2-2-JednadzbeS

```
var('x y a b c')
```

```
(x, y, a, b, c)
```

Jednadžbe

U Sageu kao znak jednakosti u jednadžbama stoji "==", jer je uobičajeni znak "=" , kako smo gore već vidjeli, rezerviran za pridjeljivanje vrijednosti simbolima odnosno pridjeljivanje imena izrazima. Rješavanje jednadžbi se izvodi funkcijom `solve()`:

```
sol = solve(2*x^2 - 1 == 0, x); sol
```

```
[x == -1/2*sqrt(2), x == 1/2*sqrt(2)]
```

Valja primijetiti slijedeće:
1. Prilikom poziva funkcije `solve()` treba eksplicitno naznačiti po kojoj varijabli se traži rješavanje.
2. Pronađena su oba rješenja kvadratne jednadžbe.
3. Rezultat je ispisan u obliku liste `[...]` jednadžbi. To omogućuje uvrštavanje rješenja u neki drugi izraz, ili u samu originalnu jednadžbu radi provjere. (Zato smo gore toj listi rješenja odmah pridjelili ime `sol`).

Pristup pojedinim elementima liste ostvaruje se sintaksom `lista[n]` gdje je n indeks elementa i brojanje počinje s nulom tako da prvi element ima indeks 0. (O listama će biti više riječi u slijedećem poglavlju.). Uvrštavanje tj. supstitucija u izrazu izvodi se metodom subs(), čiji argument može biti i jednadžba:

```
(x+y).subs(sol[0])
```

```
y - 1/2*sqrt(2)
```

```
(2*x^2 - 1 == 0).subs(sol[1])
```

```
0 == 0
```

Transformacija izraza pomoću supstitucije je vrlo korisna operacija o kojoj će više riječi biti kasnije. Zasad pokažimo samo njenu najčešću upotrebu

Transformacija izraza pomoću supstitucije je vrlo korisna operacija o kojoj će više riječi biti kasnije. Zasad pokažimo samo njenu najčešću upotrebu: pridjeljivanje vrijednosti simbolima u nekom izrazu:

```
( (a+b-c)^4 ).expand()
```

```
a^4 + 4*a^3*b - 4*a^3*c + 6*a^2*b^2 - 12*a^2*b*c + 6*a^2*c^2 +
4*a*b^3 - 12*a*b^2*c + 12*a*b*c^2 - 4*a*c^3 + b^4 - 4*b^3*c +
6*b^2*c^2 - 4*b*c^3 + c^4
```

```
_.subs(b==c)
```

```
a^4
```

```
_.subs(a==2)
```

```
16
```

Funkcija solve() može rješavati i sustave jednadžbi, ako se kao argumenti zadaju liste jednadžbi odnosno liste varijabli:

```
sol2 = solve([x^2 + y^2 == 1,  x - 2*y == 0], [x, y]); sol2
```

```
[[x == -2/5*sqrt(5), y == -1/5*sqrt(5)], [x == 2/5*sqrt(5), y ==
1/5*sqrt(5)]]
```

Da bi dobili numeričke vrijednosti ovih rješenja ne možemo (kao u npr. Mathematici) jednostavno primijeniti funkciju n() na ovu listu rješenja, jer n() nije metoda liste (a ni jednadžbe), već samo primitivnijih objekata, pa je potrebno indeksiranjem ekstrahirati izraze s desne strane gornjih jednadžbi npr.

```
(x.subs(sol2[0][0]).n(), y.subs(sol2[0][1]).n())
```

```
(-0.894427190999916, -0.447213595499958)
```

(* do Zadatka) Koristeći nešto naprednije tehnike možemo ovo izvesti elegantnije tako da iteriramo preko liste rješenja i to tako da se **`n()`** primjenjuje samo na desne strane gornjih jednadžbi. To je lakše izvesti tako da zatražimo od **`solve()`** rješenje, ne u obliku liste jednadžbi, već u obliku liste *riječnika*. Vidi python tutorial: dictionary

```
soln = solve([x^2 + y^2 == 1,  x - 2*y == 0], x, y, solution_
dict=True); soln
```

```
[{y: -1/5*sqrt(5), x: -2/5*sqrt(5)}, {y: 1/5*sqrt(5), x:
2/5*sqrt(5)}]
```

Sad primjenjujemo moćni postupak obuhvaćanja liste (list comprehension) koji omogućuje kreiranje nove liste na osnovi stare u jednom koraku (vidi kasnije poglavlje o programiranju):

```
[s[y] for s in soln]
```

```
[-1/5*sqrt(5), 1/5*sqrt(5)]
```

odnosno

```
[s[y].n() for s in soln]
```

```
[-0.447213595499958, 0.447213595499958]
```

Konačno, za ljepši ispis možemo koristiti Pythonovo formatiranje stringova (slično kao u C-u)

```
["x = %f  y = %f" % (s[x].n(), s[y].n()) for s in soln]
```

```
['x = -0.894427  y = -0.447214', 'x = 0.894427  y = 0.447214']
```

Zadatak 2-2.1: Riješite sustav jednadžbi

$$x^4 + a^4 = 1\ , \quad x^2 + a^2 = 1\ ,$$

Koliko ima rješenja?

Ukoliko sustav jednadžbi ima beskonačno rješenja, dobit ćemo rješenje koje uključuje slobodni parametar ili više njih:

```
solve([x+y == 3, 2*x+2*y == 6],x,y)
```

```
[[x == -r1 + 3, y == r1]]
```

```
solve([cos(x)*sin(x) == 1/2, x+y == 0],x,y)
```

```
[[x == 1/4*pi + pi*z6, y == -1/4*pi - pi*z6]]
```

Gore je r<broj> neki realni, a z<broj> neki cijeli broj. Ovo radi samo sa sustavom jednadžbi, a ne i s jednom jednadžbom ...

```
solve(sin(x)==0, x)
```

```
[x == 0]
```

... pa ukoliko želimo i ovo rješenje zapisano kao skup rješenja možemo iskoristiti trik da kreiramo sustav u kojem je druga jednadžba razvezana:

```
sol3=solve([sin(x)==0,y==0], x,y); sol3
```

```
[[x == pi + 2*pi*z15, y == 0], [x == 2*pi*z17, y == 0]]
```

```
[a for a,b in sol3]   # (*)
```

```
[x == pi + 2*pi*z15, x == 2*pi*z17]
```

```
show(_)
```

$$[x = \pi + 2\,\pi z_{15}, x = 2\,\pi z_{17}]$$

`solve()` daje simboličke (analitičke) rješenja jednadžbi. Međutim, neka rješenja npr. jednadžbi viših stupnjeva nije moguće analitički zapisati. Npr, za slijedeću jednadžbu `solve()` nam daje samo jedno trivijalno realno rješenje:

```
eq = 9*x^6 + 4*x^4 + 3*x^3 + x - 17 == 0
solve(eq, x)
```

```
[x == 1, 0 == 9*x^5 + 9*x^4 + 13*x^3 + 16*x^2 + 16*x + 17]
```

No znamo da ta jednadžba, šestog stupnja, mora imati šest kompleksnih rješenja. Ostalih pet se ne da zapisati drugačije nego kao numeričke (floating point) brojeve. Da bismo dobili ta rješenja koristimo metodu `roots()`, gdje opcijom `ring=CC` tražimo rješenja u prstenu kompleksnih brojeva:

```
eq.roots(x, multiplicities=False, ring=CC)
```

```
[-1.10301507262981, 1.00000000000000, -0.491102035999093 -
0.988331495372071*I, -0.491102035999093 + 0.988331495372071*I,
0.542609572314000 - 1.05431152068711*I, 0.542609572314000 +
1.05431152068711*I]
```

```
len(_)
```

```
6
```

Daljnji je problem da je i `roots()` zapravo analitički rješavač jednadžbi (koristi egzaktne, a ne numeričke metode), a neke jednadžbe se ne mogu analitički egzaktno riješiti, poput onih koje uključuju transcendentalne funkcije.

```
eq2 = 2 * arctan(x) == x^2
eq2.roots(ring=CC)
```

```
Traceback (click to the left of this block for traceback)
...
TypeError: Cannot evaluate symbolic expression to a numeric
value.
```

U tom slučaju moramo pribjeći pravom numeričkom rješavanju tako da definiramo funkciju čije nul-točke su ekvivalentne rješenjima jednadžbe i onda ih tražimo pomoću funkcije `find_root()`. Problem s `find_root()` je da mu se treba dati interval u kojem traži nul-točku i da će pronaći samo jednu. Nakon toga moguće treba tražiti dalje u drugačijem intervalu koji ne uključuje pronađenu nul-točku itd.

```
f(x) = 2 * arctan(x) - x^2  # ekvivalentno eq2
find_root(f(x), 0, 10)
```

```
0.0
```

```
find_root(f(x), 0.1, 10)
```

```
1.3717743420150883
```

```
find_root(f(x),1.4, 10)
```

```
Traceback (click to the left of this block for traceback)
...
RuntimeError: f appears to have no zero on the interval
```

```
find_root(f(x), -1., 1)
```

```
-2.3268138266719179e-21
```

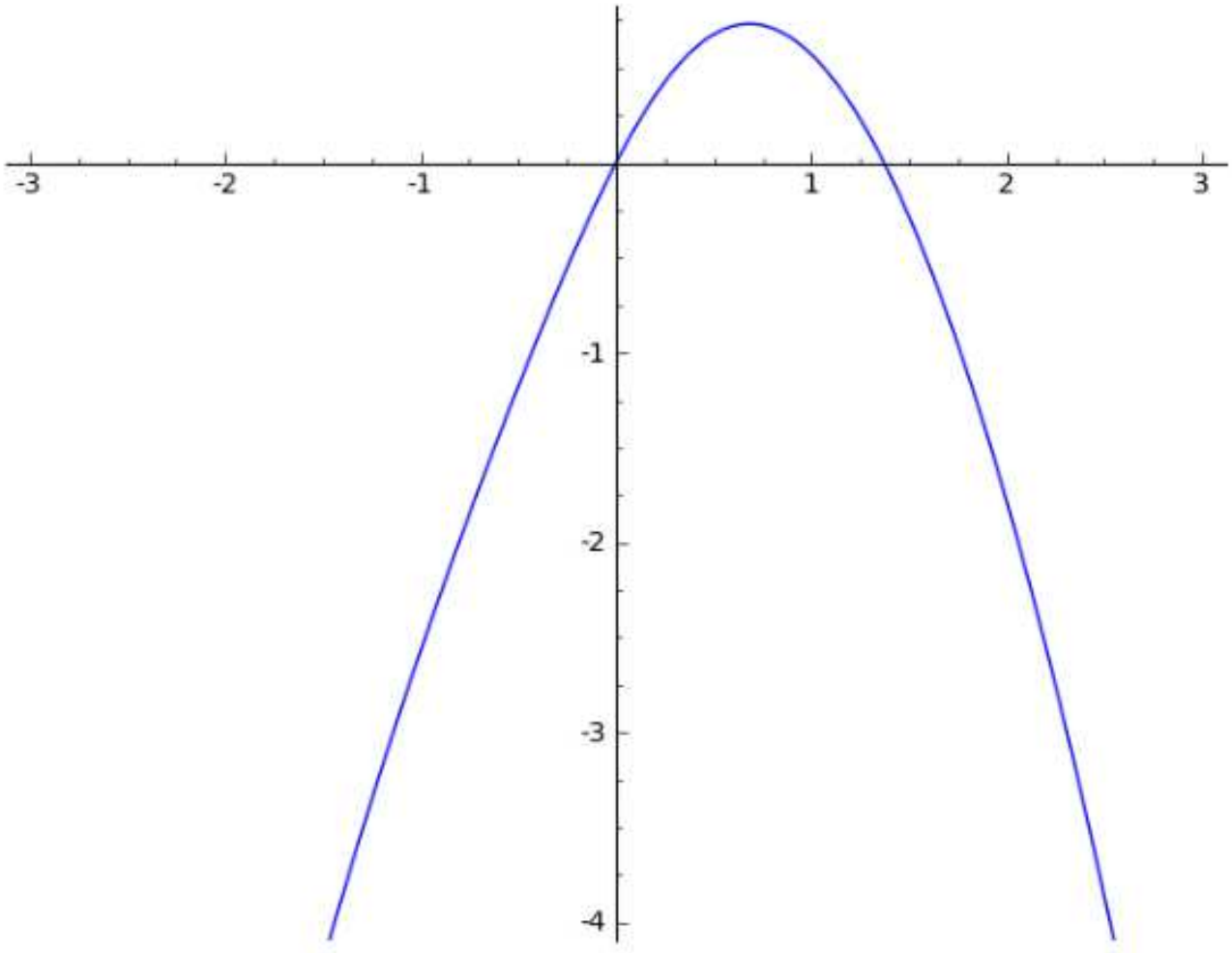
Ovo zadnje rješenje je zapravo 0 jer funkcija `find_root()` radi numeriku s konačnom preciznošću, koja po defaultu otprilike odgovara preciznosti *double precision* floating point varijabli u Fortranu ili C-u, što može promijeniti pomoću opcionalnog argumenta xtol:

```
find_root(f(x), -1., 1, xtol=1e-30)
```

```
4.3470345330696587e-32
```

Vidimo da se s povećanjem radne preciznosti rješenje još više približilo nuli. Možemo se uvjeriti da su ova gore dva rješenja zaista jedina, tako da skiciramo graf funkcije f(x) i vidimo da siječe apscisu na samo dva mjesta. Koristimo funkciju `plot()` o kojoj će kasnije biti više riječi.

```
plot(f(x), (x, -3, 3), ymin=-4)
```



Zadatak 2-2.2: Riješite jednadžbu

$$\tan x - \frac{x}{10} == 0 \ .$$

 Zadatak 2-2.3: (*) Pronađite numeričku vrijednost nekog kompleksnog rješenja jednadžbe $\sin x = 2$ i provjerite uvrštavanjem.

 Zadatak 2-2.4 Riješite nejednadžbu $x^2 + x - 12 < 0$.

 Zadatak 2-2.5 Pronađite pozicije lokalnog minimuma te lokalnog maksimuma gama funkcije $\Gamma(x)$ koji su najbliži točki x = 0.

 Zadatak 2-2.6 (*) Pronađite pozicije minimuma i maksimuma Besselove funkcije $J_1(x)$ koji su najbliži točki $x = 0$.