Sage računalno okruženje za fizičare

Distribucija 1.1

Krešimir Kumerički

14 October, 2015

Sadržaj

Premda je ovaj dokument nastao radi potreba studija fizike na Fizičkom odsjeku Prirodoslovno-matematičkog fakulteta Sveučilišta u Zagrebu, najveći dio (zapravo sve osim 5. poglavlja) bi trebao biti razumljiv i upotrebljiv svakome tko želi naučiti raditi u Sageu.

Ispisi računalnog koda u ovom dokumentu su dobiveni unutar Sage računalnog okruženja, verzija 6.9.

POGLAVLJE 1

Uvod

1.1 Python vs. IPython vs. Sage

- Python je računalni jezik opće namjene. Zahvaljujući, s jedne strane, dobroj čitljivosti koda, a s druge brojnim kvalitetnim bibliotekama za numeriku, simboliku, crtanje itd., Python se često koristi u istraživanjima i edukaciji iz fizike i srodnih područja znanosti i tehnike.
- IPython je sučelje za Python namjenjeno interaktivnom radu. Postoje inačice za terminalski rad, grafičko sučelje i sučelje putem WWW preglednika.
- Sage je matematički softver koji udružuje niz postojećih matematičkih (i drugih) biblioteka u zajedničko sučelje zasnovano na Pythonu, s ciljem kreiranja alternative komercijalnim softverima poput Mathematice, Maplea ili Matlaba.

Zbog njegove univerzalnosti, u ovom dokumentu koristit ćemo uglavnom Sage.

1.2 Zašto Python/Sage?

1. Python je izrazito elegantan za upotrebu. Popularan je u znanstvenoj zajednici, pa postoji velik broj korisnih Python biblioteka i za specijalizirane znanstvene namjene..
2. Znanost mora biti reproducibilna i u načelu vječna. Računalni kod koji ovisi o softveru "zatvorenog" koda to onemogućuje.
3. Sage je najrazvijeniji sustav za računalnu algebru (CAS - *Compute Algebra System*) otvorenog koda (*open source*)

Za detaljniju diskusiju o izboru softvera za računanje u znanosti, vidi Johansson, Introduction to scientific computing with Python, te, također, Koepke, 10 Reasons Python Rocks for Research (And a Few Reasons it Doesn't).

POGLAVLJE 2

Sučelje

2.1 Sage radni list i ćelije

Radni list (*worksheet*) je skup računalnog Sage kôda s rezultatima i tekstualnim opisom koji je

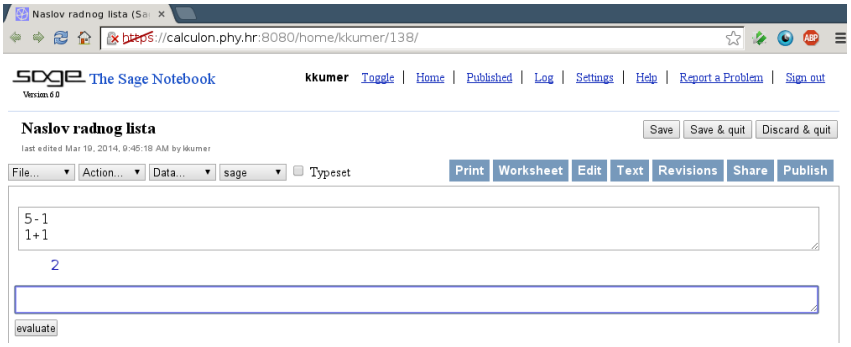
- prikazan na jednoj stranici u WWW pregledniku
- može se pohraniti u jednu datoteku (ekstenzija `.sws`)

Svi radni listovi jednog korisnika dostupni su putem poveznice `Home` na vrhu svakog radnog lista. Čitavo ovo grafičko sučelje naziva se Sage bilježnica (*Sage notebook*). (Postoji i obično tekstualno terminalsko sučelje.)

Račun, tj. računalni kôd radnog lista je organiziran u tzv. ćelije (*cell*). Svaka računska (*input*) ćelija je ograđena pravokutnikom i izvršava se *kao cjelina* na jedan od dva načina:

1. pritiskom na kombinaciju tipki `shift-Enter` dok je kursor bilo gdje u ćeliji (Pritisak samo na `Enter` otvara novi redak u ćeliji),
2. klikom mišem na gumb `Evaluate` koji se prikazuje ispod aktivne ćelije.

Sage ispisuje rezultat neposredno ispod ćelije. Ako se eksplicitno ne zatraži drugačije (npr. naredbom `print`), bit će ispisan samo rezultat zadnjeg računa/komande/retka u toj ćeliji (ali izvršit će se naravno cijela ćelija).



Napomena: Slika prikazuje Sage radni list otvoren u WWW pregledniku s jednom izvršenom ćelijom i jednom praznom aktivnom ćelijom. U ovom dokumentu ćemo primjere kôda davati onako kako bi izgledali u terminalskom sučelju gdje bi korisnikov upis slijedio nakon prompta `sage:`. Taj prompt ne smeta ni u ćelijama radnog lista, dakle primjeri iz ovog dokumenta se mogu slobodno kopirati u Sage grafičko sučelje sa ili bez prompta. Jedina razlika je da ukoliko više redaka iz ovog dokumenta kopiramo u jednu ćeliju, nećemo vidjeti međurezultate (ako ih trebamo, dodamo `print` ispred odgovarajućih redaka. Npr. kôd iz ćelije sa slike će u ovom dokumentu izgledati ovako:

```
sage: 5-1
4
sage: 1+1
2
```

Otvaranje novih ćelija se izvodi stavljanjem miša između postojećih ćelija (ili na kraj radnog lista) i klikom u trenutku kad se pojavi plava horizontalna linija. `Shift-klik` će otvoriti tekstualnu ćeliju u koju stavljamo slobodni tekst i komentare. Brisanje ćelije se izvodi brisanjem sadržaja i onda pritiskom na backspace.

Zadatak 1

Kreirajte novu input ćeliju i izračunajte `2+3`. Kreirajte zatim tekstualnu ćeliju s nekim tekstom iznad ove. Izbrišite obje ćeiije.

Važno svojstvo Sage radnog lista je da svaku ćeliju možemo i naknadno editirati i onda ponovno izvršiti. (Editiranje tekstualnih ćelija iniciramo dvostrukim klikom miša negdje na ćeliji.) To je onda sve skupa sličnije radu u tabličnom kalkulatoru (*spreadsheet*) nego standardnom programiranju. Osim samog sadržaja ćelija, trenutno stanje radnog lista (tzv. sesija, *session* ili proces) je određeno i vrijednostima varijabli u memoriji računala, a to općenito ovisi o redosljedu kojim su ćelije izvršene. Resetiranje tog stanja se izvodi putem izbornika `Action` pa `Restart worksheet`. (Što je korisna operacija u trenucima kad se Sage radni list počne neočekivano ponašati.) Zbog toga je po završetku rada dobro prekontrolirati konzistentnost i reproducibilnost cijelog računa tako da resetiramo sesiju i izvršimo sve ćelije odjednom pomoću izbornikai `Action` pa `Evaluate All`.

Svakom radnom listu pripada nezavisni Sage proces s nezavisnim varijablama.

2.2 Elementarno računanje

Sage se može koristiti kao obični kalkulator proizvoljne preciznosti:

```
sage: 3*2+1
7


sage: sqrt(9)
3


sage: factorial(22)
1124000727777607680000


sage: 13^23
41753905413413116367045797


sage: 13.^23
4.17539054134131e25
```

Uočite različit tretman cijelih (*integer*) i realnih (*floating point*) brojeva. Cijeli brojevi se uvijek tretiraju egzaktno, bez zaokruživanja ili odbacivanja nekih znamenaka.

Eksplicitno pisanje znaka množenja `*` se djelomično može izbjeći pozivanjem funkcije `implicit_multiplication(True)`, ali je i dalje nužno pisati `*` prije zagrada (kako bi bilo jasno da nije riječ o pozivanju funkcije). U ovom dokumentu se uvijek eksplicitno piše znak množenja.

Za zapis velikih brojeva može se koristiti standardni Fortran/C zapis po kojem se npr. $3.2 \cdot 10^4$ zapisuje ovako:

```
sage: 3.2e4
32000.0000000000
```

Pridruživanje vrijednosti varijablama izvodi se znakom jednakosti "=".

```
sage: x = 4
sage: print x
4

sage: x + 3
7
```

Standardne matematičke funkcije i konstante imaju uobičajena imena i ponašanje:

```
sage: sin(pi)
0

sage: log(e)
1

sage: log(sin(pi))
-Infinity

sage: oo + 1
+Infinity
```

Zadatak 2

Izračunajte $\sqrt{2\sqrt{e^{\pi}}}$.

Primijetite da niste dobili numerički već simbolički rezultat. Sage će se uvijek, ako je moguće, odlučiti za egzatni simbolički rezultat. Ako nas ipak zanima numeričko približenje možemo koristiti funkciju `numerical_approx()`, koja se skraćeno zove i `n()`:

```
sage: n(sqrt(2*sqrt(e^pi)))
3.10176639383605
```

Funkcija n() ima i *opcionalni argument* `digits` kojim možemo zatražiti i veći broj decimala numeričkog približenja:

```
sage: numerical_approx(sqrt(2*sqrt(e^pi)), digits=50)
3.1017663938360514951983183136487613918314124732866
```

Funkcije često imaju brojne opcionalne argumente (*keyword arguments*) koji se navode proizvoljnim redom, ali svakako iza obaveznih argumenata.

Zadatak 3

Odredite broj π s točnošću od 500 decimala: pridružite tu vrijednost varijabli `x` i ispišite je. Odredite `sin(x)`.

S kompleksnim brojevima radimo jednostavno. Samo treba imati na umu da je imaginarna jedinica $\sqrt{-1}$ reprezentirana velikim slovom `I`.

```
sage: I^2
-1
```

Zadatak 4

Izračunajte $e^{i\pi} + 1$.

Za upis matematičkih izraza u tekstualne ćelije stavljamo LaTeX kôd unutar dolarskih znakova. Tako se upis

```
$\alpha$
```

unutar retka prikazuje kao α. Veće jednadžbe koje trebaju stajati u posebnom redu upisuju se između parova dolarskih znakova u novom redu. Tako se:

```
$$ E = \gamma m c^2 $$
```

prikazuje kao

$$E = \gamma m c^2$$

Za formatiranje ispisa rezultata računa koristimo operator `%`, te standardne stringove za formatiranje (poznate iz sprintf() C funkcije): Npr. %5.3f formatira realni broj na 5 mjesta ukupno i 3 mjesta iza decimalne točke.)

```
sage: print "%s = %10.8f ..." % ('Ludolfov broj', pi)
Ludolfov broj = 3.14159265 ...
```

2.3 Help sustav

Da bismo pronašli potrebnu funkciju te način i primjere njene upotrebe služimo se slijedećim pristupima Sage dokumentaciji. Kao prvo, tu je `TAB-nastavljanje` (*TAB-completion*): započnemo li pisati ime neke funkcije, pritisak na tipku TAB dovršava pisanje njenog imena ako je nastavak jedinstven, a ako nije dobivamo popis svih mogućnosti (pa željenu odaberemo mišem ili kursorskim tipkama i tipkom `Enter`).

Dokumentaciju konkretne funkcije dobijemo tako da nakon imena funkcije stavimo upitnik i onda pritisnemo TAB (dobiveni tekst se može "odlijepiti" u poseban prozor pritiskom na *pop-up* link u njegovom desnom gornjem kutu). Iz dobivene dokumentacije možemo *cut-and-paste*-ati primjere u radni list. (Ukoliko umjesto jednog stavimo dva upitnika dobijemo cijeli ispis kôda koji definira tu funkciju.)

Zadatak 5

Otvorite novu ćeliju, utipkajte `int` pa pritisnite TAB. Odaberite funkciju `integrate`, dodajte upitnik `?` pa ponovno stisnite TAB da dobijete osnovnu dokumentaciju. Prekopirajte jedan primjer upotrebe ove funkcije u istu ćeliju i izvršite je.

Za pretraživanje po cijelom tekstu Sage dokumentacije postoji funkcija `search_doc()`, ali u praksi je efikasnije koristiti google koji relevantnije rezultate smješta na vrh. Kako je "sage" relativno česta riječ trik je prilikom pretraživanja koristiti kao ključnu riječ "sagemath" (što je alternativno ime tog softvera i nalazi se u WWW adresi Sage projekta).

Zadatak 6

Odredite numeričku vrijednost logaritma imaginarnog broja $16i$ u bazi 4, dakle $\log_4(16i)$, tako da u dokumentaciji nađete kako se odabire drugačija baza funkcije `log()` (defaultna baza je ona za prirodni logaritam tj. $e = 2.718...$).

Zadatak 7

Izvrijednite Eulerovu gama funkciju za z=1/2, dakle $\Gamma(1/2)$, tako da pronađete u dokumentaciji odgovarajuću funkciju.

Zadatak 8

Proučite upotrebu funkcije `sum()` za zbrajanje matematičkih redova i izračunajte:

(a) $1 + 3 + 5 + \cdots + 61$

(b) $\sum_{x=1}^{\infty} \frac{1}{x^2}$

2.4 Poruke o greškama

Ako se ogriješimo o matematička ili sintaktička pravila Sage će nam uzvratiti porukom o grešci. Klik mišem lijevo od vrha te poruke daje opširnije informacije (inače, drugi klik potpuno skriva poruku što se može koristiti i za skrivanje svih nepregledno dugačkih ispisa rezultata računa.) Za interpretaciju opširnije poruke potrebno je znanje Python programskog jezika, no ključna informacija je obično u zadnjem retku, koji je odmah vidljiv.

```
sage: 1/0
Traceback (most recent call last):
...
ZeroDivisionError: rational division by zero
```

```
sage: sin[2.3]
Traceback (most recent call last):
```

```
...
TypeError: 'Function_sin' object has no attribute '__getitem__'
```

Početniku će te poruke izgledati nejasno, ali s vremenom će poprimati sve više smisla i treba ih uvijek čitati. Npr. gornja greška *"object is unsubscriptable"* je posljedica toga što smo za poziv funkcije `sin` umjesto okruglih zagrada upotrijebili uglate, koje služe za pristup elementima (tj. indeksima, subskriptima) polja i matrica.

POGLAVLJE 3

Programiranje

3.1 Liste i drugi spremnici

3.1.1 Liste

Liste su središnji objekti programiranja u Sageu i Pythonu. Srodne su poljima (*array*) iz standardnih programskih jezika (C, Fortran), ali imaju bitno više svojstava. Elementi lista mogu biti praktički bilo koji objekti.

```
sage: t1 = [1, 3, 5, 7, 9, 11]; t1
[1, 3, 5, 7, 9, 11]
sage: t2 = [sin, cos, tan]; t2
[sin, cos, tan]
sage: t3 = [[], ["jedan"], t1]; t3
[[], ['jedan'], [1, 3, 5, 7, 9, 11]]
```

Liste se mogu npr. zbrajati, množiti i mjeriti im se duljina:

```
sage: print t1+t2
[1, 3, 5, 7, 9, 11, sin, cos, tan]
sage: 7*[3]
[3, 3, 3, 3, 3, 3, 3]
sage: print len(t1)
6
```

Kako je Python objektno orijentirani računalni jezik, mnoge operacije se izvode putem tzv. *metoda* objekta, a to su objektu pridružene funkcije koje se pozivaju sintaksom `objekt.metoda()`. I liste su objekti pa imaju niz metoda (vidi popis), od kojih je daleko najvažnija metoda `append()` koja služi za dodavanje elemenata na kraj liste:

```
sage: t1.append(13); t1
[1, 3, 5, 7, 9, 11, 13]
```

Pojedinim elementima liste se pristupa pomoću indeksiranja, gdje prvi element liste ima indeks 0, drugi ima indeks 1 itd. Pojedine elemente možemo promijeniti običnim pridruživanjem pomoću znaka jednakosti:

```
sage: t1[1] = "tri"
sage: print t1
[1, 'tri', 5, 7, 9, 11, 13]
```

Ukoliko želimo pristupiti većem broju elemenata od koristi su tzv. *rezovi* liste (engl. *slice*). Općenita sintaksa reza je `lista[početak:kraj:korak]`, gdje je `početak` uključen u rez, a `kraj` nije:

```
sage: t1[2:6]
[5, 7, 9, 11]
sage: t1[2:6:2]
[5, 9]
```

Ukoliko rez ide od samog početka ili sasvim do kraja liste, odgovarajuće indekse možemo izostaviti:

```
sage: t1[2:]
[5, 7, 9, 11, 13]
```

Negativni indeksi nam omogućuju da brojimo od kraja liste ...

```
sage: t1[-2:]    # zadnja dva elementa
[11, 13]
```

... a negativan korak da rez ide unatrag tj. da se izvrne redosljed elemenata:

```
sage: t1[::-1]  # lista natraške
[13, 11, 9, 7, 5, 'tri', 1]
```

Zadatak 1

Promijenite u gornjoj listi t1 element `"tri"` (ne element s indeksom=3!) natrag u broj `3`, ali ne eksplicitnom upotrebom indeksa `1`, već tako da "pronađete" indeks elementa `"tri"` ``pomoću metode ``index()``. (Nije dozvoljeno upotrijebiti znamenku `1`).

Za konstrukciju liste cijelih brojeva, vrlo je korisna funkcija `range(početak, kraj, korak)` gdje treba imati na umu da će konstruirana lista početi s elementom `početak`, ali će završiti s elementom `kraj-1`.

```
sage: range(3)  # rezultira listom od tri elementa
[0, 1, 2]
```

Za konstrukciju liste realnih brojeva, vidi dolje odjeljak NumPy polja.

3.1.2 Obuhvaćanje liste

U praksi je kirurško mijenjanje pojedinih elemenata liste, kao u gornjem zadatku, vrlo rijetko. Listama se u pravilu rukuje kao cjelinama i najčešće se djeluje na sve njihove elemente. U standardnim računalnim jezicima u tu svrhu se koriste petlje, ali u Pythonu je najelegantnije koristiti tzv. *obuhvaćanje liste* (engl. *list comprehension*):

```
sage: [n+1 for n in t1]
[2, 4, 6, 8, 10, 12, 14]
```

Kombiniranjem obuhvaćanja liste s funkcijom `range()` možemo konstruirati najrazličitije liste. Npr. lista od pet slučajnih brojeva se može konstruirati ovako:

```
sage: [random() for k in range(5)]
[0.2411785331722212 7, 0.4504912135031752, 0.28942549153153674,
     0.86208567873778, 0.09328097538429081]
```

Često se javlja potreba za izborom elemenata liste koji zadovoljavaju neki kriterij. To se može izvesti obuhvaćanjem liste uz dodatni uvijet:

```
sage: t5 = range(1, 11)
sage: [n for n in t5 if is_even(n)]     # izbor parnih elemenata
[2, 4, 6, 8, 10]
```

Ovdje smo koristili funkciju `is_even()` koje je jedna iz velike porodice tzv. *predikata*. Predikat je termin iz matematičke logike koji označava funkciju koja poprima isključivo vrijednosti True ili False. Većina predikata definiranih u Sageu počinje s `is_` (jer odgovaraju na pitanje "da li je ..."). Ispišimo ih (Koristimo prvo funkciju `dir()` koja ispisuje imena svih trenutno poznatih objekata i, drugo, to da su stringovi kao liste znakova pa možemo koristiti rezove na njima):

```
sage: [p for p in dir() if p[:3]=='is_']
['is_2_adic_genus', 'is_32_bit', 'is_64_bit', 'is_AbelianGroup',
'is_AbelianGroupElement', 'is_AbelianGroupMorphism',
         <... odrezan ispis ...>
'is_odd', 'is_optimal_id', 'is_pAdicField', 'is_pAdicRing',
'is_package_installed', 'is_power_of_two', 'is_prime', 'is_prime_power',
'is_pseudoprime', 'is_square', 'is_squarefree', 'is_triangular_number']
```

Zadatak 2

Konstruirajte listu svih prim-brojeva manjih od 100. Koliko ima prim-brojeva manjih od $10^4, 10^5, 10^6, \ldots$? Usporedite rezultate (i brzinu njegovog dobivanja) s ugrađenom funkcijom prime_pi().

Napomena: Mjerenje vremena potrebnog da se izvrši neka ćelija izvodi se stavljanjem `%time` u prvi red, a prekid računa koji traje predugo izvodi se putem izbornika `Action` pa `Interrupt` ili pritiskom na tipku `Escape`.

```
%time
sage: factor(29230032746618058364074216492428094683663774517 41)
120892581961462917470 6189 * 241785163922925834941 2369
CPU time: 2.80 s,  Wall time: 2.94 s
```

Zadatak 3

Izračunajte funkciju $\pi(x)$ (broj prim-brojeva manjih od x) za $x = 10^{10}$ statističkom metodom: Izaberite uzorak od n slučajnih cijelih brojeva između 1 i x i testirajte koliko ima prim-brojeva među njima. Iz tog udjela odredite $\pi(x)$. Kolika veličina uzorka vam treba da bi relativna greška prema prime_pi(10^10) razumno često pala ispod 1%?

Zadatak 4

Kreirajte listu od 100 slučajnih brojeva iz intervala `[0, 10)`, a zatim u toj listi ostavite samo brojeve koji se za manje od `0.02` razlikuju od cijelog broja. *Naputak*: Testirajte `abs(x-round(x))`.

3.1.3 Ostali spremnici

Osim lista, postoje i drugi spremnici (*containers*) za objekte. Kao prvo tu su tzv *tuplovi* (*tuple*) koji "izvana" izgledaju isto kao liste samo u okruglim zagradama.

```
sage: tpl = (1, 3, 5, 7)
sage: tpl[3]
7
```

Glavna razlika prema listama je da su tuplovi nepromjenjivi. Nije moguće niti promijeniti neki njihov element niti im dodati nove:

```
sage: tpl[3] = 2
Traceback (most recent call last):
...
TypeError: 'tuple' object does not support item assignment
```

Razlika u upotrebi između tuplova i lista je da su tuplovi obično heterogeni strukturirani skupovi u kojima pojedina mjesta u tuplu nose različita značenja, dok su liste obično homogeni skupovi istovrsnih objekata. Npr, koordinate neke točke je prirodno staviti u tupl (x, y), a ne u listu [x,y], ali niz točaka je prirodno staviti u listu takvih tuplova [(x1, y1), (x2, y2), ...].

Kad varijablama pridružujemo vrijednosti iz kraćih lista ili tuplova zgodno je koristiti tehniku *raspakiravanja*

```
sage: x, y, z = (1, 2, 3)
sage: print "Norma vektora  = %f" % sqrt(x**2 + y**2 + z**2)
Norma vektora  = 3.741657
```

Raspakiravanje se često koristi kod procesiranja listi čiji su elementi liste ili tuplovi. U slijedećem primjeru pretvaramo listu 2D kartezijevih vektora u listu njihovih normi. Vidimo kako možemo kombinirati raspakiravanje po unutarnjoj s obuhvaćanjem 2D liste po vanjskoj dimenziji:

```
sage: [sqrt(x**2 + y**2) for x,y in [(1,2), (3,4), (5,6), (7,8)]]
[sqrt(5), 5, sqrt(61), sqrt(113)]
```

Zadatak 5

Koristeći raspakiravanje tuplova u kombinaciji s obuhvaćanjem liste pretvorite ovu listu vektora

$$\mathrm{tt} = [(r_1, \theta_1), (r_2, \theta_2), \ldots]$$

```
sage: tt = [(2.23, 1.11), (5.0, 0.93), (7.81, 0.88), (10.63, 0.85)]
```

iz 2D polarnog sustava u kartezijev:

$$[(x_1, y_1), (x_2, y_2), \ldots]$$

Nakon toga transformirajte nastalu listu kartezijevih vektora u listu kartezijevih vektora s cjelobrojnim koordinatama, zaokruživanjem vrijednosti x i y koordinata pomoću funkcije `round()`.

Daljnji spremnici koji nam stoje na raspolaganju su *skupovi* (*set*), koji su skupovi različitih(!) objekata i s kojima možemo raditi standardne stvari poput unije, presjeka, komplementa ...

```
sage: s1 = set([10, 9, 10, 8, 7, 7, 7, 4]); s1
{4, 7, 8, 9, 10}
```

(Uočite da se duplikati automatski izbacuju.)

```
sage: s1.union(t1)
{1, 3, 4, 5, 7, 8, 9, 10, 11, 13}
sage: s1.intersection(t1)
{7, 9}
sage: s1.difference(t1)
{4, 8, 10}
```

Zadnji, vrlo važan, spremnik kojeg ćemo spomenuti je *rječnik* (engl. *dictionary*). Riječ je o preslikavanju `key` -> `value`, gdje je `value` bilo koji objekt, a `key` može biti tipično broj ili string (premda su i druge stvari dopustive kao key, npr tuplovi).

```
sage: d1 = {'a':1, 'c':3, 'b':2}; d1
{'a': 1, 'b': 2, 'c': 3}
sage: d2 = {1: sin, 2: cos, 3: tan}; d2
{1: sin, 2: cos, 3: tan}
sage: d3 = {'ime': 'pero', 'prezime': 'peric', 'spol': 'M'}; d3
{'ime': 'pero', 'prezime': 'peric', 'spol': 'M'}
```

Redosljed elemenata u rječniku nema značenja. Pristup pojedinim elementima rječnika je moguć "indeksiranjem" pomoću ključa:

```
sage: d1['c']
3
```

Na isti način je moguće dodavati nove elemente u rječnik:

```
sage: d2[5] = log
```

```
sage: d2.keys()
[1, 2, 3, 5]
```

Iteriranje po rječniku ne daje elemente već samo ključeve:

```
sage: [it for it in d2]
[1, 2, 3, 5]
```

```
sage: [d2[n](1.) for n in d2.keys()]
[0.841470984807897, 0.540302305868140, 1.55740772465490, 0.000000000000000]
```

Zadatak 6

Izračunajte (iteriranjem po rječniku, a ne ručno!) prosjek godina ljudi iz slijedećeg rječnika (rezultat treba biti decimalni broj):

```
sage: ages = {'john' : 74, 'paul': 71, 'george': 71, 'ringo': 73}
```

Za kraj, i *stringove* možemo interpretirati kao liste znakova i tretirati ih kao takve

```
sage: s1 = 'Samobor'
sage: s1[-3:]
'bor'
```

3.1.4 NumPy polja

Obične liste se rabe za razne namjene, uključujući simboličke račune, ali za numeriku su pogodnija tzv. NumPy polja. NumPy (*Numerical Python*) je modul za python namjenjen baratanju s multidimenzionalnim brojčanim poljima kakva se često sreću u svim područjima znanosti. Riječ je o velikom modulu koji nije automatski prisutan u Pythonu ili Sageu, već ga treba učitati pomoću naredbe `import`

```
sage: import numpy as np
```

Ovdje se istovremeno s učitavanjem uvodi skraćeno ime `np`. Svi objekti i funkcije NumPy modula sada su dostupni kao `np.<ime objekta>`.

Numpy polja su u računalu zapisana u neprekinutom slijedu memorijskih lokacija što omogućuje brži pristup no zbog toga svi elementi polja moraju biti istog tipa (`integer`, `float`, `complex` ...).

```
sage: a = np.array(range(1,11)); a   # konverzija obične liste u NumPy polje
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

Vidimo da je NumPy polje ispisano na drugačiji način nego obična lista, ali za razlikovanje tih dviju objekata izgled ispisa nije pouzdani kriterij. Neke operacije, poput `print`, ispisuju NumPy polje da izgleda kao obična lista

```
sage:: print a
[1 2 3 4 5 6 7 8 9 10]
```

Ako postoji dvojba, možemo ispitati tip objekta komandom `type()`

```
sage: type(a)
<type 'numpy.ndarray'>
```

Pri konstrukciji NumPy polja izbor tipa objekata je automatski, ali moguće ga je i odrediti opcionalnim argumentom dtype:

```
sage: b = np.array(range(1,11), dtype=np.float64); print b
[  1.   2.   3.   4.   5.   6.   7.   8.   9.  10.]
```

Tip objekata u NumPy polju je zapisan u `dtype` *atributu* polja [1].

```
sage: a.dtype
dtype('int64')
sage: b.dtype
dtype('float64')
```

Numpy polja imaju dosta više metoda od običnih lista ...

```
sage: dir(a)[-66:]
['all', 'any', 'argmax', 'argmin', 'argsort', 'astype', 'base', 'byteswap',
'choose', 'clip', 'compress', 'conj', 'conjugate', 'copy', 'ctypes',
'cumprod', 'cumsum', 'data', 'diagonal', 'dtype', 'dump', 'dumps', 'fill',
'flags', 'flat', 'flatten', 'getfield', 'imag', 'item', 'itemset',
'itemsize', 'max', 'mean', 'min', 'nbytes', 'ndim', 'newbyteorder',
'nonzero', 'prod', 'ptp', 'put', 'ravel', 'real', 'repeat', 'reshape',
'resize', 'round', 'searchsorted', 'setfield', 'setflags', 'shape', 'size',
'sort', 'squeeze', 'std', 'strides', 'sum', 'swapaxes', 'take', 'tofile',
'tolist', 'tostring', 'trace', 'transpose', 'var', 'view']
```

... no zbog efikasnosti implementacije nema upravo onih metoda koje imaju obične liste. To je zato jer je npr. umetanje elementa u listu vrlo "skupa" operacija koja uključuje brisanje i pisanje svih elemenata koji u memoriji dolaze nakon tog mjesta umetanja. Ako želimo raditi takve stvari upotreba NumPy polja nam ionako neće donijeti nikakve prednosti i treba koristiti obične liste. (Za konverziju NumPy liste u običnu koristimo funkciju `list()`.)

Jedno od vrlo korisnih svojstava NumPy lista je da se većina matematičkih operacija prirodno distribuira po elementima liste (obične liste nemaju to svojstvo):

```
sage: xs = np.linspace(1,10,3); xs
array([  1. ,   5.5,  10. ])
sage: xs + 1
array([  2. ,   6.5,  11. ])
sage: sin(xs)
array([ 0.84147098, -0.70554033, -0.54402111])
```

NumPy liste mogu biti i višedimenzionalne, no i one su u memoriji računala zapisane u jednodimenzionalnom (1D) slijedu memorijskih adresa. Stoga je preoblikovanje elemenata 1D NumPy u 2D polje proizvoljnog oblika računalno "jeftina" operacija, koja može biti od koristi

```
sage: b=a.reshape(2,5); print b
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]]


sage: b.shape  # "oblik" polja - broj redaka i stupaca
(2, 5)
```

[1] Atributi objekta su sve ono čemu se pristupa operatorom točkice `.`. Ranije spomenute *metode* su atributi koji se mogu pozivati kao funkcije. `dtype` je atribut koji je naprosto tip elementa polja. Za bolje razumijevanje svega ovog dobro je pročitati nešto o objektno-orijentiranom programiranju u Pythonu.

Pristupanje pojedinom elementu višedimenzionalnog polja je moguće očitim indeksiranjem a[i][j]..., ali je efikasnije koristiti skraćeno indeksiranje: a[i, j, ...]:

```
sage: b[1,1] = 42.


sage: print b
[[ 1  2  3  4  5]
 [ 6 42  8  9 10]]
```

Pažnja: radi efikasnosti, preoblikovanja i rezovi kroz NumPy polja ne kopiraju originalne elemente već samo manipuliraju pokazivačima na ista mjesta. Tako se npr. ova netom načinjena zamjena u listi `b` odražava i u listi `a` čijim preoblikovanjem je lista `b` nastala:

```
sage: print a
[ 1  2  3  4  5  6 42  8  9 10]
```

Obične liste nemaju takvo ponašanje:

```
sage: t5 = t1[2:4]; t5    # t1 je obična lista
[5, 7]


sage: t5[0] = 'novi'; print t5
['novi', 7]
sage: t1
[1, 3, 5, 7, 9, 11, 13]
```

Inače, rezovi kroz dvodimenzionalne NumPy liste su elegantan način ekstrakcije redaka ili stupaca:

```
sage: print b[:,1]
[ 2  42]
```

Ukoliko trebamo liste realnih (*float*) brojeva, možemo koristiti NumPy inačicu funkcije `range()` (sintaksa je ista kao za `range()`)

```
sage: np.arange(2., 9.9, 1.1)
array([ 2. ,  3.1,  4.2,  5.3,  6.4,  7.5,  8.6,  9.7])
```

Funkcija `np.range()` je pogodna kad želimo specificirati korak liste. Kad želimo specificirati broj elemenata liste koristimo funkciju `np.linspace()`.

```
sage: np.linspace(0, 10, 5)
array([  0. ,   2.5,   5. ,   7.5,  10. ])
```

Za daljnje detalje o NumPy poljima, vidi NumPy Tutorial.

Zadatak 7

Za spajanje dvije jednako duge 1D liste u 2D listu (zapravo listu tuplova) koristimo naredbu `zip`:

```
sage: xs = [11, 12, 13, 14, 15]
sage: ys = [-1, -2, -3, -4, -5]
sage: points = zip(xs, ys); points
[(11, -1), (12, -2), (13, -3), (14, -4), (15, -5)]
```

Rastavite listu `points` natrag na originalne dvije 1D liste i to dvjema različitim metodama:

1. Dva obuhvaćanja liste `points` (ova metoda nema veze s NumPy-em)
2. Dva reza kroz listu `points`, koju prvo pretvorite u NumPy polje pomoću `np.array`.

3.2 Kontrola toka izvršavanja

3.2.1 Grananje if-then

Sintaksa je

```
if <condition>:
    <block 1>
elif <condition>:
    <block 2>
else:
    <block 3>
```

Treba uočiti dvotočke prije blokova koda koji se uvjetno izvršavanju. Također je važno paziti na konzistentno uvlačenje blokova koda (tamo gdje bi u C-u bile vitičaste zagrade), jer je to jedina vodilja kompajleru da prepozna gdje su granice bloka. (Tradicionalno se uvlači za 4 mjesta. Editor u Sage ćelijama automatski radi to uvlačenje.) Npr:

```
sage: if 2>3:
....:     print "veći je"
....: elif 2==3:
....:     print "jednak je"
....: else:
....:     print "manji je"
manji je
```

U uvjetima se mogu koristiti standardni logički operatori `and`, `or`, `not`, ..., a važno je uočiti da se test usporedbe jednakosti radi s dvostrukim znakom jednakosti ==. (Jednostruka jednakost = je rezervirana za operacije pridruživanja poput `x=1`.) Operatori uspoređivanja su standardni: `==`, `!=`, `<`, `<=`, `>`, `>=`.

```
sage: if 2>3 or 2<3:
....:     print "nisu isti"
nisu isti
```

3.2.2 Petlje

`while` petlja ima standardni oblik

```
while <condition>:
    <body>
```

gdje je `condition` logički uvjet. Npr.

```
sage: k = 1
sage: while k < 10:
....:     print k,
....:     k += 1
1 2 3 4 5 6 7 8 9
```

(Zarez nakon `print` naredbe sprečava automatski prelazak u novi red.)

`For` petlja ima specifičnu sintaksu

```
for x in <iterable>:
    <body>
```

gdje je <iterable> lista, tuple, skup, rječnik, ... Za prijevremeno iskakanje iz petlje postoji komanda `break`. Npr. slijedeći algoritam pronalazi prvi cijeli broj čiji rastav sadrži više od pet različitih prostih faktora.

```
sage: for i in range(1,1e5):
....:     if len(factor(i))>5:
....:         break
sage: print i
30030
```

Alternativni i nešto elegantniji algoritam bi bio

```
sage: i = 1
sage: while len(factor(i))<6:
....:     i +=1
sage: print i
30030
```

Zadatak 1

Isprogramirajte petlju koja će pomnožiti sve brojeve od 1 do 7 (dakle, izračunati će 7!). Riješite zadatak jednom koristeći `for` petlju, a drugi put `while` petlju.

3.3 Funkcije

Mogućnost definiranja novih funkcija je osnovna stepenica k naprednijem programiranju. Trebati ćemo razlikovati dvije vrste funkcija:

1. Simboličke funkcije
2. Python funkcije

Ugrubo, simbolička funkcija dopušta više simboličkih manipulacija (poput integracije ili deriviranja),

ali ne može sadržavati kompleksne algoritme već samo jednostavne izraze [2]. Python funkcija može biti proizvoljno kompleksna, ali ne može se uvijek npr. simbolički integrirati.

3.3.1 Simboličke funkcije

Simboličke funkcije se definiraju na slijedeći prirodan način:

```
sage: f(x) = x^2
sage: f
x |--> x^2
```

S njima se mogu raditi sve uobičajene operacije:

```
sage: f(2)
4
sage: f((2*x+3)^2)
(2*x + 3)^4
sage: diff(f(x), x)  # simboličko diferenciranje, vidi slijedeće poglavlje
2*x
```

```
sage: type(sin)
<class 'sage.functions.trig.Function_sin'>
sage: type(f)
<type 'sage.symbolic.expression.Expression'>
```

Naravno, funkcija može biti i funkcija od više variabli:

```
sage: g(x, a) = x^a
sage: g(4, 2)
16
```

3.3.2 Python funkcije

Python funkcije se definiraju korištenjem ključnih riječi `def` i `return`, te blokova kôda koji su konzistentno uvučeni

```
sage: def h(x):
....:     "Kvadriraj broj x."
....:     return x^2
```

```
sage: print h(3)
9
```

Kao prvi red tijela funkcije može se, kao gore, staviti dokumentacijski string (tzv. *docstring*) kojem se kasnije može pristupiti standardnim metodama pristupa dokumentaciji. (Dakle, `h?` će ispisati dokumentacijski string.)

Funkcija može imati i opcionalne argumente s defaultnom vrijednošću:

[2] Formalno, simboličke funkcije nisu po svom tipu "funkcije", već "simbolički izrazi koji se mogu pozivati" ("*callable symbolic expression*"): Za detalje razlika između simboličkih i Python funkcija vidi ovdje)

```
sage: def fun(x, n=1, b=0):
....:     return x^n + b

sage: fun(3, 4, 5)
86
sage: fun(3, b=5, n=4)
86
sage: fun(3)
3
```

(Uočite da kad smo eksplicitno imenovali argumente nismo morali paziti na njihov poredak.)

Bilo što može biti argument funkcije. Najmoćnija stvar, obilato korištena u funkcionalnom pristupu programiranju, je da i same funkcije mogu biti argumenti funkcija:

```
sage: def gun(f, x):
....:     "Komponiraj dvaput funkciju sa samom sobom"
....:     return f(f(x))

sage: print gun(sin, 2)
sin(sin(2))
sage: gun(log, 0.1)
0.834032445247956 + 3.14159265358979*I
```

Zadatak 1

Isprogramirajte funkciju `fibProc(n)` koja računa n-ti član Fibonaccijevog niza (niz kod kojeg je svaki član definiran kao zbroj prethodna dva, a javlja se pri analizi idealizirane populacije zečeva).

Zadatak 2

Isprogramirajte funkciju `fibBinet(n)` koja računa n-ti član Fibonaccijevog niza putem Binetove formule

$$F_n = \frac{\varphi^n - \cos(n\pi)\varphi^{-n}}{\sqrt{5}}$$

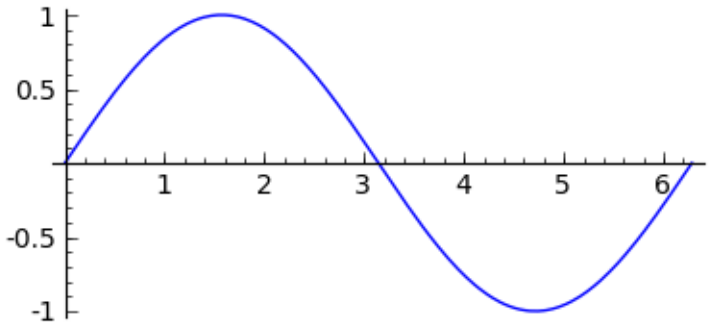
gdje je φ tzv. zlatni omjer (`golden_ratio`). Uvjerite se da dobivate dobre vrijednosti za neke `n`.

3.4 Crtanje grafova

3.4.1 2D grafovi

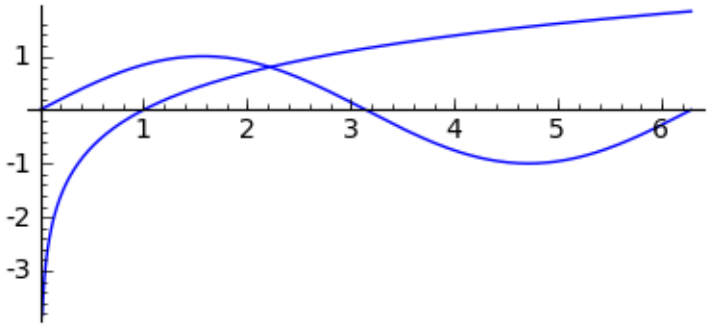
Glavna naredba za crtanje 2D grafova je `plot()`

```
sage: plot(sin(x), (x,0,2*pi), figsize=[4,2])
```



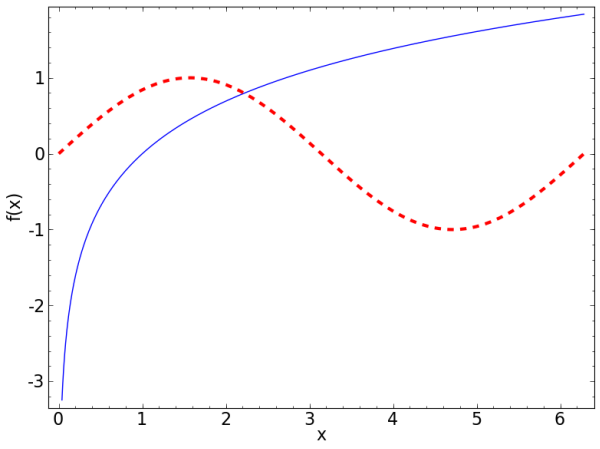
Za kombiniranje plotova može se kao prvi argument staviti lista funkcija ...

```
sage: plot([sin(x), log(x)], (x,0,2*pi), figsize=[4,2])
```



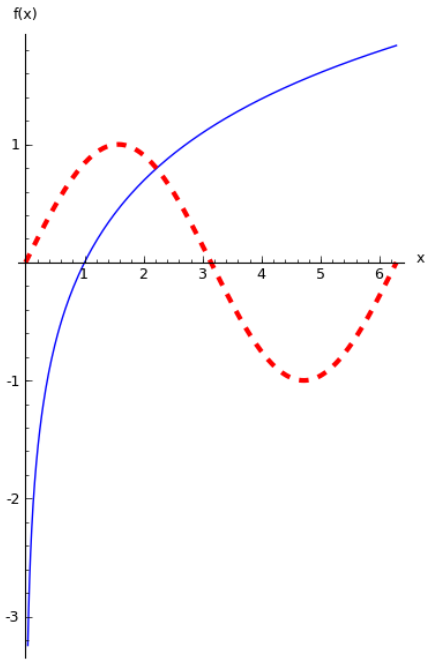
... ali je bolje naprosto zbrajati grafove pojedinih funkcija, što nam onda daje kontrolu nad svojstvima pojedinih linija (boja, debljina, ...).

```
sage: P1 = plot(log(x), (x,0,2*pi))
sage: P2 = plot(sin(x), (x,0,2*pi), color='red', linestyle='--',
....:           thickness=3)
sage: (P1+P2).show(axes_labels=['x', 'f(x)'], frame=True, axes=False,
....:           fontsize=16)
```



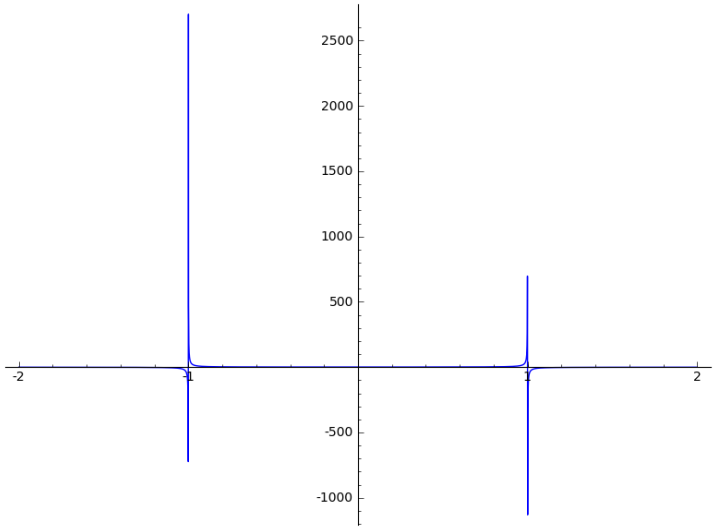
Premda to nije obavezno, gore je bilo logično svojstva pojedinih linija stavljati kao argumente funkcija `plot()`, a svojstva grafa kao cjeline u završni `show()`. Inače, korištenjem metode `show()` moguće je jednostavno ponovno iscrtavanje grafa uz promjenu nekih opcija:

```
sage: P.show(frame=False, axes=True, aspect_ratio=2, fontsize=9)
```



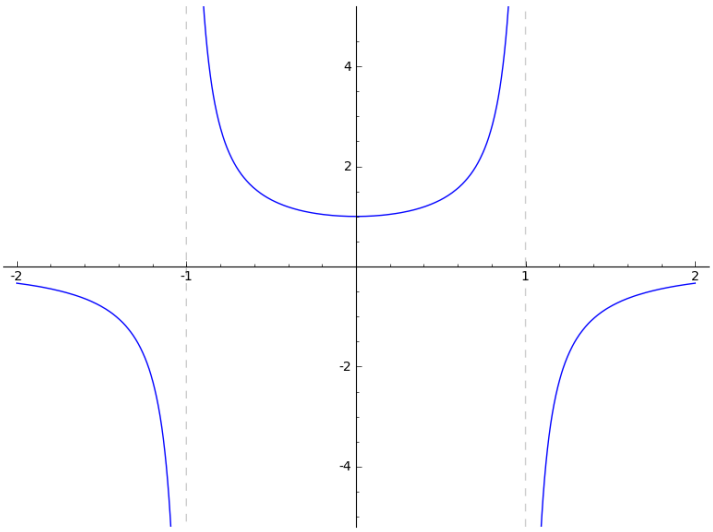
Funkcija `plot()` sama određuje raspon vrijednosti ordinate pogodan za crtanje zadanih funkcija. Međutim, ukoliko funkcija ima singularitet u području crtanja, to ispada loše:

```
sage: plot(1/(1-x^2), (x,-2,2))
```



Tada moramo eksplicitno ograničiti ordinatu opcijama ymin and ymax. (A može se i lijepo označiti položaj polova opcijom `detect_poles`.)

```
sage: plot(1/(1-x^2), (x,-2,2), detect_poles='show', ymin=-5, ymax=5)
```



Grafove često želimo upotrijebiti i izvan Sage radnog lista, npr. u nekom članku ili prezentaciji. Eksportiranje grafova i drugih objekata postiže se uporabom funkcije `save()`. Format grafičke datoteke određen je ekstenzijom. Dopuštene ekstenzije su `.png, .ps, .eps, .svg, and .sobj`.

```
sage: P.save('/tmp/graf.png')
```

Zadatak 1

Pronađite ovu datoteku na disku i prikažite je pomoću nekog programa za pregledavanje slika. (Na Windowsima je potrebno pristupiti virtualnoj Linux mašini putem ssh protokola (winscp, putty, ...) spajanjem na adresu na koju se spaja i Windows WWW browser, uz user=sage, pass=sage. Ako to nije moguće naprosto spremite sliku klikom desnim gumbom miša na nju, pa "Save Image As ...".)

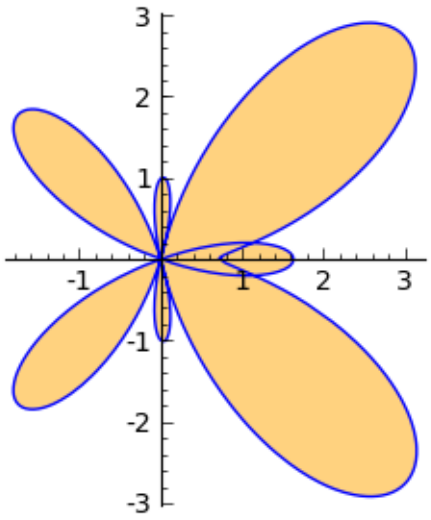
Zadatak 2

Nacrtajte graf funkcije log(x) između x=0.5 i x=1.5 bez ikakvih oznaka, okvira i osi, dakle samo liniju.

Osim funkcija eksplicitno zadanih u obliku $y = y(x)$ možemo crtati i funkcije zadane parametarski u obliku $y = y(t)$, $x = x(t)$. Npr:

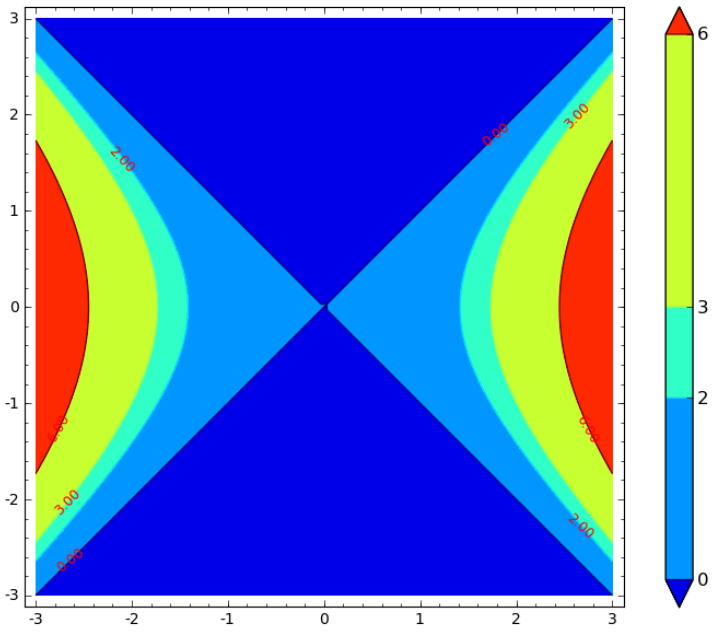
```
sage: var('t')
t
sage: ff = (exp(cos(t)) - 2*cos(4*t) + sin(t/12)^5)


sage: parametric_plot((ff*cos(t), ff*sin(t)),
....:       (t, 0, 2*pi), fill=True, fillcolor='orange', figsize=[3,3])
```



Za crtanje raznih potencijala i srodnih funkcija, zgodna je funkcija `contour_plot()`:

```
sage: var('x y')
sage: contour_plot(x^2-y^2, (x,-3,3), (y,-3,3), labels=True,
....:     label_colors='red',contours=[0,2,3,6], cmap='jet', colorbar=True)
```

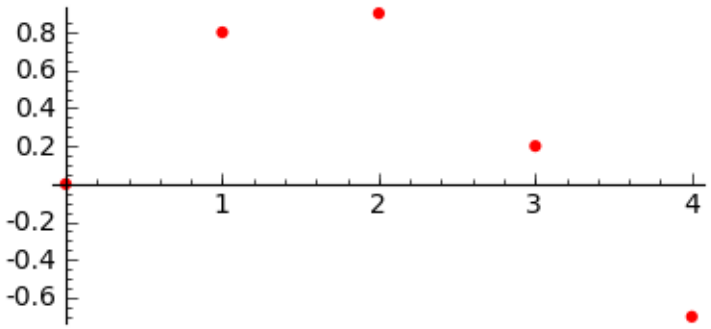


Zadatak 3

Koristeći funkciju `parametric_plot()` i trigonometrijske funkcije nacrtajte kružnicu.

Za crtanje podataka organiziranih u listu parova $[[x_1, y_1], [x_2, y_2], \ldots]$ rabimo `list_plot()`:

```
sage: data = [[0,0], [1,0.8], [2, 0.9], [3, 0.2],[4, -0.7]]
sage: list_plot(data, pointsize=20, color='red', figsize=[4,2])
```



Zadatak 4

Uočite da se funkcija `fibBinet(x)` iz *prošlog odjeljka* može izvrijedniti i za vrijednosti argumenta `x` koje nisu cjelobrojne. Nacrtajte graf te funkcije za `-4 < x < 7` i superponirajte na njega (u drugoj boji) točke koje odgovaraju vrijednostima funkcije za pozitivne cjelobrojne argumente `[(1, F_1), (2, F_2), ... (7, F_7)]`. *Naputak*: `plot(...) + list_plot(...)`.

Zadatak 5

Koristeći funkciju `parametric_plot()` i trigonometrijske funkcije nacrtajte kružnicu.

3.4.2 3D grafovi

Za crtanje 3D grafova stoje na raspolaganju dvije softverske biblioteke:

- **JMOL** (default) traži da rade Java appleti u browseru (na Linuxu samo Sun Java funkcionira - mogući problemi u F-26).
- **Tachyon** - raytracing paket, radi bez Jave, nije moguć interaktivni rad s crtežom (okretanje mišem)

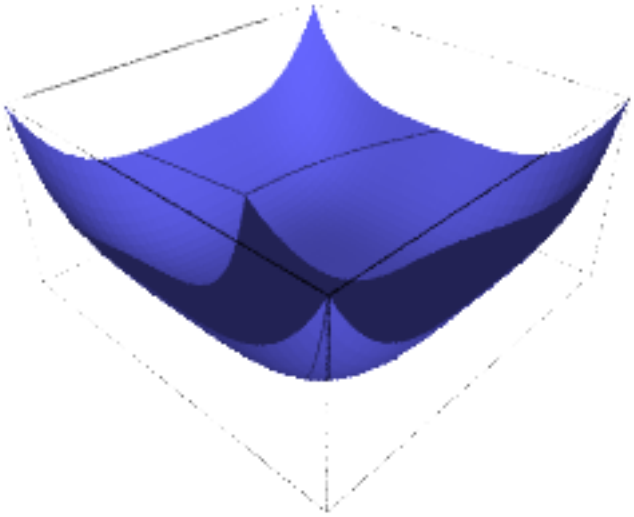
```
sage: y = var('y')

sage: P2 = plot3d(x^4+y^4, (x, -2, 2), (y, -2, 2)); P2.show()
```

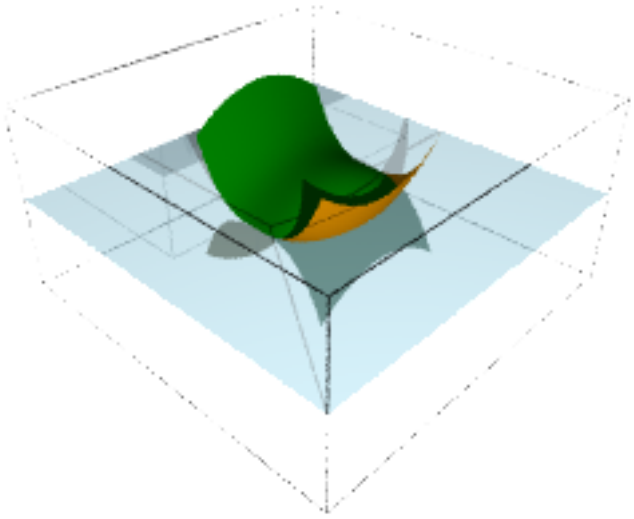
Klik mišem na sliku omogućuje mijenjanje smjera gledanja i zumiranje kotačićem miša. Desni klik otvara izbornik. Za "pravi 3D doživljaj" stavite dvobojne naočale pa onda desni klik -> Style -> Stereographic ->...

Ukoliko nemate mogućnost prikazivanja Java appleta, koristite 'tachyon':

```
sage: P2.show(viewer='tachyon', figsize=[3,3])
```



```
sage: L = plot3d(lambda x,y: 0, (-5,5), (-5,5), color="lightblue", opacity=0.8)
sage: P = plot3d(lambda x,y: 4 - x^3 - y^2, (-2,2), (-2,2), color='green')
sage: Q = plot3d(lambda x,y: x^3 + y^2 - 4, (-2,2), (-2,2), color='orange')
sage: (L + P + Q).show(viewer='tachyon', figsize=[3,3])
```



3.4.3 Matplotlib biblioteka

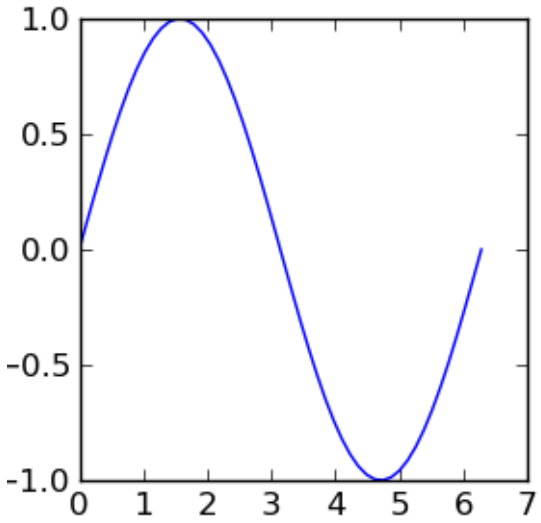
Dosad rečeno je dovoljno za osnovno skiciranje. Funkcija `plot()` i ostale navedene funkcije za 2D grafove su zapravo samo sučelje za tzv. matplotlib grafičku biblioteku. Za precizniju kontrolu nad crtanjem potrebno je izravno pozivati funkcije ove biblioteke.

Matplotlib zapravo ne crta funkcije tj. linije već samo liste točaka (slično kao list_plot() gore), pa se crtanje linija dobiva iscrtavanjem i spajanjem gustih skupova točaka. Za kreiranje ovih lista točaka možemo koristiti NumPy funkcije `linspace()` i `logspace()`. koristiti.

```
sage: import matplotlib.pyplot as plt
sage: import numpy as np
```

Elementarni primjer

```
sage: fig, ax = plt.subplots()
sage: xs = np.linspace(0, 2*np.pi)
sage: ax.plot(xs, sin(xs))
sage: fig.savefig('fig1')
```



Par komentara:

- Za komunikaciju s Matplotlib bibliotekom koristimo modul `pyplot` (kojeg smo preimenovali u `plt`) koji omogućuje nešto pristupačnije sučelje [3].
- Kao granicu koristimo numeričku vrijednost za π iz NumPy biblioteke.
- Funkcija `plot.subplots` stvara dva objekta: sliku (*figure*) i panel (*axis*). Složene slike mogu imati više panela, vidi primjer dolje.
- U čistom Pythonu, komanda za prikazivanje slike bila bi `fig.show()`, ali u Sageu je za prikaz potrebno kreirati datoteku sa `fig.savefig()` koju onda Sage automatski prikaže u radnom listu [4].

[3] Postoji i modul `pylab` sa sučeljem sasvim sličnim komercijalnom programu Matlab. Vidi usporedbu raznih Matplotlib sučelja.

[4] Primjere crtanja iz originalne matplotlib dokumentacije moguće je izravno kopirati u Sage, do na tu zamjenu `show()` sa `savefig('fig')`. Usput rečeno, `savefig()` sprema datoteku sa slikom u `\$HOME/.sage/sage_notebook.sagenb/home/<username>/<kk>/cells/<nn>/imeslike.png`, gdje je `<kk>` broj radnog lista vidljiv u URL-u ("WWW" adresi vidljivoj u WWW pregledniku), a `<nn>` je broj ćelije koji je vidljiv

- Najvažnije: sintaksa Matplolibove funkcije `plot` traži kao prve argumente *posebno* listu x-koordinata, i *posebno* listu y-koordinata:

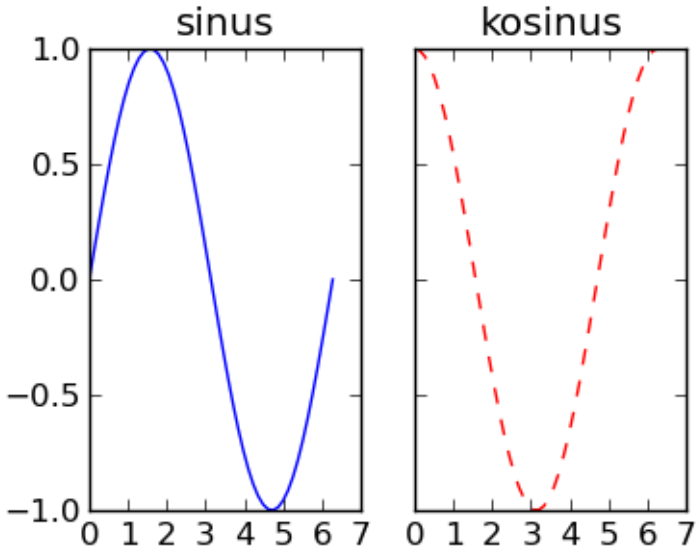
  ```
  ax.plot([x1, x2, ...], [y1, y2, ...], <opcionalni argumenti>)
  ```

  To je različito od srodne Sageove funkcije `list_plot` koja traži kao jedini argument listu parova koordinata:

  ```
  list_plot([[x1, y1], [x2, y2], ...], <opcionalni argumenti>)
  ```

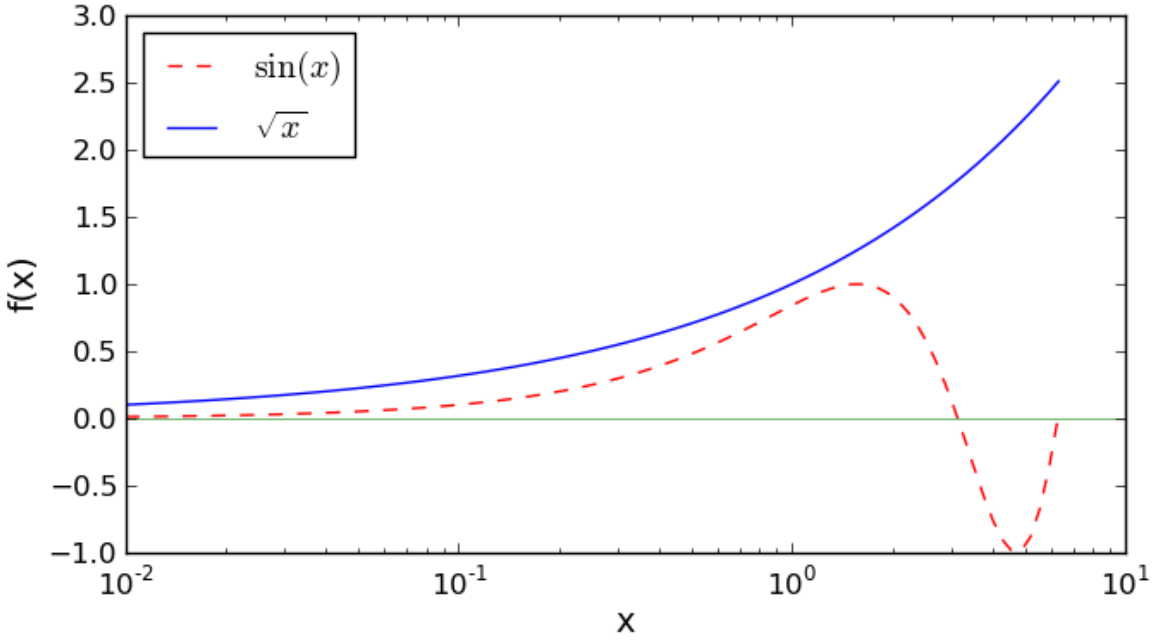
  Za pretvorbu jedne vrste liste u druge, vidi *zadatak* na kraju odjeljka o *NumPy listama*.

```
sage: fig, (ax1, ax2) = plt.subplots(1, 2, sharey=True, figsize=[4,3])
sage: ax1.plot(xs, sin(xs))
sage: ax1.set_title('sinus')
sage: ax2.plot(xs, cos(xs), color='red', linestyle='--')
sage: ax2.set_title('kosinus')
sage: fig.savefig('fig1')
```




```
sage: xs = np.logspace(-2.0, 0.8, 100)  # granice su log_10(x)
sage: fig, ax = plt.subplots(figsize=[7,4])   # specifikacija dimenzija
sage: ax.plot(xs, sin(xs), color='red', linestyle='--',
....:                              label='$\sin(x)$')  # LaTeX oznake
sage: ax.plot(xs, sqrt(xs),
....:   'b-', label='$\sqrt{x}$')  # skraćene oznake za boju i tip linije
sage: ax.set_xscale('log')
sage: ax.axhline(0, color='g', linewidth=1)  # horizontalna linija na y=0
sage: ax.set_xlabel('x', fontsize=14)
sage: ax.set_ylabel('f(x)', fontsize=14)
sage: ax.legend(loc="upper left")
sage: fig.tight_layout()   # inače se ne vide cijele oznake na osima
sage: fig.savefig('test')
```

samo u tekstualnoj varijanti radnog lista (tipke "Text" ili "Edit" u notebooku), ali može se saznati i pomoću Unix shell komande `find . -name "ime.png"` iz direktorija `...<kk>/cells`.



Zadatak 6

Nacrtajte (Matplotlibom) 100 slučajno raspoređenih točaka (hint: `np.random.rand()`, te `ax.plot(..., linestyle='None', marker='o')`) gdje su x i y koordinate tih točaka u intervalu (-1,1), a nacrtane su na dijagramu s x i y osima koje se protežu u intervalu (-2, 2). (hint: `ax.set_xlim()` i `ax.set_ylim()`).

Zadatak 7

Odredite (x,y) koordinate vrha lijevog "krila" gornjeg "leptira", dakle točke, negdje oko (x=3, y=3) koja je najudaljenija od ishodišta.

Zadatak 8

Nacrtajte kružnicu u kompleksnoj ravnini zadanu kompleksnim brojevima

$$\{z = \rho e^{i\eta}(1 + e^{i\phi}) - 1 \quad | \quad \phi \in [0, 2\pi), \rho = 1.3, \eta = 0.2\}$$

te krivulju koja se dobije kad se ova kružnica podvrgne transformaciji Žukovskog:

$$z \to w = \frac{1}{2}\left(z + \frac{1}{z}\right)$$

POGLAVLJE 4

Matematika

4.1 Simbolički izrazi

Moć paketa za simboličku matematiku, poput Sagea, leži u sposobnosti manipulacije simboličkim izrazima. Kao prvo, potrebno je na slijedeći način deklarirati varijable koje namjeravamo koristiti u simboličkim izrazima [1]

```
sage: var('a b c x y z t')
(a, b, c, x, y, z, t)
```

Pomoću ovih varijabli sad izgrađujemo simboličke izraze:

```
sage: i1 = (b+a)^3; i1
(a + b)^3
```

Sage ne provodi skoro nikakve operacije na izrazima dok to eksplicitno ne zatražimo. Recimo da želimo razviti gornji izraz koristeći binomni teorem. Za to služi funkcija `expand()`

```
sage: expand(i1)
a^3 + 3*a^2*b + 3*a*b^2 + b^3
```

Funkcija `expand()`, kao i mnoge druge, se može alternativno upotrijebiti i kao *metoda* simboličkog izraza.

```
sage: i1.expand()
a^3 + 3*a^2*b + 3*a*b^2 + b^3
```

U prvom pristupu `expand` doživljavamo kao funkciju ili operaciju, dok je izraz `i1` njen argument odnosno operand. To je način razmišljanja svojstven standardnom *proceduralnom* ili pak tzv. *funkcionalnom programiranju*.

U drugom pristupu izraz `i1` treba pak doživljavati kao *objekt* , u smislu tzv. *objektno-orijentiranog* (OO) programiranja, a `expand()` je tzv. *metoda* što je naziv za funkciju koja je pridružena tipu objekta na koji djeluje [2].

[1] Eksplicitno deklariranje simboličkih varijabli je moguće izbjeći pozivanjem funkcije `automatic_names(True)`, ali to nećemo koristiti.

[2] To onda omogućuje da metode istog imena rade različite stvari s različitim objektima (tzv. *polimorfizam*). Kako je python OO jezik, takva sintaksa se obilato koristi u Sage-u i brojne funkcije se ni ne mogu koristiti na prvi način. Jedna od prednosti takvog pristupa je da elegantno možemo saznati popis svih funkcija koje rade nešto smisleno sa zadanim objektom, i to tako da nakon što stavimo točkicu "." poslije objekta stisnemo TAB tipku. Dobit ćemo popis svih metoda tog objekta. Ovo međutim ne funkcionira s netom upisanim izrazom u trenutnoj ćeliji, već samo s ranije definiranim izrazima (objektima) kojima smo

Da bi saznali što pojedina metoda radi, upišemo je nakon odgovarajućeg objekta i operatora točkice `.`, dodamo upitnik i stisnemo TAB. Pri upotrebi metode ne smije se zaboraviti na zagrade, koje su često prazne, ali nekad sadrže opcionalne argumente kojima modificiramo ponašanje metode. Ukoliko zaboravimo zagrade Sage ne poziva funkciju već samo ispisuje njeno puno ime poput "`<metoda expand pridružena objektu 'Expression' koji je pohranjen na toj i toj adresi>`".

```
sage: i1.expand
<built-in method expand of sage.symbolic.expression.Expression object
                                                                at 0x4e88200>
```

Tek zagrade daju zahtjev interpreteru da dotičnu metodu i pozove tj. izvrši.

Naravno ovakve jednostavne izraze možemo razviti i na ruke, dok računalo blista kad radi s velikim izrazima (sve dok stanu u memoriju računala)

```
sage: i2 = (a +2*b + 3*c)^3 * (x+y)^3
sage: i2.expand()
a^3*x^3 + 6*a^2*b*x^3 + 12*a*b^2*x^3 + 8*b^3*x^3 + 9*a^2*c*x^3 +
36*a*b*c*x^3 + 36*b^2*c*x^3 + 27*a*c^2*x^3 + 54*b*c^2*x^3 + 27*c^3*x^3 +
3*a^3*x^2*y + 18*a^2*b*x^2*y + 36*a*b^2*x^2*y + 24*b^3*x^2*y +
27*a^2*c*x^2*y + 108*a*b*c*x^2*y + 108*b^2*c*x^2*y + 81*a*c^2*x^2*y +
162*b*c^2*x^2*y + 81*c^3*x^2*y + 3*a^3*x*y^2 + 18*a^2*b*x*y^2 +
36*a*b^2*x*y^2 + 24*b^3*x*y^2 + 27*a^2*c*x*y^2 + 108*a*b*c*x*y^2 +
108*b^2*c*x*y^2 + 81*a*c^2*x*y^2 + 162*b*c^2*x*y^2 + 81*c^3*x*y^2 +
a^3*y^3 + 6*a^2*b*y^3 + 12*a*b^2*y^3 + 8*b^3*y^3 + 9*a^2*c*y^3 +
36*a*b*c*y^3 + 36*b^2*c*y^3 + 27*a*c^2*y^3 + 54*b*c^2*y^3 + 27*c^3*y^3
```

Potenciranjem i razvijanjem gornjeg izraza dobivamo izraz od 550 članova ...

```
sage: i3 = expand(i2^3)
sage: len(i3)
550
```

... kojeg Sage s lakoćom faktorizira:

```
sage: factor(i3)
(a + 2*b + 3*c)^9*(x + y)^9
```

Često je korisno izraz organizirati kao polinom u nekoj varijabli. Za to služi funkcija `collect()`:

```
sage: i4=i2.expand().collect(y)
sage: i4
a^3*x^3 + 6*a^2*b*x^3 + 12*a*b^2*x^3 + 8*b^3*x^3 + 9*a^2*c*x^3 +
36*a*b*c*x^3 + 36*b^2*c*x^3 + 27*a*c^2*x^3 + 54*b*c^2*x^3 + 27*c^3*x^3 +
(a^3 + 6*a^2*b + 12*a*b^2 + 8*b^3 + 9*a^2*c + 36*a*b*c + 36*b^2*c +
27*a*c^2 + 54*b*c^2 + 27*c^3)*y^3 + 3*(a^3*x + 6*a^2*b*x + 12*a*b^2*x +
8*b^3*x + 9*a^2*c*x + 36*a*b*c*x + 36*b^2*c*x + 27*a*c^2*x + 54*b*c^2*x
+ 27*c^3*x)*y^2 + 3*(a^3*x^2 + 6*a^2*b*x^2 + 12*a*b^2*x^2 + 8*b^3*x^2 +
9*a^2*c*x^2 + 36*a*b*c*x^2 + 36*b^2*c*x^2 + 27*a*c^2*x^2 + 54*b*c^2*x^2
+ 27*c^3*x^2)*y
```

pridjelili ime. Pridjeljivanje imena objektima se izvodi znakom jednakosti i korisno je ne samo zbog navedenog razloga već i inače radi lakšeg baratanja izrazima i kasnijeg referiranja na iste.

```
sage: len(i4)
13
```

(Uočite da `collect()` ne pojednostavljuje koeficijente [3].) Za dobiti koeficijent uz neku potenciju neke varijable koristi se funkcija `coefficient()`. Npr, koeficijent uz a^9 jest

```
sage: i3.coefficient(a, 9)
x^9 + 9*x^8*y + 36*x^7*y^2 + 84*x^6*y^3 + 126*x^5*y^4 + 126*x^4*y^5 +
84*x^3*y^6 + 36*x^2*y^7 + 9*x*y^8 + y^9
```

Najsveobuhvatnija funkcija za pojednostavljivanje simboličkih izraza je `simplify_full()`:

```
sage: i5 = a/(1-a) + a/(1+a); i5
a/(a + 1) - a/(a - 1)


sage: i5.simplify_full()
-2*a/(a^2 - 1)
```

Funkcija `simplify_full()` je kompozicija elementarnijih funkcija za pojednostavljivanje izraza. Jedna od tih elementarnijih funkcija je `simplify_trig()` koja pri pojednostavljivanju rabi samo trigonometrijske identitete.

```
sage: (sin(x)^4 + 2*sin(x)^2*cos(x)^2 + cos(x)^4).simplify_trig()
1
```

Uočite da Sage neće naivno "pojednostaviti" izraze koji uključuju multifunkcije , kao na slijedećem primjeru, u kojem upoznajemo i važnu metodu `subs()` koja služi za uvrštavanje vrijednosti varijabli i druge supstitucije u izrazima

```
sage: i6 = log(a) + log(b)
sage: i6.simplify_full()
log(a) + log(b)
sage: i6.subs(a=-1,b=-1)
2*I*pi
sage: i7 = log(a*b)
sage: i7.subs(a=-1,b=-1)
0


.. end of output
```

Zadatak 1

Uzmite izraz $(a+b)((c+yx)x+tx^2)$ i algebarskim manipulacijama natjerajte Sage da ga prikaže u slijedećim ekvivalentnim oblicima (gdje redosljed faktora odnosno članova nije bitan):

1. $(a+b)(tx+xy+c)x$
2. $atx^2+ax^2y+btx^2+bx^2y+acx+bcx$

[3] To se može postići na jedan od dva slijedeća zaobilazna načina

Zadatak 2

Koristeći algebarske manipulacije pokažite da vrijedi

$$\frac{\sin^3 x + \cos^3 x}{\sin x + \cos x} = 1 - \sin x \cos x$$

Pazite na sintaksu: $\sin^3 x$ se unosi kao sin(x)^3!

Bilješke

```
sage: sum(i4.coefficient(y, a).factor()*y^a for a in range(4))
(a + 2*b + 3*c)^3*x^3 + 3*(a + 2*b + 3*c)^3*x^2*y +
           3*(a + 2*b + 3*c)^3*x*y^2 + (a + 2*b + 3*c)^3*y^3
```

```
sage: sum([it.factor()*y^eksp for it,eksp in i4.coefficients(y)])
(a + 2*b + 3*c)^3*x^3 + 3*(a + 2*b + 3*c)^3*x^2*y +
           3*(a + 2*b + 3*c)^3*x*y^2 + (a + 2*b + 3*c)^3*y^3
```

```
sage: var('x y a b c')
(x, y, a, b, c)
```

4.2 Jednadžbe

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

```
sage: 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`):

```
sage: (x+y).subs(sol[0])
y - 1/2*sqrt(2)
```

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

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

```
sage: 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 jednostavno primijeniti funkciju `numerical_approx()` (tj. `n()`) na ovu listu rješenja, jer `n()` nije metoda liste (a ni jednadžbe), već samo primitivnijih objekata:

```
sage: n(sol2)
Traceback (most recent call last):
...
TypeError: unable to coerce to a ComplexNumber: <type 'list'>
```

Stoga je potrebno nekako primijeniti `n()` samo na izraze s desne strane gornjih jednadžbi npr.

```
sage: [(x.subs(sol2[k][0]).n(), y.subs(sol2[k][1]).n()) for k in range(2)]
[(-0.894427190999916, -0.447213595499958), (0.894427190999916,
0.447213595499958)]
```

Ovo isto možemo izvesti znatno elegantnije [4] koristeći raspakiravanje tupla, te metodu `rhs()` (*right-hand side* [5]) za pristup desnoj strani jednadžbe:

```
sage: [(x.rhs().n(), y.rhs().n()) for x, y in sol2]
[(-0.894427190999916, -0.447213595499958), (0.894427190999916,
0.447213595499958)]
```

Zadatak 1

Riješite sustav jednadžbi

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

Koliko ima rješenja?

```
sage: var('x y')    # nužno, jer smo ih gore u petlji redefinirali
(x, y)
sage:    # edit here
```

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

```
sage: solve([x+y == 3, 2*x+2*y == 6],x,y)
[[x == -r1 + 3, y == r1]]
```

```
sage: solve([cos(x)*sin(x) == 1/2, x+y == 0],x,y)
[[x == 1/4*pi + pi*z38, y == -1/4*pi - pi*z38]]
```

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

[4] Općenito, računalni kod koji uključuje duboko indeksiranje višedimenzionalnih lista, poput `izraz[0][1]` je nepregledan i nepitonican.

[5] Naravno, postoji i `lhs()`.

```
sage: solve(sin(x)==0, x)
[x == 0]
```

Funkcija `solve()` daje simbolička (analitička) 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:

```
sage: eq = 9*x^6 + 4*x^4 + 3*x^3 + x - 17 == 0
sage: 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:

```
sage: 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]


sage: 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.

```
sage: eq2 = 2 * arctan(x) == x^2
sage: eq2.roots(ring=CC)
Traceback (most recent call last):
...
TypeError: Cannot evaluate symbolic expression to a numeric value.
```

U tom slučaju moramo pribjeći pravom numeričkom rješavanju, pomoću funkcije `find_root()`. Mala komplikacija s `find_root()` je da mu se treba dati interval u kojem traži rješenje i da će pronaći samo jedno. Nakon toga moguće treba tražiti dalje u zadavanjem intervala koji isključuje već pronađena rješenja:

```
sage: find_root(eq2, 0, 10)
0.0


sage: find_root(eq2, 0.1, 10)    # isključujemo 0.0
1.3717743420150883


sage: find_root(eq2, 1.4, 10)    # isključujemo i 1.37
Traceback (most recent call last):
...
RuntimeError: f appears to have no zero on the interval


sage: find_root(eq2, -1., 1)
-2.326813826671918e-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. Povećavanjem radne preciznosti pomoću opcionalnog argumenta `xtol` vidimo da se rješenje još više približava nuli:

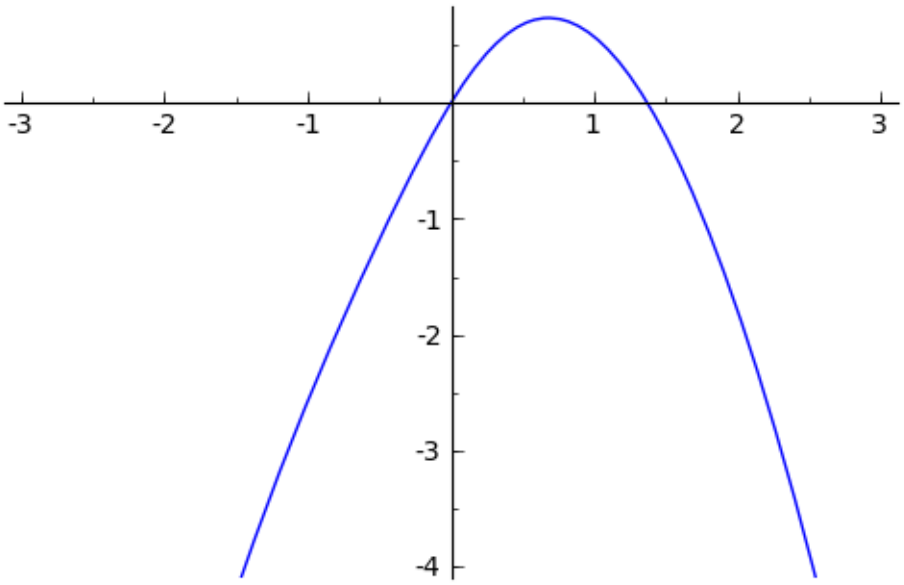
```
sage: find_root(eq2, -1., 1, xtol=1e-30)
4.3470345330696587e-32
```

Sada se možemo uvjeriti da su dva pronađena rješenja zaista jedina, i to tako da skiciramo graf funkcije

$$f(x) = 2\arctan(x) - x^2$$

(čije nultočke su ekvivalentne rješenjima naše jednadžbe) i uočimo da siječe apscisu na samo dva mjesta.

```
sage: plot(2*arctan(x) - x^2, (x, -3, 3), ymin=-4)
```



Zadatak 2

Riješite jednadžbu

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

Zadatak 3

Pronađite numeričku vrijednost nekog kompleksnog rješenja jednadžbe $\sin x = 2$ i provjerite uvrštavanjem.

Zadatak 4

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

Zadatak 5

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

Zadatak 6

Pronađite pozicije minimuma i maksimuma Besselove funkcije $J_1(x)$ koji su najbliži točki $x = 0$.

Bilješke

```
sage: var('a b c x y z t')
(a, b, c, x, y, z, t)
```

4.3 Matematička analiza

Sage sadrži sve standardne operacije koje se uče u matematičkoj analizi, poput limesa, nizova, redova i diferencijalnog računa.

4.3.1 Limesi

Limesi se izvrijednjavaju funkcijom `limit()`:

```
sage: limit(sin(x)/x,  x=0)
1
```

```
sage: limit((1+1/x)^x, x=oo)
e
```

4.3.2 Razvoj u red

Taylorov razvoj neke funkcije po nekoj varijabli oko neke točke do nekog reda radi funkcija `taylor()`:

```
sage: taylor(sin(x), x, 0, 5)
1/120*x^5 - 1/6*x^3 + x
```

Zadatak 1

Odredite Taylorov razvoj funkcije $f(x) = \arctan(x^2 + 1)$ oko točke $x = \infty$ do šestog reda.

Zadatak 2

Kolika je *relativna* pogreška koju radimo ako za izvrijednjavanje $f(2) = \arctan(5)$ koristimo red dobiven u gornjem zadatku? Da li bi greška bila manja da smo upotrebljavali razvoj do šestog reda, ali oko $x = 0$?

Zadatak 3

Za koji x greška razvoja funkcije $f(x) = \arctan(x^2 + 1)$ do šestog reda oko točke $x = 0$ postaje jednaka greški istog razvoja, ali oko točke $x = \infty$?

4.3.3 Derivacije

Deriviranje se radi funkcijom `diff()`

```
sage: diff(exp(2*x)*cos(3*x), x)
2*cos(3*x)*e^(2*x) - 3*e^(2*x)*sin(3*x)
```

Višestruko deriviranje:

```
sage: diff(exp(2*x)*cos(3*x), x, 4)
-119*cos(3*x)*e^(2*x) + 120*e^(2*x)*sin(3*x)
```

Deriviranje izraza koji uključuje opću simboličku funkciju `f(x)`, korištenjem Leibnitzovog lančanog pravila:

```
sage: f = function('f')  #  simboličke funkcije treba deklarirati
sage: diff(x^2*f(x), x)
x^2*D[0](f)(x) + 2*x*f(x)
```

4.3.4 Simboličko integriranje

Simboličko neodređeno integriranje (integriramo izraz koji smo gore dobili deriviranjem):

```
sage: integral(-3*e^(2*x)*sin(3*x) + 2*e^(2*x)*cos(3*x), x).simplify_full()
-4*cos(x)*e^(2*x)*sin(x)^2 + cos(x)*e^(2*x)
```

Ovaj je izraz ekvivalentan onom gore, ali to ovdje nije eksplicitno. Ponekad je korisno eksperimentirati i s drugim algoritmima integriranja:

```
sage: integral(-3*e^(2*x)*sin(3*x) + 2*e^(2*x)*cos(3*x), x,
....:                                     algorithm='sympy')
cos(3*x)*e^(2*x)
```

Primijetite da se konstanta integracije podrazumijeva bez navođenja.

Simboličko rješavanje određenih integrala:

```
sage: integral( x*log(x), x, 0, 1)
-1/4
```

Ukoliko integracija ovisi o nekom parametru, defaultni `maxima` algoritam za integriranje može tražiti da se izjasnimo o svojstvima tog parametrima o kojima ovisi rezultat integracije. Dodjeljivanje svojstava simboličkim varijablama izvodi se pomoću funkcije `assume()`. Kako takve pretpostavke ne bi utjecale na kasnije računa treba ih što prije pobrisati s `forget()`.

```
sage: integral(y*sqrt(x*y), y, 0, 1-x)
Is  x-1  positive, negative, or zero?
Traceback (most recent call last):
...
TypeError: Computation failed since Maxima requested additional constraints
   (try the command 'assume(x-1>0)' before integral or limit
     evaluation, for example):


sage: assume(x-1==0); integral(y*sqrt(x*y), y, 0, 1-x).factor(); forget()
2/5*sqrt(-(x - 1)*x)*(x - 1)^2
```

U ovom konkretnom slučaju rezultat zapravo ne ovisi o predznaku od x, što `maxima` izgleda ne zna, ali `sympy` zna:

```
sage: integral(y*sqrt(x*y), y, 0, 1-x, algorithm='sympy')
2/5*sqrt(x)*(-x + 1)^(5/2)
```

Zadatak 4

Izračunajte neodređeni integral

$$\int \frac{x^2+3}{x^5+x^4-x-1}dx$$

i onda provjerite dobiveni rezultat deriviranjem i *algebarskim manipulacijama*.

Zadatak 5

Izračunajte (simbolički) dvostruki određeni integral

$$\int_0^1 dx \int_0^{1-x} dy\,(x+y)\sqrt{xy^3}$$

Zadatak 6

Izračunajte dvostruki neodređeni integral

$$\int dx \int dy\,(x+y)\sqrt{xy^3}$$

i provjerite rezultat deriviranjem i *algebarskim manipulacijama*.

4.3.5 Numeričko integriranje

Neki integrali su preteški ili se naprosto ne daju prikazati u zatvorenoj formi pa Sage vraća zadani izraz:

```
sage: integral(arctan(gamma(x)), x, 0, 1)
integrate(arctan(gamma(x)), x, 0, 1)
```

Tada nam ostaje numerička integracija pomoću `numerical_integral()` čiji rezultat je dan kao tupl. Prvi element tupla je rezultat integracije, a drugi procijenjena greška.

```
sage: numerical_integral(arctan(gamma(x)), 0, 1)
(1.1020657425550975, 1.2235387620352894e-14)
```

Primijetite da se kod numeričke integracije ne navodi eksplicitno varijabla integracije već se ona određuje automatski. Integrand ionako ne smije ovisiti nego o samo jednoj varijabli da bi se mogao numerički integrirati.

Zadatak 7

Izračunajte integral

$$\int_0^\infty dt\, e^{-t}\sqrt{t}$$

simbolički i numerički i usporedite rezultate.

4.4 Linearna algebra

U Sageu postoje vektori i matrice kao specijalni objekti, ali mi ćemo koristiti NumPy polja koja nude sličnu funkcionalnost zahvaljujući rutinama NumPy modula za linearnu algebru. (Za "domaću" Sage linearnu algebru vidi *ovaj odjeljak*.)

```
sage: import numpy as np
sage: import numpy.linalg as la
```

```
sage: v1 = np.array([1,1,2])
sage: v2 = np.array((2,2,4))
```

Množenje skalarom je prirodno:

```
sage: 3*v1
array([3, 3, 6])
```

Skalarni i vektorski produkt vektora:

```
sage: np.dot(v1, v2)
12
```

```
sage: np.cross(v1, v2)
array([0, 0, 0])
```

Norma ("duljina") vektora:

```
sage: la.norm(v2)
4.8989794855663558
```

Množenje matrica te množenje matrice i vektora ide na prirodan način:

```
sage: A = np.array([[1, 2, 1], [4, 3, 3], [9, 1, 7]]); A
array([[1, 2, 1],
       [4, 3, 3],
       [9, 1, 7]])


sage: np.dot(A, A)
array([[18,  9, 14],
       [43, 20, 34],
       [76, 28, 61]])


sage: np.dot(A, v1)
array([ 5, 13, 24])
```

Determinanta i inverz matrice:

```
sage: la.det(A)    # = 7
-6.9999999999999982


sage: la.inv(A)
array([[-2.57142857,  1.85714286, -0.42857143],
       [ 0.14285714,  0.28571429, -0.14285714],
       [ 3.28571429, -2.42857143,  0.71428571]])
```

Numerika po defaultu ide s *double precision* točnošću, pa kad provjeravamo da li je $A^{-1}A = 1$ ne dobijemo egzaktno jediničnu matricu:

```
sage: should_be_unit = np.dot(la.inv(A), A); should_be_unit
array([[  1.00000000e+00,   1.38777878e-15,   8.32667268e-16],
       [  2.77555756e-17,   1.00000000e+00,  -2.77555756e-17],
       [ -1.22124533e-15,  -7.77156117e-16,   1.00000000e+00]])
```

Moguće je zatražiti od NumPyja da ispisuje ovakve male brojeve kao nule:

```
sage: np.set_printoptions(suppress=True)
sage: should_be_unit
array([[ 1.,  0.,  0.],
       [ 0.,  1., -0.],
       [-0., -0.,  1.]])
```

Zadatak 1

Za datu matricu A definiramo svojstvene vektore (eigenvectors) $\mathbf{v}$ i njima pripadajuće svojstvene vrijednosti λ (eigenvalues) kao rješenja matrične jednadžbe

$$A\mathbf{v} = \lambda\mathbf{v} \ .$$

Pomoću funkcije la.eig() odredite svojstvene vrijednosti i svojstvene vektore matrice

$$\begin{pmatrix} 2.3 & 4.5 \\ 6.7 & -1.2 \end{pmatrix} ,$$

i provjerite da dobivena rješenja zaista zadovoljavaju gornju jednadžbu.

Zadatak 2

Kreirajte 3x3 matricu sa slučajnim realnim brojevima između 0 i 10. Invertirajte je i pomnožite s originalnom matricom te se uvjerite da dobijete jediničnu matricu.

Dijagonalizacija matrice A je pronalaženje njenog rastava oblika

$$A = PDP^{-1}$$

gdje je D dijagonalna matrica. Takav rastav se također izvodi fukcijom la.eig(), koja daje svojstvene vrijednosti i svojstvene vektore matrice. Naime, stupci matrice P su upravo svojstveni vektori od A, a dijagonalna matrica D na dijagonali ima upravo odgovarajuće svojstvene vrijednosti:

```
sage: A = np.array([[3, 1], [1, 3]]); A
array([[3, 1],
       [1, 3]])


sage: evs, P = la.eig(A)


sage: D = np.diag(evs); D
array([[ 4.,  0.],
       [ 0.,  2.]])
sage: np.dot(np.dot(la.inv(P), A), P)  # provjerevamo P^-1 A P = D
array([[ 4.,  0.],
       [ 0.,  2.]])
```

Neke matrice nije moguće dijagonalizirati, u slučaju čega će matrice P i D koje vraća `la.eig()` i dalje zadovoljavati

$$AP = PD$$

ali P neće biti invertibilna:

```
sage: A = np.array([[1, 1], [0, 1]]); A
array([[1, 1],
       [0, 1]])
sage: evs, P = la.eig(A)
```

```
sage: np.dot(A, P) - np.dot(P, np.diag(evs))
array([[ 0.,  0.],
       [ 0.,  0.]])
```

Neinvertibilnost od P se ogleda u ogromnim brojevima koje dobijemo kad "invertiramo" tu matricu:

```
sage: print la.inv(P)
[[  1.00000000e+00   4.50359963e+15]
 [  0.00000000e+00   4.50359963e+15]]
```

4.5 Diferencijalne jednadžbe

4.5.1 Simboličko rješavanje

Osnovna Sage funkcija koja traži analitička rješenja običnih diferencijalnih jednadžbi je `desolve()`. Riješimo pomoću nje jednadžbu gušenog harmoničkog oscilatora

$$\frac{d^2y}{dt^2} + 2\frac{dy}{dt} + 4y = 0\,.$$

Prilikom definiranja diferencijalne jednadžbe treba eksplicitno deklarirati simboličku nezavisnu varijablu (obično je to vrijeme `t`) i nepoznatu simboličku funkciju (ovdje je to `y(t)`), pomoću funkcije `function()`:

```
sage: t = var('t')
sage: y = function('y',t)
sage: desolve(diff(y, t, 2) + 2*diff(y, t) + 4*y == 0, y)
(_K2*cos(sqrt(3)*t) + _K1*sin(sqrt(3)*t))*e^(-t)
```

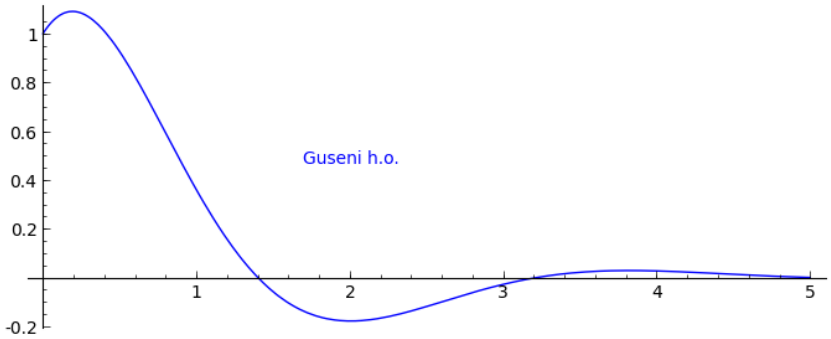
Napomene:

1. Prisjetite se da je znak jednakosti u jednadžbama ==
2. Kao što je kod običnih jednadžbi trebalo eksplicirati po kojoj varijabli se traži rješenje, tako se kod diferencijalnih specificira po kojoj funkciji se traži rješenje
3. Rješenja uključuju nepoznate konstante (k_1 i k_2) koje treba odrediti iz početnih uvjeta

Ukoliko želimo partikularno rješenje za neke konkretne početne uvjete, zadamo te uvjete u dodatnom argumentu `ics` (= *initial conditions*), a koji je lista oblika `ics` = [t_0, $y(t_0)$, $y'(t_0)$]:

```
sage: des = desolve(diff(y, t, 2) + 2*diff(y, t) + 4*y == 0, y,
....:                                              ics=[0, 1, 1]); des
1/3*(2*sqrt(3)*sin(sqrt(3)*t) + 3*cos(sqrt(3)*t))*e^(-t)
```

```
sage: plot(des,(0,5), figsize=[7,3]) + text('Guseni h.o.', (2,0.5))
```



Zadatak 1

Riješite simbolički diferencijalnu jednadžbu

$$\frac{dy}{dt} - y^2 - 1 = 0 \, ,$$

uz početni uvjet $y(0) = 1/2$, provjerite rješenje uvrštavanjem, te ga nacrtajte za t u rasponu $0 < t < 1$.

4.5.2 Numeričko rješavanje

Neke jednadžbe se ne mogu riješiti analitički, pa moramo pribjeći numeričkom integriranju. Za to ćemo koristiti funkciju `odeint` iz paketa `scipy.integrate`.

```
sage: from scipy.integrate import odeint
sage: import matplotlib.pyplot as plt
sage: import numpy as np
```

Kao prvi primjer integrirati ćemo jednadžbu iz gornjeg zadatka. Za `odeint` tu jednadžbu treba transformirati u oblik

$$\frac{dy}{dt} = f(y, t)$$

i korisnik treba definirati Python funkciju koja odgovara desnoj strani ove jednadžbe.

U našem slučaju je $f(y, t) = y^2 + 1$ pa stoga imamo

```
sage: def func(y, t):
....:     return y**2 + 1
```

Funkciju `odeint` treba pozvati s ovom funkcijom kao prvim argumentom, početnom vrijednošću $y(0)$ kao drugim argumentom i listom vremenskih točaka za koje želimo odrediti položaj sustava kao trećim argumentom:

```
sage: y0 = 0.5
sage: ts = np.linspace(0, 1)
sage: ts.shape
(50,)
```

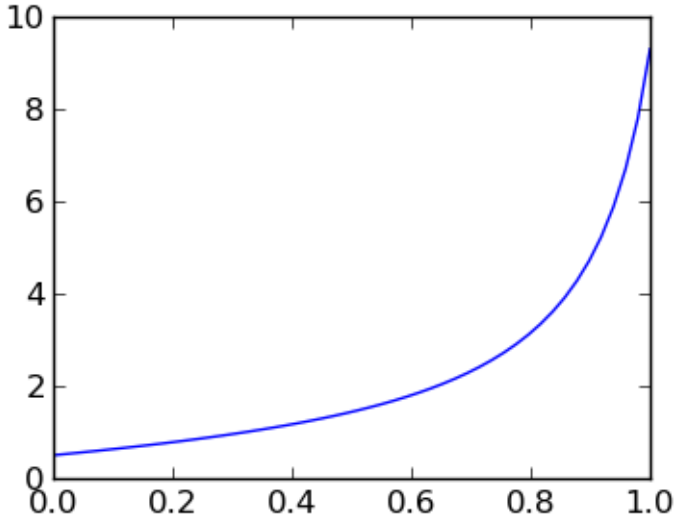
Funkcija `odeint` vraća listu položaja u traženim vremenskim točkama:

```
sage: ys = odeint(func, y0, ts)
sage: ys.shape
(50, 1)
```

```
sage: print "\n%8s  %8s" % ('vrijeme', 'položaj')
sage: print 18*'-'
sage: for t,y in zip(ts[:5], ys[:5, 0]):
....:     print "%8f  %8f" % (t, y)
sage: print "%8s  %8s" % (' (...)  ', ' (...)  ')

 vrijeme  položaj
------------------
0.000000  0.500000
0.020408  0.525777
0.040816  0.552113
0.061224  0.579049
0.081633  0.606630
 (...)      (...)
```

```
sage: fig, ax = plt.subplots(figsize=[4,3])
sage: ax.plot(ts, ys)
sage: fig.savefig('fig')
```



Kao slijedeći primjer proučiti ćemo tzv. van der Polovu jednadžbu, koja je drugog reda:

$$\frac{d^2x}{dt^2} - (1 - x^2)\frac{dx}{dt} + x = 0\,.$$

Jednadžbe višeg reda se za numeričku analizu treba prirediti tako da se svaka pretvori u sustav od dvije jednadžbe prvog reda, što se izvodi uvođenjem nove funkcije $y(t) = dx/dt$ uz pomoć koje gornju jednadžbu možemo pretvoriti u ekvivalentni sustav

$$\frac{dx}{dt} = y$$

$$\frac{dy}{dt} = (1 - x^2)y - x$$

Taj sustav sad treba zapisati u vektorskom obliku tako da poprimi strukturu

$$\frac{dy_k}{dt} = f_k(\vec{y}, t)\,, \qquad k = 1, 2$$

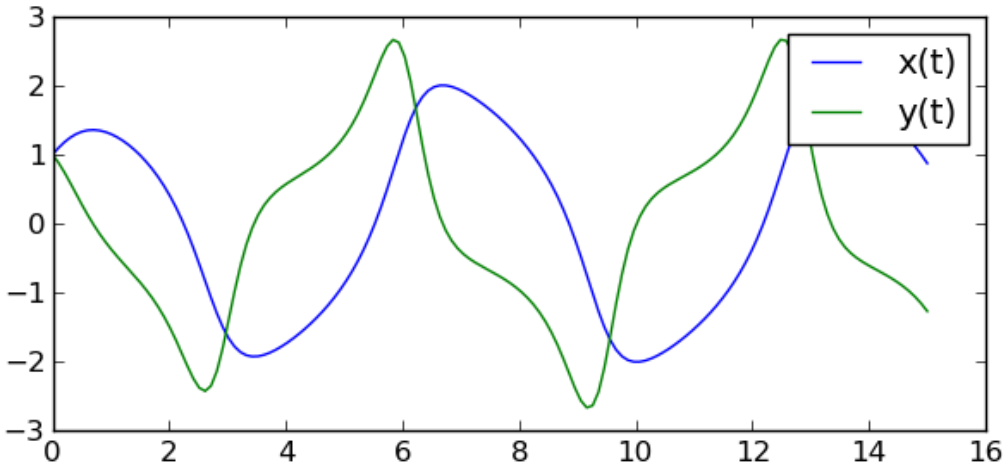
gdje je dvokomponentni vektor $\vec{y} = (x, y) \equiv$ `[y[0], y[1]]`. Sada funkcija koju prirađujemo za `odeint` opet odgovara desnoj strani ove jednadžbe i mora kao rezultat vratiti dvokomponentni vektor $\vec{f} = (f_1, f_2)$

```
sage: def func(y, t):
....:     return [y[1], (1 - y[0]**2)*y[1] - y[0]]
```

Početni uvjet je isto vektor $(x(0), y(0))$, a i rezultat će biti NumPy polje oblika `(broj vremenskih točaka) x (broj nepoznatih funkcija)`.

```
sage: ts = np.linspace(0,15,150)
sage: y0 = [1., 1.]
sage: ys = odeint(func, y0, ts)
sage: ys.shape
(150, 2)
```

```
sage: fig, ax = plt.subplots(figsize=[7,3])
sage: ax.plot(ts, ys[:,0], label='x(t)')
sage: ax.plot(ts, ys[:,1], label='y(t)')
sage: ax.legend()
sage: fig.savefig('fig')
```



Zadatak 2

Nacrtajte graf brzine gornjeg rješenja Van der Polove jednadžbe $y(t) = dx(t)/dt$, ali ne u ovisnosti o vremenu t, već o položaju $x(t)$ (tzv. fazni dijagram).

Zadatak 3

Riješite numerički diferencijalnu jednadžbu gušenog harmoničkog oscilatora. Nacrtajte funkcije `y(t)` i `x(t)`, te, na posebnom grafu, fazni dijagram `y(x)`.

4.6 Statistika

Sage ima implementirane osnovne statističke funkcije, ali nama će biti zgodnije koristiti paket `scipy.stats` čije mogućnosti su trenutno veće.

```
sage: import scipy
sage: import scipy.stats
sage: import matplotlib.pyplot as plt
sage: import numpy as np
```

Kreirajmo prvo NumPy listu brojeva. NumPy liste imaju metode za izračun osnovnih statističkih veličina poput srednje vrijednosti, varijance, standardne devijacije itd.

```
sage: xs=np.linspace(1,9,21); xs
array([ 1. ,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  5. ,
        5.4,  5.8,  6.2,  6.6,  7. ,  7.4,  7.8,  8.2,  8.6,  9. ])
```

```
sage: print "%.2f +- %.2f" % (xs.mean(), xs.std())
5.00 +- 2.42
```

Obične liste nemaju gornje statističke metode pa ukoliko nas zanima npr. srednja vrijednost brojeva u običnoj listi treba je prvo konvertirati u numpy listu funkcijom `np.array()` ili možemo primijeniti Sage funkciju `mean()` (dakle ne metodu).

```
sage: mean([3, 5, 7, 9, 11])
7
```

scipy.stats paket ima implementirane brojne statističke raspodjele:

- `norm` - normalna (Gaussova) raspodjela
- `poisson` - Poissonova raspodjela
- `gamma` - gamma raspodjela
- `chi2` - χ^2 raspodjela
- `t` - studentova t raspodjela
- ...

Svaka od tih raspodjela ima svoje parametre (npr. za Gaussovu raspodjelu su to srednja vrijednost i standardna devijacija). Oni se najčešće specificiraju putem opcionalnih argumenata, a točnu sintaksu saznajemo iz standardne dokumentacije.

Najvažnije metode distribucija su:

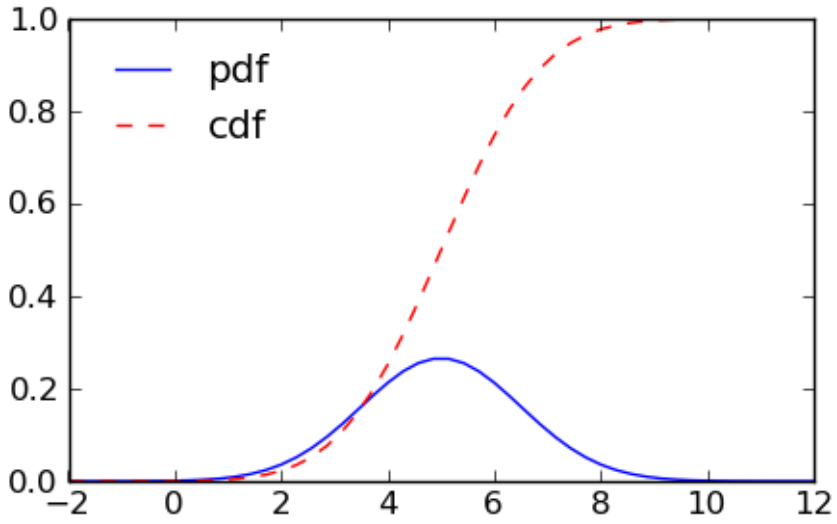
- `pdf()` - sama distribucija (*probability distribution function*) $f(x)$
- `cdf()` - integral distribucije (*cumulative distribution function*) $\Phi(x) = \int_{-\infty}^{x} f(y)dy$
- `rvs()` - slučajni brojevi (*random variates*) koji slijede distribuciju

Tako je npr. vrijednost u $x = 0$ normalne distribucije sa srednjom vrijednosti $\mu = 5$ i standardnom devijacijom $\sigma = 1.5$, $f(x = 0, \mu = 5, \sigma = 1.5)$ dan s:

```
sage: scipy.stats.norm.pdf(0, loc=5., scale=1.5)
0.0010281859975274036
```

Skica distribucije i njenog integrala:

```
sage: xs = np.linspace(-2,12)
sage: fig, ax = plt.subplots(figsize=[5,3])
sage: ax.plot(xs, scipy.stats.norm.pdf(xs, loc=5., scale=1.5), label='pdf')
sage: ax.plot(xs, scipy.stats.norm.cdf(xs, loc=5., scale=1.5), 'r--',
....:                                                          label='cdf')
sage: ax.legend(loc='upper left').draw_frame(0)
sage: fig.savefig('fig')
```



Izračunajmo vjerojatnost da se slučajna varijabla nađe unutar jedne standardne devijacije σ od srednje vrijednosti μ. Ona je dana integralom distribucije od $\mu - \sigma$ do $\mu + \sigma$. Kako na raspolaganju imamo funkciju `cdf()` koja daje integral od $-\infty$ do x, treba samo oduzeti dva takva integrala (koristimo i to da su defaultne vrijednosti `loc=0` i `scale=1`):

```
sage: scipy.stats.norm.cdf(1) - scipy.stats.norm.cdf(-1)
0.68268949213708585
```

(To je čuvenih 68% vjerojatnosti.)

Zadatak 1

Slučajna varijabla X slijedi normalnu raspodjelu sa srednjom vrijednosti $\mu = 50$ i standardnom devijacijom $\sigma = 2$. Kolika je vjerojatnost da se X nađe između 47 i 54?

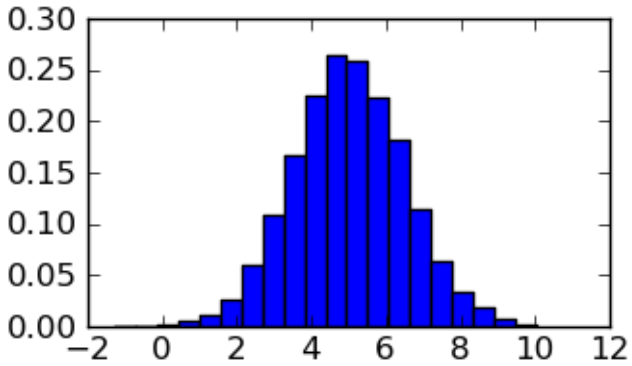
Sad ćemo metodom `rvs()` izgenerirati uzorak od 10000 brojeva raspodjeljenih po normalnoj raspodjeli s $\mu = 5$ i $\sigma = 1.5$ i testirati da su srednja vrijednost i standardna devijacija prema očekivanjima:

```
sage: pts=scipy.stats.norm.rvs(loc=5., scale=1.5, size=1e4)
sage: print "broj točaka = %i" % len(pts)
broj točaka = 10000
sage: print "srednja vrijednost = %.3f" % pts.mean()  # rel tol 0.01
srednja vrijednost = 4.998
sage: print "standardna devijacija = %.3f" % pts.std()  # rel tol 0.03
standardna devijacija = 1.493
```

Za crtanje histograma ovog uzorka koristimo Matplotlibovu `hist()` funkciju, čiji opcionalni argument

`bins` kontrolira broj "binova" dakle broj intervala apscise u kojima se prebrojavaju točke:

```
sage: fig, ax = plt.subplots(figsize=[3.5,2])
sage: ax.hist(pts, bins=20, normed=True)
sage: fig.savefig('fig')
```



Ovo je bilo za jedan uzorak. Listu npr. standardnih devijacija za 5 nezavisnih uzoraka možemo konstruirati ovako:

```
sage: [scipy.stats.norm.rvs(loc=5, scale=1.5, size=1e4).std() for k in
....:                                                 range(5)] # rel tol 0.05
[1.503340173922008, 1.5040004762159596, 1.4785845284471362,
1.4906248759178866, 1.5136584591557063]
```

Zadatak 2

Kolika je vjerojatnost da mjerenje slučajne varijable koja slijedi normalnu razdiobu sa srednjom vrijednošću μ i standardnom devijacijom σ odstupi od srednje vrijednosti za više od 3σ?

Važna raspodjela je tzv. χ^2-raspodjela

$$P(\chi^2, \nu) = \frac{(\chi^2)^{\nu/2-1} e^{-\chi^2/2}}{2^{\nu/2}\Gamma(\nu/2)}$$

Vidimo da χ^2 raspodjela, pored argumenta x, ima samo jedan parametar: ν = df = broj stupnjeva slobode (*degrees of freedom*) [6].

Srednja vrijednost χ^2 raspodjele jednaka je broju stupnjeva slobode:

```
sage: scipy.stats.chi2.rvs(3, size=1e5).mean()  # rel tol 0.01
3.0033578764955071
```

Jedno od svojstava χ^2 raspodjele je da integral repa te raspodjele (s jednim stupnjem) od 1, 4, 9 do ∞ daje vjerojatnost da mjerenje slučajne varijable raspodjeljene po Gaussovoj raspodjeli sa standardnom devijacijom σ odstupi od srednje vrijednosti za $\sigma, 2\sigma, 3\sigma, \ldots$.

```
sage: [1-scipy.stats.chi2.cdf(eps^2,1) for eps in [1,2,3]]
[0.31731050786291404, 0.045500263896358528, 0.0026997960632602069]
```

[6] Naziv tog parametra postaje jasan u kontekstima određivanja dobrote prilagodbe i statističkog *testiranja hipoteze*.

Zadatak 3

Središnji granični teorem kaže da su srednje vrijednosti dovoljno velikih uzoraka neke raspodjele raspodjeljene prema normalnoj (Gaussovoj) raspodjeli bez obzira na to kakva je originalna raspodjela iz koje te uzorke uzimamo. U to ćemo se uvjeriti na primjeru χ^2 raspodjele.

1. Nacrtajte tu raspodjelu za df=3 stupnja slobode i uvjerite se da je asimetrična oko srednje vrijednosti.
2. Uzmite jedan uzorak od milijun točaka te raspodjele i uvjerite se da je standardna devijacija $\sigma = \sqrt{2 \cdot df}$
3. Uzmite 500 uzoraka od po 400 točaka svaki i napravite listu srednjih vrijednosti tih uzoraka. Uvjerite se da je standardna devijacija vrijednosti u listi $\sigma/\sqrt{400}$, kako traži teorem.
4. Nacrtajte histogram srednjih vrijednosti u listi i uvjerite se (superponiranjem krivulje normalne razdiobe) da su one zaista raspodjeljene prema normalnoj razdiobi srednje vrijednosti $\mu = df = 3$ i standardne devijacije $\sigma/\sqrt{400}$

Za ove iste primjere, ali izvedene pomoću Sage funkcija, vidi odjeljak *Statistika - alternative*.

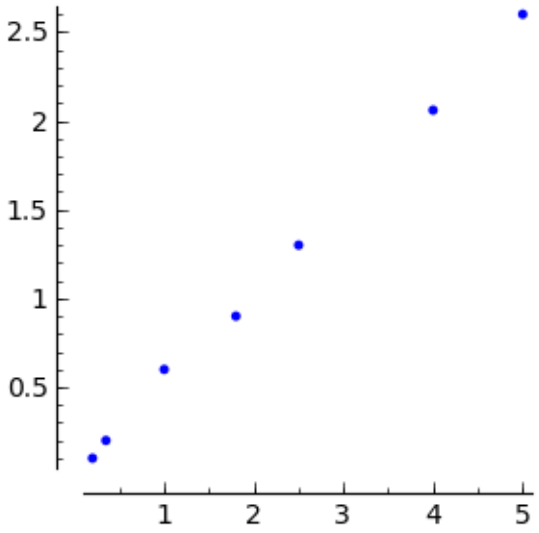
Bilješke

4.7 Prilagodba funkcije podacima

Često je potrebno neke podatke dobivene npr. mjerenjima grafički prikazati i približno opisati nekom funkcijom (tzv. prilagodba ili "fitanje").

```
sage: data = [[0.2,  0.1], [0.35,  0.2], [1, 0.6], [1.8, 0.9], [2.5, 1.3],
....:                                                [4, 2.06], [5, 2.6]]
```

:: sage: list_plot(data, figsize=[3,3])



Klasična situacija je potreba za prilagodbom pravca $f(x) = a + bx$ metodom najmanjih kvadrata (linearna regresija). Pogledajmo prvo kako bismo sami implementirali standardne formule za metodu najmanjih kvadrata:

```
sage: xs = [x for x,y in data]
sage: ys = [y for x,y in data]
sage: xs2 = [x^2 for x,y in data]
sage: ys2 = [y^2 for x,y in data]
sage: xys = [x*y for x,y in data]
sage: n = len(data)
sage: delc = n*sum(xs2) - sum(xs)^2
sage: b = (n*sum(xys)-sum(xs)*sum(ys))/delc
sage: a = (sum(xs2)*sum(ys)-sum(xs)*sum(xys))/delc
sage: sigsq = sum([(y-b*x-a)^2 for x,y in data])/(n-2)
sage: erra = sqrt(sum(xs2)*sigsq/delc)
sage: errb = sqrt(n*sigsq/delc)
sage: print "a = %.3f +- %.3f" % (a, erra)
a = 0.020 +- 0.023
sage: print "b = %.3f +- %.3f" % (b, errb)
b = 0.513 +- 0.008
```

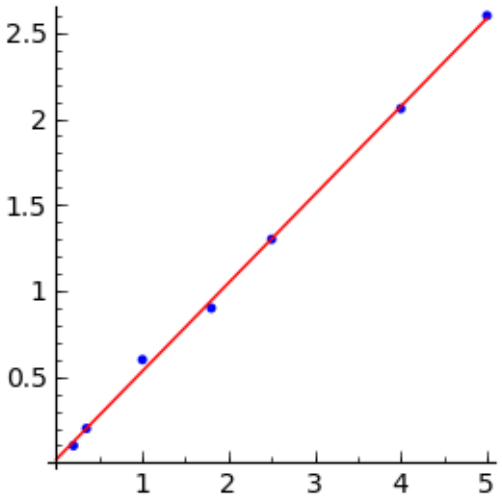
Sage sadrži funkciju `find_fit()` koja radi otprilike to isto, ali je općenitija u smislu da krivulja koju prilagođavamo može biti zadana bilo kojim matematičkim izrazom. Pogledajmo prvo prilagodbu pravca:

```
sage: var('a b')
(a, b)
sage: model(x) = a + b*x


sage: fit = find_fit(data, model, solution_dict=True); fit
{b: 0.5130561168359806, a: 0.020159524186939004}
```

Ovdje smo zatražili ispis rješenja u obliku *rječnika* kako bismo ga lako uvrstili u model putem metode `subs()`:

```
sage: list_plot(data, figsize=[3,3]) + plot(model(x).subs(fit), (x, 0, 5),
....:                                                          color='red')
```



Zadatak 1

Donjim podacima iz liste `data2` prilagodite polinom trećeg reda, te funkciju $f(x) = a\sin(x)$ te nacrtajte sve na jednom grafu. Radi lakšeg snalaženja neka krivulje budu različitih boja. Izračunajte zbroj kvadrata odstupanja $\sum_i (y_i - f(x_i))^2$ za oba slučaja.

```
sage: data2 = [[0, 0.1], [0.3, 0.4], [1, 0.7], [1.8, 1], [2.5, 0.5],
....:                                                  [4, -0.6], [5, -1.0]]
```

Nažalost, find_fit() ne daje greške parametara pravca a i b. Za to trebamo koristiti naprednije funkcije za prilagodbu. Zgodna takva funkcija je `leastsq()` iz paketa `scipy.optimize`. Ona kao svoj *prvi* argument traži funkciju koju minimiziramo (kod metode najmanjih kvadrata to je odstupanje funkcije u točki od ordinate točke, $y_i - f(x_i)$, ali ne kvadrirano jer `leastsq()` sam radi kvadriranje!). Tu funkciju treba definirati kao Python funkciju. Npr. za prilagodbu pravca funkcija koju minimiziramo treba vraćati listu odstupanja pravca od točaka:

```
sage: def distances(p, data):
....:     return [(y - p[0] - p[1]*x) for x,y in data]
```

Prvi argument ove funkcije je lista `p[]` parametara koje prilagođavamo. Početne vrijednosti tih parametara specificiraju se u *drugom* argumentu funkcije‘‘leastsq()‘‘.

Treći argument* `leastsq()` je tuple s eventualnim dodatnim argumentima funkcije koju minimiziramo (prvi argument te funkcije je uvijek lista parametara koje prilagođavamo). Ovdje ćemo taj dodatni argument iskoristiti da proslijedimo toj funkciji same podatke (`data`) na koje prilagođujemo funkciju.

Po defaultu, `leastsq()` isto vraća samo tupl s listom prilagođenih vrijednosti parametara (bez grešaka) i zastavicom (*flag*) koja označava tip pronađenog rješenja ili eventualne probleme:

```
sage: import scipy
sage: scipy.optimize.leastsq(distances, [1.,1.],  (data,))
(array([ 0.02015952,  0.51305612]), 3)
```

Ukoliko želimo i greške koristimo opcionalni argument `full_output=True`, koji nam daje i tzv. matricu kovarijanci (`cov_x` dolje) iz koje odredimo greške parametara na slijedeći način [7].

```
sage: p_final, cov_x, infodict, msg, ier = scipy.optimize.leastsq(
....:                           distances, [1.,1.],  (data,), full_output=True)
sage: vr = scipy.dot(distances(p_final, data), distances(
....:                                 p_final, data))/(len(data)-len(p_final))
sage: p_errs = sqrt(scipy.diagonal(vr*cov_x))
sage: print "p[0] =%8.3f +- %.4f" % (p_final[0], p_errs[0])
p[0] =   0.020 +- 0.0231
sage: print "p[1] =%8.3f +- %.4f" % (p_final[1], p_errs[1])
p[1] =   0.513 +- 0.0085
```

`leastsq()` također omogućuje i prilagodbu složenijih funkcija. Npr. možemo se pitati trebamo li gornjoj funkciji dodati i kvadratni član:

[7] Gornji način određivanja greške parametara se može prihvatiti kao recept, no za one koji žele znati evo objašnjenja: Kad se minimizira ispravno normalizirana funkcija onda dijagonalni elementi matrice kovarijanci izravno daju greške parametara. No, za običnu prilagodbu metodom najmanjih kvadrata, minimizira se kvadrat *apsolutne* greške pa je potrebno renormalizirati matricu kovarijanci varijancom mjerenja, a procjenitelj te varijance je upravo `var=(suma kvadrata odstupanja)/(broj stupnjeva slobode)`. Inače, kad se tako vrijednost kvadrata odstupanja iskoristi za procjenu greške mjerenja ista se više ne može jednostavno iskoristiti za procjenu dobrote fita. (Vidi diskusiju u Bevingtonu ispod Eq. (6.15).)

```
sage: def distances(p, data):
....:     return [(y - p[0] - p[1]*x - p[2]*x^2) for x,y in data]

sage: p_final, cov_x, infodict, msg, ier = scipy.optimize.leastsq(
....:                     distances, [1.,1.,1.],  (data,), full_output=True)
sage: vr = scipy.dot(distances(p_final, data), distances(p_final,
....:                                 data))/(len(data)-len(p_final))
sage: p_errs = sqrt(scipy.diagonal(vr*cov_x))
sage: for k in range(len(p_final)):
....:     print "p[%i] =%8.3f +- %.4f" % (k, p_final[k], p_errs[k])
p[0] =   0.027 +- 0.0350
p[1] =   0.502 +- 0.0373
p[2] =   0.002 +- 0.0071
```

Činjenica da je greška paramera `p[2]` znatno veća od vrijednosti samog parametra, sugerira da je kvadratni član suvišan. (Štoviše, vidimo da je i slobodni član `p[0]` od malog značaja.)

Bilješke

POGLAVLJE 5

Fizika

5.1 Mehanika

Zadatak M-1

Neka na česticu u ravnini djeluje potencijal $U(x, y) = -4x^2y^3$. Nacrtajte U(x,y) i pomoću `contour_plot()` i pomoću `plot3d()`. Nadalje, rješavajući numerički Newtonovu diferencijalnu jednadžbu gibanja, pronađite putanju čestice, uz početne uvjete $x(0) = y(0) = -1, v_x(0) = 2, v_y(0) = -3$, od trenutka $t = 0$ do trenutka $t = 1.43$. Nacrtajte tu putanju, a onda je pokušajte superponirati na oba gornja dijagrama potencijala. Igrajte se malo s početnim uvjetima i uvjerite se da je sve OK. Izvrijednite ukupnu energiju za razna vremena i uvjerite se da je sustav zaista konzervativan tj. da je energija konstantna.

Zadatak M-2

Formule za nerelativističku i relativističku kinetičku energiju tijela mase `m` koje se giba brzinom `v` su

$$K_{\rm nr} = \frac{1}{2}mv^2 \ , \qquad K_{\rm rel} = mc^2\left(\frac{1}{\sqrt{1-v^2/c^2}} - 1\right) \ .$$

1. Definirajte odgovarajuće funkcije `knr(v)` i `krel(v)` te usporedite na istom dijagramu ponašanje tih funkcija za brzine od 0 do $3 \cdot 10^8$ m/s za jediničnu vrijednost mase. Označite veličine i njihove jedinice na koordinatnim osima!
2. Odredite brzinu pri kojoj je greška nerelativističke formule tisućinku promila.
3. Ako ste definirali funkcije kao Python funkcije, ponovite zadatak sa simboličkim funkcijama i obratno.
4. Da li se `knr(v)` i `krel(v)` slažu za male vrijednosti brzine? Ako ne, korigirajte `krel()` da postignete slaganje.

5.1.1 Problem tri tijela

Numeričkim rješavanjem Newtonovih jednadžbi simuliramo gibanje tri tijela koja gravitiraju, a čije je gibanje ograničeno na x-y ravninu. Radimo s jediničnim masama i G=1 vrijednošću Newtonove gravitacijske konstante. Za tri tijela u dvodimenzionalnoj ravnini imamo 6 diferencijalnih jednadžbi drugog reda koje pretvaramo u 12 diferncijalnih jednadžbi prvog reda.

```
sage: def system(t, y):
....:     # y_i = (x1, y1, x2, y2, x3, y3
....:     #  i=     0   1   2   3   4   5
....:     #        vx1, vy1, vx2, vy2, vx3, vy3)
....:     #         6    7    8    9   10   11
....:     return [y[6], y[7], y[8], y[9], y[10], y[11],
....:     -((y[0] - y[2])/((y[0] - y[2])**2 + (y[1] - y[3])**2)**1.5) -
....:     (y[0] - y[4])/((y[0] - y[4])**2 + (y[1] - y[5])**2)**1.5,
....:     -((y[1] - y[3])/((y[0] - y[2])**2 + (y[1] - y[3])**2)**1.5) -
....:     (y[1] - y[5])/((y[0] - y[4])**2 + (y[1] - y[5])**2)**1.5,
....:     (y[0] - y[2])/((y[0] - y[2])**2 + (y[1] - y[3])**2)**1.5 -
....:     (y[2] - y[4])/((y[2] - y[4])**2 + (y[3] - y[5])**2)**1.5,
....:     (y[1] - y[3])/((y[0] - y[2])**2 + (y[1] - y[3])**2)**1.5 -
....:     (y[3] - y[5])/((y[2] - y[4])**2 + (y[3] - y[5])**2)**1.5,
....:     (y[0] - y[4])/((y[0] - y[4])**2 + (y[1] - y[5])**2)**1.5 +
....:     (y[2] - y[4])/((y[2] - y[4])**2 + (y[3] - y[5])**2)**1.5,
....:     (y[1] - y[5])/((y[0] - y[4])**2 + (y[1] - y[5])**2)**1.5 +
....:     (y[3] - y[5])/((y[2] - y[4])**2 + (y[3] - y[5])**2)**1.5,
....:     ]
```

Za početne uvjete uzimamo neke od 15 periodičkih orbita navedenih u Tablici 1 članka Šuvakov i Dmitrašinović, Phys. Rev. Lett. 110, 114301 (2013), dostupno online: arXiv:1303.0181.

```
sage: figure8=[[-1,0,1,0,0,0,0.347111,0.532728,0.347111,0.532728,-0.694222,
....:            -1.06546],[-1.05783,1.05784,-0.522919,0.522917],6.32445]
sage: goggles=[[-1,0,1,0,0,0,0.0833,0.127889,0.0833,0.127889,-0.1666,
....:            -0.255778], [-1.04973,1.04972,-0.738248,0.738249],10.4668]
sage: butterfly2=[[-1,0,1,0,0,0,0.392955,0.097579,0.392955,0.097579,-0.78591,
....:               -0.195158],[-1.07444,1.07449,-0.174197,0.17414],7.00391]
sage: yinyang2a=[[-1,0,1,0,0,0,0.416822,0.330333,0.416822,0.330333,
....:      -0.833644,-0.660666],[-1.0874,1.08734,-0.88595,0.885999],55.7898]
```

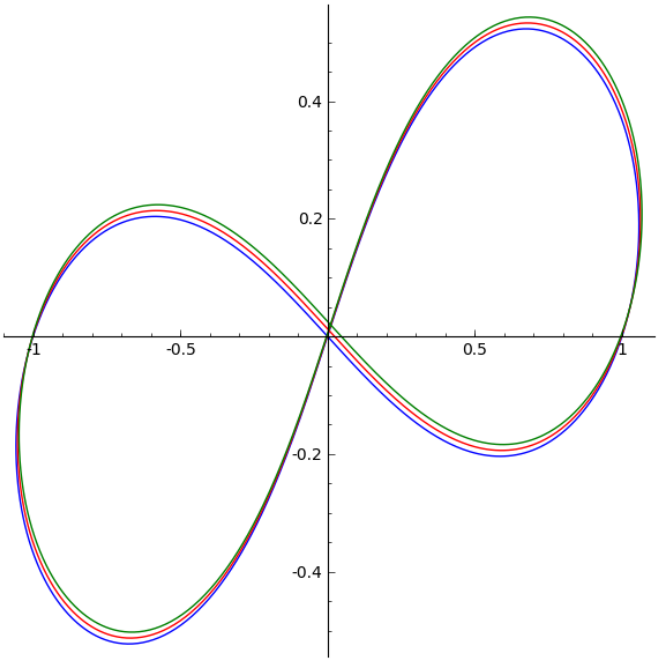
Umjesto `odeint`, koristit čemo `ode_solver`, vidi *Diferencijalne jednadžbe (alternative)*.

```
sage: S = ode_solver()
sage: S.function = system
sage: S.ode_solve(y_0=figure8[0], t_span=[0, figure8[2]], num_points=1000)
sage: sol_figure8 = S.solution
sage: S.ode_solve(y_0=goggles[0], t_span=[0, goggles[2]], num_points=2000)
sage: sol_goggles = S.solution
sage: S.ode_solve(y_0=butterfly2[0], t_span=[0, butterfly2[2]],
....:                                                 num_points=2000)
sage: sol_butterfly2 = S.solution
sage: S.ode_solve(y_0=yinyang2a[0], t_span=[0, yinyang2a[2]],
....:                                                 num_points=6000)
sage: sol_yinyang2a = S.solution
```

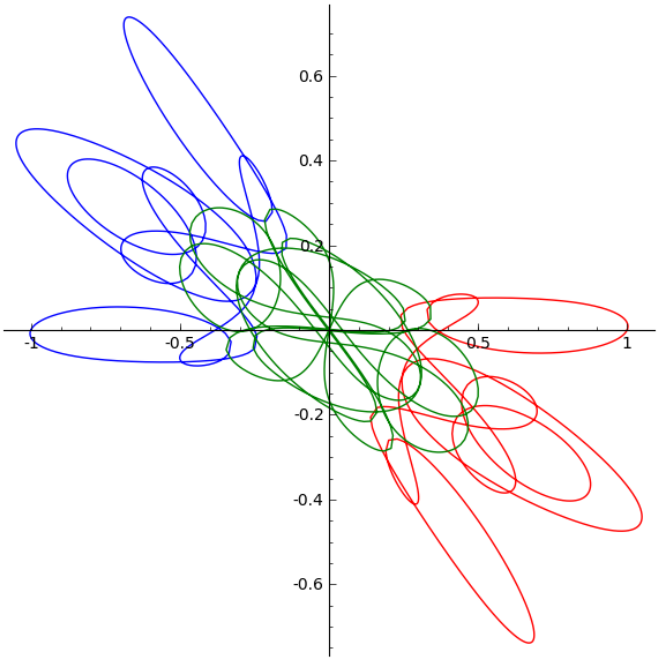
Također, umjesto crtanja Matplotlibom koristit ćemo Sageove grafičke elemente, konkretno primitivni grafički objekt `line`.

```
sage: G = Graphics()
sage: xshift = 0.005 # for visibility
sage: yshift = 0.01 # for visibility
sage: G += line([(x[0],x[1]) for t,x in sol_figure8])
sage: G += line([(x[2]+xshift,x[3]+yshift) for t,x in sol_figure8],
....:                                                   color='red')
```

```
sage: G += line([(x[4]+2*xshift,x[5]+2*yshift) for t,x in sol_figure8],
....:                                                 color='green')
sage: show(G, figsize = [6,6])
```

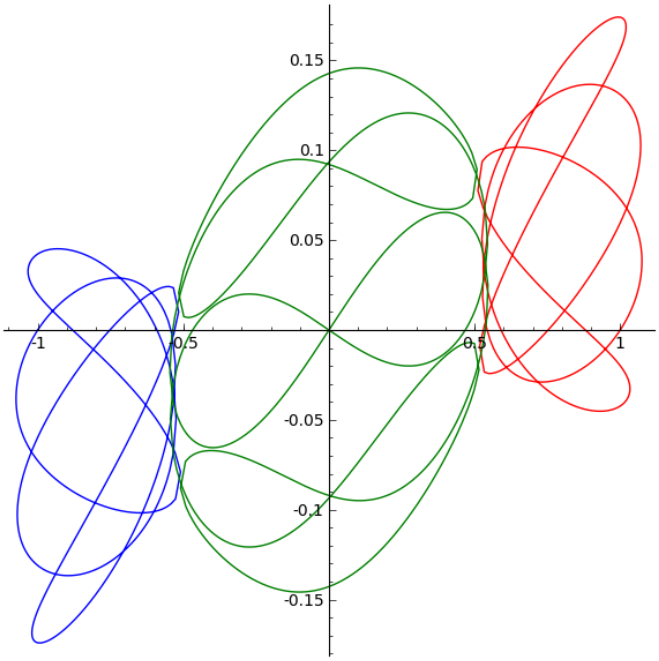



```
sage: G = Graphics()
sage: G += line([(x[0],x[1]) for t,x in sol_goggles])
sage: G += line([(x[2],x[3]) for t,x in sol_goggles], color='red')
sage: G += line([(x[4],x[5]) for t,x in sol_goggles], color='green')
sage: show(G, figsize = [6,6])
```

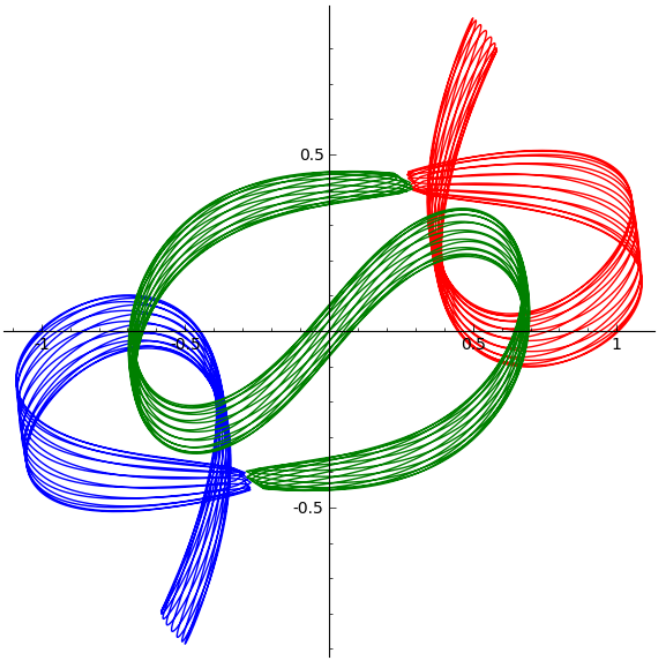



```
sage: G = Graphics()
sage: G += line([(x[0],x[1]) for t,x in sol_butterfly2])
sage: G += line([(x[2],x[3]) for t,x in sol_butterfly2], color='red')
```

```
sage: G += line([(x[4],x[5]) for t,x in sol_butterfly2], color='green')
sage: show(G, figsize = [6,6])
```




```
sage: G = Graphics()
sage: G += line([(x[0],x[1]) for t,x in sol_yinyang2a])
sage: G += line([(x[2],x[3]) for t,x in sol_yinyang2a], color='red')
sage: G += line([(x[4],x[5]) for t,x in sol_yinyang2a], color='green')
sage: show(G, figsize = [6,6])
```



5.2 Elektrodinamika

Zadatak E-1

Potencijal $\phi(\vec{r})$, u točki $\vec{r} = (x, y, z)$ koji je posljedica statičke raspodjele N nabijenih čestica s nabojima $q_1, q_2, \ldots, q_N$ i položajima $\vec{r}_1 = (x_1, y_1, z_1)$, $\vec{r}_2 = (x_2, y_2, z_2)$,..., $\vec{r}_N = (x_N, y_N, z_N)$ je dan formulom:

$$\phi(x, y, z) = \sum_{k=1}^{N} \frac{q_k}{|\vec{r} - \vec{r}_k|} \, .$$

Definirajte funkciju `plotpot(naboji, xgranice, ygranice)` koja crta ekvipotencijalne konture u ravnini $z = 0$, gdje su naboji i njihovi položaji zadani kao lista oblika $[(q_1, x_1, y_1, z_1), (q_2, x_2, y_2, z_2), \ldots]$. Upotrijebite tu funkciju da nacrtate ekvipotencijalne konture za naboje `[(-1, 1, 1, 1), (+1, -1, -1, 0.3), (+1, -1, 1, 0.5)]`.
Naputak: Koristite funkciju `contour_plot()` da nacrtate $\phi(x, y, 0)$

5.2.1 Zračenje ubrzavanog naboja

Snaga zračenja po prostornom kutu za točkasti naboj q brzine $\vec{\beta}$ (u jedinicama brzine svjetlosti c) i ubrzanja $\vec{a}$ dana je Poyntingovim vektorom:

$$\frac{dP}{d\Omega} = \frac{q^2 a^2}{16\pi^2 \epsilon_0 c^3} \frac{|\hat{n} \times ((\hat{n} - \vec{\beta}) \times \hat{a})|^2}{(1 - \hat{n} \cdot \vec{\beta})^5} \, .$$

gdje je $\hat{n}$ jedinični vektor u smjeru promatrača, a $\hat{a}$ jedinični bezdimenzionalni vektor ubrzanja. Cf. npr. Griffits, Eq. (11.72)

```
sage: var('theta phi')
(theta, phi)
```

```
sage: def unitn(theta, phi):
....:     """Unit vector \hat{n} in Cartesian coordinate system."""
....:
....:     return vector((sin(theta)*cos(phi), sin(theta)*sin(phi),
....:                                                 cos(theta)))
```

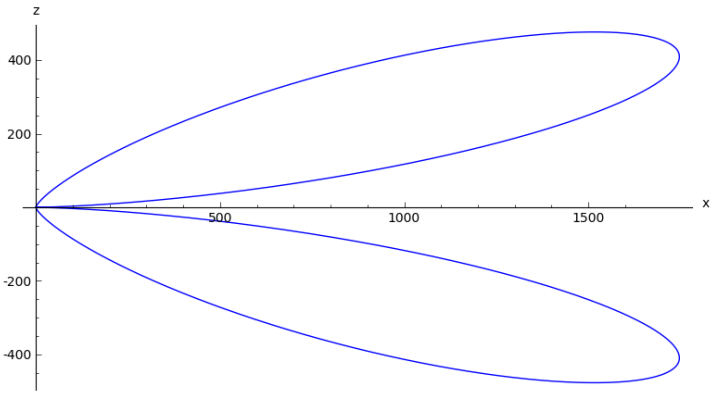
Funkcija poynting() definira samo drugi bezdimenzionalni faktor u gornjem izrazu koji opisuje kutnu ovisnost

```
sage: def poynting(theta, phi, beta, ahat):
....:     """Angular part of Poynting vector."""
....:
....:     nk = unitn(theta, phi)
....:     num = (nk.cross_product((nk-beta).cross_product(ahat))).norm()^2
....:     denom = (1-nk.dot_product(beta))^5
....:     return (num/denom) * nk
```

```
sage: def poynting2d(theta, v, a):
....:     """Projection of Poynting vector on phi=0, i.e x-z plane."""
....:
....:     p = poynting(theta, 0, v, a)
....:     return (p[0], p[2])
```

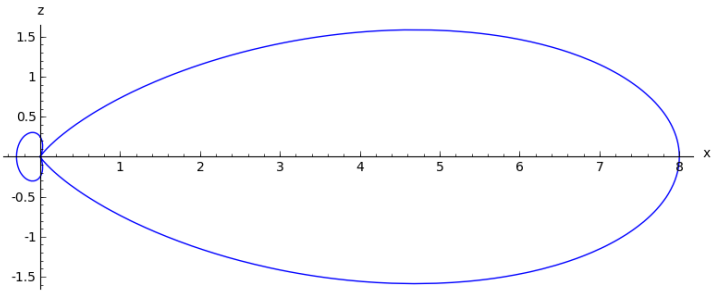
Zračenje kad je akceleracija u smjeru brzine. Cf. Griffits, slika 11.14.

```
sage: parametric_plot(poynting2d(theta, vector((0.9, 0, 0)),
....:     vector((1, 0, 0))), (theta, 0, 2*pi), axes_labels=['x', 'z'])
```



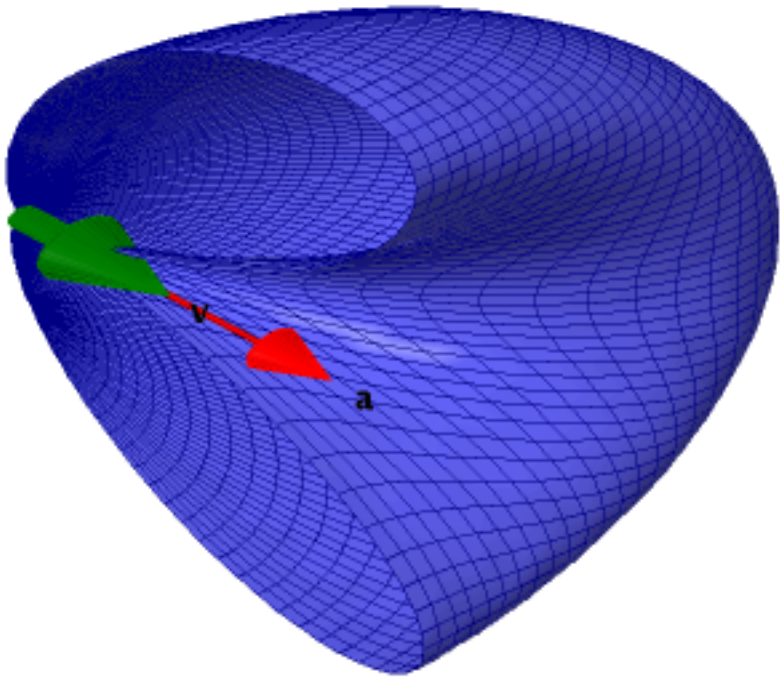
Sinhrotronsko zračenje, kad je akceleracija okomita na brzinu. Cf. Griffits, slika 11.16.

```
sage: parametric_plot(poynting2d(theta, vector((0.5, 0, 0)),
....:    vector((0, 0, 1))), (theta, 0, 2*pi), axes_labels=['x', 'z'])
```




```
sage: def plotField(v=vector((0.5, 0, 0)), a=vector((1, 0, 0))):
....:     """Plot the 3D shape of radiation field (cut along x-z plane)."""
....:
....:     R = parametric_plot3d(poynting(theta, phi, v, a), (theta, 0, pi),
....:          (phi, 0, pi), plot_points=[100,100], frame=False, opacity=0.7)
....:     Av = arrow3d((0,0,0), 3*v, color='green', width=5)
....:     Aa = arrow3d((0,0,0), 3*a, color='red', width=3)
....:     Tv = text3d("v", 3.6*v, size=15)
....:     Ta = text3d("a", 3.3*a, size=50)
....:     return R + Av + Aa +Tv + Ta

sage: plotField().show(mesh=True)
```



5.2.2 Ukupna snaga zračenja za sinhrotronsko zračenje:

Za elektron u magnetskom polju B, okomitom na njegovu trenutnu brzinu, ubrzanje usljed Lorentzove sile je

$$a = \frac{qc\beta B}{m_e} .$$

Nakon uvrštavanja, rezultat je zgodno izraziti preko Thomsonovog udarnog presjeka

$$\sigma_T = \frac{2\mu_0 e^4}{12\pi m_e^2 \epsilon_0 c^2} = 6.652\,\mathrm{m}^2 ,$$

i gustoće energije magnetskog polja

$$U_B = \frac{B^2}{2\mu_0} ,$$

pa se konačno dobije za ukupnu snagu zračenja:

$$P = \frac{3}{4\pi}\sigma_T c\beta^2 U_B \int d\Omega \frac{|\hat{n} \times ((\vec{n} - \vec{\beta}) \times \hat{a}|^2}{(1 - \hat{n} \cdot \vec{\beta})^5} .$$

Iskoristiti ćemo ovaj primjer za demonstraciju računanja s mjernim jedinicama upotrebom SciPy modula constants i Pythonovog modula units.

```
sage: from scipy import constants
sage: c = constants.c * (units.length.meter / units.time.second)
sage: q = units.charge.elementary_charge
sage: sigmaT = constants.physical_constants['Thomson cross section'
....:                                        ][0] * units.length.meter^2
sage: mu0 = constants.mu_0
sage: print "Thomsonov udarni presjek = " + str(sigmaT)
Thomsonov udarni presjek = (6.652458734e-29)*meter^2
```

```
sage: UB(B) = B^2/2/constants.mu_0 * (units.energy.joule/units.length.meter^3)
```

Relevantni integral po prostornom kutu:

```
sage: def Kint(v=vector((0.5, 0, 0)), a=vector((0, 0, 1))):
....:     return integral_numerical(lambda theta: sin(theta) *
....:       integral_numerical(lambda phi: poynting(theta, phi, v,
....:          a).dot_product(unitn(theta, phi)), 0, 2*pi)[0], 0, pi)[0]
```

```
sage: def pwr(beta, B):
....:     return (3/4/pi.n()) * sigmaT * c * beta^2 * UB(B) * Kint(
....:                                              v=vector((beta,0,0)))
```

```
sage: print "Ukupna snaga zracenja = " + str(pwr(0.5, 1))
Ukupna snaga zracenja = (7.05359483944320e-15)*joule/second
```

Za kontrolu uspoređujemo s formulom

$$P = 2\sigma_T \beta^2 \gamma^2 c U_B$$

```
sage: gamma(beta) = 1/(1-beta^2)
sage: pwr2(beta, B) = 2 * sigmaT * beta^2 * gamma(beta)^2 * c * UB(B)
```

```
sage: pwr2(0.5, 1.)
(7.05359483944320e-15)*joule/second
```

5.3 Termodinamika i statistička fizika

5.3.1 Brownovo gibanje

Zadatak T-1

Konstruirajte funkciju `brown(n)` koja daje listu $[(x_0, y_0), (x_1, y_1), \ldots]$ pozicija čestice koja se giba u ravnini slučajnim gibanjem koje je definirano rekurzijama $x_i = x_{i-1} + r$ i $y_i = y_{i-1} + s$ gdje su r i s slučajni brojevi između -1 i 1. Nacrtajte odgovarajuću putanju čestice.
Naputak:

- Inicijalizirajte listu pozicija tako da sadrži `(0,0)` kao prvi vektor pozicije i putem petlje dodajte u svakom koraku toj listi novi slučajni vektor.

Zadatak T-2

Konstruirajte funkciju `walk1D()` tako da simulira tzv. jednodimenzionalnog nasumičnog šetača. Svaki šetačev korak treba biti iste, jedinične duljine, ulijevo ili udesno. Dakle, umjesto vektora `(r, s)` iz gornjeg Brownovog gibanja trebamo slučajan odabir koraka +1 ili -1. Uvjerite se u ispravnost tvrdnje da je prosječna kvadratna udaljenost nasumičnog jednodimenzionalnog šetača od ishodišta nakon N koraka N, sa standardnom devijacijom $\sqrt{2}N$, dok je prosječna apsolutna udaljenost $\sqrt{2N/\pi}$.

5.4 Kvantna fizika

5.4.1 Pravokutna potencijalna jama

Isprogramirat ćemo funkciju koja računa kvantnomehaničke valne funkcije i pripadajuće energije vezanih stanja čestice mase m u pravokutnoj potencijalnoj jami dubine V i širine L. Koristimo oznake s Wikipedijine stranice.

```
sage: var('a k m L V E A B G H')
(a, k, m, L, V, E, A, B, G, H)


sage: import scipy.linalg as la
sage: import numpy as np
sage: import functools
```

Definiramo valnu funkciju za potencijalnu jamu širine L.

```
sage: # components of piecewise wave function
sage: psi1(G, a, x) = G*exp(a*x)
sage: psi2(A, B, k, x) = A*sin(k*x) + B*cos(k*x)
sage: psi3(H, a, x) = H*exp(-a*x)
sage: # and a complete wave function
sage: def psi(x, coefs=(0,0,0,0), wns=(0, 0), L=1, norm=1, shift=0):
....:     """ coefs = (A, B, G, H)
....:         wns = (k, a)
....:     """
....:     if x < -L/2:
....:         psi = psi1(coefs[2], wns[1], x)
....:     elif x > L/2:
....:         psi = psi3(coefs[3], wns[1], x)
....:     else:
....:         psi = psi2(coefs[0], coefs[1], wns[0], x)
....:     return norm*psi + shift
```

Valni brojevi u unutrašnjosti k i izvan jame a.

```
sage: # hbar=1
sage: wns = {k: sqrt(2*m*E), a: sqrt(2*m*(V-E))}
```

Rubni uvjeti na lijevom i desnom rubu jame su da tamo valna funkcija i njena prva derivacija budu neprekidne.

```
sage: boundary_conditions = [
....:     psi1(G, a, -L/2) == psi2(A, B, k, -L/2),
....:     psi3(H, a,  L/2) == psi2(A, B, k,  L/2),
....:     diff(psi1(G, a, x), x).subs(x=-L/2) == diff(psi2(A, B, k, x),
....:                                                 x).subs(x=-L/2),
....:     diff(psi3(H, a, x), x).subs(x= L/2) == diff(psi2(A, B, k, x),
....:                                                 x).subs(x= L/2)
....:                     ]
```

Ovi rubni uvjeti predstavljaju homogeni linearni sustav jednadžbi s nepoznanicama A, B, G i H. Energije vezanih stanja su određene zahtjevom da taj sustav ima rješenje. Konkretno, pretvorit ćemo rubne uvjete u matričnu jednadžbu oblika

$$M \begin{pmatrix} A \\ B \\ G \\ H \end{pmatrix} = 0$$

a energije su onda dane kao rješenja obične jednadžbe $\det M = 0$. Zbog trenutnog buga u Sageovoj rutini za determinantu matrica 4x4 i većih (trebao bi biti ispravljen u nadolazećoj verziji Sagea) definiramo svoju rutinu za izračun determinante:

```
sage: def mydet(m):
....:     "Evaluates determinant of 4x4 matrix m. (Sage's det() has a bug.)"
....:     d = 0
....:     for c in range(4):
....:         ci = range(4)
....:         ci.pop(c)
....:         sub = m.matrix_from_rows_and_columns([1,2,3], ci)
....:         d += (-1)^(c) * m[0,c] * det(sub)
....:     return d
```

Elemente matrice M dobivamo metodom simboličkog izraza `coef()`, a jednadžbu $\det M = 0$ rješavamo algoritmom u kojem krečemo od liste intervala `[(0,V)]`, koja sadrži u početku samo jedan interval `(0, V)` i postupamo ovako:

1. Maknemo prvi interval `(a, b)` iz liste (metoda `pop()`) i tražimo u njemu nultočku. To može imati dva ishoda

1. Našli smo nultočku (taj ishod je zagarantiran u prvom koraku jer jedno vezano stanje uvijek postoji). Zapišemo nađenu energiju, a listu intervala nadopunimo s dva nova intervala: `(a, E)` i `(E, b)`.

2. Nismo našli nultočku. Ne radimo ništa (interval je već maknut s popisa).

2. Ponavljamo postupak iz točke 1. dok ne iscrpimo sve intervale. (Radi jednoznačnosti, svi intervali gore su u samom kodu zapravo za malu vrijednost `eps` udaljeni od navedenih rubova.)

```
sage: def energies(system = {m: 1/2, L: 1, V: 25}, eps=1e-3):
....:     bcs = [c.subs(wns).subs(system) for c in boundary_conditions]
....:     # Preparing linear system  M*coefs = 0
....:     M = matrix([[(eq.lhs()-eq.rhs()).coefficient(v) for v in (A,B,G,H)]
....:                 for eq in bcs])
....:     dt = mydet(M)
....:     res = []
....:     # Energies of bound states are all solutions of det(M) = 0
....:     ints = [(eps, system[V]-eps)]
```

```
....:     while ints:
....:         int = ints.pop(0)
....:         try:
....:             e = find_root(dt, int[0], int[1])
....:             res.append(e)
....:             ints.extend([(int[0], e-eps), (e+eps, int[1])])
....:         except:
....:             pass
....:     res.sort()
....:     return res
```

Za provjeru odredit ćemo energije i odgovarajuće vrijednosti bezdimenzionalne varijable $v = kL/2$ za slučaj sa spomenute stranice na Wikipediji:

```
sage: ens = energies(system = {m: 1/2, L: 1, V: 80}); ens
[6.557920590133058, 25.767519828716722, 55.56613192980969]
```

```
sage: # compare with wikipedia values = 1.28, 2.54, 3.73
sage: [(k*L/2).subs(wns).subs({m: 1/2, L: 1, V: 80}).subs(E=en).n()
....:      for en in ens]
[1.28042186311124, 2.53808588451596, 3.72713468799458]
```

Sada definiramo funkciju koja za datu energiju određuje koeficijente valnih funkcija A, B, G, H. Sustav jednadžbi rubnih uvjeta nije dovoljno određen ($\det M = 0$!). Dodatni uvjet koji bi potpuno odredio sustav je uvjet normalizacije valnih funkcija (integral gustoće vjerojatnosti $|\psi(x)|^2$ po cijelom prostoru mora biti jedan), ali dodavanje tog uvjeta bi učinilo jednadžbe nelinearnima pa ćemo raditi na način da privremeno fiksiramo normalizaciju valne funkcije uvjetom $\psi(L//2) = 1$, što nam fiksira vrijednost koeficijenta H i onda za preostale koeficijente rješavamo nehomogenu linearnu matričnu jednadžbu

$$M_{\text{red}} \begin{pmatrix} A \\ B \\ G \end{pmatrix} = F_{\text{red}}$$

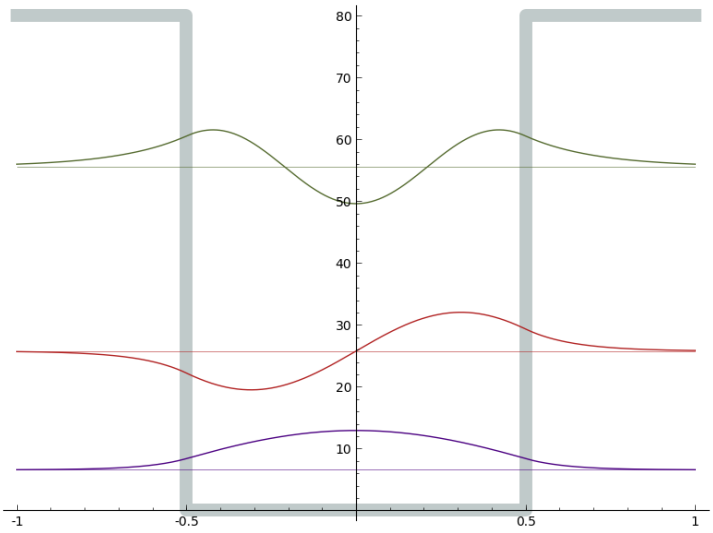
```
sage: def coefs(en, system = {m: 1/2, L: 1, V: 25}):
....:     """Returns (A, B, G, H) for given energy."""
....:     # Fixing H by normalization psi(L/2)=1
....:     hval = numerical_approx(exp(a*L/2).subs(wns).subs(system).subs(
....:                 {E: en}))
....:     bcred = [c.subs(H=hval).subs(wns).subs(system).subs({E: en})
....:                                       for c in boundary_conditions]
....:     # Preparing linear system  Mred * coefs[:-1] - Fred = 0
....:     Mred = matrix([[(eq.lhs()-eq.rhs()).coeff(v) for v in (A,B,G)]
....:                                       for eq in bcred])
....:     Fred = vector([(eq.lhs()-eq.rhs()).subs({A:0, B:0, G:0})
....:                                       for eq in bcred])
....:     cfs = la.lstsq(Mred, -Fred)[0].tolist()
....:     cfs.append(hval)
....:     return tuple(cfs)
```

Sada definiramo funkciju koja kombinira gornje funkcije, normalizira dobivene valne funkcije i vraća listu [(E_1, ψ_1), (E_2, ψ_2), ...] gdje su $\psi_i(x)$ funkcije jednog argumenta (x) dobivene parcijalnim izvrijednjavanjem (cf. *currying*) na početku definirane funkcije `psi` za različite vrijednosti njenih opcionalnih argumenata, pomoću modula `functools`. (Normalizirane funkcije još množimo factorom `scale` radi čitlijvijeg crtanja.)

```
sage: def sqwell(system = {m: 1/2, L: 1, V: 25}, scale=5.):
....:     ens = energies(system)
....:     print "Energies = %s" % str(ens)
....:     res = []
....:     for en in ens:
....:         cfs = coefs(en, system)
....:         #print cfs
....:         norm = 1/sqrt(numerical_integral(lambda x: psi(x, cfs,
....:             (k.subs(wns).subs(system).subs(E=en), a.subs(wns).subs(
....:             system).subs(E=en)), L=1)**2, (-oo, oo))[0])
....:         #print en,
....:         fun = functools.partial(psi, coefs=cfs,
....:               wns=(k.subs(wns).subs({m: 1/2, L: 1, V: 25}).subs(E=en),
....:               a.subs(wns).subs(system).subs(E=en)),
....:                         L=system[L], norm=norm*scale)
....:         #print fun(0)
....:         res.append((en, fun))
....:     return res

sage: def square_well(system = {m: 1/2, L: 1, V: 25}, span=2, scale=5.):
....:     """span = horizontal span of figure in widhts of well
....:        scale = scaling factor for wavefunctions for visibility
....:     """
....:     colors = ['indigo', 'firebrick', 'darkolivegreen', 'plum']
....:     sol = sqwell(system)
....:     P = line([[-span*system[L]/2, system[V]], [-system[L]/2,
....:           system[V]], [-system[L]/2, 0],
....:             [system[L]/2, 0], [system[L]/2, system[V]],
....:                 [span*system[L]/2, system[V]]],
....:               color='darkslategray', thickness=10, alpha=0.3)
....:     for (en, fun), k in zip(sol,range(len(sol))):
....:         P += line([[-span*system[L]/2, en], [span*system[L]/2, en]],
....:                           color=colors[k], thickness=0.4)
....:         P += plot(lambda x: fun(x, shift=en), (-span*system[L]/2,
....:                   +span*system[L]/2), color=colors[k])
....:     P.show()

sage: square_well(system = {m: 1/2, L: 1, V: 80})
Energies = [6.557920590133058, 25.767519828716722, 55.56613192980968]
```



5.5 Kozmologija

Nobelova nagrada za fiziku 2011. godine dodijeljena je Perlmutteru, Schmidtu i Riessu *"za otkriće ubrzanog širenja svemira putem opažanja dalekih supernova"*. U ovom odjeljku ćemo prikazati taj rezultat.

Udaljenost (luminosity distance) D_L (u megaparsecima Mpc) do supernove s prividnim sjajem m (mjeri se izravno) i apsolutnim sjajem M (poznat je za supernove tipa Ia) je dana formulom $\mu = m - M = 5 \log_{10} D_L + 25$. Na internet adresi http://supernova.lbl.gov/Union/ nalaze se podaci o nekoliko stotina supernovih organiziranih u tablici sa stupcima:

1. ime supernove
2. z - crveni pomak
3. μ - atenuacija sjaja supernove
4. $\Delta\mu$ - eksperimentalna neodređenost (greška) od μ

Učitavamo ih koristeći NumPy funkciju `loadtxt()` koja numeričke podatke organizirane u stupce obične datoteke pretvara u NumPy listu.

```
sage: import matplotlib.pyplot as plt
sage: import numpy as np
sage: import scipy
sage: import scipy.stats
sage: import urllib
```

```
sage: sns = urllib.urlopen(
....:  'http://supernova.lbl.gov/Union/figures/SCPUnion2_mu_vs_z.txt')
sage: dat = np.loadtxt(sns, usecols=(1,2,3)); dat.shape
(557, 3)
```

```
sage: dat[:2,:]
array([[  2.84880000e-02,   3.53355511e+01,   2.26143926e-01],
       [  5.00430000e-02,   3.66754416e+01,   1.67114252e-01]])
```

Pretvaramo sada mjerene podatke iz μ i $\Delta\mu$ u podatke za D_L i ΔD_L pazeći na ispravnu propagaciju greške.

```
sage: var('mu DL')
(mu, DL)
sage: DL(mu) = DL.subs(solve(mu == 5*log(DL, base=10) + 25, DL)[0]); DL
mu |--> 1/100000*e^(1/5*mu*log(10))


sage: DLdat = np.array([(z, DL(valmu).n(), diff(DL)(valmu).n()*errmu)
....:                               for z,valmu,errmu in dat]); DLdat.shape
(557, 3)
```

Definiramo funkcije za prilagodbu, poznate iz odjeljka o *prilagodbi funkcija*, a dodajemo i dio za ocjenu dobrote prilagodba, vidi odjeljak o *testiranju hipoteze*.

```
sage: def dist(p, fun, data):
....:     return np.array([(y - fun(p, x))/err for (x,y,err) in data])
....:
sage: def chisq(p, fun, data):
....:     return sum( dist(p, fun, data)^2 )
....:
sage: def leastsqfit(fcn, init, data):
....:     """Fit function fcn, starting from init initial values to data."""
....:     p_final, cov_x, infodict, msg, ier = scipy.optimize.minpack.leastsq(
....:                                dist, init,  (fcn, data), full_output=True)
....:     parerrs = sqrt(scipy.diagonal(cov_x))
....:     for k in range(len(init)):
....:         print "p[%d] = %.3f +- %.3f" % (k, p_final[k], parerrs[k])
....:     chi2 = chisq(p_final, fcn, data)
....:     df = len(data)-len(init)
....:     print "chi^2/d.o.f = %.1f/%i  (p-value = %.4f)" % (chi2, df,
....:       scipy.stats.chisqprob(chi2, df))
....:     return list(p_final)
```

5.5.1 Određivanje H_0 iz podataka o bliskim supernovama

Za male crvene pomake, ovisnost udaljenosti i crvenog pomaka dana je Hubbleovim zakonom

$$H_0 D_L = cz$$

gdje je c brzina svjetlosti, a H_0 Hubbleova konstanta. Određujemo Hubbleovu konstantu (u jedinicama km/s/Mpc) prilagodbom Hubbleovog zakona na podskup mjerenja za koja je $z < 0.12$.

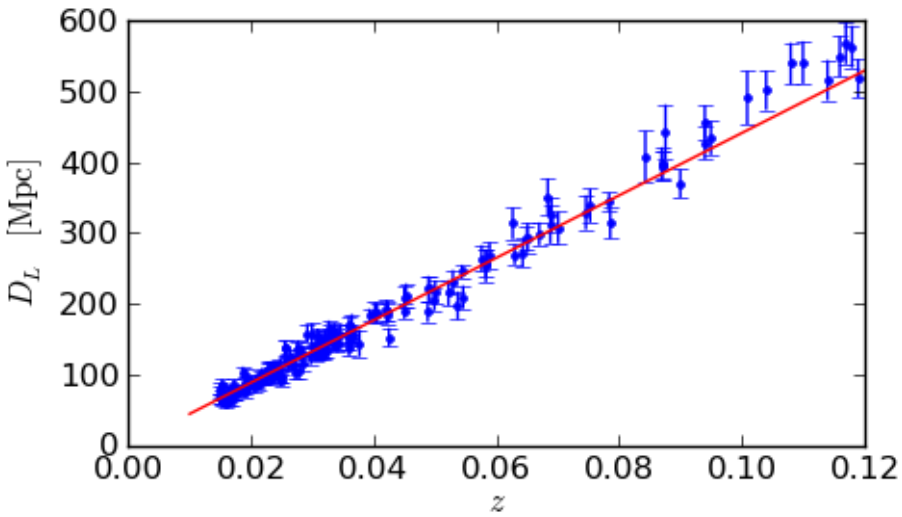
```
sage: lowz = np.array([pt for pt in DLdat if pt[0]<0.12]); lowz.shape
(174, 3)


sage: c=3.e5
sage: def linfun(p, z):
....:     return c*z/p[0]


sage: H0, = leastsqfit(linfun, (60,), lowz)
p[0] = 68.012 +- 0.385
chi^2/d.o.f = 179.1/173  (p-value = 0.3585)
sage: print "\nHubbleova constanta H0 = %.1f  km/s/Mpc" % H0
```

```
Hubbleova constanta H0 = 68.0  km/s/Mpc
```

```
sage: fig, ax = plt.subplots(figsize=[5,3])
sage: ax.errorbar(lowz[:,0], lowz[:,1], yerr=lowz[:,2], marker='.',
....:                                                   linestyle='None')
sage: zvals = np.linspace(0.01, 0.12)
sage: ax.plot(zvals, [linfun([H0], z) for z in zvals],'r-')
sage: ax.set_xlabel('$z$')
sage: ax.set_ylabel('$D_L \; {\\rm [Mpc]}$')
sage: fig.tight_layout()
sage: fig.savefig('fig')
```



5.5.2 Određivanje ubrzanja ekspanzije svemira

Za veće crvene pomake, Hubbleov zakon se modificira i u prvoj aproksimaciji se može pisati kao

$$H_0 D_L = cz\left(1 + (1 - q_0)\frac{z}{2}\right)$$

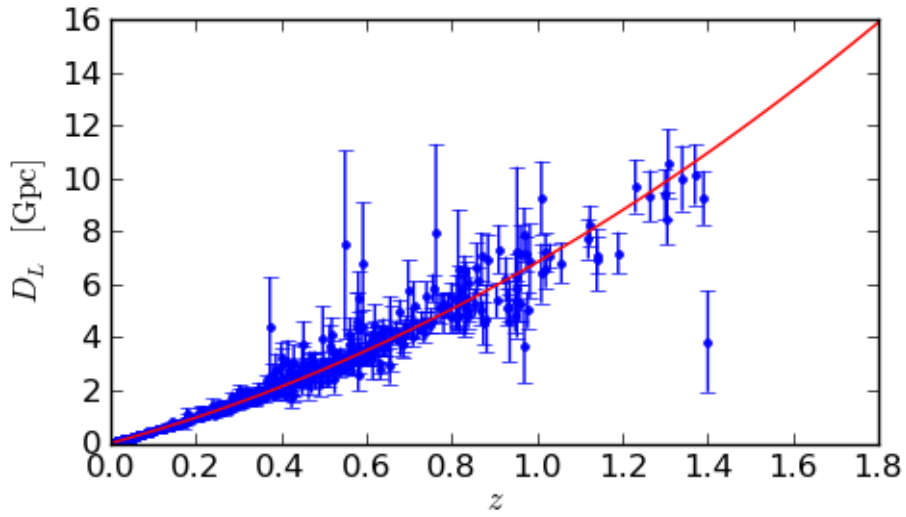
gdje je q_0 parametar deceleracije. Definiran je tako da pozitivni q_0 odgovara svemiru čija se brzina širenja usporava (kako se očekivalo prije ovih mjerenja). Određujemo taj parametar prilagodbom ove formule na cijeli skup mjerenja.

```
sage: def qfun(p, z):
....:     return c*z/p[0] * (1  +  0.5*(1 - p[1])*z)
```

```
sage: qpars = leastsqfit(qfun, (50, 0.1), DLdat)
p[0] = 69.184 +- 0.368
p[1] = -0.153 +- 0.039
chi^2/d.o.f = 553.7/555  (p-value = 0.5081)
```

```
sage: fig, ax = plt.subplots(figsize=[5,3])
sage: ax.errorbar(DLdat[:,0], DLdat[:,1]/1e3, yerr=DLdat[:,2]/1e3,
....:                                   marker='.', linestyle='None')
sage: zvals = np.linspace(0.01, 1.8)
```

```
sage: ax.plot(zvals, qfun(qpars, zvals)/1e3,'r')
sage: ax.set_xlabel('$z$')
sage: ax.set_ylabel('$D_L \; {\\rm [Gpc]}$')
sage: fig.tight_layout()
sage: fig.savefig('fig')
```



Iako iz mjernih podataka nije očito da postoji odstupanje od linearnog Hubbleovog zakona, podataka ima dovoljno da statistička analiza nedvosmisleno pokazuje da je parametar deceleracije negativan i različit od nule:

$$q_0 = -0.15 \pm 0.04$$

5.5.3 Određivanje gustoće materije i tamne energije

Gornja formula ne vrijedi za $z \approx 1$, dakle ovakav račun nije sasvim korektan i može se rabiti samo u pedagoške svrhe. Ispravan pristup zahtijeva integraciju propagacije zrake svjetlosti od supernove do promatrača kroz svemir u širenju. Ta propagacija je određena sastavom svemira (obična i tamna materija različito djeluju od tzv. tamne energije koja uzrokuje ubrzanje širenja svemira). Hubbleov zakon postaje

$$H_0 D_L = c(1+z)Z(z, \Omega_M, \Omega_\Lambda)$$

$$Z(z, \Omega_M, \Omega_\Lambda) = \int_{1/(1+z)}^{1} \frac{da}{a\sqrt{X(a, \Omega_M, \Omega_\Lambda)}}$$

$$X(a, \Omega_M, \Omega_\Lambda) = \frac{\Omega_M}{a} + \Omega_\Lambda a^2$$

$$\Omega_M + \Omega_\Lambda = 1$$

gdje je Ω_M udio materije (obične i tamne), a Ω_Λ udio tamne energije u ukupnoj energiji svemira.Još općenitije formule, koje ne rade pretpostavku $\Omega_M + \Omega_\Lambda = 1$, mogu se naći na stranicama Neda Wrighta.

Provjeravamo egzaktne formule razvojem u red po z:

```
sage: var('z a Om Ov Z X q0'); assume(z>0, a>0, Om>0, Ov>0);
(z, a, Om, Ov, Z, X, q0)
sage: X(a, Om, Ov) = Om/a + Ov*a**2 + (1-Om-Ov)
sage: DLH0 = taylor(((1+z)^2 * (Z/(1+z)) * (1 +(1-Om-Ov)*Z^2/6)).subs(
....:           Z==taylor(integral(1/(a*sqrt(X(a, Om, Ov))),
....:                             a, 1/(1+z), 1), z, 0, 2)), z, 0, 2)
sage: solve(DLH0 == z*(1  +  (1 - q0)*z/2), q0)[0]
q0 == 1/2*Om - Ov
```

Ova korespondencija s parametrom deceleracije se slaže s (Peacock 3.34).

```
sage: def J(x):
....:     if x<0:
....:         return sin(sqrt(-x))/sqrt(-x)
....:     elif x==0:
....:         return 1
....:     else:
....:         return sinh(sqrt(x))/sqrt(x)


sage: def Xfun(a, Om, Ov):
....:     return Om/a + Ov*a**2 + (1-Om-Ov)


sage: def Zfun(z, Om, Ov):
....:     return numerical_integral(lambda a: 1/(a*sqrt(X(a, Om, Ov))),
....:                                                     1/(1+z), 1)[0]


sage: def DLfun(p, z):
....:     """DL(z) with pars p=[H0, Omega_m, Omega_Lambda]."""
....:     H0 = p[0]
....:     Om = p[1]
....:     Ov = p[2]
....:     Otot = Om + Ov
....:     Zval = Zfun(z, Om, Ov)
....:     return c/H0 * (1+z) * Zval * J((1-Otot)*Zval**2)


sage: def DLfunflat(p, z):
....:     """DL(z) with pars p=[H0, Omega_Lambda]. Flat universe."""
....:     H0 = p[0]
....:     Ov = p[1]
....:     Om = 1-Ov
....:     Zval = Zfun(z, Om, Ov)
....:     return c/H0 * (1+z) * Zval
```

Prilagodbom parametara H_0, Ω_M i Ω_Λ određujemo udjele pojedinih komponenti svemira

```
sage: fullpars = leastsqfit(DLfun, (70, 0.3, 0.), DLdat) # long time
p[0] = 70.717 +- 0.466
p[1] = 0.368 +- 0.075
p[2] = 0.834 +- 0.122
chi^2/d.o.f = 527.7/554  (p-value = 0.7833)
```

Ukoliko pak fiksiramo ravnu geometriju ($\Omega_M + \Omega_\Lambda = 1$) dobivamo uobičajenu jednu trećinu materije i dvije trečine tamne energije:

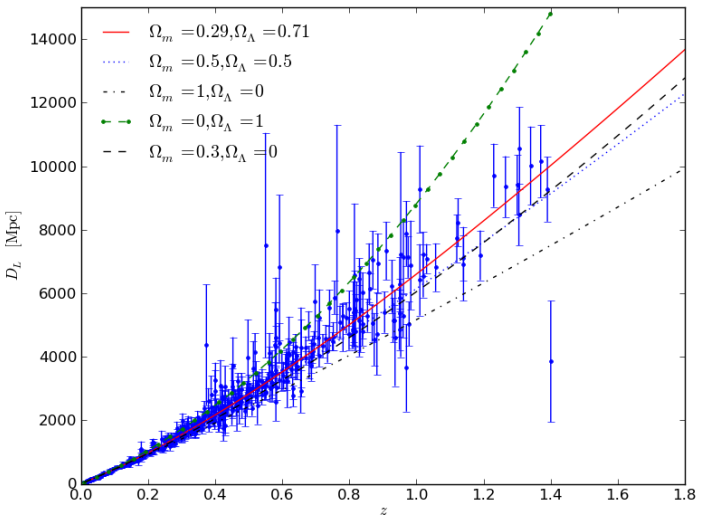
```
sage: flatpars = leastsqfit(DLfunflat, (70, 0.3), DLdat) # long time
sage: print "Om = 1-Ov = %8.3f" % (1-flatpars[1],)
p[0] = 70.412 +- 0.363
p[1] = 0.708 +- 0.021
chi^2/d.o.f = 528.7/555  (p-value = 0.7827)
Om = 1-Ov =    0.292
```

Dakle,

$$\Omega_M = 0.29$$
$$\Omega_\Lambda = 0.71$$

Crtamo ovaj rezultat (crvena linija) zajedno s nekim drugim karakterističnim scenarijima.

```
sage: fig, ax = plt.subplots(figsize=[8,6])
sage: ax.errorbar(DLdat[:,0], DLdat[:,1], yerr=DLdat[:,2], marker='.',
....:             linestyle='None')
sage: zvals = np.linspace(0.01, 1.8)
sage: ax.plot(zvals, [DLfunflat(flatpars, z) for z in zvals],'r-',
....:                 label='$\\Omega_m=0.29, \\Omega_\\Lambda=0.71$')
sage: ax.plot(zvals, [DLfun([H0, 0.5, 0.5], z) for z in zvals],'b:',
....:                     label='$\\Omega_m=0.5, \\Omega_\\Lambda=0.5$')
sage: ax.plot(zvals, [DLfun([H0, 1, 0], z) for z in zvals],'k-.',
....:                         label='$\\Omega_m=1, \\Omega_\\Lambda=0$')
sage: ax.plot(zvals, [DLfun([H0, 0, 1], z) for z in zvals],'g--.',
....:                         label='$\\Omega_m=0, \\Omega_\\Lambda=1$')
sage: ax.plot(zvals, [DLfun([H0, 0.3, 0.], z) for z in zvals],'k--',
....:                       label='$\\Omega_m=0.3, \\Omega_\\Lambda=0$')
sage: ax.set_ylim(0, 1.5e4)
sage: ax.set_xlabel('$z$')
sage: ax.set_ylabel('$D_L \; {\\rm [Mpc]}$')
sage: ax.legend(loc='upper left').draw_frame(0)
sage: fig.tight_layout()
sage: fig.savefig('fig')
```



POGLAVLJE 6

Još matematike

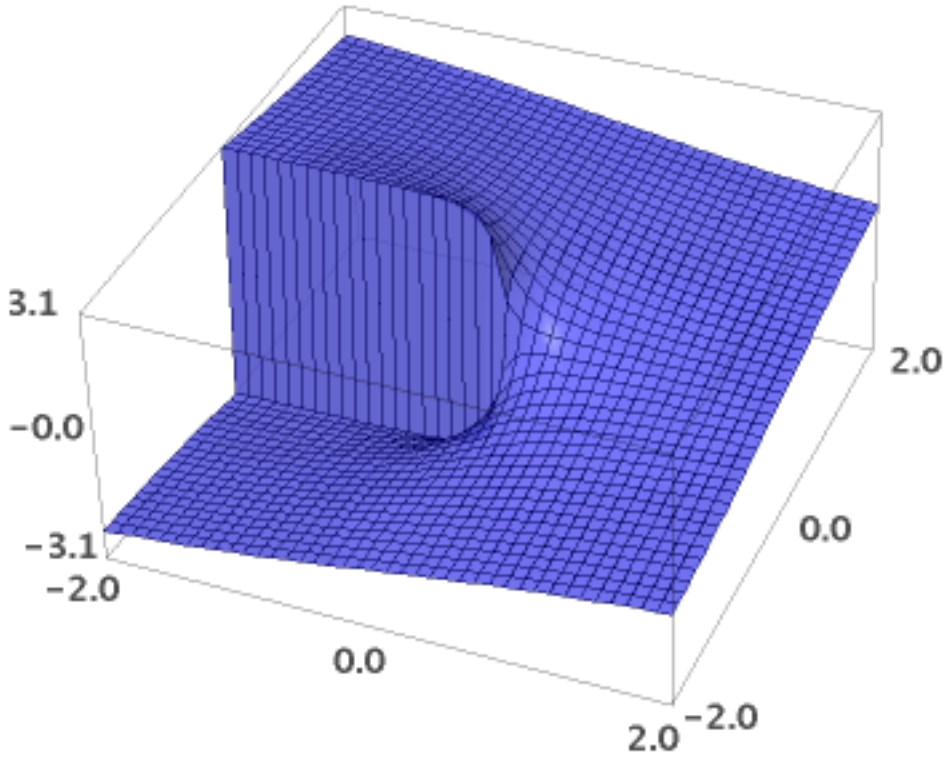
6.1 Kompleksna analiza

Matematičke funkcije se redovito ispravno ponašaju i za kompleksne vrijednosti argumenta

```
sage: log(-1+1e-8*I)
3.14159264358979*I
sage: log(-1-1e-8*I)
-3.14159264358979*I
```

Vidimo da logaritamska funkcija uredno radi s kompleksnim argumentom i daje nam glavnu granu multifunkcije s argumentom $-\pi < arg(log(z)) \leq \pi$, s diskontinuitetom od 2π prilikom prelaska negativne realne osi.

```
sage: var('x, y')
(x, y)
sage: plot3d(imag(log(x+y*1j)), (x,-2,2), (y,-2,2))
```



`taylor()` može dati i Laurentov razvoj u kompleksnoj ravnini. Npr, vrijedi

$$\frac{1}{z^2+1} = -\left(\frac{i}{2}\right)\frac{1}{z-i} - \left(\frac{i}{2}\right)^2 - \left(\frac{i}{2}\right)^3 (z-i) - \left(\frac{i}{2}\right)^4 (z-i)^2 - \cdots .$$

što do nultog reda dobivamo ovako:

```
sage: var('z')
z
sage: taylor(1/(z^2+1), z, I, 0)
-I/(2*z - 2*I) + 1/4
```

Za izračun reziduuma ne postoji posebna funkcija, ali i njega možemo dobiti pomoću funkcije `taylor()` zahvaljujući činjenici da je reziduum jednak minus prvom koeficijentu (onom uz $1/(z-z_0)$) u Laurentovom redu. Npr. tražimo reziduum funkcije $f(z) = exp(z)/z^5$ u $z = 0$:

```
sage: lrnt = taylor(e^z/z^5, z, 0, -1); lrnt
1/24/z + 1/6/z^2 + 1/2/z^3 + 1/z^4 + 1/z^5
```

Možemo koristiti i formulu s limesom u polu:

```
sage: [limit(diff(z^k*e^z/z^5, z, k-1), z=0)/factorial(k-1) for k in
....:                                                       range(1,6)]
[+Infinity, -Infinity, +Infinity, -Infinity, 1/24]
```

Obje metode ispravno daju 1/24 za traženu vrijednost reziduuma.

Zadatak 1

Definirajte funkciju `plotpoint(z)` koja crta kompleksnu ravninu sa točkom koja odgovara kompleksnom broju z. Dakle x=Re(z), y=Im(z)

Zadatak 2

Definirajte funkciju `argand(lista)` koja crta kompleksnu ravninu s točkama zadanim u listi kompleksnih brojeva `lista = [z1, z2, ...]`. Upotrijebite je da nacrtate na jednom dijagramu svih devet devetih korijena od 1.

Zadatak 3

Definirajte funkciju `plotfun(fun, min, max)` koja crta realni i imaginarni dio funkcije `fun` za interval apscise `(min, max)`.

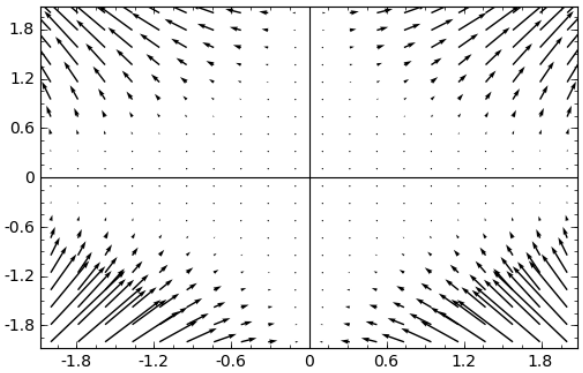
6.2 Vektorska polja

Neka se čestica u ravnini nalazi u potencijalu $U(x,y) = -4x^2y^3$. Odredimo odgovarajuću silu $\mathbf{F} = -\nabla U$ To je moguće pomoću metode simboličkih izraza `gradient()` koja vraća odgovarajuće vektorsko polje sile:

```
sage: var('x y z')
(x, y, z)
sage: U = -4*x^2*y^3
sage: F = -U.gradient(); F
(8*x*y^3, 12*x^2*y^2)
```

Dakle, polje sile je $\mathbf{F} = 8xy^3\,\mathbf{i} + 12x^2y^2\,\mathbf{j}$ i možemo ga skicirati funkcijom `plot_vector_field()`:

```
sage: plot_vector_field(F, (x,-2,2), (y,-2,2))
```



Ukoliko potencijal ovisi i o nekim parametrima moramo eksplicitno deklarirati varijable koordinatnog sustava. Isto trebamo i kad potencijal ne ovisi o nekoj koordinati. Npr. ukoliko je gornji potencijal zapravo $U(x,y,z)$, dakle u 3D prostoru samo što ne ovisi o z, onda imamo:

```
sage: F3 = -U.gradient([x,y,z]); F3
(8*x*y^3, 12*x^2*y^2, 0)
```

Sage nema specijalizirane funkcije za divergenciju i rotaciju, ali nije ih teško napisati. Jedan primjer nalazi se u datoteci dostupnoj ovako:

```
sage: load('http://www.phy.hr/~kkumer/sp/divcurl.py')
--- Loaded functions: div, curl
```

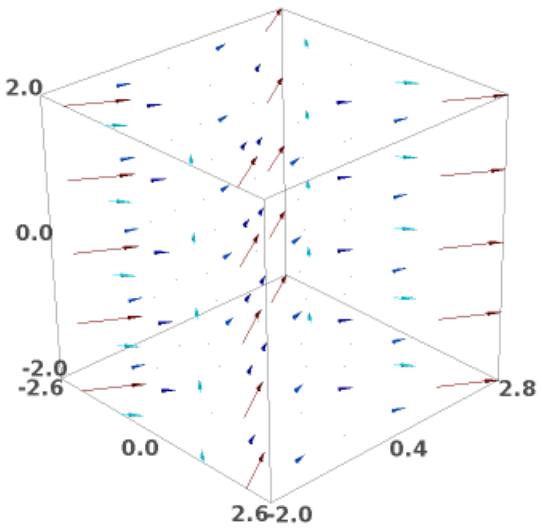
```
sage: div(F3, [x,y,z])
24*x^2*y + 8*y^3
sage: curl(F3, [x,y,z])
(0, 0, 0)
```

Provjerimo da je divergencija rotacije proizvoljne funkcije nula:

```
sage: div(curl([x^2 + 2*y, x^3 + sin(z), y*z + 2], [x,y,z]), [x,y,z])
0
```

Crtanje 3D vektorskog polja:

```
sage: plot_vector_field3d(F3, (x,-2,2), (y,-2,2), (z,-2,2))
```



6.3 Testiranje hipoteze

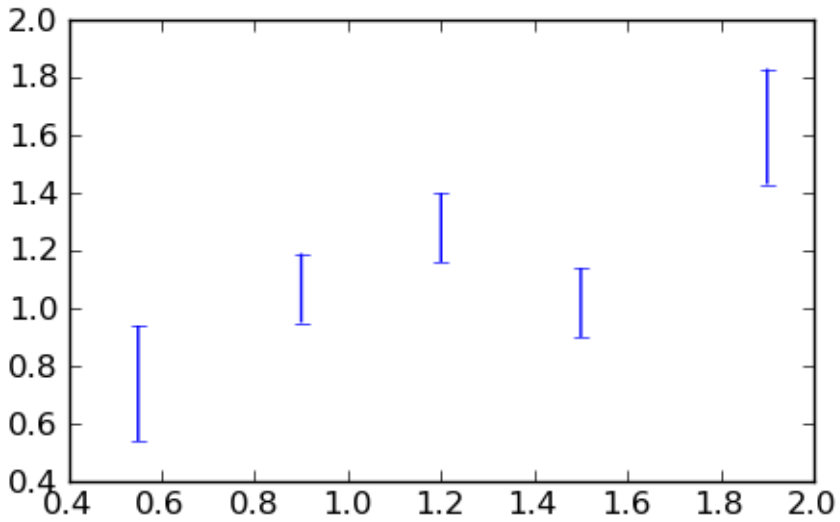
Često je potrebno kvantificirati dobrotu neke prilagodbe te odrediti da li predloženi teorijski model dobro opisuje mjerenja. Uzmimo slijedeći niz mjerenja s greškama $(x_i, y_i, \sigma_i \equiv \Delta y_i)$:

```
sage: errdata = [[0.55, 0.74, 0.20], [0.9, 1.07, 0.12],
....:       [1.2, 1.28, 0.12], [1.5, 1.02, 0.12], [1.9, 1.63, 0.20]]
sage: xs = [x for x,y,err in errdata]
sage: ys = [y for x,y,err in errdata]
sage: errs = [err for x,y,err in errdata]
```

Za crtanje točaka s greškama (*errorbars*), koristimo Matplotlib funkciju `errorbar()` gdje greške idu u opcionalni argument `yerr`:

```
sage: import matplotlib.pyplot as plt
sage: import numpy as np
sage: fig, ax = plt.subplots(figsize=[5,3])


sage: ax.errorbar(xs, ys, yerr=errs, linestyle='None')
sage: fig.savefig('fig')
```

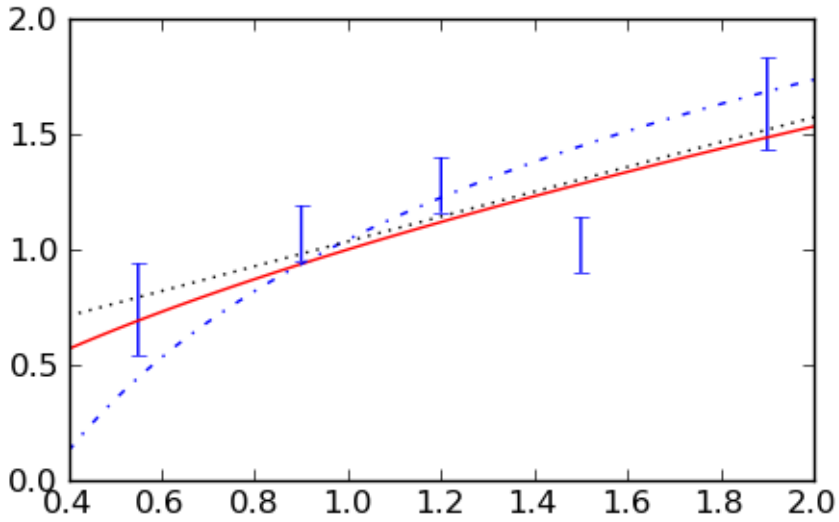


Promotrimo tri moguća teorijska opisa ovih mjerenja: potenciju x^a, logaritam $\log(ax)$ i obični pravac $a + bx$ i odredimo parametre prilagodbom pomoću funkcije *find_fit()*. (Zasad zanemarimo različite greške pojedinih mjerenja.)

```
sage: var('a, b')
(a, b)
sage: powmodel(x) = x^a
sage: logmodel(x) = log(a*x)
sage: linmodel(x) = a*x+b


sage: powfit=find_fit([[x, y] for x,y,err in errdata], powmodel,
....:                                 solution_dict=True); print powfit
{a: 0.6166290268412898}
sage: logfit=find_fit([[x, y] for x,y,err in errdata], logmodel,
....:                                 solution_dict=True); print logfit
{a: 2.836898540272243}
sage: linfit=find_fit([[x, y] for x,y,err in errdata], linmodel,
....:                                       solution_dict=True); linfit
{b: 0.49690475972372317, a: 0.5380952396683008}


sage: xvals = np.linspace(0.4, 2)
sage: ax.plot(xvals, [powmodel(x).subs(powfit).n() for x in xvals],'r')
sage: ax.plot(xvals, [logmodel(x).subs(logfit).n() for x in xvals],'b-.')
sage: ax.plot(xvals, [linmodel(x).subs(linfit).n() for x in xvals],
....:                                       color='black', linestyle=':')
sage: fig.savefig('fig')
```



Teško je "od oka" procijeniti koji je opis najbolji, a još teže koji opisi su prihvatljivi a koji nisu. Standardna mjera dobrote fita je

$$\chi^2 = \sum_i \frac{(y_i - f(x_i))^2}{\sigma_i^2}$$

gdje su σ_i greške mjerenja. (Usput, kad je riječ o mjerenju frekvencija onda je greška dana kao $\sigma_i = \sqrt{y_i}$.). Dobre su prilagodbe s $\chi^2 \approx df$, gdje je df broj stupnjeva slobode (broj mjerenja minus broj slobodnih parametara funkcije koju prilagođavamo). Za točnije kvantificiranje dobrote prilagodbe gleda se integral statističke χ^2 raspodjele s df stupnjeva slobode od izračunatog χ^2 do beskonačnosti koji se često zove p-vrijednost i koji ne bi trebao biti manji od 0.1 ili barem 0.01 ili hipotezu treba odbaciti. Taj integral je implementiran kao funkcija `scipy.stats.chisqprob(chisq, df)`. (Inače, za razliku od ovog primjera gdje tražimo statističku podršku hipotezi da funkcije opisuju mjerenja, često smo u obrnutoj situaciji gdje testiramo tzv. nul-hipotezu. Tu se npr. pitamo koja je vjerojatnost nekih mjerenja ukoliko neka čestica ne postoji ili ukoliko neki neki lijek ne djeljuje. Tada je mala p-vrijednost "poželjna" jer predstavlja statističku podršku postojanju te čestice ili dobrom djelovanju lijeka.)

Definirajmo tri funkcije koje prilagođavamo ...

```
sage: def powfun(p, x):
....:     return x^p[0]
....:
sage: def logfun(p, x):
....:     return log(p[0]*x)
....:
sage: def linfun(p, x):
....:     return p[1]*x+p[0]
```

... i odgovarajuću funkciju udaljenosti tj. odstupanja (čije kvadrate će minimizirati `leastsq()`). Tu trebamo staviti i težinski faktor $1/\sigma_i$ za svaku točku kako bi točke s manjom greškom više utjecale na prilagodbu. Ovdje ćemo dodati još jedan argument putem kojeg ćemo prosljeđivati jednu od gornje tri funkcije:

```
sage: def dist(p, fun, data):
....:     return np.array([(y - fun(p, x))/err for (x,y,err) in data])
```

Definirajmo i odgovarajući χ^2:

```
sage: def chisq(p, fun, data):
....:     return sum( dist(p, fun, data)^2 )
```

Nakon minimizacije ćemo testirati i zastavicu `ier` tako da ukoliko `leastsq()` nije uspješna dobijemo odgovarajuću poruku o grešci:

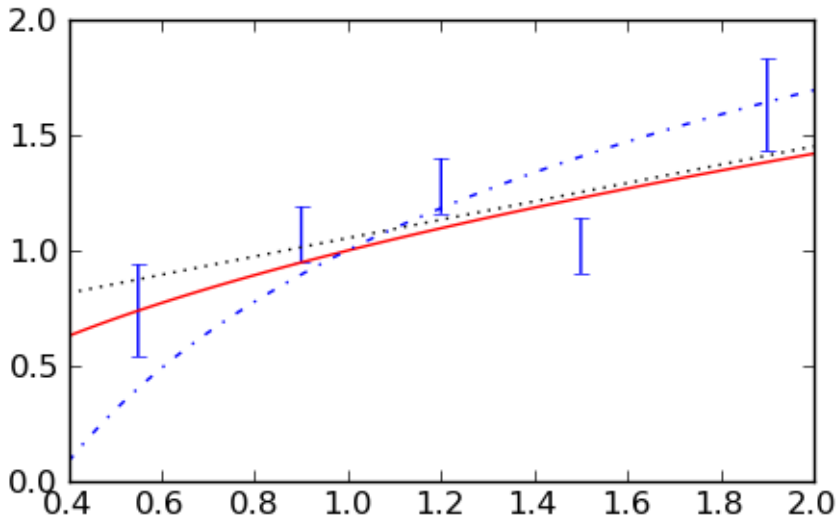
```
sage: import scipy
sage: import scipy.stats
sage: fitfun = powfun
sage: ppow_final, cov_x, infodict, msg, ier = scipy.optimize.minpack.leastsq(
....:                 dist, [1.],  (fitfun, errdata), full_output=True)
sage: if ier not in (1,2,3,4):
....:     print " -----> No fit! <--------"
....:     print msg
....: else:
....:     print "a =%8.4f +- %.4f" % (ppow_final, sqrt(cov_x[0,0]))
....:     chi2 = chisq([ppow_final], fitfun, errdata)
....:     df = len(errdata)-1
....:     print "chi^2 = %.3f" % chi2
....:     print "df = %i" % df
....:     print "p-vrijednost =  %.4f" % scipy.stats.chisqprob(chi2, df)
a =  0.5055 +- 0.1486
chi^2 = 7.880
df = 4
p-vrijednost =  0.0961
```

```
sage: fitfun = logfun
sage: plog_final, cov_x, infodict, msg, ier = scipy.optimize.minpack.leastsq(
....:                dist, [1.],  (fitfun, errdata), full_output=True)
sage: if ier not in (1,2,3,4):
....:     print " -----> No fit! <--------"
....:     print msg
....: else:
....:     print "a =%8.4f +- %.4f" % (plog_final, sqrt(cov_x[0,0]))
....:     chi2 = chisq([plog_final], fitfun, errdata)
....:     df = len(errdata)-1
....:     print "chi^2 = %.3f" % chi2
....:     print "df = %i" % df
....:     print "p-vrijednost =  %.4f" % scipy.stats.chisqprob(chi2, df)
a =  2.7219 +- 0.1693
chi^2 = 15.973
df = 4
p-vrijednost =  0.0031
```

```
sage: fitfun = linfun
sage: plin_final, cov_x, infodict, msg, ier = scipy.optimize.minpack.leastsq(
....:           dist, [1., 1.],  (fitfun, errdata), full_output=True)
sage: if ier not in (1,2,3,4):
....:     print " -----> No fit! <--------"
....:     print msg
....: else:
....:     parerrs = sqrt(scipy.diagonal(cov_x))
....:     print "p[0] =%8.3f +- %.4f" % (plin_final[0], parerrs[0])
....:     print "p[1] =%8.3f +- %.4f" % (plin_final[1], parerrs[1])
....:     chi2 = chisq(plin_final, fitfun, errdata)
....:     df = len(errdata)-2    # dva parametra
```

```
....:     print "chi^2 = %.3f" % chi2
....:     print "df = %i" % df
....:     print "p-vrijednost =  %.4f" % scipy.stats.chisqprob(chi2, df)
p[0] =   0.656 +- 0.2121
p[1] =   0.398 +- 0.1683
chi^2 = 7.116
df = 3
p-vrijednost =  0.0683
```

```
sage: fig, ax = plt.subplots(figsize=[5,3])
sage: ax.errorbar(xs, ys, yerr=errs, linestyle='None')
sage: xvals = np.linspace(0.4, 2)
sage: ax.plot(xvals, [powfun([ppow_final], x) for x in xvals],'r')
sage: ax.plot(xvals, [logfun([plog_final], x) for x in xvals],'b-.')
sage: ax.plot(xvals, [linfun(plin_final, x) for x in xvals], color='black',
....:             linestyle=':')
sage: fig.savefig('fig')
```



Ova analiza sugerira da možemo odbaciti logaritamski model, dok su druga dva modela, u nedostaku boljih, prihvatljiva.

Zadatak 4

Prilagodite funkciju $f(x) = ax$ donjim podacima. Odredite parametar a i njegovu grešku te ocijenite dobrotu fita.

```
sage: xs2, ys2, errs2 = [range(2,25,2), [5.3,14.4,20.7,30.1,35.0,41.3,52.7,
....:                 55.7,63.0,72.1,80.5,87.9], [1.5 for k in range(2,25,2)]]
sage: errdata2 = zip(xs2, ys2, errs2)
```

POGLAVLJE 7

Još programiranja

7.1 Mogućnosti ispisa rezultata:

Sage notebook sučelje može ispisati rezultat računa u različitim formatima:

1. Defaultni ispis jedini omogućuje izravni copy/paste u input ćelije.

```
sage: var('x n theta')
(x, n, theta)
sage: s1 =sum(sin(theta+x)/log(n+x), x, 1, oo)
sage: s1
sum(sin(theta + x)/log(n + x), x, 1, +Infinity)


sage: s2 = matrix(ZZ, [[1, 2, 3], [4, 5, 6]])
sage: s2
[1 2 3]
[4 5 6]
```

2. Ljepši ispis koji koristi interni LaTeX i tzv. jsMath dobiva se funkcijom `show()`, ili aktiviranjem "Typeset" gumba na vrhu notebook sučelja.

```
sage: show(s1)
sage: show(s2)
```

$$\mathrm{sum}\left(\frac{\sin\left(\theta+x\right)}{\log\left(n+x\right)}, x, 1, +\infty\right)$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

3. Ispis odgovarajućeg koda za copy/paste u LaTeX dokument.

```
sage: latex(s2)
\left(\begin{array}{rrr}
1 & 2 & 3 \\
4 & 5 & 6
\end{array}\right)
```

4. Prikaz LaTeX->PDF inaćice u posebnom prozoru (Moguće je da ovo radi samo ukoliko su Sage klijent i server na istoj mašini ili uz pažljivo podešene X-windows autorizacije.)

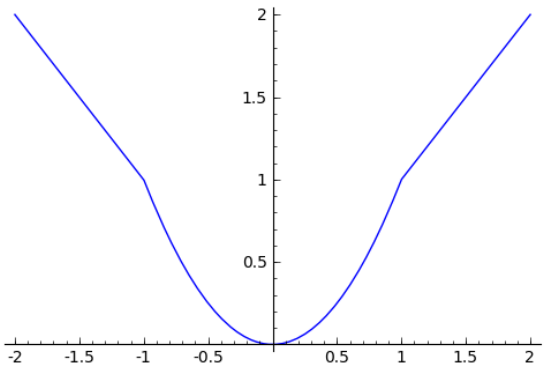
```
sage: view(s2, viewer='pdf')
```

7.2 Još o funkcijama

Funkciju definiranu po dijelovima možemo dobiti na slijedeći način:

```
sage: def fpw(x):
....:     if x<-1:
....:         return -x
....:     elif x>1:
....:         return x
....:     else:
....:         return x^2
```

```
sage: plot(fpw, -2, 2)
```



Python funkcije ne možemo općenito npr. simbolički integrirati. Posebno je opasno to što, ukoliko pokušamo, možemo dobiti pogrešan odgovor:

```
sage: integral(fpw(x), x, 1, 2)
7/3
```

No, uvijek možemo pribjeći numeričkoj integraciji koja u ovom slučaju daje točan rezultat 3/2:

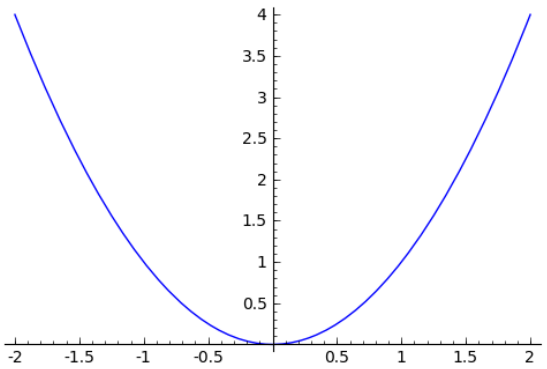
```
sage: numerical_integral(fpw, 1, 2)[0]
1.5
```

Pri gornjem pokušaju simboličke integracije, prvo je izvrijednjen sam integrand `fpw(x)`. Kako u tom trenutku `x` nema nikakvu vrijednost, on ne zadovoljava ni jedan od dva uvjeta (`x>1` ili `x<1`) pa funkcija vraća simbolički izraz `x^2` (is `else` bloka), što je pogrešno za integraciju u granicama od 1 do 2.

```
sage: fpw(x)
x^2
```

Iz istog razloga je prilikom crtanja gore bilo potrebno kao prvi argument staviti samu funkciju `fpw`, a ne izraz `fpw(x)` koji bi dao krivi crtež:

```
sage: plot(fpw(x), x, -2, 2)
```



Naravno, ukoliko znamo da python funkcija uvijek vraća korektan simbolički izraz, smijemo je integrirati i diferencirati:

```
sage: def gun(f, x):
....:     return f(f(x))
sage: diff(gun(sin, x), x)
cos(x)*cos(sin(x))
```

7.3 Funkcionalno programiranje

Isti matematički problem je često moguće riješiti na različite načine, putem algoritama iza kojih stoje sasvim različiti načini razmišljanja. Razložimo to na primjeru funkcije faktorijel.

```
sage: factorial(7)
5040
```

Klasični način programiranja, upotrebljavan od dana prvih računala, je tzv. proceduralno (ili imperativno) programiranje kod kojeg na program gledamo kao na niz naredbi koje obično postupno mijenjaju vrijednosti nekih varijabli pohranjenih u memoriji računala:

```
sage: def factorialProc(n):
....:     fac = 1
....:     i = 1
....:     while i<n:
....:         i = i + 1
....:         fac = fac * i
....:     return fac
sage: factorialProc(7)
5040
```

Ovdje je očito riječ o standardnom algoritmu kakvog bismo isprogramirali u bilo kojem proceduralnom jeziku poput Fortrana ili C-a: Imamo petlju koja se prolazi n puta i svaki puta množimo rezultat prošlog prolaza sa za 1 većim brojem.

Međutim, postoje i drugačiji pristupi. Tako je kod funkcionalnog programiranja naglasak na izvrijednjavanju funkcija tj. primjeni funkcija na izraze. Faktorijel prirodno zamišljamo kao funkciju koja daje "umnožak svih brojeva od 1 do zadanog broja".

Kao prvo, treba nam modul `operator` koji implementira funkcije koje odgovaraju standardnim matematičkim operacijama *, +, -,...

```
sage: import operator
sage: (operator.mul(2,3), operator.add(2,3))
(6, 5)
```

Zatim koristimo funkciju `reduce()`, koja kumulativno primjenjuje prvi argument (koji mora biti funkcija) na elemente drugog argumenta:

```
sage: f = function('f')
sage: reduce(f, range(1,5))
f(f(f(1, 2), 3), 4)
```

```
sage: reduce(operator.mul, range(1,8))
5040
```

Primijetite da nam ovdje nisu trebale pomoćne varijable (poput `i` i `fac` iz `factorialProc`), ali nam je trebalo više memorije jer smo trebali pohraniti čitavu listu `[1, 2, ...]` koju daje `range()`.

Funkcionalno programiranje je stil programiranja koji stavlja naglasak na izvrijednjavanje izraza, a ne na sukcesivno izvršavanje komandi. Kod čistih funkcionalnih programa obično nema pomoćnih varijabli i nepotrebnih popratnih efekata izvrijednjavanja funkcija. U tom smislu funkcionalno programiranje je pomalo slično radu s tabličnim kalkulatorom (npr. MS Excel): redosljed izračunavanja ćelija je nebitan tj. očekujemo automatsku konzistenciju. Isto, vrijednosti ćelija su dane izrazima, a ne slijedovima komandi. (Misli se na Excel, a ne Sage ćelije.)

(Više informacija o funkcionalnom programiranju nudi Functional Programming FAQ ili WWW stranice Haskell funkcionalnog jezika.)

Takav stil programiranja često rabi nekoliko specijalnih funkcija (mogli bismo ih zvati "meta-funkcije") čija je uloga upravljanje primjenivanjem drugih funkcija koje dolaze kao argumenti ovih meta-funkcija. Možda najvažnija takva funkcija je `map()` koja distribuira (*mapira*) funkciju po elementima neke liste:

```
sage: f = function('f')
sage: map(f, range(4))
[f(0), f(1), f(2), f(3)]
```

Ukoliko funkcija prima više argumenata, može se mapirati na više listi:

```
sage: map(f, range(4), range(3,7))
[f(0, 3), f(1, 4), f(2, 5), f(3, 6)]
```

Recimo da sad želimo ispisati tablicu s nizom prirodnih brojeva i njihovih faktorijela. Možemo prvo postupiti tako da prvo definiramo pomoćnu funkciju koja generira jedan red tablice i onda je pomoću `map()` primijenimo na niz brojeva:

```
sage: def auxfun(k):
....:     return (k, factorial(k))
```

```
sage: map(auxfun, range(7))
[(0, 1), (1, 1), (2, 2), (3, 6), (4, 24), (5, 120), (6, 720)]
```

Međutim, baš kao i pomoćne varijable, tako ni pomoćne funkcije nisu u duhu funkcionalnog programiranja. Zato postoje tzv. *lambda-funkcije* koje su bezimene i definiraju se i upotrebljavaju na slijedeći način:

```
sage: map(lambda k: (k, factorial(k)), range(7))
[(0, 1), (1, 1), (2, 2), (3, 6), (4, 24), (5, 120), (6, 720)]
```

```
sage: matrix(_)  # čitljiviji ispis bez uptrebe formatirajućih stringova
[  0   1]
[  1   1]
[  2   2]
[  3   6]
[  4  24]
[  5 120]
[  6 720]
```

Treba uočiti kako je često upotrebu funkcije `map()` i lambda-funkcija moguće izbjeći korištenjem *obuhvaćanja liste*. Npr, gornji primjer se elegantnije realizira ovako:

```
sage: [(k, factorial(k)) for k in range(7)]
[(0, 1), (1, 1), (2, 2), (3, 6), (4, 24), (5, 120), (6, 720)]
```

Kako su Sage/Python funkcije punopravni objekti, možemo i sami definirati funkcije koje primaju druge funkcije kao argumente. Evo funkcije koja implementira kompoziciju funkcija:

```
sage: def compose(f, n, x):
....:     """Uzastopno komponiranje funkcije sa samom sobom n puta."""
....:     if n <= 0:
....:         return x
....:     x = f(x)
....:     for i in range(n-1):
....:         x = f(x)
....:     return x
```

Npr. tzv. zlatni omjer $(\sqrt{5}+1)/2$ se može izraziti kao beskonačni razlomak

$$\frac{\sqrt{5}+1}{2} = 1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{\ldots}}}} = 1.618034\ldots$$

Takav razlomak možemo izvrijedniti na slijedeći način:

```
sage: gr = compose(lambda x: 1+1/x, 5, x); gr
1/(1/(1/(1/(1/x + 1) + 1) + 1) + 1) + 1
```

```
sage: gr.subs(x=1).n()
1.62500000000000
```

Ili, uz povećanje točnosti:

```
sage: numerical_approx((1+sqrt(5))/2)
1.61803398874989
sage: compose(lambda x: 1+1/x, 32, x).subs(x=1).n()
1.61803398874986
```

7.3.1 Primjer razvoja funkcionalnog programa

Promotrimo razvoj programa koji ispisuje tablicu frekvencija pojavljivanja elemenata u listi. Načinimo prvo jednostavnu testnu listu

```
sage: lst = ['a', 'c','b', 'a', 'c', 'a', 'd', 'b', 'c','d', 'c']
```

Metoda liste koja daje broj pojavljivanja nekog elementa je `count()`

```
sage: lst.count('c')
4
```

Da bismo ispisivali tablicu trebamo listu oblika `[(element1, frekvencija elementa1), (element2, frekvencija elementa2),...]`. Element te liste (red tablice) se može dobiti ovako:

```
sage: def el(x):
....:     return (x, lst.count(x))
sage: el('c')
('c', 4)
```

Tablicu frekvencija za različite elemente dobijemo korištenjem lambda-funkcije analogne `el()` i njenom primjenom pomoću funkcije `map()` na popis elemenata:

```
sage: map(el, ['a', 'b', 'c', 'd'])
[('a', 3), ('b', 2), ('c', 4), ('d', 2)]
```

```
sage: map(lambda x: (x, lst.count(x)), ['a', 'b', 'c', 'd'])
[('a', 3), ('b', 2), ('c', 4), ('d', 2)]
```

To je sad to, jedino još želimo izbjeći da sami moramo ručno identificirati koji se sve elementi pojavljuju u listi. To nam može raditi funkcija `uniq()` koja izbacuje duplikate iz liste:

```
sage: res = map(lambda x: (x, lst.count(x)), uniq(lst)); res
[('a', 3), ('b', 2), ('c', 4), ('d', 2)]
```

Kao zadnju stvar, poželjno je listu sortirati po frekvencijama. Funkcija `sorted()` je već definirana tako da zna sortirati liste različitih objekata, no ovdje bi po defaultu sortirala parove po prvom elementu i to po abecednom redu, ...

```
sage: sorted(res)
[('a', 3), ('b', 2), ('c', 4), ('d', 2)]
```

... dok nama treba sortiranje po drugom elementu i to po veličini. Stoga moramo putem opcionalnog argumenta `key` funkciji `sorted()` proslijediti funkciju koja će vraćati onaj dio elementa liste po kojem želimo sortiranje:

```
sage: def keyfun(item):
....:     """Return last element of item."""
....:     return item[-1]
```

```
sage: sorted(res, key=keyfun, reverse=True)
[('c', 4), ('a', 3), ('b', 2), ('d', 2)]
```

(Opcionalni argument reverse daje silazno sortiranje umjesto defaultnog uzlaznog.)

Naravno, i ovu pomoćnu funkciju `keyfun()` možemo zamijeniti lambda funkcijom:

```
sage: sorted(res, key=lambda it: it[-1], reverse=True)
[('c', 4), ('a', 3), ('b', 2), ('d', 2)]
```

I to je to. Sad definiramo kompletnu funkciju:

```
sage: def frequencies(lst):
....:     """Elements of list lst with their frequencies."""
....:     return sorted(map(lambda x: (x, lst.count(x)), set(lst)),
....:            key=lambda it: it[-1], reverse=True)
```

```
sage: for it in frequencies(lst):
....:     print "%s: %d" % it
c: 4
a: 3
b: 2
d: 2
```

Primijenimo to na frekvenciju pojavljivanja slova u Shakespeareovom Mletačkom trgovcu (zapravo koristimo engleski original The Merchant of Venice, sa WWW stranica projekta Gutenberg)

```
sage: import urllib
sage: venice = urllib.urlopen(
....:   'http://www.gutenberg.org/ebooks/2243.txt.utf-8').read()[16400:]
```

```
sage: len(venice)
120784
```

```
sage: print venice[:370]
The Merchant of Venice

Actus primus.

Enter Anthonio, Salarino, and Salanio.

  Anthonio. In sooth I know not why I am so sad,
It wearies me: you say it wearies you;
But how I caught it, found it, or came by it,
What stuffe 'tis made of, whereof it is borne,
I am to learne: and such a Want-wit sadnesse makes of
mee,
That I haue much ado to know my selfe
```

```
sage: for it in frequencies(venice)[:10]:
....:     print "%s:  %d" % it
 :  20927
e:  12250
o:  7260
t:  7227
a:  6233
h:  5560
n:  5518
s:  5317
r:  5232
i:  5206
```

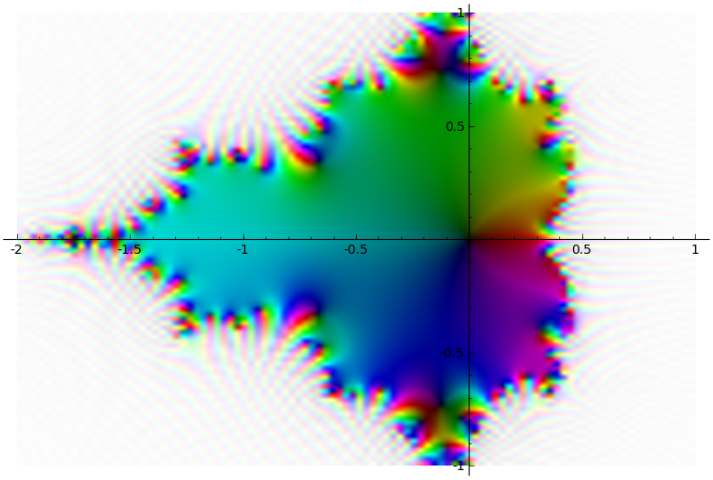
Vidimo da je "e" daleko najčešće slovo (poslije razmaka), činjenica vrlo važna pri dešifriranju engleskih tekstova.

Kao netrivijalniji primjer upotrebe gore definirane funkcije `compose()` možemo nacrtati tzv. Mandelbrotov skup, definiran kao skup svih točaka c kompleksne ravnine za koje iteracija

$$z_0 = 0\,; \qquad z_{n+1} = z_n^2 + c$$

ne divergira.

```
sage: complex_plot(compose(lambda z: z^2+x, 8, x), (-2, 1), (-1, 1))
```



(Analizirajte sami kako radi ovaj "program".)

Zadatak 1

Definirajte funkciju `allequal()` koja testira da li su svi elementi neke liste jednaki i iskoristite je da pronađete neprekidni niz od 6 istih znamenki u prvih 1000 decimala broja π. (Za pretvaranje broja u listu znamenaka možete iskoristiti funkciju `str()`.) Koristite ili obuhvaćanje liste ili `map()` tj. nije dopušteno korištenje petlji osim eventualno unutar funkcije `allequal()`, gdje to isto nije nužno (*Naputak*: `all()`).

Zadatak 2

Logističko preslikavanje je dano iteracijskom formulom $x_{n+1} = \lambda x_n(1 - x_n)$. Za male vrijednosti parametra λ, to preslikavanje za veliki n konvergira k jednoj vrijednosti. Kad λ raste, negdje blizu $\lambda = 3$, dolazi do bifurkacije i iteracije više ne konvergiraju već skaču između dvije vrijednosti. S daljnjim rastom λ dolazi do nove bifurkacije i sustav se za veliki n ponavlja s periodom četiri, itd. Rezultat se može prikazati kao tzv. Feigenbaumov bifurkacijski dijagram (vidi sliku). Nacrtajte ga tako da prvo definirate funkciju `composelist()` koja je analogna gornjoj funkciji `compose()`, ali ne vraća samo kranji rezultat kompozicije funkcija već i listu sa svim međukoracima. Nakon toga iskoristite tu funkciju za iteriranje logističkog preslikavanja. Eliminirajte prvi dio liste (dok se iteracije ne stabiliziraju) i nacrtajte drugi dio pomoću `list_plot(...., pointsize=1)`. Uočite da svaka vrijednost apscise `lambda` traži posebno iteriranje.

Zadatak 3

Odredite najmanji prirodni broj koji se *ne može* dobiti iz brojeva 2, 3, 4 i 5 korištenjem operacija zbrajanja, oduzimanja i množenja, a gdje se svaki broj smije koristiti najviše jednom. (Npr. 1=3-2, 2=3-5+4, ..., 6=2*3, ..., 14=4*5-3*2, ..., 30=(2+4)*5, ... *Naputak*: Od koristi se možda može pokazati već ranije korišteni modul `operator`, te funkcije `permutations()` i `combinations`.

Zadatak 4

Prim-brojevi blizanci su parovi prim-brojeva koji se razlikuju za dva, poput (3,5) ili (17,19). V. Brun je 1919. dokazao da suma recipročnih vrijednosti prim-brojeva blizanaca konvergira

$$B = \left(\frac{1}{3} + \frac{1}{5}\right) + \left(\frac{1}{5} + \frac{1}{7}\right) + \left(\frac{1}{11} + \frac{1}{13}\right) + \left(\frac{1}{17} + \frac{1}{19}\right) + \cdots = 1.902160583104$$

Ovako preciznu vrijednost je vrlo teško dobiti, no napišite funkciju `brun(n)` koja izračunava B koristeći listu od prvih n prim-brojeva. Pokušajte je isprogramirati i unutar paradigme funkcionalnog i unutar proceduralnog programiranja. Inače, zanimljivo je da je upravo računalno određivanje ove konstante ukazalo na bug u prvoj generaciji Pentium procesora.

Literatura

- Functional programming for Mathematicians
- Opsežniji tekst koji i kritizira upotrebu funkcionalnog programiranja u pythonu.
- Eseji Paula Grahama (oni koji se tiču programiranja)

POGLAVLJE 8

Alternative

Kako je Sage nastao združivanjem velikog broja već ranije postojećih matematičkih rutina, mnogi računi se mogu izvesti na više načina. U ovom poglavlju je ponovno izloženo nešto materijala iz poglavlja *Matematika*, ali s alternativnim izborom rutina.

8.1 Linearna algebra (alternativa)

U ovom odjeljku radimo iste stvari kao u odjeljku *Linearna algebra*, ali koristeći "domaće" funkcije Sagea, a ne NumPy modul.

Vektori i matrice se konstruiraju pomoću funkcija `vector()` i `matrix()` kojima kao argument dajemo listu elemenata, odnosno, u slučaju matrice, listu listi elemenata (listu redak-vektora)

```
sage: vec1 = vector([1,1,2])
sage: vec2 = vector((2,2,4))     # može i tupl
```

Množenje vektora skalarom je prirodno:

```
sage: 3*vec1
(3, 3, 6)
```

Skalarni produkt vektora se može ostvariti putem `dot_product()` metode vektora,

```
sage: vec1.dot_product(vec2)
12
```

ali i kao obično množenje:

```
sage: vec1*vec2
12
```

Vektorski produkt ide samo putem metode `cross_product()` (nema zapisa s $\times$!):

```
sage: vec1.cross_product(vec2)
(0, 0, 0)
```

Norma ("duljina") vektora:

```
sage: vec2.norm()
2*sqrt(6)
```

Množenje matrica te množenje matrice i vektora ide na prirodan način:

```
sage: mat = matrix([[1, 2, 1], [4, 3, 3], [9, 1, 7]]); mat
[1 2 1]
[4 3 3]
[9 1 7]


sage: mat*mat
[18  9 14]
[43 20 34]
[76 28 61]


sage: mat*vec1
(5, 13, 24)
```

Inverz matrice se može ostvariti putem metode `inverse()`, potenciranjem na minus-prvu potenciju ili operatorom ~:

```
sage: ~mat
[-18/7  13/7  -3/7]
[  1/7   2/7  -1/7]
[ 23/7 -17/7   5/7]


sage: ~mat*mat     # provjera
[1 0 0]
[0 1 0]
[0 0 1]


sage: mat2 = matrix([[1., 2.], [3., 4.]])
sage: mat2
[1.00000000000000 2.00000000000000]
[3.00000000000000 4.00000000000000]
```

Defaultno polje realnih brojeva nad kojim su definirane matrice u Sageu nije sasvim pogodno za numeričke račune pa je pogodno matrice s realnim koeficijentima kreirati uz eksplicitnu deklaraciju polja RDF ("real double field"):

```
sage: mat2 = matrix(RDF, [[1., 2.], [3., 4.]])
sage: mat2
[1.0 2.0]
[3.0 4.0]
```

Pristup pojedinim elementima matrice se isto izvodi indeksiranjem:

```
sage: mat[0, 0] = 0; mat
[0 2 1]
[4 3 3]
[9 1 7]
```

Zadatak 1

Za datu matricu A definiramo svojstvene vektore (eigenvectors) $\mathbf{v}$ i njima pripadajuće svojstvene vrijednosti λ (eigenvalues) kao rješenja matrične jednadžbe

$$A\mathbf{v} = \lambda\mathbf{v} \ .$$

Odredite svojstvene vrijednosti i svojstvene vektore matrice

$$\begin{pmatrix} 2.3 & 4.5 \\ 6.7 & -1.2 \end{pmatrix} ,$$

i provjerite da dobivena rješenja zaista zadovoljavaju gornju jednadžbu.

Zadatak 2

Kreirajte 3x3 matricu sa slučajnim realnim brojevima između 0 i 10. Invertirajte je i pomnožite s originalnom matricom te se uvjerite da dobijete jediničnu matricu.

Elementi vektora i matrica mogu biti i simboli:

```
sage: var('x y z')
(x, y, z)


sage: vec3 = vector([x, y, z])
sage: vec3.norm()
sqrt(x*conjugate(x) + y*conjugate(y) + z*conjugate(z))
```

Dijagonalizacija matrice A je pronalaženje njenog rastava oblika

$$A = PDP^{-1}$$

gdje je D dijagonalna matrica. To se izvodi metodom `eigenmatrix_right()` koja vraća matrice P i D:

```
sage: A = matrix(QQ, [[3, 1], [1, 3]])  # is_diagonalizable ne radi nad ZZ
sage: print A.is_diagonalizable()
True
sage: D, P = A.eigenmatrix_right(); (D, P)
(
[4 0]  [ 1  1]
[0 2], [ 1 -1]
)


sage: A == P*D*(~P) # provjera
True
```

Treba uočiti da su elementi dijagonalne matrice D i stupci matrice P upravo svojstvene vrijednosti odnosno svojstveni vektori od A.

```
sage: A.eigenvectors_right()
[(4, [
(1, 1)
], 1), (2, [
```

```
(1, -1)
], 1)]
```

Neke matrice nisu dijagonalizabilne, u slučaju čega će matrice P i D koje vraća `eigenmatrix_right()` i dalje zadovoljavati

$$AP = PD$$

ali P neće biti invertibilna:

```
sage: A = matrix(QQ, [[1, 1], [0, 1]])
sage: print A.is_diagonalizable()
False
sage: D, P = A.eigenmatrix_right(); (D, P)
(
[1 0]  [1 0]
[0 1], [0 0]
)
```

```
sage: A*P == P*D
True
```

```
sage: ~P
Traceback (most recent call last):
...
ZeroDivisionError: input matrix must be nonsingular
```

U slučajevima kad matricu nije moguće dijagonalizirati od koristi može biti i vrlo popularni rastav na singularne vrijednosti (singular value decomposition, SVD):

$$A = USV^\dagger$$

gdje su U i V unitarne, a S dijagonalna matrica. (Jedino treba imati na umu da je odgovarajuća metoda SVD() trenutno implementirana samo za matrice nad realnim RDF poljem pa je po potrebi potrebno prvo provesti konverziju matrice.)

```
sage: U, S, V = matrix(RDF, A).SVD()
```

```
sage: U*S*V.conjugate_transpose()
[                  1.0    1.0000000000000002]
[9.443400922348744e-17                    1.0]
```

Zadatak 3

Stupci matrice U u SVD rastavu su svojstveni vektori matrice $AA^\dagger$, a odgovarajuće svojstvene vrijednosti su kvadrati elemenata dijagonale matrice S. Uvjerite se u to eksplicitno na gornjem primjeru.

Svi vektori iste vrste i dimenzije su elementi apstraktnog vektorskog prostora koji je dostupan putem metode `parent()`:

```
sage: vec1 = vector([1,1,2])
sage: vecspace=vec1.parent(); vecspace
Ambient free module of rank 3 over the principal ideal domain Integer Ring
```

Ovdje je važno uočiti da je je prostor definiran na prstenu cijelih brojeva. Naravno, vektori mogu biti i nad drugim prstenovima/poljima:

```
sage: print vector([1/3, 1/4]).parent()
Vector space of dimension 2 over Rational Field
sage: vector([1., 2.]).parent()
Vector space of dimension 2 over Real Field with 53 bits of precision
```

Baza vektorskog prostora:

```
sage: vecspace.basis()
[
(1, 0, 0),
(0, 1, 0),
(0, 0, 1)
]
```

Matrice isto imaju svoje apstraktne prostore kojima pripadaju:

```
sage: print mat.parent()
Full MatrixSpace of 3 by 3 dense matrices over Integer Ring
sage: (~mat).parent()
Full MatrixSpace of 3 by 3 dense matrices over Rational Field


sage: mat2 = matrix([[1., 2.], [3., 4.]])
sage: mat2.parent()
Full MatrixSpace of 2 by 2 dense matrices over Real Field with 53 bits of precision
sage: mat2
[1.00000000000000 2.00000000000000]
[3.00000000000000 4.00000000000000]
```

Ukoliko pokušamo već stvorenom vektoru nad poljem nekih brojeva zamijeniti neki element simbolom, to ne ide:

```
sage: vec1[2] = x
Traceback (most recent call last):
...
TypeError: unable to convert x to an integer
```

Riječ je o tome da su npr. čisto realni vektori (nad poljem RDF) interno reprezentirani kao C-polja radi optimizacije. Da bismo mogli napraviti ovo što želimo trebamo prvo konvertirati vektor u simbolički vektor. Tu konverziju radi odgovarajući vektorski prostor nad simboličkim prstenom (`SR`, *symbolic ring*). Taj prostor dobijemo pomoću metode `parent()` vektora istog ranga, ali koji već jest simbolički. (Za detalje o takvim konverzijama vidi prvih par odjeljaka ovdje.)

```
sage: vec3.parent()
Vector space of dimension 3 over Symbolic Ring


sage: vec1_symb = vec3.parent()(vec1)  # konverzija vec1 u simbolicki vektor
```

```
sage: vec1_symb[1] = x; vec1_symb     # sad ide
(1, x, 2)
```

8.2 Diferencijalne jednadžbe (alternativa)

Alternativna rutina za numeričko rješavanje diferencijalnih jednadžbi je Sageov `ode_solver()`. Slično kao i za SciPyjev `odeint()` problem treba organizirati kao sustav diferencijalnih jednadžbi prvog reda oblika

$$\frac{dy_i}{dt} = f_i(y,t) \quad i = 1, 2, \ldots ,$$

te je potrebno definirati funkciju koja vraća listu $[f_0(t), f_1(t), \ldots]$, gdje je dopuštena i dodatna ovisnost o nekim konstantnim parametrima sustava. Npr. za van der Polovu jednadžbu takva funkcija se može definirati ovako:

```
sage: def syst(t, y, params):
....:     return [y[1], params[0]*(1 - y[0]^2)*y[1] - y[0]]
```

Sučelje za `ode_solver()` je više u stilu objektno-orjentiranog programiranja: ta funkcija kreira `ode_solver` objekt i sve ostalo se izvodi putem metoda tog objekta. Prilikom kreiranja možemo zadati sustav jednadžbi (definiran gornjom funkcijom).

```
sage: S = ode_solver(syst)
```

Metoda `ode_solver` objekta koja integrira diferencijalne jednadžbe je `ode_solve()`, čiji važni argumenti su početni uvjeti `y_0` = $[y_0(0), y_1(0), \ldots]$, vremenski interval evolucije sustava `t_span` = $[t_0, t_{\max}]$ i broj točaka integracije `num_points`, a ukoliko sustav ovisi i o nekim parametrima specificiramo ih opcijom `params`:

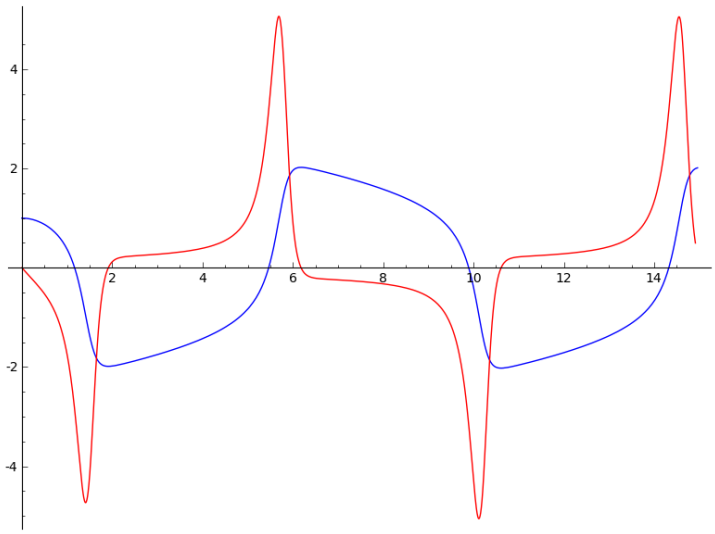
```
sage: S.ode_solve(y_0 = [1,0], t_span=[0,15], params=[3], num_points=150)
```

Rješenje je kao i u slučaju rješavanja funkcijom `odeint` lista točaka (u malo drugačijem formatu), a postoji i funkcija `interpolate_solution()` koja interpolira te točke i pretvara ih u funkciju. Opcionalni argument te funkcije `i`, određuje koju od funkcija $y_i(t)$ želimo:

```
sage: S.solution[:3]
[(0, [1, 0]), (0.1, [0.9950026686448377, -0.09990818686134167]),
(0.2, [0.9800189364740901, -0.1998574915962547])]


sage: xt = S.interpolate_solution(i=0)
sage: yt = S.interpolate_solution(i=1)


sage: plot(xt, (0,15)) + plot(yt, (0,15), color='red')
```



Jedna zanimljiva prednost `ode_solver` pristupa je da omogućuje da se izvrijednjavanje funkcija koje stoje s desne strane diferencijalnih jednadžbi preseli iz Pythona u C, pomoću tzv. *cythona*:

```
sage: %cython
sage: cimport sage.gsl.ode
sage: import sage.gsl.ode
sage: include 'gsl.pxi'
sage: cdef class van_der_pol(sage.gsl.ode.ode_system):
....:     cdef int c_f(self,double t, double *y,double *dydt):
....:         dydt[0]=y[1]
....:         dydt[1]=(1 - y[0]*y[0])*y[1] - y[0]
....:         return GSL_SUCCESS
....:     cdef int c_j(self, double t,double *y,double *dfdy,double *dfdt):
....:         dfdy[0]=0
....:         dfdy[1]=1.0
....:         dfdy[2]=-2.0*y[0]*y[1]-1.0
....:         dfdy[3]=(1 - y[0]*y[0])
....:         dfdt[0]=0
....:         dfdt[1]=0
....:         return GSL_SUCCESS
```

Ovdje smo osim funkcija $f_i(y,t)$ koje definiraju sustav, definirali i jakobijan df_i/dy_j.

```
sage: T = ode_solver()
sage: T.algorithm = "bsimp"
sage: vander = van_der_pol()
sage: T.function=vander
```

Za sustav iz ovog odjeljka ubrzanje je samo četverostruko, ali taj faktor može biti i osjetno veći za kompliciranije jednadžbe.

```
sage: %time
sage: S.ode_solve(y_0 = [1,0], t_span=[0,150], params=[1], num_points=1e5)
CPU time: 10.18 s,  Wall time: 10.65 s


sage: %time
sage: T.ode_solve(y_0 = [1,0], t_span=[0,150], num_points=1e5)
CPU time: 2.20 s,  Wall time: 2.20 s
```

Još jednu alternativu za numeričko integriranje diferencijalnih jednažbi nudi "domaća" Sageova funkcija `desolve_system_rk4()`.

8.3 Statistika (alternativa)

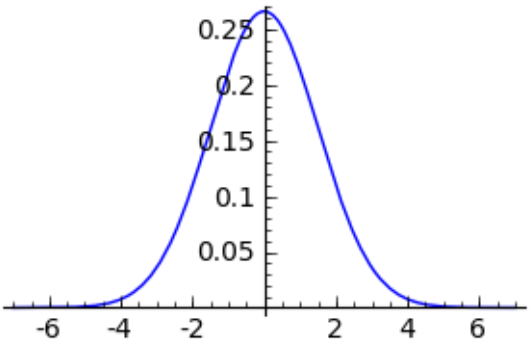
U ovom odjeljku ćemo, bez puno komentara, napraviti identične primjere kao u odjeljku *Statistika*, ali koristeći isključivo "domaće" Sage funkcije, a ne SciPy i NumPy.

```
sage: xs =[ 1. ,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  5. ,
....:           5.4,  5.8,  6.2,  6.6,  7. ,  7.4,  7.8,  8.2,  8.6,  9. ]
```

```
sage: print "srednja vrijednost = %.1f" % mean(xs)
srednja vrijednost = 5.0
sage: print "varijanca = %.3f" % variance(xs,
....:       bias=True)  # default bias=False dijeli s 1/(len(xs)-1)
varijanca = 5.867
sage: print "standardna devijacija = %.3f" % std(xs, bias=True)
standardna devijacija = 2.422
```

```
sage: G = RealDistribution('gaussian', 1.5)
sage: G.distribution_function(-5)
0.0010281859975274036
```

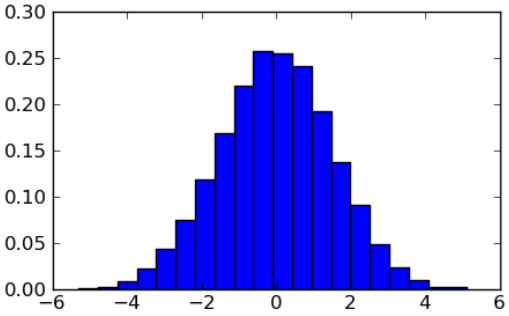
```
sage: G.plot((-7, 7), figsize=[3.5,2])
```




```
sage: G.cum_distribution_function(1.5)-G.cum_distribution_function(-1.5)
0.6826894921370859
```

```
sage: pts = [5 + G.get_random_element() for k in range(1e4)]
sage: print "broj točaka = %i" % len(pts)
broj točaka = 10000
sage: print "srednja vrijednost = %.1f" % mean(pts)  # rel tol 0.02
srednja vrijednost = 5.0
sage: print "standardna devijacija = %.3f" % std(pts) # rel tol 0.02
standardna devijacija = 1.497
```

```
sage: G.generate_histogram_plot('fig', num_samples=1e4, bins=20)
```




```
sage: [mean([5 + G.get_random_element() for k              # abs tol 0.05
....:          in range(10000)]) for k in range(5)]
[4.98977720776, 5.00428769154, 5.00979362618, 4.9901576697, 4.99635335439]


sage: CHISQ = RealDistribution('chisquared', 3)
sage: mean([CHISQ.get_random_element() for k in range(1000)])  # rel tol 0.1
3.032640511


sage: CHISQ = RealDistribution('chisquared', 1)
sage: [1-CHISQ.cum_distribution_function(eps^2) for eps in [1,2,3]]
[0.3173105078629148, 0.04550026389635842, 0.002699796063260207]
```

8.4 Prilagodba funkcije podacima (alternativa)

Alternativna mogućnost za prilagodbu funkcije podacima je poznati specijalizirani jezik za statističku analizu R, koji je sadržan u Sage distribuciji (premda je sučelje još uvijek relativno primitivno). Linearna regresija pomoću R-sučelja ide ovako:

```
sage: data = [[0.2,  0.1], [0.35,  0.2], [1, 0.6], [1.8, 0.9],
....:                                  [2.5, 1.3], [4, 2.06], [5, 2.6]]
sage: rx = r([x for x,y in data])
sage: ry = r([y for x,y in data])
sage: r.summary(r.lm(ry.tilde(rx)))

Call:
lm(formula = sage16)

Residuals:
         1          2          3          4          5          6          7
-0.0227707  0.0002708  0.0667844 -0.0436605 -0.0027998 -0.0123840  0.0145599

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 0.020160   0.023057   0.874    0.422
sage7       0.513056   0.008488  60.446 2.35e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.0381 on 5 degrees of freedom
Multiple R-squared: 0.9986, Adjusted R-squared: 0.9984
F-statistic:  3654 on 1 and 5 DF,  p-value: 2.345e-08
```

Ovdje gore dobivamo prvo `residuals`, što su odstupanja prilagođene funkcije od pojedinih točaka, zatim `coefficients`, što su parametri pravca gdje imamo i statistički test signifikantnosti parametara (zadnji stupac `Pr(>|t|)` gdje je manja brojka bolja). Na kraju, zadnji broj (`p-value`) označava dobrotu prilagodbe određenu putem tzv F-testa, gdje je prilagodba dobra ukoliko je `p-value` manji od 0.01 ili barem 0.1.

Usput, određivanje gore navedenih p-vrijednosti za parametre ide ovako:

```
sage: import scipy.stats
sage: print 2*(1-scipy.stats.t.cdf(0.02016/0.023057, 5))
0.42192862901
sage: 2*(1-scipy.stats.t.cdf(0.513056/0.008488, 5))
2.3454928665955777e-08
```