[1] Опис лиценци Креативне заједнице доступан је на адреси creativecommons.org.rs/?page_id=74.

UNIVERZITET U NOVOM SADU
PRIRODNO-MATEMATIČKI FAKULTET
DEPARTMAN ZA FIZIKU

Dr Stevan Armaković

PROGRAMIRANJE U FIZICI

NOVI SAD, 2023.

PROGRAMIRANJE U FIZICI

Autor: Dr Stevan Armaković, docent

Prirodno-matematičkog fakulteta u Novom Sadu

Recezenti: Dr Sanja Rapajić, redovni profesor

Prirodno-matematičkog fakulteta u Novom Sadu

Dr Lazar Gavanski, vanredni profesor

Prirodno-matematičkog fakulteta u Novom Sadu

Dr Svetlana Pelemiš, redovni profesor

Tehnološki fakultet Zvornik, Bosna i Hercegovina

Izdavač: Univezitet u Novom Sadu, Prirodno-matematički fakultet, Novi Sad, Trg Dositeja Obradovića 3

Glavni i odgovorni urednik: Prof. Dr Milica Pavkov Hrvojević, dekan

Prirodno-matematičkog fakulteta u Novom Sadu

Upotreba ovog udžbenika (elektronsko izdanje) je odobrena odlukom Nastavno-naučnog veća Prirodno-matematičkog fakulteta u Novom Sadu od 13.07.2023. godine, rešenje br. 0602-07-212/23-9.

CIP - Каталогизација у публикацији
Библиотеке Матице српске, Нови Сад
004.42

ARMAKOVIĆ, Stevan, 1985-
Programiranje u fizici / Stevan Armaković. - Novi Sad : Prirodno-matematički fakultet, Departman za fiziku, 2023. - 156 str.

Način pristupa (URL):
https://www.pmf.uns.ac.rs/studije/epublikacije/fizika/armakovic_programiranje_u_fizici.pdf. - Opis zasnovan na stanju na dan: 15.08.2023. - Bibliografija.

ISBN 978-86-7031-572-3

1. Gl. stv. nasl.
a) Програмирање

COBISS.SR-ID 122544137

U znak sećanja na Zoricu Armaković (rođ. Bugarić), prof. engleskog jezika

(1950 - 2017)

If you can talk with crowds and keep your virtue,

Or walk with Kings—nor lose the common touch,

If neither foes nor loving friends can hurt you,

If all men count with you, but none too much;

If you can fill the unforgiving minute

With sixty seconds' worth of distance run,

Yours is the Earth and everything that's in it,

And—which is more—you'll be a Man, my son!

Odlomak iz pesme „Ako“, Radjard Kiplinga

Contents

Zahvalnica

Najveću zahvalnost dugujem članovima najuže porodice, najpre supruzi Sanji, a zatim sinovima Filipu i Stefanu. Hvala vam na beskrajnoj podršci, inspiraciji i strpljenju.

Zahvaljujem se članovim šire porodice - tetki Gordani Protić i porodici Kler (Dragani, Josipu i Banetu), na stalnoj podršci.

Zahvaljujem se prijateljima na stalnoj podršci: kumu Nebojši Pavlović, kumi Tanji Kovačević, Marii i Nikoli Savanović, Samiri i Vitomiru Radojčić.

Zahvaljujem se recenzentima na izdvojenom vremenu i konstruktivnim komentarima.

Predgovor izdanju

Udžbenik „Programiranje u fizici" je namenjen studentima fizike, pre svega za potrebe savladavanja „programerskog" dela ispita iz predmeta „Numerička analiza i programiranje u fizici". Međutim, udžbenik može biti koristan i studentima ostalih prirodnih ili tehničkih nauka, voljnim da se upoznaju sa osnovama programiranja.

Osnovni cilj ovog udžbenika je da demonstrira kako osnovni principi programiranja mogu da se iskoriste za rešavanje različitih problema u fizici. Primeri su tako formulisani da se mogu lako prilagoditi primeni u različitim oblastima fizike, npr. molekulskoj ili nuklearnoj fizici, ali i drugim naukama, kao što su hemija, biologija, mašinsko učenje, itd.

Učenje programiranja podrazumeva odabir jednog (ili više) programskih jezika čije će se mogućnosti primenjivati za rešavanje određenih problema. Danas je dostupan ogroman broj programskih jezika, a za potrebe pisanja ovog udžbenika izabran je programski jezik python. Tokom godina, python je postao jedan od najpopularnijih programiskih jezika, a prema nekim izvorima i najpopularniji. Pronašao je primenu u praktično svim relevantnim aspektima programiranja i često se koristi u akademskoj i naučnoj zajednici. Njegova primena u industriji je posebno važna, posebno kada se radi o analizi podataka, pa je često izbor čak i u odnosu na komercijalna rešenja. Poznavanje rada u pythonu je neosporno cenjena veština na tržištu rada.

Pored navedenog, za autora ovog udžbenika, četiri su dodatna razloga za odabir pythona. Prvi razlog je izuzetno intuitivna i jednostavna sintaksa ovog programskog jezika, što je od posebnog značaja za početnike u programiranju. Drugi razlog je postojanje ogromnog broja gotovih biblioteka kojima se upotrebljivost pythona nemerljivo povećava. Treći razlog za odabir pythona je njegova dostupnost, jer je python u potpunosti besplatan i otvorenog kôda, baš kao i mnoštvo softverskih okruženja za njegovu upotrebu. Četvrti razlog je činjenica da python pripada grupi najbolje dokumentovanih programskih jezika.

Svi programi i fajlovi sa podacima koji su korišćeni u ovom udžbeniku su dostupni na web stranici https://armakovic.com/textbooks/programiranje-u-fizici-puf/, u skladu sa dostupnim resursima. Na pomenutoj adresi, programi su dostupni u obliku a) .py fajlova, b) Jupyter svezaka i c) interaktivnih programa u okviru online IDE servisa. Zahvaljujući poslednjem, studenti i svi zainteresovani mogu da puštaju i modifikuju sve programe iz ovog udžbenika interaktivno, direktno iz internet pretraživača. Na istom mestu, moguće je vežbati python programiranje u okviru online servisa, bez potrebe da se bilo šta instalira.

Konvencija

Zarad lakšeg snalaženja, u ovom odeljku će biti objašnjeni principi po kojima su označavani različiti elementi izloženog gradiva.

- Python programi su dati u posebnim poljima sa sivom pozadinom koji imaju odgovarajuć, jasan i prepoznatljiv tip formatiranja, sem u jako retkim slučajevima kada su dati kao slike dobijene „print screen“ opcijom, kako bi se predstavio specifičan izlaz koji se dobija neposredno posle programa napisanog u ćeliji *Jupyter Notebook-a*.
- U tekstu van polja sa programima nazivi python funkcija i delovi kôda koji se pominju su pisani `courier_new` fontom, s tim što su python funkcije uvek praćene praznom zagradom, npr. `print()`
- Biblioteke su obeležavane **bold** fontom
- Rezultat programa je dat neposredno nakon programa, i dalje u okviru sa sivom pozadinom, i započinje simbolima >>
- Numeracija primera, slika, tabela i jednačina je vršena u formatu XX.YY gde je XX broj koji označava redni broj poglavlja, a YY redni broj numerisanog objekta u tom poglavlju.
- Kondicionali i petlje su označavani ***italic bold*** fontom
- Tipovi podataka su pisani *italic* fontom.
- U pojedinim delovima teksta neki izrazi koji nemaju adekvatan prevod na srpski jezik su ostavljeni na engleskom jeziku.

1. O programskom jeziku python

Razvoj programskog jezika python je započeo tokom 80-ih godina prošlog veka, pod uticajem programskih jezika ABC i Modula-3. Kôd pythona, u oznaci 0.9.0., je lansiran u februaru 1991. godine. Njegov idejni tvorac i realizator je holandski programer Gvido van Rosum, koji je samostalno razvio i lansirao prve verzije pythona. Iako je naziv ovog jezika asocijacija na životinje iz porodice gmizavaca, kao inspiracija za davanje naziva je poslužila čuvena serija BBC-a „Monty Python's Flying Circus".

Python kôd u oznaci 1.0 je lansiran 1994. godine, dok je verzija 2.0 lansirana 2000. godine. Izdanje 3.0, koje je i dalje aktuelno, je lansirano 2008. godine i donelo je ogromne promene i poboljšanja u odnosu na verziju 2.0. Verzija 2.0 je održavana do 2020. godine, nakon čega je zvanična podrška za ovu verziju prekinuta. U trenutku pisanja ovog udženika, poslednja aktuelna verzija je 3.11.3.

Instaliranje, pokretanje i korišćenje programskog jezika python na ličnom računaru je lako. Dovoljno je da se poseti oficijelni vebsajt organizacije „Python Software Fondation„ koja je zadužena za razvoj pythona, skine datoteka i pokrene instalacija. Nakon instalacije, kôd za python se može pisati u praktično bilo kom editoru teksta, nakon čega se sâm program može pokretati jednostavnom naredbom u komandnoj liniji.

Međutim, postoji i jednostavniji način. Može se instalirati neki od softverskih paketa koji sadrže i programski jezik python i programe za pisanje i izvršavanje samog kôda. Radi se o tzv. integralnim razvojnim okruženjima (IDE)[1], koji imaju čitav niz pogodnosti. U ovim softverskim rešenjima se napisan kôd može direktno izvršavati, što doprinosi mnogo efikasnijem programiranju. Druga pogodnost upotrebe softverskih paketa jeste i ta što programski jezik python u tom slučaju najčešće instaliran zajedno sa najpopularnijim bibliotekama, koje u ogromnoj meri unapređuju funkcionalnost programskog jezika. Trenutno, najpopularniji softverski paketi koji sadrže python su:

- Anaconda (www.anaconda.com)
- PyCharm (https://www.jetbrains.com/pycharm/)
- Visual Studio Code (https://code.visualstudio.com/)

[1] Integrative Devlopment Environments

- Google Colaboratory (https://colab.research.google.com/)
- Thonny (https://thonny.org/)

Svako od navedenih rešenja nudi fantastične mogućnosti i skoro sva su dostupna na svim relevantnim operativnim sistemima (Linux, MacOS, Windows). Anaconda IDE sadrži više alata za programiranje u pythonu, između ostalog Jupyter Notebooks, Jupyter Lab i Spyder. Za potrebe obrazovanja, i generalno u akademskim krugovima, često se koristi Jupyter Notebooks. PyCharm i Visual Studio Code se više koristi u industriji. Google Collaboratory je onlajn interaktivno okruženje za programiranje u pythonu, sa mogućnošću integracije sa Google servisima, kao i sa podrškom za mašinsko učenje. Thonny je fantastično IDE rešenje namenjeno početnicima, ne samo u pythonu, nego generalno u programiranju.

Odabir okruženja najviše zavisi od oblasti kojom se korisnik bavi. Međutim, ono što je zajedničko za navedena rešenja, jeste da su sva besplatna. Iako neka od pomentuih rešenja imaju i svoje komercijalne verzije, besplatne verzije su i više nego dovoljne ne samo za početnike, nego i za aktivno bavljenje u različitim programerskim aktivnostima.

Jedan od razloga za popularnost pythona jeste mogućnost da se radi sa različitim tipovima podataka. U kontekstu tipova podataka, python se ističe činjenicom da je to dinamički jezik, jer se tipovi podataka automatski određuju na osnovu vrednosti koje se dodeljuju objektima. Neki od tipova podataka sa kojima python može da radi su sledeći (uključujući njihove oznake u pythonu):

- celobrojni brojevi (*int*),
- decimalni brojevi (*float*).
- tekstualni podaci (*string*),
- liste (*list*),
- nizovi (*array*),
- torke (*tuple*),
- rečnici (*dict*),
- logički (*bolean*).

Nadalje će se kao oznake za navedeve tipove podataka koristiti njihove oznake iz zagrada. Kako se bude izlagalo gradivo, biće izloženo više informacija o nekim od pomenutih tipova podataka.

2. Prvi koraci u pythonu

Jedna od izuzetnih povoljnosti programskog jezika python jeste činjenica da dolazi sa velikim brojem ugrađenih funkcija. Konkretno, samom instalacijom pythona dostupno je preko 90 funkcija. Sa druge strane, na vrlo jednostavan način moguće je definisati proizvoljnu funkciju i na taj način olakšati rešavanje nekog problema. U ovom delu će biti predstavljeno kako se mogu koristiti neke od osnovnih ugrađenih funkcija pythona, kao i kako se mogu definisati i pozivati funkcije definisane od strane korisnika.

2.1. Funkcija `print()`

Jedna od osnovnih i najčešće korišćenih funkcija, u programiranju generalno, jeste funkcija `print()`. Ona se koristi za zadavanje instrukcije programskom jeziku da ispiše[2] („štampa“ na ekranu) tekst ili vrednost neke promenljive.

U zagradi se definiše sadržaj koji će se štampati, tj. u zagradu se smešta argument funkcije. Na primer, komanda `print("Zdravo Svete")` će štampati liniju u kojoj će pisati Zdravo Svete. U ovom slučaju, jasno je da je argument funkcije `print("Zdravo Svete")` zapravo *string* tip podataka. Međutim, argument funkcije `print()` može biti i neka promenljiva ili objekat koji su prethodno definisani, i koji mogu da budu praktično bilo koji tip podataka. U slučaju kada argument funkcije `print()` nije *string*, nego neki drugi tip podatka, ova funkcija će prosto štampati vrednosti tok objekta. U primeru koji sledi je dat program u kojem se demonstrira nekoliko načina primene funkcije `print()`.

Primer 2.1. Primena funkcije `print()`

```
1.  a = 6
2.  b = "Ovo je jedan obican string"
3.  c = [1,2,3]
4.
5.  print(" Zdravo Svete ")
6.  print(a) # ovo je komentar, posle tarabe, python ne vidi nista
7.  print(b)
8.  print(c)
9.
>> Zdravo Svete
>> 6
>> Ovo je jedan obican string
>> [1,2,3]
```

[2] U daljem tekstu će se koristiti izraz štampanje

U prve tri linije su definisane vrednosti objekata, dok su u linijama 5-8 tražena štampanja vrednosti pomenutih objekata. Tip podatka u liniji 1 je ceo broj (*int*), u liniji 2 tekst (*string*), dok je u liniji 3 tip podatka lista. Takođe, važno je naglasiti i da posle simbola „#“ (tarabe), python ništa ne čita, što je korisno za komentarisanje kôda.

2.2. Funkcija `type()`

Jedna od često korišćenih funkcija jeste i funkcija `type()`. Kada se u argument ove funkcije stavi neka prethodno definisana promenljiva ili objekat, dobiće se informacija o tome koji tip podatka sadrži promenljiva. Napomenuto je da python može da radi sa velikim brojem tipova podataka. Kada se piše kôd, ili analizira nečiji kôd, često može doći do zabune koji tip podataka sadrži neka promenljiva i tada je primena ove funkcije posebno važna. U primeru koji sledi je demonstrirana upotreba ove funkcije.

Primer 2.2. Primena funkcije `type()`

```
1.  a = 6
2.  b = 6.0
3.  c = "Ovo je jedan obican string"
4.  d = [1 ,2 ,3]
5.
6.  print(type (a))
7.  print(type (b))
8.  print(type (c))
9.  print(type (d))
>> <class 'int'>
>> <class 'float'>
>> <class 'str'>
>> <class 'list'>
```

Kao izlaz programa, štampaju su informacije o tome kojoj klasi pripadaju objekti definisani u linijama od 1 do 4.

2.3. Funkcije za promenu tipa podatka

U prethodnom poglavlju su pomenuti neki od tipova podataka sa kojima se može raditi u pythonu. Sa druge strane, jednom definisan tip podatka može biti prebačen u drugi tip podatka. Npr. broj 7, koji je *int* tipa, može biti prebačen u 7.0, što je *float* tip podatka. Isti broj može biti prebačen i u *string* tip podatka. Promena tipa podataka se često naziva „kastovanje”, od engleske reči „cast”, koja znači ukalupiti ili pečatirati. Za prebacivanje u određeni tip podataka, postoje odgovarajuće funkcije, koje su date u tabeli 2.1.

Tabela 2.1. Komande za prebacivanje tipova podataka

Funkcija	Prebacuje u	Funkcija	Prebacuje u
`str()`	tekst	`dict()`	rečnik
`int()`	ceo broj	`list()`	lista
`float()`	decimalni broj	`bool()`	logički

U sledećem primeru je demonstrirano nekoliko slučajeva promene tipa podataka.

Primer 2.3. Promena tipa podataka

```
1.  a = 6
2.
3.  print("Promenljiva a ima vrednost :", a)
4.  print("Tip promenljive a je: ",type (a))
5.
6.  b = float(a)
7.  print("Promenljiva b ima vrednost :", b)
8.  print("Tip promenljive b je: ",type (b))
9.
10. c = str(b)
11.
12. print("Promenljiva c ima vrednost :", c)
13. print("Tip promenljive c je: ",type (c))
>> Promenljiva a ima vrednost: 6
>> Tip promenljive a je: <class 'int'>
>> Promenljiva b ima vrednost: 6.0
>> Tip promenljive b je: <class 'float'>
>> Promenljiva a ima vrednost: 6
>> Promenljiva c ima vrednost: 6.0
>> Tip promenljive c je: <class 'str'>
```

U primeru 2.3 u prvoj liniji je definisana promenljiva `a` kojoj je dodeljena vrednost 6, pri čemu python odmah prepoznaje da se radi o *int* tipu podatka. U trećoj liniji je traženo da se odštampa vrednost promenljive `a`, a u četvrtoj liniji je traženo da se odštampa i informacija o tome o kom tipu podataka se radi kod promenljive `a`. U šestoj liniji se definiše promenljiva `b`, čija će vrednost iznositi isto koliko i vrednost promenljive `a`, sa tom razlikom što se sada od pythona traži da promenljiva `b` sadrži *float* tip podatka. U 10. liniji je definisana promenljiva `c`, tako da vrednost bude `b`, ali konvertovana u *str* tip podatka.

2.4. Definisanje novih funkcija

Sâm programski jezik python dolazi sa preko 90 ugrađenih funkcija. Te ugrađene funkcije su tek delić neophodnih alata za pisanje programa koji rešavaju neke konkretne zadatke. Međutim, u pythonu je izuzetno lako definisati neku novu funkciju i kasnije u programu je

pozvati sa proizvoljnim argumentom. Definisanje novih funkcija se vrši uz pomoć funkcije `def()`, čiji opšti oblik sledeći:

```
def ime_funkcije(promenljiva1,promenljiva2,...):
    međuizrazi
    konstante
    return konačan izraz
```

Iz predstavljenog opšteg oblika vidi se da se funkcija `def()` u principu sastoji iz tri dela. U prvom delu kôda se poziva funkcija `def()`, zatim se daje naziv novoj funkciji, a na kraju se u zagradi stavljaju promenljive koje figurišu u novodefinisanoj funkciji, praćeno dvotačkom. Drugi deo funkcije `def()` sadrži dodatne instrukcije kako bi se definisala funkcija, npr. ako je izraz složen ovaj deo može da sadrži međuizraze, ili konstante. Drugi deo često nije neophodan, ako je novodefinisana funkcija jednostavna, a sa druge strane može da sadrži i više linija kôda. Treći deo, `return`, sadrži izraz kojim se konstruiše izlaz funkcije. Novodefinisana funkcija se kasnije u kôdu lako može pozvati, upisivanjem njenog imena i stavljanjem u zagradu odgovarajućih argumenata. Definisanje nove funkcije biće demonstrirano na primerima koji slede.

Primer 2.4. Definisanje nove funkcije

```
1.  def sabiranje(a,b):
2.      return a + b
3.
4.  rezultat = sabiranje(2,3)
5.  print("rezultat je:", rezultat)

>> rezultat je 5
```

U programu primera 2.4. definisana je nova funkcija koja će sabirati dva broja. Radi se o jednostavnoj funkciji, tako da u drugom delu funkcije `def()` nije bilo potrebe za dodatnim instrukcijama. U prvoj liniji kôda definisan je naziv nove funkcije, a u zagradu su stavljene dve promenljive, pošto će funkcija sabirati dva broja. U `return` sekciji instruisano je da izlaz funkcije bude zbir promenljivih `a` i `b`. U četvrtoj liniji definiše se nova promenljiva, a njena vrednost se dobija pozivanjem funkcije `sabiranje`, dok se u zagradi upisuju brojevi 2 i 3. U poslednjoj liniji se traži da se ispiše vrednost promenljive `rezultat`.

U prethodnom primeru je pokazano kako se definiše jednostavna funkcija. Međutim, često je neophodno definisati složenije funkcije, što je demonstrirano sledećim primerom.

Primer 2.5. Definisane složene funkcije

```
1.  def funkcija(a,b,c):
2.      a = 15
3.      b = 75 * a
4.      c = 3 * (a + 3*b)
5.      return a + b + c
6.
7.  rezultat = funkcija(2,3,4)
8.  print("rezultat je:",rezultat)

>> rezultat je 11310
```

U poslednjem primeru predstavljeno je definisanje nove funkcije uz dodatne instrukcije (linije koda 2, 3 i 4). U liniji 5 se traži da izlaz nove funkcije bude zbir parametara `a`, `b` i `c`, dok su vrednosti tih paramatera regulisani dodatnim instrukcijama u linijama 2, 3 i 4. U liniji 7 se poziva nova funkcija i kao argument se stavljaju vrednosti parametara `a`, `b` i `c`.

3. Biblioteke u python-u

U programiranju se pod izrazom biblioteka podrazumeva kôd u kome su definisane različite funkcije koje inače nisu dostupne u nekom programskom jeziku. Sa tehničkog aspekta, biblioteke su ništa drugo do fajlovi koji sadrže funkcije koje korisnici ne moraju iznova da definišu, nego mogu da iskoriste iz tih biblioteka/fajlova.

Postoji mnogo funkcija koje su korisne u programiranju, a nisu deo programskih jezika. Vremenom su korisnici prepoznali značaj tih funkcija, pa su te funkcije učinili dostupnim drugim korisnicima u obliku biblioteka. To u praksi znači da ako korisnik želi da napiše složen program, verovatno ne mora da definiše sve funkcije koje su mu neophodne za realizaciju zadatka, nego može da ih pozove iz određene biblioteke i tako sebi skrati vreme neophodno za pisanje programa.

Za biblioteke u programiranju se može reći da su gotove. Karakteristika da su „gotove“ znači da krajnji korisnik ne mora sa njima ništa drugo da radi, sem da ih instalira i/ili učita u svoj program i pozove željenu funkciju. Zato se relativno često sreće i izraz „gotove biblioteke“ umesto „biblioteke“. U daljem tekstu koristiti će se izraz samo „biblioteka“.

U opštem slučaju bibloteke se mogu podeliti na ugrađene i eksterne. Ugrađene biblioteke su dostavljene u okviru instalacije programskog jezika, ali nisu automatski dostupne, nego se

moraju učitati na početku programa. Sa druge strane, eksterne biblioteke je neophodno prvo instalirati, a zatim i učitati na početku programa.

Svakako jedna od najbitnijih karakteristika pythona jeste postojanje ogromnog broja gotovih biblioteka. Svakim danom se pojavljuje ogroman broju novih biblioteka sa novim funkcijama koje su korisne za pisanje različitih programa. Dodatna pogodnost u vezi pythona jeste što se biblioteke za python instaliraju i učitavaju veoma lako, što dodatno utiče na popularnost ovog programskog jezika.

3.1. Instalacija biblioteka

Instalacija python biblioteka je laka i može se sprovesti na više načina. Najlakši način instalacije biblioteke jeste upotrebom nekog od sistema za upravljanje bibliotekama. Jedan od najpoznatijih takvih sistema jeste **package instaler for python**, poznatiji po svom akronimu **pip**. Program **pip** se putem interneta povezuje na repozitorij dostupnih javnih biblioteka i zatim automatski instalira biblioteku na korisnikov računar. Program **pip** može da se povezuje na više različitih izvora, a jedan od najpoznatijih jeste **python package index** (PyPI). U tom repozitoriju, koji sadrži milione biblioteka, svaka od biblioteka je poznata pod svojim jedinstvenim imenom. Ukoliko korisnik želi da instalira neku od biblioteka iz pomenutog javnog repozitorijuma biblioteka, dovoljno je da otkuca sledeću komandu u komandnom terminalu

```
pip install naziv_biblioteke
```

Za brisanje biblioteke preko pip programa, koristi se sledeća komanda

```
pip uninstall naziv_biblioteke
```

Ukoliko korisnik želi da iskoristi program **pip** da instalira neku od biblioteka koja nije dostupna u javnom repozitorijumu biblioteka, nego je dostupna u nekom drugom repozitorijumu, koristi se sledeća komanda

```
pip install -i https://adresa_do/paketa <naziv_biblioteke>
```

Željena biblioteka ne mora uopšte da bude dostupna na nekom od repozitorijuma. Može da bude dostupna u okviru samog računarskog sistema korisnika (npr. preuzeta sa neke adrese). I tada se može iskoristiti program **pip** da se instalira biblioteka, i to na sledeći način

```
pip install -i /putanja/do/fajla <naziv_biblioteke>
```

PyPI je dostupan na adresi https://pypi.org/, gde korisnici mogu da vrše pretraživanje i informišu se o milionima python biblioteka. Mnoge biblioteke predstavljene na ovom servisu sadrže osnovne informacije o nameni, upotrebi, kao i adrese zvaničnih vebsajtova projekata iz kojih su proizašle.

Kao što je naglašeno ranije, python se može koristiti u okvirima različitih razvojnih okruženja, a Anaconda je jedno od najpoznatijih takvih rešenja. Slično programu **pip**, postoji i program otvorenog kôda sa istom namenom. Radi se o programu **conda**, koji je prvobitno bio deo Anaconda paketa razvijen od strane istoimene kompanije. Upotreba programa **conda** je praktično ista, i zasniva se na sledećoj opštoj komandi

```
conda install [naziv_biblioteke]
```

Naravno, ukoliko se python koristi u okvirima Anaconda distribucije, preporučeno je da se koristi **conda** menadžer. Vredi napomenuti i da Anaconda Navigator, grafički korisnički interfejs Anaconda paketa, ima i alat za organizaciju biblioteka, pa se instalacija ili deinstalacija pomenutih može činiti i na taj način. Thonny IDE okruženje takođe ima svoj alat kojim se biblioteke jednostavno instaliraju ili uklanjaju. Bez obzira na odabrani način instalacije biblioteka, preporučuje se da se za svaku biblioteku proveri da li postoji uputstvo na internetu, konkretno na zvaničnim stranicama projekata koji razvijaju biblioteke.

U vezi sa komandama za instaliranje biblioteka preko **pip** i **conda** programa, važno je naglasiti da te komande ponekad mogu sadržavati neke dodatne parametre. Međutim, do tih informacija se dolazi brzo i lako prostom pretragom na internetu.

Komande za instalaciju svih eksternih biblioteka korišćenih u python programima ovog udžbenika, upotrebom i **pip** i **conda** programa, su date u **Dodatku 1**. Takođe su navedene i internet adrese zvaničnih projekata sa kojih su upotrebljene biblioteke potekle.

3.2. Učitavanje biblioteka

Da bi se mogle iskoristiti mogućnosti biblioteke, neophodno je da se ista učita na samom početku programa. Korisno je napomenuti da biblioteke, posebno one složenije, mogu biti organizovane u module. Moduli su skupovi sličnih funkcija, klasa, promenljivih i slično, smeštenih u jedan fajl. Prednost upotrebe pythona se ogleda u činjenici da se biblioteka može učitati u celosti ili se može učitati jedan ili više modula iste, ili se čak može učitati samo jedna funkcija iz cele biblioteke.

Postavlja se pitanje zašto bi se učitavao samo deo biblioteke, a ne cela. Odgovor na to pitanje je optimizacija kôda. U slučaju velikih biblioteka, proces njihovog celokupnog učitavanja u program može trajati izvesno vreme. Sa druge strane, program može značajno brže raditi ukoliko se učita samo neophodan deo biblioteke.

Učitavanje biblioteka se vrši prvenstveno komandom **import**. Nekoliko je načina kako se može učitati neka bibliteka i oni imaju sledeću opštu formu:

```
1. import ime_biblioteke
2. from ime_biblioteke import *
3. from ime_biblioteke import deo_biblioteke
4. import ime_biblioteke as skraćeni_naziv
```

U sledećih nekoliko primera biće demonstrirano učitavanje biblioteke. Konkretno, biće demonistrirano kako se biblioteka **numpy** može učitati na nekoliko načina. Značaj biblioteke **numpy** će biti istaknut u nekom od sledećih poglavlja.

Primer 3.1. Učitavanje cele biblioteke

```
1.  import numpy
2.
3.  a = numpy.cos(30)
4.  print(a)

>> 0.15425144988758405
```

U poslednjem primeru pokazano je kako se učitava celokupna **numpy** biblioteka. Ova biblioteka je jedna od najpoznatijih python biblioteka koja sadrži ogroman broj matematičkih funkcija. Između ostalih, sadrži i trigonometrijske funkcije. Python sam po sebi ne može da se bavi naprednim matematičkim funkcijama bez učitavanja odgovarajućih biblioteka. U poslednjem primeru, neka od funkcija učitane biblioteke se poziva tako što se otkuca prvo naziv biblioteke, nakon čega sledi tačka, nakon čega sledi naziv funkcije. Valja naglasiti i da se funkcija neke biblioteke, koja se poziva nakon tačke, u programerskoj terminologiji često naziva „metod“. Drugim rečima, oznaka `numpy.cos()` se relativno često čita kao „kosinus metod biblioteke numpy“.

Ukoliko bi tražili da se vrednost kosinusa ozračuna na sledeći način

```
a = cos(30)
```

python bi izbacio grešku, jer ne bi znao gde da pronađe funkciju `cos`.

Primer 3.2. Drugi način učitavanja cele cele biblioteke

```
1.  from numpy import *
2.
3.  a = cos(30)
4.  print(a)

>> 0.15425144988758405
```

U ovom primeru se vidi kako se biblioteka može učitati u celosti, ali tako da pri pozivanju neke od funkcija iz te biblioteke nije potrebno da se piše i naziv biblioteke. Drugim rečima, za razliku od prethodnog primera, nije potrebno pisati `numpy.cos()` nego je dovoljno napisati samo `cos()`. Učitavanje dela biblioteke je demonstrirano u sledećem primeru.

Primer 3.3. Učitavanje dela biblioteke

```
1.  from numpy import cos
2.
3.  a = cos(30)
4.  print(a)

>> 0.15425144988758405
```

Primećuje se da ni u ovom slučaju ne treba pisati ime biblioteke pre naziva funkcije, tj. dovoljno je napisati samo `cos(30)`. Pošto je učitana samo funkcija `cos()`, python npr. ne bi znao da izračuna izraz `sin(30)`.

Primer 3.4. Učitavanje biblioteke i pozivanje skraćenim nazivom

```
1.  import numpy as np
2.
3.  a = np.cos(30)
4.  print(a)

>> 0.15425144988758405
```

U primeru 3.4, u prvoj liniji kôda se dala instrukcija da se učita biblioteka **numpy**, ali da se dalje u kôdu funkcije ove pozivaju skraćenicom **np**. Ovo je najčešći način učitavanja biblioteka, jer omogućava lakše pozivanje funkcija (metoda) neke biblioteke, a u isto vreme ne dolazi do „zbunjivanja“ python-a. Naime, do zbunjivanja pythona može doći ako npr. dve biblioteke imaju iste funkcije. Ukoliko se učitaju cele, kao što je predstavljeno u primeru 3.2, pozivanjem određene funkcije koja se isto zove u obe biblioteke, python ne bi znao koju funkciju da iskoristi. Treba napomenuti i da su se u programerskoj zajednici ustalili određeni nazivi kao skraćenice za biblioteke. Tako se **numpy** često obeležava sa **np**, **matplotlib.pyplot** sa **plt**, **pandas** sa **pd**, itd. Nadalje, u tekstu će se koristiti skraćeni oblici naziva biblioteka koji

se najčešće koriste u programerskoj zajednici. Imajući u vidu da je programski jezik python izuzetno dobro dokumentovan, čitaocima će tako biti lakše da na internetu pretražuju primere primene tih biblioteka.

Važno je napomenuti i da neke biblioteke sadrže ogroman broj funkcija. Primer je upravo biblioteka **numpy** koja sadrži stotine funkcija. Spisak svih funkcija dostupnih u okviru jedne biblioteke može se dobiti upotrebom ugrađene funkcije `dir()`, koja će kao izlaz dati listu svih funkcija (Primer 3.5.).

Primer 3.5. Upotreba komande `dir()`

```
1.  import numpy
2.
3.  lista = dir(numpy)
4.
5.  print("Ukupan broj funkcija bibliteke numpy iznosi: ", len(lista))

>> Ukupan broj funkcija bibliteke numpy iznosi:  608
```

U poslednjem primeru je u prvoj liniji najpre učitana biblioteka **numpy**, a zatim je u liniji broj 3 definisana lista čiji će elementi biti nazivi funkcija dostupnih u okvirima biblioteke numpy. U liniji 5 je pored funkcije `print()` iskorišćena i funkcija `len()` koja prebrojava koliko ima elemenata u listi. Vidi se da numpy biblioteka sadrži čak 608 funkcija.

Već je pomenuto da biblioteke mogu biti organizovane po modulima. Prethodni primer je demonstrirao ogroman broj funkcija biblioteke **numpy**. Kako bi se omogućila optimalna upotreba resursa, stotine funkcija ove biblioteke su organizovane po odgovarajućim modulima, što omogućava učitavanje samo željenih modula. Na primer, sledeća linija kôda:

```
from numpy.random import random
```

bi učitala **random** modul biblioteke **numpy** koji sadrži funkcije za generisanje slučajnih brojeva, a jedna od tih funkcija je istoimena funkcija `random()`.

3.3. Ugrađene biblioteke

Python poseduje značajan broj ugrađenih biblioteka. U opštem slučaju ugrađene biblioteke pythona se mogu podeliti u više grupa, a u ovom udžbeniku biće navedene sledeće grupe ugrađenih biblioteka za:

- numeričke i matematičke funkcije (**numbers**, **math**, **cmath**, **random**, **statistics**)
- rad sa tipovima podataka (**datetime**, **zoneinfo**, **calendar**,...)

- procesuiranje teksta (**string**, **re**, **readline**,...)
- rad sa sistemskim parametrima (**sys**, **sysconfig**,...)
- otklanjanje grešaka i profilisanje (**bdb**, **pdb**, **timeit**, **trace**, **tracemalloc**,...)
- grafički interfejs (**tkinter**, **tkinter.font**, **tkinter.messagebox**,...)
- rad sa fajlovima i direktorijumima (**pathlib**, **os.path**, **fileinput**, **tempfile**, ...)

U ovom udžbeniku, u poglavljima koja slede, biće demonstrirana uptreba odabranih funkcija iz nekih pomenutih biblioteka.

3.4. Eksterne biblioteke

Broj eksternih biblioteka je ogroman, što čini python jednim od najupotrebljivijih programskih jezika. Na ovom mestu, biće navedene sledeće eksterne bibliteke za:

- numeričke i matematičke funkcije (**numpy**, **scipy**, ...)
- vizualizaciju (**matplotlib**, **seaborn**, **plotly**, ...)
- analizu i manipulaciju podacima (**pandas**)
- Mašinsko učenje/Veštačku inteligenciju (**scikit-learn**, **tensorflow**, **keras**, **pytorch**, ...)

U ovom udžbeniku, biće demonstrirana upotreba odabranih funkcija iz biblioteka **numpy**, **scipy**, **matplotlib** i **pandas**. Za početak, zbog važnosti za pisanje programa u nastavku, biće demonstrirana upotreba nekoliko izuzetno korisnih funkcija biblioteke **numpy**.

3.4.1. Elementi biblioteke numpy

Biblioteka **numpy** je jedna od najpoznatijih python biblioteka. Sadrži preko 600 funkcija koje olakšavaju rad sa različitim numeričkim i matematičkim elementima. Naziv biblioteke je nastao kombinacijom reči „numerical“ i „python“. Ovde će za početak biti demonstrirana upotreba nekoliko funkcija koje su od izuzetnog značaja za pisanje programa, a to su funkcije za:

- definisanje niza sa kontrolom koraka – **range()**
- definisanje niza sa kontrolom krajnje vrednosti – **linspace()**
- definisanje niza kod kojeg svi elementi imaju vrednost 0 – **zeros()**
- definisanje niza kod kojeg svi elementi imaju vrednost 1 – **ones()**

Funkcije `arange()` i `linspace()` služe sa definisanje niza brojeva između neke definisane minimalne i maksimalne vrednosti. Opšti oblik argumenata ovih funkcija je praktično isti – `(min,max,n)`. Međutim, kod funkcije `arange()`, *n* označava vrednost koraka za koju se uvećava minimalna vrednost pa zatim ide sve do maksimalne vrednosti, sa napomenom da se krajnja vrednost ne nalazi u rezultujućem nizu. Ukoliko se kod funkcije `arange()` ne unese vrednost koraka, onda se uzima podrazumevana vrednost koraka koja iznosi 1.

Sa druge strane, kod funkcije `linspace()`, *n* označava broj tačaka na koliko se deli interval od minimalne do maksimalne vrednosti (uključujući i minimalnu i maksimalnu vrednost). Za razliku od slučaja sa funkcijom `arange()`, kod funkcije `linspace()` krajnja vrednost se obavezno nalazi u rezultujućem nizu. Upotreba navedenih funkcija je demonstrirana u primeru koji sledi, a dodatna objašnjenja su u nastavku.

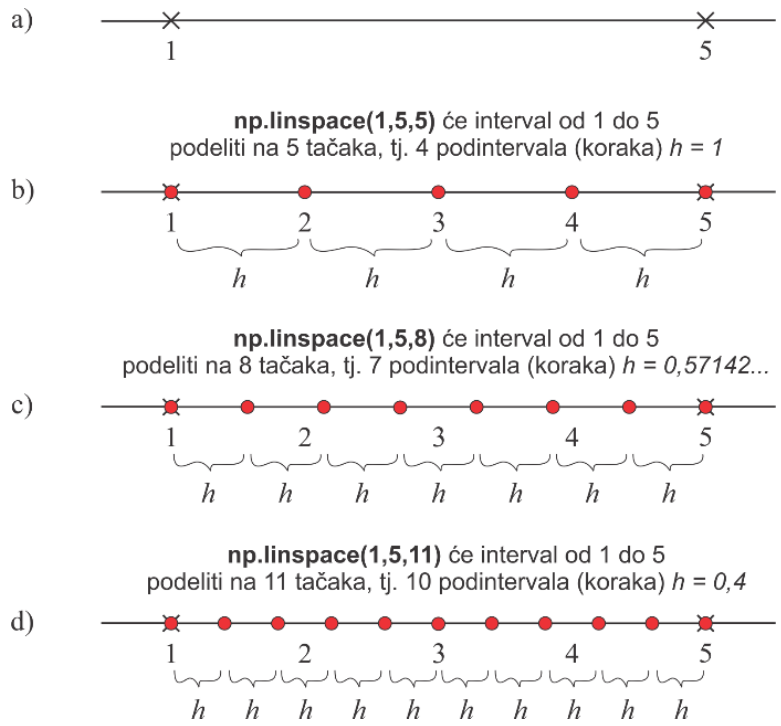
Primer 3.6. Upotreba nekih funkcija numpy biblioteke

```
1.  import numpy as np
2.
3.  niz1 = np.arange(1,10)
4.  niz2 = np.arange(1,10,1)
5.  niz3 = np.arange(1,5,0.5)
6.  niz4 = np.arange(1,10,3)
7.  niz5 = np.linspace(1,5,5)
8.  niz6 = np.linspace(1,5,7)
9.  niz7 = np.zeros(4)
10. niz8 = np.ones(4)
11.
12. print("niz1 je: ", niz1)
13. print("niz2 je: ", niz2)
14. print("niz3 je: ", niz3)
15. print("niz4 je: ", niz4)
16. print("niz5 je: ", niz5)
17. print("niz6 je: ", niz6)
18. print("niz7 je: ", niz7)
19. print("niz8 je: ", niz8)
>> niz1 je:  [1 2 3 4 5 6 7 8 9]
>> niz2 je:  [1 2 3 4 5 6 7 8 9]
>> niz3 je:  [1.  1.5 2.  2.5 3.  3.5 4.  4.5]
>> niz4 je:  [1 4 7]
>> niz5 je:  [1. 2. 3. 4. 5.]
>> niz6 je:  [1. 1.66666667 2.33333333 3. 3.66666667 4.33333333 5. ]
>> niz7 je:  [0. 0. 0. 0.]
>> niz8 je:  [1. 1. 1. 1.]
```

Što se tiče primene funkcije `arange()`, vidi se da će krajnja vrednost biti manje ili više udaljena od maksimalne vrednosti u zavisnosti od vrednosti koraka. Rezultat linija 3 i 4 će biti identičan, jer ukoliko nedostaje definisanje koraka *n* python podrazumeva da korak iznosi 1.

Linija 5 pokazuje kako definisana vrednost koraka može biti i decimalan broj. Rezultat linije 6 je niz koji ima svega tri člana, zbog veličine koraka. Uzimajući u obzir rezultate linija 3-6, zaključuje se da je primenom funkcije `arange()` lako kontrolisati korak rezultujućeg niza. Međutim, sa primenom ove funkcije treba biti jako oprezan zbog načina na koji reguliše poslednji član niza. Takođe, ukoliko je korak kod funkcije `arange()` veći od vrednosti maksimalne vrednosti, rezultujući niz će imati samo jedan element.

Primenom funkcije linspace često nije odmah vidljivo koliki će biti korak, ali korisnik može biti u potpunosti siguran da će se u rezultujućem nizu naći minimalna i maksimalna vrednost. Za definisanje nizova u okviru ovog kursa mnogo češće će se koristiti funkcija `linspace()`, pa je iz tog razloga dato i vizuelno objašnjenje ove funkcije na slici 3.1.



Slika 3.1. Vizuelno objašnjenje numpy funkcije `linspace()`

Dakle, funkcija `linspace()` će interval od minimalne do maksimalne vrednosti podeliti na $n-1$ podintervala, a u rezultujućem nizu naći će se n elemenata, uključujući i minimalnu i maksimalnu vrednost. Na slici 3.1.a dat je interval na brojevnoj pravi, od brojeva 1 do 5. Na slici 3.1.b je prikazano kako je pomenuti interval podeljen na pet tačaka, pri čemu ima 4 podintervala, od kojih je svaki jednak jedinici. Na slici 3.1.c je isti interval podeljen na 8 tačaka, dok je u na slici 3.1.d pokazano kako je pomenuti interval podeljen na 11 tačaka. Dakle, broj podintervala je uvek za jedan niži od broja tačaka na koliko se deli interval.

4. Liste, nizovi, torke i rečnici

Rad sa skupovima podataka je posebno efikasan kod programskog jezika python. Iz ovog razloga, python je čest izbor u nauci, posebno u situacijama kada je neophodno analizirati veliku količinu podataka. U pythonu, skupovi podataka se najčešće javljaju u dva oblika – kao lista ili kao niz. I jedan i drugi oblik skupova podataka imaju svoje prednosti i mane, a glavna razlika se ogleda u tome što liste mogu da sadrže različite tipove podataka, dok nizovi mogu da sadrže samo numeričke tipove podataka. U nastavku, biće predstavljeno kako se pomenuti oblici skupova podataka definišu, kao i neke osnovne operacije nad njima.

4.1. Definisanje lista i nizova

Liste se definišu praktično na isti način kao i druge promenljive. Prvo se da naziv promenljive, a zatim se posle znaka jednakosti elementi liste smeštaju u uglaste zagrade. Neki primeri definisanja listi su dati u primeru koji sledi.

Primer 4.1. Definisanje listi

```
1.  lista1 = [1,2,3,4,5]
2.  lista2 = ["pera","mika","zika","mile","djole"]
3.  lista3 = [1,2,"mika",3,"djole"]
4.
5.  print("Lista1 je: ", lista1)
6.  print("Lista2 je: ", lista2)
7.  print("Lista3 je: ", lista3)

>> Lista1 je:  [1, 2, 3, 4, 5]
>> Lista2 je:  ['pera', 'mika', 'zika', 'mile', 'djole']
>> Lista3 je:  [1, 2, 'mika', 3, 'djole']
```

U poslednjem primeru, vidi se da `lista1` sadrži samo brojeve, `lista2` samo stringove, dok je `lista3` kombinacija brojeva i stringova.

Python nema funkciju kojom se direktno definiše niz. Međutim, upotrebom **numpy** biblioteke, lista se lako može transformisati u niz. U te svrhe koristi se **numpy** funkcija `aray()`. Pored tipova podataka koje mogu sadržavati, liste i nizovi se razlikuju i po tome kako određene funkcije deluju na njih. Ova razlika između lista i nizova će biti demonstrirana u primeru 4.2.

Primer 4.2. Sabiranje lista i nizova

```
1.  import numpy as np
2.
3.  lista1 = [1,2,3,4]
4.  lista2 = [1,2,3,4]
5.
6.  niz1 = np.array(lista1)
7.  niz2 = np.array(lista2)
8.
9.  print("lista je: ",lista1)
10. print("niz je: ",niz1)
11.
12. print()
13.
14. print("Zbirovi listi i nizova\n")
15.
16. lista = lista1 + lista2
17. niz = niz1 + niz2
18.
19. print("zbir listi je: ", lista)
20. print("zbir nizova je: ", niz)

>> lista je:  [1, 2, 3, 4]
>> niz je:  [1 2 3 4]
>>
>> Zbirovi listi i nizova
>>
>> zbir listi je:  [1, 2, 3, 4, 1, 2, 3, 4]
>> zbir nizova je:  [2 4 6 8]
```

U poslednjem primeru u linijama 3 – 7 su definisane liste i nizovi upotrebom funkcije `np.array()`, dok su u linijama 16 i 17 definisani zbirovi dve liste i dva niza. Vidi se da će zbir dve liste dovesti zapravo do spajanja lista u jednu veliku listu. Sabiranje nizova će dovesti do formiranja jednog niza sa istim brojem elemenata kao i kod pojedinačnih nizova, sa članovima koji su zbirovi elemenata pojedinačnih nizova (prvi element `niza1` + prvi element `niza2`, drugi element `niza1` + drugi element `niza2`, itd). Dakle, zaključuje se da se za sprovođenje matematičkih operacija na skupovima podataka koriste nizovi.

4.2. Torke, funkcije zip() i zip(*), rečnici

U python programiranju se izraz „torka“ odnosi na nepromenjiv, uređen skup elemenata istog ili različitog tipa podataka. Torka se često koristi i za čuvanje više vrednosti koje su međusobno povezane, poput koordinate tačke. Ovaj tip skupova podataka se definiše tako što se vrednosti elemenata stavljaju u običnu zagradu, dok je funkcija kojom se npr. lista prebacuje u tip torke `tuple()`.

Sa druge strane, liste se takođe mogu spojiti u torku, upotrebom funkcije `zip()`. S tim u vezi, ukoliko se spajaju dve liste, rezultujuća torka će imati onoliko elemenata koliko imaju liste, a svaki element će biti torka sa dva elementa (uređena dvojka) tako da je prvi element prve liste na prvom mestu prve uređene dvojke, a prvi element druge liste na drugom mestu prve uređene dvojke, i tako dalje. Ukoliko se spajaju tri liste, rezultujuća torka će imati onoliko elemenata koliko imaju liste, a svaki element će biti torka sa tri elementa (uređena trojka) tako da je prvi element prve liste na prvom mestu prve uređene trojke, prvi element druge liste na drugom mestu prve uređene trojke, dok je prvi element treće liste na trećem mestu prve uređene trojke, i tako dalje.

Takođe, važno je napomenuti da postoji i funkcija sa obrnutom funkcionalnošću od `zip()`, a to je `zip(*)`, koja skup podataka gde su elementi uređene torke, raščlanjuje na odgovarajuće liste. U primeru koji sledi, biće predstavljen rad sa torkama i `zip()` funkcijama.

Primer 4.3. Rad sa torkama i `zip()` funkcijama

```
1.  torka1 = ('jabuka', 'breskva', 'sladoled')
2.  torka2 = (25, 30, 40)
3.
4.  lista1 = [1, 2, 3]
5.  lista2 = ['a', 'b', 'c']
6.
7.  parovi = zip(lista1, lista2)
8.
9.  print('rezultujuca torka je: ',list(parovi))
10.
11. torka = [('Marka','Opel'), ('Model', 'Astra'), ('kilometraza', 215000)]
12.
13. identifikatori, vrednosti  = zip(*torka)
14.
15. print('identifikatori su: ', identifikatori)
16. print('vrednosti su: ', vrednosti)
>> rezultujuca torka je:  [(1, 'a'), (2, 'b'), (3, 'c')]
>> identifikatori su:  ('Marka', 'Model', 'kilometraza')
>> vrednosti su:  ('Opel', 'Astra', 215000)
```

U datom primeru u prve dve linije su „ručno“ definisane dve torke. U linijama 4 i 5 su definisane dve liste, koje će u liniji 7 biti prebačene pomoću funkcije `zip()` u torku, tj. u skup podataka gde su elementi uređene dvojke. U liniji 9 se traži štampanje te torke, uz napomenu da se u svrhe štampanja torke dobijene `zip()` funkcijom, tip podataka promenljive `parovi` mora prvo pretvoriti u listu upotrebom funkcije `list()`.

U liniji 11 je opet ručno definisana jedna torka, konkretno opet se radi o listi uređenih dvojki. Upotrebom funkcije `zip(*)`, prvi elementi uređenih dvojki su redom smešteni u jednu listu,

a drugi elementi uređenih dvojki su redom smešteni u drugu listu. Pošto se “otpakuje” torka uređenih dvojki, neophodno je da se u liniji 11 sa leve strane jednakosti nađu dva objekta, po jedan za svaku listu koja će “ugostiti” odgovarajuće vrednosti iz torke.

U vezi sa torkama, vredi napomenuti još jednom da se redosled elemenata ne može menjati. Zbog nepromenljivosti, što osigurava integritet podataka, torka ima još jedno važno svojstvo, a to je efikasnost. Naime, zbog nepromenljivosti, torka ne zahteva dodatnu memoriju za promene ili proširivanje, zbog čega je brža i zahteva manje memorije od liste.

Rečnici su tipovi skupova podataka u obliku `"identifikator":„vrednost"`.[3] Kod rečnika je svaki identifikator jedinstven i odgovara mu izvesna vrednost. Rečnici su promenljivi, tako da se identifikatori i vrednosti mogu brisati i ažurirati. Definišu se tako što se forme “identifikator“:“vrednost“ stavljaju u vitričastu zagradu. Rečnici su vrlo korisni u python programiranju jer je pomoću njih moguće efikasno pretraživanje podataka prema vrednostima identifikatora. Takođe, vrlo su fleksibilni kada se radi o dodavanju, izmeni ili brisanju identifikatora i vrednosti. Ovaj tip podataka je u mogućnosti da kao identifikatore i vrednosti čuva različite tipove podataka. U primeru koji sledi, biće ukratko predstavljeno kako se može definisati rečnik i štampati vrednost iz istog.

Primer 4.4. Generisanje i rad sa torkama i rečnicima

```
1. osoba = {"ime": "Sanja", "godine": 25, "grad": "Novi Sad"}
2.
3. print(osoba["ime"])
4. print(osoba["godine"])
5. print(osoba["grad"])
>> Sanja
>> 25
>> Novi Sad
```

Važno je naglasiti da se dve liste mogu lako implementirati u rečnik, tako da elementi jedne liste budu identifikatori, a elementi druge liste vrednosti. To je predstavljeno u primeru koji sledi.

[3] Na engleksom jeziku za rečnik u pythonu stoji “key“:“value“, što bi moglo da se prevede i kao „ključ“:“vrednost“.

Primer 4.5. Implementiranje vrednosti dve liste u rečnik

```
1. kljucevi = ['marka', 'model', 'kilometraza']
2. vrednosti = ['Opel', 'Astra', 250000]
3.
4. torka = zip(kljucevi, vrednosti)
5.
6. recnik = dict(torka)
7.
8. print(recnik)
9. print(recnik['marka'])
>> {'marka': 'Opel', 'model': 'Astra', 'kilometraza': 250000}
>> Opel
```

Da bi se vrednosti dve liste implementirale u rečnik, te dve liste je prvo neophodno spojiti funkcijom `zip()` u jednu torku, a zatim se funkcijom `dict()` torka prebacuje u rečnik.

4.3. Indeksiranje, izdvajanje i zamena elemenata liste ili niza

U memoriji je svaki element liste ili niza indeksiran rednim brojem. Za rad sa listama i nizovima važno je napomenuti da python započinje indeksiranje od nule. Drugim rečima, prvi član liste ili niza python označava brojem 0, drugi član brojem 1, itd. Indeksi elemenata liste ili niza se mogu iskoristiti za izdvajanje elemenata. Izdvojeni element liste se može modifikovati, zameniti ili ukloniti. U narednim primerima biće demonstrirane neke od osnovnih operacija sa listama.

Primer 4.6. Rad sa listama, izdvajanje elemenata

```
1.  import numpy as np
2.
3.  lista = [1,2,"mika",4,"djole",5]
4.
5.  element1 = lista[0]
6.  element2 = lista[2]
7.  element3 = lista[5]
8.
9.  print("element1 je: ",element1)
10. print("element2 je: ",element2)
11. print("element3 je: ",element3)
12.
13. zbir_elemenata = element1 + element3
14. print("Zbir prvog i šestog elementa liste 'lista' je: ",zbir_elemenata)

>> element1 je:  1
>> element2 je:  mika
>> element3 je:  5
>> Zbir prvog i šestog elementa liste 'lista' je:  6
```

Dakle, izdvajanje elemenata listi i nizova je jednostavno, neophodno je samo da naziv liste prati uglasta zagrada sa indeksom elementa koji želi da se izdvoji. U linijama 5-7 izvršeno je

izdvajanje prvog, trećeg i šestog elementa liste. U liniji broj 13 je instruisano da se saberu prvi i šesti element liste.

Kada se zna indeks elementa njegova vrednost se može lako zameniti nekom drugom vrednošću. Zamena vrednosti elementa je demonstrirana u sledećem primeru.

Primer 4.7. Zamena vrednosti elementa liste ili niza

```
1.  import numpy as np
2.
3.  lista = [1,2,"mika",4,"djole",5]
4.
5.  print("lista je : ",lista)
6.
7.  lista[3] = "ana"
8.  print("nova lista je : ",lista)
9.
10. niz = np.array(lista)
11. print("niz je: ",niz)
12.
13. niz[3] = "mirka"
14. print("novi niz je : ",niz)

>> lista je :  [1, 2, 'mika', 4, 'djole', 5]
>> nova lista je :  [1, 2, 'mika', 'ana', 'djole', 5]
>> niz je:  ['1' '2' 'mika' 'ana' 'djole' '5']
>> novi niz je :  ['1' '2' 'mika' 'mirka' 'djole' '5']
```

U poslednjem primeru još jednom je praktično pokazano kako je neophodno znati indeks elementa sa kojim se želi manipulisati. Poznavanje indeksa elemenata liste ili niza nije problematično ukoliko je broj elemenata relativno mali, pa se može brzo prebrojati. Međutim, problemi sa indeksiranjem mogu nastati ukoliko lista ili niz sadrži ogroman broj elemenata.

Postavlja se pitanje, šta ako lista sadrži npr. 1159 elemenata, a korisnik želi da obriše baš poslednji element? Naravno, tvorci pythona su mislili i o tome, pa se u tom slučaju poslednji element niza indeksira brojem -1. Drugim rečima, ukoliko se želi izdvojiti vrednost poslednjeg elementa, to se može učiniti na sledeći način

```
poslednji_element = lista[-1]
```

Slično poslednjoj komandi, pretposlednji element se može izdvojiti na sledeći način

```
pretposlednji_element = lista[-2]
```

Često je za efikasan rad sa listama i nizovima pogodno koristiti ugrađene funkcije za identifikovanje najviših/najnižih vrednosti, kao i prebrojavanje broja elemenata liste. To se redom čini funkcijama `max()`, `min()` i `len()`. U primeru koji sledi, predstavljen je program koji koristi navedene funkcije.

Primer 4.8. Primena funkcija `max()`, `min()` i `len()`

```
1. lista = [1,50,60,100,1500]
2.
3. max_vrednost = max(lista)
4. min_vrednost = min(lista)
5. duzina_liste = len(lista)
6.
7. print('Maksimalna vrednost liste je: ',max_vrednost)
8. print('Minimalna vrednost liste je: ',min_vrednost)
9. print('Broj elemenata liste je: ',duzina_liste)
```

U primeru 4.8. najpre je definisana lista, a zatim su primenom pomenutih funkcija utvrđene maksimalna, minimalna vrednost, kao i broj elemanata te liste.

4.4. Brisanje vrednosti elementa liste ili niza

Brisanje elementa liste ili zamena njegove vrednosti je takođe jednostavna u programskom jeziku python. Naravno, opet je važno znati broj pod kojim je indeksiran element kojim želi da se manipuliše, međutim mogu se dati i instrukcije da se manipuliše sa elementima određenih vrednosti. U opštem slučaju, brisanje elementa liste može se postići upotrebom sledećih metoda:

- `remove()` – za brisanje elementa određene vrednosti
- `pop()` – za brisanje elementa prema njegovom indeksu
- `clear()` – za brisanje svih elemenata iz liste
- `del()` – za brisanje cele liste

Primena navedenih funkcija za brisanje člana liste je data u sledećem primeru.

Primer 4.9. Brisanje članova liste

```
1.  lista = [1,2,"mika",4,"djole",5]
2.  print("Lista je: ",lista)
3.
4.  lista.remove(4)
5.  print("Nova lista je: ",lista)
6.
7.  lista.remove("djole")
8.  print("Novija lista je: ",lista)
9.
10. lista.pop(2)
11. print("Još novija lista je: ",lista)
12.
13. lista.clear()
14. print("Još jos novija lista je: ",lista)
15.
16. del lista
17. print("Najnovija lista je: ",lista)
```

Dakle, u pythonu se lako mogu ukloniti elementi liste, bilo prema indeksu, bilo prema vrednosti. Napomena je i da će program iz poslednjeg primera prijaviti grešku u liniji 17, iz razloga što python neće imati šta da štampa što se tiče liste `lista`, jer je pomenuta lista u potpunosti obrisana u liniji 16.

Brisanje elemenata niza može se postići upotrebom **numpy** komande `delete()`. Argument ove komande je `(naziv_niza,indeks elementa)`. Primer upotrebe ove funkcije je demonstriran u sledećem primeru.

Primer 4.10. Brisanje elementa niza

```
1.  import numpy as np
2.
3.  lista = [1,2,"mika",4,"djole",5]
4.
5.  niz = np.array(lista)
6.  print("Niz je: ",niz)
7.
8.  niz = np.delete(niz,4)
9.  print("Novi niz je: ",niz)

>> Niz je:  ['1' '2' 'mika' '4' 'djole' '5']
>> Novi niz je:  ['1' '2' 'mika' '4' '5']
```

Za razliku od slučaja sa listom, u liniji 8 poslednjeg primera bilo je neophodno napisati `niz = np.delete(niz, 4)`, a ne samo `np.delete(niz,4)`.

5. Kondicionali i petlje

Svi programi do sada predstavljeni su *sekvencijalnog* tipa. To znači da se linije kôda izvršavaju jedna za drugom dok se ne dođe do kraja programa. Međutim, upotrebom određenih programskih struktura moguće je uticati na redosled izvršavanja linija kôda. Takve programske strukture se nazivaju *kontrolne strukture* i jedna od najpoznatijih kontrolnih struktura je kondicional ***if***.

U najopštijem slučaju, pod kondicionalom se podrazumeva programska struktura koja utvrđuje da li je ispunjen određen uslov ili ne. Ispunjenost uslova direktno određuje koja će linija kôda dalje biti izvršena, zbog čega kôd nema više čisto sekvencijalni karakter. Upotreba kondicionala ***if*** će biti demonstrirana u sledećem primeru.

Primer 5.1. Upotreba kondicionala

```
1. a = "Imate dovoljno para za patike"
2. b = "Nemate dovoljno para za patike"
3.
4. cena = 100 # evra
5.
6. print("Cena patika je: ", cena, ' evra')
7. print("Koliko imate para: ")
8. c = float(input())
9.
10. if c >= cena:
11.     print(a)
12. else:
13.     print(b)
```

Najpre, neophodno je istaći da python podrazumeva da se u okviru jedne *if* petlje nalazi sve što je u istoj meri uvučeno nakon dvotačke. To uvlačenje programi u kojima se kuca kôd obično sprovedu automatski nakon što se pritisne dugme enter nakon dvotačke. U liniji 8 je iskorišćena funkcija `input()` koja od korisnika traži unos, a taj unos je preko funkcije `float()` odmah prebačen u decimalnu vrednost. Uslov u liniji 10 proverava da li je iznos koji je uneo korisnik veći od cene patika, ili manji. Ukoliko je veći ili jednak, štampaće se vrednost promenljive `a`, dok će se u suprotnom štampati vrednost promenljive iz linije 2. Dati primer je jednostavan, jer je i sam uslov jednostavan – postoje samo dve mogućnosti i unešena suma prosto može ili ne može biti dovoljna za kupovinu patika.

Međutim, u realnosti se sreće sa složenijim situacijama, pa se utvrđivanje ispunjenosti uslova odigrava u nekoliko koraka. U složenijim slučajevima, neophodno je proveriti nekoliko uslova pre nego što se izda dalja naredba. Kada ima više od dva uslova, neophodno je da kondicional ***if*** bude praćen kondicionalom ***elif***, sve dok se ne prođe kroz sve moguće uslove. Ova situacija je demonstrirana na sledećem primeru, u kome je na osnovu unešene vrednosti količine novca sa kojom se raspolaže potrebno utvrditi koji automobil može da se priušti.

Primer 5.2. Upotreba složenijeg kondicionala

```
1.  a = "Opel Astra" # cena oko 15000 evra
2.  b = "BMW X1" # cena oko 30000 evra
3.  c = "Mercedes B200" # cena oko 30000 evra
4.  d = "Lada Vesta" # cena oko 12500 evra
5.
6.  print("Unesite koliko novca imate: ")
7.  ustedjevina = float(input())
8.
9.  if ustedjevina <= 10000:
10.     print("Ne moze Mercedes ili BMW, tesko i do Lade")
11. elif ustedjevina > 10000 and ustedjevina <= 12000:
12.     print("Ne moze Mercedes ili BMW, ali blizu ste Ladi")
13. elif ustedjevina > 12000 and ustedjevina <= 15000:
14.     print("Lada je sigurna, uz 'malu' doplatu mozda i do Astre")
15. elif ustedjevina > 15000 and ustedjevina <= 28000:
16.     print("Imate za Astru, ali vam jos fali za Mercedes ili BMW")
17. elif ustedjevina > 28000 and ustedjevina <= 30000:
18.     print("Imate za Astru, malo jos fali za Mercedes ili BMW")
19. else:
20.     print("Mozete da birate automobil koji cete da kupite")
```

U primeru 5.2. uslov je složeniji, jer unešeni iznos može da odgovara jednom od praktično četiri uslova. Kondicional u ovom primeru funkcioniše tako što se prvo proverava da li je unešeni iznos niži ili jednak vrednosti od 10000. Ukoliko jeste, ispisuje se *string* definisan u liniji 10. Ukoliko ovaj uslov nije ispunjen, prelazi se na proveru ispunjenosti ostalih uslova sve dok se ne utvrdi koji uslov je ispunjen. U zavisnosti od toga koji je uslov ispunjen, biće štampan odgovarajuć tekst.

Petlje su jedan od glavnih elemenata programiranja i služe za automatizovano ponavljanje određenog bloka kôda. Petlje se koriste za efikasno izvršavanje istih ili sličnih operacija na nizu podataka, a mogu se kombinovati sa kondicionalima kako bi se automatizovano obavljale određene aktivnosti dok je ispunjen neki uslov. Petlje su posebno korisne kada se radi sa velikom količinom podataka.

Neka postoji lista koja sadrži puno elemenata, na primer sve glavne gradove sveta i neka je iz nekog razloga neophodno odštampati svaki element te liste. U prethodnom poglavlju je predstavljeno kako se to može uraditi, međutim bilo bi zaista mučno pisati stotine linija kôda kako bi se ispisalo ime svakog od glavnih gradova. Umesto toga, moguće je iskoristiti neku od petlji, kojom će se funkcija za štampanje elemenata liste štampati onoliko puta koliko ima elemenata u listi. Dve najpoznatije vrste petlji su ***for*** i ***while***. U pythonu, petlje se lako implementiraju i omogućavaju laku automatizaciju procesa. U primeru koji sledi, prvo će biti predstavljeno nekoliko načina na koje se može upotrebiti ***for*** petlja.

Primer 5.3. Nekoliko načina upotrebe ***for*** petlje

```
1.  # 1. blok
2.  #-------------------------
3.  for i in [1, 2, 3, 4, 5]:
4.      print(i)
5.  #-------------------------
6.
7.  # 2. blok
8.  #-------------------------
9.  lista = [1,2,3,4,5]
10.
11. for i in lista:
12.     print(i)
13. #-------------------------
14.
15. # 3. blok
16. #-------------------------
17. for i in range(6):
18.     print(i)
19. #-------------------------
20.
21. # 4. blok
22. #-------------------------
23. for i in range(1,6):
24.     print(i)
25. #-------------------------
>> 1
>> 2
>> 3
>> 4
>> 5
```

U programu predstavljenom u poslednjem primeru prikazano je četiri različita načina kako se može primeniti ***for*** petlja, pri čemu se dobija identičan rezultat – štampanje brojeva od 1 do 5. Za upotrebu ***for*** petlje, neophodno je definisati brojač i vrednosti koje će brojač uzimati. Nakon završetka svakog kruga/iteracije, vrednost brojača se povećava za 1, sve dok se ne dostigne krajnja vrednost brojača. U poslednjem primeru, brojač je slovo „i“, koje se često koristi kao brojač u programiranju, mada nema nekog konkretnog pravila, jer brojač može da bude bilo koje slovo ili skup slova. Neophodno je istaći da python podrazumeva da se u okviru jedne ***for*** petlje nalazi sve što je u istoj meri uvučeno nakon dvotačke (isto kao i kod kondicionala).

U prvom i drugom bloku poslednjeg programa je predstavljeno kako se vrednosti brojača mogu definisati ručno. Ovo može biti izuzetno problematičan pristup, jer je često neophodno da brojač uzima velik skup vrednosti, međutim olakšavajuća okolnost je ta što postoje načini za automatsko generisanje vrednosti u zadatom intervalu.

Konkretno, u drugom bloku su vrednosti brojača najpre definisane u jednoj posebnoj listi, koje se zatim pozivaju u ***for*** petlji. Ovo je posebno pogodno, jer vrednosti u listi ne moraju biti

celobrojne, tj. brojač ne mora da uzima samo celobrojne vrednosti. U trećem i četvrtom bloku je predstavljeno kako se definisanje vrednosti brojača može postići funkcijom `range()`, uz napomenu da `range(n)` generiše skup vrednosti do vrednosti $n - 1$.

Pored ***for*** petlje, često je u upotrebi i ***while*** petlja. Ovaj tip petlje se koristi za ponavljanje određenog bloka kôda dok je ispunjen određeni uslov. Kod ove petlje, određeni zadatak se ponavlja sve dok je uslov istinit, a kad uslov postane netačan, petlja se završava. U tom smislu, jako je važno obratiti pažnju da se uslov dobro definiše. U slučaju da uslov nije dobro definisan, može doći do pojavljivanja beskonačne (tzv. mrtve) petlje, kada se petlja ponavnja beskonačno jer uslov nikada ne postane netačan. Primena ***while*** petlje je predstavljena u sledećem primeru.

Primer 5.4. Upotreba ***while*** petlje

```
1. broj = 1
2. while broj <= 5:
3.     print(broj)
4.     broj += 1
>> 1
>> 2
>> 3
>> 4
>> 5
```

U poslednjem programu je najpre data konkretna vrednost broja, a zatim se tokom ***while*** petlje proveravalo da li je ovaj uslov ispunjen, dok je na kraju pomenute petlje vrednost broja povećavana za 1.

Bitno je napomenuti da su kombinacije petlji i kondicionala posebno korisni. Zapravo, programiranje najvećim delom i jeste primena petlji i kondicionala. U sledećem primeru će biti predstavljeno kako se ***while*** petlja može kombinovati sa kondicionalom ***if***. Konkretno, biće predstavljeno kako se može proveravati da li je broj paran ili neparan, za šta je neophodno uvesti i kondicional ***if***.

Primer 5.5. Kombinovanje petlje i kondicionala

```
1. broj = 1
2.
3. while broj <= 6:
4.     if broj % 2 == 0:
5.         print('broj',broj,'jeste paran')
6.     else:
7.         print('broj',broj,'nije paran')
8.     broj += 1
>> broj 1 nije paran
>> broj 2 jeste paran
>> broj 3 nije paran
>> broj 4 jeste paran
>> broj 5 nije paran
>> broj 6 jeste paran
```

U poslednjem primeru najpre je definisana vrednost promenljive `broj`, a zatim se ***while*** petlji dala instrukcija da se izvršava sve dok je vrednost broja manja ili jednaka 6. U liniji 4 počinje primena kondicionala ***if***, preko koga se praktično proverava da li postoji ostatak pri deljenju sa brojem dva (u pythonu operator *%*, a često se drugde naziva *mod*). Ukoliko je uslov tačan, tj. ukoliko nema ostatka, u liniji 5 se traži štampanje tog broja i teksta da jeste paran, a ukoliko nije tačan (linija 6), u liniji 7 se traži da se taj broj štampa, ali uz štampanje i teksta da nije paran.

Kod primene petlji, često je od interesa da se petlja prekine i pre nego što se formalno ispuni inicijalni uslov. Kod pythona se u tom slučaju koristi funkcija `break()`. Jedan takav slučaj je predstavljen u sledećem primeru.

Primer 5.6. Prekidanje petlje

```
1. broj = 1
2.
3. while broj <= 10:
4.     print(broj)
5.     if broj == 5:
6.         print("Dostignut uslov, petlja se prekida")
7.         break
8.     broj += 1
>> 1
>> 2
>> 3
>> 4
>> 5
>> Uslov ispunjen, petlja se prekida
```

Inicijalni uslov je dat samom ***while*** petljom, kojim se traži iteriranje i štampanje broja sve dok je vrednost broja niža ili jednaka 10. Međutim, u liniji 5 se postavlja uslov da se proverava da li je vrednost broja jednaka 5. Ukoliko jeste, onda se u liniji 6 tražu štampanje teksta da je

postignut uslov i da se petlja prekida, a u liniji 7 se koristi funkcija `break()` kako bi se petlja zaustavila. Naravno, primena petlji, uslova i njihovih kombinacija je moguća u različitim varijantama. Unutar petlji se mogu naći čitave funkcije, definisati nove funkcije, a može se tražiti i izvršavanje različitih operacija nad listama, nizovima, rečnicima i ostalim objektima.

6. Rad sa direktorijumima, fajlovima i podacima

Još jedna od izuzetnih pogodnosti pythona jeste mogućnost da interaguje sa operativnim sistemom. Drugim rečima, upotrebom programskog jezika python moguće je izvršavati i standardne i napredne komande operativnog sistema, što između ostalog omogućava automatizaciju procesa. Jedna od najpoznatijih python biblioteka za interakciju sa operativnim sistemima jeste biblioteka **os**, koja je jedna od ugrađenih biblioteka pythona. U narednom delu, biće predstavljene sledeće mogućnosti ove biblioteke:

- utvrđivanje i promena lokacije radnog direktorijuma i listanje sadržaja,
- pravljenje i brisanje direktorijuma
- učitavanje fajlova
- pravljenje i brisanje fajlova
- upisivanje i brisanje sadržaja fajlova

6.1. Lokacija direktorijuma i listanje sadržaja

Utvrđivanje lokacije radnog direktorijuma se postiže upotrebom funkcije `getcwd()`, listanje sadržaja se vrši funkcijom `listdir()`, dok se promena radnog direktorijuma vrši funkcijom `chdir()`. Upotreba pomenutih funkcija biblioteke **os** je demonstrirana u sledećem primeru.

Primer 6.1. Lokacije direktorijuma i listanje sadržaja

```
1.  import os
2.
3.  lokacija1 = os.getcwd()
4.  print("Direktorijum u kome se nalazim je:", lokacija1)
5.
6.  print('\n')
7.
8.  sadrzaj_lokacija1 = os.listdir(lokacija1)
9.  print("Sadržaj prvog direktorijuma je: ",sadrzaj_lokacija1)
10.
11. os.chdir('../')
12.
13. print('\n')
14.
15. lokacija2 = os.getcwd()
16. print("Direktorijum u kome se nalazim je:", lokacija2)
>> Direktorijum u kome se nalazim je: D:\Google Backup\Anaconda

>> Sadržaj prvog direktorijuma je:  ['.ipynb_checkpoints',
'primena_os_biblioteke.ipynb',..., 'Untitled2.ipynb']

>> Direktorijum u kome se nalazim je: D:\Google Backup
```

U prvoj liniji kôda se učitava biblioteka **os**, dok se u trećoj liniji definište promenljiva koja će upotrebom funkcije `getcwd()` sadržavati informaciju o lokaciji foldera u kojem se korisnik trenutno nalazi. Promenljiva `lokacija1` će biti tipa *string*, i njen sadržaj se štampa preko linije 4. U liniji 8. se definiše promenljiva koja će čuvati informacije o sadržaju `lokacije1`. Da bi se otišlo u direktorijum koji se nalazi jedan nivo iznad trenutnog radnog, potrebno je kao u liniji 11 iskoristiti funkciju `chdir('../')` bez ikakvog definisanja promenljive. Nova lokacija se traži u liniji 15, a njen ispis se traži u liniji 16.

6.2. Pravljenje i brisanje direktorijuma

Jedna od osnovnih komandi operativnog sistema jeste pravljenje direktorijuma, što se pomoću **os** biblioteke čini upotrebom funkcije `mkdir()`, dok se brisanje postiže funkcijama `remove()` i `rmdir()`, uz napomenu da funkcija `remove()` briše putanju do fajla, ali ne može da obriše direktorijum. Sa druge strane, funkcija `rmdir()` briše prazan direktorijum. Sve tri poslednje funkcije su funkcije dva argumenta. Argument funkcije `mkdir()` je putanja do direktorijuma u kome se pravi novi direktorijum i naziv novog direktorijuma. Argumenti funkcije `remove()` su lokacija direktorijuma u kome se briše fajl i naziv fajla koji se briše, dok su argumenti funkcije `rmdir()` lokacija direktorijuma u kome se briše folder i naziv foldera koji se briše. Upotreba poslednje tri funkcije je demonstrirana na sledećem primeru.

Primer 6.2. Pravljenje i brisanje direktorijuma i foldera

```
1. import os
2.
3. lokacija = os.getcwd()
4. print("Direktorijum u kome se nalazim je:", lokacija)
5. sadrzaj = os.listdir()
6. print("Sadržaj direktorijuma je: ", sadrzaj)
7.
8. os.mkdir(r'D:\Google
Backup\Anaconda\Knjiga_programiranje\prazan_direktorijum')
9.
10. sadrzaj = os.listdir()
11. print("Sadržaj direktorijuma je: ", sadrzaj)
12.
13. os.rmdir("prazan_direktorijum")
14.
15. sadrzaj = os.listdir()
16. print("Sadržaj direktorijuma je: ", sadrzaj)
>>1 Direktorijum u kome se nalazim je: D:\Google
Backup\Anaconda\Knjiga_programiranje
>>2 Sadržaj direktorijuma je:  ['.ipynb_checkpoints', 'neki_direktorijum',
'primena_os_biblioteke.ipynb']
>>3 Sadržaj direktorijuma je:  ['.ipynb_checkpoints', 'neki_direktorijum',
'prazan_direktorijum', 'primena_os_biblioteke.ipynb']
>>4 Sadržaj direktorijuma je:  ['.ipynb_checkpoints', 'neki_direktorijum',
'primena_os_biblioteke.ipynb']
```

U liniji 3 iskorišćena je funkcija `getcwd()` kako bi se dobila lokacija trenutnog radnog direktorijuma, a u sledećoj liniji je traženo da se ta lokacija i ispiše (to je i prvi izlaz ovog programa). U liniji 5 se koristi funkcija `listdir()` kako bi promenljiva `sadrzaj` imala informacije o sadržaju pomenute lokacije, a ispis se traži u liniji 6. U liniji 8 se funkcijom `mkdir()` pravi novi direktorijum koji se naziva `prazan_direktorijum`. Nakon pravljenja novog direktorijuma, u liniji 10 se osvežava vrednost promenljive `sadrzaj`, čiji se ispis traži u liniji 11. Ovaj, treći, izlaz programa će sadržavati i naziv novokreiranog direktorijuma. U liniji 13 se koristi funkcija `rmdir()` za brisanje novog direktorijuma, u liniji 15 se osvežava vrednost promenljive `sadrzaj`, a njen ispis se ponovo traži u liniji 16. Drugi i četvrti izlaz programa će biti isti, jer je u međuvremenu obrisan direktorijum koji je napravljen.

6.3. Rad sa tekstualnim fajlovima

Učitavanje fajlova i izdvajanje podataka iz istih je jedna od najbitnijih aktivnosti kada se radi o analizi podataka, jer merni instrumenti često produkuju velike količine podataka koje je besmisleno unositi ručno u program za analizu. U tom smislu, programski jezik python nudi ogromne pogodnosti, jer postoji više gotovih biblioteka sa različitim mogućnostima za

učitavanje sadržaja fajlova. U ovom udžbeniku prvo će biti prikazan rad sa fajlovima koji sadrže tekst. Funkcije kojima se manipuliše sa tekstualnim fajlovima su:

- `open()` - za učitavanje sadržaja fajlova
- `read()`, `readline()` – za ispisivanje sadržaja fajlova
- `write()` – za upisivanje sadržaja fajlova
- `os.remove()` – za brisanje fajlova

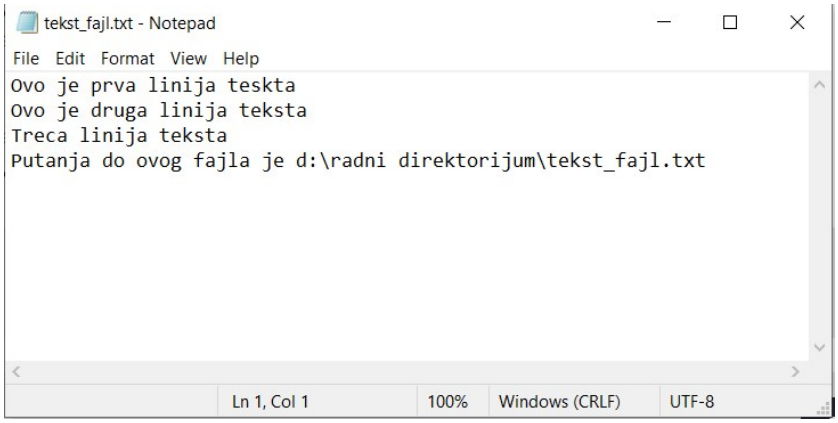
Pre nego što se demonstrira rad sa tekstualnim fajlovima, neophodno je napomenuti kako se u programskom jeziku python piše putanja. Putanja do nekog direktorijuma ili fajla se u programskom jeziku python može na napisati na dva načina:

Primer 6.3. Definisanje putanje

```
1. putanja1 = 'D:\\neki_folder\\test.txt'
2.
3. #ili
4.
5. putanja2 = r'D:\neki_folder\test.txt'
```

U opštem slučaju, kada se piše putanja do nekog fajla u komandnoj liniji, koristi se zapis tipa „D:\neki_folder\test.txt“. U pythonu, kao što je predstavljeno u primeru 6.3., simbol „\“ mora da se duplira (linija 1), ili pre cele putanje (uključujući i navodnike) mora da se stavi slovo „r“ (linija 5). U datom primeru, vrednosti promenljivih `putanja1` i `putanja2` su identične.

S obzirom da moderni merni instrumenti rezultate daju i u tekstualnoj formi, u ovom poglavlju biće demonstrirano kako se mogu učitavati, generisati i modifikovati upravo tekstualni fajlovi. Za početak, neophodno je napraviti jedan tekstualni fajl, „tekst_fajl.txt“, u radnom direktorijumu koji će se koristiti. Neka pomenuti tekstualni fajl sadrži četiri linije teksta, kao što je to predstavljeno na slici 6.1.



Slika 6.1. Sadržaj tekstualnog fajla

Sada je dostupan tekstualni fajl na kome ćemo primeniti funkcije za rad sa fajlovima.

6.3.1. Učitavanje i kreiranje tekstualnih fajlova

Učitavanje sadržaja fajla je jednostavno i vrši se pomoću ugrađene funkcije `open()`, dok se čitanje sadržaja vrši upotrebom funkcija `read()` (čitanje celog sadržaja ili definisanog broja karaktera) i `readline()` (čitanje linije po liniju). U opštem slučaju, `read()` je funkcija dva argumenta – naziva fajla i moda učitavanja fajla. Ukoliko se nalazi u trenutno aktivnom direktorijumu, što se tiče prvog argumenta dovoljno je napisati samo naziv fajla, a ukoliko se fajl nalazi u nekom drugom direktorijumu, onda je neophodno napisati i putanju do tog fajla. Što se tiče moda učitavanja, on može da uzima sledeće vrednosti:

„r“ – čitanje (eng. read), instruiše samo čitanje fajla, a ukoliko fajl ne postoji prijavljuje grešku (podrazumevana vrednost)

„a“ - dodavanje (eng. append), instruiše otvaranje fajla kako bi se nešto dodalo, a ukoliko fajl ne postoji kreira ga automatski

„w“ – upis (eng. write), instruiše otvaranje fajla kako bi se nešto upisalo, a ukoliko fajl ne postoji kreira ga automatski

„x“ – instruiše kreiranje fajla, a ako već postoji fajl sa istim imenom, prijavljuje grešku

„t“ – tekst (eng. text) instruiše da se radi o tekstualnom fajlu (podrazumevana vrednost)

„b“ – binarni tip podatka (eng. binary) instruiše da se radi o binarnom tipu podatka

Navedeni modovi učitavanja ne moraju biti navedeni prilikom upotrebe funkcije `open()`, u kom slučaju će biti uzete podrazumevane vrednosti. U primeru 6.4. najpre će biti demonstrirana upotreba učitavanja sadržaja tekstualnog fajla i njegovog ispisa.

Primer 6.4. Učitavanje i ispis sadržaja tekstualnog fajla

```
1. fajl = open(r'd:\radni direktorijum\tekst_fajl.txt')
2. print(fajl.read())
>>
Ovo je prva linija teskta
Ovo je druga linija teksta
Treca linija teksta
Putanja do ovog fajla je d:\radni direktorijum\tekst_fajl.txt
```

Dakle, u liniji 1 definisana je promenljiva `fajl`, a za vrednost joj je upotrebom funkcije `open()` dodeljen sadržaj fajla „tekst_fajl.txt“. Jednom kada je promenljivoj dodeljen neki tekst kao vrednost, moguće je koristiti funkcije `read()` i `readline()`. U liniji 2. primera 6.4. koristi se funkcija `read()` u okviru funkcije `print()` kako bi se u celosti ispisao sadržaj tekstualnog fajla koji je prethodno učitan.

Ukoliko je neophodno štampati određen broj karaktera, kao argument funkcije `read()` se stavlja broj koji označava koliko prvih karaktera da se ispiše, kao što je to demonstrirano na sledećem primeru.

Primer 6.5. Štampanje određenog broja karaktera

```
1. fajl = open(r'd:\radni direktorijum\tekst_fajl.txt')
2. print(fajl.read(10))
>> Ovo je prv
```

Kao izlaz, dati program će štampati samo prvih deset karaktera tekstualnog fajla. Često je neophodno štampati celokupne linije tekstualnog fajla. U tom slučaju, pogodnije je koristiti funkciju `readline()`, čija je upotreba data u sledećem primeru.

Primer 6.6. Upotreba funkcije `readline()`

```
1. fajl = open(r'd:\radni direktorijum\tekst_fajl.txt')
2. print(fajl.readline())
>> Ovo je prva linija teskta
```

Ukoliko je potrebno odštampati prve dve linije teksta, onda se ponovi linija 2, na način dat u sledećem primeru.

Primer 6.7. Štampanje prve dve linije teksta

```
1. fajl = open(r'd:\radni direktorijum\tekst_fajl.txt')
2. print(fajl.readline())
3. print(fajl.readline())
>>
Ovo je prva linija teskta
Ovo je druga linija teksta
```

Dakle, u poslednjem primeru je demonstrirano da će svako ponavljanje funkcije `readline()` štampati sledeću liniju teksta. Naravno, postavlja se pitanje, šta ako je od interesa štampati prvih deset linija tekstualnog fajla? U tom slučaju bilo bi nepraktično deset puta ponavljati jednu te istu liniju kôda, pa se ceo taj postupak može automatizovati upotrebom petlje, što je demonstrirano u primeru koji sledi.

Primer 6.8. Automatizacija `readline()` funkcije

```
1. fajl = open(r'd:\radni direktorijum\tekst_fajl.txt')
2.
3. for i in range(0,4):
4.     print(fajl.readline())
>>
Ovo je prva linija teskta
Ovo je druga linija teksta
Treca linija teksta
Putanja do ovog fajla je d:\radni direktorijum\tekst_fajl.txt
```

U poslednjem primeru, petlja definisana u linijama 3 i 4 će odštampati onoliko linija koliko se zada u okviru funkcije `range()`. U slučaju da se zada da se štampa više linija nego što ih ima u samom tekstualnom fajlu, python program će štampati prazne linije.

6.3.2. Upisivanje sadržaja u tekstualne fajlove

Sadržaj fajlova se lako može dopuniti ili zameniti, upotrebom funkcije `write()`. Naravno, prethodno je neophodno učitati fajl kao što je to demonstrirano u prethodnim primerima, a posebno je potrebno voditi računa o modu učitavanja. Ukoliko se fajl učita sa modom `a`, onda će se zadati sadržaj upisati dodavanjem na prethodni sadržaj, kao što je to demonstrirano u primeru koji sledi.

Primer 6.9. Dodavanje linije teksta

```
1. fajl = open(r'd:\radni direktorijum\tekst_fajl.txt','a')
2.
3. fajl.write("Dodata linija teksta \n")
4. fajl.close()
5.
6. fajl = open(r'd:\radni direktorijum\tekst_fajl.txt', 'r')
7. print(fajl.read())
>>
Ovo je prva linija teskta
Ovo je druga linija teksta
Treca linija teksta
Dodata linija teksta
```

Napomena je i da dodavanje `\n` u 3. liniji kôda dovodi do pravljenja novog reda nakon dodavanja teksta. Ukoliko bi se 3. linija koda izvršila bez `\n`, zadati tekst bi bio smešten u novu liniju tekstualnog fajla, ali bi svako sledeće ponavljanje istog kôda dodavalo tekst u nastavku poslednje linije tekstualnog fajla.

Ukoliko se fajl učita sa modom `w`, onda će se pre upisivanja novog sadržaja prethodni sadržaj brisati u potpunosti, kao što je to demonstrirano u sledećem primeru.

Primer 6.10. Upisivanje sa prethodnim brisanjem sadržaja fajla

```
1. fajl = open(r'd:\radni direktorijum\tekst_fajl.txt','w')
2.
3. fajl.write("Prethodni tekst ce biti obrisan")
4. fajl.close()
5.
6. fajl = open(r'd:\radni direktorijum\tekst_fajl.txt', 'r')
7. print(fajl.read())
>>
Prethodni tekst ce biti obrisan
```

Bez obzira koliko puta se ponovio ovaj kôd, on će uvek brisati prethodni sadržaj tekstualnog fajla i upisivati tekst zadat u liniji 3.

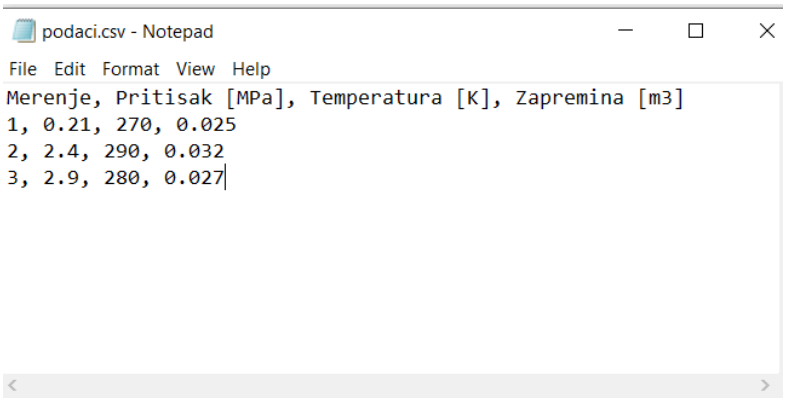
6.4. „Comma separated values“ (.csv) fajlovi

Rad sa tekstom demonstriran u prethodnim delovima je izuzetno bitan, međutim u nauci se mnogo češće sreću fajlovi u kojima su podaci tabelarno organizovani. Zarad jednostavnosti i uniformnosti, tabeliranje podataka se vrši po jasno utvrđenim pravilima. Jedan od najpoznatijih formata fajlova za organizaciju informacija je „csv“ (eng. **c**omma **s**eparated **f**iles). Kod ovih fajlova informacije su odvojene zarezima, a svaka nova linija u fajlu se odnosi na drugi skup vrednosti. Neka je data tabela sa merenim vrednostima kao u tabeli 6.1.

Tabela 6.1. Primer tabele sa podacima

Merenje	Pritisak [MPa]	Temperatura [K]	Zapremina [m^3]
1	0.21	270	0.025
2	0.24	290	0.032
3	0.29	280	0.027

Podaci iz date tabele bi u .csv fajlu bili sačuvani kao što je predstavljeno na slici 6.2.



Slika 6.2. Sadržaj .csv fajla u notepad editoru za prethodno navedenu tabelu

Dakle, u prvom redu fajla „podaci.csv“ su smešteni nazivi kolona. U drugom redu su date vrednosti za prvo merenje za svaku od kolona/veličina, u sledećem za drugo merenje i tako dalje. Sve vrednosti u svakom redu su odvojene zarezima. Iako se „ , “ najčešće koristi za odvajanje podataka kod .csv fajlova, u te svrhe može poslužiti bilo koji drugi simbol. Simbol koji razdvaja podatke kod .csv fajlova se naziva separator ilii graničnik.

Važnost .csv fajlova se ogleda u njihovoj jednostavnosti, uniformnosti i multiplatformskoj prepoznatljivosti. Različiti programi koriste različite formate za svoje baze podataka, međutim, svaki iole moderan program za rad sa podacima ima mogućnost rada sa .csv fajlovima. Python može da radi sa .csv fajlovima preko više biblioteka, a ovde će za početak biti demonstriran rad sa ugrađenom bibliotekom **csv**. Za početak potrebno je u radnom folderu kreirati .csv fajl sadržine kao u prethodnoj tabeli. Primer u kome se .csv fajl učitava preko **csv** biblioteke je sledeći.

Primer 6.11. Učitavanje sadržaja .csv fajla bibliotekom **csv**

```
1. import csv
2.
3. fajl = open(r'd:\radni direktorijum\podaci.csv')
4. citac = csv.reader(fajl)
5.
6. for linija in citac:
7.     print(linija)
>>
['Merenje', ' Pritisak [MPa]', ' Temperatura [K]', ' Zapremina [m3]']
['1', ' 0.21', ' 270', ' 0.025']
['2', ' 2.4', ' 290', ' 0.032']
['3', ' 2.9', ' 280', ' 0.027']
```

Kao i u prethodnim primerima, neophodno je prvo definisati putanju do .csv fajla (linija 3). Zatim je neophodno definisati promenljivu `citac` za čitanje .csv fajla (linija 4) u kojoj figuriše

funkcija `reader()` iz biblioteke **csv**. Linija 4 pravi jedan objekat tipa "csv.reader" čiji sadržaj ne može biti ispisan korišćenjem samo funkcije `print()`. Da bi se pročitao sadržaj objekta `citac` neophodno je to učiniti kroz petlju datu u linijiama 6 i 7.

Metod prikazan u poslednjem primeru ispisuje sadržaj celog .csv fajla. Međutim, često je neophodno prvo izvući informacije o nazivima kolona (eng. headers). To se u okvirima **csv** biblioteke može postići upotrebom funkcije `DictReader()`, a primer je sledeći:

Primer 6.12. Čitanje naziva kolona iz .csv fajla

```
1. import csv
2.
3. fajl = open(r'd:\radni direktorijum\podaci.csv')
4. citac = csv.DictReader(fajl)
5. nazivi_kolona = citac.fieldnames
6.
7. for linija in nazivi_kolona:
8.     print(linija)
>>
Merenje
Pritisak [MPa]
Temperatura [K]
Zapremina [m3]
```

Slično kao i u prethodnom primeru, najpre se definiše promenljiva `citac` (linija 4), a zatim se u liniji 5 definiše promenljiva `nazivi_kolona` koja će preko funkcije `citac.fieldnames` sadržavati nazive kolona.

Od sada pa nadalje, smatraće se da se fajl koji se učitava nalazi u istom folderu kao i program koji se piše. U tom slučaju, u funkcijama za otvaranje fajlova ne treba da se piše cela putanja do fajla koji se otvara, nego je dovoljno napisati samo naziv fajla koji se učitava. Drugim rečima, posmatrajući poslednji program, za učitavanje je dovoljna sledeća linija

```
fajl = open('podaci.csv')
```

6.5. Rad sa tabeliranim podacima pomoću biblioteke pandas

Biblioteka **pandas** je jedna od najpoznatijih python biblioteka za rad sa podacima. Nezaobilazni je alat u praktično svim naučnim oblastima kada se primenjuje programski jezik python. Rad sa tabelama i generalno sa organizovanim podacima sa ovom biblitekom ne zahteva napor, a korisnici već sa par osnovnih funkcija ove biblioteke stiču mogućnosti da efikasno rade sa podacima.

6.5.1. Učitavanje .csv fajla

Za početak, biće predstavljeno kako se učitava .csv fajl sa ovom bibliotekom. Konkretno, biće učitan fajl pod nazivom „g_const_1.csv“, koji sadrži podatke o merenjima perioda oscilovanja matematičkog klatna, na osnovu čega su izračunate vrednosti gravitacione konstante (dužina niti na koju je okačena kuglica iznosi 1 m). Podrazumeva se da je mereni period oscilovanja izražen u sekundama, a da su izračunate vrednosti gravitacione konstante u $\mathrm{m/s^2}$, ali su zarad jednostavnosti te informacije izostavljene iz pomenutog fajla.

Primer 6.13. Učitavanje .csv fajla preko biblioteke **pandas**

```
1. import pandas as pd
2.
3. df = pd.read_csv('g_const_1.csv')
4. df
```

U literaturi se biblioteka pandas najčešće označava skraćenicom **pd**. Učitavanje .csv fajla preko biblioteke **pandas** se vrši funkcijom `read_csv()`. Ova funkcija može da ima više argumenata, a osnovni je svakako naziv .csv fajla (ukoliko se fajl nalazi u radnom direktorijumu) ili putanja do željenog .csv fajla (ukoliko se fajl nalazi u nekom drugom direktorijumu). Funkcija `read_csv()` čita podatke iz .csv fajla i smešta ih u jedan objekat koji se naziva okvir podataka (eng. dataframe), pa otuda naziv promenljive `df` u liniji 3.[4]

Važno je istaći sledeće. U liniji 4 stoji samo `df`, što dovodi do ispisivanja sadržaja tog objekta (ostaviti sam naziv nekog objekta ili promenljive moguće je samo ukoliko se koristi Anaconda Jupyter, dok u ostalim slučajevima bi se prijavila greška po izvršavanju kôda). U liniji 4 se moglo pisati i `print(df)`, što bi isto ispisalo sadržaj `df`, ali na način iskorišćen u primeru ispis je lepši i izgleda kao na slici 6.3.

Kao što se može videti sa slike 6.3. učitavanje sadržaja .csv fajla u dataframe objekat zapravo indeksira svaki red. Poput situacije sa elementima listi i nizova, indeksiranje započinje brojem 0. Činjenica da je svaki red sa podacima indeksiran donosi fantastične mogućnosti za analizu tabeliranih podataka.

[4] Iz praktičnih razloga će nadalje biti korišćen izraz „dataframe“, a ne okvir podataka.

In [1]:
```
1 import pandas as pd
2
3 df = pd.read_csv('g_const_1.csv')
4 df
```

Out[1]:

	period	g_const
0	1.98	10.06
1	2.01	9.76
2	2.05	9.38
3	2.01	9.76
4	2.01	9.76
5	2.06	9.29
6	1.97	10.16
7	2.03	9.57
8	2.01	9.76
9	1.99	9.96

Slika 6.3. Ispis sadržaja .csv fajla preko **pandas** biblioteke

Važno je napomenuti da se može desiti da fajl koji korisnik želi da učita nema nazive kolona, nego da vrednosti kreću odmah od prvog reda. U tom slučaju, koristi se pogodnost argumenta `header` funkcije `read_csv()`. Ukoliko se stavi da vrednost argumenta `header` bude `None`, funkciji `read_csv()` će se staviti do znanja da nema naslova kolona i da se odmah krene sa očitavanjem vrednosti. Međutim, kako bi kasnije mogle da se izdvoje vrednosti, neophodno je definisati naslove kolona, što se može učiniti definisanjem vrednosti argumenta `names`. Konačno, modifikovana linija kôda za učitavanje .csv fajla kod kojeg nema naslova i u kojoj se definišu naslovi kolona kako bi se kasnije mogli pozvati je

```
df = pd.read_csv(fajl.csv,header=None, names=['colA', 'colB')
```

Napomenuto je da je „ , “ najčešći separator, a da se u te svrhe može upotrebiti praktično bilo koji drugi simbol ili skup simbola. Fajlovi sa podacima koji se dobijaju nakon merenja nekim digitalizovanim instrumentima često sadrže podatke, pri čemu se kao separator koristi neki drugi simbol, pa čak i samo razmak. U svakom slučaju, kakav god da je separator, funkcija `read_csv()` poseduje argument kojim se može definisati separator i tako bez problema učitati i neki drugi fajl koji nije formalno .csv formata. Na korisniku je da otvori fajl od interesa i da utvrdi šta je iskorišćeno kao separator podataka. Ukoliko je separator npr. razmak, onda je opšta linija kôda kojom se učitava takav fajl sledeća

```
df = pd.read_csv(fajl.csv, sep=' ')
```

U predstavljenoj liniji, vrednost argumenta `sep` reguliše šta će se tretirati kao separator. Vrednost ovog argumenta je simbol, ili skup simbola, stavljen pod navodnike. Naravno, svi do sada izloženi argumenti funkcije `read_csv()` mogu da se kombinuju na različite načine. Različiti instrumenti generišu različite tipove fajlova sa podacima, a velika pogodnost biblioteke **pandas** je što je njena funkcija `read_csv()` prilično fleksibilna i pažljivim podešavanjem vrednosti argumenata u principu može da se učita bilo koji fajl sa podacima.

6.5.2. Dobijanje opštih informacija o tabeli

Dalje će biti predstavljene neke od najznačajnijih mogućnosti funkcija biblioteke **pandas**, a prva od njih je funkcija `shape()`, predstavljena u sledećem primeru.

Primer 6.14. Funkcija `shape()` iz **pandas** biblioteke

```
1. import pandas as pd
2.
3. df = pd.read_csv('g_const_1.csv')
4. df.shape
>>
(10,2)
```

Jednostavna, ali izuzetno korisna, funkcija `shape()` kao izlaz daje važnu informaciju o dimenzijama objekta koji sadrži podatke. Drugim rečima, kao izlaz daje uređenu dvojku, koja sadrži vrednosti broja redova i kolona, respektivno. U primeru 6.14. vidi se da fajl „g_const_1.csv“ sadrži 10 redova i 2 kolone, na osnovu čega se zaključuje da je vršeno deset merenja perioda oscilovanja matematičkog klatna.

Druga važna funkcija biblioteke **pandas** u vezi sa inicijalnom analizom podataka učitanih iz datog fajla je `describe()`. Primenom ove funkcije dobijaju se osnovni statistički podaci za svaku kolonu dataframe objekta. Ova funkcija je demonstrirana u sledećem primeru.

Primer 6.15. Funkcija `describe` iz **pandas** biblioteke

```
1. import pandas as pd
2.
3. df = pd.read_csv('g_const_1.csv')
4. df.describe
```

Izlaz programa datog u prethodnom primeru je predstavljen na slici 6.4.

In [1]:

```
1 import pandas as pd
2
3 df = pd.read_csv('g_const_1.csv')
4 df.describe()
```

Out[1]:

	period	g_const
count	10.000000	10.000000
mean	2.012000	9.746000
std	0.028597	0.276775
min	1.970000	9.290000
25%	1.995000	9.617500
50%	2.010000	9.760000
75%	2.025000	9.910000
max	2.060000	10.160000

Slika 6.4. Izlaz koji daje funkcija `describe()`

Podaci dobijeni primenom funkcije `describe()` su vrlo značajni za inicijalnu analizu podataka. U poslednjem primeru, vidi se da srednja vrednost kolone „g_const“ (izračunate vrednosti gravitacione konstante) iznosi 9,74 m/s^2, što je u skladu sa poznatim vrednostima za ovu konstantu. Sa druge strane standardna devijacija iznosi 0,28 , što daje informacije o tačnosti merenja. Takođe je važno obratiti pažnju i na vrednosti parametra „count“. Ovaj parametar prebrojava koliko ima vrednosti u svakoj od kolona. U ovom konkretnom slučaju, obe kolone imaju po 10 vrednosti. Međutim, u slučaju velikih baza podataka, tj. .csv fajlova koji sadrže desetine hiljada redova, moguće je da pri prikupljanju vrednosti dođe do grešaka, zbog čega u nekim kolonama može biti manje vrednosti. Tada će parametar **count** imati različite vrednosti za različite kolone, što je jasan indikator da se baza podataka mora proveriti kako bi se otklonile greške.

6.5.3. Izdvajanje i uklanjanje podataka na nivou kolone

Ono što biblioteku **pandas** izdvaja od drugih jeste lakoća kojom se izdvajaju podaci. Ako je na primer potrebno izdvojiti vrednosti jedne od kolona iz prethodnog primera to se čini jednostavnim smeštanjem naziva kolone pod navodnicima u uglastu zagradu, kao što je demonstrirano u sledećem primeru.

Primer 6.16. Izdvajanje vrednosti jedne od kolona

```
1. import pandas as pd
2.
3. df = pd.read_csv('g_const_1.csv')
4. x_kolona = df['period']
5. x_kolona
```

Izlaz ovog programa će biti kao na slici 6.5.

```
Out[32]: 0    1.98
         1    2.01
         2    2.05
         3    2.01
         4    2.01
         5    2.06
         6    1.97
         7    2.03
         8    2.01
         9    1.99
         Name: period, dtype: float64
```

Slika 6.5. Izdvojene vrednosti kolone „period“

Novi dataframe objekat se može formirati tako što se iz početnog objekta ukloni jedna ili više kolona. Ovo je posebno korisno ukoliko početni dataframe objekat sadrži mnogo kolona, a neophodno je na primer izbaciti samo jednu kolonu. To se može postići upotrebom funkcije `drop(columns="naziv_kolone_koja_se_uklanja")`. U sledećem primeru, predstavljen je ovakav slučaj.

Primer 6.17. Uklanjanje jedne kolone iz pandas objekta

```
1. import pandas as pd
2.
3. df = pd.read_csv('g_const_1.csv')
4. novi_df = df.drop(columns='period')
5. novi_df
```

Kao izlaz, poslednji program će davati dataframe objekat koji sadrži samo preostalu kolonu sa podacima – kolonu sa izračunatim vrednostima gravitacione konstante. Ukoliko je neophodno ukloniti više od jedne kolone sa podacima, onda se u liniji 4 posle `columns=` u uglastu zagradu pod navodnike stavljaju nazivi kolona koje je neophodno ukloniti. U ovom konkretnom slučaju, `df.drop(columns=['period','g_const'])` bi uklonilo obe kolone iz dataframe objekta, pa bi rezultujući dataframe objekat bio prazan.

6.5.4. Izdvajanje pojedinih vrednosti iz kolone (funkcije loc i iloc)

Pomenuto je da je svaki red sa podacima indeksiran određenim brojem. Poznavanjem vrednosti indeksa, moguće je ispisati vrednosti određenog reda upotrebom funkcije `loc[[n]]` ili `loc[n]`, gde je `n` broj indeksa, što je demonstrirano na sledeća dva primera.

Primer 6.18. Izdvajanje podataka prema indeksu – varijanta sa dve uglaste zagrade

```
In [2]:
1  import pandas as pd
2
3  df = pd.read_csv('g_const_1.csv')
4
5  df.loc[[2]]
```

Out[2]:

	period	g_const
2	2.05	9.38

U ovoj varijanti, u programu je broj indeksa stavljen između dve uglaste zagrade, što dovodi do toga da se prikažu vrednosti reda sa pomenutim indeksom, međutim, ove vrednosti se iz ovakvog objekta ne mogu izdvojiti. Ukoliko je neophodno izdvojiti vrednosti iz datog reda, onda je potrebno broj indeksa smestiti u samo jednu uglastu zagradu, kao što je predstavljeno u sledećem primeru (uključujući izlaz programa).

Primer 6.19. Izdvajanje podataka prema indeksu – varijanta sa jednom uglastom zagradom

```
In [3]:
1  import pandas as pd
2
3  df = pd.read_csv('g_const_1.csv')
4
5  df.loc[2]
```

```
Out[3]: period     2.05
        g_const    9.38
        Name: 2, dtype: float64
```

Rezultat poslednjeg programa je praktično lista vrednosti, iz koje se lako mogu izdvojiti pojedinačne vrednosti. Na primer, ukoliko se želi izdvojiti vrednost perioda od 2.5 s, program će biti kao u primeru koji sledi.

Primer 6.20. Izdvajanje vrednosti preko funkcije `loc()`

```
1. import pandas as pd
2.
3. df = pd.read_csv('g_const_1.csv')
4.
5. a = df.loc[2][0]
6. print(a)
>>
2.05
```

Za izdvajanje pojedinačnih podataka iz dataframe-a posebno je korisna i funkcija `iloc()`. Argument ove funkcije je uređena dvojka, od kojih se prva vrednost odnosi na indeks (tačnije na broj reda), dok se druga vrednost odnosi na poziciju podatka u samom redu. Ova funkcija je predstavljena u sledećem primeru, gde je cilj bio ispisati prvu vrednost u šestom redu. Naravno, sve vreme se vodi računa da indeksiranje elemenata kreće od 0.

Primer 6.21. Izdvajanje vrednosti funkcijom `iloc()`

```
In [4]:  1 import pandas as pd
         2
         3 df = pd.read_csv('g_const_1.csv')
         4 print(df)
         5 print(df.iloc[5,0])
```

```
   period  g_const
0    1.98    10.06
1    2.01     9.76
2    2.05     9.38
3    2.01     9.76
4    2.01     9.76
5    2.06     9.29
6    1.97    10.16
7    2.03     9.57
8    2.01     9.76
9    1.99     9.96
2.06
```

U poslednjem primeru u liniji 4 je traženo da se odštampa sadržaj dataframe objekta. Zadatak u ovom primeru je da se štampa prva vrednost u šestom redu. Zbog toga funkcija `iloc()` ima sledeći oblik `iloc[5,0]`.

6.5.5. Pretvaranje pandas dataframe objekta u listu ili niz

Kao što je predstavljeno, vrlo lako se izdvajaju vrednosti nekih od kolona, međutim izazov je u tome što se te vrednosti i dalje nalaze u obliku **pandas** objekta. Ukoliko postoji potreba da

se nešto sa tim izdvojenim podacima i uradi, onda ih je neophodno smestiti u neku listu ili niz. Prebacivanje u listu se postiže upotrebom funkcije `values.tolist()`, što je demonstrirano u primeru koji sledi.

Primer 6.22. Pretvaranje pandas objekta u listu

```
1. import pandas as pd
2.
3. df = pd.read_csv('g_const_1.csv')
4. x_kolona = df['period']
5. x_kolona_lista = x_kolona.values.tolist()
6. x_kolona_lista
>>
[1.98, 2.01, 2.05, 2.01, 2.01, 2.06, 1.97, 2.03, 2.01, 1.99]
```

Dakle, u liniji 4 je najpre definisan novi dataframe objekat koji sadrži samo vrednosti iz kolone „period“. Zatim je u liniji 5 taj objekat upotrebom funkcije `values.tolist()` pretvoren u listu, a u liniji 6 je tražen ispis liste.

Umesto pretvaranja dataframe objekta u listu, od interesa može biti pretvaranje u niz. To se takođe lako čini, upotrebom funkcije `to_numpy()` (dakle, bez reči „values“), što je demonstrirano u primeru koji sledi.

Primer 6.23. Pretvaranje pandas objekta u niz

```
1. import pandas as pd
2.
3. df = pd.read_csv('g_const_1.csv')
4. x_kolona = df['period']
5. x_kolona_niz = x_kolona.to_numpy()
6. x_kolona_niz
>>
array([1.98, 2.01, 2.05, 2.01, 2.01, 2.06, 1.97, 2.03, 2.01, 1.99])
```

Dati program prebacuje izdvojeni dataframe objekat u niz. Naravno, prebacivanje u niz se moglo postići i prebacivanjem liste (dobijene prethodnim programom) u niz odgovarajućom funkcijom iz biblioteke **numpy**, ali to bi podrazumevalo dodatni korak, kao i učitavanje **numpy** biblioteke.

6.5.6. Upisivanje vrednosti u .csv fajl

Vrednosti elemenata listi ili nizova generisanih u programu je često korisno upisati u .csv fajl, kako bi se podaci sačuvali ili podelili. Upisivanje vrednosti elemenata listi ili nizova u kolone upotrebom biblioteke **pandas** se sprovodi tako što se vrednosti liste prvo pretvore u

dataframe objekat upotrebom funkcije `DataFrame()`, čiji je argument lista sa vrednostima elemenata, a zatim se od tog dataframe objekta pravi .csv fajl upotrebom funkcije `to_csv()`, čiji je glavni argument naziv .csv fajla u obliku stringa. Još jedan od važnih argumenata funkcije `to_csv()` je i `index`, koji može imati vrednosti `True` ili `False`. Podrazumevana vrednost ovog argumenta je `True` i u tom slučaju će u kreirani .csv fajl u prvoj koloni biti upisani indeksi svakog reda koji sadrži podatke, a u sledeće kolone vrednosti elemenata liste. Međutim, .csv fajlovi često ne sadrže indekse redova, jer se to lako vizuelizuje u programima za rad sa tabelama kao što su LibreOffice Calc ili Microsoft Excel, tako da se često stavlja da je vrednost pomenutog argumenta `False`. Program kojim se upotrebom **pandas** biblioteke upisuju vrednosti liste u jednu kolonu je dat u sledećem primeru.

Primer 6.24. Upisivanje vrednosti liste u kolonu .csv fajla

```
1. import pandas as pd
2.
3. lista1 = [1,2,3,4,5]
4.
5. df = pd.DataFrame(lista1)
6. df.to_csv('upisivanje.csv', index=False)
```

Program dat u poslednjem primeru će vrednosti liste `lista1` upisati u kolonu .csv fajla, a s obzirom da je vrednost argumenta `index=False`, indeksi redova se neće upisivati. Treba napomenuti da se nazivi kolona automatski dodeljuju od broja 0 pa nadalje, ukoliko se ne definiše drugačije, što će biti demonstrirano u jednom od sledećih primera.

Ukoliko se u .csv fajl želi upisati više lista sa vrednostima, tj. ukoliko će željeni .csv fajl sadržavati više kolona sa podacima, onda se DataFrame objekat mora definisati tako da je njegov argument lista u obliku *torke* (situacija kada su liste elementi liste). U jednom od prethodnih poglavlja je demonstrirano da se *torka* formira upotrebom funkcije `zip()`. Program u kome se vrednosti dve liste upisuju u .csv fajl u dve kolone je dat u narednom primeru.

Primer 6.25. Upisivanje u .csv fajl

```
1. import pandas as pd
2.
3. lista1 = [1,2,3,4,5]
4. lista2 = [10,20,30,40,50]
5.
6. torka = list(zip(lista1,lista2))
7.
8. df = pd.DataFrame(torka)
9. df.to_csv('probni_fajl.csv', index=False)
```

Dakle, u poslednjem primeru u liniji 3 i 4 su definisane liste sa odgovarajućim vrednostima. Vrednosti tih listi treba upisati u .csv fajl u dve kolone. U liniji 6 je definisana *torka* dobijena od te dve liste. Rezultujuća *torka* je lista koja sadrži pet elemenata, a svaki element te liste je takođe lista koja sadrži dva elementa. Tako definisana torka je argument funkcije `DataFrame()` koja *torku* pretvara u odgovarajući dataframe objekat (linija 8). Konačno, u liniji 9, upotrebom funkcije `to_csv()` se generiše .csv fajl u kome se svaka od vrednosti pomenutih listi nalazi raspoređena u odgovarajućoj koloni.

Često je pogodnije dati neki predefinisan naziv kolonama, umesto da budu nazvane rednim brojevima. Uz pomoć **pandas** biblioteke i to je moguće uraditi, upotrebom funkcije `add_prefix('naziv_kolone')`, što je demonstrirano u sledećem primeru.

Primer 6.26. Upisivanje u .csv fajl uz definisanje naziva kolona

```
1.  import pandas as pd
2.
3.  lista1 = [1,2,3,4,5]
4.  lista2 = [10,20,30,40,50]
5.
6.  torka = list(zip(lista1,lista2))
7.
8.  df = pd.DataFrame(torka).add_prefix('Kolona')
9.  df.to_csv('probni_fajl.csv', index=False)
```

Poslednji primer se u odnosu na pretposlednji razlikuje jedino u liniji 9, gde je dodata funkcija `add_prefix(„Kolona“)`, zbog čega će svaka od kolona biti nazvana „Kolona“ uz sukcesivno dodavanje broja kolone počevši od broja 0.

Ukoliko se nazivi kolona dosta razlikuju, onda metod `.add_prefix()` nije baš najpogodniji. U tom slučaju, pogodno je najpre definisati jednu listu sa nazivima kolona. Zatim se u liniji gde se definiše **pandas** dataframe objekat (linija 8 u poslednjem primeru) umesto `add_prefix()` može definisati argument `columns=` gde se nakon znaka jednakosti stavlja lista sa nazivima kolona. Takođe, nazivi kolona **pandas** dataframe objekta se mogu definisati i naknadno, gde se reč `columns` koristi praktično kao funkcija. Nakon što se definišu nazivi kolona **pandas** dataframe objekta, pri upisivanju u .csv fajl upotrebom `to_csv` funkcije, u .csv fajlu će se naći upravo ti nazivi kolona. Oba pomenuta metoda su predstavljena u programu u sledećem primeru.

Primer 6.27. Definisanje kolona u **pandas** dataframe objektu i .csv fajlu

```
1.  import pandas as pd
2.
3.  lista1 = [1,2,3,4,5]
4.  lista2 = [10,20,30,40,50]
5.  kolone = ['vreme','temperatura']
6.
7.  torka = list(zip(lista1,lista2))
8.
9.  df = pd.DataFrame(torka,columns=kolone)
10.
11. # ili su umesto linije 9 moglo da se stavi sledece:
12. # df = pd.DataFrame(torka)
13. # df.columns = kolone
14.
15. df.to_csv('probni_fajl.csv', index=False)
```

U poslednjem programu, u liniji 5 je definisana lista sa nazivima kolona. U liniji 9 je definisan **pandas** dataframe objekat, a pored argumenta koji definiše uređenu *torku* čije će se vrednosti naći u kolonama, definisan je i argument `columns` kome su dodeljene vrednosti `kolone`, tako da će nazivi kolona biti „vreme“ i „temperatura“, redom. Umesto linije 9, mogle su se iskoristiti linije 12 i 13, gde bi nazivi kolona konkretno bili definisani u liniji 13, nakon što je definisan **pandas** dataframe objekat. U principu, na ovaj način se u bilo kom delu kôda mogu promeniti nazivi kolona.

7. Vizuelizacija podataka u pythonu

Sam python nema mogućnosti vizuelizacije podataka. Međutim, tokom vremena razvijene su brojne biblioteke za ovaj programski jezik sa fantastičnim mogućnostima u pogledu vizuelizacije. Svakako jedna od najpoznatijih biblioteka za vizuelizaciju jeste biblioteka **matplotlib**, čije će mogućnosti biti predstavljene u ovom poglavlju.

7.1. Osnovne funkcije i mogućnosti biblioteke matplotlib

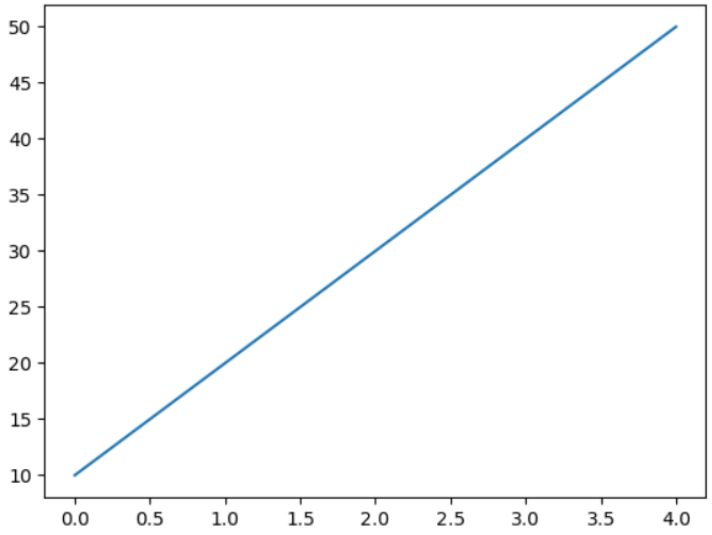
Sa bibliotekom **matplotlib** moguće je crtati različite tipove grafika, a najpre će biti predstavljeno kako se crtaju jednostavni linijski grafici i grafici sa rasutim tačkama (eng. scatter). Učitavanje biblioteke se vrši na standardan način, skraćenica koja se najčešće koristi je **plt**, dok je odgovarajuća funkcija `plot()`, gde se u argumentu nalaze promenljive koje

sadrže vrednosti koje se iscrtavaju na osama (prva pozicija argumenta se odnosi na *x*-osu, dok se druga pozicija odnosi na *y*-osu), kao i ostale opcije kojima se vrši fino podešavanje grafika.

U principu, funkciji `plot()` može da se dâ jedan jedini argument, a to su vrednosti na *y*-osi. Ukoliko se dâ samo jedan argument, onda će funkcija `plot()` smatrati da se na *x*-osi nalaze celobrojne vrednosti od broja 0 pa sve do broja koliko ima vrednosti na *y*-osi. Takođe, na ovaj način će se dobiti linijski grafik. Ovaj slučaj je predstavljen u sledećem primeru.

Primer 7.1. Crtanje grafika gde funkcija plot ima samo jedan argument sa pratećom slikom

```
In [2]:
1 import matplotlib.pyplot as plt
2
3 y = [10,20,30,40,50]
4
5 plt.plot(y)
6 plt.show()
```



U liniji 3 su definisane vrednosti na *y*-osi. U liniji 5 funkcija `plot()` ima samo jedan argument, a to su vrednosti na *y*-osi. Pošto promenljiva y ima 5 vrednosti, na *x*-osi će biti vrednosti od 0 do 4. Rezultat ovog programa je linijski grafik iscrtan tik ispod ćelije koja sadrži kôd u okviru Jupyter sveske, a tačke iz promenljive *y* se ne vide. Nakon funkcije `plot()` u sledećoj liniji mora da ide i funkcija `show()`, kako bi se grafik prikazao. Grafik se prikazuje neposredno posle ćelije u kojoj je pokrenut program.

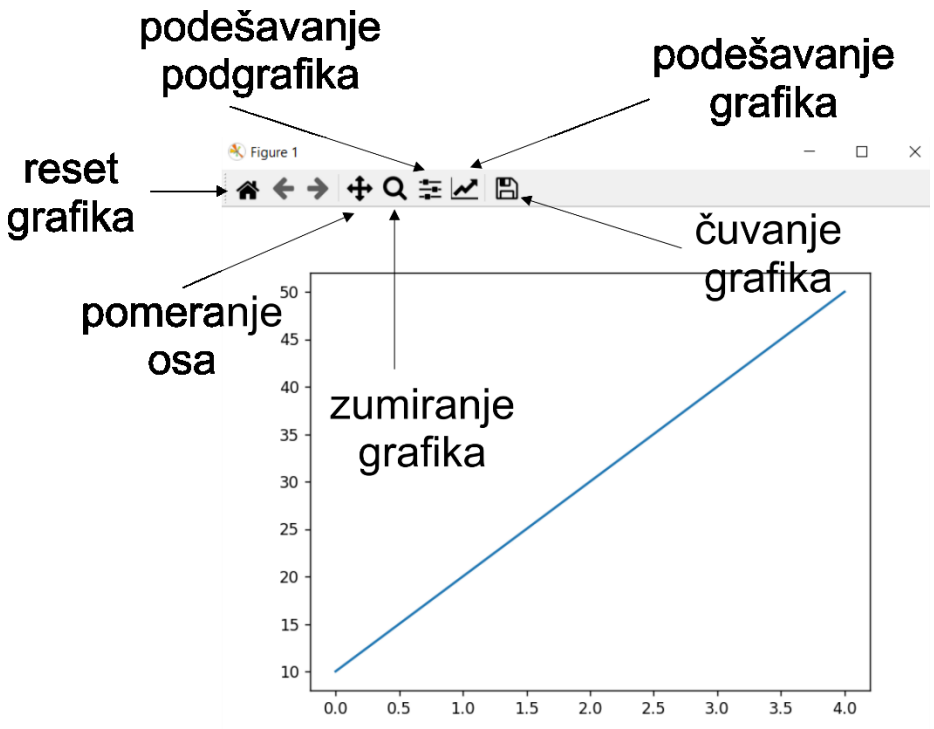
Pre nego što se pređe na ostale mogućnosti **matplotlib** biblioteke, pomenuće se i komanda `%matplotlib qt`. Upisivanjem ove komande neposredno posle učitavanja biblioteke dovešće do toga da se dobijeni grafik dobija kao poseban prozor sa komandama za različite modifikacije grafika i čuvanje istog. Treba napomenuti i da je u nekim IDE rešenjima, kao što

je to slučaj sa Thonny, pomenuta komanda učitana, pa se grafik automatski dobija u novom prozoru. Kada se koristi Jupyter Notebook to nije slučaj, zbog čega je neophodno iskoristiti pomenutu komandu, što je predstavljeno u primeru koji sledi.

Primer 7.2. Upotreba `%matplotlib qt` komande

```
1. import matplotlib.pyplot as plt
2. %matplotlib qt
3.
4. y = [10,20,30,40,50]
5.
6. plt.plot(y)
7. plt.show()
```

Važno je napomenuti da će komanda `%matplotlib qt` raditi ukoliko je **matplotlib** biblioteka instalirana sa podrškom za **PyQt** biblioteku. Ukoliko to nije slučaj, program će izbaciti grešku, pa je potrebno instalirati **PyQt** biblioteku[5]. Poslednji program će dati grafik u posebnom prozoru gde će njegova manipulacija biti lakša. Na slici 7.1. je predstavljen taj grafik u novom prozoru, zajedno sa objašnjenjima dugmića koji se tamo nalaze.



Slika 7.1. Matplotlib grafik u posebnom prozoru

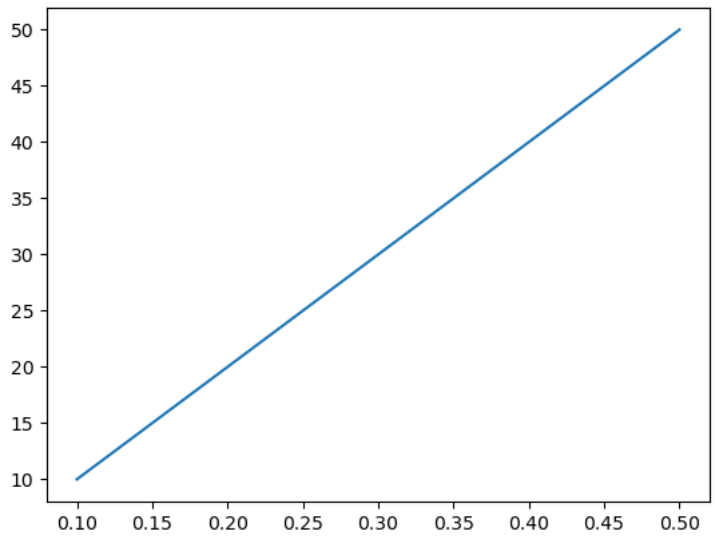
[5] uputstvo za instalaciju dato u **Dodatku 2**.

Funkcija dugmića kada je matplotlib grafik prikazan u novom prozoru je prilično intuitivna i neće biti razmatrana u detalje, ali će biti napomenuto da se čuvanje grafika može izršiti u nekoliko formata. U tom smislu, posebno je bitno napomenuti mogućnost da se grafik sačuva kako u bitmapiranom formatu, tako i u vektorskom formatu. Konkretno, što se tiče vektorskog formata, grafik je moguće sačuvati u .svg formatu (eng. **s**calable **v**ector **g**raphic). Ovaj format je izuzetno povoljan jer se grafici mogu raščlaniti na elemente (vektore), što omogućava njegovo fino modifikovanje. Jedan od programa koji može da otvori i radi sa .svg fajlovima je i Inkscape, besplatni program otvorenog kôda.

Ukoliko se funkciji `plot()` dodele dva argumenta, onda se kontrolišu i vrednosti koje će se naći na *x*-osi. Jedan takav slučaj je predstavljen na sledećem primeru.

Primer 7.3. Dva argumenta u funkciji plot sa pratećom slikom

```
1. import matplotlib.pyplot as plt
2. %matplotlib qt
3.
4. x = [0.1,0.2,0.3,0.4,0.5]
5. y = [10,20,30,40,50]
6.
7. plt.plot(x,y)
8. plt.show()
>>
```



Slika 7.2. Grafik dobijen programom iz primera 7.3.

Dakle, u primeru 7.3. u linijama 4 i 5 definisane su vrednosti na *x*- i *y*-osama, a da bi se iscrtao grafik u funkciju `plot()` se jednostavno kao argumenti stavljaju `x,y`.

7.2. Čuvanje grafika

Ukoliko se koristi komanda `%matplotlib qt` onda se čuvanje grafika postiže klikom na odgovarajuće dugme prozora koji se otvori (predstavljeno na slici 7.1), nakon čega se izabere lokacija gde se čuva fajl, kao i format u kome će se čuvati. Međutim, biblioteka **matplotlib** poseduje i odgovarajuću funkciju za čuvanje grafika, `savefig()`, pa se grafik može sačuvati i dodavanjem odgovarajuće linije kôda nakon funkcije za crtanje grafika. Ta linija kôda je sledećeg opšteg oblika

```
plt.savefig(r'd:/folder/ime_fajla.png',format='png',dpi=300)
```

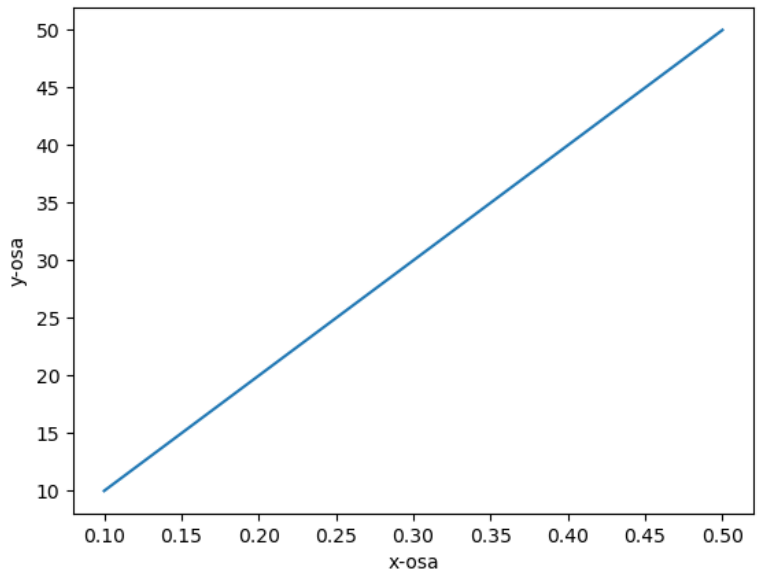
Iako deluje kao više posla, čuvanje grafika preko funkcije `savefig()` ima itekako pogodnosti, jer nudi veći stepen kontrole. U predstavljenoj liniji kôda, vidi se da u funkciji `savefig()` figuriše više argumenata. Najpre je neophodno definisati folder u kome se čuva grafik, zatim se preko vrednosti parametra `format` definiše format u kome će se čuvati grafik, a zatim i rezolucija, preko argumenta `dpi`. Iskustvo je pokazalo da grafici sačuvani preko funkcije izgledaju izuzetno dobro i pri rezolucijama nižim od 300 dpi, međutim treba imati u vidu da su zahtevi od strane izdavača naučne literature uglavnom da rezolucija grafika bude minimalno 300 dpi. Pored .png formata, mogao se izabrati i neki drugi format, npr. eps, svg, itd. Vredi napomenuti i da postoje još neki argumenti funkcije `savefig()` kojima se može fino podešavati čuvanje grafika, međutim oni prevazilaze obim ovog udžbenika.

7.3. Označavanje osa i naslovljavanje grafika

Ose grafika je neophodno označiti, a to postiže funkcijama `xlabel()` i `ylabel()`. Ove dve funkcije kao argument uzimaju *string*, čiji će sadržaj označiti *x*- i *y*-ose, respektivno. Upotreba ovih funkcija je demonstrirana u primeru koji sledi, praćeno slikom 7.3.

Primer 7.4. Označavanje osa

```
1.  import matplotlib.pyplot as plt
2.  %matplotlib qt
3.
4.  x = [0.1,0.2,0.3,0.4,0.5]
5.  y = [10,20,30,40,50]
6.
7.  plt.plot(x,y)
8.  plt.xlabel('x-osa')
9.  plt.ylabel('y-osa')
10. plt.show()
>>
```



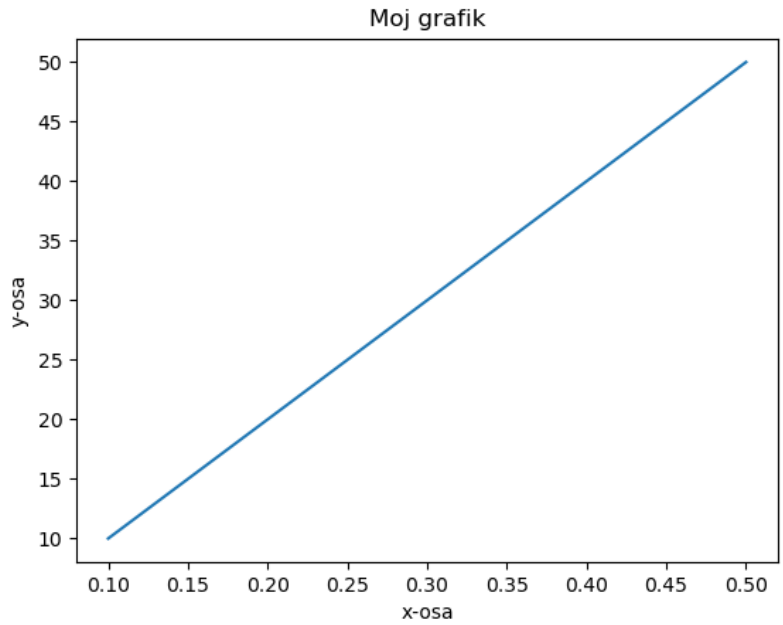
Slika 7.3. Rezultat programa u primeru 7.4. (označavanje osa)

Dakle, instrukcije za označavanje osa su date u linijama 8 i 9, za *x*- i *y*-osu, respektivno. Često je neophodno staviti i naslov grafika, a to se u slučaju matplotlib biblioteke postiže funkcijom `title()`. Kao i u slučaju `xlabel()` i `ylabel()` funkcija, argument funkcije title je *string*, čiji sadržaj će biti napisan kao naslov grafika. Označavanje grafika je demonstrirano u sledećem primeru, praćeno grafikom koji se dobija.

Primer 7.5. Naslovljavanje grafika

```
1.  import matplotlib.pyplot as plt
2.  %matplotlib qt
3.
4.  x = [0.1,0.2,0.3,0.4,0.5]
5.  y = [10,20,30,40,50]
6.
7.  plt.plot(x,y)
8.  plt.title('Moj grafik')
9.  plt.xlabel('x-osa')
10. plt.ylabel('y-osa')
11. plt.show()
>>
```

U poslednjem primeru, instrukcije za naslovljavanje grafika su date u liniji 8, a rezultujući grafik je dat na sledećoj slici 7.4. Korisno je napomenuti i da funkcije biblioteke **matplotlib** za naslovljavanje osa ili celog grafika mogu biti različito formatirane, a bez problema rade i sa ćiriličnim pismom.



Slika 7.4. Rezultat programa u primeru 7.5. (naslovljavanje grafika)

Često se ose grafika obeležavaju grčkim slovima ili nekim drugim specijalnim simbolima. Neretko se u okviru samih grafika nalaze i matematičke jednačine. Ovo se takođe može definisati pomoću **matplotlib** biblioteke i to na više načina. Naime, biblioteka **matplotlib** prepoznaje i Latex i unicode formate, pa korisnici mogu da biraju kako će definisati naslove osa ili naslov grafika.

Što se tiče Latex kôda, u slučaju **matplotlib** biblioteke njegov oblik je `$latex_kod_za_simbol$`. Na primer, Latex kôd za grčko slovo λ je `\lambda`, pa će odgovarajuća linija za obeležavanje *x*-ose biti `plt.xlabel('$\lambda$')`, dok bi linija za obeležavanje *x*-ose istim simbolom preko unicode formata bila `plt.xlabel('\u03bb')`.

Zarad preglednosti, u sledećoj tabeli će biti dati Latex i unicode kôdovi za grčki alfabet, uz napomenu da se pretragom na internetu lako može doći do kôdova za bilo koje druge specijalne simbole.

Tabela 7.1. Pregled Latex kôdova za najčešće korišćena slova grčkog alfabeta

Symbol	Latex	Unicode	Symbol	Latex	Unicode
Α	\Alpha	u0391	α	\alpha	u03b1
Β	\Beta	u0392	β	\beta	u03b2
Γ	\Gamma	u0393	γ	\gamma	u03b3
Δ	\Delta	u0394	δ	\delta	u03b4
Ε	\Epsilon	u0395	ε	\epsilon	u03b5
Ζ	\Zeta	u0396	ζ	\zeta	u03b6
Η	\Eta	u0397	η	\eta	u03b7
Θ	\Theta	u0398	θ	\theta	u03b8
Ι	\Iota	u0399	ι	\iota	u03b9
Κ	\Kappa	u039a	κ	\kappa	u03ba
Λ	\Lambda	u039b	λ	\lambda	u03bb
Μ	\Mu	u039c	μ	\mu	u03bc
Ν	\Nu	u039d	ν	\nu	u03bd
Ξ	\Xi	u039e	ξ	\xi	u03be
Ο	\Omicron	u039f	ο	\omicron	u03bf
Π	\Pi	u03a0	π	\pi	u03c0
Ρ	\Rho	u03a1	ρ	\rho	u03c1
Σ	\Sigma	u03a3	σ, ς	\sigma	u03c3
Τ	\Tau	u03a4	τ	\tau	u03c4
Υ	\Upsilon	u03a5	υ	\upsilon	u03c5
Φ	\Phi	u03a6	φ	\phi	u03c6
Χ	\Chi	u03a7	χ	\chi	u03c7
Ψ	\Psi	u03a8	ψ	\psi	u03c8
Ω	\Omega	u03a9	ω	\omega	u03c9

Svakako se preporučuje upotreba Latex kôda, zbog mogućnosti da se fino podešavaju jednačine i ostali simboli koji ovde nisu navedeni. Sa druge strane, unicode kôdovi se mogu često pronaći u upotrebi u raznim programskim jezicima, zbog čega su ovde i navedeni.

7.4. Linijski i „scatter“ grafici

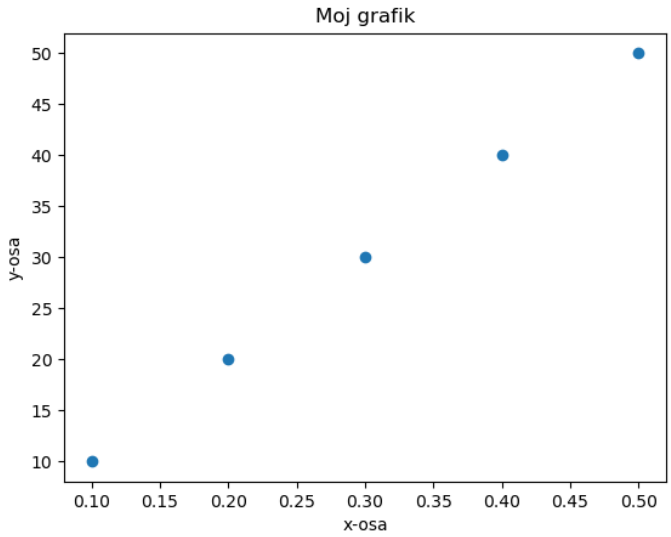
Umesto linije, mogu se dati instrukcije za crtanje tačaka na grafiku, za dobijanje tzv. rasutog (eng. scatter) grafika. To se može postići na dva načina – ili preko funkcije `plot()` ili preko funkcije `scatter()`. U slučaju da se koristi funkcija plot, dodatni argument je „o“. Dobijanje „scatter“ grafika na oba načina je demonstrirano u sledećem primeru, programi su dati uporedo i oni kao rezultat daju identičan grafik.

Primer 7.6. Crtanje „scatter“ grafika na dva načina

```
1.  import matplotlib.pyplot as plt
2.  %matplotlib qt
3.
4.  x = [0.1,0.2,0.3,0.4,0.5]
5.  y = [10,20,30,40,50]
6.
7.  plt.plot(x,y,'o')
8.  plt.title('Moj grafik')
9.  plt.xlabel('x-osa')
10. plt.ylabel('y-osa')
11. plt.show()
```

```
1.  import matplotlib.pyplot as plt
2.  %matplotlib qt
3.
4.  x = [0.1,0.2,0.3,0.4,0.5]
5.  y = [10,20,30,40,50]
6.
7.  plt.scatter(x,y)
8.  plt.title('Moj grafik')
9.  plt.xlabel('x-osa')
10. plt.ylabel('y-osa')
11. plt.show()
```

Dakle, oba programa će davati identičan grafik, kao što je to prikazano na slici 7.5.



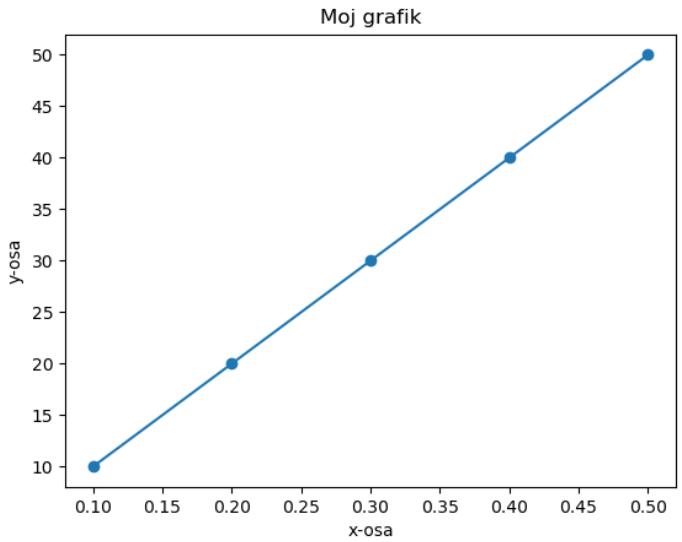
Slika 7.5. Rezultat programa iz primera 7.6. (dobijanje „scatter“ grafika)

Modifikacijom argumenta funkcije `plot()` može se postići grafik na kojem su vidljive i tačke i linije koje ih spajaju. U tom slučaju, poslednji argument funkcije plot je `'-o'` (za neisprekidanu liniju) ili `'--o'` (za isprekidanu liniju), što je prikazano u sledećem primeru.

Primer 7.7. Crtanje grafika gde su tačke povezane linijom

```
1.  import matplotlib.pyplot as plt
2.  %matplotlib qt
3.
4.  x = [0.1,0.2,0.3,0.4,0.5]
5.  y = [10,20,30,40,50]
6.
7.  plt.plot(x,y,'-o')
8.  plt.title('Moj grafik')
9.  plt.xlabel('x-osa')
10. plt.ylabel('y-osa')
11. plt.show()
```

Dati program će kao rezultat dati grafik prikazan na sledećoj slici.



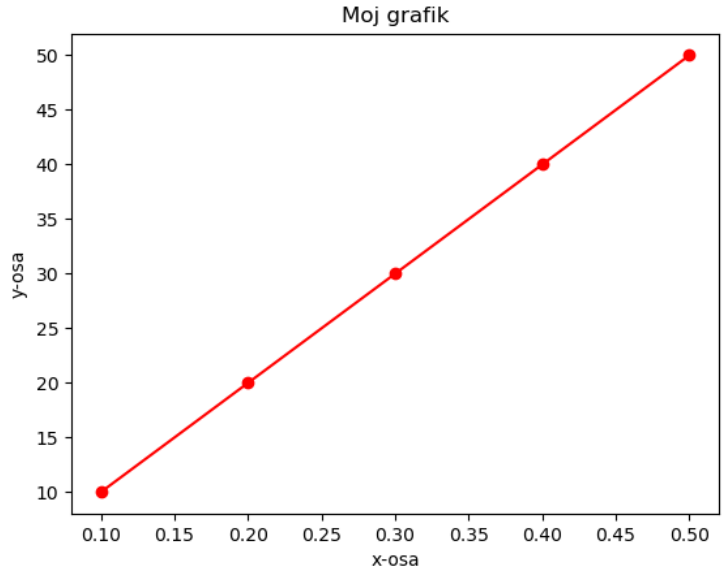
Slika 7.6. Rezultat programa iz primera 7.7. („scatter + line“ grafik)

Naravno, može se birati boja tačaka i/ili linija, jednostavnim dodavanjem argumenta `color='naziv_boje'`. Tako na primer, ako se želi crvena linija/tačka, argument u okviru odgovarajuće funkcije će biti `color='red'`. Dobijanje grafika sa obojenom linijom je demonstrirano u sledećem primeru.

Primer 7.8. Crtanje grafika gde su tačke i linija crvene boje

```
1.  import matplotlib.pyplot as plt
2.  %matplotlib qt
3.
4.  x = [0.1,0.2,0.3,0.4,0.5]
5.  y = [10,20,30,40,50]
6.
7.  plt.plot(x,y,'-o',color='red')
8.  plt.title('Moj grafik')
9.  plt.xlabel('x-osa')
10. plt.ylabel('y-osa')
11. plt.show()
```

Rezultat ovog programa je grafik predstavljen na slici 7.7.



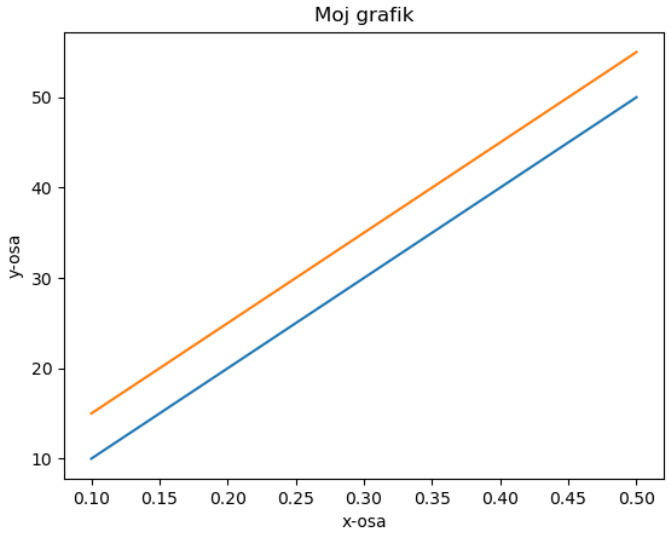
Slika 7.7. Rezultat programa iz primera 7.8. (definisanje boje linije/tačke)

Važno je istaći da se na jednom grafiku može iscrtati više linija. Za svaku od linija koja se crta piše se po jedna linija kôda sa funkcijom `plot()` ili `scatter()`, a na grafiku će biti prikazane sve linije koje su definisane takvim linijama kôda do linije u kojoj piše `plt.show()`.

Primer 7.9. Crtanje više linija na jednom grafiku

```
1.  import matplotlib.pyplot as plt
2.  %matplotlib qt
3.
4.  x = [0.1,0.2,0.3,0.4,0.5]
5.  y1 = [10,20,30,40,50]
6.  y2 = [15,25,35,45,55]
7.
8.  plt.plot(x,y1)
9.  plt.plot(x,y2)
10. plt.title('Moj grafik')
11. plt.xlabel('x-osa')
12. plt.ylabel('y-osa')
13. plt.show()
```

U linijama 4-6 su definisane vrednosti za *x*- i *y*-ose. U oba slučaja *x*-osa će imati iste vrednosti, dok se vrednosti na *y*-osama razlikuju. U liniji 8 je instrukcija da se crta linija čije su vrednosti date linijama 4 i 5, a u liniji 9 je instrukcija da se crta linija čije su vrednosti date linijama 4 i 6. Obe linije će biti na istom grafiku, jer je linija `plt.show()` na samom kraju. Program predstavljen u primeru 7.9. će dati grafik predstavljen na slici 7.8.



Slika 7.8. Rezultat programa iz primera 7.9. (crtanje dve linije na istom grafiku)

Python i biblioteka **matplotlib** automatski dodeljuju druge boje svakoj od linija, tako da to ne mora da se kontroliše. Naravno, korisnik može i sam da bira boju kojom će svaka od linija biti nacrtana, jednostavnim dodavanjem argumenta `color` u okviru funkcije `plot()`.

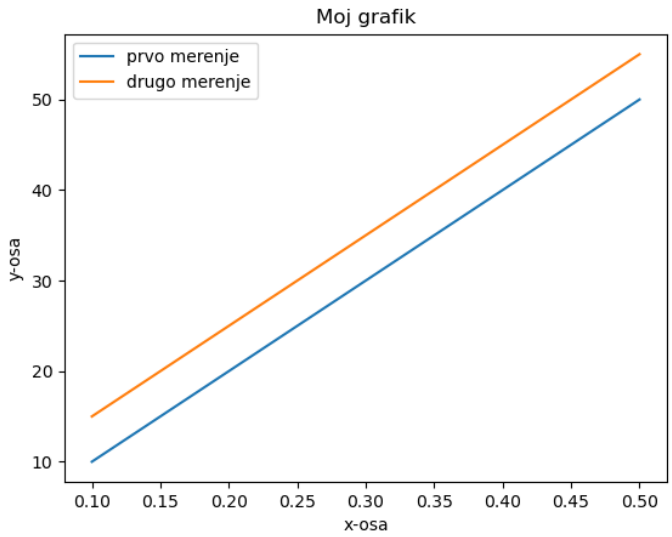
7.5. Dodavanje legendi

Kad god postoji više linija na jednom grafiku, neophodno je i da postoji legenda koja ukazuje na šta se svaka od linija odnosi. Dodavanje legendi u **matplotlib** biblioteci se vrši u dva koraka. Prvi korak je da se u okviru svake od funkcija `plot` u programu doda argument `label='naziv_linije'`, dok je drugi korak da se nakon funkcije `plot()` doda još jedna linija koda, `legend()`, koja daje instrukciju da se na grafiku štampa i legenda. Dodavanje legende na poslednji grafik je demonstrirano u sledećem primeru.

Primer 7.10. Dodavanje legende na grafik

```
1.  import matplotlib.pyplot as plt
2.  %matplotlib qt
3.
4.  x = [0.1,0.2,0.3,0.4,0.5]
5.  y1 = [10,20,30,40,50]
6.  y2 = [15,25,35,45,55]
7.
8.  plt.plot(x,y1,label='prvo merenje')
9.  plt.plot(x,y2,label='drugo merenje')
10. plt.title('Moj grafik')
11. plt.xlabel('x-osa')
12. plt.ylabel('y-osa')
13. plt.legend()
14. plt.show()
```

U poslednjem primeru informacije o nazivima linija su date u linijama 8 i 9, u okvirima funkcije `plot()`. U liniji 13 je data instrukcija da se na grafiku postavi i legenda. Program iz poslednjeg primera daje grafik predstavljen na slici 7.9.



Slika 7.9. Rezultat programa iz primera 7.10. (dodavanje legende)

Poziciju legende automatski reguliše **matplotlib** biblioteka i izabraće onaj deo na grafiku gde najmanje smeta linijama koje su iscrtane. Međutim, pozicija legende može se definisati i od strane korisnika, dodavanjem argumenta `loc` u okviru funkcije `legend()`. Argument `loc` može imati dva tipa vrednosti – numeričku ili string. U sledećoj tabeli su navedene sve moguće vrednosti koje argument `loc` može da ima.

Tabela 7.1. String i numeričke vrednosti koje argument `loc` može da uzima

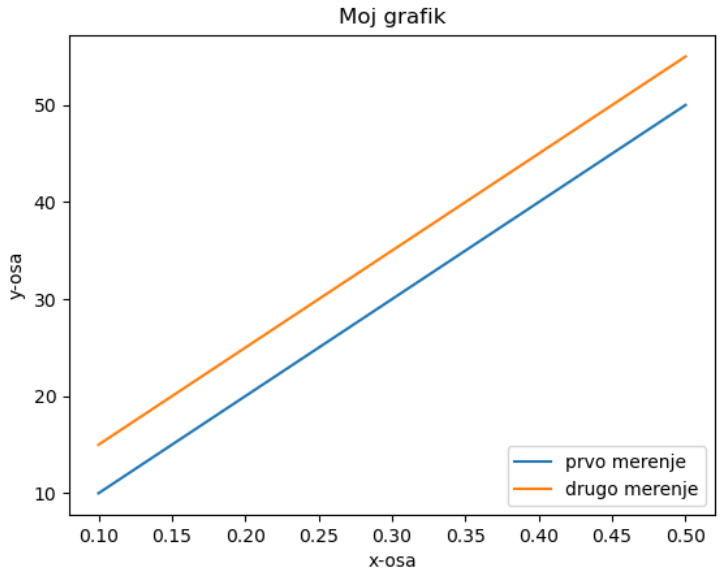
string	numerički	Pozicija legende
'best'	0	Najbolja procena
'upper right'	1	Gore desno
'upper left'	2	Gore levo
'lower left'	3	Dole levo
'lower right'	4	Dole desno
'right'	5	Desno
'center left'	6	Centralno levo
'center right'	7	Centralno desno
'lower center'	8	Dole centralno
'upper center'	9	Gore centralno
'center'	10	Centar

Dakle, svejedno je da li će argument `loc` imati vrednost `'lower left'` ili `3`, pozicija legende će biti na istom mestu, u ovom slučaju dole levo. Definisanje pozicije legende je demonstrirano u primeru koji sledi.

Primer 7.11. Definisanje pozicije legende

```
1.  import matplotlib.pyplot as plt
2.  %matplotlib qt
3.
4.  x = [0.1,0.2,0.3,0.4,0.5]
5.  y1 = [10,20,30,40,50]
6.  y2 = [15,25,35,45,55]
7.
8.  plt.plot(x,y1,label='prvo merenje')
9.  plt.plot(x,y2,label='drugo merenje')
10. plt.title('Moj grafik')
11. plt.xlabel('x-osa')
12. plt.ylabel('y-osa')
13. plt.legend(loc=4) # ili plt.legend(loc='lower right')
14. plt.show()
```

Dakle, u liniji 13 je potpuno svejedno da li će pisati `plt.legend(loc=4)` ili `plt.legend(loc='lower right')`, pozicija legende će biti dole desno. Program iz poslednjeg primera daje grafik kao na slici 7.10.



Slika 7.10. Rezultat programa iz primera 7.11. (pozicioniranje legende)

7.6. Crtanje grafika sa podacima iz fajlova

U poslednjem primeru podaci o tačkama koje definišu liniju koja se crta na grafiku su unošeni ručno. Međutim, ručno unošenje vrednosti u program se retko praktikuje, kako zbog trajanja tako i zbog mogućnosti da se pogreši. Najčešće su podaci o merenjima sačuvani u fajlovima kao što su .csv fajlovi, pa se učitavanjem tih fajlova i uzimanjem neophodnih podataka mogu dobiti neophodne informacije za crtanje grafika. U ovom delu biće pokazano kako se kombinacijom biblioteka **pandas** i **matplotlib** podaci iz .csv fajla mogu učitati, a zatim i nacrtati odgovarajući grafik.

U te svrhe, koristiće se „g_const_2.csv“ fajl, koji sadrži rezultate eksperimenta za računanje gravitacione konstante, merenjem perioda oscilovanja za različite dužine matematičkog klatna. Program za njegovo učitavanje upotrebom biblioteke **pandas**, kao i izlaz u obliku dataframe-a, je dat u sledećem primeru.

Primer 7.12. Učitavanje .csv fajla sa vrednostima eksperimenta za gravitacionu konstantu

In [1]:
```
1 import pandas as pd
2
3 df = pd.read_csv('g_const_2.csv')
4 df
```

Out[1]:

	duzina	period	period^2
0	0.50	1.41	1.99
1	0.55	1.48	2.19
2	0.60	1.54	2.37
3	0.65	1.61	2.59
4	0.70	1.68	2.83
5	0.75	1.72	2.96
6	0.80	1.80	3.24
7	0.85	1.85	3.42
8	0.90	1.90	3.61
9	0.95	1.92	3.69
10	1.00	2.03	4.12

Iz pomenutog .csv fajla se vidi da je izvršeno 10 merenja perioda oscilacija matematičkog klatna za dužine matematičkog klatna od 0,50 m do 1,00 m. U prvoj koloni su date vrednosti dužine matematičkog klatna, u drugoj koloni vrednosti perioda oscilovanja matematičkog

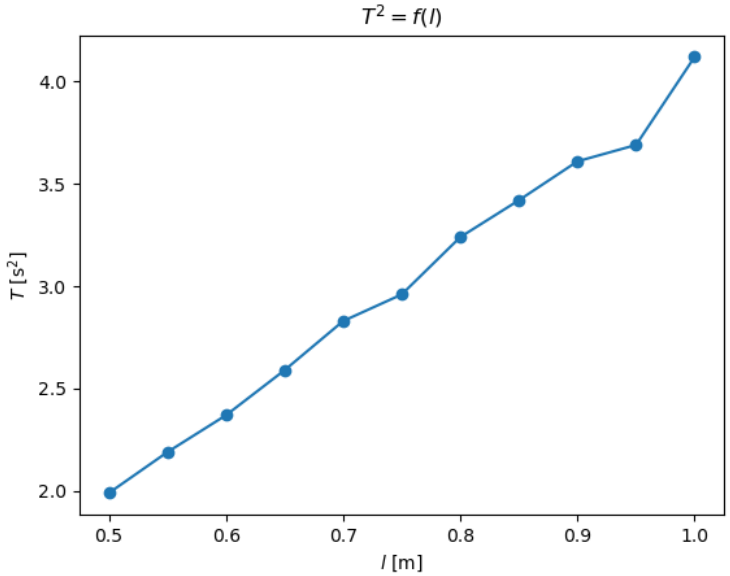
klatna, dok su u trećoj koloni date vrednosti kvadrata perioda oscilovanja matematičkog klatna. Kako bi se na osnovu nagiba prave izračunala vrednost gravitacione konstante, neophodno je nacrtati grafik na kome su na *x*-osi vrednosti dužine matematičkog klatna, a na *y*-osi vrednosti kvadrata perioda oscilovanja matematičkog klatna[6]. Za to je neophodno primeniti ono što je predstavljeno u poslednja dva poglavlja. Primer programa kojim se to postiže je sledeći:

Primer 7.13. Crtanje grafika $T^2 = f(l)$ uzimajući podatke iz .csv fajla

```
1.  import pandas as pd
2.  import matplotlib.pyplot as plt
3.  %matplotlib qt
4.
5.  df = pd.read_csv('g_const_2.csv')
6.
7.  x_df = df['duzina']
8.  x = x_df.values.tolist()
9.  y_df = df['period^2']
10  y = y_df.values.tolist()
11
12. plt.plot(x,y,'-o')
13. plt.title('$T^2=f(l)$')
14. plt.xlabel('$l$ [m]')
15. plt.ylabel('$T$ [$\mathrm{s^2}$]')
16. plt.show()
```

U primeru 7.13. kreće se od učitavanja biblioteka **pandas** i **matplotlib**. U liniji 5 se učitava sadržaj .csv fajla kao dataframe objekat. U liniji 7 definiše se promenljiva `x_df` koja će sadržavati vrednosti iz kolone „duzina“ u obliku dataframe objekta. U liniji 8 `x_df` objekat se konvertuje u listu upotrebom funkcije `values.tolist()` i sada je definisana lista vrednosti za *x*-osu. Isto se odvija u linijama 9 i 10, samo za *y*-osu. Ostatak kôda je crtanje grafika, uz napomenu da se u linijama kôda za definisanje naslova grafika i osa koristi Latex zapis (`\mathrm` se koristi da jedinice ne bi bile italic nego uspravne). Program dat u poslednjem primeru daje grafik kao na slici 7.11.

[6] Ovaj eksperiment će detaljno biti obrađen u sledećem poglavlju.



Slika 7.11. Grafik dobijen programom iz primera 7.13. (crtanje grafika $T^2 = f(l)$ učitavanjem podataka iz .csv fajla)

Vizualizacija podataka je nezaobilazna u istraživanju, a **matplotlib** biblioteka je samo jedna od biblioteka za python kojom se mogu vizuelizovati podaci. Tokom godina, ova biblioteka se ustalila kao standard, ne samo zahvaljujući fantastičnoj funkcionalnosti, nego i zahvaljujući činjenici da je to jedna od najbolje dokumentovanih biblioteka.

8. Fitovanje podataka

Fitovanje podataka predstavlja konstrukciju krive koja najbolje odgovara (fituje) dobijenim podacima i to je jedan od najvažnijih koraka u obradi podataka. Programski jezik python poseduje više alata za fitovanje podataka i procenu validnosti dobijenog fita. U ovom poglavlju će kroz nekoliko konkretnih primera biti demonstrirano kako se uz pomoć pythona mogu grafički predstaviti rezultati, fitovati podaci i proceniti kvalitet fita. Konkretno, biće demonstrirano kako se programski jezik python može iskoristiti za fitovanje podataka i određivanje željenih veličina iz realnih primera koji se često obrađuju u okviru sledećih eksperimentalnih vežbi:

- Određivanje vrednosti gravitacione konstante pomoću matematičkog klatna
- Određivanje koeficijenta samoindukcije kod RL kola
- Određivanje perioda poluraspada protaktinijuma

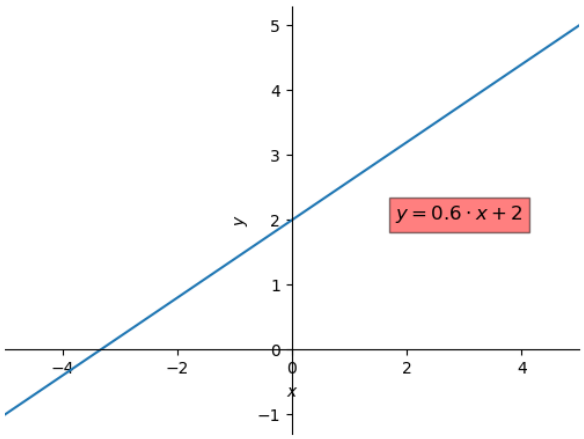
Pre konkretnih primera, biće dat pregled matematičkih funkcija kojima se podaci najčešće fituju, zatim će biti uvedene najpoznatije funkcije iz python biblioteka koje se koriste u vezi sa fitovanjem. Kod konkretnih primera, najpre će biti ukratko izloženi pomenuti eksperimenti, a nakon svakog izlaganja će biti demonstrirano kako se mogu napisati python programi za fitovanje podataka.

8.1. Najčešće matematičke funkcije za fitovanje podataka

Linearni fit. Najjednostavniji zadatak za fitovanje jeste fitovanje linearnom funkcijom oblika:

$$y = ax + b\,. \tag{8.1}$$

U ovom slučaju, fit je prava linija gde parametar a predstavlja nagib prave, dok b predstavlja odsečak na y-osi. Primer jedne linearne funkcije je dat na slici 8.1.

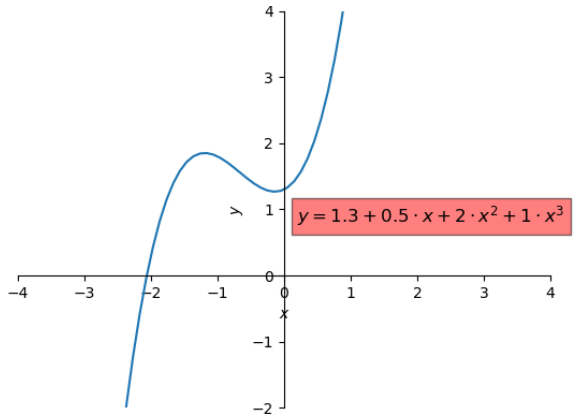


Slika 8.1. Primer linearne funkcije sa vrednostima $a = 0{,}6$ i $b = 2{,}0$

Polinomna regresija. U slučaju polinomne regresije podaci se fituju stepenim redom sledećeg oblika

$$y = a + bx + cx^2 + dx^3 + \cdots. \tag{8.2}$$

Osnovna karakteristika polinomne regresije je stepen polinoma, što predstavlja najviši stepen promenljive x. Grafički prikaz jednog polinoma trećeg reda je dat na slici 8.2.



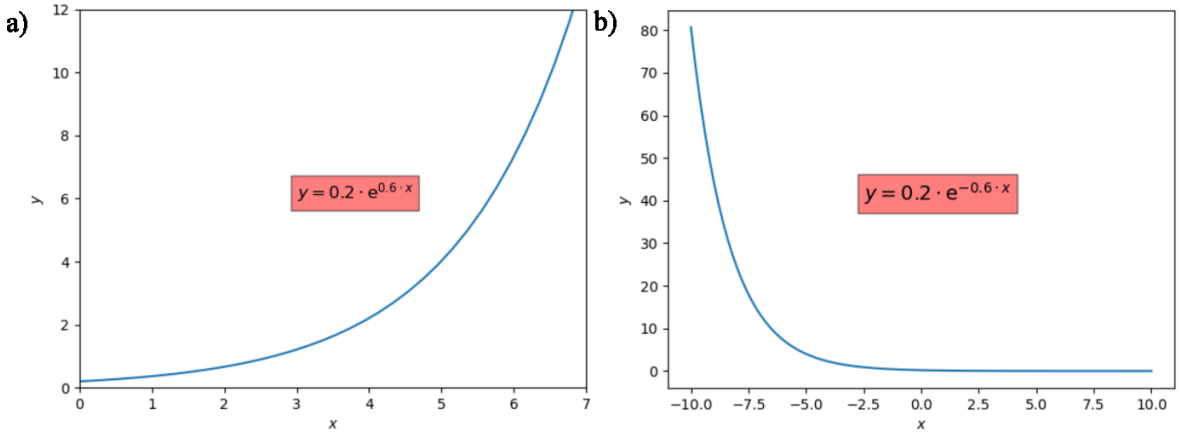
Slika 8.2. Grafički prikaz polinoma trećeg reda sa
vrednostima koeficijenta $a = 1{,}3$, $b = 0{,}5$, $c = 2{,}0$ i $d = 1{,}0$

U opštem slučaju, neki eksperimentalni podaci se mogu fitovati polinomom prvog, drugog, trećeg, itd, stepena. Iz jednačine 8.2. se vidi da je polinom prvog stepena zapravo jednačina prave linije, pa se linearni fit može dobiti i funkcijama za polinomno fitovanje. Više o tome sledi u narednim odeljcima.

Eksponencijalni rast i eksponencijalni pad. U ovom slučaju, jednačina krive koja opisuje eksponencijalni rast ili opadanje vrednosti je

$$y = ae^{\pm bx} . \qquad (8.3.)$$

U poslednjoj jednačini, kada je vrednost parametra b pozitivna, radi se o jednačini eksponencijalnog rasta. U slučaju negativne vrednosti ovog parametra, radi se o jednačini koja opisuje eksponencijalni pad. Na slici 8.3. su data dva primera, jedan za eksponencijalni rast, drugi za eksponencijalni pad, gde su vrednosti parametara $a = 0{,}2$ i $b = \pm 0{,}6$, respektivno.

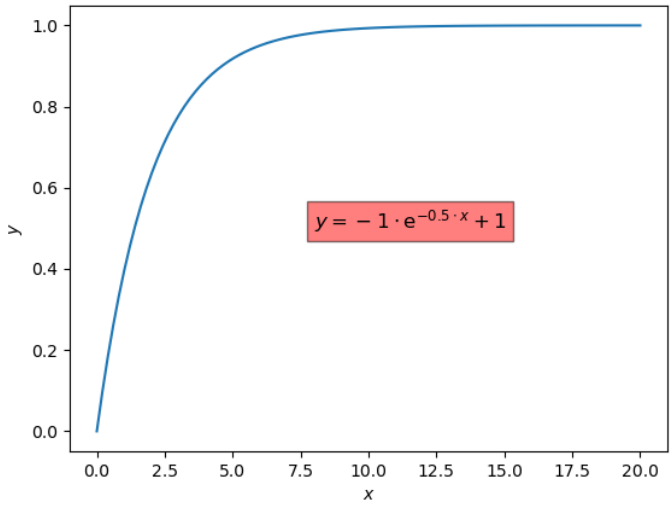


Slika 8.3. Eksponencijalni a) rast ($a = 0{,}2$, $b = 0{,}6$) i b) pad ($a = 0{,}2$, $b = -0{,}6$)

Eksponencijalni rast do limita. U ovom slučaju dolazi do rasta vrednosti funkcije, ali do određenog limita, nakon čega dolazi praktično do saturacije vrednosti (do formiranja platoa). Jednačina u ovom slučaju glasi

$$y = ae^{bx} + c\,. \tag{8.4.}$$

Ukoliko su vrednosti parametara $a = -1, b = -0{,}5$ i $c = 1$, data funkcija ima željeni oblik (u smislu da se pojavljuje jasan plato) koji je predstavljen na slici 8.4.



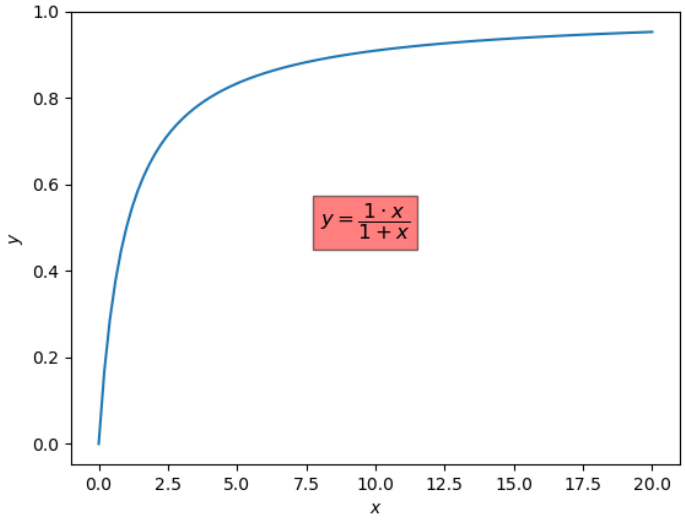
Slika 8.4. Grafik funkcije koja opisuje eksponencijalni rast sa limitom

U konkretnom primeru vidi se da vrednost funkcije naglo raste do vrednosti x oko 2,5, a zatim značajno usporava da bi na vrednosti oko $x = 8$ došlo do saturacije.

„Plato“ kriva. Pored prethodno predstavljene funkcije, pojavljivanje platoa se može opisati i sledećom funkcijom

$$y = \frac{ax}{b + x}\,. \tag{8.5.}$$

Vrednosti parametara za koje se može reprodukovati kriva sa platoom su $a = 1$ i $b = 1$. Takav slučaj je ilustrovan na slici 8.5.



Slika 8.5. Plato kriva

Funkcija 8.5. ne daje jasno definisan plato poput funkcije 8.4., ali je svakako adekvatna za fitovanje podataka u slučaju kada se javlja saturacija.

8.2. Funkcije za fitovanje u pythonu

8.2.1. Funkcija `curve_fit()`

Fitovanje podataka se uz pomoć pythona može se sprovesti na više načina. U ovom odeljku ukratko će biti predstavljeno nekoliko načina za fitovanje podataka, dok će konkretni primeri fitovanja biti izloženi u sledećim odeljcima.

U funkciji koja se fituje na podatke podatke (u daljem tekstu fit funkcija), pored promenljive, figuriše jedan ili više parametara, tako da je zadatak fitovanja da se pronađu optimalne vrednosti tih parametara kako bi se dobila kriva (ili prava linija) koja najbolje opisuje podatke. Drugim rečima, vrši se optimizacija vrednosti parametara, sve dok se ne dobije izraz koji je dobro usaglašen sa podacima.

Najpoznatija funkcija u pythonu kojom se pronalaze optimalne vrednosti parametara koji figurišu u funkciji za fitovanje je `curve_fit()`, koja je deo poznate biblioteke **scipy**. Funkcija `curve_fit()` ima tri osnovna argumenta: funkcija kojom se fituje, nezavisna promenljiva (podaci za *x*-osu) i zavisno promenljiva (podaci za *y*-osu). Dakle, da bi se koristila funkcija `curve_fit()`, neophodno je prvo definisati funkciju, odnosno model, za koji se

smatra da bi bio najbolji fit posmatranih podataka. Vrednosti parametara koje figurišu u fit funkciji na početku optimizacije uzimaju izvesne početne vrednosti parametara.

U slučaju jednostavnijih funkcija, `curve_fit()` će bez problema izračunati optimalne vrednosti parametara koji figurišu u fit funkciji bez potrebe za bilo kakvim intervencijama od strane korisnika. Međutim, u slučaju složenijih funkcija, može doći do problema. U tom smislu, bitno je naglasiti da `curve_fit()` funkcija sprovodi proces optimizacije. Između ostalog, efikasnost optimizacije zavisi od toga koliko je „sistem“ daleko od ravnotežnog stanja. Što je „sistem“ bliži ravnotežnom stanju, tj. što su početne vrednosti parametara koji figurišu u modelu bliže optimalnim vrednostima, optimizacija će biti efikasnija. Tako je i sa `curve_fit()` funkcijom. S tim u vezi, kada dođe do problema sa efikasnošću `curve_fit()` funkcije, najverovatnije se radi o lošim početnim vrednostima parametara koji figurišu u modelu i to je svakako prvo mesto u kome treba tražiti uzrok. Ukoliko su početne vrednosti parametara koji figurišu u fit modelu daleko od optimalnih vrednosti, efikasnost `curve_fit()` funkcije će biti loša.

U toj situaciji, mogu se preduzeti dva koraka. Prvi korak je da se definiše malo veća vrednost argumenta funkcije `curve_fit()` koji reguliše koliko puta se sprovodi proračun. Ovaj argument je `maxfev=n`, gde je `n` ceo broj, međutim ovaj pristup retko daje željeni rezultat. Drugi korak, daleko preporučljiviji, je da se ne uzimaju podrazumevane vrednosti parametara, nego da se konkretno definišu početne vrednosti. Ovo se postiže tako što se u funkciji `curve_fit()` u obliku liste definišu vrednosti argumenta `p0`. Ovakav slučaj će biti detaljno razrađen u odeljku 8.4. koji se odnosi na fitovanje podataka u vezi sa RL kolom.

Naravno, odmah se postavlja pitanje kako pravilno izabrati početne vrednosti parametara fit funkcije. Za najčešće korišćene funkcije za fitovanje, detaljna matematička analiza je pokazala za koje vrednosti parametara će biti reprodukovan izgled krive koji najviše odgovara sakupljenim podacima, pa se upravo oni predlažu za početne vrednosti ukoliko dođe do problema sa efikasnošću `curve_fit()` funkcije. U ovom delu biće navedene linije kôda za upotrebu `curve_fit()` funkcije za dva slučaja. Prvi slučaj je kada nije neophodno definisati početne vrednosti

```
params, covs = scipy.optimize.curve_fit(objective, x, y)
```

odnosno, kada je neophodno definisati konkretne početne vrednosti

```
params, covs = scipy.optimize.curve_fit(objective, t, i,p0=guess)
```

gde je `guess` lista sa definisanim početnim vrednostima parametara koji figurišu u fit modelu. Argument `p0` je mogao da se definiše npr. i na sledeći način: `p0=[-1,-0.5,1]`. Naravno, ta lista sadržava onoliko elemenata koliko ima parametara u fit modelu.

Važno je istaći i da `curve_fit()` funkcija kao izlaz daje dva objekta (otuda dva parametra sa leve strane jednakosti, mada se mogao definisati i samo jedan objekat, koji bi onda bio niz sa dva elementa, od kojih je svaki element opet niz). Prvi objekat, koji se u literaturi često označava sa `params`, je niz koji sadrži vrednosti parametara koji figurišu u fit funkciji. Drugi objekat je matrica kovarijansi, koja se u literaturi često označava sa `covs`. U ovoj matrici, dijagonalni elementi su vrednosti varijanse izračunatih parametara fit funkcije.

Veće vrednosti označavaju veće nesigurnosti, dok manje vrednosti označavaju manje nesigurnosti, tako da je poželjno da su dijagonalni elementi što manji. Bočni elementi ove matrice su vrednosti kovarijansi između parova parametara, koji ukazuju na to kako promena jednog parametra utiče na procenu drugog parametra. Pozitivne vrednosti ovih elemenata označavaju pozitivnu linearnu zavisnost, dok negativne vrednosti označavaju negativnu linearnu zavisnost. Konkretni primeri primene će biti dati u sledećem poglavlju.

8.2.2. Funkcija `polyfit()`

Ukoliko se podaci fituju polinomnim funkcijama, povoljno je koristiti funkciju `polyfit()` koja je deo biblioteke **numpy**. Za razliku od funkcije `curve_fit()` nije neophodno prethodno posebno definisati model kojim se fituju podaci, nego je neophodno definisati samo stepen polinoma kojim se fituju podaci. U tom smislu, osnovni argumenti funkcije `polyfit()` su podaci za nezavisnu promenljivu (tj. podaci za *x*-osu), podaci za zavisnu promenljivu (tj. podaci za *y*-osu), i na kraju stepen polinoma. Primer linije kôda kojom se traži određivanje koeficijenata polinoma drugog stepena za fitovanje podataka je sledeći

```
fit = np.polyfit(x, y, 2)
```

Izlaz funkcije `polyfit()` će biti niz koji će sadržavati vrednosti koeficijenata polinoma, od najvišeg do najnižeg koji fituje podatke. Naravno, broj elemenata tog niza će uvek biti za jedan veći od stepena polinoma.

8.2.3. Procena validnosti fita

Procena validnosti fita je od velikog značaja, posebno kada različite funkcije daju naizgled jednako dobro slaganje sa sakupljenim podacima. U obradi podataka, često se koristi tzv. koeficijent korelacije, tzv. r^2, parametar kako bi se procenilo kolika je korelacija između podataka. Najčešće se vrednost ovog koeficijenta izražava u procentima i što veće vrednosti ovog parametra označavaju veću korelaciju između veličina. S obzirom da se u ovom poglavlju radi o fitovanju podataka, jasno je da je neophodno da koeficijent korelacije između stvarnih podataka i njihovog fita bude što veći.

U pythonu koeficijent korelacije se može izračunati na više načina, a ovde će biti predstavljena dva načina. Jedan način je upotrebom `corrcoef()` funkcije iz **numpy** biblioteke, dok je drugi način upotrebom funkcije `r2_score()` iz biblioteke **sklearn**. Odgovarajuća linija kôda za određivanje koeficijenta korelacije preko **numpy** biblioteke je

```
p_coeff = np.corrcoef(podaci, fitovano)
```

Data linija kôda će zapravo dati *Pirsonov* koeficijent korelacije, dok se koeficijent korelacije dobija njegovim kvadriranjem. Linija kôda za određivanje koeficijenta korelacije preko biblioteke **sklearn** je

```
r_2 = r2_score(podaci, fitovano)
```

U oba slučaja, struktura funkcije je ista. Ona je naime argument dva parametra, a to su redom stvarni podaci i podaci dobijeni fitovanom funkcijom.

8.3. Praktični primer fitovanja - određivanje gravitacione konstante matematičkim klatnom

Znajući da je period oscilovanja matematičkog klatna dat sledećom funkcijom

$$T = 2\pi\sqrt{\frac{l}{g}}\ . \tag{8.1}$$

gravitaciona konstanta se može izračunati merenjem perioda oscilovanja u zavisnosti od dužine matematičkog klatna. Ukoliko se izraz za period oscilovanja matematičkog klatna kvadrira, dobija se sledeći izraz koji povezuje period oscilovanja i dužinu matematičkog klatna

$$T^2 = \frac{4\pi^2}{g} \cdot l \,. \tag{8.2.}$$

Ukoliko se nacrta grafik zavisnosti $T^2 = f(l)$ dobija se linearni grafik sa koga se lako može izračunati nagib. Funkcija prave linije je

$$y = ax + b \ , \tag{8.3.}$$

gde je a nagib, a b odsečak na *y*-osi. Pošto odsečka na *y*-osi u ovom slučaju nema, izraz se svodi na

$$y = ax \,. \tag{8.4.}$$

Poređenjem izraza 7.2. i 7.4. vidi se da važi sledeća relacija

$$a = \frac{4\pi^2}{g} \ . \tag{8.5}$$

Odakle se gravitaciona konstanta može izračunati preko nagiba prave na sledeći način

$$g = \frac{4\pi^2}{a} \ . \tag{8.6}$$

Dakle, vrednost gravitacione konstante se može izračunati tako što se podaci na grafiku $T^2 = f(l)$ najpre fituju linearnom funkcijom, nakon čega treba odrediti nagib koji se koristi u poslednjoj relaciji. Stoga je sledeći korak demonstrirati kako se pomenuti podaci mogu fitovati linearnim fitom.

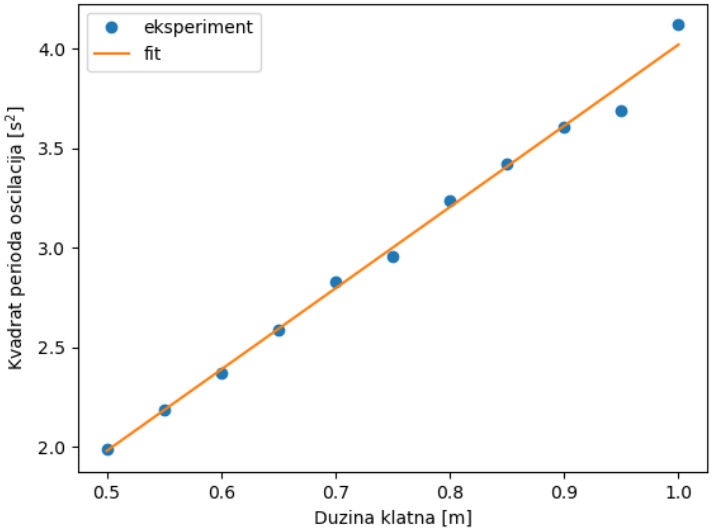
8.3.1. Fitovanje podataka kod eksperimenta sa matematičkim klatnom funkcijom `curve_fit()`

U konkretnoj eksperimentalnoj vežbi, period oscilacija se meri za različite vrednosti dužine matematičkog klatna. Podaci iz ove vežbe su dati u fajlu „g_const_2.csv“ koji je korišćen za vizuelizaciju u prethodnom poglavlju. Linearni fit će biti demonstriran primenom funkcije `curve_fit()` iz scipy biblioteke, kao i primenom funkcije `polyfit()` iz biblioteke **numpy**. Program za fitovanje podataka iz fajla „g_const_2.csv“ upotrebom `curve_fit()` funkcije je dat u sledećem primeru.

Primer 8.1. Linearni fit $T^2 = f(l)$ upotrebom `curve_fit()`

```
1.  import numpy as np
2.  import pandas as pd
3.  import matplotlib.pyplot as plt
4.  from scipy.optimize import curve_fit
5.
6.  df = pd.read_csv('g_const_2.csv')
7.
8.  l_df = df['duzina']
9.  l = l_df.to_numpy()
10. Tsq_df = df['period^2']
11. Tsq = Tsq_df.to_numpy()
12.
13. def model(x,a,b):
14.     return a*x + b
15.
16. params, covs = curve_fit(model, l, Tsq)
17.
18. a = round(params[0],4)
19. b = round(params[1],4)
20. print('a =',a,'b =',b)
21.
22. g = (4*np.pi**2)/a
23. print('Vrednost gravitacione konstante je: ',round(g,2),'m/s^2')
24.
25. plt.plot(l,Tsq,'o',label='eksperiment')
26. plt.plot(l,model(l,params[0],params[1]),label='fit')
27. plt.xlabel('Duzina klatna [m]')
28. plt.ylabel('Kvadrat perioda oscilacija [$\mathrm{s}^2$]')
29. plt.legend()
30. plt.show()
>>a = 4.08 b = -0.0591
>>Vrednost gravitacione konstante je:  9.68 m/s^2
```

Grafik koji se dobija pokretanjem programa iz poslednjeg primera je dat na slici 8.6.



Slika 8.6. Linearni fit dobijen programom iz primera 8.1

Prvih 11 linija kôda programa u primeru 8.1. su jasne iz prethodnih poglavlja i neće biti posebno komentarisane, osim dve napomene. Prvo, u liniji 4 je učitana biblioteka **scipy** zajedno sa modulom **optimize**. Druga napomena je više podsećanje da fajl „g_const_2.csv“ sadrži tri kolone podataka koje su redom: dužina matematičkog klatna, period oscilovanja i kvadrat perioda oscilovanja. Za crtanje grafika, neophodne su prva i treća kolona čiji su nazivi u .csv fajlu, redom, „duzina“ i „period^2“.

U linijama 13 i 14 definisana je funkcija koja je model prema kome će se fitovati podaci iz fajla „g_const_2.csv“. Linija 16 je praktično glavna linija u ovom programu, jer daje instrukciju da se iskoristi funkcija `curve_fit()` za pronalaženje optimalnih vrednosti parametara *a* i *b*, na osnovu podataka o dužini klatna (označeno kao `l`) i kvadrata perioda oscilacija (označeno kao `T_sq`). Kao što je već naglašeno, funkcija `curve_fit()` generiše kao izlaz dva niza, pa je stoga sa desne strane dva objekta. Prvom objektu, `params`, dodeljuju se vrednosti *a* i *b* kao elementi niza. U linijama 18 i 19 vrednosti ovih parametara se izdvajaju u posebne objekte, dok se u liniji 20 traži njihovo štampanje. Dalje se u liniji 22 računa vrednost gravitacione konstante preko izraza 8.6. dok se u liniji 23 traži štampanje vrednosti iste. U linijama 25 i 26 daju se instrukcije za crtanje eksperimentalnih tačaka i fit funkcije, respektivno. Napomena je da su argumenti funkcije `model()` u linijama za crtanje optimizovane vrednosti parametara *a* i *b*. Ovim programom se dobija vrednost gravitacione konstante u iznosu od 9,68 $\mathrm{m/s^2}$.

Važno je naglasiti da se nadalje funkcija `model()` sa optimizovanim vrednostima parametara `a` i `b` može koristiti za dobijanje vrednosti kvadrata perioda oscilacija za vrednosti dužine klatna za koje nema eksperimentalnog podatka. Na primer, za vrednost dužine klatna od 0,95 m nema eksperimentalnog podatka o kvadratu perioda oscilovanja. Jednostavnim unošenjem proizvoljne vrednosti dužine klatna, kao prvi argument funkcije `model()`, dobija se vrednost funkcije od 3,8985 $\mathrm{s^2}$. Konkretno, linija kôda u tom slučaju bi bila

```
model(0.97,a,b)
```

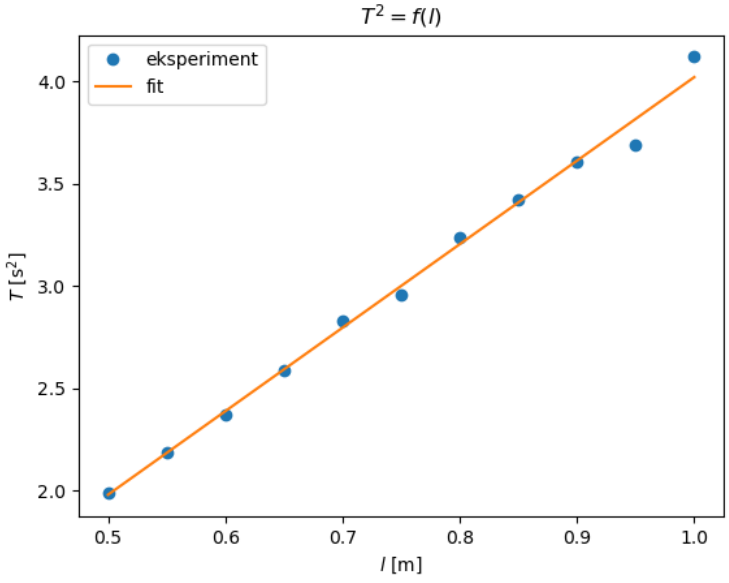
8.3.2. Fitovanje podataka eksperimenta sa matematičkim klatnom funkcijom polyfit()

U slučaju linearnog fita, python program može biti nešto jednostavniji ukoliko se koristi funkcija `polyfit()`. Argumenti ove funkcije su vrednosti sa *x*- i *y*-ose, dok je *n* stepen polinoma kojim se fituju podaci. Naravno, za linearni fit vrednost argumenta *n* će biti 1.

Ponovo je cilj crtanje grafika $T^2 = f(l)$, upotrebom podataka iz fajla „g_const_2.csv“, fitovanje podataka linearnim fitom i računanje vrednosti gravitacione konstante prema izrazu 8.6. Primer linearnog fitovanja upotrebom funkcije `polyfit()` je dat u sledećem primeru, a prateći grafik je dat na slici 8.7.

Primer 8.2. Primena linearnog fitovanja upotrebom funkcije `polyfit()`

```
1.  import pandas as pd
2.  import numpy as np
3.  import matplotlib.pyplot as plt
4.
5.  df = pd.read_csv('g_const_2.csv')
6.
7.  l_df = df['duzina']
8.  l = l_df.to_numpy()
9.  T_sq_df = df['period^2']
10. T_sq = T_sq_df.to_numpy()
11.
12. a, b = np.polyfit(l, T_sq, 1)
13. print("Nagib: a = ",round(a,2),", Odsecak: b = ",round(b,2))
14.
15. g = (4*(np.pi**2))/a
16. print("Vrednost gravitacione konstante je: ",round(g,2), " m/s^2")
17.
18. plt.plot(l,T_sq,'o',label='eksperiment')
19. plt.plot(l,a*l+b,label='fit')
20. plt.title('$T^2=f(l)$')
21. plt.xlabel('$l$ [m]')
22. plt.ylabel('$T$ [$\mathrm{s^2}$]')
23. plt.legend()
24. plt.show()
 >>
Nagib: a =  4.08 , Odsecak: b =  -0.06
Vrednost gravitacione konstante je:  9.68  m/s^2
```



Slika 8.7. Grafik koji se dobija kao izlaz programa iz primera 8.2.

U poslednjem primeru, u liniji 5 su učitani podaci iz fajla „g_const_2.csv“, a zatim su u linijama 7-10 podaci iz kolona koje sadrže podatke o dužini matematičkog klatna i kvadratu oscilacija, respektivno, pretvoreni u nizove. Konkretno, u linijama 7 i 9 su podaci za *x*- i *y*-osu respektivno. Ovi podaci će biti iskorišćeni kao osnova za **numpy** funkciju `polyfit()`, kojom će se dobiti vrednosti parametara za linearnu funkciju kojom će se fitovati podaci. `polyfit()` funkcija je iskorišćena u liniji 12. U slučaju linearnog fita, pomenuta funkcija kao rezultat daje dve vrednosti (nagib i odsečak, respektivno). Zbog toga su sa leve strane jednakosti u liniji 12 dva objekta (`a` i `b`). Da je sa leve strane jednakosti u liniji 12 stavljen samo jedan objekat, onda bi rezultat bio niz koji sadrži dve vrednosti (nagib i odsečak). U liniji 13 se traži štampanje vrednosti odsečka i nagiba, zaokruženih na dve decimale upotrebom funkcije `round()`, u liniji 15 se definiše izraz 8.6. za računanje vrednosti gravitacione konstante, dok se u liniji 16 traži štampanje izračunate vrednosti gravitacione konstante (opet zaokruženo na dve decimale). Konačno, u linijama 18 – 24 su instrukcije za crtanje grafika na kome su predstavljene eksperimentalne tačke i linearni fit.

U slučaju da korisnik želi da dođe do informacije o kvadratu perioda oscilovanja za dužinu matematičkog klatna za vrednost koja nije data u podacima, npr. od 0,97 m, u okviru **numpy** biblioteke postoji funkcija `poly1d()` kojom se to može postići. Pomenuta funkcija služi za konstruisanje polinoma na osnovu vrednosti koeficijenata, nakon čega se isti može iskoristiti za dobijanje vrednosti *y* za proizvoljnu vrednost promenljive *x*. U opštem slučaju, argument funkcije `poly1d()` je lista čije vrednosti predstavljaju koeficijente polinoma. Na primer, linija koda `np.poly1d([1.87,-0.76])` će generisati sledeći polinom

$$y = 1{,}87 \cdot x - 0{,}76\,, \tag{8.1}$$

dok će linija koda `np.poly1d([1.45,1.52,-0.55])` generisati sledeći polinom

$$y = 1{,}45 \cdot x^2 + 1{,}52 \cdot x - 0{,}55\,. \tag{8.2}$$

Generisanje polinoma na osnovu koeficijenta dobijenih funkcijom `polyfit()` je demonstrirano u sledeća dva primera:

Primer 8.3. Generisanje polinoma – verzija 1

```
1. import pandas as pd
2. import matplotlib.pyplot as plt
3. import numpy as np
4.
5. df = pd.read_csv('g_const_2.csv')
6.
7. x_df = df['duzina']
8. x = x_df.to_numpy()
9. y_df = df['period^2']
10. y = y_df.to_numpy()
11.
12. a, b = np.polyfit(x, y, 1)
13.
14. polinom = np.poly1d([a,b])
15. print(polinom)
>>
4.08 x - 0.05909
```

Primer 8.4. Generisanje polinoma – verzija 2

```
1. import pandas as pd
2. import matplotlib.pyplot as plt
3. import numpy as np
4.
5. df = pd.read_csv('g_const_2.csv')
6.
7. x_df = df['duzina']
8. x = x_df.to_numpy()
9. y_df = df['period^2']
10. y = y_df.to_numpy()
11.
12. koeficijenti = np.polyfit(x, y, 1)
13.
14. polinom = np.poly1d(koeficijenti)
15. print(polinom)
>>
4.08 x - 0.05909
```

Najpre treba naglasiti da će programi iz poslednja dva primera dati identičan izlaz, a jedine razlike su u linijama 12 i 14. Naime, određene funkcije pythona kao izlaz mogu da daju više vrednosti. U tim slučajevima, te izlazne vrednosti se mogu dodeliti jednom objektu ili većem broju objekata. Ako se izlazne vrednosti dodeljuju jednom objektu, onda će taj objekat biti niz koji će imati onoliko članova koliko posmatrana funkcija daje izlaza (u ovom konkretnom slučaju niz će imati dva elementa). Sa druge strane, ako se sa leve strane jednakosti stavi onoliko objekata koliko posmatrana funkcija daje vrednosti kao izlaz, onda će svakom od tih objekata sa leve strane jednakosti biti sukcesivno dodeljivana vrednost svakog od izlaza posmatrane funkcije. Ovo je malo teže objasniti tekstom, pa je zato odabrano da se to konkretno demonstrira pomoću poslednja dva primera.

U primeru određivanja gravitacione konstante pomoću matematičkog klatna funkcija `polyfit()` daje kao izlaz dva broja. U liniji 12 primera 8.3. stoji

```
a, b = np.polyfit(x, y, 1)
```

Pošto funkcija `polyfit(x,y,1)` kao izlaz daje dve vrednosti, a sa leve strane jednakosti stoje dva objekta (`a` i `b`), izlazne vrednosti funkcije `polyfit(x,y,1)` će sukcesivno biti dodeljivane promenljivim `a` i `b` (funkcija `polyfit()` je tako napravljena da kao izlazne vrednosti redom daje koeficijente polinoma počevši od najvišeg stepena). U skladu sa tim, u liniji 14 primera 8.2., argument funkcije `poly1d()` je lista koja sadrži vrednosti *a* i *b*.

Sa druge strane, u liniji 12 primera 8.3. stoji

```
koeficijenti = np.polyfit(x, y, 1)
```

Pošto sa leve strane jednakosti stoji samo jedan objekat, izlazne vrednosti funkcije `polyfit(x,y,1)` će biti dodeljene objektu `koeficijenti` u formi niza sa dva člana čije su vrednosti dobijeni koeficijenti. Sada je u liniji 14 primera 8.4. za argument funkcije `poly1d()` dovoljno upisati samo naziv objekta `koeficijenti` koji sadrži utvrđene koeficijente u obliku niza sa dva člana.

Kada se na osnovu izračunatih koeficijenata formira odgovarajući polinom koji najbliže opisuje eksperimentalne rezultate, lako je izračunati vrednost tog polinoma za proizvoljnu vrednost promenljive. Naime, dovoljno je samo pozvati novoformiranu funkciju `polinom()` i kao argument joj zadati željenu vrednost. U slučaju primera koji je predstavljen, ako bi korisnik želeo da izračuna vrednost kvadrata perioda oscilacija za dužinu od 0,97 m, za koju ne postoji eksperimentalni podatak, dovoljno bi bilo napisati liniju kôda

```
polinom(0.97)
```

što bi u konkretnom slučaju kao rezultat dalo kao izlaz: 3.8985090909090934. Dakle, za posmatrano matematičko klatno, tj. za njegovu dužinu od 0,97 m kvadrat perioda oscilovanja bi iznosio 3,90 s^2.

U predstavljenom primeru očigledno postoji linearna zavisnost između kvadrata perioda oscilovanja i dužine matematičkog klatna. Međutim, ukoliko bi bilo neophodno fitovati kvadratnom funkcijom, onda bi linija 12 u npr. primeru 8.4. glasila

```
koeficijenti = np.polyfit(x, y, 2)
```

Naravno, u ovom slučaju funkcija `polyfit()` bi generisala tri koeficijenta, pa bi objekat koeficijenti bio niz od tri člana.

8.3.3. Provera validnosti fita kod eksperimenta sa matematičkim klatnom

Iako je sa slika 8.1. i 8.2. jasno da dobijeni linearni fit adekvatno opisuje podatke sa grafika $T^2 = f(l)$, u sledećem primeru biće predstavljen program kojim se pored fitovanja računa i koeficijent korelacije između eksperimentalnih podataka i fitovane funkcije.

Primer 8.5. Provera validnosti fitovanja u slučaju primera 8.1.

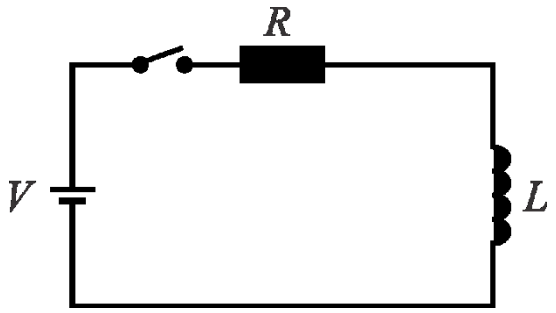
```
1.  import numpy as np
2.  import pandas as pd
3.  import matplotlib.pyplot as plt
4.  from scipy.optimize import curve_fit
5.  from sklearn.metrics import r2_score
6.
7.  df = pd.read_csv('g_const_2.csv')
8.
9.  l_df = df['duzina']
10. l = l_df.to_numpy()
11. Tsq_df = df['period^2']
12. Tsq = Tsq_df.to_numpy()
13.
14. def model(x,a,b):
15.     return a*x + b
16.
17. params, covs = curve_fit(model, l, Tsq)
18.
19. a = round(params[0],4)
20. b = round(params[1],4)
21. print('a =',a,'b =',b)
22.
23. g = (4*np.pi**2)/a
24. print('Vrednost gravitacione konstante je: ',round(g,2),'m/s^2')
25.
26. r_sq = r2_score(Tsq, model(l,a,b))
27. print('Koeficijent korelacije iznosi : ',r_sq*100)
28.
29. plt.plot(l,Tsq,'o',label='eksperiment')
30. plt.plot(l,model(l,params[0],params[1]),label='fit')
31. plt.xlabel('Duzina klatna [m]')
32. plt.ylabel('Kvadrat perioda oscilacija [$\mathrm{s}^2$]')
33. plt.legend()
34. plt.show()
>> a = 4.08 b = -0.0591
>> Vrednost gravitacione konstante je:  9.68 m/s^2
>> Koeficijent korelacije iznosi :  99.33747867479957
```

U poslednjem programu u liniji 26 se koristi funkcija `r2_score()` za računanje koeficijenta korelacije, dok se štampanje istog traži u liniji 27. Naravno, pre toga je u liniji 5

učitana biblioteka **sklearn** zajedno sa modulom **metrics**. Vidi se da se dobija vrlo visoka vrednost koeficijenta korelacije koji iznosi preko 99%.

8.4. Praktični primer fitovanja - određivanje koeficijenta samoinduktivnosti RL kola

Fitovanje linearnom funkcijom se često koristi, međutim zavisnost među veličinama ne mora biti linearna, pa će u ovom poglavlju biti predstavljeno kako se uz pomoć pythona podaci mogu fitovati nekom od funkcija navedenih u odeljku 8.1. Konkretno, razmotriće se RL kolo i biće demostrirano kako se može odrediti koeficijent samoinduktivnosti RL kola fitovanjem eksperimentalnih podataka $i = i(t)$. Pre nego što se pređe na konkretan program, biće ukratko predstavljeno RL kolo, čija je šema data na slici 8.8.



Slika 8.8. Šema RL kola

RL kolo je jedno od fundamentalnih električnih kola koje se sastoji od otpornika otpora R i kalema koeficijenta samoinduktivnosti L. Primenom Kirhofovih pravila dolazi se do sledeće diferencijalne jednačine koja opisuje RL kolo

$$\frac{di}{dt} = \frac{1}{L}(V - i \cdot R)\,. \tag{8.3}$$

Egzaktno rešenje ove diferencijalne jednačine je

$$i = \frac{V}{R}\left(1 - \mathrm{e}^{-\frac{R}{L}t}\right)\,. \tag{8.4}$$

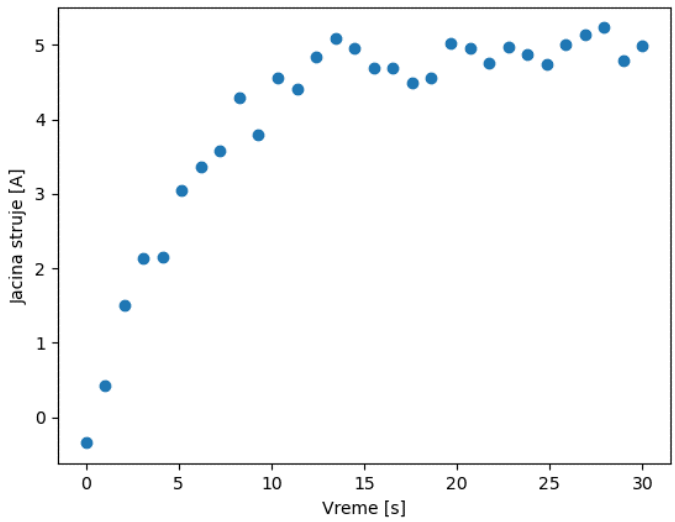
U primeru koji sledi, analiziraće se podaci dobijeni eksperimentalnim merenjem jačine struje od vremena kod RL kola kod koga je vrednost otpora $R = 1\ \Omega$, napona $V = 5\ \mathrm{V}$, dok je vrednost koeficijenta samoinduktivnosti kalema nepoznata. Neka je u eksperimentalnoj vežbi nakon zatvaranja prekidača tokom vremena od 30 s na svaku sekundu merena vrednost jačine

struje u RL kolu i neka su podaci dati u .csv fajlu pod nazivom „RL_kolo.csv“. Pomenuti fajl sadrži dve kolone sa vrednostima, gde je naziv prve kolone „t“, dok je naziv druge kolone „i“. Za početak, nacrtaće se grafik sa dobijenim vrednostima, kako bi se izveli preliminarni zaključci. Program za dobijanje grafika iz podataka u fajlu „RL_kolo.csv“ je dat u sledećem primeru.

Primer 8.6. Crtanje grafika sa vrednostima za RL kolo

```
1.  import pandas as pd
2.  import matplotlib.pyplot as plt
3.
4.  df = pd.read_csv('RL_kolo.csv')
5.
6.  t_df = df['t']
7.  t = t_df.to_numpy()
8.  i_df = df['i']
9.  i = i_df.to_numpy()
10.
11. plt.plot(t,i,'o')
12. plt.xlabel("Vreme [s]")
13. plt.ylabel("Jacina struje [A]")
14. plt.show()
```

Rezultat programa iz primera 8.6. je grafik predstavljen na slici 8.9.



Slika 8.9. Eksperimentalni podaci $i = i(t)$ kod RL kola (rezultat programa iz primera 8.6)

Grafik predstavljen na slici 8.9. ukazuje da jačina struje naglo raste u tokom prvih 5 s. Rast jačine struje je i dalje intenzivan do otprilike 10. sekunde, nakon čega se posle nekoliko sekundi dostiže maksimalna vrednost i jačina struje saturira.

Predstavljeni grafik $i = i(t)$ je tipičan za RL kolo, a ako se analiziraju najčešće matematičke funkcije kojima se fituju podaci iz poglavlja 8.1, vidi se da eksperimentalni podaci iz fajla „RL_kolo.csv“ mogu da se fituju praktično sa dve matematičke funkcije – funkcijom za eksponencijalni rast sa limitom, $y = ae^{bx} + c$, kao i “plato” funkcijom, $y = ax/(b + x)$. Eksperimentalni podaci iz pomenutog fajla će biti fitovani pomoću obe ove funkcije, nakon čega će se njihove performanse uporediti računanjem koeficijenta korelacije.

8.4.1. Fitovanje podataka RL kola funkcijom eksponencijalnog rasta sa limitom

Program za fitovanje eksperimentalnih podataka RL kola upotrebom `curve_fit()` funkcije jednačinom eksponencijalnog rasta sa limitom (8.4) je dat u sledećem primeru.

Primer 8.7. Fitovanje podataka za RL kolo

```
1.  import pandas as pd
2.  import matplotlib.pyplot as plt
3.  import numpy as np
4.  from scipy.optimize import curve_fit
5.
6.  df = pd.read_csv('RL_kolo.csv')
7.
8.  t_df = df['t']
9.  t = t_df.to_numpy()
10. i_df = df['i']
11. i = i_df.to_numpy()
12.
13. def model(t,a,b,c):
14.     return a*np.exp(b*t) + c
15.
16. params, covs = curve_fit(model, t, i)
17.
18. a = params[0]
19. b = params[1]
20. c = params[2]
21.
22. print('Parametara a, b i c su:', params)
23.
24. i_fitted = model(t,a,b,c)
25.
26. plt.plot(t,i,'o')
27. plt.plot(t,i_fitted)
28. plt.xlabel("Vreme [s]")
29. plt.ylabel("Jacina struje [A]")
30. plt.show()
>> Optimal parameters not found: Number of calls to function has reached maxfev =
800.
```

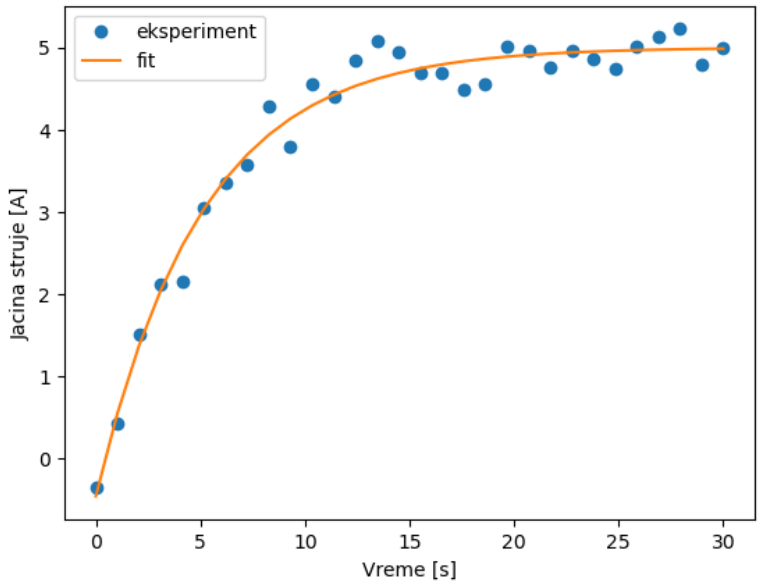
Vidi se da će ovaj program prijaviti grešku da čak ni posle 800 iteracija nisu pronađene optimalne vrednosti parametara koji figurišu u modelu definisanom u linijama 13 i 14. Kao što

je naglašeno u odeljku 8.2.1. jedna od mogućnosti jeste da se poveća vrednost argumenta `maxfev`, međutim povećanje vrednosti ovog argumenta na veliku vrednost će dovesti do toga da se dobije pogrešan fit. Konkretno, dobiće se linearna zavisnost, što uopšte ne odgovara eksperimentalnim podacima. Drugo, pravo, rešenje jeste da se definišu adekvatne početne vrednosti. Podsećanjem na funkciju (8.4.) u odeljku 8.1, vidi se da se kriva sa saturacijom (platoom) dobro reprodukuje funkcijom (8.4.) ako su vrednosti a, b i c redom: -1, -0,5 i 1. Imajući ovo u vidu, u sledećem primeru se predstavlja program koji je vrlo sličan onom iz prethodnog primera, ali u kome se u funkciji `curve_fit()` pojavljuje i argument `p0` sa definisanim početnim vrednostima koje su navedene.

Primer 8.8. Fitovanje podataka za RL kolo sa definisanim početnim vrednostima

```
1.  import pandas as pd
2.  import matplotlib.pyplot as plt
3.  import numpy as np
4.  from scipy.optimize import curve_fit
5.
6.  df = pd.read_csv('RL_kolo.csv')
7.
8.  t_df = df['t']
9.  t = t_df.to_numpy()
10. i_df = df['i']
11. i = i_df.to_numpy()
12.
13. def model(t,a,b,c):
14.     return a*np.exp(b*t) + c
15.
16. guess=[-1,-0.5,1]
17. params, covs = curve_fit(model, t, i,p0=guess)
18.
19. a = params[0]
20. b = params[1]
21. c = params[2]
22.
23. print('Parametri a, b i c su:', params)
24.
25. i_fitted = model(t,a,b,c)
26.
27. plt.plot(t,i,'o',label='eksperiment')
28. plt.plot(t,i_fitted,label='fit')
29. plt.xlabel("Vreme [s]")
30. plt.ylabel("Jacina struje [A]")
31. plt.legend()
32. plt.show()
>> Parametri a, b i c su: [-5.46333386 -0.19793854  5.00428209]
```

Kao izlaz, program iz poslednjeg primera će dati i grafik predstavljen na slici 8.10.



Slika 8.10. Fitovanje podataka RL kola

Dakle, program u primeru 8.8. se razlikuje od prethodnog programa po tome što se u liniji 16 definisala lista `guess` koja sadrži vrednosti koje će biti početne za parametre a, b i c koji figurišu u modelu za fitovanje. Takođe, u liniji 17 je u okviru funkcije `curve_fit()` morao da se definiše i argument za početne vrednosti, `p0`, koji ima vrednost upravo `guess`. Program je mogao da se napiše i bez linije 16, ali da argument za početne vrednosti u okviru `curve_fit()` funkcije bude `p0=[-1,-0.5,1]` . Vidi se da ovakav program daje odličan fit, a preostaje da se izračuna koeficijent korelacije između eksperimentalnih vrednosti i fita, što će biti urađeno u jednom od sledećih odeljaka.

8.4.2. Fitovanje podataka RL kola "plato" funkcijom

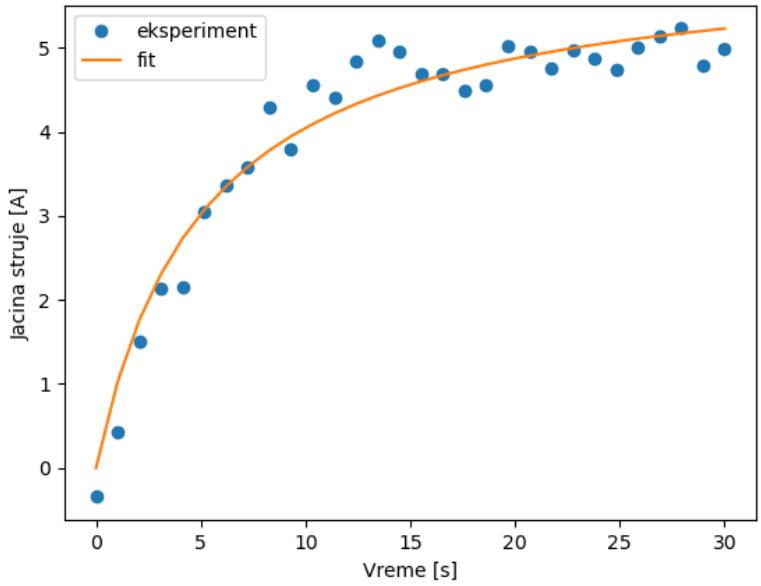
Program za fitovanje eksperimentalnih podataka RL kola upotrebom `curve_fit()` funkcije i modelom "plato" funkcije (8.5) je dat u sledećem primeru.

Primer 8.9. Fitovanje podataka RL kola "plato" funkcijom

```
1.  import pandas as pd
2.  import matplotlib.pyplot as plt
3.  import numpy as np
4.  from scipy.optimize import curve_fit
5.
6.  df = pd.read_csv('RL_kolo.csv')
7.
8.  t_df = df['t']
9.  t = t_df.to_numpy()
10. i_df = df['i']
```

```
11. i = i_df.to_numpy()
12.
13. def model(t,a,b):
14.     return (a*t)/(b+t)
15.
16. params, covs = curve_fit(model, t, i)
17.
18. print('Parametri a i b su:', params)
19.
20. a = params[0]
21. b = params[1]
22.
23. i_fitted = model(t,a,b)
24.
25. plt.plot(t,i,'o',label='eksperiment')
26. plt.plot(t,i_fitted,label='fit')
27. plt.xlabel("Vreme [s]")
28. plt.ylabel("Jacina struje [A]")
29. plt.legend()
30. plt.show()
>> Parametri a i b su: [6.12110283 5.13022823]
```

Grafik koji se dobija kao izlaz poslednjeg programa je dat na slici 8.11.



Slika 8.11. Fitovanje RL kola „plato“ funkcijom

Rezultat predstavljen na slici 8.11. pokazuje odlično slaganje eksperimentalnih podataka i fitovane funkcije. Pošto je konstrukcija kôda identična kao i u prethodnim primerima, nema potrebe za detaljnim objašnjenjima. Zanimljivo je, međutim, da u ovom slučaju nije bilo neophodno definisati početne vrednosti fit parametara za `curve_fit()` funkciju. Kako su podrazumevane početne vrednosti za fit parametre 1, a analiza „plato“ funkcije pokazuje da se

plato odlično reprodukuje upravo za vrednosti parametara $a = 1$ i $b = 1$ u (8.5.), jasno je da su podrazumevane vrednosti odličan izbor kao početne vrednosti.

8.4.3. Procena validnosti fitova podataka kod RL kola

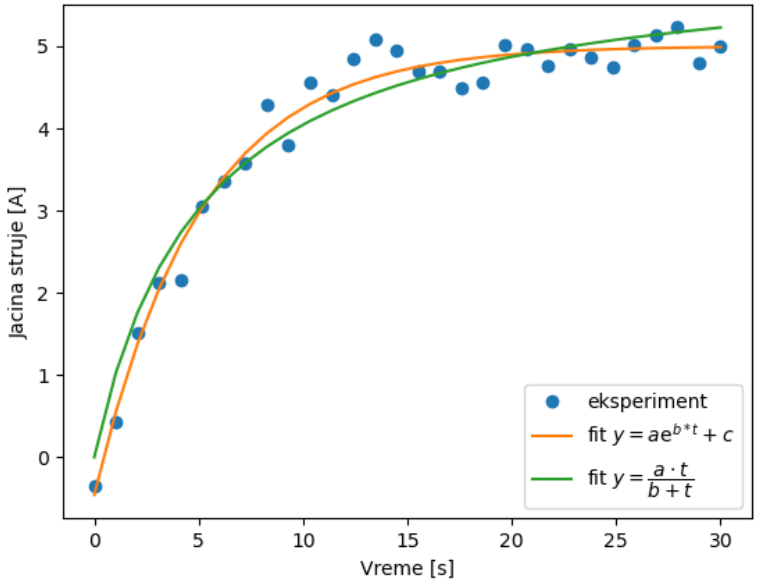
U poslednjem delu fitovanja podataka u vezi RL kola, sprovešće se procena validnosti fitova dobijenih pomoću dva modela. Ako se uporede slike 8.10. i 8.11. vidi se da fitovanje eksperimentalnih podataka upotrebom funkcija (8.4) i (8.5) daje odlične rezultate. Fitovane krive se odlično uklapaju među eksperimentalne podatke, a još lakše bi bilo uporediti ove slike ako bi se napisao program kojim bi se fitovali podaci sa obe funkcije i kojim bi se na jednom grafiku predstavili eksperimentalni rezultati i obe fitovane krive. Takav program je dat u sledećem primeru.

Primer 8.10. Fitovanje podataka RL kola sa obe funkcije i grafičko poređenje

```
1.  import pandas as pd
2.  import matplotlib.pyplot as plt
3.  import numpy as np
4.  from scipy.optimize import curve_fit
5.
6.  df = pd.read_csv('RL_kolo.csv')
7.
8.  t_df = df['t']
9.  t = t_df.to_numpy()
10. i_df = df['i']
11. i = i_df.to_numpy()
12.
13. def model1(t,a,b,c):
14.     return a*np.exp(b*t) + c
15.
16. def model2(t,a,b):
17.     return (a*t)/(b+t)
18.
19. guess=[-1,-0.5,1]
20. params1, covs1 = curve_fit(model1, t, i,p0=guess)
21.
22. params2, covs2 = curve_fit(model2, t, i)
23.
24. a1 = params1[0]
25. b1 = params1[1]
26. c1 = params1[2]
27.
28. a2 = params2[0]
29. b2 = params2[1]
30.
31. print('Parametri a, b i c za model eksp. rasta sa limitom su:',
params1)
32. print('Parametri a, b i c za model plato funkcije su:', params2)
33.
34. i_fitted1 = model1(t,a1,b1,c1)
35. i_fitted2 = model2(t,a2,b2)
```

```
36.
37. plt.plot(t,i,'o',label='eksperiment')
38. plt.plot(t,i_fitted1,label='fit $y = a \mathrm{e}^{b*t} + c$')
39. plt.plot(t,i_fitted2,label='fit $y = \dfrac{a \cdot t}{b+t}$')
40. plt.xlabel("Vreme [s]")
41. plt.ylabel("Jacina struje [A]")
42. plt.legend()
43. plt.show()
>> Parametri a, b i c za model eksp. rasta sa limitom su: [-5.46333386 -
0.19793854  5.00428209]
>> Parametri a, b i c za model plato funkcije su: [6.12110283 5.13022823]
```

Konačno, prethodnim programom se dobija sledeći grafik.



Slika 8.12. Poređenje fitova sa eksperimentalnim rezultatima

Sa slike 8.12. se vidi odlično slaganje oba modela sa eksperimentalnim podacima, a čini se da fitovanje funkcijom eksponencijalnog rasta sa limitom daje rezultat koji je ipak malo bolje usklađen sa eksperimentalnim podacima. Međutim, uvek je nezahvalno donositi zaključke o tome koji je model bolji samo na osnovu vizuelne analize, posebno u slučajevima kada su njihove performanse bliske kao što je to slučaj ovde. Iz tog razloga, performanse oba modela će biti upoređene računanjem koeficijenta korelacije. Konačan primer sa programom koji crta oba fita i grafički ih poredi sa eksperimentalnim rezultatima, zajedno sa blokovima za računanje koeficijenta korelacije je dat u sledećem primeru.

Primer 8.11. Poređenje fitova i računanje koeficijenata korelacije

```
1.  import pandas as pd
2.  import matplotlib.pyplot as plt
3.  import numpy as np
```

```
from scipy.optimize import curve_fit
from sklearn.metrics import r2_score

df = pd.read_csv('RL_kolo.csv')

t_df = df['t']
t = t_df.to_numpy()
i_df = df['i']
i = i_df.to_numpy()

def model1(t,a,b,c):
    return a*np.exp(b*t) + c

def model2(t,a,b):
    return (a*t)/(b+t)

guess=[-1,-0.5,1]
params1, covs1 = curve_fit(model1, t, i,p0=guess)

params2, covs2 = curve_fit(model2, t, i)

a1 = params1[0]
b1 = params1[1]
c1 = params1[2]

a2 = params2[0]
b2 = params2[1]

print('Parametri a, b i c za model eksp. rasta sa limitom su:',
params1)
print('Parametri a, b i c za model plato funkcije su:', params2)

i_fitted1 = model1(t,a1,b1,c1)
i_fitted2 = model2(t,a2,b2)

r2_1 = r2_score(i, i_fitted1)
print('Koef. kor. za model eksp. rasta sa limitom je: ', r2_1)

r2_2 = r2_score(i, i_fitted2)
print('Koef. kor. za model plato funkcije je:', r2_2)

plt.plot(t,i,'o',label='eksperiment')
plt.plot(t,i_fitted1,label='fit $y = a \mathrm{e}^{b*t} + c$')
plt.plot(t,i_fitted2,label='fit $y = \dfrac{a \cdot t}{b+t}$')
plt.xlabel("Vreme [s]")
plt.ylabel("Jacina struje [A]")
plt.legend()
plt.show()
>> >> Parametri a, b i c za model eksp. rasta sa limitom su: [-5.46333386 -
0.19793854  5.00428209]
>> Parametri a, b i c za model plato funkcije su: [6.12110283 5.13022823]
>> Koef. kor. za model eksp. rasta sa limitom je:  0.9772672616759431
>> Koef. kor. za model plato funkcije je: 0.953207126388102
```

Grafički rezultat poslednjeg programa neće biti dat, jer je identičan kao slika 8.12. Vidi se da je u slučaju obe fit funkcije koeficijent korelacije izuzetno visok i iznosi preko 95%. Takođe se može videti da je koeficijent korelacije za 2% viši i bolji u slučaju kada se fitovanje vrši funkcijom eksponencijalnog rasta sa limitom, tako da bi nju trebalo birati u ovom slučaju.

8.4.4. Proračun vrednosti koeficijenta samoinduktivnosti iz fitovane funkcije

Nakon utvrđivanja koja funkcija daje bolji fit, preostaje još da se izračuna koeficijent samoinduktivnosti. U te svrhe, korisno je najpre uporediti matematičke oblike funkcije kojom su fitovani podaci sa egzaktnim rešenjem za vremensku zavisnost jačine struje kod RL kola, što je učinjeno u sledećem sistemu

$$\begin{gathered} i = \frac{V}{R}\left(1 - \mathrm{e}^{-\frac{R}{L}t}\right), \\ y = a\mathrm{e}^{bx} + c\,. \end{gathered} \tag{8.5}$$

Dati sistem može da se napiše u malo drugačijem obliku. Naime, u prvoj jednačini se prostim množenjem može ukloniti zagrada. Što se tiče druge jednačine, treba uzeti u obzir da jednačina eksponencijalnog rasta sa limitom dobro opisuje plato situaciju za $a = -1$, $b = -0{,}5$ i $c = 1$. Uzimajući sve u ovo obzir, prethodni sistem postaje

$$\begin{gathered} i = \frac{V}{R} - \frac{V}{R}\mathrm{e}^{-\frac{R}{L}t}, \\ y = c - a\mathrm{e}^{-bx}\,. \end{gathered} \tag{8.6}$$

Vidi se da su jednačine u poslednjem sistemu analogne, tj. da su istog matematičkog oblika. U tom smislu, ako se uporede članovi u eksponentu, vidi se da važi sledeća relacija

$$\frac{R}{L} = b\,. \tag{8.7}$$

Vrednost parametra b je dobijena u procesu fitovanja i iznosi $b = 0{,}19793854 \approx 0{,}198$. Na početku je zadato da je vrednost otpora $R = 1\ \Omega$, pa je sada lako izračunati vrednost koeficijenta samoinduktivnosti koji iznosi

$$L = \frac{1\ \Omega}{0{,}1979\,\frac{\Omega}{\mathrm{H}}} = 5{,}0531\ \mathrm{H}\,. \tag{8.8}$$

Za generisanje slučajnih vrednosti za ovaj eksperiment je uzeto da koeficijent samoindukcije iznosi upravo 5 H, zbog čega se zaključuje da je u relaciji (8.8) dobijena tačna vrednost.

8.5. Praktični primer fitovanja - određivanje perioda poluraspada protaktinijuma

Primer značaja fitovanja biće demonstriran na još jednom primeru – određivanju perioda poluraspada radioaktivnog elementa, što predstavlja nezaobilaznu eksperimentalnu vežbu na studijama fizike. S obzirom da je fitovanje detaljno prikazano u prethodna dva primera, ovaj primer će biti obrađen u nešto kraćoj formi.

Za određivanje poluraspada bira se neki kratkoživeći element, kao što je protaktinijum (Pa) čiji je period poluraspada nešto veći od minuta. Mera za brzinu raspada je veličina koja se naziva vreme poluraspada ($T_{1/2}$) i ona daje informaciju o vremenu neophodnom da se raspadne polovina od ukupnog, početnog, broja jezgara. Ova veličina može da uzima vrednosti od nekoliko minuta (npr. radijum, protaktinijum) do nekoliko milijardi godina (npr. uranijum). Izraz koji pokazuje koliko je jezgara ostalo neraspadnuto se naziva zakon radioaktivnog raspada i dat je sledećom funkcijom

$$N = N_0 \cdot e^{-\frac{\ln(2)}{T_{1/2}} \cdot t}. \tag{8.9}$$

Procedura za generisanje Pa je relativno laka, imajući u vidu komercijalnu dostupnost nephodnih hemikalija. Što je još bitnije, procedura dobijanja Pa se lako izvodi u školskim uslovima. Ovde neće biti iznošeni detalji generisanja Pa.

Merenje perioda poluraspada Pa se svodi na upotrebu Gajger-Milerovog brojača kako bi se prebrojavale emitovane čestice, a čitav postupak traje oko tri minuta, što čini ovu vežbu pogodnom za izvođenje u školskim uslovima. Merenje broja otkucaja Gajger-Milerovog brojača se beleži na svakih deset sekundi, sa napomenom da se radi o broju detektovanih čestica u sekundi. Rezultati jednog merenja perioda poluraspada protaktinijuma su dati u fajlu „Pa_raspad.csv“, koji se može učitati u python program. Ovaj fajl ima samo dve kolone, gde je prva kolona u .csv fajlu označena kao „t“ i predstavlja vreme tokom kojeg se meri broj otkucaja Gajger-Milerovog brojača. Druga kolona u pomenutom .csv fajlu je označena kao „brojac“ i predstavlja broj otkucaja Gajger-Milerovog brojača u sekundi u vremenu datom u koloni „t“. Takođe, treba pomenuti da se pored otkucaja Gajger-Milerovog brojača koji potiče od čestica Pa, javlja i izvestan broj otkucaja od drugih čestica, tzv. pozadinski raspadi, ali se u prvoj aproksimaciji može uzeti da je broj pozadinskih raspada konstantan tokom izvođenja eksperimenta u trajanju od tri minuta.

Prvi korak jeste crtanje grafika eksperimentalno dobijenih vrednosti, što je učinjeno u programu u sledećem primeru.

Primer 8.12. Crtanje eksperimentalnih vrednosti u eksperimentu sa raspadom Pa

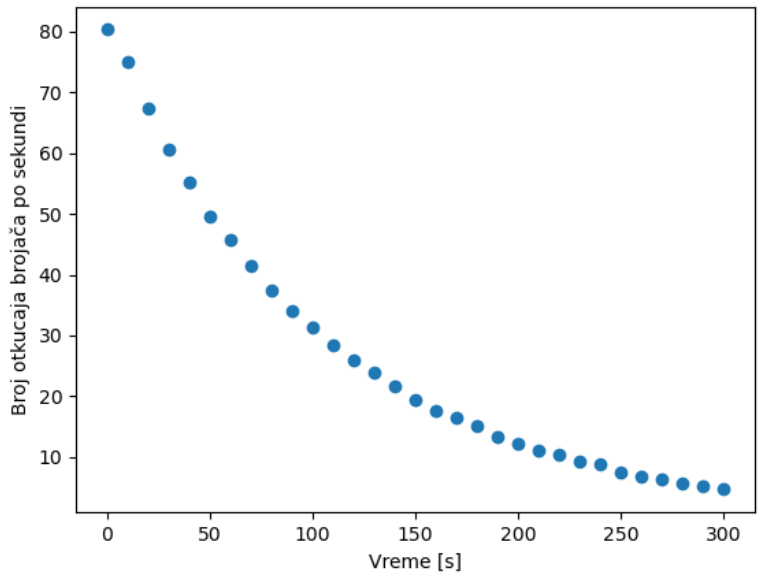
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import pandas as pd

df = pd.read_csv('Pa_raspad.csv')

t_df = df['t']
t = t_df.to_numpy()
brojac_df = df['brojac']
brojac = brojac_df.to_numpy()

plt.plot(t,brojac,'o')
plt.xlabel('Vreme [s]')
plt.ylabel('Broj otkucaja brojača po sekundi')
plt.show()
```

Rezultat programa iz poslednjeg primera je grafik predstavljen na slici 8.13.



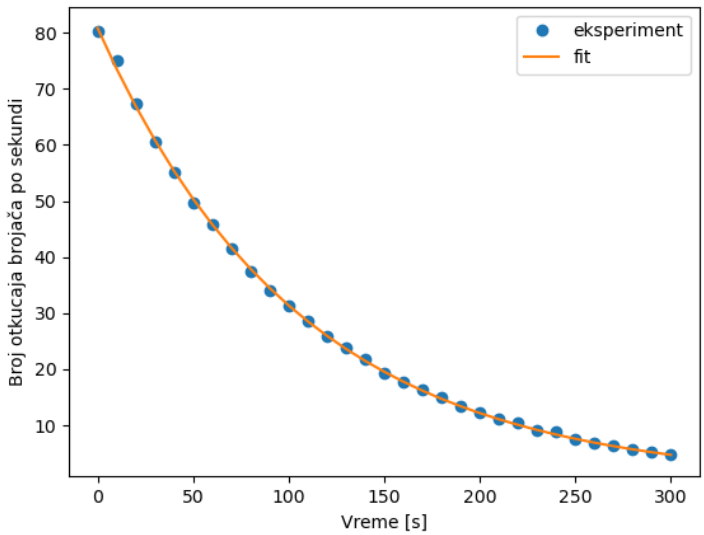
Slika 8.13. Broj otkucaja Gajger-Milerovog brojača u sekundi kod raspada Pa

Grafik vremenske zavisnosti broja otkucaja Gajger Milerovog brojača kod eksperimenta sa raspadom Pa ukazuje na to da se radi o eksponencijalnom padu. U tom smislu, jasno je da će se za fitovanje koristiti funkcija data izrazom (8.3). Program za fitovanje eksperimentalnih podataka pomenutom funkcijom, grafičku prezentaciju i računanje koeficijenta korelacije je dat u sledećem primeru.

Primer 8.13. Fitovanje podataka u vezi sa raspadom Pa

```
1.  import numpy as np
2.  import matplotlib.pyplot as plt
3.  from scipy.optimize import curve_fit
4.  import pandas as pd
5.  from sklearn.metrics import r2_score
6.
7.  df = pd.read_csv('Pa_raspad.csv')
8.
9.  t_df = df['t']
10. t = t_df.to_numpy()
11. brojac_df = df['brojac']
12. brojac = brojac_df.to_numpy()
13.
14. def model(t,a,b):
15.     return a*np.exp(-b*t)
16.
17. params, covs = curve_fit(model, t, brojac)
18.
19. a = params[0]
20. b = params[1]
21.
22. brojac_fitted = model(t,a,b)
23.
24. print('a =',a,'b =',b)
25.
26. r2 = r2_score(brojac, brojac_fitted)
27. print('Koeficijent korelacije je: ', r2)
28.
29. plt.plot(t,brojac,'o',label='eksperiment')
30. plt.plot(t,brojac_fitted,label='fit')
31. plt.xlabel('Vreme [s]')
32. plt.ylabel('Broj otkucaja brojača po sekundi')
33. plt.legend()
34. plt.show()
>> a = 80.84745144583248 b = 0.009467710098723374
>> Koeficijent korelacije je:  0.9996755884116473
```

Rezultat programa su koeficijenti a i b, koeficijent korelacije, kao i grafik gde su upoređene eksperimentalne vrednosti sa fitovanom funkcijom (predstavljeno na slici 8.14.).



Slika 8.14. Fitovanje podataka kod eksperimenta sa raspadom Pa

Na slici 8.14. se vidi izuzetno dobro slaganje, koje je potvrđeno i veoma visokim koeficijentom korelacije koji iznosi skoro 100%. Preostaje još da se odredi vrednost perioda poluraspada. Kao i u prošlom primeru, uporediće se egzaktno rešenje i funkcija kojom su fitovani podaci, što je dato u sledećem sistemu jednačina

$$\begin{aligned} N &= N_0 \cdot e^{-\frac{\ln(2)}{T_{1/2}} \cdot t}, \\ y &= a e^{-bx}. \end{aligned} \tag{8.10}$$

Poređenjem jednačina, vidi se da su one analogne, a poređenjem članova u eksponentu, zaključuje se da važi sledeća relacija

$$\frac{\ln(2)}{T_{1/2}} = b. \tag{8.11}$$

S obzirom da je vrednost parametra b izračunata tokom fitovanja, period poluraspada se može sada lako izračunati preko relacije koja sledi

$$T_{1/2} = \frac{\ln(2)}{b} = \frac{0.6931}{0.0095\,\frac{1}{\mathrm{s}}} = 72{,}9629\ \mathrm{s}. \tag{8.12}$$

Dobijena vrednost je u izuzetno dobrom slaganju sa poznatom vrednošću perioda poluraspada Pa koja iznosi 70 s.

9. Interpolacija i ekstrapolacija

Pod interpolacijom se podrazumeva pronalaženje nepoznate vrednosti funkcije između dve poznate tačke. Sa druge strane, ekstrapolacija predstavlja generisanje vrednosti funkcije van intervala pokrivenog dostupnim podacima.

Tipičan primer primene interpolacije/ekstrapolacije se sreće kod obrade rezultata eksperimenata. Pri grafičkom prikazu rezultata, može se naslutiti funkcionalna zavisnost između veličina. Međutim, procena vrednosti funkcije u tački za koju ne postoji eksperimentalan rezultat može biti izazovna, zbog čega se koriste različiti metodi kako bi se generisale adekvatne vrednosti funkcije u oblastima između ili izvan tačaka. Naravno, vremenom su razvijeni mnogi metodi interpolacije i ekstrapolacije, od kojih su za oba tipa najpoznatije linearna i polinomna metoda. U vezi sa interpolacijom, treba pomenuti i kubnu i splajn interpolaciju, kao i interpolaciju metodom najbližih suseda.

U slučaju programskog jezika python, najpoznatija funkcija za interpolaciju dolazi iz **scipy** biblioteke, konkretno njenog **integrate** modula i zove se `interp1d()`. Tri su glavna argumenta ove funkcije: podaci za x-osu, podaci za y-osu i definisanje vrednosti parametra `kind=`. Vrednost parametra `kind` je *string* tip podatka i definiše koji tip interpolacije treba da se sprovede nad podacima. U okviru funkcije `interp1d()` dostupni su mnogi poznati metodi interpolacije kao što su linerni, splajn ili metod najbližih suseda.

Sa druge strane, da bi se izvršila ekstrapolacija, prvo je potrebno identifikovati adekvatnu funkcionalnu zavisnost među sakupljenim podacima. Kada se utvrdi funkcija koja najbolje opisuje eksperimentalno dobijene podatke, u tu funkciju se mogu uneti vrednosti argumenata van intervala eksperimentalnih podataka, i na taj način predvideti ponašanje sistema. Naravno, u ovom slučaju se pretpostavlja da ista funkcionalna zavisnost važi i u odabranoj oblasti van intervala koji je pokriven sakupljenim podacima.

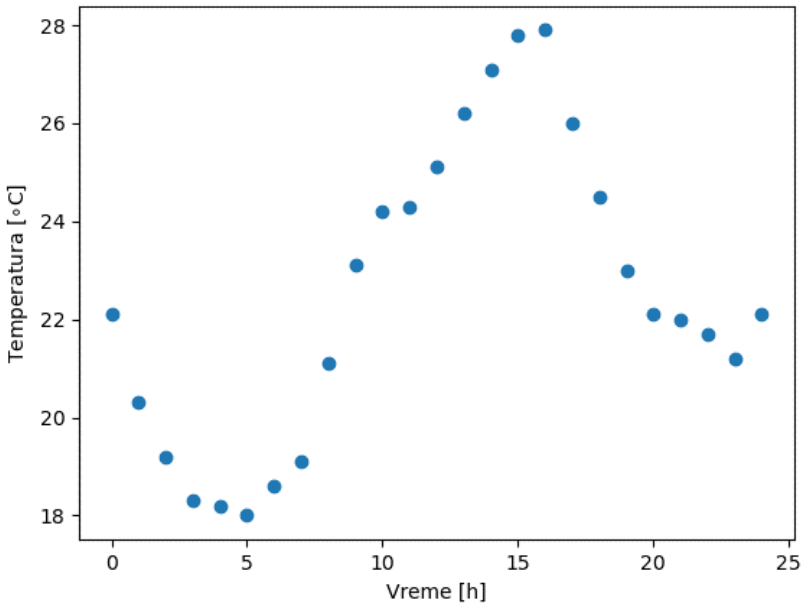
9.1. Primeri interpolacije – primena funkcije `interp1d()`

U sledećih nekoliko primera, biće iskorišćeni podaci iz fajla „interpolacija.csv“, u kome se nalaze rezultati merenja temperature u jednoj od prostorija kuće tokom 24 h. Temperatura je merena na svakih sat vremena, što je grub korak. U .csv fajlu, nalaze se podaci za vreme i temperaturu, redom. Program za crtanje ovih podataka je dat u primeru koji sledi.

Primer 9.1. Crtanje podataka iz fajla „interpolacija.csv“

```
1.  import pandas as pd
2.  import numpy as np
3.  import matplotlib.pyplot as plt
4.
5.  df = pd.read_csv('interpolacija.csv')
6.
7.  x = df["vreme"]
8.  y = df["temperatura"]
9.
10. plt.plot(x,y,'o')
11. plt.xlabel('Vreme [h]')
12. plt.ylabel('Temperatura [$\circ \mathrm{C}$]')
13. plt.show()
```

Kao izlaz, ovaj program će dati vremensku zavisnost temperature u prostoriji kao što je to prikazano na slici 9.1.



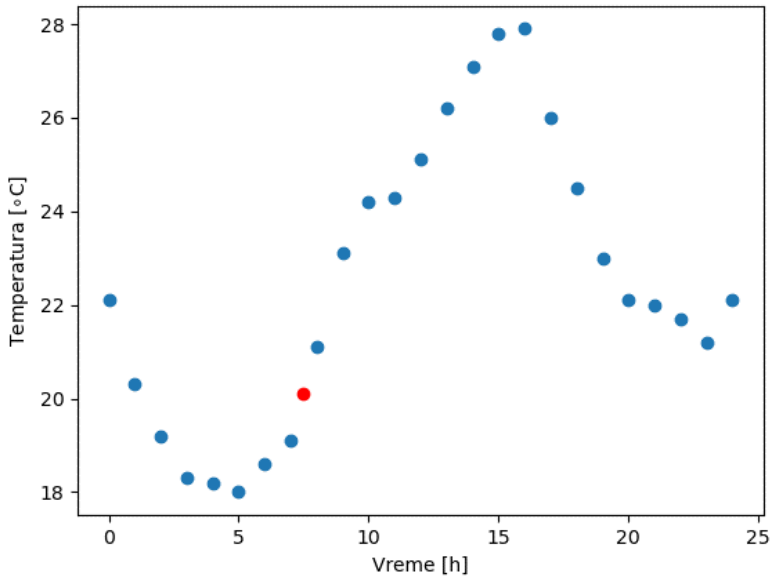
Slika 9.1. Vremenska zavisnost temperature iz fajla „interpolacija.csv“

Analizom slike 9.1. jasno se utvrđuje trend, međutim postavlja se pitanje kako pouzdano utvrditi temperaturu u trenutku od npr. 7,5 h od početka merenja temperature? U te svrhe, biće sprovedena linearna interpolacija podataka. U primeru koji sledi, biće predstavljen program u kome će podaci iz fajla biti interpolirani linearnim metodom, a zatim će se na osnovu interpolacione funkcije izračunati vrednost temperature u trenutku od 7,5 h od početka merenja, za koji ne postoji eksperimentalni podatak.

Primer 9.2. Interpolacija podataka linearnom metodom

```
1.  import pandas as pd
2.  import numpy as np
3.  import matplotlib.pyplot as plt
4.  from scipy.interpolate import interp1d
5.
6.  df = pd.read_csv('interpolacija.csv')
7.
8.  x = df["vreme"]
9.  y = df["temperatura"]
10.
11. interpolacija = interp1d(x,y,kind='linear')
12.
13. x_new = 7.5
14. y_new = interpolacija(x_new)
15. print('Temperatura u',x_new,'h iznosi ',y_new,'stepeni Celzijusa')
16.
17. plt.plot(x,y,'o')
18. plt.plot(x_new,y_new,'or')
19. plt.xlabel('Vreme [h]')
20. plt.ylabel('Temperatura [$\circ \mathrm{C}$]')
21. plt.show()
>> Temperatura u 7.5 h iznosi  20.1 stepeni Celzijusa
```

U primeru 9.2. interpolacija podataka x i y se vrši u liniji 11, gde je definisan objekat `interpolacija`. Taj objekat će sadržavati interpolacionu funkciju i u ostatku kôda se ponaša upravo kao funkcija. Dakle, ako se objektu `interpolacija` dâ argument, on će izračunati novu vrednost. Objektu `interpolacija` može da se dâ jedna vrednost kao argument, a može da se da i više vrednosti. U primeru 9.2, kao argument funkciji `interpolacija` je data samo jedna vrednost, linija 13, i to je vrednost vremena za koje nema eksperimentalnog podatka. Za tu jednu vrednost `x_new`, `interpolacija(x_new)` će dati niz koji sadrži samo jednu vrednost, a to je vrednost temperature u trenutku 7,5 h od početka merenja na osnovu interpolacione funkcije (linija 14). Izlaz od `interpolacija(x_new)` je nazvan `y_new` i u liniji 15 se traži štampanje te vrednosti. U linijama 17-21 se daju instrukcije za crtanje grafika koji će pored eksperimentalnih vrednosti crtati i jednu, interpoliranu, vrednost tačkom crvene boje, kao što je to dato na slici 9.2.



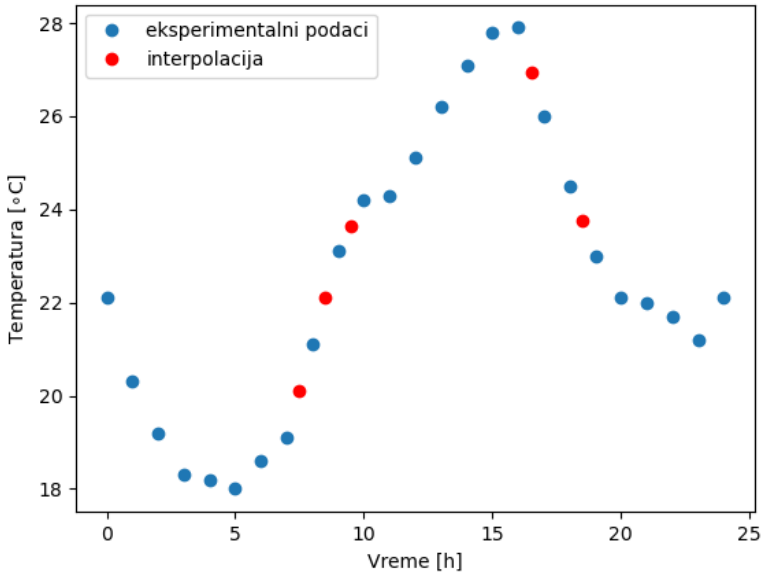
Slika 9.2. Eksperimentalne vrednosti sa inteproliranom tačkom (primer 9.2.)

U liniji 13 programa umesto jedne vrednosti, mogla se definisati lista od nekoliko vrednosti, koja bi se potom prosledila kao argument funkciji `interpolacija()`. Tada bi se kao izraz dobilo više interpoliranih vrednosti za *y*, tj. u ovom primeru za temperaturu. Takav slučaj je prikazan u sledećem primeru

Primer 9.3. Interpolacija više vrednosti

```
1.  import pandas as pd
2.  import numpy as np
3.  import matplotlib.pyplot as plt
4.  from scipy.interpolate import interp1d
5.
6.  df = pd.read_csv('interpolacija.csv')
7.
8.  x = df["vreme"]
9.  y = df["temperatura"]
10.
11. interpolacija = interp1d(x,y,kind='linear')
12.
13. x_new = [7.5,8.5,9.5,16.5,18.5]
14. y_new = interpolacija(x_new)
15. print('Temperature u',x_new,'h iznose ',y_new,'stepeni Celzijusa')
16.
17. plt.plot(x,y,'o',label='eksperimentalni podaci')
18. plt.plot(x_new,y_new,'or',label='interpolacija')
19. plt.xlabel('Vreme [h]')
20. plt.ylabel('Temperatura [$\circ \mathrm{C}$]')
21. plt.legend()
22. plt.show()
>> Temperature u [7.5, 8.5, 9.5, 16.5, 18.5] h iznose [20.1  22.1  23.65
26.95 23.75] stepeni Celzijusa
```

Konkretno, u liniji 13 je definisana lista sa 5 vrednosti temperature (za koje ne postoje eksperimentalni podaci), da bi potom bila iskorišćena kao argument funkcije `interpolacija`, koja isto tako kao izlaz daje pet vrednosti temperature. Poslednji program kao izlaz daje grafik kao na slici 9.3.



Slika 9.3. Grafik sa pet interpoliranih tačaka

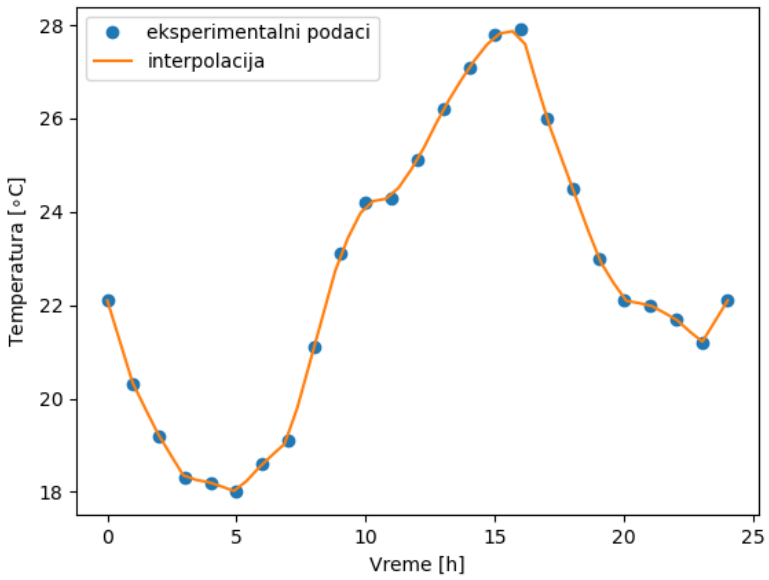
U poslednjem primeru je lista novih vrednosti bila definisana ručno, ali korisnik može to učiniti i jednostavnije, npr. upotrebom funkcije `np.linspace()`. Upotrebom konkretno funkcije `linspace()` moguće je definisati čitav skup vrednosti sa malim korakom, puno tzv. međuvrednosti, što bi dovelo do glatke krive u slučaju obrađenog primera. Jedan takav program je predstavljen u sledećem primeru.

Primer 9.4. Interpolacija sa više vrednosti definisanih funkcijom `linspace()`

```
1.  import pandas as pd
2.  import numpy as np
3.  import matplotlib.pyplot as plt
4.  from scipy.interpolate import interp1d
5.
6.  df = pd.read_csv('interpolacija.csv')
7.
8.  x = df["vreme"]
9.  y = df["temperatura"]
10.
11. interpolacija = interp1d(x,y,kind='linear')
12.
13. x_new = np.linspace(0,24,100)
14. y_new = interpolacija(x_new)
15.
16. plt.plot(x,y,'o',label='eksperimentalni podaci')
```

```
17. plt.plot(x_new,y_new,label='interpolacija')
18. plt.xlabel('Vreme [h]')
19. plt.ylabel('Temperatura [$\circ \mathrm{C}$]')
20. plt.legend()
21. plt.show()
```

Kao rezultat, poslednji program će dati grafik kao na slici 9.4.



Slika 9.4. Rezultat interpolacije sa puno međuvrednosti

U poslednjem primeru je linijom 13 generisano mnogo međuvrednosti, zbog čega ima i puno vrednosti za temperaturu dobijenih interpolacijom. Iz tog razloga je izostavljena linija za štampanje izračunatih vrednosti. Veliki broj interpoliranih vrednosti omogućava da se dobije relativno glatka kriva, pa je u liniji 17 data instrukcija da se nacrta kriva, a ne tačke.

Da bi se upotrebio neki drugi metod interpolacije, dovoljno je parametru `kind` umesto vrednosti `linear` dodeliti neku drugu podržanu vrednost. Npr. da bi se iskoristio metod kvadratne interpolacije, u poslednjem programu se linija 11 menja iz

```
interpolacija = interp1d(x,y,kind='linear')
```

u liniju

```
interpolacija = interp1d(x,y,kind='quadratic')
```

čime se kao rezultat interpolacija dobija kriva koja je mnogo glađa od krive dobijene metodom linearne interpolacije. Zarad preglednosti, u sledećoj tabeli su sumirane neke moguće vrednosti parametra `kind`, sa kratkim objašnjenjem šta koja vrednost znači.

Tabela 9.1. Neke moguće vrednosti parametra `kind`

Vrednost	Opis
linear	Metoda linearne interpolacije, podrazumevana metoda
nearest	Metoda najbližih suseda, pri čemu se zaokruživanje vrši na nižu vrednost
nearest-up	Metoda najbližih suseda, pri čemu se zaokruživanje vrši na višu vrednost
zero	Splajn interpolacija nultog reda
slinear	Splajn interpolacija prvog reda
quadratic	Splajn interpolacija drugog reda
cubic	Splajn interpolacija trećeg reda

Važno je naglasiti da postoje i druge funkcije za interpolaciju rezultata, kako u okviru paketa **scipy** tako i okviru drugih biblioteka. No, pored toga što je izuzetno funkcionalna, biblioteka **scipy** je izuzetno dobro dokumentovana, što dodatno utiče na njenu popularnost.

9.2. Primer ekstrapolacije

Kao što je naglašeno u prvom odeljku ovog poglavlja, za predviđanje vrednosti neke veličine van intervala pokrivenog sakupljenim podacima, neophodno je doći do adekvatne funkcije koja dobro opisuje vezu između veličina. Pod pretpostavkom da ista funkcionalna zavisnost važi i u drugim intervalima, može se predvideti vrednost neke veličine. Ovo se naziva ekstrapolacija i biće demonstrirana preko Gej-Lisakovog zakona.

Gej-Lisakov zakon daje vezu između pritiska određene količine idealnog gasa i njegove temperature, kada se idealni gas nalazi pod konstantnom zapreminom. Matematički gledano, Gej-Lisakov zakon se može izraziti sledećom relacijom

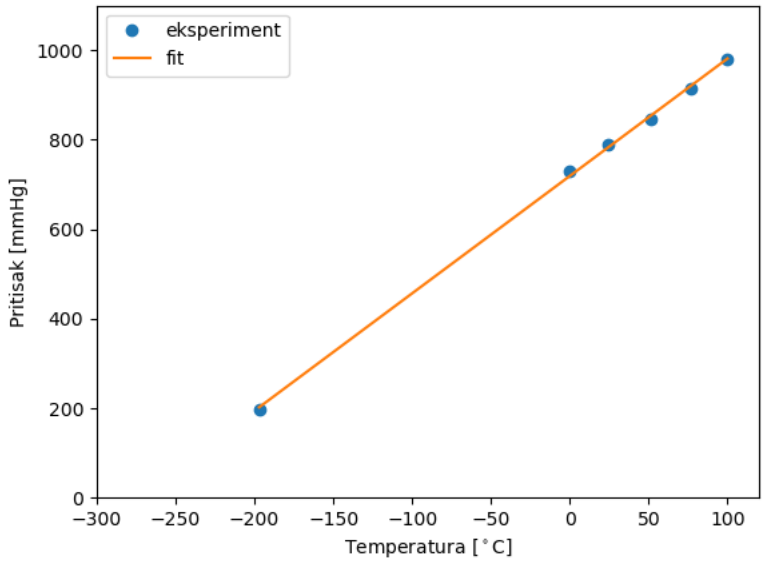
$$\frac{p}{T} = const\,. \tag{9.1}$$

Dakle, odnos pritiska i temperature idealnog gasa konstantne zapremine će uvek biti isti. Gej-Lisakov zakon se eksperimentalno utvrđuje tako što se izvesna zapremina idealnog gasa „zarobi“ u kontejner sa manometrom, tako da u svakom momentu može da se odredi pritisak gasa u tom kontejneru. Zatim se taj kontejner sa idealnim gasom stavlja na različite temperature, npr. tako što se prvo stavi u ključalu vodu, zatim u vodu na različitim temperaturama na kojima je voda u tečnom stanju, zatim u vodu sa ledom, i konačno u tečni azot. Na ovaj način, konstantna zapremina idealnog gasa se izlaže temperaturama u intervalu od 77 K do 373 K, pri čemu se mere pritisci tog idealnog gasa. Rezultati jednog takvog eksperimenta su sumirani u .csv fajlu koji se zove „gej_lisak.csv“ i u sledećem primeru će biti predstavljen program kojim se pomenuti fajl učitava, a podaci fituju linearnim fitom.

Primer 9.5. Fitovanje podataka Gej-Lisakovog eksperimenta

```
1.  import numpy as np
2.  import matplotlib.pyplot as plt
3.  from scipy.optimize import curve_fit
4.  import pandas as pd
5.  from sklearn.metrics import r2_score
6.
7.  df = pd.read_csv('gej_lisak.csv')
8.
9.  t_df = df['temperatura']
10. t = t_df.to_numpy()
11. p_df = df['pritisak']
12. p = p_df.to_numpy()
13.
14. def model(t,a,b):
15.     return a*t + b
16.
17. params, covs = curve_fit(model, t, p)
18.
19. a = params[0]
20. b = params[1]
21.
22. p_fitted = model(t,a,b)
23.
24. print('a =',a,'b =',b)
25.
26. r2 = r2_score(p, p_fitted)
27. print('Koeficijent korelacije je: ', r2)
28.
29. plt.plot(t,p,'o',label='eksperiment')
30. plt.plot(t,p_fitted,label='fit')
31. plt.xlabel('Temperatura [$^\circ \mathrm{C}$]')
32. plt.ylabel('Pritisak [mmHg]')
33. plt.xlim(-300,120)
34. plt.ylim(0,1100)
35. plt.legend()
36. plt.show()
>> a = 2.623556852859427 b = 718.9070028903044
>> Koeficijent korelacije je:  0.9992292090313496
```

U principu je program po formi identičan ostalim programima u vezi sa fitovanjem predstavljenim u prethodnim odeljcima, sa razlikom što su u linijama 33 i 34 „ručno“ definisano opsezi x- i y-osa, iz razloga koji će biti uskoro objašnjen. Pored vrednosti nagiba i odsečka, dati program daje kao rezultat grafik predstavljen na slici 9.5.



Slika 9.5. Fitovanje podataka Gej-Lisakovog eksperimenta

Na slici 9.5 je prikazana temperaturska zavisnost pritiska idealnog gasa konstantne zapremine, zajedno sa odgovarajućim linearnim fitom. Opseg duž temperaturske ose je odabran od -300 °C do 120 °C, kako bi se prikazao i opseg koji nije pokriven eksperimentalnim podacima. Naime, ukoliko bi se fit linija nastavila van opsega pokrivenog eksperimentalnim podacima, pri pritisku od 0 mmHg, linija bi temperatursku osu sekla u blizini tačke od -273 °C. Prema klasičnom shvatanju, pri toj temperaturi pritisak bi bio jednak nuli, nikakva sila ne bi delovala na čestice gasa, pa ne bi bilo ni kretanja čestica. Iz tog razloga je ta vrednost nazvana apsolutna nula, jer se ne može postići temperatura niža od te.

Pored grafičkog pristupa, vrednost apsolutne nule se na osnovu podataka iz fajla „gej_lisak.csv“ može dobiti i računski. Naime, vrednost koeficijenta korelacije iz poslednjeg programa je skoro 99,99%, što ukazuje na to da je linearni fit

$$y = a \cdot x + b\,, \tag{9.2}$$

adekvatna funkcija koja opisuje međusbnu zavisnost pritiska i temperature idealnog gasa pri konstantnoj zapremini, pa se može napisati sledeća relacija

$$p = 2{,}62 \cdot t + 718{,}91\,. \tag{9.3}$$

Pri nultom pritisku, iz poslednje relacije sledi

$$t = -\frac{718.91}{2{,}62} = -274{,}39\ ℃\,. \tag{9.4}$$

što je vrednost izuzetno bliska vrednosti apsolutne nule (-273,15 °C) utvrđenoj savremenim eksperimentima.

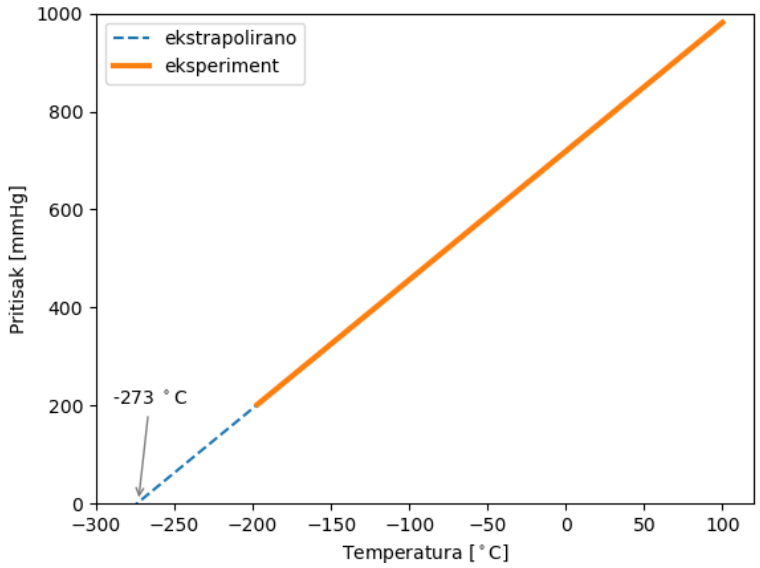
Korisno je još pokazati i kako se grafički može tretirati ovaj slučaj. Jednom kada se dobije fit funkcija, mogu se generisati nove vrednosti za x osu (konkretno one vrednosti koje su van opsega pokrivenog eksperimentalnim podacima) i zatim se one iskoristiti kao argument fit funkcije. Na ovaj način će fit funkcija generisati nove vrednosti za y osu van intervala pokrivenog eksperimentalnim podacima. Program je predstavljen u sledećem primeru.

Primer 9.6. Ekstrapolacija podataka iz Gej-Lisakovog eksperimenta grafičkim putem

```
1.  import numpy as np
2.  import matplotlib.pyplot as plt
3.  from scipy.optimize import curve_fit
4.  import pandas as pd
5.
6.  df = pd.read_csv('gej_lisak.csv')
7.
8.  t_df = df['temperatura']
9.  t = t_df.to_numpy()
10. p_df = df['pritisak']
11. p = p_df.to_numpy()
12.
13. def model(t,a,b):
14.     return a*t + b
15.
16. params, covs = curve_fit(model, t, p)
17.
18. a = params[0]
19. b = params[1]
20.
21. x_new = np.linspace(-300,100,1000)
22. y_new = model(x_new,a,b)
23.
24. plt.plot(x_new,y_new,'--',label='ekstrapolirano')
25. plt.plot(t,p,linewidth=3,label='eksperiment')
26. plt.xlabel('Temperatura [$^\circ \mathrm{C}$]')
27. plt.ylabel('Pritisak [mmHg]')
28. plt.xlim(-300,120)
29. plt.ylim(0,1000)
30. plt.annotate("-273 $^\circ \mathrm{C}$", xy=(-273, 5),
xytext=(-290, 200), arrowprops={"arrowstyle":"->", "color":"black"})
31. plt.legend()
32. plt.show()
```

Forma programa je slična prethodnom programu, sa razlikom što se u linijama 21 i 22 generišu nove vrednosti za x- i y-osu. Pre toga su podaci fitovani linearnom funkcijom i dobijene su iste vrednosti nagiba i odsečka kao u prethodnom programu, što definiše fitovanu funkciju. U nastavku se crtaju ekstrapolirana i fitovana funkcija, redom, tako što je ekstrapolirana funkcija data isprekidanom linijom, što reguliše deo koda "`--`" u liniji 24. Sa

druge strane, u liniji 25 fit linija je podebljana delom koda "`linewidth=3`". Fitovana funkcija iz linije 25 pokriva "samo" opseg eksperimentalnih podataka. Sa druge strane, zahvaljujući vrednostima iz linije 21, ekstrapolirana funkcija će se prostirati u temperaturskom intervalu od -300 ℃ do 100 ℃. U linijama 28 i 29 je definisano u kom opsegu će biti nacrtan grafik. U liniji 30 se dodaje kôd za crtanje strelice koja ukazuje na mesto na x osi kojem odgovara temperatura od -273 ℃. Kôd iz linije 30 se nije do sada objašnjavao, jer spada u specifičnosti **matplotlib** biblioteke, a korisnici mogu slobodno da ga izuzmu iz ovog programa ukoliko unosi zabunu. Program iz poslednjeg primera će dati grafik kao na slici 9.6.



Slika 9.6. Ekstrapolirana i fitovana funkcija u slučaju Gej-Lisakovog zakona

Sa slike 9.6. se vidi da ekstrapolirana funkcija za nultu vrednost pritiska seče temperatursku osu u blizini apsolutne nule, što je odlično slaganje sa poznatim činjenicama. U ovom konkretnom primeru je razrađen slučaj kada se funkcija fituje linearnom funkcijom, međutim, procedura je identična i ukoliko se radi o fitovanju nekom drugom funkcijom.

10. Rešavanje diferencijalnih jednačina i integrala

Mnogi problemi u nauci se svode na rešavanje diferencijalnih jednačina i integrala. Međutim, u slučajevima složenijih modela, mnoge diferencijalne jednačine i integrali ne mogu da se reše egzaktno, ili ih je jako teško rešiti egzaktno. Iz tog razloga se često pribegava

numeričkim (aproksimativnim) metodama rešavanja, pa je numeričko rešavanje diferencijalnih jednačina i integrala nezaobilazni alat u prirodnim i tehničkim naukama.

Programski jezik python poseduje sjajne mogućnosti po pitanju numeričkog rešavanja diferencijalnih jednačina i integrala. Postoje gotove biblioteke pomoću kojih se do numeričkog rešenja dolazi u svega par linija kôda, i one će svakako biti prikazane u ovom udžbeniku. Međutim, u programerskom kontekstu, korisno je demonstrirati kako se programski jezik python može iskoristiti i za pisanje programa u kojima se implementira neki algoritam za numeričko rešavanje od samog početka.

U ovom odeljku biće pokazano kako se mogu napisati python programi za rešavanje diferencijalnih jednačina upotrebom dobro poznatih metoda kao što su Ojlerov, Ojler-Kromerov, Runge-Kuta 4. reda. Ovi metodi će biti primenjeni na neke tipične situacije i modele iz fizike. Što se tiče numeričkog rešavanja integrala, biće razrađeni programi za numeričko rešavanje integrala primenom trapezoidnog i Simpsonovog pravila. Na kraju odeljka biće prikazano kako se i uz pomoć gotovih funkcija može doći do numeričkih rešenja diferencijalnih jednačina i integrala.

10.1. Numeričko rešavanje diferencijalnih jednačina Ojlerovim metodom

Neka je data diferencijalna jednačina prvog reda

$$\frac{dy}{dx} = y' = f(x, y)\,, \tag{10.1}$$

sa sledećim početnim vrednostima:

$y(x_0) = y_0$, tj. početne vrednosti su x_0 i y_0.

Ojlerov metod za numeričko rešavanje diferencijalne jednačine podrazumeva upotrebu sledeće formule

$$y_{i+1} = y_i + h \cdot f(x_i, y_i)\,. \tag{10.2}$$

gde h predstavlja korak (eng. *step size*), odnosno vrednost za koliko se menjaju vrednosti promenljive x i y (tako da važi da je $x_{i+1} = x_i + h$, odnosno $y_{i+1} = y_i + h$).

Podsećanja radi, najpre će primena Ojlerovog metoda biti demonstrirana bez programiranja.

Primer 10.1. Primena Ojlerovog metoda

Primenom Ojlerovog metoda rešiti diferencijalnu jednačinu

$$\frac{dy}{dx} = 2\,, \tag{10.3}$$

u tački $x = 1{,}5$, ako su početne vrednosti $x_0 = 1$, $y_0 = 1$. Egzaktno rešenje ove diferencijalne jednačine je $y = x^2$. Uzeti da vrednost koraka h iznosi 0,1 i uporediti aproksimativno i egzaktno rešenje.

Ovakav tip zadataka najjasnije je rešavati tabelarno, kao što je to učinjeno u tabeli 10.1. Najpre je neophodno definisati sve vrednosti koje uzima x. Neka su ove vrednosti označene sa x_i. S obzirom da je početna vrednost x_0 jednaka 1, da krajnja vrednost iznosi 1,5, kao i da korak iznosi 0,1, jasno je da će u tabeli x_i uzimati sledeće vrednosti: 1; 1,1; 1,2; 1,3; 1,4 i 1,5. U ovom konkretnom primeru, prisutna je samo nezavisno promenljiva, pa se Ojlerova jednačina pojednostavljuje na $y_{i+1} = y_i + h \cdot f(x_i)$. Njome se aproksimativno računaju odgovarajuće vrednosti y_i, zaključno sa vrednošću za $x_5 = 1{,}5$. Neposredno posle tabele 10.1. detaljno je prikazana procedura primene Ojlerovog metoda za aproksimativno rešenje date diferencijalne jednačine.

Tabela 10.1. Tabelarni prikaz vrednosti za primenu Ojlerovog metoda

i	x_i	y_i	y_{ex}
0	1,00	1,00	1,00
1	1,10	1,20	1,21
2	1,20	1,42	1,44
3	1,30	1,66	1,69
4	1,40	1,92	1,96
5	1,50	2,20	2,25

$$\begin{gathered} y_1 = y_0 + 0{,}1 \cdot (2 \cdot x_0) = 1 + 0{,}1 \cdot (2 \cdot 1) = 1 + 0{,}2 = 1{,}2\,, \\ y_2 = y_1 + 0{,}1 \cdot (2 \cdot x_1) = 1{,}2 + 0{,}1 \cdot (2 \cdot 1{,}1) = 1{,}2 + 0{,}22 = 1{,}42\,, \\ y_3 = y_2 + 0{,}1 \cdot (2 \cdot x_2) = 1{,}42 + 0{,}1 \cdot (2 \cdot 1{,}2) = 1{,}42 + 0{,}24 = 1{,}66\,, \\ y_4 = y_3 + 0{,}1 \cdot (2 \cdot x_3) = 1{,}66 + 0{,}1 \cdot (2 \cdot 1{,}3) = 1{,}66 + 0{,}26 = 1{,}92\,, \\ y_5 = y_4 + 0{,}1 \cdot (2 \cdot x_4) = 1{,}92 + 0{,}1 \cdot (2 \cdot 1{,}4) = 1{,}92 + 0{,}28 = 2{,}20\,, \\ y(1{,}5) \approx 2{,}20\,. \end{gathered} \tag{10.4}$$

Vidi se da je aproksimativna metoda rešavanja diferencijalne jednačine prema Ojleru dala rezultat koji je vrlo blizak egzaktnoj vrednosti, posebno kada se uzme u obzir relativno velika vrednost koraka h. Ukoliko bi se uzela manja vrednost koraka h, dobila bi se i tačnija aproksimativna vrednost.

Međutim, smanjenje vrednosti koraka h nosi sa sobom jednu značajnu poteškoću kada se radi o primeni aproksimativnih metoda rešavanja diferencijalnih jednačina. Naime, ako bi za rešavanje prethodnog zadatka umesto vrednosti $h = 0{,}1$ bila izabrana vrednost $h = 0{,}01$, to bi dovelo do toga da umesto 5 vrednosti za x_i (pored početnih vrednosti) ima 50 vrednosti. Tako bi se procedura aproksimativnog rešavanja diferencijalne jednačine povećala sa 5 iteracija na 50 iteracija! Sprovođenje velikog broja iteracija ručno je veoma zahtevno i svakako nije praktično. Zbog toga se koriste programski jezici i softverski alati. U sledećem primeru, biće predstavljen python program za rešavanje date diferencijalne jednačine.

Primer 10.2. Python program za rešavanje diferencijalne jednačine date izrazom (10.3)

```
1.  import numpy as np
2.
3.  def dy_dx(x):
4.      return 2*x
5.
6.  def y_ex(x):
7.      return x**2
8.
9.  x0 = 1
10. xf = 1.5
11. y0 = 1
12. n = 6
13. h = (xf-x0)/(n-1)
14.
15. x = np.linspace(x0,xf,n)
16. y = np.zeros(n)
17.
18. x[0] = x0
19. y[0] = y0
20.
21. for i in range(0,n-1):
22.     y[i+1] = y[i] + h * dy_dx(x[i])
23.
24. print("Aproksimativno resenje je: ",y[i+1])
25. print("Egzaktno resenje je",y_ex(xf))
```

U liniji 1 je učitana **numpy** biblioteka. U linijama 3,4 i 6,7 su definisane diferencijalna jednačina, odnosno jednačina koja predstavlja egzaktno rešenje diferencijalne jednačine. U linijama 9 i 10 su definisane početna i krajnja vrednost promenljive x. U liniji 11 je definisana početna vrednost promenljive y. Linija 12 definiše broj tačaka na koliko se interval između x_0

i x_f deli, dok je u liniji 13 definisana vrednost koraka h. U liniji 15 su preko funkcije `linspace()` definisane sve vrednosti x koje će biti korišćene (kao u tabeli 10.1), dok su u liniji 16 definisane početne vrednosti y. Vrednosti y su potpuno proizvoljne, a ideja je da se ove vrednosti zamenjuju pravim vrednostima upotrebom Ojlerove metode u ***for*** petlji koja sledi. U liniji 18 i 19 su definisane početne vrednosti od x i y. Linije 21 i 22 su glavne linije ovog programa. U njima se praktično primenjuje Ojlerov algoritam, tj. sukcesivno se vrednosti y zamenjuju pravim vrednostima primenom Ojlerove metode. ***for*** petlja se primenjuje onoliko puta koliko ima vrednosti u listama x ili y i za svaki korak se računa nova vrednost, baš kao što je to slučaj sa sistemom jednačina (10.4). Konačno, u linijama 24 i 25 štampaju se vrednosti numeričkog i egzaktnog pristupa.

10.2. Numeričko rešavanje diferencijalnih jednačina naprednijim metodama

Ojlerov metod predstavlja najjednostvniji pristup za numeričko rešavanje diferencijalnih jednačina. Bez obzira na svoju jednostavnost, primenljiv je na čitav niz primera, od kojih će neki biti prikazani u sledećem odeljku. Postoje mnogi drugi metodi pomoću kojih se mogu numerički rešavati diferencijalne jednačine, i svaki od tih metoda se zasniva na izvesnoj, manje ili više složenoj formuli. Od interesa je pomenuti neke od njih.

Unapređeni Ojlerov metod. Ovaj metod se zasniva na upotrebi sledeće formule

$$y_{i+1} = y_i + \frac{h}{2}(k_{1i} + k_{2i})\,, \tag{10.5}$$

$$k_{1i} = f(x_i, y_i)\,, \tag{10.6}$$

$$k_{2i} = f(x_i + h, y_i + h \cdot k_{1i})\,. \tag{10.7}$$

Ojler-Kromerov metod. Ovaj metod je izuzetno koristan kada se rešavaju diferencijalne jednačine drugog reda. Ove diferencijalne jednačine se svode na sistem od dve diferencijalne jednačine prvog reda, date izrazom koji sledi

$$\begin{aligned} y_{i+1} &= y_i + h \cdot f(x_i, y_i)\,, \\ x_{i+1} &= x_i + h \cdot f(x_i, y_{i+1}), \end{aligned} \tag{10.8}$$

pri čemu se vidi da se za računanje nove vrednosti x koristi nova i vrednost y (markirano žutom bojom), a ne prethodna vrednost, kao što je to slučaj kod standardnog Ojlerovog metoda.

Runge-Kuta metod 4. reda. Najpopularniji metod među Runge-Kuta metodima zasniva se na primeni sledeće formule

$$y_{i+1} = y_i + \frac{1}{6}(F_1 + 2F_2 + 2F_3 + F_4)\,, \tag{10.9}$$

gde su F_1, F_2, F_3 i F_4 dati sledećim izrazima

$$F_1 = h \cdot f(x_i, y_i)\,, \tag{10.10}$$

$$F_2 = h \cdot f\left(x_i + \frac{h}{2}, y_i + \frac{F_1}{2}\right)\,, \tag{10.11}$$

$$F_3 = h \cdot f\left(x_i + \frac{h}{2}, y_i + \frac{F_2}{2}\right)\,, \tag{10.12}$$

$$F_4 = h \cdot f(x_i + h, y_i + F_3)\,. \tag{10.13}$$

Bez obzira na svoju jednostavnost, program prikazan u primeru 10.2. može da posluži kao opšti oblik programa za primenu algoritama za numeričko rešavanje diferencijalnih jednačina, jer se napredniji metodi razlikuju skoro samo po složenosti jednačine koja se javlja u **for** petlji. U narednom odeljku biće predstavljeno kako se mogu napisati python programi za primenu pomenutih numeričkih metoda na neke tipične modele u fizici koji se svode na rešavanje običnih diferencijalnih jednačina (ODJ).

10.3. Primena numeričkih metoda rešavanja ODJ na neke tipične modele u fizici

U ovom odeljku biće predstavljeni python programi kojima se mogu numerički rešiti ODJ koje regulišu ponašanje nekih tipičnih slučajeva u fizici. Konkretno, uporediće se efikasnosti pomenutih metoda na tri slučaja: slobodan pad u vazduhu, RL kolo, kao i linearni harmonijski oscilator.

10.3.1. Slobodan pad u vazduhu

Prvi slučaj koji će u kontekstu primene numeričkih metoda rešavanja ODJ biti razmotren je slobodan pad u vazduhu. U odnosu na slobodan pad u vakuumu, ukoliko se u obzir uzme i sila otpora sredine, rezultujuća sila na telo koje slobodno pada je

$$F = mg - F_{os} = mg - k \cdot v^2\,. \tag{10.14}$$

U opštem slučaju, sila otpora sredine može biti srazmerna brzini ili kvadratu brzine. U slučaju većih brzina, što je slučaj npr. kod slobodnog pada padobranca, sila otpora sredine je srazmerna kvadratu brzine (otuda $-k \cdot v^2$ u izrazu 10.14). Uzimajući u obzir definiciju sile, dolazi se do diferencijalne jednačine koja opisuje ovaj problem

$$F = m\frac{dv}{dt} = mg - k \cdot v^2 \,, \tag{10.15}$$

odnosno

$$\frac{dv}{dt} = g - \frac{k}{m} \cdot v^2 \,. \tag{10.16}$$

Da bi se došlo do izraza koji pokazuje kako se menja vrednost brzine sa vremenom, neophodno je rešiti datu diferencijalnu jednačinu (10.16). Egzaktno rešenje ove jednačine je poznato i dato je sledećim izrazom

$$v = \sqrt{\frac{mg}{k}} \cdot \tanh\left(gt \cdot \sqrt{\frac{k}{mg}}\right) . \tag{10.17}$$

Neka se posmatra slučaj padobranca koji slobodno pada pre nego što ovori padobran. Nakon izvesnog vremena, brzina padobranca će zbog uticaja sile otpora sredine da se ustali na nekoj vrednosti i ta vrednost brzine se često naziva terminalna brzina. Do nje je lako doći, jer je u tom slučaju u izrazu (10.16) član dv/dt jednak nuli, pa se dobija

$$v_T = \sqrt{\frac{mg}{k}} \,. \tag{10.18}$$

Sa izrazom za terminalnu brzinu, egzaktno rešenje dobija oblik

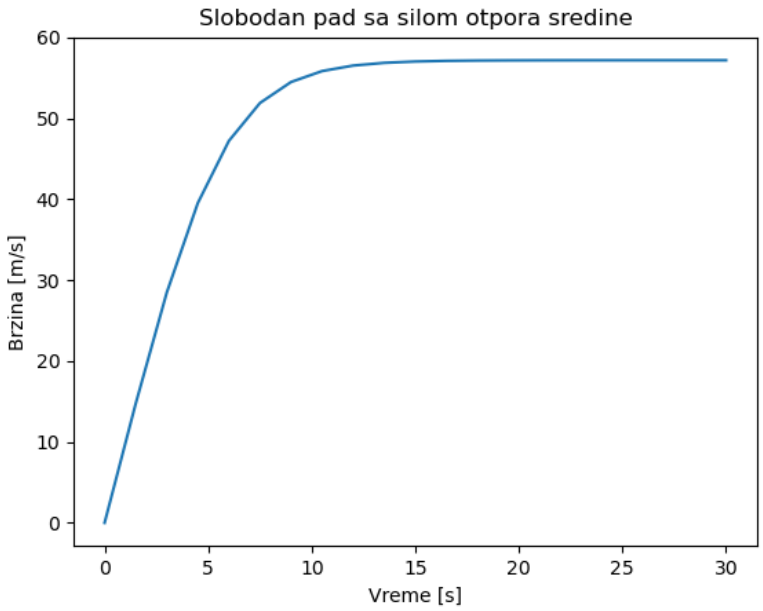
$$v = v_T \cdot \tanh\left(\frac{gt}{v_T}\right) . \tag{10.19}$$

Što se tiče koeficijenta k, pokazano je da se u slučaju padobranca može uzeti vrednost 0,24. Dakle, u slučaju padobranca mase 80 kg, terminalna brzina iznosi oko 57 km/h. Uzimajući pomenute vrednosti za masu i koeficijent otpora sredine, u sledećem primeru biće predstavljen program kojim se numerički rešava diferencijalna jednačina (10.17) primenom Ojlerovog metoda.

Primer 10.3. Primena Ojlerovog metoda na rešavanje diferencijalne jednačine u slučaju slobodnog pada u vazduhu

```
1.  import numpy as np
2.  import matplotlib.pyplot as plt
3.
4.  def dv_dt(v):
5.      return g-(k/m)*v**2
6.
7.  m = 80
8.  k = 0.24
9.  g = 9.81
10. t0 = 0
11. v0 = 0
12. tf = 30
13. n = 21
14.
15. t = np.linspace(t0,tf,n)
16. v = np.zeros(n)
17.
18. h = (tf-t0)/(n-1)
19.
20. t[0] = t0
21. v[0] = v0
22.
23. for i in range (0,n-1):
24.     v[i+1]= v[i] + h * dv_dt(v[i])
25.
26. plt.plot(t,v)
27. plt.title("Slobodan pad sa silom otpora sredine")
28. plt.xlabel("Vreme [s]")
29. plt.ylabel("Brzina [m/s]")
30. plt.show()
```

U suštini, forma programa je identična formi programa iz primera 10.1. Nakon učitavanja biblioteka, u linijama 7 - 13 su definisani početni slovi. U liniji 15 su preko `linspace()` definisane sve vrednosti vremena koje će se uzimati u jednačini Ojlerovog metoda. U liniji 16 su dodeljene nulte vrednosti brzini i ima ih onoliko koliko ima elemenata u listi definisanoj u prethodnoj liniji. Ponovo treba napomenuti da se ove vrednosti mogu uzeti proizvoljno, jer će biti zamenjene pravim vrednostima kroz petlju koja sledi. U linijama 20 i 21 su dodeljene početne vrednosti, dok je glavni deo programa, ***for*** petlja, definisana u linijama 23 i 24. U linijama 26 do 30 su date instrukcije za crtanje. Kao izlaz, program iz poslednjeg primera daje grafik predstavljen na slici 10.1.



Slika 10.1. Grafički prikaz vremenske zavisnosti brzine pri slobodnom padu u vazduhu upotrebom Ojlerovog metoda

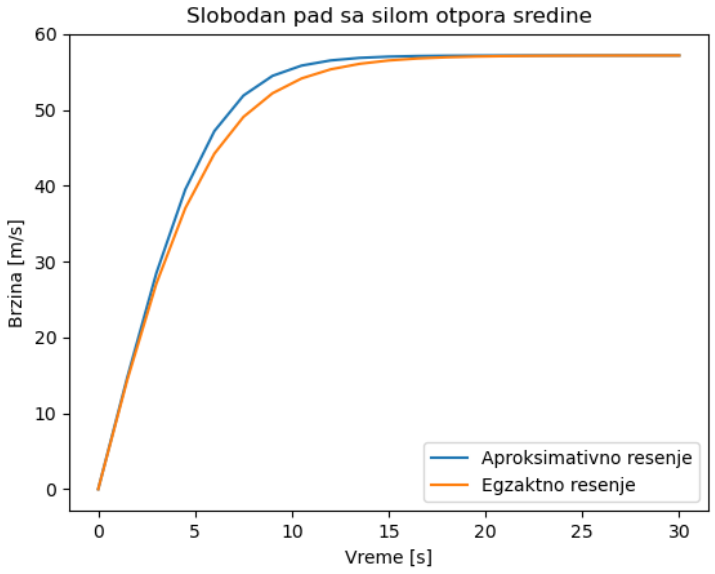
Rezultat predstavljen na slici 10.1. je u skladu sa očekivanjim, jer nakon 10 sekundi dolazi do saturacije vrednosti brzine slobodnog pada u vazduhu. Konkretno, brzina dostiže vrednost od oko 57 m/s, što je u skladu sa rezultatom primene jednačine (10.18). Međutim, postavlja se pitanje da li je i koliko je dobar Ojlerov metod u ovom slučaju, pa je od interesa napisati program koji će porediti aproksimativno i egzaktno rešenje, što je predstavljeno u sledećem primeru.

Primer 10.4. Upoređivanje Ojlerovog i egzaktnog rešenja kod slobodnog pada u vazduhu

```
1.  import numpy as np
2.  import matplotlib.pyplot as plt
3.
4.  def dv_dt(v):
5.      return g-(k/m)*v**2
6.
7.  def v_ex(t):
8.      return np.sqrt((m*g)/k)*np.tanh(g*t*(np.sqrt((k)/(m*g))))
9.
10. m = 80
11. k = 0.24
12. g = 9.81
13. t0 = 0
14. v0 = 0
15. tf = 30
16. n = 21
17.
18. t = np.linspace(t0,tf,n)
19. v = np.zeros(n)
20.
```

```
21. h = (tf-t0)/(n-1)
22.
23. t[0] = t0
24. v[0] = v0
25.
26. for i in range (0,n-1):
27.     v[i+1]= v[i] + h * dv_dt(v[i])
28.
29. v_exact = v_ex(t)
30.
31. plt.plot(t,v,label="Aproksimativno resenje")
32. plt.plot(t,v_exact,label="Egzaktno resenje")
33. plt.title("Slobodan pad sa silom otpora sredine")
34. plt.xlabel("Vreme [s]")
35. plt.ylabel("Brzina [m/s]")
36. plt.legend()
37. plt.show()
```

Struktura programa je identična prethodnim programima, uz dodatak linija kôda za definisanje funkcije egzaktnog rešenja (linije 7 i 8), linije za generisanje egzaktnih rešenja (linija 29), kao i linija za crtanje egzaktnog rešenja i linije za prikazivanje legendi. Kao izlaz, program daje grafik na kome su upoređene vremenske zavisnosti brzine kod slobodnog pada sa uticajem sile otpora sredine, slika 10.2.



Slika 10.2. Poređenje vremenske zavisnosti brzine slobodnog pada Ojlerovim metodom i egzaktnim rešenjem

Na slici 10.2. primetno je malo dostupanje aproksimativnog rešenja od egzaktnog, u oblasti između 3 i 15 sekunde. Međutim, slobodno se može konstatovati da je ovo odstupanje malo, posebno imajući u vidu jednostavnost primenjenog numeričkog metoda. Bolje slaganje bi se dobilo kada bi se smanjila vrednost koraka h, tako što bi se povećala vrednost n. Ostavlja se

kao vežba čitaocu da isproba različite manje vrednosti koraka h kako bi se utvrdila vrednost koja dovodi do zanemarljivog neslaganja sa egzaktnim rešenjem.

U primeru koji sledi, biće predstavljen program kojim se numerički rešava diferencijalna jednačina primenom Ojlerovog metoda i unapređenog Ojlerovog metoda, uz poređenje sa egzaktnim rešenjem.

Primer 10.5. Upoređivanje Ojlerovog metoda i unapređenog Ojlerovog metoda za rešavanje diferencijalne jednačine slobodnog pada u vazduhu

```
1.  import numpy as np
2.  import matplotlib.pyplot as plt
3.
4.  def dv_dt(v):
5.      return g - (k / m) * v**2
6.
7.  def v_ex(t):
8.      return np.sqrt((m*g)/k)*np.tanh(g*t*(np.sqrt(k/(m*g))))
9.
10. m = 80
11. k = 0.24
12. g = 9.81
13. t0 = 0
14. v0 = 0
15. tf = 30
16. n = 16
17.
18. t = np.linspace(t0, tf, n)
19. v_o = np.zeros(n)
20. v_uo = np.zeros(n)
21.
22. h = (tf-t0)/(n-1)
23.
24. t[0] = t0
25. v_o[0] = v0
26. v_uo[0] = v0
27.
28. # for petlja za Ojlerov metod
29. for i in range(0, n-1):
30.     v_o[i+1] = v_o[i] + h * dv_dt(v_o[i])
31.
32. #for petlja za Unapredjeni Ojlerov metod
33. for i in range(0,n-1):
34.     k1i = dv_dt(v_uo[i])
35.     k2i = dv_dt(v_uo[i] + h * k1i)
36.     v_uo[i+1] = v_uo[i] + (h/2) * (k1i + k2i)
37.
38. v_exact = v_ex(t)
39.
40. plt.plot(t, v_exact, label="Egzaktno resenje")
41. plt.plot(t, v_o, label="Aproksimativno resenje - Ojler")
42. plt.plot(t, v_uo, label="Aproksimativno resenje - Unapredjen Ojler")
43. plt.title("Slobodan pad sa silom otpora sredine")
44. plt.xlabel("Vreme [s]")
45. plt.ylabel("Brzina [m/s]")
46. plt.legend()
47. plt.show()
```

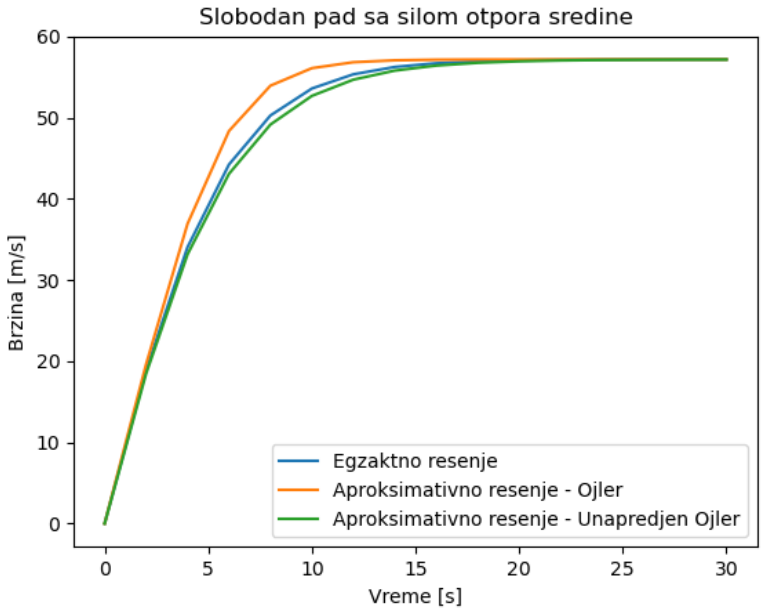
U poslednjem primeru, kôd se od prethodnog primera najviše razlikuje po bloku od linije 33 do linije 36. U tom bloku je definisana ***for*** petlja kojom se računaju vrednosti prema izrazu koji predstavlja unapređeni Ojlerov metod. Pre toga, u linijama 25 i 26, posebno su definisane veličine brzine za metod preko Ojlerovog, `v_o`, i unapređenog Ojlerovog, `v_uo`, metoda. Vrednosti vremena su iste, pa se u oba slučaja koriste vrednosti vremena definisane u liniji 24.

Vezano za blok koji sadrži ***for*** petlju za računanje novih vrednosti prema unapređenom Ojlerovom metodu, posebno je važno naglasiti sledeće. Sa desne strane diferencijalne jednačine koja se rešava (10.16) nalazi se samo zavisno promenljiva – v. Iz tog razloga u ***for*** petlji u liniji 35 za računanje koeficijenta k_{2i} figuriše samo desni deo, tj. samo deo koji se odnosi na y (deo markiran žutom bojom) u relaciji koja sledi

$$k_{2i} = f(x_i + h, y_i + h \cdot k_{1i}) \,. \tag{10.20}$$

Da je kojim slučajem desna strana diferencijalne jednačina funkcija i nezavisno promenljive (u ovom slučaju vremena) i zavisno promenljive, onda bi u liniji 35 figurisala čitava zagrada jednačine (10.20).

Rezultat programa iz poslednjeg primera je grafik na kome se nalaze vremenski zavisne promene brzine dobijenih Ojlerovim metodom, unapređenim Ojlerovim metodom i egzaktnim rešenjem, slika 10.3.



Slika 10.3. Poređenje Ojlerovog, unapređenog Ojlerovog i egzaktnog rešenja

Kako bi se demonstrirala efikasnost unapređenog Ojlerovog metoda, uzeta je mala vrednost parametra n, zbog čega je vrednost koraka h velika. U ovom slučaju rešenje prema Ojlerovom metodu je prilično grubo u odnosu na egzaktno rešenje i odstupanje je jasno vidljivo (slika 10.3.), posebno u oblasti 3. do 15. sekunde. Međutim, čak i za tako grub korak, unapređeni Ojlerov metod daje izuzetan rezultat, koji se dosta dobro poklapa sa egzaktnim rešenjem.

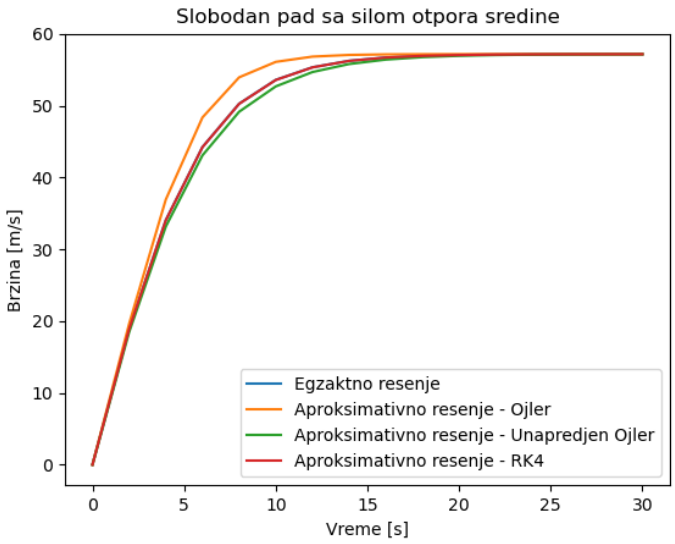
U vezi sa slobodnim padom u vazduhu, biće naveden još jedan program. U sledećem primeru konkretno će biti predstavljen program u kome se numerički rešava diferencijalna jednačina za ovaj model upotrebom Ojlerovog, unapređenog Ojlerovog i Runge-Kuta metodom četvrtog reda, zajedno sa egzaktnim rešenjem.

Primer 10.6. Poređenje Ojlerovog, unapređenog Ojlerovog i Runge-Kuta metoda 4. reda na diferencijalnu jednačinu za slobodan pad u vazduhu

```
1.  import numpy as np
2.  import matplotlib.pyplot as plt
3.
4.  def dv_dt(v):
5.      return g - (k / m) * v**2
6.
7.  def v_ex(t):
8.      return np.sqrt((m*g)/k)*np.tanh(g*t*(np.sqrt(k/(m*g))))
9.
10. m = 80
11. k = 0.24
12. g = 9.81
13. t0 = 0
14. v0 = 0
15. tf = 30
16. n = 16
17.
18. t = np.linspace(t0, tf, n)
19. v_o = np.zeros(n)
20. v_uo = np.zeros(n)
21. v_rk4 = np.zeros(n)
22.
23. h = (tf-t0)/(n-1)
24.
25. t[0] = t0
26. v_o[0] = v0
27. v_uo[0] = v0
28. v_rk4[0] = v0
29.
30. # for petlja za Ojlerov metod
31. for i in range(0, n-1):
32.     v_o[i+1] = v_o[i] + h * dv_dt(v_o[i])
33.
34. #for petlja za Unapredjeni Ojlerov metod
35. for i in range(0,n-1):
36.     k1i = dv_dt(v_uo[i])
37.     k2i = dv_dt(v_uo[i] + h * k1i)
38.     v_uo[i+1] = v_uo[i] + (h/2) * (k1i + k2i)
39.
```

```
40. # for petlja za Runge Kuta metoda 4. reda
41. for i in range(0, n - 1):
42.     F1 = h * dv_dt(v_rk4[i])
43.     F2 = h * dv_dt(v_rk4[i] + F1/2)
44.     F3 = h * dv_dt(v_rk4[i] + F2/2)
45.     F4 = h * dv_dt(v_rk4[i] + F3)
46.     v_rk4[i + 1] = v_rk4[i] + (1 / 6) * (F1 + 2 * F2 + 2 * F3 + F4)
47.
48. v_exact = v_ex(t)
49.
50. plt.plot(t, v_exact, label="Egzaktno resenje")
51. plt.plot(t, v_o, label="Aproksimativno resenje - Ojler")
52. plt.plot(t, v_uo, label="Aproksimativno resenje - Unapredjen Ojler")
53. plt.plot(t, v_rk4, label="Aproksimativno resenje - RK4")
54. plt.title("Slobodan pad sa silom otpora sredine")
55. plt.xlabel("Vreme [s]")
56. plt.ylabel("Brzina [m/s]")
57. plt.legend()
58. plt.show()
```

Forma programa iz poslednjeg primera je ista kao i u prethodnih tri primera. Najveća razlika je u bloku kôda u linijama od 41 do 46, gde je implementirana ***for*** petlja za računanje vrednosti funkcije upotrebom Runge-Kuta metoda 4. reda. Još jednom se napominje da na desnoj strani diferencijalne jednačine (10.16) figuriše samo jedna promenljiva i to zavisna promenljiva, zbog čega u izrazima za računanje koeficijenata $F_1 - F_4$ figuriše samo desna strana zagrada. Rezultat programa iz poslednjeg primera je dat na slici 10.4.

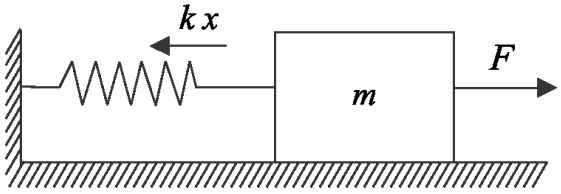


Slika 10.4. Poređenje različitih numeričkih metoda sa egzaktnim rešenjem za diferencijalnu jednačinu slobodnog pada u vazduhu

Rezultat predstavljen na slici 10.4. ukazuje na izuzetnu efikasnost Runge-Kuta metoda 4. reda. Čak i za grubu vrednost koraka h daje rešenje koje se skoro poklapa sa egzaktnim rešenjem. Tek se zumiranjem grafika može videti minimalno odstupanje od egzaktnog rešenja.

10.3.2. Linearni harmonijski oscilator

Harmonijsko oscilovanje je izuzetno bitan koncept koji se koristi u raznim oblastima. U ovom odeljku, biće predstavljeni python programi kojima se može rešiti diferencijalna jednačna koja reguliše ponašanje ovog izuzetno važnog modela. Šema linearnog harmonijskog oscilatora predstavljena na slici 10.5.



Slika 10.5. Ilustracija linearnog harmonijskog oscilatora

Uzimajući u obzir da sila elastičnosti uvek deluje suprotno od smera kretanja tela mase m, i zanemarujući silu trenja, jednačina koja opisuje gore ilustrovani sistem je:

$$F = -k \cdot x\,, \tag{10.21}$$

gde F predstavlja silu koja je telo mase m izvela iz ravnotežnog položaja za vrednost x. Nakon što se sila izrazi preko Drugog Njutnovog zakona, dobija se

$$m \cdot \frac{d^2x}{dt^2} = -k \cdot x\,, \tag{10.22}$$

odnosno

$$\frac{d^2x}{dt^2} + \frac{k}{m} \cdot x = 0\,. \tag{10.23}$$

Dobijena jednačina je poznata diferencijalna jednačine koja opisuje linearno harmonijsko oscilovanje, i čije je opšte egzaktno rešenje poznata jednačina harmonijskog oscilovanja

$$x(t) = A \sin\left(\sqrt{\frac{k}{m}} \cdot t\right) + B \cos\left(\sqrt{\frac{k}{m}} \cdot t\right)\,. \tag{10.24}$$

U poslednjoj se jednačini često uvede i smena za izraz pod korenom, $\omega = \sqrt{k/m}$, nakon čega sledi

$$x(t) = A\sin(\omega \cdot t) + B\cos(\omega \cdot t)\ . \tag{10.25}$$

Ideja za numeričko rešavanje diferencijalnih jednačina drugog reda jeste da se diferencijalna jednačina drugog reda izrazi kao sistem jednačina od dve diferencijalne jednačine prvog reda, pa da se zatim te dve diferencijalne jednačine prvog reda rešavaju simultano nekim od metoda za numeričko rešavanje diferencijalnih jednačina.

Da bi se od diferencijalne jednačine drugog reda došlo do sistema diferencijalnih jednačina prvog reda, uvodi se smena. Pre toga, pogodno je napisati diferencijalnu jednačinu tako da se deo sa izvodom nalazi na jednoj strani

$$\frac{d^2x}{dt^2} = -\frac{k}{m} \cdot x\ . \tag{10.26}$$

Prvi vremenski izvod promenljive x (dx/dt) se može napisati kao smena. Pošto se zna da je to brzina, diferencijalna jednačina (10.26) se svodi na sledeći sistem jednačina

$$\begin{aligned} \frac{dx}{dt} &= v\ , \\ \frac{dv}{dt} &= -\frac{k}{m} \cdot x\ . \end{aligned} \tag{10.27}$$

Kao što se može videti, odgovarajućom smenom dobijene su dve diferencijalne jednačine koje se simultano mogu rešavati kako bi se dobilo aproksimativno rešenje. Za njihovo rešavanje se može primeniti Ojlerov metod, a ovako dobijene jednačine se takođe nazivaju i spregnute/uparene jednačine, jer su međusobno zavisne.

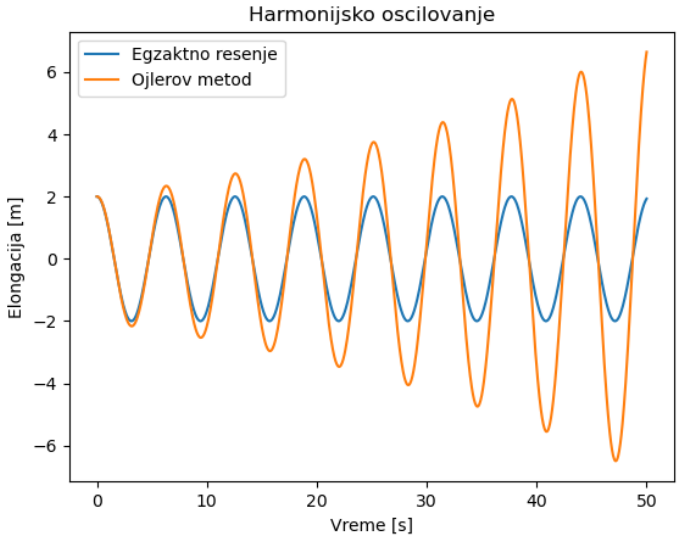
U primeru koji sledi, najpre će biti predstavljen python program za simultano rešavanje sistema jednačina (10.27) upotrebom Ojlerovog metoda. U primeru se uzima da masa tela koje osciluje iznosi 1 kg, dok je konstanta elastičnosti opruge 1. Smatra se da je telo izvedeno iz ravnotežnog položaja duž x ose u iznosu od 2 m. U datom slučaju, egzaktno rešenje je

$$x = 2 \cdot \cos\left(\sqrt{\frac{1}{1}} \cdot t\right)\ . \tag{10.28}$$

Primer 10.7. Program za rešavanje diferencijalne jednačine LHO Ojlerovim metodom

```
1.  import numpy as np
2.  import matplotlib.pyplot as plt
3.
4.  def dx_dt(v):
5.      return v
6.
7.  def dv_dt(x):
8.      return -(k/m)*x
9.
10. def x_ex(t):
11.     return 2*np.cos(np.sqrt(k/m)*t)
12.
13. k = 1
14. m = 1
15. x0 = 2
16. v0 = 0
17. t0 = 0
18. tf = 50
19. n = 1001
20. dt = (tf-t0)/(n-1)
21.
22. t = np.linspace(t0,tf,n)
23. x = np.zeros(n)
24. v = np.zeros(n)
25.
26. x[0] = x0
27. v[0] = v0
28.
29. for i in range(0,n-1):
30.     x[i+1] = x[i] + dt * dx_dt(v[i])
31.     v[i+1] = v[i] + dt * dv_dt(x[i])
32.
33. x_exact = x_ex(t)
34.
35. plt.plot(t,x_exact,label='Egzaktno resenje')
36. plt.plot(t,x,label='Ojlerov metod')
37. plt.title("Harmonijsko oscilovanje")
38. plt.xlabel("Vreme [s]")
39. plt.ylabel("Elongacija [m]")
40. plt.legend()
41. plt.show()
```

Kao i u prethodnim slučajevima, koristi se ista konstrukcija programa - nakon definisanja početnih uslova i inicijalnih vrednosti, sledi **for** petlja u okviru koje se računaju nove vrednosti u skladu sa nekom od numeričkih metoda. U konkretnom primeru, **for** petlja se nalazi u linijama 29-31 i ovoga puta se u njoj simulatno računaju dve vrednosti, jer je neophodno rešiti dve jednačine. Sve ostalo je manje-više isto i ne zahteva posebno komentarisanje. Rezultat programa u poslednjem primeru je dat na slici 10.6.



Slika 10.6. Grafički prikaz vremenske zavisnosti elongacije - egzaktno i numeričko rešenje preko Ojlerovog metoda

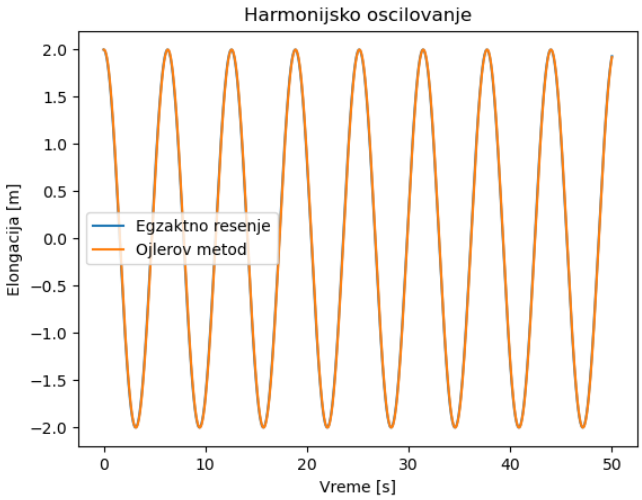
Rezultat prikazan na slici 10.6. ukazuje na veliko neslaganje između egzaktnog rešenja i aproksimativnog rešenja dobijenog primenom Ojlerovog metoda. Problem nije samo odstupanje od tačnih vrednost, nego se radi se o fundamentalnom neslaganju, jer pomenuta aproksimacija daje potpuno pogrešnu fizičku sliku. Naime, nakon što se sistem prepusti oscilovanju, sa povećanjem vremena dolazi do povećanja elongacije. S obzirom da se sistemu ne saopštava dodatna energija, ispada da sistem ni iz čega povećava svoju energiju do beskonačnosti. Odstupanje od tačnih vrednosti može malo da se ublaži smanjenjem koraka h, ali je to samo ublaženje, jer će se za dovoljno velike vrednosti vremena uvek javljati fizički neispravan rezultat.

Dakle, zaključuje se da Ojlerov metod nije dobar za slučaj LHO. Ojlerov metod zapravo generalno nije dobar za periodične funkcije i funkcije čije se vrednosti brzo menjaju. Međutim, nikako ne treba zaboraviti jednostavnost Ojlerovog metoda i činjenicu da za ogroman broj tipičnih modela daje odlične rezultate.

Kako bi se rešila situacija sa LHO, povoljno je iskoristiti tzv. Ojler-Kromerov metod, koji se zasniva na jednoj maloj modifikaciji Ojlerovog metoda. Prema (10.8), u konkretnom slučaju LHO, Ojler-Kromerov metod se sastoji u tome da se vrednosti v_{i+1} računaju tako što se ne koristi prethodna vrednost x_i, nego nova vrednost x_{i+1}. Kako bi se od poslednjeg programa dobio program koji implementira Ojler-Kromerov metod, u liniji 31 neophodno je uvesti samo jednu malu modifikaciju. Konkretno, blok kôda treba da bude sledeći (promena je naznačena žutom bojom)

```
29. for i in range(0,n-1):
30.     x[i+1] = x[i] + dt * dx_dt(v[i])
31.     v[i+1] = v[i] + dt * dv_dt(x[i+1])
```

Nakon ove jednostavne modifikacije, dobija se rezultat kao na slici 10.7.

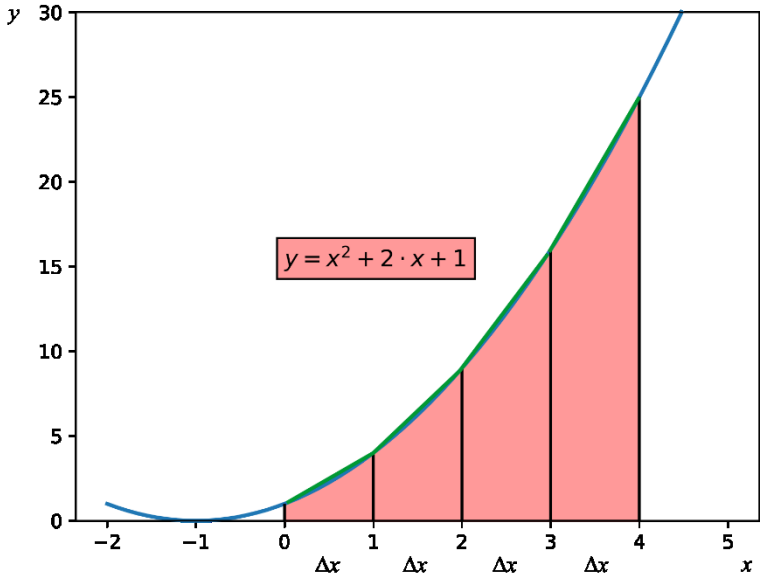


Slika 10.7. Numeričko rešavanje diferencijalne jednačine LHO Ojler-Kromer metodom

Rezultat prikazan na slici 10.7. ukazuje da Ojler-Kromerov metod daje fizički adekvatan rezultat. U konkretnom programu je iskorišćena mala vrednost koraka h, a nešto veće vrednosti koraka h bi možda dovele do eventualnog odstupanja od egzaktnog rešenja, međutim nikako ne bi dovele do toga da se elongacija povećava bez ulaganja energije u sistem.

10.4. Numeričko rešavanje integrala trapeznom formulom

Poput diferencijalnih jednačina, rešavanje integrala je jedna od glavnih aktivnosti u modelovanju različitih procesa. Postoji mnogo pristupa koji se mogu iskoristiti za numeričko rešavanje integrala, a neki od najpoznatijih su upotrebom trapezne i Simpsonove formule. U oba slučaja, ideja je da se površina ispod krive podeli na oblasti, a da se zatim na neki način izračunaju njihove površine, kao što je to ilustrovano na slici 10.8.



Slika 10.8. Podela površine ispod krive na intervale duž x-ose

Na slici 10.8. je grafički prikazana funkcija $y = x^2 + 2x + 1$ u intervalu od -2 do 5. Potrebno je izračunati površinu ispod krive markiranu crvenom bojom. Vidi se da je kriva markirana u intervalu x od 0 do 4, što znači da je tražena površina ispod krive data sledećim integralom

$$S = I = \int_0^4 (x^2 + 2x + 1)dx \ . \tag{10.29}$$

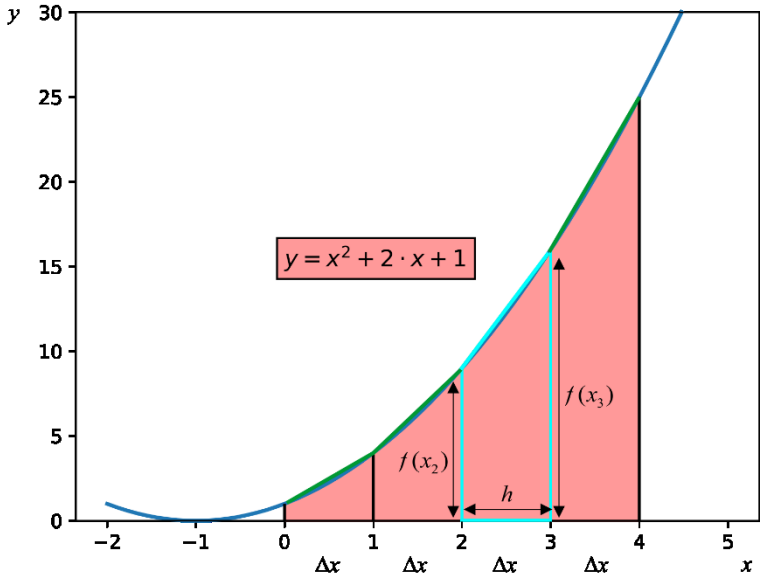
Površina ispod krive je duž x-ose izdeljena na tačno četiri jednaka intervala. Jedan od najpoznatijih metoda za numeričko rešavanje integrala jeste primenom trapezne formule. Kod ovog pristupa prvi korak je da se oblast duž x-ose ispod krive izdeli na n jednakih intervala, i ta se vrednost često naziva korak h. Zatim se svaka oblast dobijena takvom fragmentacijom tretira kao jedan pravougli trapez, kod kojeg je korak h praktično visina trapeza. Znajući da je površina pravouglog trapeza data sledećim izrazom

$$P = \frac{a+b}{2} \cdot h \, , \tag{10.30}$$

površina ispod krive, tj. vrednost integrala I, se može izraziti formulom

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n-1} \frac{f(x_i) + f(x_{i+1})}{2} \cdot h \ . \tag{10.31}$$

Primena trapezne formule biće vizuelno objašnjena još i preko slike 10.9.



Slika 10.9. Objašnjenje trapezoidnog pravila

Na slici 10.9. je interval od 0 do 4 duž x-ose podeljen na četiri podintervala. Svaki od tih podintervala sa datom krivom praktično konstituiše po jedan pravougli trapez čija se površina može izračunati preko izraza (10.30). Ako se uoči jedan konkretan trapez sa slike (10.9), uokviren svetlo plavim linijama, vidi se da je njegova površina $\big(f(x_2)+f(x_3)\big)\cdot h/2$. Kada je $n=4$, u konkretnom slučaju $h=1$, pa je primena (10.31) sledeća

$$\int_0^4 (x^2+2x+1)dx \approx \sum_{i=0}^{4-1} \frac{f(x_i)+f(x_{i+1})}{2}\cdot h\,,$$

$$\big(f(x_0)+f(x_{0+1})\big)\cdot\frac{1}{2}=\big(f(x_0)+f(x_1)\big)\cdot\frac{1}{2}=\big(f(0)+f(1)\big)\cdot\frac{1}{2}=(1+4)\cdot\frac{1}{2}=2{,}5\,,$$

$$\big(f(x_1)+f(x_{1+1})\big)\cdot\frac{1}{2}=\big(f(x_1)+f(x_2)\big)\cdot\frac{1}{2}=\big(f(1)+f(2)\big)\cdot\frac{1}{2}=(4+9)\cdot\frac{1}{2}=6{,}5\,,$$

$$\big(f(x_2)+f(x_{2+1})\big)\cdot\frac{1}{2}=\big(f(x_2)+f(x_3)\big)\cdot\frac{1}{2}=\big(f(2)+f(3)\big)\cdot\frac{1}{2}=(9+16)\cdot\frac{1}{2}=12{,}5\,,$$

$$\big(f(x_3)+f(x_{3+1})\big)\cdot\frac{1}{2}=\big(f(x_3)+f(x_4)\big)\cdot\frac{1}{2}=\big(f(3)+f(4)\big)\cdot\frac{1}{2}=(16+25)\cdot\frac{1}{2}=20{,}5\,,$$

$$\int_0^4 (x^2+2x+1)dx \approx 2+6{,}5+12{,}5+18=42{,}0\,.$$

Egzaktno rešenje ovog integrala je 41,33, tako da je dobijeni rezultat odlična aproksimacija, posebno kada se uzme u obzir koliko je velika vrednost koraka. Naravno, i sam integral je jednostavan, pa je dobro slaganje bilo očekivano.

Što je manja vrednost koraka h, numerička vrednost integrala će biti tačnija. Iz tog razloga je „ručna“ primena trapezne formule dugotrajna, zbog čega je zgodno napisati odgovarajući program. U primeru koji sledi biće predstavljen program za numeričko računanje vrednosti određenog integrala datog u izrazu (10.29).

Primer 10.8. Program za rešenje integrala $\int_0^4 (x^2 + 2x + 1)dx$ trapeznom formulom

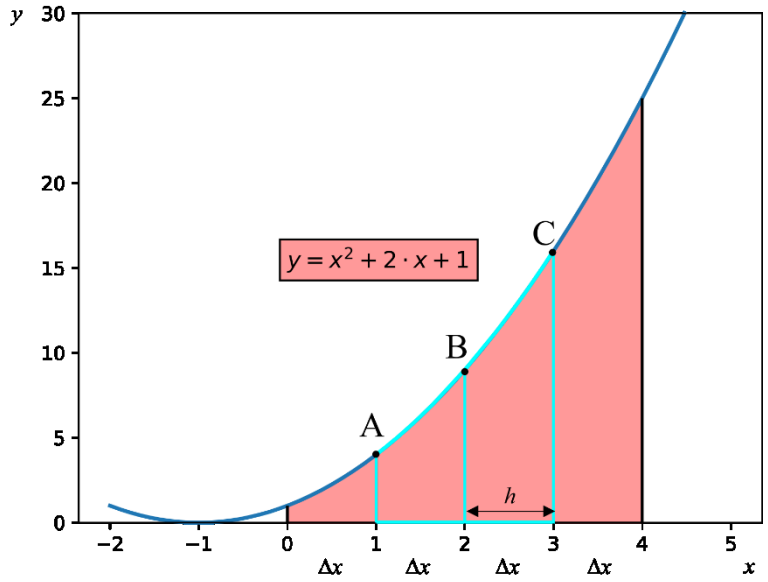
```
1.  def f(x):
2.      return x**2+2*x+1
3.
4.  a = 0
5.  b = 4
6.  n = 4
7.
8.  h = (b-a)/n
9.
10. integral = 0
11.
12. for i in range(0,n):
13.     x_pocetno = a + i * h
14.     x_krajnje = a + (i + 1) * h
15.     integral = integral + (f(x_pocetno) + f(x_krajnje)) * h/2
16.
17. print("Numericka vrednost integrala:", integral)
>> 42.0
```

U datom primeru najpre je definisana funkcija koja se integrali, a nakon toga su definisane i granice u kojima se integrali. n je vrednost na koliko se podintervala deli interval duž x-ose, tako da se korak (odnosno visina svakog od trapeza) računa kao u liniji 8. Nakon toga se definiše početna vrednost integrala koja će se kasnije „dopunjavati“ tokom ***for*** petlje. Kao i kod numeričkog rešavanja diferencijalnih jednačina, ***for*** petlja u kojoj se primenjuje neka konkretna formula za računanje vrednosti je glavni deo programa.

Suština formule (10.31) jeste da se u svakom fragmentu površine trapeza računaju tako što se određuje vrednost funkcije y u tačkama koje su temena trapeza duž x-ose. Tačka koja označava teme sa leve strane posmatranog trapeze duž x-ose je u kôdu nazvana `x_pocetno`, dok je tačka koja označava teme sa desne strane posmatranog trapeza u kôdu označena sa `x_krajnje`. U ***for*** petlji u linijama 13 i 14 su date instrukcije kako se ove vrednosti određuju. U prvoj iteraciji $i = 0$, pa je $x_{pocetno} = 0 + 0 \cdot h$, dok je $x_{krajnje} = 0 + 1 \cdot h$, i tako dalje. Program daje rezultat 42,0, a povećanjem parametra n na vrednost od 81, dobija se 41,33.

10.5. Numeričko rešavanje integrala Simpsonovom formulom

Ideja Simpsonovog metoda je slična – podeliti površinu ispod krive na jednake delove duž x-ose, a zatim na neki način odrediti površine dobijenih fragmenata. Kod metoda primenom trapezne formule, deo posmatrane krive je posmatran kao deo trapeza, dok je površina dobijenih fragmenata posmatrana kao površina trapeza, što nije loša aproksimacija za mali korak. Međutim, kod metoda na bazi Simpsonove formule, kriva svakog od fragmenata se aproksimira kvadratnom funkcijom, kao što je to naznačeno na slici 10.10.



Slika 10.10. Aproksimiranje krive kvadratnom funkcijom

Na slici 10.10. kvadratna funkcija kojom se aproksimira kriva od $x = 1$ do $x = 3$ je određena tačkama A, B i C, koje je lako odrediti jer su to zapravo vrednosti funkcije $y(x_1)$, $y(x_2)$ i $y(x_3)$, redom. Zatim se na isti način aproksimira kriva sledeća tri fragmenta i tako se postupak odvija do kraja, uz napomenu da je neophodno da n bude paran broj. Suštinski, opisani metod se zasniva na primeni sledeće (Simpsonove) formule

$$\int_a^b f(x)dx \approx [f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \cdots$$
$$\ldots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)]\frac{\Delta x}{3}\,. \qquad (10.32)$$

Formula (10.32) se može malo pregrupisati kako bi se dobio izraz koji se lakše pamti. Konkretno, vrednosti funkcije u početnoj i krajnjoj tački *x* se mogu postaviti na početak, a primećuje se da je koeficijent 4 ispred članova gde figurišu vrednosti *x* sa neparnim indeksima, dok je koeficijent 2 ispred članova gde figurišu vrednosti *x* sa parnim indeksima. Tada (10.32) dobija sledeći oblik

$$\int_a^b f(x)dx \approx [f(x_0) + f(x_n) + 4 \cdot (f(x_1) + 4f(x_3) + \cdots) + \\ +2 \cdot (f(x_2) + f(x_4) + \cdots)] \cdot \frac{\Delta x}{3} . \tag{10.33}$$

Primena (10.33) će biti predstavljena na primeru iz prethodnog odeljka, kako bi se moglo izvršiti poređenje. U tom slučaju, primena (10.33) na (10.29) daje

$$\int_0^4 (x^2 + 2x + 1)dx \approx \left[f(x_0) + f(x_4) + 4 \cdot \left(f(x_1) + f(x_3)\right) + 2 \cdot f(x_2)\right] \cdot \frac{1}{3} = \\ = \left[f(0) + f(4) + 4 \cdot \left(f(1) + f(3)\right) + 2 \cdot f(2)\right] \cdot \frac{1}{3} = \\ = [1 + 25 + 4 \cdot (4 + 16) + 2 \cdot 9] \cdot \frac{1}{3} = \frac{124}{3} = 41.33 . \tag{10.34}$$

Zanimljivo je da se primenom Simpsonove formule u ovom slučaju dobija praktično ista vrednost kao i egzaktno rešenje, $124/3 \approx 41{,}33$, a isto se koristi vrednost parametra n od samo 4. Podsećanja radi, trapezna formula je pod istim uslovima dala vrednost od 42. Naravno, olakšavajuća okolnost je i ta što je sam integral lak, a i suštinski je podintegralna funkcija upravo kvadratna funkcija, što je maksimalno kompatibilno sa Simpsonovom formulom. Naravno, u slučaju težih integrala, neophodno je uzeti manji korak, a to dovodi do potrebe za pisanjem programa. U sledećem primeru biće predstavljen python program za rešenje prethodnog integrala.

Primer 10.9. Program za rešavanje integrala $\int_0^4 (x^2 + 2x + 1)dx$ Simpsonovom formulom

```
1.  import numpy as np
2.
3.  def f(x):
4.      return x**2 + 2*x +1
5.
6.  a = 0
7.  b = 4
8.  n = 4
9.
10. h = (b-a)/n
11.
12. x = np.linspace(a,b,n+1)
13.
14. suma = 0
15. x0 = x[0]
16. xn = x[-1]
17. for i in range(1,n):
18.     if i%2 == 0:
19.         suma = suma + 2 * f(x[i])
20.     else:
21.         suma = suma + 4 * f(x[i])
22.
23. integral = (h/3) * (f(x0) + f(xn) + suma)
24.
25. print("Numericka vrednost integrala:", integral)
>> 41.3333333
```

Razlika u odnosu na formu programa iz prethodnog primera je što je u ovom slučaju pogodnije prvo definisati vrednosti *x* (linija 12) koje će kasnije u ***for*** petlji biti korišćene za izračunavanje vrednosti funkcije. Nakon vrednosti *x* definisana je i veličina `suma` (linija 14), čija je početna vrednost 0. Vrednost ove veličine će biti dopunjavana tokom ***for*** petlje. U samoj ***for*** petlji, nalazi se kondicional *if*, koji proverava da li je brojač *i* (tačnije, indeks vrednosti *x*) paran ili neparan. Ukoliko je paran, veličini `suma` se dodaje vrednost prema delu izraza gde figurišu parni indeksi vrednosti *x*, dok se u suprotnom veličini `suma` dodaje vrednost prema delu izraza gde figurišu neparni indeksi vrednosti *x*. Konačno, kada se sakupe sve vrednosti veličine `suma`, vrednost integrala se može izračunati preko linije 23. Kao što je već pomenuto, u konkretnom slučaju, dobija se rezultat 41,33.

10.6. Numeričko rešavanje diferencijalnih jednačina i integrala upotrebom gotovih funkcija

U prethodnim odeljcima bilo je predstavljeno kako se mogu napisati python programi za implementaciju nekih poznatih metoda za numeričko rešavanje diferencijalnih jednačina i integrala. Međutim, programski jezik python raspolaže i sa bibliotekama koje poseduju gotove

funkcije kojima se numerički mogu rešiti diferencijalne jednačine i integrali upotrebom najsavremenijih numeričkih metoda. Primena ovih funkcija u mnogome olakšava numeričko rešavanje i modelovanje.

Najpoznatija biblioteka sa funkcijama za numeričko rešavanje diferencijalnih jednačina i integrala je biblioteka **scipy**. Konkretno, modul ove biblioteke pod nazivom **integrate** poseduje funkcije `odeint()` i `quad()` kojim se numerički rešavaju diferencijalne jednačine i integrali, redom.

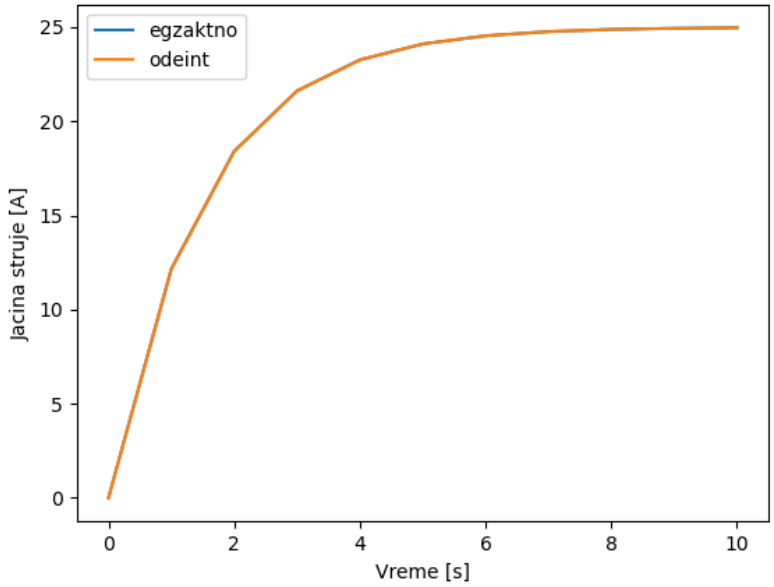
Najpre će biti predstavljena funkcija za numeričko rešavanje diferencijalnih jednačina, `odeint()`. Ova funkcija ima tri glavna argumenta - desna strana jednakosti diferencijalne jednačine, početna vrednost zavisno promenljive i vrednosti nezavisno promenljive. Primena `odeint()` funkcije za rešavanje diferencijalne jednačine koja opisuje vremensku zavisnost jačine struje kod RL kola je predstavljena u sledećem primeru.

Primer 10.10. Primena `odeint()` za rešavanje diferencijalne jednačine RL kola

```
1.  import numpy as np
2.  import matplotlib.pyplot as plt
3.  from scipy.integrate import odeint
4.
5.  def f(i,t):
6.      return (1/L)*(V-i*R)
7.
8.  def i_ex(t):
9.      return (V/R)*(1-np.exp(-(R/L)*t))
10.
11. V = 50 # V
12. R = 2 # Ohm
13. L = 3 # H
14. i0 = 0
15. t0 = 0
16. tf = 10
17. n = 11
18.
19. t = np.linspace(t0,tf,n)
20.
21. i_egzaktno = i_ex(t)
22. i_odeint = odeint(f,i0,t)
23.
24. plt.plot(t,i_egzaktno,label='egzaktno')
25. plt.plot(t,i_odeint,label='odeint')
26. plt.xlabel('Vreme [s]')
27. plt.ylabel('Jacina struje [A]')
28. plt.legend()
29. plt.show()
```

U 3. liniji programa predstavljenog u primeru 10.10. je učitana funkcija `odeint()` iz **scipy** biblioteke. U linijama 5 i 6 je definisana funkcija koja predstavlja desnu stranu jednakosti

diferencijalne jednačine koja opisuje RL kolo i koja se numerički rešava. U linijama 8 i 9 je definisano egzaktno rešenje pomenute diferencijalne jednačine. U linijama od 11 do 17 su definisani početni uslovi, dok su u liniji 19 definisane sve vrednosti vremena koje će se koristiti u proračunu. U liniji 21 su definisane vrednosti jačine struje prema egzaktnom rešenju, dok je linija 22 glavna za ovaj program. U njoj su definisane vrednosti jačine struje prema metodu implementiranog u gotovoj funkciji **odeint()**. Dalje slede instrukcije za crtanje egzaktnog i numeričkog rešenja, redom. Rezultat je grafik predstavljen na slici 10.11.



Slika 10.11. Egzaktno i `odeint()` numeričko rešenje diferencijalne jednačine za RL kolo

Rezultat prikazan na slici 10.11. govori o izuzetnoj efikasnosti `odeint()` funkcije, s obzirom da se numeričko rešenje skoro i ne razlikuje od egzaktnog, iako je vrednost koraka prilično velika.

U nastavku, sledi predstavljanje funkcije iz **scipy** biblioteke, `quad()`, koja se koristi za numeričko rešavanje integrala. Argumenti ove funkcije su, redom, funkcija koja se integrali, donja granica integraljenja, gornja granica integraljenja. Kao i kod slučajeva sa primenom trapezne i Simpsonove formule, i ova funkcija će u sledećem primeru biti demonstrirana na integralu datom u izrazu (10.29).

Primer 10.11. Program za rešenje integrala $\int_0^4 (x^2 + 2x + 1)dx$ funkcijom `quad()`

```
1.  import numpy as np
2.  import matplotlib.pyplot as plt
3.  from scipy.integrate import quad
4.
5.  def f(x):
6.      return x**2 + 2*x +1
7.
8.  a = 0
9.  b = 4
10.
11. integral = quad(f,a,b)
12.
13. print(integral)
>> (41.33333333333333, 4.588921835117313e-13)
```

U programu predstavljenom u primeru 10.11. u liniji 3 je učitana funkcija `quad()` koja će se koristiti za numeričko rešavanje pomenutog integrala. U linijama 5 i 6 je definisana podintegralna funkcija, dok su granice integraljenja definisane u linijama 8 i 9. Vrednost integrala se računa u liniji 11, gde je upotrebljena funkcija `quad()`, dok se u liniji 13 traži ispisivanje numeričke vrednosti integrala. Napomena je da funkcija `quad()` generiše kao izlaz niz sa dva numerička elementa, od kojih je prvi aproksimativna vrednost integrala, dok je druga vrednost procena greške.

Modul **integrate** iz biblioteke **scipy** ima još funkcija na raspolaganju za numeričko rešavanje integrala, od kojih su neke:

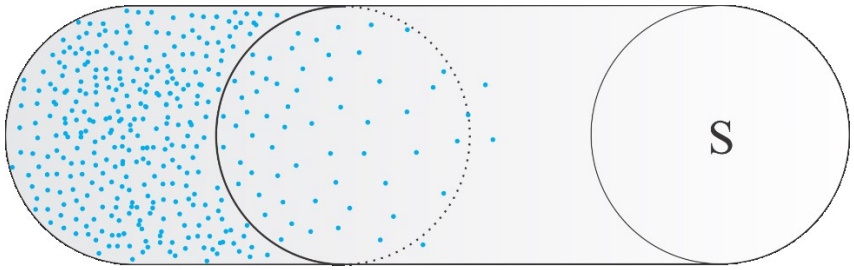
- `dblquad()` - za numeričko rešavanje dvostrukih integrala,
- `tplquad()` - za numeričko rešavanje trostrukih integrala,
- `romberg()` - za numeričko rešavanje integrala Rombergovom metodom,
- `trapz()` - za numeričko rešavanje integrala pravilom trapeza,
- `simpson()` - za numeričko rešavanje integrala Simpsonovim pravilom.

11. Slučajni brojevi i Monte Karlo metod

U opštem slučaju, modelovanje fizičkih procesa i sistema može biti determinističko i stohastičko. Osnovna odlika determinističkog modelovanja je da svaki korak simulacije zavisi od prethodnog koraka (tačnije rečeno, potpuno je određen prethodnim korakom). Rezultat determinističkog modelovanja je uvek isti izlaz, ukoliko je i skup ulaznih parametara isti. Sa

druge strane, stohastičko modelovanje se zasniva na slučajnim procesima, a takvi sistemi se nazivaju slučajni sistemi. Kod modelovanja slučajnih procesa, izlaz je uvek različit, za isti skup ulaznih parametara, sa tim da se rezultat uvek nalazi oko neke srednje vrednosti. Još jedna odlika slučajnih sistema jeste da se ne zna kada će se događaj u njemu odviti, ali se zna verovatnoća za odigravanje događaja. Takvi sistemi se nazivaju slučajni (nasumični) ili stohastički sistemi.

Primer jednog stohastičkog procesa je difuzija. U opštem slučaju, difuzija je mehanizam prenosa materije kroz materiju. Ovaj proces je pogodno ilustrovati primerom kretanja kontrastne boje kroz vodu. Ukoliko se, na primer, kapne nekoliko kapljica boje za kolače u čašu vode primetiće se da boja počinje polako da se prostire kroz vodu. Prostiranje boje kroz vodu može potrajati prilično dugo, reda veličine nekoliko sati, a krajnji rezultat će biti jednako raspodeljena boja. U opštem slučaju difuzija je proces koji zavisi od koncentracije. Supstanca koja difunduje se kreće iz oblasti više koncentracije u oblast niže koncentracije, kao što je to ilustrovano na Slici 11.1.



Slika 11.1. Difuzija molekula u cevi poprečnog preseka *S*

Proces difuzije se može posmatrati na sledeći način. Boja koja se kapne u čašu vode se sastoji od velikog broja molekula boje. Ako bi se posmatrao samo jedan molekul boje, uočilo bi se da je njegova putanja tokom mešanja sa vodom izuzetno kompleksna. Naime, molekul se kreće jednim pravcem, onda se sudara sa molekulom vode, pa nastavlja u nekom drugom pravcu, pa se opet sudara sa nekim drugim molekulom, i tako dalje. Teorijski gledano, za svaki molekul boje mogla bi da se napiše jednačina kretanja. To bi zapravo dovelo do ogromnog broja diferencijalnih jednačina koje bi u principu mogle da se reše. Međutim, takav zadatak bi bio nerešiv sa stanovišta dostupnih kompjuterskih resursa, zbog ogromnog broja jednačina koje bi morale da se reše. Ovakav pristup nije od praktične koristi i ne bi omogućio predviđanje vremena mešanja nekih drugi količina boje u vodi.

U vezi sa gore navedenim primerom difuzije boje kroz vodu, važno je napomenuti dve stvari. Prvo, ako bi se eksperiment ponovio više puta, sa uvek istom količinom boje i vode, vreme neophodno da se boja ravnomerno raspodeli u čaši vode bi za svaki eksperiment bila otprilike ista i rezultat bi bi bio uvek isti – ravnomerno raspodeljena boja u čaši vode. Sa druge strane, mešanje boje i vode bi se uvek odigravalo drugačije, u smislu da bi u svakom eksperimentu molekuli boje uvek imali drugačije trajektorije. Praktično je nemoguće predvideti kretanje svakog molekula ponaosob, drugim rečima molekuli se kreću nasumično.

Za rešenje ovakvih problema detaljno poznavanje trajektorija molekula ne igra ulogu. U ovakvom slučaju je neophodan statistički opis, tj. dovoljno je poznavanje usrednjenih svojstava trajektorije molekula.

Komplikovane trajektorije molekula boje se mogu modelovati tzv. ***slučajnim hodom***. Slučajni hod je proces u kome se „šetač“ (čestica, molekul,...) pomera jedan po jedan korak u skladu sa određenim pravilima. Sudaranje molekula boje sa molekulima vode, i posledična promena pravca kretanja, se modeluje tako što se „šetaču“ omogućava da nakon svakog koraka nasumično promeni pravac šetanja. Da bi se demonstrirao i modelovao slučajan hod, neophodno je biti u mogućnosti generisati slučajne brojeve. Zato je sledeći korak upoznati se sa mogućnostima generisanja slučajnih brojeva u pythonu.

11.1. Python i generatori nasumičnih brojeva

Alati kojima se dobijaju slučajni brojevi nazivaju se *generatori slučajnih brojeva*. Generator slučajnih brojeva je sistem koji generiše slučajne brojeve iz pravog izvora slučajnosti (nasumičnosti). Tipičan primer pravog izvora slučajnosti je Gajger-Milerov brojač, ili kockica. Međutim, ne bi bilo praktično koristiti Gajger-Milerov brojač svaki put kada nam zatreba generisanje slučajnog broja. Umesto toga možemo koristiti generatore *pseudoslučajnih brojeva*.

Generatori pseudoslučajnih brojeva su generatori koji za generisanje slučajnih brojeva koriste deterministički proces. U suštini, to su funkcije koje su izuzetno osetljive na vrednosti argumenta, pa čak i izuzetno mala promena vrednosti argumenta dovodi do ogromne razlike u vrednosti funkcije.

Pythonova standardna biblioteka za generisanje slučajnih[7] brojeva je **random** i ona sadrži određene funkcije kojima se generišu slučajni brojevi. S obzirom da je to deterministički način generisanja, to znači da se mora obezbediti ulazni parametar na osnovu koga će generator generisati slučajan broj. U slučaju generatora slučajnih brojeva, ulazni parametar za funkciju kojom se simulira dobijanje slučajnih brojeva je vreme računara izraženo u milisekundama. Vreme u milisekundama se izražava kao argument funkcije `seed()`. Vreme izraženo u milisekundama je odličan ulazni parametar, jer se stalno i brzo menja, pa u svakom trenutku daje različite vrednosti funkcije.

Ukoliko se kao argument funkcije `seed()` uvek dâ isti broj, generisani niz slučajnih brojeva će uvek biti isti. Korišćenje funkcije `seed()` je korisno kada je od interesa dobiti ponovljive rezultate generisanja slučajnih brojeva, na primer u situacijama kao što su testiranje ili reprodukcija rezultata. Ukoliko se u kôdu ne definiše vrednost funkcije `seed()`, ili se uopšte ni ne pomene, python će kao ulazni parametar uzeti trenutno vreme na kompjuteru izraženo u milisekundama. U primerima koji slede biće predstavljeno nekoliko programa koji koriste mogućnosti biblioteke **random**.

Primer 11.1. Generisanje slučajnih brojeva sa ili bez funkcije `seed()`

```
1.  from random import seed
2.  from random import random
3.
4.  seed(1)
5.
6.  print(random())
>> 0.13436424411240122
```

Dati program će kao izlaz uvek dati istu vrednost, konkretno 0.13436424411240122. Ukoliko bi se uklonila linija 4, ili ako bi se zagrada kod funkcije `seed()` ostavila prazna, kao ulazni parametar za generisanje slučajnih brojeva bi se uzelo trenutno vreme kompjutera izraženo u milisekundama i program bi uvek davao drugačiju vrednost slučajnog broja.

U slučaju da treba da se generiše više slučajnih brojeva koji bi se smestili u jednu listu, korisno je iskoristiti mogućnosti ***for*** petlje, na način predstavljen u sledećem primeru.

[7] nadalje će se pisati slučajnih, iako se radi o pseudoslučajnim, u skladu sa literaturom.

Primer 11.2. Generisanje slučajnih brojeva i smeštanje u listu

```
1.  from random import seed
2.  from random import random
3.
4.  slucajni_brojevi = []
5.
6.  for i in range(10):
7.      vrednost = random()
8.      slucajni_brojevi.append(vrednost)
9.
10. print(slucajni_brojevi)
>>     [0.3033685109329176,     0.5875806061435594,     0.8824790008318577,
0.8461974184283128,         0.5052838205796004,         0.5890022579825517,
0.034525830151341586,        0.24273997354306764,        0.7974042475543028,
0.41431399930077743]
```

Nakon učitavanja funkcija iz biblioteke **random**, definisana je u liniji 4 jedna prazna lista koja će sadržavati sve slučajne brojeve koji budu generisanu u okviru ***for*** petlje koja sledi. U liniji 8 slučajni brojevi generisani u liniji 7 se smeštaju u listu `slučajni_brojevi` upotrebom funkcije `append()`. Na kraju se sve vrednosti iz liste štampaju zahvaljujući liniji 10.

Do sada je bilo pokazano kako da se generišu decimalni slučajni brojevi, međutim često je neophodno generisati i slučajne brojeve koji su celi. U tom slučaju pogodno je koristiti funkciju `randint(x,y)`, koja sadrži dva argumenta – početnu i krajnju vrednost koje definišu interval u kojem želimo da dobijemo slučajan broj. Na primer, ukoliko se želi dobiti slučajan broj u intervalu od 0 do 20, to se može postići kôdom koji sledi.

```
1.  from random import seed
2.  from random import randint
3.
4.  print(randint(0,10))
```

Sledeći korak je generisanje slučajnih brojeva čije vrednosti nisu samo između 0 i 1, nego se nalaze u nekom proizvoljnom intervalu. U tom slučaju, može se koristiti funkcija `random.uniform(x,y)`, gde x i y definišu početak i kraj brojnog intervala. U slučaju korišćenja ove funkcije, pogodno je učitati celu **random** biblioteku. Tako na primer, za definisanje slučajnog decimalnog broja iz intervala od -5 do 25, može se koristiti sledeći kôd

```
1.  import random
2.  x = random.uniform(-5,25)
3.  print(x)
```

Na ovaj način, pokrivene su najvažnije mogućnosti generisanja slučajnih brojeva. Naravno, slučajni brojevi se mogu generisati i upotrebom nekih drugih biblioteka dostupnih za python.

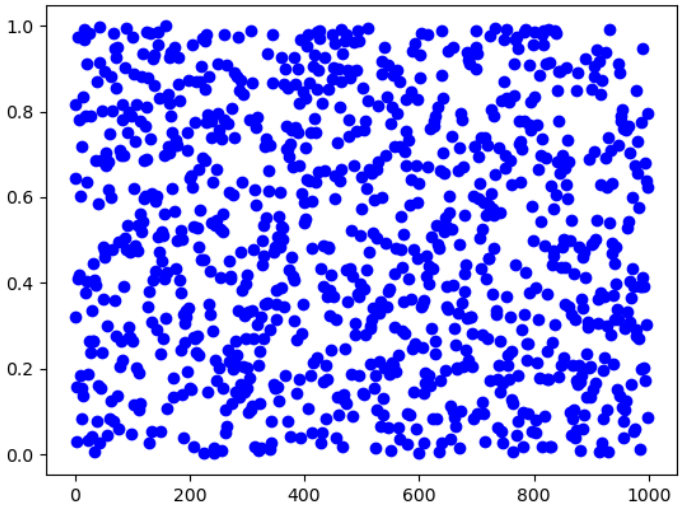
11.2. Raspodele verovatnoća

Svi dosadašnji programi za generisanje slučajnih brojeva su generisali slučajne brojeve tako da je verovatnoća pojavljivanja svakog broja jednaka. U to se može uveriti i vizelno, crtanjem svih slučajnih brojeva generisanih kôdom iz primera koji sledi.

Primer 11.3. Crtanje generisanih slučajnih brojeva

```
1.  from random import random
2.  import matplotlib.pyplot as plt
3.
4.  lista = []
5.
6.  for brojac in range(1000):
7.      vrednost = random()
8.      lista.append(vrednost)
9.
10. plt.plot(lista,'bo')
11. plt.show()
```

Dakle, datim kôdom je generisano 1000 slučajnih decimalnih brojeva čije su vrednosti između 0 i 1. Dati program kao izlaz daje grafik predstavljen na slici 11.2.



Slika 11.2. Slučajni brojevi od 0 do 1 – jednake/uniformne verovatnoće pojavljivanja

Analiza slike 11.2. jasno ukazuje da su tačke ravnomerno raspoređene. Međutim, verovatnoća prema kojoj se pojavljuju nasumični brojevi ne mora da bude uniformna.

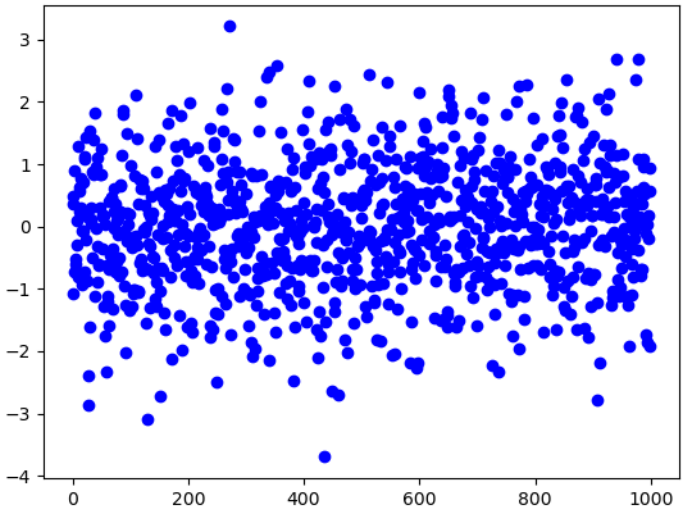
Verovatnoća pojavljivanja može da odgovara nekoj od poznatih statistčkih raspodela, i u okviru biblioteke **random** postoji nekoliko funkcija preko kojih se mogu generisati slučajni brojevi koji se podvrgavaju određenoj raspodeli.

Jedna od najpoznatijih raspodela je i Gausova normala raspodela, a slučajni brojevi u skladu sa ovom raspodelom se mogu generisati upotrebom funkcije `gauss(x,y)`, takođe iz biblioteke **random**. Vidi se da je `gauss(x,y)` funkcija kod koje figurišu dva parametra – x označava srednju vrednost (često se u literaturi označava sa μ) oko koje se „gomilaju“ slučajni brojevi, dok je y standardna devijacija (često se u literaturi označava sa σ) koja predstavlja meru varijabilnosti podataka kod normalne raspodele. Program za generisanje 1000 slučajnih brojeva prema Gausovoj raspodeli je dat u primeru koji sledi.

Primer 11.4. Generisanje slučajnih brojeva prema Gausovoj raspodeli

```
1.  from random import random
2.  from random import gauss
3.  import matplotlib.pyplot as plt
4.
5.  lista = []
6.
7.  for brojac in range(1000):
8.      vrednost = gauss(0,1)
9.      lista.append(vrednost)
10.
11. plt.plot(lista,'bo')
12. plt.show()
```

Program iz poslednjeg primera je praktično isti kao i prethodni program, tako da neće biti detaljnije razmatran. Napomenuće se samo da se u liniji 8 generišu slučajni brojevi prema Gausovoj raspodeli i to za srednju vrednost 0, dok je za standardnu devijaciju uzeto da je 1. Poslednji program će generisati grafik dat na slici 11.3.



Slika 11.3. Gausova raspodela ($\mu = 0, \sigma = 1$) dobijena programom iz primera 11.4.

Sa slike 11.3. se vidi jasno da su vrednosti najgušće oko vrednosti 0 na y-osi. U oblasti za vrednosti za 0 ± 1 (tj. za $\mu \pm \sigma$) vidi se da upada najveći broj generisanih slučajnih brojeva (iz teorije u vezi sa Gausovom raspodelom zna se da je to oko 68%).

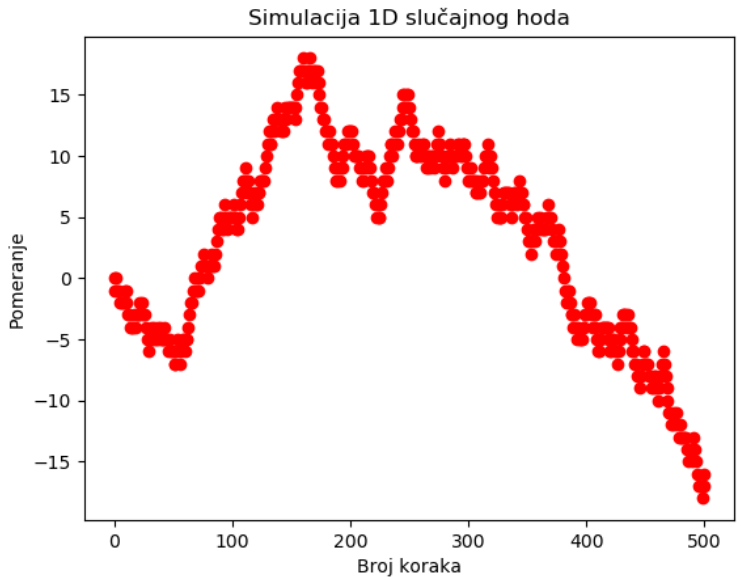
11.3. Simulacija slučajnog hoda

U prethodnim odeljcima bilo je predstavljeno kako se mogu na različite načine generisati slučajni brojevi. Sa tim tehnikama se dalje mogu simulirati neki slučajni procesi od značaja u nauci generalno. U tom smislu u primeru koji sledi će biti predstavljen program kojim se može simulirati jednodimenzioni slučajni hod.

Primer 11.5. Simulacija jednodimenzionog slučajnog hoda

```
1.  from random import randint
2.  import numpy as np
3.  import matplotlib.pyplot as plt
4.
5.  pomeranje = [0]
6.
7.  n = 500
8.  for brojac in range(n):
9.      slucajan_broj = randint(-1,1)
10.     pomeranje.append(slucajan_broj)
11.
12. pozicija_na_grafiku = np.cumsum(pomeranje)
13.
14. plt.plot(pozicija_na_grafiku,'ro')
15. plt.title('Simulacija 1D slučajnog hoda')
16. plt.xlabel('Broj koraka')
17. plt.ylabel('Pomeranje')
18. plt.show()
```

U 5. liniji kôda je definisana lista koja ima samo jedan član, i to je broj nula, da bi kretanje krenulo od nule. U liniji 7 je definisan broj slučajnih koraka, što znači da će se „šetač“ kretati na gore, dole, ili ostati u mestu. Ova lista će se popunjavati slučajnim vrednostima preko algoritma koji je zadat u linijama od 8 do 10. Te slučajne vrednosti, kao što je već napomenuto, mogu biti 1, 0 ili -1. Ono što je novost jeste funkcija iz **numpy** paketa koja se zove `cumsum()`. Ovom funkcijom se vrši kumulativno sabiranje elemenata iz liste `pomeranje`. Pod kumulativnom sumom se podrazumeva suma svih prethodnih elemenata u nizu ili sekvenci, koja se računa za svaki element pojedinačno. Drugim rečima, kumulativna suma je zbir svih prethodnih vrednosti u nizu do trenutnog elementa. Na primer, ako lista sadrži elemente [1,2,3,4,5], kumulativna suma ovih vrednosti će biti [1,3,6,10,15][8]. Kumulativnom sumom se dobijaju nove vrednosti duž y-ose koje će se crtati na grafiku. U preostalim linijama se izdaju instrukcije za crtanje grafika, a program daje grafik predstavljen na slici 11.4.



Slika 11.4. Slučajan hod dobijen programom iz primera 11.5.

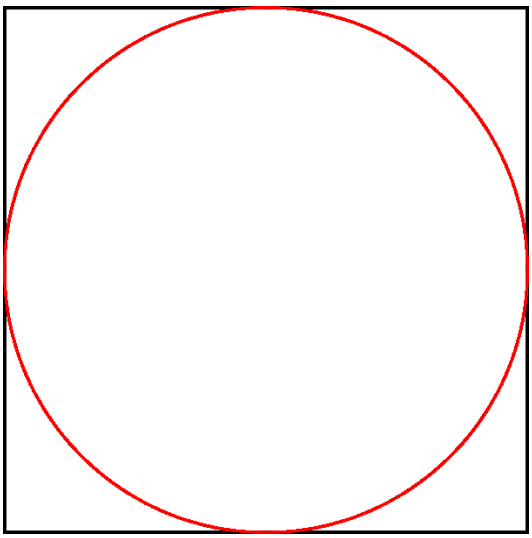
Dati kôd daje mogućnost da se nasumično „prošeta“ kroz bilo koji sistem i po potrebi uzorkuju ili snime željeni parametri. Ovo je posebno korisno kod istraživanja periodičnih struktura, jer dati program može poslužiti kao osnova za „šetanje“ kroz čvorove kristalne rešetke.

[8] Detaljnije, 1, 1+2, 3+3,6+4,10+5 će dati 1,3,6,10,15

11.4. Određivanje vrednosti π Monte Karlo simulacijom

Monte Karlo simulacije su simulacije koje se zasnivaju na slučajnom uzorkovanju. Monte Karlo metoda za aproksimaciju broja π (pi) je jedna od najpoznatijih tehnika za numeričko računanje vrednosti broja π. Ova metoda se zasniva na slučajnom uzorkovanju tačaka unutar kvadrata i računanju odnosa broja tačaka koje padaju unutar jediničnog kruga i ukupnog broja uzorkovanih tačaka. Koncept Monte Karlo metode datira iz polovine 20. veka, a naziv metode je inspirisan poznatim kockarskim gradom u Monaku, jer se u ovoj metodi koristi element slučajnosti.

U ovom odeljku će biti prikazano kako se mogu napisati python programi za dobijanje vrednosti broja π. Pre samog programa, biće uveden model koji će se koristiti. Neka je dat kvadrat čija je dužina stranice 2 i neka je u njega upisan krug poluprečnika 1, dakle $a = 2$ i $r = 1$, kao što je to ilustrovano na slici 11.5.



Slika 11.5. Model za računanje vrednosti π

Neka je površina kvadrata $P_{kv} = a^2$ i neka je površina kruga $P_{kr} = r^2 \cdot \pi$. Odnos površina kvadrata i kruga je

$$\frac{P_{kv}}{P_{kr}} = \frac{a^2}{r^2 \cdot \pi} . \tag{11.1}$$

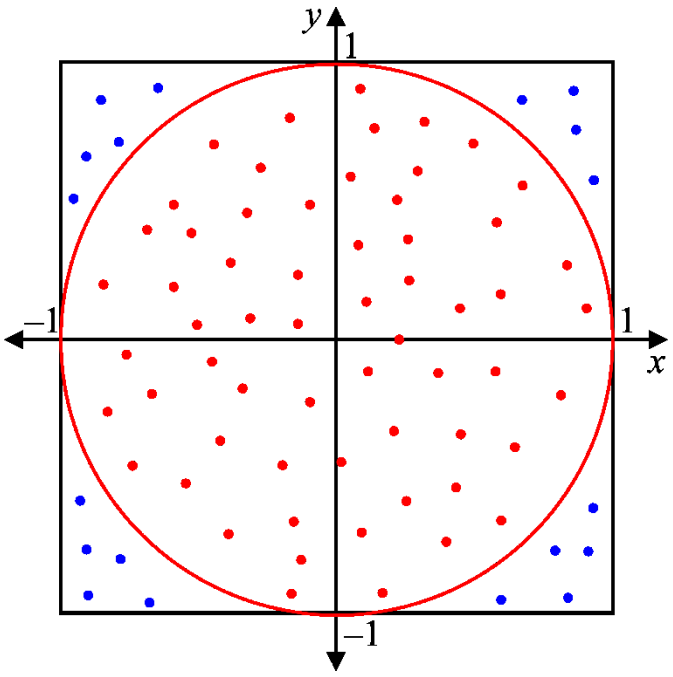
Imajući u vidu da pomenute vrednosti stranice i poluprečnika, prethodna relacija dobija sledeći oblik

$$\frac{P_{kv}}{P_{kr}} = \frac{a^2}{r^2 \cdot \pi} = \frac{2^2}{1^2 \cdot \pi} = \frac{4}{\pi} . \tag{11.2}$$

Iz poslednje relacije, π se može izraziti na sledeći način

$$\pi = 4 \cdot \frac{P_{kr}}{P_{kv}} . \tag{11.3}$$

Relacija (11.3) je ključna za primenu Monte Karlo simulacije za određivanje vrednosti π i vrlo brzo će biti iskorišćena u programu. Neka se prostor sa slike 11.5. popunjava tačkama (uniformne verovatnoće pojavljivanja), kao da se nasumično bombarduje prostor, kao što je to ilustrovano na slici 11.6.



Slika 11.6. Nasumično ispunjavanje tačkama prostora sa slike 11.5

Na slici 11.6 je dodat koordinatni sistem gde su crvenom bojom obeležene tačke koje upadaju u kružnicu, dok su plavom bojom obeležene tačke koje ne upadaju u kružnicu. Odnos površine kvadrata i površine kruga tada se može izraziti kao odnos ukupnog broja tačaka u celom prostoru i broja tačaka koje upadaju u površinu kruga

$$\frac{P_{kv}}{P_{kr}} = \frac{ukupan\ broj\ tačaka}{broj\ tačaka\ u\ krugu} . \tag{11.4}$$

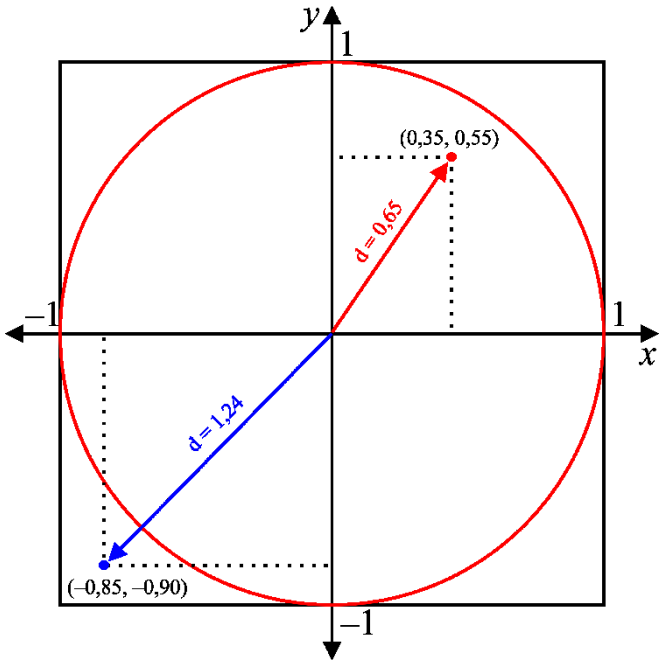
Uzimajući u obzir vrednosti stranice i poluprečnika, kombinacijom izraza (11.3) i 11.(4) dobija se sledeći izraz preko kojeg se može odrediti vrednost π

$$\pi = 4 \cdot \frac{P_{kr}}{P_{kv}} = 4 \cdot \frac{broj\ pogodaka\ u\ krug}{ukupan\ broj\ gađanja} \,. \tag{11.5}$$

Poslednji izraz omogućava određivanje vrednosti π, a sve što treba da se postigne kôdom u pythonu jeste da se prebroji ukupan broj pogodaka u krug poluprečnika 1. U te svrhe, potrebno je rastojanje neke tačke od centra posmatranog koordinatnog sistema izračunati primenom sledećeg izraza

$$d = \sqrt{x^2 + y^2}\,, \tag{11.6}$$

gde su x i y koordinate posmatrane tačke. Naravno, svaka tačka sa slike 11.6. je određena koordinatama x i y. Na slici 11.7. su ilustrovane dve tačke od kojih jedna upada u kružnicu, a druga ne upada.



Slika 11.7. Primer tačaka od kojih jedna upada, a druga ne upada u okvir kružnice

Uzimajući u obzir geometrije prikazane na slikama 11.6 i 11.7, ukoliko je rastojanje d veće od 1, tačka ne upada u kružnicu, a ukoliko je manje od 1, upada u okvir kružnice. Upravo će navedeno biti iskorišćeno u kôdu za određivanje vrednosti π.

Da bi se nasumično generisala neka tačka sa slike 11.7., u pythonu će se iskoristiti funkcija `random`() za generisanje komponenti x i y, a zatim u skladu sa izrazom (11.6) proveravati da li tako generisana tačka upada u kružnicu ili ne. Ukoliko upada, brojaču će se dodavati odgovarajuća vrednost i na taj način prebrojati ukupan broj tačaka koje upadaju u kružnicu. U

primeru koji sledi dat je python program koji implementira prethodno navedeno u svrhe računanja vrednosti broja π Monte Karlo simulacijom.

Primer 11.6. Program za određivanje vrednosti π Monte Karlo metodom

```
1.  import numpy as np
2.  import random
3.
4.  broj_tacaka = 100000
5.  broj_pogodaka = 0
6.
7.  for i in range(broj_tacaka):
8.      x = random.uniform(-1,1)
9.      y = random.uniform(-1,1)
10.     duzina = np.sqrt(x**2+y**2)
11.     if duzina <= 1:
12.         broj_pogodaka = broj_pogodaka + 1
13.
14. pi = 4*(broj_pogodaka/broj_tacaka)
15. print("Pi je priblizno jednako: ",pi)
```

Broj tačaka je definisan u liniji 4. Broj pogodaka u kružnicu, linija 5, je podešen na 0, i taj parametar suštinski služi kao brojač. Od linije 7 – 12 slede instrukcije za generisanje nasumičnih tačaka, definisanja dužine i provere (uslov) da li generisana tačka upada u kružnicu poluprečnika 1.

S obzirom na geometriju slika 11.4. – 11.6., jasno je da će komponente x i y uzimati slučajnu vrednost u intervalu od -1 do 1 (linije 8 i 9). Dužina (linija 9) je definisana u skladu sa izrazom (11.6). Ukoliko je uslov ispunjen (linija 11), parametar `broj_pogodaka` se uvećava za jedan. Njegova vrednost se pamti u memoriji i figuriše u izrazu za računanje vrednosti π prema (11.5), u liniji 14. Konačno, vrednost broja π se štampa u poslednjoj liniji.

Napomena je da tačnost vrednosti π u ovakvom kôdu zavisi od parametra `broj_tacaka`. Naravno, što je ovaj broj veći, vrednost π će biti tačnija, jer će biti izračunata na osnovu većeg broja tačaka. Ukoliko `broj_tacaka` uzima vrednost 1000, dobija se vrednost broja π sa relativno zadovoljavajućom tačnošću, uzimajući u obzir činjenicu da izvršenje kôda u tom slučaju traje manje od jedne sekunde. Sa druge strane, da bi se dobila upotrebljiva vrednost π neophodno je povećati broj tačaka za tri reda veličine. Tada simulacija traje nekoliko sekundi, ali se dobija vrednost π sa tačnošću na dve decimale.

Python ima opciju i da, upotrebom biblioteke ***time***, izmeri koliko je vremena potrebno za izvršenje kôda. Ovo je izuzetno pogodno za potrebe praćenja rada i optimizacije kôda. Pored učitavanja pomenute biblioteke, neophodno je kreirati dve linije koje obuhvataju linije kôda

čije izvršenje želi da se izmeri. Prethodni kôd za dobijanje vrednosti π, sa merenjem vremena neophodnog za izvršenje, izgleda kao u primeru koji sledi.

Primer 11.7. Merenje vremena za izvršenje koda za Monte Karlo proračun vrednosti π

```
1.  import numpy as np
2.  import random
3.  import time
4.
5.  start = time.time()
6.
7.  broj_tacaka = 100000
8.  broj_pogodaka = 0
9.  for i in range(broj_tacaka):
10.     x = random.uniform(-1,1)
11.     y = random.uniform(-1,1)
12.     duzina = np.sqrt(x**2+y**2)
13.     if duzina <= 1:
14.         broj_pogodaka = broj_pogodaka + 1
15.
16. pi = 4*(broj_pogodaka/broj_tacaka)
17.
18. end = time.time()
19.
20. print("Pi je priblizno jednako: ",pi)
21. print('Vreme izvrsenja koda:',end-start,'s')
```

Program je identičan prethodnom programu, sa dodatkom linja 5 i 18, gde su definisane veličine „start“ i „end“ koje beleže vremena u trenucima kada python očitava pomenute linije i pamti te vrednosti. Konačno, u liniji 21 se daje instrukcija python-u da nađe razliku između ta dva vremena, što predstavlja vreme izvršenja kôda.

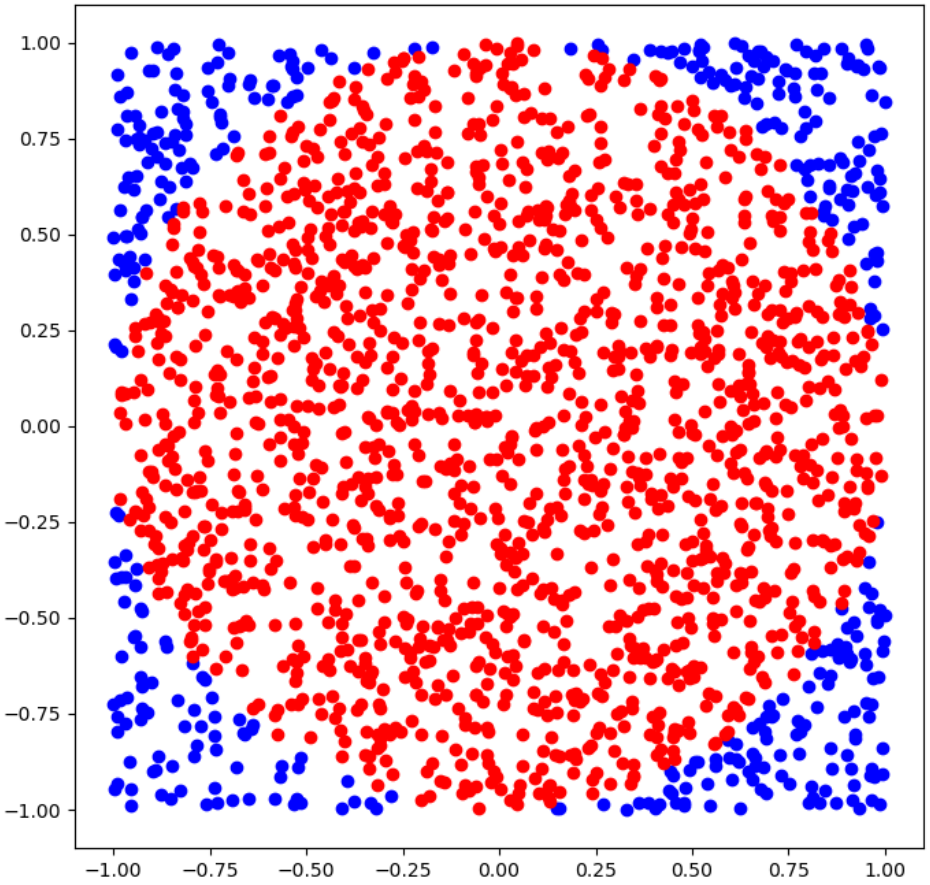
Za kraj, predstavljen je i analiziran kôd kojim se može iscrtati grafik na kome su predstavljene tačke koje upadaju u krug i tačke koje su izvan kruga. Naravno, kôd je zasnovan na prethodnom kodu i dat je u sledećem primeru.

Primer 11.8. Vizuelizacija tačaka kod Monte Karlo simulacije vrednosti broja π

```
1.  import numpy as np
2.  import random
3.  import matplotlib.pyplot as plt
4.
5.  broj_gadjanja = 3000
6.  broj_pogodaka = 0
7.
8.  upada_x = []
9.  upada_y = []
10. ne_upada_x = []
11. ne_upada_y = []
12.
13. for i in range(broj_gadjanja):
14.     x = random.uniform(-1,1)
```

```
15.     y = random.uniform(-1,1)
16.     intenzitet = np.sqrt(x**2+y**2)
17.     if intenzitet <= 1:
18.         broj_pogodaka = broj_pogodaka + 1
19.         upada_x.append(x)
20.         upada_y.append(y)
21.     else:
22.         ne_upada_x.append(x)
23.         ne_upada_y.append(y)
24.
25. plt.plot(ne_upada_x,ne_upada_y,"bo")
26. plt.plot(upada_x,upada_y,"ro")
27. plt.show()
```

Razlika u odnosu na prethodni kôd je u tome što su u linijama 10-13 dodate prazne liste za beleženje komponenata tačaka koje upadaju, tj. koje ne upadaju, u krug poluprečnika 1. Funkcija `append()` je ponovo iskorišćena u odgovarajućim linijama (19, 20 i 22, 23) za popunjavanje prethodno popunjenih listi, u zavisnosti od toga da li ispunjavaju uslov definisan u liniji 17. Linije 25-26 daju instrukcije za crtanje slike 11.8.



Slika 11.8. Monte Karlo simulacija vrednosti π – vizuelizacija generisanih tačaka

Na slici 11.8. se jasno vidi kako izvestan broj tačaka upada u kružnicu, a izvestan ne upada. Pomenuta slika je dobijena ukoliko ukupan broj tačaka iznosi 1000. Naravno, ukoliko je premali broj ukupnih tačaka (npr. 100), ne samo da će vrednost π biti nezadovoljavajuća, nego ni slika neće jasno ukazivati na formiranje kruga od strane crvenih tačaka. U slučaju velikog ukupnog broja tačaka (oko 50000), raspodela tačaka može biti toliko gusta, da se vidi potpuno obojen crveni krug u plavom kvadratu.

Literatura

[1] A. Martelli, A.M. Ravenscroft, S. Holden, P. McGuire, Python in a Nutshell, O'Reilly Media, Inc., 2023.

[2] S. Armaković, S.J. Armaković, Atomistica. online–web application for generating input files for ORCA molecular modelling package made with the Anvil platform, Molecular Simulation. 49 (2023) 117–123.

[3] L. Gojković, S. Malijević, S. Armaković, Modeling of fundamental electronic circuits by the Euler method using the Python programming language, Physics Education. 55 (2020) 055016.

[4] A. Gezerlis, Numerical methods in physics with Python, Cambridge University Press, 2020.

[5] D.J. Pine, Introduction to Python for science and engineering, CRC Press, 2019.

[6] J. Hao, T.K. Ho, Machine learning made easy: a review of scikit-learn package in python programming language, Journal of Educational and Behavioral Statistics. 44 (2019) 348–361.

[7] D. Phillips, Python 3 object-oriented programming: Build robust and maintainable software with object-oriented design patterns in Python 3.8, Packt Publishing Ltd, 2018.

[8] R.H. Landau, M.J. Páez, Computational Problems for Physics: With Guided Solutions Using Python, CRC Press, 2018.

[9] R.L. Halterman, Fundamentals of Python programming, (2018).

[10] K. Srinath, Python–the fastest growing programming language, International Research Journal of Engineering and Technology. 4 (2017) 354–357.

[11] F. Kane, Hands-on data science and python machine learning, Packt Publishing Ltd, 2017.

[12] A.C. Müller, S. Guido, Introduction to machine learning with Python: a guide for data scientists, O'Reilly Media, Inc., 2016.

[13] S. Monk, Programming the Raspberry Pi: getting started with Python, McGraw-Hill Education, 2016.

[14] H.P. Langtangen, A primer on scientific programming with Python, Springer, 2016.

[15] J.V. Guttag, Introduction to computation and programming using Python: With application to understanding data, MIT press, 2016.

[16] J.M. Perkel, Programming: Pick up python, Nature. 518 (2015) 125–126.

[17] R.H. Landau, M.J. Páez, C.C. Bordeianu, Computational physics: Problem solving with Python, John Wiley & Sons, 2015.

[18] A. Bogdanchikov, M. Zhaparov, R. Suliyev, Python to learn programming, Journal of Physics: Conference Series. 423 (2013) 012027. https://doi.org/10.1088/1742-6596/423/1/012027.

[19] M. Lutz, Learning python: Powerful object-oriented programming, O'Reilly Media, Inc., 2013.

[20] J. Kiusalaas, Numerical methods in engineering with Python 3, Cambridge university press, 2013.

Dodatak 1. Fajlovi sa podacima i python programi

Svi fajlovi sa podacima i python programi predstavljeni u ovom udžbeniku su dostupni na web adresi https://armakovic.com/textbooks/programiranje-u-fizici-puf/, u skladu sa dostupnim resursima, u obliku:

- .py fajlova,
- Jupyter Notebook i
- interaktivnih programa u okviru online IDE servisa, zahvaljujući čemu korisnici mogu da puštaju sve programe iz ovog udžbenika direktno iz internet pretraživača.

Na istom mestu, moguće je vežbati python programiranje u okviru online servisa.

Dodatak 2. Korišćene biblioteke

U ovom odeljku se navodi spisak korišćenih eksternih biblioteka, komande za njihovu instalaciju preko **conda** i **pip** servisa, kao i adrese njihovih zvaničnih internet prezentacija.

Biblioteka	Instalacija preko conda	Instalacija preko pip	Web adresa
numpy	conda install numpy	pip install numpy	https://numpy.org/
matplotlib	conda install matplotlib ili conda install -c conda-forge matplotlib	python -m pip install -U pip python -m pip install -U matplotlib	https://matplotlib.org/
PyQt5	conda install -c anaconda pyqt	pip install PyQt5	https://www.riverbankcomputing.com/software/pyqt/
scipy	conda install -c anaconda scipy	python -m pip install scipy	https://scipy.org/
scikit-learn	conda install -c anaconda scikit-learn	pip install -U scikit-learn	https://scikit-learn.org
pandas	conda install -c anaconda pandas	pip install pandas	https://pandas.pydata.org/

Dodatak 3. Korišćeni fajlovi

U ovom odeljku su dati sadržaji .csv fajlova sa podacima koji su korišćeni u python programima ovog udžbenika. U slučaju .csv fajlova, podaci su predstavljeni u obliku tabele, kao i u .csv formatu. Svi fajlovi sa podacima korišćenim u python programima ovog udžbenika su dostupni na web sajtu https://armakovic.com/textbooks/programiranje-u-fizici.

Spisak fajlova redosledom kojim su korišćeni u udžbeniku:

- tekst_fajl.txt
- g_const_1.csv
- upisivanje.csv
- probni_fajl.csv
- g_const_2.csv
- RL_kolo.csv
- Pa_raspad.csv
- interpolacija.csv
- gej_lisak.csv

Naziv fajla: tekst_fajl.txt

Sadržaj tekstualnog fajla
Prva linija teksta Druga linija teksta Treca linija teksta Putanja do ovog fajle je d:\radni direktorijum\tekst_fajl.txt

Naziv fajla: g_const_1.csv

Tabelarni format	.csv format
period g_const 1.98 10.06 2.01 9.76 2.05 9.38 2.01 9.76 2.01 9.76 2.06 9.29 1.97 10.16 2.03 9.57 2.01 9.76 1.99 9.96	period,g_const 1.98,10.06 2.01,9.76 2.05,9.38 2.01,9.76 2.01,9.76 2.06,9.29 1.97,10.16 2.03,9.57 2.01,9.76 1.99,9.96

Naziv fajla: upisivanje.csv

Tabelarni format	.csv format
0 1 2 3 4 5	0 1 2 3 4 5

Naziv fajla: probni_fajl.csv

Tabelarni format	.csv format
Kolona0 Kolona1 1 10 2 20 3 30 4 40 5 50	Kolona0,Kolona1 1,10 2,20 3,30 4,40 5,50

Naziv fajla: g_const_2.csv

Tabelarni format			.csv format
duzina	period	period^2	duzina,period,period^2
0.5	1.41	1.99	0.50,1.41,1.99
0.55	1.48	2.19	0.55,1.48,2.19
0.6	1.54	2.37	0.60,1.54,2.37
0.65	1.61	2.59	0.65,1.61,2.59
0.7	1.68	2.83	0.70,1.68,2.83
0.75	1.72	2.96	0.75,1.72,2.96
0.8	1.8	3.24	0.80,1.80,3.24
0.85	1.85	3.42	0.85,1.85,3.42
0.9	1.9	3.61	0.90,1.90,3.61
0.95	1.92	3.69	0.95,1.92,3.69
1	2.03	4.12	1.00,2.03,4.12

Naziv fajla: RL_kolo.csv

Tabelarni format		.csv format
t	i	t,i
0	-0.34291	0,-0.3429121
1.034483	0.428503	1.034482759,0.428502967
2.068966	1.508195	2.068965517,1.508195174
3.103448	2.126549	3.103448276,2.126548973
4.137931	2.153206	4.137931034,2.153205794
5.172414	3.051246	5.172413793,3.051245854
6.206897	3.361631	6.206896552,3.361630942
7.241379	3.572623	7.24137931,3.572622651
8.275862	4.284998	8.275862069,4.284997938
9.310345	3.794469	9.310344828,3.794469181
10.34483	4.551359	10.34482759,4.551359349
11.37931	4.41138	11.37931034,4.411379843
12.41379	4.845044	12.4137931,4.845044237
13.44828	5.085261	13.44827586,5.085261146
14.48276	4.946227	14.48275862,4.946227388
15.51724	4.687203	15.51724138,4.687202729
16.55172	4.686864	16.55172414,4.686864344
17.58621	4.495951	17.5862069,4.495950858
18.62069	4.558648	18.62068966,4.558647922
19.65517	5.016643	19.65517241,5.016642642
20.68966	4.959599	20.68965517,4.95959865
21.72414	4.763192	21.72413793,4.763192028
22.75862	4.970859	22.75862069,4.970858959
23.7931	4.863309	23.79310345,4.863309478
24.82759	4.742114	24.82758621,4.742113534
25.86207	5.012004	25.86206897,5.012003529
26.89655	5.131899	26.89655172,5.1318994
27.93103	5.229475	27.93103448,5.229475045
28.96552	4.794964	28.96551724,4.794963722
30	4.995408	30,4.995407709

Naziv fajla: Pa_raspad.csv

Tabelarni format		.csv format
t	i	t,brojac
0	-0.34291	0,80.3
1.034483	0.428503	10,75
2.068966	1.508195	20,67.3
3.103448	2.126549	30,60.5
4.137931	2.153206	40,55.2
5.172414	3.051246	50,49.6
6.206897	3.361631	60,45.7
7.241379	3.572623	70,41.5
8.275862	4.284998	80,37.4
9.310345	3.794469	90,34.1
10.34483	4.551359	100,31.3
11.37931	4.41138	110,28.5
12.41379	4.845044	120,25.9
13.44828	5.085261	130,23.9
14.48276	4.946227	140,21.7
15.51724	4.687203	150,19.4
16.55172	4.686864	160,17.6
17.58621	4.495951	170,16.4
18.62069	4.558648	180,15.1
19.65517	5.016643	190,13.4
20.68966	4.959599	200,12.3
21.72414	4.763192	210,11.2
22.75862	4.970859	220,10.5
23.7931	4.863309	230,9.2
24.82759	4.742114	240,8.9
25.86207	5.012004	250,7.5
26.89655	5.131899	260,6.9
27.93103	5.229475	270,6.4
28.96552	4.794964	280,5.7
30	4.995408	290,5.3
		300,4.7

Naziv fajla: interpolacija.csv

Tabelarni format		.csv format
vreme	temperatura	vreme,temperatura
0	22.1	0,22.1
1	20.3	1,20.3
2	19.2	2,19.2
3	18.3	3,18.3
4	18.2	4,18.2
5	18	5,18
6	18.6	6,18.6
7	19.1	7,19.1
8	21.1	8,21.1
9	23.1	9,23.1
10	24.2	10,24.2
11	24.3	11,24.3
12	25.1	12,25.1
13	26.2	13,26.2
14	27.1	14,27.1
15	27.8	15,27.8
16	27.9	16,27.9
17	26	17,26
18	24.5	18,24.5
19	23	19,23
20	22.1	20,22.1
21	22	21,22
22	21.7	22,21.7
23	21.2	23,21.2
24	22.1	24,22.1

Naziv fajla: gej_lisak.csv

Tabelarni format		.csv format
temperatura	pritisak	temperatura,pritisak
100	980	100,980
77.1	915	77.1,915
51	845	51,845
24	790	24,790
0	730	0,730
-197	198	-197,198

O AUTORU

Stevan Armaković je rođen u Sremskoj Mitrovici 1985. godine, gde je završio osnovnu školu „Slobodan Bajić Paja“, nižu muzičku školu „Petar Krančević“, instrument klarinet, i prirodno-matematički smer gimnazije „Ivo Lola Ribar“. Na republičkim takmičenjima klarinetista dva puta je osvojao treću nagradu (jednom kao solista, jednom u okviru kamernog dueta). Aktivno je igrao košarku za košarkaški klub „Srem“ do odlaska na studije.

Studije fizike je upisao 2004. godine na Departmanu za fiziku, Prirodno-matematičkog fakulteta Univerziteta u Novom Sadu, gde je diplomirao 2009. godine. 2011. godine je upisao doktorske studije na istom mestu, a doktorsku disertaciju je odbranio 2014. godine. Na Departmanu za fiziku je zaposlen od 2010. godine, najpre u svojstvu istraživača pripravnika, zatim istraživača saradnika (2012-215), naučnog saradnika (2015-2020). Od 2019. godine je u zvanju docenta i angažovan je u nastavi Departmana na više predmeta, između ostalog i na predmetu „Numeričke metode i programiranje u fizici“.

Bavi se primenom kompjuterskog modelovanja materijala i do sada je publikovao oko 140 radova u časopisima sa impakt faktorom. Prema bazi SCOPUS, citiran je 2957 puta, dok mu h-indeks iznosi 35. Prema bazi Web of Science, do sada je uradio 380 recenzija radova u časopisima sa impakt faktorom. Dobitnik je priznanja za naučnu izuzetnost istaživača na teritoriji AP Vojvodine, kao treći najcitiraniji istraživač za 2021. godinu u podoblasti za fiziku, hemiju i fizičku geografiju. Najcitiraniji je fizičar na teritoriji AP Vojvodine.

Inicijator je i koosnivač Udruženja za međunarodni razvoj akademske i naučne saradnje, https://aidasco.org. Glavni je urednik časopisa AIDASCO Reviews, ISSN: 2956-0888 (Online), https://publishing.aidasco.org/journals/index.php/aire. Koosnivač je i glavni programer u okviru projekta atomistica.online, besplatne online platforme za molekulsko modelovanje dostupne na adresi https://atomistica.online.

Oženjen je Sanjom, sa kojom ima sinove Filipa i Stefana.

IZ RECENZIJE

„Udžbenik „Programiranje u fizici“ autora dr Stevana Armakovića je namenjen studentima fizike za savladavanje dela predmeta „Numeričke metode i programiranje u fizici“ koji se odnosi na programiranje.“

„Udžbenik je napisan jasno i pregledno, a osnove programiranja su demonstrirane upotrebom programskog jezika Pajton. Upotreba programskog jezika pajton ima više prednosti, jer sa radi o potpuno besplatnom kompjuterskom alatu, jasne i intuitivne sintakse, sa mnoštvom gotovih biblioteka koje proširuju njegovu upotrebljivost“

„Sa pedagoškog aspekta, treba istaći da je autor omogućio pokretanje i modifikovanja svih programa predstavljenih u ovom udžbeniku putem internet servisa dostupnog preko linka koji je naveden u predgovoru, kao i da se programira u pajtonu u okviru veb stranice, bez potrebe za bilo kakvom instalacijom i održavanjem.“