Lab-a.
%matplotlib inline
```

Nacrtajmo graf kvadratne funkcije i apsolutne vrijednosti.
Ovo je vrlo slično kao u Matlab-u.

```
x = np.linspace( -2, 2, 21 ); # x = 21 točaka u domeni [-2, 2].
y = x ** 2; # y = njihovi kvadrati.
z = np.abs( x ); # z = njihove aps. vrijednosti.

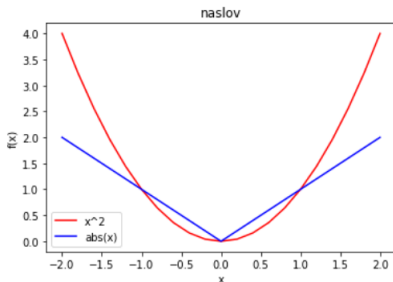
fig = figure(); # Stvori novi graf.
plot( x, y, 'r', label="x^2" ); # Crvenom bojom ('r') -> podaci iz x i y.
plot( x, z, 'b', label="abs(x)" ); # Plavom bojom ('b') -> podaci iz x i z.
xlabel( 'x' ); # x-os označi sa 'x'.
ylabel( 'f(x)' ); # y-os označi sa 'f(x)'.
title( 'naslov' ); # Stavi naslov slike 'naslov'.
legend(); # Dodaj legendu sa "x^2" i "abs(x)" na sliku.
show(); # Prikaži sliku.
```

Matplotlib

Isti efekt možemo postići i korištenjem objektnog pristupa.

```
import matplotlib.pyplot as plt;

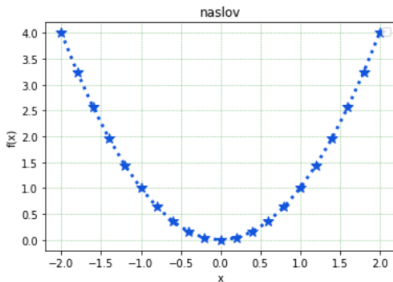
[fig, ax] = plt.subplots();
ax.plot( x, y, 'r', label="x^2" );
ax.plot( x, z, 'b', label="abs(x)" );
ax.set_xlabel( 'x' );
ax.set_ylabel( 'f(x)' );
ax.set_title( 'naslov' );
ax.legend();
```



Možemo postavljati razne stilove linija.

```
# color = HTML-boja
# lw = debljina linije
# ls = stil linije ('-', ':", '--', '-. ') - iscrtkana, istočkana...
# marker = ('o', '*', 's', 'd', '+') - oznake na liniji
ax.plot( x, y, color="#1155dd", lw=3, ls=':', marker='*', markersize=10 )

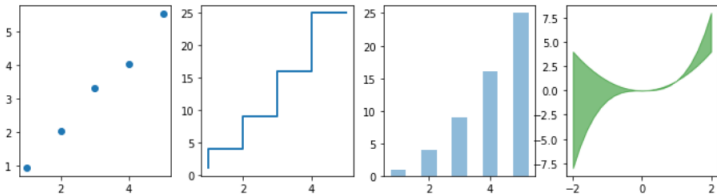
# Dodajemo i zelenu mrežu u pozadinu grafa.
ax.grid( color='g', alpha=0.5, linestyle='dashed', linewidth=0.5 )
```



Matplotlib

Postoji još puno drugih tipova grafa.

```
n = np.array( [1, 2, 3, 4, 5] );  
  
[fig, axes] = plt.subplots( 1, 4, figsize=(12,3) )  
axes[0].scatter( n, n + 0.25*np.random.randn(len(n)) )  
axes[1].step( n, n**2, lw=2 )  
axes[2].bar( n, n**2, align="center", width=0.5, alpha=0.5 )  
axes[3].fill_between( x, x**2, x**3, color="green", alpha=0.5 );
```



Spremanje slike na disk.

```
fig.savefig( "ime.png" );
```

Zadatak 2

Generirajte slučajnu 100×100 matricu, izračunajte njezine svojstvene vrijednosti i nacrtajte ih u kompleksnoj ravnini.

Postoji li razlika ako generirate matricu sa `rand` i sa `randn` ?

SciPy: rješavanje ODJ (1)

Korištenjem funkcije `odeint` možemo rješavati ODJ.

```
from scipy.integrate import odeint;
```

Riješimo ODJ

$$y'(t) = t - y, \quad y(0) = 1.$$

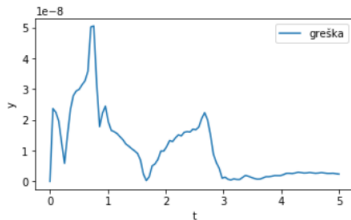
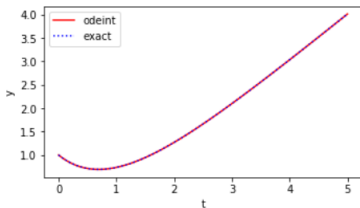
Znamo eksplicitno rješenje $y(t) = t - 1 + 2e^{-t}$.

```
def f(y, t):  
    # Funkcija koja vraća desnu stranu ODJ  $y'(t) = f(y, t)$   
    return t - y;  
  
t = np.linspace( 0, 5, 100 ); # točke u kojima tražimo  $y(t)$   
y0 = 1.0; # početni uvjet  
y = odeint( f, y0, t ); # rješavamo ODJ  
y = y[:, 0]; # vidi help(odeint), pod output  
  
y_exact = t - 1 + 2*np.exp(-t); # egzaktno rješenje
```

SciPy: rješavanje ODJ (2)

Korištenjem funkcije `odeint` možemo rješavati ODJ.

```
# Nastavak...slika koja uspoređuje izračunato s egzaktnim rješenjem.
[fig, ax] = plt.subplots( 1, 2, figsize=(12,3) );
ax[0].plot( t, y, 'r', label="odeint" );
ax[0].plot( t, y_exact, 'b:', label="exact" );
ax[0].set_xlabel( 't' );
ax[0].set_ylabel( 'y' );
ax[0].legend();
ax[1].plot( t, abs(y-y_exact), label="greška" );
ax[1].set_xlabel( 't' );
ax[1].set_ylabel( 'y' );
ax[1].legend();
```



Animacije pomoću Matplotlib

Jedan od načina kako raditi animacije pomoću Matplotlib unutar Jupyter Lab-a.

```
%matplotlib inline
import matplotlib.pyplot as plt;
plt.rcParams["animation.html"] = "jshtml";

import matplotlib.animation;
import numpy as np;

x = np.linspace(0, 2*np.pi);
y = np.sin(x);

[fig, ax] = plt.subplots();
[L,] = ax.plot( [0, 2*np.pi], [-1, 1] );

def animate( i ):
    # Funkcija kaže što treba napraviti u i-tom koraku animacije:
    # u plot-u L prikaži samo prvih i koordinata iz x-a i y-a.
    L.set_data( x[:i], y[:i] );

ani = matplotlib.animation.FuncAnimation( fig, animate, frames=len(x) );
plt.close();
ani
```