

Uporaba SAGE-a (besplatnog matematičkog software-a) u inženjerskoj nastavi

Strukan, Šimun

Undergraduate thesis / Završni rad

2012

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture / Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje**

Permanent link / Trajna poveznica: <https://urn.nsk.hr/urn:nbn:hr:235:703462>

Rights / Prava: [In copyright](#)/[Zaštićeno autorskim pravom.](#)

Download date / Datum preuzimanja: **2025-02-22**

Repository / Repozitorij:

[Repository of Faculty of Mechanical Engineering and Naval Architecture University of Zagreb](#)



SVEUČILIŠTE U ZAGERBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Voditelj rada:
dr.sc. Mario Essert

Šimun Strukan

ZAGREB, rujan 2012

SADRŽAJ

1	Uvod	4
1.1	Otvoreni kod, ukratko	4
2	Implementacija	5
2.1	Instalacija	5
2.1.1	Linux i OS X	5
2.1.2	Microsoft Windows	5
2.2	Proširenje osnovne konfiguracije Sage-a	5
2.2.1	Mathematica	7
2.2.2	Octave i Scilab	7
2.2.3	Implementacija	8
2.3	Pregled sučelja i osnovnih karakteristika	9
3	Sage paket	11
3.1	Uvod u Python	11
3.1.1	Brojevi	11
3.1.2	Nizovi	12
3.1.3	Liste	13
3.1.4	Prvi korak prema programiranju	13
3.1.5	<i>if</i> naredba	14
3.1.6	<i>for</i> naredba	14
3.1.7	Funkcija <i>range</i>	15
3.1.8	Naredbe <i>break</i> and <i>continue</i> i <i>else</i> uvijet u petlji	15
3.1.9	<i>pass</i> naredba	15
3.2	Uporaba Sage paketa	16
3.2.1	Dodjeljivanje vrijednosti, jednakosti i aritmetika	16
3.2.2	Pomoć u Sageu	18
3.2.3	Funkcije, uvlačenja i brojanje	19
4	Python programiranje u Sage okruženju	24
4.1	Objektno programiranje	24
4.1.1	Varijable, objekti i imena	24
4.1.2	Objektno orjentirana programska paradigma	25
4.2	Funkcionalno programiranje	27
4.2.1	Funkcionalno programiranje uz korištenje <i>map</i>	28
4.2.2	Definiranje malih funkcija koristeći <i>lambda</i>	28
4.2.3	Filtriranje koristeći <i>filter</i>	29
4.3	Python matematički moduli	30
4.3.1	NumPy	30
4.3.2	SciPy	32
4.3.3	Matplotlib	34

5	Latex i Sage	37
5.1	Primjena	38
6	Primjena u nastavi FSB-a i nastavi tehničke prirode	41
6.1	Riješavanje jednažbi	41
6.2	Numeričke aproksimacije	43
6.3	Deriviranje, Integriranje, itd	44
6.4	Riješavanje diferencijalnih jednažbi	45
6.5	Grafika	47
6.6	Operacije s matricama	49
6.7	Regresija	52
6.8	Interakcija	54
7	Dokumentacija i korišteni alati	56
7.1	Sphinx	56
7.1.1	Osnovna sintaksa	56
7.1.2	Izrada dokumentacija	57
8	Zaključak	60

POPIS SLIKA

2.1	Unos virtualizacijskog Sage paketa	6
2.2	Postavke paketa	6
2.3	Virtualizacijski Sage paket instaliran u VirtualBox	7
2.4	Mathematica u Sageu	8
2.5	Sage CLI sučelje	9
2.6	Sage početni pogled	10
2.7	Popis radnih stranica(eng. worksheets) u Sageu	10
3.1	Sage: Dodjelivanje vrijednosti	16
3.2	Sage: Osnovna aritmetika	17
3.3	Sage: Osnovne funkcije	17
3.4	Sage: Numerička aproksimacija	18
3.5	Sage: Korisnička funkcija	20
3.6	Sage: Greška u sintaksi	21
3.7	Sage: Ispravljena sintaksa	21
3.8	Sage: Tablični prikaz rješenja	22
3.9	Sage: Korisnička klasa sa listom brojeva	23
4.1	Sage: Ponašanje objekata	25
4.2	Sage: Primjer korisničke klase	27
4.3	Sage: Funkcija <code>filter</code>	30
4.4	Sage: Kombiniranje funkcija <code>lambda</code> i <code>filter</code>	30
4.5	Sage: Grafički prikaz rješenja	33
4.6	Sage: Korištenje Matplotlib paketa	34
4.7	Sage: Korištenje “Pie Chart” prikaza	35
4.8	Sage: Histogramski prikaz	35
4.9	Sage: 3D koordinatni sustav	36
5.1	Sage: Graf prikazan u pdf dokumentu	40
6.1	Sage: rješavanje jednostavne jednadžbe	41
6.2	Sage: sustav jednadžbi sa jednom nepoznanicom	42
6.3	Sage: sustav jednadžbi sa više nepoznanica	42
6.4	Sage: Sustav ne lineranih jednadžbi	42
6.5	Sage: numerička aproksimacija rješenja	43
6.6	Sage: Jednadžba ne riješiva funkcijom <code>solve</code>	43
6.7	Sage: Numerička aproksimacija funkcijom <code>find_root</code>	43
6.8	Sage: Jednostavna derivacija	44
6.9	Sage: Parcijalna derivacija	44
6.10	Sage: Integracija	44

6.11 Sage: Jednostavna diferencijalna jednačba	45
6.12 Sage: Laplaceova transformacija	45
6.13 Sage: Sustav opruga	45
6.14 Sage: Laplace prve jednačbe	46
6.15 Sage: Riješenje sustava opruge	46
6.16 Sage: Inverzna Laplaceova transformacija riješenja	46
6.17 Sage: Graf sinus funkcije	47
6.18 Sage: Graf riješenja sustava opruga	47
6.19 Sage: Vektorsko polje	48
6.20 Sage: Kombinirani 3D graf	48
6.21 Sage: Deformirani torus	49
6.22 Sage: operacije s matricama	51
6.23 Aproksimacijske krivulje	52
6.24 Regresija namanjeg kvadrata	52
6.25 Sage: Regresijska krivulja	53
6.26 Sage: Interaktivna matrica	54
6.27 Sage: Osnovni prikaz funkcije	55
6.28 Sage: Interaktivno promjenjena funkcija	55
7.1 Ispis HTML dokumentacije	58

Napomena: Izjavljujem da sam ovaj rad izradio samostalno sa stečenim znanjem i navedenom literaturom.
Zahvaljujem se svom mentoru na vodstvu i korisnim savjetima.

UVOD

Primarna zadaća ovog završnog rada je prikazati kako se besplatni matematički alat SAGE može integrirati u nastavu FSB-a. Razmotriti koja su područja primjene ovog alata, kratko razjasniti kako funkcionira, te kako može pridnjeliti poboljšanju kvalitete nastave svih tehničkih fakulteta, a ne samo one unutar FSB-a. Odmah treba navesti njegovu primarnu prednost, ujedno i razlog zašto je tema ovog rada, a to je da je ovaj alat besplatan. Mnogi će se pitati, kako je moguće da jedan takav alat bude besplatan? Smatram da je potrebno ukratko razjasniti taj fenomen, jer sam do sada iz iskustva zaključio da ljudi često ne vjeruju u kvalitetu takvih alata i da ime generalno nije jasan fenomen *open source* koda, koji omogućuje pojavu takvog besplatnog software-a.

1.1 Otvoreni kod, ukratko

Open source ili u doslovnom prijevodu otvoreni kod, kako mu ime kaže, je kod koji je na raspolaganju svima koji imaju volje i znanja da se bave takvim tehnologijama. Mislim da je ideja prirodno evolvirala još iz perioda Aleksandrijske knjižnice kada je postojala želja za skladištenjem i djeljenjem znanja. Ideja da se organizira jedno mjesto gdje se može pohraniti znanje koje će onda biti dostupno svima bez naplate ili troška, je po meni, stara koliko i pisana povijest. Smatram da je danas taj cilj gotovo postignut i to u obliku interneta. Ogromna interaktivna baza znanja gdje se dohvaćanje informacije mjeri u milisekundama, globalno znanje dostupno svima, neograničeno podnevljem, rasom, jezikom... Za razvoj takvog sustava, poput interneta, bila je potrebna sinergija mnogih znanstvenih i tehničkih disciplina, no u konačnici cijeli sustav počiva na programskom kodu i algoritmima koje pišu ljudi iz raznih sredina svi motivirani nekim svojim ciljevima. Dok otvoreni kod nije ideja koja je nastala u kontekstu interneta nego nešto ranije, smatram da je internet pogodovao širenju takve ideologije.

U srži ideje otvorenog koda je ta stara ideja o izmjeni znanja. Upravo je internet omogućio svakom pojedincu da brzo i efikasno dijeli svoje programske ideje i riješenja sa svijetom. Neki bi rekli da se na taj način svatko može iskoristiti ne stečenim znanjem, te se gubi smisao prodaje, konkurancije, zarade itd. No dogodilo se upravo suprotno, programerska zajednica se počela formirati u male grupice oko svojih programskih ideja, te umjesto da su "krali" jedni od drugih oni su počeli davati jedni drugima. Drugim riječima pojedinac bi objavio svoj kod, a zatim bi ga zajednica preuzela i unaprijedila. Počeli su se javljati internet servisi kao *github*, *sourceforge* i sl. Takvi servisi su poticali kolaboraciju ljudi koji se nikada nisu upoznali, te se na internetu danas mogu naći nevjerovatna programska riješenja koja evolviraju fenomenalno brzo i sve to na način da su dostupna svima, bilo za učenje, korištenje ili unaprijednje vlastitog softwarea. Ubrzo su potencijal takvog razvoja softwarea shvatile i neke tvrtke, te su preuzele filozofiju takvog programiranja. Također se na internetu formiraju grupe ljudi oko raznih projekta koji se razvijaju za internetsku populaciju, a koja takve alate može koristiti besplatno. SAGE je jedan od takvih alata. Razvijen na principima otvorenog koda, besplatan i dostupan svima. Napisan je u potpunosti u Python programskog jeziku, koji je također besplatan te je na raspolaganju svima.

IMPLEMENTACIJA

2.1 Instalacija

Sage podržava mogućnost instalacije na sve najpopularnije operacijske sustave, poput Windowsa, Mac OSa, te većinu zastupljenih Linux distribucija.

2.1.1 Linux i OS X

Instalacija iz pripremljenog binarnog paketa bi na duge staze bio najlakši i najbrži način instaliranja Sagea. U tom slučaju je instalacija vrlo jednostavna i svodi se na preuzimajne datoteke koja pri pokretanju sve operacije izvodi automatski, korisnik jedino teba odrediti lokaciju gdje će se Sage instalirati. Takav je proces poznat svim korisnicima računala jer se ne razlikuje od standardne procedure instalacije aplikacija. Treba imati na umu da je Sage sučelje za programiranje, tako da bi instalacija kompajliranjem koda dala maksimalnu fleksibilnost dugoročno gledano. U tom slučaju potrebno je imati dosta predznanja o prirodi operativnog sustva na koji se Sage instalira, također kompajliranje može trajati nekoliko dana, što ga čini dosta ne praktičnim. Iako je sa striktno tehničke strane možda bolja varijanta, ovakava se instalacija ne preporuča u većini slučajeva, a pogotovo ako korisnik ne barata dodatnim tehničkim znanjem. Potrebno je računalo s najmanje 2 GB slobodnog prostora na disku i operacijski sustav Linux (32-bitna ili 64-bitni) ili OS X (10.4 ili noviji). Također je potrebno instalirati LaTeX paket. Ako želite vidjeti animacije, trebali biste instalirati bilo ImageMagick ili FFmpeg. ImageMagick ili dvipng se koristi za prikazu nekih LaTeX izlaza u Sage okruženje. Sage instalacijski paket se može preuzeti na lokaciji: <http://www.sagemath.org/download.html>

2.1.2 Microsoft Windows

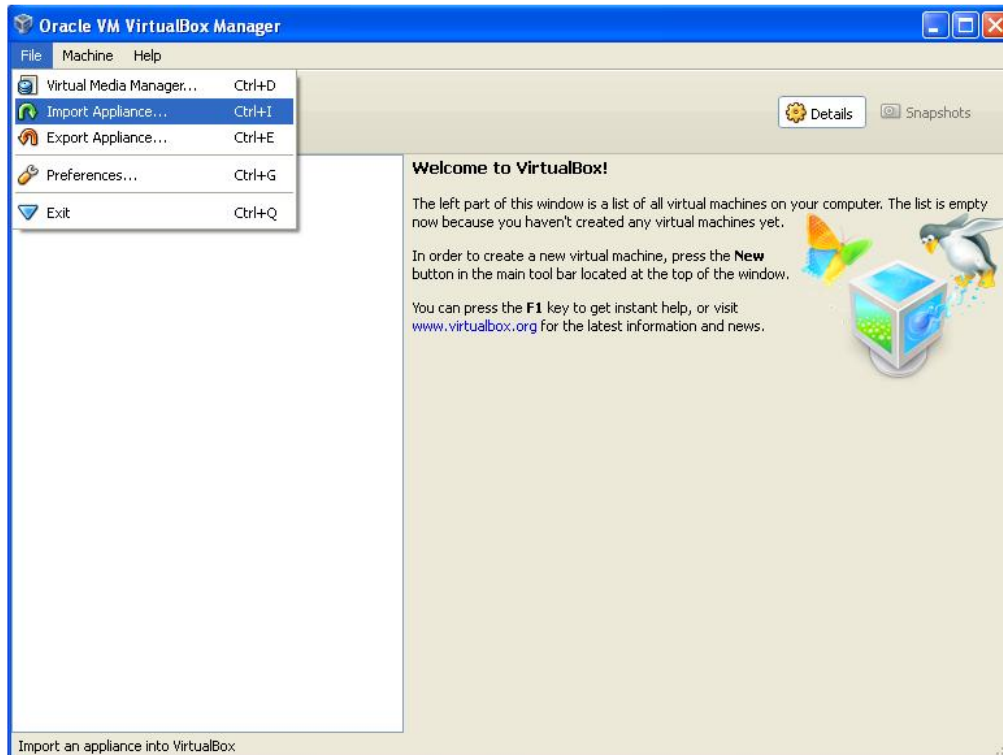
Jedini način za instalaciju Sage-a na Windows je instalirati [VirtualBox za Windows](#), a zatim skinuti i instalirati VirtualBox distribuciju Sage-a. U procesu instalacije na Windows sustav se koristi mehanizam virtualizacije koji nam omogućuje VirtualBox software. Postoje mnogi drugi alati koji nam omogućuju takvu virtualizaciju no Sage razvojni tim preporuča spomenutu aplikaciju, te su za potrebe Windows operacijskog sustava razvili dosta jednostavan proces instalacije s kojim jedan inženjer ili student tehničke struke ne bi trebao imati previše problema. Po instalaciji VirtualBox aplikacije, čiji je proces vrlo jednostavan na svim operacijskim sustavim, potrebno je učitati Virtualizacijski sage paket u VritualBox, preuzet sa Sage web stranice.

Učitavanjem sage virtualnog paketa se pokreće čarobnjak za instalaciju, te se od tog trena sve svodi na prihvaćanje ili modifikaciju Sage postavki.

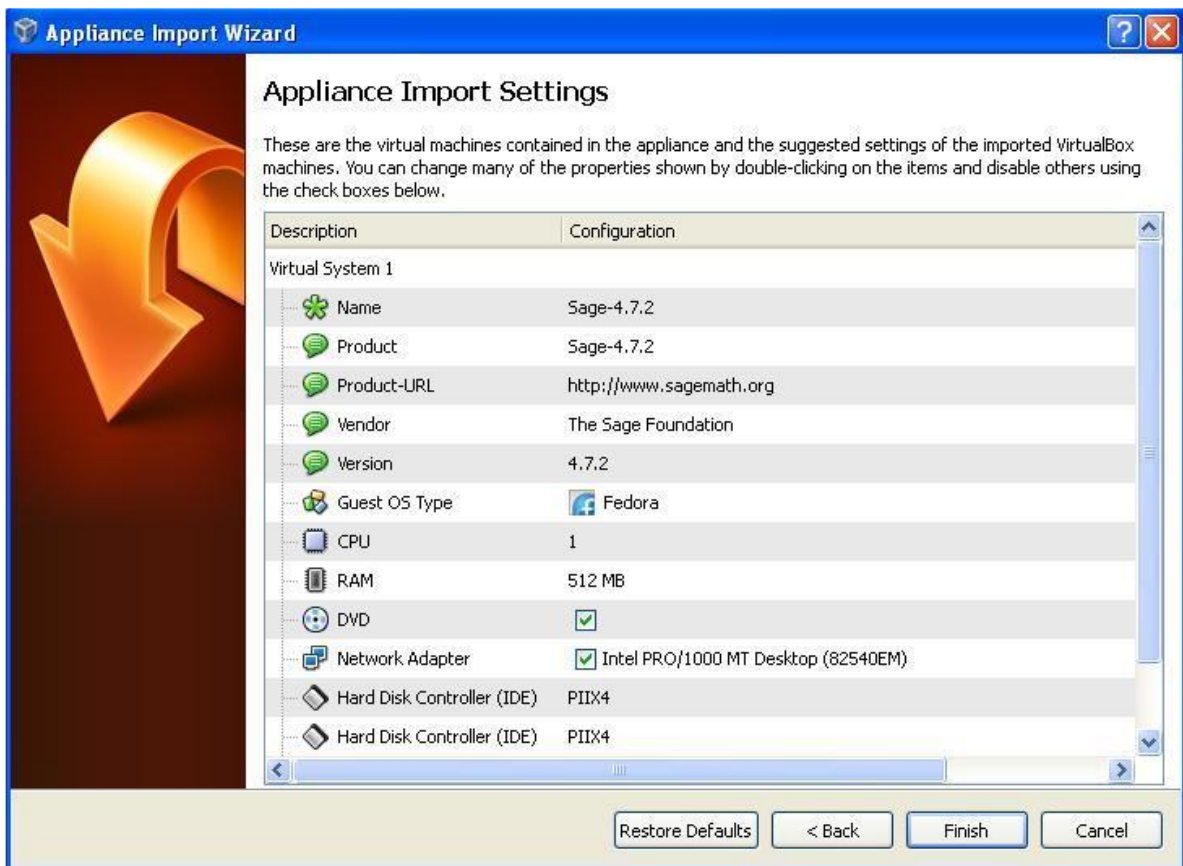
Proces same instalacije je relativno brz, a po završetku bi trebali vidjeti na listi virtualnih strojeve i novo instalirani Sage, sve što nam preostaje je pritisnuti tipku start.

2.2 Proširenje osnovne konfiguracije Sage-a

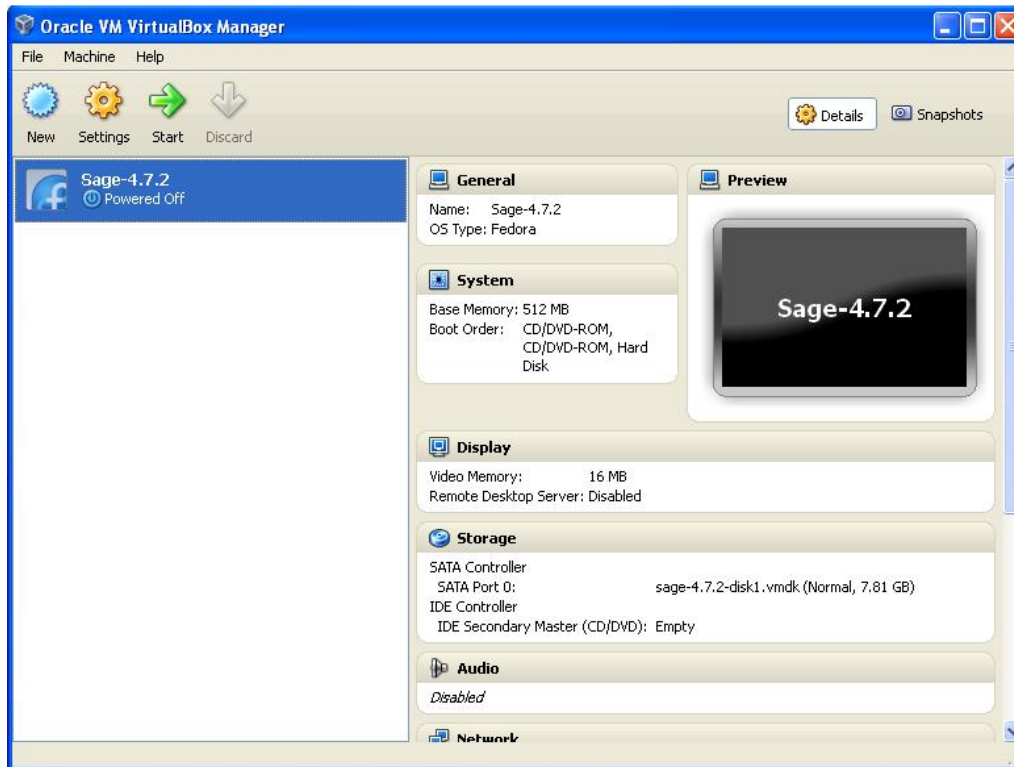
Sage u svojoj jezgri ima integrirane određene pakete koji su dostupni za korištenje odmah po instalaciji. No snaga Sagea je upravo u njegovoj modularnosti i već spomenutoj činjenici da se u njegovu radnu površinu i jezgru mogu



Slika 2.1: Unos virtualizacijskog Sage paketa



Slika 2.2: Postavke paketa



Slika 2.3: Virtualizacijski Sage paket instaliran u VirtualBox

uključivati dodatni paketi, kao i koristiti drugi računski alati. Paket se odabire za svaku radnu površinu specifično, izbornik se nalazi na samom vrhu sučelja, te su nam ponuđeni osnovni(standarni) i opcionalni paketi. Dodatni paket kao takvi nisu napisani konkretno za Sage uporabu, već su to uglavnom samostalne aplikacije i matematički alati koje Sage vješto integrira u svoje sučelje. Od svih mogućih paketa naveo bih samo neke najzanimljivije s kojim sam se osobno susretao, poput: Mathematica, Matlab, Scilab, Octave... Osobno, posjedujem licencu samo za Mathematicu stoga sam za integraciju u Sage koristio Mathematicu, te besplatne alate SciLab i Octave. Integriranje vanjskih paketa je isto u 98% posto slučajeva, te su jedine varijacije koje postoje vezane za lokacije i konkretnih datoteka i naredbi koje dodatno treba upisati, u konačnici proširenja su izvedena dosta jednostavno i praktično. Ukratko bih se samo osvrnuo u par redaka na pojedine pakete.

2.2.1 Mathematica

Wolfram Mathematica je jedan od najpoznatijih svjetskih alata za matematičke proračune, a i šire. Osobno sam Mathematicu intenzivno koristio u mehanici fluida, gdje se jednostavno mogu riješavati problemi linearnih jednadžbi i sl. Ono što je specifično za Mathematicu je činjenica da se odmaklo od klasične programske paradigme, te je sučelje dizajnirano kako bi simuliralo okruženje olovke i papira i odmaklo se od programerskog pristupa koji često može zbuniti ljude i distancirati ih od korištenja. Mathematica je dizajnirana tako da se koristi specifično dizajniran set naredbi i gotovih funkcija te se tako stvorila individualna paradigma specifična za Mathematicu. Sve te karakteristike se mogu prenjeti u Sage sučelje uključanjem Mathematica jezgre(kernel).

2.2.2 Octave i Scilab

Octave, kao i Scilab, je besplatni matematički alat koji se kao i Mathematica može koristiti kao individualna aplikacija za matematičke kalkulacije. Za razliku od Mathematice ovi alati više naginju programskoj paradigmi, odnosno koriste se kao okruženje za programiranje sa velikom paletom paketa, metoda i funkcija koje su dizajnirane za rješavanje inženjerske matematike. Octave je najlakše opistai kao besplatni Matlab, nastao je kao besplatna kopija Matlabu u Linux okruženju, te je do danas proširen do te mjere da se može mjeriti sa moćnim Matlabom. Octave je kao i Matlab više orijentiran na upotrebu u automatici, upravljanju i regulaciji, te stabilnosti



Slika 2.4: Mathematica u Sageu

sustava i sl. Scilab, vrlo sličan, po metodologiji rada Octaveu, je pak više orijentiran na zadovoljavanje numeričkih zahtjeva, sadrži rješenja Galerkinove metode, te raznih numeričkih alata poput metode konačnih elemenata.

2.2.3 Implementacija

Pri proširivanju dodatnim paketima, vodio sam se dokumentacijom koja dolazi sa instalacijom Sage-a. Gotovo svi paketi imaju isti pristup pri implementaciji, a prvi korak je da preuzmete paket koji vas interesira, obično sa interneta. Instalacija varira ovisno operacijskom sustavu, te je ovdje potrebno malo objašnjenje. Originalna ideja Sage-a je izrasla iz prirode otvorenog koda, a on je kao takav evolvirao na Unix operacijskim sustavima, stoga trenutno ne postoji inačica Sage-a i dodatnih paketa koja se može pokretati na Windows sustavu kao njegova vlastita aplikacija. Kao u slučaju osnovne instalacije Sage-a za proširivanje je potrebno koristiti VirtualBox aplikaciju, točnije potrebno je pokrenuti Sage virtualni stroj, te se svi dodatni paketi od tog trena instaliraju kroz taj virtualni stroj. Pošto je Sage virtualni stroj u suštini Linux sa instalacijom Sage-a, sve daljnje operacije koje se koriste u virtualiziranom okruženje se mogu koristiti i u samostalnoj Linux instalaciji. Bilo da je riječ o virtualnoj Sage inačici na Windows sustavu ili Sage-u koji radi na Linux sustavu, potrebno je preuzeti pakete za Linux operacijski sustav. To je moguće ili direktnim preuzimanje dotičnih paketa sa pojedinih web stranica ili kroz Linux sustav za distribuciju aplikacija. Ovisno o distribuciji Linuxa taj proces može biti više ili manje kompliciran, stoga je preporučljivo preuzeti pojedine instalacijske datoteke za svaku aplikaciju pojedinačno. Kad je riječ o Mathematici jedini način je da se instalira je da se preuzmu datoteke sa službenih web stranica i zatim instalira.

Pri instalaciji dodatnih paketa na MacOS sustavu procedura je dobrim dijelom identična. Potrebno je preuzeti instalacijske datoteke za MacOS te ih instalirati na operacijski sustav. Dodatni zahvat koji je potrebno obaviti na MacOS sustavu je kreiranje binarne datoteke koju će Sage pokrenuti te na taj način uspostaviti komunikaciju sa željenim paketom. Na Unix sustavima te datoteke je potrebno staviti u `/usr/bin`, te će za svaki paket biti potrebno napraviti posebnu datoteku adekvatnog imena.

Mathematica binaran datoteka `math`:

```
#!/bin/sh
/Applications/Mathematica.app/Contents/MacOS/MathKernel $@
```

Octave binarna datoteka `octave`:

```
#!/bin/sh
/Applications/Octave.app/Contents/Resources/bin/octave $@
```

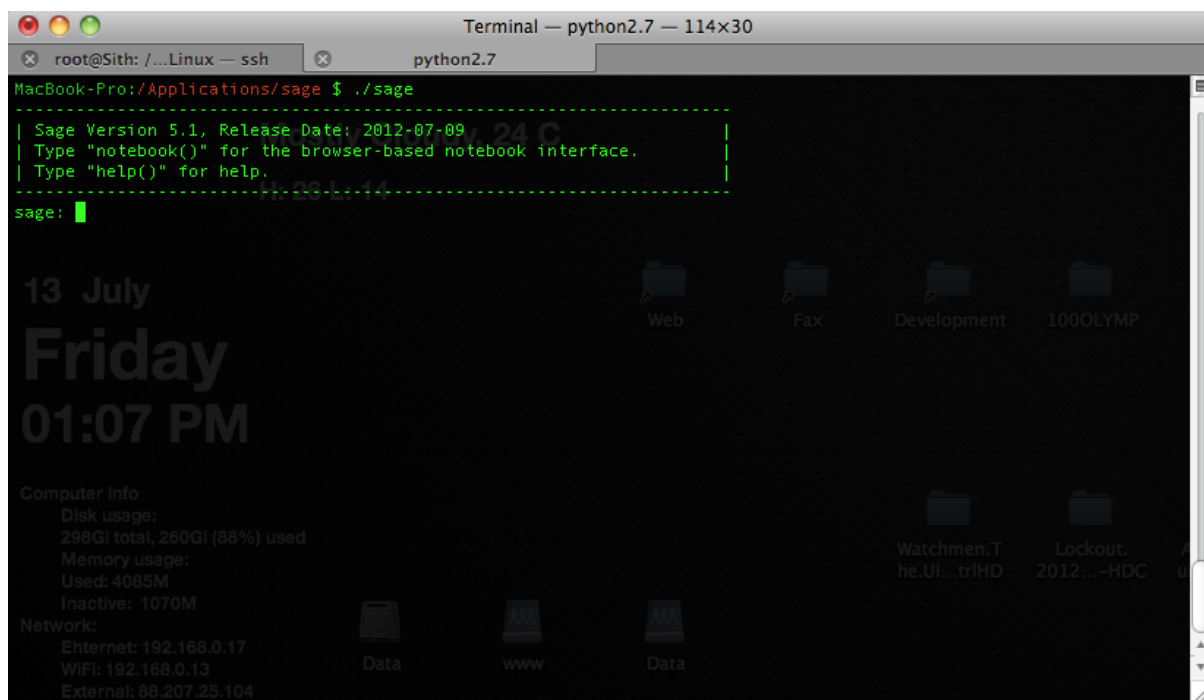
Scilab binarna datoteka scilab:

```
#!/bin/sh
/Applications/scilab-5.3.3.app/Contents/MacOS/bin/scilab $@
```

Imena datoteka su definirana u Sage dokumentaciji, te ne mogu biti proizvoljna. Sage Razvojni tim tvrdi da je za instalaciju dodatnih paketa na Linux operacijske sustave (a time i na Windows sustavu) dovoljno sam instalirati pakete, no u nekim slučajevim spomenute datotke je potrebno napraviti i unutar Linux sustava, te je u tom slučaju procedura identična ovoj opisanoj za MacOS sustav.

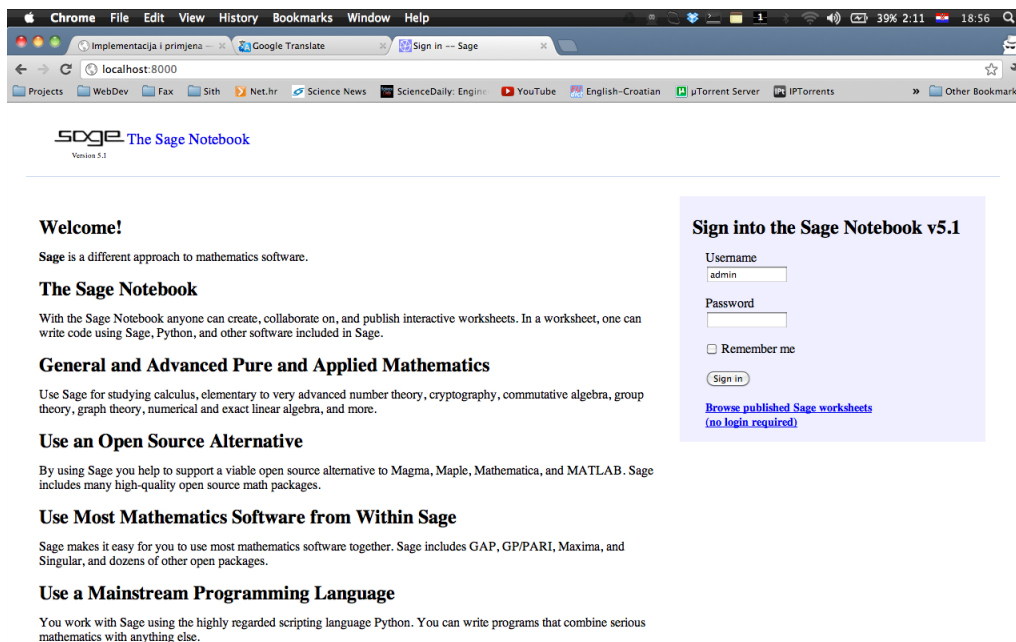
2.3 Pregled sučelja i osnovnih karakteristika

Sage ima dva sučelja za korištenje, jedno je grafičko sučelje kojemu se pristupa kroz bilo koji internet preglednik, dok je drugo CLI (eng. Command Line Interface) sučelje koje se svodi na prozor u kojem se upisuju tekstualne naredbe. Prosječnom korisniku ili studentu je u interesu koristiti grafičko sučelje pošto je takav pristup svima poznat. Sučelje Sagea u internet formatu simulira sučelje aplikacija koje smo već imali prilike vidjeti kod Mathematice, Matlaba ili Octavea. U sučelju postoji prozor za unos matematičkih formulacija koje se računaju, te se izračunavaju kao i u Mathematici, pritiskom tipki *Shift+Enter(Return)*. CLI sučelje, kako je već navedeno, nema nikakav grafički indikator već se koriste tekstualne naredbe.



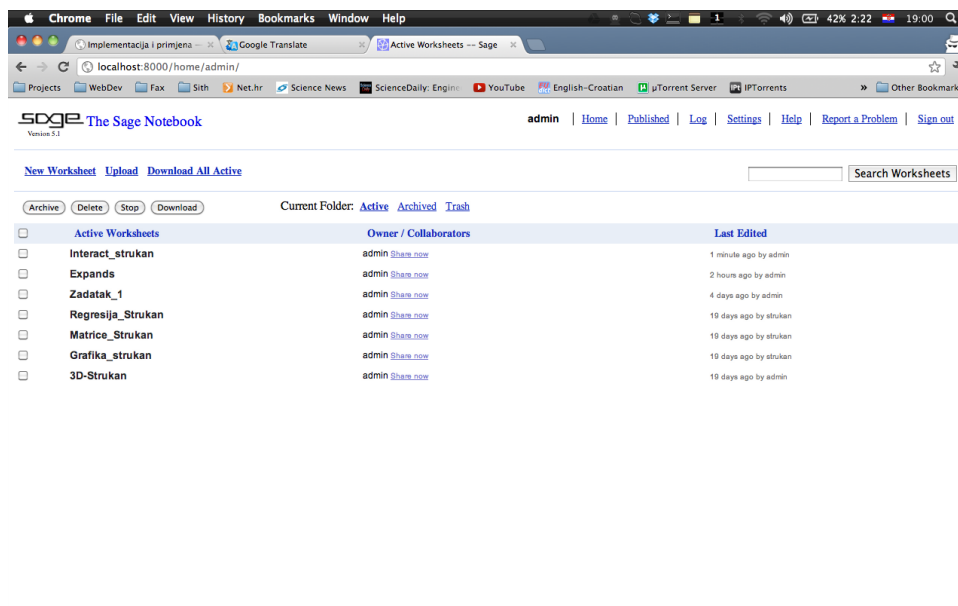
Slika 2.5: Sage CLI sučelje

Unos samih matematičkih izraza se ne razlikuju unosili ih u grafičkom sučelju ili u CLI sučelju. Ono što je vrlo karakteristično za Sage i gdje on dolazi do svog punog sjaja, je činjenica da on za svoje izračune koristi preko 100 “open-source” paketa koji služe za izračunavanje raznih matematičkih problema. Također se u sučelju može idabrati neki od postojećih matematičkih paketa poput Matlaba. Drugim riječima Sage objedinjuje mnoge matematičke alate kroz jedno sučelje, te je ustanju centralizirati procesne operacije na jednom serveru. Neovisno o tipu instalacije, bilo to na Windows sustavu ili Unix sustavu, da bi došli do grafičkog sučelja potrebno je prvo pokrenuti sage server na lokalnom ili mrežnom stroju, te onda u internet pregledniku upisati adresu <http://localhost:8000>. Ako je sve valjano namješteno trebali bi vidjeti stranicu na kojoj postoji *Dobrodošli* naslov to prozor u kojem upisujemo korisničko ime i lozinku za ulaz u sustav.



Slika 2.6: Sage početni pogled

Po ulasku u sustav imamo uvid u sve radne stranice(eng. worksheets) koje su vezane za upisani profil.



Slika 2.7: Popis radnih stranica(eng. worksheets) u Sageu

Postoje mnoge opcije kako manipulirati svojim radom, te su možda neke od najzanimljivijih opcija one za objavljivanje i kolaboraciju na određenim radnim stranicama. Naime, ako postoji potreba da se više od jednog čovjeka uključi na jedan projekt postoji opcija da se za određene radne stranice dodaju kolaboratori koji dobivaju pristup vašim radnim stranicama. Na taj način može se kolaborirati sa ljudima unutar fakulteta ili s drugog kraja svijeta. Druga zanimljiva opcija je objavljivanje radova, u trenutku kada želite objaviti vaš rad, Sage vam dodjeljuje web adresu koja je javna i dostupna svima preko internet preglednika, a možete uključiti opciju kojom će sve naknadne promjene biti automatski objavljene.

SAGE PAKET

Sage je napisan u Python prgoramskom jeziku, stoga je njegova sintaksa sve prisutna u Sageu. Bilo bi dobro za prvi korak malo razjasniti osnovne elemente pythona i prikazati njegovu sintaksu, kako bi se lakše snalazili koristeći Sage.

3.1 Uvod u Python

Python je u stanju računati sve jednostavnije matematičke operacije, te na taj način služi kao svestrani kalkulator. Neke malo kompliciranije operacije su također podržane sa već ugrađenim funkcijama, dok neke zahtjevnije jednostavno treba programirati.

3.1.1 Brojevi

Tumač(interpreter) djeluje kao jednostavan kalkulator: Možete unositi matamatičke izraze te će ih on procijeniti i izračunati. Sintaksa je jednostavna: Operatori +, -, * i / rade baš kao i u većini drugih jezika (na primjer, Pascal ili C), zagrade se mogu koristiti za grupiranje. Na primjer

```
>>> 1+1
2
>>> # Ovak se ubacuje komentar
... 1+1
2
>>> 1+1 # Ovakva komentar je u istoj liniji sa kodom
4
>>> (43-3*2)/5
7
>>>
... 10/2
5
```

Znak jednakosti (=) se koristi za dodjeljivanje vrijednosti u varijablu. Ta operacije neće izbaciti nikakav rezultat dok ne upišemo sljedeću naredbu(operaciju) koja uključuje dotičnu varijablu:

```
>>> sirina = 15
>>> visina = 3*8
>>> sirina * visina
360
```

Vrijednost se može dodijeliti nekoliko varijabli istovremeno

```
>>> a = b = c = 4
>>> a
4
>>> b
4
```



```
>>> c
4
```

Varijable moraju biti “definirane” (imati dodjeljene vrijednost) prije nego što se mogu koristiti ili će se generirati pogreška

```
>>> # ispis ne definirane varijable
... n
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'n' is not defined
```

Podržani su svi tipovi brojeve, tako i cijeli brojevi te pomješane operacije.

```
>>> 3 * 3.75 / 1.5
7.5
>>> 7.0 / 2
3.5
```

Kompleksni brojevi su također podržani, imaginarni brojevi pišu s nastavkom od j ili J . Kompleksni brojevi s stvarnom komponente različitom od nule su pisani u obliku (real + imagj), ili mogu biti kreiran s `complex(real, imag)` funkcijom.

```
>>> 1j * 1J
(-1+0j)
>>> 1j * complex(0,1)
(-1+0j)
>>> 3+1j*3
(3+3j)
>>> (3+1j)*3
(9+3j)
>>> (1+2j)/(1+1j)
(1.5+0.5j)
```

Kompleksni brojevi su uvijek predstavljeni kao dva cijela broja, realna i imaginarna komponenta. Da bi Izdvojili te dijelove iz kompleksnog broja `z`, koristimo `z.real` i `z.imag`.

```
>>> a=1.5+0.5j
>>> a.real
1.5
>>> a.imag
0.5
```

3.1.2 Nizovi

Osim brojeva, Python može manipulirati nizove (string), koji mogu biti izraženi na više načina. Definiraju se jednostrukim navodnicima ili dvostrukim navodnicima.

```
>>> 'jedan string'
'jedan string'
>>> 'Rekao je: "Mozes"'
'Rekao je: "Mozes"'
>>> "Rekao je: \"Mozes\""
'Rekao je: "Mozes"'
```

Interpreter ispisuje nizove na isti način kao što je upisan, naredba `print` će ispisati niz u čitljivijem obliku. Znakovni nizovi mogu biti spojeni (slijepljeni) pomoću `+` operatora, i mogu se ponavljati pomoću `*`

```
>>> ime = 'Simun'+ 'Strukan'
>>> ime
'SimunStrukan'
>>> print ime
SimunStrukan
```

```
>>> '<' + ime*5 + '>'
'<SimunStrukanSimunStrukanSimunStrukanSimunStrukanSimunStrukan>'
```

3.1.3 Liste

U Pythonu postoje *pomješane* vrste podataka, koriste se za grupiranje raznih vrijednosti i podataka. Najsvestranija takva grupa je *Lista*, koja se može napisati kao lista vrijednosti razdvojenih zarezom između uglatih zagrada. Stavke liste ne moraju sve biti istog tipa

```
>>> a = ['list', 'pile', 120, 4321]
>>> a
['list', 'pile', 120, 4321]
```

Stavke liste, imaju svoje indexe, koji počinju sa 0. Liste mogu biti spojene, rezane, itd.

```
>>> a[0]
'list'
>>> a[3]
4321
>>> a[-2]
120
>>> a[1:-1]
['pile', 120]
>>> a[:2] + ['kokos', 3*3]
['spam', 'eggs', 'kokos', 9]
>>> 3*a[:3] + ['Boba']
['list', 'pile', 120, 'list', 'pile', 120, 'list', 'pile', 120, 'Boba']
```

Za razliku od nizova, koji su *nepromjenjivi*, moguće je promijeniti pojedine stavke liste.

```
>>> a
['list', 'pile', 120, 4321]
>>> a[2] = a[2] - 20
>>> a
['list', 'pile', 100, 4321]
```

Moguće je gnjezditi liste (stvarati liste čiji su elementi liste), na primjer

```
>>> p = [1, 'string', 33]
>>> q = [55, p, 'niz']
>>> q
[55, [1, 'string', 33], 'niz']
>>> q[1][2]
33
>>> q[1][1]
'string'
```

3.1.4 Prvi korak prema programiranju

Naravno, možemo koristiti Python za složenije zadatke od zbranja dva broja. Na primjer, možemo napisati početni pod-slijed *fibonaccijevog* niza

```
>>> a,b = 0,1
>>> while b < 10:
...     print b
...     a,b = b, a+b
...
1
1
2
3
```

5
8

Ovaj primjer uvodi nekoliko novih značajki.

- Naredba `while` se izvršava dok je uvjet (`b < 10`) istinit. U Pythonu, kao u C, bilo koji broj različit od nule se podrazumjeva kao istinita tvrdnja, nula je neistinita. Uvjet može biti niz (string) ili popis vrijednosti, zapravo bilo koja sekvenca, bilo što ne-nulte duljine je istina, prazne sekvence su neistina. Test koji se koristi u primjeru je jednostavna usporedba. Standardni operatori usporedbe su pisani isti kao u C: `<` (manje od), `>` (veći od), `==` (jednako), `<=` (manje od ili jednako), `>=` (veće ili jednako) i `!=` (nije jednako). Ono što `while` petlja ovdje radi je nešto što se naziva *kontrola toka*, odnosno kod se izvršava uz neke postavljene uvjete. Postoji nekoliko naredbi kojima vršimo takve provjere i time kontroliramo proces koji se provodi
- *Tijelo* petlje je *uvučeno*: Uvlačenje u Pythonu je način grupiranja naredbi. Imajte na umu da svaki redak u osnovnom blok mora biti uvučen za isti iznos.

3.1.5 *if* naredba

Jedna onda najpoznatijih naredbi za provjeru nekog uvjeta, *if* naredba, npr:

```
>>> n = int(raw_input("Unesite broj: "))
Unesite broj: 3
>>> if n % 2 == 0:
...     print 'Broj je paran!'
... else:
...     print 'Broj je neparan!'
...
Broj je neparan!
```

Pri provjeri uvjeta možemo neogranično nizati `elif` naredbe, također `else` naredba je opcionalna. Naredba `elif` je skraćeno za `else if`, te je korisna kako bi se izbjeglo prekomjerna uvlačenja. Opisana sekvenca naredbi je zamjena za `switch` ili `case` naredbe koje se koriste u drugim jezicima.

3.1.6 *for* naredba

Naredba `for` se u Pythonu malo razlikuje od onoga što se može koristiti u C ili Pascala. Umjesto iteracije kroz aritmetičku progresiju brojeva (kao u Pascalu), ili dajući korisniku mogućnost za definiranje koraka iteracije i uvjeta za zaustavljanje iteracije (kao C), u Python-u `for` iterira kroz stavke i sekvence (lista ili string), redsljedom kojim se pojavljuju u sekvenci.:

```
>>> lista = ['kokos', 'jaje', 'pile']
>>> for el in lista:
...     print el
...
kokos
jaje
pile
```

Nije preporučljivo mjenjati sekvence kroz koje se iterira u petlji (to se uglavnom provodi u promjenjivim sekvencama poput lista). Ako je potrebno mijenjati listu kroz koju se iterira (na primjer, kako bi kopirali odabrane stavke), preporuča se iteracija kroz kopiju.

```
>>> lista = ['kokos', 'jaje', 'vrlo dugi element']
>>> for x in lista[:]:
...     if len(x) > 6: lista.insert(0, x)
...
>>> lista
['vrlo dugi element', 'kokos', 'jaje', 'vrlo dugi element']
```

3.1.7 Funkcija *range*

Ako je potrebno iterirati kroz niz brojeva, ugrađena funkcije `range` je vrlo korisna. Stvara listu koja sadrži aritmetičku progresiju

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Zadnji argument (broj) nikada nije dio proizvedene liste, `range(10)` generira listu od 10 stavki. Moguće je raspon iteracije početi na nekom drugom mjestu, ili se može odrediti različiti prirast (čak i negativan, te se ponekad takav prirast zove 'korak'):

```
>>> range(5, 10)
[5, 6, 7, 8, 9]
>>> range(0, 10, 3)
[0, 3, 6, 9]
>>> range(-10, -100, -30)
[-10, -40, -70]
```

Za iteraciju preko indeksa neke sekvence, može se kombinirati: `range` i `len` na način:

```
>>> for i in range(len(lista)):
...     print i, lista[i]
...
0 vrlo dugi element
1 kokos
2 jaje
3 vrlo dugi element
```

3.1.8 Naredbe *break* and *continue* i *else* uvijet u petlji

Naredba `break`, kao u C, zaustavlja najmanji blok `for` ili `while` petlje. Naredba `continue`, logično onda nastavlja sa sljedećom iteracijom petlje. Petlje mogu imati `else` uvijet u sebi koji se aktivira kada petlja prođe kroz sve iteracije (kod `for`) ili kada uvijet postane netočan (kod `while`), ali ne kada se petlja prekine sa naredbom `break`:

```
>>>for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print n, 'jednako je', x, '*', n/x
...             break
...         else:
...             # petlja nije pronašla parni faktor
...             print n, 'je prosti broj'
...
2 je prosti broj
3 je prosti broj
4 jednako je 2 * 2
5 je prosti broj
6 jednako je 2 * 3
7 je prosti broj
8 jednako je 2 * 4
9 jednako je 3 * 3
```

3.1.9 *pass* Naredba

Naredba `pass` ne radi ništa. Ona se može koristiti kada se izjava traži sintaktički, ali program ne zahtijeva akciju. Na primjer

```
>>> while True:
...     pass # Čeka prekid od strane korisnika (Ctrl+C)
... 
```

To se obično koristi za stvaranje minimalne klase

```
>>> class MojaKlasa:
...     pass
... 
```

Drugo mjesto gdje se `pass` može koristiti je kao rezervirano mjesto za funkciji ili uvjetna tijelo kad se radi na novom kodu, na taj način možemo razmišljati na apstraktnoj razini.

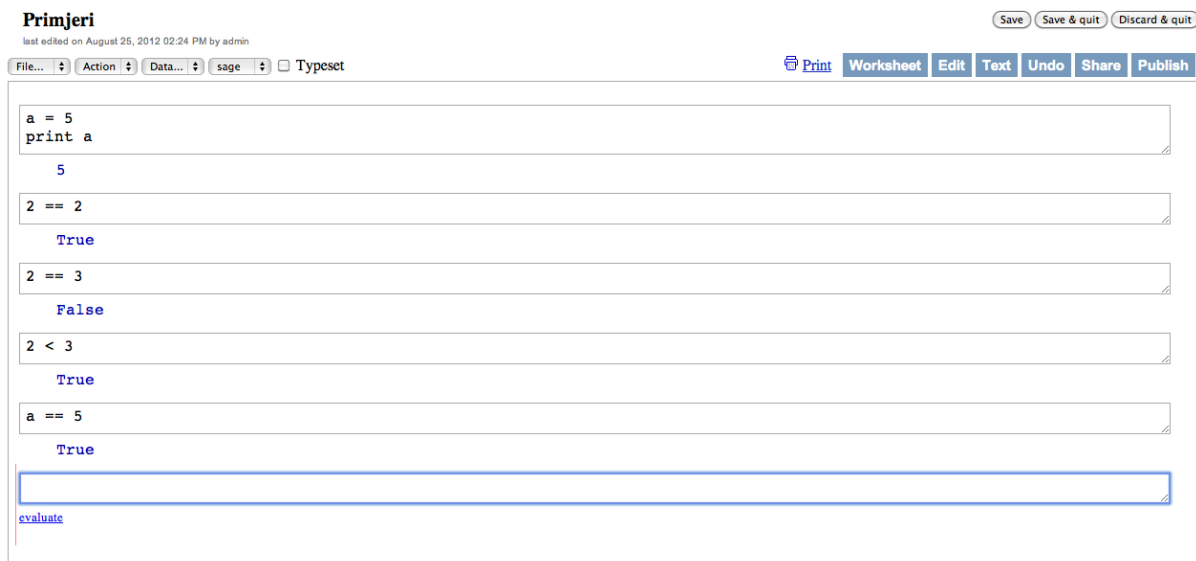
```
>>> def initlog(*args):
...     pass # funkcija koja ima ime ali će se logika kasnije napisati!
... 
```

3.2 Uporaba Sage paketa

Sada kada su razjašnjene neke osnovne karakteristike Python programskog jezika, možemo iste metode koristiti i pri programiranju u Sage okruženju.

3.2.1 Dodjeljivanje vrijednosti, jednakosti i aritmetika

Uz neke manje iznimke, Sage se koristi kao Python programski jezik, tako da većina znanja i literature o Pythonu može pomoći u korištenju Sagea. Sage kao u Pythonu koristi `=` za dodjeljivanje vrijednosti, dok znakove `==` `<=` `>=` `<` `>` koristi za usporedbe:



Slika 3.1: Sage: Dodjeljivanje vrijednosti

Matematičke operacije:

Računanje izraza poput $3^2 * 4 + 2\%$ ovisi o poredku kojim se operacije primjenjuju, redosljed je definiran u tablici “operator precedence table”. Sage također pruža mnoge poznate matematičke funkcije, ovdje je samo nekoliko primjera:

```
sage: sqrt(3.4)
1.84390889145858
sage: sin(5.135)
```

The screenshot shows the SageMath interface with the following input and output pairs:

- Input: `2**3 #operacija exponenta`, Output: `8`
- Input: `2^3 #alternativna operacije exponenta`, Output: `8`
- Input: `10 % 3 # % je ekvivalent mod funkcije, povjerava da li je broj djeliv sa ostatkom ili bez`, Output: `1`
- Input: `10 % 5`, Output: `0`
- Input: `10/4`, Output: `5/2`
- Input: `10//4 #kao rezultat dobivamo zaokruženu vrijednost`, Output: `2`
- Input: `4 * (10 // 4) + 10 % 4 == 10`, Output: `True`
- Input: `3^2*4 + 2%5`, Output: `38`

Slika 3.2: Sage: Osnovna aritmetika

```
-0.912021158525540
sage: sin(pi/3)
1/2*sqrt(3)
```

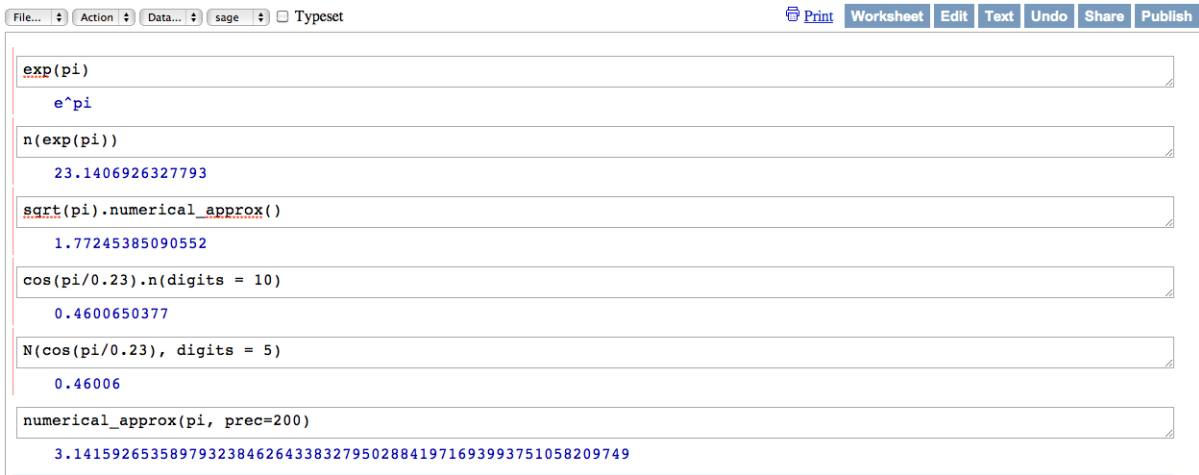
The screenshot shows the SageMath interface with the following input and output pairs:

- Input: `sqrt(3.4)`, Output: `1.84390889145858`
- Input: `sin(5.135)`, Output: `-0.912021158525540`
- Input: `sin(pi/3)`, Output: `1/2*sqrt(3)`

Slika 3.3: Sage: Osnovne funkcije

Kako primjer pokazuje, neki matematički izrazi će vratiti “točna” rješenja, umjesto numeričkih aproksimacija. Da biste dobili numeričke aproksimacije, koristite ili funkciju `n` ili metodu `n` (obje imaju duži naziv `numerical_approx`, a funkcija `N` je ista kao `n`). Metode primaju dodatni argument `prec`, koji definira stupanj preciznost u bitovima, i `digits`, koji određuje broj decimala, po standardu je zadano 53 bita preciznosti.

```
sage: exp(pi)
e^pi
sage: n(exp(pi))
23.1406926327793
sage: sqrt(pi).numerical_approx()
1.77245385090552
sage: cos(pi/0.23).n(digits = 10)
0.4600650377
sage: N(cos(pi/0.23), digits = 5)
0.4600650377
sage: numerical_approx(pi, prec=200)
3.1415926535897932384626433832795028841971693993751058209749
```



Slika 3.4: Sage: Numerička aproksimacija

3.2.2 Pomoć u Sageu

Sage ima veliku ugrađenu dokumentaciju, dostupnu upisivanjem naziv funkcije ili konstante, iza koje slijedi upitnik:

```
sage: tan?
Type:      <class 'sage.calculus.calculus.Function_tan'>
Definition: tan( [noargspec] )
Docstring:
```

The tangent function

EXAMPLES:

```
sage: tan(pi)
0
sage: tan(3.1415)
-0.0000926535900581913
sage: tan(3.1415/4)
0.999953674278156
sage: tan(pi/4)
1
sage: tan(1/2)
tan(1/2)
sage: RR(tan(1/2))
0.546302489843790
```

```
sage: log2?
Type:      <class 'sage.functions.constants.Log2'>
Definition: log2( [noargspec] )
Docstring:
```

The natural logarithm of the real number 2.

EXAMPLES:

```
sage: log2
log2
sage: float(log2)
0.69314718055994529
sage: RR(log2)
0.693147180559945
sage: R = RealField(200); R
Real Field with 200 bits of precision
sage: R(log2)
0.69314718055994530941723212145817656807550013436025525412068
```

```

sage: l = (1-log2)/(1+log2); l
(1 - log(2))/(log(2) + 1)
sage: R(l)
0.18123221829928249948761381864650311423330609774776013488056
sage: maxima(log2)
log(2)
sage: maxima(log2).float()
.6931471805599453
sage: gp(log2)
0.6931471805599453094172321215          # 32-bit
0.69314718055994530941723212145817656807 # 64-bit
sage: sudoku?
File:      sage/local/lib/python2.5/site-packages/sage/games/sudoku.py
Type:      <type 'function'>
Definition: sudoku(A)
Docstring:

```

Solve the 9x9 Sudoku puzzle defined by the matrix A.

EXAMPLE:

```

sage: A = matrix(ZZ,9,[5,0,0, 0,8,0, 0,4,9, 0,0,0, 5,0,0,
0,3,0, 0,6,7, 3,0,0, 0,0,1, 1,5,0, 0,0,0, 0,0,0, 0,0,0, 2,0,8, 0,0,0,
0,0,0, 0,0,0, 0,1,8, 7,0,0, 0,0,4, 1,5,0, 0,3,0, 0,0,2,
0,0,0, 4,9,0, 0,5,0, 0,0,3])
sage: A
[5 0 0 0 8 0 0 4 9]
[0 0 0 5 0 0 0 3 0]
[0 6 7 3 0 0 0 0 1]
[1 5 0 0 0 0 0 0 0]
[0 0 0 2 0 8 0 0 0]
[0 0 0 0 0 0 0 1 8]
[7 0 0 0 0 4 1 5 0]
[0 3 0 0 0 2 0 0 0]
[4 9 0 0 5 0 0 0 3]
sage: sudoku(A)
[5 1 3 6 8 7 2 4 9]
[8 4 9 5 2 1 6 3 7]
[2 6 7 3 4 9 5 8 1]
[1 5 8 4 6 3 9 7 2]
[9 7 4 2 1 8 3 6 5]
[3 2 6 7 9 5 4 1 8]
[7 8 2 9 3 4 1 5 6]
[6 3 5 1 7 2 8 9 4]
[4 9 1 8 5 6 7 2 3]

```

Sage također pruža tzv. *tab completion*: Utipkajte prvih nekoliko slova funkcija, a zatim pritisnite tipku TAB. Na primjer, ako upišete `ta` slijedi TAB, Sage će se ispisati `tachyon`, `tan`, `tanh`, `Taylor`. Tako imamo na raspolaganju jednostavan alat za traženje imena funkcija i struktura u Sageu.

3.2.3 Funkcije, uvlačenja i brojanje

Da biste definirali novu funkciju u Sageu, koristite naredbu `def` i dvotočku nakon popisa naziva varijabli. Na primjer:

```

def parnost(number):
    if number%2 == 0:
        print 'broj je paran!'
    else:
        print 'broj je neparan!'

```

Pri kreiranju funkcija ne određuju se vrste ulaznih argumenata. Može se navesti više argumenata, od kojih svaki može imati opcionalnu vrijednost. Na primjer, zadana funkcija će dodjeliti `divisor=2` ako pozivatelj funkcije

ne dodjeli vrijednost navedenom argumentu.

```
def djeljivost(br, djelitelj = 2):
    if djelitelj == 2:
        print 'Ispitujemo je li broj paran...'
        return parnost(br)
    else:
        print 'ispitujemo da li je broj', br, 'djeljiv sa', djelitelj
        if br%djelitelj == 0:
            print 'Broj %d je djeljiv sa %d' %(br, djelitelj)
        else:
            print 'Broj %d nije djeljiv sa %d' %(br, djelitelj)
```

```
def parnost(number):
    if number%2 == 0:
        print 'broj je paran!'
    else:
        print 'broj je neparan!'
```

```
parnost(4)
```

```
broj je paran!
```

```
def djeljivost(br, djelitelj = 2):
    if djelitelj == 2:
        print 'Ispitujemo je li broj paran...'
        return parnost(br)
    else:
        print 'ispitujemo da li je broj', br, 'djeljiv sa', djelitelj
        if br%djelitelj == 0:
            print 'Broj %d je djeljiv sa %d' %(br, djelitelj)
        else:
            print 'Broj %d nije djeljiv sa %d' %(br, djelitelj)
```

```
djeljivost(5)
```

```
Ispitujemo je li broj paran...
broj je neparan!
```

```
djeljivost(12, 5)
```

```
ispitujemo da li je broj 12 djeljiv sa 5
Broj 12 nije djeljiv sa 5
```

jsMath

Slika 3.5: Sage: Korisnička funkcija

U Pythonu, blokovi koda se ne grupiraju vitičastim zagradama ili `begin` i `end` blokovima kao u mnogim drugim jezicima. Umjesto toga, blokovi koda označeni su uvlačenjem, koje se mora točno podudarati. Na primjer, sljedeće je sintaktička pogreška, jer `else` izjava nije uvučena za isti iznos kao i druge linije gore.

```
def parnost(number):
    if number%2 == 0:
        print 'broj je paran!'
else:
    print 'broj je neparan!'
```

Ako se razina uvlačenja popravi, funkcija radi

```
def parnost(number):
    if number%2 == 0:
        print 'broj je paran!'
    else:
        print 'broj je neparan!'
```

Točka zarezu nisu potrebni na krajevima linija, linija je u većini slučajeva završila prelaskom u novi red. Međutim, možete staviti više izjava u jednoj liniji koda odvojene točka zarezom

```
sage: a = 5; b = a + 3; c = b^2; c
64
```

U Sageu, možemo pobrojavati tako da iteriramo po rasponu brojeva. Na primjer, Prva linija u nastavku ista kao `for (i = 0; i < 5; i++)` u C++ ili Javi:

```
def parnost(number):
    if number%2 == 0:
        print 'broj je paran!'
    else:
        print 'broj je neparan!'

evaluate
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "_sage_input_54.py", line 10, in <module>
    exec compile(u'open("__code__.py","w").write("# -*- coding: utf-8 -*-\n" + _support_.preparse_worksheet_cell(
  File "", line 1, in <module>

  File "/private/var/folders/Bs/Bs1fLVZiGjWj2UQHhT23Sk+++TM/-Tmp-/tmpgIAeJ2/__code__.py", line 6
    else:
    ^
SyntaxError: invalid syntax
```

Slika 3.6: Sage: Greška u sintaksi

```
def parnost(number):
    if number%2 == 0:
        print 'broj je paran!'
    else:
        print 'broj je neparan!'

parnost(22)

broj je paran!
```

Slika 3.7: Sage: Ispravljena sintaksa

```
sage: for i in range(5):
...     print i
0
1
2
3
4
```

Sljedeća linija odgovara izrazu `for (i=4; i<10; i++)`

```
sage: for i in range(4,10):
...     print i
4
5
6
7
8
9
```

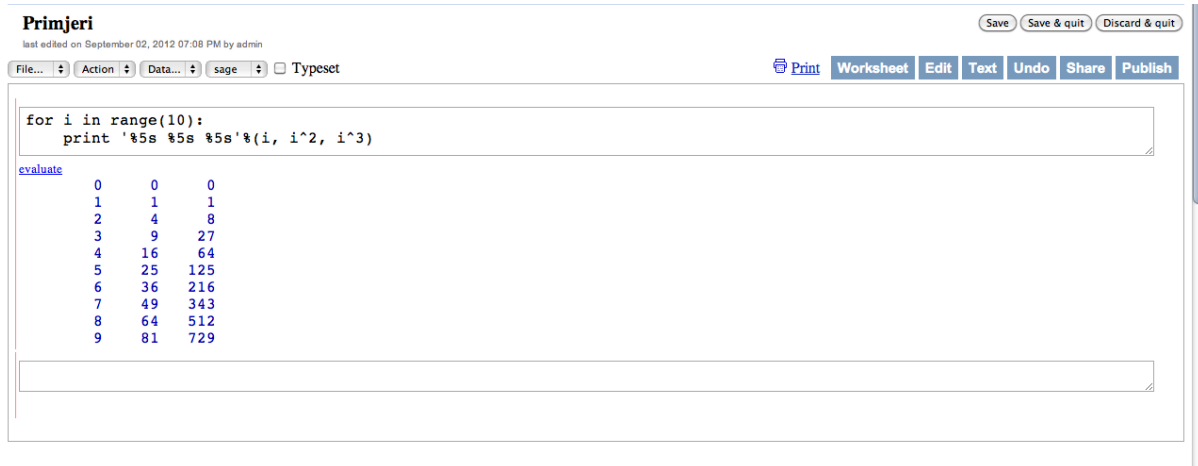
Treći argument određuje veličinu koraka, stoga je sljedeći primjer jednak izrazu `for (i=1; i<6; i+=2)`.

```
sage: for i in range(4,12,3):
...     print i
4
7
10
```

Često je potrebno stvoriti lijepu tablicu za prikaz brojeva koju su izračunati pomoću Sagea. Jednostavan način da to učinite je da koristite oblikovanje nizova (string formatting). U nastavku smo stvorili tri stupca svaki širine 6 i napravili tablicu kvadrata i kuba.

```
sage: for i in range(10):
...     print '%5s %5s %5s'%(i, i^2, i^3)
```

Najosnovnijih struktura podataka u Sage je lista, koji je - kako ime sugerira - samo popis proizvoljnih objekata. Na primjer, `range` naredba koju smo koristili stvara listu:



Slika 3.8: Sage: Tablični prikaz rješenja

```
sage: range(2,10)
[2, 3, 4, 5, 6, 7, 8, 9]
```

Malo kompliciranija lista

```
sage: lista = [1, "bok", 2/3, sin(x^3)]
sage: v
[1, 'bok', 2/3, sin(x^3)]
```

Koristimo `len(lista)` da dobijemo dužinu od lista koristimo `lista.append(obj)` kako bi dodali novi objekt na kraj liste `lista` i koristimo `del lista[i]` da izbrišemo i -ti elemnt liste

```
sage: len(lista)
4
sage: lista.append(1.5)
sage: lista
[1, 'hello', 2/3, sin(x^3), 1.500000000000000]
sage: del lista[1]
sage: lista
[1, 2/3, sin(x^3), 1.500000000000000]
```

Druga važna struktura podataka je rječnik (ili asocijativne polje). Funkcionira kao lista, osim što se stavke mogu indeksirati s gotovo bilo kojim objektom (indeksi moraju biti nepromjenjivi):

```
sage: assoc_polje = {'br':-2, 3/8:pi, e:pi}
sage: assoc_polje['br']
-2
sage: assoc_polje[e]
pi
```

Mogu se definirati nove vrste podataka pomoću klase. Pisanje matematički formulacija u klase je moćna tehnika koja može pomoći da se pojednostavi i organizirati programski kod. U nastavku ćemo definirati klasu koja predstavlja popis pozitivnih brojeva do n ;

```
class Pozitiva(list):
    def __init__(self, n):
        self.n = n
        list.__init__(self, range(2, n+1, 2))
    def __repr__(self):
        return "Pozitivni parni brojevi do %d" %(self.n)
```

`__init__` metoda se poziva kako bi se inicijalizirao objekt kada se stvori; `__repr__` metoda ispisuje željenje podak iz objekta. Pozivamo metodu konstrukcije liste u drugom retku konstruktora `__init__` metode. Sada smo stvorili objekt klase `Pozitiva` na sljedeći način:

```
sage: e = Pozitiva(25)
sage: e
Pozitivni parni brojevi do 25
```

Imajte na umu da `e` ispisuje pomoću `__repr__` metode koju smo definirali. Da biste vidjeli temeljni popis brojeva, koristite `list` funkciju:

```
sage: list(e)
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24]
```

Možemo pristupiti n -tom elementu ili objektu od `e` kao što bi pristupili elementu liste

```
sage: e.n
25
sage: e[2]
6
```

Primjeri
last edited on August 25, 2012 02:24 PM by admin

Save Save & quit Discard & quit

File... Action... Data... sage... Typeset

Print Worksheet Edit Text Undo Share Publish

```
class Pozitiva(list):
    def __init__(self, n):
        self.n = n
        list.__init__(self, range(2, n+1, 2))
    def __repr__(self):
        return "Pozitivni parni brojevi do %d" % (self.n)
```

```
e = Pozitiva(25)
e
Pozitivni parni brijevi do 25
```

```
list(e)
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24]
```

```
e.n
25
```

```
e[2]
6
```

jMath

Slika 3.9: Sage: Korisnička klasa sa listom brojeva

PYTHON PROGRAMIRANJE U SAGE OKRUŽENJU

4.1 Objektno programiranje

Pošto je Python objektni jezik, ponašanje varijabli u Pythonu je nešto drugačije od onoga koje možemo vidjeti u programskom jeziku C ili Php. Razlog je tome što Python nema varijable već ima imena. Također je potrebno malo razjasniti kako se ponašaju objekti i kako pisati vlastite objekte.

4.1.1 Varijable, objekti i imena

Objašnjenje je najlakše prikazati kroz primjer, u mnogim drugim jezicima dodjeljivanje vrijednosti varijabli je kao stavljanje nekog sadržaja u kutiju

```
int a = 1;
```

Kutija “a” sada sadrži broj 1. Dodjeljivanjem nove vrijednosti istoj varijabli ćemo zamjeniti dosadašnji sadržaj kutije i zamjeniti ga novim sadržajem.

```
a = 2;
```

Sada naša kutija sadrži broj 2. Ako dodjelimo jednu varijablu drugoj sadržaj i iz prve varijable će se kopirati i zamjeniti sadržaj varijable kojoj dodjeljujemo.

```
int b = a;
```

Sada imamo kutiju “a” i “b” i obje sadrže isti broj 2. U Pythonu je to ponašanje drugačije. Barato sa “imenima” ili “identifikatorima” koji se ponašaju poput pločica sa imenom koje dodjeljujemo pojedinim objektima.

```
a = 1
```

U ovom slučaju imamo objekt broj 1 kojem je dodjeljena “pločica” na kojoj piše “a”, odnosno objektu 1 je dodjeljeno ime “a”. Ako imenu a dodjelimo novu vrijednost, u biti smo premjestili ime sa objekta broj 1 na objekt broj 2.

```
a = 2
```

Originalni objekt broj 1 više nema svoje ime. On može i dalje postojati ali mu više ne možemo pristupiti kroz ime “a”. Ako dodjelimo jedno ime drugom imenu mi u biti samo istom objektu dodjeljujemo još jedno ime.

```
b = a
```

Ime “b” je sada još jedno ime kojim zovemo isti objekt kao i imenom “a”. Iako u praksi referiramo na varijable u Pythonu one to u biti nisu, ali koristimo takvu uvriježenu terminologiju. Ono što se u Pythonu podrazumjeva pod varijabla je u biti “ime” ili “identifikator”, te ih vizualiziramo kao pločice sa imenom koje dodjeljujemo objektima, a ne kao označene kutije u koje spremamo vrijednosti.

Kao posljedicu imamo sljedeće ponašnje: Kada imamo dva imena za isti objekt promjena sadržaja jednog imena će rezultirati promjenom sadržaja drugog imena, jer u konačnici mi pristupamo istom objektu(ili sadržaju) kroz dva različita imena

```
sage: a = ['ime', 'prezime', 'spol']
sage: b = a
sage: b[1] = 'strukan'
sage: a
['ime', 'Strukan', 'spol']
```



```
a = ['ime', 'prezime', 'spol']
b = a
b[1] = 'Strukan'
a
evaluate
['ime', 'Strukan', 'spol']
```

Slika 4.1: Sage: Ponašanje objekata

No treba imati na umu da ako preraspodjelimo ime “b” nema nikakvog utjecaja na objekt imena “a”

```
sage: b = 7
sage: b
7
sage: a
['ime', 'Strukan', 'spol']
```

Objekti sa sobom donose još dosta zanimljivih karakteristika koje znatno mjenjaju pristup programiranju od onog na koji smo navikli u recimo php-u ili C-u, te donosi jednu drugačiju paradigmu programiranja, tzv. objektu paradigmu.

4.1.2 Objektno orjentirana programska paradigma

Objektno programiranje počiva na sljedećim fundamentalnim pravilima:

1. Sve što dolazi iz stvarnog (ili matematičkog) svijeta a treba biti manipulirano od strane računala je oblikovano objektom.
2. Svaki objekt je manifestacije neke klase.

Objekt bi mogli opisati kao dio memorije koji sadrži informacije potrebne da bi računalo moglo manipulirati nekakav stvarni pojam, dok klasa definira tipove podataka koji služe da pohrane objekt koji je manifestacija klase u kojoj je definirano ponašanje objekta. Mogli bi metaforom opisati tu vezu: Ako je objekt motor koji obavlja neku radnju, recimo elektromotor koji podiže neki teret, onda je klasa tehnički nacrt i dokumentacija koja nam govori kako napraviti taj motor, koliku će snagu prozvesti, te kako ga pokrenuti i koristiti. Tako je i klasa skup programski naredbi koje definiraju kako se neki objekt pokreće, koristi i koje su mu karakteristike. Podaci koji su povezani sa nekim objektom su pohranjeni u tzv. **atribute**. Atributima možemo pristupiti kroz sintaksu `obj.ime_atributa`. Ponašanje pojedinog objekta je definirano **metodama**, te njima pristupamo na isti način `obj.ime_metode`. Dakle da bi definirali objekt prvo je potrebno napisati klasu. Unutar klase definiramo metode i attribute koji onda pribadaju objektu koji ćemo napraviti. U pythonu je sve objekt, što logično znači da za svaki objekt postoji klasa koja ga definira i sadrži operacije koje je moguće obaviti na ili sa objektom. Identično je i sa brojevima ili listama koje možemo koristiti u pythonu. Na primjer možemo vidjeti klasu koja definira objekt broj odnosno **integer**, ako u pythonu upišemo naredbu `help(int)`

```
class int(object)
| int(x[, base]) -> integer
|
| Convert a string or number to an integer, if possible. A floating point
| argument will be truncated towards zero (this does not include a string
| representation of a floating point number!) When converting a string, use
| the optional base. It is an error to supply a base when converting a
| non-string. If base is zero, the proper base is guessed based on the
```

```

| string content. If the argument is outside the integer range a
| long object will be returned instead.
|
| Methods defined here:
|
| __abs__(...)
|     x.__abs__() <==> abs(x)
|
| __add__(...)
|     x.__add__(y) <==> x+y
|
| __and__(...)
|     x.__and__(y) <==> x&y
|
| __cmp__(...)
|     x.__cmp__(y) <==> cmp(x, y)
|
| __coerce__(...)
|     x.__coerce__(y) <==> coerce(x, y)
|
| __div__(...)
|     x.__div__(y) <==> x/y
|
| __divmod__(...)
|     x.__divmod__(y) <==> divmod(x, y)
|
| __float__(...)
|     x.__float__() <==> float(x)
|
| __floordiv__(...)
|     x.__floordiv__(y) <==> x//y
|
| __format__(...)
|
| __getattr__(...)
|     x.__getattr__('name') <==> x.name
|
| __getnewargs__(...)
|
| ...
| ...

```

Ovdje vidimo listu metoda koje možemo pokrenuti kroz objekt `int`, drugim riječima u python jeziku je definirana metoda `__add__` koja se pokreće kada zbrojimo dva broja `x+y`. Na taj način vidimo da su sve naredbe koje pokrećemo u pythonu u biti metode nekog objekta, te se jedino razlikuju po sintaksi upotrebe. Tako, kada pozivamo funkciju `len()` kojom dobivamo dužine neke liste, u biti pozivamo metodu `__len__(...)` koja je metoda `list` objekta, a koju pozivamo na način:

```

sage: b = [1,2,3,4]
sage: len(b)
4

```

Možemo pozvati metodu iz objekta `list` koju ćemo primjeniti na samom objektu `list`:

```

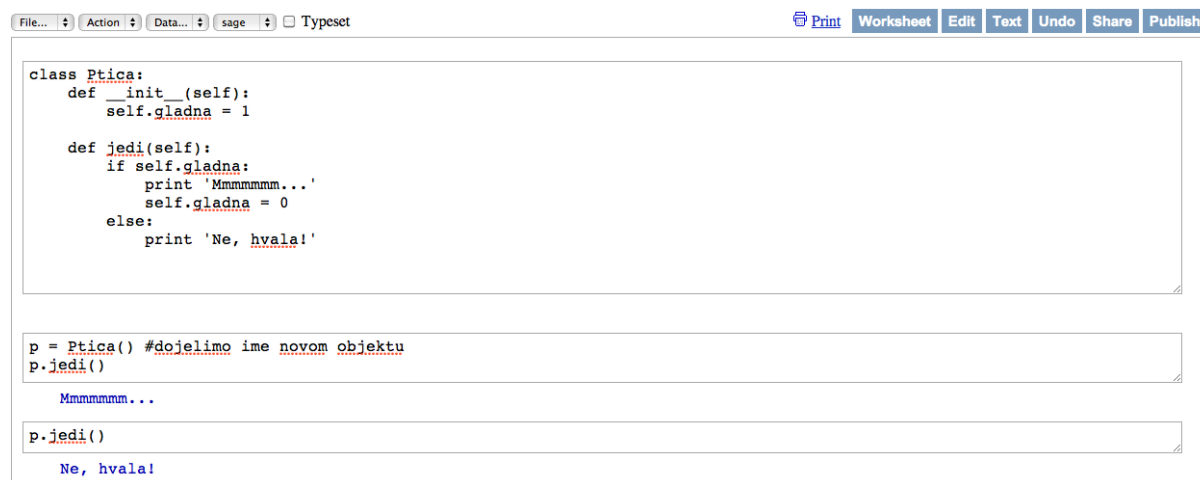
sage: b.append(5)
sage: b
[1, 2, 3, 4, 5]

```

Cjeli je sustav u biti vrlo jednostavan kada se jednom shvati što je što, generalno treba zapamtiti da je metoda tip funkcije koje se asocira sa zadanim objektom putem njegove klase, te se može koristiti da dobijemo informacije o objektu ili manipuliramo objektom. Atribut je varijabla u kojoj su spremljene informacije o objektu. Metode i atributi su građevni elementi svake klase. Sljedeći logični zaključak bi bio, da svaki programer može pisati svoje vlastite klase, što znači da može dizajnirati svoje objekte uz one koji su predefimirani u samoj srži python jezika. Na primjer evo jedne klase koja može stvoriti objekt ptice, koja sadrži metodu za hranjenje:

```
class Ptica:
    def __init__(self):
        self.gladna = 1

    def jedi(self):
        if self.gladna:
            print 'Mmmmmmm...'
            self.gladna = 0
        else:
            print 'Ne, hvala!'
```



Slika 4.2: Sage: Primjer korisničke klase

Činjenica da svatko može napisati svoju klasu i time kreirati svoj objekt, je sama srž objektnog programiranja. Na taj način logika programa se djeli na funkcionalne cjeline, te se djelovi koda kroz objekte mogu koristiti na više načina i na različitim mjestima. Svi programi se izvršavaju kroz manifestacije klasa u objekte koje su postavljene u neki oblik interakcije te na taj način čine program, s time da se uzme u obzir da su i sami objekt poput malih nezavisnih programa. Ponekad je potreban samo jedan objekt da se obavi neka zadaća, kao u slučaju sa našom pticom, a ponekad postoje stotine objekata i njihovih klasa koji u kompliciranoj interakciji mogu izvršavati niz zahtjvnih operacija.

4.2 Funkcionalno programiranje

Ovaj stil programiranja je interesantan matematičarima ili ljudima koje koriste Python za proračune i kompjutacije u kontekstu raznih znanstvenih disciplina. Python podržava nekoliko stilova programiranja. Već je spomenut objektni stil programiranja, no postoje i drugi stilovi. Jedan od najstarijih pristupa je proceduralno programiranje, odnosno pisanje koda kao listu instrukcija koje se izvršavaju jedna za drugom.:

```
sage: x = 2+2
sage: x = 2*3
sage: if x%2 == 0:
....:     print 'x je paran broj'
....:
x je paran broj
```

Python modul `operator`, definira nekoliko standardnih aritmetičkih operacija kao funkcije. Zbrajanje je definirano kao ugrađena funkcija `operator.add`, a operacija množenja kao funkcija `operator.mul`. Stoga se varijabla `x` iz prethodnog primjera može napisati kao:

```
sage: from operator import add
sage: from operator import mul
sage: add(2, 2)
```



```
4
sage: mul(2, 3)
6
```

Drugim riječima uz opciju da programski kod grupiramo u klase i pokrećemo kroz objekte, postoji mogućnost da kod grupiramo u manje cjeline, tj. funkcije. Određene karakteristike koje imamo kroz striktno objektnog programiranja se djelomično gube, no pojavljuju se neke nove mogućnosti koje često koristi znanstvena zajednica.

4.2.1 Funkcionalno programiranje uz korištenje `map`

Python ima predefinirane funkcije `map`, `reduce` i `filter` koje podpomažu takav stil programiranja. Funkcija:

```
map(funkcija, seq1, seq2, ...)
```

Uzima funkciju `funkcija`, te sekvencu koja slijedi i zatim primjenjuje `funkcija` na elemente te sekvence. U tom slučaju, rezultat je lista oblika:

```
[funkcija(seq1[0], seq2[0], ...), funkcija(seq1[1], seq2[1], ...), ...]
```

Često korištenje funkcije `map` omogućava izražavanje programske logike na transparentniji i svjesniji način. Također ponekad je znatno lakše čitati i snaći se u kodu. Recimo da imamo dvije liste brojeva koje želimo zbrojiti po elementima. Jedan način na koji to možemo obaviti bi bio:

```
sage: v1 = [1, 4, 5, 7]
sage: v2 = [4, 5, 7, 8]
sage: [v1[i]+v2[i] for i in range(len(v1))]
[5, 9, 12, 15]
```

Analogno možemo iskoristiti spomenutu funkciju zbrajanja `operator.add` u kombinaciji sa funkcijom `map`, kako bi dobili isti rezultat:

```
sage: v1 = [1, 4, 5, 7]
sage: v2 = [4, 5, 7, 8]
sage: map(add, v1, v2)
[5, 9, 12, 15]
```

Prednost koji dobivamo time je da ne moramo provesti petlju po svakom elementu liste i zbrajati, te je kod jednostavniji.

4.2.2 Definiranje malih funkcija koristeći `lambda`

Ponekad je potrebno definirati vrlo malu funkciju:

```
sage: def parnost(var): return var%2
...
```

Alternativa je da koristimo naredbu `lambda` kako bi dobili isti učinak samo uz čišći i jednostavniji kod:

```
sage: par = lambda var: var%2
sage: par(3)
1
```

Da bi kreirali nasumičnu matricu mogli bismo definirati matricu sa praznim mjestima i zatim popuniti svako mjesto pojedinačno nasumičnim elementom:

```
sage: MS = MatrixSpace(ZZ, nrows=6, ncols=4)
sage: MS.random_element() # nasumična matrica

[18  1  0  3]
[-1  1  0 -1]
[-2  0  2  2]
[-2  1  6 -4]
```

```
[ 0  0 -2  0]
[ 1 -1 -1 -3]
```

Alternativa je da koristimo funkciju `random_matrix`:

```
sage: random_matrix(ZZ, nrows=6, ncols=4) # nasumična matrica
[-12  3  4  0]
[  1 -7  0 51]
[  0  1 -1 -16]
[ 10 -2  0  1]
[  0  2  0  1]
[  0  1  4  0]
```

Slijedeći primjer koristi funkciju `map` kako bi generirali listu nasumičnih brojevnih matrica:

```
sage: red = [randint(1, 12) for i in range(12)]
sage: stup = [randint(1, 12) for i in range(12)]
sage: rings = [ZZ]*12
sage: M = map(random_matrix, rings, red, stup)
sage: M[0] # nasumična matrica
[-1 -1 -10  0]
[ 0  0  4 -1]
[-1 166  1  1]
[-1  0  0 -2]
[ 2 -3  2  1]
[-1  0  0  2]
[-1  1 -3  1]
[ 0 -1  2  0]
[ 0 -1 -1  2]
```

Ako želimo više opcija pri definiranju matrice možemo koriatiti naredbu `lambda` u kombinaciji sa funkcijom `map`:

```
sage: nas_red = lambda n: [randint(1, 12) for i in range(n)]
sage: nas_mat = lambda nrows, ncols: [nas_red(ncols) for i in range(nrows)]
sage: matrix(nas_mat(4, 10)) # nasumična matrica
[12  1 10 10 12  7  7  8  7  7]
[ 6  1  8  8 12  7  3 10  1 10]
[11  1  4  8  2  2  3  2  9  2]
[11  7  7 12  4  1  5  1  4  8]

sage: red = [randint(1, 8) for i in range(8)]
sage: stup = [randint(1, 8) for i in range(8)]
sage: M = map(nas_mat, red, stup)
sage: M = map(matrix, M)
sage: M[0] # nasumična matrica
[ 3 12 10  1]
[10 10  7 10]
[ 9 10  4  5]
[10  9  6 10]
[11  4  9  8]
[ 1  4  2  9]
```

4.2.3 Filtriranje koristeći `filter`

U python je ugrađena funkcija `filter` u koju se uvrsti argument i neka sekvenca. Funkcija zatim vraća listu atributa koji su zadovoljili zadani argument. Možemo prikazati primjenu tako da neđemo parne brojeve u zadanom nizu, na način da ćemo primjeniti vlastitu funkciju koja vraća samo parne brojeve

```
def paran(var):
    if var%2 == 0:
        return var
```

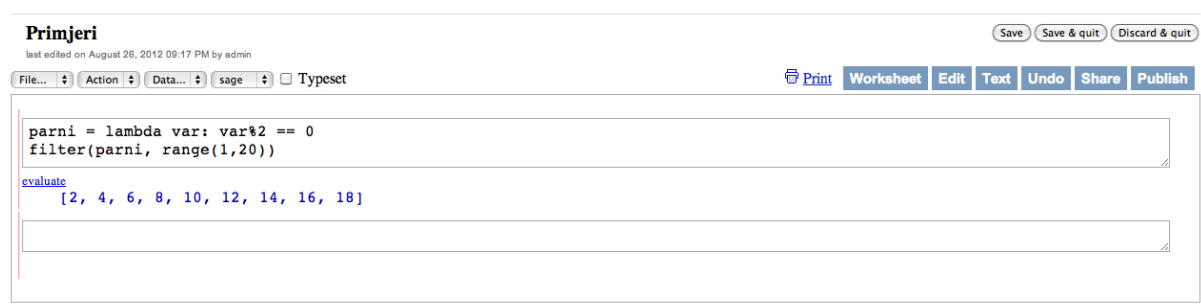
```
filter(paran, [1..30])
```



Slika 4.3: Sage: Funkcija filter

Funkcionalni pristup istom problemu bi bio da se koristi `lambda` kako bi se odredili parni do 20. Zatim može iskoristiti `filter` kako bi dobili listu traženih brojeva:

```
parni = lambda var: var%2 == 0
filter(parni, range(1,20))
```



Slika 4.4: Sage: Kombiniranje funkcija lambda i filter

4.3 Python matematički moduli

4.3.1 NumPy

NumPy je jedan od osnovnih znanstvenih paket koji se može koristiti u Pythonu. Omogućava brze operacija sa poljima, matematičke i logičke prirode. Također su moguće razne operacije sortiranja, primjena diskretnih Fourierovih transformacija, osnovna linearna algebra, statističke operacije, nasumične simulacije i sl.

NumPy nije uključen u Sage automatski te ga je potrebno ručno aktivirati:

```
sage: import numpy
```

Osnovni objekt koji se koristi u NumPy je polje. Definicija polja je jednostavna:

```
sage: array = numpy.array([111, 20, 3])
sage: array
array([111, 20, 3])
```

NumPy polje može sadržavati bilo koji tip python objekta. Kada numpy polje sadrži Sage ili python objekt, tip podataka se eksplicitno prikazuje kao objekt. Ako tip podataka nije specificiran tada će autotaski tip podataka biti definiran kao primitivni tip (float ili int):

```
sage: array = numpy.array([111**20,20**20,3**20])
sage: array
array([80623115361291412000794652124944906491201,
      104857600000000000000000000000, 3486784401], dtype=object)

sage: a=45.00000000000000002
sage: a.prec()
60

sage: numpy.array([a,2*a,3*a])
array([45.000000000000000, 180.00000000000000, 135.00000000000000], dtype=object)
```

Atribut `dtype` polja nam može reći o kakvom tipu podatka je riječ:

```
sage: array = numpy.array([1.0, 2.0, 3.0])
sage: array.dtype
dtype('float64')
```

Možemo kreirati polje specifičnih tipova podatka tako da dodjelimo atributu `dtype` adekvatnu vrijednost.

```
sage: array = numpy.array([4,555,0], dtype=float)
sage: array.dtype
dtype('float64')
```

Elementima NumPy polja možemo pristupiti kao i bilo kojim elementima python liste, također možemo provoditi sve klasične operacije sa listama.

```
sage: array=numpy.array(range(20),dtype=float)
sage: array[17]
17.0

sage: l[5:12]
array([ 5.,  6.,  7.,  8.,  9., 10., 11.]
```

Možemo obavljati klasične aritmetičke operacije:

```
sage: array+array
array([ 0.,  2.,  4.,  6.,  8., 10., 12.,
      14., 16., 18., 20., 22., 24., 26.,
      28., 30., 32., 34., 36., 38.])

sage: 2.5*array
array([ 0.,  2.,  4.,  6.,  8., 10., 12.,
      14., 16., 18., 20., 22., 24., 26.,
      28., 30., 32., 34., 36., 38.])
```

Treba imati na umu da će naredba `l*1` množiti pojedine elemente polja, a ako želimo dobiti skalarni produkt moramo koristiti `numpy.dot`

```
sage: numpy.dot(array,array)
2470.0
```

Možemo napraviti i dvodimenzijaska polja:

```
sage: matrix = numpy.array([[4,3],[5,8]])
sage: matrix
array([[4, 3], [5, 8]])
sage: matrix[0,1]
3
```

Što je u biti ekvivalent:

```
sage: matrix=numpy.matrix([[4,3],[5,8]])
sage: matrix
matrix([[4, 3], [5, 8]])
```

```
sage: m[1,1]
8
```

Razlika je u sljedećem: Kada koristimo `numpy.array()`, `m` se tretira kao polje podatka, što znači kada bi množili $m \times m$, množenje će se provesti po svakoj komponenti polja. No, ako koristimo `numpy.matrix()`, tada će se operacija množenja $m \times m$, provesti kao standardna operacija množenja dvaju matrica. Moguće je zbrajati matrice kao i množiti vektore i matrice.

```
sage: x = numpy.matrix([[4,3],[5,8]],dtype=float)
sage: y = numpy.array([[4],[5]],dtype=float)
sage: x*y
matrix([[ 31.], [ 60.]])
sage: x+x
matrix([[ 8.,  6.], [ 10., 16.]])
```

Sva NumPy polja imaju atribut oblika, `shape`.

```
sage: array = numpy.array(range(10),dtype=float)
sage: array
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.])
sage: array.shape=(2,5)
sage: array
array([[ 0.,  1.,  2.,  3.,  4.],
       [ 5.,  6.,  7.,  8.,  9.]])
```

Ovakvom upotrebom možemo vrlo brzo mjenjati oblike polja, kao u danom primjeru gdje je jednodimenzionalno polje pretvoreno u *2times 5* polje. Vrlo korisna naredba u području strojarstva je `numpy.linalg`, kojom možemo riješavati jednadžbe tipa $Ax=b$

```
sage: import numpy
sage: from numpy import linalg
sage: A=numpy.random.randn(4,4)
sage: b=numpy.array(range(1,5))
sage: x=linalg.solve(A,b)
sage: numpy.dot(A,x)
array([ 1.,  2.,  3.,  4.]])
```

U ovom smo primjeru napravili nasumičnu 4×4 matricu A , te zatim riješavamo jednadžbu $Ax=b$ gdje je $b=[1, 2, 3, 4]$. Postoje još dodatne, specifične metode za riješavanje sustava linearnih jednadžbi i slično. Sve detalje moguće je saznati naredbom `<function name>?` koja nam onda daje uvid u dokumentaciju za dotičnu funkciju.

4.3.2 SciPy

U velikoj mjeri paket sličan NumPy-u. Okruženje u kojem postoje moduli za statistiku, optimizaciju, integraciju, linearnu algebru, Fourierove transformacije, analizu signala itd. SciPy je u biti nadogradnja NumPy paketa, te u potpunosti ovisi NumPy sustavu manipulacije poljima. Konkretnije, SciPy je napravljen kao proširenje NumPy naredbi za obavljanje operacije nad poljima. Postoje mnogi korisni SciPy moduli, poput: `scipy.optimize`, `scipy.stats`, `scipy.linalg`, `scipy.linalg`, `scipy.sparse`, `scipy.integrate`, `scipy.fftpack`, `scipy.signal`, `scipy.special`. Većina ima vrlo dobru dokumentaciju i često je iz samog imena shvatljivo kako i što modul radi. Kao i NumPy paket SciPy paket je potrebno ručno uključiti program.

```
sage: import scipy
sage: from scipy import optimize
```

Vrlo lako možemo doznati koje funkcije sadrži module `optimize`

```
sage: optimize.[tab]

sage: optimize.
optimize.Tester           optimize.diagbroyden     optimize.linesearch
optimize.anderson        optimize.excitingmixing  optimize.minpack
```

```

optimize.anderson2
optimize.anneal
optimize.approx_fprime
optimize.bench
optimize.bisect
optimize.bracket
optimize.brent
optimize.brenth
optimize.brentq
optimize.broyden1
optimize.broyden2
optimize.broyden3
optimize.broyden_generalized
optimize.brute
optimize.check_grad
optimize.cobyla
optimize.curve_fit
optimize.fixed_point
optimize.fmin
optimize.fmin_bfgs
optimize.fmin_cg
optimize.fmin_cobyla
optimize.fmin_l_bfgs_b
optimize.fmin_ncg
optimize.fmin_powell
optimize.fmin_slsqp
optimize.fmin_tnc
optimize.fminbound
optimize.fsolve
optimize.golden
optimize.lbfgsb
optimize.leastsq
optimize.line_search
optimize.minpack2
optimize.moduleTNC
optimize.newton
optimize.newton_krylov
optimize.nnls
optimize.nonlin
optimize.optimize
optimize.ridder
optimize.rosen
optimize.rosen_der
optimize.rosen_hess
optimize.rosen_hess_prod
optimize.slsqp
optimize.test
optimize.tnc
optimize.zeros

```

Modul `optimize` sadrži mnogobrojne funkcije, kao i do sad za pojedinačnu funkciju dovoljno je upisati `optimize.fmin_cg`? i možemo saznati kako funkcija radi i kako ju koristiti. Vrlo nam je zanimljiv i modul `scipy.integrate`, koji sadrži funkcije za numeričko rješavanje diferencijalnih jednačbi, tj za numeričku integraciju. Možemo prikazati kako radi rješavanjem primjera:

$$x''(t) + ux'(t)(x(t)^2 - 1) + x(t) = 0$$

Koji možemo zapisati kao sustav:

$$x' = y$$

$$y' = -x + \mu y(1 - x^2).$$

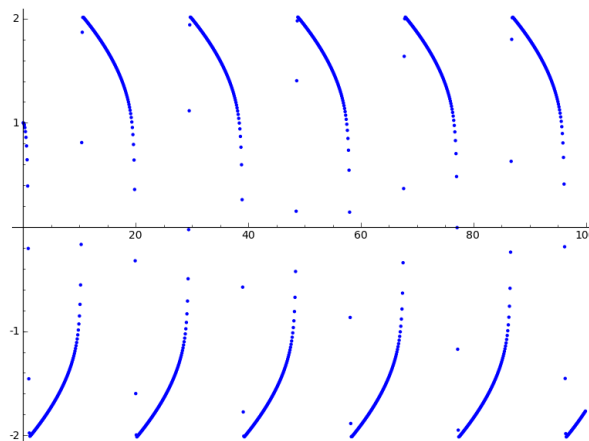
Potrebna nam je funkcija `odeint` koja se nalazi u `scipy.integrate` modulu. Možemo riješiti jednačbu za 1000 točaka u intervalu od 0, do 100 koristeći kod:

```

import scipy
from scipy import integrate
def f_1(y,t):
    return [y[1], -y[0]-10*y[1]*(y[0]**2-1)]
def j_1(y,t):
    return [ [0, 1.0], [-2.0*10*y[0]*y[1]-1.0, -10*(y[0]*y[0]-1.0)] ]
x= scipy.arange(0,100,.1)
y=integrate.odeint(f_1, [1,0], x, Dfun=j_1)
p=[point((x[i],y[i][0])) for i in range(len(x))]
plot(p).show()

```

Za dotično rješenje smo i napravili grafički ispis za zadani interval.



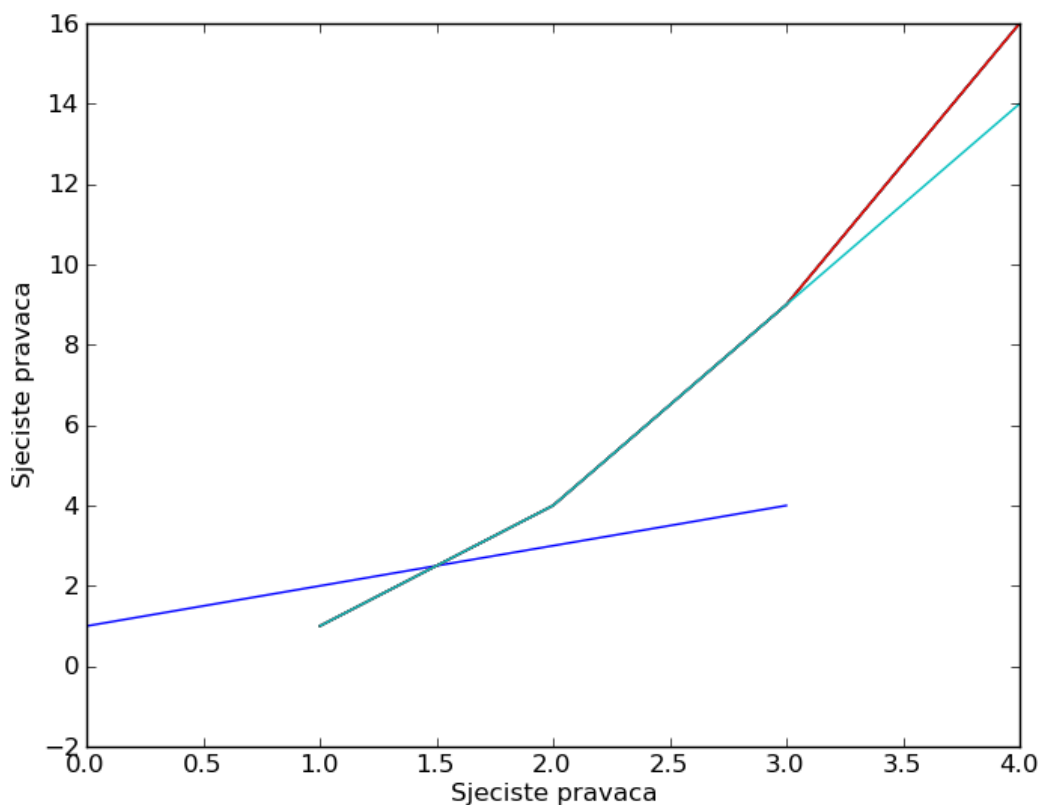
Slika 4.5: Sage: Grafički prikaz rješenja

4.3.3 Matplotlib

U inženjerskim strukama, vizualizacija je ključni korak. Grafički prikaz matematičkih konstrukata ili fizikalnih procesa znatno pomaže pri razumjevanju istih. U kontekstu raznih navedenih matematičkih alata, postoje mnogi alati za prikazivanje. U python su može koristiti Matplotlib. Ovaj paket se koristi za 2D crtanje visoke kvalitete. Pomoću ovog paket možemo kreirati razne, grafove, histograme, mape, ispise polja i sve to sa setom vrlo jednostavnih funkcija i naredbi.

matplotlib.pyplot je kolekcija funkcija kojima se matplotlib ponaša poput matlaba, te se pristup naučen u matlabu može koristiti kroz ovaj modul.

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4], [1,4,9,14])
plt.ylabel('Sjeciste pravaca')
plt.xlabel('Sjeciste pravaca')
plt.savefig('eg_1.png')
```



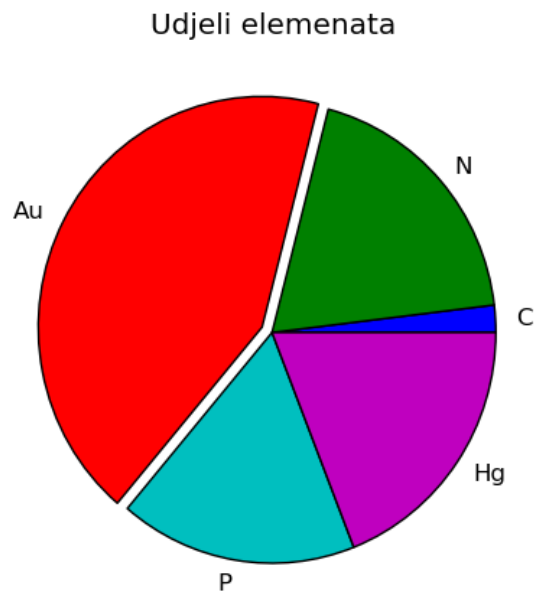
Slika 4.6: Sage: Korištenje Matplotlib paketa

Postoje mnogi dodatni moduli unutar matplotlib paketa koji su vrlo jednostavni za korištenje, te ću ovdje kroz nekoliko primjera prikazati neke osnovne mogućnosti:

Takozvani “Pie chart”:

```
import numpy
import matplotlib.pyplot as plt
data = [2, 20.0, 45.0, 18.0, 20.0]
explode = numpy.zeros(len(data))
explode[2] = 0.05
plt.figure(figsize=(5, 5))
plt.pie(data, explode=explode, labels=['C', 'N', 'Au', 'P', 'Hg'])
```

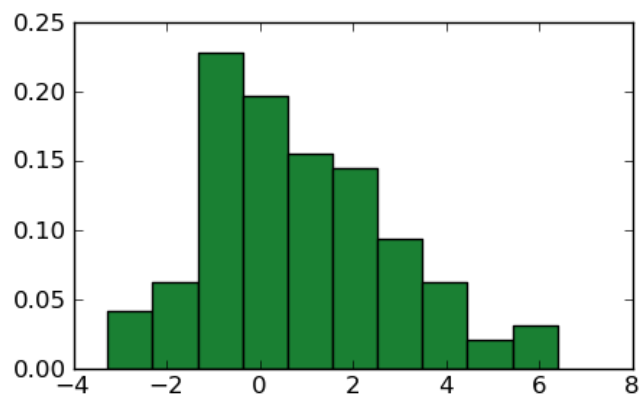
```
plt.title('Udjeli elemenata')
plt.savefig('Pie_chart.png')
plt.close()
```



Slika 4.7: Sage: Korištenje "Pie Chart" prikaza

Histogram:

```
import numpy
import matplotlib.pyplot as plt
data = numpy.random.normal(1, 2, size=100)
plt.figure(figsize=(5, 3))
plt.hist(data, normed=True, facecolor=(0.1, 0.5, 0.2))
plt.savefig('Histogram.png')
plt.close()
```



Slika 4.8: Sage: Histogramski prikaz

3D dijagram:


```

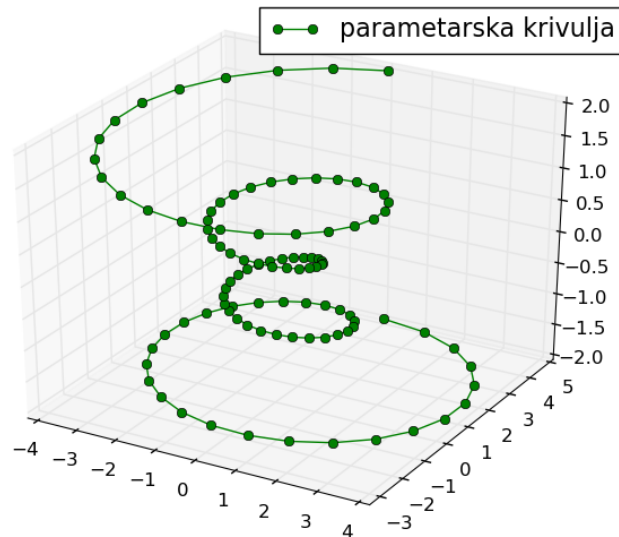
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt

mpl.rcParams['legend.fontsize'] = 16

fig = plt.figure()
ax = fig.gca(projection='3d')
theta = np.linspace(-4 * np.pi, 4 * np.pi, 100)
z = np.linspace(-2, 2, 100)
r = z**2 + 1
x = r * np.sin(theta)
y = r * np.cos(theta)
ax.plot(x, y, z, color='green', marker='o', label='parametarska krivulja')
ax.legend()

plt.savefig('matplot_3d.png')
plt.close()

```



Slika 4.9: Sage: 3D koordinatni sustav

LATEX I SAGE

Objasniti kako latex funkcioniра u Sage okruženju je najlakše ako se prvo ukratko upoznamo sa tri osnovna mehanizma korištenja latexa u Sageu.

1. Svaki objekt koji se koristi u Sageu ima svoju latex inačicu odnosno reprezentaciju. Možemo pristupiti tim inačicama kroz grafičko sučelje ili kroz CLI sučelje, na način: `latex(foo)` gdje je `foo` neki objekt u Sageu. Kao ispis dobijemo niz, koji predstavlja TeX inačicu Sage objekta `foo` u matematičkom okruženju (npr. zatvoreno sa znakovima `$. . . $`). Na taj način kroz Sage možemo na učinkovit način kreirati djelove latex dokumenta kao rezultate kompliciranih matematičkih proračuna.
2. Grafičko sučelje je konfigurirano za korištenje JsMath paketa, koji se koristi za učitavanje latex entiteta iz matematičkog okruženja u web preglednik. JsMath je skup JavaScript naredbi i povezanih fontova. Obično su ti fontovi učitani sa servera direktno u web preglednik, zajedno sa web stranicom koja ih prikazuje. JsMath je u stanju učitati velik broj TeX naredbi, no u konačnici nije tolikog kapaciteta kao i sam TeX. Ne može ispisivati velike i komplicirane tablice, nema mogućnosti upravljanja sadržajem dokumenta u onoj mjeri na kojoj smo to navikli koristeći latex. Kako mu ime kaže, primarna zadaća je prikazivanje matematičkih formula u web pregledniku
3. Treći način korištenja latexa u Sageu je već spomenuto korištenje latex paketa. Kada je kod koji koristimo za kreiranje našeg latex dokumenta znatno kompliciraniji od onog što jsMath može procesuirati, možemo povezati na sistemsku instalaciju latexa. Na taj način doživljavamo Sage samo kao mjesto gdje ubacujemo kod koji onda on predaje lokalno varijnti latexa odnosno TeXa. Stoga ako želimo kroz Sage u potpunosti koristiti latex, potrebno je imati instaliranu verziju latexa na istom računalu na kojem je instaliran i Sage.

Najbolje je objašnjenje nadopuniti primjerom, stoga evo jednog primjera korištenja `latex()` funkcije:

```
Sage: var('x')
x
Sage: latex(sqrt(x))
\sqrt{x}
Sage: latex(integrate(x^(x+1), x))
\int x^{x + 1}\, {d x}
Sage: latex('x string')
\verb|x|\phantom{x}\verb|string|
Sage: latex(ZZ)
\Bold{Z}
Sage: latex(matrix(ZZ, 4, 4, [[2,4,6,4], [-1,-1,-1,-1], [0,0,0,1], [1,2,3,4]]))
\left (\begin{array}{rrrr}
2 & 4 & 6 & 4 \\
-1 & -1 & -1 & -1 \\
0 & 0 & 0 & 1 \\
1 & 2 & 3 & 4
\end{array}\right)
```

Osnovna jsMath funkcionalnost je dobrim djelom automatska kada ju koristimo u grafičkom okruženju. Možemo djelom prikazati kako jsMath radi ako pokrenemo `eval` funkciju, te na taj način možemo vidjeti pretvorbu Sage objekta u latex inačicu, koji je pak okružen HTML elementim koji pomoću CSS-a onda adekvatno prikazuju latex elemente.

```

Sage: from Sage.misc.latex import JSMath
Sage: js = JSMath()
Sage: var('z')
z
Sage: js(sqrt(x))
<html><div class="math">\newcommand{\Bold}[1]{\mathbf{#1}}\sqrt{x}</div></html>
Sage: js(ZZ)
<html><div class="math">\newcommand{\Bold}[1]{\mathbf{#1}}\Bold{Z}</div></html>
Sage: js(QQ[x])
<html><div class="math">\newcommand{\Bold}[1]{\mathbf{#1}}\Bold{Q}[x]</div></html>
Sage: js(integrate(x^(x+1), x))
<html><div class="math">\newcommand{\Bold}[1]{\mathbf{#1}}\int x^{x + 1}\,, {d x}</div></html>

```

5.1 Primjena

U kratkom pregledu svih metoda kojima je latex integriran u Sage, prikazana je funkcija `latex()`, za koju bi mogli reći da je najjednostavniji način za generiranje latex sintakse. Tako genreirane latex naredbe možemo integrirati u samostalne latex dokumente.

Kada želimo dobiti latex sintaksu koju bi interpretirao jsMath, potrebno je koristiti funkciju `view(foo)`. Na taj način ćemo dobiti kombinaciju html-a i css-a koju onda možemo prikazati u web okruženju poput onog prisutnog u Sage grafičkom okruženju. Kada koristimo Sage u grafičkom okruženju, možemo aktivirati kućicu “Typeset”, te na taj način će sva rješenja koja dobijemo iz programskih proračuna bit prikazana kao da smo pozvali `view()` funkciju.

Možemo na nekoliko načina manipulirati latex kodom koji generira Sage. Na primjer kreirati i uređivati matrice:

```

Sage: M = matrix(QQ, 4, 4, range(1))
Sage: latex(A)
\left(\begin{array}{rrrr}
0 & 1 & 2 & 3 \\
4 & 5 & 6 & 7 \\
8 & 9 & 10 & 11 \\
12 & 13 & 14 & 15
\end{array}\right)
Sage: latex.matrix_delimiters(left='[' , right=']')
Sage: latex(M)
\left[\begin{array}{rrrr}
0 & 1 & 2 & 3 \\
4 & 5 & 6 & 7 \\
8 & 9 & 10 & 11 \\
12 & 13 & 14 & 15
\end{array}\right]
Sage: latex.matrix_delimiters(left='\\{', right='\\}')
Sage: latex(A)
\left\\{\begin{array}{rrrr}
0 & 1 & 2 & 3 \\
4 & 5 & 6 & 7 \\
8 & 9 & 10 & 11 \\
12 & 13 & 14 & 15
\end{array}\right\\}

```

Kao i u latexu, koristeći `latex()` funkciju možemo uključivati pakete, slagati zaglavlje latex dokumenta i uključivati makro naredbe. Možemo dodavati bilo što u zaglavlje dokumenta koristeći funkciju `latex.add_to_preamble`. U sljedećem primjeru dodajemo gemoterijski paket.

```

Sage: latex.extra_preamble('')
Sage: latex.extra_preamble()
''
Sage: latex.add_to_preamble('\\usepackage{foo-bar-unchecked}')
Sage: latex.extra_preamble()

```

```
'\usepackage{foo-bar-unchecked}'
Sage: latex.add_package_to_preamble_if_available('foo-bar-checked')
Sage: latex.extra_preamble()
'\usepackage{foo-bar-unchecked}'
```

Lako se dogodi da pokušamo učitati paket koji nam nije dostupan, Sage ima integriranu provjeru unutar `latex()` funkcije, a provjeravamo koristeći `latex.add_package_to_preamble_if_available`.

```
Sage: latex.extra_preamble('')
Sage: latex.extra_preamble()
''
Sage: latex.add_to_preamble('\usepackage{foo-bar-unchecked}')
Sage: latex.extra_preamble()
'\usepackage{foo-bar-unchecked}'
Sage: latex.add_package_to_preamble_if_available('foo-bar-checked')
Sage: latex.extra_preamble()
'\usepackage{foo-bar-unchecked}'
```

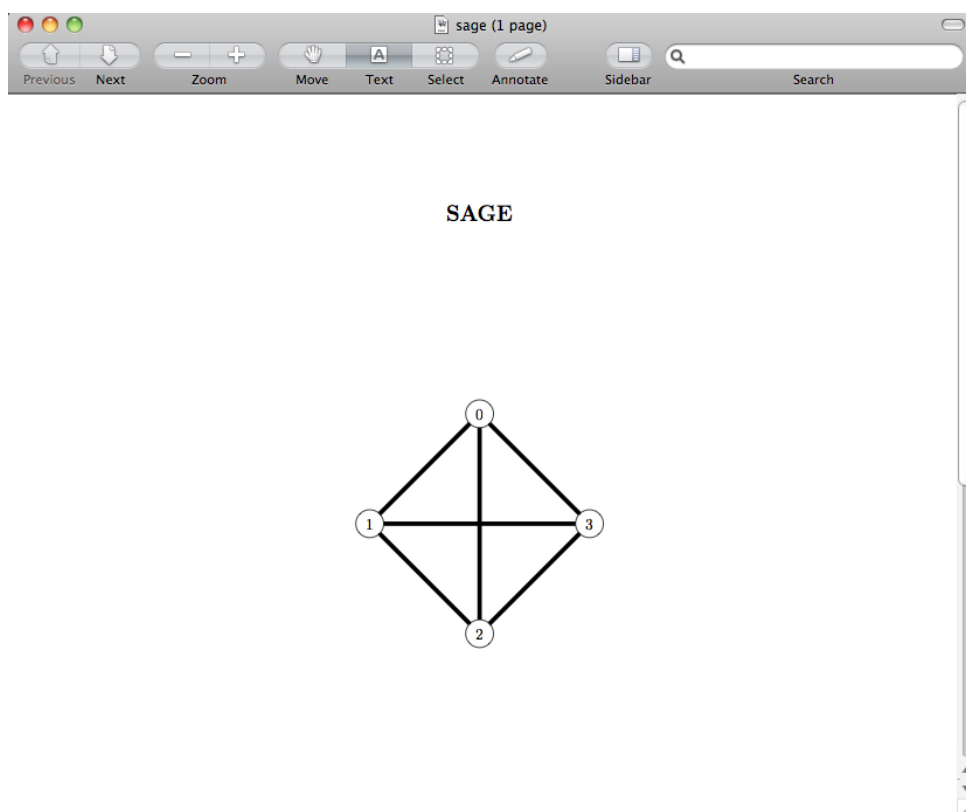
Također je moguće kontrolirati koja varijanta TeXa se koristi za invokaciju na razini operacijskog sustava, a time i utjecati na prirodu izlaz. Isto tako je moguće kontrolirati kad će se koristiti jsMath (za jednostavnije latex elemente), a kada globalna TeX instalacija (složeniji LaTeX izrazi). Ponekad može biti potrebno zaobići jsMath interpreter kako bi se postigli kompliciraniji latex oblici.

```
Sage: latex.jsmath_avoid_list([])
Sage: latex.jsmath_avoid_list()
[]
Sage: latex.jsmath_avoid_list(['foo', 'bar'])
Sage: latex.jsmath_avoid_list()
['foo', 'bar']
Sage: latex.add_to_jsmath_avoid_list('tikzpicture')
Sage: latex.jsmath_avoid_list()
['foo', 'bar', 'tikzpicture']
Sage: latex.jsmath_avoid_list([])
Sage: latex.jsmath_avoid_list()
[]
```

Latex je poznat po svojoj mogućnosti da generira lijepe dokumente sa kvalitetnim grafičkim prikazima. Ljepe grafove možemo dobiti koristeći `tkz-graph` paket, koji je potrebno uključiti u zaglavlje dokumenta.

```
Sage: from Sage.graphs.graph_latex import setup_latex_preamble
Sage: setup_latex_preamble()
Sage: latex.extra_preamble()
'\usepackage{tikz}\n\usepackage{tkz-graph}\n\usepackage{tkz-berge}\n'
Sage: latex.engine('pdflatex')
Sage: latex.add_to_jsmath_avoid_list('tikzpicture')
Sage: latex.jsmath_avoid_list()
['tikzpicture']
```

Pokretanjem naredbe poput, `view(graphs.CompleteGraph(4))`, dobiti ćemo pdf ekranizaciju grafa u pdf dokumentu. Drugim riječima sve karaktersitike Latex su dostupne kroz Sage, a listu naredbi možemo dobiti ako upišem funkciju `latex()` i zatim stisnemo tipku “tab”.



Slika 5.1: Sage: Graf prikazan u pdf dokumentu

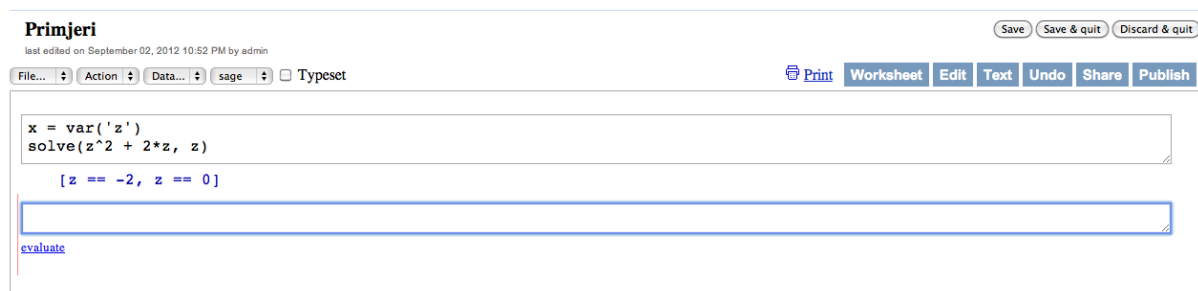
PRIMJENA U NASTAVI FSB-A I NASTAVI TEHNIČKE PRIRODE

Cilj Sage alata je približiti moderne programske i numeričke metode studentima koji sudjeluju u nastavi tehničke prirode. U slučaju kada se koristi Sage se ne ovisi o raznim licencama i financijama potrebnim za postavljanje jednog računalnog laboratorija, ili u krajnjem slučaju kako je objašnjeno ovisi se samo o jednoj licenci. Dakle jedna od ključnih karakteristika sage je činjenica da on može koristiti sve metode i alate koji su integrirani u njegovo okruženje. Ti alati ne uključuju samo aplikacije matematičkog tipa, već i alate poput latex-a, možemo upisivati direktno html sintaksu ili python sintaksu. U kontekstu striktno matematičkih proračuna već sam Sage alat ima veliku paletu mogućnosti za izračunavanje raznih matematičkih problema. Sage može obavljati različite izračune algebre i infinitezimalnog računa poput derivacija, integracije i Laplaceovih transformacija. Matematika kao takva je ključna u strojarskoj praksi, te generalno u nastavi FSB-a. U sljedećih par primjera se može vidjeti kako se Sage može koristiti za elementarne matematičke izračune.

6.1 Riješavanje jednadžbi

Naredba `solve` riješava jednadžbe. Da bi se koristila, prvo je potrebno odrediti neke varijable, a zatim su argumenti funkcije `solve` jednadžba (ili sustav jednadžbi), zajedno s varijablama za koje se sustav riješava:

```
x = var('z')
solve(z^2 + 2*z, z)
```



Slika 6.1: Sage: riješavanje jednostavne jednadžbe

Možemo riješavati sustav sa jednom nepoznicom.

```
a,b,c = var('a b c')
solve([e^x + b*x + c == 0], x)
```

Moguće je riješavati i za sustav sa više varijabli

```
x, y = var('x, y')
solve([x+y==12, x-y==34], x, y)
```

The screenshot shows the SageMath web interface. At the top, it says "Primjeri" and "last edited on September 02, 2012 10:52 PM by admin". There are buttons for "Save", "Save & quit", and "Discard & quit". Below that are menu items: "File...", "Action", "Data...", "sage", and "Typeset". On the right, there are buttons for "Print", "Worksheet", "Edit", "Text", "Undo", "Share", and "Publish". The main input area contains the following code:

```
a,b,c = var('a b c')
solve([e^x + b*x + c == 0], x)

[z == -(c + e^z)/b]
```

Below the code is an empty output area and a blue "evaluate" button.

Slika 6.2: Sage: sustav jednačbi sa jednom nepoznanicom

The screenshot shows the SageMath web interface. At the top, it says "Primjeri" and "last edited on September 02, 2012 10:52 PM by admin". There are buttons for "Save", "Save & quit", and "Discard & quit". Below that are menu items: "File...", "Action", "Data...", "sage", and "Typeset". On the right, there are buttons for "Print", "Worksheet", "Edit", "Text", "Undo", "Share", and "Publish". The main input area contains the following code:

```
x, y = var('x, y')
solve([x+y==12, x-y==34], x, y)

[[x == 23, y == -11]]
```

Below the code is an empty output area and a blue "evaluate" button.

Slika 6.3: Sage: sustav jednačbi sa više nepoznanica

Ovaj primjer pokazuje kako Sage može riješiti sustav ne linearnih jednačbi:

```
var('x, y, p, q')
jed1 = p+q==9
jed2 = q*y+p*x== -6
jed3 = q*y^2+p*x^2==120
solve([jed1, jed2, jed3, p==1], p, q, x, y)
```

The screenshot shows the SageMath web interface. At the top, it says "Primjeri" and "last edited on September 02, 2012 10:52 PM by admin". There are buttons for "Save", "Save & quit", and "Discard & quit". Below that are menu items: "File...", "Action", "Data...", "sage", and "Typeset". On the right, there are buttons for "Print", "Worksheet", "Edit", "Text", "Undo", "Share", and "Publish". The main input area contains the following code:

```
var('x y p q')
jed1 = p+q==9
jed2 = q*y+p*x== -6
jed3 = q*y^2+p*x^2==120
solve([jed1, jed2, jed3, p==1], p, q, x, y)

[[p == 1, q == 8, x == -4/3*sqrt(58) - 2/3, y == 1/6*sqrt(2)*sqrt(29) - 2/3], [p == 1, q == 8, x == 4/3*sqrt(58) - 2/3, y == -1/6*sqrt(2)*sqrt(29) - 2/3]]
```

Below the code is an empty output area and a blue "evaluate" button.

Slika 6.4: Sage: Sustav ne lineranih jednačbi

Također je moguće dobiti numeričku aproksimaciju rješenja:

```
solns = solve([jed1, jed2, jed3, p==1], a, b, c, d, solution_dict=True)
[[s[a].n(15), s[b].n(15), s[c].n(15), s[d].n(15)] for s in solns]
```

The screenshot shows a Sage Notebook window titled "Primjeri". The code in the input field is:

```

var('x,y,p,q')
jed1 = p+q==9
jed2 = q*y+p*x== -6
jed3 = q*y^2+p*x^2==120
solve([jed1,jed2,jed3,p==1],p,q,x,y)

```

The output shows the solutions:

```

[[p == 1, q == 8, x == -4/3*sqrt(58) - 2/3, y == 1/6*sqrt(2)*sqrt(29) - 2/3], [p == 1, q == 8, x == 4/3*sqrt(58) - 2/3, y == -1/6*sqrt(2)*sqrt(29) - 2/3]]

```

Below the code, the user has executed a command to format the solutions:

```

solns = solve([jed1,jed2,jed3,p==1],p,q,x,y, solution_dict=True)
[[s[p].n(15), s[q].n(15), s[x].n(15), s[y].n(15)] for s in solns]

```

The resulting numerical solutions are displayed as:

```

[[1.000, 8.000, -10.82, 0.6026], [1.000, 8.000, 9.488, -1.936]]

```

Slika 6.5: Sage: numerička aproksimacija rješenja

6.2 Numeričke aproksimacije

Često `solve` ne može dobiti točno rješenje za zadani sustav jednačbi, u slučaju kada podbaci možemo koristiti `find_root` kako bi dobili numeričku aproksimaciju. Na primjer, za zadani sustav `solve` nam ne vraća ništa zanimljivo.

```

alfa = var('alfa')
solve(2^cos(alfa)==2^sin(alfa), theta)

```

The screenshot shows a Sage Notebook window titled "Primjeri". The code in the input field is:

```

alfa = var('alfa')
solve(2^cos(alfa)==2^sin(alfa), alfa)

```

The output shows the equation that could not be solved:

```

[2^sin(alfa) == 2^cos(alfa)]

```

Slika 6.6: Sage: Jednadžba ne riješiva funkcijom solve

Stoga koristimo `find_root` kako bi našli rješenje za navedeni sustav u rasponu $0 < \phi < \pi/2$

```

phi = var('phi')
find_root(2^cos(alfa)==2^sin(alfa), -1,1)

```

The screenshot shows a Sage Notebook window titled "Primjeri". The code in the input field is:

```

alfa = var('alfa')
find_root(2^cos(alfa)==2^sin(alfa), -1,1)

```

The output shows the numerical solution:

```

0.7853981633974484

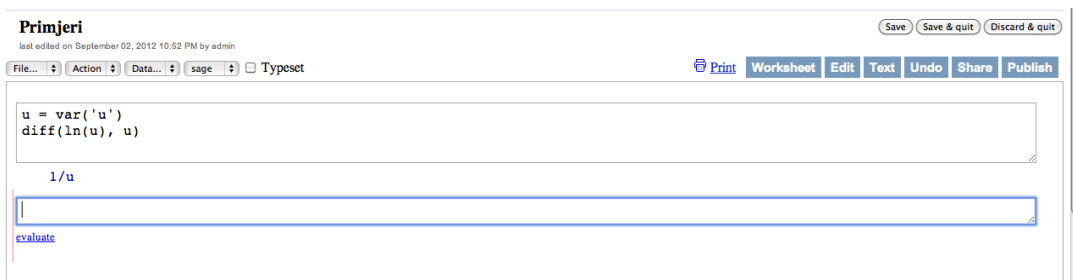
```

Slika 6.7: Sage: Numerička aproksimacija funkcijom find_root

6.3 Deriviranje, Integriranje, itd

Infinitezimalni račun se razvio u kontekstu Newtonovske fizike, paralelno sa Leibnitzovim radom na istom području. Rezultat je da sva područja mehanike, dinamika, statika, sudari i slično, koriste infinitezimalan račun za objašnjavanje takvih stanja. Sage dolazi sa brojnim alatima koji su u stanju riješavati takve probleme. Sage zna kako derivirati i integrirati mnoge funkcije.

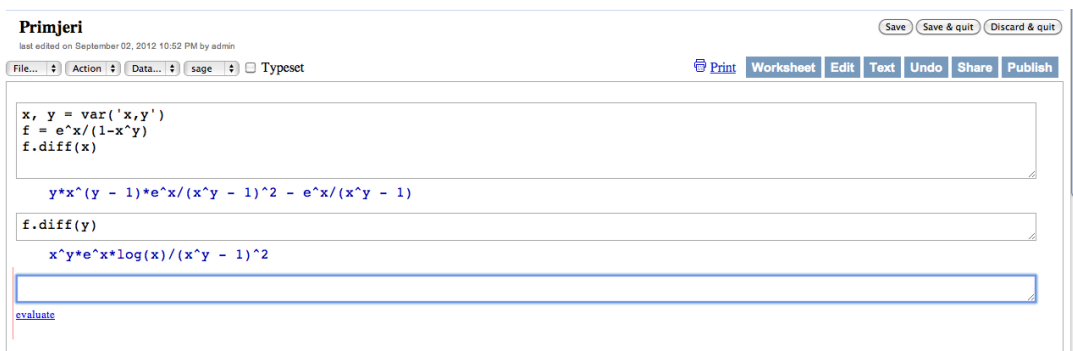
```
u = var('u')
diff(ln(u), u)
```



Slika 6.8: Sage: Jednostavna derivacija

Moguće je derivirati i kompliciranije izraze, $e^x/(1-x)$ po x i y :

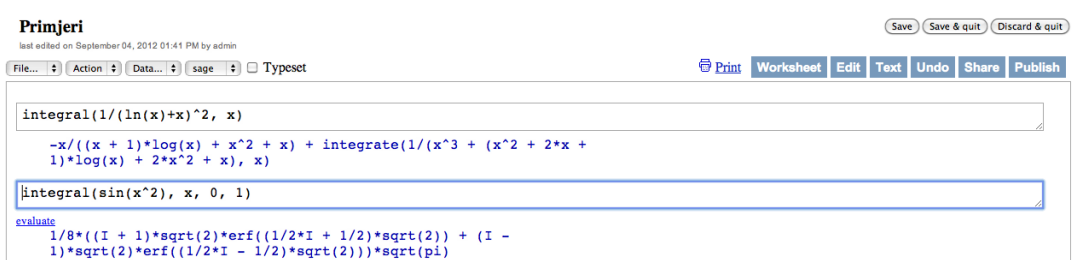
```
x, y = var('x,y')
f = e^x/(1-x)
f.diff(x)
f.diff(y)
```



Slika 6.9: Sage: Parcijalna derivacija

Naravno podržana je integracija neodređenih i određenih integrala:

```
integral(1/(ln(x)+x)^2, x)
integral((x^(1-x))/x^2, x, 0, 1)
```

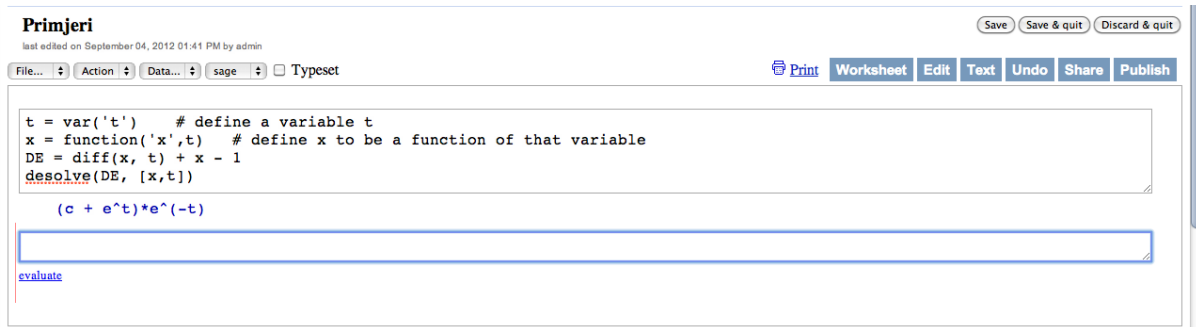


Slika 6.10: Sage: Integracija

6.4 Riješavanje diferencijalnih jednađbi

Mnogi problemi tehničke struke se opisuju diferencijalnim jednađbama, komplicirani sustavi statike, vibracije, analiza signala i sl. U kontekstu nastave FSB-a studenti se susreću sa diferencijalnim jednađbama kroz automatu, robotiku, itd. Generalno možemo reći da je priroda oko nas opisana tim analognim matamtičkim zapisom. Često se za riješenje nekog problem trebaju postaviti i zatim riješiti komplicirane diferencijalne jednađbe. Nekada se to radilo ručno, a danas zahvaljujući računalima, takvi se problemi mogu riješavati puno brže i efikasnije, te Sage ima moćne alate koji nam u tome pomažu.

Riješimo jednađbu, $x' + x - 1 = 0$:



The screenshot shows the SageMath web interface. At the top, there's a header with 'Primjeri' and 'last edited on September 04, 2012 01:41 PM by admin'. Below that are navigation buttons: 'File...', 'Action', 'Data...', 'sage', and 'Typeset'. On the right, there are buttons for 'Print', 'Worksheet', 'Edit', 'Text', 'Undo', 'Share', and 'Publish'. The main area contains a code editor with the following code:

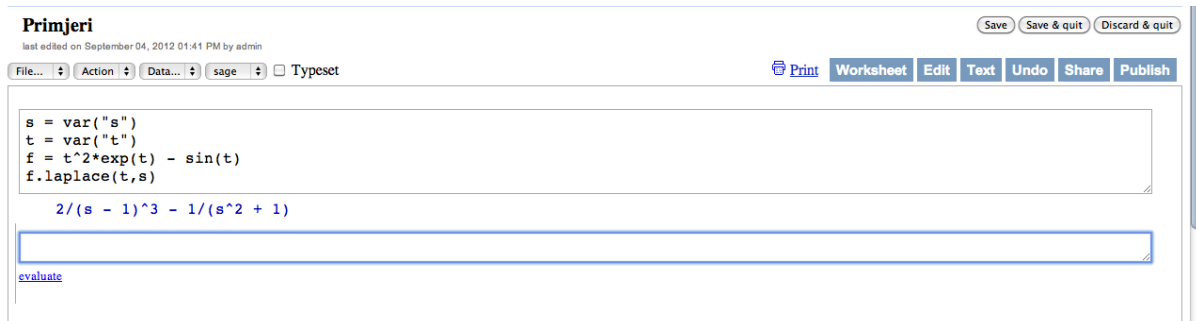
```
t = var('t') # define a variable t
x = function('x',t) # define x to be a function of that variable
DE = diff(x, t) + x - 1
desolve(DE, [x,t])
```

The output of the code is $(c + e^t) \cdot e^{-t}$. Below the code editor is an 'evaluate' button.

Slika 6.11: Sage: Jednostavna diferencijalna jednađba

Dobivamo riješenje: $x(t) = e^{-t}(e^t + c)$.

U automati i autmatskom upravljanju se obilato koriste Laplaceove transformacije, koje se relativno jednostavno mogu izračunati u Sageu. Možemo izračunati Laplaceovu transformaciju izraza $t^2e^t - \sin(t)$:



The screenshot shows the SageMath web interface. At the top, there's a header with 'Primjeri' and 'last edited on September 04, 2012 01:41 PM by admin'. Below that are navigation buttons: 'File...', 'Action', 'Data...', 'sage', and 'Typeset'. On the right, there are buttons for 'Print', 'Worksheet', 'Edit', 'Text', 'Undo', 'Share', and 'Publish'. The main area contains a code editor with the following code:

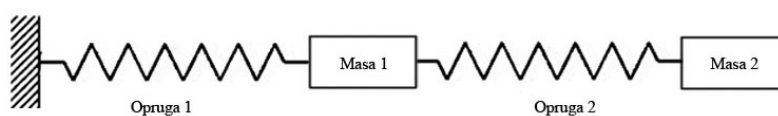
```
s = var("s")
t = var("t")
f = t^2*exp(t) - sin(t)
f.laplace(t,s)
```

The output of the code is $2/(s - 1)^3 - 1/(s^2 + 1)$. Below the code editor is an 'evaluate' button.

Slika 6.12: Sage: Laplaceova transformacija

Možemo postaviti i malo praktičniji primjer u obliku problema spjenih opruga koje se uzbuđuju iz stanja mirovanja. Problem je opisan sustavom diferencijalnih jednađbi drugog reda.

$$\begin{aligned} m_1 x_1'' + (k_1 + k_2)x_1 - k_2 x_2 &= 0 \\ m_2 x_2'' + k_2(x_2 - x_1) &= 0, \end{aligned}$$

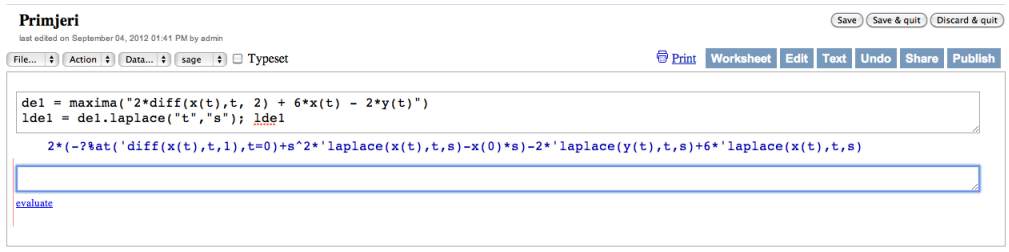


Slika 6.13: Sage: Sustav opruga

Gdje je m_i masa tereta i , x_i je količina pomaka od točke mirovanja za teret i i k_i je faktor elastičnosti opruge i . Koristeći Sage je sada potrebno riješiti ovaj problem sa zadanim parametrima: $m_1 = 2$, $m_2 = 1$, $k_1 = 4$, $k_2 = 2$, $x_1(0) = 3$, $x_1'(0) = 0$, $x_2(0) = 3$, $x_2'(0) = 0$.

Prvo računamo Laplaceovu transformaciju za prvu jednadžbu sa točnim indexima $x = x_1$, $y = x_2$:

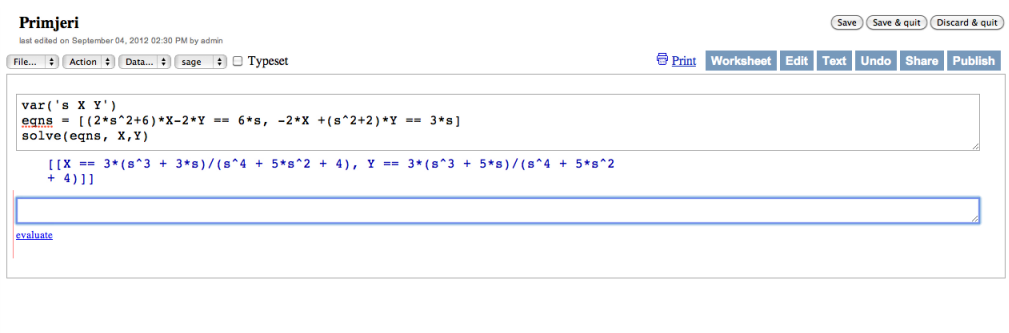
```
de1 = maxima("2*diff(x(t),t, 2) + 6*x(t) - 2*y(t)")
lde1 = de1.laplace("t","s"); lde1
```



Slika 6.14: Sage: Laplace prve jednadžbe

Rezultat je: $-2x'(0) + 2s^2 * X(s) - 2sx(0) - 2Y(s) + 6X(s) = 0$. Analogno dobivamo i Laplaceovu transformaciju druge jednadžbe nakon toga možemo uvrstiti inicijalne uvjete, te riješiti dobiveni sustav jednadžbi:

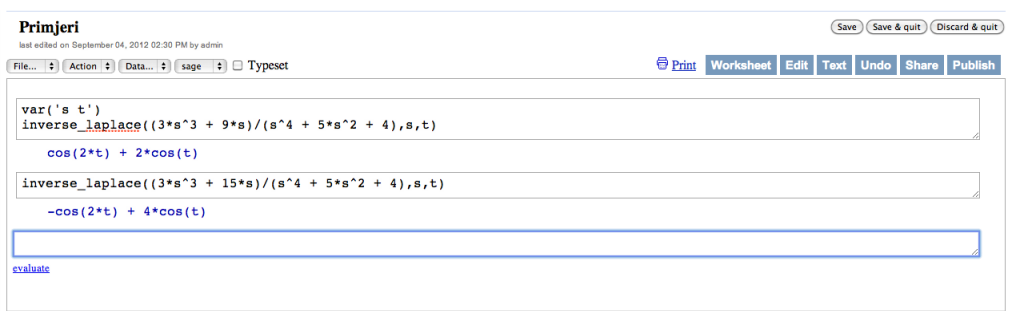
```
var('s X Y')
eqns = [(2*s^2+6)*X-2*Y == 6*s, -2*X + (s^2+2)*Y == 3*s]
solve(eqns, X, Y)
```



Slika 6.15: Sage: Riješenje sustava opruge

Kako bi dobili konačno rješenje moramo izračunati inverznu Laplaceovu transformaciju za dobiveni izraz:

```
var('s t')
inverse_laplace((3*s^3 + 9*s)/(s^4 + 5*s^2 + 4),s,t)
inverse_laplace((3*s^3 + 15*s)/(s^4 + 5*s^2 + 4),s,t)
```



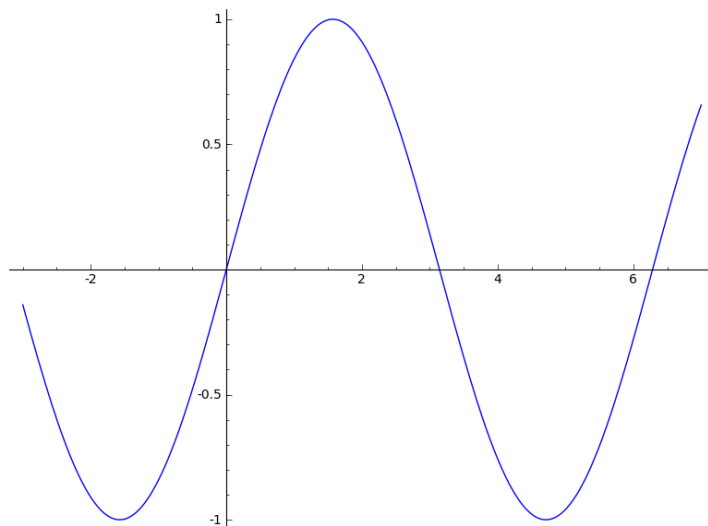
Slika 6.16: Sage: Inverzna Laplaceova transformacija rješenja

Stoga, rješenje je: $x_1(t) = \cos(2t) + 2 \cos(t)$, $x_2(t) = 4 \cos(t) - \cos(2t)$.

6.5 Grafika

Kao što je prikazano u prethodnom primjeru Sage ima ugrađene alate za crtanje. Grafički prikazi nekih procesa ili sustava se često javljaju u nastavi tehničkih struka, te su ključni pri interpretaciji rješenja ili postavljanju problema. Sage ima mogućnost prikaza raznih grafičkih struktura u 2D i 3D okruženju. Nama su možda najzanimljiviji grafički prikazi raznih funkcija, koje je jednostavno možemo dobiti:

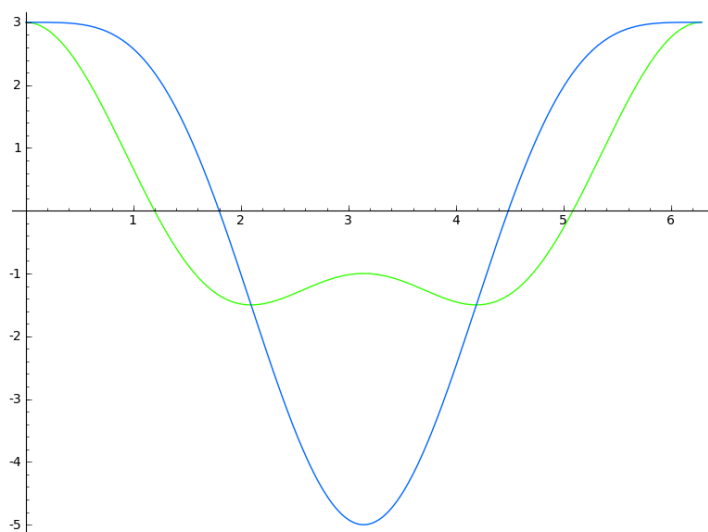
```
plot(sin, (-3,7))
```



Slika 6.17: Sage: Graf sinus funkcije

Tako možemo prikazati i grafove više funkcija u istom koordinatnom sustavu, te možemo prikazati grafičko rješenje našeg sustava s oprugama:

```
t = var('t')
p1 = plot(cos(2*t) + 2*cos(t), (t,0, 2*pi), rgbcolor=hue(0.3))
p2 = plot(4*cos(t) - cos(2*t), (t,0, 2*pi), rgbcolor=hue(0.6))
show(p1 + p2)
```



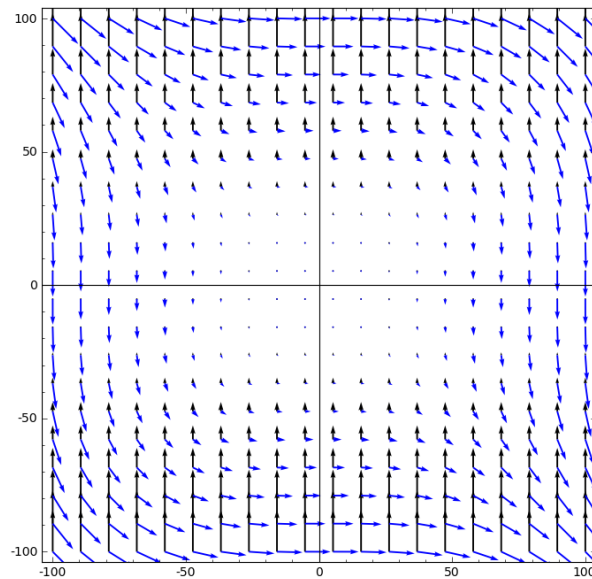
Slika 6.18: Sage: Graf rješenja sustava opruga

Vektorska polja se također mogu jednostavno prikazati

```

var('x, y')
a = plot_vector_field((x**2, y**2), (x, -100, 100), (y, -100, 100), color='black' )
b = plot_vector_field((y**2, -x**2), (x, -100, 100), (y, -100, 100), color='blue')
show(a + b, aspect_ratio=1, figsize=(7, 7))

```



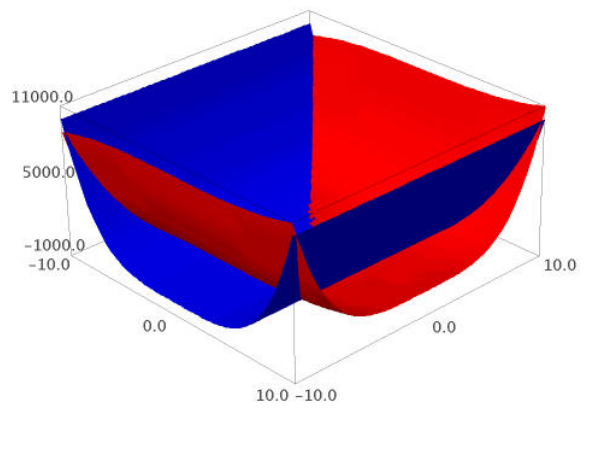
Slika 6.19: Sage: Vektorsko polje

Sage se može koristiti za izradu svakakvih trodimenzionalnih crteža. Standardna je postavka da se za prikaz koristi besplatan paket [Jmol], koji podržava interakciju sa 3D modelima, te ih možemo rotirati, povećavati i smanjivati po volji.

```

x, y, z = var('x,y,z')
p1 = plot3d(x^3 + y^4, (x,-10,10), (y,-10,10), color="red")
p2 = plot3d(x^4 - 2*y, (x,-10,10), (y,-10,10), color="blue")
show(p1 + p2)

```



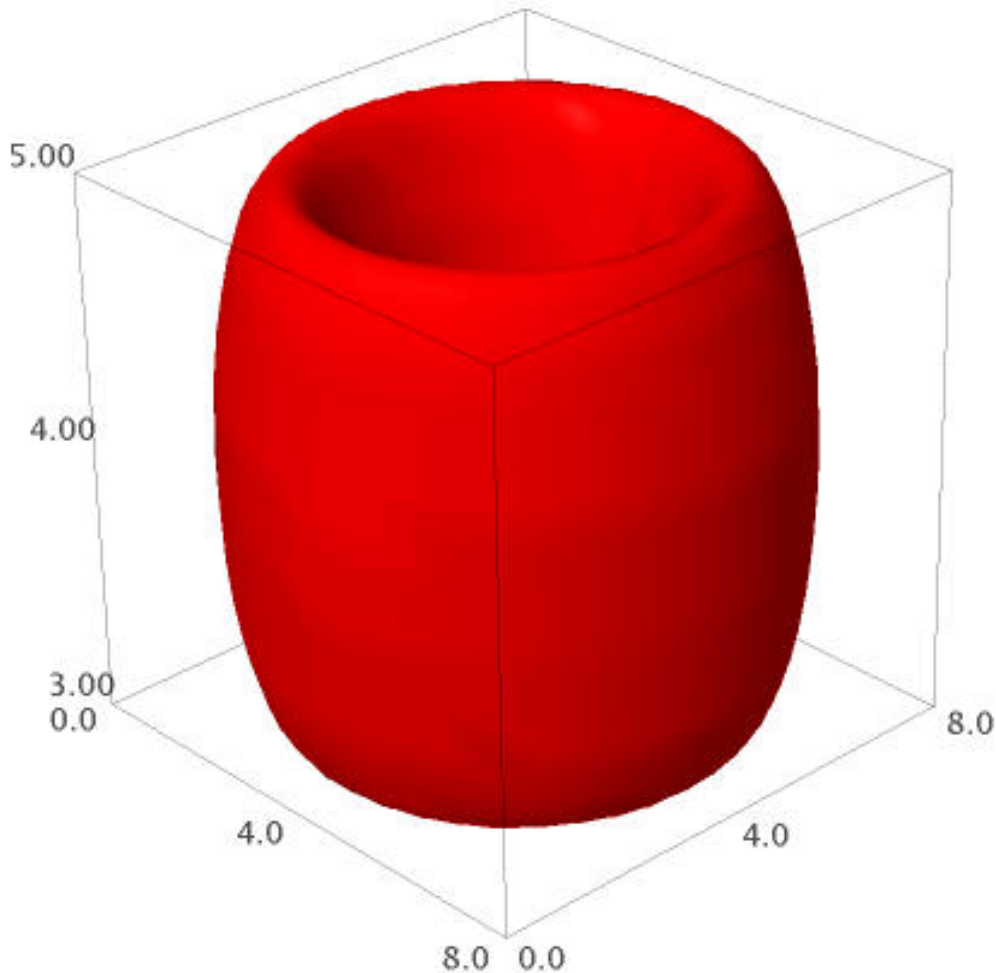
Slika 6.20: Sage: Kombinirani 3D graf

Sage podržava nekoliko načina na koji možemo dobiti neke impresivne oblike. Za spomenuti su naredbe

```

parametric_plot3d, implicit_plot3d isl.
var('u, v')
f1 = (4 + (3 + cos(v)) * sin(u), 4 + (3 + cos(v)) * cos(u), v)
p1 = parametric_plot3d(f1, (u, 0, 2 * pi), (v, 0, 2 * pi), texture="red")
show(p1)

```



Slika 6.21: Sage: Deformirani torus

6.6 Operacije s matricama

Sustavi linearnih jednadžbi se mogu gotovo uvijek zapisati u matričnom obliku. Često se komplicirani sustavi lakše daju riješiti matričnim zapisom, a pojavom računala zapisivati matematičke probleme u matričnom obliku je sve zahvalnije. Numeričke metode u strojarstvu se većim djelom zasnivaju na operacijama sa matricama, planimetrijski sustavi, razna opterećenja, tenzori, svi se opisuju matricama koje mogu sadržavati brojeve, funkcije, matrice ili diskretne diferencijalne jednadžbe. Matričnim računom se mogu riješavati sustavi ili se ovisno o tipu i ponašanju matrica može ustvrditi stanje i stabilnost sustava, ponašanje fluida i slično. Poanta je da se u današnjim tehničkim strukama, matrični račun koristi svakodnevno, a razvoj računala koja danas imaju ogromnu

moć operacija, te matricne račune može izvršavati zadivljujućom brzinom. Sage se kao i svi ostali matematički alati odlično snalazi sa matricama, te u sljedećem primjeru možemo pokazati neke osnovne operacije, koje često koristimo kod linearne algebre:

```
#Choose the size D of the square matrix:
D = 3

example = [[1 if k==j else 0 for k in range(D)] for j in range(D)]
example[0][-1] = 2
example[-1][0] = 3

@interact
def _(M=input_grid(D,D, default = example,
                  label='Matrix to invert', to_value=matrix),
      tt = text_control('Enter the bits of precision used'
                       ' (only if you entered floating point numbers)'),
      precision = slider(5,100,5,20),
      auto_update=False):
    if det(M)==0:
        print 'Failure: Matrix is not invertible'
        return
    if M.base_ring() == RR:
        M = M.apply_map(RealField(precision))
    N=M
    M=M.augment(identity_matrix(D))
    print 'Konstruiramo proširenu matricu'
    show(M)
    for m in range(0,D-1):
        if M[m,m] == 0:
            lista = [(M[j,m],j) for j in range(m,D)]
            maxi, c = max(lista)
            M[c,:],M[m,:]=M[m:],M[c:]
            print 'We permute rows %d and %d'%(m+1,c+1)
            show(M)
        for n in range(m+1,D):
            a=M[m,m]
            if M[n,m]!=0:
                print "Zbrojimo umnožak %s puta %d sa redom %d"%(-M[n,m]/a, m+1, n+1)
                M=M.with_added_multiple_of_row(n,m,-M[n,m]/a)
                show(M)
    for m in range(D-1,-1,-1):
        for n in range(m-1,-1,-1):
            a=M[m,m]
            if M[n,m]!=0:
                print "Zbrojimo %s puta red %d sa redom %d"%(-M[n,m]/a, m+1, n+1)
                M=M.with_added_multiple_of_row(n,m,-M[n,m]/a)
                show(M)
    for m in range(0,D):
        if M[m,m]!=1:
            print 'Podjelimo red %d sa %s'%(m+1,M[m,m])
            M = M.with_row_set_to_multiple_of_row(m,m,1/M[m,m])
            show(M)
    M=M.submatrix(0,D,D)
    print 'Izdvojimo desnu podmatricu, koja je inverzna'
    html ('$M^{-1}=%s$'%latex(M))
    print 'Provjerimo da li je inverzna'
    html ('$M^{-1}*M=%s*s=%s$'%(latex(M), latex(N), latex(M*N)))
```

[sage](#)

Matrix to invert	1	0	2
	0	1	0
	3	0	1

Enter the bits of precision used (only if you entered floating point numbers)
 precision 20

Konstruiramo proširenu matricu

$$\begin{pmatrix} 1 & 0 & 2 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 3 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Zbrojimo umnožak -3 puta 1 sa redom 3

$$\begin{pmatrix} 1 & 0 & 2 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -5 & -3 & 0 & 1 \end{pmatrix}$$

Zbrojimo $2/5$ puta red 3 sa redom 1

$$\begin{pmatrix} 1 & 0 & 0 & -\frac{1}{5} & 0 & \frac{2}{5} \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -5 & -3 & 0 & 1 \end{pmatrix}$$

Podjelimo red 3 sa -5

$$\begin{pmatrix} 1 & 0 & 0 & -\frac{1}{5} & 0 & \frac{2}{5} \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & \frac{3}{5} & 0 & -\frac{1}{5} \end{pmatrix}$$

Izdvojimo desnu podmatricu, koja je inverzna

$$M^{-1} = \begin{pmatrix} -\frac{1}{5} & 0 & \frac{2}{5} \\ 0 & 1 & 0 \\ \frac{3}{5} & 0 & -\frac{1}{5} \end{pmatrix}$$

Provjerimo da li je inverzna

$$M^{-1} \cdot M = \begin{pmatrix} -\frac{1}{5} & 0 & \frac{2}{5} \\ 0 & 1 & 0 \\ \frac{3}{5} & 0 & -\frac{1}{5} \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 0 \\ 3 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Slika 6.22: Sage: operacije s matricama

6.7 Regresija

Često u inženjerskoj praksi postoje neki odnosi između varijabli, te se želi izraziti taj odnos nekim matematičkim konstruktom tako da odredimo jednadžbu koja povezuje te varijable. Obično je prvi korak sakupljanje podataka koji korespondiraju vrijednostima spomenutih varijabli. Za primjer, pretpostavimo da x i y predstavljaju visinu i težinu odraslog čovjeka. Tada uzorak od n pojedinaca sadrži podatke visine x_1, x_2, \dots, x_n i odgovarajuće težine y_1, y_2, \dots, Y_n . Sljedeći je korak unošenje točaka $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ u koordinatni sustav, rezultat je skup rasutih točaka. U takvom je dijagramu često moguće vizualizirati krivulju koja aproksimira tim točkama, te se takva krivulja naziva aproksimacijska krivulja.

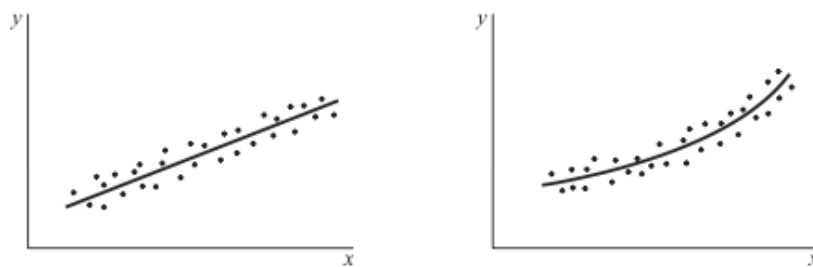
Tako se javljaju slučajevi u kojima je aproksimacijska krivulja pravac, te možemo reći da je odnos među točkama linearan.

$$y = a + bx$$

S druge strane postoje slučajevi gdje ne možemo povući aproksimacijski pravac, te kažemo da je odnos među točkama ne linearan.

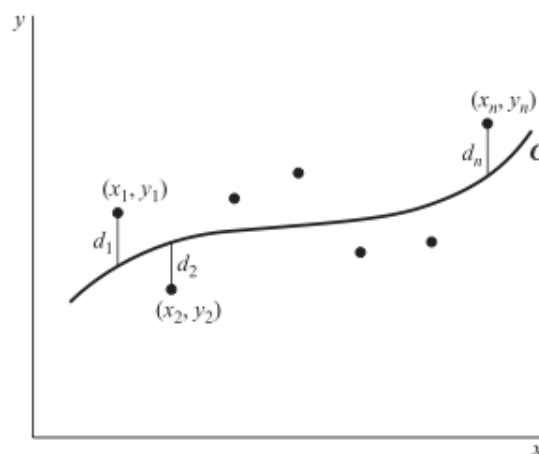
$$y = a + bx + cx^2$$

Uvrštavanjem ovih aproksimacijskih krivulja pokušavamo procijeniti pojedine varijable kao zavisne i nezavisne, te se proces te procijene naziva regresija. Jednadžba koja procjenjuje varijablu x u odnosu na y se naziva regresijska jednadžba a krivulja regresijska krivulja.



Slika 6.23: Aproksimacijske krivulje

Kao rezultat će se često dobiti više od jedne krivulje koja će aproksimirati podatke. Da bi se izbjegla individualna prosidba o kvaliteti aproksimacijske krivulje, potrebno je definirati metodu kojom se ocjenjuje kvaliteta aproksimacije, odnosno krivulje. Na slici je prikazana krivulja koja aproksimira skup točaka $(x_1, y_1), \dots, (x_n, y_n)$.



Slika 6.24: Regresija namanjeg kvadrata

Za zadanu vrijednost x_1 postojati će razlika između vrijednosti y_1 korespondirajuće točke na krivučji C . Tu razliku smo označili sa d_1 , koja se u literaturi naziva devijacija, greška, ili rezidual, te može biti pozitivna ili

nula. Kvaliteta aproksimacijske krivulje se određuje veličinom zbroja $d_1^2 + d_2^2 + \dots + d_n^2$, ako je zbroj malen tafa je aproksimacija (krivulja) kvalitetna, ako je zbroj velik aproksimacija je loša. Možemo izreću definiciju: Od svih krivulja u danoj obitelji krivulja koje aproksimiraju skup od n podataka, krivulja za koju vrijedi,

$$d_1^2 + d_2^2 + \dots + d_n^2 = \min$$

je najbolje odgovarajuća krivulja. Takva krivulja se naziva regresijska krivulja najmenjeg kvadrata ili jednostavnije, krivulja najmanjeg kvadrata. Do sada smo pokazali svestranost sage paketa, te ne iznenađuje činjenica da se i regresija može jednostavno računati u raznim slučajevim mjerenja:

```
def RawGraph(Data, Temp, labels, sig = 4, hueoffset = 0.28):
graph = text("", (0,0))
l = len(Data)

latexlabels = ["$"+labels[0]+"$", "$"+labels[1]+"$"]

maxesx = []
maxesy = []

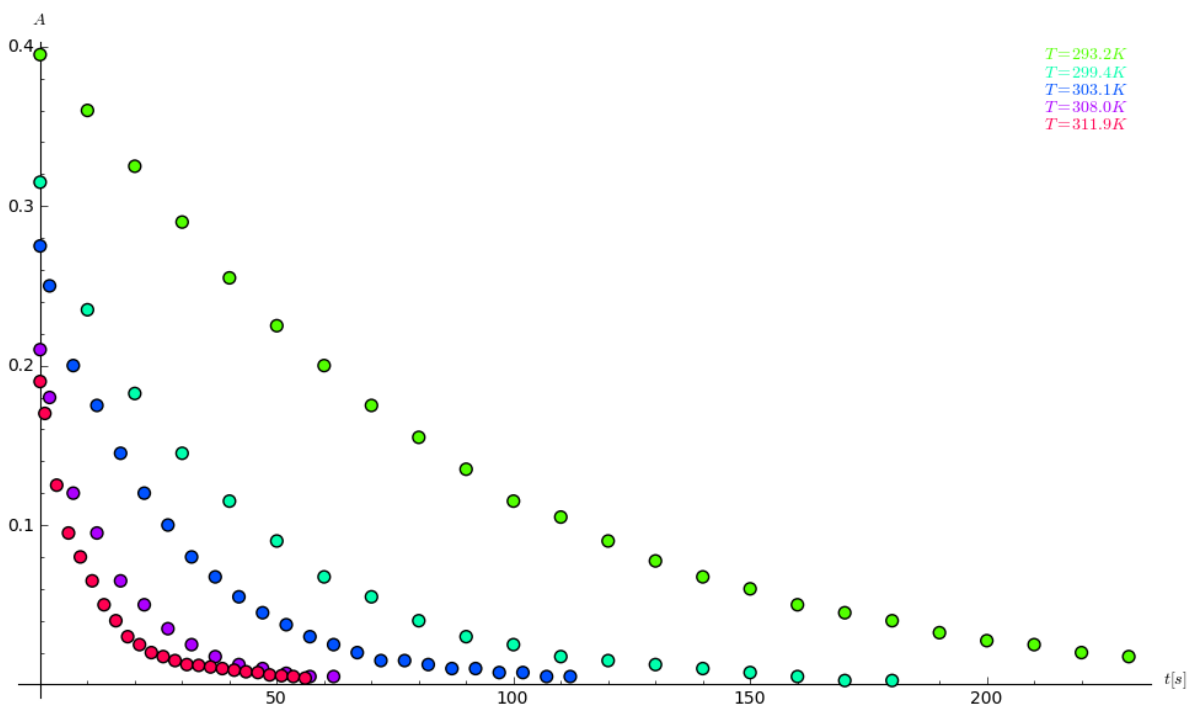
for i in range(0, l):
graph += scatter_plot(Data[i], facecolor=hue(i/(l+1)+hueoffset))
maxesx.append(max([x for x,y in Data[i]]))
maxesy.append(max([y for x,y in Data[i]]))

maxes = (max(maxesx), max(maxesy))

for i in range(0, l):
graph += text("$T = " + str(Temp[i].n(digits=sig)) + " K$", (maxes[0], maxesy[1] - i * maxesy[1]))

graph.axes_labels(latexlabels)
graph.show(figsize=[10, 6])
```

RawGraph(Data, Temperatures, ["t [s]", "A"])



Slika 6.25: Sage: Regresijska krivulja

6.8 Interakcija

U Sageu je moguće integrirati elemente koji omogućuju direktnu komunikaciju sa izvršavanjem kodom. Ova karakteristika počiva na jednom python elementu koji se naziva dekorator. Ukratko, dekoratori se mogu pozvati na svaku funkciju, objekt ili metodu, te su su stanju mjenjati funkcionalnost svih navedenih, bez da se pozivaju podklase ili da se mjenja originalni kod dane funkcije. Obično se poziva se prefixom “@” prije funkcije na koju utječe.:

```
@interact
def _(color = color_selector(default=(1,0,0),
label='Choose a color:', widget='farbtastic', hide_box=False)):
print color
```

U sage postoji dekorator “interact” pomoću kojeg možemo generirati interaktivne elementi u bilježnici našeg programa. Možemo dodavati polja za unos ili razne izbornike, koji mogu interaktivno ili u realnom vremenu mjenjati ispis koda koji smo napisali. Tako možemo na licu mjesta unositi elemente matrice, koja će se onda dalje obrađivati napisanim programom.

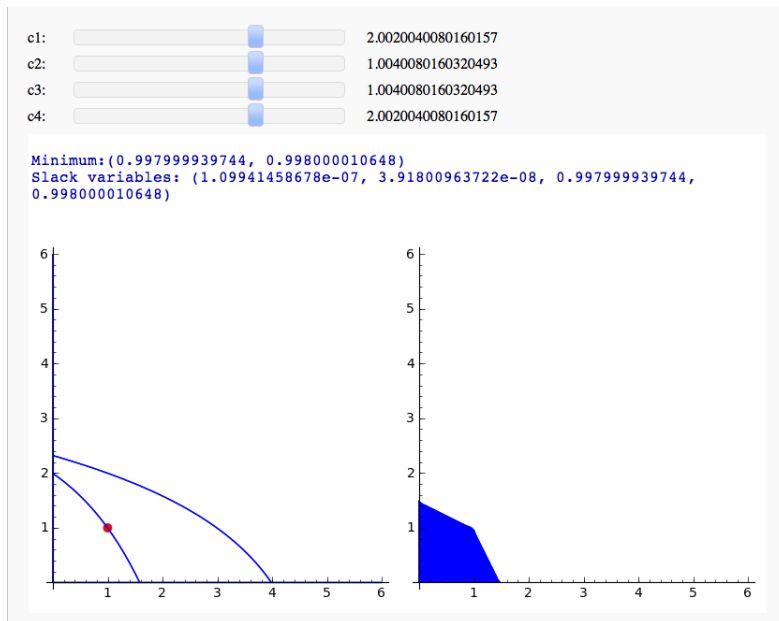
```
default_matrix = Matrix(QQ, 2, 4)
@interact # Shortcut
def _(input_matrix = default_matrix):
print input_matrix
```



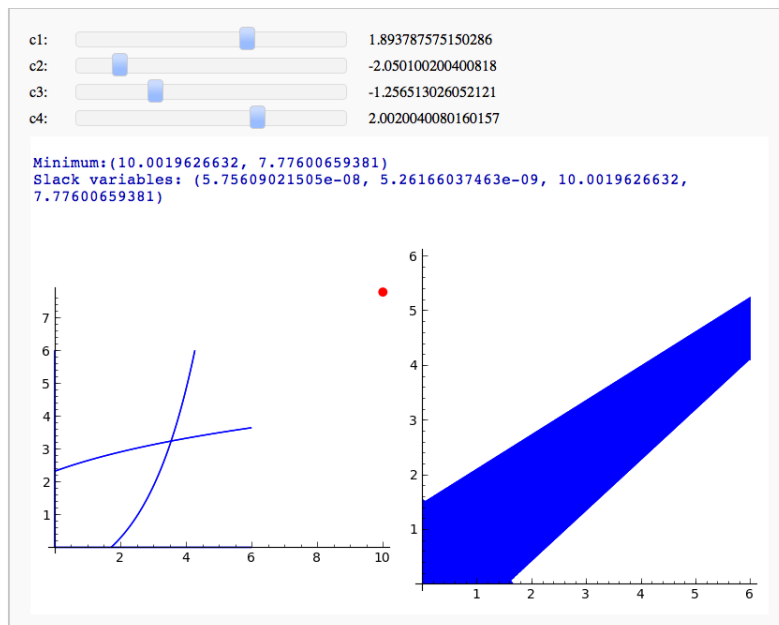
Slika 6.26: Sage: Interaktivna matrica

Također možemo imati program, koji ispisuje funkciju, a kojom možemo manipulirati i odmah vidjeti njen izgled u grafu.

```
var('x,y')
@interact
def _(c1 = slider(vmin=0, vmax=3, default=2, label='c1:'),
c2 = slider(vmin=-3, vmax=3, default=1, label='c2:'),
c3 = slider(vmin=-3, vmax=3, default=1, label='c3:'),
c4 = slider(vmin=0, vmax=3, default=2, label='c4:')):
c=vector(RDF, [-4, -5])
G=matrix(RDF, [[c1, c2], [c3, c4], [-1, 0], [0, -1]])
h=vector(RDF, [3, 3, 0, 0])
sol=linear_program(c, G, h)
print "Minimum:" + str(sol['x'])
print "Slack variables: " + str(sol['s'])
c1_plot = implicit_plot(c1*x + c2*y == 3, (x, 0, 6), (y, 0, 6))
c2_plot = implicit_plot(c3*x + c4*y == 5, (x, 0, 6), (y, 0, 6))
c3_plot = implicit_plot(x == 0, (x, 0, 6), (y, 0, 6))
c4_plot = implicit_plot(y == 0, (x, 0, 6), (y, 0, 6))
min_plot = point(sol['x'], color='red', size=50)
rp = region_plot([c1*x + c2*y <= 3, c3*x + c4*y <= 3, x >= 0, y >= 0], (x, 0, 6), (y, 0, 6))
g = graphics_array([c1_plot+c2_plot+c3_plot+c4_plot+min_plot, rp], 1, 2)
g.show(aspect_ratio=1)
```



Slika 6.27: Sage: Osnovni prikaz funkcije



Slika 6.28: Sage: Interaktivno promjenjena funkcija

DOKUMENTACIJA I KORIŠTENI ALATI

U ovom poglavlju bih se osvrnuo na Sage dokumentaciju kao i na alat kojim je ona napravljena. Velika dokumentacija kao i index svih bitnih pojmova vezanih za Sage sde mogu naći na njihovim službenim internetskim stranicam: <http://www.sagemath.org/doc/>. Radi boljeg snalaženja dokumentacija je podjeljena u nekoliko cjelina. Uz standardnu *Pitanja i odgovori* i *Dio za programere*, postoje još zanimljive cjeline koje su orijentirane na specifičnu tematiku ili korisničku skupinu. Tako imamo vodič za potpune početnike (eng. PREP Tutorials), koji može voditi korisnika, koji ima vrlo malo znanja o računalima općenito, kroz uporaba Sagea, te mu zorno prikazati veći dio Sage funkcionalnosti. Također se kroz vodič početnik susreće sa setom vježbi koje brzo grade osnovno znanje iz programiranja. Druga vrlo zanimljiva cjelina u dokumentaciji, je tematski vodič (eng. Thematic Tutorials). Ovaj je vodič orijentiran na već iskusniju grupu korisnika koji dolaze iz nekog specifičnog područja, te imaju solidno znanje iz programiranja. Tematika varira od funkcionlanog programiranja za matematičare do teorije brojeva i kriptografije. U konačnici Sage ima veliku bazu podataka i primjera koji se mogu naći u dokumentaciji, a uz to postoji i velika mreža korisnika koji na raznim mjestima na internetu objavljuju svoje radove ili otkrića, bilo u znanosti ili u samoj funkcionalnosti Sagea. Sama dokumentacija je generirana automatski alatom koji je korišten pri izradi ovog rada. Riječ je o Sphinx aplikaciji koja služi kao pametni generator dokumentacije i vodiča.

7.1 Sphinx

Sphinx aplikacija napisana u Pythonu, originalno zamišljena kao automatski generator dokumentacije. Automatski je u onom smislu da može izvlačiti djelove koda koji je napisan i prikazati ga u čitljivom obliku, no sam tekst je ipak potrebno napisati ručno te je za tu potrebu razvijena minimalna Sphinx sintaksa. Ubrzo je zbog svoje fleksibilnosti i lakoće korištenja Sphinx postao vrlo popularan te se proširio kao alat za generiranje dokumentacije i nekih drugih jezika, poput Java, C ili čak PHPa. Osnovni format u kojem se piše je RST (reStructuredText) format. Sphinx iz tog formata može generirati PDF putem latexa ili HTML, te zahvaljujući svom mehanizmu generiranja HTMLa ili PDFa koristi vrlo jednostavnu i laganu sintaxu.

7.1.1 Osnovna sintaksa

Mogli bi u podjeltit na dvije cjeline. Jedno je čisti tekst koji želimo prikazati koji se piše direktno bez dodatnog označavanja. Svaki novi red označava novi paragraf i on se mora naglasiti tako da se između dva paragrafa ostavi jedan red prazno. Drugi dio su direktive, koje su u biti riječi sa određenim dodatnim znakovima koje Sphinx tumači na određeni način za danu situaciju. Poziv određene direktive s vrši pisanjem dvije točke `..`, ako želimo recimo učitati neku sliku.

```
.. image:: picture.png
   :scale: 50%
```

Ili ako želimo deklarirati python funkciju koja će biti ljepo prikazana:

```
.. function:: enumerate(sequence[, start=0])
```

Osim direktiva možemo koristiti i znakove za ukrašavanje teksta koji se mogu koristiti u samom tekstu. Tako na primjer ako želim jače otisnuti tekst mora ga staviti između znakova `**`

```
**Bold**
```

Time ćemo dobiti jače otisnut **tekst**. Sve naredbe koje Shpinx ima se u konačnici primjenjuju na jedan od ova dva načina, dakle ili upisivanjem direktiva koje imaju svoje parametere ili ubacivanjem specifičnih znakova ili ključnijih riječi u sam paragraf. Jedna od bitnih stvari koje sam primjetio i koje mi nisu bile jasne u početku, je da python registrira prazno mjesto dobivenom razmakom (space), uvlačenjem (tab) ili prelaskom u novi red (enter, return). Prazna mjesta u tekstu ovdje imaju funkcionalnu težinu, stoga ako želimo staviti neki tekst u prozor za prikazivanje koda.

```
::
    #ovo je primjer koda u tekstu
    v = var('v')
```

Ispravan način za prikazivanje je da iza dvostruke točke mora biti prazan redak i da kod koji želimo prikazati mora biti uvučen, stoga na ovaj način:

```
::
#Neki kod
if x == 1:
    print('x je jedan')
```

Nećemo dobiti pravilni ispis koda u tekstu. U konačnici treba imati na umu da je Sphinx napisan u Pythonu, a sam python je osjetljiv na relativnu poziciju koda u svojoj sintaksi, te se ta karakteristika preslikala i u Sphinx sintaksu. Detaljne liste svih direktiva i oznaka za uređivanje teksta se vrlo lako mogu naći na službenim stranicama Sphinx-a, te ih neću ovdje nabrajati.

7.1.2 Izrada dokumentacija

Za izradu dokumentacije prvo je potrebno instalirati Sphinx na računalo. To se jednostavno može obaviti upisivanjem `easy_install` naredbe i Linux terminal ili Windows konzolu, a za detalje može se referirati na službenu Sphinx stranicu, generalno je instalacija vrlo jednostavna. Kada je Shpinx instaliran, pokretanje prvog projekta se provodi tako da u konzolu/terminal upišemo naredbu

```
$ sphinx-quickstart
```

Pokrenuti će se automatski proces generiranja dokumentacije, te je potrebno postaviti neke osnovne stavke, nije za prvi korak vrlo bitno što se upisuje jer se kasnije sve vrijednosti mogu promijeniti. Po završetku inicijalizacije će Sphinx u direktoriju koju smo odredili stvoriti određenu strukturu podataka. Unutar zadane strukture se nalaze dvije elementarne datoteke: `index.rst` i `conf.py`. U `index.rst` uključujemo sve ostale datoteke i generiramo sadržaj, dok je `conf.py` glavna konfiguracijska datoteka. Uključivati strane datoteke u `index` možemo tako da samo upišemo ime datoteke bez ikakve ekstenzije, te Shpinx automatski zna o čemu je riječ.

```
.. toctree::
   :numbered:

   sadrzaj_1
   sadrzaj_2
   sadrzaj_3
```

Dok će te iste datoteke u direktoriju gdje generiramo dokumentaciju imati ekstenziju `.rst`. U datoteci `conf.py` možemo mijenjati prethodno postavljene postavke ili dodavati nove. Možemo postaviti jezik u kojem će dokumentacija biti generirana, postaviti vremensku zonu i sl. Kada konačno napišemo nekoliko rečenica naše dokumentacije, možemo generirati ispis. Ispis dolazi u dva oblika, HTML ili PDF.

Konfiguracija i oblik ispisa HTML se određuje u datoteci `conf.py`, ona je podjeljena na cjeline vezane za oblik ispisa. Tako imam dio koji sadrži listu parametara čijom modifikacijom utječemo na izgled html stranice naše dokumentacije. Moguće je mijenjati globalni izgled dokumentacije kroz teme:

```
html_theme = "default"
html_theme_options = {
    "rightsidebar": "true",
```

```
"relbarbgcolor": "black"
}
```

Ili po želji možemo mjenjati pojedine elemente, koristeći parametere poput `html_title`, `html_style`, `html_logo`, svi ovi parametri su postavljeni u `conf.py`, te su zakomentirani (ovisno da li su aktivno korišteni ili ne):

```
# Theme options are theme-specific and customize the look and feel of a theme
# further. For a list of options available for each theme, see the
# documentation.
#html_theme_options = {}

# Add any paths that contain custom themes here, relative to this directory.
#html_theme_path = []

# The name for this set of Sphinx documents. If None, it defaults to
# "<project> v<release> documentation".
#html_title = None

# A shorter title for the navigation bar. Default is the same as html_title.
#html_short_title = None

# The name of an image file (relative to this directory) to place at the top
# of the sidebar.
#html_logo = None

# The name of an image file (within the static path) to use as favicon of the
# docs. This file should be a Windows icon file (.ico) being 16x16 or 32x32
# pixels large.
#html_favicon = None
```

Stoga je vrlo jednostavno mjenjati postavke HTML ispisa, pošto su sve opcije postavljene jedino je potrebno odkomentirati i unjesti željeni parametar. Lista parametara je zorno prikazana na stranicama dokumentacije sa željenim efektom. Kako bi se generirao HTML ispis potrebno je u terminal ili konzolu upisati:

```
make html
```

Nakon toga će sphinx izbaciti poruku sa postignutim rezultatom i likacijom gdje se HTML datoteke mogu naći.



Slika 7.1: Ispis HTML dokumentacije

Drugi oblik ispisa je PDF, Sphinx će pokrenuti latex i izgenerirati PDF naredbom:

```
make latexpdf
```

Dodatno uređivanje se vodi također u `conf.py` datoteci. Bilo bi korisno ovdje naglasiti par detalja koji nisu odmah očigledni. Naime, pri generiranju PDFa, možemo koristiti osnovne postavke Sphinx-a, no moguće pomoću naredbe `latex_elements{}` definirati vlastite latex elemente. Možemo definirati dodatne pakete u zaglavlju (preamble), kao i kreirati vlastitu naslovnu stranu, također je moguće modificirati zaglavlje i dno stranice, možemo prikazati primjerom:

```
latex_elements = {
    'releasename' : u'Šimun Strukan',
    'preamble' : r'\setcounter{tocdepth}{10} \pagestyle{empty}',
    'tableofcontents' : r'\tableofcontents \listoffigures \newpage',
    'papersize' : 'a4paper',
    'fncychap' : r'\usepackage[Conny]{fncychap}',
    'maketitle' : ur'\thispagestyle{empty} \begin{center} \LARGE SVEUČILIŠTE U ZAGERBU
        \\ FAKULTET STROJARSTVA I BRODOGRADNJE \end{center}
        \begin{center} \vspace{2in} \LARGE \textbf{ZAVRŠNI RAD}\end{center} \vspace{3in}
        \begin{flushleft} \Large Voditelj rada: \\ dr.sc.
        Mario Essert \end{flushleft} \begin{flushright}
        \Large Šimun Strukan\end{flushright} \vspace{2in} \begin{center} ZAGREB, rujan
        2012 \end{center}' ,
}
```

Na ovaj način smo zaobišli osnovne postavke za generaciju PDF dokumenata, te sam napravio vlastitu naslovnicu i recimo dodao popis slika u svoj sadržaj. Parametri `'releasename'` i `'preamble'` su predefinirani Sphinxom, te se detaljna lista tih parametara, zajedno sa objašnjenjima, može naći u dokumentaciji. Detalj koji bi ovdje napomenuo su varijable `u` i `r`, naime ako želimo u parametar napisati “sirovi” latex kod moramo ispred jednostrukih novodnika pod kojim je kod staviti varijablu `r`, u suprotnome će Sphinx pri gradnji PDFa generirati grešku i neće prepoznati upisano kao latex kod. Druga varijabla pak mogućava upise utf-8 znakova, odnosno koristimo ju ako želimo upisati hrvatske dijakritičke znakove, jer u suprotnome će Shoinx ponovno javiti grešku odnosno da ne može pročitati unešene znakove. Stoga ako želim na naslovnu stranu unjesti dodatne informacije na hrvatskom jeziku morao bih na sljedeći način unjesti kod:

```
latex_elements = {
    'maketitle' : ur'\Large Šimun Strukan',
}
```

Postoje dodatne naredbe analogne onima za HTML ispis, a detaljni opis se nalazi na stranicama dokumentacije.

ZAKLJUČAK

Ovim radom je prikazan Sage matematički alat, te njegove mogućnosti primjene. Po samoj svojoj prirodi ovaj alat je namjenjen korištenju u znanstvenim i tehničkim strukama, stoga se može automatski primjeniti i u strojarskim i sličnim disciplinama. Ideja je bila da se pokaže povezanost između onoga što Sage ima za ponuditi i onoga što je tematski potrebno u nastavi FSB-a. Malo je reći da smo jedva dotakli površinu ovim izlaganjem, jer Sage ima puno više toga za ponuditi. Znanstvena zajednica je dobrim djelom već prihvatila sve mogućnosti koje ima, te je velik broj uglednih fakulteta objavio mnoštvo zanimljivih radova koristeći Sage, koji onda pokazuju njegove mogućnosti u punom svijetlu.

Mislim da su ključni elementi koji ga čine popularnim, činjenica da je alat besplatan, te činjenica da je modularan i kompatibilan sa već postojećim alatima na tržištu. Kako sam ranije objasnio u prirodi otvorenog koda, Sage kontinuirano raste i to ne prema razvojnoj ideji jednog proizvođača, već gotovo organski raste u smjerovima najveće potražnje i kvalitetne primjene. Također treba imati na umu da Sage nije jedini takav alat, no spada definitivno među neke od najboljih, a činjenica da je pisan u pythonu mu omogućava da u svoj sustav integrira sva rješenja koja python ima ponuditi, kojih se u proteklih petnaestak godinu nakupilo podosta. Smatram da bi ovakav alat vrlo poboljšao kvalitetu nastave na FSB-u, ako ništa da se makne naglasak sa algebarskih problema srednje škole, kako bi pristup studenata mogao evolvirati prema jednom više inženjerskom načinu razmišljanja. Na taj način bi se omogućio jedan više inženjerski pristup studiju, dok bi znanje stečeno snalaženjem u python ili Sage kodu, bilo vrlo vrijedno životno znanje svakom budućem inženjeru