

Rešavanje sistema ODJ

Mnogi realni problemi, kao npr. trenutno aktuelno širenja virusa u pouplaciji, modeluju se pomoću sistema diferencijalnih jednačina.

Sistemi diferencijalnih jednačina se takođe koriste pri rešavanju ODJ reda višeg od 1.

Iz tih razloga, prvi deo današnjeg predavanja posvećen je numeričkim metodama za rešavanja sistema ODJ.

Suština sistema ODJ je da imamo više ODJ koje modeluju fenomene koji zavise jedan od drugog, pa se zato jedanične moraju rešavati simultano (zajedno) kao sistem.

Na primer, Lotka-Volterra model populacija predatora i plena predstavlja sistem od dve jedanične koje se formiraju po sledećim pretpostavkama:

- Plen raste stopom koja je proporcionalna trenutnoj populaciji plena, ako nema predatora.
- Populacija predatora smanjuje se stopom koja je proporcionalna trenutnoj populaciji predatora, ako nema plena.
- Broj susreta predatora i plena proporcionalan je proizvodu njihovih polulacija.
- Svaki susret predatora i plena povećava polulaciju predatora i smanjuje populaciju plena.

Na osnovu prethodne 4 pretpostavke foriran je sledeći sistem ODJ:

$$\begin{aligned}y_1' &= a y_1 - b y_1 y_2 \\ y_2' &= c y_1 y_2 - d y_2\end{aligned}$$

gde je $a > 0$, stopa rasta plena, $b > 0$ stopa opadanja plena pri intreakciji sa predatorom, $c > 0$ stopa porasta predatora pri interakciji sa plenom, i $d > 0$ stopa smanjenja predatora.

Vidimo da, da populacija plena u nekom trenutku zavisi od populacije predatora i obrnuto, pa jednačine ne možemo da rešavamo odvojeno.

Sistem jednačina Lotka-Volterra modela se u praksi najčešće rešava baš numeričkim metodama zato što su jednačine nelinearne.

Numeričke metode za rešavanje sistema ODJ

Sistema ODJ rešavamo pomoću svih numeričkih metoda za rešavanje ODJ koje smo učili, samo što sada te metode primenjujemo na sisteme od više jednačina istovremeno.

Priklimo primena numeričkih metoda, sisteme ODJ reprezentujemo pomoću vektora jedaničina i vektora početnih uslova:

$$F(x, Y) = \begin{bmatrix} f_1(x, y) \\ f_2(x, y) \\ \vdots \\ f_n(x, y) \end{bmatrix} \quad Y_0 = Y(x_0) = \begin{bmatrix} y_1(x_0) \\ y_2(x_0) \\ \vdots \\ y_n(x_0) \end{bmatrix}$$

Pokazaćemo sada primer jednog sistema za Lotka-Volterra model na koji ćemo primeniti Ojlerov metod.

$$\begin{aligned}y_1' &= 0.5y_1 - 0.2y_1y_2 = y_1(0.5 - 0.2y_2) \\ y_2' &= 0.004y_1y_2 - 0.4y_2 = y_2(0.04y_1 - 0.4) \\ y_1(0) &= 200, y_2(0) = 30\end{aligned}$$

$$F(x, y) = \begin{bmatrix} y_1(0.5 - 0.2y_2) \\ y_2(0.04y_1 - 0.4) \end{bmatrix} \quad Y(0) = \begin{bmatrix} 200 \\ 30 \end{bmatrix}$$

Ojlerov metod za sisteme ODJ:

Ojlerov metod za rešavanje sistema ODJ funkcioniše na istin način kao kada rešavamo jednu jednačinu, samo ga sada primenjujemo na vektore jedaničina. Iz tog razloga formulu prikazujemo pomoću vektorskih oznaka:

$$Y(x_{i+1}) = Y(x_i) + hF(x_i, Y_i)$$

Primenjujemo sada Ojlerov metod na sistem za predator-plen model. Koristimo $h = 0.1$:

$$Y(0+0.1) = Y(0, Y_0) + 0.1F(0, Y_0)$$

$$Y_1 = Y(0.1) = \begin{bmatrix} 200 \\ 30 \end{bmatrix} + 0.1 \begin{bmatrix} y_1(0.5 - 0.2y_2) \\ y_2(0.04y_1 - 0.4) \end{bmatrix}$$

$$Y_1 = Y(0.1) = \begin{bmatrix} 200 \\ 30 \end{bmatrix} + 0.1 \begin{bmatrix} 200(0.5 - 0.2 \cdot 30) \\ 30(0.04 \cdot 200 - 0.4) \end{bmatrix}$$

$$Y_1 = Y(0.1) = \begin{bmatrix} 200 \\ 30 \end{bmatrix} + 0.1 \begin{bmatrix} -2 \\ 1.2 \end{bmatrix}$$

$$Y_1 = Y(0.1) = \begin{bmatrix} 198 \\ 31.2 \end{bmatrix}$$

Urađićemo sada još jedan korak.

$$Y_2 = Y(0.1 + 0.1) = Y_1 + 0.1F(0.1, Y_1)$$

$$Y_2 = Y(0.2) = \begin{bmatrix} 198 \\ 31.2 \end{bmatrix} + 0.1 \begin{bmatrix} y_2(0.5 - 0.2y_2) \\ y_2(0.04y_1 - 0.4) \end{bmatrix}$$

$$Y_2 = Y(0.2) = \begin{bmatrix} 198 \\ 31.2 \end{bmatrix} + 0.1 \begin{bmatrix} 198(0.5 - 0.2 \cdot 31.2) \\ 31.2(0.04 \cdot 198 - 0.4) \end{bmatrix}$$

$$Y_2 = Y(0.2) = \begin{bmatrix} 195.545 \\ 32.423 \end{bmatrix}$$

Napisaćemo sada kod za Ojlerov metod za rešavanje sistema ODJ. Razlika u odnosu na funkciju koja se primenjuje na jednu jednačinu je u tome da sada radimo sa matricama, a ne vektorima.

Matrica Y kao prvi kolonu sadrži vektor početnih rešenja, dok ostale kolone popunjavamo pomoću formule za Ojlerov metod.

Još jedna razlika u odnosu na kod koji smo pisali na prošlom predavanju je da sada nemamo fiksnu krajnju tačku već zadajemo broj koraka primene metoda.

```
In [70]: import numpy as np

def ojler_sistem(x0,y0,h,odj,br_koraka):
    n = len(y0)
    Y = np.zeros([n,br_koraka+1])
    X = np.zeros([n,br_koraka+1])
    Y[:,0] = y0
    X[:,0] = x0
    for i in range(1,br_koraka+1):
        X[:,i] = X[:,i-1] + h * odj(X[:,i-1], Y[:,i-1])
        Y[:,i] = Y[:,i-1] + h * odj(X[:,i-1], Y[:,i-1])
    return [Y,X]
```

Zadaćemo sada sistem kao listu koji ima onoliko elemenata koliko ima jedaničina u sistemu. Svaka jedaničina je jedan element tog vektora.

```
In [71]: def odj_sistem(x,y):
    z1 = y[0] * (0.5 - 0.02 * y[1])
    z2 = y[1] * (0.004 * y[0] - 0.4)
    return np.array([z1,z2])
```

Pozivamo sada funkciju koju smo napisali. Početne vrednosti su sada vektori.

```
In [72]: x0=np.array([0,0])
y0=np.array([200,30])
h=0.1
br_koraka=10
[Y,X]=ojler_sistem(x0,y0,h,odj_sistem,br_koraka)

print(X)
print(Y)
print(Y.shape)
```

```
[[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
 [0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]]
[[200.  198.  195.5448  192.64172626 189.3043312
 185.55247451 181.41219717 176.91537463 172.09914879 167.00515239
 161.67855488]
 [ 30.  31.2  32.42304  33.66218115  34.90959018
 36.15662122  37.39393658  38.6116656  39.79959789  40.94740474
 42.04487958]]
(2, 11)
```

Povećavamo broj koraka i crtamo grafik rešenja.

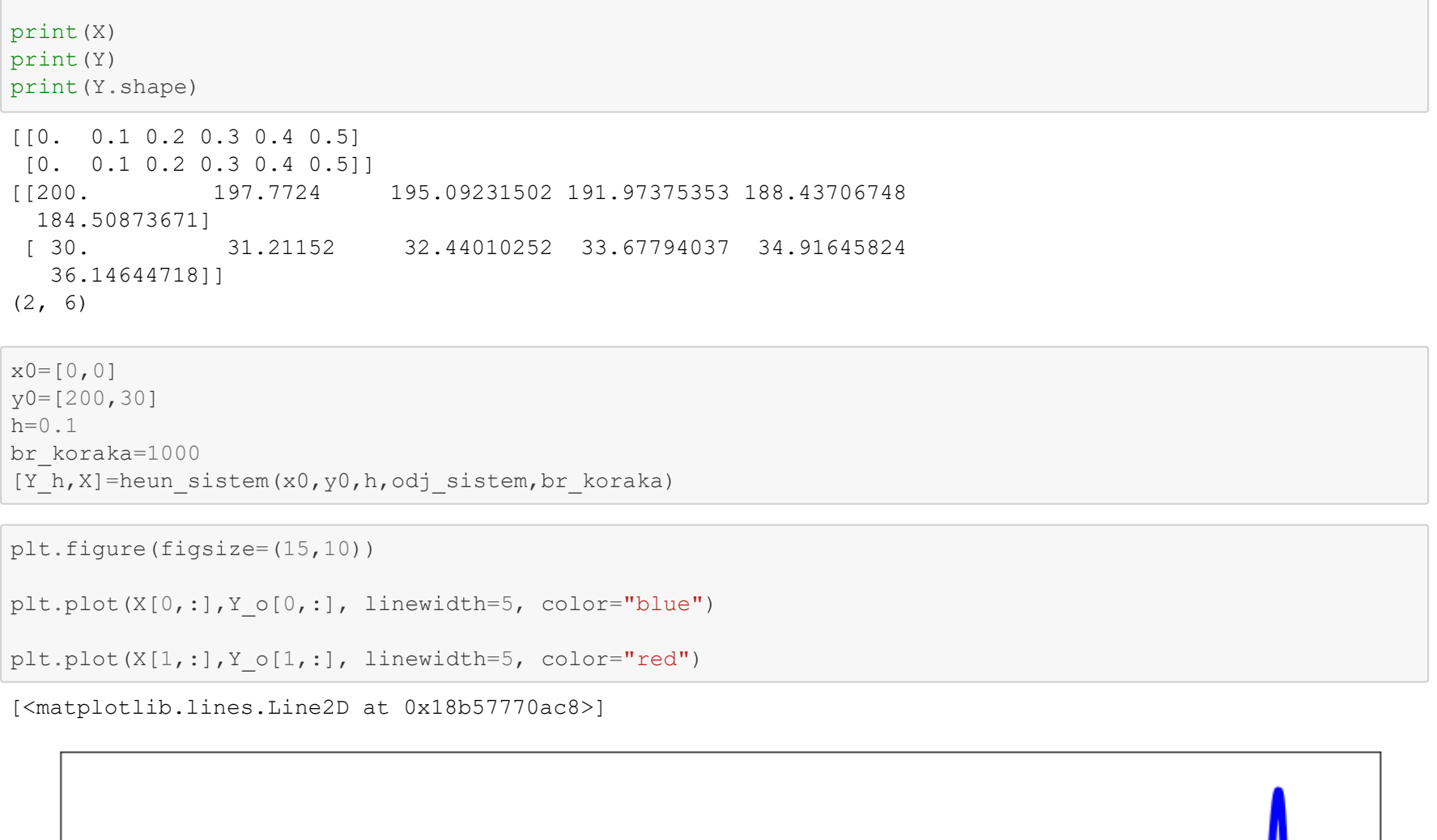
```
In [73]: x0=[0,0]
y0=[200,30]
h=0.1
br_koraka=1000
[Y_o,X]=ojler_sistem(x0,y0,h,odj_sistem,br_koraka)
```

Prikazujemo sada rešenja obe jednačine na jednom grafiku. U zavisnosti od početnih uslova možemo da vidimo šta će se dogoditi sa populacijama predatora i plena kroz vreme.

```
In [74]: import matplotlib.pyplot as plt

plt.figure(figsize=(15,10))

plt.plot(X[0,:],Y_o[0,:], linewidth=5, color="blue") #plen
plt.plot(X[1,:],Y_o[1,:], linewidth=5, color="red") #predator
```



Na isti način na koji smo implementirali Ojlerov metod mogu se implementirati sve druge numeričke metode za rešavanje pojedinačnih jednačina.

U nastavku je prikazan primer primene Heunovog metoda, a nakon toga i kod za Heunov metod.

$$Y(x_{i+1})^0 = Y(x_i) + hF(x_i, Y_i)$$

$$Y(x_{i+1}) = Y(x_i) + \frac{1}{2}h(F(x_i, Y_i) + F(x_{i+1}, Y_i^0))$$

Primenjujemo sada Heunov metod na sistem za predator-plen model:

$$F(x, y) = \begin{bmatrix} y_1(0.5 - 0.2y_2) \\ y_2(0.04y_1 - 0.4) \end{bmatrix} \quad Y(0) = \begin{bmatrix} 200 \\ 30 \end{bmatrix}$$

Koristimo $h = 0.1$:

$$Y(0+0.1)^0 = Y(0, Y_0) + 0.1F(0, Y_0)$$

$$Y(0.1)^0 = \begin{bmatrix} 200 \\ 30 \end{bmatrix} + 0.1 \begin{bmatrix} 200(0.5 - 0.2 \cdot 30) \\ 30(0.04 \cdot 200 - 0.4) \end{bmatrix} = \begin{bmatrix} 198 \\ 31.2 \end{bmatrix}$$

$$Y_1 = Y(0.1)^1 = \begin{bmatrix} 200 \\ 30 \end{bmatrix} + \frac{1}{2} \cdot 0.1 \left[\begin{bmatrix} 200(0.5 - 0.2 \cdot 30) \\ 30(0.04 \cdot 200 - 0.4) \end{bmatrix} + \begin{bmatrix} 198(0.5 - 0.2 \cdot 31.2) \\ 31.2(0.04 \cdot 198 - 0.4) \end{bmatrix} \right] = \begin{bmatrix} 197.772 \\ 31.212 \end{bmatrix}$$

```
In [75]: def heun_sistem(x0,y0,h,odj,br_koraka):
    n = len(y0)
    Y = np.zeros([n,br_koraka+1])
    X = np.zeros([n,br_koraka+1])
    Y[:,0] = y0
    X[:,0] = x0
    for i in range(1,br_koraka+1):
        X[:,i] = X[:,i-1] + h * odj(X[:,i-1], Y[:,i-1])
        Y[:,i] = Y[:,i-1] + h * odj(X[:,i-1], Y[:,i-1])
        Y[:,i] = Y[:,i-1] + h * ( odj(X[:,i-1], Y[:,i-1]) + odj(X[:,i], Y[:,i]) ) / 2
    return [Y,X]
```

```
In [76]: def odj_sistem(x,y):
    z1 = y[0] * (0.5 - 0.02 * y[1])
    z2 = y[1] * (0.004 * y[0] - 0.4)
    return np.array([z1,z2])
```

```
In [77]: x0=[0,0]
y0=[200,30]
h=0.1
br_koraka=5
[Y,X]=heun_sistem(x0,y0,h,odj_sistem,br_koraka)
```

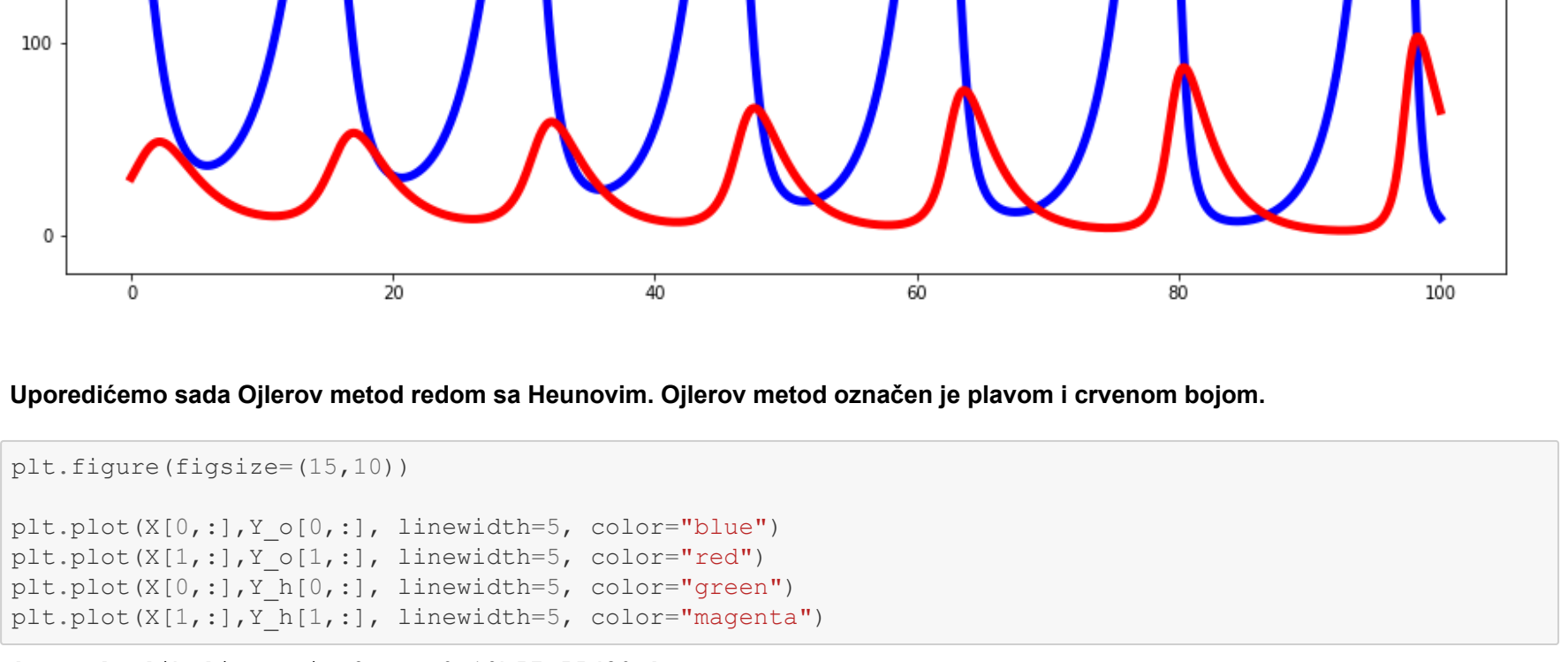
```
print(X)
print(Y)
print(Y.shape)
```

```
[[0.  0.1 0.2 0.3 0.4 0.5]
 [0.  0.1 0.2 0.3 0.4 0.5]]
[[200.  197.7724  195.99231502 191.97375353 188.43706748
 184.50873671]
 [ 30.  31.21152  32.44010252  33.67794037  34.91645824
 36.14644718]]
(2, 6)
```

```
In [78]: x0=[0,0]
y0=[200,30]
h=0.1
br_koraka=1000
[Y_h,X]=heun_sistem(x0,y0,h,odj_sistem,br_koraka)
```

```
In [79]: plt.figure(figsize=(15,10))

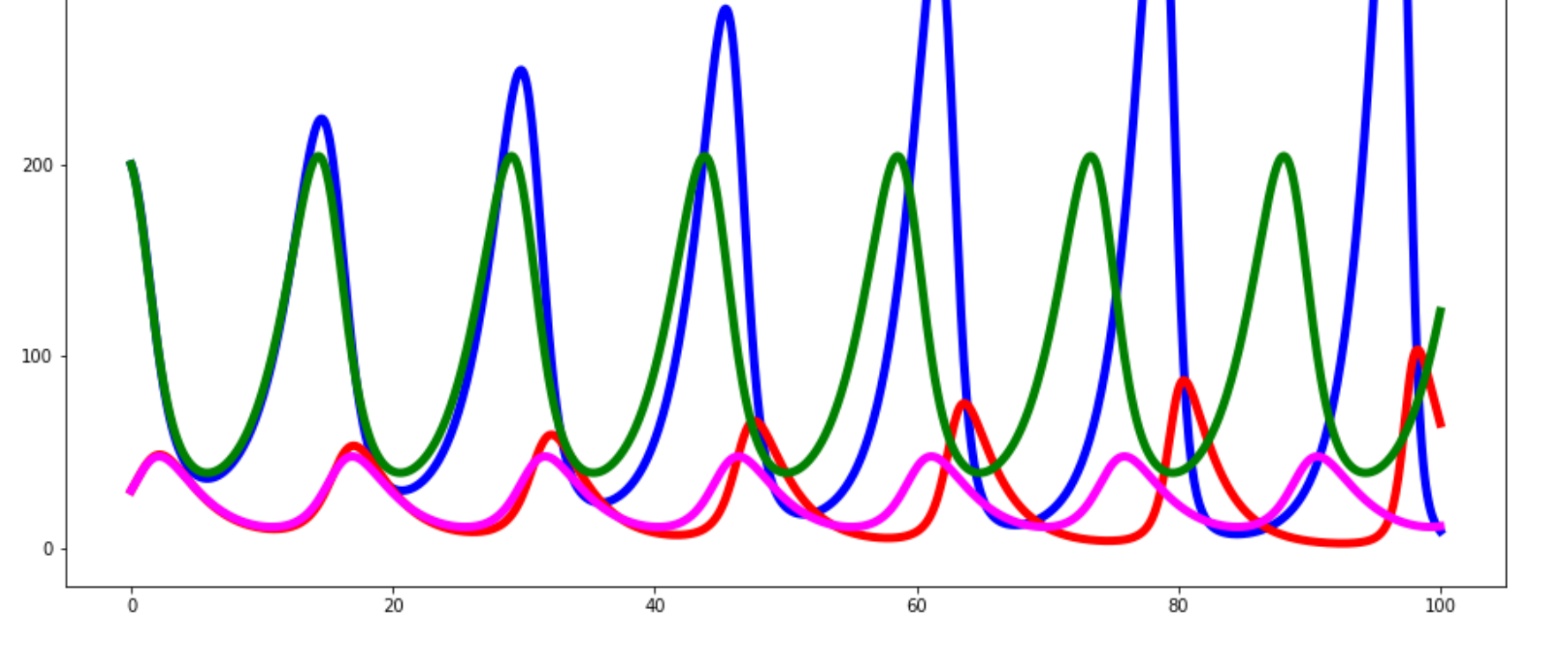
plt.plot(X[0,:],Y_o[0,:], linewidth=5, color="blue")
plt.plot(X[1,:],Y_o[1,:], linewidth=5, color="red")
```



Uporedićemo sada Ojlerov metod redom sa Heunovim. Ojlerov metod označen je plavom i crvenom bojom.

```
In [84]: plt.figure(figsize=(15,10))

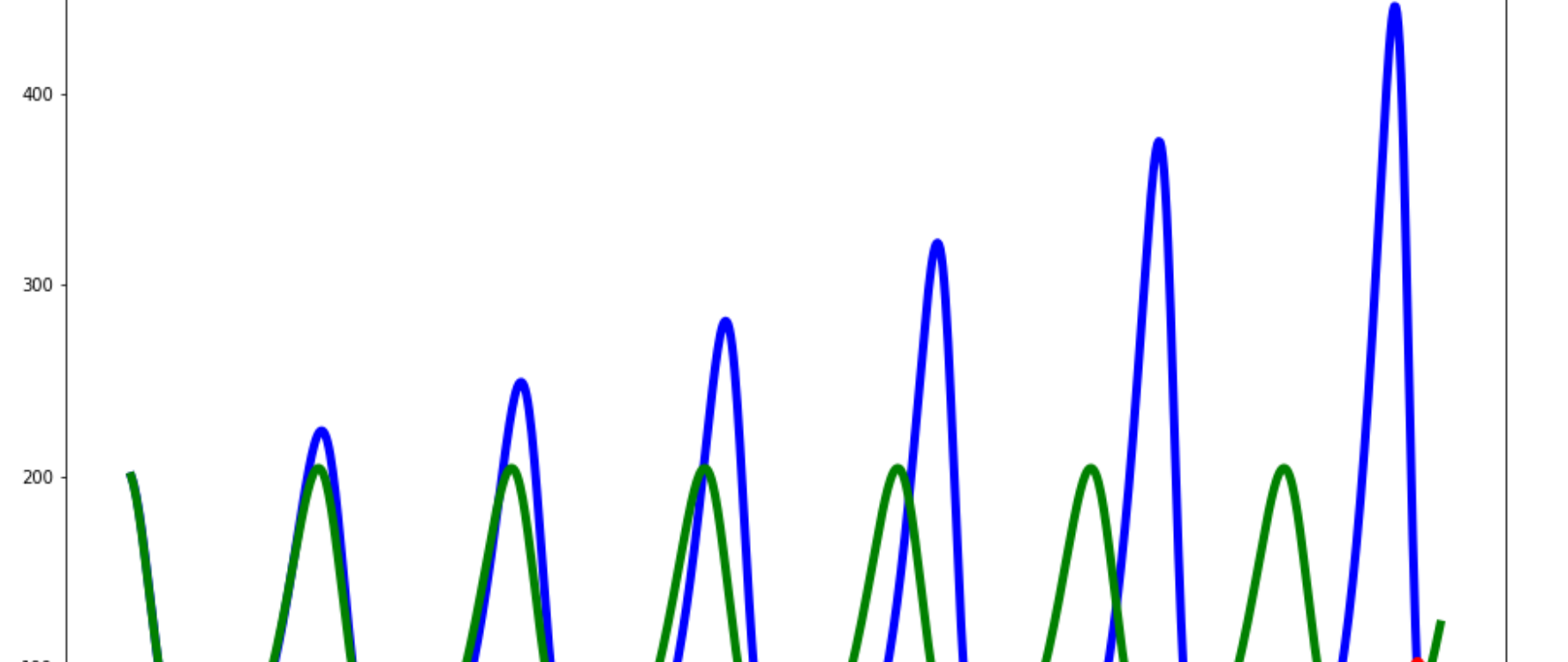
plt.plot(X[0,:],Y_o[0,:], linewidth=5, color="blue")
plt.plot(X[1,:],Y_o[1,:], linewidth=5, color="red")
plt.plot(X[0,:],Y_h[0,:], linewidth=5, color="green")
plt.plot(X[1,:],Y_h[1,:], linewidth=5, color="magenta")
```



Uporedićemo sada Ojlerov metod redom sa RK4. Ojlerov metod označen je plavom i crvenom bojom.

```
In [85]: plt.figure(figsize=(15,10))

plt.plot(X[0,:],Y_o[0,:], linewidth=5, color="blue")
plt.plot(X[1,:],Y_o[1,:], linewidth=5, color="red")
plt.plot(X[0,:],Y_r[0,:], linewidth=5, color="green")
plt.plot(X[1,:],Y_r[1,:], linewidth=5, color="magenta")
```



Rešenje dobijeno Ojlerovom metodom razlikuje se od rešenja dobijenog Heunovim i RK4 metodom. Sa obzirom da Ojlerov metod vrši najgublju procenu sledeće tačke, on generarno daje loše rezultate kada je rešenje funkcija koja osciluje, što je naš slučaj kod ovog predator-plen modela.

Primenićemo sada Ojlerov metod na još jedan primer.

$$\begin{aligned}y_1' &= y_2 \\ y_2' &= 1 - y_1 \\ y_1(0) &= 1, y_2(0) = -2\end{aligned}$$

$$F(x, y) = \begin{bmatrix} y_2 \\ 1 - y_1 \end{bmatrix} \quad Y(0) = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

Kod ovog primera izvod prve funkcije koju tražimo je druga funkciju koju tražimo. Takve sisteme dobijamo kada rešavamo ODJ reda većeg od 1 pa iz tog razloga obrađujemo baš ovakav primer.

$$Y(0+0.1) = Y(0, Y_0) + 0.1F(0, Y_0)$$

$$Y_1 = Y(0.1) = \begin{bmatrix} -1 \\ 1 \end{bmatrix} + 0.1 \begin{bmatrix} 1 \\ 1 - (-1) \end{bmatrix} = \begin{bmatrix} -0.9 \\ 1.2 \end{bmatrix}$$

$$Y_1 = Y(0.2) = \begin{bmatrix} -0.9 \\ 1.2 \end{bmatrix} + 0.1 \begin{bmatrix} 1.2 \\ 1 - (-0.9) \end{bmatrix} = \begin{bmatrix} -0.78 \\ 1.39 \end{bmatrix}$$

```
In [86]: def odj_sistem(x,y):
    z1 = y[1]
    z2 = 1-y[0]
    return np.array([z1,z2])
```

```
In [87]: x0=[0,0]
y0=[-1,1]
h=0.1
br_koraka=50
[Y,X]=ojler_sistem(x0,y0,h,odj_sistem,br_koraka)
```

```
In [88]: plt.figure(figsize=(15,10))

plt.plot(X[0,:],Y_o[0,:], linewidth=5, color="blue") #funkcija y1(x)
plt.plot(X[1,:],Y_o[1,:], linewidth=5, color="red") #prvi izvod funkcije y1(x)
```

Out [88]:

Prethodni sistem u stvari predstavlja diferencijalnu jednačinu drugog reda sledećeg oblika:

$$\ddot{y} + y = 1, \quad y(0) = -1, \dot{y}(0) = 1$$

Funkcija $y_1(x)$, označena plavom bojom na grafiku iznad, predstavlja rešenje date diferencijalne jednačine drugog reda, dok funkcija $y_2(x)$, označena crvenom bojom na grafiku iznad, predstavlja prvi izvod funkcije $y_1(x)$.

Sa grafika se vidi da prvi izvod počinje da bude negativan negde između 2 i 3, pa tada funkcija kreće da opada. Takođe se vidi da je ekstrem (u ovom slučaju maksimum) funkcije u tački u kojoj prvi izvod ima vrednost 0.

Rešavanje ODJ višeg reda

Videli smo da se ODJ višeg reda može rešavati kao sistem ODJ. U nastavku ćemo pokazati na koji način se ODJ višeg reda konvertuje u sistem ODJ.

Navodimo prvo korake procesa konverzije, pa onda nekoliko primera.

1. Uvodimo nove zavisne promenljive. Uzimamo zavisnu promenljivu i sve njene izvode od jednog manje reda od najvećeg. Zamjenjujemo ih novim promenljivama.

2. Zapišujemo date diferencijalne jednačine u drugom obliku upotrebom novih zavisnih promenljivih tako što dodajemo po jednu jednačinu za svaku novu zavisnu promenljivu.

Primer 1.

$$\ddot{y} + y = 1, \quad y(0) = -1, \dot{y}(1) = 1$$

Nove zavisne promenljive su:

$$\begin{aligned}y_1 &= y \\ y_2 &= \dot{y}\end{aligned}$$

Prva nova jednačina je:

$$\dot{y}_1 = y_2$$

Druga nova jednačina je:

$$\dot{y}_2 + y_1 = 1 \implies \dot{y}_2 = 1 - y_1$$

Dobijeni sistem u matricnom obliku:

$$F(x, y) = \begin{bmatrix} y_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} y_2 \\ 1 - y_1 \end{bmatrix} \quad Y(0) = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

Primer 2.

$$\ddot{y} + 2\ddot{y} + 2\dot{y} + 8y = 1, \quad y(0) = 4, \dot{y}(0) = 1, \ddot{y}(0) = 9$$

Nove zavisne promenljive su:

$$\begin{aligned}y_1 &= y \\ y_2 &= \dot{y} \\ y_3 &= \ddot{y}\end{aligned}$$

Prva nova jednačina je:

$$\dot{y}_1 = y_2$$

Prva nova jednačina je:

$$\dot{y}_2 = y_3$$

Druga nova jednačina je:

$$y_3 + 2y_3 + 2y_2 + 8y_1 = 1 \implies \dot{y}_3 = 1 - 2y_3 - 2y_2 - 8y_1$$

Dobijeni sistem u matricnom obliku:

$$F(x, y) = \begin{bmatrix} y_2 \\ y_3 \\ y_3 \end{bmatrix} = \begin{bmatrix} y_2 \\ y_3 \\ 1 - 2y_3 - 2y_2 - 8y_1 \end{bmatrix} \quad Y(0) = \begin{bmatrix} 4 \\ 1 \\ 9 \end{bmatrix}$$

Dakle, videli smo na koji način se vrši konverzija ODJ višeg reda. U nastavku rešavamo prethodni Primer 2, pomoću RK4 metode.

```
In [89]: def odj_sistem(x,y):
    z1 = y[1]
    z2 = y[2]
    z3 = 1 - 2 * y[2] - 2 * y[1] - 8 * y[0]
    return np.array([z1,z2,z3])
```

```
In [90]: x0=[0,0,0]
y0=[4,1,9]
h=0.1
br_koraka=50
[Y,X]=RungeKutta4_sistem(x0,y0,h,odj_sistem,br_koraka)
```

```
In [91]: plt.figure(figsize=(15,10))

plt.plot(X[0,:],Y_o[0,:], linewidth=5, color="blue")
plt.plot(X[1,:],Y_o[1,:], linewidth=5, color="red")
plt.plot(X[2,:],Y_o[2,:], linewidth=5, color="green")
```

Out [91]:

Funkcija $y_1(x)$, označena plavom bojom na grafiku iznad, predstavlja rešenje date diferencijalne jednačine trećeg reda. Funkcija $y_2(x)$, označena crvenom bojom, je prvi izvod funkcije $y_1(x)$. Funkcija $y_3(x)$, označena zelenom bojom je drugi izvod funkcije $y_1(x)$.

Sa grafika na primer možemo da vidimo da je funkcija konvekсна tamo gde je drugi izvod veći od 0, a konkavna tamo gde je manji od 0.

Problem graničnih uslova (PGU)

Za razliku od problema početnih uslova, kod problema graničnih uslova početni uslovi zadati su u više tačka, tipično na granicama nekog regiona na kome se rešava diferencijalna jednačina.

Na ovom kursu razmatraćemo samo probleme kod kojih su početni uslovi zadati u dve tačke.

Na primer, sledeća jednačina drugog reda (koju smo rešavali gore) može se rešavati tako da zahtevamo da rešenje (funkcija $y(x)$) u tački 1 ima vrednost 2:

$$\ddot{y} + y = 1, \quad y(0) = -1, \dot{y}(1) = 2$$

Primitet da sada nemamo dat početni uslov za prvi izvod funkcije.

Metod gađanja

Prvi metod za rešavanje PGU koji ćemo pokazati je metod gađanja (*shooting method*).

Kod ovog metoda meta nam je drugi granični uslov, a u nastavku ćemo pokazati kako gađamo, tj. kako nišanim.

Korake metoda gađanja objašnjavamo na primeru. Rešavamo sledeći problem:

$$\ddot{y} + y = 1, \quad y(0) = -1, \dot{y}(1) = 2$$

Prvi korak je konverzija u sistem ODJ. Konverziju smo već uradili gore, a dobijeni sistem u matricnom obliku je:

$$F(x, y) = \begin{bmatrix} y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} y_2 \\ 1 - y_1 \end{bmatrix} \quad Y(0) = \begin{bmatrix} -1 \\ ? \end{bmatrix}$$

Da li možemo primenimo neki od numeričkih metoda da rešimo sistem?

Očigledno da ne možemo jer nam nedostaje početni uslov za prvi izvod. Naš problem je zadat na takav način da umesto toga znamo vrednost funkcije $y(x)$ u nekoj drugoj tački.

Drugi korak metoda gađanja je zadavanje početnih uslova koji nedostaju. To je korak u kome nišanim. Ako gađamo prvi put, početni uslov biramo na slučajjan način. Recimo da smo odabrali vrednost 0:

$$\ddot{y} + y = 1, \quad y(0) = -1, \dot{y}_2(0) = 0$$

Rešavamo sada zadati sistem pomoću nekog numeričkog metoda, na primer RK4.

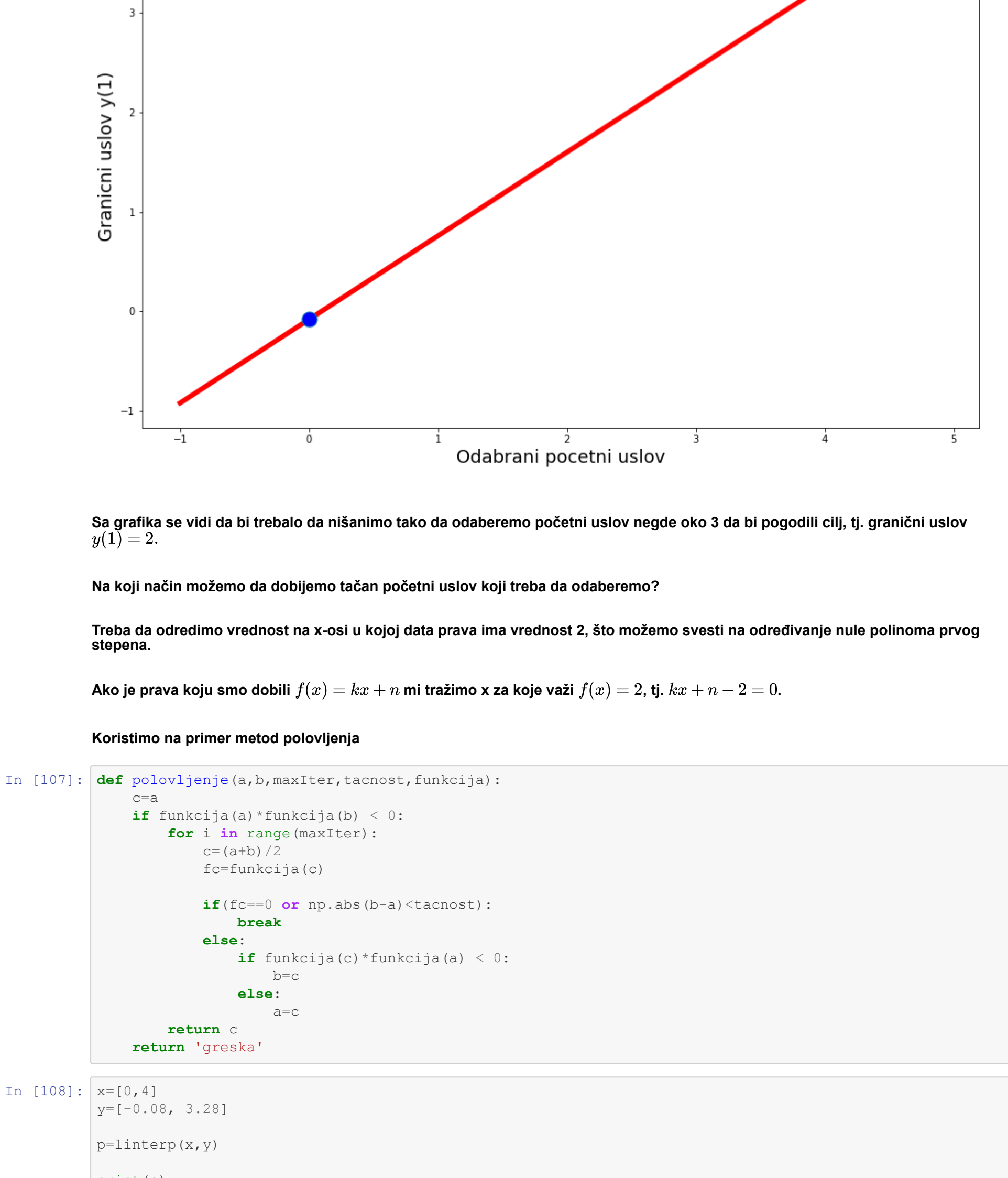
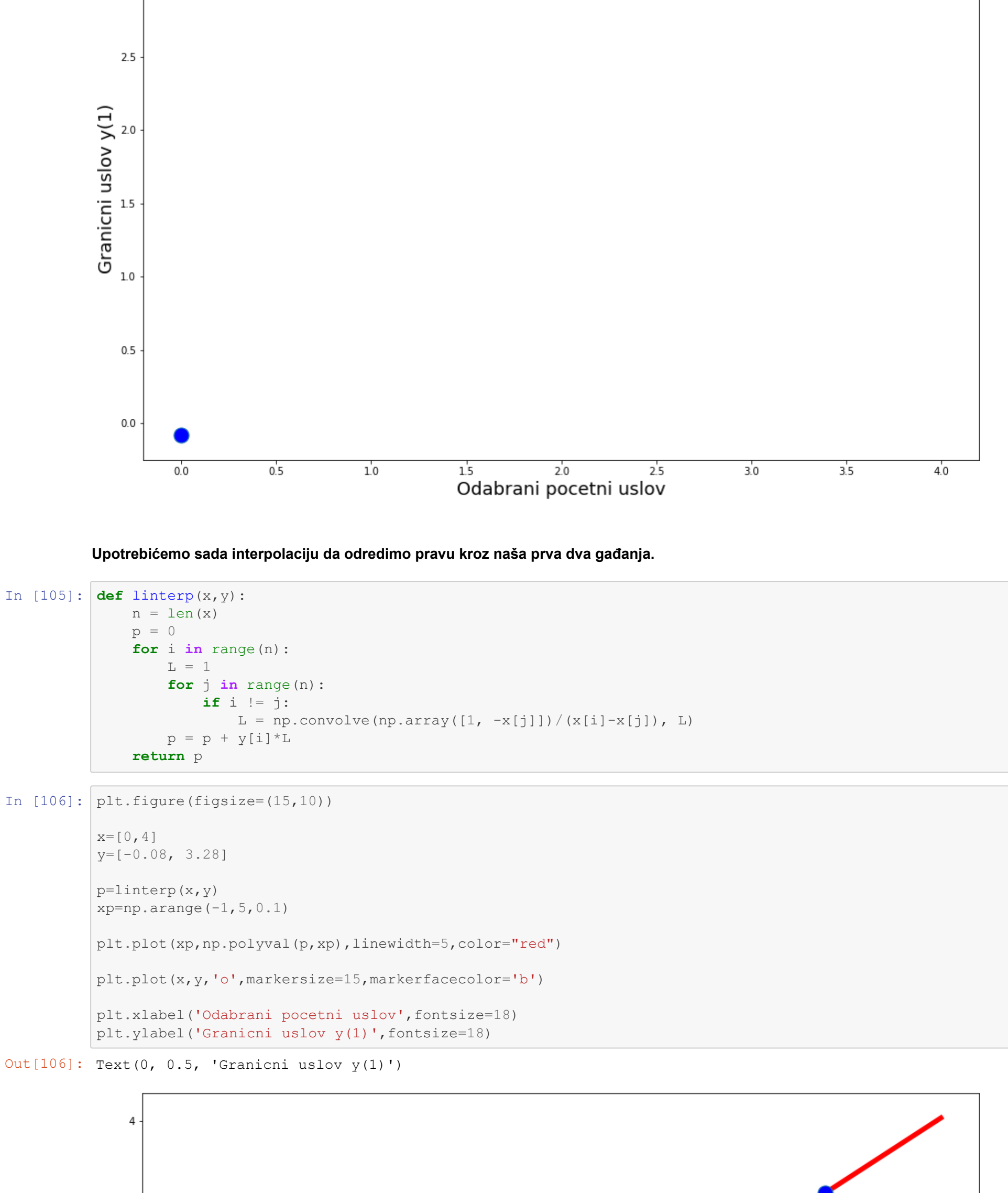
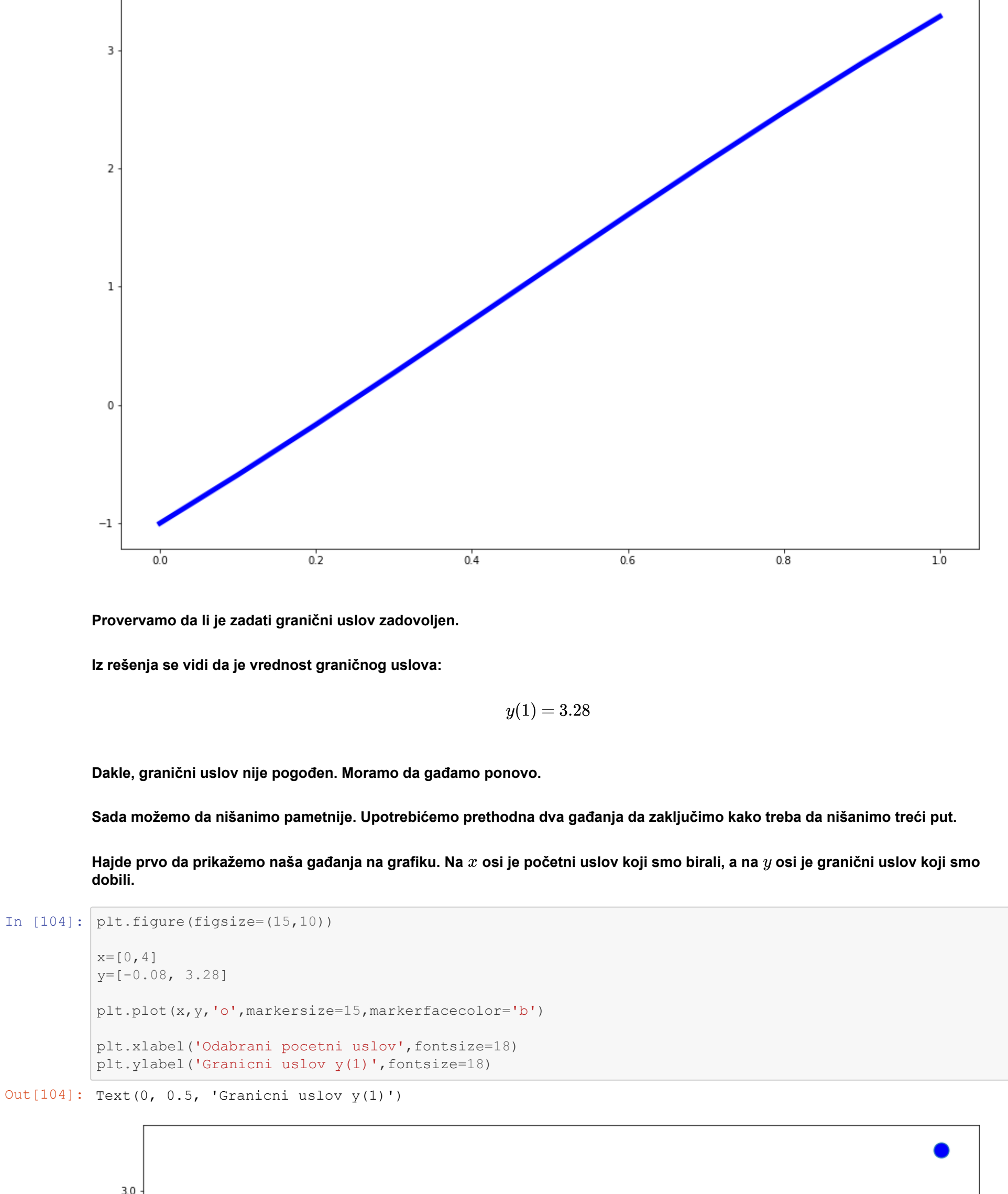
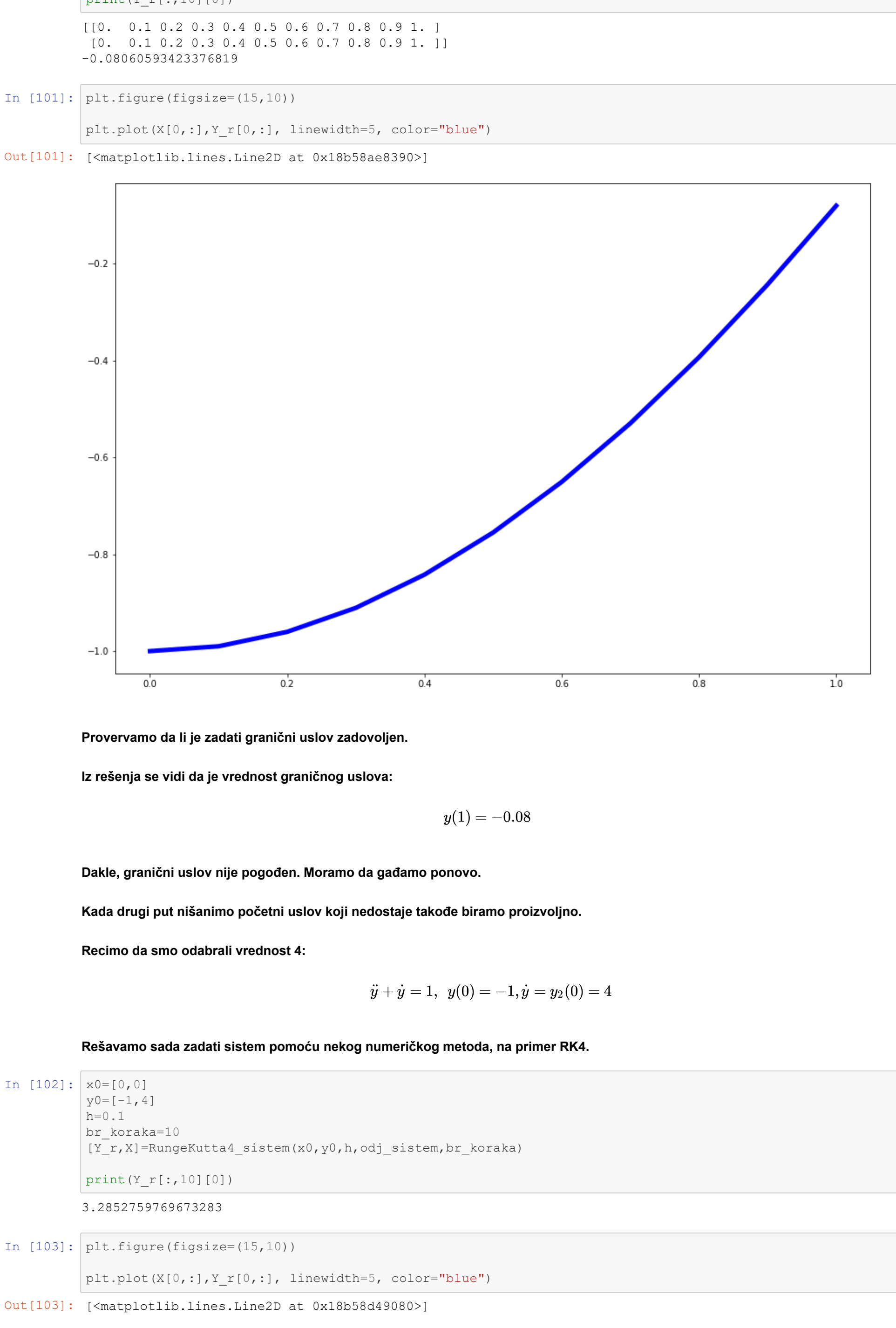
```
In [92]: def odj_sistem(x,y):
    z1 = y[1]
    z2 = y[2]
    return np.array([z1,z2])
```



```
In [100]: x0=[0,0]
y0=[-1,0]
h=0.1
br_koraka=10
[Y_r,X]=RungeKutta4_sistem(x0,y0,h,odj_sistem,br_koraka)

print(X)
#prijat(Y_r)
print(Y_r[:10])

[[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
 [0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]]
-0.0806059342376819
```



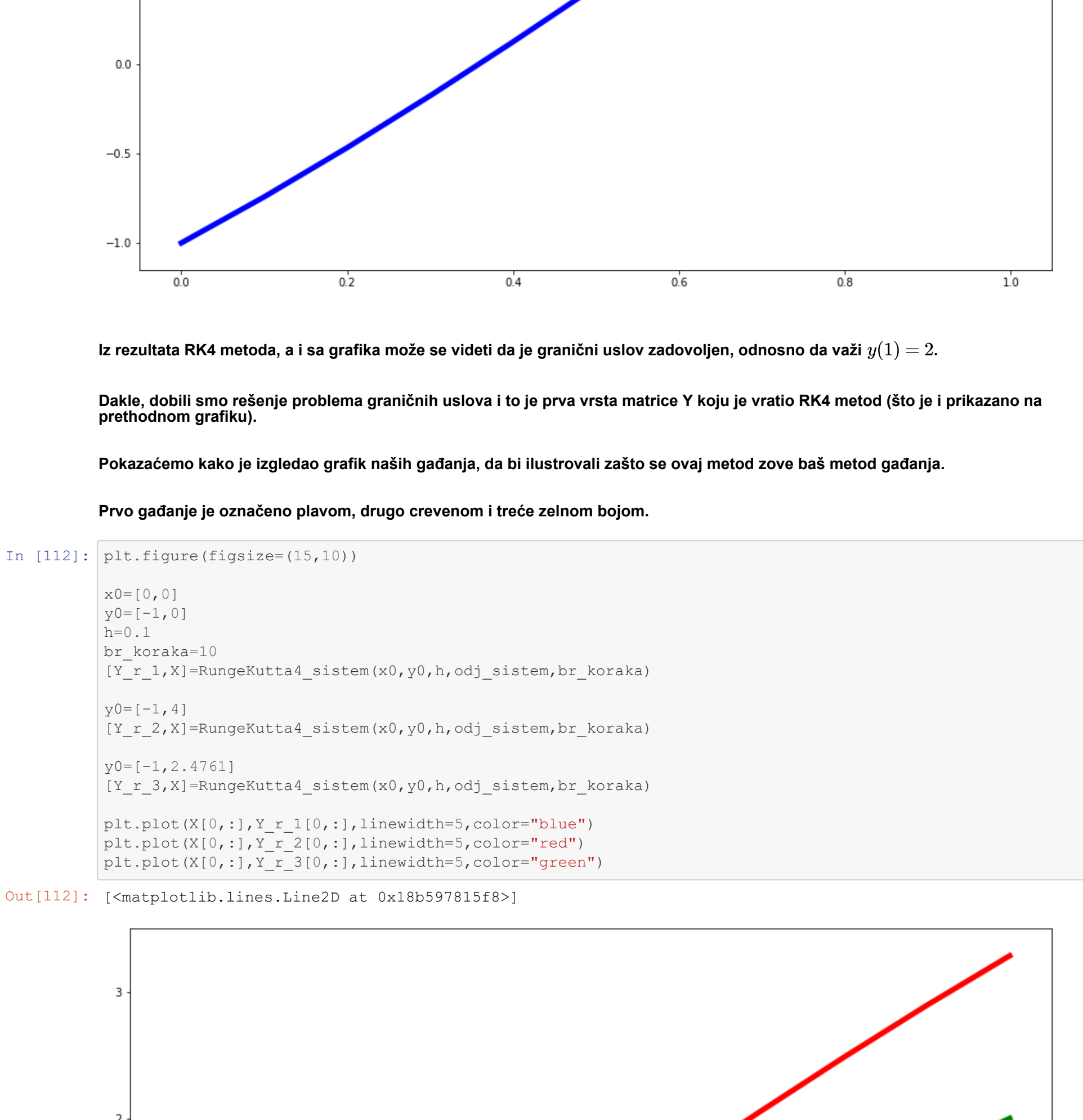
Sa grafika se vidi da bi trebalo da nisanimo tako da odaberemo početni uslov negde oko 3 da bi pogodili cilj, tj. granični uslov $y(1) = 2$.

Na koji način možemo da dobijemo tačan početni uslov koji treba da odaberemo?

Treba da odredimo vrednost na x-osi u kojoj data prava ima vrednost 2, što možemo svesti na određivanje nule polinoma prvog stepena.

Ako je prava koju smo dobili $f(x) = kx + n$ mi tražimo x za koji važi $f(x) = 2$, tj. $kx + n - 2 = 0$.

Koristimo na primer metod polovljenja

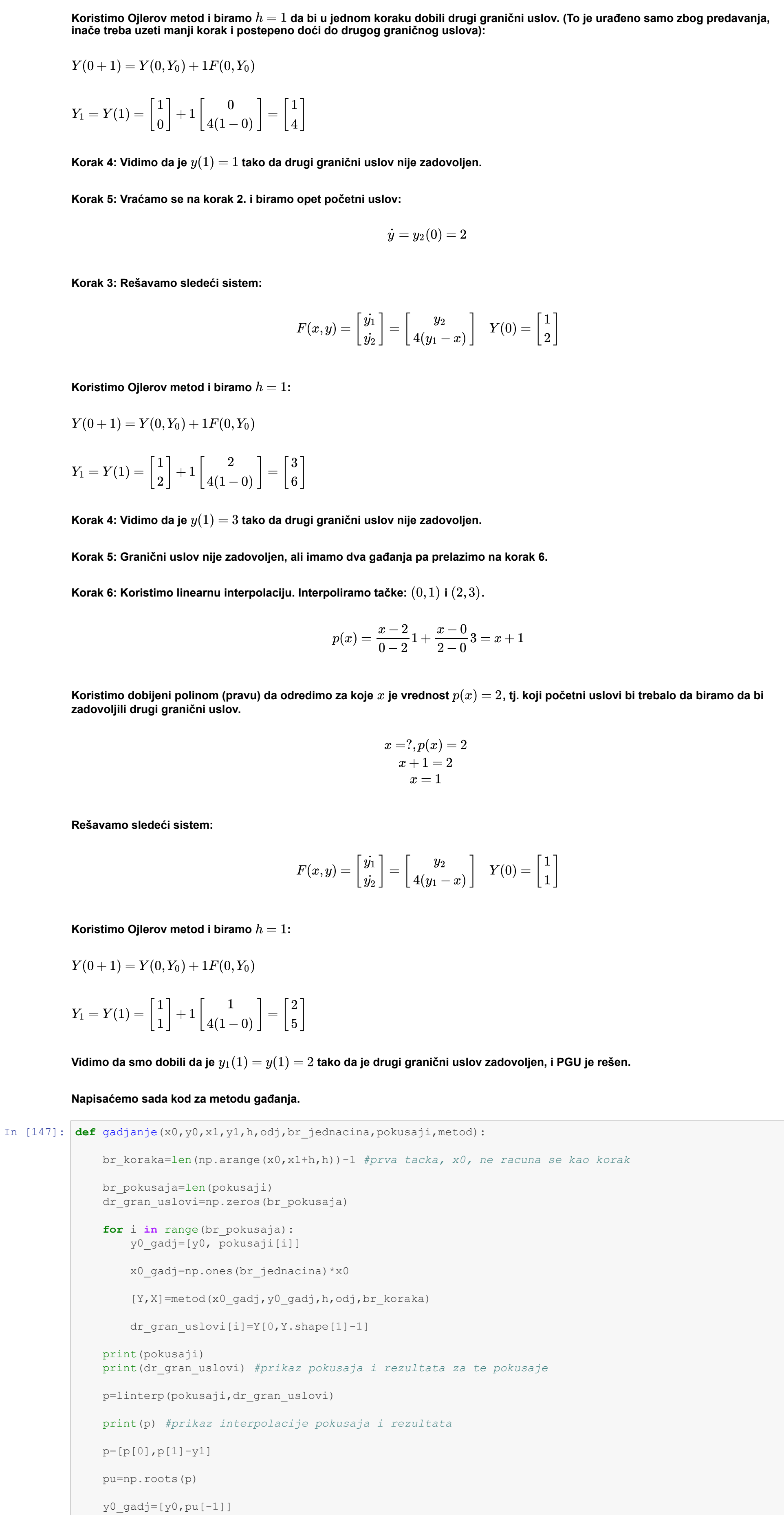


Iz rezultata RK4 metoda, a i sa grafikom može se videti da je granični uslov zadovoljen, odnosno da važi $y(1) = 2$.

Dakle, dobili smo rešenje problema graničnih uslova i to je prva vrstica Y koju je vratio RK4 metod (što je i prikazano na prethodnom grafiku).

Pokazaćemo kako je izgledao grafik naših gađanja, da bi ilustrovali zašto se ovaj metod zove baš metod gađanja.

Prvo gađanje je označeno plavom, drugo crvenom i treće zelenom bojom.



Sa grafika se vidi da smo gađali 3 puta. Svaki put je početni uslov $y(0) = -1$ bio isti, ali se početni uslov $\hat{y}(0)$ razlikovao. Trećim gađanjem (zeleni boja) pogodili smo metu $y(1) = 2$.

Algoritam metode gađanja

Sistematizovaćemo sada korake metode gađanja koje smo pokazali izad.

Dat nam je problem graničnih uslova.

Korak 1: Konvertujemo problem u sistem ODJ.

Korak 2: Na slučajan način birmo početni uslov koji nedostaje.

Korak 3: Rešavamo sistem ODJ nekom od numeričkih metoda.

Korak 4: Proveravamo da li je zadovoljen drugi granični uslov.

Korak 5: Ako je drugi granični uslov zadovoljen, završavamo algoritam. Ako nije, a nemamo bar dva gađanja, vraćamo se na korak 2, inače prelazimo na sledeći korak.

Korak 6: Ako imamo bar dva gađanja, odnosno dva odabira početnog uslova i dva rezultata za granični uslov, koristimo linearnu interpolaciju da odredimo koji tačno početni uslov treba da birmo.

Korak 7: Rešavamo sistem ODJ uz početni uslov iz koraka 6, i tako dobijamo rešenje.

Metoda gađanja - primer 2.

Primenićemo sada metodu gađanja na još jedan primer, ali ovog puta većinu koraka izvodimo ručno.

$$\ddot{y} - 4y + 4z = 0, \quad y(0) = 1, y(1) = 2$$

Korak 1: Kovertujemo ODJ drugog reda u sistem:

$$F(x,y) = \begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} y_2 \\ 4(y_1 - x) \end{bmatrix} \quad Y(0) = \begin{bmatrix} 1 \\ ? \end{bmatrix}$$

Korak 2: Birmo početni uslov:

$$\hat{y} = y_2(0) = 0$$

Korak 3: Rešavamo sledeći sistem:

$$F(x,y) = \begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} y_2 \\ 4(y_1 - x) \end{bmatrix} \quad Y(0) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Koristimo Ojlerov metod i birmo $h = 1$ da bi u jednom koraku dobili drugi granični uslov. (To je urađeno samo zbog predavanja, inače treba uzeti manji korak i postepeno doći do drugog graničnog uslova):

$$Y(0+1) = Y(0, Y_0) + 1F(0, Y_0)$$
$$Y_1 = Y(1) = \begin{bmatrix} 1 \\ 0 \end{bmatrix} + 1 \begin{bmatrix} 0 \\ 4(1 - 0) \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \end{bmatrix}$$

Korak 4: Vidimo da je $y(1) = 1$ tako da drugi granični uslov nije zadovoljen.

Korak 5: Vraćamo se na korak 2, i birmo opet početni uslov:

$$\hat{y} = y_2(0) = 2$$

Korak 3: Rešavamo sledeći sistem:

$$F(x,y) = \begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} y_2 \\ 4(y_1 - x) \end{bmatrix} \quad Y(0) = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Koristimo Ojlerov metod i birmo $h = 1$:

$$Y(0+1) = Y(0, Y_0) + 1F(0, Y_0)$$
$$Y_1 = Y(1) = \begin{bmatrix} 1 \\ 2 \end{bmatrix} + 1 \begin{bmatrix} 2 \\ 4(1 - 0) \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \end{bmatrix}$$

Korak 4: Vidimo da uslov je $y(1) = 3$ tako da drugi granični uslov nije zadovoljen.

Korak 5: Granični uslov nije zadovoljen, ali imamo dva gađanja pa prelazimo na korak 6.

Korak 6: Koristimo linearnu interpolaciju. Interpoliramo tačke: $(0, 1)$ i $(2, 3)$.

$$p(x) = \frac{x-2}{0-2} + \frac{x-0}{2-0} = x+1$$

Koristimo dobijeni polinom (pravu) da odredimo za koje x je vrednost $p(x) = 2$, tj. koji početni uslovi bi trebalo da birmo da bi zadovoljili drugi granični uslov.

$$x = ?, p(x) = 2$$
$$x+1 = 2$$
$$x = 1$$

Rešavamo sledeći sistem:

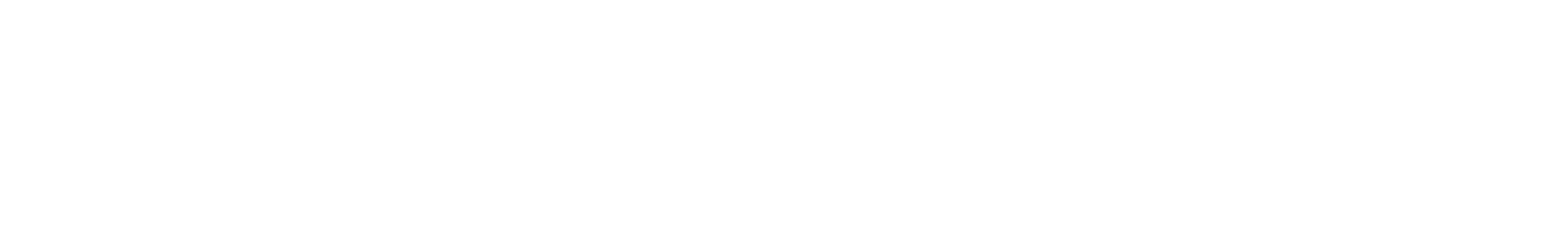
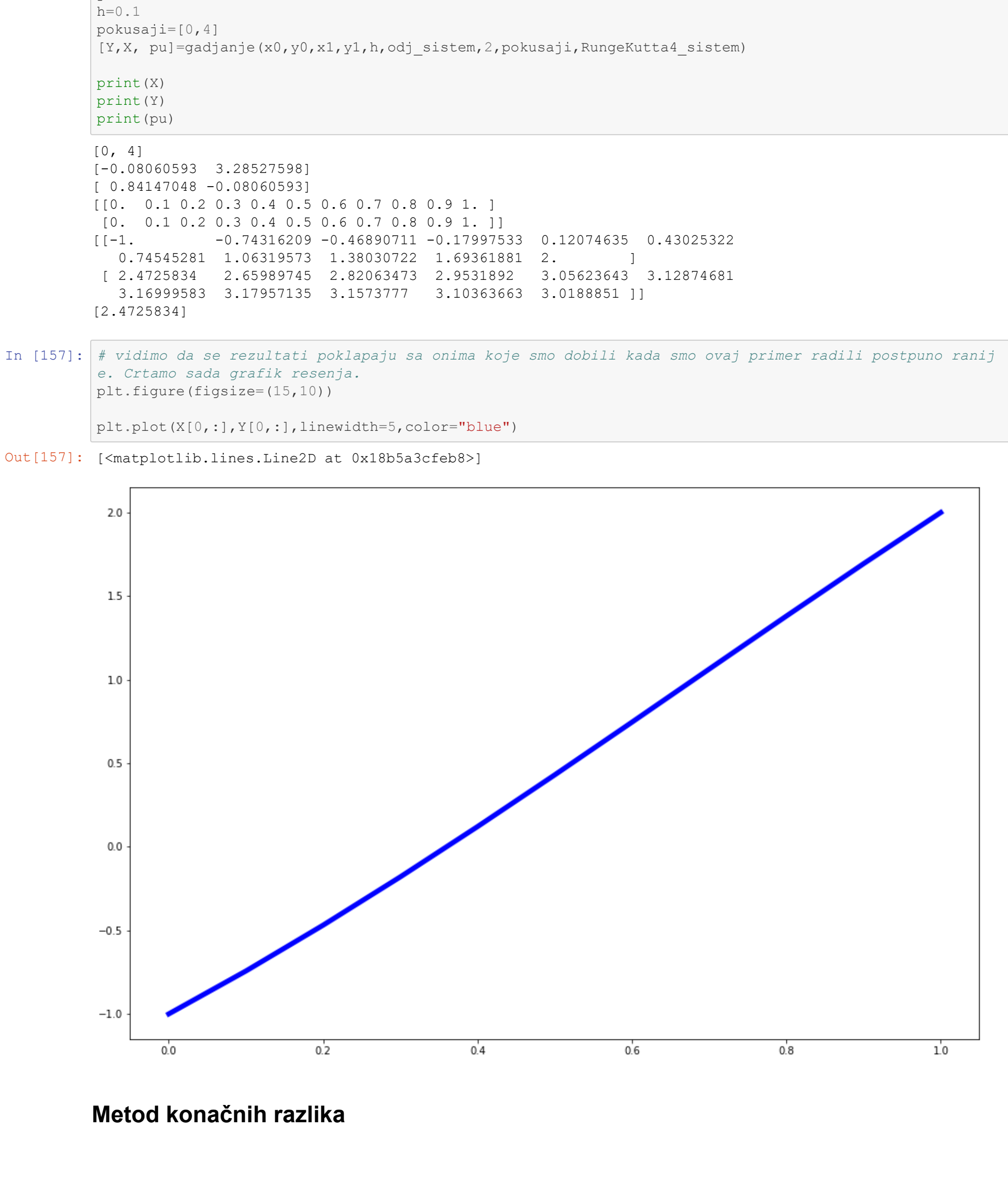
$$F(x,y) = \begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} y_2 \\ 4(y_1 - x) \end{bmatrix} \quad Y(0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Koristimo Ojlerov metod i birmo $h = 1$:

$$Y(0+1) = Y(0, Y_0) + 1F(0, Y_0)$$
$$Y_1 = Y(1) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} + 1 \begin{bmatrix} 1 \\ 4(1 - 0) \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \end{bmatrix}$$

Vidimo da smo dobili da je $y_1(1) = y(1) = 2$ tako da je drugi granični uslov zadovoljen, i PGU je rešen.

Napišaćemo sada kod za metodu gađanja.



Pošto imamo samo dve tačke $(0, 1)$ i $(1, 2)$, metoda plot iscrtava pravu kao grafik.

Pozvraćemo sada funkciju za gađanje, ali sa manjim h i pomoću RK4 metoda. Cilj nam je da dobijemo kvalitetnije rešenje.

Sada imamo više tačaka pa grafik izgleda drugačije, odnosno bolje aproksimira rešenje diferencijalne jednačije koju rešavamo.

Drugi primer:

$$\ddot{y} + y = 1, \quad y(0) = -1, y(1) = 2$$

Sistem dobijen nakon konverzije:

$$F(x,y) = \begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} y_2 \\ 1 - y_1 \end{bmatrix} \quad Y(0) = \begin{bmatrix} -1 \\ ? \end{bmatrix}$$

Metod konačnih razlika

Ovaj metod drugačije pristupa problemu PGU. Progled PGU konvertuje se u sistem algebarskih jednačina, a ne diferencijalnih kao kod metoda gađanja.

Rešenje PGU određuje se kao rešenje sistema algebarskih jednačina.

Postupak konverzije uključuje više koraka koje ćemo prvo pokazati na primeru, kao kod metode gađanja:

Rešavamo sledeći problem:

$$\ddot{y} + 2\dot{y} + y = x^2, \quad y(0) = 0.2, y(1) = 0.8$$

Da bi diferencijalnu jednačinu konvertovali u algebarsku, moramo izvesti da zamenimo nekim vrednostima.

Kao što sam naziv metoda kaže, izvide ćemo zameniti konačnim razlikama. Konačne razlike se koriste kao numerička zamena za analitički izvod funkcije u tački, i predstavljaju količnik (razlomak) koji izračunava malu promenu vrednosti y podeljenu sa malom promenom x . U nastavku navodimo formule za konačne razlike za prvi i drugi izvod, a na posebnom predavanju ćemo detaljno obraditi numeričko diferenciranje.

Recimo da imamo tačku x_i za koju hoćemo da odredimo izvod funkcije $y = f(x)$. Za izračunavanje male promene vrednosti x koristimo neku malu vrednost h :

$$\begin{aligned} x_{i+1} &= x_i + h \\ x_{i-1} &= x_i - h \end{aligned}$$

Formula za konačnu razliku za prvi izvod je:

$$\frac{y_{i+1} - y_{i-1}}{2h}$$

gde je: $y_i = f(x_i)$, $y_{i+1} = f(x_{i+1})$, $x_{i+1} - x_{i-1} = 2h$.

Formula za konačnu razliku za drugi izvod je:

$$\frac{y_{i+1} - 2y_i - y_{i-1}}{h^2}$$

Zamenićemo sada konačne razlike u naš PGU problem:

$$\ddot{y} + 2\dot{y} + y = x^2, \quad y(0) = 0.2, y(1) = 0.8$$

$$\frac{y_{i+1} - 2y_i - y_{i-1}}{h^2} + 2\frac{y_{i+1} - y_{i-1}}{2h} + y_i = x_i^2, \quad y(0) = 0.2, y(1) = 0.8$$

Sada treba da odredimo koje ćemo tačno tačke x_i da koristimo. To radimo tako što uzimamo granične uslove kao kranje tačke, i onda interval $[0, 1]$ delimo na delove veličine h .

Recimo da uzmemo da je $h = 0.25$ onda imamo sledeće tačke:

$$x_0 = 0, x_1 = 0.25, x_2 = 0.5, x_3 = 0.75, x_4 = 1$$

Ubacićemo sada $h = 0.25$ u PGU sa konačnim razlika i sredićemo malo jednačinu:

$$\frac{y_{i+1} - 2y_i - y_{i-1}}{0.25^2} + 2\frac{y_{i+1} - y_{i-1}}{0.50} + y_i = x_i^2, \quad y(0) = 0.2, y(1) = 0.8$$

$$16(y_{i+1} - 2y_i - y_{i-1}) + 8(y_{i+1} - y_{i-1}) + y_i = x_i^2, \quad y(0) = 0.2, y(1) = 0.8$$

$$24y_{i+1} - 39y_i + 16y_{i-1} = x_i^2, \quad y(0) = 0.2, y(1) = 0.8$$

Sada treba da iskoristimo formulu $24y_{i+1} - 39y_i + 16y_{i-1} = x_i^2$ da formiramo sistem jednačina. Koliko nam jednačina treba? Hajde da vidimo koliko imamo nepoznatih. Prvo ćemo videti koje su nam tačke poznate. Poznati su nam granični uslovi, odnosno poznate su nam tačke:

$$y_0 = 0.2, y_4 = 0.8$$

Sve druge vrednosti y_i su nam nepoznate:

$$y_1 = ?, y_2 = ?, y_3 = ?$$

Dakle imamo 3 nepoznate, pa nam treba 3 jednačine:

$$24y_2 - 39y_1 + 16y_0 = x_1^2$$

$$24y_3 - 39y_2 + 16y_1 = x_2^2$$

$$24y_4 - 39y_3 + 16y_2 = x_3^2$$

Sređujemo malo jednačine pre nego što formiramo sistem u matricnom obliku:

$$24y_2 - 39y_1 + 16 \cdot 0.2 = 0.25^2$$

$$24y_3 - 39y_2 + 16y_1 = 0.5^2$$

$$24 \cdot 0.8 - 39y_3 + 16y_2 = 0.75^2$$

Formiramo sistem:

$$A = \begin{bmatrix} -39 & 24 & 0 \\ 16 & -39 & 24 \\ 0 & 16 & -39 \end{bmatrix} \quad b = \begin{bmatrix} 0.25 - 16 \cdot 0.2 \\ 0.5^2 \\ 0.75^2 - 24 \cdot 0.8 \end{bmatrix}$$

Rešavamo sada sistem u kodu:

```
In [158]: from numpy import linalg as la
A=[[-39., 24., 0.], [16.0, -39., 24.], [0., 16., -39.]]
b=[0.25 - 16 * 0.2, 0.5**2, 0.75 - 24 * 0.8]
y=la.solve(A,b)

print(A)
print(b)
print(y)
print(y)
```

```
[[-39.0, 24.0, 0.0], [16.0, -39.0, 24.0], [0.0, 16.0, -39.0]]
[-2.95, 0.25, -18.450000000000003]
[0.46812238  0.6377822  0.73473116]
```

Dobili smo vrednosti za tačke y_1, y_2, y_3, y_4 , tako da uz granične uslove sada imamo:

$$y_0 = 0.2, y_1 = 0.46812, y_2 = 0.63778, y_3 = 0.73473, y_4 = 0.8$$

Pronašli smo rešenje, odnosno aproksimirali smo funkciju y u intervalu $[0, 1]$. Prikazaćemo sada rešenje grafički.



Za kraj, sistematizujemo korake metoda konačnih razlika za rešavanje PGU:

Koristimo konačne razlike da zamenimo izvode.

Biramo korak h i tako dobijamo tačke koje koristimo tj. delimo interval sa graničnim uslovima na pod-intervale. Tako dobijamo sistem algebarskih jednačina.

Rešavamo sistem algebarskih jednačina da bi dobili rešenje PGU.