

```
In [1]: import numpy as np
import matplotlib.pyplot as plt
import numpy.linalg as la

def plot_points(x,y):
    plt.figure(figsize=(15, 10))
    plt.plot(x,y,'ob', markersize = 10, markerfacecolor = 'b')
    plt.xlabel("Dani u nedelji")
    plt.ylabel("Broj zarazenih")
```

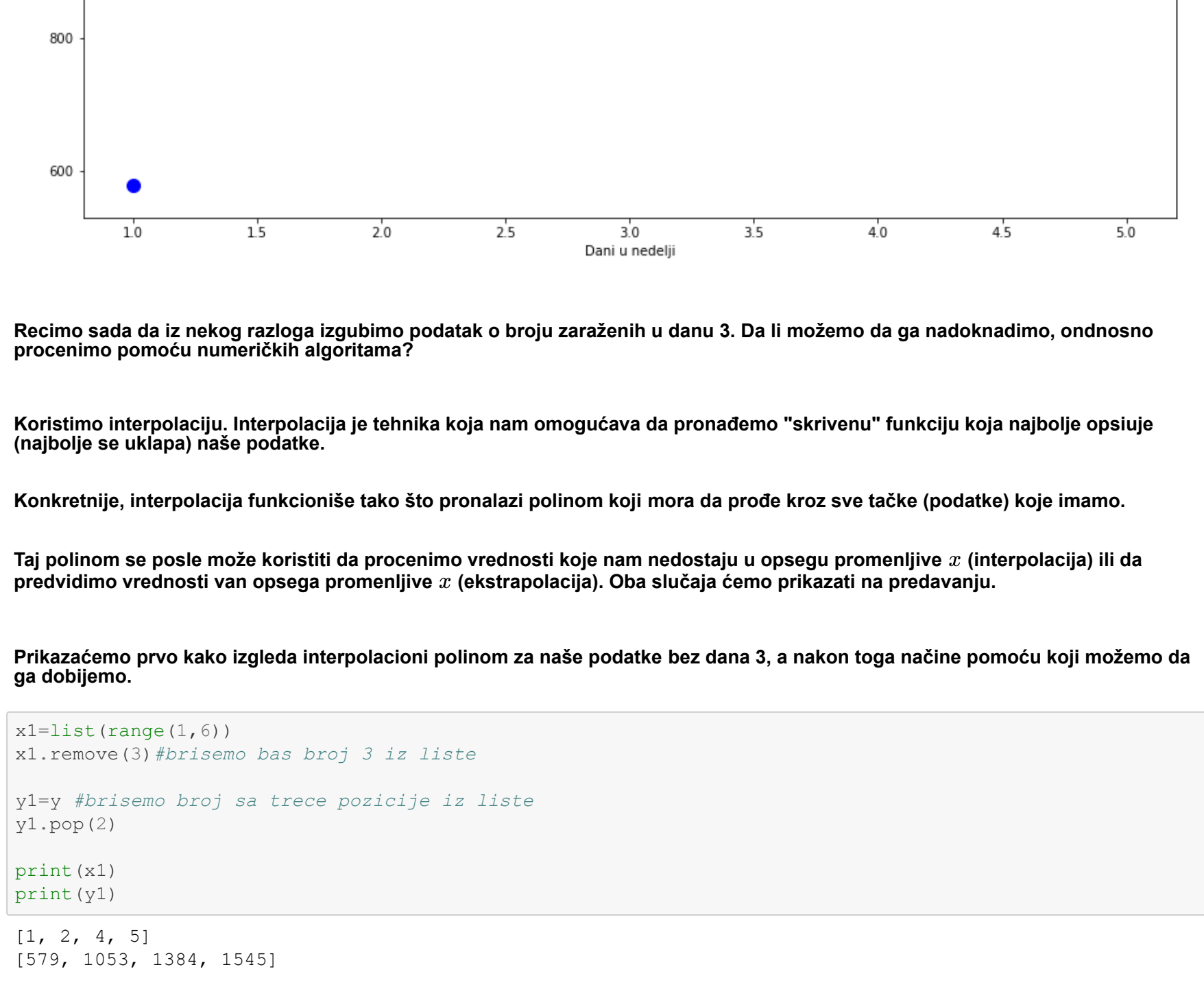
Interpolacija

Recimo da imamo podatke o 5 dana u nedelji o broju zarazenih virusom COVID-19 u Srbiji. Podaci su dati u obliku dva vektora. Vektor x sadrži dane u nedelji označene rednim brojevima, dok vektor y sadrži podatka ob broju zarazenih.

```
In [2]: x=np.arange(1,6)
y=[579,1053,1328,1384,1545]
```

Naručaćemo podatke kao tačke na 2d grafiku.

```
In [3]: plot_points(x,y)
```



Recimo sada da iz nekog razloga izgubimo podatke o broju zarazenih u danu 3. Da li možemo da ga nadoknadimo, odnosno procenimo pomoću numeričkih algoritama?

Koristimo interpolaciju. Interpolacija je tehnika koja nam omogućava da pronademo "skrivenu" funkciju koja najbolje opisuje (najbolje se uklapa) naše podatke.

Konkretnije, interpolacija funkcioniše tako što pronalazi polinom koji mora da prođe kroz sve tačke (podatke) koje imamo.

Taj polinom se posle može koristiti da procenimo vrednosti koje nam nedostaju u opsegu promenljive x (interpolacija) ili da predvidimo vrednosti van opsega promenljive x (ekstrapolacija). Oba slučaja ćemo prikazati na predavanju.

Prikazaćemo prvo kako izgleda interpolacioni polinom za naše podatke bez dana 3, a nakon toga načine pomoću koji možemo da ga dobijemo.

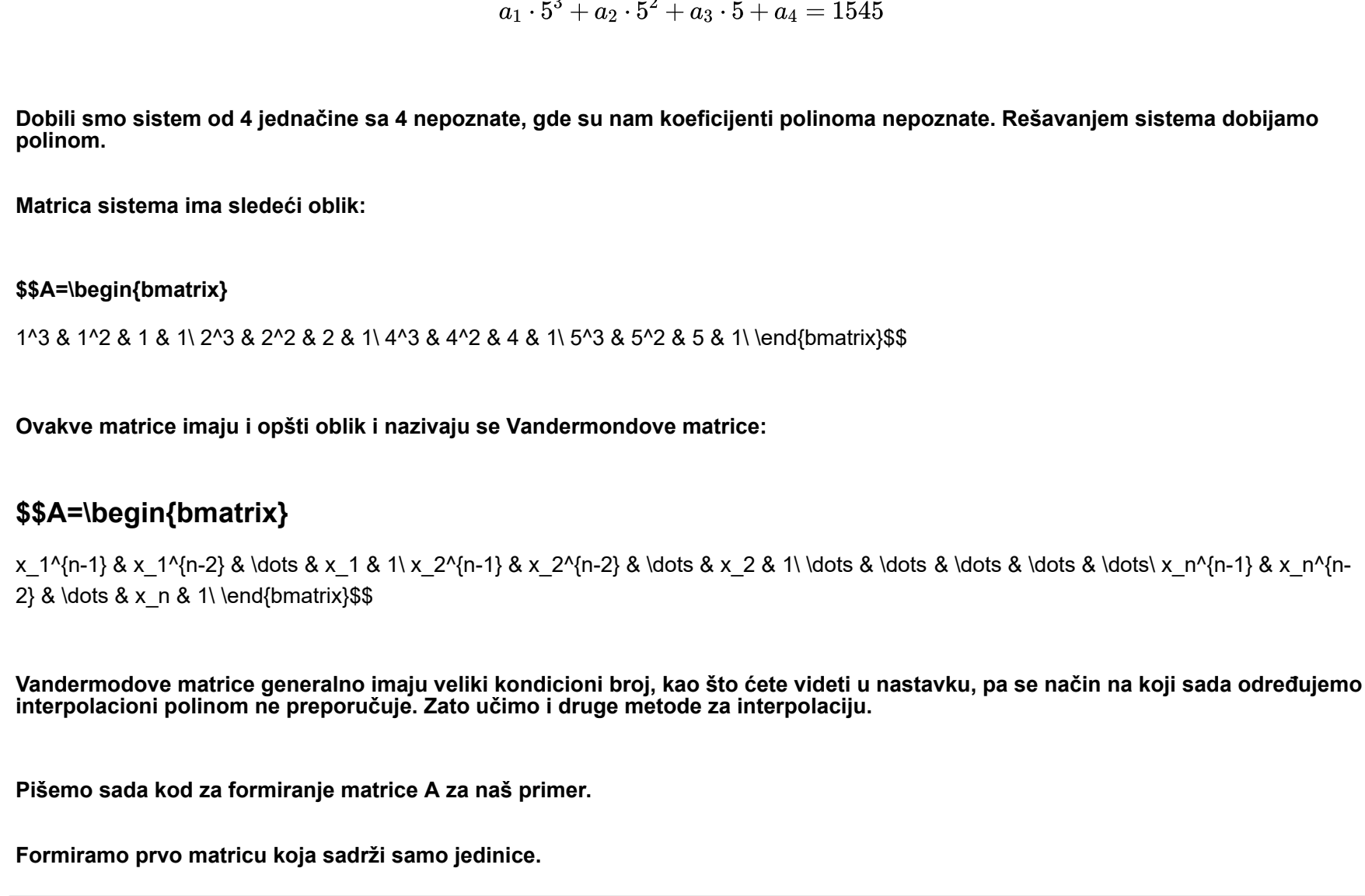
```
In [4]: x1=list(range(1,6))
x1.remove(3)#brišemo bas broj 3 iz liste
y1=y #brišemo broj sa treće pozicije iz liste
y1.pop(2)

print(x1)
print(y1)

[1, 2, 4, 5]
[579, 1053, 1384, 1545]
```

```
In [5]: p=np.polyfit(x1, y1, 3)
xp=np.linspace(1,np.max(x1),100)
xp=np.linspace(1,np.max(x1),100)
plt.plot(xp,np.polyval(p,xp))

print(p)
#ovo je polinom koji odgovara vekturu koeficijenata p:
#25.333333x^3 - 280.17*x^2 + 1137.2*x^1 - 303.33
[ 25.33333333 -280.16666667 1137.16666667 -303.33333333]
```



Interpolacija - prvi način

Interpolacioni polinom trećeg stepena formiramo rešavanjem sistema linearnih jednačina.

Šta mislite zašto formiramo polinom 3.eg trećeg stepena? (Mala pomoć: imamo 4 tačke. Ako imamo dve tačke kog stepena je polinom koji prolazi kroz njih?)

Opšti oblik polinoma trećeg stepena:

$$p(x) = a_1x^3 + a_2x^2 + a_3x^2 + a_4$$

Možda ste primetili da smo koeficijente poredali tako da je onaj sa najmanjim indeksom (a_0) uz najveći stepen (x^3) tako redom. To je zato što MATLAB i Octave reprezentuju polinome kao vektore koeficijenata, tako da prvom indeksu u vekturu odgovara koeficijent uz najveći stepen, a drugom koeficijent uz stepen manje itd.

Sistem linearnih jednačina pomoću koga dobijemo polinom formiramo tako što koristimo uslove koje polinom mora da zadovolji.

Konkretno, polinom mora da prođe kroz sve tačke koje imamo. To znači da za svako x iz podataka $p(x)$ mora da bude jednako odgovarajućem y .

Za naš primer uslovi su sledećeg oblika:

$$\begin{aligned} x &= (1, 2, 4, 5) \\ y &= (579, 1053, 1384, 1545) \\ p(1) &= 579, p(2) = 1053, p(4) = 1384, p(5) = 1545 \\ a_1 \cdot 1^3 + a_2 \cdot 1^2 + a_3 \cdot 1 + a_4 &= 579 \\ a_1 \cdot 2^3 + a_2 \cdot 2^2 + a_3 \cdot 2 + a_4 &= 1053 \\ a_1 \cdot 4^3 + a_2 \cdot 4^2 + a_3 \cdot 4 + a_4 &= 1384 \\ a_1 \cdot 5^3 + a_2 \cdot 5^2 + a_3 \cdot 5 + a_4 &= 1545 \end{aligned}$$

Dobili smo sistem od 4 jednačina sa 4 nepoznate, gde su nam koeficijenti polinoma nepoznate. Rešavanjem sistema dobijamo sistem.

Matrica sistema ima sledeći oblik:

```
$$A=$begin{bmatrix}1^3&1^2&1&1&2^3&2^2&2&2&4^3&4^2&4&4&5^3&5^2&5&5\end{bmatrix}$$
```

Ovakve matrice imaju i opšti oblik i nazivaju se Vandermondove matrice:

\$\$A=\$begin{bmatrix}

x_1^{n-1} & x_1^{n-2} & x_1^{n-3} & x_1^{n-4} & x_1^{n-5} & x_1^{n-6} & x_1^{n-7} & x_1^{n-8} & x_1^{n-9} & x_1^{n-10} & x_1^{n-11} & x_1^{n-12} & x_1^{n-13} & x_1^{n-14} & x_1^{n-15} & x_1^{n-16} & x_1^{n-17} & x_1^{n-18} & x_1^{n-19} & x_1^{n-20} & x_1^{n-21} & x_1^{n-22} & x_1^{n-23} & x_1^{n-24} & x_1^{n-25} & x_1^{n-26} & x_1^{n-27} & x_1^{n-28} & x_1^{n-29} & x_1^{n-30} & x_1^{n-31} & x_1^{n-32} & x_1^{n-33} & x_1^{n-34} & x_1^{n-35} & x_1^{n-36} & x_1^{n-37} & x_1^{n-38} & x_1^{n-39} & x_1^{n-40} & x_1^{n-41} & x_1^{n-42} & x_1^{n-43} & x_1^{n-44} & x_1^{n-45} & x_1^{n-46} & x_1^{n-47} & x_1^{n-48} & x_1^{n-49} & x_1^{n-50} & x_1^{n-51} & x_1^{n-52} & x_1^{n-53} & x_1^{n-54} & x_1^{n-55} & x_1^{n-56} & x_1^{n-57} & x_1^{n-58} & x_1^{n-59} & x_1^{n-60} & x_1^{n-61} & x_1^{n-62} & x_1^{n-63} & x_1^{n-64} & x_1^{n-65} & x_1^{n-66} & x_1^{n-67} & x_1^{n-68} & x_1^{n-69} & x_1^{n-70} & x_1^{n-71} & x_1^{n-72} & x_1^{n-73} & x_1^{n-74} & x_1^{n-75} & x_1^{n-76} & x_1^{n-77} & x_1^{n-78} & x_1^{n-79} & x_1^{n-80} & x_1^{n-81} & x_1^{n-82} & x_1^{n-83} & x_1^{n-84} & x_1^{n-85} & x_1^{n-86} & x_1^{n-87} & x_1^{n-88} & x_1^{n-89} & x_1^{n-90} & x_1^{n-91} & x_1^{n-92} & x_1^{n-93} & x_1^{n-94} & x_1^{n-95} & x_1^{n-96} & x_1^{n-97} & x_1^{n-98} & x_1^{n-99} & x_1^{n-100} & x_1^{n-101} & x_1^{n-102} & x_1^{n-103} & x_1^{n-104} & x_1^{n-105} & x_1^{n-106} & x_1^{n-107} & x_1^{n-108} & x_1^{n-109} & x_1^{n-110} & x_1^{n-111} & x_1^{n-112} & x_1^{n-113} & x_1^{n-114} & x_1^{n-115} & x_1^{n-116} & x_1^{n-117} & x_1^{n-118} & x_1^{n-119} & x_1^{n-120} & x_1^{n-121} & x_1^{n-122} & x_1^{n-123} & x_1^{n-124} & x_1^{n-125} & x_1^{n-126} & x_1^{n-127} & x_1^{n-128} & x_1^{n-129} & x_1^{n-130} & x_1^{n-131} & x_1^{n-132} & x_1^{n-133} & x_1^{n-134} & x_1^{n-135} & x_1^{n-136} & x_1^{n-137} & x_1^{n-138} & x_1^{n-139} & x_1^{n-140} & x_1^{n-141} & x_1^{n-142} & x_1^{n-143} & x_1^{n-144} & x_1^{n-145} & x_1^{n-146} & x_1^{n-147} & x_1^{n-148} & x_1^{n-149} & x_1^{n-150} & x_1^{n-151} & x_1^{n-152} & x_1^{n-153} & x_1^{n-154} & x_1^{n-155} & x_1^{n-156} & x_1^{n-157} & x_1^{n-158} & x_1^{n-159} & x_1^{n-160} & x_1^{n-161} & x_1^{n-162} & x_1^{n-163} & x_1^{n-164} & x_1^{n-165} & x_1^{n-166} & x_1^{n-167} & x_1^{n-168} & x_1^{n-169} & x_1^{n-170} & x_1^{n-171} & x_1^{n-172} & x_1^{n-173} & x_1^{n-174} & x_1^{n-175} & x_1^{n-176} & x_1^{n-177} & x_1^{n-178} & x_1^{n-179} & x_1^{n-180} & x_1^{n-181} & x_1^{n-182} & x_1^{n-183} & x_1^{n-184} & x_1^{n-185} & x_1^{n-186} & x_1^{n-187} & x_1^{n-188} & x_1^{n-189} & x_1^{n-190} & x_1^{n-191} & x_1^{n-192} & x_1^{n-193} & x_1^{n-194} & x_1^{n-195} & x_1^{n-196} & x_1^{n-197} & x_1^{n-198} & x_1^{n-199} & x_1^{n-200} & x_1^{n-201} & x_1^{n-202} & x_1^{n-203} & x_1^{n-204} & x_1^{n-205} & x_1^{n-206} & x_1^{n-207} & x_1^{n-208} & x_1^{n-209} & x_1^{n-210} & x_1^{n-211} & x_1^{n-212} & x_1^{n-213} & x_1^{n-214} & x_1^{n-215} & x_1^{n-216} & x_1^{n-217} & x_1^{n-218} & x_1^{n-219} & x_1^{n-220} & x_1^{n-221} & x_1^{n-222} & x_1^{n-223} & x_1^{n-224} & x_1^{n-225} & x_1^{n-226} & x_1^{n-227} & x_1^{n-228} & x_1^{n-229} & x_1^{n-230} & x_1^{n-231} & x_1^{n-232} & x_1^{n-233} & x_1^{n-234} & x_1^{n-235} & x_1^{n-236} & x_1^{n-237} & x_1^{n-238} & x_1^{n-239} & x_1^{n-240} & x_1^{n-241} & x_1^{n-242} & x_1^{n-243} & x_1^{n-244} & x_1^{n-245} & x_1^{n-246} & x_1^{n-247} & x_1^{n-248} & x_1^{n-249} & x_1^{n-250} & x_1^{n-251} & x_1^{n-252} & x_1^{n-253} & x_1^{n-254} & x_1^{n-255} & x_1^{n-256} & x_1^{n-257} & x_1^{n-258} & x_1^{n-259} & x_1^{n-260} & x_1^{n-261} & x_1^{n-262} & x_1^{n-263} & x_1^{n-264} & x_1^{n-265} & x_1^{n-266} & x_1^{n-267} & x_1^{n-268} & x_1^{n-269} & x_1^{n-270} & x_1^{n-271} & x_1^{n-272} & x_1^{n-273} & x_1^{n-274} & x_1^{n-275} & x_1^{n-276} & x_1^{n-277} & x_1^{n-278} & x_1^{n-279} & x_1^{n-280} & x_1^{n-281} & x_1^{n-282} & x_1^{n-283} & x_1^{n-284} & x_1^{n-285} & x_1^{n-286} & x_1^{n-287} & x_1^{n-288} & x_1^{n-289} & x_1^{n-290} & x_1^{n-291} & x_1^{n-292} & x_1^{n-293} & x_1^{n-294} & x_1^{n-295} & x_1^{n-296} & x_1^{n-297} & x_1^{n-298} & x_1^{n-299} & x_1^{n-300} & x_1^{n-301} & x_1^{n-302} & x_1^{n-303} & x_1^{n-304} & x_1^{n-305} & x_1^{n-306} & x_1^{n-307} & x_1^{n-308} & x_1^{n-309} & x_1^{n-310} & x_1^{n-311} & x_1^{n-312} & x_1^{n-313} & x_1^{n-314} & x_1^{n-315} & x_1^{n-316} & x_1^{n-317} & x_1^{n-318} & x_1^{n-319} & x_1^{n-320} & x_1^{n-321} & x_1^{n-322} & x_1^{n-323} & x_1^{n-324} & x_1^{n-325} & x_1^{n-326} & x_1^{n-327} & x_1^{n-328} & x_1^{n-329} & x_1^{n-330} & x_1^{n-331} & x_1^{n-332} & x_1^{n-333} & x_1^{n-334} & x_1^{n-335} & x_1^{n-336} & x_1^{n-337} & x_1^{n-338} & x_1^{n-339} & x_1^{n-340} & x_1^{n-341} & x_1^{n-342} & x_1^{n-343} & x_1^{n-344} & x_1^{n-345} & x_1^{n-346} & x_1^{n-347} & x_1^{n-348} & x_1^{n-349} & x_1^{n-350} & x_1^{n-351} & x_1^{n-352} & x_1^{n-353} & x_1^{n-354} & x_1^{n-355} & x_1^{n-356} & x_1^{n-357} & x_1^{n-358} & x_1^{n-359} & x_1^{n-360} & x_1^{n-361} & x_1^{n-362} & x_1^{n-363} & x_1^{n-364} & x_1^{n-365} & x_1^{n-366} & x_1^{n-367} & x_1^{n-368} & x_1^{n-369} & x_1^{n-370} & x_1^{n-371} & x_1^{n-372} & x_1^{n-373} & x_1^{n-374} & x_1^{n-375} & x_1^{n-376} & x_1^{n-377} & x_1^{n-378} & x_1^{n-379} & x_1^{n-380} & x_1^{n-381} & x_1^{n-382} & x_1^{n-383} & x_1^{n-384} & x_1^{n-385} & x_1^{n-386} & x_1^{n-387} & x_1^{n-388} & x_1^{n-389} & x_1^{n-390} & x_1^{n-391} & x_1^{n-392} & x_1^{n-393} & x_1^{n-394} & x_1^{n-395} & x_1^{n-396} & x_1^{n-397} & x_1^{n-398} & x_1^{n-399} & x_1^{n-400} & x_1^{n-401} & x_1^{n-402} & x_1^{n-403} & x_1^{n-404} & x_1^{n-405} & x_1^{n-406} & x_1^{n-407} & x_1^{n-408} & x_1^{n-409} & x_1^{n-410} & x_1^{n-411} & x_1^{n-412} & x_1^{n-413} & x_1^{n-414} & x_1^{n-415} & x_1^{n-416} & x_1^{n-417} & x_1^{n-418} & x_1^{n-419} & x_1^{n-420} & x_1^{n-421} & x_1^{n-422} & x_1^{n-423} & x_1^{n-424} & x_1^{n-425} & x_1^{n-426} & x_1^{n-427} & x_1^{n-428} & x_1^{n-429} & x_1^{n-430} & x_1^{n-431} & x_1^{n-432} & x_1^{n-433} & x_1^{n-434} & x_1^{n-435} & x_1^{n-436} & x_1^{n-437} & x_1^{n-438} & x_1^{n-439} & x_1^{n-440} & x_1^{n-441} & x_1^{n-442} & x_1^{n-443} & x_1^{n-444} & x_1^{n-445} & x_1^{n-446} & x_1^{n-447} & x_1^{n-448} & x_1^{n-449} & x_1^{n-450} & x_1^{n-451} & x_1^{n-452} & x_1^{n-453} & x_1^{n-454} & x_1^{n-455} & x_1^{n-456} & x_1^{n-457} & x_1^{n-458} & x_1^{n-459} & x_1^{n-460} & x_1^{n-461} & x_1^{n-462} & x_1^{n-463} & x_1^{n-464} & x_1^{n-465} & x_1^{n-466} & x_1^{n-467} & x_1^{n-468} & x_1^{n-469} & x_1^{n-470} & x_1^{n-471} & x_1^{n-472} & x_1^{n-473} & x_1^{n-474} & x_1^{n-475} & x_1^{n-476} & x_1^{n-477} & x_1^{n-478} & x_1^{n-479} & x_1^{n-480} & x_1^{n-481} & x_1^{n-482} & x_1^{n-483} & x_1^{n-484} & x_1^{n-485} & x_1^{n-486} & x_1^{n-487} & x_1^{n-488} & x_1^{n-489} & x_1^{n-490} & x_1^{n-491} & x_1^{n-492} & x_1^{n-493} & x_1^{n-494} & x_1^{n-495} & x_1^{n-496} & x_1^{n-497} & x_1^{n-498} & x_1^{n-499} & x_1^{n-500} & x_1^{n-501} & x_1^{n-502} & x_1^{n-503} & x_1^{n-504} & x_1^{n-505} & x_1^{n-506} & x_1^{n-507} & x_1^{n-508} & x_1^{n-509} & x_1^{n-510} & x_1^{n-511} & x_1^{n-512} & x_1^{n-513} & x_1^{n-514} & x_1^{n-515} & x_1^{n-516} & x_1^{n-517} & x_1^{n-518} & x_1^{n-519} & x_1^{n-520} & x_1^{n-521} & x_1^{n-522} & x_1^{n-523} & x_1^{n-524} & x_1^{n-525} & x_1^{n-526} & x_1^{n-527} & x_1^{n-528} & x_1^{n-529} & x_1^{n-530} & x_1^{n-531} & x_1^{n-532} & x_1^{n-533} & x_1^{n-534} & x_1^{n-535} & x_1^{n-536} & x_1^{n-537} & x_1^{n-538} & x_1^{n-539} & x_1^{n-540} & x_1^{n-541} & x_1^{n-542} & x_1^{n-543} & x_1^{n-544} & x_1^{n-545} & x_1^{n-546} & x_1^{n-547} & x_1^{n-548} & x_1^{n-549} & x_1^{n-550} & x_1^{n-551} & x_1^{n-552} & x_1^{n-553} & x_1^{n-554} & x_1^{n-555} & x_1^{n-556} & x_1^{n-557} & x_1^{n-558} & x_1^{n-559} & x_1^{n-560} & x_1^{n-561} & x_1^{n-562} & x_1^{n-563} & x_1^{n-564} & x_1^{n-565} & x_1^{n-566} & x_1^{n-567} & x_1^{n-568} & x_1^{n-569} & x_1^{n-570} & x_1^{n-571} & x_1^{n-572} & x_1^{n-573} & x_1^{n-574} & x_1^{n-575} & x_1^{n-576} & x_1^{n-577} & x_1^{n-578} & x_1^{n-579} &


```
[46]: plot_points(x,y)

xp=np.linspace(1,np.max(x),100)
plt.plot(xp,np.polyval(p,xp))

Out[46]: <matplotlib.lines.Line2D at 0x22841e3ca90>
```



Koristimo sada tačke (dane) 1, 2, 4 i 5 za interpolaciju, a nakon toga procenjujemo vrednost za dan 3.

```
In [47]: x=[1,2,4,5]
y=[579,1053,1384,1545]

p=interp(x,y)

print(p)
print(p_pomocu_SLAJ)
print(np.polyval(p,3))

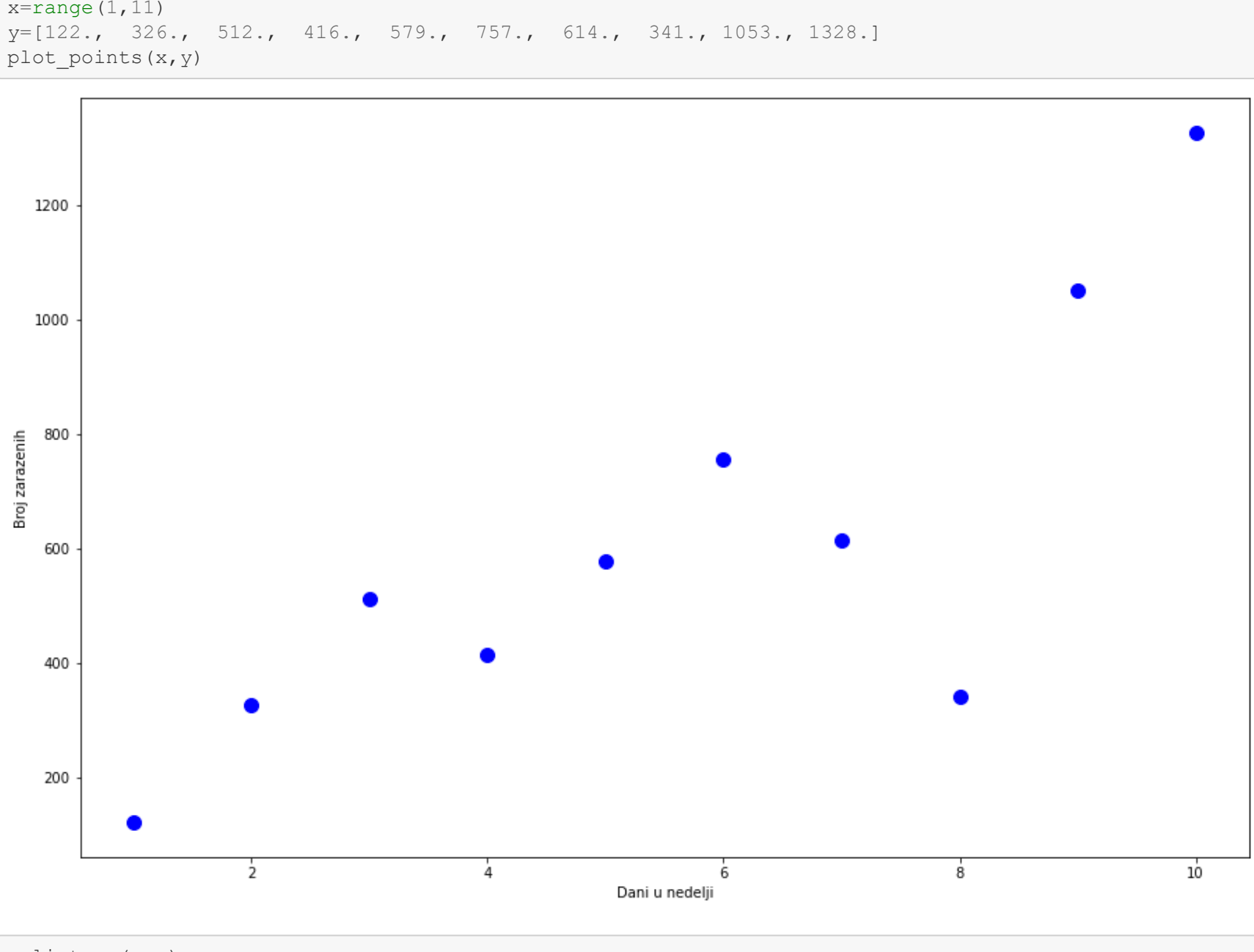
[ 25.33333333 -280.16666667 1137.16666667 -303.33333333]
[ 25.33333333 -280.16666667 1137.16666667 -303.33333333]
1270.66666666672
```

Vidimo da je Lagranžova interpolacija proizvela isti polinom kao metod pomoću sistema linearnih jednačina. To je zato što je interpolacioni polinom jedinstven. Iako postoji formalan dokaz, dovoljno je da razmislimo o tome koliko pravih može da se povuče kroz dve date tačke.

```
In [48]: plot_points(x,y)
xp=np.linspace(1,np.max(x),100)
plt.plot(xp,np.polyval(p,xp))

plt.plot(3,np.polyval(p,3),'ob',markersize=15,markerfacecolor='r')
plt.plot(3,y[2],'ob',markersize=15,markerfacecolor='g')
```

```
Out[48]: <matplotlib.lines.Line2D at 0x22841eb22e8>
```



```
In [49]: predikcija_p_pomocu_SLAJ = np.polyval(p_pomocu_SLAJ,3)
print(predikcija_p_pomocu_SLAJ)

1270.666666666667
```

```
In [50]: predikcija_p_linterp = np.polyval(p,3)
print(predikcija_p_linterp)

1270.66666666672
```

```
In [51]: tacna_vrednost = y[2]
print(tacna_vrednost)

1384
```

Interpolacija splajnom

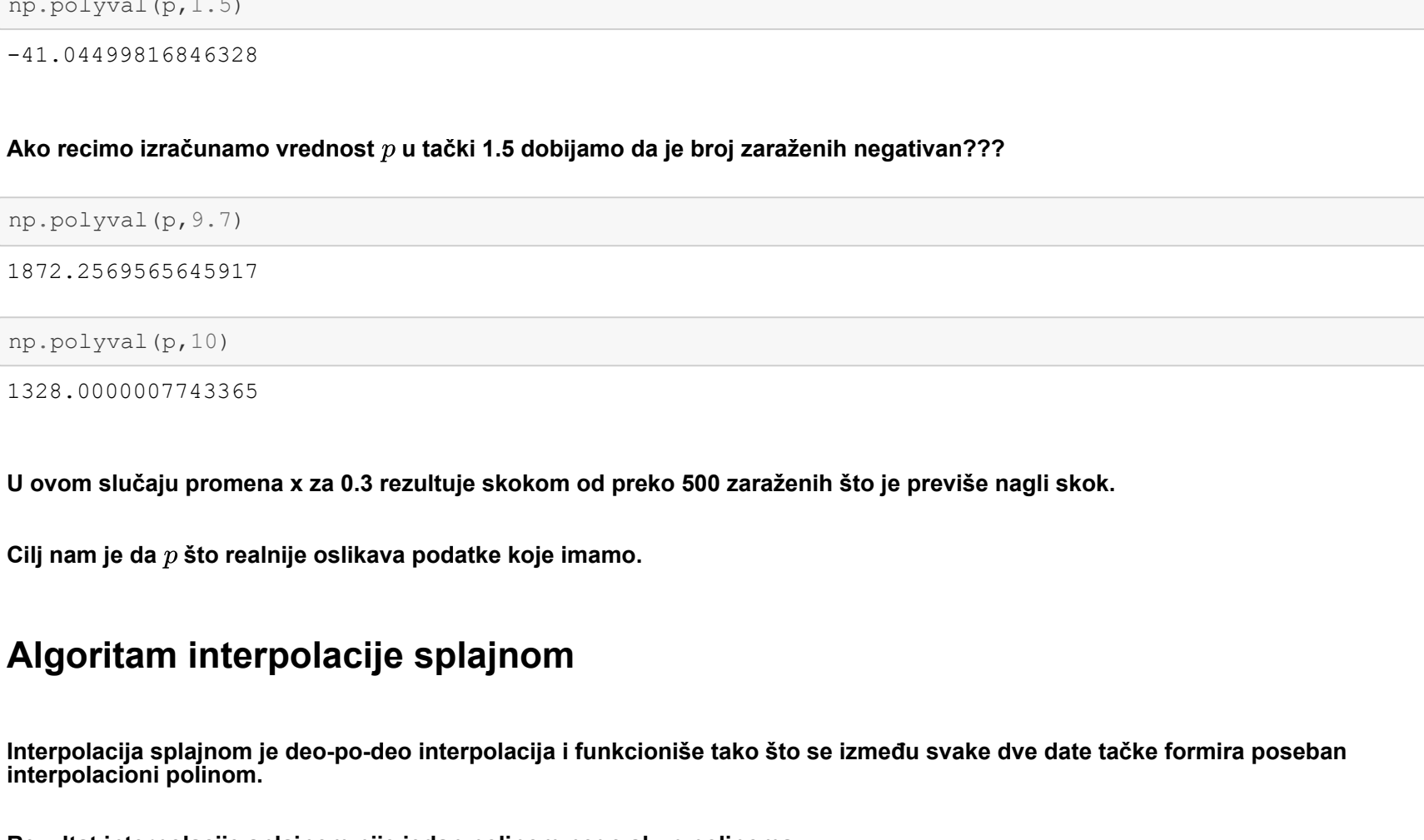
Interpolacione metode koje smo do sada radili nisu pogodne za upotrebu kada je broj interpolacionih tačaka veći.

Kada je broj interpolacionih tačaka veći interpolacioni polinom postaje "nestabilan", tj. ima velike oscilacije (nagle skokove i padove) što se može prepoznati po velikim apsolutnim vrednostima koeficijenata.

Intuitivno, nestabilnost je posledica interpolacionih uslova, odnosno zahteva da jedan polinom prođe baš kroz sve tačke.

Nestabilost interpolacionog polinoma ilustrovaćemo na primeru od 10 dana i broja zaraženih u Srbiji.

```
In [52]: x=range(1,11)
y=[122., 326., 512., 416., 579., 757., 614., 341., 1053., 1328.]
plot_points(x,y)
```



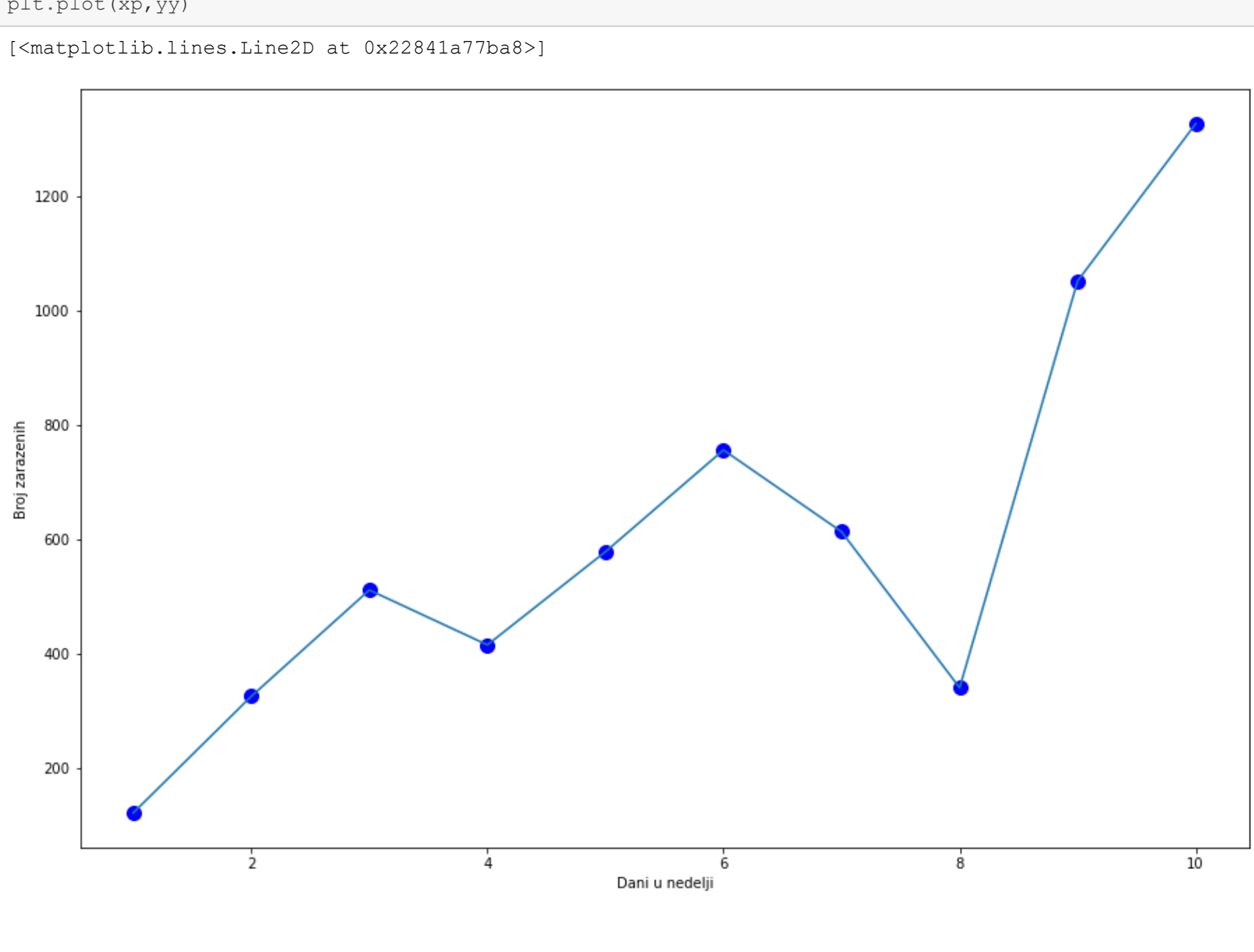
```
In [53]: p=interp(x,y)
print(p)

[-1.56994049e-02  7.61259921e-01 -1.60984623e+01  1.94538194e+02
 -1.47228459e+03  7.16194757e+03 -2.20494251e+04  4.05832530e+04
 -3.93856762e+04  1.51050000e+04]
```

Vidimo da p ima dosta velike vrednosti koeficijenata.

```
In [54]: plot_points(x,y)
xp=np.linspace(1,np.max(x),100)
plt.plot(xp,np.polyval(p,xp))

Out[54]: <matplotlib.lines.Line2D at 0x228420125c0>
```



Sa grafika se vidi da p ima nagle uspone i padove.

```
In [55]: np.polyval(p,1.5)

Out[55]: -01.04499816846328
```

Ako recimo izračunamo vrednost p u tački 1.5 dobijamo da je broj zaraženih negativan???

```
In [56]: np.polyval(p,9.7)

Out[56]: 1872.2569566463917
```

```
In [57]: np.polyval(p,10)

Out[57]: 1328.000000743365
```

U ovom slučaju promena x za 0.3 rezultuje skokom od preko 500 zaraženih što je previše nagli skok.

Cilj nam je da p što realnije oslikava podatke koje imamo.

Algoritam interpolacije splajnom

Interpolacija splajnom je deo-po-deo interpolacija i funkcioniše tako što se između svake dve date tačke formira poseban interpolacioni polinom.

Rezultat interpolacije splajnom nije jedan polinom nego skup polinoma.

Linearni splajn

Kod ovog splajna između svake dve tačke formiramo pravu.

Za datih n tačaka rezultat je $n-1$ prava, zato što toliko imamo pod-intervalu.

Nacrtaćemo sada linearni splajn za naš primer sa 10 tačaka.

```
In [58]: from scipy.interpolate import interp1d
x=range(1,11)
y=[122., 326., 512., 416., 579., 757., 614., 341., 1053., 1328.]
plot_points(x,y)
xp=np.linspace(1,np.max(x),100)
p = interp1d(x, y, kind='linear')
y = p(xp)#rezultat interpolacije u ovom slucaju je funkcija kojoj mogu da se proslede vrednosti za x
# i želimo da dobijemo y
plt.plot(xp,yy)
```

```
Out[58]: <matplotlib.lines.Line2D at 0x22841cefd8b>
```



Splajnovne koristimo tako što za odgovarajuću tačku koju hoćemo da ubacimo u splajn pronađemo tačku između kojih se nalazi i onda ubacimo tačku u pravu između njih. U Octave za nas to automatski obavlja funkcija ppval.

Očigledno je da, iako je veoma jednostavn, linearni splajn je veoma gruba interpolacija jer je retko slučaj da je prelaz iz jedne u drugu tačku baš linearni.

Pored toga linearni splajn ima nagle promene nagiba (izvoda) u datim tačkama što znači da se njegov oblik naglo menja na prelazima iz jedne tačke u drugu. Ako u našim podacima postoji trend, vrlo je verovatno da ne sadrži tako nagle prelaze, pa bi bilo dobro da pronađemo koji način za interpolaciju.

Kvadratni splajn

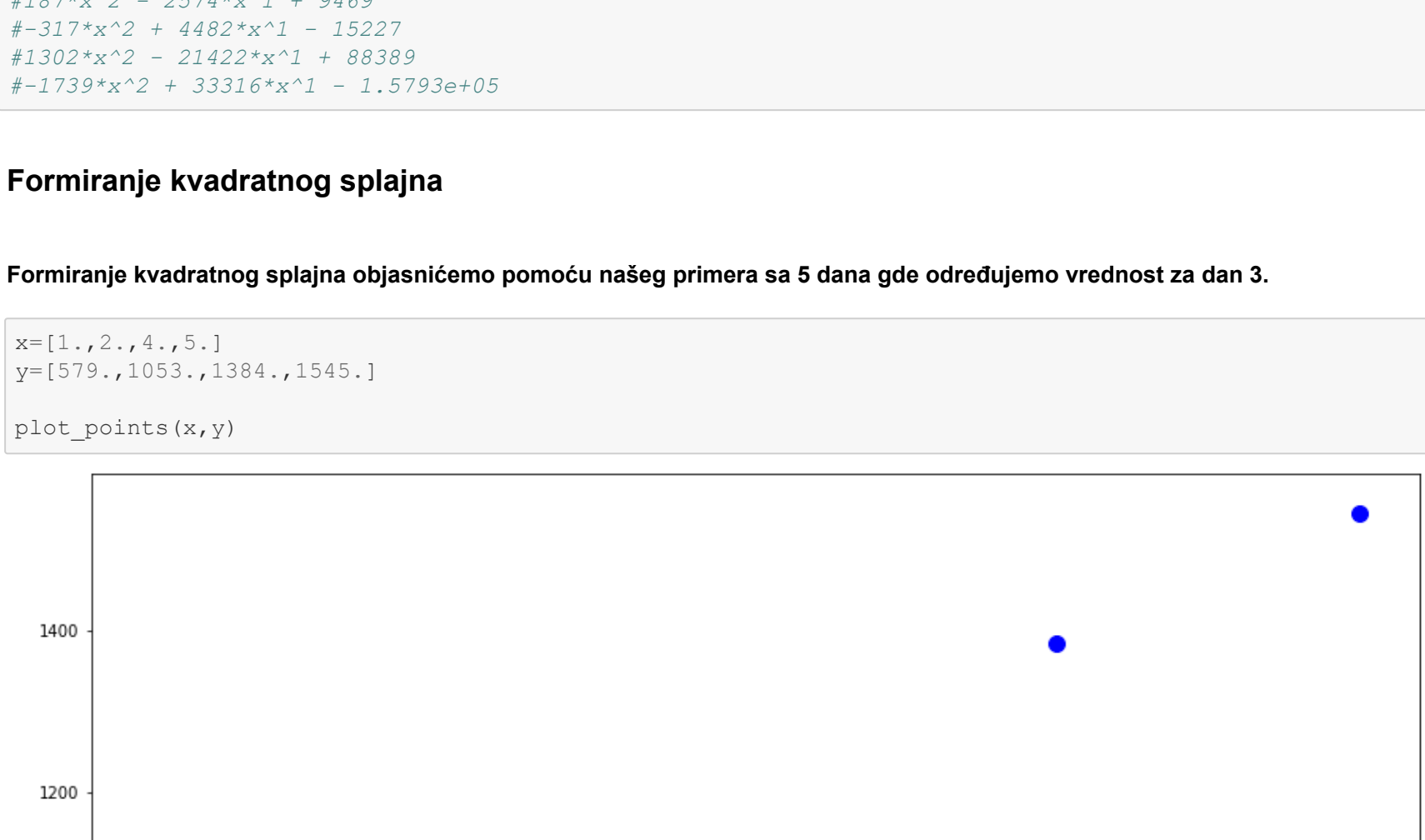
Kao što samo ime kaže u ovom slučaju između svake dve tačke formiramo polinom drugog stepena.

Za razliku od linearnog splajna, prelazi između tačaka neće biti nagli jer kao uslov za formiranje kvadratni splajn zahteva da izvodi u unutrašnjim tačkama (gde se dve splajna spajaju) budu jednaki. Dakle, nagib kojim se jedan splajn završava isti je kao i nagib kojim sledeći splajn počinje. Sada imamo glatke prelaze, a ne špičeve.

Crtamo kvadratni splajn za naš primer sa 10 tačaka.

```
In [59]: x=range(1,11)
y=[122., 326., 512., 416., 579., 757., 614., 341., 1053., 1328.]
plot_points(x,y)
xp=np.linspace(1,np.max(x),100)
p = interp1d(x, y, kind='quadratic')
yy = p(xp)
plt.plot(xp,yy)
```

```
Out[59]: <matplotlib.lines.Line2D at 0x22841b8a7b8>
```

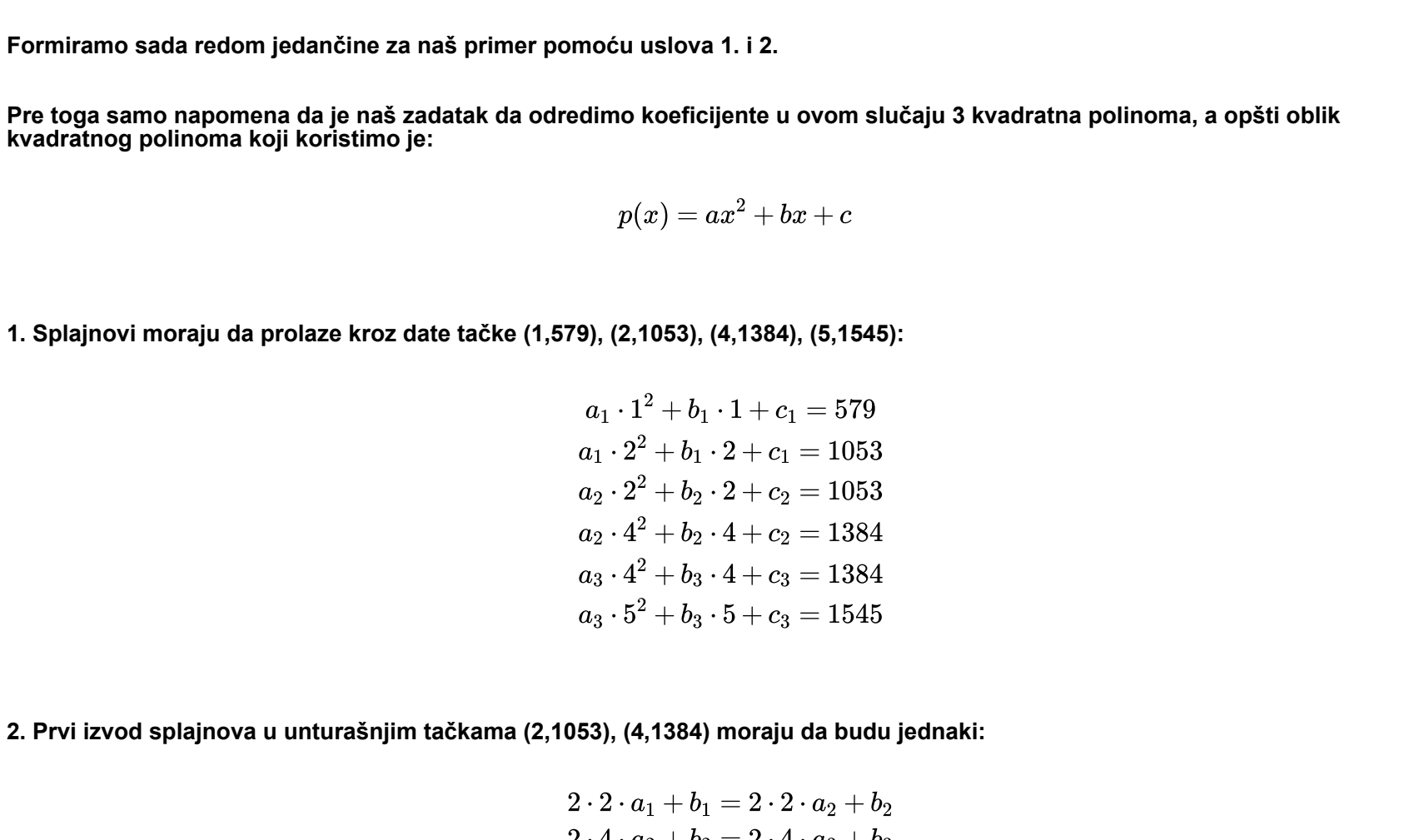


Kao što vidimo kod kvadratnog splajna prelazi u interpolacionim tačkama su blaži. Nemamo špičeve, tj. prekidne prvog izvoda kao kod linearnog splajna.

Uporedićemo sada kvadratni splajn i Lagranžov interpolacioni polinom.

```
In [103]: x=range(1,11)
y=[122., 326., 512., 416., 579., 757., 614., 341., 1053., 1328.]
plot_points(x,y)
xp=np.linspace(1,np.max(x),100)
my_kv_splajn = quadratic_spline(x,y)
yy = eval_spline(my_kv_splajn,x,[1.5])
#224.
print(yy)
```

```
Out[103]: <matplotlib.lines.Line2D at 0x22841cefd8b>
```



```
In [104]: print(np.polyval(linterp_pol,1.5))
print(eval_spline(my_kv_splajn,x,[1.5]))

-01.04499816846328
[224.]
```

```
In [105]: print([y[8], y[9]])
print(np.polyval(linterp_pol,9.5))
print(eval_spline(my_kv_splajn,x,[9.5]))

[1053.0, 1328.0]
1773.264816865354
[1625.25]
```

Očigledno je da kvadratni splajn nema oscilacije koje ima Lagranžov interpolacioni polinom. Nema negativnih vrednosti u tački 1.5. Takođe prolaz između tačka 9 i 10 je mnogo blaži, tj. realniji.

Pokazaćemo sada kako izgleda svaki od polinoma kvadratnog splajna.

```
In [106]: my_kv_splajn = quadratic_spline(x,y)
print(my_kv_splajn)

[ 0.00000e+00  2.04000e+02 -8.20000e+01 -1.80000e+01  2.76000e+02
 -4.54000e+02  1.02240e+04 -1.75200e+03 -2.36800e+03  5.23000e+02
 -5.45400e+01  1.02240e+04 -5.08000e+02  5.76600e+03 -1.5510e+04
 1.87000e+02 -2.57400e+03  9.46900e+02 -3.17000e+02  4.48200e+03
 -1.52270e+04  1.30200e+03 -2.14220e+04  6.83890e+04 -1.73900e+03
 3.33160e+04 -1.57932e+05]
```

```
In [ ]: #0*x^2 + 204*x^1 - 82
#-82*x^2 + 276*x^1 - 154
#-264*x^2 + 1752*x^1 - 2368
#523*x^2 - 4544*x^1 + 10224
#-508*x^2 + 3766*x^1 - 15551
#237*x^2 - 2574*x^1 + 9469
#-317*x^2 + 4482*x^1 - 15227
#1302*x^2 - 21422*x^1 + 88389
#-1739*x^2 + 33316*x^1 - 1.57932e+05
```

Formiranje kvadratnog splajna

Formiranje kvadratnog splajna objasnilićemo pomoću našeg primera sa 5 dana gde određujemo vrednost za dan 3.

```
In [107]: x=[1,2,4,5]
y=[579,1053,1384,1545]
plot_points(x,y)
```



Koeficijente kvadratnog splajna određujemo rešavanjem sistema linearnih jednačina, ali taj sistem formiramo na drugačiji način u odnosu na sistem koji smo rešavali na početku predavanja.

Sistem linearnih jednačina formiramo pomoću uslova koje kvadratni splajn mora da zadovoljava:

1. Splajnovi moraju da prolaze kroz date tačke. Prvi splajn mora da prolazi kroz prvu i drugu, drugi kroz drugu i treću, treći kroz četvrtu i petu.

2. Prvi izvod splajnova u unutrašnjim tačkama moraju da budu jednaki. Prvi izvod prvog i drugog splajna moraju da budu jednaki u drugoj tački, prvi izvod drugog i trećeg splajna moraju da budu jednaki u trećoj tački.

Formiramo sada redom jednačine za naš primer pomoću uslova 1. i 2.

Pre toga samo napomena da je naš zadatak da odredimo koeficijente a u ovom slučaju 3 kvadratna polinoma, a opšti oblik kvadratnog polinoma koji koristimo je:

$$p(x) = ax^2 + bx + c$$

1. Splajnovi moraju da prolaze kroz date tačke (1,579), (2,1053), (4,1384), (5,1545):

$$\begin{aligned} a_1 \cdot 1^2 + b_1 \cdot 1 + c_1 &= 579 \\ a_1 \cdot 2^2 + b_1 \cdot 2 + c_1 &= 1053 \\ a_2 \cdot 2^2 + b_2 \cdot 2 + c_2 &= 1053 \\ a_2 \cdot 4^2 + b_2 \cdot 4 + c_2 &= 1384 \\ a_3 \cdot 4^2 + b_3 \cdot 4 + c_3 &= 1384 \\ a_3 \cdot 5^2 + b_3 \cdot 5 + c_3 &= 1545 \end{aligned}$$

2. Prvi izvod splajnova u unutrašnjim tačkama (2,1053), (4,1384) moraju da budu jednaki:

$$\begin{aligned} 2 \cdot 2 \cdot a_1 + b_1 &= 2 \cdot 2 \cdot a_2 + b_2 \\ 2 \cdot 4 \cdot a_2 + b_2 &= 2 \cdot 4 \cdot a_3 + b_3 \end{aligned}$$

Vidimo da imamo 8 jednačina, a 9 nepoznatih koeficijenata.

Ovo je normirano kod formiranja kvadratnog splajna, i nije vezano za naš primer. Ako imamo n tačaka, odnosno $n-1$ splajnova, gde svaki splajn ima 3 nepoznata, uvek ćemo pomoću uslova dobiti:

$$1 + 2(n-2) + 1 + 1 + n - 2 = 2 + 2n - 4 + n - 2 = 3n - 4$$

jednačina.

Koliko nepoznatih imamo? Pošto imamo $n-1$ splajnova, a svaki ima 3 nepoznata koeficijenta imamo:

$$3(n-1) = 3n - 3$$

nepoznatih. Znači nedostaje nam jedna jednačina.

Postoji više dodatnih uslova pomoću kojih se rešava problem nedostajuće jednačine i ti uslovi se obično zovu granični uslovi.

Jedan od graničnih uslova koji se često koristi kod kvadratnog splajne je tzv. prirodni splajn kod koga je koeficijent uz x^2 kod prvog splajna jednak nuli. Tađa je prvi splajn prava, a ne kvadratni polinom, pošto je $a_1 = 0$.

Koristimo uslov za prirodni splajn, formiramo i rešavamo sistem jednačina.

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2^2 & 2 & 1 & 0 & 0 \\ 0 & 0 & 4^2 & 4 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4^2 & 4 \\ 0 & 0 & 0 & 0 & 0 & 5^2 & 5 \\ 1 & 0 & -4 & -1 & 0 & 0 & 1 \\ 0 & 0 & 8 & 1 & 0 & -8 & -1 \end{bmatrix} \begin{bmatrix} b_1 \\ c_1 \\ b_2 \\ c_2 \\ b_3 \\ c_3 \\ a_1 \end{bmatrix} = \begin{bmatrix} 579 \\ 1053 \\ 1053 \\ 1384 \\ 1384 \\ 1545 \\ 0 \end{bmatrix}$$

```
In [108]: def quadratic_spline(x,y):
n=len(x)
dim_A=n-4
A=np.zeros((dim_A,dim_A))
b=np.zeros((dim_A,1))

A[0,0]=x[0]
A[0,1]=1

A[1,0]=x[1]
A[1,1]=1

b[0]=y[0]
b[1]=y[1]

row_pos=2
col_pos=2
for j in range(1,n-1):
    for i in range(1,n-1):
        A[row_pos,col_pos+j]=x[i]**(2-j)
        A[row_pos+1,col_pos+j]=x[i+1]**(2-j)

        b[row_pos]=y[i]
        b[row_pos+1]=y[i+1]

    row_pos=row_pos+2
    col_pos=col_pos+3

A[row_pos,0]=1
A[row_pos,2]=-2*x[1]
A[row_pos,3]=-1

row_pos=row_pos+1
col_pos=2

for i in range(2,n-1):
    A[row_pos,col_pos]=2*x[i]
    A[row_pos,col_pos+1]=1
    col_pos=col_pos+3
    A[row_pos,col_pos]=-2*x[i]
    A[row_pos,col_pos+1]=-1
    row_pos=row_pos+1

#print(A)
#print(b)

return np.insert(la.solve(A,b),0,0)
```

```
In [109]: x=[1,2,4,5]
y=[579,1053,1384,1545]

my_kv_splajn = quadratic_spline(x,y)
```

```
In [110]: print(my_kv_splajn)

[  0.      474.    1053.   -154.25 1091.    -512.     304.    -2575.
  6820.   ]
```

```
In [111]: #0*x^2 + 474*x^1 + 105
#-154.25*x^2 + 1091*x^1 - 512
#304*x^2 - 2575*x^1 + 6820
```

```
In [112]: def eval_spline(spline,x,points):
n=len(points)
result=np.zeros(n)
for i in range(n):
    if i<len(x):
        p=spline[0:2]
        results[i]=np.polyval(p,points[i])
    else:
        points[i]=len(x)-1
        p=spline[len(spline)-1:len(spline)-1]
        results[i]=np.polyval(p,points[i])
    for j in range(1,len(x)-1):
        if x[i]<points[i] and points[i]<=x[i+1]:
            p=spline[3*j+3:j+3]
            results[i]=np.polyval(p,points[i])
        break
return results
```

```
In [113]: x=[1,2,4,5]
y=[579,1053,1384,1545]

my_kv_splajn = quadratic_spline(x,y)

print(eval_spline(my_kv_splajn,x,[1]))
print(-154.25*x^2 + 1091*x^1 - 512)
print('tacna_vrednost = 1384')
```

```
[1372.75]
1372.75
tacna_vrednost = 1384
```

```
In [114]: x=[1,2,4,5]
y=[579,1053,1384,1545]

plot_points(x,y)

my_kv_splajn = quadratic_spline(x,y)
xp=np.linspace(1,np.max(x),100)

plt.plot(xp,eval_spline(my_kv_splajn,x,[1]),'ob',markersize=15,markerfacecolor='r')

plt.plot(3,y[2],'ob',markersize=15,markerfacecolor='g')
```

```
Out[114]: <matplotlib.lines.Line2D at 0x228432423c8>
```


Kubni splajn

Kod kubnog splajna između svake dve tačke formira se kubni polinom:

$$ax^3 + bx^2 + cx + d$$

Pošto kubni polinom ima 4 koeficijenta, u slučaju da imamo n tačaka, tj. $n - 1$ splajn ukupno imamo $4n - 4$ nepoznata koeficijenta. Što znači da nam treba $4n - 4$ jednačina.

Kao što smo videli ranije, uslovi za kvadratni splajn nam daju $3n - 4$ jednačine tako da nam nedostaje n uslova.

Iz tog razloga kod kubnog splajna dodaje je se i uslov poklapanja drugih izvoda u unutrašnjim tačkama. Grubo gledano to znači da neće biti naglih promena zakrivljenosti u unutrašnjim tačkama.

Do promena zakrivljenosti dolazi u slučaju da drugi izvod funkcije menja znak u nekoj tački. Na grafiku ispod data je funkcija

$$f(x) = 5x^3 + 2x^2 - 3x$$

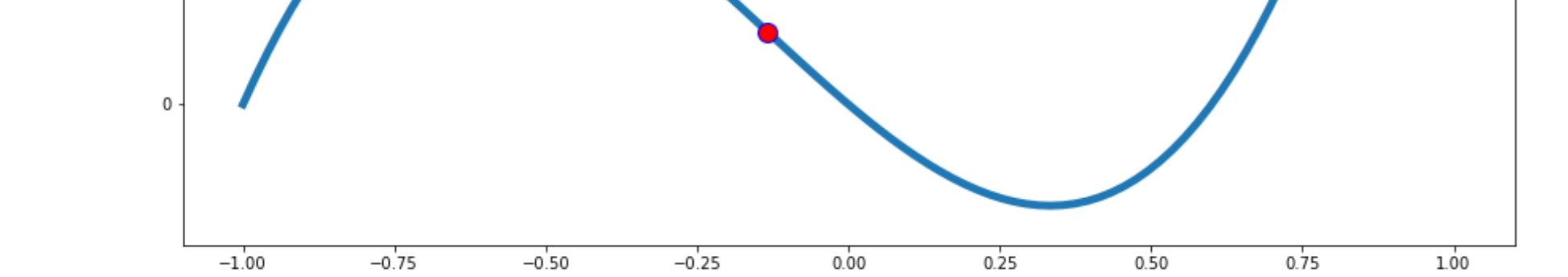
Njen drugi izvod je $f(x) = 30x + 4$. Drugi izvod menja znak u tački $x = -2/15$. To znači da je funkcija do tačke $x = -\frac{2}{15}$ konvexna, a od tačke $x = -\frac{2}{15}$ konkavna, tj. njena zakrivljenost se promenila.

Time što zadržavamo da su drugi izvod splajnova u unutrašnjim tačkama jednaki, obezbedili smo da neće biti promene zakrivljenosti splajna a unutrašnjim tačkama.

Grafik funkcije $f(x) = 5x^3 + 2x^2 - 3x$:



Grafik drugog izvoda funkcije $f(x) = 5x^3 + 2x^2 - 3x$:



Pošto unutrašnjih tačaka ima $n - 2$, a zahtevamo poklapanje drugog izvoda u njima to znači da smo dobili još $n - 2$ uslova. Dakle, nedostaje nam još 2 uslova odnosno još dve jednačine. Ta dva uslova biraju se na različite načine, a u nastavku su navedena dva najčešće korišćena.

1. Prirodni splajn (natural spline) – pretpostavimo da su drugi izvodi u krajnim tačkama jednaki nuli.

Ovaj uslov znači da će u okolini pre prve i nakon poslednje tačke funkcija biti linearna jer u tački u kojoj je drugi izvod nula funkcija manje zakrivljenost pa je tokom te promene u jednom malom delu linearna (pogledati grafik funkcije iznad). Cilj ovog uslova je da splajn oko krajeva nema oscilacije, tj. da eventualne ekstrapolacije blizu krajnjih tačaka ne budu "nepredvidive". Naziv "prirodan" potiče od tumačenja splajna kao elastičnog štapa koji smo počeli da savijamo u krajnjim tačkama.

2. "Not-a-knot" splajn – pretpostaviti jednakost trećeg izvoda u drugoj i preposlednjoj tački. Ovaj splajn zove se "Not-a-knot" jer faktički druga i preposlednja tačka nisu više čvorovi, u smislu da je splajn pre i posle njih isti. Recimo da imamo neku tačku x_i , pogledaćemo kako izgleda jednakost dva kubna polinoma u toj tački redom do trećeg izvoda:

$$a_1x_i^2 + b_1x_i^2 + c_1x_i + d_1 = a_2x_i^3 + b_2x_i^2 + c_2x_i + d_2$$

$$3a_1x_i^2 + 2b_1x_i + c_1 = 3a_2x_i^2 + 2b_2x_i + c_2$$

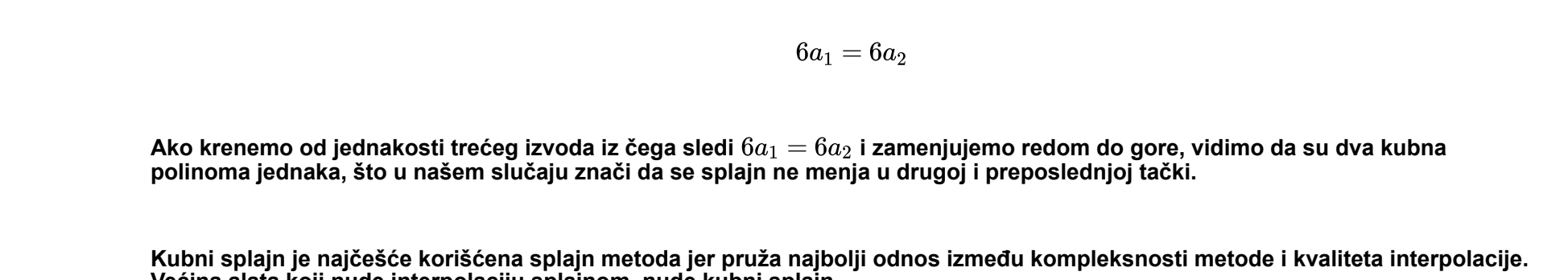
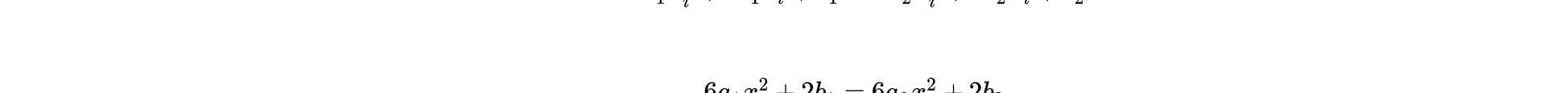
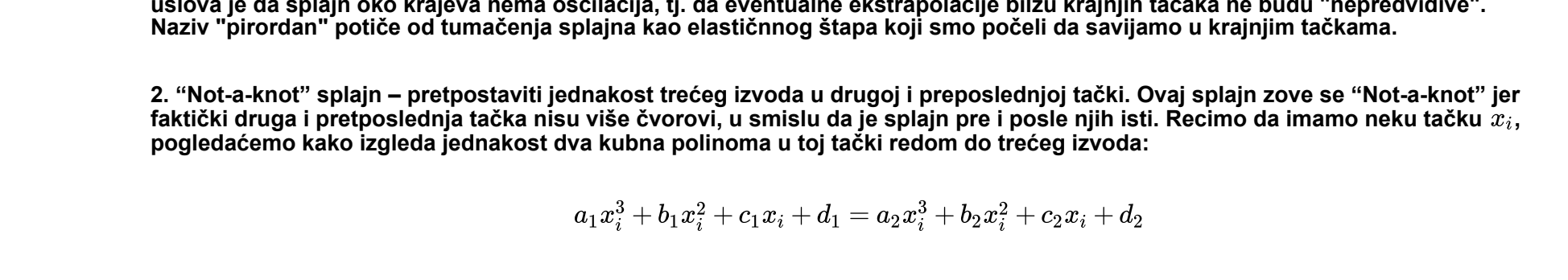
$$6a_1x_i^2 + 2b_1 = 6a_2x_i^2 + 2b_2$$

$$6a_1 = 6a_2$$

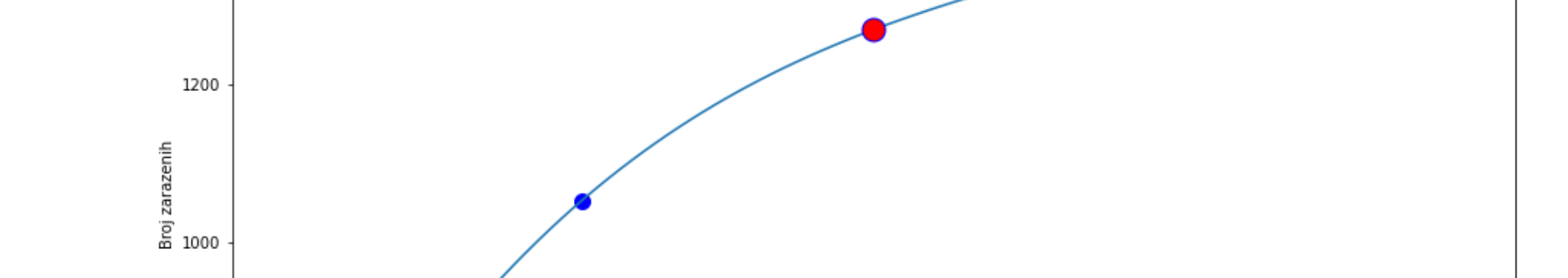
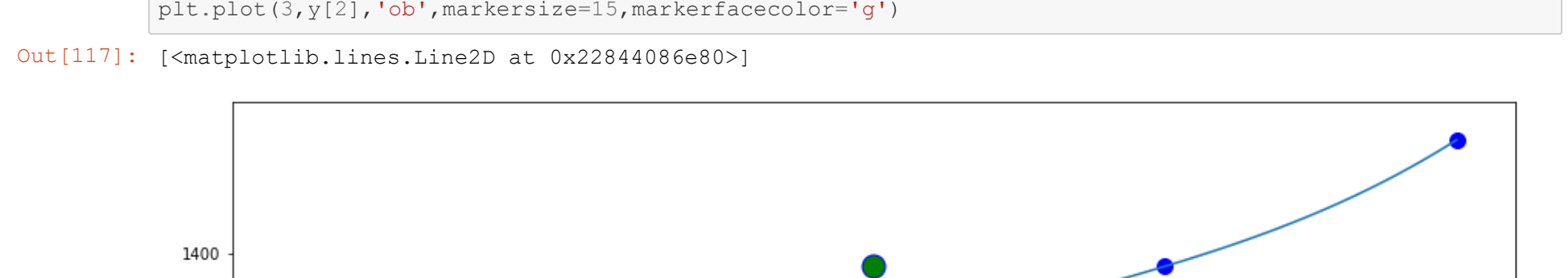
Ako krenemo od jednakosti trećeg izvoda iz čaga sledi $6a_1 = 6a_2$ i zamenjujemo redom do gore, vidimo da su dva kubna polinoma jednaka, što u našem slučaju znači da se splajn ne menja u drugoj i preposlednjoj tački.

Kubni splajn je najčešće korišćena splajn metoda jer pruža najbolji odnos između kompleksnosti metode i kvaliteta interpolacije. Većina alata koji nude interpolaciju splajnom, nude kubni splajn.

Prikazaćemo sada upotrebu kubnog splajna za naša dva primera (sa 4 i 10 tačaka).



Poredimo kubni i kvadratni splajn.

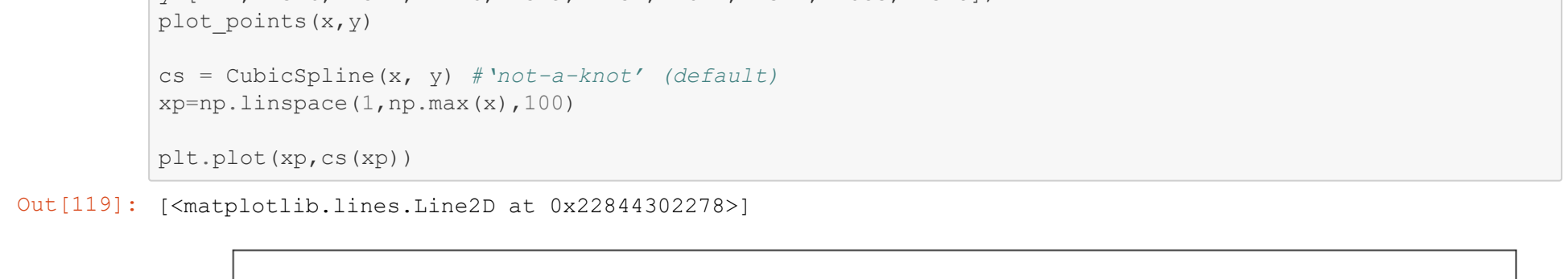


Ekstrapolacija

Ekstrapolacija je upotreba interpolacionih polinoma za izračunavanje vrednosti van opsega x-koordinata tačka koje su date.

U našem primeru ekstrapolacija bilo bi izračunavanje broja zaraženih za svaki dan posle dana 10, ili teoretski moguće za svaki dan pre dana 1.

Ekstrapolacija može da rezultuje nepredvidim vrednostima, naročito ako se koristi jedan interpolacioni polinom koji, kao što smo videli, može da ima velike oscilacije ako imamo puno datih tačaka.



U nastavku je primer "prirodnog" splajna kod koga je ekstrapolacija bar u okolini krajnjih tačaka bolja nego kod "not-a-knot".

